Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(інститут)

Факультет інформаційних технологій
(факультет)

Кафедра Програмного забезпечення комп'ютерних систем
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *Вознюка Костянтина Юрійовича*
(ПІБ)

академічної групи *122-18-2*
(шифр)

спеціальності *122 Комп'ютерні науки*
(код і назва спеціальності)

освітньої програми *Комп'ютерні науки*
(назва освітньої програми)

на тему: *Розробка серверної частини соціального месенджера на основі Java/Spring*

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікаційної роботи	*доц. Реута О.В.*			
розділів:				
спеціальний	*доц. Реута О.В.*			
економічний	*доц. Касьяненко Л.В.*			
Рецензент				
Нормоконтролер	*доц. Гуліна І.Г.*			

Дніпро
2022

Міністерство освіти і науки України
НТУ «Дніпровська політехніка»

ЗАТВЕРДЖЕНО:
завідувач кафедри
програмного забезпечення комп'ютерних систем
(повна назва)

_____ І.М. Удовик
(підпис) (прізвище, ініціали)

« » _____ 2022 року

ЗАВДАННЯ
на кваліфікаційну роботу

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *122-18-2* *Вознюка Костянтина Юрійовича*
(група) (прізвище та ініціали)

тема кваліфікаційної роботи *Розробка серверної частини соціального меседжера на основі Java/Spring*

затверджена наказом ректора НТУ «ДП» від «18» травня 2022р. № 268-с

Розділ	Зміст виконання	Термін виконання
Спеціальний	***На основі матеріалів проєктно-технологічної практики та інших науково-технічних джерел провести аналіз стану рішення проблеми та постановку задачі. Обґрунтувати вибір та здійснити реалізацію методів вирішення проблеми***	***13.05.2022 р.***
Економічний	***Провести розрахунок трудомісткості розробки програмного забезпечення, витрат на створення ПЗ й тривалості його розробки***	***27.05.2022 р.***

Завдання видав _____ *доц. Реута О.В.*
(підпис) (посада, прізвище, ініціали)

Завдання прийняв до виконання _____ *Вознюк К.Ю.*
(підпис) (прізвище, ініціали)

Дата видачі завдання: *14.01.2022 р.*

Термін подання кваліфікаційної роботи до ЕК: *13.06.2022*

РЕФЕРАТ

Пояснювальна записка: 85 с., 22 рис., 3 дод., 23 джерел.

Об’єкт розробки: система функціонування месенджера.

Мета кваліфікаційної роботи: розробка серверної частини соціального месенджера на основі Java/Spring.

У вступі розглядається аналіз та сучасний стан проблеми, конкретизується мета кваліфікаційної роботи та галузь її застосування, наведено обґрунтування актуальності теми та уточнюється постановка завдання.

У першому розділі проведено аналіз предметної галузі, визначено актуальність завдання та призначення розробки, розроблена постановка завдання, задані вимоги до програмної реалізації, технологій та програмних засобів.

У другому розділі виконано аналіз існуючих рішень, обрано вибір платформи для розробки, виконано проектування і розробка програми, наведено опис алгоритму і структури функціонування системи, визначені вхідні і вихідні дані, наведені характеристики складу параметрів технічних засобів, описаний виклик та завантаження застосунку, описана робота програми.

В економічному розділі визначено трудомісткість розробленої інформаційної підсистеми, проведений підрахунок вартості роботи по створенню застосунку та розраховано час на його створення.

Практичне значення полягає у створенні системи, що надає можливість обміну повідомленнями, перегляди та оновленню особистої інформації, пошуку інших користувачів.

Актуальність системи обміну повідомленнями обумовлена необхідністю швидкої та ефективної комунікації.

Список ключових слів: БАЗА ДАНИХ, ПОВІДОМЛЕННЯ, СЕРВЕР, КЛІЄНТ, ПРОГРАМНИЙ ІНТЕРФЕЙС, ЗАСТОСУНОК, ШИФРУВАННЯ.

ABSTRACT

Explanatory note: 85 p., 26 figs, 3 apps, 23 sources.

Object of development: the system of functioning messenger.

The purpose of the diploma project: development of the server part of the social messenger based on Java / Spring.

The introduction considers the analysis and the current state of the problem, specifies the purpose of the qualification work and the scope of its application, provides a justification for the relevance of the topic and clarifies the problem.

In the first section the analysis of the subject area is carried out, the urgency of the task and purpose of development is defined, the statement of the task is developed, requirements to software realization, technologies and software are set.

The second section analyzes the existing solutions, selects the platform for development, performs design and development of the program, describes the algorithm and structure of the system, determines the input and output data, provides characteristics of the parameters of technical means, describes the call and application load, describes the program .

The economic section determines the complexity of the developed information subsystem, calculates the cost of work to create an application and calculates the time for its creation.

The practical significance is creation of a system that provides the ability to exchange messages, view and update personal information, search for other users.

Keywords: DATABASE, MESSAGE, SERVER, CLIENT, SOFTWARE INTERFACE, APPLICATION, ENCRYPTION

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

БД - база даних

API - Application Programming Interface

REST - Representational State Transfer

JSON - JavaScript Object Notation

HTML - HyperText Markup Language

SQL - Structured Query Language

IT - Інформаційні технології

HTTP - Hypertext Transfer Protocol

ВСТУП

Спілкування в месенджерах стало звичайним ділом для нас, кожного дня ми обмінюємось повідомленнями з нашими близькими або друзями. Із розвитком та поширенням Інтернету виникла ідея створення нового способу спілкування між людьми – за допомогою соціальних мереж. Можливість швидко та легко надіслати повідомлення іншій людині змінила привичний спосіб спілкування. З розвитком технологій соціальні мережі стали більш орієнтованими під мобільні пристрої, цей чинник, в свою чергу, сприяв тому, що люди почали приділяти соціальним мережам більше часу. На мою думку, соціальні мережі не несуть у собі небезпеки залежності при раціональному використанні.

Популярність месенджерів стрімко зросла за останні роки, люди все більше відходять від класичних соціальних мереж таких як Facebook (Meta) та переходять до спілкування у месенджерах. Цьому є декілька пояснень: відсутність у багатьох популярних месенджерах реклами, швидкодія та надійність, мінімалістичний та зручний інтерфейс, наявність лише потрібного функціоналу. Також слід зазначити, що поява корпоративних месенджерів дозволила відійти від класичного способу комунікації працівників всередині компанії – обміну повідомленнями на пошту. Це дозволило налагодити більш швидку та ефективну комунікацію.

Зараз на ринку існують декілька популярних додатків, які виконують роль месенджерів. Це такі додатки як: Telegram, Slack, Viber, WhatsApp тощо. В кожного додатку є свої плюси та мінуси, наявність вбудованої реклами, рівень захисту даних користувачів, зручність інтерфейсу.

Основною метою даної роботи є створення зручного програмного інтерфейсу з відкритим програмним кодом, який може бути використаний мобільним клієнтом або веб-браузером.

Завдання даної кваліфікаційної роботи покриває навички та компетенції, якими повинні володіти бакалаври напряму 122 «Комп’ютерні науки» галузі знань 12 «Інформаційні технології».

РОЗДІЛ 1

АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1. Загальні відомості з предметної галузі

Розвиток та поширення швидкісного Інтернету повністю змінило світ. Суспільство отримало потужний інструмент комунікації та отримання інформації: «Інтернет доступний будь-кому і представляє собою безкінечне інформаційне поле. В ньому представлена сама різноманітна інформація – легальна і нелегальна, моральна та аморальна, соціальна, політична, правова, медична, економічна, релігійна, комп'ютерна та будь-яка інша, яка завантажена людиною в це поле за допомогою комп'ютера. Ще 50 років тому Інтернет сміливо міг би стати предметом (темою) фантастичних романів, але зараз – це реальний, окремий, створений людиною світ, в якому можна опинитися завдяки відповідній комп'ютерній техніці [1]».

Із розвитком Інтернету почали з'являтися різні соціальні мережі, які дозволяли користувачам з різних куточків світу обмінюватись повідомленнями, ділитися новинами, поширювати особисті фотографії.

Зараз модель соціальних мереж початку 00-х, а саме перевантажені функціоналом, який потрібний далеко не всім користувачам, системи у минулому. З'являються додатки, які орієнтовані на певну дію, наприклад, поширення особистих фотографій або обмін миттєвими повідомленнями.

Миттєві повідомлення або система обміну миттєвими повідомленням – це телекомунікаційна служба для обміну текстовими повідомленнями між комп'ютерами або іншими пристроями через Інтернет [2].

Клієнтську програму для системи обміну повідомленнями часто називають месенджером.

На відміну від електронної пошти, система обміну повідомленнями надає можливість спілкуватися в режимі реального часу, аби отримати повідомлення

користувачу непотрібно йти у поштову скриньку та перевіряти чи прийшли йому нові повідомлення. Це є можливим через те, що система використовує у своїй роботі спеціальні протоколи, наприклад, вебсокет – протокол, що забезпечує двонаправлений повнодуплексний канал зв'язку через один TCP-сокет.

Найпопулярнішими месенджерами в Україні станом на сьогодні є Viber, яким користувалися хоча б раз 99% користувачів смартфонів, Facebook Messenger та Telegram, кількість користувачів останнього зараз стрімко зростає. Популярний на заході WhatsApp не володіє широкою аудиторією в нашій країні, ним користується менше половини користувачів.

Основні компоненти системи:

- система ідентифікації (адресації) клієнтів – унікальні ідентифікатори користувачів, що дозволяють відрізняти їх один від одного;
- система обліку стану клієнтів – спеціальні статуси, що дозволяють повідомити іншим користувачам про статус користувача;
- система доставки повідомлень – передача повідомлення від одного користувача до іншого;
- список контактів – користувач має змогу створювати власні списки контактів;
- система зберігання особистої інформації – користувач може задавати особисту інформацію, редагувати або видаляти її.

Розроблюваний додаток при створені орієнтувався на існуючі аналоги такі як Viber та Telegram, використовуючи їх переваги та уникаючи недоліків.

Viber

Viber запустили для широкої аудиторії у грудні 2010 року, у 2018 компанія звітувала про понад 1 мільярд користувачів у 190 країнах світу. Зараз права на Viber належать японській компанії Rakuten.

Автором ідеї є американсько-ізраїльский розробник Тальмон Марко. Цікавим фактом є те, що через «ізраїльське коріння» додаток є забороненим

урядами Єгипту та Лівану на території їхніх країн, це пояснюється зв’язками з «сіоністськими шпигунами». Проте, численні арабські сайти пропонують користувачам поради, як обійти ці блокування [3].

Рис. 1.1. Логотип додатку Viber

Додаток Viber є безкоштовним для завантаження, проте має платні функції серед яких: відключення реклами, деякі набори стікерів, якими користувач може обмінюватись у повідомленнях, створення комерційних ботів.

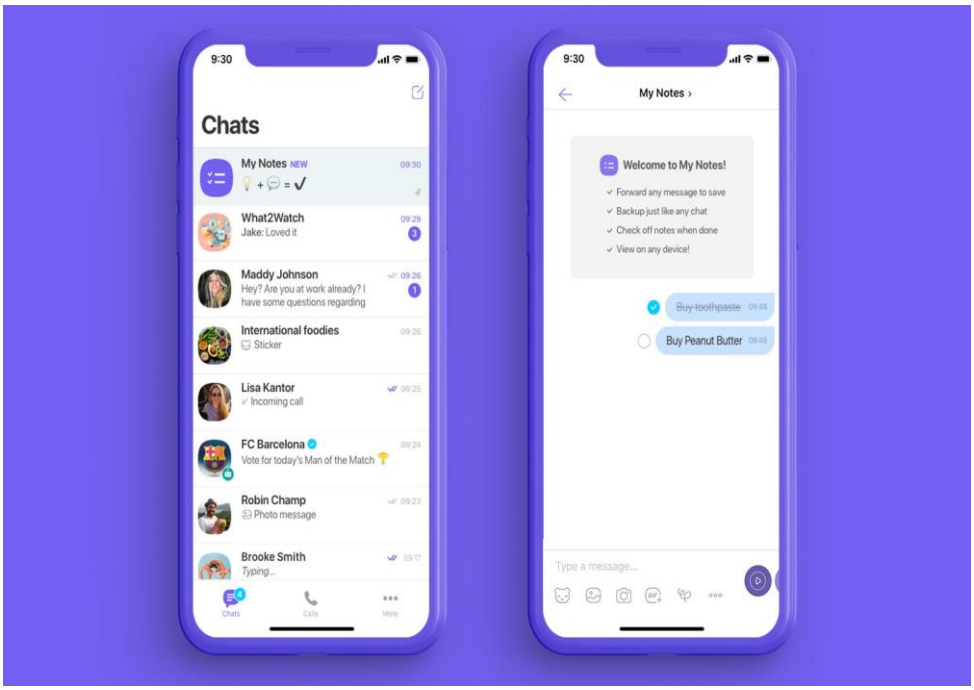

Рис. 1.2. Меню та чат Viber

Переваги:

- безкоштовний доступ до основного функціоналу;
- доволі приємний дизайн;
- широке використовування різними компаніями для ефективного спілкування з клієнтами.

Недоліки:

- відсутність веб-браузер клієнту;
- наявність реклами;
- платні функції.

На відміну від Viber розроблюваний додаток може бути використаний як веб-браузером так і мобільним клієнтом. У розроблюваному додатку повністю відсутня будь-яка реклама та платні функції.

Telegram

Telegram було засновано у 2013 році братами Павлом та Миколою Дуровими. Кількість активних користувачів щомісяця становить близько 500 мільйонів осіб. Проект практично повністю фінансує Павло у обсязі понад 13 мільйонів доларів щорічно [4].

Рис. 1.3. Логотип додатку Telegram

За останні роки популярність Telegram зростає, особливо серед молоді. Telegram полюбляють за швидкість його роботи та зручний інтерфейс.

Рис. 1.4. Чат в Telegram

Переваги:

- відсутність реклами;
- швидкодія;
- зручний інтерфейс.

Недоліки: З останніми оновлюваннями в Telegram додається багато функцій, які не потрібні більшій кількості користувачів.

Розроблювана система на відміну від Telegram не є перевантаженою.

1.2. Призначення розробки та галузь її застосування

Даний додаток може бути використаний користувачами для спілкування через мережу Інтернет.

Необхідність створення даного програмного продукту обумовлена бажанням надати зручний сервіс для спілкування без будь-якої комерційної складової та який може бути використаний як мобільним клієнтом так і браузерним.

Призначення розробки – створення зручного API, який надає можливість користувачу обмінюватись повідомленнями з іншими користувачами, шукати інших користувачів. додавати їх до списку контактів, редагувати власний обліковий запис.

1.3. Підстави для розробки

Згідно навчального плану та освітньої програми, у кінці навчання студент має виконати кваліфікаційну роботу.

Тема роботи має бути узгоджена із керівником проекту, випусковою кафедрою.

Отже, підставами для виконання кваліфікаційної роботи є:

- навчальний план та графік навчального процесу;
- освітня програма напряму 122 «Комп’ютерні науки»;
- наказ ректора Національного технічного університету «Дніпровська політехніка» № 268-с від 18 травня 2022 р.

1.4. Постановка завдання

Метою кваліфікаційної роботи є розробка серверної частини соціального месенджера та надання зручного API, який задовольняє вимоги клієнта.

Реалізація поставленої мети передбачає виконання наступних завдань:

- проектування моделі бази даних;
- обрання доцільної архітектури продукту;
- розробка програмного коду продукту;

– розгортання програмного продукту на сервері.

Кінцевий продукт являє собою універсальний API виконаний за стандартами REST.

1.5. Вимоги до програми або програмного виробу

1.5.1. Вимоги до функціональних характеристик

Розроблюваний програмний продукт повинен мати наступні функції:

– користувач має можливість зареєструватися у додатку;
– користувач не повинен мати можливість зареєструватися за існуючим логіном, номером телефону або електронною поштою;
– користувач має можливість авторизуватися у додаток за логіном, який було обрано при реєстрації;
– користувач може редагувати власні дані, які були указані при реєстрації;
– користувач має можливість змінити власний пароль;
– користувач може шукати інших користувачів за логіном;
– користувач має список контактів та може видаляти та додавати інших користувачів;
– користувач має можливість надсилати повідомлення іншим користувачам;
– користувач має можливість видаляти та редагувати власні повідомлення;
– користувач має можливість видалити чат з іншим користувачем із втратою повідомлень для обох сторін;
– при отримані повідомлення від іншого користувача, якщо користувач знаходиться у мережі він має бути проінформований про нове повідомлення, одразу як воно було відправлено;

- дані, які надходять від серверу до клієнту повинні бути представлені у JSON форматі;
- API додатку повинен бути задокументований та описаний на окремій HTML сторінці.

1.5.2. Вимоги до інформаційної безпеки

Розроблюваний продукт має бути безпечним для особистих даних користувачів та дотримуватись наступних вимог:

- сесія користувача не повинна бути «нескінченною» , користувача потрібно повторно авторизувати через деякий проміжок часу. Цей функціонал має бути реалізований шляхом JWT-токенів;
- запити повинні бути захищенні за допомогою CORS-headers;
- пароль користувача повинен бути надійно зашифрований;
- повідомлення користувачів повинні бути зашифровані;
- серверна частина повинна бути захищена від SQL ін’єкцій;
- протокол передачі даних HTTPS.

