

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування інституту, назва факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

бакалавр

(ступінь вищої освіти)

на тему РОЗРОБКА ВЕБЗАСТОСУНКУ З БРОНЮВАННЯ
КВИТКІВ НА АВТОБУСИ

DEVELOPMENT OF WEB APPLICATION FOR BUS TICKET
BOOKING

Виконав: студент(ка) 3 курсу, групи КНТ-
121сп

Спеціальності 121 Інженерія програмного
забезпечення

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

КОЛЧЕВ А.О.

(прізвище та ініціали)

Керівник КАПЛІЄНКО Т.І.

(прізвище та ініціали)

Рецензент ШИТІКОВА О.В.

(прізвище та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
(повне найменування закладу вищої освіти)

Інститут, факультет ФКНТ
Кафедра програмних засобів
Ступінь вищої освіти бакалавр
Спеціальність 121 Інженерія програмного забезпечення
(код і найменування)
Освітня програма (спеціалізація) Інженерія програмного забезпечення
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
С.О. Субботін
“ ” 20 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

Колчева Андрія Олександровича
(прізвище, ім'я, по батькові)

1. Тема проєкту (роботи) Розробка вебзастосунку з бронювання квитків на автобуси.

Development of Web Application for Bus Ticket Booking

керівник проєкту (роботи) Каплієнко Тетяна Ігорівна, к.т.н., доцент,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом закладу вищої освіти від “24” квітня 2024 року № 168

2. Строк подання студентом проєкту (роботи) 10 червня 2024 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз проблеми та постановка завдань дослідження. 2. Матеріали і методи. 3. Розробка архітектури застосунку. 4. Розробка застосунку. 5. Експлуатація, тестування та експериментальне дослідження застосунку. 6. Додатки

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

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-6 Основна частина	КАПЛІЄНКО Т.І., доцент		
Нормоконтроль	КАМІНСЬКА Ж.К., асистент		

7. Дата видачі завдання “ 15 ” квітня 2024 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	1 тиждень	Розділ 1
3	Розробка та удосконалення методів, моделей й алгоритмів вирішення задачі.	2 тиждень	Розділ 2
4	Розробка архітектури програми.	3 тиждень	Розділ 3
5	Розробка програми.	4 тиждень	Розділ 4
6	Тестування та експериментальне дослідження програмного забезпечення.	5 тиждень	Розділ 5
7	Оформлення пояснювальної записки та документів до неї.	6 тиждень	Додатки
8	Нормоконтроль та рецензування.	7 тиждень	
9	Захист роботи.	8 тиждень	

Студент(ка)

_____ Андрій КОЛЧЕВ
(підпис) (Ім'я ПРИЗВИЩЕ)

Керівник проєкту (роботи)

_____ Тетяна КАПЛІЄНКО
(підпис) (Ім'я ПРИЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 100 с., 4 табл., 37 рис., 2 дод., 11 джерел.

ВЕБЗАСТОСУНОК, БРОНЮВАННЯ КВИТКІВ, ТРАНСПОРТНІ ПОСЛУГИ, АВТОМАТИЗАЦІЯ, ЕЛЕКТРОННА КОМЕРЦІЯ.

Об'єкт дослідження – вебзастосунки для бронювання квитків на автобуси.

Мета роботи – створення вебзастосунку, який дозволяє користувачам зручно бронювати квитки на автобуси онлайн.

Методи дослідження: аналіз предметної області, розробка та вдосконалення методів, моделей і алгоритмів, розробка архітектури програми, програмування, тестування та експериментальне дослідження програмного забезпечення.

Результати: створено вебзастосунок, який дозволяє користувачам бронювати квитки на автобуси через інтернет, надаючи зручний інтерфейс та функціонал для управління бронюванням.

Висновки: розроблений вебзастосунок спрощує процес бронювання квитків на автобуси, роблячи його більш доступним та ефективним для користувачів.

ABSTRACT

Explanatory note to the bachelor's degree qualification work: 100 pages, 4 tables, 37 figures, 2 appendices, 11 sources.

WEB APPLICATION, TICKET BOOKING, TRANSPORT SERVICES, AUTOMATION, E-COMMERCE.

Object of the study – web application for booking bus tickets.

Purpose of the work – to create a web application that allows users to conveniently book bus tickets online.

Research methods: analysis of the subject area, development and improvement of methods, models, and algorithms, program architecture development, programming, testing, and experimental research of the software.

Results: a web application has been created that allows users to book bus tickets via the internet, providing a convenient interface and functionality for managing bookings.

Conclusions: the developed web application simplifies the bus ticket booking process, making it more accessible and efficient for users.

ЗМІСТ

	С.
Перелік скорочень та умовних познач.....	8
Вступ.....	9
1 Характеристика об'єкта проектування	10
1.1 Поняття онлайн-сервісу для бронювання.....	10
1.2 Функціональні можливості вебзастосунку.....	11
1.3 Аналіз аналогів.....	12
2 Вибір структури системи та його обґрунтування	15
2.1 Обґрунтування вибору середовища для серверної частини	15
2.2 Обґрунтування вибору середовища для клієнтської частини	17
2.3 Обґрунтування вибору бази даних.....	21
3 Основні рішення щодо реалізації компонентів системи.....	25
3.1 Алгоритм роботи головної сторінки.....	25
3.2 Алгоритм роботи сторінки бронювання.....	26
3.3 Алгоритм роботи сторінки реєстрації та сторінки авторизації.....	26
3.4 Алгоритм роботи сторінки адміністратора.....	28
4 Керівництво програміста.....	30
4.1 Призначення і умови використання вебзастосунку.....	30
4.2 Характеристики продукту.....	30
4.3 Структура бази даних.....	31
4.4 Структура застосунку.....	34
5 Керівництво системного адміністратора.....	35
5.1 Розгортання сайту.....	35
5.1.1 Серверна частина.....	35
5.1.2 Клієнтська частина.....	36
5.2 Адміністраторська сторінка.....	37
6 Керівництво користувача.....	40
6.1 Призначення сайту.....	40
6.2 Бронювання квитків та авторизація.....	40

Висновки	45
Перелік джерел посилання	46
Додаток А Текст програми.....	47
Додаток Б Слайди презентації	93

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

API – Application Programming Interface;

DOM – Document Object Model;

БД – база даних;

ВСТУП

В сучасному світі, де швидкість та зручність є ключовими факторами успіху, автоматизація процесів бронювання квитків на транспорт є важливою складовою ефективного функціонування транспортних компаній та підвищення задоволеності клієнтів. Однією з таких інновацій є розробка вебзастосунків для бронювання квитків на автобуси, які надають можливість користувачам швидко та зручно резервувати місця, здійснювати оплату та отримувати електронні квитки.

Ця дипломна робота присвячена розробці вебзастосунку для бронювання квитків на автобуси. Мета роботи полягає у створенні зручної та ефективної системи, яка полегшує процес бронювання та управління квитками для користувачів.

1 ХАРАКТЕРИСТИКА ОБ'ЄКТА ПРОЄКТУВАННЯ

1.1 Поняття онлайн-сервісу для бронювання

Онлайн-сервіс для бронювання – це сервіс, який в реальному часі дозволяє клієнту (юридичній або фізичній особі) забронювати місце в черзі, транспорті, та ін.. Якщо бронювання з оплатою, клієнт оплачує його за допомогою банківської картки.

Бронювання – закріплення місця за клієнтом з попередньою оплатою (повною або частковою), або без оплати, але з фіксованою ціною.

Онлайн-сервіси для бронювання існують вже більше двадцяти п'яти років. За цей доволі довгий час популярність таких сервісів постійно зростала. Тепер майже кожен, хто збирається поїхати у подорож, або відвідати кінотеатр або ресторан, користується такими сервісами. А кожна компанія пов'язана з подорожами або заклад, у котрого доволі багато клієнтів, має свій застосунок для бронювання.

Електронна комерція (е-комерція) – це процес купівлі-продажу товарів, послуг або інформації за допомогою електронних засобів та Інтернету [1]. Вебзастосунок з бронювання квитків також є прикладом електронної комерції.

При бронюванні білетів клієнтом задіюється електронний правочин.

Електронний правочин - це волевиявлення особи, спрямоване на набуття, зміну або припинення цивільних прав та обов'язків, яке здійснюється з використанням інформаційно-телекомунікаційних систем(пп. 7 ч. 1 ст. 3 Закону № 675).

Електронні угоди - це домовленості між сторонами, укладені за допомогою інформаційно-комунікаційних технологій, де пропозиція і акцепт здійснюються в електронній формі.

Електронна торгівля відноситься до «електронної комерції», що визначена у пп. 1 год. 1 ст. 3 Закону № 675: відносини, спрямовані на отримання прибутку, що виникають при скоєнні угод щодо придбання, зміни

або припинення цивільних прав та обов'язків, здійснені дистанційно з використанням інформаційно-телекомунікаційних систем, унаслідок чого в учасників таких відносин виникають права та обов'язки майнового характеру [2].

Коли клієнт бронює квиток на сайті, відбувається онлайн-покупка. Прикладом такого вебзастосунку є inbus.ua, український сервіс з бронювання квитків. За допомогою подібного вебзастосунку ви легко і швидко зможете спланувати час та дату своєї поїздки, і зарезервувати білети, знаходячись при цьому де завгодно.

1.2 Функціональні можливості вебзастосунку

Вебзастосунок планується розробляти для комфортного бронювання квитків на автобуси по Україні. При розробці багато уваги приділялося функціональності, надійності, та зручності користувача. У вебзастосунку реалізовані наступні можливості:

- інтуїтивно зрозумілий пошук інформації (розкладів за заданими маршрутами або обраною автостанцією);
- пошук та бронювання квитків;
- можливість створити особистий кабінет, у якому буде інформація про останні придбані квитки;
- можливість залишати скарги та пропозиції для покращення сервісу;
- формування квитка (бланку, котрий можна обміняти на квиток у касі на автостанції).

Людина, що збирається у подорож, та вирішить скористатися цим сервісом, витратить менше сил та часу, аніж як би вона купувала білети на автостанції. Функціонал цього вебзастосунку дозволяє планувати подорожі швидко та з комфортом.

1.3 Аналіз аналогів

Провівши аналіз трьох популярних сайтів для бронювання квитків, можна виділити наступні ключові аспекти.

Ключові аспекти Kayak.com (рис 1.1) [3]:

- простий і зрозумілий інтерфейс, який дозволяє швидко знайти необхідні рейси та порівняти ціни;
- широкий вибір фільтрів для пошуку, включаючи відстань до аеропорту, тривалість перельоту, авіакомпанії тощо;
- функція "Price Forecast" допомагає визначити найвигіднішу дату для покупки квитків;
- мобільний застосунок з такими ж функціями, як і на вебсайті;
- можливість бронювання не лише авіаквитків, але й проживання та оренди автомобілів.

Ключові аспекти Expedia.com (рис 1.2) [4]:

- великий асортимент пропозицій: авіаквитки, проживання, оренда автомобілів, пакетні тури;
- чудовий інструмент для візуального планування подорожі на карті;
- програма лояльності для постійних клієнтів з накопиченням бонусних балів;
- корисні фільтри для пошуку, такі як "Найкраща ціна" та "Найкращий час вильоту";
- розділ з відгуками туристів про готелі та пункти призначення.

Ключові аспекти Skyscanner.com (рис 1.3) [4]:

- простий і зрозумілий пошуковий інтерфейс для швидкого порівняння цін на квитки;
- функція "Найдешевші місяці" допомагає знайти найвигідніший час для подорожі;

- гнучкий пошук "Куди завгодно", який показує найдешевші напрямки з вибраного міста;
- можливість відстежувати ціни на квитки і отримувати сповіщення про знижки;
- якісна мобільна версія сайту та застосунок для смартфонів.

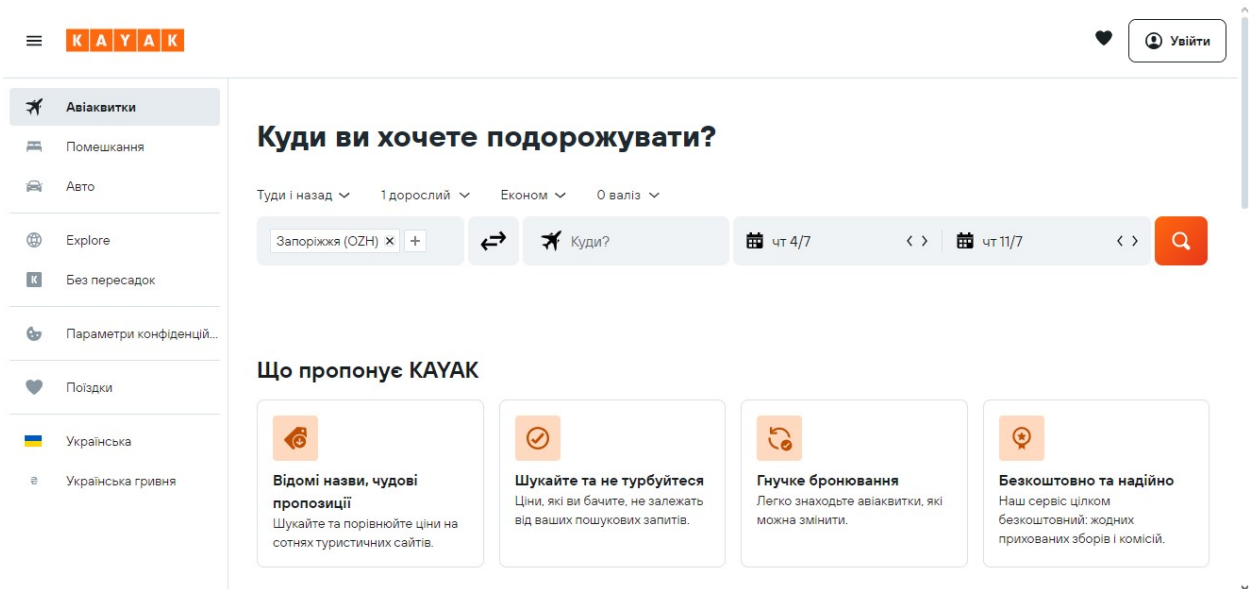


Рисунок 1.1 – Сайт Kayak.com.

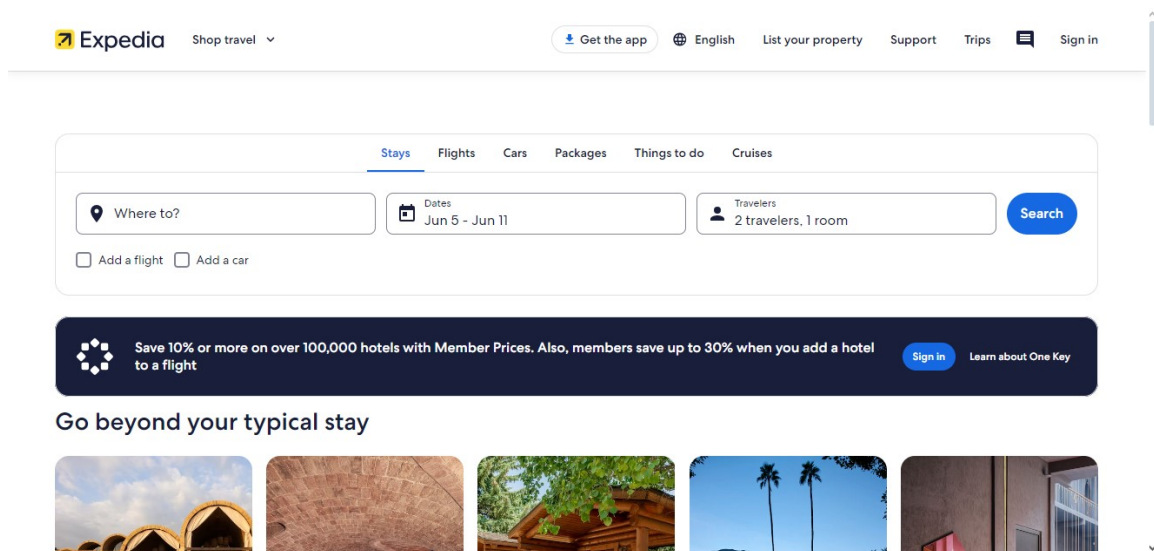


Рисунок 1.2 – Сайт Expedia.com.

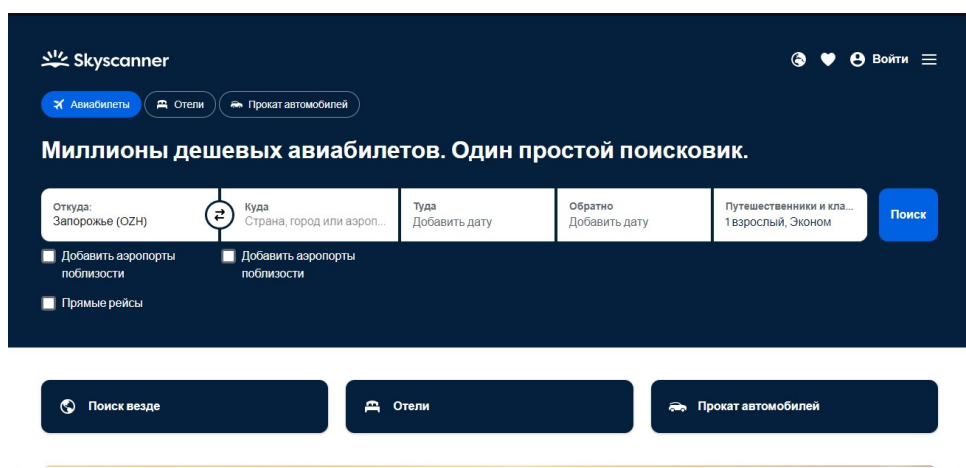


Рисунок 1.3 – Сайт Skyscanner.com.

У цілому, всі три сайти пропонують зручний пошук і бронювання авіаквитків, а також додаткові послуги з оренди автомобілів та бронювання готелів. Вони відрізняються деякими унікальними функціями і дизайном, але в основному забезпечують надійний і комфортний сервіс для планування подорожей.

1.4 Висновки за розділом

Було проведено дослідження об'єкта проектування та аналіз аналогів, сформовано набір термінів, що характеризують принципи функціонування вебзастосунку для бронювання квитків. Онлайн-сервіс для бронювання квитків на автобуси, як і в будь-якій іншій галузі, є необхідним і зручним інструментом для сучасного користувача. Він дозволяє ефективно планувати подорожі та здійснювати бронювання квитків у зручний для клієнта спосіб. Зростаюча популярність онлайн-сервісів для бронювання свідчить про їхню важливість у сфері туризму та транспортної логістики. Функціональні можливості таких вебзастосунків спрощують процес бронювання та дозволяють користувачам ефективно керувати своїми подорожами. Такі сервіси забезпечують зручність, швидкість та доступність, що робить їх важливим інструментом для сучасного мандрівника.

2. ВИБІР СТРУКТУРИ СИСТЕМИ ТА ЙОГО ОБҐРУНТУВАННЯ

2.1 Обґрунтування вибору середовища для серверної частини

Для розробки серверної частини вебзастосунку було обрано NodeJS.

Node або Node.js - програмна платформа, заснована на двигуні V8 (компілюючому JavaScript в машинний код), що перетворює JavaScript з вузькоспеціалізованої мови на мову загального призначення. Node.js додає можливість JavaScript взаємодіяти з пристроями вводу-виводу через свій API, котрий написаний на C++, підключати інші зовнішні бібліотеки, написані різними мовами, забезпечуючи виклики до них із JavaScript-коду [6].

NodeJS має низку переваг, що роблять його чудовим вибором для розробки серверної частини проєкту:

- асинхронна, подійно-орієнтована архітектура – NodeJS використовує неблокуючу, асинхронну модель введення-виведення, що дозволяє ефективно обробляти багато одночасних з'єднань з мінімальним використанням пам'яті та ресурсів процесора. Це особливо важливо для застосунків реального часу, таких як чати або ігри;
- високопродуктивний JavaScript – використання JavaScript як мови програмування для серверної частини дозволяє розробникам використовувати одну і ту саму мову для клієнтської та серверної частини, що значно спрощує розробку та підтримку проєкту;
- велика екосистема пакетів – завдяки менеджеру пакетів npm, NodeJS має доступ до величезної бібліотеки модулів та пакетів, які можна легко інтегрувати у проєкт, заощаджуючи час і зусилля на розробку;
- масштабованість – NodeJS дозволяє легко масштабувати застосунки за допомогою кластеризації та балансування навантаження, що робить його ідеальним вибором для створення високонавантажених вебзастосунків;
- швидкий цикл розробки – завдяки своїй швидкості та легкості, NodeJS забезпечує швидкий цикл розробки, що дозволяє розробникам

ітерувати та вносити зміни швидше, ніж при використанні традиційних серверних технологій;

- підтримка різних баз даних – NodeJS підтримує підключення до різноманітних типів баз даних, включаючи реляційні (як MySQL та PostgreSQL), NoSQL (як MongoDB та Cassandra), а також кешуючі системи (як Redis);

- активна спільнота – NodeJS має велику спільноту розробників, яка забезпечує постійну підтримку, оновлення та розробку нових бібліотек і модулів.

