МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РОЗРОБКА ОНЛАЙН ЧАТУ НА БАЗІ ПРОТОКОЛУ WEBSOCKET ЗАСОБАМИ LARAVEL ТА REACT»**

Виконав: студент 4 курсу, групи 6.1210-2пі
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми програмна інженерія
(назва освітньої програми)
Д.С. Мельник
(ініціали та прізвище)
Керівник доцент кафедри фундаментальної та прикладної математики, доцент, к.ф.-м.н. Панасенко Є.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
Рецензент доцент кафедри комп'ютерних наук, доцент, к.т.н. Матвіїшина Н.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти бакалавр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма програмна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2023 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Мельнику Дмитрові Сергійовичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка онлайн чату на базі протоколу WebSocket засобами Laravel та React

керівник роботи Панасенко Євген Валерійович, к.ф.-м.н., доцент
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 21 » грудня 2023 року № 2180-с

2. Строк подання студентом роботи 03.06.2024 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі.
2. Основні теоретичні відомості.
3. Проєктування та розробка онлайн чату.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація за темою доповіді

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 25.12.2023 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	09.01.2024	
2.	Збір вихідних даних.	29.01.2024	
3.	Обробка методичних та теоретичних	26.02.2024	
	джерел.		
4.	Розробка першого та другого розділу.	08.04.2024	
5.	Розробка третього розділу.	20.04.2024	
6.	Оформлення та нормоконтроль		
	кваліфікаційної роботи бакалавра.	27.05.2024	
7.	Захист кваліфікаційної роботи.	19.06.2024	

Студент _____ Д.С. Мельник
(підпис) (ініціали та прізвище)

Керівник роботи _____ Є.В. Панасенко
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Розробка онлайн чату на базі протоколу WebSocket засобами Laravel та React»: 63 с., 20 рис., 15 джерел.

БАЗА ДАНИХ, ВЕБЗАСТОСУНОК, КОМПОНЕНТ, КОНТРОЛЕР, МОДЕЛЬ, ОНЛАЙН ЧАТ, ФРЕЙМВОРК, WEBSOCKET.

Об’єкт дослідження – процес розробки онлайн чату.

Мета роботи: розробка онлайн чату.

Методи дослідження – методи об’єктно-орієнтованого програмування, методи програмної інженерії.

Предмет дослідження – розробка онлайн чату на базі протоколу WebSocket засобами Laravel та React.

У роботі наведено етапи розробки онлайн чату на базі протоколу WebSocket, виявлено ключові проблеми та застосовано нові методи для збору та аналізу вимог до програмного забезпечення. Описано предметну область. Розроблено онлайн чат на базі протоколу WebSocket засобами Laravel та React, який використовуються для реалізації онлайн комунікації в реальному часі. Зростання попиту на інтерактивні онлайн сервіси робить важливим розв’язок таких задач, розгортання та підтримки онлайн чатів. Тому, розробка онлайн чату на базі протоколу WebSocket є актуальною задачею.

SUMMARY

Bachelor’s qualifying paper “Development of an Online Chat Based on the WebSocket Protocol Using Laravel and React” 63 pages, 20 figures, 15 references.

DATABASE, WEB APPLICATION, COMPONENT, CONTROLLER, MODEL, ONLINE CHAT, FRAMEWORK, WEBSOCKET.

Research Object: the process of developing an online chat. Objective: Development of an online chat.

Research Methods: methods of object-oriented programming, software engineering methods.

Subject of Research: development of an online chat based on the WebSocket protocol using Laravel and React.

The thesis outlines the stages of developing an online chat based on the WebSocket protocol, identifies key issues, and applies new methods for gathering and analyzing software requirements. The subject area is described. An online chat based on the WebSocket protocol was developed using Laravel and React, aimed at facilitating real-time online communication. The increasing demand for interactive online services underscores the importance of addressing such tasks, including the deployment and maintenance of online chats. Therefore, the development of an online chat based on the WebSocket protocol is a relevant task.

ЗМІСТ

ВСТУП

Розвиток технологій кардинально змінює різні аспекти нашого життя, включаючи способи комунікації та надання послуг в Інтернеті. У сучасній цифровій епосі важливо, щоб вебсайти мали ефективні та зручні інструменти для надання онлайн консультацій. Сучасні користувачі очікують миттєвої взаємодії та швидких відповідей на свої запити. Це особливо актуально для бізнесів, які прагнуть забезпечити високий рівень обслуговування клієнтів та покращити користувацький досвід на своїх вебресурсах.

Сьогодні вебсайти є невід'ємною частиною будь-якого бізнесу, слугуючи основним засобом комунікації з клієнтами та відвідувачами. В умовах зростаючої конкуренції компаніям необхідні рішення, які надають можливість реальної взаємодії з клієнтами в режимі реального часу, відповідаючи на їх запитання та надаючи необхідну допомогу. Використання онлайн консультантів на вебсайтах дозволяє досягти цього, забезпечуючи миттєве оновлення вмісту та передачу повідомлень.

Цей проєкт присвячений розробці онлайн консультанта для вебсайтів з використанням потужного протоколу WebSocket та сучасних технологій, таких як Laravel та React. Метою проєкту є створення ефективного інструменту для швидкої та плавної взаємодії з відвідувачами вебсайтів. WebSocket є протоколом двостороннього зв'язку, який дозволяє вебсайтам встановлювати постійне з'єднання з сервером та обмінюватися даними в режимі реального часу. Це значно покращує користувацький досвід, оскільки забезпечує швидку передачу повідомлень та оновлення контенту без необхідності перезавантаження сторінки.

В рамках цього проєкту будуть розроблені необхідні компоненти та інтерфейс для онлайн консультанта. Використання WebSocket дозволить забезпечити швидке і ефективне передавання повідомлень між користувачем і сервером. Крім того, будуть розглянуті можливості зберігання даних та

інтеграції з базами даних для забезпечення зручного керування консультаціями та клієнтськими запитами. Використання Laravel забезпечить надійну серверну частину, а React дозволить створити інтуїтивно зрозумілий та інтерактивний інтерфейс.

Особливу увагу буде приділено питанням безпеки та захисту даних, оскільки онлайн консультації часто включають обмін конфіденційною інформацією. Впровадження сучасних методів шифрування та аутентифікації допоможе забезпечити безпечне середовище для користувачів.

Цей проєкт має велике значення, оскільки він дозволяє покращити якість обслуговування клієнтів та відвідувачів вебсайту, забезпечуючи швидку та ефективну комунікацію. Використання WebSocket та сучасних технологій дозволяє створити інтерактивне середовище, яке задовольняє потреби користувачів та сприяє підвищенню їх задоволення від взаємодії з вебсайтом. Запровадження такого рішення також може сприяти підвищенню лояльності клієнтів, збільшенню рівня їх задоволення та покращенню загального іміджу компанії.

Таким чином, розробка онлайн консультанта на базі протоколу WebSocket з використанням Laravel та React є важливим кроком у напрямку створення ефективних та інноваційних вебзастосунків, що відповідають сучасним вимогам ринку та потребам користувачів.

1 ОГЛЯД ТЕОРЕТИЧНИХ ВЛАСТИВОСТЕЙ ТА ПЛАНУВАННЯ ПРОЄКТУ

У цьому розділі розглянемо теоретичні основи, пов’язані з розробкою онлайн чату, а також заплануємо основні етапи проєкту. Особливу увагу буде приділено історії розвитку технологій, що використовуються в онлайн чатах, принципам роботи протоколу WebSocket, а також фреймворкам Laravel та React. Додатково проведемо порівняння з іншими технологіями та протоколами, які можуть бути використані для реалізації подібних проєктів.

1.1 Історія та розвиток технологій онлайн чату

Історія технологій онлайн чату сягає своїм корінням досить далеко в минуле, але справжній прорив в цій сфері відбувся в другій половині 20 століття, коли почали розвиватися перші системи миттєвого обміну повідомленнями [3].

Одним із перших великих досягнень у цьому напрямку був створений в 1960-х роках на Масачусетському технологічному інституті (MIT) система CTSS (Compatible Time-Sharing System). Хоча CTSS була спрямована в основному на обмін даними та роботу з файлами, вона також мала функцію обміну повідомленнями, яка, можливо, є однією з перших спроб створення онлайн чату [5].

Наприкінці 20-го століття з’явилися перші версії онлайн чатів, які використовувалися для обміну текстовими повідомленнями. Ці системи, такі як IRC (Internet Relay Chat), ICQ (I Seek You) та AIM (AOL Instant Messenger), відкрили нові можливості для спілкування в Інтернеті.

Розглянемо детальніше про кожен із цих протоколів, та поринемо в їх історію.

Протокол IRC використовує текстовий інтерфейс та набір команд для взаємодії з системою. Користувачі можуть входити в різні канали, де вони можуть спілкуватися на різні теми з іншими учасниками. Кожен канал має свій власний ідентифікатор і може бути відкритим або приватним [6].

IRC став популярним серед тих, хто цікавився комп'ютерами та технологіями, та швидко став основним засобом спілкування для Інтернет-спільноти. Багато програм спеціально розроблені для підключення до серверів IRC, що дозволяє користувачам легко взаємодіяти з системою.

Хоча популярність IRC з часом зменшилася через появу нових соціальних мереж та месенджерів, він все ще використовується в деяких спеціалізованих спільнотах та для спілкування у великих групах користувачів. Він залишається важливим компонентом історії Інтернету та онлайн-спільнот.

ICQ месенджер надавав користувачам можливість обмінюватися повідомленнями в реальному часі через Інтернет. Особливістю ICQ була можливість ідентифікації користувачів за унікальними номерами, які називалися UIN (Universal Internet Number) [4].

ICQ отримав велику популярність в кінці 1990-х та на початку 2000-х років і став одним з основних засобів спілкування в Інтернеті. Він відкрив нові можливості для людей, які хотіли зберегти зв'язок з друзями та рідними, незалежно від їх місця знаходження [7].

ICQ включав в себе такі функції, як обмін текстовими повідомленнями, можливість відправляти файли та використання «статусів», які показували, чи доступний користувач для спілкування. Поступово до ICQ було додано нові функції, такі як відео-чат та голосові дзвінки.

AIM надавав користувачам можливість обмінюватися повідомленнями в реальному часі через Інтернет. Він був одним з перших месенджерів, який отримав широку популярність в Сполучених Штатах та інших країнах.

Однією з особливостей AIM була можливість користувачів встановлювати "screen names", які виступали як унікальні ідентифікатори для кожного користувача. Ці screen names дозволяли людям знаходити своїх друзів та спілкуватися з ними в режимі реального часу [5].

AIM також включав в себе такі функції, як обмін текстовими повідомленнями, можливість відправляти файли, голосові та відео-чати. Він був популярним серед користувачів, які шукали зручний спосіб спілкування з друзями та рідними.

З поширенням Інтернету та зростанням популярності соціальних мереж у 2000-х роках, вебчати стали загальноприйнятими серед спільноти Інтернету. Платформи, такі як Facebook, забезпечували можливість спілкування в реальному часі за допомогою чатів та приватних повідомлень. Деякі соціальні мережі навіть впроваджували відео- та голосові чати, що розширило можливості спілкування.

У сучасній ері мобільних технологій месенджери стали основним інструментом для спілкування в реальному часі. Платформи, такі як WhatsApp, Telegram, Viber, роблять акцент на безкоштовному обміні повідомленнями, а також відео- та голосовому спілкуванні. Мобільні додатки дозволяють користувачам бути зв’язаними навіть у рухливому режимі, забезпечуючи постійну доступність до чату [9].

З використанням сучасних технологій, таких як протокол WebSocket, який забезпечує швидку та ефективну двосторонню комунікацію, онлайн чати продовжують розвиватися. Завдяки цим інноваціям, користувачі можуть спілкуватися в реальному часі з будь-якої точки світу, забезпечуючи безперервний потік інформації та зручне спілкування.

1.2 Протокол WebSocket: принципи роботи та особливості

WebSocket – це комунікаційний протокол, який дозволяє встановлювати двостороннє з’єднання між клієнтом і сервером в режимі реального часу через одне TCP-з’єднання. Цей протокол забезпечує безперервний обмін даними між клієнтом та сервером, що дозволяє миттєво передавати оновлення без необхідності повторного запиту на сервер.

Основні принципи роботи WebSocket включають:

- **ініціація з'єднання** – клієнт і сервер встановлюють з'єднання через HTTP, після чого клієнт ініціює перехід на протокол WebSocket, надсилаючи спеціальний запит;
- **рукостискання (Handshake)** – отримання запиту сервер перевіряє його та відповідає згодою на встановлення з'єднання WebSocket, після цього з'єднання переходить у режим WebSocket і готове до обміну даними;
- **бінарний та текстовий обмін даними** – дозволяє як текстовий, так і бінарний обмін даними між клієнтом і сервером – це дозволяє передавати різноманітні типи інформації, включаючи текст, зображення, аудіо та відео;
- **двостороння комунікація** – після встановлення з'єднання WebSocket обидва боки можуть надсилати дані один одному в будь-який час без необхідності повторного встановлення з'єднання.

