НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено:
Завідувач кафедри
_____ Сергій СТІРЕНКО_
(підпис)
“__” _____2024 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою “Комп’ютерні системи та мережі”

спеціальності 123 “Комп’ютерна інженерія”

на тему: Веб-застосунок для спілкування з використанням технологій шифрування та конфіденційності даних

Виконав: студент 4 курсу, групи ІО-01
(шифр групи)

Глюза Андрій Іванович
(прізвище, ім’я, по батькові) (підпис)

Керівник асистент Череватенко Олексій Володимирович
(посада, науковий ступінь, вчене звання, прізвище, ім’я, по батькові) (підпис)

Консультант (нормоконтроль) ст. викладач Виноградов Ю. М.
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент доцент кафедри ІСТ, к.т.н., Шимкович Володимир Миколайович
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2024 р.

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалавр)

Освітньо-професійна програма

“Комп’ютерні системи та мережі”

спеціальність 123 “Комп’ютерна інженерія”

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Сергій СТІРЕНКО
(підпис)
“__” _____2024 р.

ЗАВДАННЯ
на бакалаврський дипломний проєкт студента

Глюзи Андрія Івановича

1. Тема проєкту *Веб-застосунок для спілкування з використанням технологій шифрування та конфіденційності даних*
керівник проєкту асистент Череватенко Олексій Володимирович,
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від 27 травня 2024 року № 2112-с
2. Термін здачі студентом закінченого проєкту *3 червня 2024 р.*
3. Вихідні дані до проєкту *технічна документація, теоретичні дані.*
4. Зміст розрахунково-пояснювальної записки (перелік питань, які розробляються)
Аналіз предметної області.
Огляд та вибір технологій для розробки.
Проектування та розробка системи.
Тестування роботи застосунку.
5. Перелік графічного матеріалу (з точним позначенням обов’язкових креслень)
Структура системи (Схема структурна), діаграма класів системи (Схема функціональна), алгоритм функціонування системи (Схема принципова).
6. Консультанта проєкту, з вказівкою розділів проєкту, які до них вносяться

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Нормоконтроль	Виноградов Ю. М.		

7. Дата видачі завдання «7» жовтня 2023 р.

Календарний план

№ п/п	Найменування етапів дипломного проєкту	Терміни виконання етапів проєкту	Примітки
1.	*Затвердження теми проєкту*	*15.03.2024 - 17.03.2024*	
2.	*Вивчення та аналіз завдання*	*17.03.2024 - 25.03.2024*	
3.	*Розробка архітектури та загальної структури системи*	*25.03.2024 - 05.04.2024*	
4.	*Розробка структур окремих підсистем*	*05.04.2024 - 15.04.2024*	
5.	*Програмна реалізація системи*	*15.04.2024 - 10.05.2024*	
6.	*Оформлення пояснювальної записки*	*10.05.2024 - 30.05.2024*	
7.	*Захист програмного продукту*	*04.06.2024*	
8.	*Передзахист*	*07.06.2024*	
9.	*Захист*	*22.06.2024*	

Студент-дипломник _____ ГЛЮЗА Андрій
(підпис)

Керівник проєкту _____ ЧЕРЕВАТЕНКО Олексій
(підпис)

АНОТАЦІЯ

В бакалаврському дипломному проєті було реалізовано BACK-END та FRONT-END частину веб-застосунку для спілкування. Метою розробки було створення вітчизняного месенджера з використанням технологій захищеного спілкування на основі шифрування даних та з залученням функцій адміністрування. Для цього в ході підготовки до розробки було проаналізовано сучасні аналоги подібних систем в результаті чого виділено стек технологій, що будуть використовуватись.

Для створення захищеного каналу зв'язку було обрано технологію на основі асиметричного шифрування та використано WebSockets. Побудова серверної частини здійснювалась з використанням мови програмування Java та фреймворку Spring та таких його складових як Spring-Boot та Spring Security. Для збереження інформації використовувалась база даних PostgreSQL. Клієнтська частина була створена з використанням сучаного та швидкого фреймворку ReactJS. В результаті розробки веб-застосунку було отримано готовий та масштабований продукт готовий для використання кінцевими користувачами.

Ключові слова: веб-застосунок, FRONT-END, BACK-END, WebSockets, Java, Spring, Spring-Boot, Spring Security, ReactJS.

ANNOTATION

In the Bachelor's degree project was implemented BACK-END and FRONT-END part of the web application for communication. The purpose of the development was to create a domestic messenger using secure communication technologies based on data encryption and involving administrative functions. In order to achieve this goal, modern analogs of similar systems were analyzed, resulting in the identification of the technology stack to be used.

A technology based on asymmetric encryption and WebSockets were chosen in order to establish a secure communication channel. The server side was built using the Java programming language and the Spring framework, including such components as Spring-Boot and Spring Security. PostgreSQL database was used to store data. The client-side was developed using modern and fast ReactJS framework. As a result of the web application development, a ready-to-use scalable product was obtained for end-users.

Keywords: web application, FRONT-END, BACK-END, WebSockets, Java, Spring, Spring-Boot, Spring Security, ReactJS.

Справки	Формат	*Значення*	*Найменування*	Кіл. листів	№ екземпляра	*Додаток*
			Документація загальна			
			Знову розроблена			
	А4	***ІАЛЦ.467100.002 ТЗ***	Веб-застосунок для спілкування	5		
			з використанням технологій			
			шифрування та			
			конфіденційності даних			
			Технічне завдання			
	А4	***ІАЛЦ.467100.003 ПЗ***	Веб-застосунок для спілкування	91		
			з використанням технологій			
			шифрування та			
			конфіденційності даних			
			Пояснювальна записка			
	А4	***ІАЛЦ.467100.004 Е1***	Веб-застосунок для спілкування	1		
			з використанням технологій			
			шифрування та			
			конфіденційності даних			
			Структура системи			
			Схема структурна			
	А4	***ІАЛЦ.467100.005 Д2***	Веб-застосунок для спілкування	1		
			з використанням технологій			
			шифрування та			
			конфіденційності даних			
			Діаграма класів системи			
			Схема функціональна			
	А4	***ІАЛЦ.467100.006 Д3***	Веб-застосунок для спілкування	1		
			з використанням технологій			
			шифрування та			
			конфіденційності даних			
			Алгоритм функціонування			
			системи			
			Схема принципова			
	А4	***ІАЛЦ.467100.007 Д4***	Веб-застосунок для спілкування	65		
			з використанням технологій			
			шифрування та			
			конфіденційності даних			
			Код застосунку			

					ІАЛЦ.467200.001 ОА
Зм.	*Арк.*	*№ докум.*	*Підпис*	*Дата*	
Розроб.		Глюза А. І.			Веб-застосунок для спілкування з використанням технологій шифрування та конфіденційності даних Опис Альбому
Перевір.		Череватенко О.В.			
Н. Контр.		Виноградов Ю.М.			
Затвердив					

Літ.	*Арк.*	*Аркушів*
	1	1

КПІ ім. Ігоря Сікорського ФІОТ, Група ІО-01

ТЕХНІЧНЕ ЗАВДАННЯ
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «*Веб-застосунок для спілкування з використанням технологій шифрування та конфіденційності даних*»

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ.....2
2. ПІДСТАВИ ДЛЯ РОЗРОБКИ.....2
3. МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ.....2
4. ДЖЕРЕЛА РОЗРОБКИ.....2
5. ТЕХНІЧНІ ВИМОГИ.....3
5.1. Вимоги до розробленого продукту.....3
5.2. Вимоги до програмного забезпечення.....4
5.3. Вимоги до апаратної частини.....4
6. ЕТАПИ РОЗРОБКИ.....5

					ІАЛЦ.467100.002 ТЗ		
Зм.	*Арк.*	*№ докум.*	*Підпис*	*Дата*			
Розроб.		Глюза А. І.			Веб-застосунок для спілкування з використанням технологій шифрування та конфіденційності даних Технічне завдання	*Літ.*	*Арк.* 1 / *Аркушів* 5
Перевір.		Череватенко О.В.					
Н. Контр.		Виноградов Ю.М.				***КПІ ім. Ігоря Сікорського ФІОТ, Група ІО-01***	
Затвердив							

1. НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Дане технічне завдання поширюється на розробку веб-застосунку для безпечного та захищеного спілкування користувачів з застосуванням технологій шифрування даних.

Область застосування: Вітчизняна альтернатива популярним світовим продуктам для обміну повідомленнями серед користувачів.

2. ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки даного веб-застосунку є поставлене завдання на виконання роботи кваліфікаційно-освітнього рівня «Бакалавр» спеціальності 123 «Комп'ютерна інженерія» затверджене кафедрою Обчислювальної техніки Національного технічного університету України «Київський Політехнічний інститут імені Ігоря Сікорського».

3. МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою даної роботи є створення вітчизняного веб-застосунку для обміну інформацією з використанням технологій шифрування та конфіденційності даних. Призначенням розробки є створення альтернативного безпечного рішення для обміну інформацією між користувачами у разі непередбаченої поведінки або витоків даних при використанні іноземних аналогів

4. ДЖЕРЕЛА РОЗРОБКИ

Джерелами розробки даного дипломного проєкту були офіційні джерела та ресурси з документацією для технологічних рішень використаних при розробці, а саме: документація мови Java, фреймворків Spring та ReactJS та ін., а також публікації та статті у мережі Інтернет, що стосуються обраних технологій, науково-технічна література.

5. ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до розробленого продукту

- Простий та інтуїтивно зрозумілий користувацький інтерфейс.
- Зручний та надійний механізм реєстрації та авторизації користувачів.
- Надати користувачам можливість пошуку інших користувачів у системі з реалізованим механізмом зручної фільтрації для пришвидшення пошуку.
- Надати можливість користувачам підписуватись на інших користувачів.
- Надати можливість користувачам створювати пости.
- Надати користувачам можливість створення групових та приватних чатів для спілкування в системі.
- Кожен чат повинен мати систему учасників побудовану на ієрархії, що включає в себе дві ролі: роль звичайного учасника групи та роль адміністратора групи з розширеним набором прав в групі.
- Кожен учасник чату повинен мати можливості для видалення та редагування раніше відправлених повідомлень, а також можливість для залишення чату.
- Кожен адміністратор чату повинен мати всі права звичайних користувачів, а також можливість видаляти небажаних користувачів групи та їх повідомлення, а також право на видалення всієї групи.
- Кожне повідомлення чату, перед відправкою на сервер для збереження, повинне шифруватись використовуючи технології асиметричного та симетричного шифрувань.
- Шифрування повинне бути реалізовано на основі обміну між сервером та користувачем публічних та приватних ключів у момент входу користувача в систему.
- Повідомлення користувачів повинні зберігатись в зашифрованому вигляді в базі даних.

- В момент передачі між клієнтською та серверною частиною повідомлення не може бути розшифроване ніким іншим окрім власне отримувача повідомлення або ж самим сервером.
- Система веб-застосунку повинна мати користувачів з правами адміністратора для управління нею.
- Користувачі з правами адміністратора повинні мати можливість блокувати або видаляти користувачів, що порушують правила платформи, а також функцію для створення нових адміністраторів для допомоги в управлінні системою.
- Система повинна мати головного адміністратора, що має додаткову функцію видалення звичайних адміністраторів.

5.2. Вимоги до програмного забезпечення

Вимоги для клієнтської частини:

- ОС Windows 10, Mac чи Linux.
- Будь-який з сучасних веб-браузерів (Google Chrome, Microsoft Edge, Mozilla Firefox, Opera Browser та ін.).

Вимоги для серверної частини:

- Java версії 17 або вище.
- База даних PostgreSQL.

5.3. Вимоги до апаратної частини

- Рекомендовано 8 ГБ та більше RAM.
- ЦП Intel Core (5+ покоління) або AMD з двома та більше ядрами.
- Зовнішня пам’ять 64 ГБ.
- Модуль WIFI або роз’єм RJ45 для підключення кабелю для забезпечення доступу до мережі інтернет.

6. ЕТАПИ РОЗРОБКИ

Назва етапів виконання	Термін виконання
Затвердження теми роботи	15.03.2024 - 17.03.2024
Вивчення та аналіз завдання	17.03.2024 - 25.03.2024
Розробка архітектури та загальної структури системи	25.03.2024 - 05.04.2024
Розробка структур окремих підсистем	05.04.2024 - 15.04.2024
Програмна реалізація системи	15.04.2024 - 10.05.2024
Оформлення пояснювальної записки	10.05.2024 - 30.05.2024

ПОЯСНЮВАЛЬНА ЗАПИСКА
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «*Веб-застосунок для спілкування з використанням технологій шифрування та конфіденційності даних*»

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА ТЕРМІНІВ 4
ВСТУП 6
РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ 8
1.1 Загальне поняття та основна ідея застосунку для спілкування 8
1.2 Популярність та актуальність розробки 9
1.3 Особливості роботи месенджерів 12
1.4 Популярні засоби захисту інформації в месенджерах 14
1.4.1 Симетричне шифрування 15
1.4.2 Асиметричне шифрування 15
1.4.3 Шифрування “в транзиті” 16
1.4.4 Наскрізне шифрування (End-to-end encryption, E2EE) 17
1.5 Огляд існуючих аналогів програмного продукту та їх особливості 18
1.5.1 WhatsApp 18
1.5.2 Telegram 19
1.5.3 Signal 20
1.5.4 Session 21
1.5.5 Facebook Messenger 21
ВИСНОВОК ДО РОЗДІЛУ 1 23
РОЗДІЛ 2. ОГЛЯД ТА ВИБІР ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ 24
2.1 Архітектура веб-застосунку 24
2.2 Технології клієнтської частини 27
2.2.1 Вибір мови програмування та засобів розмітки сторінки 27
2.2.2 Вибір фреймворку 29
2.3 Технології серверної частини 31

2.3.1 Вибір мови програмування та платформи.....31
2.3.2 Огляд фреймворку.....33
2.4 База даних.....40
ВИСНОВОК ДО РОЗДІЛУ 2.....41
РОЗДІЛ 3. ПРОЕКТУВАННЯ ТА РОЗРОБКА СИСТЕМИ.....42
3.1 Розробка серверної частини.....42
3.1.1 Налаштування бази даних.....44
3.1.2 Проектування бази даних.....46
3.1.3 Підключення бази даних до проекту.....50
3.1.4 Налаштування Spring Security та прав доступу.....50
3.1.5 Налаштування механізму для обміну повідомленнями.....57
3.1.6 Створення інструментів для шифрування даних.....59
3.2 Розробка клієнтської частини.....62
3.2.1 Налаштування маршрутизації.....63
3.2.2 Налаштування функціоналу для комунікації з серверною частиною.....65
3.2.3 Створення функціоналу для обміну повідомленнями.....66
ВИСНОВОК ДО РОЗДІЛУ 3.....68
РОЗДІЛ 4. ТЕСТУВАННЯ РОБОТИ ЗАСТОСУНКУ.....69
4.1 Огляд функціоналу.....69
4.1.1 Реєстрація та автентифікація.....71
4.1.2 Сторінка користувача.....73
4.1.3 Обмін повідомленнями в застосунку.....76
4.1.4 Панель адміністратора.....80
4.2 Розвиток платформи в майбутньому.....82
4.3 Використання застосунку у військовій сфері.....83
ВИСНОВОК ДО РОЗДІЛУ 4.....85
ВИСНОВКИ.....86
СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ.....87
ДОДАТОК 1.....1

ДОДАТОК 2..... 1
ДОДАТОК 3..... 1
ДОДАТОК 4..... 1

ПЕРЕЛІК СКОРОЧЕНЬ ТА ТЕРМІНІВ

API	(Application Programming Interface) Прикладний програмний інтерфейс.
HTTP	(HyperText Transfer Protocol) Протокол передачі гіпертекстових документів.
TCP	(Transmission Control Protocol) Протокол управління передачею даних
URL	(Uniform Resource Locator) Уніфікований локатор ресурсів або адреса ресурсу.
URI	(Unified Resource Identifier) Уніфікований ідентифікатор ресурсів.
WebSocket	Протокол, що призначений для обміну інформацією між браузером та вебсервером в режимі реального часу.
БД	База даних.
СУБД	Система управління базами даних.
ORM	(Object Relational Mapping) Об’єктно-реляційне відображення.
JWT	(JSON Web Token) Стандарт токена доступу на основі JSON.
E2EE	(End-to-end-encryption) Наскрізне шифрування
OOP	(Object-oriented programming) Об’єктно-орієнтоване програмування.
IoC	(Inversion of Control) Інверсія керування
DI	(Dependency Injection) Впровадження залежностей
Hibernate	Засіб відображення між об’єктами та реляційними структурами для платформи Java.
JDBC	(Java DataBase Connectivity) Прикладний програмний інтерфейс Java для з’єднання з базами даних на Java.

CRUD	(Create, Read, Update, Delete) - Основні функції управління даними «створення, читання, оновлення, видалення».
SQL	(Structured Query Language) Мова запитів для взаємодії користувача з базами даних.
JSON	(JavaScript Object Notation) Текстовий формат обміну даними між комп’ютерами.
HTML	(HyperText Markup Language) Стандартизована мова розмітки гіпертексту для перегляду вебсторінок у браузері.
CSS	(Cascading Style Sheets) Каскадні таблиці стилів.
JS	(JavaScript) Динамічна об’єктно-орієнтована мова програмування.
REST	(Representational State Transfer) Підхід до побудови архітектури мережевих протоколів, які надають доступ до інформаційних ресурсів.
SOAP	(Simple Object Access Protocol) протокол обміну структурованими повідомленнями в розподілених обчислювальних системах, що базується на форматі XML.
ACID	(Atomicity, Consistency, Isolation, Durability) Набір властивостей, що гарантують надійну роботу транзакцій бази даних: атомарність, узгодженість, ізольованість, довговічність.
STOMP	(Simple Text Oriented Messaging Protocol) Протокол, заснований на фреймах, змодельованих на основі HTTP.

ВСТУП

Спілкування та обмін інформацією через Інтернет у сучасному світі стали невід'ємними частинами повсякденного життя будь-якої людини. Саме тому питання безпеки даних та захисту конфіденційної інформації стає все більш актуальним. Особливо гостро ця проблема постає у сфері онлайн комунікацій, де ризики витоку інформації, кібершахрайства, злому акаунтів користувачів та компроментації особистої інформації є одними з найвищих. У зв'язку з цим зростає потреба в створенні надійних та безпечних каналів зв'язку, які гарантують захист даних в мережі "Інтернет". Саме наявність таких каналів є визначальним фактором при виборі відповідного ресурсу в мережі.

Одним з рішень проблеми витоку даних в засобах призначених для комунікації таких як месенджери є підтримка застосунком технологій шифрування даних. Однією за найпопулярніших технологій в цій сфері є наскрізне шифрування (End-to-end encryption). Такий спосіб наразі використовують більшість популярних месенджерів. Перевагою такого типу шифрування є те, що лише відправник та одержувач повідомлення можуть його прочитати. Навіть якщо потенційний зловмисник перехопить повідомлення в процесі надсилання, він не зможе його розшифрувати, оскільки для цього потрібен спеціальний ключ, який є лише у одержувача. Крім того, варто зазначити, гарним тоном вважається зашифровувати інформацію не лише при передачі її через мережу, а й при збереженні її на сервері. Така процедура додатково убезпечує дані на сервері від несанкціонованого доступу ззовні.

Для створення безпечного спілкування в мережі "Інтернет" в даній роботі було обрано реалізувати спілкування у формі веб-застосунку з асиметричним шифруванням повідомлень "в транзиті", що частково схоже на наскрізне шифрування. Такий тип застосунку має декілька переваг в

порівнянні з мобільними та десктопними аналогами. По-перше, це доступність. До веб-застосунків можна отримати доступ з будь-якого пристрою, який має підключення до мережі та будь-який встановлений браузер. Також варто відзначити простоту у використанні. Такі застосунки, зазвичай простіші оскільки не вимагають встановлення додаткового програмного забезпечення, що у свою чергу допомагає економити місце на пристрої користувача. Ще однією перевагою є економічна вигідність, адже не потрібно окремо створювати різні варіанти продукту для різних операційних систем. З урахуванням всіх перерахованих факторів, створення програмного продукту на веб-платформі є найвигіднішим.

РОЗДІЛ 1.
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Загальне поняття та основна ідея застосунку для спілкування

Потреба в спілкуванні є однією з базових потреб в людському житті та суспільстві загалом. Спілкування лежить в основі формування стосунків, обміну інформацією, навчання та розвитку. Воно дає нам відчути себе з'єднаними з іншими, допомагає співпрацювати над спільними цілями та вирішувати проблеми.

З розвитком глобальної мережі "Інтернет" ця потреба лише зросла. Інтернет відкрив нові можливості для обміну інформацією та радикально змінив спосіб, яким ми спілкуємось один з одним. Завдяки появі різноманітних платформ, таких як соціальні мережі, месенджери, електронна пошта тощо, комунікувати стало зручніше, доступніше та швидше, а далекі відстані більше не є перепоною для вирішення різноманітних питань. Кожен з цих новітніх сервісів має свої особливості, переваги та недоліки, проте загалом їх всіх можна назвати застосунками для спілкування.

Отже, застосунок для спілкування - це програмне забезпечення, яке дозволяє користувачам обмінюватись повідомленнями різного типу. Це можуть бути текстові повідомлення, мультимедійні файли або ж файли інших видів. Більшість сучасних застосунків беруть за основу саме обмін текстовою інформацією та надалі розширюють функціонал для можливості здійснювати голосові дзвінки або відеодзвінки. В даній роботі також було обрано напрямок реалізації повідомлень саме текстового формату. Варто зазначити, що платформою для реалізації подібного виду застосунків може бути будь-яка з популярних операційних систем, але універсальним рішенням вважається реалізація у вигляді веб-застосунку, адже це дозволяє відкривати сервіс на

будь-якому якому пристрої без прив’язки до конкретної платформи.

1.2 Популярність та актуальність розробки

Одним з найпопулярніших рішень для реалізації веб-застосунків для спілкування є месенджери. Їх основною ідеєю є створення ефективного та надійного способу для взаємодії між користувачами у режимі реального часу незалежно від географічного положення користувачів.

Згідно статистичних даних інтернет-ресурсу businessofapps.com [1], серед 10 найбільш популярних додатків 2023 року по всьому світу, як мінімум 5 додатків є месенджерами або ж такими, що містять функціонал обміну повідомленнями в реальному часі. Це відповідно Instagram, WhatsApp, Telegram. SnapChat, WhatsApp Business.

Таблиця 1.1 - Топ 10 найпопулярніших застосунків згідно даних ресурсу Business Of Apps [1].

App	Downloads, m
Instagram	696
TikTok	654
Facebook	553
WhatsApp	475
CapCut	389
Telegram	355
Snapchat	343
Temu	274
WhatsApp Business	267
Spotify	248

Окрім того, згідно ресурсу statista.com [2] лише за березень 2024 року у всьому світі було здійснено 58 млн. скачувань застосунку Instagram, 42 млн. - WhatsApp та 27 млн. - Telegram.

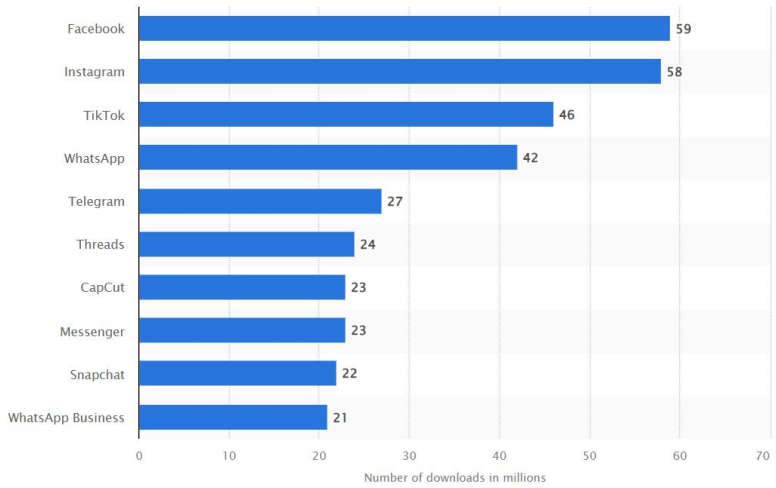


Рисунок 1.1 - Застосунки з найбільшою кількістю скачувань за березень 2023 року згідно даних ресурсу Statista [2].

Щодо України, то ситуація дуже схожа з світовою. Відповідна статистика за 2023 рік наведена в таблиці нижче.

Таблиця 1.2 - Найпопулярніші застосунки в Україні в 2023 році згідно даних ресурсу web-promo.ua [3].

№	App	Downloads, m
1	TikTok	5.49
2	Telegram	4.71
3	Instagram	3.00
4	Viber	2.63
5	Nova Poshta	2.49
6	Royal Match	2.45
7	WhatsApp Messenger	2.40
8	Prom.ua	2.38
9	Privat24	2.37
10	Дія	2.23

Враховуючи наведену статистику можна зрозуміти, що популярність застосунків для спілкування лише зростає. При цьому, незважаючи на різноманіття існуючих додатків, постійно з’являються нові програми, що повторюють функціонал свої попередників, але при цьому пропонують щось своє.

Крім того популярність застосунку може залежати від країни. Так, WhatsApp найбільш поширений в країнах Південної Америки, Близького сходу, Європи, а також в Австралії, в той час як Facebook Messenger має популярність в США та Канаді. Також варто зазначити, що деякі люди бояться користуватись платформами інших країн, адже думають, що влада країни може збирати особисті дані користувачів у своїх інтересах. Саме тому більшість віддає перевагу розробкам власних держав. Гарним прикладом є Китай, де найпопулярнішим месенджером є застосунок WeChat розроблений китайською телекомунікаційною компанією Tencent.

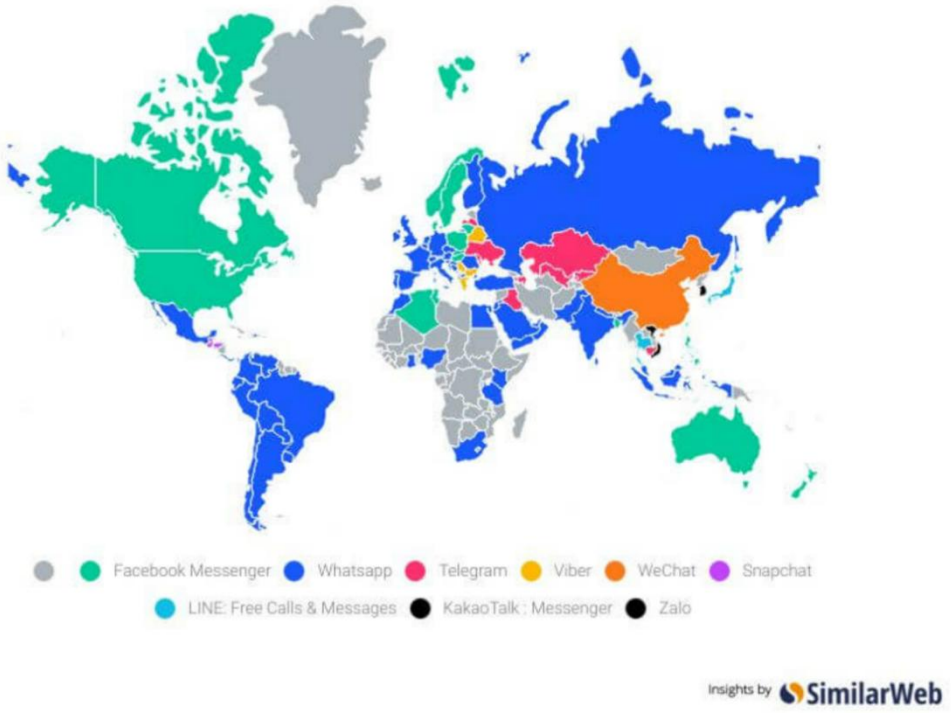


Рисунок 1.2 - Найпопулярніші застосунки з розподілом по країнам станом на 2022 рік згідно даних веб-ресурсу similarweb.com [4].

На жаль, Україна не має популярних та безпечних вітчизняних аналогів світовим застосункам для спілкування. Саме тому, а також враховуючи наведені вище факти та статистику, дана дипломна робота має на меті створення такої платформи.

1.3 Особливості роботи месенджерів

Принцип роботи месенджера полягає у встановленні зв’язку між двома або більше пристроями. Фундаментом для цього слугує клієнт-серверна архітектура. Такий тип архітектури полягає в розподілі обов'язків між декількома компонентами: клієнтом та сервером.

- Клієнтський компонент - це програмне забезпечення або додаток, який користувачі встановлюють на свої пристрої. Цей компонент відповідає за відображення інтерфейсу користувача, обробку та відправку повідомлень та інших даних.
- Серверний компонент - це програмне забезпечення, яке розміщується на веб-сервері і забезпечує збереження та обробку даних та відправлення повідомлень. Коли користувач відкриває месенджер на своєму пристрої, клієнтська програма встановлює зв’язок з сервером, що дозволяє обмінюватися повідомленнями. Після цього клієнт може надсилати запити до сервера, який в подальшому обробляє ці запити, за потреби зберігає дані та надсилає відповіді з інформацією, необхідною клієнту.

В месенджерах такий обмін повідомленнями між сервером та користувачами зазвичай досягається за допомогою системи обміну миттєвими повідомленнями.

Технологія обміну миттєвими повідомленнями - це служба для обміну текстовими повідомленнями між комп'ютерами або іншими пристроями користувачів через комп'ютерні мережі. Повідомлення зазвичай передаються між двома або більше сторонами, коли кожен користувач вводить текст і запускає передачу одержувачу, який підключений до спільної мережі. Цей спосіб відрізняється від електронної пошти тим, що розмови через миттєві повідомлення відбуваються в режимі реального часу. Для цього можуть використовуватися різні технології, такі як веб-сокети, push сповіщення, pull технологія та ін. Розглянемо детальніше кожну з представлених технологій.

- WebSocket [5, 6] — це двонаправлений повнодуплексний протокол зв’язку між клієнтом та сервером, що використовує TCP з’єднання та призначений для обміну інформацією в режимі реального часу. На відміну від HTTP-протоколу, який працює за принципом «запит — відповідь», у вебсокетах обмін повідомленнями проходить через єдиний канал зв’язку по якому і сервер, і клієнт можуть надсилати один одному повідомлення. При цьому кожна сторона комунікації здатна одночасно отримувати та відправляти дані. Щоб встановити WebSocket-з'єднання, клієнт надсилає handshake-запит — запит на встановлення довіри. Після цього канал зв’язку залишається відкритим протягом усієї комунікації і за необхідності будь-яка зі сторін може його закрити.

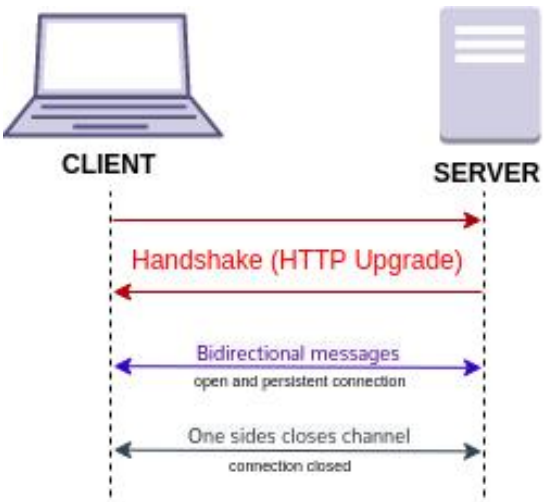


Рисунок 1.3 - Діаграма, що описує
принцип роботи протоколу веб-сокетів.

Також варто зазначити, що специфікація визначає дві URI схеми для вебсокетів: ws:// для нешифрованого з’єднання та wss:// для шифрованого.

- Технологія pull [7] — це інтернет-протокол, у якому запит на отримання даних надходить від клієнта, а потім сервер відповідає на нього. Pull-запити даних складають основу мережевих обчислень, де багато клієнтів вимагають дані з централізованих серверів. Pull широко використовується в Інтернеті для запиту контексту сторінки HTTP з вебсайту.

- Технологія push [8] - це Інтернет-протокол, при якому ініціатором події є видавець або центральний сервер. На відміну від pull технології, де запит на отримання даних надходить від клієнта, при push технології клієнт підписується на різні інформаційні канали, що пропонуються сервером. Як тільки на одному з цих каналів з'являється новий вміст, сервер автоматично надсилає його клієнту, забезпечуючи своєчасну доставку актуальної інформації без необхідності запиту з боку клієнта.

Іншою, не менш важливою, особливістю месенджера є наявність механізму реєстрації та аутентифікації. Щоб використовувати месенджер, користувачам зазвичай потрібно зареєструвати обліковий запис. Під час реєстрації користувачі надають ідентифікаційну інформацію та створюють ім’я користувача та пароль, які надалі будуть потрібні для входу в акаунт. Коли користувач входить у свій обліковий запис у месенджері, клієнтський компонент відправляє запит на аутентифікацію до серверного компонента. Сервер перевіряє дані аутентифікації, щоб переконатися, що користувач має право доступу до облікового запису.

Окрім основного спектру задач описаного вище, багато месенджерів також надають користувачам доступ до низки інших функцій, таких як створення групових чатів та інформаційних каналів. Деякі надають можливість текстового розпізнавання голосових повідомлень та проведення відеоконференцій. Крім того месенджери можуть бути оснащені функціоналом для взаємодії з ботами - автоматизованими програмами, які здатні виконувати певні завдання за запитом користувача.

1.4 Популярні засоби захисту інформації в месенджерах

Попри можливості, що принесла з собою поява Інтернету, з’явились і нові ризики. Тепер інформація, що передавалась через мережу могла бути легко перехоплена зловмисниками, а тому потрібні були новітні рішення, що дозволили б убезпечити користувацькі дані.

На допомогу прийшли різні способи шифрування даних [9]. Розглянемо детальніше деякі з основних підходів для цього.

1.4.1 Симетричне шифрування

Шифрування з симетричними ключами [10] — схема шифрування, у якій ключ шифрування та ключ дешифрування збігаються або один легко обчислюється з іншого та навпаки.

Переваги симетричного шифрування даних

- Забезпечує швидке та ефективне шифрування, так як використовується тільки один ключ.
- Ідеально підходить для шифрування великої кількості даних.
- Забезпечує високу продуктивність і вимагає менше обчислювальної потужності.

Недоліки симетричного шифрування даних

- Для шифрування даних зазвичай використовується невелика довжина ключа (128-256 біт).
- З’являються ризики у зв’язку з необхідністю безпечної передачі ключа одержувачу повідомлення.

Поширені алгоритми, що використовуються для симетричного шифрування: RC4, AES, DES, 3DES.

1.4.2 Асиметричне шифрування

Асиметричне шифрування - набір методів криптографічного шифрування, в яких використовують два ключі: відкритий (публічний), що використовується для шифрування інформації та таємний (приватний) - для розшифрування. Таке шифрування ще називають шифруванням з відкритим ключем. Варто зазначити, що жоден із ключів не може бути обчислений з іншого. При цьому розшифровування даних за допомогою відкритого ключа

також неможливе. При такому способі відкритий ключ можна публічно розповсюджувати, а закритий ключ власник повинен тримати в секреті [11, 12].

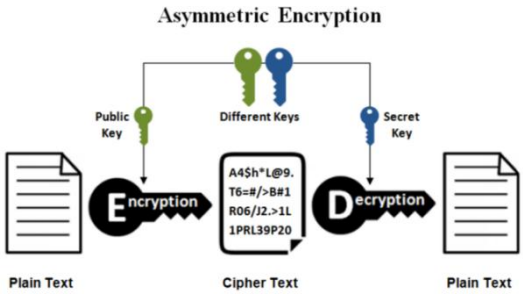


Рисунок 1.4 - Схема симетричного алгоритму для шифрування даних.

Переваги симетричного шифрування даних

- Безпечніший за симетричне шифрування, оскільки використовується пара ключів.
- Метод симетричного шифрування більш стійкий до перехоплення.
- У зв’язку з тим, що використовується пара ключів [13] – процес складний.
- Зазвичай використовуються довгі ключі шифрування (1024-4096 біт).

Недоліки симетричного шифрування даних

- Асиметричні системи шифрування повільніші, порівняно з симетричними.
- Вимагає більшої обчислювальної потужності на відміну від симетричного шифрування.
- Використовується при шифруванні невеликого об’єму даних.

Поширені алгоритми, що використовуються для симетричного шифрування: RSA, Diffie-Hellman, El Gamal, DSA.

1.4.3 Шифрування “в транзиті”

Шифрування в транзиті - це метод шифрування даних, який застосовується для захисту конфіденційності під час їх передачі через мережу. У багатьох системах обміну повідомленнями, зокрема в

електронній пошті та месенджерах, повідомлення проходять через сервер і потім доставляються отримувачеві. Шифрування "в транзиті" означає, що повідомлення зашифровані лише під час передання до сервера та з сервера до користувача.

1.4.4 Наскрізне шифрування (End-to-end encryption, E2EE)

Наскрізне шифрування [14, 15, 16] – це спосіб шифрування даних, за якого тільки кінцеві користувачі, які беруть участь в обміні даними, мають доступ до розшифрованого контенту.

Станом на зараз, наскрізне шифрування вважається одним з найбезпечніших, оскільки практично робить неможливим для сторонніх осіб можливість отримати доступ до даних користувача у незашифрованому вигляді.

Недоліки наскрізного шифрування даних

Такий спосіб убезпечення інформації може сповільнити передачу даних, що може бути неприйнятним для деяких додатків. Також можуть виникнути проблеми у створенні умов для безпеки ключів шифрування, щоб вони не потрапили до рук зловмисників.

Крім того, одним з головних недоліків наскрізного шифрування є те, що при втраті ключа шифрування доступ до даних може бути втрачено назавжди. Це означає, що якщо користувач забуде або втратить ключ, він не зможе розшифрувати свої дані, і навіть провайдер мережі не зможе допомогти відновити доступ до цих даних.

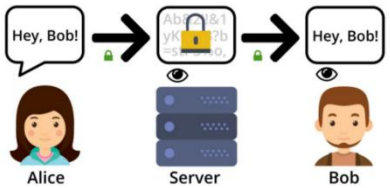


Рисунок 1.5 - Наочне представлення роботи наскрізного шифрування.

Проаналізувавши всі переваги та недоліки різних типів шифрування даних, в даній роботі було обрано спосіб асиметричного шифрування “в транзиті” в поєднанні з симетричним шифруванням при якому сервер та клієнт окремо один від одного генерують власні пари публічних та приватних ключів. Надалі, в момент аутентифікації користувача на платформі, відбувається обмін публічними ключами відповідно між сервером та кінцевим користувачем. Після цього, за допомогою асиметричних ключів, відбувається обмін симетричним ключем, що буде використовуватись для шифрування повідомлень в чатах перед їх відправкою. Для збереження повідомлень в базі даних також було обрано використовувати симетричне шифрування. В даному випадку, оскільки ключ для симетричного шифрування не потрібно передавати через мережу адже він зберігається лише на сервері, то відповідно і ризиків перехоплення ключа зловмисником також не має.

1.5 Огляд існуючих аналогів програмного продукту та їх особливості

Станом на сьогодні, існує досить багато всесвітньо відомих реалізацій застосунків для спілкування. Кожен з них має свої особливості, а також певні переваги та недоліки. Розглянемо детальніше деякі з таких рішень.

1.5.1 WhatsApp

WhatsApp є одним із найпоширеніших додатків для обміну повідомленнями, відомий своєю простотою у використанні та функціями, такими як обмін місцезнаходженням та файлами. Для створення безпеки він використовує потужний протокол шифрування, розроблений для Signal компанією Open Whisper Systems, який вважається галузевим стандартом. Також присутня функція шифрування Perfect Forward Secrecy. Це означає, що якщо комусь вдасться викрасти ключ розшифровки розмови, вони зможуть

побачити лише останнє повідомлення, яке було надіслано.

Рисунок 1.6 - Логотип WhatsApp.

Переваги WhatsApp

- Широка популярність по всьому світу.
- Наскрізне шифрування увімкнене за замовчуванням.
- Підтримує можливість телефонних та відео-дзвінків.
- Використовує протокол шифрування від Signal.
- Існує мобільна та десктопна версія застосунку.

Недоліки WhatsApp

- Для використання потрібен доступ до номера телефону користувача та його контактів.

1.5.2 Telegram

Telegram працює подібно до WhatsApp. Він дозволяє надсилати повідомлення та телефонувати іншим користувачам, створювати групові чати та надсилати файли. Telegram має зручний, інтуїтивно зрозумілий інтерфейс, який особливо чудовий для групового обміну повідомленнями. На жаль, протокол шифрування Telegram має недоліки, адже його розробила власна команда з невеликим досвідом у криптографії. Сервери Telegram не є відкритими, тому код не перевірявся третіми сторонами.

Рисунок 1.7 - Логотип Telegram.

Переваги Telegram

- Має функціонал для створення груп та каналів.
- Підтримує наскрізне шифрування.

- Застосунок доступний для всіх популярних операційних систем.
- Існує веб-версія застосунку.
- Дозволяє приховувати номер телефону.

Недоліки Telegram

- Наскрізне шифрування не ввімкнено за замовченням.
- Для реєстрації потрібен номер телефону.

1.5.3 Signal

Signal вважається однією з найбезпечніших програм для обміну повідомленнями на ринку. Він використовує наскрізне шифрування з протоколом власної розробки для телефонних викликів та для обміну повідомленнями. Його протокол шифрування настільки надійний, що його також використовують інші провідні програми, такі як WhatsApp і Facebook Messenger.

Рисунок 1.8 - Логотип Signal.

Переваги Signal

- Використовує наскрізне шифрування захищене open source технологією.
- Не зберігає метадані користувачів.
- Використовує потужний протокол власної розробки.

Недоліки Signal

- Потребує доступ до номера телефону.
- Має невеликий функціонал в порівнянні з іншими месенджерами на ринку.

