МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД
«УЖГОРОДСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ СИСТЕМ

Реєстраційний №_____
Дата _____

КУРСОВА РОБОТА

з дисципліни «Розробка інтернет клієнт-серверних систем»
спеціальність 121 Інженерія програмного забезпечення
Тема: «Розробка сайту для туристичного агентства»

Рекомендована до захисту
“____” _____ 2023р.
Робота захищена
“____” _____ 2023р.
з оцінкою
_____
Підписи членів комісії

Виконав
Студент 3-го курсу
Денної форми навчання
Групи ІПЗ
Молнар Василь Васильович
Науковий керівник :
Викл. Сегін І.М.

Ужгород – 2023

Зміст

ВСТУП

Сучасний світ подорожей та туризму пропонує безліч можливостей для відкриття нових місць, знайомства з різноманітними культурами та насолоди від незабутніх подорожей. З розвитком інтернету та технологій все більше людей вибирають онлайн-сервіси та веб-додатки для планування та бронювання своїх подорожей.

Туристичні агентства, які бажають залишатись конкурентоспроможними на цьому ринку, мають мати сучасний та зручний веб-сайт, який надає користувачам зручність, швидкість та персоналізований досвід. Мета цієї курсової роботи полягає у розробці веб-сайту для туристичного агентства з використанням стеку MERN (MongoDB, Express.js, React.js, Node.js). Цей стек технологій вважається потужним та ефективним для створення сучасних веб-додатків, оскільки він забезпечує повний цикл розробки, включаючи збереження даних, серверну логіку та інтерфейс користувача.

У розділі "Теоретична частина" буде розглянуто поняття туристичного агентства, основні вимоги до веб-сайту туристичного агентства, а також огляд технологій, що використовуються в стеку MERN. Також будуть розглянуті поняття потоку даних, реактивного програмування та переваги використання підходу BLoC (Business Logic Component) для управління станом додатку.

У практичній частині буде розглянуто архітектуру додатку, зв'язок з бек-ендом, процес будування інтерфейсу та реалізацію основних функцій, таких як перегляд туристичних продуктів, створення заявок та обробка деталей продуктів. Для цього будуть використані технології і інструменти, такі як Redux Toolkit для керування станом додатку, RTK Query для виконання запитів до сервера та кешування даних, Formik для роботи з формами, UPT (User Profile Template) для реалізації профілів користувачів та аутентифікації.

1. Теоретична частина

1.1. Огляд стеку MERN

Стек MERN відноситься до сучасних технологій розробки веб-додатків та надає повний набір інструментів для побудови повнофункціональних та масштабованих додатків. Складається він з наступних основних компонентів: MongoDB, Express.js, React.js та Node.js.

1.1.2. Опис технологій MERN

MongoDB: MongoDB є документ-орієнтованою базою даних, яка забезпечує гнучкість та швидкість роботи з даними. Вона використовує NoSQL-підхід до збереження даних у вигляді документів, що дозволяє зберігати різнорідні дані без жорстких схем. MongoDB забезпечує зручний доступ до даних та широкі можливості для маніпуляції та запитів.

Express.js: Express.js є легковаговим фреймворком для побудови серверних додатків на Node.js. Він надає простий та елегантний спосіб створення API та обробки маршрутів на серверній стороні. Express.js дозволяє швидко налаштувати сервер та реалізувати необхідну логіку для обробки запитів.

React.js: React.js є популярною бібліотекою для розробки інтерфейсу користувача. Він використовує компонентний підхід до побудови інтерфейсу, що дозволяє розділити його на незалежні компоненти зі своїм внутрішнім станом та поведінкою. React.js забезпечує швидкий та ефективний рендеринг інтерфейсу, а також простоту управління станом додатку.

Node.js: Node.js є середовищем виконання JavaScript, яке дозволяє запускати код JavaScript на сервері. Воно побудоване на движку V8, що забезпечує швидкість та ефективність виконання. Node.js дозволяє розробникам використовувати JavaScript для розробки серверної логіки, створення API та обробки запитів.

Воно є потужним інструментом для побудови масштабованих та ефективних серверних додатків.

1.1.3. Переваги використання стеку MERN для веб-розробки

Стек MERN пропонує цілий набір інструментів, які доповнюють один одного та спрощують розробку веб-додатків. Використання MongoDB дозволяє зберігати та керувати даними, Express.js забезпечує легку реалізацію серверної логіки та маршрутизації, React.js дозволяє побудувати потужний та динамічний інтерфейс користувача, а Node.js надає середовище для виконання серверного коду. Використання стеку MERN спрощує розробку та підтримку веб-додатків, а також забезпечує швидкість та продуктивність.

1) Єдиноформатний JavaScript: Одним з основних переваг стеку MERN є використання JavaScript як мови програмування на всіх рівнях стеку - від клієнтської сторони (React.js) до серверної (Node.js). Це дозволяє розробникам працювати з однією мовою та повторно використовувати код між фронтендом і бекендом, що спрощує розробку та підтримку додатків.
2) Гнучкість MongoDB: Використання MongoDB як бази даних дозволяє гнучко зберігати та керувати даними. MongoDB використовує документ-орієнтований підхід, що дозволяє зберігати різноманітні дані без жорстких схем. Це особливо корисно для проектів, де схема даних може змінюватися з часом або вимагає гнучкості.
3) Ефективна розробка інтерфейсу: React.js, який є частиною стеку MERN, є потужною бібліотекою для розробки інтерфейсу користувача. Він пропонує компонентний підхід до побудови інтерфейсу, що дозволяє розділити його на незалежні компоненти зі своїм внутрішнім станом та поведінкою. Це спрощує розробку та підтримку інтерфейсу, а також забезпечує швидкий та ефективний рендеринг.

4) Швидкість та масштабованість Node.js: Використання Node.js на серверній стороні дозволяє розробникам виконувати JavaScript-код на сервері. Node.js продовжує використовувати асинхронну та подієву модель програмування, що забезпечує високу продуктивність та швидку відповідь на запити. Це особливо важливо для веб-додатків, які мають велику кількість одночасних запитів та потребують швидкого оброблення даних.
5) Багатофункціональність Redux: Якщо розробка вимагає управління станом додатка, використання Redux в стеку MERN є великим плюсом. Redux є потужною бібліотекою для управління станом додатка, яка дозволяє зберігати, оновлювати та синхронізувати стан додатка на різних компонентах. Це спрощує управління станом, покращує швидкодію та підтримку додатка.

1.2. Redux

Розширена бібліотека Redux в стеку MERN є одним із ключових компонентів, який надає розробникам багатофункціональність та зручність управління станом додатка. Ось деякі переваги використання Redux: Централізоване управління станом: Redux зберігає стан додатка у центральному об'єкті, відомому як "store". Це дозволяє зручно управляти станом додатка та забезпечує однозначність даних у всьому додатку. Ви можете оновлювати стан через диспетчерські дії (actions) і застосовувати зміни до стану через функцію-редуктор (reducer). Простота тестування: Redux робить тестування додатка більш простим. Оскільки стан зберігається у центральному місці, ви можете легко написати тести для дій (actions) та функцій-редукторів (reducers), щоб переконатися, що ваш стан оновлюється та обробляється правильно. Зручність управління складним станом: Коли ваш додаток має складний стан, Redux допомагає управляти цим станом ефективно. Ви можете організувати ваш стан у вигляді дерева, де кожен розділ стану відповідає певній частині додатку. Це полегшує розподіл і обробку стану у вашому додатку.

Легке відновлення стану: Redux надає можливість легкого відновлення стану додатка після перезавантаження сторінки або після помилок. Збереження стану в локальному сховищі дозволяє вам відновити попередній стан додатка при наступному завантаженні.

1.2.1. Redux Toolkit

Redux Toolkit є офіційним набором інструментів для спрощення розробки з використанням Redux. Він надає зручніші API, швидкішу налаштування та вбудовані практики для ефективної роботи зі станом додатка. Ось деякі переваги використання Redux Toolkit:

1) Спрощена настройка: Redux Toolkit пропонує набір готових інструментів та функцій, які допомагають скоротити кількість коду для налаштування Redux. Наприклад, він автоматично генерує reducer та actions на основі вашої конфігурації. Скорочення коду:
2) Redux Toolkit використовує концепцію "slices", які дозволяють організувати код пов'язаних частин стану та дій. Це допомагає зменшити кількість дубльованого коду та полегшує управління станом. Вбудована обробка незмінюваних даних: Redux Toolkit використовує immer.js для автоматичної обробки незмінюваних даних. Це означає, що ви можете змінювати стан безпосередньо, не втрачаючи імутабельність даних. Вбудована підтримка асинхронності:
3) Redux Toolkit має вбудовану підтримку асинхронних дій з використанням redux-thunk або redux-saga. Це дозволяє зручно працювати з асинхронними операціями, такими як отримання даних з сервера.
4) Вбудована підтримка девтулзів: Redux Toolkit автоматично підключає девтулзи для розширення, такі як Redux DevTools, що дозволяють відстежувати та налагоджувати стан додатка в режимі розробки.

5) Інтеграція з React: Redux Toolkit підтримує інтеграцію з React, зокрема з використанням React Hooks.

1.2.2. Redux RTK Query

Redux RTK Query є додатковою бібліотекою, розробленою Redux Toolkit, яка надає спрощений та декларативний підхід до управління запитами і кешування даних в Redux. Ось деякі переваги використання Redux RTK Query:

1) Автоматична генерація запитів та мутацій: Redux RTK Query автоматично генерує функції для виконання запитів до сервера та мутацій. Ви описуєте схему запиту, а бібліотека створює відповідні функції, які додаються до Redux-стору.
2) Кешування даних: Redux RTK Query має вбудовану систему кешування даних, яка автоматично зберігає та оновлює дані, отримані з сервера. Це дозволяє зберігати локальну копію даних та використовувати її для швидкого відгуку на запити.
3) Маніпулювання даними на клієнті: Redux RTK Query дозволяє вам маніпулювати отриманими даними на клієнті, наприклад, фільтрувати, сортувати або групувати їх. Це зменшує навантаження на сервер та дозволяє ефективно працювати з великими обсягами даних.
4) Обробка помилок та статусів: Redux RTK Query надає зручні механізми для обробки помилок та відстеження статусу запитів. Ви можете визначити, як обробляти помилки та як відображати статуси запитів у вашому інтерфейсі користувача. Інтеграція з Redux Toolkit: Redux RTK Query повністю інтегрований з Redux Toolkit і використовує його підходи та API.

5) Ми можете легко комбінувати Redux RTK Query з іншими функціональностями Redux Toolkit, такими як створення срізів (slices), використання Redux DevTools та інші.

1.3. Node.js

Node.js є вільною та відкритою середовищем виконання JavaScript, яке базується на двигуні V8 від Google. Ось деякі переваги використання Node.js:

1) Висока продуктивність: Node.js працює на однопотоковій архітектурі, що дозволяє йому ефективно обробляти багатопотокові завдання. Він використовує неблокуючий (non-blocking) I/O підхід, що дозволяє обробляти багато запитів одночасно без блокування потоків.
2) Розширені можливості серверного програмування: Node.js дозволяє створювати високопродуктивні серверні додатки. Він має вбудовану підтримку мережевих протоколів, таких як HTTP, HTTPS, TCP і UDP, що дозволяє легко створювати веб-сервери, API і інші мережеві додатки.
3) Широке співробітництво з JavaScript: Node.js використовує мову JavaScript, яка є однією з найпопулярніших і широко використовується на різних платформах. Це дозволяє розробникам використовувати свої знання JavaScript для розробки як на фронтенді, так і на бекенді.
4) Велика кількість пакетів і модулів: Node.js має велику екосистему пакетів і модулів, доступних через менеджер пакетів npm. Це дозволяє розробникам швидко використовувати готові рішення і бібліотеки для розробки своїх додатків.
5) Розширені можливості веб-розробки: Node.js дозволяє використовувати JavaScript для розробки як на серверному, так і на клієнтському боці веб-додатків. Він підтримує рендерінг на стороні сервера (SSR) і може використовуватись у комбінації з фреймворками.

1.4. Express.js

Express.js є легковаговим та гнучким веб-фреймворком для Node.js, який дозволяє швидко та ефективно створювати веб-додатки та API. Ось деякі переваги використання Express.js:

1) Простота використання: Express.js пропонує простий та легкий у використанні API, що дозволяє швидко розпочати роботу з ним. Він має мінімальний набір необхідних функцій, але при цьому забезпечує достатню гнучкість для реалізації різноманітних веб-додатків.
2) Middleware: Express.js підтримує middleware-функції, які дозволяють обробляти запити до сервера перед їхньою обробкою. Це дає можливість виконувати різноманітні операції, такі як автентифікація, логування, обробка помилок та багато інших.
3) Роутинг: Express.js надає потужні засоби для організації роутингу веб-додатків. Ви можете визначити маршрути для різних URL-адрес, задати обробники (handlers) для цих маршрутів та здійснювати різні дії в залежності від запитів.
4) Шаблонізація: Express.js підтримує різні шаблонні двіжки (template engines), такі як Pug, EJS, Handlebars, що дозволяють використовувати шаблони для генерації HTML-сторінок. Це спрощує процес розробки і дозволяє створювати динамічний контент на стороні сервера.
5) Розширюваність: Express.js є дуже розширюваним фреймворком. Його можна комбінувати з різними модулями та бібліотеками для реалізації різноманітних функцій.

1.5. MongoDB

MongoDB - це документ-орієнтована база даних, що забезпечує високу продуктивність, гнучкість та масштабованість. Її можна використовувати в різних проектах, від невеликих веб-додатків до великих корпоративних систем. MongoDB працює на різних операційних системах, включаючи Windows, Linux та macOS. Основні переваги MongoDB полягають у її гнучкості та масштабованості. MongoDB дозволяє легко зберігати та опрацьовувати дані будь-якої структури, забезпечуючи швидкий доступ до них.

Крім того, MongoDB може легко масштабуватися з допомогою горизонтального масштабування, що дозволяє збільшувати потужність бази даних за допомогою додавання нових серверів. Використання MongoDB у стеку MERN забезпечує роботу з даними у форматі JSON, що дозволяє просто взаємодіяти з базою даних з Node.js, Express.js та React.js. Крім того, MongoDB має вбудовану підтримку для схем, що дозволяє легко валідувати та обробляти дані на сервері. Загалом, використання MongoDB у стеку MERN дозволяє розробникам швидко та ефективно створювати веб-додатки з високою продуктивністю та масштабованістю.

1.6. React Router

React Router - це бібліотека для навігації та маршрутизації веб-додатків, розроблена для використання з React.js. Вона надає потужні засоби для управління сторінками, URL-адресами та станом додатку в React-додатках. Основні переваги використання React Router в стеку MERN включають: Декларативний підхід: React Router дозволяє визначати маршрути та навігацію у декларативному стилі. Замість визначення складних логік та управління URL-адресами вручну, ви можете використовувати компоненти React для опису сторінок та їхніх маршрутів.

Вбудована маршрутизація: React Router надає потужні механізми для маршрутизації, включаючи можливість визначати динамічні параметри маршрутів, опціональні параметри, вкладені маршрути та багато іншого. Це дозволяє легко створювати складну навігацію та структуру додатків. Історія та навігація: React Router надає доступ до історії браузера, що дозволяє здійснювати навігацію між сторінками за допомогою кнопок "Назад" та "Вперед". Ви також можете використовувати програмне керування навігацією, щоб переходити до конкретних маршрутів після певних подій або дій користувача. Інтеграція з React-екосистемою: React Router добре поєднується з іншими інструментами та бібліотеками React, такими як Redux, форми, анімація та інші. Ви можете легко інтегрувати маршрутизацію вже існуючого додатку та забезпечити консистентну навігацію та стан.

