Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра мультимедійних систем факультету інформатики

РОЗРОБКА ВЕБ-ЗАСТОСУНКУ НА БАЗІ ФРЕЙМВОРКУ SPRING BOOT

Текстова частина до курсової роботи
за спеціальністю „Інженерія програмного забезпечення” 121

Керівник курсової роботи
с.в. _Борозенний С.О.__
(прізвище та ініціали)
_____
(підпис)
“____” _____ 2025 р.

Виконав студент _____
_____Гібський В. І._____
(прізвище та ініціали)
“____” _____ 2025 р.

Міністерство освіти і науки України
Київ 2025

Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра мультимедійних систем факультету інформатики

ЗАТВЕРДЖУЮ
доцент _____
“____”_____2025 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ
на курсову роботу
студенту Гібському Владиславу Ігоровичу факультету інформатики 3-го курсу

Тема: Розробка веб-застосунку на базі фреймворку Spring Boot
Зміст ТЧ до курсової роботи:

Індивідуальне завдання
Вступ
Частина 1: Аналіз предметної області та постановка завдання
Частина 2: Теоретичні основи створення веб-застосунків із використанням Spring Boot
Частина 3: Розробка веб-застосунку
Висновки
Список літератури
Додатки (за необхідністю)

Дата видачі “__” _____ 2025 р.
Керівник _____ (підпис)
Завдання отримав _____ (підпис)

Календарний план виконання роботи:

№	Назва етапу	Термін виконання	Примітки
1.	Отримання завдання на курсову роботу.	22.10.2024	
2.	Вивчення літератури та джерел за темою роботи.	01.12.2024	
3.	Встановлення та налаштування необхідного програмного забезпечення.	15.12.2024	
4.	Розробка веб-застосунку на базі Spring Boot.	01.04.2025	
5.	Написання текстової частини курсової роботи.	20.04.2025	
6.	Підготовка доповіді до захисту.	05.05.2025	
7.	Захист курсової роботи.	12.05.2025	

Гібський В. І. _____

Борозенний О. С. _____

“_____” _____

Зміст

Перелік прийнятих скорочень

REST – Representational State Transfer

HTML – Hyper Text Markup Language

CSS – Cascading Style Sheets

SQL – Structured Query Language

HTTP – Hypertext Transfer Protocol

API – Application Programming Interface

JPA – Java Persistence API

ORM – Object-Relational Mapping

JWT – JSON Web Token

JSON – JavaScript Object Notation

MVC – Model-View-Controller

CRUD – Create, Read, Update, Delete

URL - Uniform Resource Locator

STOMP - Simple Text Oriented Messaging Protocol

Анотація

Курсова робота присвячена розробці веб-застосунку з використанням фреймворку Spring Boot. Темою застосунку було вибрано месенджер, який дозволяє в реальному часі переписуватися в групових чатах.

Під час розробки було реалізовано основні можливості сучасних веб-застосунків: REST-архітектуру для роботи з API, серверний рендеринг сторінок через шаблонізацію, обробку подій в реальному часі за допомогою веб-сокетів. У проєкті застосовано PostgreSQL для зберігання даних, контейнеризацію бази даних через Docker, а також технології HTML, CSS та JavaScript для створення користувацького інтерфейсу.

Ключові слова: веб-застосунок, месенджер, Spring Boot, JPA, REST API, WebSocket, Thymeleaf, Spring Security, PostgreSQL, HTML, CSS, JavaScript.

Вступ

У сучасному світі інформаційні технології розвиваються надзвичайно стрімко, і веб-застосунки відіграють ключову роль у багатьох сферах людської діяльності. Підвищені вимоги до швидкості роботи, а також безпеки, масштабованості та простоти використання веб-сервісів потребують сучасних технологічних рішень і відповідних фреймворків. Одним із таких фреймворків є Spring Boot, який дозволяє розробникам швидко створювати масштабовані, безпечні та прості в обслуговуванні веб-додатки.

На сьогодні існує велика кількість технологій для створення веб-застосунків, однак саме Spring Boot завдяки своїй гнучкості, широкому функціоналу та чисельній спільноті розробників залишається одним із найпопулярніших рішень. А розробка власного застосунку дозволить отримати глибше розуміння принципів побудови серверної логіки та клієнтського інтерфейсу, взаємодії з базою даних через ORM (Spring Data JPA), організації асинхронної комунікації через WebSocket, а також роботу механізмів автентифікації та авторизації користувачів (через JWT-токени).

Для реалізації курсового проєкту було обрано розробку месенджера - застосунку, що може об'єднати і продемонструвати найсильніші сторони Spring Boot у розробці веб-застосунків.

Метою курсової роботи є розробка повноцінного веб-застосунку-месенджера, що одночасно використовує REST-архітектуру, шаблонізацію веб-інтерфейсу, двосторонній обмін повідомленнями через веб-сокети та сучасні засоби безпеки.

Призначення курсової роботи полягає у набутті практичних навичок побудови сучасних веб-застосунків із використанням сучасних технологій на прикладі реального проєкту.

Текстова частина курсової роботи містить в собі 3 основні розділи:

Перший розділ присвячений аналізу предметної області та постановці завдання курсової роботи. У ньому розглянуто чому для практичної частини було обрано створення веб-застосунку саме для обміну повідомленнями, визначено основні вимоги до функціональності месенджера, а також сформульовано цілі та задачі проєкту.

Другий розділ містить теоретичні основи створення веб-застосунків із використанням Spring Boot. Розглянуто архітектурні підходи до розробки серверної частини застосунку, особливості побудови REST API та WebSocket-комунікацій, принципи безпечної аутентифікації користувачів за допомогою JWT, а також використання інших ключових компонентів Spring Boot (Spring Data JPA тощо).

Третій розділ описує безпосередню розробку веб-застосунку. Показано структуру застосунку, вибір технологій для клієнтської частини (HTML, CSS, JavaScript, Thymeleaf), організацію бази даних, а також продемонстровано можливості реалізованого месенджера.

Основна частина

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАВДАННЯ

1.1 Аналіз предметної області

Веб-застосунки є важливою складовою сучасного програмного забезпечення. Вони широко застосовуються в найрізноманітніших сферах: розваги, спілкування, банкінг, онлайн-шопінг, медицина, освіта, торгівля тощо. Основною перевагою таких систем є надзвичайно легка доступність: вони працюють на будь-якому пристрої з підключенням до інтернету, незалежно від операційної системи, без необхідності скачувати додаткове програмне забезпечення. Завдяки централізованим оновленням і хмарним технологіям, такі рішення легко масштабуються, є безпечними, економічно вигідними в розгортанні, та забезпечують стабільний користувацький досвід [1].

Фреймворк Spring Boot на основі мови Java є дуже потужним інструментом для створення веб-застосунків. Завдяки автоматизованій конфігурації, вбудованим веб-серверам, підтримці REST-архітектури, та гнучкій роботі з різними джерелами даних, Spring значно зменшує кількість шаблонного коду й дозволяє зосередитись безпосередньо на логіці застосунку. Spring Boot значно спрощує процес розробки і дозволяє швидко створювати як невеликі сервери, так і масштабовані багаторівневі системи [2].