Звичайно, особливості протоколу WebSocket – це ключові характеристики, які роблять його привабливим для використання в різних вебдодатках. Розглянемо деякі з них більш детально:

- однією з основних переваг WebSocket є здатність забезпечити швидку та ефективну передачу даних між клієнтом і сервером, у порівнянні з традиційними методами, такими як HTTP, WebSocket зменшує затримку та кількість обміну повідомленнями, що дозволяє отримувати дані миттєво;
- після того, як з'єднання WebSocket встановлено, воно залишається відкритим протягом тривалості сеансу, що дозволяє клієнтам і серверам обмінюватися даними без необхідності повторного встановлення з'єднання для кожного запиту;
- WebSocket дозволяє вебдодаткам взаємодіяти в реальному часі, що дозволяє створювати додатки з миттєвою оновленням та сповіщеннями для користувачів – це особливо важливо для таких додатків, як онлайн чати, спільні документи, ігри тощо;

– є крос-платформеним протоколом, що означає, що він підтримується на різних пристроях і платформах, таких як комп’ютери, смартфони, планшети тощо – це дозволяє створювати вебдодатки, які працюють на будь-якому пристрої, що має підтримку вебтехнологій;
– дозволяє зменшити навантаження на сервер та мережу, оскільки він дозволяє підтримувати постійне з’єднання, а не встановлювати нове з’єднання для кожного запиту, як у випадку з традиційним HTTP.

Завдяки своїм принципам роботи та особливостям, протокол WebSocket став популярним засобом для реалізації вебдодатків, які вимагають швидкої та ефективної двосторонньої комунікації між клієнтом і сервером [6].

1.3 Огляд фреймворків Laravel та React

Давайте тепер поговоримо про конкретні технології для реалізації нашого проєкту, а саме фреймворк React для написання фронтенду та Laravel для написання беккенду. Розглянемо кожний з них окремо та вивчимо їх головні аспекти. Почнемо з налаштування серверної частини нашого проєкту, а саме з фреймворку Laravel.

Laravel використовують розробники backend, які пишуть код на PHP. Він допомагає визначити структуру вебдодатку і служить для неї каркасом. Фреймворк написаний на PHP і розширює його можливості. Розповсюджується безкоштовно під ліцензією MIT [10].

Мета Laravel полягає у створенні вебдодатків та сайтів на основі MVC (model-view-controller). Це варіант архітектури, за якого компоненти програми поділяються на три частини:

– модель (model) надає дані та методи роботи з ними: запити до бази даних, перевірка на коректність;
– представлення (view) відображає користувачу ці дані і змінюється, якщо змінюється модель;

– контролер (controller) направляє дані від користувача до системи і навпаки.

Коли користувач працює з додатком, побудованим за схемою MVC, він взаємодіє з представленням та контролером. Представлення – це те, що він бачить, наприклад, інформація, яка відображена в візуальному інтерфейсі. А контролеру користувач віддає команди.

Наприклад, натискання кнопки в додатку буде зверненням до контролера. Він змінить модель. Потім оновиться представлення: воно отримає з моделі нові дані та змінить стан. Так користувач побачить результат натискання на кнопку.

Наглядна взаємодія представлена на рисунку 1.1 [1].

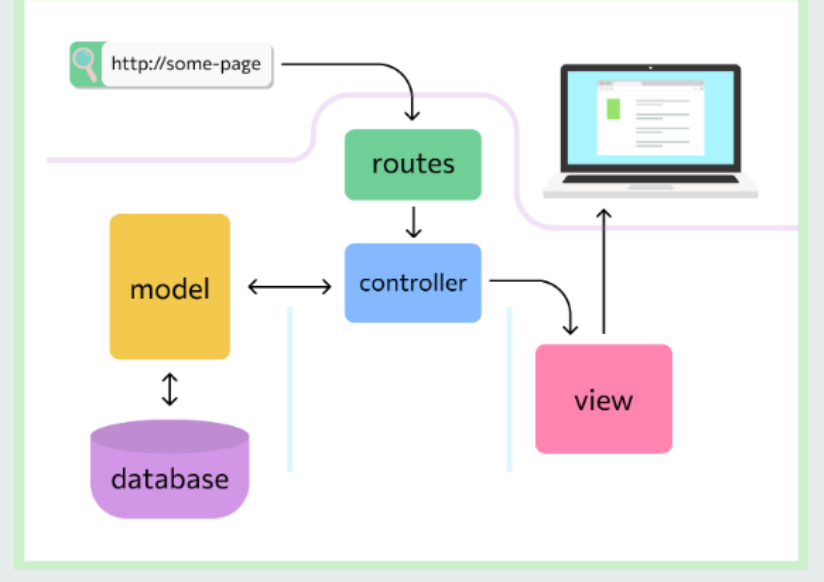


Рисунок 1.1 – Архітектура MVC [3]

Архітектура MVC дозволяє писати більш зрозумілий код і зробити процес розробки більш комфортним, розділяючи роботу між frontend – та backend-розробниками.

Artisan – це інтерфейс командного рядка, вбудований в Laravel. Він дозволяє генерувати моделі, нові тести, контролери, повідомлення з командного рядка. Це набагато зручніше, ніж копіювати шаблон класу звідки-небудь або писати його вручну.

Консоль використовують для роботи з базами даних, керування контролером та іншими діями. Розробник може написати для неї власні команди [6].

ORM – це технологія програмування для зв’язку бази даних і мови програмування. Використання ORM дозволяє прискорити розробку. Для PHP існує багато реалізацій ORM, але Laravel використовує власну. Вона називається Eloquent і працює за схемою ActiveRecord, згідно з якою кожній таблиці в базі відповідає один клас. Eloquent відомий своєю зручністю і можливістю писати зрозумілий код, який легко підтримувати, захист від SQL-ін’єкцій – способу вторгнення на сайти та програми. ORM можна завантажити окремо від інших компонентів Laravel.

Конструктор Fluent – це конструктор для швидкої побудови запитів до бази даних, який повністю сумісний з ядром Eloquent ORM. Fluent – потужний інструмент, який дозволяє з легкістю модифікувати записи в базі даних. Усі запити використовують підготовлені вирази та захищені від SQL-ін’єкцій [12].

Шаблонізатори використовуються для перетворення HTML-шаблонів в готові сторінки. Шаблони – це заготовки для майбутніх вебсторінок, які включають HTML-верстку без контенту та PHP-код. Задача програми-шаблонізатора – виконати PHP-код та підставити в шаблон контент, щоб перетворити його в готову сторінку. PHP в кінцевій вебсторінці бути не повинно.

Blade – це шаблонізатор фреймворка Laravel. Він не має обмежень на чистий PHP у шаблонах – це зручніше для backend-розробника. Відсутність обмежень не створює додаткового навантаження на додаток.

Валідація – це перевірка вхідних даних. Вона необхідна для того, щоб вчасно виявити та обробити випадкову помилку. Логіку валідації можна написати самостійно. Якщо користувач введе невірні дані, він буде перенаправлений на попередню сторінку. Сайт не «впаде» і залишиться працездатним [15].

Юніт-тестування ще називають модульним тестуванням. Це «базовий рівень» проведення тестів. На працездатність перевіряються окремі модулі коду. Це необхідно, щоб пізніше не стикнутися з помилкою на вищих рівнях.

Прилад піраміди етапів тестування наведена на рисунку 1.2 [3].

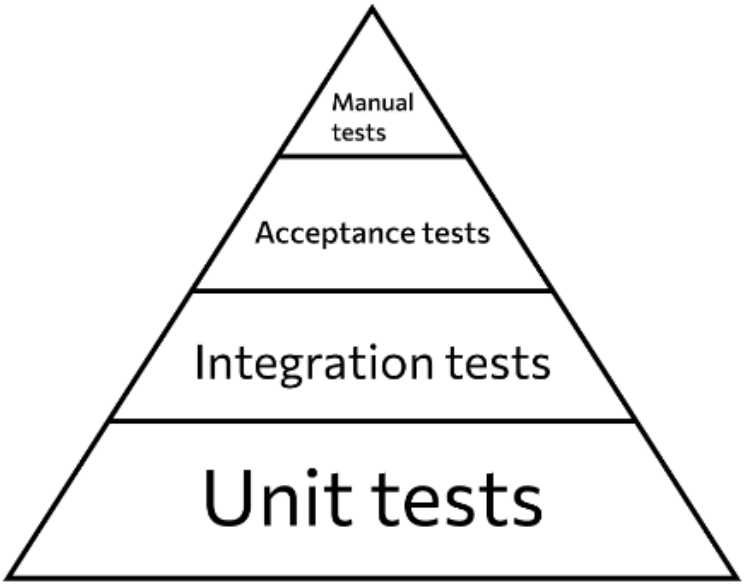


Рисунок 1.2 – Етапи тестування [3]

За допомогою Laravel можна реалізувати вхід на сайт за допомогою логіна та пароля або через соціальні мережі.

Наприклад, пакети Laravel Passport та Laravel Socialite дозволяють виконувати вхід через API інших додатків за стандартом OAuth. Це протокол автентифікації, який дає можливість авторизуватися за допомогою облікового запису іншого сайту, частіше за все – соціальної мережі.

Пакет Laravel Sanctum відповідає за аутентифікацію для простих додатків та односторінкових сайтів. Він є більш легким у використанні, ніж Passport, та авторизує користувача за допомогою системи токенів – фізичних пристроїв [12].

Також я би хотів виділити декілька особливостей та переваг Laravel:

– висока продуктивність;
– безпека;
– відкритий код;

– зручний синтаксис;
– гнучка маршрутизація;
– легка міграція;
– багатофункціональність.

Незважаючи на широкий функціонал, Laravel вимагає від розробників значної кількості ручної роботи. Це забезпечує гнучкість у створенні програмних рішень, але може також призвести до затримок у роботі. Крім того, вивчення Laravel може бути складнішим у порівнянні з конкурентами [13].

Незважаючи на численні переваги, фреймворк Laravel має деякі недоліки:

– складність для початківців;
– продуктивність;
– оновлення та сумісність;
– багатофункціональність;
– обсяг коду;
– змінність;
– навчання та переоцінка команди;
– залежність від сторонніх пакетів.

Тепер більше поговоримо про фреймворк для написання клієнтської частини, а саме про фреймворк React.js.

React.js – це бібліотека для JavaScript з відкритим вихідним кодом для розробки користувацьких інтерфейсів. Вона допомагає швидко і легко реалізувати реактивність – явище, коли у відповідь на зміну одного елемента змінюється все інше.

React використовують фронтенд-розробники. У моделі MVC, де Model-View-Controller – це назва, що розшифровується як Model-View-Controller, інтерфейс – це View, представлення, з яким взаємодіє користувач, та частина вебсайту або додатка, яка видна людині. Фахівці, які з нею працюють, зокрема, використовують React.js. Також з React можуть

працювати верстальники, тестувальники та інші спеціалісти, задіяні у створенні вебінтерфейсів [11].

Основною функцією React є створення односторінкових і багатосторінкових додатків, розробки великих сайтів. Наприклад, за його допомогою написаний сервіс стрімінгу Netflix і реалізовані новинні стрічки найбільших соціальних мереж. Бібліотека призначена для:

– створення функціональних інтерактивних вебінтерфейсів, з якими працюють, не потрібно постійно оновлювати сторінку;
– швидкої та зручної реалізації окремих компонентів та сторінок в цілому – елементи в React легко використовувати повторно;
– легкої розробки складних програмних структур – їх просто описувати, якщо використовувати реалізований в React підхід;
– допрацювання нової функціональності з будь-яким початковим набором технологій: бібліотека не залежить від решти інструментарію і працюватиме добре на будь-якому коді;
– розробки односторінкових і багатосторінкових додатків (SPA і PWA);
– роботи з серверною частиною сайту або розробки мобільних додатків – таких випадках React використовують разом із засобами, які адаптовують вебтехнології для інших цілей.

React має ряд особливостей, які роблять його гнучким та потужним інструментом. Ми розглянемо деякі з них [7].

React не є повноцінним фреймворком, а лише бібліотекою для створення користувацьких інтерфейсів. Важливо розуміти різницю між фреймворками та бібліотеками.

Фреймворк – це комплексне рішення, яке надає набір інструментів, бібліотек та правил для розробки додатків. Фреймворк часто визначає структуру додатка, а розробнику надається менше вільності у виборі інструментів та архітектурних рішень. Приклади фреймворків включають Angular та Ember.

Бібліотека – це набір функцій та компонентів, які допомагають у вирішенні певних задач, але не нав'язують загальну архітектуру додатка. Розробник має більшу свободу вибору інших інструментів та архітектурних рішень. React – це приклад бібліотеки, яка зосереджена на створенні користувацьких інтерфейсів.

Таким чином, React надає інструменти для створення компонентів користувацького інтерфейсу, керування станом та взаємодії з DOM, але не нав'язує конкретну архітектуру додатка. Розробники можуть використовувати React у поєднанні з іншими бібліотеками та інструментами на свій розсуд, що робить його гнучким і дозволяє створювати різноманітні типи вебдодатків.

Всі вебінтерфейси базуються на HTML-документах та CSS-стилях, до яких підключено JavaScript-код. Структура HTML-документа, а точніше його модель, відома як DOM-дерево (DOM розшифровується як Document Object Model, об'єктна модель документа). Це деревоподібна модель, в якій ієрархічно зібрані всі елементи, що використовуються на сторінці.

React включає механізм, який керує процесом рендерингу компонентів. Рендеринг – це процес створення візуального представлення компонента, яке відображається в браузері або мобільному додатку.

Особливість React полягає в тому, що він створює та зберігає в кеші віртуальне DOM-дерево – копію DOM, яка змінюється швидше, ніж реальна структура. Це потрібно для швидкого оновлення сторінок. Якщо користувач виконає дію або виникне подія, DOM повинна змінитися, оскільки зміняться об'єкти на сторінці. Але реальна об'єктна модель може бути великою, її оновлення – повільним процесом. Тому React працює не з нею, а з віртуальною копією в кеші, яка важить менше.

Коли відбувається подія, через яку код має оновити об'єкт, зміна швидко відображається у віртуальному DOM. Після цього оновлюється реальна об'єктна модель. Для користувача це означає, що зміни на сторінці відобразяться миттєво, а не після тривалого завантаження [1].

