Центральноукраїнський національний технічний університет
Механіко-технологічний факультет
Кафедра кібербезпеки та програмного забезпечення

”Допущено до захисту”
Завідувач кафедри кібербезпеки
та програмного забезпечення
д.т.н., професор
_____ Олексій СМІРНОВ
“___” _____ 20__ р.

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
за другим (магістерським) рівнем вищої освіти
на тему
“Дослідження та програмна реалізація комп’ютерної моделі репутаційної системи соціальної мережі”

Виконав здобувач вищої освіти
ІІ курсу, групи КН-21м
ОПП «Комп’ютерні науки»
спеціальності 122 «Комп’ютерні науки»
_____ Мосольд М.І.
«____» _____ 20 р.

Керівник проекту
Доктор технічних наук, професор
_____ Єлизавета МЕЛЕШКО
«____» _____ 20 р.
Рецензент _____
_____

м. Кропивницький

Центральноукраїнський національний технічний університет

Факультет *Механіко-технологічний*

Кафедра *Кібербезпеки та програмного забезпечення*

Освітній ступінь *бакалавр*

Галузь знань *12 "Інформаційні технології"*

Спеціальність *122 "Комп'ютерні науки"*

Освітньо-професійна (освітньо-наукова) програма *"Комп'ютерні науки"*

ЗАТВЕРДЖУЮ
Завідувач кафедри
д.т.н., проф.
_____Олексій СМІРНОВ
« 10 » грудня 2022 року

ЗАВДАННЯ НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ РОБОТУ ЗА ДРУГИМ (МАГІСТЕРСЬКИМ) РІВНЕМ ВИЩОЇ ОСВІТИ ЗДОБУВАЧА ВИЩОЇ ОСВІТИ

Мосольду Максиму Івановичу
(прізвище, ім'я, по батькові)

1. Тема роботи *Дослідження та програмна реалізація комп'ютерної моделі репутаційної системи соціальної мережі*

2. Керівник роботи *Мелешко Єлизавета Владиславівна, доктор техн. наук, професор*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу № від року

3. Строк подання роботи до захисту *20.12.2022 р.*

4. Мета та завдання випускної кваліфікаційної роботи: *Метою роботи є дослідження та програмна реалізація комп'ютерної моделі репутаційної системи соціальної мережі*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Призначення та область використання.
2.Перегляд аналогічних існуючих систем.
3. Опис і обґрунтування проектних рішень.
4. Етапи програмування системи.
5. Впровадження системи в промислову експлуатацію.
6. Наукова новизна
7. Економічна ефективність розробленої програми.
8. Заходи з охорони праці та техніки безпеки.
9. Висновки.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

Наукова новизна	*1 аркуш*
Структурна схема системи	*1 аркуш*
Функціональна схема системи	*1 аркуш*
Блок-схема алгоритму роботи додатку	*7 аркуша*
Діаграма процесів	*1 аркуш*
Показники економічної ефективності	*1 аркуш*

6. Консультанти по роботі, із зазначенням розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	Савеленко Г.В., к.т.н., доцент	14.11.2022	5.12.2022
Охорона праці	Оришака О.В., к.т.н., доцент	16.11.2022	7.12.2022

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти	Строк виконання етапів випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти	Примітка
1.	Аналіз існуючих систем	10.9.2022 р.	
2.	Постановка задачі, оформлення ТЗ	15.9.2022 р.	
3.	Розробка моделі компонента	20.9.2022 р.	
4.	Розробка структур даних	5.10.2022 р.	
5.	Розробка алгоритмів зв’язку та відображення	14.10.2022 р.	
6.	Програмування алгоритмів	21.10.2022 р.	
7.	Розрахунок економічної ефективності	14.11.2022 р.	
8.	Розрахунки з охорони праці та техніки безпеки	16.11.2022 р.	
9.	Оформлення ПЗ	26.11.2022 р.	
10.	Попередній захист роботи	10.12.2022 р.	

Дата видачі завдання
«__»_____20 р.

Підпис керівника

_____ _____
(прізвище та ініціали)

Завдання прийнято до виконання
«__»_____20 р.

Підпис здобувача

_____ _____
(прізвище та ініціали)

АНОТАЦІЯ

Мосольд М.І. Дослідження та програмна реалізація комп'ютерної моделі репутаційної системи соціальної мережі. 122 Комп'ютерні науки. Центральноукраїнський національний технічний університет. Кропивницький. 2022.

Метою розробки є дослідження та комп'ютерна реалізація моделі репутаційної системи соціальної мережі.

Об'єктом дослідження є процес симуляції репутаційної системи соціальної мережі в комп'ютерних моделях.

Предметом дослідження є методи та алгоритми симуляції комп'ютерної моделі репутаційної системи соціальної мережі.

Методи дослідження базуються на теорії об'єктно-орієнтованого програмування, теорії алгоритмів, методах комп'ютерного моделювання, а також теорії розповсюдження інформації.

В даній магістерській роботі досліджено та розроблено програмне забезпечення, яке призначено для реалізації комп'ютерної моделі репутаційної системи соціальної мережі.

В процесі роботи над програмною реалізацією виконано дослідження та аналіз існуючих програмних та апаратних засобів. В повній мірі описані всі компоненти розробленого програмного забезпечення.

Розроблено зручний графічний інтерфейс користувача. Наведені інструкції по роботі з програмними засобами.

Розроблене програмне забезпечення розповсюджується по ліцензії "open-source software".

Програма може використовуватися на ПЕОМ архітектури IBM PC з ОС Windows 7/8/10/11 та Linux.

Програму розроблено в середовищі IntelliJ IDEA Community Edition 2022 на мові програмування Java.

Ключові слова: комп'ютерні науки, програмна реалізація, комп'ютерна модель, репутаційна система, соціальна мережа, база даних, дослідження, Java.

ABSTRACT

Mosold M.I. Research and software implementation of a computer model of a social network reputation system. 122 Computer Science. Central Ukrainian National Technical University. Kropyvnytskyi 2022

The purpose of the development is research and computer implementation of computer model of the social network reputation system.

The object of the study is the process of modeling the social network reputation system in computer models.

The subject of the research is methods and algorithms for modeling the computer model of the social network reputation system.

Research methods are based on the theory of object-oriented programming, the theory of algorithms, computer-modeling methods, and the theory of information dissemination.

In this magister thesis researched and developed software designed to implement a computer model of the reputation system of a social network.

The result of the work is the software implementation of a computer model of the reputation system of a social network.

In the process of working on a software implementation of a computer model of a reputational social network an analysis of existing hardware and software was performed. All components of the software developed are fully described.

User-friendly graphical interface is developed. Developed instructions for working with software.

The software is distributed under the "Open-Source Software" license.

The program can be used on an IBM PC running Windows 7/8/10/11 and Linux PC.

The program was developed in the IntelliJ IDEA Community Edition 2022 environment in the Java programming language.

Keywords: computer science, software implementation, computer model, reputation system, social network, database, research, Java.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

ОС	–	Операційна система, сукупність пов'язаних між собою програмних засобів, які дозволяють управляти ресурсами та апаратною частиною ПК.
Iterator	–	Інтерфейс, що використовується для перебору елементів деякої структури даних або контейнера.
InfoSec	–	Практика захисту та збереження даних від несанкціонованого доступу, спотворення, розкриття, використання, розповсюдження та зміни.
ПЗ	–	Програмне забезпечення.
IDE	–	Комплекс програмних засобів, що містять все необхідне для розробки, компіляції, пошуку помилок та налагодження коду.
Driver	–	Спеціальне програмне забезпечення, за допомогою якого операційна система взаємодіє з апаратним забезпеченням пристрою, який було під’єднано до неї.
HTTP	–	Протокол передачі гіпертексту.
СУБД	–	Система управління базами даних.
VPN	–	Віртуальна приватна мережа. Логічна мережа, створена на базі віртуальних каналів інших мереж.
Git	–	Система управління та контролю версіями розроблюваного продукту, утиліта для збереження, відстеження та ведення історії змін у проекті.
Java	–	Строго типізована об'єктно-орієнтована мова програмування загального призначення.

БД	–	Сукупність інформації, яка зберігається відповідно до деякої схеми даних, відповідно до правил засобів моделювання даних виконуються операції над ними.
SQLite	–	Компактна вбудована система управління базами даних.
API	–	Інтерфейс прикладного програмування.
Agile	–	Гнучка методологія розробки, яка об’єднує у собі багато практик і підходів, що засновані на дванадцяти принципах Маніфесту гнучкої розробки програмного забезпечення і практичні підходи до його розробки.
Python	–	Високорівнева мова програмування.
Backup	–	Резервна копію для аварійного відновлення даних.
SQL	–	Мова програмування структурованих запитів.
JDK	–	Комплект розробника додатків на мові Java, що включає компілятор, стандартні бібліотеки класів, приклади, документацію, різні утиліти і виконавчу систему Java.
ID	–	Унікальний ідентифікатор.
GitHub	–	Прогресивний веб-сервіс для хостингу репозиторіїв проектів, що дозволяє спільно працювати над ними.
Script	–	Сценарій, що виконує послідовний набір дій.
XML	–	Стандарт побудови мов розмітки ієрархічно структурованих даних для обміну між різними застосунками.
GUI	–	Графічний інтерфейс користувача, який складається з графічних елементів, таких як вікна, різноманітні меню, кнопки та інші віджети.
JVM	–	Віртуальна машина Java – основна частина виконуючої системи мови програмування Java.

ВСТУП

При виборі товару в інтернеті, обранні клініки для діагностики захворювання або рекомендації контенту та можливих друзів в мережі, всюди потрібно якимось чином вимірювати репутацію перерахованих об’єктів. Все, що нас оточує, до чого ми можемо звернутися задля отримання послуги, товару або просто поспілкуватися може в тій або іншій мірі мати репутацію. За допомогою комп’ютерних технологій це все можна достатньо точно вирахувати багатьма способами. Такі особливості сучасного світу, в якому ми живемо.

Характерна людська риса полягає в тому, що репутація інших об’єктів соціальної мережі дозволяє нам зрозуміти кому можна довіряти, а з ким краще не взаємодіяти. В таких умовах, ефективний розрахунок репутації та управління її результатами являє собою цінний ресурс, поряд з фінансовими, матеріальними, людськими та іншими ресурсами.

В першу чергу репутація – це громадська думка про об’єкт, що склалося на основі його минулих дій, про його наміри та норми. Інтерпретуємо це як кількісну оцінку, яка розраховується на основі дій об’єкта, що спостерігаються іншими об’єктами репутаційної системи соціальної мережі. Важливо розділяти поняття репутації та довіри – репутація є об'єктивною величиною, а довіра – ні. Репутація існує у свідомості безлічі об’єктів, довіра – одного суб'єкта довіри. Довіряти можна як на основі репутації, так і всупереч їй, тобто репутація може бути джерелом довіри.

Одним із найважливіших факторів, що впливають на репутацію, є довіра фізичних осіб та організацій. У процесі взаємовідносин формуються загальні погляди, уявлення, думки, інтереси, вони діляться інформацією, міркуваннями, приймають спільні рішення. Тривалі продуктивні зв'язки, зумовлені роздільними матеріальними і моральними цінностями, створюють передумови довірливих відносин і повторних зв’язків. Це декілька з факторів, що можуть бути присутні у моделі репутаційної системи соціальної мережі.

Ніхто не знає повного обсягу тих коштів, що витрачаються на підтримку репутації багатьох компаній, особистостей чи фірм. Але можна навести чимало прикладів, які б підтверджували той факт, що ці витрати однозначно виправдані негативним ефектом стихійного формування репутації.

Особливе значення репутація має для великих і добре відомих організацій. Така організація постійно в центрі уваги громадськості. Їй нема куди сховатися від суспільного осуду і неможливо безслідно розчинитися в ринковій стихії. Тому великі організації постійно працюють із громадською думкою, використовуючи як власні сили, так і залучаючи зовнішні агентства.

Під «використанням» репутації мається на увазі не лише обчислення репутації та її моніторинг або штучна зміна, а й забезпечення підтримки прийняття рішень за допомогою вихідних даних моделей.

Глобальний розвиток інтернету та розвиток платіжних систем на початку XXI століття спричинили появу нової бізнес-моделі для спільного використання благ. Однією з форм реалізації цієї моделі є онлайн-платформа, посередник, що надає послуги зв'язку між об'єктами даної мережі. Очевидно, що ключовим аспектом у такій моделі є довіра об'єктів один до одного.

Репутаційні моделі – це моделі, що обчислюють репутацію кількісно чи використовують значення репутації на вирішення інших завдань. Відповідно, у цій роботі розглядаються моделі, які стосуються репутації, а не довіри.

Актуальність теми. Спілкування, фізичний контакт та обмін інформацією між особами присутні в житті людини з самого початку зародження людства. Але з розширенням нашої популяції перед людством постає проблема швидкої та якісної комунікації.

В процесі формування нових носіїв інформації та способів її розповсюдження, все більш стрімко почали розвиватися новітні засоби для досягнення швидкого обміну інформації. З появою комп'ютерних технологій інформація почала зберігатись в електронному вигляді, а оброблятись та переноситись на електронних носіях. Винаходження всесвітньої мережі та

інтернету дозволило ще легше та зручніше її передавати, в тому числі на більшу відстань та ширшому колу осіб. Коли людство досягне надвеликої потужності в обчисленні даних, чи зможемо ми прогнозувати наслідки взаємодій між соціальними одиницями? Багато людей розходяться в думках, коли намагаються відповісти на це питання. І дійсно: прогнозувати взаємодію між людьми дуже складно, особливо, коли багато що залежить від зовнішніх чинників та стрімкого розвитку нових способів комунікації і різних проявів репутації в соціумі. Але, тим не менш, саме по собі явище репутаційної системи соціальної мережі підпорядковується певним фундаментальним правилам і деяку схожість в динаміці різних соціальних груп ми можемо вже сьогодні спостерігати в реальному часі.

Під час виконання цієї роботи буде досліджена та створена програмна реалізація комп'ютерної моделі репутаційної системи соціальної мережі з врахуванням широкого кола різних факторів та які будуть мати різні стани та характеристики. Комп'ютерна модель повинна працювати на основі деякого набору початкових даних, що якісно налаштовується для отримання певної користі від результатів. В цю систему буде закладено основні принципи репутаційної мережі, а також можливість самостійно вносити зміни до всіх початкових даних та параметрів соціальної мережі.

Репутація є однією з основних характеристик об'єктів суспільного життя, а її роль продовжує зростати. Вона дозволяє пом'якшити інформаційну асиметрію якщо розглядати відносини між об'єктами як вигідні один одному, наприклад, через сигнали для споживачів або штрафні санкції для тих, хто пропонує певні послуги.

Зараз більшість рішень на ринку, що використовують репутацію, реалізує лише збір інформації про репутацію та розрахунок рівня репутації. Моделювання репутаційних систем соціальних мереж для ухвалення стратегічних рішень не пропонується.

Мета й завдання дослідження. Метою роботи є дослідження та програмна реалізація комп'ютерної моделі репутаційної системи соціальної мережі.

Для досягнення поставленої мети визначена програма дослідження, що складається з наступних завдань:

а) огляд існуючих систем репутаційної системи соціальної мережі;

б) дослідження системи репутаційної системи соціальної мережі;

в) програмна реалізація комп'ютерної моделі репутаційної системи соціальної мережі.

Об'єктом дослідження є процес симуляції репутаційної системи соціальної мережі в комп'ютерних моделях.

Предметом дослідження є методи та алгоритми комп'ютерного моделювання репутаційної системи соціальної мережі.

Методи дослідження базуються на теорії об'єктно-орієнтованого програмування, теорії алгоритмів, методах комп'ютерного моделювання, а також теорії розповсюдження інформації.

Наукова новизна отриманих результатів. У процесі рішення завдань, обумовлених цілями дослідження, отримані наступні результати:

1. Розроблено метод комп'ютерного моделювання репутаційних систем соціальних мереж, який відрізняється від існуючих параметризацією об'єктів мережі, що дозволяє слідкувати за тим, як різні чинники впливають на репутацію однакових за характеристиками об'єктів.

2. Розроблено вітчизняний продукт комп'ютерного моделювання репутаційних систем соціальних мереж, який має більш широкі можливості, на відміну від існуючих аналогів.

Практична цінність отриманих результатів полягає в тому, що розроблені алгоритми дозволять успішно вирішувати задачі прогнозування взаємодій, вибору соціальних груп та виявлення закономірностей в процесах, що відбуваються в репутаційних систем соціальних мережах. Також результати

роботи можуть використовуватися фахівцями в соціальній сфері, сфері маркетингу та в якості допоміжного інструменту в наукових дослідженнях даної тематики.

Подальші дослідження у сфері використання репутаційних моделей можуть використовувати приведені у даній роботі напрацювання. Отримані результати можуть бути корисними компаніям, що постачають корпоративні IT-рішення, які працюють з репутацією, а також компаніям, які потребують впровадження рішення щодо використання репутації для прийняття рішень.

Дослідження процесів моделі репутаційної системи соціальної мережі є важливим завданням у наш час. Ця модель дозволяє ефективно маніпулювати характерними особливостями людей, адже люди бажаюсь довіряти тим, з ким вони найбільш часто взаємодіють.

Достовірність наукових результатів підтверджена теоретичними викладками, даними комп'ютерного моделювання, коректними дослідженнями параметрів на функціонуючій обчислювальній моделі, а також відповідністю отриманих результатів окремим результатам, наведеним у науковій літературі.

Таким чином, виходячи з вищеперерахованого, дослідження та програмна реалізація комп’ютерної моделі репутаційної системи соціальної мережі є актуальною задачею, яка потребує вирішення у даній кваліфікаційній роботі.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1 Призначення системи

Під час написання магістерської роботи, тема дослідження та програмної реалізації комп'ютерної моделі репутаційної системи соціальної мережі є дуже актуальною. Репутаційна система соціальної мережі використовується на багатьох торгових майданчиках, а також в різних спільнотах питаннях і відповідях. Навіть у повсякденному житті ми постійно нею користуємось, і не замислюємось про це, адже ми постійно взаємодіємо з іншими людьми і репутаційна система вибудовується самостійно. В час швидкого розвитку обчислювальних машин не завадить розібратися в принципах репутаційної системи соціальної мережі, і спробувати провести власне дослідження використовуючи програмну реалізацію комп'ютерної моделі репутаційної системи соціальної мережі шляхом проведення розрахунків з різними наборами початкових даних.

Хочеться підкреслити, що наведена нижче розроблена програмна реалізація комп'ютерної моделі репутаційної системи соціальної мережі являється спрощеною та не претендує на реальне відображення ситуації з навколишнім світом, але вона має великі перспективи на модернізацію та розвиток задля отримання близьких результатів до тих, що спостерігаються. Мета даної роботи – представити модель, на основі якої можна навчитися основним вмінням моделювання складних систем. Навчальні системи повинні бути достатньо простими, щоб в них можна було розібратися за достатньо короткий проміжок часу та охопити основні чинники впливу та процеси даної теми. Така система повинна мати невелику кількість параметрів, сенс яких був би достатньо зрозумілим.

Розроблювана модель відноситься до ситуацій, коли в процес розвитку репутаційної системи соціальної мережі не втручаються штучні чинники по

перешкоджанню цьому процесу. Це дає можливість спостерігати репутаційну систему соціальної мережі в деякій обмеженій області. Використовувані числові значення не обов'язково відповідають якимось реальним значенням.

При розгляді малої за обсягом репутаційної системи соціальної мережі можна знехтувати зміною демографічної структури населення за час спостереження та зовнішнім впливом на мережу.

Зовнішнім чинником початку репутаційної системи соціальної мережі є будь-яка взаємодія серед жителів міста або їх спілкування в інтернеті. Моделювання даної системи починається з моменту його виникнення до моменту, коли суттєві зміни більше не будуть відбуватися і в систему не будуть додаватися інші об’єкти для спостереження.

Поняття репутаційної системи соціальної мережі можна розуміти по різному. Репутаційна система – це принцип, що дозволяє користувачам оцінювати один одного в онлайн-спільнотах, тим самим підвищуючи або знижуючи рівень довіри до оцінюваного. Також це взаємозв’язки людей як у реальному світі, так і у соціальних мережах чи на сайтах з системою рейтингів.

Основним процесом взаємодії є процес контакту між об’єктами спостереження. Об’єктом спостереження може бути людина в соціуму або в інтернет просторі, аканти на торгових майданчиках чи сайтах, де за кожним зберігається репутація. Також спостерігати можна за об’єктами, які пропонують свої послуги в різних сферах життя, що охоплюють велике коло звичайних людей.

Система онлайн репутації – це технологія, яка дозволяє по-новому і ефективно маніпулювати характерними особливостями людей. Такі системи з'явилися в результаті бажання людей довіряти тим, з ким вони взаємодіють як онлайн, так і в реальному світі.

Дана система знадобиться тим, хто прагне самостійно дізнатися про принципи репутаційної системи соціальної мережі та хоче спостерігати за динамікою змін в даній системі, дізнатися про чинники, які впливають на зв’язки

між об’єктами, та, вносячи зміни до початкових параметрів або в процесі її виконання, самостійно створити модель репутаційної системи соціальної мережі.

1.2 Область застосування

Головна мета розроблюваної системи полягає в описі, дослідженні, програмній реалізації комп’ютерної моделі репутаційної системи соціальної мережі та демонстрація її працездатності і можливості аналізувати за допомогою неї закономірності процесів в залежності від різних початкових параметрів.

Комп’ютерна модель, яка була б близькою до реальної, з усіма можливими факторами впливу, являється дуже складною задачею. Розроблювана система, а також демонстрація особливостей математичного моделювання, являє собою зручний інструмент для використання в різних процесах нашого життя.

По результатам роботи програмної реалізації комп’ютерної моделі репутаційної системи соціальної мережі можна зробити висновки про необхідність прийняття тих чи інших соціальних інструментів, наприклад, плітки, дозволяють нам розуміти, кому довіряти, кому довіряють інші, хто важливий, а хто вирішує, хто важливий.

Отриманий досвід буде корисним при прийнятті рішень управління та планування якості контенту і взаємодій.

Репутаційні системи лежать в основі тенденції збільшення потреби у допомозі з прийняттям рішень при користуванні послугами, що надаються через інтернет. Зі зростанням популярності інтернет-магазинів, спільнот питань і відповідей та інших місць обміну інформацією репутаційні системи стали одним з ключових чинників, що визначають якість онлайн-досвіду.

Ідея репутаційних систем полягає в тому, що навіть якщо споживач не має можливості фізично випробувати продукт або послугу або проконсультуватися з іншою людиною, він все одно здатний скласти реалістичні очікування від товару або послуги, спираючись на довіру до репутаційної системи. Дана робота

покликана дати певний досвід у прогнозуванні та аналізі взаємодій компонентів такої мережі.

Таким чином, виходячи з вищеперерахованого, дослідження та програмна реалізація комп'ютерної моделі репутаційної системи соціальної мережі є актуальною задачею, яка потребує вирішення у даній науковій практиці.

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

2.1 Огляд існуючих систем, технологій, архітектур та програмних рішень за профілем кваліфікаційної магістерської роботи

У ході вивчення літератури з репутаційних моделей головна увага приділялася їх передумовам та завданню, яку автор моделі пропонує вирішити з її допомогою. Розглянуті моделі використовують методи теорії графів і методи, які залучають репутацію до вирішення інших завдань. У результаті вивчення літератури цього напряму було розглянуто 9 моделей.

Існують підходи та поради щодо репутаційної системи соціальної мережі, а також інструменти, за допомогою яких можна спростити шлях до досягнення потрібної мети. Їх короткі описи наведені нижче.

Банки репутації

Зростаюча економіка спільної участі підвищує значущість довіри на рівноправних торгових майданчиках та сервісах. Користувачі можуть заробляти репутацію на одних сайтах, але зазвичай не мають можливості перенести її на інші. Це всього лише питання часу, перш ніж з'явиться якась мережа, яка об'єднує репутаційний капітал по декількох каналах спільного споживання. Такі системи часто називають банками репутації. Вони намагаються надати користувачам платформу для управління своїм репутаційних капіталом відразу на декількох сайтах.

Стохастична модель

Стохастична модель – це математична модель, в якій параметри, характеристики стану об'єкта і умови функціонування об'єкта, що моделюється, представлені випадковими величинами і пов'язані випадковими, непостійними залежностями; або вихідна інформація також представлена стохастичними величинами.

Отже, характеристики стану об'єкта в моделі визначаються не однозначно,

а через закони розподілу їх ймовірностей. Моделюються стохастичні процеси в теорії масового обслуговування, а застосовуються в мережевому плануванні та управлінні і в інших областях.

При побудові даної моделі зазвичай застосовуються методи регресійного і кореляційного аналізів, а також деякі інші статистичні методи.

Детермінована модель

Детерміновані моделі, як правило, призначені для поглиблення в певний основний механізм або природний процес. Вони відрізняються від статистичних моделей (наприклад, лінійної регресії), метою яких є практично оцінити відносини між змінними. Детермінована модель розглядається як корисне наближення до дійсності, яку простіше побудувати і інтерпретувати, ніж стохастичну. Проте, такі моделі можуть бути надзвичайно складними через велику кількість вхідних і вихідних даних, і, отже, часто незворотними; фіксований єдиний набір вихідних даних може бути отриманий за допомогою декількох наборів вхідних [2]. Таким чином, використання надійних параметрів і невизначеності моделі є вирішальним можливо, навіть більшою мірою, ніж для стандартних статистичних моделей, але це та область, яка отримала мало уваги зі статистики.

У математичному моделюванні, детерміновані моделі не містять випадкових величин і ступеню випадковості, і в основному складаються з рівнянь, наприклад диференційних. Ці розрахунки мають відомі вхідні дані і вони стають наслідком унікального набору вихідних, на противагу стохастичному моделюванню, яке містять в собі випадкові змінні.

Детерміновані моделі використовують в наукових дослідженнях, які ми можемо знайти в різних дослідженнях: в сфері населення, розвитку клімату, забруднення навколишнього середовища, а також в інших областях, як інженерія, хімія і ведення політики.

Детерміновані моделі отримали увагу в літературі по статистиці під загальною темою комп'ютерних експериментів. Комп'ютерні експерименти

моделюють складні системи, що вимагають певної кількості вхідних даних. Використання стохастичної системи набагато дешевше, але також є неточним і спрощеним способом.

Корпоративна репутація

У більшості досліджень, пов'язаних з бізнесом, репутація представляється у дискретному чи безперервному вигляді. Дискретне уявлення більше підходить для випадків, що вимагають лише простого судження хорошої проти поганої репутації. Однак згодом стало зрозуміло, що репутація набагато складніше явище, ніж здавалося. Безперервне уявлення визначає репутацію як чисельну міру чи актив. Крім того, репутація може бути розглянута як узагальнене враження або інформація про це враження, накопичена будь-яким об’єктом з репутацією. Існує декілька основних концептуальних ідей даного явища. Деякі з них розрізняють репутацію на основі соціальних очікувань, корпоративної культури та причин довіряти та не довіряти. Такий підхід має низку корисних моментів на практиці, але має справу в основному з психологічними та соціологічними питаннями, які важко описати математично.

У теоретико-економічному підході репутація сприймається як сигнали чи характерна риса. Ці сигнали можуть виникати в різних контекстах. Іншим можливим типом сигналу є оренда репутації, наприклад, найм топ-менеджменту, що зарекомендував себе.

Прихильники стратегічного погляду на репутацію намагаються оволодіти конкурентною перевагою за неї. Проте вони розглядають репутацію всередині ринку та ринкових як спосіб створення бар'єрів розвитку для конкурентів.

З погляду маркетингового підходу репутація допомагає створювати необхідний образ товару чи послуги споживача, цим беручи участь у його просуванні. Крім того, репутація може бути мірою ефективності рекламних компаній.

Репутація може застосовуватись і для управління персоналом. Так можна використовувати репутацію компанії для транслювання її цінностей і культури,

зв'язування міжнаціональних підрозділів. А також, репутація окремих працівників та колективна репутація можуть використовуватись для формування команд.

Нарешті, обліковий підхід розглядає репутацію як нематеріальний актив. Найбільш важливими проблемами тут є управління репутаційними ризиками та облік репутації. Тут важливим запитанням залишається як оцінювати приріст репутації та її цінність для компанії та як розрахувати ефективність інвестицій у неї.

Колаборативна фільтрація

Колаборативна фільтрація – це метод, який використовується деякими рекомендаційними системами. Цей термін має два значення: вузьке і більш загальне. В цілому, колаборативна фільтрація – процес фільтрації інформації або зразків за допомогою методів за участю співробітництва між декількома агентами, точками зору, джерелами даних.

Застосування колаборативної фільтрації, як правило, пов'язане з дуже великими наборами даних. Колаборативні методи фільтрації були застосовані до різних видів даних, зокрема до таких як зондування та моніторинг даних, які виникають при розвідці корисних копалин на великих площах; до фінансових даних, таких як установи фінансових послуг, які об'єднують багато фінансових джерел; або в електронній торгівлі та вебдодатках, що зосереджуються на даних користувача, і т. д. Решта цієї дискусії зосереджена на колаборативній фільтрації даних, призначених для користувача, хоча деякі з методів та підходів можуть застосовуватися так само і у багатьох інших випадках.

У більш новому, вужчому значенні колаборативна фільтрація – це один з методів побудови прогнозу в рекомендаційних системах, який використовує відомі уподобання (оцінки) групи користувачів для прогнозування невідомих уподобань іншого користувача. Основне припущення колаборативної фільтрації полягає в наступному: ті, хто однаково оцінювали будь-які предмети в минулому, схильні давати схожі оцінки інших предметів і в майбутньому. Наприклад, за

допомогою колаборативної фільтрації музичний додаток здатний прогнозувати, яка музика сподобається користувачеві, маючи неповний список його уподобань (симпатій та антипатій).

Прогнози складаються індивідуально для кожного користувача, хоча інформація, що використовується, зібрана від багатьох учасників. Це відрізняє колаборативну фільтрацію від більш простого підходу, дає усереднену оцінку для кожного об'єкта інтересу, наприклад того, що базується на кількості поданих за нього голосів. Дослідження в даній області активно ведуться і в наш час, що зокрема обумовлюється наявністю невирішених проблем у методі колаборативної фільтрації.

Колаборативна фільтрація заснована на пам'яті – цей підхід використовує дані про рейтинг користувача для розрахунку схожості між користувачами або предметами. Він використовується для вироблення рекомендацій. Це був початковий підхід, що використовувався в багатьох торгових системах.

Колаборативна фільтрація заснована на сусідстві – алгоритм, заснований на сусідстві, обчислює подібність двох користувачів або виробів, виробляє прогноз для користувача, приймаючи середнє зважене всіх оцінок. Обчислення схожості між виробами або користувачами є важливою частиною цього підходу. Переваги цього підходу включають в себе: очікуваність результатів, що є важливим аспектом рекомендаційних систем; просте створення і використання; просте полегшення нових даних; добра масштабованість зі співавторами рейтингових пунктів.

Є також кілька недоліків при такому підході. Його продуктивність знижується, коли дані становляться розрідженими, що трапляється часто з виробами, пов'язаними з мережею. Це ускладнює масштабованість такого підходу і створює проблеми з великими наборами даних. Хоча він може ефективно обробляти нових користувачів, тому що спирається на структури даних, додавання нових елементів стає більш складним, що, як правило, спирається уявленням про конкретну складову векторного простору. Додавання

нових елементів вимагає включення нового пункту і повторного включення всіх елементів у структурі.

Колаборативна фільтрація заснована на моделі – Даний підхід надає рекомендації, вимірюючи параметри статистичних моделей для оцінок користувачів. Моделі розробляються з використанням інтелектуального аналізу даних, алгоритмів машинного навчання, щоб знайти закономірності на основі навчальних даних. Число параметрів в моделі може бути зменшено в залежності від типу за допомогою методу головних компонент.

Цей підхід є більш комплексним і дає більш точні прогнози, оскільки допомагає розкрити латентні фактори, що пояснюють спостережувані оцінки. Даний підхід має ряд переваг. Він обробляє розріджені матриці краще, ніж підхід заснований на сусідстві, що в свою чергу допомагає з масштабністю великих наборів даних. Недоліки цього підходу полягають в «дорогому» створенні моделі. Необхідний компроміс між точністю і розміром моделі, тому що можна втратити корисну інформацію у зв'язку із скороченням моделей.

Колаборативна фільтрація гібридного підходу – даний підхід об'єднує в собі підхід заснований на сусідстві і заснований на моделі. Гібридний підхід є найпоширенішим при розробці рекомендаційних систем для комерційних сайтів, так як він допомагає подолати обмеження початкового оригінального підходу (заснованого на сусідстві) і поліпшити якість прогнозів. Цей підхід також дозволяє подолати проблему розрідженості даних і втрати інформації. Однак даний підхід складний і дорогий у реалізації та застосуванні.

Колаборативна фільтрація широко використовується в комерційних сервісах і соціальних мережах. Перший сценарій використання — це створення рекомендації щодо цікавої і популярної інформації на основі врахування «голосів» спільноти. Інша сфера використання полягає у створенні персоналізованих рекомендацій для користувача, на основі його попередньої активності і даних про переваги інших, схожих з ним користувачів.

Соціальний граф

Соціальний граф – це граф, вузли якого представлені соціальними об'єктами, такими як профілі користувача з різними атрибутами, співтовариства, медіа-контент, а ребра – соціальними зв'язками між ними.

Неявний соціальний граф – це такий граф, який можна сформувати на основі взаємодій користувача зі своїми друзями та групами друзів в соціальній мережі. У цьому графі на відміну від звичайного соціального графа немає явної вказівки друзів, тобто немає явних соціальних зв'язків.

Особливості соціального графа характеризується такими метриками, як: метрики взаємин, метрики зв'язків та сегментації. Для вирішення завдань на соціальному графі використовуються спеціальні моделі, за допомогою яких можна замінити «реальні» графи. За допомогою соціальних графів вирішують такі завдання, як: ідентифікація користувачів; соціальний пошук; генерація рекомендацій з вибору «друзів», медіа-контенту, новин, тощо; виявлення «реальних» зв'язків або збір відкритої інформації для моделювання графа. Обробка даних соціальних графів пов'язана з низкою проблем, як наприклад відмінності соціальних мереж, закритість соціальних даних.

Модель середнього значення

Модель заснована на розрахунку репутації об’єкта на основі оцінок, що даються йому в результаті взаємодії з іншими об’єктами. За результатом взаємодії об’єкта має можливість залишити репутацію без змін, збільшити її на певне значення або зменшити її. У різних варіантах моделі результуюче значення – сума позитивних та негативних оцінок або середня оцінка. Такий метод дає досить неточний результат, проте дуже простий у використанні. Модель потребує наявності уніфікованого механізму оцінювання. Також можуть використовуватися додаткові механізми, що забезпечують істинність оцінок. Використовується з різними модифікаціями у великих торгових майданчиках в інтернеті.

PageRank

PageRank – сімейство алгоритмів оцінки важливості веб сторінок за допомогою розв'язання систем лінійних рівнянь. Для кожної сторінки обчислює дійсне число, чим більше число – тим важливіша сторінка.

Замість прямого підрахунку кількості посилань PageRank інтерпретує посилання сторінки А на сторінку Б як голос сторінки А на користь сторінки Б. Після цього PageRank оцінює рейтинг сторінки відповідно до кількості отриманих голосів.

PageRank також враховує значимість кожної сторінки, що отримала голос, адже голоси деяких сторінок є важливішими, і відповідно до цього підвищується значущість сторінки, посилання на яку вони містять. Важливі сторінки отримують більш високу оцінку PageRank і відображаються на перших позиціях результатів пошуку. Для визначення значущості сторінки технологія Google використовує колективний інтелект всесвітньої мережі. Людина не бере участі в обробці результатів. Пошукова система Google не спотворює інформацію про позиції платою за результати пошуку.

За основу PageRank був обраний академічний підхід оцінки важливості публікації автора по числу її згадок в бібліографічних посиланнях інших авторів. Для адаптації до застосування в Інтернет в алгоритм були внесені наступні зміни: вага кожного посилання враховується індивідуально і нормується за кількістю посилань на сторінці. Крім того, PageRank може бути інтерпретовано в термінах випадкового блукання.

Висновки огляду існуючих систем

Для світової економіки, яка побудована на довірі та впевненості у майбутньому, виявилося, що людству доведеться ще багато чому навчитися. В межах України в недостатній мірі представлені вітчизняні розробки в області досліджень та програмної реалізації комп'ютерної моделі репутаційної системи соціальної мережі. У відкритому доступі мало напрацювань на дану тему, тому тема наукової практики є досить актуальною, а вирішення поставленої задачі

допоможе прогнозувати результати репутаційної системи соціальної мережі та вчасно приймати правильні рішення щодо актуалізації, використання та подальшого розвитку і впровадження в сучасне життя.

В ході огляду існуючих систем було виявлено, що всі алгоритми які реалізовані в існуючих репутаційних системах соціальних мереж, або не мають доступних екземплярів, або існують лише у наукових роботах, що не дають змоги звичайному користувачу здійснити самостійне дослідження з використанням комп'ютерної моделі. Тому основна задача полягає у дослідженні та розробці програмної реалізації комп'ютерної моделі репутаційної системи соціальної мережі, яка б мала високу якість, швидкий процес виконання поставлених цілей та легку доступність для звичайного користувача.

2.2 Обґрунтування вибору засобів для побудови системи та мови програмування

В сучасній розробці прикладного комп'ютерного програмного забезпечення досить важливу роль відіграє мова програмування Java. Вона відноситься до найбільш затребуваних і популярних мовам програмування, про що свідчать численні рейтинги і аналіз пропозицій на ринку розробки програмних продуктів.

Мова програмування Java задумувалась як універсальна мова програмування, яку можна застосовувати для різноманітних завдань. Зараз вона перетворилася з просто універсальної мови на цілу платформу та екосистему, яка поєднує різні технології, що використовуються для цілого ряду завдань: від створення десктопних додатків до написання великих веб-порталів та сервісів. Крім того, мова Java активно застосовується для створення програмного забезпечення для багатьох пристроїв: звичайних ПК, планшетів, смартфонів та мобільних телефонів і навіть побутової техніки. Велику роль також відіграє ОС Android, більшість програм для якої пишуться саме на Java. Ще вона

застосовується для роботи з Big Data, розробки програм для наукових цілей, наприклад, обробки мов.

Ключовою особливістю мови Java є те, що код спочатку транслюється в спеціальний байт-код, незалежний від платформи. Потім цей байт-код виконується віртуальною машиною. У цьому плані Java відрізняється від стандартних інтерпретованих мов, код яких відразу виконується інтерпретатором.

Подібна архітектура забезпечує кросплатформенність та апаратну портативність програм на Java, завдяки чому подібні програми без перекомпіляції можуть виконуватись на різних платформах – Windows, Linux, Mac OS та інших. Для кожної з платформ може бути своя реалізація віртуальної машини JVM, але кожна з них може виконувати той самий код.

Віртуальна машина Java діє як додатковий рівень абстракції між платформою Java та базовим апаратним забезпеченням системи користувача. Вихідний код Java може працювати тільки на тих системах, на яких встановлена віртуальна машина Java.

Ще однією ключовою особливістю Java є те, що вона підтримує автоматичне збирання сміття. А це означає, що не треба самостійно звільняти вручну пам'ять від об'єктів, що раніше використовувалися, як в C++, так як збирач сміття це зробить автоматично.

Java є об'єктно-орієнтованою мовою. Вона підтримує поліморфізм, наслідування, статичну типізацію. Об'єктно-орієнтований підхід дозволяє вирішити завдання по побудові великих, але в той же час гнучких, масштабованих додатків з можливістю розширення [86].

Java має багато активних користувачів і спільноту, яка може підтримати розробників коли вони стикаються з труднощами при написанні коду. ПЗ Java також регулярно підтримується та оновлюється. Мова Java пропонує різні інструменти для підтримки автоматизованого редагування, налагодження, тестування, розгортання та керування змінами. Ці інструменти роблять програмування на Java економічним та швидким.

Користувачі можуть завантажувати ненадійний код Java через мережу і запускати його в безпечному середовищі, в якому він не може завдати жодної шкоди. Ненадійний код не може заразити вірусом систему, а також не може читати або записувати файли з жорсткого диска. Рівні безпеки та обмеження у Java також легко налаштовуються.

Для розробки на мові програмування Java потрібний спеціальний комплект інструментів, який називається Java Development Kit. Варто відзначити, що існують різні реалізації JDK, хоча всі вони використовують ту саму мову – Java. Дві найбільш популярні реалізації – Oracle JDK і OpenJDK.

Найбільші їх відмінності зустрічаються в ліцензуванні. Відповідно до ліцензії Oracle JDK можна використовувати безкоштовно для персональних потреб, а також для розробки, тестування та демонстрації програм. В інших випадках потрібна комерційна ліцензія у вигляді підписки. А OpenJDK повністю безкоштовна.

У плані функціоналу та можливостей Oracle JDK і OpenJDK практично не повинні відрізнятися. А ось у плані продуктивності Oracle JDK працює дещо швидше, ніж OpenJDK, але різниця не суттєва.

В даній роботі використовується саме OpenJDK задля розповсюдження розробленої програмної реалізації моделі репутаційної системи соціальної мережі по ліцензії "open-source software". Вихідний код таких програм доступний для перегляду, вивчення та зміни, що дозволяє переконатися у відсутності вразливостей і неприйнятних для користувача функцій а також з’являється можливість додавання своїх власних модулів або повна модернізація застосунка.

SQLite

SQLite – компактна вбудована СУБД, яка підтримує досить повний набір команд SQL. Слово "вбудована" означає, що SQLite не використовує схему клієнт-сервер, тобто двигун SQLite не є окремо працюючим процесом, з яким взаємодіє програма, а являє собою бібліотеку, з якої програма компонується, і двигун стає складовою частиною програми. Таким чином, в якості протоколу

обміну використовуються виклики функцій (API) бібліотеки SQLite. Такий підхід зменшує накладні витрати, час відгуку і спрощує програму. SQLite зберігає всю базу даних (включаючи визначення, таблиці, індекси і дані) в єдиному стандартному файлі на тому комп'ютері, на якому виповнюється програма.

Кілька процесів або потоків можуть одночасно без будь-яких проблем читати дані з однієї бази даних. Запис в базу можна здійснити тільки в тому випадку, якщо ніяких інших запитів в даний момент не обслуговується; в іншому випадку спроба запису закінчується невдачею і повертається код помилки.

SQLite підтримує динамічне типізування даних. Можливі типи значень: INTEGER, REAL, TEXT і BLOB. Також підтримується спеціальне значення NULL. Простота та зручність пошуку SQLite привели до того, що бібліотека використовується в браузерах, музичних плеєрах та багатьох інших програмах.

Коли застосунок використовує SQLite, їх зв'язок проводиться за допомогою функціональних і прямих викликів файлів, що містять дані, а не якогось інтерфейсу, що підвищує швидкість і продуктивність операцій. Добре підходить для розробки і навіть тестування: під час етапу розробки більшості потрібно масштабування. SQLite, зі своїм багатим набором функцій, може надати більш ніж достатній функціонал, при цьому будучи досить простою для роботи з одним файлом і пов'язаної бібліотеки Java.

Простота системи не єдиною гідністю системи, SQLite також є дуже надійною системою, яка стійка до помилок, що дає меншу кількість помилок під час апаратних збоїв.

Для роботи з SQLite на мові програмування Java потрібна зовнішня бібліотека SQLite JDBC – це спеціальна бібліотека, що дає можливість доступу та створення файлів бази даних SQLite на Java. Вона не потребує конфігурації, оскільки рідні бібліотеки для основних операційних систем зібрані в один файл JAR (архів Java).

Java Swing

Коли справа доходить до екосистеми Java, Swing виступає як інструментарій для створення графічного інтерфейсу користувача. Так як Java виконується на великій кількості різноманітних пристроїв, Swing створює зовнішній вигляд, що нагадує більшість платформ. Ці створені компоненти інтерфейсу користувача є не тільки передовими з точки зору зовнішнього вигляду і відчуття, але і за своєю природою підключаються. Це означає, що базова платформа не обмежена певним набором компонентів інтерфейсу користувача. Компоненти інтерфейсу користувача, такі як кнопки, мітки і прапорці, можуть бути легко створені за допомогою API Swing. Таким чином, Swing достатньо гнучкий за своєю природою.

Swing не тільки надає дизайнеру звичайний компонент інтерфейсу користувача, але і просунуті компоненти, такі як панель з вкладками, панелі прокручування, таблиці, дерева і т. д. Компоненти інтерфейсу користувача в Swing повністю розроблені на Java і, таким чином, вони не залежать від платформи. Будь-який розробник може мати доступ до всіх доступних класів Swing з повною документацією в посібнику з API Java. Одним з класичних аспектів Swing є його модульна архітектура, оскільки розробник може створити власну реалізацію стандартних компонентів інтерфейсу користувача, що призведе до нового визначення їх реалізації за умовчанням з використанням концепції успадкування Java.

Основним конкурентом даного інструменту створення графічного інтерфейсу на мові програмування Java є JavaFX – платформа або інструментарій для створення графічних додатків на мові програмування Java. Вона дозволяє створювати програми з багатою насиченою графікою завдяки використанню апаратного прискорення графіки та можливостей GPU.

За допомогою JavaFX можна створювати програми для різних операційних систем і для різних пристроїв. Додаток на JavaFX буде працювати скрізь, де встановлене середовище Java.

