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

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

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
за першим (бакалаврським) рівнем вищої освіти

на тему

“Програмне забезпечення системи кібербезпеки для захищеного обміну даними у мережі Інтернет”

Виконав здобувач вищої освіти
IV курсу, групи КБ-20-3СК
ОПП «Кібербезпека»
спеціальності 125 «Кібербезпека»
_____ Агапов В. В.
«____» _____ 20 р.

Керівник проекту
доктор технічних наук, професор
_____ Мелешко Є. В.
«____» _____ 20 р.
Рецензент _____
_____

м. Кропивницький

Міністерство освіти і науки України
Центральноукраїнський національний технічний університет

Факультет *Механіко-технологічний*
Кафедра *Кібербезпеки та програмного забезпечення*
Освітній ступінь *бакалавр*
Спеціальність *125 Кібербезпека*

ЗАТВЕРДЖУЮ
Завідувач кафедри
д.т.н., проф.
_____О.А.Смірнов
« __ » _____ 20__ року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТУ

Агапову Віктору Вікторовичу
(прізвище, ім'я, по батькові)

1. Тема роботи *Програмне забезпечення системи кібербезпеки для захищеного обміну даними у мережі Інтернет*

керівник роботи *Мелешко Єлизавета Владиславівна, д-р техн. наук, професор*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу №13-02 від 05.01.2023 року

2. Строк подання студентом роботи до захисту *20.05.2023 р.*

3. Мета та завдання кваліфікаційної бакалаврської роботи*: Метою розробки є програмне забезпечення системи кібербезпеки для захищеного обміну даними у мережі Інтернет*

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

1. Призначення та область використання.
2.Перегляд аналогічних існуючих систем.
3. Опис і обґрунтування проектних рішень.
4. Етапи програмування системи.
5. Впровадження системи в промислову експлуатацію.
6. Висновки

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

Структурна схема системи	*1 аркуш*
Функціональна схема системи	*1 аркуш*
Діаграма процесів	*1 аркуш*
Блок-схема алгоритму роботи додатку	*2 аркуша*

6. Дата видачі завдання « 21 » грудня 2022 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної бакалаврської роботи	Строк виконання етапів кваліфікаційної бакалаврської роботи	Примітка
1.	Аналіз існуючих систем	10.03.2023 р.	
2.	Постановка задачі, оформлення ТЗ	15.03.2023 р.	
3.	Розробка моделі компонента	20.03.2023 р.	
4.	Розробка структур даних	25.03.2023 р.	
5.	Розробка алгоритмів зв’язку та відображення	30.03.2023 р.	
6.	Програмування алгоритмів	10.04.2023 р.	
7.	Оформлення ПЗ	17.04.2023 р.	
8.	Попередній захист роботи	22.05.2023 р.	

Студент _____ _____
(підпис) (прізвище та ініціали)

Керівник роботи _____ _____
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Агапов В.В. Програмне забезпечення системи кібербезпеки для захищеного обміну даними у мережі Інтернет. 125 Кібербезпека. Центральноукраїнський національний технічний університет. Кропивницький. 2023.

У даній кваліфікаційній бакалаврській роботі розроблено програмне забезпечення, яке призначено для системи кібербезпеки захищеного обміну даними у мережі Інтернет.

Метою роботи є створення системи кібербезпеки для захищеного обміну даними у мережі Інтернет.

Результат роботи – програмна реалізація системи кібербезпеки для захищеного обміну даними у мережі Інтернет.

В процесі роботи над реалізацією системи виконано дослідження існуючих методів, алгоритмів та програмних засобів. Розроблено та реалізовано власне програмне забезпечення, здійснено опис всіх його компонентів.

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

Програма може використовуватися на ПЕОМ архітектури IBM PC з ОС Windows 10/11.

Програму розроблено на мовах програмування JavaScript та PHP.

Ключові слова: кібербезпека, захист інформації, обмін даними, комп’ютерні мережі

ABSTRACT

Ahapov V.V. Cybersecurity system software for secure data exchange on the Internet. 125 Cybersecurity. Central Ukrainian National Technical University. Kropyvnytskyi 2023

In this bachelor's qualification work, software that is designed for system for secure data exchange on the Internet was developed.

The purpose of the development is the software of the system for secure data exchange on the Internet.

The result of the work is the software implementation of the system for secure data exchange on the Internet.

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

A user-friendly interface is developed. The instructions for working with the software are provided.

The program can be used on an IBM PC running Windows 10/11.

The software is developed in the JavaScript and PHP programming languages.

Keywords: cybersecurity, information protection, data exchange, computer networks

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

Apache	–	відкритий веб-сервер Інтернет.
API	–	application programming interface, опис способів взаємодії однієї комп’ютерної програми з іншими.
CC	–	Creative Commons, різновид вільної ліцензії.
CSS	–	мова стилю веб-сторінок, яка використовується для опису їх зовнішнього вигляду.
HTML	–	мова розмітки гіпертексту для веб-сайтів.
MySQL	–	мова запитів для керування реляційними базами даних.
OSI		Open Systems Interconnection, модель стеку мережевих протоколів OSI/IOS.
PHP	–	скриптова мова програмування, призначена для генерації HTML-сторінок на стороні веб-сервера.
RSA	–	асиметричний алгоритм шифрування, абревіатура від прізвищ його авторів Rivest, Shamir, Adleman.
URL	–	Uniform Resource Locator, єдиний вказівник на ресурс – стандартизована адреса певного ресурсу в Інтернеті.
БД	–	база даних.
ОС	–	операційна система.
СУБД	–	система управління базами даних.

ВСТУП

Захист інформації у мережі Інтернет є актуальною проблемою в наш час, оскільки у сучасному світі все більше інформації зберігається в електронному вигляді і доступ до неї можуть мати тисячі людей, а також зростає кількість кібератак та кіберзлочинності. Захист інформації у комп'ютерних мережах є необхідною умовою для успішного функціонування різного приватного бізнесу, державних організацій та держави в цілому. Для забезпечення захисту важливих даних, що передаються по комп'ютерним мережам, необхідно використовувати системи кібербезпеки. Криптографічний захист даних дозволяє забезпечувати конфіденційність та цілісність інформації під час її передачі.

Програмне забезпечення для системи кібербезпеки є важливим елементом у забезпеченні інформаційної безпеки в мережі Інтернет. Такі системи повинні бути ефективними, надійними та простими у використанні. Вони можуть включати в себе різні компоненти для захисту комп'ютерної мережі, такі як брандмауери, системи виявлення вторгнень, антивірусні програми, а також криптографічні механізми для шифрування даних. Важливим під час їх розробки є забезпечення стійкості до новітніх видів кібератак, оскільки зловмисники постійно вдосконалюють свої методи. У даній кваліфікаційній роботі було вирішено розробити систему криптографічного захисту даних для захищеного обміну повідомленнями у мережі Інтернет.

Проведення захищеного обміну даними у мережі Інтернет є важливим елементом діяльності багатьох підприємств та установ. Недостатня захищеність даних може призвести до витоку конфіденційної інформації, втрати даних або порушення законодавства про захист персональних даних.

Мета й завдання дослідження. Метою роботи є програмне забезпечення системи кібербезпеки для захищеного обміну даними у мережі Інтернет.

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

– Дослідження існуючих методів захищеного захищеного обміну даними для комунікування у мережі Інтернет.

– Розробка алгоритмів системи кібербезпеки для захищеного обміну даними у мережі Інтернет.

– Програмна реалізація системи кібербезпеки для захищеного обміну даними у мережі Інтернет.

Практична цінність отриманих результатів полягає в тому, що розроблені алгоритми дозволяють успішно вирішувати задачі шифрування та передачі інформації для захищеного обміну даними у мережі Інтернет.

Отже, розробка та впровадження програмного забезпечення системи кібербезпеки для захищеного обміну даними у мережі Інтернет є актуальною задачею, яка вимагає постійного вдосконалення і розробки нових рішень та вирішувалася у цій кваліфікаційній роботі.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1 Призначення системи

Програмне забезпечення системи кібербезпеки для захищеного обміну даними у мережі Інтернет має широкий спектр застосувань у різних сферах діяльності. Однією з найважливіших є фінансова сфера, де обмін чутливими даними, такими як банківські реквізити та особиста інформація, є необхідністю. Банківські установи використовують системи кібербезпеки для захисту від кібератак та крадіжок інформації.

Іншою важливою сферою є урядові інституції та оборона. Урядові агентства обмінюються чутливою інформацією, такою як військова та дипломатична інформація, яка потребує високого рівня захисту. Системи кібербезпеки дозволяють забезпечити конфіденційність та цілісність такої інформації під час її передачі через Інтернет.

Також, системи кібербезпеки знаходять застосування в комерційній сфері, де важливий захист від крадіжок інтелектуальної власності та конфіденційної інформації, наприклад, при обміні даними між компаніями-конкурентами. Крім того, системи кібербезпеки можуть застосовуватися в освіті та науці, де важливим є захист конфіденційної інформації та обмін даними між дослідниками.

Не слід забувати, що також особиста переписка звичайних користувачів потребує інформаційної безпеки та конфіденційності з різних причин як особистих, так і законодавчих.

Усі ці розглянуті сфери потребують захищеного обміну даними в Інтернеті, і розроблюване програмне забезпечення системи кібербезпеки може відігравати важливу роль в реалізації їх захисту.

1.2 Область застосування

Програмне забезпечення системи кібербезпеки для захищеного обміну даними у мережі Інтернет може бути застосоване в різних сферах діяльності та в різних контекстах. Основні можливі області застосування розроблюваного програмного забезпечення:

– Компанії, які займаються фінансовими послугами, такими як банківські установи та фондові біржі. Система кібербезпеки для захищеного обміну даними може допомогти захистити конфіденційні фінансові дані та транзакції клієнтів, запобігти крадіжкам та шахрайству в Інтернеті.

– Установи державної влади та правоохоронні органи. Система кібербезпеки для захищеного обміну даними може допомогти захистити державну та конфіденційну інформацію від злочинних елементів та збереження даних у таємниці.

– Установи охорони здоров'я, які зберігають конфіденційну медичну інформацію про пацієнтів. Система кібербезпеки для захищеного обміну даними може допомогти захистити ці дані від несанкціонованого доступу, збереження та передачі в Інтернеті.

– Установи, що здійснюють наукові дослідження та розробки. Система кібербезпеки для захищеного обміну даними може допомогти захистити інтелектуальну власність, конфіденційні дані та інформацію про дослідження, що зберігається та передається в Інтернеті.

– Компанії, що займаються інформаційними технологіями та мережевою безпекою. Система кібербезпеки для захищеного обміну даними може бути використане як частина послуг, що надаються клієнтам.

Таким чином, програмне забезпечення системи кібербезпеки для захищеного обміну даними у мережі Інтернет має широкий спектр застосування в різних галузях діяльності – від великих корпорацій до дрібних підприємств та індивідуальних користувачів, яким треба забезпечувати свою кібербезпеку.

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

2.1 Огляд існуючих систем, технологій, архітектур, програмних рішень за профілем теми кваліфікаційної бакалаврської роботи

Широке поширення та доступність комп'ютерних систем та мереж сприяє виникненню все більшої кількості програмних засобів та технологій, які використовуються в них для обміну захищеними повідомленнями. Було проведено дослідження відповідних існуючих програмних засобів.

Vypress Chat є комерційним текстовим чатом, призначеним для використання в локальних мережах. Це програма, яка надає всі необхідні можливості для невеликих офісів і домашніх мереж, і вона легка у налаштуванні та використанні. Відмінністю Vypress Chat є те, що вона не потребує окремого сервера - кожна копія програми може одночасно виступати як сервер і клієнт, зробивши всіх учасників чату рівноправними. Після запуску Vypress Chat автоматично приєднується до загального каналу спілкування, який називається "#Main", що дозволяє користувачам бачити, хто вже приєднався до загальної розмови, надсилати повідомлення іншим користувачам (всім або вибірково), обмінюватися файлами та створювати власні канали для обговорення приватних питань.

Основними перевагами Vypress Chat є такі:

- програма не потребує виділеного сервера;
- окрім загального каналу, можна створювати персональні канали, що будуть захищені паролем;
- підтримка передачі файлів;
- фільтрація небажаних повідомлень;
- мінімалістичний та зрозумілий інтерфейс;

Потрібно відмітити і наступні недоліки програми:

- відсутність гарантії доставки повідомлення;

- відсутність голосового та відео чату;

- створює помітне навантаження на мережу за рахунок багатоадресних повідомлень, особливо при великій кількості користувачів;

Відсутність гарантії доставки повідомлення до адресата зумовлена тим, що для передачі повідомлень в програмі використовується протокол UDP, що є одним з найпростіших протоколів транспортного рівня моделі OSI. Він виконує обмін дейтаграмами без підтвердження та гарантії доставки. При використанні протоколу UDP обробка помилок і повторна передача даних має виконуватися протоколом вищого рівня.

Vypress Chat є платним програмним забезпеченням, хоч це і не можна віднести до недоліків програми, але цей факт може бути вирішальним в ринковій конкуренції з іншими безкоштовними аналогами.

MyChat є програмним забезпеченням клієнт-серверного типу, призначеним для обміну текстовими повідомленнями та файлами в локальних та глобальних мережах. Однією з особливостей цього чату є наявність єдиної деревоподібної книги контактів, що значно полегшує пошук потрібних учасників чату.

Для передачі даних програма MyChat використовує протокол SSL, який забезпечує конфіденційність обміну даними між клієнтом і сервером. Шифрування виконується за допомогою асиметричного алгоритму з використанням відкритого ключа. При використанні асиметричного шифрування використовуються два ключі, і кожен з них може бути використаний для шифрування повідомлення. Таким чином, якщо один ключ використовується для шифрування, то для розшифрування потрібно використовувати інший ключ.

SSL надає канал, що має три основні властивості:

– автентифікація – сервер завжди автентифікований, в той час як клієнт автентифікований в залежності від алгоритму;

– цілісність – обмін повідомленнями включає в себе перевірку цілісності;

– конфіденційність каналу – шифрування використовується після встановлення з'єднання і для всіх наступних повідомлень.

За рахунок широкої функціональності системи вона має дуже насичений та складний інтерфейс з користувачем, що робить роботу з нею менш ефективною та комфортною.

Проте, MyChat має також версію веб-чату для спілкування користувачів без встановлення програмного додатку, використовуючи браузер. Інтерфейс даної версії програми дуже простий і зрозумілий.

Тож відмінними характерними рисами MyChat є:

– можливість віддаленого адміністрування сервера;

– відправка відкладених повідомлень для користувачів, що в даний момент не з'єднані сервером;

– наявність зручної дошки оголошень;

– вбудовані антифлуд і антиспам системи;

– захищеність каналу передачі повідомлень;

– наявність веб-версії чату;

– підтримка плагінів і скриптів, що розширюють можливості системи.

CommuniCrypt, розроблений компанією CommuniCrypt Software, є ефективним програмним засобом для обміну повідомленнями в реальному часі в локальних мережах та Інтернеті. Одним з ключових особливостей цього програмного забезпечення є використання алгоритму шифрування з відкритим ключем RSA за схемою RSA-OAEP для обміну повідомленнями.

Ця схема шифрування передбачає використання двох функцій хешування перед процесом шифрування, що сприяє ефективним обчисленням. Це значно підвищує стійкість алгоритму шифрування. Довжина відкритого та закритого ключів може варіюватися від 338 до 7120 біт, що робить шифрування більш стійким проти злому.

Таким чином особливості CommuniCrypt наступні:

– відсутність центрального сервера, всі дані передаються безпосередньо

від вузла до вузла;

– наявність можливості передачі файлів;

– використання алгоритму шифрування RSA за схемою RSA-OAEP.

Cute Chat – це повнофункціональний веб-чат, розроблений з використанням технології ASP.NET. Щоб скористатися цим додатком, вам не потрібно встановлювати жодне додаткове програмне забезпечення. Доступ до Cute Chat здійснюється через веб-браузер. Після цього користувач може вибрати режим спілкування в чаті, наприклад, приватний розмова, кімната з паролем і т. д., і отримати доступ до сторінки, яка представляє інтерфейс чату. Загалом інтерфейс майже не відрізняється від звичайного чату.

Потрібно виділити такі можливості веб-чату Cute Chat:

– кросплатформеність;

– підтримка передачі файлів;

– відсутність необхідності встановлення системи на комп’ютер;

– простий, інтуїтивно зрозумілий інтерфейс;

– фільтрування ненормативної лексики;

– підтримка приватних кімнат та повідомлень;

– здатність обслуговувати дуже велику кількість користувачів.

Tor Chat – це простий чат, що використовує систему Tor і шифрування повідомлень для забезпечення анонімного та захищеного обміну повідомленнями. Даний чат має дуже простий інтерфейс користувача.

Можливості цього чату обмежені і включають передачу текстових повідомлень, обмін файлами, пошук користувачів за їх унікальним ідентифікатором, наявність колекції значків для переписки і зміну стану користувачів в мережі. Однак, важливо відзначити, що всі ці дії забезпечують повну анонімність та захищеність інформації. Використання мережі Tor гарантує, що ніхто не може контролювати інтернет-трафік, щоб дізнатися, з ким спілкується той чи інший користувач та обмінюється файлами, а також його географічне положення. Tor Chat реалізує з'єднання між двома Tor-клієнтами за

схемою один до одного (P2P), що означає, що трафік ніколи не покидає межі чату. Саме завдяки забезпеченню анонімності та високого рівня захищеності багато користувачів віддають перевагу цьому програмному додатку.

Tor - це система, створена для забезпечення анонімності в Інтернеті. Клієнтське програмне забезпечення Tor маршрутизує Інтернет-трафік через глобальну мережу добровільних серверів з метою приховання географічного розташування користувача. Крім того, використання Tor ускладнює відстеження Інтернет-активності як на рівні веб-сайту, так і на рівні Інтернет-провайдера, включаючи перегляд веб-сторінок, залишення повідомлень та коментарів на різних ресурсах, обмін миттєвими повідомленнями та інші форми зв'язку.

Система Tor має багатоступеневу схему шифрування: початкові дані шифруються та дешифруються декілька разів, потім передаються через ряд вузлів системи, кожен з яких розшифровує певний рівень шифру перед передачею даних наступному вузлу і, зрештою, до місця призначення. Це значною мірою зменшує ймовірність злому шифру і доступу до оригінальних даних в процесі передачі. Використання самої системи Tor, а також клієнтського програмного забезпечення є абсолютно безкоштовним.

Encrypted Multicasting Chat System – система групового обміну повідомленнями, яка для захисту інформації використовує алгоритм шифрування RSA та стандарт сертифікації X.509. Дана система складається з клієнтської та серверної частин. Сервер представляє собою віддалений менеджер ключів та сертифікаційний центр, а клієнт – звичайний програмний додаток, який надає користувачеві можливість захищеного спілкування у певній локальній мережі.

2.2 Обґрунтування вибору засобів для побудови системи та мови програмування

В сучасній веб-розробці дуже популярною є розробка SPA-додатку.

Односторінковий додаток (англ. single-page application, SPA), також

відомий як односторінковий інтерфейс (англ. single-page interface, SPI) – це веб-застосунок чи веб-сайт, який вміщується на одній сторінці з метою забезпечити користувачу досвід близький до користування настільною програмою.

В односторінковому додатку весь необхідний код – HTML, JavaScript, та CSS – завантажується разом із сторінкою, або динамічно довантажується за потребою, зазвичай у відповідь на дії користувача. Сторінка не оновлюється і не перенаправляє користувача на іншу сторінку в процесі роботи з нею. Взаємодія з односторінковим додатком часто включає в себе динамічний зв'язок з веб-сервером.

Термін "односторінковий додаток" був придуманий Стівом Єном в 2005 році, хоча концепція обговорювалась ще на початку 2003 року, також Стюарт (stunix) Моріс описав "автономний веб-сайт" (Self-Contained website) з такими ж цілями та функціоналом в 2002 році.

Для побудови одно сторінкового веб-додатку я обрав такі засоби:

Мова програмування Javascript

Мова JavaScript була розроблена з метою надати web-сторінкам інтерактивність. Програми, написані на цій мові, називаються скриптами. Вони безпосередньо підключаються до HTML у браузері і виконуються миттєво після завантаження сторінки.

Програми на JavaScript є звичайним текстом і не вимагають спеціальної підготовки. В цьому відношенні JavaScript відрізняється від іншої мови програмування, відомої як Java.

Сучасний JavaScript є "безпечною" мовою загального призначення. Вона не надає низькорівневих можливостей роботи з пам'яттю та процесором, оскільки спочатку була спрямована на використання в браузерах, де такі можливості не є необхідними.

Що ж стосується інших можливостей – вони залежать від оточення, в якому запущений JavaScript. У браузері JavaScript вміє робити все, що відноситься до маніпуляції зі сторінкою, взаємодії з відвідувачем і, в якійсь мірі, з сервером:

Створювати нові HTML-теги, видаляти існуючі, змінювати стилі елементів, ховати, показувати елементи і т.п.

Реагувати на дії відвідувача, обробляти кліки миші, переміщення курсору, натискання на клавіатуру і т.п.

Посилати запити на сервер і завантажувати дані без перезавантаження сторінки (ця технологія називається "AJAX").

Отримувати і встановлювати cookie, запитувати дані, виводити повідомлення, тощо.

JavaScript – швидка і потужна мова, але браузер накладає на її виконання деякі обмеження.

Це зроблено для безпеки користувачів, щоб зловмисник не міг за допомогою JavaScript отримати особисті дані або якось нашкодити комп'ютеру користувача. Цих обмежень немає там, де JavaScript використовується поза браузером, наприклад на сервері. Крім того, сучасні браузери надають свої механізми по установці плагінів і розширень, які володіють розширеними можливостями, але вимагають спеціальних дій по установці від користувача.

JavaScript не може читати / записувати довільні файли на жорсткий диск, копіювати їх або викликати програми. Він не має прямого доступу до операційної системи. Сучасні браузери можуть працювати з файлами, але ця можливість обмежена спеціально виділеної Директорією - «пісочницею». Можливості щодо доступу до пристроїв також опрацьовуються в сучасних стандартах і частково доступні в деяких браузерах.

JavaScript, що працює в одній вкладці, не може взаємодіяти з іншими вкладками і вікнами, за винятком випадку, коли він сам відкрив це вікно або декілька вкладок з одного джерела (однаковий домен, порт, протокол). Є способи це обійти, і вони розкриті в підручнику, але вони вимагають спеціального коду на обидва документи, які знаходяться в різних вкладках або вікнах. Без нього, з міркувань безпеки, залізти з однієї вкладки в іншу за допомогою JavaScript можна.

З JavaScript можна легко посилати запити на сервер, з якого прийшла

сторінка. Запит на інший домен теж можливий, але менш зручний, так як і тут є обмеження, пов'язані з безпекою.

React.JS

React.js, здебільшого називають React, це open-source JavaScript бібліотека для створення інтерфейсів користувача, яка покликана вирішувати проблеми часткового оновлення вмісту веб-сторінки, з якими стикаються в розробці односторінкових додатків. Розробляється Facebook, Instagram і спільнотою індивідуальних розробників.

React дозволяє розробникам створювати великі веб-додатки, які використовують дані, які змінюються з часом, без перезавантаження сторінки. Його мета полягає в тому, щоб бути швидким, простим, масштабованим. React обробляє тільки користувацький інтерфейс у додатках. Це відповідає View у шаблоні модель-вид-контролер (MVC), і може бути використаний в поєднанні з іншими JavaScript бібліотеками або в великих фреймворках MVC, таких як AngularJS. Він також може бути використаний з React на основі надбудов, щоб піклуватися про частини без користувацького інтерфейсу побудови веб-додатків.

В даний час React використовують Khan Academy, Netflix, Yahoo, Airbnb, Sony, Atlassian та інші.

Особливості React.JS:

Однонаправлена передача даних. Властивості передаються в рендерер компоненту, як властивості html тега. Компонент не може напряму змінювати властивості, що йому передані, але може їх змінювати через callback функції. Такий механізм називають «властивості донизу, події нагору».

Віртуальний DOM. React підтримує віртуальний DOM, а не покладається виключно на DOM браузера. Це дозволяє бібліотеці визначити, які частини DOM змінилися, порівняно (diff) зі збереженою версією віртуального DOM, і таким чином визначити, як найефективніше оновити DOM браузера. Таким чином програміст працює зі сторінкою, вважаючи що вона оновлюється вся, але бібліотека самостійно вирішує які компоненти сторінки треба оновити.

JSX. Компоненти React зазвичай написані на JSX. Код, написаний на JSX, компілюється у виклики методів бібліотеки React. Розробники можуть так само писати на чистому JavaScript. JSX нагадує іншу мову, яку створили у компанії Фейсбук для розширення PHP, XHP.

Не лише рендерінг HTML в браузері. React використовують не лише для рендерінгу HTML в браузері. Наприклад, Facebook має динамічні графіки, які рендеряться в теги <canvas>, а Netflix та PayPal використовують ізоморфне завантаження для рендерінгу ідентичного HTML на сервері та клієнті.

Redux.JS

У міру того, як вимоги до одно сторінкових JavaScript додатків стають все більш високими, веб-програмісти змушені керувати все більшою кількістю станів (State) за допомогою JavaScript. Ці стани можуть включати в себе відповіді сервера, кешовані дані, а також дані, створені локально, але ще не збережені на сервері. Це також відноситься до UI-станів, таким як активний маршрут (route), виділений таб, показаний спіннер або нумерація сторінок і т.д.

Керувати станами, які постійно змінюються, складно. Якщо модель може оновити іншу модель, то уявлення може оновити модель, яка оновлює іншу модель, а це, в свою чергу, може викликати оновлення іншого представлення. У якийсь момент Ви більше не знаєте, що відбувається у вашому додатку. Ви більше не можете контролювати коли, чому, і як стан оновився. Коли система стає непрозорою і недетермінованою, важко виявити помилки або додавати нову функціональність.

Це досить кепсько, беручи до уваги нові вимоги, які стають звичними для фронтенд розробки, такі як, обробка оптимістичних оновлень (optimistic updates), рендер на сервері, вилучення даних перед виконанням переходу на сторінку і так далі. Front-end розробники намагаються впоратися зі складністю, з якою ніколи не мали справи, і тому неминуче виникає запитання: настав час здатися?