Нижче наведено таблицю 2.1 з порівнянням Node.js, Python (Django/Flask), Ruby (Ruby on Rails), PHP (Laravel), Java (Spring).

Таблиця 2.1 – Порівняння характеристик Node.js з його основними аналогами.

Характеристика	Node.js	Python (Django/Flask)	Ruby (Ruby on Rails)	PHP (Laravel)
Мова програмування	JavaScript	Python	Ruby	PHP
Однопоточність	Так	Ні	Ні	Ні
Асинхронність	Так (вбудована підтримка)	Так (через бібліотеки)	Так (через бібліотеки)	Ні (частково через бібліотеки)
Продуктивність	Висока (завдяки V8)	Середня	Середня	Середня
Простота використання	Відносно проста	Відносно проста	Складніше, ніж Python/Node.js	Відносно проста
Модульність	Висока (NPM)	Висока (PyPI)	Висока (Gems)	Висока (Composer)

Продовження таблиці 2.1

Характеристика	Node.js	Python (Django/Flask)	Ruby (Ruby on Rails)	PHP (Laravel)
Взаємодія з базами даних	Висока (через ORM, драйвери)	Висока (Django ORM, SQLAlchemy)	Висока (ActiveRecord)	Висока (Eloquent ORM)
Спільнота	Велика, активна	Велика, активна	Менша, але активна	Велика, активна
Готові рішення	Багато (Express, NestJS, Meteor)	Багато (Django, Flask)	Багато (Ruby on Rails)	Багато (Laravel, Symfony)
Розгортання	Легке (Heroku, Vercel, AWS)	Легке (Heroku, AWS)	Легке (Heroku, AWS)	Легке (Heroku, AWS)
Використання	Вебсервери, API, реального часу	Вебсервери, скрипти	Вебсервери	Вебсервери

Хоча NodeJS має свої обмеження, особливо для важких обчислювальних завдань, він залишається одним із найкращих варіантів для розробки швидких, масштабованих та надійних мережеских застосунків, що вимагають обробки великої кількості одночасних з'єднань.

2.2 Обґрунтування вибору середовища для клієнтської частини

Для розробки клієнтської частини вебзастосунку було обрано ReactJS. Звісно не обійшлося без використання HTML та CSS.

Html – стандартизована мова розмітки документів для перегляду вебсторінок у браузері [7].

Css – це спеціальна мова стилю сторінок, що використовується для опису їхнього зовнішнього вигляду. Самі ж сторінки написані мовами розмітки даних.

CSS є основною технологією всесвітньої павутини, поряд із HTML та JavaScript [8].

JavaScript – це гнучка й потужна мова програмування, яка дозволяє створювати динамічні та інтерактивні вебсторінки. Вона базується на стандарті ECMAScript і має об'єктно-орієнтовану природу з використанням концепції прототипів. Основне призначення JavaScript – надавати вебсторінкам "інтелект" і реагувати на дії користувача на клієнтській стороні. За допомогою цієї мови можна взаємодіяти з браузером, керувати його поведінкою, змінювати зміст і зовнішній вигляд сторінок, а також забезпечувати асинхронний обмін даними з сервером. JavaScript відіграє ключову роль у створенні динамічних, інтерактивних та багатофункціональних вебзастосунків, надаючи можливості для розробки складних користувацьких інтерфейсів та реалізації різноманітної функціональності на стороні клієнта [9].

React – це потужна JavaScript бібліотека з відкритим вихідним кодом, яка призначена для створення динамічних користувацьких інтерфейсів. Вона допомагає розробникам ефективно вирішувати проблеми, пов'язані з частковим оновленням вмісту вебсторінок, що є особливо важливим при розробці односторінкових застосунків. Завдяки React можна створювати масштабовані вебзастосунки, які працюють з даними, що змінюються в часі, без необхідності повного перезавантаження сторінки. Ключовими перевагами React є його швидкість, простота використання та можливість масштабування. Ця бібліотека дозволяє розробникам ефективно керувати станом застосунку, забезпечуючи плавне і реактивне оновлення інтерфейсу відповідно до змін у даних [10].

React є однією з найпопулярніших бібліотек для розробки клієнтської частини вебзастосунків, і є низка причин, чому він є чудовим вибором для цієї мети:

- компонентна архітектура – React базується на концепції компонентів, які є невеликими, повторно використовуваними шматками коду. Це спрощує розробку складних інтерфейсів користувача, забезпечуючи модульність та підвищену підтримуваність;
- віртуальний DOM – React використовує віртуальний DOM, який є легковісною копією справжнього DOM. Коли стан компонента змінюється, React обчислює мінімальні зміни, необхідні у віртуальному DOM, а потім ефективно оновлює справжній DOM. Це значно підвищує продуктивність порівняно з традиційними методами маніпуляції DOM;
- односпрямований потік даних – React використовує односпрямований потік даних, або архітектуру Flux, що робить код більш передбачуваним і полегшує відстеження походження змін у стані застосунку;
- JSX – React використовує JSX, розширення синтаксису JavaScript, що дозволяє описувати структуру компонентів за допомогою подібного до HTML коду безпосередньо у файлах JavaScript. Це робить код більш читабельним та зручним для розробників;
- велика екосистема – React має величезну екосистему бібліотек, інструментів та фреймворків, створених спільнотою. Це включає в себе React Router для маршрутизації, Redux для управління станом, а також утиліти для тестування, оптимізації та розробки;
- рендеринг на стороні сервера – React підтримує рендеринг на стороні сервера (SSR), що покращує продуктивність та SEO для вебзастосунків, забезпечуючи початкове відображення сторінки з сервера;
- активна спільнота та підтримка – React підтримується Facebook та має велику і активну спільноту розробників, що забезпечує постійні оновлення, покращення та ресурси для вивчення.

Далі наведено порівняння React, Angular, Vue.js, Svelte у таблиці 2.2.

Таблиця 2.2 – Порівняння React з його аналогами.

Характеристика	React	Angular	Vue.js	Svelte
Мова програмування	JavaScript (JSX)	TypeScript/JavaScript	JavaScript	JavaScript
Підхід	Бібліотека для UI компонентів	Повноцінний фреймворк	Прогресивний фреймворк	Компільований фреймворк
Вивчення та використання	Відносно просте	Складніше	Відносно просте	Просте
Документація	Відмінна	Відмінна	Відмінна	Відмінна
Готові компоненти	Багато (Material-UI, Ant Design)	Багато (Angular Material)	Багато (Vuetify, Quasar)	Обмеженіше
Двостороння прив'язка даних	Ні	Так	Налаштовується	Ні
Віртуальний DOM	Так	Так	Так	Ні
Розмір	Малий	Великий	Малий	Дуже малий
Продуктивність	Висока	Висока	Висока	Дуже висока
Сумісність з мобільними застосунками	React Native	Ionic + Angular	Weex (для Alibaba)	Sapper
Спільнота	Дуже велика	Велика	Велика	Розвивається

Хоча є й інші бібліотеки та фреймворки для розробки клієнтської частини, React виділяється своєю гнучкістю, продуктивністю та простотою в розробці складних користувацьких інтерфейсів. Завдяки своїм перевагам, він

став одним з найбільш широко використовуваних інструментів для створення сучасних вебзастосунків.

2.3 Обґрунтування вибору бази даних

Системою управління базами даних для проєкту було обрано PostgreSQL.

PostgreSQL є потужною об'єктно-реляційною системою управління базами даних з відкритим вихідним кодом. Вона слугує альтернативою як комерційним системам керування базами даних, таким як Oracle Database, Microsoft SQL Server та IBM DB2, так і іншим відкритим рішенням, наприклад, MySQL, Firebird чи SQLite.

На відміну від деяких інших відкритих проєктів, розробка PostgreSQL не контролюється якоюсь однією компанією. Натомість, завдяки співпраці численних розробників та компаній, які мають інтерес у використанні та вдосконаленні цієї системи керування базами даних, PostgreSQL постійно розвивається та інтегрує новітні досягнення в галузі.

Така модель спільного розвитку дозволяє PostgreSQL залишатися гнучким, сучасним та високоякісним рішенням, що відповідає найвищим стандартам галузі та потребам користувачів [11].

PostgreSQL має низку переваг, що роблять її чудовим вибором для розробки вебзастосунків:

- надійність та стійкість – PostgreSQL відомий своєю високою надійністю та стійкістю до збоїв. Він підтримує потужні функції, такі як ACID-сумісні транзакції, відновлення після збоїв та реплікацію, що забезпечує безперервну доступність даних;
- масштабованість – PostgreSQL може ефективно працювати з великими обсягами даних та великою кількістю одночасних підключень, що робить його ідеальним вибором для масштабованих вебзастосунків із високим навантаженням;

- сумісність з стандартами SQL – PostgreSQL в основному дотримується стандартів SQL, що полегшує міграцію з інших реляційних баз даних та забезпечує сумісність з широким колом інструментів та бібліотек;
- розширюваність – PostgreSQL має потужну систему розширень, що дозволяє додавати нові типи даних, функції, оператори та процедури на різних мовах програмування, таких як PL/pgSQL, PL/Python, PL/Perl та інші;
- повна підтримка JSON – PostgreSQL має вбудовану підтримку JSON, що дозволяє ефективно зберігати та обробляти структуровані дані, що корисно для вебзастосунків з гнучкими схемами даних;
- геопросторові можливості – PostgreSQL має розширення PostGIS, що надає потужні можливості для роботи з геопросторовими даними, включаючи підтримку географічних об'єктів, просторових індексів та геометричних операцій;
- активна спільнота та відкритий вихідний код – PostgreSQL має велику та активну спільноту розробників, що забезпечує постійну підтримку, оновлення та розробку нових функцій. Оскільки це проєкт з відкритим вихідним кодом, розробники можуть переглядати та модифікувати вихідний код за потребою.

Далі наведено порівняння PostgreSQL, MySQL, SQLite, MongoDB у таблиці 2.3.

Таблиця 2.3 – Порівняння PostgreSQL з його аналогами.

Характеристика	PostgreSQL	MySQL	SQLite	MongoDB
Тип	Реляційна СУБД	Реляційна СУБД	Реляційна СУБД	Документо-орієнтована СУБД
Мова запитів	SQL	SQL	SQL	BSON (JSON-орієнтований)
Підтримка транзакцій	Так	Так	Частково (без WAL)	Ні (підтримка ACID на рівні колекцій)

Продовження таблиці 2.3

Характеристика	PostgreSQL	MySQL	SQLite	MongoDB
Підтримка JSON	Так (JSONB)	Так (JSON)	Ні	Рідний формат
Підтримка схем	Так	Так	Ні	Ні
Розширюваність	Висока (через розширення)	Обмежена	Ні	Висока (плагіни, модулі)
Реплікація	Так	Так	Ні	Так
Масштабованість	Висока	Висока	Низька	Висока
Підтримка гео-даних	Так (PostGIS)	Частково (через розширення)	Ні	Ні
Безпека	Висока (розширена підтримка)	Висока	Низька	Висока
Продуктивність	Висока	Висока	Середня	Висока (на великих даних)
Підтримка користувачів	Велика, активна спільнота	Велика, активна спільнота	Менша спільнота	Велика, активна спільнота

Хоча існують й інші реляційні та нереляційні бази даних, PostgreSQL виділяється своєю надійністю, масштабованістю, сумісністю зі стандартами та розширюваністю. Ці властивості роблять його ідеальним вибором для створення складних та високонавантажених вебзастосунків, що вимагають надійного та гнучкого зберігання даних.

2.4 Висновок за розділом

Для розробки серверної частини вебзастосунку було обрано NodeJS завдяки його асинхронній, подійно-орієнтованій архітектурі, високій продуктивності JavaScript, великій екосистемі пакетів, масштабованості, швидкому циклу розробки, підтримці різних баз даних та активній спільноті розробників.

Для розробки клієнтської частини вебзастосунку було обрано ReactJS, комбінуючи його з HTML та CSS. ReactJS був обраний завдяки компонентній архітектурі, використанню віртуального DOM, односпрямованому потоку даних, JSX, великій екосистемі, підтримці рендерингу на стороні сервера та активній спільноті.

В якості бази даних для проєкту була обрана PostgreSQL через її надійність, стійкість, масштабованість, сумісність зі стандартами SQL, розширюваність, вбудовану підтримку JSON, геопросторові можливості, активну спільноту та відкритий вихідний код.

Загалом, вибір цих технологій обґрунтовується їх перевагами у швидкості, масштабованості, продуктивності, гнучкості та підтримці спільнотою, що робить їх ідеальним рішенням для створення сучасного та високопродуктивного вебзастосунку.

3 ОСНОВНІ РІШЕННЯ ЩОДО РЕАЛІЗАЦІЇ КОМПОНЕНТІВ СИСТЕМИ

3.1 Алгоритм роботи головної сторінки

Розглянемо алгоритм роботи головної сторінки. Коли відкривається головна сторінка, користувач бачить поля для вводу даних, за рахунок яких ведеться пошук маршрутів у базі. Користувач вводить пункт відправлення, пункт прибуття, та дату, на яку планується поїздка. Коли користувач ввів дані, та натиснув на кнопку «Знайти», відправляється запит з заданими даними у БД, яка в свою чергу відправляє список маршрутів, який потім записується у сформовану нижче таблицю. При натисненні на відповідну кнопку «Оформити» напроти рядка таблиці з маршрутами, користувача переадресує на сторінку оформлення квитків. Також на кожній сторінці у хедері є можливість реєстрації та авторизації.

Блок-схема алгоритму роботи головної сторінки наведено нижче на рисунку 3.1.

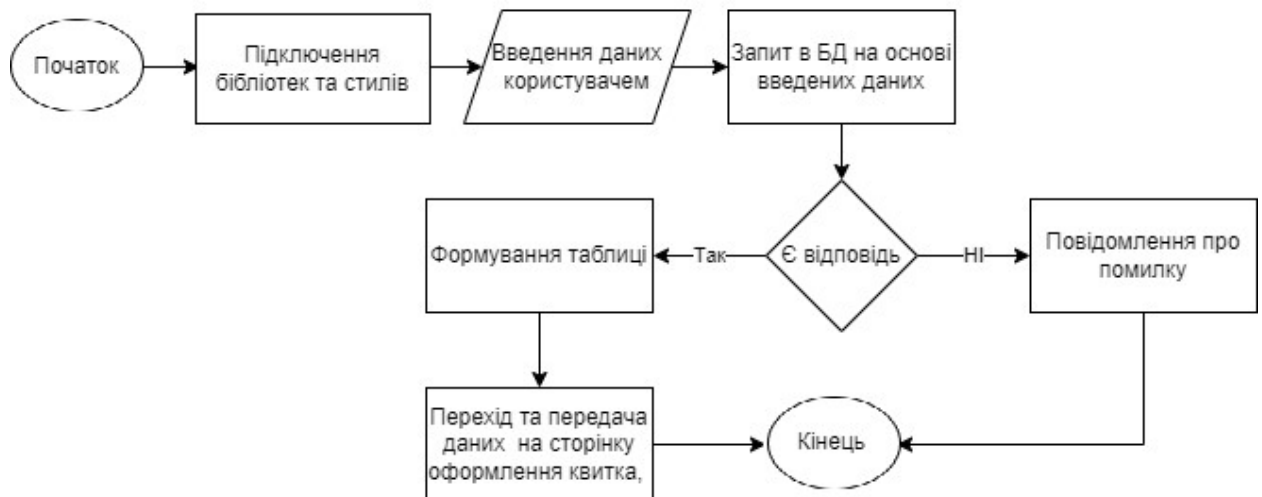


Рисунок 3.1 – Блок-схема основного алгоритму роботи головної сторінки.

3.2 Алгоритм роботи сторінки бронювання

При переході з головної на сторінку бронювання, туди передаються дані про обраний рейс. Користувач обирає місце в автобусі та вводить свої особисті дані, або вони вводяться при переході на сторінку бронювання автоматично, у разі якщо користувач авторизований. Після введення даних та натискання на кнопку «Перейти до оплати», робиться запит до БД, аби додати дані про нового клієнта та оновити дані про зайняте місце.

Блок-схема основного алгоритму цієї сторінки надана нижче на рисунку 3.2.

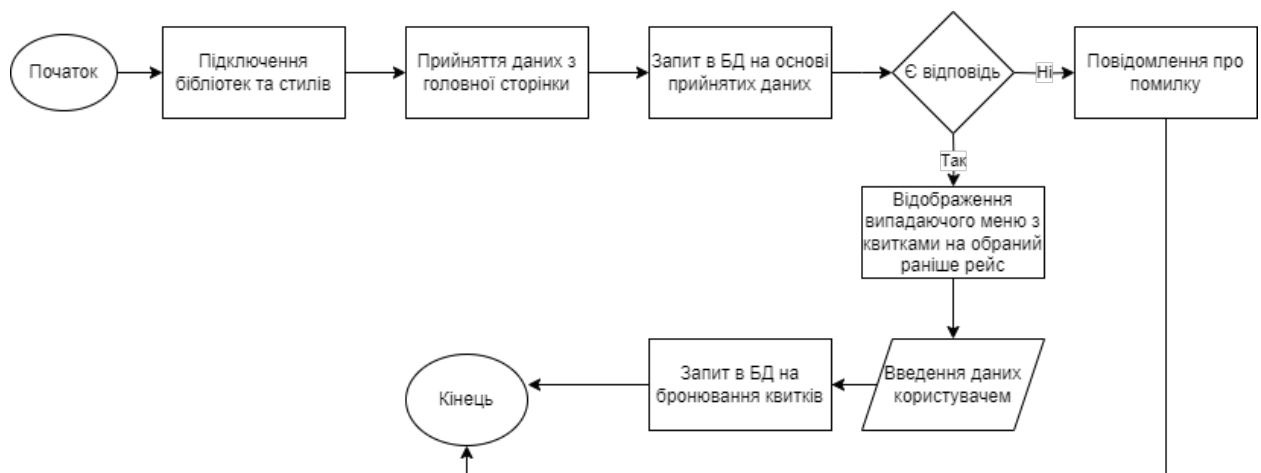


Рисунок 3.2 – Блок-схема основного алгоритму роботи сторінки бронювання.

3.3 Алгоритм роботи сторінки реєстрації та сторінки авторизації

Сторінки реєстрації та авторизації мають схожий алгоритм. Потрапити на них можна при натисненні на відповідну кнопку у хедері на головній сторінці або на сторінці бронювання. Після того як клієнт потрапить на сторінку реєстрації, на екрані відобразяться поля для введення особистих даних. Коли користувач введе дані та натисне кнопку «Зареєструватися», сформується запит у БД, котрий додасть туди новий аккаунт користувача,

після чого з сервера повернуться ці дані, та система автоматично виконає авторизацію.

Блок-схему алгоритму роботи сторінки реєстрації надано далі на рисунку 3.3.

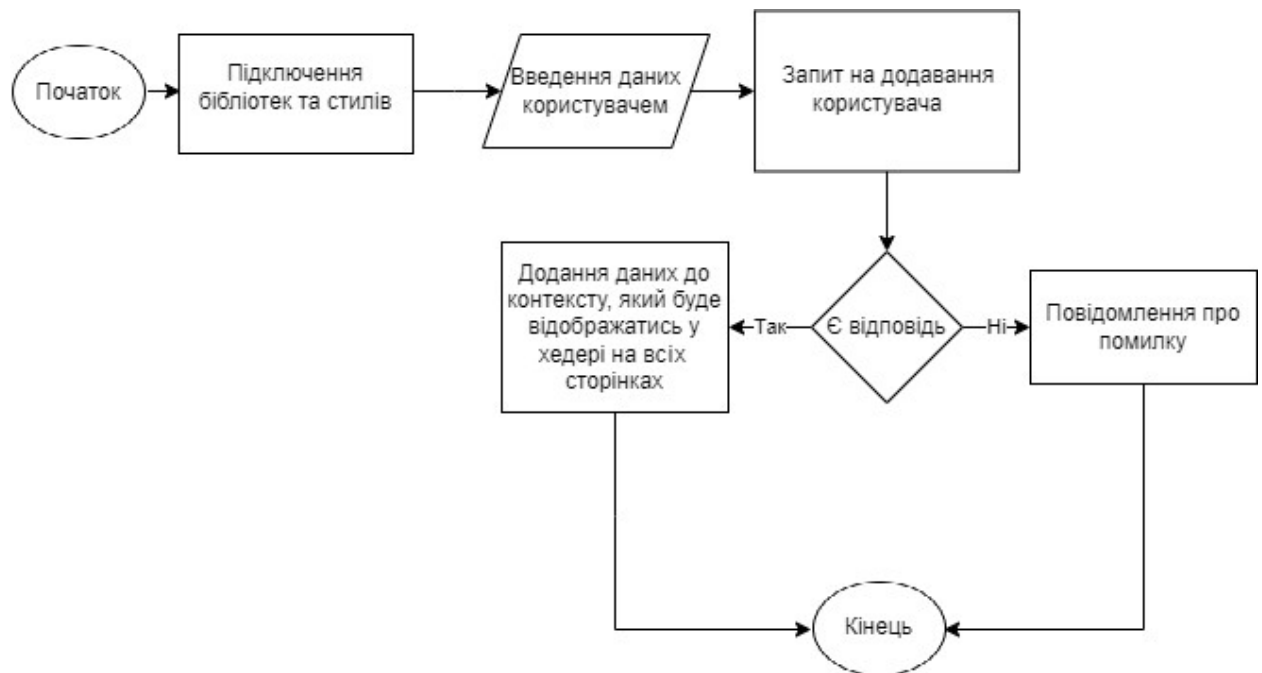


Рисунок 3.3 – Блок-схема алгоритму сторінки реєстрації.

