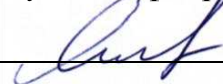


МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний університет кораблебудування імені адмірала Макарова
Навчально-науковий інститут комп'ютерних наук та управління проектами
Кафедра інформаційних управляючих систем та технологій

"Допущений до захисту"

Завідувач кафедри ІУСТ



к.т.н, доц. Михелєв І.Л.

" ____ " _____ 2025 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня вищої освіти "бакалавр"

на тему: Розроблення інформаційної системи з реалізації продажу квитків на
сайті авіакомпанії

Виконав: студент групи 4145



(підпис)

Пономарчук В. А.

Керівник роботи:

К.Т.Н., ДОЦЕНТ

(посада, науковий ступінь вчене звання)



(підпис)

Книрик Н. Р.

м. Миколаїв – 2025 р.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний університет кораблебудування імені адмірала Макарова
Навчально-науковий інститут комп'ютерних наук та управління проектами
Кафедра інформаційних управляючих систем та технологій

Спеціальність 124 "Системний аналіз"

Освітня програма "Системний аналіз"

"ЗАТВЕРДЖУЮ"

Гарант освітньої програми



к.т.н, доц. Гайдаєнко О. В.

"___" _____ 2025 р.

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ на здобуття ступеня вищої освіти "бакалавр"

Студенту Пономарчук Владислав Андрійович

(прізвище, ім'я, по батькові)

1. Тема роботи: Розроблення інформаційної системи з реалізації продажу квитків на сайті авіакомпанії

Керівник роботи Книрик Наталія Ромуальдівна

Затверджені наказом ректора № 295-уч від "27" березня 2025 року.

2. Термін подання роботи: 20 червня 2025 р.

3. Вихідні дані до проекту (роботи) ДСТУ щодо обробки інформації, літературні джерела, технічна документація на існуючі аналоги систем)

4. Перелік питань, що належать до розробки (найменування розділів) Вступ(Актуальність теми, Front-end, Back-end), опис інформаційних систем продажу квитків, загальний опис процесу продажу квитків, опис сайту для продажу квитків авіакомпанії, переваги сайту, унікальність проекту, структура проекту сайту, планування проекту, front-end, back-end, розробка інформаційної системи продажу/купівлі квитків, основні функції сайту та захист даних, реєстрація/вхід в профіль користувача, реалізація форми реєстрації на front-end і back-end, реалізація форми входу на front-end і back-end, перегляд та купівля квитків, аналіз передачі даних квитків з back-end на сайт, логіка виконання транзакцій, реалізація функціоналу багатомовності сайту, реалізація додавання карток для оплати квитків, висновок.

5. Перелік презентаційних матеріалів додатки, схеми, презентація, репозиторій Github <https://ponomarchukvlad.github.io/my-react-app>.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1	Книрік Н. Р.	03.02.2025	22.02.2025
2	Книрік Н. Р.	22.02.2025	21.03.2025
3	Книрік Н. Р.	21.03.2025	25.04.2025
4	Книрік Н. Р.	25.04.2025	07.06.2025

7. Дата видачі завдання 03 лютого 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів роботи	Примітка
1	Аналіз предметної галузі	13.02.2025	
2	Постановка задачі	15.02.2025	
3	Розробка концепції	22.02.2025	
4	Розробка проекту ІС	21.03.2025	
5	Реалізація проекту	25.04.2025	
6	Розробка питань з охорони праці	07.06.2025	
7	Розробка графічних матеріалів	12.06.2025	
8	Подання роботи на перевірку на плагіат	10.06.2025	
9	Подання роботи рецензенту	15.06.2025	
10	Подання роботи на захист	20.06.2025	

Студент


 (підпис)

Пономарчук В. А.
 (ПІБ)

Керівник роботи


 (підпис)

Книрік Н. Р.
 (ПІБ)

АННОТАЦІЯ

У цій дипломній роботі ми розглянемо створення сайту авіакомпанії FashaSkyLine. Як створювався сайт, опис використаних для цього кодів. Всі переваги та недоліки такого сайту і як це полегшує замовлення квитків і так само збільшуючи ваш комфорт у відпустці. Розгляд послуг, що надаються нашою організацією.

Авіакомпанія «FashaSkyLine» – гарантія вашого спокійного та комфортного відпочинку, ми надаємо широкий вибір країн для подорожей із зручним розрахунком ціни, враховуючи ваше місцезнаходження, а також різні способи безконтактної оплати.

Об'єктом дослідження є процес створення сайту.

Предмет – Розробка сайту HTML, CSS, JavaScript, Spring Boot.

Ціль роботи – розробка веб-сайту авіакомпанії FashaSkyLine.

Завдання дослідження:

1. Ознайомитись із сучасними Інтернет-технологіями;
2. Вивчити основні поняття та програмний інструментарій, що застосовується для розробки та створення Web-сайтів;
3. Ознайомитися з основними правилами та рекомендаціями щодо розробки та створення Web-сайтів;
4. створити структуру сайту;
5. Подати покрокову стратегію розробки сайту.

Методи дослідження: вивчення літератури, аналіз, синтез, класифікація та узагальнення.

Ключові слова: React, Spring Boot, HTML, CSS, JavaScript, PostgreSQL, SQL-інєкції, хешування, реляційні бази даних, Java, API, валідація, анотації, URL-адреси, маршрутизація, репозиторії, контролери, Entity, фреймворки, сервіси, структури, методи, front-end, back-end, JPA, HTTP, JSX, Router, Routes, реляційні бази даних, XSS, REST API, Entity.

ABSTRACT

In this thesis, we will consider the creation of the FashaSkyLine airline website. How the website was created, a description of the codes used for this. All the advantages and disadvantages of such a website and how it facilitates ordering tickets and also increases your comfort on vacation. Consideration of the services provided by our organization.

FashaSkyLine airline is a guarantee of your peaceful and comfortable vacation, we provide a wide selection of countries for travel with convenient price calculation, taking into account your location, as well as various methods of contactless payment.

The object of the study is the process of creating a website.

Subject – Website development HTML, CSS, JavaScript, Spring Boot.

The purpose of the work is to develop a website for FashaSkyLine airline.

Research objectives:

1. Get acquainted with modern Internet technologies;
2. Study the basic concepts and software tools used to develop and create websites;
3. Get acquainted with the basic rules and recommendations for the development and creation of Web sites;
4. create a site structure;
5. Present a step-by-step site development strategy.

Research methods: literature review, analysis, synthesis, classification and generalization.

Keywords: React, Spring Boot, HTML, CSS, JavaScript, PostgreSQL, SQL integration, hashing, relational databases, Java, API, validation, annotations, URLs, routing, repositories, controllers, Entity, frameworks, services, structures, methods, front-end, back-end, JPA, HTTP, JSX, Router, Routes, relational databases, XSS, REST API, Entity.

ЗМІСТ

ВСТУП	8-9
РОЗДІЛ 1. ОПИС ІНФОРМАЦІЙНИХ СИСТЕМ ПРОДАЖУ КВИТКІВ	10-16
1.1. Загальний опис процесу продажу квитків.....	10-12
1.2. Опис сайту для продажу квитків авіакомпанії.....	12-14
1.3. Переваги сайту.....	14
1.4. Унікальність проєкту.....	14-16
РОЗДІЛ 2. СТРУКТУРА ПРОЄКТУ САЙТУ	17-30
2.1. Планування проєкту.....	17
2.2. Fron-tend.....	17-22
2.3. Back-end.....	22-30
РОЗДІЛ 3. РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ ПРОДАЖУ/КУПІВЛІ КВИТКІВ, ОСНОВНІ ФУНКЦІЇ САЙТУ ТА ЗАХИСТ ДАНИХ	31-52
3.1. Реєстрація/Вхід в профіль користувача.....	31-33
3.2. Реалізація форми реєстрації на fron-tend і back-end.....	33-35
3.3. Реалізація форми входу на front-end і back-end.....	35-38
3.4. Перегляд та купівля квитків. Front-end.....	38-44
3.5. Аналіз передачі даних квитків з back-end на сайт.....	44-46
3.6. Логіка виконання транзакцій.....	46-48
3.7. Реалізація функціоналу багатомовності сайту.....	48-49
3.8. Реалізація додавання карток для оплати квитків.....	49-52
ВИСНОВОК	53-54
СПИСОК ЛІТЕРАТУРИ	55
ДОДАТКИ	56-90
ДОДАТОК А.....	56-59
ДОДАТОК Б.....	59-62
ДОДАТОК В.....	62-70
ДОДАТОК Г.....	71-74
ДОДАТОК Д.....	74-79
ДОДАТОК Е.....	79-82

ДОДАТОК Є.....	82-90
----------------	-------

ВСТУП

Актуальність теми. У сучасну цифрову епоху веб-сайти авіакомпаній стали ключовим інструментом взаємодії з клієнтами. Зростаючий попит на онлайн-бронювання, вимоги до зручних інтерфейсів, безпеки транзакцій та багатомовної підтримки зумовлюють необхідність розробки сучасних інформаційних систем. Створення ефективного веб-сайту для продажу квитків не лише підвищує конкурентоспроможність авіакомпанії, але й забезпечує користувачам швидкий, безпечний та комфортний досвід планування подорожей, що особливо важливо в глобалізованому світі.

Створення веб-сайту – це складний процес, який враховує всі уподобання користувачів; важливо, щоб він не лише виконував основну функцію придбання квитків, але й приваблював своїм зовнішнім виглядом.

Мета роботи. Розробка інформаційної системи продажу квитків для авіакомпанії FashaSkyLine на основі сучасних веб-технологій (React, Spring Boot, PostgreSQL) зі зручним інтерфейсом, безпекою даних та ефективними процесами транзакцій.

Для початку розглянемо, що взагалі являють собою сайти та поняття її веб-розробки.

Сайт – це одна або декілька веб-сторінок. Вони зберігають логічно пов'язану інформацію, сприймаються користувачами як єдине ціле. Веб-сайт має унікальну адресу, а доступ до нього відбувається за протоколом HTTP або HTTPS [1].

Веб-сайт – це спосіб взаємодії користувачів із інформацією. Робота із сайтом проводиться за допомогою браузера. Браузер забезпечує користувачам взаємодію з веб-сторінками, лише за його допомогою можливий перегляд інформації [1]. Це головний читач, на нього потрібно орієнтуватися під час написання програмного коду.

Веб-розробка – це аспекти створення веб-сайту. Вона поділяється на дві категорії: front-end та back-end.

Front-end пов'язаний із зовнішньою стороною ресурсу. Це те, що бачать на екрані та з чим взаємодіють користувачі при вході на сайт. Front-end розробник займається створенням макету, інтерфейсу, верстки шаблону. Його завдання – створення структури сторінок HTML, оформлення їх стилями за допомогою CSS, а також додавання інтерактивних елементів JavaScript [2].

Такий фахівець також використовує сучасні фреймворки, React, Vue або Angular для створення динамічних інтерфейсів користувача. Крім того, front-end забезпечує інтеграцію із серверною частиною через API та реалізує валідацію форм.

Back-end пов'язаний із програмно-апаратною стороною, не видимою користувачам. Саме back-end відповідає за функціонування веб-сайту. Back-end розробник займається внутрішнім змістом системи, її логікою, яка забезпечує сайту: обробку даних, зберігання інформації в базі даних, авторизацію користувачів, роботу із запитами та відповідями. Back-end фахівець зазвичай використовує мову програмування Python, Node.js, PHP або Java, налаштовує роботу серверів, розробляє API, забезпечує захист даних та оптимізує продуктивність системи. Йому потрібно добре орієнтуватися в архітектурі додатків, принципах безпеки, роботі з реляційними та нереляційними базами даних, а також у деплойменті додатків на сервері чи хмарному середовищі [2].

Технології створення сайтів постійно покращуються, з'являються нові методи веб-розробки. Виділяють три основні методи розробки сайтів:

Самостійна розробка веб-сайту з нуля будь-якою мовою програмування за допомогою популярних веб-інструментів;

- використання frameworks;
- веб-розробка CMS ;

Однозначної відповіді, на чому написати сайт немає. Усі методи створення сайту мають свої нюанси використання.

РОЗДІЛ 1 ОПИС ІНФОРМАЦІЙНИХ СИСТЕМ ПРОДАЖУ КВИТОКІВ

1.1 Загальний опис процесу продажу квитків.

Процес продажу квитків у сучасних умовах є невід’ємною складовою діяльності транспортних компаній, зокрема авіаційних перевізників. Інформаційні системи для продажу квитків автоматизують усі етапи взаємодії з клієнтом – від вибору рейсу до підтвердження покупки та отримання електронного квитка. Це сприяє зменшенню часу обслуговування клієнтів, підвищенню точності обробки даних та зменшенню кількості помилок у системі.

Основні етапи процесу продажу квитків можна поділити на кілька частин:

1. Пошук рейсу – клієнт використовує інформаційну систему для пошуку доступних рейсів за вказаними параметрами (дата, напрямок, кількість пасажирів, клас обслуговування).

Процес пошуку рейсу на сайті авіакомпанії починається з введення користувача ключових параметрів, таких як дата вильоту, напрямок, кількість пасажирів та бажаний клас обслуговування. Для цього використовуються пошукові форми, що інтегруються з базами даних авіакомпаній. Після того, як користувач натискає кнопку пошуку, система звертається до сервера або API, що містить інформацію про доступні рейси. Запит передається на сервер, де здійснюється пошук баз даних рейсів, відповідних введеним параметрам.

Алгоритм пошуку відфільтровує рейси за критеріями користувача та повертає список доступних варіантів.

2. Вибір квитка – після отримання результатів пошуку клієнт вибирає відповідний рейс виходячи із вартості, часу відправлення та прибуття.

Після того, як користувач отримав список доступних рейсів, він може вибрати конкретний рейс виходячи з вартості, зручності часу вильоту та інших факторів. На цьому етапі система також може відображати додаткові опції, наприклад, можливість вибору місця в салоні літака або придбання додаткових послуг, таких як багаж або харчування.

Користувач, натиснувши на вибраний рейс, надсилає запит на сервер. Вибір рейсу фіксується у сесії користувача, що дозволяє зберігати всі дії в межах одного сеансу взаємодії з сайтом. Це важливо для подальшого процесу бронювання та оплати.

3. Бронювання квитка – система дозволяє забронювати квиток на вибраний рейс, резервуючи місце у літаку на певний період.

Після вибору рейсу користувач переходить у процес бронювання. Система створює тимчасовий запис у базі даних, яка резервує місце на вибраному рейсі на час (зазвичай 10-30 хвилин). Цей процес включає такі етапи:

Перевірка наявності місць – система перевіряє в реальному часі, чи вільні місця залишаються на обраному рейсі.

Реєстрація даних пасажирів – користувач заповнює інформацію про пасажирів, що зберігається у базі даних (ім'я, прізвище, дату народження, паспортні дані тощо).

Резервація – після введення даних система створює бронювання на сервері авіакомпанії або через GDS (Global Distribution System) – глобальні системи дистрибуції, які забезпечують резервування квитків у реальному часі.

4. Оплата – після бронювання клієнт здійснює оплату квитка за допомогою зручних платіжних інструментів (банківські картки, електронні гаманці тощо).

Коли бронювання завершено, користувач переходить до оплати. Для цього система інтегрується із платіжними шлюзами. (Банківські системи, такі як Visa, MasterCard, електронні платіжні системи типу PayPal або інші локальні платіжні рішення). Процес оплати відбувається через такі кроки:

Шифрування даних – система забезпечує безпечну передачу платіжної інформації за допомогою SSL сертифікатів або інших протоколів захисту.

Перевірка платіжної інформації – після введення реквізитів картки або інших платіжних даних, ці дані передаються на сервер платіжної системи для авторизації платежу.

Підтвердження транзакції – після успішного оброблення платежу користувач отримує підтвердження оплати, а система оновлює статус бронювання на "оплачений".

Платіжні системи інтегруються із сервером через API платіжних шлюзів, що забезпечує передачу даних про транзакцію в реальному часі та дозволяє відстежувати статус платежу. Після успішного підтвердження транзакції система генерує електронний квиток.

5. Отримання квитка – після підтвердження оплати клієнт отримує електронний квиток, який надсилається електронною

поштою або доступний для завантаження з особистого кабінету на сайті.

Після успішної оплати система автоматично генерує електронний квиток. Цей документ формується на основі даних бронювання та містить усі необхідні відомості для реєстрації на рейс, такі як унікальний код бронювання (PNR), ім'я пасажирів, рейс, дата та час вильоту, місце в салоні тощо.

Електронний квиток надсилається на електронну пошту користувача, а також стає доступним у його особистому кабінеті на сайті. Система може використовувати хмарні технології для зберігання даних квитка, що дозволяє клієнту завантажувати його з будь-якого пристрою у час. Крім того, деякі системи можуть надавати можливість зберігання квитка у вигляді QR-коду, який можна використовувати для швидкої реєстрації в аеропорту.

Таким чином, можна зробити висновок, що з технічного погляду система продажу квитків є комплексним рішенням, яке включає кілька ключових етапів: пошук рейсу, вибір квитка, бронювання, оплату та отримання електронного квитка. Всі ці процеси забезпечуються за допомогою взаємодії інтерфейсу користувача з сервером і базою даних, використовуючи API, платіжні шлюзи та інші інтеграції для забезпечення безпеки і зручності використання.

1.2 Опис сайту для продажу квитків авіакомпанії

Проект розробляється для авіакомпанії, яка пропонує своїм клієнтам зручний та ефективний спосіб купівлі авіаквитків через свій веб-сайт. Цей сайт є не просто інструментом продажу квитків, а ключовим елементом у взаємодії між авіакомпанією та її пасажирів. Основна мета розробки сайту – забезпечити інтуїтивний та безперешкодний досвід користувача, що дозволяє пасажирів легко знайти, забронювати та сплатити авіаквитки, а також отримати всю необхідну інформацію про рейси.

Інтуїтивний інтерфейс для пошуку рейсів

Один із головних аспектів сайту – зручний та простий у використанні інтерфейс для пошуку рейсів. Веб-сайт містить пошукову систему, яка дозволяє пасажирів вводити такі параметри як дати подорожі, міста вильоту та призначення, кількість пасажирів та клас обслуговування (економ, бізнес чи перший клас). Для полегшення пошуку передбачені додаткові

фільтри – наприклад, за ціною квитка, часом вильоту або прильоту, а також тривалістю перельоту.

Ця система працює в режимі реального часу, звертаючись до бази даних авіакомпанії або глобальної системи дистрибуції (GDS) для відображення актуальної інформації про доступні рейси. Інтерфейс пошуку також враховує наявність акцій чи знижок, що дозволяє клієнтам вибирати найвигідніші варіанти.

Особистий кабінет

Особистий кабінет на сайті є важливим інструментом для постійних клієнтів. Веб-сайт дозволяє кожному користувачеві створити обліковий запис, в якому зберігаються особисті дані (ім'я, контактна інформація, дані про паспорти тощо), історія бронювання та оплати, а також переваги подорожей (наприклад, вибір місць у літаку або регулярні маршрути).

Ця функція полегшує процес бронювання, оскільки при подальших покупках клієнтам не потрібно знову вводити свої особисті дані – система автоматично підтягує їх із профілю. Крім того, в особистому кабінеті користувачі можуть відстежувати статус своїх рейсів, отримувати повідомлення про зміни в розкладі та мати доступ до своїх електронних квитків.

