МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ КОМП'ЮТЕРНИХ НАУК ТА ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач випускової кафедри
_____ Аліна САВЧЕНКО
«__»_____2023 р.

КВАЛІФІКАЦІЙНА РОБОТА
(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ МАГІСТР
ЗА ОСВІТНЬО-ПРОФЕСІЙНОЮ ПРОГРАМОЮ
«ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ ПРОЕКТУВАННЯ»

Тема: «Вебзастосунок для листування в режимі реального часу»

Виконавець: Володимир ПОТІЙЧУК

Керівник: к.т.н., доцент Вікторія СИДОРЕНКО

Нормоконтролер: к.т.н., доцент Олена ТОЛСТІКОВА

КИЇВ 2023

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних наук та технологій*
Кафедра *комп'ютерних інформаційних технологій*
Спеціальність *122 «Комп'ютерні науки»*
Освітньо-професійна програма *«Інформаційні технології проектування»*

ЗАТВЕРДЖУЮ:
завідувач кафедри КІТ
_____Аліна САВЧЕНКО
(підпис)
«___»_____2023 р.

ЗАВДАННЯ
на виконання кваліфікаційної роботи
Потійчука Володимира Сергійовича
(ПІБ випускника)

1. Тема роботи: «Вебзастосунок для листування в режимі реального часу» затверджена наказом ректора № 1976/ст від 29.09.2023р.

2. Термін виконання роботи: з 02 жовтня 2023 року по 31 грудня 2023 року.

3. Вихідні дані до роботи: застосунок на мові програмування Elixir для демонстрації роботи листування в режимі реального часу.

4. Зміст пояснювальної записки: 1. Поняття вебзастосунку в режимі реального часу. 2. Проектування застосунку. 3. Розробка та тестування застосунку.

5. Перелік обов'язкового ілюстративного матеріалу: 1. Аналіз відомих застосунків для листування. 2. Моделі сутностей вебзастосунку. 3. Інші використані технології. 4. Демонстрація застосунку.

6. Календарний план-графік

№ з/п	Завдання	Термін виконання	Підпис керівника
1.	Аналіз предметної області та огляд аналогів. Написання 1 розділу, аналіз та поняття технології	20.11.2023-26.11.2023	
2.	Вибір та опис використаних технологій. Написання 2 розділу, проектування застосунку	26.11.2023-30.11.2023	
3.	Написання 3 розділу, розробка та тестування застосунку	31.12.2023-14.12.2023	
4.	Загальне редагування та друк пояснювальної записки	15.12.2023-20.12.2023	
5.	Проходження нормоконтролю, перепліт пояснювальної записки	16.12.2023-20.12.2023	
6.	Розробка тексту доповіді. Оформлення графічного матеріалу для презентації	19.12.2023-22.12.2023	

7. Дата видачі завдання 02.10.2023р.

Керівник кваліфікаційної роботи _____ Вікторія СИДОРЕНКО
(підпис керівника)

Завдання прийняв до виконання _____ Володимир ПОТІЙЧУК
(підпис випускника)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи на тему: «Вебзастосунок для листування в режимі реального часу» містить: 84 сторінки, 29 рисунків, 15 інформаційних джерел.

Об'єкт дослідження – процес розробки вебзастосунку для листування в режимі реального часу.

Предмет дослідження – методи, способи та засоби розробки застосунків для листування в режимі реального часу.

Мета кваліфікаційної роботи – дослідження напрямків і технологій розробки вебзастосунків для листування в режимі реального часу. Розробка вебзастосунку на базі TCP протоколу Websocket з використанням Elixir Phoenix фреймворку для високонавантажених та масштабованих систем.

Методи дослідження – мова програмування Elixir, текстовий редактор neovim**.**

Результати кваліфікаційної роботи рекомендується використовувати для демонстрації роботи листування в режимі реального часу, та в подальшій інтеграції у компанії та підприємства.

ВЕБЗАСТОСУНОК, ELIXIR, PHOENIX, LIVE VIEW, TCP, WEBSOCKET, TAILWIND, ECTO, GUARDIAN, EXUNIT, JAVASCRIPT, AJAX

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

TCP (Transmission Control Protocol) – Протокол управління передачею

IDE (Integrated development environment) – Інтегроване середовище розробки

HTTP (HyperText Transfer Protocol) – Протокол передачі гіпертекстових документів

AJAX (Asynschronyous Javascript And XML) – Асинхронний Javascript та XML

OIDC (Open ID Connector Github) – шар даних, побудований на основі фреймворку OAuth 2.0

ВСТУП

Сучасний світ вимагає швидкого та ефективного спілкування, особливо в онлайн-просторі, де відстані та часові зони не повинні бути перешкодою. Традиційні методи листування, такі як електронна пошта або форуми, часто зазнають затримок у доставці повідомлень, що може ускладнити спілкування та співпрацю. Крім того, існуючі платформи можуть не забезпечувати достатній рівень безпеки або приватності, що є критично важливим для користувачів, які обмінюються конфіденційною інформацією. Також, з ростом кількості користувачів та обсягу даних, виникає потреба в масштабованості та високій доступності сервісів листування.

Застосунок для листування в режимі реального часу може бути використаний для різноманітних цілей, як у персональному, так і в професійному контексті, наприклад в особистому листуванні, де друзі та сім'я можуть використовувати такі застосунки для підтримки зв'язку, обміну новинами, фотографіями та відео, незалежно від відстані. Інший приклад використання це робоча комунікація, команди можуть використовувати застосунки для листування для координації проектів, швидкого обміну інформацією та прийняття рішень у реальному часі.

Метою роботи є дослідження напрямків і технологій створення вебзастосунків для листування в режимі реального часу. Розробка вебзастосунку на базі TCP протоколу Websocket з використанням Elixir Phoenix фреймворку для високонавантажених та масштабованих систем.

Для досягнення поставленої мети необхідне рішення наступних завдань:

- огляд та аналіз існуючих вебзастосунків для листування в режимі реального часу;
- аналіз сфер застосування існуючих вебзастосунків;
- розробка структури програмного забезпечення.
- створення вебзастосунку у вигляді системи для листування в режимі реального часу.

Об'єктом досліджень є процес створення вебзастосунку для листування в режимі реального часу.

Предметом досліджень є методи та засоби створення застосунків для листування в режимі реального часу за допомогою протоколу Websocket з використанням фреймворку Phoenix на базі мови Elixir.

Актуальність теми кваліфікаційної роботи «Вебзастосунок для листування в режимі реального часу» ґрунтується на тому, що в теперішній час, люди мають потребу в миттєвому листуванні, оскільки сучасний темп життя вимагає миттєвого обміну інформацією. Люди хочуть отримувати відповіді на свої питання відразу, що є можливим завдяки веб-застосункам для листування. Також у глобалізованому світі люди та компанії спілкуються з партнерами, друзями та родиною, які можуть знаходитись в різних куточках світу. Вебзастосунки для листування в режимі реального часу дозволяють підтримувати зв'язок без обмежень, пов’язаних з відстанню.

Відповідно до поставленої мети роботи визначено основні **завдання дослідження**:

− провести аналіз наукової та методичної літератури;

− проаналізувати актуальні технології для розробки вебзастосунків в режимі реального часу;

− провести аналіз існуючих вебзастосунків

− розробити вебзастосунок для демонстрації роботи листування в режимі реального часу;

− описати принцип роботи вебзастосунку, його коду та провести тестування.

Для досягнення поставленої мети й виконання завдань використано метод системно-структурного аналізу існуючих вебзастосунків, який дав змогу показати особливості існуючих систем для листування в режимі реального часу та розробки таких вебзастосунків за допомогою мови програмування Elixir. Для формулювання і систематизації висновків використано методи аналізу, формалізації, абстрагування та узагальнення.

Наукова новизна розробленого в рамках кваліфікаційної роботи вебзастосунку полягає не лише у створенні портативного інструменту для листування в режимі реального часу, але й у його високій адаптивності та інтегрованості. Застосунок

розроблено з урахуванням сучасних вимог до корпоративних комунікаційних систем, що дозволяє йому легко вписуватися в існуючі бізнес-процеси різноманітних комерційних компаній. Його модульна структура та використання відкритих стандартів забезпечують можливість швидкої адаптації під специфічні потреби організацій, а також легке оновлення та масштабування системи.

Проект відзначається високим рівнем безпеки, що є критично важливим для корпоративного використання. Розроблена система аутентифікації та шифрування даних забезпечує захист від несанкціонованого доступу та витоку конфіденційної інформації, що є значним кроком у забезпеченні корпоративної безпеки. Це особливо актуально в умовах зростаючих кіберзагроз та посилення вимог до захисту даних.

Застосунок також відрізняється високою швидкістю роботи та оптимізацією для різних пристроїв, що робить його доступним для користувачів з будь-якими технічними можливостями. Інтуїтивно зрозумілий інтерфейс сприяє швидкому залученню користувачів та знижує поріг входу для нових співробітників компанії. Враховуючи тенденції до дистанційної роботи та необхідності ефективної комунікації між віддаленими командами, розроблений месенджер може стати незамінним інструментом для підтримки злагодженої роботи та координації дій.

Перспективи використання розробленого вебзастосунку в комерційному секторі є величезними, оскільки він відповідає сучасним вимогам до мобільності, гнучкості та індивідуалізації корпоративних систем. Його впровадження може сприяти підвищенню продуктивності праці, оптимізації бізнес-процесів та покращенню внутрішньої та зовнішньої корпоративної комунікації. У підсумку, розроблений месенджер має всі шанси стати важливим активом для бізнесу, що прагне до інновацій та ефективності у своїй діяльності.

РОЗДІЛ 1
АНАЛІЗ ТА ПОНЯТТЯ ТЕХНОЛОГІЇ

1.1. Поняття веб-застосунку в режимі реального часу

Веб-застосунки, що працюють в режимі реального часу, мають ряд визначальних характеристик, які відрізняють їх від традиційних веб-додатків. Однією з ключових особливостей є здатність до двостороннього з'єднання, яке забезпечується за допомогою технологій, таких як WebSocket. Це дозволяє серверу та клієнту вести безперервний діалог, обмінюючись даними в обидва боки без затримок.

Ще однією важливою характеристикою є миттєве оновлення інформації. Користувачі веб-застосунків в режимі реального часу отримують оновлення без необхідності вручну оновлювати сторінку або виконувати будь-які інші дії. Це створює враження безперервної взаємодії та актуальності даних.

Масштабованість є критично важливою для веб-застосунків, які повинні обслуговувати велику кількість користувачів одночасно. Ефективні системи розробляються таким чином, щоб витримувати високе навантаження без втрати продуктивності або швидкості роботи.

Інтерактивність також відіграє важливу роль, оскільки користувачі можуть взаємодіяти з застосунком в реальному часі, що збільшує їх залученість та покращує загальний досвід використання. Синхронізація даних між різними користувачами та пристроями в реальному часі дозволяє всім учасникам бути в курсі останніх подій.

Розглядаючи кроки роботи веб-застосунку в режимі реального часу, першим етапом є встановлення з'єднання між клієнтом та сервером, яке часто ініціюється через WebSocket. Після встановлення з'єднання користувачі можуть пройти аутентифікацію та авторизацію, щоб отримати доступ до персоналізованих функцій застосунку.

Користувачі також підписуються на оновлення, що дозволяє їм отримувати. сповіщення про нові повідомлення або події. Обмін даними відбувається в реальному часі, і сервер може надсилати оновлення даних до клієнта, як тільки вони стають доступними. Сервер також обробляє запити від клієнтів, виконуючи необхідні операції та генеруючи відповіді.

Клієнт отримує оновлення від сервера та відображає їх у веб-застосунку, що забезпечує актуальність інформації без перезавантаження сторінки. Стан застосунку синхронізується між різними користувачами, забезпечуючи консистентність даних. У разі помилок або втрати з'єднання, застосунок намагається автоматично відновити з'єднання та синхронізувати стан. Коли користувач закриває застосунок або відключається, з'єднання з сервером закривається, і сервер оновлює статус користувача.

Технології, які використовуються для розробки вебзастосунків в режимі реального часу, включають різноманітні інструменти та платформи, які допомагають розробникам створювати ефективні та надійні рішення. Вибір технологій залежить від конкретних вимог до застосунку, його масштабованості, надійності, підтримки браузерів, а також легкості розробки та інтеграції.

Кроки роботи вебзастосунку в режимі реального часу можуть бути описані наступним чином:

Коли користувач відкриває вебзастосунок для листування в режимі реального часу, першим кроком є встановлення з'єднання між клієнтом (браузером користувача) та сервером. Це з'єднання зазвичай встановлюється за допомогою технологій, таких як WebSocket, які дозволяють створити постійний канал для двостороннього обміну даними.

Далі відбувається процес автентифікації та авторизації, де користувачі проходять перевірку своєї особистості та отримують доступ до функцій застосунку, це забезпечує безпеку та персоналізацію взаємодії в рамках сервісу.

Після автентифікації користувачі підписуються на оновлення з сервера, що можуть включати нові повідомлення, сповіщення про дії інших користувачів або інші

події, які відбуваються в застосунку. Це дозволяє користувачам отримувати актуальну інформацію без необхідності вручну оновлювати сторінку.

Обмін даними відбувається в реальному часі між сервером та клієнтом. Сервер може надсилати оновлення даних до клієнта, як тільки вони стають доступними, а клієнт може надсилати запити або інформацію на сервер. Це забезпечує динамічне та інтерактивне середовище для користувачів.

На сервері відбувається обробка запитів від клієнтів, виконання операцій, таких як зміна бази даних, та генерація відповідей. Це вимагає високої продуктивності та надійності серверної частини, щоб забезпечити безперебійну роботу застосунку.

Клієнт отримує оновлення від сервера та відображає їх у веб-застосунку. Це відбувається без перезавантаження сторінки, що забезпечує плавність та швидкість користувацького досвіду.

Синхронізація стану між різними користувачами та їхніми сесіями є важливою для забезпечення консистентності та актуальності даних у веб-застосунку. Це дозволяє користувачам бачити однакову інформацію в реальному часі.

У разі помилок або втрати з'єднання, застосунок намагається автоматично відновити з'єднання та синхронізувати стан, щоб користувачі могли продовжити роботу без значних перерв.

Коли користувач закриває застосунок або відключається, з'єднання з сервером закривається, і сервер оновлює статус користувача як "відключений". Це дозволяє системі коректно управляти ресурсами та забезпечувати актуальність даних про користувачів.

Ці кроки створюють основу для безперервної взаємодії між користувачем та веб-застосунком, що є ключовим для ефективної роботи застосунків в режимі реального часу.

1.2. Принципи функціонування вебзастосунку для листування

Спочатку веб-додатки були створені за принципом простої клієнт-серверної архітектури, де браузер (клієнт) відправляє запит на сервер для отримання даних. Типовий процес в такій моделі виглядає так: клієнт запитує веб-сторінку з сервера

через HTTP; сервер обробляє запит і формує відповідь; відповідь від сервера надсилається назад до клієнта [6].

Однак, з розвитком потреби у функціональності веб-додатків у реальному часі, традиційна модель виявилася недостатньою для забезпечення необхідної швидкості відгуку. Це призвело до пошуку нових, більш інноваційних методів маніпуляції стандартним HTTP-запитом та відповіддю.

Одним із таких методів є коротке опитування (Short Polling), де клієнт регулярно відправляє запити до сервера через XMLHttpRequest/AJAX, щоб перевірити наявність нових даних. Процес короткого опитування полягає в тому, що клієнт ініціює запити через короткі проміжки часу, а сервер відповідає на них, як на звичайні HTTP-запити. Цей метод вимагає значних ресурсів, оскільки кожен новий запит потребує встановлення з'єднання, передачі HTTP-заголовків, запиту на дані та відправлення відповіді, яка часто не містить нової інформації. Після цього з'єднання закривається, і ресурси звільняються. Щоб досягти оновлення в реальному часі, запити на оновлення даних потрібно було б відправляти майже кожну секунду, що створює велике навантаження на сервер, оскільки він мусить обробляти велику кількість запитів, навіть коли немає змін у вмісті.

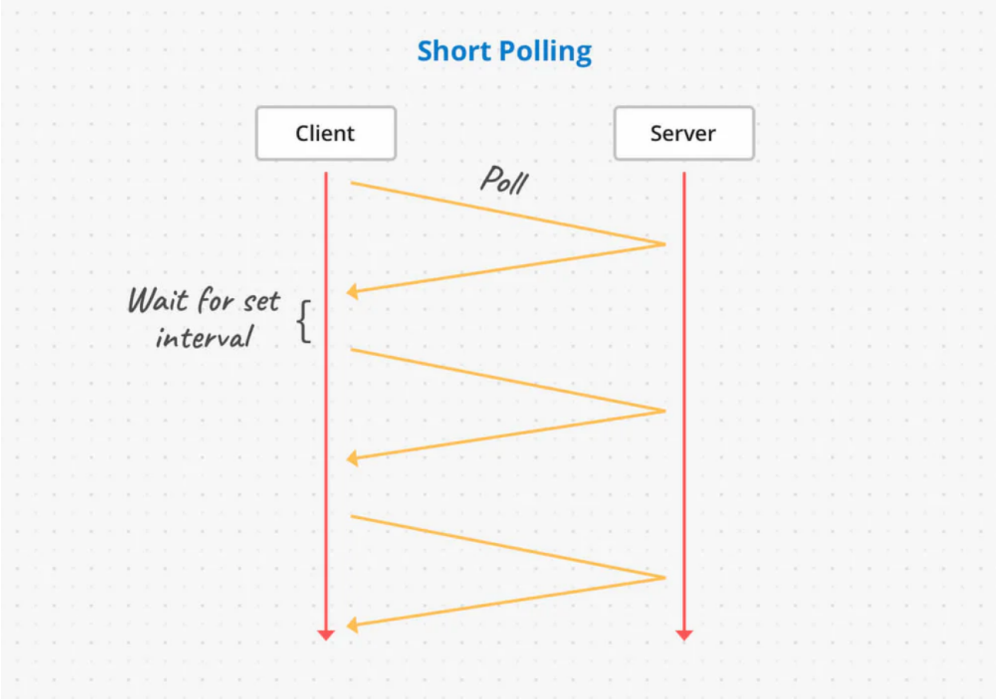

Рис. 1.1. Схема Short Polling

Техніка довгого опитування (long polling) є методом, який використовується в розробці веб-додатків для реалізації взаємодії між клієнтом та сервером в реальному часі. Вона полягає в створенні HTTP-запиту клієнтом, який сервер не обробляє негайно, а замість цього утримує відкритим. Сервер чекає на наявність нових даних, які можуть бути подією, повідомленням чи іншою актуалізацією, перед тим як надіслати відповідь.

Цей процес відрізняється від традиційного опитування (polling), де клієнт регулярно надсилає запити до сервера з певною періодичністю, щоб перевірити наявність нових даних. У випадку звичайного опитування, більшість запитів можуть бути неефективними, оскільки нові дані можуть з'являтися не так часто, як запити відсилаються.

З іншого боку, довге опитування зменшує кількість непотрібних запитів, оскільки сервер відповідає тільки тоді, коли є нові дані для передачі. Якщо нових даних немає, сервер може утримувати запит відкритим до досягнення максимально допустимого часу очікування, після чого він відправить порожню відповідь або статус, що свідчить про відсутність нових даних. У будь-якому випадку, після отримання відповіді від сервера, клієнт негайно ініціює новий запит, щоб продовжити очікування наступних оновлень.

Цей метод є компромісом між необхідністю забезпечення актуальності даних та бажанням знизити навантаження на мережу та сервер. Хоча довге опитування не є таким ефективним, як WebSocket або інші технології двостороннього зв'язку, воно може бути корисним у ситуаціях, де використання більш сучасних методів неможливе через обмеження середовища або сумісності.