JavaFX надає великі можливості в розробці графічного інтерфейсу. Це і великий набір елементів управління, і можливості роботи з мультимедіа, двовимірною та тривимірною графікою, декларативний спосіб опису інтерфейсу за допомогою мови розмітки FXML, можливість стилізації інтерфейсу за допомогою CSS, інтеграція зі Swing та багато іншого.

Java Swing і Java FX активно використовуються в екосистемі Java для створення компонентів інтерфейсу користувача. Вони не залежать від базової платформи, розробники повинні знати про концепції Java і можуть створювати компоненти інтерфейсу користувача. Swing має значну кількість доступних компонентів інтерфейсу користувача, які мають вбудовані класи для кожної потреби. Java FX, з іншого боку, має компоненти інтерфейсу користувача, які спочатку засновані на Swing, так як Swing прибув набагато раніше, ніж JavaFX.

Узагальнивши все вищезгадане, для реалізації комп'ютерної моделі репутаційної системи соціальної мережі було обрано саме стандартний інструмент для Java розробника в створенні графічних інтерфейсів. Swing має складніший набір компонентів GUI, повністю підтримує і надає готові компоненти користувальницького інтерфейсу, тоді як JavaFX має компоненти користувальницького інтерфейсу, які все ще розвиваються. Ще одною перевагою є вбудований графічний редактор інтерфейсів в середовищі програмування IntelliJ IDEA (інтегроване середовище розробки програмного забезпечення).

2.3 Розгорнута постановка завдання

Роботи, присвячені репутації, з'явилися ще у середині XX століття. Але існуючі дослідження або описують розробку моделі для окремого випадку та заздалегідь встановлених бізнес-умов, або висвітлюють виключно технічну сторону впровадження їх у бізнес процеси компаній. Недостатньо уваги приділено саме дослідженню та програмній реалізації комп'ютерної моделі репутаційної системи соціальної мережі.

Згідно з технічним завданням на кваліфікаційну магістерську роботу, реалізації підлягає програмне забезпечення, яке призначено для дослідження та реалізації комп'ютерної моделі репутаційної системи соціальної мережі.

В процесі розробки кваліфікаційної магістерської роботи необхідно виконати наступний обсяг роботи:

а) провести аналіз існуючих систем-аналогів для виявлення їх позитивних і негативних якостей. Результати аналізу врахувати в подальших розробках;

б) вибрати та обґрунтувати методику побудови системи контролю роботи технологічного обладнання на виробництві в автоматизованому режимі. Розробити функціональну та структурну схеми системи;

в) розробити програмне забезпечення системи, що дозволить реалізувати поставлену технічним завданням задачу. Побудувати блок-схеми алгоритмів програми та підпрограми;

г) організувати інтерфейс користувача з метою формування та виводу на екран ЕОМ повідомлень про некоректні дії користувача та нестандартні ситуації в роботі технологічного обладнання;

д) розробити рекомендації по організаційних та методичних заходах, які забезпечать впровадження системи в промислову експлуатацію та її подальшу успішну експлуатацію;

е) провести розрахунки по визначенню економічної ефективності розробленої системи;

ж) розробити заходи по охороні праці при впровадженні та експлуатації системи, а також розробити заходи з цивільного захисту;

з) сформувати висновки про виконаний обсяг робіт та одержані результати.

3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

3.1 Опис функціонування системи

Програмна реалізація комп'ютерної моделі репутаційної системи соціальної мережі складається з одного програмного застосунку, який має зручний графічний інтерфейс. Цей застосунок надає можливість користувачу задати початкові налаштування для подальшої роботи комп'ютерної моделі. Проте, в процесі його виконання користувачу також дається можливість втручатися в роботу застосунка і впливати на результат. Це дає змогу досліджувати ще більше варіантів вихідних даних.

Оперуючи багатьма даними та чинниками, що впливають на роботу моделі та кінцевий результат, комп'ютерна модель відображає статистичну інформацію про всі да, що задіяні в репутаційній системі соціальної мережі.

В подальшому цей застосунок можна покращувати за рахунок закладеної в них масштабованості, завдяки чому вони зможуть оперувати ще більшою кількістю факторів, які можуть впливати на розвиток репутаційної системи соціальної мережі.

Для зберігання даних розробленої програмної реалізації комп'ютерної моделі використовується SQLite 3. SQLite – це вбудована кросплатформна СУБД, яка підтримує досить повний набір команд SQL і має досить зручний програмний інтерфейс для мови програмування Python [6]. Вона є дуже надійною та простою у використанні. SQLite ідеально підходить для використання в даній системі застосунків, тому що ця СУБД швидка, не вимагає спеціальної установки, а база даних зберігається на диску у вигляді одного файлу.

Щоб переглянути структуру та дані, що записані до бази даних, можна використовувати безкоштовний застосунок DB Browser. Цей застосунок являє собою візуальний інструмент з відкритим вихідним кодом для створення, розробки і редагування файлів баз даних, сумісних з SQLite. Структуру

розробленої таблиці даних можна побачити на рисунку 3.1.

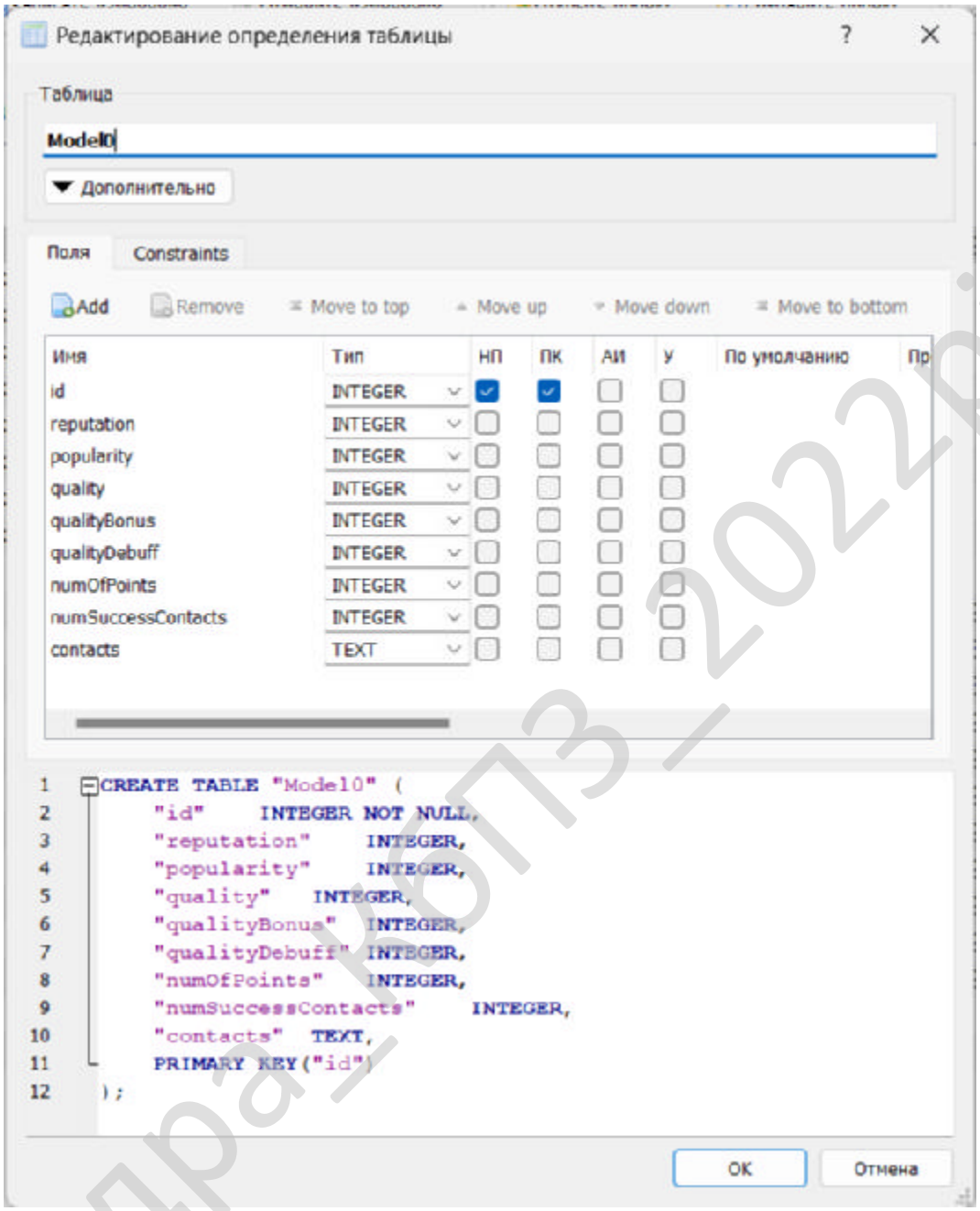

Рисунок 3.1 – Структура таблиці даних в DB Browser

Таблиці, що створюються та використовуються під час виконання розробленого застосунка, мають по 9 полів:

1) ID – унікальний ідентифікатор найменшої соціальної одиниці;

2) reputation – репутація, може бути низькою, середньою та високою. Рахується від 0 до 100. Впливає на ймовірність того, що інші соціальні одиниці

будуть мати бажання зв’язуватися з даним об’єктом репутаційної системи соціальної мережі;

3) popularity – популярність, відображає кількість контактів з іншими об'єктами. Характеризує ступінь популярності. Впливає на якість та хороший результат контакту;

4) quality – якість взаємодії впливає на оцінку репутації, що об’єкт отримає після контакту. Рахується від 0 до 100. Вище значення – вища ймовірність якісної взаємодії;

5) quality_bonus – бонус до якості взаємодії. Розраховується в залежності від популярності. Чим популярніший об’єкт, тим більша ймовірність того, що він залишає хороше враження, адже популярність не здобувається за рахунок негативних чинників;

6) quality_debuff – негативне значення до якості взаємодії. Прямий наслідок низької репутації;

7) numOfPoints – кількість зароблених балів від взаємодії від інших контактів;

8) numSuccessContacts – загальна кількість успішних контактів з іншими об’єктами комп’ютерної моделі репутаційної системи соціальної мережі;

9) contacts – список останніх контактів. Об’єкти репутаційної системи соціальної мережі пам’ятають свої зв’язки з іншими об’єктами. Зберігає ID інших об’єктів та числовий результат взаємодії з ними від попередніх контактів.

Наповнення бази даних може відбуватися однаковими за характеристиками об’єктами. Це дозволяє слідкувати за тим, як різні чинники впливають на однакові за характеристиками об’єкти. Також можна додати більше різноманіття, щоб знаходити закономірності в контрольованому хаосі.

При першому запуску додатку зовнішньої бази даних не існує. Вона буде створена в процесі його виконання для збереження отриманих результатів. Окрім одної основної таблиці, яка має нульовий індекс, створюються допоміжні. В допоміжних таблицях присутні альтернативні результати. Всі вони

узагальнюються в головній таблиці для отримання більш точних результатів в результаті узагальнення багатьох паралельних процесів моделювання репутаційної системи соціальної мережі. Всі результуючі таблиці та структуру деяких з них можна побачити на рисунку 3.2.

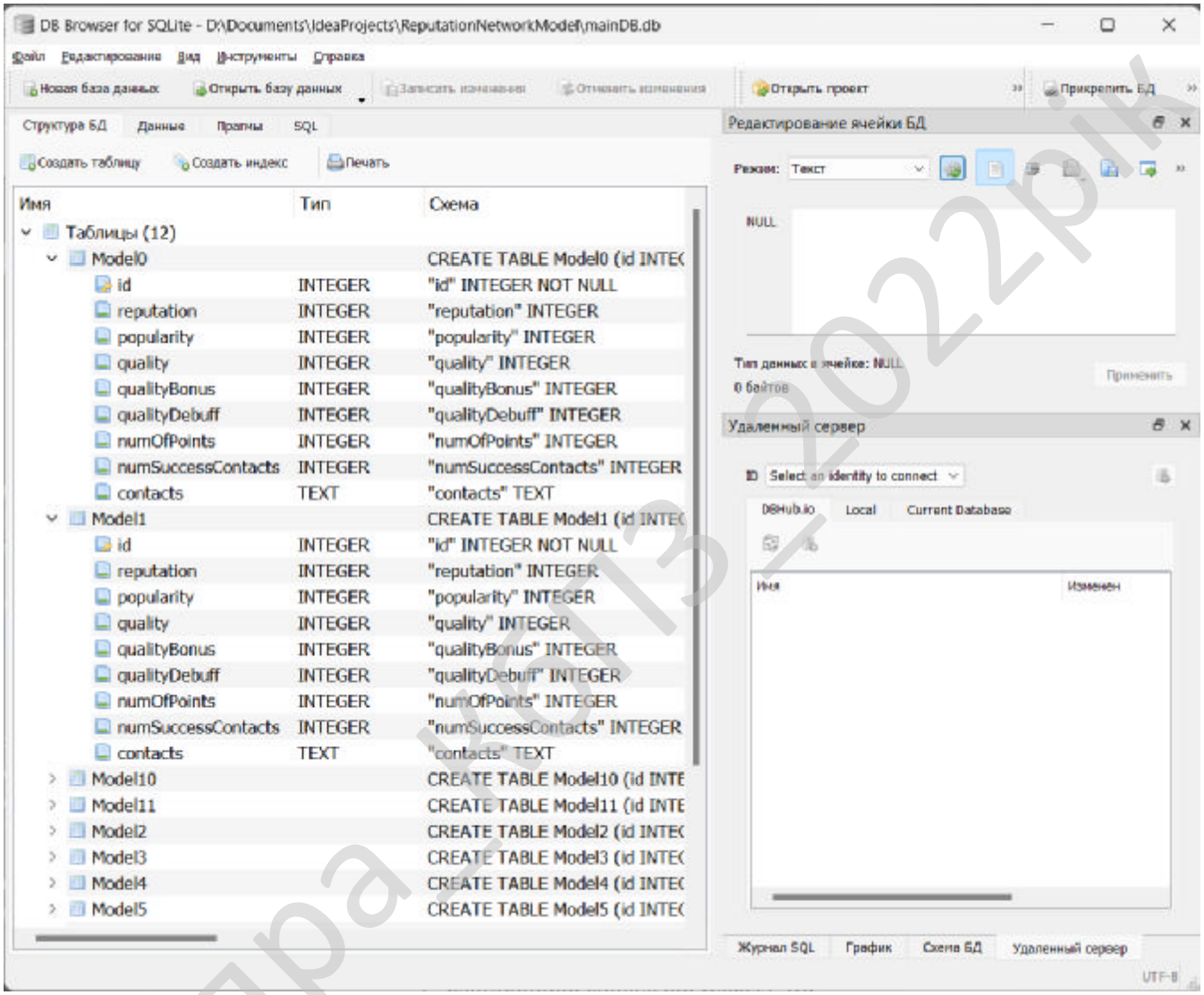

Рисунок 3.2 – Вигляд всіх результуючих таблиць та структура деяких з них

Кількість паралельних обчислень налаштовується в самому додатку. Їх більша кількість дає більш узагальнені та точні результати для введеного набору початкових налаштувань моделі. Структура таблиць однакова, адже вони використовуються для зберігання даних в одному і тому самому форматі, але різними потоками обчислень.

Як вже було згадано вище, всі результати узагальнюються в головній таблиці з індексом нуль. Додаткові таблиці не видаляються для того, щоб можна було ознайомитись з конкретними результатами обчислення деякого потоку.

Приклад даних, що зберігається в базі даних після виконання комп'ютерної моделі репутаційної системи соціальної мережі можна побачити на рисунку 3.3. Це корисно для кращого розуміння структури таблиці, а самі результати краще отримати в розробленому програмному забезпеченні.

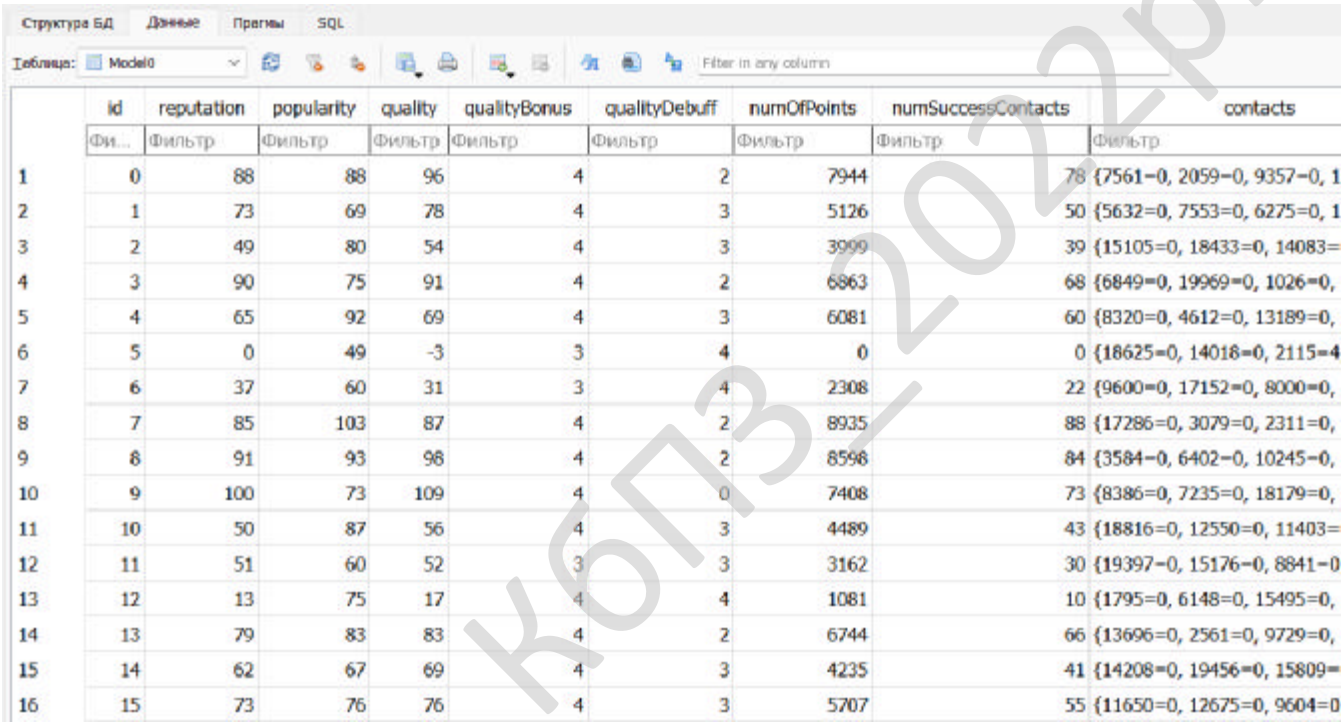
Структура БД | Данные | Прагмы | SQL

Таблица: Model0 | Filter in any column

	id	reputation	popularity	quality	qualityBonus	qualityDebuff	numOfPoints	numSuccessContacts	contacts
	Фи...	Фильтр	Фильтр	Фильтр	Фильтр	Фильтр	Фильтр	Фильтр	Фильтр
1	0	88	88	96	4	2	7944	78	{7561=0, 2059=0, 9357=0, 1
2	1	73	69	78	4	3	5126	50	{5632=0, 7553=0, 6275=0, 1
3	2	49	80	54	4	3	3999	39	{15105=0, 18433=0, 14083=
4	3	90	75	91	4	2	6863	68	{6849=0, 19969=0, 1026=0,
5	4	65	92	69	4	3	6081	60	{8320=0, 4612=0, 13189=0,
6	5	0	49	-3	3	4	0	0	{18625=0, 14018=0, 2115=4
7	6	37	60	31	3	4	2308	22	{9600=0, 17152=0, 8000=0,
8	7	85	103	87	4	2	8935	88	{17286=0, 3079=0, 2311=0,
9	8	91	93	98	4	2	8598	84	{3584=0, 6402=0, 10245=0,
10	9	100	73	109	4	0	7408	73	{8386=0, 7235=0, 18179=0,
11	10	50	87	56	4	3	4489	43	{18816=0, 12550=0, 11403=
12	11	51	60	52	3	3	3162	30	{19397=0, 15176=0, 8841=0
13	12	13	75	17	4	4	1081	10	{1795=0, 6148=0, 15495=0,
14	13	79	83	83	4	2	6744	66	{13696=0, 2561=0, 9729=0,
15	14	62	67	69	4	3	4235	41	{14208=0, 19456=0, 15809=
16	15	73	76	76	4	3	5707	55	{11650=0, 12675=0, 9604=0

Рисунок 3.3 – Приклад даних з результуючої таблиці

Налаштування комп'ютерної моделі репутаційної системи соціальної мережі є чи не найголовнішим кроком в процесі моделювання. Адже саме від початкового стану даних та налаштованих параметрів залежать отримані результати. В розробленому додатку наявний найбільш оптимальний набір параметрів налаштувань. Зрозуміло, що в реальному світі на репутаційні системи впливають дуже багато різноманітних чинників, але більшість з них не мають значного впливу на отриманий результат. Меню налаштувань застосунка для підготовки бази даних можна побачити на рисунку 3.4.

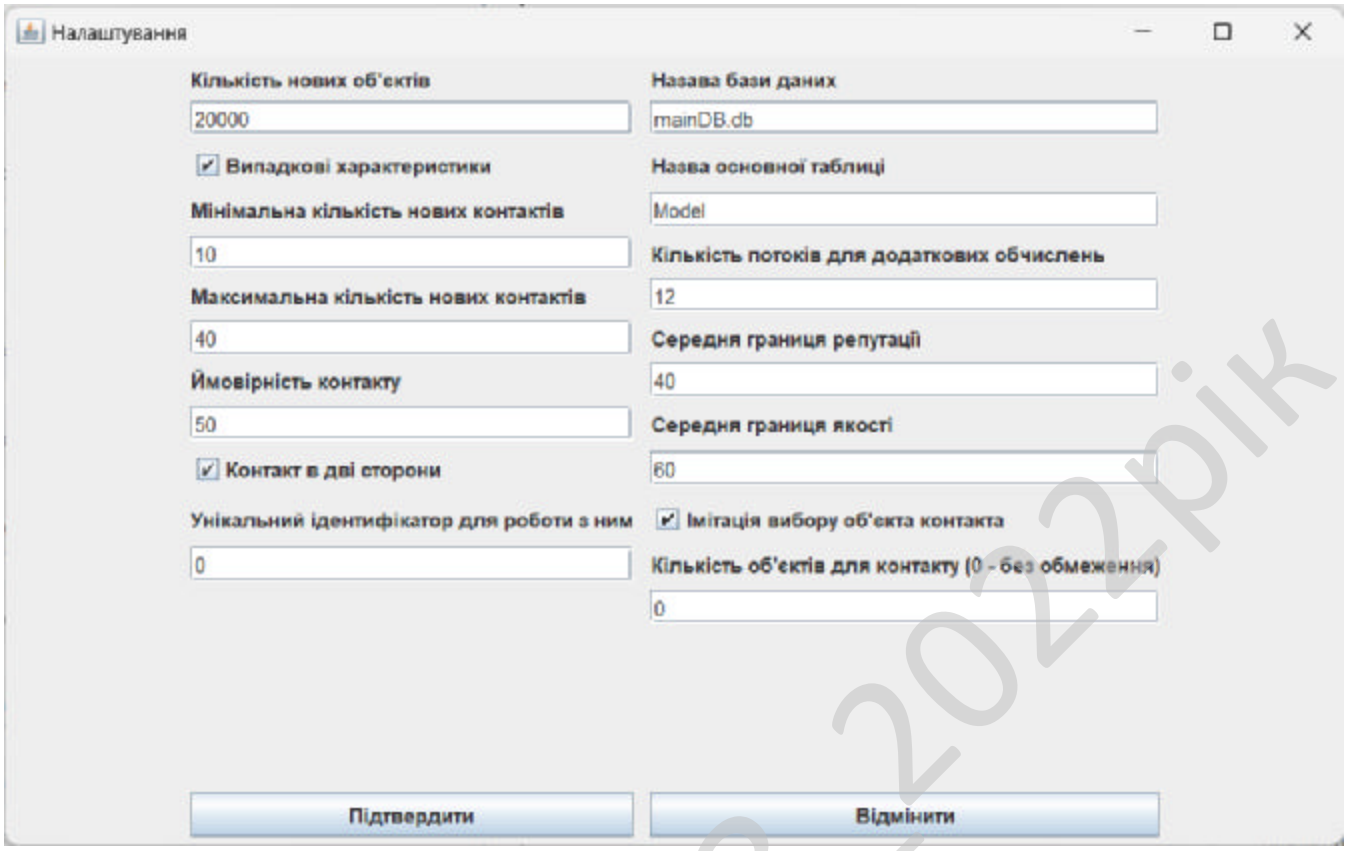

Рисунок 3.4 – Меню налаштувань застосунка

Кількість нових об’єктів вказується для кожного нового процесу додавання об’єктів до моделі. Таких партій може бути стільки завгодно. Репутаційна система соціальної мережі не є замкненою. Під час моделювання, в будь-який момент часу, користувачу ніщо не завадить додати нові об’єкти. Звичайно цього можна і не робити, тоді це вже буде схоже на замкнену систему. Тобто, дане налаштування вносить до процесу моделювання декілька різних підходів, які повинні вплинути на кінцевий результат.

Кожний об’єкт може мати випадкові початкові характеристики для того, щоб ввести в модель трішки контрольованого хаосу та спробувати знайти закономірності в репутаційній системі соціальної мережі, що складається з різноманітних за характеристиками об’єктів.

Мінімальна та максимальна кількість нових контактів обмежує процес створення контактів між об’єктами вказаними значеннями. Можна спробувати створити репутаційну систему соціальної мережі, наприклад, де кожний об’єкт

має або замалу, або завелику кількість контактів з іншими об’єктами. Можна задати такі значення, щоб модель була більш різноманітною.

Прапор контакту в дві сторони дозволяє розглядати два типи достатньо різних репутаційних систем соціальної мережі. Перший тип, коли обидва об’єкти, що приймають участь в контакті, запам’ятовують цю подію і формують враження по один одного, що подальшому впливатиме на їх репутацію. Другий тип дозволяє лише одному контакту запам’ятовувати дану подію і лише у нього формується враження про другий об’єкт. Таким чином, репутація формується лише у другого об’єкту, з яким здійснювався контакт.

В наступному полі можна вказати унікальний ідентифікатор, який буде використовуватися для, наприклад, малювання графіку змін його характеристик, або виведення більш детальної інформації саме для цього об’єкту із бази даних. В вікні більш детальної інформації, окрім її перегляду, можна змінити певні характеристики обраного об’єкту. Це можна зробити як на самому початку, коли об’єкт тільки був доданий до моделі, так і в процесі її виконання. На рисунку 3.5 зображене вікно виведення детальної інформації та її редагування.

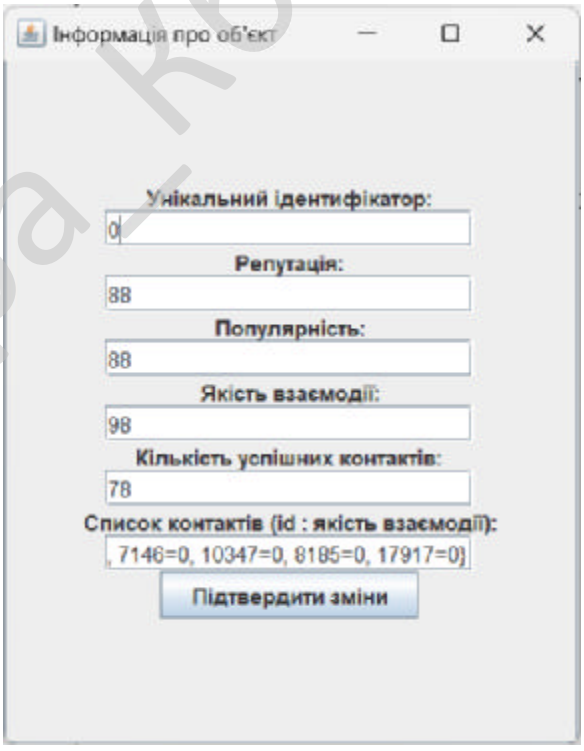

Рисунок 3.5 – Вікно перегляду детальної інформації об’єкту та його редагування

В налаштуваннях можна задати нові назви бази даних та назву таблиці з вихідною інформацією. Назва бази даних грає важливу роль, адже цікаві екземпляри можна зберегти в окрему папку для їх подальшого більш детального дослідження. Можна зробити колекцію з різними базами даних, які були сформовані в результаті різних початкових наборів налаштувань.

Кількість потоків для додаткових обчислень визначає те, наскільки багато буде результуючих таблиць, інформація із яких буде узагальнена для отримання більш точних результатів виконання комп'ютерної моделі репутаційної системи соціальної мережі. Цей процес аналогічний тому, що користувач буде запускати модель з одними і тими самими початковими даними та налаштуваннями багато разів. Але в створеній програмній реалізації комп'ютерної моделі цей момент вже передбачений та є повністю автоматичним. В результаті користувач дуже швидко отримає результат з вказаною точністю.

Середня границя якості взаємодії та репутації налаштовують збір статистики в процесі роботи моделі. Можна дізнатися скільки об'єктів мають ті чи інші значення по вказаним характеристикам. Популярність тут не присутня, тому що вона напряму залежить від часу, який був витрачений на роботу комп'ютерної моделі репутаційної системи соціальної мережі. Чим довше та більше відбувається процесів під час роботи моделі, тим більшу популярність отримують об'єкти. Але деякі об'єкти, звісно отримують або більше, або меншу популярність, в залежності від їх інших характеристик.

Імітація вибору об'єкта для контакту дозволяє ініціатору контакту нібито самостійно обирати інший об'єкт, з яким цей контакт буде відбуватися. Насправді його вибір залежить від репутації інших об'єктів, чим вона більша – тим більший шанс контакту. З вимкненим прапорцем на контакт залишає свій вплив ймовірність контакту, яка була розглянута трохи раніше.

Кількість об'єктів для контакту вказує як багато об'єктів взагалі можуть відповідати на контакт. Тобто, якщо це значення не дорівнює нулю, з'являється обмеження в певну кількість об'єктів і вони, в свою чергу, можуть виступати в

ролі досліджуваних об’єктів. Всі інші об’єкти обирають ціль для зв’язку тільки з зарезервованих об’єктів. Це можна порівняти з тим, як працює репутація в великих гравців в репутаційній системі соціальної мережі, коли репутацією інших об’єктів можна знехтувати. Об’єднавши вказане налаштування з контактом в дві сторони, можна отримати цікаві результати.

На рисунку 3.6 можна побачити меню статистики до початку виконання якихось дій в застосунку.

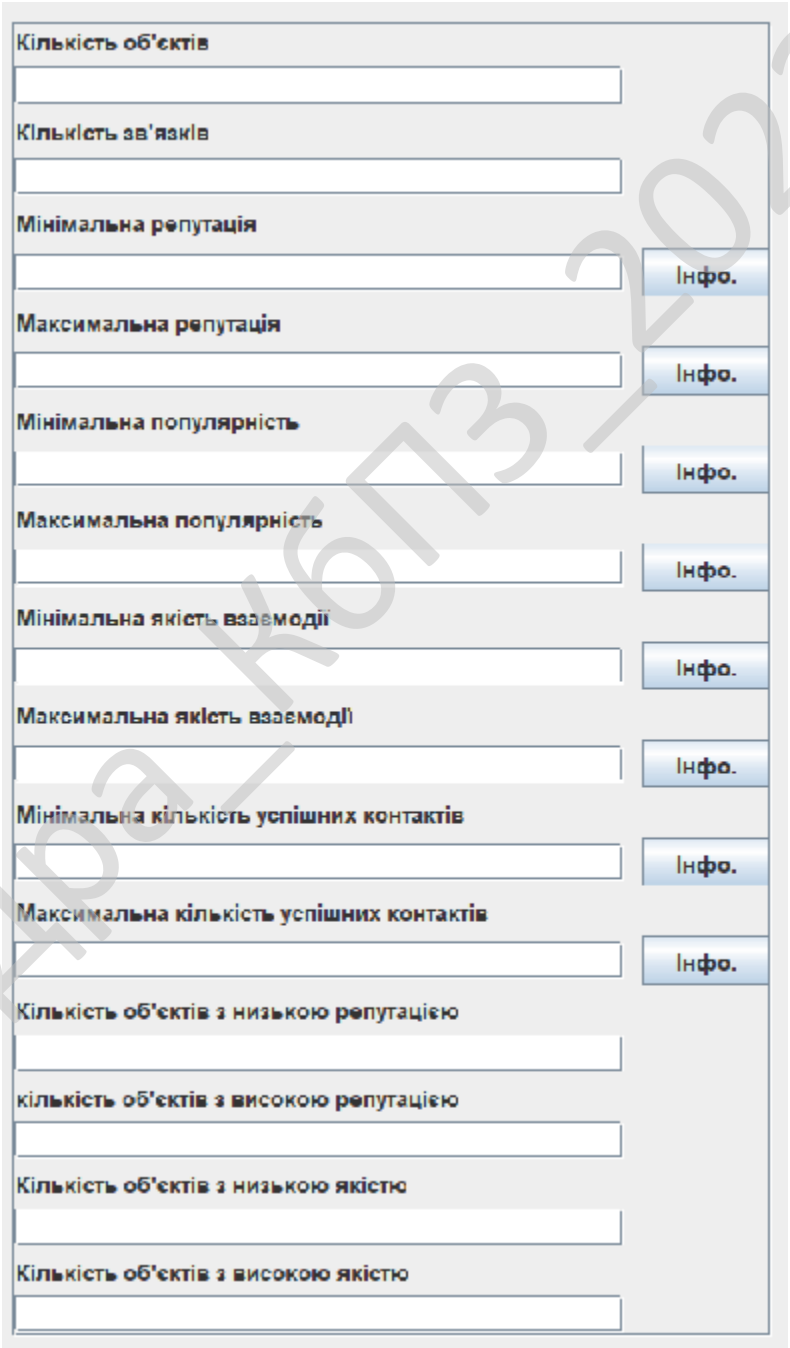

Рисунок 3.6 – Меню статистики

Статистика зберігає інформацію про загальну кількість об’єктів, що приймають участь в моделюванні процесів репутаційної системи соціальної мережі та кількість зв’язків між ними.

Далі йде багато полів з мінімальними та максимальними значеннями по вказаним характеристикам об’єктів. Кожний такий об’єкт можна переглянути більш детально за допомогою кнопки "Інфо." поруч та, за потреби, змінити деякі значення.

Графічний інтерфейс створено за допомогою Java FX з використанням вбудованого в IntelliJ IDEA редактору інтерфейсу. Шаблон зберігається в "*.form" файлах.

Рисунок 3.7 показує можливі дії, які доступні користувачу в процесі роботи з розробленим програмним застосунком.

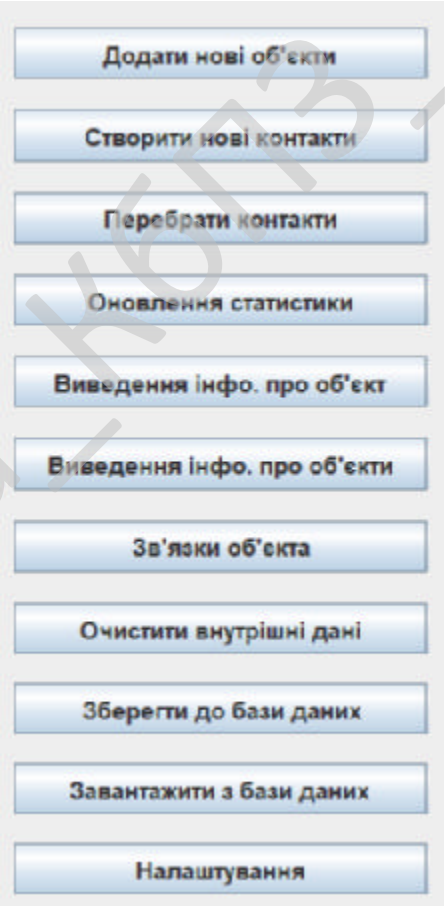

Рисунок 3.7 – Меню команд

Застосунок дає можливість користувачу в будь-який момент додавати нові об’єкти до системи.

При натисненні на другу кнопку об’єкти взаємодіють між собою, створюють нові контакти, оновлюють репутацію в залежності від отриманої якості контакту.

Третя кнопка запускає додаткові потоки для паралельного моделювання одних і тих самих процесів комп’ютерної моделі репутаційної системи соціальної мережі. В результаті всі результати об’єднуються в один, що призводить до більш точного результату моделювання.

Кнопка оновлення статистики в потрібний момент часу оновлює інформацію бокового меню.

Наступні дві кнопки потрібні для виведення інформації про вибраний об’єкт та якомога повної інформації про об’єкти відповідно. Також в меню статистики завжди присутня інформація, яка не залежить від вказаних кнопок і вона може оновлюватися.

Зв’язки об’єкта визначаються за трьома категоріями: зв’язки першого, другого та третього порядку. Можна дізнатися наскільки багато є зв’язків у обраного об’єкта з іншими на різних рівнях.

Очистка внутрішніх даних повертає початковий порожній стан моделі. Корисно для переходу до іншого набору початкових налаштувань щоб не доводилось кожного разу перезапускати додаток.

Збереження до бази даних інформації потрібне для повторного її відкриття в майбутньому для більш детального аналізу або просто щоб зберегти цікавий результат комп’ютерної моделі репутаційної системи соціальної мережі.

Завантаження з бази даних надає можливість продовжити працювати з даними, які вже були збережені до цього. Назва бази даних вказується в налаштуваннях.

Ну і остання кнопка відкриває вікно з налаштуваннями. Елементи даного вікна були розглянуті вище.

Останній, але достатньо важливий елемент графічного інтерфейсу користувача, це графік. На графіку відображається зміна внутрішніх станів обраного об’єкту з плином часу в процесі моделювання репутаційної системи соціальної мережі. Приклад графіку наведено на рисунку 3.8.

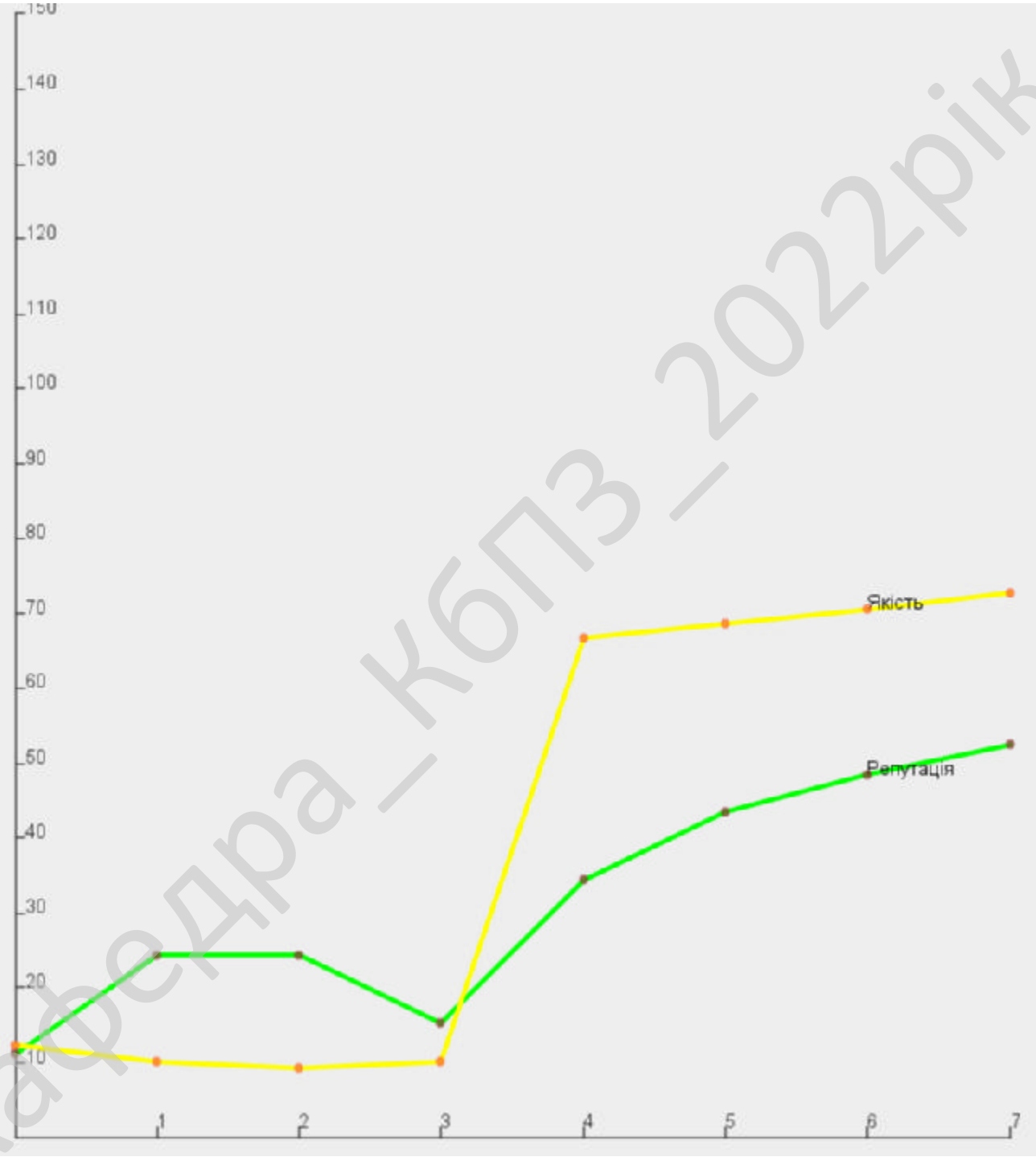

Рисунок 3.8 – Приклад графіку

На рисунку 3.8 зображено як змінюється якість взаємодії з іншими об’єктами та його репутація під час моделювання. Спочатку і якість, і репутація на достатньо низькому рівні. Ще можна сказати, що після початкового зростання репутація почала зменшуватися через низьку якість взаємодії. Але після того, як якість об’єкта зросла (були внесені корективи), репутація також почала зростати.

Кожна з підпрограм виконується в окремому незалежному потоці від графічного інтерфейсу користувача та одна від одної. Це надає змогу графічному інтерфейсу не зависати під час виконання підпрограм.

3.2 Розробка структурної схеми

Структурна схема розробленої програмної реалізації комп’ютерної моделі репутаційної системи соціальної мережі зображена на рисунку 3.9.

Рисунок 3.9 – Структурна схема застосунка

У базу даних можна додати потрібну кількість об’єктів репутаційної системи соціальної мережі. Максимальну кількість користувачів мережі, що буде додано до бази даних, можна задати в налаштуваннях застосунка. Інформація про задані налаштування завжди зберігається в двох форматах: у текстовому файлі

для змоги відновити її, та в оперативній пам’яті, щоб програма могла швидко визначати налаштовані параметри.

Створення результуючих таблиць відбувається, якщо вони не були створені раніше. Очищення результуючих таблиць відбувається для того, щоб видалити застарілу інформацію та звільнити місце для нової.

Застосунок працює з однією і тією ж базою даних. Якщо користувач має декілька готових баз даних, він може в налаштуваннях вказати з якою саме базою потрібно працювати застосунку.

Підготовка бази даних до роботи полягає у створенні потрібних таблиць для моделювання та наповнення їх коректною інформацією. Вікно застосунка у робочому стані можна побачити на рисунку 3.10.

