Хмельницький національний університет
Факультет інформаційних технологій
Кафедра інженерії програмного забезпечення

# КВАЛІФІКАЦІЙНА РОБОТА

Веб-система для безпечного обміну повідомленнями з використанням криптографічних алгоритмів

Назва теми

Рівень вищої освіти Перший (бакалаврський)

Галузь знань 12 «Інформаційні технології»

Спеціальність 121 «Інженерія програмного забезпечення»

Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

Шифр КвРІПЗ.190145.19.23.ПЗ

Виконав студент IV курсу група ІПЗ-19-1 _____ Д. О. Юртаєв
Підпис Ініціали, прізвище

Керівник канд. пед. наук, доцент _____ Н. І. Праворська
Науковий ступінь, звання Підпис Ініціали, прізвище

Нормоконтролер канд. тех. наук, доцент _____ О. М. Яшина
Підпис Ініціали, прізвище

**До захисту допускаю:**
Завідувач кафедри інженерії програмного забезпечення

_____ Л. П. Бедратюк
Підпис Ініціали, прізвище

6 червня 2023 р.

Хмельницький 2023

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет інформаційних технологій
Кафедра Інженерії програмного забезпечення
Рівень вищої освіти Перший (бакалаврський)
Галузь знань 12 «Інформаційні технології»
Спеціальність 121 «Інженерія програмного забезпечення»
Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри ІПЗ
Л. П. Бедратюк
05 02 2023 р.

# ЗАВДАННЯ
# НА КВАЛІФІКАЦІЙНУ РОБОТУ

Юртаєву Денису Олександровичу
Прізвище, ім'я, по батькові студента

1. Тема проекту (роботи) Веб-система для безпечного обміну повідомленнями з використанням криптографіних алгоритмів

Керівник проекту (роботи) Праворська Наталя Іванівна, канд. пед. наук, доцент
Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 01.03.2023 р. № 5
2. Строк подання студентом проекту (роботи) на кафедру 01.06.2023 р.
3. Вихідні дані до проекту (роботи) Матеріали переддипломної практики
4. Зміст пояснювальної записки (перелік питань, які потрібно розробити) Дослідження предметної області та постановка задачі, проектування програмного забезпечення, програмна реалізація, тестування програмного забезпечення
5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень) Три креслення формату А3

6. Консультанти розділів дипломного проекту

| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
|---|---|---|---|
| | | завдання видав | завдання прий |
| Нормоконтроль | Яшина О. М., доцент кафедри ІПЗ | 5.06.2023 | 5.06.2023 |
| Антиплагіат | Гурман І. В., доцент кафедри ІПЗ | 5.06.2023 | 5.06.2023 |

7. Дата видачі завдання « 05 » лютого 2021р.

## КАЛЕНДАРНИЙ ПЛАН

| Назва етапів (розділів) дипломного проекту (роботи) | Строк виконання етапів проекту (роботи) | Примітка |
|---|---|---|
| 2 Дослідження предметної області, в якій планується використання програмного засобу (ПЗ), визначення задач та вимог, розробка технічного завдання | 02.01–31.01 2023 | |
| 3 Проектування програмного забезпечення | 01.02–28.02 2023 | |
| 4 Програмна реалізація | 01.03–10.04 2023 | |
| 5 Тестування програмного забезпечення | 11.04–30.04 2023 | |
| 6 Написання вступу, загальних висновків, оформлення джерел посилання та додатків. Оформлення пояснювальної записки КвР згідно вимог стандартів | 01.05–25.05 2023 | |
| 7 Попередній захист КвР | Травень 2023 (згідно графіка) | |
| 8 Перевірка КвР на плагіат, нормоконтроль, отримання відгуків та рецензій. Брошурування (зшиття) пояснювальної записки | 26.05 – 30.05.2023 | |
| 9 Підготовка до захисту та захист КвР | з 01.06.2023 | |

Студент _____ Підпис Д. О. Юртаєв Ініціали, прізвище

Керівник проекту (роботи) _____ Підпис Н. І. Праворська Ініціали, прізвище

# АНОТАЦІЯ

Тема кваліфікаційної роботи: Веб-система для безпечного обміну повідомленнями з використанням криптографічних алгоритмів.

Автор проекту: Юртаєв Денис Олександрович.

Керівник проекту: Праворська Наталя Іванівна.

Пояснювальна записка: 62 с., 16 рис., 1 табл., 4 дод., 40 джерел.

Графічна частина: 3 креслення формату А3.

Об'єктом дослідження кваліфікаційної роботи є розробка системи для миттєвого та безпечного обміну повідомленнями з використанням криптографічних алгоритмів, а також розробка клієнтського веб-застосунку.

Метою кваліфікаційної роботи є проектування та розробка системи для обміну повідомленнями.

У кваліфікаційній роботі було проведено аналіз предметної області, досліджено існуючі програмні засоби, їх переваги та недоліки. Було визначено вимоги до програмної системи, здійснено його проектування та вибрано відповідні технології для його реалізації.

В результаті було розроблено програмне забезпечення, яке відповідає визначеним вимогам.

30.05.2023
Дата

Підпис

# ВІДОМІСТЬ ДОКУМЕНТІВ

| № рядка | Формат | Позначення документа | Найменування документа | К-сть аркушів | № екз. | Примітка |
|---|---|---|---|---|---|---|
| | | | | | | |
| | | | Текстові документи | | | |
| | | | | | | |
| 1 | А4 | КвРІПЗ.190145.19.25.ПЗ | Пояснювальна записка | 69 | | |
| 2 | А4 | | Завдання на кваліфіка- | 1 | | |
| | | | ційну роботу | | | |
| 3 | А4 | | Анотація | 1 | | |
| | | | | | | |
| | | | Графічні документи | | | |
| | | | | | | |
| 4 | А3 | КвРІПЗ.190145.19.25.Е8 | Діаграма використання | 1 | | |
| | | | | | | |
| 5 | А3 | КвРІПЗ.190145.19.25.Е8 | ER-діаграма бази даних | 1 | | |
| | | | | | | |
| 6 | А3 | КвРІПЗ.190145.19.25.Е8 | Діаграма модулів | 1 | | |
| | | | | | | |
| 7 | А4 | | Презентаційні матеріали | 9 | | |

| Змн. | Арк. | № докум. | Підпис | Дат |
|---|---|---|---|---|
| Виконав | | Юртаєв Д.О. | | 6.06 |
| Керівник | | Праворська Н.І. | | 6.06 |
| Н. Контр. | | Яшина О.М. | | 6.06 |
| Зав. Каф. | | Бедратюк | | 6.06 |

КвРІПЗ.190145.19.23.ВД

Веб-система для безпечного обміну повідомленнями з використанням криптографічних алгоритмів

Відомість документів

| Літ. | Арк. | Акрушів |
|---|---|---|
| | 1 | 1 |

ХНУ, ІПЗ-19-1

# ЗМІСТ



| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | КвРІПЗ.190145.19.23.ПЗ | | |
| Змн. | Арк. | № докум. | Підпис | Дат | | | |
| Виконав | | Юртаєв Д.О. | | 6.06 | Веб-система для безпечного обміну повідомленнями з використанням криптографічних алгоритмів | Літ. | Арк. 1 / Акрушів 69 |
| Керівник | | Праворська Н.І. | | 6.06 | | | |
| Н. Контр. | | Яшина О.М. | | 6.06 | Відомість документів | ХНУ, ІПЗ-19-1 | |
| Зав. Каф. | | Бедратюк | | 6.06 | | | |

# ВСТУП

Обмін інформацією за допомогою мережі Інтернет це те що ми використовуємо кожного дня. Від просто спілкування з друзями за допомогою месенджера до комунікації під час роботи. Існує досить багато різних додатків, які надають змогу спілкуватись як за допомогою текстових повідомлень так і за допомогою відеозв'язку. Деякі додатки містять більше функціоналу, деякі менше, але основною метою всіх цих додатків є базовий обмін текстовими повідомленнями, файлами та голосовий або відео-зв'язок.

Важливим атрибутом кожного месенджеру є його безпека. В даному випадку ми маємо на увазі те як дані передаються по каналам зв'язку, як вони зберігаються, та чи є дані користувача конфіденційними. В ситуації коли два користувача обмінюються один з одним повідомленнями вони повинні бути впевнені, що вміст їх повідомлень не прочитає ніхто сторонній, навіть компанія яка володіє месенджером.

В AppStore, Play Market або просто в мережі можна знайти десятки додатків, які користувачі використовуються для спілкування, будь то месенджер чи соціальна мережа. Проте є досить мало додатків яким можна повністю довіряти. У випадку з соціальними мережами взагалі тяжко бути впевненим в надійності їх сервісів, оскільки досить багато платформ обробляють та продають персональні дані користувачів третій особі тощо. Досить багато користувачів, особливо тих хто має невеликий обсяг знань про інформаційні технології можуть просто ігнорувати ці факти.

Також ще один важливий атрибут додатку це зручність його використання, доступність на різних платформах, UI/UX дизайн додатку, швидкість його роботи та скільки ресурсів він споживає. Під час огляду деяких месенджерів можна зазначити те що не всі вони є досить зручними як з точки UI/UX дизайну так і з точки зору функціональних можливостей, який іноді буває не достатньо.

Актуальність даної теми полягає в тому що існує досить багато додатків чи сервісів для спілкування, обміну повідомлення, медіа-даними, але не всі вони є безпечними та зручними у користуванні, а деяким бракує деяких функцій. Отже буде розроблено веб-система та веб-застосунок з відкритим початковим кодом, який зможе задовольнити вимоги безпеки та зручності використання додатку.

Мета кваліфікаційної роботи – розробити веб-систему та веб-застосунок за допомогою користувачі отримають базові можливості обміну тестовими повідомленнями з використанням алгоритмів шифрування, що забезпечать безпеку кожного користувача та його даних.

Для успішного виконання мети кваліфікаційної роботи потрібно виконати наступні задачі:

– детально дослідити предмету область;

– проаналізувати існуючі рішення;

– визначити вимоги до кваліфікаційної роботи та створити технічне завдання;

– розглянути інструменти та технології які допоможуть вирішити завдання;

– розробити архітектуру системи.

– створити зручний та сучасний UI/UX дизайн клієнтського додатку;

– виконати програмну реалізацію, провести тестування та розмістити готовий до використання застосунку та систему у мережі.

# 1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

## 1.1 Змістовний аналіз предметної області, її структурних та функціональних особливостей

Месенджер – це система, яка надає користувачам такі базові можливості як миттєво обмінюватись текстовими повідомленням, файлами, медіа-даними, голосовими повідомленнями, дзвонити один одному з або без відео-зв’язку. Окрім того що все це має відбувати у режимі реального часу, тобто миттєво без затримок, сама система має бути безпечною для користувача. Повідомлення користувачів не повинні потрапити до третьої особи.

Тут є досить важливі питання, як саме будуть оброблятись та зберігатись дані користувачів, який протокол передачі даних буде використовуватись. Як захистити дані користувачів навіть у випадку витоку бази даних поза межі серверу в загальний доступ.

В першу чергу, варто зазначити те, що на даний момент досить безпечним є протокол передачі даних HTTPS, за допомогою якого користувачі виконують досить багато різних операцій, таких як внесення даних банківської карти для оплати в Інтернет-магазині, завантаження конфіденційних даних на хмарні сховища, тощо. Для додаткової безпеки можна вигадати свій протокол передачі даних, який зможе використовуватись поверх HTTP(S), TCP/IP, Web-Sockets протоколів, тобто сам канал транспортування даних залишається HTTP, але формат даних та те як вони шифруються будуть відрізнятись. Дана опція не є обов’язковою для безпечного месенджера, але має місце бути у деяких перевірених системах обміну даними.

Оскільки є досить безпечний протокол передачі даних, варто подумати про те, як вберегти повідомлення користувачів від самого серверу. Для вирішення цієї задачі потрібно використовувати end-to-end шифрування. У такому випадку тільки користувачі, які беруть участь у діалозі, зможуть читати ці дані.

Минулий приклад захисту повідомлень досить добре покаже себе для шифрування саме діалогу де спілкуються лише 2 користувача та використовують для спілкування лише одні ті самі пристрої. Коли мова йде про більше ніж 2 користувача, це вже називається чат, деякі чати в месенджера можуть бути публічними, куди можуть доєднатись будь-які користувачі, які в свою чергу можуть використовувати більше одного пристрою (наприклад ПК та телефон), ситуація стає складнішою.

Для виходу з цієї ситуації потрібно поділити чати на різні типи:

– секретний чат. Чат який буде захищений за допомогою end-to-end шифрування, учасниками чату можу бути лише два користувача, а також чат та його вміст зберігаються на стороні клієнта. Оскільки приватний ключ фізично знаходиться у користувача на одному пристрої, він не зможе прочитати цей чат з іншого пристрою;

– звичайний чат. Чат на двох або більше користувачів, який буде захищено асиметричним шифруванням на стороні серверу що дасть зможу читати цей чат з різних пристроїв усім учасникам, а також не сильно вплине на обчислювальні можливості серверу.

## 1.2 Аналіз наявного програмно-технічного забезпечення предметної області

Розглянемо реалізовані варіанти систем месенджерів, які найбільш популярні серед користувачів, для врахування їх переваг та недоліків, що допоможе покращити майбутні програмний продукт, уникнути не дуже вдалих рішень та зрозуміти, чому користувачі обрали ці додатки, а не інші.

Viber є досить простою та популярною системою для обміну повідомленнями. Має досить багато користувачів. Дозволяє обмінюватись текстовими повідомленнями, телефонувати іншим користувача з або без використання відео-зв’язку. Застосунок має досить простий та класичний для

месенджеру інтерфейс (рисунок 1.1). Також присутня можливість використання секретних чатів з додатковим рівнем безпеки.

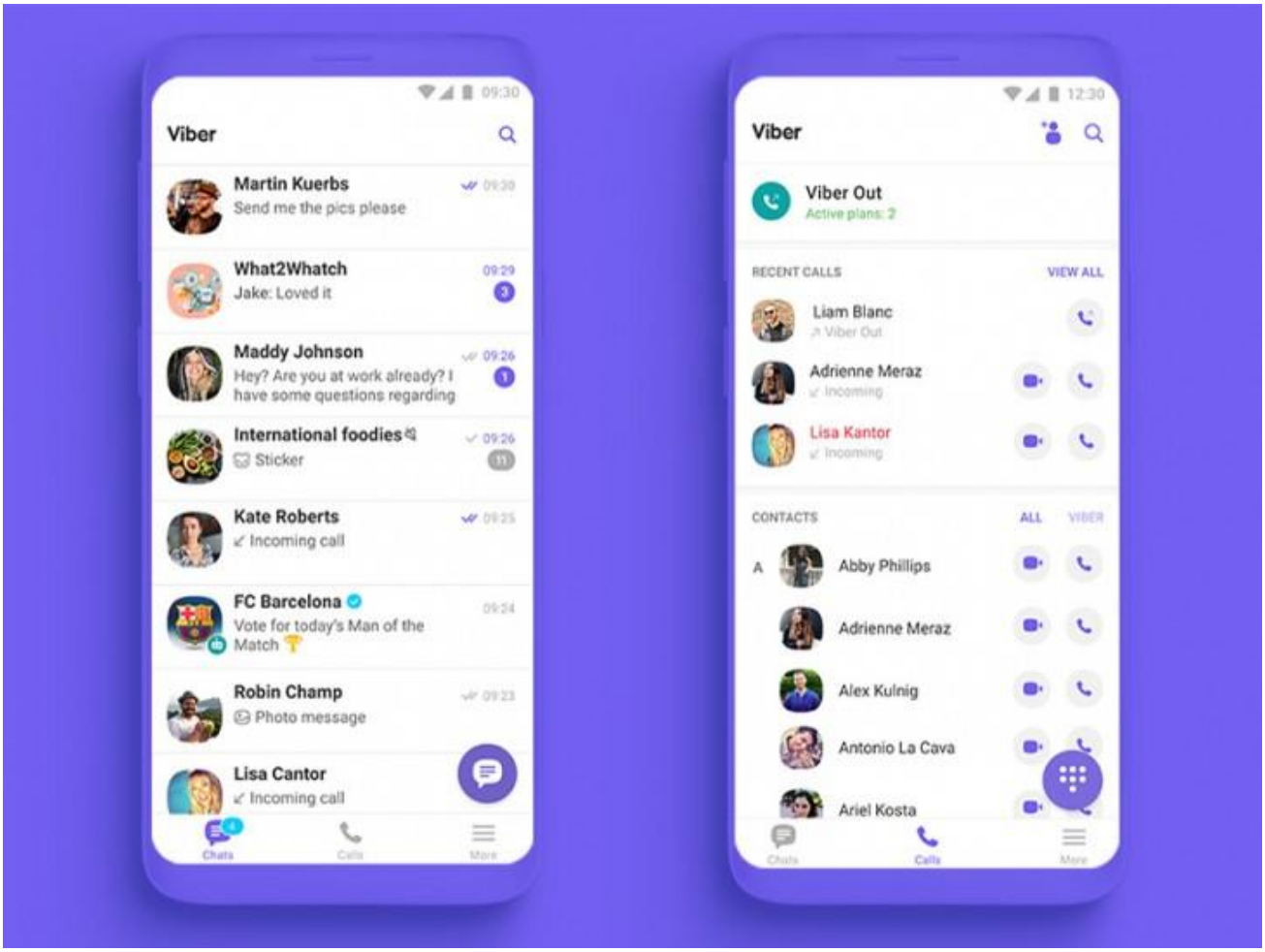



Рисунок 1.1 − Інтерфейс користувача Viber.

Хоча здавалось би що Viber в цілому є безпечним так як може використовувати end-to-end шифрування, але рівень безпеки додатку все ще не високий, іноді в ньому можуть виникати проблеми зі швидкістю роботи. Сервіс використовує номер телефону користувача для створення акаунту. Дані на серверах на зажди зберігаються зашифрованими, а також досить часто користувачі знаходять різні вразливості додатку на різних платформах. Застосунок досить популярний серед деяких користувачів через його простоту та мінімальну кількість налаштувань, але для інших користувачів це може бути скоріше мінусом додатку, так як кастомізація та налаштування це досить важливий аспект, який покращує взаємодію користувача з додатком.

Розглянемо наступний застосунок Telegram. Він вже має набагато більше функцій ніж Viber та у порівнянні з ним відчувається досить великий технічний прогрес. Застосунок окрім базових можливостей месенджеру, надає багато додаткових функцій, починаючи від таких дрібниць як стікери, самостійне створення стікерів, сегрегація чатів та каналів по окремим група, які можна налаштувати до використання та налаштування proxy для додатку, перегляду активних сесії користувача, автоматичне видалення акаунту у разі бездіяльності користувача.

Також інтерфейс користувача має дуже приємний та сучасний зовнішній вигляд (Рисунок 1.2). Підхід до UI/UX дизайну під час розробки був досить серйозний, що лише мотивує тебе використовувати саме цей застосунок. Окрім того весь UI має можливість кастомізації, від шрифту, його розміру та background для чатів до зміни кольорової гами всього додатку. Також є дуже детальні налаштування автозавантаження медіа-даних, таких як голосові повідомлення, фото та відео. Ця функція дозволяє вимкнути всі автозавантаження, що під час низької швидкості Інтернету дозволить комфортно використовувати застосунок, оскільки якщо завантажувати тільки текстові повідомлення, застосунок працюватиме дуже швидко з мінімальними затратами ресурсів.

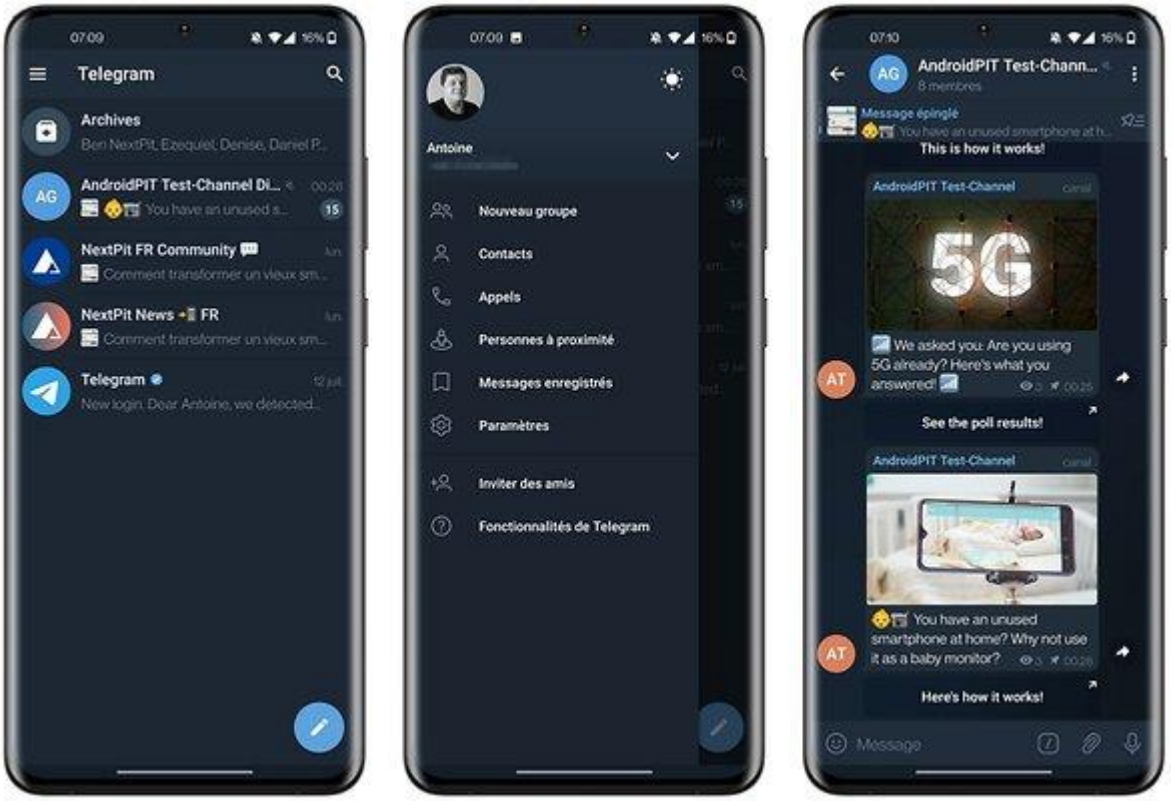



Рисунок 1.2 − Інтерфейс користувача Telegram.

Також Telegram має свій унікальний протокол, який називається MTProto (Рисунок 1.3). Даний протокол є досить потужним та стійким з точки зору безпеки. Він використовує client-side шифрування за допомогою власного алгоритму шифрування, який базується на auth_key (ключ який генерується у клієнта при створенні сесії) та msg_key який динамічно вираховується для кожного повідомлення. В результаті певних операцій вони компонують AES-258 ключ яким шифруються дані що йдуть до серверу. Також дані на серверах є зашифрованими. У разі використання секретних чатів, які на сервері не зберігаються, там за допомогою асиметричного шифрування дані передаються по цьому ж самому протоколу, тобто секретний чат має 2 рівня захисту. Спочатку клієнт шифрує дані асиметричним шифруванням, далі вони шифруються за допомогою алгоритму MTProto та доставляються до отримувача. Telegram в даному випадку виконує лише транспортування даних

від користувача до користувача. Для транспортування даних використовуються протоколи HTTP, HTTPS, TCP, Websockets, Websockets over HTTPS.

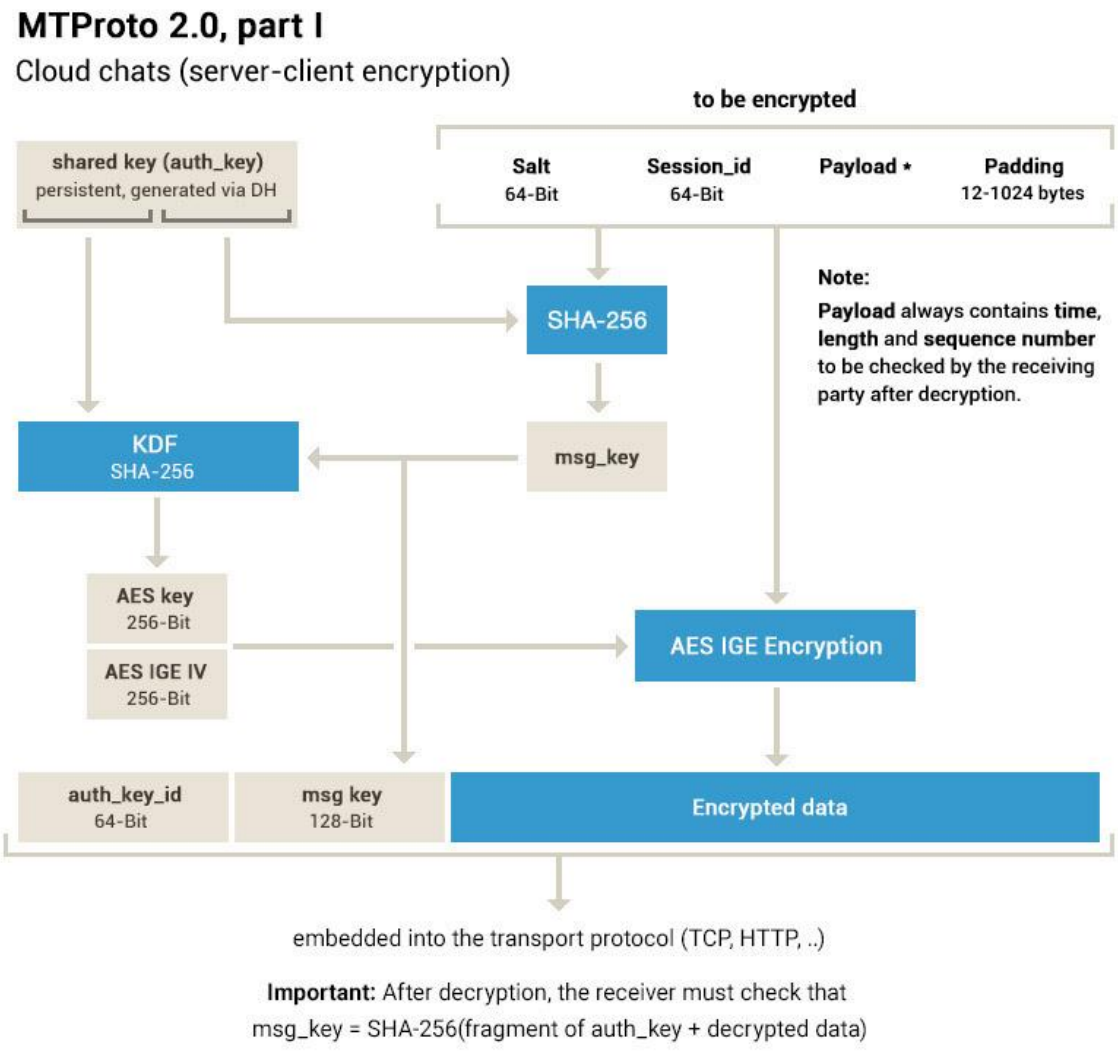



Рисунок 1.3 – Схема протоколу MTProto.

Telegram завжди працює над тим щоб дані користувачів були в безпеці, саме тому в них є bug bountry програма, яка передбачає винагороду за знайдення вразливостей від 100 до 100000 доларів. Деякий час вони навіть пропонували 300 тисяч доларів тому хто зможе знайти вразливості протоколу MTProto. Це досить ефективна практика для підняття рівня безпеки власної системи.

Наступний сервіс Discord. В першу чергу позиціонує себе як засіб для комунікації геймерів (Рисунок 3.4). Доступний на різних платформах. Має мобільну версію, версію для ПК, а також веб-застосунок для браузера.

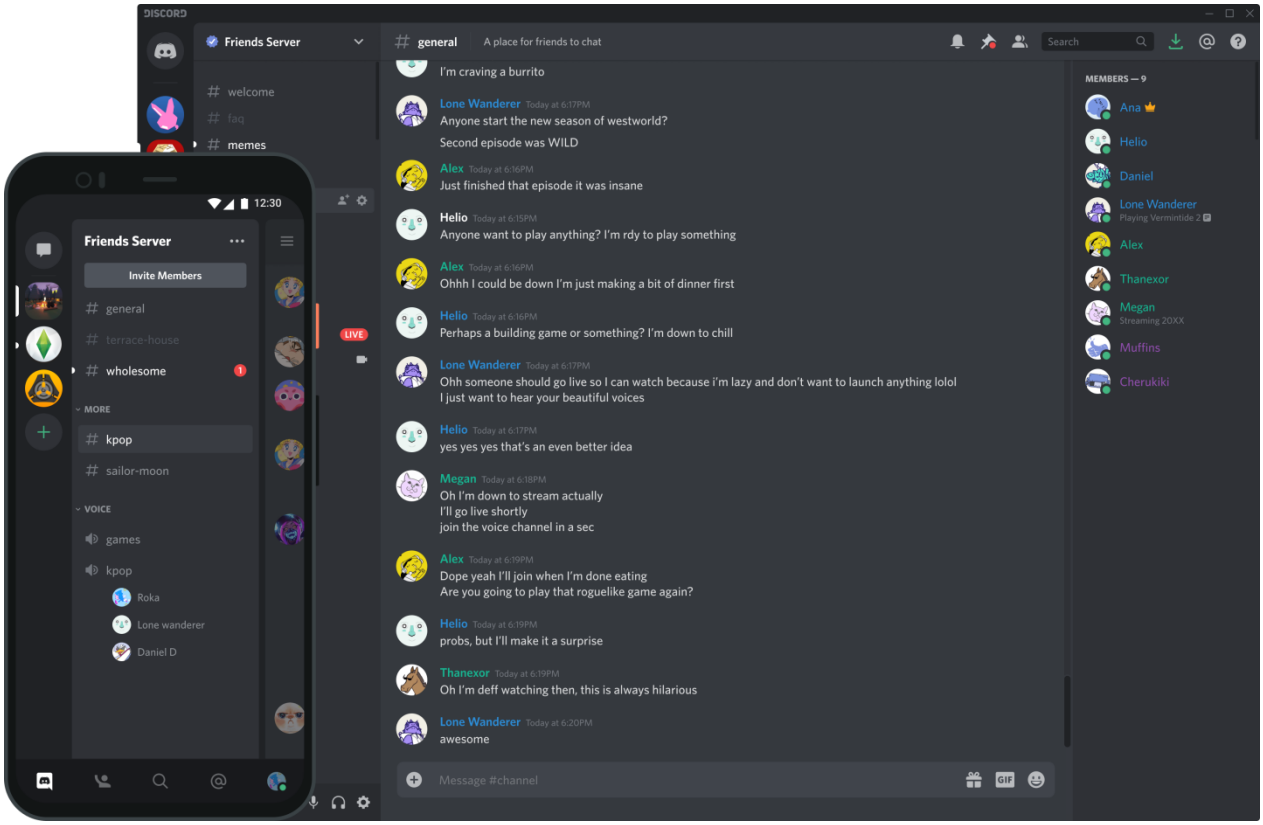



Рисунок 3.4 − Інтерфейс користувача Discord.

Система дозволяє користувача все так само обмінюватись повідомленнями, створювати чат з двома або більше учасниками, робити виклики з або без використання відео-зв’язку. ПК версія дозволяє транслювати екран або застосунок замість або з своїм відео з веб-камери, що є досить зручним для геймерів, так як можна проводити трансляцію того як ти граєш в якусь гру для інших користувачів. Основною відмінністю від інших засобів комунікації є так звані “сервера”. Сервер є таким собі каналом куди можуть додаватись користувачів. Ці канали вміщують в собі деяку кількість текстових та голосових чатів. Можуть вміщувати ролі як створить адміністратор сервера, з різними правами доступу. Права доступу діють на чати, для одних користувачів вони є відкритими, для інших закритими, або текстовий канал може бути тільки для читання куди адмін може додавати якісь новини. Також в Discrod як і в Telegram присутня система ботів та їх API. Боти є досить важливою частиною цих систем, так як можуть розроблятись користувачами з різними цілями, наприклад такими як адміністрування чатів чи ведення статистики. Discord

досить швидко працює та є оптимізованим, а також вміщує оверлей для десктоп версії що дозволяє користуватись ним під час гри в відеогру без перемикання вікон, що дуже спрощує використання додатку, як вже раніше зазначалось, геймерами. Дана система спілкування стала дуже популярної не тільки серед геймерів, а й серед звичайних користувачів, які збираються там в різних каналах, серверах, чатах з різними інтересами та проводять свій час за спілкуванням та комунікацією один з одним.

## 1.3 Визначення вимог до розроблюваної системи

Для визначення вимог програмного забезпечення варто використати деякі діаграми UML та таблиці. Перш за все потрібно описати акторів системи. Вони представлені в таблиці 3.1. Як бачимо акторів лише два, це неаунтифікований користувач та користувач який увійшов в систему.