У випадку зі сторінкою авторизації користувач вводить Email та пароль, та натискає на кнопку «Увійти». Ця дія формує запит до БД, в якому порівнюються дані введені клієнтом, та записані у БД. Якщо дані вірні, то система авторизує користувача.

Нижче наведено блок-схему алгоритму роботи сторінки авторизації на рисунку 3.4.

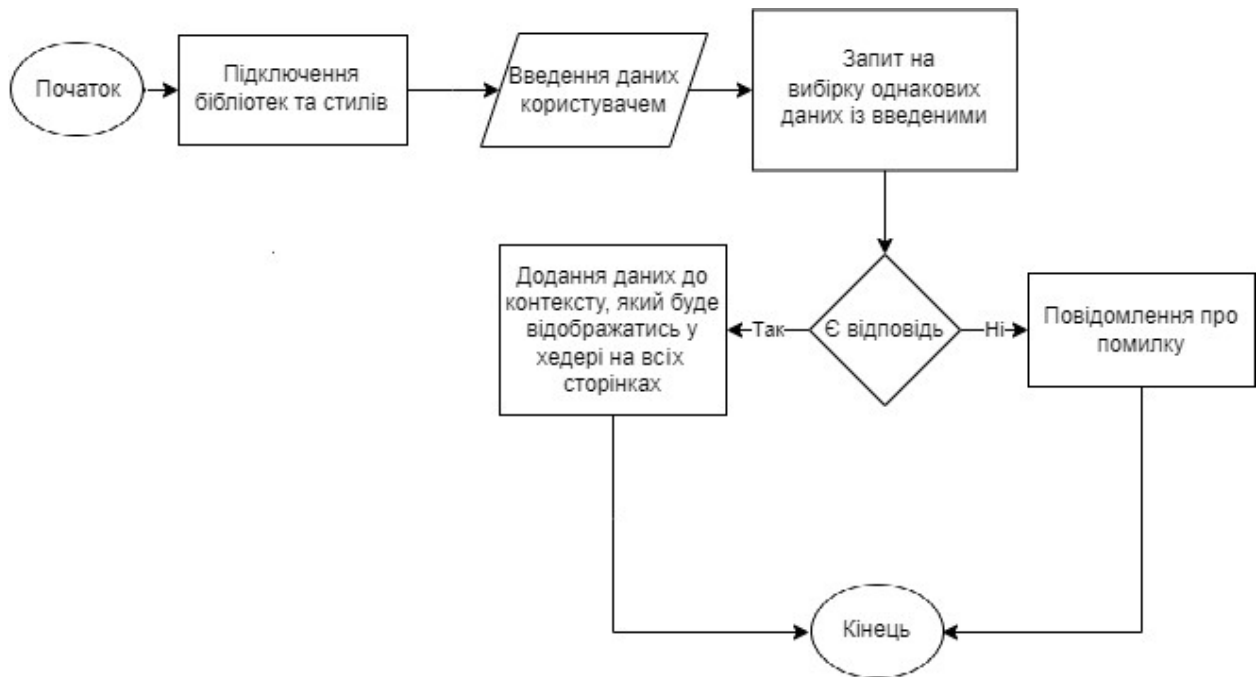


Рисунок 3.4 – Блок-схема алгоритму роботи сторінки авторизації.

3.4 Алгоритм роботи сторінки адміністратора

Перейти на сторінку адміністратора можна якщо ввести на сторінці авторизації в поля Email та пароль «admin». У хедері на цій формі 6 кнопок: «Вихід», «Міста», «Станції», «Маршрути», «Клієнти», «Білет».

При натисненні на кнопку виходу, користувача буде перенаправлено на головну сторінку. Інші кнопки формують запит у БД до відповідної таблиці.

Після натиснення однієї з них буде відображено весь вміст таблиці, а також панель управління, яка містить поля для вводу даних, та кнопки: «Очистити», «Оновити запис», «Додати запис», «Видалити запис», «Знайти», «Відміна фільтру». При натисканні на кнопку «Очистити» поля для введення даних очищуються. Кнопка «Знайти» шукає в таблиці дані, що схожі на введені в одне або кілька полів. «Відміна фільтру» знову відображає повну таблицю. Кнопка «Оновити запис» відправляє до БД запит з відредагованими даними. «Видалити запис» відправляє запит на видалення обраного запису. Для оновлення та видалення даних зручно обирати записи натискаючи на них

у таблиці. І нарешті кнопка «Додати запис» відправляє запит на додавання введених даних до БД.

Блок-схема алгоритму роботи сторінки адміністратора наведена далі на рисунку 3.5.

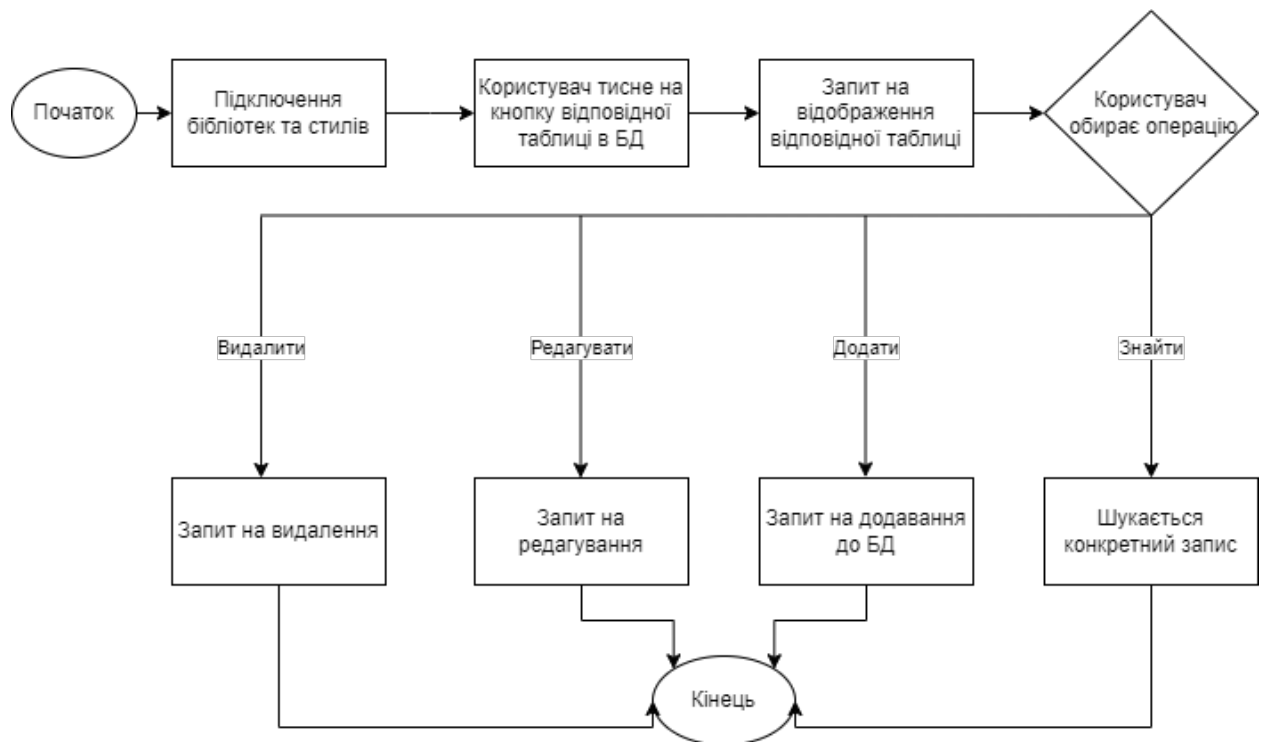


Рисунок 3.5 – Блок-схема алгоритму роботи сторінки адміністратора.

4 КЕРІВНИЦТВО ПРОГРАМІСТА

4.1 Призначення і умови використання вебзастосунку

Вебзастосунок призначений для автоматизації бронювання квитків на автобуси.

Клієнтська частина розроблена за допомогою гіпертекстової мови розмітки HTML, каскадних таблиць стилів CSS, та JavaScript бібліотеки React. Серверна частина розроблена за допомогою Node.js, та СКБД PostgreSQL.

Задля надійної експлуатації застосунку потрібен інтернет та вебсервер, з усіма встановленими модулями сайту та БД на ньому.

Для роботи вебсайту потрібен будь який пристрій, що здатен відображати вебсторінки, має можливість вводу даних, має вихід в інтернет, та має попередньо встановлений веббраузер, наприклад: Google Chrome, Opera, Mozilla Firefox, Internet Explorer, або інші.

4.2 Характеристики продукту

Під час розробки застосунку були зроблені кілька логічних блоків, на які можна умовно поділити сайт:

- statPage.jsx – логічний блок, у якому відбувається пошук рейсів;
- booking.jsx – блок безпосередньо бронювання квитків;
- admin.jsx – блок для управління БД адміністратором.

Опис деяких з функцій наведені нижче.

Функція пошуку рейсів потрібна аби користувач зміг знайти рейс на основі введених даних про початкову і кінцеву точки маршруту, та дату. Блок-схема цієї функції на рисунку 4.1.

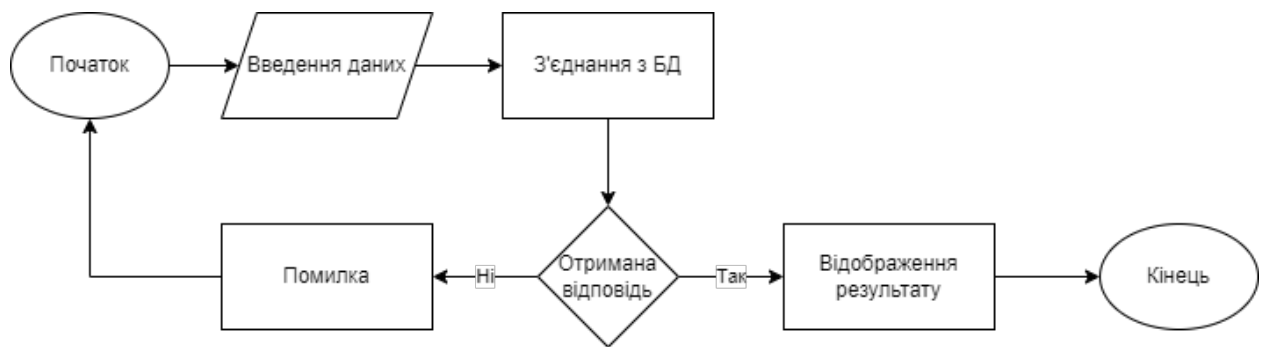


Рисунок 4.1 – Блок-схема функції пошуку рейсів.

Функція авторизації корисна для користувачів, які часто подорожують. Аби не вводити кожен раз свої дані достатньо авторизуватися, і система введе їх сама.

Функція бронювання квитка розроблена для додавання до бази запису про клієнта та зміни стану вільного місця на зайняте.

4.3 Структура бази даних

В основі будь-якого вебзастосунку лежить база даних. Цей сайт не став виключенням. БД складається з 5 таблиць зв'язаних «один до багатьох», та однієї не зв'язаної:

- місто – містить інформацію про міста;
- автовокзал – містить інформацію про автовокзали, які є в містах;
- рейс – містить інформацію про існуючі рейси;
- місця – містить інформацію про доступні та зайняті місця;
- клієнти – містить інформацію про людей, що забронювали квитки;
- профілі – містить інформацію про профілі клієнтів.

Як саме зв'язані між собою таблиці представлено на схемі нижче, рисунок 4.2.

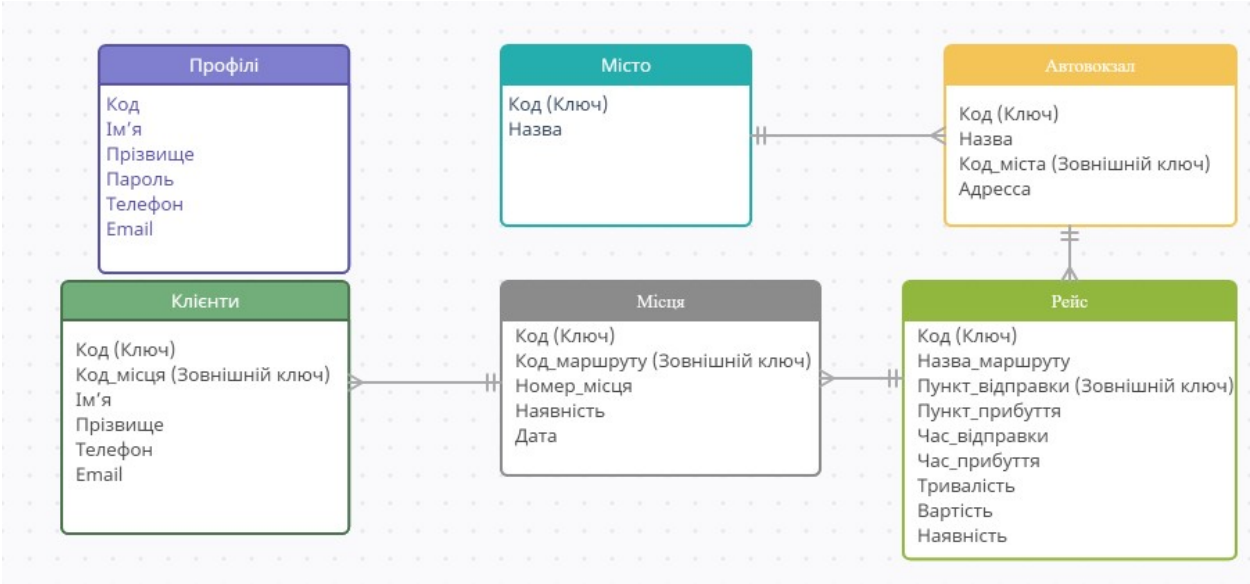


Рисунок 4.2 – Схема структури бази даних.

Зв’язувався застосунок з базою даних за допомогою мови запитів SQL. Приклади SQL запити наведено у таблиці 4.1 нижче.

Таблиця 4.1 – Приклади запитів у застосунку.

SQL запит	Опис запиту
<pre>SELECT routes.id, routes.route_name, departure.bus_station_name AS point_of_departure, arrival.bus_station_name AS point_of_arrival, place.place_date, routes.time_of_departure, routes.time_of_arrival, routes.duration, routes.price FROM (bus_station AS departure INNER JOIN (routes INNER JOIN bus_station AS arrival ON routes.point_of_arrival = arrival.id) ON departure.id = routes.point_of_departure) INNER JOIN place ON routes.id = place.id_route WHERE routes.point_of_departure = \$1 AND routes.point_of_arrival = \$2 AND</pre>	Пошук маршрутів

Продовження таблиці 4.1

SQL запит	Опис запиту
<pre>place.place_date = \$3 AND routes.route_existence = 'true' GROUP BY routes.id, routes.route_name, departure.bus_station_name, arrival.bus_station_name, place.place_date, routes.time_of_departure, routes.time_of_arrival, routes.duration, routes.price</pre>	
<pre>SELECT * FROM profiles WHERE email = \$1 AND profile_password = \$2</pre>	Пошук даних аккаунту людини для порівняння при авторизації
<pre>UPDATE place SET place_existence = 'false' WHERE id_route = \$1 AND place_number = \$2 AND place_date = \$3 RETURNING *</pre>	Бронювання білета
<pre>INSERT INTO routes (route_name, point_of_departure, point_of_arrival, time_of_departure, time_of_arrival, duration, price, route_existence) VALUES (\$1, \$2, \$3, \$4, \$5, \$6, \$7, \$8) RETURNING *</pre>	Створення маршруту
<pre>UPDATE routes SET route_name = \$1, point_of_departure = \$2, point_of_arrival = \$3, time_of_departure = \$4, time_of_arrival = \$5, duration = \$6, price = \$7, route_existence = \$8 WHERE id = \$9 RETURNING *</pre>	Редагування маршруту

Дані з програми у запит записувались через змінні, які мають такий вигляд: «\$1».

4.4 Структура застосунку

Умовно сайт можна поділити на дві частини: клієнтську та адміністраторську. Клієнтська частина відповідає за роботу з клієнтом, тобто здійснює пошук рейсів, вільних місць, та бронювання квитків. Адміністраторська частина в свою чергу відповідає за керування базою даних. Лише адміністратор може зайти на неї. На цій сторінці можна додавати, редагувати, видаляти, та фільтрувати дані з усіх таблиць. Схему сайту зображено далі на рисунку 4.5.



Рисунок 4.5 – Схема структури сайту.

5 КЕРІВНИЦТВО СИСТЕМНОГО АДМІНІСТРАТОРА

5.1 Розгортання сайту

5.1.1 Серверна частина

Для початку потрібно підключити флешку та знайти на ній архів «tickets_app.rar»

Потім потрібно перенести папку tickets_app в папку серверної частини, а папку tickets – до клієнтської.

Перед тим, як перейти до наступного етапу, потрібно впевнитись, що на сервері встановлені такі програми:

- Node.js і npm;
- PostgreSQL;
- Nginx.

Далі налаштування тилової частини сайту. Переходимо до директорії з серверною частиною, та встановлюємо залежності вводом наступних команд.

```
cd /path/to/your/project/backend  
npm install
```

У папці tickets_app є файл з базою даних «tickets.backup». Потрібно підключитися до PostgreSQL та створити нову базу даних і користувача. Для цього виконаємо наступні команди.

```
psql -U postgres  
CREATE DATABASE tickets;  
CREATE USER myuser WITH ENCRYPTED PASSWORD  
'mypassword';  
GRANT ALL PRIVILEGES ON DATABASE tickets TO myuser;  
\q
```

Далі в терміналі потрібно ввести команду для відновлення бази даних з файлу «tickets.backup».

```
pg_restore -U myuser -W -d tickets -v Шлях до
БД/tickets.backup
```

Після того як БД успішно перенесено на сервер, потрібно створити файл конфігурації для підключення до неї. Це можна зробити наступною командою.

```
DATABASE_URL=postgres://myuser:mypassword@localhost:5432
/tickets
```

Тепер можна запускати сервер. Для цього введемо команду запуску.

```
node server
```

5.1.2 Клієнтська частина

Перейдемо до директорії фронтенду та встановимо залежності наступними командами.

```
cd /path/to/your/project/frontend
npm install
```

Далі створимо production build застосунку.

```
npm run build
```

Після створення production build переконайтеся, що Nginx встановлений, та налаштуйте його для проксування запитів до backend та обслуговування статичних файлів frontend. Потрібно створити файл конфігурації сайту. Це робиться за рахунок наступної команди.

```
sudo nano /etc/nginx/sites-available/tickets
```

Далі конфігурації:

```
server {
    listen 80;
    server_name yourdomain.com;

    location / {
        root /path/to/your/project/frontend/build;
```

```

        try_files $uri /index.html;
    }

    location /api {
        proxy_pass http://localhost:3000;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host $host;
        proxy_cache_bypass $http_upgrade;
    }
}

```

Активуйте конфігурацію і перезапустіть Nginx:

```

sudo ln -s /etc/nginx/sites-available/tickets
/etc/nginx/sites-enabled/
sudo systemctl restart nginx

```

Для керування процесами Node.js можна використовувати pm2:

Тепер застосунок повинен бути доступний за вказаним доменом. Не забудьте налаштувати DNS для домену, щоб він вказував на IP-адресу сервера.

5.2 Адміністраторська сторінка

Щоб зайти на адміністраторську сторінку потрібно авторизуватися як адміністратор. Для цього потрібно ввести в поле «Email» та «пароль» слово admin, як показано на рисунку 5.1.

