Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)
Кафедра Інформатики
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

РОЗРОБКА ОНЛАЙН-МЕСЕНДЖЕРА З ВИКОРИСТАННЯМ БІБЛІОТЕКИ SOCKET.IO

(тема)

Виконав:
студент 4 курсу, групи ІТІНФ-18-2

Перетокін Е. Ю.
(прізвище, ініціали)

Спеціальності 122 Комп'ютерні науки
(код і повна назва спеціальності)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

Керівник ст. викл. Путятіна О.Є.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ Кобилін О.А.
(підпис) (прізвище, ініціали)

2022 р.

Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)

Кафедра Інформатики
(повна назва)

Рівень вищої освіти перший (бакалаврський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____ 2022 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Перетокіну Едуарду Юрійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка онлайн-месенджера з використання бібліотеки Socket.IO

затверджена наказом університету від 16 травня 2022 року № 541Ст

2. Термін подання студентом роботи до екзаменаційної комісії 30 травня 2022 р.

3. Вихідні дані до роботи Socket.IO, MongoDB Cluster, AWS S3, Jimp, React, Zustand, NestJS, Google Authorization Tokens.

4. Перелік питань, що потрібно опрацювати в роботі

1. Оглянути заглані методів реалізації вебзастосунка.
2. Оглянуті основі методів реалізації месенджера.
3. Розглянути практичну реалізацію розробки онлайн-месенджера.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) Актуальність проблеми, мета роботи, постановка задачі, етапи виконання роботи, результати розрахунків, перспективи роботи.

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата
Консультант з дотримання діючих стандартів та норм	Доцент Бєлова Н.В.		

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	18.04.2022	
2	Аналіз завдання, підбір літератури	18.04.22-21.04.22	
3	Аналіз літератури з досліджуваної проблеми	22.04.22-25.04.22	
4	Аналіз необхідних даних	26.04.22-30.04.22	
5	Розрахунки	01.05.22-14.05.22	
6	Аналіз результатів	15.05.22-23.05.22	
7	Оформлення пояснювальної записки	24.05.22-26.05.22	
8	Перевірка на плагіат	27.05.22	
9	Рецензування	28.05.22	
10	Підготовка презентації та доповіді	29.05.22-30.05.22	
11	Занесення роботи в електронний архів	29.05.22	
12	Попередній захист кваліфікаційної роботи	06.06.22	

Дата видачі завдання 18 квітня 2022 р.

Студент _____
(підпис)

Керівник роботи _____ ст. викл. Путятіна О.Є.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ/ABSTRACT

Пояснювальна записка до кваліфікаційної роботи: 68 с., 49 рис., 3 дод., 30 джерел.

РОЗРОБКА ОНЛАЙН МЕСЕНДЖЕРА, REACT, NESTJS, MONGO.DB, ESBUILD, RESTFULAPI, MATERIAL API V5, БІБЛІОТЕКА SOCKET.IO, ТЕСТУВАННЯ.

Об’єктом роботи є розробка сучасного онлайн менеджера з використанням бібліотеки Socket.IO.

Метою роботи є розробка методів відправки повідомлень від клієнта до клієнта у їх власній кімнаті, а також можливість надсилати фотографії.

Використано архітектурний стиль взаємодії компонентів розподіленої програми у мережі RESTFul API, документоорієнтована система управління базами даних MongoDB, бібліотека Material UI v5, також під час процесу створення месенджера для формування коду було використано такі інструменти як Eslint та Prettier.

ONLINE MESSAGING DEVELOPMENT, REACT, NESTJS, MONGO.DB, ESBUILD, RESTFULAPI, MATERIAL API V5, LIBRARY SOCKET.IO, TESTS.

The object of the work is the development of a modern online-messenger using the Socket.IO library.

The purpose of the qualification work is to develop methods of sending messages from client to client in their own room, and the ability to send media.

I used the architectural style of distributed program components in the RESTFul API network, the document-oriented MongoDB database management system, and the Material UI v5 library, and I used tools such as Eslint and Prettier during the process of creating a messenger to generate code.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

HTTP − HyperText Transfer Protocol

HTTPS – HyperText Transfer Protocol Secure

HTML – HyperText Markup Language

XML – eXtensible Markup Language

MVC – Model View Controller

URI – Uniform Resource Identifier

REST – Representational State Transfer

SOAP – Simple Object Access Protocol

API – Application Programming Interface

ORM – Object-Relational Mapping

DI – Dependency Injection

ВСТУП

Уявити сучасну людину без месенджера досить складно. Листування у застосунках дозволило не тільки увійти в нову еру спілкування, але й покращити бізнес-процеси. Адже подібні програми активно використовуються у діловому середовищі: для спілкування з клієнтами, партнерами, усередині компанії.

Месенджер – це програма (застосунок) для смартфона або персонального комп'ютера, що дозволяє миттєво обмінюватися з друзями текстовими повідомленнями, телефонними дзвінками та навіть розмовляти з використанням відео зв'язку.

Додаткові можливості програм обміну миттєвими повідомленнями (месенджерів):

– створення групових чатів для спілкування одночасно з кількома друзями;

– можливість додавати у повідомлення картинки, стікери та смайли, що дозволяють висловлювати співрозмовнику свої емоції;

– надсилання файлів: фотографій, відео, аудіо;

– запис голосових повідомлень (або набір тексту голосом), що сподобається тим, хто не любить друкувати або в певний час не може цього зробити;

– за певну плату – дзвінки з месенджера на стільниковий телефон другові, який не встановив застосунок (ця функція є не у всіх програмах).

Безсумнівно величезною перевагою месенджерів є збереження повідомлень і в будь-який момент можна знайти необхідну інформацію, це також призводить до неактуальності телефонних дзвінків.

1 ОГЛЯД МЕТОДІВ РЕАЛІЗАЦІЇ ВЕБЗАСТОСУНКУ

1.1 Основні поняття

1.1.1 REST

REST API – це спосіб взаємодії сайтів та вебзастосунків із сервером. Його також називають RESTful [1].

Термін складається з двох абревіатур, які розшифровуються в такий спосіб. API – це код, який дозволяє двом програмам обмінюватися даними з сервера. Російською його заведено називати програмним інтерфейсом програми. REST (Representational State Transfer) – це спосіб створення API за допомогою протоколу HTTP [2]. Є список методів за якими можна спілкуватися використовуючи HTTP (рис. 1.1), основні з них:

– GET – відправка запросу на отримання даних;
– POST – відправка запросу з новими даними;
– PUT – відправка запросу на зміну даних;
– DELETE – відправка запросу на видалення.

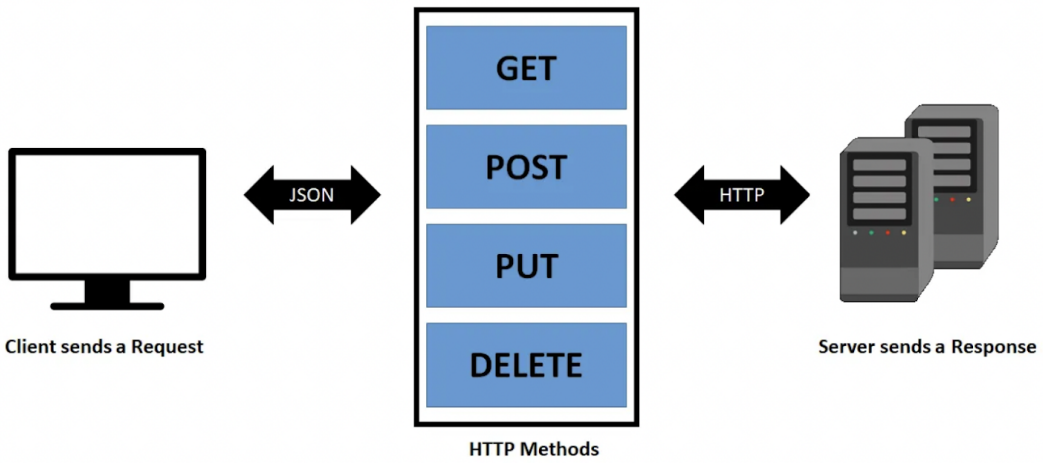

Рисунок 1.1 – Основні методи HTTP

Технологію REST API застосовують скрізь, де користувачеві сайту або вебпрограми потрібно надати дані з сервера. Наприклад, при натисканні іконки з відео на відеохостингу REST API проводить операції та запускає ролик із сервера у браузері. Зараз це найпоширеніший спосіб організації API. Він витіснив раніше популярні методи SOAP [3] і WSDL [4].

У RESTful немає єдиного стандарту роботи: його називають «архітектурним стилем» для операцій із роботи з серверів. Такий підхід у 2000 році у своїй дисертації запровадив програміст та дослідник Рой Філдінґ, один із творців протоколу HTTP [5].

У RESTful є 7 принципів написання коду інтерфейсів:

– відділення клієнта від сервера (Client-Server). Клієнт – це інтерфейс сайту або програми, наприклад, пошуковий рядок відеохостингу. У REST API код запитів залишається на стороні клієнта, а код доступу до даних – на стороні сервера. Це спрощує організацію API, дозволяє легко переносити інтерфейс користувача на іншу платформу і дає можливість краще масштабувати серверне зберігання даних;

– відсутність запису стану клієнта (Stateless). Сервер не повинен зберігати інформацію про стан (проведені операції) клієнта. Кожен запит від клієнта повинен містити тільки інформацію, яка потрібна для отримання даних від сервера;

– кешування (Cacheable). У даних запиту має бути зазначено, чи потрібно кешувати дані (зберігати у спеціальному буфері для частих запитів). Якщо така вказівка є, клієнт отримає право звертатися до цього буфера за потреби;

– єдність інтерфейсу (Uniform Interface). Усі дані повинні запитуватись через одну URL-адресу стандартними протоколами, наприклад, HTTP. Це спрощує архітектуру сайту або програми та робить взаємодію з сервером зрозумілішим;

– багаторівневість системи (Layered System). У RESTful сервері можуть розташовуватися різних рівнях, у своїй кожен сервер взаємодіє лише з найближчими рівнями і пов’язаний запитами коїться з іншими;

– надання коду на запит (Code on Demand). Сервери можуть надсилати клієнтові код (наприклад, скрипт для запуску відео). Так загальний код програми або сайту стає складнішим лише за необхідності;

– початок від нуля (Starting with the Null Style). Клієнт знає лише одну точку входу на сервер. Подальші можливості взаємодії забезпечуються сервером.

1.1.2 База даних

База даних (БД) – це організована структура, призначена для зберігання, зміни й обробки взаємопов’язаної інформації, переважно великих обсягів.

Бази даних активно використовують для динамічних сайтів зі значними обсягами даних – часто це інтернет-магазини, портали, корпоративні сайти. Такі сайти зазвичай розроблені за допомогою серверної мови програмування (наприклад, PHP) або на базі CMS (наприклад, WordPress), і не мають готових сторінок з даними за аналогією з HTML-сайтами. Сторінки динамічних сайтів формуються «на льоту» в результаті взаємодії скриптів і баз даних після відповідного запиту клієнта до вебсервера [6].

Бази даних для сайтів дають змогу зберігати інформацію, що виглядає як зв’язані між собою таблиці. Саме в БД зберігаються вся необхідна та корисна інформація для функціонування сайту (клієнтські дані, прайс-лист, список товарів).

Щоб створити запит до бази даних часто використовують Structured Query Language. SQL [7] дає змогу додавати, редагувати та видаляти інформацію, що міститься у таблицях.

Під час програмування сайтів використовують різні системи управління БД. До основних СУБД, відносять:

– об’єктно-реляційна система управління базами даних Oracle Database;

– вільна система управління базами даних PostgreSQL [8];

– система керування базами даних Microsoft SQL Сервер [9];

– вільна система управління базами даних MySQL [10].

Розподілені бази даних (РБД) – це сукупність логічно пов’язаних між собою БД, які є розподіленими у комп’ютерній мережі.

Переваги РБД:

– ця модель відображає інформацію в найбільш простій для користувача формі;

– заснована на розвиненому математичному апараті, який дозволяє досить лаконічно описати основні операції з даними;

– дозволяє створювати мови маніпулювання даними не процедурного типу;

– маніпулювання даними на рівні вихідний БД і можливість зміни.

Недоліки РБД:

– найповільніший доступ до даних;

– трудомісткість розробки.

СУБД – це програмні засоби, які виступають посередником між БД та її користувачами. Завдяки сукупності мовних та програмних засобів, СУБД сприяють створенню, ведення та спільного використання БД різними користувачами.

Сучасна програма СУБД складаються з ядра, процесору мови БД, підсистеми підтримки часу виконання та сервісних програм, які надають додаткові можливості для обслуговування інформаційних систем.

1.2 Основні технології

1.2.1 Фреймворк React

React – це бібліотека для розробки інтерфейсу користувача на основі JavaScript. Facebook та спільнота розробників з відкритим вихідним кодом керують ним. Хоча React – це бібліотека, а не мова, вона широко використовується у веброзробці.

Бібліотека вперше з'явилася в травні 2013 року і тому зараз є однією з клієнтських бібліотек, що найчастіше використовуються у веброзробці [11].

React пропонує різні розширення для всієї архітектурної підтримки застосунків, таких як Flux або React Native, крім простого інтерфейсу користувача [12].

Популярність React сьогодні перевершила популярність всіх інших фреймворків фронтенд-розробки. Ось чому:

– просте створення динамічних програм: React спрощує створення динамічних вебпрограм, тому що він вимагає меншого кодування і пропонує більше функціональності, на відміну від JavaScript, де кодування часто дуже швидко ускладнюється;

– покращена продуктивність: React використовує віртуальний DOM, тим самим швидше створюючи вебпрограми. Virtual DOM порівнює попередні стани компонентів і оновлює тільки елементи в Real DOM, які були змінені замість того, щоб знову оновлювати всі компоненти, як це роблять звичайні вебзастосунки;

– багаторазові компоненти: Компоненти є будівельними блоками будь-якої програми React, і одна програма зазвичай складається з декількох компонентів. Ці компоненти мають свою логіку та елементи управління, і їх можна повторно використовувати в усьому застосунку, що, своєю чергою, значно скорочує час розробки програми;

– однонапрямний потік даних: React слідує за односпрямованим потоком даних. Це означає, що при розробці React розробники часто вводять

дочірні компоненти в батьківські компоненти. Оскільки дані течуть в одному напрямку, стає легше налагоджувати помилки та знати, де виникає проблема у застосунку на цей час;

– його можна використовувати для розробки як вебзастосунків, так і мобільних застосунків: ми вже знаємо, що React використовується для розробки вебзастосунків, але це ще не все, що він може зробити. Існує фреймворк під назвою React Native, похідний від самого React, який є надзвичайно популярним і використовується для створення красивих мобільних застосунків. Таким чином, насправді React можна використовувати для створення вебзастосунків, так і мобільних застосунків;

– спеціальні інструменти для легкого налагодження: Facebook випустив розширення Chrome, яке можна використовувати для налагодження програм React. Це робить процес налагодження вебзастосунків React швидше та простіше;

– вищезгадані причини більш ніж виправдовують популярність бібліотеки React і те, чому вона використовується великою кількістю організацій та підприємств.

Тепер ознайомимось із функціями React. React пропонує деякі визначні функції, які роблять його найбільш поширеною бібліотекою для розробки фронтенд-застосунку [13]. Ось перелік цих основних особливостей.