1.5.3. Вимоги до складу та параметрів технічних засобів

Для коректної та швидкої роботи розроблюваної програми необхідні наступні умови:

- операційна система Linux або Windows 10;
- оперативна пам’ять [RAM] не менше 2 ГБ;
- відеопам’ять [VRAM]: 64МБ;
- швидкісний Інтернет;
- HDD або SSD об’ємом від 8 ГБ;
- процесор Intel Core i3 або AMD Ryzen 3.

1.5.4. Вимоги до інформаційної та програмної сумісності

Розроблене програмне забезпечення повинно бути:

- сумісним с операційними системами Linux та Windows 10;
- сумісним з будь-яким постачальником послуг розгортання;
- вихідні коди програми повинні бути розміщені на одному з сервісів, що надає послуги контролю версій розроблюваних систем;
- для роботи та розробки необхідно використовувати IntellijIDEA або Eclipse.

РОЗДІЛ 2

ПРОЕКТУВАННЯ ТА РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1. Функціональне призначення системи

Функціональним призначенням системи є надання альтернативної мережі для спілкування користувачів.

Основні функції:

- реєстрація та авторизація користувачів додатку;
- обмін повідомлення у режимі реального часу;
- пошук співрозмовників;
- редагування власного облікового запису.

2.2. Опис застосованих математичних методів

У розроблюваній інформаційній системі математичні методи не використовувались.

2.3. Опис використаних технологій та мов програмування

Розроблюваний програмний продукт є серверним додатком, тому при розробці були використані технології націлені на створення саме таких додатків. При розробці додатку були використані різноманітні REST принципи, які використовуються при сучасній розробці в IT індустрії.

REST — це архітектурний стиль, який просуває певні стандарти між комп’ютерними системами, значно спрощуючи їх комунікацію. Системи, які використовують REST принципи часто називають RESTful. Ці системи

характерні тим, що у них відсутній стан та розділяють задачі клієнту та серверу[5]. Отже, REST архітектура базується на таких принципах:

1. відсутність станів – це означає, що серверу непотрібно знати будь-що про те в якому стані перебуває клієнт та навпаки. Це дозволяє набути системі гнучкості та надійності;
2. розділення клієнту та серверу – згідно цього принципу реалізація клієнта та сервера має бути виконана незалежно від один одного. Це дозволяє досягти змін в кодовій базі клієнту або серверу, які не впливають на іншу сторону, бо клієнт не знає нічого про сервер, а сервер про клієнт;
3. комунікація між сервером та клієнтом будується на таких стандартах:
    3.1. існує 4 базових HTTP дієслова, які застосовують в REST системі – GET для отримання ресурсів, POST для створення ресурсів, PUT для оновлення ресурсу, DELETE для видалення ресурсу;
    3.2. шляхи на сервер мають інформативно вказувати на ресурс та бути лаконічними, наприклад, users/5/chat/1;
    3.3. в заголовках запитів клієнт надсилає тип контенту, який він може обробити;
    3.4. відповіді від сервера містять статус коди, що повідомляють клієнту про успішність операції. На кожне HTTP дієслово є певний успішний статус код.
4. для забезпечення швидкодії додатку деякі запити можуть бути закешовані.

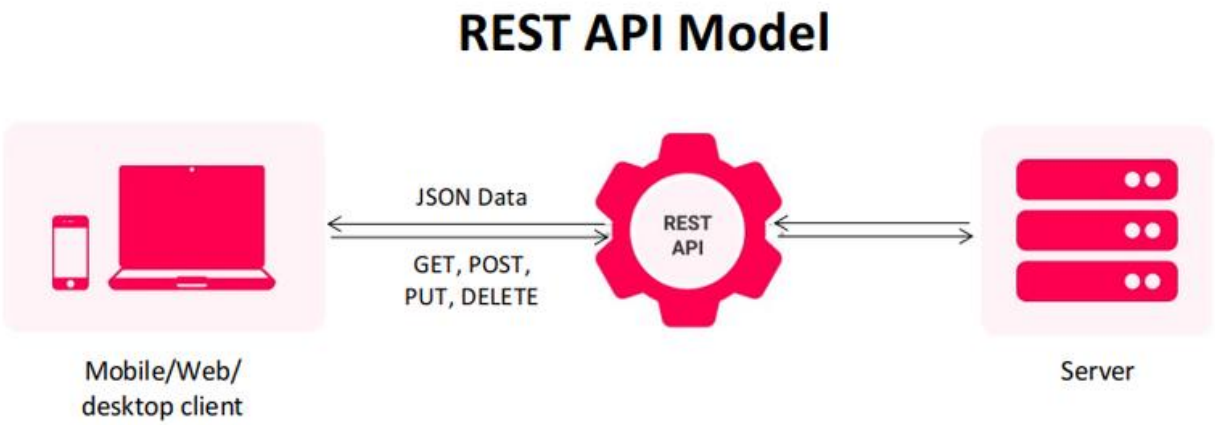

Рис. 2.1. Взаємодія компонентів REST архітектури

В якості мови програмування серверної частини додатку була обрана Java та спеціальний фреймворк для розробки веб-застосунків Spring й споріднені з ним фреймворк такі як Spring Boot, Spring Security, Spring Data. В якості бази даних була обрана NoSQL база даних MongoDB. Також при розробці використовувався Docker.

Java

Java — це об’єктно-орієнтована мова програмування із строгою типізацію. Синтаксис Java значно запозиченний з таких мов як C та C++, зокрема з C++ було взято за основу об’єктну модель, яка була дещо модифікована. Java є однією з найпопулярніших мов програмування для розробки серверної частини WEB-додатків.

Рис. 2.2. Логотип Java

Spring

Spring — це фреймворк з відкритим програмним кодом, розроблений компанією Pivotal. Spring надає контейнер, який ще називають контекстом програми, що створює та керує програмними компонентами. Ці компоненти пов'язані один з одним всередині контейнеру і разом створюють цілісний додаток [6]. Розробка із використанням Spring значно заощаджує час та полегшує сам процес порівняно із класичною веб-розробкою із використанням Java EE.

Рис. 2.3. Логотип фреймворку

Spring Boot

Spring Boot спрощує створення автономних додатків на основі Spring, надаючи ще більше комфорту та гнучкості при розробці. Ось деякі особливості, які надає Spring Boot:

- вбудований Tomcat, Jetty або Undertow сервери;
- надання необхідних програмних бібліотек таким чином, що розробник може не піклуватися про несумісність версій;
- автоматична конфігурація проекту – створення необхідних компонентів для функціонування додатку без участі розробника;
- надання метрик, що відображають стан серверу.

Spring Security

Spring Security — це фреймворк, орієнтований забезпечення як автентифікації, і авторизації додатків Java. Як і у всіх проектах Spring, реальна сила Spring Security полягає в тому, наскільки легко його можна розширити для

задоволення потреб користувача. Це де-факто стандарт для захисту програм на основі Spring [7].

Spring Data

Spring Data — це фреймворк, основна ціль якого зробити роботу з базою данних легшою для розробника. Фреймворк дозволяє розробнику абстрагуватись від написання запитів до бази даних, використовуючи спеціальні програмні методи. Spring Data працює як з реляційними базами даних так із нереляційними. Головною особливістю фреймворку є побудова запитів за допомогою словосполучень, наприклад findUserById або updateUserWhereIdIs.

NoSQL

Бази даних NoSQL були створені для певних моделей даних, вони мають гнучкі схемами, що дозволяє розробляти сучасні програми. NoSQL набуло широкого поширення через простоту розробки, високу продуктивність за великих масштабів даних та функціональність.

Навідміну від реляйційних БД, в NoSQL немає таблиць, ці бази зберігають дані по іншому. NoSQL БД бувають таких типів:

- графові;
- документні;
- ключ-значення;
- широка колонка.

У таблиці 2.1. наведено порівняння обох типів баз данних, їх ключові відмінності

Таблиця 2.1.

Відміності реляційних та нереляційних БД

	Реляційні БД	NoSQL БД
Робочі нагрузки	Реляційні БД призначені для транзакційних додатків, де обробки транзакцій йдуть у режимі реального часу та добре підходять для аналітичної обробки у режимі реального часу.	Бази даних NoSQL призначені для роботи з додатками в яких необхідна низька затримка. Пошукові бази даних NoSQL призначені для аналітики частково структурованих даних.
Модель даних	Реляційна модель нормалізує дані та зберігає їх у вигляді таблиць, що складаються з рядків та стовпців. Схема чітко визначає таблиці, рядки, стовпці, індекси, відносини між таблицями та інші елементи БД. Така БД забезпечує цілісність посилань у відносинах між таблицями.	Бази даних NoSQL зберігають дані по іншому, існують такі моделі як пари «ключ-значення», документи та графи, оптимізовані для високої продуктивності та масштабованості.

Продовження таблиці 2.1.

	Реляційні БД	NoSQL БД
Продуктивність	Продуктивність залежить від дискової підсистеми. Для максимальної продуктивності необхідна оптимізація запитів, індексів та структури таблиці.	Продуктивність зазвичай залежить від розміру кластера базового апаратного забезпечення, затримки мережі та додатка.
Масштабування	Реляційні бази даних зазвичай масштабуються шляхом збільшення обчислювальних можливостей апаратного забезпечення або додавання окремих копій робочих навантажень читання.	Бази даних NoSQL зазвичай підтримують високу роздільність завдяки шаблонам доступу з можливістю масштабування на основі розподіленої архітектури. Це підвищує пропускну здатність та забезпечує стійку продуктивність майже в необмежених масштабах.
Властивості ACID	Реляційні бази даних забезпечують набір властивостей ACID: атомарність, несуперечність, ізольованість, надійність.	Бази даних NoSQL часто пропонують компроміс, пом'якшуючи жорсткі вимоги властивостей ACID для більш гнучкої моделі даних, яка припускає горизонтальне масштабування.

Закінчення таблиці 2.1.

	Реляційні БД	NoSQL БД
API	Запроси до БД виконуються на мові SQL, ці запити аналізує та виконує реляційна БД	Об’єктно-орієнтовані API дозволяє розробникам легко зчитувати та записувати дані. Завдяки використанню ключів секцій програми можуть вести пошук по парам «ключ-значення», наборам стовпців або частково структурованим документам, що містять серійні об’єкти та атрибути програм.

MongoDB

MongoDB — документно-орієнтована нереляційна база даних з відкритим вихідним кодом. MongoDB зберігає документи в JSON-подібному форматі, має гнучку мову для формування запитів, може створювати індекси для різних збережених атрибутів, ефективно забезпечує зберігання великих бінарних об'єктів, підтримує журналювання операцій зі зміни і додавання даних в БД [8].

Можливості MongoDB:

- гнучка мова формування запитів;
- ефективне зберігання бінарних даних великих обсягів, фото або відео;
- документно-орієнтоване сховище;
- відмовостійкість та масштабованість;
- підтримка індексів;

– журнулювання операцій.

Рис. 2.4. Логотип MongoDB

Swagger

Swagger – це фреймворк, завдяки якому можна візуалізувати доступні запити, що оброблюються сервером, дізнатися про вхідний та вихідний формат даних. Також swagger зручно виконувати запити на сервер.

Зазвичай swagger використовують backend-розробники для надання певної документації про можливості програми іншим розробникам. Swagger відображає усі оброблювані сервером запити, лише у пару кліків можна зробити необхідний запит та подивитись на відповідь серверу.

У розроблюваній програмі Swagger активно використовувався для виконання запитів на сервер під час тестування та розробки.

Docker

Docker — це платформа для контейнеризації з відкритим кодом. Программа дозволяє розробникам упаковувати програми в контейнери — стандартизовані виконувані компоненти, що поєднують вихідний код програми з бібліотеками операційної системи і залежностями, необхідними для виконання цього коду в будь-якому середовищі.

Контейнери спрощують розгортання розподілених додатків і стають дедалі популярнішими, оскільки організації переходять на власну хмарну розробку та гібридні мультихмарні середовища. Розробники можуть створювати контейнери без Docker, але платформа робить розгортання та керування контейнерами легшим та безпечнішим [9].

2.4. Опис структури системи та алгоритмів її функціонування

Програма логічно розбита на директорії, які відповідають за ту чи іншу тематику:

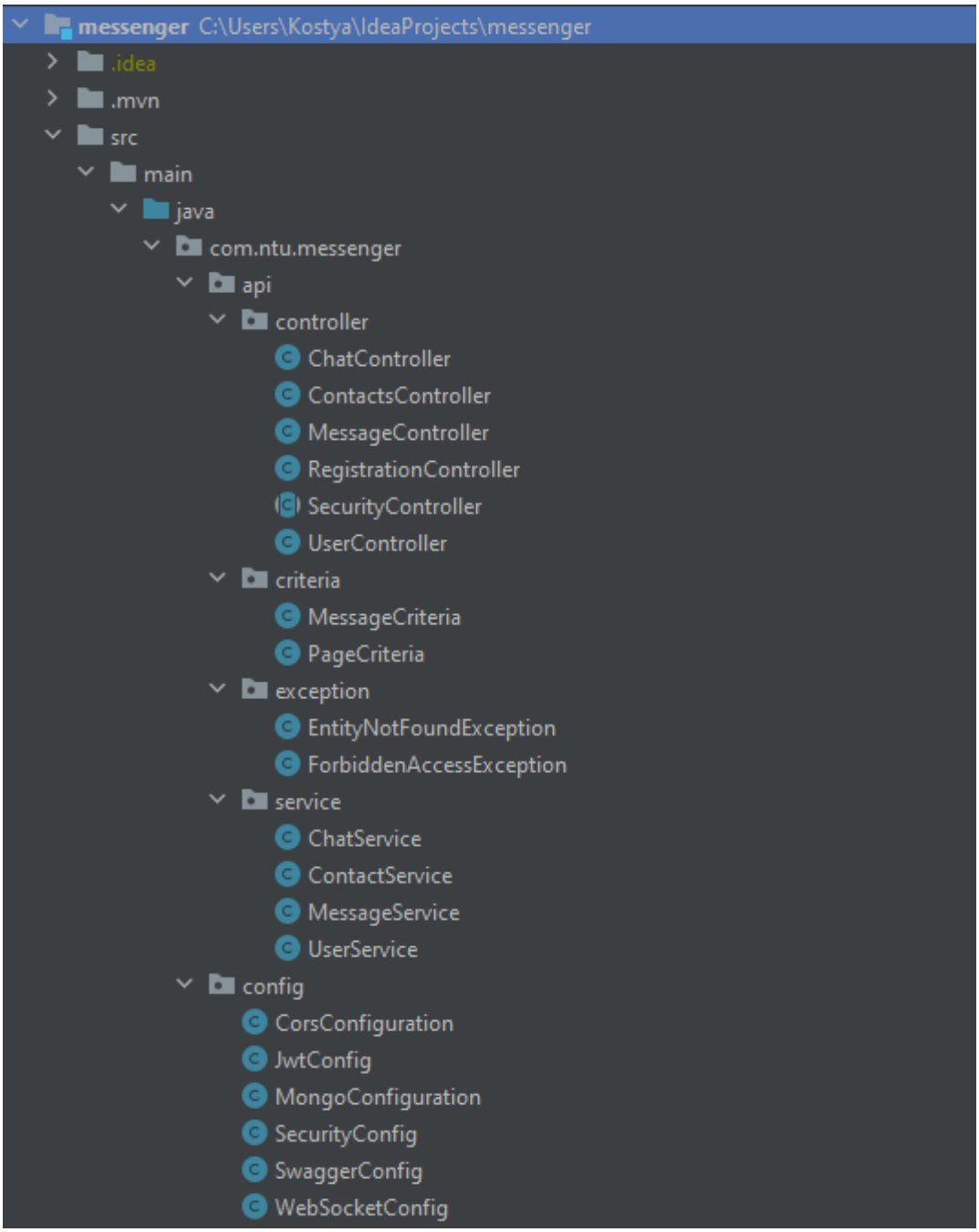

Рис. 2.5. Директорії Controller, Criteria, Exception, Service, Config

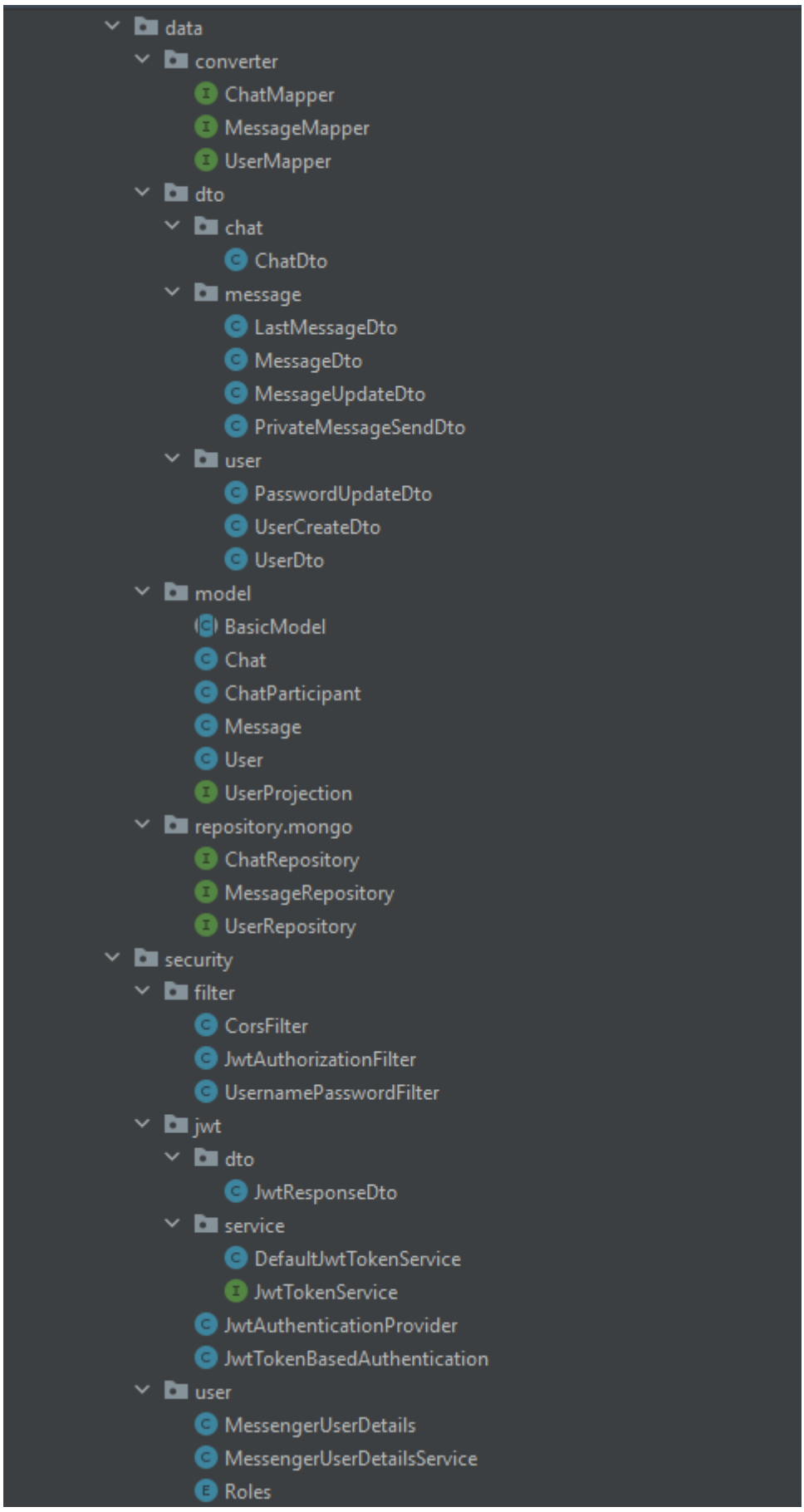

Рис. 2.6. Інші директорії з класами

Рис. 2.7. Інші файли проекту

Структурно додаток можна поділити на дві основні частини:

– директорії з java класами;

– директорії з допоміжними файлами (файли конфігурацій, скрипти).