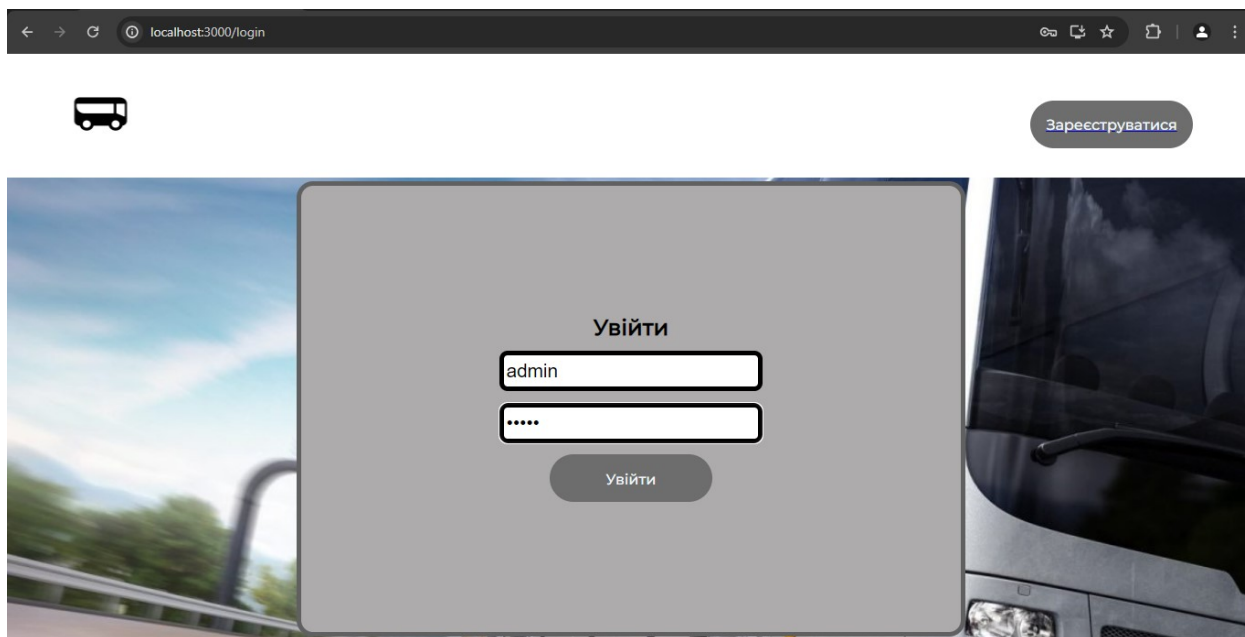


Рисунок 5.1 – Вхід на адміністраторську сторінку.

Після входу ми побачимо сторінку адміністратора як на рисунку 5.2.

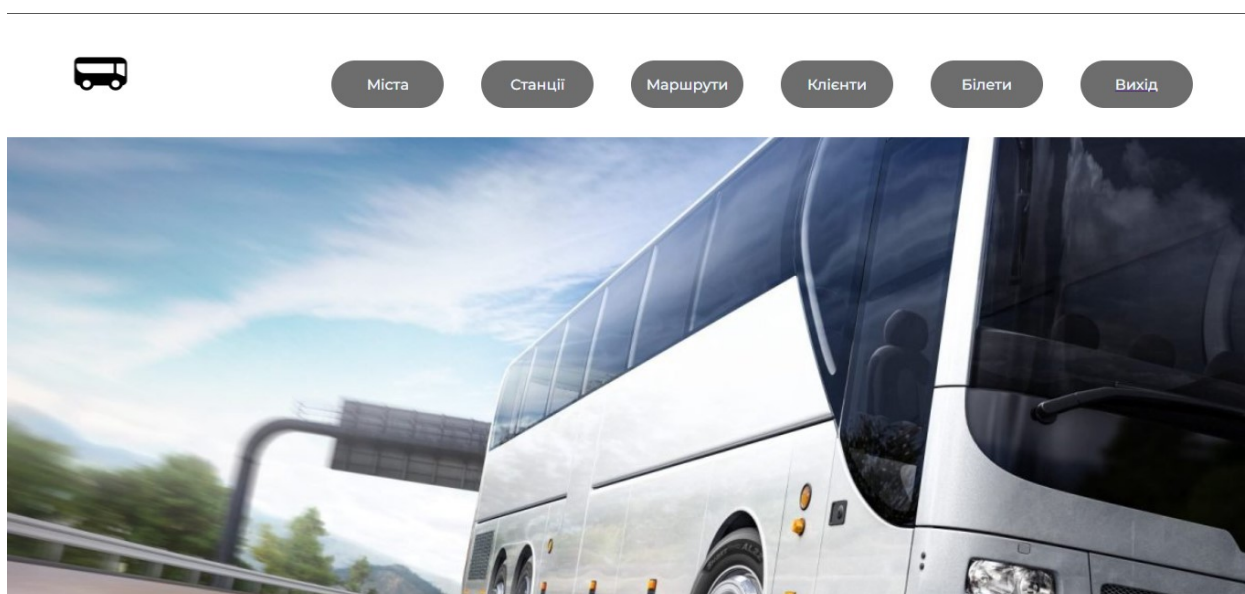


Рисунок 5.2 – Адміністраторська сторінка.

На цій сторінці знаходяться кнопки, які відповідають за відображення елементів керування відповідними таблицями бази даних. Структура цих елементів однакова, тому розглянемо як приклад елемент управління таблицею «Маршрути». Побачити його можна на рисунку 5.3.

id	Маршрут	Пункт відправлення	Пункт прибуття	Час відправлення	Час прибуття	Тривалість поїздки	Ціна	Наявність
1	Київ-Запоріжжя	1	2	12:20	18:40	6 год. 20 хв.	555 грн	true
2	Запоріжжя-Київ	2	1	12:00	18:20	6 год. 20 хв.	555 грн	true
3	Київ-Харків	1	4	09:15	13:30	4 год. 15 хв.	438 грн	true
4	Харків-Київ	4	1	14:00	18:15	4 год. 15 хв.	438 грн	true
5	Харків-Полтава	4	6	10:00	13:00	3 год.	325 грн	true
6	Полтава-Харків	6	4	14:00	17:00	3 год.	325 грн	true
7	Харків-Запоріжжя	4	2	11:00	14:00	5 год.	2550 грн	true
8	Запоріжжя-Харків	2	4	15:00	20:00	5 год.	2550 грн	true

Рисунок 5.3 – Вигляд елемента «Маршрути».

Тут можна взаємодіяти з таблицею «Маршрути». Щоб додати запис потрібно заповнити поля (окрім поля id, бо це поле ключ, що обирається системою. У разі заповнення цього поля нічого не станеться, просто система сама обере значення цього поля у записі.), та натиснути на кнопку «Додати запис», після чого таблиця оновиться, і в її кінці вже буде відображено новий рядок.

Щоб редагувати, або видалити запис, його потрібно обрати. Іноді зручніше просто ввести потрібні дані в поля, а іноді просто обрати потрібний рядок у таблиці. Щоб знайти потрібний рядок по одному або декількох відомих значеннях, потрібно ввести ці дані у відповідні поля, та натиснути кнопку «Знайти». Після цієї маніпуляції в таблиці залишаться лише відповідні введеним даним записи, і можна натиснути на потрібний рядок, аби вибрати його. Після цього можна відредагувати або видалити цей запис, натиснувши на відповідну кнопку. Коли фільтр буде не потрібен, треба натиснути на кнопку «Відміна фільтру», щоб таблиця повернулась у свій повний вигляд. Якщо потрібно очистити поля для вводу, треба натиснути на кнопку «Очистити». Так само працюють і елементи інших таблиць. Якщо сторінка більше не потрібна, можна натиснути «Вихід», і система поверне користувача на головну сторінку.

6 КЕРІВНИЦТВО КОРИСТУВАЧА

6.1 Призначення сайту

Сайт призначений для бронювання білетів на автобуси онлайн. Він зроблений для того, щоб зекономити час як користувачів, так і працівників цієї сфери, та спростити в ній роботу, автоматизувавши її частину.

6.2 Бронювання квитків та авторизація

При переході за адресою, яку визначить замовник сайту, користувач переходить на головну сторінку. Головна сторінка продемонстрована далі на рисунку 6.1.

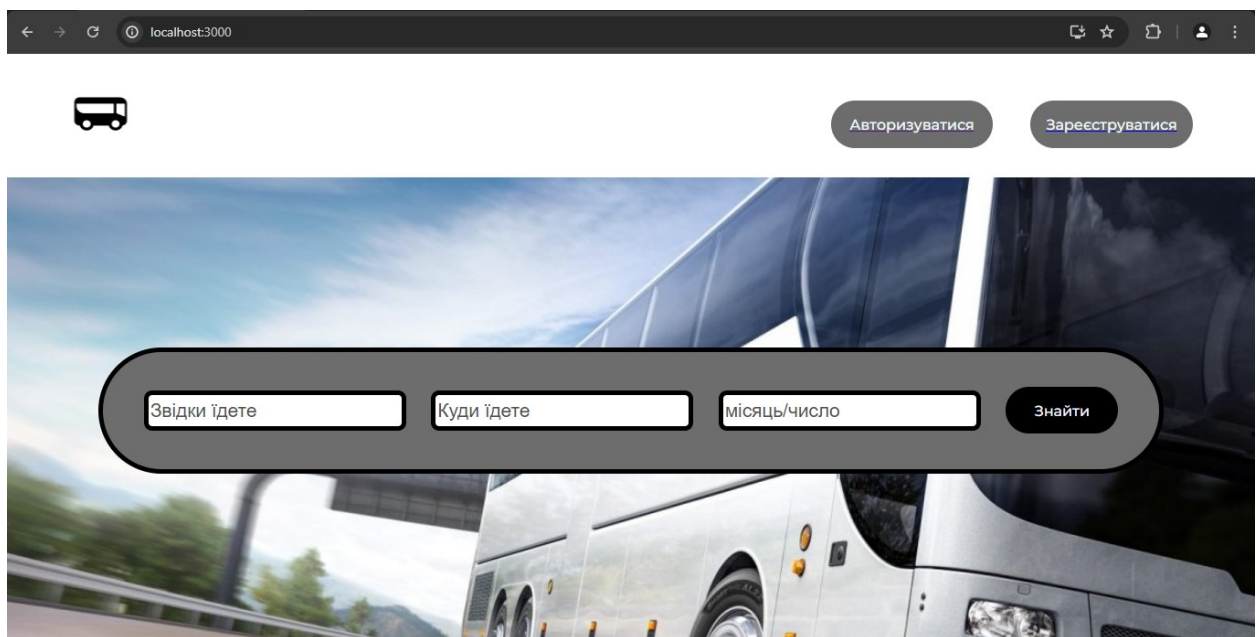


Рисунок 6.1 – Головна сторінка.

Тепер у користувача є два варіанти: авторизуватися, або залишитися не авторизованим, і потім вводити свої дані самому. У разі вибору першого варіанту треба натиснути на кнопку «Авторизуватися». Ця кнопка переадресує клієнта на сторінку авторизації, яку можна побачити далі на рисунку 6.2.

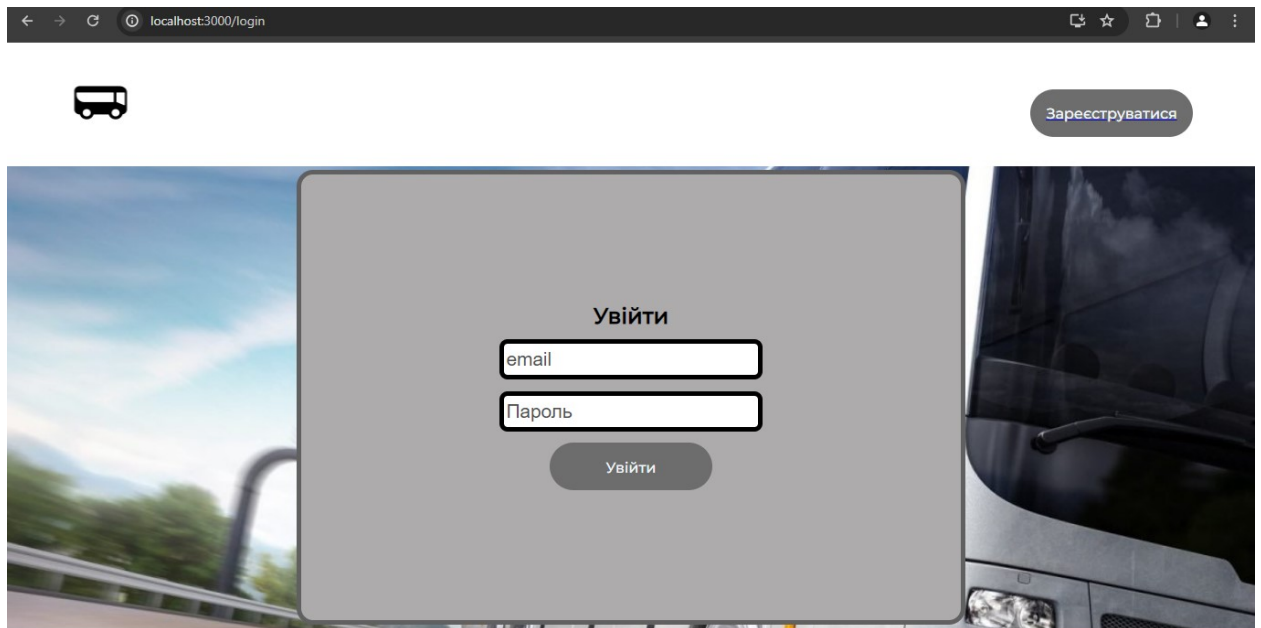


Рисунок 6.2 – Сторінка авторизації.

Користувачу потрібно ввести свою електронну пошту та пароль, та натиснути кнопку «Увійти». Тоді система авторизує його і переадресує на головну сторінку. У разі якщо у користувача ще немає акаунту, потрібно натиснути на кнопку «Зареєструватися». Тоді його переадресує на сторінку реєстрації, яка зображена на рисунку 6.3.

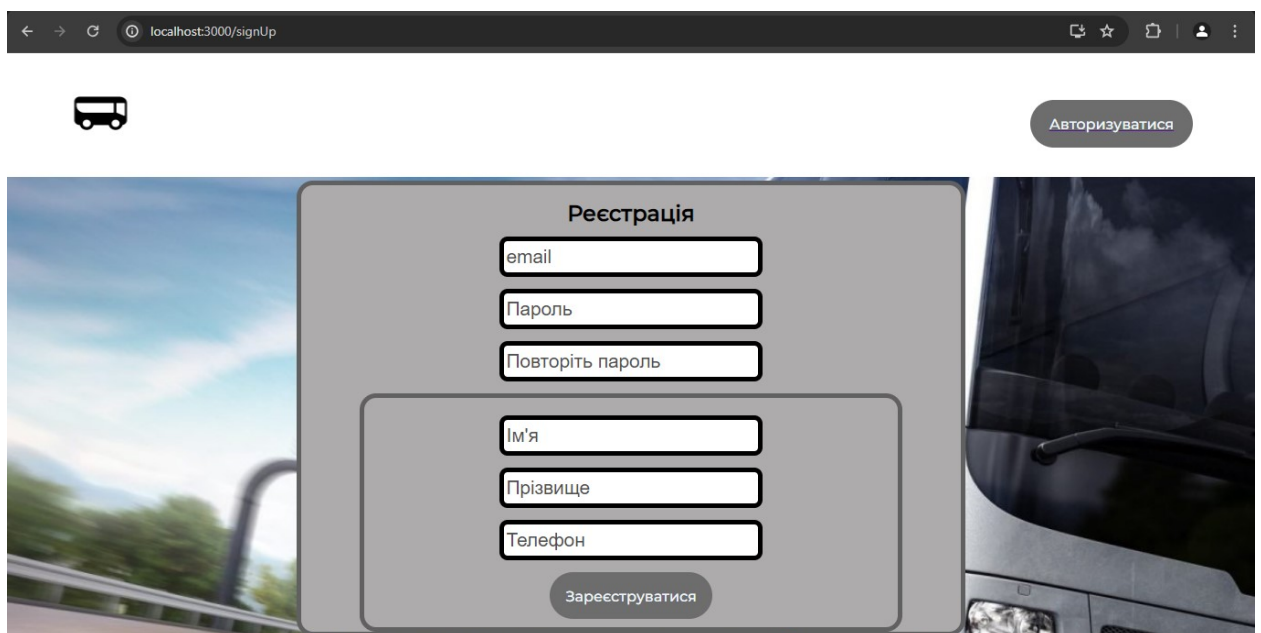


Рисунок 6.3 – Сторінка реєстрації.

На цій сторінці користувач має ввести свої дані та натиснути на кнопку «Зареєструватися». Після цього система збереже його особисті дані, авторизує та переадресує на головну сторінку.

У авторизованого користувача зміниться шапка сайту, і з'явиться можливість перейти до особистого кабінету. Це зображено на рисунку 6.4.

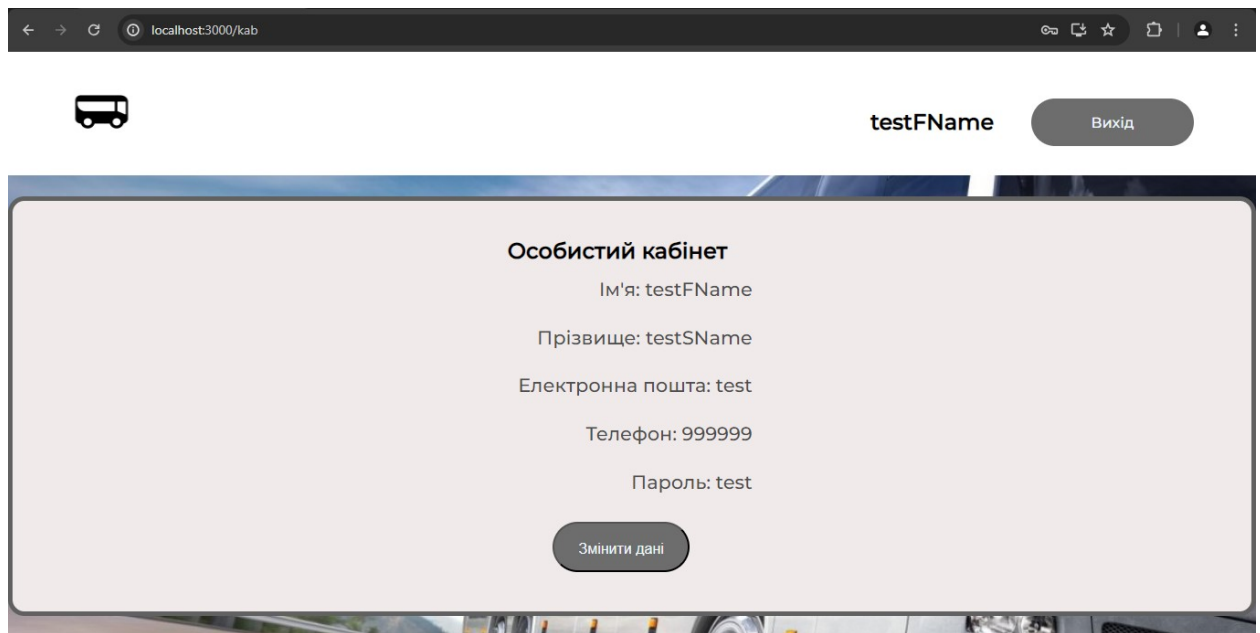


Рисунок 6.4 – Особистий кабінет.

В особистому кабінеті є можливість змінити дані користувача. Для цього потрібно натиснути кнопку «Змінити дані», ввести нові дані в поля, що після цього з'явилися, та зберегти їх, натиснувши на кнопку «Зберегти». Ця функція продемонстрована на рисунку 6.5.

Особистий кабінет

Ім'я: testFName

Прізвище: testSName

Електронна пошта: test

Телефон: 999999

Пароль: test

Зберегти

Рисунок 6.5 – Редагування особистих даних.

Щоб повернутися на головну потрібно натиснути на іконку в шапці сайту.

Після авторизації, або якщо користувач не захотів авторизуватись, він має ввести звідки, куди, і коли він хоче поїхати, та натиснути на кнопку «Знайти». Нижче сформується таблиця рейсів, що є на обрану дату. Це зображено на рисунку 6.6.

testFName Особистий кабінет Вихід

Запоріжжя Харків 01/01 Знайти

Маршрут	Пункт відправлення	Пункт прибуття	Час відправлення	Час прибуття	Тривалість поїздки	Ціна	Дата	
Запоріжжя-Харків	Запоріжжя	Харків	15:00	20:00	5 год.	2550 грн	2024-01-01	Оформити
Запоріжжя-Харків	Запоріжжя	Харків	18:00	23:00	5 год.	2550 грн	2024-01-01	Оформити

Рисунок 6.6 – Таблиця рейсів.

