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
“____” _____ 2022 р.

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА

за другим (магістерським) рівнем вищої освіти

на тему

“ Дослідження та програмна реалізація застосування об’єктно-орієнтованих баз даних в ІС ”

Виконав здобувач вищої освіти
ІІ курсу, групи КІ-21МЗ-1.4
ОПП «Комп'ютерна інженерія»
спеціальності 123 «Комп'ютерна інженерія»
_____ Бушуєв Р.В.
«____» _____ 2022р.

Керівник проекту
кандидат технічних наук, доцент
_____ Босько В.В.
«____» _____ 2022 р.
Рецензент _____
_____

м. Кропивницький

Центральноукраїнський національний технічний університет

Факультет *Механіко-технологічний*

Кафедра *Кібербезпеки та програмного забезпечення*

Освітній ступінь *магістр*

Галузь знань *12 "Інформаційні технології"*

Спеціальність *123 "Комп'ютерна інженерія"*

Освітньо-професійна (освітньо-наукова) програма *"Комп'ютерна інженерія"*

ЗАТВЕРДЖУЮ
Завідувач кафедри
д.т.н., проф.
_____Олексій СМІРНОВ
"____" _____20___року

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

Бушуєва Романа Володимировича
(прізвище, ім'я, по батькові)

1. Тема роботи *Дослідження та програмна реалізація застосування об'єктно-орієнтованих баз даних в ІС*

2. Керівник роботи *Босько Віктор Васильович, канд. техн. наук, доцент*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу № 20-13 від 17.08.22

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

4. Мета та завдання випускної кваліфікаційної роботи: *Метою розробки є програмне дослідження та програмна реалізація застосування ООБД в ІС*

5. 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.

6. 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	*2 аркуша*
Діаграма процесів 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	09.11.2022 р.	17.11.2022 р.
Охорона праці	Оришака О.В., к.т.н., доцент	03.11.2022 р.	21.11.2022 р.

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	12.10.2022 р.	
2.	Постановка задачі, оформлення ТЗ	18.10.2022 р.	
3.	Розробка моделі компонента	23.10.2022 р.	
4.	Розробка структур даних	25.10.2022 р.	
5.	Розробка алгоритмів зв’язку та відображення	32.10.2022 р.	
6.	Програмування алгоритмів	11.11.2022 р.	
7.	Розрахунок економічної ефективності	13.11.2022 р.	
8.	Розрахунки з охорони праці та техніки безпеки	16.11.2022 р.	
9.	Оформлення ПЗ	18.11.2022 р.	
10.	Попередній захист роботи	04.12.2022 р	

Дата видачі завдання
«__»_____20 р.

Підпис керівника

_____ _____
(прізвище та ініціали)

Завдання прийнято до виконання
«__»_____20 р.

Підпис здобувача

_____ _____
(прізвище та ініціали)

АНОТАЦІЯ

Бушуєв Р.В. Дослідження та програмна реалізація застосування об'єктно-орієнтованих баз даних в ІС. 123 Комп'ютерна інженерія. Центральноукраїнський національний технічний університет. Кропивницький. 2022.

В даній магістерській роботі розроблено програмне забезпечення, яке призначено для реалізації застосувань об'єктно-орієнтованих баз даних в розробці ПЗ.

Метою розробки є дослідження та програмна реалізація додатку з використанням об'єктно-орієнтованих баз даних .

Об'єктом дослідження є процес реалізації ПЗ з застосуванням ООБЗ.

Предметом дослідження є методи реалізації розробки ПЗ з застосуванням об'єктно-орієнтованих баз даних .

Методи дослідження базуються на методах теорії кодування, методах математичної статистики, методах розробки програмного забезпечення.

Результат роботи – програмна реалізація проекту засобами фреймворку AngularJS та ООБЗ.

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

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

Програма може використовуватися у браузері Chrome, Firefox, Safari. Програму розроблено в середовищі JavaScript та PHP з використанням ООБД VelocityDB та фреймворку AngularJS.

Ключові слова: комп'ютерна інженерія, веб-сайт, AngularJS, бази даних, ООП, ООБД.

ANNOTATION

Bushuev R.V. Research and software implementation of the application of object-oriented databases in IS. 123 Computer engineering. Central Ukrainian National Technical University. Kropyvnytskyi. 2022.

In this master's thesis, software was developed, which is intended for the implementation of object-oriented database applications.

The goal of the development is the research and software implementation of the application using object-oriented databases.

The object of the study is the process of software implementation using OOBZ.

The subject of the study is the implementation methods of software development using object-oriented databases.

Research methods are based on coding theory methods, mathematical statistics methods, and software development methods.

The result of the work is the software implementation of the project using the AngularJS framework and OOBZ.

In the process of working on the software model, an analysis of existing hardware and software was performed. All components of the developed software are fully described.

A convenient user interface has been developed. Instructions for working with software tools are provided.

The program can be used in Chrome, Firefox, and Safari browsers. The program was developed in the JavaScript and PHP environment using the VelocityDB database.

Keywords: computer engineering, website, AngularJS, databases, OOP, OOBD.

ЗМІСТ

					ВКРМ-123.22.0085.00.00.ПЗ	
Вим.	Арк.	№ докум.	Підпис	Дат		
Розроб.		Бушуєв Р.В			*Дослідження та програмна реалізація застосування об'єктно-орієнтованих баз даних в ІС*	Літ. М / Арк. 1 / Аркушів
Перевір.		Босько В.В				
Н. Контр.		Гермак В.С				ЦНТУ КІ-21МЗ
Затверд.		Смірнов О.А.				

ПЕРЕЛІК СКОРОЧЕНЬ, СИМВОЛІВ ТА СПЕЦІАЛЬНИХ ТЕРМІНІВ

ІС – Інформаційна система

ООБД – Об’єктно-Орієнтована База Даних

СУБД – Система Управління Базою Даних

JSON - JavaScript Object Notation

MIME - Multipurpose Internet Mail Extensions

REST - Representational State Transfer

MVC - Model-View-Controller

CSS - Cascading Style Sheets

HTML - Hypertext Markup Language

ОС - операційна система

XML - Extensible Markup Language

CMS - Content Management System

ORM - Object-relational mapping

Sass - Syntactically Awesome Stylesheets

DOM - Document Object Model

EJS - Embedded JavaScript templating

ВСТУП

Актуальність теми. Впровадження інформаційних технологій зробило можливим значно підвищити ефективність роботи підприємств, організацій, проведення наукових досліджень, підвищити якість роботи в різних галузях освіти, науки, культури. Широке практичне використання технологій баз даних обумовлено значними досягненнями у цій галузі провідними комп'ютерними компаніями світу, потребою суспільства в ефективних засобах зберігання і обробки інформації. Отже, зростає потреба у фахівцях, здатних розробляти і застосовувати системи баз даних.

Виникнення технологій баз даних припадає на початок 60-х років. Їх швидкому розвитку сприяли потреби в обробці інформації, досягнення в суміжних областях інформаційних технологій таких, як операційні системи, мови програмування, технічне забезпечення. Спочатку зароджувалися певні ідеї щодо управління ресурсами даних, формувалися основи методології побудови систем баз даних. Із самого початку було зрозуміло, що цей напрям має самостійне значення і буде відігравати одну з ключових ролей у побудові інформаційних систем різного призначення.

Інформаційна система виконує функції збирання, зберігання, розповсюдження і обробки інформації. Під інформацією розуміють будь-які відомості про будь-яку подію, сутність, процес і т.ін., які є об'єктом певних операцій: передачі, перетворення, зберігання або використання. Дані можна визначити, як інформацію зафіксовану у певній формі, яка придатна для подальшої обробки, зберігання і передачі. Предметна область – область застосування конкретної бази даних.

Об'єктно-орієнтована модель – модель даних, яка базується на понятті об'єкта, тобто сутності, що володіє станом і поведінкою. Стан об'єкта визначається його атрибутами, а поведінка визначається сукупністю операцій,

що визначені для цього об'єкта. Також передбачається можливість підтримки зв'язків між типами об'єктів.

Мета й завдання дослідження. Метою роботи є дослідження та програмна реалізація застосування ООБД в ІС.

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

– Огляд існуючих типів БД;

– Дослідження концепції об’єктно-орієнтованих баз даних.

– Програмна реалізація ІС на основі ООБД у якості сховища даних.

– Дослідження ООБД на швидкодію при високому навантаженні на сервер;

– Визначення можливостей та функціоналу ООБД відповідно до сучасних вимог у проектуванні БД;

– Програмна реалізація проекту з застосуванням ООБД.

– Порівняння ООБД з реляційними та графовими БД.

Об’єктом дослідження є процес розробки бази даних використовуючи об’єктно-орієнтовану базу даних .

Предметом дослідження є методи з розробки ООБД.

Методи дослідження базуються на методах теорії кодування, методах побудови архітектури програмного забезпечення, методах розробки програмного забезпечення та методах зберігання даних.

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

– Розроблено та порівняно швидкодії різних типів БД, зроблено висновки про їх переваги та недоліки;

– Розроблено рекомендації по використанню ООБД;

– VelocityDB все частіше використовується для зберігання даних користувачів, замість класичних реляційних баз даних.

Практична цінність отриманих результатів полягає в тому, що розроблені з використанням ООБД веб-сайти та ПЗ інколи значно підвищують швидкість доступу до даних.

Достовірність наукових результатів підтверджена теоретичними викладенням та отриманими даними під час тестування розроблених систем.

Таким чином, виходячи з вищеперерахованого, дослідження та програмна реалізація системи аналізу застосування різних типів БД в сучасних ІС, є актуальною задачею, яка потребує вирішення у даній магістерській роботі. Можна зробити висновок що на сьогодні існують нові більш сучасні та розвинуті БД для створення веб-сайтів та ПЗ, але для цього інколи немає необхідності переходити на нові технології, якщо цього не вимагають поточні або майбутні вимоги.

ВСТУП

Актуальність теми. Впровадження інформаційних технологій зробило можливим значно підвищити ефективність роботи підприємств, організацій, проведення наукових досліджень, підвищити якість роботи в різних галузях освіти, науки, культури. Широке практичне використання технологій баз даних обумовлено значними досягненнями у цій галузі провідними комп'ютерними компаніями світу, потребою суспільства в ефективних засобах зберігання і обробки інформації. Отже, зростає потреба у фахівцях, здатних розробляти і застосовувати системи баз даних.

Виникнення технологій баз даних припадає на початок 60-х років. Їх швидкому розвитку сприяли потреби в обробці інформації, досягнення в суміжних областях інформаційних технологій таких, як операційні системи, мови програмування, технічне забезпечення. Спочатку зароджувалися певні ідеї щодо управління ресурсами даних, формувалися основи методології побудови систем баз даних. Із самого початку було зрозуміло, що цей напрям має самостійне значення і буде відігравати одну з ключових ролей у побудові інформаційних систем різного призначення.

Інформаційна система виконує функції збирання, зберігання, розповсюдження і обробки інформації. Під інформацією розуміють будь-які відомості про будь-яку подію, сутність, процес і т.ін., які є об'єктом певних операцій: передачі, перетворення, зберігання або використання. Дані можна визначити, як інформацію зафіксовану у певній формі, яка придатна для подальшої обробки, зберігання і передачі. Предметна область – область застосування конкретної бази даних.

Об'єктно-орієнтована модель – модель даних, яка базується на понятті об'єкта, тобто сутності, що володіє станом і поведінкою. Стан об'єкта визначається його атрибутами, а поведінка визначається сукупністю операцій,

що визначені для цього об'єкта. Також передбачається можливість підтримки зв'язків між типами об'єктів.

Мета й завдання дослідження. Метою роботи є дослідження та програмна реалізація застосування ООБД в ІС.

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

– Огляд існуючих типів БД;

– Дослідження концепції об’єктно-орієнтованих баз даних.

– Програмна реалізація ІС на основі ООБД у якості сховища даних.

– Дослідження ООБД на швидкодію при високому навантаженні на сервер;

– Визначення можливостей та функціоналу ООБД відповідно до сучасних вимог у проектуванні БД;

– Програмна реалізація проекту з застосуванням ООБД.

– Порівняння ООБД з реляційними та графовими БД.

Об’єктом дослідження є процес розробки бази даних використовуючи об’єктно-орієнтовану базу даних .

Предметом дослідження є методи з розробки ООБД.

Методи дослідження базуються на методах теорії кодування, методах побудови архітектури програмного забезпечення, методах розробки програмного забезпечення та методах зберігання даних.

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

– Розроблено та порівняно швидкодії різних типів БД, зроблено висновки про їх переваги та недоліки;

– Розроблено рекомендації по використанню ООБД;

Практична цінність отриманих результатів полягає в тому, що розроблені з використанням ООБД веб-сайти та ПЗ інколи значно підвищують швидкість доступу до даних.

Достовірність наукових результатів підтверджена теоретичними викладенням та отриманими даними під час тестування розроблених систем.

Таким чином, виходячи з вищеперерахованого, дослідження та програмна реалізація системи аналізу застосування різних типів БД в сучасних ІС, є актуальною задачею, яка потребує вирішення у даній магістерській роботі. Можна зробити висновок що на сьогодні існують нові більш сучасні та розвинуті БД для створення веб-сайтів та ПЗ, але для цього інколи немає необхідності переходити на нові технології, якщо цього не вимагають поточні або майбутні вимоги.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1. Призначення системи

Використання баз даних є однією з характерних особливостей більшості сучасних інформаційних систем. По суті, бази даних — це те, на чому побудована інформаційна система кожної компанії. Тому достатня увага приділяється теорії створення та практиці використання баз даних у період функціонування інформаційних систем. Довгий час основним типом були реляційні бази даних, які сьогодні вже вважаються класичними. Однак розвиток інформаційних систем поставив перед сучасними базами даних завдання, які неможливо вирішити в рамках використання лише реляційних баз даних. Крім класичних завдань, сучасні бази даних повинні забезпечувати багатомашинну обробку та зберігання великих обсягів інформації, оперативний аналіз даних, інтеграцію з Інтернетом, розмежування доступу користувачів до інформації, що зберігається, захист інформації під час її передачі по мережі. Хоча на практиці використовується багато різних баз даних, більшість із них мають ряд загальних характеристик, як з точки зору розробки, так і з точки зору використання. Це дає можливість вивчати сучасні бази даних та відповідне прикладне та системне програмне забезпечення на прикладах, які, незважаючи на свою новизну, вже стали класичними. В магістерській роботі поставлено за мету показати можливості застосування ООБД та проаналізувати їх переваги та недоліки.

1.2. Область застосування

База даних (БД) — це впорядкований набір логічно взаємопов’язаних даних, який є спільним і призначений для задоволення інформаційних потреб

користувачів. У технічному сенсі він також включає систему управління базами даних.

Основним завданням бази даних є гарантоване збереження великих обсягів інформації (так званих записів даних) і забезпечення доступу до них користувача або прикладної програми. Таким чином, база даних складається з двох частин: інформації, що зберігається, і системи управління. Для забезпечення оперативності доступу запис даних організовано як набір фактів (елемент даних).

Дані - відомості, відомості, показники, необхідні для пізнання когось, чогось, характеристики когось, чогось або для прийняття певних висновків, рішень. Інформація відіграє важливу роль у базах даних. Інформація - це абстрактне поняття, яке має різні значення залежно від контексту.

Кожна створена база даних має свою предметну область.

Предметна область - необхідний для розробки бази даних об'єкт, який містить дані, які будуть зберігатися в базі даних.

Модель даних - це абстрактне представлення реального світу, яке відображає лише ті об’єкти, які мають безпосереднє відношення до програми. Це, як правило, визначає певну групу об'єктів, їх атрибутивне значення та зв'язок між ними.

Відомі два підходи до організації інформаційних рядків: файлова організація та організація, подібна до бази даних. Організація файлів передбачає спеціалізацію та зберігання інформації, зазвичай зосереджену на одній прикладній задачі, і забезпечується розробником програми. Така організація дає змогу досягти високої швидкості обробки інформації, але характеризується рядом недоліків. Характерною особливістю доступу до файлів є вузька спеціалізація як програм обробки, так і файлів даних, що є причиною високої надмірності, оскільки одні і ті ж елементи даних зберігаються в різних системах. Оскільки управління здійснюється різними особами (групами осіб), неможливо виявити порушення невідповідності інформації, що зберігається. Файли,

розроблені для спеціалізованих прикладних програм, не можна використовувати для задоволення вимог користувачів, які перетинаються з двома чи більше доменами. Крім того, організація даних файлів через відмінності в структурі записів і форматах передачі даних не забезпечує виконання багатьох інформаційних вимог навіть у випадках, коли всі необхідні елементи даних містяться в існуючих файлах. Тому необхідно відокремити дані від їхнього опису, визначити таку організацію зберігання даних з урахуванням існуючих зв'язків між ними, яка б уможливила одночасне використання цих даних для багатьох програм. Ці причини призвели до появи баз даних.

Базу даних можна визначити як структурований набір даних, який підтримується в активному стані та відображає властивості об'єктів зовнішнього (реального) світу. База даних містить не тільки дані, але й описи даних, і тому інформація про форму зберігання більше не прихована в комбінації «файл-програма», а явно оголошена в базі даних.

База даних орієнтована на інтегровані запити, а не на одну програму, як у випадку доступу до файлів, і використовується для інформаційних потреб багатьох користувачів. У зв'язку з цим бази даних дозволяють значно зменшити надмірність інформації. Перехід від структури бази даних до потрібної структури в програмі користувача відбувається автоматично за допомогою системи керування базами даних (СУБД).

Системи керування базами даних - це програмні засоби, які дозволяють створювати, заповнювати та працювати з базами даних. У світі існує багато різних систем керування базами даних. Багато з них насправді є не готовими продуктами, а спеціалізованими мовами програмування, за допомогою яких кожен, хто вивчає цю мову, може створити потрібні йому структури і ввести в них необхідні елементи управління. До таких мов відносяться Clipper, Paradox, FoxPro та інші.

База даних має бути простою у використанні та відповідати всім стандартам розробки баз даних. Однією з найважливіших функцій бази даних є

швидкість обробки запиту користувача та швидке вивільнення інформації для нього.

База даних також повинна мати зручний інтерфейс користувача. Завдяки цьому користувач може легко шукати інформацію, яка його зацікавить і допоможе вибрати кращий товар для покупки. Під час створення бази даних дуже важливою частиною та етапом розробки була відповідь на запит користувача. Отже, в цій базі даних користувач може отримати зрозумілі та, за потреби, відсортовані дані.

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

2.1. Огляд існуючих систем

Програмне забезпечення для управління та підтримки продуктивності бази даних називається системою керування базами даних (СКБД). СКБД виконує введення даних, перевірку, систематизацію, пошук і обробку даних, роздруковуючи їх у вигляді звітів.

Вибір системи управління базами даних (СУБД) є складним завданням з багатьма параметрами і є одним із важливих етапів розробки додатків баз даних. Обраний програмний продукт повинен відповідати як поточним, так і майбутнім потребам компанії з урахуванням фінансових витрат на придбання необхідного обладнання, самої системи, розробки необхідного програмного забезпечення на її основі, а також навчання персоналу. Крім того, необхідно переконатися, що нова СУБД може принести реальну користь підприємству.

За характером використання СУБД поділяють на персональні (СУБД) і багатокористувацькі (СУБД).

Персональні СУБД включають VISUAL FOXPRO, ACCESS тощо. До багатокористувацьких СУБД відносяться, наприклад, СУБД ORACLE і INFORMIX. Багатокористувацькі СУБД включають сервер БД і клієнтську частину, працюють в неоднорідному комп'ютерному середовищі, допускають роботу з різними типами комп'ютерів і різними операційними системами. Отже, можливе створення інформаційної системи на основі СУБД, яка функціонує за технологією клієнт-сервер.

СУБД — це набір мовних і програмних засобів, призначених для створення, підтримки та використання бази даних.

Персональні СУБД надають можливість створювати персональні бази даних і недорогі додатки, що працюють з ними, а при необхідності створювати додатки, що працюють з сервером баз даних.

Інтерпретатори (оператори) команд і компілятори використовуються для обробки команд користувача або програмних операторів у СУБД. За допомогою компіляторів у багатьох СУБД можна отримати виконувані autonomous-applications-eche-programs.

Забезпечення цілісності бази даних є необхідною умовою успішного функціонування бази даних. Цілісність бази даних - властивість бази даних, яка означає, що база даних містить повну та послідовну інформацію. Щоб забезпечити цілісність бази даних, накладаються обмеження цілісності у вигляді деяких умов, яким повинні відповідати дані, що зберігаються в базі даних. Прикладом таких умов може бути обмеження діапазону можливих значень атрибутів об'єктів, інформація про які зберігається в базі даних, або відсутність повторюваних записів у таблицях реляційної бази даних.

Серед численних СУБД для IBM PC найбільш часто використовуваними програмними пакетами dbase різних версій є Foxbase+, Foxpro, Fox Soft Ware, Clipper, сумісні з dbase за командою та файловою системою.

Наприклад, база даних, створена в одній СУБД, може бути використана в іншій сумісній з нею СУБД, яка має формат файлу dbase (*.dbf). Однак існують інші СУБД, такі як PARADOX і Rbase, які несумісні з dbase. Крім СУБД для DOS існують СУБД, що працюють в середовищі Windows, наприклад Access, MS Works і т.д.

В даний час великою популярністю серед розробників баз даних (БД) користується реляційна СУБД ACCESS, яка входить в пакет Microsoft Office 2003. Приємний інтерфейс і простота налаштування, ефективні інструменти для створення таблиць, форм, запитів, інтеграція з іншими програмами пакету. , засоби організації роботи з базами даних та захисту інформації - це далеко не повний перелік переваг цього додатка.

Основними функціями СУБД є опис структури бази даних, обробка та керування даними.

База даних - сукупність інформації про реальні об’єкти, процеси, події чи явища, пов’язані з певною темою чи завданням, організована таким чином, щоб забезпечити зручне представлення цієї сукупності як у цілому, так і в будь-якій її частині. Реляційна база даних - це набір взаємопов'язаних таблиць, кожна з яких містить інформацію про об'єкти певного типу. Кожен рядок таблиці містить дані про один об’єкт (наприклад, клієнт, машини, документи), а стовпці таблиці містять різні характеристики цих об’єктів – атрибути (наприклад, імена та адреси клієнтів, переїзди та ціни на автомобілі). Рядки таблиці називаються записами, всі записи мають однакову структуру - складаються з полів, в яких зберігаються атрибути об'єкта. Кожне поле в записі містить одну ознаку об'єкта і має строго визначений тип даних (наприклад, текстовий рядок, число, дата). Усі записи мають однакові поля, але містять різні значення атрибутів.

Будь-яка СУБД дозволяє виконувати чотири прості операції над даними:

- додати один або декілька записів до таблиці;
- видалити один або кілька записів з таблиці;
- повернути значення деяких полів в одному або кількох записах;
- знайти один або кілька записів, які задовольняють задану умову.

Для виконання цих операцій використовується механізм запитів. Результатом запиту є або набір записів, відібраних за певними критеріями, або зміна в таблицях.

Кожна прикладна програма являє собою представлення деякої частини реального світу і тому містить її формалізований опис у вигляді даних. Великі набори даних, як правило, зберігаються окремо від виконуваної програми та організовані у вигляді бази даних. З 60-х років для роботи з даними використовуються спеціальні програмні пакети, які називаються системами управління базами даних (СУБД). Системи управління базами даних відповідають за:

- фізичне розміщення даних та їх опис;
- Пошук даних;

- підтримання баз даних в актуальному стані;

- захист даних від некоректного оновлення та несанкціонованого доступу;

обслуговування одночасних запитів даних від кількох користувачів (прикладних програм).

Моделі даних

Те, що зберігається в базі даних, має певну логічну структуру, тобто представлено моделлю, яку підтримує СУБД. Наступні моделі даних є одними з найважливіших:

- ієрархічний;

- мережевий;

- реляційний;

- об'єктно-орієнтований;

В ієрархічній моделі дані представлені у вигляді деревоподібної (ієрархічної) структури. Він підходить для роботи з ієрархічно впорядкованою інформацією та об’ємною для інформації зі складними логічними зв’язками.

Модель мережі означає представлення даних у вигляді довільного графа. Перевагою мережевих та ієрархічних моделей даних є можливість ефективної реалізації ними показників споживання пам’яті та ефективності роботи. Недоліком мережевої моделі даних є висока складність і жорсткість побудованої на її основі схеми бази даних.

Реляційна модель даних (RMD) названа за англійським терміном Relation. Модель даних описує певний набір загальних концепцій і характеристик, якими повинні володіти всі конкретні СУБД і БД, якими вони керують, якщо вони засновані на цій моделі.

Об’єктно-орієнтована модель даних – це коли база даних зберігає не лише дані, а й методи їх обробки у вигляді програмного коду. Це перспективний напрямок, який ще не отримав широкого поширення через складність створення та використання такої СУБД.

База даних - це сукупність записів різних типів, які містять перехресні посилання.

Файл - це набір однотипних записів, у яких відсутні перехресні посилання.

Крім того, у визначенні не згадується архітектура комп’ютера. Справа в тому, що, хоча в більшості випадків база даних дійсно містить один або (частіше) кілька файлів, їх фізична організація істотно відрізняється від логічної. Таблиці можна зберігати як окремими файлами, так і всі разом. І навпаки, іноді для зберігання однієї таблиці використовується декілька файлів. Додаткові спеціальні файли зазвичай виділяються для підтримки перехресних посилань і швидкого пошуку.

Тому при роботі з базами даних зазвичай використовують поняття більш високого логічного рівня: запис і таблиця, не вдаючись у подробиці їх фізичної структури.

Отже, сама база даних — це просто набір таблиць із перехресними посиланнями. Для того щоб універсально витягувати з нього групи записів, обробляти їх, змінювати і видаляти, потрібні спеціальні програми під назвою СУБД.

Ієрархічна модель

В ієрархічній моделі зв'язки між даними можна описати за допомогою упорядкованого графа (чи дерева). Спрощене представлення зв'язків між даними в ієрархічній моделі показане на рисунку 2.1

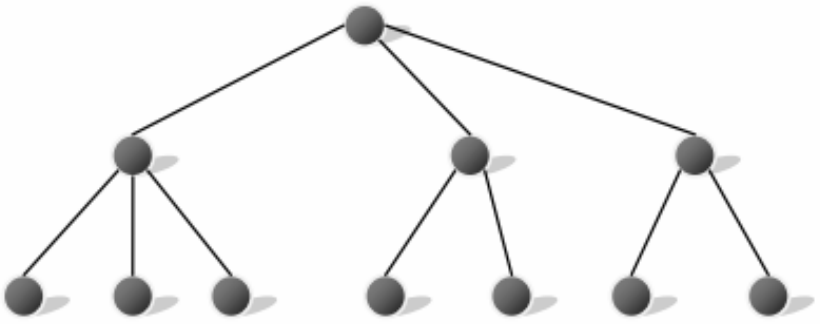

Рисунок 2.1 – Представлення зв’язків в ієрархічній моделі даних

Деякі мови програмування використовують тип даних «дерево» для опису структури (схеми) ієрархічної бази даних. Тип дерева відповідає об'єктно-орієнтованому підходу до програмування. Дозволяє вкладати типи, кожен на певному рівні. Тип «дерево» — композитний. Він містить підтипи («піддерева»), кожен з яких, у свою чергу, є типом «дерева». Кожен із типів «дерева» складається з одного «кореневого» типу та впорядкованого набору (можливо порожніх) дочірніх типів. Кожен з елементарних типів, включених до типу "дерево", є простим або складним типом "запису". Простий «запис» складається з одного типу, наприклад числового, тоді як складний «запис» поєднує певну комбінацію типів, наприклад ціле число, рядок і посилання. Приклад типу «дерево» як набору типів показано на рисунку 2.2.

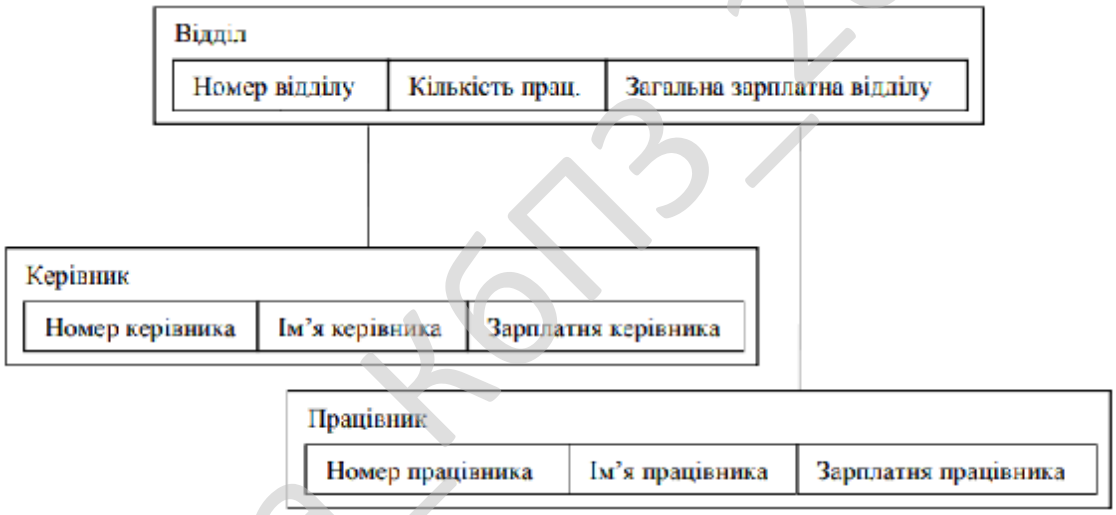

Рисунок 2.2 – Приклад типу «дерево»

Базовий тип — це тип, який має дочірні типи і сам по собі не є підтипом. Підпорядкований тип (підтип) є нащадком типу, який діє як його предок (батько). Нащадки одного типу є близнюками по відношенню один до одного.

Загалом тип «дерева» — це ієрархічно організований набір типів «записів». Ієрархічна БД — це впорядкована сукупність екземплярів даних типу «дерево» (tree), яка містить екземпляри типу «запис» (record). Часто зв'язок між типами переноситься на зв'язок між самими записами. Поля запису зберігають власні числові або символьні значення, які утворюють основний вміст бази

даних. Обхід усіх елементів ієрархічної бази даних зазвичай здійснюється зверху вниз і зліва направо. В ієрархічних СУБД може використовуватися термінологія, відмінна від наведеної. Таким чином, в системі IMS термін «запис» відповідає терміну «сегмент», а «запис бази даних» відноситься до всієї сукупності записів, що належать одному екземпляру типу «дерево». Дані в базі даних із заданою схемою (рисунок 2.2) можуть виглядати, наприклад, як показано на рисунку 2.3.

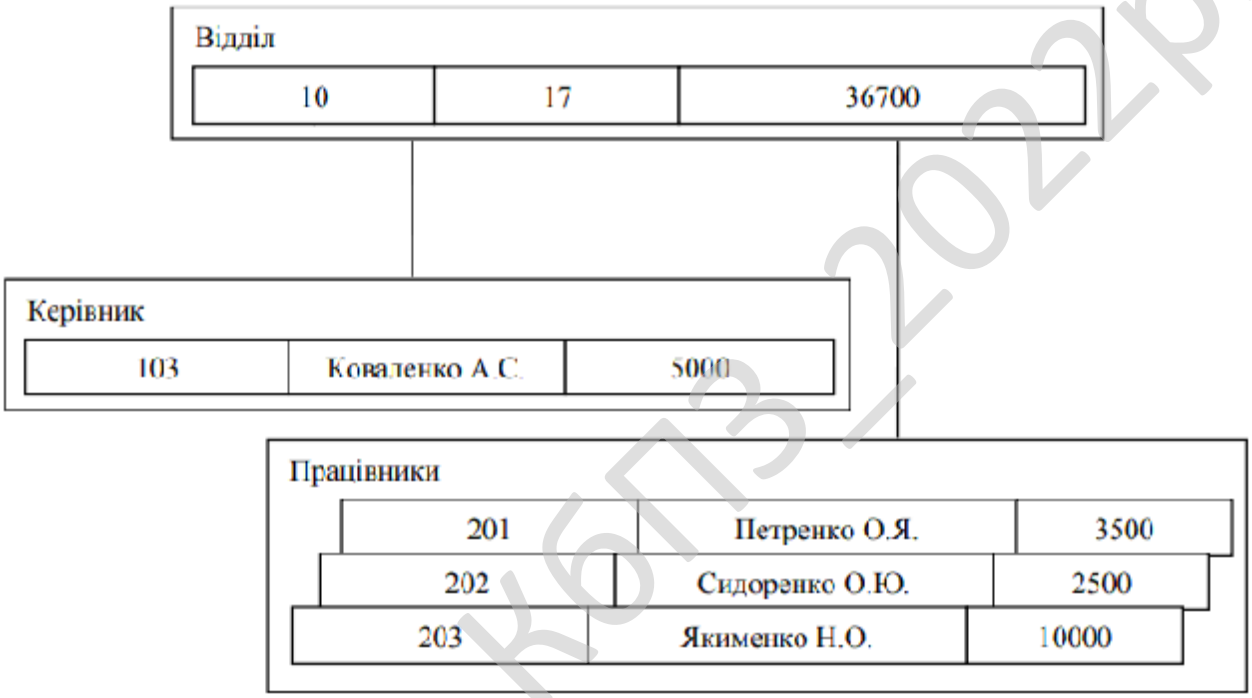

Рисунок 2.3 – Дані в ієрархічній базі

Для організації фізичного зберігання ієрархічних даних у пам’яті комп’ютера можна використовувати наступні групи методів:

- лінійне відображення списку з послідовним виділенням пам'яті (адресна арифметика, структури лівого списку);
- представлення через зв'язані лінійні списки (методи з використанням покажчиків і каталогів).