Таблиця 3.1 – опис акторів системи.

| Актор | Опис |
|---|---|
| Неаунтифікований користувач | Має доступ лише до таких команд як login, register, restore password. |
| Аунтифікований користувач | Має всі інші можливості які надає йому система. Від створення чатів до відправки повідомлень та інше. |

Для розуміння базових речей які може виконувати система, гарним прикладом буде розробка UML-діаграми використання, яка зможе описати що актор (в даному випадку користувач) може робити в системі.

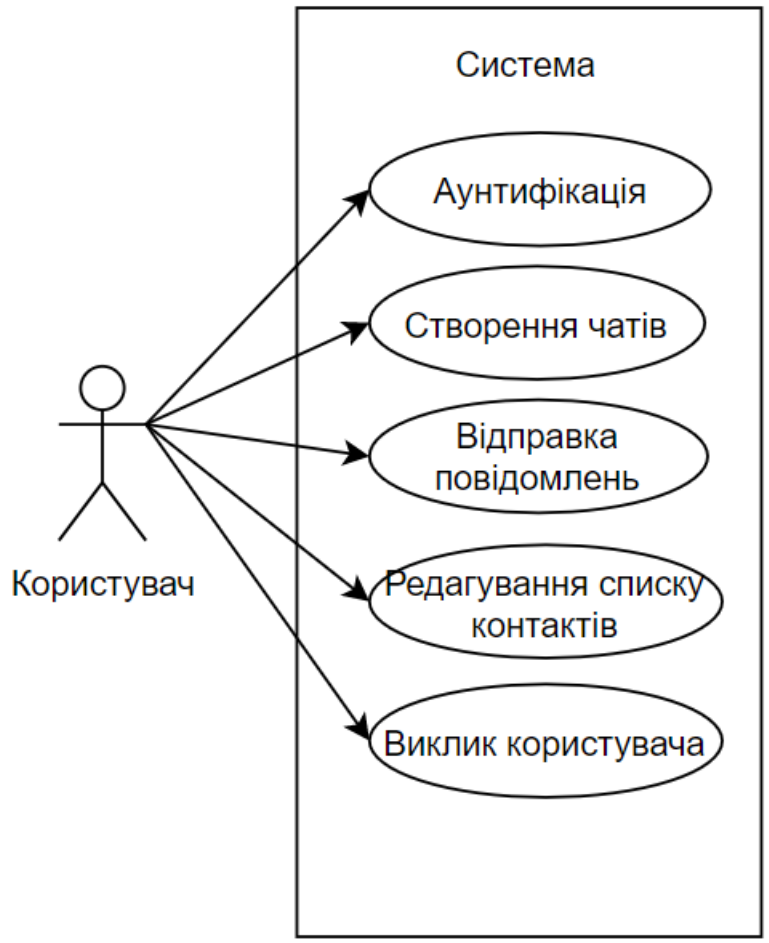



Рисунок 3.5 – UML-діаграма варіантів використання.

З вище представленої діаграми ми можемо побачити базові функції які буде виконувати користувач в розроблюваній системі. Користувач зможе створювати чати з іншими користувачами, відправляти повідомлення, редагувати список контактів, дзвонити іншому користувачу.

## 1.4 Висновок. Постановка задачі

Отже, месенджер це веб-сервіс, який дозволяє користувача он-лайн миттєво обмінюватись повідомленнями. Під час розробки програмного забезпечення необхідно врахувати такі вимоги, як безпека користувачів,

зручний та інтуїтивно зрозумілий інтерфейс, забезпечення миттєвого обміну повідомленнями.

Дизайн клієнтського веб-застосунку має бути досить зручним та привабливим, для цього можна використати найпоширеніші шаблони дизайну месенджерів, так як вони вже перевірені часом та користувачу буде дуже легко звикнути до всіх елементів користувацького інтерфейсу.

В результаті виконаної роботи було визначено базовий фунціонал та функціональні вимоги, необхідні системі для того щоб користувачі могли комфортно використовувати розроблювану систему обміну повідомленнями.

До основних функцій веб-системи відносяться:

– авторизація користувача;
– реєстрація користувача;
– миттєвий обмін повідомленнями;
– створення приватних чатів;
– створення публічних чатів;
– можливість видалення чату;
– редагування чату;
– редагування профілю користувача.

Варто розуміти що розробка системи для миттєвого обміну повідомленнями досить складний процес, який потребує глибоких знань та розумінь багатьох сфер інформаційних технологій, як з точки зору розробки серверної частини, так і з точки зору проектування розробки клієнтського застосунку та дизайну.

Отже дослідивши предметну область, визначивши базовий функціонал системи та розробивши відповідні вимоги можемо перейти до наступного етапу, а саме проектування системи.

# 2 ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

## 2.1 Вибір типу архітектури та шаблонів проектування

Спершу визначимо типи архітектури, які підходять для проектування даної системи. Таких типів архітектури всього два, а саме клієнт-серверна та P2P (peer-to-peer), кожна з яких має свої переваги та недоліки.

У випадку p2p архітектури ми маємо мережу, яка складається з рівноправних вузлів. Клієнтів та серверів як таких немає, кожен вузол є автономним, який ніким не контролюється. Тобто ми маємо децентралізовану мережу, яка до того ж дуже добре дозволяє зберігати працездатність системи, оскільки все децентралізовано, але виникає багато питань щодо безпеки цієї мережі, та чи можуть насправді одні вузли довіряти іншим. Також є питання щодо масштабування мережі при зростанні кількості вузлів, оскільки в даному випадку буде складно масштабувати мережу.

Якщо розглядати клієнт-серверну архітектуру то в даному випадку ми маємо сервер та клієнтів. Мережа є централізованою, клієнти звертаються до серверу, а той в свою чергу обробляє команди, які відправляє йому клієнт. Оскільки сервер обробляє всі операції користувачів, нам потрібно одразу розраховувати потенційну кількість можливих користувачів та навантаження на сервер, щоб правильно масштабувати систему. Даний підхід є більш безпечним з точки зору безпеки оскільки ми маємо сервер, який буде займатись обробкою за зберіганням даних користувачів.

Розглянемо основні недоліки P2P архітектури:

– недостатньо безпеки;

– розподіленість мережі між вузлами що ставить під питання те як і де дані повинні зберігатись;

– продуктивність може постраждати при зростанні кількості вузлів, виникає питання масштабування мережі.

Далі розглянемо основні недоліки клієнт-серверної архітектури:

– централізованість мережі, що при проблемах на сервері зробить недоступною всю мережу;

– більша вартість розробки та підтримки.

Розглянемо переваги клієнт-серверної архітектури:

– дані зберігаються на сервері, тому вони є досить захищеними;

– масштабувати клієнт-серверні додатки при зростаючому кількості користувачів та навантаженні досить просто як за допомогою вертикального так і горизонтального масштабування.

Розглянемо переваги P2P архітектури:

– досить недорога вартість розробки;

– відсутність серверів оскільки система є децентралізованою;

– кожен вузол має можливість запитувати та надавати послуги.

Отже, можемо підсумувати результати та визначити те що в даному випадку нам підходить саме клієнт-серверна архітектура, оскільки масштабування системи та безпека користувачів є досить важливими факторами. Дана архітектура забезпечить стабільність роботи системи, а недоліки які вона має не є значними, оскільки навіть такий недолік як централізованість системи можна виправити за допомогою декількох серверів, тоді при неполадках на першому сервері, запити клієнтів будуть оброблятись іншим сервером. Також дана архітектура потенційно може бути змішаною з використанням мікросервісної архітектури тощо.

Клієнт-серверна архітектура має дві основні одиниці, а саме сервер та клієнт. Ідея полягає в тому що сервер надає певні послуги, а клієнт може їх використовувати. Це в основному і є головним правилом даної архітектури. Клієнт виконує такі функції як відправлення запитів, отримання даних з запитів, їхня обробка та візуалізація за допомогою інтерфейсу користувача. В той час сервер обробляє запити, зберігає та обробляє дані.

Клієнтський застосунок може бути реалізовано по різному, як мобільний застосунок, як веб-застосунок, так і десктопний застосунок. Цей застосунок повинен використовуватись користувачем для взаємодії з сервером.

Взаємодія між клієнтом та сервером зазвичай відбувається за допомогою певного мережевого протоколу наприклад TPC/IP, UPD.

Реалізувати клієнт-серверну архітектуру можна за допомогою використання багаторівневої архітектури. Розглянемо для початку простий варіант цієї архітектури, а саме трирівневу. В даному випадку у нас є 3 основних компонента, а саме рівень графічного (інтерфейсного) представлення, рівень бізнес-логіки та рівень даних. Простими словами графічний рівень є клієнтським додатком, який звертається до рівня бізнес-логіки, тобто серверу щоб отримати певні дані, а сервер в свою чергу звертається до рівня даних, тобто сервер бази даних. В багаторівневій архітектурі зазвичай передбачається те що кожен рівень комунікує лише з наступним та попереднім рівнем. Зображення трирівневої архітектури показано на рисунку 2.1.

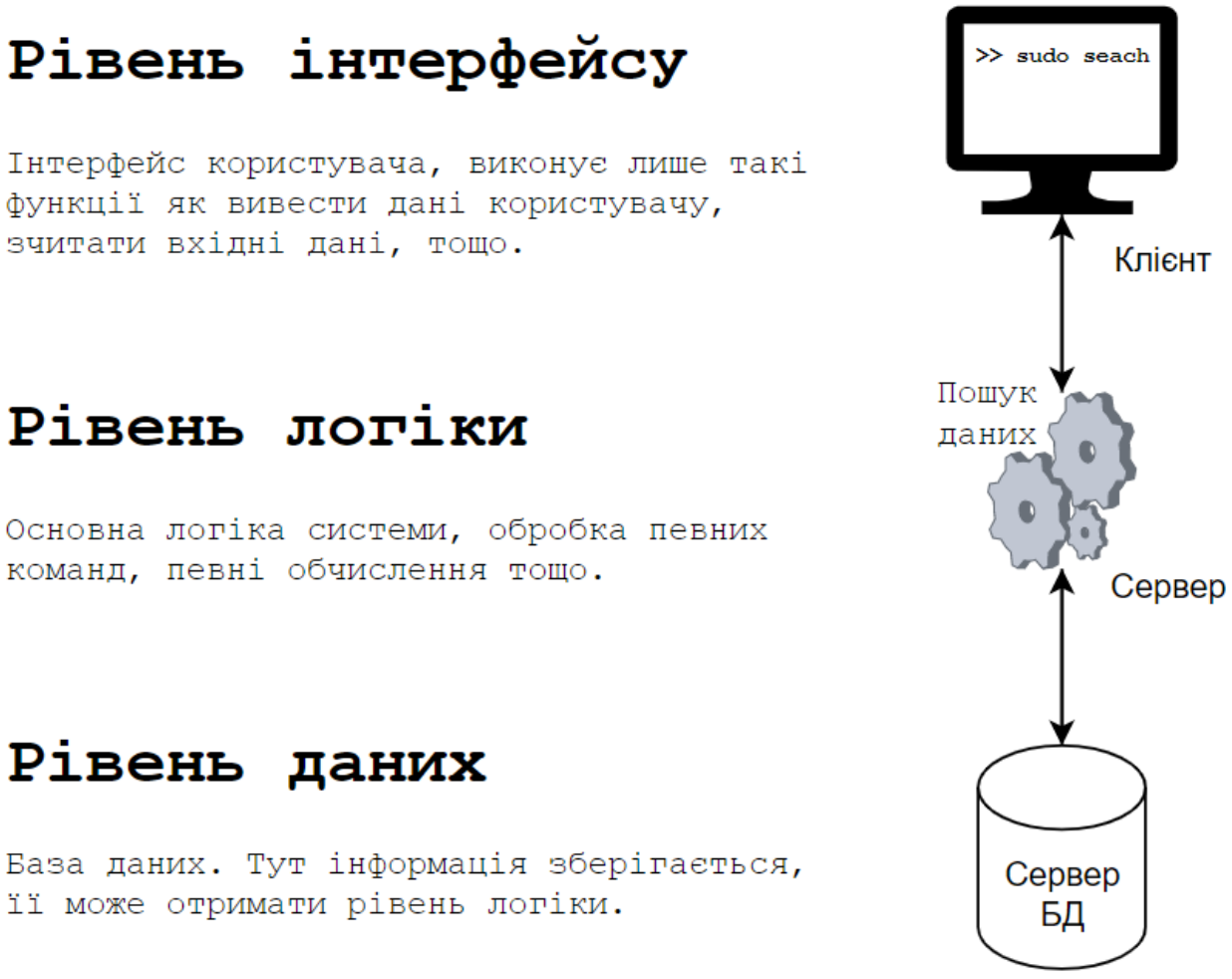



Рисунок 2.1 − Трирівнева архітектура.

Отже, вирішено те, що буде використовуватись клієнт-серверна архітектура та багаторівнева архітектура для розробки майбутньої системи. Далі варто розглянути певні шаблони проектування, оскільки правильно вибрані шаблони та патерни в майбутньому допоможуть скоріше та зручніше розробити надійну систему, а також легко її підтримувати та масштабувати.

Оскільки використовуємо клієнт-серверну архітектуру, то будемо розробляти серверну частину, а також клієнтський застосунок, тому можна одразу розглянути шаблони проектування MVC та MVVM.

Розпочнемо з огляду шаблону MVC (Рисунок 2.2). Даний патерн досить часто використовується під час розробки та проектування програмного забезпечення. Він ділить систему на 3 компоненти, а саме модель, контролер, представлення. Представлення може бути будь-яким представленням інформації, контролер отримує вхідні дані та вирішує, які команди виконувати та керує моделлю. Модель в той час відповідає за зберігання даних та їх структуру.

Метою даного шаблону є гнучкий дизайн програмного забезпечення, який полегшує майбутні зміни та розширення програми, а також дозволяє перевикористовувати деякі компоненти програми.

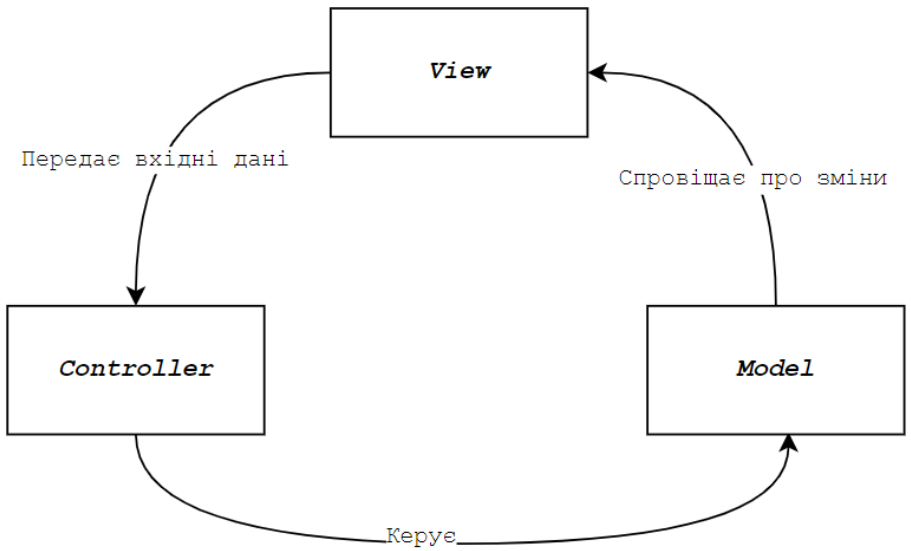



Рисунок 2.2 − Шаблон MVC.

Наступний шаблон MVVM використовується під час проектування та створення клієнтських додатків. Даний патерн полегшує відокремлення розробки графічного інтерфейсу від розробки бізнес логіки. Коли мова йде про розробку саме клієнтського додатку MVVM куди зручніше використовувати замість MVC, оскільки він дозволяє розділити працю дизайнера та програміста, а також якщо в додатку присутній data-binding (зв'язування даних), тобто коли дані зв'язуються з візуальними елементами в обидві сторони.

Шаблон MVVM (Рисунок 2.3) має три основних компонента:

– модель (Model). В цілому те саме що й в MVC, дані необхідні для роботи додатку, системи;

– вигляд (View). Графічний інтерфейс;

– модель вигляду (ViewModel). Абстракція над виглядом та в той же час обгортка для моделі. Призначена для здійснення зв'язку між моделлю та виглядом, відслідковування зміни даних, а також виконання логіки роботи вигляду. Також вміщує у собі певні команди, які вигляд використовує щоб впливати на модель.

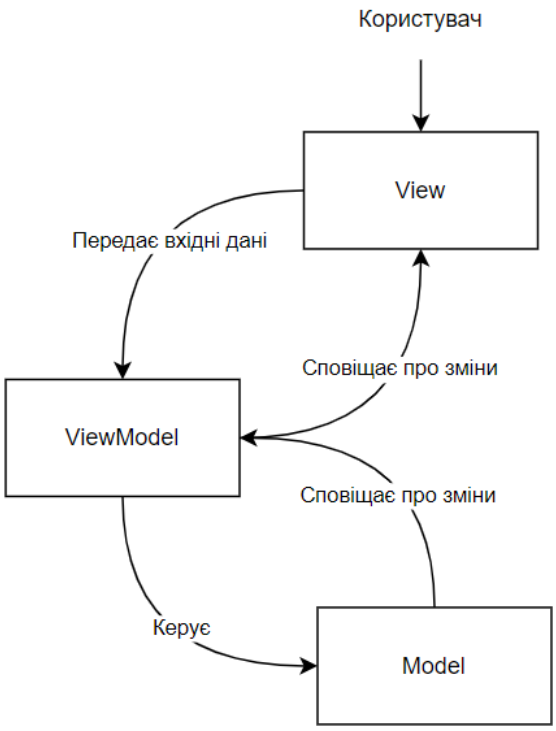



Рисунок 2.3 − Шаблон MVVM.

Також варто розглянути один з досить важливих шаблонів проектування, а саме mediator (посередник). Патерн посередник змушує об’єкти спілкуватися через окремий об’єкт-посередник, який знає, кому потрібно перенаправити той або інший запит. Завдяки цьому компоненти системи залежатимуть тільки від посередника, а не від десятків інших компонентів. Зображення роботи шаблону mediator показане на рисунку 2.4.

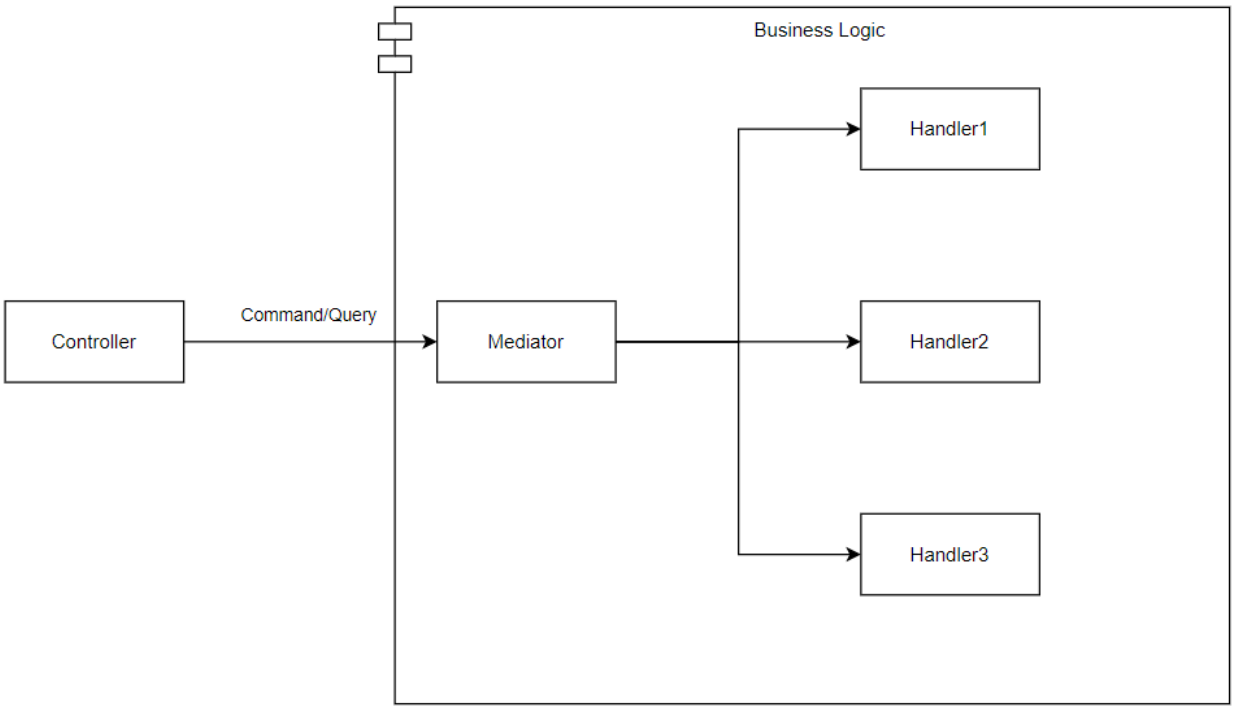



Рисунок 2.4 − Шаблон посередник (mediator).

Оскільки ми проектуємо веб-систему, яка включатиме логіку обробки веб-запитів, як на стороні серверу, так і на стороні клієнту, варто розглянути ще один досить важливий шаблон проектування, а саме Chain of Responsibility (ланцюг обов’язків). Зазвичай під час обробки веб-запитів на стороні серверу необхідно додати різну логіку, наприклад перевірити те, чи авторизований користувач, те чи знаходиться він в своїй ролі, чи ще якусь подібну логіку, але в той же час треба зробити це лаконічно, щоб довелося описати цю перевірку лише один раз і використовувати її повторно багато разів. Те саме стосується й клієнтського застосунку у випадку коли при розробці хочемо використати

наприклад, декілька схем аутентифікації такі як JWT та звичайні сесії та додати ще якусь логіку до попередньої обробки веб-запиту, перед тим, як відправити його серверу.

Патерн пропонує зв’язати всі об’єкти обробників в один ланцюжок. Кожен обробник міститиме посилання на наступного обробника в ланцюзі. Таким чином, після отримання запиту обробник зможе не тільки опрацювати його самостійно, але й передати обробку наступному об’єкту в ланцюжку. Також варто зазначити що обробник не обов’язково має передавати запит далі. Для прикладу якщо користувач не авторизований то обробник одразу може перервати ланцюг щоб уникнути подальших перевірок, оскільки вони більше не мають сенсу. Зображення шаблону показане на рисунку 2.5.

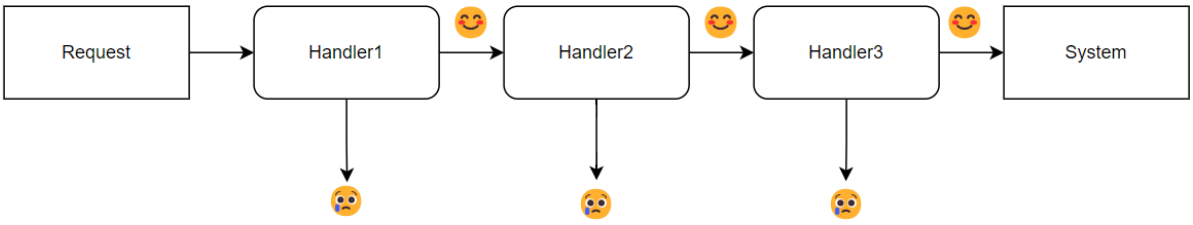



Рисунок 2.5 − Шаблон chain of responsibility.

Отже, для розробки серйозного клієнтського додатку варто застосувати шаблон проектування MVVM, оскільки він є гнучким та зручним для розробки та підтримки додатків, реалізує зв’язування даних, а також дозволяє розділити роботу програміста та дизайнера. В той час для розробки серверної частини в цілому буде хорошим рішенням використати шаблон MVC, оскільки серверна частина це в основному сервіси WebAPI, тому там немає необхідності в інтерфейсі користувача, а тому й немає необхідності у використанні MVVM.

Також під час розробки будуть використані такі шаблони проектування як Mediator та Chain of Responsibility.

## 2.2 Декомпозиції та детальне проектування

Наступним етапом після проектування архітектури програмного забезпечення є детальне проектування, що допоможе підготувати завдання роботи до його розробки та реалізації.

Спершу варто розпочати проектування з бази даних, оскільки це є одною з найважливіших частин в програмному забезпеченні. Всі наступні етапи розробки програмного продукту будуть базуватись на тому з якими даними ми будемо працювати, тому етап проектування бази даних має виконуватись перш за все інше.

Для проектування бази даних варто використати модель сутність-зв'язок або як вона ще називається ER-модель. Дана модель представляє будь-що об’єктами, які мають певні атрибути (властивості об’єкту) та зв’язки встановлені з іншими об’єктами.

В даному випадку у нас будуть наступні об’єкти: User, Session, Chat, Message. Також є об’єкт UsersChats який допомагає зв’язати сутність User і Chat за допомогою відношення багато-до-багатьох.

Тобто в результаті маємо 4 основні сутності:

– User (користувач додатку);
– Session (сесія користувача);
– Chat (певний чат);
– Message (повідомлення).

Зображення ER-діаграми показано на рисунку 2.6.

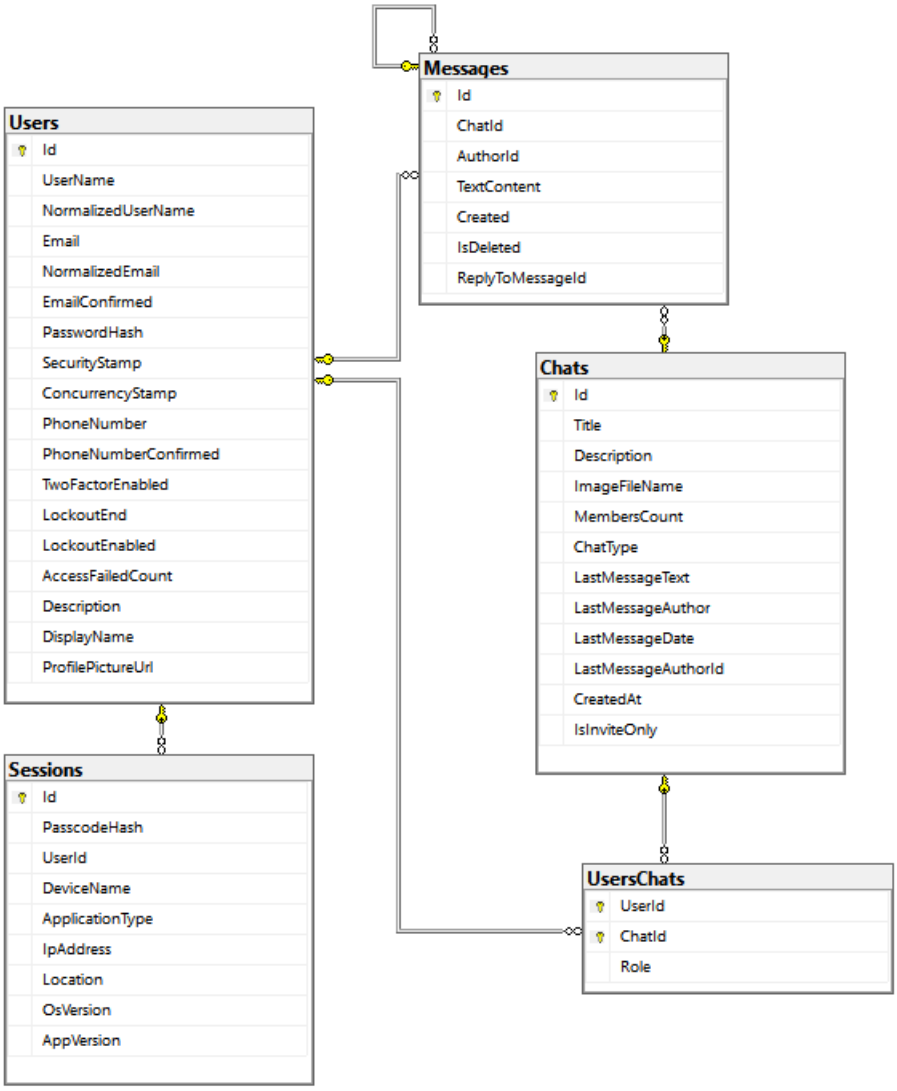



Рисунок 2.6 − ER-діаграма бази даних.

Отже, далі можна почати проектувати інтерфейс користувача та способи взаємодії з ним.

Даний етап досить важливий оскільки інтерфейс користувача це саме те з чим буде взаємодіяти користувач. Для більшості користувачів сам інтерфейс і є програмою, тому він має бути досить простим та зрозумілим.

Інтерфейс користувача буде складатись з наступних сторінок:

– сторінка авторизації;

– сторінка реєстрації;

– сторінка відновлення паролю;

– сторінка з чатами та повідомленнями;

– сторінка налаштувань користувача.

Також окрім сторінок будуть окремі компоненти, наприклад у вигляді модального вікна:

– компонент для детального перегляду інформації про чат;

– компонент для перегляду сесій користувача та їх налаштувань;

– компонент для пошуку користувачів та запрошення їх в чат;

– компонент для пошуку публічних чатів.

Далі розглянемо всі ці компоненти та їх призначення більш детально.

Сторінка авторизації − зазвичай це початкове вікно з якого починається навігація користувача в додатку. Містить поля UserName та Password для введення користувачем, а також посилання для навігації до реєстрації нового акаунту та відновлення паролю.

Сторінка реєстрації − сторінка для реєстрації нового користувача. Містить наступні поля для введення даних: UserName (унікальне ім’я користувача), DisplayName (ім’я користувача яке зазвичай буде відображати в додатку), Email (пошта користувача) та Password (пароль користувача).

Сторінка відновлення паролю − сторінка з відновленням паролю за допомогою введення свого поштового адресу.

Сторінка з чатами та повідомленнями − основна сторінка з якою користувач взаємодіє, оскільки містить список чатів користувачів, а також контейнер з повідомленнями, поле для введення нового повідомлення для вибраного чату, а також інші компоненти для взаємодії з чатом. На даній сторінці можна вийти з чату за допомогою контекстного меню або додати нового користувача в чат. Також якщо відкрити компонент для перегляду детальної інформації про чат, можна завантажити фото для чату.

Сторінка налаштувань користувача − дана сторінка містить дані користувача, такі як UserName, Email, DisplayName, дозволяє змінити пароль, завантажити фото для профілю. Також з цієї сторінки можна відкрити список

сесій користувача щоб переглянути та видалити їх (якщо користувач бажає видалити сесії наприклад для пристрою, який він загубив).

Основна навігація в додатку відбувається за допомогою Sidenav Menu. Даний тип меню зазвичай знаходиться в лівій частині додатку, та може бути прихованим. Там знаходяться кнопки для навігації до налаштувань користувача, чатів, а також деякі кнопки з певними функціями, такі як створення нового чату чи пошук існуючого публічного чату.

Графічне зображення навігації в клієнтському додатку користувача показано на рисунку 2.7.

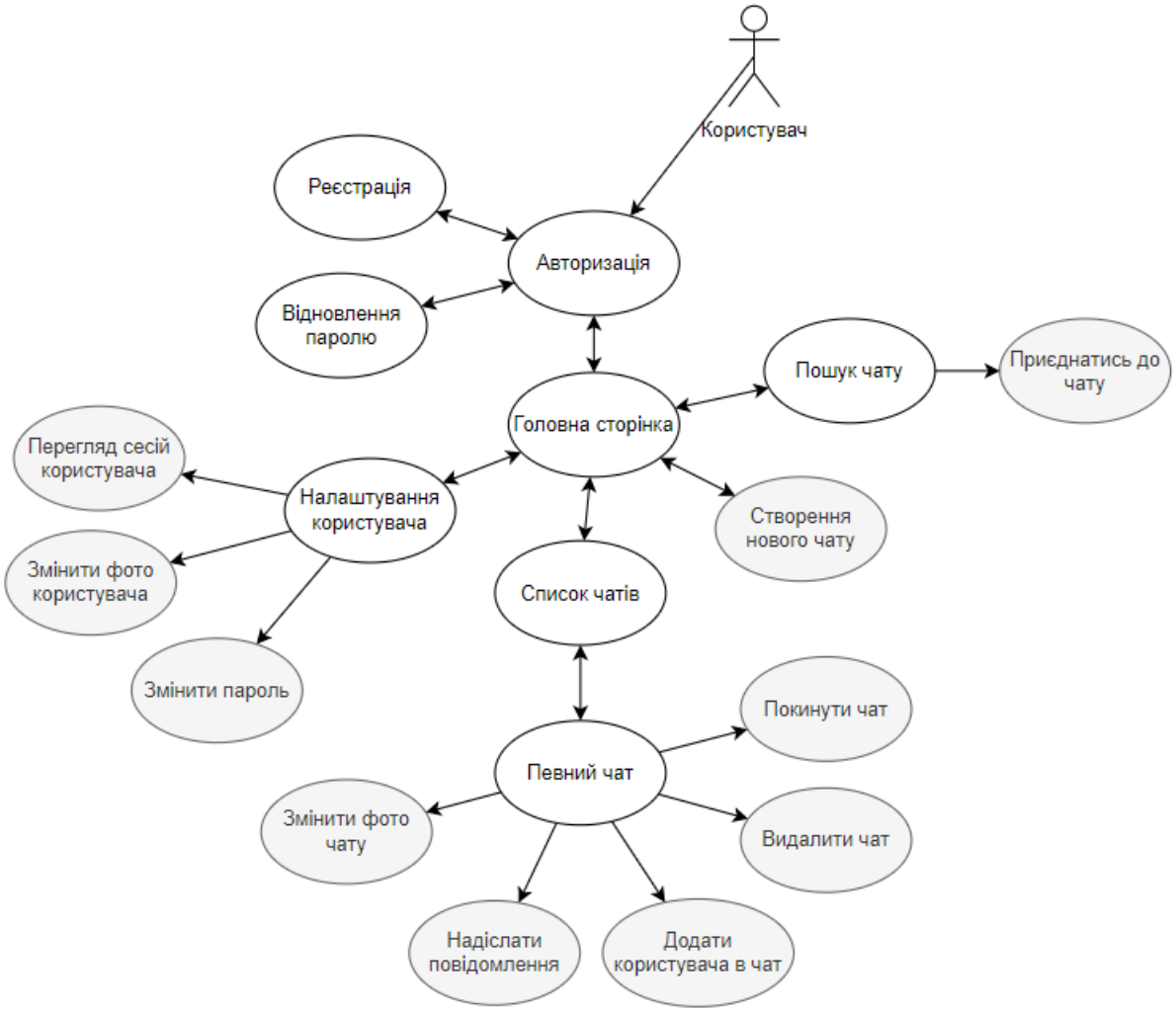



Рисунок 2.7 − Графічне зображення навігації в додатку користувача.