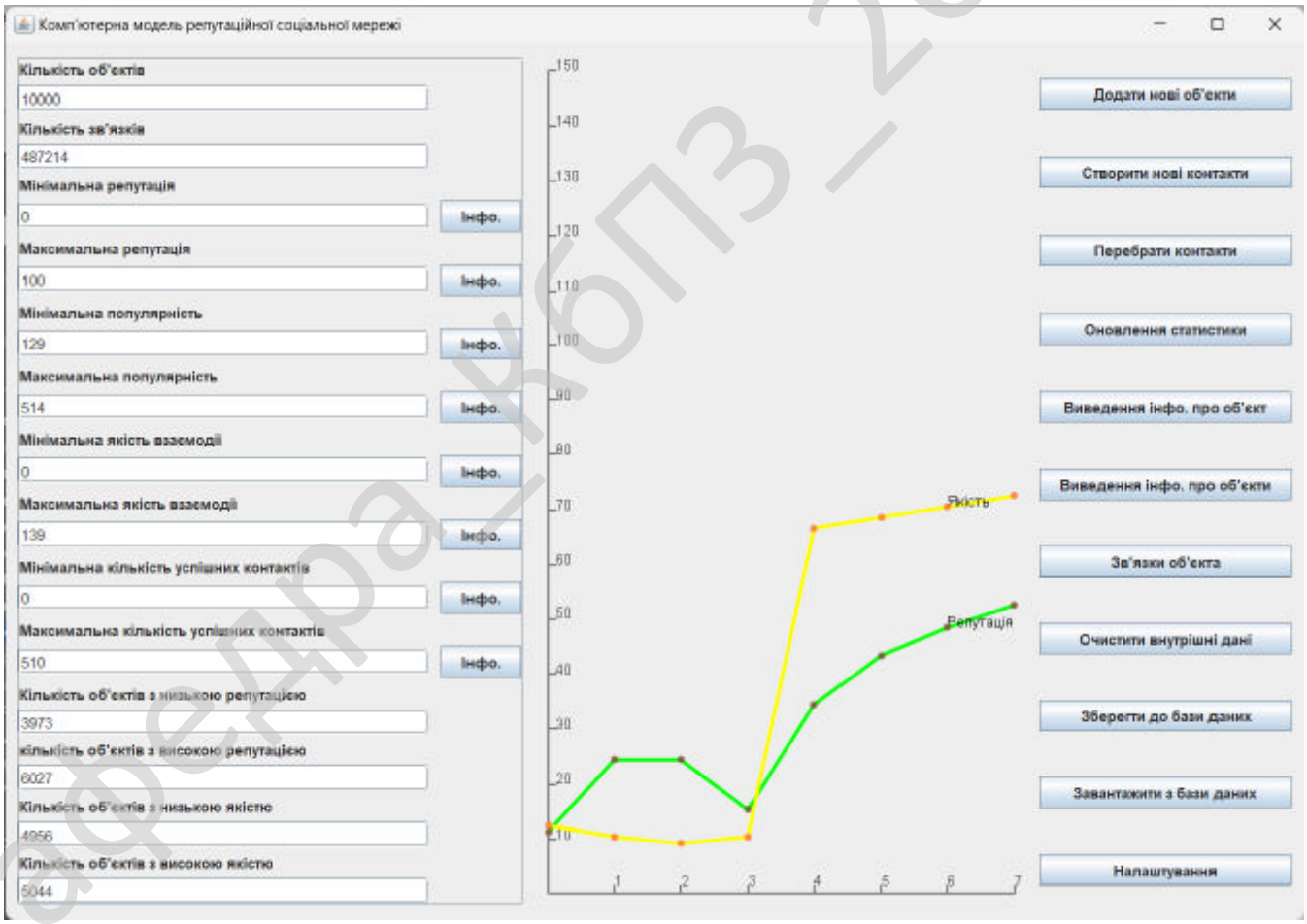

Рисунок 3.10 – Вікно застосунка у робочому стані

В графічному інтерфейсі користувача передбачено багато додаткових вікон з повідомленнями, що дозволяють користувачу краще зрозуміти процес моделювання. Від так, якщо користувач буде робити недопустимі речі, наприклад, вводити помилкові дані в текстові поля, буде виводитися повідомлення про це і внесені зміни не будуть застосовані.

Якщо база даних не створена, або файл з налаштуваннями також не існуватиме, програма коректно опрацює даний кейс і усуне проблему створивши дані файли.

Алгоритм моделювання складається з декількох модулів і додаванням об’єктів, створенням зв’язків та їх перебором він не обмежується. В додатку реалізовано достатньо складний механізм паралельних обчислень результатів для їх узагальнення. Також графічний інтерфейс незалежний від інших процесів, що відбуваються за його межами. Тому весь алгоритм моделювання винесено в окремий модуль, що може легко модифікуватися або взагалі замінюватися.

3.3 Розробка функціональної схеми

Функціональна схема розробленого програмного застосунка, що відповідає за програму реалізацію комп’ютерної моделі репутаційної системи соціальної мережі, зображена на рисунку 3.11.

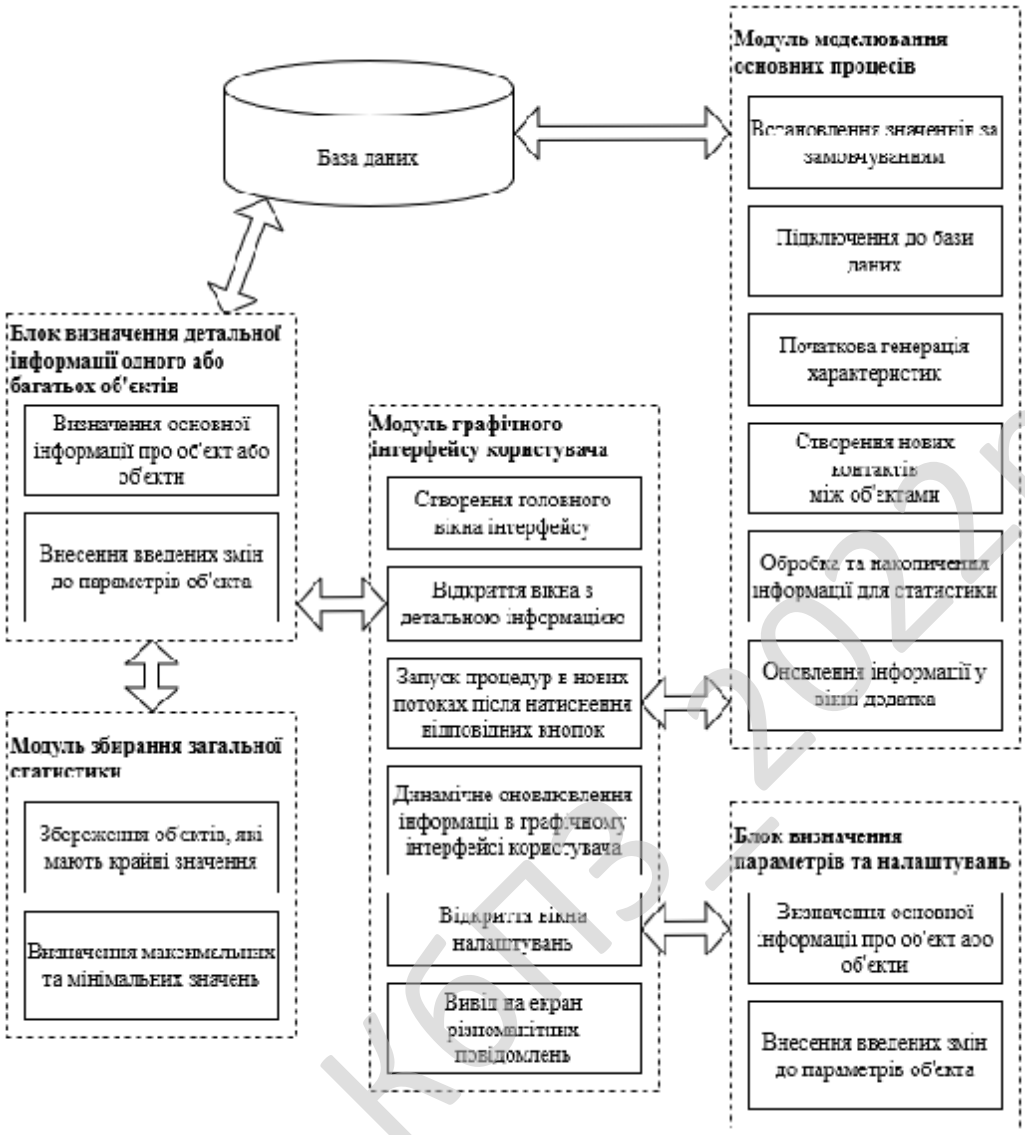

Рисунок 3.11 – Функціональна схема застосунка

З рисунка видно, що розроблена комп’ютерна модель репутаційної системи соціальної мережі представлена наступним чином:

а) блок визначення детальної інформації одного або декількох об’єктів – використовується в двох ситуаціях, коли потрібно знайти детальну інформацію одного об’єкта та загальну інформацію про багато об’єктів. Інформація береться від бази даних. Отримана інформація виводиться на вікно за допомогою динамічної побудови графічних елементів;

б) база даних об’єктів – зберігає усі об’єкти моделі репутаційної системи соціальної мережі. Складається з однієї головної таблиці та багатьох допоміжних, що використовуються в паралельному обчисленні результатів;

в) модуль графічного інтерфейсу користувача – початкова точка для інших процедур та вікон. За допомогою нього відбувається спілкування між користувачем та застосунком, виводиться вся потрібна інформація, відповідає за виведення інших вікон та текстових повідомлень;

г) модуль моделювання основних процесів. Містить основну функціональність, правила та принципи для взаємодії об’єктів. Тісно співпрацює з налаштуваннями, графічним інтерфейсом та базою даних;

д) блок визначення параметрів та налаштувань. Містить в собі всі процеси, що напряму відносяться до задання налаштувань додатку, їх збереження та надання до цих значень доступу іншим частинам програмного засобу.

Точкою входу до даного модуля являєтеся графічний інтерфейс користувача. Від нього можна потрапити до вікна налаштувань, вікна детальної інформації як про один, так і про багато об’єктів, вікно зв’язків по порядкам.

В застосунку можна обрати об’єкт за допомогою його id та подивитися про нього детальну інформацію і переглянути графік зміни його параметрів. Також це можна зробити за допомогою безкоштовного застосунка DB Browser, якщо відкрити ним створену базу даних.

3.4 Розробка діаграми процесів

Діаграма процесів комп’ютерної моделі репутаційної системи соціальної мережі зображена на рисунку 3.12.

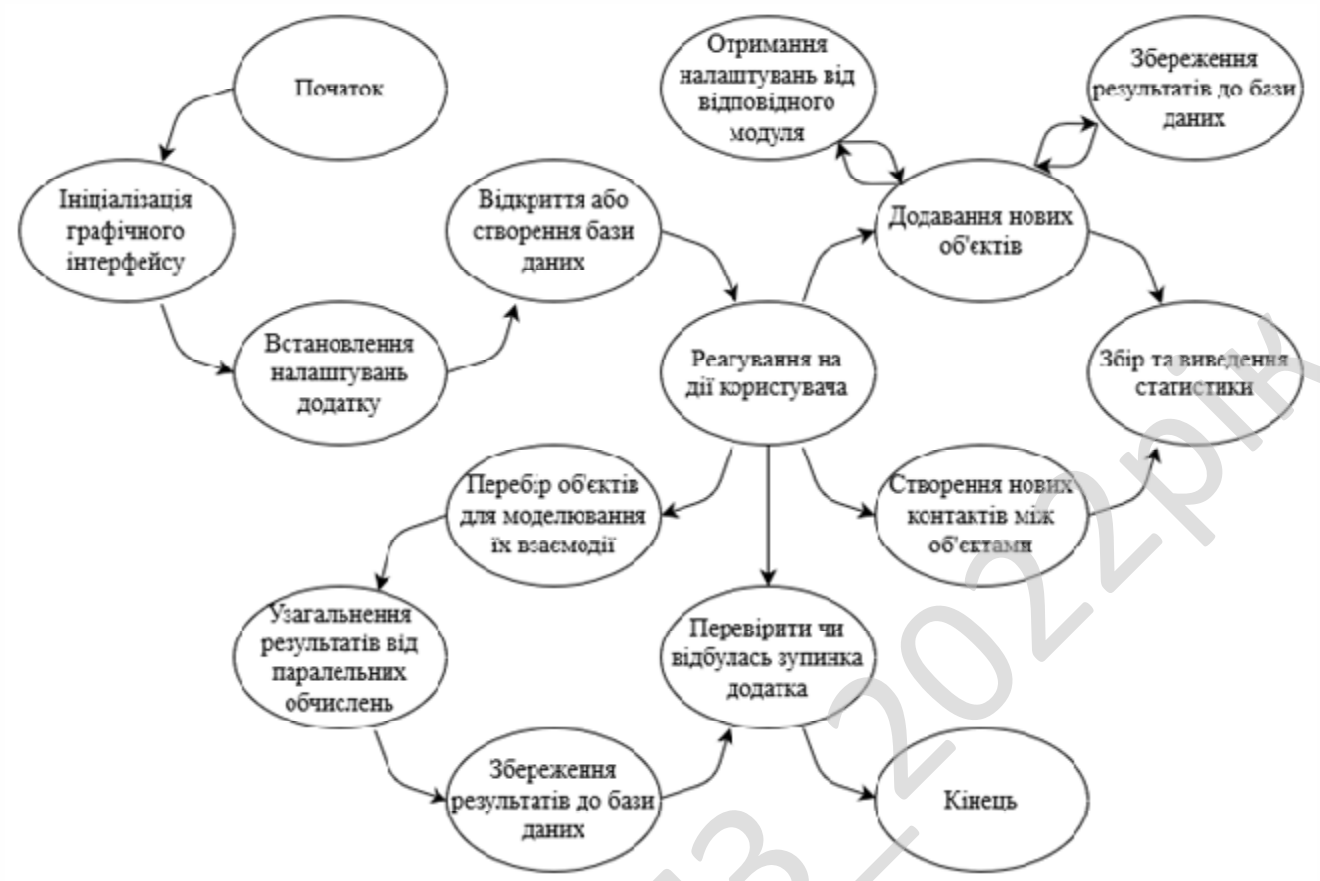

Рисунок 3.12 – Діаграма процесів

З даної діаграми процесів видно, що робота з репутаційною системою соціальної мережі починається з ініціалізації графічного інтерфейсу. Формуються всі групи графічних елементів та їх розташування.

Після цього відбувається початкове налаштування додатку. Самі налаштування можуть зчитатися зі спеціального файлу. Якщо файл налаштувань відсутній, додаток та його параметри будуть мати значення за замовчуванням. Внесені зміни до цих даних зберігаються до вихідного файлу, який буде прочитано при наступному запуску додатка.

Майже схожий процес відбувається із базою даних. Якщо користувач в своєму запасі має готові бази даних від попередніх запусків, їх можна відкрити з самого початку виконання програмного засобу та продовжити дослідження цього набору даних, або продовжити саме моделювання. Якщо база даних не існує, в процесі виконання комп'ютерної моделі репутаційної системи соціальної мережі

буде створено нову базу даних з отриманими результатами. За бажанням користувач може зробити свою бібліотеку з базами даних, що були утворені від різних початкових наборів налаштувань.

З цього моменту система готова до моделювання. Саме моделювання досить гнучке і його алгоритм значно залежить від дій користувача. Адже в процесі виконання моделі користувач може вносити зміни до параметрів моделі, може переглядати всю необхідну інформацію та вирішувати які подальші кроки будуть виконуватися в моделі.

Додавання нових об’єктів до комп’ютерної моделі репутаційної системи соціальної мережі є гнучким, адже об’єкти, їх кількість та поводження гнучко налаштовується. Тому цей процес залежить від тих налаштувань, що ввів користувач. В результаті всі нові об’єкти будуть додані до вже існуючих об’єктів, якщо вони існували.

Створення нових контактів між об’єктами відбувається самими об’єктами моделі, адже саме в них та в налаштуваннях зберігаються всі дані, які визначаюсь кількість нових контактів, яким чином це буде відбуватися, який тип зв’язку і тому подібне.

Збір та виведення статистики поділяється на декілька видів. Є загальна статистика, що виводиться на головному вікні програмного засобу та відображає стан моделі загалом. Також є інформація про конкретний об’єкт, що обрав користувач в налаштуваннях. Цю інформацію можна не тільки переглянути, а ще і змінити її за потреби. Така можливість дає більше контролю над системою та можна створювати різні події щоб відслідкувати як вона вплине на комп’ютерну модель репутаційної системи соціальної мережі. Третій тип статистики виводить на екран інформацію про кожний об’єкт, що знаходиться в системі.

Корисним типом виведення статистики про об’єкт є підрахунок зв’язків обраного об’єкту по порядкам. Таких порядків визначається три, але для формування репутаційного графу використовується лише перший, адже з іншими порядками кількість об’єктів стає дуже великою, що ускладнює розуміння

системи. Приклад вікна інформації про зв’язки об’єкта та репутаційного графу можна побачити на рисунку 3.13.

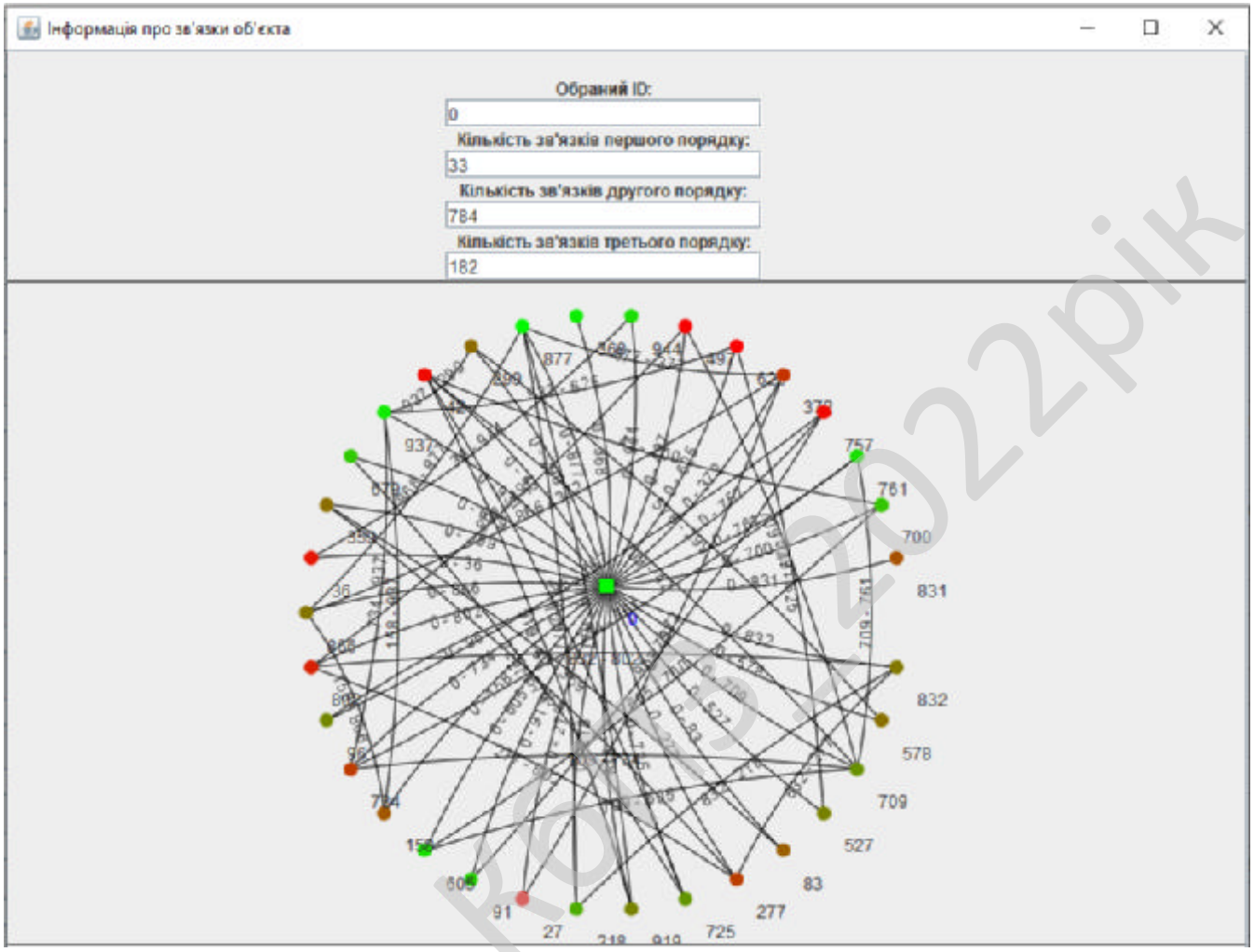

Рисунок 3.13 – Вікно інформації про зв’язки та репутаційний граф

З рисунка 3.13 видно, що кожний об’єкт представлений певною вершиною, яка має колір та унікальний номер. Колір знаходиться в діапазоні від темно-червоного до світло-зеленого і відповідає низькій та високій репутації відповідно. Унікальний номер вершини збігається з унікальним номером об’єкта з комп’ютерної моделі репутаційної системи соціальної мережі.

Вершини між собою зв’язані ребрами, вони відображають зв’язки між об’єктами в системі. Кожне ребро підписане унікальними номерами двох вершин, що воно з’єднує.

Таким чином, розглянувши опис системи, структурну, функціональну схеми системи, та діаграму взаємодії процесів перейдемо до опису блок-схем основної програми, та підпрограм, які використовуються, для реалізації комп’ютерної моделі репутаційної системи соціальної мережі.

4 РЕАЛІЗАЦІЯ ПРОЕКТУ. РОЗРАХУНКИ І ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ПРАВИЛЬНІСТЬ ПРОЕКТНИХ РІШЕНЬ

4.1 Розробка блок–схем та опис алгоритмів функціонування системи

Розглянемо алгоритм роботи програмної реалізації комп'ютерної моделі репутаційної системи соціальної мережі. На рисунку 4.1 зображено блок-схему запуску програмного застосунка.

Рисунок 4.1 – Блок-схема запуску програмного застосунка

З рисунка 4.1 видно, що програма відразу створює графічний інтерфейс та створює всі необхідні об’єкти для подальшої роботи алгоритмів. Після налаштування інтерфейсу, він виводиться на екран і потік, що відповідає за графічний інтерфейс, переходить до внутрішнього циклу, в якому відбувається реагування на різні події, в тому числі, натиски на кнопки користувачем.

Далі слід розглянути алгоритми, що починають своє виконання в той момент, коли користувач самостійно їх запустить. Один з таких алгоритмів зображено на рисунку 4.2, що відображає механізм додавання нових об’єктів до комп’ютерної моделі репутаційної системи соціальної мережі.

Рисунок 4.2 – Блок-схема механізму додавання нових об’єктів

З налаштувань зчитується інформація про кількість нових об’єктів та їх характеристика. Наступний цикл створює вказану кількість об’єктів по черзі з такою внутрішньою інформацією, що відповідає налаштуванням системи і додає їх до внутрішньої бази даних, що зберігає об’єкти в оперативній пам’яті.

Ще один важливий алгоритм – створення нових зв’язків між об’єктами. Цей алгоритм представлено на рисунку 4.3.

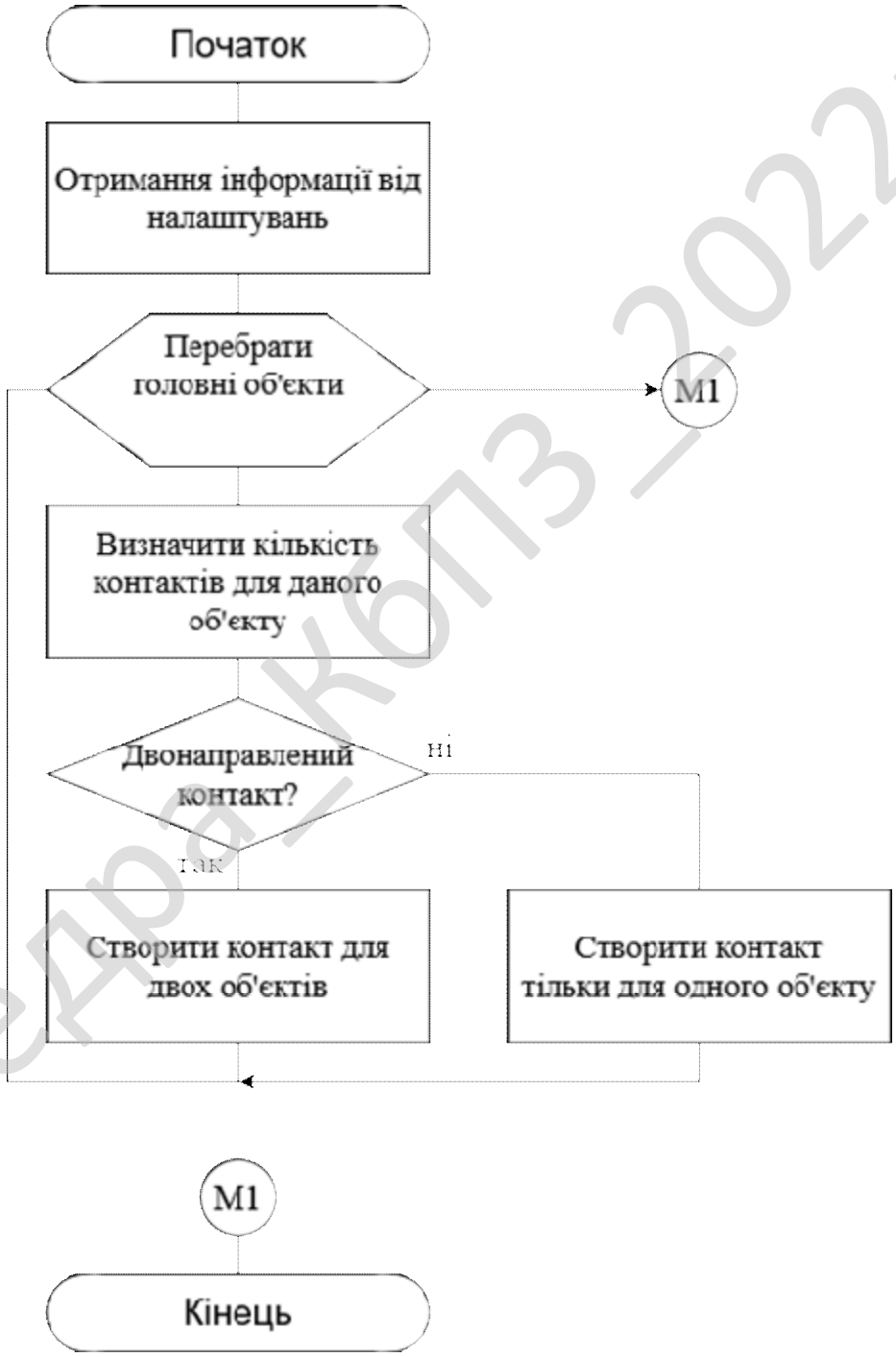

Рисунок 4.3 – Блок-схема створення нових зв’язків між об’єктами

З рисунка 4.3 видно, що алгоритм знову отримує дані налаштувань і потім відбуваються обчислення. В циклі перебираються головні об'єкти – це ті об'єкти, що можуть мати зв'язки з рештою об'єктів. Дане налаштування використовується для моделювання комп'ютерної моделі репутаційної системи соціальної мережі, якщо потрібно дослідити систему з найбільш важливими об'єктами. В їх ролі можуть виступати компанії, зірки або інші об'єкти, що пропонують якісь послуги.

Контакти можуть бути двох типів: одно-направлені та двох-направлені. Зв'язок в одну сторону дозволяє тільки одному об'єкту пам'ятати про здійснений контакт. Відповідно, при подвійному напрямку обидва об'єкти зберігають про це дані.

Наступний цикл буде повторюватися в залежності від кількості контактів, що потрібно додати в мережу. За допомогою алгоритму визначається об'єкт, з яким буде здійснено контакт. Інформація про цю подію записується до обох об'єктів якщо не порушуються правила взаємозв'язків. Час від часу відбувається оновлення популярності – це ресурс затратна процедура, що перебирає всіх об'єктів з якими вже відбулися контакти і заново визначається популярність.

Через те, що в графічному інтерфейсі можна запускати на виконання незалежні функції, деякі з них мають малий об'єм, але виконують важливі завдання, тому не має сенсу розглядати кожну з них. Дані процедури працюють окремо одна від одної та запускаються користувачем різними кнопками в графічному інтерфейсі.

Наступний алгоритм перебору об'єктів для здійснення контактів розділяється на два, адже його в один момент часу може виконувати багато незалежних потоків. Перший головний потік створює допоміжні, дає їм задачі, стежить за тим, щоб не виникали помилки та узагальнює отримані результати. Такий підхід значно збільшує точність отриманих результатів. Користувач може один раз запустити цей комплексний алгоритм і відразу отримати результат, наприклад, десяти одинарних таких повторень. Алгоритми головного та

допоміжних потоків пропоную розглянути окремо. На рисунку 4.4 зображено алгоритм головного потоку перебору об’єктів.

Рисунок 4.4 – Блок-схема головного потоку перебору об’єктів

З попереднього рисунка можна сказати, що алгоритм спочатку зберігає дані до бази даних. З бази даних всі допоміжні потоки будуть брати початкові дані для здійснення розрахунків.

Перебираючи додаткові потоки, основний алгоритм створює допоміжні потоки, налаштовує їх та запускає кожний на виконання. Збереження до списку відбувається щоб їх не втратити.

Далі чекає поки всі вони закінчать виконувати обчислення та збережуть результати до своїх таблиць в базі даних. Після цього почнеться процес узагальнення результатів.

Результат формується за рахунок перебору всіх результатів від допоміжних потоків та обчислення фінального результату, який і буде збережено до основної таблиці з даними. Допоміжні таблиці не видаляються аби користувач мав можливість подивитися результати кожного окремого допоміжного потоку.

Далі розглянемо цей самий алгоритм але з іншої сторони. На рисунку 4.5 показано алгоритм допоміжного потоку здійснення контактів.

Рисунок 4.5 – Блок-схема допоміжного потоку здійснення контактів

Алгоритм з рисунка 4.5 працює майже так само, як і алгоритм створення контактів, що було розглянуто раніше. Але в цьому випадку дані беруться з бази даних. Цей алгоритм повинен бути достатньо простим, адже він в один момент часу виконується багатьма потоками, що може значно ускладнити пошук помилки, якщо щось піде не так.

Для кращого розуміння алгоритму створення контактів між об’єктами, варто переглянути алгоритм зі сторони об’єкта. Це зображено на рисунку 4.6.

Рисунок 4.6 – Блок-схема алгоритму створення контактів зі сторони об’єкта

В першу чергу, до об’єкта звернулись і він тепер більш популярний, тому це значення збільшується. В процесі взаємодії між об’єктами відбувається визначення якості їх взаємодії. Вона прямим чином впливає на репутацію об’єкта і залежить від можливості об’єкта здійснювати якісні контакти.

Якщо контакт між двома об’єктами вже відбувався, потрібно перерахувати враження про об’єкт до якого звернулись. Інакше, сформувати перше враження.

Збільшуються значення які дозволяють значно оптимізувати роботу комп’ютерної моделі репутаційної системи соціальної мережі за рахунок присутності додаткових полів даних в об’єкті. Ці поля видаляють потребу дуже часто перераховувати репутацію та ряд інших значень для кожного об’єкту моделі. Щоб це зробити без них, складність алгоритму від одиничного переходила б до квадратичного. Тому варто не забувати стежити за оптимізацією програмних засобів.

Збір статистики про роботу моделі відбувається в різних ділянках алгоритму і вони знаходяться в різних модулях моделі. Через це неможливо якимось чином даний алгоритм об’єднати до однієї блок-схеми.

А от алгоритм виведення інформації про обраний об’єкт є більш визначеним і його можна побачити на рисунку 4.7.

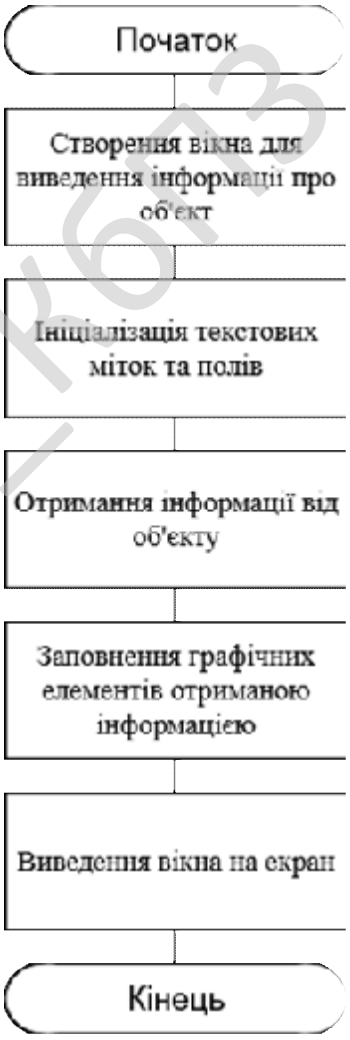

Рисунок 4.7 – Блок-схема алгоритму виведення інформації про обраний об’єкт

На рисунку 4.8 зображено схожий алгоритм виведення інформації про всі об’єкти комп’ютерної моделі репутаційної системи соціальної мережі.

Рисунок 4.8 – Блок-схема алгоритму виведення інформації про всі об’єкти

Основна відмінність алгоритму виведення інформації про всі об’єкти від попереднього алгоритму для одного об’єкту полягає в тому, що тут використовується цикл для перебору багатьох об’єктів та інформація додається динамічно.

На черзі алгоритм визначення зв’язків об’єкта по їх порядках. На першому рівні знаходяться об’єкти, що напряму мають зв’язок з обраним об’єктом. Да другому рівні об’єкти з одним проміжним об’єктом. І на третьому, відповідно, об’єкти з двома проміжними. Даний алгоритм використовується для формування та виведення на екран репутаційного графу. Розглянемо його детальніше на рисунку 4.9, після чого подивимось як саме формується репутаційний граф.

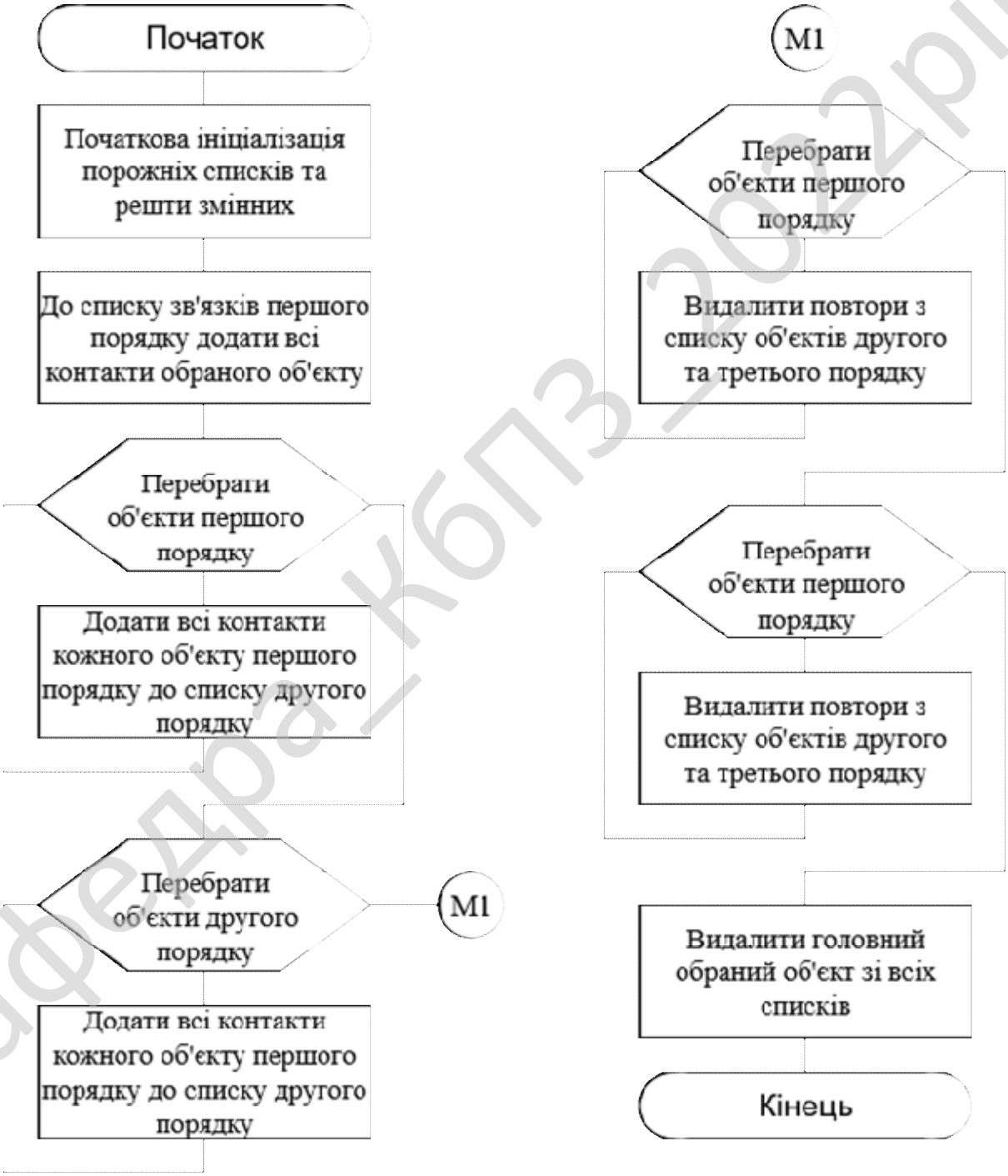

Рисунок 4.9 – Блок-схема алгоритму визначення зв’язків об’єкта по їх порядках

Результат алгоритму визначення зв’язків об’єкта по їх порядку є три списки. Перший список є найменшим, в ньому зберігаються лише ті об’єкти, зв’язки з якими здійснені напряму від головного об’єкту. Останній список, зазвичай, є найбільшим, адже в ньому зберігаються об’єкти що пов’язані з головним за допомогою двох проміжних об’єктів.

Приклад виконання даного алгоритму зображено на рисунку 4.10.

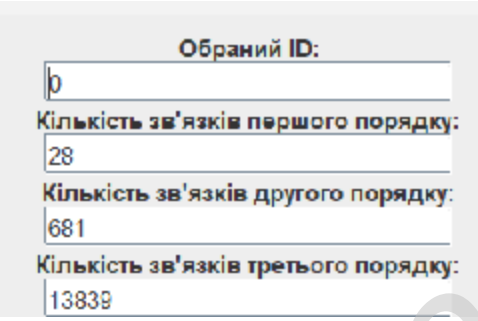

Рисунок 4.10 – Приклад виконання алгоритму визначення зв’язків об’єкту

У комп’ютерній моделі репутаційної системи соціальної мережі приймало участь 50000 об’єктів для формування такої статистики. За основу було взято об’єкт з унікальним номером 0. Між цими 50000 об’єктами сформовано 1198298 зв’язків. З головним об’єктом лише 28 сформували зв’язок. 681 об’єкт на шляху до головного об’єкту мали один проміжний. А 13839 об’єктів вже мали два проміжні об’єкти.

Завдяки алгоритму по пошуку зв’язків було легше реалізувати репутаційний граф. На рисунку 4.11 зображено алгоритм формування репутаційного графу без врахування попереднього алгоритму.

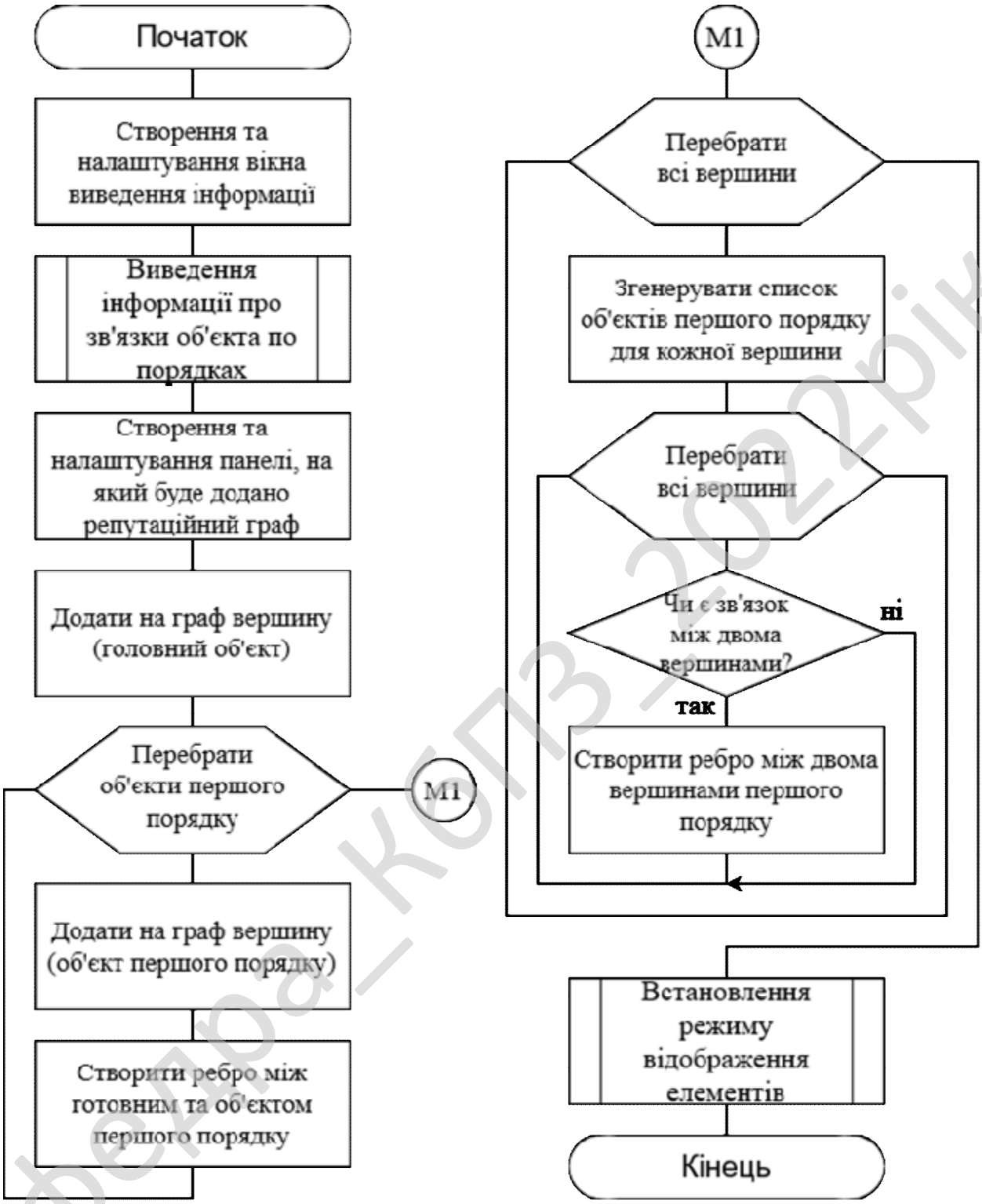

Рисунок 4.11 – Блок-схема алгоритму формування репутаційного графу

Як і для виведення зв’язків об’єкта по порядках, все починається зі створення графічного вікна. На ньому відразу виводяться інформація про зв’язки. А от нижче додається спеціальна панель, на яку і буде виводитися граф.

Від початку в граф додається початковий об’єкт, той, що обрав користував і для якого виводиться інформація про зв’язки. За допомогою алгоритму знаходження зв’язків першого порядку, відбувається їх подальший перебір для створення відповідних вершин і додання їх до репутаційного графу. Для кожної нової вершини створюється ребро з основним об’єктом, адже нова вершина є зв’язком першого порядку.

Механізм утворення ребер між вершинами першого порядку більш складний. Для цього необхідно перебирати відразу дві вершини, перевіряти що вони не посилаються на одну і ту саму вершину. Жодна з двох обраних вершин не повинна бути головною, адже з нею ребра вже готові. Тільки тепер можна створювати ребро між вершинами, що перебираються.

Пропоную переглянути приклад репутаційного графу на рисунку 4.12.

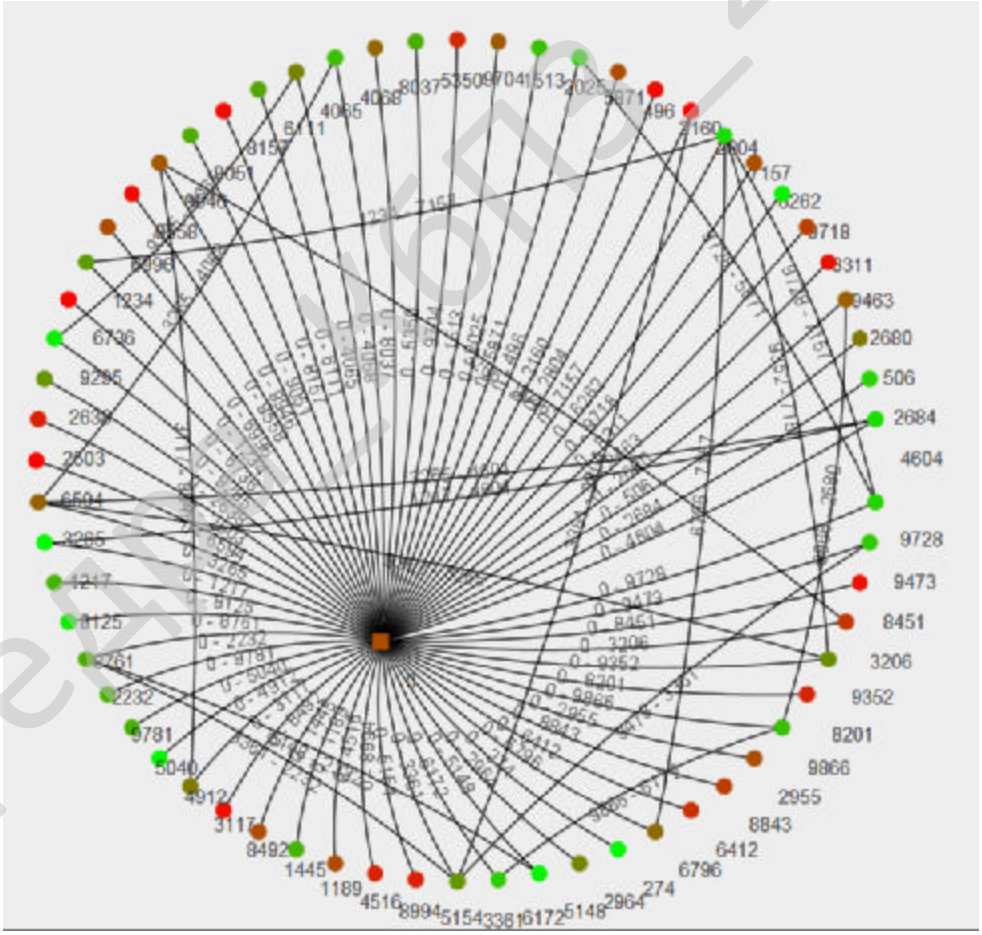

Рисунок 4.12 – Приклад репутаційного графу

Репутаційний граф з попереднього малюнку був утворений в результаті виконання комп'ютерної моделі репутаційної системи соціальної мережі, в якій було 10000 об'єктів та 686662 зв'язків.