Модуль оплати

Одним з найважливіших елементів сайту є модуль для безпечної та зручної оплати квитків. Для цього на сайті реалізовано інтеграцію з різними платіжними системами. Користувачі можуть вибирати з кількох варіантів оплати: банківські картки (Visa, MasterCard, American Express), електронні гаманці (PayPal, Google Pay, Apple Pay) та місцеві платіжні системи залежно від регіону.

Процес оплати відбувається через захищене з'єднання за допомогою шифрування SSL, що гарантує безпеку особистих даних користувачів. Після підтвердження платежу клієнт отримує квиток у цифровому форматі, який можна завантажити чи роздрукувати.

Підтримка мобільних пристроїв

Враховуючи сучасні тенденції використання мобільних пристроїв, веб-сайт авіакомпанії розробляється з адаптивним дизайном, що дозволяє зручно користуватися сайтом як на комп'ютері, так і на смартфонах або планшетах. Це забезпечує гнучкість для клієнтів, дозволяючи їм бронювати квитки у будь-який час та з будь-якого місця.

Адаптивність сайту полягає в тому, що інтерфейс автоматично підлаштовується до розміру екрана пристрою. Користувачі мобільних пристроїв мають доступ до всіх функцій сайту – пошуку рейсів, бронюванню, оплаті та перегляду особистого кабінету з тією ж зручністю, що й на настільних комп'ютерах.

Онлайн підтримка

Для забезпечення високого рівня обслуговування пасажирів на сайті реалізовано функцію онлайн підтримки. Це може бути як онлайн-чат, де клієнти можуть миттєво отримати консультацію в реальному часі від операторів служби підтримки, так і форму зворотнього зв'язку, що дозволяє пасажирам надсилати запити або запитання, що виникають під час використання сайту.

Онлайн підтримка допомагає вирішити технічні проблеми, пов'язані з бронюванням або оплатою квитків, а також відповідає на питання щодо умов подорожі, зміни рейсів, скасування бронювання тощо. Впровадження цієї функції значно підвищує рівень довіри клієнтів до авіакомпанії, оскільки вони завжди мають можливість отримати допомогу у будь-який момент.

1.3 Переваги сайту

Швидкість та зручність – користувачі можуть швидко знаходити та бронювати квитки без виходу з дому. Інформація про доступні рейси та їхню вартість оновлюється в режимі реального часу.

Безпека даних – сайт використовує сучасні технології шифрування для захисту персональних та платіжних даних клієнтів.

Адаптивний дизайн – зручність використання на будь-яких пристроях, від стаціонарних комп'ютерів до смартфонів.

1.4 Унікальність проєкту

Проєкт сайту для авіакомпанії має кілька важливих переваг, що виділяють його серед інших подібних платформ. Основні аспекти унікальності полягають у таких елементах:

Інтуїтивний інтерфейс та простота використання. Однією з головних цілей сайту є забезпечення максимального комфорту для користувачів при пошуку та купівлі квитків. Навігація на сайті

продумана таким чином, щоб користувач міг легко знайти потрібний рейс за мінімальної кількості кліків. Простота та логічність інтерфейсу дозволяють навіть недосвідченим користувачам швидко зорієнтуватися та виконати необхідні операції.

Швидкий процес бронювання та оплати. Завдяки інтеграції з провідними платіжними системами сайт дозволяє проводити швидко та безпечно оплату квитків. Особлива увага приділена зручності бронювання – користувачі можуть оформити квитки лише за кілька кроків, що значно заощаджує їхній час.

Система знижок та акцій. Сайт підтримує функціонал для управління знижками, акціями та спецпропозиціями. Це дозволяє авіакомпанії гнучко керувати ціноутворенням та стимулювати продаж квитків у певні періоди. Користувачі можуть легко знаходити акційні квитки або скористатися промокодами для отримання знижок.

Підтримка мультимовного інтерфейсу. Однією з унікальних характеристик сайту є вибір мови для інтерфейсу, який розширює коло потенційних користувачів. Наявність кількох мовних версій (наприклад, російська, англійська, польська) робить сайт доступним для клієнтів із різних країн.

Адаптивний дизайн. Сайт створений з урахуванням сучасних вимог до адаптивності, що забезпечує зручний перегляд та використання різних пристроїв – від стаціонарних комп'ютерів до смартфонів та планшетів. Це дозволяє користувачам бронювати квитки незалежно від того, де вони знаходяться та який пристрій використовують.

Безпека та захист даних. Проєкт приділяє особливу увагу безпеці персональних та платіжних даних користувачів. Впроваджені сучасні протоколи шифрування та захисту інформації гарантують, що дані клієнтів не потраплять до рук зловмисників. Це підвищує довіру користувачів до сайту та стимулює їх повторне використання для придбання квитків.

У першому розділі було розглянуто процес продажу квитків авіакомпанії “FashaSkyLine” та основні етапи його функціонування, включаючи пошук рейсів, вибір квитків, бронювання, оплату та отримання електронного квитка. Було встановлено, що сучасні інформаційні системи значно спрощують взаємодію з клієнтами, забезпечуючи швидкість, зручність та безпеку транзакцій. Особливу увагу приділено інтуїтивному

інтерфейсу сайту, який дозволяє користувачам легко знаходити та бронювати квитки, а також адаптивному дизайну, що забезпечує комфортне використання на будь-яких пристроях. Крім того, було підкреслено важливість інтеграції з платіжними системами та захисту даних, що робить сайт надійним інструментом для покупців.

РОЗДІЛ 2. СТРУКТУРА ПРОЄКТУ САЙТУ

2.1 Планування проєкту.

Створення сайту починається із планування. Спочатку визначають ціль сайту, що він має робити і яка його цільова аудиторія. Далі визначається технічне завдання: як має виглядати структура сайту, його функціонал та дизайн. Після узгодження всіх деталей розробку можна розбити на дві частини: front-end і back-end.

2.2 Front-end

Front-end – клієнтська частина це те, що бачить і з чим взаємодіє користувач у браузері. У front-end розробку входять HTML, CSS – структури та стилі сторінки, JavaScript – логіка взаємодії: анімації, кнопки, форми, API. Сучасні та популярні фреймворки, які використовують для побудови динамічних інтерфейсів: React, Vue, Angular. Основною метою клієнтської частини є зробити сайт красивим, зручним та інтерактивним.

У нашому випадку для створення сайту ми використовуємо React. Чому саме цей фреймворк у React код пишеться в JavaScript – файлах з HTML-подібним синтаксисом завдяки JSX. JSX – це розширення синтаксису JavaScript, що дозволяє писати HTML код прямо в JavaScript . Приклад такого коду:

```
function Welcome() {  
  return (  
    <h1>Головна сторінка!</h1>  
    <p>Сторінка React </p>  
  );  
}
```

React дозволяє будувати інтерфейси як набір компонентів і тому зручності логіка і розмітка перебуває у одному файлі з фреймворком. Переходимо до етапу створення і структурування (схема 1).

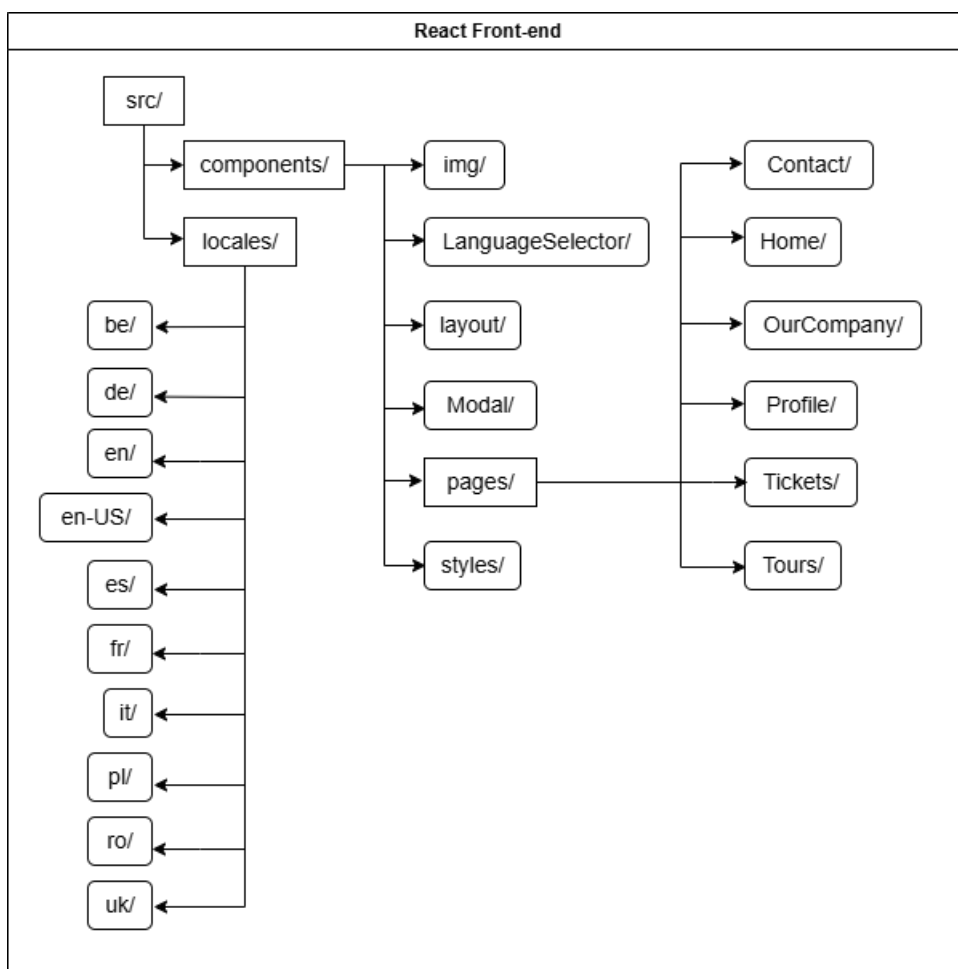


Схема 1

src/components – головні компоненти;

img – зображення іконок/фото, що використовуються на сайті;

languageSelector – містить реалізацію функціоналу зміни мови на сайті, що взаємодіє з папкою locales;

layout – головний компонент сайту, що поєднує в собі всі інші сторінки;

modal – реалізація модальних вікон, що з'являються при реєстрації/логіні на сайт, купівлі квитків тощо, тобто спливаючі вікна;

pages – містить в собі функціональні сторінки сайту;

styles – зберігає стилі для сторінок;

locales – зберігає файли різними мовами для реалізації зміни мови в languageSelector для всіх сторінок сайту;

Для спрощення пошуку та редагування стилів CSS кожна сторінка чи функціонал має свій CSS файл, який знаходиться разом із реалізацією сторінки (рисунок 1):

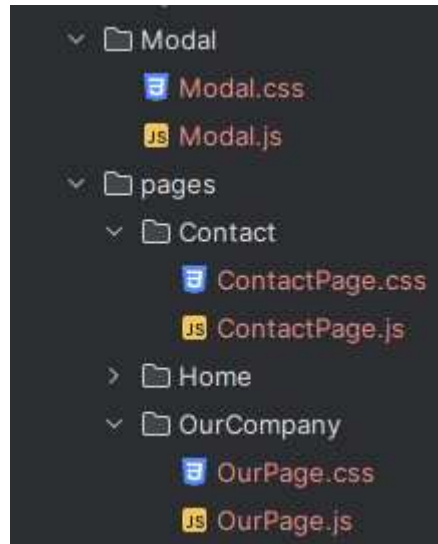


Рисунок 1

На даний момент сайт має шість сторінок, усі вони перераховані у директорії page. Для взаємодії між різними сторінками потрібен маршрутизатор, який зміг би надсилати користувача на певну адресу, щоб реалізувати цю методику в проєкті використовується бібліотека “react-router-dom”, що дозволяє використовувати такі імпорти:

```
import {BrowserRouter as Router, Routes, Route} from
"react-router-dom";
```

React-router-dom – це стандартна бібліотека для маршрутизації (routing) у React - додатках, тобто для організації переходів між сторінками без перезавантаження. Що таке взагалі маршрутизація в React? Зазвичай у звичайному HTML/JS проєкті кожна сторінка – це окремий файл (index.html, register.html і т.д.), але в React – це SPA (Single Page Application): вся програма працює на одній сторінці, і маршрути визначають, який компонент показувати при якому URL-адресу.

Розглянемо тепер детально про кожен елемент Rout:

Router – це базовий компонент, який відповідає за роботу маршрутизації, але сам по собі Router – абстракція. Зазвичай пишеться разом із BrowserRouter.

```
import {BrowserRouter as Router} from “react-router-dom”;
```

Routes – це конкретний всім `<Route />` визначальний набір маршрутів. Працює як “перемикач”, показуючи лише один `<Route />` на основі URL.

```
<Routes>

<Route path="/home" element = {<HomePage />} />

<Route path="/profile" element = {<ProfilePage />} />

</Routes>
```

Route – це окремий маршрут, що вказує який компонент показувати при певному шляху (path).

```
<Route path="/list" element = {<ListPage />} />
```

У прикладі є визначення path і element , path – вказує URL-адресу за якою користувач може перейти на певну сторінку, element – вказує сторінку, яка буде прив'язана до URL-адресу.

За допомогою бібліотеки “react-router-dom” було створено маршрутизацію між основними компонентами сайту FashaSkyLine (рисунок 2).

```
1  import React from "react";
2  import { BrowserRouter as Router, Routes, Route } from "react-router-dom";
3  import HomePage from "../pages/Home/HomePage";
4  import ProfilePage from "../pages/Profile/ProfilePage";
5  import MainLayout from "../layout/MainLayout";
6  import TicketsPage from "../pages/Tickets/TicketsPage";
7  import TourPage from "../pages/Tours/TourPage";
8  import OurPage from "../pages/OurCompany/OurPage";
9  import ContactPage from "../pages/Contact/ContactPage";
10
11  const App = () => { Show usages
12    return (
13      <Router basename={process.env.PUBLIC_URL}>
14        <Routes>
15          <Route path="/" element={<MainLayout />}>
16            <Route index element={<HomePage />} />
17            <Route path="profile" element={<ProfilePage />} />
18            <Route path="tickets" element={<TicketsPage />} />
19            <Route path="tour" element={<TourPage />} />
20            <Route path="our" element={<OurPage />} />
21            <Route path="contact" element={<ContactPage />} />
22          </Route>
23        </Routes>
24      </Router>
25    );
26  };
27
28  export default App; Show usages
29
```

Рисунок 2

На рисунку 3 зображено головний елемент Rout, що вказує на те коли користувач заходитиме на сайт то його направлятиме до головної сторінки за URL - адресою “/” а саме:

```
<Route path="/" element = {<MainLayout />} />
<Route index element = {<HomePage />} />
```

Чому ми бачимо сторінку HomePage за адресою “/” а не MainLayout? MainLayout – це статична сайту, яка поєднує в собі всі інші елементи, але сама не змінюється, змінюються тільки вкладені сторінки вказані в MainLayout. HomePage не має вказаного path а натомість використовує ‘index’ визначальний для HomePage URL-адреса “/” методу <Route path = “/” element = {<MainLayout />}/>.

Всі елементи, вкладені в MainLayout, можна визначити як динамічні сторінки, що змінюються в залежності від вибору користувача:

```
<Route index element = {<HomePage />} />
<Route path = "profile" element = {<ProfilePage />} />
<Route path = "tickets" element = {<TicketsPage />} />
<Route path = "tour" element = {<TourPage />} />
<Route path = "our" element = {<OurPage />} />
<Route path = "contact" element = {<ContactPage />} />
```

MainLayout реалізує базову структуру сайту, що повторюється, на всіх сторінках – зокрема навігаційну панель Navbar та область виведення основного контенту. Для вставлення вмісту відповідної сторінки використовується компонент <Outlet /> з бібліотеки react-router-dom.

<Outlet /> динамічно підставляє дочірній маршрут залежно від поточної адреси у браузері. Таким чином реалізується вкладена маршрутизація коли при переході на головну сторінку “/” в <Outlet /> завантажується HomePage, а при переході на “/profile” - ProfilePage. Такий підхід дозволяє повторно використовувати спільні елементи інтерфейсу без дублювання коду на кожній окремій сторінці, що відповідає принципам компонентної архітектури та полегшує орієнтування у проєкті. Приклад MainLayout:

```
Const MainLayout =()=> (
  <div>
    <Navbar />
    <Outlet />
```

```

    </div>
);

```

Важливими сторінками на сайті є TicketsPage та ProfilePage, а також функціонал зміни мови LanguageSelector. У цих двох сторінках реалізується головна мета сайту, продаж квитків та додаткові функції для зареєстрованих користувачів.

Невід’ємною частиною сучасних сайтів є інтеграція для мобільних пристроїв, не всі люди мають можливість використовувати комп’ютери або ноутбуки для перегляду веб-сторінок, тому це не тільки ефективність залучення нових користувачів, але й зручність використання сайту, що позитивно впливають на відгуки. @media – запити Media Queries що дозволяють задавати різні стилі для різних розмірів екранів у звичайному CSS:

```

@media (max -width: 769 px) {
    .container {
        padding: 10px;
    }
}

@media (min-width: 769px) and (max-width: 1024px) {
    .container {
        padding : 20 px ;
    }
}

```

За допомогою інтеграції Media Queries користувачі мобільних пристроїв можуть замовити квитки в кілька кліків, витративши на це мінімум часу та зусиль.

2.3 Back-end

Back-end – це серверна частина сайту/веб-додатку, що відповідає за обробку даних, зберігання інформації, взаємодію з базами даних та інші функції, приховані від користувача. Back-end забезпечує взаємодію front-end з внутрішньою логікою та даними, які потім відображаються на сторінках. Розробка серверної частини включає наступні мови програмування та програми, які можуть слухати запити і повертати відповідь: JavaScript (Node.js), Java (Spring Boot, Jakarta EE), Python (Django, Flask), PHP (Laravel), C# (.NET). Back-end обов’язково працює з базами даних де

зазвичай зберігається інформація про користувачів, замовлення, повідомлення і т.д. Поділяють два типи баз даних:

SQL (структуровані): MySQL, PostgreSQL, SQLite.

NoSQL (гнучкі): MongoDB, Firebase.

Структурована база даних – це реляційна база даних, де дані зберігаються у вигляді таблиць із чіткою схемою, такі дані збережені та організовані за допомогою спеціальних програм, що дозволяють ефективно зберігати, керувати та отримувати доступ до інформації. Структуровані бази даних базуються на певному наборі правил, що визначають структуру, формат, розмір даних, які забезпечують швидкий доступ, а також легко автоматизують операції [3].

Гнучкі БД – це нереляційні бази даних, які забезпечують гнучкість у структурі даних, дозволяючи легко адаптуватися до вимог, що змінюються, і потребам на відміну від реляційних баз даних. Такі бази даних не обмежені строгою структурою, а дозволяють зберігати дані в різних форматах і структурах, таких як документи, пари ‘ключ-значень’ або графи [3].

Головною та невід’ємною частиною back-end є API. API – це частина веб-програми або програми, яка має певний набір правил, за якими front-end взаємодіє з back-end та відповідає за обробку запитів, взаємодію з базою даних, логіку бізнесу. API – зазвичай реалізується за допомогою REST або GraphQL запитів. Приклад REST API:

GET/users/5 – отримання даних користувача з ID 5;

POST/login – авторизація;

Є багато різних видів запитів REST API, але виділяють найпоширеніші:

GET – отримання даних;

POST – створення нових даних;

PUT або PATCH – оновлення;

DELETE – видалення;

Back-end також відповідає за безпеку, тому обов'язково реалізує захист даних, наприклад: хешування паролів, авторизацію та аутентифікацію, валідацію вхідних даних, захист від SQL атак SQL -інекції, XSS і т.д.