Далі розглянемо ще те як підтримувати real-time спілкування між клієнтом там сервером. Для цього буде використаний додатковий модуль, а саме бібліотека SignalR. Дана бібліотека дозволяє серверному коду надсилати

асинхронні сповіщення клієнтським додаткам. Це досить спрощує розробку додатків оскільки дана бібліотека має декілька механізмів для цього процесу, а саме з використанням веб-сокетів та з використанням стандартного http, для клієнтів, які не підтримують веб-сокети, але в даному випадку ми зосередимось на веб-сокетах. Дана бібліотека також містить в собі імплементований шаблон RPC (remote procedure call), що дозволить викликати деякі процедури як зі сторони клієнта на стороні сервера, так і в зворотньому порядку. Схема RPC зображена у вигляді діаграми послідовності на рисунку 2.8.

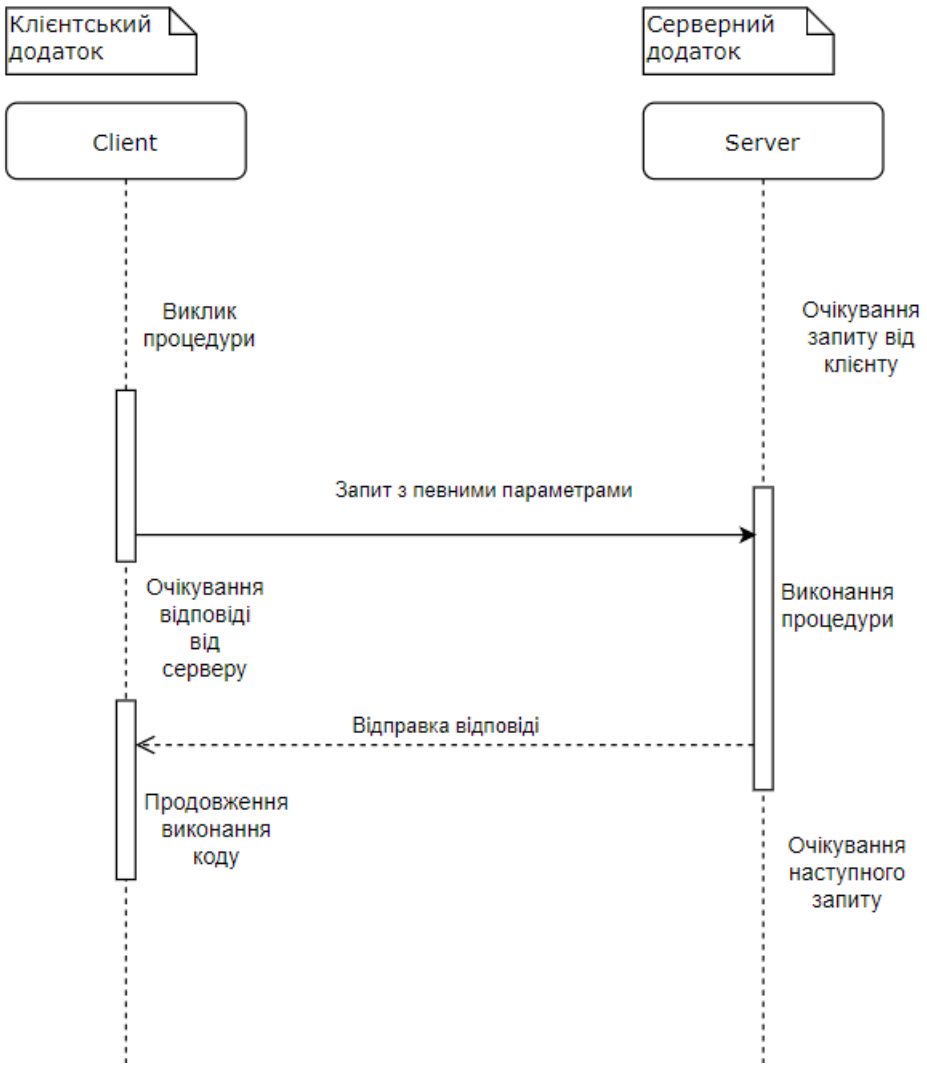



Рисунок 2.8 − Схема RPC.

Даний підхід з RPC працює в обидві сторони, як від сервера до клієнта так і від клієнта до сервера. Перш за все використання даної бібліотеки і її переваг дозволить нам сповіщати клієнтів про зміни в чаті, такі як створення нового

повідомлення і реагувати на це з клієнтського коду, одразу ж додаючи нове повідомлення в чат.

Також SignalR містить механізм груп, в які можна додавати певні з'єднання, в даному випадку це підключені до веб-сокетів клієнти. Їх ми будемо додавати в групи по ідентифікатору чату, з якого клієнт хоче отримувати сповіщення про зміни. Спершу клієнт викликає відповідний метод на сервері, який додає користувача в групу по ідентифікатору чату. Далі якщо нове повідомлення додається в чат, сервер сповіщає клієнтів підключених до даного чату про нове повідомлення. Схема роботи даного процесу зображено на рисунку 3.5.

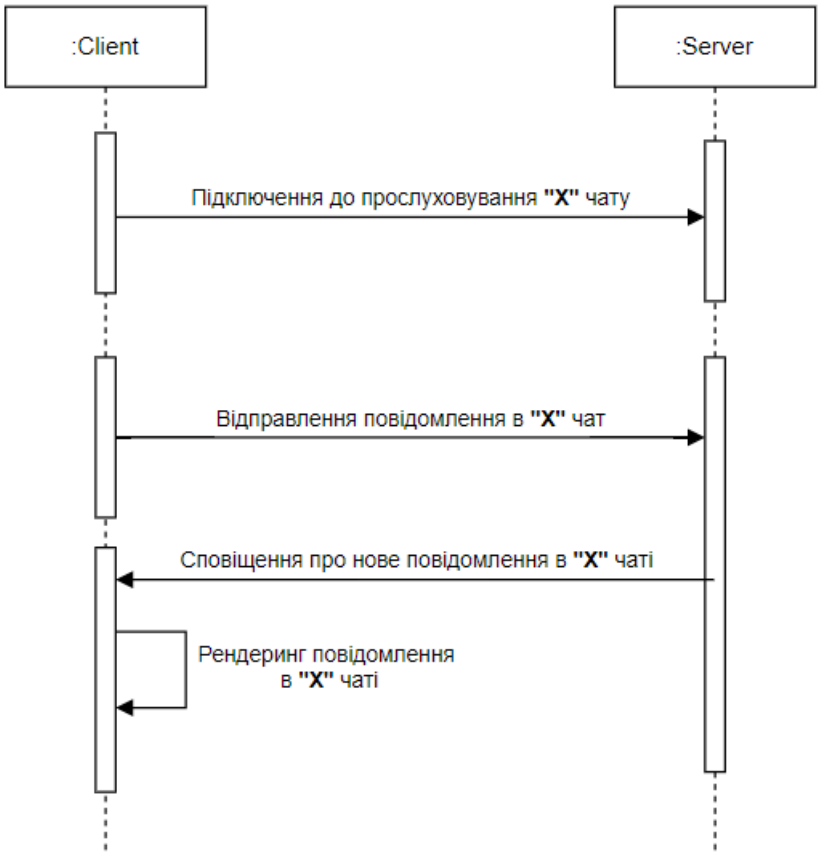



Рисунок 2.7 − Схема прослуховування нових повідомлень в чаті.

Далі розглянемо серверну частину. Серверна частина є рівнем логіки в нашій трирівневій архітектурі, а також в той час надає певний програмний інтерфейс для клієнтського додатку. В даному випадку серверна частина буде розглядатись як WebAPI для клієнтського додатку. Також ми вирішили що

будемо використовувати шаблон MVC для розробки серверу, отже можемо зобразити концептуальну декомпозицію додатку серверу.

Сервер має певний набір функцій, які використовуються клієнтським додатком, дані які клієнт передає на сервер обробляються відповідними програмними модулями та в результаті вносять зміни в базу даних або повертають певні дані користувачу.

## 2.3 Аналіз та вибір технологій для реалізації програмної системи

Отже, вибір технологій, те від чого залежить наскільки зручною буде розробка та масштабування додатку. Оскільки ми передбачаємо те що буде розроблятись як серверна частина так і клієнтський застосунок, потрібно вибрати відповідні технології для розробки кожного з цих елементів.

Розпочнемо з серверної частини. В даному випадку є такий варіант як використання фреймворку ASP.NET Core на платформі .NET 7 та мови програмування C#.

Фреймворк ASP.NET Core дозволяє зручно розробляти веб-сервіси, такі як RestAPI або просто WebAPI, використовувати протокол передачі даних HTTP(S) та WebSockets. Дана технологія досить широко використовується у багатьох підприємствах, оскільки розроблювані програмні продукти досить легко масштабувати та підтримувати. Платформа .NET постійно розвивається та доступна на основних популярних платформах. Остання версія на даний момент поки що .NET 7. В ній було додано досить багато різних можливостей, наприклад такі як Rate Limits, Native AOT (Ahead-of-time) та в цілому покращення продуктивності платформи. Native AOT поки що доступний лише для консольних додатків, але це є досить потужною технологією оскільки дозволяє компілювати нативний застосунок під конкретну платформу, наприклад під Linux, що буде дуже позитивно впливати на продуктивність додатку, швидкість його роботи, а також використання ресурсів.

Мовою програмування для даної платформи та фреймворку є C#, об’єктно-орієнтована мова програмування, яка має досить багато можливостей. Дана мова програмування є досить популярною протягом останніх років та основною платформою для неї є .NET.

Мова програмування використовується для вирішення наступних задач:

– веб-розробка, оскільки потужний фреймворк ASP.NET дозволяє досить легко розробляти, підтримувати та масштабувати у сфері вебу, будь то Інтернет-магазин або певний веб-сервіс, окрім того присутня можливість розробки веб-застосунків з використанням фреймворку Blazor (даний фреймворк дозволяє розробляти веб-застосунки з використанням web-assembly);

– Windows застосунки, досить часто розробляють за допомогою даної мови програмування та технології WPF, проте за допомогою деяких інших технологій розробляти додатки можна не тільки для ОС Windows, але й для Linux та IOS, особливо якщо використовувати не стару версію .NET Framework, а саме .NET Core;

– розробка ігор. За допомогою Unity можна розробляти відеоігри для різних платформ, будь то мобільна гра або гра для настільного ПК.

Сама мова програмування має наступні переваги, які роблять її чудовим вибором для вирішення багатьох завдань:

– швидкий час розробки. C# має досить багато функцій, які дозволяють розробникам скоріше кодувати чим на інших мовах програмування. Деякі з цих функцій включають статично типізовану та легку для читання мову, синтаксис, який виглядає як розширена версія Java, і величезну бібліотеку, наповнену функціональністю високого рівня;

– висока масштабованість. Статичний характер кодування C# перетворює всі його програми на надійні продукти, які можна легко налаштувати та змінити. Це означає, що інженери можуть швидко вносити зміни та надбудовувати будь-яку програму C#, щоб розширити її функціональність і підтримати більше користувачів;

– об’єктно-орієнтована мова програмування. C# охопив об’єктно-орієнтоване програмування таким чином, що це може бути мовою, яка краще його використовує. Насправді об’єктно-орієнтованість дозволяє C# бути високоефективним і надзвичайно гнучким, усе це робить розробку легшою та менш ресурсоємною.;

– досить легка крива навчання. Як мову високого рівня, C# дуже легко вивчити та зрозуміти. І це без урахування багатьох вбудованих функцій, які дуже прості у використанні. Більше того, будь-який інженер, який уже знає C++ або Java, почуватиметься як вдома, коли вперше скористається C#, оскільки ці мови мають багато однакових функцій і загальний підхід до програмування;

– велика спільнота. C# є однією з найпоширеніших мов у світі, а це означає, що є багато розробників C#, готових допомогти вам із нею. це ще не все Будучи продуктом Microsoft, C# має підтримку технічного гіганта, що означає допомогу експертів, додаткові ресурси та часті оновлення.

Для розробки клієнтського додатка варто використати фреймворк Angular та мову програмування TypeScript. Даний фреймворк розробляється та підтримується компанією Google та спільнотою. Angular досить часто використовується підприємствами для розробки складних JavaScript додатків оскільки має досить велику кількість переваг що до інших фреймворків. Він використовує ієрархію компонентів як основну концепцію архітектури, а також MVVM що дозволяє розділити роботу дизайнера та програміста. Модульність Angular дозволяє розробляти новий функціонал досить зручно, не впливаючи вже на існуючий код, а також зручну перевикористовувати деякі компоненти.

Оскільки основною мовою програмування в Angular є TypeScript, але в той же час нам нічого не заважає використовувати JavaScript, проте рекомендовано використовувати саме TypeScript.

Те що ми використовуємо TypeScript дає нам наступні переваги:

– класи, інтерфейси, об’єктно-орієнтоване програмування;

– система типізації. Окрім того що це зручно під час розробки та робить код більш зрозумілим, типізація також допомагає компілятору виявляти помилки на етапі компіляції;

– розширений синтаксис та можливості, які відсутні в JavaScript.

Також варто розглянути деякі алгоритми шифрування, які будуть досить корисними для використання у розроблюваній системі.

Для шифрування даних на стороні серверну доречно буде використати алгоритм AES. Дане шифрування є симетричним, що дозволяє використовувати один і той самий ключ для шифрування та дешифрування даних. Сам алгоритм є досить потужним та захищеним, а те що ми маємо лише 1 ключ для шифрування та дешифрування позитивно вплине на використання ресурсів нашої системи. AES був розроблений для забезпечення найвищого рівня безпеки для найбільш конфіденційних даних. Агентство національної безпеки (NSA) та інші агентства вибрали цей метод як стандарт безпеки уряду США через його потужний, непроникний захист. AES також використовується в багатьох інших державних установах і галузях. Банки та інші фінансові установи покладаються на шифрування AES, щоб захистити особисту інформацію та інформацію про транзакції своїх клієнтів.

Також AES (Advanced Encryption Standard) має наступні переваги:

– для шифрування використовуються ключі вищої довжини, наприклад 128, 192 і 256 біт. Таким чином, це робить алгоритм AES більш надійним проти злому;

– це найпоширеніший протокол безпеки, який використовується для різноманітних програм, таких як бездротовий зв’язок, фінансові операції, електронний бізнес, зберігання зашифрованих даних тощо;

– це одне з найпоширеніших комерційних рішень із відкритим кодом у всьому світі.

– оскільки він реалізований, як в апаратному, так і в програмному забезпеченні, це найнадійніший протокол безпеки.

Але навіть даний алгоритм має свої недоліки, наприклад такі як використання надто простих алгебраїчних структур та те що кожен блок завжди шифрується однаково.

Наступним алгоритмом є алгоритм Diffie-Hellman. Він дозволяє узгоджувати ключі для двох учасників, що дозволить нам реалізувати end-to-end чат. Алгоритм є основою для багатьох протоколів та алгоритмів шифрування.

Даний алгоритм має наступні переваги:

– відправнику та одержувачу не потрібні попередні знання один про одного;

– після обміну ключами передача даних може здійснюватися через незахищений канал;

– спільне використання секретного ключа є безпечним.

Проте даний алгоритм має наступні недоліки:

– його не можна використовувати для підпису цифрових підписів;

– оскільки він не аутентифікує жодну сторону в передачі, обмін ключами Diffie Hellman чутливий до атаки man-in-the-middle;

Дані недоліки алгоритму не є суттєвими у нашому випадку, оскільки цифрові підписи у розроблюваній системі відсутні, а вразливість до атаки man-in-the-middle можна виправити в майбутньому якщо використати зашифроване з'єднання між користувачем та сервером, наприклад той самий HTTPS протокол.

## 2.4. Огляд модулів програмної системи

Проектування модулів програмної системи варто почати з розгляду серверної частини системи. У даному випадку для розробки серверної частини використовується ASP.NET Core фреймворк, а сам серверний застосунок буде представлено у вигляді API, з яким клієнт може обмінюватись даними використовуючи протокол HTTPS та WebSockets.

Всього серверний застосунок було розділено на три основних модуля, а також деякі допоміжні модулі. Основними модулями є:

– Presentation. Даний модуль містить перш за все контролери, типи даних, які є вхідними та вихідними для контролерів, певну конфігурацію додатку та pipeline (конвеєр) для обробки запитів до серверу перед тим як вони будуть направлені до обробників бізнес-логіки;

– BusinessLogic. Даний модуль містить всю бізнес логіку додатку, команди та запити, моделі даних, які стосуються саме предметної області;

– Data. Даний модуль містить всю модель даних та її конфігурацію.

## 2.5 Висновки

Отже було проведено проектування системи. Було вибрано декілька патернів та шаблонів проектування, а саме MVVM, MVC, багаторівневу архітектуру та клієнт-серверну архітектуру. Були розглянуті такі патерни як Mediator та Chain of Responsibility, які допоможуть під час проектування та розробки такої складної системи як месенджер.

Потім було розглянуто декілька алгоритмів шифрування, серед яких основними були AES (Advanced Encryption Standard) та протокол Діффі-Геллмана. Провівши детальний аналіз та порівнявши їхні переваги та недоліки було визначено що AES є досить потужним алгоритмом шифрування, який є сенс використати в нашій системи. Також протоколу Діффі-Геллмана є місце у нашій системі, оскільки він дозволяє обмінюватись ключами, що є досить корисним для розробки end-to-end шифрування.

Далі за допомогою декомпозиції та детального проектування було спроектовано інтерфейс користувача та ER-діаграму бази даних. Всі сутності були описані в реляційній моделі даних.

Результатом виконаної став вибір технологій розробки під час якого було обрано мову програмування C# та фреймворк AspNetCore для розробки серверної частини системи. Для розробки клієнтського веб-застосунку було

обрано мову програмування TypeScript та фреймворк Angular. Також було розглянуто потенційні алгоритми для шифрування, а саме AES та Diffe-Hellman.

Зроблено огляд програмних модулів, які будуть використані при розробці програмного застосунку.

# 3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

## 3.1 Детальне проектування модулів

Спираючись на вже розглянути модулі програмної системи у п.2.4., почнемо докладніше працювати з ними.

Розпочнемо огляд модуля Presentation. Як вже було вказано, всі контролери знаходяться саме в цьому модулі. Там же розміщаються типи даних, які будуть для контролерів як вхідними, так і вихидними. Конфігурація та конвеєр (pipeline), який призначений для обробки запитів до серверу перед тим як вони будуть направлені до обробників бізнес-логіки теж містяться в модулі.

Даний модуль відповідає за представлення нашого сервера, а саме описує те які в нас є HTTP запити, як вони обробляються, які типи вхідних та вихідних даних використовують ці запити, конвеєр обробки запиту до того як він перейде в обробку бізнес-логікою.

Розглянемо конвеєр обробки HTTP запиту, то все починається з запиту від клієнта, далі сервер приймає даний HTTP запит, та намагається проаналізувати який метод викликає клієнт. Якщо обробник знайдено далі йде опціональна перевірка на авторизацію користувача, а потім валідація даних, які передав клієнт. У разі позитивної перевірки запиту на всіх етапах конвеєру, запит передається до подальшої обробки в бізнес-логіку.

Те, як працює конвеєр обробки HTTP запиту до передачі подальшої обробки логіки в модуль бізнес-логіки зображено на рисунку 3.1.

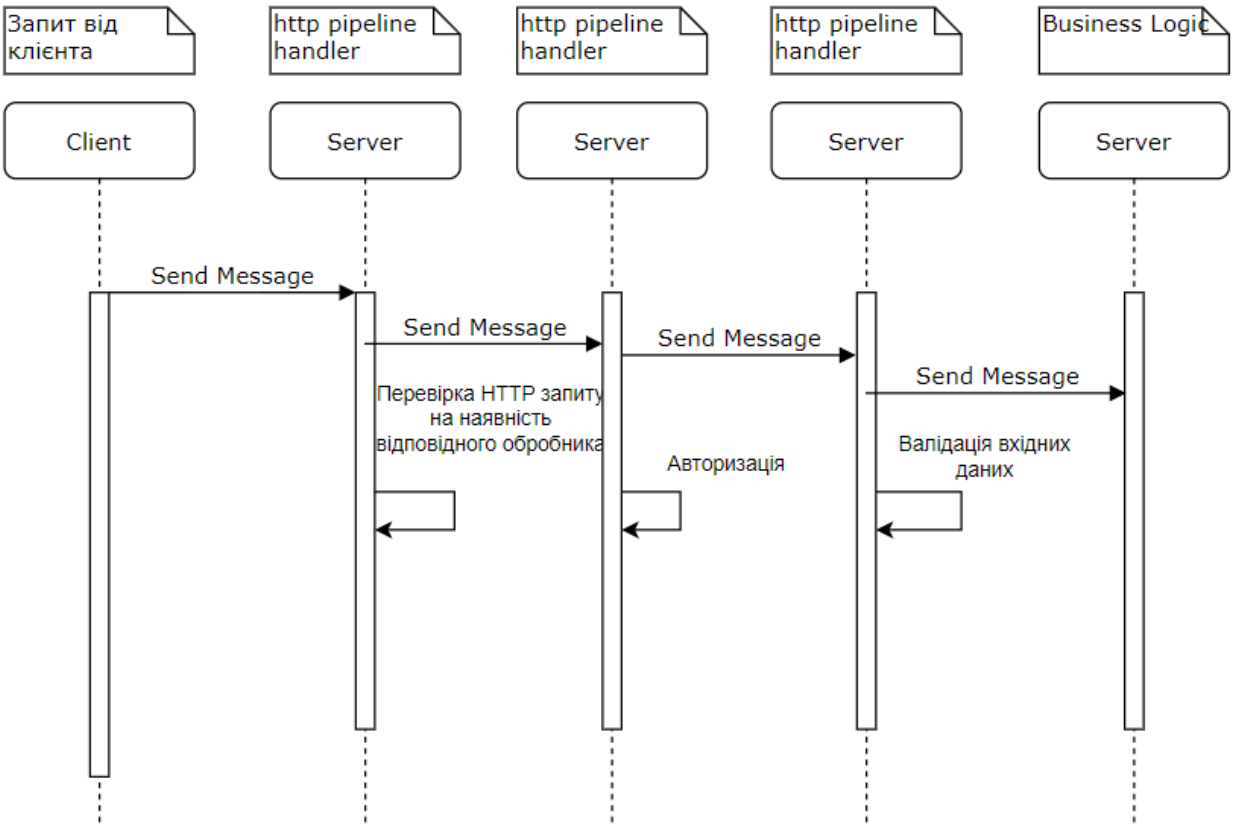



Рисунок 3.1 − Обробка HTTP запиту в конвеєрі.

Також даний модуль містить певні конфігурації додатку, такі як конфігурація з'єднань до бази даних, авторизації тощо. Конфігурація буде братись з json файлів та змінних середовища.

Для авторизації буде використано схеми JWT та сесії. Для більшості операцій буде використовуватись саме JWT авторизація, а для отримання токену JWT та деяких специфічних операцій схема з сесіями. Сама сесія для клієнта представлена у вигляді токена, який складається з двох частин, а саме ідентифікатора сесії та паролю сесії. Сама сутність сесії знаходиться в базі даних, та містить певну інформацію, таку як назва пристрою з якого був виконаний вхід, IP адресу, локацію, версію ОС, версію додатку та тип додатку. Необхідно це для того щоб користувач міг переглядати активні сесії та по необхідності видаляти їх, наприклад якщо він втратив доступ до пристрою з якого був авторизований у додатку тощо. Структура сутностей сесії та користувача зображена на рисунку 3.2.

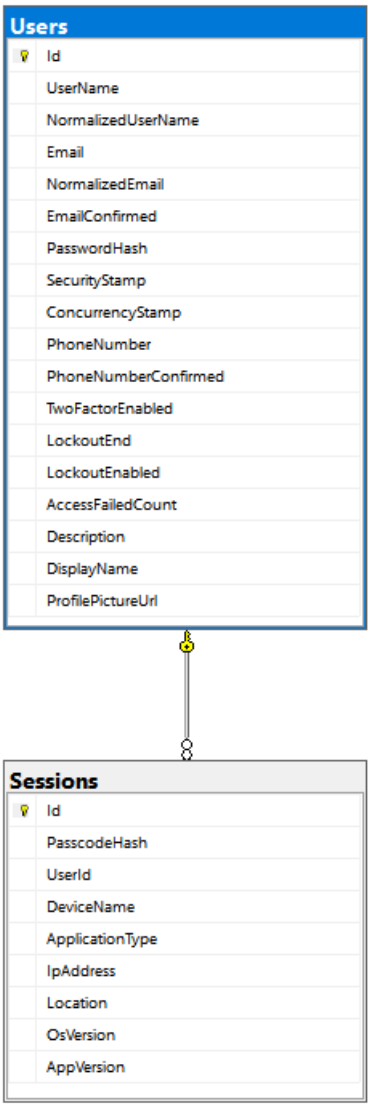



Рисунок 3.2 − ER схема сесії та користувача.

Модуль бізнес-логіки основний модуль, який вміщує та описує всі команди, запити та їх обробники. Команди та запити в даному випадку відповідно до цього модулю це певні дії, які описані з точки зору предметної області, наприклад як команда відправлення повідомлення в чат, або команда створення чату. Різниця між командою і запит в тому, що команда є певною дією, яка вносить певні зміни в систему, в той час як запит (не плутати з http запитом, оскільки даний контекст це бізнес-логіка) це певна дія, яка лише отримує дані без змін в системі. Доступ до бізнес логіки буде виконуватись за допомогою використання шаблону медіатор.

Як раніше було зазначено, даний модуль містить команди та запити спроектовані самі під предмету область, тому в даній частині будуть знаходить команди та запити сегреговані по наступним групам:

– account (команди та їх обробники, які стосуються маніпуляцій з акаунтом користувача);

– chats (команди та їх обробники, які стосуються маніпуляцій з чатами);

– messages (команди та їх обробники для керування повідомленнями);

– profile (команди та їх обробники, які керують профілем користувача);

– users (запити та їх обробники для отримання публічної інформації про користувачів);

Команда чи запит самі з себе представляють певний об’єкт, який має певний обробник, а також об’єкт результату.

Наступним модулем є Data. Даний модуль містить все що нам необхідно для схеми бази даних, в даному випадку це типи сутностей бази даних, їх конфігурації та контекст бази даних, оскільки ми будемо використовувати ORM для роботи з реляційною БД.

Список сутностей бази даних:

– ChatEntity (чат);

– MessageEntity (повідомлення);

– SessionEntity (сесія користувача);

– UserChatEntity (користувач-чат);

– UserEntity (користувач).

Відповідно до кожної з цих сутностей буде також конфігурація, яка визначатиме певні властивості моделі з точки зору БД, від назви таблиці в БД до обмеження довжини певного поля таблиці.

В результаті маємо певний контекст бази даних, який вміщує множини сутностей бази даних:

```
public sealed class AppDbContext : DbContext
{
   public DbSet<UserEntity> Users { get; set; }
```

```
public DbSet<SessionEntity> Sessions { get; set; }

public DbSet<ChatEntity> Chats { get; set; }

public DbSet<MessageEntity> Messages { get; set; }

public DbSet<UserChatEntity> UsersChats { get; set; }

public AppDbContext(DbContextOptions options) : base(options)
{
}

protected override void OnModelCreating(ModelBuilder modelBuilder)
{
   base.OnModelCreating(modelBuilder);
   modelBuilder.ApplyConfigurationsFromAssembly(typeof(AppDbContext).Assembly);
}
      }
```

Далі завдяки тому що це ORM, ми зможемо працювати з сутностями бази даних як з об’єктами.

Далі розглянемо детальне проектування клієнтського додатку. Всього застосунок буде складатись з двох основних частин, першою є імплементація API для взаємодії з сервером, другою частиною є вже сам застосунок.

API додатку вміщує всі необхідні типи даних, які приймає чи повертає як результат сервер, аутентифікацію та сервіси. В даному випадку кожен сервіс API буде містити необхідні методи, які в свою чергу звертаються до webAPI за допомогою HTTP.

Далі сам застосунок, він буде імплементований всього як один модуль та містить наступні основні частини:

– Components (компоненти Angular, є певною частиною UI, яка має свій код та розмітку);

– Guards (об’єкт за допомогою якого виконуються обмеження навігації по додатку для користувача);

– Interceptors (HTTP обробники, які представляють з себе імплементацію шаблону ланцюг відповідальностей (Chain-of-responsibility), дозволяють обробляти HTTP запити до того як вони будуть направлені до серверу,

наприклад підставивши в HTTP хедер JWT токен аутентифікації та у разі відповіді 401 запросити новий JWT токен та повторити запит);

– Dialogs (теж компоненти, але які представлені у вигляді діалогових вікон);

– Services (сервіси додатку, вміщують всю основну логіку додатку, в той час як компоненти відповідають лише за візуальний вигляд додатку);

– Validators (валідатори для вхідних даних від користувача, як асинхронні так і синхронні, наприклад валідація паролю користувача на відповідність шаблону потужного паролю або валідація пошти);

– Pages (компоненти, які представляють з себе певну сторінку, змінюють один одного в залежності від навігації користувача по додатку);

В результаті маємо приблизний шаблон додатку, де всі частини розділені по своїм обов’язкам. Angular дозволяє імплементувати частину додатку, яка пов’язана з API у вигляді окремого модуля, але це в цілому не обов'язково і пов’язано з модулями Angular. Модулі Angular дозволяють розділити застосунок на частини, які взаємодіють один з одним та в результаті представляють один цілий застосунок.

В нашому випадку всього є два основних розділення логіки, це сам застосунок та API. Спочатку можна розробити API паралельно з серверною частиною, протестувати їх, а далі розробляти сам клієнтський застосунок, який в свою чергу буде використовувати готовий модуль API для взаємодії з сервером. Схема яка показує основні складові додатку зображена на рисунку 3.3.

