Міністерство освіти і науки
України Університет митної справи та фінансів

Факультет інноваційних технологій
Кафедра комп'ютерних наук та інженерії програмного забезпечення

Кваліфікаційна робота бакалавра

на тему: «Розробка клієнтської частини веб-серверного застосунку для он-лайн купівлі квитків в системі автобусних перевезень»

Виконала: студентка групи К20-2
Спеціальність 122 «Комп'ютерні науки»
Раджабова Саміра Гамлетівна
(прізвище та ініціали)
Керівник к.т.н., доц. Ульяновська Ю.В.
(науковий ступінь, вчене звання, прізвище та ініціали)
Рецензент Університет митної справи та фінансів
доцент кафедри кібербезки та інформаційних
технологій, к.т.н., доц. Савченко Ю.В.
(науковий ступінь, вчене звання, прізвище та ініціали)

Дніпро – 2024

АННОТАЦІЯ

Раджабова С.Г. Розробка клієнтської частини веб-серверного застосунку для он-лайн купівлі квитків в системі автобусних перевезень. Кваліфікаційна робота на здобуття освітнього ступеня бакалавра за спеціальністю 122 «Комп'ютерні науки». – Університет митної справи та фінансів, Дніпро, 2024.

Дана кваліфікаційна робота присвячена розробці клієнтської частини веб-серверного застосунку для он-лайн купівлі квитків в системі автобусних перевезень. Застосунок розроблено на основі сучасного фреймворку Vue3 з використанням мови програмування TypeScript та бібліотеки Pinia для керування станом. У роботі розглянуто технічні аспекти створення клієнтської частини, включаючи інтеграцію з серверною частиною через API, реалізацію функціоналу для пошуку квитків, динамічне змінення даних, фільтрацію, а також збереження даних в локальному сховищі браузера для покращення продуктивності та зручності користувачів. Було проведено тестування та оптимізацію інтерфейсу для зручності користування.

В процесі роботи були застосовані сучасні методології розробки програмного забезпечення, такі як Agile.

Отримані результати мають практичне значення для компаній, що надають послуги автобусних перевезень, оскільки розроблений застосунок дозволяє автоматизувати процес купівлі квитків, зменшити черги та покращити обслуговування клієнтів. Застосунок відрізняється сучасним дизайном, високою продуктивністю та надійністю, що робить його важливим інструментом у сфері електронної комерції. У перспективі можливий розвиток системи шляхом додавання нових функцій, таких як інтеграція з іншими транспортними системами та впровадження штучного інтелекту для аналізу даних і прогнозування попиту.

Ключові слова: Vue3, TypeScript, Pinia, веб-застосунок, он-лайн купівля квитків, автобусні перевезення, API, динамічне змінення даних, фільтрація, локальне сховище браузера, Agile.

Список публікацій:

1. Міністерство освіти і науки України, Університет митної справи та фінансів, Рада молодих вчених Університету митної справи та фінансів. Економіко-правові та соціально-технічні напрями еволюції цифрового суспільства. Матеріали міжнародної науково-практичної конференції, Том 2, 02 червня 2022 р. С. 513. URL: https://biblio.umsf.dp.ua/xmlui/bitstream/handle/123456789/4797/%D0%9A%D0%9E%D0%9D%D0%A4%D0%95%D0%A0%D0%95%D0%9D%D0%A6%D0%86%D0%AF%20%D0%A7%D0%95%D0%A0%D0%92%D0%95%D0%9D%D0%AC%20%D0%A2%D0%9E%D0%9C%202.pdf?sequence=1&isAllowed=y

2. Міністерство освіти і науки України, Університет митної справи та фінансів, Рада молодих вчених Університету митної справи та фінансів. Економіко-правові та управлінсько-технологічні виміри сьогодення: молодіжний погляд. Матеріали міжнародної науково-практичної конференції, Том 3, 03 листопада 2023 р. С. 288. URL: https://drive.google.com/file/d/1JEG4IPGzqAcDAI1kyZ2XxE53f8FV01DQ/view

ABSTRACT

Radzhabova S.H. Development of the Client-Side of a Web-Server Application for Online Ticket Purchasing in a Bus Transportation System. Bachelor's thesis in Computer Science, specialty 122 "Computer Science". – University of Customs and Finance, Dnipro, 2024.

This bachelor's thesis is dedicated to the development of the client-side of a web-server application for online ticket purchasing in a bus transportation system. The application is developed using the modern framework Vue3, with the programming language TypeScript and the state management library Pinia. The thesis examines the technical aspects of creating the client-side, including integration with the server-side via API, implementation of ticket search functionality, dynamic data updating, filtering, and data storage in the local browser storage to improve performance and user convenience. Testing and optimization of the interface for usability were conducted.

Modern software development methodologies, such as Agile, were applied during the development process.

The results obtained are of practical significance for companies providing bus transportation services, as the developed application allows automating the ticket purchasing process, reducing queues, and improving customer service. The application features a modern design, high performance, and reliability, making it an important tool in the field of e-commerce. Future development of the system may include adding new features, such as integration with other transportation systems and the implementation of artificial intelligence for data analysis and demand forecasting.

Keywords: Vue3, TypeScript, Pinia, web application, online ticket purchasing, bus transportation, API, dynamic data updating, filtering, local browser storage, Agile.

List of publications:

1. Ministry of Education and Science of Ukraine, University of Customs and Finance, Council of Young Scientists of the University of Customs and Finance. Economic-legal and socio-technical directions of the evolution of the digital society. Materials of the international scientific and practical conference, Volume 2,

June 02 2022, p. 513. URL: https://biblio.umsf.dp.ua/xmlui/bitstream/handle/123456789/4797/%D0%9A%D0%9E%D0%9D%D0%A4%D0%95%D0%A0%D0%95%D0%9D%D0%A6%D0%86%D0%AF%20%D0%A7%D0%95%D0%A0%D0%92%D0%95%D0%9D%D0%AC%20%D0%A2%D0%9E%D0%9C%202.pdf?sequence=1&isAllowed=y

2. Ministry of Education and Science of Ukraine, University of Customs and Finance, Council of Young Scientists of the University of Customs and Finance. Today's economic-legal and management-technological dimensions: a youth perspective. Materials of the international scientific and practical conference, Volume 3, November 03, 2023, p. 288. URL: https://drive.google.com/file/d/1JEG4IPGzqAcDAI1kyZ2XxE53f8FV01DQ/view

ЗМІСТ

ВСТУП

Актуальність дослідження. У сучасному світі інформаційні та цифрові технології відіграють значну роль у сферах людської діяльності, у тому числі в таких сфері надання транспортних послуг. Актуальним завданням є автоматизація процесу купівлі квитків на автобусний транспорт, оскільки це сприяє підвищенню ефективного обслуговування клієнтів, оптимізації роботи транспортних компаній та зменшенню черг і скупчення людей. Зростання з кожним роком популярності он-лайн сервісів потребує розробки зручних, простих і надійних у використанні веб-застосунків, що дозволять потенційним користувачам безпечно й швидко здійснити купівлі через Інтернет. Завдяки сучасним технологіям розробка веб-застосунків досягла нового рівня, що дозволяє ефективніше залучати користувачів і легко інтегруватися з різними системами, забезпечуючи оптимальне управління ресурсами та інформацією. В епоху глобалізації та розширення пасажирських перевезень надзвичайно важливо віддати пріоритет швидкому та безперешкодному доступу до послуг, які надають транспортні компанії. Мінімізація фізичних контактів і можливість віддаленої купівлі квитків є критично важливими.

Метою даної роботи є автоматизація процесу купівлі квитків користувачами в системі автобусних перевезень.

Методи дослідження. У відповідності до поставленої мети, в кваліфікаційній роботі були використані наступні методи: аналіз сучасних технологій розробки веб-застосунків, проектування та розробка програмного забезпечення з використанням фреймворку Vue3, мови програмування TypeScript та сховища Pinia, інтеграція з серверною частиною через API задля отримання даних та оптимізація інтерфейсу користувача й тестування.

Відповідно до поставленої мети у кваліфікаційній роботі необхідно виконати наступні *завдання дослідження*:

1. Провести аналіз існуючих методів та засобів для розробки клієнтських веб-застосунків.

2. Розробити технічні вимоги до клієнтської частини веб-серверного застосунку для онлайн купівлі квитків.

3. Розробити та реалізувати клієнтську частину програми з використанням новітніх технологій.

4. Провести тестування та оптимізацію користувальницького інтерфейсу задля забезпечення зручного використання.

Об'єкт дослідження - процес автоматизації купівлі квитків в системі автобусних перевезень.

Предмет дослідження - клієнтська частина веб-серверного застосунку для он-лайн купівлі квитків.

Практичне значення отриманих результатів. Розроблений веб-застосунок дає змогу зменшити черги, автоматизувати процес купівлі квитків та покращити обслуговування клієнтів, що суттєво впливає на ефективність роботи транспортних компаній. Висока продуктивність, надійність та простота використання веб-застосунку роблять його інструментом, що дуже важливий у сфері електронної комерції.

Основні технічні характеристики продукту:

1. Створення реактивних та динамічних користувальницьких інтерфейсів, за допомогою використання фреймворку Vue3

2. Забезпечення строгої типізації, що дозволяє зменшити кількість помилок в коді, за допомогою використання мови програмування TypeScript.

3. Ефективне керування даними, отриманими із API, за допомогою використання бібліотеки Pinia.

4. Отримання й обробка даних, які були отримані за допомогою інтеграції із серверною частиною через API.

5. Забезпечення актуальної інформації, за допомогою динамічного оновлення даних та можливості фільтрації пошуку.

6. Покращення зручності й продуктивності користувачів, за допомогою збереження певних даних в локальному сховищі браузера.

7. Тестування інтерфейсу користувача для забезпечення високої зручності використання.

Очікувані технічні та економічні ефекти від реалізації веб-застосунку включає скорочення часу на придбання квитків, підвищення задоволеності клієнтів, оптимізацію роботи перевізників та підвищення їхньої конкурентоспроможності на ринку транспортних послуг.

Структура роботи. Робота складається зі вступу, трьох розділів, висновків, списку використаних джерел та одного додатку. Об’єм роботи 110 сторінок, робота містить 53 рисунка, 0 таблиць.

РОЗДІЛ 1. ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧ ДОСЛІДЖЕННЯ

1.1. Аналіз існуючих он-лайн систем купівлі квитків

Швидкість та зручність сервісу є основними факторами, що впливають на вибір користувача. Це відноситься і до сфери автобусних перевезень, де зручність придбання квитків грає вирішальну роль. Традиційні методи придбання квитків через каси потребують багато часу та фізичної присутності, що є доволі незручним для людей.

Он-лайн системи купівлі квитків дозволяють вирішити ці проблеми, забезпечуючи наступні переваги:

– Доступність 24/7. Пасажири мають змогу придбати квиток у будь-який час, не залежно від графіку роботи кас.

– Економія часу. Відсутність необхідності стояти в чергах.

– Широкий вибір. Можливість порівняти різні рейси та обрати найбільш зручний.

– Зручність оплати. Є можливість оплатити квиток багатьма способами, такими як банківські карти, електронні гаманці тощо.

Збільшення користувачів інтернету та телефонних пристроїв створює не аби-який попит на веб-застосунки, що забезпечують швидкий та зручний доступ до потрібних послуг. Розробка клієнтської частини веб-серверного застосунку для он-лайн купівлі квитків на автобусні перевезення є необхідною з кількох причин:

– Підвищення конкурентоспроможності перевізників. Перевізники, які надають можливість он-лайн купівлі квитків, мають перевагу перед конкурентами, які користуються лише традиційними методами.

– Задоволення потреб користувачів. Сучасні користувачі очікують зручних і швидких рішень для купівлі квитків. Веб-застосунок відповідає цим очікуванням.

– Оптимізація роботи перевізників. Автоматизація процесу продажу квитків зменшує навантаження на персонал та знижує ймовірність помилок.

– Збір та аналіз даних. Веб-застосунок дозволяє збирати дані про купівлі, що може бути використано для аналізу та покращення сервісу.

Актуальність та необхідність розробки клієнтської частини веб-серверного застосунку для он-лайн купівлі квитків на автобусні перевезення обумовлена зростаючими потребами споживачів у зручних та швидких сервісах, а також конкурентними перевагами для перевізників. Впровадження такої системи дозволяє підвищити якість обслуговування, оптимізувати роботу перевізників та сприяти соціально-економічному розвитку.

Існує багато он-лайн платформ для купівлі квитків на автобуси. Серед найпопулярніших можна виділити Busfor, Ecolines та Blablacar. Кожна з цих платформ має свої унікальні характеристики, переваги та недоліки.

Busfor одна з провідних онлайн-систем для купівлі автобусних квитків он-лайн. Однією з основних переваг цієї системи є її широка база перевізників, що дозволяє користувачам знайти найбільш підходящий рейс за різними критеріями. Інтерфейс сервісу є досить зручним та інтуїтивно зрозумілим, що полегшує процес пошуку та купівлі квитків (див. рис. 1.1). Також користувачі мають можливість обирати конкретні місця в автобусі під час бронювання, що додає зручності. Крім того, сайт підтримує декілька мов, що робить його доступним для користувачів з різних країн [1].

