МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: **«РЕАЛІЗАЦІЯ WEBMAIL СЕРВІСУ»**

Виконав: студент 2 курсу, групи 8.1210
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми інженерія програмного забезпечення
(назва освітньої програми)
М.Д. Марчук
(ініціали та прізвище)

Керівник доцент кафедри програмної інженерії, доцент, к.ф.-м.н. Горбенко В.І.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент доцент кафедри фундаментальної та прикладної математики, доцент, к.ф.-м.н. Панасенко Є.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2021

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти магістр

Спеціальність 121 інженерія програмного забезпечення

(шифр і назва)

Освітня програма інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент
_____ Лісняк А.О.
(підпис)
« 09 » 06 2021 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Марчуку Максиму Денисовичу

(прізвище, ім'я та по-батькові)

1. Тема роботи (проекту) Реалізація Webmail сервісу

керівник роботи (проекту) Горбенко Віталій Іванович, к.ф.-м.н., доцент

(прізвище, ім'я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 09 » червня 2021 року № 851-с

2. Строк подання студентом роботи 24.11.2021

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

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Збір даних.
2. Реалізація.

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
1	Горбенко В. І., доцент. кафедри ПІ	09.06.2021	10.09.2021
2	Горбенко В. І., доцент. кафедри ПІ	13.09.2021	03.11.2021
3	Горбенко В. І., доцент. кафедри ПІ	04.11.2021	12.11.2021

7. Дата видачі завдання 09.06.2021

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	09.06.2021	
2.	Збір вихідних даних.	16.06.2021	
3.	Обробка методичних та теоретичних	20.06.2021	
	джерел.		
4.	Розробка першого розділу.	03.07.2021	
5.	Розробка другого розділу.	14.09.2021	
6.	Оформлення та нормоконтроль	13.11.2021	
	кваліфікаційної роботи.		
7.	Захист кваліфікаційної роботи.	17.12.2021	

Студент _____ М.Д. Марчук
(підпис) (ініціали та прізвище)

Керівник роботи _____ В.І. Горбенко
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ С.П. Швидка
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота магістра «Реалізація Webmail сервісу»: 44 с., 30 рис., 11 джерел.

ВЕБ-БРАУЗЕР, ВЕБ-ПОШТА, ВЕБ-КЛІЄНТ, СЕРВІС, НАСКРІЗНЕ ШИФРУВАННЯ.

Об’єкт дослідження – реалізація Webmail сервісу.

Мета роботи – розробка клієнтської програми електронної пошти у веб-браузері.

Методи дослідження – методи об’єктно-орієнтованого програмування.

В даній роботі висвітлюються етапи розробки Webmail сервісу.

Під час збору даних були оброблені методичні матеріали по використанню протоколів для трансферу та обробки електронних листів POP3, IMAP, SMTP, а також засоби наскрізного шифрування даних.

Реалізація під собою передбачає розробку на мові JavaScript із використанням додаткових технологій, як: React, Redux, npm, Webpack, Babel, Express тощо.

Під час розробки окремих частин програмного продукту були взяті за основу декілька парадигм програмування:

– структурна (розробка серверу);

– об’єктно-орієнтоване з використанням автоматного програмування (розробка основних компонентів програми).

Підсумковим результатом розробки вважається функціонуючий Webmail сервіс.

SUMMARY

Master's qualifying paper "Implementation of Webmail Service": 44 pages, 30 figures, 11 references.

BROWSER, WEBMAIL, E2EE, SERVICE, WEB-CLIENT.

The object of the study is the implementation of Webmail service.

The aim of the study is to develop an email client in a browser.

The methods of research are methods of object-oriented programming.

This paper highlights the stages of development of Webmail service.

During the collection of information, methodical materials on the use of protocols for the transfer and processing of e-mails POP3, IMAP, SMTP, as well as means of end-to-end data encryption were processed.

The implementation involves development via JavaScript language using additional technologies such as React, Redux, npm, Webpack, babel, express etc.

During the development of some parts of the software product were taken as a basis for several programming paradigms:

– structural (server development);

– object-oriented with the use of automatic programming (development of the main components of the program).

The final result of the development is considered a functioning Webmail service.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

Веб-браузер або браузер	– (від англ. web browser) програмне забезпечення для перегляду сторінок, веб-документів, комп’ютерних файлів; керування веб-додатками та ін.;
URL	– Uniform Resource Locator, стандартизована адреса певного ресурсу в Інтернеті;
API	– Application Programming Interface, інтерфейс програмування додатків;
DOM	– Document Object Model;
СУБД	– Система управління базами даних, набір взаємопов'язаних даних і програм для доступу до цих даних, що надають можливості створення, збереження, оновлення та пошуку інформації в базах даних;
HTTP	– Hypertext Transfer Protocol, протокол передачі гіпертекстових документів в комп’ютерних мережах;
HTML	– Hypertext Markup Language;
JWT	– JSON Web Token, стандарт токена доступу на основі JSON, що використовується для верифікації тверджень.

ВСТУП

Електронна пошта [1] використовується повсякденно та вже стала невід’ємною частиною життя багатьох людей. Реєстрація облікових записів на більшості веб-сайтів включає в себе і вказання електронної адреси, що унеможливлює використання деяких веб-продуктів без наявної адреси електронної пошти.

Як і звична фізична пошта, електронна служить для відправлення та отримання листів. Електронний лист складається із заголовків, на яких зазначена службова інформація (відправник, отримувач, тема, отримувачі копії, тип змісту) та зміст повідомлення. Змістом повідомлення може бути майже будь-який тип даних, будь то відео-/аудіо-запис, документ чи форматований текст.

Перший прототип е-пошти працював на комп'ютерах в Массачусетському Технологічному Інституті (MIT) в програмі «MAILBOX» 1965 року. Користувачі комп'ютерів в MIT могли залишати повідомлення в цій програмі на комп'ютерах університету для інших користувачів, які бачили ці повідомлення наступного разу, коли ввійдуть на комп'ютер. А перший електронний лист було надіслано Реєм Томлінсоном самому собі 1971 року.

Після появи розподіленої глобальної системи імен DNS для вказівки адреси стали використовуватися доменні імена — username@domain.com — користувач username на машині domain.com. Одночасно з цим для пошти стали використовуватися виділені сервери, на які не мали доступу звичайні користувачі, при цьому пошта приходила не на робочі машини користувачів, а на поштовий сервер, звідки користувачі забирали свою пошту за різними мережевими протоколами (найпоширеніші з них POP3, IMAP).

1 ЗБІР ДАНИХ

1.1 Наскрізне шифрування

Наскрізне шифрування – це шифрування даних, що виключає можливість отримання доступу до криптографічних ключів третім особам, що, в свою чергу, забезпечує безпечний обмін повідомленнями між двома. Для обміну ключами можна застосовувати симетричний і асиметричний алгоритми. Наскрізне шифрування передбачає, що ключі шифрування відомі тільки сторонам, що спілкуються між собою, тож для реалізації цієї умови може бути використана схема з попереднім поділом секрету або, наприклад, протокол Діффі-Хелмана, який використовується в месенжерах WhatsApp та Telegram.

Шляхом аналізу сучасних додатків використовуючих алгоритми наскрізного шифрування був обраний алгоритм управління ключами «Double Ratchet» [2] та алгоритм узгодження ключей «X3DH» [3] з месенжеру Signal.

Алгоритм подвійного храповика (англ. Double Ratchet Algorithm) – алгоритм управління ключами, розроблений Тревором Перріном (англ. Trevor Perrin) та Моксі Марлінспайком (англ. Moxie Marlinspike) у 2013 році. Цей алгоритм може бути використаний як частина криптографічного протоколу для того, щоб забезпечити наскрізне шифрування для обміну зашифрованими повідомленнями з урахуванням спільного секретного ключа. Зазвичай, сторони використовують протокол початкової угоди Extended Triple Diffie-Hellman (X3DH), у якому відбувається потрійний обмін ключами у тому, щоб узгодити спільний секретний ключ. Після цього учасники з'єднання використовуватимуть Double Ratchet для надсилання та отримання зашифрованих повідомлень. При кожному повідомленні Double Ratchet сторони повинні отримувати нові ключі для кожного повідомлення Double Ratchet, щоб попередні ключі не могли бути розраховані з пізніших. Також

учасники з'єднання направляють загальні значення Діффі-Хеллмана, що додаються до їх повідомлень. Результати обчислень Діффі-Хелмана поєднуються з похідними ключами, так що пізніші ключі не можуть бути розраховані з більш ранніх. Ці показники дають певний захист зашифрованим повідомленням, що приходять раніше злому чи після нього, у разі компрометації ключів сторони.

Розширений потрійний протокол узгодження ключів Діффі-Хелмана (англ. Extended Triple Diffie-Hellman Key Agreement protocol, скор. X3DH) – це протокол встановлення спільного секретного ключа із взаємною автентифікацією сторін на основі відкритих ключів. X3DH доповнює стандартний протокол узгодження ключів Діффі-Хелмана взаємною автентифікацією обох сторін, а також забезпечує пряму секретність (англ. forward secrecy) та криптографічну стійкість.

Враховуючи технології та вимоги для успішного використання наскрізного шифрування було вирішено обрати бібліотеку 2key-ratchet [8], в якій реалізован програмний інтерфейс дотримуючись об’єктно-орієнтованої парадигми. Обрана бібліотека всередині себе інкапсулює реалізацію алгоритмів та протоколів, що дозволяє використовувати її інтерфейс у вигляді класів і не контролювати взаємодію ключів власноруч.

1.2 Протоколи отримання та відправки е-листів POP3, IMAP, SMTP

Перш ніж розробляти систему обміну електронними листами, потрібно було дослідити засоби відправки та отримання е-листів в Інтернеті. Таким чином було виділено 2 основні протоколи отримання е-листів – POP3 і IMAP, а також протокол відправки SMTP [4].

Post Office Protocol (POP3) – це протокол отримання електронних листів серед вищевказаних. Е-листи завантажуються з серверу на пристрій користувача та після цього знищуються на сервері, надалі взаємодія з

отриманими листами виконується на стороні клієнта навіть якщо немає доступу в Інтернет. Недоліки:

– ризик зараження шкідливим програмним забезпеченням (вірусами) при завантаженні всіх листів на пристрій;

– незручність використання пошти на декількох пристроях;

– необхідність резервування пошти завантаженої на пристрій.

Interent Message Access Protocol (IMAP) – удосконалена версія протоколу POP3, хоча і не є його безпосереднім нащадком. Е-листи зберігаються та обробляються на сервері, тобто є можливість роботи з поштою через декілька пристроїв. Недоліки:

– неможливість отримання пошти без доступу в Інтернет;

– цілісність та стан пошти залежить від стану серверу.

