МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра комп'ютерних наук

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РОЗРОБКА МЕСЕНДЖЕРУ З ВИКОРИСТАННЯМ JAVASCRIPT ФРЕЙМВОРКІВ»**

Виконав: студент 5 курсу, групи 6.1229-з
спеціальності 122 Комп'ютерні науки
(шифр і назва спеціальності)
освітньої програми Комп'ютерні науки
(назва освітньої програми)
М.С. Кирилюк
(ініціали та прізвище)

Керівник: доцент кафедри комп'ютерних наук, доцент, д.е.н. Полуектова Н. Р.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент: завідувач кафедри фундаментальної та прикладної математики, професор, д.т.н., Гребенюк С.М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя
2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра комп'ютерних наук

Рівень вищої освіти бакалавр

Спеціальність 122 Комп'ютерні науки
(шифр і назва)

Освітня програма Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних наук,
д.т.н., професор

_____ Шило Г. М.
(підпис)

" ____ " _____ 2024 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Кирилюку Максиму Сергійовичу
(прізвище, ім'я та по-батькові)

1. Тема роботи Розробка месенджеру з використанням Javascript фреймворків

керівник роботи Полуектова Наталія Робертівна, д.т.н., професор
(прізвище, ім'я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 21 » 12 2023 року № 2181-с

2. Строк подання студентом роботи 15.05.2024 р

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

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі.
2. Основні теоретичні відомості.
3. Розробка месенджеру з використанням javascript фреймворків

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
презентація

6. Консультанти розділів роботи

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

7. Дата видачі завдання 25.12.2023

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	10.01.2024	
2.	Збір вихідних даних.	10.02.2024	
3.	Обробка методичних та теоретичних	01.03.2024	
	джерел.		
4.	Розробка першого та другого розділу.	10.04.2024	
5.	Розробка третього розділу.	01.05.2024	
6.	Оформлення та нормоконтроль	14.05.2024	
	кваліфікаційної роботи бакалавра.		
7.	Захист кваліфікаційної роботи.	30.05.2024	

Студент _____ М.С. Кирилюк
(підпис) (ініціали та прізвище)

Керівник роботи _____ Н.Р. Полуектова
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ О.Г. Спиця
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Розробка месенджеру з використанням Javascript фреймворків»: 37 с., 7 рис., 10 джерел, 1 додатків.

API, AWS, LAMBDA, REACT, SERVERLESS, S3, WEB-MESSENGER, WEBSOCKETS.

Об’єкт дослідження – технології розробки месенджерів

Предмет дослідження – розробка web-месенджеру за допомогою JavaScript фреймворків з використанням хмарних сервісів.

Мета роботи: створення web-додатку простого месенджера з використанням JavaScript фреймворків та хмарних технологій.

Методи дослідження – аналіз та порівняння, синтез та аналогія, експеримент.

Було створено простий web-месенджер який може використовуватись у організаціях, сервісах підтримки для взаємодії між співробітниками, або у спілкуванні з користувачами.

SUMMARY

Bachelor's Qualification Theses «Development of a messenger using Javascript frameworks»: 37 p., 7 figures, 10 sources, 1 addition.

API, AWS, LAMBDA, REACT, SERVERLESS, S3, WEB-MESSENGER, WEBSOCKETS.

Object of research – messenger development technologies

The subject of research is the development of a web messenger using JavaScript frameworks with the use of cloud services.

Purpose: to create a web application of a simple messenger using JavaScript frameworks and cloud technologies.

Research methods: analysis and comparison, synthesis and analogy, experiment.

A simple web messenger was created that can be used in organizations, support services for interaction between employees, or in communication with users.

ЗМІСТ

ВСТУП

У сучасному світі швидкість та доступність інформації визначають ефективність комунікації. Це особливо актуально у сфері Інтернет-комунікацій, де web-месенджери займають ключове місце у взаємодії користувачів. Розвиток технологій, зокрема Javascript фреймворків, відкриває нові можливості для створення web-додатків, забезпечуючи швидкий та зручний доступ до них через браузери.

Ця кваліфікаційна робота спрямована на дослідження процесу розробки web-месенджерів на базі Javascript фреймворків. Ці додатки стають все більш популярними через їхню зручність та доступність. Використання сучасних фреймворків у поєднанні з можливостями мови програмування Javascript дозволяє розробникам створювати потужні, функціональні та естетично привабливі web-месенджери.

Мета цієї роботи полягає у вивченні основних принципів та підходів до розробки web-месенджерів на базі відомих Javascript фреймворків, таких як React, Angular або Vue.js. Під час дослідження будуть проаналізовані переваги та недоліки кожного фреймворку, їхні можливості та обмеження у контексті розробки месенджерів. Крім того, у роботі будуть розглянуті сучасні тенденції у web-розробці та їх вплив на процес створення web-месенджерів.

Аналіз і порівняння різних підходів до розробки web-месенджерів на базі Javascript фреймворків дозволить визначити оптимальні стратегії та інструменти для реалізації подібних проектів. Отримані результати можуть бути корисними для розробників, які прагнуть покращити якість та ефективність своїх web-додатків, зокрема web-месенджерів, заснованих на сучасних технологіях.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТІ ІНСТРУМЕНТІВ РОЗРОБКИ

1.1 Призначення і основні функції web-месенджерів

Web-месенджери є важливим елементом сучасного спілкування в онлайн-середовищі. Їхнє призначення полягає в наданні користувачам зручного та ефективного засобу обміну повідомленнями через Інтернет. У цьому розділі розглянемо основні функції, які вони виконують, і їхнє значення для користувачів.

До основних функцій web-месенджерів відносяться наступні [8].[1]

- **обмін текстовими повідомленнями**. Користувачі можуть відправляти текстові повідомлення один одному через месенджери, що дозволяє швидко та зручно обмінюватися інформацією;
- **відео та аудіо дзвінки**. Більшість web-месенджерів підтримують можливість здійснювати відео- та аудіо дзвінки, що дає можливість спілкуватися з друзями, родиною або колегами у реальному часі;
- **відправлення медійних файлів**. Користувачі можуть надсилати мультимедійні файли, такі як фотографії, відео, аудіо, документи тощо, з одного пристрою на інший через месенджер;
- **групові чати та конференції**. Користувачі можуть надсилати мультимедійні файли, такі як фотографії, відео, аудіо, документи тощо, з одного пристрою на інший через месенджер;
- **стікери та емодзі**. В багатьох месенджерах є вбудована колекція стікерів та емодзі, які можна використовувати для вираження почуттів та емоцій у спілкуванні;
- **шифрування для забезпечення конфіденційності**. Безпека даних є важливою функцією web-месенджерів. Багато з них використовують

[1] Стаття-огляд на веб-месенджери та їх призначення. Webwise.ie. URL: https://www.webwise.ie/parents/explained-what-is-messenger/ (дата звернення: 13.04.2024).