Ця складність виникає через змішування двох, дуже нелегких для розуміння концепцій: зміни (mutation) і асинхронність (asynchronicity). Обидві ці

концепції можуть бути прекрасними окремо, але разом вони перетворюються в безлад. Бібліотеки, аналогічні React, намагаються вирішити цю проблему на рівні уявлення, видаляючи асинхронність і пряме маніпулювання DOM. Проте, React залишає управління станом даних за програмістом. І тут в справу вступає Redux.

Йдучи слідами Flux, CQRS і Event Sourcing, Redux намагається зробити зміни стану передбачуваними, шляхом введення деяких обмежень на те, як і коли можуть відбутися оновлення. Ці обмеження відображені в трьох принципах Redux:

– Стан всього додатку збережено в дереві об'єктів всередині одного сховища;

– Стан тільки для читання;

– Мутації написані, як чисті функції;

Socket.IO

Socket.IO - JavaScript-бібліотека для веб-додатків і обміну даними в реальному часі. Складається з двох частин: клієнтської, яка запускається в браузері і серверної для node.js. Обидва компоненти мають схожу API. Подібно node.js, Socket.IO подієво-орієнтована.

Socket.IO головним чином використовує протокол WebSocket, але якщо потрібно, використовує інші методи, наприклад Adobe Flash сокети, JSONP запити або AJAX запити, надаючи той же самий інтерфейс. Крім того, що Socket.IO може бути використана, як оболонка для WebSocket, вона містить багато інших функцій, включаючи мовлення на кілька гнізд, зберігання даних, пов'язаних з кожним клієнтом, і асинхронний ввід / вивід.

Може бути встановлена через npm (node packaged manager).

Node.JS

Node.js – платформа з відкритим кодом для виконання високопродуктивних мережевих застосунків, написаних мовою JavaScript. Засновником платформи є Раян Дал (Ryan Dahl). Платформа node.js перетворила мову JavaScript, що в основному використовувалась в браузерах на мову

загального використання з великою спільнотою розробників.

Node.js характеризується такими властивостями:

– асинхронна однопотокова модель виконання запитів

– неблокуючий ввід/вивід

– система модулів CommonJS

– рушій JavaScript Google V8

Для управління модулями використовується пакетний менеджер npm (node package manager).

NPM

NPM є найпопулярнішим менеджером пакетів для мови програмування JavaScript. Це менеджер пакетів за замовчуванням для середовища виконання JavaScript Node.js. Він складається з клієнта командного рядка, який також називається NPM і онлайнової бази даних публічних пакетів, який називається npm реєстром.

WebRTC

WebRTC (Web Real-Time Communications) – це технологія, яка дозволяє Web-додаткам і сайтам захоплювати і транслювати аудіо та/або відео медіа-потоки опціонально, а також передавати між браузерами довільні дані, без обов'язкового використання посередників. Набір стандартів, які включає в себе технологія WebRTC, дозволяє обмінюватися даними і проводити телеконференції в режимі вузол-вузол без необхідності користувачеві встановлювати плагіни або будь-яке інше додаткове програмне забезпечення. WebRTC складається з декількох взаємопов'язаних програмних інтерфейсів (API) і протоколів, які працюють разом.

Його включення до World Wide Web Consortium (W3C) підтримується Google, Mozilla та Opera.

WebRTC розповсюджується за ліцензією BSD-3, а вихідний код засновується на продукті від Global IP Solution, яка була придбана компанією Google у травні 2010.

Для сучасних веб-додатків, тим паче – односторінкових, необхідна серверна частина додатку – так названий Back-End, або API. Він буде приймати, обробляти запити від клієнтської частини, керувати та зберігати дані в базі даних.

В якості основної серверної мови програмування була обрана PHP.

PHP (англ. PHP: Hypertext Preprocessor — PHP: гіпертекстовий препроцесор), попередня назва: Personal Home Page Tools — скриптова мова програмування, була створена для генерації HTML-сторінок на стороні веб-сервера. PHP є однією з найпоширеніших мов, що використовуються у сфері веб-розробок (разом із Java, .NET, Perl, Python, Ruby). PHP підтримується переважною більшістю хостинг-провайдерів. PHP — проект відкритого програмного забезпечення.

PHP інтерпретується веб-сервером у HTML-код, який передається на сторону клієнта. На відміну від скриптової мови JavaScript, користувач не бачить PHP-коду, бо браузер отримує готовий html-код. Це є перевага з точки зору безпеки, але погіршує інтерактивність сторінок. Але ніхто не забороняє використовувати PHP для генерування JavaScript-кодів, які виконуються вже на стороні клієнта.

Для підвищення зручності, швидкості та продуктивності розробки і експлуатації проекту застосовується PHP-фреймворк Laravel 5.3

Laravel — безкоштовний PHP-фреймворк з відкритим кодом, створений Taylor Otwell і призначений для розробки веб-додатків відповідно до шаблону model–view–controller (MVC). Деякими з особливостей Laravel є модульна система упаковки з виділеним менеджером залежностей, різні способи для доступу до реляційних баз даних, утиліти, які допомагають в розгортанні додатків і технічного обслуговування, а також його орієнтація на синтаксичний цукор.

Станом на березень 2015 року, Laravel вважається одним з найпопулярніших PHP фреймворком.

Ключові особливості, що лежать в основі архітектури Laravel:

– Пакети (англ. Packages) - дозволяють створювати і підключати модулі в

форматі Composer до додатка на Laravel. Багато додаткових можливостей вже доступні у вигляді таких модулів.

– Eloquent ORM - реалізація шаблону проектування ActiveRecord на PHP. Дозволяє строго визначити відносини між об'єктами бази даних. Стандартний для Laravel будівник запитів Fluent підтримується ядром Eloquent.

– Логіка додатка – частина розроблюваного додатку, оголошена або за допомогою контролерів, або маршрутів (функцій-замикань). Синтаксис оголошень схожий на синтаксис, який використовується в каркасі Sinatra.

– Зворотня маршрутизація пов'язує між собою генеровані додатком посилання і маршрути, дозволяючи змінювати останні з автоматичним оновленням пов'язаних посилань. При створенні посилань за допомогою іменованих маршрутів Laravel автоматично генерує кінцеві URL.

– REST-контролери – додатковий шар для поділу логіки обробки GET- і POST-запитів HTTP.

– Автозавантаження класів - механізм автоматичного завантаження класів PHP без необхідності підключати файли їх визначень в include. Завантаження на вимогу запобігає завантаження непотрібних компонентів; завантажуються тільки ті з них, які дійсно використовуються.

– Укладачі уявлень (англ. View composers) – блоки коду, які виконуються при генерації уявлення (шаблону).

– Інверсія управління (англ. Inversion of Control) - дозволяє отримувати екземпляри об'єктів за принципом зворотного управління. Також може використовуватися для створення і отримання об'єктів-одинаків (англ. Singleton).

– Міграції – система управління версіями для баз даних. Дозволяє пов'язувати зміни в коді програми зі змінами, які потрібно внести в структуру БД, що спрощує розгортання і оновлення програми.

– Модульне тестування (юніт-тести) – грає дуже велику роль в Laravel, який сам по собі містить велику кількість тестів для запобігання регрессий (помилок внаслідок поновлення коду або виправлення інших помилок).

– Сторінкова пагінація (англ. Pagination) – спрощує генерацію сторінок, замінюючи різні методи вирішення цієї задачі єдиним механізмом, вбудованим в Laravel.

MySql

MySQL є найбільш пристосованою для застосування в середовищі web СУБД (системою управління базами даних). Не секрет, що для виконання додатків клієнта на більшості хостинг-майданчиків провайдери надають невелику кількість ресурсів (як обчислювальних, так і дискових). Тому для даного застосування необхідна високоефективна СУБД, що володіє при цьому високою надійністю (більшість web-додатків і сайтів повинні працювати в режимі 24/7).

З усіх цих причин MySQL стала непорушним стандартом в області СУБД для web, а тепер в ній розвиваються можливості для використання її в будь-яких критичних бізнес-додатках, тобто конкурує на рівних з такими СУБД таких виробників, як Oracle, IBM, Microsoft і Sybase.

Основні переваги MySQL:

– багатопоточність, підтримка декількох одночасних запитів;
– оптимізація зв'язків з приєднанням багатьох даних за один прохід;
– записи фіксованої і змінної довжини;
– ODBC драйвер;
– гнучка система привілеїв і паролів;
– гнучка підтримка форматів чисел, рядків змінної довжини і міток часу;
– інтерфейс з мовами C і Perl, PHP;
– швидка робота, масштабованість;
– сумісність з ANSI SQL;
– безкоштовна в більшості випадків;
– хороша підтримка з боку провайдерів послуг хостингу;
– швидка підтримка транзакцій через механізм InnoDB.

Composer

Composer — менеджер пакетів прикладного рівня для мови програмування

PHP що забезпечує стандартний формат для управління залежностями у програмному забезпеченні та необхідними бібліотеками. Він був розроблений Нілом Адерманом і Хорді Боггіано, які і досі супроводжують проект. Вони почали розробку в квітні 2011 року і вперше випустили його 1 березня 2012 року. Composer брав натхнення з «npm» для Node.js і «bundler» для Ruby's. Спочатку в основі був алгоритм залежностей проекту з openSUSE's libzypp.

Composer працює з командного рядка і встановлює залежності (наприклад, бібліотеки) для застосунку. Він також дозволяє користувачам встановлювати PHP пакети, доступні на «Packagist», який є його основним сховищем і містить доступні пакети. Він також реалізує автозавантажувач класів, для встановлених бібліотек і це полегшує використання коду від сторонніх розробників.

Composer використовується як складова частина декількох популярних PHP проектів з відкритим вихідним кодом, наприклад: Laravel, Symfony.

Redis

Redis – розподілене сховище пар ключ-значення, які зберігаються в оперативній пам'яті, з можливістю забезпечувати довговічність зберігання за бажанням користувача. Це програмне забезпечення з відкритим сирцевим кодом написане на ANSI C. Розробка Redis фінансується VMware. Сирцеві тексти проекту поширюються в рамках ліцензії BSD. Redis надає схожі на Memcached функції для зберігання даних в форматі ключ/значення, розширені підтримкою структурованих даних, таких як списки, хеші і множини. На відміну від Memcached, Redis забезпечує постійне зберігання даних на диску і гарантує збереження БД у разі аварійного завершення роботи. Клієнтські бібліотеки доступні для більшості популярних мов, включаючи Perl, Python, PHP, Java, Ruby і Tcl.

Sass

Sass (англ. Syntactically Awesome Stylesheets) – скриптова метамова, яка інтерпретується в каскадні таблиці стилів (CSS). Спроектована Гемптоном Кетліном та розроблена Наталі Вейзенбаум. Sass призначений для підвищення

рівня абстракції коду та спрощення файлів CSS.

ASP.NET – технологія створення веб-додатків та веб-сервісів від компанії Microsoft. Вона входить до складу платформи Microsoft.NET, тому розробники можуть писати код для ASP.NET, використовуючи практично будь-які мови програмування, що входять у комплект NET Framework (C#, Visual Basic.NET, JScript.NET та інші). ASP.NET має перевагу у швидкості у порівнянні зі скриптовими технологіями, тому що при першому зверненні код компілюється і поміщається в спеціальний кеш, а потім, при подальших зверненнях, виконується з кешу, не вимагаючи витрат часу на парсинг, оптимізацію, і подібні операції.

Основні переваги технології ASP.NET:

– наявність засобів кешування;

– зручні засоби розробки дизайну – майстер-сторінки, обкладинки;

– вбудовані засоби збереження даних сесії і додатку на сервері;

– підтримка великої кількості мов програмування;

– сувора типізація, що зменшує вірогідність створення коду з помилками, а також підвищує швидкодію програми;

– відділення коду від візуальної частини програми.

Архітектура P2P – варіант архітектури системи, в основі якої лежить мережа рівноправних вузлів. Комп'ютерні мережі типу peer-to-peer (або P2P) засновані на принципі рівноправності учасників і характеризуються тим, що їх елементи можуть зв'язуватися між собою безпосередньо, на відміну від традиційної клієнт-серверної архітектури, в якій зв'язок відбувається лише між клієнтами та центральним сервером. В чистій P2P-мережі не існує поняття клієнтів або серверів, є лише рівні вузли, які одночасно функціонують як клієнти та сервери по відношенню до інших вузлів мережі. Така організація дозволяє зберігати працездатність мережі при будь-якій конфігурації її учасників. Проте практикується використання P2P мереж які все ж таки мають сервери, але їх роль полягає вже не у наданні сервісів, а у підтримці інформації з приводу сервісів, що надаються клієнтам мережі.

Використання архітектури мережі типу P2P дає наступні переваги:

– розподіл та зменшення вартості розгортання мережі – архітектура P2P може допомогти розподілити вартість серед користувачів;

– об'єднання ресурсів - кожен вузол в системі P2P надає певні ресурси як, наприклад, обчислювальна потужність або пам'ять;

– динамічність оточення;

– забезпечення високого рівня автономності та надійності;

– спрощена масштабованість системи.

Головним недоліком архітектури P2P є те, що через відсутність центрального сервера P2P-мережі працюють значно повільніше. З тієї ж причини клієнти таких мереж змушені пропускати через себе значну кількість запитів від інших користувачів, що зменшує швидкість передачі їх власних запитів.

Стандарт X.509 був прийнятий в 1988 році. Він визначає жорстку ієрархічну систему сертифікаційних центрів, які створюють сертифікати. Цей стандарт схожий на модель веб-довіри, в якій кожен може поставити свій підпис, таким чином підтвердивши законність інших сертифікатів, що використовують ключі. За схемою X.509 сертифікаційний центр видає сертифікат, закріплюючи публічний ключ за конкретним або за альтернативним (адреса електронної пошти чи DNS-запис) іменем.

Сертифікат відкритого ключа підпису або шифрування являє собою структурований двійковий запис, що містить елементи даних, які супроводжуються підписом видавця сертифікату. У сертифікаті є десять основних полів: шість обов'язкових і чотири параметричних. Велика частина інформації, що вказується в сертифікаті не є обов'язковою, а зміст обов'язкових полів сертифіката може варіюватися. До обов'язкових полів належать:

– серійний номер сертифіката;

– ідентифікатор алгоритму, за яким здійснювався електронний підпис;

– ім’я видавця сертифіката;

– термін придатності сертифіката;

– відкритий ключ суб’єкта – сторона, яка контролює секретний ключ, що відповідає даному відкритому;

– ім’я суб’єкта сертифіката.

2.3 Розгорнута постановка завдання

Згідно з технічним завданням на кваліфікаційну бакалаврську роботу, реалізації підлягає програмне забезпечення, яке призначено для системи кібербезпеки захищеного обміну даними у мережі Інтернет.

В процесі розробки бакалаврської роботи необхідно виконати наступні за:

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

б) вибрати та обґрунтувати методику побудови системи, розробити функціональну та структурну схеми системи;

в) розробити алгоритми для реалізації програмного забезпечення стосовно теми роботи, побудувати блок-схеми алгоритмів програми та підпрограм;

г) розробити програмне забезпечення системи, що дозволить реалізувати задачу, поставлену технічним завданням задачу;

д) організувати інтерфейс користувача та обробку виключних ситуацій;

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

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

3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

3.1 Опис функціонування системи

У процесі створення системи обміну повідомленнями, для ефективної передачі та маршрутизації пакетів даних, було обирано мережеві протоколи TCP і IP, які є частиною стеку протоколів TCP/IP. Для захисту даних використовується алгоритм асиметричного шифрування RSA.

Транспортний протокол TCP гарантує відповідальну доставку даних через попередньо встановлене з'єднання між вузлами мережі. Завдяки міжмережевому протоколу IP, можна використовувати IP-адресацію мереж та мережевих вузлів для направлення пакетів даних. Використання цих протоколів значно полегшує впровадження мережевого взаємодії в процесі розробки системи. Отже, в цьому розділі ми розглянемо структуру та ключові протоколи, які входять до набору протоколів TCP/IP.

TCP/IP, або Transmission Control Protocol / Internet Protocol (Протокол контролю передачі / Інтернет-протокол), фактично являє собою набір різноманітних протоколів. Саме через цю різноманітність TCP/IP часто називають стеком протоколів, де TCP і IP є основними компонентами. TCP/IP є основними протоколами Інтернету, відповідаючи за поділ вихідного повідомлення на пакети (TCP), доставку цих пакетів до вузла отримувача (IP) та відновлення оригінального повідомлення з пакетів (TCP).

Університет Берклі вніс значний вклад у розвиток стека протоколів TCP/IP, який отримав свою назву від широко використовуваних протоколів IP та TCP, інтегрувавши ці протоколи в свою версію операційної системи UNIX. Висока популярність цієї ОС сприяла глобальному розповсюдженню протоколів TCP, IP та інших протоколів, що входять до стека. На сьогоднішній день цей стек протоколів використовується для обміну даними між комп'ютерами у всесвітній

інформаційній мережі Інтернет, а також у безлічі корпоративних мереж.

Протягом багатьох років експлуатації в мережах різних країн та організацій, стек TCP/IP інтегрував в себе велику кількість протоколів додаткового рівня. Серед них відзначаються такі відомі протоколи, як FTP для передачі файлів, протокол емуляції терміналу Ethernet, поштовий протокол SMTP, використовуваний для електронної пошти в мережі Ethernet, гіпертекстові сервіси WWW та багато інших. На рисунку 3.1 показані рівні TCP/IP.

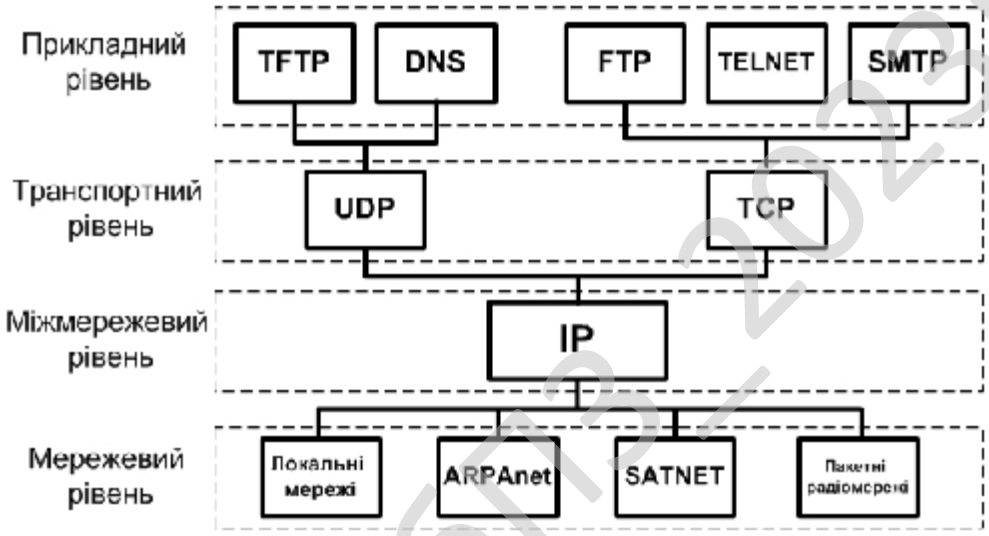

Рисунок 3.1 – Схема рівнів протоколу TCP/IP

Оскільки стек TCP/IP був спочатку розроблений для глобальної мережі Інтернет, він має численні особливості, які надають йому перевагу перед іншими протоколами при побудові мереж, що охоплюють глобальні зв'язки. Однією з особливо корисних властивостей, яка дозволяє застосовувати цей протокол у великих мережах, є його здатність до фрагментації пакетів. Фактично, велика складна мережа часто складається з мереж, побудованих на різних принципах. У кожній з цих мереж може бути своя власна максимальна довжина передаваного кадру. У таких випадках, при переході з однієї мережі з великою максимальною довжиною до мережі з меншою максимальною довжиною, може знадобитися розбиття переданого кадру на кілька частин. Протокол IP у стеку TCP/IP

ефективно вирішує цю проблему.

Ще одною особливістю TCP/IP є гнучка система адресації, яка дозволяє більш просто, порівняно з іншими протоколами аналогічного призначення, об'єднувати мережі різних технологій.

У TCP/IP дуже ефективно використовуються можливості широкомовних повідомлень. Ця властивість є необхідною при роботі з повільними каналами зв'язку, які характерні для територіальних мереж.

Натомість, протоколи стеку TCP/IP мають високі вимоги до ресурсів і складність адміністрування IP-мереж. Їх широкий функціонал потребує значних обчислювальних ресурсів для реалізації. Гнучка система адресації та відмова від широкомовних розсилок призводять до наявності різних централізованих служб, таких як DNS, DHCP і т.д., у IP-мережі. Кожна з цих служб спрямована на спрощення адміністрування мережі, включаючи конфігурування обладнання, але вимагає уважного контролю з боку адміністраторів.

IP протокол (IP - Internet Protocol) є найбільш поширеною реалізацією ієрархічної схеми мережевої адресації. Цей протокол відповідає за адресацію пакетів в мережі Інтернет, але не відповідає за встановлення з'єднань. Він є ненадійним і забезпечує лише негарантовану доставку даних. Термін "протокол без встановлення з'єднання" означає, що протокол не потребує виділеного каналу для взаємодії, як це відбувається, наприклад, під час телефонного дзвінка. Також він не вимагає процедури виклику перед передачею даних між вузлами мережі. Протокол IP вибирає найефективніший шлях з доступних варіантів, використовуючи прийняті методом маршрутизації рішення. Відсутність надійності і негарантована доставка не означає, що система працює погано або ненадійно, а вказує лише на те, що протокол IP не виконує перевірку чи був пакет доставлений за призначенням. Ці функції виконують протоколи транспортного та вищих рівнів (рис. 3.2). Транспортний рівень також відповідає за збірку пакетів у повідомлення в потрібній послідовності.

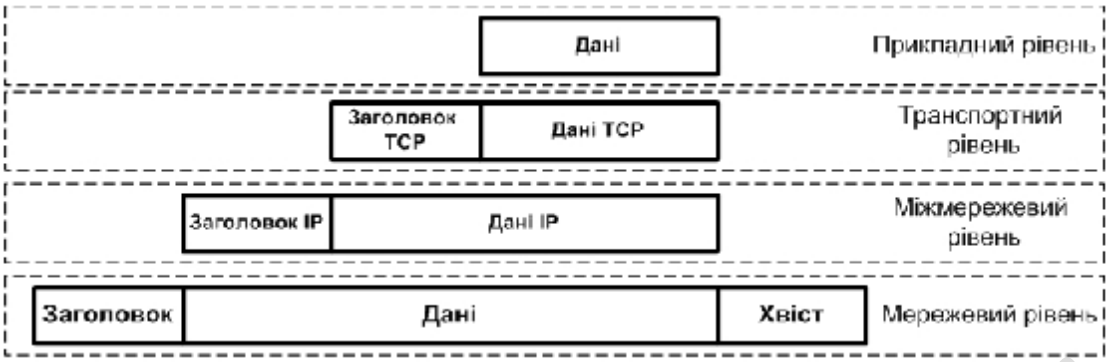

Рисунок 3.2 – Процес обробки пакета на різних рівнях

Інформація поступово просувається через рівні моделі OSI, і на кожному рівні вона обробляється протоколами цього рівня. Протокол IP розпізнає формат заголовка пакета, але не аналізує дані, які містяться в ньому. Він просто приймає і передає будь-які дані, що були передані протоколами верхніх рівнів.

Концепція клієнт-сервер є найпоширенішою архітектурою при розробці розподілених мережевих додатків. Вона описує взаємодію між клієнтською та серверною частинами системи і включає наступні основні компоненти:

- сервер – програма, процес або комп'ютер, який надає інформацію, ресурси або інші послуги програмам, які до нього звертаються;

- клієнт – програма, процес або комп'ютер, який запитує у сервера деяку інформацію або використовує сервіси, що надаються сервером;

- канал зв’язку – сукупність апаратних та програмних засобів, що забезпечують взаємодію між клієнтом та сервером.

Системи побудовані за архітектурою клієнт-сервер можуть включати декілька клієнтів і декілька серверів, при цьому всі сервери, так як і клієнти, можуть функціонувати паралельно і незалежно один від одного. Зазвичай один сервер одночасно обслуговує запити декількох клієнтів.

В роботі клієнт-серверної системи можна завжди виділити два процеси: серверний і клієнтський. Обмін інформацією між ними найчастіше відбувається за такою схемою (рисунок 3.3):

1. Клієнт посилає запит серверу через канал зв’язку і починає очікувати відповідь на свій запит;

2. Сервер приймає запит клієнта і виконує певні дії або шукає запитувані дані, а потім відсилає відповідь клієнту.

Рисунок 3.3 – Схема взаємодії клієнта та сервера

Таким чином всю інформацією, якою обмінюються клієнт і сервер можна розділити на два типи: запити і відповіді. Структура, процес формування та передачі повідомлень визначаються протоколом, за яким відбувається взаємодія клієнтської та серверної частин додатку.

Модель клієнт-серверної взаємодії передбачає розподіл обов'язків між клієнтом та сервером. При цьому виокремлюють три рівні операцій:

– рівень подання даних – являє собою інтерфейс користувача і відповідає за представлення даних користувачу і отримання від нього команд;

– прикладний рівень – реалізує основну логіку додатку, на цьому рівні здійснюється необхідна обробка інформації;

– рівень керування даними – забезпечує проведення операцій збереження даних та здійснення доступу до них.

Дворівнева архітектура використовується в клієнт-серверних системах, де сервер відповідає на клієнтські запити безпосередньо і в повному обсязі, при цьому використовуючи лише власні ресурси. Тобто сервер не викликає сторонні мережеві додатки і не звертається до сторонніх ресурсів для

виконання якої-небудь частини запиту. На рисунку 3.4 наведено схему взаємодії клієнта і сервера при дворівневій архітектурі.

Рисунок 3.4 – Дворівнева модель взаємодії клієнта і сервера

Дворівнева клієнт-серверна архітектура передбачає взаємодію двох програмних модулів – клієнтського та серверного. В залежності від того, як між ними розподіляються операції, які повинна виконувати система, розрізняють:

– модель тонкого клієнта, в рамках якої вся логіка додатку та управління даними зосереджена на сервері. Клієнтська програма забезпечує тільки функції рівня представлення;

– модель товстого клієнта, в якій сервер тільки керує даними, а обробка інформації та інтерфейс користувача зосереджені на стороні клієнта.