В залежності від різних наборів початкових параметрів, система видає різні результати, що відповідають очікуваним результатам. Також були протестовані всі налаштування даного застосунку та наглядно показана зручність розробленого графічного інтерфейсу.

4.2 Захист розробленого програмного забезпечення

Як правило, ліцензія на програмне забезпечення дозволяє одержувачеві використовувати одну або декілька копій програми, причому без ліцензії таке використання розглядалося б в рамках закону як порушення авторських прав видавця. По суті, ліцензія виступає гарантією того, що видавець ПЗ, якому належать виключні права на програму, не подасть у суд на того, хто нею користується.

Ліцензії на програмне забезпечення, в цілому, діляться на дві великі групи: невільні (власницькі, вони ж пропрієтарні; і напіввільні) і ліцензії вільного і відкритого ПЗ. Їх відмінності сильно впливають на права кінцевого користувача щодо використання програми.

Основною характеристикою пропрієтарних ліцензій є те, що видавець ПЗ в ліцензії дає дозвіл її одержувачу використовувати одну або кілька копій програми, але при цьому сам залишається правовласником всіх цих копій. Один з наслідків такого підходу полягає в тому, що практично всі права на ПЗ залишаються за видавцем, а користувачеві передається лише дуже обмежений набір чітко окреслених прав. Для пропрієтарних ліцензій типово перерахування великої кількості умов, що забороняють певні варіанти використання ПЗ, навіть тих, які без цієї заборони були б дозволені законом про авторське право.

Найбільш значним наслідком застосування проприетарной ліцензії є те, що кінцевий користувач зобов'язаний прийняти її, оскільки згідно із законом власником ПО є не він, а видавець програми. У разі відмови прийняти ліцензію користувач взагалі не може працювати з програмою.

На відміну від пропрієтарних, вільні і відкриті ліцензії не залишають права на конкретну копію програми її видавцеві, а передають найважливіші з них кінцевому користувачеві, який і стає власником. В результаті користувач за замовчуванням отримує важливі права, які закон про авторське право за замовчуванням дає тільки власнику копії, проте всі авторські права на ПЗ як і раніше залишаються у видавця.

Головною відмінною рисою вільних ліцензій є те, що вони абсолютно не обмежують особисте користування – користувач може приймати чи не приймати їх: працювати з програмою він може і без ліцензії. Однак якщо йому потрібно будь-яка з додаткових прав, які дає ліцензія – він зобов'язаний прийняти ліцензію і діяти в її рамках.

Вихідний код програмного забезпечення з відкритим вихідним кодом доступний для перегляду, вивчення та зміни, що дозволяє переконатися у відсутності вразливостей і неприйнятних для користувача функцій, взяти участь в доопрацюванні відкритої програми, використовувати код для створення нових програм і виправлення в них помилок – через запозичення вихідного коду, якщо це дозволяє сумісність ліцензій, або через вивчення використаних алгоритмів, структур даних, технологій, методик і інтерфейсів.

Різниця між рухами відкритого програмного забезпечення і вільного програмного забезпечення полягає, в основному, в пріоритетах. Прихильники терміна "open source" роблять акцент на ефективності відкритих початкових кодів як методу розробки, модернізації та супроводу програм. Прихильники терміна "free software" вважають, що саме права людини на вільне поширення, модифікацію і вивчення використовуваних ним програм є головною перевагою вільного відкритого ПЗ.

Узагальнивши все вищесказане, було прийнято рішення, що дана система дослідження та програмна реалізація комп'ютерної моделі репутаційної системи соціальної мережі буде розповсюджуватися по вільній відкритій ліцензії. Це значить, що кожний охочий зможе переглянути вихідний код розроблених застосунків та внести до них зміни; матиме можливість вільно розповсюджувати дане програмне забезпечення. Це надасть змогу якомога більшій кількості людей вільно отримати доступ до даної системи для власного використання, а також для вдосконалення їх кожним бажаючим.

Весь вихідний код розробленого програмного забезпечення та додаткові матеріали, як, наприклад, база даних та зображення, буде розміщено на GitHub. Цей сервіс є найбільшим веб-сервісом для хостингу IT-проектів та їхньої спільної розробки. Сервіс безкоштовний для проектів з відкритим вихідним кодом, з наданням користувачам усіх своїх можливостей. Окрім розміщення коду, учасники можуть спілкуватись, коментувати редагування один одного, а також слідкувати за новинами знайомих. За допомогою широких можливостей Git програмісти можуть поєднувати свої репозиторії – GitHub дає зручний інтерфейс для цього і може показувати вклад кожного учасника в вигляді дерева.

5 ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

Для того, щоб інтегрувати компоненти розробленого програмного забезпечення в існуючу апаратну систему, необхідно виконати декілька кроків:

а) для початку потрібно встановити на персональний комп’ютер Java SE Development Kit 19 або більш нову версію з офіційної сторінки компанії Oracle;

б) після того, як Java завершить процес завантаження і установки, відкрийте "Windows PowerShell", використовуючи меню "Пуск". Після відкриття PowerShell введіть "java -version", щоб переконатися, що встановлення на комп'ютер завершилось успішно;

в) якщо версію Java не виявлено на комп’ютері, потрібно додати нову змінну в системне середовище Path, з тою умовою, що налаштування відбуваються в Windows. Для цього потрібно в параметрах системи перейти до "додаткові параметри системи" та натиснути на "змінні середовища". У новому вікні в полі змінних середовища користувача необхідно знайти змінну Path та змінити її значення. А саме, необхідно додати новий шлях до папки "bin", яка знаходиться в головній папці встановленого програмного забезпечення (Java SE Development Kit 19, або інше).

г) тепер можна запустити розроблене програмне забезпечення одним із нижче наведених способів:

1) перейдіть в каталог з застосунком та натисніть на нього подвійним кліком лівої кнопки миші, або натисніть правою кнопкою миші по файлу та в меню "Відкрити за допомогою" оберіть "Java";

2) відкрийте Windows PowerShell та введіть команду: "java -jar $", замінивши знак долара на повний шлях до потрібного файлу.

Якщо є можливість скомпілювати програмний додаток з вихідних файлів, які мають розширення java, варто розглянути структуру проекту більш детально.

Файл "Main.java" в директорії розробленого програмного забезпечення відповідає за запуск цього програмного забезпечення. Файл "setting.txt" – зберігає

налаштування додатку. А файл "*.db" відповідає за збережену в процесі виконання програмного забезпечення інформацію, його назва налаштовується під час роботи додатку. Два останні файли спочатку можуть бути відсутні, але після першого запуску вони обов’язково з’являться.

Файли з розширенням form містять дані в форматі XML. У них міститься вся необхідна інформація для побудови графічного інтерфейсу користувача. В папці з проектом також містяться файли з такою самою назвою, але з розширенням java. В них містяться алгоритми роботи вікна та обробки подій.

Файл "Model.java" зберігає основні алгоритми, що відносяться саме до моделювання процесів репутаційної системи соціальної мережі. Файл "User.java" описує один об’єкт моделювання, а файли "Statistics.java", "Settings.java" та "LinksList.java" містять алгоритми збирання статистики по моделі, її налаштування та формування списку зв’язків відповідно.

В файлі "SQLite.java" прописані алгоритми пов’язані з роботою із базами даних. І нарешті файл "DrawGraph.java" відповідає за формування графіку на графічному інтерфейсі користувача.

Програма має простий та інтуїтивно зрозумілий інтерфейс, який зображений на рисунку 5.1. Хоч розмір розробленого інтерфейсу і можна змінювати як завгодно, зручніше його розкривати на повний екран. На рисунку 5.1 його навмисно звужено.

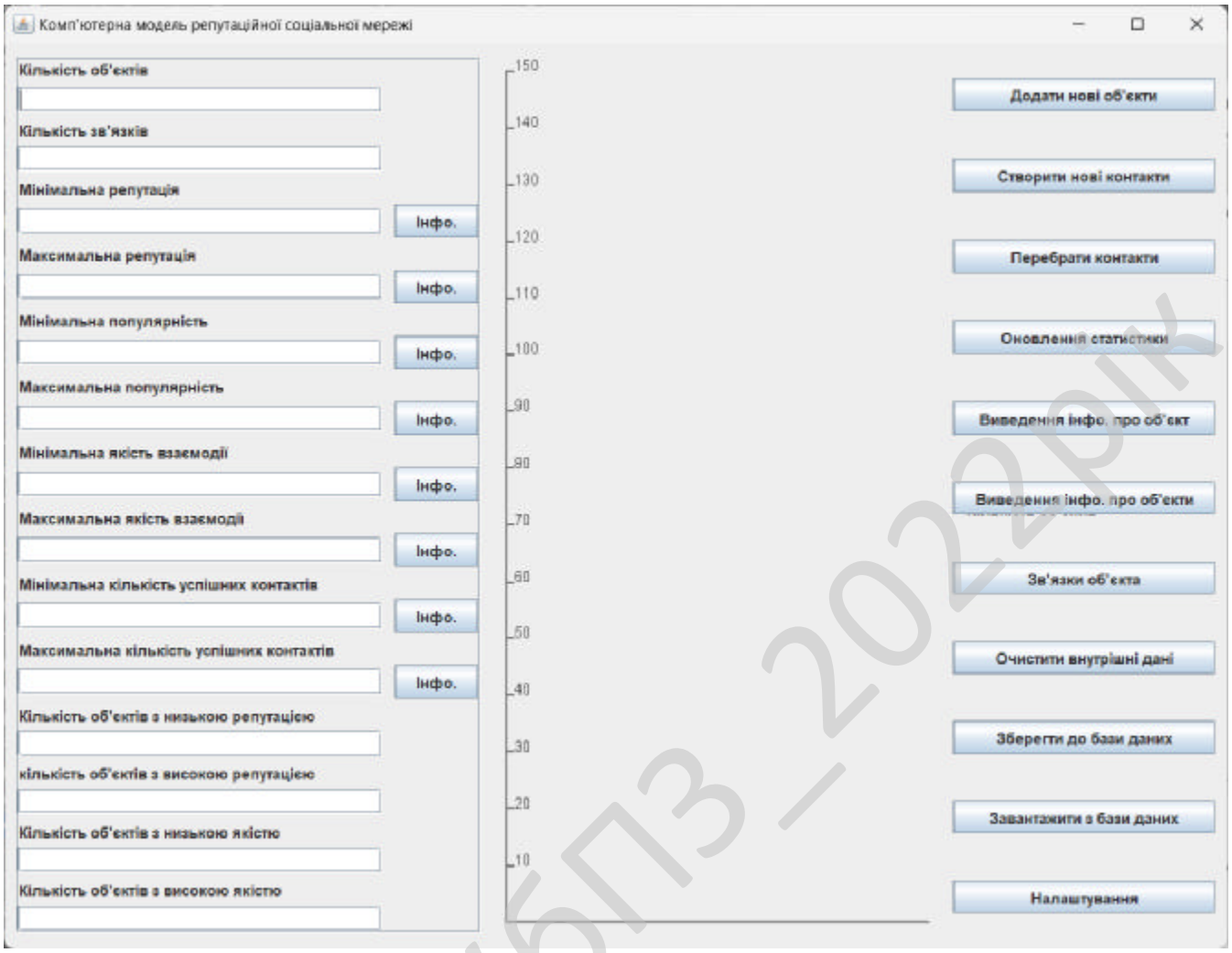

Рисунок 5.1 – Інтерфейс застосунка "Комп’ютерна модель репутаційної системи соціальної мережі"

Для кращого розуміння можливостей налаштування комп’ютерної моделі репутаційної системи соціальної мережі, на рисунку 5.2 також зображено вікно налаштувань.

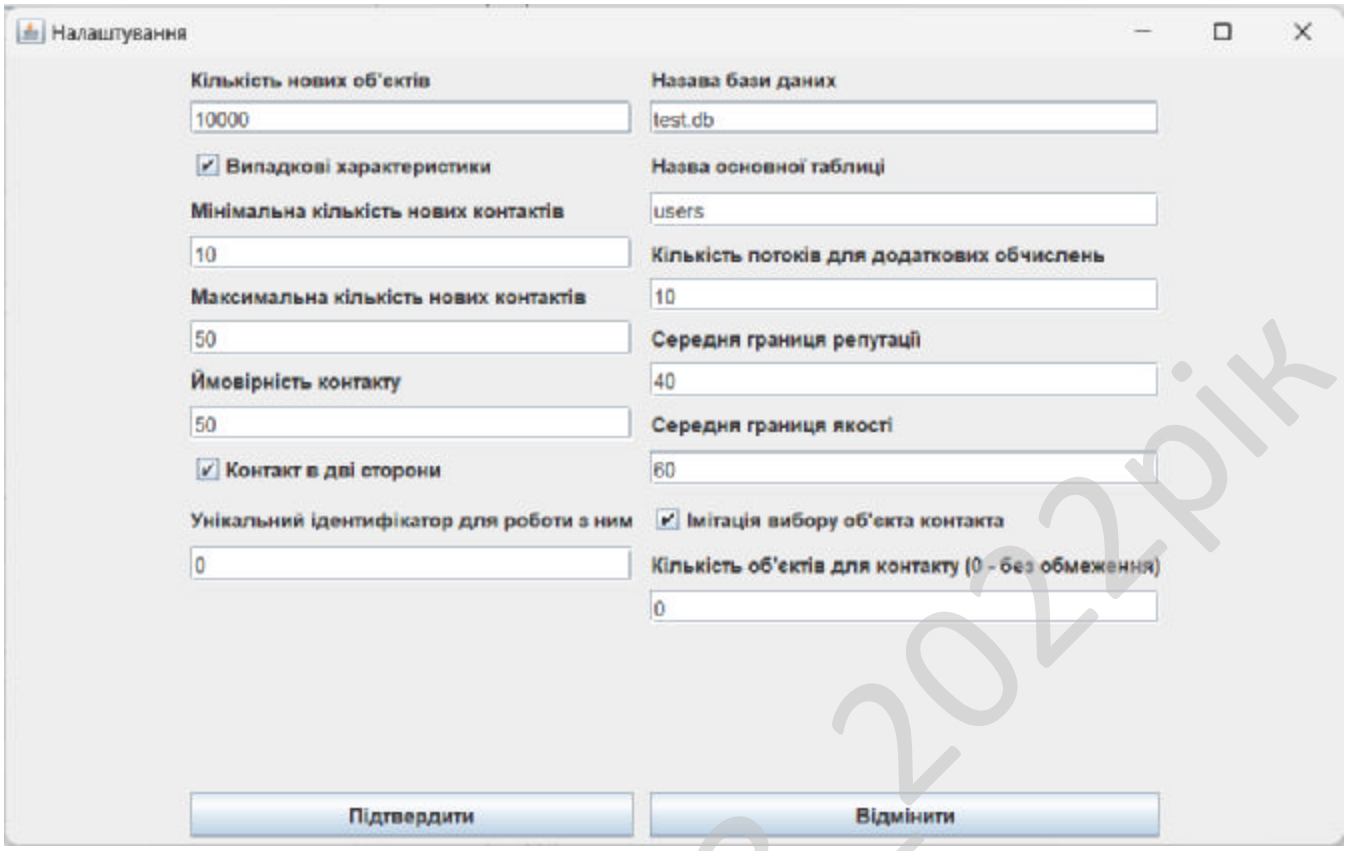

Рисунок 5.2 – Інтерфейс вікна налаштувань застосунка "Комп’ютерна модель репутаційної системи соціальної мережі"

6 НАУКОВА НОВИЗНА

Метою роботи є дослідження та програмна реалізація комп'ютерної моделі репутаційної системи соціальної мережі.

Об'єктом дослідження є процес симуляції репутаційної системи соціальної мережі в комп'ютерних моделях.

Предметом дослідження є методи та алгоритми комп'ютерного моделювання репутаційної системи соціальної мережі.

Методи дослідження базуються на теорії об'єктно-орієнтованого програмування, теорії алгоритмів, методах комп'ютерного моделювання, а також теорії розповсюдження інформації.

Наукова новизна отриманих результатів. У процесі рішення завдань, обумовлених цілями дослідження, отримані наступні результати:

1. Розроблено метод комп'ютерного моделювання репутаційних систем соціальних мереж, який відрізняється від існуючих параметризацією об'єктів мережі, що дозволяє слідкувати за тим, як різні чинники впливають на репутацію однакових за характеристиками об'єктів.

2. Розроблено вітчизняний продукт комп'ютерного моделювання репутаційних систем соціальних мереж, який має більш широкі можливості, на відміну від існуючих аналогів.

У цій роботі розроблено комп'ютерну модель репутаційної системи соціальної мережі, в якій можна моделювати два типи зв'язків: рівноцінні та односторонні. Перші зв'язки виникають в системах з об'єктами однакового рівня, наприклад, в реальному житті між людьми, на онлайн платформах між її відвідувачами та інше. Другі зв'язки виникають в системах з найбільш популярними об'єктами, це може бути якийсь бізнес та його клієнти, відомі люди та фанати і тому подібне.

Змодельовано репутаційну систему соціальної мережі з заданими початковими характеристиками. Моделювання можна перезапускати з різними

початковими даними та отримувати нові дані для дослідження та виявлення в них закономірностей.

Розраховане співвідношення репутації об'єктів комп'ютерної моделі репутаційної системи соціальної мережі до їх популярності та якості взаємодії з ними. В результаті чого підтверджено, що ці три чинники мають найбільший вплив один на одного.

Виявлено закономірності успішних взаємозв'язків між об'єктами моделювання репутаційної системи соціальної мережі. Репутація залежить від багатьох факторів, більшість із яких було розглянуто в даній роботі.

Визначено межі застосування для моделей. Межі застосування виявлені на основі обмежень і поверхневої вартісної оцінки для подолання цих обмежень. Таким чином, обмеження проходить по лінії рівності вартості подолання та фінансових можливостей компанії, що проводить дослідження або впровадження системи.

До напрямів майбутніх досліджень у галузі можна віднести, передусім, опис практичних кейсів впровадження моделей. Опис практичних кейсів може переслідувати кілька цілей, наприклад, виявлення нових чинників впливу та обмежень або уточнення меж застосування. Крім того, пропонується розширювати, доповнювати, змінювати комп'ютерну модель, додаючи нові модулі.

За результатами дослідження можна зробити такі висновки:

• Більшість існуючих репутаційних моделей найкраще застосовуються всередині репутаційних систем. Поза цими системами немає встановленого механізму оцінювання об'єктами одне одного, що призводить до високої варіативності оцінок і збільшує витрати ресурсів з їхнього вилучення.

• Деякі моделі можуть бути застосовані лише для компаній на ринках з невеликою кількістю продавців – через складність обчислень, що зростає, при зростанні кількості конкурентів.

7 ДАНІ ПРО ЕКОНОМІЧНУ ЕФЕКТИВНІСТЬ РОЗРОБЛЕНОЇ ПРОГРАМИ

7.1 Техніко-економічне обґрунтування теми дипломної роботи

Після ознайомлення з підприємством та засобами розробки програмної продукції був розроблений план розробки програми. Був підрахований необхідний час для розробки та впровадження програми. Цей час склав 60 днів (три місяці).

В магістерській роботі було проведене дослідження та виконана програмна реалізація комп'ютерної моделі репутаційної системи системи соціальної мережі.

Розроблене програмне забезпечення має достатню надійність і задовольняє усім поставленим умовам, а саме:

а) невеликий розмір;

б) невеликі системні потреби;

в) незалежність від встановлених на комп'ютері баз даних;

г) зручність у користуванні та надійність.

Таблиця 7.1 – Початкові дані

Показники	Позначення	Характеристика або величина
1	2	3
1. Кількість розроблених програм період, шт.	N	1
2. Кількість екземплярів програм, шт.	Ne	22
3. Запланований термін розробки, днів	Fpq	60 (3 місяці)
4. Група задачі підсистеми управління (1-6)	–	1
5. Ступінь новизни задачі (А, Б, В, Г)	–	Б
6. Складність алгоритму (1, 2, 3)	–	2

Продовження таблиці 7.1

1	2	3
7. Кількість макетів вхідної інформації	–	3
8. Кількість форм вихідної інформації.	–	4
9. Мова програмування (1-6)	–	3
10.Попередній досвід (1-6)	–	3
11.Гнучкість проекту ПП (1-6)	–	3
12.Детальність проекту ПП (1-6)	–	2
13.Рівень спрацьованості колективу (1-6)	–	2
14.Ступінь вимірності процесів (1-6)	–	3
15.Необхідна надійність програмного забезпечення (1-6)	–	2
16.Розмір бази даних (порівняно з розміром програми) (1-6)	–	2
17.Складність кінцевого програмного продукту (1-6)	–	2
18.Необхідний рівень забезпечення повторного використання (1-6)	–	2
19.Документованість відповідно до планованого життєвого циклу (1-6)	–	2
20.Вимоги до швидкодії ПП (1-6)	–	2
21.Обмеження на розміри основного сховища даних (1-6)	–	2
22.Різноманітність використовуваних обчислювальних платформ (1-6)	–	2
23.Професійний рівень аналітиків (1-6)	–	2
24.Професійний рівень програмістів (1-6)	–	2
25.Постійність складу команди розробників (1-6)	–	2
26.Досвід розробки додатків (1-6)	–	2
27.Досвід роботи з обчислювальною платформою (1-6)	–	2

Продовження таблиці 7.1

1	2	3
28.Досвід роботи з мовою і інструментами середовища розробки (1-6)	–	2
29.Досвід роботи з програмними інструментами розробки (1-6)	–	3
30.Розробка ПЗ для декількох серверів одночасно (1-6)	–	2
31.Вимоги до дотримання встановленого графіка робіт (1-6)	–	2
32.Вартість ПЗ у розробника (НМА), грн.	–	22000
33.Норматив додаткової зарплати, % :	$Н_д$	10
34.Норматив відрахувань у соціальні фонди, %	Нс	22
35.Норматив загальногосподарських витрат, %	Нг	15
36.Норматив витрат на освоєння нових мов програмування, %	Нп	15
37.Рівень рентабельності програмної продукції, %	Ре	50
38.Ставка податку на додану вартість, %	Ндв	20

7.2 Розрахунок трудомісткості розробки програмної продукції

Трудомісткість розробки ПЗ для стадій ТЗ, ЕК, ТП та ВП визначаємо по типовим нормам часу приведеним в додатках МВ. Стадія РП є найбільш тривалою і трудомісткою, що значно впливає на інші стадії проекту.

Визначимо трудомісткість розробки ПЗ для стадії РП.

Обчислюємо номінальні трудовитрати, люд-міс.:

$$T_{ном} = A \, \text{Size}^B, \tag{7.1}$$

де: A – коефіцієнт Боема, A = 2,45;

Size – загальний об'єм відлагодженого програмного коду, тис. рядків;

В – показник ступеня, що визначається співвідношенням:

$$B = 1{,}01 + 0{,}001\sum W_i, \qquad (7.2)$$

де: *Wi* – сумарне значення п'яти показників (МВ, додаток 2), що відображають особливості розробки проекту програмного продукту (ПП) і колективу розробників.

$$B = 1{,}01+0{,}001(2{,}43+3{,}64+3{,}38+3{,}95+2{,}73) = 1{,}027.$$

$$T_{ном} = 2{,}45 \cdot 2{,}7^{1{,}026} = 6{,}78 \text{ люд-міс.}$$

Визначаємо уточнені (з урахуванням приведених в МВ додатку 3 сімнадцяти додаткових коефіцієнтів) трудовитрати, люд-міс.:

$$T_{уточн} = T_{ном}\, ПВ_j, \qquad (7.3)$$

де: $ПВ_j$ – добуток сімнадцяти додаткових коефіцієнтів, приведених в МВ додатку 3.

$$T_{уточн} = 6{,}78 \cdot (0{,}88 \cdot 0{,}93 \cdot 0{,}88 \cdot 0{,}91 \cdot 0{,}95 \cdot 1 \cdot 1 \cdot 0{,}87 \cdot 1{,}22 \cdot 1{,}16 \cdot 1{,}1 \cdot 1{,}1 \cdot$$
$$\cdot 1{,}12 \cdot 1{,}1 \cdot 1 \cdot 1{,}1 \cdot 1{,}1) = 9{,}37 \text{ люд-міс.}$$

Ці коефіцієнти дозволяють диференційовано оцінювати результати роботи програмістів, беручи до уваги швидкодію програми, використання різноманітних обчислювальних платформ і інструментів розробки, взаємодію декількох серверів, вимоги до об'ємів баз даних і ін.

Визначаємо підсумкові трудовитрати по стадії робочий проект, люд-дні:

$$T_{РП} = 0{,}3 C T_{уточн}^{0{,}33+0{,}2(B-1{,}01)} S, \qquad (7.4)$$

де: *C* – визначений емпірично коефіцієнт, запропонований авторами методики, (МВ, додаток 4);

S – коефіцієнт стиснення (або подовження) графіка робіт %, що дозволяє коректувати терміни розробки ПЗ згідно встановленим вимогам. Вибираємо в межах (25…350)%.

$$T_{РП} = 0{,}3 \cdot 2{,}85 \cdot 9{,}37^{0{,}33+0{,}2(1{,}026-1{,}01)} \cdot 65 = 117 \text{ люд/день.}$$

Для зручності визначення загальної трудомісткості на розробку програмного забезпечення результати розрахунків по стадіям зводимо до таблиці 7.2.

Таблиця 7.2 – Визначення трудомісткості розробки програмного забезпечення

Стадії розробки	Трудомісткість за типовими нормами та розрахунками	
	Величина, люд/дні	Підстава
Технічне завдання	9	Д5
Ескізний проект	10	Д6
Технічний проект	9	Д7
Робочий проект	117	Ф 7.1-7.4
Впровадження	13	Д13
Всього	158	–

7.3 Визначення чисельності виконавців і планового фонду зарплати

Чисельність ставок інженерів-програмістів для розробки програмного забезпечення визначається за формулою:

$$Ч = \frac{T_{пз} N}{F_{рq} - H_{ев}}, \quad (7.5)$$

де: $F_{рq}$ – плановий фонд робочого часу одного спеціаліста, днів;

$T_{пз}$ – трудомісткість розробки програмного забезпечення люд-дні.

$$Ч = \frac{158 \cdot 1}{60 - 5} = 2,8 \text{ ставки.}$$

Чисельність інженерів-електронщиків для проведення технічного обслуговування та ремонту комп'ютерних мереж визначається в залежності від наявності технічних засобів і норм витрат часу на виконання профілактичних робіт на протязі року.

Визначаємо затрати часу на виконання профілактичних робіт по обслуговуванню обладнання за період розробки. Результати розрахунку зводимо до таблиці 7.3.

Таблиця 7.3 – Затрати часу на виконання профілактичних робіт по обслуговуванню обладнання за розрахунковий період

Найменування обладнання	Профілактичне обслуговування			
	Кількість хв. на один. обл.	Кількість облад-нання	Затрати часу в хв.	Затрати часу в год.
Системний блок ПК	90	10	900	15
Монітор	60	10	600	10
Клавіатура	30	10	300	5
Маніпулятор «мишка»	30	10	300	5
Принтер матричний	60	0	0	0,0
Принтер лазерний	120	1	120	2
Принтер струминний	60	1	60	1
Сканер	20	1	20	0,33
Концентратор–маршрутизатор	30	2	60	1
Кабельні господарства ЛВС на 1 м. п.	2,5	280	700	11,67
Копіювальний апарат	140	1	140	2,33
Усього за рік:			$З_ч$	53,33

Час на профілактику обладнання в загальному балансі робочого часу інженерів-електронщиків неповинен складати більше 10%.

Виходячи з цього фонд робочого часу інженерів-електронщиків складає:

$$Ф_{др}^{с} = \frac{З_ч \cdot n_{міс}}{1,2}, \quad (7.6)$$

$$Ф_{др}^{с} = \frac{53 \cdot 3}{1,2} = 132,5 \text{ год.}$$

Визначаємо необхідну кількість ставок штатного персоналу сектора ТО:

$$Ч_{ел} = \frac{Ф^{с}_{др}}{F_{др} \cdot T_{зм}}, \quad (7.7)$$

$Ч_{ел}$ = 132,5/(60·8) = 0,27 ставки.

Для забезпечення нормального технічного обслуговування засобів ТО та мереж, необхідно прийняти найбільше ціле значення розрахункової чисельності інженерів-електронщиків.

Чисельність інженерів-системотехніків, адміністраторів мережі, дизайнерів WEB вузлів, системних програмістів (аналітиків), бухгалтерів-економістів визначається за потребою в залежності від функціональних обов'язків. Після визначення чисельності персоналу складається штатний розклад.

Таблиця 7.4 – Розрахунок чисельності штатного персоналу сектору системного та адміністративного обслуговування засобів ОТ та комп'ютерних мереж

Посада	Вид роботи	Час	К-ть штатних одиниць
Адміністратор загальної мережі, аналітик	Адміністрування локальної мережі, поштового та серверу DNS (ОС FreeBSD), маршрутизатора Cisco, доменного контролеру Windows Server 2012 R2, серверу доступу ADSL (ОС Linux), налаштування ADSL, VPN, PPPoE, Frame Relay, Wi-Fi	2	0,5
	Налаштування і конфігурування базової станції безпровідного зв'язку (CMTS)	0,5	
	Розробка та впровадження проектів з організації зв'язку між віддаленими об'єктами, ЛОМ	0,5	
	Забезпечення цілодобової роботи зв'язку клієнтів до мережі Інтернет	1	
Всього		4	

Продовження таблиці 7.4

Посада	Вид роботи	Час	К-ть штатних одиниць
Продакт-менеджер	Презентації нової продукції, пошук каналів збуту	1	0,25
	Підтримка постійних клієнтів	0,5	
	Оформлення договорів, ведення тендерів	0,25	
	Контроль взаєморозрахунків з постачальниками	0,25	
Всього		2	
Дизайнер WEB	Розробка концепції оформлення та інтерфейсу сайту, оптимізація дизайну існуючих, проектує їх структуру та навігацію	1	0,25
	Створення графічних і стилістичних елементів сайту	0,5	
	Оформлення банерів і промо-сторінок	0,25	
	Розміщення графіки і контенту на Інтернет сторінках	0,25	
Всього		2	
Інженер верстальник	Розробка та верстка макетів рекламної продукції та технічної документації	1	0,25
	Верстка друкованих видань	0,5	
	Додрукова підготовка макетів	0,25	
	Розміщення графіки і контенту на Інтернет сторінках	0,25	
Всього		2	

Складемо штатний розклад виконавців.

Таблиця 7.5 – Штатний розклад виконавців

Посада	Кількість ставок	Середньомісячний оклад, грн.	Всього за період розробки, грн.
Керівник (ІТ-менеджер)	1	18132	54396
Продакт-менеджер	0,25	14000	10500
Інженер-програміст	2,8	18000	151200
Інженер-електронщик	0,27	14000	11340
Інженер-системотехнік	0,25	14000	10500
Адміністратор мережі	0,5	12000	18000
Системний програміст	0,25	11500	8625
Дизайнер WEB	0,25	12000	9000
Інженер-верстальник	0,25	11700	8775
Бухгалтер-економіст	0,5	12500	18750
Всього за період розробки	$R_{сп}$ = 6,32	-	$Ф_{роб}$ = 301086

Розрахуємо середньоденну зарплату одного виконавця:

$$З_{сд} = \frac{Ф_{роб}}{R_{сп} F_{рq}}, \qquad (7.8)$$

де: $Ф_{роб}$ – загальна сума зарплати за плановий період, грн.

$$З_{сд} = \frac{301086}{6,32 \cdot 60} = 794 \; грн.$$

7.4 Розрахунок капітальних вкладень та амортизаційних відрахувань у розробника

Балансова вартість будівель визначається з урахуванням кількості робочих місць виконавців, питомої площі на одне робоче місце, та вартості одного квадратного метра виробничої площі:

$$Б_{уд} = R^1_{сп} S_у Ц_{пл}, \quad (7.9)$$

де: $R^1_{сп}$ – кількість робочих місць виконавців, шт. Приймаємо 8 робочих місць;

$S_у$ – питома площа на одне робоче місце, $м^2$;

$Ц_{пл}$ – вартість одного квадратного метра площі, грн.

Згідно даних інтернет ресурсу DOM.RIA (https://dom.ria.com) ціна одного квадратного метра площі, вік якої не перевищує 30 років, по місту складає 500…1600 $у.о./м^2$. Враховуючи, що курс складає 1 у.о. = 38 грн. приймаємо для розрахунку вартість одного метра квадратного рівною 20000 грн./м^2. На кожне робоче місце у середньому потрібно 8 $м^2$. З урахуванням цього:

$$Б_{уд} = 8 \cdot 8 \cdot 20000 = 1280000 \text{ грн.}$$

Вартість передавальних пристроїв складає 10% від вартості будівель, і у даному випадку вона складе: 128000 грн.

Балансова вартість інвентарю розраховується за нормою 3500 грн. на одне робоче місце. Тобто:

$$І_{нв} = R^1_{сп} \cdot Ц_м, \quad (7.10)$$

де: $Ц_м$ – ціна меблів для одного робочого місця, грн.

$$І_{нв} = 8 \cdot 3500 = 28000 \text{ грн.}$$

Балансова вартість обчислювальної техніки визначається по оптовим цінам постачальника з врахуванням витрат на транспортування.

Специфікація на обчислювальну техніку наведена в таблиці 7.7.

Дані по оптовій ціні на обладнання та комплектуючі вибирались за пропозицією інтернет ресурсу hotline за 26.10.22 – джерело https://hotline.ua

Таблиця 7.6 – Специфікація

Найменування комплектуючої або обладнання	Тип	Оптова ціна
Персональний комп’ютер		11186
Системний блок		6490
Процесор	Intel Core i5-750 S-1156 (8M Cache, 2.66 GHz)	-
Системна плата	MB Asrock H55M-LE s1156 (H55, s1156, DDR3 2600(OC)x, ntel(R) HD Graphics, 1xPCI-E 16x, 2xSATAIII, 4xSATAII, Lan 1000 Mb/s, SB 7.1) mATX	-
Відеокарта	nVidia GeForce 8600 GTS, 256 MB GDDR3, 128-bit / 2x DVI 1x S-Video	-
Жорсткий диск	SSD 480 Gb Samsung	-
Оперативна пам’ять	DIMM 2048Mb DDR3 PC3-10600 1333Mhz Samsung	-
DVD-привод	Super Multi LG SATA DVD±RW R+22x/-22x, RW+8x/-6x, DL+16x/-12x, RAM 12x, SecurDisc, black (GH22NS40RBB)	-
Корпус	ASUS TA-861 500W FSP ATX-500W (Black/Silver panel) ATX (90-PL861AF5C4-53CZ)	-
Кулер	–	–
Кардрідер внутрішній	USB 2.0 Card reader STORM CR-35U1A4-B, int. 3.5", 1*USB2.0+AUDIO+1394, multi: All Type Cards, black	240
інше	Клавіатура, мишка	-

Продовження таблиці 7.6

Найменування комплектуючої або обладнання	Тип	Оптова ціна
Монітор	22" TFT, ASUS VW223D (5ms, 300/3000: 1, 170/160, D-SUB, Wide)	3200
Принтер лазерний	Canon i-SENSYS LBP6030W	2700
Принтер струминний	Epson Stylus Photo P50 (C11CA45341) + USB cable	5500
Сканер	Epson Perfection V37 Photo	2970
Копіювальний апарат	Canon i-SENSYS MF217W with Wi-Fi	5965
Пристрій живлення	UPS APC BACK-UPS ES 525VA 230V RUSSIA (BE525-RS)	1496

Витрати на транспорт, монтаж та випробування можуть бути прийняті в межах до 10% від оптової ціни. Для визначення необхідної кількості капітальних вкладень складемо таблицю 7.8.

Таблиця 7.7 – Балансова вартість обчислювальної техніки

Найменування обчислювальної техніки	Кількість, шт.	Ціна за одиницю, грн.	Витрати на транспортування, монтаж та випробовування.	Загальна вартість, грн.
Персональні комп’ютери	8	11186	8948,8	98436,8
Принтер лаз.	2	2700	540	5940
Принтер струм.	1	5500	550	6050
Сканери	1	2970	8948,8	98436,8
Копіюв. апарат	1	5965	540	5940
Всього	–	–	–	214803,6

Таблиця 7.8 – Вартість основних фондів та амортизаційні відрахування розробника

Групи та види основних фондів	Балансова вартість, грн.	Амортизація	
		Норма, %	Відрахування, грн.
1	2	3	4
Група 3			
Будівлі	1280000	-	-
Передавальні пристрої	128000	-	-
Всього по групі	1408000	5	70400
Група 4			
Обчислювальна техніка	214804	-	-
Всього по групі	214804	50	107402
Група 5, 6			
Вимірювальні пристрої	5190	25	-
Транспортні засоби	0	20	-
Господарський інвентар	28000	25	-
Всього по групі	33190	-	8297,5
Нематеріальні активи			
Нематеріальні активи	22000	10	2200
Разом	$К_р$ = 1677994		$А_р$ = 188299,5

Примітка: вартість автомобіля приймаємо рівною нулю.

7.5 Визначення собівартості розробки та ціни програмної продукції

Визначимо основну зарплату виконавців:

$$З_о = \frac{З_{сд} \cdot Т_{пз}}{N_е}, \quad (7.11)$$

де: $N_е$ – кількість екземплярів програм, шт.

$$З_о = 794 \cdot 158/22 = 5700 \text{ грн.}$$

Визначимо додаткову зарплату (оплата відпусток, виконання державних та суспільних обов'язків) на рівні 10%:

$$З_д = З_о \cdot Н_q \cdot 0{,}01, \quad (7.12)$$

де: $Н_q$ – норматив додаткової зарплати, %.

$$З_д = 5700 \cdot 10 \cdot 0{,}01 = 570 \text{ грн.}$$

Відрахування на соціальні потреби за нормативом $Н_с = 22\%$ від суми основної та додаткової зарплати:

$$С_{оц} = 0{,}01 \cdot Н_с (З_о + З_д), \quad (7.13)$$

де: $Н_с$ – відрахування на соціальні потреби, %.

$$С_{оц} = 0{,}01 \cdot 22(5700+570) = 1379 \text{ грн.}$$

Визначимо загальногосподарські витрати (електроенергію, ремонт і утримання приміщень і т.д) за нормативом $Н_г = 15\%$ від основної зарплати:

$$Г_{осп} = З_о \cdot Н_г \cdot 0{,}01, \quad (7.14)$$

де: $Н_г$ – загальногосподарські витрати, %.

$$Г_{осп} = 5700 \cdot 15 \cdot 0{,}01 = 855 \text{ грн.}$$

Визначимо витрати на матеріали для розробки програмної продукції за нормами споживання та діючими цінами за одиницю виміру:

$$З_М = (З_{М1} + З_{М2} + З_{М3})/N_е, \quad (7.15)$$

де: $З_{М1}$ – вартість паперу, грн.;

$З_{М2}$ – вартість запам'ятовуючих пристроїв, грн.;

$З_{М3}$ – вартість фарби, картріджей, тонеру, грн.;

N_e – кількість екземплярів програм, шт.

Згідно прийнятих норм на підприємстві $n_{вит}$ приймаємо 0,75 пачки паперу на період розробки. Тоді, враховуючи, що вартість пачки паперу складає $Ц_п$=200 грн., визначаємо вартість паперу за період розробки:

$$З_{М1} = Ц_п \cdot N_м. \quad (7.16)$$

$$З_{М1} = 200 \cdot 0,75 = 150 \text{ грн.}$$

Згідно прийнятих норм по комплектації до вартості запам'ятовуючих пристроїв входить вартість CD/DVD дисків. Їх кількість дорівнює кількості коробочних версій запропонованого продукту (приймаємо 10):

$$З_{М2} = \sum Ц_д\ , \quad (7.17)$$

де: $Ц_д$ – вартість дисків CD/DVD: CDR box – 27,6 грн./шт., DVD-R box – 32,15 грн./шт.

$$З_{М2} = 10 \cdot 27,6 = 276 \text{ грн.}$$

Згідно виданих викладачем норм одноразовій заправці підлягають усі друкуючі пристрої і становить:

$$З_{М3} = \sum Ц_з., \quad (7.18)$$

де: $Ц_з$ – вартість розхідних матеріалів друкуючих пристроїв: відновлення та заправка картриджу для Canon i-SENSYS LBP6030W – 574 грн.; картридж для Epson Stylus Photo P50 – 558 грн.; відновлення картриджу для MF217W – 570 грн.

$$З_{М3} = 574+558+570 = 1702 \text{ грн.}$$

$$З_М = (150 + 276 + 1702)/22 = 97 \text{ грн.}$$

Визначимо витрати на освоєння нових мов програмування або операційних систем за нормативом ($Н_п$ = 15%) від основної зарплати виконавців:

$$О_п = З_о \cdot Н_п \cdot 0,01, \quad (7.19)$$

де: $Н_п$ – норматив витрат на освоєння нових мов програмування, %.

$$О_п = 5700 \cdot 15 \cdot 0,01 = 855 \text{ грн.}$$

Визначимо витрати на амортизацію основних фондів з урахуванням загальної річної суми амортизаційних відрахувань та кількості екземплярів програм (N_e = 22 прим.):

$$A_м = \frac{A_р \cdot N_{міс}}{N_е \cdot 12}, \quad (7.20)$$

де: $A_р$ – загальна річна сума амортизаційних відрахувань, грн.

$$A_м = 188300 \cdot 3/(22 \cdot 12) = 2140 \text{ грн.}$$

Повна собівартість ПЗ визначається як сума витрат за попередніми статтями калькуляції:

$$C_п = З_о + З_д + С_{оц} + Г_{осп} + З_м + О_п + А_м. \quad (7.21)$$

$$C_п = 5700+570+1379+855+97+855+2140 = 11596 \text{ грн.}$$

Визначимо плановий прибуток за рівнем рентабельності ($P_п$) програмної продукції, яка залежить від складності програми та ступеня новизни задачі.

Для даного програмного забезпечення рівень рентабельності складає 50%.

$$П_р = 0{,}01 \cdot P_п \cdot C_п, \quad (7.22)$$

де: $P_п$ – рівень рентабельності, %.

$$П_р = 0{,}01 \cdot 50 \cdot 11596 = 5798 \text{ грн.}$$

Величини ціна підприємства, податок на додану вартість, відпускна ціна програмної продукції визначаються за формулами, приведеними в таблиці 7.9

Таблиця 7.9 – Нормативна калькуляція собівартості розробки програмного забезпечення задачі

Найменування статей витрат	Позначення	Величина, грн.
1	2	3
1. Основна зарплата виконавців	$З_о$	5700
2. Додаткова зарплата виконавців	$З_д$	570
3. Відрахування на соціальні потреби	$С_{оц}$	1379
4. Загальногосподарські витрати	$Г_{осп}$	855
5. Витрати на матеріали	$З_М$	97
6. Освоєння нових операційних систем, мов програмування	$О_п$	855

Продовження таблиці 7.9

1	2	3
7. Амортизація основних фондів	$А_м$	2140
8. Повна собівартість програмного забезпечення	$С_п$	11596
9. Плановий прибуток	$П_р$	5798
10.Ціна підприємства $Ц_п = С_п + П_р$	$Ц_п$	17394
11.Податок на додану вартість $ПДВ = 0.01 \cdot Н_{дв} \cdot Ц_п$	$ПДВ$	3478,8
12.Відпускна ціна програмної продукції $Ц = Ц_п + ПДВ$	$Ц$	20872,8

7.6 Визначення об’єму капітальних вкладень у споживача програмної продукції

Об’єм капітальних вкладень у споживача програмної продукції визначаємо на основі балансової вартості основних фондів, яка враховує ціну, транспортно-заготівельні витрати, вартість будівель, монтажних та пусконалагоджувальних робіт, а також витрати на випробування у виробничих умовах. Результати розрахунків зводимо у таблицю 7.9.

Таблиця 7.10 – Розрахунок об’єму капітальних вкладень у споживача програмної продукції

Найменування капітальних вкладень	Сума за варіантами, грн.	
	Базовий	Новий
Вартість програмної продукції	–	20873
Всього капітальних витрат	–	20873

7.7 Визначення експлуатаційних витрат

Експлуатаційні витрати у споживача програмної продукції визначаємо при умові роботи підсистеми на протязі року. Результати зводимо до таблиці 7.11.

Таблиця 7.11 – Розрахунок експлуатаційних витрат у споживача програмної продукції

Найменування статей витрат	Позначення	Сума витрат за варіантами, грн.	
		Базовий	Новий
1. Витрати на обслуговування	$З_р$	80520	28182
2. Витрати на електроенергію	$З_{ел}$	-	-
Витрати на амортизацію	$З_{ам}$	-	5218
Всього витрат за рік	$І$	80520	33400

Витрати на обслуговування:

$$З_р = Т_р \cdot З_г \cdot (1 + 0{,}01 \cdot Н_q) \cdot (1 + 0{,}01 \cdot Н_c), \qquad (7.23)$$

де: $Т_р$ – кількість годин, що витрачається на обслуговування на рік, *год.* (приймаємо 1000 год.);

$З_г$ – заробітна плата обслуговуючого персоналу, *грн / год*.

$$З_{р\ баз} = 1000 \cdot 60 \cdot 1{,}1 \cdot 1{,}22 = 80520\ грн,$$

до:

$$З_{р\ нов} = 350 \cdot 60 \cdot 1{,}1 \cdot 1{,}22 = 28182\ \ грн.$$

Витрати на електроенергію визначаються з урахуванням споживаємої потужності ($П_{ел}$) в кіловатах, часу експлуатації технічних засобів ($Т_р$) в годинах та ціни однієї кіловат-години ($Ц_{ел}$):

$$З_{ел} = П_{ел} \cdot Т_р \cdot Ц_{ел}. \qquad (7.24)$$

$$З_{ел\ баз} = З_{ел\ нов}$$

Витрати по амортизації визначаються на основі норм амортизаційних відрахувань, вартості програмної продукції і основних фондів. Для розрахунку складаємо таблицю 7.12.