сучасні методи шифрування для захисту приватності користувачів та їхніх даних.

Ці основні функції дозволяють користувачам зручно та ефективно спілкуватися між собою через web-месенджери, роблячи їх невід'ємною частиною їхнього цифрового життя.

1.2 Клієнт-серверна архітектура web-додатків

Технологія клієнт-сервер, яка давно використовується для роботи з базами даних у мережах, має широку популярність, особливо у великих компаніях. З розвитком Інтернету ця технологія стала ще більш привабливою для розробників програмного забезпечення, оскільки в сучасному світі накопичено велику кількість інформації з різних питань, яка зазвичай зберігається в базах даних.

Мережева архітектура визначає ключові компоненти мережі, описує її загальну логічну структуру, а також технічне і програмне забезпечення. Вона також охоплює методи кодування, принципи роботи та користувацькі інтерфейси.

Архітектура клієнт – сервер (client-server architecture) – це концепція мережі, де основні ресурси зосереджені на серверах які обслуговують клієнтів (Рисунок 1.1).[**Помилка! Джерело посилання не знайдено.**][2] Ця архітектура включає два типи компонентів: сервери та клієнти.

Сервер надає сервіси за запитами клієнтів, обробляє їхні завдання і надсилає результати. Серверна функція реалізується за допомогою прикладних програм, які виконують різні процеси.

[2] Архітектура клієнт-сервер. ВСП «Павлоградський фаховий коледж НТУ «Дніпровська політехніка». URL: http://inter.ptngu.com/kompyuterni-merezhi/arhitektura-kliyent-server (дата звернення: 25.04.2024).

Клієнт, який може бути програмою або користувачем, ініціює запити на сервіси. Клієнти – це робочі станції, що використовують ресурси сервера і мають зручні інтерфейси користувача.

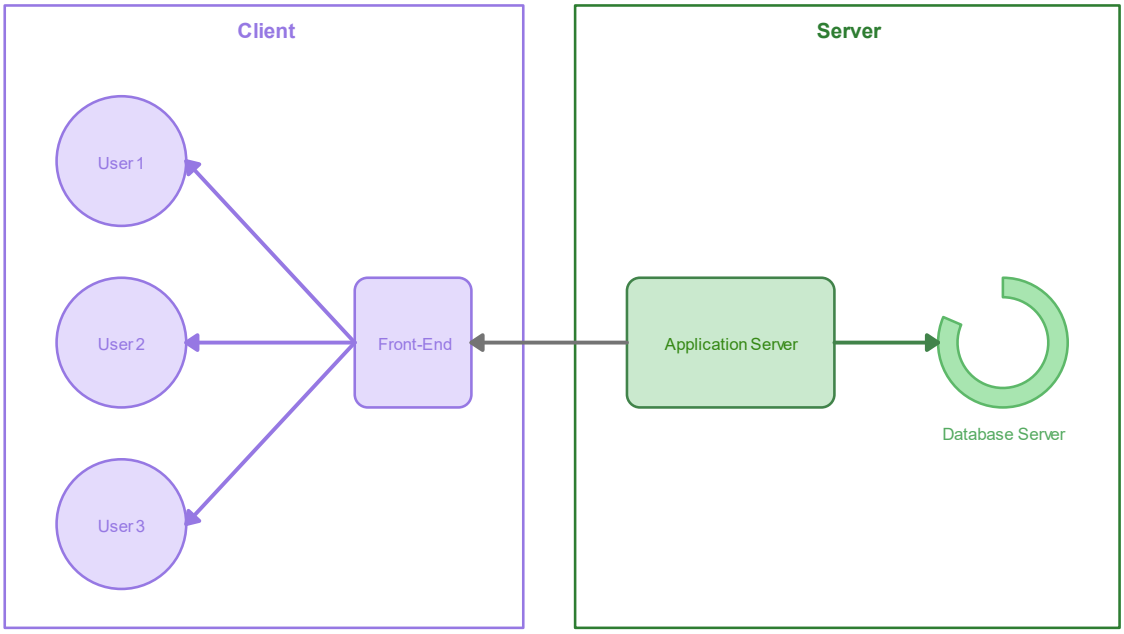

Рисунок 1.1 – Архітектура «Клієнт-Сервер»

У мережах з виділеним файловим сервером сервером є ПК з серверною операційною системою, такою як NetWare, Windows NT, UNIX або Linux. Серверні мережі забезпечують кращу продуктивність та надійність.

У клієнт-серверній архітектурі виділяються чотири групи об’єктів: клієнти, сервери, дані та мережеві служби. Клієнти розміщені на робочих станціях, дані зберігаються на серверах, а мережеві служби забезпечують спільний доступ до ресурсів і керують обробкою даних.

Переваги клієнт-серверних мереж:

– можливість організації великих мереж;
– централізоване управління користувачами та безпекою;
– ефективний доступ до ресурсів;
– один пароль для доступу до всіх ресурсів.

Недоліки:

– збої на сервері можуть зробити мережу недоступною;

- Необхідність кваліфікованого персоналу для адміністрування;
- вища вартість мереж та обладнання.

1.3 Порівняльний аналіз технологій клієнт-серверної розробки

У цьому розділі проведемо порівняльний аналіз різних технологій, які використовуються для розробки клієнт-серверних додатків, з урахуванням їхніх переваг, недоліків та сфер застосування. [7][3]

Технологія Java EE

- **переваги**:
  - масштабованість: Java EE дозволяє легко масштабувати додатки за допомогою технологій, таких як Enterprise JavaBeans (EJB) та Java Messaging Service (JMS);
  - безпека: Фреймворк має вбудовані механізми безпеки, такі як Java Authentication and Authorization Service (JAAS) та Java EE Security API, що дозволяють забезпечити захист даних;
- **недоліки**:
  - складність: Java EE може бути складним для вивчення та розгортання через велику кількість компонентів та конфігурацій;
  - великі вимоги до ресурсів: Додатки, розроблені з використанням Java EE, можуть вимагати більших обсягів ресурсів, таких як пам'ять та обчислювальна потужність.

Технологія ASP.NET

- **переваги**:
  - інтеграція з іншими продуктами Microsoft: ASP.NET добре інтегрується з іншими продуктами Microsoft, такими як Microsoft SQL Server, що полегшує розробку та підтримку;

[3] Клієнт-серверна модель – Вікіпедія. Wikipedia, the free encyclopedia. URL: https://en.wikipedia.org/wiki/Client–server_model (дата звернення: 20.03.2024).