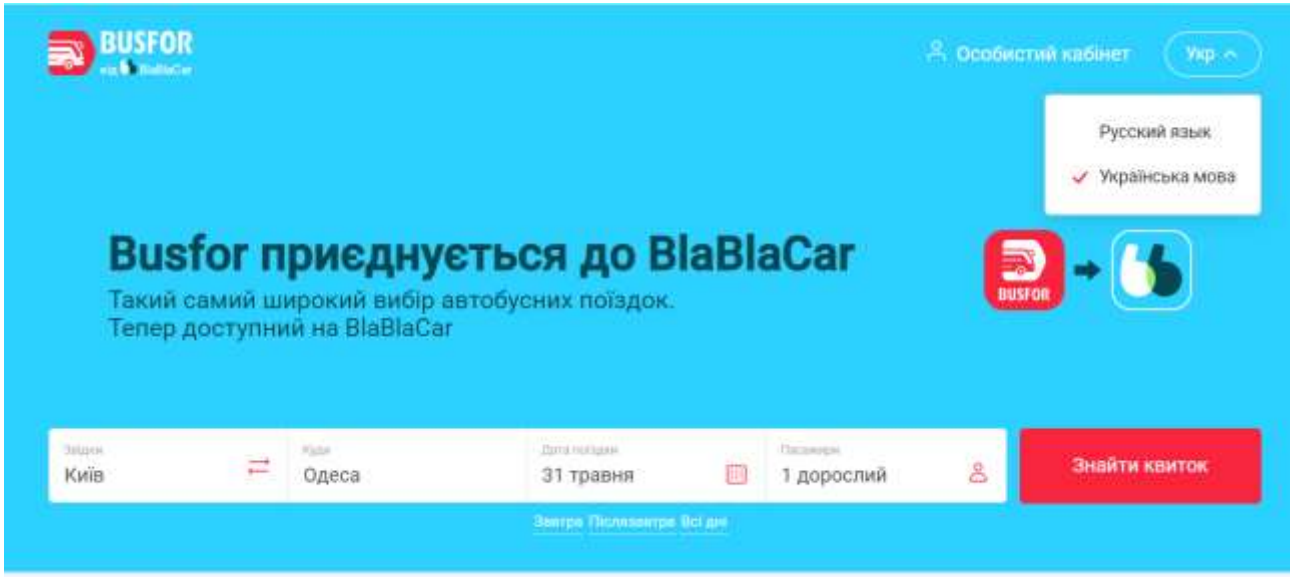

Рисунок 1.1 – Інтерфейс Busfor

Однак Busfor має і свої недоліки. Наприклад, сервіс стягує значні комісійні збори за купівлю квитків, що може значно збільшити вартість поїздки. Крім того, система не підтримує всі можливі способи оплати, що може бути незручним для деяких користувачів. Ще одним суттєвим недоліком є відсутність інтерактивних карт маршрутів (див. рис. 1.2), що ускладнює планування подорожей.

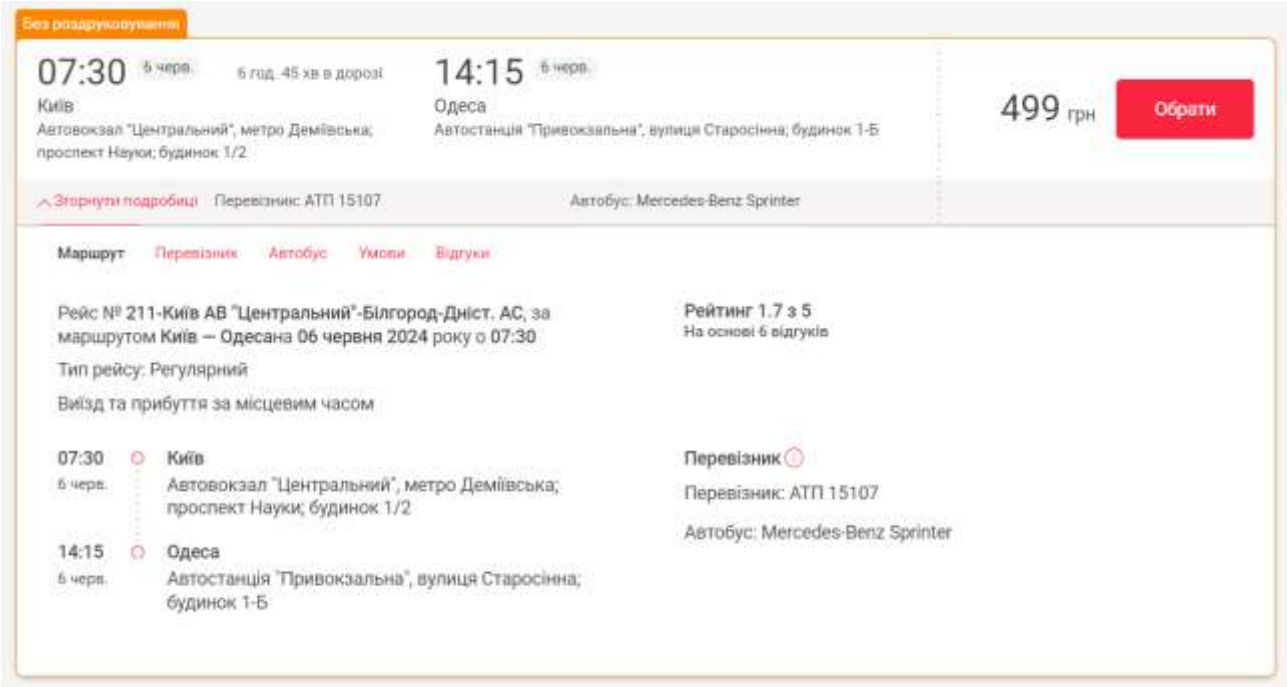

Рисунок 1.2 – Інтерфейс квитку Busfor

Ecolines є міжнародною компанією, що спеціалізується на автобусних перевезеннях. Компанія пропонує рейси по всій Європі, забезпечуючи високий рівень обслуговування. Основною перевагою Ecolines є можливість здійснювати міжнародні рейси, що робить його зручним для подорожей між країнами. Компанія також пропонує знижки для постійних пасажирів, що стимулює користувачів до повторних поїздок. Веб-сайт Ecolines має зрозумілий інтерфейс (див. рис. 1.3), який полегшує процес пошуку та бронювання квитків [2].

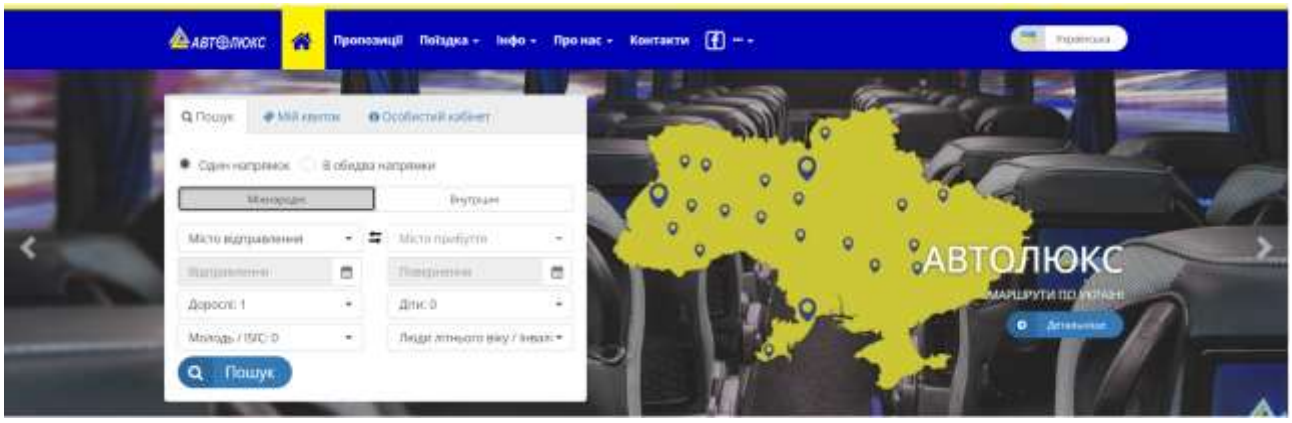

Рисунок 1.3 – Інтерфейс Ecolines

Втім, Ecolines має обмежену кількість маршрутів порівняно з іншими системами (див. рис. 1.4), що обмежує вибір для користувачів. Крім того, відсутність інтерактивної підтримки клієнтів може створювати проблеми при виникненні питань або проблем. Іноді процес обробки замовлень може займати більше часу, ніж очікувалося, що також може бути недоліком.

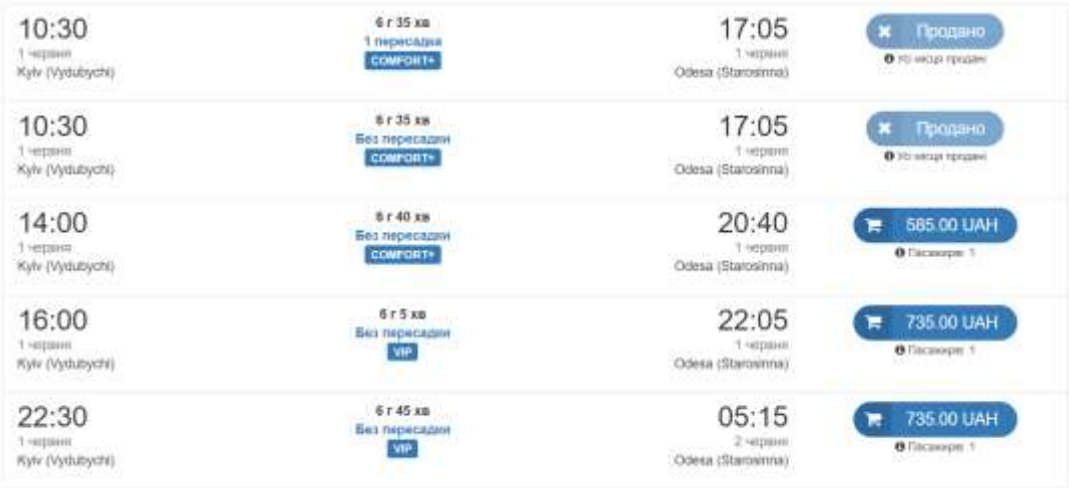

Рисунок 1.4 – Інтерфейс квитку Ecolines

Blablacar є платформою для спільних поїздок, що дозволяє користувачам знайти попутників для подорожей, як показано на рис. 1.5.

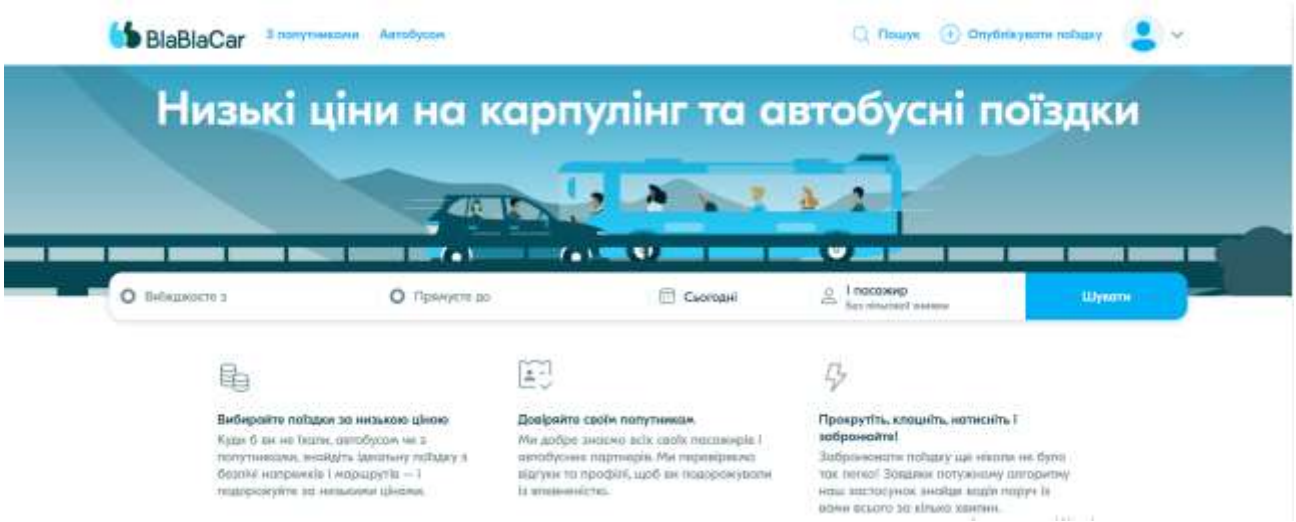

Рисунок 1.5 – Інтерфейс BlaBlaCar

Основною перевагою Blablacar є можливість вибору перевізника з урахуванням його рейтингу та відгуків інших пасажирів, що дозволяє обирати найбільш надійних та комфортних перевізників (див. рис. 1.6). Вартість поїздок на Blablacar зазвичай нижча порівняно з іншими системами завдяки спільному використанню автомобілів [3].

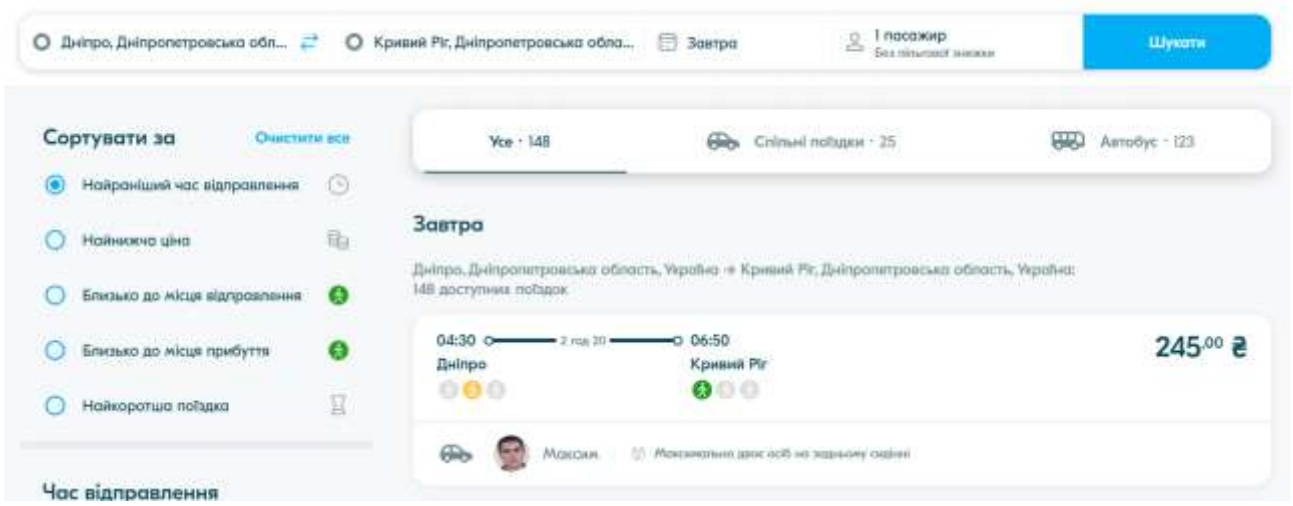

Рисунок 1.6 - Інтерфейс BlaBlaCar при пошуку квитків

Однак Blablacar не гарантує якість послуг, оскільки якість може варіюватися залежно від перевізника. Це може створювати непередбачуваність та ризики для пасажирів. Крім того, перевізники можуть скасувати поїздку в останній момент, що створює незручності для пасажирів. Вибір маршрутів також може бути обмеженим, особливо в менш популярних напрямках, що обмежує можливості користувачів. Також варто зазначити, що в Blablacar відсутня детальна інформація про квитки. У той час як у запропонованому веб-застосунку користувач може отримати детальну інформацію про рейс, маршрут, наявність зручностей та інші важливі аспекти, в Blablacar такої інформації часто бракує.