Як зазвичай може бути реалізований Front-end і Back-end, прикладом можна взяти онлайн-магазин:

Front-end: користувач бачить товар, додає його до кошика, натискає кнопку ‘Купити’.

Back-end: обробляє замовлення, перевіряє наявність товару, виконує транзакцію, записує в базу даних та відправляє відповідь до користувача або email.

Проект FashaSkyLine використовує фреймворк Spring Boot на мові програмування Java для серверної частини, PostgreSQL для баз даних та REST API для запитів до front-end .

Spring Boot – це фреймворк, що спрощує та прискорює розробку додатків на основі Spring Framework [4]. Він автоматизує налаштування та допомагає швидко створити, запустити та розгорнути веб-додатки. Основні переваги:

Мінімум конфігурацій;

Вбудовані сервери;

Підтримка REST API;

Інтеграція з базами даних;

Структурований підхід до написання back-end;

Просте підключення безпеки, авторизації, реєстрації;

Працює в екосистемі Spring (Spring MVC, Spring Security, Spring Data);

Щоб створити проєкт Spring Boot для взаємодії з front-end потрібно дотримуватися наступних кроків:

Вибрати залежність Spring Web, Spring Data JPA, MySQL Driver;

Створити Java класи-контролери, які будуть обробляти запити;

Створити Entity – класи для роботи з базою;

Налаштувати підключення до БД в application.properties;

Чому саме Spring Boot використано при створенні проєкту?

Spring Boot дозволяє швидко розгорнути робочий сервер без складної конфігурації. Завдяки анотації @RestController проєкт відразу зможе обробляти HTTP-запити, а з використанням Spring

Data JPA легко підключити базу даних PostgreSQL без написання SQL-запитів достатньо створити Entity-класи та відповідні репозиторії.

Забезпечення зручної реалізації API-ендпоінтів (наприклад, /api/login, /api/register, /api/profile), які використовуються для взаємодії з front-end. Для захисту маршрутів використано Spring Security, що дозволило додати авторизацію, токени доступу та обмежити доступ до ресурсів.

Завдяки чіткій структурі проєкту (контролери, сервіси, репозиторії) отримано модульність та масштабованість, а також можливість легко підтримувати та доповнювати додаток у майбутньому. Крім того, Spring Boot має гарну інтеграцію з такими інструментами, як Maven, Docket і середовище IntelliJ IDEA, що значно спростило процес розробки та розгортання програми.

При створенні back-end для сайту FashaSkyLine, була визначена зручна та чітка структура проєкту (схема 2).

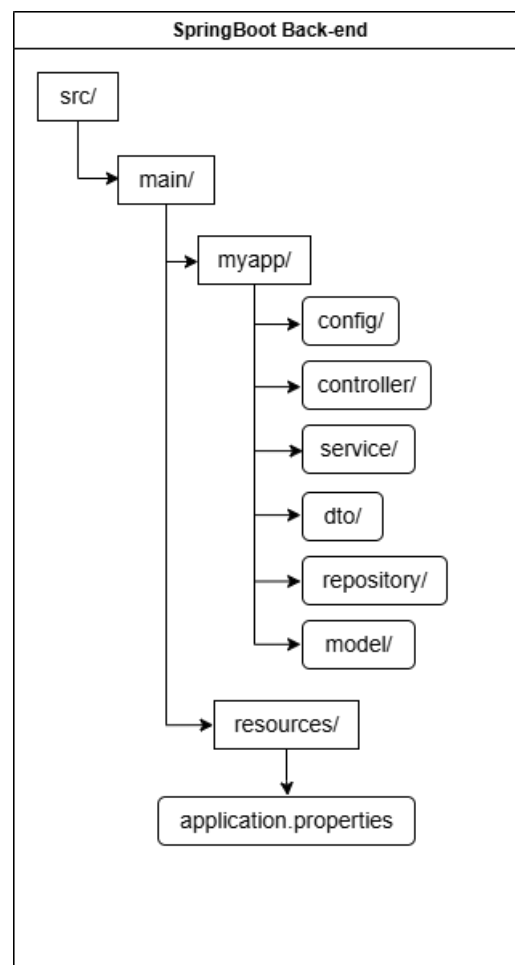


Схема 2

Config – визначає конфігурацію сервера для прийому даних від певних методів (GET, POST, PUT, DELETE), а також виконує захисну функцію перевірки запитів HTTP-адрес.

Controller – включає API-контролери, які приймають запити з React та надсилають оброблені дані на сайт.

Dto – це об'єкт, який використовується для передачі лише тих даних, які дійсно потрібні між різними шарами проєкту, зазвичай між back-end та front-end або між сервісами та контролерами. Наприклад, користувач шукає іншого користувача в таблиці на сайті тому React робить запит до back-end бази даних, що містить у собі id, name, lastname, email, password, щоб користувач не отримав всі дані з бази даних використовується dto який безпечно надає тільки ту інформацію, яку може переглядати звичайний користувач id, name, lastname.

Model – зберігає класи Entity, які безпосередньо взаємодіють і реалізують структуру бази даних.

Repository – це шар у програмі, що відповідає за доступ до бази даних. Він містить методи зчитування, збереження, оновлення та видалення даних.

Service – це частина архітектури, що містить основну логіку обробки даних, перевірки, обчислення, взаємодії з базою даних і т.д. Сервіси не працюють безпосередньо з HTTP-запитами або збереженням даних вони приймають дані від контролера і взаємодіють з репозиторіями, щоб потім повернути результат.

Щоб зберігати дані про користувачів, квитки або замовлення було вирішено використовувати потужну об'єктно-реляційну базу даних PostgreSQL, яка підтримує не лише стандартні реляційні функції, але й складні типи даних, JSON розширення та зберігання процедури. PostgreSQL відмінно підходить для проєктів, де важлива надійна обробка даних та зберігання цілісності навіть при великих обсягах інформації, а також повністю підтримується ORM- бібліотеками, наприклад Spring Data JPA дозволяє автоматично створювати запити до БД через Java-класи. Перед використанням бази даних були створені таблиці, з якими взаємодіятиме back-end і front-end:

Users – таблиця зареєстрованих користувачів, яка використовується для входу користувачів у свій профіль на сайті, а також при купівлі квитків або додаванні кредитної картки (таблиця 1).

user
id;
last_name;
first_name;
email;
password_hash;
created_at;

Таблиця 1

Credit_card – таблиця для імітації кредитних карток за допомогою якої можна оплачувати купівлю квитка (таблиця 2).

credit_card
id;
card_number;
expiry_date;
cvv;
balance;
user_id;

Таблиця 2

Tickets – таблиця для зберігання інформації про квитки, яка буде використана для створення замовлення та оплати через платіжну картку (таблиця 3).

tickets
id; from_country; from_airport; to_country; to_airport; departure_days; departure_time; price;

Таблиця 3

Orders – таблиця замовлень, що зберігає інформацію про користувача що купив квиток, час придбання, дані квитка, вартість оплати та статус замовлення (таблиця 4).

orders
id; user_id; ticket_id; order_date; departure_date; flight_class; final_price; status;

Таблиця 4

Щоб детально зрозуміти взаємодію між таблицями PostgreSQL створено наступний фрагмент (схема 3).

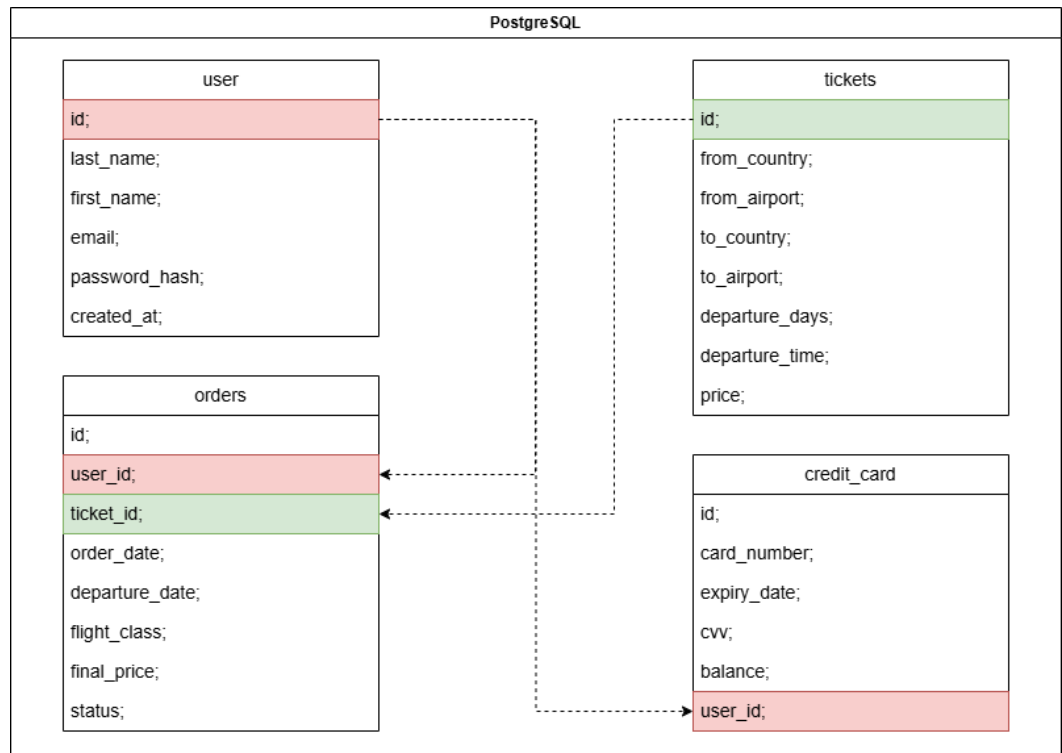


Схема 3

Тепер для реалізації бази даних у проєкті back-end у файлі `application.properties` використовуються визначення для підключення до бази даних PostgreSQL:

```

spring.datasource.url=jdbc:postgresql://localhost:5432/postgres
spring.datasource.username=postgres
spring.datasource.password=2045
spring.datasource.driver-class-name=org.postgresql.Driver
  
```

Після цього у директорії `model` створено Entity клас прикладом буде таблиця `users` (Додаток A1).

Тепер розглянемо використані інструкції в класі `User`:

`@Entity` – вказує на те, що клас є сутністю (entity), тобто його об'єкти будуть зберігатися в базі даних як рядки таблиці.

`@Table (name = "users")` – вказує назву таблиці у базі даних, з якою пов'язаний клас `User`.

`@Id` – означає початковий ключ таблиці. У нашому випадку – це поле `id`.

`@GeneratedValue (strategy = GenerationType.IDENTITY)` – означає, що значення `id` буде автоматично згенеровано базою даних під час створення нового запису.

`@Column` – визначає кожне поле, відповідне колонці таблиці.

`@Column (unique = true, nullable = false)` – інструкції всередині `Column` вказують на те, що поле буде унікальним – тобто в базі не можна зберігати два однакові значення і також є обов'язковим до заповнення.

`Lombok (@Getter , @Setter)` – це анотації бібліотеки `Lombok` , які автоматично генерують гетери та сеттери для всіх полів, щоб не писати їх вручну.

Таким чином реалізовано модель `Entity`-класу яка безпосередньо взаємодіє з базою даних та підготовлена до використання у `dto` та репозиторіях.

Другий розділ присвячено структурі проєкту сайту, який поділено на дві основні частини: фронтенд та бекенд. У розробці фронтенду було використано фреймворк `React`, що дозволило створити динамічний та інтерактивний інтерфейс за допомогою компонентного підходу. Для маршрутизації використовувалася бібліотека “`react-router-dom`”, яка забезпечила зручну навігацію між сторінками. У розробці бекенду було використано фреймворк `Spring Boot`, що спростило створення серверної частини, забезпечивши взаємодію з базою даних `PostgreSQL` через `Spring Data JPA`. Особлива увага була приділена безпеці даних, включаючи хешування паролів та захист від `SQL-ін'єкцій` [5]. Структура проєкту є модульною, що сприяє його супроводженню та масштабуванню.

3. РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ ПРОДАЖУ/КУПІВЛІ КВИТКІВ, ОСНОВНІ ФУНКЦІЇ САЙТУ ТА ЗАХИСТ ДАНИХ.

3.1 Реєстрація/Вхід в профіль користувача.

На сайті компанії FashaSkyLine розроблено функцію для користувачів, які хочуть отримати більше можливостей для керування своїми особистими даними, платіжними картками або переглядом замовлених квитків. Для переходу в особистий кабінет користувачеві потрібно зареєструватися, тому при натисканні інтуїтивно зрозумілої іконки на головній стрічці сайту у верхньому правому куті (рисунок 3) з'являється вікно з двома кнопками 'увійти' або 'зареєструватися' (рисунок 4).



Рисунок 3

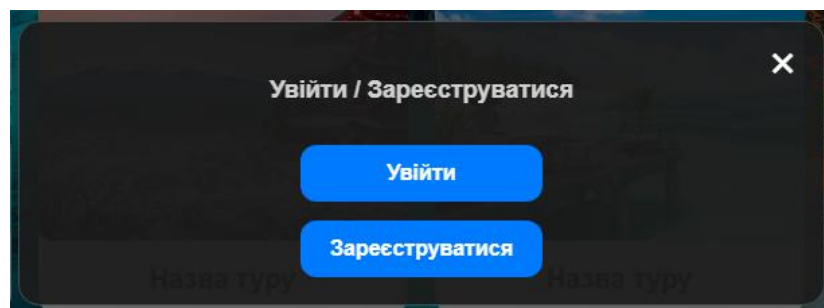


Рисунок 4

Розглянемо, як це виглядає у коді проєкту. Щоб викликати вікно, реалізується кілька методів виклику у різних JavaScript файлах.

Navbar.js – відповідає за навігаційну панель сайту в якій і знаходиться кнопка (Додаток A2).

При натисканні “profile-button” яке відслідковується за допомогою обробника подій натискання “onClick={...}” викликається наступна перевірка “if(!user)”, яка перевіряє чи зареєстрований користувач якщо так то користувач надсилається

за адресою “/profile”, а якщо ні, то спрацьовує функція “on-profileClick”. Також реалізовано відображення “user” імені замість іконки профілю, а саме “lastName” та “firstName” за умови, що користувач зареєстрований або увійшов до свого акаунту.

Функція “onProfileClick()” передається до головного компонента сторінки MainLayout.js, де реалізується логіка відкриття вікна входу/реєстрації LoginRegisterWindow.js (Додаток A3).

`<Navbar onProfileClick {...}/>` – властивість “onProfileClick”, яка передається в Navbar головно компонента MainLayout.js встановлюючи стан “isLoginOpen” – true, тобто дозволяє відкриття модального вікна LoginRegisterWindow.js. Після успішного відображення вікна з’являються кнопки входу і реєстрації, щоб реалізувати логіку кнопок спочатку була створено функція handleChange (Додаток A4).

HandleChange – використовується для обробки подій зміни в полях введення ‘input’, для форм register і login. Дозволяє динамічно оновлювати стан компонентів React залежно від активного кроку (входу / реєстрації).

Параметр ‘e’ – об’єкт події, що генерується браузером під час взаємодії користувача з полем введення. Далі виконується умова, якщо (step === “register”), то оновлюється стан formData через функцію setFormData, що зберігає дані реєстраційної форми. (step === “login”) ідентична умова, але оновлює інший стан loginData.

Реалізація кнопок у HTML (Додаток A5).

Логіка перемикавання стану в HTML (Додаток A6).

Взаємодія файлів та функцій між собою (схема 4).

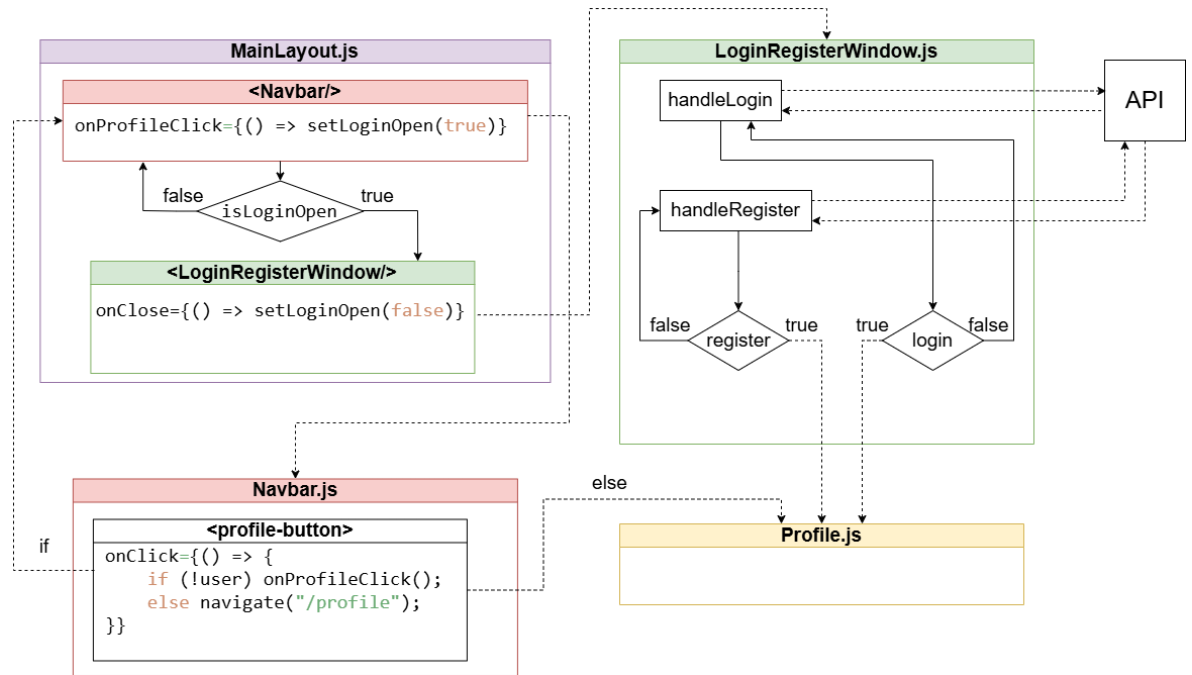


Схема 4

3.2 Реалізація форми реєстрації на front-end і back-end.

Якщо користувач вибрав форму реєстрації та заповнив усі поля (рисунок 5), то після підтвердження дані будуть надіслані через API запит до back-end.

The screenshot shows a registration form titled "Реєстрація" (Registration) with a close button (X) in the top right corner. The form contains four input fields:

- Field 1: "user" (placeholder text)
- Field 2: "users" (placeholder text)
- Field 3: "user@gmail.com" (placeholder text)
- Field 4: "123456789" (placeholder text) with an eye icon for toggling visibility.

At the bottom of the form, there are two blue buttons:

- "Зареєструватися" (Register)
- "Назад" (Back)

Рисунок 5

За відправку даних реєстрації в back-end з front-end відповідає функція `handleRegister` що викликає API запит (Додаток Б1).

Оголошується асинхронна функція, що приймає об'єкт події 'e'.