Для поліпшення продуктивності React не повністю оновлює DOM. Він зберігає в пам’яті дві легші копії: поточну та попередню. Коли щось оновлюється, бібліотека порівнює версії між собою і змінює лише ту частину дерева, яка дійсно змінилась. Це потрібно, щоб не перезавантажувати DOM повністю і не уповільнювати сторінку. Підхід може здатися складним, але він важливий для оптимізації завантаження.

Реакт базується на компонентах – окремих елементах вебінтерфейсу. Компоненти інкапсульовані, що означає, що в кожному з них розміщені всі необхідні методи та дані. Детальніше про інкапсуляцію можна прочитати в нашій статті про ООП: це один з ключових критеріїв об’єктно-орієнтованого підходу. У випадку з компонентами Реакт інкапсуляція також означає, що стан елемента зберігається в ньому самому, а не в окремому об’єкті.

Інкапсульовані самостійні компоненти можна використовувати повторно, розміщати в іншому місці коду, в іншому розділі або на іншій сторінці. Дані можна переносити по всьому додатку, використовувати поза DOM конкретної сторінки. Це прискорює розробку та скорочує кількість дій для створення працюючого інтерфейсу. Благодаря відсутності складних залежностей інкапсуляція також спрощує відлагодження [5].

Компоненти можуть передавати свої властивості та дані один одному, але тільки в одному напрямку – від «батьківських» до «дочірніх». Це допомагає реалізувати чітку ієрархію та спрощує відлагодження. Однонапрямковий потік даних означає, що програміст завжди може зрозуміти, звідки саме до елементу надійшли дані.

Ще одна особливість потоку даних – відсутність можливості змінювати властивості безпосередньо. Для цього існують спеціальні зворотні виклики (callback-функції). Так називаються частини виконуваного коду, які передаються від одного компонента до іншого у вигляді параметра функції. Це дозволяє підтримувати стабільність: властивості залишаються незмінними після створення, а callback-функція, мовляв, перестворює їх. В результаті використання колбеків потік даних виглядає так: властивості та інформація передаються зверху вниз, а події відбуваються знизу вгору [14].

У старій версії React, керування станами зазвичай відбувалося за допомогою класів – спеціальних конструкцій, про які можна дізнатися більше в статті про ООП. Зараз в React.js існує підтримка хуків – так називаються спеціальні функції-«крючки», які «підключаються» до стану елемента або до методу. Зміна стану або виклик методу «тягне» за собою ці функції, і вони автоматично виконуються – це допомагає уникнути використання класів, полегшує і спрощує написання коду.

У React існують вбудовані хуки, але розробник може створювати й власні – це допомагає повторно використовувати код і зменшити його кількість.

React Developer Tools – це додаток для браузера, розроблений створювачами React.js, щоб полегшити процес перевірки та відлагодження React-додатків. Цей інструмент надає розширені можливості для відлагодження React-додатків прямо в браузері. Він дозволяє переглядати і аналізувати структуру компонентів, їх стан та властивості, а також здійснювати взаємодію з компонентами безпосередньо з панелі розробника браузера [5].

React Developer Tools є незамінним інструментом для фронтенд-розробників, оскільки вони дозволяють ефективно відлагоджувати та аналізувати React-додатки безпосередньо в браузері.

Розглянемо переваги React.js:

– швидкість рендерингу;
– компонентна архітектура;
– односторонній потік даних;
– велика спільнота та екосистема;
– декларативний синтаксис.

Не зважаючи на всі переваги, React.js має свої недоліки:

– великий поріг входження;.
– незручність при роботі з формами;
– необов’язкове використання JSX;

– потреба в додаткових бібліотеках для повноцінного SPA;
– проблеми з SEO.

Отже, у цьому розділі ролзглянуто переваги та недоліки React.js та Laravel. React відрізняється високою продуктивністю та гнучкістю завдяки використанню компонентної архітектури та засобам для роботи з віртуальним DOM. Проте, його навчання може бути вимогливим, а також можливі незручності з JSX та формами.

У свою чергу, Laravel пропонує розширені можливості для швидкого розробки вебзастосунків, включаючи готові інструменти для маршрутизації, аутентифікації та кешування. Однак, він може виявитися складним для новачків та має певні обмеження у продуктивності.

Обидва інструменти мають велику спільноту користувачів та активно підтримуються розробниками, що робить їх популярними виборами для розробки сучасних вебзастосунків [8].

1.4 Порівняння з іншими технологіями та протоколами

Так як ми вже розглянули фронтенд та бекенд, та проаналізували фреймворки, давайте порівняємо їх з іншими популярними технологіями та фреймворками для створення онлайн чату. Для початку порівняємо заклятого ворога React, а саме Vue [3].

Що ж спільного між React і Vue:

– обидва вони мають компонентно-орієнтований підхід, який дозволяє розбивати додаток на менші, легко повторно використовувані та тестирувані частини;
– використовують реактивну модель, яка автоматично оновлює представлення при зміні даних моделі;
– використовують шаблони у вигляді HTML-синтаксису, що дозволяє розробникам легко візуалізувати компоненти та більш наочно бачити їх структуру;

– мають активно розвиваючеся співтовариство та широко використовуються в промисловій розробці.

Хоча React та Vue мають багато спільного, між ними є кілька ключових відмінностей:

– архітектура: React має більш модульну архітектуру, де компоненти та стан є окремими сутностями, тоді як Vue має більш інтегровану архітектуру, де компоненти та стан є більш пов’язаними;
– можливості: React має розширені можливості для розробки складних та масштабованих додатків, але Vue має простий та інтуїтивно зрозумілий API, що робить його більш підходящим для початківців та менш складних проєктів;
– екосистема: React має потужну та масштабну екосистему, з багатьма інструментами та бібліотеками, які можуть бути використані для розробки, тоді як Vue має меншу екосистему, але вона більш інтегрована та легко налаштовується;
– використання: React широко використовується для розробки великих та масштабованих додатків, таких як вебпортали та платформи, тоді як Vue широко використовується для створення менш складних інтерактивних вебдодатків, таких як корпоративні сайти та інтерактивні вебсайти;
– спільнота: React розробляється та підтримується компанією Facebook, має більше велике співтовариство розробників та має більше інструментів та бібліотек, які можуть бути використані для розробки. Vue ж розробляється та підтримується спільнотою.

Незважаючи на те, що дискусія про те, хто кращий, триває вже не перший рік, аналізуючи всі переваги та недоліки, ми можемо сказати, що вони є рівними. Все залежить від конкретних завдань вашого проєкту. Кожен з цих фреймворків має свої сильні сторони і недоліки, і правильний вибір залежить від ваших потреб, вимог та умов. А тепер почнемо порівнювати Symfony та Laravel [10].

Схожість між цими технологіями досить велика. По-перше, обидва фреймворки розроблені на PHP, мають відкритий вихідний код за ліцензією MIT і базуються на архітектурному шаблоні MVC. При докладнішому порівнянні виявляється ще більше схожостей:

– вбудований шаблонізатор (Twig і Blade);
– ORM для роботи з популярними базами даних (Doctrine і Eloquent);
– інтерфейс командного рядка (CLI і Artisan);
– інструменти для unit-тестування;
– сумісність з різними операційними системами;
– розширюваність сторонніми модулями та компонентами;
– багатомовність.

На перший погляд, платформи можуть здаватися майже ідентичними, але основні відмінності ховаються у дрібницях. Наприклад:

– MVC-архітектура: залежності, встановлені за замовчуванням, можуть обмежувати розробників, тоді як модульність, навпаки, може надати більше можливостей;
– шаблонізатор: Twig, на відміну від Blade, забезпечує додатковий рівень безпеки та додаткові функції, а також сумісний з Slim та Yii;
– ORM: Doctrine підтримує більше баз даних з коробки, включаючи Oracle та SAP Sybase SQL (процес міграції даних тут автоматизований, а для моделювання даних потрібно менше знань SQL, у Eloquent міграції виконуються вручну, але не потребують визначення полів).

Обидві платформи вже довели свою ефективність, і щоб не помилитися у виборі, важливо врахувати пріоритети а саме можливість підтримувати велику користувацьку базу чи швидкість виходу на ринок. Ретельний аналіз цих аспектів допоможе прийняти оптимальне рішення та забезпечити успіх нашого проєкту.

У першому випадку без вагань обирайте Symfony. Ця платформа призначена для великомасштабних проєктів рівня підприємства, і її

використовують такі компанії, як Spotify, Trivago та Dailymotion. Вартість розробки буде вищою на перших етапах, але всі інвестиції повернуться під час супроводу та розробки системи [12].

На пошук та найм досвідчених розробників можуть піти місяці, початкове налаштування та підготовка архітектури, а також функціональність пройде час. Усе це погано впливає на TTM, критичний показник у конкурентній гонці. Обравши Laravel, ви зекономите значну кількість ресурсів на старті і уникнете зайвих витрат на функції, які можуть виявитися непотрібними.

Обидві платформи дозволяють створювати сучасні вебдодатки з унікальними функціональними можливостями. Наприклад, це може бути корпоративний портал з системою особистих кабінетів, CRM або ERP, додаток для електронної комерції та не лише [13].

2 ПРОЄКТУВАННЯ ТА АНАЛІЗ ВИМОГ ДО ОНЛАЙН ЧАТУ

2.1 Опис поставленої задачі

Онлайн чат є невід'ємною частиною сучасних комунікаційних платформ, які забезпечують миттєвий обмін повідомленнями між користувачами в режимі реального часу. Предметна область даного проєкту охоплює розробку системи, яка дозволяє користувачам спілкуватися один з одним за допомогою текстових повідомлень через вебінтерфейс.

Основні аспекти предметної області:

– система повинна забезпечувати миттєву передачу повідомлень між користувачами, що вимагає використання технологій, які підтримують двосторонній зв'язок, таких як WebSocket – це дозволяє знизити затримки в обміні повідомленнями і забезпечити безперервний потік даних між клієнтом і сервером;
– Laravel: PHP-фреймворк, який буде використовуватися для побудови серверної частини додатка, Laravel забезпечує зручність у роботі з базами даних, управлінням маршрутизацією, аутентифікацією та іншими аспектами бекенду;
– React: JavaScript-бібліотека, яка буде використана для побудови інтерфейсу користувача, React дозволяє створювати динамічні та швидкодійні вебдодатки завдяки своїй компонентній архітектурі;
– основні користувачі системи – це зареєстровані користувачі, які можуть створювати та підтримувати діалоги (крім звичайних користувачів, можуть бути також адміністратори, які мають розширені права для управління чатами та користувачами);
– інтерфейс користувача повинен бути інтуїтивно зрозумілим та зручним, забезпечуючи легкий доступ до основних функцій чату – це включає в себе можливість перегляду історії повідомлень, створення

нових чатів, додавання або видалення учасників, а також налаштування профілю користувача;

– забезпечення безпеки даних користувачів і конфіденційності їхніх повідомлень є критично важливим аспектом системи – це включає в себе використання сучасних методів шифрування для передачі даних, а також механізмів аутентифікації та авторизації;
– система повинна бути спроєктована таким чином, щоб витримувати навантаження великої кількості одночасних користувачів без втрати продуктивності – це вимагає продуманої архітектури та використання ефективних алгоритмів для обробки повідомлень;
– можливість інтеграції з іншими системами та сервісами, такими як соціальні мережі, служби аутентифікації, а також підтримка сторонніх плагінів і розширень, що додають нові функціональні можливості.

Для більшого розуміння проєкту розробки онлайн чату на базі протоколу WebSocket засобами Laravel та React, створимо UML діаграму бази даних (див. рис. 2.1).

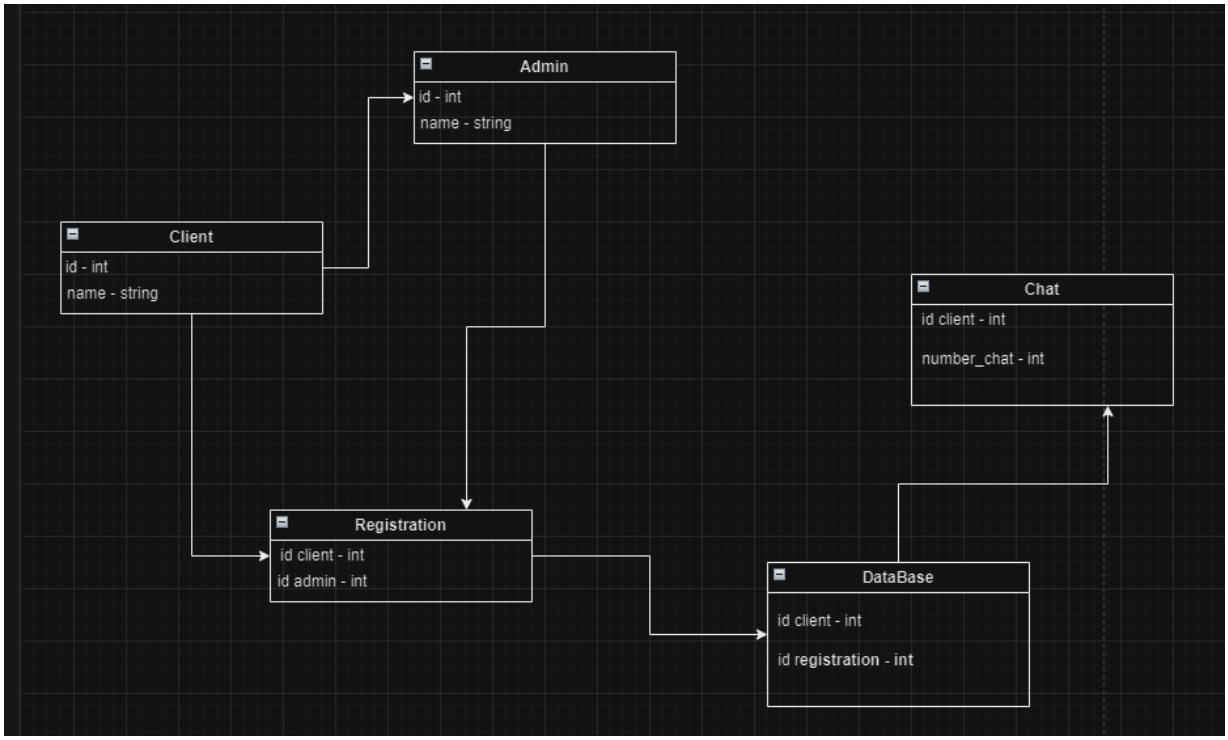


Рисунок 2.1 – ER-Діаграма бази даних

Діаграма прецедентів є однією з основних діаграм, що використовуються при проєктуванні програмного забезпечення. Вона ілюструє взаємодії між системою (або програмою) та її користувачами, відомими як актори. Діаграма прецедентів включає акторів, прецеденти та взаємодії між ними. Актори представляють користувачів системи або інші системи, що взаємодіють з нею. Прецеденти описують дії, які система або її частина може виконати. Взаємодії показують, як актор взаємодіє із системою. Така діаграма є корисною при проєктуванні програмного забезпечення, оскільки допомагає зрозуміти вимоги користувачів і функціональність системи. Вона також служить інструментом для спілкування між розробниками та клієнтами, надаючи чітке візуальне уявлення про те, як буде використовуватися програмне забезпечення. На рисунку 2.2 представлено діаграму прецедентів [1].