Ще одним значним недоліком Blablacar є відсутність чіткої валідації форми при пошуку квитків. Якщо користувач не заповнить всі необхідні поля, система просто підсвічує їх сірим кольором, але не вказує конкретну помилку. Це може ввести в оману користувача, оскільки не всі зрозуміють, що саме потрібно виправити. У запропонованому веб-застосунку валідація форми є більш інтуїтивною, з чіткими повідомленнями про помилки, що значно покращує користувацький досвід.

При пошуку квитків у Blablacar також відсутня функція відображення найближчих дат рейсів, якщо на обрану дату квитки не знайдено. У запропонованому веб-застосунку користувачам буде показано рейси на найближчі дати, що значно підвищує шанси знайти відповідний рейс. Крім того, в Blablacar немає можливості обирати валюту для відображення цін, що може бути незручним для міжнародних користувачів. У нашому веб-застосунку користувачі можуть обирати валюту для відображення цін, що робить систему більш гнучкою та зручною.

Ще однією значною перевагою нового веб-застосунку є можливість вибору перевезення тварин. Користувачі зможуть зазначити наявність тварин та їх вагу, що дозволить відразу обрати перевізника, який надає відповідні умови для подорожі з тваринами. Це значно полегшує процес планування поїздок для власників домашніх улюбленців.

У порівнянні з Blablacar, новий веб-застосунок пропонує більш детальну інформацію про квитки та рейси. Користувачі зможуть отримати всю необхідну інформацію про рейс, маршрут, наявність зручностей та інші важливі аспекти, що значно покращить процес планування подорожі.

Таким чином, аналіз існуючих он-лайн систем купівлі квитків показав, що, незважаючи на їхні численні переваги, є значний потенціал для вдосконалення та розвитку нових рішень. Створення нової системи дозволить врахувати сучасні вимоги та очікування користувачів, забезпечуючи високий рівень зручності, функціональності та надійності. Запропонований веб-застосунок не лише вирішує наявні проблеми існуючих систем, але й надає додаткові можливості, що роблять його більш привабливим та зручним для користувачів.

Важливою перевагою запропонованого веб-застосунку є розширені можливості для вибору пасажирів та їх параметрів. Користувачі можуть зазначити кількість дорослих, дітей та тварин, а також вказати вік дітей та вагу тварин. Це дозволяє відразу обрати перевізника, який надає відповідні умови для подорожі з дітьми або тваринами, що значно полегшує процес планування та підвищує рівень зручності для користувачів.

Ще одним важливим аспектом є впровадження системи лояльності та знижок для постійних пасажирів. Це стимулює користувачів до повторних поїздок та підвищує їхню лояльність до сервісу. Система лояльності дозволяє накопичувати бонуси за кожну поїздку, які можуть бути використані для отримання знижок на наступні поїздки.

Оптимізація роботи перевізників також є важливою складовою запропонованої системи. Автоматизація процесу продажу квитків зменшує навантаження на персонал та знижує ймовірність помилок. Крім того, система дозволяє збирати дані про купівлі, що може бути використано для аналізу та покращення сервісу. Аналіз даних дозволяє перевізникам оптимізувати свої маршрути, підвищити ефективність роботи та краще розуміти потреби своїх клієнтів.

Таким чином, розробка нового веб-застосунку для он-лайн купівлі квитків на автобус має значні переваги порівняно з існуючими системами. Інтуїтивний інтерфейс, інтерактивні карти, розширені можливості вибору пасажирів та системи лояльності забезпечують високий рівень зручності та функціональності для користувачів. Оптимізація роботи перевізників та можливість збору і аналізу даних сприяють підвищенню ефективності та якості обслуговування.

1.2. Формулювання задачі

В кваліфікаційній роботі поставлено завдання розробити клієнтську частину веб-серверного застосунку для он-лайн купівлі квитків для системи автобусного транспорту. Враховуючи важливість автоматизації процесу купівлі квитків на транспортні послуги, мета полягає в тому, щоб створити надійний, зручний і швидкий інструмент для користувача, який забезпечить ефективну взаємодію з частиною системи серверу і використанням зручного інтерфейсу користувача.

Задачі, які мають бути вирішені під час розробки клієнтської частини, включають в себе:

По-перше, необхідно зробити аналіз сучасних технологій та інструментів для створення реактивних і динамічних веб-інтерфейсів. Це включає вивчення переваг та недоліків різних фреймворків та бібліотек, зокрема Vue3, TypeScript та Pinia. Даний аналіз допоможе обрати найбільш оптимальний стек технологій для реалізації клієнтської частини веб-застосунку.

Після вибору технологій, потрібно розробити технічні вимоги до клієнтської частини. Ці вимоги охоплюють як функціональні аспекти, такі як можливість пошуку, фільтрації та сортування квитків, так і нефункціональні вимоги, такі як продуктивність, безпека та зручність використання.

На етапі реалізації клієнтської частини необхідно використати Vue3 для створення реактивних інтерфейсів, що забезпечують швидку та ефективну роботу веб-застосунку. TypeScript буде використовуватись для забезпечення

строгої типізації та зменшення кількості помилок у коді, а Pinia - для ефективного керування станом веб-застосунку.

Інтеграція з серверною частиною через API є наступним важливим кроком. Вона передбачає реалізацію механізмів для безпечної передачі даних між клієнтською та серверною частинами, забезпечує актуальною інформацією про доступність квитків, ціни та інші дані з сервера в реальному часі.

Наступним кроком є розробка функціоналу пошуку квитків, фільтрації та сортування отриманих даних. Форма для пошуку квитків повинна включати такі критерії, як місце відправлення, місце прибуття, дата та кількість пасажирів. Бокова панель фільтрації повинна дозволяти користувачам обирати подорож з пересадками чи без, тип транспорту (автобус чи мікроавтобус), час відправлення, а також групу комфорту (наявність Wi-Fi, туалету, розетки, клімат-контролю, напоїв, відеосистеми, аудіосистеми в автобусі, сидінь підвищеного комфорту). Функціонал для сортування даних повинен дозволяти сортувати результати за часом відправлення, зростанням та зниженням цін.

Функція "детальніше" для квитків є важливою складовою інтерфейсу. Кожен квиток повинен мати кнопку "детальніше", яка відкриває вікно з табами, які включають різну інформацію про певний рейс. Таби повинні мати деталі рейсу (повна інформація про квиток та галерея автобусу), маршрут (інформація про зупинки по містах та селах), знижки, правила (правила поїздки та інформація про багаж), та умови повернення квитків.

Крім того, важливо інтегрувати клієнтську частину з іншими компонентами сторінки. Це включає відображення списку схожих рейсів за результатами пошуку, додавання секції з питаннями та відповідями, а також секції з відгуками користувачів для покращення довіри та забезпечення зворотного зв’язку.

Щоб досягти високого рівня зручності використання веб-застосунку для кінцевих користувачів, необхідно ретельно протестувати та оптимізувати користувальницький інтерфейс. Це передбачає виявлення та усунення будь-яких

дефектів, оптимізацію продуктивності та забезпечення динамічного оновлення даних.

1.3. Аналіз методів та засобів розв’язування задачі

Під час розробки клієнтської частини веб-серверного застосунку для онлайн купівлі квитків в системі автобусних перевезень важливим етапом є аналіз існуючих методів та засобів, які можуть бути використані для ефективного розв’язання поставленої задачі. Аналіз допомагає знайти найкращі практики та інструменти, що забезпечать високу зручність використання, продуктивність та безпеку веб-застосунку.

1.3.1. Аналіз сучасних фреймворків для розробки клієнтської частини

Проведемо аналіз сучасних фрейморків й оберемо найкращий варіант для поставленого завдання. Розглянемо три найпопулярніші фрейморки серед front-end розробки: Vue.js, React та Angular. Кожен користується великою популярністю та попитом, має свої переваги та недоліки. Розглянемо детальніше кожен з цих фреймворків.

Простий у вивченні Vue.js є таким завдяки зрозумілій, легкій та детальній документації та активній спільноті, що оновлює її. Він пропонує високу модульність та гнучкість, що дозволяє створювати окремі компоненти та масштабні програми. Vue.js має реактивну систему, яка автоматично відстежує зміни, що відбуваються в даних і оновлює користувальницький інтерфейс в режимі реального часу. Недоліки Vue.js є обмежена підтримка великих корпоративних проектів та може мати меншу кількість готових рішень і інструментів у порівнянні з React [4].

React використовує віртуальний DOM для забезпечення високої продуктивності. Це дозволяє мінімізувати операції з реальним DOM і збільшує швидкість рендерингу сторінок. React також дозволяє створювати як невеликі

компоненти, так і великі масштабовані застосунки, забезпечуючи гнучкість і розширюваність. Завдяки активній і великій спільноті, React має доступ до великої кількості документації, бібліотек та інструментів, зразків, які допомагають швидко знаходити відповіді на питання та вирішення проблеми. Однак, React потребує значних початкових зусиль із налаштування та використання додаткових бібліотек та інструментів [5].

Angular — це повнофункціональний фреймворк, який надає повний набір інструментів і рішень для розробки веб-додатків, включає вбудовані модулі для роботи з формами, HTTP-запитами, маршрутизацією та керуванням станом. Angular має достатню популярність серед великих компаній через свою стабільність та здатність підтримувати крупні проекти, де є велика кодова база. Модульна архітектура дозволяє розділяти код на окремі частини, що полегшує його підтримку та розширення. Однак Angular може стати перешкодою для менш досвідчених розробників, оскільки він має круту криву навчання, особливо для новачків, і вимагає знання TypeScript і розуміння складних концепцій [5].

Насправді, кожен з цих фреймворків має свої переваги та популярний у своєму використанні, тому вибір для проекту фреймворку має бути індивідуальним та обран під умови та наповненість самого проекту..

Після проведення детального аналізу було прийнято рішення використовувати Vue.js для розробки клієнтської частини веб-застосунку. Рішення засноване на простоті та гнучкості Vue.js, його потужній системі реактивності. Ці характеристики роблять Vue.js ідеальним вибором для проекту, який має на меті створити зручний та ефективний інструмент для користувачів.

1.3.2. Використання TypeScript для забезпечення надійності коду

Ключова вимога до розробки клієнтської частини веб-застосунку є забезпечення надійності та стійкості коду. Це найголовніше, що є в розробці веб-застосунку, якщо грамотно не написати код, методи, то в якійсь момент програма може зламатись і шукати причину серед всього коду буде складно, тому на

допомогу приходить TypeScript. У цьому контексті використання TypeScript є оптимальним рішенням, оскільки цей інструмент значно підвищує якість коду та знижує ризик помилок. Розглянемо переваги використання цієї мови програмування та недоліки.

Основні переваги використання TypeScript:

1) Статична типізація. TypeScript додає до JavaScript строгий контроль типізації, що дозволяє розробникам визначати типи змінних, параметрів в функціях та значеннях, що повертаються. Це допомагає уникнути багатьох типових помилок, таких як передача некоректних типів даних в функції або використання змінних до їх ініціалізації чи відправлення на сервер не того типу данні.

2) Виявлення помилок на етапі компіляції. На відміну від JavaScript, де помилки в коді часто виявляються лише під час виконання програми, TypeScript дозволяє виявляти та виправляти помилки ще на етапі компіляції. Це значно знижує ризик помилок у готовому продукті та підвищує стабільність застосунку, особливо коли проект масштабний, то важливість виявлення одразу помилок є невід'ємною частиною розробки [6].

3) Покращена підтримка інструментів розробки. TypeScript надає кращу інтеграцію з інструментами розробки, такими як інтегровані середовища розробки і редакторами коду. Завдяки цьому розробники отримують доступ до потужних функцій автоматичного доповнення коду, навігації по коду та рефакторингу, що підвищує продуктивність та зручність роботи.

4) Спрощена підтримка та розширення коду. Завдяки чітким типам та інтерфейсам, код, написаний на TypeScript стає більш читабельним та структурованим. Це полегшує підтримку коду та його розширення, особливо при роботі з великими командами або довгостроковими проектами.

5) Сумісність з JavaScript. TypeScript — це надмножина JavaScript, що означає, що будь-який код на JavaScript є дійсним кодом на TypeScript. Це дозволяє поступово переходити на TypeScript в поточному проекті, не

порушуючи роботу. Крім того, TypeScript компілюється на чистому JavaScript, забезпечуючи сумісність з будь-яким середовищем, яке підтримує JavaScript.

Перейдемо до розгляду недоліків цієї мови програмування:

1) Час для розробки потрібен більше.

2) В момент компіляції може бути витрачено більше часу, тому що TypeScript перетворюється у JavaScript, а це займає час.

3) Під час розробки проекту та використанню сторонніх бібліотек модуть виникнути проблеми із сумісністю. При підключенні бібліотек треба читати про їх сумісність із TypeScript та якщо що, то завантажувати в проект бібліотеку із розширенням під TypeScript.

4) Найменування — ускладення, через те, що забагато інтерфейсів і для кожного треба придумати підходящий неймінг [7].

Проаналізувавши переваги та недоліки можна дійти до висновку, що переваг набагато більше ніж недоліків.

Строга типізація, виявлення помилок на етапі компіляції, покращена підтримка інструментів розробки та легша підтримка коду роблять TypeScript надзвичайно цінним інструментом для створення стабільних, надійних та масштабованих веб-застосунків. Зокрема, для проекту з розробки клієнтської частини веб-застосунку для он-лайн купівлі квитків TypeScript забезпечить високий рівень якості коду та сприятиме ефективній розробці та підтримці проекту в довгостроковій перспективі.

1.3.3. Управління станом застосунку за допомогою Pinia

Управління станом додатку є важливою частиною розробки сучасних веб-застосунків, особливо коли в проекті велика кількість взаємопов'язаних компонентів. В рамках даної кваліфікованої роботи для управління станом обрано бібліотеку Pinia. Вона забезпечує ефективне та гнучке керування станом у застосунках, розроблених на базі Vue.js. Концепція представлення «одностороннього потоку даних» (див. рис. 1.7).

Перейдемо до розгляду переваг використання Pinia:

1) Простий та зрозумілий API. Pinia надає зрозумілий API, в якому легко розібратись та вивчити. Це надає розробникам швидкість інтегрувати Pinia у проекти.

2) Модульність. Pinia підтримує модульний підхід до управління станом, що дозволяє розділяти логіку стану на окремі модулі. Така перевага дозволяє зробити код в сховищі простим та читабельним, розділити на модулі, наприклад, по сторінкам чи компонентам. Коли проекти великі, це дуже допомагає не загубитись, та чітко розуміти, де знаходяться ті чи інші дані.