1.7. Mongoose

Mongoose - це об'єктно-документна моделююча бібліотека для Node.js, яка надає простий та зручний спосіб взаємодії з базою даних MongoDB. Вона дозволяє вам визначати схеми для документів, виконувати операції з базою даних та використовувати механізми валідації, хуків та інших функцій, що спрощують роботу з MongoDB. Основні переваги використання Mongoose в стеку MERN включають: Об'єктно-документне відображення: Mongoose дозволяє вам визначати схеми для документів у MongoDB, що спрощує роботу з даними. Ви можете визначати типи полів, валідацію, значення за замовчуванням та багато іншого. Це допомагає створювати структуровані та консистентні дані в базі даних. Валідація та хуки: Mongoose надає можливість визначати правила валідації для полів схеми. Ви можете перевіряти введені дані перед збереженням, виконувати додаткові перевірки та забезпечувати цілісність даних. Крім того, Mongoose підтримує хуки, що дозволяють виконувати додаткові дії перед або після збереження, оновлення або видалення документів.

Запити до бази даних: Mongoose надає потужні засоби для виконання запитів до MongoDB. Ви можете використовувати ланцюжки запитів, шукати документи за певними критеріями, виконувати упорядкування, обмеження та групування результатів. Це дозволяє ефективно взаємодіяти з базою даних та отримувати потрібні результати.

2. Практична частина

2.1. Архітектура додатку

Архітектура додатку - один із важливих аспектів розробки сайту для туристичного агентства. Добре спроектована архітектура забезпечує організацію коду, простоту розширення та підтримку, а також полегшує спільну роботу в команді розробників. При проектуванні архітектури додатку на стеку MERN можна використовувати підхід, відомий як "розділення за розділами" (component-based architecture). Основна ідея полягає у тому, щоб розділити додаток на компоненти, які виконують конкретні функції та мають чітко визначену відповідальність.

1) Кореневий розділ (Root): index.js: Файл, що ініціалізує та запускає додаток. App.js: Основний компонент додатку, що включає головний макет та відповідні маршрути.
2) Розділ маршрутизації (Routing): Routes.js: Файл, що визначає основні маршрути додатку та відповідні компоненти, які будуть відображатися для кожного маршруту.

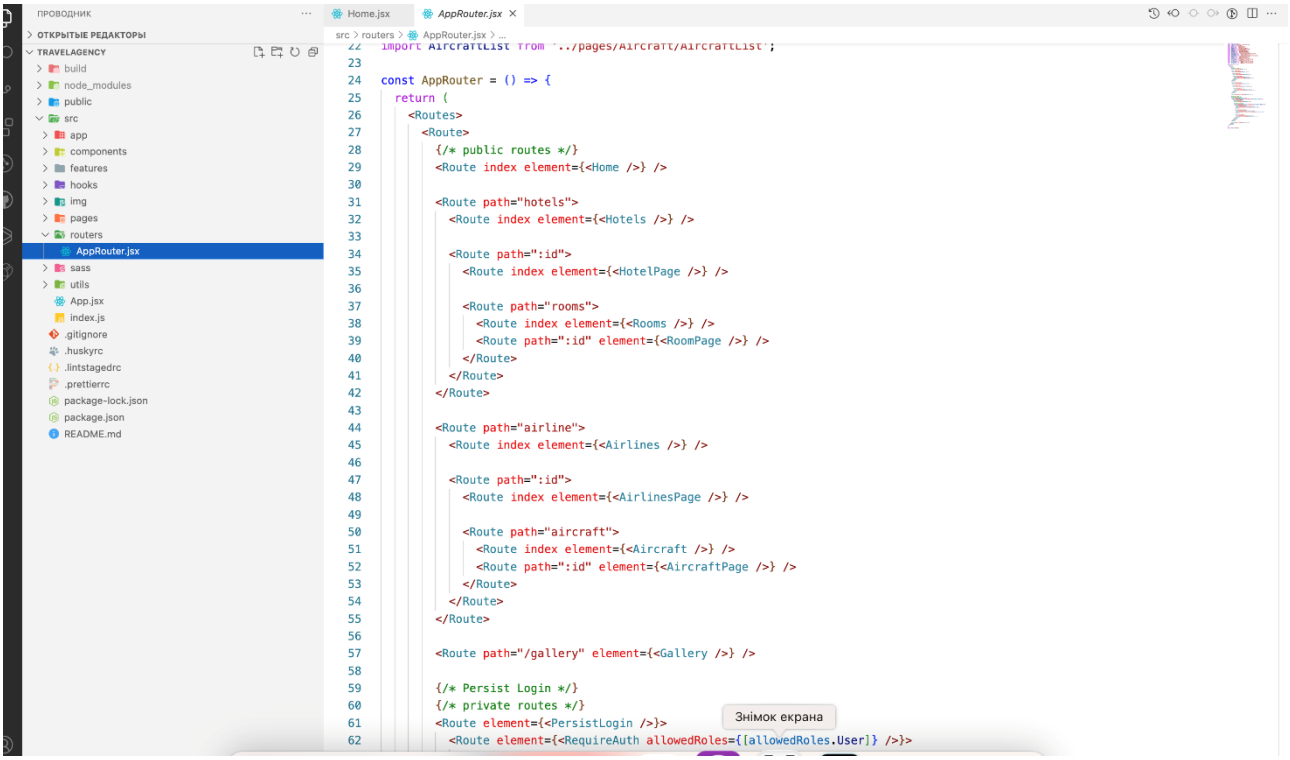

Рис.1

3) Компоненти сторінок (Page Components): Наприклад, HomePage.js, ProductPage.js, BookingPage.js і т. д. Кожен компонент відповідає за відображення конкретної сторінки. Розділ компонентів (Components)

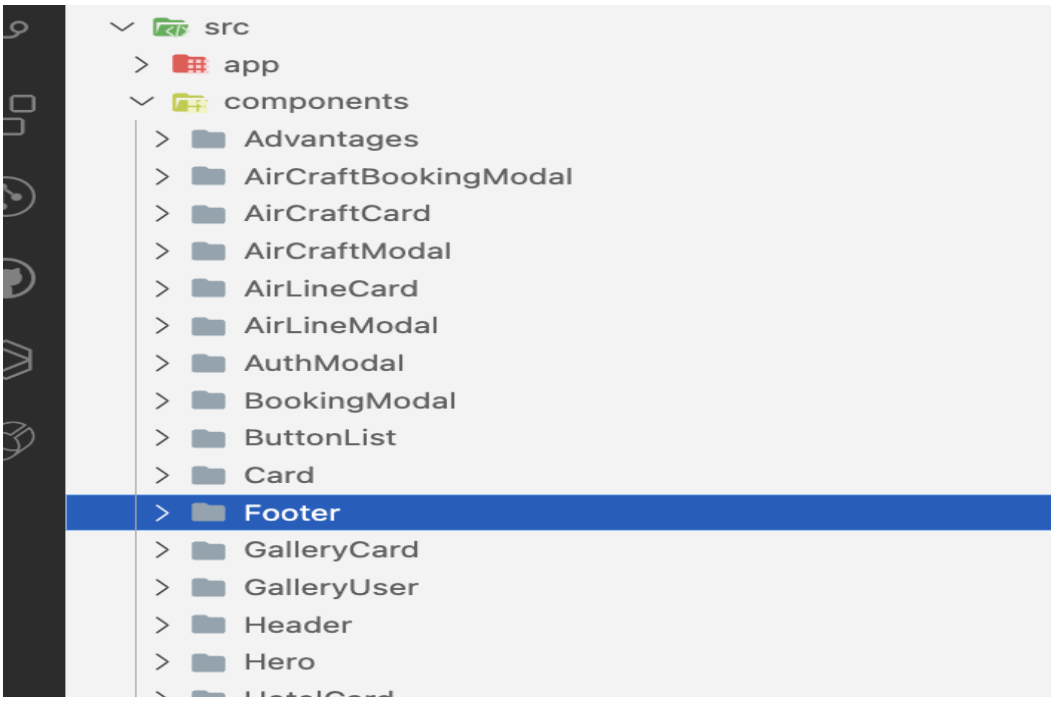

Рис. 2

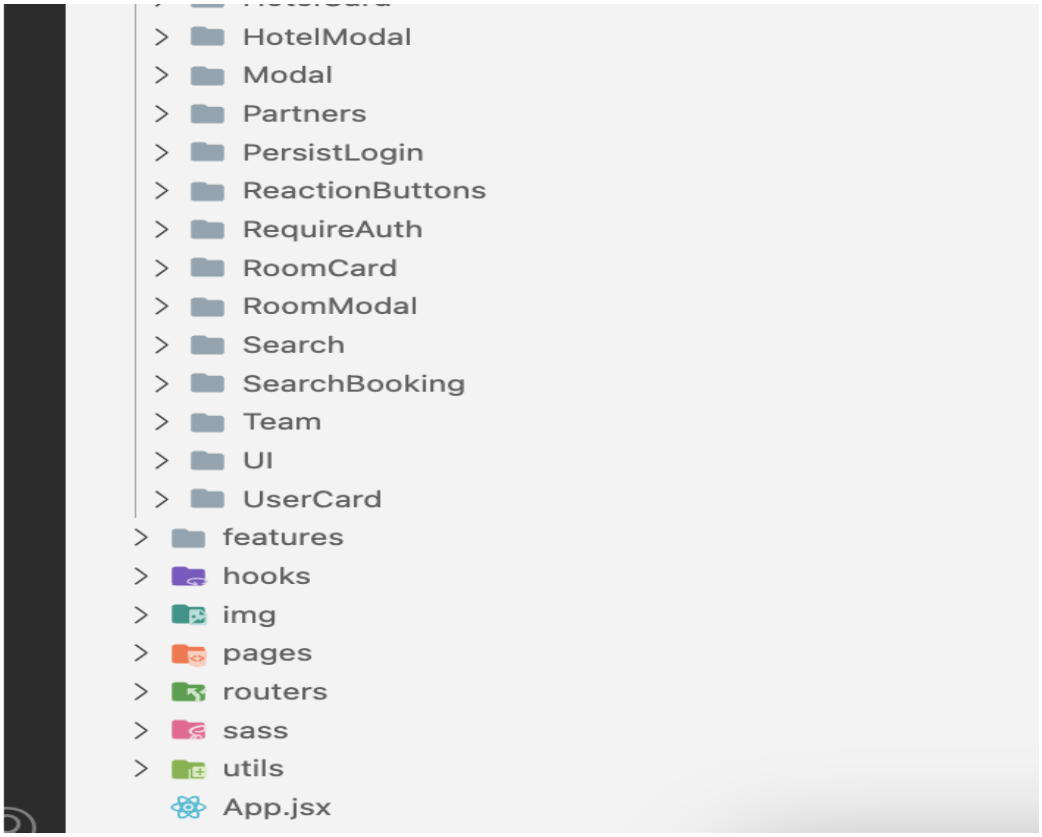

Рис. 3

2.1.1. Планування та структура додатку

Планування та структура додатку є важливими етапами розробки, які допомагають організувати код та забезпечити його ефективну роботу.

1) Визначення вимог: Встановлення основних функціональних вимог до додатку, таких як можливість перегляду та бронювання турів, пошук турів за різними критеріями, керування користувачами та інше.
2) Визначення необхідних технологій та інструментів для реалізації функціональності.
3) Розробка макету та дизайну: Створення макетів та дизайну інтерфейсу додатку, враховуючи вимоги клієнта та потреби користувачів. Використання інструментів для дизайну, таких як Adobe Photoshop, Sketch або Figma, для створення графічних елементів та макетів інтерфейсу.

4) Розбиття на компоненти: Розділення додатку на компоненти, використовуючи підхід розділення за функціональністю. Кожен компонент відповідає за певну частину функціональності або візуальний елемент. Розподіл компонентів на папки та структурування їх за логічними групами.

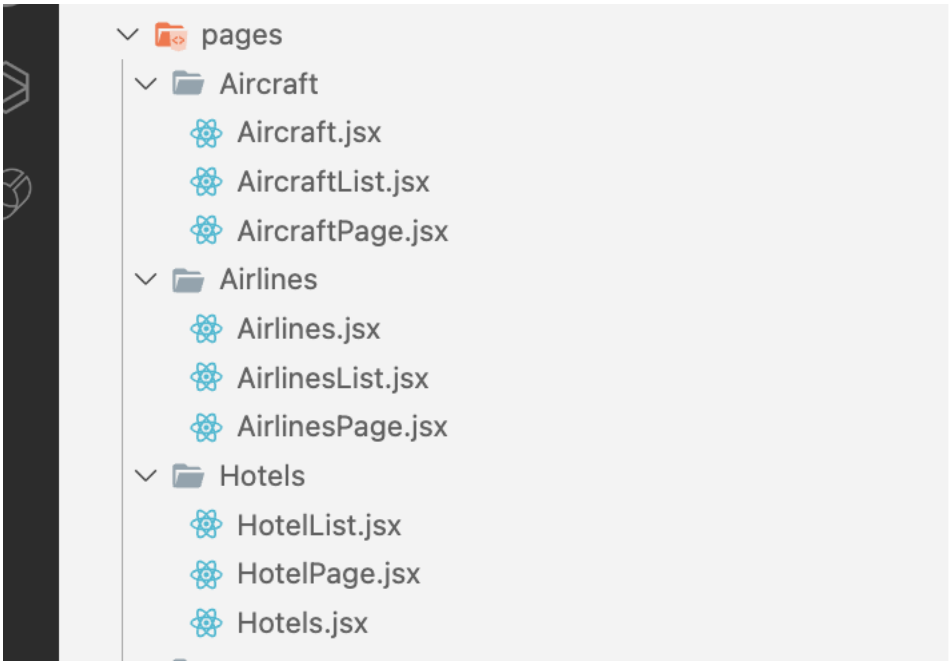

Рис. 4

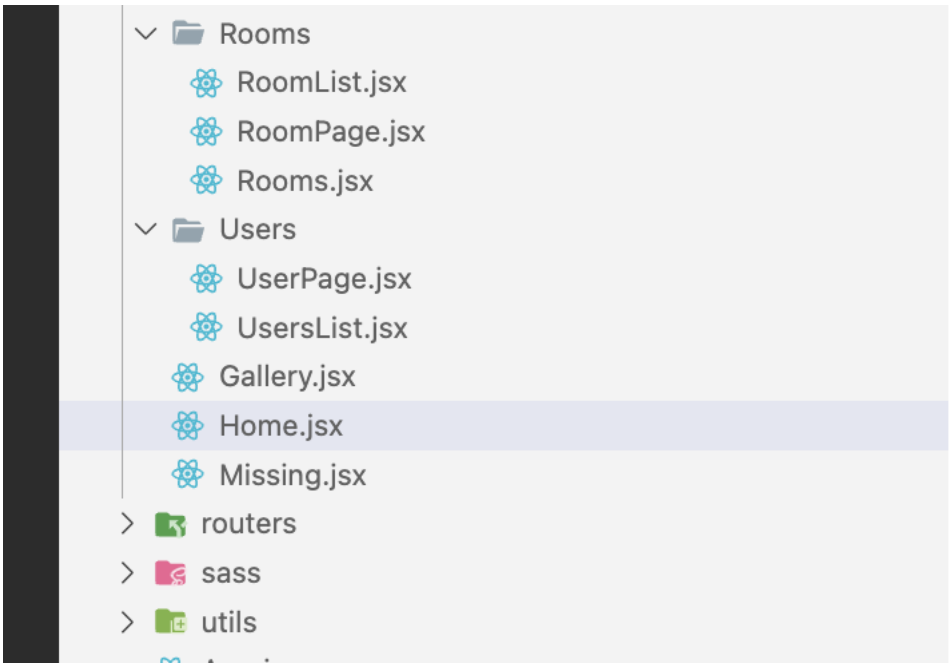

Рис.5

5) Роутинг та навігація: Використання React Router або іншої бібліотеки для створення маршрутизації в додатку. Визначення шляхів (routes) для кожної сторінки або функціональної частини додатку.