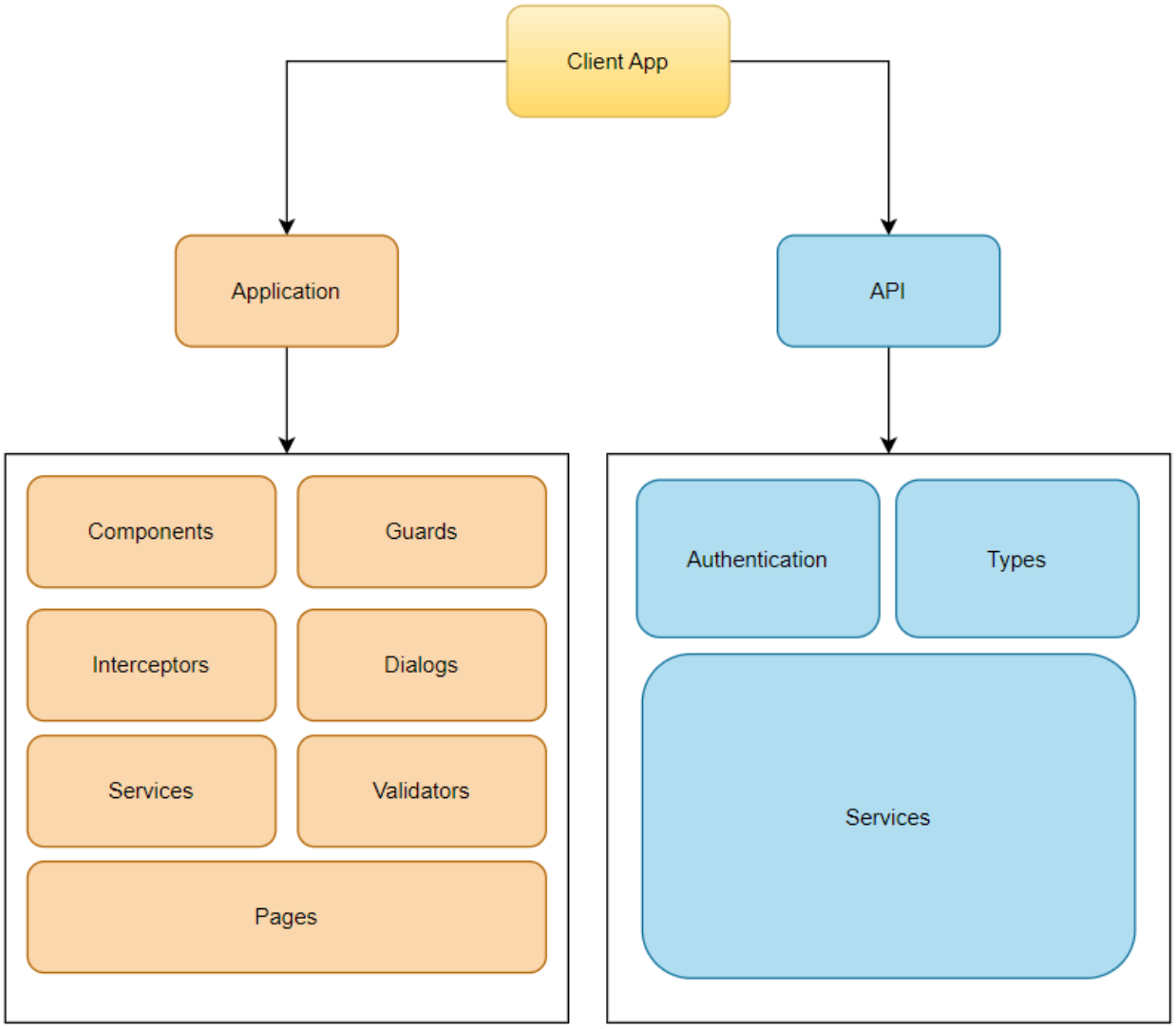



Рисунок 3.3 − Основні складові клієнтського веб-застосунку.

## 3.2 Програмна реалізація модулів.

Спершу почнемо з реалізації серверної частини. Почнемо з реалізації модулю Data, оскільки він самий простий в реалізації. Всього було розроблено декілька типів, які відповідають сутностям в базі даних, їх конфігурації та контекст бази даних. Для взаємодії з базою даною використовується ORM технологія, а саме EntityFramework Core та база даних SqlServer.

Контекст бази даних містить поля типу DbSet<T>, де T це тип нашої сутності, наприклад UserEntity або ChatEntity, а також маємо метод OnModelCreating в якому ми конфігуруємо модель бази даних, в даному

випадку ми робимо це за допомогою класів-конфігурацій та рефлексії, яка в свою чергу аналізує ці класи-конфігурації. В цілому конфігурувати повність всю модель бази даних можна в методі OnModelCreating без використання класів-конфігурацій, але набагато краще розділити конфігурації, щоб потім легше орієнтуватись в коді.

Маємо розроблений контекст бази даних:

```
public sealed class AppDbContext : DbContext
{
   public DbSet<UserEntity> Users { get; set; }

   public DbSet<SessionEntity> Sessions { get; set; }

   public DbSet<ChatEntity> Chats { get; set; }

   public DbSet<MessageEntity> Messages { get; set; }

   public DbSet<UserChatEntity> UsersChats { get; set; }

   public AppDbContext(DbContextOptions options) : base(options)
   {
   }

   protected override void OnModelCreating(ModelBuilder modelBuilder)
   {
      base.OnModelCreating(modelBuilder);
      modelBuilder.ApplyConfigurationsFromAssembly(typeof(AppDbContext).Assembly);
   }
         }
```

Також розглянемо конфігурацію для чату. Даний клас потрібний щоб налаштувати необхідні поля таблиці в базі даних. Відповідно для кожної сутності бази даних потрібно розробити конфігурацію по даному прикладу.

Клас ChatEntityConfiguration:

```
internal sealed class ChatEntityConfiguration : IEntityTypeConfiguration<ChatEntity>
{
   public void Configure(EntityTypeBuilder<ChatEntity> builder)
   {
      builder.HasKey(x => x.Id);

      builder.Property(x => x.Title).IsRequired().HasMaxLength(64);

      builder.Property(x => x.Description).IsRequired(false).HasMaxLength(256);

      builder.Property(x => x.ImageFileName).IsRequired(false);

      builder.Property(x => x.MembersCount).IsRequired();

      builder.Property(x => x.ChatType).IsRequired();

      builder.Property(x => x.LastMessageText).IsRequired(false);
```

```
        builder.Property(x => x.LastMessageAuthor).IsRequired(false);

        builder.Property(x => x.LastMessageDate).IsRequired(false);

        builder.Property(x => x.IsInviteOnly).IsRequired(true);
    }
        }
```

Отже маючи готовий контект бази даних, можна приступати до розробки основних модулів. Почнемо з конфігурації aspnetcore додатку та розробки контролерів.

Сам застосунок aspnet починається з конфігурації певних компонентів, таких як визначення контексту бази даних, конфігурації контролерів, додання певних сервісів в DI та інше. Все це відбувається у файлі Program.cs.

Приклад конфігурації контексту бази даних:

```
builder.Services.AddDbContext<DreamNetworkContext>(options =>
{
    options.UseSqlServer(EnvironmentConstants.SqlConnectionString());
});
```

Далі необхідно розробити контролери, які будуть оброблювати запити від клієнта. Під час розроби використовувався шаблон медіатор з відповідною бібліотекою MediatR, яка має досить багато гнучких конструкцій для розробки. Тому був розроблений абстрактний клас, який буде базовим для всіх контролерів, а саме MediatorController. Даний клас містить конструктор, в який ми передаємо екземпляр IMediator та два методи, відповідно до методів самого медіатора Send та Publish. В цілому основною метою даного класу є те щоб ми по замовчуванню передавали HttpContext.RequestAborted в медіатор, оскільки весь код є асинхронним і може бути перерваний, якщо користувач відмінить запит, тому це лише опціональна фіча для економії ресурсів.

Клас MediatorController:

```
using MediatR;
using Microsoft.AspNetCore.Mvc;

namespace DreamNetwork.Presentation.Controllers;

public abstract class MediatorController : ControllerBase
{
        protected IMediator _mediator;

        public MediatorController(IMediator mediator)
```

```
        {
            _mediator = mediator;
        }

        protected async Task<T> SendAsync<T>(IRequest<T> request)
        {
            return await _mediator.Send(request, HttpContext.RequestAborted);
        }

        protected async Task PublishAsync<T>(IRequest<T> request)
        {
            await _mediator.Publish(request, HttpContext.RequestAborted);
        }
}
```

Почнемо розробку з контролера AccountController. В ньому будуть всі необхідні методи для логіну, реєстрації, зміни пароля, отримання токена доступу, тощо. Одразу перейдемо до методу Login. Маємо певні вхідні дані, в даному випадку LoginRequest, який містить UserName та Password користувача. Далі створюємо екземпляр команди типу LoginUserCommand та відправляємо її в медіатор. Медіатор сам розуміє в який саме обробник далі передати команду, відповідно виконує дане завдання та повертає результат про операції логіну. В результаті відправляємо користувачу Jwt токен та сесію або 401 помилку.

Приклад методу Login:

```
[HttpPost("login")]
        [SwaggerOperation(
            Description = "Logins user, returns session and jwtToken.",
            Summary = "Login in system.")]
        [AllowAnonymous]
        [ProducesResponseType(typeof(LoginUserResult), StatusCodes.Status200OK)]
        [ProducesResponseType(StatusCodes.Status401Unauthorized)]
        public async Task<IActionResult> Login([FromBody] LoginRequest request)
        {
            var command = _mapper.Map<LoginUserCommand>(request);

            var result = await SendAsync(command);

            return result.Success ? Ok(result) : Unauthorized();
        }
```

В вище наведеному принципу будуть реалізовані майже всі контролери та їх методи. Спершу йде обробка вхідних даних, створення відповідної команди чи запиту та передача його в медіатор, а по результату виконання той чи іншої команди ми формуємо відповідну відповідь для користувача.

Далі розглянемо що ж є таке насправді MediatR на прикладі команди логіну користувача у систему. Сам MediatR містить в собі інтерфейси IRequest та IRequestHandler.

Код інтерфейсів:

```
public interface IRequest : IRequest<Unit> { }

public interface IRequest<out TResponse> : IBaseRequest { }

public interface IBaseRequest { }

public interface IRequestHandler<in TRequest, TResponse>
   where TRequest : IRequest<TResponse>
         {

   Task<TResponse> Handle(TRequest request, CancellationToken cancellationToken);
         }
```

Тому створюємо необхідні класи LoginUserCommand, LoginUserCommandHandler та LoginUserResult.

Клас LoginUserCommand:

```
public sealed record LoginUserCommand : IRequest<LoginUserResult>
{
   public string? UserName { get; init; }

   public string? Password { get; init; }

   public string? DeviceName { get; init; }

   public ApplicationType ApplicationType { get; init; }

   public string? OsVersion { get; init; }

   public string? AppVersion { get; init; }
         }
```

Далі маємо обробник LoginUserCommandHandler. Він імплементує відповідний інтерфейс IRequestHandler<LoginUserCommand>, який потім використовується медіатором для створення зв’язку між типом команди та типом обробника. Містить всього лише один метод Handle в якому й відбувається вся логіка.

Код LoginUserCommandHandler:

```
public sealed class LoginUserCommandHandler : IRequestHandler<LoginUserCommand, LoginUserResult>
{
   private readonly SignInManager<UserEntity> _signInManager;
   private readonly IJwtCreator _jwtCreator;
   private readonly ISessionsManager _sessionsManager;
   private readonly IHttpContextAccessor _httpContextAccessor;
```

```
    public LoginUserCommandHandler(
        SignInManager<UserEntity> signInManager,
        IJwtCreator jwtCreator,
        ISessionsManager sessionsManager,
        IHttpContextAccessor httpContextAccessor)
    {
        _signInManager = signInManager;
        _jwtCreator = jwtCreator;
        _sessionsManager = sessionsManager;
        _httpContextAccessor = httpContextAccessor;
    }

    public async Task<LoginUserResult> Handle(LoginUserCommand request, CancellationToken
cancellationToken)
    {
        var user = await _signInManager.UserManager.FindByNameAsync(request.UserName!);

        if (user == null)
        {
            return new LoginUserResult { Success = false };
        }

        var signInResult = await _signInManager.CheckPasswordSignInAsync(user, request.Password!,
lockoutOnFailure: false);

        if (!signInResult.Succeeded)
        {
            return new LoginUserResult
            {
                Success = false
            };
        }

        var sessionPasscode = PasscodeGenerator.GeneratePasscode(length: 64);

        var session = new SessionEntity
        {
            DeviceName = request.DeviceName,
            ApplicationType = request.ApplicationType,
            OsVersion = request.OsVersion,
            AppVersion = request.AppVersion,
            IpAddress = _httpContextAccessor.HttpContext?.Connection?.RemoteIpAddress?.ToString() ??
"unknown",
            Location = "Lutsk, Ukraine"
        };

        await _sessionsManager.CreateSessionAsync(user, session, sessionPasscode, cancellationToken);

        return new LoginUserResult
        {
            Success = true,
            SessionPasscode = sessionPasscode,
            AccessToken = _jwtCreator.GenerateJwt(user),
            SessionId = session.Id
         };
    }
          }
```

Сам процес логіну користувача використовує стандартну перевірку по відповідності паролю до того що в нас в базі даних, а вірніше хешу пароля, самий класичний метод входу користувача в систему. У разі успіху ми генеруємо нову сесію, яка складається з паролю сесії та ідентифікатора сесії, а також одразу генеруємо JWT токен та повертаємо все це користувачу.

Сама ідея полягає в тому що ми маємо клас певного команди, відповідний клас обробника, та відповідний тип результату. Передаючи в медітатор певну команду, він в свою чергу викликає певний обробник команди, який повертає певний результат і все це динамічно конфігурується для aspnet DI за допомогою рефлексії. Також команди можна перевикористовувати в різних методах контролеру, або в деяких випадках викликати по декілька команд одразу.

Варто також розглянути конфігурацію сесії для даного aspnet додатку. В даному випадку ми використовуємо дві схеми аутентифікації, перша це JWT, а друга це власні сесії, і на останніх варто зупинитись більш детально.

Спершу було створено наступний клас, який представляє нашу схему аутентифікації:

```
public sealed class SessionAuthenticationSchemeOptions : AuthenticationSchemeOptions
{
        }
```

Далі статичний клас з декількома константами, які нам знадобляться:

```
public static class SessionDefaults
{
   public const string SessionScheme = "DreamNetworkSessionScheme";

   public const string SessionHeaderKey = "DreamNetworkSession";
        }
```

І тепер вже створюємо клас SessionAuthentictionHandler, який і є основним класом обробником для авторизації користувача по нашій схемі аутентифікації. В даному методі ми отримуємо значення відповідного хедера з HTTP запиту, в якому і знаходиться токен сесії, який користувачі передають для аутентифікації по даній схемі. Як раніше було описано, токен складається з ідентифікатора сесії та паролю сесії, тому ми шукаємо сесії в базі даних за допомогою ідентифікатора, а далі звіряємо хеші паролів і у випадку успіху авторизуємо користувача. Приклад коду представлений нижче:

```
public sealed class SessionAuthenticationHandler : AuthenticationHandler<SessionAuthenticationSchemeOptions>
{
    private const string InvalidSessionMessage = "Invalid session";

    private readonly ISessionsManager _sessionsManager;

    public SessionAuthenticationHandler(
        IOptionsMonitor<SessionAuthenticationSchemeOptions> options,
        ILoggerFactory logger,
        UrlEncoder encoder,
        ISystemClock clock,
        ISessionsManager sessionsManager) : base(options, logger, encoder, clock)
    {
        _sessionsManager = sessionsManager;
    }

    protected override async Task<AuthenticateResult> HandleAuthenticateAsync()
    {
        var sessionToken = Context.SessionToken();

        if (sessionToken == null)
        {
            return AuthenticateResult.Fail(InvalidSessionMessage);
        }

        var (sessionId, passcode) = sessionToken.Value;

        var session = await _sessionsManager.FindSessionByIdAsync(sessionId);

        if (session == null || !_sessionsManager.VerifySession(session, passcode))
        {
            return AuthenticateResult.Fail(InvalidSessionMessage);
        }

        var claims = new Claim[]
        {
            new(ClaimTypes.NameIdentifier, session.UserId.ToString()),
            new(ClaimTypes.Name, session.User!.UserName!)
        };

        var identity = new ClaimsIdentity(claims, SessionDefaults.SessionScheme);

        var claimsPrincipal = new ClaimsPrincipal(identity);

        var ticket = new AuthenticationTicket(claimsPrincipal, SessionDefaults.SessionScheme);

        return AuthenticateResult.Success(ticket);
    }
        }
```

Даний механізм дозволяє власноруч додати будь-яку схему аутентифікації користувача, яку ми захочемо, а також в результаті виконання даного коду ми додаємо певні дані про користувача в HttpContext, що може буде корисним для деяких сценаріїв у методах контролерів. Все що необхідно виконати далі так просто це додати дану схему аутентифікації, в даному випадку ми підключаємо

одразу JWT схему та нашу розроблену схему за допомогою нижче наведеного коду:

```
builder.Services
        .AddAuthentication(o =>
        {
            o.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;
            o.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;
        })
        .AddJwtBearer(o =>
        {
            var key = builder.Configuration[$"{jwtOptionsConfigurationName}:{nameof(JwtOptions.Key)}"];

            if (string.IsNullOrEmpty(key))
            {
                throw new ArgumentException($"'{nameof(JwtOptions)}' key is null or empty. Check configuration
to solve this error.");
            }

            var issuer = builder.Configuration[$"{jwtOptionsConfigurationName}:{nameof(JwtOptions.Issuer)}"];

            var audience =
builder.Configuration[$"{jwtOptionsConfigurationName}:{nameof(JwtOptions.Audience)}"];


            o.TokenValidationParameters = new TokenValidationParameters
            {
                ValidateIssuerSigningKey = true,
                ValidateIssuer = true,
                ValidateAudience = true,
                ValidateLifetime = true,
                ValidIssuer = issuer,
                ValidAudience = audience,
                IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(key))
            };

            o.Events = new JwtBearerEvents()
            {
                OnMessageReceived = (context) =>
                {
                    var token = context.Request.Query["access_token"];
                    var path = context.HttpContext.Request.Path;

                    if (path.StartsWithSegments("/notify"))
                    {
                        context.Token = token;
                    }

                    return Task.CompletedTask;
                }
            };
        })
                .AddScheme<SessionAuthenticationSchemeOptions,
SessionAuthenticationHandler>(SessionDefaults.SessionScheme, null);
```

Також вище наведений код для JWT одразу сконфігурований і для веб-сокетів, оскільки вони також використовуються аутентифікацію, але у нашому випадку для по JWT схемі.

Далі ще варто розглянути як підключити SignalR після встановлення необхідного пакету. Для початку потрібно додати необхідні компоненти SignalR в сервіси додатку за допомогою нижче наведеного коду:

```
builder.Services.AddSignalR();
```

Далі треба використати його в нашому конвеєрі за допомогою методу app.MapHub, в якому ми вказуємо необхідний Hub та route. Приклад коду наведений нижче:

```
app.MapHub<ChatHub>("/notify");
```

Далі розглянемо що таке використаний ChatHub. Даний клас наслідується від класу Hub<T>, де T є інтерфейс в якому містяться методи, які в свою чергу є процедурами, які можуть бути викликані сервером на стороні клієнту. А методи самого ChatHub є методами, які може викликати клієнт, в більшості випадків там будуть звичайні підключення клієнтів до необхідних груп, щоб потім звертаючись до цих груп, викликати необхідні процедури на стороні клієнта.

Приклад класу ChatHub:

```
[Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]
public sealed class ChatHub : Hub<IChatHubClient>
{
   public Task SubscribeToGroup(string groupId)
   {
      /*підключення клієнта до оновлень чату*/
   }

   public Task SubscribeToEvents()
   {
      /*пікдлючення клієнт до інших подій*/
   }
}
```

Приклад інтерфейсу IChatHubClient:

```
public interface IChatHubClient
{
   Task MessageSentAsync(SendMessageNotification notification);

   Task ChatCreatedAsync(ChatCreatedNotification notification);

   Task RemovedFromChatAsync(RemovedFromChatNotification notification);
        }
```

Далі варто розглянути клієнтський застосунок. Клієнтський застосунок розробляється за допомогою фреймворку Angular. Для початку варто зазначити що в Angular все базується на компонентах, тому в нас є основний компонент app.component та модуль AppRouting в якому ми вказуємо всі роутинги нашого додатку. Основний компонент використовує <router-outlet/>, в якому в залежності від відповідного route буде рендеритись компонент з нашого модуля AppRouting. Приклад коду представлений нижче.

Код app.component.html:

```
<router-outlet/>
```

Код app-routing.module.ts:

```
const routes: Routes = [
  { path: '', pathMatch: 'full', redirectTo: 'main' },
  { path: 'login', component: LoginComponent },
  { path: 'register', component: RegisterComponent },
  {
    path: 'main',
    component: MainComponent,
    canActivate: [AuthorizedGuard],
    children: [
      { path: '', pathMatch: 'full', redirectTo: 'chats' },
      { path: 'chats', component: ChatsComponent, title: 'Chats', canActivate: [AuthorizedGuard] },
      { path: 'account', component: AccountComponent, title: 'Account', canActivate: [AuthorizedGuard] }
    ]}
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

Відповідно до наведеного коду ми бачимо що в нас є три основних компонента, це login, register та main. Відповідно в залежності від певного роутингу, який динамічно буде змінюватись в нашому url адресі під час використання браузеру, буде змінюватись рендеринг в DOM на відповідний компонент. При цьому самого перезавантаження сторінки як такої не

відбувається, тому що даний тип додатку називається SPA (single page application). MainComponent в свою чергу має також певні розділення на chats та account, які там само будуть рендеритись в місці тегу <router-outlet/>, тільки цього разу цей тег знаходиться в компоненті Main. По такому принципу можна налаштовувати навігацію Angular додатку.

Розглянемо одразу MainComponent. Даний компонент містить Sivenav меню та контейнер в якому відбувається рендеринг компоненту в залежності від нашої навігації по додатку. Код розмітки компоненту наведений нижче:

```
<mat-sidenav-container class="container">
  <mat-sidenav #sidenav class="sidenav" mode="over">
    <mat-nav-list>
      <a (click)="joinToChat(); sidenav.toggle()" mat-list-item>
        <i>Join to chat</i>
      </a>

      <a (click)="createChat(); sidenav.toggle()" mat-list-item>
        <i>Create Chat</i>
      </a>

      <a
        mat-list-item
        routerLink="/main/chats"
        (click)="sidenav.toggle()"
      >
        <strong>Chats</strong>
      </a>
      <a
        mat-list-item
        routerLink="/main/account"
        (click)="sidenav.toggle()"
      >
        <strong>Account</strong>
      </a>
    </mat-nav-list>
  </mat-sidenav>
  <mat-sidenav-content class="sidenav-content">
    <div>
```

```
    <mat-toolbar color="primary">
      <button mat-icon-button (click)="sidenav.open()">
        <mat-icon>menu</mat-icon>
      </button>
    </mat-toolbar>
    <router-outlet></router-outlet>
   </div>
  </mat-sidenav-content>
</mat-sidenav-container>
```

Як бачимо є певні елементи меню, які відповідають за навігацію або за виклик певний діалогових вікон та самий основний тег <router-outlet/>, який відповідає за рендеринг компонентів в залежності від навігації по додатку. Кожен компонент в Angular складається з трьох основних частин, ми поки що розглядали лише розмітку, але крім цього є файл стилів та сам безпосередньо файл typescript, який містить клас компонента. Є також механізм який дозволяє писати розмітку, стилі та typescript код в одному файлі, але це не завжди є хорошою практикою, оскільки розділення розмітки від коду допомагає краще орієнтуватись та розробляти застосунок. Розглянемо TS код даного компонента MainComponent. Основне що варто тут виділити так це метод ngOnInit, який виконується один раз при ініціалізації компоненту. Це один з методів, які прив’язані до циклу життя компоненту в Angular. В даному методі ми підключаємось до веб-сокетів серверу за допомогою SignalR, а також підписуємось на відповідні події, на які застосунок в майбутньому буде реагувати. Приклад коду наведений нижче:

```
@Component({
  selector: 'app-main',
  templateUrl: './main.component.html',
  styleUrls: ['./main.component.css']
})
export class MainComponent implements OnInit, OnDestroy {
  private destroy$ = new Subject<void>();

  constructor(
    private signalRService: SignalRService,
    private dialog: MatDialog) {}
```

```
ngOnInit(): void {
  this.signalRService.$onSignalConnected
    .asObservable()
    .pipe(takeUntil(this.destroy$))
    .subscribe(() => {
      this.signalRService.subscribeToUserEvents();
    });

  this.signalRService.openConnection();
  this.signalRService.turnOnEventListeners();
}

ngOnDestroy(): void {
  this.destroy$.next();
  this.destroy$.complete();
}

createChat(): void {
  this.dialog.open(CreateChatDialogComponent, {
    minWidth: 320,
    maxHeight: 600,
  });
}

joinToChat(): void {
  this.dialog.open(SearchChatsDialog, {
    minWidth: 450,
    minHeight: 600
  });
 }
}
```

Також можемо розглянути розроблений SignalRService. Даний клас є сервісом тому відповідно додається в DI Angular фреймворку. Розглянемо самі основні частини даного сервісу. Спершу було створено декілька змінних типу EventEmitter<T>. Даний об’єкт представляє певний потік подій певно типу, на які ми в майбутньому можемо підписатись. Приклад коду наведений нижче:

```
$onMessageReceived = new EventEmitter<SendMessageNotification>();
 $onChatCreated = new EventEmitter<ChatCreatedNotification>();
```

```
$onRemovedFromChat = new EventEmitter<RemovedFromChatNotification>();
```

Далі використовуємо з’єднання SignalR в методі turnOnEventListeners щоб приймати повідомлення про певні події такі як нове повідомлення чи створення чату. В результаті ми ставимо обробники на ці повідомлення, які в свою чергу передають ці події у відповідний EventEmitter, що дозволяє в майбутньому досить зручно підписуватись на ці події з інших компонентів. Нижче наведений приклад коду методу turnOnEventListeners:

```
turnOnEventListeners(): void {
  this.connection.on(MESSAGE_SENT, (notification: SendMessageNotification) => {
    this.$onMessageReceived.emit(notification);
  });

  this.connection.on(CHAT_CREATED, (notification: ChatCreatedNotification) => {
    this.$onChatCreated.emit(notification);
  });

  this.connection.on(REMOVED_FROM_CHAT, (notification: RemovedFromChatNotification) => {
    this.$onRemovedFromChat.emit(notification);
  });
 }
```

Використання EventEmitter ми розглянемо в наступному компоненті, а саме ChatsComponent. Розмітка даного компоненту досить проста. Вона має список чатів, які ми поміщаємо в Sidenav меню, та контейнер чату який ми переглядаємо. Також присутнє використання директиви *ngFor для рендерингу списку чатів. Дана директива прив’язується до моделі даних, в даному випадку списку чатів та реагує на його зміни, динамічно додаючи новий компонент чату в DOM чи видаляючи його, в залежності від того додали нас в певний чат чи видалив з чату. Розмітка компоненту наведена нижче:

```
<mat-sidenav-container class="container">
  <mat-sidenav class="sidenav" mode="side" opened>
    <mat-nav-list>
      <div *ngFor="let chat of chatsStorageService.allChats;">
        <app-chat-item
          [isActive]="chat.id === chatsStorageService.currentChat?.id"
```

```
        title="{{chat.title}}"
        lastMessageText="{{chat.lastMessageText}}"
        lastMessageAuthor="{{chat.lastMessageAuthor}}"
        chatLogoUrl="{{chat.chatType === 0
          ? this.resourcesService.createPathToGroupAvatar(chat.imageFileName)
          : this.resourcesService.createPathToUserAvatar(chat.imageFileName)}}"
        (click)="openChat(chat.id)"/>
      </div>
    </mat-nav-list>
  </mat-sidenav>
  <mat-sidenav-content>
    <app-chat-container
      [chat]="chatsStorageService.currentChat"
      [messages]="chatsStorageService.currentChatMessages"/>
  </mat-sidenav-content>
</mat-sidenav-container>
```

Далі в даному компоненті в методі ngOnInit ми використовуємо раніше створені EventEmitter. Ми відкриваємо з'єднання з сервером через SignalR. Далі підписуємо на кожен EventEmitter, тому коли ми будемо отримувати нове повідомлення чи будемо додані в певний чат ми одразу ж будемо реагувати на ці події та змінювати модель даних, яка знаходиться у нас в додатку, а саме список чатів, список повідомлень відкритого чату та інше. В результаті маніпуляцій з моделлю даних в нас динамічно буде мінятись рендеринг компонентів та з'являтись нові чати у списку.

```
ngOnInit(): void {
  this.signalRService.openConnection();

  this.chatsService.getChats().subscribe(res => {
    this.chatsStorageService.allChats = res;
  });

  this.signalRService.$onChatCreated
    .asObservable()
    .pipe(takeUntil(this.destroy$))
    .subscribe(notification => {
      this.chatsService.getMyChatById(notification.chatId).subscribe(chat => {
        this.chatsStorageService.allChats.push(chat);
```

```
      });
    });

    this.signalRService.$onRemovedFromChat
      .asObservable()
      .pipe(takeUntil(this.destroy$))
      .subscribe(notification => {
        this.chatsStorageService.allChats = this.chatsStorageService.allChats.filter(x => x.id != notification.chatId);
        if (this.chatsStorageService.currentChat?.id == notification.chatId) {
          this.chatsStorageService.currentChat = undefined;
        }
      });

    this.signalRService.$onMessageReceived
      .asObservable()
      .pipe(takeUntil(this.destroy$))
      .subscribe(notification => {
        const chatIndex = this.chatsStorageService.allChats.findIndex(x => x.id == notification.chatId)
        this.chatsStorageService.allChats[chatIndex].lastMessageText = notification.textContent;
        this.chatsStorageService.allChats[chatIndex].lastMessageAuthor = notification.authorName;
        this.chatsStorageService.allChats[chatIndex].lastMessageAuthorId = notification.authorId;
        this.chatsStorageService.allChats[chatIndex].lastMessageDate = notification.createdAt;
      })
  }
```

Розглянемо те як відбувається відправлення нового повідомлення в чат. Це відбувається в компоненті ChatContainer після того як ми написати повідомлення у відповідному текстовому полі та натиснули відповідну кнопку за допомогою методу sendNewMessage. Даний метод використовує MessagesService для відправлення повідомлення. Приклад коду наведений нижче:

```
  sendNewMessage(messageText: string): void {
    const sendMessageRequest = {
      chatId: this.chat?.id,
      textContent: messageText
    } as SendMessageRequest;
```

```
    this.messagesService.sendMessage(sendMessageRequest).subscribe();
  }
```

Сам сервіс MessagesService містить всього два методи, перший для отримання списку повідомлень, другий для відправлення нового повідомлення в чат. Код сервісу наведений нижче:

```
@Injectable({
  providedIn: 'root'
})
export class MessagesService {
  private static readonly messagesRoute = `${environment.baseApiUrl}/api/messages`;
  constructor(private http: HttpClient) { }

  getMessages(chatId: string): Observable<Message[]> {
    return this.http.get<Message[]>(
      `${MessagesService.messagesRoute}/${chatId}`,
      { context: new HttpContext().set(IS_JWT_BEARER_SCHEME_ENABLED, true) }
    );
  }

  sendMessage(request: SendMessageRequest): Observable<SendMessageResult> {
    const formData = new FormData();

    formData.set('chatId', request.chatId);

    if (request.textContent) {
      formData.set('textContent', request.textContent);
    }

    if (request.inReplyToMessageId) {
      formData.set('inReplyToMessageId', request.inReplyToMessageId);
    }

    if (request.attachments) {
      for (let file in request.attachments) {
        formData.append('attachments', file);
      }
    }
```

```
    return this.http.post<SendMessageResult>(
      MessagesService.messagesRoute,
      formData,
      { context: new HttpContext().set(IS_JWT_BEARER_SCHEME_ENABLED, true) }
    );
  }
}
```

Відповідно до вище наведеного сервісу, розглянемо метод sendMessage. Даний метод приймає SendMessageRequest та формує екземпляр FormData, який представляє HttpContent типу multipart/form-data. Даний тип контенту був обраний у зв'язку з тим, що якщо доведеться відправляти медіадані, звичайний json не зможе цього якісно зробити. Більшість інших методів сервісів, які використовують HTTP протокол спілкуються за допомогою JSON формату даних. Повернемось до методу, ми формуємо передаємо контент в http за допомогою методу post, передаємо відповідний шлях до нашого API та передаємо в контекст інформацію про те що схемою аутентифікації для даного методу є JWT.

Як бачимо Angular дозволяє робити досить багато речей досить зручно, навіть той самий HttpContext. Даний контекст ми використовуємо в HttpInterceptor, в даному випадку JwtInterceptor. Даний клас виконує попередню обробку HTTP запиту до того як він буде відправлений до серверу. В методі intercept ми розглядаємо значення контексту і якщо він містить інформацію про те що ми використовуємо JWT схему аутентифікації то ми підставляємо відповідний токен в хедер HTTP запиту. Приклад коду наведений нижче:

```
@Injectable()
export class JwtInterceptor implements HttpInterceptor {

  constructor(private tokensService: TokensService) {}

  intercept(request: HttpRequest<unknown>, next: HttpHandler): Observable<HttpEvent<unknown>> {
    if (request.context.get(IS_JWT_BEARER_SCHEME_ENABLED)) {
      const accessToken = this.tokensService.getAccessToken() ?? "";
      const handledRequest = request.clone({
        headers: setJwtBearerHeader(request.headers, accessToken)
```