У директорії api/controller знаходяться класи, в яких міститься інформація про те, які сервер оброблює адреси та які результати повинен повернути після обробки.

Директорія api/criteria містить два класи, які дозволяють реалізувати на сервері пагінацію запитів.

Директорія api/exception містить два класи, які використовуються коли виникає певна виключна ситуація – не було знайдено запис в базі, користувач не має доступу до функціоналу.

Директорія api/service містить класи, які виконують основний функціонал додатку.

У директорії config містяться класи, що повністю конфігурують компоненти додатку: налаштування бази даних, системи безпеки тощо.

Директорія data/converter містить допоміжні інтерфейси для легкої конвертації одного класу в інший.

Директорія data/dto містить так звані DTO (Data Transfer Object) полегшені класи, які використовують аби розбити певний клас на декілька тематичних класів менших за розміром.

Директорія data/model містить класи, що описують таблиці в базі даних.

У директорії repository/mongo містяться інтерфейси, які дозволяють отримати доступ до операцій з базою даних, такий функціонал надає фреймворк Spring Data.

Директорія security/filters містить необхідні фільтри які проходить кожен HTTP запит перед тим як він буде оброблений сервером.

Директорія security/jwt містить логіку пов’язану з аутентифікацією користувача по jwt-токену.

Також важливими файлами проекту є:

- application.yaml – файл, що містить певні конфігурації, але у yaml форматі, є альтернативою до конфігурування у Java коді;
- pom.xml – файл, який містить перелік бібліотек, які необхідні для роботи проекту, при процесі побудови проекту необхідні бібліотеку завантажуються у систему;
- docker-compose.yaml – файл, з описом програмних компонентів, який необхідний системі Docker для контейнеризації додатку.

Алгоритм обробки запитів розроблюваної програми зображена на рисунку нижче.

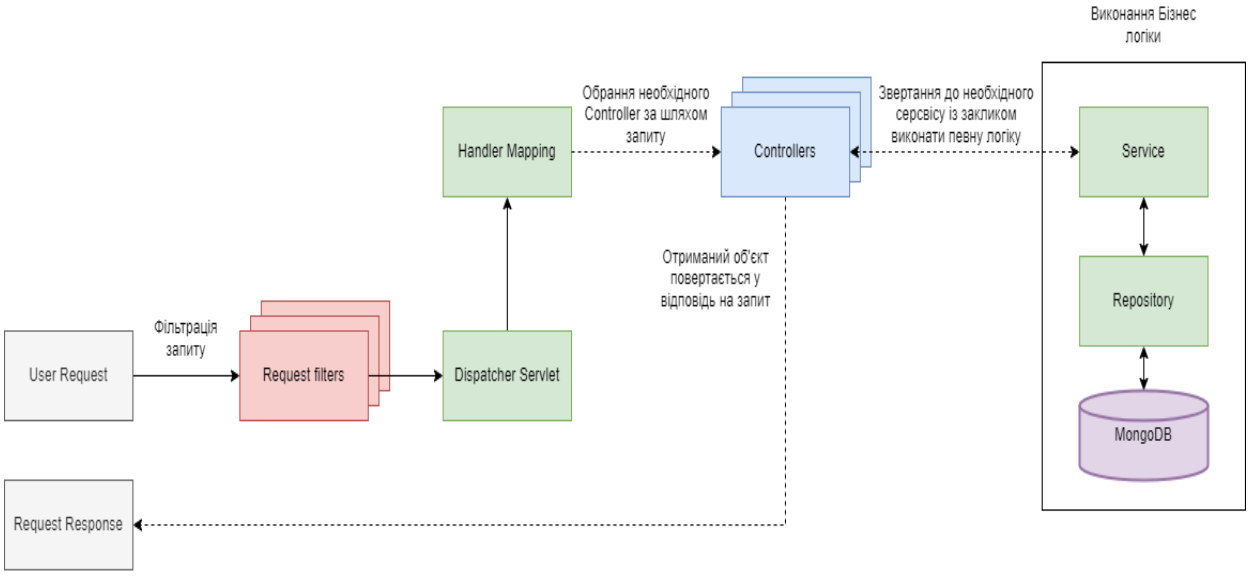

Рис. 2.8. Обробка запитів

Спочатку усі запити проходять через фільтри, де перевіряється чи користувач той, за кого себе видає, з якого домену йде запит, чи не закінчився час дії JWT-токена користувача.

При успішній фільтрації запит потрапляє в DispatcherServlet, спеціальний клас який є вхідною точкою усіх запитів. В DispatcherServlet використовується Handler Mapping – метод, який визначає до якого контролеру спрямувати запит.

Після попадання у контролер, програма передає параметри запиту до необхідного сервісу, який у свою чергу використовує потрібний йому репозиторій, що взаємодіє з базою даних.

Після певних операцій сервіс повертає об’єкт-результат контролеру, а контролер повертає відповідь клієнту.

Алгоритм запитів до БД

Запити до БД створюються через спеціальний прошарок абстракції, який надає фреймворк Spring Data, це звільнює розробника від написання типових запитів та дуже прискорює розробку. Від розробника необхідно створити так званий репозиторій та унаслідувати його від MongoRepository, створити сигнатуру методу без реалізації, слідуючи певним семантичним правилам. Наприклад, якщо у моделі даних є поле Username і необхідно знайти користувача за цим ім’ям, то розробник оголошує метод findByUsername та передає у якості аргументу методу Username, а сам фреймворк вже реалізовує побудову подібного запиту до БД. Фреймворк дозволяє створювати запити різноманітної складності, використовуючи сортування або фільтрування.

```
public interface MessageRepository extends MongoRepository<Message, String> {

    1 usage  Kostiantyn Vozniuk
    Page<Message> getMessagesByChatIdOrderByCreateDate(String chatId, Pageable page);
    1 usage  Kostiantyn Vozniuk
    Optional<Message> findFirstByChatIdOrderByCreateDateDesc(String chatId);
    1 usage  Kostiantyn Vozniuk
    void deleteAllByChatId(String chatId);
}
```

Рис. 2.9. Приклад репозиторію MessageRepository

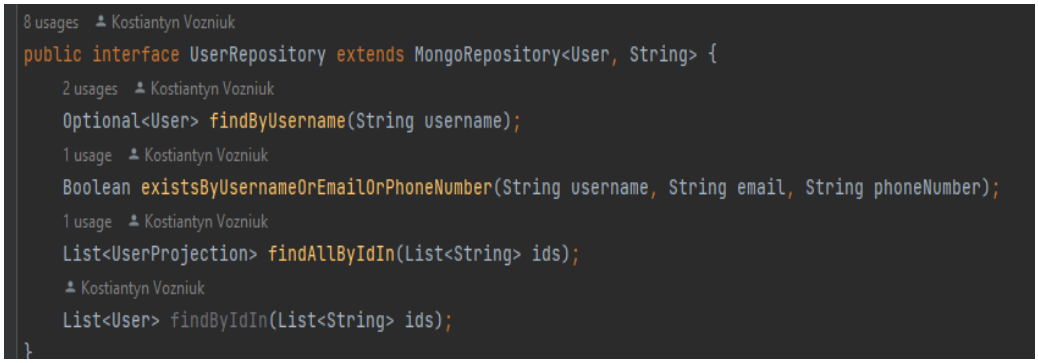

Рис 2.10. Приклад репозиторій UserRepository

Також репозиторій надають API, який дозволяє зберігати, видаляти, модифікувати записи у БД, основні методи:

– findAll() – знаходить усі записи у БД;
– findById(String id) – знаходить запис за переданим ідентифікатором;
– save(Object object) – зберігає дані в базу або оновлює їх якщо вони існують;
– delete(Object object) – видаляє запис з бази;
– deleteById(String id) – видаляє запис за ідентифікатором.

Іноді засобів фреймворку недостатньо і тоді розробник має використовувати інший API для комунікації з БД. У розроблюваній програмі для таких випадків був використаний MongoTemplate, клас, що дозволяє спілкуватися з БД у об’єктно-орієнтованому стилі.

Рис. 2.11. Приклад використання MongoTemplate

Комунікація між сервером та клієнтом

Комунікація між сервером та клієнтом відбувається шляхом надсилання запитів . Усі запити повинні мати у своїх заголовках заголовок Authorization, який містить JWT-токен, токен можна отримати при авторизації.

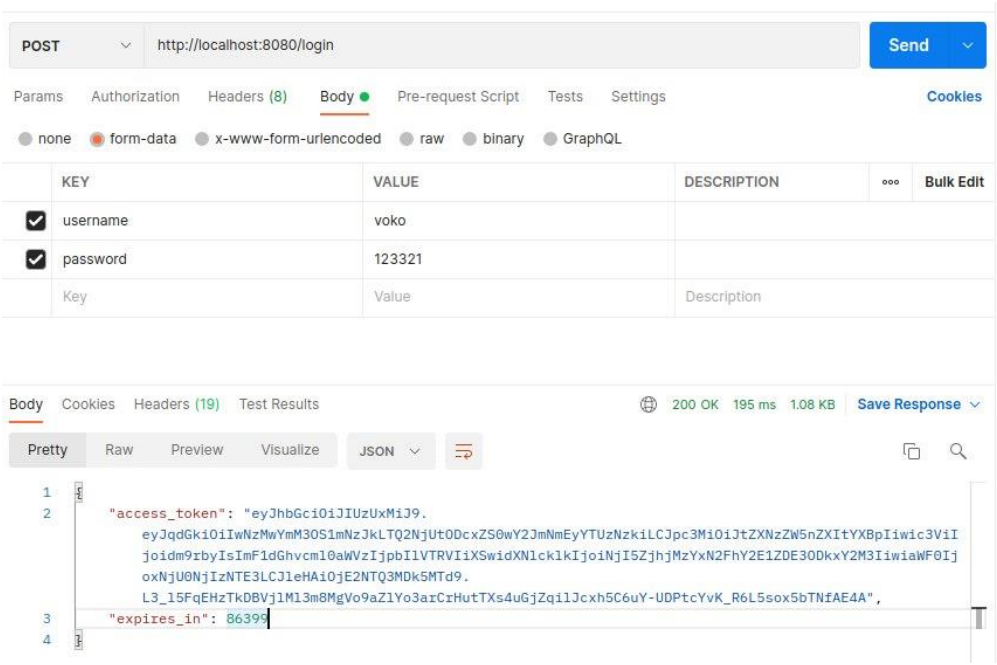

Рис. 2.12. Отримання токену від серверу

Таблиця 2.2.

Опис запитів до серверу

HTTP метод	Шлях	Тіло запиту (назва поля - тип)	Параметри	Опис
GET	api/v1/chat/	-	-	Повертає список чатів
DELETE	api/v1/chat/{chatId}	-	Ідентифікатор чату	Видаляє чат
GET	api/v1/contacts?size={size}&page={page}	-	Сторінка, розмір	Повертає список контактів

Продовження таблиці 2.2.

HTTP метод	Шлях	Тіло запиту (назва поля - тип)	Параметри	Опис
GET	api/v1/contacts/present/{id}	-	Ідентифікатор користувача	Повертає чи користувач є у списку контактів
POST	api/v1/contacts/{userId}	-	Ідентифікатор користувача	Додає користувача до списку контактів
DELETE	api/v1/contacts/{userId}	-	Ідентифікатор користувача	Видаляє користувача із списку контактів
GET	api/v1/messages/last	-	-	Повертає останнє повідомлення кожного чату користувача
GET	api/v1/messages/last/chat/{id}	-	-	Повертає останнє повідомлення певного чату
POST	api/v1/messages/	text - string, recepientId - string	-	Надсилає повідомлення користувачу
PUT	api/v1/messages/{id}	text - string	-	Редагування повідомлення
DELETE	api/v1/messages/{id}	-	-	Видалення повідомлення
POST	/login	username - string, password - string	-	Повертає у відповідь JWT токен для використання API

Закінчення таблиці 2.2.

HTTP метод	Шлях	Тіло запиту (назва поля - тип)	Параметри	Опис
POST	/sign-up	username - string, firstname - string, lastname - string, password - string, email - string, phoneNumber - string	-	Створення користувача у системі
GET	api/v1/user/	-	-	Повертає інформацію про авторизованого користувача
PUT	api/v1/user/	username - string, firstname - string, lastname - string, email - string, phoneNumber - string	-	Оновлює особисту інформацію користувача
PUT	api/v1/user/password	password - string	-	Оновлює пароль користувача

Структура бази даних

У розроблюваній програмі використовуються нереляційна база даних MongoDB. Замість поняття таблиці у таких БД використовують поняття колекції, а замість рядку таблиці – документ. Так як такі бази дозволяють створювати гнучкі моделі, не прив’язані до жорсткого формату, то кількість рядків і структура може відрізнятися у різних документах. Нижче наведена структура БД у більш звичному реляційному форматі.

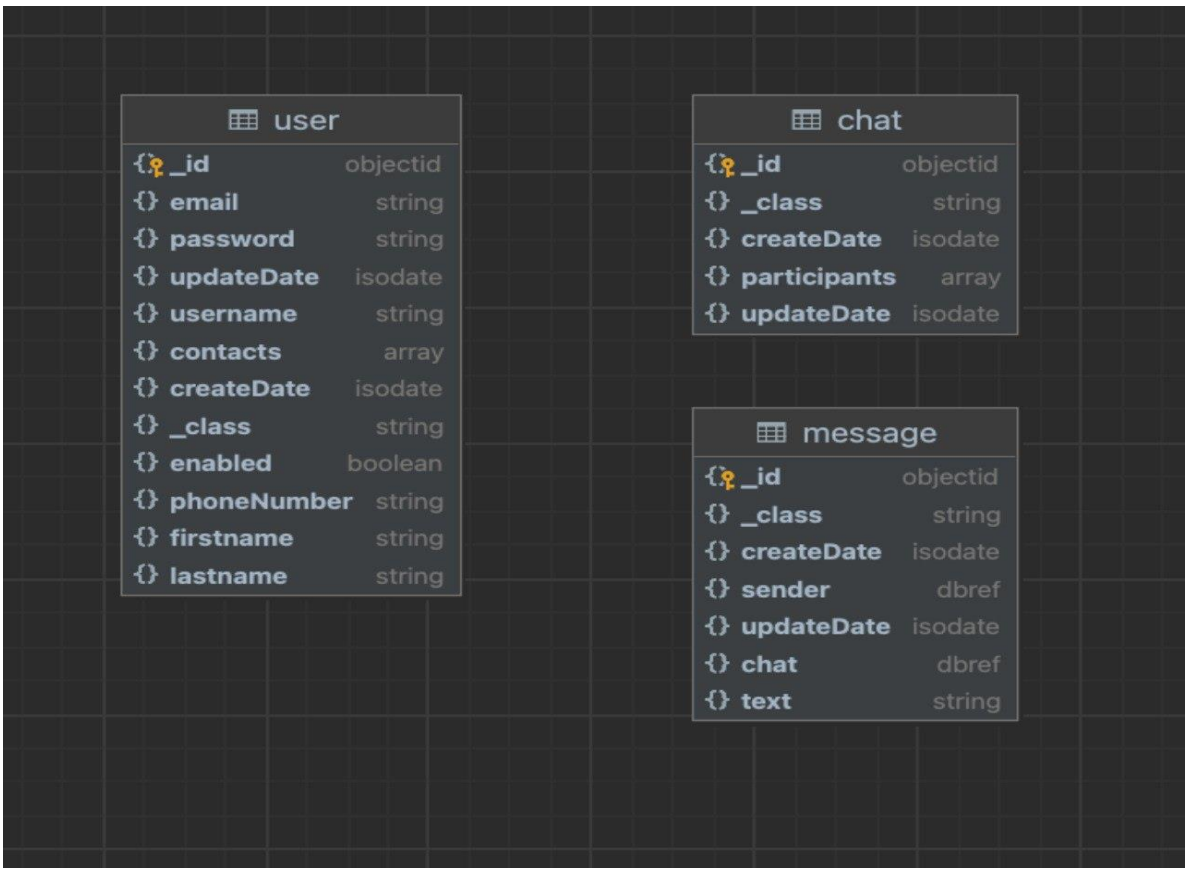

Рис. 2.13. Структура таблиць БД

Таблиця «user» містить дані про користувача, список його контактів, пароль у зашифрованому вигляді.