3) Реактивність. Pinia використовує реактивність Vue.js для автоматичного відслідковування змін у стані та оновлення компонентів у режимі реального часу. Ця перевага надає велику зручність, дані в інтерфейс змінюються автоматично та користувачу не потрібно перезавантажувати сторінку кожен раз [8].

4) Підтримка TypeScript. Pinia має відмінну інтеграцію з TypeScript, що дозволяє використовувати переваги строгого типізаційного контролю для управління станом.

5) Інтеграція з іншими інструментами. Pinia легко інтегрується з іншими бібліотеками та інструментами Vue.js, це дозволяє розробникам використовувати її з Vue Router, Vuex-PersistedState та іншими популярними рішеннями [8].

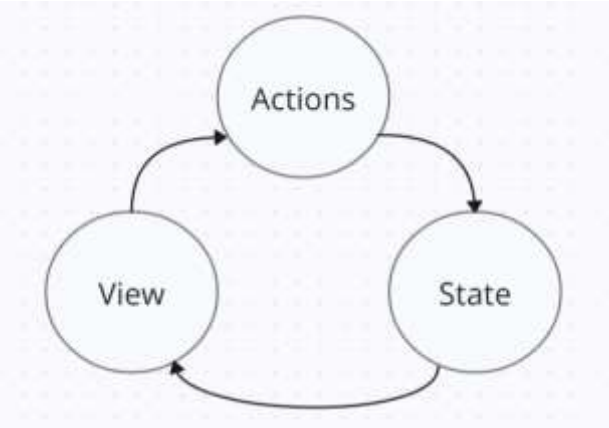

Рисунок 1.7 – Концепція одностороннього потоку даних

1.3.4. Інтеграція з серверною частиною через API

1) Інтеграція клієнтської частини веб-застосунку із серверною частиною через API є важливим аспектом для забезпечення постійного зв'язку між користувачами та системою. В кваліфікаційній роботі для інтеграції використовується метод fetch. Він дозволяє клієнтському застосунку виконувати HTTP-запити до сервера для отримання та обробки даних. Використання fetch забезпечує простоту та гнучкість інтеграції, а також підтримує основні методи HTTP (GET, POST, PUT, DELETE), необхідні для виконання CRUD операцій [9].

2) Розглянемо головні аспекти, які є в інтеграції із серверною частиною через API:

3) Використання fetch. Fetch API надає інтерфейс для виконання асинхронних HTTP-запитів з клієнта до сервера. Він підтримує Promesses, тому можна обробляти результати запиту або помилки, що можуть виникнути під час виконання запиту.

4) Формат даних. Формат JSON використовується для обміну даними між клієнтом і сервером. Це простий, зручний для читання формат, який полегшує серіалізацію та десеріалізацію даних з обох сторін взаємодії.

5) Безпека. Для забезпечення безпеки передачі даних між клієнтом і сервером використовується HTTPS. Це дає змогу захистити дані від перехоплення та несанкціонованого доступу під час передачі через мережу.

6) Обробка помилок. Важливим аспектом інтеграції є обробка помилок, що можуть виникати під час взаємодії з сервером. Це включає обробку мережевих помилок, неправильних даних або відсутності доступу до ресурсу. Відповідні повідомлення про помилки повинні бути відображені користувачеві, щоб забезпечити інформативний зворотний зв'язок.

1.4. Висновки до першого розділу. Постановка задачі дослідження

В першому розділі було проведено детальний аналіз методів розробки клієнтської частини веб-серверного застосунку для он-лайн купівлі квитків в системі автобусних перевезень, проаналізувавши все, можемо перейти до поставлення задач. З урахуванням актуальності автоматизації процесу придбання квитків на транспортні послуги, метою є створення зручного, надійного і швидкодіючого інструменту для користувачів, що забезпечує ефективну взаємодію з серверною частиною системи та зручний користувальницький інтерфейс.

Основні задачі, які мають бути вирішені в процесі розробки клієнтської частини, включають:

1) Забезпечення отримання актуальної інформації про доступність квитків, ціни та інші необхідні дані з сервера в реальному часі.

2) Створення форми для пошуку квитків за різними критеріями, такими як місце відправлення, місце прибуття, дата і час поїздки.

3) Додавання бокової панелі фільтрації результатів пошуку за параметрами, такими як:

- Подорож з пересадками чи без.
- Вибір типу транспорту: автобус чи мікроавтобус.
- Вибір часу відправлення.
- Група комфорту: наявність Wi-Fi, туалету, розетки, клімат-контролю, напоїв, відеосистеми, аудіосистеми в автобусі, сидінь підвищеного комфорту.

4) Реалізація функціоналу для сортування даних за обраними критеріями: за часом відправлення, зростанням цін та зниженням цін.

5) Додавання кнопки "детальніше" для кожного квитка, яка відкриває вікно з вкладками, що містять різну інформацію про рейс.

6) Вкладка "Деталі рейсу": повна інформація про квиток та галерея автобусу.

7) Вкладка Маршрут: інформація про зупинки по містах та селах.

8) Вкладка Знижки: інформація про доступні знижки.

9) Вкладка Правила: правила поїздки та інформація про багаж.

10) Вкладка Умови повернення: умови повернення квитків.

11) Відображення списку схожих рейсів за результатами пошуку для надання альтернативних варіантів.

12) Додавання секції з питаннями та відповідями для допомоги користувачам.

13) Додавання секції з відгуками для покращення довіри користувачів та забезпечення зворотного зв’язку.

14) Проведення всебічного тестування клієнтської частини для виявлення та усунення недоліків.

15) Оптимізація продуктивності та забезпечення динамічного оновлення даних, що забезпечить високий рівень зручності використання для кінцевих користувачів.

Цей розділ демонструє значення та важливість розробки даної системи, що є основою для подальших досліджень та розробок у цьому напрямку.

РОЗДІЛ 2. ПОЧАТКОВЕ ПРОЕКТУВАННЯ ВЕБ-ЗАСТОСУНКУ

2.1. Проектування веб-застосунку

Перед тим, як перейти до написання коду програми, потрібно ретельно продумати її логіку, це дуже важливий етап в роботі над проектом, що дає змогу в подальшому не зробити глобальних помилок та витрачати час на переробку зроблених дій. Побудова логіки вимагає всебічного аналізу та планування функціональності програми, її взаємодії з користувачем та обробки потенційних помилок. Розробка логіки програми включає кілька важливих етапів:

– визначення функцій програми

– розбиття на компоненти

– обробка помилок

– використання типів даних

1) Визначення функцій програми

На початковому етапі проектування необхідно визначити основні функції, які буде виконувати веб-застосунок. Для он-лайн купівлі квитків в системі автобусних перевезень такими функціями є:

– Пошук квитків за заданими параметрами (місце відправлення, місце прибуття, дата, час).

– Фільтрація та сортування результатів пошуку.

– Відображення детальної інформації про рейс (деталі рейсу, маршрут, знижки, правила та умови повернення квитків).

– Взаємодія з API для отримання даних про рейси.

2) Розбиття на компоненти

Після визначення функціоналу програми необхідно розділити її на окремі компоненти. Кожен компонент повинен виконувати конкретну функцію і бути незалежним від інших компонентів. Це дозволить спростити розробку,

тестування та підтримку застосунку. Основні компоненти веб-застосунку включають:

– Форма пошуку квитків. Інтерфейс для введення параметрів пошуку.

– Бокова панель фільтрації. Можливість фільтрації результатів пошуку за різними параметрами.

– Результати пошуку. Відображення списку квитків, що відповідають заданим критеріям.

– Компонент квитка. Відображення інформації про окремий квиток з можливістю бронювання та перегляду детальної інформації.

– Вікно з детальною інформацією про квиток. Вкладки з інформацією про деталі рейсу, маршрут, знижки, правила та умови повернення квитків.

3) Обробка помилок

При розробці веб-застосунку необхідно передбачити обробку помилок, які можуть виникати під час взаємодії користувачів із застосунком. Основні аспекти обробки помилок включають:

– Обробка помилок валідації. Перевірка введених користувачем даних на коректність перед відправкою на сервер.

– Відображення повідомлень про помилки. Інформування користувача про виниклі помилки у зрозумілій формі, що дозволяє користувачу вжити необхідні дії для їх виправлення (наприклад, повідомлення про некоректний формат даних або про помилки під час пошуку квитка).

В результаті все виглядає так, як показано на рисунку 2.1. На цьому рисунку показані основні функції програми, точка входу в програму. У кожному з наступних блоків відображається пункт роботи веб-застосунку.

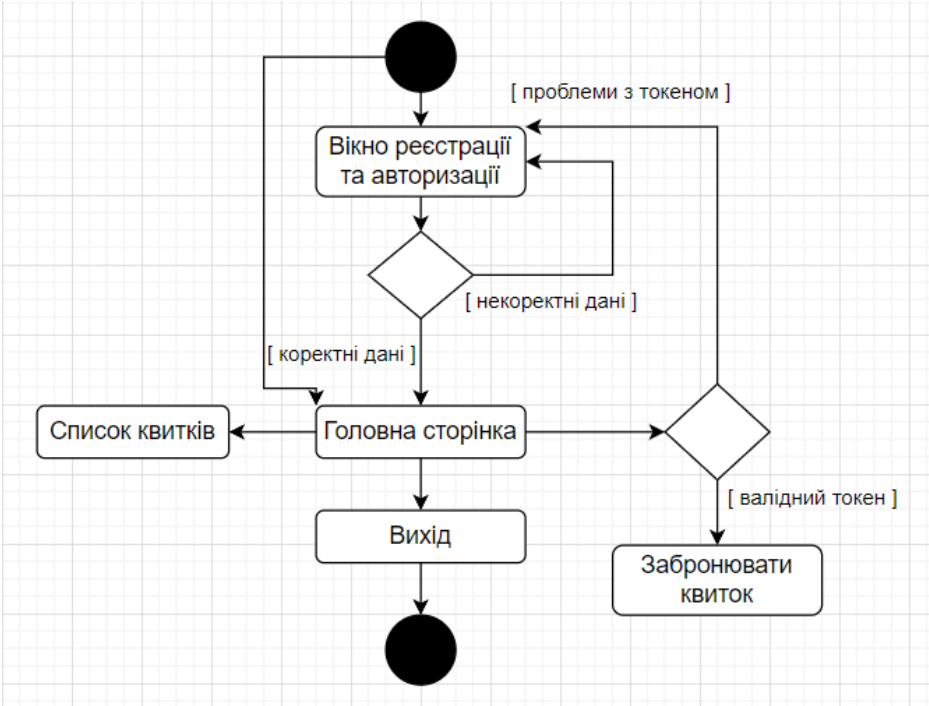

Рисунок 2.1 – Діаграма програми

2.2. Розробка інтерфейсу користувача

Взагалі, дизайн інтерфейсу користувача є одним з ключових факторів розробки. Хоча і написання коду та функціоналу за думкою багатьох є найважливішим, але дизайн не менш важливий в цьому питанні, йому потрібно приділяти багато часу та продумати, щоб інтерфейс веб-застосунку був легким, зрозумілим та гарним. Психіка людей працює таким чином, що як тільки щось не зрозуміло, заплутано, не гарно, то перше, що прийде людині в голову — це вийти із цього вікна, та знайти інший веб-застосунок, який буде зрозумілішим та легким у використанні.

Мета цього етапу – забезпечити зручний, інтуїтивно зрозумілий та функціональний інтерфейс, який дозволить користувачам легко взаємодіяти із системою, здійснювати пошук, бронювання та купівлю квитків.

Розглянемо основні принципи розробки інтерфейсу користувача:

1) Інтуїтивність. Інтерфейс має бути легким для розуміння та використання навіть для нових користувачів, а особливо для тих, хто ніколи не

користувався веб-застосунками. Всі елементи управління повинні бути зрозумілими та легко доступними.

2) Мінімалізм. Потрібно уникати перевантаження інтерфейсу зайвими елементами, не потрібно робити весь простір заповненим, чим більше місця вільного є, тим легше дихати користувачу. Зосередження має бути на основних функціях та інформації, яка потрібна користувачу для виконання його потреб.

3) Відгук. Інтерфейс повинен швидко реагувати на дії користувача, надаючи зворотний зв’язок про виконані дії.

4) Адаптивність. Інтерфейс повинен бути адаптивним та коректно відображатися на різних пристроях та екранах. Майже всі користувачі інтернету, користуються мобільними телефонами, тому гарним та великим плюсом буде зробити адаптив під всі розміри телефонів, до мінімального, щоб у будь-якого користувача все мало гарний та зрозумілий вигляд.

Проект розробляється для компанії автобусних перевезень Rubikon, їх основні кольору (див. рисунок 2.2) – це жовті відтінки та сірі.

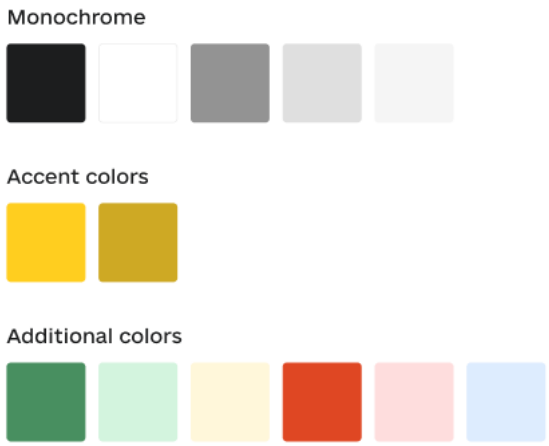

Рисунок 2.2 – Кольори, що використовуються у розробці веб-застосунку

Перейдемо до розгляду дизайну. Шапка сторінки містить логотип компанії, номери телефоні для з’єднання та уточнення інформації та кнопки для переходу на сторінку підтримки та в особистий кабінет користувача (див. рис. 2.3).

Форма для пошуку квитків містить такі поля (див. рис. 2.3): звідки виїзд, куди, дата поїздки та пасажири.

Рисунок 2.3 – Компонент шапки веб-застосунку та форми для пошуку квитків

При натисканні на поля «Звідки» та «Куди» вводиться місто або країна та відкривається вікно з вибором (див. рис. 2.4). Також між полями «Звідки» та «Куди» є перемикач, тобто зміна місцями даних.