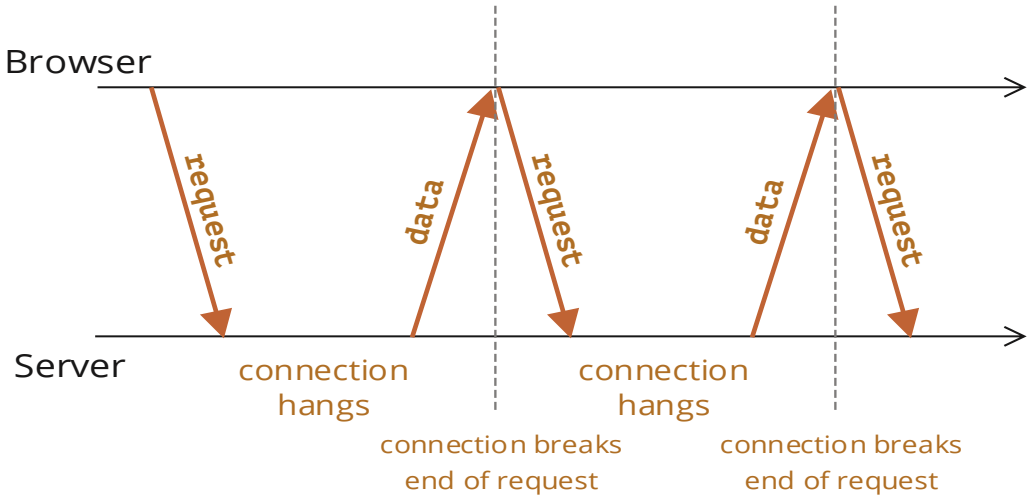

Рис. 1.2. Схема Long Polling

На клієнтській стороні управління зводиться до відправлення єдиного запиту на сервер. Після отримання відповіді, клієнт може відправити наступний запит, продовжуючи цикл відправлення та отримання даних за потреби. Механізм довгого опитування працює таким чином: клієнт робить накопичення запитів та зайвого мережевого трафіку через постійні нові запити та заголовки. Плюси та мінуси цього підходу схожі на коротке опитування, але з деякими відмінностями. Незважаючи на певні переваги, технології короткого та довгого опитування зараз значно поступаються за продуктивністю та зручністю сучасним методам і зазвичай використовуються в застарілих проектах. Більшість нових чат-додатків розробляються з використанням технології WebSocket, яка забезпечує оптимальне навантаження на сервер та мінімальні затримки в обміні даними.

Протокол WebSocket представляє собою передову технологію, яка відкриває нові горизонти для інтерактивної комунікації між клієнтом та сервером у вигляді двостороннього повно дуплексного з'єднання. Цей протокол, що працює на бінарному рівні, дозволяє обмінюватися даними майже без затримок, що є незамінним для додатків, які потребують швидкої відповіді сервера, таких як онлайн-ігри, системи миттєвих повідомлень та інші застосунки, де критично важливою є максимальна актуальність інформації [4].

Опис дії WebSocket:

- XHR/AJAX запит на сервер для отримання даних;
- сервер утримує відповідь, чекаючи на наявність нових даних;
- як тільки нові дані стають доступними, сервер надсилає їх клієнту;
- клієнт отримує дані та відразу ж ініціює новий запит, що реініціює цикл;

WebSocket відмінно інтегрується з існуючою інфраструктурою Інтернету, оскільки він сумісний з HTTP-протоколом, але при цьому пропонує значно більш ефективний механізм обміну даними. Протокол використовує стандартні порти 80 та 443 для незашифрованих та зашифрованих з'єднань відповідно, що дозволяє йому легко працювати через більшість HTTP-проксі та мережевих посередників, забезпечуючи широку доступність та легкість впровадження.

Процес встановлення з'єднання WebSocket починається з ініціації рукостискання з боку клієнта, який відправляє спеціальний запит з заголовком Upgrade, сигналізуючи про бажання перейти на протокол WebSocket. Сервер, отримавши такий запит, аналізує його та, у разі згоди, відправляє підтвердження, після чого з'єднання вважається встановленим, і обидві сторони можуть почати обмін даними. У випадку, якщо сервер не може встановити з'єднання, він відправляє відповідь про неможливість підключення. Після успішного встановлення з'єднання, клієнт та сервер можуть обмінюватися повідомленнями в обох напрямках, що створює умови для реалізації інтерактивних додатків з високою швидкістю відповіді.

З'єднання залишається активним до тих пір, поки одна зі сторін не вирішить його закрити, після чого воно розривається.

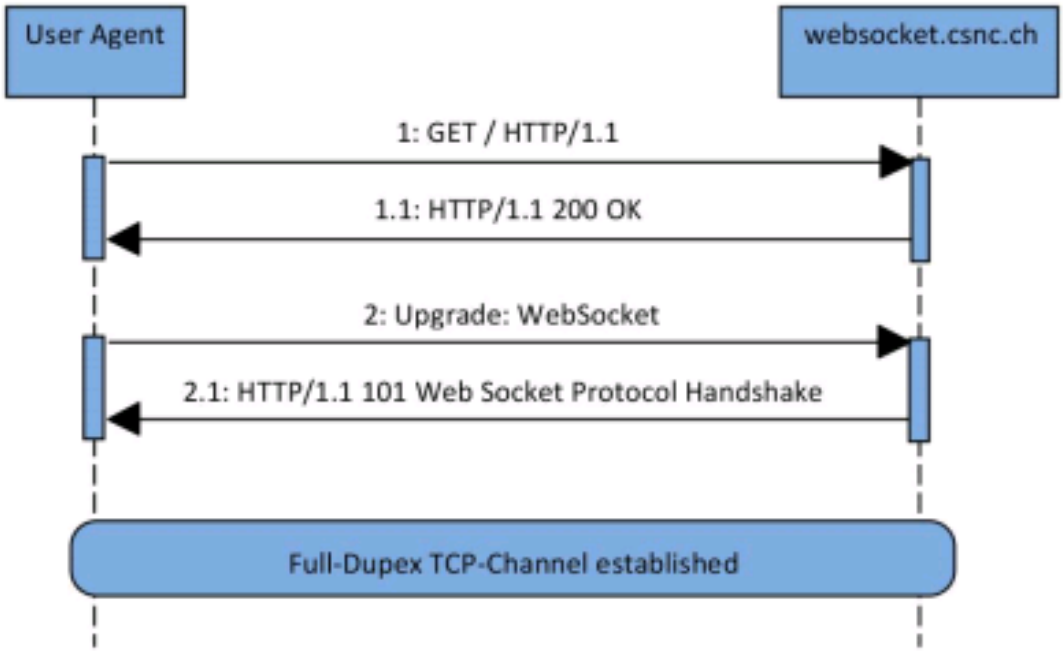

Рис. 1.3. Схема Websocket

Однією з ключових переваг використання WebSocket [8]t у серверній архітектурі є те, що після ініціалізації з'єднання через HTTP рукостискання, подальше спілкування відбувається через спеціалізований протокол обміну повідомленнями. Це відрізняється від традиційних HTTP-запитів і пропонує значні переваги з точки зору продуктивності та структури системи. Наприклад, використовуючи Elixir, можна ефективно управляти пам'яттю, дозволяючи різним сокетним з'єднанням взаємодіяти з одними й тими ж даними в оперативній пам'яті, без необхідності постійного звернення до бази даних як проміжного сховища, як це було б з AJAX або іншими техніками, такими як long polling у мові програмування PHP.

Це дозволяє розробникам створювати більш швидкі та ефективні веб-додатки, оскільки дані можуть бути негайно передані між клієнтами та сервером або навіть розподілені між різними сокетами в реальному часі. Такий підхід підвищує швидкість відгуку додатків та забезпечує кращий користувацький досвід.

Проте, існують певні недоліки при використанні WebSocket. Оскільки WebSocket не є сумісним з HTTP після рукостискання, потрібно налаштувати спеціалізований сервер, що може ускладнити процес налагодження. Крім того, хоча сучасні браузери мають досить хорошу підтримку WebSocket, існують старіші версії,

які можуть не підтримувати цей протокол, що вимагає від розробників реалізовувати запасні механізми для забезпечення сумісності [5].

Незважаючи на ці виклики, WebSocket залишається популярним вибором для розробки інтерактивних веб-додатків, які потребують двостороннього зв'язку в реальному часі, завдяки своїй здатності забезпечувати швидкі та надійні з'єднання між клієнтами та серверами [7].

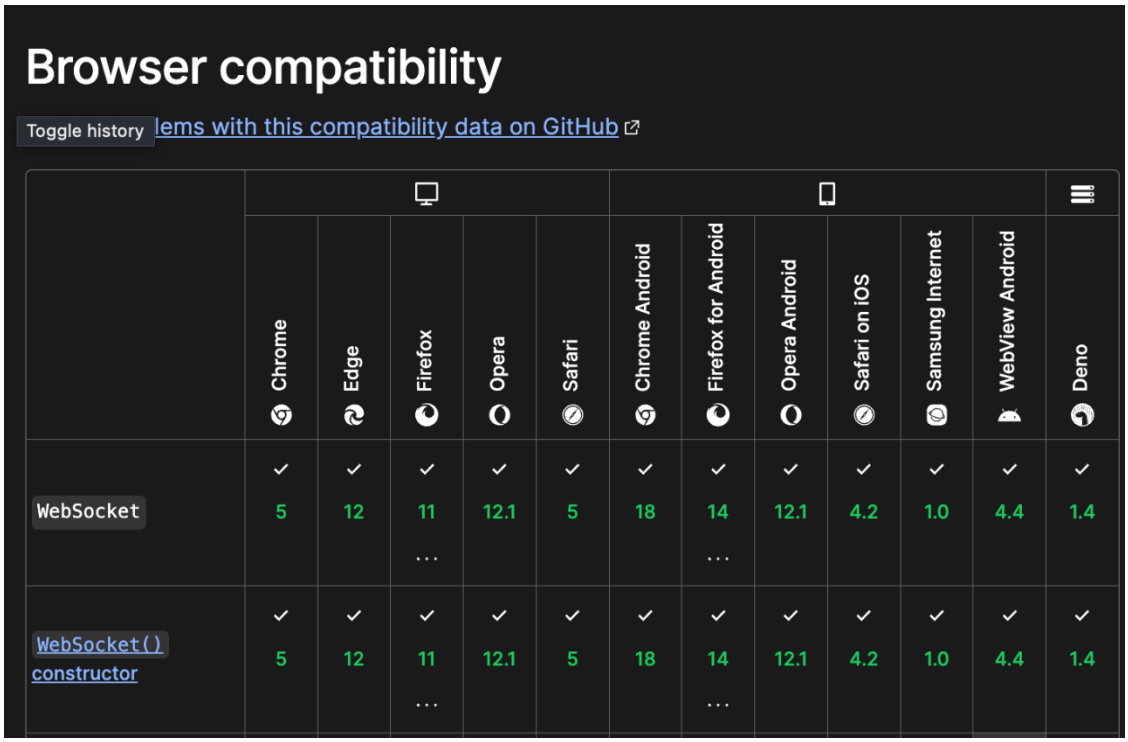

Рис. 1.4. Схема сумісності Websocket з браузерами

1.2. Аналіз відомих застосунків для листування

Проаналізуємо відомі рішення для листування, такі як:

- Telegram;
- Google Classroom;
- Djinni;
- Slack.

Telegram є одним з найпопулярніших веб-застосунків для листування в режимі реального часу, який також доступний у вигляді мобільного додатку та десктопної програми. Цей застосунок надає користувачам можливість миттєво обмінюватися

текстовими повідомленнями, зображеннями, відео та іншими типами файлів. Telegram славиться своєю швидкістю та безпекою, пропонуючи сильне шифрування та приватність.

Основною проблемою, яку вирішує Telegram, є потреба в швидкому, безпечному та надійному способі комунікації. Він дозволяє користувачам створювати групові чати з великою кількістю учасників, канали для розсилки повідомлень широкій аудиторії, а також секретні чати з можливістю самознищення повідомлень.

Метою Telegram є надання користувачам платформи для миттєвого обміну повідомленнями, яка була б доступна на будь-якому пристрої та забезпечувала б високий рівень безпеки. Застосунок використовує MTProto протокол, який дозволяє швидко передавати великі обсяги даних, забезпечуючи при цьому сильне шифрування.

Telegram також відомий своєю масштабованістю, оскільки може обслуговувати сотні мільйонів активних користувачів одночасно. Його інтерфейс є простим та інтуїтивно зрозумілим, що робить його доступним для широкого кола користувачів, незалежно від їхнього технічного досвіду.

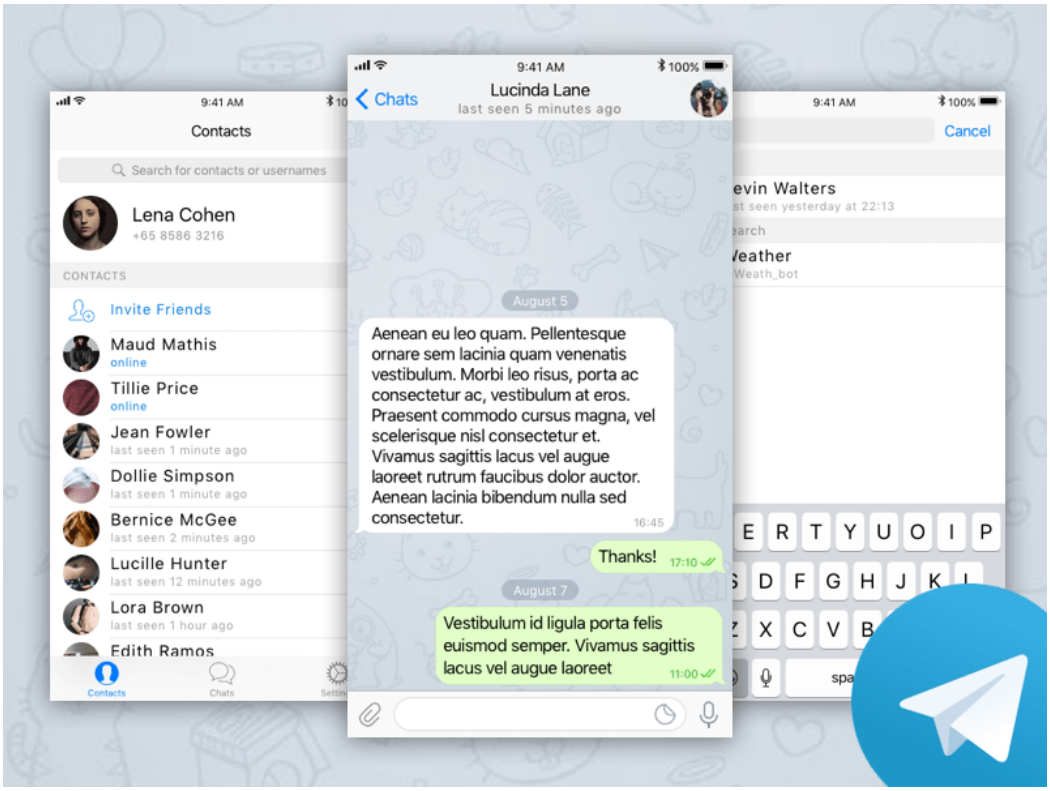

Рис. 1.5. Вигляд вебзастосунку Telegram

Google Classroom є інтерактивним освітнім веб-застосунком, розробленим компанією Google, який дозволяє викладачам та студентам організовувати навчальний процес онлайн. Цей застосунок спрощує створення, розподіл та оцінювання навчальних завдань в цифровому форматі, а також надає можливість для ефективної комунікації між учасниками навчального процесу.

Google Classroom вирішує виклики, пов'язані з дистанційним навчанням та колаборацією, надаючи платформу, де викладачі можуть управляти курсами, ділитися матеріалами, збирати завдання та надавати зворотний зв'язок студентам. Студенти, у свою чергу, можуть відправляти виконані завдання, брати участь у дискусіях та отримувати оновлення курсу в одному централізованому місці.

Застосунок інтегрується з іншими сервісами Google, такими як Google Docs, Sheets, Slides та Drive, що дозволяє легко створювати та розподіляти навчальний контент. Однією з ключових переваг є можливість спільної роботи над документами в режимі реального часу, що сприяє груповій взаємодії та співпраці.

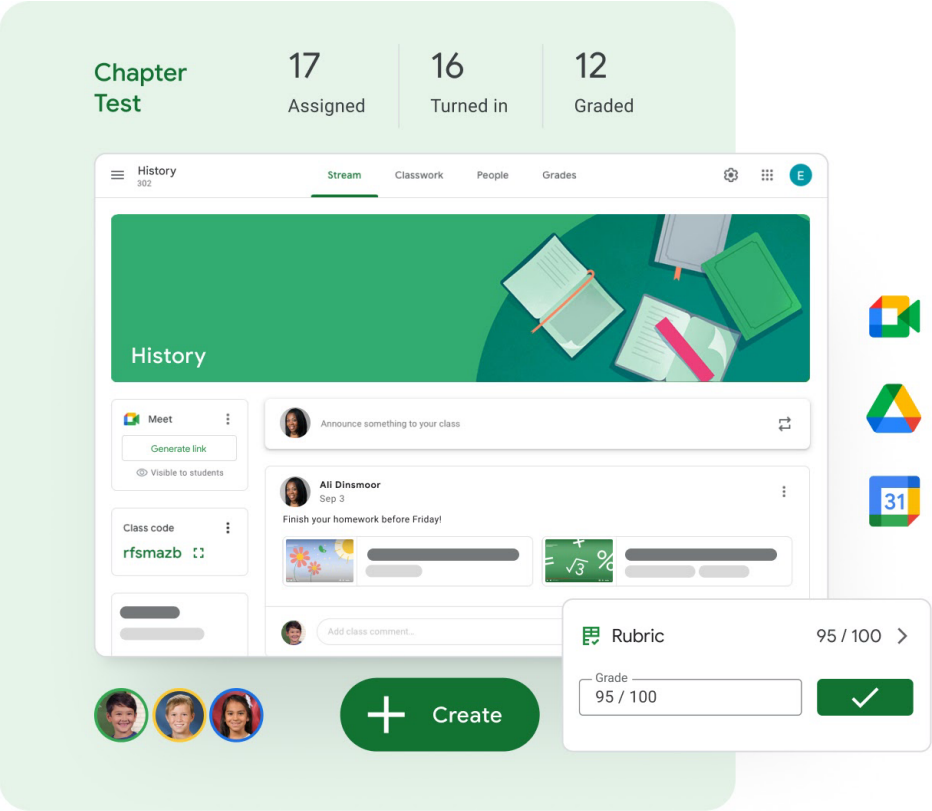

Рис. 1.6. Вигляд вебзастосунку Google Classroom

Slack є платформою для комунікації команд, яка забезпечує централізоване місце для обміну повідомленнями, файлами та інформацією, сприяючи співпраці та продуктивності в робочих процесах. Він використовує модель каналів для організації діалогів за темами, проектами або командами, що дозволяє користувачам легко відстежувати обговорення та участь у них.

Переваги Slack:

- організація комунікації, структуровані канали дозволяють розділити обговорення за темами, що спрощує пошук інформації та зменшує інформаційний шум;
- інтеграція з іншими інструментами, Slack може інтегруватися з багатьма зовнішніми сервісами та додатками, що розширює його функціональність і автоматизує робочі процеси;
- пошукова система, потужні можливості пошуку дозволяють користувачам швидко знаходити потрібні повідомлення, файли та згадки;
- доступність, Slack доступний на різних платформах, включаючи веб-браузери, настільні операційні системи та мобільні пристрої, що забезпечує зручність використання;
- сповіщення та налаштування, користувачі можуть налаштувати сповіщення таким чином, щоб отримувати важливі оновлення без перевантаження непотрібними повідомленнями.

Недоліки Slack:

- перевантаження інформацією, велика кількість повідомлень та активність у каналах може призвести до відчуття перевантаження інформацією;
- Вартість, преміум-функції Slack вимагають підписки, що може бути дороговартісним для деяких організацій, особливо з великою кількістю користувачів;
- складність управління, управління правами доступу, архівація каналів та інші адміністративні завдання можуть бути складними в організаціях з великою кількістю користувачів;

- залежність від інтернету, як і будь-який вебзастосунок, Slack залежить від стабільного інтернет-з'єднання, що може бути проблемою в місцях з поганим покриттям;
- конфіденційність, хоча Slack пропонує шифрування даних, питання конфіденційності та безпеки даних залишаються важливими для організацій, які обробляють чутливу інформацію.