Simple Mail Transfer Protocol (SMTP) – це синхронний протокол для відправки електронних листів, в якому з'єднання відбувається лише за ініціативи відправника, тобто односторонньо. Протокол складається із серії команд, що посилаються клієнтом, та відповідей серверу. Враховуючи, що як таких аналогів протоколу відправки немає, то й немає необхідності шукати і порівнювати недоліки.

Проаналізувавши та поглибившись у специфікації і призначення протоколів, було вирішено будувати систему взаємодіючу з SMTP-серверами.

1.3 Сервіс електронної пошти AWS SES

Оскільки SMTP-сервери повинні взаємодіяти з серверами провайдерів Інтернету, то розробка власного SMTP-серверу не є раціональним варіантом. Існує безліч платних SMTP-провайдерів з налаштованим середовищем та безліччю функцій, як: вірус-/спам-фільтри, різноманітний API, цілодобова підтримка, панель приладів (dashboard) тощо.

Одним із таких провайдерів послуг відправки та отримання електронної пошти є Simple Email Service [5] від Amazon Web Services (AWS).

Simple Email Service (SES) – це економічний, гнучкий та масштабований сервіс електронної пошти від AWS, за допомогою якого розробники можуть надсилати електронні листи з будь-якої програми. Можна швидко налаштувати AWS SES і обрати кілька варіантів використання електронної пошти, включаючи надсилання транзакцій, маркетингових листів або виконання масової розсилки. Сервіс включає різні можливості розгортання IP-адрес та автентифікації електронною поштою, які дозволяють підвищити ефективність доставки та захистити репутацію відправника, а також надає аналітику, за допомогою якої можна проаналізувати ефективність кожного відправленого листа.

Насамперед, для роботи з сервісом потрібно зазначити домен або електронну адресу та підтвердити право власності ним. Зазначивши домен потрібно внести DKIM-записи до DNS-реєстру аби сервер сервісу міг звернутися до серверу провайдера домену, що, в свою чергу, буде підтвердженням права власності доменом. Таким чином, додані DKIM-записи будуть посилатися на адреси SMTP-серверів сервісу SES, що позволить перенаправляти е-листи до цих адрес.

Для можливості отримання повідомлень потрібно виконати декілька кроків:

– створити кошик (bucket) в Simple Storage Service [6] від AWS для зберігання вхідних е-листів;
– створити кінцеву точку (endpoint) на сервері додатку для обробки сповіщень про вхідні повідомлення;
– створити тему (topic) в Simple Notification Service [7] від AWS, яка є так званим «каналом повідомлень», що розсилає сповіщення до зазначених кінцевих точок при отриманні повідомлення від обраного раніше джерела;

– вказати раніше створений кошик з S3 в сервісу SES для збереження вхідних е-листів;
– вказати раніше створену тему з SNS в сервісу SES для відсилання сповіщення до кінцевої точки про надходження нового е-листа.

1.3.1 Сервіс хмарного зберігання даних AWS S3

Simple Storage Service (S3) – це сервіс зберігання об'єктів, що пропонує найкращі в галузі показники продуктивності, масштабованості, доступності та безпеки даних від AWS. Клієнти будь-якої величини та з будь-якої промислової галузі можуть зберігати та захищати необхідний обсяг даних для практично будь-якого прикладу використання. Наприклад, для озер даних, хмарних додатків та мобільних додатків. Вигідні класи сховища та прості у використанні інструменти адміністрування дозволяють оптимізувати витрати, організувати дані та точно налаштувати обмеження доступу відповідно до потреб бізнесу чи законодавчих вимог.

1.3.2 Сервіс обміну повідомленнями AWS SNS

Simple Notification Service (SNS) – це повністю керований сервіс обміну повідомленнями для зв'язку між програмами (A2A), а також між програмами та користувачами (A2P) від AWS.

Функції підписки та публікації A2A надають теми з високою пропускною здатністю для push-повідомлень між розподіленими системами, мікросервісами та безсерверними програмами на основі подій за моделлю «багато до багатьох». Використовуючи теми AWS SNS, системи публікацій можуть надсилати повідомлення для паралельної обробки великому числу систем. Функціональні можливості A2P дозволяють надсилати користувачам будь-яку кількість повідомлень у вигляді SMS, мобільних push-повідомлень та електронних листів.

2 РЕАЛІЗАЦІЯ

2.1 Ініціалізація проекту

При виконанні поточної роботи використовувалася мова програмування JavaScript, яка є найпопулярнішою мовою в сфері веб-розробки.

Також для швидкого та зручного керування зовнішніми пакетами та модулями використовувався менеджер пакетів npm для якого потрібна бути встановлена програмна платформа Node.js. Менеджер пакетів npm дає можливість встановити та використати будь-який публічний модуль в реєстрі npm за назвою.

Для того, щоб усі необхідні файли проекту були оптимізовані та зібрані в одному місці, використовувався пакувальник модулів Webpack. Він дозволяє перетворювати формати файлів у необхідні для фінальної версії продукту, також оптимізуючи їх. Також Webpack пропонує використовувати вбудований HTTP-сервер (Webpack Dev Server) для розробки, щоб без необхідності не створювати сервер власноруч для перевірки розробленого інтерфейсу користувача; найнеобхідніше, що входить в його можливості: автоматичне перепаковування змінених файлів, оптимізація та стиснення упакованих файлів.

2.2 Розробка серверної частини

В даній роботі сервер був створений за допомогою фреймворку Express для Node.js [9]. Express дає можливість створювати сервер, додавати маршрути до нього та підключати плаґіни, такі, як CORS [10] (Cross-Origin Resource Sharing), bodyParser тощо.

Перш ніж розробляти структуру маршрутів для взаємодії з API додатку, необхідно створити базу даних для зберігання даних користувачів та їх налаштування, повідомлення та вкладення до них використовуючи систему управління базою даних (СУБД) PostgreSQL. PostgreSQL є однією з найпопулярніших доступних СУБД для зручної маніпуляції інформаційними одиницями в реляційній базі даних.

Була створена база даних з чотирма таблицями (див. рис 2.1):

1) users (див. рис. 2.2);

2) settings (див. рис. 2.3);

3) messages (див. рис. 2.4);

4) messages_attachments (див. рис. 2.5).

Рисунок 2.1 – Структура бази даних

id	bigint		not null	nextval('users_id_seq'::regclass)
username	character varying(64)		not null	
fullname	character varying(100)			
avatar_link	text			
bio	character varying(200)			''::character varying
status	enum_users_status		not null	'active'::enum_users_status
type	enum_users_type		not null	'public'::enum_users_type
public_key	text			
is_public_key_hidden	boolean		not null	true
active_at	timestamp with time zone			
created_at	timestamp with time zone		not null	
updated_at	timestamp with time zone		not null	
email	character varying(254)			

Рисунок 2.2 – Структура таблиці users

id	bigint		not null	nextval('settings_id_seq'::regclass)
user_id	bigint		not null	
export_on_log_out	boolean			true
notifications_email	text			''::text
s3_object_url	text			''::text
updated_at	timestamp with time zone		not null	

Рисунок 2.3 – Структура таблиці settings

id	bigint		not null	nextval('messages_id_seq'::regclass)
subject	character varying(100)		not null	
sender	character varying(254)		not null	
receiver	character varying(254)		not null	
text	text		not null	
backup	text		not null	''::text
status	enum_messages_status		not null	'pending'::enum_messages_status
sent_at	timestamp with time zone		not null	
updated_at	timestamp with time zone		not null	
payload	json			
is_external	boolean		not null	false
carbon_copy	json			'[]'::json
html	text		not null	''::text

Рисунок 2.4 – Структура таблиці messages

id	bigint		not null	nextval('messages_attachments_id_seq'::regclass)
message_id	bigint			
filename	text			'unnamed-file'::text
type	character varying(200)			
url	text		not null	
last_modified	timestamp with time zone			
created_at	timestamp with time zone			

Рисунок 2.5 – Структура таблиці messages_attachments

Для того, щоб програмний код серверної частини сервісу міг взаємодіяти безпосередньо з базою даних, необхідно використовувати Object-Relational Mapping (ORM) технології, як Sequelize [9]. Sequelize ORM дозволяє маніпулювати базою даних, зокрема, таблицями. Для того, щоб додати можливість взаємодії серверу з базою даних потрібно:

– створити підключення до СУБД (надати назву бази даних, облікові дані та задати опції);
– створити відповідні моделі для кожної з таблиць (User, Settings, Message, MessageAttachments).

2.2.1 Проміжні обробники

Перш ніж створювати маршрути необхідно створити проміжні обробники (middleware), які можуть змінювати або переривати запити по необхідності.

Необхідно було створити наступні проміжні обробники.

1. isAuth(shouldBeAuthenticated):

    1) Якщо запит містить значення у заголовку «Authorization», яке є токеном із зашифрованим ім’ям користувача, що існує в таблиці

«users», то отримати дані цього користувача за його ім’ям та зберегти їх у властивості «requester» об’єкту запиту, оновити поле «activeAt» цього користувача теперішньою датою, передати запит далі до обробників;

2) Якщо запит не містить значення у заголовку «Authorization», то встановити 401 код стану запиту та відповісти зі значенням за замовчуванням «Unauthorized», тобто «Неавторизований»;

3) Якщо запит містить некоректне значення у заголовку «Authorization» і воно не може бути розшифровано, то встановити 403 код стану запиту та відповісти зі значенням за замовчуванням «Forbidden», тобто «Заборонено»;

4) Якщо запит містить коректне значення у заголовку «Authorization», але в таблиці «users» запис, маючий ім’я користувача за таким значенням, відсутній, то встановити 404 код стану запиту та відповісти зі значенням «User does not exist», тобто «Користувач не існує»;

5) Якщо запит містить коректне значення у заголовку «Authorization», тобто користувач автентифікований, але проміжний обробник був викликаний з аргументом «shouldBeAuthenticated» із значенням «false», що забороняє автентифікованому користувачу надсилати запит до маршруту, на якому встановлений цей проміжний обробник, то встановити 403 код стану запиту та відповісти зі значенням за замовчуванням «Forbidden», тобто «Заборонено».

2. handleAvatar() – створений для збереження зображення профілю при обробці запиту на оновлення даних користувача:
    1) Якщо файл був наданий у об’єкті запиту, то отримати назву файлу, отримати посилання на теперішнє зображення профілю користувача через властивість запиту «requester» та, якщо посилання існує, то видалити об’єкт з кошику AWS S3 за

посиланням, зберегти нове зображення профілю в кошику AWS S3, отримати URL збереженого об’єкту та зберегти у властивості тіла запиту під назвою «avatarLink», передати об’єкт запиту далі в обробник;

2) Якщо файл не був наданий у об’єкті, то передати об’єкт запиту далі в обробник.

2.2.2 Маршрути

Маршрути – це кінцеві точки для звернення до серверу за допомогою HTTP-методів запиту, таких як:

– GET для отримання даних;

– POST, PUT для створення об’єктів;

– PATCH для оновлення об’єктів;

– DELETE для видалення об’єктів.