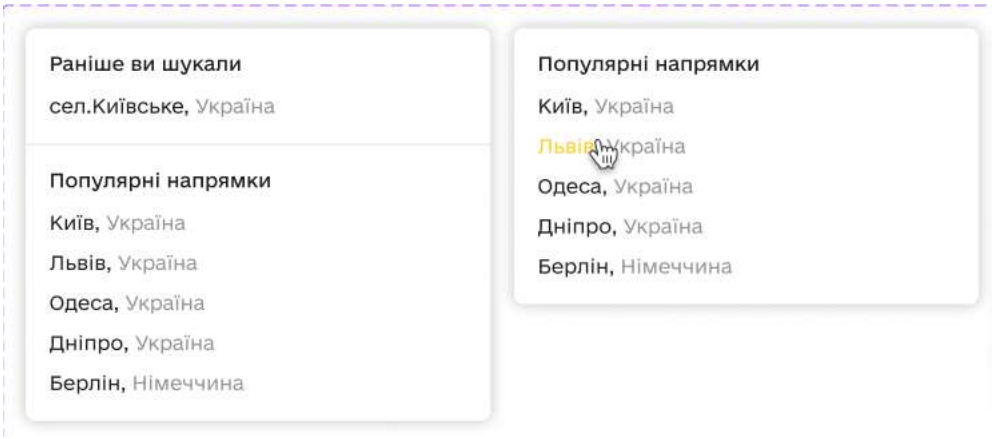

Рисунок 2.4 – Вікно для результатів пошуку по полям «Звідки» та «Куди»

При натисканні на поле «Дата поїздки» відкривається календар (див. рис. 2.5), де можна обрати день, місяць та рік поїздки.

Рисунок 2.5 – Відкритий календар

При натисканні на поле с пасажирами відкривається вікно (див. рис. 2.6) для обрання пасажирів, є такі варіанти вибору: дорослий, дитячий та з тваринами поїздка.

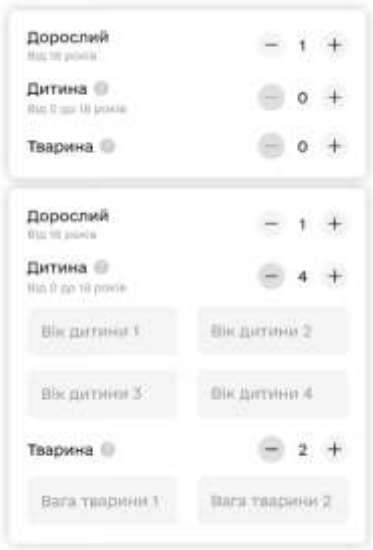

Рисунок 2.6 – Вікно для обрання пасажирів

Обов’язковим етапом є малювання помилок користувачеві (див. рис. 2.7), так як не всі розуміють, що заповнення усіх полів пошуку є обов’язковим моментом для отримання очікуваного результату.

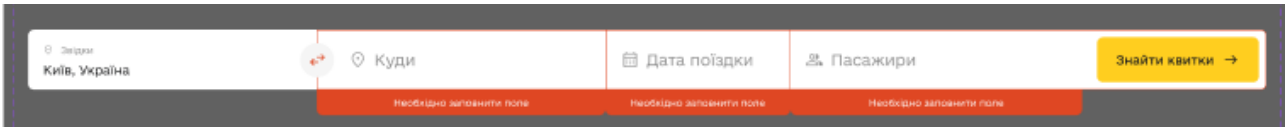

Рисунок 2.7 – Обробка помилок при пошуку квитків

Задля того, щоб веб-застосунок чіпляв користувача, потрібно зробити різні стани для кнопок (див. рис. 2.8).

Рисунок 2.8 – Різні стани для кнопок

Тепер перейдемо до розгляду інтерфейсу самого квитка (див. рис. 2.9). Сам квиток може мати значок-схвалення від самої компанії Rubikon, так як ця компанія співпрацює з різними автобусними компаніями. Квиток дає інформацію користувачу про час та місце відправлення та прибуття, показує які послуги є в автобусі для зручної поїздки, години, які пасажир проведе у автобусі, скільки є місць вільних, дає можливість забронювати квиток.

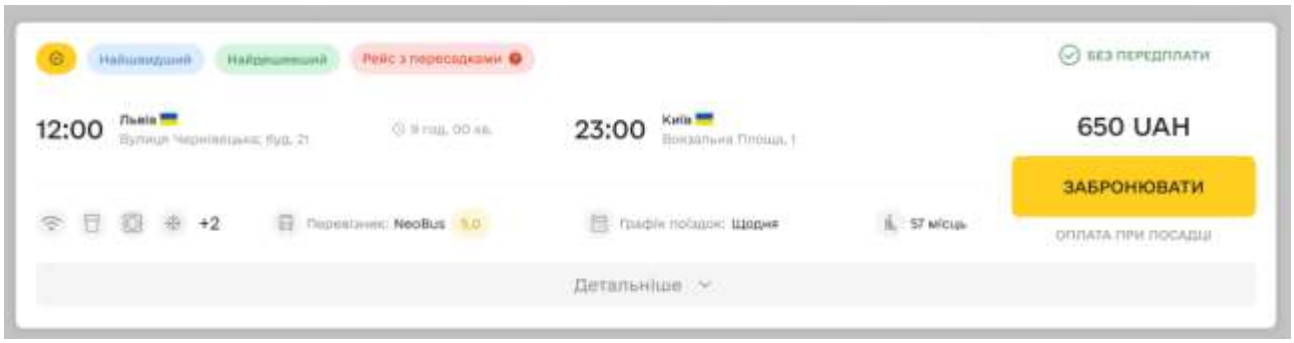

Рисунок 2.9 – Дизайн квитка

Як видно з рисунку 2.9, квиток складається з інформації про нього та кнопки «Детальніше», при натисканні на неї домальовується блок з детальною інформацію про рейс.

Розглянемо першу табу «Деталі» (див. рис. 2.10). Це вікно має інформацію про сам автобус, комфорт (послуги, які можна отримати при поїздці в цьому автобусі), умови купівлі квитка, додаткову інформацію про перевізника та галерею автобусу. Галерея є великою перевагою, так як пасажир може подивитись як виглядає автобус зовні та всередині.

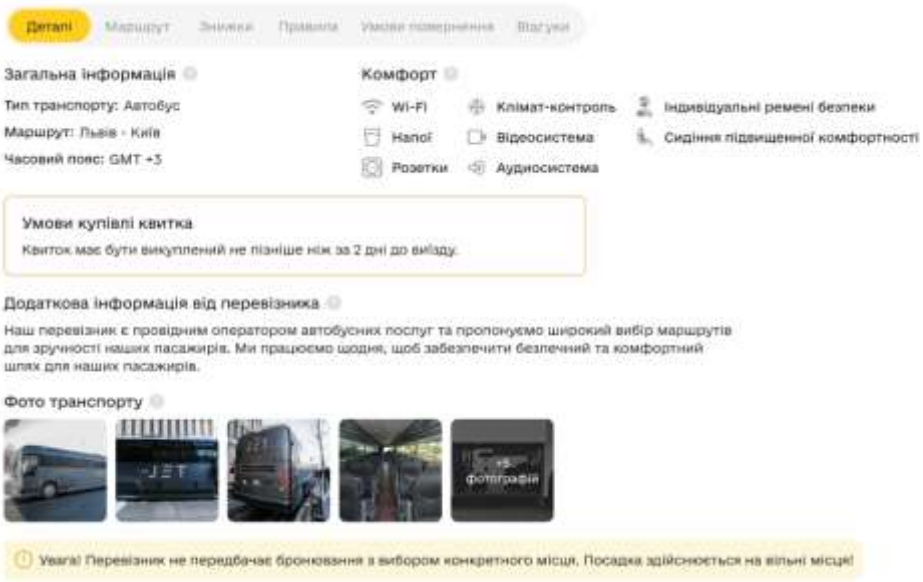

Рисунок 2.10 – Таба «Деталі»

Перейдемо до розгляду таби «Маршрут» (див. рис. 2.11). Ця таба дає зрозумілий маршрут поїздки, показуючи час прибуття в кожне місто чи село, пересадки, затримки та карту.

Рисунок 2.11 – Таба «Маршрут»

Розглянемо наступну табу «Знижки», вона містить інформацію про доступні знижки:

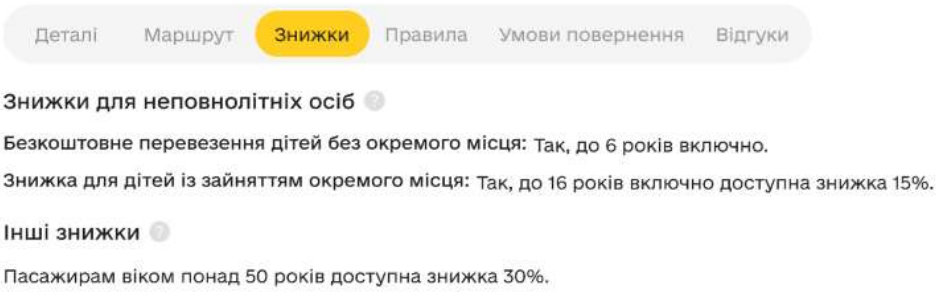

Рисунок 2.12 – Таба «Знижки»

Таба «Правила» містить інформацію про правила перевезення дітей та тварин, це дає змогу пасажирам заздалегіть знати всі деталі та не опинитись в складній ситуації, особливо з тваринами (див. рис. 2.13).

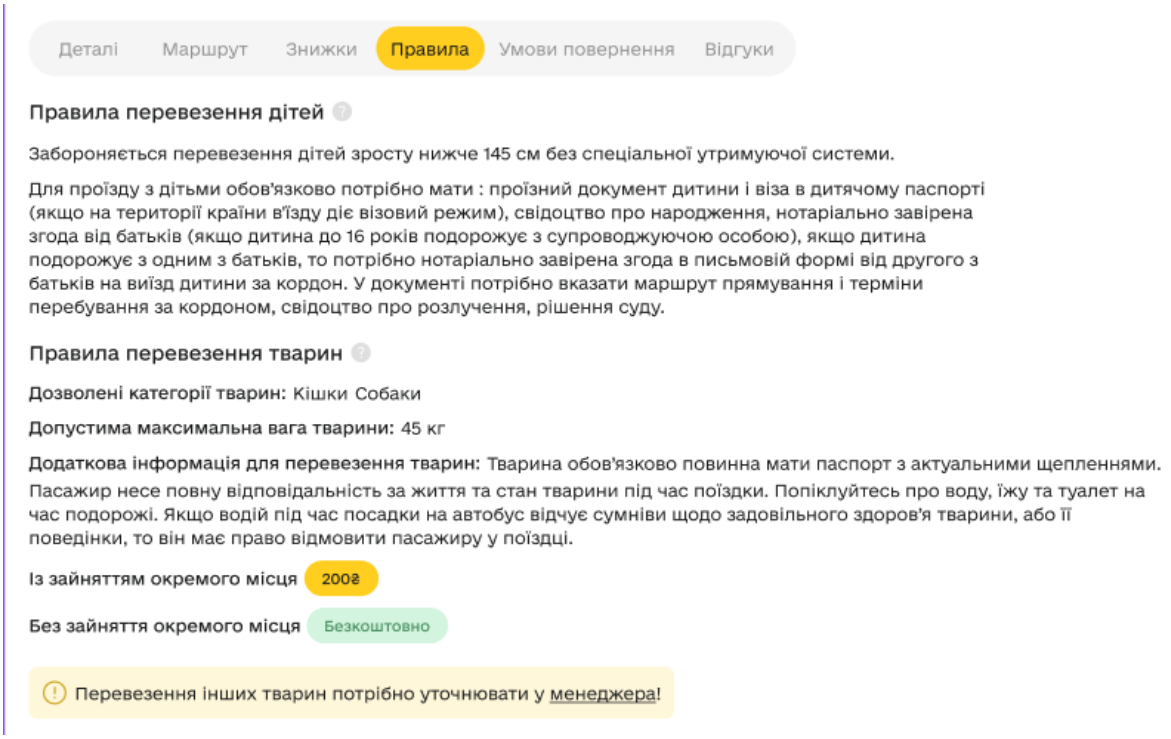

Рисунок 2.13 – Таба «Правила»

Секція «Питання та відповіді» має вигляд, показаний на рисунку 2.14, це допомагає користувачам отримати відповіді на найпопулярніші питання, що збереже їх власний час на з'ясування відповіді.

Рисунок 2.14 – Секція «Питання та відповіді»

Бокова панель для вибору фільтрів складається із блоків, як показано на рисунку 2.15:

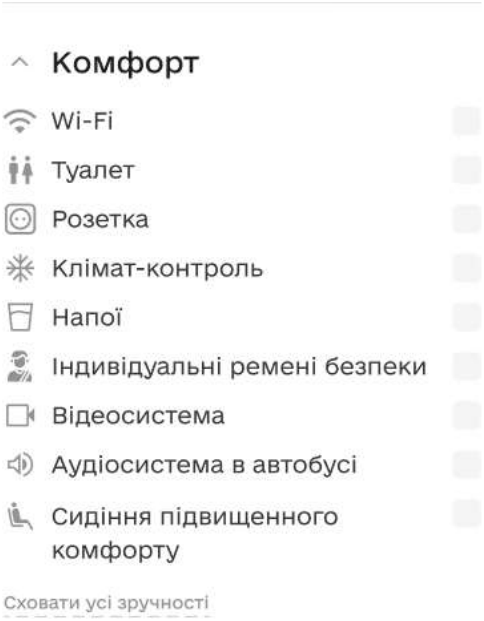

Рисунок 2.15 – Частина бокової панелі вибору фільтрів

2.3. Висновки до другого розділу

У другому розділі було розглянуто ключові аспекти розробки веб-застосунку для он-лайн купівлі квитків на автобус. Проектування веб-застосунку та розробка інтерфейсу користувача є важливими етапами, що забезпечують створення функціонального, надійного та зручного для користувачів продукту. Ці процеси нерозривно пов’язані з комп’ютерними науками, оскільки вимагають глибоких знань та навичок у галузі програмування, системного аналізу, проектування та забезпечення якості програмного забезпечення.

Проектування веб-застосунку включає визначення основних функцій, розбиття на компоненти, обробку помилок та використання типів даних. Визначення функцій програми допомагає чітко окреслити завдання веб-застосунку, такі як пошук квитків, їх фільтрація та сортування, бронювання та купівля квитків, управління обліковими записами користувачів та взаємодія з API. Розбиття на компоненти забезпечує модульність та легкість у підтримці та

розвитку застосунку. Кожен компонент виконує конкретну функцію, що сприяє кращій організації коду, забезпечуючи принципи об'єктно-орієнтованого програмування та модульності, які є основою комп'ютерних наук.