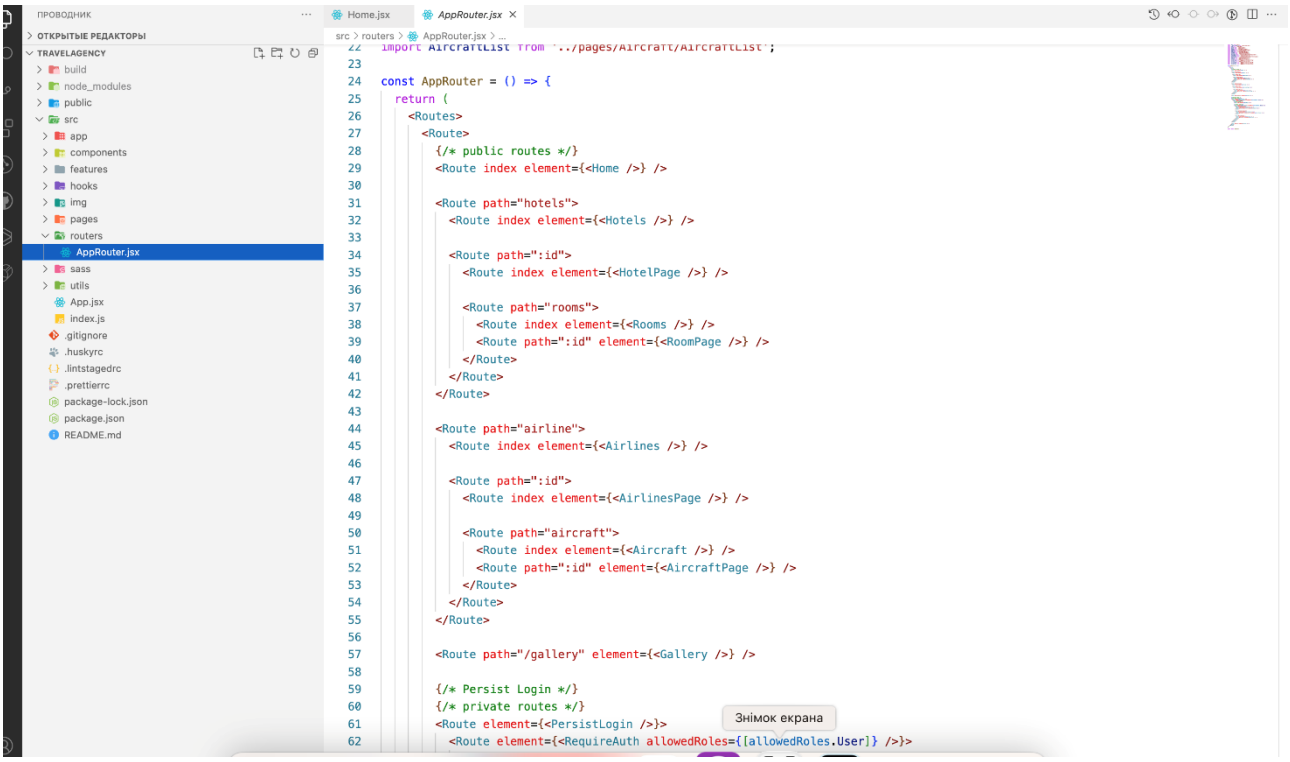

Рис. 6

6) Розробка функціональних компонентів: Створення функціональних компонентів з використанням React Hooks для управління станом та реалізації функціональності.
7) Використання Redux бібліотеки для управління станом додатку.
8) Інтеграція з бекендом: Налаштування комунікації між фронтендом та бекендом, за допомогою REST API .
9) Розробка необхідних API запитів для отримання та збереження даних.
10) Тестування: Розробка тестів для перевірки функціональності додатку. Використання інструментів, таких як Jest або React Testing Library, для автоматизованого тестування компонентів.

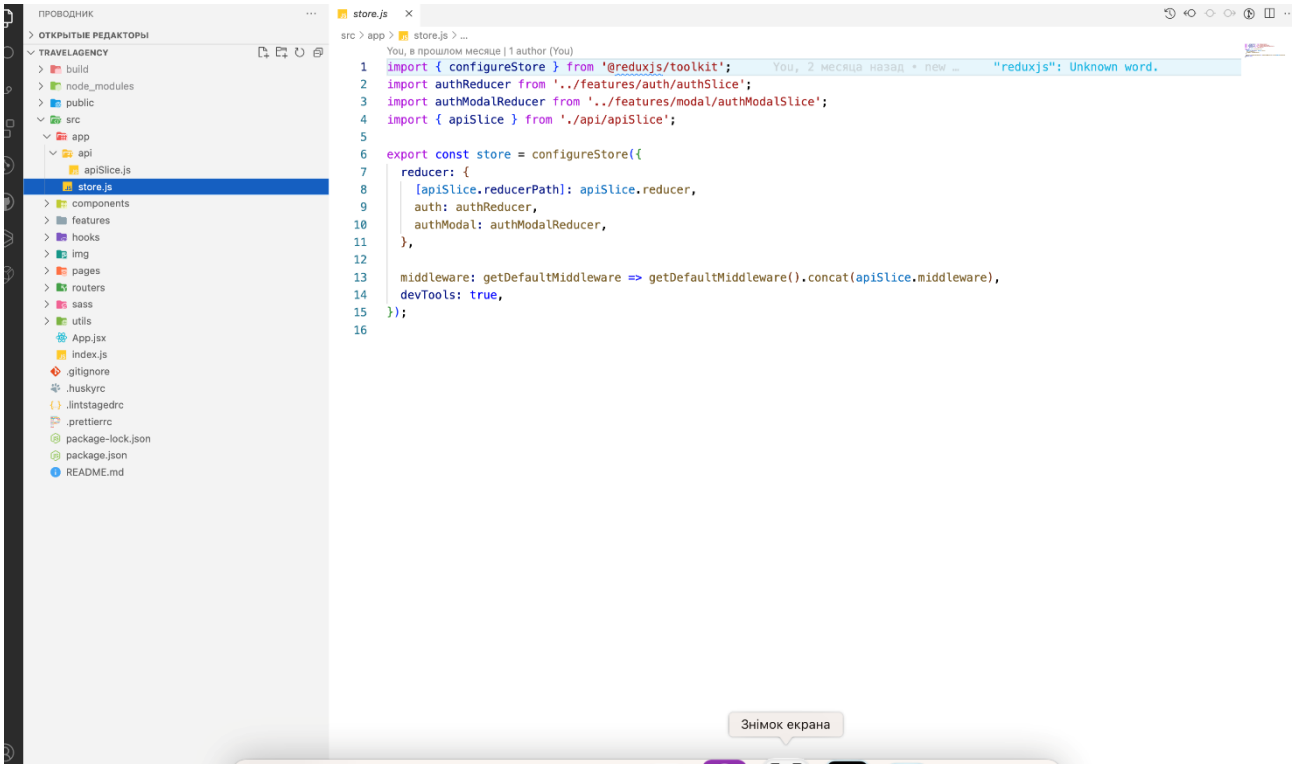
ПРОВОДНИК
ОТКРЫТЫЕ РЕДАКТОРЫ
TRAVELAGENCY
build
node_modules
public
src
app
api
apiSlice.js
store.js
components
features
hooks
img
pages
routers
sass
utils
App.jsx
index.js
.gitignore
.huskyrc
.lintstagedrc
.prettierrc
package-lock.json
package.json
README.md
store.js
src > app > store.js > ...
You, в прошлом месяце | 1 author (You)
import { configureStore } from '@reduxjs/toolkit';
You, 2 месяца назад • new ... "reduxjs": Unknown word.
import authReducer from '../features/auth/authSlice';
import authModalReducer from '../features/modal/authModalSlice';
import { apiSlice } from './api/apiSlice';
export const store = configureStore({
reducer: {
[apiSlice.reducerPath]: apiSlice.reducer,
auth: authReducer,
authModal: authModalReducer,
},
middleware: getDefaultMiddleware => getDefaultMiddleware().concat(apiSlice.middleware),
devTools: true,
});
Знімок екрана

Рис.7

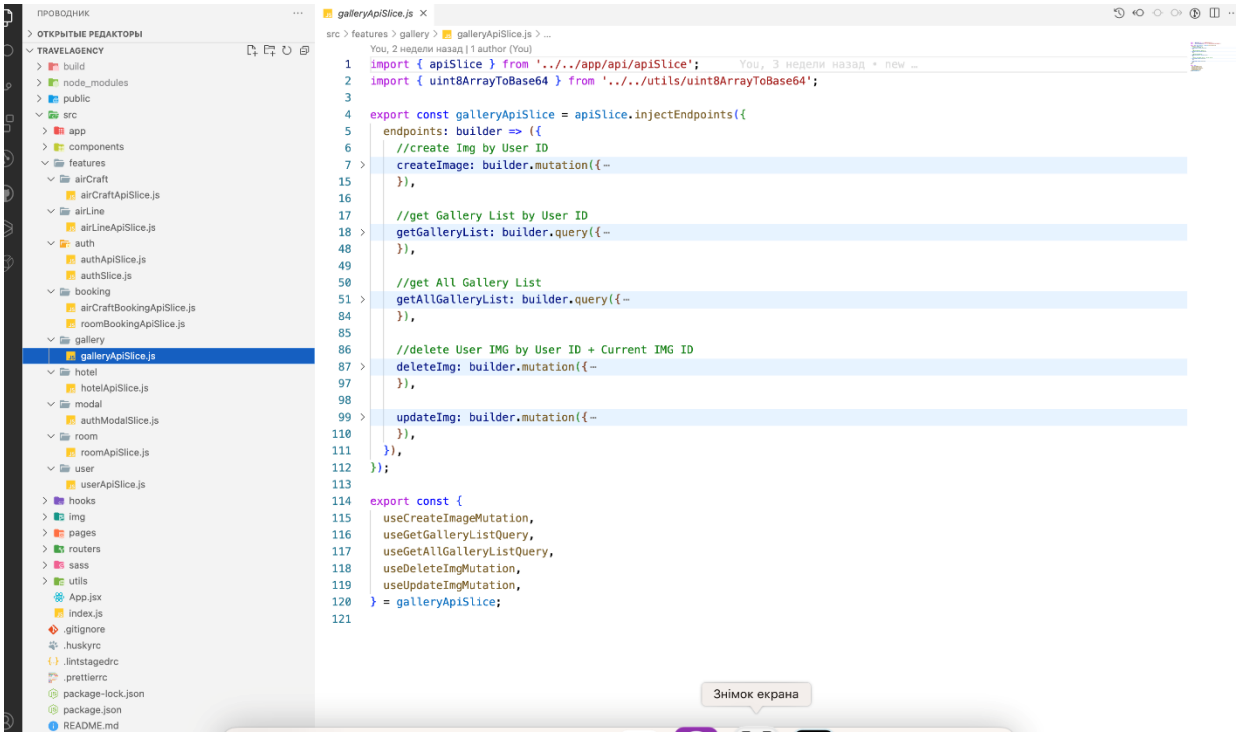
ПРОВОДНИК
ОТКРЫТЫЕ РЕДАКТОРЫ
TRAVELAGENCY
build
node_modules
public
src
app
components
features
airCraft
airCraftApiSlice.js
airLine
airLineApiSlice.js
auth
authApiSlice.js
authSlice.js
booking
airCraftBookingApiSlice.js
roomBookingApiSlice.js
gallery
galleryApiSlice.js
hotel
hotelApiSlice.js
modal
authModalSlice.js
room
roomApiSlice.js
user
userApiSlice.js
hooks
img
pages
routers
sass
utils
App.jsx
index.js
.gitignore
.huskyrc
.lintstagedrc
.prettierrc
package-lock.json
package.json
README.md
galleryApiSlice.js
src > features > gallery > galleryApiSlice.js > ...
You, 2 недели назад | 1 author (You)
import { apiSlice } from '../../app/api/apiSlice';
You, 3 недели назад • new ...
import { uint8ArrayToBase64 } from '../../utils/uint8ArrayToBase64';
export const galleryApiSlice = apiSlice.injectEndpoints({
endpoints: builder => ({
//create Img by User ID
createImage: builder.mutation({...
}),
//get Gallery List by User ID
getGalleryList: builder.query({...
}),
//get All Gallery List
getAllGalleryList: builder.query({...
}),
//delete User IMG by User ID + Current IMG ID
deleteImg: builder.mutation({...
}),
updateImg: builder.mutation({...
}),
}),
});
export const {
useCreateImageMutation,
useGetGalleryListQuery,
useGetAllGalleryListQuery,
useDeleteImgMutation,
useUpdateImgMutation,
} = galleryApiSlice;
Знімок екрана

Рис.8

2.1.2. REST API для комунікації між клієнтом та сервером

REST (Representational State Transfer) є архітектурним стилем, який використовується для комунікації між клієнтом та сервером в розподілених системах. REST API (Application Programming Interface) використовується для взаємодії з сервером шляхом виконання HTTP-запитів, таких як GET, POST, PUT і DELETE. Основні принципи REST API включають:

1) Використання ресурсів (resources): Сервер представляє дані у вигляді ресурсів, наприклад, "тур", "користувач", "бронювання". Кожен ресурс має унікальний ідентифікатор (URI).
2) Використання HTTP-методів: Клієнт виконує HTTP-запити до сервера за допомогою методів, таких як GET (отримання ресурсу), POST (створення ресурсу), PUT (оновлення ресурсу) і DELETE (видалення ресурсу).
3) Використання статус-кодів: Сервер повертає HTTP-статус-коди у відповідь на клієнтські запити, що вказують на успішність або помилку операції.
4) Використання безстановості: Клієнтські запити до сервера не зберігаються на сервері, тобто сервер не залежить від попередніх запитів.
5) Повна інформація про вміст: Сервер повертає повну інформацію про ресурси у відповіді на запити клієнта. Клієнт може використовувати цю інформацію для подальшої роботи з ресурсами.

2.1.3. Розподіл функціональності на рівні клієнта та сервера

У розробці веб-додатків на стеці MERN (MongoDB, Express.js, React, Node.js), функціональність можна розподілити між рівнями клієнта та сервера. Враховуючи цей розподіл, основні завдання можна розподілити таким чином:

Рівень клієнта:

1)Візуальний інтерфейс: Розробка користувацького інтерфейсу з використанням HTML, CSS, SASS, JavaScript.