1.5.4 Session

Session - це програма обміну миттєвими повідомленнями з наскрізним шифруванням, яка наголошує на конфіденційності та анонімності

користувача. На від Signal та інших месенджерів, для реєстрації не потрібен номертелефону або адреса електронної пошти. Натомість під час створення облікового запису генерується випадковий унікальний ID. Іншою важливою особливістю є те, що Session приховує IP-адреси користувачів тим самим забезпечуючи анонімність користувачів.

Рисунок 1.9 - Логотип Session.

Переваги Session

- Використовує наскрізне шифрування за замовчуванням.
- Не збирає метадані користувачів.
- Не потребує номера телефону користувача.
- Приховує IP-адресу користувача.

Недоліки Session

- Не має великої популярності.
- Має невеликий функціонал в порівнянні з іншими месенджерами на ринку.

1.5.5 Facebook Messenger

Facebook Messenger є популярним сервісом для безпечного обміну повідомленнями, який належить компанії Meta. Він дозволяє користувачам обмінюватися текстовими, фото-, відео- та аудіоповідомленнями, а також створювати групові чати. Застосунок є основним способом для спілкування з друзями у Facebook та дає можливість зв'язку з телефонними контактами.

Рисунок 1.10 - Логотип Facebook Messenger.

Переваги Facebook Messenger

- Простий у використанні інтерфейс.
- Використовується як ефективний додаток до застосунку Facebook для обміну повідомленнями.
- Підтримує наскрізне шифрування.
- Можна використовувати, навіть після деактивації облікового запису Facebook

Недоліки Facebook Messenger

- Наскрізне шифрування відключено за замовчуванням.
- Збирає метадані користувачів.
- Для реєстрації потрібен акаунт Facebook.
- Facebook був пов'язаний з декількома скандалами щодо приватності даних, що породжує питання щодо безпеки користувачів Messenger.

ВИСНОВОК ДО РОЗДІЛУ 1

У даному розділі дипломної роботи було здійснено аналіз технологій необхідних для побудови ефективних та надійних застосунків для спілкування. Як результат, проведене дослідження дозволило розставити пріоритети під час розробки власної системи.

По-перше, у ході роботи було визначено основне поняття та ідею застосунку для спілкування, а також його актуальність. Відповідно до знайденої статистичної інформації, можна констатувати той факт, що потреба в таких застосунках лише продовжує зростати. В той же час збільшується необхідність саме в захищених застосунках, що убезпечують користувацькі дані та запобігають від перехоплення їх зловмисниками. Саме тому далі був проведений огляд існуючих способів шифрування даних. Дослідження переваг та недоліків кожного з цих видів шифрування, дозволило обрати оптимальне рішення для власної системи яка б одночасно забезпечувала захищеність даних, а також підтримувала швидкодію на високому рівні. Результатом вибору стало поєднання асиметричного шифрування “в транзиті” з симетричним шифрування для передавання повідомлень між сервером та клієнтом, та симетричне шифрування для зберігання інформації в базі даних.

Наприкінці розділу було оглянуто аналогічні застосунки та їх особливості. Як виявилось, кожен з таких додатків має свої переваги та недоліки. Крім того, досить глобально вирізняється проблема збирання особистих даних користувачів на деяких платформах. Враховуючи наведені факти можна заявити що, розробка власної захищеної системи для спілкування є надзвичайно важливою задля протидії стороннім особам та організаціям отримувати особисту інформацію користувачів та для створення захищеного простору для спілкування.

РОЗДІЛ 2.
ОГЛЯД ТА ВИБІР ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ

2.1 Архітектура веб-застосунку

Розробка та проектування гарної архітектури є невід'ємної частиною будь-якого продукту. Архітектура веб-додатку описує взаємозв'язок між його компонентами, структуру та спосіб взаємодії. Вона визначає, як розподіляються функції та обов'язки між різними частинами додатку, щоб забезпечити його ефективну роботу та масштабованість.

Одним з найпоширеніших підходів до архітектури веб-додатків є клієнт-серверна модель. Зазвичай архітектура такого типу складається з трьох основних частин: клієнтської та серверної частин і бази даних.

- Клієнтська частина представляє програмне забезпечення або інтерфейс, що запускається на пристрої користувача, на приклад у вигляді браузера. Вона включає HTML, CSS та JavaScript, які використовуються для створення інтерактивних елементів та забезпечують гарний користувацький досвід.
- Серверна частина - це сервер, який обробляє запити клієнта, виконує бізнес-логіку, зберігає дані та повертає відповіді клієнту. Для цього можуть використовуватись різні мови програмування в поєднанні з відповідними фреймворками для пришвидшення розробки.
- База даних (Сервер бази даних) - це спосіб зберігання інформації у вигляді організованої структури даних, які пов'язані між собою спільною ознакою або властивістю.

Окрім усталеного спілкування між front-end та back-end частинами додатку у вигляді "запит - відповідь", поширеним також є підхід до створення асинхронних задач, що дозволяють клієнту не чекати відповіді від сервера тим самим пришвидшуючи роботу застосунку. Одним з найпоширеніших випадків

такого типу задач є відправка електронних листів. Замість того, щоб негайновідправляти листи під час обробки запиту, додаток може обробити ці завдання асинхронно. Це дозволяє звільнити основний потік виконання додатку від тривалих операцій і прискорити відповідь на запити користувачів.

Іншим прикладом для виконання асинхронного типу задач є використання планувальників задач, наприклад Cron. З використанням Cron з'являється можливість запускати операції відповідно до графіку або з певним часовим інтервалом. Такі задачі виконуються в окремому фоновому процесі, окремо від основного потоку виконання, що також пришвидшує роботу застосунку.

Також варто зазначити, що для комунікації між сервером та клієнтом можуть використовуватись декілька протоколів. Найпоширенішим є протокол HTTP або ж його розширення HTTPS, що підтримує шифрування даних задля підвищення безпеки. Крім того іншим поширеним протоколом, що досить часто використовується для миттєвого обміну повідомленнями є протокол веб-сокетів. Він забезпечує двонаправлений повнодуплексний канал зв'язку з використанням TCP протоколу.

Таким чином, для розробки цього веб-застосунку буде використана архітектура описана вище з застосуванням асинхронних задач. Для реалізації цього буде створено два окремих проекта, відповідно для front-end та back-end частин та налаштовано сервер для бази даних. Створення та конфігурацію бази даних для розробки було вирішено реалізувати з використанням технології контейнеризації на базі застосунку Docker.

Docker [17] - це платформа для розробки, доставки та запуску програмного забезпечення в ізольованих середовищах, відомих як контейнери. Контейнери використовують технологію віртуалізації на рівні операційної системи, яка дозволяє запускати програми та їх залежності у стандартизованому середовищі, відокремленому від основної операційної системи. Основне призначення Docker полягає в тому, щоб спростити процес

розробки, доставки та експлуатації програмного забезпечення.

Рисунок 2.1 - Логотип Docker.

Створення контейнерів в Docker базується на технології образів.

Образ Docker (Docker image) — містить операційну систему, застосунок і всі його залежності. Образи в Docker складаються з шарів. Якщо потрібно створити образ з вебсервером, то за основу можна взяти дистрибутив операційної системи, додати залежність - вебсервер, і записати це як новий образ, який матиме два шари — один з ОС, наступний з вебсервером.

Більшість таких образів можна знайти на DockerHub - загальнодоступному реєстрі образів. Крім того, багато великих постачальників баз даних розміщують офіційні образи для власних продуктів саме на DockerHub звідки їх можна безкоштовно завантажити собі локально на комп'ютер.

Враховуючи специфіку дипломного проекту, в процесі спілкування між серверним та клієнтським кодом будуть застосовуватись два протоколи під різні потреби. Для базових задач, таких як реєстрація користувача буде використовуватись HTTP протокол. В той же час для задач, що потребують миттєвого обміну інформацією, таких як спілкування в чатах, або ж сповіщення користувачів про певні події потрібно буде використовувати протокол веб-сокетів.

Рисунок 2.2 - Клієнт-серверна архітектура.

Щодо формату передачі даних то існує два головних підходи: REST та SOAP. Головна відмінність полягає в тому, що в REST для передачі інформації використовується формат даних JSON, а в SOAP - XML.

JSON (JavaScript Object Notation) [18] — це текстовий формат обміну даними між комп'ютерами. JSON базується на тексті, що може бути прочитаним людиною. Формат дає змогу описувати об'єкти та інші структури даних і зазвичай використовується для передавання структурованої інформації через мережу завдяки процесу серіалізації.

XML (EXtensible Markup Language) [19] — стандарт запропонований для побудови мов розмітки ієрархічно структурованих даних для обміну між різними застосунками, зокрема, через Інтернет. XML-документ складається із текстових знаків, і придатний до читання людиною.

Враховуючи, що на сьогодні JSON формат вважається стандартом при обміні інформацією, а також є простішим для читання людиною та компактнішим в порівнянні з форматом XML, то використовувати будемо саме його.

2.2 Технології клієнтської частини

Вибір технологій для клієнтської частини є важливою складовою розробки адже це безпосередньо впливає на досвід користувача. Саме з клієнтською частиною користувач взаємодіє найбільше. Невірно зроблений вибір може повпливати на подальші відносини користувача з застосунком. Як правило, продуманий та гарно зроблений інтерфейс є одним з ключових аспектів, що змушує користувача залишитися на обраному інтернет-ресурсі.

2.2.1 Вибір мови програмування та засобів розмітки сторінки

Серед мов програмування клієнтської частини загалом можна виділити два основних варіанти: JavaScript та TypeScript.

- JavaScript (JS) - це динамічна, високорівнева мова програмування, яка використовується в основному для створення інтерактивних та динамічних веб-сторінок. Вона була розроблена Бренданом Айком у 1995 році під час його роботи в компанії Netscape Communications.
- TypeScript - це мова програмування, яка є надмножиною JavaScript, розроблена компанією Microsoft і вперше випущена у 2012 році. Основна мета TypeScript — розширити можливості JavaScript, додаючи статичну типізацію та об'єктно-орієнтовані концепції, що робить її більш підходящою для розробки великих і складних додатків.

Останнім часом саме TypeScript здобув широку популярність. TypeScript підтримує класи, спадкування, інтерфейси, абстрактні класи та інші концепції об'єктно-орієнтованого програмування, що полегшує розробку складних програм та збереження коду організованим.

Незважаючи на це JavaScript все ще залишається основною мовою веб-розробки оскільки більшість проектів написано саме з її використанням. Всі ці проекти потрібно підтримувати, а отже потреба в знанні JavaScript нікуди не зникає. В дипломному проекті також буде використовуватись ця мова.

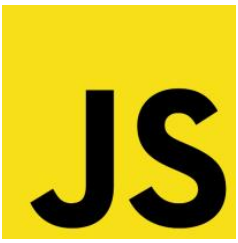


Рисунок 2.3 - Логотип мови JavaScript.

Для розмітки та стилістики веб-сторінок існує лише один варіант: HTML та CSS відповідно.

HTML (Hypertext Markup Language) - це стандартна мова розмітки для створення веб-сторінок та веб-додатків. Вона використовується для структурування та відображення вмісту веб-сторінок у вигляді тексту та мультимедійних елементів. Для цього використовується структура DOM.

DOM (Document Object Model) - це стандартне інтерфейсне API (Application Programming Interface) для HTML та XML документів. DOM представляє структуру документа у вигляді дерева об'єктів, де кожен елемент документа є об'єктом, а атрибути та текстові вузли - це властивості та значення цих об'єктів.

CSS (Cascading Style Sheets) - це мова опису стилів, що використовується для стилізації веб-документів, написаних на мові розмітки HTML. CSS визначає вигляд та форматування елементів на веб-сторінці, включаючи кольори, шрифти, розміри, розташування та інші візуальні аспекти.

Особливості CSS

- Використовується для розділення вмісту та вигляду веб-сторінки.
- Є можливість використання селекторів для вибору елементів на веб-сторінці.
- Наявний принцип каскадування для стилів.
- Для пристосування до різних пристроїв існують технології адаптивного дизайну, такі як Flexbox та Grid.
- Є можливість застосування ефектів анімації та створення переходів.

2.2.2 Вибір фреймворку

Сьогодні майже жоден проект не пишеться без використання фреймворків. Фреймворки (Frameworks) - це набори готового програмного коду, які мають певну структуру та функціональність, що допомагають розробникам швидше створювати програмне забезпечення.

Вибір правильного фреймворку не лише пришвидшує розробку, але й створює фундамент для масштабованої архітектури застосунку. У front-end розробці виділяють три основні фреймфорки: ReactJS, Angular та Vue.js. У даній роботі було прийнято рішення використовувати саме ReactJS.

React (React.js, ReactJS) [20] - відкрита JavaScript бібліотека для

створення інтерфейсів користувача, яка покликана вирішувати проблеми часткового оновлення вмісту вебсторінки, з якими стикаються в розробці односторінкових застосунків. Ця бібліотека була створена компанією Meta. Спочатку її розробляли й використовували виключно для внутрішніх потреб компанії, але згодом зробили публічною, надавши до неї доступ широкому загалу програмістів.

У React кожен компонент представлений функцією, яка приймає аргументи й повертають розмітку. React використовує особливий синтаксис — JSX (Javascript Extended). JSX дозволяє приховувати набір методів і функцій, покликаних будувати репрезентацію DOM в Javascript (Virtual DOM). Цей синтаксис максимально наближений до HTML DOM, що полегшує створення відповідного відображення для користувача.

Переваги React

- Зрозумілий синтаксис.
- Низький поріг входу.
- Присутній функціональний підхід до написання компонентів.
- Компоненто-центристський підхід до написання застосунку.
- Висока швидкість рендерингу. Забезпечується тим, що стан DOM зберігається й обчислюється у віртуальному DOM.
- Велика спільнота розробників.

Недоліки React

- Для маршрутизації потрібно окремо підключати бібліотеку.
- ReactJS часто оновлюється, що не залишає розробникам багато часу для документування.

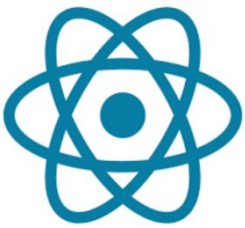


Рисунок 2.4 - Логотип ReactJS.

2.3 Технології серверної частини

Серверна частина будь-якого веб-застосунку є ядром його функціонування. Саме вона відповідає за обробку та збереження даних, а тому вибір технологій для розробки back-end проекту є найважливішим етапом у розробці надійного та безпечного сервісу. Окрім того не менш важливою складовою є забезпечення масштабованості продукту у процесі його розвитку та набуття популярності.

2.3.1 Вибір мови програмування.

Мов програмування та платформ, що спеціалізуються на написанні серверного коду на сьогодні існує дуже багато. Серед популярних рішень є декілька:

- Java та фреймворк Spring.
- JavaScript та Node.js.
- Python та фреймворки Django, Flask.
- Ruby та фреймворк Ruby on Rails.
- C# та ASP.NET.

У даній роботі було обрано використовувати мову Java. Розглянемо детільніше її особливості та переваги.

Java [21, 22] — це високорівнева об'єктно-орієнтована мова програмування загального призначення, розроблена компанією Sun Microsystems (згодом придбана корпорацією Oracle) у 1995 році. Java використовується для розробки різноманітних програм, від веб-додатків і мобільних додатків до вбудованих систем і великих корпоративних програм.

У Java є слоган: “Write once, run anywhere”. Особливістю мови є те, що програми на Java не компілюються безпосередньо в машинний код. Спочатку код на мові Java компілюється в байт-код який передається віртуальній машині Java (Java Virtual Machine) [23]. В подальшому віртуальна машина Java

перетворює байт-код шляхом його трансляції в машинний код, зрозумілий для конкретної операційної системи. Завдяки цьому досягається переносимість програми. Крім того поєднання звичайної компіляції, інтерпретація, а також JIT-компіляції підвищує швидкодію програми.

JIT компіляція [24] - це техніка оптимізації, яка використовується віртуальною машиною Java (JVM) для безпосереднього покращення швидкодії виконання Java програм. Розберемо як працює JIT-компіляція поетапно.

1. Профілювання: JVM збирає дані про те, які методи та функції викликаються найчастіше під час виконання програми. Ці дані називаються профілями виклику.

2. Компіляція в машинний код: На основі профілів виклику, JIT компілятор вибирає найбільш важливі та часто використовувані фрагменти коду програми та компілює їх безпосередньо в машинний код, який може бути виконаний на конкретній апаратній платформі. Цей машинний код зберігається в кеші для подальшого використання.

3. Виконання оптимізованого коду: Після компіляції фрагментів коду в машинний код, вони виконуються безпосередньо, замість інтерпретації байт-коду. Це дозволяє покращити швидкодію виконання програми.

Іншою особливістю Java є те, що мова має досить гарну обернену сумісність з минулими версіями. Також варто відмітити строгу типізацію цієї мови, що зменшує ймовірність програмістом зробити помилки на моменті написання програми та таким чином підвищує стабільність коду та надійність програми. Враховуючи це, можна дійти висновку, що написання застосунків відбувається швидше порівняно з іншими мовами програмування.

Також, Java має дуже гарну підтримку багатопотоковості та містить велику кількість засобів для керування конкурентним доступом до ресурсів. Це робить її одним з найкращих виборів для розробки багатокористувацьких застосунків.

Окрім перелічених переваг, у Java є ще свій механізм для знищення не

використовуваних об’єктів з пам’яті, який називається Garbage Collector.

На відміну від мови C++, де знищення об’єктів покладається на програміста, тут за це відповідає розумний алгоритм збирача сміття. Такий підхід, зменшує ризики пов’язані з витоком пам’яті та допомагає підтримувати стабільну роботу застосунку протягом всього періоду функціонування.

Загалом, Java дуже гарно зарекомендувала себе як швидка та надійна мова програмування та є відмінним вибором для enterprise розробок.

Рисунок 2.5 - Логотип мови програмування Java.

2.3.2 Огляд фреймворку

Враховуючи, що для розробки серверної частини застосунку була обрана мова програмування Java, то відповідно при виборі фреймворку будемо опиратись на варіанти, що підтримують дану мову.

Серед відомих рішень найпопулярнішим та одним з найпотужніших для розробки веб-додатків та корпоративного програмного забезпечення на сьогодні вважається Spring Framework [25]. Фреймворк був уперше випущений під ліцензією Apache 2.0 license у червні 2003 року. Перша стабільна версія 1.0 була випущена в березні 2004. Поточна версія – 6.

Рисунок 2.6 - Логотип Spring Framework.

Варто зазначити, що Spring - це не єдиний фреймворк, а скоріше набір декількох проектів одночасно. При цьому кожен з таких проектів можна

вважати окремим фреймворком. Розглянемо деякі з його основних складових.

Spring контекст (Spring Context) є ключовою складовою Spring Framework і представляє собою конфігураційну інформацію для всіх бінів, які створюються та управляються Spring IoC контейнером. Він дозволяє визначати та організовувати біни, встановлювати їх взаємозв'язки, налаштовувати параметри та керувати їх життєвим циклом.

Spring Bean - це об'єкт, який створюється та керується Spring IoC контейнером. Він є основною одиницею конфігурації та управління життєвим циклом в Spring Framework. За своєю структурою bean - це звичайний POJO (Plain Old Java Object).

Spring контейнер - це основна складова Spring Framework, яка відповідає за управління створенням, конфігурацію та керування об'єктами додатка. Він реалізує принцип інверсії управління (IoC) і забезпечує втручання контейнера у створення та управління об'єктами, відокремлюючи цю логіку від коду додатка.

Саме Spring контейнер відповідальний за механізми IoC та DI.

Inversion of Control (IoC) - це принцип проектування програмного забезпечення, за яким керування об'єктами та їх залежностями відокремлені від коду, який використовує ці об'єкти. Контейнер бере на себе відповідальність за створення та управління об'єктами, замість того, щоб цю логіку реалізувати у коді додатка. Це дозволяє забезпечити розділення конфігурації та бізнес-логіки, що робить додаток більш гнучким та легко підтримуваним.

Dependency Injection (DI) - це підхід, за якого залежності об'єкта передаються йому зовні. Замість того, щоб клас створював свої залежності, вони надаються йому через конструктор, методи або властивості. Spring IoC контейнер відповідає за впровадження цих залежностей, що дозволяє полегшити тестування та підтримку коду.

Варто зазначити, що за замовчуванням Spring не розпізнає звичайний POJO як Bean. Щоб це відбулося Bean треба оголосити. Розглянемо як це можна зробити.

Існує три способи створення бінів:

- За допомогою XML конфігурація (застарілий спосіб)
- Створення у Java коді (Новий спосіб)
- За допомогою анотацій (Новий спосіб)

Spring MVC

Spring MVC (Model-View-Controller) [26] - це веб-фреймворк, який надає архітектурну модель для розробки веб-додатків на основі Spring Framework. Він дозволяє розбити додаток на три основні компоненти: модель (Model), представлення (View) та контролер (Controller), що спрощує розробку веб-додатків з чіткою розділеністю на рівні представлення, бізнес-логіки та управління потоком даних.

Основні концепції та характеристики Spring MVC

- Модель (Model): Модель представляє собою бізнес-логіку або дані, з якими працює додаток. В Spring MVC модель може бути представлена як POJO (Plain Old Java Object), який зазвичай містить дані, які повинні бути відображені у вигляді відповіді.
- Представлення (View): Представлення відповідає за відображення даних для користувача. У Spring MVC представлення зазвичай є шаблонами, такими як JSP (JavaServer Pages) або Thymeleaf, які містять HTML-код з вбудованими тегами або виразами для відображення динамічних даних.
- Контролер (Controller): Контролер обробляє запити користувачів та взаємодіє з моделлю та представленням. В Spring MVC контролери зазвичай є класами, які анотовані відповідно до конфігурації та містять методи, які обробляють різні запити та повертають відповіді.

- DispatcherServlet: DispatcherServlet є головним контролером у Spring MVC додатку, який відповідає за маршрутизацію запитів до відповідних контролерів. Він обробляє вхідні HTTP-запити, визначає, який контролер повинен бути викликаний та передає йому виконання.

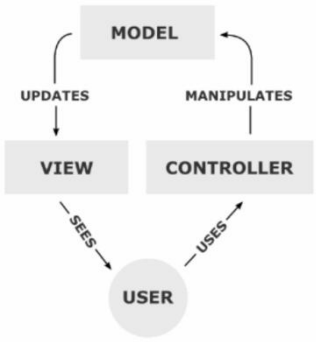


Рисунок 2.7 - Архітектурний шаблон MVC.

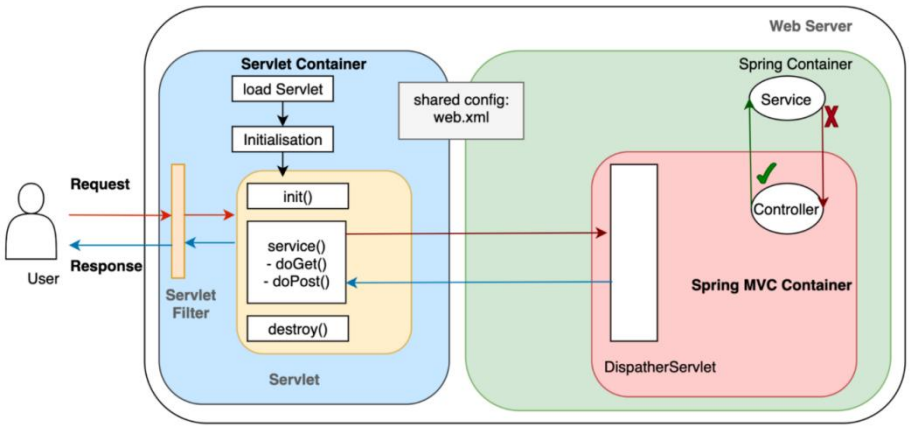


Рисунок 2.8 - Представлення роботи Spring Framework.

Spring Data

Spring Data - це проект в екосистемі Spring Framework, який спрощує роботу з доступом до даних у веб-додатках. Він надає абстракції та утиліти для взаємодії з різними джерелами даних, такими як SQL та NoSQL бази даних, забезпечуючи однорідний інтерфейс для роботи з ними. Однією з ключових функцій Spring Data є підтримка репозиторіїв, які надають гнучкий інтерфейс для взаємодії з базами даних. Репозиторії дозволяють виконувати CRUD (Create Read Update Delete) операції за допомогою стандартних методів.

Spring Security

Spring Security [27, 28] - це потужний інструмент для забезпечення аутентифікації, авторизації та захисту веб-додатків на основі Spring Framework.

Основні можливості Spring Security

- Аутентифікація: Spring Security надає різноманітні методи аутентифікації, включаючи базову аутентифікацію, аутентифікацію на основі токенів [28] та інші.
- Авторизація: Spring Security дозволяє налаштовувати права доступу користувачів до ресурсів додатка. Він підтримує визначення прав доступу на основі ролей, а також дозволяє створювати власні правила авторизації за допомогою виразів Spring Security.
- Управління сеансами: Spring Security дозволяє керувати сеансами користувачів
- Захист від атак: Spring Security забезпечує різноманітні механізми захисту від різних видів атак, таких як атаки перехоплення сесій (Session Hijacking), перехоплення паролів (Password Snooping), атаки на основі CSRF (Cross-Site Request Forgery) та інші.
- Легка інтеграція з іншими модулями Spring: Spring Security легко інтегрується з іншими модулями Spring Framework, такими як Spring MVC, Spring Boot, Spring Data ітд., що дозволяє розробникам побудувати комплексні та безпечні веб-додатки.
- Розширюваність: Spring Security надає гнучкий механізм розширення, що дозволяє розробникам додавати власні фільтри, аутентифікатори, авторизатори та інші компоненти для вирішення специфічних вимог щодо безпеки.

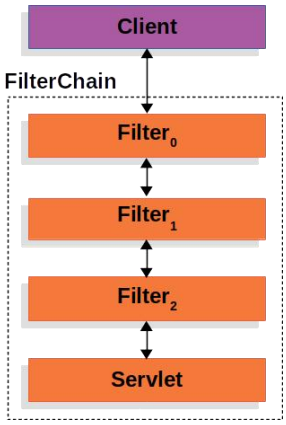


Рисунок 2.9 - Ланцюжок фільтрів у Spring Security.

Spring Boot та Hibernate

Spring Boot [29] - це фреймворк для швидкої розробки веб-додатків на основі Spring Framework. Він надає широкий спектр стандартних конфігурацій, що дозволяє створювати додатки з мінімальними зусиллями та без необхідності великих обсягів конфігурації. Spring Boot також надає вбудований веб-сервер, що спрощує розгортання додатків.

Наряду з Spring фреймворком досить часто використовують інший дуже популярний фреймворк - Hibernate.

Hibernate [30] - це фреймворк для Java, який забезпечує відображення між об'єктами і реляційними базами даних (object-relational mapping, ORM). Hibernate є вільним програмним забезпеченням, що розповсюджується на умовах GNU Lesser General Public License. Hibernate надає легкий для використання фреймворк для відображення між об'єктно-орієнтованою моделлю даних і традиційною реляційною базою даних.

Варто зазначити, що він має дуже тісну інтеграцію з Spring та розцінюється як розширена альтернатива до Spring Data. Hibernate дозволяє розробникам докладно налаштувати і контролювати поведінку свого додатку. Він надає багато параметрів конфігурації та можливостей для оптимізації роботи з базою даних. Також Hibernate може бути легко інтегрований з існуючими проектами Java, оскільки він підтримує стандарти JPA і JDBC. Це

дозволяє розробникам легко переходити на Hibernate [31, 32] або інтегрувати його в існуючі проекти без великих зусиль.

Рисунок 2.10 - Логотип Hibernate.

Відповідно до проведеного огляду, в серверній частині застосунку для дипломної роботи будуть застосовуватись комбінація з декількох фреймворків.

По-перше, задля пришвидшення роботи та зменшення необхідності в додаткових конфігураціях проекту, буде використовуватись Spring Boot. Spring Boot зручний тим, що має вбудований сервер Apache Tomcat, а отже додатково не потрібно завантажувати до себе локально та окремо конфігурувати сервер.

По-друге, для налаштувань пов’язаних з аутентифікацією та авторизацією буде використовуватись компонента Spring Security разом з JWT (JSON Web Token) [33, 34]. Такий підхід убезпечить застосунок від неконтрольованого доступу до нього сторонніх осіб. JSON Web Tokens будуть відігравати своєрідну роль ключа, по якому користувачам буде надаватись доступ до сервісу.

По-третє, для зручного управління базою даних, буде застосовуватись фреймворк Hibernate. Як зазначалось раніше, він має ряд переваг порівняно з стандартним Spring Data, а отже дозволить більш гнучко керувати даними та зберігати їх в застосунку.

Для управління додатковими залежностями в застосунку буде використано систему автоматичного збирання Gradle. За допомогою цієї системи можна буде легко підключати в проект додаткові бібліотеки. Розглянемо деякі з них.

- Lombok: Використовується для зменшення рутинних операцій з написання коду на Java.
- Flyway: Використовується для автоматизації процесу міграції баз даних.
- JsonWebTokens: Використовується для роботи з JWT.
- Spring Boot Starter Mail: Використовується для операцій пов’язаних з відправленням листів на електронну пошту користувачів.

Окрім маніпулювання залежностями, Gradle також дозволить швидко зібрати застосунок для подальшого його розгортання на сервері.

2.4 База даних

В якості бази даних для застосунку було вибрано реляційну СКБД (Систему керування базами даних) PostgreSQL [35, 36]. PostgreSQL є альтернативою як комерційним СКБД (Oracle Database, Microsoft SQL Server, IBM DB2 та інші), так і СКБД з відкритим кодом (MySQL, Firebird, SQLite).

Особливості PostgreSQL

- Відкритий код та безкоштовність: PostgreSQL розповсюджується під ліцензією PostgreSQL, яка є вільною та відкритою для використання.
- Підтримка стандартів SQL: PostgreSQL дотримується стандартів мови запитів SQL, що робить її сумісною з багатьма іншими СКБД та дозволяє легко переносити код між різними системами.
- Висока надійність та стабільність: PostgreSQL відомий своєю високою надійністю та стабільністю, що дозволяє використовувати її в критичних для бізнесу системах.
- Підтримка транзакцій та ACID властивостей: PostgreSQL підтримує транзакційну модель обробки даних, яка дозволяє забезпечити принципи ACID (Atomicity, Consistency, Isolation, Durability).
- Розширюваність: PostgreSQL має широкий набір розширень та додаткових модулів, які дозволяють розширити її функціональність та адаптувати її до конкретних потреб проекту.

ВИСНОВОК ДО РОЗДІЛУ 2

У даному розділі дипломної роботи було оглянуто популярні технології для розробки застосунків з клієнт-серверною архітектурою та відповідно обрано оптимальні рішення для реалізації сервісу.

В якості фреймворку для front-end частини буде використано ReactJS в комбінації з мовою програмування JavaScript, мовою розмітки вебсторінок HTML та мовою стилів CSS.

Для реалізації back-end частини було обрано мову програмування Java та відповідно фреймворки Spring та Hibernate. Для забезпечення захисту від несанкціонованого доступу буде використано Spring Security в поєднанні з JSON Web Token. Окрім того будуть використовуватись додаткові залежності для розсилки повідомлень через електронну пошту після реєстрації користувача та залежність для налаштування в проекті вебсокетів для спілкування в реальному часі. Вся система буде збиратись з застосуванням технології Gradle.

Базою даних для зберігання особистої інформації користувачів буде PostgreSQL. Подана СКБД на сьогодні вважається популярним та надійним рішенням для зберігання інформації у веб-застосунках.

РОЗДІЛ 3.
ПРОЕКТУВАННЯ ТА РОЗРОБКА СИСТЕМИ

3.1 Розробка серверної частини

Правильно побудована серверна частина є запорукою надійності та швидкості всього застосунку, а тому розробку варто розпочати саме з неї. Під час її створення планується побудувати базу даних та програмну компоненту, що буде керувати обробкою даних.

Для створення проекту на основі Spring Boot існує спеціальний сайт https://start.spring.io. Цей ресурс є офіційним сайтом від Spring, який надає набір функціоналу для ініціалізації проекту.

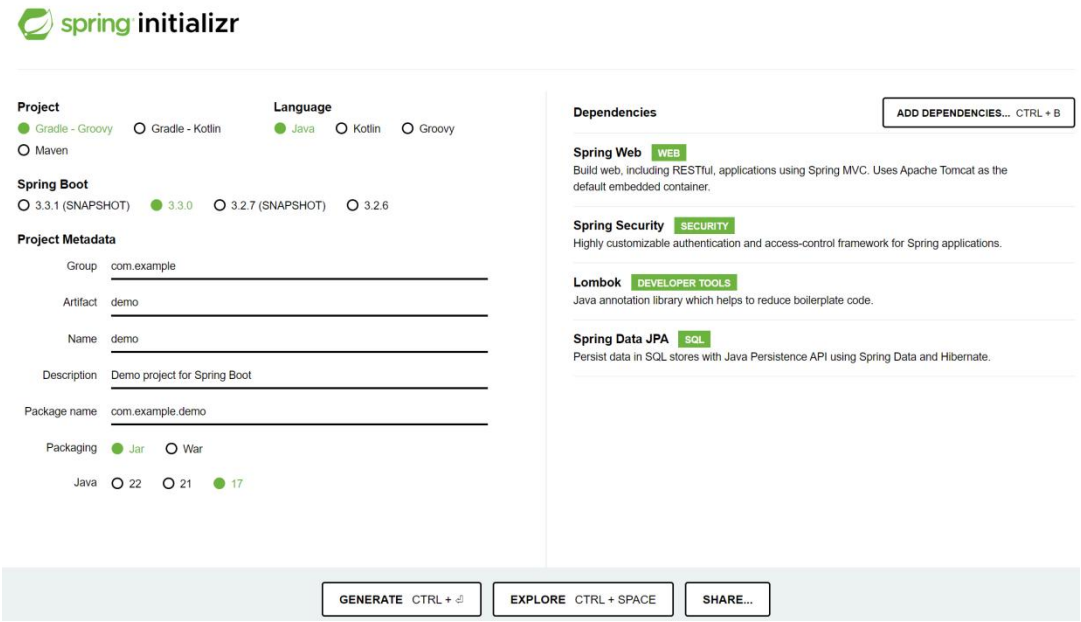


Рисунок 3.1 - Сайт для ініціалізації проекту на основі Spring Boot.

На відповідному ресурсі, потрібно вибрати версію мови, вказати назву кореневої папки проєкту, вибрати засіб для збирання проєкту та версію Spring Boot.

За бажання, одразу можна підключити необхідні для старту розробки бібліотеки, такі як Spring Web, Spring Security, Lombok та Spring Data JPA.

Після створення проекту, додамо в нього всі інші залежності, що будуть використані в процесі розробки. Для цього використаємо централізований ресурс Maven Repository.

Maven Repository - це централізоване сховище для управління залежностями у середовищі розробки програмного забезпечення заснованого на Maven або Gradle. Це важлива складова інфраструктури в екосистемі, яка дозволяє розробникам з легкістю керувати залежностями проекту та забезпечує доступність публічних або приватних бібліотек.

```
repositories {
    mavenCentral()
}

dependencies { Edit Starters...
    implementation 'org.springframework.boot:spring-boot-starter-web'

    implementation 'org.springdoc:springdoc-openapi-starter-webmvc-ui:2.1.0'

    implementation 'org.springframework.boot:spring-boot-starter-security'
    implementation 'io.jsonwebtoken:jjwt-api:0.11.5'
    implementation 'io.jsonwebtoken:jjwt-impl:0.11.5'
    implementation 'io.jsonwebtoken:jjwt-jackson:0.11.5'

    implementation 'org.springframework.boot:spring-boot-starter-websocket'

    implementation 'org.springframework.boot:spring-boot-starter-data-jpa'
    implementation 'org.postgresql:postgresql:42.6.0'
    implementation 'org.flywaydb:flyway-core:9.22.2'

    implementation 'org.springframework.boot:spring-boot-starter-mail'

    implementation 'org.springframework.boot:spring-boot-starter-thymeleaf'

    compileOnly 'org.projectlombok:lombok'
    annotationProcessor 'org.projectlombok:lombok'

    testImplementation 'org.springframework.boot:spring-boot-starter-test'
}
```

Рисунок 3.2 - Список доданих залежностей.

Розглянемо деякі з доданих залежностей:

- 'org.springframework.boot:spring-boot-starter-web' - надає необхідні бібліотеки для створення веб-застосунків, включаючи підтримку RESTful веб-сервісів, Spring MVC та вбудований Tomcat сервер.
- 'org.springframework.boot:spring-boot-starter-security' - надає засоби для забезпечення аутентифікації та авторизації.
- 'org.springframework.boot:spring-boot-starter-mail' - забезпечує підтримку відправки електронних листів за допомогою JavaMailSender.

3.1.1 Налаштування бази даних

Після створення проекту з базовими налаштуваннями варто зосередити увагу на розробці бази даних. Як зазначалось в попередньому розділі, розгортання бази даних буде здійснено за допомогою технології контейнеризації. Для цього буде використано Docker. Є декілька способів завантаження та конфігурації бази даних в Docker.

По-перше, можна вручну скачати потрібний образ з DockerHub та в подальшому для підняття локально бази використовувати команду **docker run**. Проте при такому підході потрібно буде кожного разу вручну зв'язувати порти в контейнері з портами локальної машини, що не дуже зручно.

Іншим способом є створення **Dockerfile**. Це спеціальний файл в якому можна прописати всі необхідні конфігурації для створення потрібного образу на основі якого надалі можна запустити контейнер. Щоб зібрати образ потрібно буде в директорії в якій міститься **Dockerfile** прописати команду **docker build** після чого можна використовувати команду **docker run**. Недоліком є те, що такий файл можна використати для побудови лише одного образу.

Найзручнішим варіантом, що дозволить одночасно задати всі конфігурації є створення **docker-compose.yml** файлу. Це універсальний файл, який є зрозумілим Docker та який дозволяє одночасно піднімати декілька контейнерів, Крім того в ньому можна прописати конфігурації для створення між контейнерами спеціальної мережі для їх спілкування та створення томів. Томи - це місця на диску виділені докером для зберігання інформації про контейнер.

Після відповідної конфігурації, для запуску бази даних достатньо буде прописати команду **docker-compose up** і все буде працювати. За замовчуванням всі логи контейнеру будуть виводитись в консоль. Щоб уникнути цього можна використовувати команду з ключом **-d**, що дозволить

запустити контейнер в фоновому режимі.

docker-compose.yml

```yaml
version: '3.7'
services:

  postgres-db:
    image: postgres:14.2
    restart: always
    container_name: postgres-db
    environment:
      POSTGRES_PASSWORD: 'postgres'
    ports:
      - '5432:5432'
    volumes:
      - postgres-db-volume:/var/lib/postgresql/data

volumes:
  postgres-db-volume:
    driver: local
```

Рисунок 3.3 - Налаштування docker-compose.yml файлу для проекту.

Розглянемо детальніше конфігурацію бази даних:

- Директива **version: ‘3.7’** вказує на поточну версію docker-compose.yml файла.
- Розділ **services** визначає сервіси (контейнери), які повинні бути запущені за допомогою цього файлу.
- **postgres-db** - є довільною назвою для сервісу бази даних.
- Директива **image** вказує на образ який буде використовуватись для створюваного контейнеру. В даному випадку це образ бази даних PostgreSQL версії 14.2
- Налаштування **restart: always** вказує Docker перезапускати контейнер автоматично в разі його відмови.
- Директива **container_name** є лише директивою для задання імені для контейнера та не є обов’язковою. У разі її відсутності Docker сам надасть довільне ім’я для цього контейнера.

- Далі йде налаштування змінних середовища, які будуть передаватись контейнеру. У цьому випадку, встановлюється пароль для користувача PostgreSQL.
- Секція **ports** потрібна для прокидування портів контейнера на порти хоста.
- Секція **volumes** визначає томи даних, які мають бути монтовані до контейнера. У цьому випадку, використовується том з ім'ям "postgres-db-volume", який монтується до **/var/lib/postgresql/data** у контейнері.

```
PS C:\Users\Legion\Personal\IT\MyProjects\web-app-messenger\web-app-messenger-project\messenger> docker-compose up -d
[+] Running 2/2
 ✓ Network messenger_default  Created
 ✓ Container postgres-db      Started
PS C:\Users\Legion\Personal\IT\MyProjects\web-app-messenger\web-app-messenger-project\messenger> docker ps
CONTAINER ID   IMAGE           COMMAND                  CREATED         STATUS         PORTS                    NAMES
2abf3d3777e3   postgres:14.2   "docker-entrypoint.s…"   7 seconds ago   Up 5 seconds   0.0.0.0:5432->5432/tcp   postgres-db
PS C:\Users\Legion\Personal\IT\MyProjects\web-app-messenger\web-app-messenger-project\messenger> docker exec -it 2abf3d3777e3 /bin/bash
root@2abf3d3777e3:/# psql -U postgres
psql (14.2 (Debian 14.2-1.pgdg110+1))
Type "help" for help.

postgres=#
```

Рисунок 3.4 - Запуск контейнера та підключення до бази даних.

3.1.2 Проектування бази даних

Гарно спроектована база даних є запорукою побудови масштабованого додатку та структурованості інформації в застосунку.

По-перше, потрібно проаналізувати які сутності будуть потрібні в створюваному застосунку та які зв’язки між ними повинні бути. Основною складовою буде сутність користувача застосунку, яка буде зберігати інформацію вказану користувачем при реєстрації та модифікації акаунту. Для цього створимо таблицю **users**.

Крім того враховуючи специфіку застосунку база даних повинна мати таблиці для чатів, повідомлень та стану повідомлень. Стан повідомлень буде відображати чи є повідомлення прочитаним для відправника та кожного отримувача повідомлення. Окремо потрібно створити таблицю для збереження інформації про учасників чату, яка буде містити інформацію про те до яких чатів відноситься кожен з користувачів.

Враховуючи те, що застосунок буде підтримувати функцію підписки на акаунти, потрібно буде створити окрему таблицю для зберігання інформації такого типу.