Ці етапи розробки веб-застосунку безпосередньо пов'язані зі спеціальністю «Комп'ютерні науки». Аналіз вимог до системи, проектування архітектури веб-застосунку та розробка його компонентів включає розбиття на модулі, що є важливою частиною навчальної програми з комп'ютерних наук. Розробка інтерфейсу користувача, що забезпечує зручність та інтуїтивність використання, включає принципи дизайну UI/UX, які є важливими аспектами комп'ютерних наук. Створення механізмів для тестування функціональності та обробки помилок демонструє застосування методів забезпечення якості програмного забезпечення, що є критично важливим для стабільної роботи веб-застосунку.

Таким чином, розробка веб-застосунку для он-лайн купівлі квитків на автобус не тільки підкреслює важливість комп'ютерних наук у сучасних інформаційних технологіях, але й демонструє практичне застосування теоретичних знань у реальних проектах. Від проектування архітектури та інтерфейсу до реалізації та тестування, всі ці етапи вимагають глибокого розуміння та застосування основних принципів комп'ютерних наук. Це включає в себе не тільки технічні навички програмування, але й здатність аналізувати та вирішувати комплексні проблеми, проектувати ефективні рішення та забезпечувати високу якість і безпеку кінцевого продукту.

Отже, другий розділ цієї роботи не тільки детально розкриває процеси проектування та розробки веб-застосунку, але й демонструє тісний зв’язок між цими процесами та фундаментальними принципами комп’ютерних наук. Це підтверджує, що успішне створення сучасних веб-застосунків вимагає не лише практичних навичок, але й глибокого розуміння теоретичних основ, що надаються в рамках спеціальності «Комп’ютерні науки».

РОЗДІЛ 3. РОЗРОБКА ВЕБ-ЗАСТОСУНКУ ДЛЯ ОН-ЛАЙН КУПІВЛІ КВИТКІВ В СИСТЕМІ АВТОБУСНИХ ПЕРЕВЕЗЕНЬ

3.1. Розробка програми та її опис

Розробка веб-застосунку для он-лайн купівлі квитків у системі автобусних перевезень є складним і багатоступеневим процесом, що включає планування, написання коду, тестування та оптимізацію. У цьому розділі розглянемо ключові аспекти розробки програми, її архітектуру та використані технології.

Для розробки веб-застосунку було обрано інтегроване середовище розробки Visual Studio Code (VS Code). Це популярне середовище, яке забезпечує потужні інструменти для розробки веб-додатків, включаючи підтримку різних мов програмування, розширення для підвищення продуктивності, інтеграцію з системами контролю версій та багато інших корисних функцій [10].

Веб-застосунок розроблений з використанням кількох ключових технологій та інструментів. На фронтенд частині застосунку було використано фреймворк Vue.js, який дозволяє створювати реактивні та динамічні інтерфейси користувача [11]. Для забезпечення строгої типізації та зменшення кількості помилок у коді використовується мова програмування TypeScript. Управління станом додатку здійснюється за допомогою бібліотеки Pinia, яка забезпечує гнучкість і простоту використання. Для виконання HTTP-запитів до сервера для обміну даними використовується Fetch API. Основні технології для створення структури та стилізації веб-сторінок включають HTML та CSS.

Архітектура веб-застосунку включає кілька основних компонентів, кожен з яких виконує окрему функцію. Одним з ключових компонентів є форма пошуку квитків, яка містить поля для введення місця відправлення, прибуття, дати та кількості пасажирів. Після заповнення цих полів користувач може натиснути кнопку для виконання пошуку квитків. Бокова панель фільтрації дозволяє користувачам налаштовувати фільтри для вибору типу транспорту, наявності

зручностей та інших параметрів, а також сортувати результати пошуку за різними критеріями.

Результати пошуку відображаються у вигляді списку доступних квитків, де кожен квиток показує основну інформацію про рейс. Користувач також має можливість переглянути детальну інформацію про кожен квиток, натиснувши на відповідну кнопку. Детальна інформація про квиток включає вікно з вкладками, де можна дізнатися деталі рейсу, маршрут, знижки, правила та умови повернення квитків.

Програма розроблена на мові TypeScript, що забезпечує високий рівень типізації і дозволяє уникнути багатьох типових помилок на етапі розробки. Використання Vue.js дозволяє створювати інтерактивні та реактивні інтерфейси, що покращує користувацький досвід. Pinia забезпечує ефективне управління станом додатку, а Fetch API використовується для надійної та безпечної взаємодії з сервером. Завдяки використанню цих технологій вдалося створити функціональний та зручний для користувачів веб-застосунок, який відповідає сучасним вимогам і стандартам розробки програмного забезпечення.

Розглянемо вирішення певних задач в коді, та проаналізуємо вибір методів розв’язування цих задач.

3.1.1. Реалізація пошуку квитків

Для реалізації пошуку квитків у веб-застосунку було створено окремий компонент SearchForm. Цей компонент включає кілька підкомпонентів, які забезпечують різні аспекти функціональності пошуку. Зокрема, він містить поля для введення місця відправлення та прибуття, календар для вибору дати поїздки, компонент для вибору пасажирів та кнопку для виконання пошуку.

Поля для введення місця відправлення та прибуття мають модальні вікна, які відкриваються при пошуку і дозволяють користувачам вибрати місто, як наведено на рис. Б1 додатку Б. Поле для введення місця відправлення

називається "Звідки", а поле для введення місця прибуття - "Куди". Це забезпечує зручність введення та дозволяє уникнути помилок під час вибору місця.

Компонент календаря (дейтпікера) дозволяє користувачам обрати дату поїздки, що є важливою частиною пошукового запиту. Також у пошуковій формі є компонент для вибору пасажирів, який дозволяє вказати кількість людей, що беруть участь у поїздці, включаючи дітей, дорослих та тварин. Після заповнення всіх полів користувач може натиснути кнопку для виконання пошуку квитків.

Для забезпечення зручності використання було реалізовано функцію toggleDirection, як наведено на рис. Б2 додатку Б, яка дозволяє користувачам змінювати місцями введені дані в полях "Куди" та "Звідки". Це корисно, коли користувачі помилково переплутали місця відправлення та прибуття. Функція спочатку зберігає значення поля "Звідки" у змінну cityFrom, потім оновлює значення цього поля значенням з поля "Куди" і навпаки. Після цього відповідні зміни вносяться в об'єкт fetchQuery, який використовується для формування запиту.

Функція findTrips, як наведено на рис. Б3 додатку Б, відповідає за виконання пошуку квитків на основі заданих параметрів. Вона робить асинхронний запит до сервера і отримує список доступних рейсів. Спочатку функція викликає метод fetchTrips з tripsStore, який робить запит до сервера для отримання даних про рейси. Якщо дані успішно отримані, створюється копія об'єкта fetchQuery, який містить параметри запиту. Потім виконується навігація до сторінки з результатами пошуку, передаючи параметри запиту через URL.

Щоб запобігти виконанню пошуку при наявності помилок у введених даних, було створено комп'ютед hasDirectionError, як наведено на рис. Б4 додатку Б, який визначає, чи є помилки у валідації даних. Цей комп'ютед перевіряє, чи є помилки в полях "Куди" або "Звідки". Якщо такі помилки є, значення hasDirectionError буде true, що дозволяє заблокувати кнопку пошуку до виправлення помилок.

Таким чином, реалізація пошуку квитків у веб-застосунку включає створення компоненту SearchForm, який об'єднує кілька підкомпонентів для забезпечення зручності введення параметрів пошуку. Функції для зміни місцями полів "Куди" та "Звідки", виконання пошуку квитків і блокування кнопки пошуку при наявності помилок забезпечують високу функціональність та зручність використання застосунку. Використання TypeScript допомагає забезпечити надійність та структурованість коду, зменшуючи кількість помилок та покращуючи підтримку і розвиток застосунку.

3.1.2. Робота з фільтрами

Фільтрація є одним із ключових аспектів веб-застосунку для он-лайн купівлі квитків на автобус. Вона дозволяє користувачам налаштовувати параметри пошуку відповідно до їхніх потреб, забезпечуючи зручність та ефективність використання застосунку. Розглянемо, як була реалізована фільтрація з використанням TypeScript та інших технологій.

Для забезпечення структурованості та уникнення помилок при використанні ключів фільтрації, було створено константу QUERY_KEY з усіма можливими ключами фільтрів. Це зроблено за допомогою перерахування (enum) у TypeScript, як наведено на рис. Б5 додатку Б.

Цей об'єкт служить для зберігання поточного стану фільтрів, що використовуються при пошуку квитків. Він містить усі можливі параметри, які можуть бути передані в запит для отримання даних.

Щоб отримати дані про рейси з урахуванням встановлених фільтрів, використовується асинхронний метод fetchTripDetails, який відправляє HTTP-запит до серверу з параметрами фільтрації, як наведено на рис. Б6 додатку Б.

Цей метод приймає ідентифікатор рейсу та об'єкт запиту, що містить параметри фільтрації, і виконує HTTP-запит до API. Використання Fetch API дозволяє отримати дані з сервера і обробити їх у клієнтському застосунку.

Розробка фільтрації у веб-застосунку для он-лайн купівлі квитків на автобус була здійснена з використанням TypeScript для визначення ключів фільтрації та створення структури даних, що зберігає стан фільтрів. Це забезпечує структурованість, зручність та уникнення помилок при роботі з параметрами фільтрації. Використання асинхронних запитів до API дозволяє отримувати актуальні дані з сервера відповідно до встановлених фільтрів, забезпечуючи зручність та ефективність використання застосунку.

3.1.3. Реалізація галереї в деталях квитку

У веб-застосунку для он-лайн купівлі квитків на автобус важливо забезпечити користувачам доступ до детальної інформації про рейс. Одним з елементів цієї інформації є галерея з фотографіями автобусів, яка дозволяє користувачам візуально оцінити транспорт. Для реалізації цієї галереї було використано бібліотеку Fancybox, яка забезпечує зручний та функціональний інтерфейс для перегляду зображень.

Галерея була реалізована за допомогою компонента Fancybox [12], який включає налаштування та відображення фотографій. Ось фрагмент коду, який відповідає за відображення галереї в деталях квитка, як наведено на рис. Б7 додатку Б.

Цей код створює галерею з фотографіями автобусів, де кожне фото відображається у вигляді посилання (<a>), що відкриває зображення у Fancybox при натисканні. Компонент <NuxtImg> використовується для відображення зображень, забезпечуючи оптимізацію завантаження та рендерингу.

Для налаштування поведінки та вигляду Fancybox використовуються спеціальні опції, визначені у вигляді об'єкта optionsFancy, як наведено на рис. Б8 додатку Б.

Для інтеграції Fancybox у веб-застосунок був створений окремий компонент, який забезпечує ініціалізацію та керування галереєю, як наведено на рис. Б9 додатку Б.

Цей компонент використовує бібліотеку Fancybox для створення галереї. Він приймає пропс options, що містить налаштування Fancybox. Компонент включає кілька життєвих циклів Vue, щоб керувати галереєю:

– onMounted: Ініціалізує Fancybox при монтуванні компонента, прив'язуючи його до елементу fancyGallery.

– onUpdated: Оновлює Fancybox при зміні компонента, відв'язуючи стару галерею, закриваючи її та знову прив'язуючи до елементу.

– onUnmounted: Знищує Fancybox при демонтуванні компонента, звільняючи ресурси.

Таким чином, реалізація галереї в деталях квитка дозволяє користувачам зручно переглядати фотографії автобусів. Інтеграція Fancybox забезпечує високий рівень зручності та інтерактивності, а використання TypeScript та Vue.js робить код структурованим і легко підтримуваним.

3.1.4. Реалізація сортування списку квитків

Реалізація сортування списку квитків у веб-застосунку є важливою функціональністю, яка дозволяє користувачам впорядковувати результати пошуку за різними критеріями, такими як ціна, час відправлення та інші параметри. Це забезпечує кращий користувацький досвід, дозволяючи швидко знаходити потрібні рейси.

У компоненті відображення списку квитків додано елемент інтерфейсу для вибору сортування. Цей елемент включає заголовок "Сортувати за" та випадаючий список, реалізований за допомогою компонента AppSelect, як наведено на рис. Б10 додатку Б.

Компонент AppSelect приймає список опцій для сортування sortFields, вибраний елемент getSelectedSort та обробник подій updateSort, який викликається при виборі опції.

Функція updateSort відповідає за обробку вибору опції сортування та оновлення списку квитків відповідно до вибраного критерію, як наведено на рис. Б11 додатку Б.

Ця функція виконує такі дії:

– Закриває модальне вікно вибору сортування за допомогою функції closeModal.

– Оновлює об'єкт fetchQuery, встановлюючи значення ключа SORT на id вибраної опції сортування.

– Викликає функцію fetchTrips для повторного завантаження списку квитків з новим критерієм сортування.

– Оновлює URL з новими параметрами запиту, використовуючи функцію updateURLQuery.

3.2. Інструкція користувачеві

При заході на сайт, користувача вітає головна сторінка, яка одразу містить форму для пошуку квитків. Користувачу потрібно заповнити всі поля даними.

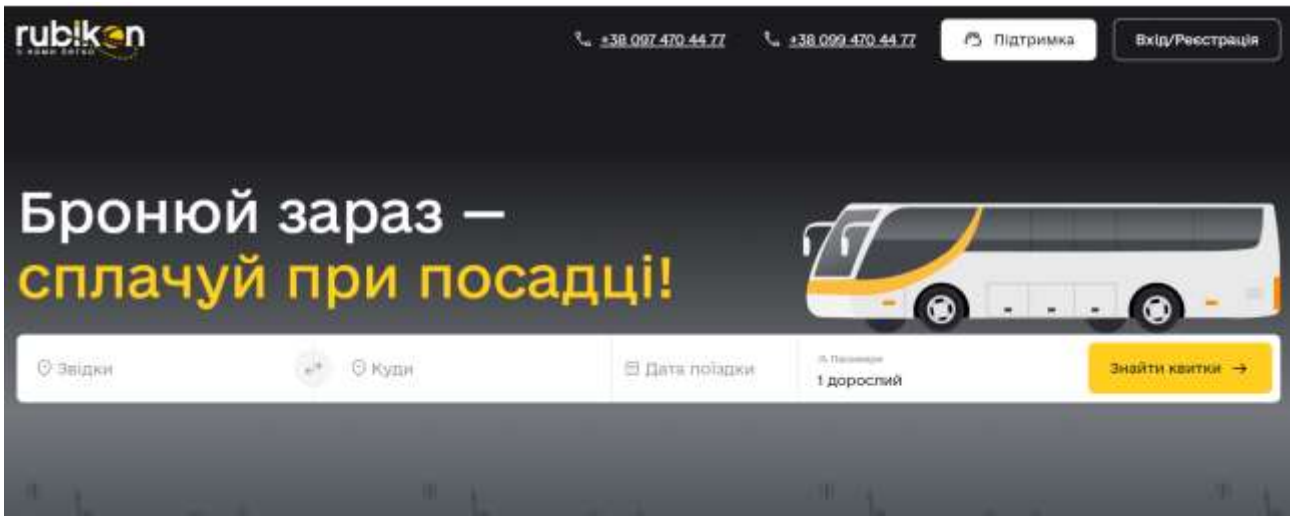

Рисунок 3.11 – Головна сторінка, форма для пошуку квитків

При введенні даних у поле "Звідки" та "Куди" відкриється віконечко з вибором міст та країн за пошуком. Користувачу потрібно натиснути та обрати місто та країну.