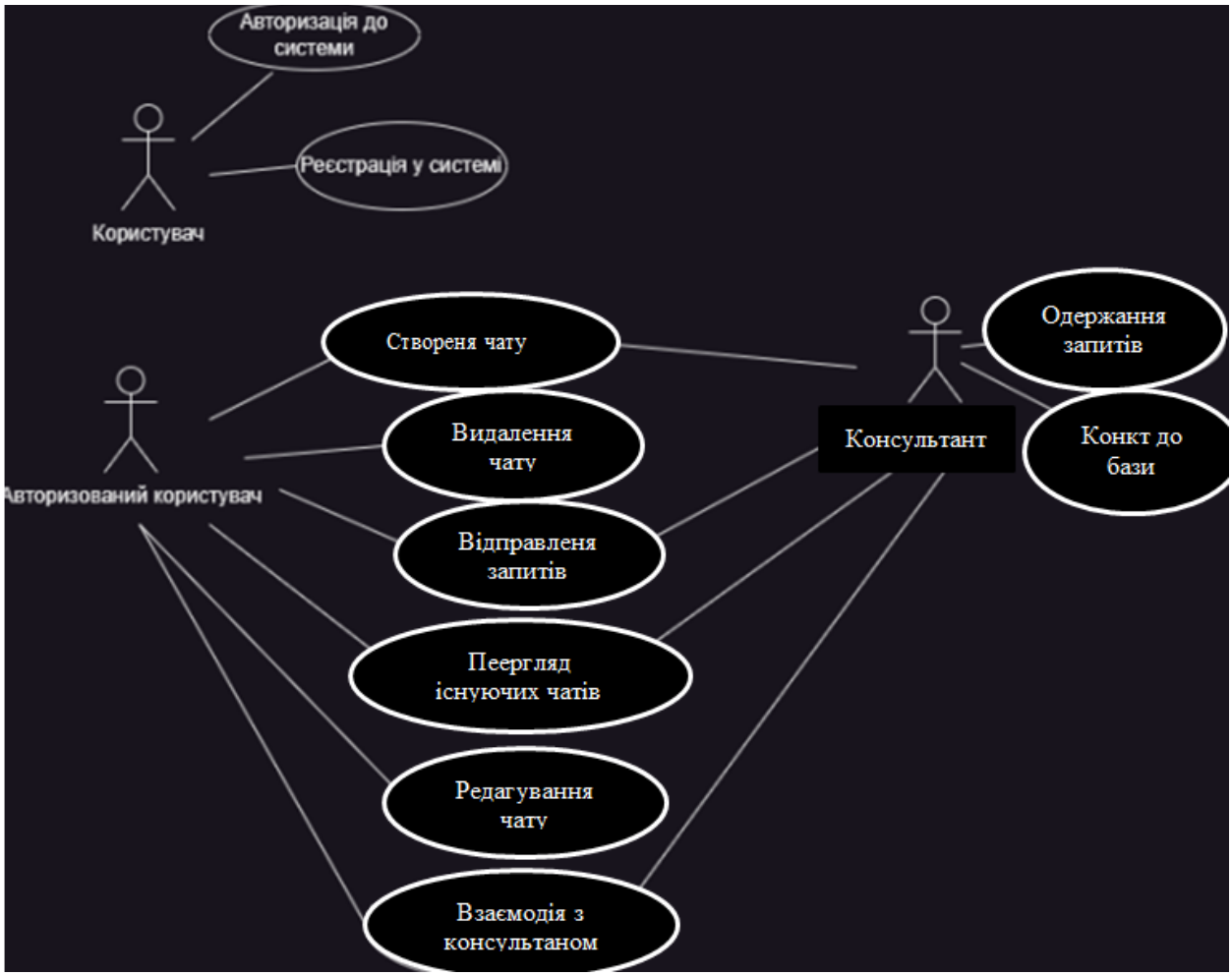


Рисунок 2.2 – Діаграма прецедентів

2.2 Функціональні вимоги

Функціональні вимоги визначають, яку функціональність повинна забезпечувати система для задоволення потреб користувачів. Вони описують конкретні дії, які повинна виконувати система, щоб користувачі могли досягти своїх цілей.

Для розробки онлайн чату на базі протоколу WebSocket засобами Laravel та React було визначено наступні функціональні вимоги:

- користувачі повинні мати можливість створювати нові облікові записи;
- користувачі повинні мати можливість входити у систему, використовуючи свої облікові дані;
- система повинна забезпечувати функціональність відновлення пароля;
- адміністратори повинні мати можливість переглядати список зареєстрованих користувачів;
- адміністратори повинні мати можливість блокувати або видаляти користувачів;
- користувачі повинні мати можливість надсилати та отримувати повідомлення у реальному часі;
- система повинна підтримувати групові чати та приватні повідомлення;
- повідомлення повинні зберігатися на сервері для подальшого перегляду;
- користувачі повинні отримувати сповіщення про нові повідомлення;
- система повинна підтримувати налаштування сповіщень (увімкнення/вимкнення звуку тощо);
- повідомлення повинні передаватися з мінімальною затримкою;
- система повинна забезпечувати безпечну передачу даних за допомогою шифрування;

- система повинна мати механізми для запобігання несанкціонованому доступу до даних;
- інтерфейс користувача повинен бути зручним та інтуїтивно зрозумілим;
- користувачі повинні мати можливість налаштовувати вигляд чату відповідно до своїх вподобань;
- система повинна зберігати історію повідомлень для кожного чату;
- користувачі повинні мати можливість переглядати попередні повідомлення.

Визначення цих функціональних вимог дозволяє чітко уявити, яку функціональність повинен мати онлайн чат, щоб відповідати потребам користувачів та забезпечити їх ефективне спілкування [1].

2.3 Нефункціональні вимоги

Для успішної реалізації онлайн чату на базі протоколу WebSocket, розробленого засобами Laravel та React, було враховано кілька ключових нефункціональних вимог. Ці вимоги забезпечують високу якість, надійність і зручність використання системи. Давайте розглянемо детально кожну з категорій нефункціональних вимог, які були враховані під час розробки проєкту.

Продуктивність системи є одним із головних аспектів, що визначає її ефективність та задоволеність користувачів.

Час відгуку: чат забезпечує миттєву реакцію на дії користувачів. Час відгуку на відправку та отримання повідомлень складає не більше 200 мілісекунд. Це досягається завдяки оптимізації роботи WebSocket-з’єднань та ефективному використанню ресурсів серверу.

Пропускна здатність: система здатна одночасно обслуговувати щонайменше 500 активних користувачів без зниження продуктивності. Це

стало можливим завдяки масштабованій архітектурі та використанню балансувальників навантаження.

Час завантаження: сторінка чату завантажується не більше ніж за 3 секунди при нормальному мережевому з'єднанні. Це покращує користувацький досвід та знижує ймовірність відмови користувачів від використання чату.

Надійність системи охоплює її доступність, відмовостійкість та можливість відновлення після збоїв.

Доступність: система є доступною не менше 99.9% часу. Це забезпечується завдяки використанню резервних серверів та механізмів автоматичного відновлення.

Відмовостійкість: чат продовжує функціонувати навіть при відмові окремих компонентів. Для цього використовуються резервування та автоматичне відновлення критичних сервісів.

Відновлення після збоїв: система швидко віднвлюється після збоїв. Час відновлення не перевищує 10 хвилин, що мінімізує перерви в роботі та забезпечує безперервний доступ користувачів до чату.

Безпека системи включає захист даних користувачів, аутентифікацію та авторизацію, а також захист від атак.

Захист даних: повідомлення та дані користувачів передаються зашифрованими каналами. Ми використовуємо SSL/TLS для забезпечення безпеки передачі даних.

Аутентифікація та авторизація: система забезпечує надійну аутентифікацію та авторизацію користувачів, запобігаючи несанкціонованому доступу до чату.

Захист від атак: система захищена від типових атак, таких як SQL-ін'єкції, XSS, CSRF та атаки типу «відмова в обслуговуванні» (DoS).

Масштабованість системи визначає її здатність обробляти зростаючу кількість користувачів та навантаження:

– горизонтальна масштабованість: система підтримує горизонтальну

масштабованість, що дозволяє додавати нові сервери для обробки збільшеного навантаження;

– вертикальна масштабованість: система підтримує вертикальну масштабованість, що дозволяє збільшувати потужності існуючих серверів.

Зручність користування охоплює вимоги до інтерфейсу користувача та загальної взаємодії користувачів з системою.

Інтерфейс користувача: інтерфейс є інтуїтивно зрозумілим та легким у використанні. Користувачі можуть швидко освоїти систему без необхідності тривалого навчання.

Мобільна підтримка: система працює на різних пристроях, включаючи мобільні телефони та планшети. Це забезпечує доступність чату для користувачів незалежно від типу пристрою.

Підтримка та обслуговування системи включають вимоги до її оновлення, моніторингу та технічної підтримки.

Оновлення: система має можливість регулярного оновлення без значних перебоїв у роботі. Оновлення включають виправлення помилок та додавання нових функцій.

Моніторинг: система має інструменти для моніторингу її стану та продуктивності в режимі реального часу, що дозволяє швидко виявляти та усувати проблеми.

Технічна підтримка: користувачі мають доступ до технічної підтримки у разі виникнення проблем або питань, що забезпечує швидке вирішення проблем і покращує загальний досвід користування системою.

Нефункціональні вимоги є критично важливими для забезпечення якості, надійності та зручності використання онлайн чату. Врахування цих вимог на етапі розробки допомогло створити стабільну, безпечну та продуктивну систему, яка задовольняє потреби користувачів та забезпечує їм позитивний досвід [11].

2.4 Інтерфейсні вимоги

Інтерфейсні вимоги визначають, як користувачі взаємодіють із системою, а також встановлюють стандарти для дизайну інтерфейсу, щоб забезпечити зручність та ефективність роботи з системою. Нижче наведено ключові інтерфейсні вимоги для онлайн чату на базі протоколу WebSocket, розробленого засобами Laravel та React. Розглянемо декілька з них:

– вимоги до інтерфейсу користувача;
– вимоги до взаємодії користувача з системою;
– вимоги до мобільної підтримки;
– вимоги до доступності;
– вимоги до інтеграції з іншими системами.

Розглянемо кожну із вимог більш детально, та наведемо прості приклади для нашого чату.

Інтерфейс користувача в онлайн чаті має бути простим, зрозумілим і зручним для використання. Наведемо основні вимоги до інтерфейсу користувача.

Чітка навігація: функції чату повинні бути доступні через просту і зрозумілу навігаційну панель або меню, користувач повинен легко знайти потрібні опції та функції.

Зрозумілі елементи керування: кнопки, поля введення тексту та інші елементи управління повинні мати зрозумілі значки або підказки, щоб користувачі могли швидко зрозуміти їх призначення.

Консистентність: дизайн інтерфейсу повинен бути однорідним на всіх сторінках та екранах чату – це допоможе користувачам швидко орієнтуватися та знаходити потрібні функції.

Адаптивний дизайн: інтерфейс має бути адаптований для різних розмірів екрану, включаючи комп’ютери, планшети та мобільні телефони – це забезпечить комфортне використання чату на будь-яких пристроях.

Швидкість реакції: система повинна реагувати миттєво на дії

користувачів, включаючи надсилання та отримання повідомлень, затримка між дією користувача та реакцією системи повинна бути мінімальною.

Чіткі повідомлення про стан системи: система повинна надавати користувачам інформацію про статус їх дій, наприклад, підтвердження про відправку повідомлення, успішне підключення до чату тощо.

Сповіщення про нові повідомлення: користувачі повинні отримувати сповіщення про нові повідомлення, навіть коли вони не активно використовують додаток – це допоможе їм не пропустити важливу інформацію.

Ефективна взаємодія користувачів з системою є ключовим аспектом успішної реалізації онлайн чату. Наведемо основні вимоги до взаємодії користувача з системою.

Легка зрозумілість структури: система повинна мати просту та зрозумілу структуру навігації, яка дозволяє користувачам легко знаходити потрібні функції та розділи.

Інтуїтивність взаємодії: елементи інтерфейсу, такі як кнопки та меню, повинні мати інтуїтивне призначення, щоб користувачі могли без зусиль зрозуміти, як ними користуватися.