Для механізмів реєстрації, авторизації та аутентифікації в застосунку будуть використовуватись допоміжні таблиці.

Таблиця **jwts** буде потрібна для збереження токенів доступу користувачів до ресурсу. Таблиця буде підтримувати два типи токенів: **ACCESS** та **REFRESH**. **ACCESS** токен є токеном з невеликим терміном дії, який безпосередньо використовується для перевірки прав користувачів та надання доступу до сервісу. На відміну від **ACCESS** токена, **REFRESH** має довший термін дії та буде використовуватись для оновлення **ACCESS** токена у разі закінчення його терміну придатності. Поєднання таких двох ключів дозволить створити умови при яких користувачу застосунку не потрібно буде дуже часто авторизовуватись на ресурсі, але при цьому збережеться захищеність акаунту.

В такому випадку навіть при непомітному викраденні зловмисником токену доступу, він не зможе використовувати його протягом довгого періоду часу. При цьому користувачу не потрібно буде перезаходити в акаунт для поновлення токенів, адже за поновлення буде відповідати **REFRESH** токен.

Також варто зазначити, що механізм доступу до сервісу буде побудований на основі ролей. Інформація про роль користувача буде зберігатись в токені та в таблиці **users**.

Іншою таблицею, що буде використовуватись при реєстрації буде **user_accounts**. Подана таблиця буде мітити інформацію про те в якому стані знаходиться акаунт користувача. Будуть існувати три стани: **ACTIVATED**, **REQUIRE_ACTIVATION**, **BLOCKED**. Одразу після реєстрації акаунт буде знаходитись в стані **REQUIRE_ACTIVATION**. Після цього користувачу на пошту буде надісланий спеціальний код, який потрібно буде ввести в застосунок для активації акаунта.

Для збереження кодів буде використовуватись таблиця **user_account_activation_codes**.

Для створення постів відповідає таблиця **posts**.

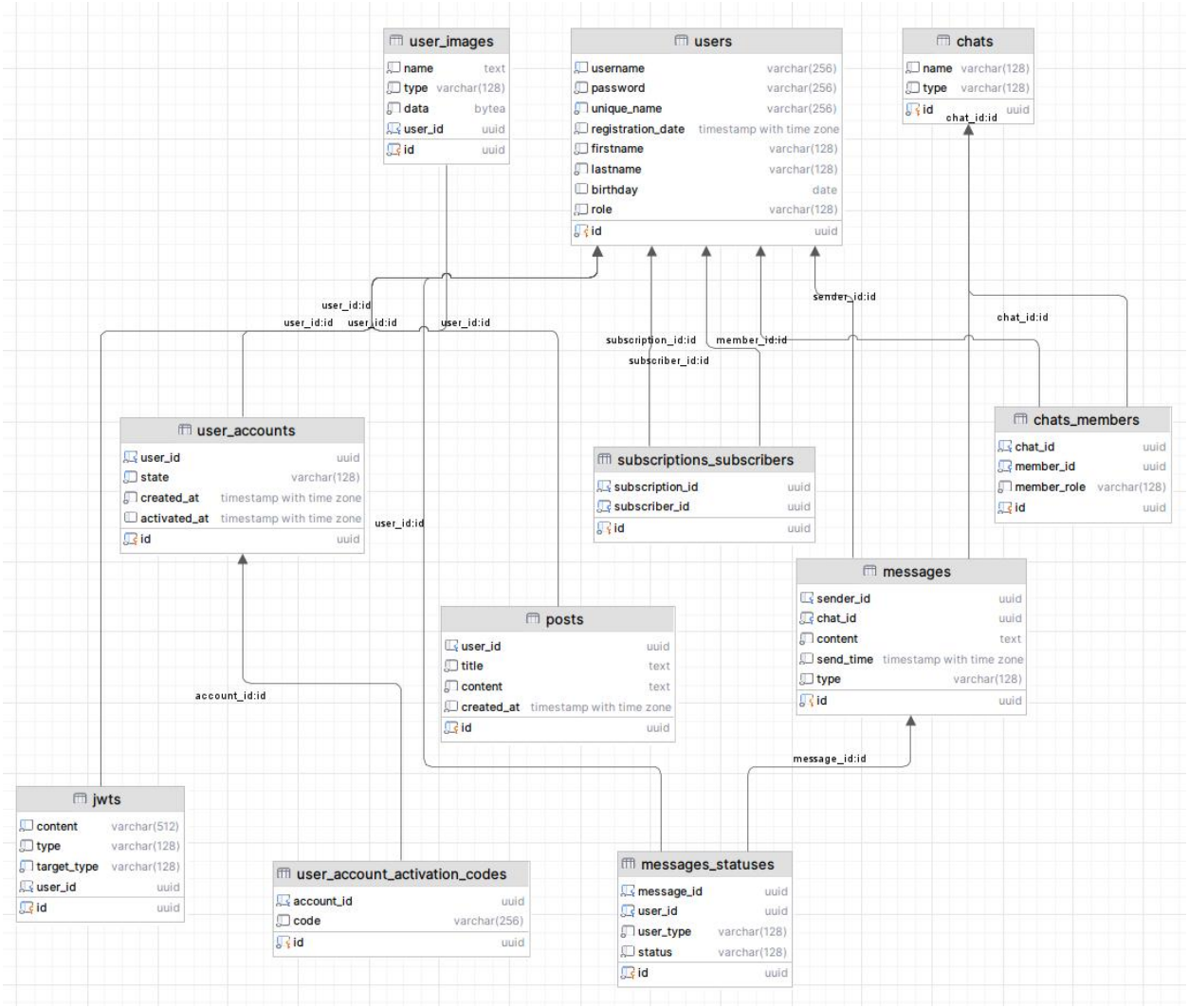


Рисунок 3.5 - Зв’язок таблиць в базі даних.

Для зручного створення бази даних при запуску застосунку, буде використовуватись популярна бібліотека для міграції **Flyway**.

Міграція баз даних - це процес управління змінами схеми бази даних, який дозволяє розробникам контролювати та автоматизувати зміни структури бази даних.

В файлі build.gradle за це відповідає залежність 'org.flywaydb:flyway-core:9.22.2'

Основні концепції Flyway включають:

- Міграції: Міграції - це набір SQL скриптів, які описують зміни, які потрібно застосувати до бази даних. Ці скрипти можуть містити зміни в структурі бази даних, такі як створення таблиць, зміна колонок або індексів, і так далі.
- Метадані: Flyway використовує спеціальну таблицю **flyway_schema_history**, щоб відстежувати стан міграцій. Вона зберігає інформацію про те, які міграції були вже застосовані до бази даних, що дозволяє уникнути повторного виконання скриптів.
- Версіонування: Flyway дозволяє версіонувати міграції, тобто вказувати порядок, в якому міграції повинні бути застосовані. Це допомагає управляти залежностями між змінами схеми бази даних.
- Автоматична міграція: Flyway може бути легко інтегрована в процес розгортання додатків або в конфігурацію Docker контейнерів, щоб автоматично застосовувати міграції при запуску або оновленні додатку.

Для того щоб скрипти міграції виконувались при запуску додатку, їх потрібно розмістити в папці **src/main/resources/db/migration** проекту.

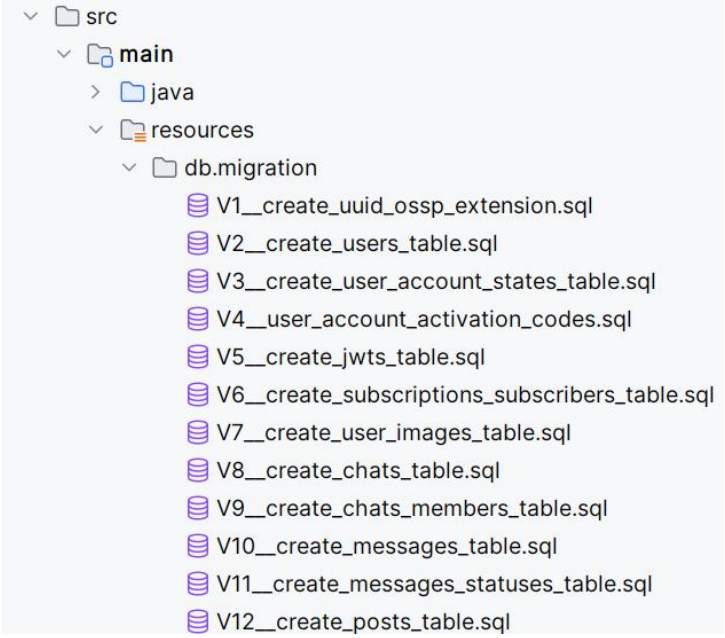


Рисунок 3.6 - Налаштування скриптів для бази даних.

```
DROP TABLE IF EXISTS users CASCADE;
CREATE TABLE users (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    username VARCHAR(256) UNIQUE NOT NULL,
    password VARCHAR(256) NOT NULL,
    unique_name VARCHAR(256) UNIQUE NOT NULL,
    registration_date TIMESTAMPTZ NOT NULL DEFAULT current_timestamp,
    firstname VARCHAR(128) NOT NULL,
    lastname VARCHAR(128) NOT NULL,
    birthday DATE,
    role VARCHAR(128) NOT NULL DEFAULT 'USER',
    CONSTRAINT users_username_check CHECK(username ~*'^[^[:space:]]+@[^[:space:]]+$'),
    CONSTRAINT users_role_check CHECK(role IN ('USER', 'ADMIN', 'ROOT'))
);
```

Рисунок 3.7 - Приклад скрипта для створення таблиці **users**.

3.1.3 Підключення бази даних до проекту

Підключення бази даних в Spring Boot проекті є досить простим. Для налаштувань існує спеціальний файл **application.yml**. В цьому файлі можна вказувати всі необхідні параметри та змінні для застосунку.

```
spring:
  datasource:
    driver-class-name: org.postgresql.Driver
    url: ${DATABASE_URL}
    username: ${DATABASE_USERNAME}
    password: ${DATABASE_PASSWORD}
```

Рисунок 3.8 - Змінні в файлі **application.yml**.

Гарним тоном вважається не прописувати пароль на інші вразливі налаштування безпосередньо в цьому файлі. Так, наприклад, DATABASE_URL, DATABASE_USERNAME, DATABASE_PASSWORD - є змінними середовища, які потрібно буде вказати при запуску застосунку. Такий підхід убезпечить від витоку цих даних назовні та потенційного перехоплення їх зловмисниками.

3.1.4 Налаштування Spring Security та прав доступу

Для налаштування прав доступу в застосунку було створено три ролі користувачів: USER, ADMIN, ROOT.

- **USER** - це звичайний користувача, що створюється після успішної реєстрації.
- **ADMIN** - це адміністратор платформи, що може видаляти та блокувати інших користувачів, а також має можливість створювати інших адміністраторів.
- **ROOT** - це адміністратор платформи з розширеними можливостями в порівнянні з звичайним ADMIN, так як в його спектр можливостей також входить функція видалення звичайних адміністраторів. ROOT в системі є лише один, а тому пароль для доступу до його акаунту повинен бути максимально складним для взлому.

Кожна з ролей в застосунку має певні права. Розглянемо їх детальніше.

- **READ_USER**, **READ_ADMIN**, **READ_ROOT** - це набір прав на читання акаунтів звичайних користувачів та адміністраторів платформи.
- **CREATE_USER**, **CREATE_ADMIN** - користувачі, що мають ці права можуть створювати інших користувачів або адмінів відповідно.
- **UPDATE_USER**, **UPDATE_ADMIN**, **UPDATE_ROOT** - використовуються для можливості модифікації акаунтів.
- **DELETE_USER**, **DELETE_ADMIN** - потрібні для видалення користувачів.

В програмі для перелічених ролей та прав було створено два класи перерахувань (enum), з відповідними назвами **Role** та **Permission**.

```
AndriiHliuza *
@RequiredArgsConstructor
public enum Role {
    9 usages
    USER(
            Set.of(
                    Permission.READ_USER,
                    Permission.UPDATE_USER,
                    Permission.DELETE_USER
            )
    ),
```

Рисунок 3.9 - Права користувача з роллю **USER**.

```
    ADMIN(
            Set.of(
                    Permission.READ_USER,
                    Permission.CREATE_USER,
                    Permission.UPDATE_USER,
                    Permission.DELETE_USER,
                    Permission.READ_ADMIN,
                    Permission.CREATE_ADMIN,
                    Permission.UPDATE_ADMIN,
                    Permission.DELETE_ADMIN
            )
    ),
    9 usages
    ROOT(
            Set.of(
                    Permission.READ_USER,
                    Permission.CREATE_USER,
                    Permission.UPDATE_USER,
                    Permission.DELETE_USER,
                    Permission.READ_ADMIN,
                    Permission.CREATE_ADMIN,
                    Permission.UPDATE_ADMIN,
                    Permission.DELETE_ADMIN,
                    Permission.READ_ROOT,
                    Permission.UPDATE_ROOT
            )
    );
    private final Set<Permission> permissions;
    1 usage   AndriiHliuza
    public List<SimpleGrantedAuthority> getAuthorities() {
        List<SimpleGrantedAuthority> authorities = permissions
                .stream() Stream<Permission>
                .map(
                        permission -> new SimpleGrantedAuthority(permission.name())
                ) Stream<SimpleGrantedAuthority>
                .collect(Collectors.toList());

        authorities.add(new SimpleGrantedAuthority( role: "ROLE_" + name()));

        return authorities;
    }
}
```

Рисунок 3.10 - Налаштування прав **ADMIN** та **ROOT**.

За допомогою методу **getAuthorities** здійснюється отримання прав користувачів у момент надходження нового **HTTP** запиту на сервер. Для цього використовується вбудований в **Spring** клас **SimpleGrantedAuthority**, а також метод в класі **USER**, що на основі поточної ролі користувача викликає метод **role.getAuthorities()**. Варто зазначити, що на відміну від дозволів, до ролей потрібно додавати префікс **ROLE_**, щоб Spring Security міг розпізнати поточну роль користувача в момент доступу до ресурсу.

Перейдемо до конфігурації Spring Security. Для налаштувань Spring Security потрібно створити відповідний клас конфігурації. Для позначення класа Java таким, що містить конфігураційні дані, потрібно перед його оголошенням вказати анотацію **@Configuration**. Щоб показати, що цей клас містить налаштування безпеки, додатково потрібно вказати анотацію **@EnableWebSecurity**.

```java
AndriiHliuza
@Configuration
@EnableWebSecurity
@RequiredArgsConstructor
@EnableMethodSecurity
public class SecurityConfig {
    private final JwtAuthenticationFilter jwtAuthenticationFilter;
    private final UserDetailsService userDetailsService;
    private final LogoutHandler logoutHandler;
    private final UserAuthenticationEntryPoint userAuthenticationEntryPoint;

    AndriiHliuza
    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
        return http
                .csrf(AbstractHttpConfigurer::disable)
                .cors(Customizer.withDefaults())
                .httpBasic(
                        httpSecurityHttpBasicConfigurer ->
                                httpSecurityHttpBasicConfigurer.authenticationEntryPoint(userAuthenticationEntryPoint)
                )
                .authorizeHttpRequests(
                        request -> {
                            request.requestMatchers(
                                    HttpMethod.POST,
                                    "/api/registration",
                                    "/api/authentication",
                                    "/api/user/*/account/activation",
                                    "/api/user/*/account/activation/email",
                                    "/api/jwt/**"
                                ).permitAll()
```

Рисунок 3.11 - Оголошення та налаштування конфігураційного файлу безпеки.

При налаштуванні конфігураційного файлу безпеки варто зазначати, шляхи та доступи до них. Розглянемо детальніше подані доступи:

Шляхи з відкритим доступом:

- "**/api/registration**" та "**/api/authentication**" - забезпечують функціональність для створення акаунтів користувачів та механізму авторизації.
- "**/api/ws/****" - необхідний для веб-сокет з'єднання. Налаштування безпеки для цього шляху, було розглянуто при створенні перехоплювача для протоколу веб-сокетів.
- "**/v2/api-docs**" - використовується в допоміжних цілях, таких як тестування програмного забезпечення.

Шляхи з обмеженням доступу відповідно до ролі користувача:

- "**/api/users/****" - шлях доступний, для користувачів з ролями: **USER**, **ADMIN**, **ROOT**.
- "**/api/admins/****" - доступ до цього шляху мають лише адміністратори.
- "**/api/roots/****" - доступ мають лише користувачі з роллю **ROOT**. Звичайні адміністратори не можуть використовувати цей шлях.

```
            .sessionManagement(
                    httpSecuritySessionManagementConfigurer -> httpSecuritySessionManagementConfigurer
                            .sessionCreationPolicy(SessionCreationPolicy.STATELESS)
            )
            .authenticationProvider(authenticationProvider())
            .addFilterBefore(jwtAuthenticationFilter, UsernamePasswordAuthenticationFilter.class)
            .logout(
                    httpSecurityLogoutConfigurer -> httpSecurityLogoutConfigurer
                            .logoutUrl("/api/logout")
                            .addLogoutHandler(logoutHandler)
                            .logoutSuccessHandler(
                                    (request, response, authentication) -> SecurityContextHolder.clearContext()
                            )

            )
            .build();
    }

    AndriiHliuza
    @Bean
    public AuthenticationProvider authenticationProvider() {
        DaoAuthenticationProvider daoAuthenticationProvider = new DaoAuthenticationProvider();
        daoAuthenticationProvider.setUserDetailsService(userDetailsService);
        daoAuthenticationProvider.setPasswordEncoder(passwordEncoder());
        return daoAuthenticationProvider;
    }

    AndriiHliuza
    @Bean
    public PasswordEncoder passwordEncoder() { return new BCryptPasswordEncoder(); }

    AndriiHliuza
    @Bean
    public AuthenticationManager authenticationManager(AuthenticationConfiguration authenticationConfiguration) throws Exception {
        return authenticationConfiguration.getAuthenticationManager();
    }
}
```

Рисунок 3.12 - Налаштування конфігураційного файлу безпеки.

Розберемо основні налаштування.

- Метод securityFilterChain створює ланцюжок фільтрів безпеки.

  Ключові етапи конфігурації в цьому методі:

  - Включаємо CORS (Cross-Origin Resource Sharing) за допомогою .cors(Customizer.withDefaults()).
  - Налаштовуємо HTTP Basic Authentication та вказуємо власну точку входу (userAuthenticationEntryPoint).
  - Налаштовуємо авторизацію запитів (authorizeHttpRequests), для шляхів застосунку. Деякі шляхи мають дозвіл на доступ для всіх користувачів, інші доступні лише певним ролям.
  - Налаштовуємо аутентифікаційний провайдер та додаємо JWT фільтр перед стандартним фільтром аутентифікації.

- Метод authenticationProvider створює та налаштовує DaoAuthenticationProvider, який використовує userDetailsService для отримання даних користувачів. Також вказуємо passwordEncoder для перевірки паролів.
- Метод passwordEncoder повертає BCryptPasswordEncoder, який використовується для хешування паролів.
- Метод authenticationManager створює AuthenticationManager за допомогою конфігурації аутентифікації.

Розглянемо принцип роботи JWT фільтра.

На початку роботи **JwtAuthenticationFilter** дістає з заголовку HTTP запита токен користувача. З токена дістається **username** користувача та сам токен перевіряється на валідність. Під час валідації перевіряється наявність токена в базі даних, порівнюються ім’я користувача з іменем в базі даних та перевіряється чи не вийшов термін придатності. У разі успішного проходження всіх етапів перевірки, користувач вважається таким, що має доступ. Після цього відповідна інформація потрапляє в Security Context.

```
@Override
protected void doFilterInternal(
        @NonNull HttpServletRequest request,
        @NonNull HttpServletResponse response,
        @NonNull FilterChain filterChain
) {
    try {
        final String authenticationHeader = request.getHeader(HttpHeaders.AUTHORIZATION);
        final String jwt;
        final String username;

        if (authenticationHeader != null && authenticationHeader.startsWith("Bearer ")) {
            jwt = authenticationHeader.substring( beginIndex: 7);
            username = jwtUtil.extractUsername(jwt);

            if (username != null && SecurityContextHolder.getContext().getAuthentication() == null) {
                UserDetails userDetails = userDetailsService.loadUserByUsername(username);

                if (jwtUtil.isTokenValid(jwt, userDetails)) {

                    UsernamePasswordAuthenticationToken usernamePasswordAuthenticationToken = new UsernamePasswordAuthenticationToken(
                            userDetails,
                             credentials: null,
                            userDetails.getAuthorities()
                    );

                    usernamePasswordAuthenticationToken.setDetails(
                            new WebAuthenticationDetailsSource().buildDetails(request)
                    );

                    SecurityContextHolder.getContext().setAuthentication(usernamePasswordAuthenticationToken);
                }
            }
        }
        filterChain.doFilter(request, response);
    } catch (Exception exception) {
        handlerExceptionResolver.resolveException(request, response, null, exception);
    }
}
}
```

Рисунок 3.13 - Програмна реалізація методу для перевірки JWT.

Наступною ланкою до якою потрапляє запит є контролери. Контролери є своєрідними точками які надають доступ до різних функцій застосунку.

```
AndriiHliuza
@GetMapping("/users/{username}/account")
@PreAuthorize("hasAuthority('READ_USER')")
public UserAccountDto getUserAccountByUserUsername(@PathVariable String username) throws Exception {
    return userService.getUserAccountByUserUsername(username);
}

AndriiHliuza
@PatchMapping("/users/{username}/account")
@PreAuthorize("hasAnyRole('ADMIN', 'ROOT')")
public UserAccountDto modifyUserAccount(
        @PathVariable String username,
        @RequestBody UserAccountDto userAccountDto
) throws Exception {
    return userService.modifyUserAccount(username, userAccountDto);
}
```

Рисунок 3.14 - Приклад методів в класі UserController.

Анотації **@GetMapping**, **@PostMapping**, **@PutMapping**, **@DeleteMapping** є відображеннями відповідних HTTP методів: GET, POST, PUT, DELETE.

Цікавою є анотація **@PreAuthorize**. Вона перевіряє чи є у користувача потрібна роль або дозвіл на доступ до метода контролера.

За допомогою анотацій **@PathVariable**, **@RequestParam**, **@RequestBody** приймаються дані у форматі JSON передані від front-end до back-end.

Після отримання контролером даних, подальша їх обробка є доволі однотипною. Зазвичай дані передаються сервісам, що призначені для їх обробки, після чого дані або зберігаються в базі даних, або дістаються з неї та повертаються клієнтській частині.

Для того щоб Spring вмів відрізняти різні компоненти та успішно керував залежностями існують відповідні анотації:

- **@Component** - використовується для позначення довільного компонента.
- **@Controller**, **@RestController** - позначають контролери.
- **@Service** - позначає сервіси. Сервіси використовуються для обробки даних.
- **@Repository** - позначає репозиторій. Репозиторії використовуються для операцій пов’язаних з базою даних.

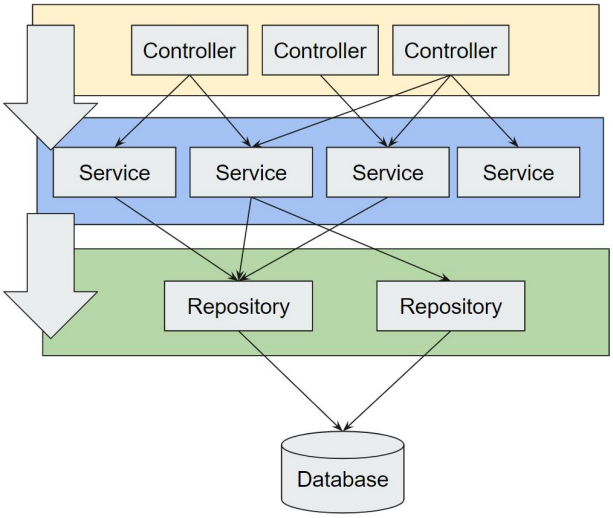


Рисунок 3.15 - Схема комунікації між сервісами серверної частини застосунку.

3.1.5 Налаштування механізму для обміну повідомленнями

За обмін миттєвими повідомленнями в застосунку відповідають протоколи веб-сокетів та STOMP [37]. Для цього було створено окремий конфігураційний клас з відповідними налаштуваннями.

```
no usages  AndriiHliuza
@Override
public void registerStompEndpoints(StompEndpointRegistry registry) {
    registry.addEndpoint( ...paths: "/api/ws")
            .setAllowedOriginPatterns("*")
            .withSockJS();
}

no usages  AndriiHliuza
@Override
public void configureMessageBroker(MessageBrokerRegistry registry) {
    registry.enableSimpleBroker( ...destinationPrefixes: "/api/messaging/topic");
    registry.setApplicationDestinationPrefixes("/api/messaging");
}
```

Рисунок 3.16 - Конфігурація веб-сокетів.

Метод **registerStompEndpoints** використовується для реєстрації точки доступу WebSocket. В даному випадку реєструємо точку доступу **/api/ws**. У методі **configureMessageBroker** налаштовуємо брокера повідомлень для веб-сокетів.

По-перше, ввімкнемо простого брокера (enableSimpleBroker) для шляху **/api/messaging/topic**, що дозволить використовувати анотації **@SendTo** та метод **convertAndSend()** для відправки повідомлень. Крім того, встановлюємо префікс **/api/messaging**, що означає, що всі повідомлення, які починаються з цього шляху, будуть відправлені в додаток.

Додатково налаштуємо перехоплювача, який буде перевіряти користувацький токен у момент встановлення з'єднання веб-сокетів між front-end та back-end частинами.

```
@Override
public Message<?> preSend(@NonNull Message<?> message, @NonNull MessageChannel channel) {
    StompHeaderAccessor accessor = MessageHeaderAccessor.getAccessor(message, StompHeaderAccessor.class);
    List<StompCommand> stompCommands = List.of(
            StompCommand.CONNECT,
            StompCommand.SUBSCRIBE,
            StompCommand.SEND
    );

    if (accessor == null) {
        throw new IllegalArgumentException("Accessor is null");
    }
    StompCommand receivedStompCommand = accessor.getCommand();
    if (stompCommands.contains(receivedStompCommand)) {
        boolean isJwtValid = false;
        List<String> authorizationHeader = accessor.getNativeHeader( headerName: "Authorization");
        if (authorizationHeader != null && !authorizationHeader.isEmpty()) {
            String authorizationHeaderValue = authorizationHeader.get(0);
            if (authorizationHeaderValue != null && authorizationHeaderValue.startsWith("Bearer ")) {
                String jwt = authorizationHeaderValue.substring( beginIndex: 7);
                if (!jwt.isBlank()) {
                    String username = jwtUtil.extractUsername(jwt);
                    UserDetails userDetails = userDetailsService.loadUserByUsername(username);
                    if (userDetails != null) {
                        try {
                            if (jwtUtil.isTokenValid(jwt, userDetails)) {
                                isJwtValid = true;
                            }
                        } catch (Exception e) {
                            throw new IllegalArgumentException(e);
                        }
                    }
                }
            }
        }

        if (!isJwtValid) {
            throw new InvalidTokenException("Jwt is invalid");
        }
    }
    return message;
}
```

Рисунок 3.17 - Налаштування для безпечного підключення через протокол веб-сокетів.

Як і звичайний JWT фільтр, перехоплювач, також перевіряє валідність JWT токена. Проте на відміну від JWT фільтра, який здійснює перевірку при кожному запиті, веб-сокет перехоплювач можна налаштувати лише для деяких етапів комунікації через протокол веб-сокетів. Так, в даному випадку перевірка здійснюється в момент надсилання таких команд як: **CONNECT**, **SUBSCRIBE** та **SEND**.

3.1.6 Створення інструментів для шифрування даних

Для шифрування даних в застосунку було створено два окремих сервіси: **EncryptionService** та **AsymmetricEncryptionService**.

EncryptionService призначений для здійснення симетричного шифрування на базі алгоритму AES (Advanced Encryption Standard). Даний алгоритм є одним з найпопулярніших алгоритмів шифрування і широко використовується в різних сферах, включаючи інформаційну безпеку.

EncryptionService містить 3 метода: **encrypt**, **decrypt** та **match**. Метод **match** потрібен для порівнянні зашифрованого тексту з звичайним текстом. Головним призначенням цього сервісу є шифрування інформації перед збереженням її в базу даних. Наприклад, він використовується для шифрування повідомлень в чатах та токенів користувачів.

```
@Service
public class EncryptionServiceImpl implements EncryptionService {
    5 usages
    private final Cipher cipher;
    3 usages
    private final SecretKey secretKey;

    AndriiHliuza
    public EncryptionServiceImpl(
            @Value("${application.security.service.encryption.algorithm}") String encryptionAlgorithm,
            @Value("${application.security.service.encryption.aes-key}") String encryptionKey
    ) throws Exception {
        cipher = Cipher.getInstance(encryptionAlgorithm);
        secretKey = new SecretKeySpec(encryptionKey.getBytes(), encryptionAlgorithm);
    }

    13 usages  AndriiHliuza
    public String encrypt(String rawData) throws Exception {
        cipher.init(Cipher.ENCRYPT_MODE, secretKey);
        byte[] encryptedValueBytes = cipher.doFinal(rawData.getBytes());
        return Base64.getEncoder().encodeToString(encryptedValueBytes);
    }

    3 usages  AndriiHliuza
    public String decrypt(String encryptedData) throws Exception {
        cipher.init(Cipher.DECRYPT_MODE, secretKey);
        byte[] encryptedValueBytes = Base64.getDecoder().decode(encryptedData);
        byte[] decryptedValueBytes = cipher.doFinal(encryptedValueBytes);
        return new String(decryptedValueBytes);
    }

    2 usages  AndriiHliuza
    public boolean matches(String rawData, String encryptedData) throws Exception {
        String encryptedRawValue = encrypt(rawData);
        return encryptedRawValue.equals(encryptedData);
    }
}
```

Рисунок 3.18 - Програмна реалізація **EncryptionService**.

AsymmetricEncryptionService є складнішим, оскільки використовує асиметричне шифрування на основі приватних та публічних ключів. Даний тип шифрування використовується в момент обміну повідомленнями між клієнтською та серверної частинами. Для збереження ключів, сервер використовує структуру даних HashMap. Ключі створюються окремо на сервері та клієнтській частині застосунку та в момент аутентифікації користувача відбувається обмін публічними складовими. Для кожного клієнта сервер генерує та надає окремий публічний ключ. Для шифрування використовується алгоритм RSA.

RSA (Rivest–Shamir–Adleman) - це криптографічний алгоритм, який використовується для шифрування та цифрового підпису даних. Він названий на честь його винахідників: Рональда Рівеста, Аді Шаміра та Леонарда Адлемана. RSA є одним з найбільш поширених алгоритмів криптографії та використовується в різних сферах, таких як захищені комунікації в Інтернеті, електронний підпис, захист конфіденційності даних та ін.

```
1 usage   AndriiHliuza
public EncryptionKeyDto exchangePublicKeys(EncryptionKeyDto encryptionKeyDto) throws Exception {
    User currentUser = authenticationService.getCurrentUser();

    KeyPair serverKeyPair = generateKeyPair();
    PrivateKey serverPrivateKey = serverKeyPair.getPrivate();
    PublicKey serverPublicKey = serverKeyPair.getPublic();
    PublicKey userPublicKey = convertStringToPublicKeyAndGet(encryptionKeyDto.getEncryptionKey());
    CryptoKeysContainer cryptoKeysContainer = CryptoKeysContainer
            .builder()
            .serverPrivateKey(serverPrivateKey)
            .serverPublicKey(serverPublicKey)
            .userPublicKey(userPublicKey)
            .build();
    cryptoKeysContainers.put(currentUser.getId(), cryptoKeysContainer);

    return EncryptionKeyDto
            .builder()
            .encryptionKey(Base64
                    .getEncoder()
                    .encodeToString(serverPublicKey.getEncoded())
            )
            .build();
}

1 usage   AndriiHliuza
private KeyPair generateKeyPair() throws NoSuchAlgorithmException {
    KeyPairGenerator keyPairGenerator = KeyPairGenerator.getInstance(ALGORITHM);
    keyPairGenerator.initialize( keysize: 2048);
    return keyPairGenerator.generateKeyPair();
}
```

Рисунок 3.19 - Генерація пари ключів та обмін ними.

```
13 usages  AndriiHliuza
@Override
public String encrypt(String rawData) throws Exception {
    PublicKey publicKey = getPublicKey();
    if (publicKey == null) {
        throw new EncryptionKeyNotFoundException("User's public encryption key does not exist");
    }
    Cipher cipher = Cipher.getInstance(ALGORITHM);
    cipher.init(Cipher.ENCRYPT_MODE, publicKey);
    byte[] encryptedBytes = cipher.doFinal(rawData.getBytes());
    return Base64.getEncoder().encodeToString(encryptedBytes);
}

3 usages  AndriiHliuza
@Override
public String decrypt(String encryptedData) throws Exception {
    PrivateKey privateKey = getPrivateKey();
    if (privateKey == null) {
        throw new EncryptionKeyNotFoundException("Server's private encryption key does not exist");
    }
    Cipher cipher = Cipher.getInstance(ALGORITHM);
    cipher.init(Cipher.DECRYPT_MODE, privateKey);
    byte[] encryptedBytes = Base64.getDecoder().decode(encryptedData);
    byte[] decryptedBytes = cipher.doFinal(encryptedBytes);
    return new String(decryptedBytes);
}
```

Рисунок 3.20 - Методи асиметричного шифрування та розшифрування.

Для програмної реалізації алгоритму асиметричного шифрування на мові Java було обрано використовувати вбудовані засоби самої мови такі як клас **Cipher** з пакету **javax.crypto**, а також класи **PublicKey**, **PrivateKey** та **KeyGenerator** з пакету **java.security**.

javax.crypto.Cipher є частиною Java Cryptography Extension (JCE) та використовується для виконання криптографічних операцій, таких як шифрування та розшифрування.

Основні методи класу Cipher:

- Cipher.getInstance(ALGORITHM): Створює об'єкт Cipher для заданого алгоритму.
- init(int opmode, Key key): Ініціалізує Cipher для шифрування (Cipher.ENCRYPT_MODE) або розшифрування (Cipher.DECRYPT_MODE), використовуючи заданий ключ.
- doFinal(byte[] input): Виконує остаточну операцію шифрування або розшифрування на заданому масиві байтів.

Для зберігання згенерованих ключів для користувача був створений клас **CryptoKeysContainer**.

В результаті розробки серверної частини було отримано таку структуру папок:

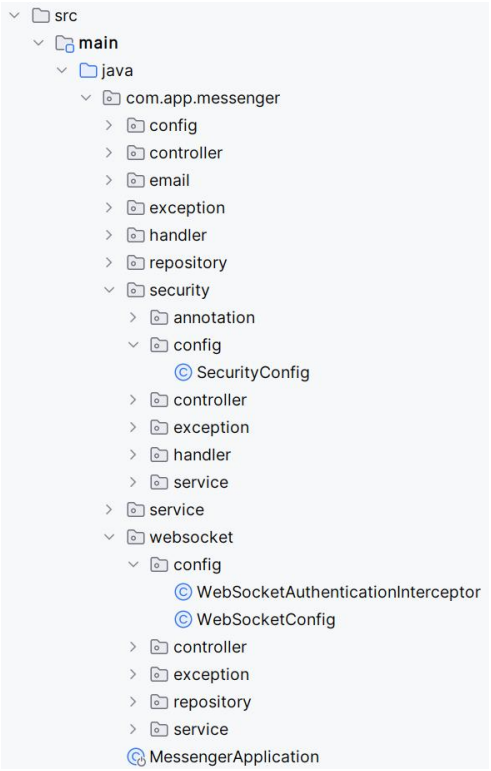


Рисунок 3.21 - Структура back-end проекту.

3.2 Розробка клієнтської частини

Клієнтська частина застосунку складається з компонентів React, що взаємодіють один з одним. Додатково до проекту було підключено декілька бібліотек для розширення функціоналу застосунку. Розглянемо деякі з них.

- **react-router-dom** - бібліотека потрібна для забезпечення маршрутизації в застосунку.
- **axios** [38] - надає зручний функціонал для відправлення запитів до серверної частини.
- **sockjs-client** та react-stomp - потрібні для встановлення веб-сокет з'єднання.
- **jwt-decode** - декодує інформацію з jwt токенів.
- **formik** - має зручний функціонал для роботи з формами.
- **node-forge** - бібліотека для асиметричного шифрування.

3.2.1 Налаштування маршрутизації

За створення маршрутів в застосунку відповідає компонент **RoutesProcessor**.

```
{/* Routes for not authenticated users */}
<Route element={<NotAuthenticatedUserRoute />} >
    <Route path={SIGH_IN_ROUTE} element={<LoginPage />} />
    <Route path={SIGN_UP_ROUTE} element={<RegistrationPage />} />
    <Route path="/user/:uniqueName/account/activation" element={<AccountActivationPage />} />
</Route>

{/* Routes for authenticated users */}
<Route element={<AuthenticationBasedRoute />} >

    <Route element={<RoleUserRoute />} >
        <Route path={USER_ROUTE} element={<UserRoute />} >
            <Route index element={<ViewUsersRoute />} />
            <Route path=":uniqueName" element={<UserProfilePage />} />
            <Route path=":uniqueName/account" element={<ProfileModificationPage />} />
            <Route path=":uniqueName/chats" >
                <Route index element={<ViewChatsRoute />} />
                <Route path="creation-panel" element={<ChatCreationPage />} />
            </Route>
            <Route path=":uniqueName/subscribers" element={<UserSubscribersPage />} />
            <Route path=":uniqueName/subscriptions" element={<UserSubscriptionsPage />} />
            <Route path="search" element={<SearchUsersPage />} />
        </Route>
    </Route>

    <Route path={CHATS_ROUTE} element={<ChatRoute />} >
        <Route path=":chatId" element={<ChatPage />} />
    </Route>

    <Route element={<RoleAdminRoute />} >
        <Route path={ADMIN_ROUTE} element={<AdminRoute />} >
            <Route index element={<ViewAdminsRoute />} />
            <Route path=":uniqueName" element={<AdminProfilePage />} />
            <Route path=":uniqueName/account" element={<AdminProfileModificationPage />} />
            <Route path="creation-panel" element={<AdminCreationPage />} />
            <Route path="search" element={<SearchAdminsPage />} />
        </Route>
    </Route>

    <Route element={<RoleRootRoute />} >
        <Route path={ROOT_ROUTE} element={<RootRoute />} >
            <Route path=":uniqueName" element={<RootProfilePage />} />
            <Route path=":uniqueName/account" element={<RootProfileModificationPage />} />
        </Route>
    </Route>
```

Рисунок 3.22 - Шляхи та компоненти front-end застосунка.

Маршрути побудовані таким чином, щоб розділити функціонал для користувачів з різними ролями, а також для звичайних відвідувачів сайту.

Для неавтентифікованих користувачів вибір маршрутів є невеликим. Вони можуть перейти або на головну сторінку застосунку, або на сторінки реєстрації та аутентифікації.

Для користувачів, що пройшли етап аутентифікації існує **AuthenticationBasedRoute**. При потраплянні сюди у користувача

перевіряється його роль. Роль користувача дістається з токена при аутентифікації та зберігається у пам'яті застосунку. Самі ж токени зберігаються у локальному сховищі браузера та за необхідності здійснення запиту до back-end дістаються звідти.

Крім того, на етапі проходження компонента **AuthenticationBasedRoute** здійснюється веб-сокет з'єднання та обмін ключами для шифрування.

```
let socket = new SockJS(API_WEB_SOCKET_URL);
let stompClient = Stomp.over(socket);
```

Рисунок 3.23 - Створення STOMP клієнта.

```
async function exchangeKeys() {
    await exchangePublicEncryptionKeys(user.username);
    let aesKeyResponse = await getEncryptedAesKey();
    let aesKeyResponseData = aesKeyResponse?.data;
    if (aesKeyResponseData) {
        let userPrivateKeyString = localStorage.getItem("user-private-key");
        userPrivateKeyString = "-----BEGIN RSA PRIVATE KEY-----\n" + userPrivateKeyString + "\n-----END RSA PRIVATE KEY-----";
        let userPrivateKey = forge.pki.privateKeyFromPem(userPrivateKeyString);
        let decryptedAesKey = userPrivateKey.decrypt(forge.util.decode64(aesKeyResponseData?.encryptionKey));
        localStorage.setItem("aes-key", decryptedAesKey);
    }
}
exchangeKeys();

if (!stompClient.connected && user.authenticated && user.role !== Role.ADMIN && user.role !== Role.ROOT) {
    let accessToken = localStorage.getItem("access-token");
    stompClient.connect({ Authorization: `Bearer ${accessToken}` }, onWebSocketConnected, onWebSocketConnectionError);
}
```

Рисунок 3.24 - Обмін ключами на стороні клієнта та встановлення веб-сокет з'єднання.

За це відповідає команда **stompClient.connect({ Authorization: `Bearer ${accessToken}` }, onWebSocketConnected, onWebSocketConnectionError);**

У разі успішного з'єднання клієнт веб-сокетів підписується на чати до яких належить користувач. Також, окремо відбувається підписка на особисті повідомлення користувача.

```
let currentUserChatsResponse = await getCurrentUserChats(3);
let currentUserChats = currentUserChatsResponse?.data;
if (currentUserChats) {
    for (let i = 0; i < currentUserChats.length; i++) {
        let currentUserChat = currentUserChats[i];
        if (currentUserChat) {
            let chatId = currentUserChat?.id;
            if (chatId) {
                stompClient.subscribe(API_WEB_SOCKET_MESSAGING_TOPIC_URL + "/chats/" + chatId + "/messages", onChatMessageReceived
            }
        }
    }
    setUserChats(currentUserChats);
}
```

Рисунок 3.25 - Підписка на канал повідомлень чату.

```
const onWebSocketConnected = async () => {
    let accessToken = localStorage.getItem("access-token");
    stompClient.subscribe(API_WEB_SOCKET_MESSAGING_TOPIC_URL + "/" + user.username + "/notifications", onUserNotificationReceived
```

Рисунок 3.26 - Підписка на канал сповіщень користувача.