  - висока продуктивність: Дотримання великих проектів та високий рівень продуктивності є одними з ключових переваг ASP.NET.
- **недоліки**:
  - залежність від платформи: Технологія ASP.NET відома своєю залежністю від платформи Windows, що може обмежувати переносимість та розгортання на інших операційних системах;
  - вартість: Використання платформи ASP.NET може бути вартим, оскільки вона часто вимагає плати за ліцензії та підтримку

Технологія Node.js

- **переваги**:
  - висока продуктивність: Node.js забезпечує високу продуктивність завдяки асинхронному та подієвому програмуванню;
  - переносимість: Додатки, розроблені з використанням Node.js, можуть бути легко перенесені між різними платформами;
- **недоліки**:
  - загроза блокування потоків: Асинхронне програмування може стати складним для розуміння та керування, особливо в складних програмах;
  - Відсутність вбудованих механізмів безпеки: Node.js не має вбудованих механізмів безпеки, тому розробнику потрібно бути обережним при розробці застосунків.

Технологія Flask (Python)

- **переваги**:
  - простота використання: Flask є легким та простим у використанні фреймворком, що дозволяє швидко розробляти web-додатки на Python;
  - гнучкість: Flask надає велику гнучкість у виборі компонентів та бібліотек для розробки, що дозволяє розробникам вибирати оптимальні рішення для своїх проектів;

- **недоліки**:
  - відсутність вбудованих інструментів безпеки: Flask не має вбудованих механізмів безпеки, тому розробник повинен бути обережним та вручну додавати необхідні заходи безпеки;
  - відсутність великої спільноти: У порівнянні з іншими фреймворками, спільнота Flask менша, що може призвести до меншої кількості доступних ресурсів та підтримки.

Вибір технології для розробки клієнт-серверних додатків залежить від ряду факторів, включаючи потреби проекту, ресурси, вміння розробника та вимоги до продуктивності та безпеки. Кожна з технологій має свої переваги та недоліки, тому важливо ретельно проаналізувати їх перед прийняттям рішення.

2 РЕАЛІЗАЦІЯ WEB-МЕСЕНДЖЕРА

2.1 Постановка задачі, опис вимог, основні функції

Створити простий web-месенджер для обміну текстовими повідомленнями зі спрощеною реєстрацією, який базується на хмарних сервісах AWS. Основною функцією системи є можливість обміну текстовими повідомленнями між зареєстрованими користувачами.

Опис вимог

1. **Реєстрація користувачів**: Користувачі можуть створювати облікові записи з мінімальною інформацією.
2. **Аутентифікація**: Користувачі можуть використовувати свої облікові записи для входу до системи.
3. **Обмін повідомленнями**: Зареєстровані користувачі можуть надсилати та отримувати текстові повідомлення в реальному часі.
4. **Список контактів**: Користувачі можуть додавати і видаляти інших користувачів у список контактів.

Основні функції

1. **Реєстрація користувачів**:
   - Форма реєстрації з полями для введення ім'я користувача
   - Збереження облікових даних користувача у базі даних.
2. **Аутентифікація**:
   - Форма входу для користувачів з полями для введення логіну користувача.
   - Перевірка введених даних з базою даних для авторизації користувача.
3. **Обмін повідомленнями**:
   - Інтерфейс для надсилання та отримання повідомлень.
   - Зберігання повідомлень у базі даних.

- Відображення повідомлень в реальному часі за допомогою web-сокетів.

4. **Список контактів**:
   - Можливість переглядати список контактів.
   - Можливість вибирати зі списку співрозмовника.

2.2 Обґрунтування вибору бази даних

При створенні простого web-месенджера для обміну текстовими повідомленнями зі спрощеною реєстрацією, вибір бази даних відіграє критичну роль у забезпеченні продуктивності, масштабованості та доступності системи. DynamoDB від хмарних сервісів Amazon виступає найбільш оптимальним варіантом для цієї цілі, з урахуванням таких факторів:

1. Хмарна база даних:

DynamoDB є повністю керованою базою даних у хмарі, що означає, що ви не потребуєте власної інфраструктури або адміністративних зусиль для розгортання та керування базою даних. Це дозволяє зосередитися на розробці додатку, не витрачаючи час на управління інфраструктурою.

2. Висока доступність та надійність:

DynamoDB гарантує високу доступність та надійність за рахунок реплікації даних на кількох центрах даних AWS. Це забезпечує практично беззавдане відновлення при випадку відмови та мінімізує можливість втрати даних.

3. Масштабованість:

DynamoDB легко масштабується, як горизонтально, так і вертикально. Ви можете змінювати потужність та місткість бази даних в залежності від

потреб вашого додатку, що дозволяє підтримувати його продуктивність під навантаженням.

4. Висока продуктивність:

DynamoDB пропонує низьку затримку та високу швидкодію завдяки розподіленому архітектурному рішенню та оптимізаціям внутрішнього механізму обробки даних. Це забезпечує швидку відповідь на запити користувачів та збереження рівноваги в системі навіть при великому навантаженні.

5. Гнучкість схеми даних:

DynamoDB підтримує гнучкі схеми даних, що дозволяє легко змінювати структуру даних без необхідності зупинення роботи системи або перевірки цілісності даних. Це дозволяє швидко адаптувати базу даних до змін у вимогах додатку.

Узагальнюючи, використання DynamoDB від Amazon Web Services для бази даних у вашому web-месенджері забезпечить надійну, масштабовану та високопродуктивну архітектуру, яка відповідає потребам сучасних хмарних додатків.

2.3 Структура БД

1. Таблиця «Clients»

Ця таблиця призначена для зберігання інформації про клієнтів (Рисунок *2* 2.1), які підключені до web-месенджера.

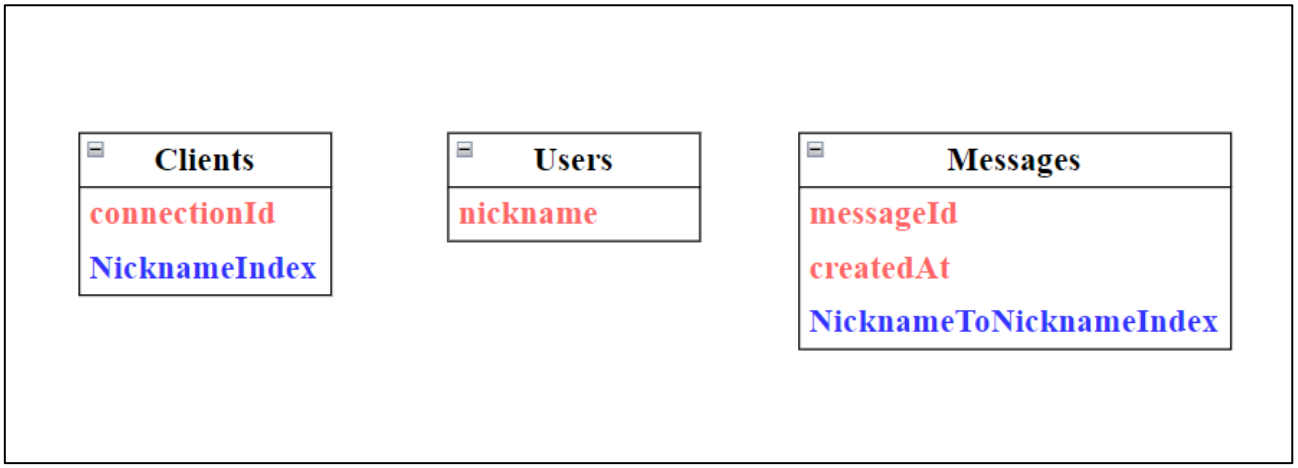

Рисунок 2.1 – Структура БД

Основні характеристики таблиці:

- **Primary Key**:
  - **connectionId (HASH)**: Унікальний ідентифікатор з'єднання клієнта, який генерується при підключенні до месенджера. Цей ідентифікатор використовується для ідентифікації конкретного клієнта.
- **Global Secondary Index**:
  - **NicknameIndex**: Індекс, який дозволяє шукати клієнтів за їхнім ім'ям (nickname).