Основні операції для роботи з ієрархічно організованими даними включають наступне:

- пошук зазначеного екземпляра БД (наприклад, дерева зі значенням 10 у полі «Номер відділу»);

- перехід від одного дерева до іншого;

- перехід від одного запису до іншого в межах дерева (наприклад, до наступного запису типу «Працевлаштований»); вставляти новий запис у вказану позицію;

- видалення поточного запису тощо.

Відповідно до визначення типу «дерево» можна зробити висновок, що контроль цілісності посилань автоматично підтримується між предками та нащадками. Основне правило контролю цілісності формулюється так: нащадок не може існувати без батька, а деякі батьки можуть не мати нащадків. Не існує механізмів підтримки цілісності зв'язків між записами різних дерев.

Переваги ієрархічної моделі даних включають ефективне використання пам’яті комп’ютера та хороший час виконання основних операцій з даними. Ієрархічна модель даних підходить для роботи з ієрархічно організованою інформацією.

Недоліком ієрархічної моделі є її громіздкість для обробки інформації з досить складними логічними зв'язками, а також складність розуміння для звичайного користувача. Відносно обмежена кількість СУБД заснована на ієрархічній моделі даних, включаючи іноземні системи IMS, PC/Focus, Tcani-Lp і Data Edge, а також вітчизняні системи Oka, INES і MIRIS.

Мережева модель

Мережева модель даних дозволяє відображати різноманітні взаємозв'язки елементів даних у виді довільного графа, узагальнюючи тим самим ієрархічну модель даних (рисунок. 2.4)

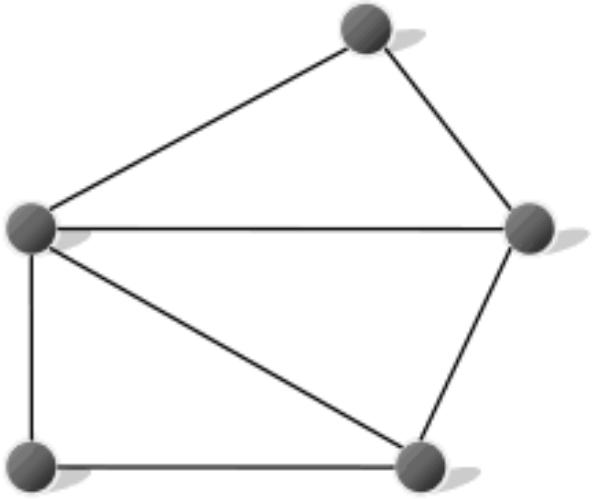

Рисунок 2.4 – Представлення зв’язків у мережевій моделі

Для опису схеми мережевої бази даних використовуються дві групи типів: «запис» і «зв'язок». Тип «відношення» визначено для двох типів «записів»: предка та нащадка. Змінні типу "посилання" є прикладами посилань.

Мережева база даних складається з набору записів і набору відповідних посилань. Ніяких особливих обмежень на формування підключення не накладається. Якщо в ієрархічних структурах запис-нащадок може мати лише один запис-предка, то в мережевій моделі даних запис-нащадок може мати довільну кількість записів-предка (предка).

Приклад схеми найпростішої оперативної бази даних наведено на рисунку 2.5. Типи посилань тут позначаються мітками на лініях, що з’єднують типи записів.

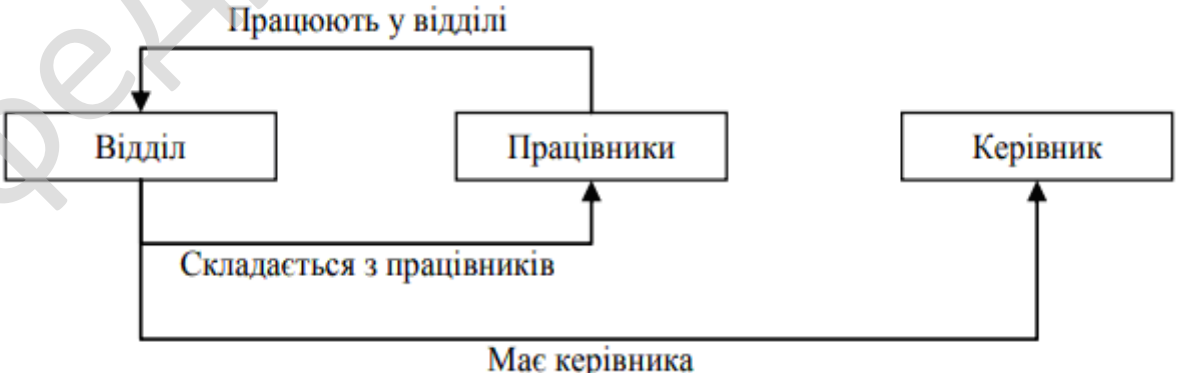

Рисунок 2.5 – Приклад схеми мережевої БД

У різних СУБД грід-типу різні терміни часто використовуються для позначення, по суті, однакових понять. Наприклад, такі як: елементи даних та агрегати, записи, набори, області тощо.

Фізичне розміщення даних у базах даних мережевого типу можна організувати за допомогою майже тих же методів, що й в ієрархічних базах даних. Серед найважливіших операцій маніпулювання даними в базах даних сіткового типу є наступні:

- пошук записів у базі даних;
- перехід від предка до першого нащадка;
- перехід від нащадка до предка;
- створення нового запису;
- видалення поточного запису;
- відновлення існуючого запису;
- включення записів у спілкування;
- виключення записів з підключення;
- зміна з'єднань та ін.

Перевагою мережевої моделі даних є можливість ефективної реалізації з точки зору споживання пам'яті та ефективності. У порівнянні з ієрархічною моделлю, мережева модель пропонує великі можливості щодо допустимості формування довільних зв'язків.

Недоліком мережевої моделі даних є велика складність і жорсткість побудованої на її основі схеми бази даних, а також складність для звичайного користувача зрозуміти та обробити інформацію в базі даних. Крім того, мережева модель даних має слабкі елементи керування. цілісність зв'язків за рахунок допустимості встановлення довільних зв'язків між записами. Системи на основі мережевої моделі не отримали широкого застосування на практиці. Найвідоміші онлайн-бази даних: IDMS, db_VistaII, NETVORK, SETOR і KOMPAS.

Реляційна модель даних

Реляційна модель даних була запропонована співробітником IBM Едгаром Кодом і заснована на концепції відношення. Відношення — це набір елементів, які називаються кортежами. Звична для людського сприйняття двовимірна таблиця є візуальним відображенням взаємозв'язків. Таблиця має рядки (записи) і стовпці (стовпці). Кожен рядок таблиці має однакову структуру і складається з полів. Кортежі відповідають рядкам таблиці, а атрибути зв’язку – стовпцям. За допомогою однієї таблиці зручно описати найпростіший вид зв'язку між даними, тобто поділ одного об'єкта (явища, сутності, системи тощо), інформація про який зберігається в таблиці, на кілька підоб'єктів, кожен з яких відповідає рядку або запису таблиці. У той же час кожен із підоб’єктів має однакову структуру або властивості, описані відповідними значеннями поля запису. Наприклад, таблиця може містити інформацію про групу студентів, кожен з яких має такі характеристики; прізвище, ім'я та по батькові, стать, вік та освіта. Оскільки в рамках однієї таблиці неможливо описати більш складні логічні структури даних з предметної області, використовується зв'язування таблиць. Фізичне розміщення даних у реляційних базах даних на зовнішніх носіях легко виконується за допомогою простих файлів. Перевага реляційної моделі даних полягає в її простоті, зрозумілості та легкості фізичної реалізації на комп’ютері. Простота і зрозумілість для користувача стала основною причиною його широкого використання. Проблеми ефективності обробки даних такого типу виявилися технічно цілком вирішуваними. Основними недоліками реляційної моделі є відсутність стандартних засобів ідентифікації окремих записів і складність опису ієрархічних і мережевих зв'язків. Прикладами іноземних реляційних баз даних для персональних комп’ютерів є: Access (Microsoft), Clarion (Clarion Software), Ingres (ASK Computer Systems), Oracle (Oracle), Visual FokPro, dBaseIII Plus і dBase II (Ashton-Tate Company), DV2 (IBM), R: BASE (Microrim), FoxPro ранні версії, FoxBase (Fox Software), Paradok і dBASE для Windows (Borland)... До реляційного типу СУБД належать системи:

PALMA (ІЦ АН СРСР).), а також система NuTesh (MIT). Відзначимо, що останні версії реляційних баз даних мають деякі властивості об'єктно-орієнтованих систем. Такі бази даних часто називають об'єктно-реляційними. Прикладом такої системи є продукти Oracle 8.k+. Системи до Oracle 7.k вважаються «чисто» реляційними.

Постреляційна модель

Класична реляційна модель передбачає подільність даних, що зберігаються в полях записів таблиці. Це означає, що інформація в таблиці представлена в першій нормальній формі. Є ряд випадків, коли це обмеження перешкоджає ефективній реалізації програм.

Постреляційна модель даних — це розширена реляційна модель, яка усуває обмеження на подільність даних, що зберігаються в записах таблиці. Постреляційна модель даних допускає багатозначні поля – поля, значення яких складаються з підзначень. Набір значень багатозначного поля вважається незалежною таблицею, вбудованою в основну таблицю.

На рисунках 2.6 і 2.7 показано ті самі дані з використанням реляційної (рис. 2.6) і постреляційної (рис. 2.7) моделей як приклад інформації про рахунки-фактури та товари для порівняння. Таблиця «РАХУНОК» містить дані про номери рахунків («Номер рахунку») та номери клієнтів («Номер клієнта»). Таблиця «НАКЛАДНІ-ТОВАР» містить дані по кожному з накладних: «Номер накладної», «Найменування товару», «Кількість товару». Таблиця «Накладна» пов'язана з таблицею «Накладна-товар» полем «Номер накладної».

Як видно з малюнка, порівняно з реляційною моделлю, в постреляційній моделі дані зберігаються більш ефективно, і під час обробки не потрібно виконувати операцію з’єднання даних з двох таблиць. Як доказ, на малюнку 2.8 показано приклади команд SELECT для вибору даних з усіх полів бази даних у SQL для реляційної (a) і постреляційної (b) моделей. Окрім надання вкладених полів, постреляційна модель підтримує багатозначні (багатогрупові) асоційовані поля. Набір асоційованих полів називається асоціацією. При цьому в рядку

перше значення одного стовпця асоціації відповідає першому значенню всіх інших стовпців асоціації. Усі інші значення стовпців тощо. з'єднані подібним чином. Довжина полів і кількість полів у записах таблиці не підлягають вимогам постійності. Це означає, що структури даних і таблиці мають велику гнучкість.

НАКЛАДНІ

Номер накладної	Номер покупця
0373	8723
8374	8232
7364	8723

НАКЛАДНІ-ТОВАРИ

Номер накладної	Назва товару	Кількість товару
0373	Сир	3
0373	Риба	2
8374	Лимонад	1
8374	Сік	6
8374	Печиво	2
7364	Йогурт	1

Рисунок 2.6 – Представлення даних за допомогою реляційної моделі

НАКЛАДНІ

Номер накладної	Номер покупця	Назва товару	Кількість товару
0373	8723	Сир	3
		Риба	2
8374	8232	Лимонад	1
		Сік	6
		Печиво	2
7364	8723	Йогурт	1

Рисунок 2.7 – Представлення даних за допомогою постреляційної моделі

```
а) SELECT
«НАКЛАДНІ».«Номер накладної», «Номер покупця», «Назва товару», «Кількість товару» FROM
«НАКЛАДНІ», «НАКЛАДНІ-ТОВАРИ» WHERE
«НАКЛАДНІ».«Номер накладної» = «НАКЛАДНІ-ТОВАРИ».«Номер накладної»;
```

Рисункок 2.8(а) – Оператори SQL для реляційної моделі

```
6) SELECT
«Номер накладної», «Номер покупця», «Назва товару», «Кількість товару» FROM
«НАКЛАДНІ»;
```

Рисунок 2.8(б) -– Оператори SQL для постреляційної моделі

Оскільки постреляційна модель дозволяє зберігати ненормалізовані дані в таблицях, виникає проблема забезпечення цілісності та узгодженості даних. Ця проблема вирішується шляхом включення в СУБД механізмів, схожих на збережені процедури в клієнт-серверних системах. Для опису функцій керування значеннями в полях можна створювати процедури (коди перетворення та коди кореляції), які автоматично викликаються до або після доступу до даних. Кореляційні коди виконуються відразу після зчитування даних, перед їх обробкою. Коди перетворення, з іншого боку, виконуються після обробки даних. Перевагою постреляційної моделі є можливість представити набір пов'язаних реляційних таблиць за допомогою однієї постреляційної таблиці. Це забезпечує високу наочність подання інформації та підвищує ефективність її обробки. Недоліком постреляційної моделі є складність вирішення проблеми забезпечення цілісності та узгодженості збережених даних. Розглянута постреляційна модель даних підтримується СУБД uniVers. Серед інших СУБД, заснованих на постреляційній моделі даних, також є системи Bubba і Dasdb.

Багатовимірна модель

Багатовимірний підхід до представлення даних в базі даних з'явився майже одночасно з реляційним, але до недавнього часу дійсно функціональних багатовимірних СУБД (БВСКБД) було дуже мало. Проте з середини 90-х років інтерес до них почав набувати масового характеру. У 1993 р. стимулом послужила програмна стаття одного з основоположників реляційного підходу Е. Код. У ньому сформульовано 12 основних вимог до систем класу OLAP (OnLine Analytical Processing), найважливіші з яких пов'язані з можливостями концептуального представлення та обробки багатовимірних даних.

Багатовимірні системи дозволяють швидко обробляти інформацію для аналізу та прийняття рішень. У розвитку концепції ІБ можна виділити наступні два напрямки:

- операційні (транзакційні) системи обробки;

- системи аналітичної обробки (системи підтримки прийняття рішень).

Реляційні СУБД призначалися для оперативної обробки інформації інформаційних систем і були дуже ефективні в цій області. У системах аналітичної обробки вони виявилися дещо незграбними і недостатньо гнучкими. Більш ефективними тут є багатовимірні СУБД (БВСКБД).Багатовимірні СКБД є вузькоспеціалізованими СКБД, призначеними для інтерактивної аналітичної обробки інформації. Розкриємо основні поняття, використовувані в цих СКБД: агрегуємість, історичність та прогнозованість даних.

Агрегування даних передбачає розгляд інформації на різних рівнях їх узагальнення. В інформаційних системах рівень деталізації подання інформації для користувача залежить від його рівня:

- аналітик;

- користувач-оператор;

- менеджер;

- менеджер

Історичність даних передбачає забезпечення високого рівня статичності (незмінності) реальних даних та їхніх зв'язків, а також обов'язковість зв'язку даних із часом. Статичний характер даних дозволяє використовувати при їх обробці спеціалізовані методи завантаження, зберігання, індексації та вибірки. Тимчасова зв'язка даних необхідна для частого виконання запитів, які мають вибіркові значення часу та дати. Необхідність упорядковувати дані за часом у процесі обробки та представлення даних користувачеві висуває вимоги до механізмів зберігання та доступу до інформації. Тому, щоб скоротити час обробки запитів, бажано завжди сортувати дані в тому порядку, в якому вони найчастіше запитуються.

Передбачуваність даних означає встановлення функцій прогнозу та їх застосування до різних інтервалів часу.

Багатовимірність моделі даних означає не багатовимірність візуалізації цифрових даних, а багатовимірність логічного представлення інформаційної структури при описі та в операціях маніпулювання даними. Порівняно з реляційною моделлю багатовимірна організація даних має більшу наочність та інформативність. Для ілюстрації на рисунку 2.9 показано реляційне (а) і багатовимірне (б) представлення тих самих даних про продаж автомобілів.

Модель	Місяць	Обсяг
«BMW»	Червень	12
«BMW»	Липень	24
«BMW»	Серпень	5
«Mercedes»	Червень	2
«Mercedes»	Липень	18
«Toyota»	Липень	19

Модель	Червень	Липень	Серпень
«BMW»	12	24	5
«Mercedes»	2	18	-
«Toyota»	-	19	-

Рисунок 2.9 – Реляційне і багатовимірне представлення даних

Якщо це багатовимірна модель з розмірністю більше двох, то інформація не обов'язково візуально представляється у вигляді багатовимірних об'єктів (три-, чотири- і більше вимірних гіперкубів). У цих випадках користувачеві зручніше мати справу з двовимірними таблицями або графіками. При цьому дані є «зрізами» (точніше, «фрагментами») з багатовимірного сховища даних, зроблених з різним ступенем деталізації. Давайте розглянемо основні поняття МЩКТ, які включають вимірювання та клітину.

Розмір — це набір даних одного типу, які складають одну з граней гіперкуба. Отже, прикладами загальновживаних вимірювань часу є дні, місяці, квартали та роки. Міста, округи, регіони та країни широко використовуються як географічні виміри. У багатовимірній моделі даних вимірювання діють як індекси, які визначають конкретні значення в комірках гіперкуба.

Комірка або індикатор — це поле, значення якого однозначно визначається фіксованим набором розмірів. Тип поля найчастіше визначається як числове. Залежно від того, як генеруються значення клітинок, зазвичай це може бути змінна (значення змінюються та можуть бути завантажені із зовнішнього джерела даних або згенеровані програмно) або формула (значення, подібні до клітинок формул у таблицях, є розраховується за попередньо визначеними формулами). В існуючому БМСКБД використовуються два основних варіанти (схеми) організації даних: гіперкуб і полікуб.

У полікубічній схемі передбачається, що кілька гіперкубів різних розмірів і різних розмірів можуть бути визначені в базі даних як грані. Прикладом системи, яка підтримує полікубічну версію бази даних, є Oracle Express Server.

У випадку схеми гіперкуба передбачається, що всі показники визначаються одним і тим же набором вимірювань. Це означає, що якщо є кілька гіперкубів БД, усі вони мають однаковий розмір і відповідні вимірювання. Очевидно, що в деяких випадках інформація в базі може бути надмірною (якщо потрібне обов'язкове заповнення клітинок). У випадку багатовимірної моделі даних використовується низка спеціальних операцій, серед яких: формування «зрізу», «обертання», агрегування та деталізація.

«Зріз» — це підмножина гіперкуба, отримана шляхом фіксації одного або кількох вимірів. Формування «зрізів» здійснюється для обмеження значень, використовуваних користувачем, оскільки всі значення гіперкуба майже ніколи не використовуються одночасно.

Операція «обертання» (Rotate) використовується в двовимірному представленні даних. Його суть полягає в зміні порядку вимірювань при візуальному відображенні даних.

Операції «агрегування» (Drill Up) і «деталізація» (Drill Down) означають, відповідно, перехід до більш загального і до більш детального відображення інформації користувачеві з гіперкуба.

Основною перевагою багатовимірної моделі даних є зручність і ефективність аналітичної обробки великих обсягів даних, пов'язаних з часом. При організації обробки подібних даних на основі реляційної моделі спостерігається нелінійне зростання складності операцій в залежності від розміру бази даних і значне збільшення витрат оперативної пам'яті на індексацію. Недоліком багатовимірної моделі даних є її громіздкість для найпростіших завдань звичайної оперативної обробки інформації. Прикладами систем, які підтримують багатовимірні моделі даних, є Essbase (Arbor Software), Media Multi-matrix (Speedware), Oracle Express Server (Oracle) і Cache (InterSistems).

Деякі програмні продукти, наприклад Menia / MR (Speedvarc), дозволяють одночасно працювати з багатовимірними і реляційними базами даних. У кеш-СУБД, в якій внутрішньою моделлю даних є багатовимірна модель, реалізовано три методи доступу до даних: прямий (на рівні вузлів багатовимірних масивів), об'єктний і реляційний.

Об'єктно-орієнтована модель

В об'єктно-орієнтованій моделі при представленні даних можлива ідентифікація окремих записів бази даних. Зв'язки встановлюються між записами бази даних і функціями їх обробки за допомогою механізмів, подібних до відповідних засобів в мовах об'єктно-орієнтованого програмування.

Стандартизована об'єктно-орієнтована модель описана в рекомендаціях стандарту ODMG-93 (Object Database Management Group). Повністю реалізувати рекомендації ОДМГ-93 поки не вдається. Щоб проілюструвати ключові ідеї, розглянемо трохи спрощену модель об'єктно-орієнтованої бази даних.

Структуру об'єктно-орієнтованої бази даних (OODB) можна графічно представити у вигляді дерева, вузлами якого є об'єкти. Властивості об'єкта описуються деяким стандартним типом (наприклад, рядок) або створеним користувачем типом (визначеним як клас).

Значенням властивості string є рядок символів. Значенням властивості класу є об’єкт, який є екземпляром відповідного класу. Кожен об'єкт-примірник класу вважається нащадком об'єкта, в якому він визначений як властивість. Екземпляр об’єкта класу належить до власного класу та має одного батька. Загальні зв’язки в базі даних утворюють узгоджену ієрархію об’єктів.

Приклад логічної структури ООБД «Бібліотека» наведено на рисунку 2.10.

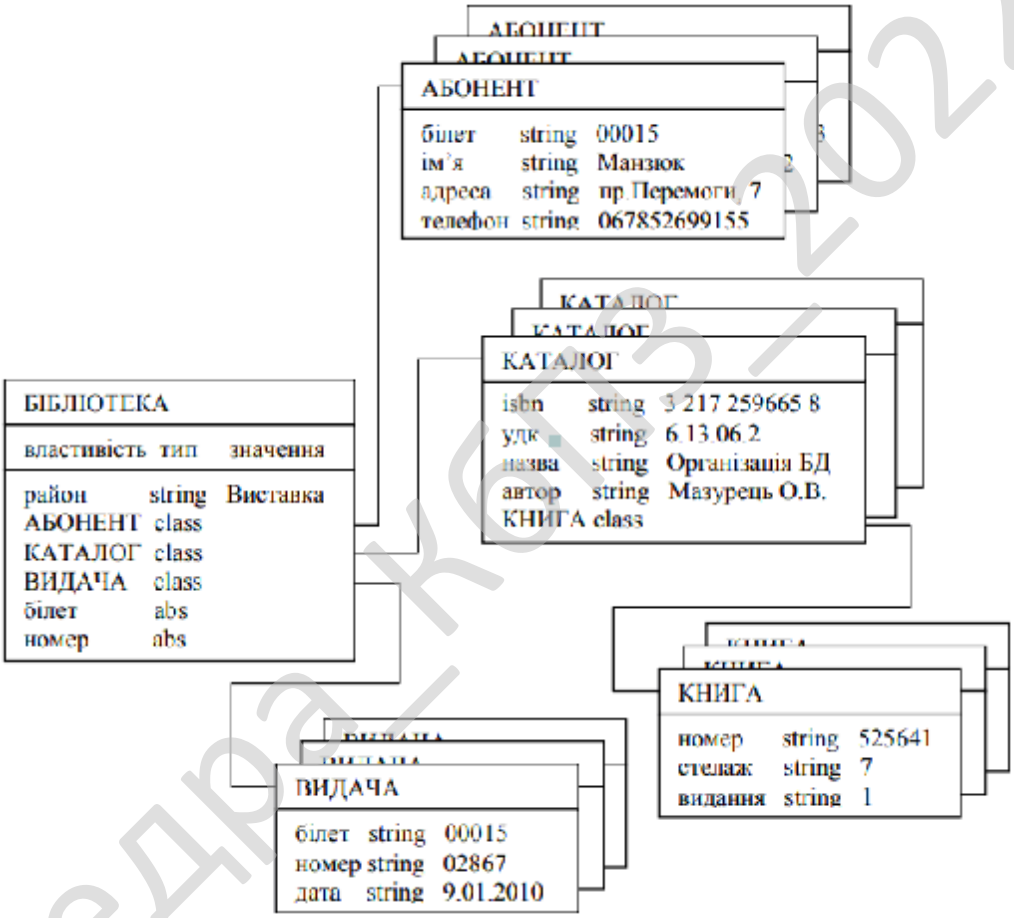

Рисунок 2.10 – Логічна структура ООБД «Бібліотека»

Об'єктно-орієнтовані механізми інкапсуляції, успадкування та поліморфізму. Операції, подібні до команд SQL (наприклад, створення бази даних), можна використовувати обмежено.

Створення та модифікація бази даних супроводжується автоматичним формуванням і подальшою корекцією індексів (індексних таблиць), що містять

інформацію для швидкого пошуку даних. Давайте коротко розглянемо концепції інкапсуляції, успадкування та поліморфізму стосовно об’єктно-орієнтованої моделі бази даних.

Інкапсуляція обмежує область дії назви властивості межами об’єкта, в якому вона визначена. Отже, якщо ми додамо властивість, яка містить номер телефону автора книги, до об’єкта типу CATALOG і маємо назву phone, то ми отримаємо однойменні властивості в об’єктах SUBSCRIBER і CATALOG. Значення такої властивості буде визначатися об'єктом, в якому воно інкапсульоване.

З іншого боку, спадкування розширює сферу власності на всіх нащадків об’єкта. Отже, всім об’єктам типу КНИГА, які є нащадками об’єктів типу КАТАЛОГ, можна присвоїти властивості батьківського об’єкта: isbn, УДК, назву та автора. Якщо необхідно поширити дію механізму успадкування на об’єкти, які не є безпосередніми родичами (наприклад, між двома нащадками одного батька), то в загальному предку визначається абстрактна властивість типу ahs. Таким чином, визначення карти абстрактних властивостей і числа в об'єкті БІБЛІОТЕКА призводить до успадкування цих властивостей усіма підлеглими об'єктами ПЕРЕДПЛАНЕЦЬ, КНИГА і ВИДАННЯ. Не випадково значення властивостей квитка класів SUBSCRIBER і ISSUER, зображене на малюнку, буде однаковим - 00015.

Поліморфізм в об'єктно-орієнтованих мовах програмування означає здатність одного і того ж програмного коду працювати з різними типами даних. Іншими словами, це означає визнання існування методів (процедур або функцій) з однаковими іменами в об'єктах різних типів. Під час виконання об’єктної програми одні й ті ж методи діють на різних об’єктах залежно від типу аргументу. З точки зору нашого OODB, поліморфізм означає, що об’єкти класу BOOK, які мають відмінні від батьків класу CATALOG, можуть мати інший набір властивостей. Тому програми для роботи з об'єктами класу BOOK можуть містити поліморфний код.

Пошук в об'єктно-орієнтованій базі даних полягає в пошуку подібності між об'єктом, указаним користувачем, і об'єктами, що зберігаються в базі даних. Визначений користувачем об’єкт (цільовий об’єкт, властивість об’єкта має тип goat), загалом, може представляти підмножину всієї ієрархії об’єктів, що зберігаються в базі даних. Цільовий об'єкт, а також результат запиту можуть зберігатися в базі даних. Приклад запиту на зчитування номерів білетів та прізвищ абонентів, які отримали хоча б одну книгу в бібліотеці, наведено на рисунку 2.11.

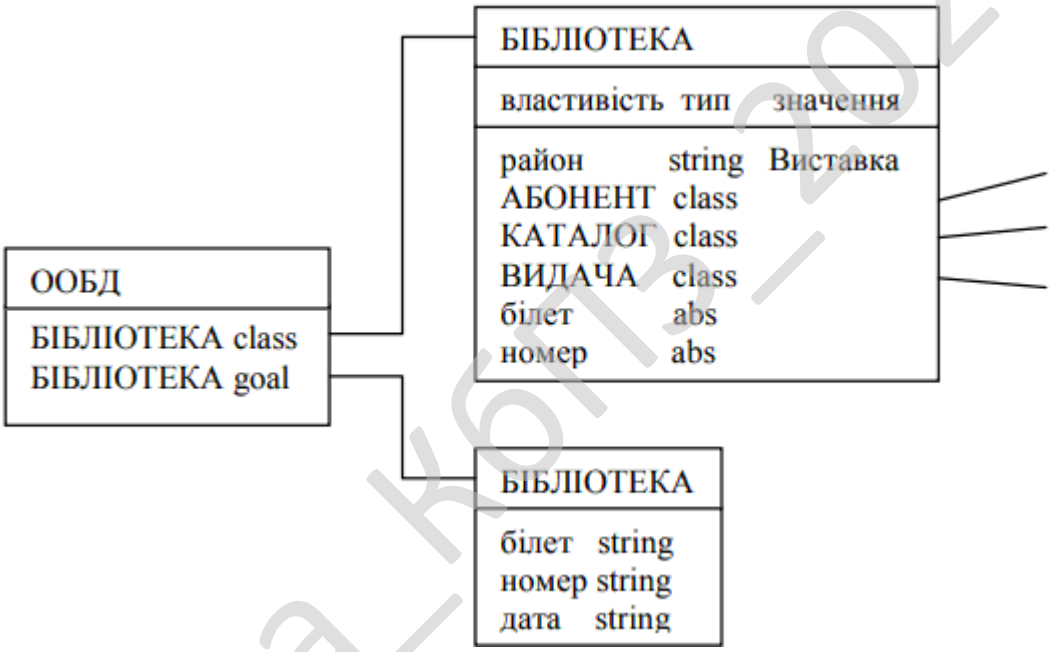

Рисунок 2.11 – Фрагмент ООБД з об’єктом – метою

Головна перевага об’єктно-орієнтованої моделі даних порівняно з реляційною – можливість відображення інформації про складні взаємозв’язки об’єктів. Об'єктно-орієнтована модель даних дозволяє ідентифікувати певний запис бази даних і визначити функції його обробки.

Недоліками об'єктно-орієнтованої моделі є висока концептуальна складність, незручність обробки даних і низька швидкість виконання запитів. У 1990-х роках існували лише експериментальні прототипи об'єктно-орієнтованих систем управління базами даних. На сьогоднішній день такі системи набули значного

поширення, до них зокрема відносяться такі СУБД: ROET (ROET Software), Jasmine (Computer Associates), Versant (Versant Technologies), 02 (Ardent Software), ODB-Jupiter (Науково-виробничий центр «Інтелтек»). Plus"), а також Iris, Orion і Postgres.

2.2 Обґрунтування вибору методів розробки

Більшість інформаційних систем сьогодні зберігають дані в сховищах, які називаються базами даних. Найпопулярніші бази даних - Microsoft SQL Server, Oracle, PostgreSQL або MySQL - використовують реляційну модель представлення даних. У цих базах даних сутності представлені у вигляді рядків таблиці, їх параметри — у вигляді стовпців, а зв’язок між сутностями різних типів здійснюється за допомогою відношення «один до одного» або «один до багатьох».

Типи даних, які можуть мати стовпці, обмежені основними типами (ціле, подвійне, varchar, текст). Системи керування базами даних зазвичай не дозволяють розширити систему типів шляхом додавання власного типу даних.

Реляційні бази даних мають власний математичний апарат. Для реляційних баз даних свого часу Едгар Код заклав основи математичного апарату реляційної алгебри. Цей математичний інструмент пояснює, як виконувати основні операції зі зв’язками бази даних.

У 1986 році був прийнятий перший стандарт SKL-86, який визначив всю долю реляційних баз даних. Після прийняття стандарту всі розробники реляційних СУБД були зобов'язані йому слідувати. Так, реляційні бази даних зазвичай використовують SQL як мову запитів.

Реляційні бази даних дуже успішні на ринку, і було написано багато програм, які використовують ці бази даних як сховище даних. У багато з цих продуктів вкладено багато грошей, і клієнти продовжуватимуть інвестувати в їх розвиток.

Але традиційні бази даних не завжди є ідеальними для зберігання даних. Так, іноді важко використовувати реляційні бази даних, де програма може мати, наприклад, типи даних для великих і неструктурованих мультимедійних об’єктів, географічних інформаційних систем або систем автоматизованого проектування/автоматизованого виробництва.

Складність полягає в тому, що, по-перше, необхідно розробити не тільки архітектуру класів програми, але і схему бази даних. По-друге, також необхідно приділити час розробці функціональності, яка зможе відображати табличні дані у вигляді об’єктів класу і навпаки.

Проаналізувавши результати, можна зробити висновок, що всі бази даних є життєздатними та дають хороші результати, особливо на невеликих обсягах даних.

Концепція ООБД

Об'єктно-орієнтовані бази даних - бази даних, в яких інформація представлена у вигляді об'єктів, як в об'єктно-орієнтованих мовах програмування.

Причиною появи об'єктно-орієнтованих систем баз даних була потреба в більш адекватному представленні та моделюванні об'єктів реального світу, оскільки OOBD забезпечують набагато більш розвинену модель даних, ніж традиційні реляційні бази даних.

Парадигма OODB базується на низці основних понять, таких як об’єкт, ідентифікатор об’єкта, клас, успадкування, перевантаження та пізнє зв’язування. Кожен об'єкт при створенні отримує унікальний ідентифікатор, згенерований системою, який пов'язаний з об'єктом протягом його існування і не змінюється при зміні стану об'єкта. Кожен об'єкт має стан і поведінку.

1. Стан об'єкта - це набір значень його атрибутів.

2. Значенням атрибута об’єкта також є певний об’єкт або набір об’єктів.

3. Поведінка об’єкта – набір методів (програмний код), які діють на стан об’єкта.

Об'єктно-орієнтовані бази даних зазвичай рекомендуються для тих випадків, коли потрібна високопродуктивна обробка даних зі складною структурою.