Slack є потужним інструментом для спілкування та співпраці в командах, який використовує переваги реального часу для підвищення продуктивності та ефективності робочих процесів.

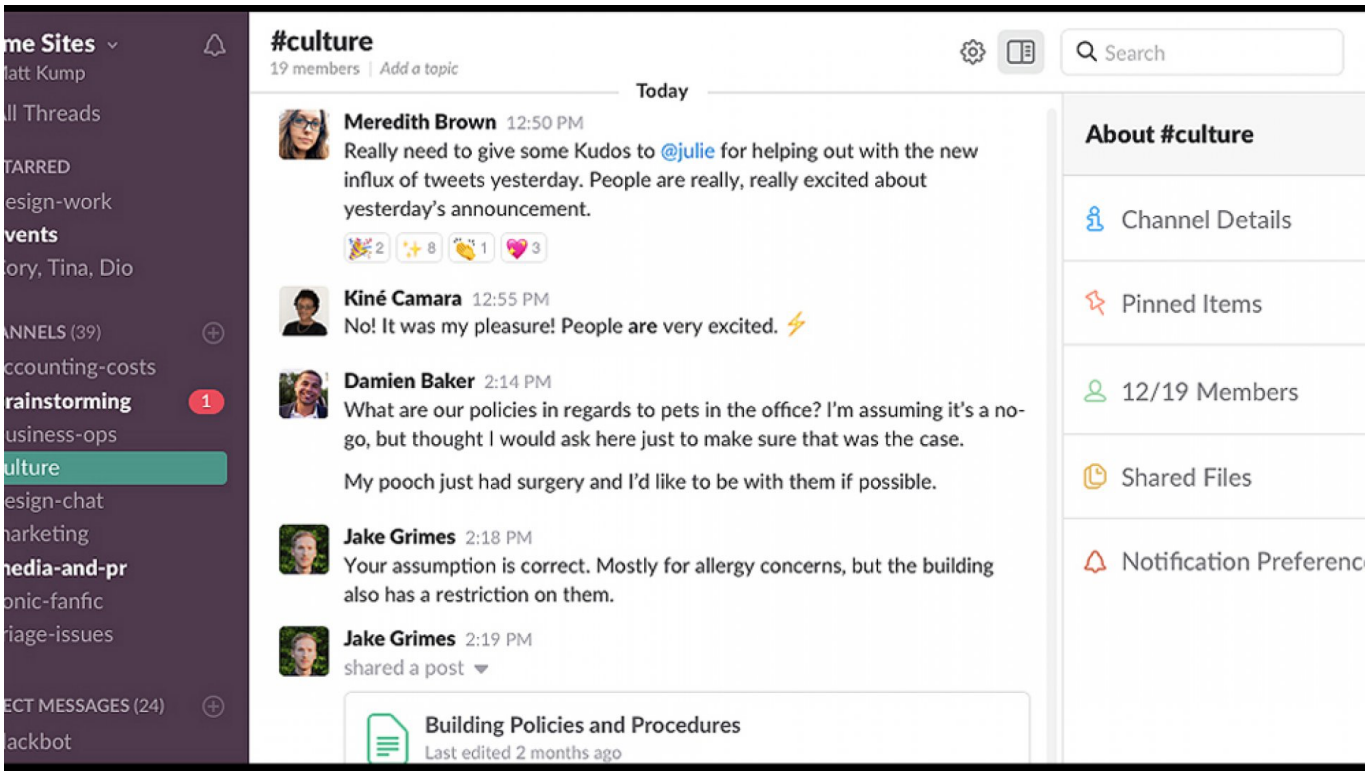

Рис. 1.7. Вигляд вебзастосунку Slack

Discord є комунікаційною платформою, яка спочатку була популярною серед геймерської спільноти, але з часом знайшла ширше застосування серед різноманітних груп користувачів. Цей застосунок надає можливість створення серверів, які поділяються на канали з текстовими та голосовими чатами для спілкування та обміну медіа.

Переваги Discord:

- голосовий чат, Discord вирізняється високоякісним голосовим чатом, який дозволяє користувачам спілкуватися у режимі реального часу без значних затримок;
- гнучкість налаштувань, користувачі можуть налаштовувати сервери, канали та ролі, що надає велику гнучкість у керуванні спільнотами та командами;
- безкоштовне використання, базові функції Discord доступні безкоштовно, що робить його доступним для широкого кола користувачів;
- інтеграція з іграми та програмами, Discord інтегрується з багатьма популярними іграми та програмами, що забезпечує додаткові можливості для спілкування та взаємодії;
- підтримка великої кількості користувачів, Discord може обслуговувати великі спільноти з тисячами учасників, забезпечуючи стабільність та якість зв'язку.

Недоліки Discord;

- складність для новачків, нові користувачі можуть зіткнутися зі складнощами у навігації та використанні всіх функцій Discord;
- модерація контенту, на великих серверах модерація контенту та поведінки учасників може бути складною та вимагати значних зусиль;
- приватність, хоча Discord пропонує шифрування, існують питання щодо збору та обробки даних користувачів;
- залежність від платформи, хоча Discord працює на багатьох пристроях, його функціональність може відрізнятися в залежності від платформи;
- використання ресурсів, Discord може вимагати значних ресурсів системи, особливо при використанні голосових каналів та відеодзвінків.

Discord є потужним інструментом для створення спільнот та команд, який забезпечує різноманітні можливості для спілкування та обміну інформацією в режимі реального часу

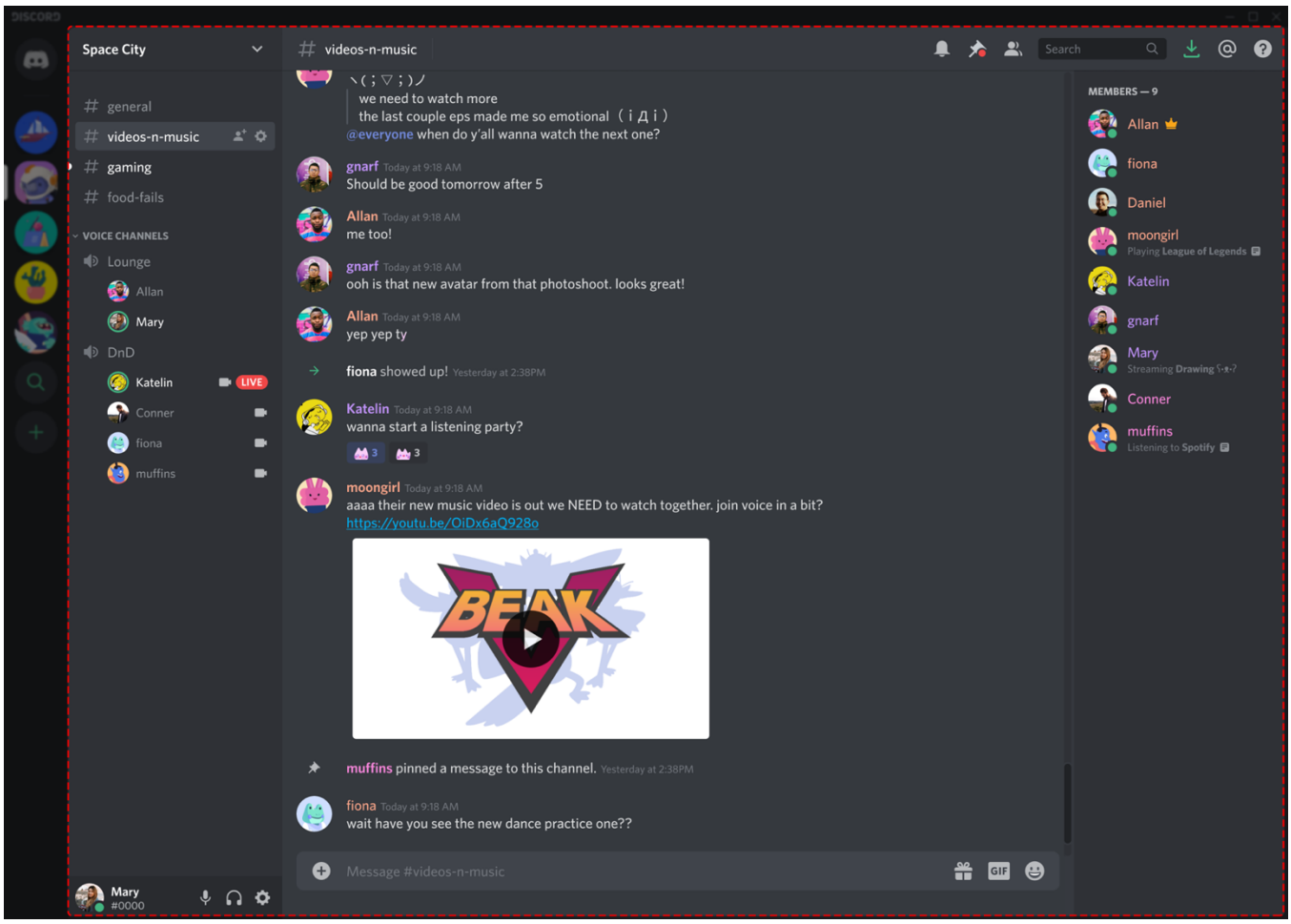

Рис. 1.8. Вигляд вебзастосунку Discord

1.3. Сфери використання веб-застосунків для листування в режимі реального часу

Веб-застосунки для миттєвого обміну повідомленнями стали невід'ємною частиною соціальних мереж, де вони використовуються для забезпечення приватного та колективного діалогу між користувачами. Ці інструменти дозволяють людям підтримувати зв'язок, ділитися думками та зміцнювати соціальні зв'язки в онлайн-просторі.

У корпоративному секторі, платформи для миттєвого обміну повідомленнями відіграють ключову роль у внутрішній комунікації, надаючи співробітникам засоби для обговорення робочих процесів, спільної роботи над проектами та швидкого обміну файлами та інформацією.

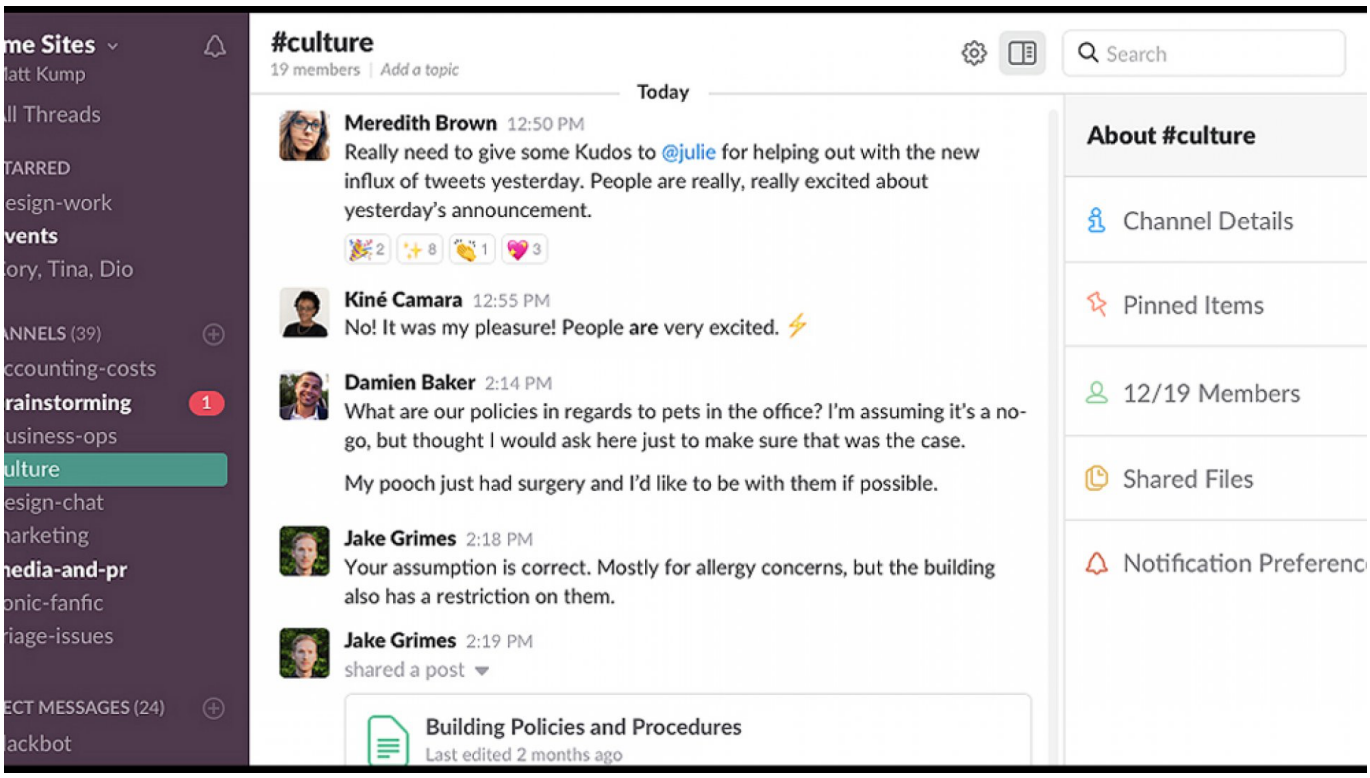

Рис. 1.9. Вигляд вебзастосунку Slack

Сфера освіти також втілює в собі переваги використання цих застосунків, оскільки вони допомагають викладачам та студентам взаємодіяти та співпрацювати над навчальними матеріалами. Це сприяє більш ефективному навчальному процесу та підтримує активне навчальне середовище.

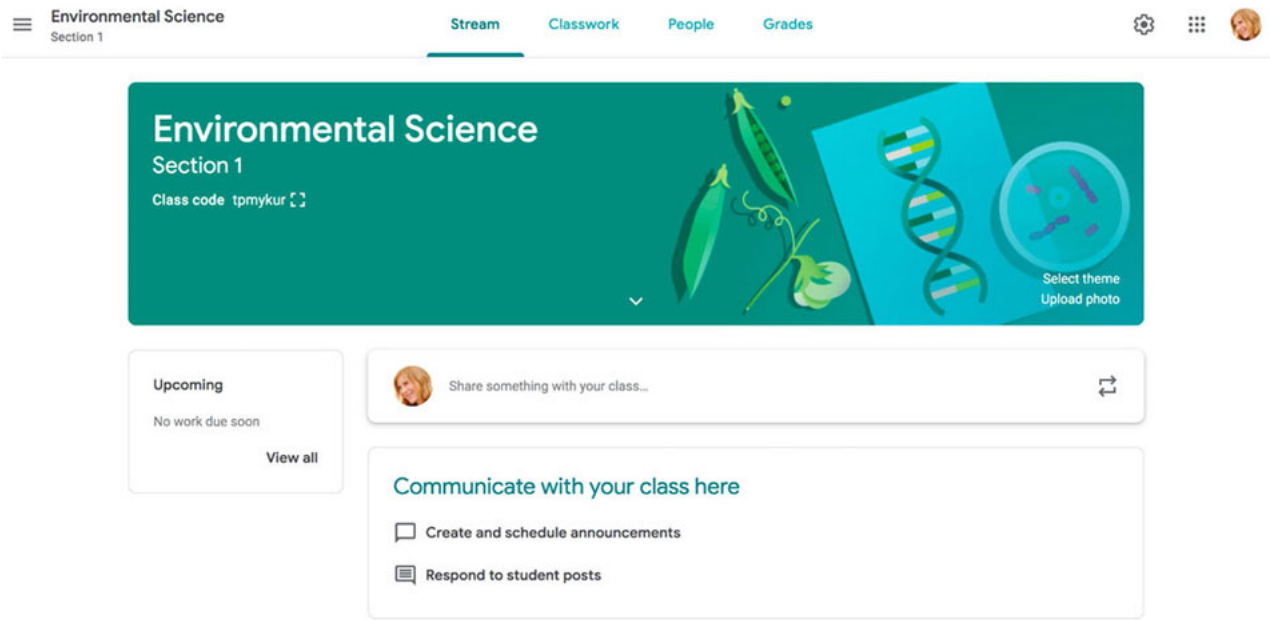

Рис. 1.10. Вигляд вебзастосунку Google Classroom

Технічна підтримка використовує веб-застосунки для листування для надання швидкої допомоги користувачам, що забезпечує ефективне вирішення проблем та підтримку продуктів.

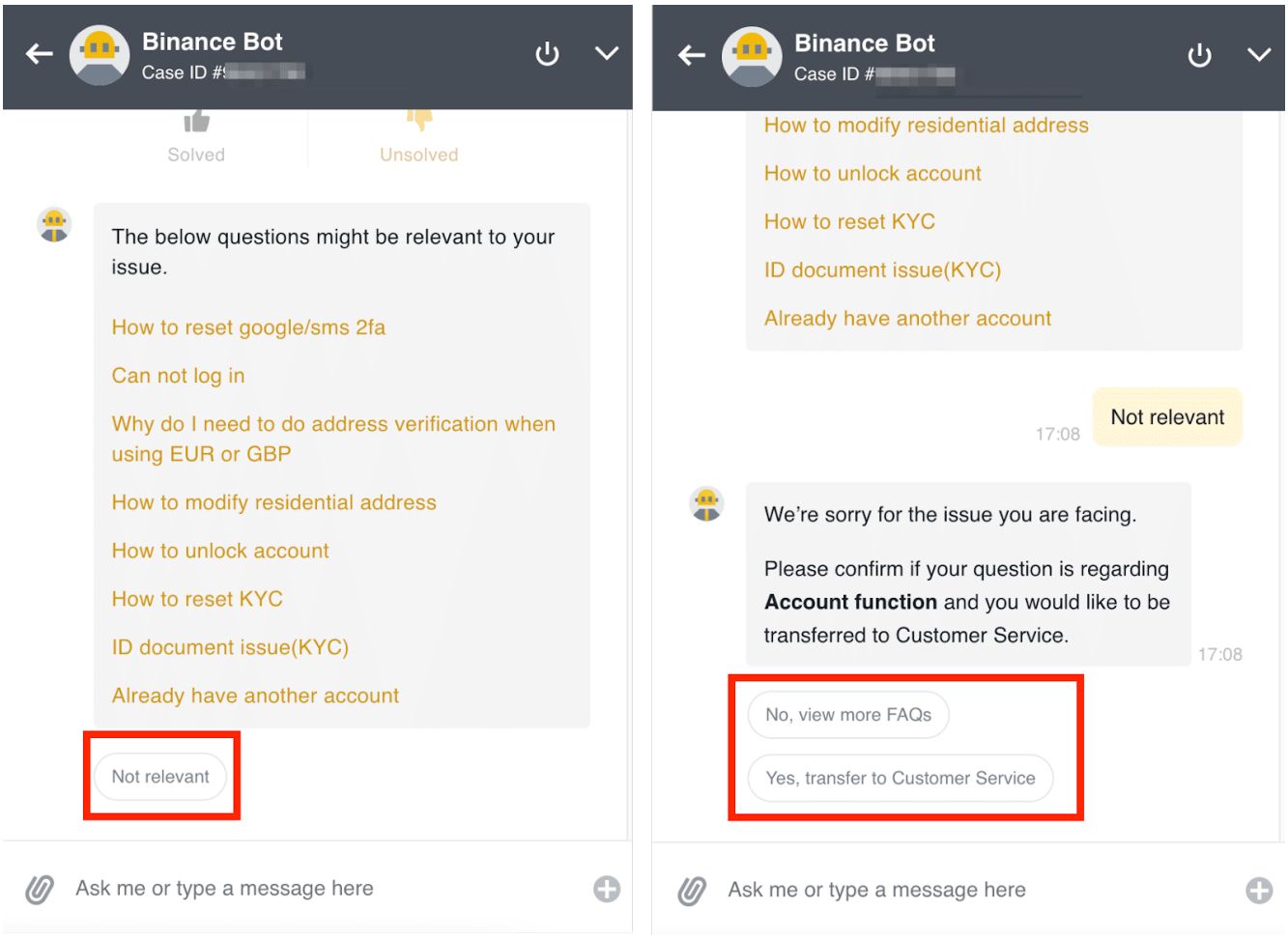

Рис. 1.11. Вигляд вебзастосунку Binance Bot

Геймінгова індустрія також інтегрує ці застосунки для створення більш згуртованих спільнот, де гравці можуть обговорювати стратегії, координувати свої дії та просто спілкуватися під час гри.