2. Таблиця «Users»

Ця таблиця містить інформацію про користувачів месенджера. Основні характеристики таблиці:

- **Primary Key**:

- **nickname (HASH)**: Унікальне ім'я користувача, яке використовується для ідентифікації кожного користувача в системі. Це поле є обов'язковим для реєстрації користувача.

3. Таблиця «Messages»

Ця таблиця використовується для зберігання текстових повідомлень, які обмінюються користувачами месенджера. Основні характеристики таблиці:

- **Primary Key**:
  - **messageId (HASH)**: Унікальний ідентифікатор кожного повідомлення.
  - **createdAt (RANGE)**: Дата та час створення повідомлення, які використовуються для сортування повідомлень за часом.
- **Global Secondary Index**:
  - **NicknameToNicknameIndex**: Індекс, який дозволяє шукати повідомлення за інформацією про відправника та отримувача повідомлення, а також за датою створення.

Кожна з цих таблиць має оптимізовану структуру для забезпечення ефективного зберігання та отримання даних. Глобальні вторинні індекси дозволяють шукати дані за різними критеріями, що підвищує продуктивність та зручність роботи з базою даних.

2.4 Обґрунтування вибору фреймворків для реалізації інтерфейсу

Вибір React та Typescript для реалізації інтерфейсу цього web-месенджера базується на декількох важливих факторах, які впливають на якість, продуктивність та масштабованість проекту.

React є одним з найпопулярніших фреймворків для створення користувацьких інтерфейсів. Він надає широкий набір готових компонентів та дозволяє ефективно використовувати компонентний підхід у розробці. Typescript, у свою чергу, додає статичну типізацію до вашого проекту, що

сприяє виявленню та усуненню помилок на етапі розробки і полегшує підтримку великих проектів.

Обидва фреймворки мають велику активну спільноту розробників, що забезпечує швидку підтримку та поширення навичок. Використання React та Typescript сприяє масштабуванню проекту та забезпечує швидкий розвиток інтерфейсу з високою продуктивністю та зручністю в підтримці.

Також варто відзначити, що React та Typescript мають велику сумісність з іншими бібліотеками та інструментами, що полегшує їх інтеграцію та розширення. Це дозволяє вам легко і швидко використовувати нові функції та інструменти для вашого проекту.

Загалом, використання React та Typescript для реалізації інтерфейсу вашого web-месенджера є обґрунтованим рішенням, яке забезпечить продуктивну розробку, швидку та масштабовану систему з надійним та ефективним кодом

2.5 Обґрунтування вибору фреймворку для серверної розробки

Для реалізації серверної частини був обранний Serverless framework. Serverless framework – це інструмент для розробки, розгортання та керування серверними програмами та функціями в хмарних сервісах, таких як AWS, Azure, Google Cloud та інші. Він дозволяє розробникам будувати і виконувати додатки без необхідності управління інфраструктурою або серверами.

Ось деякі ключові аспекти Serverless framework:

- Serverless framework надає простий та зручний інтерфейс для розробки, розгортання та керування додатками. Він використовує конфігураційні файли для опису структури вашого додатку та налаштування ресурсів, що дозволяє легко керувати всіма аспектами вашого додатку;
- Serverless framework дозволяє розгортати додатки та функції швидко та ефективно. Він автоматизує багато процесів, пов'язаних з розгортанням,

таких як створення та налаштування ресурсів, що дозволяє швидко реагувати на зміни в додатку та виконувати нові версії програми.

– Одним з головних переваг Serverless framework є його масштабованість. Можна легко масштабувати свої додатки вгору або вниз, змінюючи кількість ресурсів, що використовуються, в залежності від потреб вашого додатку. Це дозволяє ефективно використовувати ресурси та забезпечує високу доступність та продуктивність вашого додатку.
– Serverless framework підтримує інтеграцію з різними хмарними сервісами, такими як AWS Lambda, Amazon API Gateway, DynamoDB, S3 та інші. Це дозволяє легко використовувати різноманітні сервіси та інструменти для розробки додатку.

Узагальнюючи, Serverless framework є потужним інструментом для розробки серверних додатків та функцій у хмарних сервісах. Він надає простий та зручний спосіб розробки та розгортання додатків, що дозволяє ефективно використовувати ресурси та швидко реагувати на зміни в додатку.

2.6 Особливості реалізації програмних модулів

При побудові додатку ми використали просту файлову структуру проекту. Розділили клієнтську та серверну частину на окремі директорії. На клієнтській системі ми ініціювали шаблон для web-додатку на фреймворці React, поєднаному зі строгою типізацією яку нам дає бібліотека Typescript, який потім переробили у наш додаток месенджеру.

Розглянемо більш детально клієнтську частину. В ній ми будуємо UI нашого додатку. Структура проекту виглядає наступним чином (рис. 2.2):

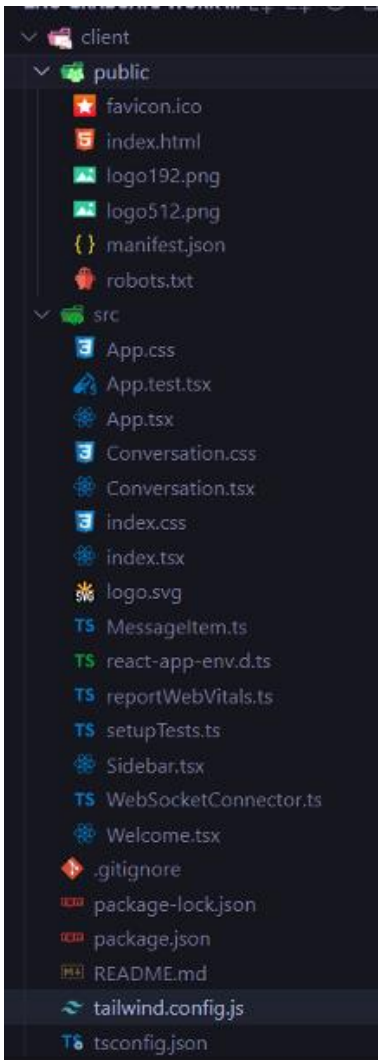

Рисунок 2.2 – Структура UI частини

На верхньому рівні лежать файли package.json та package-lock.json відповідають за збереження залежностей для розгортання проекту та встановлення необхідних пакетів. Файл README.md тримає в собі інструкції щодо розгортання, .gitignore вказує на те які файли ігнорувати при використанні системи контролю версій, такі як нод-пакети, які зазвичай займають великий обсяг.