У дворівневій моделі істотною проблемою є розміщення на двох комп'ютерних системах трьох логічних рівнів – подання, виконання додатка й керування даними. Тому в даній моделі часто виникають або проблеми з масштабованістю й продуктивністю, якщо обрано модель тонкого клієнта, або проблеми, пов'язані з керуванням системою, якщо використовується модель товстого клієнта. Щоб уникнути цих проблем, необхідно застосувати альтернативний підхід – трирівневу або багаторівневу модель.

Трирівнева архітектура передбачає відділення прикладного рівня від управління даними. Відокремлюється окремий програмний рівень, на якому

зосереджується прикладна логіка додатку, а управління даними здійснюється сервером даних.

Трирівнева архітектура може бути розширена до багаторівневої шляхом виділення додаткових серверів, кожен з яких представлятиме власні сервіси і користується послугами інших серверів різних рівнів (рисунок 3.5).

Рисунок 3.5 – Трирівнева модель взаємодії клієнта і сервера

Таким чином, дворівнева архітектура простіша, так як всі запити обслуговуються одним сервером, але саме через це вона менш надійна і пред'являє підвищені вимоги до продуктивності сервера.

Трирівнева архітектура складніша, але завдяки тому, що функції розподілені між серверами другого і третього рівня, ця архітектура забезпечує:

– високий ступінь гнучкості і масштабованості;

– високий рівень захищеності системи;

– високу продуктивність системи.

Сокет – програмний інтерфейс для забезпечення обміну даними між процесами, який являє собою абстрактну точку мережевого з'єднання. З'єднуючи два сокета, з'являється можливість обміну даними між локальними або віддаленими процесами.

Вперше сокети були розроблені для операційної системи UNIX в Каліфорнійському університеті в Берклі. За допомогою них було реалізовано

метод введення-виведення в ОС UNIX, що виконувався за наступним алгоритмом: відкрити ресурс, задавши відповідні параметри; прочитати дані з відкритого ресурсу або записати їх в ресурс; закрити ресурс. Згодом цей алгоритм було використано для реалізації мережевої взаємодії додатків.

Введені в операційну систему UNIX засоби міжпроцесної взаємодії і мережевого обміну працюють за одним і тим самим принципом. Усі ресурси, які використовуються для обміну інформацією, ідентифікуються дескриптором, який може посилатися на файл, пам'ять або інший канал зв'язку. Фактично, такий дескриптор представляє собою вказівник на внутрішню структуру даних операційної системи. Сокет, будучи таким же ресурсом, теж має дескриптор, а отже життєвий цикл сокета може мати три фази:

– відкриття сокета (створення дескриптора);

– запис або читання інформації (використання дескриптора);

– закриття сокета (видалення дескриптора).

Сокети поділяють на два типи: потокові і дейтаграмні.

Потоковий сокет – сокет із встановленим з'єднанням, який являє собою двонаправлений потік байтів, тобто його можна використовувати як для прийому, так і для передачі даних. Потоковий сокет забезпечує корекцію помилок при передачі даних, обробляє та зберігає послідовність отриманих даних. Цей тип сокетів використовується для обміну впорядкованими даними великого обсягу. Його характеристики залежать від використаного протоколу TCP, який забезпечує безпомилкову та послідовну передачу даних. При використанні потокових сокетів з'єднання між сторонами встановлюється перед початком передачі повідомлення. Це гарантує, що обидві сторони, що беруть участь у взаємодії, готові до цієї взаємодії. Потоки передачі даних формуються на явних з'єднаннях: сокет А посилає запит на з'єднання сокету Б, а сокет Б або приймає цей запит і встановлюється з'єднання, або відхиляє його і з'єднання не встановлюється.

Дейтаграмний сокет – сокет без організації з'єднання. При використанні таких сокетів явного з'єднання між ними не встановлюється – повідомлення

відправляється на вказаний сокет і може отримуватися від цього ж сокета.

В дейтаграмних сокетах передбачається використання для передачі даних протоколу UDP, який накладає деякі обмеження на розмір повідомлень, а також не гарантує доставку інформації адресату.

У порівнянні з дейтаграмними сокетами, потокові сокети забезпечують більш надійну передачу даних, але вимагають додаткових часових затрат на встановлення з'єднання. Це може бути неприпустимою ситуацією для деяких випадків, наприклад, для серверів, які забезпечують синхронізацію часу для своїх клієнтів. Тому для зменшення часових затрат у таких випадках використовуються дейтаграмні сокети.

Окремо слід зазначити низькорівневі сокети, які отримують пакети даних з мережі та обходять рівні TCP і UDP стека протоколів TCP/IP, передаючи отримані дані безпосередньо додатку або процесу. При використанні цього типу сокетів пакети даних не проходять через фільтр TCP/IP і не підлягають обробці перед надходженням до процесу. У таких випадках обробка отриманих даних і виконання дій, таких як видалення заголовків та розбір полів мережевого пакета, реалізується безпосередньо самим додатком, що отримав дані. Низькорівневі сокети найчастіше використовуються при розробці протокольних додатків нижчого рівня, таких як ping, traceroute або arp.

Сокет складається з двох складових частин: IP-адреси комп'ютера та номера порта. IP-адреси є унікальними в межах мережі Інтернет, а номера портів є унікальними для кожного окремого комп'ютера. Тому комбінація цих двох параметрів утворює унікальний сокет в мережі Інтернет. Це дозволяє процесу взаємодіяти з іншими процесами через мережу, використовуючи лише номер сокета.

Зазвичай клієнт-серверні додатки, які використовують сокети, складаються з клієнтської та серверної частин. Клієнтська частина ініціює з'єднання з сервером, а серверна частина, крім виконання своїх основних завдань, постійно перевіряє наявність та приймає запити на з'єднання від нових клієнтів,

прослуховуючи певний порт. Коли сервер отримує запит на з'єднання, він погоджується і створює новий сокет для взаємодії з цим клієнтом, який надіслав запит. Після цього клієнт і сервер можуть взаємодіяти один з одним через цей новий сокет, тоді як прослуховуючий сокет сервера продовжує приймати нові запити на з'єднання від інших клієнтів. Схема такої взаємодії показана на рисунку 3.6.

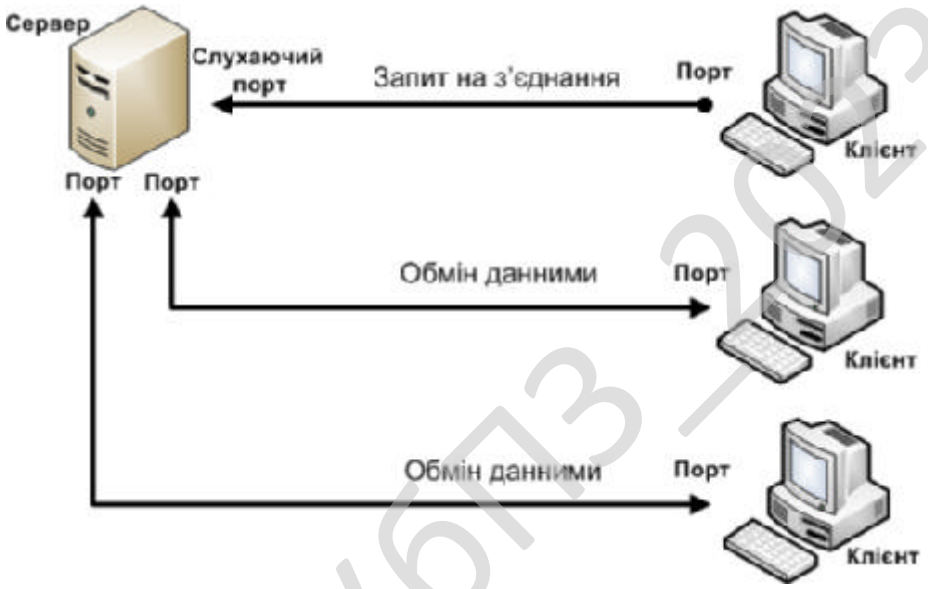

Рисунок 3.6 – Схема процесу підключення клієнтів до сервера з використанням сокетів

На сьогоднішній день існує велика кількість бібліотек і класів для різних мов програмування, які дозволяють використовувати сокети при розробці програмного забезпечення. Деякі з найпопулярніших бібліотек і класів включають WinSock, RawSocket, JSSE, Socket.IO, Net.Sockets.Socket та багато інших.

Для захисту даних та забезпечення приватності під час обміну повідомленнями по мережі Інтернет було вирішено використовувати криптографічні алгоритми DES, RSA та SHA-1. Алгоритм DES використано для шифрування вмісту електронних файлів, а алгоритми RSA та SHA-1 для створення цифрового підпису. Алгоритм DES відноситься до симетричних систем

шифрування, працює більш швидко, ніж асиметричні системи, але не може бути використаний сам по собі для захисту електронних документів при передачі по мережі інтернет. А алгоритм RSA відноситься до асиметричних систем шифрування. У таких системах для шифрування даних використовується один ключ, а для дешифрування – інший, тому їх також називають асиметричними. Перший ключ є відкритим і може бути опублікованим для використання всіма користувачами системи, які шифрують дані. Проте розшифрувати дані за допомогою відкритого ключа неможливо. Для дешифрування отримувач використовує другий ключ, який він тримає в таємниці.

Безпека алгоритму RSA базується на складності факторизації цілих чисел, тобто розкладанні їх на прості співмножники. Алгоритм використовує два ключі – відкритий і закритий (секретний). Відкритий і відповідний йому секретний ключі утворюють збалансовану пару ключів, які взаємно обертаються для задачі шифрування. Якщо повідомлення було зашифровано відкритим ключем, його можна розшифрувати лише за допомогою відповідного секретного ключа.

Алгоритм шифрування DES. Алгоритм DES (Data Encryption Standard) є симетричним блоковим шифром, який використовується для шифрування електронних файлів. Основний принцип роботи DES полягає у перетворенні блоків вхідних даних фіксованого розміру за допомогою ключа шифрування.

Основні етапи роботи алгоритму DES наступні:

1. **Генерація підключів.** Вхідний ключ шифрування, який складається з 64 бітів, перетворюється в 56-бітовий ключ шифрування шляхом видалення паритетних бітів. Потім 56-бітовий ключ розбивається на 16 підключів, по одному на кожен раунд шифрування.

2. **Перестановка початкових даних (Initial Permutation).** Вхідні дані, які також мають розмір 64 біта, переставляються згідно з фіксованою таблицею перестановок.

3. **Розбиття на ліву та праву частини.** Вихідні дані після перестановки діляться на дві рівні частини: ліву (32 біти) та праву (32 біти).

4. **Раунди шифрування.** У DES використовується 16 раундів шифрування, які застосовуються послідовно до лівої та правої частин даних. Кожен раунд включає в себе такі операції:

1) *Розширення правої частини (Expansion Permutation).* 32-бітна права частина даних розширюється до 48 бітів шляхом використання фіксованої таблиці розширення. Кожен біт правої частини дублюється та заміщується відповідним бітом з таблиці.

2) *XOR з підключем (Key Mixing).* Розширена права частина ксориться з підключем, що відповідає поточному раунду. Кожен біт розширеної правої частини ксориться з відповідним бітом підключа.

3) *Перетворення S-блоків (S-box Substitution).* Отримані змішані дані подаються на вхід S-блоків. S-блоки складаються з восьми таблиць, кожна з яких має 4 входи та 6 виходів. Кожні шість бітів змішаних даних використовуються як індекс для відповідної таблиці S-блока, і результатом є чотири біти. Таким чином, 48-бітовий вхід розбивається на вісім 4-бітових блоків, а потім кожен блок замінюється відповідними 4-бітовими значеннями з таблиці S-блока.

4) *Перестановка (P-блок).* Результати з S-блоків проходять через фіксовану таблицю перестановки, що перемішує біти. Це крок, що допомагає зберігати залежність між бітами даних після проходження через S-блоки.

5) *Об'єднання та зворотне перетворення (Combining and Reverse Permutation).* Ліва та права частини даних об'єднуються шляхом обміну місцями. Ліва частина стає новою правою частиною, а права частина стає новою лівою частиною. Цей обмін допомагає забезпечити залежність між частинами даних після кожного раунду.

6) *Завершальні раунди.* Останні два раунди DES відрізняються від попередніх. Після останнього раунду не виконується обмін лівої та правої частин даних. Замість цього, результати з останнього раунду об'єднуються без змін і

проходять через обернену таблицю перестановки, що відновлює їх початковий порядок.

5. **Вихідні дані.** Отримані дані після зворотного перетворення є зашифрованими даними.

При дешифруванні, процедура виконується в зворотному порядку. Зашифровані дані вводяться у зворотному напрямку, пройшовши кроки об'єднання та зворотного перетворення, перестановки S-блоків, XOR з підключем і розширення правої частини. Останні два раунди виконуються з оберненими підключами.

Алгоритм DES шифрує дані блоками розміром 64 біти і використовує 56-бітові ключі шифрування. Повторення цих кроків для кожного блоку даних забезпечує надійне і безпечне шифрування в рамках алгоритму DES.

Процес створення відкритого і закритого ключів шифрування для алгоритму RSA представляє собою ряд математичних операцій, виконання яких, в кінцевому результаті, дозволить отримати модуль ключа, відкриту та закриту експоненти, з яких, в свою чергу, будуть сформовані відкритий і закритий ключі.

Алгоритм генерації пари ключів такий:

1) обрати два великих простих числа p і q фіксованого розміру;

2) обчислити їх добуток (модуль) n;

3) обчислити функцію Ейлера:

$$\varphi(n) = (p-1)*(q-1); \tag{3.1}$$

4) підібрати ціле взаємнопросте з φ(n) число e таке, що $1 < e < \varphi(n)$;

5) за розширеним алгоритмом Евкліда обчислити число d, таке, що

$$e*d \equiv 1*(\mathrm{mod}\,\varphi(n)). \tag{3.2}$$

Після виконання цих дій будуть отримані наступні параметри для використання алгоритму шифрування: n – модуль ключа, e – відкрита експонента; d – секретна експонента, пара чисел (n,e) – відкритий ключ, пара чисел (n,d) – закритий ключ. При цьому числа p і q після генерації пари ключів можуть бути знищені, але в жодному разі не повинні бути розкриті. При генерації ключів,

розмір модуля ключів повинен бути якомога більшим, починаючи від 512 біт і більше. Це гарантуватиме високу стійкість до злому шифру.

Алгоритм Евкліда являє собою ефективний метод обчислення найбільшого спільного дільника двох цілих чисел. Названий на честь старогрецького математика Евкліда. Найбільший спільний дільник двох чисел це найбільше число, на яке діляться обидва дані числа без залишку. Алгоритм Евкліда заснований на тому, що найбільший спільний дільник не змінюється, якщо від більшого числа, із двох даних, відняти менше. Оскільки більше число постійно зменшується, повторне виконання цього кроку дає все менші числа, доки одне з них не стає рівним нулю. Коли це стається, то найбільший спільний дільник знайдено, ним буде те число, що залишилося більшим за нуль.

Алгоритм Евкліда можна використати в процесі генерації ключів для перевірки взаємної простоти двох чисел. При цьому знаходиться найбільший спільний дільник двох заданих чисел, і якщо він не рівний 1, то числа не є взаємно простими, інакше – числа взаємно прості.

Для знаходження найбільшого спільного дільника двох чисел a і b за алгоритмом Евкліда потрібно виконати наступну послідовність дій:

1) обчислити r – остачу від ділення a на b: $a = b * q + r$;

2) якщо r рівне нулю, то b є найбільшим спільним дільником і виконання алгоритму закінчується;

3) інакше, пара чисел (a,b) замінюється парою (b,r) і алгоритм повторюється, починаючи з другого пункту.

Ефективність роботи алгоритму можна описати як необхідну кількість кроків помножену на обчислювальні витрати кожного кроку. В 1844 році Ґабріель Ламі довів, що кількість необхідних кроків ніколи не перевищує кількості цифр меншого з чисел a і b помноженої на 5. На практиці алгоритм працює достатньо швидко.

Розширений алгоритм Евкліда. Для генерації одного з ключів в алгоритмі RSA використовується розширений алгоритм Евкліда, який окрім

знаходження найбільшого спільного дільника двох цілих чисел a і b, також знаходить деякі коефіцієнти x і y такі, що:

$$a * x + b * y = \gcd(a, b), \tag{3.3}$$

де a, b – числа, для яких шукається найбільший спільний дільник; x, y – коефіцієнти, що задовольняють умову 3.5; $gcd(a, b)$ – найбільший спільний дільник чисел a і b.

Тобто, розширений алгоритм Евкліда дозволяє знайти коефіцієнти, за допомогою яких найбільший спільний дільник двох чисел виражається через ці ж самі числа. Даний алгоритм є ефективним лише у випадку, якщо a і b являються взаємно простими числами, оскільки x – обернене до a за модулем b, а y – обернене до b за модулем числа a.

Таким чином, обчислення числа y, що являє собою один з ключів, зводиться до вирішення рівняння:

$$a * x + b * y = 1. \tag{3.4}$$

При цьому число x не має суттєвого значення.

Алгоритм вирішення рівняння 3.6 наступний:

1) визначити матрицю $E = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$;

2) обчислити r – остачу від ділення a на b: $a = b * q + r$;

3) якщо r рівне нулю, то другий стовпчик матриці дає рішення: $\begin{bmatrix} x \\ y \end{bmatrix}$ і виконання алгоритму закінчується;

4) інакше, обчислюється добуток: $E = E * \begin{bmatrix} 0 & 1 \\ 1 & -q \end{bmatrix}$;

5) пара чисел (a,b) замінюється парою (b,r) і алгоритм повторюється, починаючи з другого пункту.

У даному алгоритмі всі обчислення можна виконувати по модулю більшого з чисел a і b. Від'ємне значення числа q замінюється додатним, яке отримується шляхом віднімання від числа взятого в якості модуля числа q.

Для того, щоб сервер і клієнт мали можливість обмінюватись зашифрованими за алгоритмом RSA повідомленнями, необхідно визначити протокол обміну криптографічними ключами між клієнтом і сервером. Протокол обміну ключами визначає послідовність дій, яку виконують сторони комунікації з метою обміну публічними ключами.

Алгоритм шифрування RSA є алгоритмом з відкритим ключем, тобто в ході його реалізації кожною стороною генерується два ключі – відкритий і секретний. Відкритий ключ використовується для шифрування даних і є публічним, а закритий – для дешифрування, він повинен зберігатися в таємниці. Якщо повідомлення було зашифровано відкритим ключем, то розшифрувати його можна лише відповідним йому закритим ключем. Завдяки цій властивості криптографічних алгоритмів з відкритим ключем, для створення захищеного каналу кожній зі сторін достатньо згенерувати пару ключів: секретний і публічний та надіслати публічний ключ протилежній стороні.

Шифрування та дешифрування секретного ключа DES. Для шифрування секретного ключа DES був використаний алгоритм RSA. При шифруванні будь-якої інформації за допомогою алгоритму RSA її потрібно представити в числовому вигляді. Тому перед проведенням шифрування текстової інформації її спочатку перетворюють на набір цілих чисел. Зазвичай це реалізується шляхом перетворення тексту в потік байтів, або ж в набір цілих чисел, що представляють собою ASCII-коди або юнікоди відповідних символів. Ще однією вимогою до повідомлення, яке підлягає шифруванню є те, що воно повинно бути меншим за модуль ключа. Інакше повідомлення розбивається на блоки, які є меншими ніж модуль ключа, і операції шифрування та дешифрування проводяться над кожним блоком окремо.

Шифрування повідомлення відбувається наступним чином:

$$c = m^e \bmod n, \tag{3.5}$$

де c – зашифроване повідомлення; m – відкрите повідомлення; e – відкритий ключ; n – модуль ключа.

Після проведення перерахованих дій буде отримано зашифроване повідомлення, з якого можна буде отримати початкове повідомлення, використовуючи секретний ключ.

Дешифрування повідомлення відбувається так:

$$m = c^d \bmod n, \tag{3.6}$$

де m – відкрите повідомлення; c – зашифроване повідомлення; d – закритий ключ; n – модуль ключа.

Основними перевагами асиметричних алгоритмів шифрування, в тому числі і алгоритму RSA є такі:

– відсутність потреби передачі секретного ключа каналами зв'язку;

– порівняно довгий час життя ключів;

– забезпечення високої криптостійкості;

– досить проста програмна реалізація алгоритму.

До недоліків алгоритмів шифрування з відкритим ключем належать:

– використання дуже довгих ключів для забезпечення високого рівня надійності;

– порівняно низька швидкодія алгоритму.

Через низьку швидкодію алгоритму RSA зазвичай використовується гібридна система шифрування, яка полягає в тому, що повідомлення шифрують за допомогою продуктивніших симетричних алгоритмів з випадковим (сеансовим) ключем, а за допомогою RSA шифрують лише цей ключ. Після цього зашифрований ключ передається всім учасникам. Такий варіант захисту інформації є достатньо швидкодіючим і ефективним, але при його використанні необхідно вирішити проблему конфіденційного розподілу ключа сесії та гарантування його своєчасності. Для шифрування та дешифрування даних невеликого об'єму можливо використовувати стандартний алгоритм RSA.

Алгоритм створення цифрового підпису з використанням RSA та SHA-1 в загальних рисах включає наступні кроки:

1. **Генерація ключа RSA.** Спочатку генерується пара ключів RSA – приватний і публічний ключі. Приватний ключ залишається секретним і використовується для підпису, тоді як публічний ключ поширюється для перевірки підпису.

2. **Вибір файлу.** Файл, для якого треба створити цифровий підпис, вибирається у програмному забезпеченні.

3. **Обчислення хеш-значення файлу алгоритмом SHA-1.** Вираховується хеш повідомлення за допомогою алгоритму хешування SHA-1. SHA-1 генерує фіксованої довжини (160 бітів) хеш-значення на основі вхідного повідомлення.

4. **Підписування повідомлення.** Приватний ключ RSA використовується для зашифрування хешу повідомлення. Результатом є цифровий підпис, який представляє собою криптографічний перетворений хеш повідомлення.

5. **Передача файлу отримувачу.** Передача повідомлення та підпису: Повідомлення разом з цифровим підписом передається отримувачу.

6. **Верифікація підпису**. Перевірка підпису, отримувач використовує публічний ключ RSA, щоб розшифрувати цифровий підпис і отримати хеш повідомлення. Потім обчислює хеш повідомлення повторно, використовуючи алгоритм SHA-1. Якщо отриманий хеш співпадає з розшифрованим хешем з цифрового підпису, то підпис вважається валідним, що свідчить про збереження цілісності файлу та засвідчує його автора.

Шифрування за допомогою RSA забезпечує можливість перевірити автентичність повідомлення за допомогою публічного ключа, в той час як SHA-1 гарантує відсутність змін у повідомленні. Разом вони створюють надійний механізм для створення та перевірки цифрових підписів.

3.2 Розробка структурної схеми

На рисунку 3.7 зображена структурна схема розробленого додатку.

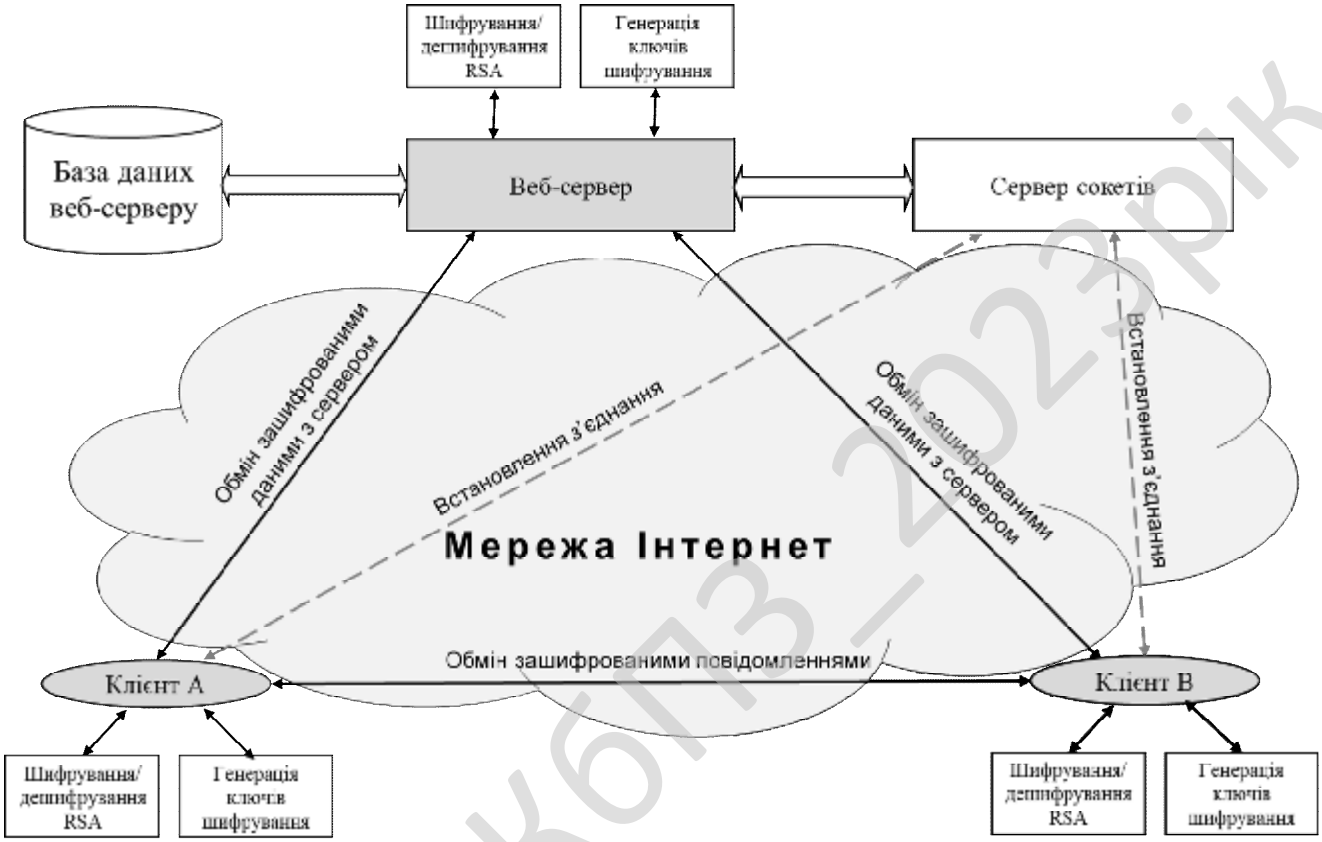

Рисунок 3.7 – Структурна схема додатку

На ній показана взаємодія розроблених частин додатку по своїм ролям між собою та дані, що передаються. До серверної частини відноситься веб-сервер, на якому розвернута розроблена система. До її обов'язків входить реєстрація, логін користувача,відправка сторінки з клієнтським додатком в браузер користувача і обробка асинхронних запитів від клієнтських додатків з усіма супутніми перевірками, відповідями, змінами в БД. Також у системі присутній сервер сокетів(також виступає як сигнальний сервер), який додає інтерактивності клієнтському додатку і розсилає повідомлення про події. В базі даних зберігаються дані про користувачів, дружбу між ними, групи, повідомлення,

таблиця для зброду паролів та таблиця з даними про версіонування БД. Клієнтський додаток є Javascript SPA-додатком, який працює весь період роботи користувача і зберігає дані в центральному сховищі. Клієнтські додатки можуть налаштовувати між собою peer-to-peer зв'язок для передачі даних за допомогою сигнального сервера, надсилаючи пропозиції з'єднання, відповіді, завершення та ICE кандидати. Усі повідомлення, що передаються по мережі шифруються алгоритмом RSA.