Таблиця «chat» містить список учасників.

Таблиця «message» містить інформацію про повідомлення, текст у зашифрованому вигляді, спеціальний об’єкт-посилання на документ в базі, що відображає відправника, та посилання на чат.

У таблицях 2.2. – 2.4. наведено описи полів колекцій.

Таблиця 2.3.

Колекція User

Колонка	Тип	Опис
_id	ObjectId	Унікальний ідентифікатор
username	String	Логін, унікальне поле

Закінчення таблиці 2.3.

Колонка	Тип	Опис
phoneNumber	String	Номер телефону, унікальне поле
firstName	String	Ім'я користувача
lastName	String	Прізвища користувача
enabled	Boolean	Чи активний обліковий запис
password	String	Пароль у зашифрованому вигляді
contacts	Array	Список ідентифікаторів інших користувачів
createDate	IsoDate	Час створення користувача
updateDate	IsoDate	Час оновлення користувача
email	String	Пошта, унікальне поле

Таблиця 2.4.

Колекція Message

Колонка	Тип	Опис
_id	ObjectId	Унікальний ідентифікатор
sender	DBRef	Відправник повідомлення
chat	DBRef	Чат у який повідомлення адресовано
text	String	Текст повідомлення у зашифрованому вигляді
createDate	IsoDate	Час створення повідомлення

Закінчення таблиці 2.4.

Колонка	Тип	Опис
updateDate	IsoDate	Час оновлення повідомлення

Таблиця 2.5.

Колекція Chat

Колонка	Тип	Опис
_id	ObjectId	Унікальний ідентифікатор
participants	Array	Ідентифікатори учасників
createDate	IsoDate	Час створення чату
updateDate	IsoDate	Час оновлення чату

Нижче наведено приклади даних та як вони виглядають у MongoDB.

Рис. 2.14. Приклад даних з таблиці User

Рис. 2.15. Приклад даних з таблиці Message

2.5. Обґрунтування та організація вхідних та вихідних даних програми

Вхідні дані потрапляють до розроблюваної програми у JSON форматі, вхідні дані формує клієнт. Вихідні дані також представлені у JSON форматі, їх структура та розмір залежать від запиту та кількості наявних даних.

2.6. Опис розробленої системи

2.6.1. Використані технічні засоби

Під час розроби ІС були використані наступні технічні засоби:

- оперативна пам’ять обсягом 16 ГБ;
- процесор Intel Core i5 6500 3.2GHz;
- SSD об’ємом 500ГБ;
- HDD об’ємом 1ТБ
- відеоадаптер NVidia GeForce 970;

2.6.2. Використані програмні засоби

Під час розробки використовувались наступні програмні засоби:

- середовище розробки IntelliJ IDEA;

– операційні системи Windows 10 та Ubuntu;
– Docker Desktop;
– Heroku;
– Git, Github.

IntelliJIDEA

IntelliJIDEA – інтегроване середовище розробки, найчастіше використовується для розробки на Java, Scala, Kotlin або Groovy. Має як комерційну так і безкоштовну версії.

Docker Desktop

Docker Desktop – це проста в установці програма для Mac або Windows, яка дає змогу створювати контейнерні програми та мікросервіси та ділитися ними. Docker Desktop включає Docker Engine, клієнт CLI Docker, Docker Compose, Docker Content Trust, Kubernetes і Credential Helper [10].

Heroku

Heroku – це хмарна PaaS платформа, яка підтримує багато популярних мов програмування. Heroku використовується розробниками для розгортання, масштабування та управління додатками. Основна перевага Heroku перед іншими подібними сервісами у тому, що у плафторми присутня можливість безкоштовного розготрання додатків.

Процес розгортання на Heroku дуже простий та швидкий. Найлегшим способом розготання додатку на Heroku є вказування репозиторію с програмний кодом на Github. Heroku візьме існуючий код та автоматично зробить розготання вказаної гілки. Також якщо додаток використовує певні ресурси, наприклад базу даних, і розробник бажає контролювати їх разом із додатком на Heroku то існує можливість створити їх при розгортанні.

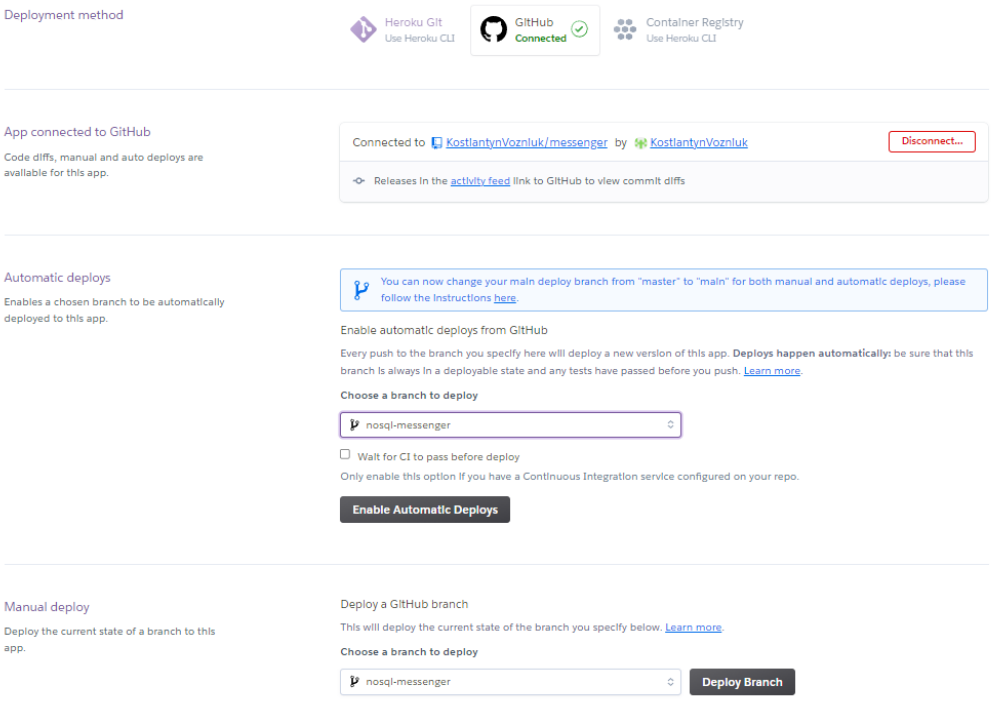

Рис 2.16. Розгортання на Heroku

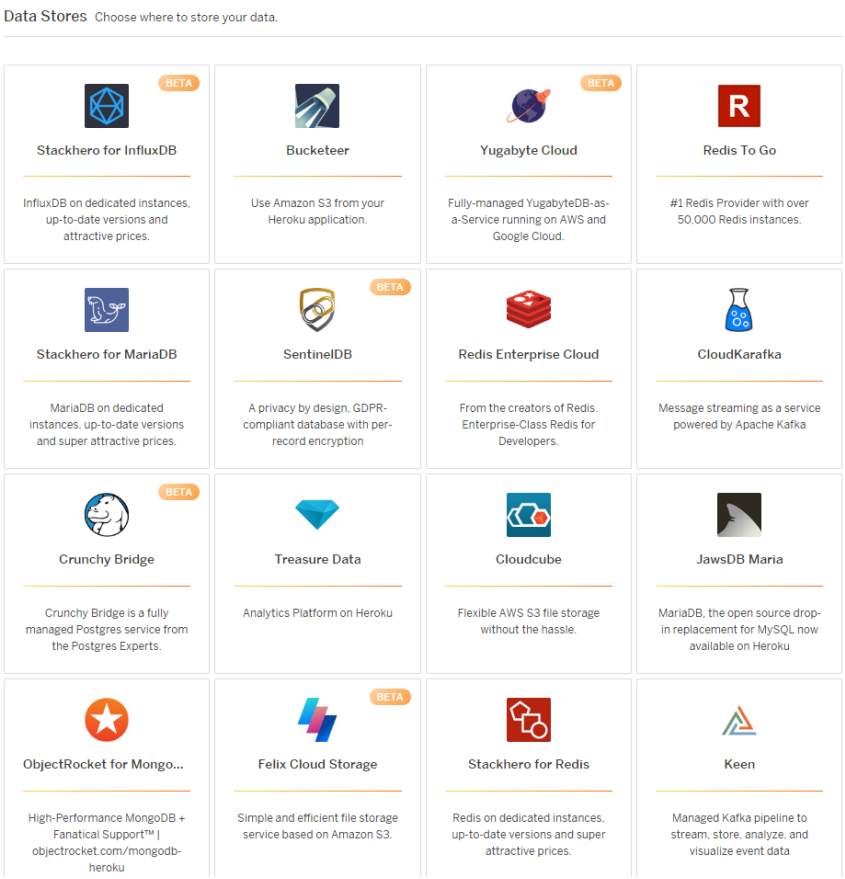

Рис 2.17. Додаткові ресурси Heroku

Git, Github

Git – це безкоштовна розподілена система контролю версій з відкритим вихідним кодом, призначена для швидкої та ефективної роботи з усім, від малих до дуже великих проектів. Він перевершує інструменти SCM, такі як Subversion, CVS, Perforce і ClearCase, завдяки таким функціям, як локальне розгалуження, зручні області переходу та кілька робочих процесів [11].

Github – вебсервіс, що надає можливість зберігання та версіонування програмного забезпечення. Базується на системі керування версіями Git.

2.6.3. Виклик та завантаження програми

Розроблений програмний продукт, а саме серверну та клієнтську частини можна знайти за адресами [12] та [13] відповідно. Також додаток можна розгорнути локально, для цього необхідно заватанжити вихідний програмний код та середовище Docker, після завантаження необхідно викликати команду docker-compose up -d у командному рядку.

2.7.3. Опис інтерфейсу користувача

Інтерфейс користувача створено за допомогою фреймворку Swagger. На сторінці прусутні усі ендпоінти, які обролює сервер, також наводиться список об’єктів, які слугують тілами запросів.