Одним із найпопулярніших напрямів сучасної веб-розробки є створення систем обміну повідомленнями (месенджерів), які вже стали невід’ємною частиною повсякденної комунікації. Такі сервіси, як

Telegram, WhatsApp, Viber, здобули широку популярність завдяки своїй зручності, швидкодії, функціональності та доступності як через мобільні додатки, так і через веб-браузери.

Месенджер як клас програмних продуктів має низку ключових функціональних характеристик, таких як обмін повідомленнями в реальному часі, підтримка багатокористувацького режиму, автентифікація та авторизація користувачів, зберігання історії переписки.

З огляду на це, для практичної частини курсової роботи було обрано реалізацію месенджера, що дозволяє продемонструвати та протестувати такі важливі компоненти веб-застосунків, як REST-контролери, WebSocket-з’єднання, шаблонізація сторінок, механізми безпеки, робота з базою даних.

Таким чином, обране прикладне завдання надало можливість комплексно реалізувати функціональність сучасного веб-застосунку та на практиці продемонструвати переваги використання Spring Boot як потужного фреймворку для розробки веб-систем.

1.2 Постановка завдання

У межах курсової роботи поставлено завдання розробити веб-застосунок у вигляді месенджера на мові Java на базі фреймворку Spring Boot. Система повинна забезпечувати базову функціональність сучасного сервісу обміну повідомленнями, включаючи автентифікацію користувачів, створення та перегляд чатів, надсилання й отримання повідомлень у реальному часі, а також збереження історії повідомлень.

Для реалізації миттєвої передачі повідомлень між користувачами використовується протокол WebSocket. Обробка REST-запитів здійснюється за допомогою контролерів Spring. Для забезпечення захисту

доступу до ресурсу має працювати система реєстрації, автентифікації та авторизації користувачів із використанням Spring Security та JWT-токенів. Усі дані про користувачі, повідомлення і чати мають зберігаються в реляційній базі даних PostgreSQL, взаємодія з якою організована за допомогою Spring Data JPA. Для спрощення роботи з базою даних, вона розгортається у Docker-контейнері.

Клієнтська частина реалізована з використанням HTML, CSS, JavaScript та шаблонізатора Thymeleaf. Інтерфейс дозволяє користувачам реєструватися, авторизуватися, переглядати свої чати, створювати нові та надсилати повідомлення (а також видаляти їх). Користувач також повинен мати можливість покинути чат, або редагувати чи видалити його (якщо він являється власником даного чату), або ж передати ці повноваження іншому користувачу. Також повинна бути сторінка для налаштувань, де користувач зможе переглянути і поміняти інформацію про себе, змінити пароль, вийти зі свого акаунту, чи назавжди його видалити.

Тож, курсова робота спрямована на комплексне застосування теоретичних знань та практичних навичок розробки веб-застосунків. Реалізація месенджера дозволяє охопити більшість важливих аспектів сучасної веб-розробки: від проєктування REST API до створення інтерактивного інтерфейсу та організації безпечної взаємодії між клієнтом і сервером. Розробка месенджера дозволяє продемонструвати широкий спектр можливостей фреймворку Spring Boot та супутніх технологій у створенні сучасних веб-застосунків.

РОЗДІЛ 2. ТЕОРЕТИЧНІ ОСНОВИ СТВОРЕННЯ ВЕБ-ЗАСТОСУНКІВ ІЗ ВИКОРИСТАННЯМ SPRING BOOT

2.1 Архітектура веб-застосунків

Більшість сучасних веб-застосунків побудовані за принципом клієнт-серверної архітектури, де клієнт – це програма чи пристрій, який взаємодіє з сервером, відправляючи запити і отримуючи відповіді. Сервер, в свою чергу, обробляє запити клієнтів, виконує необхідні операції та відправляє результати назад. Це дозволяє організувати централізовану обробку даних і зберігання на сервері, тоді як клієнт зосереджений на взаємодії з користувачем. Така модель значно полегшує реалізацію масштабованості та гнучкості. Клієнт може бути будь-яким пристроєм, який має доступ до інтернету і підтримує веб-браузер або спеціалізовані програми [3].

Архітектура веб-застосунків зазвичай розділяється на дві основні частини: фронтенд і бекенд. Фронтенд – це частина веб-застосунку, з якою безпосередньо взаємодіє користувач. Це все, що користувач бачить на екрані: інтерфейс, кнопки, поля для вводу, таблиці, діаграми тощо. Для розробки фронтенду використовуються технології, як HTML, CSS, JavaScript, а також можуть використовуватися фреймворки React, Angular, або Vue.js. Бекенд – це серверна частина веб-застосунку, яка відповідає за обробку запитів від клієнта, взаємодію з базою даних і виконання бізнес-логіки. Бекенд зазвичай включає в себе сервер, базу даних, API для комунікації між фронтендом і бекендом, а також механізми безпеки, аутентифікації та авторизації [3].

Основним протоколом для обміну даними між клієнтом і сервером є HTTP, або його захищена версія HTTPS. Ці протоколи використовуються для здійснення запитів від клієнта до сервера та отримання відповідей, що включають HTML-сторінки, зображення, файли, JSON або інші формати даних.

Для побудови взаємодії між фронтендом і бекендом найчастіше використовується підхід REST. Це архітектурний стиль, що передбачає використання стандартних HTTP-методів для доступу до ресурсів, кожен з яких представлений у вигляді унікального ідентифікатора (URI). REST API дозволяє фронтенду ефективно взаємодіяти з сервером, отримуючи або змінюючи дані без необхідності перезавантаження сторінки. Кожен запит повинен містити всю необхідну інформацію для обробки, тобто сервер не зберігає стан клієнта між запитами [4].

Проте для задач у реальному часі, REST API має обмеження. У таких випадках використовується протокол WebSocket, який дозволяє встановити двостороннє з'єднання між клієнтом і сервером. Це дає можливість миттєво передавати дані клієнту без потреби надсилання запиту. У застосунках типу месенджерів WebSocket є критично важливим для забезпечення комунікації між користувачами в реальному часі.

Важливим є також підхід до організації архітектури самого веб-застосунку. Розділяють монолітну та мікросервісну архітектури. У моноліті усі компоненти застосунку об’єднані в одному кодовому проєкті. Це простіше для початкової реалізації або невеликих проєктів, але викликає проблеми при масштабуванні. Мікросервісна архітектура передбачає розділення системи на незалежні модулі (сервіси), кожен з яких реалізує окрему функцію і може розгортатися окремо. Такий підхід складніший у реалізації, але він значно краще підходить для великих розподілених систем.