Коли користувач обере потрібний йому рейс, він має натиснути на кнопку «Оформити» у рядку його рейсу. Після цього система переадресує клієнта на сторінку бронювання, яка зображена на рисунку 6.7.

The screenshot shows a web browser at the address `localhost:3000/booking/10`. The page features a bus icon in the top left and two buttons: "Авторизуватися" (Log in) and "Зареєструватися" (Register). The main content area displays a booking summary for the route "Запоріжжя-Харків" (Zaporizhzhia-Kharkiv). The summary includes the following details:

Запоріжжя-Харків	
Пункт відправлення: Запоріжжя	Пункт прибуття: Харків
Час відправлення: 18:00	Час прибуття: 23:00
Час поїздки: 5 год.	Ціна: 2550 грн
Дата поїздки: 2024-01-01	

Below the summary is a dropdown menu for "Місце №19". To the right of the summary is a form for user registration with fields for "Ім'я" (Name), "Прізвище" (Surname), "Номер телефону" (Phone number), and "Email". A button labeled "Перейти до оплати" (Proceed to payment) is located below the registration fields.

Рисунок 6.7 – Сторінка бронювання квитків.

Тут потрібно обрати місце та ввести свої дані. Якщо користувач авторизований, особисті дані будуть введені автоматично, а йому залишиться тільки обрати місце та натиснути на кнопку «Перейти до оплати».

ВИСНОВКИ

В результаті виконання дипломної роботи був проведений детальний аналіз вимог до системи та її функціональних можливостей, і на основі цього розроблений вебзастосунок для бронювання квитків на автобуси.

Проведений аналіз включав:

- визначення потреб користувачів;
- аналіз ринку;
- технічний аналіз;
- аналіз результатів тестування.

Розроблений застосунок надає користувачам можливість легко і швидко бронювати квитки через інтернет, використовуючи зручний та інтуїтивно зрозумілий інтерфейс. Застосунок забезпечує необхідний функціонал для реєстрації користувачів, вибору рейсів, бронювання місць, здійснення оплати та отримання електронних квитків.

Проведене тестування показало високу ефективність та надійність розробленого програмного забезпечення. Вебзастосунок відповідає всім поставленим вимогам та може бути впроваджений для використання в реальних умовах.

Таким чином, розроблений вебзастосунок не тільки полегшує процес бронювання квитків на автобуси, але й підвищує ефективність роботи транспортних компаній, забезпечуючи високий рівень задоволеності користувачів.

Результати роботи апробовані на науково-практичній конференції «Тиждень науки-2024», яка дистанційно проводилася у Національному університеті «Запорізька політехніка» у 2024 році.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Електронна комерція [Електрон. ресурс]. – Режим доступу:
https://ua.prostopravo.com.ua/prava_biznesu/dogovirni_vidnosini/statti/zakon_pro_elektronnu_komertsiyu#1
2. Електронна торгівля [Електрон. ресурс]. – Режим доступу:
https://protocol.ua/ua/pro_elektronnu_komertsiyu/
3. Kayak.com [Електрон. ресурс]. – Режим доступу:
<https://www.ua.kayak.com>
4. Expedia.com [Електрон. ресурс]. – Режим доступу:
<https://www.expedia.com>
5. Skyskanner.com [Електрон. ресурс]. – Режим доступу:
<https://www.skyscanner.com.ua>
6. Node.js [Електрон. ресурс]. – Режим доступу:
<https://uk.wikipedia.org/wiki/Node.js>
7. HTML [Електрон. ресурс]. – Режим доступу:
<https://uk.wikipedia.org/wiki/HTML>
8. CSS [Електрон. ресурс]. – Режим доступу:
<https://uk.wikipedia.org/wiki/CSS>
9. JavaScript [Електрон. ресурс]. – Режим доступу:
<https://uk.wikipedia.org/wiki/JavaScript>
10. React [Електрон. ресурс]. – Режим доступу:
<https://uk.wikipedia.org/wiki/React>
11. PostgreSQL [Електрон. ресурс]. – Режим доступу:
<https://uk.wikipedia.org/wiki/PostgreSQL>

ДОДАТОК А
Текст програми

Серверна частина

Server.js

```
const express = require('express');
const app = express();
const port = process.env.PORT || 5000;

const placeRouter = require('./tickets/src/server/routes/place.routes');
const bus_stationRouter =
require('./tickets/src/server/routes/bus_station.routes');
const cityRouter = require('./tickets/src/server/routes/city.routes');
const clientRouter = require('./tickets/src/server/routes/client.routes');
const routesRouter = require('./tickets/src/server/routes/routes.routes');
const profilesRouter = require('./tickets/src/server/routes/profiles.routes');

app.use(express.json())

app.use('/api', placeRouter)
app.use('/api', bus_stationRouter)
app.use('/api', cityRouter)
app.use('/api', clientRouter)
app.use('/api', routesRouter)
app.use('/api', profilesRouter)

app.listen(port, () => console.log(`Listening on port ${port}`));

app.get('/express_backend', (req, res) => {
  res.send({ express: 'YOUR EXPRESS BACKEND IS CONNECTED TO REACT' });
});
```

Контролери

City.controller.js

```
const db = require('../db')

class CityController {
  async createCity(req, res){
    const {city_name} = req.body
    const newCity = await db.query('INSERT INTO city (city_name) VALUES ($1)
RETURNING *', [city_name])
    res.json(newCity.rows[0])
  }
  async getCities(req, res){
    const cities = await db.query('SELECT * FROM city');
    res.send(cities.rows);
  }
}
```

```

    }
    async getOneCity(req, res){
        const id = req.params.id
        const city = await db.query('SELECT * FROM city WHERE id = $1', [id])
        res.json(city.rows[0])
    }
    async updateCity(req, res){
        const {id, city_name} = req.body
        const updateCity = await db.query('UPDATE city SET city_name = $1 WHERE
id = $2 RETURNING *', [city_name, id])
        res.json(updateCity.rows[0])
    }
    async deleteCity(req, res){
        const id = req.params.id
        const deleteCity = await db.query('DELETE FROM city WHERE id = $1', [id])
        res.json(deleteCity.rows[0])
    }
}

module.exports = new CityController()

```

bus_station.controller.js

```

const db = require('../db')

class Bus_stationController {
    async createBus_station(req, res){
        const {bus_station_name, city_id, bus_station_address} = req.body
        const newBus_station = await db.query('INSERT INTO bus_station
(bus_station_name, city_id, bus_station_address) VALUES ($1, $2, $3) RETURNING
*', [bus_station_name, city_id, bus_station_address])
        res.json(newBus_station.rows[0])
    }
    async getBus_stations(req, res){
        const bus_stations = await db.query('SELECT * FROM bus_station')
        res.json(bus_stations.rows)
    }
    async postBus_stationId(req, res){
        const {bus_station_name} = req.body
        const bus_stations = await db.query('SELECT bus_station.id FROM
bus_station WHERE bus_station_name = $1', [bus_station_name])
        res.json(bus_stations.rows)
    }
    async getOneBus_station(req, res){
        const id = req.params.id
        const bus_station = await db.query('SELECT * FROM bus_station WHERE id =
$1', [id])
        res.json(bus_station.rows[0])
    }
    async updateBus_station(req, res){

```

```

    const {id, bus_station_name, city_id, bus_station_address} = req.body
    const updateBus_station = await db.query('UPDATE bus_station SET
bus_station_name = $1, city_id = $2, bus_station_address = $3 WHERE id = $4
RETURNING *', [bus_station_name, city_id, bus_station_address, id])
    res.json(updateBus_station.rows[0])
  }
  async deleteBus_station(req, res){
    const id = req.params.id
    const deleteBus_station = await db.query('DELETE FROM bus_station WHERE
id = $1', [id])
    res.json(deleteBus_station.rows[0])
  }
}

module.exports = new Bus_stationController()

```

client.controller.js

```

const db = require('../db')

class ClientController {
  async createClient(req, res){
    const {id_place, first_name, second_name, phone_number, email} = req.body
    const newClient = await db.query('INSERT INTO client (id_place,
first_name, second_name, phone_number, email) VALUES ($1, $2, $3, $4, $5)
RETURNING *', [id_place, first_name, second_name, phone_number, email])
    res.json(newClient.rows[0])
  }
  async getClients(req, res){
    const clients = await db.query('SELECT * FROM client')
    res.json(clients.rows)
  }
  async getOneClient(req, res){
    const id = req.params.id
    const client = await db.query('SELECT * FROM client WHERE id = $1', [id])
    res.json(client.rows[0])
  }
  async updateClient(req, res){
    const {id, id_place, first_name, second_name, phone_number, email} =
req.body
    const updateClient = await db.query('UPDATE client SET id_place = $1,
first_name = $2, second_name = $3, phone_number = $4, email = $5 WHERE id = $6
RETURNING *', [id_place, first_name, second_name, phone_number, email, id])
    res.json(updateClient.rows[0])
  }
  async deleteClient(req, res){
    const id = req.params.id
    const deleteClient = await db.query('DELETE FROM client WHERE id = $1',
[id])
    res.json(deleteClient.rows[0])
  }
}

```

```

    }
  }

  module.exports = new ClientController()

```

place.controller.js

```

const db = require('../db')

class PlaceController {
  async createPlace(req, res){
    const {id_route, place_number, place_existence, place_date} = req.body
    const newPlace = await db.query('INSERT INTO place (id_route,
place_number, place_existence, place_date) VALUES ($1, $2, $3, $4) RETURNING *',
[id_route, place_number, place_existence, place_date])
    res.json(newPlace.rows[0])
  }
  async getPlaces(req, res) {
    try {
      const places = await db.query('SELECT * FROM place');
      const formattedPlaces = places.rows.map(place => {
        const date = new Date(place.place_date);
        date.setMinutes(date.getMinutes() - date.getTimezoneOffset());
        return {
          id: place.id,
          id_route: place.id_route,
          place_number: place.place_number,
          place_existence: place.place_existence,
          place_date: date.toISOString().split('T')[0]
        };
      });
      res.json(formattedPlaces);
    } catch (error) {
      console.error('Error fetching places:', error);
      res.status(500).json({ error: 'Internal server error' });
    }
  }

  async postBookingPlaces(req, res) {
    try {
      const {id_route, place_date} = req.body
      const places = await db.query('SELECT * FROM place WHERE id_route =
$1 AND place_date = $2', [id_route, place_date]);
      //console.log(places.rows)
      const formattedPlaces = places.rows.map(place => {
        // Преобразование даты с учетом временной зоны
        const date = new Date(place.place_date);
        date.setMinutes(date.getMinutes() - date.getTimezoneOffset());
        return {
          id: place.id,

```

```

        id_route: place.id_route,
        place_number: place.place_number,
        place_existence: place.place_existence,
        place_date: date.toISOString().split('T')[0]
    };
    });
    res.json(formattedPlaces);
} catch (error) {
    console.error('Error fetching places:', error);
    res.status(500).json({ error: 'Internal server error' });
}
}

async getOnePlace(req, res){
    const id = req.params.id
    const place = await db.query('SELECT * FROM place WHERE id = $1', [id])
    res.json(place.rows[0])
}

async updatePlace(req, res){
    const {id, id_route, place_number, place_existence, place_date} =
req.body
    const updatePlace = await db.query('UPDATE place SET id_route = $1,
place_number = $2, place_existence = $3, place_date = $4 WHERE id = $5 RETURNING
*', [id_route, place_number, place_existence, place_date, id])
    res.json(updatePlace.rows[0])
}

async buyPlace(req, res) {
    const { id_route, place_number, place_date } = req.body;
    try {
        const updatePlace = await db.query(
            'UPDATE place SET place_existence = \'false\' WHERE id_route = $1
AND place_number = $2 AND place_date = $3 RETURNING *',
            [id_route, place_number, place_date]
        );
        if (updatePlace.rows.length === 0) {
            return res.status(404).json({ error: 'Place not found' });
        }
        res.json(updatePlace.rows[0]);
    } catch (error) {
        console.error('Error updating place:', error);
        res.status(500).json({ error: 'Internal Server Error' });
    }
}

async deletePlace(req, res){
    const id = req.params.id
    const deletePlace = await db.query('DELETE FROM place WHERE id = $1',
[id])
    res.json(deletePlace.rows[0])
}
}

```

```
module.exports = new PlaceController()
```

routes.controller.js

```
const db = require('../db')

class RouteController {
  async createRoute(req, res){
    const {route_name, point_of_departure, point_of_arrival,
time_of_departure, time_of_arrival, duration, price, route_existence} = req.body
    const newRoute = await db.query('INSERT INTO routes (route_name,
point_of_departure, point_of_arrival, time_of_departure, time_of_arrival,
duration, price, route_existence) VALUES ($1, $2, $3, $4, $5, $6, $7, $8)
RETURNING *', [route_name, point_of_departure, point_of_arrival,
time_of_departure, time_of_arrival, duration, price, route_existence])
    res.json(newRoute.rows[0])
  }
  async getRoutes(req, res){
    const routes = await db.query('SELECT * FROM routes')
    res.json(routes.rows)
  }
  async postSearchRoutes(req, res) {
    try {
      const { point_of_departure, point_of_arrival, place_date } =
req.body;
      const routes = await db.query(`
        SELECT
          routes.id,
          routes.route_name,
          departure.bus_station_name AS point_of_departure,
          arrival.bus_station_name AS point_of_arrival,
          place.place_date,
          routes.time_of_departure,
          routes.time_of_arrival,
          routes.duration,
          routes.price
        FROM
          (bus_station AS departure
          INNER JOIN (routes
          INNER JOIN bus_station AS arrival ON routes.point_of_arrival
= arrival.id)
          ON departure.id = routes.point_of_departure)
          INNER JOIN place ON routes.id = place.id_route
        WHERE
          routes.point_of_departure = $1
          AND routes.point_of_arrival = $2
          AND place.place_date = $3
          AND routes.route_existence = 'true'
        GROUP BY
```

```

        routes.id,
        routes.route_name,
        departure.bus_station_name,
        arrival.bus_station_name,
        place.place_date,
        routes.time_of_departure,
        routes.time_of_arrival,
        routes.duration,
        routes.price
    `), [point_of_departure, point_of_arrival, place_date]);

    const formattedRoutes = routes.rows.map(route => {
        // Преобразование даты с учетом временной зоны
        const date = new Date(route.place_date);
        date.setMinutes(date.getMinutes() - date.getTimezoneOffset());
        return {
            id: route.id,
            route_name: route.route_name,
            point_of_departure: route.point_of_departure,
            point_of_arrival: route.point_of_arrival,
            place_date: date.toISOString().split('T')[0],
            time_of_departure: route.time_of_departure,
            time_of_arrival: route.time_of_arrival,
            duration: route.duration,
            price: route.price
        };
    });

    res.json(formattedRoutes);
} catch (error) {
    console.error('Error fetching routes:', error);
    res.status(500).json({ error: 'Internal server error' });
}

}

async getOneRoute(req, res){
    const id = req.params.id
    const route = await db.query('SELECT * FROM routes WHERE id = $1', [id])
    res.json(route.rows[0])
}

async updateRoute(req, res){
    const {id, route_name, point_of_departure, point_of_arrival,
time_of_departure, time_of_arrival, duration, price, route_existence} = req.body
    const updateRoute = await db.query('UPDATE routes SET route_name = $1,
point_of_departure = $2, point_of_arrival = $3, time_of_departure = $4,
time_of_arrival = $5, duration = $6, price = $7, route_existence = $8 WHERE id =
$9 RETURNING *', [route_name, point_of_departure, point_of_arrival,
time_of_departure, time_of_arrival, duration, price, route_existence, id])
    res.json(updateRoute.rows[0])
}

```

```

    async deleteRoute(req, res){
        const id = req.params.id
        const deleteRoute = await db.query('DELETE FROM routes WHERE id = $1',
[id])
        res.json(deleteRoute.rows[0])
    }
}

module.exports = new RouteController()

```

profiles.controller.js

```

const db = require('../db')

class ProfileController {
    async createProfile(req, res){
        const {first_name, second_name, profile_password, phone_number, email} =
req.body
        const newProfile = await db.query('INSERT INTO profiles (first_name,
second_name, profile_password, phone_number, email) VALUES ($1, $2, $3, $4, $5)
RETURNING *', [first_name, second_name, profile_password, phone_number, email])
        res.json(newProfile.rows[0])
    }
    async getProfiles(req, res){
        const profiles = await db.query('SELECT * FROM profiles')
        res.json(profiles.rows)
    }
    async getOneProfile(req, res){
        const id = req.params.id
        const profile = await db.query('SELECT * FROM profiles WHERE id = $1',
[id])
        res.json(profile.rows[0])
    }
    async login(req, res){
        const {email, profile_password} = req.body
        const profile = await db.query('SELECT * FROM profiles WHERE email = $1
AND profile_password = $2', [email, profile_password])
        res.json(profile.rows[0])
    }
    async updateProfile(req, res){
        const {id, first_name, second_name, profile_password, phone_number,
email} = req.body
        const updateProfile = await db.query('UPDATE profiles SET first_name =
$1, second_name = $2, profile_password = $3, phone_number = $4, email = $5 WHERE
id = $6 RETURNING *', [first_name, second_name, profile_password, phone_number,
email, id])
        res.json(updateProfile.rows[0])
    }
    async deleteProfile(req, res){
        const id = req.params.id

```

```

        const deleteProfile = await db.query('DELETE FROM profiles WHERE id =
$1', [id])
        res.json(deleteProfile.rows[0])
    }
}

module.exports = new ProfileController()

```

Poymepu Bus_station.otes.js

```

const Router = require('express')
const bus_stationController = require('../controller/bus_station.controller')
const router = new Router()

router.post('/bus_station', bus_stationController.createBus_station)
router.get('/bus_station', bus_stationController.getBus_stations)
router.post('/bus_station-id', bus_stationController.postBus_stationId)
router.get('/bus_station/:id', bus_stationController.getOneBus_station)
router.put('/bus_station', bus_stationController.updateBus_station)
router.delete('/bus_station/:id', bus_stationController.deleteBus_station)

module.exports = router

```

City.roytes.js

```

const Router = require('express')
const cityController = require('../controller/city.controller')
const router = new Router()

router.post('/city', cityController.createCity)
router.get('/city', cityController.getCityes)
router.get('/city/:id', cityController.getOneCity)
router.put('/city', cityController.updateCity)
router.delete('/city/:id', cityController.deleteCity)

module.exports = router

```

Client.routes.js

```

const Router = require('express')
const clientController = require('../controller/client.controller')
const router = new Router()

router.post('/client', clientController.createClient)
router.get('/client', clientController.getClients)
router.get('/client/:id', clientController.getOneClient)
router.put('/client', clientController.updateClient)
router.delete('/client/:id', clientController.deleteClient)

```

```
module.exports = router
```

Place.routes.js

```
const Router = require('express')
const placeController = require('../controller/place.controller')
const router = new Router()

router.post('/place', placeController.createPlace)
router.get('/place', placeController.getPlaces)
router.post('/place-booking', placeController.postBookingPlaces)
router.get('/place/:id', placeController.getOnePlace)
router.put('/place', placeController.updatePlace)
router.put('/place-buy', placeController.buyPlace)
router.delete('/place/:id', placeController.deletePlace)

module.exports = router
```

Profiles.routes.js

```
const Router = require('express')
const profileController = require('../controller/profiles.controller')
const router = new Router()

router.post('/profile', profileController.createProfile)
router.get('/profile', profileController.getProfiles)
router.post('/profile-login', profileController.login)
router.get('/profile/:id', profileController.getOneProfile)
router.put('/profile', profileController.updateProfile)
router.delete('/profile/:id', profileController.deleteProfile)

module.exports = router
```

routes.routes.js

```
const Router = require('express')
const routeController = require('../controller/routes.controller')
const router = new Router()

router.post('/routes', routeController.createRoute)
router.get('/routes', routeController.getRoutes)
router.post('/search-routes', routeController.postSearchRoutes)
router.get('/routes/:id', routeController.getOneRoute)
router.put('/routes', routeController.updateRoute)
router.delete('/routes/:id', routeController.deleteRoute)
```

```
module.exports = router
```

Керування БД

Db.js

```
const Pool = require('pg').Pool;

const pool = new Pool({
  user: "postgres",
  password: "listok123",
  host: "localhost",
  port: 5432,
  database: "tickets",
  client_encoding: 'utf8'
});

module.exports = pool;
```