Система маршрутизації на сервері складається з наступних маршрутів та обробників.

1. /auth для керування обліковими записами:
    1) POST-запит /auth/signup для реєстрації нового облікового запису;
    2) GET-запит /auth/login/:username для входу в обліковий запис за ім’ям користувача.
2. /user для керування даними користувача:
    1) PATCH-запит /user для оновлення даних користувача за його e-адресою;
    2) GET-запит /user/search для отримання даних користувачів за їх ім’ям користувача та повним ім’ям;
    3) GET-запит /user/profile для отримання даних одного користувача за його ідентифікатором, ім’ям користувача або електронною адресою;

4) PATCH-запит /user/settings для оновлення налаштувань облікового запису користувача за його ідентифікатором.

3. /message для керування повідомленнями:
    1) POST-запит /message для відправлення повідомлення;
    2) GET-запит /message/:status для отримання повідомлень за статусом;
    3) PUT-запит /message/attachment для збереження вкладення;
    4) DELETE-запит /message/attachment для видалення вкладення.
4. /aws для керування сервісами AWS:
    1) POST-запит /aws/sns для обробки сповіщень (про нові е-листи) з AWS SNS.

2.3 Розробка клієнтської частини

Перш за все потрібно ініціалізувати npm-модуль і створити package.json, щоб було можливим встановлювати зовнішні npm-модулі. Далі створити конфігураційні файли для транспілятора Babel, пакувальника Webpack та менеджера змінних середовища dotenv.

Наступним кроком потрібно створити структуру проекту (див. рис. 2.6 та рис. 2.7).

Для того, щоб скористатися сервісом, користувачеві потрібно створити обліковий запис, а потім увійти в нього. Маючи домашню сторінку для неавтентифікованих користувачів за маршрутом «/home» (див. рис. 2.8) було вирішено розробити систему автентифікації.

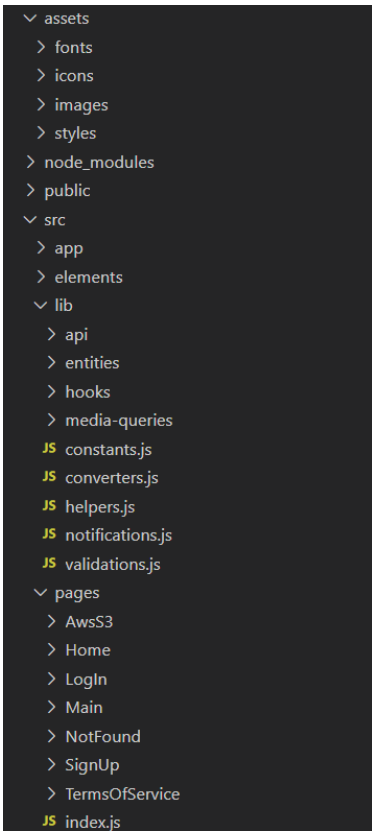

Рисунок 2.6 – Перша частина структури клієнтської частини

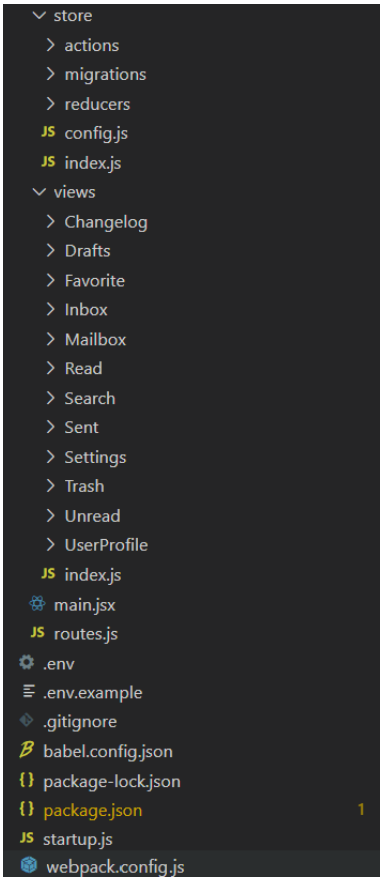

Рисунок 2.7 – Друга частина структури клієнтської частини

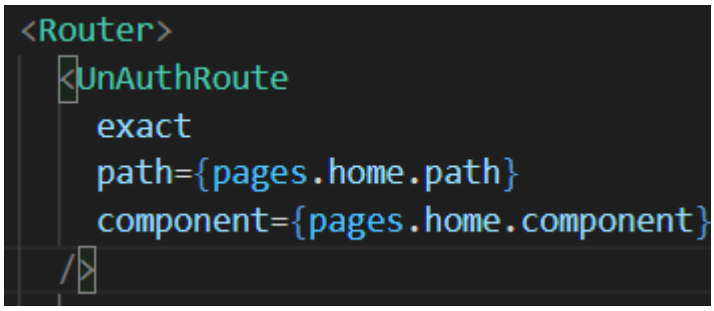

Рисунок 2.8 –Маршрут домашньої сторінки

Потік дій для створення облікового запису такий:

1) Перейти за маршрутом «/sign-up»;

2) Ввести ім’я користувача;

3) Ввести повне ім’я (за бажанням);

4) Натиснути кнопку «Sign Up»;

5) Скопіювати та зберегти згенерований ключ;

6) Увімкнути прапорець;

7) Натиснути кнопку «Continue».

Дотримуючись потоку дій перш за все потрібно створити маршрут «/sign-up» за допомогою модулю «react-router» [11], але до цього маршруту можуть мати доступ лише неавтентифіковані користувачі, тому додатково потрібно створити 2 компоненти.

Компонент «AuthRoute», що є компонентом вищого порядку та використовує компонент «Route» з модулю «react-router», дозволяє умовно відображати дочірні компоненти. Якщо користувач автентифікований, тобто має збережену властивість «identity» відмінну від значення «null» в локальному сховищі браузеру, то відобразити дочірні компоненти для нього, інакше перенаправити користувача на маршрут «/home».

Компонент «UnAuthRoute», що є компонентом вищого порядку та використовує компонент «Route» з модулю «react-router», дозволяє умовно відображати дочірні компоненти. Якщо користувач неавтентифікований, тобто збережене значення властивості «identity» дорівнює «null» або не

визначене в локальному сховищі браузеру, то відобразити дочірні компоненти для нього, інакше перенаправити користувача на маршрут «/».

З використанням компоненту «UnAuthRoute» був створений маршрут (див. рис. 2.9) «/sign-up», при переході на який відображається сторінка реєстрації нового користувача (див. рис. 2.10):

– поле для вводу ім’я користувача;

– поле для вводу повного ім’я;

– кнопка «Log In instead» для переходу на сторінку входу до облікового запису;

– кнопка «Sign Up» для завершення реєстрації облікового запису;

– кнопка «Back to home» для повернення на сторінку «Home».

```
<Router>
  <Switch>
    <UnAuthRoute
      exact
      path={pages.home.path}
      component={pages.home.component}
    />

    <UnAuthRoute
      exact
      path={pages.signUp.path}
      component={pages.signUp.component}
    />
```

Рисунок 2.9 – Маршрут сторінки реєстрації нового користувача

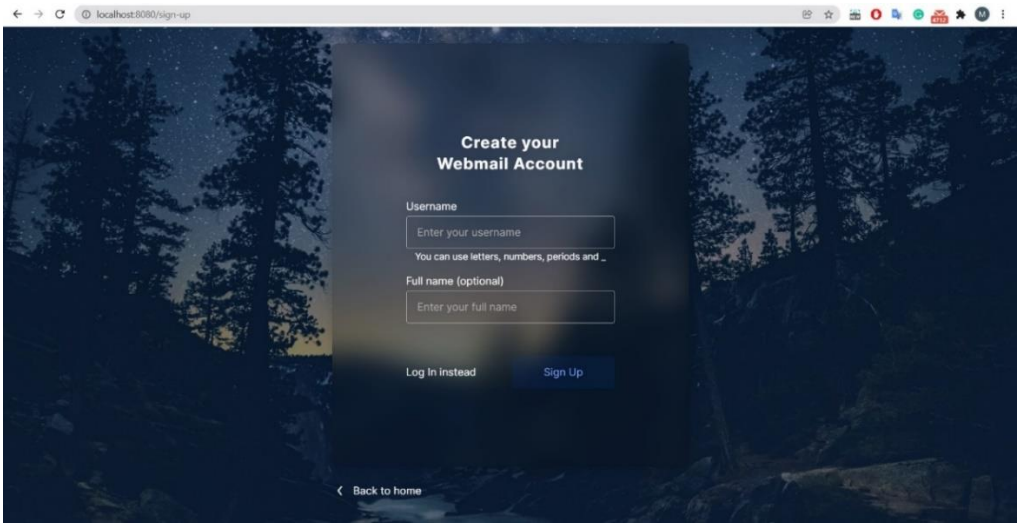

Рисунок 2.10 – Сторінка реєстрації нового користувача

Після введення даних в поле для введення ім’я користувача, кнопка «Sign Up» стає активна, після натискання на яку виконуються такі дії:

– форматуються ім’я користувача та повне ім’я;
– створюється особистість (identity) користувача та перетворюється в текстовий ключ;
– створюється відкритий ключ особистості нового користувача;
– ім’я користувача, повне ім’я та відкритий ключ користувача компонуються та запит на реєстрацію нового користувача надсилається на сервер;
– дані нового користувача зберігаються в локальному сховищі браузеру для подальшого входу в систему;
– виконується вхід користувача в систему;
– JWT та ключ користувача зберігаються в локальному сховищі браузера.

Після успішної реєстрації користувача відображається відповідна сторінка (див. рис 2.11), яка містить такі елементи:

– ключ користувача в текстовому вигляді;
– кнопка копіювання ключа;
– кнопка завантаження ключа;
– прапорець для підтвердження збереження ключа;
– кнопка «Continue» для переходу на головну сторінку.

Підхід до розробки системи автентифікації в сервісі полягає в тому, що особистий ключ генерується під час реєстрації та надається користувачеві лише один раз для збереження, після чого цей ключ може бути використаний користувачем для входу в систему шляхом вставки або завантаження у вигляді файлу. Особисті ключі користувачів не зберігаються на сервері сервісу, що надає користувачеві впевненості в тому, що відповідальність за цілісність даних облікового запису лежить на ньому.