JSX – це синтаксичне розширення JavaScript. Це термін, який використовується в React для опису того, як повинен виглядати інтерфейс користувача [14]. Можна написати HTML структури в тому ж файлі, що і код JavaScript [15], але його головним правилом є те, що будь-який елемент має мати одного батька. Для цього використовують JSX (або TSX, якщо використовуєте Typescript [16]) (рис. 1.2).

Рисунок 1.2 – JSX формат

Virtual DOM – це полегшена версія Real DOM від React. Реальні маніпуляції DOM значно повільніші, ніж віртуальні маніпуляції DOM. Коли стан об’єкта змінюється, віртуальний DOM оновлює цей об’єкт у реальному DOM, а не всі з них (рис. 1.3).

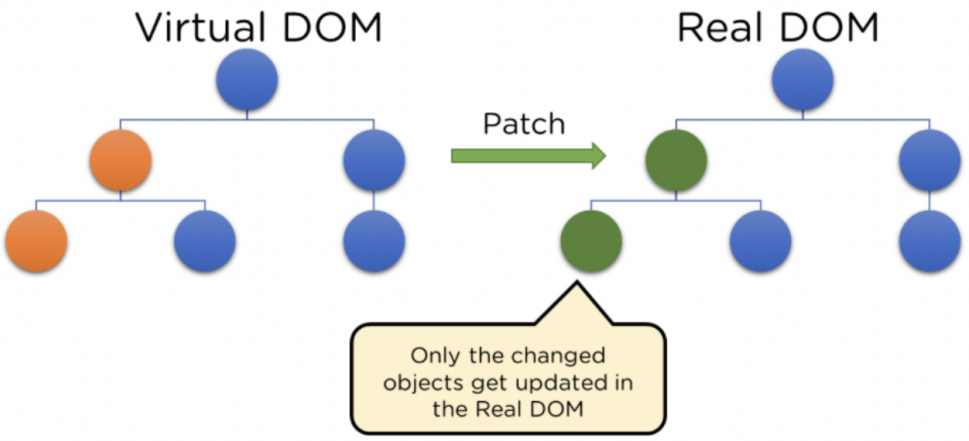

Рисунок 1.3 – Візуалізація Virtual DOM

DOM (об’єкт-модель документа) розглядає XML або HTML-документ як деревоподібну структуру, в якій кожен вузол є об’єктом, що є частиною документа (рис. 1.4).

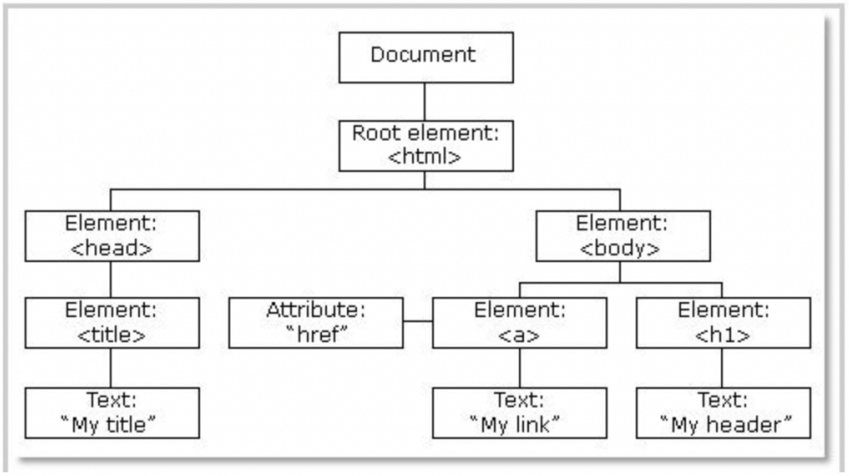

Рисунок 1.4 – DOM структура сторінки

В архітектурі Model View Controller (MVC) React є «Переглядом», відповідальним за зовнішній вигляд та відчуття програми.

MVC – це архітектурний шаблон, який розділяє прикладний шар на модель, вид та контролер. Модель належить до всієї логіки, пов'язаної з даними; уявлення використовується для логіки інтерфейсу користувача, а контролер є інтерфейсом між моделлю і уявленням [17].

React виходить за рамки простого створення фреймворку інтерфейсу користувача; він містить безліч розширень, які охоплюють всю архітектуру програми. Це допомагає створювати мобільні програми та забезпечує відображення на стороні сервера. Flux та Redux, серед іншого, можуть розширити React.

1.2.2 Бібліотека Socket.IO

Socket.IO – це бібліотека JavaScript для вебзастосунків реального часу. Він забезпечує двосторонній зв'язок у реальному часі між вебклієнтами та серверами [18]. Він складається з двох частин: клієнтської бібліотеки, яка

запускається у браузері, та серверної бібліотеки для node.js. Обидва компоненти мають ідентичний API.

Для встановлення з'єднання та обміну даними між клієнтом та сервером Socket.IO використовує Engine.IO. Це реалізація нижчого рівня, що використовується під капотом. Engine.IO використовується для реалізації сервера, а Engine.IO-клієнт використовується для клієнта (рис. 1.5).

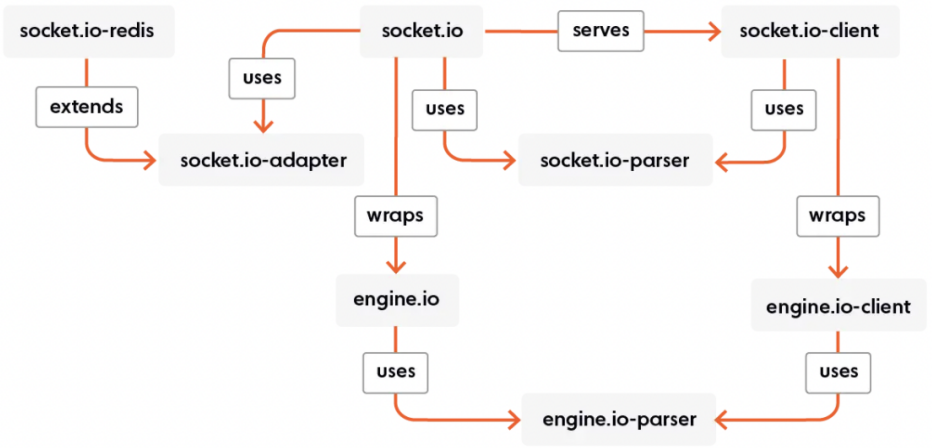

Рисунок 1.5 – UML діаграма Engine.IO

Популярним способом продемонструвати двосторонній зв'язок, що надається Socket.IO, є базова програма для чату (ми поговоримо про деякі інші сценарії використання нижче). За допомогою сокетів, коли сервер отримає нове повідомлення, він надішле його клієнту та повідомить його, оминаючи необхідність надсилання постійних запитів між клієнтом та сервером. Саме тому використання сокетів є найбільш ефективним способом взаємозв'язку між клієнт-сервером в цьомувипадку (рис. 1.6).

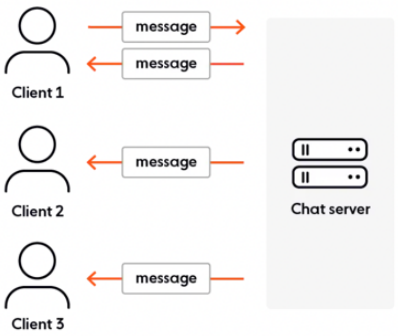

Рисунок 1.6 – Зв’язок між клієнтом та сервером

Socket.IO надає деякі додаткові функції у порівнянні з підключенням Engine.IO:

– автоматичне перепідключення;

– буферизація пакетів;

– подяки;

– трансляція всім клієнтам або підгрупі клієнтів (що ми називаємо «Кімната»);

– мультиплексування (що ми називаємо «простір імен»).

Написання програми для реального часу з використанням популярних стеків вебзастосунків, таких як LAMP (PHP), традиційно було дуже важким. Він включає опитування сервера на наявність змін, відстеження тимчасових міток, і це набагато повільніше, ніж має бути.

Сокети традиційно були рішенням, навколо якого будується більшість систем реального часу, забезпечуючи двонапрямний канал зв’язку між клієнтом та сервером. Це означає, що сервер може надсилати повідомлення клієнтам. Щоразу, коли відбувається подія, ідея полягає в тому, що сервер отримає його і відправить зацікавленим підключеним клієнтам.

Socket.IO досить популярний, його використовують Microsoft Office, Yammer, Zendesk, Trello та багато інших організацій для створення надійних систем реального часу. Це одна з найпотужніших JavaScript-фреймворків на

GitHub та найбільш залежна від модуля NPM (Node Package Manager). Socket.IO також має величезну спільноту, що означає, що знайти допомогу досить легко.

1.2.3 Material UI

Material UI – це інтерфейсний фреймворк з відкритим вихідним кодом для компонентів React, який має понад 60 500 зірочок на github. Він побудований за допомогою Less. Less (означає Leaner Style Sheets) – це зворотно-сумісне розширення мови для CSS [19]. Material UI заснований на Material Design від Google, щоб забезпечити високоякісний цифровий досвід під час розробки інтерфейсної графіки. Material Design зосереджується на забезпеченні сміливого та чіткого дизайну – він створює текстури, зосереджуючись на тому, як саме ці компоненти відкидають тіні та відбивають світло [20].

Існує кілька ключових переваг проєктування за допомогою Material UI:

– деякі інтерфейсні фреймворки не дуже добре задокументовані, тому їх важко розробляти. Проте Material UI має детальну документацію, яка полегшує навігацію по фреймворку;

– інтерфейс матеріалу залишається останнім з регулярними оновленнями;

– компоненти узгоджені за дизайном та кольоровими тонами, що дозволяє розробленому застосунку/вебсторінки виглядати естетично.

1.2.4 Node.js

Node.js – це «упакована компіляція движка JavaScript V8 від Google, рівня абстракції платформи libuv та основної бібліотеки, яка сама переважно написана на JavaScript» [21].

Середовище виконання використовує Chrome V8, який є механізмом виконання JavaScript. Це додає додаткові варіанти використання до репертуару Node.js, наприклад доступ до внутрішньої функціональності системи (наприклад, мережа).

Node.js використовує архітектуру «Single Threaded Event Loop» для одночасної обробки кількох клієнтів. Щоб зрозуміти, чим це відрізняється від інших середовищ виконання, нам потрібно зрозуміти, як багатопотокові одночасні клієнти обробляються в таких мовах, як Java.

У багатопотоковій моделі запит-відповідь кілька клієнтів надсилають запит, і сервер обробляє кожен з них, перш ніж надіслати відповідь назад. Однак для обробки одночасних викликів використовуються кілька потоків. Ці потоки визначаються в пулі потоків, і щоразу, коли надходить запит, для його обробки призначається окремий потік.

Node.js працює інакше. Можно розглянути кожен крок, який він проходить:

– Node.js підтримує обмежений пул потоків для обслуговування запитів;

– щоразу, коли надходить запит, Node.js поміщає його в чергу;

– тепер з’являється однопотоковий «цикл подій» – основний компонент. Цей цикл подій чекає на запити необмежено довго;

– коли надходить запит, цикл забирає його з черги та перевіряє, чи потрібна для нього операція блокування введення/виводу (I/O). Якщо ні, він обробляє запит і надсилає відповідь;

– якщо запит має виконати операцію блокування, цикл подій призначає потік із внутрішнього пулу потоків для обробки запиту. Доступні

обмежені внутрішні потоки. Ця група допоміжних потоків називається робочою групою;

– цикл подій відстежує запити блокування та розміщує їх у черзі після обробки завдання блокування. Таким чином він зберігає свою не блокуючу природу.

Архітектура Node.js приведена на рисунку 1.7

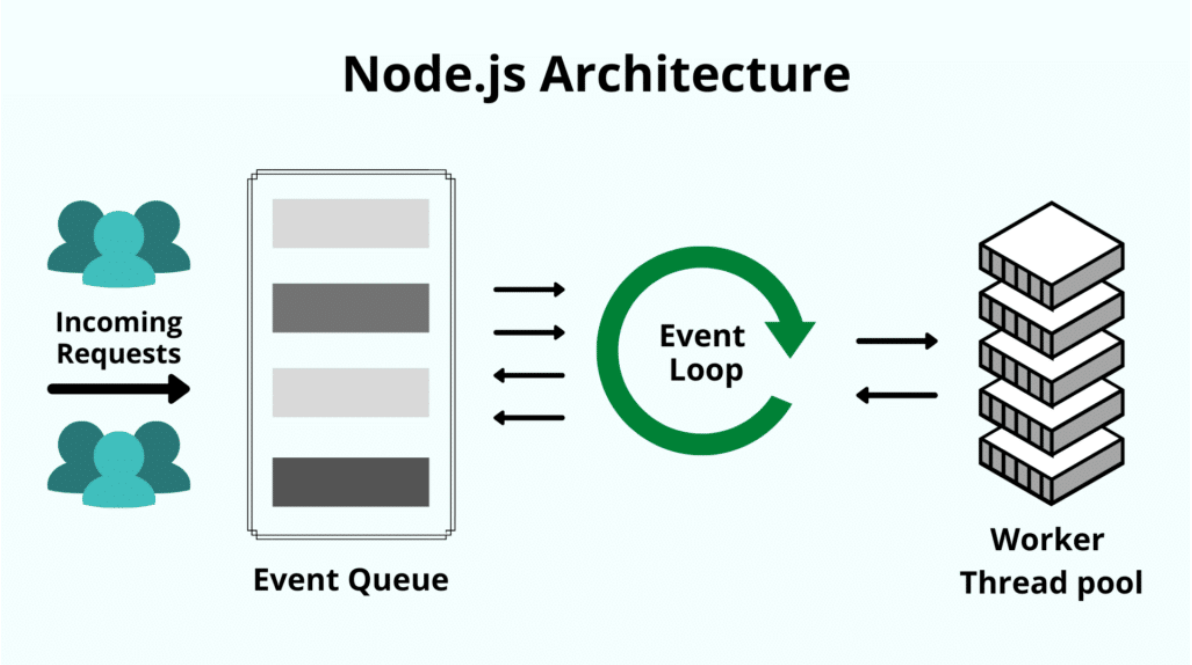

Рисунок 1.7 – архітектура Node.js

Node.js використовується для широкого спектра застосунків (рис. 1.8). Розгляньмо деякі популярні випадки використання, коли Node.js є хорошим вибором:

– чати в режимі реального часу – завдяки однопотоковій асинхронній природі Node.js добре підходить для обробки спілкування в реальному часі;

– інтернет речей – програми IoT зазвичай містять кілька датчиків, оскільки вони часто надсилають невеликі шматки даних, які можуть накопичуватися у великій кількості запитів. Node.js є хорошим вибором, оскільки він здатний швидко обробляти ці одночасні запити;

– передача даних – такі компанії, як Netflix, використовують Node.js для цілей потокової передачі. В основному це пов’язано з тим, що Node.js є легким і швидким, крім того, Node.js надає вбудований API потокової передачі;

– комплексні односторінкові програми (SPA) – у SPA весь застосунок завантажується на одній сторінці. Зазвичай це означає, що у фоновому режимі виконується кілька запитів для певних компонентів. Тут на допомогу приходить цикл подій у Node.js, оскільки він обробляє запити не блокуючим способом;

– програми на основі REST API – JavaScript використовується як у інтерфейсі, так і в серверній частині сайтів. Таким чином, сервер може легко спілкуватися з інтерфейсом через REST API за допомогою Node.js. Node.js також надає такі пакети, як Express.js і Koa, які спрощують створення вебзастосунків.