У самому додатку ми не використовуємо багато сторінок, використовується одна головна сторінка де ми маємо два компоненти:

– Сторінка авторизації/регистрації. Наш додаток представляє собою MVP (Minimal Value Product), що ставить за мету лише показати можливий

вид продукту, тому регистрація у нас спрощена і являє собою 2 в 1 і регистрацію і авторизацію;

– Сторінка чату, де розташовується основний функціонал додатку. Тут ми можемо обрати співбесідника, відправити повідомлення йому та водночас подивитись повідомлення від нього.

За допомогою бібліотеки React.js ми можемо дуже легко в залежності від умов, показувати той чи інший компонент, та зручно декомпозувати та удосконалювати наявні компоненти за допомогою модульного підходу. Лістинг основних компонентів буде представлений у додатках.

Стосовно серверної частини У цій дипломній роботі було використано Serverless Framework. Цей підхід дозволив створити динамічний та масштабований додаток з мінімальними витратами часу та ресурсів. Фреймворки Serverless можна використовувати для створення web-месенджерів з різними функціями. Наприклад, можна використовувати AWS Lambda для створення безсерверної функції, яка обробляє вхідні повідомлення, а потім зберігає їх у базі даних Amazon DynamoDB.

Також можна використовувати API Gateway для створення API, який дозволяє клієнтам надсилати та отримувати повідомлення. S3 можна використовувати для зберігання зображень користувачів та інших даних.

Файлова структура виглядає наступним чином (рис.2.3):

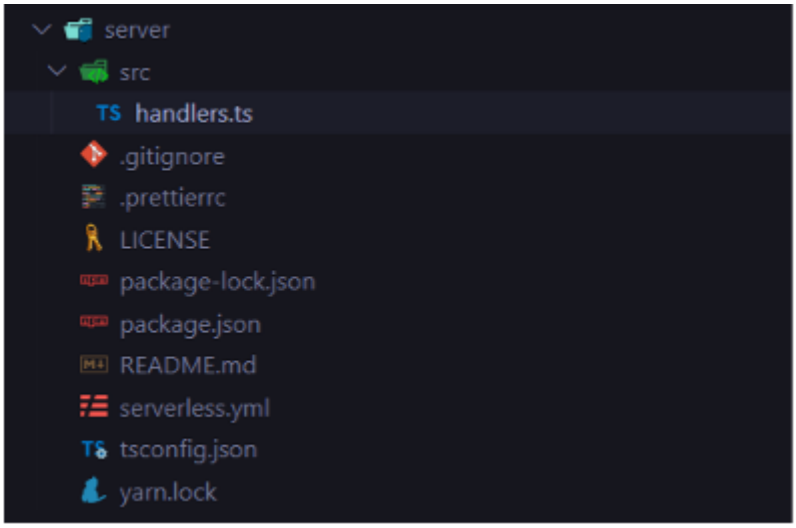

Рисунок 2.3 – Файлова структура бекенду

Файл serverless.yml (див. Додаток А) відповідає за конфігурацію розгортання нашого додатку на серверах Amazon. Файл handlers.ts відповідає за конфігурацію функцій які будуть визиватися на ті чи інші події. Лістинги обох файлів можна знайти у додатках. Розглянемо більш детально ці два файли.

Файл serverless.yml (див. Додаток А) є головним файлом конфігурації для нашого серверного застосунку з використанням Serverless Framework. Він визначає базові параметри та використовує значення за замовчуванням.

- **service:** Назва вашого сервісу (serverless-chat у цьому випадку)
- **frameworkVersion:** Версія Serverless Framework, яку ви використовуєте (3 у цьому випадку)
- **provider:** Налаштування провайдера хмарної платформи (AWS у цьому випадку)
  - **runtime:** Версія Node.js, яку використовуватиме ваш серверless-функції (nodejs20.x у цьому випадку)
  - **lambdaHashingVersion:** Версія хешування для ваших Lambda-функцій (20201221 у цьому випадку)
- **environment:** Змінні середовища, доступні нашим Lambda-функціям. У цьому випадку, WSSAPIGATEWAYENDPOINT містить URL для нашого API Gateway.
- **iam:** Роль IAM, призначена нашим Lambda-функціям. Ця роль надає їм необхідні дозволи для доступу до DynamoDB.
- **functions:** Опис наших Lambda-функцій. Вони обробляють запити від клієнтів та взаємодіють з DynamoDB.
  - **websocketHandler:** Функція обробляє WebSocket-з'єднання та реагує на різні події, такі як підключення, відключення, надсилання та отримання повідомлень, отримання списку клієнтів та користувачів.

- **plugins:** Плагіни Serverless Framework, які можуть розширити його функціональність. У цьому випадку використовується плагін serverless-plugin-typescript для підтримки TypeScript.
- **resources:** Ресурси AWS, які створюються нашим стеком Serverless. У цьому випадку створюються три таблиці DynamoDB:
    - Clients: Таблиця для зберігання інформації про підключених клієнтів, зокрема їх connectionId та nickname.
    - Users: Таблиця для зберігання інформації про користувачів, зокрема їх nickname.
    - Messages: Таблиця для зберігання повідомлень, які надсилаються між користувачами.

Файл handlers.ts містить код наших Lambda-функцій, написаний мовою TypeScript. Він експортує одну функцію handle, яка обробляє вхідні події від API Gateway.

- **handle:** Ця функція є точкою входу для ваших Lambda-функцій. Вона отримує **APIGatewayProxyEvent** як аргумент, який містить інформацію про запит, та повертає **APIGatewayProxyResult** як відповідь.
    - Вона витягує connectionId клієнта та routeKey (тип події) з події.
    - Залежно від routeKey, вона викликає відповідну функцію для обробки події:
        - handleConnect: Обробляє підключення нового клієнта, зберігаючи його дані в таблиці Clients та сповіщаючи інших клієнтів про зміни.
        - handleDisconnect: Обробляє відключення клієнта, видаляючи його дані з таблиці Clients та сповіщаючи інших клієнтів про зміни.
        - handleGetClients: Отримує список підключених клієнтів та надсилає його клієнту, який зробив запит.
        - handleGetUsers: Отримує список користувачів та надсилає його клієнту, який зробив запит.

- handleSendMessage: Обробляє надсилання повідомлення від одного клієнта до іншого. Зберігає повідомлення в таблиці Messages та надсилає його клієнту-одержувачу через WebSocket-з'єднання.
- handleGetMessages: Отримує повідомлення між двома конкретними користувачами з таблиці Messages та надсилає їх клієнту, який зробив запит.