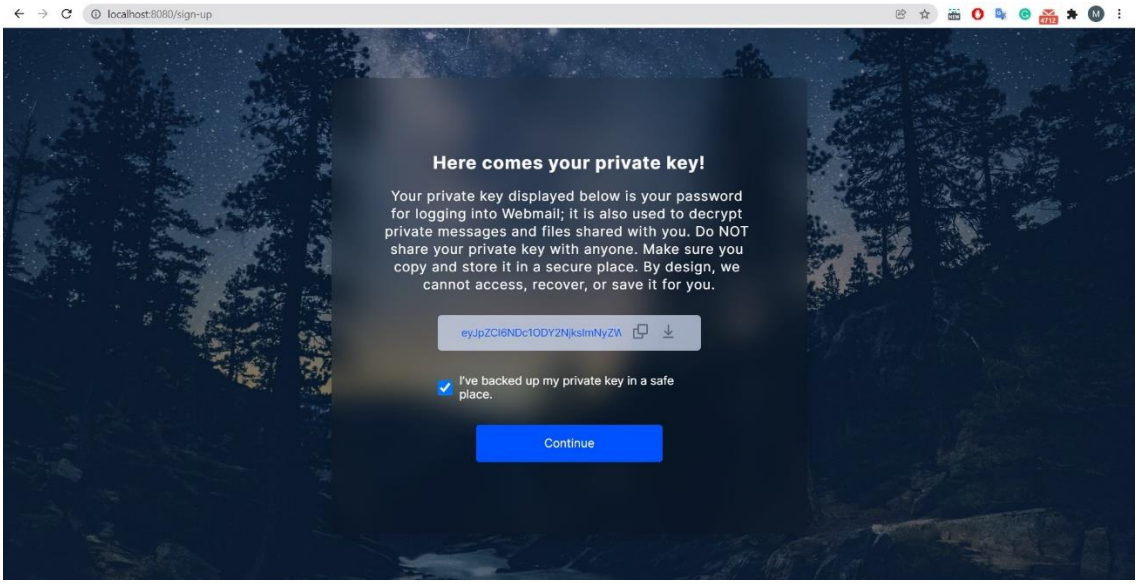

Рисунок 2.11 – Сторінка успішної реєстрації нового користувача

Далі, враховуючи, що користувачі мають можливість створити обліковий запис, то наступним етапом має бути вхід до створеного облікового запису.

Вхід до облікового запису користувача має відбуватися на відповідній сторінці. Сторінка доступна на маршруті «/log-in» та містить такі елементи (див. рис. 2.12):

– поле для введення ім’я користувача;

– поле для введення особистого ключа користувача;

– кнопка для завантаження файлу з особистим ключем користувача;

– кнопка «Sign Up instead» для переходу на сторінку реєстрації нового користувача;

– кнопка «Log In» для входу в систему;

– кнопка «Back to home» для повернення на домашню сторінку.

Потік дій для входу в систему такий:

1) Ввести ім’я користувача;

2) Вставити особистий ключ в поле для введення або завантажити за допомогою відповідної кнопки;

3) Натиснути кнопку «Log In».

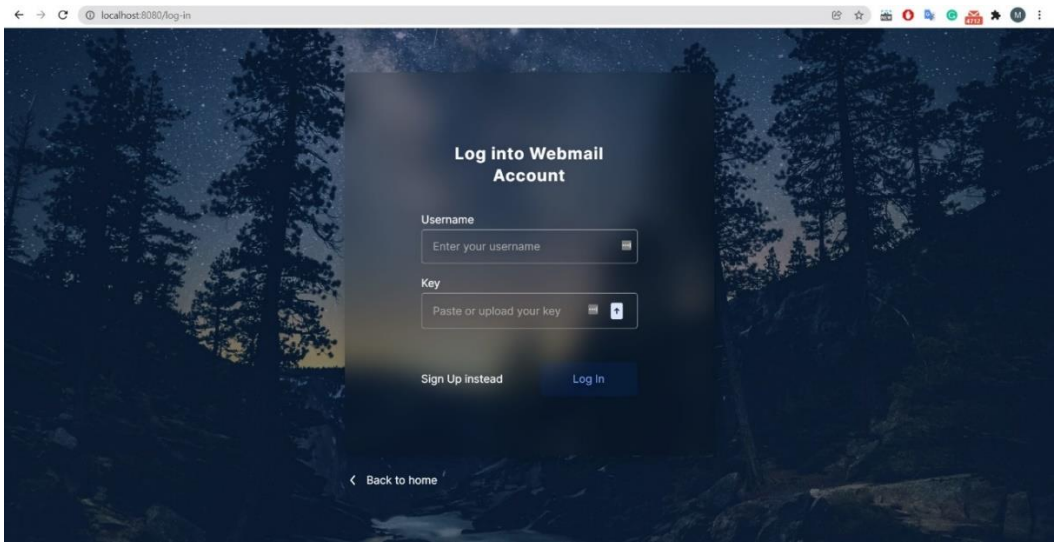

Рисунок 2.12 – Сторінка входу в систему

Під час входу в систему виконуються такі дії:

1) З особистого ключа створюється особистість у вигляді класу;
2) Ім’я користувача форматується та надсилається на сервер у запиті;
3) Відповідь з сервера, що містить зашифрований JWT, розшифровується за допомогою особистості користувача;
4) Розшифрований JWT та введений особистий ключ зберігаються в локальному сховищі браузеру;
5) Дані користувача запитуються на сервері та встановлюються в локальному сховищі браузеру.

Лише власник особистого ключа може розшифрувати повідомлення, що було зашифроване з використанням відкритого ключа користувача, тому на сервері, під час входу в систему, створюється JWT, що містить ім’я користувача, який намагається увійти в систему, а потім шифрується з використанням відкритого ключа, що призначений для наскрізного шифрування повідомлень для певного отримувача, а тому і зберігається в базі даних як відкрита інформація користувача.

2.3.1 Головна сторінка

Головна сторінка (див. рис. 2.13) сервісу повинна містити найважливіші елементи для користувача, наприклад, поля для відправки повідомлення, список чатів, список повідомлень в чаті, налаштування, категорії чатів, пошук користувачів.

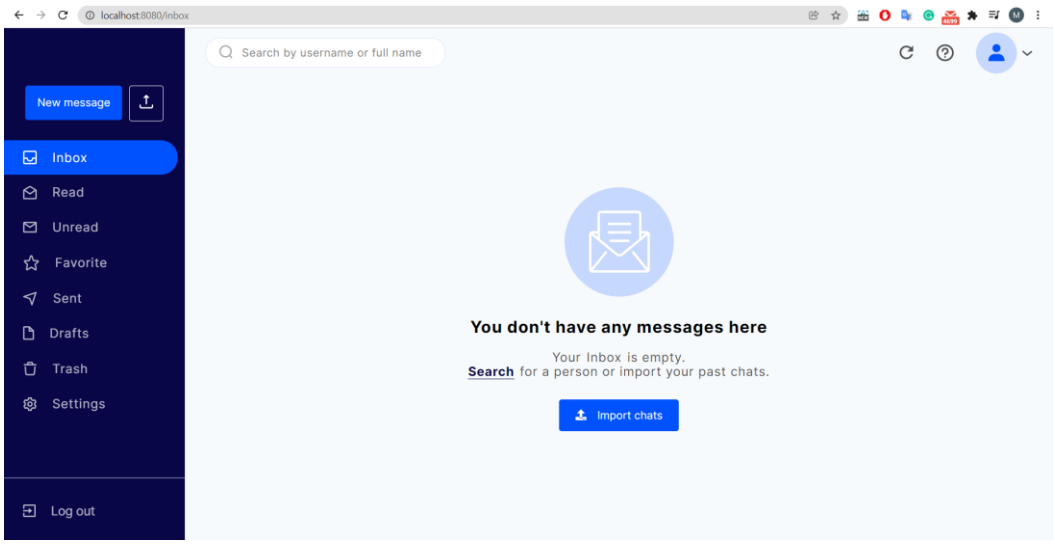

Рисунок 2.13 – Головна сторінка автентифікованого користувача

Основною функцією додатку є обмін повідомленнями, тому користувачі повинні мати змогу надіслати та отримати повідомлення. Приклад компонування нового повідомлення зображено на рис. 2.14.

Для відправлення повідомлення, користувач має ввести текст повідомлення, адресу отримувача, яка є іменем користувача зареєстрованого в сервісі (див. рис. 2.14) або електронною адресою, та за бажанням тему повідомлення і адреси отримувачів копії повідомлення (див. рис. 2.15).

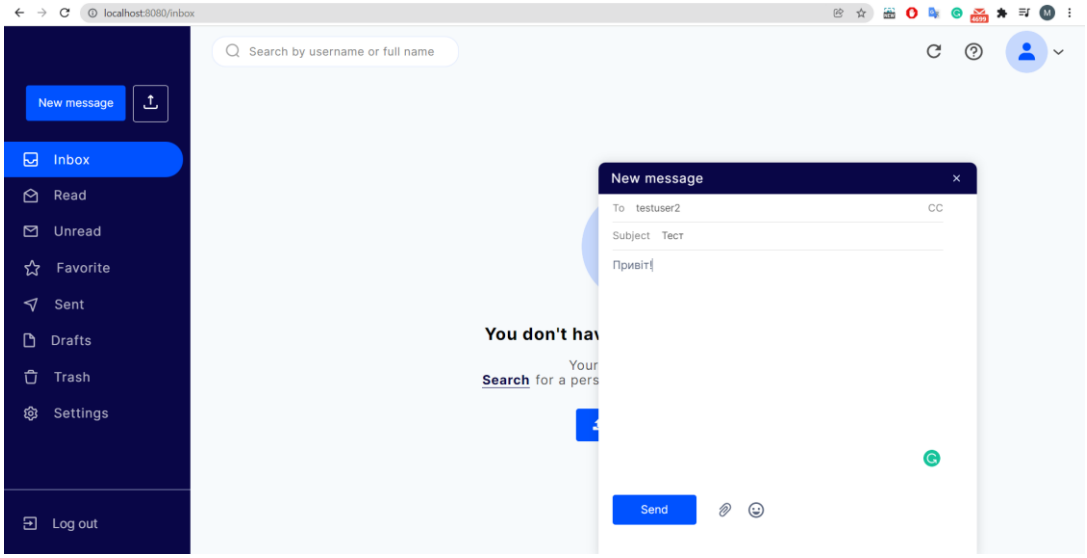

Рисунок 2.14 – Вікно відправлення нового повідомлення

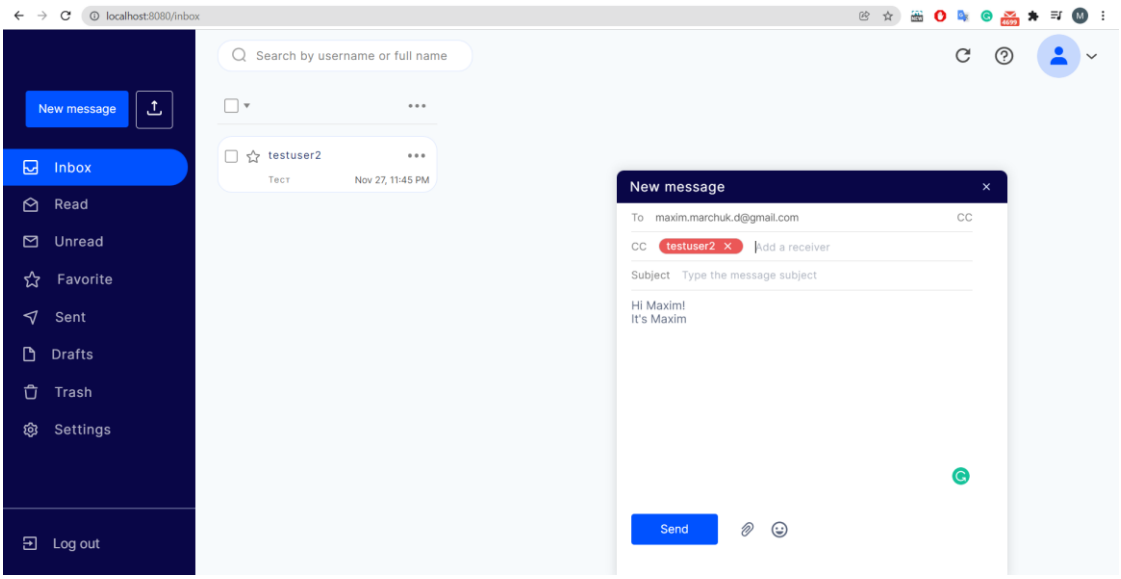

Рисунок 2.15 – Вікно відправлення нового повідомлення декільком отримувачам з різними адресами

Текст повідомлення шифрується наскрізним шифруванням для кожного отримувача окремо і кожна копія повідомлення зберігається в базі даних. Після відправлення повідомлення створюється чат з отримувач(-ем/-ами), якщо раніше не було чату з ним(-и), що має таку ж тему та список отримувачів (див. рис. 2.16).