Основні труднощі об'єктно-орієнтованого моделювання даних є наслідком того, що настільки розвиненого математичного апарату, на якому могла б базуватися загальна об'єктно-орієнтована модель даних, не існує.

Розгляд характеристик OOBD привів до визначення OOBD, яке було представлено на Першій міжнародній конференції з дедуктивних та об’єктно-орієнтованих баз даних у формі маніфесту в 1989 році. Він представляє можливості об'єктно-орієнтованих баз даних.

Обов'язкові характеристики ООБД:

1. Супровід складних об'єктів

Механізм складеного об’єкта дозволяє об’єкту мати атрибут, який також може бути об’єктом. Атрибутами можуть бути рядки, списки, хеш-таблиці тощо.

2. Ідентифікація об'єктів

Кожна сутність у базі даних має унікальний OID. Це властивість об’єкта, яка відрізняє його від інших об’єктів і існує протягом усього життєвого циклу об’єкта. Ідентифікатор об'єкта повинен бути незалежним від значень його атрибутів

3. Інкапсуляція

Об'єктно-орієнтована модель представлення даних передбачає інкапсуляцію та приховування інформації. Це означає, що стан об'єкта можна приховати, щоб запобігти доступу до внутрішньої логіки роботи об'єкта.

4. Підтримка структур і класів

В ООП структура поєднує загальні характеристики набору сутностей. Наприклад, векторна структура поєднує координати X, I та Z.

Концепція класу схожа на структуру, але більше стосується розподілу пам’яті для об’єктів класу під час виконання програми.

5. Імітація та поліморфізм

Імітація є однією з основних концепцій об'єктно-орієнтованого програмування. Це пояснюється тим, що клас може мати дочірній клас, який має ті самі властивості, що й базовий клас, але також має нові властивості.

Метод базового класу можна замінити в класі-нащадку. Ця властивість називається поліморфізмом.

6. Повнота розрахунків

SQL не має всіх функцій звичайних мов програмування. Такі мови, як Pascal, C, C# або Java, можна назвати обчислювально повними, оскільки вони використовують усі обчислювальні можливості комп'ютера. SKL можна назвати реляційно повним, оскільки в ньому реалізована вся логіка реляційної алгебри. Так, будь-який код, написаний на SQL, можна переписати на C++, але не весь код на C++ можна переписати на SQL.

Тому більшість програм, які використовують реляційні бази даних, передбачають використання вбудованих запитів SQL у звичайній мові програмування.

Це означає, що мова обробки даних має бути мовою програмування загального призначення.

7. Паралелізм

8. Реставрація

9. Система спеціальних запитів

Реляційні бази даних використовують SQL як мову запитів.

Об'єктно-орієнтовані бази даних також повинні мати засоби надсилання запитів до бази даних. База даних повинна забезпечувати високорівневі, незалежні від програми засоби надсилання запитів. Це не обов'язково має бути мова запитів, наприклад SQL.

10. Можливість додавання нових типів даних

Набір типів даних має бути розширюваним. Користувач повинен мати засоби для створення нових типів даних на основі набору визначених системою

типів. Крім того, не повинно бути різниці між тим, як використовуються типи даних системи та користувача.

Об'єктно-орієнтована СУБД

По суті, OOSDB — це об’єктно-орієнтована база даних, яка надає можливості СУБД для об’єктів, створених за допомогою об’єктно-орієнтованої мови програмування.

Прикладні програми, що використовують об'єктно-орієнтовану базу даних, написані на об'єктно-орієнтованих мовах програмування, а можливості СУБД існують завдяки спеціальному API, в якому є клас бази даних або Persistent base class, що реалізує функції для роботи з даними.

OOSUBD дозволяє працювати з об'єктами бази даних так само, як і з об'єктами в об'єктно-орієнтованих мовах програмування. СУБД розширюють мови програмування, прозоро вводячи довгострокові дані, керування паралелізмом, відновлення даних, асоціативні запити та інші можливості. Деякі об'єктно-орієнтовані бази даних розроблені для тісної роботи з об'єктно-орієнтованими мовами програмування, такими як Python, Java, C#, Visual Basic .NET, C++, Objective-C і Smalltalk; інші мають власні мови програмування. СУБД використовують точно таку ж модель, що й об’єктно-орієнтовані мови програмування.

Переваги використання OODBMS:

1. Не існує проблеми неузгодженості між моделлю даних у програмі та базою даних (невідповідність імпедансу). Усі дані зберігаються в базі даних у тому ж вигляді, що й у програмній моделі.

2. Немає необхідності окремо підтримувати модель даних на стороні СУБД.

3. Усі об’єкти на рівні джерела даних суворо типізовані. Більше немає назв стовпців у масиві!

Переваги використання об'єктно-орієнтованої бази даних перед реляційними базами даних:

1. Об’єктно-орієнтовані бази даних дозволяють представляти складні об’єкти більш прямим способом, ніж реляційні системи.

2. Визначення власних абстракцій. Об'єктно-орієнтовані бази даних надають можливість визначати нові абстракції та керувати реалізацією таких абстракцій.

Сучасні пакети OODB дають користувачеві можливість створювати новий клас з атрибутами та методами, мати класи, успадковувати атрибути та методи від суперкласів, створювати екземпляри класу, кожен з унікальним ідентифікатором об’єкта, витягувати ці екземпляри по одному або в групах. , а також завантажувати та виконувати методи, визначати об’єкти як колекції інших об’єктів, визначати властивості, які також можуть мати складну структуру та визначатися за допомогою конструктора колекцій.

3. Спрощена конструкція деяких з'єднань. В об’єктно-орієнтованих базах даних підтримується інструмент зворотного відношення для вираження взаємних посилань між двома об’єктами (бінарне відношення). Така система забезпечує посилальну цілісність, встановлюючи відповідне посилання зворотного зв'язку відразу після створення прямого посилання.

4. Немає необхідності у визначених користувачем ключах. Модель OOBD має концепцію ідентифікаторів об'єктів, вони автоматично генеруються системою і гарантовано унікальні для кожного об'єкта. Ця обставина, поряд з тим, що модель OOBD усуває необхідність у визначених користувачем ключах, надає об'єктно-орієнтованим базам даних інші переваги. По-перше, програма не може змінити ідентифікатор об'єкта. По-друге, концепція ідентифікованого об’єкта передбачає чітку та послідовну концепцію ідентичності, незалежно від того, як здійснюється доступ до об’єкта або як об’єкт моделюється за допомогою описових даних.

5. Наявність присудка порівняння. У RDB порівняння завжди базується лише на значеннях. У цій моделі два записи є однією сутністю, якщо всі їхні

ключові атрибути мають однакові значення. Однак інші типи порівнянь були розроблені та визначені в моделі OOBD.

6. Менше потреби в зв'язках. Реляційні об’єднання — це механізм, який відображає зв’язки між таблицями на основі значень відповідних пар атрибутів у цих зв’язках. В ООБД потреба в таких поєднаннях значно менша. Але оскільки два класи можуть мати відповідні пари атрибутів в OOBD, можливість реляційного зв’язку все ще може бути збережена в цій моделі.

7. Виграш у продуктивності. У більшості OOBD, коли об’єкт завантажується в пам’ять, ідентифікатори, що зберігаються в цьому об’єкті, перетворюються на покажчики в пам’яті.

8. Алгебра об'єктів. Об’єктно-орієнтована алгебра не така деталізована та розвинена, як реляційна алгебра. Але як би там не було, така алгебра існує.

Недоліки використання об'єктно-орієнтованих баз даних порівняно з реляційними:

1. Недостатні можливості оптимізації запитів. Однією з найбільш значущих проблем в OOBD є оптимізація декларативних запитів. Оптимізація запитів щодо OOBD ускладняється додатковою складністю самої об’єктно-орієнтованої моделі даних.

2. Відсутність стандартної алгебри запитів. Ця обставина також ускладнює оптимізацію вимог.

3. Питання безпеки. RDB підтримує авторизацію, тоді як більшість OOBD – ні. RDB дозволяють користувачам передавати та скасовувати права на читання або зміну. ООБД зможуть отримати більш широке поширення у сфері бізнесу лише за умови вдосконалення цієї функції в них.

4. Обмежені можливості налаштування продуктивності. Більшість OODB мають лише обмежені можливості налаштування продуктивності. У RBD інсталяторам надається можливість налаштовувати продуктивність системи, встановлюючи велику кількість параметрів, встановлених системним адміністратором.

5. Недостатня підтримка складних об'єктів. Повна функціональність складних об'єктів ще не підтримується. За допомогою цих посилань можна переміщатися по посиланнях і кодувати операції, але немає визначених загальних операцій, які б використовували різні типи семантики посилань.

6. Обмежена інтеграція з системами об'єктно-орієнтованого програмування. Конфлікти імен; необхідність перегляду класової ієрархії; Схильність OOBD перевантажувати системні операції.

Як бачите, OOBD має дуже цікаву концепцію. Маніпулювання даними у вигляді об'єктів дозволяє зберігати дані зі складною структурою і не витрачати зусилля на вирішення такої проблеми, як неузгодженість імпедансів. Це дає можливість отримати досить великий виграш у продуктивності інформаційної системи. Розглянуті приклади демонструють легкість виконання запитів порівняно з реляційними базами даних. Однак варто зауважити, що OOBD ще не ідеальні, і на ринку є серйозні конкуренти, куди вкладаються великі гроші.

Для розробки програмного забезпечення системи управління проектами були обрані наступні засоби та мови програмування:

HTML — це мова гіпертекстової розмітки. Документ HTML обробляється в браузері і відображається на екрані у звичному для людини вигляді.

HTML у магістерській роботі буде використано для створення фреймворків веб-сторінок, до яких будуть підключені різні типи баз даних і виконуватиме тестування швидкості системи залежно від вибраних баз даних.

2. CSS і CSS3 анімації. CSS - каскадні стилі найчастіше використовуються для візуального представлення сторінок, написаних на HTML. Анімація CSS3 дозволяє анімувати більшість елементів HTML без використання технологій JavaScript або Flash.

CSS у цій статті буде використано для створення приємного інтерфейсу користувача з поєднанням анімації CSS3.

3. PHP — це сценарна мова програмування, створена для створення HTML-сторінок на стороні веб-сервера. Веб-сервер інтерпретує його в код HTML, який

передається на сторону клієнта. На відміну від скриптової мови JavaScript, користувач не бачить PHP-код, тому що браузер отримує готовий HTML-код. Це перевага з точки зору безпеки.

У цьому документі PHP буде використано для створення HTML-сторінок з даними, які будуть отримані з бази даних (БД) за допомогою класу PDO php.

4. Бази даних – об’єктно-орієнтовані бази даних будуть перевірені на їх використання як сховища даних. І для порівняння з класичними реляційними базами даних.

Об'єктно-орієнтовані бази даних - Cache, db4object, Prest, Volante, VelocitiDb.

Реляційні бази даних - Microsoft SQL Server, PostgreSQL.

2.3 Розгорнута постановка завдання

Відповідно до технічного завдання магістерської роботи буде реалізовано програмне забезпечення, що відображає роботу ООБД.

У процесі розробки магістерської роботи необхідно виконати наступний обсяг робіт:

а) проаналізувати існуючі аналогові системи для виявлення їх позитивних і негативних характеристик. Результати аналізу будуть враховані при подальшій розробці;

б) вибрати та обґрунтувати спосіб побудови системи контролю за роботою технічного обладнання виробництва в автоматизованому режимі. Розробляти системні функціональні та структурні системи;

в) розробити системне програмне забезпечення, що забезпечує реалізацію поставленого технічним завданням завдання. Побудувати блок-схеми програмних алгоритмів і підпрограм;

г) організувати інтерфейс користувача для створення та відображення повідомлень про некоректні дії користувача та нетипові ситуації;

д) розробити рекомендації щодо організаційно-методичних заходів щодо забезпечення впровадження системи у промислове використання та її подальшої успішної експлуатації;

д) виконати розрахунки для визначення економічної ефективності розробленої системи;

ж) розробити заходи з охорони праці під час впровадження та експлуатації системи та розробити заходи з охорони праці;

з) зробити висновки про обсяги виконаної роботи та досягнуті результати.

3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

3.1 Опис функціонування системи

Виходячи з теми магістерської роботи, необхідно розробити системне програмне забезпечення підприємства. Компанія працює з клієнтами, тому дані клієнтів зберігаються в базі даних нашої інформаційної системи. Крім того, співробітники працюють з клієнтами, кожен співробітник в цій інформаційній системі може маніпулювати інформацією про клієнтів від свого імені. Тому в програмі створюється така ієрархія класів. Базовий клас Person зберігає загальну інформацію про людину: ім'я, список адрес електронної пошти. Його нащадками є класи Client і Employee. У першому зберігається деяка інформація про клієнта, у другому - дані для входу. На рисунку 1.1 показана архітектура класів цієї інформаційної системи.

системи.

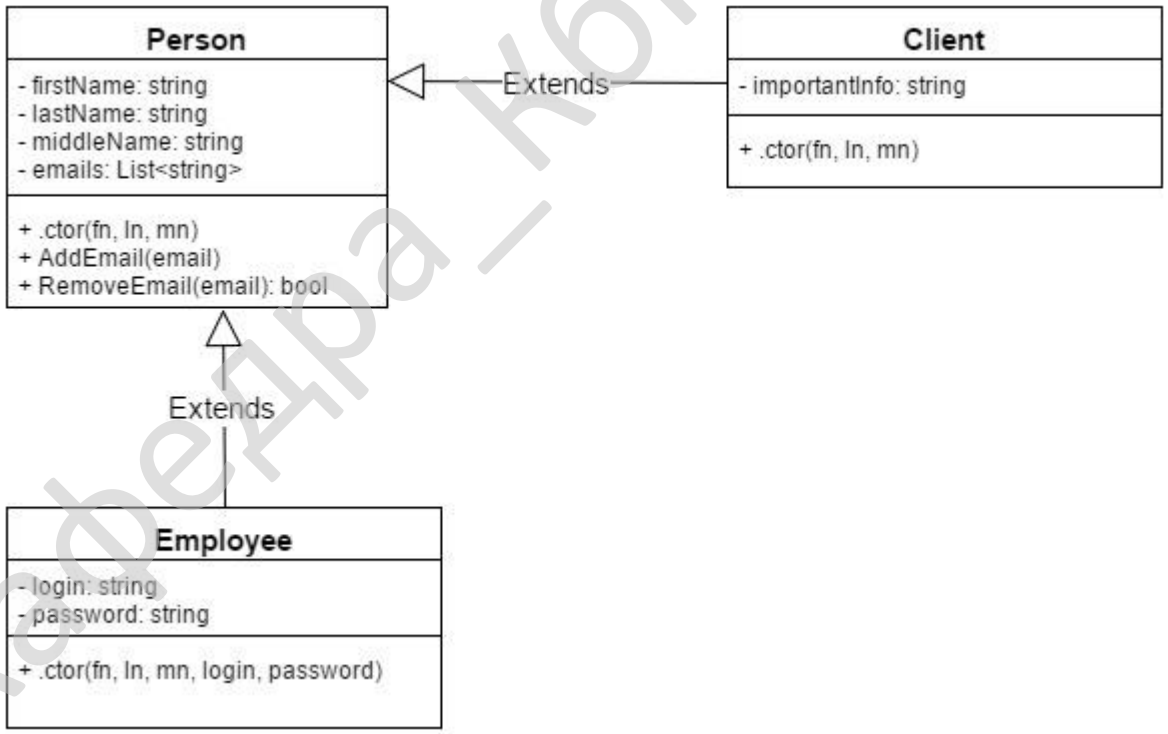

Рисунок 3.1 – UML-діаграма класів ІС

У випадку з використанням реляційних баз даних в нашій інформаційній системі схема бази даних виглядала б так, як на рисунку 3.2. Наслідування відповідає відносинам один-до-одного між таблицями. Також виникла необхідність додати ще одну таблицю Email з складовим основним ключем, першою частиною якого є зовнішній ключ на таблицю Person, а другий – сама електронна адреса.

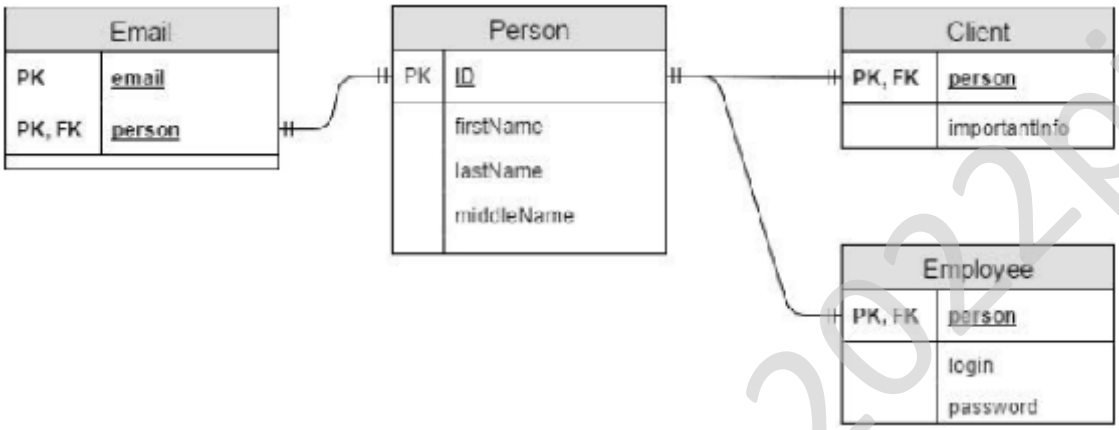

Рисунок 3.2 – Реляційне представлення даних інформаційної системи

Тепер давайте порівняємо, як надсилаються запити з боку користувача. Щоб отримати інформацію про клієнта, вам потрібно об’єднати таблиці Client і Person за допомогою ключового слова JOIN і використати конструкцію WHERE, щоб визначити умови, які мають бути виконані для клієнта, інформацію про якого ми шукаємо. У нашому випадку ми шукаємо клієнта за ID.

```
1. SELECT Person.firstName,
2.        Person.lastName,
3.        Person.middleName,
4.        Client.importantInfo
5.   FROM Person
6.        JOIN Client
7.          ON Client.person = Person.id
8.  WHERE Person.id = 25;
```

Рисунок 3.3 – Вивантаження даних з допомогою SQL-запиту

Додавання нових даних виконується за допомогою запиту INSERT. Щоб додати нового клієнта, потрібно спочатку додати новий запис до таблиці Person. Далі ми додаємо новий запис до таблиці Client, ідентифікатором якого є ідентифікатор щойно доданого запису в таблиці Person. Виконання цих запитів має бути безперервним, тому вони загорнуті в одну транзакцію. Недоліком

цього підходу до зберігання інформації є те, що дані в таблицях Person і Client не пов’язані між собою. Можна, наприклад, видалити запис із таблиці «Клієнт», але не видалити запис із цим ідентифікатором із таблиці «Особа».

```
1. BEGIN TRANSACTION;
2. INSERT INTO Person (firstName, lastName, middleName)
3.      VALUES ('Andrew', 'Tokarskiy', NULL);
4. INSERT INTO Client (person, importantInfo)
5.      VALUES ((SELECT MAX(Person.id) FROM Person), '!!Some info!!');
6. COMMIT;
```

Рисунок 3.4 – Додавання даних з допомогою SQL-запитів

Якщо вам потрібно видалити запис з таблиці Person, вам потрібно видалити або змінити всі записи з інших таблиць, пов’язаних з ідентифікатором цієї особи, на рисунку 3.5 показано операцію видалення користувача. Усі операції видалення мають бути загорнуті в транзакцію.

```
1. BEGIN TRANSACTION;
2. DECLARE @id INT;
3. SET @id = 25;
4. DELETE FROM Email WHERE person = @id;
5. DELETE FROM Client WHERE person = @id;
6. DELETE FROM Person WHERE id = @id;
7. COMMIT;
```

Рисунок 3.5 – Видалення даних з допомогою SQL-запитів

Як бачите, зберігати такі дані в реляційних базах даних досить важко для програміста. Розглянемо, як ті самі дані зберігаються в об'єктних базах даних. Операція вивантаження клієнта в OODB відрізняється від операції вивантаження клієнта в реляційних базах даних тим, що немає необхідності використовувати JOIN для підключення сутностей. Той факт, що Клієнт є нащадком Особи, вже відомий базі даних. На рисунку 3.6 показані дані вивантаження з OOBD.

```
1. Client GetClientById(Database db, int id)
2. {
3.     return db.AsQueryable<Client>()
4.              .First(person => person.Id == id);
5. }
```

Рисунок 3.6 – Вивантаження даних з допомогою ООБД

Це ж стосується і додавання нової інформації, як видно із рисунка 3.7. Також немає необхідності створювати в базі даних нову сутність для зберігання у ній електронних адресів. Вони вже зберігаються у полі emails, що має тип даних List.

```
1. var client = new Client
2. {
3.     FirstName = "John",
4.     LastName = "Smith",
5.     MiddleName = null,
6.     ImportantInfo = "!!Some info!!"
7. };
8. client.AddEmail("smith@email.com");
9.
   db.Persist(client);
```

Рисунок 3.7 – Додавання даних з допомогою ООБД

Видалити об'єкт з об'єктно-орієнтованої бази даних дуже просто. Досить лише вказати об'єкт, який потрібно видалити.

Як бачите, OOBD має дуже цікаву концепцію. Маніпулювання даними у вигляді об'єктів дозволяє зберігати дані зі складною структурою і не витрачати зусилля на вирішення такої проблеми, як неузгодженість імпедансів. Це дає можливість отримати досить великий виграш у продуктивності інформаційної системи.

Розглянуті приклади демонструють легкість виконання запитів порівняно з реляційними базами даних. Однак варто зауважити, що OOBD ще не ідеальні, і на ринку є серйозні конкуренти, куди вкладаються великі гроші.

3.2 Розробка структурної схеми

Схема структури надає інформацію про взаємодію майбутніх частин

програми, за яку функціональність вони будуть відповідати і як вони будуть спілкуватися між собою. Не можна сказати, що вони інформативні, тому що не можна простежити, як дані на них передаються і змінюються. Тому залежно від технічного завдання створюють структурні схеми окремих частин програми. Метод покрокової деталізації зазвичай використовується для розробки структурної діаграми, щоб визначити всі компоненти, які складають діаграму, а також набори підпрограм і пов’язаних бібліотек.

Структурними компонентами програми можна назвати підсистеми, бази даних, бібліотеки тощо. А за допомогою посилань можна продемонструвати, як вони спілкуються між собою, обмінюються інформацією, підключають нові бібліотеки. При розробці структурної схеми програмного забезпечення були розглянуті основні модулі програми та їх взаємодія. Дуже важливо під час розробки, особливо якщо проект великий і має складну архітектуру, розділити програму на структури, щоб показати логіку програми, показати, де підключені бібліотеки і за що вони відповідають. У заданій схемі взаємодія модулів відображається в ієрархічній схемі. Однак там, де розроблені модулі підключені до основного модуля, схема не відображає порядок роботи системи. Схема доповнюється розшифровкою функцій, які з'єднують модулі.

Структурна схема (рисунок. 3.8) показує зовнішні специфікації програми, які дають розуміння функціональної частини програми, за що вони відповідають, як взаємодіють із вхідними та вихідними даними. Особливо важливо розуміти тип структури даних.

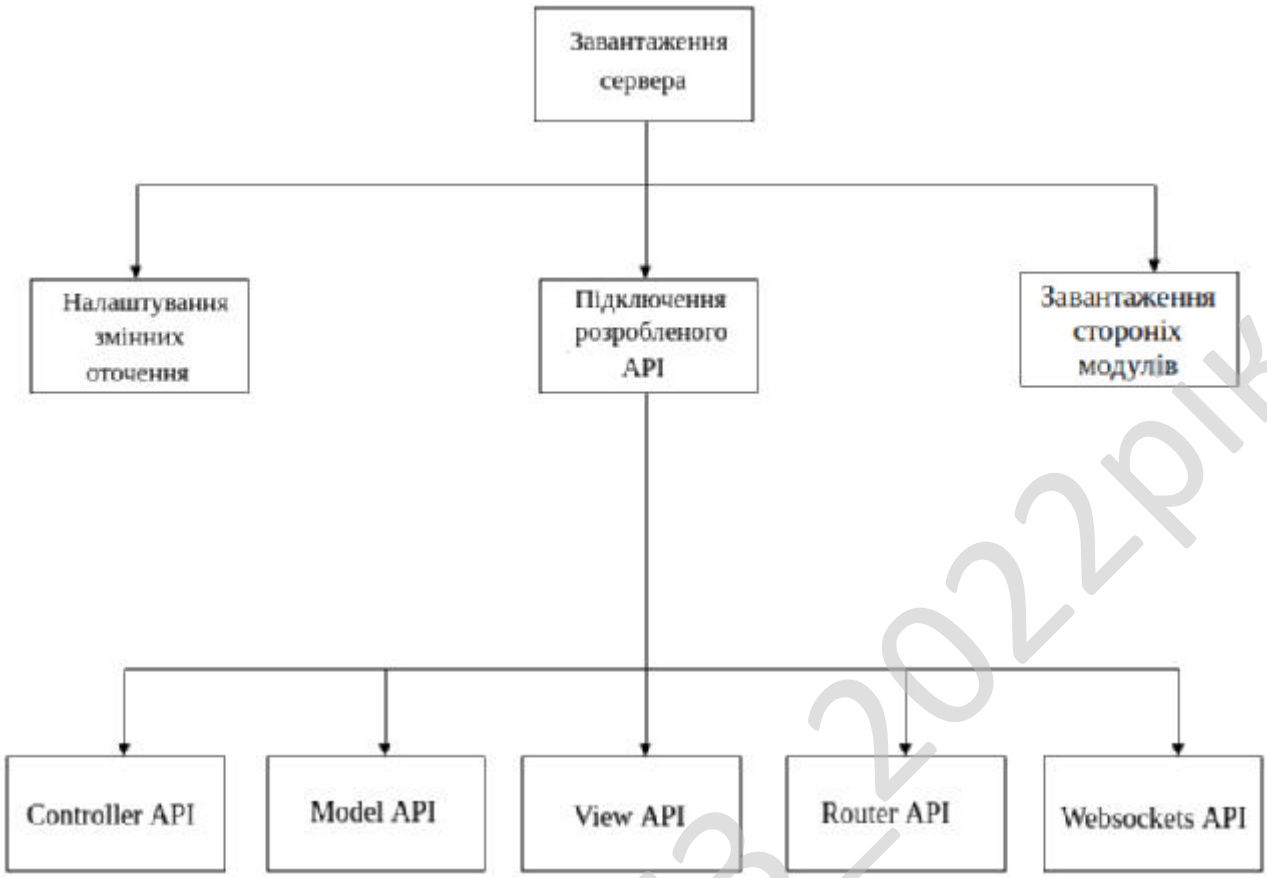

Рисунок 3.8 - Структурна схема

Поділ програми на окремі модулі здійснювався за принципом від загального до конкретного, при цьому окремі модулі виступали як сервіси, які виконували окремі завдання. Хоча існують інші варіанти створення ієрархій, циклічного перегляду кількох параметрів і підтримки реалізації шаблону MVC, цей дистрибутив забезпечує найзручніший спосіб налаштування частин, які буде легко підтримувати та тестувати. Структурна схема відображає головний принцип програми, її основні та другорядні блоки, їх призначення та взаємну взаємодію. Це допомагає зрозуміти, як працює програма, що полегшує подальше обслуговування.

3.3 Розробка функціональної схеми

Функціональна схема, або також звана схемою даних, відображає взаємодію компонентів програми, вказуючи склад даних, а також їх призначення. При створенні такої схеми використовуються позначення, прийняті стандартом.

Розроблена функціональна схема допомагає відстежувати виконання існуючих вимог і функцій, відстежувати, які компоненти і модулі використовуються, як вони взаємодіють один з одним, за допомогою вхідних і вихідних даних. Така схема відображає роль кожного функціонального елемента в додатку і з якими компонентами він взаємодіє. Для розробки даної схеми за основу була взята структурна схема.

Переглядаючи функціональну схему, ви можете прослідкувати принцип роботи всієї програми, як взаємодіють між собою клієнтська та серверна частини, а також які бібліотеки та технології використовуються для зберігання та передачі даних.

Цей тип діаграм зазвичай використовується для наочної демонстрації роботи алгоритму в спрощеній формі, щоб ви могли стежити за функціями та діями, які виконує алгоритм, які дані він використовує та який результат отримує.

Це допомагає оцінити складність алгоритму, зрозуміти принципи його роботи та контролювати можливі зміни та коригування. Подивіться, як функціональні схеми пов’язані одна з одною. Для опису компонентів та їх взаємодії використовувався національний стандарт. Там, де прямокутник показує окремі функціональні частини або як вони об’єднані у функціональні групи, причому кожна група має власну позначку та назву.

Є кілька можливостей представлення функціональної схеми, використовуючи принципову схему або змішану структуру, але в будь-якому випадку функціональна схема повинна відображати призначення пристрою.

На схемі (рисунок 3.9) показано:

- Умовна позначка в прямокутнику, якщо це функціональна частина.

- Умовне графічне позначення, що відображає положення і дію функціональної частини.

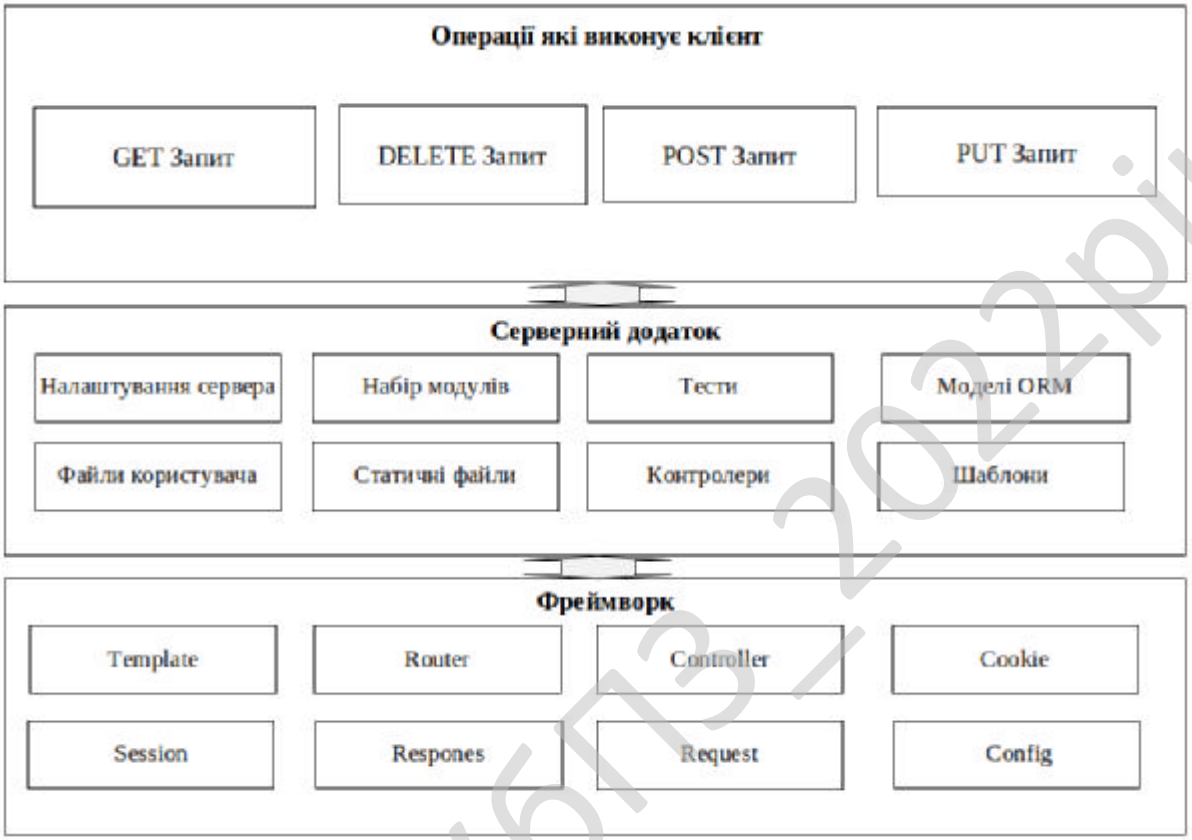

Рисунок 3.9 - Функціональна схема системи

У випадках, коли використовується принципова схема, функціональні частини разом з позиційними позначеннями компонентів мають бути схожими на схемі. У таких випадках перелік елементів не використовується, через використання даних принципової схеми. Коли функціональна схема розробляється без використання принципової, то використовуються загальні правила для відображення функціональних частин.

3.4 Розробка діаграми процесів

При моделюванні поведінки розробленої системи можна відслідковувати, як змінювалася робота програми, її стан, вхідні та вихідні дані, і як все це

вплинуло на розробку програми. Як працюють алгоритми в додатку і які результати вони дають в залежності від вхідних даних.

На діаграмі процесу за допомогою графічних об’єктів можна побачити в зручному для сприйняття форматі, які операції виконуються та як вони взаємопов’язані за допомогою стрілок.

На діаграмі (рисунок 3.3) можна побачити всі фази програми, від самого початку програми, коли користувач робить запит, як він потрапляє на сервер, потрапляє на контролер, обробляється, отримує дані з бази даних. і повертає його як відповідь користувачеві.

Об’єкти на схемі позначають процеси, які виконуються під час роботи програми, стрілки показують, як переходить управління від одного об’єкта до іншого. Існують також структурні діаграми, вони схожі на діаграми процесів, але вони відображають статичність даних, такий опис більше описує зв'язки об'єктів, тоді як діаграми процесів відображають динаміку програми.