2.2 Фреймворк Spring та Spring Boot

Фреймворк Spring є потужним інструментом для створення масштабованих, модульних і легко підтримуваних Java-застосунків. Метою створення Spring було спростити розробку, тестування та підтримку Java-застосунків завдяки зручному управлінню залежностями, гнучкій конфігурації та можливості чіткого розділення відповідальностей між компонентами системи. Одним із головних принципів, закладених у Spring, є інверсія керування (inversion of control, IoC), яка дозволяє передати створення та зв'язування об'єктів спеціальному контейнеру, а не здійснювати це вручну. Завдяки цьому підходу значно знижується зв’язаність компонентів і підвищується легкість тестування застосунку [5]. Ще однією важливою особливістю Spring є підтримка аспектно-орієнтованого програмування (aspect-oriented programming, AOP), що дозволяє відокремити такий функціонал як логування, обробка помилок, кешування чи безпека від основної бізнес-логіки. Це забезпечує більшу гнучкість і чистоту коду [6].

Фреймворк має модульну структуру, завдяки чому можна використовувати лише ті компоненти, які необхідні в конкретному проєкті. Серед найважливих модулів є Spring Core (основа фреймворку, що надає базову функціональність для роботи з IoC контейнером), Spring MVC (модуль, що реалізує архітектурну модель Model-View-Controller для створення веб-застосунків), Spring Data (модуль для зручної роботи з базами даних, інтегрується з ORM-фреймворками), Spring Security (модуль, що забезпечує засоби безпеки, автентифікації та авторизації).

Незважаючи на свою гнучкість, класичний Spring може вимагати багато конфігурацій, що уповільнює процес розробки. Для спрощення цієї задачі було створено надбудову Spring Boot, яка дозволяє значно

прискорити створення готових до розгортання веб-застосунків із мінімальною кількістю налаштувань. До основний його переваг належать автоматична конфігурація залежностей, вбудований веб-сервер, що дозволяє запускати застосунок без зовнішнього сервера, зручну структуру для організації REST-контролерів, сервісів, репозиторіїв. Також з особливостей Spring Boot є можливість створення стартового проєкту через веб-інструмент Spring Initializr, де після обирання необхідних залежностей, генерується базовий каркас застосунку [7].

Таким чином, Spring і Spring Boot у комплексі дозволяють розробникам зосередитися на бізнес-логіці застосунку, не витрачаючи надто багато часу на технічну конфігурацію. Це робить їх особливо привабливими для створення сучасних веб-застосунків з чистою архітектурою, зрозумілою логікою, високою гнучкістю, забезпечуючи масштабованість, безпеку й ефективну взаємодію клієнта та сервера.

2.3 Створення REST API за допомогою Spring Boot

Spring Boot надає зручні інструменти для реалізації REST API. Основною складовою є для цього є використання спеціальних контролерів, які відповідають за обробку HTTP-запитів і повернення відповідей у форматі JSON. Вони позначаються анотацією @RestController, яка об'єднує в собі @Controller та @ResponseBody, що автоматично перетворює відповіді методів у JSON. Обробка конкретних HTTP-методів виконується через анотації @GetMapping, @PostMapping, @PutMapping, @DeleteMapping тощо. Аргументи запитів (шляхові параметри, тіло запиту або query-параметри) передаються через анотації @PathVariable, @RequestBody, @RequestParam. Такий підхід дозволяє швидко визначити маршрут, прив’язати його до методу та отримати

необхідні параметри з URL або тіла запиту. Завдяки автоматичній серіалізації Java-об’єктів у JSON і навпаки, обмін даними відбувається без потреби в додатковому кодуванні чи декодуванні.

Реалізація REST API в Spring Boot базується на трирівневій структурі, яка забезпечує чітке розділення обов’язків у застосунку. Такий підхід забезпечує чисту архітектуру застосунку, що легко тестується й масштабується. До цих рівнів належать:

a) Контролер – рівень, що безпосередньо взаємодіє з клієнтом. Контролери приймають HTTP-запити, викликають відповідні методи сервісів і повертають клієнту відповідь. Контролер не містить бізнес-логіки – його завдання полягає лише в маршрутизації й передачі даних далі. Позначаються анотацією @Controller.
b) Сервісний рівень – відповідає за обробку бізнес-логіки застосунку. Тут реалізуються основні правила та умови роботи програми (наприклад, перевірка прав доступу, валідація даних, перетворення об’єктів, фільтрація). Сервіси позначаються анотацією @Service і часто використовують один або кілька репозиторіїв для доступу до даних. Такий рівень дає змогу повторно використовувати бізнес-логіку в різних частинах застосунку, незалежно від способу звернення.
c) Репозиторій – рівень доступу до даних, який забезпечує взаємодію з базою даних за допомогою Spring Data JPA. У Spring Boot це зазвичай інтерфейси, що розширюють JpaRepository або CrudRepository – вони автоматично надають базові методи для роботи з базою (пошук, збереження, оновлення, видалення). Репозиторії позначаються анотацією @Repository [8].

Spring Boot також надає дозволяє обробляти помилки через глобальні винятки (@ControllerAdvice) та власні повідомлення про

помилки, що дозволяє повертати зрозумілі й структуровані відповіді з кодом помилки, описом проблеми та можливими вирішеннями.

2.4 Робота з базами даних у Spring Boot

Однією з основних задач серверної частини будь-якого веб-застосунку є зберігання, читання та обробка даних. У Spring Boot робота з базами даних реалізується зручно та ефективно, завдяки через ORM з використанням Spring Data JPA.

Spring Data JPA є абстракцією над стандартом Java Persistence API (JPA) і забезпечує автоматичну реалізацію основних операцій з базою даних: збереження, створення, оновлення, пошук і видалення сутностей – без необхідності писати SQL-запити вручну. Для цього створюються інтерфейси, які розширюють JpaRepository або CrudRepository. Spring автоматично генерує необхідні реалізації методів, які прописані в інтерфейсі на основі їхніх назв. Наприклад, якщо потрібно знайти користувача за його електронною поштою, достатньо описати метод «Optional<User> findByEmail(String email);», а в реалізації буде автоматично згенерує SQL-запит типу «SELECT * FROM users WHERE email = ?». Цей підхід дозволяє швидко створювати запити без ручного SQL, прискорює розробку та робить код легким для читання. У разі потреби складнішої логіки завжди можна використовувати власні SQL-запити, передаючи їх в анотацію @Query над методом репозиторію [9].

Кожна сутність бази даних (таблиця) відображається у вигляді Java-класу, що позначається анотацією @Entity. Поля класу відповідають стовпцям таблиці, а @Id вказує на первинний ключ. Крім того, можна використовувати анотації @OneToOne, @OneToMany, @ManyToOne, @ManyToMany для визначення зв’язків між сутностями. Це дозволяє

досить просто будувати складні моделі даних, які відображають реальні взаємозв’язки.

Конфігурація з'єднання з базою даних у Spring Boot виконується через application.properties. Тут вказується URL бази, логін, пароль, драйвер.