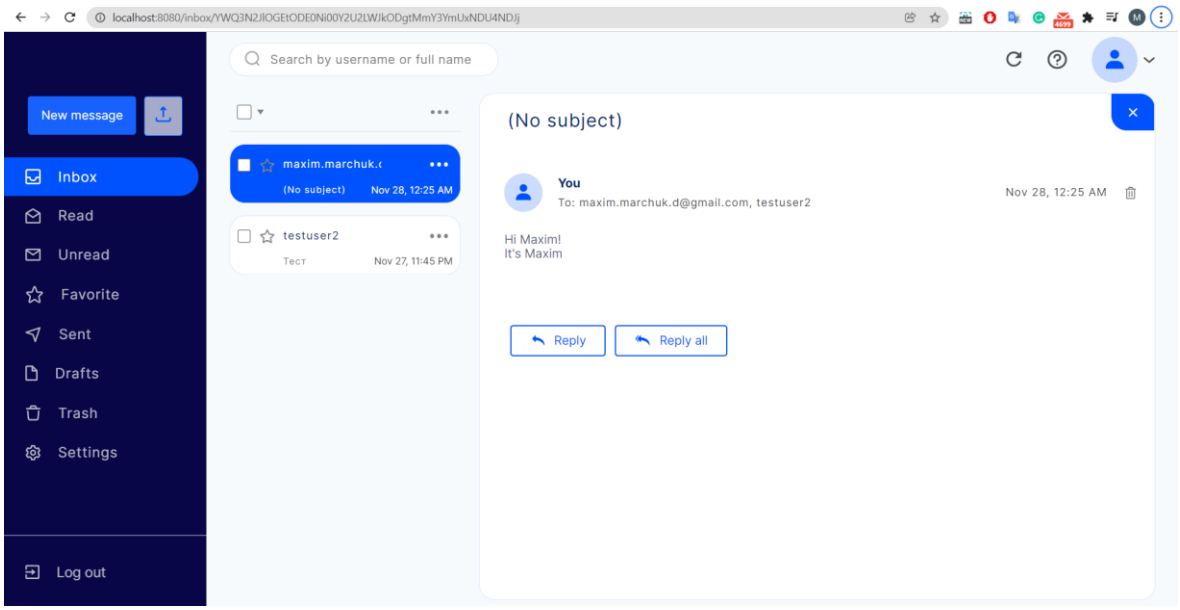

Рисунок 2.16 – Відкритий для прочитання останній чат

При отриманні нових повідомлень, отримувач може бачити кількість нових повідомлень навпроти самого чату та кількість непрочитаних чатів навпроти категорії «Unread» (див. рис. 2.17) і при відкритті чату з новими повідомленнями вони стають прочитаними, що відображається на кількості непрочитаних чатів, як зображено на рис. 2.18.

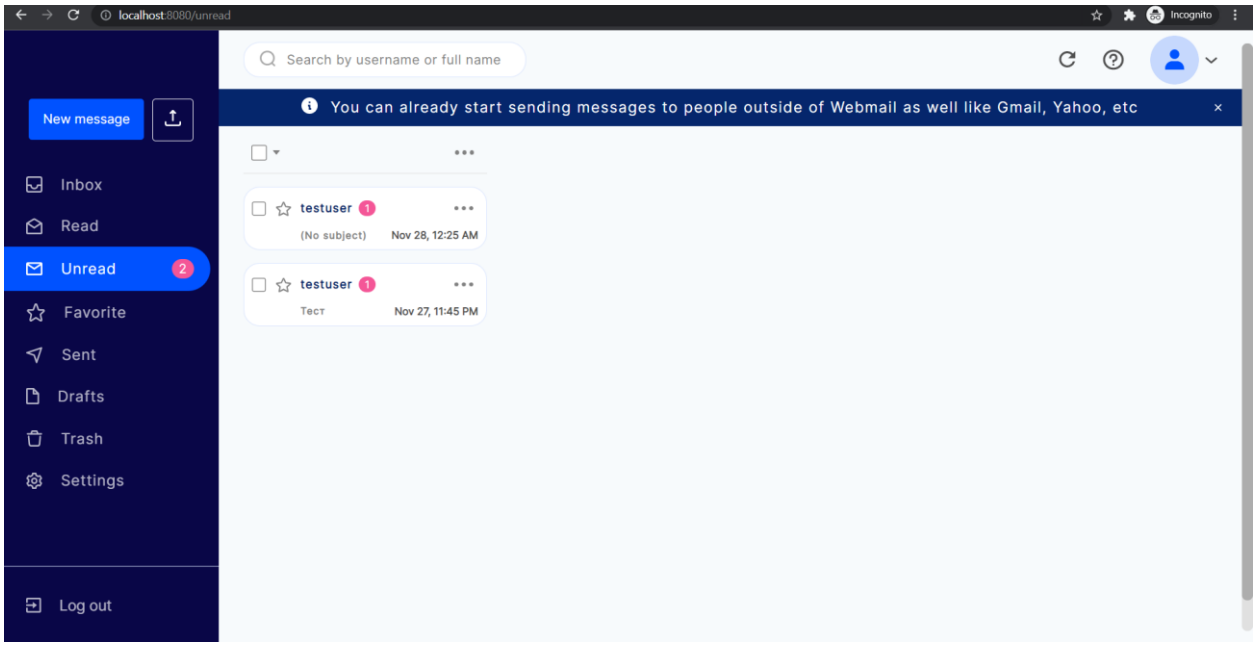

Рисунок 2.17 – Непрочитані чати в категорії «Unread»

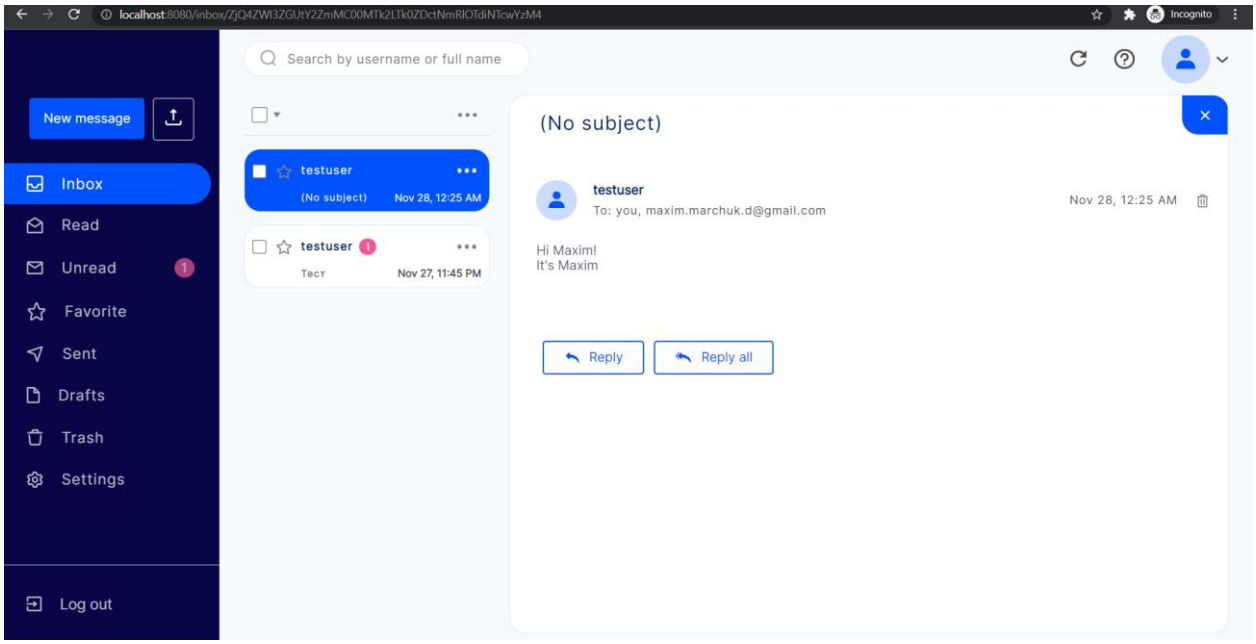

Рисунок 2.18 – Відкритий прочитаний чат

При отриманні нового повідомлення користувач може захотіти відповісти на повідомлення в чаті, що можна зробити натиснувши кнопку «Reply» для того, щоб відповісти відправнику повідомлення (див. рис. 2.19), та кнопку «Reply all», щоб відповісти та розіслати копію повідомлення всім користувачам чату (див. рис. 2.20).