Основною характеристикою діаграми процесу є динаміка, на відміну від інших, де переважає сценарій. У таких діаграмах сценарій не повинен конфліктувати з послідовністю виконання програми і бути безперервним. Проблеми виникають, коли об'єкт повинен знаходитися в кількох місцях одночасно. Це трапляється, коли послідовність дій у програмі супроводжується незрозумілою логікою.

Метою створення діаграми є розуміння послідовності дій процесу в програмі для отримання кінцевого результату. Спостерігаючи за тим, як працюють процеси, які дані вони отримують, як вони їх обробляють і передають користувачеві, ви можете передбачити, як має працювати програма.

Існують різні способи побудови діаграм, які допомагають слідкувати за логікою та послідовністю дій у процесах, але найкраще зарекомендував себе покроковий алгоритмічний метод побудови діаграм.

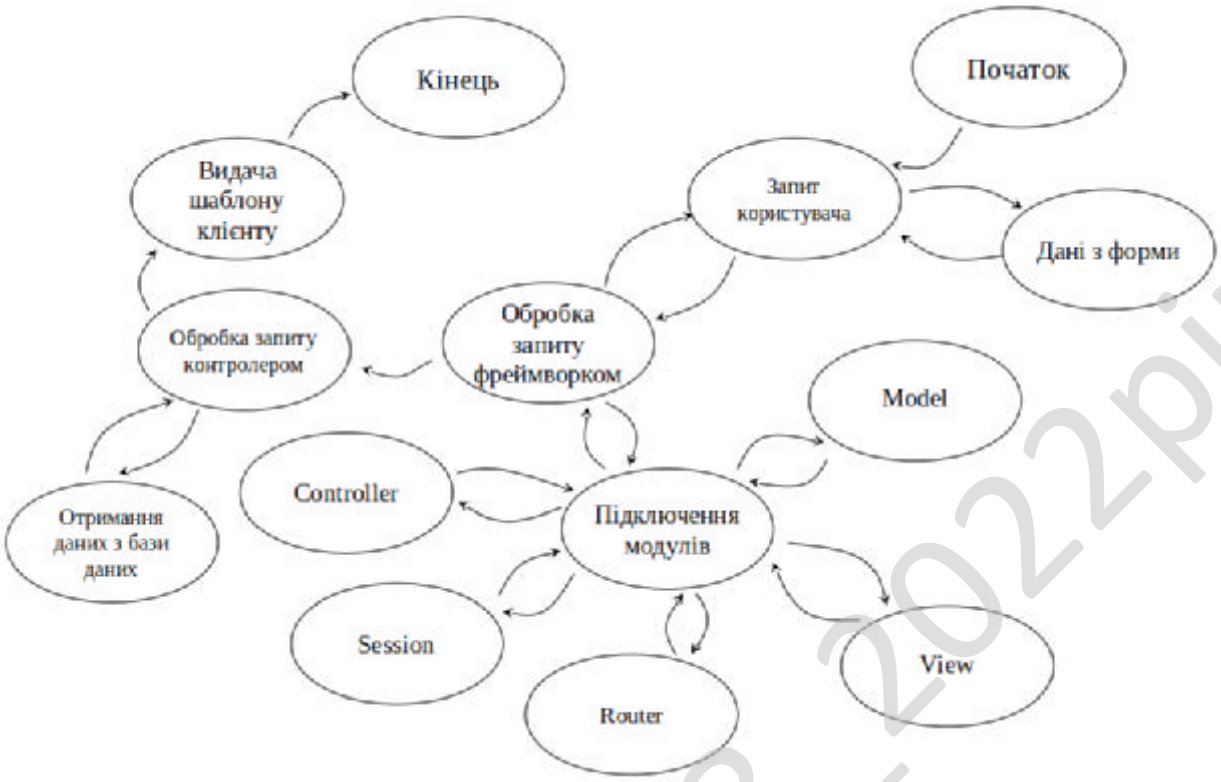

Рисунок 3.10 – Діаграма процесів системи

З усіх згаданих методів і технічних рішень у проектуванні та розробці програмного забезпечення, дивлячись на результати, можна зробити висновок, що я вибрав оптимальні дизайнерські рішення, які відповідають основним потребам, які були поставлені переді мною на початку проекту. Як видно з результатів, програма виконує всі поставлені завдання.

Розроблена діаграма дає представлення роботи додатку, як клієнтської так і серверної частини. Демонструє як взаємодіють між собою окремі частини додатку, та який результат отримує користувач при роботі з програмним забезпеченням.

4 РЕАЛІЗАЦІЯ РОБОТИ. РОЗРАХУНКИ ТА ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ВІРНІСТЬ ПРОЕКТНИХ І ПРОГРАМНИХ РІШЕНЬ

4.1 Розробка блок-схем та опис алгоритмів функціонування системи

Блок-схема надає можливість за допомогою графічних блоків та елементів, представити роботу додатку у зручній для сприйняття формі. За допомогою графічних блоків можна представити потоки, операції, дані, як вони обробляються та модифікуються. Дані які використовуються для вводу або виводу інформації прийнято позначати паралелограмом, дані які обробляються та модифікуються — прямокутником, графічний блок для прийняття рішення за допомогою заданої умови - ромбом, початок та кінець алгоритму — еліпсом. Всі ці графічні позначення використовуються згідно стандарту ДСТУ ISO 5807:2016. Використовуючи схему зручно простежити як виконується програма, з якими даними повинна працювати, та як вони послідовно змінюють свій стан. Це допомагає зрозуміти головне призначення програми, та її функціонал.

Але при створенні блок-схеми для додатка на AngularJS та NodeJS важливо брати до уваги його асинхронну модель, яка схоже працює як в браузері і NodeJS. Ця модель заснована на обробці подій в неблокуючому режимі та використанні обробників зворотних викликів. Це забезпечує виконання окремих функцій тільки у випадку виклику спеціальних сигналів, які виконуються викликом зворотних функцій або викликом слухачів подій, наприклад можна звернутися до бази даних, щоб отримати результат запиту, коли відповідь буде готова, тоді буде викликана функція зворотного виклику з отриманим результатом. Такі функції допомагають отримати результат, обробити його не витрачаючи зайві ресурси. Слухачі подій виконують таку саму функцію, як і зворотні функції, але є більш читабельні. Прикладом такої події може бути

натискання клавіші в браузері, на сервері це може бути HTTP запит. Для створення власних прослуховувачів подій, добре підходить EventEmitter — це клас, який можна використовувати при наслідуванні, та для обробки власних подій.

При використанні асинхронних подій, слід уважно стежити за послідовністю виконання операцій в алгоритмі, а також за даними і їх змінними в циклах або в операторах вибору, оскільки це може спричинити проблеми в роботі логіки програми. Під час виконання асинхронного коду, деякі частини коду можуть виконуватися незалежно від іншої частини програми, або виконуватися тільки до або після певної частини коду. Для цього існує концепція (flow control), яка слідкує за асинхронними операціями, розбиває їх на черги, об'єднує в групи, та слідкує за правилами оптимізації такого коду.

Асинхронні операції можуть виконуватися послідовно або паралельно, також важливо чи потрібні отриманні дані для передачі в наступні функції або чи важливий порядок виконання асинхронних функцій. Такі випадки можуть спричинити велику кількість залежного і важко зрозумілого коду, але на сьогодні існують різні технології для їх обробки. Наприклад технологія async-await, проміси. Нові ж версії Angular використовують бібліотеку RxJS, яка реалізує концепції реактивного програмування і використовуючи тип Observable спрощує використання асинхронного коду. Observables має велику функціональність і здатен оброблювати велику кількість потоків даних: відповіді сервера, примітивні типи, синхронний і асинхронний код.

Node це платформа, яка реалізує асинхронну подієво-керовану модель робот з даними, це схоже на роботу JavaScript в браузері. В обох цих середовищах використовується цикл подій, який не блокує виконання коду під час операцій вводу-виводу. У середовищі NodeJS варто пам'ятати про клас EventEmitter, який можна отримати зі стандартного модуля events, він використовується в багатьох стандартних модулях, наприклад для обробки HTTP запитів, або для роботи з потоками. У створених модулях, за допомогою нього

зручно обробляти подію «error», та додавати нові події за допомогою «newListener», та видаляти за допомогою «removeListener». А за допомогою on і emit можна створювати власні події, де функція on прив'язує функцію, а emit запускає подію. Цей клас активно використовувався при розробці чата на веб-сокетах.

На блок-схемі (рисунок 4.1) можна простежити послідовність виконання дій розробленого клієнтського додатка, як в сукупності завантажується в браузері, як він приймає дії користувача, обробляє їх та надсилає на сервер.

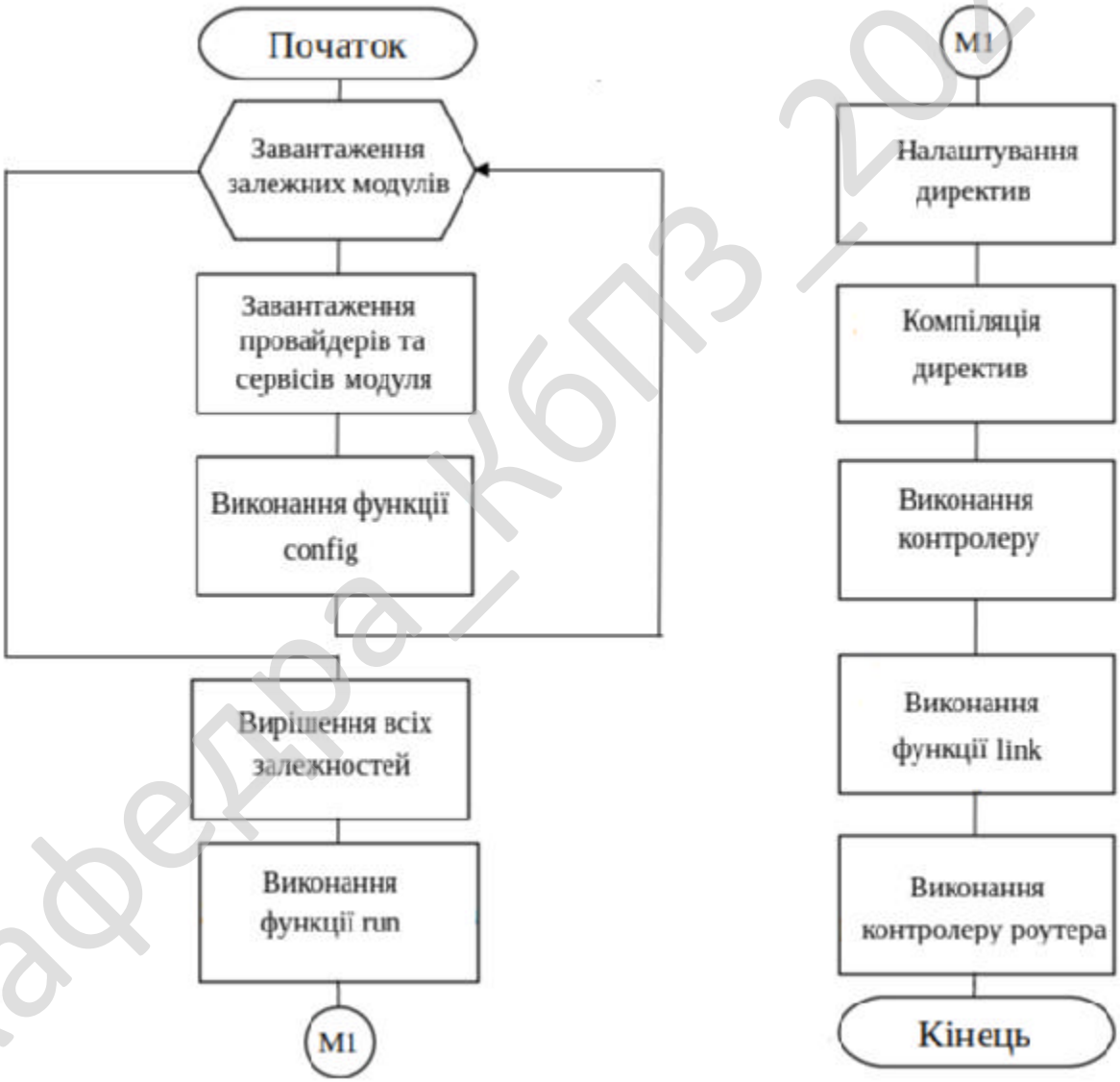

Рисунок 4.1 - Блок-схема головного модулю

Оскільки код є досить великим і має велику кількість не пов'язаних між собою сервісів та запитів, він поділений на окремі компоненти, в різних папках. Це допоможе зробити код більш зрозумілим, та зручним для сприйняття логіки програми. У більшості випадків модуль представлений у вигляді одного контролера, в якому представлена вся бізнес-логіка модуля. Крім контролера, також є сервіс, в якому знаходяться допоміжні функції, та репозиторій який містить функції, які роблять запити до бази даних.

На серверній стороні головним модулем є файл main.ts, який ініціює весь проект, завантажує документацію, валідацію та починає прослуховувати запити. Файл app.module.ts імпортує інші розроблені модулі, та виконує всі необхідні налаштування, на серверній частині додатку. NestJS вимагає від користувача дотримуватися архітектурного стилю MVC, при розробці логіки програми. Читаючи файл конфігурації користувача, він налаштовує сервер відповідно, до вказаних вимог, або обирає налаштування за замовчуванням з файла .env. Використовуючи вище згадані парадигми, до головного модулю, підключаються не тільки розроблені сервіси, а й база даних MongoDB та Redis, які відповідають за дані користувача та за горизонтальне масштабування. Такі вимоги фреймворку повинні допомогти користувачу програмного забезпечення на стороні сервера побудувати зручну для використання та підтримки архітектуру програми.

У головному файлі програми, який є основою всього програмного забезпечення, знаходиться декоратор @Module(), де відбувається підключення всіх налаштувань та створених компонентів, які допоможуть побудувати архітектуру MVC. В методі bootstrap зчитається файл налаштування сервера, імпортуються та підключаються необхідні модулі, та завантажується сервер на обраному домені та порту.

Однією з перших дій, які виконує програма це читання файлу користувача .env, де повинні бути вказані основні властивості налаштування веб-серверу, якщо файл відсутній, або більшість інформації відсутня програма бере необхідні

дані з файлу constants.js, де вказані всі властивості за замовчуванням. Для читання файлу і отримання необхідної інформації, використовується бібліотека @nestjs/config, яка використовує модуль dotenv. Після встановлення модулю потрібно в файлі app.module.ts імпортувати ConfigModule в корінь AppModule, після чого можна використовувати ConfigServise для отримання глобальних налаштувань в інших модулях. В цьому файлі відбувається підключення бібліотеки mongoose для роботи з базами даних, обробка запитів користувача контролером, перегляд встановленої маршрутизації.

Наступними діями є створення необхідних компонентів, які будуть містити набір API для взаємодії з додатком, за логікою MVC. В першу чергу це створення компоненту авторизації, якій містить контролер, сервіс, репозиторій та модуль. За допомогою декоратора @Controller(), який імпортується з стандартного NestJS фреймворку @nestjs/common. В контролері відбувається вся бізнес логіка запиту, де користувач може зареєструватися, перевірити свої дані, та отримати JWT токен для подальших запитів. У цілях безпеки перед збереженням паролю користувача він спершу хешується за допомогою бібліотеки bcrypt та функцій hash, а за допомогою функції compare перевіряє валідність паролю. За допомогою JwtService з бібліотеки @nestjs/jwt виписуються access та refresh токен, які використовуються для перевірки юзера. При отриманні запиту AuthGuard з бібліотеки @nestjs/passport перевіряє валідність токена. А за допомогою бібліотеки class-validator перевіряються вхідні дані запиту при реєстрації та створюється документація у сваггері. Бібліотеки які використовувалися при розробці серверної частини додатку:

- @nestjs/common — надає набір декораторів для створення контролерів, модулів, провайдерів, документації, для отримання тіла запиту, валідації, обгортки навколо Request та Response та статуси для повернення відповіді.

- @nestjs/config — використовує бібліотеку dotenv для отримання доступу до налаштувань.

- @nestjs/core — за допомогою цього модулю можна обрати на основі fastify чи express буде побудований додаток.

- @nestjs/jwt — модуль для генерації та перевірки токенів авторизації.

- @nestjs/mongoose — модуль для підключення бази даних MongoDB.

- @nestjs/passport — це проміжне програмне забезпечення, для перевірки аутентифікації.

- @nestjs/swagger — модуль надає декоратори для створення документації за допомогою сваггера.

- @nestjs-modules/mailer — використовується для надсилання e-mail повідомлень.

- bcrypt — бібліотека для шифрування даних.

- class-validator — надає набір декораторів для перевірки валідації.

- express — фреймворк для написання додатків на NodeJS.

- handlebars — інструмент для створення шаблонів.

- ioredis - клієнт для роботи з Redis.

- supertest — бібліотека для тестування запитів.

При кожному HTTP запиті, на сервері, викликається відповідний контролер, в якості аргументу може приймати декоратори Request і Response, але краще використовувати декоратори для отримання тіла запиту чи параметра. Щоб вернути відповідь досить просто вернути результат з функції. Така структура дозволяє працювати з асинхронною логікою, але якщо не буде відповіді запит залишиться відкритим. Оскільки код є досить великим і має велику кількість не пов'язаних між собою сервісів та запитів, він поділений на окремі компоненти, в різних папках. Це допоможе зробити код більш зрозумілим, та зручним для сприйняття логіки програми. У більшості випадків модуль представлений у вигляді одного контролера, в якому представлена вся бізнес-логіка модуля. Крім контролера, також є сервіс, в якому знаходяться допоміжні функції, та депозитарій який містить функції, які роблять запити до бази даних.

На серверній стороні головним модулем є файл main.ts, який ініціює весь проект, завантажує документацію, валідацію та починає прослуховувати запити. Файл app.module.ts імпортує інші розроблені модулі, та виконує всі необхідні налаштування, на серверній частині додатку. NestJS вимагає від користувача дотримуватися архітектурного стилю MVC, при розробці логіки програми. Читаючи файл конфігурації користувача, він налаштовує сервер відповідно, до вказаних вимог, або обирає налаштування за замовчуванням з файла .env. Використовуючи вище згадані парадигми, до головного модулю, підключаються не тільки розроблені сервіси, а й база даних MongoDB та Redis, які відповідають за дані користувача та за горизонтальне масштабування. Такі вимоги фреймворку повинні допомогти користувачу програмного забезпечення на стороні сервера побудувати зручну для використання та підтримки архітектуру програми.

У головному файлі програми, який є основою всього програмного забезпечення, знаходиться декоратор @Module(), де відбувається підключення всіх налаштувань та створених компонентів, які допоможуть побудувати архітектуру MVC. В методі bootstrap зчитається файл налаштування сервера, імпортуються та підключаються необхідні модулі, та завантажується сервер на обраному домені та порту.

Однією з перших дій, які виконує програма це читання файлу користувача .env, де повинні бути вказані основні властивості налаштування веб-серверу, якщо файл відсутній, або більшість інформації відсутня програма бере необхідні дані з файлу constants.js, де вказані всі властивості за замовчуванням. Для читання файлу і отримання необхідної інформації, використовується бібліотека @nestjs/config, яка використовує модуль dotenv. Після встановлення модулю потрібно в файлі app.module.ts імпортувати ConfigModule в корінь AppModule, після чого можна використовувати ConfigServise для отримання глобальних налаштувань в інших модулях. В цьому файлі відбувається підключення бібліотеки mongoose для роботи з базами даних, обробка запитів користувача контролером, перегляд встановленої маршрутизації.

Наступними діями є створення необхідних компонентів, які будуть містити набір API для взаємодії з додатком, за логікою MVC. В першу чергу це створення компоненту авторизації, якій містить контролер, сервіс, репозиторій та модуль. За допомогою декоратора @Contr

Окрім стандартних та сторонніх бібліотек було розроблено декілька окремих сервісів, які оголошують окремі класи і надають додаткові можливості розробнику серверного додатку. Кожен файл містить свій об'єкт з набором властивостей та методів, який відповідає за окремий компонент програмного забезпечення, його функціонал та логіку. Модулі відповідають за різні компоненти програми, які можна використати, щоб зробити логіку програми більш незалежною.

Для структурування серверної частини були взяті принципи архітектурного типового рішення MVC. Розглянемо основні компонентні для налаштування серверної частини, які були розроблені:

- Маршрутизатор (Route) - відповідно до маршруту обирає який контролер буде обробляти запит користувача.

- Контролер (Controller) - відповідає за обробку запитів. Містить методи для реалізації функціональних вимог до веб додатку. Використовує модель для отримання даних предметної області та передачі їх до шаблонів.

- Модель (Model) - відповідає за бізнес-логіку безпосередньо пов'язану з предметною областю. Є моделлю суті. Надає інтерфейс для роботи з сутностями. Інкапсулює обробку даних відповідної їй суті. Mongoose - представляє "обгортку" для моделей, надає функціональність для роботи з моделями.

- Шаблон (Template) - представлення інтерфейсу користувача у вигляді HTML розмітки.

- Вид (View) - представлення, код який необхідно обробити сервером додатка.

- API - стандартизований інтерфейс для роботи з даними предметної області. Інкапсуляція бізнес-логіки за певним маршрутом. Повертає всі дані в

форматі JSON. Запит до API визначається параметром "api" в рядку URL. Відповідно до REST, на прикладі роботи, семантика запитів була побудована в такий спосіб.

На стороні клієнта AngularJS активно використовує атрибути та директиви для побудови додатку. Наприклад головний атрибут ng-app знаходиться в кореневому тегі html і використовується для позначення елемента HTML, який AngularJS буде розцінювати як головний (корневий) елемент програми. За допомогою такого тегу можна повідомити AngularJS де він буде використовуватися, та де знаходиться початок програми, та де йому потрібно завантажувати подвійне фігурне зв'язування

Також в додатках створених за допомогою AngularJS часто можна побачити подвійне фігурне зв'язування за допомогою подвійних фігурних скобок. Так в AngularJS працює шаблонування і так він вставляє необхідні дані у DOM, а також оновлює ці дані при їх модифікації.

Завантаження програми за допомогою директиви дуже зручне, але при використанні великого об'єму коду і винесенні його в скрипт, варто використовувати ручний спосіб завантаження програми. При цьому відбувається:

- Створюється новий інжектор, який вводить нові залежності

- За допомогою інжектора створюється контекст у вигляді кореневої області

- AngularJS компілює DOM, починаючи з кореневого елемента, обробляючи всі директиви та прив'язки на своєму шляху

Використовуючи модель MVC, після запуску, додаток очікує нові події від користувача, наприклад натискання клавіши, щоб оновити модель та відобразити нові зміни користувачу. Дані моделі передаються всередину UserListController контролера. Контролер є просто функцією конструктора, який приймає параметр $scope.

Після реєстрації в головному модулі userApp модуля UserListControlleri, він буде визначений під час завантаження програми. Оскільки контролер

виступає у ролі контекста, він буде встановлювати зв’язок між моделлю та поданням. Це відбувається за допомогою директиви HgController, яка розташована в body і через імя контролера підключає дані на $scope. Ця область завантажується при створенні програми і доступна для всіх прив’язок, розташованих у тезі body.

В AngularJS область сприймається як основа, яка звязує шаблони, модель та контролер. Це допомагає тримати модель та представлення даних окремо, але синхронізованими між собою. Тому будь-які оновлення в моделі, будуть зразу відображені користувачу.

Якщо контролер керує логікою і контентом для даних шаблону, то шаблон виступає в якості форми, яка буде представлена користувачу. Через таку поширену методику, AngularJS пропонує об’єднувати їх в компоненти, а також створить область ізоляції для кожного екземпляра компонента, щоб вони були незалежними між собою. Для створення компонента використовується метод .component(), який приймає ім’я компонента та його налаштування (об'єкт визначення).

З шаблону до контролера, можна звернутися через псевдонім. Також можна використати директиву $ctrl для псевдоніму контролера, або перекрити доступ, якщо буде потрібно.

Завдяки модульній архітектурі в AngularJS, код можна використовувати повторно, особливо корисно коли функції декларують свій власний модуль, а всі пов’язані суб'єкти повинні бути в об’явленому модулі. Це дасть змогу переносити код не тільки в проекті, а й між додатками.

Наприклад всі компоненти містять шаблони, які будуть виводитися користувачу у вигляді HTML. Використовуючи CDO (Component Definition Object) можна вказати шаблон для компонента у вигляді рядка, це не є правильним підходом, особливо у великих шаблонах, тому зазвичай HTML-код виносять в окремий файл, що робить код в компонентах чистішим.

Хоча це і корисна практика використовувати CDO, але краще використовувати зовнішні шаблони в компонентах. У цьому допоможе властивість templateUrl в якій вказується URL, щоб визначити зовнішній шаблон.

Директива ngModel допомагає робити пошук в списку за заданими критеріями. Це можливо за допомогою технології зв'язування даних. При завантажені сторінки, поля прив'язуються до моделі даних, вказаної за допомогою з ngModeli та синхронізує ці дані. Наприклад, коли користувач введе у поле пошуку нові дані, вони зразу від сортируються у списку, через те що зміни в моделі даних зразу призводить до оновлення DOM, і відображення поточної інформації.

Ще хорошим прикладом двосторонньої прив'язки даних є випадаючи меню, де можна відсортувати список на найновіші. Це відбувається у контролері за допомогою властивості orderPropв 'date', при завантаженні додатку. Тобто спочатку оновлюється модель після чого зміни відображаються на шаблоні. Якщо ж вибрати в випадаючому меню "алфавіт" то прив'язка спрацьовує в зворотному напрямку, спочатку оновляться дані інтерфейсу, а після дані моделі.

AngularJS також має набір вбудованих служб, наприклад $http використовується для надсилання HTTP запитів. Служби керуються за допомогою DI підсистемою в AngularJS. Введення залежностей допомагає задати структуру додатку та послабити звязки між сутностями. В результаті додаток набагато простіше тестувати.

За допомогою метода $http робиться HTTP запит на веб-сервер, для отримання даних з бази, та перетворюючи їх в формат JSON. Служба $http повертає дані асинхронно та присвоює ці дані контролеру, після чого лані з JSON можна отримати в функціях зворотнього виклику. Цей метод приймає декілька параметрів, статус запиту і адресу запиту, після чого за допомогою функції then можна перевірити відповідь на помилку та отримати дані. Крі методу get є і інші методи: post, delete, put, head, jsonp за допомогою яких можна реалізувати повноцінний REST API.

Інжектор залежносте в AngularJS слідкує за залежностями, та надає ряд послуг контролеру, які можуть мати служби, в першу чергу він звертає увагу на назви аргументів. Якщо ж треба створити власні послуги, в цьому допоможе метод scope та методи, які мають префікс $ перед іменем. Префікс $ означає що це AngularJS-послуги.

Щоб реалізувати маршрутизацію, треба використовувати модуль ngRoute, який не є стандартним в AngularJS. Для його використання треба встановити додатковий модуль з npm: angular-route. Після чого можна оголосить нові маршрути за допомогою $routeProvider , який є постачальником послуги $route. Ця послуга має гарний функціонал, який дозволяє з'єднувати контролери, переглядати шаблони та поточну URL-адресу в браузері, або використовувати історію веб-переглядача та закладки.

Як можна побачити всі вищезгадані фреймворки: Angular, AngularJS, NestJS використовують патер проектування MVC. Його поширеність полягає в зручному розподілі логіки і даних. Спочатку користувач робить запит на сервер, де контролер запрошує дані додатку у моделі і передає їх дані шаблону, який виконує остаточне форматування перед тим як надати дані користувачу. Шаблон часто реалізують за допомогою шаблонізатора, на серверній стороні це виконує модуль handlebars, він приймає дані повернуті моделлю і вказує який шаблон для відображення треба використовувати.

Шаблони зазвичай містять фрагменти HTML, CSS і іноді логіку JavaScript коду, для впровадження динамічної поведінки (наприклад віджетів) або для зміни інтерфейсу. Але оскільки шаблони більше зв'язані з представленням даних, розробники серверної та клієнтської частини повинні рівномірно розподілити логіку.

При кожному запиті користувача, викликається відповідний контролер. Спочатку за допомогою бібліотеки class-validator перевіряються вхідні дані, якщо на контролері є інтерсептор він буде викликаний наступний, після чого буде перевірка гарди. Якщо під час виконання контролера була викликана помилка,

вона буде записана у файлі з логами з використанням модуля winston. Дані повернені з функції будуть надіслані клієнту.

Блок-схема демонструє роботу програми сервера та клієнта. На схемі можна побачити послідовність дій, які виконують клієнт щоб зробити запит на сервер та отримати необхідні дані з бази даних. Для цього спершу клієнту потрібно пройти авторизацію, щоб підтвердити свої дані. Дивлячись на схему можна прослідкувати які операції виконуються, як дані взаємодіють, та як програма змінює свій стан.

Схема (рисунок 4.2) містить головні функції програми, та побудована так, щоб при її перегляді було зрозуміло функціонал та призначення системи. Вона є досить інформативна, щоб зрозуміти як побудована програма.

Розроблений веб-додаток складається з наступних компонентів:

- app — головний компонент, який ініціалізує проект.

- auth — компонент який містить логіку авторизації та автентифікації.

- public_chat — компонент який відповідає за чат, та активно використовує сокети.

- users - компонент для роботи з даними юзера.

- mail — компонент для відсилання e-mail повідомлень.

У розроблених прикладах за допомогою фреймворків AngularJS та NestJS було створено функціонал для реєстрації, авторизації, а також надавало інформацію про користувачів за допомогою операції CRUD (операцій читання, створення, оновлення та видалення), а за допомогою клієнтських фреймворків Angular та AngularJS ці дані надавались користувачу у зручній для сприйняття формі.

В додатку було реалізовано концепцію REST-API, існують також інші способи взаємодії клієнта та сервера, але у даному випадку REST підійшов найкраще.

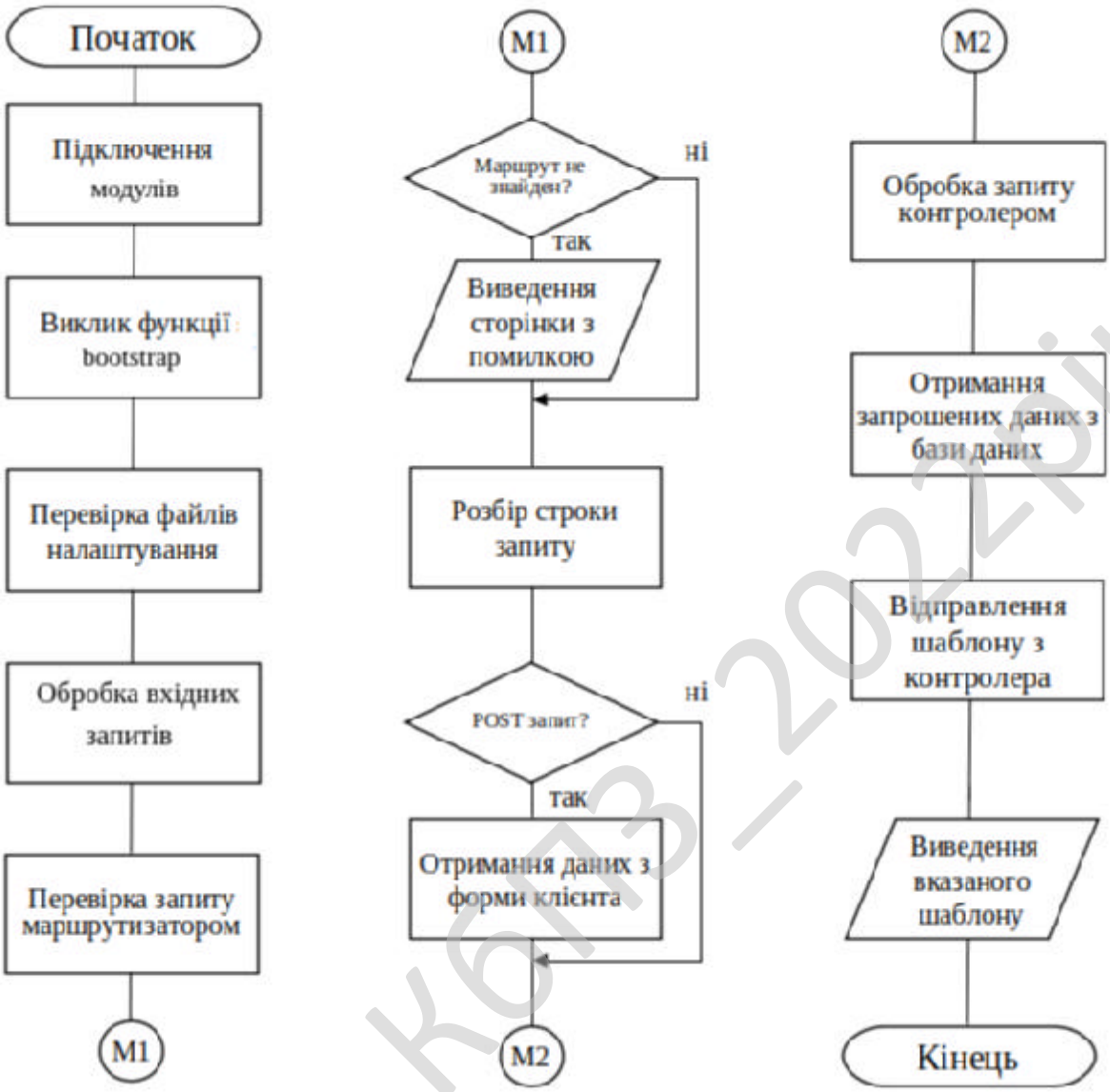

Рисунок 4.2 - Блок-схеми роботи системи

Всі запити виконувалися за допомогою HTTP-методів: GET, POST, PUT і DELETE, які використовувалися для створення, оновлення, читання та видалення даних з сервера. Кожний метод відповідав за власну операцію:

- POST — створював та додавав нові дані в базу.
- GET — отримував список заданих полів з сервера.
- DELETE — видаляв дані з бази.
- PUT — оновлював дані в базі.