Після успішного з'єднання відбувається розподіл шляхів відповідно до ролей. Користувач з роллю USER взагалі не буде мати доступу до шляхів адміністраторів та не буде навіть підозрювати, що в системі є адміністратор. В той же час адміністратор має доступ майже до всіх шляхів користувача. Він може переглядати його акаунт, блокувати користувача за потреби, переглядати всіх користувачів в системі. Проте адміністратор не має доступу до особистих чатів та переписок користувачів. Таким чином створюється безпека в додатку та не розповсюдження особистих даних.

3.2.2 Налаштування функціоналу для комунікації з серверною частиною

Як вже зазначалось раніше, до проетку була додано бібліотека axios.

Axios - це бібліотека для виконання HTTP-запитів у JavaScript, яка працює на стороні браузера. Axios надає зручний та простий інтерфейс для виконання різноманітних HTTP-запитів, таких як GET, POST, PUT, DELETE і т.д., а також для встановлення заголовків, обробки помилок та багато іншого.

В даному застосунку було створено три клієнта **axios**: **axiosClient**, **authAxiosClient** та **refreshAuthAxiosClient**.

Розберемо призначення кожного з них.

- axiosClient - відповідає для створення запитів до back-end які не потребують авторизації користувача.
- authAxiosClient - потрібен для запитів з авторизацією. Перед відправкою запиту, до його HTTP заголовка додається токен користувача, що дістається з локального сховище браузера.
- refreshAxiosClient - потрібен для поновлення ACCESS токена за допомогою REFRESH токена.

```
export const axiosClient = axios.create({
    baseURL: BASE_URL
});

export const authAxiosClient = axios.create({
    baseURL: BASE_URL
});

export const refreshAuthAxiosClient = axios.create({
    baseURL: BASE_URL
});

authAxiosClient.interceptors.request.use(async (request) => {
    const accessToken = localStorage.getItem("access-token");
    request.headers.Authorization = `Bearer ${accessToken}`;
    return request;
});

refreshAuthAxiosClient.interceptors.request.use((request) => {
    const refreshToken = localStorage.getItem("refresh-token");

    if (refreshToken) {
        request.headers.Authorization = `Bearer ${refreshToken}`;
    }

    return request;
});
```

Рисунок 3.27 - Axios клієнти.

3.2.3 Створення функціоналу для обміну повідомленнями

Для надсилання повідомлень був створений окремий метод. У момент входу в акаунт сервер та клієнт обмінюються публічними ключами. Після цього за допомогою публічних ключів відбувається обмін AES ключем для симетричного шифрування. Клієнт зберігає AES ключ сервера у локальному сховищі і у момент відправки повідомлення шифрує повідомлення за допомогою збереженого ключа.

```
const onSendMessageButtonClick = async () => {
    let chatId = chat.id;
    if (textValue && chatId) {
        let aesKey = localStorage.getItem("aes-key");
        if (aesKey) {
            let rawKey = CryptoJS.enc.Base64.parse(aesKey);
            let encryptedText = CryptoJS.AES.encrypt(textValue, rawKey, { mode: CryptoJS.mode.ECB, padding: CryptoJS.pad.Pkcs7 });
            let encryptedData = encryptedText.ciphertext.toString(CryptoJS.enc.Base64);
            let response = await sendMessageToChat(chatId, encryptedData, MessageType.NEW_MESSAGE);
            let sentMessage = response?.data;
            if (!sentMessage) {
                setInformMessage(`Your message "${textValue}" wasn't updated`);
            }
            setTextValue("");
            setShowEmojiPicker(false);
        }
    }
}
```

Рисунок 3.28 - Налаштування відправки повідомлень.

Розшифровка повідомлень відправлених сервером також відбувається подібним чином з використанням відповідного ключа.

```
let aesKey = localStorage.getItem("aes-key");
if (message?.content) {
    let decryptedText = CryptoJS.AES.decrypt(
        { ciphertext: CryptoJS.enc.Base64.parse(message?.content) },
        CryptoJS.enc.Base64.parse(aesKey),
        { mode: CryptoJS.mode.ECB, padding: CryptoJS.pad.Pkcs7 }
    )
    setMessageContent(decryptedText.toString(CryptoJS.enc.Utf8));
}
```

Рисунок 3.29 - Розшифровка повідомлень.

Для шифрування повідомлень використовується модуль **CryptoJS**. При використанні модуля можна обирати необхідний алгоритм. В даному випадку у якості алгоритму налаштовуємо **AES**.

CryptoJS.enc.Base64.parse перетворює **Base64** рядок на об'єкт типу **WordArray**, який може бути використаний **CryptoJS**.

В результаті розробки клієнтської частини було отримано таку структуру папок:

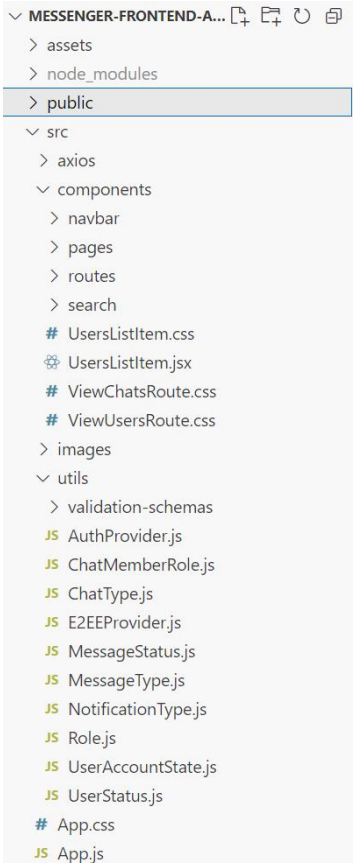


Рисунок 3.30 - Структура front-end проекту.

ВИСНОВОК ДО РОЗДІЛУ 3

У даному розділі дипломної роботи було розроблено програмну частину застосунку. Для коду серверної частини був використаний фреймворк Spring разом з мовою програмування Java, а для клієнтського коду - фреймворк React та мова JavaScript.

Відповідно до потреб на серверній стороні вдалося реалізувати механізми аутентифікації, авторизації та реєстрації користувачів. Для цього було проведено конфігурацію фреймворку Spring Security та налаштовано доступ відповідно до ролей користувачів.

Крім того у застосунку було налаштовано функціонал для шифрування та обміну повідомленнями. Обмін повідомленнями здійснюється з застосуванням протоколу веб-сокетів. Для цього було створено відповідні класи та методи з конфігураціями як на серверній, так і на клієнтській частині з залученням потрібних бібліотек.

Також варто відзначити проведену роботу з налаштування бази даних додатку. Для запуску СКБД було використано технологію контейнеризації на основі Docker. З DockerHub було завантажено потрібний образ та налаштовано підключення на хості. Додатково до серверної частини було підключено бібліотеку Flyway для ефективного керування SQL скриптами та для пришвидшення розробки.

Як результат було отримано повністю реалізований відповідно до задуму програмний продукт з клієнт-серверною архітектурою. За кожну окрему функцію у додатку відповідають свої компоненти та сервіси. Сервіси серверної частини реалізовані таким чином, що вони розширяються інтерфейси або ж абстрактні класи, що дозволяє масштабувати платформу та легко додавати до неї новий функціонал.

РОЗДІЛ 4.
ТЕСТУВАННЯ РОБОТИ ЗАСТОСУНКУ

Тестування та аналіз роботи програмного продукту є одним з найважливіших етапів розробки застосунку. Розглянемо основні аспекти чому варто проводити тестування.

- Виявлення помилок: Тестування допомагає виявити помилки, які можуть призвести до некоректної роботи програми або її аварійного завершення. Чим раніше виявляться помилки, тим легше їх виправити.
- Забезпечення якості: Проведення тестування дозволяє забезпечити високу якість програмного продукту. Якісне програмне забезпечення важливе для задоволення потреб користувачів і уникнення негативних наслідків від помилок.
- Перевірка відповідності вимогам: Тестування допомагає перевірити, чи відповідає розроблений застосунок вимогам, визначеним на початковому етапі розробки. Це дозволяє впевнитися, що програма відповідає потребам користувачів і бізнес-вимогам.
- Оптимізація продуктивності: Тестування допомагає виявити та виправити проблеми з продуктивністю програми, такі як довгий час відгуку або велика кількість споживаної пам'яті. Це дозволяє забезпечити оптимальну роботу застосунку.

4.1 Огляд функціоналу

Проведемо огляд функціоналу застосунку. Оскільки застосунок поділяється на front-end та back-end частини, то дослідження роботи можна проводити окремо для кожної частини або ж тестувати взаємодію всіх сервісів разом.

Для перевірки роботи серверної частини, до проекту було додано бібліотеку Swagger [39, 40].

Swagger - це набір інструментів для розробки, документування і використання веб-сервісів. Він включає у себе кілька компонентів, які спільно допомагають створити специфікацію API [41, 42] і зробити її доступною для розробників інших служб або клієнтських додатків.

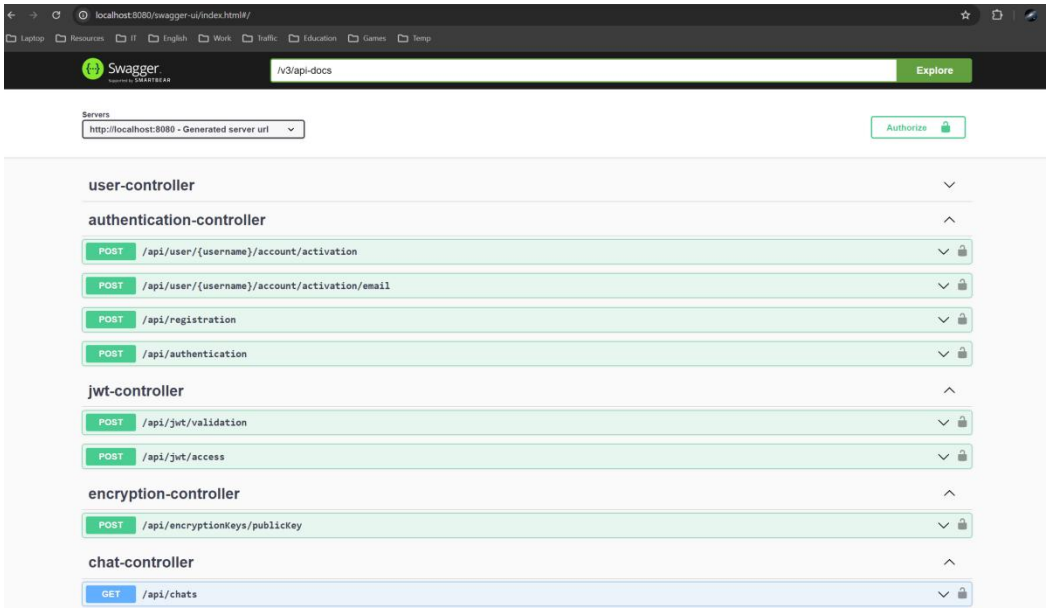


Рисунок 4.1 - Вигляд панелі Swagger для тестування back-end.

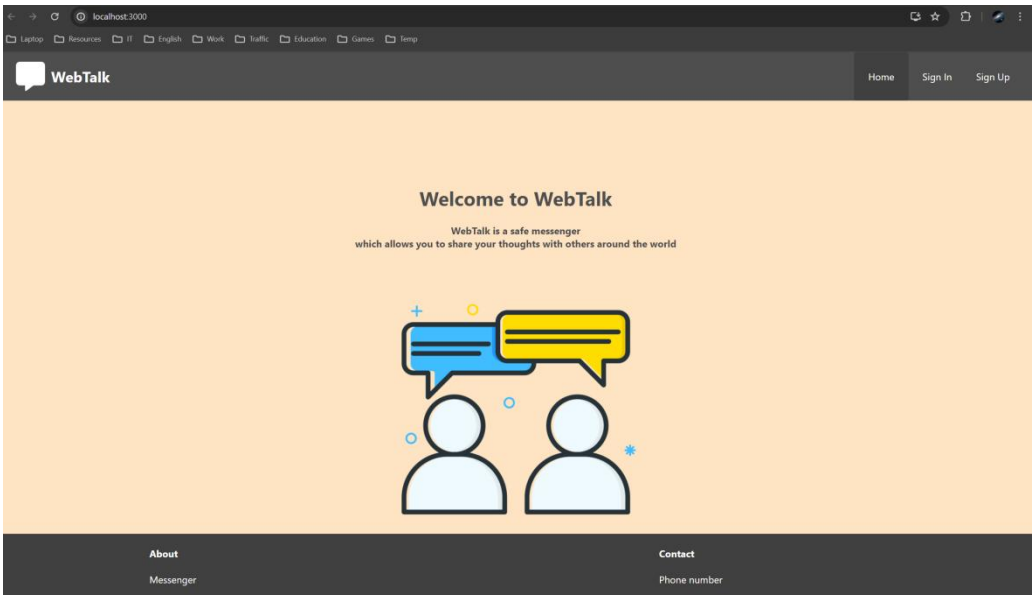


Рисунок 4.2 - Вигляд головної сторінки front-end застосунку.

4.1.1 Реєстрація та автентифікація

Розглянемо процес реєстрації. Для цього в шапці застосунку перейдемо на вкладку **Sign Up** та заповнимо відповідні поля необхідні для реєстрації.

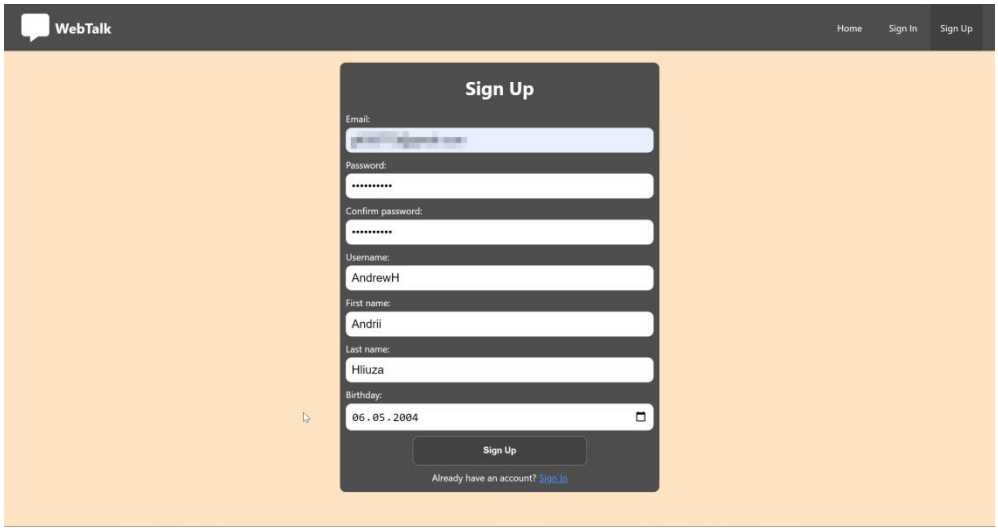


Рисунок 4.3 - Форма реєстрації.

Якщо спробувати невірно ввести якісь з полів то отримаємо попередження.

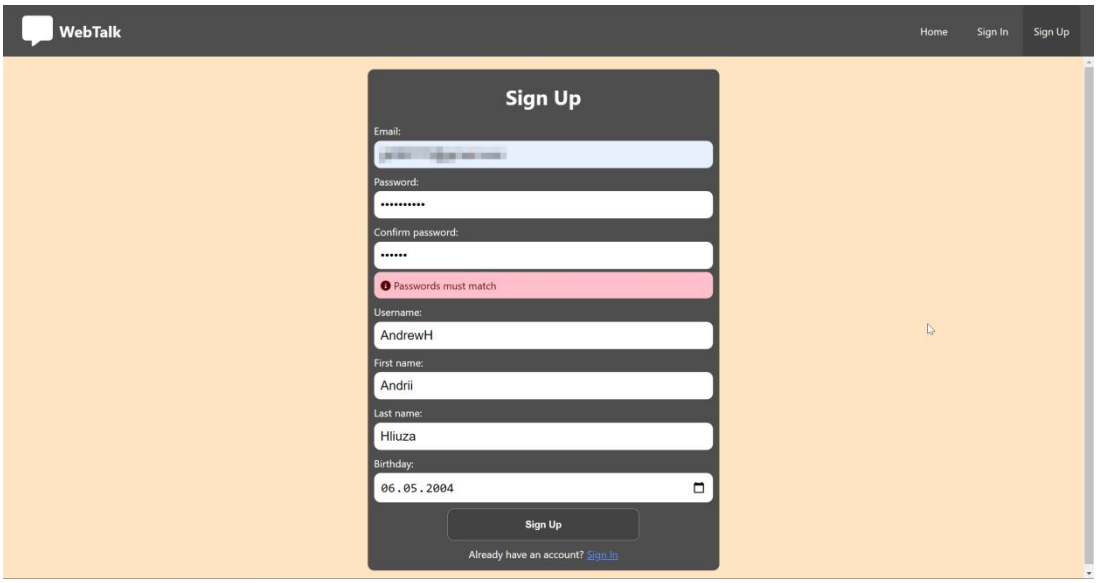


Рисунок 4.4 - Попередження у разі невірного заповнення полів форми.

Після правильного заповнення всіх полів та натисненні кнопки **Sign Up** на пошту користувачу повинно прийти повідомлення з кодом для активації акаунта.

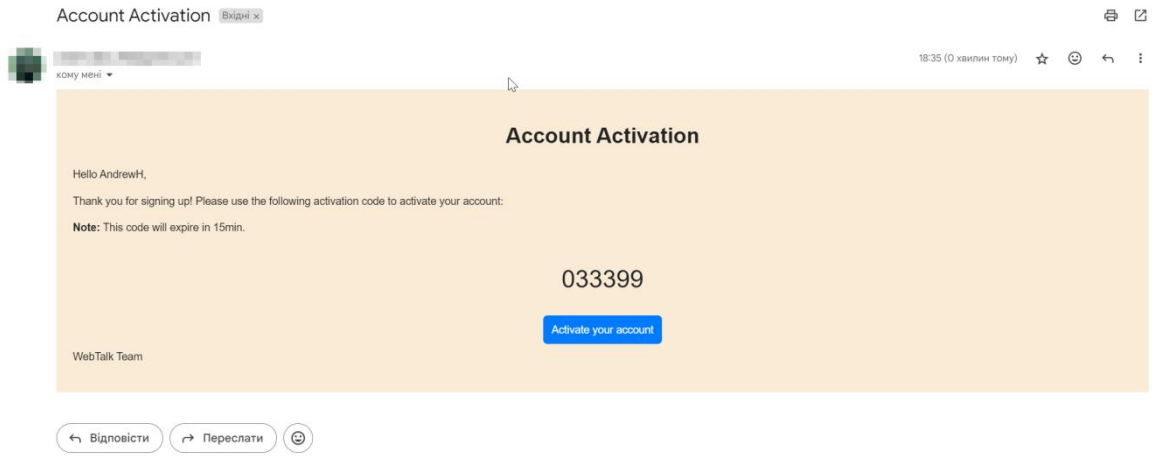


Рисунок 4.5 - Лист для підтвердження акаунту користувача.

Для активації потрібно натиснути на кнопку **Activate your account** та в поле вводу ввести отриманий код. Варто зазначити, що код дійсний лише протягом 15 хвилин. Якщо термін дії коду вичерпався, користувачу потрібно заново пройти процес реєстрації.

Після успішної активації акаунта користувач зможе автентифікуватись в застосунку. Для цього переходимо на вкладку **Sign In** та вводимо email та пароль, що були використані при реєстрації в форму.

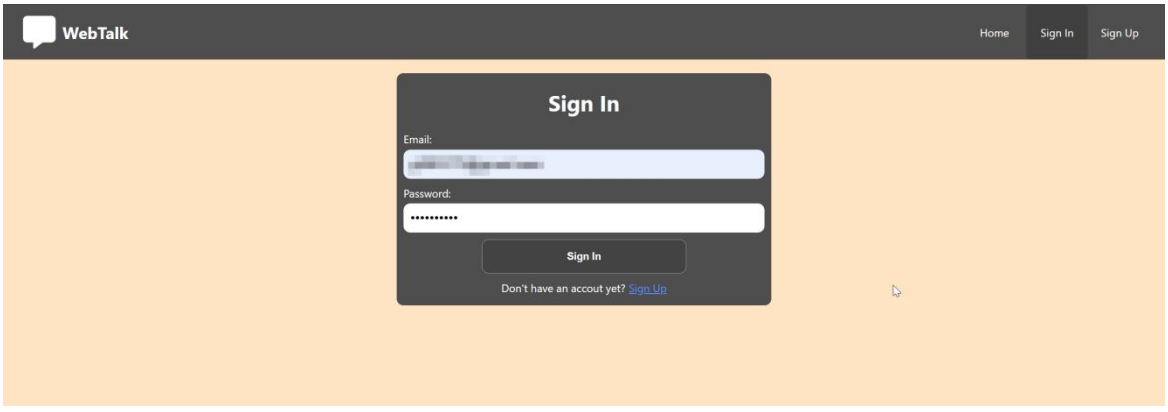


Рисунок 4.6 - Процес входу до акаунту.

4.1.2 Сторінка користувача

Одразу після реєстрації та аутентифікації користувач потрапляє на свою сторінку.

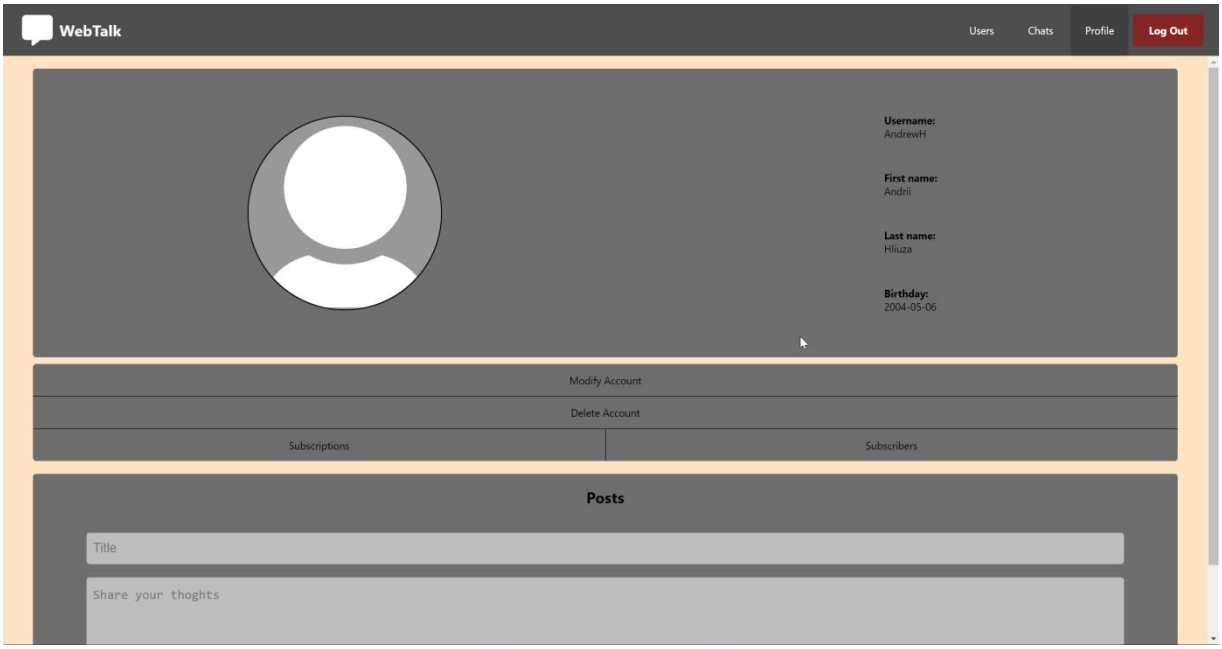


Рисунок 4.7 - Акаунт користувача.

Користувач може робити декілька операцій з власною сторінкою. Розберемо їх детальніше.

По-перше на сторінці є функції для модифікації акаунту. Натиснувши на відповідну кнопку, можна змінити дані, що вводив користувач при реєстрації, такі як: username, ім’я, прізвище, дату народження та пароль. Також можна додати фото. Варто зазначити, що для виконання операцій пов’язаних з модифікацією акаунта потрібно буде ввести поточний пароль користувача.

Також, за бажання, користувач може видалити акаунт. Варто зазначити, що операція видалення є незворотною. Після видалення всі дані користувача моментально видаляються з бази даних, а отже відновити їх буде неможливо.

При видаленні, з’являється вікно у якому користувача ще раз запитують чи бажає він здійснити цю операцію. У цьому вікні потрібно остаточно підтвердити або ж відмінити поточну операцію.

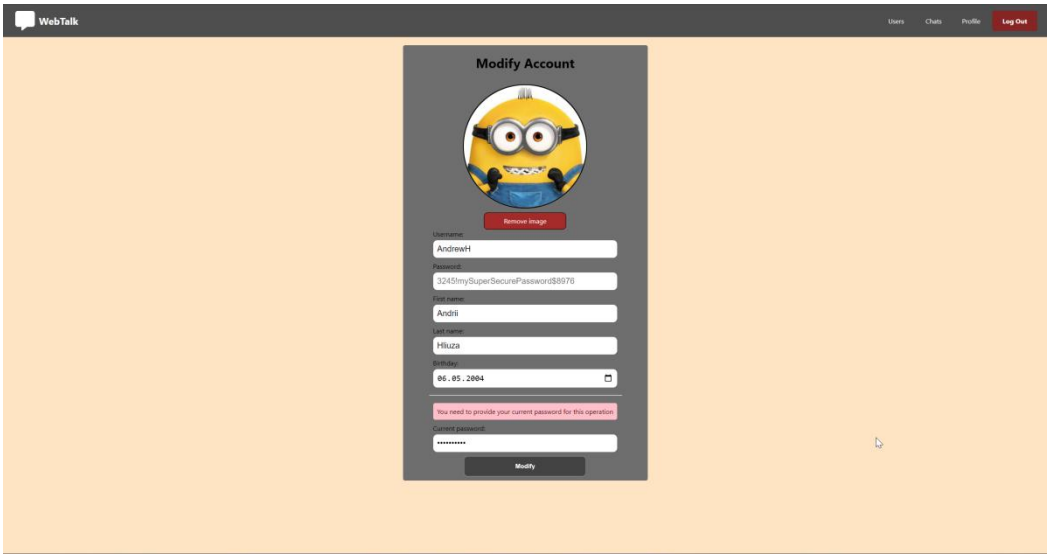


Рисунок 4.8 - Вікно модифікації сторінки.

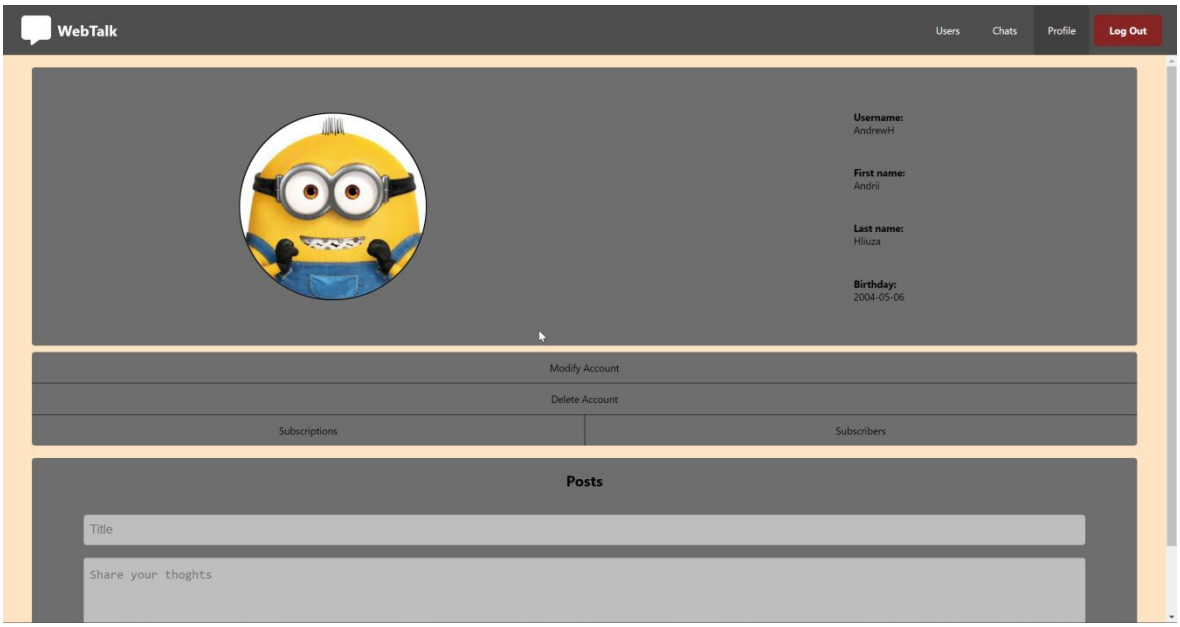


Рисунок 4.9 - Сторінка після модифікації.

Іншими функціями доступними на сторінці є можливість перегляду підписок на інші акаунти та підписок на акаунт користувача. За це відповідають відповідно кнопки **Subscriptions** та **Subscribers**. Натиснувши на будь-яку з цим кнопок відкриється список, де можна буде побачити акаунти інших користувачів та за бажанням перейти на їхні сторінки.

Крім того, для того щоб побачити взагалі всіх користувачів в системі, можна скористатись вкладкою **Users** в шапці веб-сайту.

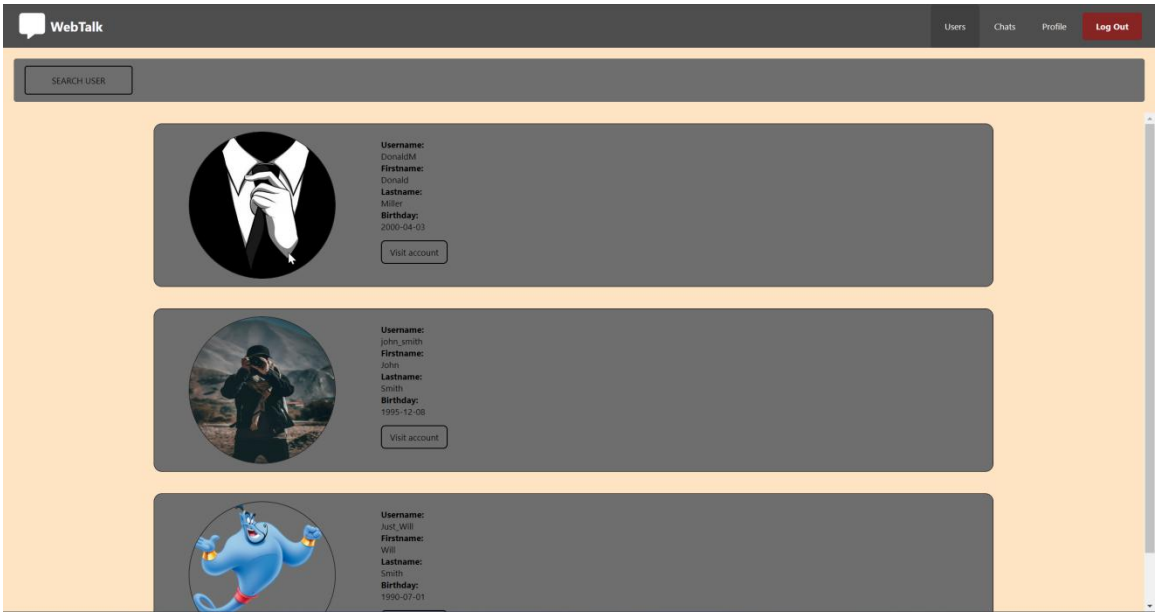


Рисунок 4.10 - Сторінка перегляду користувачів застосунку.

Щоб знайти певного користувача можна натиснути на кнопку пошуку згори сторінки. Після цього відкриється вікно куди можна вписати ім’я потрібного користувача на платформі.

Натиснувши на кнопку **Visit account** відкриється сторінка відповідного користувача. На сторінці іншого користувача можна здійснювати декілька операцій.

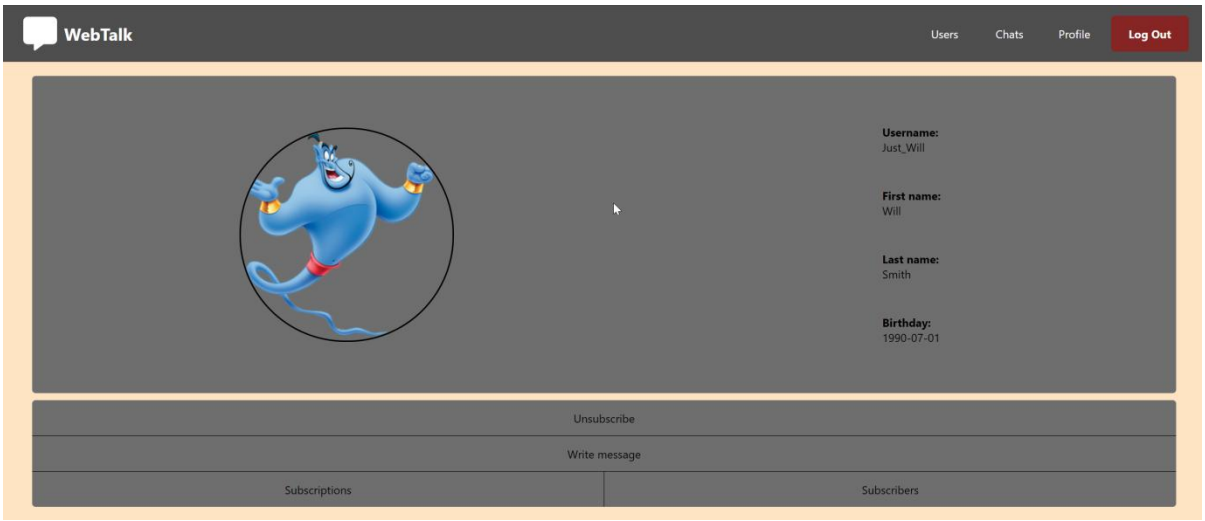


Рисунок 4.11 - Сторінка іншого користувача в системі.

По-перше можна підписатись на користувача натиснувши на кнопку **Subscribe**. Після цього користувачу на сторінку якого було оформлено підписку прийде сповіщення про нового підписника.

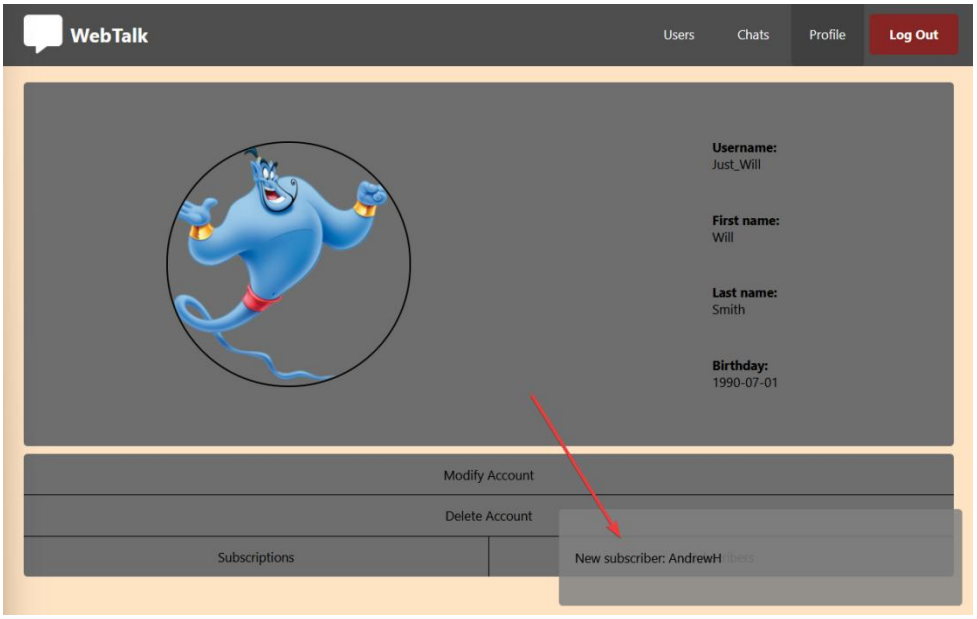


Рисунок 4.12 - Тестування функції підписки.

Іншою можливістю на сторінці користувача є написати йому приватне повідомлення. Щоб зробити це потрібно натиснути кнопку **Write message** в результаті чого відкриється вікно чату.

4.1.3 Обмін повідомленнями в застосунку

Для обміну повідомленнями потрібно створити чат. В застосунку існує два види чатів: приватні та групові.

Приватні чати створюються в момент натискання на кнопку **Write message** на сторінці користувача та можуть містити лише двох користувачів.

На відміну від приватних чатів, групові чати можуть мітити два та більше учасників. Щоб створити груповий чат потрібно перейти на вкладку Chats та натиснути на кнопку **CREATE CHAT**. У вікні, що відкриється можна задати ім’я чату та додати учасників.

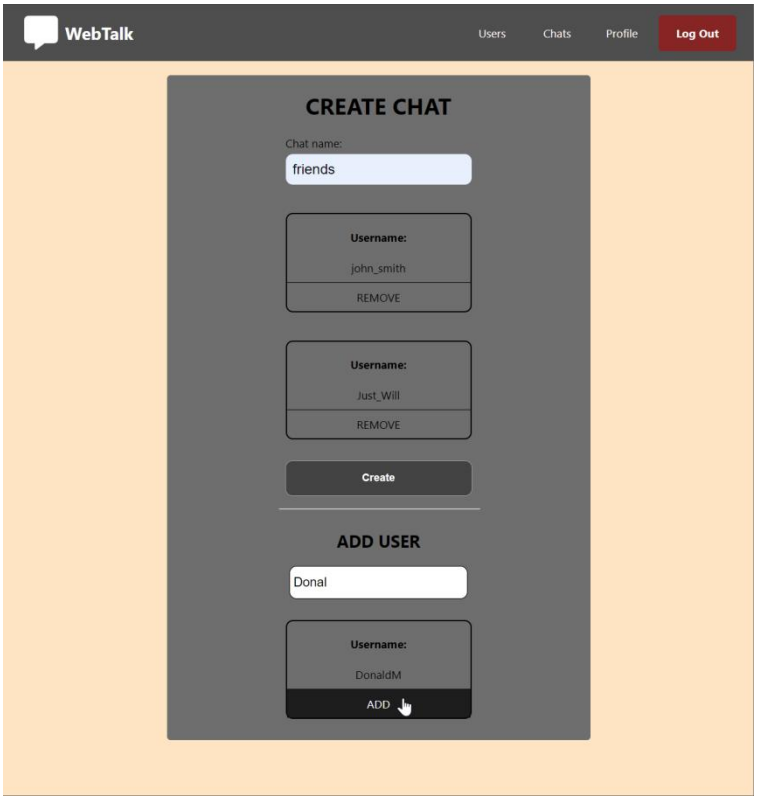


Рисунок 4.13 - Вікно створення чату.

Після натискання кнопки **Create** буде створений новий чат, який можна буде побачити перейшовши на вкладку **Chats**.

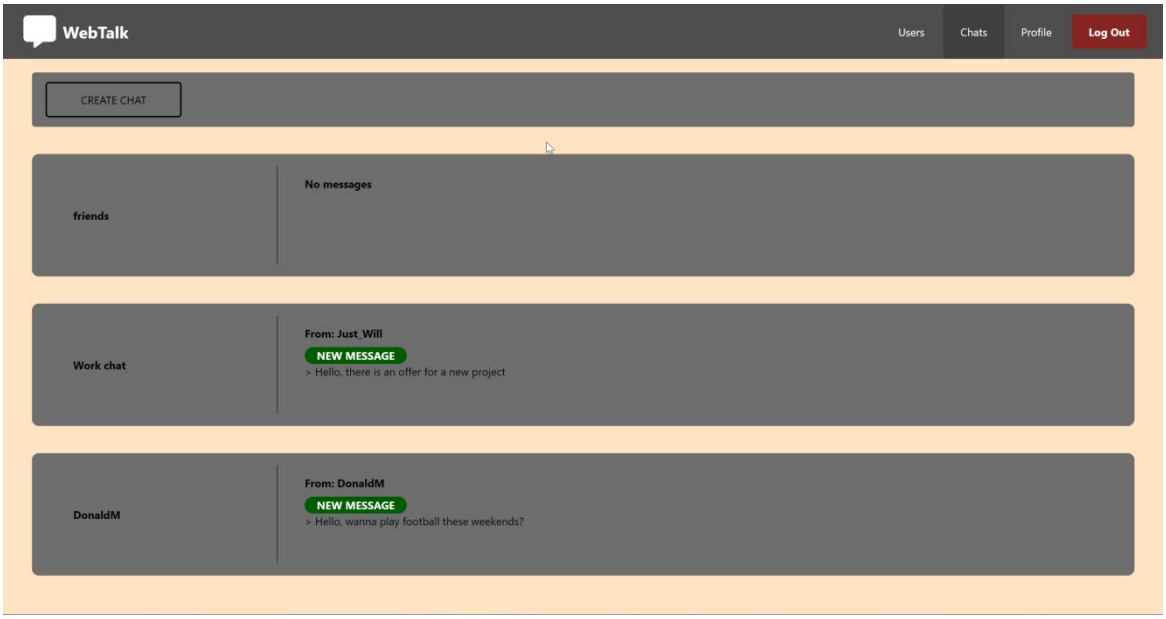


Рисунок 4.14 - Список чатів користувача.

В списку чатів можна побачити, що деякі чати мають непрочитані повідомлення від інших учасників. Натискання на конкретний чат призведе до переходу на сторінку цього чату.