Клієнтська частина

App.js

```
import React, { useEffect, useState } from 'react';
import { BrowserRouter, Routes, Route } from "react-router-dom";
import { UserProvider } from './components/UserContext';

import StartPage from './components/StartPage';
import Admin from './components/Admin';
import SignUp from './components/SignUp';
import Login from './components/Login';
import Booking from './components/Booking';
import Kab from './components/Kab';

import './App.css';

function App() {

  const [state, setState] = useState(null);

  const callBackendAPI = async () => {
    const response = await fetch('/express_backend');
    const body = await response.json();

    if (response.status !== 200) {
      throw Error(body.message)
    }
  }
}
```

```

    }
    return body;
  };

  useEffect(() => {
    callBackendAPI()
      .then(res => setState(res.express))
      .catch(err => console.log(err));
  }, [])

  return (
    <body>
      <BrowserRouter>
        <UserProvider>
          <Routes>
            <Route path="/" element={<StartPage/>}/>
            <Route path="/admin" element={<Admin/>}/>
            <Route path="/signUp" element={<SignUp/>}/>
            <Route path="/login" element={<Login/>}/>
            <Route path="/booking/:id" element={<Booking/>}/>
            <Route path="/kab" element={<Kab/>}/>
          </Routes>
        </UserProvider>
      </BrowserRouter>
    </body>
  );
}

export default App;

```

StartPage.jsx

```

import React, { useState, useEffect, useContext } from "react";
import './styles/reset.css';
import './styles/Style.css';
import axios from "axios";
import { useLocation, useNavigate, Link } from "react-router-dom";
import { UserContext } from './UserContext';

const StartPage = () => {
  const { userData, setUserData } = useContext(UserContext);

  const [busStationsFrom, setBusStationsFrom] = useState(null);
  const [busStationsTo, setBusStationsTo] = useState(null);
  const [searchResult, setSearchResult] = useState([]);

  const [from, setFrom] = useState('');

```

```

const [to, setTo] = useState('');
const [date, setDate] = useState('');
const [dateF, setDateF] = useState('');

const [table, setTable] = useState(true);

const navigate = useNavigate();

const handleFromChange = (event) => {
  setFrom(event.target.value);
};

const handleToChange = (event) => {
  setTo(event.target.value);
};

const handleDateChange = (event) => {
  setDate(event.target.value);
};

useEffect(() => {
  const currentYear = new Date().getFullYear();
  setDateF(`${currentYear}/${date}`);
}, [date]);

useEffect(() => {
  console.log(searchResult);
}, [searchResult]);

async function fetchBusStationsFrom() {
  const busStationName = from;
  const response = await axios.post('http://localhost:3000/api/bus_station-id', { "bus_station_name": busStationName });
  const { id } = response.data[0];
  return id;
}

async function fetchBusStationsTo() {
  try {
    const busStationName = to;
    const response = await
    axios.post('http://localhost:3000/api/bus_station-id', { "bus_station_name":
    busStationName });
    const { id } = response.data[0];
    return id;
  } catch {
    alert('Немає таких рейсів');
  }
}

```

```

    async function fetchSearch(point_of_departure, point_of_arrival, place_date)
    {
        try {
            console.log('date=', place_date);
            const response = await axios.post('http://localhost:3000/api/search-
routes', { "point_of_departure": point_of_departure, "point_of_arrival":
point_of_arrival, "place_date": place_date });
            setSearchResult(response.data);
        } catch {
            alert('Немає таких рейсів');
        }
    }

    const searchClick = async () => {
        try {
            setTable(false);
            const [fromId, toId] = await Promise.all([fetchBusStationsFrom(),
fetchBusStationsTo()]);
            setBusStationsFrom(fromId);
            setBusStationsTo(toId);
            await fetchSearch(fromId, toId, dateF);
        } catch (error) {
            console.error("Error fetching data", error);
            alert('Немає білетів');
        }
    };

    useEffect(() => {
        console.log(busStationsFrom);
    }, [busStationsFrom]);

    useEffect(() => {
        console.log(busStationsTo);
    }, [busStationsTo]);

    const handleBookingClick = (id, route_name, point_of_departure,
point_of_arrival, time_of_departure, time_of_arrival, duration, price,
place_date) => {
        navigate(`/booking/${id}`, { state: { route_name, point_of_departure,
point_of_arrival, time_of_departure, time_of_arrival, duration, price, place_date
} });
    };

    const handleLogout = () => {
        setUserData(null);
    };

    return (
        <>
        <header className="header">
            

```

```

        <nav className="nav-block">
          <ul className="nav-menu">
            {userData ? (
              <>
                <li className="nav-
item"><h1>{userData.first_name}</h1></li>
                <li>
                  <Link className="nav-item button search-
header-button" to="/kab">Особистий кабінет</Link>
                </li>
                <li className="nav-item button search-header-
button" onClick={handleLogout}>Вихід</li>
              </>
            ) : (
              <>
                <Link to="/login"><li className="nav-item button
search-header-button">Авторизуватися</li></Link>
                <Link to="/signUp"><li className="nav-item button
search-header-button">Зареєструватися</li></Link>
              </>
            )}
          </ul>
        </nav>
      </header>
      <div className="bg-main" />
      <div className="container-main">
        <ul className="search-routes">
          <li>
            <input
              type="text"
              className="search-input"
              placeholder="Звідки їдете"
              value={from}
              onChange={handleFromChange}
            />
          </li>
          <li>
            <input
              type="text"
              className="search-input"
              placeholder="Куди їдете"
              value={to}
              onChange={handleToChange}
            />
          </li>
          <li>
            <input
              type="text"
              className="search-input"
              placeholder="місяць/число"
              value={date}
            />
          </li>
        </ul>
      </div>
    </div>
  </div>
</div>

```

```

        onChange={handleDateChange}
      />
    </li>
    <li className="nav-item button search-button"
onClick={searchClick}>Знайти</li>
  </ul>
</div>
<div className="container-main">
  <table && (
    <section className="search-block">
      <div className="search-table">
        <table>
          <thead>
            <tr>
              <th>Маршрут</th>
              <th>Пункт відправлення</th>
              <th>Пункт прибуття</th>
              <th>Час відправлення</th>
              <th>Час прибуття</th>
              <th>Тривалість поїздки</th>
              <th>Ціна</th>
              <th>Дата</th>
              <th></th>
            </tr>
          </thead>
          <tbody>
            {searchResult.map(({ id, route_name,
point_of_departure, point_of_arrival, time_of_departure, time_of_arrival,
duration, price, place_date }, index) => (
              <tr key={index}>
                <td>{route_name}</td>
                <td>{point_of_departure}</td>
                <td>{point_of_arrival}</td>
                <td>{time_of_departure}</td>
                <td>{time_of_arrival}</td>
                <td>{duration}</td>
                <td>{price}</td>
                <td>{place_date}</td>
                <td><button className="tab-button"
onClick={() => handleBookingClick(id, route_name, point_of_departure,
point_of_arrival, time_of_departure, time_of_arrival, duration, price,
place_date)}>Оформити</button></td>
              </tr>
            ))}
          </tbody>
        </table>
      </div>
    </section>
  )}
</div>
</>

```

```

    );
  };

export default StartPage;

```

Admin.jsx

```

import React, { useState } from "react";
import './styles/reset.css';
import './styles/Style.css';
import axios from "axios";
import { Link } from "react-router-dom";

const Admin = () => {

  const [state, setState] = useState([]);
  const [originalState, setOriginalState] = useState([]);

  async function fetchCityes() {
    const response = await axios.get('http://localhost:3000/api/city')
    console.log(response.data)
    setState(response.data)
    setOriginalState(response.data)
  }

  async function fetchBus_stations() {
    const response = await axios.get('http://localhost:3000/api/bus_station')
    console.log(response.data)
    setState(response.data)
    setOriginalState(response.data)
  }

  async function fetchRoutes() {
    const response = await axios.get('http://localhost:3000/api/routes')
    console.log(response.data)
    setState(response.data)
    setOriginalState(response.data)
  }

  async function fetchPlaces() {
    const response = await axios.get('http://localhost:3000/api/place')
    console.log(response.data)
    setState(response.data)
    setOriginalState(response.data)
  }
}

```

```

async function fetchClients() {
  const response = await axios.get('http://localhost:3000/api/client')
  console.log(response.data)
  setState(response.data)
  setOriginalState(response.data)
}

const [cityes, setCityes] = useState(true);
const [stations, setStations] = useState(true);
const [busRoutes, setBusRoutes] = useState(true);
const [clients, setClients] = useState(true);
const [places, setPlaces] = useState(true);

const Cityes = function(){
  setCityes(false)
  setStations(true)
  setBusRoutes(true)
  setClients(true)
  setPlaces(true)
  setState([])
  fetchCityes()
}

const Stations = function(){
  setCityes(true)
  setStations(false)
  setBusRoutes(true)
  setClients(true)
  setPlaces(true)
  setState([])
  fetchBus_stations()
}

const BusRoutes = function(){
  setCityes(true)
  setStations(true)
  setBusRoutes(false)
  setClients(true)
  setPlaces(true)
  setState([])
  fetchRoutes()
}

const Clients = function(){
  setCityes(true)
  setStations(true)
  setBusRoutes(true)

```

```

        setClients(false)
        setPlaces(true)
        setState([])
        fetchClients()
    }

    const Places = function(){
        setCityes(true)
        setStations(true)
        setBusRoutes(true)
        setClients(true)
        setPlaces(false)
        setState([])
        fetchPlaces()
    }

    const [inputCityValues, setInputCityValues] = useState(['', '']);
    const [inputStationValues, setInputStationValues] = useState(['', '', '', '']);
    const [inputRouteValues, setInputRouteValues] = useState(['', '', '', '', '', '']);
    const [inputClientValues, setInputClientValues] = useState(['', '', '', '', '']);
    const [inputPlaceValues, setInputPlaceValues] = useState(['', '', '', '']);

    const placeholdersCity = ["id", "Місто"];
    const placeholdersStation = ["id", "Станція", "id Міста", "Адреса станції"];
    const placeholdersRoute = ["id", "Маршрут", "Пункт відправлення", "Пункт прибуття", "Час відправлення", "Час прибуття", "Тривалість поїздки", "Ціна", "Наявність"];
    const placeholdersClient = ["id", "№ білета", "Ім'я", "Прізвище", "Номер телефону", "email"];
    const placeholdersPlace = ["id", "№ маршруту", "№ місця", "Наявність", "Рік-місяць-число"];

    const handleCityChange = (index, event) => {
        const newInputCityValues = [...inputCityValues];
        newInputCityValues[index] = event.target.value;
        setInputCityValues(newInputCityValues);
    };

    const handleCityClear = () => {
        setInputCityValues(['', '']);
    };

    const handleStationChange = (index, event) => {
        const newInputStationValues = [...inputStationValues];
        newInputStationValues[index] = event.target.value;
        setInputStationValues(newInputStationValues);
    };

```

```

const handleStationClear = () => {
  setInputStationValues(['', '', '', '']);
};

const handleRouteChange = (index, event) => {
  const newInputRouteValues = [...inputRouteValues];
  newInputRouteValues[index] = event.target.value;
  setInputRouteValues(newInputRouteValues);
};

const handleRouteClear = () => {
  setInputRouteValues(['', '', '', '', '', '', '', '']);
};

const handleClientChange = (index, event) => {
  const newInputClientValues = [...inputClientValues];
  newInputClientValues[index] = event.target.value;
  setInputClientValues(newInputClientValues);
};

const handleClientClear = () => {
  setInputClientValues(['', '', '', '', '']);
};

const handlePlaceChange = (index, event) => {
  const newInputPlaceValues = [...inputPlaceValues];
  newInputPlaceValues[index] = event.target.value;
  setInputPlaceValues(newInputPlaceValues);
};

const handlePlaceClear = () => {
  setInputPlaceValues(['', '', '', '']);
};

const handleCityRowClick = (id, city_name) => {
  setInputCityValues([id, city_name]);
};

const handleStationRowClick = (id, bus_station_name, city_id,
bus_station_address) => {
  setInputStationValues([id, bus_station_name, city_id,
bus_station_address]);
};

const handleRouteRowClick = (id, route_name, point_of_departure,
point_of_arrival, time_of_departure, time_of_arrival, duration, price,
route_existence) => {
  setInputRouteValues([id, route_name, point_of_departure,
point_of_arrival, time_of_departure, time_of_arrival, duration, price,
route_existence]);
};

const handleClientRowClick = (id, id_place, first_name, second_name,
phone_number, email) => {

```

```

        setInputClientValues([id, id_place, first_name, second_name,
phone_number, email]);
    };

    const handlePlaceRowClick = (id, id_route, place_number, place_existence,
place_date) => {
        setInputPlaceValues([id, id_route, place_number, place_existence,
place_date]);
    };

    const applyCityFilter = () => {
        const filtered = originalState.filter(item =>
            (inputCityValues[0] === '' ||
            item.id.toString().includes(inputCityValues[0])) &&
            (inputCityValues[1] === '' ||
            item.city_name.includes(inputCityValues[1]))
        );
        setState(filtered);
    };

    const applyStationFilter = () => {
        const filtered = originalState.filter(item =>
            (inputStationValues[0] === '' ||
            item.id.toString().includes(inputStationValues[0])) &&
            (inputStationValues[1] === '' ||
            item.bus_station_name.includes(inputStationValues[1])) &&
            (inputStationValues[2] === '' ||
            item.city_id.toString().includes(inputStationValues[2])) &&
            (inputStationValues[3] === '' ||
            item.bus_station_address.includes(inputStationValues[3]))
        );
        setState(filtered);
    };

    const applyRouteFilter = () => {
        const filtered = originalState.filter(item =>
            (inputRouteValues[0] === '' ||
            item.id.toString().includes(inputRouteValues[0])) &&
            (inputRouteValues[1] === '' ||
            item.route_name.includes(inputRouteValues[1])) &&
            (inputRouteValues[2] === '' ||
            item.point_of_departure.toString().includes(inputRouteValues[2])) &&
            (inputRouteValues[3] === '' ||
            item.point_of_arrival.toString().includes(inputRouteValues[3])) &&
            (inputRouteValues[4] === '' ||
            item.time_of_departure.includes(inputRouteValues[4])) &&
            (inputRouteValues[5] === '' ||
            item.time_of_arrival.includes(inputRouteValues[5])) &&
            (inputRouteValues[6] === '' ||
            item.duration.includes(inputRouteValues[6])) &&

```

```

        (inputRouteValues[7] === '' ||
item.price.toString().includes(inputRouteValues[7])) &&
        (inputRouteValues[8] === '' ||
item.route_existence.toString().includes(inputRouteValues[8]))
    );
    setState(filtered);
};

const applyClientFilter = () => {
    const filtered = originalState.filter(item =>
        (inputClientValues[0] === '' ||
item.id.toString().includes(inputClientValues[0])) &&
        (inputClientValues[1] === '' ||
item.id_place.toString().includes(inputClientValues[1])) &&
        (inputClientValues[2] === '' ||
item.first_name.includes(inputClientValues[2])) &&
        (inputClientValues[3] === '' ||
item.second_name.includes(inputClientValues[3])) &&
        (inputClientValues[4] === '' ||
item.phone_number.includes(inputClientValues[4])) &&
        (inputClientValues[5] === '' ||
item.email.includes(inputClientValues[5]))
    );
    setState(filtered);
};

const applyPlaceFilter = () => {
    const filtered = originalState.filter(item =>
        (inputPlaceValues[0] === '' ||
item.id.toString().includes(inputPlaceValues[0])) &&
        (inputPlaceValues[1] === '' ||
item.id_route.toString().includes(inputPlaceValues[1])) &&
        (inputPlaceValues[2] === '' ||
item.place_number.toString().includes(inputPlaceValues[2])) &&
        (inputPlaceValues[3] === '' ||
item.place_existence.toString().includes(inputPlaceValues[3])) &&
        (inputPlaceValues[4] === '' ||
item.place_date.includes(inputPlaceValues[4]))
    );
    setState(filtered);
};

const resetFilter = () => {
    setState(originalState);
};

async function createCity() {
    const cityName = inputCityValues[1];
    const response = await axios.post('http://localhost:3000/api/city',
{"city_name":cityName})
    console.log(response)
}

```

```

        fetchCityes()
    }
    async function updateCity() {
        const idCity = inputCityValues[0];
        const cityName = inputCityValues[1];
        const response = await axios.put('http://localhost:3000/api/city',
{"id":idCity, "city_name":cityName})
        console.log(response)
        fetchCityes()
    }
    async function deleteCity() {
        const idCity = inputCityValues[0];
        const response = await
axios.delete(`http://localhost:3000/api/city/${idCity}`)
        console.log(response)
        fetchCityes()
    }

    async function createStation() {
        const bus_station_name = inputStationValues[1];
        const city_id = inputStationValues[2];
        const bus_station_address = inputStationValues[3];
        const response = await
axios.post('http://localhost:3000/api/bus_station',
{"bus_station_name":bus_station_name, "city_id":city_id,
"bus_station_address":bus_station_address})
        console.log(response)
        fetchBus_stations()
    }
    async function updateStation() {
        const id = inputStationValues[0];
        const bus_station_name = inputStationValues[1];
        const city_id = inputStationValues[2];
        const bus_station_address = inputStationValues[3];
        const response = await axios.put('http://localhost:3000/api/bus_station',
{"id":id, "bus_station_name":bus_station_name, "city_id":city_id,
"bus_station_address":bus_station_address})
        console.log(response)
        fetchBus_stations()
    }
    async function deleteStation() {
        const id = inputStationValues[0];
        const response = await
axios.delete(`http://localhost:3000/api/bus_station/${id}`)
        console.log(response)
        fetchBus_stations()
    }

    async function createRoutes() {
        const route_name = inputRouteValues[1];

```

```

    const point_of_departure = inputRouteValues[2];
    const point_of_arrival = inputRouteValues[3];
    const time_of_departure = inputRouteValues[4];
    const time_of_arrival = inputRouteValues[5];
    const duration = inputRouteValues[6];
    const price = inputRouteValues[7];
    const route_existence = inputRouteValues[8];
    const response = await axios.post('http://localhost:3000/api/routes',
{"route_name":route_name, "point_of_departure":point_of_departure,
"point_of_arrival":point_of_arrival, "time_of_departure":time_of_departure,
"time_of_arrival":time_of_arrival, "duration":duration, "price":price,
"route_existence":route_existence})
    console.log(response)
    fetchRoutes()
  }
  async function updateRoutes() {
    const id = inputRouteValues[0];
    const route_name = inputRouteValues[1];
    const point_of_departure = inputRouteValues[2];
    const point_of_arrival = inputRouteValues[3];
    const time_of_departure = inputRouteValues[4];
    const time_of_arrival = inputRouteValues[5];
    const duration = inputRouteValues[6];
    const price = inputRouteValues[7];
    const route_existence = inputRouteValues[8];
    const response = await axios.put('http://localhost:3000/api/routes',
{"id":id, "route_name":route_name, "point_of_departure":point_of_departure,
"point_of_arrival":point_of_arrival, "time_of_departure":time_of_departure,
"time_of_arrival":time_of_arrival, "duration":duration, "price":price,
"route_existence":route_existence})
    console.log(response)
    fetchRoutes()
  }
  async function deleteRoutes() {
    const id = inputRouteValues[0];
    const response = await
axios.delete(`http://localhost:3000/api/routes/${id}`)
    console.log(response)
    fetchRoutes()
  }

  async function createClient() {
    const id_place = inputClientValues[1];
    const first_name = inputClientValues[2];
    const second_name = inputClientValues[3];
    const phone_number = inputClientValues[4];
    const email = inputClientValues[5];
    const response = await axios.post('http://localhost:3000/api/client',
{"id_place":id_place, "first_name":first_name, "second_name":second_name,
"phone_number":phone_number, "email":email})
    console.log(response)
  }

```

```

        fetchClients()
    }
    async function updateClient() {
        const id = inputClientValues[0];
        const id_place = inputClientValues[1];
        const first_name = inputClientValues[2];
        const second_name = inputClientValues[3];
        const phone_number = inputClientValues[4];
        const email = inputClientValues[5];
        const response = await axios.put('http://localhost:3000/api/client',
{"id":id, "id_place":id_place, "first_name":first_name,
"second_name":second_name, "phone_number":phone_number, "email":email})
        console.log(response)
        fetchClients()
    }
    async function deleteClient() {
        const id = inputClientValues[0];
        const response = await
axios.delete(`http://localhost:3000/api/client/${id}`)
        console.log(response)
        fetchClients()
    }

    async function createPlace() {
        const id_route = inputPlaceValues[1];
        const place_number = inputPlaceValues[2];
        const place_existence = inputPlaceValues[3];
        const place_date = inputPlaceValues[4];
        const response = await axios.post('http://localhost:3000/api/place',
{"id_route":id_route, "place_number":place_number,
"place_existence":place_existence, "place_date":place_date})
        console.log(response)
        fetchPlaces()
    }
    async function updatePlace() {
        const id = inputPlaceValues[0];
        const id_route = inputPlaceValues[1];
        const place_number = inputPlaceValues[2];
        const place_existence = inputPlaceValues[3];
        const place_date = inputPlaceValues[4];
        const response = await axios.put('http://localhost:3000/api/place',
{"id":id, "id_route":id_route, "place_number":place_number,
"place_existence":place_existence, "place_date":place_date})
        console.log(response)
        fetchPlaces()
    }
    async function deletePlace() {
        const id = inputPlaceValues[0];
        const response = await
axios.delete(`http://localhost:3000/api/place/${id}`)
        console.log(response)
    }