- Інші функції допомагають основній функції handle виконувати свої завдання. Вони виконують такі дії, як:
  - Отримання інформації про клієнта за його connectionId.
  - Перевірка наявності клієнта за його nickname.
  - Зберігання даних в DynamoDB.

3 ОПИС ВПРОВАДЖЕННЯ ТА ВИКОРИСТАННЯ РОЗРОБКИ

3.1 Керівництво для адміністратора

Всі можливості адміністратора в даному випадку будуть у адмін-панелі AWS-Dashboard (рис. 3.1):

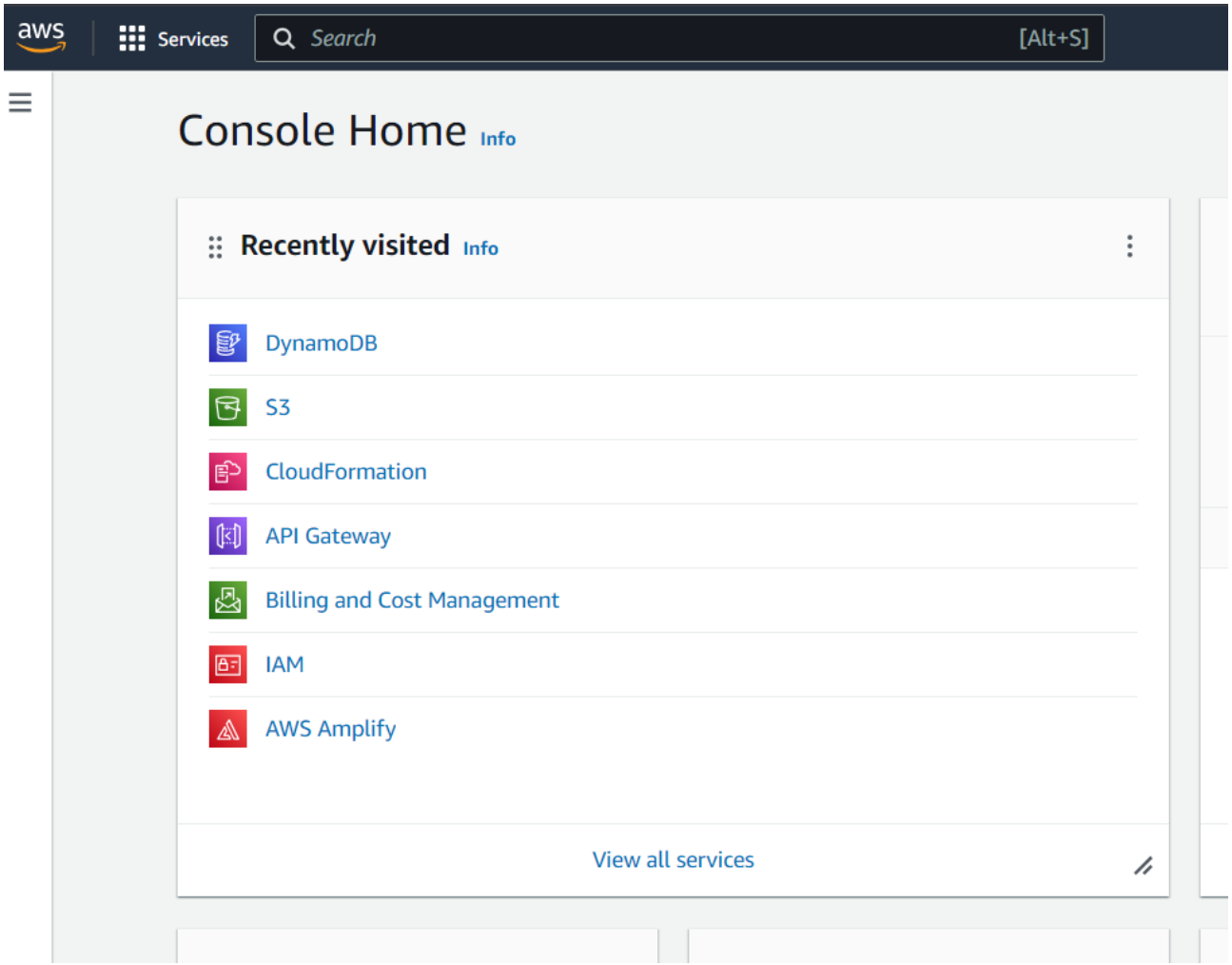

Рисунок 3.1 – Вид панелі адміністратора

Звідси є доступ до всіх потрібних сервісів, таких як DynamoDB, звідки адміністратор зможе керувати базою даних і S3 де він зможе керувати статичними файлами для додатку.

3.2 Керівництво для користувача

Інтерфейс же для користувача доволі простий. Він може зареєструватися на екрані логіну (рис. 3.2):