Angular новішої версії має трохи іншу структура. Головним файлом в нього є main.ts, а в папці src знаходяться всі компоненти, стилі та модулі. Angular надає ряд декораторів: @Component, @NgModule, за допомогою яких створюються контролери, сервіси та модулі. Вони мають зручний інтерфейс, але будуть складними для розробників які бачать їх вперше. Прив'язка даних в Angular має декілька форм. Наприклад її можна реалізувати, за допомогою фігурних дужок або прив'язки DOM елемента до компоненту. Для двосторонньої прив'язки використовуються [(ngModel)], цей вираз допомагає прив'язати DOM елемент до значення, яке використовується на компоненті.

Також варто сказати про технології, які використовувалися під час розробки додатку:

Postman — інструмент призначений для тестування розробленого API. Він надає можливість зберігати запити, створювати колекції, документацію, а також слідкувати за показниками запиту. За допомогою цього інструмента можна значно прискорити розробку та тестування додатків.

MongoDB Compass — представляє графічну оболонку для роботи з базою даних MongoDB. Він був розроблений MongoDB Inc і надає велику кількість функціоналу для адміністрування бази даних. За допомогою нього можна: додавати, переглядати, оновлювати дані в базі, додавати або видаляти індекси, виконувати агрегації. Також він показує швидкість запиту, його навантаження, та кількість займаємо памяті. Але треба бути обережним при його використанні щлб не пошкодити дані в базі.

Для заповнення бази, даними для тестування або налаштування бази в продакшині потрібно створювати міграції. Краще за всіх з цим справляється пакет migrate-mongo. Для нього потрібна мінімальна кількість коду, а це означає що можна витратити більше часу на впровадження необхідної логіки для міграцій. Також він надає набір команд для роботи з міграціями, наприклад migrate_status виводить список всіх сценаріїв а також їх статус. Команди migrate_down та migrate_up запускають файли міграції.

Аналіз і вибір бази даних

Порівняємо продуктивність реляційних та об'єктно-орієнтованих баз даних на практиці. Для цього ми розглянемо швидкість та об'єм пам'яті, що використовується при додаванні, зчитуванні та повторному зчитуванні великої кількості даних. Також критерієм для порівняння буде розмір бази даних, заповненої даними.

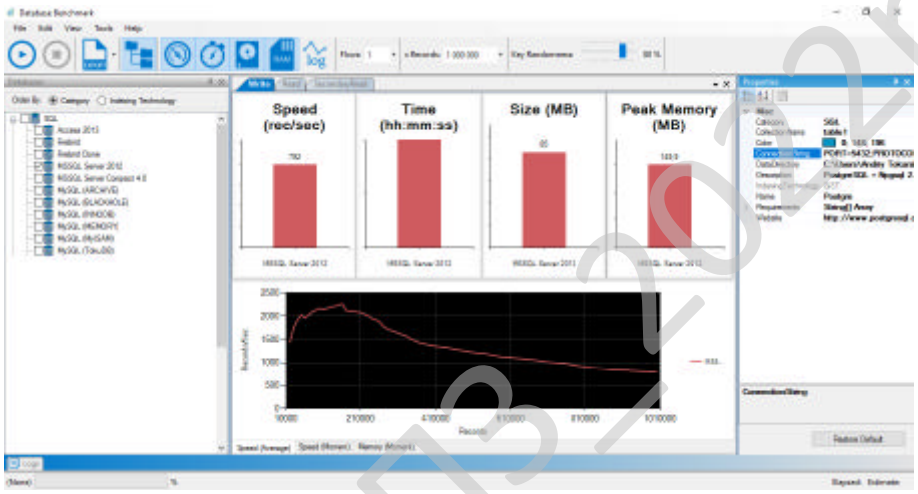

Рисунок 4.3 - Інтерфейс Database Benchmark

Для порівняння будемо використовувати програму Database Benchmark v3.0.0. Ця програма може вимірювати продуктивність дуже великої кількості баз даних, серед яких є як реляційні, так і об'єктні. Database Benchmark дозволяє спостерігати за процесом тестування у реальному часі. Розроблена із використанням .NET Framework + WPF.

Порівнювати будемо такі бази даних, як Microsoft SQL Server 2016, Db4Objects, VelocityDb, Volante, Perst. Всі ці бази даних можливо тестувати із використанням Database Benchmark.

Тестувати будемо швидкість і затрати пам'яті на додавання та читання одного мільйона записів, а також розмір відповідних баз даних.

Таблиця 4.1 - Швидкість операцій, виконуваних базами даних (у записах/сек)

	Додавання	Читання	Повторне читання
Db4Objects	9898	4839	4868
Perst	1894	1175	1194
VelocityDb	145603	30452	426333
Volante	15447	136968	119346
MS SQL Server 2016	792	412031	435002

Таблиця 4.2 - Максимальні затрати оперативної пам'яті (у Мб)

	Додавання	Читання	Повторне читання
Db4Objects	1134.1	1134.2	738.3
Perst	130	122.3	112.1
VelocityDb	256.2	272.5	272.6
Volante	2423.5	2505.7	2498
MS SQL Server 2016	149.9	130.5	130.5

Таблиця 4.3 - Розмір заповнених баз даних (у Мб)

Db4Objects	Perst	VelocityDb	Volante	MS SQL Server 2016
229.9	188.6	53.1	123	85

У таблиці 4.1-4.3 занесені результати вимірювання продуктивності баз даних.

Таблиця 4.1 показує швидкість виконання операцій, 4.2 показує затрати пам'яті на виконання цих операцій, а 4.3 – розміри баз даних.

Результати тестування реляційних баз даних

Для порівняння із об'єктно-орієнтованими базами даних мною було протестована реляційну базу даних Microsoft SQL Server. Database Benchmark

згенерував таблицю і заповнив її такими даними: число типу BIGINT, що виступає основним ключем таблиці, дві колонки типу VARCHAR(255), дві колонки типу INT, дві колонки типу REAL, і колонка типу DATETIME. Таблицю було заповнено одним мільйоном записів, що видно із рисунка 3.2.

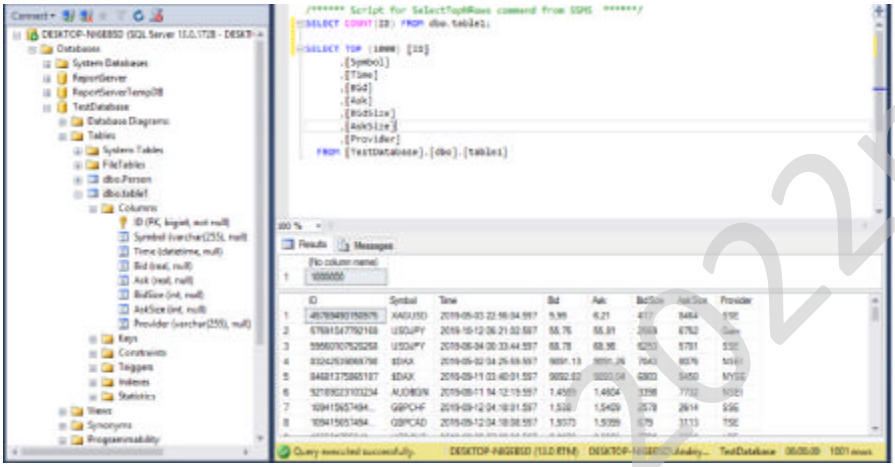

Рисунок 4.4 – Заповнення таблиці для тестування

Із результатів виконання програми Database Benchmark отримуємо, що Microsoft SQL Server є найповільнішою серед протестованих баз даних по швидкості додання нових записів.

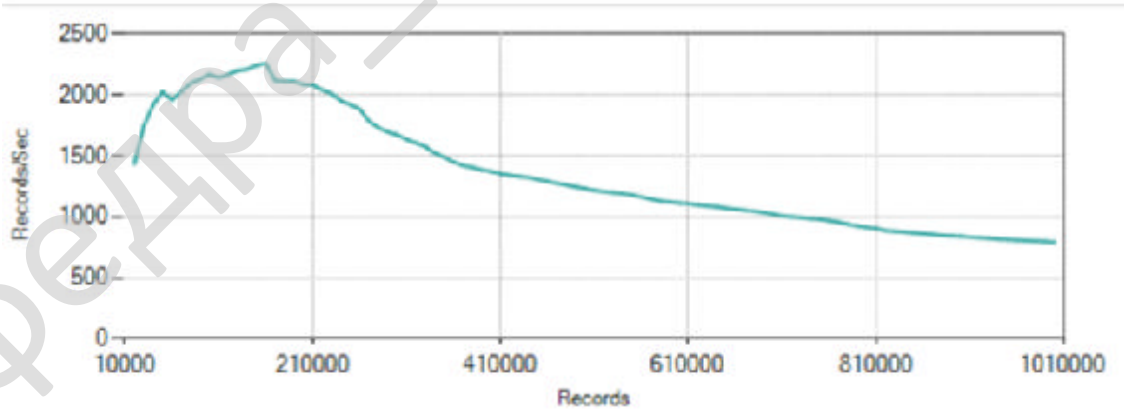

Рисунок 4.5 - Графік залежності середньої швидкості додання нових записів від кількості записів в SQL Server.

Рисунок 4.5 демонструє, що швидкість додання нових записів починає стрімко спадати після того, як кількість записів досягне приблизно 200000, але навіть в кращому випадку SQL Server не може сперечатись із більш швидкими у цьому плані базами даних VelocityDb, db4objects, Volante.

Проте розмір бази даних дорівнює всього 85Mbyte, що є другим результатом після VelocityDb.

Але жодна із протестованих об'єктних баз даних не перегнала SQL Server у плані швидкості читання даних. Крім того, SQL Server не вимагала великих затрат оперативної пам'яті. Менше оперативної пам'яті вимагає лише база даних Perst.

MS SQL Server є ідеальною для збереження даних, якщо в інформаційній системі виникає необхідність швидкого вилучення даних, а нові дані заносяться не дуже часто. Також важливою перевагою SQL Server над іншими базами даних є велика кількість гарно структурованої документації, чого не можна сказати про інші бази даних.

Результат тестування об'єктно-орієнтованих баз даних представлено на рисунку 4.6

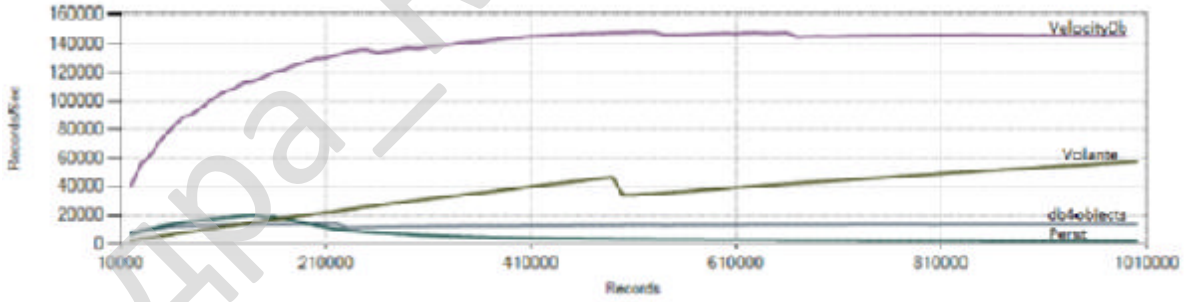

Рисунок 4.6 – Тестування ООБД

Серед об'єктних баз даних, по-перше слід виділити базу даних, що явно програла іншим у плані продуктивності. Цією базою даних є Perst. Хоча вона показує найкращі результати в плані затрат оперативної пам'яті, вона явно програє всім у плані швидкості вилучення інформації, і виграє лише SQL Server у плані швидкості додавання інформації.

Єдиною причиною використовувати Perst є необхідність мінімізації затрат оперативної пам’яті, але MS SQL Server програє Perst у цьому плані не дуже сильно. Крім цього, Perst не має багатьох очевидних сильних сторін, що є у SQL Server.

База даних VelocityDb виграє відразу у трьох характеристиках: розмір заповненої бази даних та швидкість додання нових записів. Крім того, швидкість повторного читання даних близька до значення цієї швидкості в MS SQL Server.

БД Volante виграє VelocityDb по значенню швидкості першого читання записів, але всі операції додавання, читання і повторного читання записів дуже дорогі. Додавання і читання одного мільйону записів із використанням Volante коштувало близько 2.5 Gbyte RAM.

БД Volante і VelocityDb являються in-memory database. Вони вирішують проблему дуже повільного доступу до пам’яті на жорсткому диску. In-memory databases спираються на збереження даних на оперативному запам’ятовуючому пристрої. Це робить доступ до даних дуже швидким.

Серед протестованих об’єктних баз даних, очевидно, найкращою можна назвати VelocityDb. Її можна використовувати при необхідності швидкої обробки великої кількості даних, а також для швидкого зчитування даних, особливо при необхідності їх повторного зчитування. У випадку необхідності використання безкоштовної об’єктної бази даних хорошим варіантом буде Db Volante.

Опис інформаційної системи

У магістерській роботі реалізована інформаційна система, що використовує у якості сховища даних об’єктно-орієнтовану базу даних VelocityDb. Вибір бази даних обґрунтований результатом порівняльного аналізу баз даних та метою магістерської роботи.

Інформаційна система зберігає у базі інформацію про працівників і клієнтів, а також про посібники та історію їх перегляду або скачування. Працівник може додавати нові книги, а також відмічати, що конкретний клієнт(студент) користувався(скачував) зазначеною літературою .

Також інформаційна система здатна повідомляти користувача про те, що у якихось його колег сьогодні день народження, а також є можливість перегляду книг, які переглядали колеги.

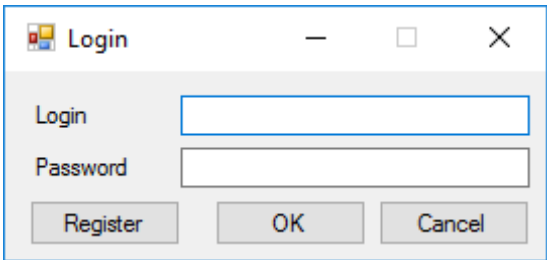

Рисунок 4.7 – Форма входу користувача

Для початку користування програмою користувачу потрібно буде авторизуватись (форма авторизації зображена на рисунку 4.7). Якщо у працівника існує обліковий запис, можна ввести логін і пароль, у іншому випадку – потрібно натиснути кнопку Register.

Для реєстрації користувача потрібно обов'язково вказати свої прізвище та ім'я, а також логін та пароль (рисунок 4.8).

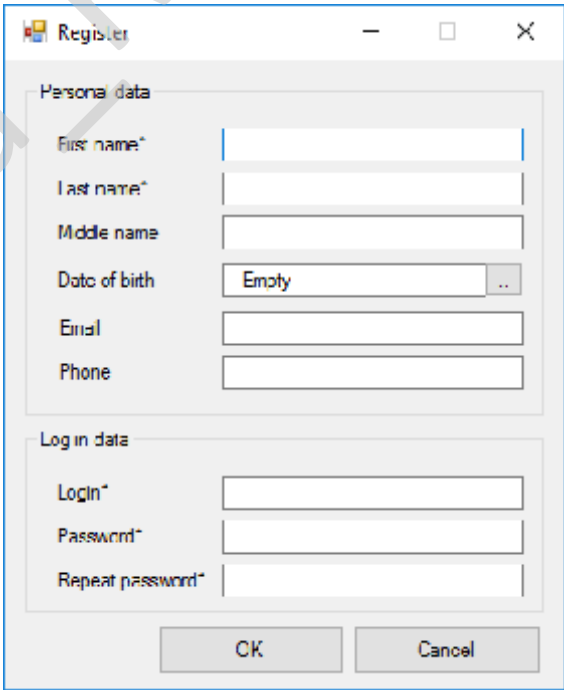

Рисунок 4.8 – форма реєстрації

Після авторизації програма перенаправляє користувача на вікно із головним меню програми, зображеним на рисунку 4.9.

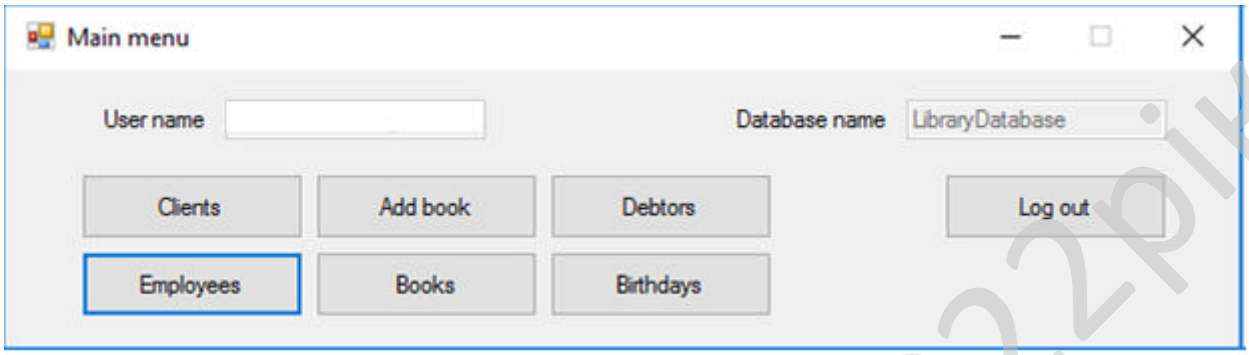

Рисунок 4.9 - Головне меню програми

Із меню можна перейти на вікна із клієнтами, працівниками. Також можна перейти на вікно додання книги, списку книг та списку людей, у яких сьогодні день народження.

Для перегляду списку книг потрібно натиснути на клавішу Books. Відкриється вікно, зображене на рисунку 4.10. Подвійний клік на рядок із книгою відкриє детальну інформацію про книгу. Тоді відкриється вікно, зображене на рисунку 4.10.

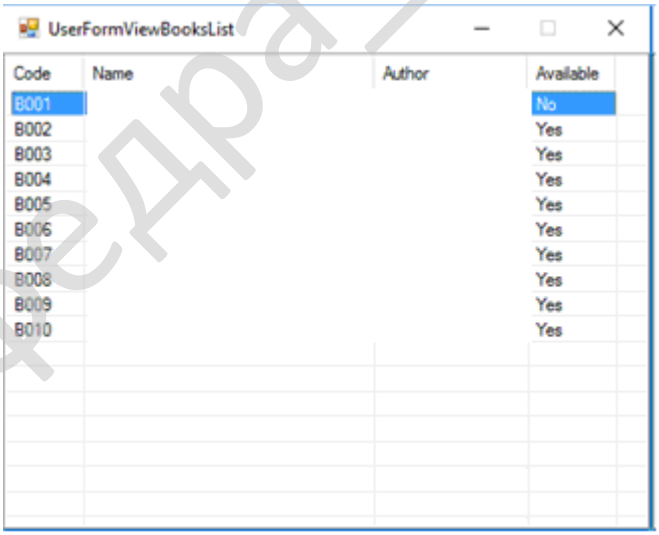

Рисунок 4.10 – Вікно із списком посібників

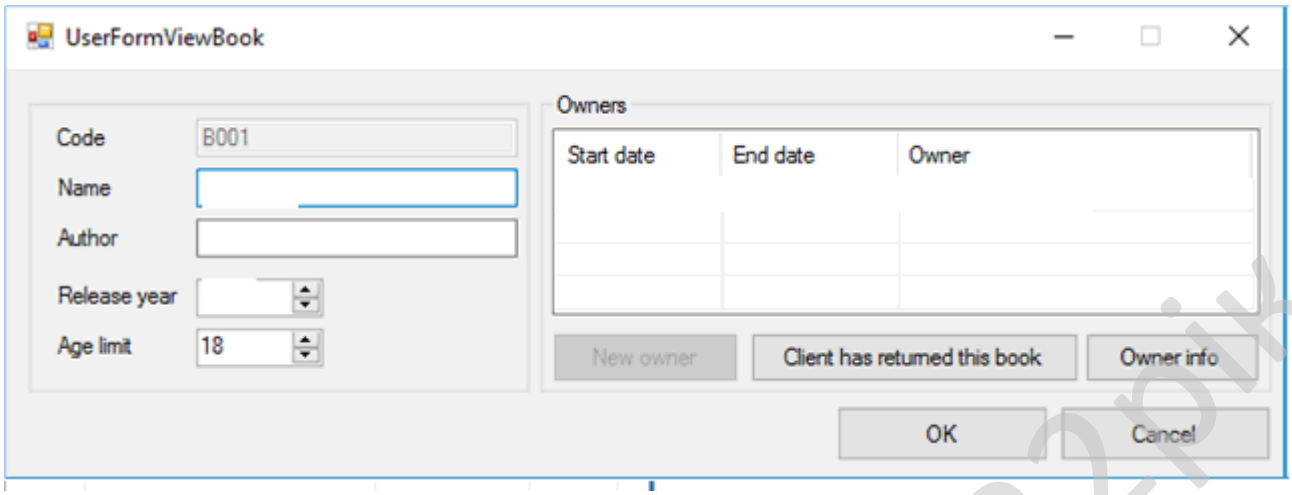

Рисунок 4.11 – Вікно із детальною інформацією про посібники

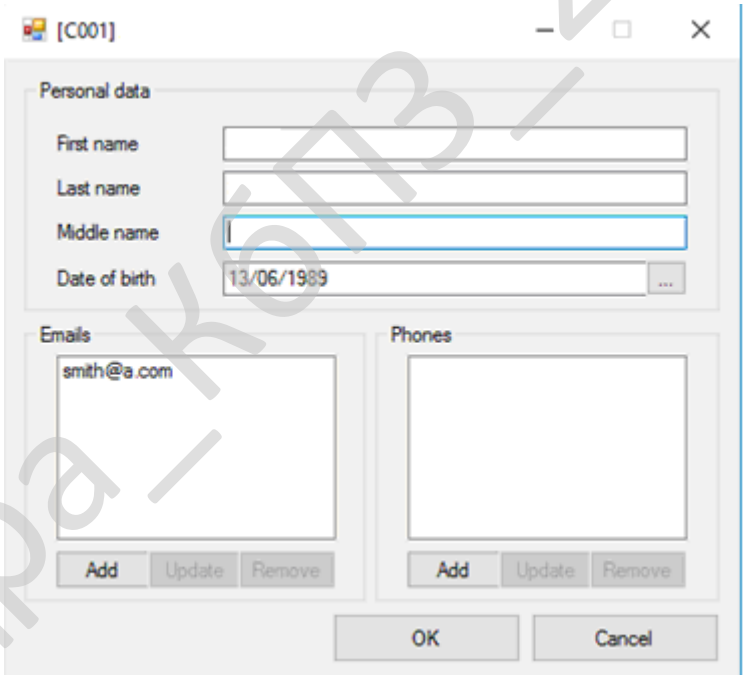

Рисунок 4.12 – Вікно із інформацією про людину

На рисунку 4.12 зображене це вікно. Як бачимо, серед персональної інформації людини є її ПІБ, дата народження, що може бути і не вказаною, а також списки її телефонів та електронних адрес. Натиснення кнопки ОК підтверджує зміни інформації.

Вище вказані вікна показують структуру даних, що зберігаються у розробленій інформаційній системі.

Серед інформації, що зберігається у книзі, можна виділити її код, назву, ім'я автора, рік випуску, вікові обмеження та історію користування. Із інформації із вікна, зображеного на рисунку 4.12, можна сказати, що користувач із кодом C001 та іменем скачав книгу Witcher 13 червня 2012 року. Натиснення кнопки Client has returned this book підтверджує, що клієнт повернув книгу. Із допомогою кнопки New owner можна вибрати нового користувача книгою, а дата початку користування для нього відразу ж встановиться на сьогоднішню. До речі, для книг із віковим обмеженням не можна вказати клієнта із невказаним віком або віком молодше значення вікового обмеження у якості користувача.

4.2 Захист розробленого програмного забезпечення

В першу чергу варто перевірити розроблений додаток автоматизованими засобами, вони зможуть перевірити додаток на найрозповсюдженіші помилки. Вони перевіряють не тільки код написаний розробником, а й бібліотеки та модулі які використовує розробник, а вони складають найбільшу частку зломів. Згідно з останніми перевірками, понад 50% npm модулів, не оновлюються більше декількох років.

Є декілька способів перевірити модулі перед їх використанням, в першу чергу варто переглянути код, та перевірити чи повністю модуль покриває потреби розробника. Він повинен перевірити які ще бібліотеки використовує цей модуль і коли вони останній раз оновлювалися, чим менше він має залежностей, тим краще.

Великі компанії мають спеціальних працівників, які проводять ревізії пакетів перед їх використанням, та складають спеціальні білі списки дозволених модулів.

Сама npm занепокоєна тим, як часто у бібліотеки почали впроваджувати майнерів та віруси для збору приватних даних. Тому вони випустили спеціальний модуль npm audit. Він сканує встановленні модулі в проект і порівнює їх з чорним списком модулів, які містять уразливості. Сьогодні навіть команда npm install підкаже, чи мають встановленні модулі вразливості. А нова команда npm audit fix, пропонує замінити вразливі пакети, або оновити до більш захищених версій, якщо такі існують. Але так можна знайти лише ті модулі, про які вже повідомили користувачі та розробники.

Також для перевірки безпеки додатку, важливим є написання тестів. Зазвичай розробники на AngularJS використовують концепцію Jasmine's Behavior-Driven Development (BDD), при написанні тестів. Jasmine − це вільний фреймворк для тестування коду написаного за допомогою JavaScript, його можна запустити на будь-якій платформі. Його перевагою є зручний інтерфейс, та зрозуміле налаштування. Для тестування одиничних тестів часто використовують бібліотеку Karma, головна мета якого наблизити тестування додатку до його налаштувань в продакшен.

Також важливо використовувати протокол HTTPS, він є набагато надійнішим HTTP. HyperText Transfer Protocol Secure (HTTPS) — забезпечує шифрування даних і дозволяє надійно захищати дані користувачів при передачі в Інтернеті. Сьогодні більшість сайтів використовують саме цей протокол, оскільки він є обов'язковим при передачі приватних даних на сервер, особливо коли передаються паролі або дані кредитних карт. Також варто бути обережним с cookie файлами, які передаються під час авторизації. Зловмисник може перехопити їх та підробити запит на сервер, щоб цього уникнути потрібно використовувати HTTPS протокол.

Також слід брати до уваги поширені CORS та CSRF атаки. CSRF- атака це коли на сторінки, робиться непомітна форма, щоб відправляє запит з приватними даним користувача. Якщо на сайті авторизація відбувається тільки за допомогою кукі, то така атака може бути успішна. Щоб цього уникнути потрібно

використовувати спеціальні токени. Для підпису XMLHttpRequest токен також зберігається в кукі. Тоді JavaScript може прочитати його з домену і додати в заголовок, а сервер - перевірити, що в заголовку міститься коректний токен.

При крос-домених запитів, браузер додає заголовок Origin, який зберігає домен, з якого відбуваються запити. Сервер у цьому випадку повинен перевірити домен і відповісти спеціальним заголовком. Якщо сервер дозволяє доступ, він повинен відправити заголовок Access-Control-Allow-Origin, що зберігає домен запиту. Якщо Access-Control-Allow-Origin відсутній, то сервер завершує запит з помилкою. При таких запитах не передаються куки і заголовки HTTP-авторизації.

5 ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

У Магістерській роботі реалізована інформаційна система, що використовує у якості сховища даних об'єктно-орієнтовану базу даних VelocityDb. Вибір бази даних обґрунтований результатом порівняльного аналізу баз даних, результати якого показані у попередньому розділі.

VelocityDb – об'єктно-орієнтована база даних для роботи із платформою .NET. Функціонал для відправки запитів для читання чи обробки інформації реалізовані у виді методів абстрактного класу SessionBase, наслідники якого і визначають режим підключення до бази даних.

Класи, що зберігаються, повинні бути потомками класу OptimizedPersistable, що містить у собі ідентифікатор об'єкта і методи для додавання об'єкта у базу та для його видалення. Класи, що не є потомками OptimizedPersistable, зберігаються у базі лише той час, поки є на них хоча б одне посилання від об'єктів OptimizedPersistable. Також у бібліотеці реалізовані класи VelocityDbList<T> та VelocityDbDictionary<T>, що являються аналогами List<T> та Dictionary<T>, але водночас є і потомками OptimizedPersistable.

Для відправки запитів зручно користуватись LINQ, як це зображено на рисунку 5.1. LINQ – вбудований у мови програмування платформи .NET Framework (C#, VB.NET, F# тощо) синтаксис мови запитів, що нагадує SQL.

```
1.  private void SendQuery()
2.  {
3.      using (var conn = new SessionNoServer("FolderName"))
4.      {
5.          conn.BeginRead();
6.          var query =
7.              from person in conn.AllObjects<Person>()
8.              where person.BirthDate != null
9.              let day = person.BirthDate.Value.Day
10.             let month = person.BirthDate.Value.Month
11.             where day == DateTime.Today.Day
12.                 && month == DateTime.Today.Month
13.             orderby person.FirstName, person.LastName
14.             group person by person.GetType();
15.
16.         foreach (var personGroup in query)
17.         {
18.             Console.WriteLine(personGroup.Key == typeof(Employee)
19.                 ? "Employees" : "Clients");
20.             foreach (var person in personGroup)
21.             {
22.                 Console.WriteLine("\t{0}", person);
23.             }
24.         }
25.
26.         conn.Commit();
27.     }
28. }
```

Рисунок 5.1– Приклад запиту до бази даних VelocityDb із використанням LINQ

Розглянемо приклад запиту. Нехай у базі даних компанії люди можуть буди як співробітниками, так і клієнтами. Є база даних людей. Потрібно вивести всіх людей у яких сьогодні день народження, але згрупувати клієнтів і співробітників окремо. На рисунку 5.1 зображене вирішення даної задачі.

Функції BeginRead() та Commit() визначають межі транзакції. Функція AllObjects<T>() вилучає об’єкти типу T. СУБД VelocityDb має два види транзакцій – Read і Update, які об’являються викликом методів BeginRead() і BeginUpdate() відповідно і завершені викликом методу Commit().

Read-транзакції дозволяють лише читати інформацію і не допускають внесення ніяких змін у дані. Це зроблено, щоб кілька Read-транзакцій могли безболісно використовуватись одночасно, що значно прискорює вилучення даних при одночасній роботі багатьох клієнтів.

Блок головного меню програми

Для початку користування програмою користувачу потрібно буде авторизуватись. Якщо у працівника існує обліковий запис, можна ввести логін і пароль, у іншому випадку – потрібно натиснути кнопку Register.

Для реєстрації користувача потрібно обов'язково вказати свої прізвище та ім'я, а також логін та пароль.

Розроблений додаток використовує AngularJS та NestJS для демонстрації можливостей сучасних веб-технологій. Також другий розроблений додаток був розроблений для порівняння технологій AngularJS та його новішої версії Angular. В обох випадках у якості серверної частини використовувався платформа NodeJS та фреймворк NestJS.

Оба фреймворки AngularJS та Angular використовуються для створення SPA(Single Page) додатків, які все більше поширюються на просторах Інтернету. В розроблених додатках було продемонстровано стандартний функціонал, який використовується на більшості сайтів: REST API, авторизація та чат на сокетах.

Висновок

Була реалізована інформаційна система, що використовує базу даних VelocityDb у якості сховища даних. Розроблена із використанням .NET Framework + WinForms. База даних із допомогою інформаційної системи була заповнена тестовими даними про книги, клієнтів і працівників. Запити виконуються достатньо швидко.

Слід відмітити, що архітектура ООБД, хоч і повністю повторює архітектуру класів, але для використання ООБД необхідні зміни у архітектуру класів. Так, наприклад, є необхідність класи даних, що зберігаються, робити потомками класу OptimizedPersistable.

Також об'єктні бази даних незручно використовувати для переносу даних на іншу об'єктну базу даних через необхідність знову ж міняти архітектуру.

Розроблений додаток використовує REST API для взаємодії клієнта з сервером, особливо було зручно використовувати RxJS для отримання та обробки

даних. Оскільки його підтримують як Angular на клієнті, так і NestJS на сервері. Дані передавались у форматі JSON, такий формат добре підходить для додатків, які повинні швидко обмінюватися даними. REST API-інтерфейс забезпечують зручну взаємодію з користувачем, легко масштабується.

Завдяки використанню шаблону MVC, додаток має чітку структуру, зручну для підтримки та подальшого розширення, де бізнес-логіка відділена від інтерфейсу користувача. В результаті, додаток легше масштабувати, тестувати та підтримувати. Була розроблена інфраструктура з гнучкою логікою, простою маршрутизацією, інтуїтивно зрозумілі представлення даних.

На рисунку можна побачити доступний інтерфейс та функціонал:

- Головну сторінку та форму реєстрації
- Доступну документації
- Чат створений на сокетах
- Можливість зберігати медіа-файли
- Профільну сторінку юзера

Розроблена програма має простий, зрозуміло інтуїтивний інтерфейс. Якщо користувач вже користувався браузером, він без проблем зможе зрозуміти де який функціонал знаходиться.