`e.preventDefault()` – забороняє стандартну поведінку браузера, яка зазвичай перезавантажує сторінку після надсилання форми.

`setRegisterError ("")` – очищення попередніх помилок реєстрації, якщо вони були.

`const response = await axios.post (...)` – надсилає POST-запит на сервер за вказаною адресою разом з даними 'formData' і чекає відповідь 'await'.

`localStorage` – зберігає дані користувача у вигляді рядка JSON що дозволяє зберігати сесію навіть після перезавантаження сторінки.

Після успішної реєстрації користувача направляє у його профіль `"/profile"`, але перед тим як розглянути профіль подивимось, що відбувається на back-end, щоб дізнатися, як зберігаються дані та які методи захисту використані для забезпечення безпеки паролів користувача.

За адресою API запиту дані надходять до `AuthController` – контролер для авторизації, який відповідає за обробку HTTP-запитів реєстрації або входу користувачів (Додаток Б2).

Дані користувача надсилаються до класу `ResponseEntity`, на що вказує інструкція `@PostMapping("/register")`, тобто адреса формується так `/api/auth/register`. Після цього JSON тіло десеріалізується у DTO `RegisterRequest`, що містить певні поля передачі даних на обробку в `UserService`, де реалізована головна логіка (Додаток Б3).

Клас `UserService` викликає інтерфейс `UserRepository` для доступу до бази даних користувачів та об'єкт `PasswordEncoder` для шифрування паролів перед збереженням [6]. Також у коді є конструктори 'this', які автоматично передають залежності. Перейдемо до методу `UserResponse`, який приймає дані від DTO, що передаються перед контролером. Спочатку відбувається перевірка, щоб уникнути повторної реєстрації користувачів з однаковим 'email', далі створюється новий об'єкт `User` який заповнюється такими даними: `firstName`, `lastName`, `email`, `password`, але пароль не зберігається у відкритому вигляді, а перед записом хешується за допомогою `passwordEncoder.encode(...)`. Відправлення даних відбувається у репозиторій методом `"User`

savedUser =...” і після успішного виконання створюється об’єкт DTO у UserResponse, щоб віддати лише потрібні клієнту дані (схема 5).

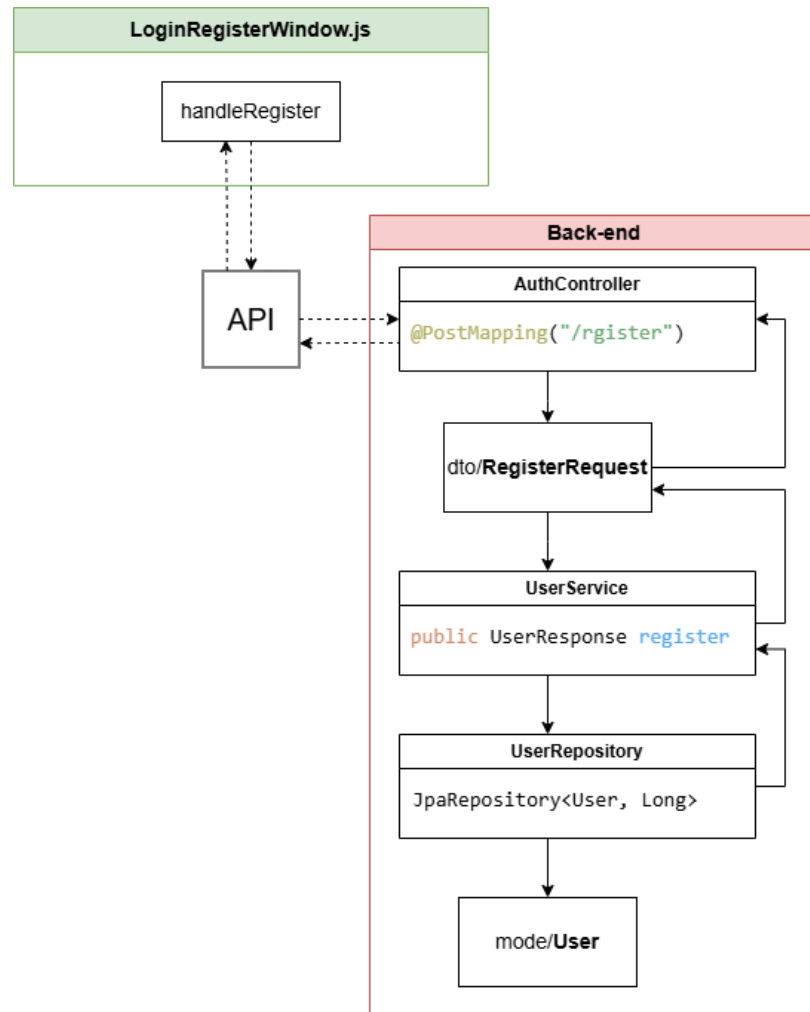


Схема 5

3.3 Реалізація форми входу на front-end і back-end

Якщо користувач вже має обліковий запис на сайті FashaSkyLine і вирішив увійти, то після введення особистих даних ‘email’ і ‘password’ (рисунок 6) створиться запит API до back-end.

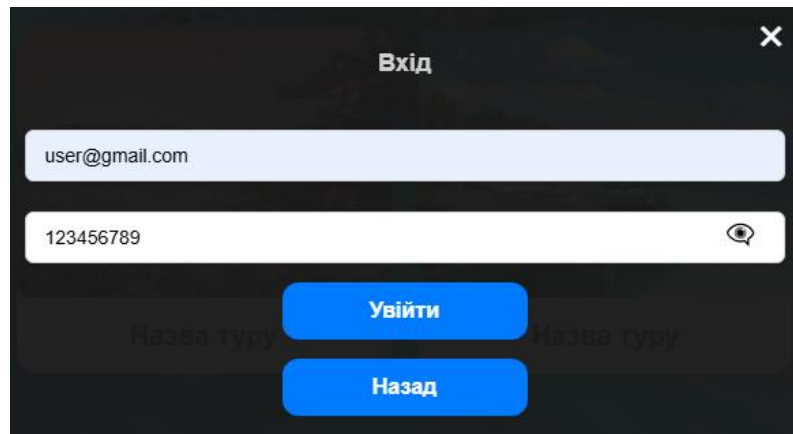


Рисунок 6

Запит на вхід в особистий обліковий запис користувача виглядають майже ідентично як і реєстрація, змінюється тільки адреса і обсяг даних, що надсилаються. Метод реалізується в функції `handleLogin` (Додаток Б4).

Тепер дані надходять на адресу `/api/auth/login` де в `AuthController` є аргумент `@PostMapping(/login)` що вказує на метод `ResponseEntity` (Додаток Б5).

У цьому коді використовується новий компонент `Spring Security AuthenticationManager`, який перевіряє правильність логіну та пароля, хешується та зберігає успішну авторизацію в контексті безпеки.

Клас `UserResponse` робить запит до репозиторію який у свою чергу перевіряє базу даних на схожі дані і якщо такий користувач є, то відбувається повернення даних з back-end у front-end про успішність входу до облікового запису (Додаток Б6) (схема 6).

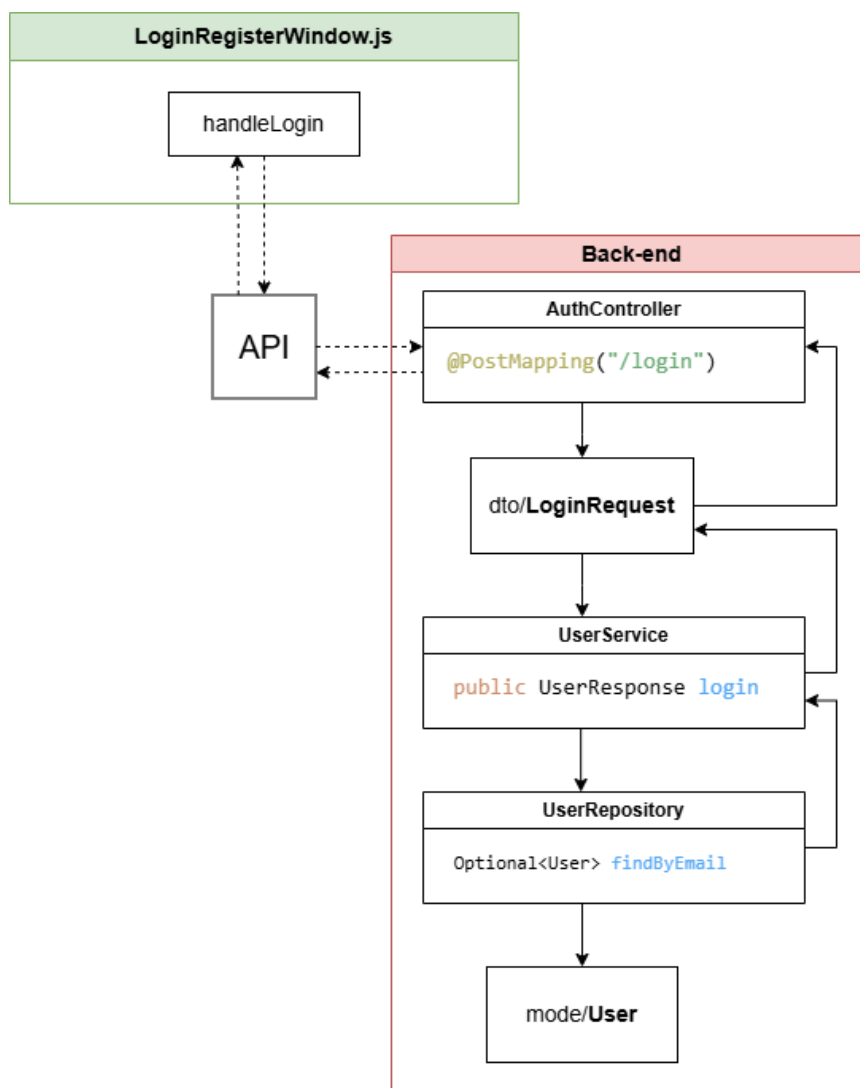


Схема 6

Таким чином, користувач потрапляє у свій профіль на сайті FashaSkyLine (рисунок 7) і отримує перелік додаткових функцій таких як: перегляд історії квитків, додавання платіжної картки, редагування профілю та вихід з облікового запису.

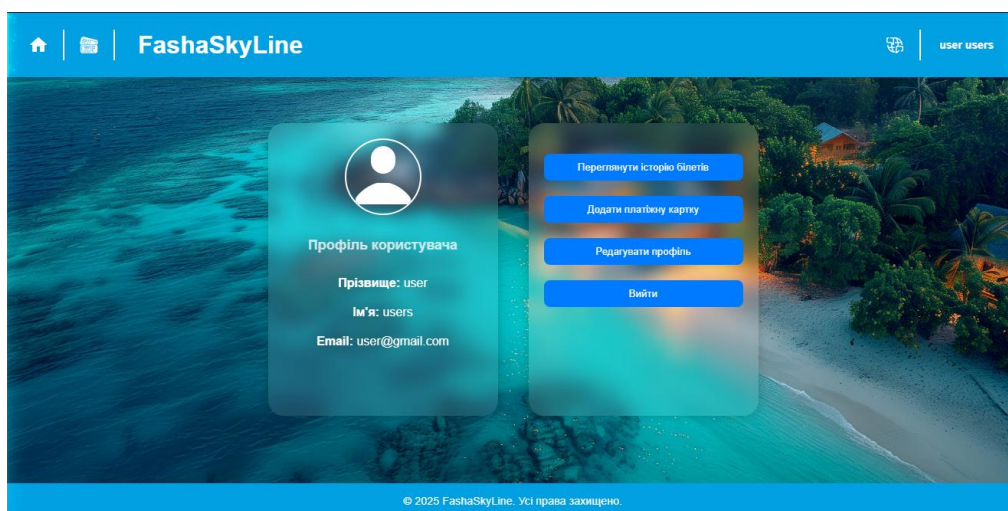


Рисунок 7

3.4 Перегляд та купівля квитків. Front-end.

Основною функцією сайту є продаж квитків для користувачів, які хочуть полетіти до інших країн, при цьому зручно та швидко замовити їх з будь-якого місця. Щоб реалізувати систему продажу квитків створено окрему сторінку `TicketsPage.js` до якої можна перейти натиснувши іконку квитків у лівому верхньому кутку (рисунок 8).



Рисунок 8

Після переходу на сторінку користувачеві відразу виводяться доступні квитки, які можна купити за вказаним напрямком з однієї країни у всі доступні або вибрані (рисунок 9). Також є функція фільтрів, яка допоможе знайти потрібний квиток за вказаною максимальною або мінімальною ціною.

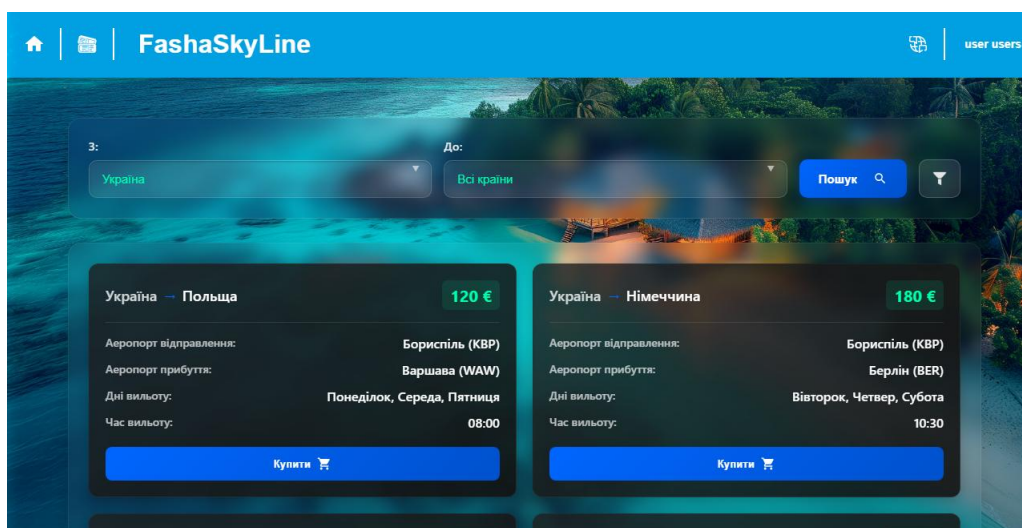


Рисунок 9

Кнопка в `Navbar.js` для переходу на сторінку квитків виглядає набагато простіше тому що виконує звичайне переміщення користувача за адресою, вказаною у файлі маршрутизації `App.js` (Додаток B1).

Так описана звичайна кнопка HTML-структури, яка при натисканні відстежується завдяки `'onClick'` і направляє користувача на адресу `'/tickets'`. Для рендерингу `TicketsPage.js` у головній структурі компоненту `MainLayout.js` використано спеціальний компонент бібліотеки `React Router`, який є місцем для вставки дочірніх маршрутів. Ось як це виглядає:

`Main` – HTML-обгортка для основного вмісту сторінки, де відбувається рендеринг `<Outlet />` різних сторінок залежно від поточного URL (Додаток B2).

Тепер перейдемо до вмісту `TicketsPage` та реалізації пошукової панелі та панелі для виведення інформації про квитки, що є важливою функціональною частиною сторінки (Додаток B3).

Обробник `'onSubmit'` у компоненті `<form>` відповідає за відправку пошукового запиту в функцію `'handleSearch'` викликається автоматично або при натисканні кнопки пошуку. Блок `"country-selectors"` містить два списки `<select>` вибору країни відправлення та призначення `'fromCountry'`, `'toCountry'` які динамічно формуються за допомогою `'countries(...)'`, що дозволяє створити список `<option>` (Додаток B4).

Такий вигляд має список доступних країн. У коді всі назви записані як ключі, які використовуються для перекладу іншими мовами. Також є окремі унікальні ключі, які створює функція

‘generateTicketKey’ для кожного окремого квитка у списку React для оптимізації рендеру (Додаток B5).

Далі перейдемо до блоку, де реалізовано відображення результату пошуку квитків (Додаток B6).

Тому що в контейнері ‘tickets–result’ є умовний рендер тобто відображення стану квитків залежно від умови “{loading?(...): tickets.length > 0?}” якщо (loading === true):

Є квитки (tickets.length > 0)

Немає квитків (tickets.length === 0)

Щоб користувач розумів, чи завантажуються квитки чи ні, створено контейнер із станом ‘loading-container’, який візуалізує цей процес. Після успішного знаходження створюється сітка для відображення карток з отриманою інформацією в блоці ‘tickets-grid’. У тегах відображаються отримані дані від back-end: країна відправлення, країна прибуття, ціна, аеропорт відправлення, аеропорт прибуття, дати польоту, час вильоту. Якщо квитки за обраними критеріями були знайдені, тоді використовується блок ‘no–tickets’ повідомляє користувача про їх відсутність.

Вибравши квиток, користувач натискає кнопку ‘buy-button’, яка викликає і передає значення ‘ticket’ в ‘openModal(ticket)’, а саме вікно відкривається окремо, але рендер відбувається в TicketsPage (Додаток B7)

<OrderModal> отримує набір вхідних параметрів, які використовуються для відображення, керування станом та оплатою квитка (схема 7). Модальне вікно можна розділити на дві частини: перша – вибір класу польоту бізнес/економ, друга – підтвердження квитка та оплата.