Spring Boot може автоматично генерувати структуру таблиць у базі даних на основі описаних сутностей. Ця функціональність активується через параметри конфігурації spring.jpa.hibernate.ddl-auto. Для неї можна задавати значення create-drop, яке вказує, що при запуску застосунку база даних має заново перестворитися; update – дозволяється створювати нові таблиці, додавати поля або оновлювати їхню структуру відповідно до змін у класах-моделях; validate – перевіряє, чи структура сутностей в коді відповідає структурі реальних сутностей у базі даних, і не дає запуститися застосунку, якщо це не справджується; none – відключає дану функціональність [9].

Важливим аспектом роботи з базами даних є ведення історії змін структури таблиць. Для цього використовуються інструменти міграції баз даних, в основному це Flyway або Liquibase. Вони дозволяють створювати окремі SQL-скрипти зі змінами і гарантують узгодженість структури бази даних між різними середовищами: локальним, тестовим і продакшеном.

Крім того, Spring підтримує і надає простий функціонал для роботи з транзакціями – можливість об’єднувати кілька дій з базою в єдину логічну операцію. Транзакції дозволяють забезпечити цілісність даних, зокрема у випадках, коли під час виконання операції трапляється помилка. Анотація @Transactional дозволяє позначити метод або клас, у якому всі дії будуть виконані або повністю, або не виконані взагалі.

Тож, робота з базою даних у Spring Boot побудована на принципах зручності, безпеки й масштабованості. Завдяки використанню JPA, Spring Data, підтримці транзакцій та засобам міграції, розробник отримує повний

контроль над даними без необхідності працювати з SQL безпосередньо, при цьому маючи можливість у разі потреби розширити функціонал власними запитами.

2.5 Аутентифікація та авторизація у Spring Boot

Захист веб-застосунку є критично важливою складовою сучасної розробки. Spring Boot надає потужний і гнучкий інструментарій для реалізації аутентифікації та авторизації через модуль Spring Security, який інтегрується з іншими компонентами фреймворку та дозволяє контролювати доступ до ресурсів, обробляти логіку входу, виходу та перевірку прав користувачів.

Аутентифікація – це процес перевірки особи користувача, найчастіше через логін і пароль. Авторизація – процес перевірки прав доступу до певних функцій або ресурсів після підтвердження особи.

У застосунках, побудованих на Spring Boot, типова реалізація автентифікації здійснюється за допомогою JWT (JSON Web Token). Цей підхід дозволяє реалізувати аутентифікацію без збереження сесій на сервері. Після успішного логіну сервер генерує токен, у якому зашифрована інформація про користувача, і який клієнт зберігає й надсилає з кожним наступним запитом у заголовку Authorization.

У Spring Security процес аутентифікації та авторизації реалізований як ланцюжок фільтрів (Security Filter Chain), який перехоплює всі HTTP-запити ще до того, як вони потрапляють у контролери. Головним у цьому ланцюжку є фільтр UsernamePasswordAuthenticationFilter, який обробляє дані для логіну. Якщо проєкт використовує JWT, то додається власний фільтр (JwtAuthenticationFilter), який перевіряє наявність токена у заголовках запиту та розшифровує його й додає розшифровану

інформацію про користувача у Security Context, який зберігається в контексті поточного запиту [10].

У Spring Security можна задати ролі користувачів і прив’язати доступ до певних ендпоінтів тільки для користувачів з відповідними повноваженнями. Це можна реалізувати у конфігурації HttpSecurity, або у контролері за допомогою анотацій.

Паролі користувачів зберігаються в базі у зашифрованому вигляді. Для цього використовується PasswordEncoder, найчастіше BCryptPasswordEncoder, який забезпечує надійне хешування з сольовим захистом.

Spring Security дозволяє реалізувати як прості, так і складні сценарії безпеки з максимальною гнучкістю. Завдяки чіткому розділенню відповідальностей, можливості конфігурації та наявності багатьох готових рішень, цей модуль є стандартом для захисту веб-застосунків на базі Spring Boot.

2.6 Робота з WebSocket у Spring Boot

HTTP-протокол є однонаправленим і синхронним протоколом, тобто сервер відповідає тільки на надісланий клієнтом запит. Проте часто у веб-застосунках необхідна постійна двостороння підтримка передачі даних, наприклад у системах сповіщень, онлайн-іграх, чатах. Це є і завданням протоколу WebSocket.

Spring Boot надає зручні засоби для інтеграції WebSocket у застосунок через модуль spring-boot-starter-websocket, який завдяки STOMP (Simple Text Oriented Messaging Protocol) дозволяє зручно спрямовувати повідомлення.

Для реалізації такого зв’язку створюється клас, який імплементує інтерфейс WebSocketMessageBrokerConfigurer, де вказуються точки підключення, канал для підписки на повідомлення та їх обробки. На стороні клієнта WebSocket-з’єднання реалізується зазвичай за допомогою бібліотек SockJS і STOMP.js, які дозволяють підключатися до сервера, підписуватися на канали й надсилати повідомлення у зручному форматі [11].

Тож, завдяки легкій конфігурації, інтеграції з іншими компонентами застосунку, маршрутизації повідомлень через STOMP та підтримці великої кількості з’єднань одночасно, Spring Boot має зручні і потужні можливості для створення веб-застосунків, що працюють у реальному часі.

2.7 Шаблонізація у Spring Boot

Однією з важливих частин веб-застосунку є формування динамічного HTML-контенту, який буде відображатися користувачеві. Одним з найбільш поширених способів є використання шаблонізаторів, які забезпечують процес формування HTML-сторінок на сервері на основі заздалегідь підготовлених шаблонів. У фреймворку Spring Boot найпопулярнішим і одним з найкращих є шаблонізатор Thymeleaf, що дозволяє інтегрувати динамічні дані у HTML-сторінки без потреби у складному клієнтському рендерингу.

Thymeleaf – це сучасний серверний шаблонізатор, який особливо добре інтегрується зі Spring MVC, що дозволяє легко передавати дані з контролера у вигляді моделі на сторінку. Серед його головних переваг є те, що шаблони є повністю сумісними з HTML і їх можна відкривати у звичайному браузері без обробки на сервері. Він підтримує вставку

фрагментів, циклів, умовних конструкцій. Thymeleaf дозволяє зручно працювати з формами з прив'язкою до об'єктів та автоматичним відображенням і валідацією полів [12].

Ключовою особливістю Thymeleaf є використання спеціальних атрибутів у HTML-тегах, таких як th:text, th:each, th:if, th:href, th:value тощо. Вони дають змогу безпосередньо у HTML-розмітці виконувати динамічне заповнення сторінки: відображення значень змінних, циклічний вивід елементів, умовне відображення блоків або передавання параметрів у форми. Наприклад, атрибут th:text="${user.name}" дозволяє вставити всередину тегу значення імені користувача з моделі, що передається з контролера [12].