Фреймворки дуже допомагають при проектуванні архітектури ПЗ, оскільки в концепціях фреймворків закладені кращі практики розробки, тому просто дотримуючись цих правил можна уникнути багатьох проблем і помилок при проектуванні додатку. Як правило фреймворки представляють собою набір класів та функцій, які активно взаємодіють між собою і підтримують головну методологію фреймворку, а також надають свій функціонал для розширення можливостей додатку використовуючи свої концепції.

Також фреймворки впливають на швидкість розробки, та слідкують щоб розробник не зробив помилок при проектуванні додатку. Оскільки вибір фреймворку такий важливий, треба звертати увагу на його підтримку, а також на кількість розробників які його використовують. Щоб при виявленні якоїсь

помилки, можна було її швидко усунути та повідомити ком'юніті.

Крім фреймворків також можна використати CMS, але вони як правило працюють набагато повільніше, їх важче масштабувати, та додавати новий функціонал, вони витримують менші навантаження. Також у фреймворків рівень безпеки як правило кращий, але розробка на CMS на багато швидша, як і рівень для їх використання, оскільки не потрібно вивчати концепції.

Для демонстрації роботи додатку використовувався ngrok — інструмент, який дозволяє розвернути додаток в Інтернеті, але з певними обмеженнями. Щоб використовувати його повний функціонал потрібно купувати підписку. Сам додаток завантажувався через Docker, використовуючи Dockerfile були встановленні всі налаштування для сервера, був використаний образ node:14.17.1-slim, який оснований на Debian, за допомогою команди COPY були скопійовані всі файли проекту, а за допомогою команди RUN встановлено всі необхідні залежності. Потім встановлювався порт, використовуючи команду EXPOSE, та запускався додаток через команду CMD. База даних та Redis теж підіймались в Докері, вони використовували локальні ресурси підіймались, але сайти mongodb atlas та redis надають обмежені ресурс, для зберігання даних. Для зберігання медіа файлів використовувався minio. При впровадженні програмного забезпечення потрібно урахувати наступні дії:

- Налаштувати доступи
- Підключитись до БД
- Підключитись до Redis
- Підключитись AWS, або сервера який буде зберігати медіа файли
- Підключитись до SendGrid для надсилання повідомлень

Також перед впровадженням системи слід запустити тести, для перевірки працездатності всієї програми. Тестування включає не тільки пошук помилок, але й випробування доступного навантаження на сервер та швидкість запитів. За допомогою результатів тестів, можна отримати:

- Максимальне навантаження;

- Перевірка працездатності запитів;

- Швидкість обробки запиту;

- Виконання поставлених вимог;

- Результати усіх вихідних даних;

- Час виконання функцій;

З розроблене програмного забезпечення можна отримати необхідні дані для прийняття рішень про вибір технології відповідно до заданих вимог розробника. А також використати як шаблон для створення нових додатків, оскільки розроблене ПЗ використовує більшість стандартного функціоналу, який використовується в більшості веб-сайтів.

6 НАУКОВА НОВИЗНА

У магістерській роботі розроблено програмне забезпечення, яке призначено продемонструвати можливості застосування різних типів баз даних та дослідити можливості застосування ООБД.

Метою розробки є програмна реалізація та порівняння застосування ООБД, з метою отримання результатів працездатності та визначення можливостей технологій, для розуміння їх вибору в залежності від вимог розробника.

Об'єктом дослідження є процес визначення можливостей ООБД в сучасному світі веб-розробки та IT технологій.

Предметом дослідження є методи для визначення можливостей ООБД.

Методи дослідження базуються на методах теорії кодування, паттернах проектування, методологіях розробки програмного забезпечення та технологіях взаємодії клієнт-серверних додатках.

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

– Порівняно та отримано результати роботи різни типів база даних, представлена інформація в яких випадках краще використовувати реляційні бази даних а коли ООБД.

– Розроблено шаблони для швидкого ініціювання проектів з технологіями: Angular, AngularJS, NestJS, MongoDB, Redis, Docker, SocketIO, MinIO або AWS.

– Розроблено ПЗ для можливості тестування різних типів баз даних.

7 ДАНІ ПРО ЕКОНОМІЧНУ ЕФЕКТИВНІСТЬ РОЗРОБЛЕНОЇ ПРОГРАМИ

7.1 Техніко-економічне обґрунтування теми магістерської роботи

Після ознайомлення з засобами розробки та необхідними технологіями які використовувались для впровадження та тестування системи, був розроблений план розробки програмного забезпечення. Був визначений необхідний час для розробки та впровадження програми, який склав 22 дні.

В магістерській роботі було проведене дослідження та розроблено два клієнтських додатка та один серверний з використанням бази даних, для виявлення переваг та недоліків кожної з платформ. Розроблені додатки мають функціонал, які використовують більшість веб-сайтів в Інтернеті, що дає можливість чітко порівняти їх можливості. Серед основного функціоналу наявні:

- Можливість спілкуватися за допомогою чата
- Редагувати профіль користувача, змінювати медіа файли
- Надійна реєстрація та авторизація
- Можливість надсилати e-mail сповіщення
- Динамічне оновлення інформації на стороні клієнта

Таблиця 7.1 – Початкові дані

Показники	Позначення	Характеристика або величина
1	2	3
1. Кількість розроблених програм період, шт.	N	3
2. Кількість екземплярів програм, шт.	Ne	240
3. Запланований термін розробки, днів	Fpq	22
4. Група задачі підсистеми управління (1-6)	–	3
5. Ступінь новизни задачі (А, Б, В, Г)	–	Г
6. Складність алгоритму (1, 2, 3)	–	3
7. Кількість макетів вхідної інформації	–	8
8. Кількість форм вихідної інформації.	–	6
9. Мова програмування (1-6)	–	6
10. Попередній досвід (1-6)	–	3
11. Гнучкість проекту ПП (1-6)	–	4
12. Детальність проекту ПП (1-6)	–	4
13. Рівень спрацьованості колективу (1-6)	–	3
14. Ступінь вимірності процесів (1-6)	–	4
15. Необхідна надійність програмного забезпечення (1-6)	–	4
16. Розмір бази даних (порівняно з розміром програми) (1-6)	–	2
17. Складність кінцевого програмного продукту (1-6)	–	3
18. Необхідний рівень забезпечення повторного використання (1-6)	–	3
19. Документованість відповідно до планованого життєвого циклу (1-6)	–	3

Продовження таблиці 7.1

20. Вимоги до швидкодії ПП (1-6)	–	4
21. Обмеження на розміри основного сховища даних (1-6)	–	
22. Різноманітність використовуваних обчислювальних платформ (1-6)	–	4
23. Професійний рівень аналітиків (1-6)	–	2
24. Професійний рівень програмістів (1-6)	–	2
25. Постійність складу команди розробників (1-6)	–	3
26. Досвід розробки додатків (1-6)	–	3
27. Досвід роботи з обчислювальною платформою (1-6)	–	4
28. Досвід роботи з мовою і інструментами середовища розробки (1-6)	–	3
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	–	3
32. Вартість ПЗ у розробника (НМА), грн.	–	36000
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	30%
38. Ставка податку на додану вартість, %	Ндв	20

7.2 Розрахунок трудомісткості розробки програмної продукції

Трудомісткість розробки програмного забезпечення визначається в людино-днях. Такі норми часу охоплюють всі стадії розробки документації до технічного завдання, технічного проекту та впровадження у виробництво. Стадія технічного завдання (ТЗ) включає в себе збирання необхідної інформаії, розрахування можливих методів вирішення задачи, вибір мови програмування, визначення часу на розробку документації, та затвердження технічгог завдання. Стадія технічного проекту (ТП) включає розробку блок-схеми, визначення технологій, розробку документації, визначення вхідних та вихідних даних а також структуру програми. Стадія робочого проекту (РП) включає етап розробки програми та її налаштування. Стадія впровадження (ВП) включає перевірку і вихід програмного забезпечення в промислову експлуатацію. Проце створення програмного забезпечення, в першу чергу починається з планування. Правильна оцінка трудовитрат є основою для планування всього подальшого проекту.

Існує багато методик для визначення трудовитрат, але найбільшу отримала методика COCOMO (Constructive Cost Model) в основі якої лежить визначення об'єму робіт по кількості строчок коду. Використаємо її для визнаення трудоміскості розробки ПЗ для тадії 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;

B – показник ступеня, що визначається співвідношенням:

$$B = 1{,}01 + 0{,}001\sum Wi\ , \tag{7.2}$$

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

Таблиця 7.2 – Масштабуючі показники:

Попередній досвід	3	2,43
Гнучкість проекту ПП	4	2,43
Детальність проекту ПП	4	1,69
Рівень спрацьованості колективу	3	2,97
Ступінь вимірності процесів	4	2,97

$$\sum Wi = 2,43 + 2,43 + 1,69 + 2,97 + 2,97 = 12,49,$$

$$B = 1,01+0,001*12,49= 1,02249,$$

$$T_{ном} = 2,45 \cdot 2,5^{\,1,02249} = 6,2525 \text{ люд-міс.}$$

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

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

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

Таблиця 7.3 – Коефіцієнти трудовитрат:

Необхідна надійність програмного забезпечення	4	1,15
Розмір бази даних (порівняно з розміром програми)	2	0,93
Складність кінцевого програмного продукту	3	1
Необхідний рівень забезпечення повторного використання	3	1
Документованість відповідно до планового життєвого циклу	3	1

Продовження таблиці 7.3

Вимоги до швидкодії ПП	4	1,11
Обмеження на розміри основного сховища даних	1	1
Різноманітність використовуваних обчислювальних платформ	4	1,15
Професійний рівень аналітиків	2	1,22
Професійний рівень програмістів	2	1,16
Постійний склад команди розробників	3	1
Досвід розробки додатків	3	1
Досвід роботи з обчислювальною платформою	4	0,88
Досвід роботи з мовою і інструментами середовища розробки	3	1
Досвід роботи з програмними інструментами розробки	3	1
Розробка ПО для декількох серверів одночасно	2	1,10
Вимоги до дотримання встановленого графіка робіт	3	1

$$\Pi v_j = 1{,}15*0{,}93 *1*1*1*1{,}11*1*1{,}15*1{,}22*1{,}16*1*1*0{,}88*1*1*1{,}10*1 = 1{,}8702$$

$$T_{уточн} = 6{,}2525*1{,}8702 = 11{,}6934 \text{ люд-міс.}$$

Визначаємо підсумкові трудовитрати, люд-дні:

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

де: C – визначений емпірично коефіцієнт;

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

$$T_{РП} = 0,3 \cdot 2,66 \cdot 11,6934^{0,33+0,2(1,02249-1,01)} \cdot 100 = 180,75 \text{ люд/день.}$$

Для зручності визначення загальної трудомісткості на розробку програмного забезпечення результати розрахунків по стадіям зводимо до таблиці 7.4.

Таблиця 7.4 – Визначення трудомісткості розробки програмного забезпечення

Стадії розробки	Трудомісткість за типовими нормами та розрахунками	
	Величина, люд/дні	Підстава
Технічне завдання	9	Додаток №5
Ескізний проект	10	Додаток №6
Технічний проект	6	Додаток №7
Робочий проект	180,75	Ф 7.1-7.4
Впровадження	18	Додаток №11
Всього	223,75	–

7.3 Визначення чисельності виконавців і планового фонду зарплати

Кількість інженерів-програмістів розраховується за формолою:

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

$T_{пз}$ - трудомісткість розробки програми

N - кількість розроблених програм за рік

$F_{рq}$ - запланований термін розробки

$H_{ев}$ - невиходи за поважних причин, днів

$$Ч = (223,75 * 1) / (22- 3) = 11,7763$$

Кількість працівників для проведення технічного обслуговування визначається від наявності технічних засобів і норм витрат часу на виконання профілактичних робіт на протязі року.

Таблиця 7.5 – 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	385	3	1155	19
Монітор	160	3	480	8
Клавіатура	140	3	420	7
Маніпулятор «мишка»	30	4	120	2
Принтер матричний	185	1	185	3
Принтер лазерний	355	1	355	6
Принтер струминний	300	1	300	5
Сканер	155	1	155	2.5
Концентратор-маршрутизатор	155	2	310	5
Кабельні господарства ЛВС на 1 м п.	2,5	5	12,5	0.2
Копіювальний апарат	285	1	285	5
Усього за рік:			$З_ч$	62,7

Норми часу на профілактику ЕОМ дорівнюють 10%.

$$Ф^c_{dp} = (З_ч * n_{mic})/1,2 \quad (7.6)$$

$$Ф^c_{dp} = (62,7 * 1)/1,2 = 52,25 \text{ год}$$

Визначаємо необхідну кількість ставок штатного персоналу сектора ТО:

$$Ч_{ел} = Ф^{с}_{др} / (F_{рq} * T_{зм}) \quad (7.7)$$

$$Ч_{ел} = 52{,}25 /(22 \cdot 8) = 0{,}296875 \text{ ставки.}$$

Чисельність персоналу інженерів-системотехніків, адміністраторів мереж, 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 7.6 – 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 20012 R2, серверу доступу ADSL (ОС Linux), налаштування ADSL, VPN, PPPoE, Frame Relay, Wi-Fi	1,5	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	1,5	
Всього		4	
Посада	Вид роботи	Час	К-ть штатних одиниць
Продакт-менеджер	Презентації нової продукції, пошук каналів збуту	2	0,5
	Підтримка постійних клієнтів	1	
	Оформлення договорів, ведення тендерів	0,5	
	Контроль взаєморозрахунків з постачальниками	0,5	
Всього		4	
Дизайнер WEB	Розробка концепції оформлення та інтерфейсу сайту, оптимізація дизайну існуючих, проектує їх структуру та навігацію	2	0,5
	Створення графічних і стилістичних елементів сайту	1	

Продовження таблиці 7.6

	Оформлення банерів і промо-сторінок	0,5	
	Розміщення графіки і контенту на Інтернет сторінках	0,5	
Всього		4	
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 7.7 - Штатний розклад виконавців

Посада	Кількість ставок	Середньомісячний оклад, грн.	Всього за період розробки, грн.
Керівник (IT-менеджер)	1	10000	10000
Продакт-менеджер	1	8000	8000
Інженер-програміст	3	8000	24000
Інженер-електронщик	1	8000	8000
Інженер-системотехнік	0,5	8000	4000
Адміністратор мережі	0,5	8000	4000
Системний програміст	1	8000	8000
Дизайнер WEB	0,5	8000	4000
Інженер-верстальник	0,5	8000	4000
Бухгалтер-економіст	1	8000	8000
Всього за період розробки	$R_{сп} = 10$	-	$Ф_{роб} = 82000$

Розрахуємо середньоденну зарплату одного виконавця:

$$З_{cd} = Ф_{роб} / (R_{сп} * F_{рq}) \quad (7.8)$$

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

$$З_{cd} = 82000/ (10*22) = 372 \text{ грн}$$

7.4 Розрахунок капітальних вкладень та амортизаційних відрахувань у розробника

Балансова вартість складається з ціни, транспортно-заготівельних витрат, вартості будівельно-монтажних та пуско-налагоджувальних робіт, а також витрат на випробовування у виробничих умовах. Норма амортизації визначається в залежності від нормативного строку служби.

Балансова вартість будівель (В буд) визначається з урахуванням кількості робочих місць виконавців (R ñї 1), питомої площі на одне робоче місце (S у = 8...12 м 2) та вартості одного квадратного метра виробничої площі (Ц пл = 1500 грн. і вище).

$$В_{буд} = R_{сп} * S_{у} * Ц_{пл} \quad (7.9)$$

де: $R_{сп}$ – кількість робочих місць виконавців, шт. Приймаємо 13 робочих місць;

$S_{у}$ – питома площа на одне робоче місце, м^2;

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

$$В_{буд} = 10 * 8 * 1500 = 75000$$

Вартість передавальних пристроїв (електромережі, водопровід, тепломережі тощо) можна орієнтовно прийняти до 12% вартості будівель і у даному випадку вона складе: 9000 грн.

$$В_{перед} = 0,01В_{буд}П_{перед} \quad (7.10)$$

$$В_{перед} = 0,01*75000*5 = 3750$$

Балансову вартість господарського інвентарю (меблі, стелажі, крісла тощо) доцільно розрахувати за орієнтовною нормою від 800 грн. до 6000 грн. на одне робоче місце:

$$І_{нв} = R_{сп} * Ц_{м} \quad (7.11)$$

де: $Ц_{м}$ – ціна меблів для одного робочого місця, грн.

$$І_{нв} = 10 \cdot 4000 = 40000 \text{ грн.}$$

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

Специфікація на обчислювальну техніку наведена в таблиці 7.8.

Дані по оптовій ціні на обладнання та комплектуючі вибирались фірми Rozetka за 11.11.21– джерело http://rozetka.com.ua/

Вартість вимірювальних пристроїв можна прийняти до 5% від вартості персональних комп'ютерів:

$$В_{вим.пристр} = 0,01В_{пк}П_{вим.пристр}, \text{ грн} \quad (7.12)$$

$$В_{вим.пристр} = 0,01 * 51150 * 5 = 2557,5$$

де $В_{ПК}$ – вартість персонального комп’ютера;

$П_{вим.пристр}$ – відсоток, який встановлює залежність між вартістю вимірювальних пристроїв та вартістю персонального комп’ютера.

Таблиця 7.8 – Специфікація

Найменування комплектуючої або обладнання	Тип	Оптова ціна
Персональний комп’ютер		15500
Системний блок		

Продовження таблиці 7.8

Процесор	Intel Core i3-10105F 3.7GHz/6MB (BX8070110105F) s1200 BOX	2 599
Системна плата	Asus Prime H410M-K (s1200, Intel H410, PCI-Ex16)	2 118
Відеокарта	Gigabyte PCI-Ex GeForce GT 710 2048MB DDR3 (64bit) (954/1800) (HDMI, DVI, VGA) (GV-N710D3-2GL)	1 949
Жорсткий диск	Toshiba P300 1TB 7200rpm 64MB HDWD110UZSVA 3.5 SATA III	1 049
Оперативна пам’ять	AMD 8 GB DDR4 2400 MHz (R748G2400U2S-U)	792
DVD-привод	Asus DVD±R/RW USB 2.0 SDRW-08D2S-U LITE Black External	861
Корпус	GameMax MT508-NP-2U3	751
Кулер	–	–
Кардрідер внутрішній	Gembird SD/MMC/RS-MMC/MicroSD картами памяти и 2.5" HDD/SSD (FDI2-ALLIN1-03)	468
Інше (Клавіатура, мишка)	A4Tech KM-72620D USB	399

Продовження таблиці 7.8

Монітор	23.8" Acer SA240YBbmipux (UM.QS0EE.B01) USB Type-C 15W	4 499
Принтер лазерний	HP LaserJet 107a (4ZB77A)	4 421
Найменування комплектуючої або обладнання	Тип	Оптова ціна
Принтер струминний	Canon PIXMA Ink Efficiency E414 (1366C009)	1 709
Сканер	IRISCan Express 4 (458510)	4 019
Копіювальний апарат	RICOH MP1600	8 344
Пристрій безперебійного живлення	LogicPower LPM U1400VA-P (LP10394)	3 014

Витрати на транспорт, монтаж та випробування можуть бути прийняті в межах до 10% від оптової ціни.

Для визначення необхідної кількості капітальних вкладень складемо таблицю 7.9.

Таблиця 7.9 – Балансова вартість обчислювальної техніки

Найменування обчислювальної техніки	Кількість, шт.	Середня ціна за одиницю, грн.	Витрати на транспортування, монтаж та випробовування.	Загальна вартість, грн.
Персональні комп’ютери	3	15500	4650	51150
Принтер лаз.	1	4 421	442,1	4863,1

Продовження таблиці 7.9

Принтер струм.	1	1 709	170,9	1879,9
Сканери	1	4 019	401,9	4420,9
Копіюв. апарат	1	8 344	834,4	9178,4
Всього	–	–	–	71492,3

Таблиця 7.10 – 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			
1. Будівлі	*75000*	-	-
2. Передавальні пристрої	*9000*	-	-
Всього по групі	84000	5	4200
Група 4			
3. Обчислювальна техніка	71493	50	35746,5
4. Інше обладненння	2557,5	50	1278.75
Всього по групі	74050,5	50	37025,25
Група 5			
5. Транспортні засоби	714,92	20	142,944
Група 6			
6. Вимірювальні пристрої	2557,5	-	-
7. Господарський інвентар	40000	-	-
Всього по групі	42557,5	25	10639,375
8. Нематеріальні активи	36000	50	18000
Разом	$К_р$ = 237322,92		$А_р$ = 70007,569

Примітка: вартість не враховуємо. Тому вартість транспортних засобів приймаємо рівним нулю.

7.5 Визначення собівартості розробки та ціни програмної продукції

Визначимо основну зарплату виконавців:

$$З_о = (З_{сд} * Т_{пз})/N_е \quad (7.13)$$

де: N_e – кількість екземплярів програм, шт.

$$З_о = 372 * 223,75/ 240 = 347$$

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

$$З_д = З_о * Н_д * 0,01, \quad (7.14)$$

де: H_q – норматив додаткової зарплати, %.

$$З_д = 347 * 10 * 0,01 = 34,7$$

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

$$С_{оц} = 0,01 * Н_с * (З_о + З_д), \quad (7.15)$$

де: H_c – відрахування на соціальні потреби, %.

$$С_{оц} = 0,01 \cdot 22(346 + 34,7) = 83,754$$

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

$$Г_{осп} = З_о * Н_с * 0,01, \quad (7.16)$$

де: $Н_г$– загальногосподарські витрати, %.

$$Г_{осп} = 347*15*0,01 = 52,05$$

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

$$З_М = (З_{М1} + З_{М2} + З_{М3})/N_е, \quad (7.17)$$

де: $З_{М1}$– вартість паперу, грн.;

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

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

$N_е$ – кількість екземплярів програм, шт.

Згідно виданих викладачем норм приймаємо одну пачку паперу на місяць розробки. Тоді, враховуючи, що вартість пачки паперу складає $Ц_п$ = 100 грн., визначаємо вартість паперу за період розробки $N_м$ = 1 міс:

$$З_{М1} = Ц_п \cdot N_м. \quad (7.18)$$

$$З_{М1} = 100 \cdot 1 = 100 \text{ грн.}$$

Згідно виданих викладачем норм до вартості запам'ятовуючих пристроїв входить вартість CD дисків в кількості, що дорівнює кількості екземплярів програм та одного DVD диска для збереження резервної копії програми:

$$З_{М2} = \sum Ц_д \,, \tag{7.19}$$

де: $Ц_д$ – вартість дисків

Наприклад: CDR TDK 700Mb, 80Min, 52x Cake box – 5 грн/шт., DVD-R LG 4,7Gb, 16x speed Cake box - 7 грн/шт.

$$З_{М2} = 240 \cdot 12 = 2880 \text{ грн.}$$

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

$$З_{М3} = \sum Ц_з., \tag{7.20}$$

де: $Ц_з$ – вартість розхідних матеріалів друкуючих пристроїв: картридж для CANON LBP-3010 Black Canon 712 – 574 грн.; картридж для EPSON STYLUS PHOTO R390 – 558 грн.; картридж для CANON IR-1022A – LJ Q2612A Cart. HP LJ 1010/1012/1015/3015/3020/3030 (2500 стр.) – 570 грн.

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

$$З_М = (100 + 2880 + 1702)/240 = 19{,}5 \text{ грн.}$$

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

$$О_п = З_о * Н_п * 0,01 \quad (7.21)$$

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

$$О_п = 347 * 15 * 0,01 = 52,05 \text{ грн.}$$

Визначимо витрати на амортизацію основних фондів з урахуванням загальної річної суми амортизаційних відрахувань та кількості екземплярів програм ($N_е$ = 240 прим.):

$$А_м = (А_р * N_{міс})/(N_е * 12) \quad (7.22)$$

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

$$А_м = 70007,569 * 1/(240*12) = 24,30 \text{ грн.}$$

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

$$С_п = З_о + З_д + С_{оц} + Г_{осп} + З_М + О_п + А_м \quad (7.23)$$

$$С_п = 347+34,7 + 83,754 +52,05 + 19,5 +52,05 + 24,30 =613,354 \text{ грн.}$$

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

Для даного програмного забезпечення рівень рентабельності складає 30%.

$$П_р = 0,01 * Р_п * С_п \quad (7.24)$$

де: P_n – рівень рентабельності, %.

$$П_р = 0,01 * 30 * 613,354 = 184,0062 \text{ грн.}$$

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

Оптова ціна підприємства складається з повної собівартості і планового прибутку. Визначаємо оптову ціну підприємства:

$$Ц_п = С_п + П_р \quad (7.25)$$

$$Ц_п = 613,354 + 184,0062 = 797,3602$$

Величина податку на додану вартість:

$$ПДВ = 0,01 \square 20 * 797,3602 = 159,47204$$

де: Нпдв – ставка податку на додану вартість.

Відпускна ціна програмної продукції:

$$Ц = Цп + ПДВ \quad (7.26)$$

$$Ц = 797,3602 + 159,47204 = 956,83224$$

Таблиця 7.11 – Нормативна калькуляція собівартості розробки програмного забезпечення задачі

Найменування статей витрат	Позначення	Величина, грн.
1	2	3
Основна зарплата виконавців	$З_о$	347
Додаткова зарплата виконавців	$З_д$	34,7
Відрахування на соціальні потреби	$С_{оц}$	83,754
Загальногосподарські витрати	$Г_{осп}$	52,05
Витрати на матеріали	$З_М$	19,5
Освоєння нових операційних систем, мов програмування	$О_п$	52,05
Амортизація основних фондів	$А_м$	24,30
Повна собівартість програмного забезпечення	$С_п$	613,354
Плановий прибуток	$П_р$	184,0062
Ціна підприємства $Ц_п = С_п + П_р$	$Ц_п$	797,3602
Податок на додану вартість $ПДВ = 0.01 \cdot Н_{дв} \cdot Ц_п$	$ПДВ$	159,47204
Відпускна ціна програмної продукції $Ц = Ц_п + ПДВ$	$Ц$	956,83224

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 7.10.

Таблиця 7.12 – Розрахунок об’єму капітальних вкладень у споживача програмної продукції

Найменування капітальних вкладень	Сума за варіантами, грн.	
	Базовий	Новий
Вартість програмної продукції	–	956,83224
Всього капітальних витрат	–	956,83224

7.7 Визначення експлуатаційних витрат

Експлуатаційні витрати у споживача програмної продукції визначаємо при умові роботи підсистеми на протязі року.

Таблиця 7.13 – Розрахунок експлуатаційних витрат у споживача програмної продукції

Найменування статей витрат	Позначення	Сума витрат за варіантами, грн.	
		Базовий	Новий
1. Витрати на технічне обслуговування	$З_р$	15030	9662
2. Витрати на електроенергію	$З_{ел}$	1188	743
3. Витрати на амортизацію	$З_{ам}$	0	1580
Всього витрат за рік	$І$	16218	11985

Витрати на профілактичні роботи:

$$З_р = Т_р * З_г * (1 + 0,01 * Н_д) * (1 + 0,01 * Н_е) \quad (7.27)$$

де: $Т_р$ – кількість годин обслуговування кожного комп’ютера за рік, год;

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

Після купівлі нового програмного забезпечення кількість профілактичних годин робіт зменшилася з 280 годин на рік до 180 годин на рік, тому витрати на технічне обслуговування зменшилися з:

$$З_{рбаз} = 280 * 40 *1,1 * 1,22 = 15030,$$

$$З_{рнов} = 180 * 40 * 1,1 * 1,2 - 9662.$$

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 ($Ц_{ел}$):

$$З_{ел} = П_{ел} \cdot Т_р \cdot Ц_{ел,} \quad (7.28)$$

$$З_{ел\ баз} = 0,45 \cdot 1200 \cdot 2,2 = 1188 \text{ грн.},$$

$$З_{ел\ нов} = 0,45 \cdot 750 \cdot 2,2 = 743 \text{ грн.}$$

Витрати по амортизації визначаються на основі норм амортизаційних відрахувань, вартості програмної продукції і основних фондів. Для розрахунку складаємо таблицю 7.14.

Таблиця 7.14 – 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	–	956,83224	–	239.20806
Всього відрахувань	-	–	956,83224	–	239.20806

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:

$$Е_в = (Ц_в — С_п) * N_е - Е_р * К_р \quad (7.29)$$

де: $К_р$ – балансова вартість основних фондів розробника, грн.

$$Е_в = (797,3602 — 613,354) * 240 — 0,15 * 237322,92 * 1/12 = 41194,9515$$

Визначимо період окупності додаткових капітальних вкладень у виробника програмної продукції:

$$Т_в = К_р / ((Ц_в — С_п) * N_е), \quad (7.30)$$

$$Т_в = 237322,92/((797,3602 — 613,354) * 240 * 12/1) = 0,447831604$$

Таблиця 7.15 – Показники економічної ефективності програмної продукції

Найменування показників	Одиниця виміру	Величина
1. Кількість екземплярів програми	Прим.	240
2. Повна собівартість розробленої програми	Грн.	613,354
3. Ціна розробленої програми	Грн.	797,3602
4. Плановий прибуток від реалізації розробленої програми	Грн.	184,0062
5. Рентабельність програмної продукції	%	30
6. Об’єм додаткових капітальних вкладень у виробника програмної продукції	Грн.	237322,92

Продовження таблиці 7.15

7. Загальний прибуток від реалізації програмної продукції	Грн.	44161
8. Величина економічного ефекту при виготовлені програмної продукції	Грн.	41194
9. Період окупності додаткових капітальних вкладень у виробника програмної продукції	Рік	0,4
10. Об'єм додаткових капітальних вкладень у споживача програмної продукції	Грн.	956,83224
11. Величина економічного ефекту у користувача програмної продукції	Грн.	3993,79
12. Період окупності додаткових капітальних вкладень у користувача програмної продукції	Років	0,22

Визначимо величину економічного ефекту у користувача програмної продукції за формулою:

$$E_{сп} = (I_б - I_н) — E_н (K_н - K_б), \quad (7.31)$$

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.

$$E_{сп} = (16218-11985) — 0,25 * 956,83224 = 3993,79194 \text{ грн.}$$

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

$$T_{сп} = (K_н - K_б,)/(I_б - K_н), \quad (7.32)$$

$$T_{сп} = 956,83224 /(16218-11985) = 0,226041162 \text{ роки}$$

Показники економічної ефективності програмної продукції зводимо до таблиці 7.15.

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 З ОХОРОНИ ПРАЦІ ТА ТЕХНІКИ БЕЗПЕКИ

8.1 Аналіз умов праці програміста

Інтернет відіграє важливу роль у житті сучасної людини. Кожного дня мільйони людей використовують Інтернет для пошуку необхідної інформації, спілкуванні у соціальних мережах, перегляду новин. Багато людей користуються Інтернетом у професійних цілях, оскільки завдяки Інтернету зявилося багато нових професій. Тому для веб-розробника так важливо розробити зручний інтерфейс для зручного сприйняття інформації, та необхідний функціонал, який буде відповідати необхідним вимогам та навантаженням. Все це вимагає багато багато часу та великого навантаження з боку розробників.

Тому так важливо слідкувати за умовами праці, в яких відбувається робочий процес. Оскільки захворювання можуть бути спричинені надмірним фізичним або розумовим навантаженням, через велику нервово-емоційну напругу, або через виробниче середовище. В даному розділі магістерської роботи проведемо аналіз основних чинників при роботі програміста.

При роботі за компютером, розробник має велике зорове навантаження, тому йому необхідне належне освітлення приміщення. Якщо в приміщенні недостатньо природного освітлення, потрібно використовувати спеціальні світильники. Також оскільки розробник значний час працює з електричними приборами є можливість, бути ураженим електричним струмом, тому потрібно дотримуватись всіх необхідних норм. Серед основних чиників, які впливають на розробників під час трудової діяльності можна виділити[46]:

1. Рівень освітлення в приміщенні.
2. Температура, вологість в приміщенні.
3. Рівень шуму на робочому місці.

4 .Напруга в електричному ланцюзі, електричні показники.

Розберемо кожний з чиників окремо, проаналізуємо які повинні бути стандарти кожного з чиників, відповідно до правил з охорони праці.

Рівень штучного освітлення

Головним документом для встановлення норм необхідних показників освітлення є ДБН В.2.5-28:2018 «Природне та штучне освітлення» [47].

Сьогодні найбільш розповсюдженішими є світлодіодні ламп. В середньому світловіддача від таких ламп знаходиться на рівні 80-120 Лм/Вт [48]. Джерелом живлення прийнято вважати електричну мережу у 220В. А освітленіть робочого приміщення повина бути Е =300-500 Лк, оскільки робота програміста відноситься до робіт середньої точності з присвоєнням розряду зорових робіт IV[49].

Рівень сітла повинен бути достатнім, щоб працівник міг працювати без навантаження на зір. Це залежить від системи освітлення, кількості світильниув, їх типу та розміщення у приміщенні. Допустиме значення освітленості робочої поверхні приймається Е = 400 лк [50].

Для покращення освітлення комп'ютерній лабораторії будуть використовуватися світлодіодні лампи, а саме FL-LED T8-900 світловий потік яких F=1500лм.

Мікроклімат робочої зони: температура, відносна вологості, швидкість руху повітря

Головним документом для встановлення норм мікроклімату робочої зони є ДСН 3.3. 6.042 -99 «Державні санітарні норми мікроклімату виробничих приміщень» [51].