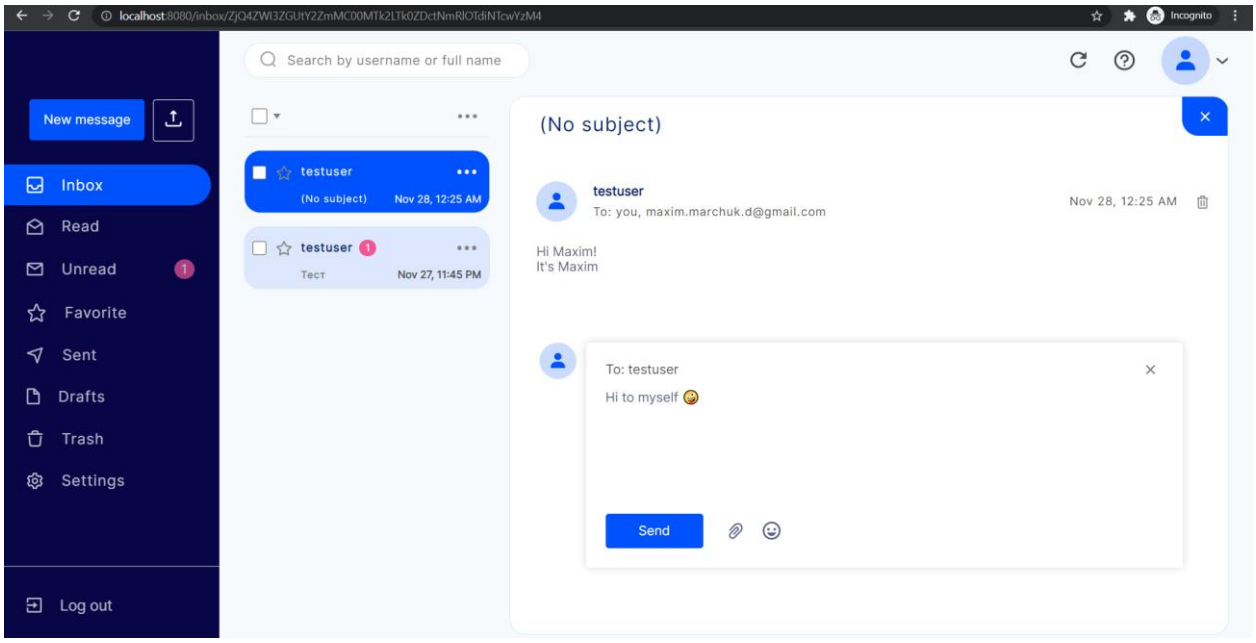

Рисунок 2.19 – Відповідь на повідомлення відправнику

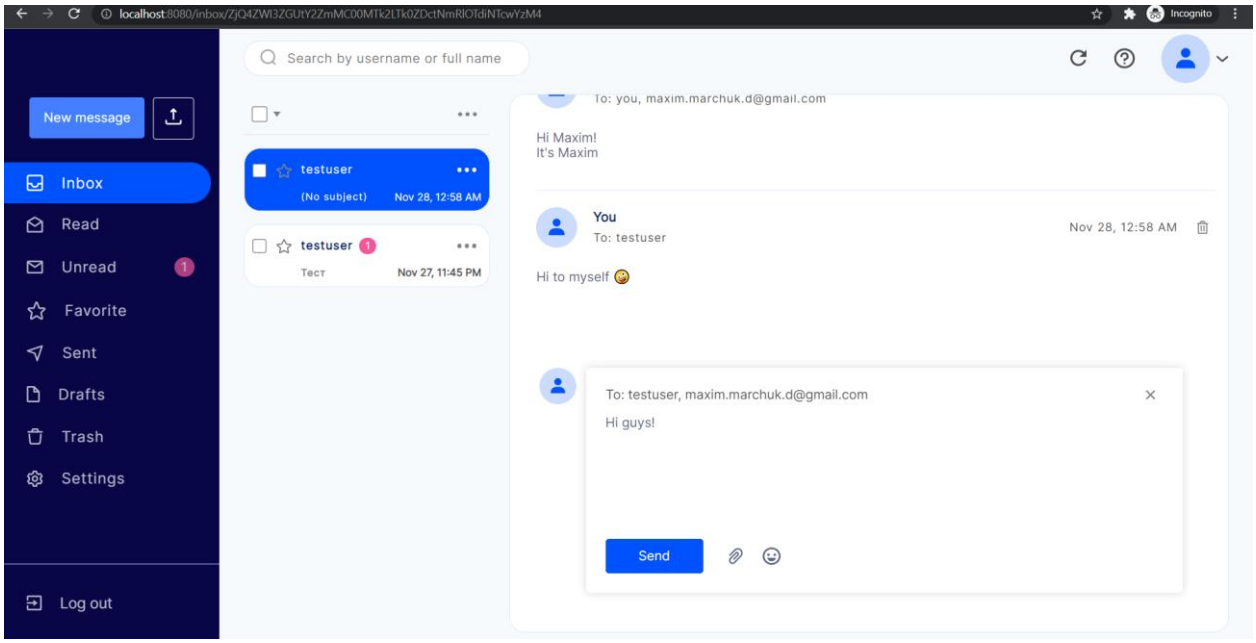

Рисунок 2.20 – Відповідь на повідомлення всім користувачам чату

Як і в ординарних webmail сервісах, важливою частиною є сортування чатів за категоріями. Тому були створені такі основні категорії:

– Inbox для зберігання всіх чатів, окрім видалених;

– Read для зберігання прочитаних чатів;

– Unread для зберігання непрочитаних чатів;

– Favorite для зберігання обраних чатів;

– Sent для зберігання чатів, що містять надіслані повідомлення поточного користувача;

– Drafts для зберігання чернеток;

– Trash для зберігання видалених чатів.

2.3.2 Сторінка пошуку користувачів

В сервісі є можливість пошуку користувачів за їх ім’ям користувача та повним ім’ям (див. рис. 2.21).

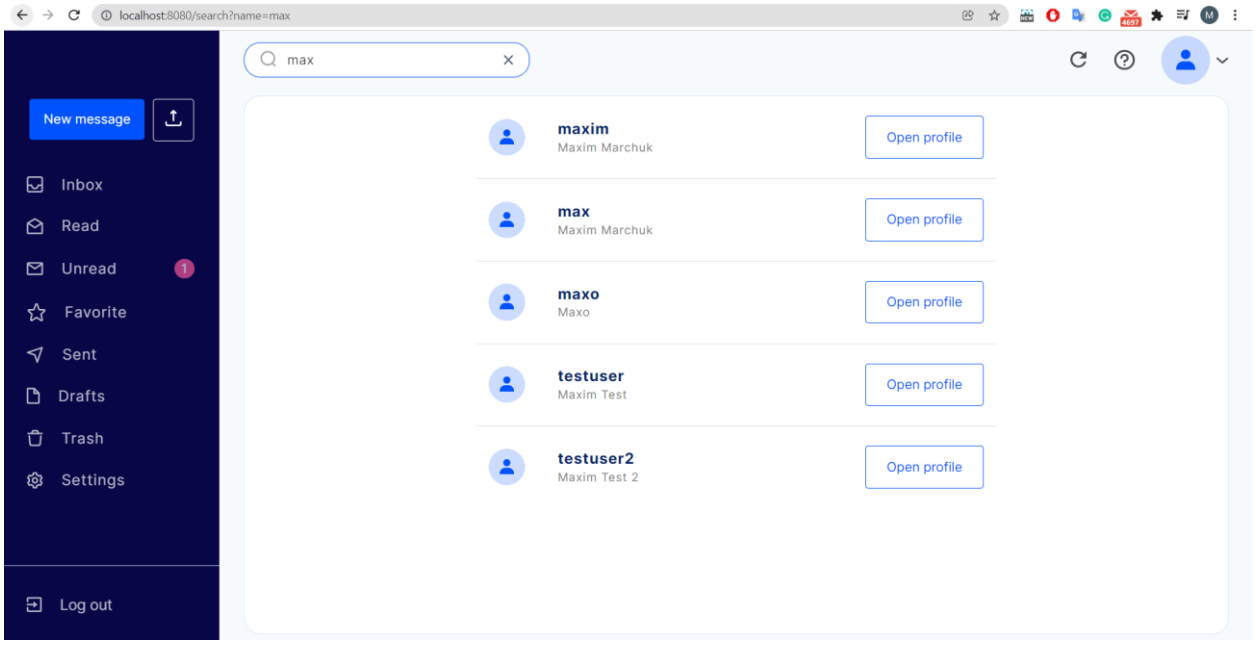

Рисунок 2.21 – Результати пошуку користувачів

При відображенні результатів пошуку можна відкрити профіль кожного з них, щоб побачити зображення, повне ім’я, біографію та зв’язатися з ним натиснувши на кнопку «Start chat» (див. рис. 2.22).

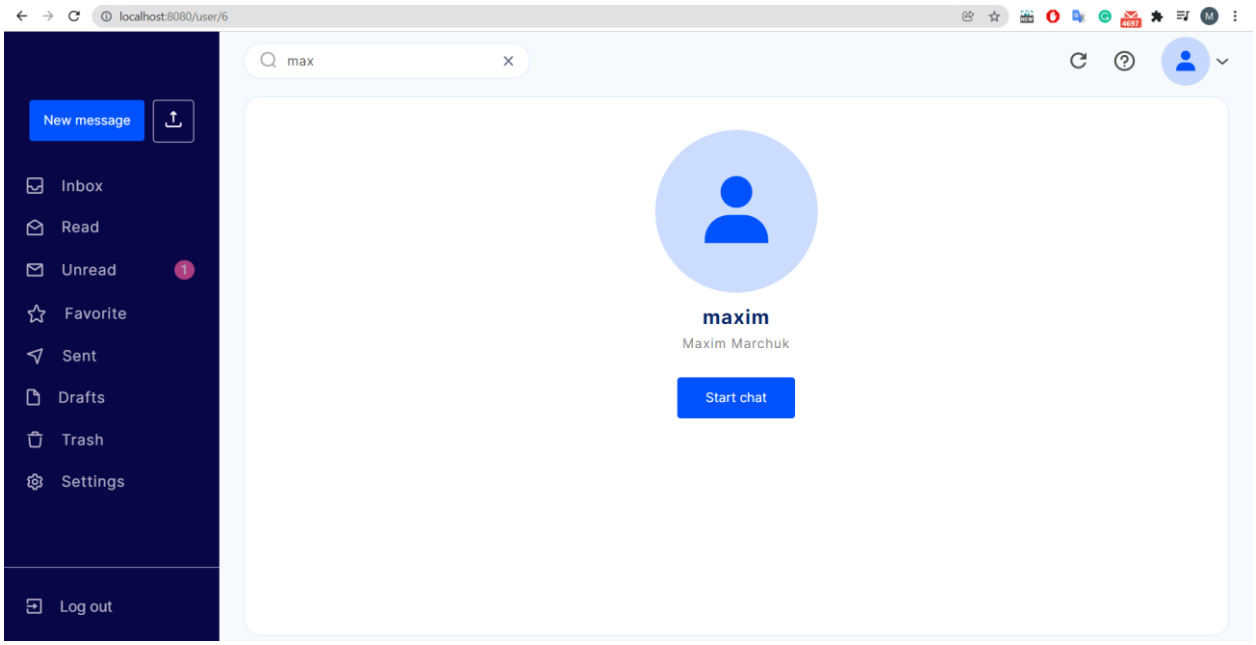

Рисунок 2.22 – Профіль знайденого користувача

2.3.3 Сторінка налаштувань

Користувачі мають змогу змінити такі дані на сторінці налаштування профілю (див. рис. 2.23):

– повне ім’я;

– біографію;

– зображення профілю.