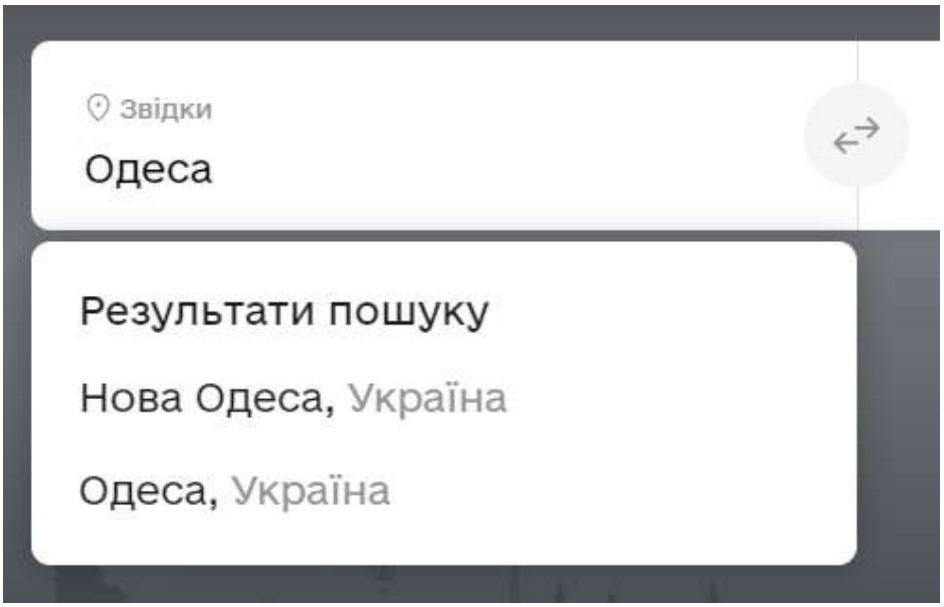

Рисунок 3.12 – Обрання «Звідки»

Натискаючи на календар, відкриється вікно календаря. Користувачеві потрібно просто натиснути на число, також є можливість обрати місяць та рік. Після натискання на дату, календар автоматично закриється та підставить обране значення в поле.

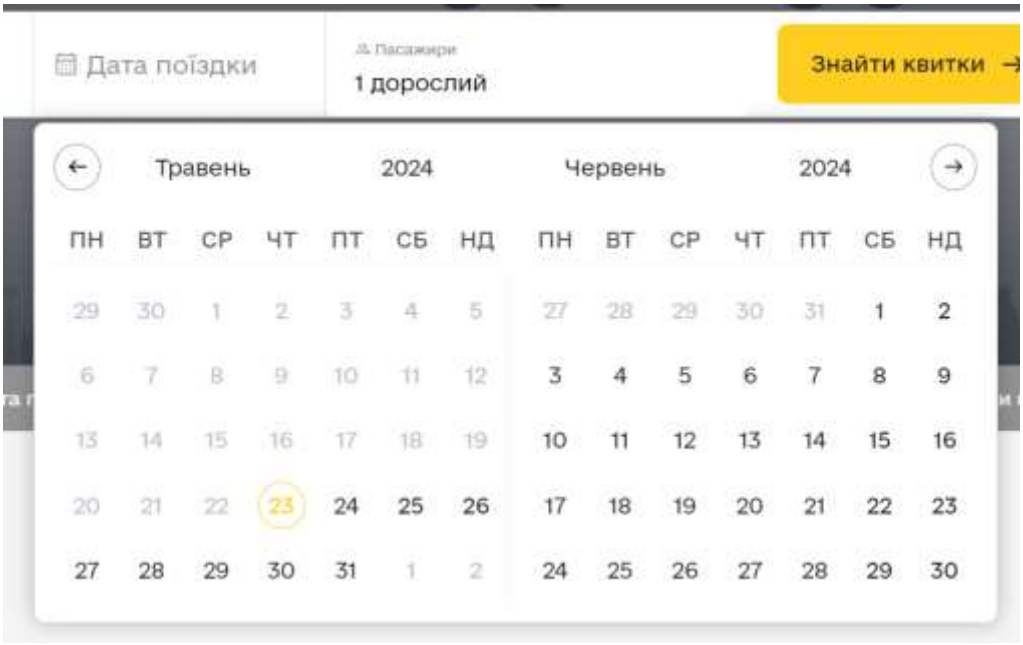

Рисунок 3.13 – Відкриття календаря

При натисканні на поле "Пасажири" відкривається вікно, де можна обрати кількість дорослих, дітей та тварин. Якщо обираєте дитину, то потрібно ввести в інпут вік дитини, а якщо тварину, то її вагу.

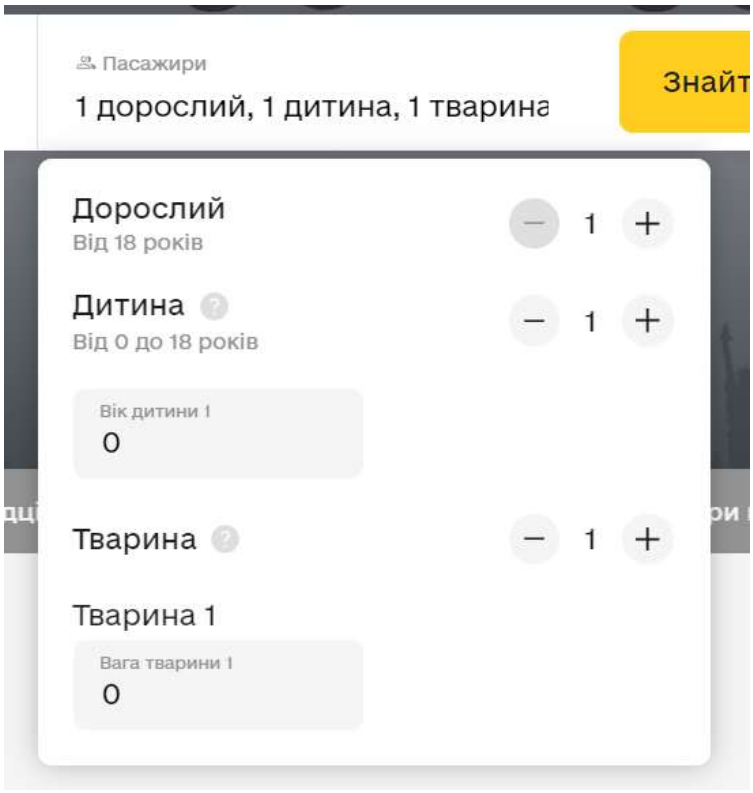

Рисунок 3.14 – Обрання пасажирів

Після заповнення всіх полів натискайте на кнопку "Знайти квитки". Є два варіанти розвитку подій:

1) Валідація не пройшла: Виводяться помилки під полями, де саме це сталось. Після виправлення помилок знову можна виконати пошук.

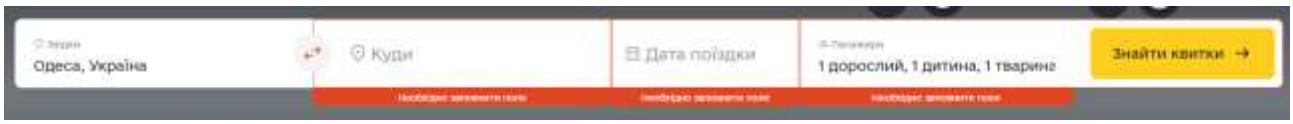

Рисунок 3.15 – Валідація не пройшла

2) Валідація пройшла успішно: Ви будете перенаправлені на сторінку з результатами пошуку.

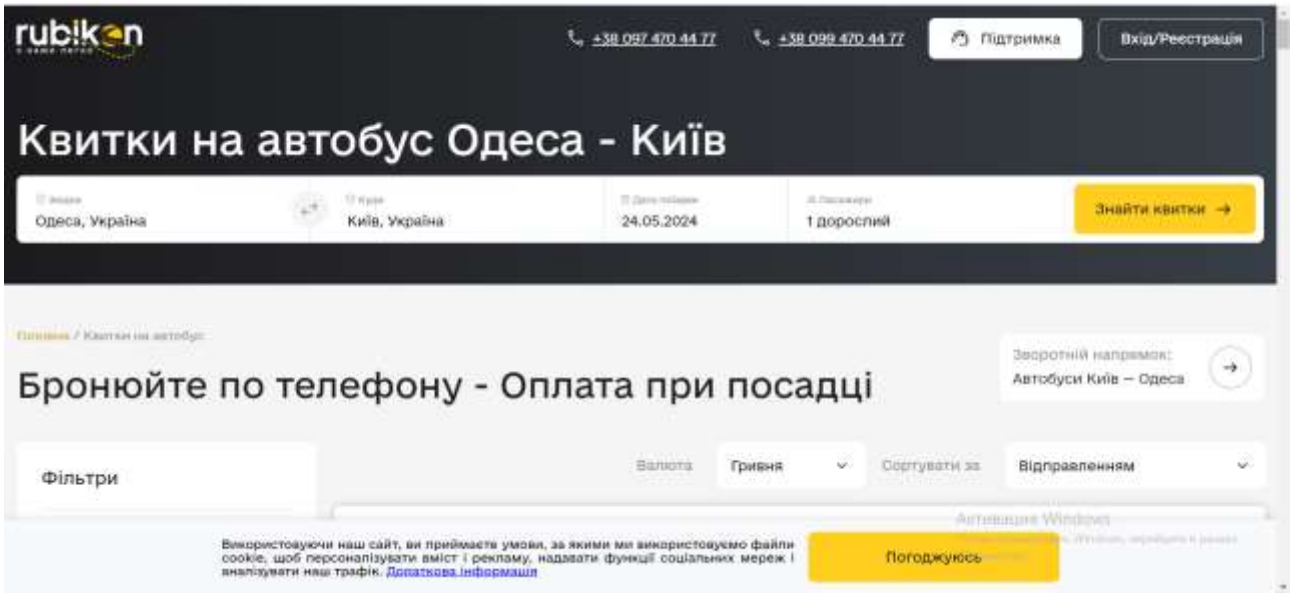

Рисунок 3.16 – Валідація пройшла успішно

На сторінці з результатами пошуку є бокова панель з фільтрами, де можна налаштувати додаткові параметри пошуку. Основна частина сторінки містить вибір сортування та список квитків. Є два варіанти відображення:

1) Список квитків: Висвітлюється список доступних квитків.

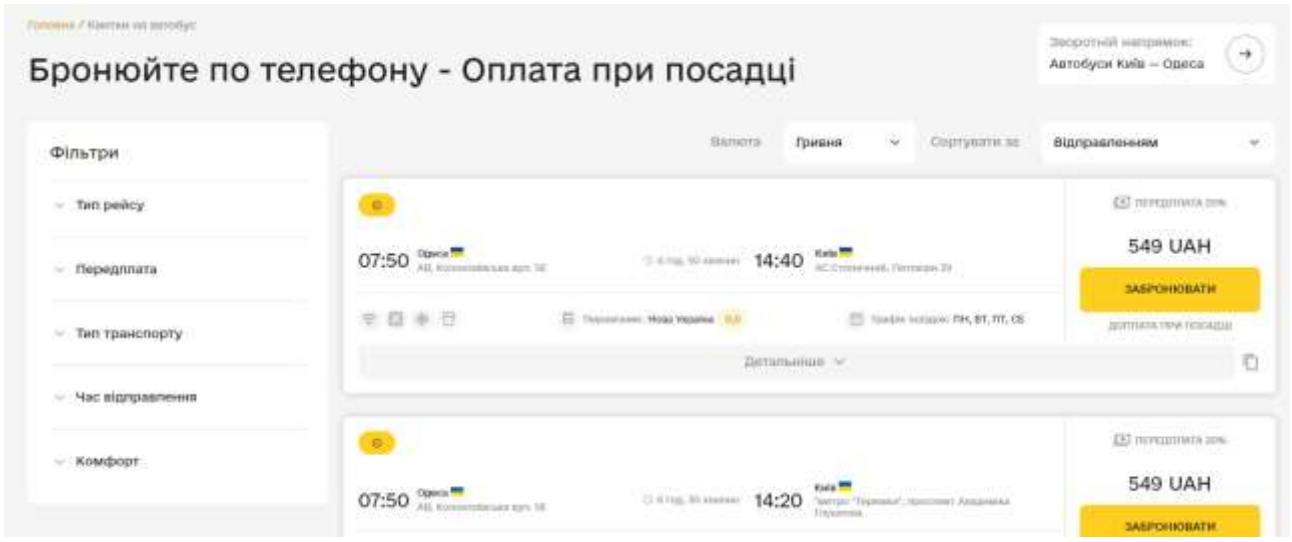

Рисунок 3.17 – Список квитків

2) Нічого не знайдено: Виводиться повідомлення, що на обрану дату нічого не знайдено, з можливістю перегляду рейсів на найближчі дати.