Рис. 1.12. Вигляд чату в онлайн грі

Телемедицина використовує веб-застосунки для листування для забезпечення зв'язку між пацієнтами та медичними фахівцями, що дозволяє проводити онлайн-консультації та обговорення лікування без необхідності особистих візитів.

Рис. 1.13. Вигляд чату в вебзастосунку для телемедецини

У фінансовому секторі, ці інструменти допомагають клієнтам отримувати поради від консультантів, обговорювати інвестиційні плани та управління активами в зручний для них спосіб.

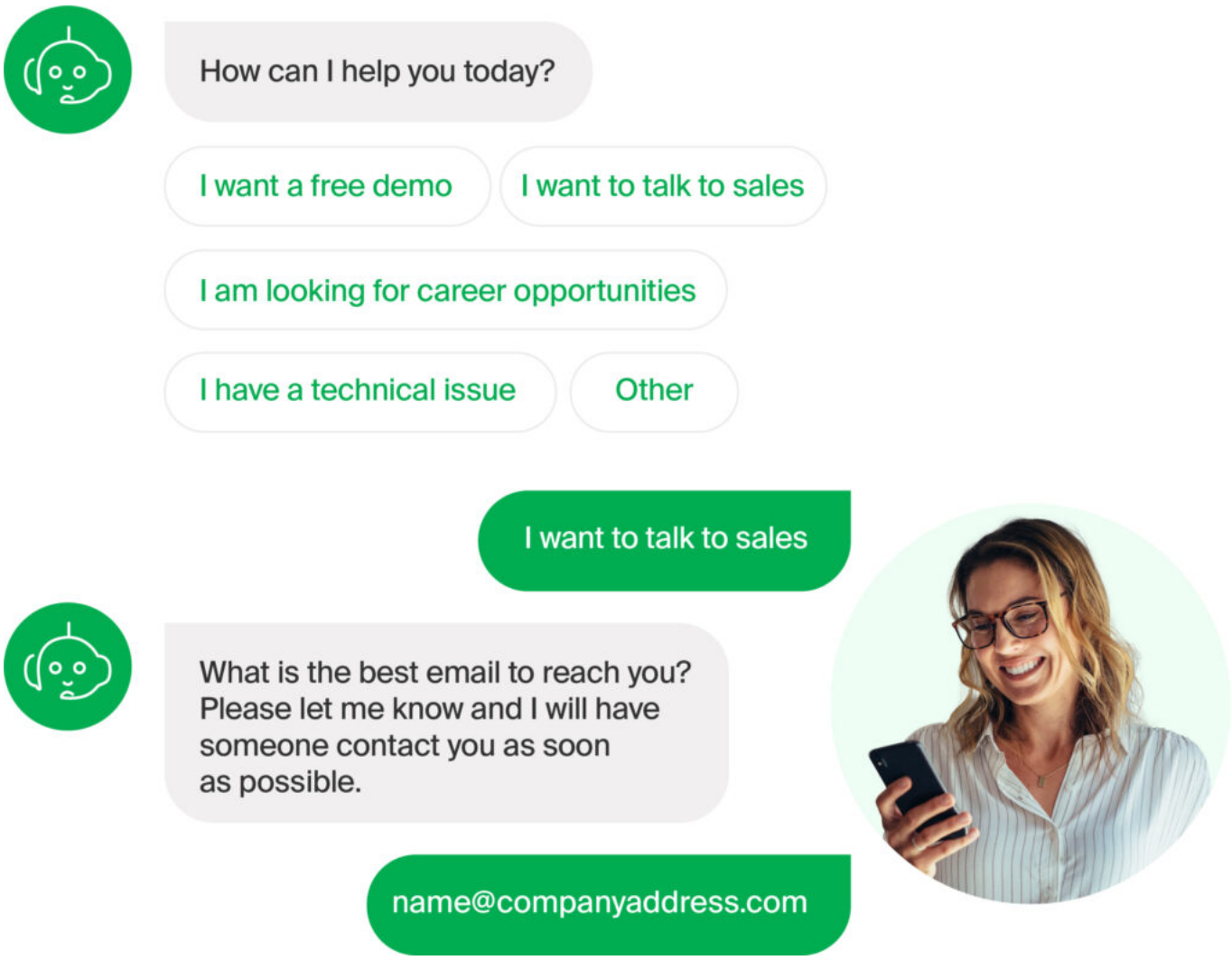

Рис. 1.14. Вигляд чату в вебзастосунку у фінансовому секторі

У сфері логістики та доставки, веб-застосунки для листування використовуються для координації процесів доставки, обговорення деталей замовлень та відстеження статусу відправлень.

Ці приклади підкреслюють важливість веб-застосунків для листування в режимі реального часу, які стали ключовими інструментами для спілкування та співпраці в різних сферах сучасного життя.

1.4. Порівняння

Для порівняння листування в режимі реального часу між дипломним проектом та іншими існуючими платформами, розглянемо наступні ознаки:

Автентифікація:

- кваліфікаційна робота: підтримує автентифікацію користувачів;
- Telegram: підтримує двофакторну автентифікацію;
- Google Classroom: використовує Google акаунти для автентифікації;
- Djinni: автентифікація через електронну пошту;
- Slack: підтримує автентифікацію через електронну пошту, Google акаунт, а також SSO (Single Sign-On).

2. Мобільна адаптація:
   - кваліфікаційна робота: адаптований для мобільних пристроїв;
   - Telegram: має додатки для всіх основних мобільних платформ;
   - Google Classroom: має мобільні додатки та адаптований веб-інтерфейс;
   - Djinni: веб-платформа, оптимізована для мобільних пристроїв;
   - Slack: має мобільні додатки та адаптований веб-інтерфейс.
3. Листування з іншими користувачами:
   - кваліфікаційна робота: підтримує листування в режимі реального часу;
   - Telegram: підтримує листування, групові чати та канали;
   - Google Classroom: комунікація через коментарі та пошту, не зосереджена на миттєвому обміні повідомленнями;
   - Djinni: комунікація здійснюється переважно через електронну пошту;
   - Slack: підтримує листування, групові чати, канали та тематичні розмови.
4. Пошук інших користувачів:
   - кваліфікаційна робота: має функціонал пошуку користувачів.
   - Telegram: пошук користувачів за номером телефону або іменем.
   - Google Classroom: пошук учасників класу за іменем.
   - Djinni: пошук кандидатів та рекрутерів за спеціалізацією.
   - Slack: Пошук учасників робочого простору за іменем або електронною поштою.
5. Основна мова програмування:
   - кваліфікаційна робота: використовує Elixir;
   - Telegram: використовує комбінацію мов, включаючи C++ для серверної частини;

- Google Classroom: ймовірно, використовує комбінацію мов, включаючи Java та JavaScript;
- Djinni: не відомо, але можливо використовує стек технологій, типовий для веб-платформ;
- Slack: використовує комбінацію мов, включаючи PHP, Java, Ruby та інші.

6. Швидкість:
   - кваліфікаційна робота: висока швидкість завдяки Elixir та Phoenix Channels;
   - Telegram: відомий своєю високою швидкістю та надійністю;
   - Google Classroom: швидкість залежить від сервісів Google, зазвичай висока;
   - Djinni: швидкість не є ключовим фактором, оскільки платформа зосереджена на рекрутингу;
   - Slack: швидкість може варіюватися, але загалом висока для реального часу.
7. Компактність:
   - кваліфікаційна робота: компактний та ефективний завдяки використанню Elixir та віртуальної машини Erlang;
   - Telegram: легкий та швидкий, незважаючи на широкий функціонал;
   - Google Classroom: фокус на освітніх функціях, не на компактності;
   - Djinni: простий веб-інтерфейс, орієнтований на зручність користувачів;
   - Slack: має багатий функціонал, що може впливати на компактність.
8. Функціональність:
   - кваліфікаційна робота: фокус на листуванні в режимі реального часу з базовими функціями;
   - Telegram: широкий спектр функцій, включаючи секретні чати, ботів, файли;
   - Google Classroom: освітній інструмент з функціями для класів та завдань;
   - Djinni: платформа для пошуку роботи та співробітників, фокус на рекрутингу;

- Slack: розширений набір функцій для спілкування та співпраці в команді.

Кваліфікаційна робота, яка фокусується на листуванні в режимі реального часу, відрізняється від інших розглянутих платформ своєю спеціалізацією та технологічним стеком. Використання Elixir та Phoenix Channels надає проекту високу швидкість та ефективність обробки одночасних з'єднань, що є ідеальним для реалізації функціоналу чату в реальному часі. Це створює основу для надійного та швидкого обміну повідомленнями, який може конкурувати з такими платформами, як Telegram та Slack, що також відомі своєю швидкістю.

На відміну від більш універсальних платформ, таких як Google Classroom, яка орієнтована на освітній процес, або Djinni, яка зосереджена на рекрутингу, кваліфікаційна робота зосереджена виключно на миттєвому обміні повідомленнями, що робить його більш прямолінійним та легким у використанні для кінцевих користувачів.

Щодо автентифікації, проект підтримує базову аутентифікацію користувачів, що є достатнім для забезпечення безпеки, але не має додаткових рівнів безпеки, таких як двофакторна автентифікація в Telegram. Це може бути розглянуто як потенційне поле для подальшого розвитку та вдосконалення.

Мобільна адаптація дипломного проекту забезпечує доступність на різних пристроях, що є важливим для сучасних користувачів, які очікують зручності використання додатків на смартфонах та планшетах.

У підсумку до першого розділу, кваліфікаційна робота вирізняється своєю спеціалізацією на листуванні в режимі реального часу, високою швидкістю та ефективністю, а також гнучкістю та масштабованістю, які надає використання Elixir та Phoenix Channels. Це робить його конкурентоспроможним серед інших платформ, надаючи користувачам простий та швидкий спосіб комунікації.

РОЗДІЛ 2
ПРОЕКТУВАННЯ ЗАСТОСУНКУ

2.1. Опис вимог до застосунку

Застосунок для демонстрації роботи вебзастосунку для листування в режимі реального часу:

- застосунок повинен мати функціонал реєстрації профілю за допомогою унікального ідентифікатору та паролю для подальшого використання
- застосунок повинен мати функціонал авторизації за допомогою унікального ідентифікатору та паролю вказаного при реєстрації
- застосунок повинен мати функціонал шифрування паролів користувачів
- застосунок повинен мати функцію анулювання авторизації
- застосунок повинен мати функцію пошуку інших користувачів
- застосунок повинен мати функцію створення чату з користувачем за допомогою пошуку
- застосунок повинен мати функцію листування в режимі реального часу

У результаті аналізу функціональних вимог та огляду аналогів, сформовано наступні нефункціональні вимоги:

- застосунок повинен мати зручний та мінімалістичний інтерфейс;
- застосунок повинен працювати в будь-яких браузерах підтримуючих TCP протокол Websocket.
- застосунок повинен бути швидким та компактним

2.2. Вибір мови програмування

Elixir було обрано як мову програмування для дипломного проекту через її унікальні характеристики та переваги, які відповідають вимогам сучасної веб-розробки. Elixir є функціональною мовою, побудованою на віртуальній машині Erlang VM (BEAM), яка відома своєю високою надійністю, масштабованістю та паралельною обробкою.

Elixir пропонує потужні інструменти для розробки, такі як Mix для управління проектами та Hex для управління залежностями. Ці інструменти спрощують створення, конфігурацію та управління проектами Elixir, забезпечуючи ефективний робочий процес [9].

Мова має багатий набір бібліотек та фреймворків, зокрема Phoenix Framework, який використовується в дипломному проекті. Це дозволяє швидко створювати інтерактивні веб-додатки з реальними часовими оновленнями.

Elixir вирізняється своєю високою продуктивністю та надійністю, які є результатом використання Erlang VM (BEAM). Erlang VM була спеціально розроблена для створення розподілених та відмовостійких систем, що можуть легко масштабуватися та управляти великою кількістю одночасних процесів. Це робить Elixir ідеальним вибором для веб-додатків, телекомунікаційних систем та високонавантажених сервісів, де потрібна висока доступність та здатність до швидкого відновлення після помилок [10].

Elixir також пропонує потужні інструменти для паралельного виконання коду завдяки легковажним процесам, які управляються віртуальною машиною. Ці процеси виконуються незалежно один від одного, що забезпечує високий рівень ізоляції та зменшує ризики, пов'язані з багатопотоковістю, такі як взаємоблокування та змагання за ресурси.

Мова програмування Elixir відома своєю модульністю та розширюваністю. Вона підтримує метапрограмування за допомогою макросів, що дозволяє розробникам генерувати код під час компіляції та створювати доменно-специфічні мови (DSLs) [2]. Це сприяє написанню чистого та виразного коду, який легко підтримувати та розширювати.

Elixir також має багату екосистему, яка включає в себе фреймворк Phoenix для розробки веб-додатків, бібліотеку Ecto для роботи з базами даних та інструмент Mix для управління проектами [3]. Ці інструменти разом з активною спільнотою розробників та великою кількістю доступних бібліотек роблять Elixir привабливим вибором для широкого спектру застосунків.

2.3. Вибір та використання середовища розробки

Neovim є розширеною версією класичного текстового редактора Vim, який відомий своєю ефективністю та гнучкістю. Цей відкритий проект має на меті покращити Vim, зберігаючи сумісність з його плагінами, але також додаючи нові можливості та поліпшуючи інтерфейс для сучасних користувачів.

Neovim пропонує покращену підтримку сучасних графічних інтерфейсів, асинхронне виконання плагінів та підтримку вбудованих терміналів, що робить його більш привабливим для розробників, які шукають більш потужний та налаштовуваний редактор коду. Neovim також покращує роботу з системами контролю версій та інтегрується з різними інструментами розробки, що робить його відмінним вибором для програмістів, які цінують швидкість та ефективність Vim, але потребують розширених функцій сучасних IDE.

Neovim спрямований на спрощення процесу розробки, надаючи потужні інструменти для редагування коду та кастомізації середовища, що дозволяє користувачам створювати ідеальне середовище для своїх потреб. Завдяки активній спільноті та відкритому джерельному коду, Neovim продовжує розвиватися та адаптуватися до нових викликів у світі програмування.

Рис. 2.1. Іконка neovim

Основні особливості

Neovim можна використовувати як основний текстовий редактор для розробки програмного забезпечення, оскільки він підтримує широкий спектр мов програмування та має потужні можливості для редагування коду. Його модульність та налаштовуваність дозволяють користувачам створювати власні робочі середовища з оптимізованими конфігураціями для конкретних завдань або проектів.

Крім того, Neovim є ідеальним інструментом для письменників та редакторів, які віддають перевагу використанню простих текстових файлів для створення вмісту. Завдяки ефективним командам для маніпуляції текстом та можливості розширення функціоналу за допомогою плагінів, Neovim може служити потужним інструментом для написання та редагування.

Neovim також може бути використаний для автоматизації рутинних завдань завдяки його підтримці скриптів та макросів. Розробники можуть створювати скрипти для автоматизації складних редагувань або для інтеграції з іншими інструментами та сервісами, що підвищує продуктивність та зменшує можливість помилок.

Оскільки Neovim підтримує вбудовані термінали, він може використовуватися як централізоване середовище для розробки, де можна одночасно редагувати код, виконувати команди та взаємодіяти з системою контролю версій. Це робить Neovim зручним інструментом для розробників, які віддають перевагу працювати в одному додатку замість перемикання між різними програмами.

Рис. 2.2. Вигляд терміналів neovim

Варто зазначити про підтримку Language Server Protocol (LSP). LSP є стандартом для інтеграції різноманітних інструментів розробки, таких як автодоповнення коду, переходи до визначень, показ документації, лінтингу та рефакторингу, безпосередньо в редакторі коду.

Neovim, починаючи з версії 0.5, включає вбудовану підтримку LSP, що дозволяє розробникам легко підключати та налаштовувати різні LSP-сервери для мов програмування, які використовуються в проекті. Це забезпечує більш ефективний процес розробки, оскільки розробник отримує доступ до розширених можливостей аналізу коду та інших функцій, які зазвичай пропонуються повнофункціональними інтегрованими середовищами розробки (IDE).

Для інтеграції LSP в Neovim було використано плагіни, такі як `nvim-lspconfig`, який спрощує процес налаштування LSP-серверів, надаючи готові конфігурації для багатьох популярних мов. Це дозволило розробнику швидко налаштувати середовище для роботи з конкретними мовами та фреймворками, використовуючи перевірені та оптимізовані конфігурації.

```
sessionStorage.setItem("token", token);
       function fetch(input: RequestInfo | URL, init?: RequestInit): Promise<Response>
await fetch(`/auth`, {
```

Рис. 2.3. Приклад роботи LSP

2.4. Типи (моделі) сутностей вебзастосунку

В застосунку є три типу моделей (сутностей):

- модель User відповідає за логіку звязану з користувачами;
- модель Chat відповідає за логіку звязану з чатами;
- модель Message відповідає за логіку звязану з листуванням.

Сутність "користувачі" містить особисті дані, такі як електронна пошта, псевдонім, а також поля для зберігання пароля та його хешу. Пароль зберігається у віртуальному полі, що означає, що він не зберігається в базі даних, а використовується тимчасово для аутентифікації.

Сутність "чати" включає назву чату та відношення до повідомлень та користувачів. Кожен чат може містити багато повідомлень та бути пов'язаним з одним користувачем, який, можливо, є його створювачем або адміністратором. Також чат може включати багатьох користувачів через багато-до-багатьох відношення, що дозволяє створювати групові чати.

Сутність "повідомлення" містить текст повідомлення та відношення до чату та користувача, що вказує, в якому чаті було надіслано повідомлення та хто є його автором.

Остання сутність "user_chat" є зв'язуючою таблицею, яка визначає відношення багато-до-багатьох між користувачами та чатами. Вона використовується для відстеження, які користувачі є учасниками яких чатів. Ця таблиця має два основні поля, які вказують на конкретного користувача та конкретний чат, і обидва поля виступають як первинні ключі.

Кожна сутність також містить мітки часу створення та останнього оновлення, що дозволяє відстежувати історію змін. заміні або видаленні користувача чи чату, пов'язані записи у зв'язуючій таблиці також будуть видалені.

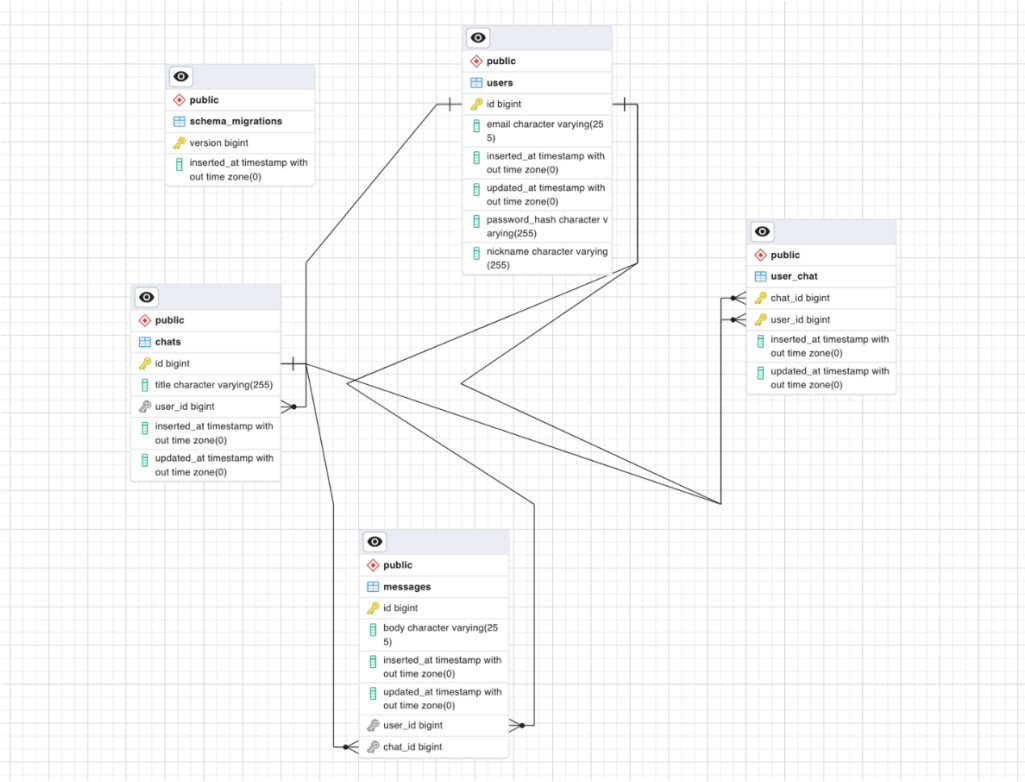

Рис. 2.4. Схема сутностей проекту

2.5. Вибір Elixir фреймворку

Elixir Phoenix є високопродуктивним веб-фреймворком, створеним для розробки надійних та швидких веб-додатків. Він побудований на мові програмування Elixir і використовує потужну віртуальну машину Erlang, що робить його ідеальним для обробки великої кількості одночасних з'єднань і високих навантажень.