Праця програміста за важкістю відноситься до легкої фізичної роботи категорії Іа [49]. Де вказано, що в приміщенні, я кому знаходиться компютерне обладнення, повнині бути встановлені певні норми, оскільки офісна техніка є джерелом тепловиділень, що може спричинити підвищення температури. Серед

потимальних параметрів для роботи встановлена температура 23 – 25^{0}С і вологість на рівні 40 – 60% в залежності ві періоду року.

Для підтримки комфортної температури можна використовувати як організаційні методи, наприклад розпорядок дня, так і технічне обладнення, наприклад кондиціонери, вентіляцію. Як правило в холодний період часу використовуються додаткове опалення для підтримання комфортної температури, а в літку встановлюються кондеціонери.

Рівень шуму на робочому місці

Як правило при використанні великої кількості компютерів в одному приміщенні, через гудіння, рівень шуму має значення більше норми. Допустима норма становить менше 50 дБ [52].

Гучний шум негативно впливає на умови праці та організм людини. Якщо шум триває тривалий час цу може спричинии головні болі, біль у вухах, підвищення стомлюваності, зниження концентрації та уваги. Такі симптоми можуть викликати стресові ситуації у людини. Все це шкодить продуктивності працініка та його стану здоровья.

Щоб встановити необхідний рівень шуму, використовують додаткову звукоізоляцію. Для цього найчатіше використовуються мати та плити із скляного та мінерального волокна, м'які плити з деревних стружок, картон, гуму, утеплений лінолеум, а також заміна вікон на звукоізолюючі.

8.2 Заходи профілактики при роботі з комп’ютерною технікою

Санітарно-гігієнічні норми є важливим критерієм при роботі в приміщенні. Від них залежить здоровє працівників, їх рівнень працездатності, втомлюваність. Щоб всього цього уникнути потрібно стежити за нормами на робочому місці.

Якщо говорити про електробезпеку в приміщенні, то в приміщенні необхідно устаткування розподільних щитів спеціальними розетками з

заземлюючими контактами, повині бути заземлені всі прилади і пристрої, час від часу повина проводитися перевірка всіх приладів, щорічна здача іспитів з охорони праці.

Для оптимальних показників мікроклімату та освітленості потрібні використовувати дефлектор, для організації вентиляції, та повітрообміну. Та перевірку освітленості в приміщенні згідно відділом охорони праці, щоб відповідати номам для зорової роботи .

Ще однією проблемою з якою часто зустрічаються програмісти є мала рухливість та повний сидячий робочий день. Тому рекомендується час від часу робити невеликі перерви, під час обіднього перериву вживати їжу не на робочому місці. У окремих випадках, коли при дотриманні всіх санітарних норм, працівник все одно себе погано почуває, дозволяється індивідуальний підхід для обмеження роботи з обчислювальними пристроями. Тривалість роботи за комп'ютером не повинна безперервно тривати більше 4 годин.

Для зменшення зорового та нервово-емоційного навантаження, та поліпшення мозкової діяльності рекомендується робити перерви для психологічного та фізичного розвантаження.

В приміщеннях також повинні бути протипожежне обладнання, та інструкція у разі надзвичайних ситуаціях. Повинна бути особа, яка відповідає за пожежну безпеку, перевіряє обладнання, та системи протипожежного захисту а також щорічне проведення інструктажів серед працівників.

Автоматична пожежна сигналізація повинна відповідати вимогам ДБН ДБН В.2.5-56:2014, яке вимагає використання вогнестійких кабелів та автоматичну роботу системи оповіщення та евакуації людей у випадку надзвичайної ситуації [53].

При перевірці, приміщення повинно відповідати всім нормам пожежної безпеки. Це виконується за допомогою перевірки пожежної охорони та техніки, проведенню інструктажу і своєчасне інформування пожежної охорони про несправність пожежної техніки, впровадження систем протипожежного

захисту. Організаційні та технічні заходи, спрямовані на попередження виникнення пожежі, обмеження поширення вогню та успішної евакуації людей.

8.3 Розрахнок занулення глухозаземленої нейтралі

Занулення, як основний засіб захисту, застосовується в електроустановках до 1 кВ з глухозаземленою нейтраллю трансформатора або генератора [54]. Початкові дані для розрахунку занулення глухозаземленої нейтралі трансформатора виробничого приміщення:

Загальна потужність: P = 5 кВт.

Кількість електродвигунів: m = 10

Потужність освітлювальних приладів: Po = 3 кВт.

Довжина магістрального кабеля: LM =70 м.

Довжина розгалудження: l=22м.

Лінійна напруга U=380 В.

Фазна напруга Uф=220 В.

Визначаємо силу номінального струму електроустановки:

$$Iном = Imax = (P * 1000) / (\sqrt{3} * Uл * cosф) \quad (8.1)$$

$$Iном = Imax = (5 * 1000)/(\sqrt{3} * 380 * 0,85) = 8,9 А$$

де: P − номінальна сумарна потужність електроприладів, кВт;

Uл − лінійна напруга, В;

cosф − коефіцієнт потужності, приймається в залежності від типу електрообладнання в межах 0,8..0,87.

Визначаємо силу пускового струму електродвигуна:

$$Iпус = 5 * Iном \quad (8.2)$$

$$Іпус = 5 * 8,9 = 44,5 А$$

Визначаємо номінальну силу струму апарата захисту:

$$Ін = Іпус/b \quad (8.3)$$

$$Ін = 44,5 /2,5 = 17,8 А$$

b − коефіцієнт пуску електродвигуна − для легких умов пуску − 2,5..3.

Вибираємо запобіжник ПН 2-100 з плавкою вставкою Іном = 50 А. Визначаємо найменше допустиме по умовам спрацьовування захисту значення сили струмукороткого замикання:

$$Іктіп = Ін * К \quad (8.4)$$

$$Іктіп = 50 * 3 = 150 А$$

Ін−номінальний струм апарата захисту;

К − коефіцієнт надійності;

Знаходимо переріз провода або кабеля розгалуження з умови допустимого нагрівання:

$$Ідоп = Івст/а \quad (8.5)$$

$$Ідоп = 50 / 3 = 16,6 А$$

Вибираємо площу перерізу 10 мм^2 (Sф) при числі проводів і = 4 розташований у повітрі. Визначаємо максимальний робочий струм:

$$Iроб = Ко(Кз*(Р*1000)/(\sqrt{3}*Uл*cosф)*m+Кз*(Ро*1000)/(\sqrt{3}*Uл*cosф)) \quad (8.6)$$

$$Iроб = 0,75*(0,85*((5*1000)/(1,73*380*0,85)*10+(3*1000)/(1,73*380*0,85))=60,4 \text{ А}$$

Ко − коефіцієнт одночасності роботи групи електроприймачів;

Ко = 0.7...0.8; Кз = 0.8...0.9;

Кз – коефіцієнт завантажених електродвигунів;

Ро =3кВт − потужність освітлювальної мережі;

Визначається струм короткочасного перевантаження магістрального кабеля:

$$Iпер = Ко(Кз*(Р*1000)/(\sqrt{3}*Uл*cosф)*n+Кз*(Ро*1000)/(\sqrt{3}*Uл*cosф))+Iпус \quad (8.7)$$

$$Iпер = 0,75*(0,85 *(5*1000)/(1,73*380*0,85)*9+0,85*(30*1000)/(1,73*380 * 0,85)) + 44,5 = 130,0643 \text{ А}$$

Струм спрацювання електромагнітного розчеплювача додатково перевіряємо по максимальному струму перевантаження лінії:

$$Iспр >= 1,25 * Iпер \quad (8.7)$$

$$Iспр >=1,25 * 130,0643 = 162,5803 \text{ А}$$

Приймаємо Iспр = 162,5803 А. Вимикач : А3714Б.

Вибираємо площу перерізу Sф магістрального кабеля (провідника) по Iдоп. Sф = 70mm^2 , − кабель АВРГ прокладений в землі, і=3 (число проводів). Вибрану площу перерізу перевіряємо для автоматів з електромагнітним розчеплювачем:

$$Iдоп >= Iспр/4,5 \quad (8.8)$$

$$Iдоп >= 162/4,5 = 36 А$$

Проводимо узгодження з номінальним струмом автомата:

$$Iдоп = Iспр/3 \quad (8.9)$$

$$Iдоп = 162/3 = 54 А$$

Значення 36 і 54 А. менше ніж Imax= 60,4 А., значить площа перерізу кабеля вибрана вірно. Визначаємо потужність трансформатора:

$$Nтр = ((Кп * Рном)/cosф) \quad (8.10)$$

$$Nтр = (0,7 * 53)/0,8 = 46,375 кВ*А$$

Рном − сумарна потужність електроприймачів, кВт;

Cosф − середній коефіцієнт потужності електроприймачів (0,8);

Кп − коефіцієнт попиту (0,7);

Одержане значення потужності трансформатора округляємо до ближчого стандартного значення. Визначаємо опір трансформатора Z_T. Вибираємо трансформатор на 40 кВА (Z_T =0,562 Ом). визначаємо орієнтовно площу перерізу провідника. Для магістрального кабеля:

$$Sn_1 >= 0,5 * Sф \quad (8.11)$$

$$Sn_1 >= 0,5 * Sф = 0,5*70=35 мм^2$$

Визначаємо для розгалуження:

$$Sn_2 >= 0{,}5*10 = 5 \text{ мм}^2$$

Округляємо ці значення до найближчих більших 35 мм^2 (Sn1), і 6 мм^2 (Sn2). Визначаємо активний і індуктивний опір фазного і нульового захисного провідників на дільницях 1 і 2:

$$Rф = p * (Lм/Sф_1) + p * (L/Sф_2) \quad (8.12)$$

$$Rф = 0{,}028 * (70/70) + 0{,}028 * (22/10) = 0{,}0896 \text{ Ом}$$

$$Rn = p * (Lm/Sn_1) + p * (L/Sn_2) \quad (8.13)$$

$$Rn = 0{,}028 * (70/35) + 0{,}028 * (22/6) = 0{,}1586 \text{ Ом}$$

Для окремо проложених нульових провідників иого приимають рівним 0,6 Ом/км. При прокладці кабелем, або в сталєвих трубах індуктивним опором нехтують.

Знаходимо дійсне значення (модуль) струма однофазного короткого замикання:

$$Iкр = Uф/((Zт/3) + \sqrt{((Rф + Rn)^2 + (Xф + Xn + X'n^2))} \quad (8.14)$$

$$Iкр = 220/((0{,}562/3) + \sqrt{(0{,}0896 + 0{,}1586)^2}) = 418{,}18 \text{ А}$$

Визначення максимальноі напруги на корпусі обладнання відносно землі при замиканні фази на корпус.

$$Uктах = Iкр * Zн \quad (8.15)$$

$$Uкmax = 418,18 * 0,1586 = 66,32 В$$

Z_H − повний опір нульового провідника.

Умова не виконується. Необхіно збільшити перерізи Sn 1 та Sn 2 до Sф1 та Sф2 і зробити перерахунок:

$$Rn = 0,028 * (70/70) + 0,028 * (22/10) = 0,0896 Ом,$$

$$Iкр = 220/((0,562/3) + \sqrt{(0,0896 + 0,0896)^2}) = 600,21 А,$$

$$Uкmax = 600,21 * 0,0896 = 53,77 В.$$

Умова не виконується, необхідно або замінити запобіжник з плавкою вставкою на автоматичний вимикач із струмовим реле, що дає можливість зменшити час замикання на копус і підвищити допустиму напругу на корпусі або застосувати повторне заземлення нульового захисного провідника. Повторне заземлення нульового захисного провідника:

$$Rn=(Uдоп * Ro)/((Iкр * Zн)−Uдоп) \quad (8.15)$$

$$Rn= 36 * 4/((600,21 * 0,0896) - 36) = 8,09 Ом.$$

8.4 Висновки

Існує багато факторів, які можуть вплинути на роботу розробника, через некомфортні або навіть небезпечні умови праці, які знаходяться в приміщенні. Серед основних причин виділяється: недостатній рівень світла, гучний шум, високий рівень навантаження, умови мікроклімату. Під час дослідження теми, були переглянуті можливі шкідливі та небезпечні ситуації, які можуть виникнути на робочому місці та способи їх уникнення та ліквідації.

В результаті можна зробити висновки, які умови повинні бути для продуктивної роботи працівника, як повинно бути організоване приміщення і його робоче місце. За допомогою проведених обчислень, можна становити необхідні для комфортної роботи умови.

9 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання магістерської роботи, призначено для порівняння та виявлення кращих технологій відповідно до поставлених сучасних вимог у галуз веб-розробки, а саме застосування різних типів БД при розробці додатків.

У магістерській роботі представлені дані з яких можна зробити висновки, які технології сьогодні використовуються, у яких сферах, та для його функціоналу призначені бази даних різних типів. Які можливості та для яких проектів краще використовувати класичні реляційні а коли ООБД.

Для отримання результатів були проведені наступні дослідження:

– Було створено односторінковий додаток засобами AngularJS та NodeJS.

– Було розробленно та протестовано функціонал REST API та веб-сокети за допомогою AngularJS.

– Було створенно сервер засобами NestJS, з підтримкою бази даних MongoDB, Redis, MySQL,VelocityDB.

– Було створено шаблони, які можна надалі використовувати при створенні нових додатків.

Розроблені під час виконання магістерської роботи додатки дозволяють чітко зрозуміти для яких проектів краще підходять реляційні а коли NOSQL бази даних, які їх переваги та недоліки.

Розроблені додатки підтримують функціонал, який використовується на більшості сайтів в Інтернеті, серед основних можливостей це додавання медіа файлів, можливість надсилання e-mail повідомлень, використання сокетів для спілкування в чаті, авторизація, динамічний інтерфейс. За допомогою фреймворків Angular була продемонстрована технологія односторінкових додатків, яка надала можливість розробити швидкий та зручний для сприйняття інтерфейс.

При розробці використовувалися мови JavaScript, та TypeScript що дало можливість порівняти на скільки сильно відрізняються фреймворки Angular та AngularJS та їх основні концепції. З цього можна зробити висновки скільки займає час розробки, надійність та можливість подальшого підтримання продукту.

Програма призначена для використання у браузерах для перегляду сайта, та на Linux для завантаження сервера. В роботі були надані всі необхідні дані для розуміння роботи системи та її запуску.

В результаті розроблені додатки відповідають поставленим вимогам технічного завдання, та мають можливість на подальше вдосконалення. Створені додатки можуть бути впровадженні у виробництво.

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.

Розроблена програма має реальний економічний ефект від її впровадження у виробництво у сумі 3993 грн. Враховуючи вартість розробки та необхідне обладнання, строк окуплення становить 0,22 роки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Бахрушин В. Є. Методи аналізу даних: Навч. посібник / В. Є. Бахрушин. – Запоріжжя: КПУ, 2011. – 268 с
2. Шумейко А. А. Интеллектуальный анализ данных (Введение в Data Mining) / А. А. Шумейко, С. Л. Сотник. – Днепропетровск: Белая Е. А., 2015. – 212 с.
3. http://www.nbuv.gov.ua/eb/ep.html - Національна бібліотека України імені В.І.Вернадського

4. Node.js v14.0.0 Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://nodejs.org/api/

5. М. Кантелон , М. Хартер, Т. Головайчук, Н. Райлих // “Node.js в действии”.

6. Янг А., Мек Б., Кантелон М. // “Node.js в действии. 2-е издание”.

7. John Resig, Bear Bibeault, Josip Maras // “Secrets of theJavaScript Ninja”.

8. Express [Електронний ресурс] – Режим доступу до ресурсу: http://expressjs.com/

9. Фреймворк AngularJS [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/AngularJS

10. AngularJS [Електронний ресурс] – Режим доступу до ресурсу: https://angularjs.org/

11. Bootstrap [Електронний ресурс] – Режим доступу до ресурсу: https://getbootstrap.com/

12. React.js [Електронний ресурс] – Режим доступу до ресурсу: https://https://uk.reactjs.org/

13. AngularJS MVC [Електронний ресурс] – Режим доступу до ресурсу: https://www.tutorialspoint.com/angularjs/angularjs_mvc_architecture.htm

14. Vue.js [Електронний ресурс] – Режим доступу до ресурсу: https://vuejs.org/

15. Angular 2 [Електронний ресурс] – Режим доступу до ресурсу: https://angular.io/

16. Односторінковий застосунок [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Односторінковий_застосунок

18. Build Node.js Apps [Електронний ресурс] – Режим доступу до ресурсу: https://code.visualstudio.com/docs/nodejs/nodejs-tutorial

19. REST [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/REST

20. What is REST [Електронний ресурс] – Режим доступу до ресурсу: https://restfulapi.net/

22.Веб приложение [Електронний ресурс] – Режим доступу до ресурсу: https://webcase.com.ua/blog/cho-takoe-web-prilozhenie-vse-vidy/

23.Мова розмітки гіпертексту [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/HTML

24. HTML [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/uk/docs/Web/HTML

25. Каскадні таблиці стилів [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/CSS

26. Стек MEAN [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/MEAN_(веброзробка)

27. MongoDB [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/MongoDB

28. Веб-технології для розробників [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/uk/docs/Web

29. CORS, XSS [Електронний ресурс] – Режим доступу до ресурсу: https://dev.to/maleta/cors-xss-and-csrf-with-examples-in-10-minutes-35k3

30. AJAX [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/AJAX

31. Fetch API [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/ru/docs/Web/API/Fetch_API

32. AngularJS Tutorial [Електронний ресурс] – Режим доступу до ресурсу: https://www.w3schools.com/angular/default.asp

33. jQuery [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/JQuery

34. Основи Web-технологій [Електронний ресурс] – Режим доступу до ресурсу: https://pidruchniki.com/1243020547796/informatika/web-tehnologiyi_pidpriyemstvah

35. npm [Електронний ресурс] – Режим доступу до ресурсу: https://www.npmjs.com/

36. Install MongoDB [Електронний ресурс] – Режим доступу до ресурсу: https://docs.mongodb.com/manual/administration/install-on-linux/

37. Как установить Node.js [Електронний ресурс] – Режим доступу до ресурсу: https://www.digitalocean.com/community/tutorials/node-js-ubuntu-18-04-ru

38. Linux [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Linux

39. npm-audit [Електронний ресурс] – Режим доступу до ресурсу: https://docs.npmjs.com/cli/audit

40. TypeScript [Електронний ресурс] – Режим доступу до ресурсу: https://www.typescriptlang.org/

41. SQL [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/SQL

42. MongoDB Compass [Електронний ресурс] – Режим доступу до ресурсу: https://www.mongodb.com/products/compass

43. Postman [Електронний ресурс] – Режим доступу до ресурсу: https://www.postman.com/

44. SQL [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/SQL

45. SPA (Single-page application) [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Glossary/SPA

46. Охорона праці [Електронний ресурс]: реферат – Режим доступу до ресурсу: https://revolution.allbest.ru/life/00468031_0.html

47. Природне і штучне освітлення ДБН В.2.5-28:2018: державні будівельні норми України [Електронний ресурс] / Ю. Громадський, С. Облакевич, М.Громадський, Г. Фаренюк, Є. Фаренюк, О. Підгорний, О. Сергейчук, Є.Рейцен, В. Єгорченков, Л. Коваль, Д. Радомцев, В. Злоба, Н. Кучеренко, Г.Кожушко, О. Гончар, О. Козенко, Б. Шабашкевіч, Ю. Добровольський, В.Акіменко, С. Гозак, А. Яригін, В. Назаренко, В. Мартиросова, В. Сорокін, Є.Пугачов - Київ 2018 - Режим доступу до ресурсу: https://ledeffect.com.ua/images/__branding/dbn2018.pdf

48. Розрахунок світлодіодного освітлення кімнати [Електронний ресурс] – Режим доступу до ресурсу: https://luxled.biz.ua/rozrahynok-svitlodoidnogo-osvitlennja-kimnatu-v-kvarturi-abo-bydunky

49. Охорона праці, охорона праці та безпека в надзвичайних ситуаціях : метод. вказ. до викон. розділів у дипломних роботах / [укл. В.М. Челябієва, О.Л. Гуменюк] - Чернігів ЧДТУ 2013 - [Електронний ресурс] – Режим доступу до ресурсу: http://ir.stu.cn.ua/bitstream/handle/123456789/12461/Охорона праці та безпека. в надзв. ситуац;метод.вказ..pdf?sequence=1&isAllowed=y

50. Освітленість робочих місць [Електронний ресурс] – Режим доступу до ресурсу:https://ua-referat.com/Освітленість_робочих_місць_сучасні_підходи_до_вимірів_і_оцінки

51. Температурний режим праці [Електронний ресурс] – Режим доступу до ресурсу: http://poltava.medprof.org.ua/poltava/zakhist-trudovikh-ta-socialno-ekonomichnikh-prav-pracivnikiv-galuzi/pravova-dopomoga/temperaturnii-rezhim-praci-jakim-vin-maje-buti/

52. Шум. Методи захисту від його дії : метод. вказ. до лабораторної роботи / [укл. В. І. Шмирко, С. М. Журавель] — Запоріжжя: ЗНТУ, 2014. - 14 с.

[Електронний ресурс] – Режим доступу до ресурсу: https://zp.edu.ua/sites/default/files/konf/oop_shum-2014.pdf

53. Системи протипожежного захисту ДБН В.2.5-56:2014 / Б. Платкевич, В. Носач, В. Федюк, В. Мусійчук, В. Євстіфєєв, Г. Дубінський, В. Сокол, А.Бушиленко, В. Дунюшкін, Р. Уханський, С. Пономарьов, В. Приймаченко, А.Приймаченко, С. Пітайчук, Н. Морозова, І. Колосов, О. Лагода, П. Мізін, В.Савченко, М. Федорович, П. Шаповалов, Л. Фесенко — Київ 2015 — [Електронний ресурс] – Режим доступу до ресурсу: http://kbu.org.ua/assets/app/documents/dbn2/98.1. ДБН В.2.5-56~2014. Системи протипожежного захисту.pdf

54. Охорона праці. Ч. 2. Занулення : метод. вказ. до викон. розрахунків з викор. персон. ЕОМ IBM−сумісного типу / [укл. О. В. Оришака, Є. К. Солових, В. О. Оришака, А. Е. Солових, С. Е. Катеринич] ; Центральноукраїн. нац. техн. ун-т. - 2−ге вид., перероб. та доп. - Кропивницький : ЦНТУ, 2019. - 27 с.[Електронний ресурс] – Режим доступу: http://dspace.kntu.kr.ua/jspui/handle/123456789/8769

55. The Top JavaScript Frameworks [Електронний ресурс] – Режим доступу до ресурсу: https://www.freecodecamp.org/news/complete-guide-for-front-end-developers-javascript-frameworks-2019/

56. JavaScript Frameworks 2020 [Електронний ресурс] – Режим доступу до ресурсу: https://www.npmjs.com/package/migrate-mongo

57. Web Technology [Електронний ресурс] – Режим доступу до ресурсу: https://www.geeksforgeeks.org/web-technology

Додаток А
(обов'язковий)

Технічне завдання

Зміст

1 Найменування та область застосування

Це технічне завдання розповсюджується на розробку програмного забезпечення системи застосування об’єктно-орієнтованих баз даних в ІС з розробкою додатку.

2 Підстава для розробки

Підставою для розробки служить завдання на магістерську роботу, видане на кафедрі кібербезпеки та програмного забезпечення (нак. № 20-13 від 17.08.2022 року).

3 Мета та призначення розробки

Метою магістерської роботи є розробка програмного забезпечення для реалізація застосування об’єктно-орієнтованих баз даних в ІС

4 Джерела розробки

Джерелом цієї магістерської роботи є відносна до теми література і існуючі аналоги.

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:

– Розробку додатку;

– систему підключення та тестування баз даних ;

– цілісність даних у процесі роботи та при зберіганні;

– простий, інтуїтивно зрозумілий інтерфейс.

5.3 Вимоги до функціональних характеристик

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

5.4 Вимоги до архітектури

Компонент, що розробляється повинен використовувати системні засоби та апаратні 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 процедур, функцій, методів і форм, визначених технічною документацією на середовище розробки.

5.6 Умови експлуатації

Робочі місця користувачів ПЗ повинні задовольняти наступним умовам експлуатації:

– температура повітря: 19-20 град. по Цельсію;

– відносна вологість повітря до 80%;

– атмосферний тиск 107 кПа.

5.7 Вимоги до складу та параметрів технічних засобів

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

5.8 Вимоги до інформаційної і програмної сумісності

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

5.8.1 Обладнання

Комп'ютер Intel® Celeron/8 Mb/1.2 Gb/SVGA 14" 1Mb або сумісні з ним.

5.8.2 Мова програмування

Середовище PHP та C++.

Бази даних SQL та NOSQL

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, схем та опису алгоритму, 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 та техніки безпеки в магістерській роботі повинен бути розглянутий аналіз умов праці програміста та розрахунок штучного захисного заземлення.

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 – 2 аркуша.
– Показники економічної ефективності – 1 аркуш.
– Пояснювальна записка – 133 аркушів.

10 Етапи розробки

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

10.2 Проведення досліджень або експериментальних робіт для уточнення основних положень бакалаврської дипломної 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 Оформлення пояснювальної записки і виконання робіт по графічній частині.

9 Порядок контролю та приймання

11.1 Подання бакалаврської дипломної роботи на попередній захист 10.12.2022 р.

1.2 Подання магістерської роботи на захист 10.12.2022 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
_____ Босько В.В.

Дослідження та програмна реалізація застосування об’єктно-орієнтованих баз даних в ІС

Лістинг програми

Код документу 12

Носій: CD/DVD-диск

Загальна кількість аркушів: 28

Літера: РП

Кропивницький – 2022 року

Файл основного файлу програми для завантаження AngularJS

```
<!DOCTYPE html>
<html ng-app="app">
<head>
    <meta charset="utf-8" />
    <title>AngularJS</title>
    <link rel="stylesheet"
href="//netdna.bootstrapcdn.com/bootstrap/3.1.1/css/bootstrap.min.css" />
    <link href="app-content/app.css" rel="stylesheet" />
</head>
<body>
    <div class="jumbotron2">
        <div class="container">
            <div class="col-sm-8 col-sm-offset-2">
                <div ng-class="{ 'alert': flash, 'alert-success':
flash.type === 'success', 'alert-danger': flash.type === 'error' }" ng-
if="flash" ng-bind="flash.message"></div>
                <div ng-view></div>
            </div>
        </div>
    </div>

    <script src="//code.jquery.com/jquery-3.1.1.min.js"></script>
    <script src="//code.angularjs.org/1.6.0/angular.min.js"></script>
    <script src="//code.angularjs.org/1.6.0/angular-
route.min.js"></script>
    <script src="//code.angularjs.org/1.6.0/angular-
cookies.min.js"></script>

    <script src="app.js"></script>
    <script src="app-services/authentication.service.js"></script>
    <script src="app-services/flash.service.js"></script>

    <script src="app-services/user.service.local-storage.js"></script>

    <script src="home/home.controller.js"></script>
    <script src="login/login.controller.js"></script>
    <script src="register/register.controller.js"></script>
    <script src="sidebar/sidebar.controller.js"></script>
    <script src="chat/chat.controller.js"></script>
    <script src="technologies/technologies.controller.js"></script>
    <script src="company/company.controller.js"></script>
    <script src="settings/settings.controller.js"></script>
</body>
</html>
```

Файл який відповідає за маршрутизацію додатка на AngularJS

```
(function () {
    'use strict';

    angular
        .module('app', ['ngRoute', 'ngCookies'])
        .config(config)
        .run(run);

    config.$inject = ['$routeProvider', '$locationProvider'];
    function config($routeProvider, $locationProvider) {
        $routeProvider
            .when('/', {
                controller: 'HomeController',
                templateUrl: 'home/home.view.html',
                controllerAs: 'vm'
            })
            .when('/login', {
                controller: 'LoginController',
                templateUrl: 'login/login.view.html',
                controllerAs: 'vm'
            })
            .when('/register', {
                controller: 'RegisterController',
                templateUrl: 'register/register.view.html',
                controllerAs: 'vm'
            })
            .when('/sidebar', {
                controller: 'RegisterController',
                templateUrl: 'sidebar/sidebar.view.html',
                controllerAs: 'vm'
            })
            .when('/technologies', {
                controller: 'TechnologiesController',
                templateUrl: 'technologies/technologies.view.html',
                controllerAs: 'vm'
            })
            .when('/settings', {
                controller: 'SettingsController',
                templateUrl: 'settings/settings.view.html',
                controllerAs: 'vm'
            })
            .when('/company', {
                controller: 'CompanyController',
                templateUrl: 'company/company.view.html',
                controllerAs: 'vm'
            })
            .when('/chat', {
                controller: 'ChatController',
                templateUrl: 'chat/chat.view.html',
                controllerAs: 'vm'
            })
            .otherwise({ redirectTo: '/login' });
    }

    run.$inject = ['$rootScope', '$location', '$cookies', '$http'];
    function run($rootScope, $location, $cookies, $http) {
        $rootScope.globals = $cookies.getObject('globals') || {};
        if ($rootScope.globals.currentUser) {
            $http.defaults.headers.common['Authorization'] = 'Basic ' +
$rootScope.globals.currentUser.authdata;
        }
    }

})();
```

Файл який відповідає логіку реєстрації додатка на AngularJS

```
(function () {
    'use strict';

    angular
        .module('app')
        .controller('RegisterController', RegisterController);

    RegisterController.$inject = ['UserService', '$location',
'$rootScope', 'FlashService'];
    function RegisterController(UserService, $location, $rootScope,
FlashService) {
        var vm = this;

        vm.register = register;

        function register() {
            vm.dataLoading = true;
            UserService.Create(vm.user)
                .then(function (response) {
                    if (response.success) {
                        FlashService.Success('Registration successful',
true);
                        $location.path('/login');
                    } else {
                        FlashService.Error(response.message);
                        vm.dataLoading = false;
                    }
                });
        }
    }

})();
```

Файл який відповідає за представлення реєстрації додатка на AngularJS

```
      <div class="col-md-6 col-md-offset-3">
          <h2>Register</h2>
          <form name="form" ng-submit="vm.register()" role="form">
              <div class="form-group" ng-class="{ 'has-error':
form.firstName.$dirty && form.firstName.$error.required }">
                  <label for="username">First name</label>
                  <input type="text" name="firstName" id="firstName"
class="form-control" ng-model="vm.user.firstName" required />
                  <span ng-show="form.firstName.$dirty &&
form.firstName.$error.required" class="help-block">First name is required</span>
              </div>
              <div class="form-group" ng-class="{ 'has-error':
form.lastName.$dirty && form.lastName.$error.required }">
                  <label for="username">Last name</label>
                  <input type="text" name="lastName" id="Text1" class="form-
control" ng-model="vm.user.lastName" required />
                  <span ng-show="form.lastName.$dirty &&
form.lastName.$error.required" class="help-block">Last name is required</span>
              </div>
              <div class="form-group" ng-class="{ 'has-error':
form.username.$dirty && form.username.$error.required }">
                  <label for="username">Username</label>
                  <input type="text" name="username" id="username" class="form-
control" ng-model="vm.user.username" required />
                  <span ng-show="form.username.$dirty &&
form.username.$error.required" class="help-block">Username is required</span>
              </div>
              <div class="form-group" ng-class="{ 'has-error':
form.password.$dirty && form.password.$error.required }">
                  <label for="password">Password</label>
                  <input type="password" name="password" id="password"
class="form-control" ng-model="vm.user.password" required />
                  <span ng-show="form.password.$dirty &&
form.password.$error.required" class="help-block">Password is required</span>
              </div>
              <div class="form-actions">
                  <button type="submit" ng-disabled="form.$invalid ||
vm.dataLoading" class="btn btn-primary">Register</button>
                  <img ng-if="vm.dataLoading"
src="
05FVFNDQVBFMi4wAwEAAAAh/hpDcmVhdGVkIHdpdGggYWpheGxvYWQuaW5mbwAh+QQJCgAAACwAAAAAE
AAQAAADMwi63P4wyklrE2MIOggZnAdOmGYJRbExwroUmcG2LmDEwnHQLVsYOd2mBzkYDAdKa+dIAAAh+
QQJCgAAACwAAAAAEAAQAAADNAi63P5OjCEgG4QMu7DmikRxQlFUYDEZIGBMRVsaqHwctXXf7WEYB4Ag1
xjihkMZsiUkKhIAIfkECQoAAAAsAAAAABAAEAAAAzYIujIjK8pByJDMlFYvBoVjHA70GU7xSUJhmKtwH
PAKzLO9HMaoKwJZ7Rf8AYPDDzKpZBqfvwQAIfkECQoAAAAsAAAAABAAEAAAAzMIumIlK8oyhpHsnFZfh
YumCYUhDAQxRIdhHBGqRoKw0R8DYlJd8z0fMDgsGo/IpHI5TAAAIfkECQoAAAAsAAAAABAAEAAAAzIIu
nInK0rnZBTwGPNMgQwmdsNgXGJUlIWEuR5oWUIpz8pAEAMe6TwfwyYsGo/IpFKSAAAh+QQJCgAAACwAA
AAAEAAQAAADMwi6IMKQORfjdOe82p4wGccc4CEuQradylesojEMBgsUc2G7sDX3lQGBMLAJibufbSlKA
AAh+QQJCgAAACwAAAAAEAAQAAADMgi63P7wCRHZnFVdmgHu2nFwlWCI3WGc3TSWhUFGxTAUkGCbtgENB
MJAEJsxgMLWzpEAACH5BAkKAAAALAAAAAAQABAAAAMyCLrc/jDKSatlQtScKdceCAjDII7HcQ4EMTCpy
rCuUBjCYRgHVtqlAiB1YhiCnlsRkAAAOwAAAAAAAAAAA==" />
                  <a href="#!/login" class="btn btn-link">Cancel</a>
              </div>
          </form>
      </div>
```