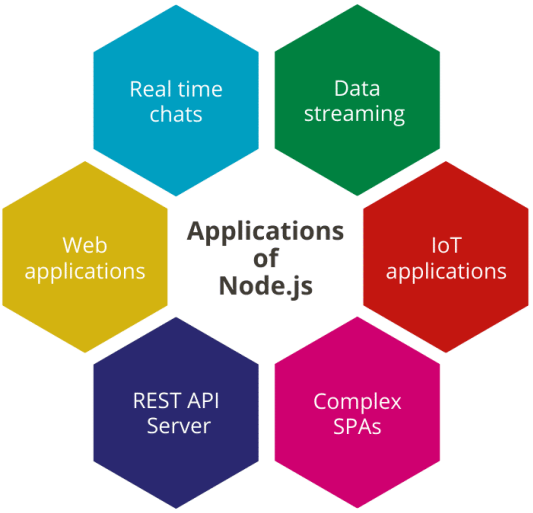

Рисунок 1.8 – Можливі вебзастосунки на Node.js

1.3 Постановка задачі

Таким чином, використання сокетів та графічного фреймворку є актуальними технологіями для розробки месенджера, бо це дозволяє зробити його інтерактивним та зручним для користування.

Об’єктом роботи є створення онлайн менеджера з використанням бібліотеки Socket.IO.

Метою роботи є розробка методів посилання повідомлень від клієнта до клієнта у їх власній кімнаті, а також можливість надсилати фотографії.

Для досягнення мети необхідно вирішити такі завдання:

– налагодити зв’язок між сервером та клієнтом за допомогою сокетів;

– розробити алгоритм надсилання повідомлень різних форматів;

– реалізувати алгоритм створення повідомлень та чатів з можливість їх зберігання у базі даних;

– розробити алгоритм створення попереднього перегляду зображення;

– реалізувати веб застосунок, що буде використовувати усі описані методи із сучасним дизайном та швидким відкликом застосунку.

2 ОГЛЯД ОСНОВНИХ МЕТОДІВ РЕАЛІЗАЦІЇ ВЕБЗАСТОСУНКУ

2.1 Надсилання повідомлення за допомогою Socket.IO

Надсилання повідомлення конкретному користувачеві за допомогою Socket.IO є дуже важливою функцією, якщо ви створюєте приватний застосунок для чату.

Тут відстежуються усі користувачі у масиві object. Коли користувачі підключаються до сервера Node.js, сервер надсилає клієнту масив об’єктів у вигляді списку онлайн-користувачів (рис. 2.1). Цей об’єкт складається з унікального socket.id, а також імені користувача, яке надається клієнтом, коли клієнт підключається до сервера [22].

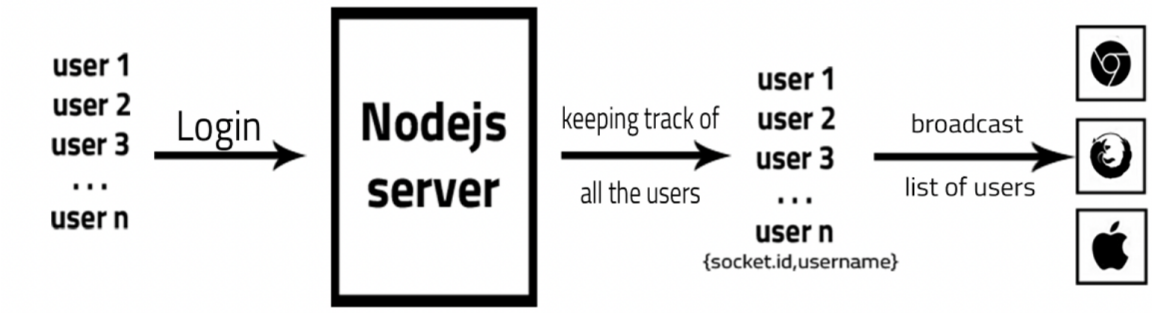

Рисунок 2.1 – Підключення юзера на NodeJS сервер

Щоб надіслати повідомлення конкретному клієнту, треба надати серверу socket.id цього клієнта, а на стороні сервера Socket.IO подбає про надіслання цього повідомлення за допомогою:

socket.broadcast.to(’ID ’).emit(’send msg ’, {somedata : somedata_server});

Наприклад, користувач 3 хоче надіслати повідомлення користувачеві 1. Припустимо, що користувач 3 зв’язується з браузера Chrome, а користувач 1 підключений до браузера Firefox, як показано на рисунку 2.2.

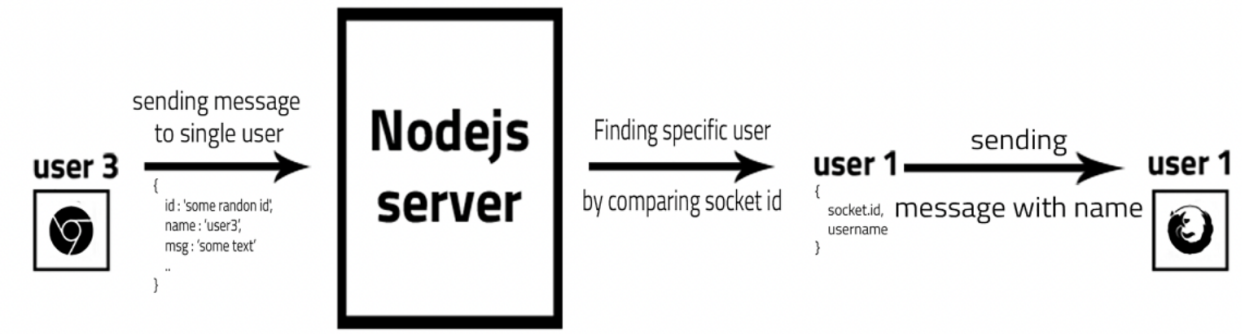

Рисунок 2.2 – Відправка повідомлення певному користувачу

2.2 Зберігання даних в базі даних використовуючи Node.js

Node.js – це середовище виконання JavaScript з відкритим вихідним кодом, яке дозволяє розробникам серверної частини використовувати JavaScript для створення серверних програм і API.

Використання Node.js дозволяє розробникам інтерфейсу (наприклад, React, Vue.js, навіть jQuery) використовувати ту ж мову програмування, JavaScript, що й розробники серверної частини. Це дає змогу розвивати більш багатофункціональну та гнучкість. Використання Node.js широко поширене та постійне, і я настійно рекомендую його для нових розробок.

Створюючи базу даних, можна налаштувати її локально або в хмарі. Наприклад, можна розгорнути інсталяцію MongoDB на локальному хості вручну, завантаживши та встановивши MongoDB. Проте, ручне встановлення супроводжується поточними витратами на технічне обслуговування та утримання. Краще уникати ручної установки MongoDB, щоб зменшити ризик помилок.

Для хмарних пропозицій існує MongoDB Atlas, база даних як послуга від MongoDB. Він не залежить від платформи, що дозволяє створювати кластери баз даних на AWS, Azure та Google Cloud, а масштабувати його так само просто, як натиснути кнопку. Вони також полегшили початок роботи,

отримавши безкоштовний рівень M0. Загальний час від реєстрації до створення бази даних для мене становив менше ніж п’ять хвилин [23].

MongoDB – це документно-орієнтована база даних NoSQL, яка використовується для зберігання даних великого обсягу. Замість використання таблиць і рядків, як у традиційних реляційних базах даних, MongoDB використовує колекції та документи. Документи складаються з пар ключ-значення, які є основною одиницею даних у MongoDB. Колекції містять набори документів і функцій, що є еквівалентом таблиць реляційної бази даних. MongoDB – це база даних, яка з’явилася приблизно в середині 2000-х років.

Наведений приклад на рисунку 2.3 показує, як документ можна змоделювати в MongoDB.

Поле _id додається MongoDB, щоб однозначно ідентифікувати документ у колекції.

Можна відзначити, що дані замовлення (OrderID, Product і Quantity), які в RDBMS зазвичай зберігаються в окремій таблиці, тоді як у MongoDB вони фактично зберігаються як вбудований документ у саму колекцію. Це одна з ключових відмінностей у тому, як дані моделюються в MongoDB.

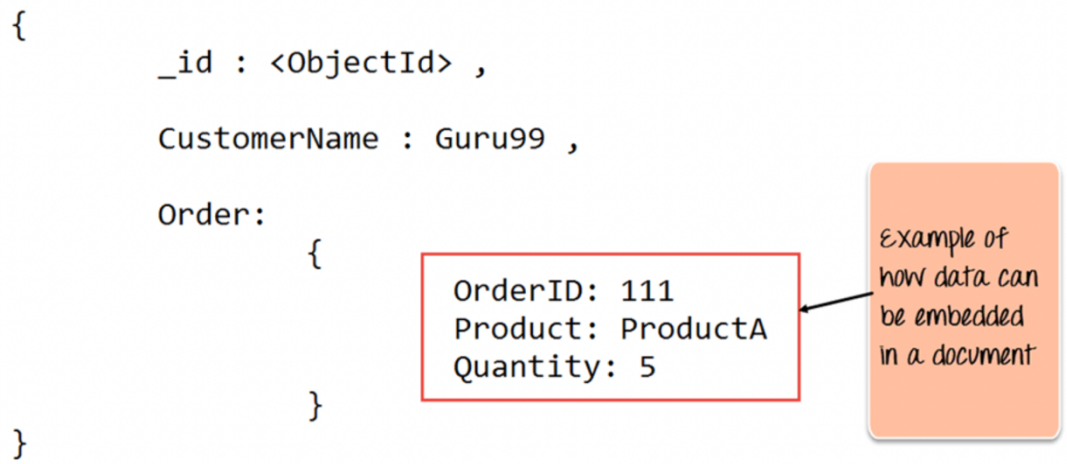

Рисунок 2.3 – Моделювання документа в MongoDB

2.3 Надсилання файлів до сервера

При надсиланні зображення використовується той же підхід, що й для надсилання текстових повідомлень.

2.3.1 Буфер зображення JS/Blob

У Chrome є доступ до зображення за допомогою clipboardData (це можливо тільки в обробці події paste). Усередині обробника можно пройти по всьому вмісту буфера та отримати зображення. Потім створивши його blob образ і використовуючи урл для зображення, можно додати його сторінку.

Firefox не має об’єкта clipboardData і для вставки зображення використовується можливість вставляти зображення в елемент з атрибутом contenteditable. Хак SetTimeout (checkInput, 1) використовується для отримання зображення після його вставки.

Blob можна легко використовувати як URL-адресу для тегів <a>, <img> або інших, щоб показати його вміст.

Завдяки типу можна завантажувати/вивантажувати об’єкти Blob, і в мережевих запитах тип, природно, стає Content-Type (рис. 2.4).

Також можна динамічно створити посилання на JavaScript і змоделювати клацання link.click(), після чого завантаження почнеться автоматично.

```
let link = document.createElement('a');
link.download = 'hello.txt';

let blob = new Blob(['Hello, world!'], {type: 'text/plain'});

link.href = URL.createObjectURL(blob);

link.click();

URL.revokeObjectURL(link.href);
```

Рисунок 2.4 – Приклад кода, який змушує користувача завантажувати динамічно створений файл Blob без HTML

URL.createObjectURL бере Blob і створює для нього унікальну URL-адресу у формі blob:<origin>/<uuid>.

Для кожної URL-адреси, згенерованої URL.createObjectURL браузером, зберігається URL-адреса → Blob зіставлення всередині. Таким чином, такі URL короткі, але дозволяють отримати доступ до Blob.

Згенерована URL-адреса (і, отже, посилання з нею) дійсна тільки в поточному документі, поки вона відкрита. І це дозволяє посилатися на Blobin <img>, <a>практично на будь-який інший об’єкт, який очікує на URL-адресу.

Хоча є побічний ефект. Поки є зіставлення для Blob, воно знаходиться в пам’яті. Браузер не може його звільнити.

Порівняння автоматично очищається при вивантаженні документа, тому Blob-об’єкти звільняються. Але якщо програма живе довго, то це відбудеться не скоро.

Тому, якщо створювати URL-адресу, вона буде висіти в пам’яті, навіть якщо вона більше не потрібна.

URL.revokeObjectURL (url) видаляє посилання з внутрішнього зіставлення, що дозволяє Blob видалити (якщо немає інших посилань) і звільнити пам’ять.

В останньому прикладі припускається, що Blob використовували лише один раз, для миттєвого завантаження, тому викликається URL.revokeObjectURL(link.href) негайно.

У попередньому прикладі з інтерактивним HTML-посиланням не було викликано URL.revokeObjectURL(link.href), тому що це зробило б BlobURL-адресу недійсною. Після відкликання, оскільки зіставлення видалено, URL-адреса більше не працює.

Можна створити Blob зображення, частину зображення або навіть зробити скриншот сторінки (рис. 2.5). Це зручно, щоб завантажити його кудись.

Операції із зображенням виконуються через елемент <canvas>:

– намалювати зображення (або його частину) на полотні за допомогою canvas.drawImage;

– викликати метод canvas .toBlob(зворотний виклик, формат, якість), який створює BLOB-об’єкт і запускає з ним зворотний виклик після завершення.

```
// take any image
let img = document.querySelector('img');

// make <canvas> of the same size
let canvas = document.createElement('canvas');
canvas.width = img.clientWidth;
canvas.height = img.clientHeight;

let context = canvas.getContext('2d');

// copy image to it (this method allows to cut image)
context.drawImage(img, 0, 0);
// we can context.rotate(), and do many other things on canvas

// toBlob is async operation, callback is called when done
canvas.toBlob(function(blob) {
  // blob ready, download it
  let link = document.createElement('a');
  link.download = 'example.png';

  link.href = URL.createObjectURL(blob);
  link.click();

  // delete the internal blob reference, to let the browser clear memory from
  URL.revokeObjectURL(link.href);
}, 'image/png');
```

Рисунок 2.5 – Приклад коду для створення посилання

2.3.2 Amazon S3

Amazon Simple Storage Service (Amazon S3) – це служба зберігання об’єктів, яка пропонує провідну в галузі масштабованість, доступність даних, безпеку та продуктивність. Клієнти будь-яких розмірів і галузей можуть використовувати Amazon S3 для зберігання та захисту будь-якої кількості даних для різних випадків використання, таких як озера даних, вебсайти, мобільні затсосунки, резервне копіювання та відновлення, архівування, корпоративні програми, пристрої Інтернету речей та великі дані. аналітика. Amazon S3 надає функції керування, щоб можна було оптимізувати, організувати та налаштувати доступ до своїх даних відповідно до конкретних вимог бізнесу, організації та відповідності. Її діаграма роботи наведена у рисунку 2.6.