Thymeleaf також підтримує розділення шаблонів на фрагменти (th:replace, th:insert), що сприяє повторному використанню частин інтерфейсу. Це покращує організацію коду та полегшує підтримку проєкту. Крім того, шаблонізатор дозволяє працювати з локалізованими ресурсами, забезпечуючи багатомовність інтерфейсу.

У Spring Boot підтримка Thymeleaf вмикається автоматично, якщо в проєкті підключено залежність spring-boot-starter-thymeleaf. Шаблони розміщуються у папці resources/templates, а контролери повертають просто назви сторінок як стрічку.

РОЗДІЛ 3. РОЗРОБКА ВЕБ-ЗАСТОСУНКУ

3.1 Робота з базою даних

3.1.1 Структура бази даних

База даних веб-застосунку розроблена у відповідності до логіки месенджера між користувачами в рамках чатів. Вона є реляційною та складається з чотирьох основних таблиць: users, chats, messages та chat_users. Їх структура дозволяє забезпечити коректне зберігання користувацьких даних, організацію чатів, збереження історії повідомлень і підтримку взаємозв’язків між користувачами та чатами.

a) users – таблиця зберігає дані про зареєстрованих користувачів. Кожен запис містить id, ім’я користувача, телефон, імейл, хешований пароль, ім’я та опис. Імейл, телефон та ім’я користувача є унікальними.
b) chats – таблиця, яка описує окремі чати. Вона містить id чату, назву, id користувача-власника, назву та час останньої зміни в чаті.
c) messages – таблиця для зберігання повідомлень. Повідомлення прив’язані до конкретного чату та автора. Вони також мітять id, текст повідомлення, дату надсилання.
d) chat_users – таблиця зв’язків між користувачами та чатами. Це реалізація зв’язку багато-до-багатьох, коли один чат може мати багато учасників, і один користувач може брати участь у багатьох чатах.

3.1.2 Робота з базою через Spring Data JPA

Для взаємодії з реляційною базою даних у застосунку використовується підхід ORM на основі бібліотеки Spring Data JPA. Тож, у застосунку реалізовано чотири основні сутності: User, Chat, Message, ChatUser. Кожен із цих класів позначений анотацією @Entity, що вказує на його відповідність таблиці в базі даних, а також через @Table вказується ім’я таблиці. Ідентифікатори позначено через @Id та @GeneratedValue, а зв’язки між сутностями через @OneToMany, @ManyToOne, @JoinColumn, @ManyToMany та інші відповідні анотації.

```java
@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
@Entity
@Table(name = "message")
public class MessageEntity implements Identifiable<String> {

    @Id
    @Column(name = "id")
    @GeneratedValue(strategy = GenerationType.UUID)
    private String id;

    @Column(name = "content", nullable = false)
    private String content;

    @CreationTimestamp
    @Temporal(TemporalType.TIMESTAMP)
    @Column(name = "sent_at", nullable = false, updatable = false)
    @OrderColumn
    private LocalDateTime sentAt;

    @ManyToOne(optional = false)
    @JoinColumn(name = "chat_id", nullable = false, updatable = false)
    @ToString.Exclude
    @EqualsAndHashCode.Exclude
    private ChatEntity chat;

    @ManyToOne
    @JoinColumn(name = "sender_id", updatable = false)
    @ToString.Exclude
    @EqualsAndHashCode.Exclude
    private UserEntity sender;
}
```

Рисунок 3.1. Приклад створеної сутності MessageEntity

На рисунку 3.1 показано приклад реалізованої сутності MessageEntity, яка відповідає таблиці message у базі даних. За допомогою @GeneratedValue вказується, що id генерується автоматично у форматі uuid, а через анотацію @CreationTimestamp sentAt автоматично заповнюється часом створення повідомлення.

Також у проєкті широко використовується бібліотека Lombok – інструмент для автоматичної генерації повторюваного коду, такого як гетери, сетери, конструктори, методи toString(), equals() тощо. Це дозволяє значно скоротити обсяг коду сутностей та підвищити його читабельність, не впливаючи на логіку коду, оскільки всі анотації працюють лише під час компіляції [13].

У класі MessageEntity, як і в інших сутностях, використовуються такі анотації Lombok: @Data – генерує гетери, сетери, toString(), equals() та hashCode() для всіх полів; @Builder – дозволяє створювати об'єкти з допомогою патерну "будівельник", що зручно при тестуванні та ініціалізації; @NoArgsConstructor і @AllArgsConstructor – створюють конструктори без параметрів і з усіма полями відповідно. Щоб уникнути потенційних циклічних залежностей у методах toString() та equals(), у полях типу chat та sender додатково використовуються @ToString.Exclude та @EqualsAndHashCode.Exclude.

Для доступу до даних використовуються репозиторії, що розширюють інтерфейс JpaRepository. Spring Data JPA автоматично реалізує стандартні CRUD-операції на основі сигнатури методів. У застосунку створено такі репозиторії для кожної сутності.

```
public interface UserRepository extends BaseRepository<UserEntity, String> {

    1 usage  Vladyslav Hibskyi
    boolean existsByEmail(String email);

    1 usage  Vladyslav Hibskyi
    boolean existsByPhone(String phone);

    1 usage  Vladyslav Hibskyi
    boolean existsByUsername(String username);

    1 usage  Vladyslav Hibskyi
    Optional<UserEntity> findByEmail(String email);

    1 usage  Vladyslav Hibskyi
    Optional<UserEntity> findByPhone(String phone);

    2 usages  Vladyslav Hibskyi
    Optional<UserEntity> findByUsername(String username);
}
```

Рисунок 3.2. Приклад реалізації репозиторію UserRepository

3.1.3 Міграції бази даних

Для керування версіями структури бази даних застосовується Flyway. Його основна функція полягає в тому, щоб синхронізувати структуру бази з кодом проєкту та забезпечити автоматичне виконання змін у схемі під час кожного запуску застосунку. Flyway сканує спеціальну директорію в ресурсах проєкту (за замовчуванням resources/db/migration) у пошуках файлів із назвою за шаблоном V{номер}__{опис}.sql. Кожен файл виконується послідовно відповідно до своєї версії. Flyway зберігає інформацію про всі виконані міграції в окремій службовій таблиці flyway_schema_history [14].

3.1.4 Реалізація доступу до бази даних

Для зручності локального розгортання база даних PostgreSQL запускається у Docker-контейнері. Це дозволяє швидко запускати повністю налаштоване середовище, уникати конфліктів локальних СУБД, забезпечити однакове середовище для всієї команди. Сервіс бази даних описано у файлі docker-compose.yml, після запуску якого в докері база даних стає доступною для застосунку [15].

Для підключення до бази даних у Spring Boot використовується файл конфігурації application.properties, який розміщено в папці resources. У ньому задаються всі необхідні параметри для встановлення з’єднання з PostgreSQL та для роботи з JPA, Hibernate і Flyway. і Spring Boot автоматично підключається до бази даних відповідно до вказаних параметрів.