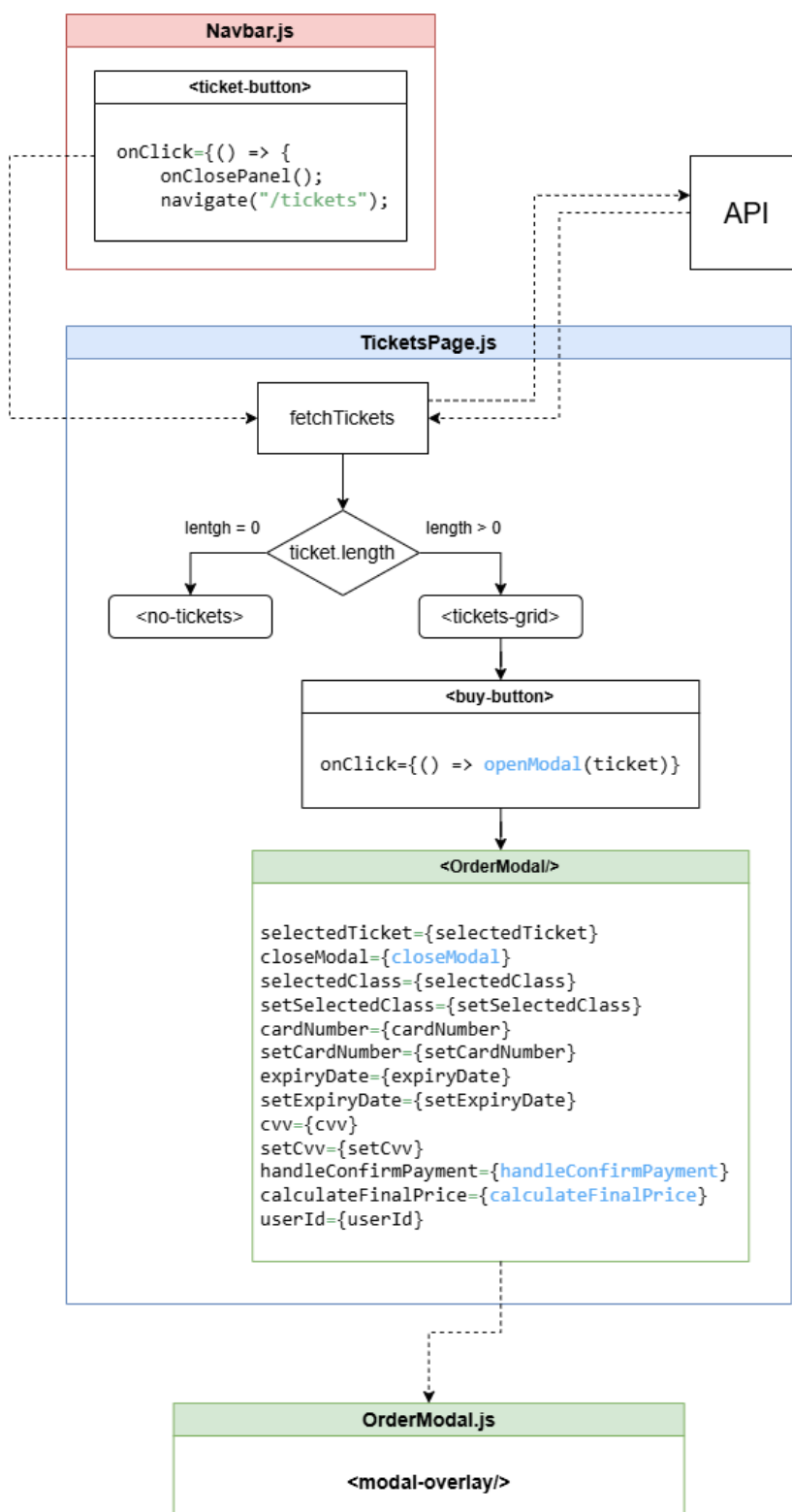


Схема 7

У першій частині все просто, 'modalStage' значення змінної або 'select' якщо так відкривається вікно з повторенням даних квитка та вибором певного класу для польоту в блоці 'modal-option' (рисунок 10)(Додаток В8). Після вибору в 'handleClassSelect' зазначеної в кнопках передається значення

‘economy’ або ‘busines’, у свою чергу функція дає згоду на перехід до наступного модального вікна (рисунок 11, 12), запускає анімацію переходу та надсилає дані через `setSelectedClass(classType)` для обробки фінальної ціни (Додаток B9).

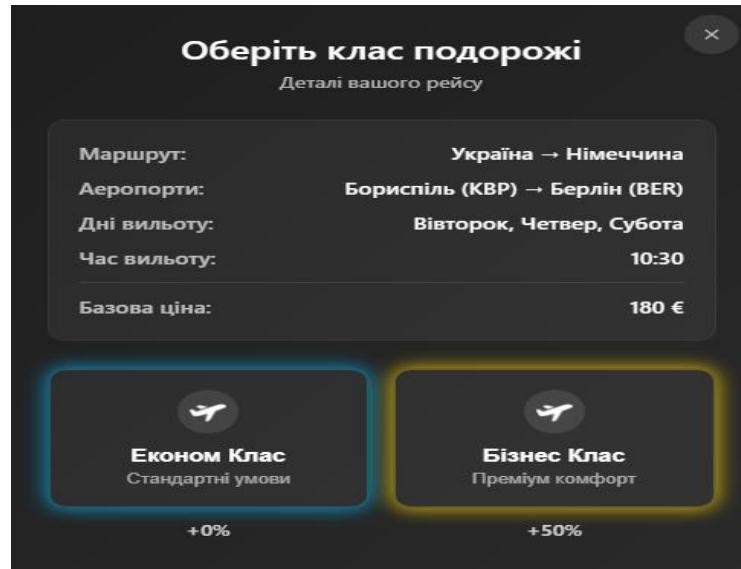


Рисунок 10

Обробка фінальної ціни функцією ‘`calculateFinalPrice`’ в `TicketsPage.js` (Додаток B10).

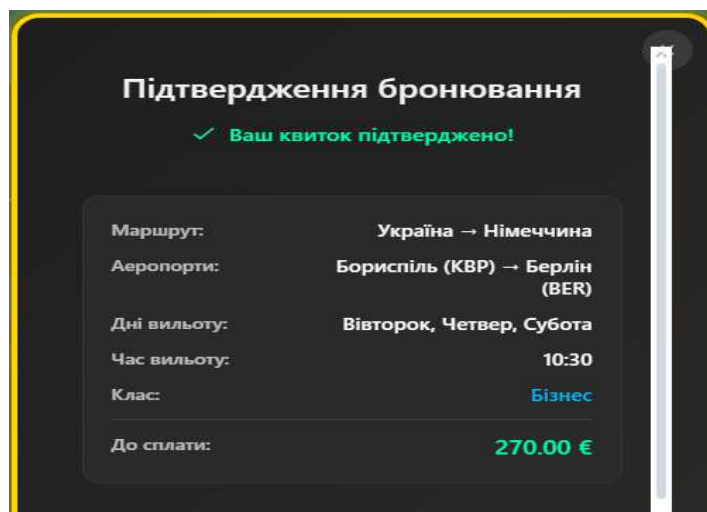


Рисунок 11

Платіжні дані

Номер картки
1234 5678 9012 3456

Термін дії
MM/YY

CVV
123

☐ Зберегти картку для майбутніх платежів

Підтвердити оплату ✓

Рисунок 12

Друга частина складається з передачі всіх даних про квиток, фінальну ціну та визначення кредитної картки, якщо користувач додавав її до профілю, то вона автоматично буде використана для зручної оплати (рисунок 13).

Підтвердження бронювання

✓ Ваш квиток підтверджено!

Маршрут: Україна → Німеччина

Аеропорти: Бориспіль (KBP) → Берлін (BER)

Дні вильоту: Вівторок, Четвер, Субота

Час вильоту: 10:30

Клас: Бізнес

До сплати: 270.00 €

Картка: **** * 9878
Баланс: 0 €

Підтвердити оплату ✓

Рисунок 13

Далі переглянемо код створення такого вікна на підтвердження оплати (Додаток B11).

Блок 'ticket-details' включає інформацію про квиток і фінальну ціну яка розрахована на back-end повернена до front-end, надалі йде завантаження платіжної картки якщо користувач вже має додану картку до свого облікового запису в структурі 'existing-

card’, але якщо у користувач немає карти відбувається завантаження полів для введення даних картки. Кнопка ‘confirm-button’ викликає перевірку даних та виводить користувачу певну відповідь залежно від успішності виконання операції (схема 8).

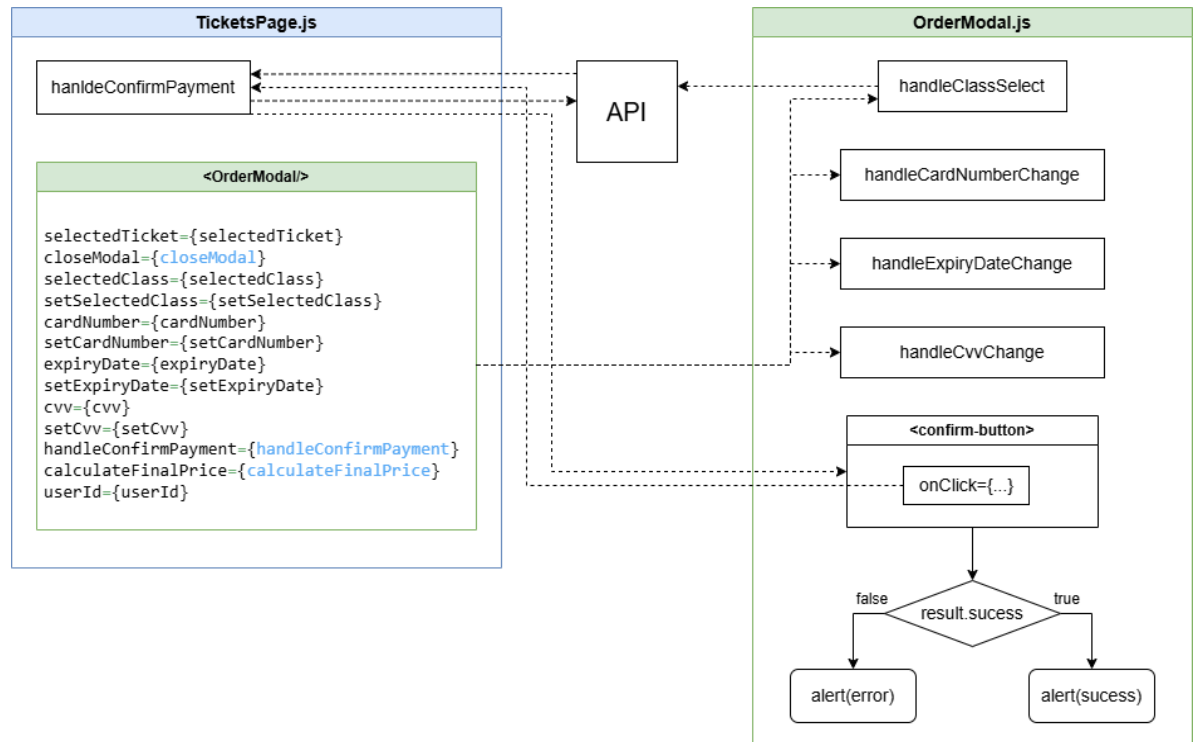


Схема 8

3.5. Аналіз передачі даних квитків з back-end на сайт

Для початку щоб сформувати таблицю з переліком квитків сторінка TicketsPage використовує функцію ‘fetchTickets’ викликає запит до серверної частини методом ‘response’ у відповідь при успішному виконанні сервер повертає масив ‘data’ який за допомогою утиліти ‘URLSearchParams()’ розпаковується та формується рядок запиту append(‘fromCountry’, fromCountry) інші додаються, лише якщо вони задані (Додаток Г1).

Щоб завжди відображати актуальні зміни у таблиці квитків використовується бібліотека React ‘useEffect’, яка завжди буде оновлювати fetchTickets() (Додаток Г2).

Перейдемо до серверної частини, де відбувається отримання запиту на надсилання даних про всі доступні рейси. Для роботи з квитками створено окремий TicketController що містить автоматичне впровадження @Autowired класу TicketService без

якого не може працювати контролер і методи 'searchTickets' який є масивом. 'List <Ticket>' також приймає різні @RequestParam які не є обов'язковими, але це можуть бути параметри фільтра max/min ціни або країни відправлення та прибуття (Додаток Г3).

Далі дані надсилаються з контролера до класу логіки через return. TicketService також містить @Autowired та виклик класу TicketRepository. Метод 'searchTickets' є списком, який надсилає параметри пошуку до репозиторію TicketRepository для взаємодії з базою даних (Додаток Г4). У TicketRepository є анотації @Query і @Param які захищають від SQL-ін'єкцій, що підставляють лише безпечні іменовані параметри, наприклад, якщо введеться запит 'DROP TABLE ticket', він буде сприйнятий, як рядкове значення, а не SQL-запит. Також у репозиторію формується масив з квитків, який потім буде відправлений назад у front-end (Додаток Г5) (схема 9). Таблиця квитків Ticket(Додаток Г6).

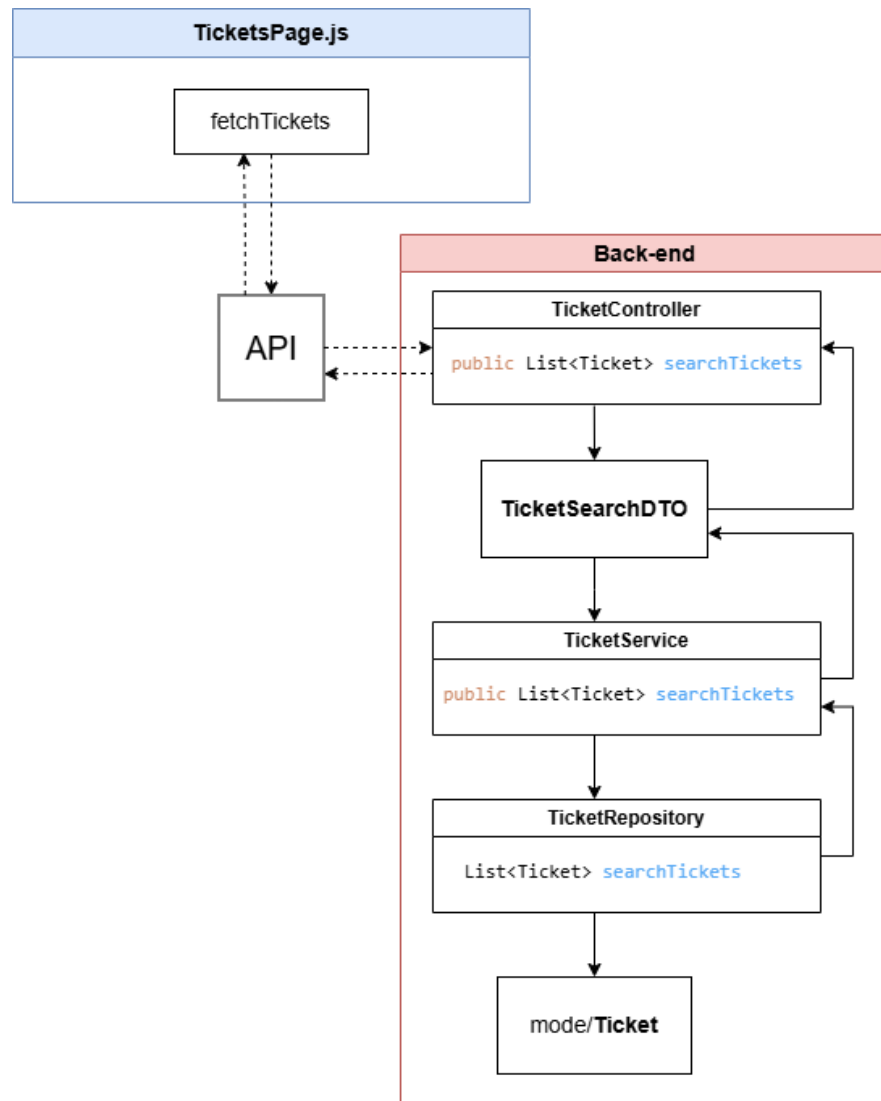


Схема 9

3.6 Логіка виконання транзакцій.

Користувач після вибору класу польоту переходить на фінальну частину замовлення, тому формується ордер. Для цього викликається функція `handleConfirmPayment` в `TicketPage.js`, що надсилає дані на обробку до сервера і підставляє платіжну картку якщо користувач додав її, якщо ні, тоді використовується стандартний метод введення даних. Константа `orderResponse` відправляє запит на створення ордера зі сформованими даними до back-end (Додаток Г1).

За прийом даних на формування ордера відповідає `OrderController`, структура схожа з іншими контролерами. Щоб створити ордер, використовується функція `createOrder`, яка використовує дані користувача, інформацію про квиток, фінальну ціну та дані картки. Перш ніж розпочати транзакцію, дані валідуються, тобто перевіряються, а вже потім все надсилається в `OrderService` (Додаток Г2).

Перейдемо до `OrderService`, який отримав дані та починає виконувати транзакцію.

Транзакція – це будь-яка дія або кілька дій, що змінюють стан об'єкта чи систему. Зазвичай передбачається взаємодія з базою даних, фінансові операції чи обмін даними. Особливість транзакції в тому, що якщо під час цього процесу виникає помилка, то весь процес скасовується, наприклад користувач замовив товар і при оплаті сталася помилка, товар не був куплений, але кошти списалися з картки – це не правильне виконання оплати, тому використовуються транзакції.

У нашому випадку для їх реалізації в `OrderService` є анотація `@Transactional` до функції `createOrder`, що гарантує цілісність операції – якщо десь сталася помилка, всі зміни скасовуються. Переглянемо, які операції відбуваються під час транзакції (Додаток Г3).

- Пошук id користувача .
- Пошук id квитка.
- Перевірка даних платіжної картки.
- Пошук карти у базі даних.
- Перевірка балансу картки.

- Зняття коштів.
- Створення замовлення є ордером.
- Збереження до бази даних

Коли операція успішно виконується дані готового замовлення відправляються в репозиторій `OrderRepository`, де метод `JpaRepository` зберігає об'єкт `Order` в базу даних (Додаток Г4) (схема 10).

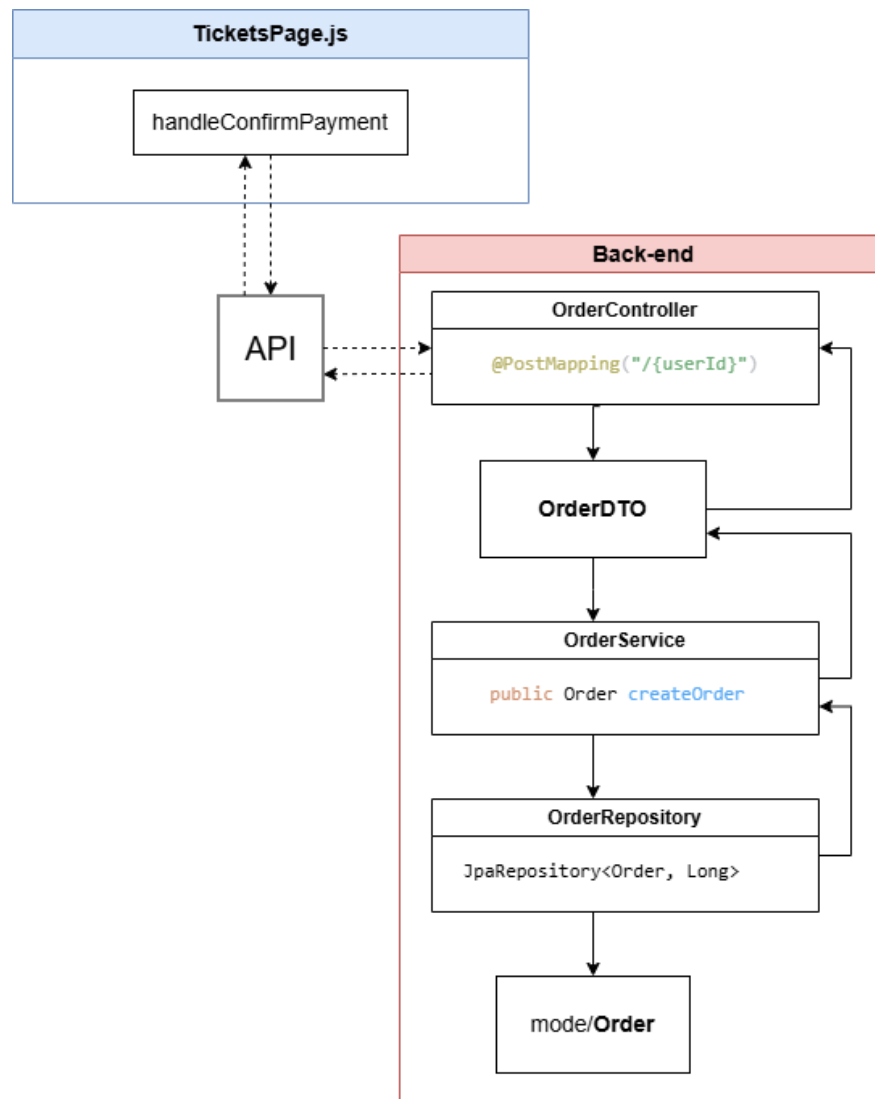


Схема 10

Вигляд бази даних у моделі `Order` і яка його структура (Додаток Г5).