1) Використання бібліотеки React для створення компонентів і взаємодії з користувачем.
2) Маршрутизація: Використання React Router для налаштування маршрутів та переходів між сторінками додатку.
3) Управління станом: Використання Redux або Redux Toolkit для управління станом додатку та обміну даними між компонентами.
4) Валідація форм: Використання бібліотеки Formik для створення форм і валідації введених даних.
5) Відображення повідомлень: Використання бібліотек, таких як React-Toastify або React-Notifications, для відображення сповіщень, повідомлень про помилки або підтверджень дій.

Рівень сервера:

1) Маршрутизація API: Використання Express.js для створення маршрутів API, які оброблюють запити від клієнта.
2) Обробка запитів: Реалізація обробки запитів на сервері, включаючи перевірку та обробку вхідних даних, виконання логіки та взаємодію з базою даних.
3) Взаємодія з базою даних: Використання бібліотеки Mongoose для взаємодії з базою даних MongoDB, включаючи створення моделей, збереження, оновлення та видалення даних.
4) Автентифікація та авторизація: Реалізація механізмів автентифікації та авторизаці ї, таких як використання JWT (JSON Web Tokens) та збереження сесій на сервері.

Розподіл функціональності між клієнтом та сервером забезпечує високу швидкодію та ефективність додатку. Клієнтська частина відповідає за інтерфейс користувача та взаємодію з ним, тоді як серверна частина забезпечує обробку запитів, взаємодію з базою даних та забезпечення безпеки та автентифікації. Такий розподіл дозволяє створити надійний веб-додаток для туристичного агентства.

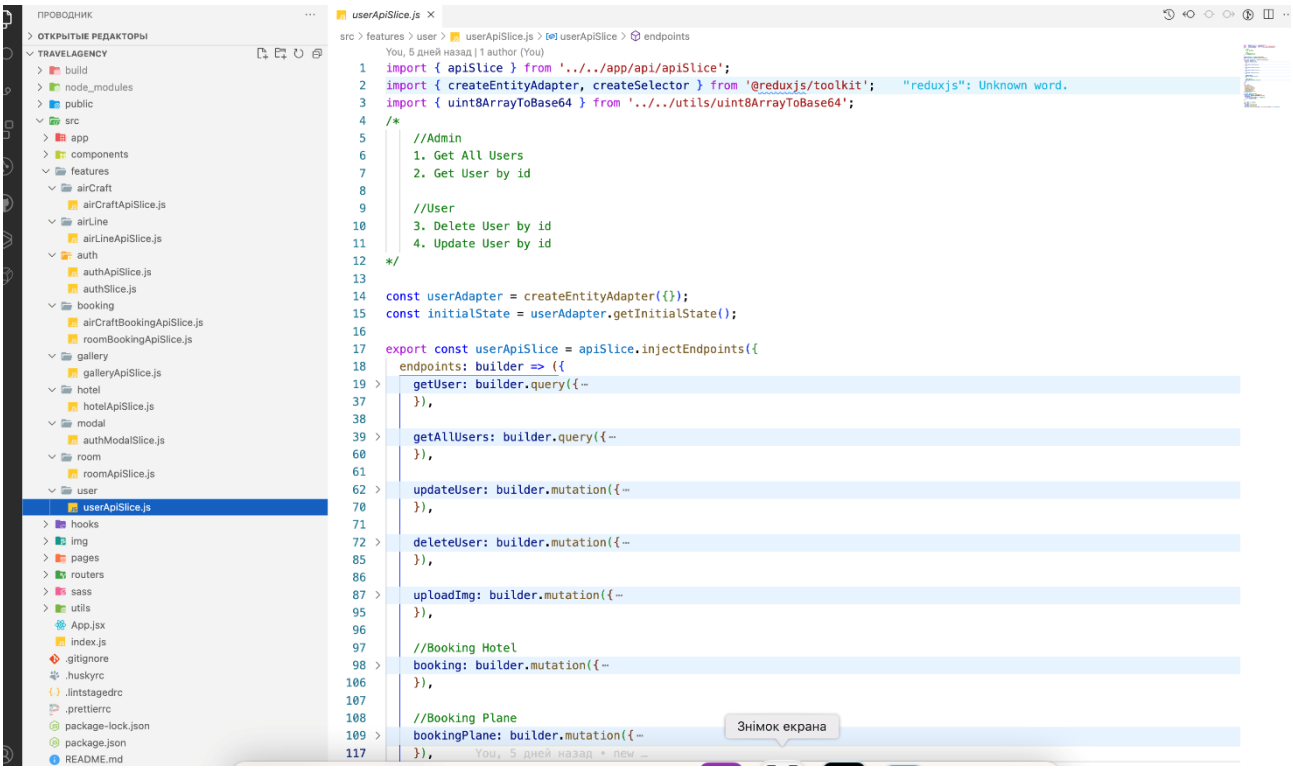

Рис. 9

2.2. Важливі аспекти безпеки

В розробці веб-додатків для туристичних агентств важливо приділяти належну увагу аспектам безпеки, щоб забезпечити захист даних і запобігти можливим загрозам.

1) Аутентифікація та авторизація: Забезпечення механізмів аутентифікації (перевірка ідентичності користувача) та авторизації (керування правами доступу) для обмеження доступу до конфіденційної інформації та

функціональності додатку.

2) Захист від кросс-сайтового скриптінгу (XSS): Забезпечення обробки та очищення вхідних даних, щоб уникнути виконання шкідливих скриптів на боковому скрипті користувача та збереження даних від XSS-атак.
3) Захист від впровадження віддаленого коду (Remote Code Execution, RCE): Уникання виконання небезпечних команд або коду, надання обмежених привілеїв для запуску серверного коду та перевірка вхідних даних перед виконанням. Захист від внутрішнього переламу (Insecure Direct Object References, IDOR): Коректне налаштування доступу до ресурсів, контроль прав доступу до об'єктів та обмеження можливості отримати доступ до конфіденційних даних без належних дозволів.
4) Захист бази даних: Використання захищених методів з'єднання, належний контроль доступу до бази даних, шифрування конфіденційних даних, регулярне оновлення та встановлення запобіжних заходів проти SQL-ін'єкцій та інших типів атак.
5) Захист від міжсайтового поділу ресурсів (Cross-Site Resource Sharing, CSRF): Використання механізмів захисту, таких як токени CSRF, для запобігання несанкціонованому виконанню дій через міжсайтовий поділ ресурсів.
6) Шифрування даних: Використання шифрування для захисту конфіденційних даних, які зберігаються на сервері або передаються через мережу, за допомогою протоколів шифрування, таких як SSL / TLS.
7) Моніторинг та аудит безпеки: Встановлення системи моніторингу безпеки, яка виявляє можливі загрози, реагує на події безпеки та реєструє аудиторські події для подальшого аналізу.
8) Оновлення та патчі: Регулярне оновлення фреймворків, бібліотек, серверного програмного забезпечення та системного програмного забезпечення, щоб усунути виявлені уразливості та використовувати останні версії з виправленими помилками.

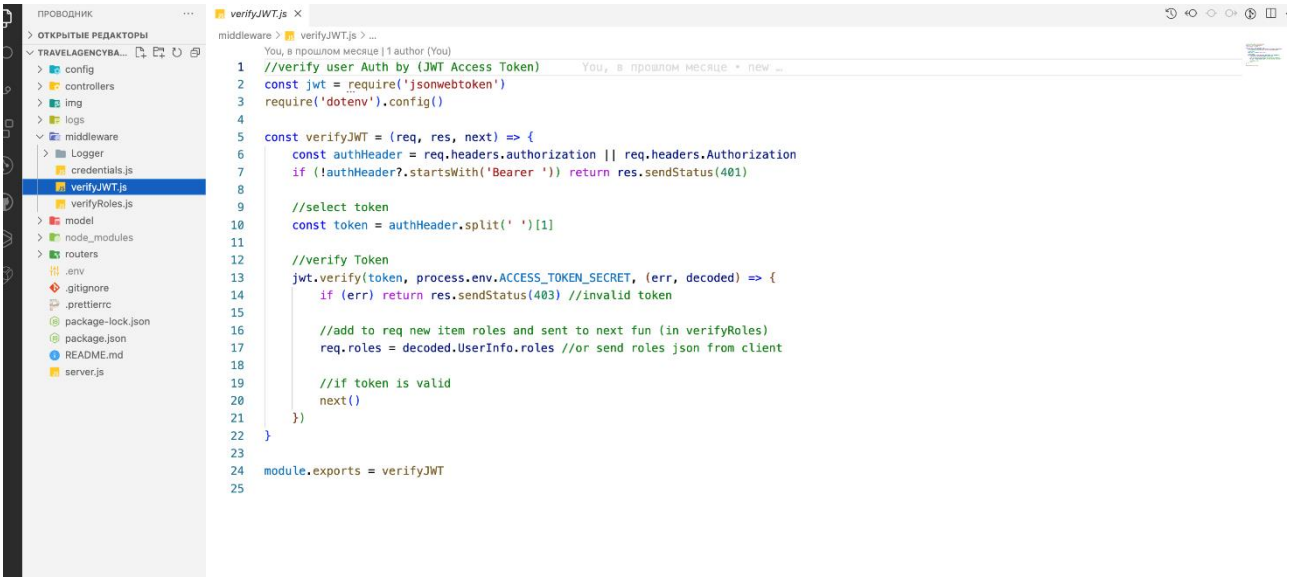
ПРОВОДНИК
ОТКРЫТЫЕ РЕДАКТОРЫ
TRAVELAGENCYBA...
config
controllers
img
logs
middleware
Logger
credentials.js
verifyJWT.js
verifyRoles.js
model
node_modules
routers
.env
.gitignore
.prettierrc
package-lock.json
package.json
README.md
server.js
verifyJWT.js
middleware > verifyJWT.js > ...
You, в прошлом месяце | 1 author (You)
//verify user Auth by (JWT Access Token)
const jwt = require('jsonwebtoken')
require('dotenv').config()
const verifyJWT = (req, res, next) => {
const authHeader = req.headers.authorization || req.headers.Authorization
if (!authHeader?.startsWith('Bearer ')) return res.sendStatus(401)
//select token
const token = authHeader.split(' ')[1]
//verify Token
jwt.verify(token, process.env.ACCESS_TOKEN_SECRET, (err, decoded) => {
if (err) return res.sendStatus(403) //invalid token
//add to req new item roles and sent to next fun (in verifyRoles)
req.roles = decoded.UserInfo.roles //or send roles json from client
//if token is valid
next()
})
}
module.exports = verifyJWT

Рис. 10

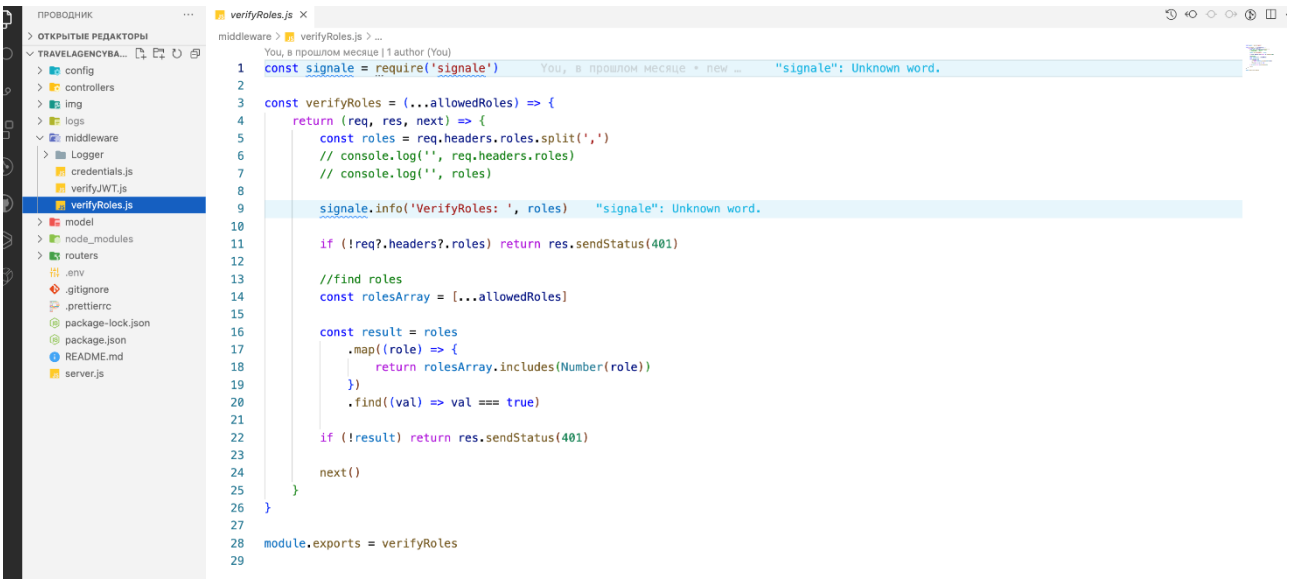
ПРОВОДНИК
ОТКРЫТЫЕ РЕДАКТОРЫ
TRAVELAGENCYBA...
config
controllers
img
logs
middleware
Logger
credentials.js
verifyJWT.js
verifyRoles.js
model
node_modules
routers
.env
.gitignore
.prettierrc
package-lock.json
package.json
README.md
server.js
verifyRoles.js
middleware > verifyRoles.js > ...
You, в прошлом месяце | 1 author (You)
const signale = require('signale')
"signale": Unknown word.
const verifyRoles = (...allowedRoles) => {
return (req, res, next) => {
const roles = req.headers.roles.split(',')
// console.log('', req.headers.roles)
// console.log('', roles)
signale.info('VerifyRoles: ', roles)
"signale": Unknown word.
if (!req?.headers?.roles) return res.sendStatus(401)
//find roles
const rolesArray = [...allowedRoles]
const result = roles
.map((role) => {
return rolesArray.includes(Number(role))
})
.find((val) => val === true)
if (!result) return res.sendStatus(401)
next()
}
}
module.exports = verifyRoles

Рис. 11

2.2.1. Захист конфіденційності даних користувачів

Захист конфіденційності даних користувачів є одним із найважливіших аспектів безпеки веб-додатків для туристичних агентств. Враховуючи регуляторні вимоги і збільшену обізнаність користувачів щодо захисту своїх особистих даних, необхідно вживати належних заходів для забезпечення конфіденційності.

1) Шифрування даних: Важливо шифрувати конфіденційні дані, які зберігаються в базі даних або передаються через мережу.
2) Використання механізмів шифрування, таких як SSL / TLS, забезпечує захист від перехоплення інформації під час передачі даних.

Рис. 12

3) Захищеність паролів: Забезпечення безпеки паролів користувачів шляхом їх хешування та солювання.
4) Використання сильних алгоритмів хешування та зберігання та перевірка паролів забезпечує захист від несанкціонованого доступу до облікових записів користувачів.
5) Управління доступом: Використання правильної системи авторизації та контролю доступу до різних частин додатку.

6) Забезпечення, щоб користувачі отримували доступ лише до необхідної інформації, відповідно до їх ролей та дозволів.
7) Захист від SQL-ін'єкцій: Перевірка та очищення вхідних даних, щоб уникнути SQL-ін'єкцій, які можуть дозволити несанкціоновану доступ до бази даних та витік конфіденційної інформації.

2.2.2. Захист від атак типу (XSS) та (CSRF)

Захист від атак типу XSS (міжсайтового скриптінгу) та CSRF (міжсайтового поділу ресурсів) є важливими аспектами безпеки веб-додатків для туристичних агентств.