3.2 Реалізація бізнес-логіки

Бізнес-логіка застосунку реалізована відповідно до принципів багаторівневої архітектури, де кожен шар виконує чітко визначену функцію. Уся логіка отримання, збереження, оновлення, видалення чатів, користувачів чи повідомлень, перевірка дозволів, валідація, виклик маперів окремі сервісні класи, які реалізують відповідні інтерфейси.

У застосунку дані, які надходять або відправляються у відповідь, не передаються напряму через JPA-сутності. Для цього створено окремі класи: *View для вхідних даних, і *Response для вихідних. Об’єкти view використовуються для отрияння даних під час створення нової сутності чи її оновлені. Натомість об’єкти response застосовуються у REST API, де

дані передаються клієнту у вигляді JSON. Таке розділення дозволяє чітко контролювати, які поля користувач має заповнювати вручну, а які генеруються автоматично; та уникати витоку чутливої інформації, наприклад пароль користувача (хоч і захешований).

Для перетворення між entity і view/response-об’єктами застосовуються спеціальні класи-мапери (*Mapper). Вони реалізують інтерфейс BaseMapper<E, V, R>. Він вимагає імплементації методів mapToEntity(view) та mapToResponse(entity), а також merge(view, entity) для об’єднання даних з entity та view (при оновленні даних сутності). Мапери інкапсулюють усю логіку перетворення та дозволяють легко контролювати, що саме буде використано в контролерах або повернено у відповідь.

Однак, просто створити чи змінити об'єкт у системі недостатньо — перш ніж зберігати або навіть повертати дані, необхідно перевірити, чи має користувач на це право, чи об’єкт існує, і чи дані є коректними. У застосунку ця відповідальність покладена на окремий рівень валідаторів, які реалізують чіткі правила доступу та валідації для кожної сутності. Вони зберігаються в окремому пакеті validator і побудовані на базі спільного узагальненого класу BaseValidatorImpl, який забезпечує універсальні методи перевірки: validateForCreate, validateForUpdate, validateForView, validateForDelete. Валідатори перевіряють коректність вхідних даних, а також наявність дозволу у поточного користувача. Конкретні валідатори розширюють базову логіку: наприклад, ChatValidator перевіряє, чи всі запрошені до чату користувачі реально існують у системі.

Якщо перевірка у валідаторі не проходить, наприклад, користувач намагається змінити об'єкт, який йому не належить, або передає некоректні дані, у застосунку викидаються власні винятки. Вони розташовані у пакеті exception і всі наслідують BaseException, який має

статус-код, назву та повідомлення – список стрічок. Ці винятки перехоплюються у глобальних обробниках винятків (RestControllerExceptionHandler для API та PageControllerExceptionHandler для HTML), які формують відповідну відповідь користувачеві: або JSON з інформацією про помилку, або сторінку помилки.

Всі ці компоненти взаємодіють і об’єднуються всередині сервісів, де виконується основна логіка. Саме сервіс виступає як координатор: через валідатор він перевіряє коректність даних і, чи має користувач право діяти, за допомогою репозиторію дістає об’єкт з бази даних, викликає мапер для перетворення, і повертає результат. Кожній сутності відповідає свій сервісний інтерфейс *Service, для якого існує своя імплементація. Такий підхід створює чистий контракт: інтерфейс визначає, що має бути зроблено, а реалізація – як це зробити; а також дозволяє легко замінити реалізацію сервісу в коді, наприклад, при unit-тестуванні.

Усі сервіси реалізовують повний набір CRUD-операцій для своїх сутностей. Але оскільки логіка цих операцій для багатьох сутностей є ідентична і просто містить купу повторюваного коду, у застосунку було створено універсальний базовий сервіс BaseServiceImpl<E, V, R>, який реалізує спільну функціональність для всіх сутностей. Він взаємодіє з відповідним репозиторієм, валідатором і мапером і надає готову базову реалізацію CRUD-операцій для своїх нащадків. А імплементація конкретних сервісів просто розширюють BaseServiceImpl і додають власні специфічні дії, уникаючи дублювання загального коду.

Завдяки такій побудові з чітким розмежуванням інтерфейсів, реалізацій, повторно використовуваних маперів, валідаторів і базових сервісів застосунок отримав структуровану, розширювану та чисту бізнес-логіку. Кожен компонент має свою ділянку відповідальності, що не лише спрощує підтримку та тестування, а й дозволяє легко масштабувати функціональність без порушення вже наявної архітектури.

3.3 Система безпеки

Безпека веб-застосунку є критично важливою складовою, особливо коли мова йде про сервіси, де здійснюється передача персональних даних, таких як повідомлення, профілі користувачів та історія чатів. У даному застосунку реалізовано повноцінну систему автентифікації та авторизації на основі технології JWT. Уся логіка безпеки побудована у відповідності до принципів stateless-архітектури, де стан користувача не зберігається на сервері, а передається у вигляді токена на клієнті. Після успішного входу користувача в систему йому видається спеціальний токен, який підтверджує його ідентичність. JWT-токен зберігається у cookie.

Система безпеки реалізована у пакеті security, що містить декілька ключових компонентів. Першим з них є клас SecurityConfig, який виконує конфігурацію Spring Security. У цьому класі відключено CSRF-захист, оскільки застосунок не використовує сесій, та встановлено політику SessionCreationPolicy.STATELESS, що означає повну відсутність збереження стану на сервері. Також у цьому класі підключаються всі необхідні фільтри безпеки, визначаються дозволені публічні маршрути (наприклад, для логіну чи реєстрації) і зазначається, що решта маршрутів потребують автентифікації.

Клас JWTUtility виконує створення, підпис і розбір JWT-токенів. У токені міститься ідентифікатор користувача, термін його дії, а також сигнатура, яка дозволяє його перевірити. У випадку, якщо токен підроблений або строк його дії сплив, запит відхиляється.

Важливою частиною є система фільтрів, яка замінює стандартний механізм автентифікації Spring Security. Перший із них, AuthenticationFilter, обробляє логін користувача. Він зчитує дані з JSON-запиту (електронну пошту та пароль), перевіряє їх через спеціальний

менеджер автентифікації AuthenticationManagerImpl, і у разі успішної перевірки генерує JWT, який записується в cookie клієнта.

Другий фільтр JWTAuthorizationFilter відповідає за перевірку наявності токен. Він дістає токен з cookie, перевіряє його за допомогою JWTUtility, і якщо токен дійсний – додає дані з токена в SecurityContextHolder.

Окремо реалізовано AuthExceptionHandlerFilter, який перехоплює всі помилки, пов’язані з авторизацією. Наприклад, якщо токен прострочений, некоректний або взагалі відсутній, він поверне інформацію про помилку та її HTTP-статус. Такий підхід дозволяє централізовано обробляти всі типи помилок в одному місці. Важливо розуміти, що помилки під час проходження ланцюжка фільтрів не обробляються глобальними контролерами винятків у контролерах, оскільки запит на цей момент туди ще не доходить.