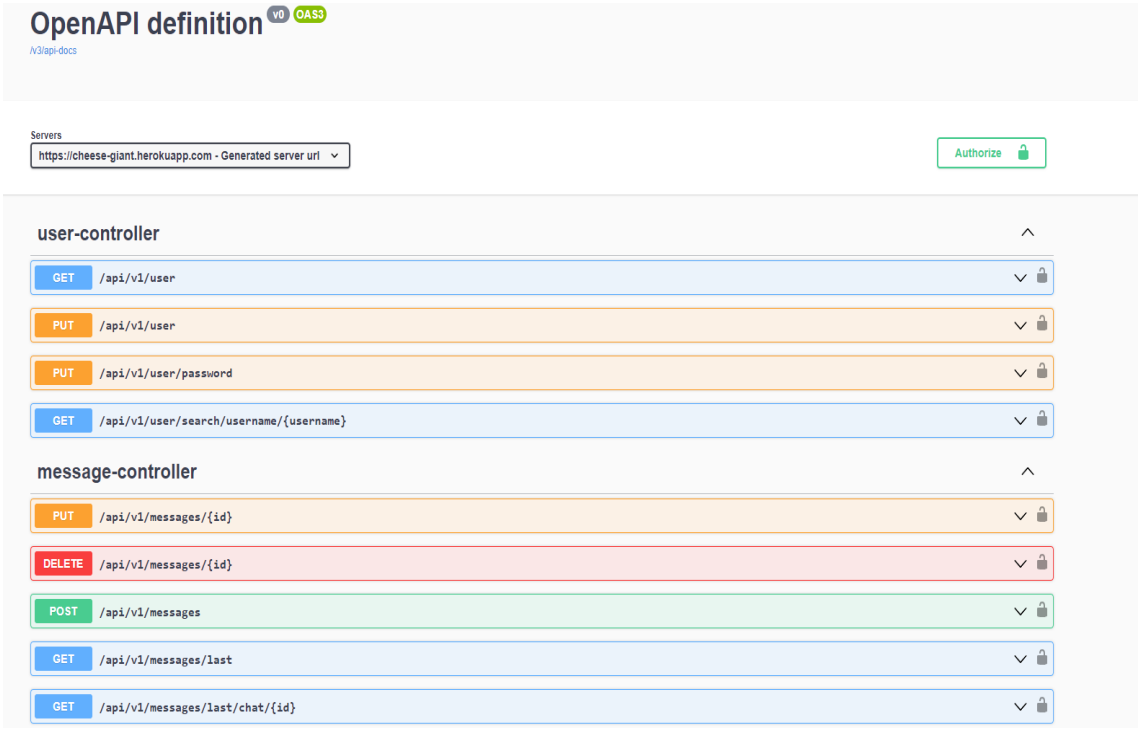

Рис. 2.18. Список існуючих ендпоінтів, частина 1

Рис. 2.19. Список існуючих ендпоінтів, частина 2

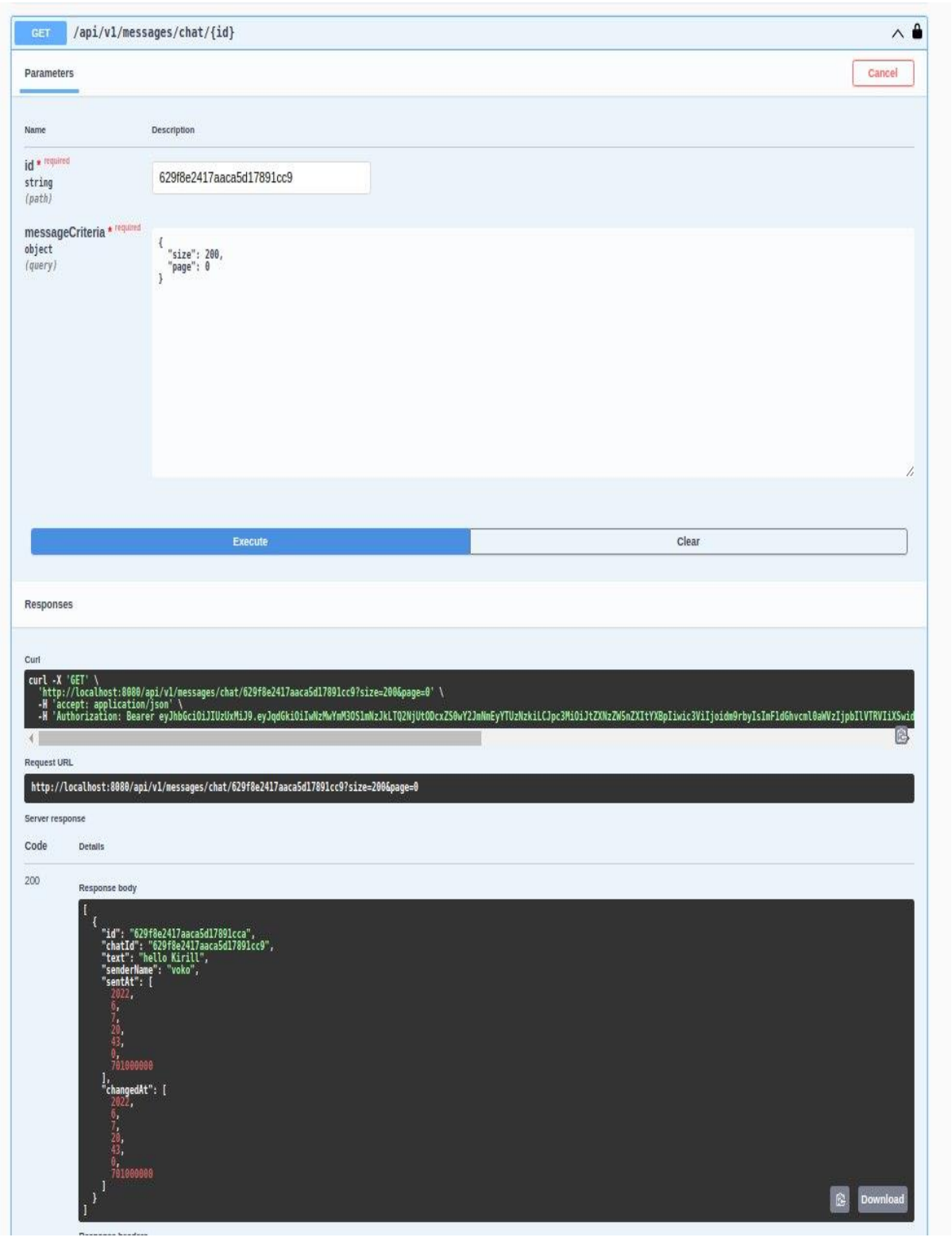

Рис. 2.20. Приклад відповіді від серверу на GET запит з параметрами

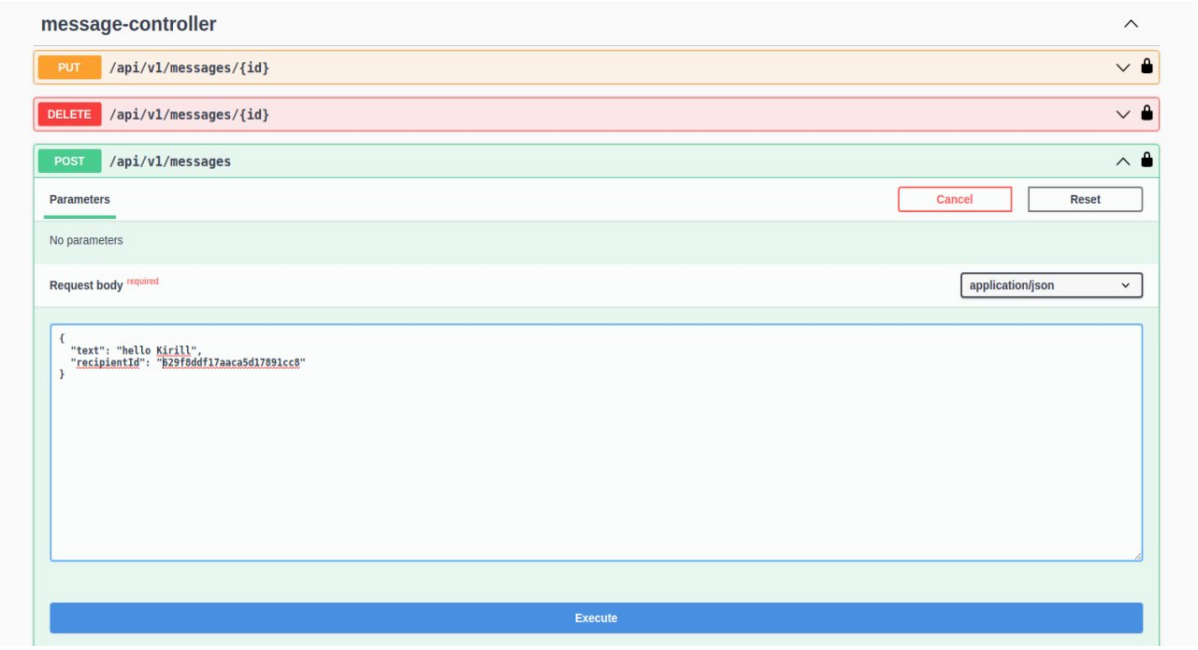

Рис 2.21. Відправка повідомлення через Swagger

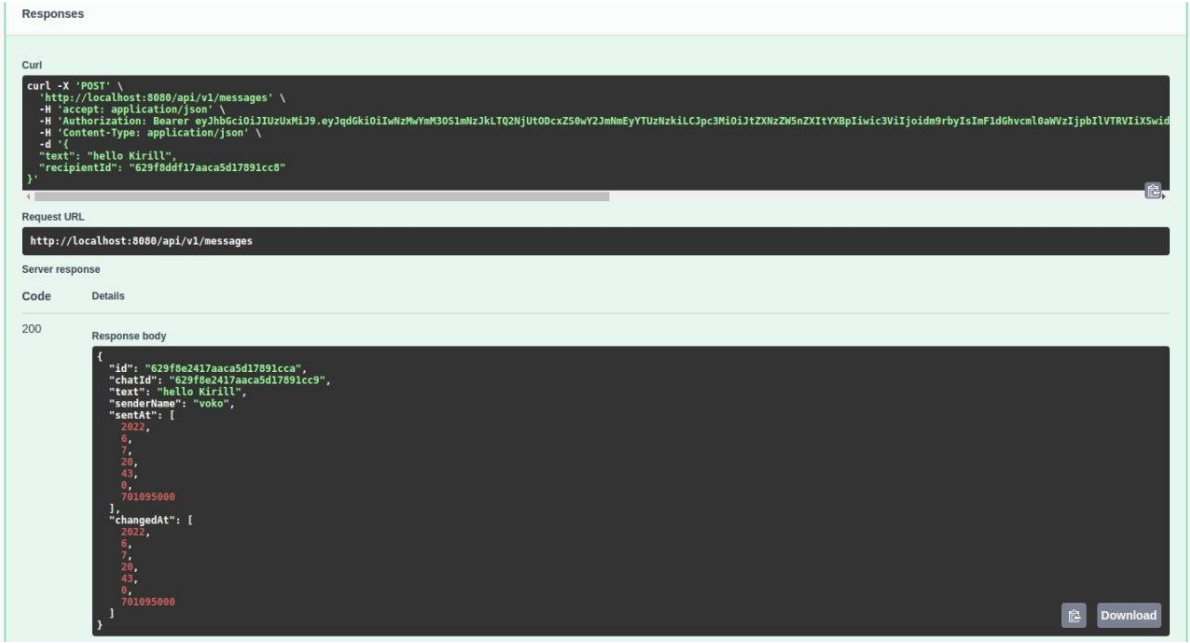

Рис. 2.22. Відповідь від серверу на відправку повідомлення

РОЗДІЛ 3

ЕКОНОМІЧНИЙ РОЗДІЛ

3.1. Розрахунок трудомісткості та вартості програмного продукту

Початкові дані:

1. передбачуване число операторів програми – 1771;
2. коефіцієнт корекції програми в ході її розробки – 0,05;
3. коефіцієнт складності програми – 1,5;
4. коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі – 1,3;
5. годинна заробітна плата програміста – 174 грн/год;

Середня годинна оплата програміста була вирахувана спираючись на відкриті джерела, а саме портал «Української спільноти програмістів (DOU)». В середньому Java розробник з досвідом роботи близько року отримує 700-1200 американських долларів [14]. Поточний курс НБУ станом на початок червня 2022 року становить 29,25 гривень за один американський доллар [15], отже середня зарплата в гривнях на місяць становить 27787 грн (950 долларів США).

6. коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності – 0,8;
7. вартість машино-години ЕОМ – 16,33 грн/год.

Комп’ютер у робочому стані споживає приблизно 250 Ватт / год, що еквівалентно 0,25 кВт / год. Вартість одного кВт на годину, станом на червень 2022 року, становить 1,68 грн [16]. Таким чином вартість споживання електроенергії комп’ютером за годину становить 0,42 грн. Вартість оренди комп’ютера на місяць становить 2800 грн. [17], при графіку роботи 176 годин на місяць, вартість оренди комп’ютера становитиме 15,91 грн. на годину. Остаточна вартість машинно-годино ЕОМ дорівнює 16,33 грн.

Нормування праці в процесі створення ПЗ істотно ускладнено в силу творчого характеру праці програміста. Тому трудомісткість розробки ПЗ може бути розрахована на основі системи моделей з різною точністю оцінки.

Трудомісткість розробки ПЗ розраховується за формулою:

$$t = t_o + t_u + t_a + t_n + t_{отл} + t_д \text{, людино-годин,} \quad (3.1)$$

де t_o – витрати праці на підготовку й опис поставленої задачі (приймається 50 людино-годин);

t_u – витрати праці на дослідження алгоритму рішення задачі;

t_a – витрати праці на розробку блок-схеми алгоритму;

t_n – витрати праці на програмування по готовій блок-схемі;

$t_{отл}$ – витрати праці на налагодження програми на ЕОМ;

$t_д$ – витрати праці на підготовку документації.

Через умовне число операторів програмного забезпечення визначаються складові витрати.

Умовне число операторів (підпрограм):

$$Q = q \cdot C \cdot (1 + p) \text{, людино-годин,} \quad (3.2)$$

де q – передбачуване число операторів (1771);

C – коефіцієнт складності програми (1,5);

p – коефіцієнт корекції програми в ході її розробки (0,05).

Звідси умовне число операторів в програмі:

$$Q = 1{,}5 \cdot 1771 \cdot (1 + 0{,}05) = 2789{,}325$$

Витрати праці на вивчення опису задачі t_u визначається з урахуванням уточнення опису і кваліфікації програміста:

$$t_u = \frac{Q \cdot B}{(75..85) \cdot k}\ , \text{людино-годин,} \qquad (3.3)$$

де B – коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі;

k – коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності. При стажі роботи до 2 років становить 0,8.

$$t_u = (2789{,}325 \cdot 1{,}2) / (75 \cdot 0{,}8) = 55{,}786 \text{ людино-годин.}$$

Витрати праці на розробку алгоритму рішення задачі можна обчислити за формулою:

$$t_a = \frac{Q}{(20\ldots25) \cdot k}\ , \text{людино-годин,} \qquad (3.4)$$

де Q – умовне число операторів програми;

k – коефіцієнт кваліфікації програміста.

Підставивши відповідні значення в формулу (3.2), отримаємо:

$$t_a = 2789{,}325 / (20 \cdot 0{,}8) = 174{,}3 \text{ людино-годин.}$$

Витрати на складання програми по готовій блок-схемі:

$$t_n = \frac{Q}{(20\ldots25) \cdot k}\ , \text{людино-годин.} \qquad (3.5)$$

$$t_n = 2789{,}325 / (25 \cdot 0{,}8) = 139{,}47 \text{ людино-годин.}$$

Витрати праці на налагодження програми на ЕОМ:

– за умови автономного налагодження одного завдання:

$$t_{отл} = \frac{Q}{(4..5) \cdot k} \text{ , людино-годин.} \quad (3.6)$$

$t_{отл}$ = 2789,325 / (5 · 0,8) = 697,33 людино-годин.

– за умови комплексного налагодження завдання:

$$t^{k}_{отл} = 1{,}5 \cdot t_{отл} \text{ , людино-годин.} \quad (3.7)$$

$t^{k}_{отл}$ = 1,5 · 697,33 = 1045,995 людино-годин.

Витрати праці на підготовку документації можна визначити за формулою:

$$t_{д} = t_{др} + t_{до} \text{ , людино-годин,} \quad (3.8)$$

де $t_{др}$ – трудомісткість підготовки матеріалів і рукопису:

$$t_{др} = \frac{Q}{(15..20) \cdot k} \text{ , людино-годин,} \quad (3.9)$$

$t_{до}$ - трудомісткість редагування, печатки й оформлення документації:

$$t_{до} = 0{,}75 \cdot t_{др} \text{ , людино-годин.} \quad (3.10)$$

Підставивши відповідні значення, отримаємо:

$t_{др}$ = 2789,325 / (17 · 0,8) = 205,1 людино-годин.

$t_{до}$ = 0,75 · 205,1 = 153,825 людино-годин.

$t_{д}$ = 205,1 + 153,825 = 358,925 людино-годин.

За допомоги формули (3.1) отримаємо повну оцінку трудомісткості розробки ПЗ.

t = 50 + 55,786 + 174,3 + 139,47 + 697,33 + 358,925 = 1476 людино-годин.

3.2. Розрахунок витрат на створення програми

Витрати на створення ПЗ включать у себе витрати на заробітну платню виконавця та витрати машинного часу необхідні для налагодження програми на ЕОМ.

$$K_{ПО} = З_{ЗП} + З_{МВ}, \text{ грн}, \tag{3.11}$$

де $З_{ЗП}$ – заробітна плата виконавців, що визначається за формулою:

$$З_{ЗП} = t \cdot C_{ПР}, \text{ грн}, \tag{3.12}$$

де t – загальна трудомісткість, людино-годин;

$C_{ПР}$ – середня годинна заробітна плата програміста, грн/година.

$З_{мв}$ – Вартість машинного часу, необхідного для налагодження програми на ЕОМ:

$$З_{мв} = t_{отл} \cdot C_{мч}, \text{ грн}, \tag{3.13}$$

де $t_{отл}$ - трудомісткість налагодження програми на ЕОМ, год;

$C_{мч}$ - вартість машино-години ЕОМ, грн/год (16,33 грн/год).

З урахуванням середньої плати у розмірі 174 грн/година та загальною трудомісткістю 1476 людиного-годин отримуємо:

$$З_{ЗП} = 1476 \cdot 174 = 256\ 824 \text{ грн.}$$

$$З_{мв} = 697{,}33 \cdot 16{,}33 = 11\ 387{,}4 \text{ грн.}$$

$$К_{ПО} = 256\ 824 + 11\ 387{,}4 = 268\ 211{,}4 \text{ грн.}$$

Очікуваний період створення ПЗ:

$$T = \frac{t}{B_K \cdot F_P} \text{ міс.} \qquad (3.14)$$

F_p - місячний фонд робочого часу (при 40 годинному робочому тижні складає 176 годин).

Звідси очікуваний час на створення програмного продукту:

$$T = 1476 / 1 \cdot 176 \approx 8{,}39 \text{ міс.}$$

Висновок: Вартість розробки програмного забезпечення, а саме серверної частини соціального месенджера, становить приблизно 268 211,4 грн, дане програмне забезпечення не передбачає додаткових витрат. Очікуваний період створення програмного продукту при 40-годинному робочому тижні становить 1476 години або 8,39 місяці.

ВИСНОВКИ

Метою кваліфікаційної роботи було створення інформаційної системи, що може бути використана мобільним або браузерним клієнтом, матиме зручний програмний інтерфейс та необхідний функціонал.

В результаті виконання кваліфікаційної роботи була розроблена серверна частина соціального месенджеру. Не зважаючи на існуючи конкурецнію, розробка додатку є актуальною, через наявність певних у існуючих аналогів таких недоліків як наявність реклами, перевантаженість, відсутність кросплатформенності.

Під час виконання завдання кваліфікаційної роботи були винонані наступні задачі:

1. створення алгоритму для реалізації задачі;
2. розробка серверної частини програми;
3. створення моделі бази даних;
4. розгортання програми на стороньому сервісі.

Створений програмний продукт надає наступний функціонал:

1. можливість зареєструватися та авторизуватися у системі;
2. можливість редагувати особисту іноформацію;
3. користувач має книгу контактів та може редагувати її;
4. можливість пошуку інших користувачів;
5. обмін повідомлення з іншими користувачами у режимі реального часу;
6. редагування та видалення власних повідомлень.

Програма розроблена з використанням мови програмування Java та спеціального фреймворку для WEB розробки Spring.

У кваліфікаційній роботі було визначено трудомісткість та порахована вартість розробленої системи. Загальна вартість програмного продукту становить 268 211,4 грн, час на створення – 1476 години або 8,39 місяці.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Дещо про історію виникнення глобальної мережі інтернет [Електронний ресурс] – Режим доступу: http://conf.nlu.edu.ua/bis-2016/paper/viewFile/3963/663
2. Миттєві повідомлення [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/Миттєві_повідомлення
3. Viber [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/viber
4. Telegram [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/telegram
5. What is REST [Електронний ресурс] – Режим доступу: https://www.codecademy.com/article/what-is-rest
6. Крейг Воллс. Spring in Action. – 521 с. ISBN: 9781617294945
7. Spring Security [Електронний ресурс] – Режим доступу: https://spring.io/projects/spring-security
8. MongoDB [Електронний ресурс] – Режим доступу: https://en.wikipedia.org/wiki/MongoDB
9. Docker [Електронний ресурс] – Режим доступу: https://www.ibm.com/in-en/cloud/learn/docker
10. Docker Desktop [Електронний ресурс] – Режим доступу: https://docs.docker.com/desktop/
11. Git [Електронний ресурс] – Режим доступу: https://git-scm.com/
12. Серверна частина додатку [Електронний ресурс] – Режим доступу: https://cheese-giant.herokuapp.com/swagger-ui/index.html
13. Клієнтська частина додатку [Електронний ресурс] – Режим доступу: https://messenger-memesenger.herokuapp.com/
14. Середня заробітна плата Junior Software Engineer [Електронний ресурс] – Режим доступу: https://jobs.dou.ua/salaries/?period=2021-12&position=Junior%20SE&technology=Java&experience=0-1

15. Курс НБУ [Електронний ресурс] – Режим доступу: https://bank.gov.ua/ua/markets/exchangerates
16. Тарифи на електроенергію [Електронний ресурс] – Режим доступу: https://yasno.com.ua/b2c-tariffs
17. Оренда ноутбука [Електронний ресурс] – Режим доступу: http://prokatgopro.dp.ua/arenda_noutbuka_dnepr.html
18. MongoDb tutorial [Електронний ресурс] – Режим доступу: https://www.baeldung.com/spring-data-mongodb-tutorial
19. Introduction to Docker-compose [Електронний ресурс] – Режим доступу: https://www.baeldung.com/ops/docker-compose
20. Кристіна Чодоров. MongoDB: The Definitive Guide. – 349 с. ISBN: 9781491954393
21. Introduction to Spring Data MongoDB [Електронний ресурс] – Режим доступу: https://www.baeldung.com/spring-data-mongodb-tutorial
22. Використання WebSocket для створення веб-програми [Електронний ресурс] – Режим доступу: https://spring.io/guides/gs/messaging-stomp-websocket/
23. Intro to WebSockets with Spring програми [Електронний ресурс] – Режим доступу: https://www.baeldung.com/websockets-spring