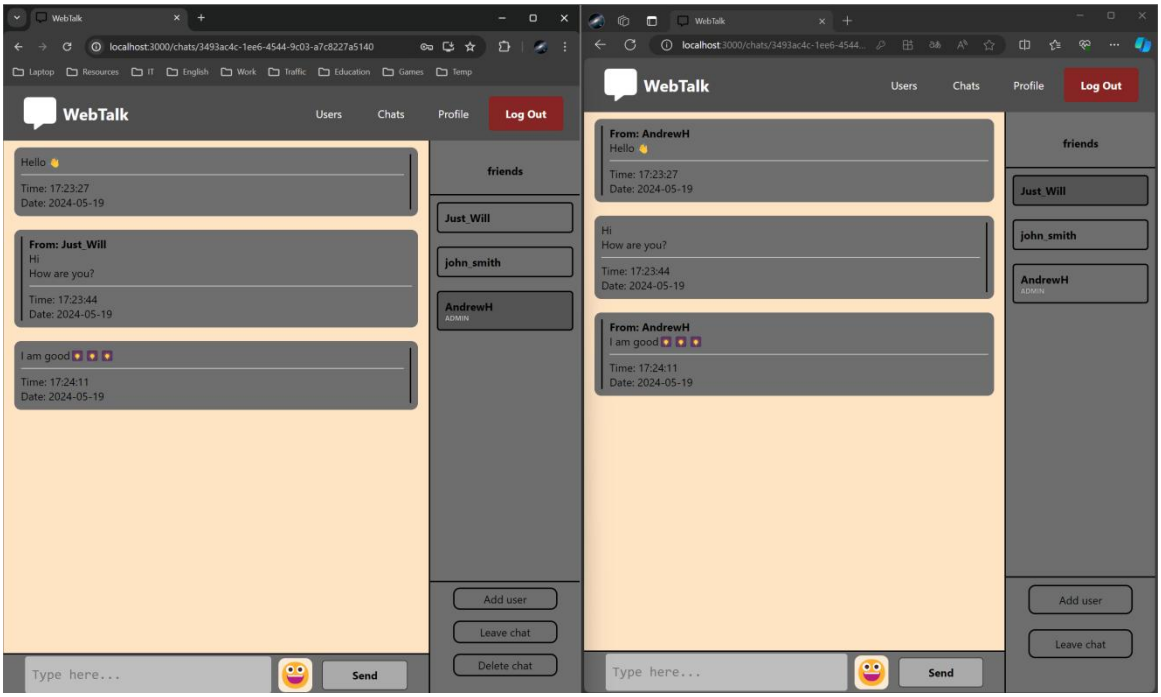


Рисунок 4.15 - Сторінка чату.

На сторінці можна побачити попередню переписку та можна відправити нове повідомлення.

Також власник повідомлення може редагувати, видаляти своє повідомлення, а також переглянути стан повідомлення. У повідомлення є два стани в якому воно може знаходитись: прочитане (**READ**) та непрочитане (**UNREAD**). Прочитаним повідомлення становиться у той момент коли його переглянуть інші користувачі. Крім того на повідомленні відображається час коли воно було відправлене та хто його відправив.

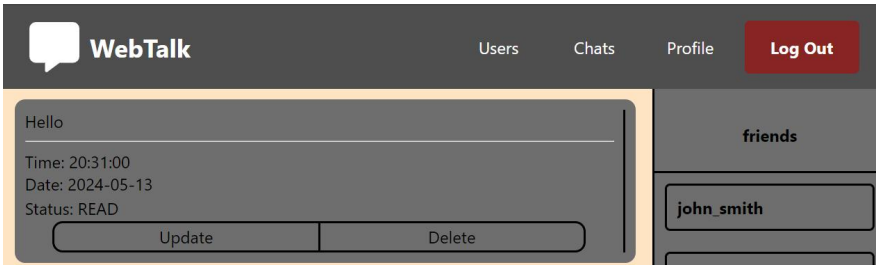


Рисунок 4.16 - Повідомлення в чаті.

Учасники чату поділяються на звичайних користувачів (**USER**) та адміністраторів (**ADMIN**). У будь-якого користувача в чаті є право додавати інших учасників, в той час як право видалення є лише у адміністраторів чату. Коли чат тільки створився, він містить лише одного адміністратора - це людина, яка створила чат. Щоб додати нового адміністратора, будь-який інший адміністратор може натиснути на довільному учаснику правою кнопкою миші та в меню, що відкриється вибрати опцію **Make ADMIN**.

Рисунок 4.17 - Додавання адміністратора до чату.

Також адміністратор чату може змінювати його назву.

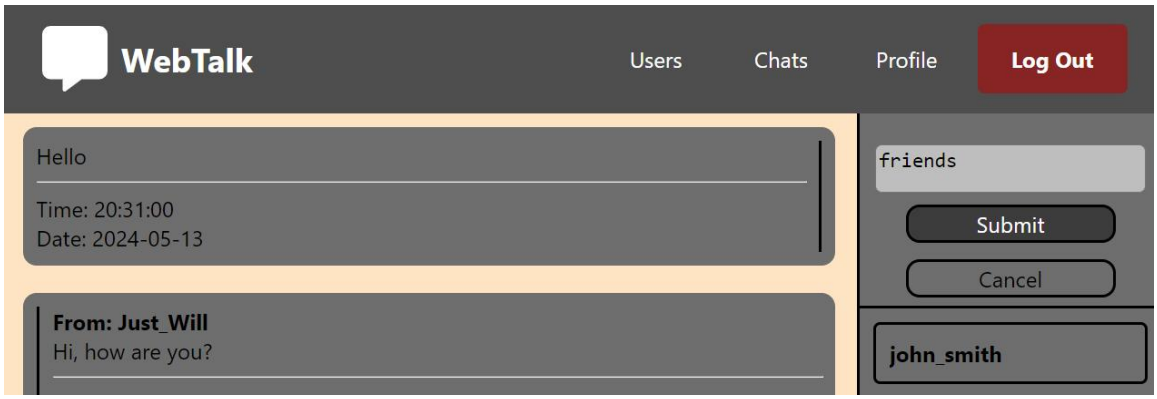


Рисунок 4.18 - Зміна назви чату.

Останньою функцією в чаті є можливість видалення чату. При видаленні чату звичайним учасником, чат видаляється лише для нього. На відміну від звичайного користувача, в адміністратора є можливість видалити чат для всіх учасників або ж видалити лише для себе. За це відповідають кнопки **Delete chat** та **Leave chat**.

4.1.4 Панель адміністратора

Окрім звичайних користувачів, в системі також присутні адміністратори, які мають ряд додаткових функцій. З замовчуванням, при запуску застосунку автоматично створюється два адміністратори: з роллю **ADMIN** та роллю **ROOT**.

Рисунок 4.19 - Панель адміністратора.

Адміністратор, як і звичайний користувач може модифікувати та видаляти власний акаунт. Окрім цього у нього також є функції для блокування та видалення акаунтів звичайних користувачів, а також можливість створення ще одного адміністратора в системі.

У разі блокування акаунта користувача, в базі даних його стан змінюється з ACTIVATED на BLOCKED. Такий акаунт за потреби може бути знову активований адміністратором. При виборі опції для видалення акаунта, всі дані користувача повністю видаляються з бази даних.

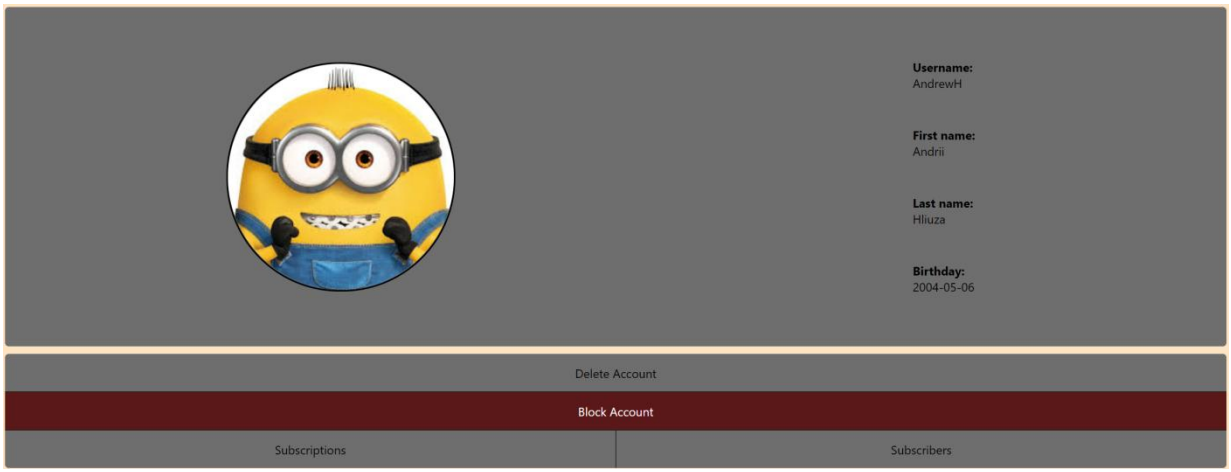


Рисунок 4.20 - Вигляд сторінки звичайного користувача для адміністратора.

Щоб переглянути всіх адміністраторів в системі, потрібно перейти на вкладку **Admins**.

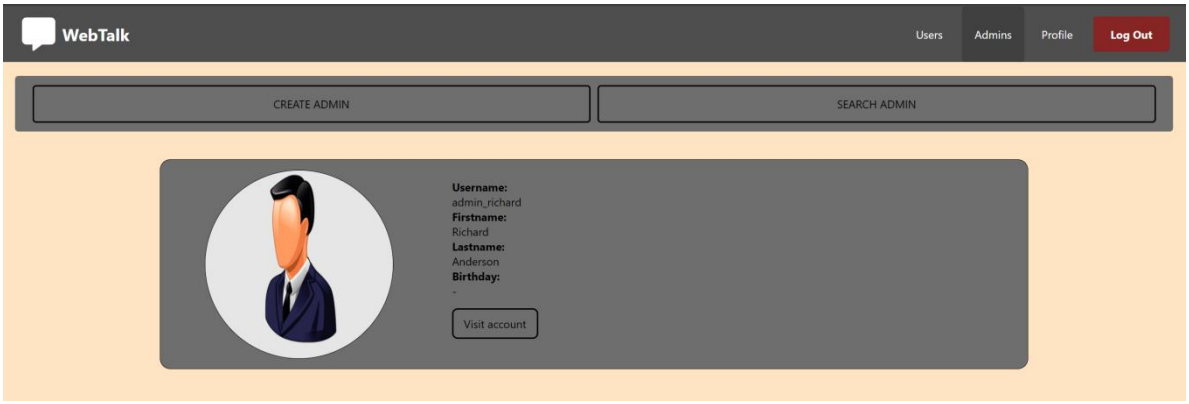


Рисунок 4.21 - Список адміністраторів в системі.

Варто зазначити, що адміністратор з роллю **ROOT** в цьому списку не відображається. Він є адміністратором верхнього рівня, до якого ніхто не має доступу. Такий користувач має всі ті ж права, що і звичайні адміни, але додатково у нього з'являється можливість видаляти інших адміністраторів в системі.

Такий підхід до адміністрування, створює додатковий шар, що дозволяє більш ефективно керувати платформою.

4.2 Розвиток платформи в майбутньому

Наразі веб-застосунок має базову функціональність необхідну для комфортного користування. За бажання, в майбутньому функціонал системи можна розширювати.

По-перше окрім чатів, можна додати підтримку каналів, що будуть публікувати інформацію на певну тематику. Таким чином користувачі зможуть підписуватись на цікаві їм ресурси та слідкувати за новинами спорту, культури та ін.

По-друге, можна поліпшити механізми шифрування даних, створювати нові алгоритми та комбінувати старі. Це дозволить зробити систему безпечнішою та підвищить довіру кінцевого користувача до застосунку.

Додатково, можна покращити механізми адміністрування шляхом введення нових ролей в систему та використання ефективних алгоритмів для автоматичного виявлення та блокування небажаного контенту.

Загалом, архітектура системи побудована таким чином, щоб в майбутньому була можливість додавати нові можливості та функції до платформи, що дозволять покращити користувацький досвід.

4.3 Використання застосунку у військовій сфері

Під час військових дій, ситуація на фронті може досить швидко змінюватись, а тому важливо забезпечувати ефективну координацію між військовими підрозділами. Своєчасно отримана інформація є запорукою гарної оборони та наступу окремих військових угрупувань, адже допомагає відслідковувати рух противника на полі бою. В деяких випадках отримання важливої інформації в потрібний час може навіть зберегти життя.

В той же час важливо також забезпечувати секретність будь-якої інформації під час її передачі між військовими, адже однією з найголовніших задач ворога є виявлення слабких місць та відслідковування різного роду витоків даних. Саме для таких випадків і існують різні шифровані канали зв'язку, в тому числі месенджери з підтримкою шифрування даних. Незважаючи, на те, що наразі існує багато рішень в сфері захищеного спілкування, проте всі подібні застосунки належать компаніям інших країн, а отже немає гарантії того, як саме дані користувачів обробляються та в якому вигляді вони зберігаються на серверах цих компаній, а головне - невідомо хто має доступ до таких даних. Саме тому в даному дипломному проекті було розроблено вітчизняний застосунок для спілкування з використанням технологій шифрування даних.

Під час процесу обміну даними застосовується комбінований підхід, що поєднує технологій симетричного та асиметричного типів шифрування. Використання асиметричних ключів дозволяє створити захищений канал, через який можна передавати вразливу інформацію, по типу симетричних ключів, що застосовуються для шифрування кінцевих даних. Загалом використання симетричного ключа для шифрування кінцевої інформації також пришвидшує роботу застосунку, адже процес симетричного шифрування загалом простіший за асиметричний метод.

Додатково, варто зазначити, що шифрування в застосунку побудовано

таким чином, що ключі оновлюються при кожному новому вході в акаунт, а отже навіть при їх втраті, або ж перехваті противником, достатньо буде перезайти в акаунт для генерації нової пари відповідних ключів.

Підсумовуючи, варто сказати, що хоч застосунок наразі і не має деякої кількості додаткових функцій, що мають зарубіжні аналоги, проте він забезпечує найголовнішу функціональність, а саме: створення незалежного та безпечного каналу для спілкування. А отже, маючи фундамент, в подальшому буде не складно покращувати сервіс шляхом додавання нових функцій.

ВИСНОВОК ДО РОЗДІЛУ 4

У даному розділі дипломного проєкту було протестовано роботу системи та визначено напрям для розвитку її в майбутньому.

В ході тестування було розглянуто механізми реєстрації та автентифікації користувачів. Для підтвердження особи, що реєструється використовувалась електронна пошта на яку відправлявся код для активації акаунту.

Окрім того, було розглянуто функціонал звичайного користувача та можливості панелі адміністратора. Окрему увагу було приділену механізму обміну повідомленнями. Для обміну повідомленнями в системі існує два підходи: через приватні або групові чати. Для керування чатами існує окрема роль адміністратора чату.

Як результат проведеного дослідження, можна сказати, що функціонал застосунку відповідає всім поставленим під час написання коду умовам.

Система показала себе, як надійний та безпечний спосіб для спілкування. Завдяки гарно написаному коду, продуктивність застосунку теж залишалась на високому рівні. Веб-застосунок працював швидко при одночасному доступі до нього кількох користувачів та попри додаткову потребу у шифруванні повідомлень при передачі.

В кінці було розглянуто перспективи для вдосконалення застосунку та функціонал, який за бажання можна додати до платформи, атакож було досліджено можливості використання сервісу для військових цілей.

ВИСНОВКИ

Бакалаврська дипломна робота мала на меті розробку вітчизняного аналогу захищеного застосунку для спілкування.

У ході виконання проєкту було проаналізовано схожі рішення на світовому ринку та наведено статистичні дані, що підтверджують актуальність розробки. Зокрема дослідження функціоналу існуючих платформ та їх особливостей дозволило визначити набір функцій необхідних застосунку.

Наступний етап полягав у виборі необхідних для реалізації програми технологій. Оскільки на сьогодні існує багато підходів до створення продукту, то головною метою на цьому етапі було визначити набір оптимальних рішень для розробки. Переглянувши декілька варіантів технологій та оцінивши їх плюси та мінуси, було вирішено для серверної частини використовувати фреймворк Spring та мову Java, а для клієтської - фреймворк ReactJS та мову JavaScript.

Шифрування інформації в застосунку здійснювалось за допомогою поєднання синхронного та асинхронного методів з залученням алгоритмів AES та RSA. Для цього на стороні back-end частини було використано стандартні засоби мови програмування Java та для front-end підключено сторонню бібліотеку. Як результат, отримали повністю функціональну систему для обміну повідомленнями.

В останньому розділі, було проведено тестування роботи системи. Застосунок показав себе як надійне рішення з широким функціоналом. У ході тестування було перевірено стійкість системи до навантаження шляхом одночасного перебування на платформі декількох користувачів. Система показала себе добре. Інтерфейс користувача є інтуїтивно зрозумілим, а система загалом є досить швидкою.

Технічне завдання на дипломний бакалаврський проект виконано повністю.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Most Popular Apps (2024). Business of Apps. [Електронний ресурс] - https://www.businessofapps.com/data/most-popular-apps/ (дата звернення: 17.03.2024).

2. Most popular messaging apps 2024 | Statista. [Електронний ресурс] - https://www.statista.com/statistics/258749/most-popular-global-mobile-messenger-apps/ (дата звернення: 18.03.2024).

3. Рейтинг мобільних додатків за 2023 рік. ТОП популярних в Україні. Webpromo. [Електронний ресурс] - https://web-promo.ua/ua/blog/rejting-populyarnyh-mobilnyh-prilozhenij-ukrainy-v-2021-godu/ (дата звернення: 18.03.2024).

4. Similarweb. Most Popular Messaging Apps Worldwide 2023. [Електронний ресурс] - Режим доступу до ресурсу: https://www.similarweb.com/blog/research/market-research/worldwide-messaging-apps/ (дата звернення: 19.03.2024).

5. WebSockets: навіщо потрібні та як з ними працювати. [Електронний ресурс] - Режим доступу до ресурсу: https://proit.org.ua/websockets-navishcho-potribni-ta-iak-z-nimi-pratsiuvati-2/ (дата звернення: 20.03.2024).

6. WebSocket API and protocol explained: How they work, are used and more. Ably Realtime. [Електронний ресурс] - Режим доступу до ресурсу: https://ably.com/topic/websockets (дата звернення: 20.03.2024).

7. What Is Pull Technology? | Website Builders.com. [Електронний ресурс] - Режим доступу до ресурсу: https://websitebuilders.com/how-to/glossary/pull/ (дата звернення: 21.03.2024).

8. What is Push Technology? | Medium. [Електронний ресурс] - Режим доступу до ресурсу: https://medium.com/@webcastingandvirtualevents/what-is-push-technology-fc2af3c0d28c (дата звернення: 21.03.2024).

9. What is encryption? - Cloudflare [Електронний ресурс] - Режим доступу до ресурсу: https://www.cloudflare.com/learning/ssl/what-is-encryption/ (дата звернення: 24.03.2024).
10. Symmetric Key Cryptography - GeeksforGeeks. [Електронний ресурс] - Режим доступу до ресурсу: https://www.geeksforgeeks.org/symmetric-key-cryptography/ (дата звернення: 25.03.2024).
11. Asymmetric Encryption: Definition, Architecture, Usage. [Електронний ресурс] - Режим доступу до ресурсу: https://www.okta.com/identity-101/asymmetric-encryption/ (дата звернення: 27.03.2024).
12. What is Asymmetric Encryption? - GeeksforGeeks. [Електронний ресурс] - Режим доступу до ресурсу: https://www.geeksforgeeks.org/what-is-asymmetric-encryption/ (дата звернення: 28.03.2024).
13. How does public key cryptography work? – Cloudflare. [Електронний ресурс] - Режим доступу до ресурсу: https://www.cloudflare.com/learning/ssl/how-does-public-key-encryption-work/ (дата звернення: 02.04.2024).
14. What is End-to-End Encryption (E2EE) and How Does it Work? [Електронний ресурс] - Режим доступу до ресурсу: https://www.techtarget.com/searchsecurity/definition/end-to-end-encryption-E2EE (дата звернення: 05.04.2024).
15. Наскрізне шифрування – Вікіпедія. [Електронний ресурс] - Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Наскрізне шифрування (дата звернення: 09.04.2024).
16. End-to-End Encryption: The Ultimate Guide to How it Works. [Електронний ресурс] - Режим доступу до ресурсу: https://www.preveil.com/blog/end-to-end-encryption/ (дата звернення: 09.04.2024).
17. Docker Documentation. [Електронний ресурс] - Режим доступу до ресурсу: https://docs.docker.com/ (дата звернення: 09.04.2024).
18. Introducing JSON. [Електронний ресурс] - Режим доступу до ресурсу: https://www.json.org/json-en.html (дата звернення: 10.04.2024).

19. XML introduction - XML: Extensible Markup Language | MDN. MDN Web Docs. [Електронний ресурс] - Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Web/XML/XML_introduction (дата звернення: 10.04.2024).

20. React Reference Overview – React. [Електронний ресурс] - Режим доступу до ресурсу: https://react.dev/reference/react (дата звернення: 10.04.2024).

21. What is Java? - Java Programming Language Explained - AWS. Amazon Web Services, Inc. [Електронний ресурс] - Режим доступу до ресурсу: https://aws.amazon.com/what-is/java/?nc1=h_ls (дата звернення: 11.04.2024).

22. Overview (Java SE 17 & JDK 17). [Електронний ресурс] - Режим доступу до ресурсу: https://docs.oracle.com/en/java/javase/17/docs/api/ (дата звернення: 12.04.2024).

23. Schildt H. Java: The Complete Reference, Tenth Edition (Complete Reference Series). McGraw-Hill Education, 2017. 1500 с.

24. Horstmann C. S. Core Java Volume I--Fundamentals (11th Edition). Prentice Hall, 2018. 928 с.

25. Spring Framework Documentation :: Spring Framework. Spring | Home. [Електронний ресурс] - Режим доступу до ресурсу: https://docs.spring.io/spring-framework/reference/ (дата звернення: 13.04.2024).

26. Key Differences Between Spring vs Spring Boot vs Spring MVC | Simplilearn [Електронний ресурс] - Режим доступу до ресурсу: https://www.simplilearn.com/tutorials/spring-boot-tutorial/spring-vs-spring-boot (дата звернення: 15.04.2024).

27. Spring Security :: Spring Security. Spring | Home. [Електронний ресурс] - Режим доступу до ресурсу: https://docs.spring.io/spring-security/reference/ (дата звернення: 16.04.2024).

28. Introduction to Java Config for Spring Security | Baeldung. [Електронний ресурс] - Режим доступу до ресурсу: https://www.baeldung.com/java-config-spring-security (дата звернення: 18.04.2024).

29. Spring Boot Reference Documentation. Spring | Home. [Електронний ресурс] - Режим доступу до ресурсу: https://docs.spring.io/spring-boot/docs/current/reference/html/ (дата звернення: 18.04.2024).
30. Your relational data. Objectively. - Hibernate ORM. Hibernate. [Електронний ресурс] - Режим доступу до ресурсу: https://hibernate.org/orm/ (дата звернення: 19.04.2024).
31. Spring Transaction Management: @Transactional In-Depth. [Електронний ресурс] - Режим доступу до ресурсу: https://www.marcobehler.com/guides/spring-transaction-management-transactional-in-depth (дата звернення: 20.04.2024).
32. Transactions with Spring and JPA | Baeldung. [Електронний ресурс] - Режим доступу до ресурсу: https://www.baeldung.com/transaction-configuration-with-jpa-and-spring (дата звернення: 20.04.2024).
33. Introduction to JSON Web Tokens. [Електронний ресурс] - Режимдоступудо ресурсу: https://jwt.io/introduction (дата звернення: 24.04.2024).
34. Spring Boot API Security with JWT and Rule-Based Authorization [Електронний ресурс] - Режим доступу до ресурсу: https://medium.com/@akhileshanand/spring-boot-api-security-with-jwt-and-role-based-authorization-fea1fd7c9e32 (дата звернення: 28.04.2024).
35. PostgreSQL Documentation. [Електронний ресурс] - Режим доступу до ресурсу: https://www.postgresql.org/docs/ (дата звернення: 02.05.2024).
36. Database Normalization [Електронний ресурс] - Режим доступу до ресурсу: https://www.freecodecamp.org/news/database-normalization-1nf-2nf-3nf-table-examples/ (дата звернення: 02.05.2024).
37. Using Spring Boot for WebSocket Implementation with STOMP [Електронний ресурс] - Режим доступу до ресурсу: https://www.toptal.com/java/stomp-spring-boot-websocket (дата звернення: 02.05.2024).
38. Axios API. [Електронний ресурс] - Режим доступу до ресурсу: https://axios-http.com/docs/api_intro (дата звернення: 02.05.2024).

39. API Documentation & Design Tools for Teams | Swagger. [Електронний ресурс] - Режим доступу до ресурсу: https://swagger.io/ (дата звернення: 08.05.2024).

40. Що таке Swagger? [Електронний ресурс] - Режим доступу до ресурсу: https://qagroup.com.ua/publications/what-is-swagger/ (дата звернення: 12.05.2024).

41. Setting Up Swagger | Baeldung. [Електронний ресурс] - Режим доступу до ресурсу: https://www.baeldung.com/swagger-2-documentation-for-spring-rest-api (дата звернення: 14.05.2024).

42. OpenAPI – Вікіпедія. [Електронний ресурс] - Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/OpenAPI (дата звернення: 20.05.2024).

ДОДАТОК 1

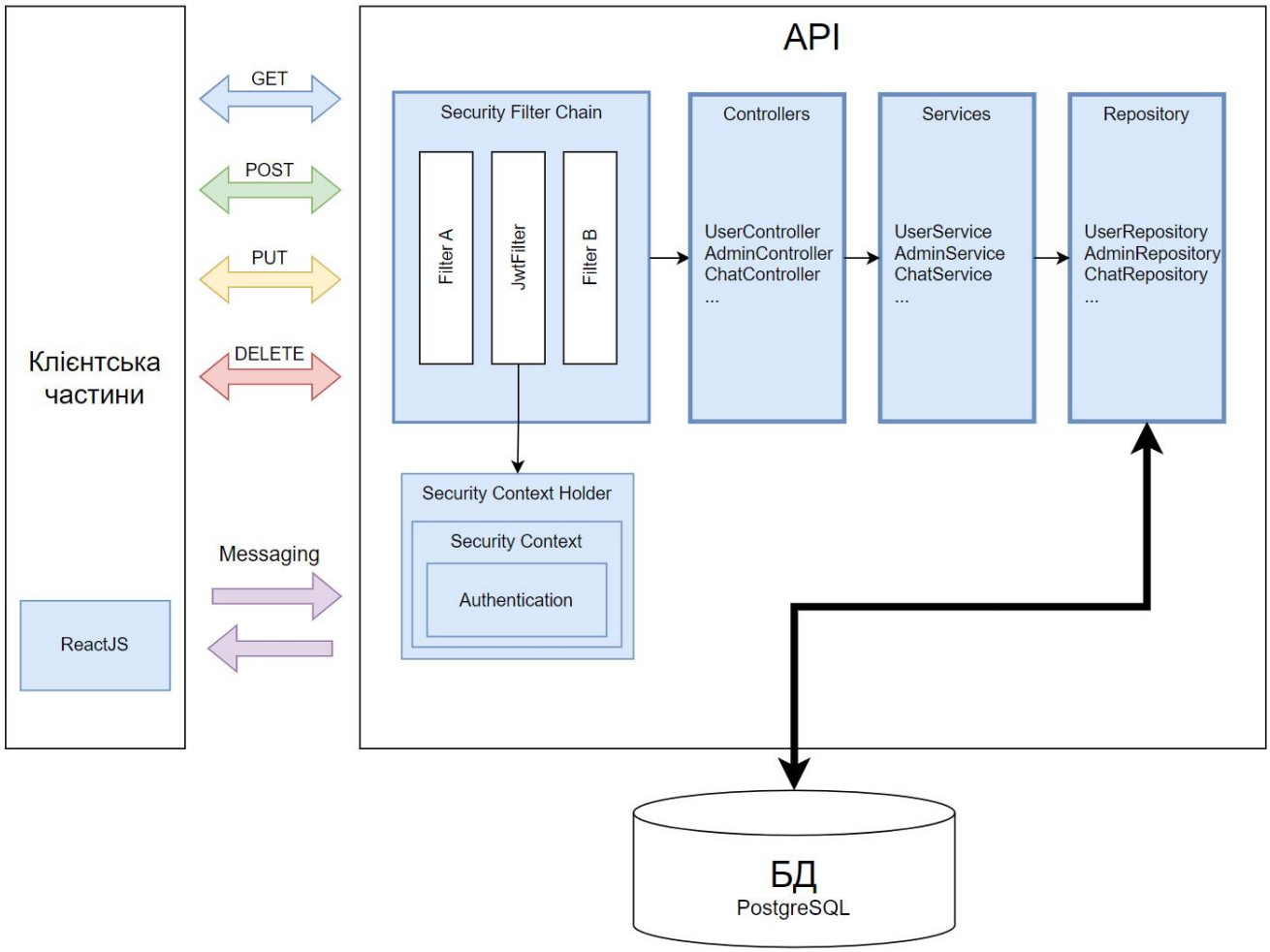
Веб-застосунок для спілкування з використанням технологій шифрування та конфіденційності даних

Структура системи
(Схема структурна)

ІАЛЦ.467100.004 Е1

Аркушів 1

Київ 2024



					ІАЛЦ.467100.004 Е1		
Зм.	*Арк.*	*№ докум.*	*Підпис*	*Дата*			
Розроб.		Глюза А. І.			Веб-застосунок для спілкування з використанням технологій шифрування та конфіденційності даних Структура системи Схема структурна	*Літ.*	*Арк.* 1 / *Аркушів* 1
Перевір.		Череватенко О.В.					
Н. Контр.		Виноградов Ю.М.				***КПІ ім. Ігоря Сікорського ФІОТ, Група ІО-01***	
Затвердив							

ДОДАТОК 2

Веб-застосунок для спілкування з використанням технологій шифрування та конфіденційності даних

Діаграма класів системи
(Схема функціональна)

ІАЛЦ.467100.005 Д2

Аркушів 1

Київ 2024

					ІАЛЦ.467100.005 Д2	
Зм.	*Арк.*	*№ докум.*	*Підпис*	*Дата*		
Розроб.		Глюза А. І.			Веб-застосунок для спілкування з використанням технологій шифрування та конфіденційності даних Діаграма класів системи Схема функціональна	*Літ.* / *Арк.* 1 / *Аркушів* 1
Перевір.		Череватенко О.В.				
Н. Контр.		Виноградов Ю.М.				***КПІ ім. Ігоря Сікорського ФІОТ, Група ІО-01***
Затвердив						

ДОДАТОК 3

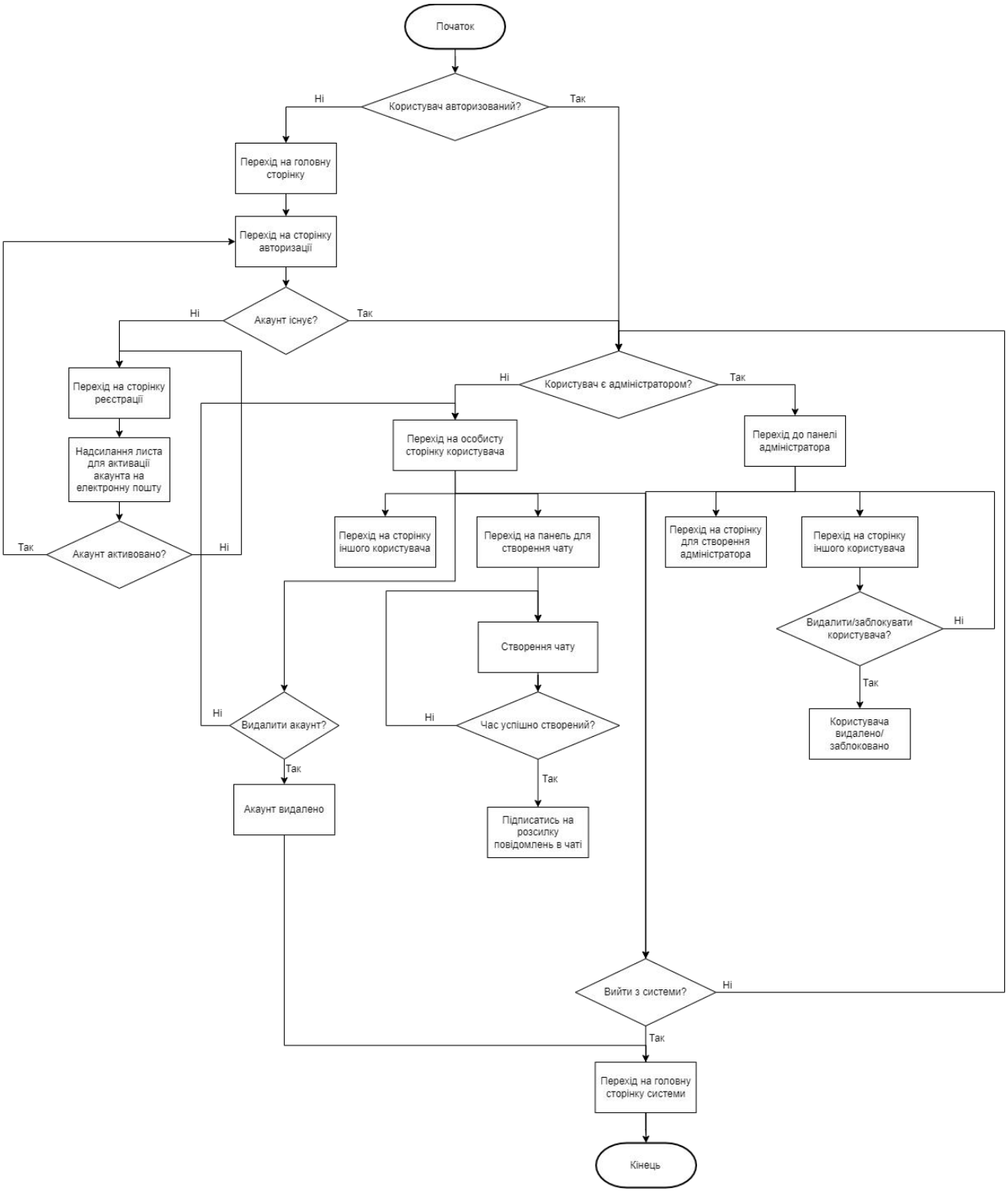
Веб-застосунок для спілкування з використанням технологій шифрування та конфіденційності даних

Алгоритм функціонування системи (Схема принципова)

ІАЛЦ.467100.006 Д3

Аркушів 1

Київ 2024



					ІАЛЦ.467100.006 Д3	
Зм.	*Арк.*	*№ докум.*	*Підпис*	*Дата*		
Розроб.		Глюза А. І.			Веб-застосунок для спілкування з використанням технологій шифрування та конфіденційності даних Алгоритм функціонування системи Схема принципова	*Літ.* / *Арк.* 1 / *Аркушів* 1
Перевір.		Череватенко О.В.				
Н. Контр.		Виноградов Ю.М.				***КПІ ім. Ігоря Сікорського ФІОТ, Група ІО-01***
Затвердив						

ДОДАТОК 4

Веб-застосунок для спілкування з використанням технологій шифрування та конфіденційності даних