1) Валідація та очищення вхідних даних: Перевірка та очищення всіх вхідних даних, що надходять від користувачів, перед виведенням на сторінку. Використання спеціальних функцій або бібліотек для екранування та уникнення внедрення зловмисного JavaScript коду.
2) Використання безпечних HTML шаблонів: Використання шаблонів, які автоматично екранують потенційно небезпечний контент, такі як фреймворки шаблонізації або бібліотеки, що уникнуть виконання вкладеного JavaScript коду.
3) CSRF (міжсайтовий поділ ресурсів): Використання токенів CSRF: Включення токенів CSRF (Cross-Site Request Forgery) до всіх запитів, які змінюють стан сервера. Ці токени генеруються на стороні сервера і передаються разом із запитами.
4) Перевірка правильності токенів на сервері перед виконанням запиту допомагає запобігти атакам CSRF.
5) Оголошення заголовків SameSite та CSRF: Встановлення відповідних значень заголовків SameSite та CSRF у відповідях сервера для обмеження доступу до кукі-файлів тільки для доменів, з яких були встановлені.

2.3. Розробка серверної частини

Розробка серверної частини Розробка серверної частини MERN (MongoDB, Express.js, React, Node.js) включає в себе наступні кроки:

1) Встановлення середовища: Спочатку ми повині переконатись, маємо встановлені Node.js та MongoDB на своєму комп'ютері або сервері.
2) Створення проекту: Створіть папку для вашого проекту та відкрийте її в командному рядку. Запустіть команду npm init для ініціалізації нового проекту Node.js та створення package.json файлу.
3) Встановлення залежностей: Встановіть необхідні пакети для розробки серверної частини. Зазвичай це включає express для створення веб-сервера, mongoose для взаємодії з MongoDB, а також інші пакети, які можуть знадобитись, наприклад, для аутентифікації, валідації тощо.
4) Використовуйте команду npm install для встановлення пакетів.
5) Створення серверу: Створіть файл server.js або index.js, який буде служити основним файлом вашого сервера. У цьому файлі ми повинні підключити необхідні пакети, налаштувати підключення до бази даних MongoDB та створити маршрутизацію для обробки запитів.
6) Маршрутизація: Папка routes і в ній створіть файли для обробки різних маршрутів.
7) Використовуємо Express.js для створення маршрутів, налаштуйте різні HTTP-запити, такі як GET, POST, PUT, DELETE, та обробляйте їх за допомогою відповідних функцій контролерів.
8) Контролери: Створіть папку controllers і в ній створіть файли для різних контролерів. Контролери виконують логіку обробки запитів, взаємодіють з базою даних, валідують дані та відправляють відповідь клієнту.
9) Підключення до бази даних: Використовуємо пакет mongoose для підключення до MongoDB. Створіть підключення в server.js файлі та використовуйте моделі, щоб взаємодіяти з даними у базі даних.

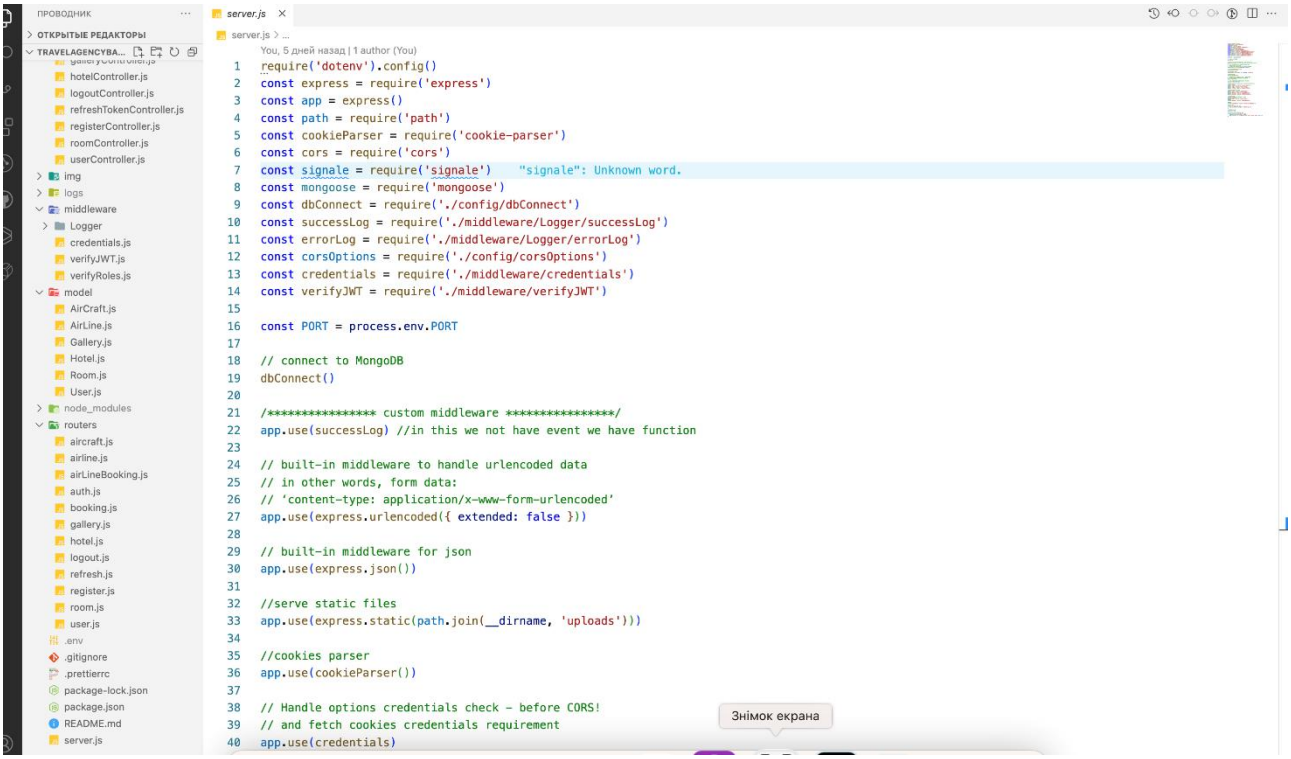

Рис. 13

2.3.1.Реалізація REST API з використанням Express.js та Node.js

1) Планування: Першим кроком було створення плану розробки REST API. Були визначені основні функціональні вимоги, такі як операції CRUD (створення, отримання, оновлення, видалення) для різних ресурсів. Також були визначені необхідні маршрути та параметри запитів для кожної операції.

2) Налаштування проекту: Створено нову папку проекту і встановлені необхідні залежності за допомогою пакетного менеджера npm. Були встановлені пакети Express.js та body-parser.

3)Створення сервера: Було створено файл server.js для ініціалізації сервера. За допомогою пакету Express.js було створено екземпляр сервера та встановлено порт для прослуховування запитів.

4)Реалізація маршрутів: Визначено маршрути для REST API. Для кожного маршруту було використано відповідний HTTP метод та шлях. Наприклад, для отримання списку користувачів було використано метод GET та маршрут /api/users.

5)Реалізація логіки: Для кожного маршруту була реалізована відповідна логіка обробки запитів. Це включало отримання даних від клієнта, взаємодію з базою даних, валідацію даних та відправку відповіді назад до клієнта.

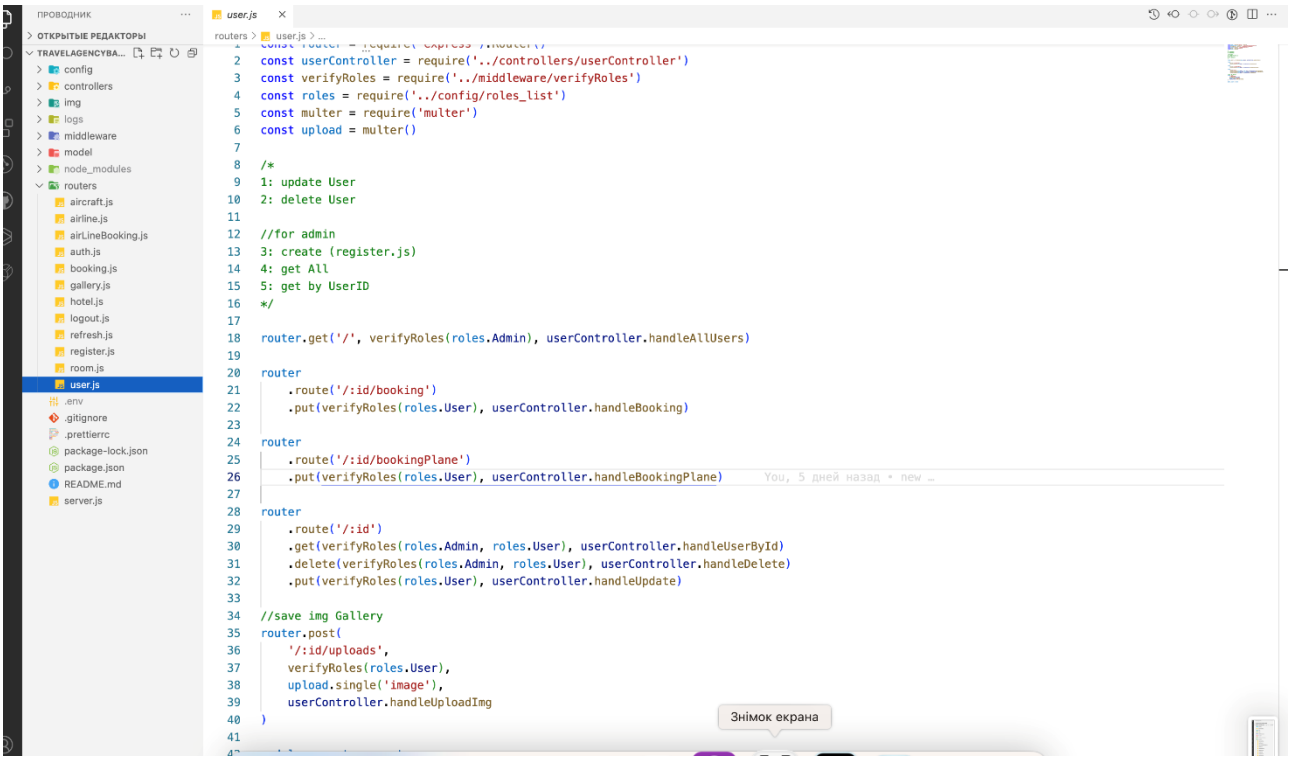

Рис. 14

2.3.2.Аутентифікація та авторизація користувачів

За допомогою механізмів аутентифікації та авторизації, ми забезпечуємо безпеку доступу до ресурсів та функціональності системи.

1) Планування: Перед початком реалізації аутентифікації та авторизації, було проведено аналіз вимог і визначено потреби системи.

2) Були визначені ролі користувачів, доступ до ресурсів, необхідність захисту конфіденційності даних та інші функціональні вимоги.
3) Реалізація аутентифікації: Було використано пакет Passport.js для реалізації аутентифікації користувачів.

   Було налаштовано стратегію аутентифікації, яка використовує локальну стратегію з використанням логіну та пароля користувача. При успішній аутентифікації, генерується JWT (JSON Web Token), який використовується для подальшої авторизації користувача.
4) Реалізація авторизації: Для реалізації авторизації користувачів було використано рольову модель доступу. Кожному користувачеві присвоюється роль з визначеними правами доступу.

   При кожному запиті до ресурсів, авторизація перевіряє, чи має користувач необхідні права доступу до запитуваного ресурсу.
5) Захист конфіденційності: Для захисту конфіденційності даних користувачів, паролі не зберігаються відкритим текстом в базі даних.
6) Тестування: Було проведено тестування аутентифікації та авторизації, включаючи перевірку ролей та прав доступу користувачів.

   Було впевненося, що система працює коректно і надає доступ до ресурсів тільки авторизованим користувачам з відповідними правами.

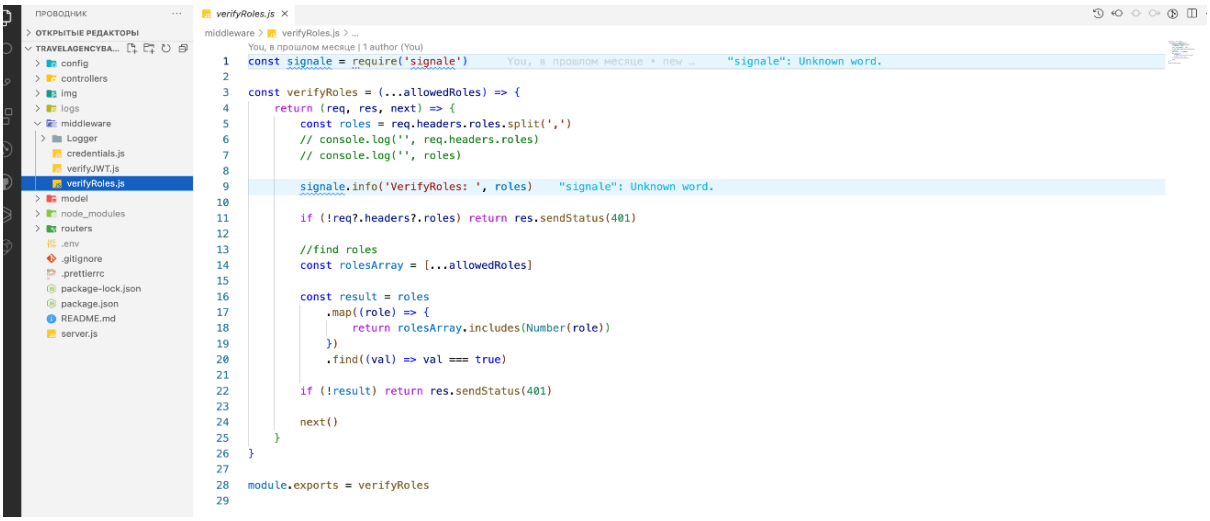

Рис. 15

2.3.3.Реалізація логіки обробки запитів та взаємодії з базою даних

1) Планування: Перед початком розробки логіки обробки запитів та взаємодії з базою даних, було проведено аналіз вимог і визначено потреби системи.

2) Були визначені типи запитів, структура бази даних та необхідність валідації та перевірки даних.