- ID замовлення.
- `User_id` користувача до якого прив'язане замовлення.

- Ticket_id квитка що купують.
- Order_date запис дати створення замовлення.
- Fligth_class вибрано клас польоту.
- Final_price фінальна ціна була сплачена.
- Status статус замовлення.
- Card_number номер картки якої було сплачено квиток.

Після успішного придбання користувач отримує повідомлення на сайті що квиток був куплений.

3.7 Реалізація функціоналу багатомовності сайту

Компанія FashaSkyLine представляє не тільки зручність купівлі квитків, а й багатомовність сайту для користувачів з різних країн. Щоб змінити мову створено окремий файл LanguageSelector.js відповідальний за відображення списку різних мов, але для роботи з перекладом використовується окрема бібліотека i18next яка локалізує текст у веб-додатках. Разом з react-i18next вона дозволяє легко змінювати мову інтерфейсу, не перезавантажуючи сторінку (Додаток E1).

У цьому коді функція 'changeLanguage' приймає змінну 'lng' яка у свою чергу передається від обраної користувачем мови. Після цього значення передається у конфігурацію i18n.js, де вказано шлях до JSON-файлу що містить ключі з перекладом (Додаток E2) (схема 11).

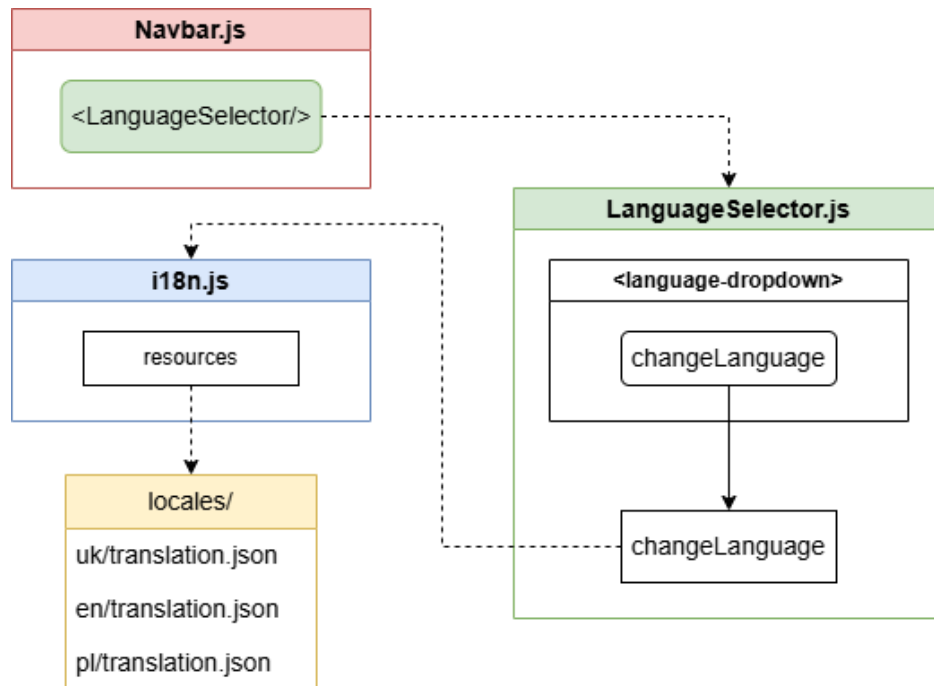


Схема 11

Ключі перекладу зберігаються в папках `locales` зі скороченою назвою: `uk`, `en`, `es`, `pl`, `it` і використовуються сайтом замінюючи звичайні слова в `MainLayout` на `'modal.h2'` (Додаток E3).

3.8 Реалізація додавання карток для оплати квитків.

Щоб додати картку для оплати користувачеві потрібно зайти в профіль і натиснути кнопку 'Додати платіжну картку', після чого викликається перевірка `'fetchUserData'` в активному `'useEffect'`, що дозволяє завжди відстежувати зміни у профілі для коректного відображення інформації. Якщо картка знайдена кнопка 'Додати платіжну картку' не з'явиться, а відбуватиметься наступне (Додаток Є1).

Цей метод запитує серверну частину `CreditCardController` за адресою `/api/cards/${user.id}` для пошуку користувача та його платіжної картки (Додаток Є2).

Метод `'getCard'` отримує `'id'` користувача і відправляє його в `CreditCardService`, який у свою чергу запитує `CreditCardRepository`, щоб перевірити наявність карти і відправити відповідь назад у `front-end` (Додаток Є3).

Надсилання даних в репозиторій (Додаток Є4).

Завантаження платіжної картки на стороні `front-end` відбувається лише коли вона існує. З метою безпеки користувачу

за допомогою функції `maskCardNumber` виводиться лише останні чотири цифри (рисунок 14) (Додаток Є5) (схема 12).

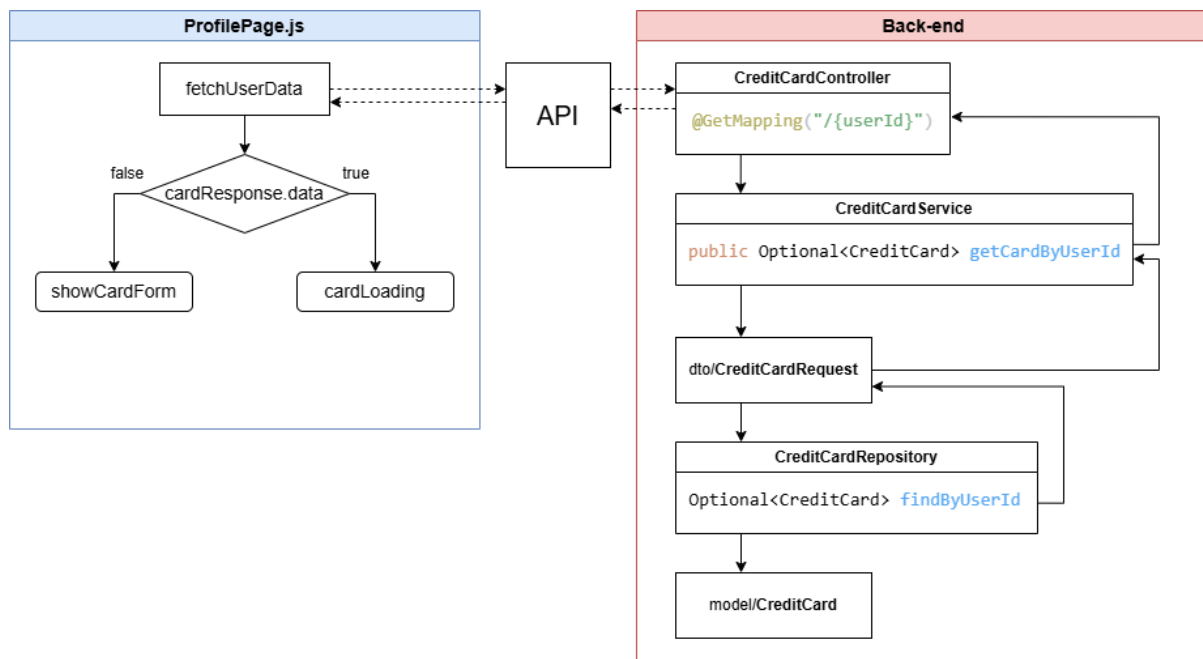


Схема 12

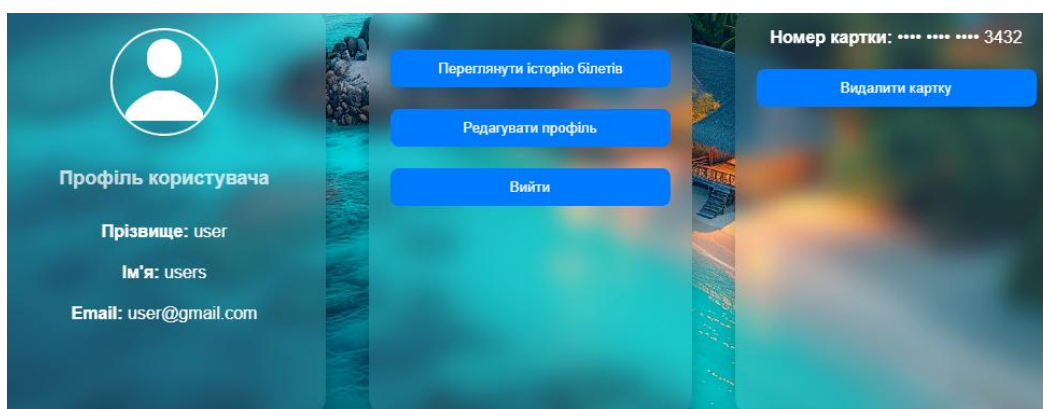


Рисунок 14

Тепер розглянемо, що відбувається, якщо у користувача немає платіжної картки і він хоче додати її. У HTML-структурі на сторінці профілю користувача відкривається блок, в якому є поля для введення даних для надсилання їх на сервер (рисунок 15) (Додаток Є6).

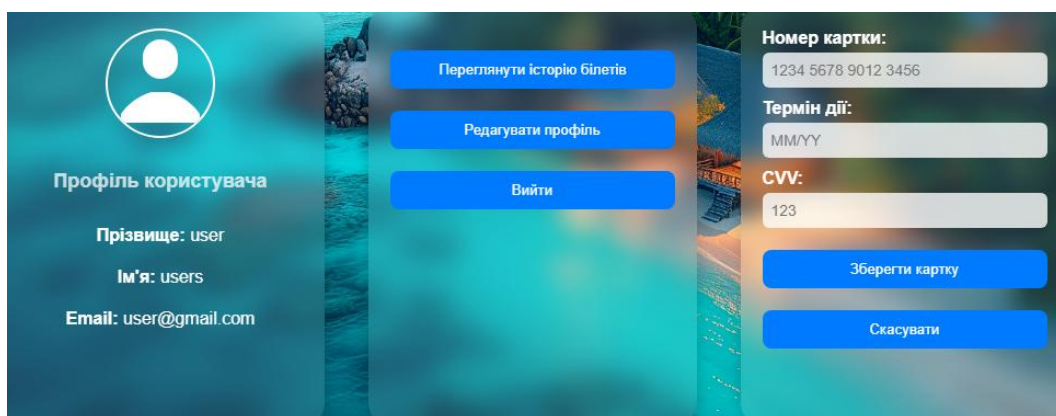


Рисунок 15

Для надсилення даних використовується функція 'handle SaveCard', яка очищає поля введення від непотрібних символів, що автоматично створюються і посилає запит на сервер CreditCardController за адресою /api/cards/add (Додаток Є7).

Запит приймає функцію addCard і відправляє на обробку в CreditCardService (Додаток Є8).

У CreditCardService перевіряється, чи має користувач платіжну карту і чи є ця карта дублікатом карти іншого користувача, тоді створюється об'єкт і заповнюється отриманою інформацією від контролера, щоб далі передати їх у репозиторій CreditCardRepository, який збереже їх у базу даних з допомогою JpaRepository<CreditCard> (Додаток Є9, Є10) (схема 13).

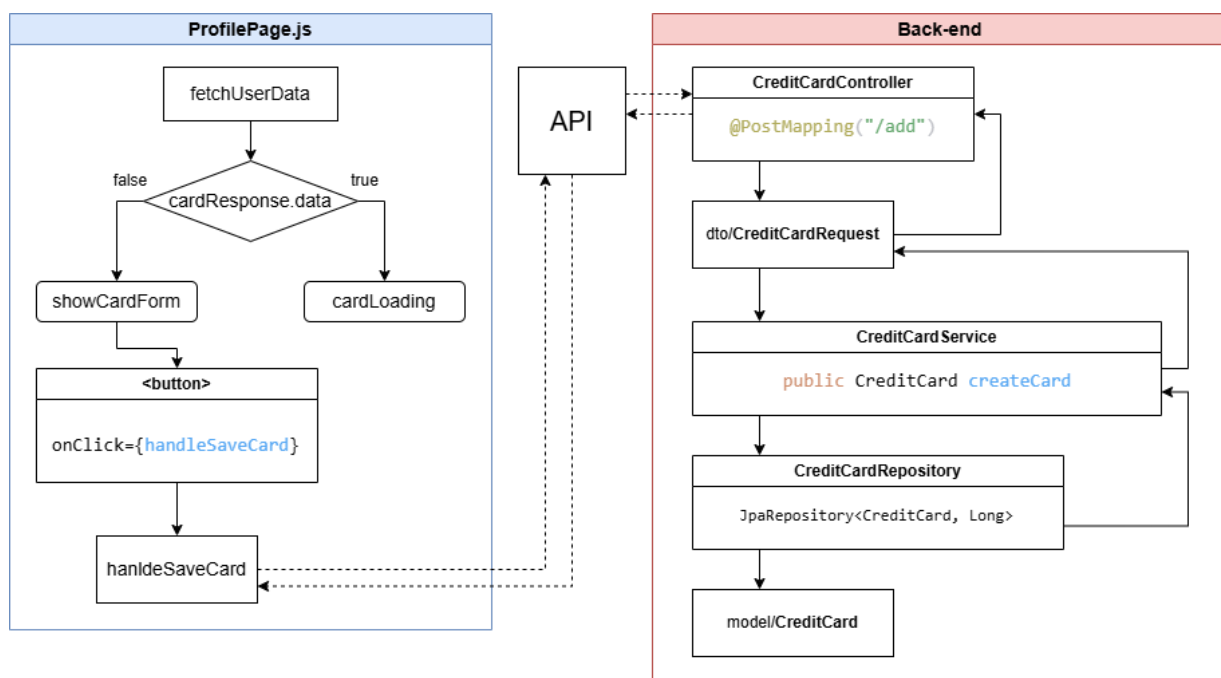


Схема 13

Структура бази даних платіжних карток (Додаток E11).

У третьому розділі детально розглядаються основні функції сайту, такі як реєстрація користувачів, перегляд та купівля квитків, реалізація багатомовності та додавання платіжних карток. Було показано, як фронтенд взаємодіє з бекендом через REST API, забезпечуючи передачу даних у режимі реального часу. Важливим аспектом є реалізація транзакцій, що гарантують цілісність даних під час оплати квитків. Також було підкреслено важливість багатомовності для залучення міжнародних клієнтів. Захист даних реалізується за допомогою шифрування, перевірки вхідних даних та обмеження доступу до конфіденційної інформації. Всі ці функції роблять сайт зручним, безпечним та ефективним інструментом для продажу квитків.

ВИСНОВОК

При створенні сайту використовувалася бібліотека React - JavaScript-бібліотека з відкритим вихідним кодом для розробки інтерфейсів користувача. Спочатку React призначався для Інтернету, створення сайтів, але пізніше з'явилася платформа React Native, яка вже призначалася для мобільних пристроїв.

За своєю суттю React допомагає розробникам створювати повторно використовувані компоненти інтерфейсу користувача, які є будівельними блоками сучасних веб-інтерфейсів, також спрощує подальшу підтримку та оновлення програми.

React відносно простий в освоєнні, має зрозумілий і лаконічний синтаксис, застосовує декларативний підхід до розробки інтерфейсу користувача.

Відмінною особливістю бібліотеки є концентрація на компонентах – ми можемо створити окремі, незалежні один від одного фрагменти коду, а потім легко повторно використовувати, переносити їх із проєкту до проєкту.

Ще одна особливість React – використання JSX. JSX представляє комбінацію коду JavaScript та XML та надає простий та інтуїтивно зрозумілий спосіб визначення коду візуального інтерфейсу.

Фреймворк Spring Boot став ключовим для впровадження ефективного та безпечного back-end для сайту. Він значно пришвидшив розробку завдяки автоматичному налаштуванню, вбудованому серверу Tomcat та зручній структурі (контролери, сервіси, репозиторії). Використання Spring Data JPA спростило та оптимізувало роботу з PostgreSQL, мінімізуючи ручні SQL-запити. Spring Security забезпечив надійний захист, включаючи шифрування паролів, автентифікацію та захист від SQL-ін'єкцій. Розробка REST API для зв'язку з фронтендом була спрощена.

Back-end виконував критично важливу функціональність: централізоване управління даними (користувачі, квитки, замовлення, картки) в реляційній базі даних; реалізацію складної бізнес-логіки (транзакції оплати квитків з перевіркою балансу); забезпечення безпеки за допомогою хешування, перевірки даних та захисту від загроз; інтеграцію з імітованими платіжними шлюзами та глобальними системами. Система також розроблена для масштабованості, здатна обробляти високі робочі навантаження завдяки асинхронним операціям.

На основі цих фреймворків та бібліотек створено сайт нашої авіакомпанії «FashaSkyLine».