Код застосунка

ІАЛЦ.467100.007 Д4

Аркушів 65

Київ 2024

```java
@OpenAPIDefinition(
        security = {
                @SecurityRequirement(
                        name = "bearerAuth"
                )
        }
)
@SecurityScheme(
        name = "bearerAuth",
        type = SecuritySchemeType.HTTP,
        bearerFormat = "JWT",
        scheme = "bearer",
        in = SecuritySchemeIn.HEADER
)
public class OpenApiConfig {
}
```

```java
@Configuration
public class WebMvcConfig implements WebMvcConfigurer {
    @Value("${application.cors.origins.url}")
    private String CORS_ALLOWED_ORIGINS;
    @Override
    public void addCorsMappings(CorsRegistry registry) {
        registry
                .addMapping("/**")
                .allowedOrigins(CORS_ALLOWED_ORIGINS)
                .allowedMethods(
                        HttpMethod.GET.name(),
                        HttpMethod.POST.name(),
                        HttpMethod.PUT.name(),
                        HttpMethod.PATCH.name(),
                        HttpMethod.DELETE.name()
                )
                .allowedHeaders("*")
                .allowCredentials(true);
    }
}
```

```java
@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
public class UserDto {
    private String username;
    private String uniqueName;
    private String registrationDate;
    private String firstname;
    private String lastname;
    private String birthday;
    private Role role;
    private UserImageDto userImage;
}
```

```java
@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
public class UserModificationRequest {
    @NotNull
    @Password
    private String currentPassword;
    private String password;
    private String uniqueName;
    private String firstname;
    private String lastname;
    @Past
    private LocalDate birthday;
    private MultipartFile userImage;
}
@RestController
@RequiredArgsConstructor
@RequestMapping("/api")
@ResponseStatus(HttpStatus.OK)
public class AdminController {
    private final UserService userService;
    @PostMapping("/admin")
    public RegistrationResponse createAdmin(
            @RequestBody RegistrationRequest registrationRequest
    ) throws Exception {
        return userService.createAdmin(registrationRequest);
    }

    @GetMapping("/admin")
    @PreAuthorize("hasAuthority('READ_ADMIN')")
    public UserDto getAdminByUsername(@RequestParam String username) throws
Exception {
        return userService.getUserByUsername(Role.ADMIN, username);
    }
@GetMapping("/root")
    @PreAuthorize("hasAuthority('READ_ROOT')")
    public UserDto getRootByUsername(@RequestParam String username) throws
Exception {
        return userService.getUserByUsername(Role.ROOT, username);
    }

    @GetMapping("/roots/{uniqueName}")
    @PreAuthorize("hasAuthority('READ_ROOT')")
    public UserDto getRootByUniqueName(@PathVariable String uniqueName) throws
Exception {
        return userService.getUserByUniqueName(Role.ROOT, uniqueName);
    }
    @GetMapping("/admins/uniqueNames")
    @PreAuthorize("hasAuthority('READ_ADMIN')")
    public Collection<UserDto>
findAdminsDifferentFromTheCurrentUserByTheirUniqueNameStartingWithPrefix(
            @RequestParam String prefix,
            @RequestParam(defaultValue = "5") int usersNumber
    ) throws Exception {
        return
userService.findUsersDifferentFromTheCurrentUserByTheirUniqueNameStartingWithPr
efix(prefix, usersNumber, Role.ADMIN);
    }
}
```

```java
@RestController
@RequiredArgsConstructor
@RequestMapping("/api/jwt")
@ResponseStatus(HttpStatus.OK)
public class JwtController {
    private final JwtService jwtService;
    @PostMapping("/access")
    public void getNewAccessToken(
            HttpServletRequest request,
            HttpServletResponse response
    ) throws Exception {
        jwtService.getNewAccessToken(request, response);
    }
    @PostMapping("/validation")
    public TokenValidationResponse validate(@RequestBody Token token) throws
Exception {
        return jwtService.validate(token);
    }
}
```

```java
@RestController
@RequiredArgsConstructor
@RequestMapping("/api")
@ResponseStatus(HttpStatus.OK)
public class UserController {
    private final UserService userService;
    private final UserImageService userImageService;
@GetMapping("/user")
@PreAuthorize("hasAuthority('READ_USER')")
public UserDto getUserByUsername(@RequestParam String username) throws
Exception {
    return userService.getUserByUsername(Role.USER, username);
}
```

```java
@GetMapping("/users/{username}/account")
@PreAuthorize("hasAuthority('READ_USER')")
public UserAccountDto getUserAccountByUserUsername(@PathVariable String
username) throws Exception {
    return userService.getUserAccountByUserUsername(username);
}
@PatchMapping("/users/{username}/account")
@PreAuthorize("hasAnyRole('ADMIN', 'ROOT')")
public UserAccountDto modifyUserAccount(
        @PathVariable String username,
        @RequestBody UserAccountDto userAccountDto
) throws Exception {
    return userService.modifyUserAccount(username, userAccountDto);
}
@GetMapping("/users")
@PreAuthorize("hasAuthority('READ_USER')")
public Collection<UserDto> getUsersPagedAndSorted(
        @RequestParam(defaultValue = "0") int page,
        @RequestParam(defaultValue = "5") int size,
        @RequestParam(defaultValue = "ASC") String order
) throws Exception {
    return userService.getUsersPagedAndSorted(page, size, order, Role.USER);
}
@DeleteMapping("/users/{username}")
@PreAuthorize("hasAuthority('DELETE_USER')")
public UserDto deleteUser(@PathVariable String username) throws Exception {
    return userService.deleteUser(username);
}
```

```
@GetMapping("/users/{uniqueName}/image")
@PreAuthorize("hasAuthority('READ_USER')")
public ResponseEntity<byte[]> getUserImage(@PathVariable String uniqueName)
throws Exception {
    UserImageDto userImageDto = userImageService.getUserImage(uniqueName);
    return ResponseEntity
            .ok()
            .contentType(MediaType.valueOf(userImageDto.getType()))
            .body(userImageDto.getData());
}
@PatchMapping("/users/{username}")
@PreAuthorize("hasAnyAuthority('UPDATE_USER', 'UPDATE_ADMIN', 'UPDATE_ROOT')")
public UserDto updateUser(
        @PathVariable String username,
        @RequestParam String currentPassword,
        @RequestParam(required = false) String password,
        @RequestParam(required = false) String uniqueName,
        @RequestParam(required = false) String firstname,
        @RequestParam(required = false) String lastname,
        @RequestParam(required = false) LocalDate birthday,
        @RequestParam(required = false) MultipartFile userImage
) throws Exception {
    return userService.updateUser(
            username,
            currentPassword,
            password,
            uniqueName,
            firstname,
            lastname,
            birthday,
            userImage
    );
}
@GetMapping("/users/{uniqueName}/image/metadata")
@PreAuthorize("hasAuthority('READ_USER')")
public UserImageDto getUserImageMetadata(@PathVariable String uniqueName)
throws Exception {
    return userImageService.getUserImage(uniqueName);
}
@GetMapping("/users/{username}/subscriptions")
@PreAuthorize("hasAuthority('READ_USER')")
public List<UserDto> getUserSubscriptions(@PathVariable String username) throws
Exception {
    return userService.getUserSubscriptions(username);
}
@GetMapping("/users/{username}/subscribers")
@PreAuthorize("hasAuthority('READ_USER')")
public List<UserDto> getUserSubscribers(@PathVariable String username) throws
Exception {
    return userService.getUserSubscribers(username);
}
@PostMapping("/users/{authenticatedUserUniqueName}/subscriptions")
@PreAuthorize("hasRole('USER')")
public Subscription subscribe(
        @PathVariable String authenticatedUserUniqueName,
        @RequestBody Subscription subscription
) throws Exception {
    return userService.subscribe(authenticatedUserUniqueName, subscription);
}
```

```
@DeleteMapping("/users/{userUniqueName}/subscriptions/{userSubscriptionUniqueNa
me}")
@PreAuthorize("hasRole('USER')")
public Subscription unsubscribe(
       @PathVariable String userUniqueName,
       @PathVariable String userSubscriptionUniqueName
) throws Exception {
    return userService.unsubscribe(userUniqueName, userSubscriptionUniqueName);
}
@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
public class EmailDto {
    @Email
    private String from;
    private String to;
    private String subject;
    private String text;
}
@Getter
public enum EmailTemplate {
    ACCOUNT_ACTIVATION("account_activation", "Account Activation");
    private final String name;
    private final String description;
    EmailTemplate(String name, String description) {
        this.name = name;
        this.description = description;
    }
}
@Service
@RequiredArgsConstructor
public class EmailServiceImpl implements EmailService {
    private final JavaMailSender javaMailSender;
    private final SpringTemplateEngine springTemplateEngine;
    private String FROM;
    @Value("${application.cors.origins.url}")
    private String CORS_ALLOWED_ORIGINS;
    @Override
    @Async
    public void sendHtmlPageEmail(
           String to,
           String subject,
           String templateName,
           Map<String, Object> extraProperties
    ) throws MessagingException {
        MimeMessage mimeMessage = javaMailSender.createMimeMessage();
        MimeMessageHelper mimeMessageHelper = new MimeMessageHelper(
               mimeMessage,
               MimeMessageHelper.MULTIPART_MODE_MIXED,
               StandardCharsets.UTF_8.name()
        );
        Context context = new Context();
        context.setVariables(extraProperties);
        mimeMessageHelper.setFrom(FROM);
        mimeMessageHelper.setTo(to);
        mimeMessageHelper.setSubject(subject);
        String template = springTemplateEngine.process(templateName, context);
        mimeMessageHelper.setText(template, true);
        javaMailSender.send(mimeMessage);
    }
```

```
    @Override
    @Async
    public void sendEmailForAccountActivation(
            String to,
            String username,
            String activationCode) throws MessagingException {
            Map<String, Object> extraProperties = new HashMap<>();
            extraProperties.put("username", username);
            extraProperties.put("confirmationUrl", CORS_ALLOWED_ORIGINS + "/user/" +
    username + "/account/activation");
            extraProperties.put("activationCode", activationCode);
            sendHtmlPageEmail(
                    to,
                    EmailTemplate.ACCOUNT_ACTIVATION.getDescription(),
                    EmailTemplate.ACCOUNT_ACTIVATION.getName(),
                    extraProperties
            );
        }
        @Override
        public EmailDto buildEmail(String to, String subject, String text) {
            return EmailDto
                    .builder()
                    .from(FROM)
                    .to(to)
                    .subject(subject)
                    .text(subject)
                    .build();
        }
    }
    @Data
    @Builder
    @NoArgsConstructor
    @AllArgsConstructor
    @Entity
    @Table(name = "jwts")
    public class Jwt {
        @Id
        @GeneratedValue(strategy = GenerationType.UUID)
        private UUID id;
        @Column(unique = true, nullable = false)
        private String content;
        @Enumerated(EnumType.STRING)
        private TokenType type;
        @Enumerated(EnumType.STRING)
        private TokenTargetType targetType;
        @ManyToOne
        @JoinColumn(name = "user_id", referencedColumnName = "id")
        private User user;
        @Transient
        private String plainContent;
    }

    @RequiredArgsConstructor
    public enum Role {
        USER(
                Set.of(
                        Permission.READ_USER,
                        Permission.UPDATE_USER,
                        Permission.DELETE_USER
                )
        ),ADMIN(
            Set.of(
```

```
                Permission.READ_USER,
                Permission.CREATE_USER,
                Permission.UPDATE_USER,
                Permission.DELETE_USER,
                Permission.READ_ADMIN,
                Permission.CREATE_ADMIN,
                Permission.UPDATE_ADMIN,
                Permission.DELETE_ADMIN
        )
    ),ROOT(
            Set.of(
                    Permission.READ_USER,
                    Permission.CREATE_USER,
                    Permission.UPDATE_USER,
                    Permission.DELETE_USER,
                    Permission.READ_ADMIN,
                    Permission.CREATE_ADMIN,
                    Permission.UPDATE_ADMIN,
                    Permission.DELETE_ADMIN,
                    Permission.READ_ROOT,
                    Permission.UPDATE_ROOT
            )
    );
private final Set<Permission> permissions;
    public List<SimpleGrantedAuthority> getAuthorities() {
       List<SimpleGrantedAuthority> authorities = permissions
               .stream()
               .map(
                       permission -> new SimpleGrantedAuthority(permission.name())
               )
               .collect(Collectors.toList());
       authorities.add(new SimpleGrantedAuthority("ROLE_" + name()));
       return authorities;
    }
}
@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
@Entity
@Table(name = "users")
public class User implements UserDetails {
    @Id
    @GeneratedValue(strategy = GenerationType.UUID)
    private UUID id;
    @Column(unique = true, nullable = false)
    private String username;
    @Column(nullable = false)
    private String password;
    @Column(unique = true, nullable = false)
    private String uniqueName;
    @Column(nullable = false)
    private ZonedDateTime registrationDate;
    @Column(nullable = false)
    private String firstname;
    private String lastname;
    private LocalDate birthday;
    @Enumerated(EnumType.STRING)
    private Role role;
    @OneToOne(mappedBy = "user", cascade = CascadeType.ALL)
    private UserAccount userAccount;
```

```
    @OneToMany(mappedBy = "user", cascade = CascadeType.ALL)
    private List<Jwt> jwts;
    @OneToMany(mappedBy = "subscriber", cascade = CascadeType.ALL)
    private List<SubscriptionSubscriber> subscriptions;
    @OneToMany(mappedBy = "subscription", cascade = CascadeType.ALL)
    private List<SubscriptionSubscriber> subscribers;
    @OneToOne(mappedBy = "user", cascade = CascadeType.ALL)
    private UserImage userImage;
    @OneToMany(mappedBy = "member", cascade = CascadeType.ALL)
    private List<ChatMember> chats;
    @OneToMany(mappedBy = "sender", cascade = {
            CascadeType.PERSIST,
            CascadeType.MERGE,
            CascadeType.REFRESH,
            CascadeType.DETACH
    })
    private List<Message> messages;
    @OneToMany(mappedBy = "user", cascade = CascadeType.ALL)
    private List<MessageStatus> messageStatuses;
@Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
        return role.getAuthorities();
    }
    @Override
    public boolean isAccountNonExpired() {
        return true;
    }
    @Override
    public boolean isAccountNonLocked() {
        return true;
    }
    @Override
    public boolean isCredentialsNonExpired() {
        return true;
    }
    @Override
    public boolean isEnabled() {
        return true;
    }
}
@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
@Entity
@Table(name = "user_account_activation_codes")
public class UserAccountActivationCode {
    @Id
    @GeneratedValue(strategy = GenerationType.UUID)
    private UUID id;
    @OneToOne
    private UserAccount userAccount;
    private String code;
}

@Repository
public interface JwtRepository extends JpaRepository<Jwt, UUID> {
    boolean existsByContent(String content);
    List<Jwt> findByUserId(UUID id);
    Optional<Jwt> findByUserIdAndTargetType(UUID userId, TokenTargetType
targetType);
}
```

```
@Repository
public interface UserAccountActivationCodeRepository extends
JpaRepository<UserAccountActivationCode, UUID> {
}
@Repository
public interface UserRepository extends JpaRepository<User, UUID>,
JpaSpecificationExecutor<User> {
    Optional<User> findByUsername(String username);
    Optional<User> findByUniqueName(String uniqueName);
    Optional<User> findByRoleAndUsername(Role role, String username);
    Optional<User> findByRoleAndUniqueName(Role role, String uniqueName);
    Optional<User> findByUsernameAndUserAccountState(String username,
AccountState state);
    boolean existsByUsername(String username);
    boolean existsByUsernameAndUserAccountState(String username, AccountState
state);
    boolean existsByUniqueName(String uniqueName);
    boolean existsByUniqueNameAndUserAccountState(String uniqueName,
AccountState state);
    List<User> findAllByUniqueNameStartingWithAndRole(String uniqueNamePrefix,
Role role);
}
@Configuration
@EnableWebSecurity
@RequiredArgsConstructor
@EnableMethodSecurity
public class SecurityConfig {
    private final JwtAuthenticationFilter jwtAuthenticationFilter;
    private final UserDetailsService userDetailsService;
    private final LogoutHandler logoutHandler;
    private final UserAuthenticationEntryPoint userAuthenticationEntryPoint;
    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws
Exception {
        return http
                .csrf(AbstractHttpConfigurer::disable)
                .cors(Customizer.withDefaults())
                .httpBasic(
                        httpSecurityHttpBasicConfigurer ->
httpSecurityHttpBasicConfigurer.authenticationEntryPoint(userAuthenticationEntr
yPoint)
                )
                .authorizeHttpRequests(
                        request -> {
                            request.requestMatchers(
                                    HttpMethod.POST,
                                    "/api/registration",
                                    "/api/authentication",
                                    "/api/user/*/account/activation",
                                    "/api/user/*/account/activation/email",
                                    "/api/jwt/**"
                            ).permitAll()
                            .requestMatchers(
                                    "/api/ws/**",
                                    "/v2/api-docs",
                                    "/v3/api-docs",
                                    "/v3/api-docs/**",
                                    "/swagger-resources",
                                    "/swagger-resources/**",
                                    "/configuration/ui",
```

```
                                        "/configuration/security",
                                        "/swagger-ui/**",
                                    "/webjars/**",
                                    "/swagger-ui.html"
                            ).permitAll()

                            .requestMatchers(
                                    "/api/user/**",
                                    "/api/users/**"
                            ).hasAnyRole(
                                    Role.USER.name(),
                                    Role.ADMIN.name(),
                                    Role.ROOT.name()
                            )
                            .requestMatchers(
                                    "/api/admin/**",
                                    "/api/admins/**"
                            ).hasAnyRole(
                                    Role.ADMIN.name(),
                                    Role.ROOT.name()
                            )
                            .requestMatchers(
                                    "/api/root/**",
                                    "/api/roots/**"
                            ).hasRole(Role.ROOT.name())

                            .anyRequest().authenticated();

                    }
            ).sessionManagement(
                    httpSecuritySessionManagementConfigurer ->
httpSecuritySessionManagementConfigurer
                            .sessionCreationPolicy(SessionCreationPolicy.STATELESS)
            )
            .authenticationProvider(authenticationProvider())
            .addFilterBefore(jwtAuthenticationFilter,
UsernamePasswordAuthenticationFilter.class)
            .logout(
                    httpSecurityLogoutConfigurer -> httpSecurityLogoutConfigurer
                            .logoutUrl("/api/logout")
                            .addLogoutHandler(logoutHandler)
                            .logoutSuccessHandler(
                                    (request, response, authentication) ->
SecurityContextHolder.clearContext()
                            )

            )
            .build();
    }
    @Bean
    public AuthenticationProvider authenticationProvider() {
        DaoAuthenticationProvider daoAuthenticationProvider = new
DaoAuthenticationProvider();
        daoAuthenticationProvider.setUserDetailsService(userDetailsService);
        daoAuthenticationProvider.setPasswordEncoder(passwordEncoder());
        return daoAuthenticationProvider;
    }

    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }
```

```
@Bean
public AuthenticationManager authenticationManager(AuthenticationConfiguration
authenticationConfiguration) throws Exception {
       return authenticationConfiguration.getAuthenticationManager();
    }
}
@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
public class AuthenticationRequest {
    @Email
    private String username;
    @Password
    private String password;
}
@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
public class AuthenticationResponse {
    private String accessToken;
    private String refreshToken;
}
@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
public class RegistrationRequest {
    @Email
    @NotNull
    private String username;
    @NotNull
    @Password
    private String password;
    @NotNull
    private String uniqueName;
    @NotNull
    private String firstname;
    @NotNull
    private String lastname;
    @Past
    private String birthday;
}
@RestController
@RequiredArgsConstructor
@RequestMapping("/api")
@ResponseStatus(HttpStatus.OK)
public class AuthenticationController {
    private final AuthenticationService authenticationService;
    @PostMapping("/registration")
    public RegistrationResponse register(
           @Valid @RequestBody RegistrationRequest registrationRequest
    ) throws Exception {
       return authenticationService.register(registrationRequest);
    }
    @PostMapping("/authentication")
    public AuthenticationResponse authenticate(
           @Valid @RequestBody AuthenticationRequest authenticationRequest
    ) throws Exception {
```

```
        return authenticationService.authenticate(authenticationRequest);
    }
    @PostMapping("/user/{username}/account/activation/email")
    public EmailDto sendEmailForUserAccountActivation(
            @PathVariable String username,
            @Valid @RequestBody EmailDto emailDto) throws Exception {
        return authenticationService.sendEmailForUserAccountActivation(username,
emailDto);
    }
    @PostMapping("/user/{username}/account/activation")
    public UserDto activateUserAccount(
            @PathVariable String username,
            @RequestBody UserAccountActivationRequest
userAccountActivationRequest) throws Exception {
        return authenticationService.activateUserAccount(username,
userAccountActivationRequest);
    }
}
@RestController
@RequiredArgsConstructor
@RequestMapping("/api")
@ResponseStatus(HttpStatus.OK)
public class EncryptionController {

    private final E2EEService e2eeService;
    @PostMapping("/encryptionKeys/publicKey")
    public E2EEDto exchangeE2EEPublicKeys(@RequestBody E2EEDto e2eeDto) throws
Exception {
        return e2eeService.exchangeE2EEPublicKeys(e2eeDto);
    }
}

@Service
@RequiredArgsConstructor
@Slf4j
public class AuthenticationServiceImpl implements AuthenticationService {
    private final UserRepository userRepository;
    private final UserAccountActivationCodeRepository
userAccountActivationCodeRepository;
    private final JwtRepository jwtRepository;
    private final UserRegistrationConverter userRegistrationConverter;
    private final UserConverter userConverter;
    private final JwtUtil jwtUtil;
    private final AuthenticationManager authenticationManager;
    private final CodeGenerator codeGenerator;
    private final EncryptionService encryptionService;
    private final EmailService emailService;
    private final AccountActivationCodeDeletionScheduler
accountActivationCodeDeletionScheduler;
@Override
public RegistrationResponse register(RegistrationRequest registrationRequest)
throws Exception {
    if
(userRepository.existsByUsernameAndUserAccountState(registrationRequest.getUser
name(), AccountState.ACTIVATED)) {
        throw new UserAlreadyExistsException("User with username " +
registrationRequest.getUsername() + " already exists in database");
    } else if {
(userRepository.existsByUniqueNameAndUserAccountState(registrationRequest.getUn
iqueName(), AccountState.ACTIVATED)) {
        throw new UserAlreadyExistsException("User with uniqueName " +
```

```
registrationRequest.getUniqueName() + " already exists in database");
    }
deleteNotActivatedUser(registrationRequest.getUsername());
User userToSave = userRegistrationConverter.toEntity(registrationRequest);
UserAccount userAccount = UserAccount
       .builder()
       .user(userToSave)
       .state(AccountState.REQUIRE_ACTIVATION)
       .createdAt(ZonedDateTime.now())
       .build();

String activationCode = codeGenerator.generate(6);
UserAccountActivationCode userAccountActivationCode = UserAccountActivationCode
       .builder()
       .userAccount(userAccount)
       .code(encryptionService.encrypt(activationCode))
       .build();

userAccount.setUserAccountActivationCode(userAccountActivationCode);
userToSave.setUserAccount(userAccount);
User savedUser = userRepository.save(userToSave);
UserDto registeredUserDto = userConverter.toDto(savedUser);
accountActivationCodeDeletionScheduler.scheduleUserDeletionForNotActivatedUser(
savedUser);
emailService.sendEmailForAccountActivation(
       registeredUserDto.getUsername(),
       registeredUserDto.getUniqueName(),
       activationCode
);

    return RegistrationResponse
           .builder()
           .user(registeredUserDto)
           .isRegistrationSuccessful(true)
           .build();
}

@Override
public AuthenticationResponse authenticate(AuthenticationRequest
authenticationRequest) throws Exception {
authenticationManager.authenticate(
       new UsernamePasswordAuthenticationToken(
              authenticationRequest.getUsername(),
              authenticationRequest.getPassword()
       )
);
    User user = userRepository
           .findByUsername(authenticationRequest.getUsername())
           .orElseThrow(
                  () -> new UsernameNotFoundException(
                         "User with username: "
                                + authenticationRequest.getUsername() +
                                "was not found in database"
                  )
           );
    validateUserAccount(user);
    return buildAuthenticationResponse(user);
}
@Override
public EmailDto sendEmailForUserAccountActivation(String username, EmailDto
emailDto) throws Exception {
User user = userRepository
       .findByUsernameAndUserAccountState(emailDto.getTo(),
```

```
AccountState.REQUIRE_ACTIVATION)
        .orElseThrow(
                () -> new UserNotFoundException("User with username " +
emailDto.getTo() + " not found")
        );

String activationCode = codeGenerator.generate(6);
UserAccountActivationCode userAccountActivationCode =
user.getUserAccount().getUserAccountActivationCode();
userAccountActivationCode.setCode(encryptionService.encrypt(activationCode));
User savedUser = userRepository.save(user);

emailService.sendEmailForAccountActivation(
        savedUser.getUsername(),
        savedUser.getUniqueName(),
        activationCode
);

    return emailService.buildEmail(
            savedUser.getUsername(),
            EmailTemplate.ACCOUNT_ACTIVATION.getDescription(),
            EmailTemplate.ACCOUNT_ACTIVATION.getDescription()
    );
}

@Override
public UserDto activateUserAccount(String username,
UserAccountActivationRequest userAccountActivationRequest) throws Exception {
    String userAccountActivationRequestUsername =
userAccountActivationRequest.getUsername();
    if (username == null
|| !username.equals(userAccountActivationRequestUsername)) {
        throw new IllegalArgumentException("Usernames are different");
    }
    String activationCode = userAccountActivationRequest.getActivationCode();

    User user = userRepository
            .findByUsernameAndUserAccountState(username,
AccountState.REQUIRE_ACTIVATION)
            .orElseThrow(
                    () -> new UserNotFoundException("User with username " +
username + " not found")
            );
    if (activationCode == null) {
        throw new IllegalArgumentException("Activation code is null");
    }
    User savedUser = null;
    UserAccount userAccount = user.getUserAccount();
    UserAccountActivationCode userAccountActivationCode =
userAccount.getUserAccountActivationCode();
    String encryptedActivationCode = userAccountActivationCode.getCode();
    if (encryptionService.matches(activationCode, encryptedActivationCode)) {
        userAccount.setState(AccountState.ACTIVATED);
        userAccount.setActivatedAt(ZonedDateTime.now());
        userAccount.setUserAccountActivationCode(null);
        savedUser = userRepository.save(user);
    userAccountActivationCodeRepository.deleteById(userAccountActivationCode.ge
tId());
    }
    return userConverter.toDto(savedUser);
}
    private AuthenticationResponse buildAuthenticationResponse(User user)
throws Exception {
```

```
        Jwt accessToken = jwtUtil.generateToken(user, TokenTargetType.ACCESS);
        Jwt refreshToken = jwtUtil.generateToken(user, TokenTargetType.REFRESH);
        jwtRepository.save(accessToken);
        jwtRepository.save(refreshToken);
        return AuthenticationResponse
                .builder()
                .accessToken(accessToken.getPlainContent())
                .refreshToken(refreshToken.getPlainContent())
                .build();
    }

    private AuthenticationResponse buildAuthenticationResponse(User user) throws
    Exception {
        Jwt accessToken = jwtUtil.generateToken(user, TokenTargetType.ACCESS);
        Jwt refreshToken = jwtUtil.generateToken(user, TokenTargetType.REFRESH);
        jwtRepository.save(accessToken);
        jwtRepository.save(refreshToken);
        return AuthenticationResponse
                .builder()
                .accessToken(accessToken.getPlainContent())
                .refreshToken(refreshToken.getPlainContent())
                .build();
    }

    @Override
    public UserDetails getAuthenticatedUserUserDetailsFromSecurityContext() {
        Authentication authentication =
    SecurityContextHolder.getContext().getAuthentication();
        UserDetails authenticatedUser = null;
        if (authentication != null) {
            authenticatedUser = (UserDetails) authentication.getPrincipal();
        }
        return authenticatedUser;
    }

    @Override
    public User getCurrentUser() throws UserNotAuthenticatedException {
        UserDetails currentUserUserDetails =
    getAuthenticatedUserUserDetailsFromSecurityContext();
        if (currentUserUserDetails == null) {
            throw new UserNotAuthenticatedException("User not authenticated");
        }
        return (User) currentUserUserDetails;
    }


    private void deleteNotActivatedUser(String username) {
        userRepository.findByUsernameAndUserAccountState(
                username,
                AccountState.REQUIRE_ACTIVATION
        ).ifPresent(userRepository::delete);
    }
       private void validateUserAccount(User user) throws Exception {
            AccountState userAccountState = user.getUserAccount().getState();
            if (userAccountState.equals(AccountState.REQUIRE_ACTIVATION)) {
                throw new UserAccountNotActivatedException("User account for user
    with username "
                        + user.getUsername() + " is not activated");
            }
            if (userAccountState.equals(AccountState.BLOCKED)) {
                throw new UserAccountBlockedException("User account for user with
    username "
```

```
                    + user.getUsername() + " is blocked");
            }
        }
    }

    @Service
    @RequiredArgsConstructor
    public class E2EEServiceImpl implements E2EEService {
        private final AuthenticationService authenticationService;
        private final ConcurrentHashMap<UUID, CryptoKeysContainer>
    cryptoKeysContainers = new ConcurrentHashMap<>(); // userId cryptoKeysContainer
        private KeyPair generateKeyPair() throws NoSuchAlgorithmException {
            KeyPairGenerator keyPairGenerator = KeyPairGenerator.getInstance("RSA");
            keyPairGenerator.initialize(2048);
            return keyPairGenerator.generateKeyPair();
        }
        @Override
        public E2EEDto exchangeE2EEPublicKeys(E2EEDto e2eeDto) throws Exception {
            User currentUser = authenticationService.getCurrentUser();
            KeyPair serverKeyPair = generateKeyPair();
            PrivateKey serverPrivateKey = serverKeyPair.getPrivate();
            PublicKey serverPublicKey = serverKeyPair.getPublic();
            PublicKey userPublicKey =
    convertStringToPublicKeyAndGet(e2eeDto.getPublicKey());
            CryptoKeysContainer cryptoKeysContainer = CryptoKeysContainer
                    .builder()
                    .serverPrivateKey(serverPrivateKey)
                    .serverPublicKey(serverPublicKey)
                    .userPublicKey(userPublicKey)
                    .build();
            cryptoKeysContainers.put(currentUser.getId(), cryptoKeysContainer);
            return E2EEDto
                    .builder()
                    .publicKey(Base64
                            .getEncoder()
                            .encodeToString(serverPublicKey.getEncoded())
                    )
                    .build();
        }

    private PublicKey convertStringToPublicKeyAndGet(String publicKeyString) throws
    Exception {
        publicKeyString = publicKeyString.trim().replaceAll("\\s+", "");
        byte[] publicKeyBytes = Base64.getDecoder().decode(publicKeyString);
        X509EncodedKeySpec x509EncodedKeySpec = new
    X509EncodedKeySpec(publicKeyBytes);
        KeyFactory keyFactory = KeyFactory.getInstance("RSA");
        return keyFactory.generatePublic(x509EncodedKeySpec);
    }
    private PublicKey getPublicKey() throws Exception {
        User currentUser = authenticationService.getCurrentUser();

        CryptoKeysContainer cryptoKeysContainer =
    cryptoKeysContainers.get(currentUser.getId());
        return cryptoKeysContainer != null ?
    cryptoKeysContainer.getServerPrivateKey() : null;
    }

        @Override
    public String encrypt(String plainText) throws Exception {
        PublicKey publicKey = getPublicKey();
        if (publicKey == null) {
```

```
            throw new E2EEKeyNotFoundException("User's public encryption key does
not exist");
        }
        Cipher cipher = Cipher.getInstance("RSA");
        cipher.init(Cipher.ENCRYPT_MODE, publicKey);
        byte[] encryptedBytes = cipher.doFinal(plainText.getBytes());
        return Base64.getEncoder().encodeToString(encryptedBytes);
    }


        @Override
        public String decrypt(String encryptedText) throws Exception {
            PrivateKey privateKey = getPrivateKey();
            if (privateKey == null) {
                throw new E2EEKeyNotFoundException("Server's private encryption key
does not exist");
            }
            Cipher cipher = Cipher.getInstance("RSA");
            cipher.init(Cipher.DECRYPT_MODE, privateKey);
            byte[] encryptedBytes = Base64.getDecoder().decode(encryptedText);
            byte[] decryptedBytes = cipher.doFinal(encryptedBytes);
            return new String(decryptedBytes);
        }
    }
    @Service
    public class EncryptionServiceImpl implements EncryptionService {
        private final Cipher cipher;
        private final SecretKey secretKey;
        public EncryptionServiceImpl(
                @Value("${application.security.service.encryption.algorithm}") String
    encryptionAlgorithm,
                @Value("${application.security.service.encryption.aes-key}") String
    encryptionKey
        ) throws Exception {
            cipher = Cipher.getInstance(encryptionAlgorithm);
            secretKey = new SecretKeySpec(encryptionKey.getBytes(),
    encryptionAlgorithm);
        }

    public String encrypt(String value) throws Exception {
        cipher.init(Cipher.ENCRYPT_MODE, secretKey);
        byte[] encryptedValueBytes = cipher.doFinal(value.getBytes());
        return Base64.getEncoder().encodeToString(encryptedValueBytes);
    }

    public String decrypt(String encryptedValue) throws Exception {
            cipher.init(Cipher.DECRYPT_MODE, secretKey);
            byte[] encryptedValueBytes = Base64.getDecoder().decode(encryptedValue);
            byte[] decryptedValueBytes = cipher.doFinal(encryptedValueBytes);
            return new String(decryptedValueBytes);
        }
        public boolean matches(String rawValue, String encryptedValue) throws
    Exception {
            String encryptedRawValue = encrypt(rawValue);
            return encryptedRawValue.equals(encryptedValue);
        }
    }
    @Component
    public class JwtAuthenticationFilter extends OncePerRequestFilter {
        private final JwtUtil jwtUtil;
        private final UserDetailsService userDetailsService;
        private final HandlerExceptionResolver handlerExceptionResolver;
```

```
    public JwtAuthenticationFilter(
            JwtUtil jwtUtil,
            UserDetailsService userDetailsService,
            @Qualifier("handlerExceptionResolver") HandlerExceptionResolver
handlerExceptionResolver) {
        this.jwtUtil = jwtUtil;
        this.userDetailsService = userDetailsService;
        this.handlerExceptionResolver = handlerExceptionResolver;
    }
    @Override
    protected void doFilterInternal(
            @NonNull HttpServletRequest request,
            @NonNull HttpServletResponse response,
            @NonNull FilterChain filterChain
    ) {
        try {
            final String authenticationHeader =
request.getHeader(HttpHeaders.AUTHORIZATION);
            final String jwt;
            final String username;
            if (authenticationHeader != null &&
authenticationHeader.startsWith("Bearer ")) {
                jwt = authenticationHeader.substring(7);
                username = jwtUtil.extractUsername(jwt);
                if (username != null &&
SecurityContextHolder.getContext().getAuthentication() == null) {
                    UserDetails userDetails =
userDetailsService.loadUserByUsername(username);
                    if (jwtUtil.isTokenValid(jwt, userDetails)) {
                        UsernamePasswordAuthenticationToken
usernamePasswordAuthenticationToken = new UsernamePasswordAuthenticationToken(
                                userDetails,
                                null,
                                userDetails.getAuthorities()
                        );
                    usernamePasswordAuthenticationToken.setDetails(
                                new
WebAuthenticationDetailsSource().buildDetails(request)
                        );
SecurityContextHolder.getContext().setAuthentication(usernamePasswordAuthentica
tionToken);
                    }
                }
            }
            filterChain.doFilter(request, response);
        } catch (Exception exception) {
            handlerExceptionResolver.resolveException(request, response, null,
exception);
        }
    }
}
@Service
@Slf4j
@RequiredArgsConstructor
@Transactional
public class JwtUtil {


@Value("${application.jwt.secret-encryption-key}")
    private String SECRET_ENCRYPTION_KEY;
    @Value("${application.jwt.refresh-token.expiration-date}")
    private int REFRESH_TOKEN_EXPIRATION_DATE;
```

```
    @Value("${application.jwt.access-token.expiration-date}")
    private int ACCESS_TOKEN_EXPIRATION_DATE;
    private final JwtRepository jwtRepository;
    private final EncryptionService encryptionService;
    public Jwt generateToken(User user, TokenTargetType targetType) throws
Exception {
        Jwt token = jwtRepository
                .findByUserIdAndTargetType(user.getId(), targetType)
                .orElse(null);
        if (token != null) {
            jwtRepository.delete(token);
        String plainContent = switch (targetType) {
            case ACCESS -> generateAccessToken(user);
            case REFRESH -> generateRefreshToken(user);
        };
        String encryptedContent = encryptionService.encrypt(plainContent);
        token = Jwt
                .builder()
                .content(encryptedContent)
                .type(TokenType.BEARER)
                .targetType(targetType)
                .user(user)
                .plainContent(plainContent)
                .build();
    if (jwtRepository.existsByContent(encryptedContent)) {
        jwtRepository.delete(token);;
    return token;
}
public String generateAccessToken(User user) {
    Map<String, Object> extraClaims = new HashMap<>();
    extraClaims.put("role", user.getRole().name());
    extraClaims.put("target", TokenTargetType.ACCESS.name());
    return buildToken(extraClaims, user, ACCESS_TOKEN_EXPIRATION_DATE);
}
public String generateRefreshToken(User user) {
    Map<String, Object> extraClaims = new HashMap<>();
    extraClaims.put("target", TokenTargetType.REFRESH.name());
    return buildToken(extraClaims, user, REFRESH_TOKEN_EXPIRATION_DATE);
}
private String buildToken(
        Map<String, Object> extraClaims,
        UserDetails userDetails,
        int expirationTime
) {
    final Date creationDate = new Date();
    final Date expirationDate = new Date(creationDate.getTime() +
expirationTime);
    return Jwts
            .builder()
            .setClaims(extraClaims)
            .setSubject(userDetails.getUsername())
            .setIssuedAt(creationDate)
            .setExpiration(expirationDate)
            .signWith(getSigningKey(), SignatureAlgorithm.HS256)
            .compact();
}
public boolean isTokenValid(String jwt, UserDetails userDetails) throws
Exception {
    final String username = extractUsername(jwt);
    final TokenTargetType tokenTargetType = extractTokenTargetType(jwt);
    if (tokenTargetType == null) {return false;}
```

```
        Jwt userJwt = jwtRepository
                .findByUserIdAndTargetType(((User) userDetails).getId(),
    tokenTargetType)
                .orElse(null);
        return username.equals(userDetails.getUsername())
                && !isTokenExpired(jwt)
                && (userJwt != null)
                && encryptionService.matches(jwt, userJwt.getContent());
    }


    private boolean isTokenExpired(String jwt) {
            return extractExpiration(jwt).before(new Date());
        }

        private Date extractExpiration(String jwt) {
            return extractClaim(jwt, Claims::getExpiration);
        }

        public String extractUsername(String jwt) {
            return extractClaim(jwt, Claims::getSubject);
        }

        private TokenTargetType extractTokenTargetType(String jwt) {
            String targetClaim = extractClaim(jwt, claims -> claims.get("target",
    String.class));
            if (targetClaim != null) {
                return TokenTargetType.valueOf(targetClaim.toUpperCase());
            }
            return null;
        }
    private <T> T extractClaim(String jwt, Function<Claims, T> claimsResolver) {
            final Claims claims = extractAllClaims(jwt);
            return claimsResolver.apply(claims);
        }
        private Claims extractAllClaims(String token) {
            return Jwts
                    .parserBuilder()
                    .setSigningKey(getSigningKey())
                    .build()
                    .parseClaimsJws(token)
                    .getBody();
        }
        private Key getSigningKey() {
            byte[] keyBytes = Decoders.BASE64.decode(SECRET_ENCRYPTION_KEY);
            return Keys.hmacShaKeyFor(keyBytes);
        }
    }
    @Service
    @Transactional
    @RequiredArgsConstructor
    public class LogoutHandlerImpl implements LogoutHandler {
        private final JwtRepository jwtRepository;
        private final UserRepository userRepository;
        private final JwtUtil jwtUtil;

        @Override
        public void logout(
                HttpServletRequest request,
                HttpServletResponse response,
                Authentication authentication
        ) {
```

```
        final String authorizationHeader = request.getHeader("Authorization");
        final String jwtContent;
        final String username;
        final User user;
        final List<Jwt> tokens;
        if (authorizationHeader != null &&
authorizationHeader.startsWith("Bearer ")) {
            jwtContent = authorizationHeader.substring(7);
            username = jwtUtil.extractUsername(jwtContent);

            if (username != null) {
                user = userRepository
                        .findByUsername(username)
                        .orElseThrow(
                                () -> new UsernameNotFoundException("User with
username " + username + " not found")
                        );
                try {
                    if (user != null && jwtUtil.isTokenValid(jwtContent, user)) {
                        tokens = jwtRepository.findByUserId(user.getId());

                        if (!tokens.isEmpty()) {
                            jwtRepository.deleteAll(tokens);
                        }
                    }
                } catch (Exception e) {
                    throw new RuntimeException(e);
                }
            }
        }
    }
}
```

```
@Service
public class UserAccountActivationCodeGenerator implements CodeGenerator {
    @Override
    public String generate(int length) {
        StringBuilder codeBuilder = new StringBuilder();
        SecureRandom secureRandom = new SecureRandom();
        for (int i = 0; i < length; i++) {
            int randomIndex = secureRandom.nextInt(10);
            codeBuilder.append(randomIndex);
        }
        return codeBuilder.toString();
    }
}
```

```
@Component
public class UserAccountStateFilter extends OncePerRequestFilter {
    private final JwtUtil jwtUtil;
    private final UserDetailsService userDetailsService;
    private final HandlerExceptionResolver handlerExceptionResolver;
    public UserAccountStateFilter(
            JwtUtil jwtUtil,
            UserDetailsService userDetailsService,
            @Qualifier("handlerExceptionResolver") HandlerExceptionResolver
handlerExceptionResolver) {
        this.jwtUtil = jwtUtil;
        this.userDetailsService = userDetailsService;
      this.handlerExceptionResolver = handlerExceptionResolver;
    }
```

```
    @Override
    protected void doFilterInternal(
            @NonNull HttpServletRequest request,
            @NonNull HttpServletResponse response,
            @NonNull FilterChain filterChain
    ) {
        try {
            final String authenticationHeader =
request.getHeader(HttpHeaders.AUTHORIZATION);
            final String jwt;
            final String username;
            if (authenticationHeader != null &&
authenticationHeader.startsWith("Bearer ")) {
                jwt = authenticationHeader.substring(7);
                username = jwtUtil.extractUsername(jwt);
                if (username != null) {
                    UserDetails userDetails =
userDetailsService.loadUserByUsername(username);
                    if (userDetails != null) {
                        User user = (User) userDetails;
                        UserAccount userAccount = user.getUserAccount();
                    }
                }
            }
            filterChain.doFilter(request, response);
        } catch (Exception exception) {
            handlerExceptionResolver.resolveException(request, response, null,
exception);
        }
    }
}
@Component
public class UserAuthenticationEntryPoint implements AuthenticationEntryPoint {
    private final ObjectMapper objectMapper = new ObjectMapper();
    @Override
    public void commence(
            HttpServletRequest request,
            HttpServletResponse response,
            AuthenticationException authException
    ) throws IOException, ServletException {
        response.setStatus(HttpServletResponse.SC_UNAUTHORIZED);
        response.setHeader(HttpHeaders.CONTENT_TYPE,
MediaType.APPLICATION_JSON_VALUE);
        if (authException instanceof InsufficientAuthenticationException) {
            objectMapper.writeValue(
                    response.getOutputStream(),
                    ExceptionResponse
                            .builder()
                            .message("Bad credentials")
                            .build()
            );
        } else {
            objectMapper.writeValue(
                    response.getOutputStream(),
                    ExceptionResponse
                            .builder()
                            .message("Authorization exception")
                            .build()
            );
```

```
        }
    }
}
@Service
@RequiredArgsConstructor
public class UserDetailsServiceImpl implements UserDetailsService {
    private final UserRepository userRepository;
    @Override
    public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {
        return userRepository
                .findByUsername(username)
                .orElseThrow(
                        () -> new UsernameNotFoundException("User: " + username + "
not found")
                );
    }
}
@Service
@RequiredArgsConstructor
public class ChatConverter implements Converter<ChatDto, Chat> {
    private final AuthenticationService authenticationService;
    private final UserRepository userRepository;
    private final ChatMemberRepository chatMemberRepository;