Таблиця 7.12 – Розрахунок амортизаційних відрахувань

Групи основних фондів	Норма амортизації %	Балансова вартість, грн., за варіантами		Сума відрахувань, грн., за варіантами	
		Базовий	Новий	Базовий	Новий
Програмна продукція	25	–	20873	–	5218,25
Всього відрахувань	-	–	20873	–	5218,25

7.8 Визначення економічної ефективності програмної продукції

Економічна ефективність програмного забезпечення визначається для виготовлювача і споживача за такими показниками.

Величина економічного ефекту при виготовленні програмної продукції, розраховуємо за формулою:

$$E_в = (Ц_п - C_п) \cdot N_е - \sum_{i=1}^{m} E_{p_m} \cdot K_{p_m}, \quad (7.25)$$

де: $K_р$ – балансова вартість основних фондів розробника, грн.; $E_р$ – розрахунковий коефіцієнт капіталовкладень.

$$Ев = (17394 - 11596) \cdot 22 - (0,05 \cdot 1408000 + 0,5 \cdot 214804 +$$
$$+0,25 \cdot 33190 + 0,1 \cdot 22000) \cdot 3/12 = 80481 \; грн.$$

Визначимо період окупності додаткових капітальних вкладень у виробника програмної продукції:

$$T_в = \frac{K_р}{(Ц_п - C_п) \cdot N_е}, \quad (7.26)$$

де: $K_р$ – балансова вартість основних фондів розробника.

$$T_в = \frac{1677994}{(17394 - 11596) \cdot 21 \cdot 12 / 3} = 3,2 \text{ років}.$$

Визначимо величину економічного ефекту у користувача програмної продукції за формулою:

$$E_{сп} = (I_б - I_н) - E_н(K_н - K_б), \quad (7.27)$$

де: $I_б$, $I_н$ – величина експлуатаційних витрат за базовим и новим варіантом відповідно; $K_б$, $K_н$ – об'єм капітальних вкладень за варіантами, що порівнюються.

$$E_{сп} = (80520\text{-}33400)\text{-}0,25 \cdot 20873 = 41902 \text{ грн.}$$

Визначимо період окупності додаткових капітальних вкладень у споживача програмної продукції за рахунок зниження експлуатаційних витрат:

$$T_{сп} = \frac{K_н - K_б}{I_б - I_н}, \quad (7.28)$$

$$T_{сп} = \frac{20873}{80520 - 33400} = 0,44 \text{ року}.$$

Показники економічної ефективності програмної продукції зводимо до таблиці 7.13.

Таблиця 7.13 – Показники економічної ефективності програмної продукції

Найменування показників	Одиниця виміру	Величина
1. Кількість екземплярів програми	Прим.	22
2. Повна собівартість розробленої програми	Грн.	11596
3. Ціна розробленої програми	Грн.	17394
4. Плановий прибуток від реалізації одної програми	Грн.	5798
5. Рентабельність програмної продукції	%	50
6. Об'єм додаткових капітальних вкладень у виробника програмної продукції	Грн.	1677994
7. Загальний прибуток від реалізації програмної продукції	Грн.	127556

Продовження таблиці 7.13

Найменування показників	Одиниця виміру	Величина
8. Величина економічного ефекту при виготовлені програмної продукції	Грн.	80481
9. Період окупності додаткових капітальних вкладень у виробника програмної продукції	Рік	3,2
10.Об'єм додаткових капітальних вкладень у споживача програмної продукції	Грн.	20873
11.Величина економічного ефекту у користувача програмної продукції	Грн.	41902
12.Період окупності додаткових капітальних вкладень у користувача програмної продукції	Рік	0,44

7.9 Висновки

Розроблена програма економічно вигідна. За рахунок впровадження програмного забезпечення досягається скорочення часу обробки інформації, підвищується культура праці, підвищення якості приймаючих управлінських рішень.

8 ЗАХОДИ ЩОДО ОХОРОНИ ПРАЦІ І ТЕХНІКИ БЕЗПЕКИ

8.1 Вступ

Законом України “Про охорону праці” [93] регламентуються загальні положення державної політики в галузі охорони праці, а конкретизуються ці положення нормативно-правовими актами про охорону праці, зокрема Наказом Міністерства соціальної політики України 14.02.2018 № 207, який зареєстровано в Міністерстві юстиції України 25 квітня 2018 р. за №508/31960 «Про затвердження Вимог щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями» [95], яким затверджено нормативно-правовий акт з охорони праці НПАОП 0.00-7.15-18, «Правила охорони праці під час експлуатації електронно-обчислювальних машин», та «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин» ДСанПіН 3.3.2-007-98 [92].

Загальні вимоги пожежної безпеки під час експлуатації комп’ютерної техніки визначають «Правила пожежної безпеки в Україні» (затверджені наказом МВС від 30.12.2014 № 1417) [103], комп’ютерних класів — пункт 3 розділу VIII «Правил пожежної безпеки для навчальних закладів та установ системи освіти України» (затверджені наказом МОН від 15.08.2016 № 974). [102] та інші державні стандарти, що регламентують експлуатування комп’ютерної техніки як радіоелектронної апаратури.

Охорона здоров'я робітників, забезпечення безпеки умов праці, ліквідація та профілактика професійних захворювань і виробничого травматизму складає одну з головних турбот людського суспільства.

8.2 Пожежна безпека

Вимоги до пожежної безпеки на підприємстві неухильно повинен дотримуватися кожен співробітник, а організаційна складова при цьому покладається на посадових осіб за відповідним рішенням керівництва і прописується в посадових інструкціях і положеннях по структурним підрозділам.

Зокрема, вказуються конкретні території, ділянки, зони, об’єкти, цілі будівлі і їх частини, поверхи, на яких відповідального співробітника повинне проводити такі організаційні роботи.

Відповідальні особи зобов’язуються розробити, впровадити та підтримувати в певному інструкцією і положенням на ввірених їм об’єктах протипожежний режим і інструкції відповідно до вимог, викладених в нормативних актах.

Пожежі в приміщеннях з оргтехнікою становлять особливу небезпеку, бо поєднані з великими матеріальними збитками. Пожежа може виникнути при взаємодії горючих речовин і джерел запалювання. Горючими речовинами є будівельні та опоряджувальні матеріали, пластмасові корпуси техніки, шнури тощо. Джерелами запалювання можуть бути електронні схеми комп'ютерів, принтерів, пристроїв електроживлення, де внаслідок різних порушень виникає перегрівання елементів, утворюються електричні іскри та дуги, здатні спричинити займання горючих матеріалів.

При обслуговуванні, ремонтних та профілактичних роботах використовуються різні легкозаймисті рідини, прокладаються тимчасові електропровідники, здійснюється паяння. Виникає додаткова пожежна небезпека, яка потребує відповідних заходів пожежного захисту. До засобів гасіння пожежі, призначених для локалізації невеликих займань, належать вогнегасники, сухий пісок, азбестові ковдри.

Приміщення, в який встановлено комп'ютери і де немає необхідності влаштування систем автоматичного пожежогасіння, необхідно оснащувати

переносними вуглекислотними з розрахунку 2 шт. на кожні 20 м^2 в приміщеннях. Звуковбирне облицювання стін, стель приміщень треба виконувати з негорючих та важко горючих матеріалів.

З метою виявлення початкової стадії займання необхідно використовувати пристрої систем автоматичного пожежогасіння там, де цього вимагають Правила пожежної безпеки.

З точки зору забезпечення пожежної безпеки до цих заходів можна віднести наявність схеми евакуації з приміщення вузла, у випадку пожежі, повішену на вхідні двері.

8.3 Характеристика умов праці програміста

В приміщенні, в якому проводиться розробка і дослідження програмного продукту, відсутні умови, які можуть створювати підвищену або особливо підвищену небезпеку, тому воно відноситься до класу звичайних приміщень згідно Правил улаштування електроустановок (ПУЕ). Джерелом живлення є трифазна мережа напруги 380/220 В з глухо заземленою нейтралі, з частотою 50 Гц згідно за пожежо-вибухонебезпекою приміщення відноситься до класу В. В таблиці 8.2 наведена загальна характеристика приміщення щодо вибухо-пожеженобезпеки та важкістю робіт.

Таблиця 8.2 – Загальна характеристика приміщення щодо вибухо-пожежонебезпеки та важкістю робіт

Характеристика приміщень	Загальна характеристика	Категорія за важкістю робіт згідно ГН 3.3.5-8.6.6.1 -2002
В – пожежонебезпечне клас П – II	Звичайне без ознак хімічного забруднення та нормальної вологості і за санітарними нормами	1а…….до 139 Вт/м2 1б…….до 140-174 Вт/м2 Клас умов праці – оптимальний

Температура повітря в приміщенні визначається температурою зовнішнього повітря і тепловою енергією, що виділяється всередині приміщення. Джерелами теплоти в даному приміщенні є люди, електроустаткування, а також освітлювальні прилади в темний час доби. Зовнішнім джерелом надлишкового тепла є сонячна радіація у світлий час доби. Робота, виконувана в даному приміщенні, відноситься до категорії I-а. Людиною в цьому випадку виділяється до 120 ккал теплової енергії в годину. Вологість повітря в приміщенні визначається вологістю атмосферного і видихуваного людьми повітря, а також випарами з поверхні шкіри.

У таблиці 8.3 приведені оптимальні значення параметрів мікроклімату для категорії ваги робіт I-а, а також фактичні значення цих параметрів у розглянутому приміщенні. У приміщеннях з використанням обчислювальної техніки рекомендується застосування тільки оптимальних значень показників мікроклімату, тобто таких, при яких людина почуває себе комфортно.

Таблиця 8.3 – значення параметрів небезпечних факторів праці

Найменування параметра	Значення параметра		Нормативний документ
	Фактичне	Нормоване	
1	2	3	4
Освітленість штучна, лк	300	300	ДБН.В 2.5-28:2018 [91]
Значення КПО,%	1,0	1,1	ДБН.В 2.5-28:2018 [91]
Повітрообмін, м /год взимку влітку	 76 36	 80 80	 ДСН 3.3.6.042-99 ДСН 3.3.6.042-99
Температура повітря. °С взимку влітку	 22 24	 21-25 27-28	 ДСанПіН 3.3.2-007-98 ДСанПіН 3.3.2-007-98 [92]
Відносна вологість,% взимку влітку	 60 55	 <75 <60	 ДСанПіН 3.3.2-007-98 ДСанПіН 3.3.2-007-98 [92]

Продовження таблиці 8.3

Швидкість переміщення повітря, м/с взимку	0,16	<0,2	ДСанПіН 3.3.2-007-98 [92]
влітку	0,10	<0,2	ДСанПіН 3.3.2-007-98 [92]

У приміщенні немає виділення шкідливих газів. Тому що в ньому не проводиться монтажних робіт, пайки чи інших робіт, при яких виділяються шкідливі гази.

Для нормалізації параметрів повітряного середовища також періодично здійснюється провітрювання приміщення і вологе прибирання. У всьому будинку діє встановлена загально обмінна витяжна вентиляція.

Раціональне освітлення приміщення сприяє кращому виконанню виробничого завдання і забезпеченню комфорту при роботі. Для забезпечення нормального освітлення застосовуються природне, однобічне, бічне і штучне освітлення, а також сполучене, нормуються згідно ДБН В.2.5-28-2006 Природне і штучне освітлення [91]. Дані по нормах освітлення наведені в таблиці 8.4

Таблиця 8.4 – Норми освітлення

Мінімальний розмір об'єкта розрізнювання, мм	Фон	Контрас	Розряд, під розряд зорової роботи	Нормоване значення		
				Природне освітлення КПО,%	Штучне освітлення	
					$E_{мін.}$ лк	Тип ламп
Від 0,3 до 0,5	Світлий	Середній	IIIг	1,5	300	Газоро зрядні

За результатами виміру освітленості величина освітленості від системи загального штучного освітлення дорівнює 310 лк, що відповідає вимогам, які пред'являються до даного приміщення.

Основними джерелами шуму на робочих місцях, обладнаних відео дисплейними терміналами, є принтер, сканер факс і обладнання для кондиціювання повітря, в самих відео дисплейних терміналів – вентилятори систем охолодження і трансформатори.

Згідно ДСанПіН 3.3.2.007-98 [92] допустимий еквівалентний рівень шуму для робочого місця програміста складає 50 дБА (акустичних децибела).

8.4 Розробка заходів з умов поліпшення охорони праці

Важливими є заходи щодо забезпечення умов праці на робочому місці користувача ПК.

З точки зору забезпечення електробезпеки до цих заходів можна віднести: устаткування розподільних щитів спеціальними розетками з заземлюючими контактами; організація заземлення всіх приладів і пристроїв; періодична перевірка всіх приладів і пристроїв; щорічна здача іспитів з охорони праці.

З точки зору забезпечення оптимальних умов мікроклімату, рівня звуку і освітленості до цих заходів можна віднести: організацію природної вентиляції, за допомогою дефлектора, для забезпечення необхідного повітрообміну в приміщенні вузла; організацію системи центрального опалювання, для підтримки оптимальної температури в холодний період року; організацію штучного загального освітлення, для забезпечення необхідних умов зорової роботи, що відповідають, оформлення паспорта на приміщення вузла, з занесенням в нього вимірювань освітленості і рівня звуку, проведених відділом охорони праці.

Крім рекомендацій щодо конкретного приміщення, де було проведено дослідження умов праці, існують загальні вимоги, які зарекомендовані відповідними нормативними документами.

Правильна організація робочих місць запобігає передчасній втомлюваності користувача і сприяє збереженню здоров'я. Організація робочого місця передбачає:

- правильне розміщення робочого місця у виробничому приміщенні;
- вибір ергономічного обґрунтованого робочого положення, виробничих меблів з урахуванням характеристик людини;.
- раціональне компонування обладнання на робочих місцях;
- урахування характеру й особливостей трудової діяльності.

Стосовно робочих місць користувача ВДТ, то організація робочого має забезпечуватися відповідно до ДСанПіН 3.3.2-007-98. Для запобігання перевтомленню необхідно виконувати вправи для очей та дотримуватись розпорядку роботи та відпочинку. На робочому місці реалізовувався режим відпочинку: кожні дві години – перерва для виконання фізичних вправ для м‘язів очей.

Додаткові заходи: регулярні періодичні наочні огляди персоналом шляхів для евакуації людей із приміщення, відповідно до плану евакуації (який повинен розташовуватись на видному місці у приміщенні), включення до колективного договору мінімально можливого вмісту аптечок з обов’язковою наявністю масок-клапанів, або іншого спорядження для штучного дихання.

8.5 Розрахункова частина

Для захисного штучного заземлення застосовуються ветикальні електроди: металевий куток *63·63·6 мм*., (згідно з ДСТУ 2251-93 «Кутики сталеві гарячекатані рівнополичні. Сортамент») довжиною *L=1,5 м.,* та горизонтальний електрод — металева полоса з перетином *60·5 мм*. Напруга — *220/380 В*. Розрахункова схема розташування заземлюючих електродів — по контуру (прямокутником).

Розрахунок проведем за допустимим опором розтіканню струму заземлювача.

Початкові дані для розрахунку захисного заземлення: тип верхнього шару грунта — чорнозем, нижнього шару грунта — глина (питомий опір $\rho_2 = 40$ *Ом·м)*.

Умовна товщина верхнього шару грунта: *H=0,45 м*. Відстань між вертикальними заземлювачами (електродами) *A=3 м*. Глибина закладення горизонтального контура заземлення *t=0,65 м*. Опір заземлювача, який нормується: $R_{ЗН}$ *= 4 Ом*. Необхідно визначити необхідну кількість вертикальних заземлювачів та длину полоси (горизонтального заземлювача).

Розрахунок.

Відстань від центра вертикального заземлювача до поверхні землі:

T=t+L/2=0,65 +1,5/2=1,35 м.

Розрахунковий питомий опір грунта (з врахуванням того, що фактично вся конструкція заземлювача розташовується у нижньому шарі грунта):

$\rho = \psi\rho = 1{,}36 \cdot 40 = 54{,}5$ *Ом·м.*

де $\psi = 1{,}36$ - табличне значення коефіцієнта сезонності для відповідної климатичной зони у багатошаровому грунті [101];

$\rho_2 = 40$ *Ом·м.* - табличне значення питомого опору нижнього шару грунта (глина) [101].

Еквівалентний діаметр ветикальнього електрода (кутка) [101]:

$$D_в = 0{,}95 \cdot K = 0{,}95 \cdot 63 = 59{,}85 \text{ мм.} = 0{,}0598 \text{ м.}$$

де *K= 63 мм.* - розмір метелевого кутка (задан).

Відношення *A/L=3/1,5=2*

Опір розтіканню електричного струму одного електрода вертикального заземлювача з урахуванням заглиблення заземлювача [101]:

$$R_O = 0{,}366(\rho/L)[\lg(2L/D_в) + (1/2)\lg((4T+L)/(4T-L))] =$$
$$= 0{,}366(54{,}5/1{,}5)[(\lg(2 \cdot 1{,}5/0{,}0598) + (1/2)\lg((4 \cdot 1{,}35+1{,}5)/(4 \cdot 1{,}35-1{,}5))] =$$
$$= 24{,}2 \text{ Ом.}$$

Визначаємо коефіцієнт екранування вертикальних електродів *Кев=0,62* при орієнтовній кількості вертикальних електродів, яке дорівнює 5 [101].

Визначаємо необхідну кількість вертикальних електродів заземлювача (без врахування горизонтального заземлювача), при $R_{ЗН} = 4$ *Ом*:

$$N=R_O /(Кев\ R_{ЗН})= 24,2 / (0,62\cdot4)= 9,76 \approx 10\ шт.$$

Визначаємо довжину з'єднуючої полоси:

$$L_П =1,05\cdot A\cdot N= 1,05\cdot3\cdot10=30,75\approx 31\ м.$$

Опір розтіканню електричного струму з'єднуючої полоси з урахуванням кліматичного коефіцієнта питомого опору грунта $K_П$ [101]:

$$R_П =0,366(\rho\cdot K_П/L_П)lg(2\cdot L_П^{\,2} /(B\cdot t))=$$
$$=0,366(40\cdot5/31)\cdot lg((2\cdot31^{\,2})/(0,06\cdot0,65))=11,4\ Ом.$$

де $K_П=5$ - табличне значення кліматичного коефіцієнта питомого опору грунта для відповідної кліматичної зони для з'єднуючої полоси [101]:

$B = 60\ мм. = 0,06\ м.$- ширина з'єднуючої полоси (задана).

Загальний опір розтіканню електричного струму заземлювача [11]:

$$R =(R_O\cdot R_П)/(R_O\cdot\eta_П+ N\cdot R_П\cdot Кев)=$$
$$=(24,2\cdot 11,4)/(24,2\cdot 0,6+ 10\cdot 11,4\cdot 0,62)=3,28\ Ом.$$

де $\eta_П = 0,6$ - табличне значення коефіцієнта екранування з'єднуючої полоси [101].

Умова $R \le R_{ЗН}$ виконується ($3,28 \le 4$).

Так як при *10* вертикальних електродах R суттєво більше $R_{ЗН}$, зменшимо кількість вертикальних електродів N до *8* і виконаємо перерахунок. У результаті остаточно отримали: $R=3,95$ *Ом.* при кількості вертикальних електродів $N=8$.

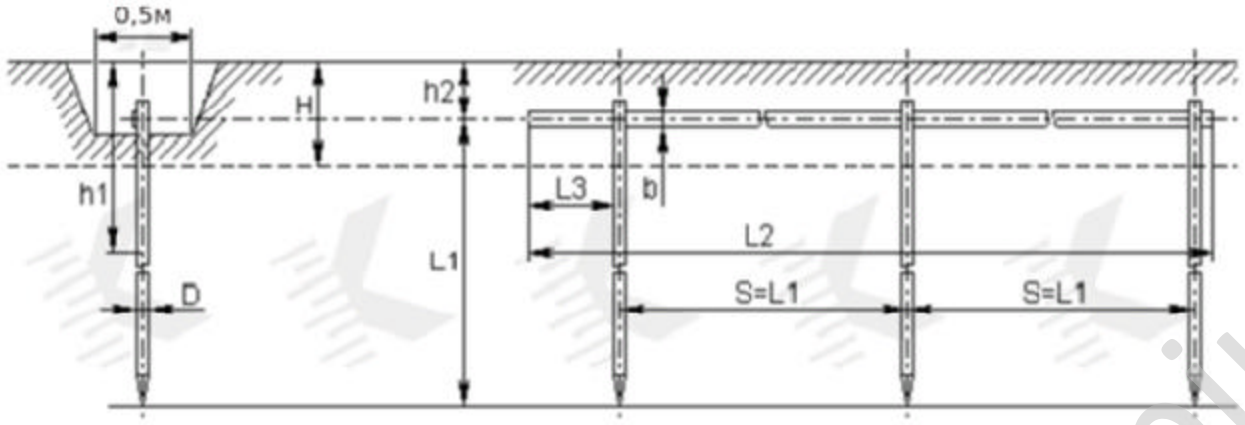

Рис. 8.1 — Схема штучного заземлення .

8.6 Висновки до розділу

У даному розділі магістерської роботи проведено аналіз умов праці працівника, робота якого пов'язана з комп'ютерною технікою. Проведено аналіз основних санітарно – гігієнічних показників в заданому приміщені, де працівник зайнятий постійною роботою за комп'ютером. Створені умови повинні забезпечувати комфортну роботу. На підставі вивченої літератури з даної проблеми, були зазначені оптимальні параметри мікроклімату, освітлення, допустимі рівні шуму при роботі з ПЕОМ.

Виконано розрахунок захисного штучного заземлення, як одного з ключових факторів безпеки програміста.

9 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання кваліфікаційної магістерської роботи, призначене для дослідження та програмної реалізація комп'ютерної моделі репутаційної системи соціальної мережі.

В межах України в недостатній мірі представлені вітчизняні розробки в цій області. Рішення завдання полягало у вирішенні наступних задач:

а) було проведено огляд існуючих систем дослідження та програмні реалізації комп'ютерної моделі репутаційної системи соціальної мережі;

б) досліджена програмна реалізація комп'ютерної моделі репутаційної системи соціальної мережі;

в) на основі отриманих результатів досліджень створена програмна реалізація комп'ютерної моделі репутаційної системи соціальної мережі.

Розроблені під час виконання кваліфікаційної магістерської роботи алгоритми дозволяють успішно вирішувати завдання моделювання та дослідження репутаційної системи соціальної мережі.

Розроблене програмне забезпечення має простий, дружній та зручний інтерфейс користувача, що забезпечує легкість у освоєнні роботи програмного продукту, зручність у використанні, і не потребує особливих спеціальних знань.

При створені програмного забезпечення було використано об'єктно-орієнтований підхід, що відповідає сучасним тенденціям у галузі розробки комерційних програмних систем та забезпечує механізм масштабування програмного засобу у можливих майбутніх оновленнях або покращеннях.

Програма реалізована на мові високого рівня Java. Дана мова програмування дозволяє найбільш ефективно обробляти дані що використовуються в моделях репутаційних систем соціальних мереж. Це дозволило мінімізувати строк розробки програмного забезпечення, і зменшити витрати на його розробку.

Програма призначена для виконання під управлінням багатозадачної операційної системи Windows 7/8/10/11 або Linux.

Даються необхідні рекомендації з інтегрування компонентів розробленого програмного забезпечення в існуючу апаратну систему.

Розроблене програмне забезпечення розповсюджується по ліцензії "open-source software". Вихідний код таких програм доступний для перегляду, вивчення та зміни, що дозволяє переконатися у відсутності вразливостей і неприйнятних для користувача функцій а також з’являється можливість додавання своїх власних модулів або повна модернізація застосунка.

В цілому створене програмне забезпечення підтверджує правильність використаних проектних рішень та повністю відповідає вимогам технічного завдання. Створене програмне забезпечення має потенційну можливість для подальшого вдосконалення і застосування у різних галузях.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Ляпин Д.А. PHP — это просто. Начинаем с видеоуроков (+ CD-ROM); БХВ-Петербург - М., 2017. - 881 с.

2. Бенкен Елена PHP, MySQL, XML. Программирование для Интернета; БХВ-Петербург - М., 2017. - 336 с.

3. Ляпин Дмитрий, Никитин Александр PHP - это просто. Начинаем с видеоуроков; БХВ-Петербург - М., 2017. - 642 с.

4. Яргер, Р.Дж.; Риз, Дж.; Кинг, Т. MySQL и mSQL: Базы данных для небольших предприятий и Интернета; СПб: Символ-Плюс, 2013. - 560 с.

5. Дунаев В.В. HTML, скрипты и стили; БХВ-Петербург - М., 2017. - 527 с.

6. Жадаев Александр PHP для начинающих; Питер - М., 2016. - 768 с.

7. Аткинсон, Леон MySQL. Библиотека профессионала; М.: Вильямс, 2008. - 624 с.

8. Бибо Бер, Кац Иегуда jQuery. Подробное руководство по продвинутому JavaScript; Символ-плюс - М., 2017. - 624 с.

9. Дунаев Вадим HTML, скрипты и стили; БХВ-Петербург - М., 2015. - 816 с.

10. Інструмент розробника PyQt5 [Електронний ресурс] – Режим доступу до ресурса: https://github.com/Abdelatief/Pyuic5-Tool

11. Скоринкин А. И., "Математическое моделирование биологических процессов", Казань: Казан. ун-т, 2015, с. 86.

12. Бардзелл Джеффри Macromedia Dreamweaver MX 2004 с ASP, ColdFusion и PHP. Из первых рук (+ CD-ROM); Эком - М., 2016. - 560 с.

13. Стоунз, Ричард; Мэттью, Нейл PostgreSQL. Основы; СПб: Символ-Плюс, 2007. - 640 с.

14. Джеймс Р. Грофф, Пол Н. Вайнберг, Эндрю Дж. Оппель, "SQL: полное руководство, 3-е издание", Вильямс, 2014, с. 960.

15. Прохоренок Н. А., "Python 3 и PyQt. Разработка приложений", СПб.: БХВ-Петербург, 2012, с. 704.

16. John P. A. Loannidis, "Infection fatality rate of COVID-19 inferred from seroprevalence data", Bulletin of the World Health Organization, 2020, с. 37.

17. John B. Moore, Carl H. June, "Cytokine release syndrome in severe COVID-19", Science, 2020, с. 473-474.

18. Олифер В. Г., Олифер Н. А., "Компьютерные сети. Принципы, технологии, протоколы: Учебник для вузов", СПб.: Питер, 2001, с. 672.

19. Глыбочко П.В., Фомин В.В., Авдеев С.Н., "Клиническая характеристика 1007 больных тяжелой SARS-CoV-2 пневмонией, нуждавшихся в респираторной поддержке", Клиническая фармакология и терапия: журнал, 2020, с. 21-29.

20. Крис Фиайли, "SQL: Руководство по изучению языка", Peachpit Press, 2003, с. 456.

21. Иванов М. А., " Криптографические методы защиты информации в компьютерных системах и сетях", М.: КУДИЦ-ОБРАЗ, 2001, с. 368.

22. C. J. Date, "Date on Database", Apress, 2006, с. 566.

23. Антипова Л.А. Оцінювання рівня дезорганізації в діяльності підрозділів промислового підприємства / Л.А. Антипова, А.А. Алєйніков. – Економіка. Менеджмент. Підприємництво. – 2011. – No23(II). – С. 166-171.

24. Matt Weisfeld, " The Object-Oriented Thought Process", Addison-Wesley Professional, 2013, с. 336.

25. Локхарт Джош Современный PHP. Новые возможности и передовой опыт; ДМК Пресс - М., 2016. - 304 с.

26. Балабанова Л.В. Управління конкурентоспроможністю підприємства / Л.В. Балабанова, Г.В. Кривенко, І.В. Балабанова. - К.: Профессионал, 2009. –256 с.

27. Сафронов Марк Разработка веб-приложений в Yii 2; ДМК Пресс - М., 2016. - 392 с.

28. Gideon Meyerowitz-Katz, Lea Merone, "A systematic review and meta-analysis of published research data on COVID-19 infection-fatality rates", International Journal of Infectious Diseases, 2020, с. 138–148.

29. Жасмин Бланшет, Марк Саммерфилд, "Qt 4. Программирование GUI на C++", КУДИЦ-Пресс, 2008, с. 736 .

30. SQLite [Електронний ресурс] – Режим доступу до ресурса: http://sqlite.org/

31. Мержевич, Влад HTML и CSS на примерах / Влад Мержевич. - М.: "БХВ-Петербург", 2012. - 448 с.

32. J.M. Podolny, J.N. Baron, "Resources and relationships: Social networks and mobility in the workplace", American Sociological Review, 1997.

33. M. Kilduff, W. Tsai, "Social networks and organisations", Sage Publications, 2003, с. 172.

34. Shaozhi Ye, Juan Lang, Felix Wu, "Crawling Online Social Graphs", APWEB'12, 2010.

35. Xu, Guandong et al, "Web Mining and Social Networking: Techniques and Applications", Springer, 2010, с. 209.

36. Колисниченко Денис PHP и MySQL. Разработка Web-приложений; БХВ-Петербург - М., 2017. - 560 с.

37. Ротач В. Я., "Теория автоматического управления", Издательский дом МЭИ, 2008, с. 333.

38. Poole, David, Adrian E. Raftery, "Inference for Deterministic Simulation Models: The Bayesian Melding Approach", Journal of the American Statistical Association 95, 2000, с. 1244–1255.

39. Дьяконов В. П., Круглов В. В., "MATLAB. Анализ, идентификация и моделирование систем. Специальный справочник", СПб.: "Питер", 2002, с. 448.

40. Гарсиа-Молина Г., Ульман Дж., Уидом Дж., " Системы баз данных. Полный курс ", Вильямс, 2003, с. 1088.

41. Кузнецов Максим, Симдянов Игорь Самоучитель PHP 5; БХВ-Петербург - М., 2017. - 560 с.

42. Дронов Владимир JavaScript и AJAX в Web-дизайне; БХВ-Петербург - М., 2015. - 736 с.

43. Кузнецов Максим Самоучитель PHP 5/6; БХВ-Петербург - М., 2017. - 972 с.

44. Гурцевич В. Э., "Онкогенные вирусы человека: от латентного вирусоносительства до возникновения опухоли", РОНЦ им. Н. Н. Блохина, 2013.

45. Дунаев Вадим JavaScript. Самоучитель; Питер - М., 2015. - 400 с.

46. Boudewijn Rempt, "GUI Programming with Python: QT Edition", OpenDocs, 2002.

47. Горбань А. Н., Хлебопрос Р. Г., "Демон Дарвина: Идея оптимальности и естественный отбор", М: Наука. Гл ред. физ.-мат. лит., 1988, с. 208.

48. Гевко І.Б. Методи прийняття управлінських рішень / І.Б. Гевко. – К.: Кондор, 2009. – 187 с.

49. Ізмайлова К.В. Сучасні технології фінансового аналізу: Навч. посіб. / К.В. Ізмайлова. – К.: МАУП, 2003. – 148 с.: іл. – Бібліогр.: с. 142-144.

50. Дж. Вандер Плас, "Python для сложных задач. Наука о данных и машинное обучение", Питер, 2017, с. 576.

51. Бочаров В.В. Финансовый анализ / В.В. Бочаров. – СПб.: Питер, 2007. – 240 с.: ил. – (Серия «Краткий курс»).

52. Гизберт Дамашке PHP и MySQL; НТ Пресс - М., 2016. - 569 с.

53. К. Дж. Дейт., "Введение в системы баз данных", Пер. с англ. — 8-е изд. – М.: Вильямс, 2005, с. 1328.

54. Коннолли Т., Бегг К., "Базы данных. Проектирование, реализация и сопровождение. Теория и практика", 3-е изд. – М.: Вильямс, 2003, с. 1436.

55. Пфаффенбергер HTML, XHTML и CSS. Библия пользователя / Пфаффенбергер и др. - М.: Вильямс; Издание 3-е, 2015. - 752 с.

56. P. J. Carrington, J. Scott, "The Sage Handbook of Social Network Analysis", SAGE, 2011, с. 640.

57. Харрис Энди PHP/MySQL для начинающих; КУДИЦ-Образ - М., 2016. - 384 с.

58. Фейерштейн, С.; Прибыл, Б. Oracle PL/SQL для профессионалов; СПб: Питер, 2005. - 941 с.

59. Gray G. C., Goswami P. R., Malasig M. D., "Adult adenovirus infections:loss of orphaned vaccines precipitates military respiratory disease epidemics", Clinical Infectious Diseases, 2000, с. 663-670.

60. Фленов Михаил PHP глазами хакера; БХВ-Петербург - М., 2016.-991 с.

61. EpiModel [Електронний ресурс] – Режим доступу до ресурса: https://www.epimodel.org/

62. Жасмин Бланшет, Марк Саммерфилд, "Qt 4. Программирование GUI на C++", КУДИЦ-Пресс, 2008, с. 736.

63. Арнольд В. И., "Жёсткие и мягкие математические модели", М.: МЦНМО, 2004.

64. Zdzislaw Brzezniak, "Basic Stochastic Processes: A Course Through Exercises (Springer Undergraduate Mathematics Series)", 2000, с. 236.

65. Дж. Макконнелл, "Основы современных алгоритмов", 2004, с. 230.

66. Когаловский М.Р., "Энциклопедия технологий баз данных", М.: Финансы и статистика, 2002, с. 800.

67. Никсон Робин Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript и CSS; Питер - М., 2017. - 204 с.

68. Хадсон Пол PHP. Справочник; КУДИЦ-Пресс - М., 2016. - 448 с.

69. Майер Б., "Объектно-ориентированное конструирование программных систем", М.: Русская редакция, 2005.

70. Таранчук В.Б., "Основные функции систем компьютерной алгебры", Минск: БГУ, 2013, с. 59.

71. Антошкіна Л.І. Потенціал і розвиток підприємства [Текст] : навч.-метод. посіб. : навч. посіб. для студ. вищ. навч. закл. / Л. І. Антошкіна, О. Л. Горяча. - Донецьк : Юго-Восток, 2012. - 450 с.

72. My T. Thai, Panos M. Pardalos, "Handbook of Optimization in Complex Networks: Communication and Social Networks", Springer, 2011, с. 541.

73. Скляр Дэвид, Трахтенберг Адам PHP. Рецепты программирования; Питер - М., 2017. - 784 с.

74. Стефани Сазерленд, "Как COVID-19 поражает органы чувств", В мире науки, 2021, с. 58-61.

75. Создай свой веб-сайт с помощью HTML и CSS.-М.: Питер, 2013.-569 с.

76. Нанда, А. и др. Oracle PL/SQL для администраторов баз данных; Символ, 2008. - 496 с.

77. Бадд Т., "Объектно-ориентированное программирование", Питер, 2-е издание, 1997, с. 296.

78. Зандстра Мэтт PHP. Объекты, шаблоны и методики программирования; Вильямс - М., 2016. - 560 с.

79. Грофф, Джеймс; Вайнберг, Пол SQL: полное руководство; Киев: BHV, 2005. - 608 с.

80. Число Данбара [Електронний ресурс] – Режим доступу до ресурса: https://ru.wikipedia.org/wiki/Число_Данбара

81. Дронов В. PHP, MySQL и Dreamweaver. Разработка интерактивных Web-сайтов; БХВ-Петербург - М., 2016. - 480 с.

82. Кузнецов М., Симдянов И., Голышев С. PHP 5. Практика создания Web-сайтов; БХВ-Петербург - М., 2017. - 960 с.

83. Шнайдер, Роберт Microsoft SQL Server 6.5. Проектирование высокопроизводительных баз данных; М.: Лори, 2010. - 361 с.

84. C. McCarthy, "Facebook: One Social Graph to Rule Them All?", CBS Interactive Inc., 2010.

85. Мышкис А. Д., "Элементы теории математических моделей", 3-е изд., испр. – М.: КомКнига, 2007, с. 192.

86. Иан Грэхем, "Объектно-ориентированные методы. Принципы и практика", М.: Вильямс, 2004, с. 880.

87. M. Granovetter, "The strength of weak ties", American Journal of Sociology, 1773, с. 1360-1380.

88. Melville P.,Mooney R., Nagarajan R., "Content-Boosted Collaborative Filtering for Improved Recommendations", University of Texas, 2002, с. 187-192.

89. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж., "Приемы объектно-ориентированного проектирования. Паттерны проектирования", М.: Вильямс, 2015, с. 368.

90. Andrew G. Harrison, Tao Lin, Penghua Wang, "Mechanisms of SARS-CoV-2 Transmission and Pathogenesis", Trends in Immunology, 2020, с. 1100-1115.

91. Державні будівельні норми України: ДБН В.2.5-28:2018. - Режим доступу до ресурсу: https://goo.su/9AkQ (дата звернення 19.10.22).

92. Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин: ДСанПІН 3.3.2-007-98. - Режим доступу до ресурсу: https://zakon.rada.gov.ua/rada/show/v0007282-98 (дата звернення 19.10.22).

93. Закон України «Про охорону праці» від 14.10.1992 р. № 2694-XII. - Режим доступу до ресурсу: https://zakon.rada.gov.ua/laws/show/2694-12 (дата звернення 19.10.22).

94. Зеркалов Д. В. Охорона праці в Галузі: Загальні вимоги: навч. посіб. Київ: Основа. 2011. 551 с.

95. Наказ Міністерства соціальної політики України 14.02.2018 № 207 «Про затвердження Вимог щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями». - Режим доступу до ресурсу: https://zakon.rada.gov.ua/laws/show/z0508 (дата звернення 19.10.22).

96. Постанова № 42 від 01.12.1999 Головного державного санітарного лікаря України «Санітарні норми мікроклімату виробничих приміщень ДСН 3.3.6.042-99. - Режим доступу до ресурсу: https://zakon.rada.gov.ua/rada/show/va042282-99 (дата звернення 19.09.22).

97. Оришака, О. В. Основи охорони праці: навч. посіб. / О. В. Оришака, Г. П. Горбачова, К. М. Марченко; М-во освіти і науки України, Центральноукраїн. нац. техн. ун-т. - Кропивницький : ЦНТУ, 2022. - 175 с. – Режим доступу до ресурсу: http://dspace.kntu.kr.ua/jspui/handle/123456789/12161 (дата звернення 19.09.22).

98. Оришака О.В. Охорона праці в галузі та цивільний захист / О.В Оришака, Г.П. Горбачова, О.М. Мезенцева, К.М. Марченко, К.О. Буравченко; М-во освіти і науки України, Центральноукраїн. нац. техн. ун-т. - Кропивницький: Видавець Лисенко В.Ф., 2019. − 226 с. – Режим доступу до ресурсу: http://dspace.kntu.kr.ua/jspui/handle/123456789/9258 (дата звернення 19.09.22).

99. Методичні рекомендації до виконання розділу "Заходи з охорони праці та техніки безпеки" випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти для здобувачів вищої освіти спеціальностей 123 "Комп'ютерна інженерія" та 122 "Комп'ютерні науки" / М-во освіти і науки України, Центральноукраїн. нац. техн. ун-т, каф. кібербезпеки та програм. забезпечення; [укл. О.В. Оришака, К.М. Марченко]. - Кропивницький: ЦНТУ, 2022. — 19 с. [Електронний ресурс]. – Режим доступу: http://dspace.kntu.kr.ua/jspui/handle/123456789/12240 (дата звернення 19.09.22).

100. Охорона праці. Ч. 1. Захисне заземлення: метод. вказ. до викон. розрахунків з викор. персон. ЕОМ ІВМ сумісного типу / Кіровоград. ін-т с.-г. машинобуд. ; [укл. О. В. Оришака, Є. К. Солових, В. О. Оришака]. - Кіровоград: КІСМ, 1997. - 20 с. – Режим доступу: http://dspace.kntu.kr.ua/jspui/handle/123456789/4358 (дата звернення 19.09.22).

101. Сакулин В.П., Шептовицкий В.М. Безопасность труда при монтаже и эксплуатации электроустановок / В.П.Сакулин, В.М.Шептовицкий. – Л. : “Колос”, 1973. – 238 с.

102. Наказ Міністерства освіти і науки України 15.08.2016 № 974 «Про затвердження Правил пожежної безпеки для навчальних закладів та установ системи освіти України» - Режим доступу до ресурсу https://zakon.rada.gov.ua/laws/show/z1229-16#Text (дата звернення 19.10.22).

103. Наказ Міністерства внутрішніх справ України 30.12.2014 №1417 «Про затвердження Правил пожежної безпеки України» - Режим доступу до ресурсу https://zakon.rada.gov.ua/laws/show/z0252-15#Text (дата звернення 19.10.22).

104. Дронов Владимир JavaScript. Народные советы; БХВ-Петербург - М., 2016. - 458 с.

Додаток А
(обов'язковий)

Технічне завдання

Зміст

					ВКРМ-122.22.0008.00.00.ТЗ		
Вим.	Арк.	№ документа	Підпис	Дата			
Розробив		Мосольд М.І..			Дослідження та програмна реалізація комп'ютерної моделі репутаційної системи соціальної мережі	Літ.	Аркуш 1 / Аркушів 6
Перевірив		Мелешко Є.В.				М	
Н. Контр.		Гермак В.С.				ЦНТУ КН-21м	
Затв.		Смірнов О.А.					

1 Найменування та область застосування

Це технічне завдання розповсюджується на дослідження та програмну реалізацію комп'ютерної моделі репутаційної системи соціальної мережі.

2 Підстава для розробки

Підставою для розробки служить завдання на кваліфікаційну магістерську роботу, видане студенту на кафедрі кібербезпеки та програмного забезпечення (нак. № від року).

3 Мета та призначення розробки

Метою магістерської роботи є дослідження та програмна реалізація комп'ютерної моделі репутаційної системи соціальної мережі.

4 Джерела розробки

Джерелом цієї магістерської роботи є існуючі аналоги та стосовна до теми література.

5 Технічні вимоги

5.1 Склад продукції

Складниками розробки є:

– вибір і обґрунтування методів реалізації проекту;

– розробка програмної частин системи, а також розробка взаємодії системи з ОС та з користувачем;

– розробка програми, що реалізує спроектовані алгоритми роботи системи.

5.2 Показники призначення

Система повинна забезпечувати:

– дослідження репутаційної системи соціальної мережі;

– програмну реалізацію комп'ютерної моделі репутаційної системи соціальної мережі;

– цілісність даних у процесі роботи та при зберіганні;

– інтуїтивно зрозумілий та простий інтерфейс.

5.3 Вимоги до функціональних характеристик

Розроблене програмне забезпечення не повинно мати обмежень на версію операційної системи та драйверів.

5.4 Вимоги до архітектури

Компонент, що розробляється повинен використовувати апаратні засоби та системні засоби, що на даному етапі розвитку обчислювальної техніки найбільше поширені.

5.5 Вимоги до надійності

Програмні модулі написані по всім правилам, які стосуються стандартних викликів функцій, процедур, форм і методів, визначених технічною документацією на середовище розробки.

5.6 Умови експлуатації

Робочі місця користувачів ПЗ повинні задовольняти наступним умовам експлуатації:

– температура повітря: 18-20 град. по Цельсію;

– відносна вологість повітря не більше ніж 80%;

– атмосферний тиск 106 кПа.

5.7 Вимоги до складу та параметрів технічних засобів

Програмне забезпечення повинно бути реалізоване на ПЕОМ архітектури IBM PC, працювати в ОС Windows 7/8/10/11 та Linux і з сумісними з цією платформою прикладним програмним забезпеченням та пристроями.

5.8 Вимоги до інформаційної і програмної сумісності

Портативність програмного забезпечення повинна бути забезпечена за рахунок його реалізації стандартного інтерфейсу взаємодії з ОС, що працюють під управлінням ОС Windows 7/8/10/11 та Linux.

5.8.1 Обладнання

Комп’ютер Intel Celeron/512 Mb/2.6 Gb/SVGA 16” 8Mb або сумісні з ним.

5.8.2 Мова програмування

Середовище Java SE 12.

5.8.3 Вхідні дані

Опис алгоритму роботи запропонованої системи.

5.8.4 Вихідні дані

Робоча програма.

6 Вимоги до програмної документації

Програмна продукція повинна бути представлена у виді опису структури даних, опису алгоритму та схем, а також текстів вихідних модулів програмного забезпечення згідно ЄСПД.

7 Економічні вимоги

7.1 Для ПЗ необхідно виробити функціонально-вартісний аналіз варіантів розробки.

7.2 Виконати розрахунок витрат показників економічного ефекту з урахуванням цін на 3 вересня 2022 року.

8 Вимоги щодо охорони праці

В частині охорони праці магістерської роботи повинна бути розглянуті умови праці оператора ПЕОМ, аналіз основних санітарно – гігієнічних показників в приміщені та психофізіологічна сутність і структура виробничої діяльності з позиції дослідження інформаційного навантаження оператора ЕОМ.