Рис. 2.5. Логотип Phoenix Framework

Phoenix вирізняється своєю продуктивністю та ефективністю, завдяки чому він може обслуговувати більше запитів на одиницю апаратних ресурсів порівняно з багатьма іншими веб-фреймворками. Це робить його відмінним вибором для розробки високонавантажених систем, таких як веб-сервіси, системи для обробки потокових даних та інтерактивні додатки в реальному часі [11].

Phoenix також пропонує вбудовану підтримку WebSockets через канали (channels), що дозволяє легко створювати інтерактивні функції, такі як чати, нотифікації та інші реалізації в реальному часі. Ця функціональність робить Phoenix особливо привабливим для сучасних веб-додатків, які потребують швидкого обміну даними між сервером та клієнтом.

Фреймворк також славиться своєю модульністю та розширюваністю, надаючи розробникам гнучкість у виборі потрібних компонентів та інтеграції з іншими інструментами та сервісами. Це дозволяє створювати додатки, які точно відповідають специфічним потребам проекту, без зайвої складності або непотрібного навантаження.

Phoenix також має добре продуману систему для роботи з базами даних, включаючи потужний запитовий інтерфейс, який спрощує створення та управління

запитами до бази даних. Це, разом з чистотою та зрозумілістю коду Elixir, робить процес розробки більш приємним та ефективним [12].

У підсумку, Phoenix на Elixir є відмінним вибором для розробників, які шукають сучасний, надійний та масштабований веб-фреймворк, здатний впоратися з великими обсягами даних та користувачів, забезпечуючи при цьому високу продуктивність та якість розробки.

2.6. Вибір фронт-енд фреймворку

У рамках дипломного проекту, що базується на Elixir та Phoenix Framework, було вирішено використовувати Phoenix LiveView як основну бібліотеку для розробки фронтенду. Цей вибір був зумовлений кількома переконливими перевагами, які LiveView пропонує для створення інтерактивних веб-додатків.

Рис. 2.6. Логотип Phoenix LiveView

Перш за все, Phoenix LiveView дозволяє реалізувати динамічні інтерфейси без написання складного JavaScript-коду. Використовуючи той самий Elixir-код для бекенду та фронтенду, LiveView спрощує процес розробки, дозволяючи розробникам зосередитися на логіці застосунку, а не на синхронізації між двома різними мовами програмування.

Другою важливою перевагою є те, що LiveView використовує постійне з'єднання через WebSockets, що забезпечує швидке оновлення сторінки в реальному часі. Це ідеально підходить для додатків, які потребують високої інтерактивності,

таких як чати, ігри, або системи моніторингу, де користувачі очікують миттєвої реакції на свої дії.

Третім фактором, який вплинув на вибір LiveView, є його висока продуктивність та оптимізація ресурсів. Завдяки ефективному використанню серверних ресурсів та мінімізації кількості переданих даних, LiveView дозволяє створювати швидкі та масштабовані веб-додатки, що є критично важливим для успішного дипломного проекту.

Враховуючи ці аспекти, Phoenix LiveView була обрана як ключова бібліотека для фронтенду дипломного проекту, оскільки вона відповідає сучасним вимогам до розробки інтерактивних веб-додатків, забезпечуючи при цьому високу продуктивність та зручність у використанні [13].

2.7. Вибір бібліотеки для UI компонентів

Petal Components представляє собою набір компонентів інтерфейсу користувача (UI), розроблений спеціально для використання з Elixir Phoenix. Цей UI-кіт включає готові до використання елементи дизайну, які допомагають розробникам швидко та ефективно створювати привабливі та функціональні веб-інтерфейси.

Рис. 2.7. Логотип Petal Components

Однією з ключових переваг Petal Components є його тісна інтеграція з Phoenix та LiveView, що дозволяє розробникам використовувати динамічні можливості Phoenix для створення інтерактивних веб-додатків без необхідності писати складний JavaScript. Компоненти оптимізовані для роботи з LiveView, що робить їх ідеальними для розробки додатків з багатим користувацьким інтерфейсом.

Petal Components відрізняється легкістю використання, оскільки він надає набір стандартизованих компонентів, таких як кнопки, форми, навігаційні панелі та модальні вікна, які можна легко налаштувати та адаптувати до потреб конкретного проекту. Це дозволяє розробникам зосередитися на унікальних аспектах своїх додатків, замість того, щоб витрачати час на створення базових елементів інтерфейсу з нуля.

Крім того, Petal Components підтримує консистентність дизайну, що є важливим для створення професійного та зрозумілого користувацького досвіду. Використання цього UI-кіту допомагає забезпечити єдність стилів та поведінки компонентів по всьому додатку, що сприяє кращій взаємодії з користувачем та знижує криву навчання для кінцевих користувачів.

Набори компонентів інтерфейсу користувача засновані на Tailwind CSS. Tailwind CSS, у свою чергу, є утилітарним фреймворком для стилізації веб-інтерфейсів, який надає набір класів низького рівня, що можуть бути застосовані безпосередньо до HTML елементів для створення індивідуальних дизайнів без потреби писати власний CSS.

Використання Tailwind дозволяє ефективно керувати макетом, відступами, шрифтами, кольорами та іншими аспектами дизайну, використовуючи готові класи. Це сприяє швидкому прототипуванню та ітеративному підходу до дизайну, оскільки зміни можуть бути легко внесені та відображені в реальному часі.

Tailwind також підтримує концепцію "мобільного спочатку" (mobile-first), що є важливим для створення респонсивних веб-додатків, які ефективно працюють на різних пристроях та розмірах екранів. Це дозволяє дипломному проекту бути доступним та зручним для широкої аудиторії користувачів.

Крім того, Tailwind пропонує можливості кастомізації через конфігураційний файл, де розробники можуть визначати свої теми, кольорові палітри та брендові стилі, що забезпечує унікальний вигляд та відчуття дипломного проекту.

Використання Petal Components у проектах на Elixir Phoenix є відмінним вибором для розробників, які прагнуть швидко створити естетично привабливі та функціонально багаті веб-інтерфейси, використовуючи переваги Elixir та Phoenix для бекенду та інтерактивності в реальному часі.

2.8. Вибір бібліотеки для аутентифікації

У контексті кваліфікаційної роботи, розробленого на мові програмування Elixir, було прийнято рішення використовувати бібліотеку Guardian для реалізації аутентифікації користувачів. Вибір Guardian обумовлений кількома ключовими факторами, які роблять її підходящою для вимог проекту.

По-перше, Guardian використовує JSON Web Tokens (JWT), що є стандартом в індустрії для безпечної передачі інформації між клієнтом та сервером у вигляді компактного та самодостатнього JSON-об'єкта. Це забезпечує надійність та гнучкість у процесі аутентифікації, дозволяючи легко масштабувати систему та інтегрувати зі сторонніми сервісами.

По-друге, Guardian надає широкий спектр налаштувань та конфігурацій, що дозволяє детально кастомізувати процес аутентифікації під конкретні потреби проекту. Розробник може визначити власні стратегії перевірки токенів, обробки сесій та відновлення доступу, що забезпечує високий рівень безпеки та контролю над користувацькими сесіями.

По-третє, Guardian інтегрується з Phoenix Framework, що є основою для дипломного проекту, забезпечуючи плавну взаємодію між компонентами веб-застосунку. Ця інтеграція спрощує розробку, оскільки розробникам не потрібно писати додатковий код для зв'язування аутентифікаційної логіки з іншими частинами застосунку [14].

Враховуючи ці переваги, Guardian виявилася оптимальним вибором для забезпечення аутентифікації в дипломному проекті, оскільки вона відповідає високим вимогам до безпеки, масштабованості та гнучкості сучасних веб-застосунків.

2.9. Вибір бібліотеки Websocket

У кваліфікаційній роботі, який полягає у створенні вебзастосунку для листування в режимі реального часу, було прийнято стратегічне рішення використовувати Phoenix Channels [1] для реалізації двостороннього зв'язку між клієнтами та сервером. Phoenix Channels є частиною більш широкої екосистеми Phoenix Framework, яка надає розробникам інструменти для створення високопродуктивних веб-додатків на мові програмування Elixir.

Архітектура Phoenix Channels базується на легковазі Erlang OTP, яка відома своєю здатністю до обробки великої кількості одночасних процесів. Це робить Channels ідеальними для реалізації таких функцій, як миттєві повідомлення, статуси "онлайн/офлайн" користувачів, та інші інтерактивні елементи, які є невід'ємною частиною сучасних чат-додатків.

Phoenix Channels дозволяють розробникам визначати "топіки", які функціонують як канали комунікації. Кожен топік може мати багато підписників і підтримувати різні події, такі як відправлення повідомлень, оновлення статусів користувачів, або навіть складніші взаємодії, такі як ігри в реальному часі. Це дозволяє створювати структуровані та добре організовані веб-додатки, де кожен аспект комунікації має своє місце.

Для забезпечення безпеки, Phoenix Channels використовують механізми аутентифікації та авторизації, що дозволяє контролювати доступ до каналів та відповідних подій. Розробник може впровадити системи перевірки токенів, що забезпечують, що лише авторизовані користувачі можуть під'єднуватися до чатів та взаємодіяти з іншими учасниками.

Порівняно з іншими рішеннями для реалізації вебсокетів, такими як Socket.IO чи WebSockets API, Phoenix Channels вирізняються високою інтеграцією з Elixir та Phoenix, що забезпечує більшу консистентність та ефективність розробки. Вони

також пропонують вбудовані механізми для обробки з'єднань, які втрачено, та автоматичного відновлення з'єднань, що є важливим для підтримки надійності веб-додатку.

Під час розробки вебзастосунку для листування, розробник стикнувся з викликами, пов'язаними з управлінням станом на клієнтській стороні та синхронізацією даних між клієнтами. Phoenix Channels допомогли вирішити ці проблеми, надаючи засоби для ефективного розподілу повідомлень та забезпечення консистентності даних у реальному часі [15].

У другому розділі були описані функціональні та нефункціональні вимоги до застосунку.

Було описано принцип дії, переваги та недоліки відомих аналогів, порівняння аналогів з дипломним проектом.

Програма була розроблена з використанням сучасного стеку технологій, що використовується при створенні вебзастосунків з використанням сучасного середовища розробки neovim та функціональної мови програмування Elixir.

РОЗДІЛ 3
РОЗРОБКА ТА ТЕСТУВАННЯ ЗАСТОСУНКУ

3.1. Структура проекту

Для розробки був використаний фреймворк мови Elixir – Phoenix, який пропонує готове рішення для швидко масштабованих систем.

Команда `mix phx.new` є генератором нових проектів у Phoenix Framework, який використовується для створення нового веб-застосунку з нуля. Ця команда створює стандартну структуру директорій та файлів, необхідних для розробки веб-застосунку на Elixir з використанням Phoenix.

Коли виконується `mix phx.new [назва_проекту]`, Mix, інструмент для управління залежностями та завданнями в Elixir, автоматично генерує всі необхідні файли та конфігурації. Це включає конфігураційні файли, модулі для роутингу, контролери, в'юхи, шаблони та статичні файли, а також тестові файли.

Команда також встановлює залежності за допомогою `mix deps.get` та створює базу даних за допомогою `mix ecto.create`, якщо в проекті використовується Ecto для роботи з базою даних. Крім того, `mix phx.new` надає опції для налаштування проекту, такі як вибір бази даних або створення проекту без Ecto чи без Brunch/Webpack для обробки статичних файлів.

В результаті, команда `mix phx.new` значно спрощує процес ініціалізації нового проекту Phoenix, дозволяючи розробникам швидко перейти до розробки бізнес-логіки та функціональності веб-застосунку.

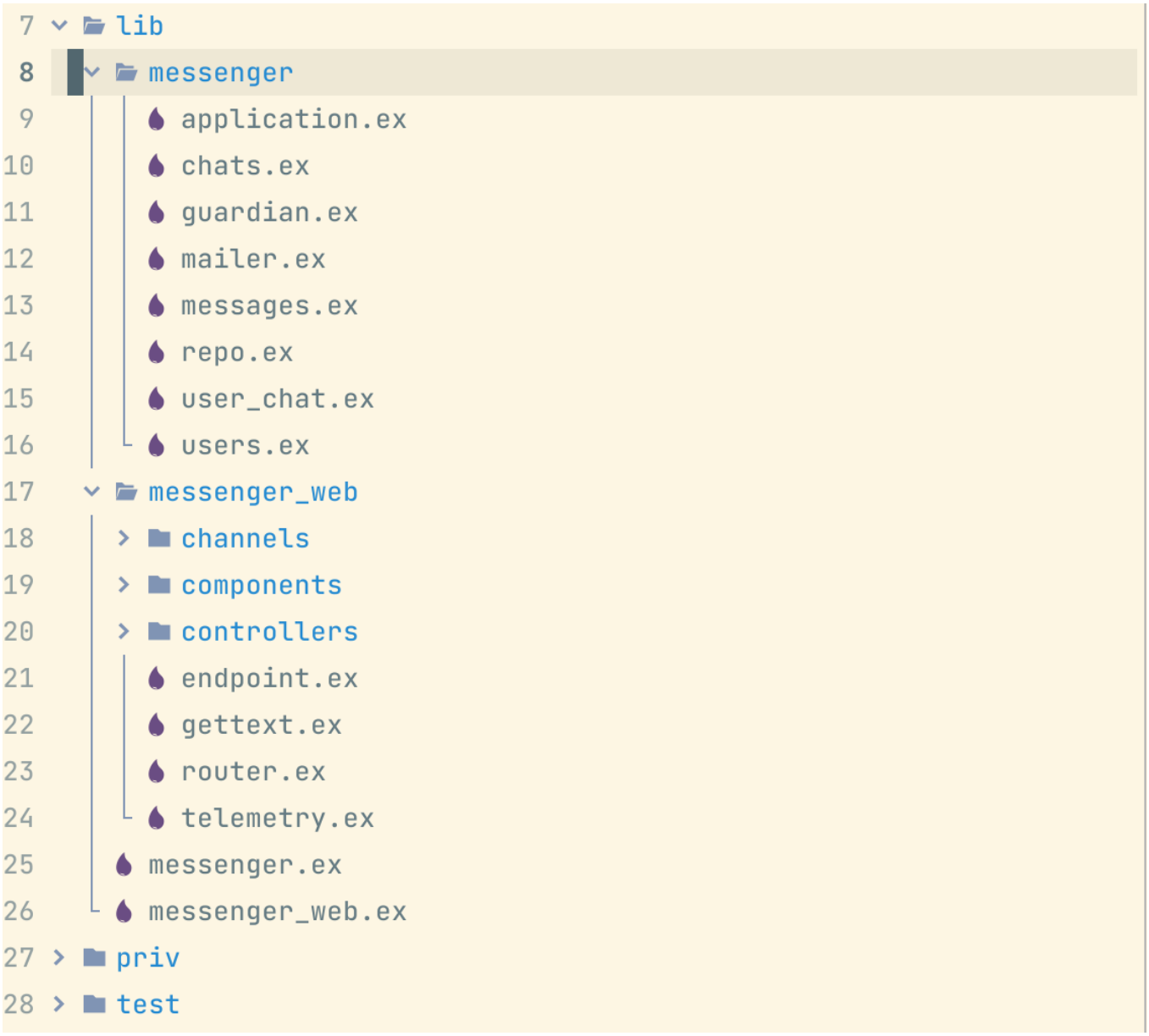

Рис. 3.1. Структура проекту

Структура файлів відповідає за наступне:

- /lib/messenger, папка для зберігання всієї бізнес логіки нашого вебзастосунку, тут можна побачити моделі для користувачів (users.ex), кімнат листування (chats.ex) та листів (messages.ex). Також тут зберігається логіка нашої авторизації (guardian.ex) та процесів (application.ex) які використовуються під час роботи проекту.
- /lib/messenger_web, папка яка використовується для розміщення модулів та файлів, що стосуються веб-частини застосунку. Вона служить контейнером для

коду, який обробляє веб-запити, відповіді, рендеринг шаблонів та інші веб-специфічні задачі.

- /priv, папка яка використовується для зберігання приватних даних, які не повинні бути безпосередньо доступні через веб. Це можуть бути різні ресурси, такі як скрипти для бази даних, конфігураційні файли, статичні файли (якщо вони не призначені для публічного доступу) та інші важливі дані, які використовуються самим застосунком, але не користувачами напряму.
- /test, папка яка відповідає за тестування вебзастосунку, вона містить всі тестові модулі та скрипти, які забезпечують перевірку коректності роботи додатку. Тестування є невід'ємною частиною розробки програмного забезпечення, оскільки воно допомагає забезпечити надійність та стабільність коду перед його розгортанням у виробництво. В папці `/test` зазвичай знаходяться підпапки, які відповідають за різні аспекти тестування. Наприклад, можуть бути підпапки для модульних тестів, інтеграційних тестів, тестів контролерів, каналів та інших компонентів додатку. Модульні тести фокусуються на перевірці окремих функцій та модулів коду, тоді як інтеграційні тести перевіряють взаємодію між різними частинами системи. Кожен файл у папці `/test` зазвичай містить тести для конкретного компонента або функціональності додатку. Файли називаються згідно з конвенцією іменування Elixir, де ім'я тестового файлу відповідає імені модуля, який він тестує, з додаванням суфіксу `_test.exs`. Це дозволяє легко ідентифікувати тести та асоціювати їх з відповідними частинами додатку.

3.2. Створення моделей за допомогою Elixir Ecto

Модель User:

```
schema "users" do
  field(:email, :string)
  field(:nickname, :string)
  field(:password, :string, virtual: true)
  field(:password_hash, :string)
```

```
  many_to_many(:chats, Messenger.Chat, join_through: Messenger.UserChat, on_replace:
:delete)

  timestamps()
end
```

Модель Chat:

```
schema "chats" do
  field :title, :string

  has_many(:messages, Messenger.Message)
  belongs_to(:user, Messenger.User)
  many_to_many(:users, Messenger.User, join_through: Messenger.UserChat, on_replace:
:delete)

  timestamps()
end
```

Модель Message:

```
schema "messages" do
  field :body, :string

  belongs_to(:chat, Messenger.Chat)
  belongs_to(:user, Messenger.User)

  timestamps()
end
```

Приклад міграції для моделі користувача за допомогою Elixir Ecto:

```
defmodule Messenger.Repo.Migrations.CreateUsers do
 use Ecto.Migration

 def change do
  create table(:users) do
  add :email, :string

  timestamps()
 end
end
```

Цей код визначає модуль міграції для Ecto, який є частиною Elixir бібліотеки для роботи з базами даних. Модуль `Messenger.Repo.Migrations.CreateUsers` використовується для створення нової таблиці в базі даних, яка призначена для зберігання даних користувачів.