    private final MessageRepository messageRepository;
    private final UserConverter userConverter;
    private final MessageConverter messageConverter;

@Override
public ChatDto toDto(Chat chat) throws Exception {
    UUID chatId = chat.getId();
    List<ChatMemberDto> convertedChatMembers = new ArrayList<>();
    List<MessageDto> convertedChatMessages = new ArrayList<>();
    List<ChatMember> chatMembers =
chatMemberRepository.findAllByChatId(chatId);
    for (ChatMember chatMember : chatMembers) {
        UserDto userDto = userConverter.toDto(chatMember.getMember());
        MemberRole memberRole = chatMember.getMemberRole();
        String convertedChatId = chatMember.getChat().getId().toString();
        convertedChatMembers.add(
                ChatMemberDto
                        .builder()
                        .chatId(convertedChatId)
                        .user(userDto)
                        .role(memberRole)
                        .build()
        );
    }
    List<Message> chatMessages =
messageRepository.findAllByChatIdOrderBySendTime(chatId);
    for (Message message : chatMessages) {
        convertedChatMessages.add(messageConverter.toDto(message));
    }
    return setChatNameForCurrentUser(ChatDto
            .builder()
            .id(chat.getId().toString())
            .name(chat.getName())
            .type(chat.getType().name())
            .members(convertedChatMembers)
            .messages(convertedChatMessages)
            .build());
}
```

```
    @Override
public Chat toEntity(ChatDto chatDto) throws Exception {
    User currentUser = authenticationService.getCurrentUser();
    ChatType chatType = ChatType.valueOf(chatDto.getType().toUpperCase());

    Chat chat = Chat
           .builder()
           .name(chatDto.getName())
           .type(chatType)
           .build();
    List<ChatMember> membersToAddToChat = new ArrayList<>();
    Collection<ChatMemberDto> members = chatDto.getMembers();
    for (ChatMemberDto chatMemberDto : members) {
        UserDto userDto = chatMemberDto.getUser();
        String username = userDto.getUsername();
        MemberRole memberRole = chatMemberDto.getRole();
        userRepository
               .findByUsername(username)
               .ifPresent(user -> membersToAddToChat.add(
                      ChatMember
                             .builder()
                             .chat(chat)
                             .member(user)
                             .memberRole(memberRole)
                             .build()
               ));
    }
    if (membersToAddToChat.isEmpty()) {
        throw new IllegalArgumentException("Chat has no members to add");
    }
    membersToAddToChat.add(
           ChatMember
                  .builder()
                  .chat(chat)
                  .member(currentUser)
                  .memberRole(MemberRole.ADMIN)
                  .build()
    );
    chat.setMembers(membersToAddToChat);
    return chat;
}
    private ChatDto setChatNameForCurrentUser(ChatDto chatDto) throws
UserNotAuthenticatedException {
        User currentUser = authenticationService.getCurrentUser();
        ChatType chatType = ChatType.valueOf(chatDto.getType().toUpperCase());
        String chatName = switch (chatType) {
            case GROUP_CHAT -> chatDto.getName();
            case PRIVATE_CHAT -> chatDto
                   .getMembers()
                   .stream()
                   .filter(chatMemberDto -> !chatMemberDto
                          .getUser()
                          .getUsername()
                          .equals(currentUser.getUsername()))
                   .findFirst()
                   .map(chatMemberDto -> chatMemberDto.getUser().getUniqueName())
                   .orElse(null);
        };
        chatDto.setName(chatName);
        return chatDto;
    }
}
```

```java
@Service
@Transactional
@RequiredArgsConstructor
@Slf4j
public class JwtServiceImpl implements JwtService {
    private final JwtUtil jwtUtil;
    private final JwtRepository jwtRepository;
    private final UserRepository userRepository;
    private final ObjectMapper objectMapper = new ObjectMapper();
    public void getNewAccessToken(
            HttpServletRequest request,
            HttpServletResponse response
    ) throws Exception {
        final String authenticationHeader =
request.getHeader(HttpHeaders.AUTHORIZATION);
        final String jwt;
        final String username;

if (authenticationHeader != null && authenticationHeader.startsWith("Bearer "))
{
    jwt = authenticationHeader.substring(7);
    username = jwtUtil.extractUsername(jwt);
            if (username != null) {
                User user = userRepository
                        .findByUsername(username)
                        .orElseThrow(
                                () -> new UsernameNotFoundException("User with
username " + username + " not found")
                        );

                if (jwtUtil.isTokenValid(jwt, user)) {
                    Jwt accessJwt = jwtUtil.generateToken(user,
TokenTargetType.ACCESS);
                    jwtRepository.save(accessJwt);
                    Token accessToken = Token
                            .builder()
                            .content(accessJwt.getPlainContent())
                            .build();
                    objectMapper.writeValue(response.getOutputStream(),
accessToken);
                }
            }
        }
    }
    @Override
    public TokenValidationResponse validate(Token token) throws Exception {
        final String jwt = token.getContent();
        String username = jwtUtil.extractUsername(jwt);
        boolean isValid = false;
        if (username != null) {
            User user = userRepository
                    .findByUsername(username);
            isValid = jwtUtil.isTokenValid(jwt, user);
        }
        return TokenValidationResponse
                .builder()
                .token(token.getContent())
                .isValid(isValid)
                .build();
    }
}
```

```java
@Service
@RequiredArgsConstructor
public class MessageConverter implements Converter<MessageDto, Message> {
    private final ChatRepository chatRepository;
    private final UserConverter userConverter;
    private final AuthenticationService authenticationService;
    private final MessageStatusRepository messageStatusRepository;
    private final EncryptionService encryptionService;
@Override
public MessageDto toDto(Message message) throws Exception {
    UUID messageId = message.getId();
    User currentUser = authenticationService.getCurrentUser();
    UUID currentUserId = currentUser.getId();
    String currentUserUsername = currentUser.getUsername();

    User sender = message.getSender();
    UserDto senderDto = userConverter.toDto(sender);
    String chatId = message.getChat().getId().toString();
    String decryptedContent = encryptionService.decrypt(message.getContent());
    MessageStatus messageStatusForCurrentUser = messageStatusRepository
           .findByMessageIdAndUserId(messageId, currentUserId);
    Status status = messageStatusForCurrentUser.getStatus();
    return MessageDto
           .builder()
           .id(messageId.toString())
           .sender(senderDto)
           .chatId(chatId)
           .content(decryptedContent)
           .sendTime(message.getSendTime().toString())
           .type(message.getType())
           .status(status)
           .build();
}
    @Override
    public Message toEntity(MessageDto messageDto) throws Exception {
        User currentUser = authenticationService.getCurrentUser();
        UUID currentUserId = currentUser.getId();
        String username = currentUser.getUsername();
        UUID chatId = UUID.fromString(messageDto.getChatId());
        Chat chat = chatRepository
               .findByChatIdAndChatMemberId(chatId, currentUserId);
        String encryptedContent =
encryptionService.encrypt(messageDto.getContent());
        MessageType messageType = messageDto.getType();
        if (messageDto.getContent() == null ||
messageDto.getContent().isBlank()) {
            throw new IllegalArgumentException("Illegal message content");
        }
        return Message
               .builder()
               .sender(currentUser)
               .chat(chat)
               .content(encryptedContent)
               .sendTime(ZonedDateTime.now())
               .type(messageType)
               .build();
    }
}
```

```
@Service
@RequiredArgsConstructor
public class UserConverter implements Converter<UserDto, User> {
    private final UserImageConverter userImageConverter;
    private final DateTimeFormatter dateTimeFormatter;
    @Override
    public UserDto toDto(User user) throws Exception {
        UserDto userDto = null;
        if (user != null) {
            UserImage userImage = user.getUserImage();
            UserImageDto userImageDto = userImageConverter.toDto(userImage);
            String birthdayToReturn = null;
            LocalDate birthday = user.getBirthday();
            if (birthday != null) {
                birthdayToReturn = birthday.format(dateTimeFormatter);
            }
            userDto = UserDto
                    .builder()
                    .username(user.getUsername())
                    .uniqueName(user.getUniqueName())
                    .registrationDate(user.getRegistrationDate().toString())
                    .firstname(user.getFirstname())
                    .lastname(user.getLastname())
                    .birthday(birthdayToReturn)
                    .role(user.getRole())
                    .userImage(userImageDto)
                    .build();
      }
        return userDto;
    }
    @Override
    public User toEntity(UserDto userDto) throws Exception {
        UserImageDto userImageDto = userDto.getUserImage();
        UserImage userImage = userImageConverter.toEntity(userImageDto);
        ZonedDateTime registrationDate =
ZonedDateTime.parse(userDto.getRegistrationDate());
        LocalDate birthday = null;
        String receivedBirthday = userDto.getBirthday();
        if (receivedBirthday != null && !receivedBirthday.isBlank()) {
            birthday = LocalDate.parse(receivedBirthday, dateTimeFormatter);
        }
        return User
                .builder()
                .username(userDto.getUsername())
                .uniqueName(userDto.getUniqueName())
                .registrationDate(registrationDate)
                .firstname(userDto.getFirstname())
                .lastname(userDto.getLastname())
                .birthday(birthday)
                .role(userDto.getRole())
                .userImage(userImage)
                .build();
    }
}
@Service
@RequiredArgsConstructor
public class UserServiceImpl implements UserService {
    private final UserRepository userRepository;
    private final UserAccountRepository userAccountRepository;
    private final SubscriptionSubscriberRepository
subscriptionSubscriberRepository;
```

```
    private final UserConverter userConverter;
    private final UserAccountConverter userAccountConverter;
    private final UserRegistrationConverter userRegistrationConverter;
    private final SubscriptionSubscriberConverter
subscriptionSubscriberConverter;
    private final UserModifier userModifier;
    private final AuthenticationService authenticationService;
    private final PasswordEncoder passwordEncoder;
    private final MessageService messageService;
    @Override
public UserDto getUserByUsername(Role role, String username) throws Exception {
    User user = userRepository
           .findByRoleAndUsername(role, username);
    return userConverter.toDto(user);
}
@Override
public UserDto getUserByUniqueName(Role role, String uniqueName) throws
Exception {
    User user = userRepository
           .findByRoleAndUniqueName(role, uniqueName);
    return userConverter.toDto(user);
}
@Override
public UserAccountDto getUserAccountByUserUsername(String username) throws
Exception {
    User user = userRepository
           .findByUsername(username);
    UserAccount userAccount = user.getUserAccount();
    return userAccountConverter.toDto(userAccount);
}
@Override
public UserAccountDto modifyUserAccount(String username, UserAccountDto
userAccountDto) throws Exception {
    if (username == null || userAccountDto == null) {
       throw new IllegalArgumentException("UserAccountDto or username is
null");
    }
    UserDto userDto = userAccountDto.getUserDto();
    User user = userRepository
           .findByUsername(username);
    UserAccount userAccount = userAccountRepository
           .findByUserUsername(username);
    AccountState accountState = userAccountDto.getState();
    if (!userAccount.getState().equals(accountState)) {
       userAccount.setState(accountState);
    }
       UserAccount savedUserAccount = userAccountRepository.save(userAccount);
    return userAccountConverter.toDto(savedUserAccount);
}
    @Override
public Collection<UserDto> getUsersPagedAndSorted(int page, int size, String
order, Role userRole) throws Exception {
    Sort.Direction sortOrder = Sort.Direction.valueOf(order.toUpperCase());
    Sort sort = Sort.by(sortOrder, "uniqueName");
    Pageable pageable = PageRequest.of(page, size, sort);
    Collection<User> users = getUsersPagedAndSortedForCurrentUser(pageable,
userRole).getContent();
    List<UserDto> usersToReturn = new ArrayList<>();
    for (User user : users) { usersToReturn.add(userConverter.toDto(user)); }
    return usersToReturn;
}
```

```
@Override
public UserDto updateUser(
        String username,
        String currentPassword,
        String password,
        String uniqueName,
        String firstname,
        String lastname,
        LocalDate birthday,
        MultipartFile userImage
) throws Exception {
    User user = userRepository
            .findByUsername(username)
            .orElseThrow(
                    () -> new UserNotFoundException("User with username " +
username + " not found")
            );
    if (!passwordEncoder.matches(currentPassword, user.getPassword())) {
        throw new PasswordNotValidException("Passwords do not match for user
with username " + username);
    }
    UserModificationRequest userModificationRequest = UserModificationRequest
            .builder()
            .currentPassword(currentPassword)
            .password(password)
            .uniqueName(uniqueName)
            .firstname(firstname)
            .lastname(lastname)
            .birthday(birthday)
            .userImage(userImage)
            .build();
    user = userModifier.modify(userModificationRequest, user);
    user = userRepository.save(user);
    return userConverter.toDto(user);
}
@Override
public UserDto deleteUser(String username) throws Exception {
    User userToDelete = userRepository
            .findByUsername(username);
    validateUserDeletion(userToDelete);
    UserDto userToDeleteDto = userConverter.toDto(userToDelete);
    messageService.sendMessageToChatOnUserDeletion(userToDeleteDto);
    userRepository.delete(userToDelete);
    return userToDeleteDto;
}
@Override
public List<UserDto> getUserSubscriptions(String username) throws Exception {
    User user = userRepository
            .findByUsername(username)
            .orElseThrow(
                    () -> new UserNotFoundException("User with username " +
username + " not found")
            );
    List<SubscriptionSubscriber> subscriptionSubscribers =
subscriptionSubscriberRepository.findBySubscriberId(user.getId());
    return convertUserStreamToUserDtoList(subscriptionSubscribers
            .stream()
            .map(SubscriptionSubscriber::getSubscription));
}
```

```
@Override
public List<UserDto> getUserSubscribers(String username) throws Exception {
    User user = userRepository
            .findByUsername(username);
    List<SubscriptionSubscriber> subscriptionSubscribers =
subscriptionSubscriberRepository.findBySubscriptionId(user.getId());
    return convertUserStreamToUserDtoList(subscriptionSubscribers
            .stream()
            .map(SubscriptionSubscriber::getSubscriber));
}


private List<UserDto> convertUserStreamToUserDtoList(Stream<User> userStream)
throws Exception {
    List<User> users = userStream.toList();
    List<UserDto> usersToReturn = new ArrayList<>();
    for (User user : users) {
        usersToReturn.add(userConverter.toDto(user));
    }
    return usersToReturn;
}

private Page<User> getUsersPagedAndSortedForCurrentUser(Pageable pageable, Role
userRole) throws UserNotAuthenticatedException {
    UserDetails currentUser =
authenticationService.getAuthenticatedUserUserDetailsFromSecurityContext();
    if (currentUser == null) {
        throw new UserNotAuthenticatedException("User not authenticated");
    }
    String usernameToExclude = currentUser.getUsername();
    return userRepository.findAll((root, query, criteriaBuilder) -> {
        List<Predicate> predicates = new ArrayList<>();
        predicates.add(criteriaBuilder.equal(root.get("role"), userRole));
        predicates.add(criteriaBuilder.notEqual(root.get("username"),
usernameToExclude));
        return
query.where(predicates.toArray(Predicate[]::new)).getRestriction();
    }, pageable);
}

@Override
public Subscription subscribe(String subscriberUniqueName, Subscription
subscription) throws Exception {

    if (!subscriberUniqueName.equals(subscription.getSubscriberUniqueName())) {
        throw new IllegalArgumentException("Unique names of subscriber does not
match");
    }

    SubscriptionSubscriber subscriptionSubscriber =
subscriptionSubscriberConverter.toEntity(subscription);
    if (subscriptionSubscriberRepository.existsBySubscriptionAndSubscriber(
            subscriptionSubscriber.getSubscription(),
            subscriptionSubscriber.getSubscriber()
    )) {
        throw new SubscriptionSubscriberAlreadyExistsException("Subscription
already exists");
    }

    SubscriptionSubscriber savedSubscriptionSubscriber =
subscriptionSubscriberRepository.save(subscriptionSubscriber);
    return subscriptionSubscriberConverter.toDto(savedSubscriptionSubscriber);
}
```

```
@Override
public Subscription unsubscribe(String userUniqueName, String
userSubscriptionUniqueName) throws Exception {
    Subscription subscriptionDto = Subscription
            .builder()
            .subscriptionUniqueName(userSubscriptionUniqueName)
            .subscriberUniqueName(userUniqueName)
            .build();
    SubscriptionSubscriber subscriptionSubscriber =
subscriptionSubscriberConverter.toEntity(subscriptionDto);
    User subscription = subscriptionSubscriber.getSubscription();
    User subscriber = subscriptionSubscriber.getSubscriber();

SubscriptionSubscriber storedSubscriptionSubscriber =
subscriptionSubscriberRepository.findBySubscriptionAndSubscriber(

            subscription,
            subscriber
    );

    if (storedSubscriptionSubscriber == null) {
        throw new SubscriptionSubscriberNotExistsException("User with uniqueName
" + userUniqueName + "is not subscribed on user with uniqueName " +
userSubscriptionUniqueName);
    } else {
        subscriptionSubscriberRepository.delete(storedSubscriptionSubscriber);
    }

    return subscriptionSubscriberConverter.toDto(storedSubscriptionSubscriber);
}
@Override
public RegistrationResponse createAdmin(RegistrationRequest
registrationRequest) throws Exception {
    if (userRepository.existsByUsername(registrationRequest.getUsername())) {
        throw new UserAlreadyExistsException("User with username " +
registrationRequest.getUsername() + " already exists in database");
    } else if
(userRepository.existsByUniqueName(registrationRequest.getUniqueName())) {
        throw new UserAlreadyExistsException("User with uniqueName " +
registrationRequest.getUniqueName() + " already exists in database");
    }
    User userToSave = userRegistrationConverter.toEntity(registrationRequest);
    userToSave.setRole(Role.ADMIN);
    userToSave.setUserAccount(UserAccount
            .builder()
            .user(userToSave)
            .state(AccountState.ACTIVATED)
            .createdAt(ZonedDateTime.now())
            .activatedAt(ZonedDateTime.now())
            .build());
    User savedUser = userRepository.save(userToSave);
    UserDto registeredUserDto = userConverter.toDto(savedUser);
    return RegistrationResponse
            .builder()
            .user(registeredUserDto)
            .isRegistrationSuccessful(true)
            .build();
}

@Component
@RequiredArgsConstructor
public class WebSocketAuthenticationInterceptor implements ChannelInterceptor {
    private final JwtUtil jwtUtil;
```

```
    private final UserDetailsService userDetailsService;

    @Override
    public Message<?> preSend(@NonNull Message<?> message, @NonNull
MessageChannel channel) {
        StompHeaderAccessor accessor =
MessageHeaderAccessor.getAccessor(message, StompHeaderAccessor.class);
        List<StompCommand> stompCommands = List.of(
                StompCommand.CONNECT,
                StompCommand.SUBSCRIBE,
                StompCommand.SEND
        );
        if (accessor == null) {
            throw new IllegalArgumentException("Accessor is null");
        }
            StompCommand receivedStompCommand = accessor.getCommand();
        if (stompCommands.contains(receivedStompCommand)) {
            boolean isJwtValid = false;
            List<String> authorizationHeader =
accessor.getNativeHeader("Authorization");
            if (authorizationHeader != null && !authorizationHeader.isEmpty()) {
                String authorizationHeaderValue = authorizationHeader.get(0);
                if (authorizationHeaderValue != null &&
authorizationHeaderValue.startsWith("Bearer ")) {
                    String jwt = authorizationHeaderValue.substring(7);
                    if (!jwt.isBlank()) {
                        String username = jwtUtil.extractUsername(jwt);
                        UserDetails userDetails =
userDetailsService.loadUserByUsername(username);
                        if (userDetails != null) {
                            try {
                                if (jwtUtil.isTokenValid(jwt, userDetails)) {
                                    isJwtValid = true;
                                }
                            } catch (Exception e) {
                                throw new IllegalArgumentException(e);
                            }
                        }
                    }
                }
            }
            if (!isJwtValid) {
                throw new InvalidTokenException("Jwt is invalid");
            }
        }
        return message;
    }
}
@Configuration
@RequiredArgsConstructor
@EnableWebSocketMessageBroker
public class WebSocketConfig implements WebSocketMessageBrokerConfigurer {

    private final WebSocketAuthenticationInterceptor
webSocketAuthenticationInterceptor;

    @Override
    public void registerStompEndpoints(StompEndpointRegistry registry) {
        registry.addEndpoint("/api/ws")
                .setAllowedOriginPatterns("*")
                .withSockJS();
    }
```

```
    @Override
    public void configureMessageBroker(MessageBrokerRegistry registry) {
        registry.enableSimpleBroker("/api/messaging/topic");
        registry.setApplicationDestinationPrefixes("/api/messaging");
    }
    @Override
    public boolean configureMessageConverters(List<MessageConverter>
messageConverters) {
        DefaultContentTypeResolver resolver = new DefaultContentTypeResolver();
        resolver.setDefaultMimeType(MimeTypeUtils.APPLICATION_JSON);
        MappingJackson2MessageConverter converter = new
MappingJackson2MessageConverter();
        converter.setObjectMapper(new ObjectMapper());
        converter.setContentTypeResolver(resolver);
        messageConverters.add(converter);
        return false;
    }
    @Override
    public void configureClientInboundChannel(ChannelRegistration registration)
{ registration.interceptors(webSocketAuthenticationInterceptor); }
}
```

```
@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
@JsonTypeInfo(use = JsonTypeInfo.Id.NAME, property = "class")
@JsonSubTypes({
        @JsonSubTypes.Type(value = NotificationDto.class, name =
"notification"),
        @JsonSubTypes.Type(value = ChatNotificationDto.class, name =
"chatNotification")
})
public class NotificationDto implements Cloneable {
    private String senderUsername;
    private String receiverUsername;
    private String content;
    private String time;
    private String type;
    public NotificationDto cloneAndSetReceiverUsername(String receiverUsername)
throws CloneNotSupportedException {
        NotificationDto clonedNotificationDto = clone();
        clonedNotificationDto.setReceiverUsername(receiverUsername);
        return clonedNotificationDto;
    }

    @Override
    public NotificationDto clone() throws CloneNotSupportedException {
        return (NotificationDto) super.clone();
    }
}
```

```
@RestController
@RequiredArgsConstructor
public class ChatController {
    private final ChatService chatService;
    private final SimpMessagingTemplate simpMessagingTemplate;
@GetMapping("/api/chats")
@PreAuthorize("hasRole('USER')")
@ResponseStatus(HttpStatus.OK)
public Collection<ChatDto> getCurrentUserChats(
        @RequestParam(required = false) String chatType,
        @RequestParam(required = false) Long
```

```
 numberOfMessagesToRetrieveFromEveryChat
) throws Exception {
    return chatService.getCurrentUserChats(chatType,
numberOfMessagesToRetrieveFromEveryChat);
}

@PostMapping("/api/chats")
@PreAuthorize("hasRole('USER')")
@ResponseStatus(HttpStatus.OK)
public ChatDto createChat(@RequestBody ChatDto chatDto) throws Exception {
    return chatService.createChat(chatDto);
}

@GetMapping("/api/chats/{chatId}")
@PreAuthorize("hasRole('USER')")
@ResponseStatus(HttpStatus.OK)
public ChatDto getChatById(@PathVariable String chatId) throws Exception {
    return chatService.getChatById(chatId);
}

@PatchMapping("/api/chats/{chatId}")
@PreAuthorize("hasRole('USER')")
@ResponseStatus(HttpStatus.OK)
public ChatDto updateChat(
        @PathVariable String chatId,
        @RequestBody ChatDto chatDto
) throws Exception {
    return chatService.updateChat(chatId, chatDto);
}

@DeleteMapping("/api/chats/{chatId}")
@PreAuthorize("hasRole('USER')")
@ResponseStatus(HttpStatus.OK)
public ChatDto deleteChat(@PathVariable String chatId) throws Exception {
    return chatService.deleteChat(chatId);
}

@PostMapping("/api/chats/{chatId}/members/{username}")
@PreAuthorize("hasRole('USER')")
@ResponseStatus(HttpStatus.OK)
public ChatMemberDto addUserToChat(
        @PathVariable String chatId,
        @PathVariable String username,
        @RequestBody UserDto user
) throws Exception {
    return chatService.addUserToChat(chatId, username, user);
}

@PatchMapping("/api/chats/{chatId}/members/{username}")
@PreAuthorize("hasRole('USER')")
public ChatMemberDto updateChatMember(
        @PathVariable String chatId,
        @PathVariable String username,
        @RequestBody ChatMemberDto chatMemberDto
) throws Exception {
    return chatService.updateChatMember(chatId, username, chatMemberDto);
}
```

```
    @DeleteMapping("/api/chats/{chatId}/members/{username}")
    @PreAuthorize("hasRole('USER')")
    public ChatMemberDto deleteChatMember(
            @PathVariable String chatId,
            @PathVariable String username
    ) throws Exception {
        return chatService.deleteChatMember(chatId, username);
    }
        @RestController
    @RequiredArgsConstructor
    @RequestMapping("/api")
    @ResponseStatus(HttpStatus.OK)
    public class MessageController {
        private final MessageService messageService;
        @GetMapping("/chats/{chatId}/messages")
        @PreAuthorize("hasRole('USER')")
        public Collection<MessageDto> getAllMessagesFromChat(@PathVariable String
    chatId) throws Exception {
            return messageService.getAllMessagesFromChat(chatId);
        }
        @PostMapping("/chats/{chatId}/messages")
        @PreAuthorize("hasRole('USER')")
        public MessageDto sendMessageToChat(@RequestBody MessageDto messageDto)
    throws Exception {
            return messageService.sendMessageToChat(messageDto);
        }
        @PatchMapping("/chats/{chatId}/messages")
        @PreAuthorize("hasRole('USER')")
        public ChatMessagesStatusUpdateDto updateMessagesStatusesInChat(
                @PathVariable String chatId,
                @RequestBody ChatMessagesStatusUpdateDto chatMessagesStatusUpdateDto
                ) throws Exception {
            return messageService.updateMessagesStatusesInChat(chatId,
    chatMessagesStatusUpdateDto);
        }
            @PatchMapping("/chats/{chatId}/messages/{messageId}")
        @PreAuthorize("hasRole('USER')")
        public MessageDto updateMessageInChat(
                @PathVariable String chatId,
                @PathVariable String messageId,
                @RequestBody MessageDto message
        ) throws Exception {
            return messageService.updateMessageInChat(chatId, messageId, message);
        }
            @DeleteMapping("/chats/{chatId}/messages/{messageId}")
        @PreAuthorize("hasRole('USER')")
        public MessageDto deleteMessageFromChat(
                @PathVariable String chatId,
                @PathVariable String messageId
        ) throws Exception {
            return messageService.deleteMessageFromChat(chatId, messageId);
        }
    }
        @RestController
    public class NotificationController {
        private final NotificationService notificationService;
        @MessageMapping("/notification")
        public void processAndSendNotificationToUser(@Payload NotificationDto
    notificationDto) {
            notificationService.processAndSendNotificationToUser(notificationDto);
        }
```

```
    @MessageMapping("/notifications")
    public void processAndSendNotificationToUsers(@Payload
MultiUserNotificationDto multiUserNotificationDto)
{   notificationService.processAndSendMultiUserNotification(multiUserNotificat
ionDto);
    }
}
@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
@Entity
@Table(name = "chats")
public class Chat {
    @Id
    @GeneratedValue(strategy = GenerationType.UUID)
    private UUID id;
    @Column(nullable = false)
    private String name;
    @Enumerated(EnumType.STRING)
    private ChatType type;
    @OneToMany(mappedBy = "chat", cascade = CascadeType.ALL)
    private List<ChatMember> members;
    @OneToMany(mappedBy = "chat", cascade = {
            CascadeType.PERSIST,
            CascadeType.MERGE,
            CascadeType.REFRESH,
            CascadeType.DETACH
    })
    private List<Message> messages;
}
@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
@Entity
@Table(name = "chats_members")
public class ChatMember {
    @Id
    @GeneratedValue(strategy = GenerationType.UUID)
    private UUID id;
    @ManyToOne
    @JoinColumn(name = "chat_id", referencedColumnName = "id", nullable =
false)
    private Chat chat;
    @ManyToOne
    @JoinColumn(name = "member_id", referencedColumnName = "id", nullable =
false)
    private User member;
    @Enumerated(EnumType.STRING)
    private MemberRole memberRole;
}
@Data
@Builder
@AllArgsConstructor
@Entity
@Table(name = "messages")
public class Message {
    @Id
    @GeneratedValue(strategy = GenerationType.UUID)
    private UUID id;
```

```
    @ManyToOne
    @JoinColumn(name = "sender_id", referencedColumnName = "id")
    private User sender;
    @ManyToOne
    @JoinColumn(name = "chat_id", referencedColumnName = "id", nullable =
false)
    private Chat chat;
    @Column(nullable = false)
    private String content;
    @Column(nullable = false)
    private ZonedDateTime sendTime;
    @Enumerated(EnumType.STRING)
    private MessageType type;
    @OneToMany(mappedBy = "message", cascade = CascadeType.ALL)
    private List<MessageStatus> statuses;
}

@Data
@Builder
@AllArgsConstructor
@Entity
@Table(name = "messages_statuses")
public class MessageStatus {
    @Id
    @GeneratedValue(strategy = GenerationType.UUID)
    private UUID id;
    @ManyToOne
    @JoinColumn(name = "message_id", referencedColumnName = "id", nullable =
false)
    private Message message;
    @ManyToOne
    @JoinColumn(name = "user_id", referencedColumnName = "id", nullable =
false)
    private User user;
    @Enumerated(EnumType.STRING)
    private UserType userType;
    @Enumerated(EnumType.STRING)
    private Status status;
}

@Repository
public interface ChatMemberRepository extends JpaRepository<ChatMember, UUID> {
    List<ChatMember> findAllByChatId(UUID chatId);
    Optional<ChatMember> findByChatIdAndMemberUsername(UUID chatId, String
username);
    boolean existsByMemberIdAndChatId(UUID memberId, UUID chatId);
    boolean existsByMemberIdAndChatIdAndMemberRole(UUID memberId, UUID chatId,
MemberRole memberRole);
}

@Repository
public interface ChatRepository extends JpaRepository<Chat, UUID> {
    @Query("""
            SELECT c FROM Chat c
            JOIN ChatMember cm ON c.id = cm.chat.id
            JOIN User u ON cm.member.id = u.id
            WHERE u.username = :username
            """)
    List<Chat> findAllChatsByChatMemberUsername(String username);
    @Query("""
            SELECT c FROM Chat c
            JOIN ChatMember cm ON c.id = cm.chat.id
            JOIN User u ON cm.member.id = u.id
```

```
            WHERE u.username = :username AND c.type = :type
            """)
    List<Chat> findAllChatsByChatMemberUsernameAndChatType(String username,
ChatType type);
    @Query("""
            SELECT c FROM Chat c
            JOIN ChatMember cm1 ON c.id = cm1.chat.id
            JOIN User u1 ON cm1.member.id = u1.id AND u1.username
= :firstMemberUsername
            JOIN ChatMember cm2 ON c.id = cm2.chat.id
            JOIN User u2 ON cm2.member.id = u2.id AND u2.username
= :secondMemberUsername
            WHERE c.type = 'PRIVATE_CHAT'
            """)
    Chat findPrivateChatByPrivateChatMembersUsernames(String
firstMemberUsername, String secondMemberUsername);
    @Query("""
            SELECT c FROM Chat c
            JOIN ChatMember cm ON c.id = cm.chat.id
            WHERE c.id = :chatId AND cm.member.id = :chatMemberId
            """)
    Optional<Chat> findByChatIdAndChatMemberId(UUID chatId, UUID chatMemberId);
}

@Repository
public interface MessageRepository extends JpaRepository<Message, UUID> {
    boolean existsByIdAndChatId(UUID id, UUID chatId);
    Optional<Message> findByIdAndChatId(UUID id, UUID chatId);
    Optional<Message> findByIdAndSenderIdAndChatId(UUID id, UUID senderId, UUID
chatId);
    List<Message> findAllByChat(Chat chat);
    @Query("""
            SELECT m FROM Message m
            WHERE m.chat.id = :chatId
            AND m.id NOT IN (SELECT m.id FROM Message m
            JOIN MessageStatus ms ON m.id = ms.message.id
            WHERE m.chat.id = :chatId AND ms.user.id = :userId)
            """)
    List<Message>
findAllMessagesByChatIdThatWereNotSentToUserWithProvidedId(UUID chatId, UUID
userId);
    List<Message> findAllByChatIdOrderBySendTime(UUID chatId);;
}

@Repository
public interface MessageStatusRepository extends JpaRepository<MessageStatus,
UUID> {
    Optional<MessageStatus> findByMessageIdAndUserId(UUID messageId, UUID
userId);
    List<MessageStatus> findAllByChatIdAndUserIdAndStatus(UUID chatId, UUID
userId, Status status);

    @Query("""
            SELECT ms FROM MessageStatus ms
            JOIN Message m ON m.id = ms.message.id
            WHERE m.chat.id = :chatId AND ms.status = :status
            """)
    List<MessageStatus> findAllByChatIdAndStatus(UUID chatId, Status status);
}

@Service
public class ChatServiceImpl implements ChatService {
    private final AuthenticationService authenticationService;
```

```
    private final NotificationService notificationService;
    private final MessageService messageService;
    @Service
    @RequiredArgsConstructor
    public class ChatServiceImpl implements ChatService {
        private final AuthenticationService authenticationService;
        private final NotificationService notificationService;
        private final MessageService messageService;
        private final UserRepository userRepository;
        private final ChatRepository chatRepository;
        private final ChatMemberRepository chatMemberRepository;
        private final ChatConverter chatConverter;
        private final ChatMemberConverter chatMemberConverter;
        @Override
        public Collection<ChatDto> getCurrentUserChats(String chatType, Long
    numberOfMessagesToRetrieveFromEveryChat) throws Exception {
            User currentUser = authenticationService.getCurrentUser();
            String currentUserUsername = currentUser.getUsername();
            ChatType type = null;
            List<Chat> chats;
            if (chatType != null) {
                type = ChatType.valueOf(chatType.toUpperCase());
            }
            if (type != null) {
                chats =
    chatRepository.findAllChatsByChatMemberUsernameAndChatType(currentUserUsername,
    type);
            } else {
                chats =
    chatRepository.findAllChatsByChatMemberUsername(currentUserUsername);
            }
            List<ChatDto> chatsToReturn = new ArrayList<>();
            for (Chat chat : chats) {
                chatsToReturn.add(chatConverter.toDto(chat));
            }
        if (numberOfMessagesToRetrieveFromEveryChat != null) {
            chatsToReturn = modifyNumberOfMessagesInEveryChat(chatsToReturn,
    numberOfMessagesToRetrieveFromEveryChat);
        }
        for (ChatDto chatDto : chatsToReturn) {
            Collection<MessageDto> encryptedMessages = messageService
                    .encryptAllMessagesUsingE2EEWithCurrentUserPublicKey(chatDto.getM
    essages());
            chatDto.setMessages(encryptedMessages);
        }
        return chatsToReturn;
    }
    public ChatDto getChatById(String chatId) throws Exception {
        User currentUser = authenticationService.getCurrentUser();
        UUID userId = currentUser.getId();
        String username = currentUser.getUsername();
        UUID chatUUID = UUID.fromString(chatId);
        Chat chat = chatRepository
                .findByChatIdAndChatMemberId(chatUUID, userId);
        ChatDto chatDtoToReturn = chatConverter.toDto(chat);
        Collection<MessageDto> encryptedMessages = messageService
                .encryptAllMessagesUsingE2EEWithCurrentUserPublicKey(chatDtoToReturn.
    getMessages());
        chatDtoToReturn.setMessages(encryptedMessages);
        return chatDtoToReturn;
    }
```

```
    @Override
    public ChatDto updateChat(String chatId, ChatDto chatDto) throws Exception {
        if (!chatDto.getId().equals(chatId)) {
            throw new IllegalArgumentException("Chat ids are different");
        }
        UUID convertedChatId = UUID.fromString(chatId);
        Chat chat = chatRepository
                .findById(convertedChatId)
                .orElseThrow(
                        () -> new ChatNotFoundException("Chat with id " + chatId + "
    not found")
                );
        User currentUser = authenticationService.getCurrentUser();
        if (!chatMemberRepository.existsByMemberIdAndChatIdAndMemberRole(
                currentUser.getId(),
                chat.getId(),
                MemberRole.ADMIN)
        ) {
            throw new ChatMemberNotFoundException("Chat member with id " +
    currentUser.getId()
                    + " and role " + MemberRole.ADMIN
                    + " not found in chat with id " + chatId);
        }
        String newChatName = chatDto.getName();
        if (newChatName == null || newChatName.isBlank()) {
            throw new IllegalArgumentException("Chat name is null or blank");
        }
        chat.setName(newChatName);
        Chat savedChat = chatRepository.save(chat);
        ChatDto chatDtoToReturn = chatConverter.toDto(savedChat);
        messageService.sendMessageToChatOnChatUpdate(chatDtoToReturn);
        return chatDtoToReturn;
    }

    @Override
    public ChatDto deleteChat(String chatId) throws Exception {
        UUID convertedChatId = UUID.fromString(chatId);
        Chat chat = chatRepository
                .findById(convertedChatId)
                .orElseThrow(
                        () -> new ChatNotFoundException("Chat with id " + chatId + "
    not found")

                );
        User currentUser = authenticationService.getCurrentUser();
        if (!chatMemberRepository.existsByMemberIdAndChatIdAndMemberRole(
                currentUser.getId(),
                chat.getId(),
                MemberRole.ADMIN)
        ) {
            throw new ChatMemberNotFoundException("Chat member with member id " +
    currentUser.getId()
                    + " and member role " + MemberRole.ADMIN
                    + " not found in chat with id  " + chatId);
        }
        ChatDto chatDtoToReturn = chatConverter.toDto(chat);
        chatRepository.deleteById(chat.getId());
        notificationService.notifyUsersOnChatDeletion(chatDtoToReturn);
        return chatDtoToReturn;
    }
    public ChatDto
    getCurrentUserPrivateChatWithAnotherUserByAnotherUserUsername(String
```

```
 anotherUserUsername) throws Exception {
    ChatDto chatDtoToReturn = null;
    User currentUser = authenticationService.getCurrentUser();
    String currentUserUsername = currentUser.getUsername();
    Chat chat =
chatRepository.findPrivateChatByPrivateChatMembersUsernames(currentUserUsername
, anotherUserUsername);
    if (chat != null) {
        chatDtoToReturn = chatConverter.toDto(chat);
        Collection<MessageDto> encryptedMessages = messageService
               .encryptAllMessagesUsingE2EEWithCurrentUserPublicKey(chatDtoToRet
urn.getMessages());
        chatDtoToReturn.setMessages(encryptedMessages);
    }

    return chatDtoToReturn;
}
@Override
public ChatDto createChat(ChatDto chatDto) throws Exception {
    Chat chat = chatConverter.toEntity(chatDto);
    Chat savedChat = chatRepository.save(chat);
    ChatDto chatDtoToReturn = chatConverter.toDto(savedChat);
    notificationService.notifyUsersOnChatCreation(chatDtoToReturn);
    return chatDtoToReturn;
}

@Override
public ChatMemberDto addUserToChat(String chatId, String username, UserDto
user) throws Exception {
    if (!username.equals(user.getUsername())) {
        throw new IllegalArgumentException("Usernames are different");
    }
    UUID convertedChatId = UUID.fromString(chatId);
    Chat chat = chatRepository
           .findById(convertedChatId)
           .orElseThrow(
                  () -> new ChatNotFoundException("Chat with id " + chatId + "
not found")
           );

    if (chat.getType().equals(ChatType.PRIVATE_CHAT)) {
        throw new IllegalArgumentException("User can not be added to private
chat");
    }
    User userToAdd = userRepository
           .findByUsername(username);
    if (chatMemberRepository.existsByMemberIdAndChatId(userToAdd.getId(),
chat.getId())) {
        throw new ChatMemberAlreadyExistsException("Chat member with username "
               + userToAdd.getUsername() + " already exists in chat with id " +
chatId);
    }

        ChatMember chatMember = ChatMember
            .builder()
            .chat(chat)
            .member(userToAdd)
            .memberRole(MemberRole.MEMBER)
            .build();

        ChatMember savedChatMember = chatMemberRepository.save(chatMember);

        messageService.createMessagesStatusesForUserInChat(
```

```
                username,
                UserType.RECEIVER,
                chat.getId(),
                Status.READ_MESSAGE
        );
        ChatMemberDto chatMemberToReturn =
    chatMemberConverter.toDto(savedChatMember);
        messageService.sendMessageToChatOnActionOnUserInChat(chatMemberToReturn,
    MessageType.CHAT_MEMBER_ADDED_TO_CHAT);
        notificationService.notifyChatMemberOnActionOnChatMemberInChat(
                chatConverter.toDto(chat),
                chatMemberToReturn,
                NotificationType.USER_ADDED_TO_CHAT_NOTIFICATION
        );
        return chatMemberToReturn;
    }
    public ChatMemberDto updateChatMember(String chatId, String username,
    ChatMemberDto chatMemberDto) throws Exception {
        UserDto userDto = chatMemberDto.getUser();
        if (!userDto.getUsername().equals(username)) {
            throw new IllegalArgumentException("Usernames are different");
        }
        if (!chatMemberDto.getChatId().equals(chatId)) {
            throw new IllegalArgumentException("Chat ids are different");
        }
        UUID convertedChatId = UUID.fromString(chatId);
        Chat chat = chatRepository
                .findById(convertedChatId)
                .orElseThrow(
                        () -> new ChatNotFoundException("Chat with id " + chatId + "
    not found")
                );
        ChatMember chatMember = chatMemberRepository
                .findByChatIdAndMemberUsername(chat.getId(), username);
        if (chatMemberDto.getRole() == null) {
            throw new IllegalArgumentException("Chat member role is null");
        }
        chatMember.setMemberRole(chatMemberDto.getRole());
        ChatMember savedChatMember = chatMemberRepository.save(chatMember);
        ChatMemberDto chatMemberDtoToReturn =
    chatMemberConverter.toDto(savedChatMember);
        messageService.sendMessageToChatOnActionOnUserInChat(chatMemberDtoToReturn,
    MessageType.NEW_STATUS_IN_CHAT_MEMBER);
        notificationService.notifyChatMemberOnActionOnChatMemberInChat(
                chatConverter.toDto(chat),
                chatMemberDtoToReturn,
                NotificationType.NEW_ADMIN_IN_CHAT_NOTIFICATION
        );
        return chatMemberDtoToReturn;
    }
    @Override
    public ChatMemberDto deleteChatMember(String chatId, String username) throws
    Exception {
        UUID convertedChatId = UUID.fromString(chatId);
        Chat chat = chatRepository
                .findById(convertedChatId)
                .orElseThrow(
                        () -> new ChatNotFoundException("Chat with id " + chatId + "
    not found")
                );
        if (chat.getType().equals(ChatType.PRIVATE_CHAT)) {
```

```
            throw new ChatModificationException("Users can not be deleted from
private chats");
        }
        ChatMember chatMemberToDelete = chatMemberRepository
                .findByChatIdAndMemberUsername(chat.getId(), username)
                .orElseThrow(
                        () -> new ChatMemberNotFoundException("Chat member with
username " + username
                                + " not found in chat with id " + chatId)
                );
        User userToDeleteFromChat = chatMemberToDelete.getMember();
        User currentUser = authenticationService.getCurrentUser();
        chatMemberRepository.delete(chatMemberToDelete);
        ChatMemberDto chatMemberDtoToReturn =
chatMemberConverter.toDto(chatMemberToDelete);
            ChatMemberDto chatMemberDtoToReturn =
chatMemberConverter.toDto(chatMemberToDelete);
            NotificationType notificationType =
NotificationType.DELETED_CHAT_MEMBER_NOTIFICATION;
            MessageType messageType = MessageType.CHAT_MEMBER_DELETED_FROM_CHAT;
            if
(currentUser.getUsername().equals(chatMemberDtoToReturn.getUser().getUsername()
)) {
                notificationType = NotificationType.MEMBER_LEFT_CHAT_NOTIFICATION;
                messageType = MessageType.CHAT_MEMBER_LEFT_CHAT;
            }
            List<ChatMember> chatMembers =
chatMemberRepository.findAllByChatId(chat.getId());
            if (chatMembers.isEmpty()) {
                notificationType = NotificationType.DELETED_GROUP_CHAT_NOTIFICATION;
                chatRepository.deleteById(convertedChatId);
            }
            notificationService.notifyChatMemberOnActionOnChatMemberInChat(
                    chatConverter.toDto(chat),
                    chatMemberDtoToReturn,
                    notificationType
            );
            if
(!notificationType.equals(NotificationType.DELETED_GROUP_CHAT_NOTIFICATION)) {

messageService.sendMessageToChatOnActionOnUserInChat(chatMemberDtoToReturn,
messageType);
            }
            return chatMemberDtoToReturn;
        }

@Service
@RequiredArgsConstructor
public class MessageServiceImpl implements MessageService {
    private final UserRepository userRepository;
    private final UserConverter userConverter;
    private final MessageRepository messageRepository;
    private final MessageStatusRepository messageStatusRepository;
    private final MessageConverter messageConverter;
    private final ChatRepository chatRepository;
    private final ChatMemberRepository chatMemberRepository;
    private final AuthenticationService authenticationService;
    private final SimpMessagingTemplate simpMessagingTemplate;
    private final EncryptionService encryptionService;
    private final E2EEService e2eeService;
```

```
    @Override
    public Collection<MessageDto> getAllMessagesFromChat(String chatId) throws
    Exception {
        UUID chatUUID = UUID.fromString(chatId);
        Chat chat = chatRepository
                .findById(chatUUID)
                .orElseThrow(
                        () -> new ChatNotFoundException("Chat with id: " + chatId +
    "does not exist")
                );
        List<Message> messages = messageRepository.findAllByChat(chat);
        List<MessageDto> messagesToReturn = new ArrayList<>();
        for (Message message : messages) {
            messagesToReturn.add(messageConverter.toDto(message));
        }
        return messagesToReturn;
    }
    @Override
    public MessageDto sendMessageToChat(MessageDto messageDto) throws Exception {
        if (!messageDto.getType().equals(MessageType.NEW_MESSAGE)) {
            throw new IllegalArgumentException("Invalid message type");
        }
    String decryptedText = e2eeService.decrypt(messageDto.getContent());
    messageDto.setContent(decryptedText);
    Message message = messageConverter.toEntity(messageDto);
    Message savedMessage = messageRepository.save(message);
    UUID senderId = savedMessage.getSender().getId();
    UUID chatId = savedMessage.getChat().getId();
    List<ChatMember> chatMembers = chatMemberRepository.findAllByChatId(chatId);