3.3 Розробка функціональної схеми

На рисунку 3.8 зображена структурна схема розробленого додатку.

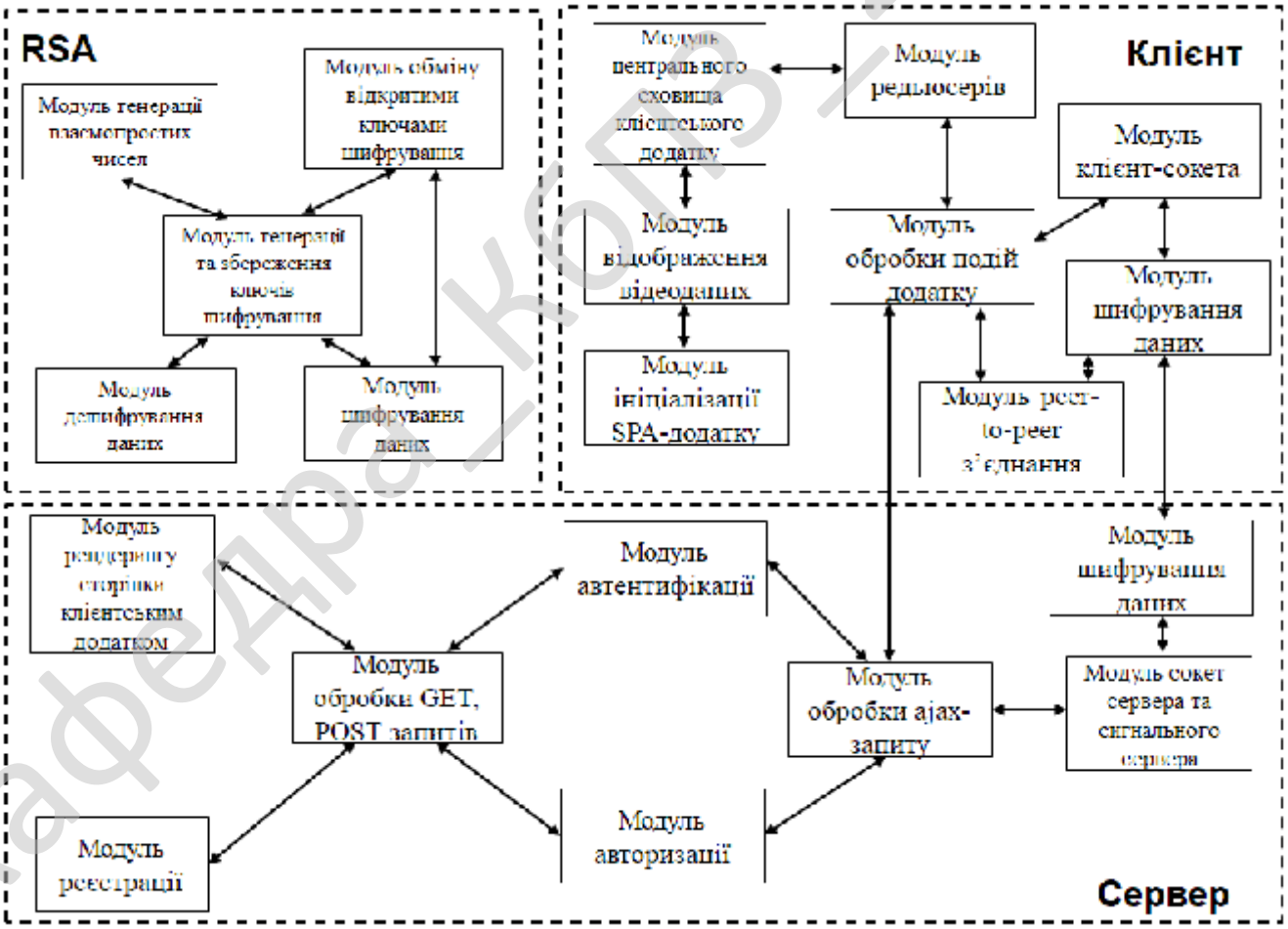

Рисунок 3.8 – Функціональна схема додатку

Вона поділена на 3 частини: серверну, клієнтську та модуль шифрування алгоритмом RSA. Серверна частина відповідає за зберігання та обробку даних, отримання та обробку запитів від клієнта (браузера користувача), та роботу сокет-сервера, який, в тому числі, виступає сигнальним сервером. Клієнтська частина є Javascript SPA-додатком, що дозволяє досягнути зменшення навантаження на сервер та збільшення швидкодії шляхом зменшення кількості запитів та їх об'єму, оскільки немає необхідності кожен раз генерувати та пересилати HTML код. Таким чином користувач може отримати такі ж відчуття, як від користування комп'ютерним додатком. Алгоритмом RSA також у розробленому ПЗ реалізовано мовою JavaScript.

3.4 Розробка діаграми процесів

За допомогою діаграми процесів можна розглянути послідовність виконання програмного забезпечення, а також визначити стани в яких може перебувати програмне забезпечення в певний момент часу та взаємозв'язок між цими станами.

Розглянемо діаграму процесів програмного забезпечення серверної та клієнтської частин розробленої системи, які зображено на рисунку 3.9.

Після запуску програмного забезпечення сервера з'являється головне вікно програми. В ньому відображається журнал подій та список під'єднаних клієнтів. Оскільки сервер ще не активний, то список клієнтів в будь-якому випадку буде порожнім. Після запуску сервера генерується пара ключів сервера: публічний та секретний і тільки після цього сервер переходить в режим прослуховування запитів на з'єднання, що надходитимуть від клієнтів. В разі отримання такого запиту сервер встановлює з'єднання з клієнтом і починає його обслуговування в новому потоці, при цьому продовжуючи приймати запити від інших клієнтів. В процесі встановлення з'єднання відбувається обмін відкритими ключами. Потім, через встановлене з'єднання, відбувається обмін

повідомленнями. Перед відправкою повідомлення воно спочатку шифрується, а після прийому повідомлення його потрібно дешифрувати. В процесі роботи сервера відбувається постійне оновлення списку активних клієнтів, в залежності від стану з'єднання з ними, а також – відображення в журналі всіх подій системи.

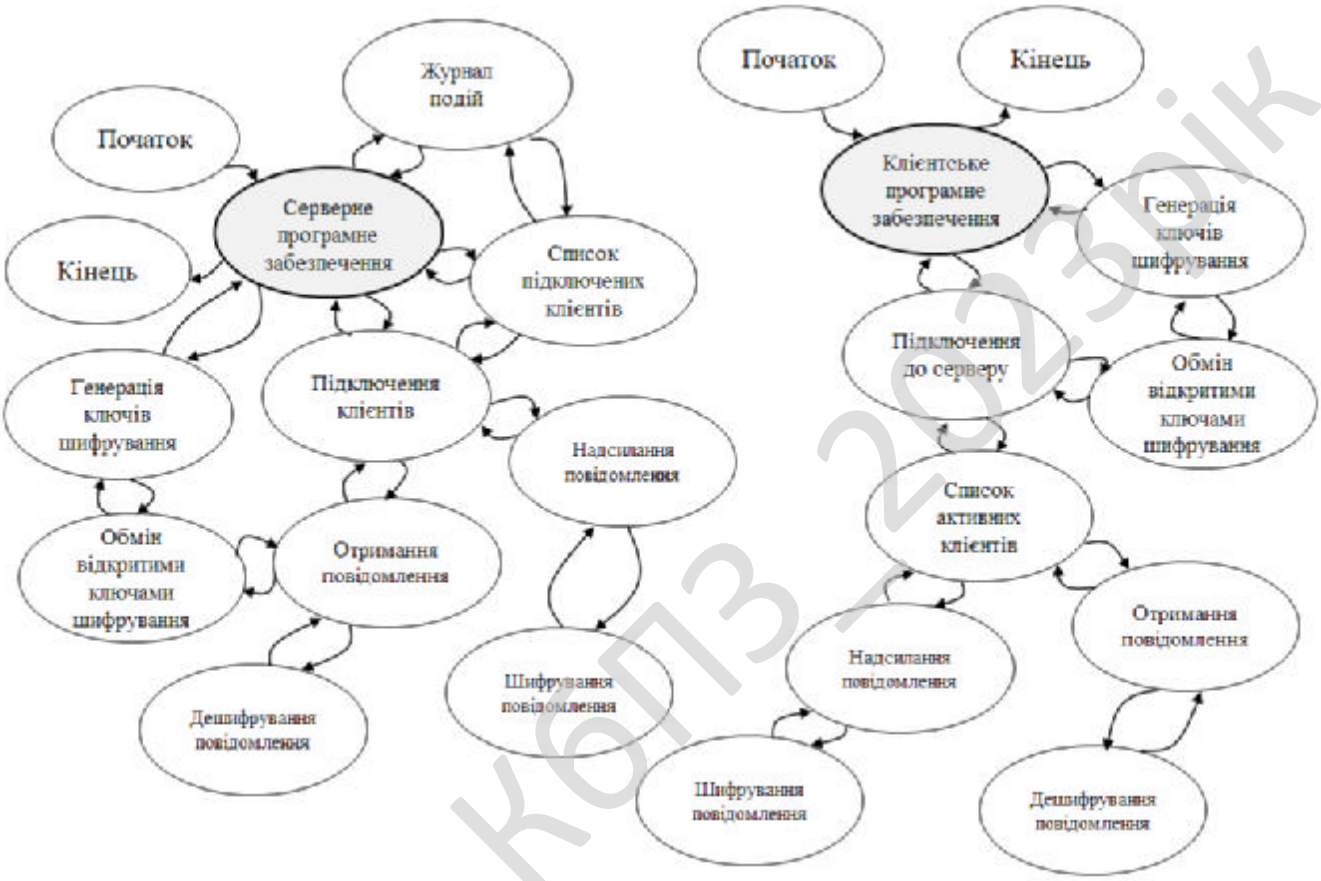

Рисунок 3.9 – Діаграма процесів системи

Робота клієнтської програми дещо схожа на роботу сервера. В даному випадку в головному вікні теж відображається список активних клієнтів, який надсилається сервером кожному клієнту. Після запуску додатка-клієнта також генерується пара ключів. Після цього на сервер посилається запит на з'єднання. І якщо сервер активний, то відбувається встановлення з'єднання і обмін відкритими ключами.

В процесі обміну повідомленнями, в разі відправки деякої інформації на сервер, виконується спочатку її введення, шифрування і лише потім відправка.

При отриманні деякої інформації, вона спочатку дешифрується і лише потім відображається у вікні переписки.

В нормальному режимі функціонування системи, тобто виключаючи аварійні ситуації, відключення та завершення роботи сервера та клієнта відбувається через інтерфейс користувача.

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

4.1 Блок-схеми та опис алгоритмів функціонування системи

В процесі проектування програмного забезпечення системи було розроблено ряд блок-схем, які відображають послідовність виконання і взаємозв'язок операцій, що виконуються в програмі. На рис. 4.1 зображена блок-схема роботи клієнтської частини розробленого програмного забезпечення.

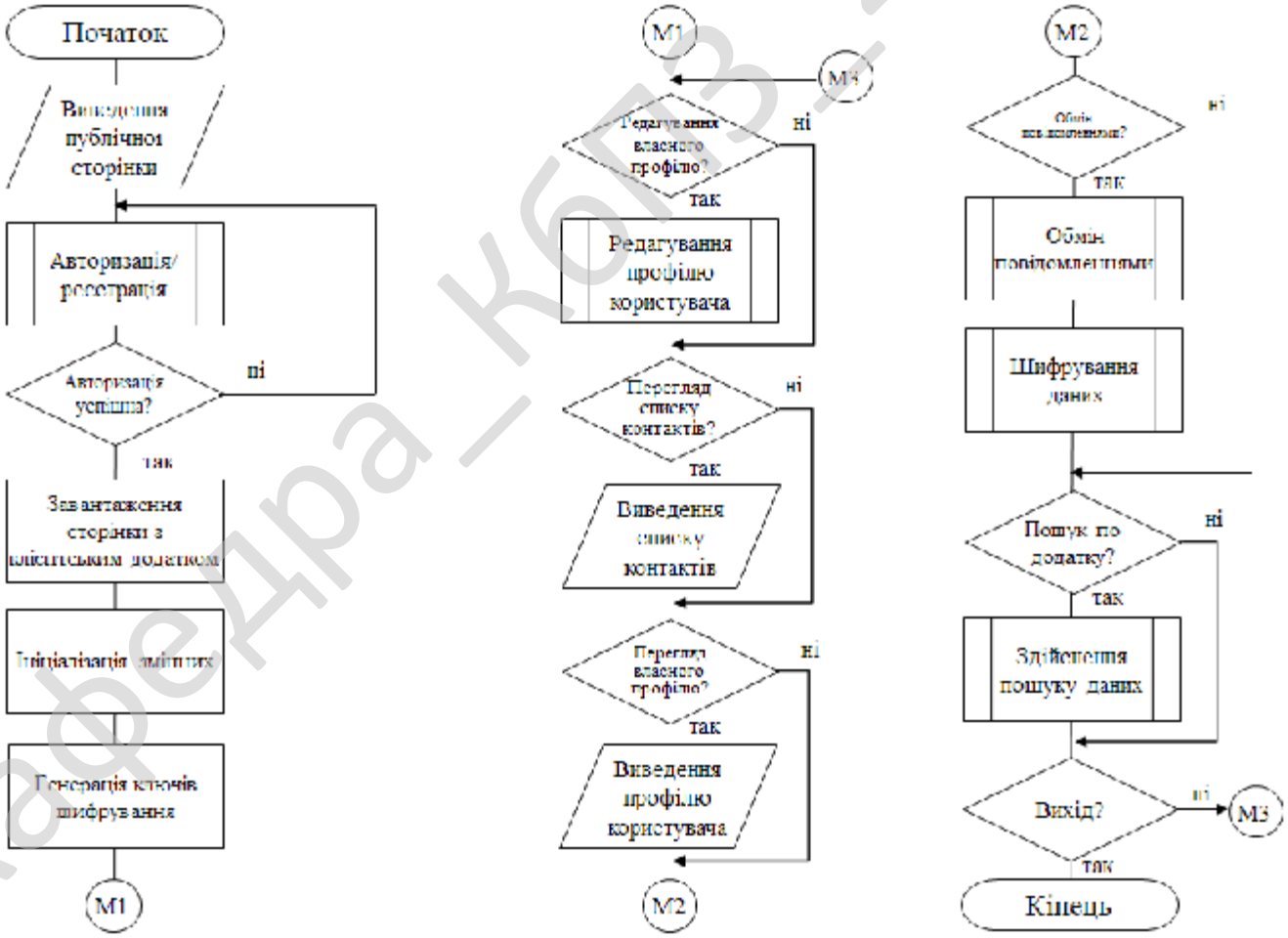

Рисунок 4.1 – Блок-схема клієнтської частини програмного забезпечення

Блок-схема функціонування серверної частини програмного забезпечення відображена на рисунку 4.2.

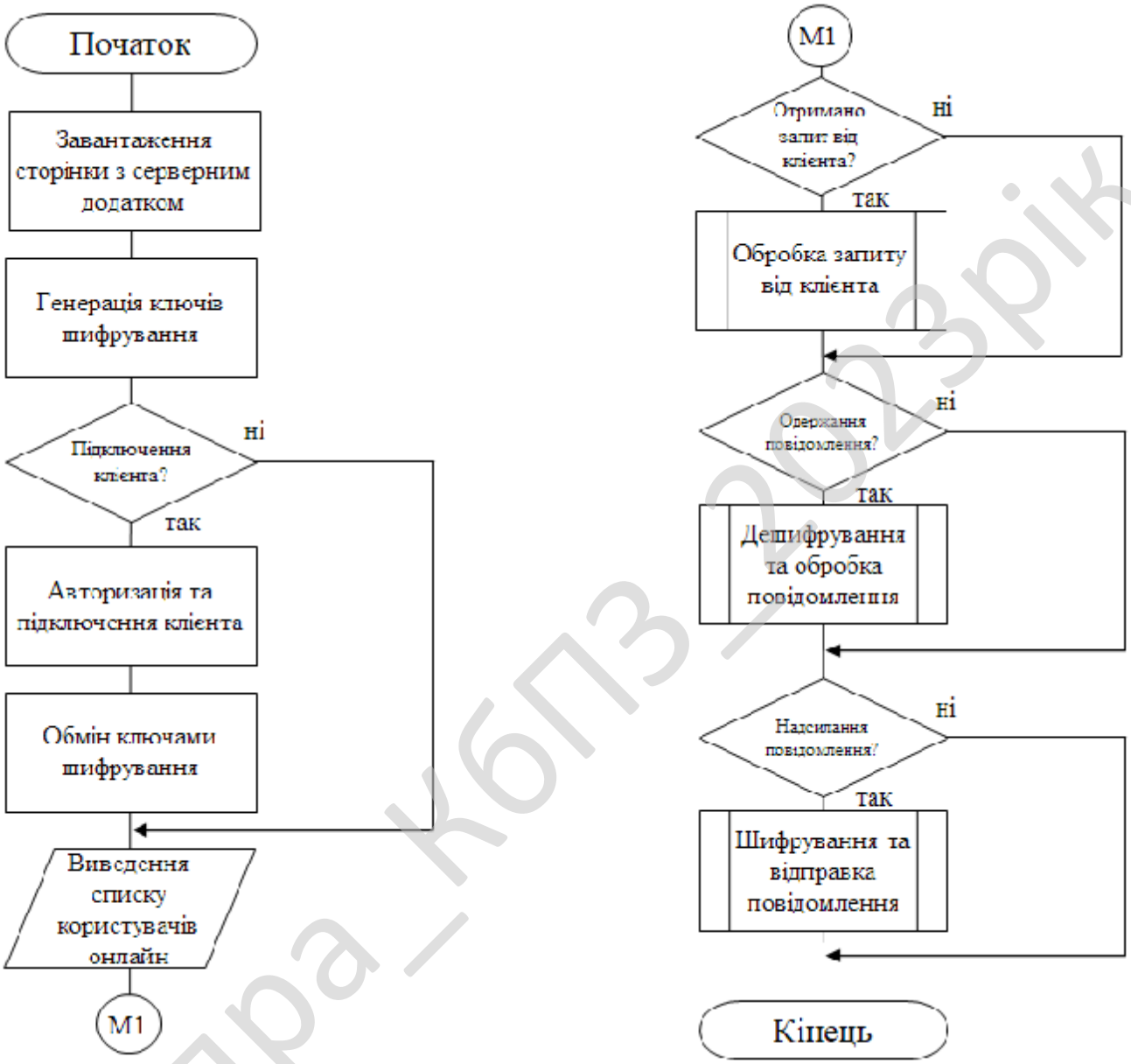

Рисунок 4.2 – Блок-схема серверної частини програмного забезпечення

Як видно з рис. 4.1, клієнтська частина програмного забезпечення може виконувати прийом і передачу повідомлень лише після авторизації та з'єднання з сервером. Тому спочатку розглянемо алгоритм роботи серверу (рис. 4.2).

Серверний додаток системи є багатопоточним. Кожен під'єднаний клієнт обслуговується в окремому потоці, що дозволяє паралельно працювати з кількома

клієнтами та підвищує ефективність роботи програми в цілому.

Після запуску сервера визначається IP-адреса комп'ютера, на якому запущено програмне забезпечення, створюється список клієнтів, який містить такі поля: IP-адреса клієнта, ідентифікатор та відкритий ключ клієнта.

Потім генерується пара ключів сервера і створюється сокет для прийому запитів на з'єднання від клієнтів. Прослуховування починає виконуватися в новому потоці, після чого сервер вважається активним і може приймати запити від клієнтів. В такому стані сервер буде знаходитися до того моменту, доки не буде зупинений користувачем або доки не виникне деяка аварійна ситуація в мережі. Після надходження запиту на з'єднання від клієнта, сервер вносить його в список клієнтів, відсилає йому свій відкритий ключ, отримує від нього його відкритий ключ і починає його обслуговування в новому потоці. При цьому, прийом запитів на з'єднання продовжується.

Всі дії, що відбуваються в системі відображаються в журналі подій сервера разом з точним часом їх здійснення.

Перед завершенням роботи серверної частини додатку виконується відключення всіх під'єднаних клієнтів та видалення динамічних ресурсів програми. Після цього всі клієнти втрачають можливість надсилати та отримувати повідомлення.

У випадку, якщо клієнт надіслав на сервер повідомлення, то воно приймається і дешифрується секретним ключем сервера. У випадку, якщо сервер має повідомлення для даного клієнта, то воно шифрується відкритим ключем даного клієнта і надсилається йому. Обмін даними можливий лише при наявності з'єднання між сервером і клієнтом. Якщо розрив з'єднання був ініційований клієнтом, то він видаляється зі списку активних клієнтів, а програмний потік, в якому обслуговувався клієнт, завершує своє виконання. При цьому в журналі подій сервера відображається інформація про всі здійснювані операції, а також час їх здійснення.

Виконання клієнтського додатку (рис. 4.1) починається з визначення

поточної IP-адреси комп'ютера на якому його було запущено та встановлення порту для з'єднання. Після цього генерується пара ключів клієнта. Для посилки запиту на з'єднання серверу, потрібно ввести ідентифікатор користувача, що буде використовуватися для його ідентифікації в чаті та IP-адресу сервера, з яким потрібно встановити з'єднання. Якщо ці дані вказані, то створюється сокет, через який здійснюється відправка запиту на з'єднання. Після цього, у випадку успішного з'єднання з сервером, відбувається обмін публічними ключами та прийом списку активних клієнтів. До тих пір, доки з'єднання з сервером буде встановлене у клієнта буде можливість приймати та надсилати повідомлення іншим активним клієнтам. Після прийому повідомлення проводиться його дешифрування, за допомогою секретного ключа клієнта. Перед відправкою, повідомлення вводиться і шифрується відкритим ключем сервера.