9 Перелік документів, що розробляються

- Наукова новизна – 1 аркуш.
- Структурна схема системи – 1 аркуш.
- Функціональна схема системи – 1 аркуш.
- Діаграма процесів – 1 аркуш.
- Блок-схема алгоритму роботи програми – 7 аркушів.
- Показники економічної ефективності – 1 аркуш.
- Пояснювальна записка – 113 аркушів.

10 Етапи розробки

10.1 Збір і обробка інформації по темі магістерської роботи. Постановка задачі на виконання магістерської роботи (складання ТЗ).

10.2 Проведення досліджень або експериментальних робіт для уточнення основних положень магістерської роботи.

10.3 Розробка функціональних схем, блок схем алгоритмів роботи програмного забезпечення.

10.4 Побудова схем взаємодії даних.

10.5 Створення прототипу ПЗ.

10.6 Віднаходження ПЗ, аналіз отриманих результатів.

10.7 Робота над питанням охорони праці і техніки безпеки.

10.8 Розрахунок з техніко-економічного обґрунтування.

10.9 Оформлення пояснювальної записки і виконання робіт по графічній частині.

11 Порядок контролю та приймання

11.1. Подання магістерської роботи на попередній захист 10.12.2022 р.

11.2. Подання магістерської роботи на захист 20.12.2022 р.

Додаток Б

(обов’язковий)

Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

ЗАТВЕРДЖУЮ

Керівник випускної кваліфікаційної роботи

за другим (магістерським) рівнем вищої освіти

_____ Є.В. Мелешко

Дослідження та програмна реалізація комп’ютерної моделі репутаційної системи соціальної мережі

Лістинг програми

Код документу 12

Носій: DVD-диск

Загальна кількість аркушів: 69

Літера: РП

Кропивницький – 2022 року

Основна програма

Файл Model.java основної програми

```
import javax.swing.*;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.LinkedList;
import java.util.Random;

/**
 * Комп'ютерна модель репутаційної соціальної мережі.
 * Між об'єктами створюється подвійний зв'язок.
 */
public class Model extends Thread {
    /**
     * Базове ім'я для потоку.
     */
    private static final String NAME = "Model";
    /**
     * Генератор псевдовипадкових чисел.
     */
    private static final Random RANDOM = new Random();
    /**
     * Затримка між перевірками виконання дій.
     */
    private static final int PAUSE = 200;
    /**
     * Головне вікно.
     */
    private final MainForm mainForm;
    /**
     * Об'єкт, який зберігає налаштування додатка під час його виконання.
     */
    private final Settings settings;
    /**
     * База даних всіх об'єктів, що знаходяться в моделі репутаційної соціальної
мережі.
     * Об'єкти зберігаються за їх унікальними ідентифікаторами.
     */
    private HashMap<Integer, User> database;
    private final ArrayList<Integer> reputationScores;
    private final ArrayList<Integer> qualityScores;

    /**
     * Конструктор класу.
     */
    Model(int number, MainForm mainForm, Settings settings) {
        super(NAME + number);
        this.mainForm = mainForm;
        this.settings = settings;
        database = new HashMap<>();
        reputationScores = new ArrayList<>();
        qualityScores = new ArrayList<>();
    }

    // Частина коду, що належить до багатопотокового програмування.
    /**
     * Чи потрібно додати нові об'єкти.
     */
    public boolean isAddUsers = false;
    /**
     * Чи потрібно додати нові контакти.
     */
    public boolean isCreateContacts = false;
```

```
    /**
     * Чи потрібно перебрати старі контакти.
     */
    public boolean isEnumerateContacts = false;
    /**
     * Чи потрібно оновити статистику.
     */
    public boolean isGetStatistics = false;
    /**
     * Чи потрібно вивести на екран інформацію про всіх користувачів.
     */
    public boolean isPrintUsers = false;
    /**
     * Чи потрібно вивести вікно інформації про зв'язки об'єкта.
     */
    public boolean isLinksList = false;
    /**
     * Чи потрібно завантажити дані з зовнішньої бази даних.
     */
    public boolean isLoadFromFile = false;
    /**
     * Чи потрібно зберегти дані до зовнішньої бази даних.
     */
    public boolean isSaveToFile = false;
    /**
     * Чи потрібно завершити виконання даного потоку.
     */
    public boolean isEnd = false;
    /**
     * Чи потрібно виконати набір команд для додаткових потоків.
     */
    public boolean isMacros = false;

    /**
     * Перевіряє булеві зміні та визначає що саме потрібно зробити.
     * На булеві зміні впливає користувач, натискаючи на кнопки головного
графічного вікна.
     */
    @Override
    public void run() {
        while (!isEnd) {
            if (isAddUsers) {
                addUsers(settings.getNumberNewUsers(), settings.isRandStat());
                isAddUsers = false;
                JOptionPane.showMessageDialog(null,
                        "Додавання нових об'єктів завершено.");
                mainForm.changeButtonsStatus(true);
            } else if (isCreateContacts) {
                createContacts(settings.getMinCount(), settings.getMaxCount());
                if (refreshGraph()) {
                    JOptionPane.showMessageDialog(null,
                            "Перебір старих зв'язків завершено.");
                } else {
                    JOptionPane.showMessageDialog(null,
                            "Помилка! Можливо дані для моделювання відсутні.");
                }
                isCreateContacts = false;
                mainForm.changeButtonsStatus(true);
            } else if (isEnumerateContacts) {
                if
(!Thread.currentThread().getName().equals(MainForm.MAIN_THREAD_NAME)) {
                    JOptionPane.showMessageDialog(null,
                            "Сталася помилка. Додатковий потік взяв роль
головного потоку.");
                    isEnumerateContacts = false;
                    mainForm.changeButtonsStatus(true);
                }
                mainEnumerateContacts();
                if (refreshGraph()) {
```

```
                    JOptionPane.showMessageDialog(null,
                            "Перебір старих зв'язків завершено.");
                } else {
                    JOptionPane.showMessageDialog(null,
                            "Помилка! Можливо дані для моделювання відсутні.");
                }
                isEnumerateContacts = false;
                mainForm.changeButtonsStatus(true);
            } else if (isGetStatistics) {
                Statistics statistics = getStatistics();
                if (statistics == null) {
                    JOptionPane.showMessageDialog(null,
                            "Не вдалось зібрати статистику. Можливо база даних
порожня.");
                } else {
                    mainForm.printStatistics(statistics);
                    JOptionPane.showMessageDialog(null,
                            "Статистику оновлено.");
                }
                isGetStatistics = false;
                mainForm.changeButtonsStatus(true);
            } else if (isPrintUsers) {
                UserInfoForm infoForm = new UserInfoForm(mainForm, database);
                infoForm.setVisible(true);
                isPrintUsers = false;
            } else if (isLinksList) {
                LinksList linksList = new LinksList(settings.getIdToString(),
                    this);
                if (linksList.isSuccess()) {
                    UserInfoForm infoForm = new UserInfoForm(mainForm,
                        linksList);
                    infoForm.setVisible(true);
                } else {
                    mainForm.changeButtonsStatus(true);
                }
                isLinksList = false;
            } else if (isLoadFromFile) {
                database = loadFromFile(settings.getDatabaseName(),
                    MainForm.MAIN_THREAD_NAME);
                isLoadFromFile = false;
                JOptionPane.showMessageDialog(null,
                        "Завантаження даних з бази даних завершено.");
                mainForm.changeButtonsStatus(true);
            } else if (isSaveToFile) {
                saveToFile(settings.getDatabaseName(),
                    Thread.currentThread().getName());
                isSaveToFile = false;
                JOptionPane.showMessageDialog(null,
                        "Збереження даних до бази даних завершено.");
                mainForm.changeButtonsStatus(true);
            } else if (isMacros) {
                // Цей код виконують тільки допоміжні потоки.
                // Для потоку потрібно наповнити його базу даних початковими
даними.
                database = loadFromFile(settings.getDatabaseName(),
                    MainForm.MAIN_THREAD_NAME);
                // Основний метод, заради якого все це і потрібно.
                enumerateContacts(settings.getContactProbability(),
                    settings.isDualDirection());
                // Збереження результатів до зовнішньої бази даних.
                // Цей результат буде оброблено основним потоком.
                saveToFile(settings.getDatabaseName(),
                    Thread.currentThread().getName());
                isMacros = false;
                isEnd = true;
            }
            try {
                Thread.sleep(PAUSE);
            } catch (InterruptedException e) {
```

```
                JOptionPane.showMessageDialog(null,
                        "Помилка! Виникла проблема з затримкою потоків.");
                throw new RuntimeException(e);
            }
        }
    }

    private boolean refreshGraph() {
        try {

reputationScores.add(database.get(settings.getIdToString()).getReputation());

qualityScores.add(database.get(settings.getIdToString()).getQuality());
            mainForm.drawGraph.setScores(reputationScores, qualityScores);
            return true;
        } catch (NullPointerException ignored) {
        }
        return false;
    }

    /**
     * Перебір зв'язків об'єктів для моделювання їх повторного контакту.
     * Виконується головним потоком моделювання.
     * Створює допоміжні потоки, що і виконують основні обчислення.
     * Наприкінці узагальнює результати.
     */
    private void mainEnumerateContacts() {
        // Допоміжні потоки інформацію беруть з бази даних, тому наповнюємо її.
        saveToFile(settings.getDatabaseName(),
            Thread.currentThread().getName());
        // Список додаткових потоків.
        LinkedList<Model> models = new LinkedList<>();
        // Створити потоки та додати їх до списку. Запустити кожний на
виконання.
        for (int i = 1; i < settings.getNumOfThreads(); i++) {
            models.add(new Model(i, mainForm, settings));
            models.getLast().isMacros = true;
            models.getLast().start();
        }

        // Зачекати поки всі додаткові потоки виконають свою роботу.
        for (Model model : models) {
            try {
                model.join();
            } catch (InterruptedException e) {
                JOptionPane.showMessageDialog(null,
                        "Помилка! Виникла проблема з додатковими потоками.");
                throw new RuntimeException(e);
            }
        }

        // Кожний потік створив свою таблицю в БД з результатами.
        // Їх потрібно завантажити в основний потік для подальшого узагальнення.
        ArrayList<HashMap<Integer, User>> allDB = new ArrayList<>();
        for (int i = 1; i < settings.getNumOfThreads(); i++) {
            allDB.add(loadFromFile(settings.getDatabaseName(), NAME + i));
        }

        // Кількість нових таблиць з результатами.
        int sizeOfDB = settings.getNumOfThreads() - 1;
        // Кількість об'єктів в одній таблиці.
        int size = database.size();
        database = new HashMap<>();
        // Перебрати кожний об'єкт у всіх таблицях.
        for (int id = 0; id < size; id++) {
            int reputation = 0, popularity = 0, quality = 0, qualityBonus = 0,
qualityDebuff = 0;
            int numOfPoints = 0, numSuccessContacts = 0;
            HashMap<Integer, Integer> contacts = new HashMap<>();
```

```
            // Перебрати таблиці.
            for (HashMap<Integer, User> oneDB : allDB) {
                User tmpUser = oneDB.get(id);
                if (tmpUser == null) {
                    continue;
                }
                reputation += tmpUser.getReputation();
                popularity += tmpUser.getPopularity();
                quality += tmpUser.getQuality();
                qualityBonus += tmpUser.getQualityBonus();
                qualityDebuff += tmpUser.getQualityDebuff();
                numOfPoints += tmpUser.getNumOfPoints();
                numSuccessContacts += tmpUser.getNumSuccessContacts();
                // Формування списку контактів для об'єкта.
                for (int key : tmpUser.getContactsIds()) {
                    if (contacts.containsKey(key)) {
                        contacts.put(key, contacts.get(key) +
                            tmpUser.getValueFromContacts(key));
                    } else {
                        contacts.put(key, tmpUser.getValueFromContacts(key));
                    }
                }
                // Суму всіх значень потрібно привести до середнього
арифметичного.
                for (int key : contacts.keySet()) {
                    contacts.put(key, contacts.get(key) / sizeOfDB);
                }
            }

            // В нову базу даних середніх результатів додати об'єкти з
результатами.
            // Суму всіх значень потрібно привести до середнього арифметичного.
            database.put(id, new User(id, reputation / sizeOfDB, popularity /
sizeOfDB, quality / sizeOfDB, qualityBonus / sizeOfDB, qualityDebuff / sizeOfDB,
numOfPoints / sizeOfDB, numSuccessContacts / sizeOfDB, contacts));
        }
    }

    /**
     * Додає до бази даних нові об'єкти.
     *
     * @param number    Кількість нових об'єктів.
     * @param isRandStat Чи потрібно генерувати випадкові значення.
     */
    private void addUsers(int number, boolean isRandStat) {
        User user;
        for (int i = 0; i < number; i++) {
            user = new User(isRandStat);
            database.put(user.getID(), user);
        }
    }

    /**
     * Створює нові зв'язки між об'єктами.
     *
     * @param minCount Мінімальна кількість зв'язків.
     * @param maxCount Максимальна кількість зв'язків.
     */
    private void createContacts(int minCount, int maxCount) {
        int numContacts, randID;
        // Перебрати об'єкти з бази даних.
        for (int i = settings.getNumMainUsers(); i < database.size(); i++) {
            User user = database.get(i);
            // Визначити кількість зв'язків для конкретного об'єкта.
            numContacts = RANDOM.nextInt(minCount, maxCount);
            // Створити зв'язки.
            for (int j = 0; j < numContacts; j++) {
                randID = getRandID(user.getID());
                if (user.getContactsIds().contains(randID)) {
```

```
                    // Пропустити об'єкт, якщо між ними вже є зв'язок.
                    continue;
                }
                if (settings.isDualDirection()) {
                    // Генерувати подвійний зв'язок.
                    user.makeContact(database.get(randID));
                    database.get(randID).makeContact(user);
                } else {
                    // Генерувати зв'язок від user до випадкового об'єкта.
                    user.makeContact(database.get(randID));
                }
            }
        }
    }

    /**
     * Перебір зв'язків об'єктів для моделювання їх повторного контакту.
     *
     * @param contactProbability Ймовірність контакту.
     * @param isDualDirection    Чи подвійний зв'язок між об'єктами?
     */
    private void enumerateContacts(int contactProbability, boolean
isDualDirection) {
        for (int i = settings.getNumMainUsers(); i < database.size(); i++) {
            User user = database.get(i);
            for (int id : user.getContactsIds()) {
                // Симуляція вибору дозволяє обраному об'єкту самостійно
вирішити чи здійснювати контакт.
                if (settings.isSimulationSelecting() &&
                        RANDOM.nextInt(User.MIN_PERCENT, User.MAX_PERCENT) >
user.getValueFromContacts(id)) {
                    continue;
                } else if (RANDOM.nextInt(User.MIN_PERCENT, User.MAX_PERCENT) >
contactProbability) {
                    continue;
                }
                if (isDualDirection) {
                    user.makeContact(database.get(id));
                    database.get(id).makeContact(user);
                } else {
                    user.makeContact(database.get(id));
                }
            }
        }
    }

    /**
     * Обирає випадковий id з бази даних.
     * Обраний id буде відрізнятися від переданого в метод (початкового).
     *
     * @param startId Унікальний ідентифікатор початкового об'єкта.
     * @return Унікальний ідентифікатор випадкового об'єкта.
     */
    private int getRandID(int startId) {
        int newID = startId;
        while (newID == startId) {
            int last = settings.getNumMainUsers() == 0 ? database.size() :
settings.getNumMainUsers();
            newID = RANDOM.nextInt(0, last);
        }
        return newID;
    }

    /**
     * Повертає готову базу даних з об'єктами, які отримані з зовнішньої бази
даних.
     *
     * @param databaseName Назва зовнішньої бази даних.
     * @param tableName    Назва таблиці бази даних.
```

```
     * @return База даних інформації.
     */
    public HashMap<Integer, User> loadFromFile(String databaseName, String
tableName) {
        return SQLite.selectFromTable(databaseName, tableName);
    }

    /**
     * Зберігає стан внутрішньої бази даних до зовнішньої бази даних.
     *
     * @param databaseName Назва зовнішньої бази даних.
     * @param tableName    Назва таблиці бази даних.
     */
    private void saveToFile(String databaseName, String tableName) {
        SQLite.createTable(settings.getDatabaseName(),
            Thread.currentThread().getName());
        SQLite.deleteFromTable(settings.getDatabaseName(),
            Thread.currentThread().getName());
        SQLite.insertIntoTable(databaseName, tableName, database);
    }

    /**
     * Збирає загальну статистику про модель в одному об'єкті.
     */
    private Statistics getStatistics() {
        if (database.size() == 0) {
            return null;
        }
        Statistics statistics = new Statistics(settings);
        for (User user : database.values()) {
            statistics.isMinReputation(user);
            statistics.isMaxReputation(user);
            statistics.isMinPopularity(user);
            statistics.isMaxPopularity(user);
            statistics.isMinQuality(user);
            statistics.isMaxQuality(user);
            statistics.isMinNumSuccessContacts(user);
            statistics.isMaxNumSuccessContacts(user);
            statistics.setNumContacts(statistics.getNumContacts() +
                user.getContactsSize());
            statistics.isAverageReputation(user);
            statistics.isAverageQuality(user);
        }
        statistics.setNumUsers(database.size());
        return statistics;
    }

    /**
     * Повертає рядкове представлення бази даних з усіма її внутрішніми даними.
     *
     * @return Рядкове представлення бази даних.
     */
    @Override
    public String toString() {
        return "Model{" + "database=" + database + '}';
    }

    /**
     * Повертає об'єкт з внутрішньої бази даних за його унікальним
ідентифікатором.
     *
     * @param id Унікальний ідентифікатор.
     * @return Об'єкт з внутрішньої бази даних.
     */
    public User getUser(int id) {
        return database.get(id);
    }
}
```

Файл DrawGraph.java основної програми

```
import javax.swing.*;
import java.awt.*;
import java.util.ArrayList;

/**
 * Клас перевизначає стандартну панель для того, щоб виводити на екран графіки.
 */
public class DrawGraph extends JPanel {
    /**
     * Максимальний бал.
     */
    private static final int MAX_SCORE = 110;
    private static final int PREF_W = 1000;
    private static final int PREF_H = 500;
    /**
     * Відступи від країв.
     */
    private static final int BORDER_GAP = 10;
    private static final Color REPUTATION_COLOR = Color.green;
    private static final Color QUALITY_COLOR = Color.yellow;
    private static final Color REPUTATION_POINT_COLOR = new Color(140, 55, 55,
180);
    private static final Color QUALITY_POINT_COLOR = new Color(250, 100, 50,
180);
    private static final Stroke GRAPH_STROKE = new BasicStroke(3f);
    /**
     * Висота точки.
     */
    private static final int GRAPH_POINT_WIDTH = 6;
    /**
     * Кількість відрізків на у лінії.
     */
    private static final int Y_HATCH_CNT = 11;
    /**
     * Масив значень репутації.
     */
    private ArrayList<Integer> reputationScores;
    /**
     * Масив значень якості взаємодії.
     */
    private ArrayList<Integer> qualityScores;
    /**
     * Кількість елементів у масивах.
     */
    private int scoresSize;

    /**
     * Конструктор класу.
     */
    public DrawGraph() {
        this.reputationScores = new ArrayList<>();
        this.qualityScores = new ArrayList<>();
        scoresSize = 0;
    }

    /**
     * Встановлює нові значення для списків та оновлює графік з графічного
інтерфейсу.
     *
     * @param reputationScores Значення репутації.
     * @param qualityScores    Значення якості.
     */
    public void setScores(ArrayList<Integer> reputationScores,
ArrayList<Integer> qualityScores) {
        this.reputationScores = reputationScores;
        this.qualityScores = qualityScores;
        this.scoresSize = reputationScores.size();
```

```
        this.revalidate();
        this.repaint();
    }

    /**
     * Відповідає за зовнішній вигляд графічного елемента.
     * Малює графіки.
     */
    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        Graphics2D g2 = (Graphics2D) g;
        g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
RenderingHints.VALUE_ANTIALIAS_ON);

        double xScale=((double) getWidth() - 2 * BORDER_GAP) / (scoresSize - 1);
        double yScale=((double) getHeight() - 2 * BORDER_GAP) / (MAX_SCORE - 1);

        ArrayList<Point> reputationPoints = new ArrayList<>();
        for (int i = 0; i < scoresSize; i++) {
            int x1 = (int) (i * xScale + BORDER_GAP);
            int y1 = (int) ((MAX_SCORE - reputationScores.get(i)) * yScale +
BORDER_GAP);
            reputationPoints.add(new Point(x1, y1));
        }

        ArrayList<Point> qualityPoints = new ArrayList<>();
        for (int i = 0; i < scoresSize; i++) {
            int x1 = (int) (i * xScale + BORDER_GAP);
            int y1 = (int) ((MAX_SCORE - qualityScores.get(i)) * yScale +
                BORDER_GAP);
            qualityPoints.add(new Point(x1, y1));
        }

        // Створення x та y ліній.
        g2.drawLine(BORDER_GAP, getHeight() - BORDER_GAP, BORDER_GAP,
            BORDER_GAP);
        g2.drawLine(BORDER_GAP, getHeight() - BORDER_GAP, getWidth() -
            BORDER_GAP, getHeight() - BORDER_GAP);

        // Створення позначок на y лінії.
        for (int i = 0; i < Y_HATCH_CNT; i++) {
            int x0 = BORDER_GAP;
            int x1 = GRAPH_POINT_WIDTH + BORDER_GAP;
            int y0 = getHeight() - (((i + 1) * (getHeight() - BORDER_GAP * 2)) /
                Y_HATCH_CNT + BORDER_GAP);
            int y1 = y0;
            g2.drawLine(x0, y0, x1, y1);
            g2.drawString(String.valueOf((i + 1) * 10), x1, y1);
        }

        // Створення позначок на x лінії.
        for (int i = 0; i < scoresSize - 1; i++) {
            int x0 = (i + 1) * (getWidth() - BORDER_GAP * 2) / (scoresSize - 1)
                + BORDER_GAP;
            int x1 = x0;
            int y0 = getHeight() - BORDER_GAP;
            int y1 = y0 - GRAPH_POINT_WIDTH;
            g2.drawLine(x0, y0, x1, y1);
            g2.drawString(String.valueOf(i + 1), x1, y1);
        }

        drawLine(g2, reputationPoints, REPUTATION_COLOR, "Репутація");
        drawOval(g2, reputationPoints, REPUTATION_POINT_COLOR);
        drawLine(g2, qualityPoints, QUALITY_COLOR, "Якість");
        drawOval(g2, qualityPoints, QUALITY_POINT_COLOR);
    }

    /**
```

```
     * Перетворює масив точок на лінії між ними.
     *
     * @param g2         Графічний компонент.
     * @param points     Масив точок.
     * @param graphColor Колір ліній.
     * @param text       Легенда графіка.
     */
    private void drawLine(Graphics2D g2, ArrayList<Point> points, Color
graphColor, String text) {
        g2.setColor(graphColor);
        g2.setStroke(GRAPH_STROKE);
        for (int i = 0; i < points.size() - 1; i++) {
            int x1 = points.get(i).x;
            int y1 = points.get(i).y;
            int x2 = points.get(i + 1).x;
            int y2 = points.get(i + 1).y;
            g2.drawLine(x1, y1, x2, y2);
            if (i == points.size() - 2) {
                g2.setColor(Color.black);
                g2.drawString(text, x1, y1);
            }
        }
    }

    /**
     * Переносить точки із масиву на графік.
     *
     * @param g2         Графічний компонент.
     * @param points     Масив точок.
     * @param pointColor Колір точок.
     */
    private void drawOval(Graphics2D g2, ArrayList<Point> points, Color
pointColor) {
        Stroke oldStroke = g2.getStroke();
        g2.setColor(pointColor);
        g2.setStroke(oldStroke);
        for (Point graphPoint : points) {
            int x = graphPoint.x - GRAPH_POINT_WIDTH / 2;
            int y = graphPoint.y - GRAPH_POINT_WIDTH / 2;
            int ovalW = GRAPH_POINT_WIDTH;
            int ovalH = GRAPH_POINT_WIDTH;
            g2.fillOval(x, y, ovalW, ovalH);
        }
    }

    @Override
    public Dimension getPreferredSize() {
        return new Dimension(PREF_W, PREF_H);
    }
}
```

Файл Settings.java основної програми

```
import javax.swing.*;
import java.io.FileWriter;
import java.io.IOException;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.NoSuchElementException;
import java.util.Scanner;

/**
 * Сховище всіх налаштувань програми.
 */
public class Settings {
    /**
     * Кількість нових об'єктів.
     */
    private int numberNewUsers;
    /**
     * Чи потрібно генерувати випадкові початкові значення для об'єктів.
     */
    private boolean isRandStat;
    /**
     * Мінімальна кількість нових контактів.
     */
    private int minCount;
    /**
     * Максимальна кількість нових контактів.
     */
    private int maxCount;
    /**
     * Ймовірність контакту.
     */
    private int contactProbability;
    /**
     * Чи зв'язок між об'єктами подвійний.
     */
    private boolean isDualDirection;
    /**
     * Унікальний номер об'єкта для виведення про нього інформації.
     */
    private int idToString;
    /**
     * Назва бази даних.
     */
    private String databaseName;
    /**
     * Назва таблиці в базі даних.
     */
    private String tableName;
    /**
     * Кількість допоміжних потоків для обчислень.
     */
    private int numOfThreads;
    /**
     * Середня границя репутації.
     */
    private int averageReputation;
    /**
     * Середня границя якості.
     */
    private int averageQuality;
    /**
     * Чи потрібно симулювати вибір об'єкта для зв'язку.
     */
    private boolean isSimulationSelecting;
    /**
     * Кількість об'єктів, з якими можна створювати контакти іншим об'єктам.
     * Значення 0 - без обмеження.
```

```
     */
    private int numMainUsers;

    /**
     * Конструктор класу.
     */
    public Settings() {
        // В першу чергу спробувати прочитати інформацію з файлу.
        try {
            Path path = Paths.get(SettingsForm.SETTINGS_FILE_NAME);
            Scanner scanner = new Scanner(path);
            this.numberNewUsers = Integer.parseInt(scanner.nextLine());
            this.isRandStat = Boolean.parseBoolean(scanner.nextLine());
            this.minCount = Integer.parseInt(scanner.nextLine());
            this.maxCount = Integer.parseInt(scanner.nextLine());
            this.contactProbability = Integer.parseInt(scanner.nextLine());
            this.isDualDirection = Boolean.parseBoolean(scanner.nextLine());
            this.idToString = Integer.parseInt(scanner.nextLine());
            this.databaseName = scanner.nextLine();
            this.tableName = scanner.nextLine();
            this.numOfThreads = Integer.parseInt(scanner.nextLine());
            this.averageReputation = Integer.parseInt(scanner.nextLine());
            this.averageQuality = Integer.parseInt(scanner.nextLine());
            this.isSimulationSelecting =
Boolean.parseBoolean(scanner.nextLine());
            this.numMainUsers = Integer.parseInt(scanner.nextLine());
        } catch (IOException | NoSuchElementException e) {
            // Інакше заповнити дані стандартними значеннями.
            this.numberNewUsers = 100;
            this.isRandStat = true;
            this.minCount = 10;
            this.maxCount = 50;
            this.contactProbability = 50;
            this.isDualDirection = true;
            this.idToString = 0;
            this.databaseName = "test.db";
            this.tableName = "users";
            this.numOfThreads = 6;
            this.averageReputation = 40;
            this.averageQuality = 60;
            this.isSimulationSelecting = true;
            this.numMainUsers = 0;
            JOptionPane.showMessageDialog(null,
                    "Файл налаштувань не знайдено! Встановлені налаштування за
замовчуванням.");
        }
    }

    /**
     * Зберігає інформацію про налаштування додатка до файлу.
     */
    public void settingsToFile() {
        try (FileWriter writer = new FileWriter(SettingsForm.SETTINGS_FILE_NAME,
false)) {
            writer.write(this.getNumberNewUsers() + "\n");
            writer.write(this.isRandStat() + "\n");
            writer.write(this.getMinCount() + "\n");
            writer.write(this.getMaxCount() + "\n");
            writer.write(this.getContactProbability() + "\n");
            writer.write(this.isDualDirection() + "\n");
            writer.write(this.getIdToString() + "\n");
            writer.write(this.getDatabaseName() + "\n");
            writer.write(this.getTableName() + "\n");
            writer.write(this.getNumOfThreads() + "\n");
            writer.write(this.getAverageReputation() + "\n");
            writer.write(this.getAverageQuality() + "\n");
            writer.write(this.isSimulationSelecting() + "\n");
            writer.write(this.getNumMainUsers() + "\n");
            writer.flush();
```

```
            } catch (IOException ex) {
                JOptionPane.showMessageDialog(null,
                        "Не вдалось зберегти налаштування до файлу!");
            }
        }

        /**
         * Повертає кількість допоміжних потоків для обчислень.
         *
         * @return Кількість допоміжних потоків.
         */
        public int getNumOfThreads() {
            return numOfThreads;
        }

        /**
         * Встановлює нове значення для кількості допоміжних потоків.
         *
         * @param numOfThreads Нове значення кількості допоміжних потоків.
         */
        public void setNumOfThreads(int numOfThreads) {
            this.numOfThreads = numOfThreads;
        }

        /**
         * Повертає кількість нових об'єктів.
         *
         * @return Кількість нових об'єктів.
         */
        public int getNumberNewUsers() {
            return numberNewUsers;
        }

        /**
         * Встановлює нове значення для кількості нових об'єктів.
         *
         * @param numberNewUsers Нове значення кількість нових об'єктів.
         */
        public void setNumberNewUsers(int numberNewUsers) {
            this.numberNewUsers = numberNewUsers;
        }

        /**
         * Повертає логічне значення, чи потрібно генерувати випадкові початкові
значення для об'єктів.
         *
         * @return Чи потрібно генерувати випадкові початкові значення для об'єктів.
         */
        public boolean isRandStat() {
            return isRandStat;
        }

        /**
         * Встановлює нове логічне значення чи потрібно генерувати випадкові
початкові значення для об'єктів.
         *
         * @param randStat Нове значення чи потрібно генерувати випадкові початкові
значення для об'єктів.
         */
        public void setRandStat(boolean randStat) {
            isRandStat = randStat;
        }

        /**
         * Повертає мінімальну кількість нових контактів.
         *
         * @return Мінімальна кількість нових контактів.
         */
        public int getMinCount() {
```

```
        return minCount;
    }

    /**
     * Встановлює нове значення для мінімальної кількості нових контактів
     *
     * @param minCount Нове значення мінімальної кількості нових контактів.
     */
    public void setMinCount(int minCount) {
        this.minCount = minCount;
    }

    /**
     * Повертає максимальну кількість нових контактів.
     *
     * @return Максимальна кількість нових контактів
     */
    public int getMaxCount() {
        return maxCount;
    }

    /**
     * Встановлює нове значення для максимальної кількості нових контактів.
     *
     * @param maxCount Нове значення максимальної кількості нових контактів.
     */
    public void setMaxCount(int maxCount) {
        this.maxCount = maxCount;
    }

    /**
     * Повертає ймовірність контакту.
     *
     * @return Ймовірність контакту.
     */
    public int getContactProbability() {
        return contactProbability;
    }

    /**
     * Встановлює нове значення для ймовірності контакту.
     *
     * @param contactProbability Нове значення ймовірності контакту.
     */
    public void setContactProbability(int contactProbability) {
        this.contactProbability = contactProbability;
    }

    /**
     * Повертає логічне значення, чи зв'язок між об'єктами подвійний.
     *
     * @return Чи зв'язок між об'єктами подвійний.
     */
    public boolean isDualDirection() {
        return isDualDirection;
    }

    /**
     * Встановлює нове логічне значення, чи зв'язок між об'єктами подвійний.
     *
     * @param dualDirection Нове логічне значення, чи зв'язок між об'єктами
подвійний.
     */
    public void setDualDirection(boolean dualDirection) {
        isDualDirection = dualDirection;
    }

    /**
     * Повертає унікальний номер об'єкта для виведення про нього інформації.
```

```
     *
     * @return Унікальний номер об'єкта.
     */
    public int getIdToString() {
        return idToString;
    }

    /**
     * Встановлює нове значення для унікального номера об'єкта для виведення про
нього інформації.
     *
     * @param idToString Нове значення унікального номера об'єкта.
     */
    public void setIdToString(int idToString) {
        this.idToString = idToString;
    }

    /**
     * Повертає назву бази даних.
     *
     * @return Назва бази даних.
     */
    public String getDatabaseName() {
        return databaseName;
    }

    /**
     * Встановлює нове значення для назви бази даних.
     *
     * @param databaseName Нове значення назви бази даних.
     */
    public void setDatabaseName(String databaseName) {
        this.databaseName = databaseName;
    }

    /**
     * Повертає назву таблиці в базі даних.
     *
     * @return Назва таблиці.
     */
    public String getTableName() {
        return tableName;
    }

    /**
     * Встановлює нове значення для таблиці в базі даних.
     *
     * @param tableName Нове значення таблиці.
     */
    public void setTableName(String tableName) {
        this.tableName = tableName;
    }

    /**
     * Повертає середню границю репутації.
     *
     * @return Середня границя репутації.
     */
    public int getAverageReputation() {
        return averageReputation;
    }

    /**
     * Встановлює нове значення для середньої границі репутації.
     *
     * @param averageReputation Нова середня границя репутації.
     */
    public void setAverageReputation(int averageReputation) {
        this.averageReputation = averageReputation;
```

```
    }

    /**
     * Повертає середню границю якості.
     *
     * @return Середня границя якості.
     */
    public int getAverageQuality() {
        return averageQuality;
    }

    /**
     * Встановлює нове значення для середньої границі якості.
     *
     * @param averageQuality Нова середня границя якості.
     */
    public void setAverageQuality(int averageQuality) {
        this.averageQuality = averageQuality;
    }

    /**
     * Повертає логічне значення, чи потрібно симулювати вибір об'єкта для
зв'язку.
     *
     * @return Чи потрібно симулювати вибір об'єкта для зв'язку.
     */
    public boolean isSimulationSelecting() {
        return isSimulationSelecting;
    }

    /**
     * Встановлює нове логічне значення, чи потрібно симулювати вибір об'єкта
для зв'язку.
     *
     * @param isSimulationSelecting Чи потрібно симулювати вибір об'єкта для
зв'язку.
     */
    public void setSimulationSelecting(boolean isSimulationSelecting) {
        this.isSimulationSelecting = isSimulationSelecting;
    }

    /**
     * Встановлює кількість об'єктів, з якими можна створювати контакти іншим
об'єктам
     *
     * @param numMainUsers Нове значення головних об'єктів.
     */
    public void setNumMainUsers(int numMainUsers) {
        this.numMainUsers = numMainUsers;
    }

    /**
     * Повертає кількість головних об'єктів.
     *
     * @return Кількість головних об'єктів
     */
    public int getNumMainUsers() {
        return this.numMainUsers;
    }
}
```

Файл LinksList.java основної програми

```
import javax.swing.*;
import java.util.TreeSet;

/**
 * Список контактів одного об'єкта.
 * Зберігає зв'язки першого, другого та третього порядку.
 */
public class LinksList {
    /**
     * Унікальний ідентифікатор об'єкта, для якого будується список контактів.
     */
    private final int startId;
    /**
     * Об'єкт моделі.
     */
    private final Model model;
    /**
     * Список контактів першого порядку.
     */
    private final TreeSet<Integer> firstOrder;
    /**
     * Список контактів другого порядку.
     */
    private final TreeSet<Integer> secondOrder;
    /**
     * Список контактів третього порядку.
     */
    private final TreeSet<Integer> thirdOrder;
    /**
     * Чи зміг клас отримати список зв'язків вказаного об'єкта.
     */
    private boolean isSuccess = true;

    /**
     * Конструктор класу.
     *
     * @param startId Унікальний ідентифікатор об'єкта.
     * @param model   Об'єкт моделі.
     */
    LinksList(int startId, Model model) {
        this.startId = startId;
        this.model = model;
        firstOrder = new TreeSet<>();
        secondOrder = new TreeSet<>();
        thirdOrder = new TreeSet<>();
        try {
            firstOrder.addAll(model.getUser(startId).getContactsIds());
            for (int id : firstOrder) {
                secondOrder.addAll(model.getUser(id).getContactsIds());
            }
            for (int id : secondOrder) {
                thirdOrder.addAll(model.getUser(id).getContactsIds());
            }

            // Видалення повторів.
            for (Integer id : firstOrder) {
                secondOrder.remove(id);
                thirdOrder.remove(id);
            }
            for (Integer id : secondOrder) {
                thirdOrder.remove(id);
            }
            firstOrder.remove(startId);
            secondOrder.remove(startId);
            thirdOrder.remove(startId);
        } catch (NullPointerException ex) {
            JOptionPane.showMessageDialog(null,
```

```
                "Не вдалось отримати об'єкт!");
            isSuccess = false;
        }
    }

    /**
     * Конструктор класу для знаходження зв'язків лише першого порядку.
     *
     * @param startId          Унікальний ідентифікатор об'єкта.
     * @param model            Об'єкт моделі.
     * @param isOnlyFirstOrder Без різниці яке значення.
     */
    LinksList(int startId, Model model, boolean isOnlyFirstOrder) {
        this.startId = startId;
        this.model = model;
        firstOrder = new TreeSet<>();
        secondOrder = null;
        thirdOrder = null;
        try {
            firstOrder.addAll(model.getUser(startId).getContactsIds());
            firstOrder.remove(startId);
        } catch (NullPointerException ex) {
            JOptionPane.showMessageDialog(null,
                    "Не вдалось отримати об'єкт!");
        }
    }

    /**
     * Повертає зв'язки першого порядку.
     *
     * @return Зв'язки першого порядку.
     */
    public TreeSet<Integer> getFirstOrder() {
        return firstOrder;
    }

    /**
     * Повертає кількість зв'язків першого порядку.
     *
     * @return Кількість зв'язків першого порядку.
     */
    public int getFirstOrderSize() {
        return firstOrder.size();
    }

    /**
     * Повертає зв'язки другого порядку.
     *
     * @return Зв'язки другого порядку.
     */
    public TreeSet<Integer> getSecondOrder() {
        return secondOrder;
    }

    /**
     * Повертає кількість зв'язків другого порядку.
     *
     * @return Кількість зв'язків другого порядку.
     */
    public int getSecondOrderSize() {
        return secondOrder.size();
    }

    /**
     * Повертає зв'язки третього порядку.
     *
     * @return Зв'язки третього порядку.
     */
    public TreeSet<Integer> getThirdOrder() {
```

```
        return thirdOrder;
    }

    /**
     * Повертає кількість зв'язків третього порядку.
     *
     * @return Кількість зв'язків третього порядку.
     */
    public int getThirdOrderSize() {
        return thirdOrder.size();
    }

    /**
     * Повертає унікальний ідентифікатор об'єкта, для якого будується список
контактів.
     *
     * @return Унікальний ідентифікатор об'єкта.
     */
    public int getStartId() {
        return startId;
    }

    /**
     * Повертає об'єкт моделі.
     *
     * @return Об'єкт моделі.
     */
    public Model getModel() {
        return model;
    }

    /**
     * Чи зміг клас отримати список зв'язків вказаного об'єкта.
     *
     * @return Чи все добре.
     */
    public boolean isSuccess() {
        return isSuccess;
    }
}
```

Файл Main.java основної програми

```
/**
 * Початкова точка виконання додатка.
 */
public class Main {
    public static void main(String[] args) {
        MainForm mainForm = new MainForm();
        mainForm.setVisible(true);
    }
}
```

Файл SettingsForm.java основної програми

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

/**
 * Вікно налаштувань.
 */
public class SettingsForm extends JFrame {
    /**
     * Назва файлу, який призначений для зберігання налаштувань додатку.
     */
    public static final String SETTINGS_FILE_NAME = "setting.txt";
    /**
     * Основний контейнер графічних елементів.
     */
    private JPanel panel;
    /**
     * Основне вікно додатка.
     * Використовується для зворотної взаємодії між вікнами.
     */
    private final MainForm mainForm;
    /**
     * Вікно налаштувань.
     */
    private final SettingsForm settingsForm;
    /**
     * Об'єкт, який зберігає налаштування додатка під час його виконання.
     */
    private final Settings settings;

    // Кнопки, що містяться на графічному інтерфейсі користувача.
    // Парні елементи об'єднані в граппи.
    private JButton acceptBtn, cancelBtn;
    private JTextField numberNewUsersTextField;
    private JCheckBox isRandStatCheckBox;
    private JTextField minCountTextField, maxCountTextField;
    private JTextField contactProbabilityTextField;
    private JCheckBox isDualDirectionCheckBox;
    private JTextField idToStringTextField;
    private JTextField databaseNameTextField, tableNameTextField;
    private JTextField numOfThreadsTextField;
    private JTextField averageReputationField, averageQualityField;
    private JCheckBox isSimulationSelectingCheckBox;
    private JTextField numMainUsersField;

    /**
     * Конструктор вікна налаштувань.
     */
    SettingsForm(MainForm mainForm, Settings settings) {
        // Початкова ініціалізація всіх об'єктів.
        super("Налаштування");
        this.mainForm = mainForm;
        this.settings = settings;
        this.settingsForm = this;

        // Налаштування вікна.
        this.setContentPane(panel);
        JDialog dialog = new JDialog(this);
        dialog.setDefaultCloseOperation(JDialog.DISPOSE_ON_CLOSE);
        this.setSize(800, 500);
        Toolkit toolkit = Toolkit.getDefaultToolkit();
        Dimension dimension = toolkit.getScreenSize();
        this.setLocation(dimension.width / 2 - this.getWidth() / 2,
            dimension.height / 2 - this.getHeight() / 2);
        addCloseEvent(this);
```

```
        // Перенесення інформації зі сховища (об'єкту) налаштувань до графічного
інтерфейсу.
numberNewUsersTextField.setText(String.valueOf(settings.getNumberNewUsers()));
        isRandStatCheckBox.setSelected(settings.isRandStat());
        minCountTextField.setText(String.valueOf(settings.getMinCount()));
        maxCountTextField.setText(String.valueOf(settings.getMaxCount()));
contactProbabilityTextField.setText(String.valueOf(settings.getContactProbabilit
y()));
        isDualDirectionCheckBox.setSelected(settings.isDualDirection());
        idToStringTextField.setText(String.valueOf(settings.getIdToString()));
databaseNameTextField.setText(String.valueOf(settings.getDatabaseName()));
        tableNameTextField.setText(String.valueOf(settings.getTableName()));
numOfThreadsTextField.setText(String.valueOf(settings.getNumOfThreads()));
averageReputationField.setText(String.valueOf(settings.getAverageReputation()));
averageQualityField.setText(String.valueOf(settings.getAverageQuality()));
isSimulationSelectingCheckBox.setSelected(settings.isSimulationSelecting());
        numMainUsersField.setText(String.valueOf(settings.getNumMainUsers()));

        // Початкова ініціалізація кнопок.
        acceptBtnSetup();
        cancelBtnSetup();
    }

    /**
     * Метод виконується при натисненні на кнопку acceptBtn.
     * Інформацію з текстового інтерфейсу переносить до сховища (об'єкту)
налаштувань.
     */
    private void acceptBtnSetup() {
        acceptBtn.addActionListener(e -> {
            try {
settings.setNumberNewUsers(Integer.parseInt(numberNewUsersTextField.getText()));
                settings.setRandStat(isRandStatCheckBox.isSelected());
settings.setMinCount(Integer.parseInt(minCountTextField.getText()));
settings.setMaxCount(Integer.parseInt(maxCountTextField.getText()));
settings.setContactProbability(Integer.parseInt(contactProbabilityTextField.getT
ext()));
                settings.setDualDirection(isDualDirectionCheckBox.isSelected());
settings.setIdToString(Integer.parseInt(idToStringTextField.getText()));
                settings.setDatabaseName(databaseNameTextField.getText());
                settings.setTableName(tableNameTextField.getText());
settings.setNumOfThreads(Integer.parseInt(numOfThreadsTextField.getText()));
settings.setAverageReputation(Integer.parseInt(averageReputationField.getText())
);
settings.setAverageQuality(Integer.parseInt(averageQualityField.getText()));
settings.setSimulationSelecting(isSimulationSelectingCheckBox.isSelected());
settings.setNumMainUsers(Integer.parseInt(numMainUsersField.getText()));

                mainForm.changeButtonsStatus(true);
                settingsForm.setVisible(false);
                JOptionPane.showMessageDialog(null,
                        "Налаштування збережено.");
            } catch (NumberFormatException ex) {
                JOptionPane.showMessageDialog(null,
                        "Помилка введення даних! Налаштування не збережено.");
                mainForm.changeButtonsStatus(true);
                settingsForm.setVisible(false);
            }
        });
    }

    /**
     * Метод виконується при натисненні на кнопку cancelBtn.
     * Скасовує прийняті зміни.
     */
    private void cancelBtnSetup() {
        cancelBtn.addActionListener(e -> {
            mainForm.changeButtonsStatus(true);
            settingsForm.setVisible(false);
```

```
                JOptionPane.showMessageDialog(null,
                        "Налаштування не збережено.");
            });
        }