```
        });

        return next.handle(handledRequest);
      }

      return next.handle(request);
    }
  }
```

Отже після створення серверної частини та клієнтського застосунку можемо перейти до наступного етапу, а саме керівництво користувача.

## 3.3 Керівництво користувача.

Відповідно до розробленої системи необхідно певне керівництво користувача, за допомогою якого користувач зможе зрозуміти, як користуватись системою та який функціонал вона надає.

Спершу користувачу потрібен доступ до мережі Інтернет, а далі відповідно веб-браузер. Далі потрібно увійти у свій обліковий запис, або створити його. Вигляд сторінки авторизації та реєстрації зображені відповідно на рисунках Б.1 та Б.2. Далі відповідно користувачу буде зображено список доступних чатів, а по натисканню на чат з списку, відкриються його повідомлення, приклад зображений на рисунку Б.3. Навігація по застосунку відбувається за допомогою меню зображеного на рисунку Б.4. Таким способом є можливість перейти до сторінки керуванням обліковим записом, яка зображена на рисунку Б.5. Для створення нового чату, так само необхідно використовувати меню, але в даному випадку натиснувши на кнопку створення нового чату, користувач не перейде на нову сторінку, а відкриється діалог, який дозволяє

створити чат, після введення необхідних даних в форму. Приклад діалогу зображено на рисунку Б.6.

Деякі дії в чаті, такі як, видалення чату чи запрошення до нього нового учасника відбувається за допомогою контекстного меню. В цілому інтерфейс користувача достатньо інтуїтивно зрозумілий.

## 3.4 Тестування програмного забезпечення.

Тестування розробленого програмного забезпечення досить важлива частина, так як при передачі продукту замовнику ми маємо бути впевнені що все працює так як повинно, оскільки це досить сильно впливає на подальше бажання до співпраці з нами.

Тестування є багатьох видів, від звичайного мануального до розробки автоматизованих тестів, як модульних так і інших типів. Також є досить поширеним використання технології розробки TDD (Test-Driven Development), під час якого спочатку розробники пишуть тести, а потім код для цих самих тестів, тоді розроблювана система є одразу протестованою.

AspNetCore та Angular мають необхідні компоненти для тестування, а саме модульні тести. У нашому випадку тестування є досить накладним процесом, тому ми будемо проводити лише мануальне тестування клієнтського застосунку.

## 3.5 Висновки

Отже результатом виконаної роботи є розроблена система, а саме серверна частина та клієнтський веб-застосунок, який пройшов мануальне тестування усіх компонентів.

Під час роботи було розроблено серверну частину та клієнтський веб-застосунок. Також було створено керівництво користувача, яке допоможе розібратись з тим як працює система та як її експлуатувати.

Результати тестування в цілому успішне, всі необхідні компоненти працюють належним чином, повідомлення та чати створюються миттєво, а також миттєво з’являються у додатку клієнта, що забезпечує нас миттєвим спілкування у live режимі.

# ВИСНОВКИ

При виконанні поставленого завдання кваліфікаційної роботи було проведено детальне дослідження предметної, яке дало зрозуміти всі елементи майбутньої розробки. Далі був проведений аналіз наявного програмного забезпечення, в результаті чого, було виявлено переваги та недоліки наявних систем. Наступник кроком стало виявлення потреби до програмного продукту та створення технічного завдання для розроблюваного застосунку.

Наступним, досить важливим етапом, було проектування програмного забезпечення. Було проаналізовано деякі види архітектур. Результатом став вибір необхідних архітектур, в даному випадку клієнт-серверна архітектура, шаблон MVVM, багаторівнева архітектура та MVC архітектура. Маючи обрану архітектури та шаблони відбулося детальне виконання декомпозиції розроблюваної системи, проєктування бази даних та інтерфейсу користувача. На основі цих даних, було обрано засоби розробки системи, а саме AspNetCore для серверної частини та Angular для клієнтського додатку.

Визначивши вимоги було проведено детальне проєктування модулів, яке описувало всі основні складові програмної системи, а також взаємодію між модулями та взаємодія з інтерфейсом користувача.

Також була створена інструкція користувача, яка допоможе зрозуміти те як користуватись програмним продуктом та на що він здатен.

Фінальним етапом стало тестування програмного продукту та вибір методів тестування.

Отже, програмний продукт був успішно розроблений та в цілому задовольняє поставлені вимоги.

# ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ


1. Уніфікована мова програмування UML. Портал знань : веб-сайт. – URL: http://www.znannya.org/?view=uml (дата звернення: 26.02.2023).

2. Методичні вказівки щодо його виконання для студентів спеціальності 121 «Інженерія програмного забезпечення». URL: https://msn.khmnu.edu.ua/pluginfile.php/626097/mod_resource/content/1/Методичка%20по%20ДП.pdf (дата звернення: 13.02.2023).

3. Simon K. Digital 2022: Global overview report. URL: https://datareportal.com/reports/digital-2022-global-overview-report (дата звернення: 16.02.2023).

4. Top Websites Ranking – Most Visited Websites In The World. URL: https://igroup.com.ua/seo-articles/sotsialna-merezha/ (дата звернення: 21.02.2023).

5. Найважливіші архітектурні шаблони, які необхідно знати. URL: https://hi-news.pp.ua/kompyuteri/print:page,1,11675-arhtektura-programnogo-zabezpechennya-vidi-opis-rozrobka.html (дата звернення: 20.03.2023).

6. Архітектура програмного забезпечення: види, опис, розробка URL: https://devzone.org.ua/post/nayvazhlivishi-arkhitekturni-shabloni-yaki-neobkhidno-znati (дата звернення: 20.03.2023).

7. Клієнт–серверна архітектура. URL: https://training.qatestlab.com/blog/technical-articles/client-server-architecture/ (дата звернення: 23.03.2023).

8. Клієнт-серверна архітектура та ролі сервера. URL: https://medium.com/@IvanZmerzlyi/клієнт-серверна-архітектура-та-ролі-серверів-9893d8048229 (дата звернення: 23.03.2023).

9. Nystrom R. Game Programming Patterns. Apress, 2011. 300 с.

10. Розділяй та володарюй: що таке патерни MVC і MVP, та як їх використовувати. URL: https://highload.today/uk/blogs/shho-take-mvc-ta-mvp-patterni/ (дата звернення: 25.03.2023).

11. Robert C. Martin. Clean Architecture: A Craftsman's Guide to Software Structure and Design / Robert C. Martin – Publisher(s): Pearson, 2017. – 432 с.

12. Overview of ASP.NET Core SignalR. Microsoft ASP.NET Core Documentation: веб-сайт. URL: https://learn.microsoft.com/en-us/aspnet/core/signalr/introduction?view=aspnetcore-7.0 (дата звернення: 18.04.2023).

13. Unity Asset Store. Unity Asset Store: веб-сайт. URL: https://assetstore.unity.com. (дата звернення: 18.04.2023).

14. Основні принципи створення інтерфейсу. URL: http://um.co.ua/8/8-10/8-109690.html (дата звернення: 05.04.2023).

15. Visual Studio. Visual Studio: веб-сайт. URL: https://visualstudio.microsoft.com. (дата звернення: 16.02.2023).

16. Design Basics. Figma: веб-сайт. URL: https://www.figma.com/resource-library/design-basics. (дата звернення: 06.03.2023).

17. Троелсен Ендрю. Мова програмування C# та платформа .Net і .Net Core, 8-е видання.: Пер. з англ / Ендрю Троелсен., 2018 – 1328 с.

18. Огляд видів тестування. QATestlab: веб-сайт. URL: https://training.qatestlab.com/blog/technical-articles/review-the-types-of-testing/. (дата звернення: 02.05.2023).

19. Програма Model-View-Controller MVC – що це. URL: https://hi-news.pp.ua/kompyuteri/14628-programa-model-view-controller-mvc-scho-ce-osoblivost-opis.html (дата звернення: 25.03.2023).

20. Леонов О. Тестування ПЗ (види тестування) / О. Леонов [Електронний ресурс] – Режим доступу: https://drukarnia.com.ua/articles/testuvannya-pz-vidi-testuvannya-JInS1#heading-3-4659 (дата звернення – 12.02.2023). – Назва з екрану.

21. Entity Framework Core. URL: https://www.entityframeworktutorial.net/efcore/entity-framework-core.aspx (дата звернення: 18.04.2023).

22. Overview of ASP.NET Core MVC. URL: https://learn.microsoft.com/uk-ua/aspnet/core/mvc/overview?view=aspnetcore-7.0 (дата звернення: 17.04.2023).

23. Buttfield-Addison P., Manning J., Nugent T. Unity Game Development Cookbook: Essentials for Every Game. O'Reilly Media, 2019. 406 с.

24. ASP.NET Core Documentation. URL: https://learn.microsoft.com/en-us/aspnet/core/?view=aspnetcore-7.0 (дата звернення: 20.04.2023).

25. Introduction to ASP.NET Core MVC. URL: https://www.yogihosting.com/aspnet-core-introduction/ (дата звернення: 21.04.2023).

26. Introduction to Unity UI. Kodeco: веб-сайт. URL: https://www.kodeco.com/34347684-introduction-to-unity-ui-part-1. (дата звернення: 20.04.2023).

27. Murugan M. Custom User Management in ASP.NET Core MVC with Identity. URL: https://codewithmukesh.com/blog/user-management-in-aspnet-core-mvc/ (дата звернення: 21.04.2023).

28. Use AJAX to Deliver Dynamic Updates. URL: https://learn.microsoft.com/en-us/aspnet/mvc/overview/older-versions-1/nerddinner/use-ajax-to-deliver-dynamic-updates (дата звернення: 25.04.2023).

29. Animator. Unity Documentation: веб-сайт. URL: https://docs.unity3d.com/ScriptReference/Animator.html. (дата звернення: 24.04.2023).

30. Collider’s and Triggers in Unity. Chris Hilton personal blog: веб-сайт. URL: https://christopherhilton88.medium.com/colliders-and-triggers-in-unity-understanding-the-basics-7192714f3440. (дата звернення: 26.04.2023).

31. What is Unreal Engine. BairesDevBlog: веб-сайт. URL: https://www.bairesdev.com/blog/what-is-unreal-engine/. (дата звернення: 12.04.2023).

32. New Input System. Сайт Unity Documentation: веб-сайт. URL: https://docs.unity3d.com/Packages/com.unity.inputsystem@1.6/manual/index.html. (дата звернення: 14.04.2023).

33. What is an ASP.NET Core? Сайт Microsoft .NET: веб-сайт. URL: https://dotnet.microsoft.com/en-us/learn/aspnet/what-is-aspnet-core. (дата звернення: 14.04.2023).

34. ECS For Unity. Unity: веб-сайт. URL: https://unity.com/ecs. (дата звернення: 14.04.2023).

35. Unity With MVC. Toptal: веб-сайт. URL: https://www.toptal.com/unity-unity3d/unity-with-mvc-how-to-level-up-your-game-development. (дата звернення: 15.04.2023).

36. MonoBehaviour. Unity Documentation: веб-сайт. URL: https://docs.unity3d.com/ScriptReference/MonoBehaviour.html. (дата звернення: 14.04.2023).

37. Events&Delegates in Unity. gamedevbegginer: веб-сайт. URL: https://docs.unity3d.com/Manual/EventFunctions.html. (дата звернення: 14.04.2023).

38. Scolastici C. Unity 2D Game Development Cookbook. Packt Publishing - ebooks Account, 2015. 256 с.

39. Event functions. Unity Documentation: веб-сайт. URL: https://docs.unity3d.com/Manual/EventFunctions.html. (дата звернення: 11.04.2023).

40. Офіційна документація Microsoft. .NET documentation: веб-сайт. URL: https://docs.microsoft.com/en-us/dotnet. (дата звернення: 20.03.2023).

# ДОДАТОК А

(обов’язковий)

# ПРОГРАМНИЙ КОД ОСНОВНИХ МОДУЛІВ

## А.1 Програмний код класу Program.cs.

```
using DreamNetwork.BusinessLogic.Extensions;
using DreamNetwork.BusinessLogic.Hubs;
using DreamNetwork.Data;
using DreamNetwork.Identity.Extensions.DepencyInjection;
using DreamNetwork.Presentation.Constants;
using DreamNetwork.Presentation.Extensions.DepencyInjection;
using Microsoft.EntityFrameworkCore;
using System.Reflection;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddControllers();

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddDreamNetworkSwagger();

builder.Services.AddDbContext<DreamNetworkContext>(options =>
{
   options.UseSqlServer(EnvironmentConstants.SqlConnectionString());
});

builder.AddDreamNetworkIdentity();

builder.Services.AddDreamNetworkBusinessLogic();

builder.Services.AddSignalR();

builder.Services.AddCors(o =>
{
   o.AddPolicy("CorsDefaultPolicy", p =>
   {
      p.WithOrigins("http://localhost:4200", "https://localhost:7083");
      p.AllowAnyHeader();
      p.AllowAnyMethod();
      p.AllowCredentials();
   });
});

builder.Services.AddAutoMapper(Assembly.GetExecutingAssembly());

var app = builder.Build();

app.UseSwagger();

app.UseSwaggerUI();

app.UseHttpsRedirection();

app.UseWebSockets();
```

```
app.UseCors("CorsDefaultPolicy");

app.UseAuthentication();

app.UseAuthorization();

app.MapControllers();

app.MapHub<ChatHub>("/notify");

app.Run();
```

## А.2 Програмний код базового класу контролерів MediatorController для AspNetCore застосунку

```
using MediatR;
using Microsoft.AspNetCore.Mvc;

namespace DreamNetwork.Presentation.Controllers;

public abstract class MediatorController : ControllerBase
{
	protected IMediator _mediator;

	public MediatorController(IMediator mediator)
	{
		_mediator = mediator;
	}

	protected async Task<T> SendAsync<T>(IRequest<T> request)
	{
		return await _mediator.Send(request, HttpContext.RequestAborted);
	}

	protected async Task PublishAsync<T>(IRequest<T> request)
	{
		await _mediator.Publish(request, HttpContext.RequestAborted);
	}
}
```

## А.3 Програмний код контролеру ChatsController

```
using DreamNetwork.BusinessLogic.Commands;
using DreamNetwork.BusinessLogic.Commands.Chats;
using DreamNetwork.BusinessLogic.Models;
using DreamNetwork.BusinessLogic.Queries.Chats;
using DreamNetwork.Presentation.Extensions;
using DreamNetwork.Presentation.Requests.Chats;
using MediatR;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Swashbuckle.AspNetCore.Annotations;

namespace DreamNetwork.Presentation.Controllers;

[Route("api/chats")]
[ApiController]
[Authorize]
public class ChatsController : MediatorController
{
   public ChatsController(IMediator mediator) : base(mediator)
   {
   }
```

```
[HttpGet]
[SwaggerOperation(
    Description = "Get all current user's chats.",
    Summary = "Get all user's chats.")]
[ProducesResponseType(typeof(Chat[]), StatusCodes.Status200OK)]
public async Task<IActionResult> GetMyChats()
{
    var query = new GetUserChatsQuery
    {
        UserId = HttpContext.CurrentUserId().GetValueOrDefault()
    };

    var result = await SendAsync(query);

    return Ok(result.Chats);
}

[HttpGet("{chatId:guid}")]
[SwaggerOperation(
    Description = "Returns found user chat by chatId.",
    Summary = "Get my chat by id.")]
[ProducesResponseType(typeof(Chat), StatusCodes.Status200OK)]
[ProducesResponseType(typeof(Chat), StatusCodes.Status404NotFound)]
public async Task<IActionResult> GetMyChatById([FromRoute] Guid chatId)
{
    var query = new GetChatQuery(
        chatId,
        HttpContext.CurrentUserId().GetValueOrDefault());

    var result = await SendAsync(query);

    if (!result.Success)
    {
        return NotFound(result);
    }

    return Ok(result.Chat);
}

[HttpGet("search")]
[SwaggerOperation(
    Description = "Searches public chats (with type 'Public Chat') " +
    "by title (using 'contains' operation). " +
    "Returns only chats that have 'IsInviteOnly=false'.",
    Summary = "Search chats by title.")]
[ProducesResponseType(typeof(Chat[]), StatusCodes.Status200OK)]
public async Task<IActionResult> SearchPublicChats([FromQuery] string title)
{
    var query = new SearchPublicChatsQuery(title);

    var result = await SendAsync(query);

    return Ok(result.Chats);
}

[HttpPost("direct-chat/{partnerId:guid}")]
[SwaggerOperation(
    Description = "Creates new chat of type 'Direct Chat' with specified user.",
    Summary = "Create chat with specified user.")]
[ProducesResponseType(typeof(CreateChatResult), StatusCodes.Status200OK)]
[ProducesResponseType(typeof(CreateChatResult), StatusCodes.Status409Conflict)]
public async Task<IActionResult> CreateDirectChat([FromRoute] Guid partnerId)
{
    var command = new CreateDirectChatCommand
    {
        UserId = HttpContext.CurrentUserId().GetValueOrDefault(),
        PartnerId = partnerId
    };
```

```
    var result = await SendAsync(command);

    if (!result.Success)
    {
        return Conflict(result);
    }

    return Ok(result);
}

[HttpPost("public-chat")]
[SwaggerOperation(
    Description = "Creates new chat of type 'Public Chat'.",
    Summary = "Create public chat.")]
[ProducesResponseType(typeof(CreateChatResult), StatusCodes.Status200OK)]
[ProducesResponseType(typeof(CreateChatResult), StatusCodes.Status409Conflict)]
public async Task<IActionResult> CreatePublicChat([FromBody] CreatePublicChatRequest request)
{
    var command = new CreatePublicChatCommand(
        HttpContext.CurrentUserId().GetValueOrDefault(),
        request.Title,
        request.Description,
        request.IsInviteOnly);

    var result = await SendAsync(command);

    if (!result.Success)
    {
        return Conflict(result);
    }

    return Ok(result);
}

[HttpPost("join/{chatId:guid}")]
[SwaggerOperation(
    Description = "",
    Summary = "")]
[ProducesResponseType(StatusCodes.Status204NoContent)]
[ProducesResponseType(typeof(Result), StatusCodes.Status409Conflict)]
public async Task<IActionResult> JoinChat([FromRoute] Guid chatId)
{
    var command = new JoinToChatCommand(HttpContext.CurrentUserId().GetValueOrDefault(), chatId);

    var result = await SendAsync(command);

    if (!result.Success)
    {
        return Conflict(result);
    }

    return NoContent();
}

[HttpPost("invite")]
[SwaggerOperation(
    Description = "",
    Summary = "")]
[ProducesResponseType(StatusCodes.Status204NoContent)]
[ProducesResponseType(typeof(Result), StatusCodes.Status409Conflict)]
public async Task<IActionResult> AddUserToChat([FromBody] InviteUserToChat request)
{
    var command = new AddUserToChatCommand(
        request.ChatId,
        HttpContext.CurrentUserId().GetValueOrDefault(),
        request.UserId);
```

```
    var result = await SendAsync(command);

    if (!result.Success)
    {
        return Conflict(result);
    }

    return NoContent();
}

[HttpPost("leave/{chatId:guid}")]
[ProducesResponseType(StatusCodes.Status204NoContent)]
[ProducesResponseType(typeof(Result), StatusCodes.Status409Conflict)]
public async Task<IActionResult> LeaveFromChat([FromRoute] Guid chatId)
{
    var command = new LeaveChatCommand(
        chatId,
        HttpContext.CurrentUserId().GetValueOrDefault());

    var result = await SendAsync(command);

    if (!result.Success)
    {
        return Conflict(result);
    }

    return NoContent();
}

[HttpDelete("{chatId:guid}")]
[ProducesResponseType(StatusCodes.Status204NoContent)]
[ProducesResponseType(typeof(Result), StatusCodes.Status409Conflict)]
public async Task<IActionResult> DeleteChat([FromRoute] Guid chatId)
{
    var command = new DeleteChatCommand(
        chatId,
        HttpContext.CurrentUserId().GetValueOrDefault());

    var result = await SendAsync(command);

    if (!result.Success)
    {
        return Conflict(result);
    }

    return NoContent();
}

[HttpPost("{chatId:guid}/chat-picture")]
[Consumes("multipart/form-data")]
[SwaggerOperation(
    Description = "",
    Summary = "")]
[ProducesResponseType(StatusCodes.Status204NoContent)]
[ProducesResponseType(typeof(Result), StatusCodes.Status409Conflict)]
public async Task<IActionResult> UpdateProfilePicture(IFormFile newPicture, [FromRoute] Guid chatId)
{
    var command = new UpdateChatPictureCommand(
        chatId,
        HttpContext.CurrentUserId().GetValueOrDefault(),
        newPicture);

    var result = await SendAsync(command);

    if (!result.Success)
    {
        return Conflict(result);
    }
```

```
            return NoContent();
        }

        [HttpDelete("{chatId:guid}/chat-picture")]
        [SwaggerOperation(
            Description = "",
            Summary = "")]
        [ProducesResponseType(StatusCodes.Status204NoContent)]
        [ProducesResponseType(typeof(Result), StatusCodes.Status409Conflict)]
        public async Task<IActionResult> DeleteProfilePicture([FromRoute] Guid chatId)
        {
            var command = new DeleteChatPictureCommand(
                chatId,
                HttpContext.CurrentUserId().GetValueOrDefault());

            var result = await SendAsync(command);

            if (!result.Success)
            {
                return Conflict(result);
            }

            return NoContent();
        }
    }
```

## А.4 Програмний код контролеру AccountController

```
using AutoMapper;
using DreamNetwork.BusinessLogic.Commands.Account;
using DreamNetwork.BusinessLogic.Queries.Account;
using DreamNetwork.Identity.Authentication.Session;
using DreamNetwork.Identity.Extensions.Session;
using DreamNetwork.Presentation.Extensions;
using DreamNetwork.Presentation.Requests.Account;
using DreamNetwork.Presentation.Responses.Account;
using MediatR;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using Swashbuckle.AspNetCore.Annotations;

namespace DreamNetwork.Presentation.Controllers;

[Route("api/account")]
[ApiController]
public sealed class AccountController : MediatorController
{
        private readonly IMapper _mapper;

        public AccountController(IMediator mediator, IMapper mapper) : base(mediator)
        {
                _mapper = mapper;
        }

        [HttpPost("login")]
        [SwaggerOperation(
                Description = "Logins user, returns session and jwtToken.",
                Summary = "Login in system.")]
        [AllowAnonymous]
        [ProducesResponseType(typeof(LoginUserResult), StatusCodes.Status200OK)]
        [ProducesResponseType(StatusCodes.Status401Unauthorized)]
        public async Task<IActionResult> Login([FromBody] LoginRequest request)
        {
                var command = _mapper.Map<LoginUserCommand>(request);
```

```
        var result = await SendAsync(command);

        return result.Success ? Ok(result) : Unauthorized();
}

[HttpPost("create")]
[SwaggerOperation(
        Description = "Creates new account.",
        Summary = "Register new account.")]
[AllowAnonymous]
[ProducesResponseType(StatusCodes.Status204NoContent)]
[ProducesResponseType(typeof(IdentityError[]), StatusCodes.Status400BadRequest)]
public async Task<IActionResult> Create([FromBody] RegisterRequest request)
{
        var command = _mapper.Map<CreateUserCommand>(request);

        var result = await SendAsync(command);

        return result.Success ? NoContent() : BadRequest(result.Errors);
}

[HttpPost("change-password")]
[SwaggerOperation(
        Description = $"Changes your password. {SessionDefaults.SessionScheme} scheme.",
        Summary = "Change your password.")]
[Authorize(AuthenticationSchemes = SessionDefaults.SessionScheme)]
[ProducesResponseType(StatusCodes.Status204NoContent)]
[ProducesResponseType(typeof(IdentityError[]), StatusCodes.Status400BadRequest)]
public async Task<IActionResult> ChangePassword([FromBody] ChangePasswordRequest request)
{
        var command = _mapper.Map<ChangePasswordCommand>(request) with
        {
                UserId = HttpContext.CurrentUserId().GetValueOrDefault()
        };

        var result = await SendAsync(command);

        return result.Success ? NoContent() : BadRequest(result.Errors);
}

[HttpGet("generate-access-token")]
[SwaggerOperation(
        Description = $"Returns new access (jwt) token. {SessionDefaults.SessionScheme} scheme.",
        Summary = "Get access token.")]
[Authorize(AuthenticationSchemes = SessionDefaults.SessionScheme)]
[ProducesResponseType(typeof(GenerateAccessTokenResult), StatusCodes.Status200OK)]
[ProducesResponseType(StatusCodes.Status400BadRequest)]
public async Task<IActionResult> GenerateAccessToken()
{
        var (sessionId, passcode) = HttpContext.SessionToken().GetValueOrDefault();

        var command = new GenerateAccessTokenCommand
        {
                SessionId = sessionId,
                SessionPasscode = passcode
        };

        var result = await SendAsync(command);

        if (result.Success)
        {
                return Ok(result);
        }

        return BadRequest();
}

[HttpGet("sessions")]
```

```
        [SwaggerOperation(
                Description = $"Returns all sessions for user. {SessionDefaults.SessionScheme} scheme.",
                Summary = "Get info about all sessions.")]
        [Authorize(AuthenticationSchemes = SessionDefaults.SessionScheme)]
        [ProducesResponseType(typeof(Session[]), StatusCodes.Status200OK)]
        public async Task<IActionResult> GetSessions()
        {
                var query = new GetSessionsQuery
                {
                        UserId = HttpContext.CurrentUserId().GetValueOrDefault()
                };

                var result = await SendAsync(query);

                var sessions = result.Sessions.Select(_mapper.Map<Session>);

                return Ok(sessions);
        }

        [HttpDelete("sessions")]
        [SwaggerOperation(
                Description = $"Deletes specified session. {SessionDefaults.SessionScheme} scheme.",
                Summary = "Sign out specified device.")]
        [Authorize(AuthenticationSchemes = SessionDefaults.SessionScheme)]
        [ProducesResponseType(StatusCodes.Status204NoContent)]
        public async Task<IActionResult> DeleteSession([FromBody] LogoutRequest request)
        {
                var command = _mapper.Map<LogoutCommand>(request);

                var result = await SendAsync(command);

                return result.Success ? NoContent() : BadRequest();
        }

        [HttpDelete("sessions/sign-out")]
        [SwaggerOperation(
                Description = $"Deletes all sessions. {SessionDefaults.SessionScheme} scheme.",
                Summary = "Sign out from all devices.")]
        [Authorize(AuthenticationSchemes = SessionDefaults.SessionScheme)]
        [ProducesResponseType(StatusCodes.Status204NoContent)]
        public async Task<IActionResult> DeleteAllSessions()
        {
                var command = new LogoutAllDevicesCommand
                {
                        UserId = HttpContext.CurrentUserId().GetValueOrDefault()
                };

                var result = await SendAsync(command);

                return result.Success ? NoContent() : BadRequest();
        }
}
```

## А.5 Програмний код контролеру MessagesController

```
using DreamNetwork.BusinessLogic.Commands.Messages;
using DreamNetwork.BusinessLogic.Models;
using DreamNetwork.BusinessLogic.Queries.Messages;
using DreamNetwork.Presentation.Extensions;
using DreamNetwork.Presentation.Requests.Messages;
using MediatR;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Swashbuckle.AspNetCore.Annotations;

namespace DreamNetwork.Presentation.Controllers;

[Route("api/messages")]
[ApiController]
```

```
[Authorize]
public sealed class MessagesController : MediatorController
{
    public MessagesController(IMediator mediator) : base(mediator)
    {
    }

    [HttpGet("{chatId:guid}")]
    [SwaggerOperation(
        Description = "",
        Summary = "")]
    [ProducesResponseType(typeof(Message[]), StatusCodes.Status200OK)]
    [ProducesResponseType(StatusCodes.Status404NotFound)]
    public async Task<IActionResult> GetMessages([FromRoute] Guid chatId)
    {
        var query = new GetChatMessagesQuery(HttpContext.CurrentUserId().GetValueOrDefault(), chatId);

        var result = await SendAsync(query);

        if (!result.Success)
        {
            return NotFound();
        }

        return Ok(result.Messages);
    }

    [HttpPost]
    [SwaggerOperation(
        Description = "",
        Summary = "")]
    [ProducesResponseType(typeof(SendMessageResult), StatusCodes.Status200OK)]
    [ProducesResponseType(typeof(SendMessageResult), StatusCodes.Status409Conflict)]
    public async Task<IActionResult> SendMessage([FromForm] SendMessageRequest request)
    {
        var command = new SendMessageCommand(
            HttpContext.CurrentUserId().GetValueOrDefault(),
            request.ChatId,
            request.TextContent!,
            request.InReplyToMessageId,
            request.Attachments);

        var result = await SendAsync(command);

        if (!result.Success)
        {
            return Conflict(result);
        }

        return Ok(result);
    }
}
```

## А.6 Програмний код контролеру ProfileController

```
using DreamNetwork.BusinessLogic.Commands.Profile;
using DreamNetwork.Presentation.Extensions;
using MediatR;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Swashbuckle.AspNetCore.Annotations;

namespace DreamNetwork.Presentation.Controllers;

[Route("api/profile")]
[ApiController]
[Authorize]
```

```
public sealed class ProfileController : MediatorController
{
    public ProfileController(IMediator mediator) : base(mediator)
    {
    }

    [HttpPost("avatar")]
        [Consumes("multipart/form-data")]
        [SwaggerOperation(
            Description = "Changes your profile picture.",
            Summary = "Change profile picture.")]
        [ProducesResponseType(StatusCodes.Status204NoContent)]
        [ProducesResponseType(StatusCodes.Status409Conflict)]
        public async Task<IActionResult> UploadProfilePicture(IFormFile newPicture)
        {
            var command = new UploadProfilePictureCommand(
                HttpContext.CurrentUserId().GetValueOrDefault(),
                newPicture);

            var result = await SendAsync(command);

            if (!result.Success)
            {
                return BadRequest();
            }

            return NoContent();
        }

        [HttpDelete("avatar")]
        [SwaggerOperation(
            Description = "Deletes your profile picture.",
            Summary = "Delete profile picture.")]
        [ProducesResponseType(StatusCodes.Status204NoContent)]
        public async Task<IActionResult> DeleteProfilePicture()
        {
            var command = new DeleteProfilePictureCommand(HttpContext.CurrentUserId().GetValueOrDefault());

            var result = await SendAsync(command);

            if (!result.Success)
            {
                return BadRequest();
            }

            return NoContent();
        }
}
```

## А.7 Програмний код контролеру UsersController

```
using DreamNetwork.BusinessLogic.Models;
using DreamNetwork.BusinessLogic.Queries.Users;
using DreamNetwork.Data.Entities;
using DreamNetwork.Presentation.Extensions;
using DreamNetwork.Presentation.Responses.Users;
using MediatR;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using Swashbuckle.AspNetCore.Annotations;

namespace DreamNetwork.Presentation.Controllers;

[Route("api/users")]
[ApiController]
public class UsersController : MediatorController
```

```
{
    private readonly AspNetUserManager<UserEntity> _userManager;

    public UsersController(IMediator mediator, AspNetUserManager<UserEntity> userManager) : base(mediator)
    {
        _userManager = userManager;
    }

    [HttpGet("me")]
    [SwaggerOperation(
        Description = "Returns your personal detailed profile information.",
        Summary = "Get personal profile information.")]
    [Authorize]
    [ProducesResponseType(typeof(PersonalProfile), StatusCodes.Status200OK)]
    [ProducesResponseType(StatusCodes.Status409Conflict)]
    public async Task<IActionResult> MyProfile()
    {
        var currentUserId = HttpContext.CurrentUserId().GetValueOrDefault().ToString();

        var currentUser = await _userManager.FindByIdAsync(currentUserId);

        if (currentUser == null)
        {
            return Conflict();
        }

        return Ok(new PersonalProfile
        {
            Id = currentUser.Id,
            UserName = currentUser.UserName,
            Email = currentUser.Email,
            Description = currentUser.Description,
            DisplayName = currentUser.DisplayName,
            ProfilePictureUrl = currentUser.ProfilePictureUrl
        });
    }

    [HttpGet("{userId:guid}")]
    [SwaggerOperation(
        Description = "Returns user public profile information.",
        Summary = "Get public user profile by UserId.")]
    [Authorize]
    [ProducesResponseType(typeof(UserProfile), StatusCodes.Status200OK)]
    [ProducesResponseType(StatusCodes.Status404NotFound)]
    public async Task<IActionResult> GetUser([FromRoute] Guid userId)
    {
        var user = await _userManager.FindByIdAsync(userId.ToString());

        if (user == null)
        {
            return NotFound();
        }

        return Ok(new UserProfile
        {
            Id = user.Id,
            Description = user.Description,
            DisplayName = user.DisplayName,
            ProfilePictureUrl = user.ProfilePictureUrl,
        });
    }

    [HttpGet("search")]
    [ProducesResponseType(typeof(User[]), StatusCodes.Status200OK)]
    public async Task<IActionResult> Search([FromQuery] string userName)
    {
        var query = new SearchUsersQuery(userName);
```

```
        var result = await SendAsync(query);

        return Ok(result.Users);
    }

    [HttpGet("check-username")]
        [SwaggerOperation(
            Description = "Returns 'true' if username not taken by any user.",
            Summary = "Check if username is free to use.")]
        [AllowAnonymous]
        [ProducesResponseType(typeof(bool), StatusCodes.Status200OK)]
    public async Task<IActionResult> CheckUsername([FromQuery] string value)
    {
        var userNameIsFree = (await _userManager.FindByNameAsync(value)) == null;

        return Ok(userNameIsFree);
    }

    [HttpGet("check-email")]
        [SwaggerOperation(
            Description = "Returns 'true' if email not taken by any user.",
            Summary = "Check if email is free to use.")]
        [AllowAnonymous]
        [ProducesResponseType(typeof(bool), StatusCodes.Status200OK)]
    public async Task<IActionResult> CheckEmail([FromQuery] string value)
    {
        var emailIsFree = (await _userManager.FindByEmailAsync(value)) == null;

        return Ok(emailIsFree);
    }
}
```

## А.8 Програмний код контролеру ResourcesController