Для перевірки прав доступу користувачів було реалізовано наступні класи:

```
// FriendPolicy.php – клас для перевірки прав користувача

<?php

namespace App\Policies;

use App\Models\Friend;
use App\Models\User;
use Illuminate\Auth\Access\HandlesAuthorization;

class FriendPolicy
{
    use HandlesAuthorization;

    public function deleteFriendship(User $user, Friend $friend){
        return $user->id == $friend->user_id || $user->id == $friend-
>friend_id;
    }
}

// GroupPolicy.php - клас для перевірки прав групи користувачів

<?php

namespace App\Policies;

use App\Models\Group;
use App\Models\User;
use Illuminate\Auth\Access\HandlesAuthorization;

class GroupPolicy
{
    use HandlesAuthorization;
```

```
    public function getGroup(User $user, Group $group){
        return !!$group->users()->find($user->id);
    }

    public function updateGroup(User $user, Group $group){
        return !!$group->users()->find($user->id);
    }

    public function deleteGroup(User $user, Group $group){
        return !!$group->users()->find($user->id);
    }

    public function addUserToGroup(User $user, Group $group){
        return !!$group->users()->find($user->id);
    }
}
```

Виведення та введення повідомлень реалізовані наступним чином:

```
// Messages.js – компонент виводу повідомлень

import React from 'react';
import { connect } from  'react-redux';
import '../../styles/MessagesStyles.scss';
import MessagesList from './MessagesList';
import MessagesInput from './MessagesInput';

export default class Messages extends React.Component {
    constructor(props) {
        super(props);
    }

    render() {

        let { messages, onSubmitMessage, user_id, id } = this.props;

        return(
            <div className="messages-container">
                <MessagesList
                    messages={messages}
                    user_id={user_id}
                    id={id}
                />
                <MessagesInput
                    onSubmitMessage={onSubmitMessage}
                />
            </div>
        );
    }
}

// MessagesInput.js – компонент з елементом для вводу повідомлення

import React from 'react';

export default class MessagesInput extends React.Component {
    constructor(props) {
        super(props);
```

```
        this.createMessage = this.createMessage.bind(this);
        this.onKeyPress = this.onKeyPress.bind(this);
    }

    render() {
        return (
            <div className="messages-input">
                <button onClick={this.createMessage}>OK</button>
                <textarea ref={el => this.messageInput = el}
onKeyPress={this.onKeyPress}></textarea>
            </div>
        );
    }

    createMessage() {
        if(this.messageInput.value.trim()) {
            this.props.onSubmitMessage(this.messageInput.value);
            this.messageInput.value = "";
        }
    }

    onKeyPress(e) {
        if(e.charCode == 13 && !e.shiftKey) {
            e.preventDefault();
            this.createMessage();
        }
    }
}
```

// MessagesList.js – компонент зі списком повідомлень

```
import React from 'react';
import _ from 'lodash';
import ReactDOM from 'react-dom';

export default class MessagesList extends React.Component {
    constructor(props) {
        super(props);

        this.generateMessagesList = this.generateMessagesList.bind(this);
        this.onScroll = this.onScroll.bind(this);
        this.scroll = true;
    }

    render() {
        return (
            <ul className="messages-list" onScroll={this.onScroll}>
                {this.generateMessagesList()}
            </ul>
        );
    }

    componentWillReceiveProps(nextProps) {
        if(nextProps.id != this.props.id) {
            this.scroll = true;
        }
    }

    componentDidUpdate() {
        if(this.scroll) {
            var scrollableDomEl = ReactDOM.findDOMNode(this);
            scrollableDomEl.scrollTop = scrollableDomEl.scrollHeight;
```

```
        }
    }

    onScroll(e) {
        let { target:t } = e;
        if(t.scrollTop + t.clientHeight == t.scrollHeight) {
            this.scroll = true;
        } else {
            this.scroll = false;
        }
    }

    generateMessagesList() {
        let { messages, user_id } = this.props;
        return _.map(messages, (item, index) => (
            <li key={index} className={`message-item ${item.user_id == user_id
? 'owner' : ''}`}>
                <span className="authror">{item.user.name}</span>
                <span className="created-at">{item.created_at}</span>
                <hr />
                <code>{item.text}</code>
            </li>
        ));
    }
}
```

Функція аутентифікації користувача реалізована наступним чином:

```
// checkAuth.js - функція для аутентифікації користувача в сокетах

var http = require('http');
var config = require('./config.js');

module.exports = function (token, cb) {

    var responseData = '';

    const {path, host, port} = config;

    var request = http.get(Object.assign({}, {path, host, port}, {
        headers: {
            'Accept': 'application/json',
            'Content-Type': 'application/json',
            "X-Requested-With": "XMLHttpRequest",
            'Cookie': "x-access-token=" + token.token
        }
    }));

    try {
        request.on('response', function (response) {
            if(response.statusCode < 200 || response.statusCode > 299)
            {
                console.error(`Some undefined error. Code:
${response.statusCode}, Message: ${response.statusMessage}`);
                cb(true);
                return;
            }
            response.on('end', function () {
                cb(false, JSON.parse(responseData).data);
```

```
            });

            response.on('data', function (chunk) {
                responseData += chunk;
            });

        });

        request.on('error', function (e) {
            console.log(e.message);
        });

        request.end();

    } catch (error) {
        console.log(error);
    }
};
```

Функції шифрування та дешифрування алгоритмом RSA:

```
function RSAencrypt(s, e, m) { return bexpmod(s,e,m); }

function RSAdecrypt(m, d, p, q, u)
{
    var xp = bmodexp(bmod(m,p), bmod(d,bsub(p,[1])), p);
    var xq = bmodexp(bmod(m,q), bmod(d,bsub(q,[1])), q);

    var t=bsub(xq,xp);
    if(t.length==0)
         {
         t=bsub(xp,xq);
         t=bmod(bmul(t, u), q);
         t=bsub(q,t);
    }
    else
    {
         t=bmod(bmul(t, u), q);
    }
    return badd(bmul(t,p), xp);
}
```

Генерація закритого та відкритого ключів шифрування реалізована наступним чином:

```
var rSeed=[], Rs=[];
var Sr, Rsl, Rbits, Rbits2;
var Rx=0, Ry=0;

function r(n)
{
    return Math.floor(Math.random()*n);
}

function ror(a,n)
```

```
{
    n&=7;
    return n?((a>>n)|((a<<(8-n))&255)):a;
}

function seed(s)
{
    var n=0,nn=0;
    var x, y, t;

 while(n < s.length)
    {
        while(n<s.length && s.charCodeAt(n)<=32) n++;
        if(n < s.length) rSeed[nn]=parseInt("0x"+s.substr(n,2));
        n+=3; nn++;
    }

    Rsl=rSeed.length;
    Sr=r(256);
    Rbits=0;

    if(Rs.length==0)
    {
       for(x=0; x<256; x++) Rs[x]=x;
    }
    y=0;
    for(x=0; x<256; x++)
    {
        y=(rSeed[x] + s[x] + y) % 256;
        t=s[x]; s[x]=s[y]; s[y]=t;
    }
    Rx=Ry=0;
    alert("Random seed updated. Seed size is: "+Rsl);
}

function rc()
{
    var t;


    Rx=++Rx & 255;
    Ry=(Rs[Rx] + Ry) & 255;
    t=Rs[Rx]; Rs[Rx]=Rs[Ry]; Rs[Ry]=t;
    Sr^= Rs[(Rs[Rx] + Rs[Ry]) & 255];

Sr^= r(256);

    Sr^= ror(rSeed[r(Rsl)],r(8));
    Sr^= ror(rSeed[r(Rsl)],r(8));
    return Sr;
}

function rand(n)
{
    if(n==2)
    {
        if(!Rbits)
        {
            Rbits=8;
            Rbits2=rc(256);
        }
        Rbits--;
        var r=Rbits2 & 1;
```

```
        Rbits2>>=1;
        return r;
    }

    var m=1;

    r=0;
    while (n>m && m > 0)
    {
        m<<=8; r=(r<<8) |rc();
    }
    if(r<0) r ^= 0x80000000;
    return r % n;
}


function beq(a,b)
{
    if(a.length != b.length) return 0;

    for(var n=a.length-1; n>=0; n--)
    {
        if(a[n] != b[n]) return 0;
    }
    return 1;
}

function blshift(a,b)
{
    var n, c=0;
    var r=new Array(a.length);

    for(n=0; n<a.length; n++)
    {
        c|= (a[n]<<b);
        r[n]= c & bm;
        c>>=bs;
    }
    if(c) r[n]=c;
    return r;
}

function brshift(a)
{
    var c=0,n,cc;
    var r=new Array(a.length);

    for(n=a.length-1; n>=0; n--)
    {
        c<<=bs;
        cc=a[n];
        r[n]=(cc | c)>>1;
        c=cc & 1;
    }
    n=r.length;
    if(r[n-1]) return r;
    while(n > 1 && r[n-1]==0) n--;
    return r.slice(0,n);
}

function bgcd(uu,vv)
{
```

```
    var d, t, v=vv.concat(), u=uu.concat();
    for(;;)
    {
        d=bsub(v,u);
        if(beq(d,[0])) return u;
        if(d.length)
        {
            while((d[0] & 1) ==0) d=brshift(d);
            v=d;
        }
        else
        {
            t=v; v=u; u=t;
        }
    }
}

function rnum(bits)
{
    var n,b=1,c=0;
    var a=[];
    if(bits==0) bits=1;
    for(n=bits; n>0; n--)
    {

        if(rand(2)) a[c]|=b;
        b<<=1;
        if(b==bx2)
        {
            b=1;
            c++;
        }
    }
    return a;
}

var Primes=[3, 5, 7, 11, 13, 17, 19,
 23, 29, 31, 37, 41, 43, 47, 53,
 59, 61, 67, 71, 73, 79, 83, 89,
 97, 101, 103, 107, 109, 113, 127, 131,
 137, 139, 149, 151, 157, 163, 167, 173,
 179, 181, 191, 193, 197, 199, 211, 223,
 227, 229, 233, 239, 241, 251, 257, 263,
 269, 271, 277, 281, 283, 293, 307, 311,
 313, 317, 331, 337, 347, 349, 353, 359,
 367, 373, 379, 383, 389, 397, 401, 409,
 419, 421, 431, 433, 439, 443, 449, 457,
 461, 463, 467, 479, 487, 491, 499, 503,
 509, 521, 523, 541, 547, 557, 563, 569,
 571, 577, 587, 593, 599, 601, 607, 613,
 617, 619, 631, 641, 643, 647, 653, 659,
 661, 673, 677, 683, 691, 701, 709, 719,
 727, 733, 739, 743, 751, 757, 761, 769,
 773, 787, 797, 809, 811, 821, 823, 827,
 829, 839, 853, 857, 859, 863, 877, 881,
 883, 887, 907, 911, 919, 929, 937, 941,
 947, 953, 967, 971, 977, 983, 991, 997,
 1009, 1013, 1019, 1021];


var sieveSize=4000;
var sieve0=-1* sieveSize;
var sieve=[];
```

```
var lastPrime=0;

function nextPrime(p)
{
    var n;
    if(p == Primes[lastPrime] && lastPrime <Primes.length-1)
    {
        return Primes[++lastPrime];
    }
    if(p<Primes[Primes.length-1])
    {
        for(n=Primes.length-2; n>0; n--)
        {
            if(Primes[n] <= p)
            {
                lastPrime=n+1;
                return Primes[n+1];
            }
        }
    }
var pp,m;
p++; if((p & 1)==0) p++;
    for(;;)
    {
        if(p-sieve0 > sieveSize || p < sieve0)
        {
            for(n=sieveSize-1; n>=0; n--) sieve[n]=0;
            sieve0=p;
            primes=Primes.concat();
        }

        if(sieve[p-sieve0]==0)
        {

            for(n=0; n<primes.length; n++)
            {
                if((pp=primes[n]) && p % pp ==0)
                {
                    for(m=p-sieve0; m<sieveSize; m+=pp) sieve[m]=pp;
                    p+=2;
                    primes[n]=0;
                    break;
                }
            }
            if(n >= primes.length)
            {
                return p;
            }
        }
        else
        {
            p+=2;
        }
    }
}

function divisable(n,max)
than max, else return 0
{
    if((n[0] & 1) == 0) return 2;
    for(c=0; c<Primes.length; c++)
    {
        if(Primes[c] >= max) return 0;
```

```
            if(simplemod(n,Primes[c])==0) return Primes[c];
        }
        c=Primes[Primes.length-1];
        for(;;)
        {
            c=nextPrime(c);
            if(c >= max) return 0;
            if(simplemod(n,c)==0) return c;
        }
    }

    function bPowOf2(pow)
    {
        var r=[], n, m=Math.floor(pow/bs);
        for(n=m-1; n>=0; n--) r[n]=0;
        r[m]=1<<(pow % bs);
        return r;
    }

    function mpp(bits)
    {
        if(bits < 10) return [Primes[rand(Primes.length)]];
        if(bits <=20) return [nextPrime(rand(1<<bits))];

        var c=10, m=20, B=bits*bits/c, r, q, I, R, n, a, b, d, R2, nMinus1;

        if(bits > m*2)
        {
            for(;;)
            {
                r=Math.pow(2,Math.random()-1);
                if(bits - r * bits > m) break;
            }
        }
        else
        {
            r=0.5
        }

        q=mpp(Math.floor(r*bits)+1);
        I=bPowOf2(bits-2);
        I=bdiv(I,q).q;
        Il=I.length;
        for(;;)
        {
            R=[];
            for(n=0; n<Il; n++) R[n]=rand(bx2);
            R[Il-1] %= I[Il-1];
            R=bmod(R,I);
            if(!R[0]) R[0]|=1; // must be greater or equal to 1
            R=badd(R,I);
            n=blshift(bmul(R,q),1); // 2Rq+1
            n[0]|=1;
            if(!divisable(n,B))
            {
                a=rnum(bits-1);
                a[0]|=2;
                nMinus1=bsub(n,[1]);
                var x=bmodexp(a,nMinus1,n);
                if(beq(x,[1]))
                {
                    R2=blshift(R,1);
                    b=bsub(bmodexp(a,R2,n),[1]);
```

```
                d=bgcd(b,n);
                if(beq(d,[1])) return n;
            }
        }
    }
}


function sub2(a,b)
{
    var r=bsub(a,b);
    if(r.length==0)
    {
        this.a=bsub(b,a);
        this.sign=1;
    }
    else
    {
        this.a=r;
        this.sign=0;
    }
    return this;
}

function signedsub(a,b)
{
    if(a.sign)
    {
        if(b.sign) return sub2(b,a);
        else
        {
            this.a=badd(a,b);
            this.sign=1;
        }
    }
    else
    {
        if(b.sign)
        {
            this.a=badd(a,b);
            this.sign=0;
        }
        else return sub2(a,b);
    }
    return this;
}


function modinverse(x,n)
{
    var y=n.concat(), t, r, bq, a=[1], b=[0], ts;

    a.sign=0; b.sign=0;

    while(y.length > 1 || y[0])
    {
        t=y.concat();
        r=bdiv(x,y);
        y=r.mod;
        q=r.q;
        x=t;
        t=b.concat(); ts=b.sign;
        bq=bmul(b,q);
```

```
            bq.sign=b.sign;
            r=signedsub(a,bq);
            b=r.a; b.sign=r.sign;
            a=t; a.sign=ts;
        }

        if(x.length != 1 || x[0] != 1) return [0];

        if(a.sign)
        {
            a=bsub(n,a);
        }
        return a;
    }


    var rsa_p,rsa_q,rsa_e,rsa_d,rsa_pq, rsa_u;

    function rsaKeys(bits)
    {
        var c, p1q1;

        bits=parseInt(bits);
        rsa_q=mpp(bits);
        rsa_p=mpp(bits);
        p1q1=bmul(bsub(rsa_p,[1]),bsub(rsa_q,[1]));

        for(c=5; c<Primes.length; c++)
        {
            rsa_e=[Primes[c]];
            rsa_d=modinverse(rsa_e,p1q1);
            if(rsa_d.length != 1 || rsa_d[0]!=0) break;
        }
        rsa_pq=bmul(rsa_p,rsa_q);
        rsa_u=modinverse(rsa_p,rsa_q);

        return;
    }
```

4.2 Захист розробленого програмного забезпечення

Розроблене програмне забезпечення поширюється по вільній ліцензії. Існують різні види вільних ліцензій, зокрема, такі як Creative Commons, GNU General Public License або MIT License. Для захисту розробленого програмного забезпечення було обрано вільну ліцензію Creative Commons.

Creative Commons (CC) – це американська некомерційна організація та міжнародна мережа, яка займається доступом до освіти та розширенням діапазону творчих робіт, доступних для інших, щоб легально використовувати їх і ділитися ними. Організація випустила декілька ліцензій на авторське право, відомих як ліцензії Creative Commons, безкоштовно для громадськості. Ці ліцензії

дозволяють авторам різних робіт повідомляти, які права вони зберігають, а від яких відмовляються на користь одержувачів або інших авторів.

Ліцензії Creative Commons (CC) створені з метою сприяння обміну та використанню творчих робіт. Вони використовуються в багатьох галузях, включаючи програмне забезпечення, щоб надати авторам більше гнучкості у тому, як їхні роботи можуть бути використані.

Ці ліцензії дозволяють авторам задавати умови, за якими їхні роботи можуть бути використані, розповсюджені або змінені. Наприклад, автор може дозволити безкоштовне використання своєї роботи для некомерційних цілей, але вимагати відшкодування за комерційне використання.

Застосування ліцензії CC до програмного забезпечення може мати ряд переваг. По-перше, це може допомогти авторам програмного забезпечення захистити свої авторські права, не обмежуючи при цьому інших у використанні їхніх творів. По-друге, це може сприяти обміну та покращенню програмного забезпечення, оскільки інші розробники можуть зробити внесок в проект або використовувати його як основу для власних робіт.

Типовим прикладом ліцензії Creative Commons для програмного забезпечення є Creative Commons Attribution-NonCommercial-ShareAlike (CC BY-NC-SA), що полягає у наступному:

– Attribution (визначення авторства): треба вказати авторство відповідним чином, надати посилання на ліцензію та вказати, чи внесено зміни. Це можна зробити будь-яким прийнятним способом, але не таким чином, щоб створити враження, що ліцензіар підтримує вас або ваше використання.

– NonCommercial (некомерційне використання): не можна використовувати матеріал з комерційною метою.

– ShareAlike (збереження умов): Якщо матеріал реміксується, змінюється або на його основі створюється щось нове, необхідно розповсюджувати свій внесок на умовах тієї ж ліцензії, що й оригінал.

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

Для впровадження системи в експлуатацію необхідно:

– Встановити веб-сервер Apache або Nginx;

– Встановити та налаштувати на веб-сервері PHP 8.0;

– Встановити та налаштувати PHP 8.0 в терміналі ОС;

– Встановити та налаштувати MySQL;

– Встановити та налаштувати Redis;

– В корені проекту перейменувати файл .env.example на .env, заповнити вірними даними поля, відмічені «*» та видалити зірочки;

– Встановити Node.js та npm;

– Встановити та налаштувати Composer;

В корені проекту виконати команди:

– composer install;

– npm install;

– php artisan migrate;

– php artisan config:cache;

– php artisan route:cache;

– php artisan optimize;

– npm run prod;

– npm run sockets:prod;

Після виконання цих команд система готова до використання.

На рисунку 5.1 зображена сторінка входу в систему. Для входу потрібно бути зареєстрованим і ввести пошту та пароль.

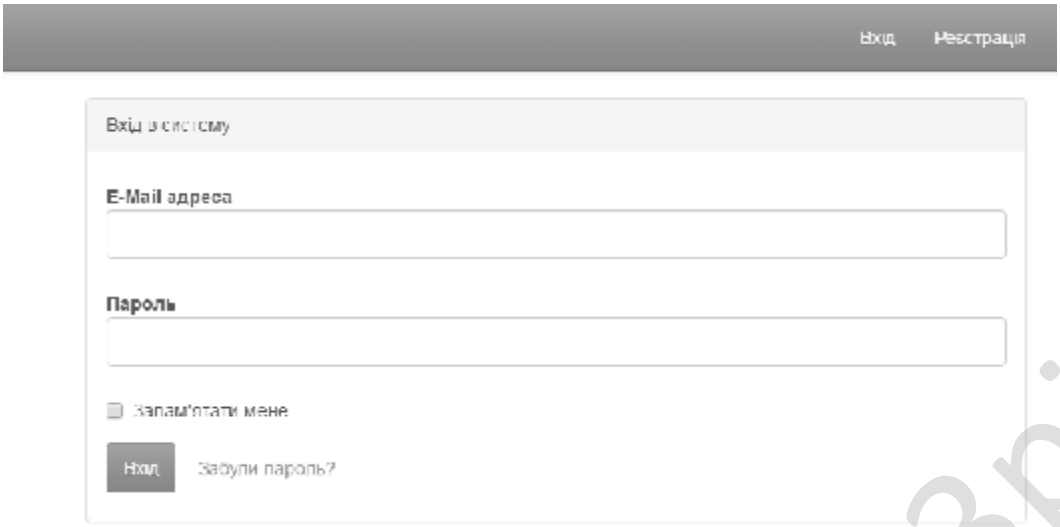

Рисунок 5.1 – Вигляд сторінки для входу в систему

На рисунку 5.2 зображена сторінка для реєстрації в системі. Для реєстрації обов'язковими полями є «Ім'я», «Електронна пошта», «Пароль» та його підтвердження. Після натискання на кнопку «Реєстрація» користувача буде успішно перенаправлено на сторінку входу в систему або будуть виведені помилки.

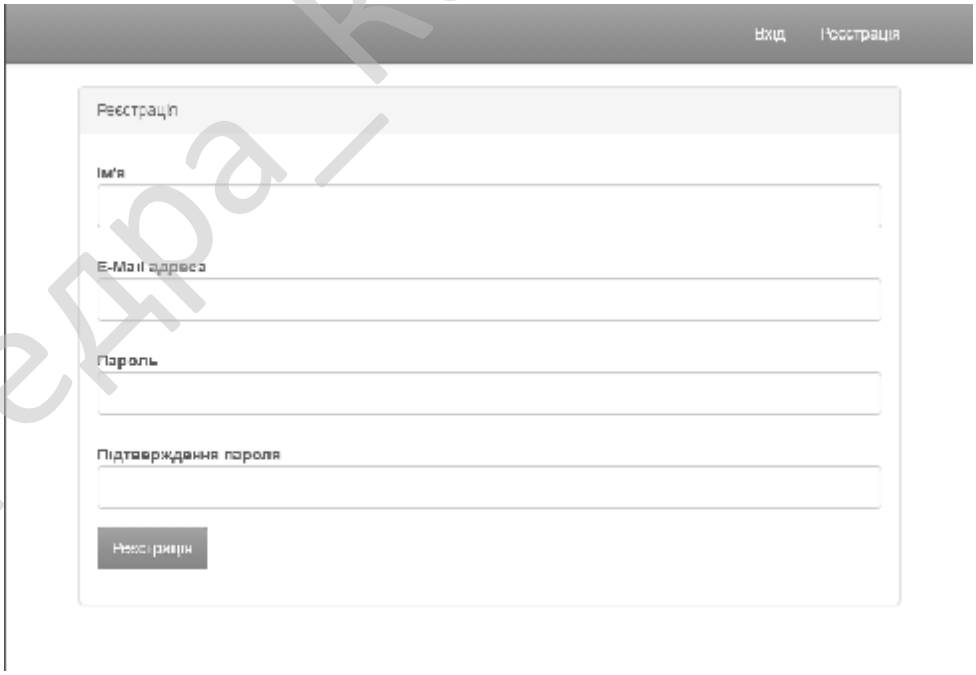

Рисунок 5.2 – Вигляд сторінки для реєстрації

На рисунку 5.3 зображена сторінка з чатом, який може бути між 2 користувачами або між учасниками певної групи. Для користувача його повідомлення мають синій задній фон, всі інші – білий. Внизу сторінки є текстове поле для написання та відправки повідомлення.

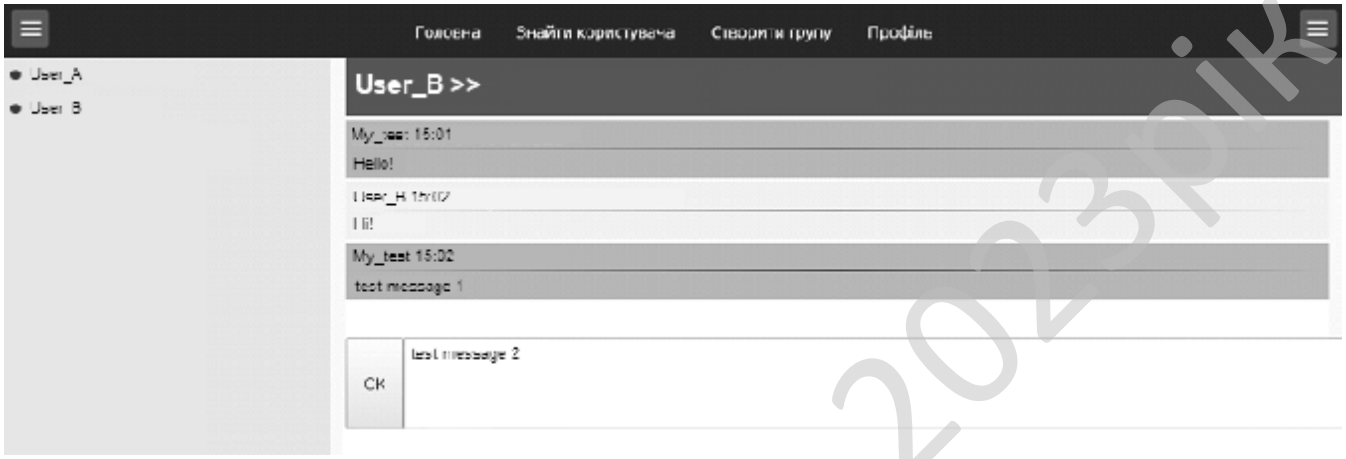

Рисунок 5.3 – Вигляд сторінки чату, захищеного шифруванням даних

6 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання кваліфікаційної роботи, призначено для системи кібербезпеки для захищеного обміну даними у мережі Інтернет.

Для вирішення поставленої мети було проведено:

– дослідження існуючих систем захищеного обміну даних для комунікування по мережі Інтернет.

– розробку алгоритмів системи кібербезпеки для захищеного обміну даними у мережі Інтернет.

– реалізацію програмного забезпечення системи кібербезпеки для захищеного обміну даними у мережі Інтернет.

Реалізовані під час виконання кваліфікаційної роботи алгоритми дозволяють успішно вирішувати завдання шифрування та передачі даних у мережі Інтернет.

Розроблене програмне забезпечення представляє собою веб-додаток, що можна використовувати практично на будь-якому сучасному комп'ютері, підключеному до інтернету, та при наявності встановленого Інтернет-браузера.

Програмне забезпечення розроблялося у об'єктно-орієнтованій парадигмі, що робить його гнучким та зручним для підтримки та масштабування.

Для розробки програмного забезпечення системи кібербезпеки захищеного обміну даними у мережі Інтернет використовувалися мови програмування JavaScript та PHP. Дані мови програмування дозволили ефективно вирішити поставлену мету та задачі кваліфікаційної роботи.

Для шифрування даних у розробленому програмному забезпеченні було використано алгоритм асиметричного шифрування RSA, що дозволило захистити як дані, так і ключі шифрування. Адже RSA дозволяє здійснювати захищений обмін ключами шифрування, розділяючи їх на відкриті та закриті ключі для

кожного користувача.

Запропоноване програмне забезпечення складається з клієнтського та серверного додатків. У п’ятому розділі пояснювальної записки надаються усі необхідні рекомендації з встановлення серверного програмного забезпечення та інструкція для використання клієнтського програмного забезпечення.

Для захисту розробленого програмного забезпечення було обрано вільну ліцензію Creative Commons.

Програмне забезпечення системи кібербезпеки для захищеного обміну даними є важливим елементом в будь-якому сучасному діловому середовищі, а також для особистого використання з метою захищеного обміну приватними даними у мережі Інтернет. Враховуючи все більшу кількість і складність кіберзагроз, використання такого роду програмного забезпечення стає не просто рекомендацією, а необхідністю. Тож, розроблене у цій кваліфікаційній роботі програмне забезпечення має важливе призначення.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Adam D. Scott Building Web Apps for Everyone. Copyright, O'Reilly Media, 2016, 245 p.

2. Aho A.V., Hopcroft J.E., Ullman J.D. Data Structures and Algorithms. – Pearson, 2001. – 620 p.

3. Alsuwaiyel, M. H. (2021). Algorithms: design techniques and analysis (Vol. 15). World Scientific.

4. Andress J. Foundations of Information Security: A Straightforward Introduction. No Starch Press, 2019.

5. Casteleyn S., Daniel F., Dolog P., Matera M. Engineering Web Applications. Berlin: Springer-Verlag, 2009, 363p.

6. Cody Lindley. Front-End Developer Handbook 2017. 2017. URL: https://frontendmasters.gitbooks.io/front-end-handbook-2017/content.