Ще одним важливим елементом є AuthenticationManagerImpl, який є реалізацією AuthenticationManager і який виконує перевірку даних користувача при логіні, використовуючи репозиторій і шифрування паролів через BCryptPasswordEncoder. Якщо користувача не знайдено або пароль неправильний, генерується відповідний виняток.

3.4 Веб-інтерфейс

Веб-інтерфейс у даному застосунку реалізовано на основі серверного рендерингу HTML-сторінок з використанням шаблонізатора Thymeleaf, а також JavaScript для реалізації динамічної поведінки клієнта. Основна ідея побудови інтерфейсу полягає у тому, що більшість контенту формується на сервері й передається вже у вигляді заповнених HTML-

сторінок, а окремі клієнтські дії обробляються за допомогою JavaScript через REST API.

Передача даних до сторінок здійснюється через web-контролери, які розміщено у пакеті controller.web. Вони повертають імена шаблонів, що розміщені у resources/templates, і передають до них об’єкти моделі, як-от поточного користувача, активний чат, список повідомлень тощо. Наприклад, ChatPageController отримує інформацію про активного користувача й обраний чат, додає їх до Model, після чого відображає сторінку чату з відповідними повідомленнями.

Основна HTML-структура зберігається у вигляді шаблонів у папці templates/pages, а повторювані елементи інтерфейсу (бічні панелі) винесено в окрему папку templates/components. Сторінки для логіну та реєстрації знаходяться у templates/auth, а також є спеціальна сторінка для красивого відображення помилки.

Наповнення сторінок здійснюється через атрибути Thymeleaf. Наприклад, в chat.html виводиться список повідомлень у чаті за допомогою th:each, виводиться вміст повідомлення через th:text, а класи повідомлень змінюються залежно від того, хто їх надіслав. Шаблон таким чином адаптується під контекст користувача.

Зовнішній вигляд інтерфейсу стилізовано за допомогою фреймворку Bootstrap, який забезпечує просту і адаптивну верстку, стилізовані кнопки, модальні вікна. Крім того, у папці resources/static містяться власні CSS- та JavaScript-файли, які відповідають за унікальне оформлення застосунку та поведінку інтерфейсу.

Ключову роль у забезпеченні динамічності інтерфейсу відіграє JavaScript, де обробляються події введення чи видалення повідомлень, відкриття контекстного меню чату, видалення або виходу з чату тощо. Для взаємодії з бекендом використовується API-запити через fetch, що

дозволяє частково оновлювати інтерфейс без повного перезавантаження сторінки.

Уся візуальна частина оформлена в мінімалістичному стилі. Головна сторінка містить список чатів ліворуч, а справа – вікно самої переписки та поле для введення і надсилання повідомлень внизу. Також реалізовано модальні вікна для керування чатом: перегляду учасників, видалення або виходу; та для перегляду інформації про користувача. Є сторінка налаштувань, яка містить вкладку для перегляду власного профілю, редагування своїх даних і налаштування облікового запису: вихід, видалення акаунту та зміна паролю.

Загалом, веб-інтерфейс застосунку реалізовано як гібридну модель, де серверна генерація забезпечує швидку початкову побудову інтерфейсу й зручну передачу даних, а JavaScript відповідає за динамічну поведінку на клієнті. Такий підхід дозволяє досягти хорошого балансу між продуктивністю, безпекою та гнучкістю у взаємодії з користувачем.