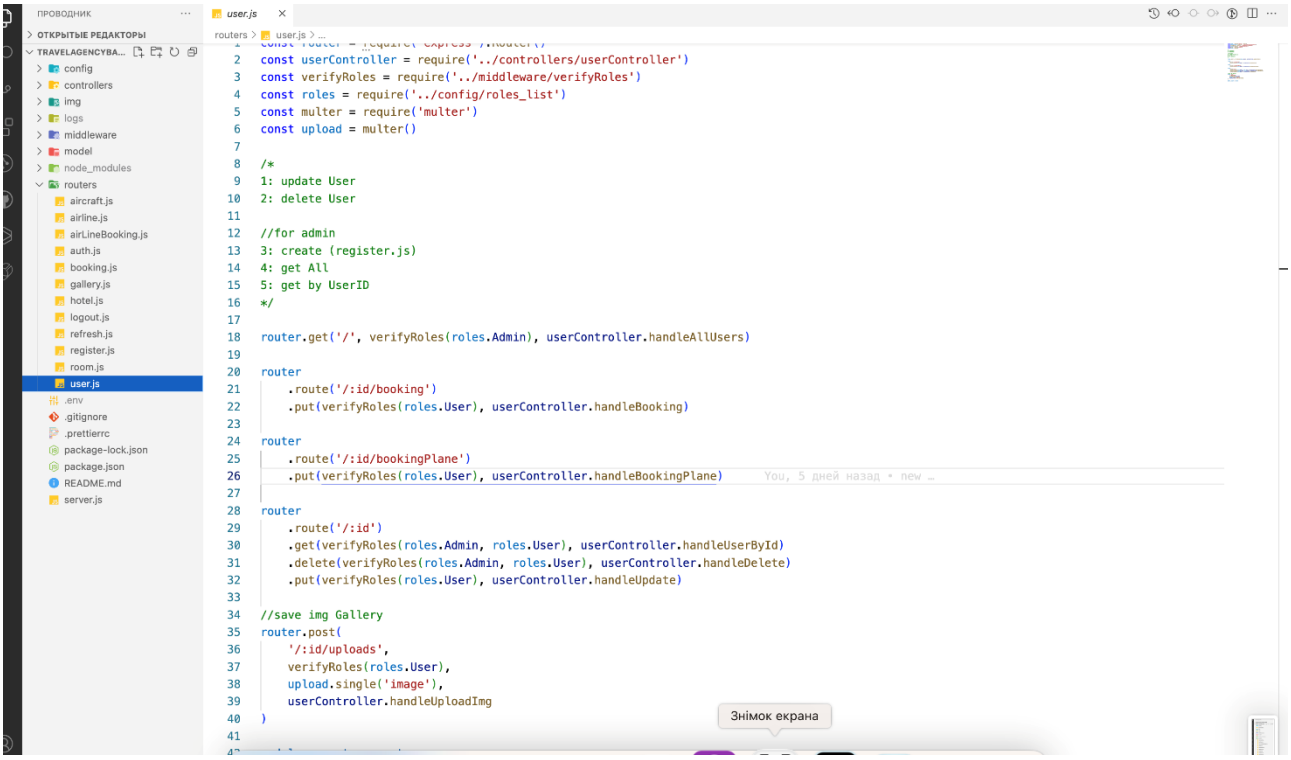

Рис.16

3) Реалізація логіки обробки запитів: За допомогою Express.js було реалізовано серверну частину додатку, яка обробляє запити від клієнта. Було визначено маршрути та контролери для кожного типу запиту.

3) Контролери містять логіку обробки запитів, включаючи валідацію даних, взаємодію з базою даних та надання відповідей клієнту.

4) Взаємодія з базою даних: Для взаємодії з базою даних MongoDB було використано Mongoose - об'єктно-реляційний відображувач даних.

Була розроблена схема даних, яка відображає структуру колекції у базі даних. За допомогою моделей Mongoose, було реалізовано функції для збереження, оновлення, видалення та отримання даних з бази.

5) Валідація та перевірка даних: Було реалізовано валідацію та перевірку даних перед збереженням у базу даних.

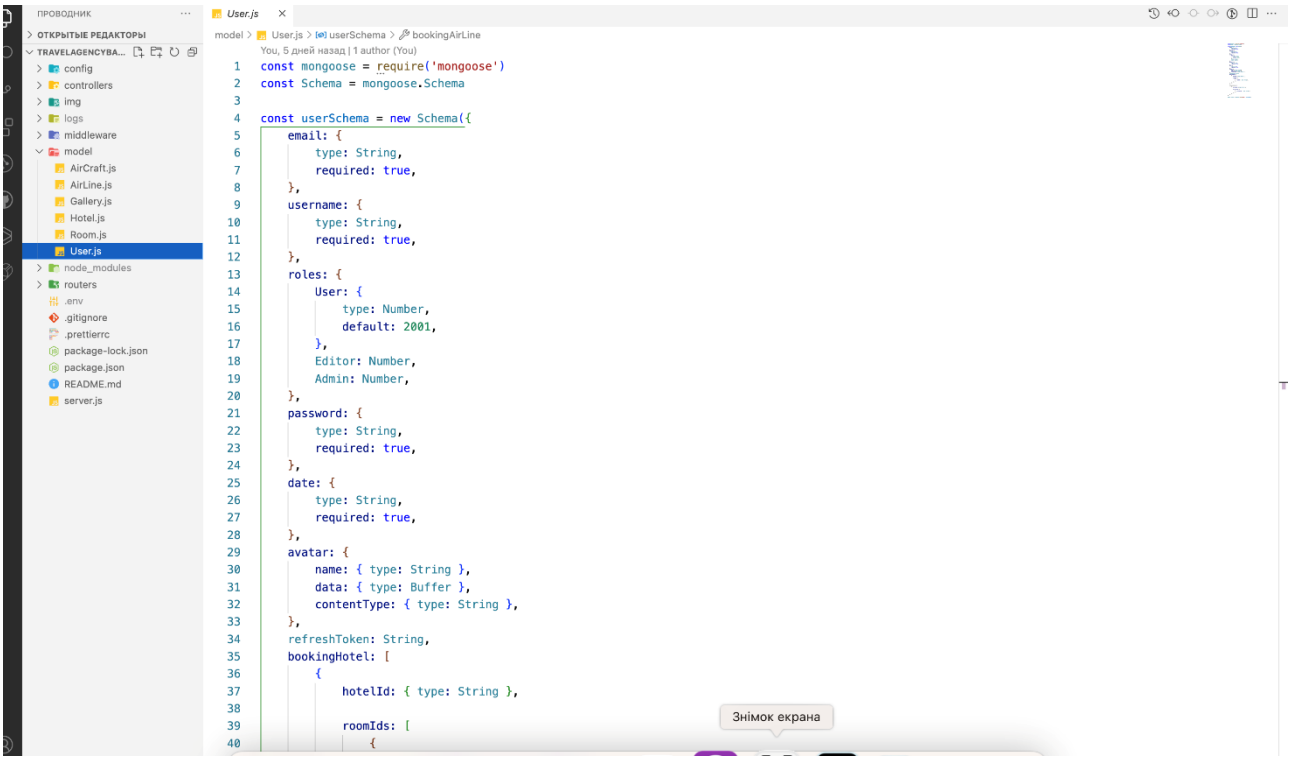

Рис.17

2.4. Розробка клієнтської частини

Створення інтерфейсу користувача, реалізацію компонентів, обробку подій та взаємодію з серверною частиною за допомогою REST API.

1) Планування: Перед початком розробки клієнтської частини було проведено аналіз вимог і визначено потреби системи. Були визначені основні функціональність, структура і дизайн інтерфейсу користувача.

2) Реалізація інтерфейсу користувача: За допомогою React було створено клієнтську частину додатку. Була розроблена структура компонентів, включаючи головний компонент додатку, сторінки, компоненти для відображення даних та форми для взаємодії з користувачем.
3) Для стилізації інтерфейсу використовувався CSS, SASS та CSS-препроцесор.
4) Обробка подій та взаємодія з сервером: За допомогою Redux, було реалізовано обробку подій та зберігання стану додатку. Компоненти були підключені до Redux Store, що дозволило передавати дані та керувати станом додатку.
5) За допомогою REST API, реалізованого на сервері, була забезпечена взаємодія з сервером, отримання та відправка даних.

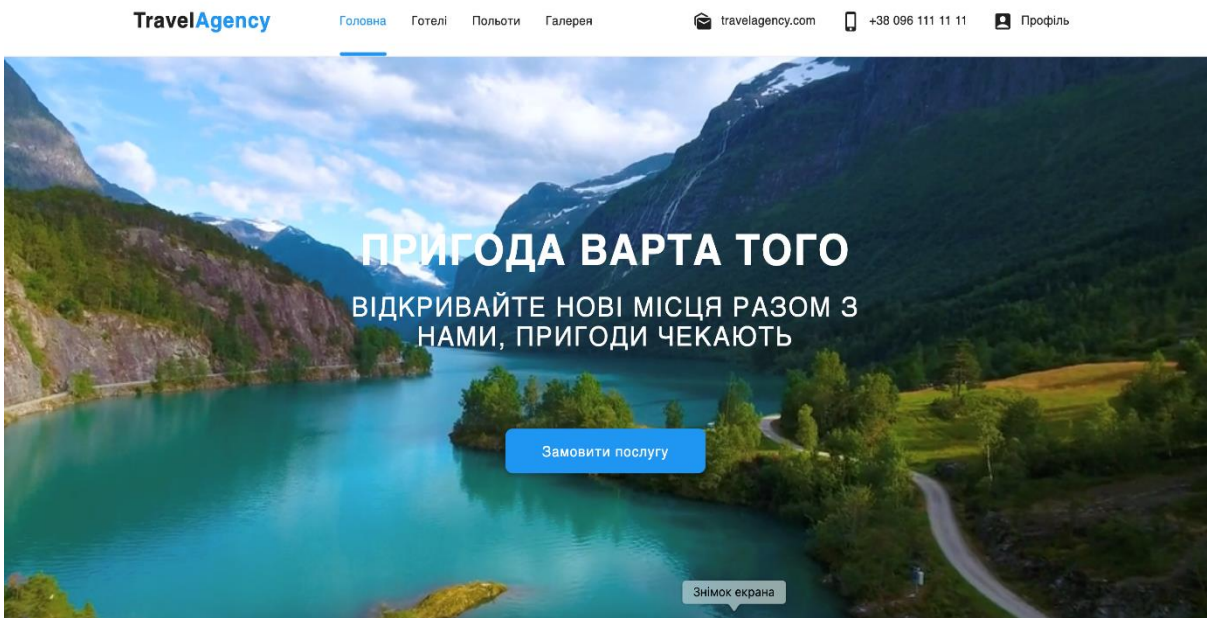

Рис.18

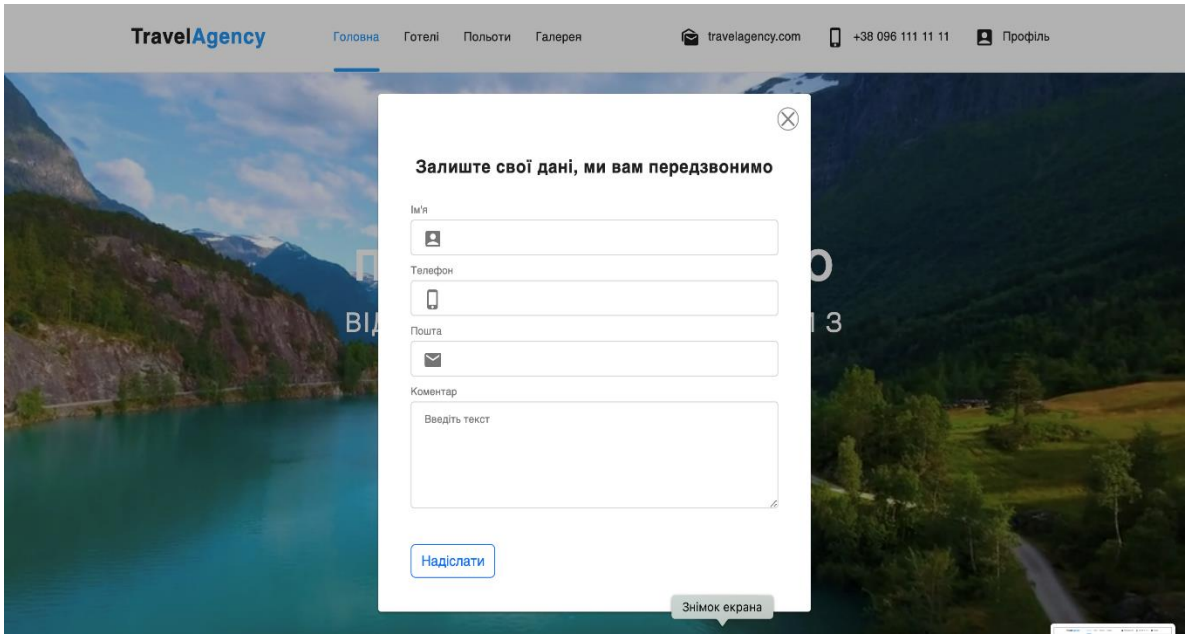

Рис.19

6) У процесі розробки клієнтської частини було використано RTQ (Redux Toolkit Query) для забезпечення ефективної обробки запитів до сервера та взаємодії з ним. RTQ є надбудовою над Redux Toolkit і надає зручний інтерфейс для виконання запитів до сервера, кешування результатів та автоматичного оновлення даних. Для взаємодії з сервером було використано HTTP методи, такі як GET, POST, PUT та DELETE, що відповідають стандартним протоколам REST API. Кожен запит виконувався з використанням відповідного URL, який містить шлях до ресурсу на сервері. RTQ надає зручну можливість визначення endpoint-ів для кожного типу запиту та обробки результатів запиту. Запити були реалізовані з використанням функцій RTQ, які приймають необхідні параметри, такі як URL, тіло запиту та налаштування. Результати запитів були оброблені за допомогою локального стейта Redux, що дозволило зберігати дані та оновлювати їх без необхідності повторного відправлення запиту до сервера.

7) Тестування: Було проведено тестування клієнтської частини для перевірки правильності роботи компонентів, обробки подій та взаємодії з сервером. Використовувалися тести одиниць (Unit tests) та тести інтеграції (Integration tests), щоб переконатися у коректному функціонуванні додатку.

2.4.1.Реалізація інтерфейсу користувача з використанням React.js

Під час розробки клієнтської частини проекту було використано бібліотеку React.js для побудови інтерфейсу користувача. React.js є потужним інструментом для створення компонентної веб-розробки, що дозволяє створювати перевикористовувані та динамічні компоненти. За допомогою React.js було розроблено набір компонентів, які відображали різні частини інтерфейсу користувача. Компоненти були структуровані і організовані таким чином, щоб забезпечити зручну навігацію та взаємодію користувача з додатком. Кожен компонент мав свою логіку та стан, що дозволяло оновлювати компоненти при зміні даних або відповідних подіях. Використовувалися стандартні методи React.js, такі як useState, useEffect, useContext та інші, для керування станом компонентів та виконання певних дій при зміні стану.