7. Cormen T.H., Leiserson C.E., Rivest R.L., Stein C. Introduction to Algorithms, 3rd Edition (The MIT Press) 3rd Edition – The MIT Press, 2019. – 1292 p.

8. Crockford, D. (2008). JavaScript: The Good Parts: The Good Parts. " O'Reilly Media, Inc.".

9. David Upton. CodeIgniter for Rapid PHP Application Development. Packt Publishing, 2007. 244 p.

10. Doglio, F., Doglio, & Corrigan. (2018). REST API Development with Node.js (Vol. 331). Apress.

11. Gusfield D. Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology 1st Edition. – Cambridge University Press, 2008. – 556 c.

12. Jain H. Data Structures & Algorithms In Go. – Hemant Jain, 2022. – 584 c.

13. Jeremy Thomas. MarkSheet. A free HTML and CSS tutorial. 2015-2017. URL: https://marksheet.io.

14. Knuth D. Art of Computer Programming, Vol. 2: Seminumerical

Algorithms 3rd Edition, Kindle Edition. – Addison-Wesley Professional, 2019. – 672 p.

15. Knuth D. The Art of Computer Programming, Vol. 1: Fundamental Algorithms, 3rd Edition 3rd Edition. – Addison-Wesley Professional, 2019. – 672 p.

16. Knuth D. The Art of Computer Programming: Vol. 3: Sorting and Searching 2nd Edition, Kindle Edition. – Addison-Wesley Professional, 2019. – 800 p.

17. Mehlhorn, K., Sanders, P., & Sanders, P. (2008). Algorithms and data structures: The basic toolbox (Vol. 55, p. 56). Berlin: Springer.

18. Oliver James. Html & CSS is hard (But it doesn't have to be). A friendly web development tutorial for complete beginners. 2017. URL: https://internetingishard.com/html-and-css.

19. Rocca La M. Advanced Algorithms and Data Structures. – Manning, 2021. – 768 p.

20. Shay Howe. Learn To Code Advanced HTML & CSS. Develop & Style Websites. 2014-2017. URL: https://learn.shayhowe.com/advanced-html-css.

21. Shay Howe. Learn To Code HTML & CSS. Develop & Style Websites. 2014-2017. URL: https://learn.shayhowe.com/html-css

22. Shklar L., Rosen Wiley R. Web Application Architecture: Principles, Protocols and Practices, 2009 – 440 p.

23. Smith R.E. Elementary Information Security, 2nd ed. Jones & Bartlett Learning, 2015.

24. Stamp M. Information Security: Principles and Practice, 2nd ed. Wiley-IEEE Press, 2011.

25. Ullman J.D., Aho A.V., Hopcroft J.E. The Design and Analysis of Computer Algorithms - International Economy Edition Paperback. – Pearson education, 1905. – 470 p.

26. Web-програмування. Частина 1 (frontend) : навч. посіб. / В. В. Босько, Л. В. Константинова, К. М. Марченко, О. С. Улічев ; М-во освіти і науки України, Центральноукраїн. нац. техн. ун-т. - Кропивницький : ЦНТУ, 2022. - 208 с.

27. Welling L., Thomson L. PHP and MySQL Web Development (4th Edition),

Addison-Wesley Professional, 2008 – 1008 p.

28. Whitman M.E. and Mattord H.J. Principles of Information Security, 6th ed. Cengage Learning, 2018.

29. Бушуєв, Р. В. Дослідження та програмна реалізація застосування об’єктно-орієнтованих баз даних в ІС : кваліфікаційна магістерська робота : спец. 123 "Комп'ютерна інженерія" / наук. кер. В. В. Босько ; Центральноукраїн. нац. тех. ун-т. - Кропивницький : ЦНТУ, 2022. - 171 с.

30. Глущук, А. І., & Баленко, О. І. (2022). Javascript як мова розробки мобільних додатків (Doctoral dissertation, ТОВ ВПП" Контраст").

31. Горовая, Н. М. (2021). Двофакторна система аутентифікації корпоративного середовища університету.

32. Демидов, З. Г. Основні види кібератак на WEB-сайти / З.Г. Демідов // Актуальні питання протидії кіберзлочинності та торгівлі людьми : зб. матеріалів Всеукр. наук.-практ. конф. (м. Харків, 15 листоп. 2017 р.) / МВС України, Харк. нац. ун-т внутр. справ; Координатор проектів ОБСЄ в Україні. – Харків : ХНУВС, 2017. – С. 131-134.

33. Куленко М.Я. Основи графічного дизайну : підручник для студентів вищих навч. закладів / Михайло Куленко; МОНУ; Київський нац. ун-т будівництва і архітектури. – 2-ге вид., виправл. та доп. – Київ : Кондор, 2007. – 492с.

34. Кучминда, Р. М. (2021). Розробка інтерфейсу веб-додатку для візуалізації результатів стороннього інтерфейсу прикладного програмування з використанням мови JavaScript (Master's thesis, ТНТУ ім. І Пулюя).

35. Матвієнко О.В., Бородкіна І.Л. Internet - технології: проектування Web-сторінки : навч. посіб. Київ : Альтерпрес, 2003, 132 с.

36. Мелешко Є.В., Якименко М.С., Поліщук Л.І. Алгоритми та структури даних: Навчальний посібник для студентів технічних спеціальностей денної та заочної форми навчання. – Кропивницький: Видавець – СПД ФО Лисенко В.Ф., 2019. – 156 с.

37. Мелешко, Є. В. Алгоритми та структури даних : навч. посіб. / Є. В. Мелешко, М. С. Якименко, Л. І. Поліщук ; М-во освіти і науки України, Центральноукраїн. нац. техн. ун-т. - Кропивницький : Лисенко В.Ф., 2019. – 156 с.

38. Методичні вказівки до виконання лабораторних робіт з дисципліни «Web-програмування» : для студ. денної та заоч. форми навч. спец. 123 «Комп'ютерна інженерія» та 125 «Кібербезпека» / уклад. Є. В. Мелешко, Л. В. Константинова ; М-во освіти і науки України, Кіровоград. нац. техн. ун-т, каф. прог. та захисту інформації. – Кропивницький : КНТУ, 2016. – 81 с.

39. Методичні рекомендації до виконання лабораторних робіт з дисципліни «Web-програмування» для студентів денної та заочної форми навчання спеціальностей 123 «Комп'ютерна інженерія», 125 «Кібербезпека» та 122 «Комп'ютерні науки» / [уклад. : Є. В. Мелешко, В. В. Босько, Л. В. Константинова] ; М-во освіти і науки України, Центральноукраїн. нац. техн. ун-т. - Кропивницький : ЦНТУ, 2023. - 87 с.

40. Микитишин А.Г., Митник М.М., Стухляк П.Д., Пасічник В.В. Компютерні мережі : навч. посіб. Львів : Магнолія, 2013, 250 с.

41. Онлайн-підручник з HTML. URL: http://www.w3schools.com/html/

42. Онлайн-підручник з JavaScript. URL: http://www.w3schools.com/js/

43. Основи захисту інформації : метод. вказ. до викон. лаб. робіт для студ. за спец. 123 "Комп'ютерна інженерія", 122 "Комп'ютерні науки"/ [уклад. : О. А. Смірнов, Є. В. Мелешко, О. К. Конопліцька-Слободенюк, В. Д. Хох, С. А. Смірнов]; М-во освіти і науки України, Центральноукраїн. нац. техн. ун-т. – Кропивницький : ЦНТУ, 2017. – 53 с.

44. Основи захисту інформації : метод. вказ. до викон. лаб. робіт для студ. за спец. 123 "Комп'ютерна інженерія", 122 "Комп'ютерні науки"/ [уклад. : О. А. Смірнов, Є. В. Мелешко, О. К. Конопліцька-Слободенюк, В. Д. Хох, С. А. Смірнов] ; М-во освіти і науки України, Центральноукраїн. нац. техн. ун-т. – Кропивницький : ЦНТУ, 2017. – 53 с.

45. Пасічник О. Г., Пасічник О. В., Стеценко І. В. Основи веб-дизайну: Навч. посіб. -К.: Вид. група BHV. 2011р. -336 с.

46. Романюк О. Н., Катєльніков Д. І., Косовець О. П. Веб-дизайн і комп'ютерна графіка. Навчальний посібник. Вінниця: ВНТУ, 2007. 142 с.

47. Смірнов О.А., Коваленко О.В., Мелешко Є.В., Константинова Л.В., Кожанова А.С. Інженерія програмного забезпечення // Навчальний посібник. – Кіровоград: Вид. КНТУ, 2012. – 409 с.

48. Тимошенко, К. О. Сучасні інструменти JS – розробника / К. О. Тимошенко, О. Є. Тесленко // Перспективні напрямки розвитку сучасних інформаційних систем та технологій : зб. тез доп. всеукр. наук.-практ. студ. конференція, 18 квіт. 2018р., м. Кропивницький. - Кропивницький : ЦНТУ, 2018. - С. 15-16.

49. Фесечко, Д. В. Дослідження та програмна реалізація веб-сайту компанії засобами фреймворку AngularJS : кваліфікаційна магістерська робота : спец. 123 «Комп'ютерна інженерія» / наук. кер. В. В. Босько ; Центральноукраїн. нац. техн. ун-т. - Кропивницький : ЦНТУ, 2021. - 142 с.

50. Цеслів О.В. WEB-програмування : навч. посібнник / О.В. Цеслів ; М-во освіти і науки, молоді та спорту України, Нац. техн. ун-т України "Київ. політехн. ін-т". – Київ : НТУУ "КПІ", 2011. – 296, с.

51. Щербакова, М. Є., & Щербаков, Є. В. (2014). Функціональні особливості JavaScript-додатків. Вісник Східноукраїнського національного університету імені Володимира Даля, (10), 142-146.

Додаток А
(обов'язковий)

Технічне завдання

Зміст

1 Найменування та область застосування

Це технічне завдання розповсюджується на розробку системи кібербезпеки для захищеного обміну даними у мережі Інтернет.

2 Підстава для розробки

Підставою для розробки служить завдання на кваліфікаційну бакалаврську роботу №13-02 від 05.01.2023 року, видане на кафедрі кібербезпеки та програмного забезпечення.

3 Мета та призначення розробки

Метою кваліфікаційної бакалаврської роботи є розробка програмного забезпечення системи кібербезпеки для захищеного обміну даними у мережі Інтернет.

4 Джерела розробки

Джерелом цієї кваліфікаційної бакалаврської роботи є стосовна 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:

– систему кібербезпеки для захищеного обміну даними у мережі Інтернет;
– 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 10/11 і з сумісними з цією платформою пристроями і прикладним програмним забезпеченням.

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 10/11.

5.8.1 Обладнання

Комп’ютер Intel Core i7/8 ГБ /1 Tb/ GeForce GT 1030 2GB або сумісні з ним.

5.8.2 Мова програмування

Програму розроблено на мовах програмування JavaScript та PHP.

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 Перелік документів, що розробляються

– Структурна схема системи.
– Функціональна схема системи.
– Діаграма процесів.
– Блок-схема алгоритму роботи програми.
– Пояснювальна записка.

8 Етапи розробки

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

8.2 Проведення досліджень або експериментальних робіт для уточнення основних положень кваліфікаційної бакалаврської роботи.

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

8.4 Побудова схем взаємодії даних.

8.5 Створення прототипу ПЗ.

8.6 Віднаходження ПЗ, аналіз отриманих результатів.

8.7 Оформлення пояснювальної записки і виконання робіт по графічній частині.

11 Порядок контролю та приймання

11.1 Подання кваліфікаційної бакалаврської роботи на попередній захист 20.05.2023 р.

11.2 Подання кваліфікаційної бакалаврської роботи на захист 12.06.2023 р.

Додаток Б

(обов’язковий)

Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

ЗАТВЕРДЖУЮ

Керівник випускної кваліфікаційної роботи

за першим (бакалаврським) рівнем вищої освіти

_____ Є.В. Мелешко

Програмне забезпечення системи кібербезпеки для захищеного обміну даними у мережі Інтернет

Лістинг програми

Код документу 12

Носій: CD/DVD-диск

Загальна кількість аркушів: 52

Літера: РП

Кропивницький – 2023 року

// ApiContr.php – Базовий клас контролерів API

```
<?php

namespace App\Http\Contrs;

use App\Models\User;
use Illuminate\Http\Request;

class ApiContr extends Contr
{
    protected $statusCode = 200;

    public function respond($data = [], $meta = [], $headers = [])
    {
        return response()->json(
            [
                'data' => $data,
                '_meta' => $meta
            ],
            $this->getStatusCode(),
            $headers
        );
    }

    public function getStatusCode()
    {
        return $this->statusCode;
    }


    public function setStatusCode($code)
    {
        $this->statusCode = $code;
        return $this;
    }

    public function user() {
        return \Auth::user();
    }
}
```

// GroupContr.php – клас відповідає за роботу клієнтського додатку з групами

```
<?php

namespace App\Http\Contrs;

use App\Events\AddedToGroupNotificationEvent;
use App\Events\UserAddedToGroupEvent;
use App\Events\UserLeavesGroupEvent;
use App\Exceptions\CustomMessageException;
use App\Models\Group;
use App\Models\User;
use Illuminate\Database\Eloquent\ModelNotFoundException;
use Illuminate\Http\Request;

class GroupContr extends ApiContr
{
    public function index(){
        $groups = $this->user()->groups()->with('users')->get();
        return $this->setStatusCode(200)->respond($groups);
    }

    public function store(Request $request) {
        $group = $this->user()->groups()->create($request->all());
        $group->load('users');
        return $this->setStatusCode(200)->respond($group);
    }

    public function show(Group $group){
        $this->authorize('getGroup', [Group::class, $group]);
        $group->load('users');
        return $this->setStatusCode(200)->respond($group);
    }

    public function update(Request $request, Group $group) {
        $this->authorize('updateGroup', [$group]);
        $group->update($request->all());
        return $this->setStatusCode(200)->respond($group);
    }

    public function destroy(Group $group) {
        $this->authorize('deleteGroup', [$group]);
        $group->delete();
        return $this->setStatusCode(200)->respond();
    }

    public function addUserToGroup(Group $group, User $user){
        $this->authorize('addUserToGroup', [$group]);
        if(!!$group->users()->find($user->id)) {
            throw new CustomMessageException('This user in already in group');
        }
        $group->users()->attach($user->id);
        event(new UserAddedToGroupEvent($group->users()->where('users.id', '!=',
$user->id)->get(['users.id']), $user, $group));
        event(new AddedToGroupNotificationEvent([$user->id], $group));
        return $this->setStatusCode(200)->respond($user);
    }

    public function userLeavesGroup(Group $group){
        if(!!$group->users()->find($this->user()->id)) {
            $group->users()->detach($this->user()->id);
            event(new UserLeavesGroupEvent($group->users()->get(['users.id']),
$group, $this->user()));
            return $this->setStatusCode(200)->respond($group);
        } else {
            throw (new ModelNotFoundException())->setModel(User::class);
        }
    }}
```

// FriendContr.php – контролер відповідає за роботу зі списком друзів

```
<?php

namespace App\Http\Contrs;

use App\Events\FriendshipCreatedEvent;
use App\Events\FriendshipDeletedEvent;
use App\Exceptions\CustomMessageException;
use App\Models\Friend;
use App\Models\User;
use Illuminate\Database\Eloquent\ModelNotFoundException;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Auth;

class FriendContr extends ApiContr
{

    public function index()
    {
        return $this->setStatusCode(200)->respond($this->user()-
>getFriends(null));
    }

    public function show(User $friend)
    {
        if($res = $this->user()->getFriendship($friend))
        {
            return $this->setStatusCode(200)->respond($res);
        }

        throw (new ModelNotFoundException())->setModel(User::class);
    }

    public function store(User $friend)
    {
        if ($friend->id == $this->user()->id) {
            throw (new ModelNotFoundException)->setModel(User::class);
        }
        $result = null;

        if ($friendship = $this->user()->getFriendship($friend, true)) {
            if ($friendship->trashed()) {
                $friendship->restore();
            } else {
                throw new CustomMessageException('You have already this user in
friends list');
            }
        } else {
            $result = new Friend([
                'sender_id' => $this->user()->id,
                'recipient_id' => $friend->id
            ]);
            $result->save();
        }

        event(new FriendshipCreatedEvent([$friend->id], $this->user()));
        return $this->setStatusCode(200)->respond($friend);
    }

    public function update(User $friend, Request $request)
    {
        if($model = $this->user()->getFriendship($friend)){
            $model->update($request->all());
            //event(new FriendshipUpdateEvent([$this->user()->id, $friend->id],
$model));
            return $this->setStatusCode(200)->respond($model);
        }
```

```
            throw (new ModelNotFoundException())->setModel(User::class);
        }

        public function destroy(User $friend)
        {
            if($res = $this->user()->getFriendship($friend)) {
                $res->delete();
                event(new FriendshipDeletedEvent([$friend->id], $this->user()));
                return $this->setStatusCode(200)->respond($friend);
            }

            throw (new ModelNotFoundException())->setModel(User::class);
        }
    }
```

// MessageContr.php – контролер відповідає за роботу з повідомленнями

```
<?php

namespace App\Http\Contrs;

use App\Events\FriendshipMessageEvent;
use App\Events\GroupMessageEvent;
use App\Models\Friend;
use App\Models\Group;
use App\Models\Message;
use App\Models\User;
use Illuminate\Database\Eloquent\ModelNotFoundException;
use Illuminate\Http\Request;

class MessageContr extends ApiContr
{
    public function friendIndex(User $friend)
    {
        if($friendship = $this->user()->getFriendship($friend)) {
            return $this->setStatusCode(200)->respond($friendship->messages()-
>with('user')->get());
        }
        throw (new ModelNotFoundException())->setModel(Friend::class);
    }

    public function friendShow(User $friend, Message $message)
    {
        $this->authorize('getFriendshipMessage', [Message::class, $friend,
$message]);
        if($this->doesMessageBelongsToModel($friend, $message)) {
            return $this->setStatusCode(200)->respond($message);
        }
        return $this->setStatusCode(404)->respond();
    }

    public function friendStore(User $friend, Request $request)
    {
        if($friendship = $this->user()->getFriendship($friend)) {
            $message = new Message();
            $message->fill($request->all());
            $message->user()->associate($this->user());
            $message->messagable()->associate($friendship);
            $message->save();
            event(new FriendshipMessageEvent([$friend->id], $message, $this-
>user()));
            return $this->setStatusCode(201)->respond($message);
        }
        throw (new ModelNotFoundException())->setModel(Friend::class);
    }

    public function friendUpdate(Friend $friend, Message $message, Request
$request)
    {
        $this->authorize('updateFriendshipMessage', [Message::class, $friend,
$message]);
        $message->update($request->all());
        event(new FriendshipMessageEvent([$friend->id, $this->user()->id],
$message, $friend));
        return $this->setStatusCode(200)->respond($message);
    }

    public function friendDestroy(Friend $friend, Message $message)
    {
        $this->authorize('destroyFriendshipMessage', [Message::class, $friend,
$message]);
        $message->delete();
        return $this->setStatusCode(200)->respond();
    }
```

```
    public function groupIndex(Group $group)
    {
        $this->authorize('getGroupMessages', [Message::class, $group]);
        return $this->setStatusCode(200)->respond($group->messages()-
>with('user')->get());
    }

    public function groupShow(Group $group, Message $message)
    {
        $this->authorize('getGroupMessage', [Message::class, $group, $message]);
        return $this->setStatusCode(200)->respond($message);
    }

    public function groupStore(Group $group, Request $request)
    {
        $this->authorize('createGroupMessage', [Message::class, $group]);
        $message = new Message($request->all());
        $message->user()->associate($this->user());
        $message->messagable()->associate($group);
        $message->save();
        event(new GroupMessageEvent($group->users()->where('users.id', '<>',
$this->user()->id)->get(['users.id']), $message, $group));
        return $this->setStatusCode(200)->respond($message);
    }

    public function groupUpdate(Group $group, Message $message, Request
$request)
    {
        $this->authorize('updateGroupMessage', [Message::class, $group,
$message]);
        $message->update($request->all());
        event(new GroupMessageEvent($group->users()->where('users.id', '<>',
$this->user()->id)->get(['users.id']), $message, $group));
        return $this->setStatusCode(200)->respond($message);
    }

    public function groupDestroy(Group $group, Message $message)
    {
        $this->authorize('destroyGroupMessage', [Message::class, $group,
$message]);
        $message->delete();
        return $this->setStatusCode(200)->respond();
    }

    public function doesMessageBelongsToModel($model, $message){
        return !!$model->messages()->find($message->id);
    }
}
```

// UserContr.php – контролер відповідає за роботу з користувачем

```
<?php

namespace App\Http\Contrs;

use App\Http\Requests\UserRequest;
use App\Models\User;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Auth;

class UserContr extends ApiContr
{

    public function index()
    {
        return $this->setStatusCode(200)->respond($this->user());
    }

    public function update(UserRequest $request)
    {
        $this->user()->update($request->all());
        return $this->setStatusCode(200)->respond($this->user());
    }

    public function search($query)
    {
        $result = User::searchUser($query)->get();

        return $this->setStatusCode(200)->respond($result);
    }
}
```

// Friend.php - модель

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\SoftDeletes;
use Illuminate\Database\Query\Builder;

class Friend extends Model
{
    use SoftDeletes;
    protected $table = 'friends';

    protected $guarded = ['id', 'created_at', 'updated_at', 'deleted_at'];


    const PENDING = 1;
    const ACCEPTED = 2;
    const DENIED = 3;
    const BLOCKED = 4;

    public function room()
    {
        return $this->morphOne(Room::class, 'roomable');
    }


    public function messages(){
        return $this->morphMany(Message::class, 'messagable');
    }


    public function sender()
    {
        return $this->belongsTo(User::class, 'sender_id');
    }

    public function recipient() {
        return $this->belongsTo(User::class, 'recipient_id');
    }

    public function scopeWhereRecipient($query, $model) {
        return $query->where('recipient_id', $model->getKey());
    }

    public function scopeOrWhereRecipient($query, $model) {
        return $query->orWhere('recipient_id', $model->getKey());
    }


    public function scopeWhereSender($query, $model) {
        return $query->where('sender_id', $model->getKey());
    }

    public function scopeOrWhereSender($query, $model) {
        return $query->orWhere('sender_id', $model->getKey());
    }

    public function scopeWhereStatus($query, $status = self::PENDING){
        return $query->where('status', $status);
    }

    public function scopeBetweenModels($query, $sender, $recipient) {
        return $query->where(function ($queryIn) use ($sender, $recipient){
            $queryIn->where(function($q) use ($sender, $recipient) {
                $q->whereSender($sender)->whereRecipient($recipient);
            })
```

```
            ->orWhere(function($q) use ($sender, $recipient) {
                $q->whereSender($recipient)->whereRecipient($sender);
            });
        });
    }

    public function getUsersAttribute(){
        return User::where('id', $this->sender_id)->orWhere('id', $this-
>recipient_id)->get();
    }
}
```

// User.php - модель

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\ModelNotFoundException;
use Illuminate\Database\Eloquent\SoftDeletes;
use Illuminate\Notifications\Notifiable;
use Illuminate\Foundation\Auth\User as Authenticatable;
use Illuminate\Support\Facades\Auth;

class User extends Authenticatable
{
    use Notifiable;
    use SoftDeletes;

    protected $guarded = ['id', 'created_at', 'updated_at', 'deleted_at'];


    protected $hidden = [
        'password', 'remember_token',
    ];


    public static function findByEmail($email)
    {
        $_this = new self;
        try {
            $user = $_this->withTrashed()->where('email', $email)-
>firstOrFail();
        } catch (ModelNotFoundException $e) {
            return null;
        }
        return $user;
    }

    public function owns($model) {
        return $this->getKey() == $model->user_id;
    }

    public function groups()
    {
        return $this->belongsToMany(Group::class);
    }

    public function feeds()
    {
        return $this->hasMany(Feed::class);
    }

    public function messages()
    {
        return $this->hasMany(Message::class);
    }

    public function friendsOfMine()
    {
        return $this->hasMany(Friend::class, 'sender_id');
    }

    public function friendOf()
    {
        return $this->hasMany(Friend::class, 'recipient_id');
    }

    public function friends()
    {
        return $this->hasMany(Friend::class, 'sender_id');
```

```
    }


    public function beFriend($recipient)
    {
        if (!$this->canBeFriend($recipient)) {
            return false;
        }

        $friendship = new Friend();
        $friendship->recipient_id = $recipient->id;
        $this->friends()->save($friendship);

        return $friendship;
    }

    public function canBeFriend($recipient)
    {
        if ($this->hasBlocked($recipient)) {
            $this->unblockFriend($recipient);
            return true;
        }

        if ($this->isBlockedBy($recipient)) {
            return false;
        }

        if ($this->areFriends($recipient)) {
            return false;
        }

        if ($friendship = $this->getFriendship($recipient)) {
            if ($friendship->status != Friend::DENIED) {
                return false;
            }
        }

        return true;
    }

    public function hasBlocked(User $recipient)
    {
        return $this->friends()->whereRecipient($recipient)-
>whereStatus(Friend::BLOCKED)->exists();
    }

    public function unblockFriend($recipient)
    {
        return $this->findFriendship($recipient)->update(['status' =>
Friend::PENDING]);
    }

    private function findFriendship(User $recipient)
    {
        return Friend::betweenModels($this, $recipient);
    }

    public function isBlockedBy(User $recipient)
    {
        return $recipient->hasBlocked($this);
    }

    public function areFriends($recipient)
    {
        return $this->findFriendship($recipient)->exists();
    }

    public function getFriendship($recipient, $wt = false)
    {
```

```
        $query = $this->findFriendship($recipient);
        if($wt) {
            $query->withTrashed();
        }
        return $query->first();
    }

    public function unFriend(User $recipient)
    {
        return $this->findFriendship($recipient)->delete();
    }

    public function hasFriendRequestFrom($recipient)
    {
        return $this->findFriendship($recipient)->whereSender($recipient)-
>whereStatus(Friend::PENDING)->exists();
    }

    public function hasSendFriendRequestTo($recipient)
    {
        return $this->findFriendship($recipient)->whereSender($this)-
>whereStatus(Friend::PENDING)->exists();
    }

    public function isFriendWith($recipient)
    {
        return $this->findFriendship($recipient)->whereStatus(Friend::ACCEPTED)-
>exists();
    }

    public function acceptFriendRequest(User $recipient)
    {
        return $this->findFriendship($recipient)->whereRecipient($this)-
>update(['status' => Friend::ACCEPTED]);
    }

    public function denyFriendRequest($recipient)
    {
        return $this->findFriendship($recipient)->whereRecipient($this)-
>update(['status' => Friend::DENIED]);
    }

    public function blockFriend($recipient)
    {
        return $this->findFriendship($recipient)->update(['status' =>
Friend::BLOCKED]);
    }

    public function getAllFriendships()
    {
        return $this->findFriendships(null)->get();
    }

    public function findFriendships($status = null)
    {
        $query = Friend::where(function($q) {
            $q->whereSender($this)
                ->orWhereRecipient($this);
        });

        if (!is_null($status)) {
            $query->whereStatus($status);
        }

        return $query;
    }

    public function getPendingFriendships()
    {
```

```
        return $this->findFriendships(Friend::PENDING)->get();
    }

    public function getAcceptedFriendships()
    {
        return $this->findFriendships(Friend::ACCEPTED)->get();
    }

    public function getDeniedFriendships()
    {
        return $this->findFriendships(Friend::DENIED)->get();
    }

    public function getBlockedFriendships()
    {
        return $this->findFriendships(Friend::BLOCKED)->get();
    }

    public function getFriendRequests()
    {
        return Friend::whereRecipient($this)->whereStatus(Friend::PENDING)-
>get();
    }

    public function getFriends($status = Friend::ACCEPTED, $columns = ['*'])
    {
        return $this->getFriendsQueryBuilder($status)->get($columns);
    }

    private function getFriendsQueryBuilder($status = Friend::ACCEPTED)
    {
        $friendships = $this->findFriendships($status)->get(['sender_id',
'recipient_id']);
        $recipients = $friendships->pluck('recipient_id')->all();
        $senders = $friendships->pluck('sender_id')->all();
        return $this->where('id', '<>', $this->getKey())->whereIn('id',
array_merge($recipients, $senders));
    }

    public function getFriendsCount()
    {
        return $this->findFriendships(Friend::ACCEPTED)->count();
    }

    public function scopeSearchUser($query, $name){
        $user = Auth::user();
        return $query->where('name', 'like', "%$name%")
            ->whereDoesntHave('friendOf', function ($q) use($user) {
                $q->where('sender_id', $user->id)->orWhere('recipient_id',
$user->id);
            })
            ->whereDoesntHave('friendsOfMine', function ($q) use ($user) {
                $q->where('sender_id', $user->id)->orWhere('recipient_id',
$user->id);
            });
    }
}
```