        /**
         * Спрацьовує при закритті вікна налаштувань.
         * Повертає активність кнопкам головного вікна.
         *
         * @param frame Вікно налаштувань.
         */
        private void addCloseEvent(JFrame frame) {
            frame.addWindowListener(new WindowAdapter() {
                @Override
                public void windowClosing(WindowEvent e) {
                    mainForm.changeButtonsStatus(true);
                }
            });
        }
    }
```

Файл MainForm.java основної програми

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.util.LinkedList;

/**
 * Головне вікно програми.
 */
public class MainForm extends JFrame {
    /**
     * Ім'я головного потоку моделювання.
     */
    public static String MAIN_THREAD_NAME = "Model0";
    /**
     * Основний контейнер графічних елементів.
     */
    private JPanel panel;
    /**
     * Панель з графіками.
     */
    private JPanel graphPanel;
    /**
     * Панель з графом.
     */
    private JPanel jungPanel;

    /**
     * Об'єкт моделі.
     * Відповідає за процеси та дані роботи моделі.
     * Виконується окремим потоком \ потоками.
     */
    private Model model;
    /**
     * Об'єкт, який зберігає налаштування додатка під час його виконання.
     */
    private final Settings settings;
    /**
     * Об'єкт, який зберігає статистичні дані.
     */
    private Statistics statistics;
    /**
     * Головне вікно.
     */
    private final MainForm mainForm;
    /**
     * Об'єкт відповідає за графік.
     */
    public DrawGraph drawGraph;

    /**
     * Кнопка додавання нових об'єктів в модель.
     */
    private JButton addUsersBtn;
    /**
     * Кнопка створення нових контактів між об'єктами.
     */
    private JButton createContactsBtn;
    /**
     * Кнопка перебору зв'язків об'єктів для моделювання їх повторного контакту.
     */
    private JButton enumerateContactsBtn;
    /**
     * Кнопка оновлення статистики.
     */
    private JButton printStatisticsBtn;
    /**
     * Кнопка виведення інформації про вказаний об'єкт.
```

```
 */
private JButton printUserBtn;
/**
 * Кнопка виведення інформації про всі об'єкти.
 */
private JButton printUsersBtn;
/**
 * Кнопка виведення інформації про зв'язки обраного об'єкта.
 */
private JButton linksListBtn;
/**
 * Кнопка очищення внутрішніх даних.
 */
private JButton clearBtn;
/**
 * Кнопка завантаження даних з зовнішньої бази даних.
 */
private JButton loadFromFileBtn;
/**
 * Кнопка збереження даних до зовнішньої бази даних.
 */
private JButton saveToFileBtn;
/**
 * Кнопка для відкриття налаштувань додатка.
 */
private JButton settingsBtn;

// Текстові поля для виводу інформації. Парні елементи об'єднані в граппи.
private JTextField numUsersField;
private JTextField numContactsField;
private JTextField minReputationField, maxReputationField;
private JTextField minPopularityField, maxPopularityField;
private JTextField minQualityField, maxQualityField;
private JTextField minNumSuccessContactsField, maxNumSuccessContactsField;
private JButton minReputationButton, maxReputationButton;
private JButton minPopularityButton, maxPopularityButton;
private JButton minQualityButton, maxQualityButton;
private JButton minNumSuccessContactsButton, maxNumSuccessContactsButton;
private JTextField numLowReputationField, numHighReputationField;
private JTextField numLowQualityField, numHighQualityField;
/**
 * Список всіх кнопок для зручної взаємодії відразу з усіма кнопками.
 */
private final LinkedList<JButton> buttons;

/**
 * Конструктор головного вікна додатка.
 */
MainForm() {
    // Початкова ініціалізація всіх об'єктів.
    super("Комп'ютерна модель репутаційної соціальної мережі");
    mainForm = this;
    buttons = new LinkedList<>();
    settings = new Settings();
    model = new Model(0, mainForm, settings);
    model.start();
    statistics = new Statistics(settings);

    // Налаштування головного вікна.
    this.setContentPane(panel);
    this.setDefaultCloseOperation(WindowConstants.DO_NOTHING_ON_CLOSE);
    this.setSize(800, 500);
    Toolkit toolkit = Toolkit.getDefaultToolkit();
    Dimension dimension = toolkit.getScreenSize();
    this.setLocation(dimension.width / 2 - this.getWidth() / 2,
        dimension.height / 2 - this.getHeight() / 2);
    this.setExtendedState(JFrame.MAXIMIZED_BOTH);
    addCloseEvent(this);
```

```
// Початкова ініціалізація всіх графічних об'єктів.
addUsersBtnSetup();
createContactsBtnSetup();
enumerateContactsBtnSetup();
printStatisticsBtnSetup();
printUserBtnSetup();
printUsersBtnSetup();
linksListBtnSetup();
clearBtnSetup();
loadFromFileBtnSetup();
saveToFileBtnSetup();
settingsBtnSetup();

createAndShowGraphics();
minReputationButton.addActionListener(e -> {
    changeButtonsStatus(false);
    UserInfoForm userInfo = new UserInfoForm(mainForm,
        statistics.getMinReputationUser());
    userInfo.setVisible(true);
});
buttons.add(minReputationButton);
maxReputationButton.addActionListener(e -> {
    changeButtonsStatus(false);
    UserInfoForm userInfo = new UserInfoForm(mainForm,
        statistics.getMaxReputationUser());
    userInfo.setVisible(true);
});
buttons.add(maxReputationButton);
minPopularityButton.addActionListener(e -> {
    changeButtonsStatus(false);
    UserInfoForm userInfo = new UserInfoForm(mainForm,
        statistics.getMinPopularityUser());
    userInfo.setVisible(true);
});
buttons.add(minPopularityButton);
maxPopularityButton.addActionListener(e -> {
    changeButtonsStatus(false);
    UserInfoForm userInfo = new UserInfoForm(mainForm,
        statistics.getMaxPopularityUser());
    userInfo.setVisible(true);
});
buttons.add(maxPopularityButton);
minQualityButton.addActionListener(e -> {
    changeButtonsStatus(false);
    UserInfoForm userInfo = new UserInfoForm(mainForm,
        statistics.getMinQualityUser());
    userInfo.setVisible(true);
});
buttons.add(minQualityButton);
maxQualityButton.addActionListener(e -> {
    changeButtonsStatus(false);
    UserInfoForm userInfo = new UserInfoForm(mainForm,
        statistics.getMaxQualityUser());
    userInfo.setVisible(true);
});
buttons.add(maxQualityButton);
minNumSuccessContactsButton.addActionListener(e -> {
    changeButtonsStatus(false);
    UserInfoForm userInfo = new UserInfoForm(mainForm,
        statistics.getMinNumSuccessContactsUser());
    userInfo.setVisible(true);
});
buttons.add(minNumSuccessContactsButton);
maxNumSuccessContactsButton.addActionListener(e -> {
    changeButtonsStatus(false);
    UserInfoForm userInfo = new UserInfoForm(mainForm,
        statistics.getMaxNumSuccessContactsUser());
    userInfo.setVisible(true);
});
```

```
        buttons.add(maxNumSuccessContactsButton);
    }

    private void createAndShowGraphics() {
        this.drawGraph = new DrawGraph();
        graphPanel.setLayout(new BorderLayout());
        graphPanel.add(this.drawGraph, BorderLayout.CENTER);
    }

    /**
     * Налаштування процесу закриття додатку.
     *
     * @param frame Головне вікно.
     */
    private void addCloseEvent(JFrame frame) {
        frame.addWindowListener(new WindowAdapter() {
            @Override
            public void windowClosing(WindowEvent e) {
                if (JOptionPane.showConfirmDialog(frame,
                  "Ви впевнені, що бажаєте закрити це вікно?", "Закрити вікно?",
                      JOptionPane.YES_NO_OPTION,
                      JOptionPane.QUESTION_MESSAGE) == JOptionPane.YES_OPTION) {
                    model.isEnd = true;
                    settings.settingsToFile();
                    System.exit(0);
                }
            }
        });
    }

    /**
     * Налаштування та опрацювання натиску кнопки addUsersBtn.
     * Вказує окремому головному потоку моделювання, що потрібно додати нові
об'єкти.
     */
    private void addUsersBtnSetup() {
        addUsersBtn.addActionListener(e -> {
            changeButtonsStatus(false);
            model.isAddUsers = true;
        });
        buttons.add(addUsersBtn);
    }

    /**
     * Налаштування та опрацювання натиску кнопки createContactsBtn.
     * Вказує окремому головному потоку моделювання, що потрібно створити нові
контакти.
     */
    private void createContactsBtnSetup() {
        createContactsBtn.addActionListener(e -> {
            changeButtonsStatus(false);
            model.isCreateContacts = true;
        });
        buttons.add(createContactsBtn);
    }

    /**
     * Налаштування та опрацювання натиску кнопки enumerateContactsBtn.
     * Вказує окремому головному потоку моделювання, що потрібно перебрати
об'єкти та перебрати контакти.
     */
    private void enumerateContactsBtnSetup() {
        enumerateContactsBtn.addActionListener(e -> {
            changeButtonsStatus(false);
            model.isEnumerateContacts = true;
        });
        buttons.add(enumerateContactsBtn);
    }
```

```
    /**
     * Налаштування та опрацювання натиску кнопки printStatisticsBtn.
     * Вказує окремому головному потоку моделювання, що потрібно оновити
статистику.
     */
    private void printStatisticsBtnSetup() {
        printStatisticsBtn.addActionListener(e -> {
            changeButtonsStatus(false);
            model.isGetStatistics = true;
        });
        buttons.add(printStatisticsBtn);
    }

    /**
     * Налаштування та опрацювання натиску кнопки printUserBtn.
     * Вказує окремому головному потоку моделювання, що потрібно вивести
інформацію про об'єкт.
     */
    private void printUserBtnSetup() {
        printUserBtn.addActionListener(e -> {
            changeButtonsStatus(false);
            UserInfoForm userInfo = new UserInfoForm(mainForm,
                model.getUser(settings.getIdToString()));
            userInfo.setVisible(true);
        });
        buttons.add(printUserBtn);
    }

    /**
     * Налаштування та опрацювання натиску кнопки printUsersBtn.
     * Вказує окремому головному потоку моделювання, що потрібно вивести
інформацію всі об'єкти.
     */
    private void printUsersBtnSetup() {
        printUsersBtn.addActionListener(e -> {
            changeButtonsStatus(false);
            model.isPrintUsers = true;
        });
        buttons.add(printUsersBtn);
    }

    /**
     * Налаштування та опрацювання натиску кнопки printUsersBtn.
     * Вказує окремому головному потоку моделювання, що потрібно вивести
інформацію всі об'єкти.
     */
    private void linksListBtnSetup() {
        linksListBtn.addActionListener(e -> {
            changeButtonsStatus(false);
            model.isLinksList = true;
        });
        buttons.add(linksListBtn);
    }

    /**
     * Налаштування та опрацювання натиску кнопки printUsersBtn.
     * Вказує окремому головному потоку моделювання, що потрібно вивести
інформацію всі об'єкти.
     */
    private void clearBtnSetup() {
        clearBtn.addActionListener(e -> {
            User.resetCount();
            model = new Model(0, mainForm, settings);
            model.start();
            JOptionPane.showMessageDialog(null,
                    "Очищення даних моделі завершено.");
        });
        buttons.add(clearBtn);
    }
```

```
    /**
     * Налаштування та опрацювання натиску кнопки loadFromFileBtn.
     * Вказує окремому головному потоку моделювання, що потрібно завантажити
дані з бази даних.
     */
    private void loadFromFileBtnSetup() {
        loadFromFileBtn.addActionListener(e -> {
            changeButtonsStatus(false);
            model.isLoadFromFile = true;
        });
        buttons.add(loadFromFileBtn);
    }

    /**
     * Налаштування та опрацювання натиску кнопки saveToFileBtn.
     * Вказує окремому головному потоку моделювання, що потрібно зберегти дані
до бази даних.
     */
    private void saveToFileBtnSetup() {
        saveToFileBtn.addActionListener(e -> {
            changeButtonsStatus(false);
            model.isSaveToFile = true;
        });
        buttons.add(saveToFileBtn);
    }

    /**
     * Налаштування та опрацювання натиску кнопки settingsBtn.
     * Відкрити вікно налаштувань.
     */
    private void settingsBtnSetup() {
        MainForm mainForm = this;
        settingsBtn.addActionListener(e -> {
            SettingsForm settingsForm = new SettingsForm(mainForm, settings);
            settingsForm.setVisible(true);
            changeButtonsStatus(false);
        });
        buttons.add(settingsBtn);
    }

    //-----------------------------------------------

    /**
     * Друк статистики.
     *
     * @param statistics Об'єкт, що зберігає статистику.
     */
    public boolean printStatistics(Statistics statistics) {
        this.statistics = statistics;
        try {
            numUsersField.setText(String.valueOf(statistics.getNumUsers()));
numContactsField.setText(String.valueOf(statistics.getNumContacts()));
minReputationField.setText(String.valueOf(statistics.getMinReputation()));
maxReputationField.setText(String.valueOf(statistics.getMaxReputation()));
minPopularityField.setText(String.valueOf(statistics.getMinPopularity()));
maxPopularityField.setText(String.valueOf(statistics.getMaxPopularity()));
            minQualityField.setText(String.valueOf(statistics.getMinQuality()));
            maxQualityField.setText(String.valueOf(statistics.getMaxQuality()));
minNumSuccessContactsField.setText(String.valueOf(statistics.getMinNumSuccessCon
tacts()));
maxNumSuccessContactsField.setText(String.valueOf(statistics.getMaxNumSuccessCon
tacts()));
numLowReputationField.setText(String.valueOf(statistics.getNumLowReputation()));
numHighReputationField.setText(String.valueOf(statistics.getNumHighReputation())
);
numLowQualityField.setText(String.valueOf(statistics.getNumLowQuality()));
numHighQualityField.setText(String.valueOf(statistics.getNumHighQuality()));
            return true;
```

```
            } catch (NullPointerException ignored) {
            }
            return false;
        }

        /**
         * Змінює стан всіх кнопок з графічного інтерфейсу користувача.
         *
         * @param isEnabled Чи дозволена взаємодія з кнопками.
         */
        public void changeButtonsStatus(boolean isEnabled) {
            for (JButton button : buttons) {
                button.setEnabled(isEnabled);
            }
        }
    }
```

Файл SQLite.java основної програми

```
import javax.swing.*;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.HashMap;

/**
 * Статичний клас надає функціональний інтерфейс для взаємодії з зовнішньою
базою даних.
 * Реалізовує наступні операції:
 * 1) підключення до БД;
 * 2) створення таблиці;
 * 3) додавання даних до таблиці;
 * 4) отримання даних з таблиці;
 * 5) оновлення даних таблиці;
 * 6) видалення даних з таблиці;
 */
public class SQLite {

    /**
     * Встановлює зв'язок з зовнішньою базою даних.
     *
     * @param databaseName Назва зовнішньої бази даних.
     * @return Зв'язок з зовнішньою базою даних.
     */
    public static synchronized Connection connect(String databaseName) {
        Connection c = null;

        try {
            c = DriverManager.getConnection("jdbc:sqlite:" + databaseName);
        } catch (Exception e) {
            JOptionPane.showMessageDialog(null,
                    "Не вдалось під'єднатися до бази даних!");
            System.out.println(e);
        }
        return c;
    }

    /**
     * Створює таблицю в базі даних.
     *
     * @param databaseName Назва зовнішньої бази даних.
     * @param tableName    Назва таблиці.
     */
    public static synchronized void createTable(String databaseName, String
tableName) {
        Connection c;
        Statement stmt;

        try {
            c = connect(databaseName);
            stmt = c.createStatement();
            String sql = String.format("CREATE TABLE IF NOT EXISTS %s ",
                tableName) +
                    "(id INTEGER PRIMARY KEY NOT NULL, " +
                    "reputation INTEGER, " +
                    "popularity INTEGER, " +
                    "quality INTEGER, " +
                    "qualityBonus INTEGER, " +
                    "qualityDebuff INTEGER, " +
                    "numOfPoints INTEGER, " +
                    "numSuccessContacts INTEGER, " +
                    "contacts TEXT)";
            stmt.executeUpdate(sql);
            stmt.close();
            c.close();
```

```
        } catch (Exception e) {
            JOptionPane.showMessageDialog(null,
                    "Не вдалось створити таблицю в базі даних!");
            System.out.println(e);
        }
    }

    /**
     * Зберігає стан внутрішньої бази даних до зовнішньої бази даних.
     *
     * @param databaseName Назва зовнішньої бази даних.
     * @param tableName    Назва таблиці.
     * @param database     Об'єкт внутрішньої бази даних.
     */
    public static synchronized void insertIntoTable(String databaseName, String
tableName, final HashMap<Integer, User> database) {
        Connection c;
        Statement stmt;

        try {
            c = connect(databaseName);
            c.setAutoCommit(false);
            stmt = c.createStatement();
            for (User user : database.values()) {
                String sql = String.format("INSERT INTO %s
(ID,reputation,popularity,quality,qualityBonus,qualityDebuff,numOfPoints,numSucc
essContacts,contacts) ",
                 tableName) +
                 String.format("VALUES (%d, %d, %d, %d, %d, %d, %d, %d, '%s');",
                 user.getID(), user.getReputation(), user.getPopularity(),
                 user.getDefaultQuality(), user.getQualityBonus(),
                 user.getQualityDebuff(), user.getNumOfPoints(),
                 user.getNumSuccessContacts(), user.contactsToString());
                stmt.executeUpdate(sql);
            }

            stmt.close();
            c.commit();
            c.close();
        } catch (Exception e) {
            JOptionPane.showMessageDialog(null,
                    "Не вдалось записати інформацію до бази даних!");
            System.out.println(e);
        }
    }

    /**
     * Наповнює внутрішню базу даних інформацією з зовнішньої бази даних.
     *
     * @param databaseName Назва зовнішньої бази даних.
     * @param tableName    Назва таблиці.
     * @return Об'єкт внутрішньої бази даних.
     */
    public static synchronized HashMap<Integer, User> selectFromTable(String
databaseName, String tableName) {
        Connection c;
        Statement stmt;
        HashMap<Integer, User> database = new HashMap<>();
        try {
            c = connect(databaseName);
            c.setAutoCommit(false);

            stmt = c.createStatement();
            ResultSet rs = stmt.executeQuery(String.format("SELECT * FROM %s;",
                tableName));
            while (rs.next()) {
                int id = rs.getInt("id");
                User user = new User(id,
                        rs.getInt("reputation"),
```

```
                        rs.getInt("popularity"),
                        rs.getInt("quality"),
                        rs.getInt("qualityBonus"),
                        rs.getInt("qualityDebuff"),
                        rs.getInt("numOfPoints"),
                        rs.getInt("numSuccessContacts"),
                        rs.getString("contacts"));
                database.put(id, user);
            }
            rs.close();
            stmt.close();
            c.close();
        } catch (Exception e) {
            JOptionPane.showMessageDialog(null,
                    "Не вдалось отримати інформацію із бази даних!");
            System.out.println(e);
        }
        return database;
    }

    /**
     * Оновлює інформацію в зовнішній базі даних.
     *
     * @param databaseName Назва зовнішньої бази даних.
     * @param tableName    Назва таблиці.
     */
    public static synchronized void updateTable(String databaseName, String
tableName) {
        Connection c;
        Statement stmt;

        try {
            c = connect(databaseName);
            c.setAutoCommit(false);

            stmt = c.createStatement();
            String sql = String.format("UPDATE %s set popularity = 250 where
ID=1;", tableName);
            stmt.executeUpdate(sql);
            c.commit();

            ResultSet rs = stmt.executeQuery(String.format("SELECT * FROM %s;",
                tableName));

            while (rs.next()) {
                int id = rs.getInt("id");
                int reputation = rs.getInt("reputation");
                int popularity = rs.getInt("popularity");
                int quality = rs.getInt("quality");
                int qualityBonus = rs.getInt("qualityBonus");
                int qualityDebuff = rs.getInt("qualityDebuff");
                int numOfPoints = rs.getInt("numOfPoints");
                int numSuccessContacts = rs.getInt("numSuccessContacts");
                String contacts = rs.getString("contacts");

                System.out.println("id = " + id);
                System.out.println("reputation = " + reputation);
                System.out.println("popularity = " + popularity);
                System.out.println("quality = " + quality);
                System.out.println("qualityBonus = " + qualityBonus);
                System.out.println("qualityDebuff = " + qualityDebuff);
                System.out.println("numOfPoints = " + numOfPoints);
                System.out.println("numSuccessContacts = " +
                    numSuccessContacts);
                System.out.println("contacts = " + contacts);
                System.out.println();
            }
            rs.close();
            stmt.close();
```

```
            c.close();
        } catch (Exception e) {
            JOptionPane.showMessageDialog(null,
                    "Не вдалось оновити інформацію в базі даних!");
        }
    }

    /**
     * Очищає інформацію в таблиці із зовнішньої бази даних.
     *
     * @param databaseName Назва зовнішньої бази даних.
     * @param tableName    Назва таблиці.
     */
    public static synchronized void deleteFromTable(String databaseName, String
tableName) {
        Connection c;
        Statement stmt;

        try {
            c = connect(databaseName);
            c.setAutoCommit(false);

            stmt = c.createStatement();
            String sql = String.format("DELETE from %s;", tableName);
            stmt.executeUpdate(sql);
            c.commit();

            stmt.close();
            c.close();
        } catch (Exception e) {
            JOptionPane.showMessageDialog(null,
                    "Не вдалось очистити інформацію і базі даних!");
            System.out.println(e);
        }
    }
}
```

Файл Statistics.java основної програми

```
/**
 * Сховище всієї статистичної інформації.
 * Більшість інформації зберігається в окремих об'єктах з моделі.
 */
public class Statistics {
    /**
     * Об'єкт, який зберігає налаштування додатка під час його виконання.
     */
    private final Settings settings;
    /**
     * Об'єкт з мінімальною репутацією.
     */
    private User minReputationUser;
    /**
     * Об'єкт з максимальною репутацією.
     */
    private User maxReputationUser;
    /**
     * Об'єкт з мінімальною популярністю.
     */
    private User minPopularityUser;
    /**
     * Об'єкт з максимальною популярністю.
     */
    private User maxPopularityUser;
    /**
     * Об'єкт з мінімальною якістю взаємодії.
     */
    private User minQualityUser;
    /**
     * Об'єкт з максимальною якістю взаємодії.
     */
    private User maxQualityUser;
    /**
     * Об'єкт з мінімальною кількістю успішних взаємодій.
     */
    private User minNumSuccessContactsUser;
    /**
     * Об'єкт з максимальною кількістю успішних взаємодій.
     */
    private User maxNumSuccessContactsUser;
    /**
     * Загальна кількість контактів між об'єктами.
     */
    private int numContacts;
    /**
     * Загальна кількість об'єктів в моделі.
     */
    private int numUsers;
    /**
     * Загальна кількість об'єктів з низькою репутацією.
     */
    private int numLowReputation;
    /**
     * Загальна кількість об'єктів з високою репутацією.
     */
    private int numHighReputation;
    /**
     * Загальна кількість об'єктів з низькою якістю.
     */
    private int numLowQuality;
    /**
     * Загальна кількість об'єктів з високою якістю.
     */
    private int numHighQuality;

    /**
```

```
     * Конструктор класу.
     */
    Statistics(Settings settings) {
        this.settings = settings;
        this.minReputationUser = null;
        this.maxReputationUser = null;
        this.minPopularityUser = null;
        this.maxPopularityUser = null;
        this.minQualityUser = null;
        this.maxQualityUser = null;
        this.minNumSuccessContactsUser = null;
        this.maxNumSuccessContactsUser = null;
        this.numContacts = 0;
        this.numUsers = 0;
    }

    //---Блок get- is- та getObject методів MinReputation---

    /**
     * Повертає значення мінімальної репутації об'єкта, що присутній в моделі.
     *
     * @return Мінімальна репутація.
     */
    public int getMinReputation() {
        return this.minReputationUser.getReputation();
    }

    /**
     * Перевіряє значення мінімальної репутації об'єкта.
     * Запам'ятовує його, якщо вхідне значення мінімальне.
     *
     * @param user Значення репутації для перевірки на мінімальність.
     */
    public void isMinReputation(User user) {
        if (this.minReputationUser == null) {
            this.minReputationUser = user;
        } else if (this.minReputationUser.getReputation() >
user.getReputation()) {
            this.minReputationUser = user;
        }
    }

    /**
     * Повертає об'єкт з мінімальною репутацією.
     *
     * @return Об'єкт з мінімальною репутацією.
     */
    public User getMinReputationUser() {
        return minReputationUser;
    }

    //---Блок get- is- та getObject методів MaxReputation---

    public int getMaxReputation() {
        return this.maxReputationUser.getReputation();
    }

    public void isMaxReputation(User user) {
        if (this.maxReputationUser == null) {
            this.maxReputationUser = user;
        } else if (this.maxReputationUser.getReputation() <
user.getReputation()) {
            this.maxReputationUser = user;
        }
    }

    public User getMaxReputationUser() {
        return maxReputationUser;
    }
```

```
    //---Блок get- is- та getObject методів MinPopularity---

    public int getMinPopularity() {
        return this.minPopularityUser.getPopularity();
    }

    public void isMinPopularity(User user) {
        if (this.minPopularityUser == null) {
            this.minPopularityUser = user;
        } else if (this.minPopularityUser.getPopularity() >
user.getPopularity()) {
            this.minPopularityUser = user;
        }
    }

    public User getMinPopularityUser() {
        return minPopularityUser;
    }

    //---Блок get- is- та getObject методів MaxPopularity---

    public int getMaxPopularity() {
        return this.maxPopularityUser.getPopularity();
    }

    public void isMaxPopularity(User user) {
        if (this.maxPopularityUser == null) {
            this.maxPopularityUser = user;
        } else if (this.maxPopularityUser.getPopularity() <
user.getPopularity()) {
            this.maxPopularityUser = user;
        }
    }

    public User getMaxPopularityUser() {
        return maxPopularityUser;
    }

    //---Блок get- is- та getObject методів MinQuality---

    public int getMinQuality() {
        return this.minQualityUser.getQuality();
    }

    public void isMinQuality(User user) {
        if (this.minQualityUser == null) {
            this.minQualityUser = user;
        } else if (this.minQualityUser.getQuality() > user.getQuality()) {
            this.minQualityUser = user;
        }
    }

    public User getMinQualityUser() {
        return minQualityUser;
    }

    //---Блок get- is- та getObject методів MaxQuality---

    public int getMaxQuality() {
        return this.maxQualityUser.getQuality();
    }

    public void isMaxQuality(User user) {
        if (this.maxQualityUser == null) {
            this.maxQualityUser = user;
        } else if (this.maxQualityUser.getQuality() < user.getQuality()) {
            this.maxQualityUser = user;
        }
```

```
    }

    public User getMaxQualityUser() {
        return maxQualityUser;
    }

    //---Блок get- is- та getObject методів MinNumSuccessContacts---

    public int getMinNumSuccessContacts() {
        return this.minNumSuccessContactsUser.getNumSuccessContacts();
    }

    public void isMinNumSuccessContacts(User user) {
        if (this.minNumSuccessContactsUser == null) {
            this.minNumSuccessContactsUser = user;
        } else if (this.minNumSuccessContactsUser.getNumSuccessContacts() >
user.getNumSuccessContacts()) {
            this.minNumSuccessContactsUser = user;
        }
    }

    public User getMinNumSuccessContactsUser() {
        return minNumSuccessContactsUser;
    }

    //---Блок get- is- та getObject методів MaxNumSuccessContacts---

    public int getMaxNumSuccessContacts() {
        return this.maxNumSuccessContactsUser.getNumSuccessContacts();
    }

    public void isMaxNumSuccessContacts(User user) {
        if (this.maxNumSuccessContactsUser == null) {
            this.maxNumSuccessContactsUser = user;
        } else if (this.maxNumSuccessContactsUser.getNumSuccessContacts() <
user.getNumSuccessContacts()) {
            this.maxNumSuccessContactsUser = user;
        }
    }

    public User getMaxNumSuccessContactsUser() {
        return maxNumSuccessContactsUser;
    }

    //---Блок get- та set- методів NumContacts---

    public int getNumContacts() {
        return numContacts;
    }

    public void setNumContacts(int numContacts) {
        this.numContacts = numContacts;
    }

    //---Блок get- та set- методів NumUsers---

    public int getNumUsers() {
        return numUsers;
    }

    public void setNumUsers(int size) {
        this.numUsers = size;
    }

    //---Блок get- та is- методів Reputation---


    public int getNumLowReputation() {
        return numLowReputation;
```

```
    }

    public int getNumHighReputation() {
        return numHighReputation;
    }

    public void isAverageReputation(User user) {
        if (user.getReputation() < settings.getAverageReputation()) {
            this.numLowReputation++;
        } else {
            this.numHighReputation++;
        }
    }

    //---Блок get- та is- методів Quality---


    public int getNumLowQuality() {
        return numLowQuality;
    }

    public int getNumHighQuality() {
        return numHighQuality;
    }

    public void isAverageQuality(User user) {
        if (user.getQuality() < settings.getAverageQuality()) {
            this.numLowQuality++;
        } else {
            this.numHighQuality++;
        }
    }
}
```

Файл UserInfoForm.java основної програми

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.util.ConcurrentModificationException;
import java.util.HashMap;

/**
 * Клас, який представляє графічне вікно виводу інформації про об'єкти.
 * У класі є можливість виведення інформації як про один об'єкт, так і про
багато об'єктів за один раз.
 * Також використовується для виведення зв'язків об'єкта.
 */
public class UserInfoForm extends JFrame {
    /**
     * Головне вікно програми.
     * Використовується для зворотного зв'язку.
     */
    private final MainForm mainForm;
    /**
     * Допоміжний елемент для виведення графічних елементів на екран.
     */
    private final GridBagConstraints constraints;
    /**
     * Головний контейнер графічного вікна.
     */
    private JPanel mainPanel;
    /**
     * Додатковий контейнер графічного вікна.
     * Зберігає інші графічні елементи.
     */
    private JPanel panel;
    private int y = 0;
    /**
     * Кнопка підтвердження внесених змін до об'єкта.
     */
    private JButton acceptBtn;
    /**
     * Текстове поле для зміни значення репутації.
     */
    private JTextField reputationField;
    /**
     * Текстове поле для зміни значення популярності.
     */
    private JTextField popularityField;
    /**
     * Текстове поле для зміни значення якості взаємодії.
     */
    private JTextField qualityField;

    /**
     * Приватний конструктор класу.
     * Використовується публічними конструкторами.
     *
     * @param mainForm Головне вікно програми.
     */
    private UserInfoForm(MainForm mainForm) {
        super("Інформація про об'єкт");
        this.mainForm = mainForm;
        this.constraints = new GridBagConstraints();

        panel.setLayout(new GridBagLayout());
        this.setContentPane(mainPanel);
        JDialog dialog = new JDialog(this);
        dialog.setDefaultCloseOperation(JDialog.DISPOSE_ON_CLOSE);
        this.setSize(400, 500);
        Toolkit toolkit = Toolkit.getDefaultToolkit();
```

```
        Dimension dimension = toolkit.getScreenSize();
        this.setLocation(dimension.width / 2 - this.getWidth() / 2,
dimension.height / 2 - this.getHeight() / 2);
        addCloseEvent(this);

        constraints.gridx = 0;
        constraints.gridy = y++;
    }

    /**
     * Конструктор класу.
     * Виводить інформацію про один об'єкт.
     *
     * @param mainForm Головне вікно програми.
     * @param user     Об'єкт для виведення інформації.
     */
    UserInfoForm(MainForm mainForm, User user) {
        this(mainForm);
        try {
            createLabel("Унікальний ідентифікатор:");
            createTextField(String.valueOf(user.getID()));
            createLabel("Репутація:");
            reputationField =
createTextField(String.valueOf(user.getReputation()));
            createLabel("Популярність:");
            popularityField =
createTextField(String.valueOf(user.getPopularity()));
            createLabel("Якість взаємодії:");
            qualityField = createTextField(String.valueOf(user.getQuality()));
            createLabel("Кількість успішних контактів:");
            createTextField(String.valueOf(user.getNumSuccessContacts()));
            createLabel("Список контактів (id : якість взаємодії):");
            createTextField(String.valueOf(user.contactsToString()));
            createLabel("");
        } catch (NullPointerException ex) {
            JOptionPane.showMessageDialog(null,
                    "Не вдалось отримати об'єкт!");
        }
        acceptBtnSetup(user);
    }

    /**
     * Конструктор класу.
     * Виводить інформацію про багато об'єктів.
     *
     * @param mainForm Головне вікно програми.
     * @param users    Структура даних з усіма об'єктами для виведення.
     */
    UserInfoForm(MainForm mainForm, HashMap<Integer, User> users) {
        this(mainForm);
        this.setTitle("Інформація про об'єкти");
        JTextArea textArea = new JTextArea(20, 20);
        try {
            for (User user : users.values()) {
                textArea.append("\nУнікальний ідентифікатор: ");
                textArea.append(String.valueOf(user.getID()));
                textArea.append("\nРепутація: ");
                textArea.append(String.valueOf(user.getReputation()));
                textArea.append("\nПопулярність: ");
                textArea.append(String.valueOf(user.getPopularity()));
                textArea.append("\nЯкість взаємодії: ");
                textArea.append(String.valueOf(user.getQuality()));
                textArea.append("\nКількість успішних контактів: ");
                textArea.append(String.valueOf(user.getNumSuccessContacts()));
                textArea.append("\nСписок контактів (id : якість взаємодії): ");
                textArea.append(String.valueOf(user.contactsToString()));
                textArea.append("\n");
            }
        } catch (NullPointerException ex) {
```

```
            JOptionPane.showMessageDialog(null,
                    "Не вдалось отримати об'єкт!");
        }
        panel.add(textArea, constraints);
        constraints.gridy = y++;
    }

    /**
     * Конструктор класу.
     * Виводить інформацію про зв'язки об'єкта.
     * Візуалізує граф репутації.
     *
     * @param mainForm  Головне вікно програми.
     * @param linksList Об'єкт, який зберігає інформацію про зв'язки.
     */
    UserInfoForm(MainForm mainForm, LinksList linksList) {
        this(mainForm);
        this.setTitle("Інформація про зв'язки об'єкта");
        createLabel("Обраний ID:");
        createTextField(String.valueOf(linksList.getStartId()));
        createLabel("Кількість зв'язків першого порядку:");
        createTextField(String.valueOf(linksList.getFirstOrderSize()));
        createLabel("Кількість зв'язків другого порядку:");
        createTextField(String.valueOf(linksList.getSecondOrderSize()));
        createLabel("Кількість зв'язків третього порядку:");
        createTextField(String.valueOf(linksList.getThirdOrderSize()));

        Toolkit toolkit = Toolkit.getDefaultToolkit();
        Dimension dimension = toolkit.getScreenSize();
        int width = dimension.width - 200;
        int height = dimension.height - 100;
        this.setSize(width, height);
        this.setLocation(dimension.width / 2 - this.getWidth() / 2,
            dimension.height / 2 - this.getHeight() / 2);

        JPanel jungPanel = new JPanel();
        jungPanel.setBorder(BorderFactory.createLineBorder(Color.black));
        BorderLayout panelMapLayout = new BorderLayout();
        jungPanel.setLayout(panelMapLayout);
        try {
            JungGraph jungGraph = new JungGraph(linksList, width - 50,
                height - 200);
            jungPanel.add(jungGraph.getVisualizationViewer());
        } catch (ConcurrentModificationException e) {
            JOptionPane.showMessageDialog(null,
                "На жаль, не вдалось побудувати репутаційний граф.");
        }     panel.add(jungPanel, constraints);
        constraints.gridy = y++;
    }

    /**
     * Допоміжний метод створення та форматування тексту.
     *
     * @param text Текст для виведення на екран.
     */
    private void createLabel(String text) {
        JLabel label = new JLabel(text);
        panel.add(label, constraints);
        constraints.gridy = y++;
    }

    /**
     * Допоміжний метод створення та форматування текстових полів.
     *
     * @param text Текст для виведення на екран.
     */
    private JTextField createTextField(String text) {
        JTextField field = new JTextField(text, 20);
        panel.add(field, constraints);
```

```
            constraints.gridy = y++;
            return field;
        }

        /**
         * Створення та налаштування кнопки підтвердження внесених змін.
         *
         * @param user Об'єкт, в який будуть внесені зміни.
         */
        private void acceptBtnSetup(User user) {
            acceptBtn = new JButton("Підтвердити зміни");
            acceptBtn.addActionListener(e -> {
                try {
                    user.setReputation(Integer.parseInt(reputationField.getText()));
                    user.setPopularity(Integer.parseInt(popularityField.getText()));
                    user.setQuality(Integer.parseInt(qualityField.getText()));
                    JOptionPane.showMessageDialog(null, "Зміни внесені.");
                } catch (NullPointerException ex) {
                    JOptionPane.showMessageDialog(null,
                            "Не вдалось отримати об'єкт!");
                }
            });
            panel.add(acceptBtn, constraints);
            constraints.gridy = y++;
        }

        /**
         * Спрацьовує при закритті вікна налаштувань.
         * Повертає активність кнопкам головного вікна.
         *
         * @param frame Вікно налаштувань.
         */
        private void addCloseEvent(JFrame frame) {
            frame.addWindowListener(new WindowAdapter() {
                @Override
                public void windowClosing(WindowEvent e) {
                    mainForm.changeButtonsStatus(true);
                }
            });
        }
    }
```

Файл JungGraph.java основної програми

```
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Rectangle;
import java.awt.Shape;
import java.awt.geom.Ellipse2D;
import java.awt.geom.Point2D;

/**
 * Даний клас візуалізує граф репутації.
 */
public class JungGraph {
    /**
     * Максимальна кількість вершин для виводу на екран.
     */
    private static final int MAX_COUNT = 100;
    private final VisualizationViewer<Integer, String> visView;

    JungGraph(LinksList linksList, int width, int height) {
        UndirectedSparseGraph<Integer, String> g = new
UndirectedSparseGraph<>();

        // Головна вершина.
        g.addVertex(linksList.getStartId());

        // Вершини першого порядку.
        int count = 0;
        for (Integer id : linksList.getFirstOrder()) {
            g.addVertex(id);
            g.addEdge(linksList.getStartId() + " - " + id,
                linksList.getStartId(), id);
            if (++count > MAX_COUNT) {
                break;
            }
        }

        // Зв'язки між вершинами першого порядку.
        for (Integer vertex : g.getVertices()) {
            LinksList links = new LinksList(vertex, linksList.getModel(), true);
            for (Integer id : linksList.getFirstOrder()) {
                if (vertex.equals(id) || vertex == linksList.getStartId()) {
                    continue;
                }
                if (links.getFirstOrder().contains(id)) {
//                  g.addVertex(id);
                    g.addEdge(vertex + " - " + id, vertex, id);
                }
            }
        }

        visView = new VisualizationViewer<>(new CircleLayout<>(g), new
Dimension(width, height));
        visView.getRenderContext().setVertexLabelTransformer(String::valueOf);
        visView.getRenderContext().setEdgeLabelTransformer(arg0 -> arg0);
        visView.getRenderer().setVertexRenderer(new
MyRenderer(linksList.getStartId(), linksList.getModel()));
        final DefaultModalGraphMouse<String, Number> graphMouse = new
DefaultModalGraphMouse<>();
        graphMouse.setMode(ModalGraphMouse.Mode.PICKING);
        visView.setGraphMouse(graphMouse);
    }

    public VisualizationViewer<Integer, String> getVisualizationViewer() {
        return visView;
    }

}
```

```
/**
 * Допоміжний клас, що відповідає за графіку.
 */
class MyRenderer implements Renderer.Vertex<Integer, String> {
    /**
     * Унікальний ідентифікатор об'єкта, для якого візуалізується граф
репутації.
     */
    private final int START_ID;
    /**
     * Об'єкт моделі.
     */
    private final Model model;

    /**
     * Конструктор класу.
     *
     * @param startId Початковий унікальний ідентифікатор.
     * @param model   Об'єкт моделі.
     */
    MyRenderer(int startId, Model model) {
        this.START_ID = startId;
        this.model = model;
    }

    /**
     * Відповідає за графіку.
     */
    @Override
    public void paintVertex(RenderContext<Integer, String> rc, Layout<Integer,
String> layout, Integer vertex) {
        GraphicsDecorator graphicsContext = rc.getGraphicsContext();
        Point2D center = layout.transform(vertex);
        Shape shape;
        Color color;
        int size = 10;
        if (vertex == START_ID) {
            shape = new Rectangle((int) center.getX() - size / 2,
                (int) center.getY() - size / 2, size, size);
        } else {
            shape = new Ellipse2D.Double(center.getX() - size / 2.0,
                center.getY() - size / 2.0, size, size);
        }
        int rep = model.getUser(vertex).getReputation();
        color = new Color((int) (250 - rep * 2.5), (int) (rep * 2.5), 0);
        graphicsContext.setPaint(color);
        graphicsContext.fill(shape);
    }
}
```

Файл User.java основної програми

```
import javax.swing.*;
import java.util.HashMap;
import java.util.Random;

/**
 * Представлення об'єкта з усіма характеристиками.
 * Найменша одиниця комп'ютерної моделі репутаційної соціальної мережі.
 */
public class User {
    /**
     * Мінімальний можливий відсоток.
     */
    public static final int MIN_PERCENT = 0;
    /**
     * Максимальний можливий відсоток.
     */
    public static final int MAX_PERCENT = 100;
    /**
     * Базовий відсоток.
     */
    private static final int INITIAL_PERCENT = (MIN_PERCENT + MAX_PERCENT) / 2;
    /**
     * Крок перевірки (оновлення) внутрішніх даних.
     */
    private static final int STEP = 5;
    /**
     * Генератор псевдовипадкових чисел.
     */
    private static final Random RANDOM = new Random();

    /**
     * Лічильник. Рахує кількість створених об'єктів.
     */
    private static int count = 0;

    /**
     * Унікальний ідентифікатор об'єкта.
     */
    private final int ID;
    /**
     * Репутація об'єкта. Залежить від якості контактів з даним об'єктом.
     */
    private int reputation;
    /**
     * Популярність об'єкта. Кількість зв'язків з іншими об'єктами, що
здійснилися.
     */
    private int popularity;
    /**
     * Якість взаємодії. Рахується від 0 до 100. Вище значення - вищий шанс
якісної взаємодії.
     */
    private int quality;
    /**
     * Бонус до якості взаємодії за кількість здійснених контактів.
     */
    private int qualityBonus;
    /**
     * Штраф до якості взаємодії за низьку репутацію.
     */
    private int qualityDebuff;
    /**
     * Кількість балів від взаємодій.
     * Коли інший об'єкт звертається до даного, він оцінює якість взаємодії.
     * Властивість використовується для оптимізації перерахунку репутації.
     */
    private int numOfPoints;
```

```
    /**
     * Кількість успішних контактів.
     */
    private int numSuccessContacts;
    /**
     * Список контактів об'єкта.
     * Ключі - id інших об'єктів, значення - загальна якість взаємодії з іншими
об'єктами.
     */
    private final HashMap<Integer, Integer> contacts;

    /**
     * Конструктор об'єкта.
     * Ініціалізує усі внутрішні властивості об'єкта початковими значеннями.
     *
     * @param randStat Визначає, чи потрібно внести частину випадковості в
початкові значення.
     */
    User(boolean randStat) {
        this.ID = User.count++;
        this.reputation = INITIAL_PERCENT;
        this.popularity = 0;
        this.quality = INITIAL_PERCENT;
        this.qualityBonus = 0;
        this.qualityDebuff = 0;
        this.numOfPoints = 0;
        this.contacts = new HashMap<>();
        if (randStat) {
            this.reputation = RANDOM.nextInt(MIN_PERCENT, MAX_PERCENT);
            this.quality = RANDOM.nextInt(MIN_PERCENT, MAX_PERCENT);
        }
    }

    /**
     * Конструктор об'єкта.
     * Ініціалізує усі внутрішні властивості об'єкта вказаними початковими
значеннями.
     * Значення contacts передається як рядок.
     *
     * @param id                Унікальний ідентифікатор об'єкта.
     * @param reputation        Репутація об'єкта.
     * @param popularity        Популярність об'єкта.
     * @param quality           Якість взаємодії.
     * @param qualityBonus      Бонус до якості взаємодії.
     * @param qualityDebuff     Штраф до якості взаємодії.
     * @param numOfPoints       Кількість балів від взаємодій.
     * @param numSuccessContacts Кількість успішних взаємодій.
     * @param contacts          Список контактів об'єкта.
     */
    public User(int id, int reputation, int popularity, int quality, int
qualityBonus, int qualityDebuff, int numOfPoints,
                int numSuccessContacts, String contacts) {
        this.ID = id;
        this.reputation = reputation;
        this.popularity = popularity;
        this.quality = quality;
        this.qualityBonus = qualityBonus;
        this.qualityDebuff = qualityDebuff;
        this.numOfPoints = numOfPoints;
        this.numSuccessContacts = numSuccessContacts;
        this.contacts = new HashMap<>();
        if (contacts.length() == 2) {
            return;
        }
        for (String str : contacts.split(", ")) {
            str = str.replace("{", "");
            str = str.replace("}", "");
            String[] numbers = str.split("=");
            this.contacts.put(Integer.parseInt(numbers[0]),
```