Рисунок 2.6 – Діаграма роботи Amazon S3

Amazon S3 має функції керування сховищем, за допомогою яких можна керувати витратами, відповідати нормативним вимогам, зменшувати затримки та зберігати кілька окремих копій даних для відповідності вимогам:

– життєвий цикл S3 – налаштовування політикі життєвого циклу, щоб керувати своїми об’єктами та ефективно зберігати їх протягом усього

життєвого циклу. Можна переносити об’єкти в інші класи зберігання S3 або об’єкти, термін дії яких закінчився;

– блокування об’єктів S3 – заопбігання видаленню або перезапису об’єктів Amazon S3 протягом фіксованого періоду часу або на невизначений термін. Можна використовувати Object Lock, щоб задовольнити нормативні вимоги, які вимагають сховища з можливістю запису один раз-читання-багато (WORM) або просто додати ще один рівень захисту від змін і видалення об’єктів;

– реплікація S3 – повторювання об’єктів та їх відповідних метаданих та тегі об’єктів в один або кілька сегментів призначення в тих самих або різних регіонах AWS, зменшує затримки;

– пакетні операції S3 – керують мільярдами об’єктів у масштабі за допомогою одного запиту S3 API або кількох кліків на консолі Amazon S3. Можна використовувати пакетні операції для виконання таких операцій, як копіювання, виклик функції AWS Lambda та відновлення для мільйонів або мільярдів об’єктів.

Amazon S3 забезпечує надійну узгодженість читання після запису для запитів PUT та DELETE об’єктів у сегменті Amazon S3 у всіх регіонах AWS. Ця поведінка застосовується як до запису в нові об’єкти, так і до запитів PUT, які перезаписують існуючі об’єкти, і до запитів DELETE. Крім того, операції читання в Amazon S3 Select, списках керування доступом Amazon S3 (ACL), тегах об’єктів Amazon S3 та метаданих об’єкта (наприклад, об’єкт HEAD) повністю узгоджені.

Оновлення одного ключа є атомарними. Наприклад, якщо робити запит PUT до існуючого ключа з одного потоку і одночасно виконати запит GET для того самого ключа з другого потоку, можна отримати або старі дані, або нові дані.

Amazon S3 досягає високої доступності завдяки реплікації даних на кількох серверах у центрах обробки даних AWS. Якщо запит PUT успішний, дані безпечно зберігаються. Будь-яке читання (запит GET або LIST),

ініційоване після отримання успішної відповіді PUT, поверне дані, записані запитом PUT.

2.3.3 Алгоритм створення попереднього перегляду зображення

Створення попереднього перегляду для зображення – це дуже цікава річ. Вона дозволяє показувати зображення у тому стані, що його можна розпізнати при цьому він буде займати пям’ять у тисячі, а то й сотні тисяч менше ніж оригінальне зображення [24].

Існує декілька способів для створення попереднього перегляду. Найбільш поширений – це створення додаткового зображення розмірами у сотні разів менші за оригінал. Саме таке зображення і буде спочатку завантажуватись як для попереднього перегляду, після чого вона буде розтягнута до оригінальних розмірів. Проблема цього способу у тому, що поки це зменшене зображення не завантажиться – користувач буде бачити пусте повідомлення.

Інший спосіб – створювати спеціальні блоки із кольорами того самого зображення. Для цього треба поділити зображення на $N \times N$ сітку, де N – це кількість кліточок. Чим більша кількість – тим більша буде якість попереднього перегляду, і тим більший його розмір (рис. 2.7).

Рисунок 2.7 – Приклад створення попереднього перегляду по точкам

Алгоритм створення дуже простий. Треба лише узяти готове зображення й витягнути кольорі цього зображення на кожній із N клітинок (рис. 2.8). Після цього при будуванні попереднього макета, треба буде

малювати отримані прямокутники із вказаними кольорами, після чого добавити ефект розмиття, тим самим попередній перегляд буде виглядати реалістичнім та дуже легким для завантаження.

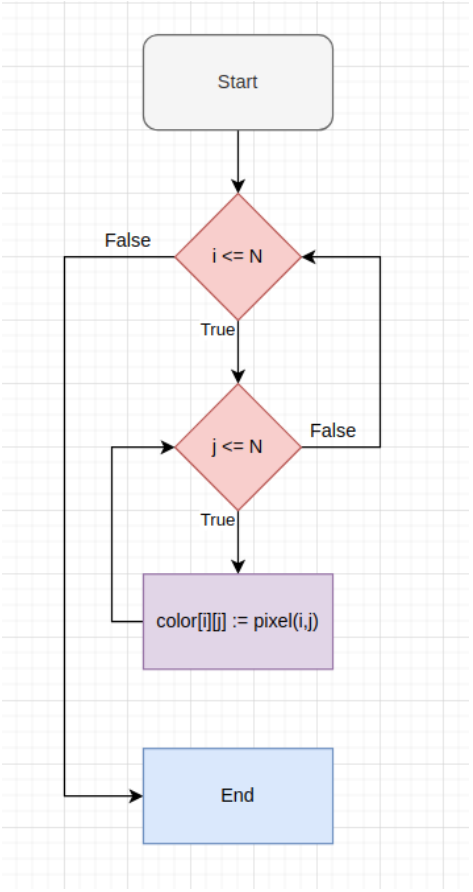

Рисунок 2.8 – Блок-схема створення попереднього перегляду

2.4 OAuth 2.0

OAuth 2.0 – протокол авторизації, що дозволяє видати одному сервісу (застосунку) права доступу до ресурсу користувача на іншому сервісі. Протокол видає необхідний доступ до логіна та пароля, а також дозволяє видавати обмежений набір прав, а не все відразу [25]. Його технологічну схему можна побачити на рисунку 2.9

OAuth 2.0 заснований на використанні базових вебтехнологій: HTTP-посиланнях, переадресаціях і інше. Тому використання OAuth можливо на будь-якій платформі з доступом до Інтернету та браузеру.

Ключове відмінність від OAuth 1.0 – просто. У новій версії немає великих схем підписів, скорочено кількість запитів, необхідних для авторизації.

Загальна схема роботи застосунків, що викростовують OAuth, таке:

– отримання авторизації;

– звертання до захищених ресурсів.

Результатом авторизації є маркер доступу – деякий ключ (звичайно просто набір символів), пред'явлення якого є пропуском до захищеного ресурсу. Звертання до нього в самому простому випадку відбувається за HTTPS з указанням в заголовках або в якості одного параметра.

У протоколі описано кілька варіантів авторизації, відповідних для різних ситуацій:

– авторизація для застосунків, що мають серверну частину (частіше за все, це сайти та вебзастосунки);

– авторизація для повністю клієнтських застосунків (мобільні та декстопні застосунки);

– авторизація по логіну і паролю;

– відновлення попередньої авторизації.

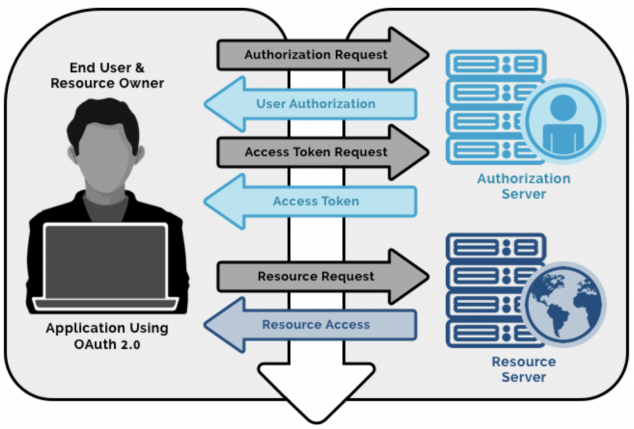

Рисунок 2.9 – Технологічна схема OAuth 2.0

OAuth 2.0 визначає чотири потоки для отримання маркера доступу. Ці потоки називаються типами грантів. Вибір того, який із них підходить для цієї задачі, залежить переважно від типу програми.

– потік коду авторизації: використовується вебпрограмами, що виконуються на сервері. Це також використовується мобільними застосунками, використовуючи так техніку, як Proof Key for Code Exchange (PKCE) [26];

– неявний потік із повідомленням у формі: використовується програмами, орієнтованими на JavaScript (односторінкові програми), що виконуються у браузері користувача;

– потік паролів власника ресурсу: використовується програмами з високою довірою;

– потік облікових даних клієнта: використовується для міжмашинного зв’язку;

– специфікація також забезпечує механізм розширюваності для визначення додаткових типів грантів. Щоб дізнатися більше про те, як працює кожен тип гранту та коли його слід використовувати, див. Потоки автентифікації та авторизації.

OAuth 2.0 використовує дві кінцеві точки: кінцеву точку /authorize і кінцеву точку /oauth/token.

Кінцева точка /authorize використовується для взаємодії з власником ресурсу та отримання авторизації на доступ до захищеного ресурсу.

Кінцева точка /oauth/token використовується програмою, щоб отримати маркер доступу або маркер оновлення. Він використовується всіма потоками, крім неявного потоку, оскільки в цьому випадку маркер доступу видається безпосередньо.

У потоці коду авторизації програма обмінює код авторизації, отриманий від кінцевої точки авторизації, на маркер доступу.

3 ПРАКТИЧНА РЕАЛІЗАЦІЯ МЕСЕНДЖЕРА

3.1 Етапи створення месенджера

3.1.1 Створення скелету вебзастосунку

На цьому етапі перш за все створюється скелет вебзастосунку. Оскільки цей проєкт може буде великий, то потрібна найбільш зрозуміла структура [27] (рис. 3.1).

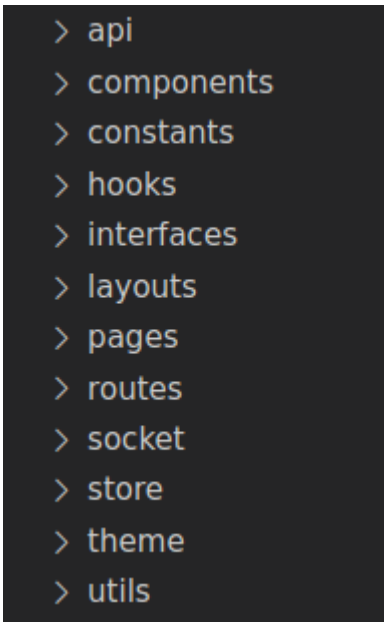

Рисунок 3.1 – Структура вебзастосунку

Пояснення щодо структури:

– api – зберігає функції та типи для взаємодії із сервером;

– components – зберігає усі елементи, що будуть відображатися у вебзастосунку;

– constants – зберігає незмінні дані, до яких ми будемо звертатися в різних місцях вебзастосунка;

– hooks – зберігає React функції;

– interfaces – зберігає опис усіх типів даних вебзастосунка;

– layouts – зберігає структури сторінок;

– pages – зберігає сторінки, що будуть відображатися;

– routes – зберігає файли, що описують навігацію по вебзастосунку;

– socket – директорія, в якій створюється та обробляються усі методи Socket.IO;

– store – зберігає глобальні стани вебзастосунка;

– theme – зберігає особливості теми вебзастосунка;

– utils – зберігає другорядні функції вебзастосунка.

3.1.2 Створення головної сторінки

На цьому етапі треба створити головну сторінку. Перш за все, описується для цього спеціальна структура, а точніше макет сторінки, яку треба буде зображати (рис. 3.2).

```
<Root>
  <Sidebar />
  <PageContent>
    <Container>{children}</Container>
  </PageContent>
</Root>
```

Рисунок 3.2 – Лістинг коду для створення макета головної сторінки

За цим макетом буде показано зліва сайдбар с користувачами, а усередині сторінку із чатом. Саме для цього було використано спеціальне ключове слово *children*, яке відповідає за те, що макет буде показувати те, що в нього вставлено (в цьому випадку буде вставлятися інший компонент, а саме сторінка із чатом). Тепер створивши компоненти сайдбару та сторінки із чатом, можна подивитися на те, як це буде виглядати зараз на вебзастосунку (рис. 3.3).

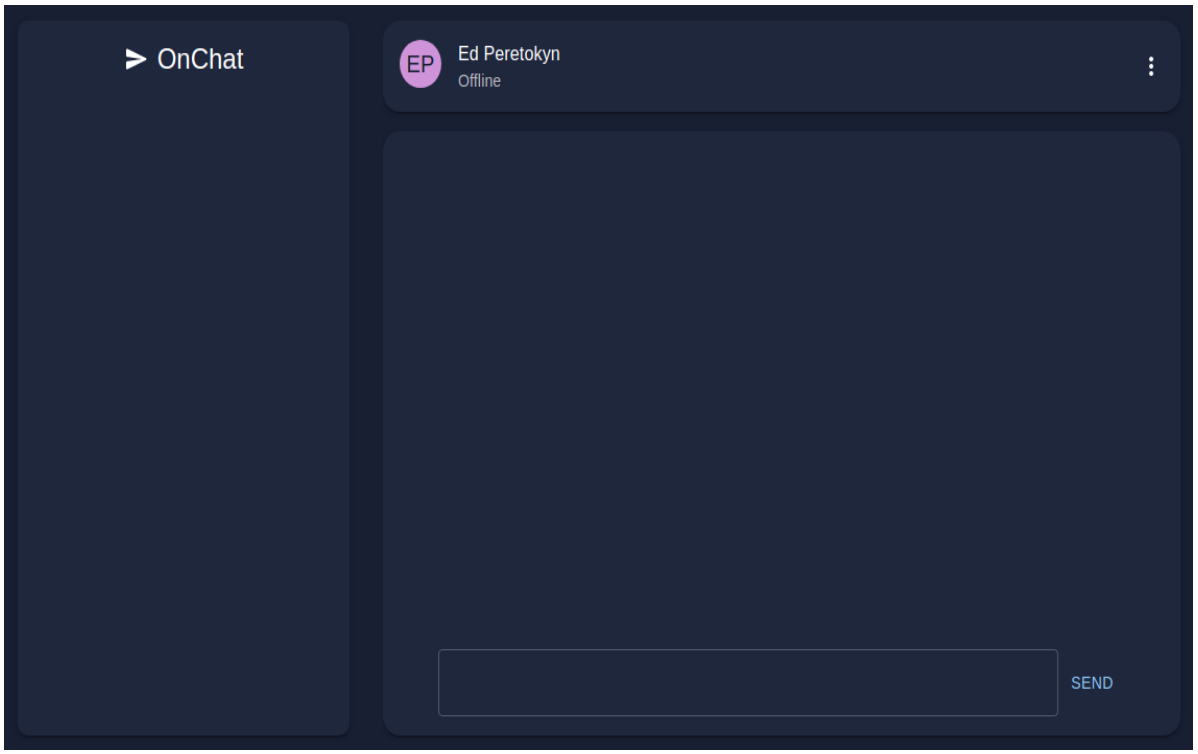

Рисунок 3.3 – Головна сторінка месенджера

3.1.3 Реалізація з’єднання використовуючи Socket.IO

Тепер коли вже є головна сторінка та можна із нею взаємодіяти, треба реалізувати підключення до сервера. А точніше WS серверу, щоб можна було у режимі реального часу передавати та приймати дані (рис. 3.4).

```
this.client = io(`${process.env.API_URL}`, {
  query: {
    id: userId,
  },
})
this.client.connect()
```

Рисунок 3.4 – Лістинг коду для підключення до WS серверу

3.1.4 Реалізація алгоритму відправлення/отримання повідомлень

Для того, щоб відправити повідомлення, потрібно перш за все створити його. Для цього описується структуру повідомлення (рис. 3.5).

```
export type Message = {
  id: string
  senderId: string
  timestamp: string
  isViewed: boolean
  isPending?: boolean
} & (TextMessageData | FileMessageData)
```

Рисунок 3.5 – Лістинг коду для опису даних повідомлень

На цьому етапі, тепер можна створювати повідомлення за цією структурою. Це було зроблено для того, щоб вона була завжди однакова та під час розробки вебзастосунку можна було її дотримуватися.