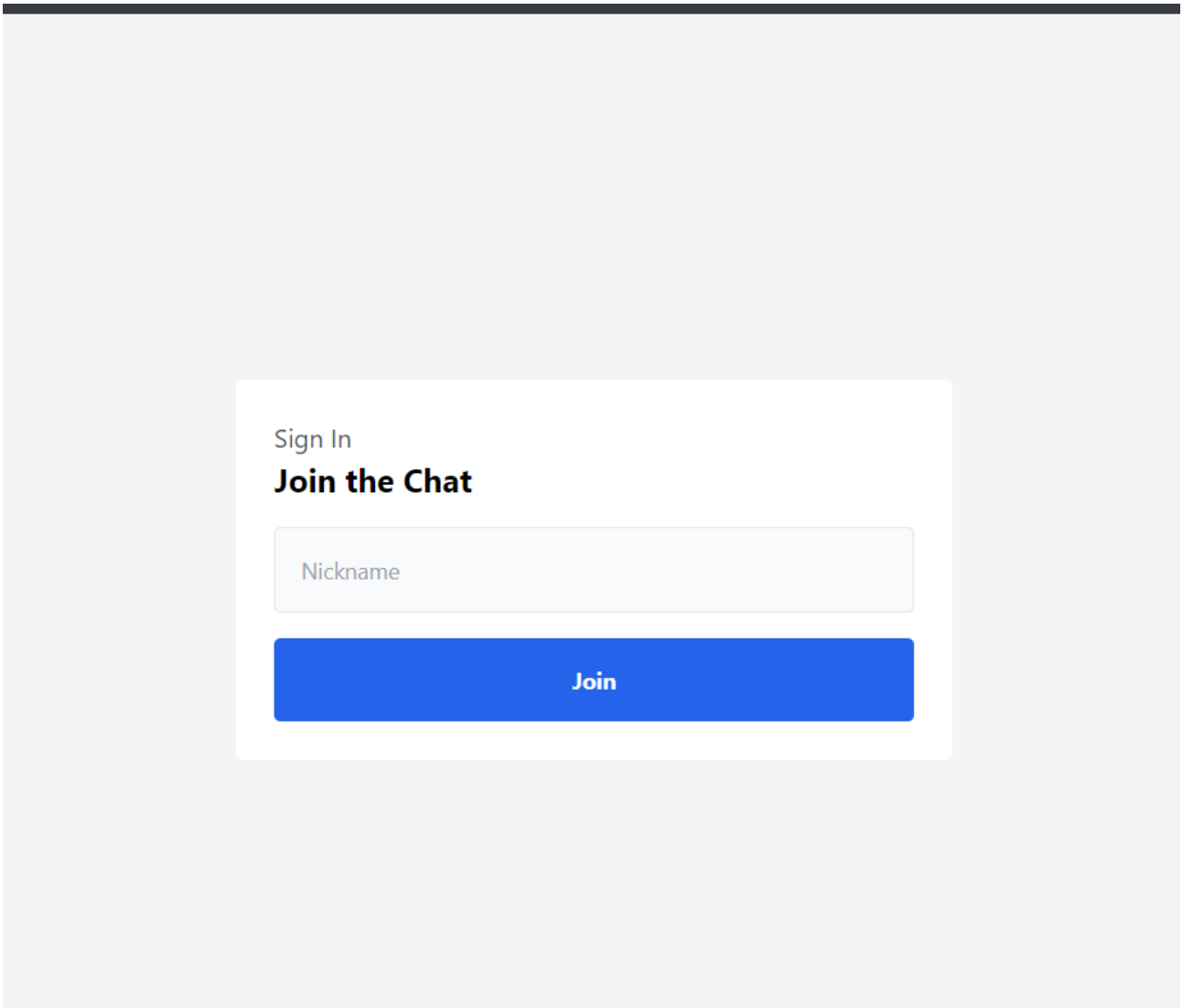

Рисунок 3.2 – Сторінка логіна

Після того як користувач введе свій логін та натисне кнопку “Join”, він опиниться на сторінці чатів (**Помилка! Джерело посилання не знайдено.** З

.3):

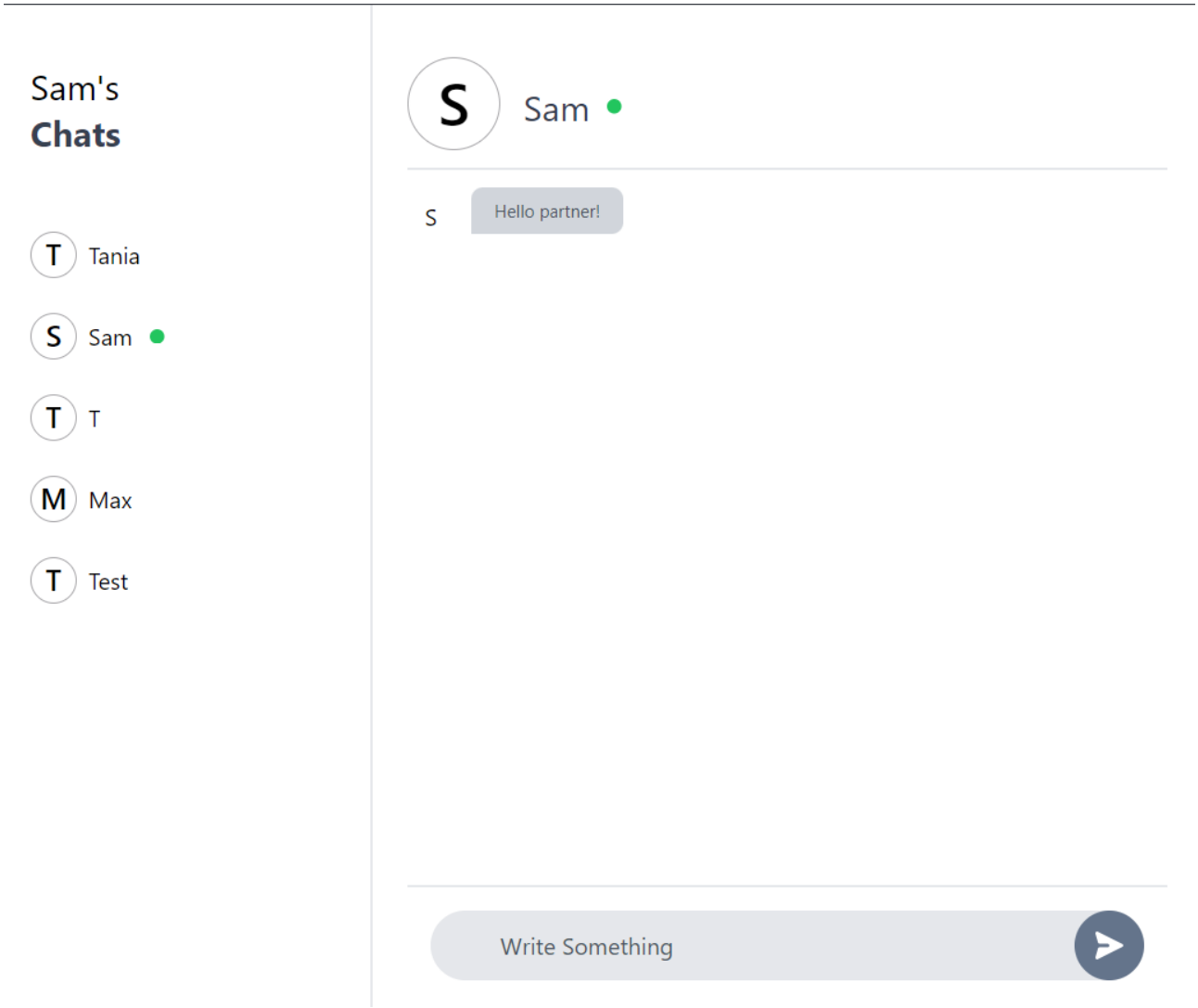

Рисунок 3.3 – Сторінка чатів

ВИСНОВКИ

В рамках цієї дипломної роботи було розроблено web-додаток на базі React та хмарних сервісів AWS (Amplify, Lambda, API Gateway, S3). Цей додаток являє собою простий web-месенджер/чат, де зареєстровані користувачі можуть відправляти та отримувати повідомлення.

Використання React дозволило створити динамічний та інтерактивний інтерфейс користувача, а інтеграція з AWS Amplify забезпечила просту та зручну розробку back-end частини. Завдяки AWS Lambda було реалізовано серверну логіку обробки повідомлень, а API Gateway надав безпечний доступ до серверних функцій. S3 використовується для зберігання зображень користувачів та інших даних.

Розроблений додаток має наступні можливості::

1) Реєстрація та авторизація користувачів;
2) Відправлення та отримання текстових повідомлень;
3) Зберігання історії повідомлень;

Цей проект продемонстрував ефективність React та AWS для розробки web-додатків з чатом. Завдяки використанню цих інструментів вдалося створити функціональний та масштабований додаток, який можна розширювати та вдосконалювати в майбутньому.