Швидкий та точний пошук: система повинна мати ефективну функцію пошуку, яка дозволяє користувачам швидко та точно знаходити потрібні повідомлення, контакти та іншу інформацію.

Фільтрація результатів: пошукова функція повинна мати можливість фільтрації результатів за різними критеріями, такими як час, автор повідомлення, або ключові слова.

Повідомлення про важливі події: система повинна надсилати користувачам сповіщення про важливі події, такі як нові повідомлення, запрошення в чат, або зміни в статусах контактів.

Придатність для різних пристроїв: сповіщення повинні бути придатні для відображення на різних пристроях, включаючи комп'ютери, планшети та мобільні телефони.

Мобільна підтримка є важливим аспектом для забезпечення доступності та зручності використання чату на різних мобільних пристроях. Нижче наведено ключові вимоги до мобільної підтримки системи чату.

Резиновий макет: дизайн системи повинен бути резиновим, щоб автоматично адаптуватися до різних розмірів екранів мобільних пристроїв – **ц**е забезпечить оптимальне використання доступного простору на екрані.

Швидка реакція інтерфейсу: взаємодія з системою на мобільних пристроях повинна бути миттєвою та безперервною, час завантаження сторінок та відповідь на взаємодію користувача повинні бути мінімальними.

Оптимізовані елементи управління: елементи інтерфейсу, такі як кнопки та поля введення тексту, повинні бути достатньо великими, щоб користувачі могли комфортно ними користуватися навіть на невеликих екранах мобільних пристроїв.

Доступність жести і тач-взаємодії: система повинна підтримувати жести та тач-взаємодію, щоб користувачі могли швидко та легко взаємодіяти з додатком за допомогою дотикового екрану.

Вимоги до мобільної підтримки, взаємодії користувача з системою та інтерфейсу визначають ключові аспекти, що забезпечують зручність, ефективність та задоволення користувачів використанням чату. Врахування цих вимог дозволяє створити мобільний додаток з інтуїтивним і доступним інтерфейсом, оптимізованим для будь-яких пристроїв, та забезпечити позитивний досвід взаємодії з системою [3].

2.5 Загальна архітектура проєкту

Загальна архітектура проєкту онлайн чату на базі протоколу WebSocket, розробленого засобами Laravel та React, базується на розподіленій клієнт-серверній моделі. В цій моделі клієнтська частина (React) відповідає за взаємодію з користувачем, а серверна частина (Laravel) обробляє запити та управляє з’єднаннями WebSocket [4].

Розглянемо повну архітектурну комплектацію проєкта, та наведемо приклади функціоналу:

– клієнтська частина (Frontend);
– серверна частина (Backend);
– база даних;
– система обміну повідомленнями.

У клієнтській частині системи використовується низка інструментів і технологій для створення зручного та ефективного інтерфейсу користувача. Основними компонентами цієї частини є React, WebSocket клієнт та Redux.

React відповідає за рендеринг інтерфейсу користувача та обробку взаємодії з користувачем. Завдяки використанню React, можна легко створювати компоненти і перевикористовувати їх, що робить розробку інтерфейсу ефективною та зручною [6].

WebSocket клієнт забезпечує реальну взаємодію в реальному часі між клієнтом та сервером через WebSocket-з’єднання. Це дозволяє отримувати та надсилати повідомлення миттєво без необхідності перезавантаження сторінки.

Redux використовується для управління станом додатка, зберігання даних про користувачів, повідомлення та статуси з’єднань. Він дозволяє централізовано керувати станом додатка і забезпечує простий та передбачуваний спосіб управління даними.

За допомогою цих інструментів і технологій клієнтська частина системи може ефективно взаємодіяти з сервером, забезпечуючи користувачам зручний та інтуїтивно зрозумілий інтерфейс [8].

У серверній частині системи використовуються інструменти та технології для обробки запитів користувачів, аутентифікації, авторизації та управління всією логікою додатка. Основними компонентами цієї частини є Laravel та WebSocket сервер на базі Ratchet.

Laravel використовується для обробки HTTP-запитів, аутентифікації, авторизації та управління користувачами. За допомогою Laravel можна швидко створити потужний та ефективний backend, який забезпечить безпеку та надійність системи [4].

Ratchet є WebSocket сервером, який забезпечує управління WebSocket-з'єднаннями та обробку повідомлень в реальному часі. Він дозволяє підтримувати постійне з'єднання між клієнтом та сервером, що дозволяє отримувати та надсилати повідомлення безперервно.

Завдяки використанню Laravel та Ratchet серверна частина додатку може ефективно керувати всіма аспектами взаємодії з клієнтами, забезпечуючи швидку та надійну роботу системи.

База даних виконує ключову роль у зберіганні та організації інформації, необхідної для функціонування системи. У цій системі використовується MySQL для зберігання різноманітних даних, таких як дані користувачів, повідомлень, історії чатів та іншої важливої інформації.

Розглянемо функціональність MySQL:

– зберігання даних користувачів: база даних зберігає дані про користувачів, такі як інформація про облікові записи, персональні налаштування тощо;
– зберігання повідомлень: повідомлення, відправлені та отримані користувачами, також зберігаються в базі даних для подальшого доступу та аналізу;
– історія чатів: інформація про історію чатів, включаючи дати та часи повідомлень, учасників чату тощо, зберігається для забезпечення можливості перегляду попередніх обговорень.

MySQL дозволяє зберігати дані ефективно та безпечно, забезпечуючи швидкий доступ до необхідної інформації для подальшого використання в системі.

На рисунку 2.3 наведена діаграмі класів та їх взаємозв'язки в рамках проєкту онлайн чату. Кожен клас відповідає певному компоненту системи і має свої властивості та методи для взаємодії з іншими частинами системи.

Діаграма допомагає візуалізувати структуру системи та з'ясувати взаємозв'язки між класами, що полегшує розуміння архітектури проєкту.

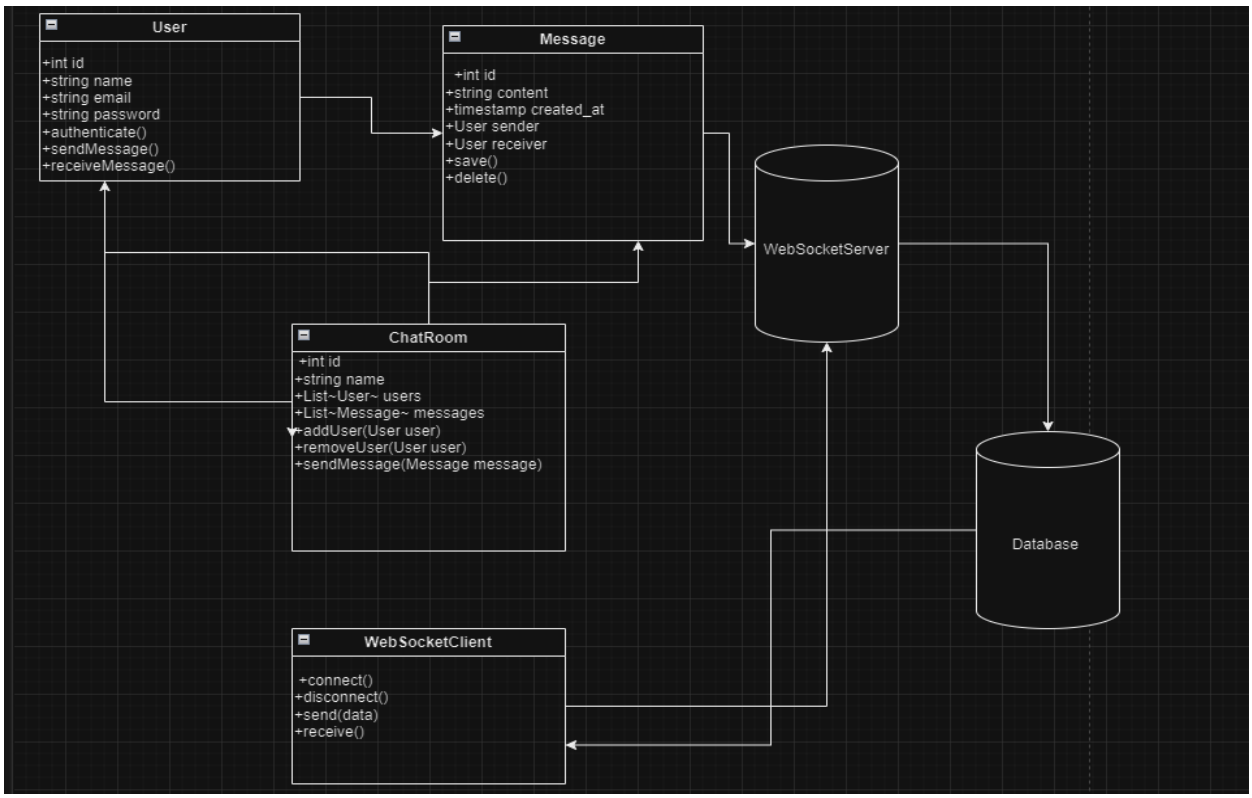


Рисунок 2.3 – Діаграма класів

Опишемо кожний компонент нашого проєкту.

User – клас, що представляє користувача системи. Містить інформацію про користувача, методи для аутентифікації, відправки та отримання повідомлень.

Message – клас, що представляє повідомлення. Містить інформацію про зміст повідомлення, час створення, відправника та отримувача. Містить методи для збереження та видалення повідомлень.

ChatRoom – клас, що представляє чат-кімнату. Містить інформацію про учасників чату та повідомлення в чаті. Містить методи для додавання/видалення користувачів та відправки повідомлень.

WebSocketClient – клас, що представляє WebSocket клієнт. Містить методи для підключення, відключення, відправки та отримання даних через WebSocket-з’єднання.

WebSocketServer – клас, що представляє WebSocket сервер. Містить методи для запуску, зупинки сервера, обробки підключень та повідомлень від клієнтів.

Database – клас, що представляє базу даних. Містить методи для підключення до бази даних, виконання запитів, вставки, оновлення та видалення даних.

Загальна архітектура проєкту онлайн чату забезпечує розподілену клієнт-серверну модель з використанням сучасних технологій та підходів. Використання Laravel та React дозволяє створити ефективний та масштабований додаток, який відповідає всім нефункціональним вимогам і забезпечує користувачам швидкий та зручний інтерфейс для спілкування в реальному часі [6].

3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

3.1 Налаштування та використання WebSocket у Laravel

Спочатку ми встановлюємо пакет Laravel WebSockets, який дозволяє запускати власний WebSocket сервер замість використання сторонніх сервісів типу Pusher.

Laravel WebSockets – це пакет для Laravel, який дозволяє створювати і запускати власний WebSocket сервер без необхідності використання сторонніх сервісів, таких як Pusher. WebSocket – це протокол зв’язку між клієнтом і сервером, який дозволяє встановити постійне з’єднання і передавати дані в реальному часі. Розглянемо основні функції Laravel WebSockets:

– власний WebSocket сервер: пакет дозволяє запустити WebSocket сервер без потреби в сторонніх сервісах – це забезпечує більшу контроль і гнучкість у розробці реального часу в додатках;
– підтримка каналів та подій: ви можете використовувати канали та події для організації та взаємодії з WebSocket з’єднаннями;
– інтеграція з Laravel Echo: за допомогою Laravel Echo ви можете легко взаємодіяти з WebSocket сервером у вашому фронтенді;
– моніторинг і статистика: пакет надає можливість моніторити активні з’єднання WebSocket і статистику використання.

На рисунку 3.1 зображено виконання команди для встановлення пакету.

```
composer require beyondcode/laravel-websockets
```

Рисунок 3.1 – Встановлення пакету

Після цього публікуємо конфігураційний файл, щоб мати можливість налаштовувати пакет під свої потреби. На рисунку 3.2 зображено виконання команди для публікації конфігурації файлів.

```
php artisan vendor:publish --provider="BeyondCode\LaravelWebSockets\WebSocketsServiceProvider"
```

Рисунок 3.2 – Налаштування конфігурації файлів

Тепер давайте налаштуємо Pusher, оскільки Laravel WebSockets використовує його як драйвер для забезпечення роботи WebSocket сервера. Для цього нам потрібно додати необхідні змінні у файл .env, що дозволить Laravel взаємодіяти з Pusher API. Це включає параметри, такі як PUSHER_APP_ID, PUSHER_APP_KEY, PUSHER_APP_SECRET і PUSHER_APP_CLUSTER, які надаються Pusher при реєстрації нової програми на їхньому вебсайті. На рисунку 3.3 зображено приклад налаштування Pusher.

```
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=laravel
DB_USERNAME=root
DB_PASSWORD=
```

Рисунок 3.3 – Налаштування Pusher

Також необхідно налаштувати файл config/broadcasting.php, щоб Laravel знав, що ми використовуємо Pusher як наш основний драйвер для трансляції подій.

У цьому файлі ми можемо вказати всі необхідні параметри, які дозволять Laravel взаємодіяти з Pusher API. Це включає налаштування ключа, секрету, ідентифікатора програми і параметрів кластера. Ці параметри забезпечують безпечне і ефективне підключення до Pusher, дозволяючи нашому додатку використовувати можливості WebSocket для трансляції подій в реальному часі.

На рисунку 3.4 наведено приклад того, як налаштувати розділ connections у файлі config/broadcasting.php, щоб він використовував Pusher як драйвер для трансляцій.

```
    */
    'apps' => [
        [
            'id' => env('PUSHER_APP_ID'),
            'name' => env('APP_NAME'),
            'key' => env('PUSHER_APP_KEY'),
            'secret' => env('PUSHER_APP_SECRET'),
            'enable_client_messages' => false,
            'enable_statistics' => true,
        ],
    ],
```

Рисунок 3.4 – Налаштування connections

Створюємо подію і налаштовуємо її для трансляції через WebSocket, що дозволить нам відправляти повідомлення в реальному часі нашим користувачам. Laravel надає простий і зручний спосіб створення подій та їх трансляції, використовуючи систему подій та трансляцій.