Ключові моменти коду:

- `defmodule Messenger.Repo.Migrations.CreateUsers do`: Оголошення нового модуля міграції, який називається `CreateUsers` і належить до контексту `Messenger.Repo.Migrations`;
- `use Ecto.Migration`: Включення функціональності Ecto.Migration в модуль, що надає набір макросів та функцій для визначення міграцій;
- `def change do`: Визначення функції `change`, яка використовується Ecto для застосування або відкату міграції. У цьому випадку, функція `change` використовується для опису змін, які мають бути зроблені в базі даних;
- `create table(:users) do`: Використання макросу `create` для створення нової таблиці з назвою `users`;
- `add :email, :string`: Додавання стовпця `email` до таблиці `users`, який буде зберігати рядкові дані (тип `:string`);

- `timestamps()`: Додавання стандартних стовпців `inserted_at` та `updated_at` до таблиці, які автоматично відстежують час створення та останнього оновлення запису.

Ця міграція, коли вона застосована, створить у базі даних таблицю `users` з двома полями: `email` для зберігання електронної пошти користувача та автоматично генерованими полями для відстеження часу створення та оновлення записів.

Також можна побачити, що весь SQL синтаксис записаний у мові Elixir завдяки Object-Relational Mapping від бібліотеки Ecto, яка дозволяє абстрагуватись від оригінального синтаксису та використовувати синтаксис Elixir в роботі з базами даних.

Кожна з моделей має набір своїх методів завдяки яким, ми можемо з ними спілкуватись. Наприклад модель User має метод create_user/1 який в свою чергу використовує два методи – create_user_changeset/2 та Repo.insert/1

```
def create_user(user) do
  changeset = create_user_changeset(%User{}, user)
  Repo.insert(changeset)
end
```

create_user_changeset/2 це метод, який використовує методи бібліотеки Ecto для перевіряє вхідних даних функції та наш метод для хешування паролю. В нашому випадку, ми перевіряємо декілька пунктів:

- щоб емейл, нікнейм та пароль були присутні у вхідних даних;

- щоб емейл був унікальним;

- щоб мінімальна довжина паролю була не менш за 2 символа.

І також хешуємо пароль і забираємо його з структури даних для безпечної передачі в місце, де функція була викликана.

3.3. Налаштування Websocket

Модуль `MessengerWeb.Socket` визначає сокет, який використовується для управління веб-сокет з'єднаннями в Phoenix Framework. Цей модуль відповідає за

встановлення з'єднання між клієнтом та сервером та маршрутизацію повідомлень до відповідних каналів.

Функція `connect/3` визначає, як користувачі можуть підключатися до сокета. У даному випадку, вона просто приймає всі з'єднання без будь-якої перевірки або аутентифікації, повертаючи `{:ok, socket}`, що означає успішне підключення. У реальному застосунку тут може бути логіка для перевірки параметрів з'єднання або аутентифікації користувачів.

Далі, модуль визначає два канали: "chat:*" та "user:*". Кожен канал відповідає за обробку повідомлень, пов'язаних з певною темою. Зірочка (*) у назві каналу означає, що канал може обробляти повідомлення для будь-якого топіка, який відповідає даному шаблону. Наприклад, "chat:lobby" або "chat:room123" будуть маршрутизовані до `MessengerWeb.ChatChannel`.

`MessengerWeb.ChatChannel` відповідає за обробку повідомлень, пов'язаних з чатами, таких як відправлення та отримання повідомлень у чат-кімнатах. `MessengerWeb.UserChannel` може обробляти повідомлення, пов'язані з діями користувачів, наприклад, оновлення статусу користувача або відстеження присутності користувачів у системі.

```
defmodule MessengerWeb.Socket do
  def connect(_params, socket, _connect_info) do
    {:ok, socket}
  end

  channel "chat:*", MessengerWeb.ChatChannel
  channel "user:*", MessengerWeb.UserChannel
end
```

Наступним кроком створюються модулі які використовується для створення каналу комунікації в реальному часі між користувачами. Модуль визначає логіку приєднання до каналу, обробки вхідних повідомлень та авторизації користувачів.

```
defmodule MessengerWeb.UserChannel do
  use MessengerWeb, :channel

  @impl true
  def join("user:" <> _room_id, payload, socket) do
    if authorized?(payload) do
      {:ok, socket}
    else
      {:error, %{reason: "unauthorized"}}
    end
  end

  @impl true
  def handle_in("ping", payload, socket) do
    {:reply, {:ok, payload}, socket}
  end

  @impl true
  def handle_in("shout", payload, socket) do
    broadcast(socket, "shout", payload)
    {:noreply, socket}
  end

  # Add authorization logic here as required.
  defp authorized?(payload) do
```

```
  with {:ok, user, _claims} <- Messenger.Guardian.resource_from_token(payload.token)
do
    true
  else
    false
  end
 end
end
```

Функція `join/3` визначає процес приєднання користувача до каналу. Вона перевіряє, чи користувач має право на доступ до каналу, використовуючи приватну функцію `authorized?/1`. Якщо користувач авторизований, він успішно приєднується до каналу, інакше він отримує повідомлення про помилку з причиною "unauthorized".

Функція `handle_in/3` зі зразком "ping" використовується для демонстрації запиту/відповіді між клієнтом та сервером. Коли сервер отримує повідомлення "ping" від клієнта, він відправляє відповідь назад з тим же вмістом у вигляді "pong".

Інша функція `handle_in/3` зі зразком "shout" використовується для розсилки повідомлень всім користувачам, які підключені до каналу. Коли повідомлення "shout" отримане, воно транслюється всім підписникам каналу, дозволяючи реалізувати функціональність групового чату.

Функція `authorized?/1` використовується для перевірки, чи має користувач валідний токен для авторизації. Це забезпечує додатковий рівень безпеки, переконуючись, що лише авторизовані користувачі можуть взаємодіяти з каналом.

Наступним кроком реалізовано підписку на канали Phoenix для користувача та розсилку повідомлень у чаті.

```
  if length(user.chats) > 0 do
    for i <- 0..(length(user.chats) - 1) do
      MessengerWeb.Endpoint.subscribe("room:#{Enum.at(user.chats, i).id}")
    end
```

```
  end


MessengerWeb.Endpoint.subscribe("user:#{user.id}")
```

Спочатку код перевіряє, чи є у користувача активні чати, використовуючи функцію `length/1` для визначення кількості чатів, асоційованих з користувачем. Якщо у користувача є хоча б один чат, код ітерує через кожен чат за допомогою циклу `for` та підписується на відповідний топік каналу, використовуючи унікальний ідентифікатор чату. Це дозволяє користувачу отримувати оновлення в реальному часі для кожного чату, до якого він приєднаний.

Далі, код підписує користувача на окремий канал, який використовується для персональних повідомлень та сповіщень, використовуючи ідентифікатор користувача для створення унікального топіка.

Нарешті, код використовує функцію `broadcast!/3` для розсилки нового повідомлення всім підписникам конкретного чату. Це робиться шляхом відправлення повідомлення з ключем "new_msg" та даними нового повідомлення до топіка, який відповідає поточному чату користувача.

Варто зазначити обробку повідомлень про створення нових чатів за допомогою Phoenix Channels. Функція `handle_info/2` визначена для обробки вхідних повідомлень, які містять інформацію про новостворені чати.

```
def handle_info(
      %Phoenix.Socket.Broadcast{
        event: "new_chat" = _event,
        payload: new_chat,
        topic: _topic
      },
      socket
    ) do
  MessengerWeb.Endpoint.subscribe("room:#{new_chat.id}")
```

```
  {:noreply,
   assign(socket,
     current_user:
       Map.put(
         socket.assigns.current_user,
         :chats,
         socket.assigns.current_user.chats ++ [new_chat]
       )
   )}
end
```

Коли користувач отримує повідомлення з подією "new_chat", ця функція активується. Вона використовує `MessengerWeb.Endpoint.subscribe/1` для підписки на тему, яка відповідає ідентифікатору нового чату, дозволяючи користувачу отримувати майбутні повідомлення, пов'язані з цим чатом.

Після підписки, функція оновлює сокет, додаючи новий чат до списку чатів поточного користувача. Це робиться шляхом зміни стану сокета за допомогою функції `assign/3`, яка додає новий чат до списку `:chats` у структурі даних `current_user`, що зберігається в `socket.assigns`.

Таким чином, функція `handle_info/2` забезпечує динамічне оновлення інтерфейсу користувача без необхідності перезавантаження сторінки, дозволяючи користувачам бачити нові чати відразу після їх створення.

3.3. Створення UI

Процес створення UI розпочинається з визначення структури сторінок та компонентів, які були необхідні для взаємодії з користувачем. Для цього використовується декларативний синтаксис LiveView для опису HTML-шаблонів, які динамічно змінюються в залежності від стану сервера. Це дозволяє створити

інтуїтивно зрозумілі форми, меню та інші інтерфейсні елементи, які відповідають на дії користувача без перезавантаження сторінки.

Для обробки подій, таких як кліки по кнопках, введення даних у форми чи навігація по додатку, були використані функції обробки подій LiveView. Ці функції викликаються безпосередньо на сервері, що дозволяє розробнику легко взаємодіяти з даними та логікою застосунку. В результаті, користувачі отримали можливість взаємодіяти з додатком в режимі реального часу, отримуючи швидкі та зрозумілі відповіді на свої дії.

На прикладі сторінки авторизації можна побачити методи життєвого циклу сторінки та методи обробки подій

```
defmodule MessengerWeb.LoginLive do
 use MessengerWeb, :live_view
 import MessengerWeb.Header

 def mount(_params, _session, socket) do
  {:ok,
   assign(
    socket,
    form: to_form(Messenger.User.create_user_changeset(%Messenger.User{}, %{})),
    error: nil
   )}
 end

 def handle_event("validate", %{"user" => user} = _params, socket) do
  form =
   %Messenger.User{}
   |> Messenger.User.login_user_changeset(user)
   |> Map.put(:action, :insert)
   |> to_form()
```

```
    {:noreply, assign(socket, form: form)}
  end

  def handle_event("save", %{"user" => user} = _params, socket) do
    with {:ok, user} <- Messenger.User.login(%{email: user["email"], password:
user["password"]}),
         {:ok, token, _claims} <- Messenger.Guardian.encode_and_sign(user) do
      {:noreply, push_event(socket, "setSession", %{token: token})}
    else
      {:error, error} ->
        {:noreply, assign(socket, error: error)}
    end
  end

  def handle_event("sign_up", _params, socket) do
    {:noreply, push_navigate(socket, to: "/register")}
  end

  def handle_event("redirect_to_chats", _params, socket) do
    {:noreply, push_navigate(socket, to: "/chats")}
  end
end
```

Важливо розуміти життєвий цикл LiveView для ефективної розробки інтерактивних веб-додатків. Життєвий цикл LiveView описує послідовність подій, які відбуваються від моменту запуску LiveView на сервері до взаємодії з користувачем та оновлення стану.

На початку життєвого циклу, коли користувач вперше завантажує сторінку, ініціюється процес LiveView, який запускає функцію `mount/3`. Ця функція

відповідає за ініціалізацію стану LiveView та підготовку початкового рендерингу HTML. Вона може також обробляти параметри сесії та запиту, що надходять від клієнта.

Після ініціалізації стану, LiveView відправляє HTML до клієнта через веб-сокет, використовуючи мінімально необхідну кількість даних. Коли користувач взаємодіє з інтерфейсом, наприклад, клікає на кнопку або вводить дані у форму, події відправляються назад на сервер.

На сервері події обробляються відповідними обробниками подій, які можуть оновлювати стан LiveView. Після кожного оновлення стану, LiveView рендерить новий HTML, який відображає оновлений стан, і відправляє його клієнту. Цей процес відбувається швидко та ефективно, забезпечуючи користувачам відчуття неперервної взаємодії.

Для оптимізації продуктивності, LiveView використовує механізм розумного патчингу, який відправляє лише ті частини HTML, які змінилися, замість повного перерендерингу сторінки. Це знижує навантаження на мережу та покращує швидкість відгуку додатку.

Також було важливо забезпечити ефективну обробку подій, що відбуваються на клієнтській стороні. Для цього використовується функція `handle_event`, яка є ключовою частиною LiveView і дозволяє реагувати на різноманітні події, такі як кліки мишею, введення даних у форми чи навіть зміни стану в інтерфейсі користувача.

Функція `handle_event` приймає як аргументи назву події, дані, пов'язані з подією, та структуру стану LiveView. Це дозволяє розробнику визначити специфічну логіку, яка буде виконуватися при виникненні певної події. Наприклад, при натисканні на кнопку відправки форми, `handle_event` може обробити введені дані, виконати валідацію та надіслати їх до сервера для подальшої обробки.

Можна побачити що в цьому коді оброблюється зберігання форми авторизації користувача та перенапревлення його на інші сторінки

HTML зберігається в іншому файлі з розширенням heex, що розшифровується як HTML Embedded Elixir, є новим форматом шаблонів, який був представлений у

Phoenix версії 1.6. Він призначений для покращення процесу розробки веб-інтерфейсів у Phoenix за допомогою вбудовування Elixir коду безпосередньо в HTML шаблони.

```
<.auth_header type="login" />
<div class="flex flex-col items-center h-[calc(100vh-45px)] justify-center">
  <.form for={@form} phx-change="validate" phx-submit="save" phx-hook="Session"
id="login_form" class="block max-w-[30rem] w-full px-4">
    <.field required type="email" field={@form[:email]} label="Email" placeholder="Your
email" class="mb-2" label_class="text-neutral-600 font-bold" wrapper_class="mb-2"/>
    <.field required type="password" field={@form[:password]} placeholder="Your
password" class="mb-2" label_class="text-neutral-600 font-bold"
    wrapper_class="mb-5"/>
    <div class="mb-3 bg-secondary-600 text-secondary-100 px-4 py-2 rounded-xl text-sm
text-center" :if={@error}><%= @error %></div>
    <.button type="submit" class="submit-button" class="w-full"
disabled={!@form.source.valid?}>Login</.button>
  </.form>
</div>
```

Компонент `<.auth_header type="login" />` викликає визначений користувацький компонент, який, ймовірно, відображає заголовок аутентифікації для сторінки входу.

Основний контейнер `<div>` використовує класи Tailwind CSS для стилізації та розміщення форми входу в центрі сторінки. Висота контейнера встановлена так, щоб він займав увесь доступний простір вікна браузера за винятком висоти заголовка.

Компонент `<.form>` створює форму для входу в систему, яка реагує на події валідації (`phx-change="validate"`) та збереження (`phx-submit="save"`). Атрибут `phx-hook="Session"` вказує на використання JavaScript-хука для обробки подій форми на клієнтській стороні.

Компоненти `<.field>` визначають поля вводу для електронної пошти та пароля. Вони включають атрибути, такі як `required`, `type`, `field`, `label`, `placeholder` та класи для стилізації. Ці поля використовуються для збору даних користувача, необхідних для входу в систему.

Блок `<div>` з класом `bg-secondary-600` відображає повідомлення про помилку, якщо таке існує в контексті `@error`. Це повідомлення показується користувачеві у випадку невдалої спроби входу.

Компонент `<.button>` створює кнопку входу, яка стає активною тільки тоді, коли форма валідна (перевірка виконується через `@form.source.valid?`).

3.4. Автентифікація та сессія

```
def handle_event("save", %{"user" => user} = _params, socket) do with {:ok, user} <-
Messenger.User.login(%{email: user["email"], password: user["password"]}),
{:ok, token, _claims} <- Messenger.Guardian.encode_and_sign(user) do
{:noreply, push_event(socket, "setSession", %{token: token})}
else
{:error, error} ->
{:noreply, assign(socket, error: error)}
end end Hooks.Session = {
mounted() {
this.handleEvent("setSession", async ({ token }) => {
sessionStorage.setItem("token", token);

await fetch(`/auth`, {
method: "post",
body: JSON.stringify({ token }),
credentials: "include",
headers: {
"x-csrf-token": csrfToken,
```

```
"Content-Type": "application/json",
},
})
.then((res) => {
return res.json();
})
.then((res) => {
this.pushEvent("redirect_to_chats", {});
});
});

this.handleEvent("removeSession", async () => {
sessionStorage.removeItem("token");

await fetch(`/logout`, {
  method: "post",
  body: JSON.stringify({}),
  credentials: "include",
  headers: {
    "x-csrf-token": csrfToken,
    "Content-Type": "application/json",
   },
})
.then((res) => {
   return res.json();
})
.then((res) => {
  console.log(res);
  this.pushEvent("redirect_to_login", {});
});
```

```
});

def authorize(conn, _params) do token = conn.body_params["token"]

with {:ok, user, _claims} <- Messenger.Guardian.resource_from_token(token) do conn
  |> put_session(:current_user, user)
  |> json(%{
  authorized: true,
  redirect_path: MessengerWeb.Router.Helpers.live_path(conn,
MessengerWeb.ChatsLive)
})
else
{:error, reason} -> conn |> json(%{authorized: false, reason: reason})
end end

def logout(conn, _params) do conn
|> put_session(:current_user, nil)
|> json(%{authorized: false})
end
```

Цей код описує процес логіну користувача, збереження сесії та вихід з системи. Функція `handle_event/3` в модулі сервера обробляє подію "save", яка спрацьовує, коли користувач намагається увійти в систему. Вона приймає параметри від клієнта, включаючи дані користувача, і спробує авторизувати користувача за допомогою функції `Messenger.User.login/1`. Якщо логін успішний, вона генерує токен за допомогою `Messenger.Guardian.encode_and_sign/1` і відправляє його назад на клієнтську сторону через подію "setSession". У разі помилки, вона надсилає повідомлення про помилку на сокет.

На клієнтській стороні, JavaScript-хук `Session` відповідає за обробку подій "setSession" та "removeSession". При отриманні події "setSession", хук зберігає токен у sessionStorage і виконує POST-запит на ендпойнт `/auth` для встановлення сесії на сервері. Після успішної відповіді, він ініціює перенаправлення користувача на сторінку чатів. Подія "removeSession" видаляє токен з sessionStorage і виконує POST-запит на ендпойнт `/logout` для очищення сесії на сервері, після чого користувача перенаправляють на сторінку логіну.

Функція `authorize/2` на сервері обробляє запит на ендпойнт `/auth`, перевіряючи токен і встановлюючи сесію для користувача. Якщо токен валідний, вона відправляє відповідь з підтвердженням авторизації та шляхом для перенаправлення. У разі помилки, вона відправляє відповідь з інформацією про невдалу авторизацію.

Функція `logout/2` обробляє запит на ендпойнт `/logout`, видаляючи інформацію про користувача з сесії та відправляючи відповідь, що користувач більше не авторизований.

3.5. Принцип роботи застосунку

Use Case (випадок використання) — це опис конкретної ситуації або сценарію, який демонструє, як користувач або інша система взаємодіє з вашим застосунком для досягнення певної мети. Він допомагає визначити функціональні вимоги та очікувану поведінку системи. Далі будуть розглянуті use cases вебзастосунку:

Проходження реєстрації

Короткий опис – даний Use Case реалізує процес реєстрації користувача в вебзастосунку.

Цілі – ввести особисті дані користувача на сторінці реєстрації та отримати можливість увійти в чат і побачити головну сторінку.

Актори – незареєстрований користувач

Основний сценарій:

1) користувач заходить на сторінку реєстрації;

2) користувач вводить дані у форматі, що вимагає система;

3) користувач отримує повідомлення про успішну реєстрацію;

4) користувач перенаправляється на головну сторінку.

Альтернативні сценарії (для п. 3):

а) дані введені у неправильному форматі – користувач отримує повідомлення про невідповідність даних та необхідність повторного вводу;

б) введений емейл вже існує в чаті – користувач отримує повідомлення про існування емейлу та необхідність обрати інший.