Для надсилання повідомлень буде використовувати метод *emit*, який надає бібліотека *Socket.IO* (рис. 3.6). У нього буде передаватися тип події, яку буде визначено та другорядні дані, які потім будуть зчитуватися на сервері.

```
socket.emit('send_message', {
  userId: selectedUser.googleId,
  text: message.text,
  identity: message.id,
})
```

Рисунок 3.6 – Лістинг коду для надсилання повідомлень використовуючи метод *emit* бібліотеки *Socket.IO*

Для отримання повідомлень, буде використовуваться схожий підхід, але замість насдилання, вебзастосунок буде слухати від сервера усі події, які він буде надсилати. Зокрема нове повідомлення (рис. 3.7).

```
const newMsgSub = socket.subscribe('new_message', (message) => {
  if (message.senderId === selectedUser.googleId) {
    setMessages((state) => [message, ...state])
  }
})
```

Рисунок 3.7 – Лістинг коду для отримання повідомлень від сервера

3.1.5 Реалізація алгоритму надсилання зображень на сервер

Для надсилання зображень можна використовувати той самий підхід що й для звичайних повідомлень. Але головна відмінність буде у тому, що перш за все треба завантажити файл до браузера. Після цього, як він буде завантажений, зчитуються його розміри та розраховується його співвідношення, щоб потім можна було його коректно показати. Для того, щоб надіслати його до сервера, буде зчитуватися його буфер та його тип, щоб відрізнити зображення від звичайного файлу.

Коли зображення готове до надсилання, можна зробити те саме, що й до цього: відправити це повідомлення що зберігає зображення, використовуючи метод *emit*. Алгоритм зчитування та надсилання зображення буде надано у Додатку А.

3.1.6 Реалізація авторизації за допомогою OAuth 2.0

Перш за все, треба створити макет для сторінки авторизації. Було зроблено теж саме, що робили для головної сторінки (рис. 3.8).

```
const AuthLayout = ({ children }: Props) => {
  return <Root>{children}</Root>
}
```

Рисунок 3.8 – Лістинг коду для створення макета для авторизації

Можна побачити, він (макет) трохи менший за попередній, бо все що треба на ньому, це показати панель авторизації посередині екрану. Тому створивши сторінку авторизації, вона буде виглядати так, як на рисунку 3.9.

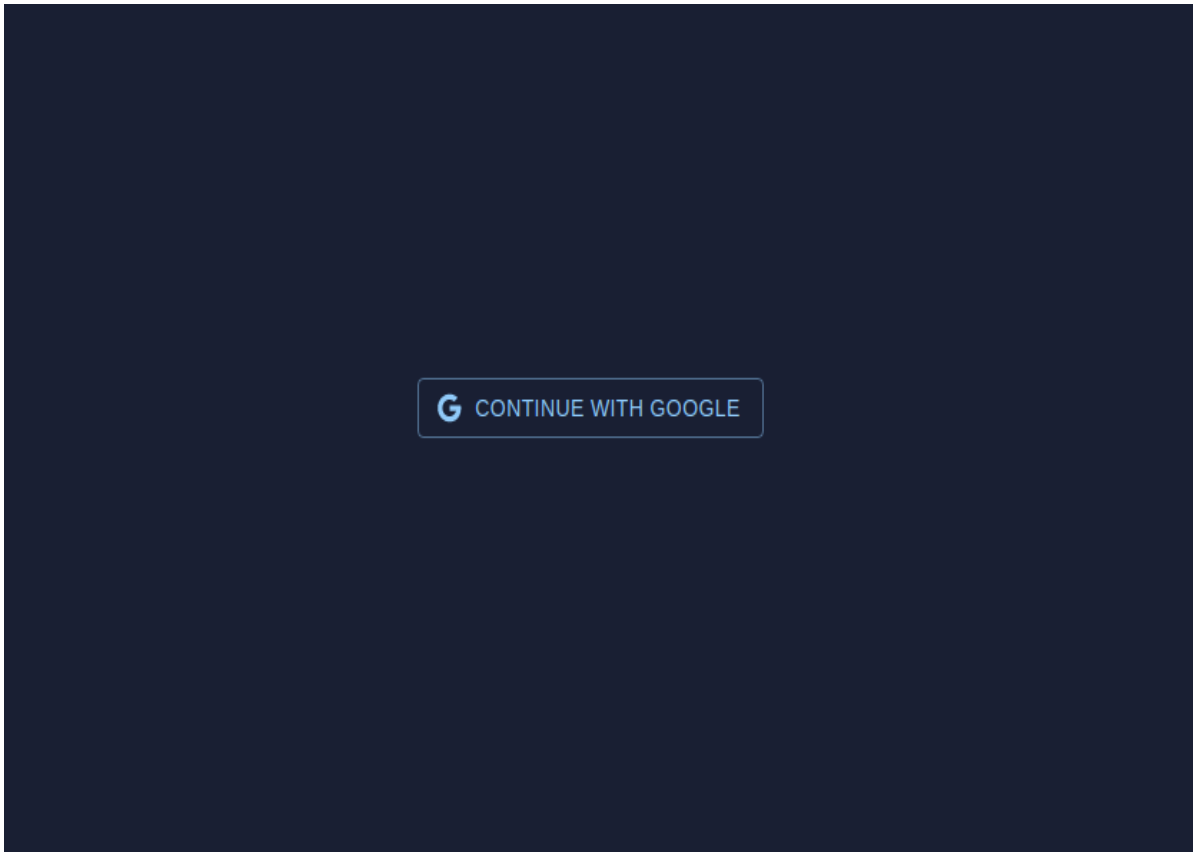

Рисунок 3.9 – Сторінка авторизації

Для реалізації авторизації було використовувати OAuth 2.0. Так як проєкт є не комерційним, то можна знехтувати такими поняттями як *access and refrest tokens*. Саме тому, в даному випадку для коритсувача важливо лише пройти авторизацію лише один раз. Для цього було використовувати *useGoogleLogin* що надає ця бібліотека (рис. 3.10).

```
const { signIn, loaded } = useGoogleLogin({
  onSuccess: (response) => {
    const imageUrl = (response as GoogleLoginResponse)
      .getBasicProfile()
      .getImageUrl()
      .replace('=s96-c', '')

    login({
      ...(response as GoogleLoginResponse).profileObj,
      imageUrl,
    })
  },
  clientId,
  isSignedIn: true,
  cookiePolicy: 'single_host_origin',
})
```

Рисунок 3.10 – Лістинг коду для авторизації використовуючи *useGoogleLogin*

3.1.7 Розробка скелета сервера

Для сервера буде використовуватися схожа структуру, але зі своїми особловостями (рис. 3.11). Головні відмінности будуть пов’язані з тим, що буде використовутися технологія *NestJS* [28], а вона, у свою чергу, працює трохи інше, бо використовує таку технологію як *DI* [29].

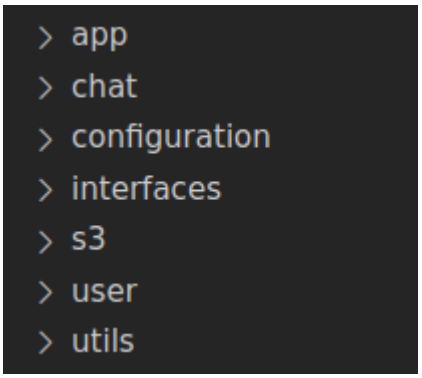

Рисунок 3.11 – Структура сервера

Пояснення щодо структури:

– app – зберігає основні модулі та провайдери кореневого проекта;

– chat – зберігає *DDD* [30] структуру для чату;

– configuration – зберігає конфігурації для підключення до *MongoDB* серверу;

– s3 – зберігає сервіс що дозволяє взаємодіяти із AWS S3;

– user – зберігає *DDD* структуру для користувачів;

– utils – зберігає другорядні функції сервера.

3.1.8 Реалізації алгоритму приєднання користувачів

Для того, щоб відстежити нового користувача, що хоче приєднатися до серверу, буде використовуватися таж сама бібліотека *Socket.IO*. Для того, щоб користувач зміг користуватися вебзастосунком, потрібно перш за все перевірити, чи є він у базі даних і тільки після цього, можна створити для нього спеціальну кімнату, в якій він буде знаходиться та на яку інші користувачі зможуть відправляти повідомлення. Спеціальна кімната створюється для того, щоб якщо користувач відкриє вебзастосунок одразу на декількох вкладках браузера, то усі ці вкладки та з’єднання йшли до однієї кімнати, куди будуть надсилати повідомлення інші користувачі (рис. 3.12). Як ще один з можливих варіантів, які використовуються в інших месенджерах, можна було б блокувати іншу вкладку. Тим самим, тільки одна з них могла б очікувати нові повідомлення. Але це не є розв’язанням проблеми, адже, якщо користувач захоче відкрити інший чат або йому зручніше користуватися одразу двома вкладками, то лише цей спосіб дасть користувачу цю можливість і не буде блокувати його.

```
async handleConnection(client: Socket) {
  const connectedUserId = client.handshake.query['id'] as string
  this.logger.log(`Client connected: ${connectedUserId}`)

  const connectedUsers =
    this.server.sockets.adapter.rooms.get(connectedUserId)
  if (!connectedUsers) {
    const interval = setInterval(async () => {
      const user = await this.userService.findByGoogleId(connectedUserId)
      if (user) {
        const updatedUser = await this.userService.updateUserStatus(
          connectedUserId,
          true,
        )
        this.logger.debug(updatedUser)
        this.server.emit('connected', updatedUser)
        clearInterval(interval)
      }
    }, 500)
  }

  client.join(connectedUserId)
}
```

Рисунок 3.12 – Лістинг коду для підключення користувача до *WS* серверу

3.1.9 Реалізація підключення до *MongoDB*

Для підключення до бази даних, треба перш за все створити її кластер. Для цього треба перейти до офіційного сайту *MongoDB* та створити новий акаунт. Після цього перейти до створення бази даних, де й буде створено новий кластер.

Після того як акаунт був створений, треба перейти до створення бази даних. Спочатку обирається хмарний провайдер (рис. 3.13).

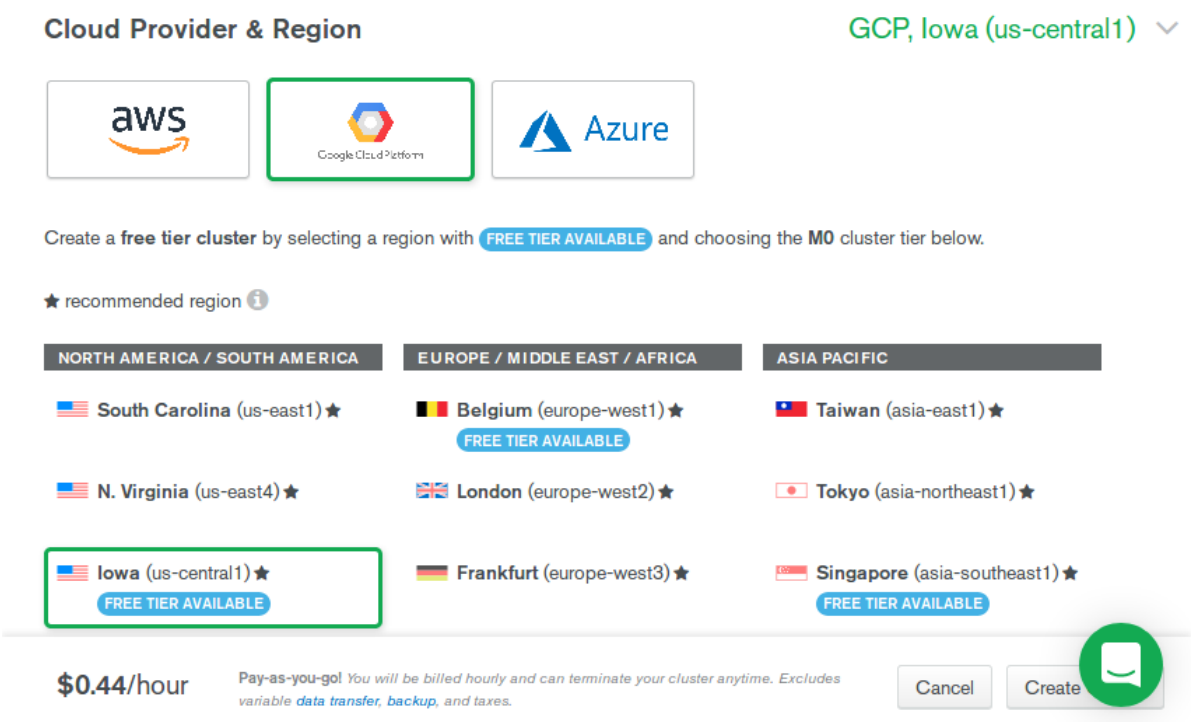

Рисунок 3.13 – Форма вибору хмарного провайдера

Після цього створюється новий кластер бази даних. Для цього проєкту буде обрано безкоштовний варіант, але якщо проєкт буде комерційним, то можна обрати необхідний для бізнесу кластер, в залежності від його потреб (рис. 3.14). В данному випадку було обрано кластер *M0*, що надає 512 МБ простору для зберігання даних, останню версію *MongoDB* з *WiredTiger* в якості механізму зберігання, а також набір реплік з трьох вузлів та обмежену, але разом з тим достатньо велику пропускну здатність 10 ГБ у неділю.

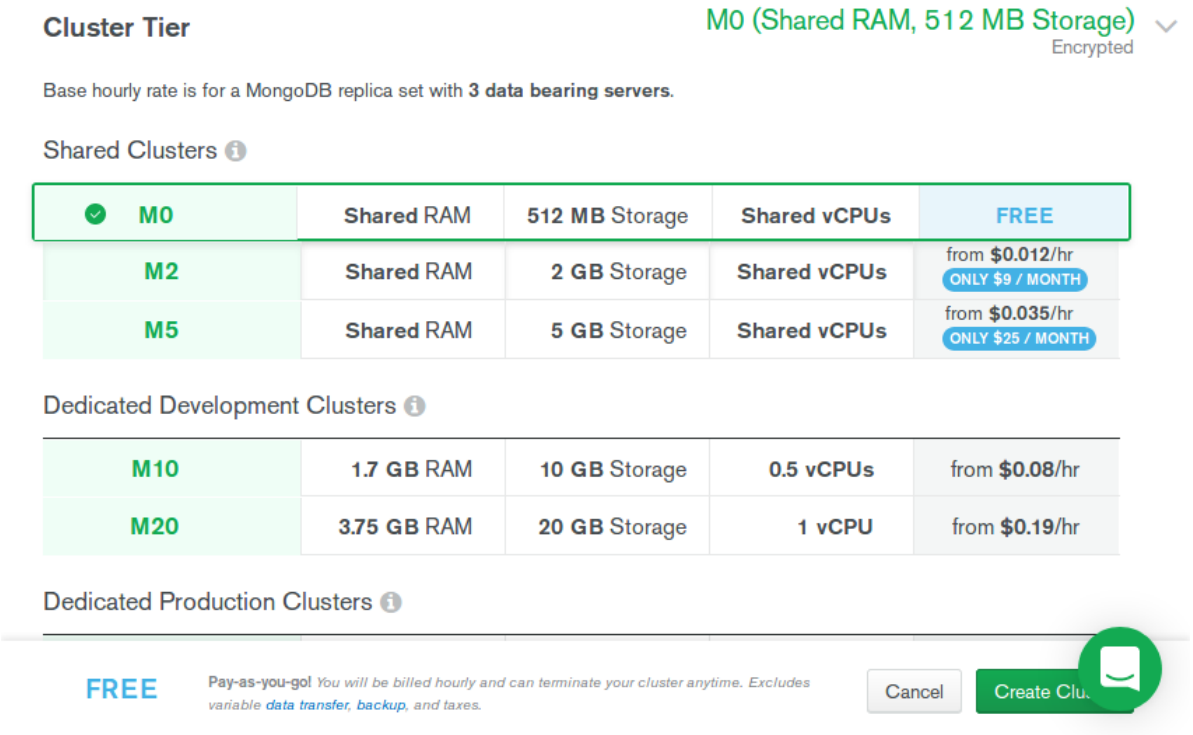

Рисунок 3.14 – Вибір кластера бази даних

Після цього треба дати назву кластеру і на цьому його створення закінчено. Далі треба буде лише його налаштувати за вказівками.