За допомогою команди “php artisan make:event MyEvent” створюємо файл MyEvent.php у директорії app/Events. Тепер відкриємо цей файл і налаштуємо його так, щоб він міг транслювати події. Нам потрібно додати властивості, які ми хочемо передати, а також налаштувати методи для трансляції.

На рисунку 3.5 наведено приклад реалізації налаштування транслювання подій.

У цьому прикладі ми створили подію MyEvent, яка містить повідомлення, яке буде передаватися через WebSocket. Ми використовуємо інтерфейс ShouldBroadcast, щоб вказати, що ця подія повинна бути транслювана. Метод broadcastOn повертає новий екземпляр класу Channel, який вказує на канал, на який буде транслюватися подія. У цьому випадку ми транслюємо на публічний канал my-channel. Це дозволяє нам легко транслювати події в реальному часі, які будуть отримуватися всіма підписаними на відповідний канал клієнтами.

```php
<?php

namespace App\Events;

use Illuminate\Broadcasting\Channel;
use Illuminate\Broadcasting\InteractsWithSockets;
use Illuminate\Broadcasting\PresenceChannel;
use Illuminate\Broadcasting\PrivateChannel;
use Illuminate\Contracts\Broadcasting\ShouldBroadcast;
use Illuminate\Foundation\Events\Dispatchable;
use Illuminate\Queue\SerializesModels;

class MyEvent implements ShouldBroadcast
{
    use Dispatchable, InteractsWithSockets, SerializesModels;

    public $message;

    /**
     * Create a new event instance.
     *
     * @return void
     */
    public function __construct($message)
    {
        $this->message = $message;
    }

    /**
     * Get the channels the event should broadcast on.
     *
     * @return Channel|array
     */
    public function broadcastOn()
    {
        return new Channel('my-channel');
    }
}

```

Рисунок 3.5 – Налаштування трансляції подій

На стороні клієнта, за допомогою Laravel Echo і Pusher, ми можемо слухати ці події та реагувати на них. Наприклад на рисунку 3.6 у нас зображено JavaScript файл, який використовується на стороні клієнта.

Таким чином, ми створюємо подію MyEvent, яка передає повідомлення через WebSocket. Ми також налаштовуємо клієнтську сторону для прослуховування цієї події і відображення повідомлення у консолі. Це

дозволяє створювати інтерактивні додатки з оновленням в реальному часі, що значно покращує користувацький досвід.

```
export default router;

import Echo from 'laravel-echo';
window.Pusher = require('pusher-js');

window.Echo = new Echo({
    broadcaster: 'pusher',
    key: process.env.MIX_PUSHER_APP_KEY,
    cluster: process.env.MIX_PUSHER_APP_CLUSTER,
    wsHost: window.location.hostname,
    wsPort: 6001,
    forceTLS: false,
    disableStats: true,
});

window.Echo.channel('my-channel')
    .listen('MyEvent', (e) => {
        console.log(e.message);
    });
```

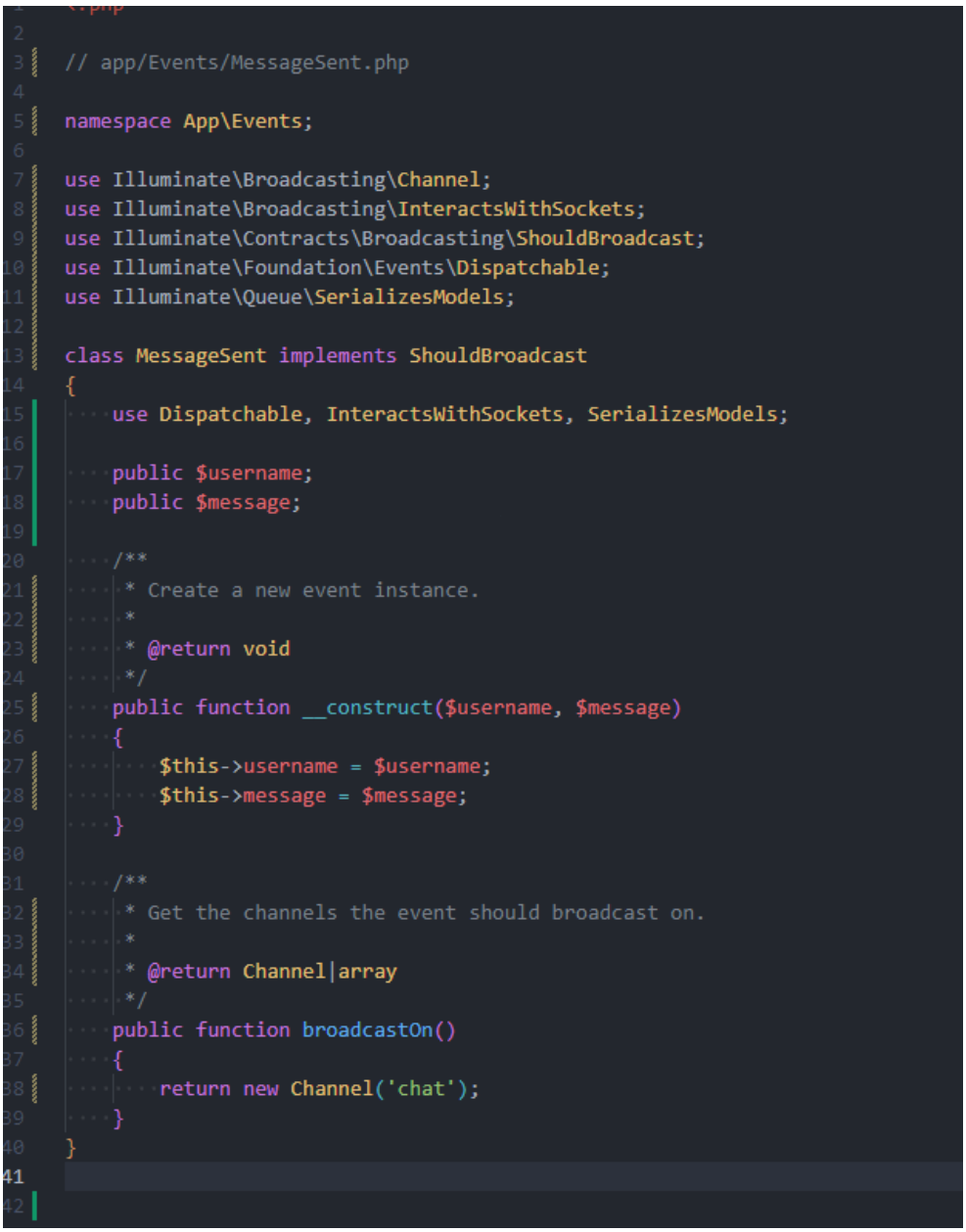
Рисунок 3.6 – Приклад прослуховування подій

3.2 Реалізація серверної частини

У цьому розділі ми розглянемо реалізацію серверної частини для онлайн чату на базі протоколу WebSocket засобами Laravel. Ми будемо використовувати пакет Laravel WebSockets, який дозволяє налаштувати WebSocket сервер у вашому додатку Laravel. Це дозволяє надсилати та отримувати повідомлення в режимі реального часу без необхідності в додатковому сервері.

Першим кроком у реалізації нашого чату буде створення події, яка буде транслювати повідомлення в чат. Подія буде використовуватися для відправки повідомлення від сервера до всіх підписаних клієнтів.

Клас MessageSent є подією, яка реалізує контракт ShouldBroadcast, що дозволяє транслювати цю подію через WebSocket на всі підписані канали (див. рис. 3.7).



```php
// app/Events/MessageSent.php

namespace App\Events;

use Illuminate\Broadcasting\Channel;
use Illuminate\Broadcasting\InteractsWithSockets;
use Illuminate\Contracts\Broadcasting\ShouldBroadcast;
use Illuminate\Foundation\Events\Dispatchable;
use Illuminate\Queue\SerializesModels;

class MessageSent implements ShouldBroadcast
{
    use Dispatchable, InteractsWithSockets, SerializesModels;

    public $username;
    public $message;

    /**
     * Create a new event instance.
     *
     * @return void
     */
    public function __construct($username, $message)
    {
        $this->username = $username;
        $this->message = $message;
    }

    /**
     * Get the channels the event should broadcast on.
     *
     * @return Channel|array
     */
    public function broadcastOn()
    {
        return new Channel('chat');
    }
}
```

Рисунок 3.7 – Клас MessageSent

Опис функціоналу коду:

– Dispatchable, InteractsWithSockets, SerializesModels: трейти надають класу необхідні функціональність для того, щоб його можна було відправляти через чергу, взаємодіяти з сокетами та серіалізувати моделі;

– конструктор (__construct($username, $message)): приймає два параметри – $username (ім’я користувача) і $message (текстове повідомлення) – ці дані встановлюються як публічні властивості класу, що дозволяє доступ до них при трансляції події;

– метод broadcastOn(): метод повертає об’єкт Channel, який вказує на канал, через який буде транслюватися подія – у нашому випадку, подія MessageSent буде транслюватися на канал chat.

Далі створимо контролер, який буде обробляти повідомлення від користувачів і викликати подію MessageSent. Контролер буде приймати повідомлення через HTTP запит, створювати подію і відправляти її через WebSocket.

Коли створено екземпляр класу MessageSent і відправлено через event() (наприклад, в контролері або сервісі), Laravel автоматично транслює цю подію через WebSocket. На рисунку 3.8 наведено приклад автоматичного транслювання подій через контролери.

```php
use App\Events\MessageSent;
use Illuminate\Http\Request;
use App\Http\Controllers\Controller;

class ChatController extends Controller
{
    public function sendMessage(Request $request)
    {
        $username = $request->input('username');
        $message = $request->input('message');

        event(new MessageSent($username, $message));

        return response()->json(['status' => 'Message sent!']);
    }
}
```

Рисунок 3.8 – Приклад відправлення подій на MessageSent

Контролер містить метод sendMessage, який приймає запит, витягує з нього ім’я користувача та повідомлення, і викликає подію MessageSent. Це дозволяє відправляти повідомлення від користувача до всіх інших користувачів в режимі реального часу.

Міграції в Laravel дозволяють вам легко керувати структурою вашої бази даних і зберігати її разом з вашим додатком у вигляді коду. Вони

використовуються для створення, зміни та видалення таблиць та інших схем даних. Для реалізації чату, вам потрібно буде створити таблицю для збереження повідомлень чату. На рисунку 3.9 наведено приклад міграції для створення таблиці messages у базі даних.

```php
use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

class CreateMessagesTable extends Migration
{
    public function up()
    {
        Schema::create('messages', function (Blueprint $table) {
            $table->id();
            $table->string('username');
            $table->text('message');
            $table->timestamps();
        });
    }

    public function down()
    {
        Schema::dropIfExists('messages');
    }
}
```

Рисунок 3.9 – Приклад міграції

Даний код міграції для Laravel створює таблицю messages у базі даних з наступними стовпцями:

– id: це автоматично зростаючий (auto-increment) унікальний ідентифікатор запису в таблиці – використовується як основний ключ;
– username: стовпець для зберігання імені користувача, який відправив повідомлення – встановлений тип string, що означає, що може містити текстові значення заданої довжини;
– message: стовпець для зберігання текстового повідомлення чату – встановлений тип text, що дозволяє зберігати більші обсяги тексту;

– timestamps: Laravel автоматично додає два стовпці created_at та updated_at, які зберігають час створення та останнього оновлення запису в таблиці, відповідно.

Опишемо стандартну роботу коду.

Up() метод: в методі up() викликається Schema::create(), що вказує Laravel створити нову таблицю з назвою messages. В тілі цієї функції визначаються всі необхідні стовпці для таблиці за допомогою $table->тип_стовпця(‘назва_стовпця’). У даному випадку визначається тип для трьох стовпців: id, username та message.