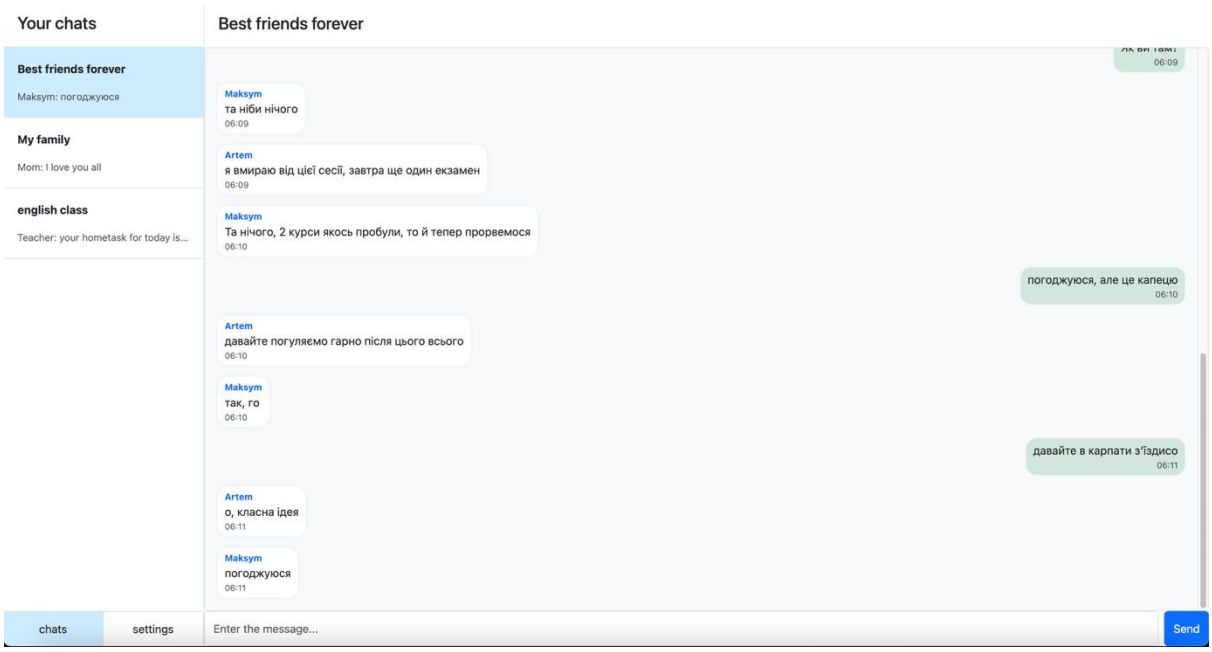

Рисунок 3.3. Вигляд головного вікна застосунку

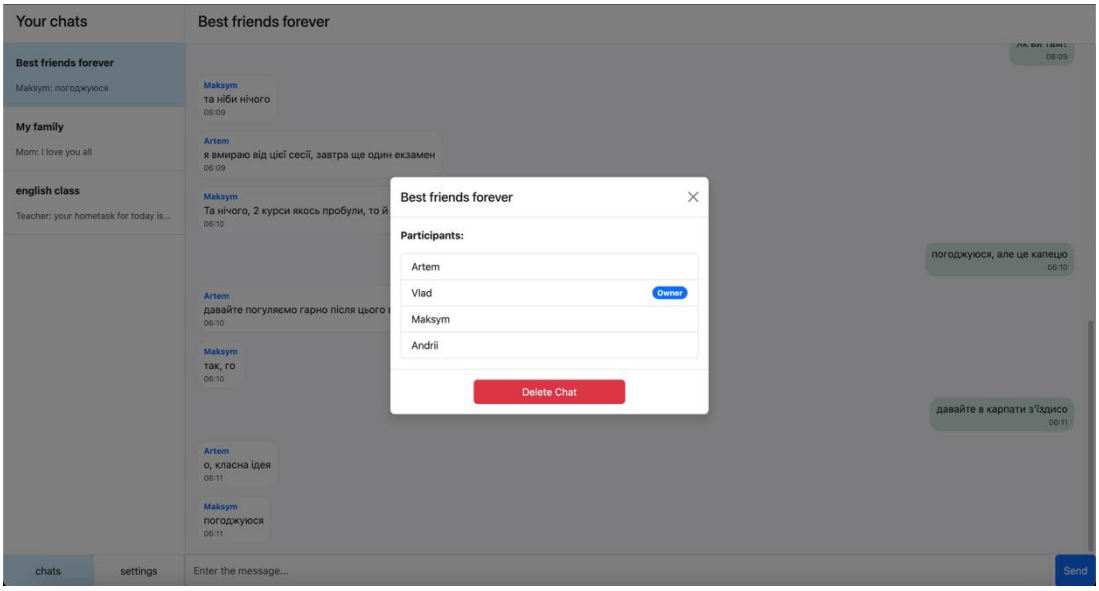

Рисунок 3.4. Модальне вікно чату

3.5 Реалізація WebSocket

Обмін повідомленнями в реальному часі є ключовою функцією месенджера, адже він дозволяє користувачам миттєво бачити нові повідомлення, зміни у чатах або дії інших учасників. У даному застосунку така функціональність реалізована за допомогою WebSocket у зв'язці з STOMP-протоколом.

В основі лежить клас WebSocketConfig, який містить анотацію @EnableWebSocketMessageBroker і реалізує інтерфейс WebSocketMessageBrokerConfigurer. Через метод registerStompEndpoints у конфігурації реєструється точка підключення /chat-websocket, яка підтримує SockJS – це забезпечує сумісність навіть з браузерами, що не підтримують WebSocket напряму. У результаті клієнт може встановлювати двостороннє з’єднання з сервером, використовуючи STOMP-повідомлення.

У сервісах MessageServiceImpl та ChatServiceImpl реалізовано логіку надсилання повідомлень усім зацікавленим клієнтам. Це реалізовано через інструмент SimpMessagingTemplate, який дозволяє передавати повідомлення у певні канали (топіки). Наприклад, при надсиланні нового повідомлення іншим учасникам чату надсилається повідомлення у топік /topic/chat/{chatId}, і лише ті, хто підписаний на цей топік (тобто є в цьому чаті) отримають це повідомлення.

У клієнтській частині, яка реалізована в JavaScript-файлах, при завантаженні сторінки здійснюється підключення до WebSocket і підписка на відповідні канали за допомогою бібліотеки sockjs-client та stomp.js. Коли відбувається зміна (наприклад, користувач надсилає повідомлення або видаляє чат), сервер публікує подію в конкретний топік, а всі підписані клієнти автоматично отримують оновлення без жодного REST-запиту або перезавантаження сторінки.

Висновок

У межах цієї курсової роботи було розглянуто теоретичні основи створення сучасних веб-застосунків на базі фреймворку Spring Boot, а також реалізовано повноцінний приклад застосунку у вигляді месенджера. У теоретичному розділі проаналізовано архітектуру веб-застосунків, принципи REST, роботу з базами даних за допомогою ORM, механізми безпеки на основі JWT, а також інтеграцію WebSocket і серверного рендерингу HTML через Thymeleaf.

У практичній частині було розроблено повнофункціональний застосунок, який підтримує реєстрацію та автентифікацію користувачів, створення й керування чатами, надсилання повідомлень у реальному часі, а також динамічну взаємодію з користувачем через веб-інтерфейс. Архітектура застосунку побудована за принципами багаторівневої організації: виділено окремі шари для контролерів, сервісів, репозиторіїв, маперів і валідації. Було реалізовано безпечну авторизацію.

Функціональність реального часу реалізовано через WebSocket-протокол і STOMP, що забезпечило оновлення інтерфейсу без додаткових запитів до сервера. Сучасний вигляд інтерфейсу досягнуто завдяки поєднанню шаблонів Thymeleaf, стилів Bootstrap та динаміки на JavaScript.

Отримані результати демонструють, що Spring Boot є надзвичайно гнучким та ефективним інструментом для побудови складних веб-систем. Виконання курсової роботи дозволило поглибити практичні навички у сфері backend-розробки, структурування застосунків, впровадження безпеки, побудови інтерактивних клієнтських інтерфейсів, і особливо в роботі з WebSocket, що було абсолютно новим для мене.

Використані джерела

1. The Provato Group. Why Choose Web Applications. https://www.theprovatogroup.com/applications/why-choose-web-applications/
2. Spring. Web applications. https://spring.io/web-applications
3. Medium. Front-end vs Back-end vs Client-side vs Server-side. https://medium.com/@priyapareek0205/front-end-vs-back-end-vs-client-side-vs-server-side-f5206a479c3e
4. FoxmindEd. REST API як спосіб спілкування компонент веб-додатків. https://foxminded.ua/shcho-take-rest-api/
5. Baeldung. Intro to Inversion of Control and Dependency Injection with Spring. https://www.baeldung.com/inversion-control-and-dependency-injection-in-spring
6. Spring. Aspect Oriented Programming with Spring. https://docs.spring.io/spring-framework/reference/core/aop.html
7. Baeldung. A Comparison Between Spring and Spring Boot. https://www.baeldung.com/spring-vs-spring-boot
8. Codefinity. Three-Level Architecture. https://codefinity.com/courses/v2/87dc501e-89a6-4a4b-afd4-8b38c46a80c7/9a2cd386-7414-4da6-a5ba-c79732024d97/3ce018a3-e609-4eda-bdbc-ee0ce9063d8b
9. Spring. Working with SQL Databases. https://docs.spring.io/spring-boot/docs/2.1.x/reference/html/boot-features-sql.html
10. Marcobehler. Spring Security: Authentication and Authorization In-Depth. https://www.marcobehler.com/guides/spring-security

11. Medium. WebSocket + Spring Boot: Build a Real-time, Bidirectional Applications. https://medium.com/simform-engineering/websocket-spring-boot-build-a-real-time-bidirectional-applications-e8c95bc19cbf
12. Baeldung. Introduction to Using Thymeleaf in Spring. https://www.baeldung.com/thymeleaf-in-spring-mvc
13. JavaRush. Бібліотека Lombok. https://javarush.com/ua/groups/posts/uk.2753.bbloteka-lombok
14. Baeldung. Database Migrations with Flyway. https://www.baeldung.com/database-migrations-with-flyway
15. Docker. How to Use the Postgres Docker Official Image. https://www.docker.com/blog/how-to-use-the-postgres-docker-official-image/