Тепер для сервера необхідно отримати строку з'єднання. Для цього треба перейти на вкладку *Overview* та натиснути кнопку *Connect*. У відкритому вікні обрати параметр *Connect Your Application* та натиснути кнопку *I'm using driver 3.6 or later*. Після цього скопіювати цю строку і зберігти її у проєкті із сервером.

Усі приватні дані слід зберігати у спеціальному файлі (рис. 3.15). Із основних даних це:

– паролі;
– пошти;
– строки з'єднання;
– секретні ключі та інше.

Розширення файлу має назву *.env* що з англійської означає *environment* (середовище). До цього файлу ніхто окрім сервера не буде мати доступ.

```
.env
1   MONGODB_DB_URI=
2   PORT=
3   AWS_S3_BUCKET=
4   AWS_ACCESS_KEY_ID=
5   AWS_SECRET_ACCESS_KEY=
6
```

Рисунок 3.15 – Лістинг коду для створення середовища

Тепер коли в є все необхідне, можна створити сервіс, що буде зчитувати отриману строку з’єднання та повертати її. Для цього треба створити новий файл у директорії *configuration* (рис. 3.16).

```
@Injectable()
export class AppConfigurationService {
  private readonly _connectionString!: string

  get connectionString(): string {
    return this._connectionString
  }

  constructor(private readonly _configService: ConfigService) {
    this._connectionString = this._getConnectionStringFromEnvFile()
  }

  private _getConnectionStringFromEnvFile(): string {
    const connectionString = this._configService.get<string>('MONGODB_DB_URI')
    if (!connectionString) {
      throw new Error(
        'No connection string has been provided in the .env file.',
      )
    }

    return connectionString
  }
}
```

Рисунок 3.16 – Лістинг коду для створення сервісу що зчитує строку підключення із базою даних

Далі використовуючи можливості *NestJS,* а також технологію *DI,* можна імпортувати модуль та зробити асинхронне з’єднання із базою даних. Для

цього буде використано бібліотеку *@nestjs/mongoose,* а саме її метод *forRootAsync* (рис. 3.17).

```
MongooseModule.forRootAsync({
  imports: [AppConfigurationModule],
  inject: [AppConfigurationService],
  useFactory: (appConfigService: AppConfigurationService) => {
    const options: MongooseModuleOptions = {
      uri: appConfigService.connectionString,
      useNewUrlParser: true,
      useUnifiedTopology: true,
    }
    return options
  },
}),
```

Рисунок 3.17 – Лістинг коду для з’єднання із базою даних

3.1.10 Реалізація алгоритму обміну повідомленнями

Осклільки вже здійснено підключення до бази даних, а також можливість користувачів з’єднуватись із сервером, тепер треба зробити так, щоб вони (користувачі) могли обмінюватися повідомленнями. Для цього треба буде слухати усі події що надсилає користувач, та якщо одна з них та, що буде визначена у реалізації цього методу, то саме цю подія требу буде обробляти. Спочатку, коли користувач надіслав повідомлення, треба перевірити, чи є він у списку користувачів, що були занесені у базу даних. Теж саме треба зробити й для того, кому він надсилає повідомлення. Робиться це для того, щоб у разі помилки, або спроби користувача зламати сервер, нічого не вийшло. Після того як усі користувачі пройшли перевірку і є справжніми користувачами вебзастосунку, можна надіслати це повідомлення користувачу, кому воно мало прийти, а відправнику повідомити що його повідомлення було успішно відправлено. Повний алгоритм надсилання повідомлення зображено у Додатку Б.

3.1.11 Зберігання файлів у хмарному сховищі

Тепер треба розширити функціонал та надати користувачу можливість відправляти не лише текстові повідомлення, а також файли. Для цього спочатку треба створити хмарне сховище. Для цього буде використовуватися безкоштовне хмарне сховище, що надає *Amazon*. Спочатку треба зареєструватися на цьому ресурсі. Після того як реєстрація була пройдена, буде перехід на головну сторінку. Далі треба перейти на вкладку *Bucket* та створити нове сховище (рис. 3.18).

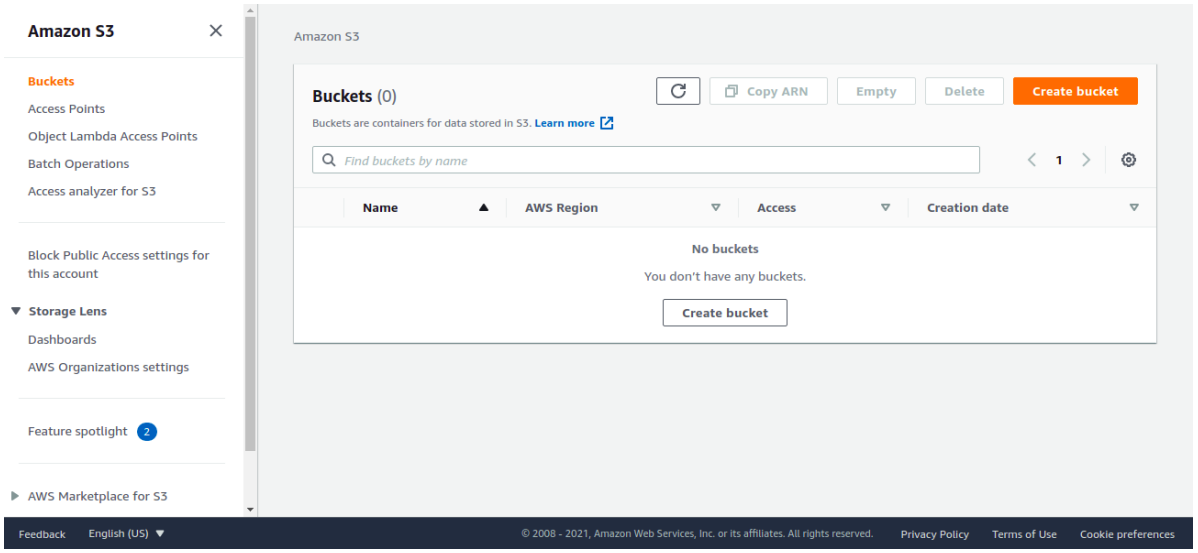

Рисунок 3.18 – Створення нового сховища

При натисканні кнопки створення сховища, можна побачити велику можливість різного налаштування. На даному етапі буде ігнорувано більша кількість, але де-які потребують розгляду.

Перший – це *AWS Region*. Цей параметр дає можливість обрати регіон сховища. Рекомендується обирати найближчий до користувача регіон. Це дозволить збільшити швидкість передачі даних та зменшить час очікуванню, що дуже важливо.

Далі йде *Block Public Access settings for this bucket*. Це налаштування та блокування публічного доступу до сховища. На даному етапі нічого не

змінювалося, але при необхідності можна буде зробити його публічним, щоб будь-які користувачі мали доступ до файлів, що буде надсилати до нього сервер (рис. 3.19).

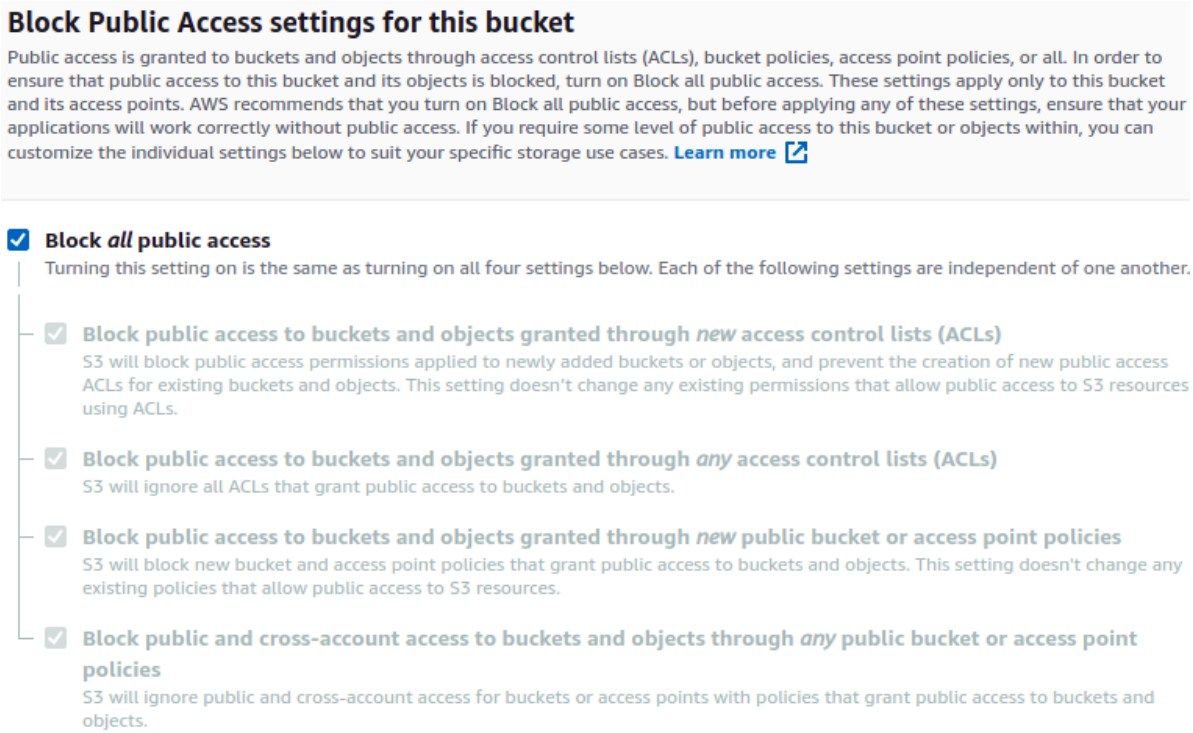

Рисунок 3.19 – Налаштування публічного доступу до сховища

Після цього як все необхідне обрано, можна створити сховище натискаючи *Create Bucket*. Коли воно буде створено, можна буде побачити відповідну сторінку із сховищем. На ній можна у будь-який час змінити будь-які налаштування сховища, а також переглянути файли, які в ньому зберігаються (рис. 3.20). Взагалі, можна створити не одне сховище. Гарною практикою є створення сховища для кожного типу файлів. Так як в цьому випадку зберігається лише один тип файлів, а саме: файли що надіслали користувачі, то досить лише одного сховища.

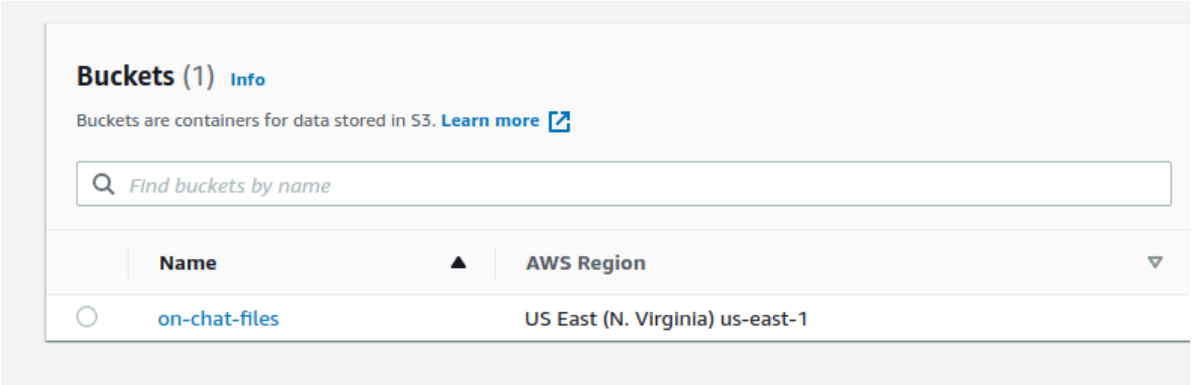

Рисунок 3.20 – Каталог створених сховищ

Після того як сховище створено, можна отримати дані для з'єднання із цим сховищем, щоб сервер міг відправляти до нього файли, що надіслав користувач. Для цього треба перейти на вкладку *My Security Credentials,* що буде у випадаючому меню (рис. 3.21).

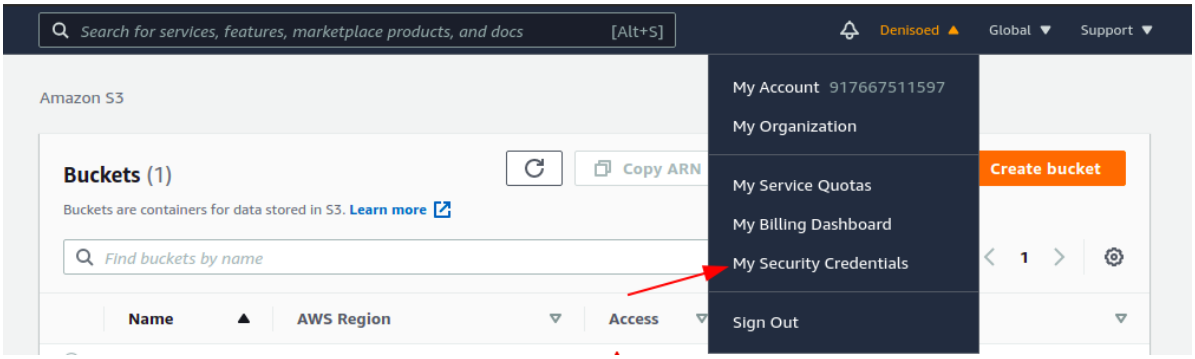

Рисунок 3.21 – Вкладка *My Security Credentials*

Далі необхідно створити новий ключ для доступу. Для цього треба натиснути кнопку *Create New Access Key*. Після цього з'являться на екрані секретні ключі, які будуть вставлено до серверу, у той самий файл середовища.

Оскільки уся необхідна інформація здобута, можна реалізувати алгоритм надсилання файлів. Для цього треба створити сервіс що буде відповідати за надсилання файла до сховища, та повертати його адресу. Цю адресу необхідно буде зберігати у базі даних, яку потім буде зчитувати

користувач та завантажувати зображення, що в кінці буде показуватися у чаті. Реалізація сервіса зображено у Додатку В.

3.1.12 Реалізація алгоритму створення попереднього перегляду

Останнім етапом розробки сервера є створення попереднього перегляду зображення. Треба зробити це для того, щоб коли користувач отримає зображення, воно одразу відобразилось для нього, але у поганій якості. Це дозволить користувачу зрозуміти, що він зараз отримає зображення, і що саме за зображення це буде.