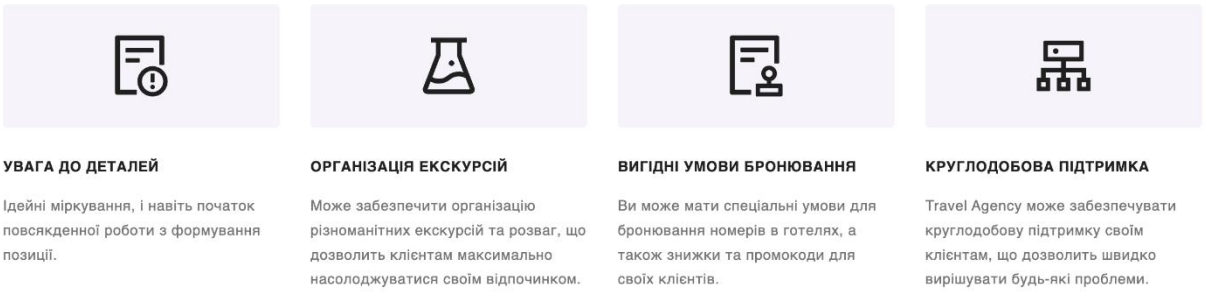

Рис.**20**

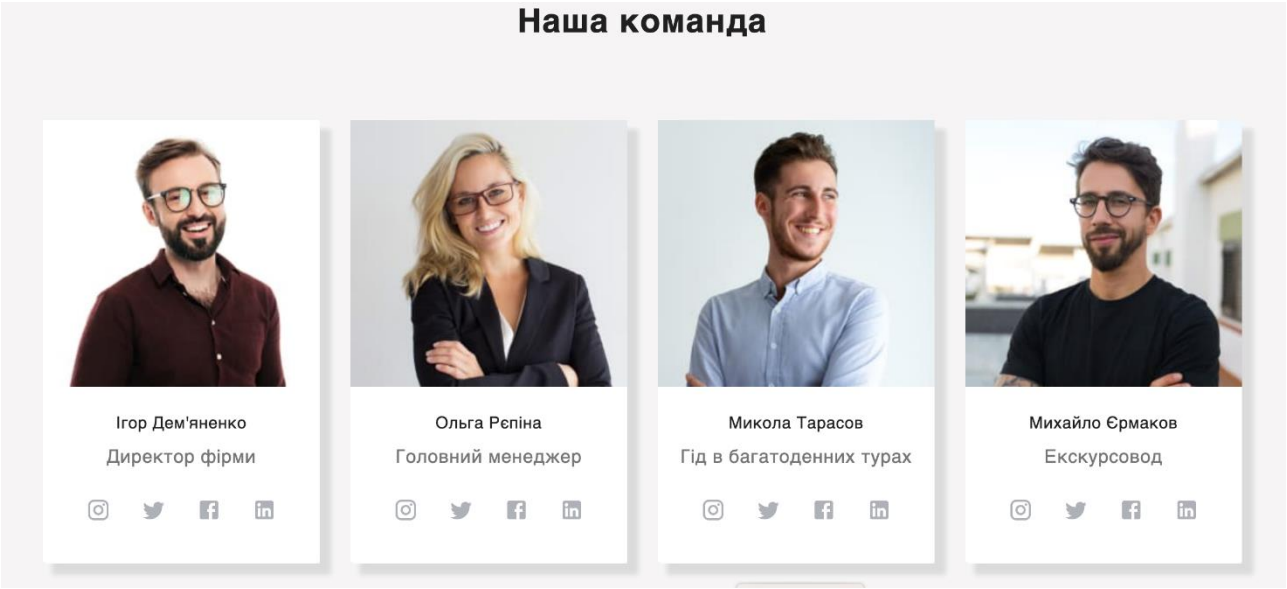
Наша команда
Ігор Дем'яненко
Директор фірми
Ольга Рєпіна
Головний менеджер
Микола Тарасов
Гід в багатоденних турах
Михайло Єрмаков
Екскурсовод

Рис.21

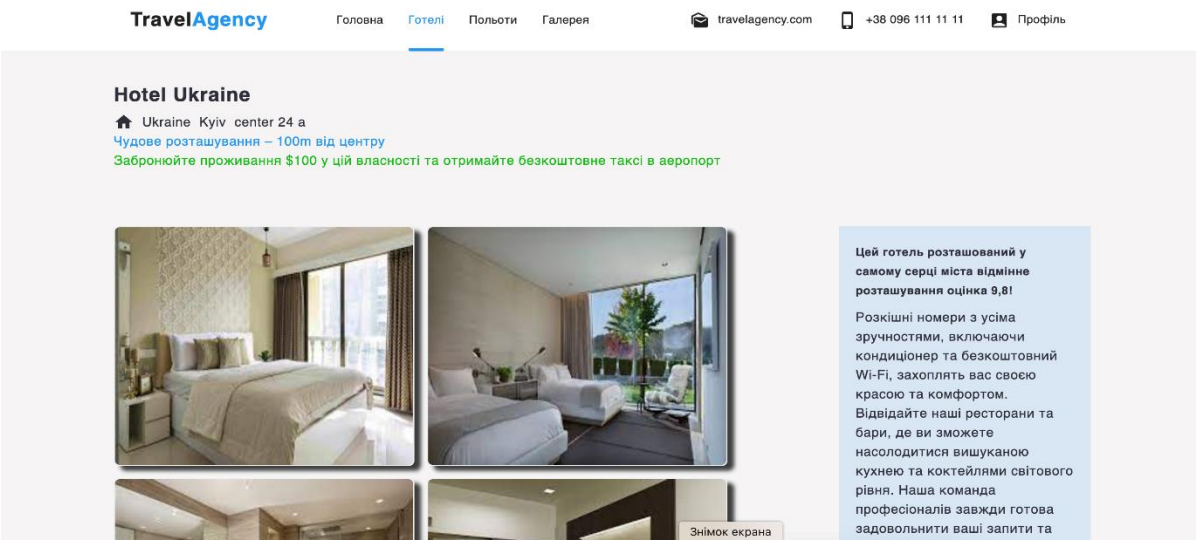
TravelAgency
Головна
Готелі
Польоти
Галерея
travelagency.com
+38 096 111 11 11
Профіль
Hotel Ukraine
Ukraine Kyiv center 24 a
Чудове розташування – 100m від центру
Забронюйте проживання $100 у цій власності та отримайте безкоштовне таксі в аеропорт
Цей готель розташований у самому серці міста відмінне розташування оцінка 9,8!
Розкішні номери з усіма зручностями, включаючи кондиціонер та безкоштовний Wi-Fi, захоплять вас своєю красою та комфортом. Відвідайте наші ресторани та бари, де ви зможете насолодитися вишуканою кухнею та коктейлями світового рівня. Наша команда професіоналів завжди готова задовольнити ваші запити та
Знімок екрана

Рис.22

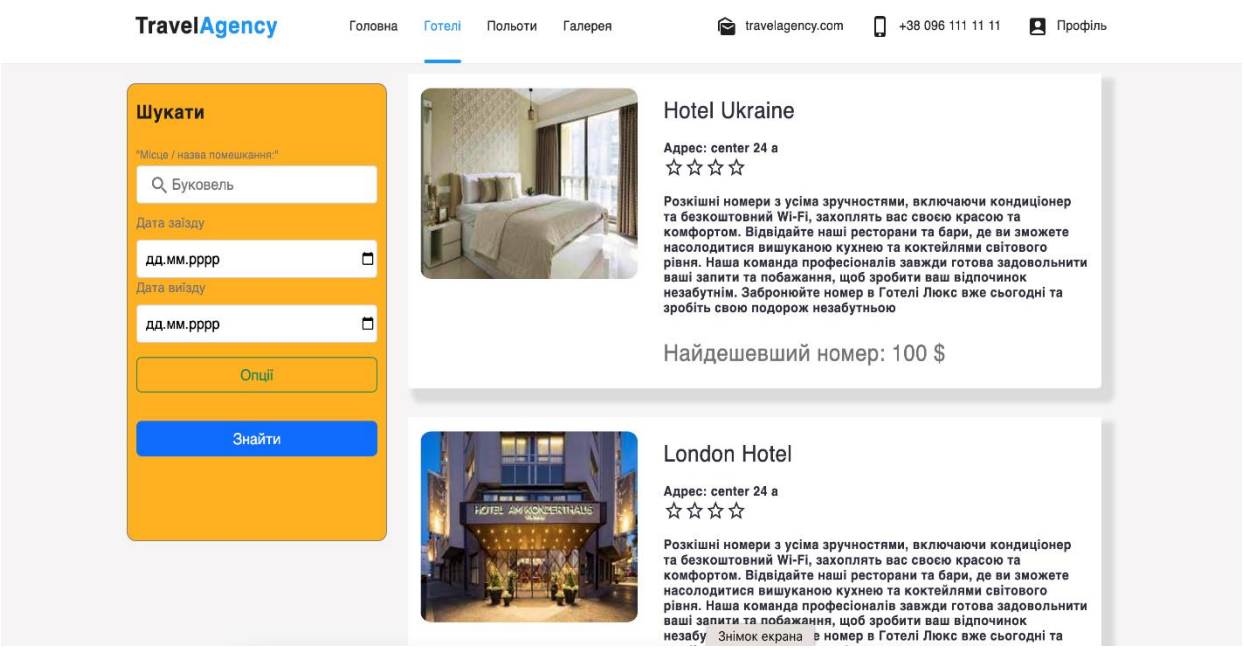

Рис.23

Також були використані бібліотеки Redux та Redux Toolkit для керування глобальним станом додатку. Redux дозволяє зберігати та оновлювати дані у централізованому стані, що полегшує обмін даними між компонентами та забезпечує єдину правду даних. Загальний інтерфейс користувача був створений за допомогою HTML та CSS, з використанням CSS-фреймворків, таких як Bootstrap або Material-UI, для швидкого та стилізованого оформлення компонентів. Результатом роботи була розроблена клієнтська частина з використанням React.js, яка забезпечує інтерактивний та зручний інтерфейс користувача, а також взаємодію з сервером для отримання та оновлення даних.

2.4.2. Компонентна архітектура та управління станом Redux

Під час розробки клієнтської частини проекту було використано компонентну архітектуру та підхід управління станом за допомогою бібліотеки Redux. Цей підхід дозволяє ефективно керувати станом додатку та забезпечує простоту управління та спільне використання даних між компонентами.

Компонентна архітектура розбиває інтерфейс користувача на невеликі, перевикористовувані компоненти, кожен з яких відповідає за свою частину інтерфейсу. Це сприяє зручній розробці, підтримці та тестуванню коду, а також полегшує реактивну оновлення інтерфейсу при зміні стану. Redux використовується для централізованого управління станом додатку. Стан зберігається в одному об'єкті, відомому як "store". Компоненти можуть читати дані зі стану та оновлювати його за допомогою дій, які надсилаються до "store". Це забезпечує єдину правду даних та простоту управління станом. Redux Toolkit є розширенням Redux, яке надає простий та зручний інтерфейс для роботи з Redux. Він включає в себе набір утиліт та шаблонів, таких як "createSlice" та "createAsyncThunk", які спрощують створення редюсерів та взаємодію з асинхронними операціями. З використанням Redux Toolkit, розробка компонентів з підтримкою Redux стає простою і зручною. Компоненти можуть підписуватися на певну частину стану та реагувати на зміни цих даних безпосередньо. Також можна легко виконуват и асинхронні операції, такі як запити до сервера, за допомогою Redux Thunk або Redux Saga.

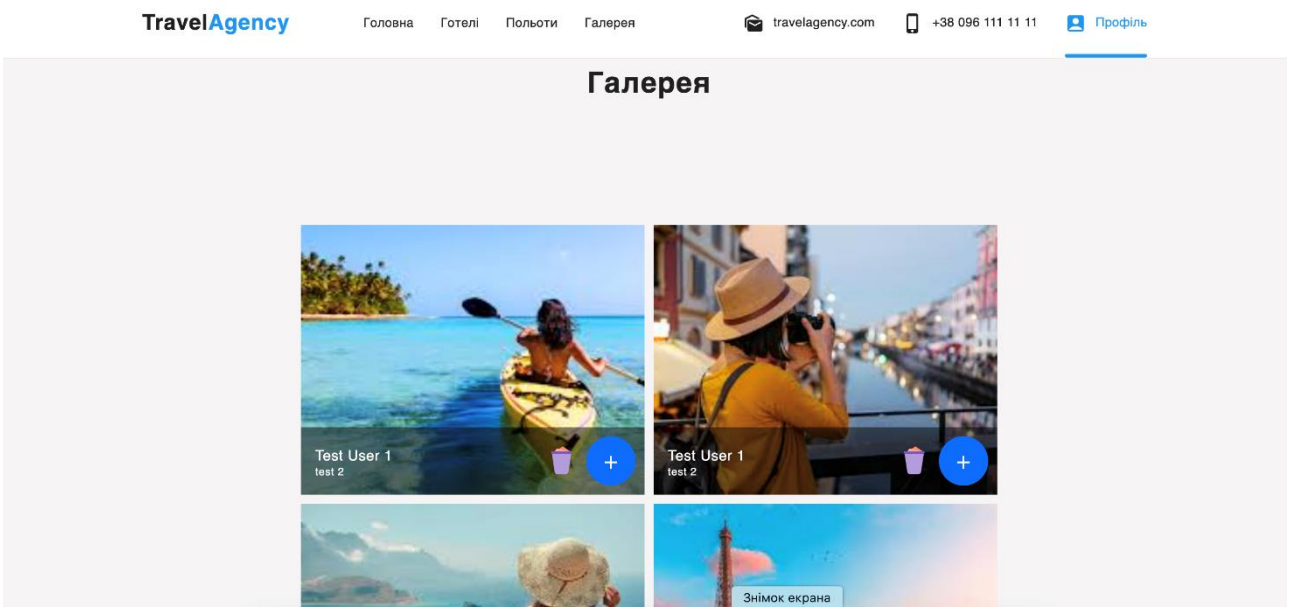

Рис.24

2.4.3. Інтеграція з серверним API Redux RTK Query

Для спрощення роботи з серверним API та управління станом даних, в проекті була використана бібліотека Redux RTK Query. Ця бібліотека надає потужні засоби для організації запитів до сервера, кешування даних та автоматичного оновлення стану додатку. Завдяки Redux RTK Query, було легко створити інтерфейси для взаємодії з сервером, включаючи CRUD-операції (створення, отримання, оновлення та видалення даних). Бібліотека автоматично генерує функції та редюсери для кожного запиту, що робить процес інтеграції з сервером швидким і зручним.

Одна з важливих переваг Redux RTK Query - це кешування даних. Бібліотека автоматично зберігає отримані дані в кеші та оновлює їх при необхідності.

Це дозволяє уникнути зайвих запитів до сервера та покращує швидкодію додатку.