СПИСОК ЛІТЕРАТУРИ

1. <https://uk.wikipedia.org/wiki/Вебсайт>
2. <https://wezom.com.ua/ua/blog/что-takoe-back-end-razrabotka>
3. [https://uk.wikipedia.org/wiki/База_даних#:~:text=Структуровані%20та%20неструктуровані%20БД,-Структуровані%20БД%20використовують&text=Існує%20декілька%20моделей%20\(чи%20типів,:%20ієрархічна%20С%20мережна%20та%20реляційна.](https://uk.wikipedia.org/wiki/База_даних#:~:text=Структуровані%20та%20неструктуровані%20БД,-Структуровані%20БД%20використовують&text=Існує%20декілька%20моделей%20(чи%20типів,:%20ієрархічна%20С%20мережна%20та%20реляційна.)
4. <https://spring.io/projects/spring-boot>
5. <https://foxminded.ua/sql-inieksii/>
6. <https://www.baeldung.com/spring-security-registration-password-encoding-bcrypt>

ДОДАТКИ

ДОДАТОК А

1. Приклад таблиці User

```
import jakarta.persistence.*;
import lombok.Getter;
import lombok.Setter;
import java.time.LocalDateTime;

@Getter
@Setter
@Entity
@Table(name = "users")
public class User {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name = "last_name", nullable = false)
    private String lastName;

    @Column(name = "first_name", nullable = false)
    private String firstName;

    @Column(unique = true, nullable = false)
    private String email;

    @Column(name = "password_hash", nullable = false)
    private String passwordHash;

    @Column(name = "created_at")
    private LocalDateTime createdAt = LocalDateTime.now();
}
```

2. Кнопка profile-button в Navbar.js

```
<button className="profile-button" onClick={()=>{
    if(!user)onProfileClick();
    else navigate("/profile");
}}>
    {user? (
        <span className="user-name" > {user.lastName}
        {user.firstName}
        </span>
    ) : (
```

```

        <img src={profileIcon} alt=" Profile"/>
      </button>
    </div>
  )}
</button>

```

3. Логіка відкриття LoginRegisterWindow.js в MainLayout.js

```

<Navbar onProfileClick={()=>setLoginOpen(true)} />
{isLoginOpen && (
  <Modal isOpen={isLoginOpen} onClose={() =>
    setLoginOpen(false)}>
    <LoginRegisterWindow onClose={() =>
      setLoginOpen(false)} />
  </Modal>
)}

```

4. Функція handleChange

```

const handleChange = (e) => {
  const {name, value} = e.target;
  if (step === "register") {
    setFormData((prev) => ({... prev, [name]:
      value }));
  } else if (step === "login") {
    setLoginData((prev) => ({... prev, [name]:
      value }));
  }
};

```

5. Реалізація кнопок LoginRegisterWindow.js у HTML

```

<div className="login-options">
  <h2 className="login title">
    {t( 'loginRegisterWindow.options.h2')}
  </h2>
  <button className="styled-button login-button"
    onClick={() => setStep("login")}>
    {t('loginRegisterWindow.options.b_1')}
  </button>
  <button className="styled-button register-button"
    onClick={() => setStep("register")}>
    {t('loginRegisterWindow.options. b_2')}
  </button>
</div>

```

6. Логіка перемикавання стану LoginRegisterWindow.js

```
{step === "login" && (
<form className="register-form" onSubmit={handleLogin}>
  <h2 className="login-title">
    {t('loginRegisterWindow.login.h2')}
  </h2>

  <input className="modal-input" type="email" name="email"
    placeholder={t('loginRegisterWindow.login.in_1')}
    value={loginData.email} onChange={handleChange} required/>

  <div className="password-wrapper">
    <input className="modal-input" type={showPassword ?
      "text" : "password"} name="password"
      placeholder={t('loginRegisterWindow.login.in_2')}
      value={loginData.password} onChange={handleChange}
      required/>
    <button type="button" className="toggle-password"
      onClick={showPassword ? ":" : ""}
    </button>
  </div>
</form>
)}

{step === "register" && (
<form className="register-form" onSubmit={handleRegister}>
  <h2 className="login
    title">{t('loginRegisterWindow.register.h2')}
  </h2>

  <input className="modal-input" type="text" name="lastName"
    placeholder={t('loginRegisterWindow.register.in_1')}
    value={formData.lastName} onChange={handleChange} required
    />

  <input className="modal-input" type="text" name="firstName"
    placeholder={t('loginRegisterWindow.register.in_2')}
    value={formData.firstName} onChange={handleChange}
    required/>

  <input className="modal-input" type="email" name="email"
    placeholder={t('loginRegisterWindow.register.in_3')}
    value={formData.email} onChange={handleChange} required/>

  <div className="password-wrapper">
    <input className="modal-input" type={showPassword ?
      "text" : "password"} name="password"
```

```

placeholder={t('loginRegisterWindow.register.in_4')}}
value={formData.password} onChange={handleChange}
required/>

        <button type="button" className="toggle-password"
onClick "O_O" : "X_X">
        </button>
    </div>
</form>
)}

```

ДОДАТОК Б

1. Функція відправки даних handleRegister.

```

const handleRegister = async(e) => {
    e.preventDefault();
    setRegisterError("");
    setIsLoading(true);

    try {
        const response = await
        axios.post("http://localhost:8080/api/auth/register",
            formData, {
                withCredentials: true
            });
        localStorage.setItem('user',
            JSON.stringify(response.data));
        window.dispatchEvent(new Event('storage'));
        onClose();
        navigate("/profile");
    } catch (error) {
        console.error(" Помилка реєстрації : ", error);
        setRegisterError(error.response?. data?.message
        || t( 'loginRegisterWindow.error_2'));
    } finally {
        setIsLoading(false);
    }
};

```

2. AuthController реєстрація

```

@RestController
@RequestMapping("/api/auth")
@CrossOrigin(origins = "http://localhost:3000", allowCredentials =
"true")
public class AuthController {
    private final UserService userService;

```

```

public AuthController(UserService userService) {
    this.userService = userService;
}

@PostMapping("/register")
public ResponseEntity<UserResponse> register(@RequestBody
RegisterRequest request) {
    UserResponse user = userService.register(request);
    authenticateUser(request.getEmail(),
request.getPassword());
    return ResponseEntity.ok(user);
}
}

```

3. UserService реєстрація

```

public class UserService implements UserDetailsService {
    private final UserRepository userRepository;
    private final PasswordEncoder passwordEncoder;

    public UserService(UserRepository userRepository,
        PasswordEncoder passwordEncoder) {
        this.userRepository = userRepository; this.passwordEncoder
        = passwordEncoder; }

    public UserResponse register(RegisterRequest request) {
        if(userRepository.findByEmail
(request.email).isPresent())
        { throw new
            ResponseStatusException(HttpStatus.BAD_REQUEST,
            "Email вже зареєстрований ");
        }
        User user = new User();
        user.setFirstName(request.firstName);
        user.setLastName(request.lastName);
        user.setEmail(request.email);
        user.setPasswordHash(passwordEncoder.encode(
request.password));

        User savedUser = userRepository.save(user);
        return new UserResponse(savedUser.getId(),
        savedUser.getLastName(), savedUser.getFirstNam e(),
        savedUser.getEmail());
    }
}

```

4. Функція виправки даних hanldeLogin


```

const handleLogin = async(e) => {
  try {
    const response = await
    axios.post("http://localhost:8080/api/auth/login",
    loginData, {
      withCredentials: true
    });
  } catch (error) {...} finally {setIsLoading(false);}
};

```

5. AuthController вхід

```

@RestController
@RequestMapping( "/api/auth")
@CrossOrigin(origins = "http://localhost:3000", allowCredentials =
"true")
public class AuthController {
  private final UserService userService;
  private final AuthenticationManager authenticationManager;

  public AuthController(UserService userService,
    AuthenticationManager authenticationManager) {
    this.userService = userService;
    this.authenticationManager = authenticationManager;
  }

  @PostMapping("/login")
  public ResponseEntity<UserResponse> login(@RequestBody
  LoginRequest request) {
    UserResponse user = userService.login(request);
    authenticateUser(request.getEmail(),
      request.getPassword());
    return ResponseEntity.ok(user);
  }

  private void authenticateUser(String email, String password) {
    Authentication authentication =
    authenticationManager.authenticate( new
    UsernamePasswordAuthenticationToken(email, password)
    );
    SecurityContextHolder.getContext().setAuthentication(authentication);
  }
}

```

6. UserService вхід

```
public class UserService implements UserDetailsService {
    private final UserRepository userRepository;
    private final PasswordEncoder passwordEncoder;

    public UserService(UserRepository userRepository,
        PasswordEncoder passwordEncoder) {
        this.userRepository = userRepository;
        this.passwordEncoder = passwordEncoder;
    }

    public UserResponse login(LoginRequest request) {
        User user = userRepository.findByEmail(
            request.email) .orElseThrow(() -> new
            RuntimeException("Неправильна пошта або пароль"));
        if(!passwordEncoder.matches(request.password,
            user.getPasswordHash()))
        { throw new RuntimeException("Неправильна пошта або
            пароль");
        }
        return new UserResponse(user.getId(), user.getLastName(),
            user.getFirstName(), user.getEmail());
    }
}
```

ДОДАТОК В

1. Кнопка переходу на сторінку з квитками в Navbar.js

```
<button className="ticket-button" onClick={() => { onClosePanel();
navigate("/tickets");}}>
```

2. Рендер <Outlet>

```
<main className="content-container">
    <Outlet context = {{ openTourModal }}/>
</main>
```

3. Пошукова панель в TicketsPage.js

```
<form className="search-bar" onSubmit={handleSearch}>
    <div className="country-selectors">
```

```

<div className="country-select">
  <label>{t('ticketsPage.l_1')}

```

4. Формування списку країн

```
const countries = [
```

```

t('ticketsPage.countries.c_1'),
t('ticketsPage.countries.c_2'),
t('ticketsPage.countries.c_3'),
t('ticketsPage.countries.c_4'),
t('ticketsPage.countries.c_5'),
t('ticketsPage.countries.c_6'),
t('ticketsPage.countries.c_7'),
t('ticketsPage.countries.c_8'),
t('ticketsPage.countries.c_9'),
t('ticketsPage.countries.c_10')
];

```

5. Генерація ключів для країн в TicketsPage.js

```

const generateTicketKey = (ticket, index) => {
  if (ticket.id) return ticket.id;
  const baseKey = `${ticket.fromCountry}- ${ticket.toCountry}-
${ticket.departureTime}`;
  const additionalInfo = `${ticket.fromAirport}-
${ticket.toAirport}-${ticket.departureDays}`;
  return `${baseKey}-${additionalInfo}-${index}`;
};

```

6. Реалізація HTML відображення результату пошуку

```

<div className={`tickets-results ${animateVisible ? "fade-in-
bottom" : ""}`} ref={resultsRef}>
  {loading ? (
    <div className="loading-container">
      <div className="loading-spinner">
        <svg className="spinner-icon" viewBox="0 0 50 50">
          <circle className="spinner-path" cx="25"
cy="25" r="20" fill="none" strokeWidth="5"/>
        </svg>
      </div>
      <p className="loading-text">{t('ticketsPage.p_1')}

```

```

        <span
className="to">{ticket.toCountry}</span>
      </h3>
      <div className="ticket-
price">{ticket.price} €</div>
    </div>
    <div className="ticket-details">
      <div className="detail-item">
        <span className="detail-
label">{t('ticketsPage.s_3')}</span>
        <span className="detail-
value">{ticket.fromAirport || t('ticketsPage.s_non')}</span>
      </div>
      <div className="detail-item">
        <span className="detail-
label">{t('ticketsPage.s_4')}</span>
        <span className="detail-
value">{ticket.toAirport || t('ticketsPage.s_non')}</span>
      </div>
      <div className="detail-item">
        <span className="detail-
label">{t('ticketsPage.s_5')}</span>
        <span className="detail-
value">{ticket.departureDays || t('ticketsPage.s_non')}</span>
      </div>
      <div className="detail-item">
        <span className="detail-
label">{t('ticketsPage.s_6')}</span>
        <span className="detail-
value">{ticket.departureTime || t('ticketsPage.s_non')}</span>
      </div>
    </div>
    <button
      className="buy-button"
      onClick={() => openModal(ticket)}
      disabled={loading}
    >
      <span>{t('ticketsPage.s_7')}</span>
    </button>
  </div>
  )))
</div>
) : (
  <div className="no-tickets">
    <h3 className="no-tickets-title">
      {fromCountry
        ? t('ticketsPage.h3.p_1')
        : t('ticketsPage.h3.p_2')}
    </h3>
  </div>
)

```

```
    })
  </div>

```

7. Виклик вікна OrderModal.js та передача даних через TicketsPage.js

```
return ( <div className="tickets-page">
  <div className={`search-panel ${animateVisible ? "fade-in-top" : ""}`}>...</div>
  <div className={`tickets-results ${animateVisible ? "fade-in-bottom" : ""}`>
    refn closeModal={closeModal}
    selectedClass={selectedClass}
    setSelectedClass={setSelectedClass}
    cardNumber={cardNumber}
    setCardNumber={setCardNumber}
    expiryDate={expiryDate}
    setExpiryD{ setCvv={setCvv}
    handleConfirmPayment
    calculateFinalPrice={calculateFinalPrice}
    userId={userId} /> )}
  </div>
);
```

8. Вибір modalStage в OrderModal.js

```
modalStage === "select" ? (
  <>
    <div className="modal-header">
      <h2>{t('orderModal.h2_1')}</h2>
      <p className="modal-subtitle">{t('orderModal.p_1')}</p>
    </div>

    <div className="ticket-summary">
      <div className="summary-item">
        <span
className="summarylabel">{t('orderModal.s_1')}</span>
        <span className="summary-value">
          {selectedTicket.fromCountry} →
          {selectedTicket.toCountry}
        </span>
      </div>
      <div className="summary-item">
        <span className="summary-label">{t('orderModal.s_2')}</span>

```

```

        <span className="summary-value">
          {selectedTicket.fromAirport} →
{selectedTicket.toAirport}
        </span>
      </div>
      <div className="summary-item">
        <span className="summary-
label">{t('orderModal.s_3')}</span>
        <span className="summary-
value">{selectedTicket.departureDays}</span>
      </div>
      <div className="summary-item">
        <span className="summary-
label">{t('orderModal.s_4')}</span>
        <span className="summary-
value">{selectedTicket.departureTime}</span>
      </div>
      <div className="summary-item highlight">
        <span className="summary-
label">{t('orderModal.s_5')}</span>
        <span className="summary-
value">{selectedTicket.price} €</span>
      </div>
    </div>

    <div className="modal-options">
      <div className="option-wrapper">
        <button
          className="option-button economy"
          onClick={() => handleClassSelect("economy")}
        >
          <div className="option-icon">
            <img src={airplaneIcon} alt="Економ" />
          </div>
          <div className="option-text">
            <span className="option-
title">{t('orderModal.s_6')}</span>
            <span className="option-
desc">{t('orderModal.s_7')}</span>
          </div>
        </button>
        <div className="option-price">+0%</div>
      </div>
      <div className="option-wrapper">
        <button
          className="option-button business"
          onClick={() => handleClassSelect("business")}
        >
          <div className="option-icon">
            <img src={airplaneIcon} alt="Бізнес" />
          </div>

```

```

        <div className="option-text">
            <span className="option-
title">{t('orderModal.s_8')}

```

9. Функція handleClassSelect

```

const handleClassSelect = (classType) => {
    setSelectedClass (classType);
    setIsFlying (true);
    setTimeout (() => {
        setModalStage ("confirmed");
        setIsFlying (false);
    }, 1000);
};

```

10. Розрахунок додаткових послуг до суми ціни квитка функцією CalculateFinalPrice

```

const calculateFinalPrice = () => {
    if (!selectedTicket) return "0 €";
    const basePrice = parseFloat(selectedTicket.price);
    let finalPrice=basePrice; if (selectedClass === "business")
{ finalPrice += basePrice * 0.5; }
    return ` $ ${finalPrice.toFixed(2)} € `;
};

```

11. Створення вікна фінального замовлення

```

(
    <div className="confirmed-content">
        <div className="modal-header">
            <h2>{t('orderModal.h2_2')}

```



```

<div className="ticket-details">
  <div className="detail-card">
    <div className="detail-row">
      <span className="detail-label">{t('orderModal.s_1')}</span>
      <span className="detail-value">
        {selectedTicket.fromCountry} →
        {selectedTicket.toCountry}
      </span>
    </div>
    <div className="detail-row">
      <span className="detail-label">{t('orderModal.s_2')}</span>
      <span className="detail-value">
        {selectedTicket.fromAirport} →
        {selectedTicket.toAirport}
      </span>
    </div>
    <div className="detail-row">
      <span className="detail-label">{t('orderModal.s_3')}</span>
      <span className="detail-value">{selectedTicket.departureDays}
      </span>
    </div>
    <div className="detail-row">
      <span className="detail-label">{t('orderModal.s_4')}
      </span>
      <span className="detail-value">{selectedTicket.departureTime}
      </span>
    </div>
    <div className="detail-row">
      <span className="detail-label">{t('orderModal.s_11')}
      </span>
      <span className="detail-value highlight">
        {selectedClass === "business" ?
          t('orderModal.s_12.p_1') :
          t('orderModal.s_12.p_2')}
      </span>
    </div>
    <div className="detail-row total">
      <span className="detail-label">{t('orderModal.s_13')}
      </span>
      <span className="detail-value price">{calculateFinalPrice()}
      </span>
    </div>
  </div>
</div>

```

```

        </div>
      </div>
    </div>

    {loadingCard ? (
      <div>{t('orderModal.loadCard.d_1')}

```

ДОДАТОК Г

1. Функція fetchTickets для отримання квитків з серверу

```
const fetchTickets = async () => {
  setLoading(true);
  try {
    const params = new URLSearchParams();
    params.append('fromCountry', fromCountry);
    if (toCountry) params.append('toCountry', toCountry);
    if (minPrice) params.append('minPrice', minPrice);
    if (maxPrice) params.append('maxPrice', maxPrice);

    const response = await
    axios.get(`http://localhost:8080/api/tickets?${params.toString()}`
    );
    setTickets(Array.isArray(response.data) ? response.data :
    []);
  } catch (error) {
    console.error("Помилка при отриманні квитків:", error);
    setTickets([]);
  } finally {
    setLoading(false);
    scrollToResults();
  }
};
```

2. Бібліотека React 'useEffect' для оновлення в реальному часі

```
useEffect( () => {
  if (fromCountry) { fetchTickets(); }
}, [fromCountry, toCountry, minPrice, maxPrice]);
```

3. TicketController пошук квитків

```
import myapp.dto.TicketSearchDTO;
import myapp.model.Ticket;
import myapp.service.TicketService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/api/tickets")
public class TicketController {
```

```

private final TicketService ticketService;

@Autowired public TicketController(TicketService
ticketService) { this.ticketService = ticketService; }

@GetMapping public List<Ticket> searchTickets(
@RequestParam(required = false) String fromCountry ,
@RequestParam(required = false) String toCountry ,
@RequestParam(required = false) Double minPrice,
TicketSearchDTO()); searchDTO.setToCountry(toCountry);
searchDTO.setMinPrice(minPrice);
}

```

4. TicketService пошук квитків

```

import myapp.dto.TicketSearchDTO;
import myapp.model.Ticket;
import myapp.repository.TicketRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;

@Service public class TicketService {

    private final TicketRepository ticketRepository;

    @Autowired public TicketService(TicketRepository
ticketRepository) {
        this.ticketRepository = ticketRepository;
    }

    public List<Ticket> searchTickets(TicketSearchDTO
searchDTO) {
        return ticketRepository.searchTickets(
searchDTO.getFromCountry(), searchDTO.getToCountry(),
searchDTO.getMinPrice(), searchDTO.getMaxPrice() );
    }
}

```

5. TicketRepository пошук квитків

```

import myapp.model.Ticket;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;
import org.springframework.stereotype.Repository;

```

```

import java.util.List;

@Repository
public interface TicketRepository extends JpaRepository<Ticket,
Long> {

    @Query("SELECT t FROM Ticket t WHERE " +
        "(:fromCountry IS NULL OR t.fromCountry = " +
        "    :fromCountry) AND " +
        "(:toCountry IS NULL OR t.toCountry = :toCountry) " +
        "AND " +
        "(:minPrice IS NULL OR t.price >= :minPrice) AND " +
        "    +
        "(:maxPrice IS NULL OR t.price <= :maxPrice)")

    List<Ticket> searchTickets(
        @Param("fromCountry") String fromCountry,
        @Param("toCountry") String toCountry,
        @Param("minPrice") Double minPrice,
        @Param("maxPrice") Double maxPrice);
}

```

6. Ticket

```

import jakarta.persistence.*;
import lombok.Data;

@Entity
@Table(name = "tickets")
@Data
public class Ticket {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name = "from_country", nullable = false)
    private String fromCountry;

    @Column(name = "from_airport", nullable = false)
    private String fromAirport;

    @Column(name = "to_country", nullable = false)
    private String toCountry;

    @Column(name = "to_airport", nullable = false)
    private String toAirport;

    @Column(name = "departure_days", nullable = false)

```

```

private String departureDays;

@Column(name = "departure_time", nullable = false)
private String departureTime;

@Column(nullable = false)
private Double price;
}