Для цього треба реалізувати алгоритм, який буде зчитувати надіслане зображення, та створювати його попередній перегляд (рис. 3.22). Для цього буде використовуватися бібліотека *Jimp*. Вона дозволяє зчитати буфер зображення, яке надіслав користувач. Далі за допомогою алгоритму розбиття зображення на сітку, треба поділити його на частини та зчитувати по одному пікселю з цієї сітки. Далі занести ці дані про пікселі до масиву, який буде зберігатися у базі даних. Оскільки розміри масиву будуть 30×30, то попередній перегляд буде займати не більше 1-2 КБ, у той час, як саме зображення може мати розміри 300 МБ або й більше.

```
import Jimp from 'jimp/es'

export const createPreview = async (buffer: Buffer) => {
  const image = await Jimp.read(buffer)
  const dHeigth = image.getHeight() / 29
  const dWidth = image.getWidth() / 29

  const pixels = new Array(30).fill(0).map((_, rIndex) =>
    new Array(30).fill(0).map((_, cIndex) => ({
      x: Math.floor(cIndex * dWidth),
      y: Math.floor(rIndex * dHeigth),
    })),
  )

  const colors = pixels.map((points) => {
    return points.map((point) => {
      const hex = image.getPixelColor(point.x, point.y)
      return Jimp.intToRGBA(hex)
    })
  })

  return colors
}
```

Рисунок 3.22 – Лістинг коду для створення попереднього перегляду зображення

3.2 Огляд створеного вебзастосунка

3.2.1 Тестування авторизації користувача

Перш за все, треба перевірити, чи працює авторизація. Для цього було використано існуючу пошту на *Gmail* акаунті. Спочатку треба авторизуватися використовуючи пошту (рис. 3.23, 3.24).

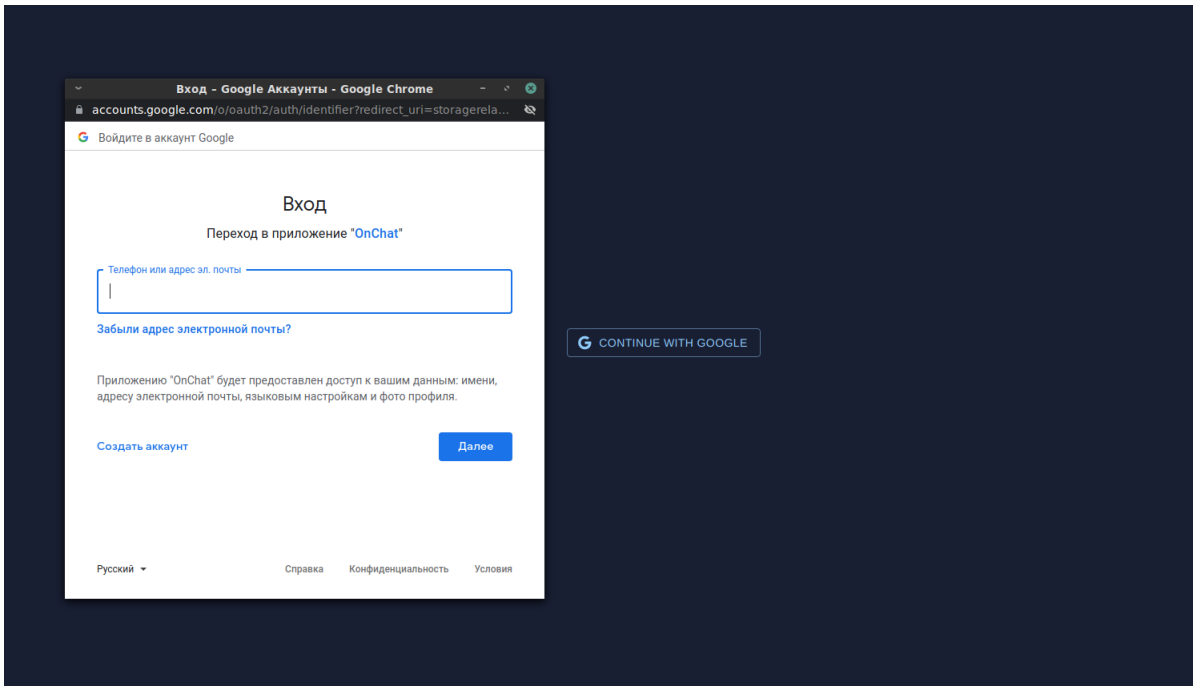

Рисунок 3.23 – Вхід до вебзастосунка використовуючи OAuth 2.0

Рисунок 3.24 – Відображення головної сторінки при успішній авторизації

Тепер зайти з іншого акаунту з іншого браузера, щоб появився користувач, якому можна надіслати повідомлення.

Після авторизації нового користувача, одразу можна побачити його у списку контактів, якому можна надіслати повідомлення (рис. 3.25).

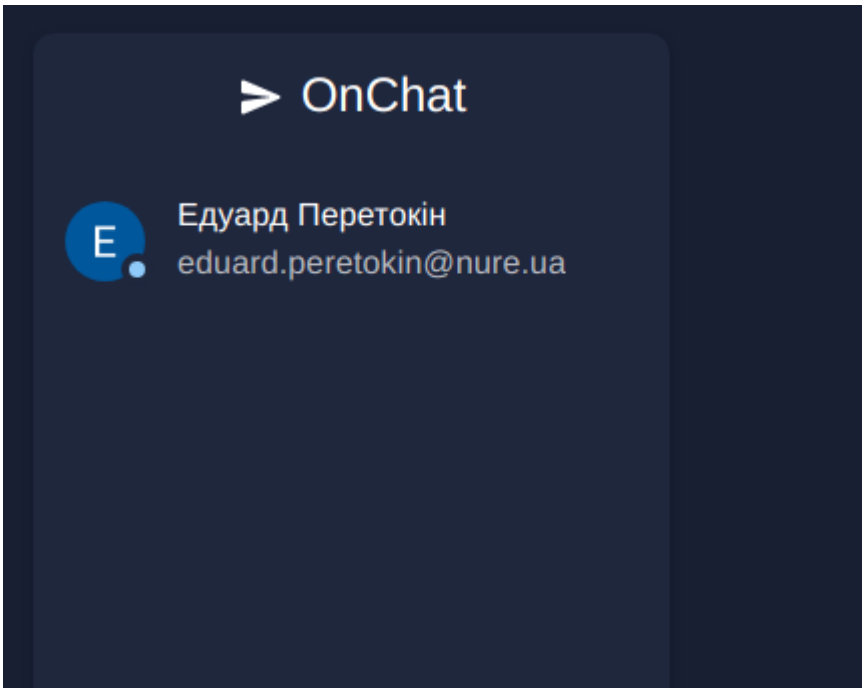

Рисунок 3.25 – Вхід нового користувача до вебзастосунку.

3.2.2 Тестування можливості надсилати повідомлення

Тепер треба написати цьому користувачу та подивитися, що трапиться на його стороні. Для цього було натиснено на нього, після чого була відкрита сторінка із чатом, в якій надіслано користувачу привітання (рис. 3.26–3.28).

Після того як повідомлення було надіслано і воно пройшло перевірку, можно побачити що повідомлення було відправлене та знаходиться у чаті. У разі якщо тепер оновити сторінку, це повідомлення залишиться.

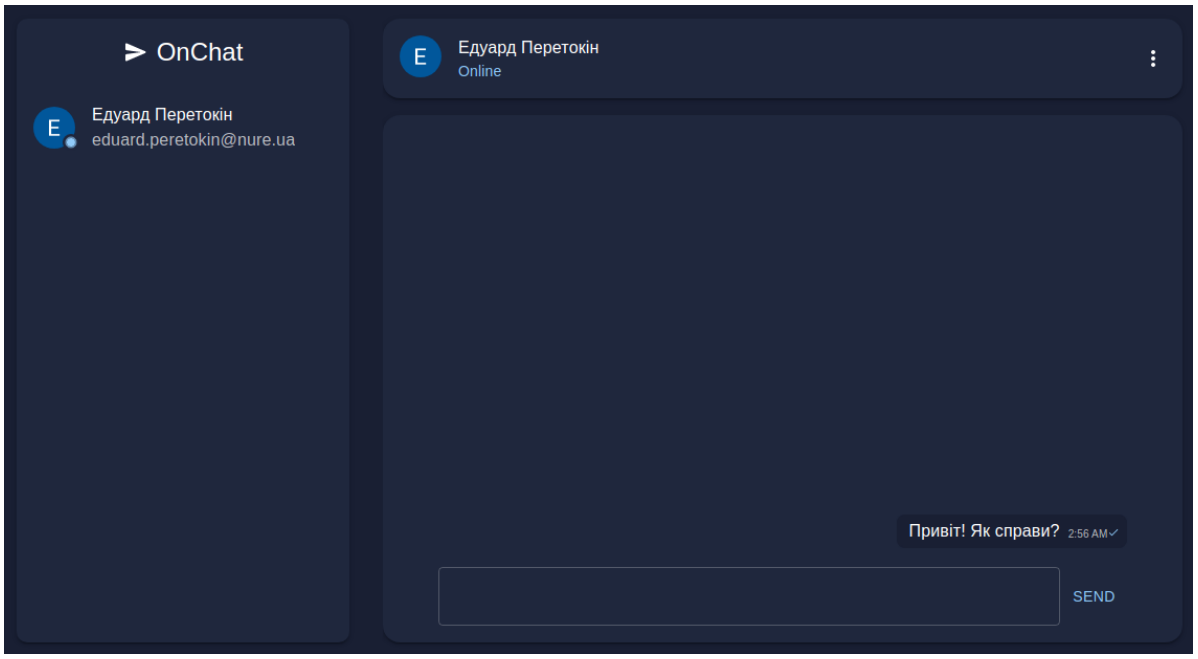

Рисунок 3.26 – Надсилання повідомлення користувачу

Сам же користувач, якому надійшло повідомлення, побачить зліва у сайдбарі оповіщення, що дехто написав йому. У кружечку буде відображатися кількість нових повідомлень від користувача, які він ще не прочитав.

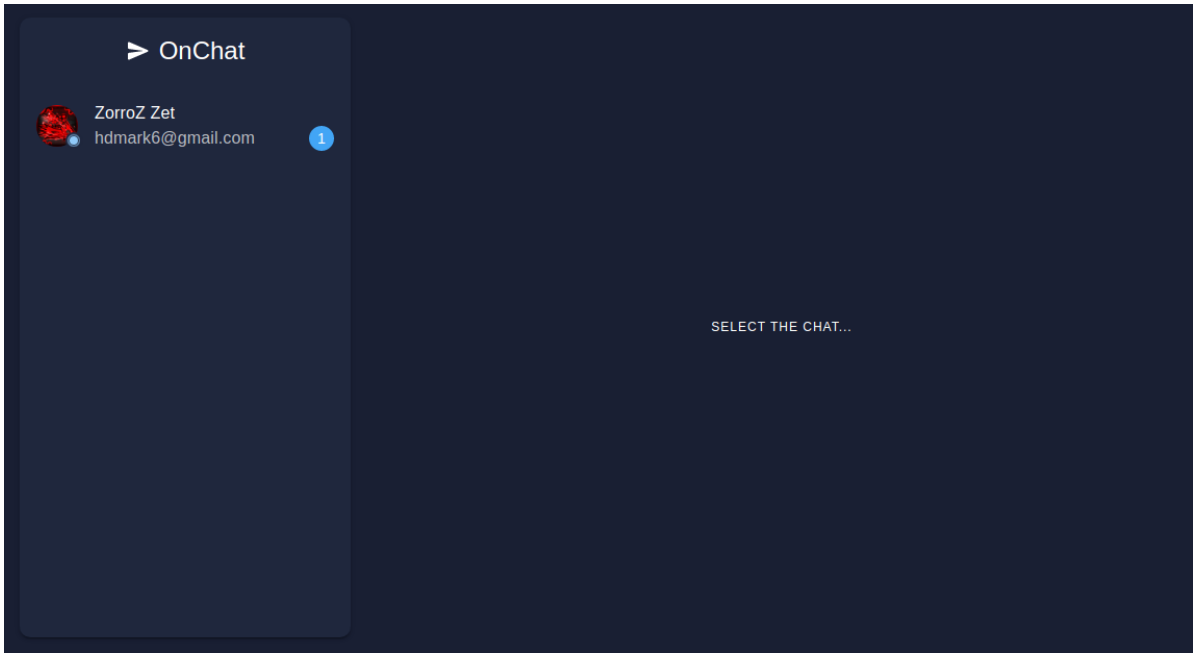

Рисунок 3.27 – Отримання повідомлення від користувача

Якщо відкрити чат з користувачем, що надіслав повідомлення, то можна побачити що оповіщення о нових повідомленнях зникло, бо було прочитано усі нові повідомлення. У самому ж чаті відображається це нове повідомлення.

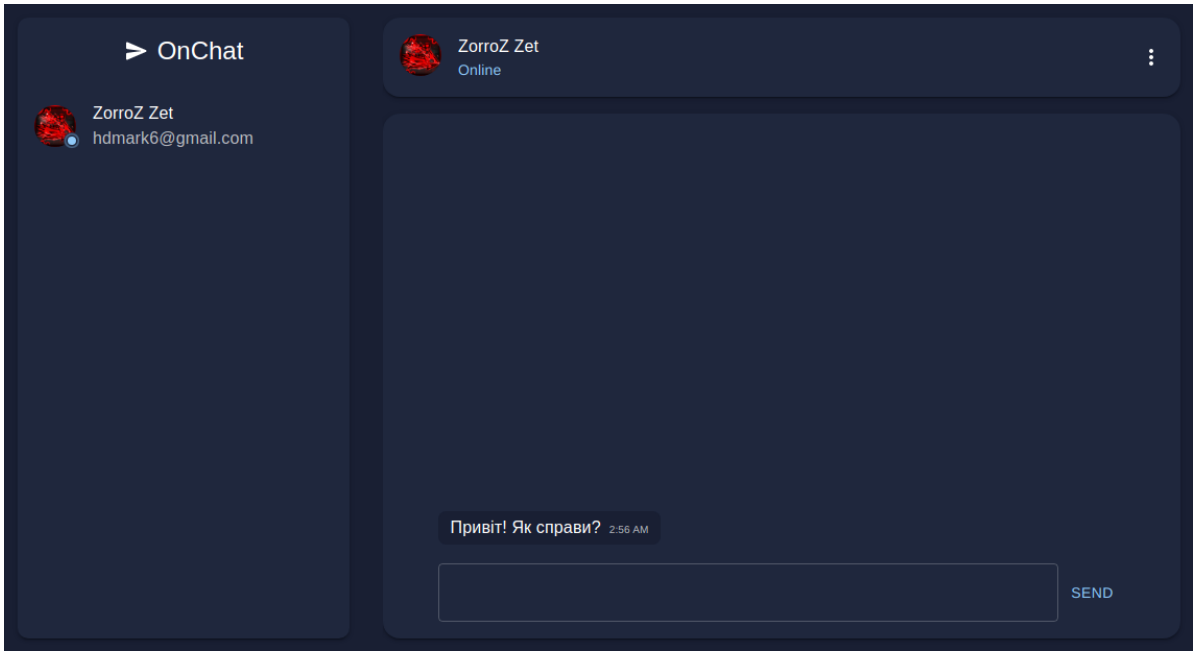

Рисунок 3.28 – Відкриття чату з новим повідомленням

3.2.3 Тестування можлиовтсі надсилати зображення

А тепер треба перевірити можливість відправити зображення, та подивитися на саме зображення, а також його попередній перегляд, який було згенерувано.