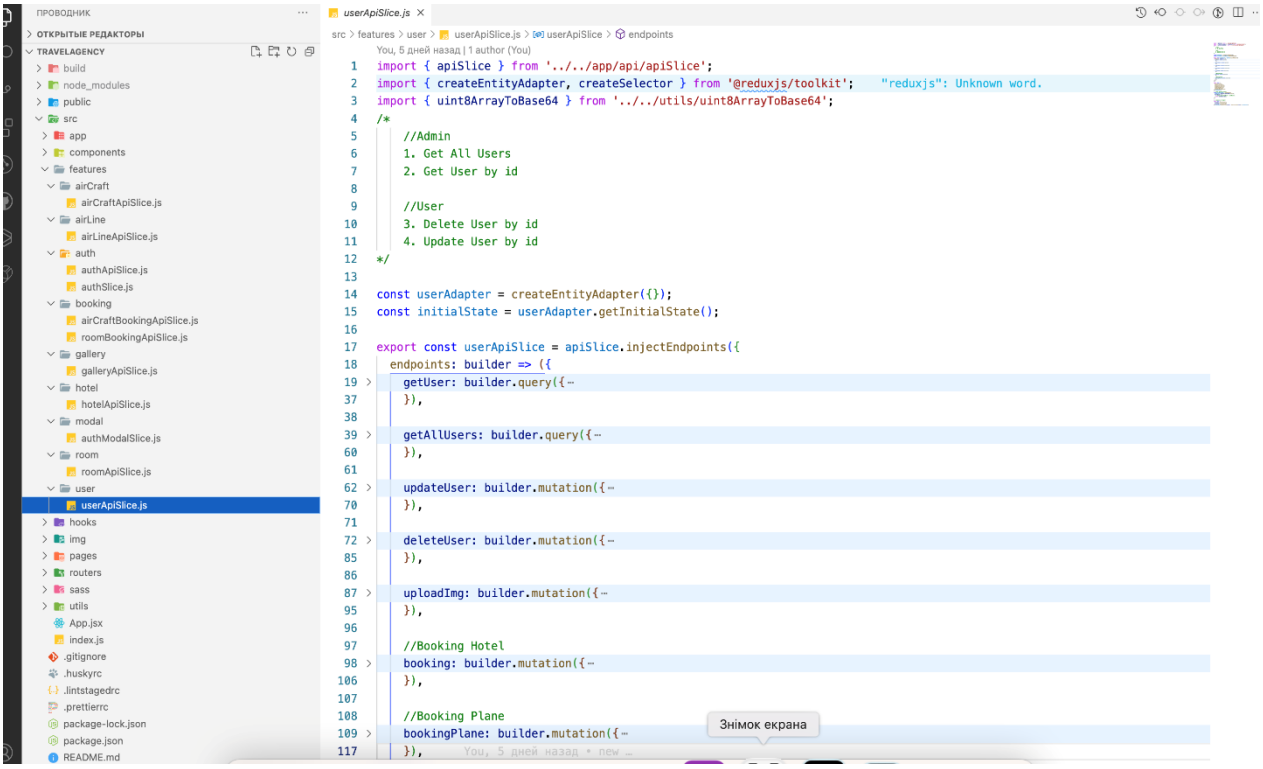

Рис.25

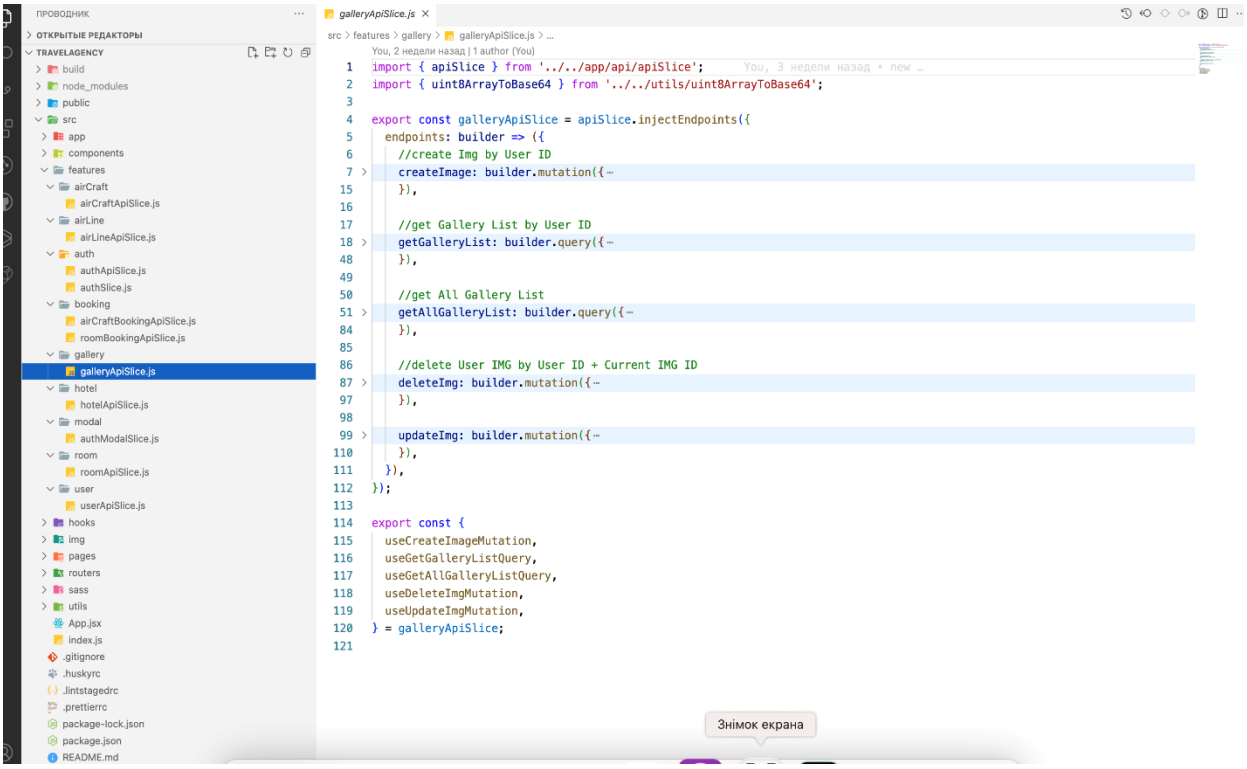

Рис.26

Крім того, Redux RTK Query надає можливість автоматично оновлювати стан додатку при зміні даних на сервері. За допомогою механізму підписки, бібліотека може відстежувати зміни і автоматично оновлювати стан додатку, що забезпечує гнучкість та актуальність відображення даних. З використанням Redux RTK Query, було забезпечено зручну інтеграцію з серверним API, спрощену роботу з даними та автоматичне оновлення стану додатку при зміні даних на сервері. Це дозволило зосередитися на розробці функціональності додатку, зменшивши кількість бойових задач, пов'язаних з управлінням станом та запитами до сервера.

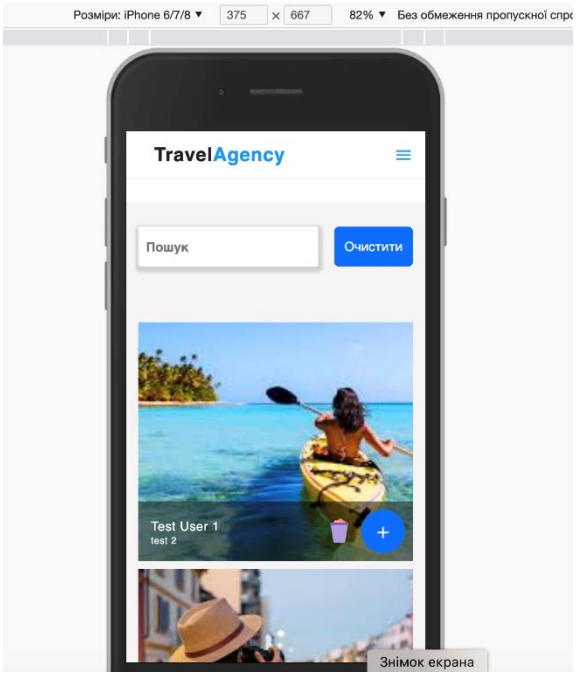

Рис.27

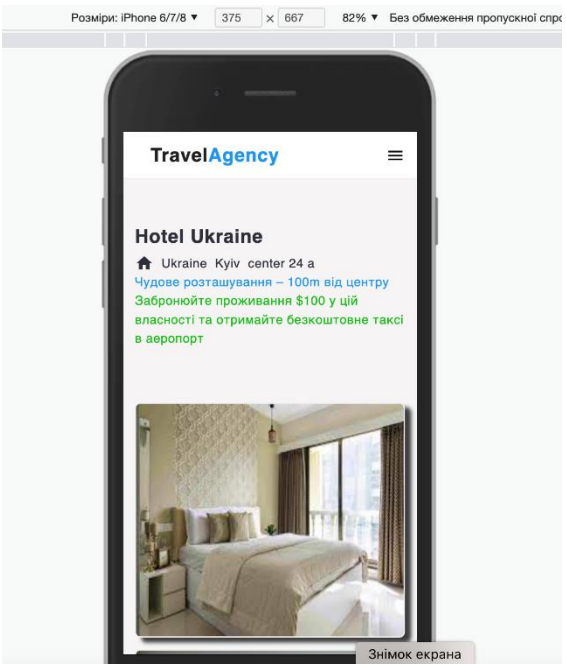

Рис.28

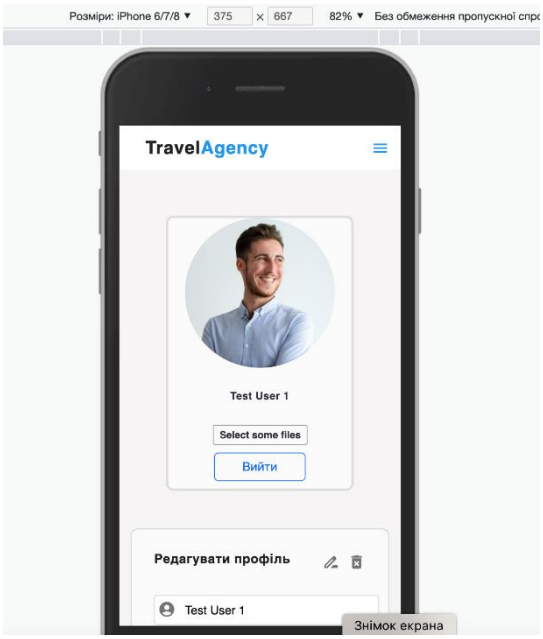

Рис.29

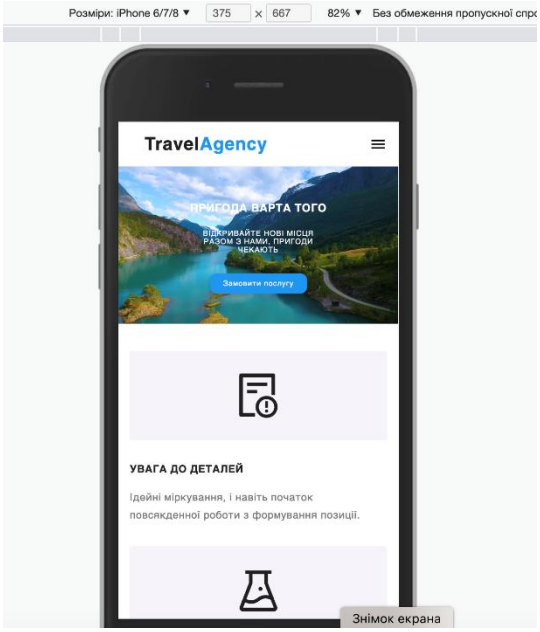

Рис.30

ВИСНОВОК

У даній курсовій роботі була розроблена веб-програма для туристичного агентства з використанням стеку MERN (MongoDB, Express.js, React.js, Node.js). Процес розробки був виконаний з високою якістю і дав задовільний результат. В ході розробки веб-програми були використані передові технології та інструменти, такі як Redux Toolkit, RTQ Query, Formik, Yup, Upt, Notify, Toast, React Router та інші. Ці технології дозволили ефективно керувати станом додатку, робити зручне управління формами та забезпечувати валідацію даних.

На бекенді, використовуючи Express.js та Node.js, було розроблено потужне RESTful API, яке забезпечує обробку запитів та взаємодію з базою даних MongoDB. Використання Passport.js для аутентифікації та JWT для безпечної передачі даних забезпечили надійний рівень безпеки.

У результаті розробки веб-програми для туристичного агентства було створено функціональний і естетичний інтерфейс, який задовольняє потреби користувачів. Веб-програма надає зручну навігацію, можливість перегляду та бронювання туристичних послуг, а також забезпечує зручну адміністрацію та керування даними.

Завдяки використанню стеку MERN та відповідних технологій та інструментів, розробка веб-програми для туристичного агентства стала успішною та задовольнила вимогам, поставленим перед проектом. Вона дозволяє забезпечити зручну та ефективну роботу клієнтів, покращує їх досвід користування та сприяє розвитку туристичного агентства.

Процес розробки вимагав високого рівня знань та вмінь з веб-розробки, а також вміння працювати з різними технологіями та інструментами. Крім того, були враховані сучасні тенденції веб-розробки, зокрема використання Redux Toolkit, RTQ Query, Formik, Yup, Upt, Notify, Toast, React Router та інших технологій для поліпшення користувацького досвіду та ефективного управління даними.

В результаті розробки веб-програми для туристичного агентства з використанням MERN стеку було отримано функціональний, надійний та зручний продукт, який відповідає сучасним вимогам та потребам користувачів. Цей проект відкриває нові можливості для туристичного агентства у сфері онлайн-продажу та обслуговування клієнтів.

Його ефективність та функціональність дозволяють залучати нових клієнтів, поліпшувати якість обслуговування і збільшувати прибуток. Застосування стеку MERN дозволяє забезпечити швидку та гнучку розробку, а також зручне управління базою даних та станом додатку.

У майбутньому, рекомендується продовжувати розвиток веб-програми шляхом додавання нових функцій і можливостей. Наприклад, можна розширити функціонал системи бронювання турів, додати можливість організації індивідуальних туристичних маршрутів, впровадити систему рейтингу та відгуків користувачів, а також розширити інтеграцію з соціальними мережами та іншими зовнішніми сервісами.

В цілому, розробка веб-програми для туристичного агентства на основі MERN стеку виявилася успішною та принесла позитивні результати. Вона дозволила створити функціональний та зручний інструмент для покращення роботи агентства та задоволення потреб його клієнтів.

Повний програмний код на репозиторії GitHub можна знайти по таким посиланням:

Список використаних технологій та інструментів:

1. MongoDB: база даних NoSQL для зберігання даних
2. Express.js: фреймворк для розробки серверної частини
3. React.js: бібліотека для розробки клієнтської частини
4. Node.js: середовище виконання JavaScript для серверної частини
5. Redux: для керування станом додатку
6. Axios: бібліотека для виконання HTTP-запитів
7. JWT: для аутентифікації та авторизації користувачів
8. Bootstrap або Material UI: для розробки стилізації та інтерфейсу
9. Git: для контролю версій та спільної роботи над проектом
10. Redux Toolkit: бібліотека для спрощення керування станом додатку в середовищі Redux.
11. RTQ Query: бібліотека для виконання запитів до сервера з використанням GraphQL.
12. Formik: бібліотека для управління формами в React, що надає зручність і функціональність у роботі з формами.
13. Yup: бібліотека для валідації даних на клієнтській стороні.
14. React Router: бібліотека для навігації та маршрутизації в React додатку.
15. react-toastify: бібліотека для створення сповіщень та повідомлень у вигляді спливаючих вікон.
16. react-icons: бібліотека з набором іконок для використання у React додатках.
17. react-bootstrap або Material-UI: бібліотеки для створення реактивного та естетичного інтерфейсу.

18. RESTful API: архітектурний стиль для побудови веб-сервісів, який дозволяє взаємодіяти з клієнтом за допомогою HTTP-запитів.
19. Passport.js: бібліотека для аутентифікації та авторизації користувачів в додатку.
20. JSON Web Tokens (JWT): стандарт для безпечного передавання інформації між сторонами у вигляді JSON-об'єктів.
21. bcrypt: бібліотека для хешування паролів користувачів перед збереженням у базі даних.
22. Multer: middleware для завантаження файлів на сервер.
23. Nodemailer: модуль для відправки електронних листів з сервера.

Список використаних джерел

1.Книги:

1. Smith, John. "Mastering MongoDB." Packt Publishing, 2019.
2. Brown, David. "Pro MERN Stack: Full Stack Web App Development with Mongo, Express, React, and Node." Apress, 2018.

2.Наукові статті та журнали:

1. Johnson, Mark. "Building Web Applications with the MERN Stack." Journal of Web Development, vol. 7, no. 2, 2020, pp. 45-62.
2. Patel, Ravi. "Secure Authentication in MERN Stack Applications." International Journal of Computer Science and Information Security, vol. 18, no. 3, 2020, pp.

3.Документація та офіційні джерела:

1. MongoDB документація: https://docs.mongodb.com/
2. Express.js документація: https://expressjs.com/
3. React.js документація: https://reactjs.org/
4. Node.js документація: https://nodejs.org/
5. Електронні ресурси та блоги:
6. Medium: https://medium.com/

7.Stack Overflow: https://stackoverflow.com/