Файл який відповідає за представлення логіна додатка на AngularJS

```
      <div class="col-md-6 col-md-offset-3">
          <h2>Login</h2>
          <form name="form" ng-submit="vm.login()" role="form">
              <div class="form-group" ng-class="{ 'has-error':
form.username.$dirty && form.username.$error.required }">
                  <label for="username">Username</label>
                  <input type="text" name="username" id="username" class="form-
control" ng-model="vm.username" required />
                  <span ng-show="form.username.$dirty &&
form.username.$error.required" class="help-block">Username is required</span>
              </div>
              <div class="form-group" ng-class="{ 'has-error':
form.password.$dirty && form.password.$error.required }">
                  <label for="password">Password</label>
                  <input type="password" name="password" id="password"
class="form-control" ng-model="vm.password" required />
                  <span ng-show="form.password.$dirty &&
form.password.$error.required" class="help-block">Password is required</span>
              </div>
              <div class="form-actions">
                  <button type="submit" ng-disabled="form.$invalid ||
vm.dataLoading" class="btn btn-primary">Login</button>
                  <img ng-if="vm.dataLoading"
src="
05FVFNDQVBFMi4wAwEAAAAh/hpDcmVhdGVkIHdpdGggYWpheGxvYWQuaW5mbwAh+QQJCgAAACwAAAAAE
AAQAAADMwi63P4wyklrE2MIOggZnAdOmGYJRbExwroUmcG2LmDEwnHQLVsYOd2mBzkYDAdKa+dIAAAh+
QQJCgAAACwAAAAAEAAQAAADNAi63P5OjCEgG4QMu7DmikRxQlFUYDEZIGBMRVsaqHwctXXf7WEYB4Ag1
xjihkMZsiUkKhIAIfkECQoAAAAsAAAAABAAEAAAAzYIujIjK8pByJDMlFYvBoVjHA70GU7xSUJhmKtwH
PAKzLO9HMaoKwJZ7Rf8AYPDDzKpZBqfvwQAIfkECQoAAAAsAAAAABAAEAAAAzMIumIlK8oyhpHsnFZfh
YumCYUhDAQxRIdhHBGqRoKw0R8DYlJd8z0fMDgsGo/IpHI5TAAAIfkECQoAAAAsAAAAABAAEAAAAzIIu
nInK0rnZBTwGPNMgQwmdsNgXGJUlIWEuR5oWUIpz8pAEAMe6TwfwyYsGo/IpFKSAAAh+QQJCgAAACwAA
AAAEAAQAAADMwi6IMKQORfjdOe82p4wGccc4CEuQradylesojEMBgsUc2G7sDX3lQGBMLAJibufbSlKA
AAh+QQJCgAAACwAAAAAEAAQAAADMgi63P7wCRHZnFVdmgHu2nFwlWCI3WGc3TSWhUFGxTAUkGCbtgENB
MJAEJsxgMLWzpEAACH5BAkKAAAALAAAAAAQABAAAAMyCLrc/jDKSatlQtScKdceCAjDII7HcQ4EMTCpy
rCuUBjCYRgHVtqlAiB1YhiCnlsRkAAAOwAAAAAAAAAAA==" />
                  <a href="#!/register" class="btn btn-link">Register</a>
              </div>
          </form>
      </div>

      <div ng-controller="SidebarController">
          <div ng-class="'test'">
              {{phone.name}}
          </div>
      </div>
```

Файл який відповідає за логіку логіна додатка на AngularJS

```
(function () {
    'use strict';

    angular
        .module('app')
        .controller('LoginController', LoginController);

    LoginController.$inject = ['$location', 'AuthenticationService',
'FlashService'];
    function LoginController($location, AuthenticationService,
FlashService) {
        var vm = this;

        vm.login = login;

        (function initController() {
            // reset login status
            AuthenticationService.ClearCredentials();
        })();

        function login() {
            vm.dataLoading = true;
            AuthenticationService.Login(vm.username, vm.password, function
(response) {
                if (response.success) {
                    AuthenticationService.SetCredentials(vm.username,
vm.password);
                    $location.path('/');
                } else {
                    FlashService.Error(response.message);
                    vm.dataLoading = false;
                }
            });
        };
    }

})();
```

Файл для завантаження сервера main.ts

```
import 'module-alias/register';

import { NestFactory } from '@nestjs/core';
import { ValidationPipe, ValidationError } from '@nestjs/common';
import { SwaggerModule, DocumentBuilder } from '@nestjs/swagger';
import ValidationExceptions from './exceptions/validation.exceptions';

import AppModule from './routes/app/app.module';

import AllExceptionsFilter from './filters/all-exceptions.filter';

async function bootstrap() {
  const app = await NestFactory.create(AppModule);

  app.useGlobalPipes(new ValidationPipe({
    exceptionFactory: (errors: ValidationError[]) => new
ValidationExceptions(errors),
  }));
  app.useGlobalFilters(new AllExceptionsFilter());

  const port = process.env.SERVER_PORT || 3000;

  const options = new DocumentBuilder()
    .setTitle('Api v1')
    .setDescription('The boilerplate API for nestjs devs')
    .setVersion('1.0')
    .addBearerAuth({ in: 'header', type: 'http' })
    .build();
  const document = SwaggerModule.createDocument(app, options);

  SwaggerModule.setup('api', app, document);

  await app.listen(port, async () => {
    console.log(`The server is running on ${port} port:
http://localhost:${port}/api`);
  });
}
bootstrap();
```

Файл контролер для реєстрації та длгіну на сервері

```
import {
  Body,
  Controller,
  HttpCode,
  Get,
  Post,
  Delete,
  Param,
  Request,
  UnauthorizedException,
  UseGuards,
  NotFoundException,
  ForbiddenException,
  HttpStatus,
  UseInterceptors,
} from '@nestjs/common';
import {
  ApiTags,
  ApiBody,
  ApiOkResponse,
  ApiInternalServerErrorResponse,
  ApiUnauthorizedResponse,
  ApiBearerAuth,
  ApiNotFoundResponse,
  ApiBadRequestResponse,
  ApiConflictResponse,
  ApiNoContentResponse,
  ApiExtraModels,
  getSchemaPath,
} from '@nestjs/swagger';
import { JwtService } from '@nestjs/jwt';
import { Request as ExpressRequest } from 'express';
import { MailerService } from '@nestjs-modules/mailer';

import UsersService from '@v1/users/users.service';
import JwtAccessGuard from '@guards/jwt-access.guard';
import RolesGuard from '@guards/roles.guard';
import { User } from '@v1/users/schemas/users.schema';
import WrapResponseInterceptor from '@interceptors/wrap-
response.interceptor';
import AuthBearer from '@decorators/auth-bearer.decorator';
import { Roles, RolesEnum } from '@decorators/roles.decorator';
import authConstants from '@v1/auth/auth-constants';
import { DecodedUser } from './interfaces/decoded-user.interface';
import LocalAuthGuard from './guards/local-auth.guard';
import AuthService from './auth.service';
import RefreshTokenDto from './dto/refresh-token.dto';
import SignInDto from './dto/sign-in.dto';
import SignUpDto from './dto/sign-up.dto';
import JwtTokensDto from './dto/jwt-tokens.dto';
import UsersEntity from '@v1/users/entity/user.entity';

@ApiTags('Auth')
@UseInterceptors(WrapResponseInterceptor)
@ApiExtraModels(JwtTokensDto)
@Controller()
export default class AuthController {
  constructor(
    private readonly authService: AuthService,
    private readonly jwtService: JwtService,
    private readonly usersService: UsersService,
    private readonly mailerService: MailerService,
  ) {}
```

```
@ApiBody({ type: SignInDto })
@ApiOkResponse({
  schema: {
    type: 'object',
    properties: {
      data: {
        $ref: getSchemaPath(JwtTokensDto),
      },
    },
  },
  description: 'Returns jwt tokens',
})
@ApiBadRequestResponse({
  schema: {
    type: 'object',
    example: {
      message: [
        {
          target: {
            email: 'string',
            password: 'string',
          },
          value: 'string',
          property: 'string',
          children: [],
          constraints: {},
        },
      ],
      error: 'Bad Request',
    },
  },
  description: '400. ValidationException',
})
@ApiInternalServerErrorResponse({
  schema: {
    type: 'object',
    example: {
      message: 'string',
      details: {},
    },
  },
  description: '500. InternalServerError',
})
@ApiBearerAuth()
@HttpCode(HttpStatus.OK)
@UseGuards(LocalAuthGuard)
@Post('sign-in')
async signIn(@Request() req: ExpressRequest): Promise<JwtTokensDto> {
  const user = req.user as User;

  return this.authService.login(user);
}

@ApiBody({ type: SignUpDto })
@ApiOkResponse({
  description: '201, Success',
})
@ApiBadRequestResponse({
  schema: {
    type: 'object',
    example: {
      message: [
        {
          target: {
            email: 'string',
```

```
                password: 'string',
              },
              value: 'string',
              property: 'string',
              children: [],
              constraints: {},
            },
          ],
          error: 'Bad Request',
        },
      },
      description: '400. ValidationException',
    })
    @ApiConflictResponse({
      schema: {
        type: 'object',
        example: {
          message: 'string',
        },
      },
      description: '409. ConflictResponse',
    })
    @ApiInternalServerErrorResponse({
      schema: {
        type: 'object',
        example: {
          message: 'string',
          details: {},
        },
      },
      description: '500. InternalServerError',
    })
    @HttpCode(HttpStatus.CREATED)
    @Post('sign-up')
    async signUp(@Body() user: SignUpDto): Promise<any> {
      const { _id, email } = await this.usersService.create(user) as
UsersEntity;

      const token = this.authService.createVerifyToken(_id);

      await this.mailerService.sendMail({
        to: email,
        from: process.env.MAILER_FROM_EMAIL,
        subject: authConstants.mailer.verifyEmail.subject,
        template: `${process.cwd()}/src/templates/verify-password`,
        context: {
          token,
          email,
          host: process.env.SERVER_HOST,
        },
      });

      return { message: 'Success! please verify your email' };
    }

    @ApiOkResponse({
      schema: {
        type: 'object',
        properties: {
          data: {
            $ref: getSchemaPath(JwtTokensDto),
          },
        },
      },
      description: '200, returns new jwt tokens',
    })
```

```
        @ApiUnauthorizedResponse({
          schema: {
            type: 'object',
            example: {
              message: 'string',
            },
          },
          description: '401. Token has been expired',
        })
        @ApiInternalServerErrorResponse({
          schema: {
            type: 'object',
            example: {
              message: 'string',
              details: {},
            },
          },
          description: '500. InternalServerError ',
        })
        @ApiBearerAuth()
        @Post('refresh-token')
        async refreshToken(
          @Body() refreshTokenDto: RefreshTokenDto,
        ): Promise<JwtTokensDto | never>  {
          const decodedUser = this.jwtService.decode(
            refreshTokenDto.refreshToken,
          ) as DecodedUser;

          if (!decodedUser) {
            throw new ForbiddenException('Incorrect token');
          }

          const oldRefreshToken:
            | string
            | null = await
this.authService.getRefreshTokenByEmail(decodedUser.email);

          // if the old refresh token is not equal to request refresh token then
this user is unauthorized
          if (!oldRefreshToken || oldRefreshToken !==
refreshTokenDto.refreshToken) {
            throw new UnauthorizedException(
              'Authentication credentials were missing or incorrect',
            );
          }

          const payload = {
            _id: decodedUser._id,
            email: decodedUser.email,
            role: decodedUser.role,
          };

          return this.authService.login(payload);
        }

        @ApiNoContentResponse({
          description: 'No content. 204',
        })
        @ApiNotFoundResponse({
          schema: {
            type: 'object',
            example: {
              message: 'string',
              error: 'Not Found',
            },
          },
```

```
          description: 'User was not found',
        })
        @HttpCode(HttpStatus.NO_CONTENT)
        @Get('verify/:token')
        async verifyUser(@Param('token') token: string): Promise<User | null> {
          const { id } = await this.authService.verifyEmailVerToken(
            token,
            authConstants.jwt.secrets.accessToken,
          );
          const foundUser = await this.usersService.getUnverifiedUserById(id) as
UsersEntity;

          if (!foundUser) {
            throw new NotFoundException('The user does not exist');
          }

          return this.usersService.update(foundUser._id, { verified: true });
        }

        @ApiNoContentResponse({
          description: 'no content',
        })
        @ApiUnauthorizedResponse({
          schema: {
            type: 'object',
            example: {
              message: 'string',
            },
          },
          description: 'Token has been expired',
        })
        @ApiInternalServerErrorResponse({
          schema: {
            type: 'object',
            example: {
              message: 'string',
              details: {},
            },
          },
          description: 'InternalServerError',
        })
        @ApiBearerAuth()
        @UseGuards(JwtAccessGuard)
        @Delete('logout/:token')
        @HttpCode(HttpStatus.NO_CONTENT)
        async logout(@Param('token') token: string): Promise<{} | never> {
          const decodedUser: DecodedUser | null = await
this.authService.verifyToken(
            token,
            authConstants.jwt.secrets.accessToken,
          );

          if (!decodedUser) {
            throw new ForbiddenException('Incorrect token');
          }

          const deletedUsersCount = await this.authService.deleteTokenByEmail(
            decodedUser.email,
          );

          if (deletedUsersCount === 0) {
            throw new NotFoundException();
          }
          return {};
        }
```

```
  @ApiNoContentResponse({
    description: 'no content',
  })
  @ApiInternalServerErrorResponse({
    schema: {
      type: 'object',
      example: {
        message: 'string',
        details: {},
      },
    },
    description: '500. InternalServerError',
  })
  @ApiBearerAuth()
  @Delete('logout-all')
  @UseGuards(RolesGuard)
  @Roles(RolesEnum.admin)
  @HttpCode(HttpStatus.NO_CONTENT)
  async logoutAll(): Promise<{}> {
    return this.authService.deleteAllTokens();
  }

  @ApiOkResponse({
    type: User,
    description: '200, returns a decoded user from access token',
  })
  @ApiUnauthorizedResponse({
    schema: {
      type: 'object',
      example: {
        message: 'string',
      },
    },
    description: '403, says you Unauthorized',
  })
  @ApiInternalServerErrorResponse({
    schema: {
      type: 'object',
      example: {
        message: 'string',
        details: {},
      },
    },
    description: '500. InternalServerError',
  })
  @ApiBearerAuth()
  @UseGuards(JwtAccessGuard)
  @Get('token')
  async getUserByAccessToken(
    @AuthBearer() token: string,
  ): Promise<DecodedUser | never> {
    const decodedUser: DecodedUser | null = await
this.authService.verifyToken(
      token,
      authConstants.jwt.secrets.accessToken,
    );

    if (!decodedUser) {
      throw new ForbiddenException('Incorrect token');
    }

    const { exp, iat, ...user } = decodedUser;

    return user;
  }
}
```

Файл сервіс для реєстрації та лгіну на сервері

```
import * as bcrypt from 'bcrypt';

import { Injectable, NotFoundException } from '@nestjs/common';
import { JwtService } from '@nestjs/jwt';
import { Types } from 'mongoose';

import UsersRepository from '@v1/users/users.repository';
import { UserInterface } from '@v1/users/interfaces/user.interface';
import { DecodedUser } from './interfaces/decoded-user.interface';
import JwtTokensDto from './dto/jwt-tokens.dto';
import { LoginPayload } from './interfaces/login-payload.interface';

import authConstants from './auth-constants';
import AuthRepository from './auth.repository';
import UsersEntity from '@v1/users/entity/user.entity';

@Injectable()
export default class AuthService {
  constructor(
    private readonly jwtService: JwtService,
    private readonly usersRepository: UsersRepository,
    private readonly authRepository: AuthRepository,
  ) {}

  public async validateUser(
    email: string,
    password: string,
  ): Promise<null | UserInterface> {
    const user = await this.usersRepository.getVerifiedUserByEmail(email)
as UsersEntity;

    if (!user) {
      throw new NotFoundException('The item does not exist');
    }

    const passwordCompared = await bcrypt.compare(password,
user.password);

    if (passwordCompared) {
      return {
        _id: user._id,
    });

    await this.authRepository.addRefreshToken(
      payload.email as string,
      refreshToken,
    );

    return {
      accessToken,
      refreshToken,
    };
  }

  public getRefreshTokenByEmail(email: string): Promise<string | null> {
    return this.authRepository.getToken(email);
  }

  public deleteTokenByEmail(email: string): Promise<number> {
    return this.authRepository.removeToken(email);
  }

  public deleteAllTokens(): Promise<string> {
```

```
    return this.authRepository.removeAllTokens();
  }

  public createVerifyToken(id: Types.ObjectId): string {
    return this.jwtService.sign(
      { id },
      {
        expiresIn: authConstants.jwt.expirationTime.accessToken,
        secret: authConstants.jwt.secrets.accessToken,
      },
    );
  }

  public verifyEmailVerToken(token: string, secret: string) {
    return this.jwtService.verifyAsync(token, { secret });
  }

  public async verifyToken(
    token: string,
    secret: string,
  ): Promise<DecodedUser | null> {
    try {
      const user = (await this.jwtService.verifyAsync(token, {
        secret,
      })) as DecodedUser | null;

      return user;
    } catch (error) {
      return null;
    }
  }
}     email: user.email,
        role: user.role,
      };
    }

    return null;
  }

  public async login(data: LoginPayload): Promise<JwtTokensDto> {
    const payload: LoginPayload = {
      _id: data._id,
      email: data.email,
      role: data.role,
    };

    const accessToken = this.jwtService.sign(payload, {
      expiresIn: authConstants.jwt.expirationTime.accessToken,
      secret: authConstants.jwt.secrets.accessToken,
    });
    const refreshToken = this.jwtService.sign(payload, {
      expiresIn: authConstants.jwt.expirationTime.refreshToken,
      secret: authConstants.jwt.secrets.refreshToken,
    });

    await this.authRepository.addRefreshToken(
      payload.email as string,
      refreshToken,
    );

    return {
      accessToken,
      refreshToken,
    };
  }
```

```
  public getRefreshTokenByEmail(email: string): Promise<string | null> {
    return this.authRepository.getToken(email);
  }

  public deleteTokenByEmail(email: string): Promise<number> {
    return this.authRepository.removeToken(email);
  }

  public deleteAllTokens(): Promise<string> {
    return this.authRepository.removeAllTokens();
  }

  public createVerifyToken(id: Types.ObjectId): string {
    return this.jwtService.sign(
      { id },
      {
        expiresIn: authConstants.jwt.expirationTime.accessToken,
        secret: authConstants.jwt.secrets.accessToken,
      },
    );
  }

  public verifyEmailVerToken(token: string, secret: string) {
    return this.jwtService.verifyAsync(token, { secret });
  }

  public async verifyToken(
    token: string,
    secret: string,
  ): Promise<DecodedUser | null> {
    try {
      const user = (await this.jwtService.verifyAsync(token, {
        secret,
      })) as DecodedUser | null;

      return user;
    } catch (error) {
      return null;
    }
  }
}
```

Файл репозиторій для реєстрації та лгіну на сервері

```
import * as Redis from 'ioredis';
import { Injectable } from '@nestjs/common';

import { RedisService } from 'nestjs-redis';
import authConstants from './auth-constants';

@Injectable()
export default class AuthRepository {
  private readonly redisClient: Redis.Redis;

  constructor(private readonly redisService: RedisService) {
    this.redisClient = redisService.getClient();
  }

  public async addRefreshToken(userEmail: string, refreshToken: string):
Promise<void> {
    await this.redisClient.set(
      userEmail,
      refreshToken,
      'EX',
      authConstants.redis.expirationTime.jwt.refreshToken,
    );
  }

  public getToken(key: string): Promise<string | null> {
    return this.redisClient.get(key);
  }

  public removeToken(key: string): Promise<number> {
    return this.redisClient.del(key);
  }

  public removeAllTokens(): Promise<string> {
    return this.redisClient.flushall();
  }
}
```

Файл модуль для маршрутизації в Angular

```
      import {NO_ERRORS_SCHEMA, NgModule} from '@angular/core';
      import { Routes, RouterModule } from '@angular/router';

      import { HomeComponent } from './home';
      import { AuthGuard } from './_helpers';

      const accountModule = () => import('./account/account.module').then(x =>
x.AccountModule);
      const usersModule = () => import('./users/users.module').then(x =>
x.UsersModule);
      const profileModule = () => import('./profile/profile.module').then(x =>
x.ProfileModule);
      const documentModule = () => import('./document/document.model').then(x =>
x.DocumentModule);
      const documentChat = () => import('./chat/chat.model').then(x =>
x.ChatModule);
      const documentCompany = () => import('./company/company.model').then(x =>
x.CompanyModule);
      const documentTechnologies = () =>
import('./technologies/technologies.model').then(x => x.TechnologiesModule);
      const documentSettings = () => import('./settings/settings.model').then(x
=> x.SettingsModule);

      const routes: Routes = [
          { path: '', component: HomeComponent, canActivate: [AuthGuard] },
          { path: 'users', loadChildren: usersModule, canActivate:
[AuthGuard] },
          { path: 'account', loadChildren: accountModule },
          { path: 'profile', loadChildren: profileModule },
          { path: 'document', loadChildren: documentModule },
          { path: 'chat', loadChildren: documentChat },
          { path: 'company', loadChildren: documentCompany },
          { path: 'technologies', loadChildren: documentTechnologies },
          { path: 'settings', loadChildren: documentSettings },

          { path: '**', redirectTo: '' }
      ];

      @NgModule({
          imports: [RouterModule.forRoot(routes)],
          exports: [RouterModule],
          schemas: [NO_ERRORS_SCHEMA]
      })
      export class AppRoutingModule { }
```

Файл який відповідає за логіку логіна додатка на Angular

```
import { Component, OnInit } from '@angular/core';
import { Router, ActivatedRoute } from '@angular/router';
import { FormBuilder, FormGroup, Validators } from '@angular/forms';
import { first } from 'rxjs/operators';

import { AccountService, AlertService } from '@app/_services';

@Component({ templateUrl: 'login.component.html' })
export class LoginComponent implements OnInit {
    form: FormGroup;
    loading = false;
    submitted = false;
    returnUrl: string;

    constructor(
        private formBuilder: FormBuilder,
        private route: ActivatedRoute,
        private router: Router,
        private accountService: AccountService,
        private alertService: AlertService
    ) { }

    ngOnInit() {
        this.form = this.formBuilder.group({
            username: ['', Validators.required],
            password: ['', Validators.required]
        });

        this.returnUrl = this.route.snapshot.queryParams['returnUrl'] ||
'/';
    }

    get f() { return this.form.controls; }

    onSubmit() {
        this.submitted = true;

        this.alertService.clear();

        if (this.form.invalid) {
            return;
        }

        this.loading = true;
        this.accountService.login(this.f.username.value,
this.f.password.value)
            .pipe(first())
            .subscribe(
                data => {
                    this.router.navigate([this.returnUrl]);
                },
                error => {
                    this.alertService.error(error);
                    this.loading = false;
                });
    }
}
```

Файл який відповідає за представлення логіна додатка на Angular

```
      <div class="card">
          <h4 class="card-header">Login</h4>
          <div class="card-body">
              <form [formGroup]="form" (ngSubmit)="onSubmit()">
                  <div class="form-group">
                      <label for="username">Username</label>
                      <input type="text" formControlName="username" class="form-
control" [ngClass]="{ 'is-invalid': submitted && f.username.errors }" />
                      <div *ngIf="submitted && f.username.errors"
class="invalid-feedback">
                          <div *ngIf="f.username.errors.required">Username is
required</div>
                      </div>
                  </div>
                  <div class="form-group">
                      <label for="password">Password</label>
                      <input type="password" formControlName="password"
class="form-control" [ngClass]="{ 'is-invalid': submitted &&
f.password.errors }" />
                      <div *ngIf="submitted && f.password.errors"
class="invalid-feedback">
                          <div *ngIf="f.password.errors.required">Password is
required</div>
                      </div>
                  </div>
                  <div class="form-group">
                      <button [disabled]="loading" class="btn btn-primary">
                          <span *ngIf="loading" class="spinner-border spinner-
border-sm mr-1"></span>
                          Login
                      </button>
                      <a routerLink="../register" class="btn btn-
link">Register</a>
                  </div>
              </form>
          </div>
      </div>
```

Файл який відповідає за логіку реєстрації додатка на Angular

```
import { Component, OnInit } from '@angular/core';
import { Router, ActivatedRoute } from '@angular/router';
import { FormBuilder, FormGroup, Validators } from '@angular/forms';
import { first } from 'rxjs/operators';

import { AccountService, AlertService } from '@app/_services';

@Component({ templateUrl: 'register.component.html' })
export class RegisterComponent implements OnInit {
    form: FormGroup;
    loading = false;
    submitted = false;

    constructor(
        private formBuilder: FormBuilder,
        private route: ActivatedRoute,
        private router: Router,
        private accountService: AccountService,
        private alertService: AlertService
    ) { }

    ngOnInit() {
        this.form = this.formBuilder.group({
            firstName: ['', Validators.required],
            lastName: ['', Validators.required],
            username: ['', Validators.required],
            password: ['', [Validators.required, Validators.minLength(6)]]
        });
    }

    get f() { return this.form.controls; }

    onSubmit() {
        this.submitted = true;

        this.alertService.clear();

        if (this.form.invalid) {
            return;
        }

        this.loading = true;
        this.accountService.register(this.form.value)
            .pipe(first())
            .subscribe(
                data => {
                    this.alertService.success('Registration successful',
{ keepAfterRouteChange: true });
                    this.router.navigate(['../login'], { relativeTo:
this.route });
                },
                error => {
                    this.alertService.error(error);
                    this.loading = false;
                });
    }
}
```

Файл який відповідає за представлення реєстрації додатка на Angular

```
      <div class="card">
          <h4 class="card-header">Register</h4>
          <div class="card-body">
              <form [formGroup]="form" (ngSubmit)="onSubmit()">
                  <div class="form-group">
                      <label for="firstName">First Name</label>
                      <input type="text" formControlName="firstName"
class="form-control" [ngClass]="{ 'is-invalid': submitted &&
f.firstName.errors }" />
                      <div *ngIf="submitted && f.firstName.errors"
class="invalid-feedback">
                          <div *ngIf="f.firstName.errors.required">First Name is
required</div>
                      </div>
                  </div>
                  <div class="form-group">
                      <label for="lastName">Last Name</label>
                      <input type="text" formControlName="lastName" class="form-
control" [ngClass]="{ 'is-invalid': submitted && f.lastName.errors }" />
                      <div *ngIf="submitted && f.lastName.errors"
class="invalid-feedback">
                          <div *ngIf="f.lastName.errors.required">Last Name is
required</div>
                      </div>
                  </div>
                  <div class="form-group">
                      <label for="username">Username</label>
                      <input type="text" formControlName="username" class="form-
control" [ngClass]="{ 'is-invalid': submitted && f.username.errors }" />
                      <div *ngIf="submitted && f.username.errors"
class="invalid-feedback">
                          <div *ngIf="f.username.errors.required">Username is
required</div>
                      </div>
                  </div>
                  <div class="form-group">
                      <label for="password">Password</label>
                      <input type="password" formControlName="password"
class="form-control" [ngClass]="{ 'is-invalid': submitted &&
f.password.errors }" />
                      <div *ngIf="submitted && f.password.errors"
class="invalid-feedback">
                          <div *ngIf="f.password.errors.required">Password is
required</div>
                          <div *ngIf="f.password.errors.minlength">Password must
be at least 6 characters</div>
                      </div>
                  </div>
                  <div class="form-group">
                      <button [disabled]="loading" class="btn btn-primary">
                          <span *ngIf="loading" class="spinner-border spinner-
border-sm mr-1"></span>
                          Register
                      </button>
                      <a routerLink="../login" class="btn btn-link">Cancel</a>
                  </div>
              </form>
          </div>
      </div>
```

Файл сервіс який відповідає за юзера додатка на Angular

```
import { Injectable } from '@angular/core';
import { Router } from '@angular/router';
import { HttpClient } from '@angular/common/http';
import { BehaviorSubject, Observable } from 'rxjs';
import { map } from 'rxjs/operators';

import { environment } from '@environments/environment';
import { User } from '@app/_models';

@Injectable({ providedIn: 'root' })
export class AccountService {
    private userSubject: BehaviorSubject<User>;
    public user: Observable<User>;

    constructor(
        private router: Router,
        private http: HttpClient
    ) {
        this.userSubject = new
BehaviorSubject<User>(JSON.parse(localStorage.getItem('user')));
        this.user = this.userSubject.asObservable();
    }

    public get userValue(): User {
        return this.userSubject.value;
    }

    login(username, password) {
        return
this.http.post<User>(`${environment.apiUrl}/users/authenticate`, { username,
password })
            .pipe(map(user => {
                localStorage.setItem('user', JSON.stringify(user));
                this.userSubject.next(user);
                return user;
            }));
    }

    logout() {
        localStorage.removeItem('user');
        this.userSubject.next(null);
        this.router.navigate(['/account/login']);
    }

    register(user: User) {
        return this.http.post(`${environment.apiUrl}/users/register`,
user);
    }

    getAll() {
        return this.http.get<User[]>(`${environment.apiUrl}/users`);
    }

    getById(id: string) {
        return this.http.get<User>(`${environment.apiUrl}/users/${id}`);
    }

    update(id, params) {
        return this.http.put(`${environment.apiUrl}/users/${id}`, params)
            .pipe(map(x => {
                if (id == this.userValue.id) {
                    const user = { ...this.userValue, ...params };
                    localStorage.setItem('user', JSON.stringify(user));
```

```
                    this.userSubject.next(user);
                }
                return x;
            }));
    }

    delete(id: string) {
        return this.http.delete(`${environment.apiUrl}/users/${id}`)
            .pipe(map(x => {
                if (id == this.userValue.id) {
                    this.logout();
                }
                return x;
            }));
    }
}
```

Файл схеми юзера на сервері

```
import { Prop, Schema, SchemaFactory } from '@nestjs/mongoose';
import { Document } from 'mongoose';

import { RolesEnum } from '@decorators/roles.decorator';

@Schema()
export class User {
  @Prop({
    required: true,
    unique: true,
    type: String,
  })
  email: string = '';

  @Prop({
    required: true,
    type: String,
  })
  password: string = '';

  @Prop({
    required: true,
    type: Boolean,
  })
  verified: boolean = false;

  @Prop({
    type: RolesEnum,
    required: false,
    default: RolesEnum.user,
  })
  role: RolesEnum = RolesEnum.user;
}

export type UserDocument = User & Document;

export const UserSchema =
SchemaFactory.createForClass(User).set('versionKey', false);
```

Файл головного модуля на сервері

```
import { Module } from '@nestjs/common';
import { RedisModule } from 'nestjs-redis';
import { ConfigModule } from '@nestjs/config';
import { MongooseModule } from '@nestjs/mongoose';
import { MailerModule } from '@nestjs-modules/mailer';
import { HandlebarsAdapter } from '@nestjs-
modules/mailer/dist/adapters/handlebars.adapter';
import V1Module from '@v1/v1.module';
import AppController from './app.controller';
import AppService from './app.service';
import AppGateway from './app.gateway';
@Module({
  imports: [
    ConfigModule.forRoot({
      isGlobal: true,
    }),
    MongooseModule.forRoot(process.env.MONGODB_URL as string, {
      autoReconnect: true,
      useCreateIndex: true,
      reconnectTries: Number.MAX_VALUE,
      reconnectInterval: 1000,
      useNewUrlParser: true,
      useUnifiedTopology: true,
    }),
    RedisModule.register({
      url: process.env.REDIS_URL,
      onClientReady: async (client): Promise<void> => {
        client.on('error', console.error);
        client.on('ready', () => {
          console.log('redis is running on 6379 port');
        });
        client.on('restart', () => {
          console.log('attempt to restart the redis server');
        });
      },
      reconnectOnError: (): boolean => true,
    }),
    MailerModule.forRoot({
      transport: {
        host: process.env.MAILER_HOST,
        port: Number(process.env.MAILER_PORT),
        secure: false,
        auth: {
          user: process.env.MAILER_USERNAME,
          pass: process.env.MAILER_PASSWORD,
        },
      },
      defaults: {
        from: process.env.MAILER_FROM_EMAIL,
      },
      template: {
        adapter: new HandlebarsAdapter(),
        options: {
          strict: true,
        },
      },
    }),
    V1Module,
  ],
  controllers: [AppController],
  providers: [AppService, AppGateway],
})
export default class AppModule {}
```

Файл домашньої сторінки AngularJS

```
(function () {
    'use strict';

    angular
        .module('app')
        .controller('HomeController', HomeController);

    HomeController.$inject = ['UserService', '$rootScope'];
    function HomeController(UserService, $rootScope) {
        var vm = this;

        vm.user = null;
        vm.allUsers = [];
        vm.deleteUser = deleteUser;

        initController();

        function initController() {
            loadCurrentUser();
            loadAllUsers();
        }

        function loadCurrentUser() {

UserService.GetByUsername($rootScope.globals.currentUser.username)
                .then(function (user) {
                    vm.user = user;
                });
        }

        function loadAllUsers() {
            UserService.GetAll()
                .then(function (users) {
                    vm.allUsers = users;
                });
        }

        function deleteUser(id) {
            UserService.Delete(id)
            .then(function () {
                loadAllUsers();
            });
        }
    }

})();
```