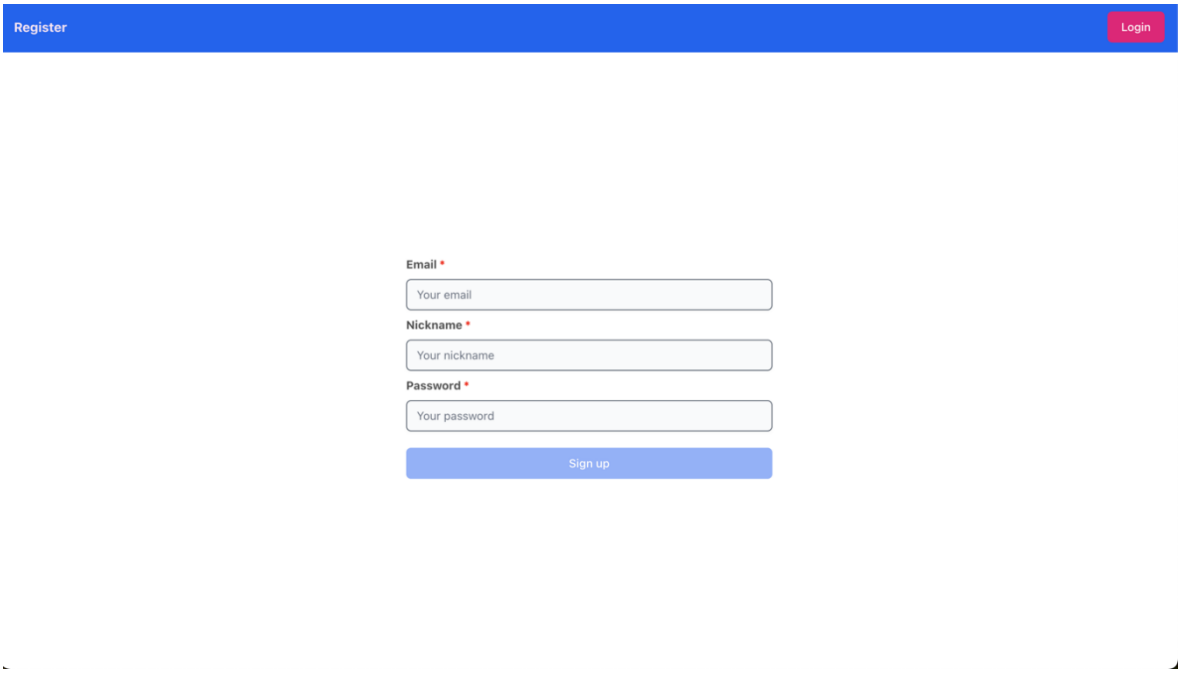

Рис. 3.2. Сторінка реєстрації

Вхід у чат

Короткий опис – даний Use Case реалізує процес входу користувача в онлайн чат.

Цілі – ввести особисті дані на сторінці входу та отримати можливість увійти в чат і побачити головну сторінку.

Актори – незареєстрований користувач

Основний сценарій:

1) користувач заходить на сторінку входу;

2) користувач вводить особисті дані;

3) користувач потрапляє на головну сторінку.

Альтернативні сценарії (для п. 3):

а) дані не знайдені в базі даних – користувач отримує повідомлення про невірно введені дані, залишаючись на сторінці логіну

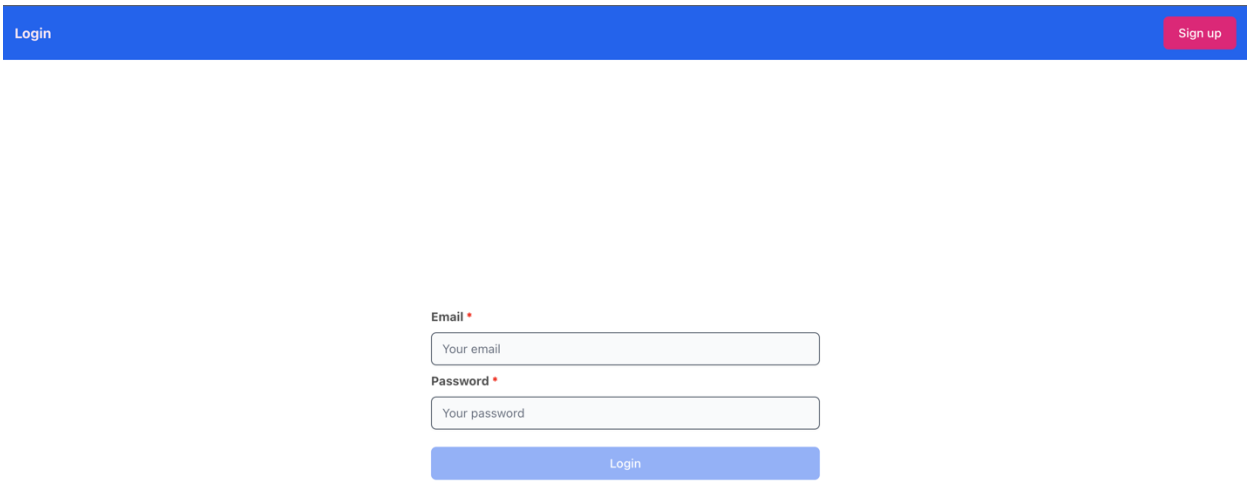

Рис. 3.3. Сторінка авторизації

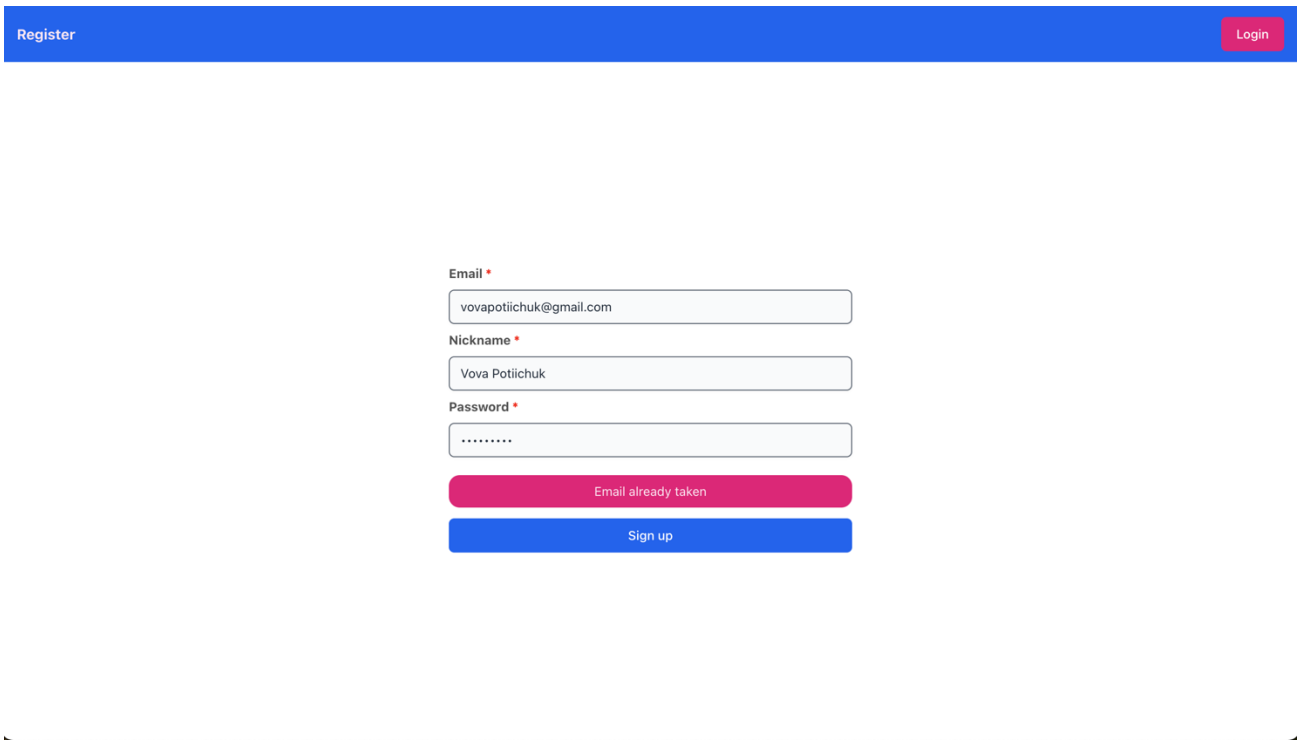

Рис. 3.4. Сторінка авторизації у випадку невдалої реєстрації

Вихід з чату

Короткий опис – даний Use Case реалізує процес виходу користувача з онлайн чату.

Цілі – натиснути кнопку виходу та потрапити на сторінку входу.

Актори – зареєстрований користувач

Основний сценарій:

1) користувач заходить на головну сторінку;
2) користувач натискає кнопку виходу;
3) користувач потрапляє на сторінку входу.

Рис. 3.5. Кнопка виходу з чату

Надсилання повідомлення

Короткий опис – даний Use Case реалізує процес надсилання користувачем повідомлення в онлайн чат.

Цілі – надіслати повідомлення та побачити його в списку повідомлення на головній сторінці.

Актори – зареєстрований користувач, отримувач повідомлення

Основний сценарій:

1) користувач заходить на головну сторінку;
2) користувач набирає повідомлення та натискає кнопку відправки;
3) користувач бачить щойно надіслане повідомлення у списку повідомлень.

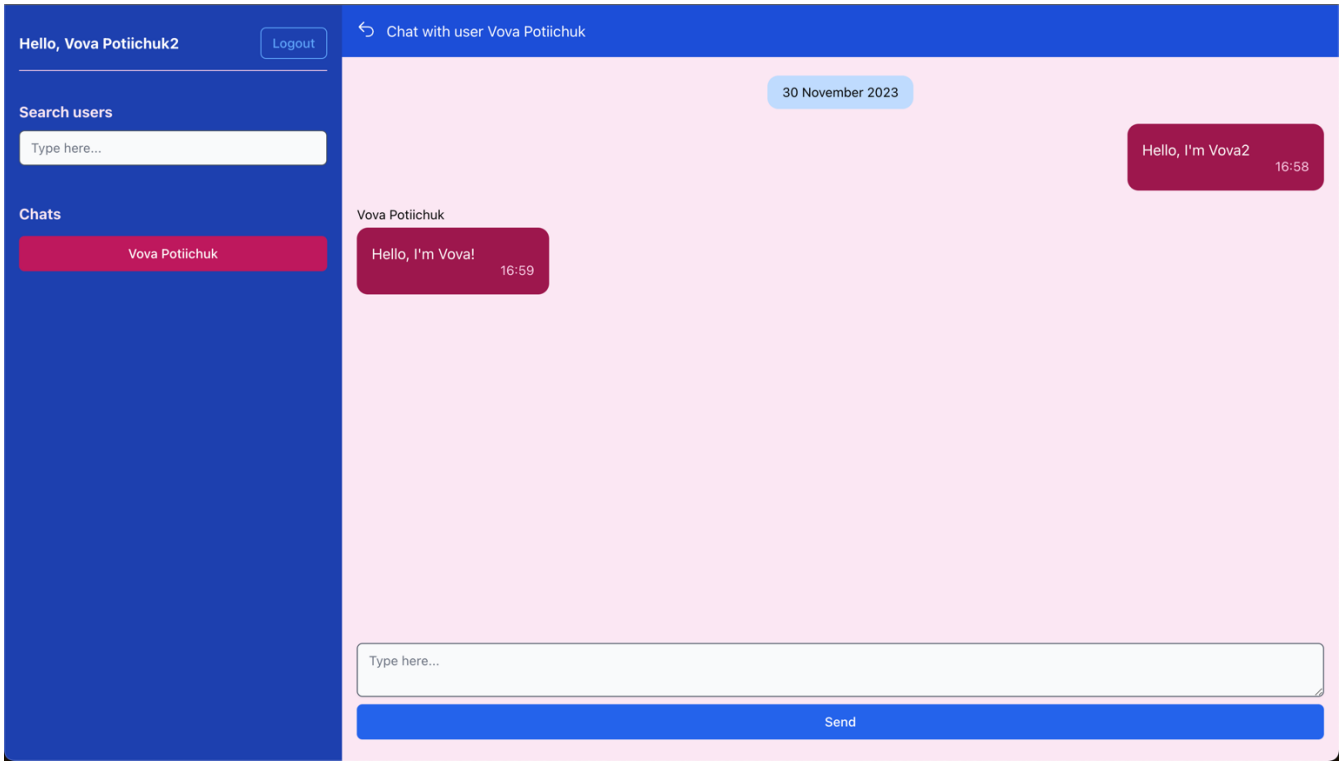

Рис. 3.6. Головний екран комунікації

Отримання повідомлення

Короткий опис – даний Use Case реалізує процес отримання користувачем повідомлення в онлайн чаті.

Цілі – побачити в списку повідомлення, що було надіслане в чат.

Актори – зареєстрований користувач, відправник повідомлення

Основний сценарій:

1) користувач заходить на головну сторінку;
2) В чат надсилається повідомлення;
3) користувач бачить щойно надіслане повідомлення у списку повідомлень.

Пошук користувачів

Короткий опис – даний Use Case реалізує процес пошуку інших користувачів зареєстрованих у вебзастосунку.

Цілі – знайти користувачів для відправки повідомлень

Актори – зареєстрований користувач, інші користувачі

Основний сценарій:

1) користувач заходить на головну сторінку;
2) в пошуковому полі вводиться емейл або нікнейм користувача
3) користувач бачить список користувачів по заданим крітеріям

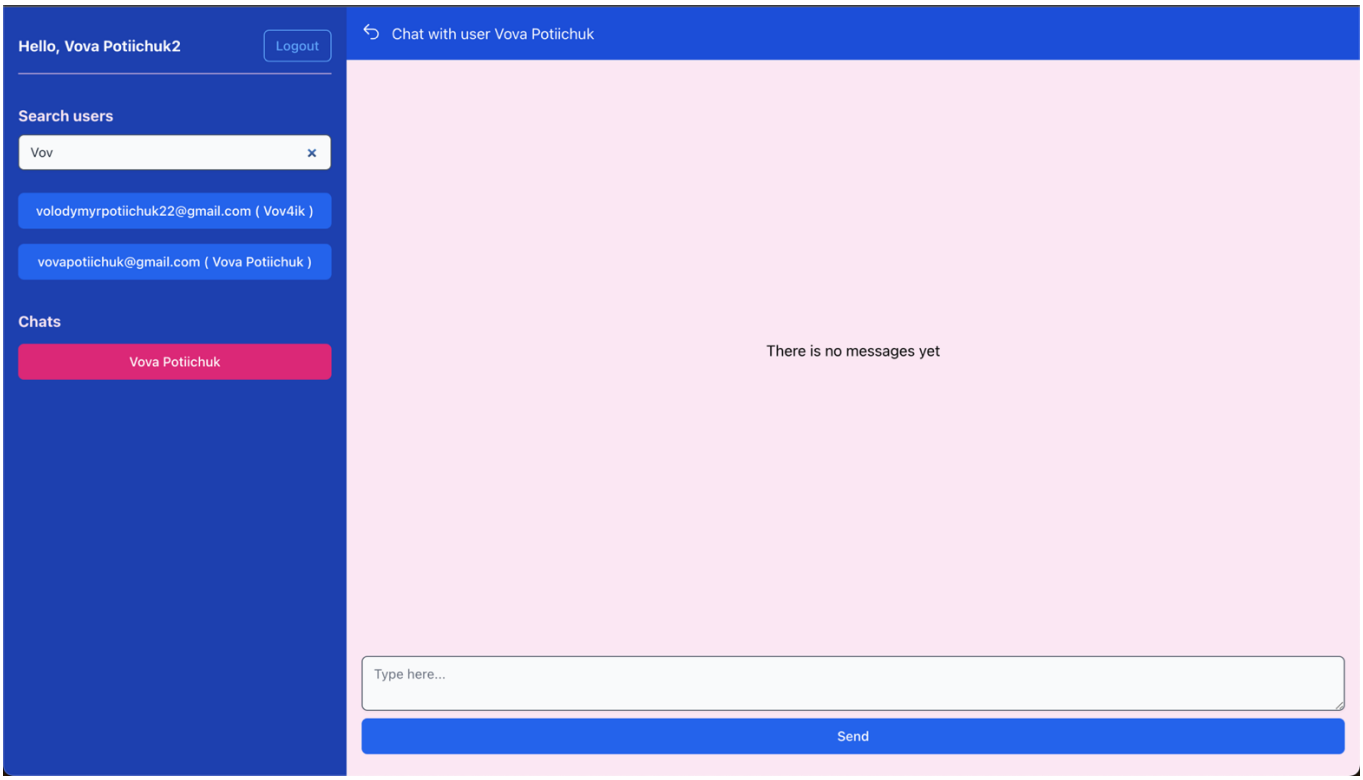

Рис. 3.7. Вікно пошуку інших користувачів

3.6. Мобільна адаптація

У сучасному цифровому світі, де значна частина користувачів доступається до веб-застосунків через мобільні пристрої, адаптація інтерфейсу під різні розміри екранів є критично важливою. У дипломному проекті було враховано цю тенденцію завдяки Tailwind CSS.

Tailwind CSS надає набір готових до використання класів, які можуть бути застосовані безпосередньо до HTML-елементів для швидкої та ефективної стилізації. Ці класи включають у себе різноманітні утиліти для налаштування макету, розмірів, шрифтів, кольорів та інших візуальних аспектів інтерфейсу. Завдяки системі класів, які реагують на зміну розміру екрану (наприклад, `sm:`, `md:`, `lg:`, `xl:`), розробники можуть легко створювати дизайн, який адаптується до різних пристроїв, забезпечуючи оптимальний вигляд та користувацький досвід на будь-якому пристрої.

Налаштування Tailwind CSS описуються як:

```
const plugin = require("tailwindcss/plugin");
const colors = require("tailwindcss/colors");
const fs = require("fs");
const path = require("path");

module.exports = {
  content: [
    "./js/**/*.js",
    "../lib/*_web.ex",
    "../lib/*_web/**/*.*ex",
    "../deps/petal_components/**/*.*ex",
  ],
  theme: {
    extend: {
      colors: {
        primary: colors.blue,
```

```
      secondary: colors.pink,
      success: colors.green,
      danger: colors.red,
      warning: colors.yellow,
      info: colors.sky,
      gray: colors.gray,
    },
  },
 },
 plugins: [
  require("@tailwindcss/forms")
 ]
}
```

фрагмент HTML-коду з використанням Tailwind CSS демонструє приклад адаптивної верстки у дипломному проекті. Він визначає блок (div), який адаптує своє розташування та розміри залежно від ширини екрану пристрою, на якому відображається веб-сторінка.

```
 <div class={"w-full sm:w-[calc(100vw-250px)] absolute right-0 top-0 left-0 bottom-0 duration-150 #{if !@active_chat_id, do: "translate-x-[100vw]"} sm:static sm:translate-x-0"}>
  <.live_component module={MessengerWeb.ActiveChatLive} id="active_chat" active_chat_id={@active_chat_id} current_user_id={@current_user.id}/>
 </div>
```

Клас `w-full` встановлює ширину блока на 100% від ширини батьківського елемента, що забезпечує його розтягування на всю доступну ширину. У медіа-запиті для екранів середнього розміру (`sm:`), використовується користувацька ширина `sm:w-[calc(100vw-250px)]`, яка вираховується як повна ширина екрану (`100vw`)

мінус 250 пікселів, дозволяючи блоку займати простір, що залишився після бічної панелі або іншого елемента шириною 250 пікселів.

Клас `absolute` позиціонує див абсолютно відносно батьківського елемента, який має відносне (`relative`) позиціонування. Класи `top-0`, `right-0`, `left-0`, `bottom-0` встановлюють позицію дива точно на верхній, правій, лівій та нижній межах батьківського елемента. Клас `duration-150` визначає тривалість анімації переходу, яка становить 150 мілісекунд.

Умовний вираз `#{if !@active_chat_id, do: "translate-x-[100vw]"}` застосовує CSS-трансформацію, яка переміщує блок на 100% ширини екрану вправо, ефективно ховаючи його з поля зору, якщо умова `!@active_chat_id` (не існує активного ідентифікатора чату) є істинною. Для екранів середнього розміру та більше (`sm:`), ця трансформація скасовується (`sm:static sm:translate-x-0`), і блок стає статично позиціонованим без зміщення.