ДОДАТОК А

КОД ПРОГРАМИ

ChatController.java

```
package com.ntu.messenger.api.controller;

import com.ntu.messenger.api.service.ChatService;
import com.ntu.messenger.api.service.UserService;
import com.ntu.messenger.config.SwaggerConfig;
import com.ntu.messenger.data.converter.ChatMapper;
import com.ntu.messenger.data.dto.chat.ChatDto;
import com.ntu.messenger.data.model.Chat;
import com.ntu.messenger.data.model.User;
import io.swagger.v3.oas.annotations.Operation;
import io.swagger.v3.oas.annotations.security.SecurityRequirement;
import lombok.RequiredArgsConstructor;
import org.springframework.http.HttpStatus;
import org.springframework.http.MediaType;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import java.util.List;

@RestController
@RequestMapping(path = "api/v1/chat")
@RequiredArgsConstructor
public class ChatController extends SecurityController {

    private final ChatService chatService;
    private final UserService userService;

    @Operation(security = {@SecurityRequirement(name = SwaggerConfig.BEARER_KEY)})
    @GetMapping(produces = MediaType.APPLICATION_JSON_VALUE)
    public ResponseEntity<List<ChatDto>> getUserChats() {
        User current = userService.findUserById(getUserDetails().getId());
        List<Chat> userChats = chatService.getUserChats(current);
        return new ResponseEntity<>(ChatMapper.MAPPER.map(userChats), HttpStatus.OK);
    }

    @Operation(security = {@SecurityRequirement(name = SwaggerConfig.BEARER_KEY)})
    @DeleteMapping(path = "{chatId}")
    @ResponseStatus(code = HttpStatus.NO_CONTENT)
    public void deleteUserChat(@PathVariable("chatId") String chatId) {
        User requester = userService.findUserById(getUserDetails().getId());
        chatService.removeChat(requester, chatId);
    }
}
```

ContactsController.java

```
package com.ntu.messenger.api.controller;

import com.ntu.messenger.api.criteria.PageCriteria;
import com.ntu.messenger.api.service.ContactService;
import com.ntu.messenger.api.service.UserService;
import com.ntu.messenger.config.SwaggerConfig;
```

```
import com.ntu.messenger.data.model.User;
import com.ntu.messenger.data.model.UserProjection;
import io.swagger.v3.oas.annotations.Operation;
import io.swagger.v3.oas.annotations.security.SecurityRequirement;
import lombok.RequiredArgsConstructor;
import org.springframework.http.HttpStatus;
import org.springframework.http.MediaType;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import javax.validation.Valid;
import java.util.List;

@RestController
@RequestMapping(path = "api/v1/contacts")
@RequiredArgsConstructor
public class ContactsController extends SecurityController {

    private final UserService userService;
    private final ContactService contactService;

    @Operation(security = {@SecurityRequirement(name = SwaggerConfig.BEARER_KEY)})
    @GetMapping(produces = MediaType.APPLICATION_JSON_VALUE)
    public ResponseEntity<List<UserProjection>> getUserContacts(@Valid PageCriteria pageCriteria) {
        User current = userService.findUserById(getUserDetails().getId());
        List<UserProjection> userContacts = contactService.getUserContacts(current, pageCriteria.getSize(),
pageCriteria.getPage());
        return new ResponseEntity<>(userContacts, HttpStatus.OK);
    }

    @Operation(security = {@SecurityRequirement(name = SwaggerConfig.BEARER_KEY)})
    @GetMapping(path = "present/{id}", produces = MediaType.APPLICATION_JSON_VALUE)
    public ResponseEntity<Boolean> isInContactList(@PathVariable("id") String id) {
        User current = userService.findUserById(getUserDetails().getId());
        return new ResponseEntity<>(contactService.isInContactList(current, id), HttpStatus.OK);
    }

    @Operation(security = {@SecurityRequirement(name = SwaggerConfig.BEARER_KEY)})
    @PostMapping("{id}")
    @ResponseStatus(HttpStatus.CREATED)
    public void addContact(@PathVariable("id") String id) {
        User current = userService.findUserById(getUserDetails().getId());
        contactService.addContact(current, id);
    }

    @Operation(security = {@SecurityRequirement(name = SwaggerConfig.BEARER_KEY)})
    @DeleteMapping("{id}")
    @ResponseStatus(HttpStatus.ACCEPTED)
    public void removeContact(@PathVariable("id") String id) {
        User current = userService.findUserById(getUserDetails().getId());
        contactService.removeContact(current, id);
    }
}
```

MessageController.java

```
package com.ntu.messenger.api.controller;

import com.ntu.messenger.api.criteria.MessageCriteria;
import com.ntu.messenger.api.service.MessageService;
import com.ntu.messenger.api.service.UserService;
import com.ntu.messenger.config.SwaggerConfig;
import com.ntu.messenger.data.converter.MessageMapper;
import com.ntu.messenger.data.dto.message.LastMessageDto;
```

```
import com.ntu.messenger.data.dto.message.MessageDto;
import com.ntu.messenger.data.dto.message.PrivateMessageSendDto;
import com.ntu.messenger.data.dto.message.MessageUpdateDto;
import com.ntu.messenger.data.model.Message;
import com.ntu.messenger.data.model.User;
import io.swagger.v3.oas.annotations.Operation;
import io.swagger.v3.oas.annotations.security.SecurityRequirement;
import lombok.RequiredArgsConstructor;
import org.springframework.http.HttpStatus;
import org.springframework.http.MediaType;
import org.springframework.messaging.simp.SimpMessagingTemplate;
import org.springframework.web.bind.annotation.*;
import javax.validation.Valid;
import java.util.List;

import static java.util.stream.Collectors.toList;

@RestController
@RequiredArgsConstructor
@RequestMapping(path = "api/v1/messages")
public class MessageController extends SecurityController {

    private final MessageService messageService;
    private final UserService userService;
    private final SimpMessagingTemplate messagingTemplate;

    @Operation(security = {@SecurityRequirement(name = SwaggerConfig.BEARER_KEY)})
    @GetMapping(path = "chat/{id}", produces = MediaType.APPLICATION_JSON_VALUE)
    public List<MessageDto> getMessagesByChat(@PathVariable("id") String chatId, @Valid MessageCriteria
messageCriteria) {
        User requester = userService.findUserById(getUserDetails().getId());
        List<Message> messages = messageService.getMessagesByChatId(chatId, requester, messageCriteria);
        return messageService.decryptMessages(messages);
    }

    @Operation(security = {@SecurityRequirement(name = SwaggerConfig.BEARER_KEY)})
    @GetMapping(path = "last")
    public List<LastMessageDto> getLastChatMessages() {
        User requester = userService.findUserById(getUserDetails().getId());
        List<Message> encrypted = messageService.getChatsLastMessages(requester);
        return messageService.decryptMessages(encrypted)
                        .stream()
                        .map(MessageMapper.MAPPER::mapToLastMessage)
                        .collect(toList());
    }

    @Operation(security = {@SecurityRequirement(name = SwaggerConfig.BEARER_KEY)})
    @GetMapping(path = "last/chat/{id}", produces = MediaType.APPLICATION_JSON_VALUE)
    public MessageDto getLastMessage(@PathVariable("id") String chatId) {
        User requester = userService.findUserById(getUserDetails().getId());
        Message message = messageService.getChatLastMessage(chatId, requester);
        return messageService.decryptMessage(message);
    }

    @Operation(security = {@SecurityRequirement(name = SwaggerConfig.BEARER_KEY)})
    @PostMapping(produces = MediaType.APPLICATION_JSON_VALUE, consumes =
MediaType.APPLICATION_JSON_VALUE)
    public MessageDto sendPrivateMessage(@RequestBody @Valid PrivateMessageSendDto privateMessageSendDto)
{
        String senderId = getUserDetails().getId();
        Message encrypted = messageService.savePrivateMessage(senderId, privateMessageSendDto);
        MessageDto dto = messageService.decryptMessage(encrypted);
```

```
        messagingTemplate.convertAndSend("/topic/messages/user/" + privateMessageSendDto.getRecipientId(), dto);
        return dto;
    }

    @Operation(security = {@SecurityRequirement(name = SwaggerConfig.BEARER_KEY)})
    @DeleteMapping(path = "{id}")
    @ResponseStatus(HttpStatus.ACCEPTED)
    public void deleteMessage(@PathVariable("id") String messageId) {
        User requester = userService.findUserById(getUserDetails().getId());
        messageService.deleteMessage(messageId, requester);
    }

    @Operation(security = {@SecurityRequirement(name = SwaggerConfig.BEARER_KEY)})
    @PutMapping(path = "{id}", consumes = MediaType.APPLICATION_JSON_VALUE)
    @ResponseStatus(HttpStatus.ACCEPTED)
    public void modifyMessage(@PathVariable("id") String messageId, @RequestBody MessageUpdateDto updateDto)
{
        User requester = userService.findUserById(getUserDetails().getId());
        messageService.modifyMessage(messageId, requester, updateDto);
    }
}
```

RegistrationController.java

```
package com.ntu.messenger.api.controller;

import com.ntu.messenger.api.service.UserService;
import com.ntu.messenger.data.dto.user.UserCreateDto;
import lombok.RequiredArgsConstructor;
import org.springframework.http.MediaType;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import javax.validation.Valid;

@RequiredArgsConstructor
@RestController
@RequestMapping(path = "/sign-up")
public class RegistrationController {

    private final UserService userService;

    @PostMapping(consumes = MediaType.APPLICATION_JSON_VALUE, produces =
MediaType.APPLICATION_JSON_VALUE)
    public ResponseEntity<String> signUp(@RequestBody @Valid UserCreateDto user) {
        return userService.signUpUser(user) ? ResponseEntity.status(201).body("Created") :
ResponseEntity.status(400).body("Rejected");
    }
}
```

SecurityController.java

```
package com.ntu.messenger.api.controller;

import com.ntu.messenger.security.user.MessengerUserDetails;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.SecurityContextHolder;

public abstract class SecurityController {

    protected MessengerUserDetails getUserDetails() {
        Authentication auth = SecurityContextHolder.getContext().getAuthentication();
```

```
        return (MessengerUserDetails) auth.getPrincipal();
    }
}
```

UserController.java

```
package com.ntu.messenger.api.controller;

import com.ntu.messenger.api.service.UserService;
import com.ntu.messenger.config.SwaggerConfig;
import com.ntu.messenger.data.converter.UserMapper;
import com.ntu.messenger.data.dto.user.UserCreateDto;
import com.ntu.messenger.data.dto.user.UserDto;
import com.ntu.messenger.data.dto.user.PasswordUpdateDto;
import com.ntu.messenger.data.model.User;
import io.swagger.v3.oas.annotations.Operation;
import io.swagger.v3.oas.annotations.security.SecurityRequirement;
import lombok.RequiredArgsConstructor;
import org.springframework.http.HttpStatus;
import org.springframework.http.MediaType;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping(path = "api/v1/user")
@RequiredArgsConstructor
public class UserController extends SecurityController {

    private final UserService userService;

    @Operation(security = {@SecurityRequirement(name = SwaggerConfig.BEARER_KEY)})
    @GetMapping(produces = MediaType.APPLICATION_JSON_VALUE)
    public ResponseEntity<UserDto> getCurrentUser() {
        User current = userService.findUserById(getUserDetails().getId());
        UserDto dto = UserMapper.MAPPER.map(current);
        return new ResponseEntity<>(dto, HttpStatus.OK);
    }

    @Operation(security = {@SecurityRequirement(name = SwaggerConfig.BEARER_KEY)})
    @PutMapping(consumes = MediaType.APPLICATION_JSON_VALUE, produces =
MediaType.APPLICATION_JSON_VALUE)
    public ResponseEntity<UserDto> updateUser(@RequestBody UserCreateDto updateDto) {
        User current = userService.findUserById(getUserDetails().getId());
        UserDto dto = UserMapper.MAPPER.map(userService.updateUser(current, updateDto));
        return new ResponseEntity<>(dto, HttpStatus.ACCEPTED);
    }

    @Operation(security = {@SecurityRequirement(name = SwaggerConfig.BEARER_KEY)})
    @PutMapping(path = "password", consumes = MediaType.APPLICATION_JSON_VALUE)
    @ResponseStatus(HttpStatus.ACCEPTED)
    public void updatePassword(@RequestBody PasswordUpdateDto updateDto) {
        User current = userService.findUserById(getUserDetails().getId());
        userService.updatePassword(current, updateDto.getPassword());
    }

    @Operation(security = {@SecurityRequirement(name = SwaggerConfig.BEARER_KEY)})
    @GetMapping(path = "search/username/{username}", produces = MediaType.APPLICATION_JSON_VALUE)
    public ResponseEntity<UserDto> searchUserByUsername(@PathVariable("username") String username) {
        return userService.findUserByUsername(username)
                .map(user -> new ResponseEntity<>(UserMapper.MAPPER.map(user), HttpStatus.OK))
                .orElseGet(() -> new ResponseEntity<>(null, HttpStatus.NO_CONTENT));
    }
}
```

MessageCriteria.java

```
package com.ntu.messenger.api.criteria;

import lombok.Getter;
import lombok.Setter;
import javax.validation.constraints.Max;
import javax.validation.constraints.Min;

@Getter
@Setter
public class MessageCriteria {

   @Min(1)
   @Max(200)
   private Integer size = 50;

   @Min(0)
   private Integer page = 0;
}
```

PageCriteria.java

```
package com.ntu.messenger.api.criteria;

import lombok.Getter;
import lombok.Setter;
import javax.validation.constraints.Max;
import javax.validation.constraints.Min;

@Getter
@Setter
public class PageCriteria {

   @Min(1)
   @Max(200)
   private int size = 20;

   @Min(0)
   private int page = 0;
}
```

EntityNotFoundException.java

```
package com.ntu.messenger.api.exception;

public class EntityNotFoundException extends RuntimeException {

   public EntityNotFoundException() {
      super();
   }

   public EntityNotFoundException(String message) {
      super(message);
   }
}
```

ForbiddenAccessException.java

```
package com.ntu.messenger.api.exception;

public class ForbiddenAccessException extends RuntimeException {

   public ForbiddenAccessException() {
      super();
   }
```

```
    public ForbiddenAccessException(String message) {
        super(message);
    }
}
```

ChatService.java

```
package com.ntu.messenger.api.service;

import com.ntu.messenger.api.exception.EntityNotFoundException;
import com.ntu.messenger.api.exception.ForbiddenAccessException;
import com.ntu.messenger.data.model.Chat;
import com.ntu.messenger.data.model.User;
import com.ntu.messenger.data.repository.mongo.ChatRepository;
import com.ntu.messenger.data.repository.mongo.MessageRepository;
import lombok.RequiredArgsConstructor;
import org.springframework.data.mongodb.core.MongoTemplate;
import org.springframework.data.mongodb.core.query.Criteria;
import org.springframework.data.mongodb.core.query.Query;
import org.springframework.stereotype.Service;
import java.util.*;

@Service
@RequiredArgsConstructor
public class ChatService {

    private final ChatRepository chatRepository;
    private final MessageRepository messageRepository;
    private final MongoTemplate mongoTemplate;

    public Chat createChatBetween(List<String> chatParticipantsIds) {
        Chat chat = new Chat();
        chat.setParticipants(new HashSet<>(chatParticipantsIds));
        return chatRepository.save(chat);
    }

    public Optional<Chat> getPrivateChat(String participantOne, String participantTwo) {
        Chat chat = mongoTemplate.findOne(Query.query(Criteria.where("participants")
                                                .all(participantOne, participantTwo)),
                                Chat.class);
        return Optional.ofNullable(chat);
    }

    public List<Chat> getUserChats(User user) {
        return mongoTemplate.find(Query.query(Criteria.where("participants").is(user.getId())), Chat.class);
    }

    public Chat getChatById(String chatId) {
        return chatRepository.findById(chatId).orElseThrow(EntityNotFoundException::new);
    }

    public void removeChat(User requester, String chatId) {
        Chat chat = chatRepository.findById(chatId).orElseThrow(EntityNotFoundException::new);
        verifyRequesterHasChatAccess(requester, chat);
        messageRepository.deleteAllByChatId(chatId);
        chatRepository.delete(chat);
    }

    private void verifyRequesterHasChatAccess(User requester, Chat chat) {
        if (chat.getParticipants().contains(requester.getId())) {
            return;
        }
```

```
            throw new ForbiddenAccessException("Access denied");
        }
    }
```

ContactService.java

```
package com.ntu.messenger.api.service;

import com.ntu.messenger.api.exception.EntityNotFoundException;
import com.ntu.messenger.data.model.User;
import com.ntu.messenger.data.model.UserProjection;
import com.ntu.messenger.data.repository.mongo.UserRepository;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;
import java.util.List;
import java.util.Set;
import java.util.stream.Collectors;

@Service
@RequiredArgsConstructor
public class ContactService {

    private final UserRepository userRepository;

    public void addContact(User user, String id) {
        User contact = userRepository.findById(id).orElseThrow(EntityNotFoundException::new);
        if (!contact.equals(user)) {
            Set<String> contacts = user.getContacts();
            contacts.add(contact.getId());
            userRepository.save(user);
        }
    }

    public void removeContact(User user, String id) {
        User contact = userRepository.findById(id).orElseThrow(EntityNotFoundException::new);
        if (!contact.equals(user)) {
            Set<String> contacts = user.getContacts();
            contacts.remove(contact.getId());
            userRepository.save(user);
        }
    }

    public List<UserProjection> getUserContacts(User user, int limit, int page) {
        var contactsIds = user.getContacts()
                        .stream()
                        .skip(page * limit)
                        .limit(limit)
                        .collect(Collectors.toList());

        return userRepository.findAllByIdIn(contactsIds);
    }

    public Boolean isInContactList(User user, String contactId) {
        User contact = userRepository.findById(contactId).orElseThrow(EntityNotFoundException::new);
        return user.getContacts().contains(contact.getId());
    }
}
```