// FriendPolicy.php – клас для перевірки прав користувача

```
<?php

namespace App\Policies;

use App\Models\Friend;
use App\Models\User;
use Illuminate\Auth\Access\HandlesAuthorization;

class FriendPolicy
{
    use HandlesAuthorization;

    public function deleteFriendship(User $user, Friend $friend){
        return $user->id == $friend->user_id || $user->id == $friend->friend_id;
    }
}
```

// GroupPolicy.php - клас для перевірки прав групи користувачів

```
<?php

namespace App\Policies;

use App\Models\Group;
use App\Models\User;
use Illuminate\Auth\Access\HandlesAuthorization;

class GroupPolicy
{
    use HandlesAuthorization;

    public function getGroup(User $user, Group $group){
        return !!$group->users()->find($user->id);
    }

    public function updateGroup(User $user, Group $group){
        return !!$group->users()->find($user->id);
    }

    public function deleteGroup(User $user, Group $group){
        return !!$group->users()->find($user->id);
    }

    public function addUserToGroup(User $user, Group $group){
        return !!$group->users()->find($user->id);
    }
}
```

// Messages.js – компонент виводу повідомлень

```
import React from 'react';
import { connect } from  'react-redux';
import '../../styles/MessagesStyles.scss';
import MessagesList from './MessagesList';
import MessagesInput from './MessagesInput';

export default class Messages extends React.Component {
    constructor(props) {
        super(props);
    }

    render() {

        let { messages, onSubmitMessage, user_id, id } = this.props;

        return(
            <div className="messages-container">
                <MessagesList
                    messages={messages}
                    user_id={user_id}
                    id={id}
                />
                <MessagesInput
                    onSubmitMessage={onSubmitMessage}
                />
            </div>
        );
    }
}
```

// MessagesInput.js – компонент з елементом для вводу повідомлення

```
import React from 'react';

export default class MessagesInput extends React.Component {
    constructor(props) {
        super(props);

        this.createMessage = this.createMessage.bind(this);
        this.onKeyPress = this.onKeyPress.bind(this);
    }

    render() {
        return (
            <div className="messages-input">
                <button onClick={this.createMessage}>OK</button>
                <textarea ref={el => this.messageInput = el}
onKeyPress={this.onKeyPress}></textarea>
            </div>
        );
    }

    createMessage() {
        if(this.messageInput.value.trim()) {
            this.props.onSubmitMessage(this.messageInput.value);
            this.messageInput.value = "";
        }
    }

    onKeyPress(e) {
        if(e.charCode == 13 && !e.shiftKey) {
            e.preventDefault();
            this.createMessage();
        }
    }
}
```

// MessagesList.js – компонент з списком повідомлень

```
import React from 'react';
import _ from 'lodash';
import ReactDOM from 'react-dom';

export default class MessagesList extends React.Component {
    constructor(props) {
        super(props);

        this.generateMessagesList = this.generateMessagesList.bind(this);
        this.onScroll = this.onScroll.bind(this);
        this.scroll = true;
    }

    render() {
        return (
            <ul className="messages-list" onScroll={this.onScroll}>
                {this.generateMessagesList()}
            </ul>
        );
    }

    componentWillReceiveProps(nextProps) {
        if(nextProps.id != this.props.id) {
            this.scroll = true;
        }
    }

    componentDidUpdate() {
        if(this.scroll) {
            var scrollableDomEl = ReactDOM.findDOMNode(this);
            scrollableDomEl.scrollTop = scrollableDomEl.scrollHeight;
        }
    }

    onScroll(e) {
        let { target:t } = e;
        if(t.scrollTop + t.clientHeight == t.scrollHeight) {
            this.scroll = true;
        } else {
            this.scroll = false;
        }
    }

    generateMessagesList() {
        let { messages, user_id } = this.props;
        return _.map(messages, (item, index) => (
            <li key={index} className={`message-item ${item.user_id == user_id ?
'owner' : ''}`}>
                <span className="authror">{item.user.name}</span>
                <span className="created-at">{item.created_at}</span>
                <hr />
                <code>{item.text}</code>
            </li>
        ));
    }
}
```

// UserProfileContainer.js – компонент профілю користувача

```
import React from 'react';

import {connect} from 'react-redux';
import {bindActionCreators} from 'redux';
import { updateCurrentUser } from '../../actions/currentUserActions';
import UserProfileForm from './UserProfileForm';
import Container from '../ContentContainer';

import '../../styles/ProfileStyles.scss';

class Profile extends React.Component {
    constructor(props) {
        super(props);

        this.saveUser = this.saveUser.bind(this);
    }

    render() {
        return (
            <Container left={true}>
                <UserProfileForm
                    user={this.props.user}
                    saveUser={this.saveUser}
                    errors={this.props.errors}
                />
            </Container>
        )
    }

    saveUser(user) {
        this.props.currentUserActions.updateCurrentUser(user);
    }

}

export default connect(
    state => ({
        user: state['currentUserReducer']['user'],
        errors: state['currentUserReducer']['errors']
    }),
    dispatch => ({
        currentUserActions: bindActionCreators({updateCurrentUser}, dispatch)
    })
)
(Profile);
```

// UserProfileForm.js – форма профілю користувача

```
import React from 'react';

export default class UserProfileForm extends React.Component {
    constructor(props) {
        super(props);
        this.state = {user: {}, errors: {}};
        if (this.props.user.id)
            this.state.user = this.props.user;

        this.handleFormSubmit = this.handleFormSubmit.bind(this);
        this.handleInputChange = this.handleInputChange.bind(this);
    }

    componentWillReceiveProps(nextProps) { console.log(nextProps);
        this.setState({
            user: nextProps.user,
            errors: nextProps.errors
        });
    }

    handleInputChange({target}) {
        if(!!this.state.errors[target.name]) {
            let errors = {...this.state.errors};
            delete errors[target.name];
            this.setState({user: { ...this.state.user,
                [target.name]: target.value
            }, errors: errors });
        } else {
            this.setState({user: { ...this.state.user, [target.name]:
target.value}});
        }
    }

    handleFormSubmit(e) {
        e.preventDefault();
        const { user } = this.state;
        var errors = {};
        if(user.name == '') errors.name = 'Can\'t be empty';
        if(user.email.search(/^\w+@\w+\.\w+$/gi) == -1) errors.email = 'Invalid
email';
        if(user.password || user.password_confirmation) {
            user.password = user.password || '';
            user.password_confirmation = user.password_confirmation || '';
            if(user.password.length < 1 || user.password_confirmation.length <
1) {
                errors.password = 'Password and password repeat should be equal
or empty';
            } else if (user.password.length < 5 ||
user.password_confirmation.length < 5){
                errors.password = 'Password must be 6 and more characters
length';
            }
        }

        if(Object.keys(errors).length == 0)
            this.props.saveUser(this.state.user);
        else
            this.setState({errors});
    }

    render() {
        const {name, email, password_confirmation = '', password = ''} =
this.state.user;

        return (
            <div id="profile-block">
                <form onSubmit={this.handleFormSubmit}>
```

```
                    <div className="form-group">
                        <label htmlFor="user-name">Name</label>
                        <input type="text" id="user-name" name="name"
placeholder="Name..." value={name}
                               onChange={this.handleInputChange}/>
                        <span className="validation-
error">{this.state.errors.name}</span>
                    </div>
                    <div className="form-group">
                        <label htmlFor="user-email">Email</label>
                        <input type="text" id="user-email" name="email"
placeholder="Email..." value={email}
                               onChange={this.handleInputChange}/>
                        <span className="validation-
error">{this.state.errors.email}</span>
                    </div>
                    <div className="form-group">
                        <label htmlFor="user-password">Passowrd</label>
                        <input type="password" id="user-password"
name="password"  placeholder="Password..." value={password}
onChange={this.handleInputChange}/>
                        <span className="validation-
error">{this.state.errors.password}</span>
                    </div>
                    <div className="form-group">
                        <label htmlFor="user-password-repeat">Repeat
password</label>
                        <input type="password" id="user-password-repeat"
name="password_confirmation" value={password_confirmation} placeholder="Repeat
password" onChange={this.handleInputChange}/>
                        <span className="validation-
error">{this.state.errors.password}</span>
                    </div>
                    <div className="form-group">
                        <button type="submit" className="btn btn-
success">Save</button>
                    </div>

                </form>
            </div>
        );
    }
}

UserProfileForm.errorsStyles = {
    color: 'red'
};
```

// SearchUserContainer.js – компонент пошуку користувачів

```
import React from 'react';
import {connect} from 'react-redux';
import {bindActionCreators} from 'redux';

import { fetchUsersByFilterString, addUserToFriendsById } from
'../../actions/usersActions';

import SearchUserPresentation from './SearchUserPresentation';
import Container from '../ContentContainer';

class SearchUserContainer extends React.Component {
    constructor(props) {
        super(props);

        this.handleSearchUsers = this.handleSearchUsers.bind(this);
        this.addUserRequest = this.addUserRequest.bind(this);
    }

    handleSearchUsers(query) {
        this.props.findUsersActions.fetchUsersByFilterString(query);
    }

    addUserRequest(id) {
        this.props.findUsersActions.addUserToFriendsById(id);
    }

    render() {
        return (<Container left={true}>
            <SearchUserPresentation
                addUserRequest={this.addUserRequest}
                users={this.props.users}
                searchString={this.props.searchFilter}
                searchUsers={this.handleSearchUsers}/>
        </Container>);
    }
}

export default connect(
    state => ({
        searchFilter: state['usersReducer']['searchFilter'],
        users: state['usersReducer']['searchedUsers']
    }),
    dispatch => ({
        findUsersActions: bindActionCreators({ fetchUsersByFilterString,
addUserToFriendsById }, dispatch)
    })
)(SearchUserContainer);
```

// SearchUserPresentation.js – презентаційний компонент пошуку користувачів

```
import React from 'react';
import {connect} from 'react-redux';
import {bindActionCreators} from 'redux';

import { fetchUsersByFilterString, addUserToFriendsById } from
'../../actions/usersActions';

import SearchUserPresentation from './SearchUserPresentation';
import Container from '../ContentContainer';

class SearchUserContainer extends React.Component {
    constructor(props) {
        super(props);

        this.handleSearchUsers = this.handleSearchUsers.bind(this);
        this.addUserRequest = this.addUserRequest.bind(this);
    }

    handleSearchUsers(query) {
        this.props.findUsersActions.fetchUsersByFilterString(query);
    }

    addUserRequest(id) {
        this.props.findUsersActions.addUserToFriendsById(id);
    }

    render() {
        return (<Container left={true}>
            <SearchUserPresentation
                addUserRequest={this.addUserRequest}
                users={this.props.users}
                searchString={this.props.searchFilter}
                searchUsers={this.handleSearchUsers}/>
        </Container>);
    }
}

export default connect(
    state => ({
        searchFilter: state['usersReducer']['searchFilter'],
        users: state['usersReducer']['searchedUsers']
    }),
    dispatch => ({
        findUsersActions: bindActionCreators({ fetchUsersByFilterString,
addUserToFriendsById }, dispatch)
    })
)(SearchUserContainer);
```

```
// base.service.js – базовий сервіс для відправки запитів

class BaseService {
    constructor(props) {

    }

    transformFriendshipModels(users, store) {
        const { id } = store.currentUser.user;
        return _.map(users, user => this.transformFriendshipModel(user, id));
    }

    transformFriendshipModel(user, id) {
        user.user = user.sender.id != id ? user.sender : user.recipient;
        delete user.sender; delete user.recipient;
        return user;
    }

    beforeRequest() {
        nprogress.start();
    }
}

export default BaseService;
```

```
// friend.service.js – сервіс для роботи зі списком друзів

import axios from 'axios';

import BaseService from './base.service';

class FriendshipService extends BaseService {
    constructor(props) {
        super(props);
    }

    getFriends() {
        return axios.get('api/friend');
    }
    getFriend(id) {
        return axios.get(`api/friend/${id}`);
    }

    storeFriend(id) {
        return axios.post(`api/friend/${id}`);
    }

    updateFriend(id, data = {}) {
        return axios.put(`api/friend/${id}`, data);
    }

    destroyFriend(id) {
        return axios.delete(`api/friend/${id}`);
    }

    loadFriendMessages(id) {
        return axios.get(`api/friend/${id}/message`);
    }


    createFriendMessage(id, text) {
        return axios.post(`api/friend/${id}/message`, {text: text});
    }


}

export default new FriendshipService();
```

// group.service.js – сервіс для роботи з групами

```
import axios from 'axios';

import BaseService from './base.service';

class GroupService extends BaseService {
    constructor(props) {
        super(props);
    }

    getGroups() {
        return axios.get(`api/group`);
    }

    getGroup(id) {
        return axios.get(`api/group/${id}`);
    }


    storeGroup(data) {
        return axios.post(`api/group`, data);
    }


    updateGroup(id, data = {}) {
        return axios.put(`api/group/${id}`, data);
    }

    destroyGroup(id) {
        return axios.delete(`api/group/${id}`);
    }

    loadGroupMessages(id) {
        return axios.get(`api/group/${id}/message`);
    }

    leaveGroup(id) {
        return axios.get(`api/group/${id}/leave`);
    }

    addUserToGroupById(group_id, user_id) {
        return axios.get(`api/group/${group_id}/adduser/${user_id}`);
    }
}

export default new GroupService();
```

// message.service.js – сервіс для роботи з повідомленнями

```
import axios from 'axios';

import BaseService from './base.service';

class MessageService extends BaseService {
    constructor(props) {
        super(props);
    }

    getFriendshipMessages(id) {
        return axios.get(`api/friend/${id}/message`);
    }


    getFriendshipMessage(id, messageId) {
        return axios.get(`api/friend/${id}/message/${messageId}`);
    }

    storeFriendshipMessage(id, data) {
        return axios.post(`api/friend/${id}/message`, data);
    }

    updateFriendshipMessage(id, messageId, data) {
        return axios.put(`api/friend/${id}/message/${messageId}`, data);
    }

    destroyFriendshipMessage(id, messageId) {
        return axios.delete(`api/friend/${id}/message/${messageId}`);
    }

    getGroupMessages(id) {
        return axios.get(`api/group/${id}/message`);
    }

    getGroupMessage(id, messageId) {
        return axios.get(`api/group/${id}/message/${messageId}`);
    }

    storeGroupMessage(id, data) {
        return axios.post(`api/group/${id}/message`, data);
    }


    updateGroupMessage(id, messageId, data) {
        return axios.put(`api/group/${id}/message/${messageId}`, data);
    }

    destroyGroupMessage(id, messageId) {
        return axios.delete(`api/group/${id}/message/${messageId}`);
    }
}

export default new MessageService();
```

```
// user.service.js – сервіс для роботи з користувачами

import axios from 'axios';

import BaseService from './base.service';

class UserService extends BaseService{
    constructor(props) {
        super(props);
    }

    fetchCurrentUser() {
        return axios.get('api/user');
    }

    updateCurrentUser(data) {
        return axios.put('api/user', data);
    }

    findUsers(string) {
        return axios.get(`api/user/${string}`);
    }
}

export default new UserService();
```

// App.js – початковий компонент клієнтського додатку додатку

```
import React from 'react';

import { connect } from 'react-redux';

import socketClient from './socketClient';

import './styles/AppResets.scss';
import './styles/AppStyles.scss';

import Header from './components/Header/HeaderContainer';
import LeftSidebar from './components/Sidebars/LeftSidebar';


class App extends React.Component {

    constructor(props) {
        super(props);
    }

    componentWillUnmount() {
        socketClient.disconnect();
    }

    render() {
        return (<div className="wrapper">
            <Header />
            <LeftSidebar />
            {this.props.children}
        </div>);
    }
}

export default connect()(App);
```

// index.js – точка входу в клієнтський додаток

```
import React from "react";
import { render } from "react-dom";
import { Redirect, IndexRoute, Route, Router, IndexRedirect, hashHistory } from
'react-router';
import { syncHistoryWithStore } from 'react-router-redux';
import '../../../node_modules/nprogress/nprogress.css';

require('promise.prototype.finally').shim();

window.nprogress = require('nprogress');
nprogress.configure({ showSpinner: false });
import { fetchCurrentUser } from './actions/currentUserActions';
import { loadFriendMessages } from './actions/friendsActions';
import { loadGroupMessages } from './actions/groupsActions';
import { loadFeeds } from './actions/feedsActions';

import socketClient from './socketClient';
//import Room from './webrtc/Room';

import configureStore from './store/configurateStore.js';
import { Provider } from 'react-redux';

import App from './App';
import Profile from './components/UserProfile/UserProfileContainer';
import SearchUsers from './components/UserSearch/SearchUserContainer';
import Feeds from './components/Feeds/FeedsContainer';
import Friend from './components/Friend/FriendContainer';
import Group from './components/Group/GroupContainer';
import CreateGroup from './components/GroupCreate/GroupCreate';

const store = configureStore({});
const history = syncHistoryWithStore(hashHistory, store);

store.dispatch(fetchCurrentUser());
socketClient.configurateStore(store);
socketClient.connect();

render((
    <Provider store={store}>
        <Router history={history}>
            <Route path="/" component={App}>
                <IndexRoute component={Feeds} onEnter={nextState =>
store.dispatch(loadFeeds())}/>
                <Route path="/user" component={Profile} />
                <Route path="/user/find" component={SearchUsers} />
                <Route path="/friend/:id" component={Friend}
onEnter={(nextState) =>
store.dispatch(loadFriendMessages(nextState.params.id))}/>
                <Route path="/group/:id" component={Group} onEnter={nextState =>
store.dispatch(loadGroupMessages(nextState.params.id))}/>
                <Route path="/creategroup" component={CreateGroup}/>
            </Route>
            <Redirect from="*" to="/" />
        </Router>
    </Provider>),
    document.getElementById('app'));
```

// socketClient.js – сокет-клієнт

```
import io from 'socket.io-client';
import cookie from 'react-cookie';
import { push } from 'react-router-redux';

import * as friendsAction from './constants/friendshipsActionsConst';
import * as groupsAction from './constants/groupsActionsConst';
import * as usersAction from './constants/usersActionsConst';

import { socket_queryOnlineUsers, friendshipCreated } from
'./actions/friendsActions';

export default new function () {
    this._socket = null;
    const self = this;
    this.connect = function () {
        if(self._socket) return self._socket;

        return self._socket = new Promise((resolve, reject) => {
            let s = io('http://localhost:3000/');

            s.on('connect', () => {
                console.log('Socket connected');

                s.emit('authorize', {token: self._getCookie('x-access-token')});

                s.on('logged', () => {
                    this.registerSocketEvents(s);
                    s.on('pong', () => {console.log('pong');});
                    return resolve(s);
                });

                s.on('reconnect', () => {
                    self._dispatch(socket_queryOnlineUsers());
                });

                s.on('disconnect', () => {
                    s.connect();
                    return reject(s);
                });
                s.on('connect_error', () => {
                    return reject(s);
                });
                s.on('error', () => {
                    return reject(s);
                });
            });
        });
    };

    this.emit = (event, data) => {
        return this.connect().then(socket => {
            return new Promise((resolve, reject) => {
                return socket.emit(event, data, (response) => {
                    if (response) {
                        if (response.error) {
                            console.error(response.error);
                            return reject(response.error);
                        }
                        return resolve(response);
                    }

                    return resolve();
                });
            });
        })
            .catch(error => console.log('error in socketClient: ', error));
```

```
    };

    this.disconnect = function () {
        if (this._socket.connected) {
            this._socket.disconnect();
        }
        this._socket = null;
    };

    this._getCookie = function (name) {
        return cookie.load(name);
    };

    this.configurateStore = ({dispatch, getState}) => {
        self._dispatch = dispatch;
        self._getState = getState;
    };

    this.registerSocketEvents = (s) => {
        s.on('userStatusUpdated', data => {
            self._dispatch({type: friendsAction.socket_userStatusChanged,
payload: data});
        });

        s.on('ne', m => {
            const { friend, message, group, user, users } = m.data;
            switch (m.event){
                case 'App\\Events\\FriendshipMessageEvent':
                    message.user = friend;
                    this._dispatch({type:
friendsAction.createFriendMessageSuccess, payload: {id: friend.id, res:
message}});
                    break;
                case 'App\\Events\\FriendshipDeletedEvent':
                    this._dispatch({type:
friendsAction.deleteUserFromFriendsSuccess, payload: friend});
                    console.log(this._getState());
                    if
(this._getState().routing.locationBeforeTransitions.pathname ==
`/friend/${friend.id}`) {
                        this._dispatch(push('/'));
                    }
                    break;
                case 'App\\Events\\FriendshipCreatedEvent':
                    this._dispatch(friendshipCreated(friend));
                    break;

                case 'App\\Events\\UserAddedToGroupEvent':
                    this._dispatch({type: groupsAction.addUserToGroupSuccess,
payload: {id: group.id, res: user}});
                    break;
                case 'App\\Events\\UserLeavesGroupEvent':
                    this._dispatch({type:
groupsAction.socket_userLeftGroupNotification, payload: {user, group}});
                    break;
                case 'App\\Events\\AddedToGroupNotificationEvent':
                    this._dispatch({type:
groupsAction.socket_addedToGroupNotification, payload: {group, users}});
                    break;
                case 'App\\Events\\GroupMessageEvent':
                    this._dispatch({type:
groupsAction.createGroupMessageSuccess, payload: {id: group.id, res: message}});
                    break;
            }
        });
    }
};
```

// main.blade.php – шаблон рендерингу сторінки з додатком

```
<!doctype html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
    @if(env('APP_DEBUG'))
        <link href="assets/preloader.css?v={{time()}}" type="text/css"
rel="prefetch" onload="this.rel='stylesheet'">
    @else
        <link href="assets/preloader.css?v={{env('APP_VER')}}" type="text/css"
rel="prefetch" onload="this.rel='stylesheet'">
    @endif
    <meta name="viewport" content="width=device-width, initial-scale=1.0,
maximum-scale=1.0, user-scalable=no">
</head>
<body>
<div id="app">
    <div id="loader-wrapper">
        <div id="loader"></div>

        <div class="loader-section section-left"></div>
        <div class="loader-section section-right"></div>
        <style type="text/css">
            body {
                background-color: #222222;
            }
        </style>
    </div>
</div>
@if(env('APP_DEBUG'))
    <script type="text/javascript" src="assets/bundle.js?v={{time()}}"></script>
@else
    <script type="text/javascript"
src="assets/bundle.js?v={{env('APP_ENV')}}"></script>
@endif
</body>
</html>
```