```
                Integer.parseInt(numbers[1]));
        }
    }

    /**
     * Конструктор об'єкта.
     * Ініціалізує усі внутрішні властивості об'єкта вказаними початковими
значеннями.
     * Значення contacts передається як структура даних.
     *
     * @param id                 Унікальний ідентифікатор об'єкта.
     * @param reputation         Репутація об'єкта.
     * @param popularity         Популярність об'єкта.
     * @param quality            Якість взаємодії.
     * @param qualityBonus       Бонус до якості взаємодії.
     * @param qualityDebuff      Штраф до якості взаємодії.
     * @param numOfPoints        Кількість балів від взаємодій.
     * @param numSuccessContacts Кількість успішних взаємодій.
     * @param contacts           Список контактів об'єкта.
     */
    public User(int id, int reputation, int popularity, int quality, int
qualityBonus, int qualityDebuff, int numOfPoints,
                int numSuccessContacts, HashMap<Integer, Integer> contacts) {
        this.ID = id;
        this.reputation = reputation;
        this.popularity = popularity;
        this.quality = quality;
        this.qualityBonus = qualityBonus;
        this.qualityDebuff = qualityDebuff;
        this.numOfPoints = numOfPoints;
        this.numSuccessContacts = numSuccessContacts;
        this.contacts = contacts;
    }

    /**
     * Здійснює новий контакт між поточним об'єктом та іншим об'єктом.
     * Поточний об'єкт звертається до іншого об'єкту та формує про нього
враження (якість взаємодії).
     * Визначає результат їх взаємодії, що впливає на репутацію між цими
об'єктами.
     *
     * @param user Інший об'єкт, з яким здійснюється контакт.
     */
    public void makeContact(User user) {
        // Популярність об'єкта, до якого звернулись, збільшується.
        user.increasePopularity();
        // Визначення якості взаємодії з user.
        int chance = RANDOM.nextInt(MIN_PERCENT, MAX_PERCENT);
        int newQuality = chance < user.getQuality() ? 100 : 0;
        // Якщо до цього контакт між ними вже був...
        if (contacts.containsKey(user.getID())) {
            // ...обрахувати середнє значення між попередніми та новим
контактами.
            // Останній контакт залишає більше враження, ніж попередні.
            contacts.put(user.getID(), (contacts.get(user.getID()) + newQuality)
/ 2);
        } else {
            // Інакше, створити новий контакт та записати початкове враження.
            contacts.put(user.getID(), newQuality);
        }
        // Додати нову кількість балів до решти балів.
        user.addNumOfPoints(newQuality);
        // Якщо контакт успішний.
        if (newQuality != 0) {
            user.increaseNumSuccessContacts();
        }
    }

    /**
```

```
     * Повертає унікальний ідентифікатор об'єкта.
     *
     * @return Унікальний ідентифікатор.
     */
    public int getID() {
        return ID;
    }

    /**
     * Повертає репутацію об'єкта.
     * Впливає на конкуренцію даного об'єкта з іншими.
     *
     * @return Репутація.
     */
    public int getReputation() {
        return reputation;
    }

    /**
     * Встановлює нове значення репутації об'єкта.
     * Штраф якості взаємодії залежить від репутації.
     * Чим нижча репутація, ти гіршою буде взаємодія.
     *
     * @param reputation Нове значення репутації.
     */
    public void setReputation(int reputation) {
//        if (reputation < 0 || reputation > 100) {
//            JOptionPane.showMessageDialog(null,
//                    "Нове значення якості взаємодії виходить за допустимі
межі!");
//        }
        if (reputation < 0)
            this.reputation = 0;
        else this.reputation = Math.min(reputation, 100);
        this.qualityDebuff = (int) Math.log(MAX_PERCENT - this.reputation + 1);
    }

    /**
     * Повертає популярність об'єкта.
     * Впливає на якість взаємодії об'єкта з іншими.
     *
     * @return Популярність.
     */
    public int getPopularity() {
        return popularity;
    }

    /**
     * Встановлює нове значення популярності.
     *
     * @param popularity Нове значення популярності.
     */
    public void setPopularity(int popularity) {
        this.popularity = popularity;
    }

    /**
     * Збільшує на один популярність об'єкта.
     * Бонус якості взаємодії залежить від популярності.
     * Чим вища популярність, ти кращою буде взаємодія.
     */
    public void increasePopularity() {
        this.popularity++;
        if (this.popularity % STEP == 0) {
            this.qualityBonus = (int) Math.log(this.popularity);
        }
    }

    /**
```

```
     * Повертає загальну якість взаємодії з об'єктом.
     * Впливає на репутацію даного об'єкта.
     * Враховує бонус та штраф якості.
     * Бонус залежить від популярності.
     * Штраф залежить від репутації.
     *
     * @return Загальна якість взаємодії.
     */
    public int getQuality() {
        return quality + qualityBonus - qualityDebuff;
    }

    /**
     * Повертає якість взаємодії з об'єктом без бонусу або штрафу.
     *
     * @return Якість взаємодії.
     */
    public int getDefaultQuality() {
        return quality;
    }

    /**
     * Встановлює нове значення якості взаємодії даного об'єкта.
     * Впливає на репутацію об'єкта.
     *
     * @param quality Нове значення якості взаємодії.
     */
    public void setQuality(int quality) {
//        if (quality < 0 || quality > 100) {
//            JOptionPane.showMessageDialog(null,
//                    "Нове значення якості взаємодії виходить за допустимі
межі!");
//        }
        if (quality < 0)
            this.quality = 0;
        else this.quality = Math.min(quality, 100);
    }

    /**
     * Повертає бонус якості взаємодії.
     *
     * @return Бонус якості взаємодії.
     */
    public int getQualityBonus() {
        return qualityBonus;
    }

    /**
     * Встановлює нове значення для бонусу якості взаємодії.
     *
     * @param qualityBonus Нове значення бонусу якості взаємодії.
     */
    public void setQualityBonus(int qualityBonus) {
//        if (qualityBonus < 0) {
//            JOptionPane.showMessageDialog(null,
//                    "Бонус якості взаємодії не може бути меншим за нуль!");
//        }
        this.qualityBonus = Math.max(qualityBonus, 0);
    }

    /**
     * Повертає штраф якості взаємодії.
     *
     * @return Штраф якості взаємодії.
     */
    public int getQualityDebuff() {
        return qualityDebuff;
    }
```

```
    /**
     * Встановлює нове значення для штрафу якості взаємодії.
     *
     * @param qualityDebuff Нове значення штрафу якості взаємодії.
     */
    public void setQualityDebuff(int qualityDebuff) {
//        if (qualityDebuff < 0) {
//            JOptionPane.showMessageDialog(null,
//                    "Штраф якості взаємодії не може бути меншим за нуль!");
//        }
        this.qualityDebuff = Math.max(qualityDebuff, 0);
    }

    /**
     * Повертає кількість зароблених балів від взаємодій з об'єктами.
     *
     * @return Кількість зароблених балів.
     */
    public int getNumOfPoints() {
        return numOfPoints;
    }

    /**
     * Додає нове значення зароблених балів від взаємодій з об'єктами.
     * Перераховує репутацію в залежності від набраних балів.
     *
     * @param numOfPoints Нове значення зароблених балів.
     */
    public void addNumOfPoints(int numOfPoints) {
//        if (numOfPoints < 0 || numOfPoints > 100) {
//            JOptionPane.showMessageDialog(null,
//                    "Кількість зароблених балів не може бути меншим за нуль
або більшим за сто!");
//        }
        if (numOfPoints > 100)
            this.numOfPoints += 100;
        else if (numOfPoints >= 0) {
            this.numOfPoints += numOfPoints;
        }
        // Загальна кількість балів / загальна кількість контактів = репутація.
        this.setReputation(this.numOfPoints / this.popularity);
    }

    /**
     * Повертає кількість успішних контактів.
     *
     * @return Кількість успішних контактів.
     */
    public int getNumSuccessContacts() {
        return numSuccessContacts;
    }

    /**
     * Збільшує на один кількість успішних контактів.
     */
    private void increaseNumSuccessContacts() {
        this.numSuccessContacts++;
    }

    /**
     * Повертає загальну якість взаємодії з об'єктом, id якого передається в
метод.
     *
     * @param id Унікальний ідентифікатор об'єкта.
     * @return Загальна якість взаємодії з вказаним об'єктом.
     */
    public int getValueFromContacts(int id) {
        if (!contacts.containsKey(id)) {
            JOptionPane.showMessageDialog(null,
```

```
                    "Унікальний ідентифікатор не знайдено у списку контактів
даного об'єкта!\n" +
                                "id = " + id);
            return 0;
        }
        return contacts.get(id);
    }

    /**
     * Повертає кількість унікальних ідентифікаторів у списку контактів.
     *
     * @return Кількість унікальних ідентифікаторів.
     */
    public int getContactsSize() {
        return contacts.size();
    }

    /**
     * Повертає список всіх унікальних ідентифікаторів зі списку контактів.
     *
     * @return Список всіх унікальних ідентифікаторів.
     */
    public java.util.Set<Integer> getContactsIds() {
        return contacts.keySet();
    }

    /**
     * Повертає рядкове представлення контактів об'єкта.
     *
     * @return Рядкове представлення контактів.
     */
    public String contactsToString() {
        return contacts.toString();
    }

    /**
     * Скинути до нуля значення для статичного лічильника створених об'єктів.
     */
    public static void resetCount() {
        User.count = 0;
    }

    /**
     * Пофертає рядкове представлення об'єкта з усіма його внутрішніми даними.
     *
     * @return Рядкове представлення об'єкта.
     */
    @Override
    public String toString() {
        return "User{" +
                "ID=" + ID +
                ", reputation=" + reputation +
                ", popularity=" + popularity +
                ", quality=" + quality +
                ", qualityBonus=" + qualityBonus +
                ", qualityDebuff=" + qualityDebuff +
                ", numOfPoints=" + numOfPoints +
                ", contacts=" + contacts +
                '}';
    }
}
```

Файл MainForm.form основної програми

```
<?xml version="1.0" encoding="UTF-8"?>
<form xmlns="http://www.intellij.com/uidesigner/form/" version="1" bind-to-
class="MainForm">
  <grid id="27dc6" binding="panel" layout-manager="GridLayoutManager" row-
count="12" column-count="5" same-size-horizontally="false" same-size-
vertically="false" hgap="-1" vgap="-1">
    <margin top="0" left="0" bottom="0" right="0"/>
    <constraints>
      <xy x="20" y="20" width="629" height="974"/>
    </constraints>
    <properties/>
    <border type="none"/>
    <children>
      <scrollpane id="bcd8f">
        <constraints>
          <grid row="1" column="1" row-span="10" col-span="1" vsize-policy="7"
hsize-policy="7" anchor="0" fill="3" indent="0" use-parent-layout="false"/>
        </constraints>
        <properties/>
        <border type="none"/>
        <children>
          <grid id="d78e" layout-manager="GridLayoutManager" row-count="28"
column-count="2" same-size-horizontally="false" same-size-vertically="false"
hgap="-1" vgap="-1">
            <margin top="0" left="0" bottom="0" right="0"/>
            <constraints/>
            <properties/>
            <border type="none"/>
            <children>
              <component id="3f923" class="javax.swing.JLabel">
                <constraints>
                  <grid row="0" column="0" row-span="1" col-span="1" vsize-
policy="0" hsize-policy="0" anchor="8" fill="0" indent="0" use-parent-
layout="false"/>
                </constraints>
                <properties>
                  <text value="Кількість об'єктів"/>
                </properties>
              </component>
              <component id="ef1d7" class="javax.swing.JLabel">
                <constraints>
                  <grid row="2" column="0" row-span="1" col-span="1" vsize-
policy="0" hsize-policy="0" anchor="8" fill="0" indent="0" use-parent-
layout="false"/>
                </constraints>
                <properties>
                  <text value="Кількість зв'язків"/>
                </properties>
              </component>
              <component id="806bc" class="javax.swing.JLabel">
                <constraints>
                  <grid row="4" column="0" row-span="1" col-span="1" vsize-
policy="0" hsize-policy="0" anchor="8" fill="0" indent="0" use-parent-
layout="false"/>
                </constraints>
                <properties>
                  <text value="Мінімальна репутація"/>
                </properties>
              </component>
              <component id="5a30b" class="javax.swing.JLabel">
                <constraints>
                  <grid row="6" column="0" row-span="1" col-span="1" vsize-
policy="0" hsize-policy="0" anchor="8" fill="0" indent="0" use-parent-
layout="false"/>
                </constraints>
                <properties>
                  <text value="Максимальна репутація"/>
```

```
              </properties>
            </component>
            <component id="fa9ff" class="javax.swing.JLabel">
              <constraints>
                <grid row="8" column="0" row-span="1" col-span="1" vsize-
policy="0" hsize-policy="0" anchor="8" fill="0" indent="0" use-parent-
layout="false"/>
              </constraints>
              <properties>
                <text value="Мінімальна популярність"/>
              </properties>
            </component>
            <component id="7ff63" class="javax.swing.JLabel">
              <constraints>
                <grid row="10" column="0" row-span="1" col-span="1" vsize-
policy="0" hsize-policy="0" anchor="8" fill="0" indent="0" use-parent-
layout="false"/>
              </constraints>
              <properties>
                <text value="Максимальна популярність"/>
              </properties>
            </component>
            <component id="d5a88" class="javax.swing.JLabel">
              <constraints>
                <grid row="12" column="0" row-span="1" col-span="1" vsize-
policy="0" hsize-policy="0" anchor="8" fill="0" indent="0" use-parent-
layout="false"/>
              </constraints>
              <properties>
                <text value="Мінімальна якість взаємодії"/>
              </properties>
            </component>
            <component id="3cff0" class="javax.swing.JLabel">
              <constraints>
                <grid row="14" column="0" row-span="1" col-span="1" vsize-
policy="0" hsize-policy="0" anchor="8" fill="0" indent="0" use-parent-
layout="false"/>
              </constraints>
              <properties>
                <text value="Максимальна якість взаємодії"/>
              </properties>
            </component>
            <component id="b697" class="javax.swing.JLabel">
              <constraints>
                <grid row="16" column="0" row-span="1" col-span="1" vsize-
policy="0" hsize-policy="0" anchor="8" fill="0" indent="0" use-parent-
layout="false"/>
              </constraints>
              <properties>
                <text value="Мінімальна кількість успішних контактів"/>
              </properties>
            </component>
            <component id="fcf6e" class="javax.swing.JLabel">
              <constraints>
                <grid row="18" column="0" row-span="1" col-span="1" vsize-
policy="0" hsize-policy="0" anchor="8" fill="0" indent="0" use-parent-
layout="false"/>
              </constraints>
              <properties>
                <text value="Максимальна кількість успішних контактів"/>
              </properties>
            </component>
            <component id="bc6f9" class="javax.swing.JTextField"
binding="numUsersField">
              <constraints>
                <grid row="1" column="0" row-span="1" col-span="1" vsize-
policy="0" hsize-policy="6" anchor="8" fill="1" indent="0" use-parent-
layout="false">
                  <preferred-size width="150" height="-1"/>
```

```
                </grid>
              </constraints>
              <properties/>
            </component>
            <component id="6f591" class="javax.swing.JTextField"
binding="numContactsField">
              <constraints>
                <grid row="3" column="0" row-span="1" col-span="1" vsize-
policy="0" hsize-policy="6" anchor="8" fill="1" indent="0" use-parent-
layout="false">
                  <preferred-size width="150" height="-1"/>
                </grid>
              </constraints>
              <properties/>
            </component>
            <component id="c378a" class="javax.swing.JTextField"
binding="minReputationField">
              <constraints>
                <grid row="5" column="0" row-span="1" col-span="1" vsize-
policy="0" hsize-policy="6" anchor="8" fill="1" indent="0" use-parent-
layout="false">
                  <preferred-size width="150" height="-1"/>
                </grid>
              </constraints>
              <properties/>
            </component>
            <component id="f9fd0" class="javax.swing.JTextField"
binding="maxReputationField">
              <constraints>
                <grid row="7" column="0" row-span="1" col-span="1" vsize-
policy="0" hsize-policy="6" anchor="8" fill="1" indent="0" use-parent-
layout="false">
                  <preferred-size width="150" height="-1"/>
                </grid>
              </constraints>
              <properties/>
            </component>
            <component id="48501" class="javax.swing.JTextField"
binding="minPopularityField">
              <constraints>
                <grid row="9" column="0" row-span="1" col-span="1" vsize-
policy="0" hsize-policy="6" anchor="8" fill="1" indent="0" use-parent-
layout="false">
                  <preferred-size width="150" height="-1"/>
                </grid>
              </constraints>
              <properties/>
            </component>
            <component id="493e5" class="javax.swing.JTextField"
binding="maxPopularityField">
              <constraints>
                <grid row="11" column="0" row-span="1" col-span="1" vsize-
policy="0" hsize-policy="6" anchor="8" fill="1" indent="0" use-parent-
layout="false">
                  <preferred-size width="150" height="-1"/>
                </grid>
              </constraints>
              <properties/>
            </component>
            <component id="339b5" class="javax.swing.JTextField"
binding="minQualityField">
              <constraints>
                <grid row="13" column="0" row-span="1" col-span="1" vsize-
policy="0" hsize-policy="6" anchor="8" fill="1" indent="0" use-parent-
layout="false">
                  <preferred-size width="150" height="-1"/>
                </grid>
              </constraints>
              <properties/>
```

```
            </component>
            <component id="477aa" class="javax.swing.JTextField"
binding="maxQualityField">
              <constraints>
                <grid row="15" column="0" row-span="1" col-span="1" vsize-
policy="0" hsize-policy="6" anchor="8" fill="1" indent="0" use-parent-
layout="false">
                  <preferred-size width="150" height="-1"/>
                </grid>
              </constraints>
              <properties/>
            </component>
            <component id="ee05c" class="javax.swing.JTextField"
binding="minNumSuccessContactsField">
              <constraints>
                <grid row="17" column="0" row-span="1" col-span="1" vsize-
policy="0" hsize-policy="6" anchor="8" fill="1" indent="0" use-parent-
layout="false">
                  <preferred-size width="150" height="-1"/>
                </grid>
              </constraints>
              <properties/>
            </component>
            <component id="96910" class="javax.swing.JTextField"
binding="maxNumSuccessContactsField">
              <constraints>
                <grid row="19" column="0" row-span="1" col-span="1" vsize-
policy="0" hsize-policy="6" anchor="8" fill="1" indent="0" use-parent-
layout="false">
                  <preferred-size width="150" height="-1"/>
                </grid>
              </constraints>
              <properties/>
            </component>
            <component id="e875f" class="javax.swing.JButton"
binding="minReputationButton">
              <constraints>
                <grid row="5" column="1" row-span="1" col-span="1" vsize-
policy="0" hsize-policy="3" anchor="0" fill="1" indent="0" use-parent-
layout="false"/>
              </constraints>
              <properties>
                <text value="Інфо."/>
              </properties>
            </component>
            <component id="b0860" class="javax.swing.JButton"
binding="maxReputationButton">
              <constraints>
                <grid row="7" column="1" row-span="1" col-span="1" vsize-
policy="0" hsize-policy="3" anchor="0" fill="1" indent="0" use-parent-
layout="false"/>
              </constraints>
              <properties>
                <text value="Інфо."/>
              </properties>
            </component>
            <component id="a3eca" class="javax.swing.JButton"
binding="minPopularityButton">
              <constraints>
                <grid row="9" column="1" row-span="1" col-span="1" vsize-
policy="0" hsize-policy="3" anchor="0" fill="1" indent="0" use-parent-
layout="false"/>
              </constraints>
              <properties>
                <text value="Інфо."/>
              </properties>
            </component>
            <component id="deb7c" class="javax.swing.JButton"
binding="maxPopularityButton">
```

```
                <constraints>
                  <grid row="11" column="1" row-span="1" col-span="1" vsize-
policy="0" hsize-policy="3" anchor="0" fill="1" indent="0" use-parent-
layout="false"/>
                </constraints>
                <properties>
                  <text value="Інфо."/>
                </properties>
              </component>
              <component id="71fc6" class="javax.swing.JButton"
binding="minQualityButton">
                <constraints>
                  <grid row="13" column="1" row-span="1" col-span="1" vsize-
policy="0" hsize-policy="3" anchor="0" fill="1" indent="0" use-parent-
layout="false"/>
                </constraints>
                <properties>
                  <text value="Інфо."/>
                </properties>
              </component>
              <component id="1626a" class="javax.swing.JButton"
binding="maxQualityButton">
                <constraints>
                  <grid row="15" column="1" row-span="1" col-span="1" vsize-
policy="0" hsize-policy="3" anchor="0" fill="1" indent="0" use-parent-
layout="false"/>
                </constraints>
                <properties>
                  <text value="Інфо."/>
                </properties>
              </component>
              <component id="c3405" class="javax.swing.JButton"
binding="minNumSuccessContactsButton">
                <constraints>
                  <grid row="17" column="1" row-span="1" col-span="1" vsize-
policy="0" hsize-policy="3" anchor="0" fill="1" indent="0" use-parent-
layout="false"/>
                </constraints>
                <properties>
                  <text value="Інфо."/>
                </properties>
              </component>
              <component id="cae3c" class="javax.swing.JButton"
binding="maxNumSuccessContactsButton">
                <constraints>
                  <grid row="19" column="1" row-span="1" col-span="1" vsize-
policy="0" hsize-policy="3" anchor="0" fill="1" indent="0" use-parent-
layout="false"/>
                </constraints>
                <properties>
                  <text value="Інфо."/>
                </properties>
              </component>
              <component id="417a2" class="javax.swing.JLabel">
                <constraints>
                  <grid row="20" column="0" row-span="1" col-span="1" vsize-
policy="0" hsize-policy="0" anchor="8" fill="0" indent="0" use-parent-
layout="false"/>
                </constraints>
                <properties>
                  <text value="Кількість об'єктів з низькою репутацією"/>
                </properties>
              </component>
              <component id="454ec" class="javax.swing.JTextField"
binding="numLowReputationField">
                <constraints>
                  <grid row="21" column="0" row-span="1" col-span="1" vsize-
policy="0" hsize-policy="6" anchor="8" fill="1" indent="0" use-parent-
layout="false">
```

```
                    <preferred-size width="150" height="-1"/>
                  </grid>
                </constraints>
                <properties/>
              </component>
              <component id="86ac2" class="javax.swing.JLabel">
                <constraints>
                  <grid row="22" column="0" row-span="1" col-span="1" vsize-
policy="0" hsize-policy="0" anchor="8" fill="0" indent="0" use-parent-
layout="false"/>
                </constraints>
                <properties>
                  <text value="кількість об'єктів з високою репутацією"/>
                </properties>
              </component>
              <component id="b9280" class="javax.swing.JTextField"
binding="numHighReputationField">
                <constraints>
                  <grid row="23" column="0" row-span="1" col-span="1" vsize-
policy="0" hsize-policy="6" anchor="8" fill="1" indent="0" use-parent-
layout="false">
                    <preferred-size width="150" height="-1"/>
                  </grid>
                </constraints>
                <properties/>
              </component>
              <component id="d2f01" class="javax.swing.JLabel">
                <constraints>
                  <grid row="24" column="0" row-span="1" col-span="1" vsize-
policy="0" hsize-policy="0" anchor="8" fill="0" indent="0" use-parent-
layout="false"/>
                </constraints>
                <properties>
                  <text value="Кількість об'єктів з низькою якістю"/>
                </properties>
              </component>
              <component id="d2c99" class="javax.swing.JTextField"
binding="numLowQualityField">
                <constraints>
                  <grid row="25" column="0" row-span="1" col-span="1" vsize-
policy="0" hsize-policy="6" anchor="8" fill="1" indent="0" use-parent-
layout="false">
                    <preferred-size width="150" height="-1"/>
                  </grid>
                </constraints>
                <properties/>
              </component>
              <component id="ef371" class="javax.swing.JLabel">
                <constraints>
                  <grid row="26" column="0" row-span="1" col-span="1" vsize-
policy="0" hsize-policy="0" anchor="8" fill="0" indent="0" use-parent-
layout="false"/>
                </constraints>
                <properties>
                  <text value="Кількість об'єктів з високою якістю"/>
                </properties>
              </component>
              <component id="d3971" class="javax.swing.JTextField"
binding="numHighQualityField">
                <constraints>
                  <grid row="27" column="0" row-span="1" col-span="1" vsize-
policy="0" hsize-policy="6" anchor="8" fill="1" indent="0" use-parent-
layout="false">
                    <preferred-size width="150" height="-1"/>
                  </grid>
                </constraints>
                <properties/>
              </component>
            </children>
```

```
          </grid>
        </children>
      </scrollpane>
      <grid id="8fc5f" layout-manager="GridLayoutManager" row-count="11" column-
count="1" same-size-horizontally="false" same-size-vertically="false" hgap="-1"
vgap="-1">
        <margin top="0" left="0" bottom="0" right="0"/>
        <constraints>
          <grid row="1" column="3" row-span="10" col-span="1" vsize-policy="3"
hsize-policy="3" anchor="0" fill="3" indent="0" use-parent-layout="false"/>
        </constraints>
        <properties/>
        <border type="none"/>
        <children>
          <component id="d861e" class="javax.swing.JButton"
binding="createContactsBtn">
            <constraints>
              <grid row="1" column="0" row-span="1" col-span="1" vsize-
policy="0" hsize-policy="3" anchor="0" fill="1" indent="0" use-parent-
layout="false"/>
            </constraints>
            <properties>
              <text value="Створити нові контакти"/>
            </properties>
          </component>
          <component id="62cb9" class="javax.swing.JButton"
binding="enumerateContactsBtn">
            <constraints>
              <grid row="2" column="0" row-span="1" col-span="1" vsize-
policy="0" hsize-policy="3" anchor="0" fill="1" indent="0" use-parent-
layout="false"/>
            </constraints>
            <properties>
              <text value="Перебрати контакти"/>
            </properties>
          </component>
          <component id="6965d" class="javax.swing.JButton"
binding="printStatisticsBtn">
            <constraints>
              <grid row="3" column="0" row-span="1" col-span="1" vsize-
policy="0" hsize-policy="3" anchor="0" fill="1" indent="0" use-parent-
layout="false"/>
            </constraints>
            <properties>
              <text value="Оновлення статистики"/>
            </properties>
          </component>
          <component id="6cfc7" class="javax.swing.JButton"
binding="printUserBtn">
            <constraints>
              <grid row="4" column="0" row-span="1" col-span="1" vsize-
policy="0" hsize-policy="3" anchor="0" fill="1" indent="0" use-parent-
layout="false"/>
            </constraints>
            <properties>
              <text value="Виведення інфо. про об'єкт"/>
            </properties>
          </component>
          <component id="2d69a" class="javax.swing.JButton"
binding="printUsersBtn">
            <constraints>
              <grid row="5" column="0" row-span="1" col-span="1" vsize-
policy="0" hsize-policy="3" anchor="0" fill="1" indent="0" use-parent-
layout="false"/>
            </constraints>
            <properties>
              <text value="Виведення інфо. про об'єкти"/>
            </properties>
          </component>
```

```
          <component id="d84ef" class="javax.swing.JButton"
binding="linksListBtn">
            <constraints>
              <grid row="6" column="0" row-span="1" col-span="1" vsize-
policy="0" hsize-policy="3" anchor="0" fill="1" indent="0" use-parent-
layout="false"/>
            </constraints>
            <properties>
              <text value="Зв'язки об'єкта"/>
            </properties>
          </component>
          <component id="425ae" class="javax.swing.JButton"
binding="saveToFileBtn">
            <constraints>
              <grid row="8" column="0" row-span="1" col-span="1" vsize-
policy="0" hsize-policy="3" anchor="0" fill="1" indent="0" use-parent-
layout="false"/>
            </constraints>
            <properties>
              <text value="Зберегти до бази даних"/>
            </properties>
          </component>
          <component id="ffc18" class="javax.swing.JButton"
binding="settingsBtn">
            <constraints>
              <grid row="10" column="0" row-span="1" col-span="1" vsize-
policy="0" hsize-policy="3" anchor="0" fill="1" indent="0" use-parent-
layout="false"/>
            </constraints>
            <properties>
              <text value="Налаштування"/>
            </properties>
          </component>
          <component id="37c94" class="javax.swing.JButton"
binding="loadFromFileBtn">
            <constraints>
              <grid row="9" column="0" row-span="1" col-span="1" vsize-
policy="0" hsize-policy="3" anchor="0" fill="1" indent="0" use-parent-
layout="false"/>
            </constraints>
            <properties>
              <text value="Завантажити з бази даних"/>
            </properties>
          </component>
          <component id="8ba8d" class="javax.swing.JButton"
binding="addUsersBtn">
            <constraints>
              <grid row="0" column="0" row-span="1" col-span="1" vsize-
policy="0" hsize-policy="3" anchor="0" fill="1" indent="0" use-parent-
layout="false"/>
            </constraints>
            <properties>
              <text value="Додати нові об'єкти"/>
            </properties>
          </component>
          <component id="e41d1" class="javax.swing.JButton" binding="clearBtn">
            <constraints>
              <grid row="7" column="0" row-span="1" col-span="1" vsize-
policy="0" hsize-policy="3" anchor="0" fill="1" indent="0" use-parent-
layout="false"/>
            </constraints>
            <properties>
              <text value="Очистити внутрішні дані"/>
            </properties>
          </component>
        </children>
      </grid>
      <hspacer id="e740">
        <constraints>
```

```
          <grid row="1" column="0" row-span="1" col-span="1" vsize-policy="1"
hsize-policy="6" anchor="0" fill="1" indent="0" use-parent-layout="false">
            <minimum-size width="10" height="-1"/>
          </grid>
        </constraints>
      </hspacer>
      <hspacer id="583d0">
        <constraints>
          <grid row="1" column="4" row-span="1" col-span="1" vsize-policy="1"
hsize-policy="6" anchor="0" fill="1" indent="0" use-parent-layout="false">
            <minimum-size width="10" height="-1"/>
          </grid>
        </constraints>
      </hspacer>
      <vspacer id="c68ed">
        <constraints>
          <grid row="11" column="1" row-span="1" col-span="1" vsize-policy="6"
hsize-policy="1" anchor="0" fill="2" indent="0" use-parent-layout="false">
            <minimum-size width="-1" height="10"/>
          </grid>
        </constraints>
      </vspacer>
      <vspacer id="68cc6">
        <constraints>
          <grid row="0" column="1" row-span="1" col-span="1" vsize-policy="6"
hsize-policy="1" anchor="0" fill="2" indent="0" use-parent-layout="false">
            <minimum-size width="-1" height="10"/>
          </grid>
        </constraints>
      </vspacer>
      <grid id="8fc2f" binding="graphPanel" layout-manager="GridLayoutManager"
row-count="1" column-count="1" same-size-horizontally="false" same-size-
vertically="false" hgap="-1" vgap="-1">
        <margin top="0" left="0" bottom="0" right="0"/>
        <constraints>
          <grid row="1" column="2" row-span="1" col-span="1" vsize-policy="3"
hsize-policy="3" anchor="0" fill="3" indent="0" use-parent-layout="false"/>
        </constraints>
        <properties/>
        <border type="none"/>
        <children/>
      </grid>
      <grid id="c81d1" binding="jungPanel" layout-manager="GridLayoutManager"
row-count="1" column-count="1" same-size-horizontally="false" same-size-
vertically="false" hgap="-1" vgap="-1">
        <margin top="0" left="0" bottom="0" right="0"/>
        <constraints>
          <grid row="2" column="2" row-span="9" col-span="1" vsize-policy="3"
hsize-policy="3" anchor="0" fill="3" indent="0" use-parent-layout="false"/>
        </constraints>
        <properties/>
        <border type="none"/>
        <children/>
      </grid>
    </children>
  </grid>
</form>
```

Файл SettingsForm.form основної програми

```
<?xml version="1.0" encoding="UTF-8"?>
<form xmlns="http://www.intellij.com/uidesigner/form/" version="1" bind-to-
class="SettingsForm">
  <grid id="27dc6" binding="panel" layout-manager="GridLayoutManager" row-
count="2" column-count="3" same-size-horizontally="false" same-size-
vertically="false" hgap="-1" vgap="-1">
    <margin top="0" left="0" bottom="0" right="0"/>
    <constraints>
      <xy x="20" y="20" width="596" height="535"/>
    </constraints>
    <properties/>
    <border type="none"/>
    <children>
      <grid id="b5d42" layout-manager="GridLayoutManager" row-count="16" column-
count="2" same-size-horizontally="false" same-size-vertically="false" hgap="-1"
vgap="-1">
        <margin top="0" left="0" bottom="0" right="0"/>
        <constraints>
          <grid row="0" column="1" row-span="1" col-span="1" vsize-policy="3"
hsize-policy="3" anchor="0" fill="3" indent="0" use-parent-layout="false"/>
        </constraints>
        <properties/>
        <border type="none"/>
        <children>
          <component id="81b5f" class="javax.swing.JLabel">
            <constraints>
              <grid row="1" column="0" row-span="1" col-span="1" vsize-
policy="0" hsize-policy="0" anchor="8" fill="0" indent="0" use-parent-
layout="false"/>
            </constraints>
            <properties>
              <text value="Кількість нових об'єктів"/>
            </properties>
          </component>
          <vspacer id="3fddb">
            <constraints>
              <grid row="14" column="0" row-span="1" col-span="1" vsize-
policy="6" hsize-policy="1" anchor="0" fill="2" indent="0" use-parent-
layout="false"/>
            </constraints>
          </vspacer>
          <component id="7c1bb" class="javax.swing.JTextField"
binding="numberNewUsersTextField">
            <constraints>
              <grid row="2" column="0" row-span="1" col-span="1" vsize-
policy="0" hsize-policy="6" anchor="8" fill="1" indent="0" use-parent-
layout="false">
                <preferred-size width="150" height="-1"/>
              </grid>
            </constraints>
            <properties/>
          </component>
          <component id="8c056" class="javax.swing.JCheckBox"
binding="isRandStatCheckBox" default-binding="true">
            <constraints>
              <grid row="3" column="0" row-span="1" col-span="1" vsize-
policy="0" hsize-policy="3" anchor="8" fill="0" indent="0" use-parent-
layout="false"/>
            </constraints>
            <properties>
              <text value="Випадкові характеристики"/>
            </properties>
          </component>
          <component id="2dae0" class="javax.swing.JLabel">
            <constraints>
```

```
            <grid row="4" column="0" row-span="1" col-span="1" vsize-
policy="0" hsize-policy="0" anchor="8" fill="0" indent="0" use-parent-
layout="false"/>
          </constraints>
          <properties>
            <text value="Мінімальна кількість нових контактів"/>
          </properties>
        </component>
        <component id="7aff7" class="javax.swing.JTextField"
binding="minCountTextField">
          <constraints>
            <grid row="5" column="0" row-span="1" col-span="1" vsize-
policy="0" hsize-policy="6" anchor="8" fill="1" indent="0" use-parent-
layout="false">
              <preferred-size width="150" height="-1"/>
            </grid>
          </constraints>
          <properties/>
        </component>
        <component id="77bbd" class="javax.swing.JLabel">
          <constraints>
            <grid row="6" column="0" row-span="1" col-span="1" vsize-
policy="0" hsize-policy="0" anchor="8" fill="0" indent="0" use-parent-
layout="false"/>
          </constraints>
          <properties>
            <text value="Максимальна кількість нових контактів"/>
          </properties>
        </component>
        <component id="97697" class="javax.swing.JTextField"
binding="maxCountTextField">
          <constraints>
            <grid row="7" column="0" row-span="1" col-span="1" vsize-
policy="0" hsize-policy="6" anchor="8" fill="1" indent="0" use-parent-
layout="false">
              <preferred-size width="150" height="-1"/>
            </grid>
          </constraints>
          <properties/>
        </component>
        <component id="4ecb3" class="javax.swing.JLabel">
          <constraints>
            <grid row="8" column="0" row-span="1" col-span="1" vsize-
policy="0" hsize-policy="0" anchor="8" fill="0" indent="0" use-parent-
layout="false"/>
          </constraints>
          <properties>
            <text value="Ймовірність контакту"/>
          </properties>
        </component>
        <component id="65a3a" class="javax.swing.JTextField"
binding="contactProbabilityTextField">
          <constraints>
            <grid row="9" column="0" row-span="1" col-span="1" vsize-
policy="0" hsize-policy="6" anchor="8" fill="1" indent="0" use-parent-
layout="false">
              <preferred-size width="150" height="-1"/>
            </grid>
          </constraints>
          <properties/>
        </component>
        <component id="1e257" class="javax.swing.JCheckBox"
binding="isDualDirectionCheckBox" default-binding="true">
          <constraints>
            <grid row="10" column="0" row-span="1" col-span="1" vsize-
policy="0" hsize-policy="3" anchor="8" fill="0" indent="0" use-parent-
layout="false"/>
          </constraints>
          <properties>
```

```
              <text value="Контакт в дві сторони"/>
            </properties>
          </component>
          <component id="bbd42" class="javax.swing.JLabel">
            <constraints>
              <grid row="11" column="0" row-span="1" col-span="1" vsize-
policy="0" hsize-policy="0" anchor="8" fill="0" indent="0" use-parent-
layout="false"/>
            </constraints>
            <properties>
              <text value="Унікальний ідентифікатор для роботи з ним"/>
            </properties>
          </component>
          <component id="c0209" class="javax.swing.JTextField"
binding="idToStringTextField">
            <constraints>
              <grid row="12" column="0" row-span="1" col-span="1" vsize-
policy="0" hsize-policy="6" anchor="8" fill="1" indent="0" use-parent-
layout="false">
                <preferred-size width="150" height="-1"/>
              </grid>
            </constraints>
            <properties/>
          </component>
          <component id="312cf" class="javax.swing.JLabel">
            <constraints>
              <grid row="1" column="1" row-span="1" col-span="1" vsize-
policy="0" hsize-policy="0" anchor="8" fill="0" indent="0" use-parent-
layout="false"/>
            </constraints>
            <properties>
              <text value="Назава бази даних"/>
            </properties>
          </component>
          <component id="187f3" class="javax.swing.JTextField"
binding="databaseNameTextField">
            <constraints>
              <grid row="2" column="1" row-span="1" col-span="1" vsize-
policy="0" hsize-policy="6" anchor="8" fill="1" indent="0" use-parent-
layout="false">
                <preferred-size width="150" height="-1"/>
              </grid>
            </constraints>
            <properties/>
          </component>
          <component id="54586" class="javax.swing.JLabel">
            <constraints>
              <grid row="3" column="1" row-span="1" col-span="1" vsize-
policy="0" hsize-policy="0" anchor="8" fill="0" indent="0" use-parent-
layout="false"/>
            </constraints>
            <properties>
              <text value="Назва основної таблиці"/>
            </properties>
          </component>
          <component id="b7af0" class="javax.swing.JTextField"
binding="tableNameTextField">
            <constraints>
              <grid row="4" column="1" row-span="1" col-span="1" vsize-
policy="0" hsize-policy="6" anchor="8" fill="1" indent="0" use-parent-
layout="false">
                <preferred-size width="150" height="-1"/>
              </grid>
            </constraints>
            <properties/>
          </component>
          <component id="56580" class="javax.swing.JLabel">
            <constraints>
```

```
            <grid row="5" column="1" row-span="1" col-span="1" vsize-
policy="0" hsize-policy="0" anchor="8" fill="0" indent="0" use-parent-
layout="false"/>
          </constraints>
          <properties>
            <text value="Кількість потоків для додаткових обчислень"/>
          </properties>
        </component>
        <component id="d0ade" class="javax.swing.JTextField"
binding="numOfThreadsTextField">
          <constraints>
            <grid row="6" column="1" row-span="1" col-span="1" vsize-
policy="0" hsize-policy="6" anchor="8" fill="1" indent="0" use-parent-
layout="false">
              <preferred-size width="150" height="-1"/>
            </grid>
          </constraints>
          <properties/>
        </component>
        <component id="fa2b5" class="javax.swing.JButton" binding="acceptBtn">
          <constraints>
            <grid row="15" column="0" row-span="1" col-span="1" vsize-
policy="0" hsize-policy="3" anchor="0" fill="1" indent="0" use-parent-
layout="false"/>
          </constraints>
          <properties>
            <text value="Підтвердити"/>
          </properties>
        </component>
        <component id="d8ccf" class="javax.swing.JButton" binding="cancelBtn">
          <constraints>
            <grid row="15" column="1" row-span="1" col-span="1" vsize-
policy="0" hsize-policy="3" anchor="0" fill="1" indent="0" use-parent-
layout="false"/>
          </constraints>
          <properties>
            <text value="Відмінити"/>
          </properties>
        </component>
        <component id="5b108" class="javax.swing.JLabel">
          <constraints>
            <grid row="0" column="0" row-span="1" col-span="1" vsize-
policy="0" hsize-policy="0" anchor="8" fill="0" indent="0" use-parent-
layout="false"/>
          </constraints>
          <properties>
            <text value=""/>
          </properties>
        </component>
        <component id="8ae0a" class="javax.swing.JLabel">
          <constraints>
            <grid row="7" column="1" row-span="1" col-span="1" vsize-
policy="0" hsize-policy="0" anchor="8" fill="0" indent="0" use-parent-
layout="false"/>
          </constraints>
          <properties>
            <text value="Середня границя репутації"/>
          </properties>
        </component>
        <component id="b14a4" class="javax.swing.JTextField"
binding="averageReputationField">
          <constraints>
            <grid row="8" column="1" row-span="1" col-span="1" vsize-
policy="0" hsize-policy="6" anchor="8" fill="1" indent="0" use-parent-
layout="false">
              <preferred-size width="150" height="-1"/>
            </grid>
          </constraints>
          <properties/>
```

```
          </component>
          <component id="6d9ec" class="javax.swing.JLabel">
            <constraints>
              <grid row="9" column="1" row-span="1" col-span="1" vsize-
policy="0" hsize-policy="0" anchor="8" fill="0" indent="0" use-parent-
layout="false"/>
            </constraints>
            <properties>
              <text value="Середня границя якості"/>
            </properties>
          </component>
          <component id="1d23f" class="javax.swing.JTextField"
binding="averageQualityField">
            <constraints>
              <grid row="10" column="1" row-span="1" col-span="1" vsize-
policy="0" hsize-policy="6" anchor="8" fill="1" indent="0" use-parent-
layout="false">
                <preferred-size width="150" height="-1"/>
              </grid>
            </constraints>
            <properties/>
          </component>
          <component id="1dde0" class="javax.swing.JCheckBox"
binding="isSimulationSelectingCheckBox">
            <constraints>
              <grid row="11" column="1" row-span="1" col-span="1" vsize-
policy="0" hsize-policy="3" anchor="8" fill="0" indent="0" use-parent-
layout="false"/>
            </constraints>
            <properties>
              <text value="Імітація вибору об'єкта контакта"/>
            </properties>
          </component>
          <component id="7b400" class="javax.swing.JLabel">
            <constraints>
              <grid row="12" column="1" row-span="1" col-span="1" vsize-
policy="0" hsize-policy="0" anchor="8" fill="0" indent="0" use-parent-
layout="false"/>
            </constraints>
            <properties>
              <text value="Кількість об'єктів для контакту (0 - без
обмеження)"/>
            </properties>
          </component>
          <component id="90d20" class="javax.swing.JTextField"
binding="numMainUsersField">
            <constraints>
              <grid row="13" column="1" row-span="1" col-span="1" vsize-
policy="0" hsize-policy="6" anchor="8" fill="1" indent="0" use-parent-
layout="false">
                <preferred-size width="150" height="-1"/>
              </grid>
            </constraints>
            <properties/>
          </component>
        </children>
      </grid>
      <hspacer id="8e4ec">
        <constraints>
          <grid row="0" column="0" row-span="1" col-span="1" vsize-policy="1"
hsize-policy="6" anchor="0" fill="1" indent="0" use-parent-layout="false"/>
        </constraints>
      </hspacer>
      <hspacer id="90571">
        <constraints>
          <grid row="0" column="2" row-span="1" col-span="1" vsize-policy="1"
hsize-policy="6" anchor="0" fill="1" indent="0" use-parent-layout="false"/>
        </constraints>
      </hspacer>
```

```
      <component id="b03ad" class="javax.swing.JLabel">
        <constraints>
          <grid row="1" column="1" row-span="1" col-span="1" vsize-policy="0"
hsize-policy="0" anchor="8" fill="0" indent="0" use-parent-layout="false"/>
        </constraints>
        <properties>
          <text value=""/>
        </properties>
      </component>
    </children>
  </grid>
</form>
```

Файл UserInfoForm.form основної програми

```
<?xml version="1.0" encoding="UTF-8"?>
<form xmlns="http://www.intellij.com/uidesigner/form/" version="1" bind-to-
class="UserInfoForm">
  <grid id="27dc6" binding="mainPanel" layout-manager="GridLayoutManager" row-
count="1" column-count="1" same-size-horizontally="false" same-size-
vertically="false" hgap="-1" vgap="-1">
    <margin top="0" left="0" bottom="0" right="0"/>
    <constraints>
      <xy x="20" y="20" width="500" height="400"/>
    </constraints>
    <properties/>
    <border type="none"/>
    <children>
      <scrollpane id="cef2b">
        <constraints>
          <grid row="0" column="0" row-span="1" col-span="1" vsize-policy="7"
hsize-policy="7" anchor="0" fill="3" indent="0" use-parent-layout="false"/>
        </constraints>
        <properties/>
        <border type="none"/>
        <children>
          <grid id="f1642" binding="panel" layout-manager="GridLayoutManager"
row-count="1" column-count="1" same-size-horizontally="false" same-size-
vertically="false" hgap="-1" vgap="-1">
            <margin top="0" left="0" bottom="0" right="0"/>
            <constraints/>
            <properties/>
            <border type="none"/>
            <children/>
          </grid>
        </children>
      </scrollpane>
    </children>
  </grid>
</form>
```