MessageService.java

```
package com.ntu.messenger.api.service;

import com.ntu.messenger.api.criteria.MessageCriteria;
import com.ntu.messenger.api.exception.EntityNotFoundException;
```

```
import com.ntu.messenger.api.exception.ForbiddenAccessException;
import com.ntu.messenger.data.converter.MessageMapper;
import com.ntu.messenger.data.dto.message.MessageDto;
import com.ntu.messenger.data.dto.message.PrivateMessageSendDto;
import com.ntu.messenger.data.dto.message.MessageUpdateDto;
import com.ntu.messenger.data.model.Chat;
import com.ntu.messenger.data.model.Message;
import com.ntu.messenger.data.model.User;
import com.ntu.messenger.data.repository.mongo.MessageRepository;
import lombok.RequiredArgsConstructor;
import org.springframework.data.domain.PageRequest;
import org.springframework.security.crypto.encrypt.TextEncryptor;
import org.springframework.stereotype.Service;
import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;

@Service
@RequiredArgsConstructor
public class MessageService {

    private final ChatService chatService;
    private final UserService userService;
    private final MessageRepository messageRepository;
    private final TextEncryptor textEncryptor;

    public Message savePrivateMessage(String senderId, PrivateMessageSendDto privateMessageSendDto) {
        var message = new Message();
        var chat = createPrivateChatIfNotExists(senderId, privateMessageSendDto.getRecipientId());

        message.setText(textEncryptor.encrypt(privateMessageSendDto.getText()));
        message.setSender(userService.findUserById(senderId));
        message.setChat(chat);

        return messageRepository.save(message);
    }

    public List<Message> getMessagesByChatId(String chatId, User requester, MessageCriteria messageCriteria) {
        verifyThatUserHasAccess(requester.getId(), chatId);
        Integer size = messageCriteria.getSize();
        Integer page = messageCriteria.getPage();
        return messageRepository.getMessagesByChatIdOrderByCreateDate(chatId, PageRequest.of(page,
size)).getContent();
    }

    public List<MessageDto> decryptMessages(List<Message> messages) {
        return messages.stream()
                .map(this::decryptMessage)
                .collect(Collectors.toList());
    }

    public MessageDto decryptMessage(Message message) {
        message.setText(textEncryptor.decrypt(message.getText()));
        MessageDto dto = MessageMapper.MAPPER.map(message);
        dto.setSenderName(message.getSender().getUsername());
        return dto;
    }

    public Message getChatLastMessage(String chatId, User requester) {
        verifyThatUserHasAccess(requester.getId(), chatId);
        return
messageRepository.findFirstByChatIdOrderByCreateDateDesc(chatId).orElseThrow(EntityNotFoundException::new);
```

```
    }

    public List<Message> getChatsLastMessages(User requester) {
        List<Chat> userChats = chatService.getUserChats(requester);
        return userChats.stream()
                .map(chat -> getChatLastMessage(chat.getId(), requester))
                .collect(Collectors.toList());
    }

    public void deleteMessage(String messageId, User requester) {
        Message msg = messageRepository.findById(messageId).orElseThrow(EntityNotFoundException::new);
        verifyUserCanModifyMessage(requester.getId(), msg);
        messageRepository.delete(msg);
    }

    public void modifyMessage(String messageId, User requester, MessageUpdateDto messageUpdateDto) {
        Message msg = messageRepository.findById(messageId).orElseThrow(EntityNotFoundException::new);
        verifyUserCanModifyMessage(requester.getId(), msg);
        msg.setText(textEncryptor.encrypt(messageUpdateDto.getText()));
        messageRepository.save(msg);
    }

    private void verifyThatUserHasAccess(String participantId, String chatId) {
        Chat chat = chatService.getChatById(chatId);
        if (chat.getParticipants().contains(participantId)) {
            return;
        }
        throw new ForbiddenAccessException("Access denied");
    }

    private void verifyUserCanModifyMessage(String senderId, Message message) {
        if (message.getSender().equals(userService.findUserById(senderId))) {
            return;
        }
        throw new ForbiddenAccessException("Access denied");
    }

    private Chat createPrivateChatIfNotExists(String senderId, String recipientId) {
        return chatService.getPrivateChat(senderId, recipientId)
                    .orElseGet(() -> chatService.createChatBetween(Arrays.asList(recipientId, senderId)));
    }
}
```

UserService.java

```
package com.ntu.messenger.api.service;

import com.ntu.messenger.api.exception.EntityNotFoundException;
import com.ntu.messenger.data.converter.UserMapper;
import com.ntu.messenger.data.dto.user.UserCreateDto;
import com.ntu.messenger.data.model.User;
import com.ntu.messenger.data.repository.mongo.UserRepository;
import lombok.RequiredArgsConstructor;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Service;
import java.util.Optional;


@Service
@RequiredArgsConstructor
public class UserService {

    private final UserRepository userRepository;
```

```
    private final PasswordEncoder passwordEncoder;

    public User findUserById(String id) {
        return userRepository.findById(id).orElseThrow(EntityNotFoundException::new);
    }

    public Optional<User> findUserByUsername(String username) {
        return userRepository.findByUsername(username);
    }

    public Boolean signUpUser(UserCreateDto userCreateDto) {
        if (!userRepository.existsByUsernameOrEmailOrPhoneNumber(userCreateDto.getUsername(),
                                                userCreateDto.getEmail(),
                                                userCreateDto.getPhoneNumber())) {
            return createNewUser(userCreateDto);
        }
        return false;
    }

    public User updateUser(User user, UserCreateDto dto) {
        UserMapper.MAPPER.updateFromDto(dto, user);
        userRepository.save(user);
        return userRepository.save(user);
    }

    public void updatePassword(User user, String newPassword) {
        user.setPassword(passwordEncoder.encode(newPassword));
        userRepository.save(user);
    }

    private Boolean createNewUser(UserCreateDto dto) {
        User user = UserMapper.MAPPER.map(dto);
        user.setPassword(passwordEncoder.encode(dto.getPassword()));
        user.setEnabled(true);
        userRepository.save(user);
        return true;
    }
}
```

ChatService.java

```
package com.ntu.messenger.api.service;

import com.ntu.messenger.api.exception.EntityNotFoundException;
import com.ntu.messenger.api.exception.ForbiddenAccessException;
import com.ntu.messenger.data.model.Chat;
import com.ntu.messenger.data.model.User;
import com.ntu.messenger.data.repository.mongo.ChatRepository;
import com.ntu.messenger.data.repository.mongo.MessageRepository;
import lombok.RequiredArgsConstructor;
import org.springframework.data.mongodb.core.MongoTemplate;
import org.springframework.data.mongodb.core.query.Criteria;
import org.springframework.data.mongodb.core.query.Query;
import org.springframework.stereotype.Service;
import java.util.*;

@Service
@RequiredArgsConstructor
public class ChatService {

    private final ChatRepository chatRepository;
    private final MessageRepository messageRepository;
    private final MongoTemplate mongoTemplate;
```

```
    public Chat createChatBetween(List<String> chatParticipantsIds) {
        Chat chat = new Chat();
        chat.setParticipants(new HashSet<>(chatParticipantsIds));
        return chatRepository.save(chat);
    }

    public Optional<Chat> getPrivateChat(String participantOne, String participantTwo) {
        Chat chat = mongoTemplate.findOne(Query.query(Criteria.where("participants")
                                                .all(participantOne, participantTwo)),
                                    Chat.class);
        return Optional.ofNullable(chat);
    }

    public List<Chat> getUserChats(User user) {
        return mongoTemplate.find(Query.query(Criteria.where("participants").is(user.getId())), Chat.class);
    }

    public Chat getChatById(String chatId) {
        return chatRepository.findById(chatId).orElseThrow(EntityNotFoundException::new);
    }

    public void removeChat(User requester, String chatId) {
        Chat chat = chatRepository.findById(chatId).orElseThrow(EntityNotFoundException::new);
        verifyRequesterHasChatAccess(requester, chat);
        messageRepository.deleteAllByChatId(chatId);
        chatRepository.delete(chat);
    }

    private void verifyRequesterHasChatAccess(User requester, Chat chat) {
        if (chat.getParticipants().contains(requester.getId())) {
            return;
        }
        throw new ForbiddenAccessException("Access denied");
    }
}
```

CorsConfiguration.java

```
package com.ntu.messenger.config;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.PropertySource;
import org.springframework.web.servlet.config.annotation.CorsRegistry;
import org.springframework.web.servlet.config.annotation.EnableWebMvc;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

@Configuration
@EnableWebMvc
@PropertySource("classpath:security.properties")
public class CorsConfiguration implements WebMvcConfigurer {

    @Value("${cors.allowed.host}")
    private String host;

    @Override
    public void addCorsMappings(CorsRegistry registry) {
        registry.addMapping("/**")
                .allowedOrigins(host)
                .allowedMethods("PUT", "DELETE", "GET", "POST", "PATCH", "OPTION")
                .allowedHeaders("*")
                .maxAge(7200)
```

```
            .allowCredentials(true);
    }


}
```

JwtConfiguration.java

```
package com.ntu.messenger.config;

import lombok.Getter;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.PropertySource;

@Configuration
@Getter
@PropertySource("classpath:security.properties")
public class JwtConfig {

    @Value("${jwt.access.ttl.minutes}")
    private Long jwtTtl;

    @Value("${jwt.access.secret}")
    private String jwtSecret;

}
```

MongoConfiguration.java

```
package com.ntu.messenger.config;

import org.springframework.context.annotation.Configuration;
import org.springframework.data.mongodb.config.EnableMongoAuditing;
import org.springframework.data.mongodb.repository.config.EnableMongoRepositories;

@Configuration
@EnableMongoAuditing
@EnableMongoRepositories(basePackages = "com.ntu.messenger.data.repository.mongo")
public class MongoConfiguration {
}
```

SecurityConfig.java

```
package com.ntu.messenger.config;

import com.ntu.messenger.data.repository.mongo.UserRepository;
import com.ntu.messenger.security.filter.CorsFilter;
import com.ntu.messenger.security.filter.JwtAuthorizationFilter;
import com.ntu.messenger.security.filter.UsernamePasswordFilter;
import com.ntu.messenger.security.jwt.JwtAuthenticationProvider;
import com.ntu.messenger.security.jwt.service.DefaultJwtTokenService;
import com.ntu.messenger.security.jwt.service.JwtTokenService;
import com.ntu.messenger.security.user.MessengerUserDetailsService;
import lombok.RequiredArgsConstructor;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.PropertySource;
import org.springframework.context.annotation.PropertySources;
import org.springframework.http.HttpStatus;
import org.springframework.security.authentication.AuthenticationProvider;
import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
```

```
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.encrypt.Encryptors;
import org.springframework.security.crypto.encrypt.TextEncryptor;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.web.authentication.HttpStatusEntryPoint;

@Configuration
@PropertySources(value = {@PropertySource("classpath:security.properties"),
@PropertySource("classpath:application.yaml")})
@RequiredArgsConstructor
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    private static final String SECRET = "You know nothing Jon Snow";
    private static final String SALT = "40932872adb21086743faf3db57a9eaf";
    private static final String API_DOCS = "/v3/api-docs/**";
    private static final String SWAGGER = "/swagger-ui/**";

    private final JwtConfig jwtConfig;
    private final UserRepository userRepository;

    @Value("${secured.api.path}")
    private String SECURE_API_PATH;

    @Value("${public.signup.path}")
    private String SIGN_UP_PATH;

    @Value("${cors.allowed.host}")
    private String CORS_ALLOWED_HOST;

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.cors().and().csrf().disable()
            .authorizeRequests().antMatchers(SIGN_UP_PATH, SWAGGER,
API_DOCS).permitAll().antMatchers(SECURE_API_PATH).authenticated()

            .and()

            .exceptionHandling().authenticationEntryPoint(new HttpStatusEntryPoint(HttpStatus.UNAUTHORIZED))

            .and()

            .addFilter(new UsernamePasswordFilter(authenticationManager(), jwtTokenService()))
            .addFilter(new JwtAuthorizationFilter(authenticationManager()))
            .addFilterBefore(simpleCorsFilter(), UsernamePasswordFilter.class)

            .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS);
    }

    @Bean
    public JwtTokenService jwtTokenService() {
        return new DefaultJwtTokenService(jwtConfig);
    }

    @Bean
    public AuthenticationProvider jwtAuthenticationProvider() {
        return new JwtAuthenticationProvider(jwtTokenService());
    }

    @Bean
    public UserDetailsService userDetailsService() {
```

```
        return new MessengerUserDetailsService(userRepository, jwtConfig);
    }

    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder(8);
    }

    @Bean
    public TextEncryptor textEncryptor() {
        return Encryptors.delux(SECRET, SALT);
    }

    @Override
    public void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth.userDetailsService(userDetailsService()).passwordEncoder(passwordEncoder());
        auth.authenticationProvider(jwtAuthenticationProvider());
    }

    @Bean
    public CorsFilter simpleCorsFilter() {
        CorsFilter corsFilter = new CorsFilter();
        corsFilter.setHost(CORS_ALLOWED_HOST);
        return corsFilter;
    }
}
```

SwaggerConfig.java

```
package com.ntu.messenger.config;

import io.swagger.v3.oas.models.Components;
import io.swagger.v3.oas.models.OpenAPI;
import io.swagger.v3.oas.models.security.SecurityScheme;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class SwaggerConfig {

    public static final String BEARER_KEY = "bearer-key";

    @Bean
    public OpenAPI securedOpenApi() {
        return new OpenAPI().components(new Components().addSecuritySchemes(BEARER_KEY, new
SecurityScheme().type(SecurityScheme.Type.HTTP)
                                                                    .scheme("bearer")
                                                                    .bearerFormat("JWT")));
    }
}
```

WebSocketConfig.java

```
package com.ntu.messenger.config;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.PropertySource;
import org.springframework.messaging.simp.config.MessageBrokerRegistry;
import org.springframework.web.servlet.config.annotation.EnableWebMvc;
import org.springframework.web.socket.config.annotation.EnableWebSocket;
import org.springframework.web.socket.config.annotation.EnableWebSocketMessageBroker;
import org.springframework.web.socket.config.annotation.StompEndpointRegistry;
import org.springframework.web.socket.config.annotation.WebSocketMessageBrokerConfigurer;
```

```
@Configuration
@EnableWebSocket
@EnableWebMvc
@EnableWebSocketMessageBroker
@PropertySource("classpath:security.properties")
public class WebSocketConfig implements WebSocketMessageBrokerConfigurer {

   @Value("${cors.allowed.host}")
   private String host;

   @Override
   public void configureMessageBroker(MessageBrokerRegistry registry) {
      registry.setCacheLimit(32768)
            .setApplicationDestinationPrefixes("/app")
            .enableSimpleBroker("/topic", "/queue");
   }
   @Override
   public void registerStompEndpoints(StompEndpointRegistry registry) {
      registry.addEndpoint("/ws")
            .setAllowedOrigins(host)
            .withSockJS();
   }
}
```

ChatMapper.java

```
package com.ntu.messenger.data.converter;

import com.ntu.messenger.data.dto.chat.ChatDto;
import com.ntu.messenger.data.model.Chat;
import org.mapstruct.Mapper;
import org.mapstruct.Mapping;
import org.mapstruct.factory.Mappers;
import java.util.List;

@Mapper
public interface ChatMapper {

   ChatMapper MAPPER =  Mappers.getMapper(ChatMapper.class);

   @Mapping(target = "modifyDate", source = "updateDate")
   @Mapping(target = "chatUsers", source = "participants")
   @Mapping(target = "chatId", source = "id")
   ChatDto map(Chat chat);

   List<ChatDto> map(List<Chat> chats);
}
```

MessageMapper.java

```
package com.ntu.messenger.data.converter;

import com.ntu.messenger.data.dto.message.LastMessageDto;
import com.ntu.messenger.data.dto.message.MessageDto;
import com.ntu.messenger.data.model.Message;
import org.mapstruct.AfterMapping;
import org.mapstruct.Mapper;
import org.mapstruct.Mapping;
import org.mapstruct.MappingTarget;
import org.mapstruct.factory.Mappers;

import java.util.List;

@Mapper
```

```
public interface MessageMapper {
    MessageMapper MAPPER = Mappers.getMapper(MessageMapper.class);

    @Mapping(target = "senderName", ignore = true)
    @Mapping(target = "sentAt", source = "createDate")
    @Mapping(target = "changedAt", source = "updateDate")
    MessageDto map(Message message);

    @AfterMapping
    default void fixDtoConvert(@MappingTarget MessageDto dto, Message message) {
        dto.setSentAt(message.getCreateDate());
        dto.setChangedAt(message.getUpdateDate());
        dto.setChatId(message.getChat().getId());
        dto.setSenderName(message.getSender().getUsername());
    }

    LastMessageDto mapToLastMessage(MessageDto dto);

    List<MessageDto> map(List<Message> messages);
}
```

UserMapper.java

```
package com.ntu.messenger.data.converter;

import com.ntu.messenger.data.dto.user.UserCreateDto;
import com.ntu.messenger.data.dto.user.UserDto;
import com.ntu.messenger.data.model.User;
import org.mapstruct.Mapper;
import org.mapstruct.Mapping;
import org.mapstruct.MappingTarget;
import org.mapstruct.NullValuePropertyMappingStrategy;
import org.mapstruct.factory.Mappers;

import java.util.List;
import java.util.Set;

@Mapper(nullValuePropertyMappingStrategy = NullValuePropertyMappingStrategy.IGNORE)
public interface UserMapper {

    UserMapper MAPPER = Mappers.getMapper(UserMapper.class);

    @Mapping(target = "password", ignore = true)
    User map(UserCreateDto userCreateDto);

    UserDto map(User user);

    List<UserDto> map(Set<User> users);

    @Mapping(target = "id", ignore = true)
    @Mapping(target = "password", ignore = true)
    void updateFromDto(UserCreateDto updateDto, @MappingTarget User user);
}
```