        List<MessageStatus> messageStatuses = new ArrayList<>();
        for (ChatMember chatMember : chatMembers) {
            User user = chatMember.getMember();
            UUID userId = user.getId();
            UserType userType = UserType.RECEIVER;
            if (userId.equals(senderId)) {
                userType = UserType.SENDER;
            }
            messageStatuses.add(MessageStatus
                    .builder()
                    .message(savedMessage)
                    .user(user)
                    .userType(userType)
                    .status(Status.UNREAD_MESSAGE)
                    .build());
        }
        messageStatusRepository.saveAll(messageStatuses);
        MessageDto messageToSend = messageConverter.toDto(savedMessage);
        String encryptedText = e2eeService.encrypt(messageToSend.getContent());
        messageToSend.setContent(encryptedText);
        processAndSendMessageToChat(messageToSend);
        return MessageDto
                .builder()
                .id(messageToSend.getId())
                .sender(messageToSend.getSender())
                .chatId(messageDto.getChatId())
                .content(messageToSend.getContent())
                .sendTime(messageToSend.getSendTime())
                .type(messageToSend.getType())
                .status(Status.UNREAD_MESSAGE)
                .build();
    }
```

```
public MessageDto updateMessageInChat(String chatId, String messageId,
MessageDto messageDto) throws Exception {
    MessageDto messageDtoToReturn = null;
    if (!messageDto.getType().equals(MessageType.MODIFIED_MESSAGE)) {
        throw new IllegalArgumentException("Illegal message type");
    }
    UUID convertedChatId = UUID.fromString(chatId);
    UUID convertedMessageId = UUID.fromString(messageId);

    if (!chatRepository.existsById(convertedChatId)) {
        throw new ChatNotFoundException("Chat with id " + chatId + " not
found");
    }
    User currentUser = authenticationService.getCurrentUser();
    Message message = messageRepository
            .findByIdAndSenderIdAndChatId(convertedMessageId,
currentUser.getId(), convertedChatId)
            .orElseThrow(
                    () -> new MessageNotFoundException("Message with id " +
messageId
                            + " and sender id " + currentUser.getId()
                            + " not found in chat with id " + chatId)
            );
String decryptedText = e2eeService.decrypt(messageDto.getContent());
String encryptedContent = encryptionService.encrypt(decryptedText);
String previousEncryptedContent = message.getContent();
if (encryptedContent.equals(previousEncryptedContent)) {
    throw new IllegalArgumentException("New message content is equal to the
previously saved message content");
}
    message.setContent(encryptedContent);
    Message savedMessage = messageRepository.save(message);
    messageDtoToReturn = messageConverter.toDto(savedMessage);
    String encryptedText =
e2eeService.encrypt(messageDtoToReturn.getContent());
    messageDtoToReturn.setContent(encryptedText);
    processAndSendMessageToChat(messageDtoToReturn);
    return messageDtoToReturn;
}
public MessageDto deleteMessageFromChat(String chatId, String messageId) throws
Exception {
    MessageDto messageDtoToReturn = null;
    UUID convertedChatId = UUID.fromString(chatId);
    UUID convertedMessageId = UUID.fromString(messageId);
    if (!chatRepository.existsById(convertedChatId)) {
        throw new ChatNotFoundException("Chat with id " + chatId + " not
found");
    }
    Message message = messageRepository
            .findByIdAndChatId(convertedMessageId, convertedChatId);
    User messageSender = message.getSender();
    User currentUser = authenticationService.getCurrentUser();
    ChatMember chatMember = chatMemberRepository
            .findByChatIdAndMemberUsername(convertedChatId,
currentUser.getUsername());
        if (chatMember.getMemberRole().equals(MemberRole.ADMIN) ||
currentUser.getId().equals(messageSender.getId())) {
        messageDtoToReturn = messageConverter.toDto(message);
        String encryptedText =
e2eeService.encrypt(messageDtoToReturn.getContent());
        messageDtoToReturn.setContent(encryptedText);
        messageDtoToReturn.setType(MessageType.DELETED_MESSAGE);
```

```
            messageRepository.deleteById(message.getId());
            processAndSendMessageToChat(messageDtoToReturn);
        }
        return messageDtoToReturn;
    }
    @Override
    public Collection<MessageDto>
    encryptAllMessagesUsingE2EEWithCurrentUserPublicKey(Collection<MessageDto>
    messagesToEncrypt) throws Exception {
        try {
            for (MessageDto messageDto : messagesToEncrypt) {
                String content = messageDto.getContent();
                String encryptedContent = e2eeService.encrypt(content);
                messageDto.setContent(encryptedContent);
            }
        } catch (E2EEKeyNotFoundException e) {
            messagesToEncrypt.forEach(messageDto -> messageDto.setContent(null));
        }
        return messagesToEncrypt;
    }
    @Async
    public void processAndSendMessageToChat(MessageDto messageDto) {
        String chatId = messageDto.getChatId();
        simpMessagingTemplate.convertAndSend(
                "/api/messaging/topic/chats/" + chatId + "/messages",
                MessageDto
                        .builder()
                        .id(messageDto.getId())
                        .sender(messageDto.getSender())
                        .chatId(chatId)
                        .content(messageDto.getContent())
                        .sendTime(messageDto.getSendTime())
                        .type(messageDto.getType())
                        .status(Status.UNREAD_MESSAGE)
                        .build()
        );
    }
    @Slf4j
    @Service
    @RequiredArgsConstructor
    public class NotificationServiceImpl implements NotificationService {
        private final SimpMessagingTemplate simpMessagingTemplate;
        private final AuthenticationService authenticationService;
        private final UserConverter userConverter;
        private final UserRepository userRepository;
        @Async
        public void processAndSendNotificationToUser(NotificationDto
    notificationDto) {
            NotificationDto notificationDtoToBeSendToUser =
    processAndBuildNotificationDtoToBeSendToUser(notificationDto);
            if (notificationDtoToBeSendToUser != null) {
                String username =
    notificationDtoToBeSendToUser.getReceiverUsername();
                if (username != null) {
                    simpMessagingTemplate.convertAndSend(
                            "/api/messaging/topic/" + username + "/notifications",
                        notificationDtoToBeSendToUser
                );
            }
        }
    }
```

```
@Async
public void processAndSendNotificationToUsers(NotificationDto notificationDto,
Collection<UserDto> usersToNotify) {
    if (notificationDto != null) {
        for (UserDto userToNotify : usersToNotify) {
            notificationDto.setReceiverUsername(userToNotify.getUsername());
            processAndSendNotificationToUser(notificationDto);
        }
    }
}
@Override
@Async
public void processAndSendMultiUserNotification(MultiUserNotificationDto
multiUserNotificationDto) {
    NotificationDto notificationDto =
multiUserNotificationDto.getNotification();
    List<UserDto> usersToNotify = multiUserNotificationDto.getUsersToNotify();
    processAndSendNotificationToUsers(notificationDto, usersToNotify);}
```

```
@SpringBootApplication
@EnableAsync
public class MessengerApplication {
    public static void main(String[] args) {
       SpringApplication.run(MessengerApplication.class, args);
    }
}
```

```
export default function NavBar() {
    const { user, setUser } = useAppContext();
    const navigate = useNavigate();
    const location = useLocation();
    const items = [];
    const [itemsToShow, setItemsToShow] = useState([]);
    useEffect(() => {
        if (user.authenticated) {
            const uniqueName = user.uniqueName;
            let userProfileRoute = USER_ROUTE;
            let role = user.role;
            switch (role) {
                case Role.USER:
                    userProfileRoute = USER_ROUTE;
                    setItemsToShow([
                        { name: "Users", href: USER_ROUTE },
                        { name: "Chats", href: userProfileRoute + "/" + uniqueName +
"/chats" },
                        { name: "Profile", href: userProfileRoute + "/" + uniqueName }
                    ]);
                    break;
                case Role.ADMIN:
                    userProfileRoute = ADMIN_ROUTE;
                    setItemsToShow([
                        { name: "Users", href: USER_ROUTE },
                        { name: "Admins", href: ADMIN_ROUTE },
                        { name: "Profile", href: userProfileRoute + "/" + uniqueName }
                    ]);
                    break;
                case Role.ROOT:
                    userProfileRoute = ROOT_ROUTE;
                    setItemsToShow([
                        { name: "Users", href: USER_ROUTE },
                        { name: "Profile", href: userProfileRoute + "/" + uniqueName }
                    ]);
```

```
                    break;
                default:
                    userProfileRoute = USER_ROUTE;
            }
        } else {
            setItemsToShow([
                ...items,
                { name: "Home", href: HOME_ROUTE },
                { name: "Sign In", href: SIGH_IN_ROUTE },
                { name: "Sign Up", href: SIGN_UP_ROUTE }
            ]);
        }
    }, [user])
    const [menuButtonClick, setMenuButtonClick] = useState(false);
    const handleMenuButtonClick = () => setMenuButtonClick(!menuButtonClick);
    const onNavItemClick = (e) => setMenuButtonClick(false);
    const onLogOut = async () => {
        let logoutConfirmed = window.confirm("Do you want to log out?")
        if (logoutConfirmed) {
            await logout();
            setUser({
                username: "",
                uniqueName: "",
                authenticated: false,
                role: Role.VISITOR
            });
            navigate(HOME_ROUTE);
        }
    }
    return (
        <header>
            <nav className="navbar">
                <div className="navbar-container">
                    <Link to="/" className="navbar-logo" onClick={onNavItemClick}>
                        <FontAwesomeIcon icon={faMessage} className="navbar-logo-img"
/>
                        <h1>WebTalk</h1>
                    </Link>
                    <div className="nav-menu">
                        <div className={menuButtonClick ? "nav-items items-sm" : "nav-
items"}>
                            {itemsToShow.map(item => (
                                <NavLink
                                    key={item.name}
                                    to={item.href}
                                    className={({ isActive }) => {
                                        return isActive && (location.pathname ===
item.href) ? "nav-item active-item" : "nav-item"
                                    }}
                                    onClick={onNavItemClick}>
                                    {item.name}
                                </NavLink>
                            ))}
                            {
                                user.authenticated &&
                                <div className="logout-button-container">
                                    <div className="logout-button"
onClick={onLogOut}>Log Out</div>
                                </div>
```

```
                        }
                    </div>
                    <div className="menu-toggler-button"
onClick={handleMenuButtonClick}>
                        {
                          menuButtonClick ? (
                           <FontAwesomeIcon icon={faXmark} className="menu-icon" />
                            ) : (
                            <FontAwesomeIcon icon={faBars} className="menu-icon" />
                            )
                        }
                    </div>
                </div>
            </div>
        </nav>
    </header>
  );
}
export default function AccountActivationPage() {
    const { setInformMessage } = useAppContext();
    const navigate = useNavigate();
    const [digit1, setDigit1] = useState("");
    const [digit2, setDigit2] = useState("");
    const [digit3, setDigit3] = useState("");
    const [digit4, setDigit4] = useState("");
    const [digit5, setDigit5] = useState("");
    const [digit6, setDigit6] = useState("");
    const handleChangeDigit1 = (event)=>setDigit1(event.target.value.substring(0, 1));
    const handleChangeDigit2 = (event)=>setDigit2(event.target.value.substring(0, 1));
    const handleChangeDigit3 = (event)=>setDigit3(event.target.value.substring(0, 1));
    const handleChangeDigit4 = (event)=>setDigit4(event.target.value.substring(0, 1));
    const handleChangeDigit5 = (event)=>setDigit5(event.target.value.substring(0, 1));
    const handleChangeDigit6 = (event)=>setDigit6(event.target.value.substring(0, 1));
    const onConfirmAccountActivationButtonClick = async () => {
        if (digit1 && digit2 && digit3 && digit4 && digit5 && digit6) {
            let username = localStorage.getItem("registratingUserEmail");
            if (username) {
                let digits = [digit1, digit2, digit3, digit4, digit5, digit6];
                let activationCode = digits.map(String).join("");
                let response = await activateUserAccount(username, activationCode);
                let data = response?.data;
                if (data) {
                    navigate(SIGH_IN_ROUTE);
                    localStorage.removeItem("registratingUserEmail");
                    setInformMessage("Account is activated");
                } else {
                    setInformMessage("Something went wrong! If you have not activated
your account but see this message, try to resend the activation code or recreate your
account");
                }
            }
        } else {
            setInformMessage("Please provide the code");
        }
    }
    return (
        <div className="account-activation-page">
            <div className="account-activation-page-container">
                <h1 className="account-activation-page-heading">Activation code</h1>
```

```
                <div className="account-activation-page-code-container">
                    <input className="account-activation-page-digit" value={digit1}
onChange={handleChangeDigit1} type="text" maxLength={1} />
                    <input className="account-activation-page-digit" value={digit2}
onChange={handleChangeDigit2} type="text" maxLength={1} />
                    <input className="account-activation-page-digit" value={digit3}
onChange={handleChangeDigit3} type="text" maxLength={1} />
                    <input className="account-activation-page-digit" value={digit4}
onChange={handleChangeDigit4} type="text" maxLength={1} />
                    <input className="account-activation-page-digit" value={digit5}
onChange={handleChangeDigit5} type="text" maxLength={1} />
                    <input className="account-activation-page-digit" value={digit6}
onChange={handleChangeDigit6} type="text" maxLength={1} />
                </div>
                <div className="account-activation-page-button-container">
                    <button onClick={onConfirmAccountActivationButtonClick}
className="account-activation-page-button">Confirm</button>
                </div>
            </div>
        </div>
    );
}
export default function LoginPage() {
    const navigate = useNavigate();
    const { setUser, setInformMessage } = useAppContext();
    const onSubmit = async (values, actions) => {
        if (!values.username) {
            setInformMessage("Provide username");
            return;
        } else if (!values.password) {
            setInformMessage("Provide password");
            return;
        }
        const data = await authenticate(
            values.username,
            values.password
        );
        actions.resetForm();
if (data?.accessToken && data?.refreshToken) {
            const username = jwtDecode(data.accessToken).sub;
            const role = jwtDecode(data.accessToken).role;
            if (username && role) {
                const response = await getUserByUsernameAndRole(username, role);
                if (response != null && response?.data?.username === username) {
                    const uniqueName = response.data.uniqueName;
                    setUser({
                        username: jwtDecode(data.accessToken).sub,
                        uniqueName: uniqueName,
                        authenticated: true,
                        role: role
                    });
                    generateKeyPair();
                }
            }
        } else {
            setInformMessage("Can not sign in. Check input data and try again!");
        }
    }
```

```
    return (
        <Formik
            initialValues={{
                username: "",
                password: ""
            }}
            validationSchema={authenticationSchema}
            onSubmit={onSubmit}>
            {({ isSubmitting }) => (
                <div className="authPage">
                    <div className="authForm">
                        <Form>
                            <FormItem
                                label="Email:"
                                id="username"
                                name="username"
                                type="email"
                                placeholder="johnsmith@gmail.com"/>
                            <FormItem
                                label="Password:"
                                id="password"
                                name="password"
                                type="password"
                                placeholder="3245!mySuperSecurePassword$8976"
                            />
                            <button disabled={isSubmitting} type="submit">Sign
In</button>
                            <div className="sign-item">
                                <span className="sign-item-text">Don't have an accout
yet?</span>
                                <Link to={SIGN_UP_ROUTE}>Sign Up</Link>
                            </div>
                        </Form>
                    </div>
                </div>
            )}
        </Formik>
    );}
export default function RegistrationPage() {
    const { setInformMessage } = useAppContext();
    const [isRegistrationSuccessful, setRegistrationSuccessful] = useState(false);
    const onSubmit = async (values, actions) => {
        const data = await register(
            values.username,
            values.password,
            values.uniqueName,
            values.firstname,
            values.lastname,
            values.birthday
        );
        actions.resetForm();
        if (data) {
            let user = data?.user;
            let isRegistrationSuccessful = data?.registrationSuccessful;
            if (user && isRegistrationSuccessful) {
                setRegistrationSuccessful(true);
            }}}
       return (
        isRegistrationSuccessful
```

```
? <AfterUserRegistrationPage />
: <Formik
    initialValues={{
        username: "",
        password: "",
        confirmedPassword: "",
        uniqueName: "",
        firstname: "",
        lastname: "",
        birthday: ""
    }}
    validationSchema={registrationSchema}
    onSubmit={onSubmit}
>
    {({ isSubmitting }) => (
        <div className="authPage">
            <div className="authForm">
                <h1>Sign Up</h1>
                <Form>
                    <FormItem
                        label="Email:"
                        id="username"
                        name="username"
                        type="email"
                        placeholder="johnsmith@gmail.com"
                    />
                    <FormItem
                        label="Password:"
                        id="password"
                        name="password"
                        type="password"
                        placeholder="3245!mySuperSecurePassword$8976"/>
                    <FormItem
                        label="Confirm password:"
                        id="confirmedPassword"
                        name="confirmedPassword"
                        type="password"
                        placeholder="3245!mySuperSecurePassword$8976"
                    />
                   <FormItem
                       label="Username:"
                       id="uniqueName"
                       name="uniqueName"
                       type="text"
                       placeholder="JohnSmith"
                   />
                   <FormItem
                       label="First name:"
                       id="firstname"
                       name="firstname"
                       type="text"
                       placeholder="John"
                   />
                   <FormItem
                       label="Last name:"
                       id="lastname"
                       name="lastname"
                       type="text"
                       placeholder="Smith"/>
```

```
                                <FormItem
                                    label="Birthday:"
                                    id="birthday"
                                    name="birthday"
                                    type="date"
                                    placeholder=""
                                />
                                <button disabled={isSubmitting} type="submit">Sign
Up</button>
                                <div className="sign-item">
                                    <span className="sign-item-text">Already have an
account?</span>
                                    <Link to={SIGH_IN_ROUTE}>Sign In</Link>
                                </div>
                            </Form>
                        </div>
                    </div>
                )}
            </Formik>
    );
}
export default function FormItem({ label, ...props }) {
    const [field, meta] = useField(props);
    return (
        <div className="formItem">
            <label htmlFor={props.id}>{label}</label>
            <input
                {...props}
                {...field}
                value={field.value ? field.value : ""}
                className={meta?.error && field?.value ? "not-valid-input" : ""}
            />
            <div className={meta?.error && field?.value ? "display-item-error-
validation-message" : "hide-item-error-validation-message"}>
                <FontAwesomeIcon icon={faInfoCircle} />
                <span className="item-error-message">{meta.error}</span>
            </div>
        </div>
    );
}
export default function ChatCreationPage() {
    const { user, setInformMessage } = useAppContext();
    const { userProfile } = useUserContext();
    const navigate = useNavigate();
    const [chosenUsers, setChosenUsers] = useState([]);
    const onSubmit = async (values, actions) => {
        let chatName = values.chatName;
        if (chosenUsers.length !== 0 && values.chatName) {
            let usersToAddToChat = [];
            for (let i = 0; i < chosenUsers.length; i++) {
                usersToAddToChat[i] = {
                    chatId: null,
                    user: chosenUsers[i],
                    role: ChatMemberRole.MEMBER
                }
            }
            const response = await createChat(
                chatName,
```

```
                    ChatType.GROUP_CHAT,
                    usersToAddToChat
                );
                let chat = response?.data;
                if (chat) {
                    actions.resetForm();
                    setInformMessage("Chat '" + chat.name + "' was successfully created");
                    navigate(USER_ROUTE + "/" + user.uniqueName + "/chats");
                } else {
                    actions.resetForm();
                }
            }
        }
        return (
            user.username === userProfile.username
                ? (
                    <Formik
                        initialValues={{
                            name: "",
                            type: ChatType.GROUP_CHAT,
                            members: [],
                        }}
                        validationSchema={chatCreationSchema}
                        onSubmit={onSubmit}
                    >
                        {({ isSubmitting }) => (
                            <div className="chat-creation-page-container">
                                <div className="chat-creation-form">
                                    <h1>CREATE CHAT</h1>
                                    <Form>
                                        <FormItem
                                            label="Chat name:"
                                            id="chatName"
                                            name="chatName"
                                            type="text"
                                            placeholder="Write chat name here..."/>
                                        <div className="filtered-user-items-container">
                                            {
                                                chosenUsers.map((chosenUser) =>
<ChatCreationUserItem key={chosenUser.username} chosenUser={chosenUser}
chosenUsers={chosenUsers} setChosenUsers={setChosenUsers} />)
                                            }
                                        </div>
                                        <button disabled={isSubmitting}
type="submit">Create</button>
                                        <Search chosenUsers={chosenUsers}
setChosenUsers={setChosenUsers} />
                                    </Form>
                                </div></div>
                        )}
                    </Formik>
                )
                : <NotFoundPage />
        );
}
export default function ChatPage() {
    const params = useParams();
    const { user } = useAppContext();
    const navigate = useNavigate();
```

```
    const { userChats } = useAuthContext();
    const [isLoading, setLoading] = useState(true);
    const [chatExists, setChatExists] = useState(false);
    const [chat, setChat] = useState(null);
    const [isMemberTabOpened, setMemberTabOpened] = useState(false);
    const [isAdmin, setAdmin] = useState(false);
    useEffect(() => {
        async function getCurrentChat() {
            let chatId = params?.chatId;
            let response = await getChatById(chatId);
            if (response === null) {
                navigate(USER_ROUTE + "/" + user?.uniqueName + "/chats");
            } else {
                let returnedChat = response?.data;
                if (chatId === returnedChat?.id) {
                    await updateMessagesStatusesInChat(returnedChat.id,
MessageStatus.READ_MESSAGE);
                    setChat(returnedChat);
                    setChatExists(true);
                }
                setLoading(false);
            }
        }
        getCurrentChat();
    }, [userChats]);

    return (
        isLoading
            ? <LoadingPage />
            : chatExists && user?.role !== Role.ADMIN && user?.role !== Role.ROOT
                ? (
                    <div className="chat-page-container">
                        <div className="chat-page-buttons-container">
                            <div className="chat-page-buttons">
                                <div className={isMemberTabOpened ? "chat-page-
messages-button" : "chat-page-messages-button chat-page-button-color"} onClick={() =>
setMemberTabOpened(false)}>Messages</div>
                                <div className={isMemberTabOpened ? "chat-page-members-
button chat-page-button-color" : "chat-page-members-button"} onClick={() =>
setMemberTabOpened(true)}>Members</div>
                            </div>
                        </div>
                        <div className="chat-page-content">
                            <ChatPageMessagesArea chat={chat} setChat={setChat}
isMemberTabOpened={isMemberTabOpened} isAdmin={isAdmin} />
                            <ChatPageInfo chat={chat}
isMemberTabOpened={isMemberTabOpened} isAdmin={isAdmin} setAdmin={setAdmin} />
    </div>
 </div>
                )
                : <NotFoundPage />
    );
}
  export default function ChatPageMessagesArea(props) {
    const chatPageMessagesArea = useRef();
    const { chat, setChat, isMemberTabOpened, isAdmin } = props;
    const [textValue, setTextValue] = useState("");
    const [updatedMessage, setUpdatedMessage] = useState(null);
    const [isMessageUpdating, setMessageUpdating] = useState(false);
```

```
    useEffect(() => {
        chatPageMessagesArea.current.scrollTop =
chatPageMessagesArea.current.scrollHeight;
    }, [chat]);
    useEffect(() => {
        let content = updatedMessage?.content;
        if (content) {
            setTextValue(content);
        } else {
            setTextValue("");
        }
    }, [updatedMessage])
    const onCancelMessageUpdatingButtonClick = () => {
        setMessageUpdating(false);
        setUpdatedMessage(null);
    }
    const onConfirmMessageUpdatingButtonClick = async () => {
        let chatId = chat.id;
        let messageId = updatedMessage?.id;
        if (textValue && chatId && messageId && textValue != updatedMessage?.content)
{
            let aesKey = localStorage.getItem("aes-key");
            if (aesKey) {
                let rawKey = CryptoJS.enc.Base64.parse(aesKey);
                let encryptedText = CryptoJS.AES.encrypt(textValue, rawKey, { mode:
CryptoJS.mode.ECB, padding: CryptoJS.pad.Pkcs7 });
                let encryptedData =
encryptedText.ciphertext.toString(CryptoJS.enc.Base64);
                let messageId = updatedMessage?.id;
                let message = {
                    id: updatedMessage?.id,
                    sender: updatedMessage?.sender,
                    chatId: updatedMessage?.chatId,
                    content: encryptedData,
                    sendTime: updatedMessage?.sendTime,
                    type: updatedMessage?.type,
                    status: updatedMessage?.status
                }
                let response = await updateMessageInChat(chatId, messageId, message);
                let data = response?.data;
                if (data) {
                    setMessageUpdating(false);
                }
                setTextValue("");
            }
        }}}
const onSendMessageButtonClick = async () => {
        let chatId = chat.id;
        if (textValue && chatId) {
            let aesKey = localStorage.getItem("aes-key");
            if (aesKey) {
                let rawKey = CryptoJS.enc.Base64.parse(aesKey);
                let encryptedText = CryptoJS.AES.encrypt(textValue, rawKey, { mode:
CryptoJS.mode.ECB, padding: CryptoJS.pad.Pkcs7 });
                let encryptedData =
encryptedText.ciphertext.toString(CryptoJS.enc.Base64);
                let response = await sendMessageToChat(chatId, encryptedData,
MessageType.NEW_MESSAGE);
                let sentMessage = response?.data;}}}
```

```
    return (
        <div className={isMemberTabOpened ? "hide-chat-page-messages-area-container" :
"chat-page-messages-area-container"}>
            <div className="chat-page-messages-area" ref={chatPageMessagesArea}>
                {
                    chat.messages.map((message) => {
                        return <ChatPageMessage
                            key={message.sendTime}
                            chat={chat}
                            setChat={setChat}
                            message={message}
                            setUpdatedMessage={setUpdatedMessage}
                            isMessageUpdating={isMessageUpdating}
                            setMessageUpdating={setMessageUpdating}
                            isAdmin={isAdmin}
                        />
                    })
                }
            </div>
            <div className="chat-page-write-message-area-container">
                <div className="chat-page-write-message-area">
                    <textarea
                        id="message"
                        name="message"
                        value={textValue}
                        placeholder='Type here...'
                        onChange={(event) => setTextValue(event.target.value)}
                        className="chat-page-write-message-textarea"
                    ></textarea>
                    {
                        isMessageUpdating
                            ? <>
                                <div className="chat-page-send-message-button"
onClick={onCancelMessageUpdatingButtonClick}>Cancel</div>
                                <div className="chat-page-send-message-button"
onClick={onConfirmMessageUpdatingButtonClick}>Update</div>
                            </>
                            : <div onClick={onSendMessageButtonClick}>Send</div>
                    }
                </div></div></div>
    );
}
export default function UserProfilePage() {
  const { user } = useAppContext();
  const { userProfile } = useUserContext();
  const [follow, setFollow] = useState(false);
  const [isLoading, setLoading] = useState(true);
  useEffect(() => {
    async function checkSubscription() {
      let response = await isSubscribed(
        userUniqueName,
        subscriptionUniqueName
      )
      if (response?.data?.subscribed) {
        setFollow(true);
      } else {
        setFollow(false);
      }
    }
```

```
    let userUniqueName = user.uniqueName;
    let subscriptionUniqueName = userProfile.uniqueName;
    if (userUniqueName !== subscriptionUniqueName && userUniqueName !== "" &&
subscriptionUniqueName !== "" && user.role !== Role.ADMIN && user.role !== Role.ROOT)
{
      checkSubscription();
    }
    setLoading(false);
  }, [userProfile, user]);
  return (
    isLoading
      ? <LoadingPage />
      : (
        <div className="profile-page">
          <ProfileInfo />
          <ProfileActionsArea follow={follow} setFollow={setFollow} />
        </div>
      )
  );
}
export default function RoleAdminRoute() {
    const { user } = useAppContext();
    return (
        user?.role === Role.ADMIN || user?.role === Role.ROOT
            ? <Outlet />
            : <NotFoundPage />
    );
}
export default function RoleRootRoute() {
    const { user } = useAppContext();
    return (
        user?.role === Role.ROOT
            ? <Outlet />
            : <NotFoundPage />
    );
}
export default function RoleUserRoute() {
    const { user } = useAppContext();
    return (
            user?.role === Role.USER || user?.role === Role.ADMIN || user?.role ===
Role.ROOT
                ? <Outlet />
                : <NotFoundPage />
    );
}
export default function UserRoute() {
  const params = useParams();
  const [userExists, setUserExists] = useState(false);
  const [isLoading, setLoading] = useState(true);
  const [userProfile, setUserProfile] = useState({
    username: "",
    registrationDate: "",
    uniqueName: "",
    firstname: "",
    lastname: "",
    birthday: "",
    role: "",
    profileImage: null
  });
```

```
  useEffect(() => {
    if (!params.uniqueName) {
      setUserExists(true);
    }
    async function checkUserExistence() {
      let profileDataResponse = null;
      let profileImageResponse = null;
      if (params.uniqueName) {
        profileDataResponse = await getUserByUniqueNameAndRole(params.uniqueName,
Role.USER);
      }
      if (profileDataResponse !== null && profileDataResponse?.data?.uniqueName ===
params.uniqueName) {
        setUserExists(true);
        const { data } = profileDataResponse;
      profileImageResponse = await getProfileImage(data.uniqueName);
        let imageToSet = null;
        if (profileImageResponse?.data
          && profileImageResponse?.data?.type.startsWith("image/")
          && profileImageResponse?.data?.size !== 0) {
          let imageBytes = profileImageResponse.data;
          const profileImageMetadataResponse = await
getProfileImageMetadata(data.uniqueName);
          if (profileImageMetadataResponse?.data) {
            const imageMetadata = profileImageMetadataResponse.data;
            const imageName = imageMetadata.name;
            const imageType = imageMetadata.type;
            const imageUsername = imageMetadata.username;
            const imageBlob = new Blob([imageBytes], { type: imageType });
            imageToSet = Object.assign(
              imageBlob,
              {
                preview: URL.createObjectURL(imageBlob),
                name: imageName,
                username: imageUsername
              }
            );
          }
        }
    setUserProfile({
          username: data.username,
          registrationDate: data.registrationDate,
          uniqueName: data.uniqueName,
          firstname: data.firstname,
          lastname: data.lastname,
          birthday: data.birthday,
          role: data.role,
          profileImage: imageToSet
        })
      }
      setLoading(false);
    }
    checkUserExistence();
  }, [params]);
  return (
    isLoading
      ? <LoadingPage />
      : userExists
        ? (
```

```
            <UserContext.Provider value={{ userProfile, setUserProfile }}>
              <Outlet />
            </UserContext.Provider>
          )
          : <NotFoundPage />
    );
}
export const AuthContext = createContext();
export const useAuthContext = () => useContext(AuthContext);
export default function AuthenticationBasedRoute() {
    const { user, setUser, setInformMessage } = useAppContext();
    const navigate = useNavigate();
    const location = useLocation();
    let socket = new SockJS(API_WEB_SOCKET_URL);
    let stompClient = Stomp.over(socket);
    const [isLoading, setLoading] = useState(true);
    const [userChats, setUserChats] = useState([]);
    const [isInitialMount, setIsInitialMount] = useState(true);
    useEffect(() => {
        async function checkIsUserAuthenticated() {
            let isAuthenticated = await isUserAuthenticated();
            if (!isAuthenticated) {
navigate(HOME_ROUTE);
                setUser({
                    username: "",
                    uniqueName: "",
                    authenticated: false,
                    role: Role.VISITOR
                });
            } else if (isAuthenticated && user?.authenticated === true &&
user?.username) {
                let userAccountResponse = await
getUserAccountByUserUsername(user?.username);
                let userAccountData = userAccountResponse?.data;
                if (!userAccountData) {
                    let userAccountState = userAccountData?.state;
                    if (userAccountState !== UserAccountState.ACTIVATED) {
                        navigate(HOME_ROUTE);
                        setUser({
                            username: "",
                            uniqueName: "",
                            authenticated: false,
                            role: Role.VISITOR
                        });
                        localStorage.clear();
                    }
                }
            }
        }
        checkIsUserAuthenticated();
    }, [location.pathname]);
    useEffect(() => {
        exchangePublicEncryptionKeys(user.username);
        if (!stompClient.connected && user.authenticated && user.role !== Role.ADMIN
&& user.role !== Role.ROOT) {
            let accessToken = localStorage.getItem("access-token");
            stompClient.connect({ Authorization: `Bearer ${accessToken}` },
onWebSocketConnected, onWebSocketConnectionError);
        }
```

```
            setLoading(false);
            return () => {
                if (stompClient.connected) {
                    stompClient.disconnect(() => {
                        console.log("Disconnected from websockets")
                    });
                }
            }
        }, [user]);
        const onWebSocketConnected = async () => {
            let accessToken = localStorage.getItem("access-token");
            stompClient.subscribe(API_WEB_SOCKET_MESSAGING_TOPIC_URL + "/" + user.username
    + "/notifications", onUserNotificationReceived, { Authorization: `Bearer
    ${accessToken}` });
            let currentUserChatsResponse = await getCurrentUserChats(3);
            let currentUserChats = currentUserChatsResponse?.data;
            if (currentUserChats) {
                for (let i = 0; i < currentUserChats.length; i++) {
                    let currentUserChat = currentUserChats[i];
                    if (currentUserChat) {
                        let chatId = currentUserChat?.id;
                        if (chatId) {
                            stompClient.subscribe(API_WEB_SOCKET_MESSAGING_TOPIC_URL +
    "/chats/" + chatId + "/messages", onChatMessageReceived, { Authorization: `Bearer
    ${accessToken}` });
                        }
                    }
                }
                setUserChats(currentUserChats);
            }
        };
        const onWebSocketConnectionError = () => {
            setInformMessage("Sorry! Error occured while conecting! Please sign in
    again!");
            navigate(HOME_ROUTE);
            setUser({
                username: "",
                uniqueName: "",
                authenticated: false,
                role: Role.VISITOR
            });
        };
        const onChatMessageReceived = async (payload) => {
            let payloadBody = payload?.body;
            if (payloadBody) {
                let message = JSON.parse(payloadBody);
                let senderUsername = message?.sender?.username;
                    let messageType = message?.type;
                if (senderUsername && messageType && senderUsername === user?.username &&
    messageType === MessageType.CHAT_MEMBER_LEFT_CHAT) {
                    return;
                }
                let currentUserChatsResponse = await getCurrentUserChats(3);
                let currentUserChats = currentUserChatsResponse?.data;
                if (currentUserChats) {
                    setUserChats(currentUserChats);
                }
            }
        };
```

```
    return (
        isLoading
            ? <LoadingPage />
            : user.authenticated
                ? (
                    <AuthContext.Provider value={{ stompClient, userChats,
setUserChats, onChatMessageReceived }}>
                        <Outlet />
                    </AuthContext.Provider>
                )
                : <NotLoggedInPage />
    )
}
export default function ChatRoute() {
    const location = useLocation();
    return (
            location.pathname !== CHATS_ROUTE
                ? <Outlet />
                : <NotFoundPage />
    );
}
export default function UserRoute() {
  const params = useParams();
  const [userExists, setUserExists] = useState(false);
  const [isLoading, setLoading] = useState(true);
  const [userProfile, setUserProfile] = useState({
username: "",
    registrationDate: "",
    uniqueName: "",
    firstname: "",
    lastname: "",
    birthday: "",
    role: "",
    profileImage: null
  });

  return (
    isLoading ? <LoadingPage />
      : userExists
        ? (
          <UserContext.Provider value={{ userProfile, setUserProfile }}>
            <Outlet />
          </UserContext.Provider>
        )
        : <NotFoundPage />
  );
}
export default function ViewUsersRoute() {
    const [isLoading, setLoading] = useState(true);
    const [isInitialMount, setIsInitialMount] = useState(true);
    const {
        fetchNextPage,
        isFetchingNextPage,
        data,
        status
    } = useInfiniteQuery({
        queryKey: ["users"],
        queryFn: ({ pageParam = 0 }) => getUsers(pageParam, 3, "ASC"),
        getNextPageParam: (lastPage, allPages) => {
            return lastPage.data.length ? allPages.length : undefined;}});
```

```
    let users = data?.pages?.reduce((prevPage, currentPage) =>
[...prevPage, ...currentPage.data], []);
const observer = useRef(null);
    const lastUserRef = useCallback(user => {
        if (!user) return;
        if (isFetchingNextPage) return;

        if (observer.current) {
            observer.current.disconnect();
        }
        observer.current = new IntersectionObserver((users) => {
            if (users[0].isIntersecting) {
                fetchNextPage();
            }
        })
        if (user) {
            observer.current.observe(user);
        }
    }, [isFetchingNextPage, fetchNextPage])
      useEffect(() => {
        if (isInitialMount) {
            setIsInitialMount(false);
            return;
        }
        setLoading(false);
    }, [users]);
    if (status === 'error') return <NotFoundPage />
    return (
        isLoading
            ? <LoadingPage />
            : (
                <>
                    <ViewUsersActionsArea />
                    {
                        users && users.length > 0
                            ? (
                                <div className="view-users-page">
                                    {users.map((user) => {
                                        return <UsersListItem key={user.username}
ref={lastUserRef} user={user} role={Role.USER} />
                                    })}
                                </div>
                            )
                            : <UsersNotFoundPage />
                    }
                </>
            )
    );}
export const generateKeyPair = () => {
    let keyPair = forge.pki.rsa.generateKeyPair({ bits: 2048});
    let privateKey = forge.pki.privateKeyToPem(keyPair.privateKey);
    let publicKey = forge.pki.publicKeyToPem(keyPair.publicKey);
    localStorage.setItem("user-private-key", extractRawKeyValue(privateKey));
    localStorage.setItem("user-public-key", extractRawKeyValue(publicKey));
}
function App() {
  const [user, setUser] = useState({
    username: "",
    uniqueName: "",
```

```
    authenticated: false,
    role: Role.VISITOR
  });
  const [isInformMessageShow, setInformMessageShow] = useState(false);
  const [informMessage, setInformMessage] = useState("");
  useEffect(() => {
    async function checkIsUserAuthenticated() {
      let isAuthenticated = await isUserAuthenticated();
      if (isAuthenticated) {
        let accessToken = localStorage.getItem("access-token");
        if (accessToken) {
          const username = jwtDecode(accessToken).sub;
          const role = jwtDecode(accessToken).role;
          if (username && role) {
            const response = await getUserByUsernameAndRole(username, role);
            const data = response?.data;
            if (data && data?.username && data?.username === username) {
              const uniqueName = data.uniqueName;
              setUser({
                username: username,
                uniqueName: uniqueName,
                authenticated: isAuthenticated,
                role: role
              });
            }
          }
        }
      }
    }
    checkIsUserAuthenticated();
  }, []);
  useEffect(() => {
    if (informMessage) {
      setInformMessageShow(true);

      setTimeout(() => {
        setInformMessageShow(false);
        setInformMessage("");
      }, 5000);
    }
  }, [informMessage]);
  return (
    <QueryClientProvider client={queryClient}>
      <AppContext.Provider value={{ user, setUser, setInformMessage }}>
        <BrowserRouter>
          <div className="App">
            <NavBar />
            <RoutesProcessor />
            <div className={isInformMessageShow ? "inform-message-container inform-
message-container-show" : "inform-message-container"}>
              <div className="inform-message">{informMessage}</div>
            </div>
          </div>
        </BrowserRouter>
      </AppContext.Provider>
    </QueryClientProvider>
  );
}
```

```
export default App;
export const BASE_URL = "http://localhost:8080";
export const API_WEB_SOCKET_URL = "http://localhost:8080/api/ws";
export const API_WEB_SOCKET_MESSAGING_URL = "/api/messaging";
export const API_WEB_SOCKET_MESSAGING_TOPIC_URL = "/api/messaging/topic";
export const API_REGISTRATION_ROUTE = "/api/registration";
export const API_TOKEN_VALIDATION_ROUTE = "/api/jwt/validation";
export const API_ACCESS_TOKEN_ROUTE = "/api/jwt/access";
export const API_LOGOUT_ROUTE = "/api/logout";
export const API_ROOTS_ROUTE = "/api/roots";
export const API_USER_ROUTE = "/api/user";
export const API_ADMIN_ROUTE = "/api/admin";
export const API_ADMINS_ROUTE = "/api/admins";
export const API_ROOT_ROUTE = "/api/root";
export const API_CHATS_ROUTE = "/api/chats";
export const API_USERS_ROUTE = "/api/users";
export const API_ENCRYPTION_KEYS_ROUTE = "/api/encryptionKeys";
export const API_AUTHENTICATION_ROUTE = "/api/authentication";
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <App />
);
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <meta name="theme-color" content="#000000" />
    <meta name="description" content="Messenger" />
    <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
    <title>WebTalk</title>
  </head>
  <body>
    <noscript>You need to enable JavaScript to run this app.</noscript>
    <div id="root"></div>
  </body>
</html>
export const axiosClient = axios.create({
    baseURL: BASE_URL
});
export const authAxiosClient = axios.create({
    baseURL: BASE_URL
});
export const refreshAuthAxiosClient = axios.create({
    baseURL: BASE_URL
});
authAxiosClient.interceptors.request.use(async (request) => {
    let isAuthenticated = await isUserAuthenticated();
    if (isAuthenticated) {
        const accessToken = localStorage.getItem("access-token");
        request.headers.Authorization = `Bearer ${accessToken}`;
    } return request;
});
refreshAuthAxiosClient.interceptors.request.use((request) => {
    const refreshToken = localStorage.getItem("refresh-token");
    if (refreshToken) {request.headers.Authorization = `Bearer ${refreshToken}`;}
    return request;
});
```