```

```

    fetchPlaces()
  }

  return(
    <>
      <header className="header">
        <Link to="/"></Link>
        <nav className="nav-block">
          <ul className="nav-menu">
            <li className="nav-item button"
onClick={Cities}>Міста</li>
            <li className="nav-item button"
onClick={Stations}>Станції</li>
            <li className="nav-item button"
onClick={BusRoutes}>Маршрути</li>
            <li className="nav-item button"
onClick={Clients}>Клієнти</li>
            <li className="nav-item button"
onClick={Places}>Білет</li>
            <Link to="/"><li className="nav-item button"
>Вихід</li></Link>
          </ul>
        </nav>
      </header>

      <div className="bg-main"/>
      <content className="content">
        {!cities &&<section className="first-block">
          <div className="table">
            <table>
              <tr>
                <th>id</th>
                <th>Місто</th>
              </tr>
              {state.map(({id, city_name}) => (
                <tr key={id} onClick={() =>
handleCityRowClick(id, city_name)}>
                  <td>{id}</td>
                  <td>{city_name}</td>
                </tr>
              ))}
            </table>
          </div>
        </section>
        {!stations &&<section className="first-block">
          <div className="table">
            <table>
              <tr>

```

```

        <th>id</th>
        <th>Станція</th>
        <th>id Міста</th>
        <th>Адреса станції</th>
      </tr>
      {state.map(({id, bus_station_name, city_id,
bus_station_address}) => (
        <tr key={id} onClick={() =>
handleStationRowClick(id, bus_station_name, city_id, bus_station_address)}>
          <td>{id}</td>
          <td>{bus_station_name}</td>
          <td>{city_id}</td>
          <td>{bus_station_address}</td>
        </tr>
      ))}
    </table>
  </div>
</section>}
{!busRoutes &&<section className="first-block">
  <div className="table">
    <table>
      <tr>
        <th>id</th>
        <th>Маршрут</th>
        <th>Пункт відправлення</th>
        <th>Пункт прибуття</th>
        <th>Час відправлення</th>
        <th>Час прибуття</th>
        <th>Тривалість поїздки</th>
        <th>Ціна</th>
        <th>Наявність</th>
      </tr>
      {state.map(({id, route_name, point_of_departure,
point_of_arrival, time_of_departure, time_of_arrival, duration, price,
route_existence}) => (
        <tr key={id} onClick={() =>
handleRouteRowClick(id, route_name, point_of_departure, point_of_arrival,
time_of_departure, time_of_arrival, duration, price, route_existence)}>
          <td>{id}</td>
          <td>{route_name}</td>
          <td>{point_of_departure}</td>
          <td>{point_of_arrival}</td>
          <td>{time_of_departure}</td>
          <td>{time_of_arrival}</td>
          <td>{duration}</td>
          <td>{price}</td>
          <td>{route_existence}</td>
        </tr>
      ))}
    </table>
  </div>

```

```

</section>}
{!clients &&<section className="first-block">
  <div className="table">
    <table>
      <tr>
        <th>id</th>
        <th>№ Місця</th>
        <th>Ім'я</th>
        <th>Прізвище</th>
        <th>Номер телефону</th>
        <th>email</th>
      </tr>
      {state.map(({id, id_place, first_name, second_name,
phone_number, email}) => (
        <tr key={id} onClick={() =>
handleClientRowClick(id, id_place, first_name, second_name, phone_number,
email)}}>
          <td>{id}</td>
          <td>{id_place}</td>
          <td>{first_name}</td>
          <td>{second_name}</td>
          <td>{phone_number}</td>
          <td>{email}</td>
        </tr>
      ))}
    </table>
  </div>
</section>}
{!places &&<section className="first-block">
  <div className="table">
    <table>
      <tr>
        <th>id</th>
        <th>№ маршруту</th>
        <th>№ місця</th>
        <th>Наявність</th>
        <th>Дата</th>
      </tr>
      {state.map(({id, id_route, place_number,
place_existence, place_date}) => (
        <tr key={id} onClick={() =>
handlePlaceRowClick(id, id_route, place_number, place_existence, place_date)}}>
          <td>{id}</td>
          <td>{id_route}</td>
          <td>{place_number}</td>
          <td>{place_existence}</td>
          <td>{place_date}</td>
        </tr>
      ))}
    </table>
  </div>

```

```

</section>}

{!cityes && <section className="second-block">
<div className="workspace-block">
  <ul className="workspace-menu">
    {[...Array(2)].map((_, index) => (
      <li ><input
        className="workspace-item-input"
        key={index}
        type="text"
        value={inputCityValues[index]}
        placeholder={placeholdersCity[index]}
        onChange={(event) => handleCityChange(index, event)}
      /></li>
    )]}
  </ul>
  <div className="button-container">
    <div>
      <button className="workspace-item-button"
onClick={handleCityClear}>Очистити</button>
      <button className="workspace-item-button"
onClick={updateCity}>Оновити запис</button>
      <button className="workspace-item-button"
onClick={applyCityFilter}>Знайти</button>
    </div>
    <div>
      <button className="workspace-item-button"
onClick={createCity}>Додати запис</button>
      <button className="workspace-item-button"
onClick={deleteCity}>Видалити запис</button>
      <button className="workspace-item-button"
onClick={resetFilter}>Відміна фільтру</button>
    </div>
  </div>
</div>
</section>}
{!stations && <section className="second-block">
  <div className="workspace-block">
    <ul className="workspace-menu">
      {[...Array(4)].map((_, index) => (
        <li ><input
          className="workspace-item-input"
          key={index}
          type="text"
          value={inputStationValues[index]}
          placeholder={placeholdersStation[index]}
          onChange={(event) => handleStationChange(index,
event)}}
        /></li>
      )]}
    </ul>
  </div>
</section>}

```

```

        </ul>
        <div className="button-container">
            <div>
                <button className="workspace-item-button"
onClick={handleStationClear}>Очистити</button>
                <button className="workspace-item-button"
onClick={updateStation}>Оновити запис</button>
                <button className="workspace-item-button"
onClick={applyStationFilter}>Знайти</button>
            </div>
            <div>
                <button className="workspace-item-button"
onClick={createStation}>Додати запис</button>
                <button className="workspace-item-button"
onClick={deleteStation}>Видалити запис</button>
                <button className="workspace-item-button"
onClick={resetFilter}>Відміна фільтру</button>
            </div>
        </div>
    </div>
</section>
}
{!busRoutes && <section className="second-block">
    <div className="workspace-block">
        <ul className="workspace-menu-routes">
            {[...Array(9)].map((_, index) => (
                <li><input
                    className="workspace-item-input-routes"
                    key={index}
                    type="text"
                    value={inputRouteValues[index]}
                    placeholder={placeholdersRoute[index]}
                    onChange={(event) => handleRouteChange(index, event)}
                /></li>
            ))}
        </ul>
        <div className="button-container">
            <div>
                <button className="workspace-item-button"
onClick={handleRouteClear}>Очистити</button>
                <button className="workspace-item-button"
onClick={updateRoutes}>Оновити запис</button>
                <button className="workspace-item-button"
onClick={applyRouteFilter}>Знайти</button>
            </div>
            <div>
                <button className="workspace-item-button"
onClick={createRoutes}>Додати запис</button>
                <button className="workspace-item-button"
onClick={deleteRoutes}>Видалити запис</button>
                <button className="workspace-item-button"
onClick={resetFilter}>Відміна фільтру</button>
            </div>
        </div>
    </div>
}

```

```

        </div>
      </div>
    </div>
  </section>}
  {!clients && <section className="second-block">
    <div className="workspace-block">
      <ul className="workspace-menu">
        {[...Array(6)].map((_, index) => (
          <li ><input
            className="workspace-item-input"
            key={index}
            type="text"
            value={inputClientValues[index]}
            placeholder={placeholdersClient[index]}
            onChange={(event) => handleClientChange(index,
event)}}
          /></li>
        )]}
      </ul>
      <div className="button-container">
        <div>
          <button className="workspace-item-button"
onClick={handleClientClear}>Очистити</button>
          <button className="workspace-item-button"
onClick={updateClient}>Оновити запис</button>
          <button className="workspace-item-button"
onClick={applyClientFilter}>Знайти</button>
        </div>
        <div>
          <button className="workspace-item-button"
onClick={createClient}>Додати запис</button>
          <button className="workspace-item-button"
onClick={deleteClient}>Видалити запис</button>
          <button className="workspace-item-button"
onClick={resetFilter}>Відміна фільтру</button>
        </div>
      </div>
    </div>
  </section>}
  {!places && <section className="second-block">
    <div className="workspace-block">
      <ul className="workspace-menu">
        {[...Array(5)].map((_, index) => (
          <li ><input
            className="workspace-item-input"
            key={index}
            type="text"
            value={inputPlaceValues[index]}
            placeholder={placeholdersPlace[index]}
            onChange={(event) => handlePlaceChange(index, event)}}
          /></li>

```

```

    )))
  </ul>
  <div className="button-container">
    <div>
      <button className="workspace-item-button"
onClick={handlePlaceClear}>Очистити</button>
      <button className="workspace-item-button"
onClick={updatePlace}>Оновити запис</button>
      <button className="workspace-item-button"
onClick={applyPlaceFilter}>Знайти</button>
    </div>
    <div>
      <button className="workspace-item-button"
onClick={createPlace}>Додати запис</button>
      <button className="workspace-item-button"
onClick={deletePlace}>Видалити запис</button>
      <button className="workspace-item-button"
onClick={resetFilter}>Відміна фільтру</button>
    </div>
  </div>
</section>
</content>
</>
);
};

export default Admin;

```

Booking.jsx

```

import React, { useState, useEffect, useContext } from "react";
import { UserContext } from './UserContext';
import { useParams, useLocation, Link } from "react-router-dom";
import axios from "axios";
import './styles/reset.css';
import './styles/Style.css';

const Booking = () => {
  const { userData, setUserData } = useContext(UserContext);
  const { id } = useParams();
  const location = useLocation();
  const { route_name, point_of_departure, point_of_arrival, time_of_departure,
time_of_arrival, duration, price, place_date } = location.state || {};

  const [tickets, setTickets] = useState([]);
  const [options, setOptions] = useState([]);
  const [selectedPlace, setSelectedPlace] = useState(null);
  const [inputFName, setInputFName] = useState('');
  const [inputSName, setInputSName] = useState('');

```

```

const [inputPhone, setInputPhone] = useState('');
const [inputEmail, setInputEmail] = useState('');

useEffect(() => {
  console.log(id, route_name, point_of_departure, point_of_arrival,
time_of_departure, time_of_arrival, duration, price, place_date);
  getTickets(id, place_date);
}, [id, route_name, point_of_departure, point_of_arrival, time_of_departure,
time_of_arrival, duration, price, place_date]);

useEffect(() => {
  if (userData !== null){
    setInputFName(userData.first_name);
    setInputSName(userData.second_name);
    setInputEmail(userData.email);
    setInputPhone(userData.phone_number);
  }
})

async function getTickets(id_route, place_date) {
  try {
    const response = await axios.post('http://localhost:3000/api/place-
booking', { id_route, place_date });
    setTickets(response.data);
    const filteredOptions = response.data
      .filter(ticket => ticket.place_existence === "true")
      .map(ticket => ({ value: ticket.place_number, label: `Micque
№${ticket.place_number}` }));
    setOptions(filteredOptions);
    console.log(response.data);
  } catch (error) {
    console.error("Error fetching tickets", error);
  }
}

const handleLogout = () => {
  setUserData(null);
};

function selectPlace(event) {
  const selectedValue = event.target.value;
  setSelectedPlace(selectedValue);
}

const handleFNameChainge = (event) => {
  setInputFName(event.target.value);
}
const handleSNameChainge = (event) => {
  setInputSName(event.target.value);
}
const handlePhoneChainge = (event) => {

```

```

        setInputPhone(event.target.value);
    }
    const handleEmailChange = (event) => {
        setInputEmail(event.target.value);
    }

    async function addClient() {
        const id_place = selectedPlace;
        const first_name = inputFName;
        const second_name = inputSName;
        const phone_number = inputPhone;
        const email = inputEmail;
        const response = await axios.post('http://localhost:3000/api/client',
{"id_place":id_place, "first_name":first_name, "second_name":second_name,
"phone_number":phone_number, "email":email})

    }
    async function changePlaceExistence(place_date) {
        try {
            const place_number = selectedPlace;
            const id_route = id;
            const response = await axios.put('http://localhost:3000/api/place-
buy', { id_route, place_number, place_date });
            console.log(response.data);
        } catch (error) {
            console.error('Error changing place existence:', error);
        }
    }

    const buyTicket = async () => {
        await addClient();
        await changePlaceExistence(place_date);
        console.log("Дата:", place_date)
    }

    return (
        <>
        <header className="header">
            <Link to="/"></Link>

            <nav className="nav-block">
                <ul className="nav-menu">
                    {userData ? (
                        <>
                            <li className="nav-
item"><h1>{userData.first_name}</h1></li>
                            <li>
                                <Link className="nav-item button search-
header-button" to="/kab">Особистий кабінет</Link>
                                </li>

```

```

        <li className="nav-item button search-header-
button" onClick={handleLogout}>Вихід</li>
    </>
    ) : (
    <>
        <Link to="/login"><li className="nav-item button
search-header-button">Авторизуватися</li></Link>
        <Link to="/signUp"><li className="nav-item button
search-header-button">Зареєструватися</li></Link>
    </>
    )}
</ul>
</nav>
</header>
<div className="bg-main"/>
<div className="container-main">
    <section className="booking-container">
        <div className="booking-block">
            <div className="route-name">
                <h1>{route_name}</h1>
            </div>
            <div className="booking-col-container">
                <div className="booking-col">
                    <h2>Пункт відправлення: {point_of_departure}</h2>
                    <h2>Час відправлення: {time_of_departure}</h2>
                </div>
                <div className="booking-col">
                    <h2>Пункт прибуття: {point_of_arrival}</h2>
                    <h2>Час прибуття: {time_of_arrival}</h2>
                </div>
            </div>
            <div className="booking-col2">
                <h2>Час поїздки: {duration}</h2>
                <h2>Ціна: {price}</h2>
                <h2>Дата поїздки: {place_date}</h2>
            </div>
            <div className="bus-el">
                <select className="down-drop"
onChange={selectPlace}>
                    {options.map((option, index) => (
                        <option key={index} value={option.value}>
                            {option.label}
                        </option>
                    ))}
                </select>
            </div>
        </div>
    </section>
    <section className="second-block">
        <div className="workspace-block booking">
            <ul className="workspace-menu">

```

```

        <li><input type="text" className="workspace-item-
input" placeholder="Ім'я" value={inputFName}
onChange={(event)=>{handleFNameChainge(event)}} /></li>
        <li><input type="text" className="workspace-item-
input" placeholder="Прізвище" value={inputSName}
onChange={(event)=>{handleSNameChainge(event)}}/></li>
        <li><input type="text" className="workspace-item-
input" placeholder="Номер телефону" value={inputPhone}
onChange={(event)=>{handlePhoneChainge(event)}}/></li>
        <li><input type="text" className="workspace-item-
input" placeholder="email" value={inputEmail}
onChange={(event)=>{handleEmailChainge(event)}}/></li>
        <li><button className="button booking-button button-
t-s" onClick={buyTicket}>Перейти до оплати</button></li>
    </ul>
</div>
</section>
</div>
</>
);
};

export default Booking;

```

Kab.jsx

```

import React, { useContext, useState } from "react";
import { Link, useNavigate } from "react-router-dom";
import { UserContext } from './UserContext';
import axios from "axios";
import './styles/reset.css';
import './styles/Style.css';
import './styles/SignUp.css';

const Kab = () => {
    const { userData, setUserData } = useContext(UserContext);
    const [isEditing, setIsEditing] = useState(false);
    const [firstName, setFirstName] = useState(userData.first_name);
    const [secondName, setSecondName] = useState(userData.second_name);
    const [email, setEmail] = useState(userData.email);
    const [phoneNumber, setPhoneNumber] = useState(userData.phone_number);
    const [profilePassword, setProfilePassword] =
    useState(userData.profile_password);

    const navigate = useNavigate();

    const handleLogoClick = () => {
        navigate('/');
    };

```

```

const handleLogout = () => {
  setUserData(null);
  navigate('/');
};

const handleEditClick = () => {
  setIsEditing(true);
};

const handleSaveClick = () => {
  setUserData({
    ...userData,
    first_name: firstName,
    second_name: secondName,
    email: email,
    phone_number: phoneNumber,
    profile_password: profilePassword
  });
  kabUpdate(firstName, secondName, email, profilePassword, phoneNumber,
userData.id);
  setIsEditing(false);
};

async function kabUpdate(fName, sName, email, password, number, id) {
  const response = await axios.put('http://localhost:3000/api/profile',
{"first_name": fName, "second_name": sName, "profile_password": password,
"phone_number": number, "email": email, "id": id})
  console.log(response)
}

return (
  <>
    <header className="header">
      <Link to="/" onClick={handleLogoClick}>
        
      </Link>
      <nav className="nav-block">
        <ul className="nav-menu">
          <li className="nav-
item"><h1>{userData.first_name}</h1></li>
          <li className="nav-item button search-header-button"
onClick={handleLogout}>
            Вихід
          </li>
        </ul>
      </nav>
    </header>
    <div className="bg-main"/>
    <content className="content">
      <div className="profile-details">

```

```

<h1 className="hh">Особистий кабінет</h1>
{isEditing ? (
  <>
    <label>
      <strong>Ім'я: </strong>
      <input
        type="text"
        value={firstName}
        placeholder="Ім'я"
        className="search-input"
        onChange={(e) =>
setFirstName(e.target.value)}
      />
    </label>
    <label>
      <strong>Прізвище: </strong>
      <input
        type="text"
        value={secondName}
        placeholder="Прізвище"
        className="search-input"
        onChange={(e) =>
setSecondName(e.target.value)}
      />
    </label>
    <label>
      <strong>Електронна пошта: </strong>
      <input
        type="email"
        value={email}
        placeholder="Електронна пошта"
        className="search-input"
        onChange={(e) => setEmail(e.target.value)}
      />
    </label>
    <label>
      <strong>Телефон: </strong>
      <input
        type="text"
        value={phoneNumber}
        placeholder="Телефон"
        className="search-input"
        onChange={(e) =>
setPhoneNumber(e.target.value)}
      />
    </label>
    <label>
      <strong>Пароль: </strong>
      <input
        type="text"
        value={profilePassword}

```

```

placeholder="Пароль"
className="search-input"
onChange={ (e) =>
setProfilePassword(e.target.value)}
    />
  </label>
  <button onClick={handleSaveClick} className="nav-item
button search-header-button buttonK">Зберегти</button>
    </>
  ) : (
    <>
      <h2><strong>Ім'я: </strong>
{userData.first_name}</h2>
      <h2><strong>Прізвище: </strong>
{userData.second_name}</h2>
      <h2><strong>Електронна пошта: </strong>
{userData.email}</h2>
      <h2><strong>Телефон: </strong>
{userData.phone_number}</h2>
      <h2><strong>Пароль: </strong>
{userData.profile_password}</h2>
      <button onClick={handleEditClick} className="nav-item
button search-header-button buttonK">Змінити дані</button>
    </>
  )}
</div>
</content>
</>
);
};

export default Kab;

```

Login.jsx

```

import React, { useState, useContext } from "react";
import axios from "axios";
import { useNavigate, Link } from "react-router-dom";
import { UserContext } from "../UserContext";
import './styles/reset.css';
import './styles/Style.css';
import './styles/SignUp.css';

const Login = () => {
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');
  const { setUserData } = useContext(UserContext);
  const navigate = useNavigate();