ChatDto.java

```
package com.ntu.messenger.data.dto.chat;

import lombok.Getter;
import lombok.Setter;

import java.time.LocalDateTime;
import java.util.List;
```

```
@Getter
@Setter
public class ChatDto {
   private String chatId;
   private List<String> chatUsers;
   private LocalDateTime modifyDate;
}
```

LastMessageDto.java

```
package com.ntu.messenger.data.dto.message;

import lombok.*;

import java.time.LocalDateTime;

@Data
@AllArgsConstructor
@NoArgsConstructor
public class LastMessageDto {
   private String id;
   private String senderName;
   private String text;
   private String chatId;
   private LocalDateTime sentAt;
   private LocalDateTime changedAt;
}
```

MessageDto.java

```
package com.ntu.messenger.data.dto.message;

import lombok.Getter;
import lombok.Setter;
import java.time.LocalDateTime;

@Getter
@Setter
public class MessageDto {
   private String id;
   private String chatId;
   private String text;
   private String senderName;
   private LocalDateTime sentAt;
   private LocalDateTime changedAt;
}
```

MessageUpdateDto.java

```
package com.ntu.messenger.data.dto.message;

import lombok.Getter;
import lombok.Setter;

@Getter
@Setter
public class MessageUpdateDto {
   private String text;
}
```

PrivateMessageSendDto.java

```
package com.ntu.messenger.data.dto.message;

import lombok.AllArgsConstructor;
import lombok.Getter;
```

```
import lombok.NoArgsConstructor;
import lombok.Setter;
import javax.validation.constraints.NotBlank;
import javax.validation.constraints.NotNull;

@Getter
@Setter
@AllArgsConstructor
@NoArgsConstructor
public class PrivateMessageSendDto {
    @NotBlank
    private String text;

    @NotNull
    private String recipientId;
}
```

PasswordUpdateDto.java

```
package com.ntu.messenger.data.dto.user;

import lombok.Data;

@Data
public class PasswordUpdateDto {
    String password;
}
```

UserCreateDto.java

```
package com.ntu.messenger.data.dto.user;

import lombok.Getter;
import lombok.Setter;
import javax.validation.constraints.Email;
import javax.validation.constraints.NotBlank;

@Getter
@Setter
public class UserCreateDto {

    @NotBlank
    private String username;

    @NotBlank
    private String firstname;

    private String lastname;

    @NotBlank
    @Email
    private String email;

    @NotBlank
    private String password;

    @NotBlank
    private String phoneNumber;
}
```

UserDto.java

```
package com.ntu.messenger.data.dto.user;

import lombok.Getter;
```

```
import lombok.Setter;

@Getter
@Setter
public class UserDto {
   private String id;
   private String username;
   private String email;
   private String lastname;
   private String firstname;
   private String phoneNumber;
}
```

BasicModel.java

```
package com.ntu.messenger.data.model;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import org.springframework.data.annotation.CreatedDate;
import org.springframework.data.annotation.LastModifiedDate;
import java.time.LocalDateTime;

@Data
@AllArgsConstructor
@NoArgsConstructor
public abstract class BasicModel {

   @CreatedDate
   protected LocalDateTime createDate;

   @LastModifiedDate
   protected LocalDateTime updateDate;
}
```

Chat.java

```
package com.ntu.messenger.data.model;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.EqualsAndHashCode;
import lombok.NoArgsConstructor;
import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.Document;
import java.util.Set;

@EqualsAndHashCode(callSuper = false)
@Data
@AllArgsConstructor
@NoArgsConstructor
@Document
public class Chat extends BasicModel {

   @Id
   private String id;
   private Set<String> participants;
}
```

Message.java

```
package com.ntu.messenger.data.model;

import lombok.*;
```

```
import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.DBRef;
import org.springframework.data.mongodb.core.mapping.Document;


@Data
@EqualsAndHashCode(callSuper = false)
@AllArgsConstructor
@NoArgsConstructor
@Document
public class Message extends BasicModel {

    @Id
    private String id;

    @DBRef(db = "messenger")
    private User sender;

    @DBRef(db = "messenger")
    private Chat chat;

    private String text;
}
```

User.java

```
package com.ntu.messenger.data.model;

import lombok.*;
import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.index.Indexed;
import org.springframework.data.mongodb.core.mapping.Document;
import java.util.HashSet;
import java.util.Set;


@EqualsAndHashCode(callSuper = false)
@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
@Document
public class User extends BasicModel {

    @Id
    private String id;

    @Indexed(unique = true)
    private String username;

    @Indexed(unique = true)
    private String email;

    @Indexed(unique = true)
    private String phoneNumber;

    private String firstname;
    private String lastname;
    private boolean enabled;
    private String password;

    private Set<String> contacts = new HashSet<>();
}
```

UserProjection.java

```
package com.ntu.messenger.data.model;

public interface UserProjection {
   String getId();
   String getEmail();
   String getUsername();
   String getFirstname();
   String getLastname();
}
```

ChatRepository.java

```
package com.ntu.messenger.data.repository.mongo;

import com.ntu.messenger.data.model.Chat;
import org.springframework.data.mongodb.repository.MongoRepository;

public interface ChatRepository extends MongoRepository<Chat, String> {
}
```

MessageRepository.java

```
package com.ntu.messenger.data.repository.mongo;

import com.ntu.messenger.data.model.Message;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.data.mongodb.repository.MongoRepository;
import java.util.Optional;


public interface MessageRepository extends MongoRepository<Message, String> {

   Page<Message> getMessagesByChatIdOrderByCreateDate(String chatId, Pageable page);
   Optional<Message> findFirstByChatIdOrderByCreateDateDesc(String chatId);
    void deleteAllByChatId(String chatId);
}
```

UserRepository.java

```
package com.ntu.messenger.data.repository.mongo;

import com.ntu.messenger.data.model.User;
import com.ntu.messenger.data.model.UserProjection;
import org.springframework.data.mongodb.repository.MongoRepository;
import java.util.List;
import java.util.Optional;

public interface UserRepository extends MongoRepository<User, String> {
   Optional<User> findByUsername(String username);
   Boolean existsByUsernameOrEmailOrPhoneNumber(String username, String email, String phoneNumber);
   List<UserProjection> findAllByIdIn(List<String> ids);
   List<User> findByIdIn(List<String> ids);
}
```

CorsFilter.java

```
package com.ntu.messenger.security.filter;

import org.springframework.web.filter.OncePerRequestFilter;

import javax.servlet.FilterChain;
import javax.servlet.ServletException;
```

```
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

public class CorsFilter extends OncePerRequestFilter {

   private String host;

   @Override
   protected void doFilterInternal(HttpServletRequest httpServletRequest, HttpServletResponse httpServletResponse,
FilterChain filterChain) throws ServletException, IOException {
      httpServletResponse.setHeader("Access-Control-Allow-Origin", host);
      httpServletResponse.setHeader("Access-Control-Allow-Credentials", "true");
      httpServletResponse.setHeader("Access-Control-Allow-Methods", "PUT, POST, GET, OPTIONS, DELETE,
PATCH");
      httpServletResponse.setHeader("Access-Control-Max-Age", "7200");
      httpServletResponse.setHeader("Access-Control-Allow-Headers", "content-type, authorization");

      if (httpServletRequest.getMethod().equals("OPTIONS")) {
         httpServletResponse.setStatus(HttpServletResponse.SC_OK);
      } else {
         filterChain.doFilter(httpServletRequest, httpServletResponse);
      }
   }

   public void setHost(String host) {
      this.host = host;
   }
}
```

JwtAuthorizationFilter.java

```
package com.ntu.messenger.security.filter;

import com.ntu.messenger.security.jwt.JwtTokenBasedAuthentication;
import io.jsonwebtoken.ExpiredJwtException;
import lombok.extern.slf4j.Slf4j;
import org.springframework.http.HttpHeaders;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.AuthenticationException;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.web.authentication.www.BasicAuthenticationFilter;
import javax.servlet.FilterChain;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

@Slf4j
public class JwtAuthorizationFilter extends BasicAuthenticationFilter {

   public JwtAuthorizationFilter(AuthenticationManager authenticationManager) {
      super(authenticationManager);
   }

   private void unsuccessfulAuthentication(HttpServletRequest request, HttpServletResponse response, Exception
failed) {
      if (failed instanceof ExpiredJwtException) {
         response.addHeader("JWT-Expired", "true");
         Throwable cause = failed.getCause();
         if (cause instanceof ExpiredJwtException) {
            response.addHeader("JWT-Expired-Details", cause.getMessage());
```

```
            }
        }
        response.setStatus(HttpServletResponse.SC_UNAUTHORIZED);
    }

    @Override
    protected void doFilterInternal(HttpServletRequest req,
                                    HttpServletResponse res,
                                    FilterChain chain) throws IOException, ServletException {
        String header = req.getHeader(HttpHeaders.AUTHORIZATION);

        if (header == null || !header.startsWith("Bearer ")) {
            chain.doFilter(req, res);
            return;
        }

        try {
            Authentication authentication = authenticate(header);
            SecurityContextHolder.getContext().setAuthentication(authentication);
            chain.doFilter(req, res);
        } catch (AuthenticationException failed) {
            log.warn("Unsuccessful authentication attempt");
            unsuccessfulAuthentication(req, res, failed);
        }
    }

    private Authentication authenticate(String authorizationHeader) {
        String token = getBearerToken(authorizationHeader);
        return getAuthenticationManager().authenticate(new JwtTokenBasedAuthentication(token));
    }

    private String getBearerToken(String headerValue) {
        String[] authTypeValue = headerValue.split(" ");

        if (authTypeValue.length != 2) {
            return null;
        }

        if (!"Bearer".equalsIgnoreCase(authTypeValue[0])) {
            return null;
        }

        return authTypeValue[1];
    }
}
```

UsernamePasswordFilter.java

```
package com.ntu.messenger.security.filter;

import com.ntu.messenger.security.jwt.dto.JwtResponseDto;
import com.ntu.messenger.security.jwt.service.JwtTokenService;
import com.ntu.messenger.security.user.MessengerUserDetails;
import org.springframework.http.HttpHeaders;
import org.springframework.http.HttpStatus;
import org.springframework.http.MediaType;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.core.Authentication;
import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;
import org.springframework.web.client.HttpClientErrorException;

import javax.servlet.FilterChain;
import javax.servlet.http.HttpServletRequest;
```

```
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

public class UsernamePasswordFilter extends UsernamePasswordAuthenticationFilter {

    private final JwtTokenService jwtTokenService;

    public UsernamePasswordFilter(AuthenticationManager authenticationManager, JwtTokenService jwtTokenService)
{
        super(authenticationManager);
        this.jwtTokenService = jwtTokenService;
    }

    @Override
    protected String obtainPassword(HttpServletRequest request) {
        String password = super.obtainPassword(request);
        if (password == null) {
            throw new HttpClientErrorException(HttpStatus.BAD_REQUEST, "Required parameter 'password' is not
present");
        }
        return password;
    }

    @Override
    protected String obtainUsername(HttpServletRequest request) {
        String login = super.obtainUsername(request);
        if (login == null) {
            throw new HttpClientErrorException(HttpStatus.BAD_REQUEST, "Required parameter 'username' is not
present");
        }
        return login;
    }

    @Override
    protected void successfulAuthentication(HttpServletRequest request, HttpServletResponse response, FilterChain
chain, Authentication authResult) throws IOException {
        MessengerUserDetails userDetails = (MessengerUserDetails) authResult.getPrincipal();
        JwtResponseDto responseDto = jwtTokenService.getToken(userDetails);

        response.addHeader(HttpHeaders.CONTENT_TYPE, MediaType.APPLICATION_JSON_VALUE);
        response.getWriter().print(responseDto);
        response.getWriter().close();
    }
}
```

JwtResponseDto.java

```
package com.ntu.messenger.security.jwt.dto;

import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;

@Getter
@Setter
@AllArgsConstructor
@NoArgsConstructor
public class JwtResponseDto {

    private String access_token;

    private Long expires_in;
```

```
    @Override
    public String toString() {
        return  "{\n" +
                "\t\"access_token\": \"" + access_token + "\",\n" +
                "\t\"expires_in\": " + expires_in + "\n" +
                "}";
    }
}
```

JwtAuthorizationProvider.java

```
package com.ntu.messenger.security.jwt;

import com.ntu.messenger.security.jwt.service.JwtTokenService;
import com.ntu.messenger.security.user.MessengerUserDetails;
import lombok.extern.slf4j.Slf4j;
import org.springframework.security.authentication.AccountStatusUserDetailsChecker;
import org.springframework.security.authentication.AuthenticationProvider;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.AuthenticationException;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsChecker;

@Slf4j
public class JwtAuthenticationProvider implements AuthenticationProvider {

    private final UserDetailsChecker userDetailsChecker = new AccountStatusUserDetailsChecker();

    private final JwtTokenService jwtTokenService;

    public JwtAuthenticationProvider(JwtTokenService jwtTokenService) {
        this.jwtTokenService = jwtTokenService;
    }

    @Override
    public Authentication authenticate(Authentication authentication) throws AuthenticationException {
        if (!supports(authentication.getClass())) {
            return null;
        }

        JwtTokenBasedAuthentication tokenAuthentication = (JwtTokenBasedAuthentication) authentication;
        MessengerUserDetails ud = (MessengerUserDetails) getUserDetails(tokenAuthentication);
        userDetailsChecker.check(ud);
        tokenAuthentication.setPrincipal(ud);
        tokenAuthentication.setAuthenticated(true);

        return tokenAuthentication;
    }

    private UserDetails getUserDetails(JwtTokenBasedAuthentication tokenAuthentication) {
        return jwtTokenService.getUserDetails(tokenAuthentication.getToken());
    }

    @Override
    public boolean supports(Class<?> aClass) {
        return JwtTokenBasedAuthentication.class.isAssignableFrom(aClass);
    }
}
```

JwtTokenBasedAuthorization.java

```
package com.ntu.messenger.security.jwt;
```

```
import com.ntu.messenger.security.user.MessengerUserDetails;
import lombok.Getter;
import lombok.Setter;
import org.springframework.security.authentication.AbstractAuthenticationToken;
import org.springframework.security.core.userdetails.UserDetails;
import java.util.HashSet;

@Getter
@Setter
public class JwtTokenBasedAuthentication extends AbstractAuthenticationToken {

    private MessengerUserDetails principal;
    private String token;

    public JwtTokenBasedAuthentication(String token) {
        super(new HashSet<>());
        this.token = token;
    }

    @Override
    public Object getCredentials() {
        return null;
    }

    @Override
    public UserDetails getPrincipal() {
        return principal;
    }
}
```

MessengerUserDetails.java

```
package com.ntu.messenger.security.user;

import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;
import lombok.experimental.SuperBuilder;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.userdetails.UserDetails;
import java.time.Instant;
import java.util.Collection;

@Getter
@Setter
@SuperBuilder
@AllArgsConstructor
@NoArgsConstructor
public class MessengerUserDetails implements UserDetails {

    private String id;
    private String username;
    private String password;
    private Boolean isEnabled;
    private Instant accessTokenExpiresAfter;
    private Collection<? extends GrantedAuthority> authorities;

    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
        return authorities;
    }
```

```
    @Override
    public String getPassword() {
        return password;
    }

    @Override
    public String getUsername() {
        return username;
    }

    @Override
    public boolean isAccountNonExpired() {
        return true;
    }

    @Override
    public boolean isAccountNonLocked() {
        return true;
    }

    @Override
    public boolean isCredentialsNonExpired() {
        return Instant.now().isBefore(this.accessTokenExpiresAfter);
    }

    @Override
    public boolean isEnabled() {
        return isEnabled;
    }
}
```

Roles.java

```
package com.ntu.messenger.security.user;
public enum Roles {
    ADMIN,
    USER
}
```

MessengerApplication.java

```
package com.ntu.messenger;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class MessengerApplication {

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

application.yaml

```
spring:
  data:
    mongodb:
      authentication-database: admin
      username: root
      password: root
      database: messenger
      host: localhost
      port: 27017
```

```
      auto-index-creation: true
cors:
  allowed:
    host: https://messenger-memesenger.herokuapp.com
logging:
  level:
    root: info
```

У даному додатку наведені усі файли програми, перелік файлів зі змістом знаходиться на електронному носієві.

ДОДАТОК Б

ВІДГУК
керівника економічного розділу
на кваліфікаційну роботу бакалавра
на тему:
«Розробка серверної частини соціального месенджера
на основі Java/Spring»
студента групи 122-18-2 Вознюка Костянтина Юрійовича

Керівник економічного розділу
доцент каф. ПЕП та ПУ, к.е.н **Л. В. Касьяненко**

ДОДАТОК В

ПЕРЕЛІК ФАЙЛІВ НА ДИСКУ

Ім’я файлу	**Опис**
Пояснювальні документи	
Диплом Вознюк.doc	Пояснювальна записка до кваліфікаційної роботи. Документ Word.
Диплом Вознюк.pdf	Пояснювальна записка до кваліфікаційної роботи в форматі PDF.
Програма	
Messenger.zip	Архів. Містить коди програми.
Презентація	
Презентація Вознюк.ppt	Презентація кваліфікаційної роботи.