```

ДОДАТОК Д

1. Функція handleConfirmPayment в TicketPage.js

```

const handleConfirmPayment = async (saveCard = false) => {
  try {
    if (!selectedTicket || !selectedClass) {
      throw new Error(t('ticketsPage.error_1'));
    }

    const cleanedCardNumber = cardNumber
      ? cardNumber.replace(/\D/g, "")
      : userCard?.cardNumber || "";
    const cleanedExpiryDate = expiryDate
      ? expiryDate.replace(/\D/g, "")
      : userCard?.expiryDate || "";
    const cleanedCVV = cvv || userCard?.cvv || "";

    if (!cleanedCardNumber || !cleanedExpiryDate ||
!cleanedCVV) {
      throw new Error(t('ticketsPage.error_2'));
    }

    const orderData = {
      ticketId: selectedTicket.id,
      flightClass: selectedClass,
      departureDate: new Date().toISOString(),
      finalPrice: parseFloat(selectedTicket.price) *
(selectedClass === "business" ? 1.5 : 1),
      cardNumber: cleanedCardNumber,
      expiryDate: cleanedExpiryDate,
      cvv: cleanedCVV
    };

    const orderResponse = await axios.post(
      `http://localhost:8080/api/orders/${userId}`,
      orderData,
      { withCredentials: true }
    );
  }
}

```

```

        alert(t('ticketsPage.a_1'));
        setSelectedTicket(null);
    } catch (error) {
        console.error("Помилка при створенні замовлення",
error.response?.data?.message || error.message);
        alert(t('ticketsPage.a_2'));
    }
};

```

2. OrderController створення замовлення

```

import myapp.dto.OrderDTO;
import myapp.model.Order;
import myapp.service.OrderService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.

import java.util.List;

@RestController
@RequestMapping("/api/orders")
@CrossOrigin(origins = "http://localhost:3000", allowCredentials =
"true")
public class OrderController {

    private final OrderService orderService;

    @Autowired public OrderController(OrderService
orderService) {
        this.orderService = orderService;
    }

    @PostMapping("/{userId}")
    public ResponseEntity<?> createOrder(
        @PathVariable("userId") Long userId,
        @RequestBody OrderDTO orderDTO) {

        if (orderDTO == null) {
            return ResponseEntity.badRequest().body("Order
data is required");
        }

        if (orderDTO.getTicketId() == null) {
            return ResponseEntity.badRequest().body("Ticket
ID is required");
        }

        if (orderDTO.getFlightClass() == null ||

```

```

        orderDTO.getFlightClass().isEmpty()) {
            return ResponseEntity.badRequest().body("Flight
class is required");
        }

        if (orderDTO.getDepartureDate() == null) {
            return
                ResponseEntity.badRequest().body("Departure
date is required");
        }

        if (orderDTO.getFinalPrice() == null ||
orderDTO.getFinalPrice() <= 0) {
            return
                ResponseEntity.badRequest().body("Invalid final
price");
        }

        try {
            Order order = orderService.createOrder(userId,
orderDTO);
            return ResponseEntity.ok(order);
        } catch (RuntimeException e) {
            return ResponseEntity.badRequest().body("Error
creating order: " + e.getMessage());
        }
    }
}

```

3. OrderService створення замовлення

```

import myapp.dto.OrderDTO;
import myapp.model.*;
import myapp.repository.CreditCardRepository;
import myapp.repository.OrderRepository;
import myapp.repository.TicketRepository;
import myapp.repository.UserRepository;
import jakarta.transaction.Transactional;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.time.LocalDateTime;
import java.util.List;
import java.util.Optional;

@Service
public class OrderService {

    private final OrderRepository orderRepository;

```



```

private final TicketRepository ticketRepository;
private final UserRepository userRepository;
private final CreditCardService creditCardService;
private final CreditCardRepository creditCardRepository;

@Autowired
public OrderService(OrderRepository orderRepository,
                    TicketRepository ticketRepository,
                    UserRepository userRepository,
                    CreditCardService creditCardService,
                    CreditCardRepository
creditCardRepository) {
    this.orderRepository = orderRepository;
    this.ticketRepository = ticketRepository;
    this.userRepository = userRepository;
    this.creditCardService = creditCardService;
    this.creditCardRepository = creditCardRepository;
}

@Transactional
public Order createOrder(Long userId, OrderDTO orderDTO) {
    User user = userRepository.findById(userId)
        .orElseThrow(() -> new
            RuntimeException("Користувача не знайдено"));

    Ticket ticket =
        ticketRepository.findById(orderDTO.getTicketId())
            .orElseThrow(() -> new
                RuntimeException("Квиток не знайдено"));

    if (orderDTO.getCardNumber() == null ||
orderDTO.getCardNumber().isEmpty() ||
        orderDTO.getExpiryDate() == null ||
orderDTO.getExpiryDate().isEmpty() ||
        orderDTO.getCvv() == null ||
orderDTO.getCvv().isEmpty())
    {
        throw new RuntimeException("Необхідно вказати
дані картки");
    }

    Optional<CreditCard> existingCard =
        creditCardRepository
            .findByCardNumberAndExpiryDateAndCvv(
                orderDTO.getCardNumber(),
                orderDTO.getExpiryDate(),
                orderDTO.getCvv());

    CreditCard card;
    if (existingCard.isPresent()) {
        card = existingCard.get();
    }
}

```

```

        if (card.getBalance() < orderDTO.getFinalPrice())
        {
            throw new RuntimeException("Недостатньо коштів на картці");
        }
        creditCardService.deductFromBalance(card.getId(),
orderDTO.getFinalPrice());
    } else {
        throw new RuntimeException("Картку не знайдено");
    }

    Order order = new Order();
    order.setUser(user);
    order.setTicket(ticket);
    order.setOrderDate(LocalDateTime.now());
    order.setDepartureDate(orderDTO.getDepartureDate());
    order.setFlightClass(orderDTO.getFlightClass());
    order.setFinalPrice(orderDTO.getFinalPrice());
    order.setStatus("CONFIRMED");
    order.setCardNumber(card.getCardNumber());

    return orderRepository.save(order);
}
}

```

4. OrderRepository створення замовлення

```

import myapp.model.Order;
import myapp.model.User;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import java.util.List;

@Repository
public interface OrderRepository extends
JpaRepository<Order, Long> {
    List<Order> findByUser(User user);
}

```

5. Order

```

import jakarta.persistence.*;
import lombok.Data;

import java.time.LocalDateTime;
import java.util.Date;

```

```

@Entity
@Table(name = "orders")
@Data
public class Order {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @ManyToOne
    @JoinColumn(name = "user_id", nullable = false)
    private User user;

    @ManyToOne @JoinColumn(name = "ticket_id", nullable =
false)
    private Ticket ticket;

    @Column(name = "order_date")
    private LocalDateTime orderDate;

    @Column(name = "flight_class", nullable = false)
    private String flightClass;

    @Column(name = "final_price", nullable = false)
    private Double finalPrice;

    @Column(nullable = false)
    private String status;

    @Column(name = "card_number")
    private String cardNumber;
}

```

ДОДАТОК Е

1. Метод зміни мови LanguageSelector.js

```

import React, { useState, useRef, useEffect } from "react";
import { useTranslation } from 'react-i18next';
import "../LanguageSelector.css";
import languageIcon from "../img/language-icon.png";

const LanguageSelector = () => {
    const { i18n } = useTranslation();
    const [isOpen, setIsOpen] = useState(false);
    const selectorRef = useRef(null);

    const toggleDropdown = () => setIsOpen(prev => !prev);

    useEffect(() => {

```

```

const handleClickOutside = (event) => {
  if (selectorRef.current &&
    !selectorRef.current.contains(event.target)) {
    setIsOpen(false);
  }
};

document.addEventListener("mousedown",
  handleClickOutside);
return () => {
  document.removeEventListener("mousedown",
    handleClickOutside);
};
}, []);

const changeLanguage = (lng) => {
  i18n.changeLanguage(lng);
  setIsOpen(false);
};

return (
  <div className="language-selector" ref={selectorRef}>
    <button className="language-button"
      onClick={toggleDropdown}>
      <img src={languageIcon} alt="Мова" />
    </button>
    {isOpen && (
      <div className="language-dropdown">
        <button onClick={() =>
changeLanguage('uk')}>Українська</button>
        <button onClick={() =>
changeLanguage('en')}>English</button>
        <button onClick={() =>
changeLanguage('fr')}>Français</button>
        <button onClick={() =>
changeLanguage('es')}>Español</button>
        <button onClick={() =>
changeLanguage('it')}>Italiano</button>
        <button onClick={() =>
changeLanguage('de')}>Deutsch</button>
        <button onClick={() =>
changeLanguage('pl')}>Polski</button>
        <button onClick={() =>
changeLanguage('be')}>Беларуская</button>
        <button onClick={() =>
changeLanguage('ro')}>Română</button>
        <button onClick={() =>
changeLanguage('enUS')}>English (US)</button>
      </div>
    )}
  </div>

```

```

    );
};

export default LanguageSelector;

```

2. Конфігурація зміни мови i18n.js

```

import i18n from 'i18next';
import { initReactI18next } from 'react-i18next';
import LanguageDetector from 'i18next-browser-languagedetector';
import enTranslation from './locales/en/translation.json';
import ukTranslation from './locales/uk/translation.json';
import frTranslation from './locales/fr/translation.json';
import esTranslation from './locales/es/translation.json';
import itTranslation from './locales/it/translation.json';
import deTranslation from './locales/de/translation.json';
import plTranslation from './locales/pl/translation.json';
import beTranslation from './locales/be/translation.json';
import roTranslation from './locales/ro/translation.json';
import enUsTranslation from './locales/en-US/translation.json';

```

```

i18n
  .use(LanguageDetector)
  .use(initReactI18next)
  .init({
    resources: {
      en: {
        translation: enTranslation
      },
      uk: {
        translation: ukTranslation
      },
      fr: {
        translation: frTranslation
      },
      es: {
        translation: esTranslation
      },
      it: {
        translation: itTranslation
      },
      de: {
        translation: deTranslation
      },
      pl: {
        translation: plTranslation
      },
      be: {
        translation: beTranslation
      },

```

```

    },
    ro: {
      translation: roTranslation
    },
    enUS: {
      translation: enUsTranslation
    }
  },
  fallbackLng: 'uk',
  debug: true,
  interpolation: {
    escapeValue: false,
  }
});

export default i18n;

```

3. Ключі локалізації JSON

```

{
  "mainLayout":{
    "copyright": "© 2025 FashaSkyLine. Усі права захищено.",
    "modal":{
      "h2": "Деталі туру",
      "p": "Ось тут можна додати більше інформації про тур"
    }
  },
  "homePanel":{
    "menu": "Меню",
    "home": "Головна",
    "tour": "Наші тури",
    "our": "Про нас",
    "contact": "Контакти"
  }
}

```

ДОДАТОК Є

1. Постійне оновлення функції fetchUserData в useEffect.

```

useEffect(() => {
  const fetchUserData = async () => {
    try {
      const storedUser = localStorage.getItem("user");

      if (!storedUser) {
        console.error("User not found in
          localStorage");
      }
    }
  }
});

```

```

        navigate("/");
        return;
    }

    const user = JSON.parse(storedUser);
    setUserId(user.id);
    setFirstName(user.firstName);
    setLastName(user.lastName);
    setEmail(user.email);

    try {
        const cardResponse = await
        axios.get(`http://localhost:8080/api/cards/${user.id}`, {
            withCredentials: true
        });
        setCard(cardResponse.data);
    } catch (cardError) {
        console.error("Error fetching card:",
        cardError);
        setCard(null);
    }

    } catch (error) {
        console.error("Unexpected error:", error);
        navigate("/");
    }
};

fetchUserData();
}, [navigate]);

```

2. CreditCardController пошук платіжної картки користувача

```

@RestController
@RequestMapping("/api/cards")
@CrossOrigin(origins = "http://localhost:3000", allowCredentials =
"true")
public class CreditCardController {

    private final CreditCardService service;

    public CreditCardController(CreditCardService service) {
        this.service = service;
    }

    @GetMapping("/{userId}")
    public ResponseEntity<?> getCard(@PathVariable Long
    userId) {
        try {

```

```

        return service.getCardById(userId)
            .map(ResponseEntity::ok)
            .orElse(ResponseEntity.internalServerError().body("
                Помилка сервера "));
    }
}

```

3. CreditCardService пошук платіжної картки користувача

```

@Service
public class CreditCardService {
    private final CreditCardRepository repository;

    public CreditCardService(CreditCardRepository repository) {
        this.repository = repository;
    }

    public Optional<CreditCard> getCardById(Long userId) {
        return repository.findById(userId);
    }
}

```

4. CreditCardRepository пошук платіжної картки користувача

```

import myapp.model.CreditCard;
import org.springframework.data.jpa.repository.JpaRepository;

import java.util.Optional;

public interface CreditCardRepository extends
    JpaRepository<CreditCard, Long> {
    Optional<CreditCard> findByCardNumberAndExpiryDateAndCvv(
        String cardNumber, String expiryDate, String cvv);
}

```

5. Безпечний вивід певних даних платіжної картки в ProfilePage.js

```

const maskCardNumber = (number) => {
    if (!number) return "";
    const cleaned =
        number.replace(/\D/g, "");
    return ".... .... ...." +
        cleaned.slice(-4);
};

return (
    <div className="profile-page">

```



```

        <div className={`profile-card ${isVisible ? "visible"
" : ""}`}...>

        <div className={`profile-card ${isVisible ? "visible"
" : ""}`}...>
{card ? (
    <div className={`profile-card ${isCardVisible ?
"visible" : ""}`}...>
{cardLoading ? (
    <div className="loading-spinner-profile">
        <svg className="spinner-icon-profile" viewBox="0 0
50 50">
            <circle className="spinner-path-profile"
cx="25" cy="25" r="20" fill="none"
strokeWidth="5"/>
        </svg>
    </div>
) : (
    <>
        <p><strong>{t('profilePage.card.p_1')}

```

6. HTML-структура додавання платіжної картки

```

return (
    <div className="profile-page">
        <div className={`profile-card ${isVisible ? "visible"
" : ""}`}...>

        <div className={`profile-card ${isVisible ? "visible"
" : "" }`}...>

```

```

{card ? (
  <div className={`profile-card ${isCardVisible ?
    "visible" : ""}`}>
) : showCardForm ? (
  <div className={`profile-card ${isCardVisible ?
    "visible" : ""}`}>
    <div className="input-group">
      <label>{t('profilePage.card.p_1')}</label>
      <input
        type="text"
        value={formatCardNumber(cardNumber)}
        onChange={(e) =>
          setCardNumber(e.target.value)}
        placeholder="1234 5678 9012 3456"
        maxLength={19}
      />
    </div>
    <div className="input-group">
      <label>{t('profilePage.card.l_1')}</label>
      <input
        type="text"
        value={formatExpiryDate(expiryDate)}
        onChange={(e) =>
          setExpiryDate(e.target.value)}
        placeholder="MM/YY"
        maxLength={5}
      />
    </div>
    <div className="input-group">
      <label>CVV:</label>
      <input
        type="text"
        value={cvv}
        onChange={(e) =>
          setCvv(e.target.value.replace(/\D/g, "").slice(0, 3))}
        placeholder="123"
        maxLength={3}
      />
    </div>
    {cardError && <p className="error-
message">{cardError}</p>}
    <button
      className="profile-button-card"
      onClick={handleSaveCard}
    >
      {t('profilePage.card.b_2')}
    </button>
    <button
      className="profile-button-card"
      onClick={() => {

```

```

        setIsCardVisible(false);
        setTimeout(() =>
setShowCardForm(false), 300);
        })
      >
        {t('profilePage.card.b_3')}
      </button>
    </div>
  ) : null}
</div>
);

```

7. Функція `handleSavedCard` для надсилання та зберігання даних які вводить користувач при додаванні платіжної картки в `ProfilePage.js`

```

const handleSaveCard = async () => {
  setCardLoading(true);
  setCardError("");

  try {
    const cleanedCardNumber = cardNumber.replace(/\D/g,
    "");
    const cleanedExpiryDate = expiryDate.replace(/\D/g,
    "");

    if (!cleanedCardNumber || !cleanedExpiryDate || !cvv){
      throw new Error(t('profilePage.error_1'));
    }

    const response = await
    axios.post("http://localhost:8080/api/cards/add", {
      userId,
      cardNumber: cleanedCardNumber,
      expiryDate: cleanedExpiryDate,
      cvv
    }, {
      withCredentials: true,
      headers: {
        'Content-Type': 'application/json'
      }
    });

    setCard(response.data);
    setShowCardForm(false);
    setCardNumber("");
    setExpiryDate("");
    setCvv("");
  } catch (error) {
    console.error("Помилка при збереженні картки:",

```

```

        error);
    setCardError(
        error.response?.data?.error ||
        error.message ||
        t('profilePage.error_2')
    );
} finally {
    setCardLoading(false);
}
};

```

8. CreditCardController додання та зберігання платіжної картки

```

@RestController
@RequestMapping("/api/cards")
@CrossOrigin(origins = "http://localhost:3000", allowCredentials =
"true")
public class CreditCardController {

    private final CreditCardService service;

    public CreditCardController(CreditCardService service) {
        this.service = service;
    }

    @PostMapping("/add")
    public ResponseEntity<?> addCard(@RequestBody
CreditCardRequest request) {
        try {
            CreditCard card = service.createCard(request);
            return ResponseEntity.ok(card);
        } catch (IllegalArgumentException e) {
            return
                ResponseEntity.badRequest().body(Map.of("error",
e.getMessage(), "timestamp", LocalDateTime.now() ));
        } catch (Exception e) {
            e.printStackTrace();
            return
                ResponseEntity.internalServerError().body(Map.of(
"error", " Внутрішня помилка сервера : " +
e.getMessage(), "timestamp", LocalDateTime.now()));
        }
    }
}

```

9. CreditCardService додання та зберігання платіжної картки

```

import myapp.dto.CreditCardRequest;
import myapp.model.CreditCard;
import myapp.repository.CreditCardRepository;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import java.util.Optional;

@Service
public class CreditCardService {
    private final CreditCardRepository repository;

    public CreditCardService(CreditCardRepository repository) {
        this.repository = repository;
    }

    public CreditCard createCard(CreditCardRequest request) {
        if (repository.existsByUserId(request.getUserId())) {
            throw new IllegalArgumentException("Користувач вже має картку");
        }
        if (repository.existsByCardNumber(request.getCardNumber())) {
            throw new IllegalArgumentException("Картка з таким номером вже існує");
        }

        double randomBalance = Math.floor(Math.random() * (5000 - 500 + 1)) + 500;

        CreditCard card = new CreditCard();
        card.setUserId(request.getUserId());
        card.setCardNumber(request.getCardNumber());
        card.setExpiryDate(request.getExpiryDate());
        card.setCvv(request.getCvv());
        card.setBalance(randomBalance);

        return repository.save(card);
    }
}

```

10. CreditCardRepository додання та зберігання платіжної картки

```

import myapp.model.CreditCard;
import org.springframework.data.jpa.repository.JpaRepository;

import java.util.Optional;

```

```

public interface CreditCardRepository extends
JpaRepository<CreditCard, Long> {
    Optional<CreditCard> findById(Long userId);
    boolean existsByUserId(Long userId);
    boolean existsByCardNumber(String cardNumber);

    Optional<CreditCard>
    findByCardNumberAndExpiryDateAndCvv(String cardNumber,
    String expiryDate, String cvv);
}

```

11. CreditCard

```

import jakarta.persistence.*;
import lombok.*;

@Entity
@Data
@NoArgsConstructor
@AllArgsConstructor
public class CreditCard {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(unique = true, nullable = false)
    private String cardNumber;

    @Column(nullable = false)
    private String expiryDate;

    @Column(nullable = false)
    private String cvv;

    @Column(nullable = false)
    private Double balance;

    @Column(nullable = false, unique = true)
    private Long userId;
}

```