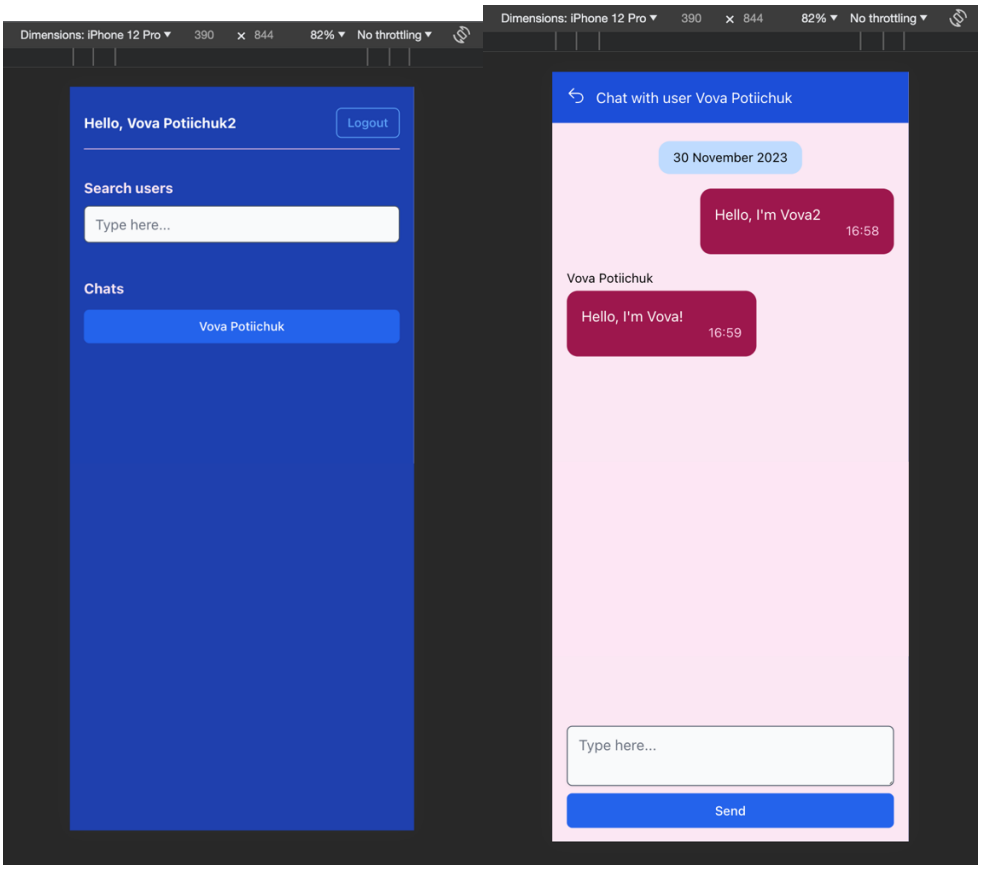

Рис. 3.8. Вигляд головного екрану чату з мобільного пристрою

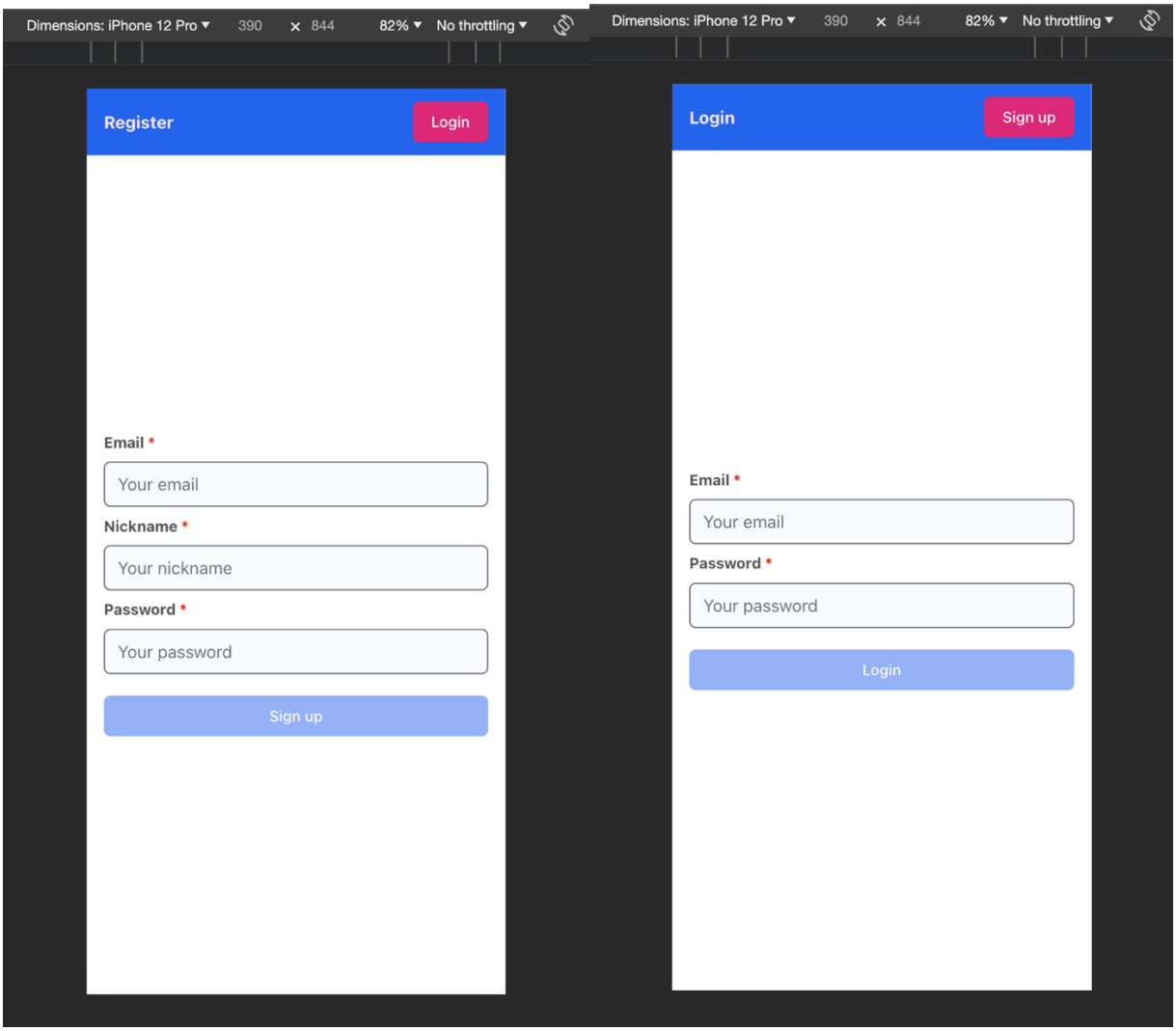

Рис. 3.9. Вигляд сторінок реєстрації та авторизації з мобільного пристрою

3.7. Тестування сутностей

Тестування сутностей відбувається за допомогою фреймворку ExUnit. ExUnit є вбудованим тестовим фреймворком в Elixir, який надає необхідні інструменти для модульного тестування коду. Використання ExUnit дозволяє розробнику ефективно перевіряти коректність логіки сутностей та їх взаємодії з іншими частинами системи.

Процес тестування починається з написання тестових сценаріїв, які відображають очікувану поведінку сутностей. Розробник визначає набір вхідних даних та очікуваних результатів для кожної функції сутності. За допомогою асерцій, які є частиною ExUnit, перевірялася відповідність фактичної поведінки сутностей до очікуваної.

Тести охоплюють різні аспекти сутностей, включаючи валідацію даних, збереження та витягування інформації з бази даних, а також взаємодію між різними сутностями. Це забезпечило високий рівень впевненості в тому, що сутності працюють належним чином та відповідають вимогам бізнес-логіки.

ExUnit також надав можливість налаштування тестового середовища перед кожним тестовим випадком та очищення його після завершення тесту, що дозволило ізолювати тести та забезпечити їх незалежність. Це важливо для отримання консистентних тестових результатів та уникнення побічних ефектів.

Завдяки тестуванню сутностей за допомогою ExUnit, розробник має змогу виявити та виправити помилки на ранніх етапах розробки, що сприяє підвищенню якості проекту та зменшенню ризику виникнення дефектів у майбутньому.

```
defmodule UserTest do
 use MessengerWeb.ConnCase, async: true

 setup_all do
  raw_user = %{email: "test@gmail.com", nickname: "test", password: "mypassword"}

  %{
   raw_user: raw_user
  }
 end

 test "login user", %{raw_user: raw_user} do
  {:ok, _created_user} = Messenger.User.create_user(raw_user)

  {:ok, _logged_user} =
   Messenger.User.login(raw_user)
 end
```

```
  test "user login with wrong password", %{raw_user: raw_user} do
    {:ok, _created_user} = Messenger.User.create_user(raw_user)

    {:error, reason} =
      Messenger.User.login(%{email: raw_user.email, password: "wrong password"})

    assert reason == "Wrong password"
  end

  test "user login with non-existing email", %{raw_user: raw_user} do
    {:ok, _created_user} = Messenger.User.create_user(raw_user)

    {:error, reason} =
      Messenger.User.login(%{email: "wrong_email@gmail.com", password:
raw_user.password})

    assert reason == "There is no such user"
  end
end
```

Цей код є частиною тестового модуля `UserTest`, модуль використовує `MessengerWeb.ConnCase`, який є тестовим допоміжним модулем, що надає додаткові функції для тестування веб-запитів у Phoenix. Вказівка `async: true` дозволяє Elixir виконувати тести асинхронно, що може прискорити процес тестування.

У функції `setup_all` визначається початковий стан, який буде використовуватися у всіх тестах цього модуля. Вона створює словник з даними користувача `raw_user` і повертає цей словник у мапі, щоб він був доступний у кожному тесті.

Перший тест "login user" перевіряє, чи може користувач успішно увійти в систему після створення облікового запису з використанням валідних даних.

Спочатку створюється новий користувач за допомогою функції `Messenger.User.create_user/1`, а потім виконується спроба входу через `Messenger.User.login/1`.

Другий тест "user login with wrong password" перевіряє сценарій, коли користувач намагається увійти з неправильним паролем. Після створення користувача тест спробує виконати вхід з неправильним паролем і перевірить, що результатом є помилка з очікуваним повідомленням "Wrong password".

Третій тест "user login with non-existing email" перевіряє сценарій, коли користувач намагається увійти з електронною адресою, яка не існує в системі. Після створення користувача тест спробує виконати вхід з невідомою електронною адресою і перевірить, що результатом є помилка з повідомленням "There is no such user".

Кожен тест використовує патерн зіставлення зі зразком (`pattern matching`) для перевірки результату виклику функцій, а також `assert` для переконання, що помилки відповідають очікуваним. Ці тести допомагають забезпечити, що система аутентифікації працює належним чином і коректно обробляє як успішні, так і невдали спроби входу.

3.8. Впровадження

Для розгортання проекту було обрано платформу Heroku, яка є облачним сервісом, що підтримує різноманітні мови програмування, включаючи Elixir. Використання Heroku дозволяє спростити процес розгортання за допомогою використання buildpacks, які автоматизують налаштування середовища та залежностей. У цьому випадку, для розгортання Elixir-додатку було використано buildpack `hashnuke/elixir`, який спеціально призначений для роботи з Elixir-додатками на Heroku.

Проект також використовує практики DevOps для прискорення процесу впровадження інновацій, оптимізуючи та автоматизуючи процеси розробки програмного забезпечення та управління інфраструктурою. Це досягається за допомогою безперервної інтеграції та безперервного розгортання (CI/CD), що зменшує час, необхідний для випуску нових оновлень програмного забезпечення.

Heroku надає власні інструменти для реалізації CI/CD, які можуть бути інтегровані з репозиторіями коду, такими як GitHub. Це дозволяє автоматизувати процеси збірки, тестування та розгортання безпосередньо з репозиторію, забезпечуючи плавне та ефективне розгортання додатків.

Використання Heroku як платформи для розгортання дозволяє розробникам зосередитися на кодуванні та інноваціях, замість витрачання часу на управління серверами та інфраструктурою. Завдяки простоті налаштування та великій кількості додаткових сервісів, Heroku є популярним вибором серед розробників для швидкого розгортання та масштабування веб-додатків.

Список кроків, які іллюструють архітектуру рішення:

1) Розробник робить зміни коду з локального сховища до репозиторію на GitHub;

2) GitHub Actions ініціює процес збірки;

3) OIDC використовує токени для автентифікації в Heroku та доступу до сервісів;

4) GitHub Actions відправляє артефакти розгортання до Heroku;

5) GitHub Actions викликає Heroku для запуску процесу розгортання;

6) Heroku розпочинає розгортання на динамічних контейнерах;

7) Heroku отримує артефакти та розгортає їх на динамічних контейнерах, забезпечуючи запуск оновленої версії застосунку.

Також для розгортання було необхідно створити базу даних PostgreSQL. Heroku пропонує різні плани для баз даних, включаючи безкоштовні та платні опції. У контексті дипломного проекту, було вирішено використовувати платний план бази даних, що коштує приблизно 5 доларів на місяць, щоб забезпечити необхідні ресурси та функціональність для стабільної роботи застосунку.

Цей план надав додаткові переваги, такі як більший обсяг ресурсів, резервне копіювання та кращу підтримку, які є важливими для довгострокової експлуатації та масштабування застосунку.

ВИСНОВКИ ДО РОЗДІЛУ 3

У висновку до третього розділу, присвяченого розробці проекту кваліфікаційної роботи, можна підкреслити, що процес створення вебзастосунку для листування в реальному часі був успішно завершений. Використання Elixir Phoenix та Phoenix LiveView дозволило реалізувати інтерактивний інтерфейс користувача, який забезпечує плавну та зручну взаємодію. Система авторизації, побудована на основі Guardian, забезпечила надійний захист даних користувачів та безпечне управління сесіями.

Протягом розробки було приділено значну увагу тестуванню сутностей за допомогою ExUnit, що сприяло виявленню та усуненню помилок на ранніх етапах та забезпечило високу якість програмного продукту. Phoenix Channels були використані для створення надійних каналів комунікації, що дозволило користувачам обмінюватися повідомленнями в режимі реального часу.

Загалом, розробка проекту для кваліфікаційної роботи продемонструвала ефективність вибраних технологій та підходів у створенні сучасних веб-додатків, а також підтвердила важливість ретельного планування, тестування та впровадження кращих практик безпеки. Результатом став функціональний та надійний вебзастосунок, який відповідає сучасним вимогам до швидкості, безпеки та користувацького досвіду.

ВИСНОВКИ

У висновку до кваліфікаційної роботи можна з упевненістю стверджувати, що розробка вебзастосунку для листування в реальному часі була виконана з дотриманням високих стандартів якості та інноваційності. Використання мови програмування Elixir у поєднанні з Phoenix Framework та Phoenix LiveView дозволило створити продукт, який не тільки відповідає сучасним технічним вимогам, але й відзначається високою продуктивністю, масштабованістю та здатністю до швидкої адаптації під змінні умови використання.

Завдяки впровадженню Phoenix Channels, було досягнуто ефективної реалізації комунікаційних каналів, що є невід'ємною частиною будь-якого додатку, що працює в режимі реального часу. Це забезпечило користувачам миттєвий обмін повідомленнями та високу швидкість відгуку системи на дії користувача, що є важливим для забезпечення позитивного користувацького досвіду.

Система авторизації, реалізована з використанням бібліотеки Guardian, забезпечила надійний захист даних користувачів, що є особливо важливим у контексті збереження конфіденційності та безпеки особистої інформації. Тестування сутностей за допомогою ExUnit виявилося надзвичайно корисним у виявленні та усуненні потенційних проблем, що сприяло створенню стабільного та надійного програмного продукту.

Використання принципів реактивного програмування дозволило розробити систему, яка може швидко реагувати на зміни стану та події, що відбуваються в системі, забезпечуючи користувачам негайну відповідь та високу інтерактивність інтерфейсу. Асинхронна обробка даних, у свою чергу, забезпечила можливість ефективної роботи з великими обсягами інформації та високим навантаженням без втрати продуктивності, що є критично важливим для додатків, які функціонують в режимі реального часу.

Проект також підкреслив важливість ретельного планування архітектури додатку та впровадження сучасних практик розробки, включаючи принципи реактивного програмування та асинхронної обробки даних. Це дозволило ефективно

вирішувати завдання, пов'язані з одночасною обробкою великої кількості запитів, забезпечуючи високу швидкість відгуку та зручність користувачів.

Завдяки глибокому аналізу вимог до проекту та вибору оптимальних технологій, розроблений месенджер не тільки відповідає поточним потребам користувачів, але й має потенціал для адаптації під майбутні виклики та технологічні зміни. Це створює можливості для його використання в різних сферах бізнесу, від стартапів до великих корпорацій, які прагнуть покращити внутрішню та зовнішню комунікацію, забезпечуючи ефективність та конфіденційність обміну інформацією.

Проект відзначається високою масштабованістю та гнучкістю, що дозволяє йому легко інтегруватися з існуючими корпоративними системами та адаптуватися до специфічних потреб бізнесу. Розроблений месенджер підтримує широкий спектр функціональності, включаючи групові чати, передачу файлів, відеодзвінки та інші необхідні інструменти для сучасного робочого процесу. Це робить його ідеальним рішенням для організацій, які шукають ефективні та безпечні способи комунікації.

Окрім того, в процесі розробки було приділено значну увагу оптимізації продуктивності та безпеки. Використання сучасних протоколів шифрування та аутентифікації забезпечує захист від несанкціонованого доступу та зовнішніх загроз, що є критично важливим для збереження конфіденційності корпоративної інформації. Також було реалізовано ряд оптимізацій, які знижують навантаження на сервери та покращують швидкість обробки даних, що важливо для великих організацій з великою кількістю користувачів.

У підсумку, кваліфікаційна робота відображає успішне застосування теоретичних знань у практичній розробці вебзастосунків. Результати роботи можуть бути використані як основа для подальших досліджень у сфері веб-розробки та як відправна точка для розширення функціональності та можливостей веб-застосунків реального часу. Розроблений месенджер має потенціал стати важливим інструментом для підприємств, що прагнуть покращити внутрішню та зовнішню комунікацію, та може бути інтегрований у різноманітні бізнес-процеси, забезпечуючи ефективність та конфіденційність обміну інформацією.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Книга: Thomas, C. Real-Time Phoenix: Build Highly Scalable Systems with Channels. Pragmatic Bookshelf, 2020. 360 с.

2. Книга: Valim, J. Crafting GraphQL APIs in Elixir with Absinthe. Pragmatic Bookshelf, 2019. 250 с.

3. Книга: Tate, B.A., Valim, J., и Junior, M.S. Programming Phoenix ≥ 1.4: Productive |> Reliable |> Fast. Pragmatic Bookshelf, 2021. 300 с.

4. Книга: Wang, V., Salim, F., Moskovits, P. The Definitive Guide to HTML5 WebSocket. Apress, 2022. 208 с.

5. Книга: Lombardi, A. WebSocket: Lightweight Client-Server Communications. O'Reilly Media, 2020. 144 с.

6. Книга: Grigorik, I. High Performance Browser Networking. O'Reilly Media, 2020. 400 с.

7. Ресурс Інтернету: MDN Web Docs. WebSocket API. URL: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API (дата звернення 10.12.2023).

8. Ресурс Інтернету: HTML5 Rocks. Introducing WebSockets: Bringing Sockets to the Web. URL: https://www.html5rocks.com/en/tutorials/websockets/basics/ (дата звернення 11.12.2023).

9. Ресурс Інтернету: Elixir School. URL: https://elixirschool.com/ (дата звернення 01.04.2023)

10. Ресурс Інтернету: Phoenix Framework Documentation. URL: https://hexdocs.pm/phoenix/overview.html (дата звернення 12.12.2023)

11. Ресурс Інтернету: The Road to 2 Million Websocket Connections in Phoenix. URL: https://phoenixframework.org/blog/the-road-to-2-million-websocket-connections (дата звернення 08.12.2023)

12. Ресурс Інтернету: Introduction to Elixir. URL: https://elixir-lang.org/getting-started/introduction.html (дата звернення 07.12.2023)

13. Ресурс Інтернету: Guardian GitHub Repository. URL: https://github.com/ueberauth/guardian (дата звернення 05.12.2023)

14. Ресурс Інтернету: Phoenix LiveView GitHub Repository. URL: https://github.com/phoenixframework/phoenix_live_view (дата звернення 04.12.2023)

15. Ресурс Інтернету: Real-Time Web Applications with Phoenix and Elixir. URL: https://www.pluralsight.com/courses/real-time-web-applications-phoenix-elixir (дата звернення 13.12.2023)