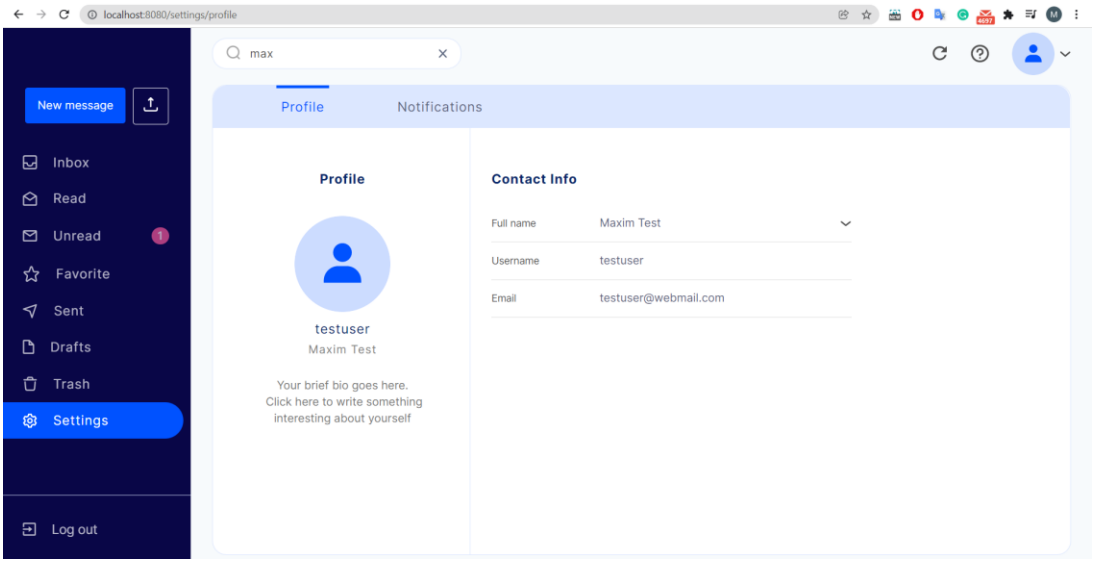

Рисунок 2.23 – Сторінка налаштування профілю

Оскільки сервіс є веб-додатком, то отримувати сповіщення про нові повідомлення користувачі не можуть, якщо веб-сторінка була закрита. Тому, щоб мати можливість зв’язатися з користувачем поза мережею було вирішено додати можливість отримувати сповіщення про нові повідомлення на зовнішню електронну пошту вказавши її на сторінці налаштування сповіщень (див. рис. 2.24).

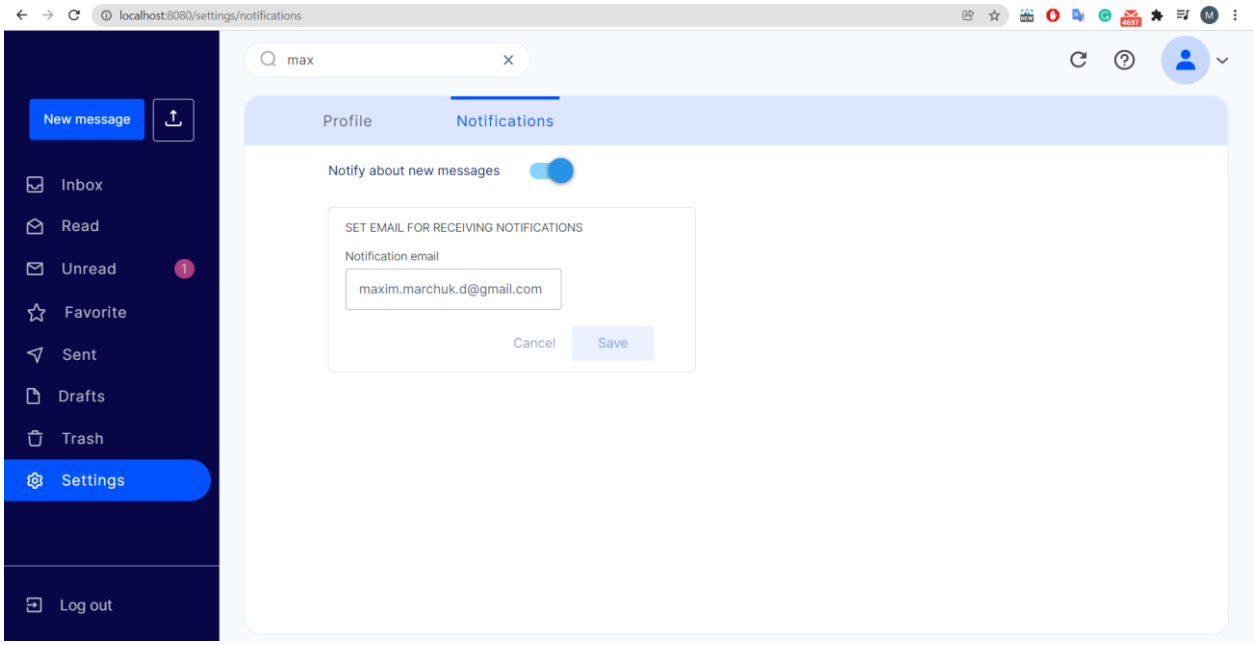

Рисунок 2.24 – Сторінка налаштування сповіщень

Якщо користувачеві було надіслано повідомлення, але він його не прочитав за 15 хвилин та, якщо в налаштуваннях вказана зовнішня електронна пошта для отримання сповіщень, то на неї він отримає сповіщення про те, що в нього є певна кількість непрочитаних повідомлень (див. рис. 2.25).

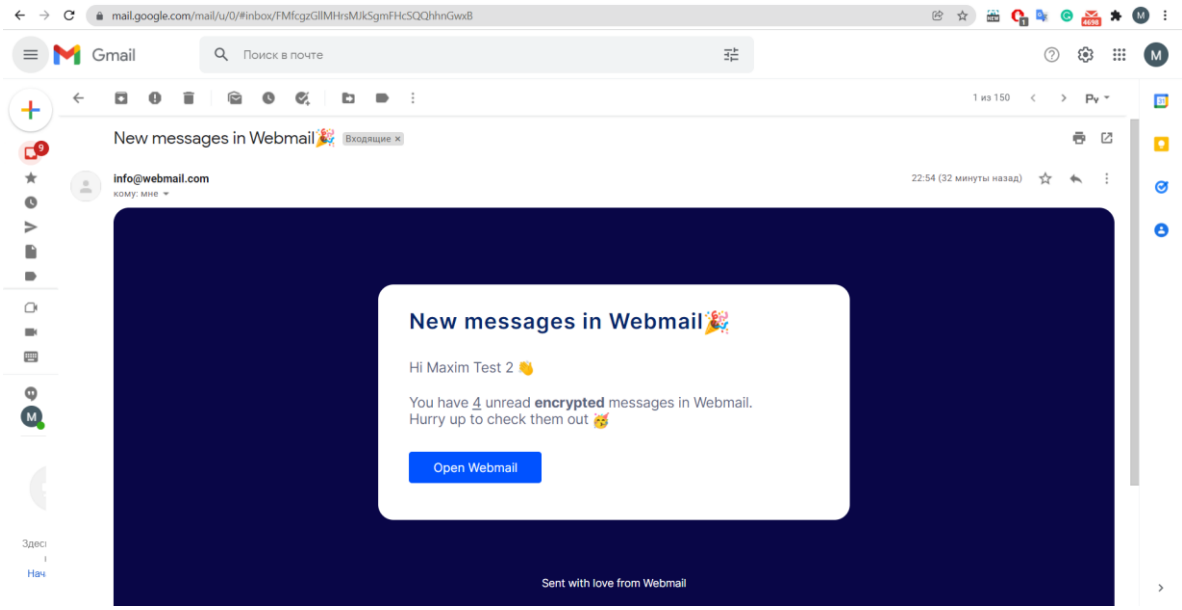

Рисунок 2.25 – Сповіщення про нові повідомлення у Webmail сервісі

3 РЕЗУЛЬТАТИ

3.1 Структура проекту

Серверна частина містить директорії, що складають таку структуру (див. рис. 3.1):

– assets для статичних елементів;

– assets/templates для шаблонів;

– dist для скомпільованих файлів;

– node_modules для зовнішніх модулів;

– src для початкових файлів;

– src/db для взаємодії з базою даних;

– src/handlers для обробників запитів;

– src/jobs для відкладених завдань;

– src/lib для утиліт;

– src/middleware для проміжних обробників;

– src/routes для маршрутів.

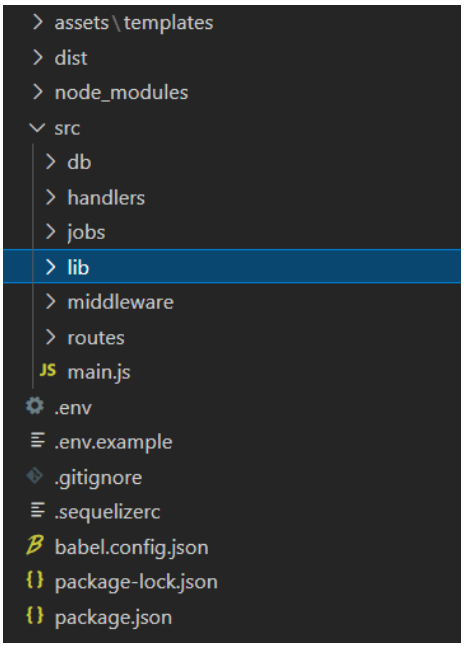

Рисунок 3.1 – Структура серверної частини сервісу

3.2 Використані модулі

Для розробки клієнтської частини були використані модулі зображені на рис. 3.2.

Рисунок 3.2 – Встановлені npm-модулі клієнтської частини

Для розробки серверної частини були використані модулі зображені на рис. 3.3.

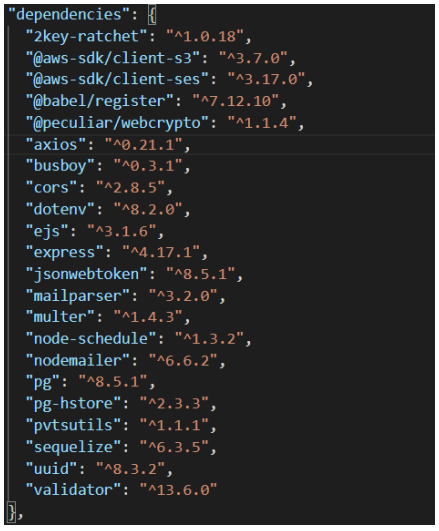

Рисунок 3.3 – Встановлені npm-модулі серверної частини

3.3 Головні сторінки

Домашня сторінка, яка є цільовою сторінкою для неавтентифікованих користувачів, зображена на рис. 3.4.