```
using DreamNetwork.BusinessLogic.Queries.Resources;
using MediatR;
using Microsoft.AspNetCore.Mvc;

namespace DreamNetwork.Presentation.Controllers;

[Route("api/resources")]
[ApiController]
public sealed class ResourcesController : MediatorController
{
    public ResourcesController(IMediator mediator) : base(mediator)
    {
    }

    [HttpGet("user-avatar/{fileName}")]
    public async Task<IActionResult> GetUserAvatar([FromRoute] string fileName)
    {
        var query = new GetUserAvatarQuery(fileName);

        var result = await SendAsync(query);

        return File(result.File, result.ContentType);
    }

    [HttpGet("chat-avatar/{fileName}")]
    public async Task<IActionResult> GetChatAvatar([FromRoute] string fileName)
    {
        var query = new GetChatAvatarQuery(fileName);

        var result = await SendAsync(query);

        return File(result.File, result.ContentType);
    }
```

```
    [HttpGet("attachment/{fileName}")]
    public async Task<IActionResult> GetAttachment([FromRoute] string fileName)
    {
        return Ok();
    }
}
```

## А.9 Програмний код контексту бази даних DreamNetworkContext

```
using DreamNetwork.Data.Entities;
using Microsoft.EntityFrameworkCore;

namespace DreamNetwork.Data;

public sealed class DreamNetworkContext : DbContext
{
    public DbSet<UserEntity> Users { get; set; }

    public DbSet<SessionEntity> Sessions { get; set; }

    public DbSet<ChatEntity> Chats { get; set; }

    public DbSet<MessageEntity> Messages { get; set; }

    public DbSet<UserChatEntity> UsersChats { get; set; }

    public DreamNetworkContext(DbContextOptions options) : base(options)
    {
    }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        base.OnModelCreating(modelBuilder);
        modelBuilder.ApplyConfigurationsFromAssembly(typeof(DreamNetworkContext).Assembly);
    }
}
```

## А.10 Програмний код app.module.ts основного модулю Angular

```
import {NgModule} from '@angular/core';
import {BrowserModule} from '@angular/platform-browser';
import {AppComponent} from './app.component';
import {LoginComponent} from './components/pages/login/login.component';
import {RegisterComponent} from './components/pages/register/register.component';
import {ChatsComponent} from './components/pages/chats/chats.component';
import {RouterOutlet} from "@angular/router";
import {AppRoutingModule} from "./app-routing/app-routing.module";
import {FormsModule, ReactiveFormsModule} from "@angular/forms";
import {HTTP_INTERCEPTORS, HttpClientModule} from "@angular/common/http";
import {JwtInterceptor} from "./interceptors/jwt.interceptor";
import {SessionInterceptor} from "./interceptors/session.interceptor";
import {AuthInterceptor} from "./interceptors/auth.interceptor";
import {BrowserAnimationsModule} from '@angular/platform-browser/animations';
import {materialModules} from "./modules/material-modules";
import {PlatformModule} from "@angular/cdk/platform";
import {AccountComponent} from './components/pages/account/account.component';
```

```
import {MainComponent} from './components/pages/main/main.component';
import {ChangePasswordFormComponent} from './components/forms/change-password-form/change-password-form.component';
import {ChangePasswordDialogComponent} from './components/dialogs/change-password-dialog/change-password-dialog.component';
import {SessionListDialogComponent} from './components/dialogs/session-list-dialog/session-list-dialog.component';
import {SessionItemComponent} from './components/session-item/session-item.component';
import {SessionCardComponent} from './components/session-card/session-card.component';
import {SessionCardDialogComponent} from './components/dialogs/session-card-dialog/session-card-dialog.component';
import {ChatItemComponent} from './components/chat-item/chat-item.component';
import {ChatContainerComponent} from './components/chat-container/chat-container.component';
import {ChatHeaderComponent} from './components/chat-container/chat-header/chat-header.component';
import {ChatInputComponent} from './components/chat-container/chat-input/chat-input.component';
import {ChatMessageComponent} from './components/chat-container/chat-message/chat-message.component';
import {CdkFixedSizeVirtualScroll, CdkVirtualForOf, CdkVirtualScrollViewport} from "@angular/cdk/scrolling";
import {CreateChatDialogComponent} from './components/dialogs/create-chat-dialog/create-chat-dialog.component';
import {ChatCardComponent} from './components/cards/chat-card/chat-card.component';
import {SearchChatsDialog} from "./components/dialogs/search-chats-dialog/search-chats-dialog.component";
import {InviteUserToChatDialog} from './components/dialogs/invite-user-to-chat-dialog/invite-user-to-chat-dialog';

@NgModule({
  declarations: [
    AppComponent,
    LoginComponent,
    RegisterComponent,
    ChatsComponent,
    AccountComponent,
    MainComponent,
    ChangePasswordFormComponent,
    ChangePasswordDialogComponent,
    SessionListDialogComponent,
    SessionItemComponent,
    SessionCardComponent,
    SessionCardDialogComponent,
    ChatItemComponent,
    ChatContainerComponent,
    ChatHeaderComponent,
    ChatInputComponent,
    ChatMessageComponent,
    CreateChatDialogComponent,
    ChatCardComponent,
    SearchChatsDialog,
    InviteUserToChatDialog
  ],
    imports: [
```

```
      BrowserModule,
      RouterOutlet,
      AppRoutingModule,
      ReactiveFormsModule,
      HttpClientModule,
      BrowserAnimationsModule,
      PlatformModule,
      materialModules,
      FormsModule,
      CdkVirtualScrollViewport,
      CdkVirtualForOf,
      CdkFixedSizeVirtualScroll
    ],
  providers: [
    {
      provide: HTTP_INTERCEPTORS,
      useClass: JwtInterceptor,
      multi: true
    },
    {
      provide: HTTP_INTERCEPTORS,
      useClass: SessionInterceptor,
      multi: true
    },
    {
      provide: HTTP_INTERCEPTORS,
      useClass: AuthInterceptor,
      multi: true
    }
  ],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

## A.11 Програмний код SessionInterceptor

```
import { Injectable } from '@angular/core';
import {
  HttpRequest,
  HttpHandler,
  HttpEvent,
  HttpInterceptor
} from '@angular/common/http';
import { Observable } from 'rxjs';
import {TokensService} from "../services/tokens.service";
import {IS_SESSION_SCHEME_ENABLED} from "../../api/authentication/session-defaults";
```

```
@Injectable()
export class SessionInterceptor implements HttpInterceptor {
  private static readonly SessionHeaderName = "DreamNetworkSession";

  constructor(private tokensService: TokensService) {}

  intercept(request: HttpRequest<unknown>, next: HttpHandler): Observable<HttpEvent<unknown>> {
    if (request.context.get(IS_SESSION_SCHEME_ENABLED)) {
      const sessionToken = this.tokensService.getSessionToken() ?? '';
      const handledRequest = request.clone({
        headers: request.headers.set(SessionInterceptor.SessionHeaderName, sessionToken)
      });

      return next.handle(handledRequest);
    }

    return next.handle(request);
  }
}
```

## A.12 Програмний код JwtInterceptor

```
import {Injectable} from '@angular/core';
import {
  HttpRequest,
  HttpHandler,
  HttpEvent,
  HttpInterceptor
} from '@angular/common/http';
import {Observable} from 'rxjs';
import {TokensService} from "../services/tokens.service";
import {IS_JWT_BEARER_SCHEME_ENABLED, setJwtBearerHeader} from "../../api/authentication/jwt-defaults";

@Injectable()
export class JwtInterceptor implements HttpInterceptor {

  constructor(private tokensService: TokensService) {}

  intercept(request: HttpRequest<unknown>, next: HttpHandler): Observable<HttpEvent<unknown>> {
    if (request.context.get(IS_JWT_BEARER_SCHEME_ENABLED)) {
      const accessToken = this.tokensService.getAccessToken() ?? "";
      const handledRequest = request.clone({
        headers: setJwtBearerHeader(request.headers, accessToken)
      });
```

```
    return next.handle(handledRequest);
   }

   return next.handle(request);
  }
}
```

## A.13 Програмний код AuthInterceptor

```
import {Injectable} from '@angular/core';
import {
  HttpErrorResponse,
  HttpEvent,
  HttpHandler,
  HttpInterceptor,
  HttpRequest, HttpStatusCode
} from '@angular/common/http';
import {catchError, Observable, of, switchMap, throwError} from 'rxjs';
import {AccountService} from "../../api/services/account.service";
import {Router} from "@angular/router";
import {TokensService} from "../services/tokens.service";
import {IS_JWT_BEARER_SCHEME_ENABLED, setJwtBearerHeader} from "../../api/authentication/jwt-defaults";
import {IS_SESSION_SCHEME_ENABLED} from "../../api/authentication/session-defaults";

@Injectable()
export class AuthInterceptor implements HttpInterceptor {

  constructor(
    private accountService: AccountService,
    private tokensService: TokensService,
    private router: Router) {}

  intercept(request: HttpRequest<unknown>, next: HttpHandler): Observable<HttpEvent<unknown>> {
    return next.handle(request).pipe(
      catchError(err => {
        if (err.status === HttpStatusCode.Unauthorized) {
          return this.handleUnauthorizedError(err, request, next);
        }

        return throwError(err);
      })
    );
  }

  private handleUnauthorizedError(
    httpError: HttpErrorResponse,
```

```
    request: HttpRequest<unknown>,
    next: HttpHandler): Observable<HttpEvent<unknown>> | Observable<any> {
    if (request.context.get(IS_SESSION_SCHEME_ENABLED)) {
      this.navigateToLogin();
      return of(undefined);
    }

    if (request.context.get(IS_JWT_BEARER_SCHEME_ENABLED)) {
      const generateAccessTokenRequest = this.accountService.generateAccessToken();
      return generateAccessTokenRequest
        .pipe(
          switchMap(response => {
            this.tokensService.setAccessToken(response.accessToken);
            const handledRequest = request.clone({
              headers: setJwtBearerHeader(request.headers, response.accessToken)
            });
            return next.handle(handledRequest);
          }),
          catchError((err: HttpErrorResponse) => {
            if (err.status == HttpStatusCode.Unauthorized) {
              this.navigateToLogin();
              return of(undefined);
            }
            throw err;
          })
        );
    }

    return throwError(() => httpError);
  }

  private navigateToLogin() {
    this.router.navigateByUrl('login').then();
  }
}
```

## A.14 Програмний код AuthirizedGuard

```
import {Injectable} from '@angular/core';
import {ActivatedRouteSnapshot, CanActivate, RouterStateSnapshot, UrlTree} from '@angular/router';
import {catchError, map, Observable, of} from 'rxjs';
import {TokensService} from "../services/tokens.service";
import {AccountService} from "../../api/services/account.service";
import {HttpErrorResponse, HttpStatusCode} from "@angular/common/http";
import {isTokenInvalidOrExpired} from "../../api/authentication/jwt-defaults";
```

```
@Injectable({
  providedIn: 'root'
})
export class AuthorizedGuard implements CanActivate {
  constructor(
    private tokensService: TokensService,
    private accountService: AccountService) {}

  canActivate(
    route: ActivatedRouteSnapshot,
    state: RouterStateSnapshot): Observable<boolean | UrlTree> | Promise<boolean | UrlTree> | boolean | UrlTree {
    const accessToken = this.tokensService.getAccessToken() ?? '';

    if (isTokenInvalidOrExpired(accessToken)) {
      this.accountService.generateAccessToken().pipe(
        map(res => {
          this.tokensService.setAccessToken(res.accessToken);
          return res.success;
        }),
        catchError((err: HttpErrorResponse) => {
          if (err.status == HttpStatusCode.Unauthorized) {
            return of(false);
          }
          throw err;
        })
      )
    }

    return true;
  }
}
```

## A.15 Програмний код компоненту сторінки LoginComponent

Розмітка:

```
<form class="login-form" [formGroup]="loginForm">
  <p>
    <mat-form-field appearance="outline">
      <mat-label>Username</mat-label>
      <input
        matInput
        id="username"
        type="text"
        formControlName="username"
        placeholder="Joseph"/>
```

```
      <mat-hint matError *ngIf="username?.touched || username?.dirty">
        <span *ngIf="username?.errors?.['required']">
          Username is <strong>required</strong>.
        </span>
      </mat-hint>
    </mat-form-field>
  </p>
  <p>
    <mat-form-field appearance="outline">
      <mat-label>Password</mat-label>
      <input
        matInput
        id="password"
        type="password"
        formControlName="password"
      />
      <mat-hint matError *ngIf="password?.touched || password?.dirty">
        <span *ngIf="password?.errors?.['required']">
          Password is <strong>required</strong>.
        </span>
      </mat-hint>
    </mat-form-field>
  </p>
  <p>
    <mat-checkbox formControlName="rememberMe" value="false">
      Remember Me
    </mat-checkbox>
  </p>
  <p>
    <button
      mat-raised-button
      color="primary"
      (click)="submit()">
      Submit
    </button>
  </p>
  <p class="error" *ngIf="isUnauthorized">
    <span>Invalid Username or Password.</span>
  </p>
  <p>
    <a routerLink="/register">Create account</a>
  </p>
</form>
```

Стилі:

```
.login-form {
  margin: 0;
  position: absolute;
  top: 50%;
  left: 50%;
  -ms-transform: translate(-50%, -50%);
  transform: translate(-50%, -50%);
  text-align: center;
}

.error {
  color: red;
}
```

## Код:

```
import {Component} from '@angular/core';
import {FormControl, FormGroup, Validators} from "@angular/forms";
import {AccountService} from "../../../../api/services/account.service";
import {LoginRequest} from "../../../../api/types/account/login-request";
import {StoreMode, TokensService} from "../../../services/tokens.service";
import {Router} from "@angular/router";
import {catchError} from "rxjs";
import {HttpErrorResponse} from "@angular/common/http";
import {Platform} from "@angular/cdk/platform";
import {ApplicationType} from "../../../../api/types/account/application-type";
import {environment} from "../../../../environments/environment";

@Component({
  selector: 'app-login',
  templateUrl: './login.component.html',
  styleUrls: ['./login.component.css']
})
export class LoginComponent {
  loginForm = new FormGroup({
    username: new FormControl('', [Validators.required]),
    password: new FormControl('', [Validators.required]),
    deviceName: new FormControl(this.getCurrentPlatform()),
    osVersion: new FormControl(this.getCurrentPlatform()),
    appVersion: new FormControl(environment.appVersion),
    applicationType: new FormControl(ApplicationType.WebApp),
    rememberMe: new FormControl()
  });

  isUnauthorized = false;
```

```
constructor(
  private accountService: AccountService,
  private tokensService: TokensService,
  private router: Router,
  private platform: Platform) {}

submit(): void {
  this.loginForm.markAllAsTouched();

  if (this.loginForm.invalid) {
    return;
  }

  const storeMode = this.loginForm.value.rememberMe === true
    ? StoreMode.LocalStorage
    : StoreMode.AppMemory;

  this.tokensService.switchStoreMode(storeMode);

  const loginRequest = {
    userName: this.username?.value,
    password: this.password?.value,
    deviceName: this.loginForm.get('deviceName')?.value,
    applicationType: this.loginForm.get('applicationType')?.value,
    osVersion: this.loginForm.get('osVersion')?.value,
    appVersion: this.loginForm.get('appVersion')?.value
  } as LoginRequest;

  this.accountService
    .login(loginRequest)
    .pipe(catchError((err: HttpErrorResponse) => {
      this.isUnauthorized = true;
      throw err;
    }))
    .subscribe(res => {
      this.tokensService.setAccessToken(res.accessToken);
      this.tokensService.setSessionToken(`${res.sessionId}:${res.sessionPasscode}`)
      this.router.navigateByUrl('/main/chats').then();
    });
}

get username() {
  return this.loginForm.get('username');
}

get password() {
```

```
    return this.loginForm.get('password');
  }

  private getCurrentPlatform(): string {
    let platforms: string[] = [];

    if (this.platform.ANDROID) { // Android
      platforms.push('Android');
    }

    if (this.platform.IOS) { // Ios
      platforms.push('IOS');
    }

    if (this.platform.FIREFOX) { // Firefox
      platforms.push('Firefox');
    }

    if (this.platform.BLINK) { // Chrome
      platforms.push('Chrome');
    }

    if (this.platform.WEBKIT) { // Webkit or Opera
      platforms.push('Webkit');
    }

    if (this.platform.TRIDENT) { // IE
      platforms.push('IE');
    }

    if (this.platform.EDGE) { // Microsoft EDGE
      platforms.push('Microsoft EDGE');
    }

    if (this.platform.SAFARI) { // Safari
      platforms.push('Safari');
    }

    return platforms.join(',');
  }
}
```

## А.16 Програмний код компоненту RegisterComponent

Розмітка:

```
<div>
  <div class="register-form">
    <ng-container *ngIf="!registrySucceed; else accountCreated">
      <form [formGroup]="registerFormGroup">
        <div class="form-field">
          <mat-form-field class="form-input" appearance="outline">
            <mat-label>Username</mat-label>
            <input
              matInput
              id="username"
              type="text"
              formControlName="username"
              placeholder="Joseph"/>
            <mat-hint matError *ngIf="username?.errors?.['required']">
              Username is <strong>required</strong>.
            </mat-hint>
            <mat-hint *ngIf="username?.pending">
              pending...
            </mat-hint>
            <mat-hint matError *ngIf="username?.errors?.['duplicateUsername']">
              Username is <strong>duplicate</strong>.
            </mat-hint>
            <mat-hint matError *ngIf="username?.errors?.['pattern']">
              Contains <strong>not allowed</strong> characters.
            </mat-hint>
            <mat-hint matError *ngIf="username?.errors?.['maxLength']">
              Max length is <strong>{{username?.errors?.['maxLength']?.['requiredLength']}}</strong>.
            </mat-hint>
          </mat-form-field>
        </div>
        <div class="form-field">
          <mat-form-field class="form-input" appearance="outline">
            <mat-label>Display name</mat-label>
            <input
              matInput
              id="displayName"
              type="text"
              formControlName="displayName"
              placeholder="Joseph"/>
            <mat-hint matError *ngIf="displayName?.errors?.['required']">
              Display name is <strong>required</strong>.
            </mat-hint>
            <mat-hint matError *ngIf="displayName?.errors?.['maxLength']">
              Max length is <strong>{{displayName?.errors?.['maxLength']?.['requiredLength']}}</strong>.
            </mat-hint>
          </mat-form-field>
```

```
</div>
<div class="form-field">
  <mat-form-field class="form-input" appearance="outline">
    <mat-label>Email</mat-label>
    <input
      matInput
      id="email"
      type="email"
      formControlName="email"
      placeholder="example@mail.com"
    />
    <mat-hint matError>
  <span *ngIf="email?.errors?.['required']">
    Email is <strong>required</strong>.
  </span>
      <span *ngIf="email?.errors?.['email']">
    Email is <strong>invalid</strong>.
  </span>
      <span *ngIf="email?.errors?.['duplicateEmail']">
    Email is <strong>duplicate</strong>.
  </span>
    </mat-hint>
    <mat-hint *ngIf="email?.pending">
      pending...
    </mat-hint>
  </mat-form-field>
</div>
<div class="form-field">
  <mat-form-field class="form-input" appearance="outline">
    <mat-label>Password</mat-label>
    <input
      matInput
      id="password"
      type="password"
      formControlName="password"
    />
    <mat-hint matError *ngIf="registerFormGroup.touched">
      <span *ngIf="password?.errors?.['required']">
        Password is <strong>required</strong>.
      </span>
      <span *ngIf="password?.errors?.['pattern']">
        <strong>Required</strong> digit, spec-char, upper-case, length 8+.
      </span>
    </mat-hint>
  </mat-form-field>
</div>
```

```
      <div class="form-field">
        <button
          mat-raised-button
          [disabled]="registerFormGroup.invalid || registerFormGroup.pending || registryPending"
          color="primary"
          (click)="createUser()">
          Create Account
        </button>
      </div>
      <div class="form-field">
        <a routerLink="/login">Go to login</a>
      </div>
    </form>
  </ng-container>
  <ng-template #accountCreated>
    <div>
      Your account was successful created.
    </div>
    <div class="form-field">
      <a routerLink="/login">Go to login</a>
    </div>
  </ng-template>
 </div>
</div>
```

## Стилі:

```
.register-form {
 margin: 0;
 position: absolute;
 top: 50%;
 left: 50%;
 -ms-transform: translate(-50%, -50%);
 transform: translate(-50%, -50%);
 width: auto;
 height: auto;
}

.form-input {
 min-width: 325px;
}

.form-field {
 padding: 2px;
}
```

Код:

```
import {Component, OnInit} from '@angular/core';
import {FormControl, FormGroup, Validators} from "@angular/forms";
import {AccountService} from "../../../../api/services/account.service";
import {RegisterRequest} from "../../../../api/types/account/register-request";
import {IdentityError} from "../../../../api/types/account/identity-error";
import {IdentityErrorCodes} from "../../../../api/types/account/identity-error-codes";
import {catchError, of, switchMap} from "rxjs";
import {HttpErrorResponse} from "@angular/common/http";
import {UsersApiService} from "../../../../api/services/users-api.service";
import {STRONG_PASSWORD_PATTERN, USERNAME_PATTERN} from "../../../../api/security/user-requirements";
import {FormValidators} from "../../../validators/form-validators";

@Component({
  selector: 'app-register',
  templateUrl: './register.component.html',
  styleUrls: ['./register.component.css']
})
export class RegisterComponent implements OnInit {
  registerFormGroup = new FormGroup({
    username: new FormControl(
      '',
      [Validators.required, Validators.pattern(USERNAME_PATTERN), Validators.maxLength(256)],
      [FormValidators.duplicateUsername(this.usersService)]),
    displayName: new FormControl(
      '',
      [Validators.required, Validators.maxLength(256)]
    ),
    email: new FormControl(
      '',
      [Validators.required, Validators.email],
      [FormValidators.duplicateEmail(this.usersService)]),
    password: new FormControl(
      '',
      [Validators.required, Validators.pattern(STRONG_PASSWORD_PATTERN)])
  });

  registryPending = false;
  registrySucceed = false;

  constructor(
    private accountService: AccountService,
    private usersService: UsersApiService) {}
```

```
ngOnInit(): void {}

createUser(): void {
  this.registerFormGroup.markAllAsTouched();

  if (this.registerFormGroup.invalid) {
    return;
  }

  const request = this.registerFormGroup.value as RegisterRequest;

  of(request)
    .pipe(
      switchMap(value => {
        this.registryPending = true;
        return this.accountService.register(value);
      }),
      catchError((err: HttpErrorResponse) => {
        this.handleErrors(err.error as IdentityError[]);
        this.registryPending = false;
        throw err;
      })
    ).subscribe(() => {
      this.registryPending = false;
      this.registrySucceed = true;
    });
}

private handleErrors(errors: IdentityError[]): void {
  if (errors.find(x => x.code == IdentityErrorCodes.DuplicateEmail)) {
    this.email?.setErrors({ duplicateEmail: true });
  }

  if (errors.find(x => x.code == IdentityErrorCodes.DuplicateUserName)) {
    this.username?.setErrors({ duplicateUsername: true });
  }
}

get username() {
  return this.registerFormGroup.get('username');
}

get displayName() {
  return this.registerFormGroup.get('displayName');
}
```

```
  get email() {
    return this.registerFormGroup.get('email');
  }

  get password() {
    return this.registerFormGroup.get('password');
  }
}
```

## A.17 Програмний код MainComponent

### Розмітка:

```
<mat-sidenav-container class="container">
  <mat-sidenav #sidenav class="sidenav" mode="over">
    <mat-nav-list>
      <a (click)="joinToChat(); sidenav.toggle()" mat-list-item>
        <i>Join to chat</i>
      </a>

      <a (click)="createChat(); sidenav.toggle()" mat-list-item>
        <i>Create Chat</i>
      </a>

      <a
        mat-list-item
        routerLink="/main/chats"
        (click)="sidenav.toggle()"
      >
        <strong>Chats</strong>
      </a>
      <a
        mat-list-item
        routerLink="/main/account"
        (click)="sidenav.toggle()"
      >
        <strong>Account</strong>
      </a>
    </mat-nav-list>
  </mat-sidenav>
  <mat-sidenav-content class="sidenav-content">
    <div>
      <mat-toolbar color="primary">
        <button mat-icon-button (click)="sidenav.open()">
          <mat-icon>menu</mat-icon>
```

```
      </button>
    </mat-toolbar>
    <router-outlet></router-outlet>
   </div>
 </mat-sidenav-content>
</mat-sidenav-container>
```

## Стилі:

```
.container {
  position: absolute;
  top: 0;
  bottom: 0;
  left: 0;
  right: 0;
  background: #eee;
}

.sidenav {
  min-width: 250px;
}
```

## Код:

```
import {Component, OnDestroy, OnInit} from '@angular/core';
import {SignalRService} from "../../../../api/services/signal-r.service";
import {Subject, takeUntil} from "rxjs";
import {MatDialog} from "@angular/material/dialog";
import {CreateChatDialogComponent} from "../../dialogs/create-chat-dialog/create-chat-dialog.component";
import {SearchChatsDialog} from "../../dialogs/search-chats-dialog/search-chats-dialog.component";

@Component({
  selector: 'app-main',
  templateUrl: './main.component.html',
  styleUrls: ['./main.component.css']
})
export class MainComponent implements OnInit, OnDestroy {
  private destroy$ = new Subject<void>();

  constructor(
    private signalRService: SignalRService,
    private dialog: MatDialog) {}

  ngOnInit(): void {
    this.signalRService.$onSignalConnected
      .asObservable()
```

```
    .pipe(takeUntil(this.destroy$))
    .subscribe(() => {
      this.signalRService.subscribeToUserEvents();
    });

  this.signalRService.openConnection();
  this.signalRService.turnOnEventListeners();
 }

 ngOnDestroy(): void {
  this.destroy$.next();
  this.destroy$.complete();
 }

 createChat(): void {
  this.dialog.open(CreateChatDialogComponent, {
    minWidth: 320,
    maxHeight: 600,
  });
 }

 joinToChat(): void {
  this.dialog.open(SearchChatsDialog, {
    minWidth: 450,
    minHeight: 600
  });
 }
}
```

## A.18 Програмний код компоненту AccountComponent

Розмітка:

```
<mat-accordion class="example-headers-align" multi>
 <mat-expansion-panel>
  <mat-expansion-panel-header>
   <mat-panel-title>
    Profile
   </mat-panel-title>
   <mat-panel-description>
    <span></span>
    <mat-icon>account_circle</mat-icon>
   </mat-panel-description>
  </mat-expansion-panel-header>

  <div>
```

```
          <img                                    width="44"                                    height="44"
[src]="this.resourcesService.createPathToUserAvatar(this.profile?.profilePictureUrl)" alt=""/>
        </div>

        <div>
          <input
            id="newPicture"
            type="file"
            (change)="avatarFileChanged($event)"
          />
          <button (click)="uploadNewAvatar()" mat-button>Upload new picture</button>
          <button (click)="deleteCurrentAvatar()" mat-button color="warn">Delete current picture</button>
        </div>

        <div>
          <mat-form-field appearance="fill">
            <mat-label>Display Name</mat-label>
            <input
              readonly
              matInput
              id="displayName"
              type="text"
              [value]="profile?.displayName"
            />
          </mat-form-field>
        </div>

      </mat-expansion-panel>
      <mat-expansion-panel>
        <mat-expansion-panel-header>
          <mat-panel-title>
            Personal data
          </mat-panel-title>
          <mat-panel-description>
            <span></span>
            <mat-icon>settings</mat-icon>
          </mat-panel-description>
        </mat-expansion-panel-header>

        <div>
          <mat-form-field appearance="fill">
            <mat-label>Username</mat-label>
            <input
              readonly
              matInput
              id="username"
```

```
        type="text"
        [value]="profile?.userName"
      />
    </mat-form-field>
  </div>

  <div>
    <mat-form-field appearance="fill">
      <mat-label>Email</mat-label>
      <input
        readonly
        matInput
        id="email"
        type="text"
        [value]="profile?.email"
      />
    </mat-form-field>
  </div>
</mat-expansion-panel>

<mat-expansion-panel>
  <mat-expansion-panel-header>
    <mat-panel-title>
      Security
    </mat-panel-title>
    <mat-panel-description>
      <span></span>
      <mat-icon>security</mat-icon>
    </mat-panel-description>
  </mat-expansion-panel-header>
  <div>
    <button
      mat-button
      class="list-action"
      color="warn"
      (click)="openChangePassword()"
    >
      Change password
      <mat-icon>key</mat-icon>
    </button>
    <mat-divider/>
    <button
      mat-button
      class="list-action"
      (click)="openSessions()"
    >
```

```
      Show all sessions
      <mat-icon>computer</mat-icon>
    </button>
    <mat-divider/>
    <button class="list-action" mat-button>
      Another action
      <mat-icon>question_mark</mat-icon>
    </button>
    <mat-divider/>
    <button class="list-action" color="warn" mat-button>
      Logout
      <mat-icon>logout</mat-icon>
    </button>
  </div>
 </mat-expansion-panel>
</mat-accordion>
<div *ngIf="isPending">
 <mat-progress-bar mode="query"></mat-progress-bar>
</div>
```

## Стилі:

```
.example-headers-align .mat-expansion-panel-header-description {
 justify-content: space-between;
 align-items: center;
}

.example-headers-align .mat-mdc-form-field + .mat-mdc-form-field {
 margin-left: 8px;
}

.list-action {
 width: 100%;
 height: 56px;
 justify-content: start;
}
```

## Код:

```
import {Component, OnDestroy, OnInit, ViewChild} from '@angular/core';
import {UsersApiService} from "../../../../api/services/users-api.service";
import {Subject, takeUntil} from "rxjs";
import {PersonalProfile} from "../../../../api/types/users/personal-profile";
import {MatAccordion} from '@angular/material/expansion';
import {MatDialog} from '@angular/material/dialog';
```

```
import {ChangePasswordDialogComponent} from "../../dialogs/change-password-dialog/change-password-dialog.component";
import {SessionListDialogComponent} from "../../dialogs/session-list-dialog/session-list-dialog.component";
import {AccountService} from "../../../../api/services/account.service";
import {ResourcesService} from "../../../../api/services/resources.service";
import {ProfileService} from "../../../../api/services/profile.service";

@Component({
  selector: 'app-account',
  templateUrl: './account.component.html',
  styleUrls: ['./account.component.css']
})
export class AccountComponent implements OnInit, OnDestroy {
  private destroy$ = new Subject<void>();

  public profile?: PersonalProfile;

  @ViewChild(MatAccordion) accordion?: MatAccordion;

  private newAvatarFile?: File;

  constructor(
    private dialog: MatDialog,
    private usersService: UsersApiService,
    private accountService: AccountService,
    private profileService: ProfileService,
    public resourcesService: ResourcesService) {}

  ngOnInit(): void {
    this.getMe();
  }

  getMe(): void {
    this.usersService.me()
      .pipe(takeUntil(this.destroy$))
      .subscribe(profile => this.profile = profile);
  }

  ngOnDestroy(): void {
    this.destroy$.next();
    this.destroy$.complete();
  }

  get isPending(): boolean {
    return !this.profile;
  }
```

```
  openChangePassword(): void {
    this.dialog.open(ChangePasswordDialogComponent, {
      minWidth: 320,
      disableClose: true
    });
  }

  openSessions(): void {
    this.accountService.getSessions().subscribe(sessions => {
      this.dialog.open(SessionListDialogComponent, {
        minWidth: 320,
        maxHeight: 600,
        data: sessions
      });
    });
  }

  avatarFileChanged(e: any): void {
    this.newAvatarFile = e.target.files[0] as File;
  }

  uploadNewAvatar(): void {
    if (!this.newAvatarFile) {
      return;
    }

    this.profileService.uploadNewProfilePicture(this.newAvatarFile)
      .pipe(takeUntil(this.destroy$))
      .subscribe(() => this.getMe());
  }

  deleteCurrentAvatar(): void {
    this.profileService.deleteProfilePicture()
      .pipe(takeUntil(this.destroy$))
      .subscribe(() => this.getMe());
  }
}
```

## А.19 Програмний код компоненту ChatsComponent

### Розмітка:

```
<mat-sidenav-container class="container">
  <mat-sidenav class="sidenav" mode="side" opened>
    <mat-nav-list>
```

```
    <div *ngFor="let chat of chatsStorageService.allChats;">
      <app-chat-item
        [isActive]="chat.id === chatsStorageService.currentChat?.id"
        title="{{chat.title}}"
        lastMessageText="{{chat.lastMessageText}}"
        lastMessageAuthor="{{chat.lastMessageAuthor}}"
        chatLogoUrl="{{chat.chatType === 0
          ? this.resourcesService.createPathToGroupAvatar(chat.imageFileName)
          : this.resourcesService.createPathToUserAvatar(chat.imageFileName)}}"
        (click)="openChat(chat.id)"/>
    </div>
  </mat-nav-list>
 </mat-sidenav>
 <mat-sidenav-content>
  <app-chat-container
    [chat]="chatsStorageService.currentChat"
    [messages]="chatsStorageService.currentChatMessages"/>
 </mat-sidenav-content>
</mat-sidenav-container>
```