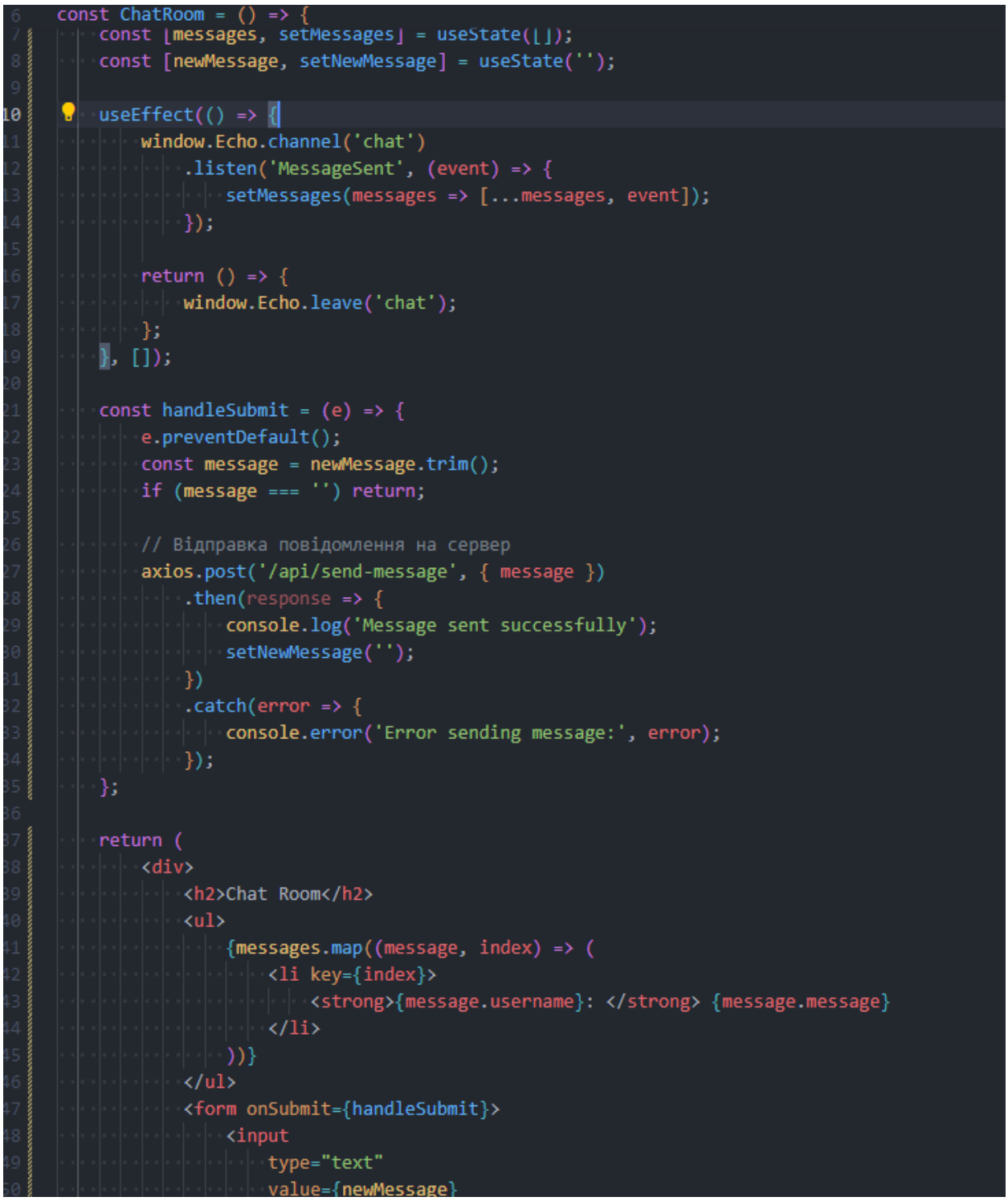
Down() метод: в методі down(), який викликається при відкаті міграції (наприклад, використовуючи команду php artisan migrate:rollback), викликається Schema::dropIfExists(), що вказує Laravel видалити таблицю messages, якщо вона існує.

Міграція дозволяє легко створити таблицю для зберігання повідомлень чату в вашому додатку Laravel. При кожному запуску міграції командою php artisan migrate Laravel автоматично створить таблицю messages у вашій базі даних або оновить її структуру, якщо вона вже існує, відповідно до описаної схеми.

3.3 Реалізація клієнтської частини

Реалізація клієнтської частини для онлайн чату на базі протоколу WebSocket з використанням React включає налаштування підписки на канал WebSocket та відображення отриманих повідомлень у реальному часі. Давайте розглянемо кроки для реалізації цієї частини.

На рисунку 3.10 зображено компонент ChatRoom.js який реалізує простий онлайн чат на React з використанням Laravel Echo для підписки на WebSocket канал і отримання повідомлень в реальному часі.



```
const ChatRoom = () => {
    const [messages, setMessages] = useState([]);
    const [newMessage, setNewMessage] = useState('');

    useEffect(() => {
        window.Echo.channel('chat')
            .listen('MessageSent', (event) => {
                setMessages(messages => [...messages, event]);
            });

        return () => {
            window.Echo.leave('chat');
        };
    }, []);

    const handleSubmit = (e) => {
        e.preventDefault();
        const message = newMessage.trim();
        if (message === '') return;

        // Відправка повідомлення на сервер
        axios.post('/api/send-message', { message })
            .then(response => {
                console.log('Message sent successfully');
                setNewMessage('');
            })
            .catch(error => {
                console.error('Error sending message:', error);
            });
    };

    return (
        <div>
            <h2>Chat Room</h2>
            <ul>
                {messages.map((message, index) => (
                    <li key={index}>
                        <strong>{message.username}: </strong> {message.message}
                    </li>
                ))}
            </ul>
            <form onSubmit={handleSubmit}>
                <input
                    type="text"
                    value={newMessage}
```

Рисунок 3.10 – Компонент ChatRoom.js

Основний опис компонента **ChatRoom**.

Стан компонента:

– messages: масив для зберігання отриманих повідомлень;

– newMessage: рядок для зберігання тексту нового повідомлення, яке користувач вводить у формі.

Ефект useEffect:

- при завантаженні компонента відбувається підписка на WebSocket канал chat за допомогою Echo.channel(‘chat’);
- встановлюється обробник подій listen, який слухає події типу MessageSent і додає нове повідомлення до стану messages за допомогою функції setMessages;
- функція повертає відписку від каналу chat під час розмонтування компонента за допомогою Echo.leave(‘chat’).

Функція handleSubmit:

- обробник події відправки форми. Вона викликається при натисканні кнопки “Send”;
- виконує перевірку, чи не є порожнім рядок newMessage;
- якщо повідомлення не порожнє, відбувається відправка повідомлення на сервер за допомогою Axios через axios.post(‘/api/send-message’, { message });
- у випадку успішної відправки, виводиться повідомлення “Message sent successfully” у консоль і стан newMessage очищується за допомогою setNewMessage(‘‘);
- в разі помилки виводиться повідомлення про помилку у консоль через console.error.

Відображення вікна чату:

- виводить список ul з усіма збереженими повідомленнями у структурі li;
- виводиться форма для введення нового повідомлення з полем вводу input і кнопкою “Send”.

Цей компонент демонструє простий інтерфейс чату з можливістю відправки повідомлень у реальному часі, використовуючи Laravel Echo для WebSocket і Axios для взаємодії з сервером.

Компонент MessageForm відповідає за введення нових повідомлень у чаті і їхню відправку на сервер за допомогою Axios (див. рис. 3.11).

```jsx
import React, { useState } from 'react';
import axios from 'axios';

const MessageForm = ({ onMessageSent }) => {
    const [newMessage, setNewMessage] = useState('');

    const handleSubmit = (e) => {
        e.preventDefault();
        const message = newMessage.trim();
        if (message === '') return;
        axios.post('/api/send-message', { message })
            .then(response => {
                console.log('Message sent successfully');
                setNewMessage('');
                onMessageSent();
            })
            .catch(error => {
                console.error('Error sending message:', error);
            });
    };

    return (
        <form onSubmit={handleSubmit}>
            <input
                type="text"
                value={newMessage}
                onChange={(e) => setNewMessage(e.target.value)}
                placeholder="Enter your message"
                required
            />
            <button type="submit">Send</button>
        </form>
    );
};

export default MessageForm;
```

Рисунок 3.11 – Компонент MessageForm

Розглянемо більш детально роботу компонента MessageForm з точки зору його функціональності та впливу на користувача у чат-додатку:

– MessageForm використовується для введення тексту повідомлення та його відправки на сервер;
– при відправці форми handleSubmit викликає Axios для POST запиту /api/send-message, що включає текст повідомлення;
– після успішної відправки очищує поле введення та викликає onMessageSent, який оновлює стан батьківського компонента чату.

Після успішної відправки повідомлення, поле введення newMessage очищається, щоб користувач міг почати вводити нове повідомлення. Також виконується переданий пропс onMessageSent(), що дозволяє оновити стан батьківського компонента, наприклад, оновити список повідомлень із новими даними.

Компонент MessageList призначений для відображення списку повідомлень у чаті на основі наданих даних з батьківського компонента (див. рис. 3.12).

```
// src/components/MessageList.js
import React from 'react';

const MessageList = ({ messages }) => {
    return (
        <ul>
            {messages.map((message, index) => (
                <li key={index}>
                    <strong>{message.username}: </strong> {message.message}
                </li>
            ))}
        </ul>
    );
};

export default MessageList;
```

Рисунок 3.12 – Компонент MessageList

Компонент MessageList відповідає за відображення списку повідомлень у чаті. Він очікує на вхід масив messages, кожен елемент якого містить властивості username і message. Для створення маркованого списку використовується тег <ul>, а кожне повідомлення з масиву messages відображається у вигляді елемента <li>. Ім’я користувача username підсилюється тегом , а саме повідомлення message відображається без змін. Цей компонент може використовуватися в батьківському компоненті для динамічного оновлення списку повідомлень у реальному часі під час взаємодії користувачів у чаті.

Компонент Chat відповідає за основну структуру і логіку чат-додатку, включаючи відображення повідомлень, форму для відправлення повідомлень і інтеграцію з WebSocket для отримання в реальному часі нових повідомлень (див. рис. 3.13).

```
import React, { useEffect, useState } from 'react';
import axios from 'axios';
import Echo from 'laravel-echo';
import MessageForm from './MessageForm';
import MessageList from './MessageList';

const Chat = () => {
    const [messages, setMessages] = useState([]);

    useEffect(() => {
        const echo = new Echo({
            broadcaster: 'pusher',
            key: process.env.REACT_APP_PUSHER_APP_KEY,
            cluster: process.env.REACT_APP_PUSHER_APP_CLUSTER,
            encrypted: true,
        });

        echo.channel('chat')
            .listen('MessageSent', (event) => {
                setMessages(messages => [...messages, event]);
            });

        return () => {
            echo.leave('chat');
        };
    }, []);

    const handleNewMessage = () => {
        axios.get('/api/messages')
            .then(response => {
                setMessages(response.data);
            })
            .catch(error => {
                console.error('Error fetching messages:', error);
            });
    };

    return (
        <div>
            <h2>Chat</h2>
            <MessageList messages={messages} />
            <MessageForm onMessageSent={handleNewMessage} />
        </div>
    );
};
```

Рисунок 3.13 – Компонент Chat

Опис роботи компоненту **Chat**.

Стан messages:

– використовує useState для зберігання списку повідомлень, початково стан ініціалізується пустим масивом [].

Ефект useEffect:

– при завантаженні компонента встановлює підписку на канал chat за допомогою Echo з параметрами Pusher, які беруться з середовищних змінних;
– при отриманні нового повідомлення через канал chat оновлює стан messages, додаючи нове повідомлення до поточного списку.

Функція handleNewMessage:

– відповідає за оновлення списку повідомлень після відправки нового повідомлення користувачем;
– виконує запит до сервера за допомогою Axios методом GET на /api/messages;
– після успішного отримання відповіді оновлює стан messages з даними отриманими з сервера;
– обробляє помилки, якщо такі виникають, виводячи їх у консоль.

Використання дочірніх компонентів:

– відображає компонент MessageList, який приймає список повідомлень messages як пропс і відображає їх у вигляді списку;
– відображає компонент MessageForm, якому передає колбек-функцію handleNewMessage через пропс onMessageSent – цей компонент відповідає за форму для відправлення нових повідомлень.

Цей компонент Chat об’єднує в собі всі необхідні елементи для функціонування чат-додатку з можливістю реального часу за допомогою WebSocket та взаємодії з сервером для отримання та відправлення повідомлень.

3.4 Тестування та відлагодження

У цьому розділі буде розглянуто процес тестування та відлагодження чат-системи, розробленої на базі протоколу WebSocket засобами Laravel та

React. Основна мета тестування – забезпечити стабільну та безперебійну роботу системи, а також виявити та виправити можливі помилки до запуску у продуктивне середовище.

Теоретичні аспекти тестування та відлагодження.

Види тестування:

– модульне тестування: перевірка окремих компонентів або модулів системи на коректність роботи, для цього використовуються фреймворки тестування, такі як PHPUnit для Laravel та Jest для React;
– інтеграційне тестування: перевірка взаємодії між різними модулями та компонентами системи – це важливо для переконання в тому, що всі частини системи працюють гармонійно;
– системне тестування: комплексна перевірка всієї системи на відповідність вимогам, включає тестування функціональності, безпеки, продуктивності тощо;
– регресійне тестування: перевірка системи після внесення змін або виправлення помилок для забезпечення відсутності нових помилок та збереження працездатності.

Інструменти та методи тестування:

– PHPUnit: використовується для модульного тестування компонентів Laravel – дозволяє створювати тести для контролерів, моделей та інших частин серверної логіки;
– Jest: використовується для тестування компонентів React – дозволяє перевіряти роботу компонентів, перевіряти рендеринг та взаємодії з користувачем;
– Selenium: використовується для автоматизованого тестування вебінтерфейсів – дозволяє писати сценарії, які емулюють дії користувача, та перевіряти коректність роботи системи в цілому;
– Postman: використовується для тестування API – дозволяє створювати запити до серверу та перевіряти коректність відповіді.

Методики відлагодження:

– логування: запис важливих подій та помилок у лог-файли для подальшого аналізу – Laravel надає потужні засоби для логування подій;
– налагоджувачі: використання налагоджувачів (debugger) для пошуку та виправлення помилок у коді – для Laravel це може бути Xdebug, а для React – React Developer Tools;
– статичний аналіз коду: використання інструментів для статичного аналізу коду, таких як ESLint для JavaScript/React та PHPStan для PHP/Laravel – це допомагає знайти потенційні проблеми ще до виконання коду;
– контроль версій: використання систем контролю версій (наприклад, Git) для відслідковування змін у коді та можливості відкотитися до стабільної версії у разі виникнення проблем.

На рисунку 3.14 показано, як реалізовано основний інтерфейс обміну повідомленнями. Ліва частина екрана містить список контактів з їх аватарами та датами останніх повідомлень, що допомагає користувачам легко знаходити потрібну розмову.

У правій частині відображається поточний чат з вибраним контактом, де користувач може бачити і відправляти повідомлення. Користувачі можуть обмінюватися інформацією в реальному часі завдяки використанню WebSocket, що забезпечує миттєву передачу даних.

На рисунку 3.15 представлена сторінка реєстрації користувача. Цей інтерфейс дозволяє новим користувачам легко створити обліковий запис для доступу до чат-системи.