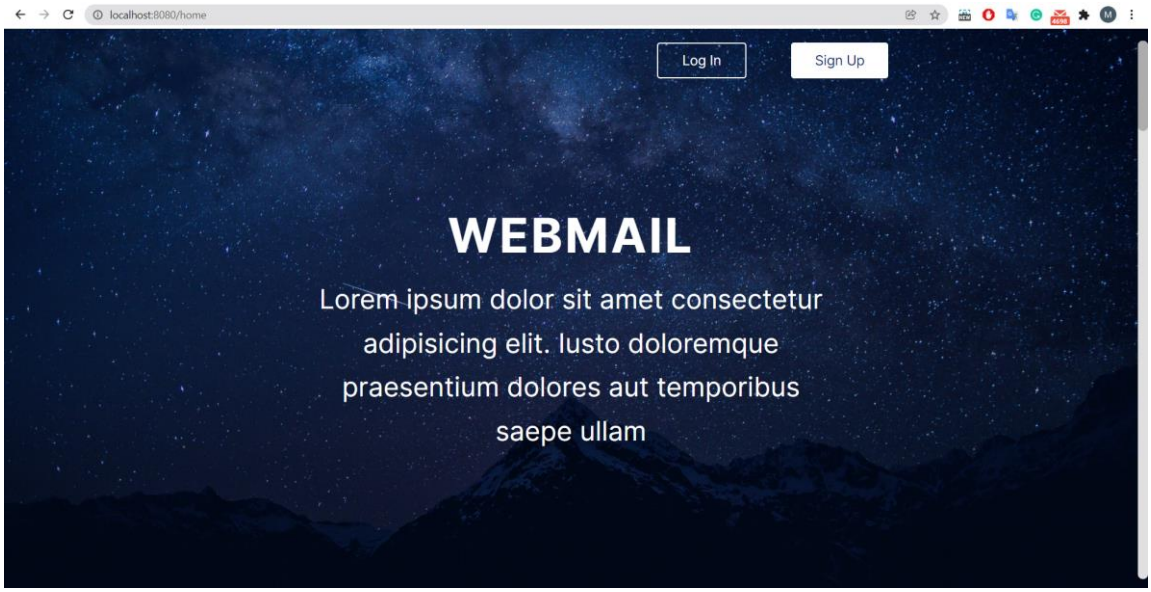

Рисунок 3.4 – Домашня сторінка

Головна сторінка, яка є цільовою сторінкою для автентифікованих користувачів, зображена на рис. 3.5.

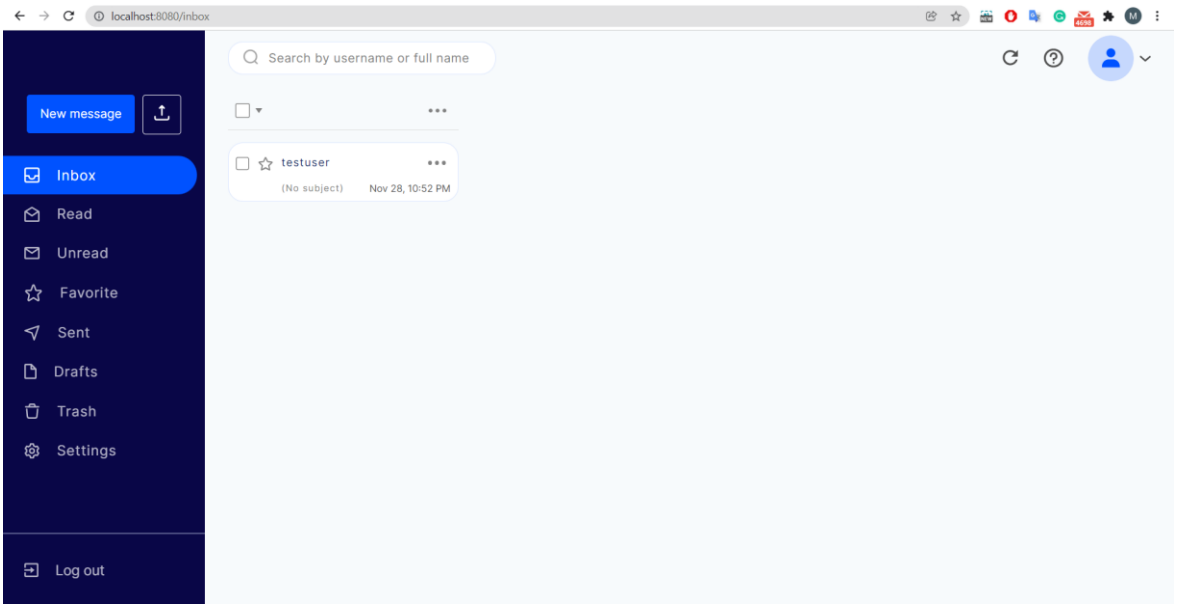

Рисунок 3.5 – Головна сторінка

ВИСНОВКИ

Метою поточної роботи була реалізація Webmail сервісу, тобто веб-клієнту електронної пошти. Були розглянуті сучасні хмарні технології для зберігання даних, обміну повідомленнями та сповіщеннями, каркаси будування веб-додатків, шаблонізатори для формування HTML-повідомлень, технології автентифікації користувачів з використанням токенів, доступні та гнучкі системи управління базою даних, технології наскрізного шифрування даних тощо.

Після розробки даного програмного забезпечення можна переконатися в тому, що сучасні технології та мови програмування дозволяють гнучко розробляти системи будь-якою складності та для будь-яких цілей. Хмарні технології зберігання даних спрощують контроль над даними, прискорюють їх отримання та розвантажують сервер, що позбавляє від необхідності зберігати їх локально. Сучасні каркаси веб-додатків суттєво спрощують будування систем і дозволяють структурувати їх найзручнішим способом.

ПЕРЕЛІК ПОСИЛАНЬ

1. Визначення електронної пошти. URL: https://en.wikipedia.org/wiki/Email (дата звернення: 16.06.2021).

2. Специфікація алгоритму Double Ratchet. URL: https://signal.org/docs/specifications/doubleratchet/ (дата звернення: 16.06.2021).

3. Специфікація протоколу X3DH. URL: https://signal.org/docs/specifications/x3dh/ (дата звернення: 16.06.2021).

4. Визначення POP3, IMAP, SMTP. URL: https://freehost.com.ua/faq/wiki/imap-pop3-smtp-chto-eto/ (дата звернення: 18.06.2021).

5. Початкова сторінка AWS SES. URL: https://aws.amazon.com/ru/ses/ (дата звернення: 20.06.2021).

6. Початкова сторінка AWS S3. URL: https://aws.amazon.com/ru/s3/ (дата звернення: 21.06.2021).

7. Початкова сторінка AWS SNS. URL: https://aws.amazon.com/ru/sns/ (дата звернення: 22.06.2021).

8. Репозиторій бібліотеки 2key-ratchet. URL: https://github.com/PeculiarVentures/2key-ratchet (дата звернення: 17.06.2021).

9. Початкова сторінка Express. URL: https://expressjs.com/ (дата звернення: 16.06.2021).

10. Визначення Cross-Origin Resource Sharing. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS (дата звернення: 16.06.2021).

11. Початкова сторінка React Router. URL: https://reactrouter.com/ (дата звернення: 17.06.2021).

ДОДАТОК А

Серверна частина

А.1 Файл для взаємодії з AWS SES

```
import { SESClient, SendRawEmailCommand } from '@aws-sdk/client-ses';
import MailComposer from 'nodemailer/lib/mail-composer';

import { composeExternalMessage } from '../compilers';


const {
  AWS_SES_REGION: region,
} = process.env;

const client = new SESClient({ region, });

export const sendEmail = async (message) => {
  try {
    const {
      sender,
      receiver,
      subject,
      text,
      html: htmlRaw,
      carbonCopy = [],
      attachments = [],
```

```
} = message;

const {
  email: senderEmail,
  fullname: senderFullname = '',
} = sender;

const senderData = {
  name: senderFullname,
  address: senderEmail,
};

const content = htmlRaw?.length > 0 ? htmlRaw : text;

const html = await composeExternalMessage(content, attachments);

const mailOptions = {
  sender: senderEmail,
  from: senderData,
  to: [
    receiver,
  ],
  cc: carbonCopy,
  subject,
  html,
};

const mail = new MailComposer(mailOptions);

const rawMessage = await new Promise((resolve, reject) => {
```

```
    mail.compile().build((error, messageBuilt) => {
      if (error) {
        return reject(error);
      }

      return resolve(messageBuilt);
    });
  });

  const params = {
    Source: senderEmail,
    Destinations: [
      receiver,
    ],
    RawMessage: {
      Data: rawMessage,
    },
  };

  const command = new SendRawEmailCommand(params);

  const response = await client.send(command);

  return response;
 } catch (error) {
  console.error('Failed to send an email', error);
 }
};
```

А.2 Файл для взаємодії з AWS S3

```
import { v4 as generateUUID } from 'uuid';
import {
  S3Client,
  DeleteObjectCommand,
  PutObjectCommand,
  GetObjectCommand,
  CopyObjectCommand,
  CreateMultipartUploadCommand,
  UploadPartCommand,
  CompleteMultipartUploadCommand,
  AbortMultipartUploadCommand,
} from '@aws-sdk/client-s3';

import { getFilenameExtension } from '../helpers.js';



const {
  AWS_S3_REGION: region,
  AWS_S3_BUCKET_ATTACHMENTS: bucketAttachments,
} = process.env;

const client = new S3Client({ region, });

const parseObjectUrl = (objectUrl) => {
  const separator = ',';

  const replacer = (match, bucketName, region, objectKey) => {
    const stringified = [bucketName, region, objectKey].join(separator);

    return stringified;
  };
```

```
    const parsedString =
objectUrl.replace(/https:\/\/(.+)\.s3\.(.+)\.amazonaws\.com\/(.+)/, replacer);

    const [bucketName, region, objectKey] = parsedString.split(separator);

    const parsedData = {
      bucketName,
      region,
      objectKey,
    };

    return parsedData;
  };

  export const deleteObjectFromBucket = async (objectKey, bucketName) => {
    try {
      const params = {
        Bucket: bucketName,
        Key: objectKey,
      };

      const command = new DeleteObjectCommand(params);

      return client.send(command);
    } catch (error) {
      throw error;
    }
  };

  export const deleteObjectFromBucketByUrl = async (objectUrl) => {
    const {
      bucketName,
      objectKey,
    } = parseObjectUrl(objectUrl);
```

```
  const response = await deleteObjectFromBucket(objectKey, bucketName);

  return response;
};

export const putObjectToBucket = async (objectKey, bucketName, body) => {
  try {
    const bucketURL = `https://${bucketName}.s3.${region}.amazonaws.com/`;

    const objectURL = `${bucketURL}${objectKey}`;

    const params = {
      Bucket: bucketName,
      Key: objectKey,
      Body: body,
    };

    const command = new PutObjectCommand(params);

    const response = await client.send(command);

    response.url = objectURL;

    return response;
  } catch (error) {
    throw error;
  }
};
```