При надсиланні нового зображення спостерігається два стану повідомлення (рис. 3.29, 3.30). Перший – це зображення надсилається до сервера. у цей час зображення буде відображатися із затемненим фоном, та із полосою завантаження. Далі, після того, як зображення буде надіслано, воно буде показуватися у тому самому виді, в якому воно є.

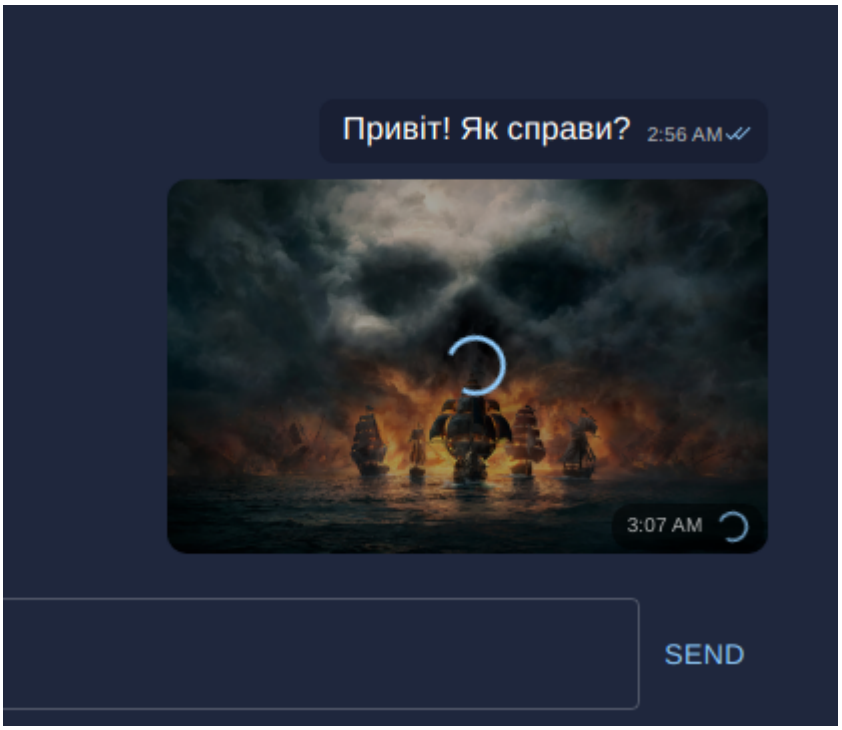

Рисунок 3.29 – Стан повідомлення, що надсилається

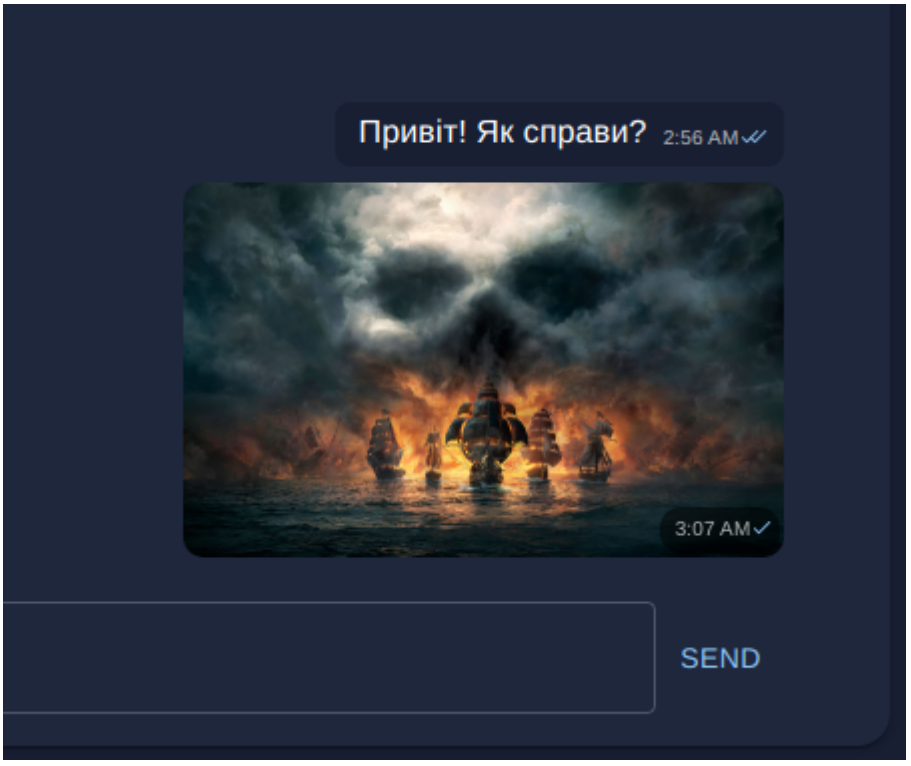

Рисунок 3.30 – Повідомлення було успішно надіслано

Після того, як повідомлення надіслано, воно так само як і звичайне текстове повідомлення надійде до користувача. Тепер треба подивитися, як це

повідомлення із зображенням буде відображатися у користувача, якому його надіслали (рис. 3.31, 3.32).

Як тільки користувач отримав зображення, він бачить його попередній перегляд.

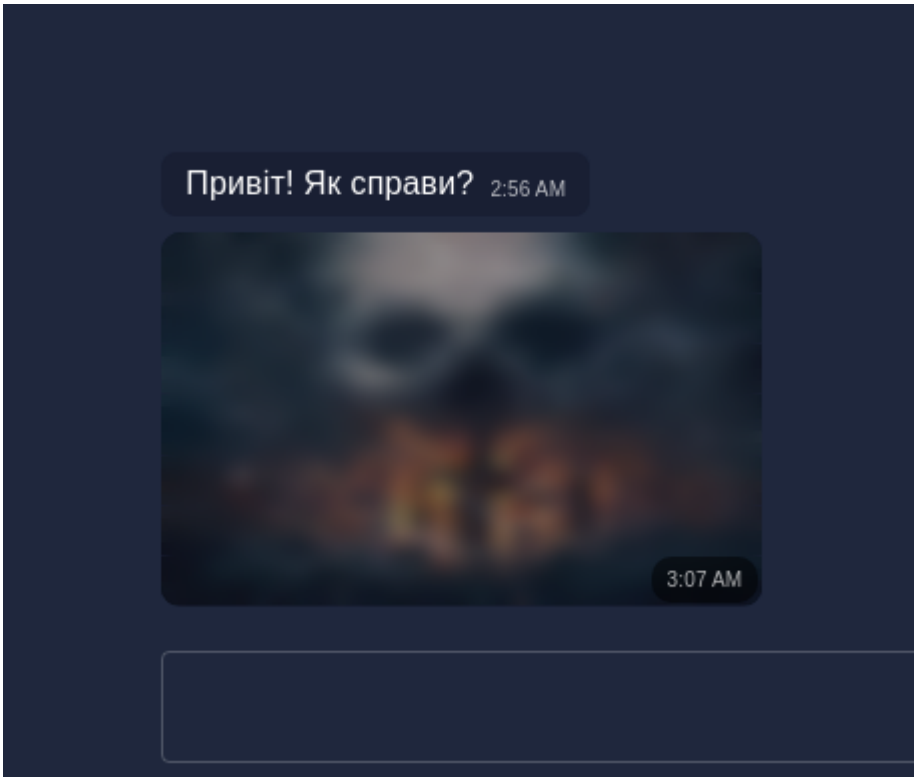

Рисунок 3.31 – Нове повідомлення із зображенням у стані попереднього перегляду

Але коли це зображення повністю завантажиться, воно одразу замінить попередній перегляд на оригінал, тим самим не треба буде бачити пустий блок з повідомленням доки зображення не завантажиться.

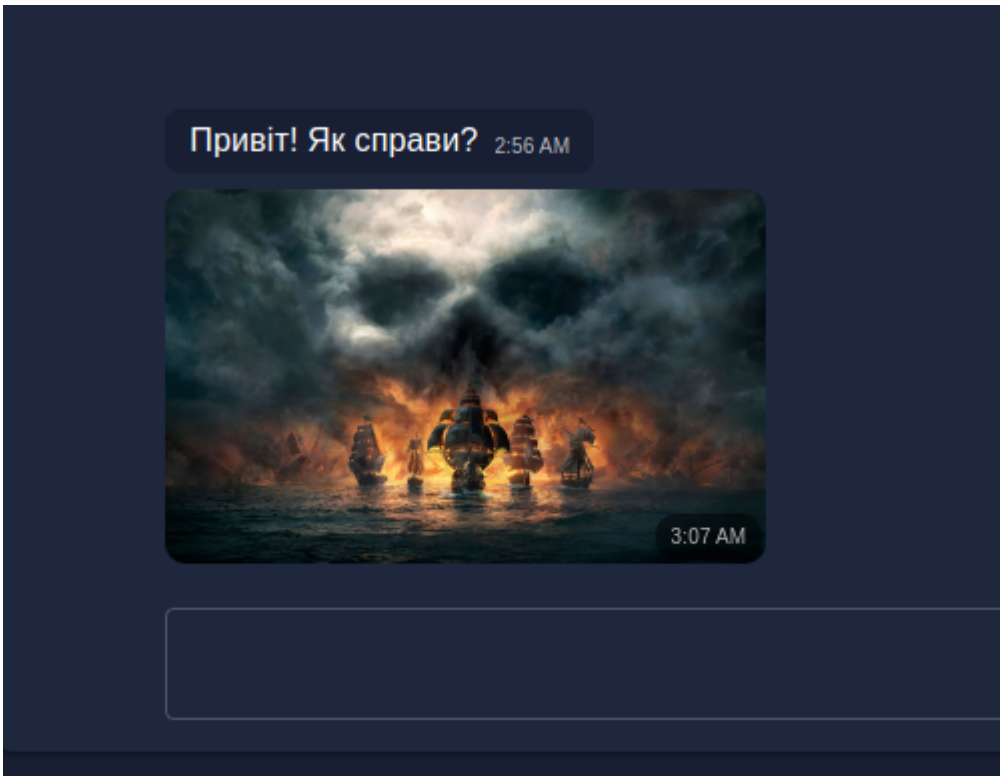

Рисунок 3.32 – Нове повідомлення із зображенням, після того, як зображення завантажилось

ВИСНОВКИ

У рамках кваліфікаційної роботи було розроблено месенджер конструктора сайту за допомогою технологій React та NestJS.

У ході роботи були вирішені наступні завдання:

– проаналізовано усю доступну літературу, документацію, щодо обраної вебтехнології та архітектури проєктування вебзастосунку;

– проаналізовано літературу, щодо можливостей обраної вебтехнології;

– розглянуто основні моменти у розробці месенджера;

– налагоджено зв’язок між сервером та клієнтом за допомогою сокетів;

– реалізовано алгоритм створення повідомлень та чатів з можливість їх зберігання у базі даних;

– обрано оптимальні структури для вебзастонку та серверу;

– створено повний проєкт із можливостями подальшого удосконалення;

– розроблена база даних для збереження даних, які були використанні у вебзастосунку;

– налаштовано з’єднання з AWS S3, та створено хмарне сховище;

– розроблено методи отримання та надсилання повідомлень;

– розроблено методи отримання та надсилання зображень;

– розроблено та реалізовано методи створення попереднього перегляду зображення.

– розроблено програмний продукт з використанням даної вебтехнології та обраної архітектури розробки;

– виконано огляд створеного вебзастосунку з усіма методами;

– проаналізовано подальший розвиток вебзастосунку:

– зроблено висновки, щодо виконаної роботи.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Masse, M. (2011). REST API design rulebook: designing consistent RESTful web service interfaces. " O'Reilly Media, Inc.".

2. Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., & Berners-Lee, T. (1999). RFC2616: Hypertext Transfer Protocol--HTTP/1.1.

3. Snell, J., Tidwell, D., & Kulchenko, P. (2001). Programming web services with SOAP: building distributed applications. " O'Reilly Media, Inc.".

4. Cerami, E. (2002). Web services essentials: distributed applications with XML-RPC, SOAP, UDDI & WSDL. " O'Reilly Media, Inc.".

5. Richardson, L., & Ruby, S. (2008). RESTful web services. " O'Reilly Media, Inc.".

6. Revesz, P. (2010). Introduction to databases. Springer, London, UK.

7. Melton, J., & Simon, A. R. (1993). Understanding the new SQL: a complete guide. Morgan Kaufmann.

8. Momjian, B. (2001). PostgreSQL: introduction and concepts (Vol. 192). New York: Addison-Wesley.

9. Mistry, R., & Misner, S. (2014). Introducing Microsoft SQL Server 2014. Microsoft Press.

10. DuBois, P. (2008). MySQL. Pearson Education.

11. Gackenheimer, C., & Paul, A. (2015). Introduction to React (Vol. 52). Apress.

12. Boduch, A. (2017). React and React Native. Packt Publishing Ltd.

13. Banks, A., & Porcello, E. (2017). Learning React: functional web development with React and Redux. " O'Reilly Media, Inc.".

14. Fedosejev, A. (2015). React. js essentials. Packt Publishing Ltd.

15. Guha, A., Saftoiu, C., & Krishnamurthi, S. (2010, June). The essence of JavaScript. In European conference on Object-oriented programming (pp. 126-150). Springer, Berlin, Heidelberg.

16. Rastogi, A., Swamy, N., Fournet, C., Bierman, G., & Vekris, P. (2015, January). Safe & efficient gradual typing for TypeScript. In Proceedings of the

42Nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (pp. 167-180).

17. Galloway, J., Haack, P., Wilson, B., & Allen, K. S. (2012). Professional ASP. NET MVC 4. John Wiley & Sons.

18. Rai, R. (2013). Socket. IO Real-time Web Application Development. Packt Publishing Ltd.

19. Duckett, J. (2011). HTML & CSS: design and build websites (Vol. 15). Indianapolis, IN: Wiley.

20. Boduch, A. (2019). React Material-UI Cookbook: Build captivating user experiences using React and Material-UI. Packt Publishing Ltd.

21. Cantelon, M., Harter, M., Holowaychuk, T. J., & Rajlich, N. (2014). Node. js in Action (pp. 17-20). Greenwich: Manning.

22. Cadenhead, T. (2015). Socket. IO Cookbook. Packt Publishing Ltd.

23. Satheesh, M., D'mello, B. J., & Krol, J. (2015). Web development with MongoDB and NodeJs. Packt Publishing Ltd.

24. Petrou, M. M., & Petrou, C. (2010). Image processing: the fundamentals. John Wiley & Sons.

25. Boyd, R. (2012). Getting started with OAuth 2.0. " O'Reilly Media, Inc.".

26. Necula, G. C. (1998). Compiling with proofs. Carnegie Mellon University.

27. Abbott, M. L., & Fisher, M. T. (2015). The art of scalability: Scalable web architecture, processes, and organizations for the modern enterprise. Addison-Wesley Professional.

28. Pham, A. D. (2020). Developing back-end of a web application with NestJS framework: Case: Integrify Oy’s student management system.

29. Mäntylä, M. (1987). An introduction to solid modeling. Computer Science Press, Inc..

30. Vernon, V. (2013). Implementing domain-driven design. Addison-Wesley.