## Стилі:

```
.container {
 position: relative;
 top: 0;
 bottom: 0;
 left: 0;
 right: 0;
 background: #eee;
 min-height: 100vh;
}

.sidenav {
 min-width: 300px;
}
```

## Код:

```
import {Component, OnDestroy, OnInit} from '@angular/core';
import {ChatsService} from "../../../../api/services/chats.service";
import {MessagesService} from "../../../../api/services/messages.service";
import {ChatsStorageService} from "../../../services/chats-storage.service";
import {SignalRService} from "../../../../api/services/signal-r.service";
import {Subject, takeUntil} from "rxjs";
import {ResourcesService} from "../../../../api/services/resources.service";
```

```
@Component({
  selector: 'app-chats',
  templateUrl: './chats.component.html',
  styleUrls: ['./chats.component.css']
})
export class ChatsComponent implements OnInit, OnDestroy {
  private destroy$ = new Subject<void>();

  constructor(
    private chatsService: ChatsService,
    private messagesService: MessagesService,
    private signalRService: SignalRService,
    public resourcesService: ResourcesService,
    public chatsStorageService: ChatsStorageService) {}

  ngOnInit(): void {
    this.signalRService.openConnection();

    this.chatsService.getChats().subscribe(res => {
      this.chatsStorageService.allChats = res;
    });

    this.signalRService.$onChatCreated
      .asObservable()
      .pipe(takeUntil(this.destroy$))
      .subscribe(notification => {
        this.chatsService.getMyChatById(notification.chatId).subscribe(chat => {
          this.chatsStorageService.allChats.push(chat);
        });
      });

    this.signalRService.$onRemovedFromChat
      .asObservable()
      .pipe(takeUntil(this.destroy$))
      .subscribe(notification => {
        this.chatsStorageService.allChats = this.chatsStorageService.allChats.filter(x => x.id != notification.chatId);
        if (this.chatsStorageService.currentChat?.id == notification.chatId) {
          this.chatsStorageService.currentChat = undefined;
        }
      });

    this.signalRService.$onMessageReceived
      .asObservable()
      .pipe(takeUntil(this.destroy$))
      .subscribe(notification => {
        const chatIndex = this.chatsStorageService.allChats.findIndex(x => x.id == notification.chatId)
```

```
      this.chatsStorageService.allChats[chatIndex].lastMessageText = notification.textContent;
      this.chatsStorageService.allChats[chatIndex].lastMessageAuthor = notification.authorName;
      this.chatsStorageService.allChats[chatIndex].lastMessageAuthorId = notification.authorId;
      this.chatsStorageService.allChats[chatIndex].lastMessageDate = notification.createdAt;
    })
  }

  ngOnDestroy(): void {
    this.destroy$.next();
    this.destroy$.complete();
  }

  openChat(chatId: string): void {
    this.chatsStorageService.currentChat = this.chatsStorageService.allChats.find(x => x.id == chatId);
    this.messagesService.getMessages(chatId).subscribe(messages => {
      this.chatsStorageService.currentChatMessages = messages;
    });
  }
}
```

## A.20 Програмний код компоненту ChatContainerComponent

Розмітка:

```
<div class="container">
  <div *ngIf="chat else selectChatToStartMessaging">
    <div class="box">
      <div class="header-container">
        <app-chat-header
          [chatId]="chat.id"
          [chatTitle]="chat.title"
          [chatMembersCount]="chat.membersCount"
          [isChatOwner]="chat.isOwner"
          (titleClick)="openDetailedChatCard()"
          (leaveFromChatClick)="leaveFromChat()"
          (deleteChatClick)="deleteChat()"
          (addUserToChatClick)="addUserToChat()"
        />
      </div>
      <div cdkScrollable class="messages-container">
        <app-chat-message *ngFor="let message of messages"
                          [authorName]="message.authorName"
                          [messageText]="message.textContent"
                          [createdAt]="message.createdAt"
                          [isMyMessage]="message.authorId === currentUserId"
                          [avatarUrl]="resourcesService.createPathToUserAvatar(message.authorImageName)"
```

```
      />
    </div>
    <div class="chat-input-container">
      <app-chat-input (newMessage)="sendNewMessage($event)"/>
    </div>
   </div>
  </div>
  <ng-template #selectChatToStartMessaging>
   <div class="select-chat-message">
    Select chat to start messaging.
   </div>
  </ng-template>
</div>
```

## Стилі:

```
.select-chat-message {
  text-align: center;
  vertical-align: center;
}

.box {
  display: flex;
  flex-flow: column;
  height: 100vh;
}

.header-container {
  flex: 0 1 auto;
}

.messages-container {
  flex: 1 1 auto;
  max-height: 100vh;
  overflow: auto;
  display: flex;
  flex-direction: column-reverse;
}

.chat-input-container {
  flex: 0 1 auto;
}
```

## Код:

```
import {Component, Input, OnDestroy, OnInit} from '@angular/core';
```

```
import {Message} from "../../../api/types/messages/message";
import {MessagesService} from "../../../api/services/messages.service";
import {SignalRService} from "../../../api/services/signal-r.service";
import {Chat} from "../../../api/types/chats/chat";
import {SendMessageRequest} from "../../../api/types/messages/send-message-request";
import {Subject, takeUntil} from "rxjs";
import {ChatsStorageService} from "../../services/chats-storage.service";
import {MatDialog} from "@angular/material/dialog";
import {ChatCardComponent} from "../cards/chat-card/chat-card.component";
import {TokensService} from "../../services/tokens.service";
import {ResourcesService} from "../../../api/services/resources.service";
import {ChatsService} from "../../../api/services/chats.service";
import {InviteUserToChatDialog} from "../dialogs/invite-user-to-chat-dialog/invite-user-to-chat-dialog";

@Component({
  selector: 'app-chat-container',
  templateUrl: './chat-container.component.html',
  styleUrls: ['./chat-container.component.css']
})
export class ChatContainerComponent implements OnInit, OnDestroy {
  private destroy$ = new Subject<void>();
  private _chat?: Chat;
  @Input() set chat(value: Chat | undefined) {
    this._chat = value;

    if (this._chat) {
      this.signalRService.subscribeToChatEvents(this._chat.id);
      this.destroy$.next();
      this.signalRService.$onMessageReceived
        .asObservable()
        .pipe(takeUntil(this.destroy$))
        .subscribe((notification) => {
          const newMessage = {
            id: notification.messageId,
            authorId: notification.authorId,
            authorName: notification.authorName,
            authorImageName: notification.authorImage,
            createdAt: notification.createdAt,
            replyToMessageId: notification.inReplyToMessage,
            textContent: notification.textContent
          } as Message;

          this.chatsStorageService.currentChatMessages?.unshift(newMessage);
        })
    }
  }
```

```
get chat(): Chat | undefined {
  return this._chat;
}

@Input() messages?: Message[];

constructor(
  private messagesService: MessagesService,
  private chatsService: ChatsService,
  private signalRService: SignalRService,
  private chatsStorageService: ChatsStorageService,
  private dialog: MatDialog,
  private tokensService: TokensService,
  public resourcesService: ResourcesService) {}

ngOnInit(): void {

}

ngOnDestroy(): void {
  this.destroy$.next();
  this.destroy$.complete();
}

sendNewMessage(messageText: string): void {
  const sendMessageRequest = {
    chatId: this.chat?.id,
    textContent: messageText
  } as SendMessageRequest;

  this.messagesService.sendMessage(sendMessageRequest).subscribe();
}

openDetailedChatCard(): void {
  this.dialog.open(ChatCardComponent, {
    minWidth: 500,
    minHeight: 500,
    data: this.chatsStorageService.currentChat
  });
}

leaveFromChat(): void {
  if (!this._chat) {
    return;
  }
```

```
    this.chatsService
      .leaveFromChat(this._chat.id)
      .subscribe();
  }

  deleteChat(): void {
    if (!this._chat) {
      return;
    }

    this.chatsService
      .deleteChat(this._chat.id)
      .subscribe();
  }

  addUserToChat(): void {
    const dialogRef = this.dialog.open(InviteUserToChatDialog, {
      minWidth: 500,
      minHeight: 600,
      data: this.chatsStorageService.currentChat?.id
    });

    dialogRef.afterClosed().subscribe(() => {
      this.chatsService
        .getChats()
        .pipe(takeUntil(this.destroy$))
        .subscribe(res => {
          this.chatsStorageService.allChats = res;
          this.chatsStorageService.currentChat = this.chatsStorageService.allChats
            .find(x => x.id == this.chatsStorageService.currentChat?.id);
        });
    });
  }

  get currentUserId() {
    return this.tokensService.getCurrentUserId();
  }
}
```

ДОДАТОК Б

(обов'язковий)

# ВИГЛЯД КОРИСТУВАЦЬКОГО ІНТЕРФЕЙСУ

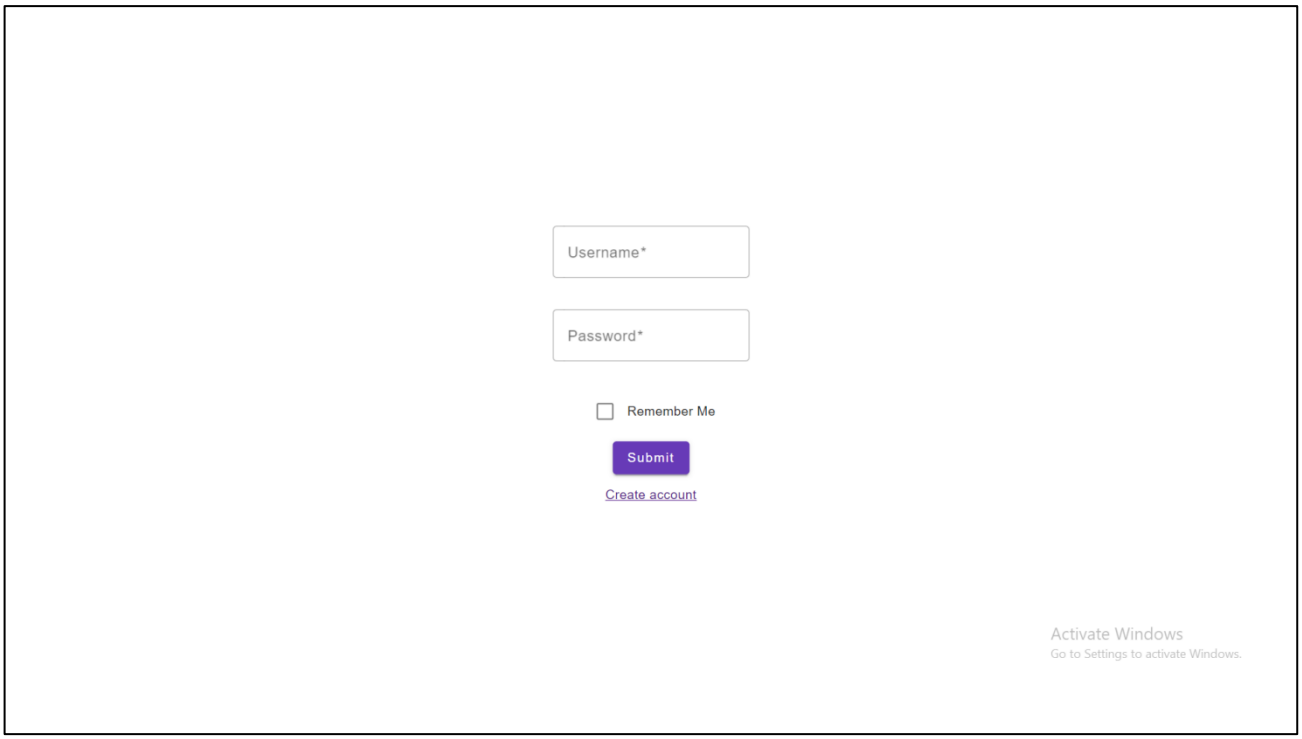



Рисунок Б.1 − Сторінка логіну.

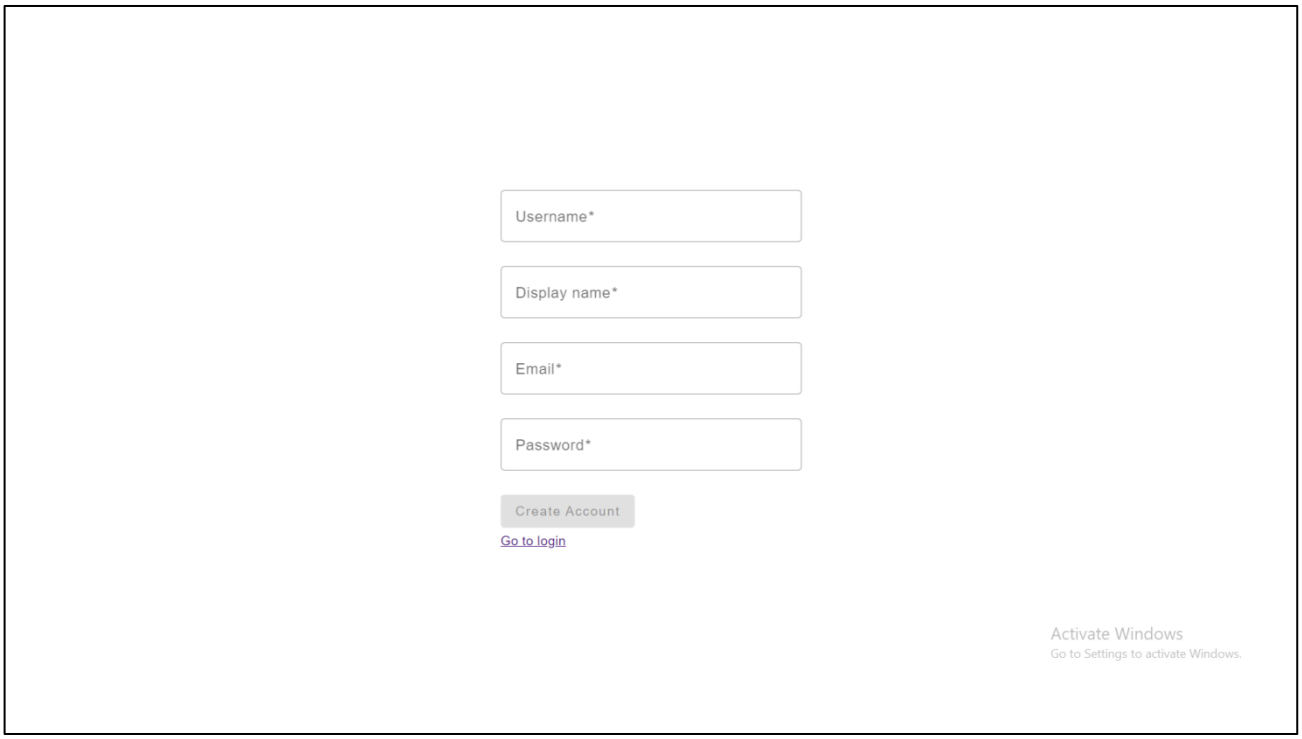



Рисунок Б.2 − Сторінка реєстрації.

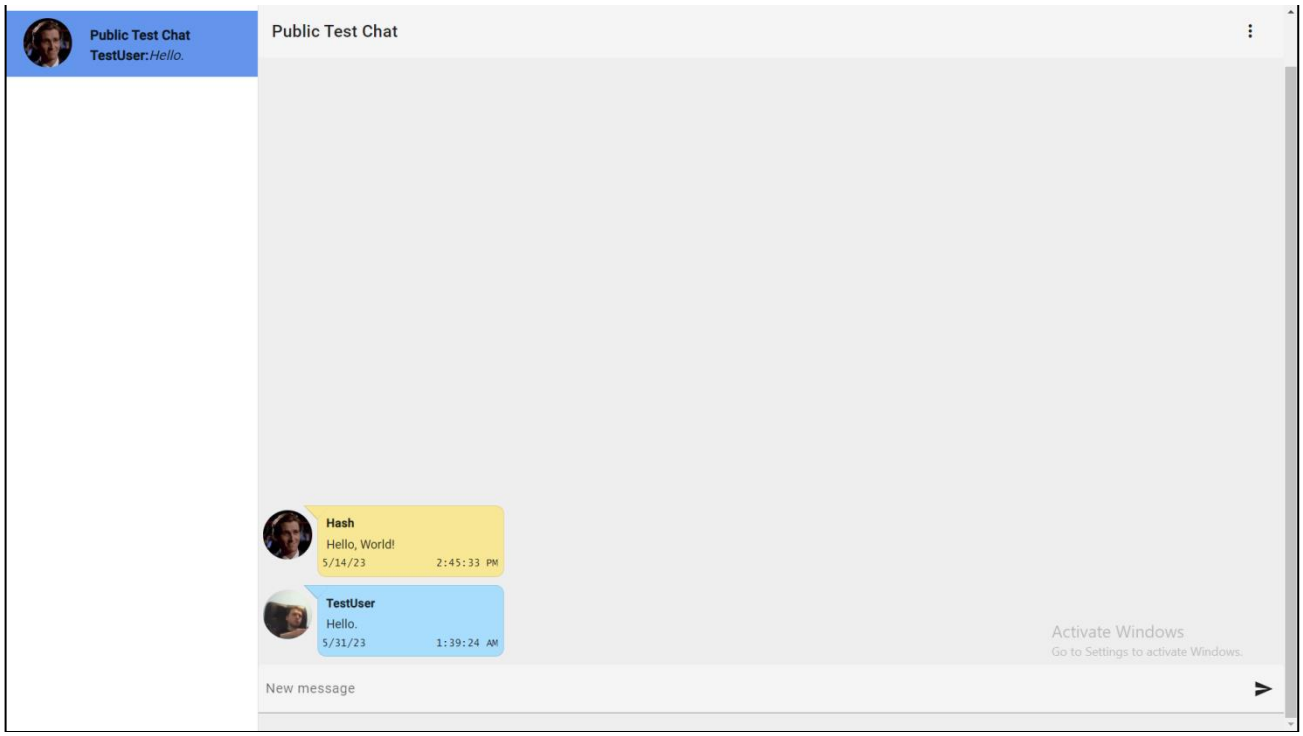



Рисунок Б.3 − Сторінка чатів.

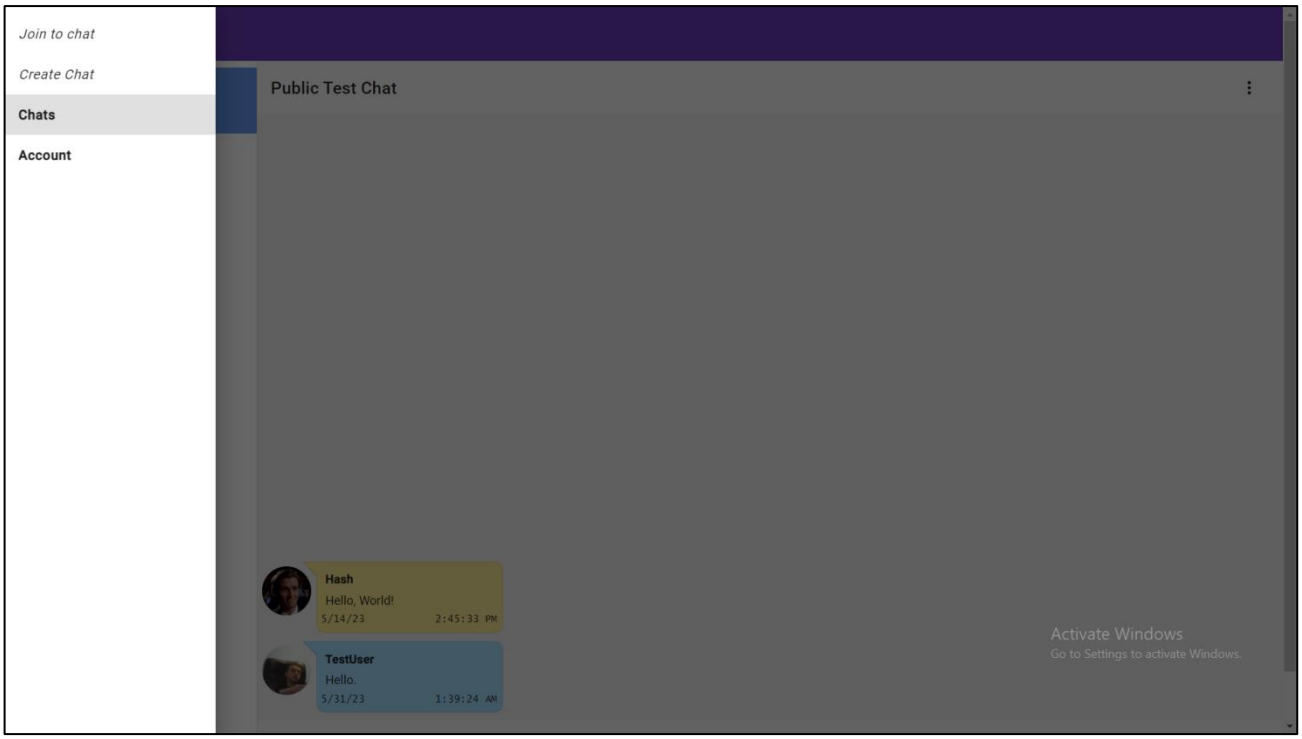



Рисунок Б.4 − Сторінка чатів (з відкритим меню).

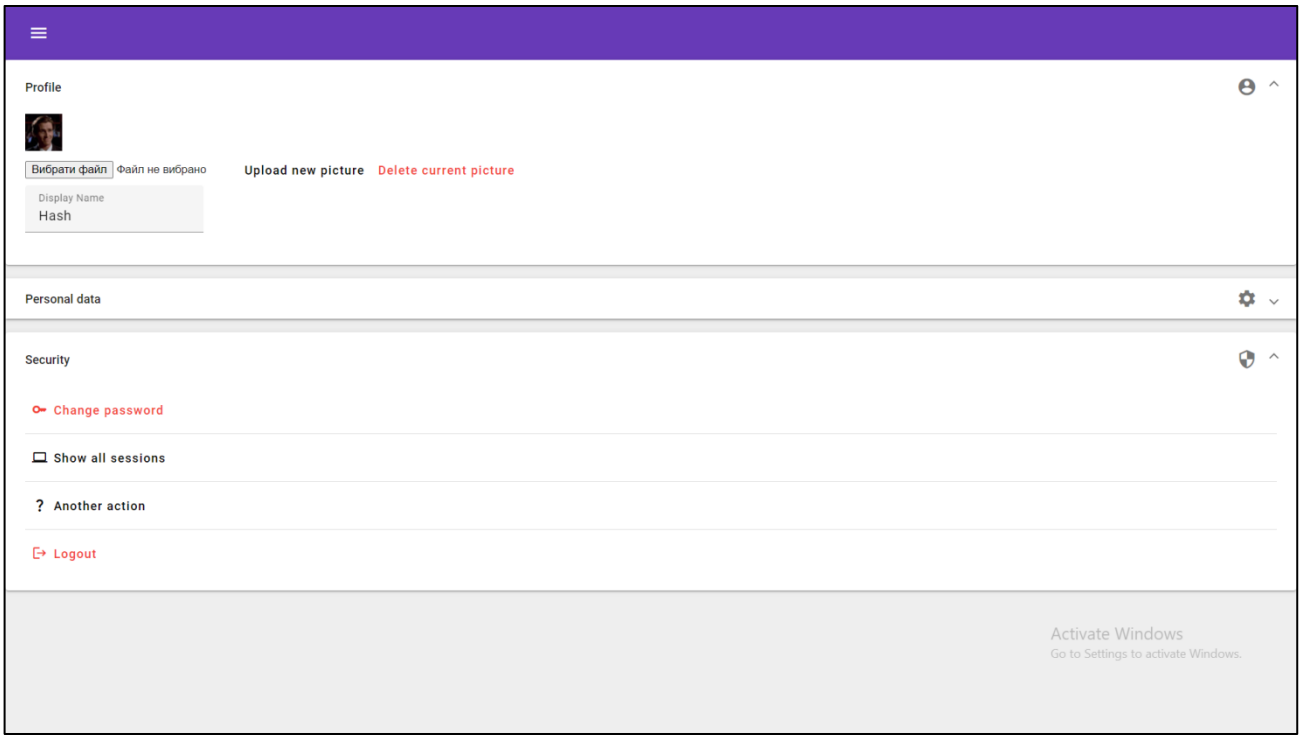



Рисунок Б.5 − Сторінка керуванням обліковим записом.

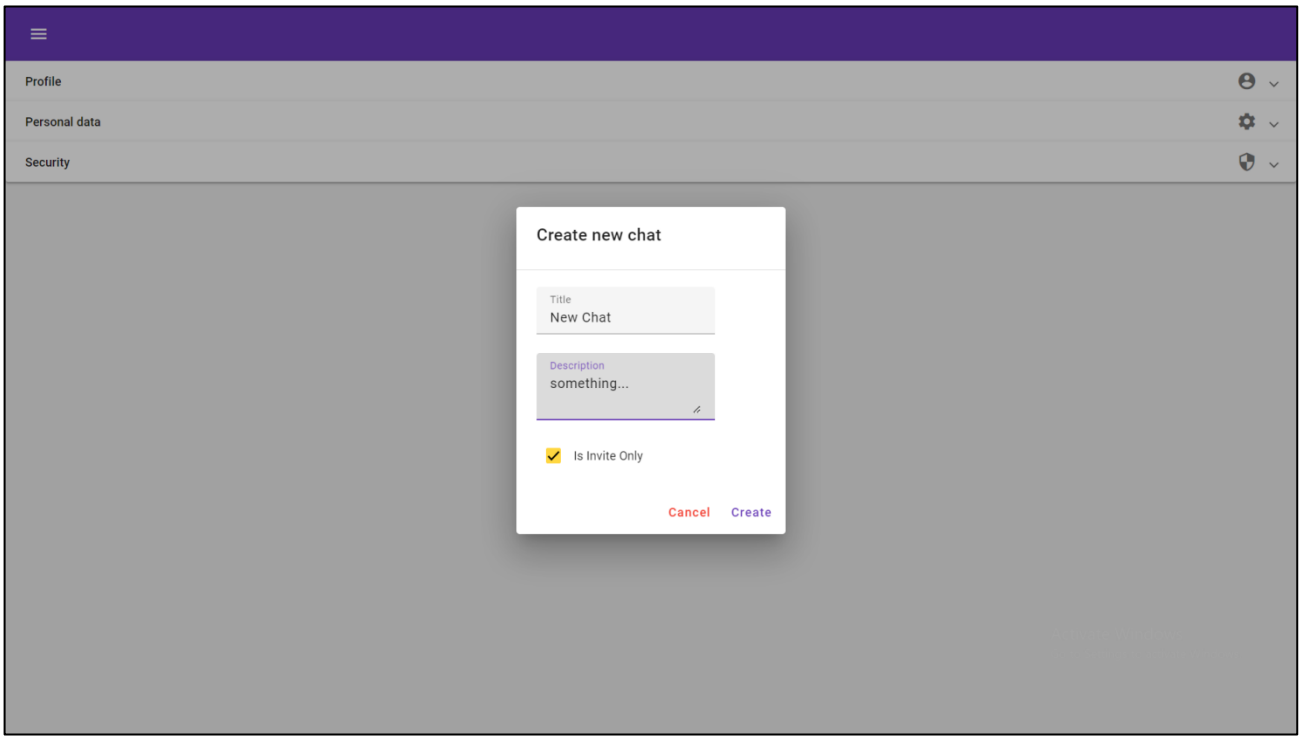
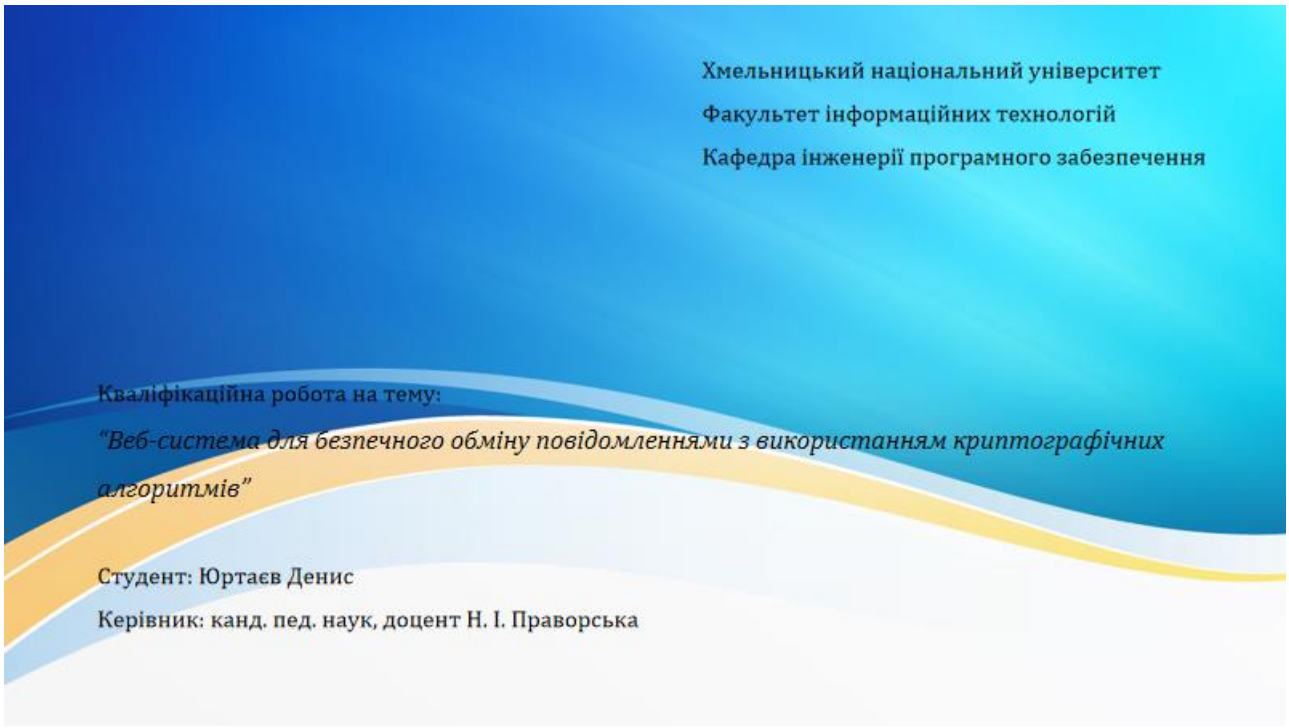


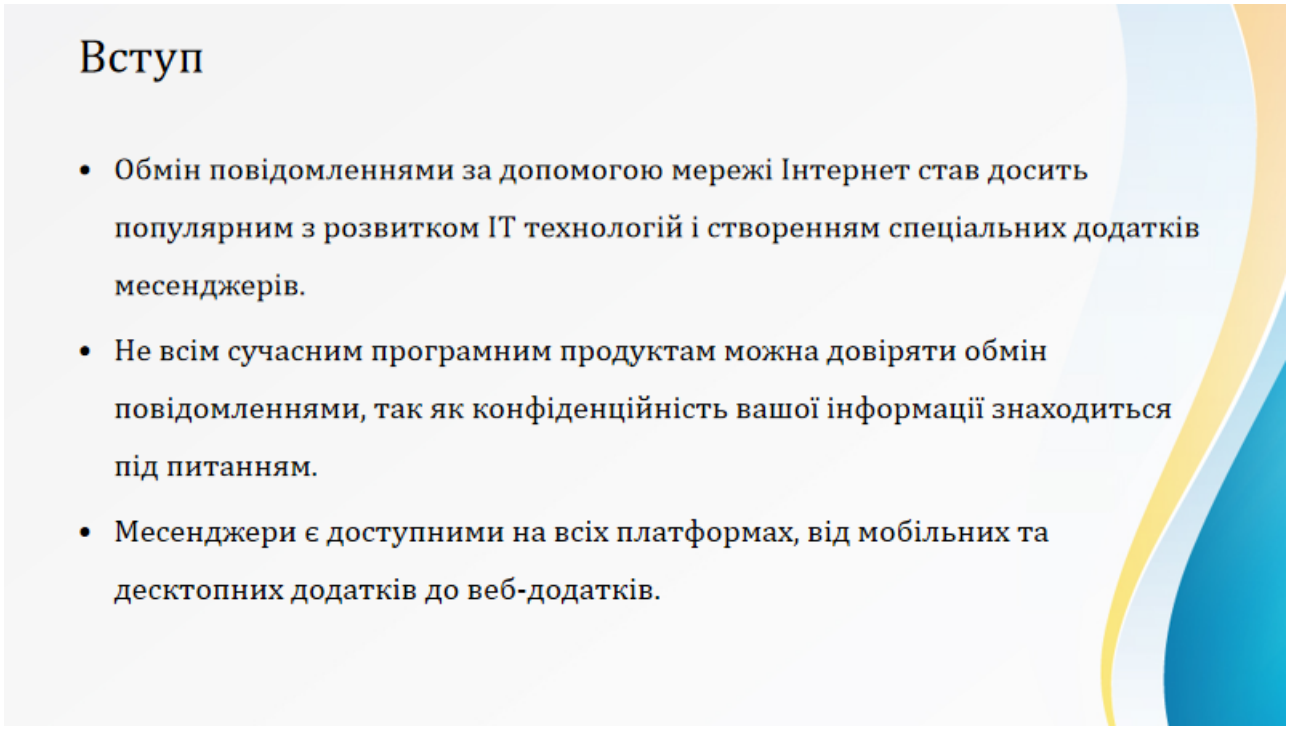

Рисунок Б.6 − Діалог для створення чату.