Якщо дивитись з точки практичної значимості то цей месенджер може бути використаний для створення корпоративних чатів, систем підтримки клієнтів, онлайн спільнот тощо.

ПЕРЕЛІК ПОСИЛАНЬ

1. Архітектура клієнт-сервер. ВСП «Павлоградський фаховий коледж НТУ «Дніпровська політехніка». URL: http://inter.ptngu.com/kompyuterni-merezhi/arhitektura-kliyent-server (дата звернення: 25.04.2024).

2. Документація з бібліотеки React.js. React. URL: https://react.dev/ (дата звернення: 23.02.2024).

3. Документація з бібліотеки по типізації. TypeScript: JavaScript With Syntax For Types. URL: https://www.typescriptlang.org/ (дата звернення: 25.03.2024).

4. Документація з сервісів AWS. Amazon Web Services. URL: https://docs.aws.amazon.com/ (дата звернення: 17.03.2024).

5. Документація з фреймворку Serverless. Serverless: Zero-Friction Serverless Apps On AWS Lambda & Beyond. URL: https://www.serverless.com/framework/docs (дата звернення: 07.03.2024).

6. Інформація про веб-сокети. Хабр. URL: https://habr.com/ru/sandbox/171066/ (дата звернення: 25.02.2024).

7. Невеличкий екскурс стосовно роботи з AWS-сервісами на прикладі розробки додатку для виклику єдинорогів замість таксі. Amazon Web Services, Inc. URL: https://aws.amazon.com/getting-started/hands-on/build-serverless-web-app-lambda-apigateway-s3-dynamodb-cognito/ (дата звернення: 04.04.2024).

8. Клієнт-серверна модель – Вікіпедія. Wikipedia, the free encyclopedia. URL: https://en.wikipedia.org/wiki/Client–server_model (дата звернення: 20.03.2024).

9. Стаття-огляд на веб-месенджери та їх призначення. Webwise.ie. URL: https://www.webwise.ie/parents/explained-what-is-messenger/ (дата звернення: 13.04.2024).

10. ДСТУ 3008-2015. Звіти у сфері науки і техніки. Структура та правила оформлювання. На заміну ДСТУ 3008-95 ; чинний від 2017-07-01. Вид. офіц. Київ, 1995. 25 с.

ДОДАТОК А

Програмний код, що забезпечує завантаження застосунку на сервер (serverless.yml)

```
frameworkVersion: "3"

provider:
  name: aws
  runtime: nodejs20.x
  lambdaHashingVersion: 20201221

  environment:
    WSSAPIGATEWAYENDPOINT:
      Fn::Join:
        - ""
        - - Ref: WebsocketsApi
          - ".execute-api."
          - Ref: AWS::Region
          - ".amazonaws.com/${sls:stage}"

  iam:
    role:
      statements:
        - Effect: Allow
          Action:
            - "dynamodb:PutItem"
            - "dynamodb:GetItem"
            - "dynamodb:DeleteItem"
```

```
      - "dynamodb:Scan"
    Resource:
      - { "Fn::GetAtt": ["ClientsTable", "Arn"] }
  - Effect: Allow
    Action:
      - "dynamodb:PutItem"
      - "dynamodb:GetItem"
      - "dynamodb:DeleteItem"
      - "dynamodb:Scan"
    Resource:
      - { "Fn::GetAtt": ["UsersTable", "Arn"] }
  - Effect: Allow
    Action:
      - "dynamodb:Query"
    Resource:
      Fn::Join:
        - "/"
        - - { "Fn::GetAtt": ["ClientsTable", "Arn"] }
          - "index"
          - "*"
  - Effect: Allow
    Action:
      - "dynamodb:PutItem"
      - "dynamodb:GetItem"
      - "dynamodb:DeleteItem"
      - "dynamodb:Scan"
    Resource:
      - { "Fn::GetAtt": ["MessagesTable", "Arn"] }
  - Effect: Allow
    Action:
```

```
          - "dynamodb:Query"
        Resource:
          Fn::Join:
            - "/"
            - - { "Fn::GetAtt": ["MessagesTable", "Arn"] }
              - "index"
              - "*"

functions:
  websocketHandler:
    handler: src/handlers.handle
    events:
      - websocket:
          route: $connect
      - websocket:
          route: $disconnect
      - websocket:
          route: getMessages
      - websocket:
          route: sendMessage
      - websocket:
          route: getClients
      - websocket:
          route: getUsers

plugins:
  - serverless-plugin-typescript

resources:
  Resources:
```

```
ClientsTable:
  Type: AWS::DynamoDB::Table
  Properties:
    TableName: Clients
    ProvisionedThroughput:
      ReadCapacityUnits: 1
      WriteCapacityUnits: 1
    AttributeDefinitions:
      - AttributeName: connectionId
        AttributeType: S
      - AttributeName: nickname
        AttributeType: S
    KeySchema:
      - AttributeName: connectionId
        KeyType: HASH
    GlobalSecondaryIndexes:
      - IndexName: NicknameIndex
        KeySchema:
          - AttributeName: nickname
            KeyType: HASH
        ProvisionedThroughput:
          ReadCapacityUnits: 1
          WriteCapacityUnits: 1
        Projection:
          ProjectionType: "ALL"
UsersTable:
  Type: AWS::DynamoDB::Table
  Properties:
    TableName: Users
    ProvisionedThroughput:
```

```
      ReadCapacityUnits: 1
      WriteCapacityUnits: 1
    AttributeDefinitions:
      - AttributeName: nickname
        AttributeType: S
    KeySchema:
      - AttributeName: nickname
        KeyType: HASH
    GlobalSecondaryIndexes:
      - IndexName: NicknameIndex
        KeySchema:
          - AttributeName: nickname
            KeyType: HASH
        ProvisionedThroughput:
          ReadCapacityUnits: 1
          WriteCapacityUnits: 1
        Projection:
          ProjectionType: "ALL"
MessagesTable:
  Type: AWS::DynamoDB::Table
  Properties:
    TableName: Messages
    ProvisionedThroughput:
      ReadCapacityUnits: 1
      WriteCapacityUnits: 1
    AttributeDefinitions:
      - AttributeName: messageId
        AttributeType: S
      - AttributeName: createdAt
        AttributeType: N
```

```
  - AttributeName: nicknameToNickname
    AttributeType: S
KeySchema:
  - AttributeName: messageId
    KeyType: HASH
  - AttributeName: createdAt
    KeyType: RANGE
GlobalSecondaryIndexes:
  - IndexName: NicknameToNicknameIndex
    KeySchema:
      - AttributeName: nicknameToNickname
        KeyType: HASH
      - AttributeName: createdAt
        KeyType: RANGE
    ProvisionedThroughput:
      ReadCapacityUnits: 1
      WriteCapacityUnits: 1
    Projection:
      ProjectionType: "ALL"
```