Користувачі вводять свої особисті дані в відповідні поля, включаючи ім’я, прізвище, електронну адресу та пароль. Також передбачена можливість додавання аватарки, що покращує візуальне сприйняття профілю. Після заповнення всіх полів, натискання кнопки реєстрації ініціює створення нового облікового запису, що дозволяє користувачу приєднатися до чат-системи.

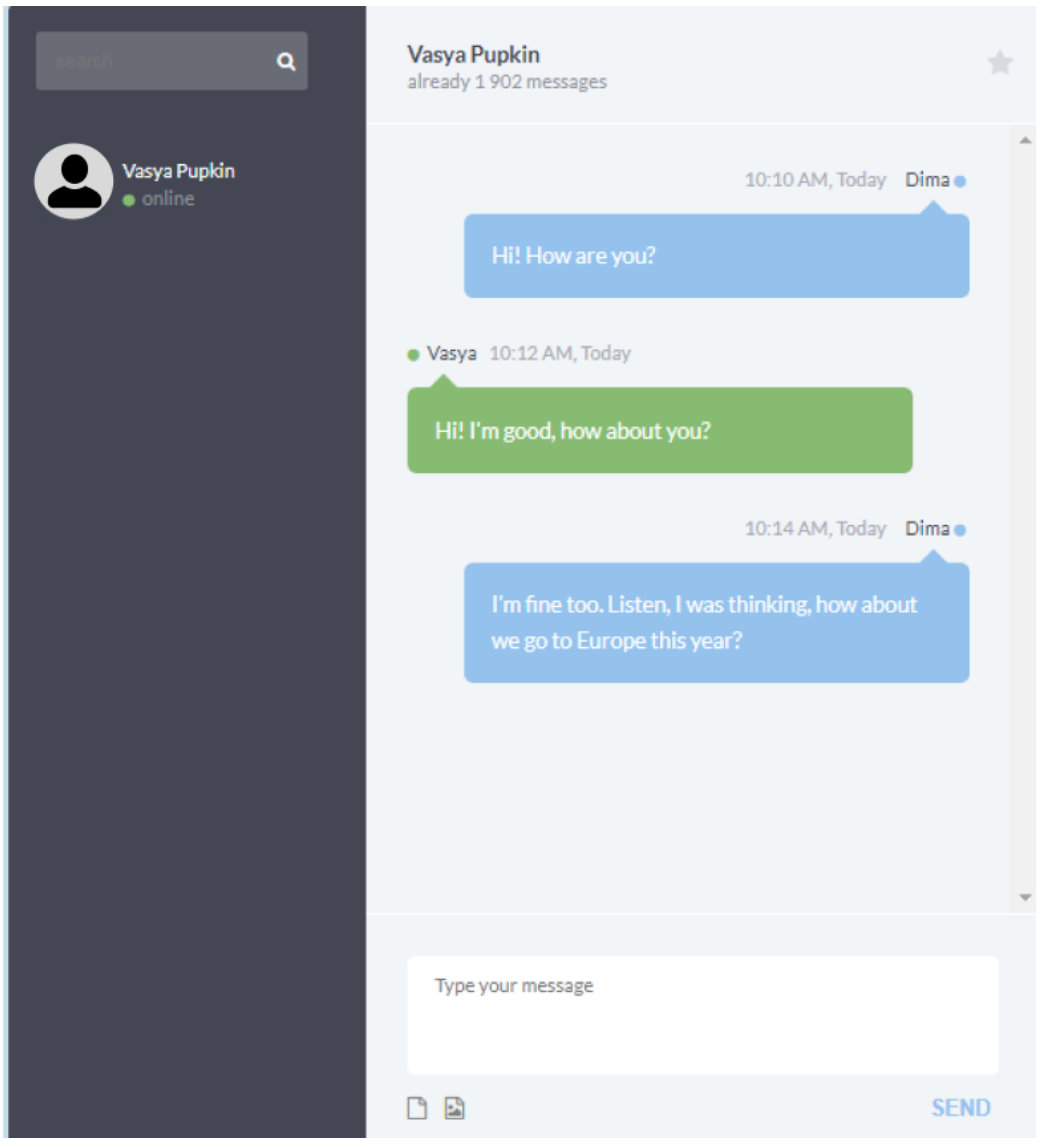


Рисунок 3.14 – Інтерфейс обміну повідомленнями

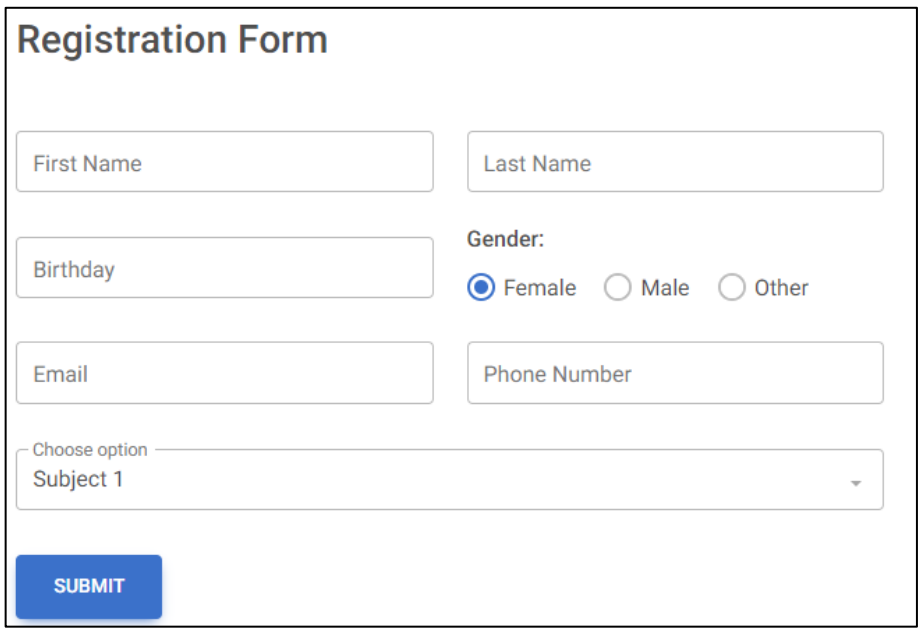


Рисунок 3.15 – Реєстрації користувача

Таким чином, реалізація інтерфейсної частини нашого чату забезпечує зручний та функціональний спосіб спілкування між користувачами. Використання протоколу WebSocket та засобів Laravel та React дозволяє забезпечити високу продуктивність та надійність роботи чат-системи.

3.5 Результати та їх обговорення

У цьому розділі розглянемо результати, отримані під час розробки онлайн-чату на базі протоколу WebSocket засобами Laravel та React, а також проведемо їх детальний аналіз.

Розглянемо функціональність нашої системи:

– реєстрація користувачів: була успішно реалізована система реєстрації нових користувачів з можливістю додавання аватарки та введення особистих даних;
– авторизація користувачів: забезпечено надійну систему авторизації користувачів з перевіркою автентичності;
– обмін повідомленнями в реальному часі: використання протоколу WebSocket дозволило реалізувати обмін повідомленнями в реальному часі між користувачами;
– сторінка профілю користувача: реалізована сторінка профілю, де користувач може переглядати та редагувати свої дані;
– список контактів: зручний інтерфейс для перегляду списку контактів та вибору поточного чату.

Також треба розглянути основні системи тестування та відлагодження:

– проведено модульне, інтеграційне та системне тестування всіх компонентів чат-системи;
– використано інструменти для автоматизованого тестування, такі як PHPUnit для Laravel та Jest для React;
– реалізовано логування та налагодження для виявлення та виправлення помилок.

Ефективність обміну повідомленнями в нашому чаті:

– використання WebSocket дозволило досягти миттєвого обміну повідомленнями між користувачами без значних затримок;
– продуктивність системи залишалась на високому рівні навіть при великій кількості одночасних користувачів завдяки асинхронній природі WebSocket.

Також найціннішим досягнення будь-якої системи це зручність користувацького інтерфейсу:

– інтуїтивно зрозумілий дизайн інтерфейсу дозволив користувачам легко орієнтуватися в системі та швидко знаходити необхідну інформацію;
– можливість додавання аватарок та перегляду статусу контактів значно покращила взаємодію між користувачами.

Не менш важливим є надійність та безпека:

– система реєстрації та авторизації забезпечила високий рівень безпеки для користувачів;
– використання надійних методів шифрування даних дозволило захистити персональні дані користувачів та їх повідомлення.

Для розробника також є важливим тестування та якість коду:

– проведене тестування забезпечило високу якість коду та зменшило кількість помилок в продуктивному середовищі;
– використання статичного аналізу коду допомогло виявити потенційні проблеми ще до їх прояву під час виконання програми.

Загалом, розробка чат-системи на базі протоколу WebSocket засобами Laravel та React була успішною. Основні функціональні можливості були реалізовані та протестовані, забезпечуючи високу продуктивність та надійність системи. Використання сучасних технологій та методів розробки дозволило створити зручний та функціональний інтерфейс для користувачів.

Основними перевагами нашої системи є:

– миттєвий обмін повідомленнями без затримок;

– інтуїтивно зрозумілий та зручний користувацький інтерфейс;
– високий рівень безпеки та захисту персональних даних.

Водночас, деякі аспекти можуть потребувати подальшого вдосконалення, зокрема оптимізація продуктивності при надмірно високому навантаженні та розширення функціональності системи відповідно до зростаючих потреб користувачів.

Таким чином, наш проєкт демонструє успішну реалізацію сучасної чат-системи з використанням протоколу WebSocket, що забезпечує високоякісне та надійне спілкування між користувачами.

ВИСНОВКИ

У процесі аналізу вимог та розробки онлайн-чату на базі протоколу WebSocket засобами Laravel та React було встановлено, що розроблений продукт відповідає поставленим завданням та вимогам замовника.

Найбільше вражає те, що система забезпечує миттєвий обмін повідомленнями без затримок, що є критично важливим для користувачів. Крім того, додавання функцій реєстрації, авторизації та профілю користувача значно розширює можливості системи, роблячи її більш інтерактивною та зручною для використання.

Важливим етапом було ретельне вивчення та аналіз технічного завдання від потенційного замовника, що стосується створення чат-системи. Застосовано технологічний стек, який виявився дієвим для реалізації поставлених завдань. Висвітлено ключові моменти, такі як обробка повідомлень у реальному часі, управління контактами та профілями користувачів, а також інтеграція з WebSocket для забезпечення стабільної роботи системи.

Зазначаючи, що використовувались не лише теоретичні знання, але й практичні навички, кожен етап розробки було висвітлено докладно. Реалізація модульного та інтеграційного тестування, використання статичного аналізу коду та конкретні приклади коду в деталях демонструють гарні приклади забезпечення якості та безпеки програмного забезпечення.

Оглядаючи роботу в цілому, можна зробити висновок, що створена чат-система відповідає вимогам, викладеним на етапі аналізу, і являє собою функціональний продукт, готовий задовольнити потреби користувачів.

ПЕРЕЛІК ПОСИЛАНЬ

1. Офіційна документація Laravel WebSockets для налаштування серверної частини. *Laravel WebSockets Documentation*. URL: https://beyondco.de/docs/laravel-websockets (дата звернення: 23.03.2024).
2. Документація Pusher, яка використовується як драйвер для Laravel WebSockets. *Pusher Channels Documentation*. URL: https://pusher.com/docs/channels (дата звернення: 29.03.2024).
3. Офіційна документація React для клієнтської частини. *React Documentation*. URL: https://reactjs.org/docs/getting-started.html (дата звернення: 09.03.2024).
4. Документація WebSocket API для реалізації бічної взаємодії між сервером і клієнтом. *WebSocket API Documentation*. URL: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API (дата звернення: 15.03.2024).
5. Документація Laravel Echo для роботи з WebSocket в Laravel. *Laravel Echo Documentation*. URL: https://laravel.com/docs/8.x/broadcasting (дата звернення: 20.03.2024).
6. Документація Axios для здійснення HTTP запитів з клієнта до сервера. *Axios Documentation*. URL: https://axios-http.com/docs/intro (дата звернення: 30.03.2024).
7. Огляд протоколу WebSocket і його можливостей. *WebSocket Protocol Overview*. URL: https://developer.mozilla.org/en-US/docs/Web/API/WebSocket (дата звернення: 05.04.2024).
8. Документація Knex.js для побудови SQL запитів в Laravel. *Knex.js Documentation*. URL: https://knexjs.org/ (дата звернення: 12.04.2024).
9. Документація Objection.js для роботи з базами даних у Laravel. *Objection.js Documentation*. URL: https://vincit.github.io/objection.js/guide/ (дата звернення: 17.04.2024).

10. Node.js v21.5.0 documentation. URL: https://nodejs.org/docs/latest/api/ (дата звернення: 20.04.2024).
11. Документація React Router для навігації в React додатках. *React Router Documentation*. URL: https://reactrouter.com/ (дата звернення: 23.04.2024).
12. Офіційна документація MySQL для роботи з реляційними базами даних. *MySQL Documentation*. URL: https://dev.mysql.com/doc/ (дата звернення: 24.04.2024).
13. Офіційна документація TypeScript для типізації коду в React проєктах. TypeScript Documentation. URL: https://www.typescriptlang.org/docs/ (дата звернення: 26.04.2024).
14. Документація ESLint для стандартизації коду в JavaScript проєктах. *ESLint Documentation*. URL: https://eslint.org/docs/user-guide/getting-started (дата звернення: 28.04.2024).
15. Документація GitHub Actions для автоматизації процесів CI/CD у проєктах на GitHub. *GitHub Actions Documentation*. URL: https://docs.github.com/en/actions (дата звернення: 30.04.2024).