ДОДАТОК В

(обов’язковий)

# ПРЕЗЕНТАЦІЙНІ МАТЕРІАЛИ



## Актуальність

- Актуальність теми роботи полягає у розробці простої програмної системи, яка дозволить користувачам безпечно обмінюватись повідомленнями у режимі реального часу за лічені секунди.
- Метою проекту є розробка системи, яка буде представлена у вигляді клієнтського додатку та серверу, матиме простий інтерфейс та використовуватиме алгоритми шифрування.

## Завдання

- виконати аналіз існуючих програмних систем;
- проаналізувати алгоритми шифрування, які підійдуть для інтеграції в систему месенджера;
- розробити клієнтський веб-додаток;
- розробити серверний додаток;
- провести тестування додатків.

## Наявне програмне забезпечення

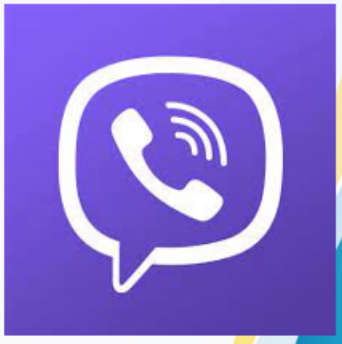
### Telegram

- обмін як текстовими повідомленнями так і аудіо або відео-повідомленнями;
- система ботів;
- секретні чати, які використовують e2e шифрування;
- дзвінки (також з використанням e2e шифрування)
- створення чатів та каналів;
- можливість зберігати досить багато файлів на хмарному сервері розміром до декількох GB.

## Наявне програмне забезпечення

### Viber

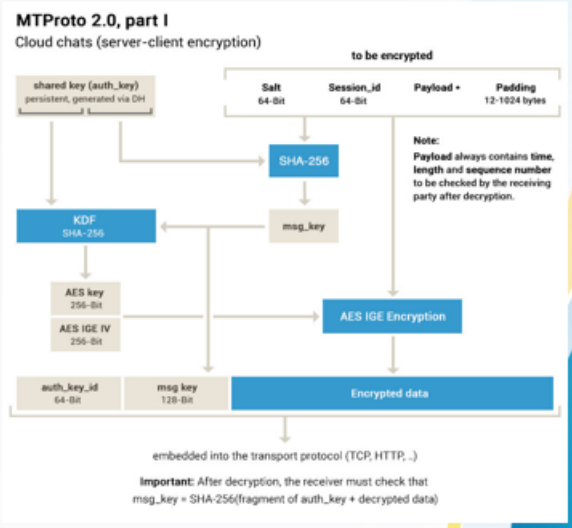
- створення чатів та каналів;
- аудіо та відео виклики;
- обмін файлами та зображеннями;
- багатоплатформенний;
- досить простий UI.



# Аналіз способів шифрування

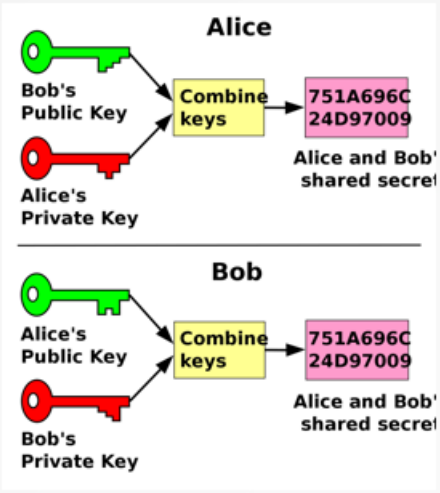
Телеграм використовує протокол MTProto, який діє навіть у випадку використання звичайних чатів, а не тільки секретних чатів.

- HTTP(S)
- WebSockets
- WebSockets over HTTPS
- TCP

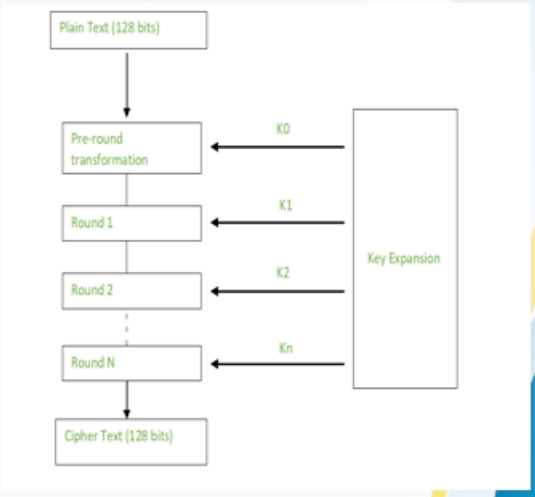



# Аналіз способів шифрування

DH (diffie hellman)

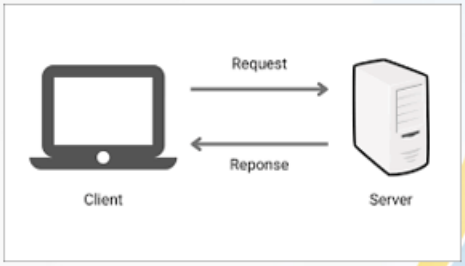



AES



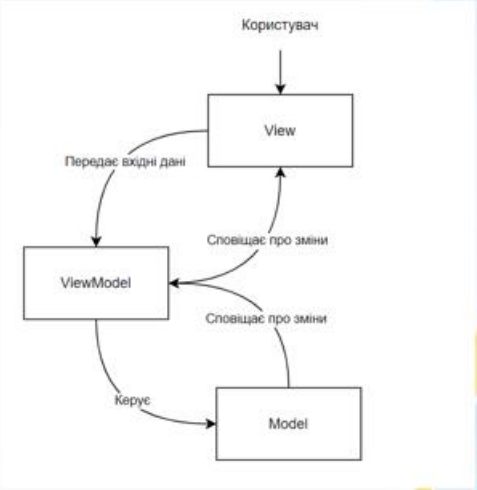
# Проектування

Для проектування системи було використано клієнт-серверну архітектуру.




# Проектування

Для проектування клієнтського додатку було обрано MVVM шаблон.

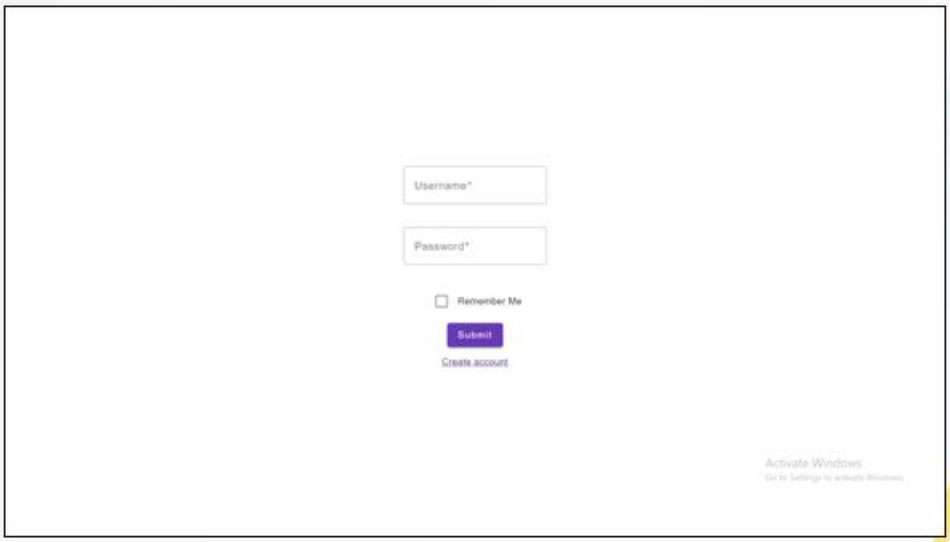


# Способи реалізації

Для реалізації клієнтського додатку було використано Angular фреймворк, а також MVVM шаблон який вже є вбудованим в даний фреймворк.

В той час для реалізації серверного додатко було використано ASP.NET Core фреймворк та платформу .NET 7.

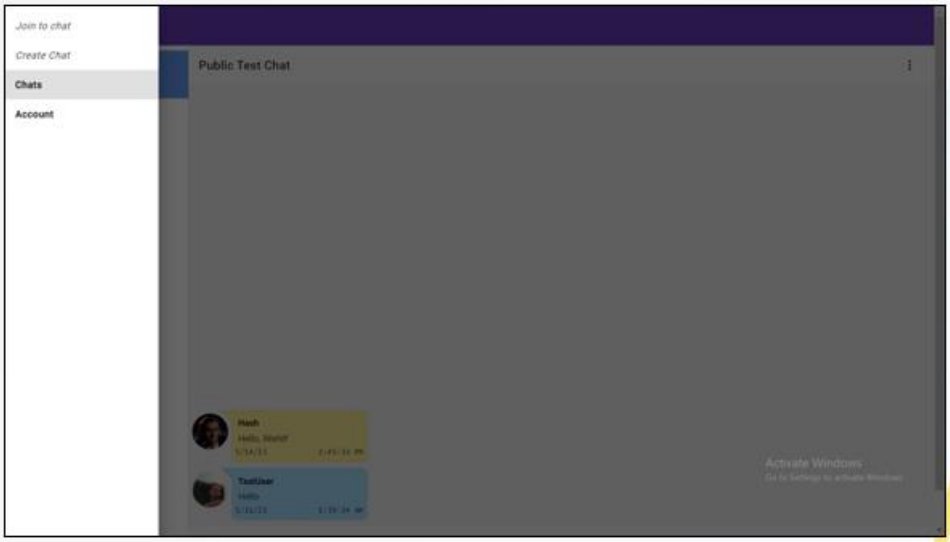

# Результати роботи



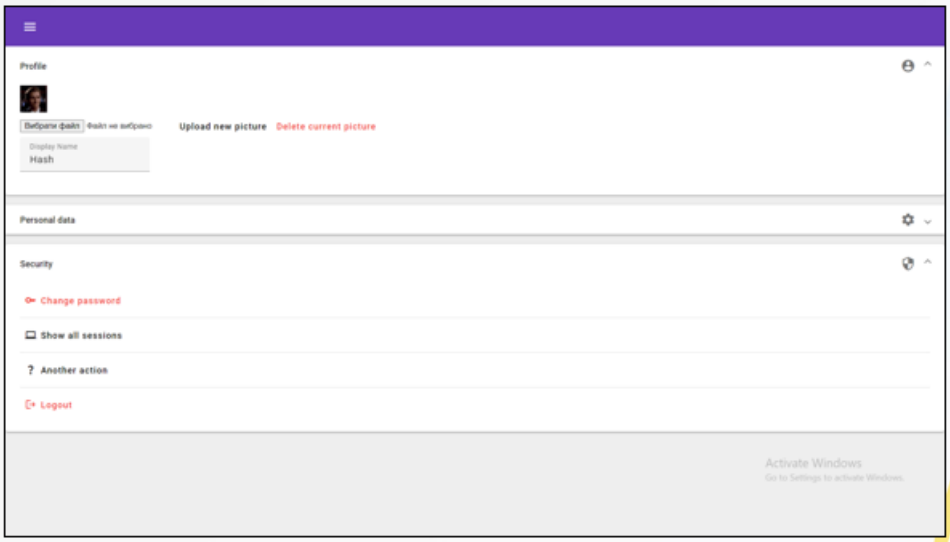
# Результати роботи


# Результати роботи

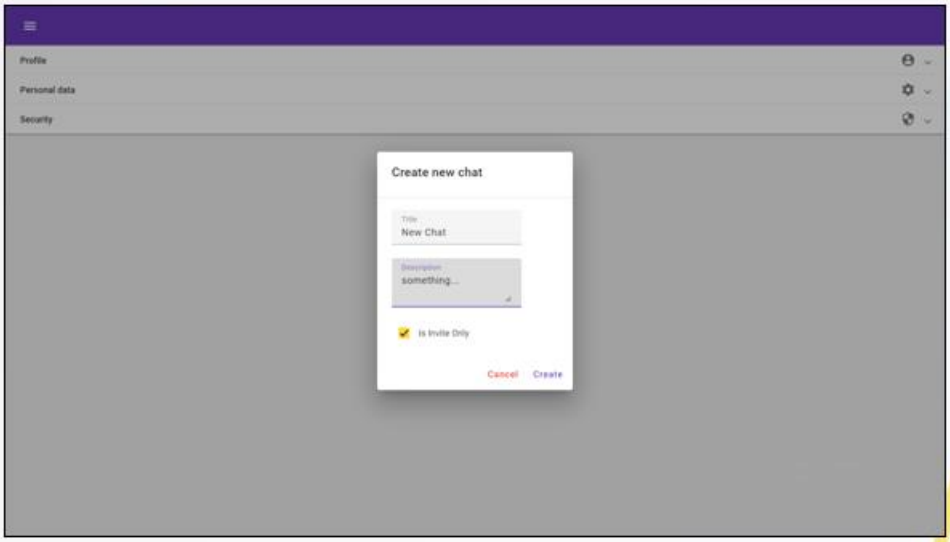
# Результати роботи




# Результати роботи



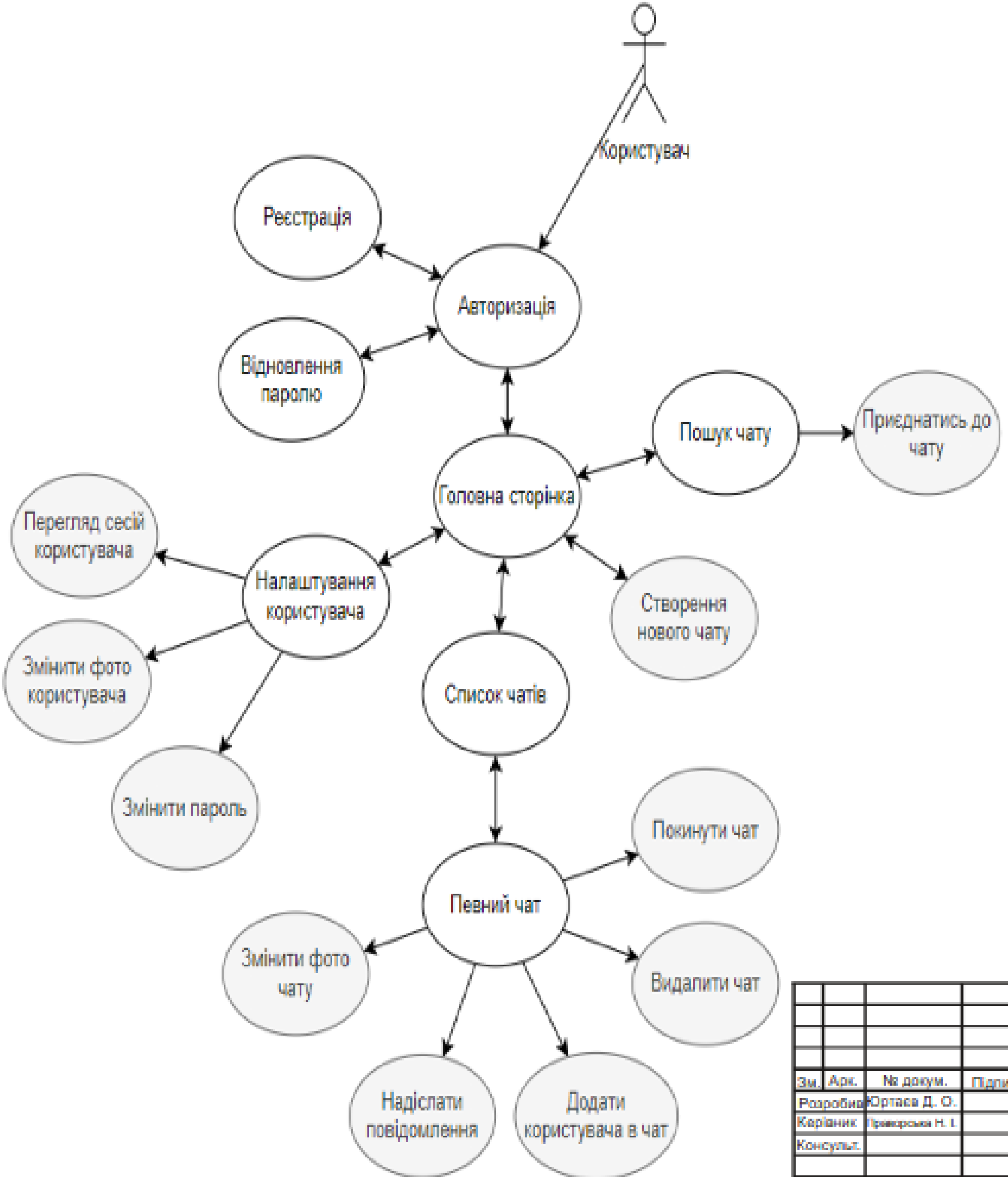
# Результати роботи



ДОДАТОК Г

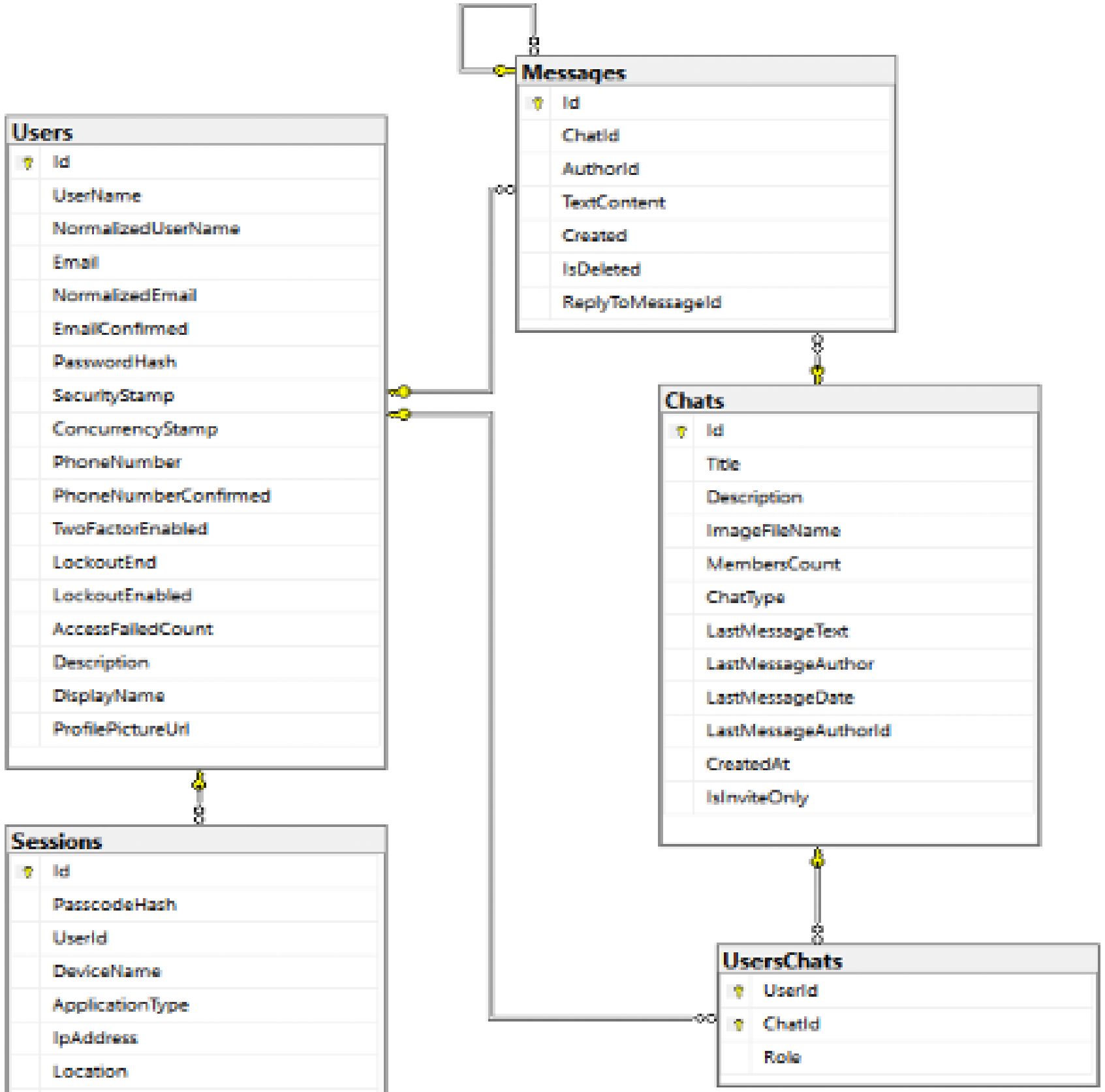
(обов’язковий)

# ГРАФІЧНА ЧАСТИНА

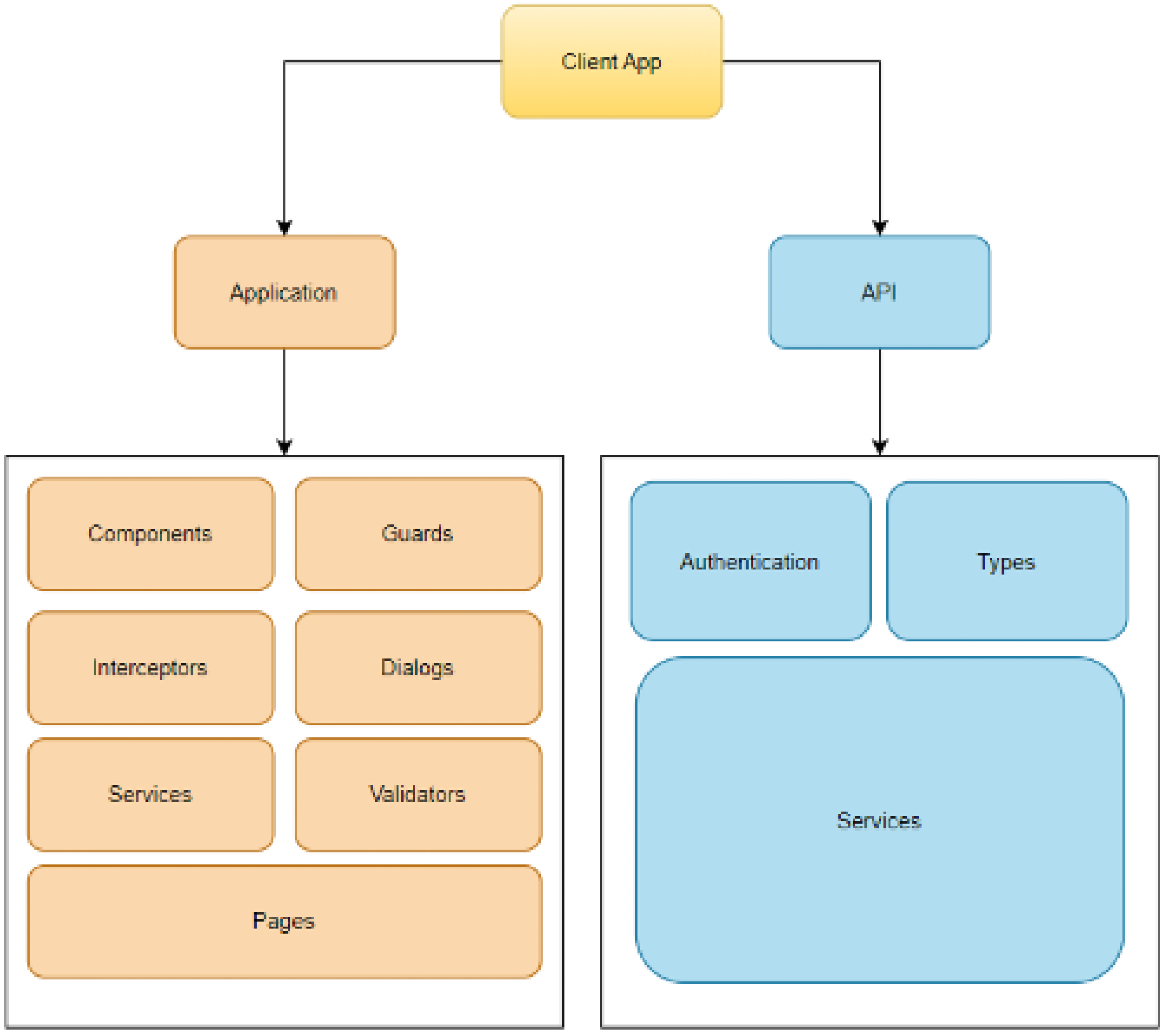



| | | | | | | |
|---|---|---|---|---|---|---|
| | | | | | КвРІПЗ.190145.19.23.Е8 | |
| Зм. | Арк. | № докум. | Підпис | Дата | | |
| Розробив | | Юртаєв Д. О. | | | Діаграма варіантів використання | Літера / Маса / Масштаб |
| Керівник | | Тарнавська Н. І. | | | | Аркуш 1 / Аркушів 3 |
| Консульт. | | | | | | ХНУ, ІПЗ-19-1 |
| Н. Контр. | | Нашин О. М. | | | | |
| Зав. каф. | | Бондаренко П. П. | | | | |

129




| | | | | | | |
|---|---|---|---|---|---|---|
| | | | | | КвРІПЗ.190145.19.23.Е8 | |
| Зм. | Арк. | № докум. | Підпис | Дата | ER-діаграма бази даних | Літера / Маса / Масштаб |
| Розробив | | Юртаєв Д. О. | | | | |
| Керівник | | Іванюрська Н. І. | | | | Аркуш 2 / Аркушів 3 |
| Консульт. | | | | | | |
| Н. Контр. | | Іванюк О. М. | | | | ХНУ, ІПЗ-19-1 |
| Зав. каф. | | Бородатюк П. П. | | | | |




| | | | | | КвРІПЗ.190145.19.23.Е8 | | |
|---|---|---|---|---|---|---|---|
| | | | | | | Літера | Маса | Масштаб |
| Зм. | Арк. | № докум. | Підпис | Дата | Діаграма модулів клієнтського веб-застосунку | | |
| Розробив | | Юртаєв Д. О. | | | | Аркуш 3 | Аркушів 3 |
| Керівник | | Грайворська Н. І. | | | | ХНУ, ІПЗ-19-1 | |
| Консульт. | | | | | | | |
| Н. Контр. | | Яшина О. М. | | | | | |
| Зав. каф. | | Бадратих І. Г. | | | | | |

# СУПРОВІДНІ ДОКУМЕНТИ

Завідувачу кафедри інженерії програмного забезпечення проф. Бедратюку Л. П.

здобувача вищої освіти
Юртаєва Д. О.
Прізвище, ініціали

факультет ІТ, 4 курс, група ІПЗ-19-1

**ЗАЯВА**

З правилами чинного Положення «Про систему забезпечення академічної доброчесності в Хмельницькому національному університеті» від 01.07.2022, згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту і застосування заходів дисциплінарної та академічної відповідальності, ознайомлений. Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на плагіат оповіщений та надаю свою згоду на обробку й збереження університетом моєї роботи в інституційному репозитарії університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-технічних засобів (Unicheck та/або Anti-Plagiarism) і використання роботи для виявлення академічного плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення плагіату в текстах робіт.

Робота для перевірки університетом надається в друкованому та електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

05.02.2023
дата

підпис

Mon Jun 05 17:23:42 EEST 2023. Хіврич Володимир Русланович. Хмельницький національний університет. Х

# Anti-Plagiarism v-15.257

**Максимальне співпадіння з одним документом 15.0%**

Словники перевірки: en_US, ru_RU, ua_UA. **Помилок в документах: 15%**

| ID: 114839<br>Назва: БКР Веб-система для безпечного обміну повідомленнями з використанням криптографічних алгоритмів<br>Додано в БД: 2023-06-05<br>Автора: Юртаєв Д.О.<br>Керівники: Праворська Н.І. к.п.н. доц.<br>Консультанти:<br>Опоненти: | Документ | | Сумарний збіг по Базі Даних | |
|---|---|---|---|---|
| | Символи | Лексеми | Символи | Лексеми |
| | 65414 | 584 | 15797 (24%) | 139 (24%) |

Джерело плагіату

| ID | Опис | Наявність плагіату в документ | |
|---|---|---|---|
| | | Символи | Лексеми |
| 111875 | Назва: Звіт з переддипломної практики бакалавр на тему: Веб-система для безпечного обміну повідомленнями з використанням криптографічних алгоритмів<br>Додано в БД: 2023-03-02<br>Автора: Юртаєва Д.О.<br>Керівники: Праворська Н.І.<br>Консультанти:<br>Опоненти: | 9941 (15.0%) | 78 (13.0%) |

UNICHECK

Ім'я користувача:
Кафедра ІПЗ

ID перевірки:
1015440026

Дата перевірки:
05.06.2023 18:21:41 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
05.06.2023 18:24:20 EEST

ID користувача:
100005589

Назва документа: Дипломний_проект_02_06_23_для_антиплагіату

Кількість сторінок: 63 Кількість слів: 11018 Кількість символів: 88174 Розмір файлу: 965.68 KB ID файлу: 1015100797

Виявлено модифікації тексту (можуть впливати на відсоток схожості)

## 12.9%
## Схожість

Найбільша схожість: 5.75% з джерелом з Бібліотеки (ID файлу: 1015045630)

6.75% Джерела з Інтернету | 827 ..... Сторінка 65

10.2% Джерела з Бібліотеки | 101 ..... Сторінка 69

## 0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

## 0%
## Вилучень

Немає вилучених джерел

## Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Підозріле форматування | 19 сторінок

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

**РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ**
освітнього ступеня «Бакалавр»

Дипломник Юртаєв Денис Олександрович

Тема Веб-система для безпечного обміну повідомленнями з використанням криптографіних алгоритмів

Спеціальність 121 – Інженерія програмного забезпечення

**Обсяг кваліфікаційної роботи:**

Кількість листів креслень 3; кількість сторінок записки 62

1.Короткий зміст пояснювальної записки та прийнятих рішень У процесі виконання кваліфікаційної роботи здійснено дослідження та аналіз предметної області, встановлені функціональні та нефункціональні вимоги. Проведений огляд наявних програм-аналогів на ринку, розглянуті їх переваги та недоліки і підтверджена актуальність розробки програмної системи. Було розглянуто різні інструменти та засоби реалізації програмного забезпечення. В результаті було розроблено програмний продукт.

2. Висновок про відповідність роботи поставленому завданню Кваліфікаційна робота виконана згідно з поставленим завданням і у відповідності до всіх вимог.

3. Характеристика виконання кожного розділу роботи, ступінь використання останніх досягнень науки і техніки та передових методів роботи У вступі була обгрунтована актуальність теми, сформульовано мету та завдання роботи. У першому розділі проведений аналіз предметної області, розглянуто наявні рішення та визначено функціональні та нефункціональні вимоги до розроблюваного проекту. У другому розділі проведено аналіз сучасних архітектур та шаблонів проектування. У третьому розділі були підготовлені усі необхідні залежності для написання коду, виконана практична розробка програмних модулів і описані їх особливості, що призвело до створення програмного продукту. Також було проведене ручне тестування розробленого продукту.

4. Позитивні сторони роботи Тематика кваліфікаційної роботи є досить актуальною. Також було застосовано новітні технології та актуальні архітектурні рішення.

5. Негативні сторони роботи Не достатньо зрозумілі відповіді від сервера під час помилок. Клієнт доступний лише у вигляді веб-застосунку і в першу чергу підлаштований під користувачів ПК.

6. Оцінка графічного оформлення та пояснювальної записки Графічне оформлення виконано відповідно до тематики кваліфікаційної роботи. Представлене у вигляді діаграм та рисунків. Пояснювальна записка оформлена відповідно до вимог

7. Відгук про кваліфікаційну роботу в цілому Кваліфікаційна робота є достойною та заслуговує позитивної оцінки. Матеріал пояснювальної записки структурований, послідовний, чіткий та легко зрозумілий, що дозволило чітко осягнути викладений матеріал в рамках теми проектування.

8. Інші зауваження

9. Оцінка кваліфікаційної роботи Кваліфікаційна робота виконана у повному обсязі, відповідає поставленій задачі та заслуговує оцінку «задовільно».

РЕЦЕНЗЕНТ РЕЦЕНЗЕНТ Мартинюк Валерій Володомирович, професор, доктор технічних наук, зав. кафедри автоматизації, комп'ютерно-інтегрованих технологій та робототехніки ХНУ.

" 7 " 06 2023 р. _____
(підпис)