// server/index.js – модуль сокет-серверу

```
var config = require('./config');
var io = require('socket.io');
var Redis = require('ioredis');
var redis = new Redis();

var socketEvents = require('./socketEvents');
var users = require('./usersContainer');
var checkAuth = require('./checkAuth');

console.log("Starting sockets server...");

io = io.listen(config.socketPort || 3000);
console.log('Socket server listens on socket: ' + config.socketPort);

try {
    redis.subscribe('general', function (err, count) {
    });

    console.log('Subscribed to general channel');
} catch (e) {
    console.log('Error happened while subscribing to general channel');
    console.log(e.message);
}

redis.on('message', function (channel, message) {
    message = JSON.parse(message);
    if (channel == 'general') {

    } else {
        id = parseInt(channel);

        if (!!users.getUserSockets(id)) {
            users.getUserSockets(id).forEach(function (item) {
                io.sockets.sockets[item].emit('ne', {channel, data:
message.data, event: message.event});
            });
        }
    }
});

io.on('connection', function (socket) {
    console.log('New user connected');
    socket.on('authorize', function (loginData) {
        if (!!loginData.token) {
            checkAuth(loginData, function (error, user) {
                if (error) {
                    socket.emit('auth-error', {code: 401, type:
'Unauthorized'});
                    socket.disconnect();
                    return;
                }
                console.log('user id: ' + user.id + " authorized");
                users.add(user.id, socket.id);
                socket.emit('logged', {payload: true});
                socket.broadcast.emit('userStatusUpdated', {id: user.id, status:
true});
                redis.subscribe(user.id);

                socketEvents(socket, user.id);

                socket.on('disconnect', function () {
                    console.log('user id: ' + user.id + ' disconnected');
                    redis.unsubscribe(user.id);
                    socket.broadcast.emit('userStatusUpdated', {id: user.id,
status: false});
                    users.delete(socket.id);
                });
```

```
            });
        } else {
            socket.on('disconnect', function () {
                console.log('Anauthorized socket disconnected');
            });
        }
    });
});
```

// checkAuth.js – функція для аутентифікації користувача в сокетах

```
var http = require('http');
var config = require('./config.js');

module.exports = function (token, cb) {

    var responseData = '';

    const {path, host, port} = config;

    var request = http.get(Object.assign({}, {path, host, port}, {
        headers: {
            'Accept': 'application/json',
            'Content-Type': 'application/json',
            "X-Requested-With": "XMLHttpRequest",
            'Cookie': "x-access-token=" + token.token
        }
    }));

    try {
        request.on('response', function (response) {
            if(response.statusCode < 200 || response.statusCode > 299)
            {
                console.error(`Some undefined error. Code:
${response.statusCode}, Message: ${response.statusMessage}`);
                cb(true);
                return;
            }
            response.on('end', function () {
                cb(false, JSON.parse(responseData).data);
            });

            response.on('data', function (chunk) {
                responseData += chunk;
            });

        });

        request.on('error', function (e) {
            console.log(e.message);
        });

        request.end();

    } catch (error) {
        console.log(error);
    }
};
```

// socketEvents.js – функція для реєстрації обробників подій

```
var usersContainer = require('./usersContainer');

module.exports = function (socket, socketUserId) {
    socket.on('queryOnlineUsers', (request, callback) => {
        if (callback && request) {
            callback(request.map(id => {
                return {id: id, status: !!usersContainer._UserSockets[id]};
            }));
        }
    });

    socket.on('offer', (data) => {
        let conn = usersContainer.getUserSockets(data.id);
        conn.forEach((item) => item.emit('offer', {data: data.offer, id:
data.id}));
    });

    socket.on('answer', (data) => {
        let conn = usersContainer.getUserSockets(data.id);
        conn.forEach(item => item.emit('answer', {data: data.answer, id:
data.id}));
    });

    socket.on('candidate', (data) => {
        let conn = usersContainer.getUserSockets(data.id);
        conn.forEach(item => item.emit('candidate', {data: data.candidate, id:
data.id}));
    });

    socket.on('leave', (data) => {
        let conn = usersContainer.getUserSockets(data.id);
        conn.forEach(item => item.emit('leave', {id:
usersContainer.getUserId(socket.id)}));
    });
};
```

// usersContainer.js – сінглтон об'єкт для збереження користувачів і їх сокетів

```
var _ = require('lodash');

module.exports = new function () {
    this._UserSockets = {};
    this._SocketUser = {};

    this.add = function (userId, socketId, cb) {
        this._UserSockets[userId] !== undefined
            ? this._UserSockets[userId].push(socketId)
            : this._UserSockets[userId] = [socketId];

        this._SocketUser[socketId] = userId;
    };

    this.delete = function(userId) {
        delete this._UserSockets[this._SocketUser[userId]];
        delete this._SocketUser[userId];
    };
    this.remove = this.delete;

    this.getSocketIds = function(userId) {
        return this._UserSockets[userId] || [];
    };

    this.getUserSockets = function(userId) {
        return this._UserSockets[userId] || [];
    };

    this.getUserId = function(socketId) {
        return this._SocketUser[socketId];
    };
};
```

// rsa.js - шифрування/дешифрування RSA

```
var bs=28;
var bx2=1<<bs, bm=bx2-1, bx=bx2>>1, bd=bs>>1, bdm=(1<<bd)-1;

var log2=Math.log(2);

function zeros(n)
{
 var r=new Array(n);

 while(n-->0) r[n]=0;
 return r;
}

function zclip(r)
{
 var n = r.length;
 if(r[n-1]) return r;
 while(n>1 && r[n-1]==0) n--;
 return r.slice(0,n);
}

function nbits(x)
{
  var n = 1, t;
  if((t=x>>>16) != 0) { x = t; n += 16; }
  if((t=x>>8) != 0) { x = t; n += 8; }
  if((t=x>>4) != 0) { x = t; n += 4; }
  if((t=x>>2) != 0) { x = t; n += 2; }
  if((t=x>>1) != 0) { x = t; n += 1; }
  return n;
}

function badd(a,b)
{
 var al=a.length;
 var bl=b.length;

 if(al < bl) return badd(b,a);

 var r=new Array(al);
 var c=0, n=0;

 for(; n<bl; n++)
 {
  c+=a[n]+b[n];
  r[n]=c & bm;
  c>>>=bs;
 }
 for(; n<al; n++)
 {
  c+=a[n];
  r[n]=c & bm;
  c>>>=bs;
 }
 if(c) r[n]=c;
 return r;
}

function bsub(a,b)
{
 var al=a.length;
 var bl=b.length;

 if(bl > al) return [];
 if(bl == al)
```

```
 {
  if(b[bl-1] > a[bl-1]) return [];
  if(bl==1) return [a[0]-b[0]];
 }

 var r=new Array(al);
 var c=0;

 for(var n=0; n<bl; n++)
 {
  c+=a[n]-b[n];
  r[n]=c & bm;
  c>>=bs;
 }
 for(;n<al; n++)
 {
  c+=a[n];
  r[n]=c & bm;
  c>>=bs;
 }
 if(c) return [];

 return zclip(r);
}

function ip(w, n, x, y, c)
{
 var xl = x&bdm;
 var xh = x>>bd;

 var yl = y&bdm;
 var yh = y>>bd;

 var m = xh*yl+yh*xl;
 var l = xl*yl+((m&bdm)<<bd)+w[n]+c;
 w[n] = l&bm;
 c = xh*yh+(m>>bd)+(l>>bs);
 return c;
}

function bsqr(x)
{
 var t = x.length;
 var n = 2*t;
 var r = zeros(n);
 var c = 0;
 var i, j;

 for(i = 0; i < t; i++)
 {
  c = ip(r,2*i,x[i],x[i],0);
  for(j = i+1; j < t; j++)
  {
   c = ip(r,i+j,2*x[j],x[i],c);
  }
  r[i+t] = c;
 }

 return zclip(r);
}


function bmul(x,y)
{
 var n = x.length;
 var t = y.length;
 var r = zeros(n+t-1);
 var c, i, j;
```

```
 for(i = 0; i < t; i++)
 {
  c = 0;
  for(j = 0; j < n; j++)
  {
   c = ip(r,i+j,x[j],y[i],c);
  }
  r[i+n] = c;
 }

 return zclip(r);
}

function toppart(x,start,len)
{
 var n=0;
 while(start >= 0 && len-->0) n=n*bx2+x[start--];
 return n;
}


function bdiv(a,b)
{
 var n=a.length-1;
 var t=b.length-1;
 var nmt=n-t;


 if(n < t || n==t && (a[n]<b[n] || n>0 && a[n]==b[n] && a[n-1]<b[n-1]))
 {
  this.q=[0]; this.mod=a;
  return this;
 }

if(n==t && toppart(a,t,2)/toppart(b,t,2) <4)
 {
  var x=a.concat();
  var qq=0;
  var xx;
  for(;;)
  {
   xx=bsub(x,b);
   if(xx.length==0) break;
   x=xx; qq++;
  }
  this.q=[qq]; this.mod=x;
  return this;
 }

 var shift2=Math.floor(Math.log(b[t])/log2)+1;
 var shift=bs-shift2;

 var x=a.concat();
 var y=b.concat();

 if(shift)
 {
  for(i=t; i>0; i--) y[i]=((y[i]<<shift) & bm) | (y[i-1] >> shift2);
  y[0]=(y[0]<<shift) & bm;
  if(x[n] & ((bm <<shift2) & bm))
  {
   x[++n]=0; nmt++;
  }
  for(i=n; i>0; i--) x[i]=((x[i]<<shift) & bm) | (x[i-1] >> shift2);
  x[0]=(x[0]<<shift) & bm;
 }

 var i, j, x2;
 var q=zeros(nmt+1);
```

```
 var y2=zeros(nmt).concat(y);
 for(;;)
 {
  x2=bsub(x,y2);
  if(x2.length==0) break;
  q[nmt]++;
  x=x2;
 }

 var yt=y[t], top=toppart(y,t,2)
 for(i=n; i>t; i--)
 {
  var m=i-t-1;
  if(i >= x.length) q[m]=1;
  else if(x[i] == yt) q[m]=bm;
  else q[m]=Math.floor(toppart(x,i,2)/yt);

  var topx=toppart(x,i,3);
  while(q[m] * top > topx) q[m]--;

    y2=y2.slice(1);
  x2=bsub(x,bmul([q[m]],y2));
  if(x2.length==0)
  {
   q[m]--;
   x2=bsub(x,bmul([q[m]],y2));
  }
  x=x2;
 }
if(shift)
 {
  for(i=0; i<x.length-1; i++) x[i]=(x[i]>>shift) | ((x[i+1] << shift2) & bm);
  x[x.length-1]>>=shift;
 }

 this.q = zclip(q);
 this.mod = zclip(x);
 return this;
}

function simplemod(i,m)
{
 var c=0, v;
 for(var n=i.length-1; n>=0; n--)
 {
  v=i[n];
  c=((v >> bd) + (c<<bd)) % m;
  c=((v & bdm) + (c<<bd)) % m;
 }
 return c;
}

function bmod(p,m)
{
 if(m.length==1)
 {
  if(p.length==1) return [p[0] % m[0]];
  if(m[0] < bdm) return [simplemod(p,m[0])];
 }

 var r=bdiv(p,m);
 return r.mod;
}

function bmod2(x,m,mu)
{
 var xl=x.length - (m.length << 1);
 if(xl > 0) return bmod2(x.slice(0,xl).concat(bmod2(x.slice(xl),m,mu)),m,mu);
```

```
 var ml1=m.length+1, ml2=m.length-1,rr;
 var q3=bmul(x.slice(ml2),mu).slice(ml1);
 var r1=x.slice(0,ml1);
 var r2=bmul(q3,m).slice(0,ml1);
 var r=bsub(r1,r2);

 if(r.length==0)
 {
  r1[ml1]=1;
  r=bsub(r1,r2);
 }
 for(var n=0;;n++)
 {
  rr=bsub(r,m);
  if(rr.length==0) break;
  r=rr;
  if(n>=3) return bmod2(r,m,mu);
 }
 return r;
}

function bexpmod(g,e,m)
{
 var a = g.concat();
 var l = e.length-1;
 var n = nbits(e[l])-2;

 for(; l >= 0; l--)
 {
  for(; n >= 0; n-=1)
  {
   a=bmod(bsqr(a),m);
   if(e[l] & (1<<n)) a=bmod(bmul(a,g),m);
  }
  n = bs-1;
 }
 return a;
}

function bmodexp(g,e,m)
{
 var a=g.concat();
 var l=e.length-1;
 var n=m.length*2;
 var mu=zeros(n+1);
 mu[n]=1;
 mu=bdiv(mu,m).q;

 n = nbits(e[l])-2;

 for(; l >= 0; l--)
 {
  for(; n >= 0; n-=1)
  {
   a=bmod2(bsqr(a),m, mu);
   if(e[l] & (1<<n)) a=bmod2(bmul(a,g),m, mu);
  }
  n = bs-1;
 }
 return a;
}

function RSAencrypt(s, e, m) { return bexpmod(s,e,m); }

function RSAdecrypt(m, d, p, q, u)
{
 var xp = bmodexp(bmod(m,p), bmod(d,bsub(p,[1])), p);
 var xq = bmodexp(bmod(m,q), bmod(d,bsub(q,[1])), q);
```

```
 var t=bsub(xq,xp);
 if(t.length==0)
 {
  t=bsub(xp,xq);
  t=bmod(bmul(t, u), q);
  t=bsub(q,t);
 }
 else
 {
  t=bmod(bmul(t, u), q);
 }
 return badd(bmul(t,p), xp);
}

function mpi2b(s)
{
 var bn=1, r=[0], rn=0, sb=256;
 var c, sn=s.length;
 if(sn < 2)
 {
    alert('string too short, not a MPI');
    return 0;
 }

 var len=(sn-2)*8;
 var bits=s.charCodeAt(0)*256+s.charCodeAt(1);
 if(bits > len || bits < len-8)
 {
    alert('not a MPI, bits='+bits+",len="+len);
    return 0;
 }

 for(var n=0; n<len; n++)
 {
  if((sb<<=1) > 255)
  {
   sb=1; c=s.charCodeAt(--sn);
  }
  if(bn > bm)
  {
   bn=1;
   r[++rn]=0;
  }
  if(c & sb) r[rn]|=bn;
  bn<<=1;
 }
 return r;
}

function b2mpi(b)
{
 var bn=1, bc=0, r=[0], rb=1, rn=0;
 var bits=b.length*bs;
 var n, rr='';

 for(n=0; n<bits; n++)
 {
  if(b[bc] & bn) r[rn]|=rb;
  if((rb<<=1) > 255)
  {
   rb=1; r[++rn]=0;
  }
  if((bn<<=1) > bm)
  {
   bn=1; bc++;
  }
 }

 while(rn && r[rn]==0) rn--;
```

```
 bn=256;
 for(bits=8; bits>0; bits--) if(r[rn] & (bn>>=1)) break;
 bits+=rn*8;

 rr+=String.fromCharCode(bits/256)+String.fromCharCode(bits%256);
 if(bits) for(n=rn; n>=0; n--) rr+=String.fromCharCode(r[n]);
 return rr;
}
```

// keygen.js – генерація ключів RSA

```
var rSeed=[], Rs=[];
var Sr, Rsl, Rbits, Rbits2;
var Rx=0, Ry=0;

function r(n)
{
 return Math.floor(Math.random()*n);
}

function ror(a,n)
{
 n&=7;
 return n?((a>>n)|((a<<(8-n))&255)):a;
}

function seed(s)
{
 var n=0,nn=0;
 var x, y, t;

 while(n < s.length)
 {
  while(n<s.length && s.charCodeAt(n)<=32) n++;
  if(n < s.length) rSeed[nn]=parseInt("0x"+s.substr(n,2));
  n+=3; nn++;
 }

 Rsl=rSeed.length;
 Sr=r(256);
 Rbits=0;

 if(Rs.length==0)
 {
  for(x=0; x<256; x++) Rs[x]=x;
 }
 y=0;
 for(x=0; x<256; x++)
 {
  y=(rSeed[x] + s[x] + y) % 256;
  t=s[x]; s[x]=s[y]; s[y]=t;
 }
 Rx=Ry=0;
 alert("Random seed updated. Seed size is: "+Rsl);
}

function rc()
{
 var t;


 Rx=++Rx & 255;
 Ry=(Rs[Rx] + Ry) & 255;
 t=Rs[Rx]; Rs[Rx]=Rs[Ry]; Rs[Ry]=t;
 Sr^= Rs[(Rs[Rx] + Rs[Ry]) & 255];

Sr^= r(256);

 Sr^= ror(rSeed[r(Rsl)],r(8));
 Sr^= ror(rSeed[r(Rsl)],r(8));
 return Sr;
}

function rand(n)
{
 if(n==2)
 {
```

```
  if(!Rbits)
  {
   Rbits=8;
   Rbits2=rc(256);
  }
  Rbits--;
  var r=Rbits2 & 1;
  Rbits2>>=1;
  return r;
 }

 var m=1;

 r=0;
 while (n>m && m > 0)
 {
  m<<=8; r=(r<<8) |rc();
 }
 if(r<0) r ^= 0x80000000;
 return r % n;
}


function beq(a,b)
{
 if(a.length != b.length) return 0;

 for(var n=a.length-1; n>=0; n--)
 {
  if(a[n] != b[n]) return 0;
 }
 return 1;
}

function blshift(a,b)
{
 var n, c=0;
 var r=new Array(a.length);

 for(n=0; n<a.length; n++)
 {
  c|= (a[n]<<b);
  r[n]= c & bm;
  c>>=bs;
 }
 if(c) r[n]=c;
 return r;
}

function brshift(a)
{
 var c=0,n,cc;
 var r=new Array(a.length);

 for(n=a.length-1; n>=0; n--)
 {
  c<<=bs;
  cc=a[n];
  r[n]=(cc | c)>>1;
  c=cc & 1;
 }
 n=r.length;
 if(r[n-1]) return r;
 while(n > 1 && r[n-1]==0) n--;
 return r.slice(0,n);
}

function bgcd(uu,vv)
{
```

```
 var d, t, v=vv.concat(), u=uu.concat();
 for(;;)
 {
  d=bsub(v,u);
  if(beq(d,[0])) return u;
  if(d.length)
  {
   while((d[0] & 1) ==0) d=brshift(d);
   v=d;
  }
  else
  {
   t=v; v=u; u=t;
  }
 }
}

function rnum(bits)
{
 var n,b=1,c=0;
 var a=[];
 if(bits==0) bits=1;
 for(n=bits; n>0; n--)
 {

  if(rand(2)) a[c]|=b;
  b<<=1;
  if(b==bx2)
  {
   b=1;
   c++;
  }
 }
 return a;
}

var Primes=[3, 5, 7, 11, 13, 17, 19,
        23, 29, 31, 37, 41, 43, 47, 53,
        59, 61, 67, 71, 73, 79, 83, 89,
        97, 101, 103, 107, 109, 113, 127, 131,
        137, 139, 149, 151, 157, 163, 167, 173,
        179, 181, 191, 193, 197, 199, 211, 223,
        227, 229, 233, 239, 241, 251, 257, 263,
        269, 271, 277, 281, 283, 293, 307, 311,
        313, 317, 331, 337, 347, 349, 353, 359,
        367, 373, 379, 383, 389, 397, 401, 409,
        419, 421, 431, 433, 439, 443, 449, 457,
        461, 463, 467, 479, 487, 491, 499, 503,
        509, 521, 523, 541, 547, 557, 563, 569,
        571, 577, 587, 593, 599, 601, 607, 613,
        617, 619, 631, 641, 643, 647, 653, 659,
        661, 673, 677, 683, 691, 701, 709, 719,
        727, 733, 739, 743, 751, 757, 761, 769,
        773, 787, 797, 809, 811, 821, 823, 827,
        829, 839, 853, 857, 859, 863, 877, 881,
        883, 887, 907, 911, 919, 929, 937, 941,
        947, 953, 967, 971, 977, 983, 991, 997,
        1009, 1013, 1019, 1021];


var sieveSize=4000;
var sieve0=-1* sieveSize;
var sieve=[];
var lastPrime=0;

function nextPrime(p)
{
 var n;
```

```
 if(p == Primes[lastPrime] && lastPrime <Primes.length-1)
 {
  return Primes[++lastPrime];
 }
 if(p<Primes[Primes.length-1])
 {
  for(n=Primes.length-2; n>0; n--)
  {
   if(Primes[n] <= p)
   {
    lastPrime=n+1;
    return Primes[n+1];
   }
  }
 }
var pp,m;
p++; if((p & 1)==0) p++;
 for(;;)
 {
  if(p-sieve0 > sieveSize || p < sieve0)
  {
   for(n=sieveSize-1; n>=0; n--) sieve[n]=0;
   sieve0=p;
   primes=Primes.concat();
  }

  if(sieve[p-sieve0]==0)
  {

   for(n=0; n<primes.length; n++)
   {
    if((pp=primes[n]) && p % pp ==0)
    {
     for(m=p-sieve0; m<sieveSize; m+=pp) sieve[m]=pp;
     p+=2;
     primes[n]=0;
     break;
    }
   }
   if(n >= primes.length)
   {
    return p;
   }
  }
  else
  {
   p+=2;
  }
 }
}

function divisable(n,max)
than max, else return 0
{
 if((n[0] & 1) == 0) return 2;
 for(c=0; c<Primes.length; c++)
 {
  if(Primes[c] >= max) return 0;
  if(simplemod(n,Primes[c])==0) return Primes[c];
 }
 c=Primes[Primes.length-1];
 for(;;)
 {
  c=nextPrime(c);
  if(c >= max) return 0;
  if(simplemod(n,c)==0) return c;
 }
}
```

```
function bPowOf2(pow)
{
 var r=[], n, m=Math.floor(pow/bs);
 for(n=m-1; n>=0; n--) r[n]=0;
 r[m]=1<<(pow % bs);
 return r;
}

function mpp(bits)
{
 if(bits < 10) return [Primes[rand(Primes.length)]];
 if(bits <=20) return [nextPrime(rand(1<<bits))];

 var c=10, m=20, B=bits*bits/c, r, q, I, R, n, a, b, d, R2, nMinus1;

 if(bits > m*2)
 {
  for(;;)
  {
   r=Math.pow(2,Math.random()-1);
   if(bits - r * bits > m) break;
  }
 }
 else
 {
  r=0.5
 }

 q=mpp(Math.floor(r*bits)+1);
 I=bPowOf2(bits-2);
 I=bdiv(I,q).q;
 Il=I.length;
 for(;;)
 {
  R=[];
  for(n=0; n<Il; n++) R[n]=rand(bx2);
  R[Il-1] %= I[Il-1];
  R=bmod(R,I);
  if(!R[0]) R[0]|=1; // must be greater or equal to 1
  R=badd(R,I);
  n=blshift(bmul(R,q),1); // 2Rq+1
  n[0]|=1;
  if(!divisable(n,B))
  {
   a=rnum(bits-1);
   a[0]|=2;
   nMinus1=bsub(n,[1]);
   var x=bmodexp(a,nMinus1,n);
   if(beq(x,[1]))
   {
    R2=blshift(R,1);
    b=bsub(bmodexp(a,R2,n),[1]);
    d=bgcd(b,n);
    if(beq(d,[1])) return n;
   }
  }
 }
}


function sub2(a,b)
{
 var r=bsub(a,b);
 if(r.length==0)
 {
  this.a=bsub(b,a);
  this.sign=1;
 }
 else
```

```
 {
  this.a=r;
  this.sign=0;
 }
 return this;
}

function signedsub(a,b)
{
 if(a.sign)
 {
  if(b.sign) return sub2(b,a);
  else
  {
   this.a=badd(a,b);
   this.sign=1;
  }
 }
 else
 {
  if(b.sign)
  {
   this.a=badd(a,b);
   this.sign=0;
  }
  else return sub2(a,b);
 }
 return this;
}


function modinverse(x,n)
{
 var y=n.concat(), t, r, bq, a=[1], b=[0], ts;

 a.sign=0; b.sign=0;

 while(y.length > 1 || y[0])
 {
  t=y.concat();
  r=bdiv(x,y);
  y=r.mod;
  q=r.q;
  x=t;
  t=b.concat(); ts=b.sign;
  bq=bmul(b,q);
  bq.sign=b.sign;
  r=signedsub(a,bq);
  b=r.a; b.sign=r.sign;
  a=t; a.sign=ts;
 }

 if(x.length != 1 || x[0] != 1) return [0];

 if(a.sign)
 {
  a=bsub(n,a);
 }
 return a;
}


var rsa_p,rsa_q,rsa_e,rsa_d,rsa_pq, rsa_u;

function rsaKeys(bits)
{
 var c, p1q1;

 bits=parseInt(bits);
```

```
 rsa_q=mpp(bits);
 rsa_p=mpp(bits);
 p1q1=bmul(bsub(rsa_p,[1]),bsub(rsa_q,[1]));

 for(c=5; c<Primes.length; c++)
 {
  rsa_e=[Primes[c]];
  rsa_d=modinverse(rsa_e,p1q1);
  if(rsa_d.length != 1 || rsa_d[0]!=0) break;
 }
 rsa_pq=bmul(rsa_p,rsa_q);
 rsa_u=modinverse(rsa_p,rsa_q);

 return;
}
```

// api.php – роути API

```
<?php

use Illuminate\Http\Request;

Route::group(['middleware' => ['auth']], function () {

    /* FeedsContr */
    Route::get('feed', 'FeedContr@index');

    /* UserContr */

    Route::get('user', 'UserContr@index');
    Route::put('user', 'UserContr@update');
    // Пошук користувачів
    Route::get('user/{query}', 'UserContr@search');

    /* FriendContr */

    Route::get('friend', 'FriendContr@index');
    Route::get('friend/{friend}', 'FriendContr@show');
    Route::post('friend/{friend}', 'FriendContr@store');
    Route::put('friend/{friend}', 'FriendContr@update');
    Route::delete('friend/{friend}', 'FriendContr@destroy');

    /* Group Contr */
    Route::resource('group', 'GroupContr', [
        'except' => ['create', 'edit']
    ]);
    Route::get('group/{group}/adduser/{user}', 'GroupContr@addUserToGroup');
    Route::get('group/{group}/leave', 'GroupContr@userLeavesGroup');

    /* Контролер групових повідомлень */

    Route::get('group/{group}/message', 'MessageContr@groupIndex');
    Route::get('group/{group}/message/{message}', 'MessageContr@groupShow');
    Route::post('group/{group}/message', 'MessageContr@groupStore');
    Route::put('group/{group}/message/{message}', 'MessageContr@groupUpdate');
    Route::delete('group/{group}/message/{message}',
'MessageContr@groupDestroy');

    /* Контролер повідомлень одній людині */

     Route::get('friend/{friend}/message', 'MessageContr@friendIndex');
     Route::get('friend/{friend}/message/{message}', 'MessageContr@friendShow');
     Route::post('friend/{friend}/message', 'MessageContr@friendStore');
     Route::put('friend/{friend}/message/{message}',
'MessageContr@friendUpdate');
     Route::delete('friend/{friend}/message/{message}',
'MessageContr@friendDestroy');
});
```