```

```

const handleEmailChange = (e) => {
  setEmail(e.target.value);
};

const handlePasswordChange = (e) => {
  setPassword(e.target.value);
};

async function login() {
  try {
    if (email === 'admin' && password === 'admin') {
      navigate('/admin');
      return;
    }

    const response = await axios.post('http://localhost:3000/api/profile-
login', {"email": email, "profile_password": password});
    const { profile_password, email: serverEmail } = response.data;
    if (email === serverEmail && password === profile_password) {
      setUserData(response.data); // Запись данных о пользователе в
контекст
      navigate('/');
    } else {
      alert('Невірний логін або пароль');
    }
  } catch (error) {
    console.error('Login error:', error);
    alert('Невірний логін або пароль');
  }
}

const handleSubmit = () => {
  login();
  console.log('Email:', email);
  console.log('Password:', password);
};

return (
  <>
    <header className="header">
      <Link to="/"></Link>
      <nav className="nav-block">
        <ul className="nav-menu">
          <Link to="/signUp"><li className="nav-item button search-
header-button">Зареєструватися</li></Link>
        </ul>
      </nav>
    </header>
    <div className="bg-main"/>
    <content className="content">

```

```

        <div className="workspaceL log-w">
          <h1>Увійти</h1>
          <ul className="reg1">
            <li>
              <input
                type="text"
                placeholder="email"
                className="search-input"
                value={email}
                onChange={handleEmailChange}
              />
            </li>
            <li>
              <input
                type="password"
                placeholder="Пароль"
                className="search-input"
                value={password}
                onChange={handlePasswordChange}
              />
            </li>
            <li
              className="nav-item button search-header-button"
              onClick={handleSubmit}
            >
              Увійти
            </li>
          </ul>
        </div>
      </content>
    </>
  );
};

export default Login;

```

SignUp.jsx

```

import React, { useState, useContext } from "react";
import axios from "axios";
import { useNavigate, Link } from "react-router-dom";
import { UserContext } from "../UserContext";
import './styles/reset.css';
import './styles/Style.css';
import './styles/SignUp.css';

const SignUp = () => {
  const navigate = useNavigate();
  const [email, setEmail] = useState('');

```

```

const [password1, setPassword1] = useState('');
const [password2, setPassword2] = useState('');
const [fName, setFName] = useState('');
const [sName, setSName] = useState('');
const [number, setNumber] = useState('');
const { setUserData } = useContext(UserContext);

const handleEmailChange = (e) => {
  setEmail(e.target.value);
};

const handlePassword1Change = (e) => {
  setPassword1(e.target.value);
};

const handlePassword2Change = (e) => {
  setPassword2(e.target.value);
};

const handleFNameChange = (e) => {
  setFName(e.target.value);
};

const handleSNameChange = (e) => {
  setSName(e.target.value);
};

const handleNumberChange = (e) => {
  setNumber(e.target.value);
};

async function createProfile(email) {
  const first_name = fName;
  const second_name = sName;
  const profile_password = password1;
  const phone_number = number;
  await axios.post('http://localhost:3000/api/profile', {
    "first_name": first_name,
    "second_name": second_name,
    "profile_password": profile_password,
    "phone_number": phone_number,
    "email": email
  });
}

async function login(email) {
  const response = await axios.post('http://localhost:3000/api/profile-
login', {
    "email": email,
    "profile_password": password1
  });
}

```

```

    const { profile_password, email: serverEmail } = response.data;
    if (email === serverEmail && password1 === profile_password) {
      setUserData(response.data);
      navigate('/');
    }
  }
}

const reg = () => {
  if (password1 !== password2) {
    alert('Паролі не співпадають');
    return;
  }
  createProfile(email)
    .then(() => login(email))
    .catch(error => {
      console.error('Registration error:', error);
      alert('Помилка реєстрації');
    });
}

return (
  <>
    <header className="header">
      <Link to="/"></Link>
      <nav className="nav-block">
        <ul className="nav-menu">
          <Link to="/login"><li className="nav-item button search-
header-button">Авторизуватися</li></Link>
        </ul>
      </nav>
    </header>
    <div className="bg-main" />
    <content className="content">
      <div className="workspaceL">
        <h1>Реєстрація</h1>
        <ul className="reg1">
          <li><input
            type="text"
            placeholder="email"
            className="search-input"
            value={email}
            onChange={handleEmailChange}
          /></li>
          <li><input
            type="password"
            placeholder="Пароль"
            className="search-input"
            value={password1}
            onChange={handlePassword1Change}
          /></li>

```

```

        <li><input
          type="password"
          placeholder="Повторіть пароль"
          className="search-input"
          value={password2}
          onChange={handlePassword2Change}
        /></li>
      </ul>
      <ul className="reg2">
        <li><input
          type="text"
          placeholder="Ім'я"
          className="search-input"
          value={fName}
          onChange={handleFNameChange}
        /></li>
        <li><input
          type="text"
          placeholder="Прізвище"
          className="search-input"
          value={sName}
          onChange={handleSNameChange}
        /></li>
        <li><input
          type="text"
          placeholder="Телефон"
          className="search-input"
          value={number}
          onChange={handleNumberChange}
        /></li>
        <li className="nav-item button search-header-button"
          onClick={reg}>Зареєструватися</li>
      </ul>
    </div>
  </content>
</>
);
};

export default SignUp;

```

UserContext.jsx

```

// UserContext.js
import React, { createContext, useState } from 'react';

const UserContext = createContext();

const UserProvider = ({ children }) => {

```

```
const [userData, setUserData] = useState(null);

return (
  <UserContext.Provider value={{ userData, setUserData }}>
    {children}
  </UserContext.Provider>
);
};

export { UserContext, UserProvider };
```

ДОДАТОК Б
Слайди презентації

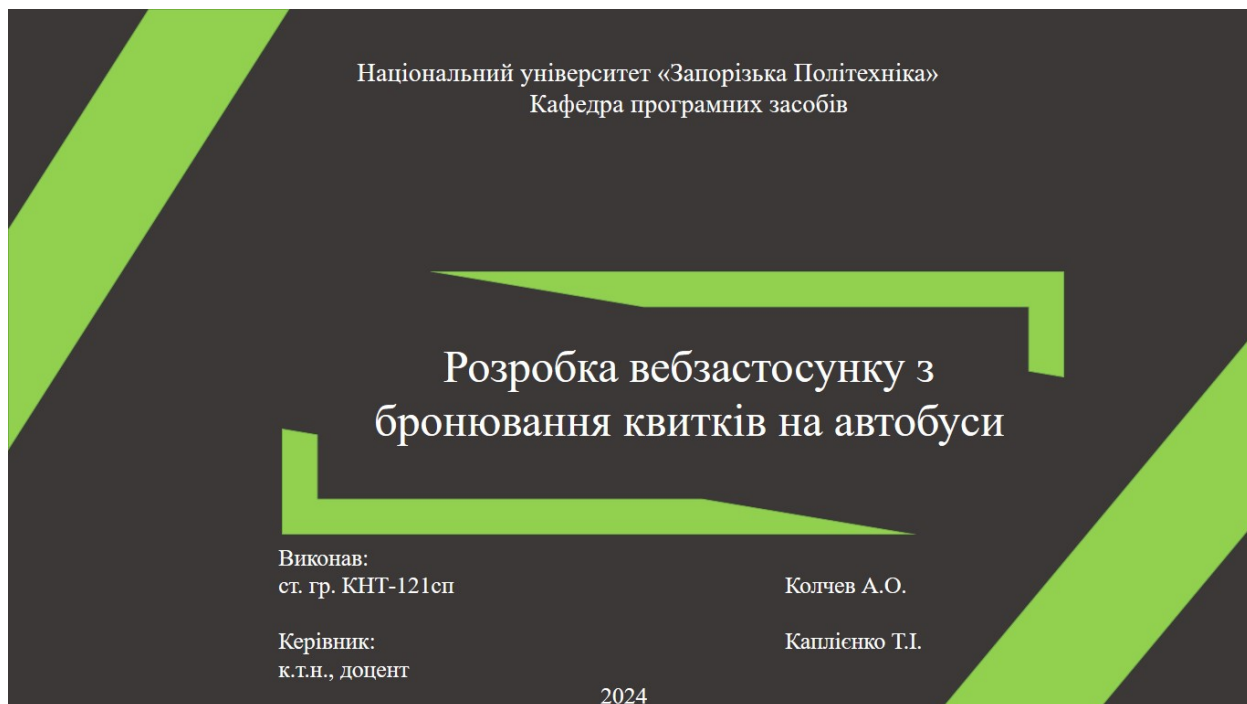


Рисунок Б.1 – Слайд №1

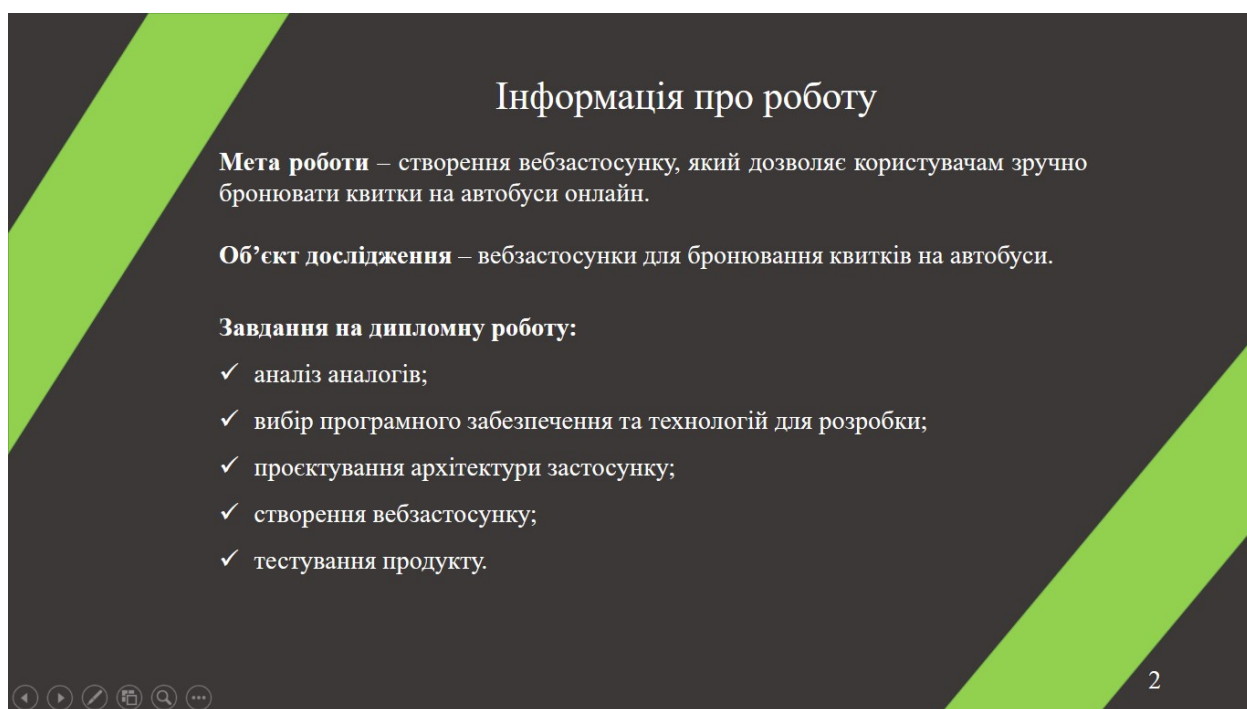


Рисунок Б.2 – Слайд №2

Порівняння переваг аналогів			
Kayak.com	Expedia.com	Skyscanner.com	
широкий вибір фільтрів для пошуку, включаючи відстань до аеропорту, тривалість перельоту, авіакомпанії тощо	великий асортимент пропозицій: авіаквитки, проживання, оренда автомобілів, пакетні тури	простий і зрозумілий пошуковий інтерфейс для швидкого порівняння цін на квитки	
функція "Price Forecast" допомагає визначити найвигіднішу дату для покупки квитків	програма лояльності для постійних клієнтів з накопиченням бонусних балів	гнучкий пошук "Куди завгодно", який показує найдешевші напрямки з вибраного міста	
можливість бронювання не лише авіаквитків, але й проживання та оренди автомобілів	розділ з відгуками туристів про готелі та пункти призначення	можливість відстежувати ціни на квитки і отримувати сповіщення про знижки	

Рисунок Б.3 – Слайд №3

Порівняння технологій для серверної частини сайту				
Характеристика	Node.js	Python (Django/Flask)	Ruby (Ruby on Rails)	PHP (Laravel)
Мова програмування	JavaScript	Python	Ruby	PHP
Однопоточність	Так	Ні	Ні	Ні
Асинхронність	Так (вбудована підтримка)	Так (через бібліотеки)	Так (через бібліотеки)	Ні (частково через бібліотеки)
Продуктивність	Висока (завдяки V8)	Середня	Середня	Середня
Розгортання	Легке (Heroku, Vercel, AWS)	Легке (Heroku, AWS)	Легке (Heroku, AWS)	Легке (Heroku, AWS)

Рисунок Б.4 – Слайд №4

Порівняння технологій для клієнтської частини сайту				
Характеристика	React	Angular	Vue.js	Svelte
Мова програмування	JavaScript (JSX)	TypeScript/JavaScript	JavaScript	JavaScript
Підхід	Бібліотека для UI компонентів	Повноцінний фреймворк	Прогресивний фреймворк	Компільований фреймворк
Вивчення та використання	Відносно просте	Складніше	Відносно просте	Просте
Розмір	Малий	Великий	Малий	Дуже малий
Сумісність з мобільними застосунками	React Native	Ionic + Angular	Weex (для Alibaba)	Sapper
Спільнота	Дуже велика	Велика	Велика	Розвивається

Рисунок Б.5 – Слайд №5

Порівняння систем керування базами даних				
Характеристика	PostgreSQL	MySQL	SQLite	MongoDB
Тип	Реляційна СУБД	Реляційна СУБД	Реляційна СУБД	Документо-орієнтована СУБД
Мова запитів	SQL	SQL	SQL	BSON (JSON-орієнтований)
Підтримка JSON	Так	Так	Ні	Рідний формат
Розширюваність	Висока (через розширення)	Обмежена	Ні	Висока (плагіни, модулі)
Безпека	Висока (розширена підтримка)	Висока	Низька	Висока
Продуктивність	Висока	Висока	Середня	Висока (на великих даних)

Рисунок Б.6 – Слайд №6

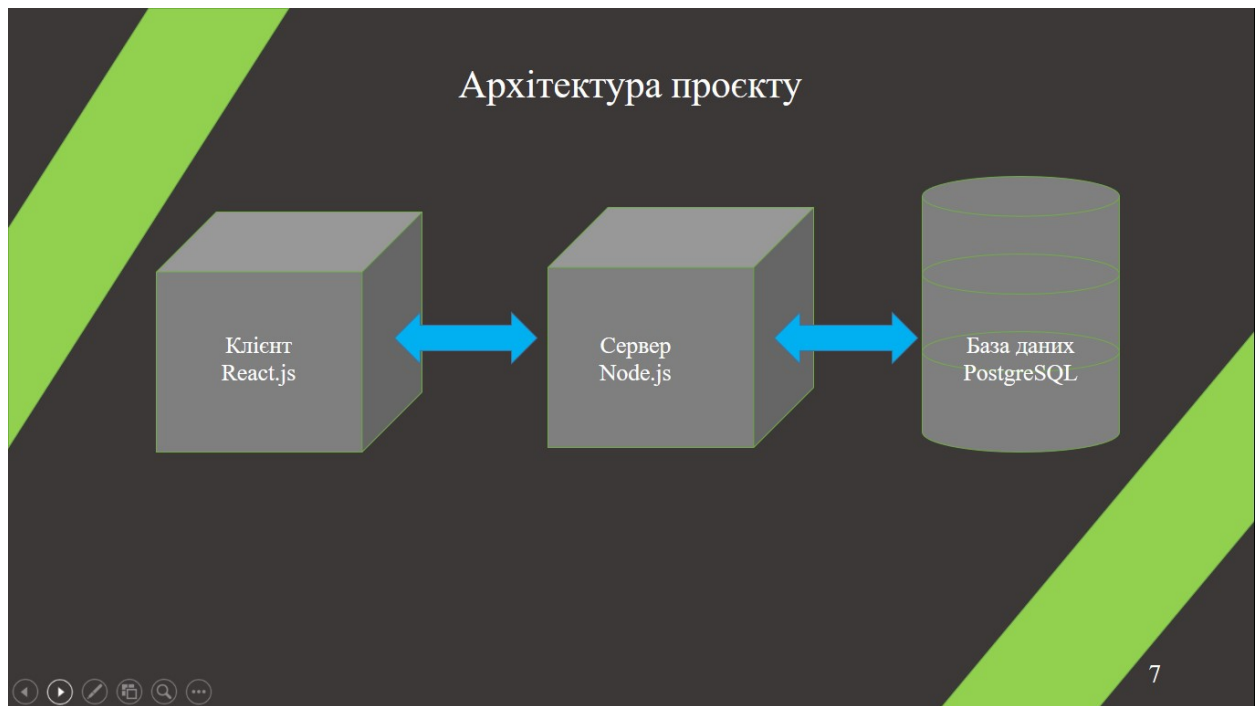


Рисунок Б.7 – Слайд №7



Рисунок Б.8 – Слайд №8



Рисунок Б.9 – Слайд №9



Рисунок Б.10 – Слайд №10



Рисунок Б.11 – Слайд №11

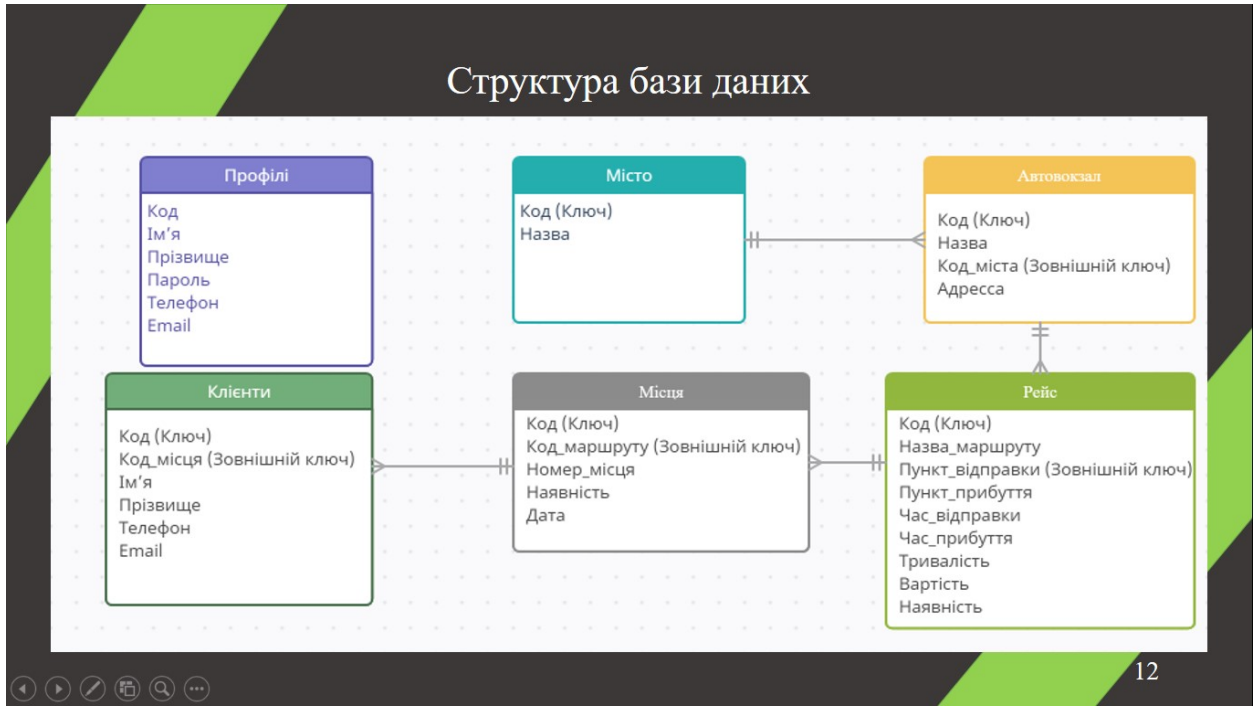


Рисунок Б.12 – Слайд №12



Рисунок Б.13 – Слайд №13

Висновки

Під час виконання дипломної роботи були виконані наступні завдання:

- ✓ проаналізовано декілька аналогів;
- ✓ обрано технології, які найкраще підходять для виконання поставленої задачі;
- ✓ спроектовано архітектуру застосунку;
- ✓ вебзастосунок створено та протестовано.

Розроблений застосунок надає користувачам можливість легко і швидко бронювати квитки через інтернет, використовуючи зручний та інтуїтивно зрозумілий інтерфейс. Застосунок забезпечує необхідний функціонал для реєстрації користувачів, вибору рейсів, бронювання місць, здійснення оплати та отримання електронних квитків.

Результати роботи апробовані на науково-практичній конференції «Тиждень науки-2024», яка дистанційно проводилася у Національному університеті «Запорізька політехніка» у 2024 році.

Рисунок Б.14 – Слайд №14