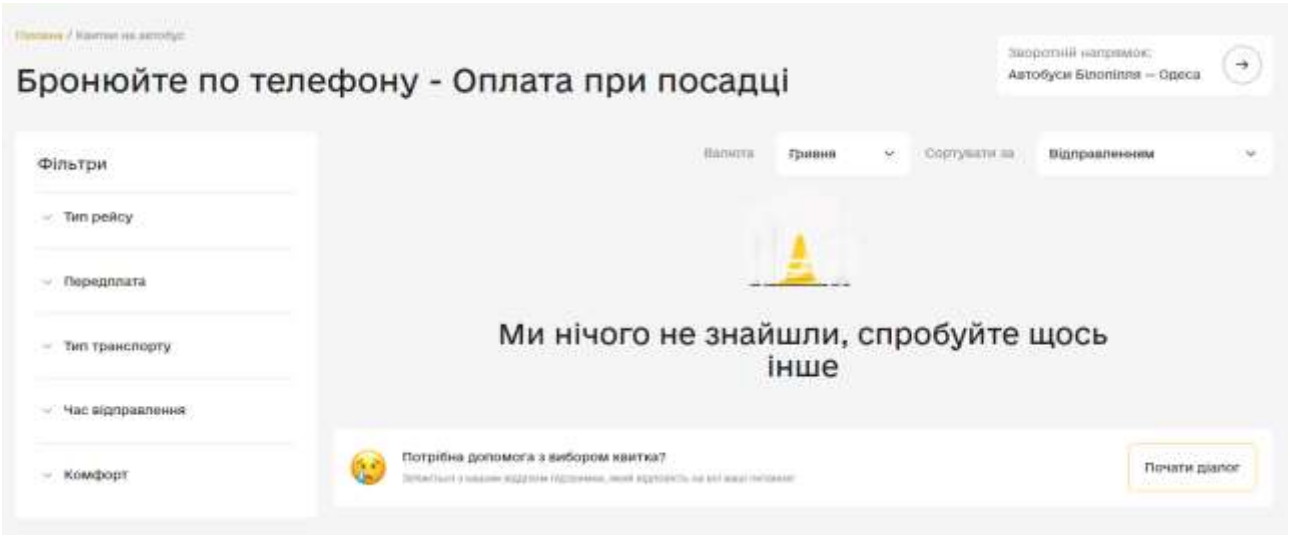

Рисунок 3.18 – Нічого не знайдено

Натискаючи на кнопку "Детальніше" на квитку, ви можете дізнатися більше про рейс. Відкривається вікно з інформацією про рейс, де можна натискати таби та переглядати цікаву інформацію.

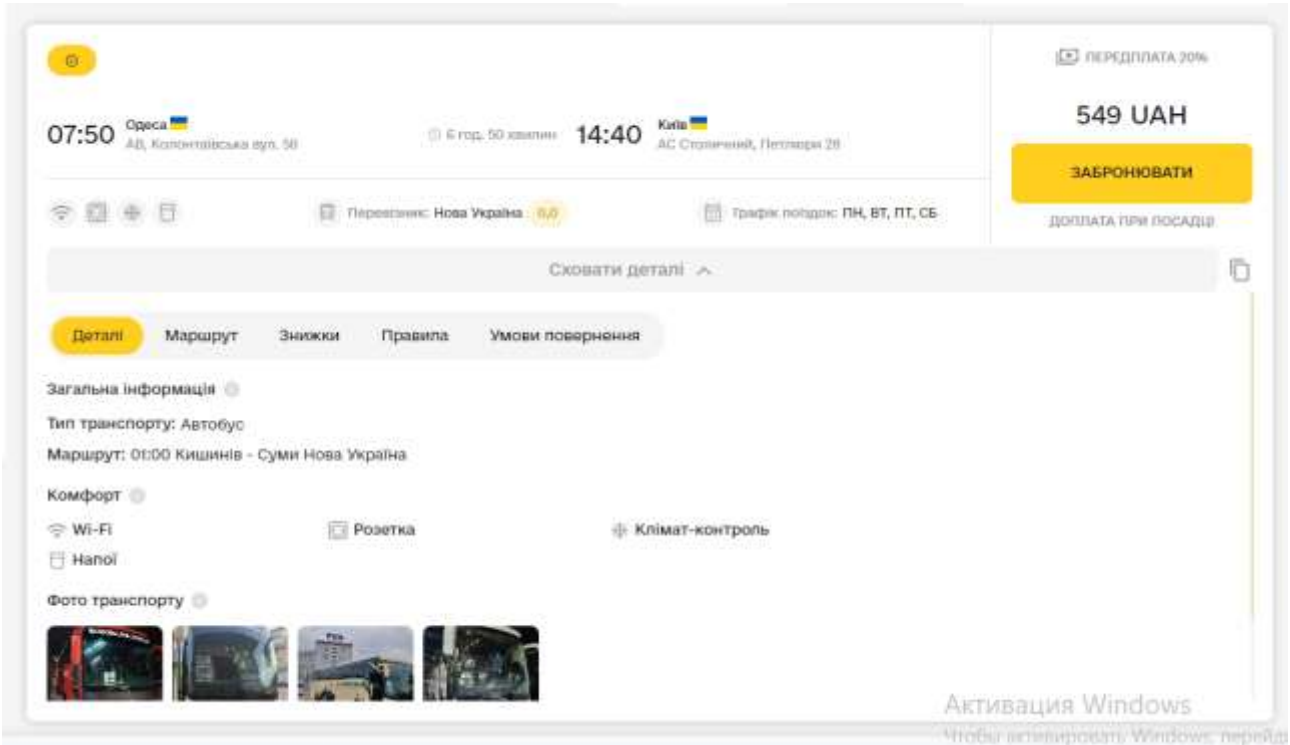

Рисунок 3.19 – Вікно «Детальніше»

На вкладці "Деталі" ви можете натискати на зображення автобуса. Це відкриває галерею, де можна переглядати зображення в більшому розмірі та листати їх.

Рисунок 3.20 – Галерея

На сторінці з результатами пошуку є бокова панель з фільтрами. Щоб використовувати фільтри, достатньо натиснути на шапку з назвою фільтра, що розкриє список з чекбоксами. Натискаючи на чекбокс, автоматично підтягнуться дані з обраним фільтром.

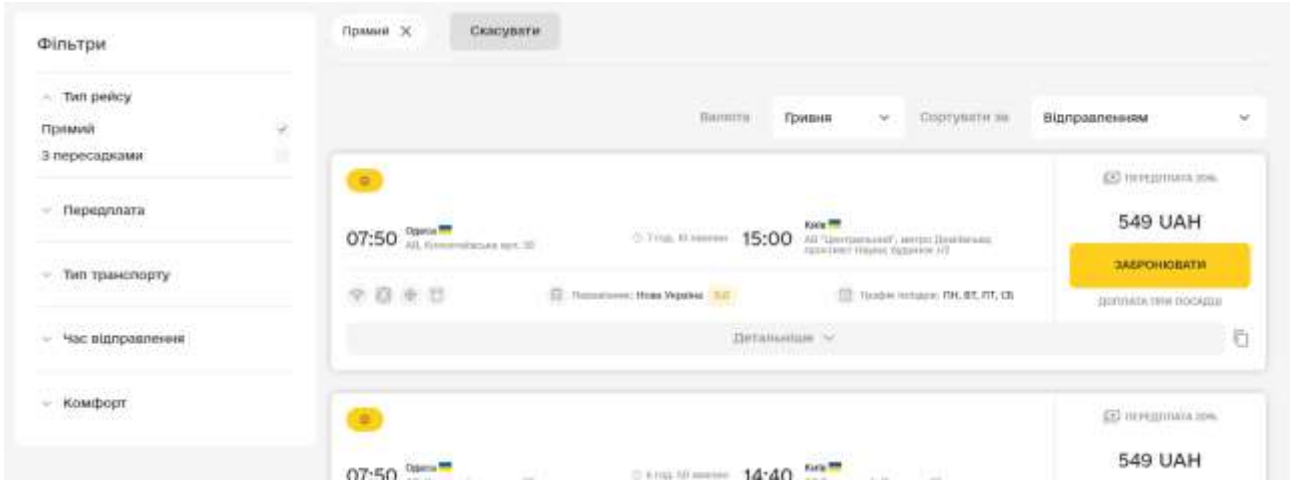

Рисунок 3.21 – Обрання фільтрів

Натискаючи на "Сортувати за", відкривається випадаючий список (селект) з вибором параметрів сортування. Потрібно просто обрати критерій сортування, натиснути на нього, і селект автоматично закриється, а дані оновляться відповідно до обраного сортування.

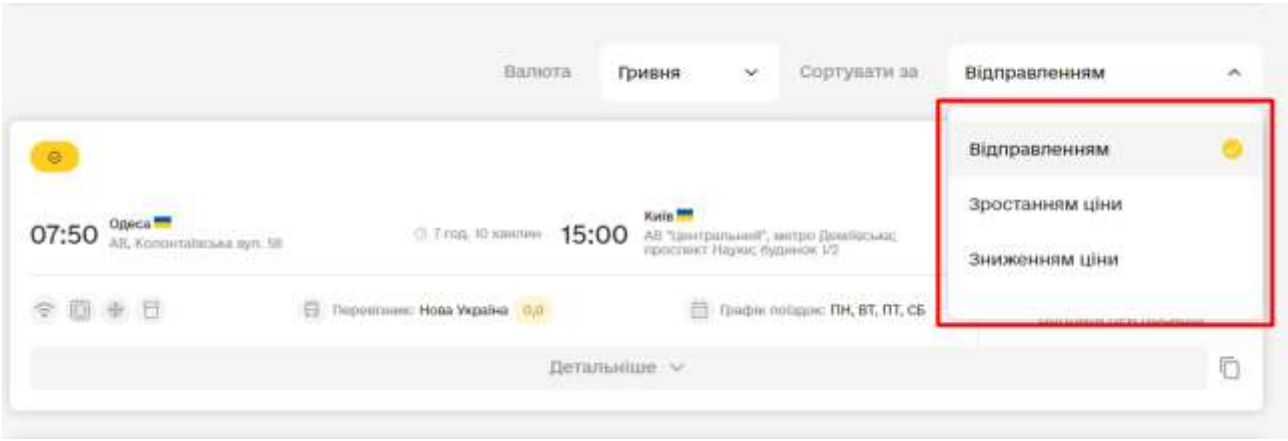

Рисунок 3.22 – Вибір сортування

Для того, щоб скинути усі фільтри, достатньо натиснути на «Скасувати», сторінка оновиться з актуальними даними та скине усі фільтри.

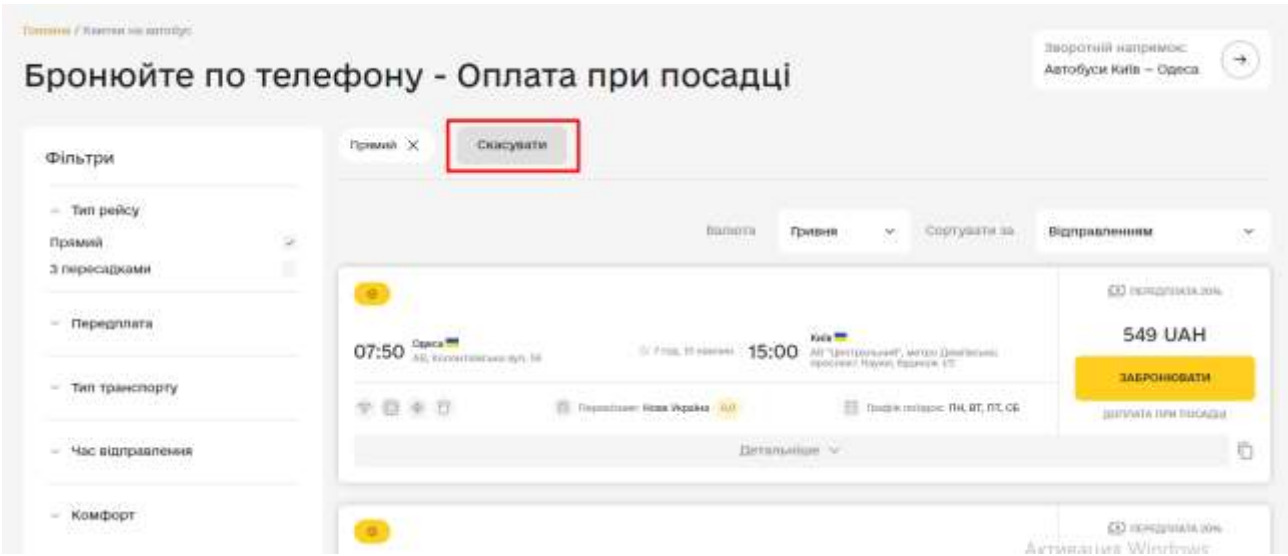

Рисунок 3.23 – Скасування усіх фільтрів

Для того, щоб скинути один фільтр, потрібно натиснути на таб з назвою фільтра того, який хочете скинути.

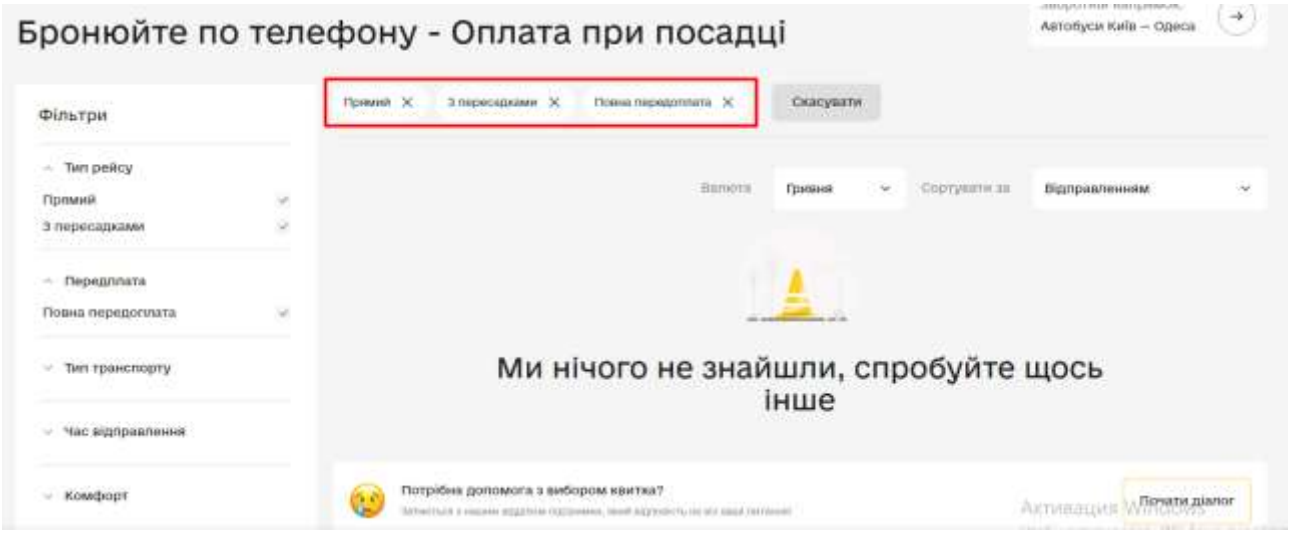

Рисунок 3.24 – Скасування певного фільтра

Дотримуючись цих кроків, можливо максимально ефективно користуватися веб-застосунком для пошуку та купівлі квитків на автобус.

3.3. Контрольне тестування та аналіз результатів

Контрольний приклад – важлива частина розробки. На цьому етапі потрібно перевірити працювання програми, перевірити поведінку при помилках.

Відкриваємо головну сторінку, яка має форму для пошуку квитків.

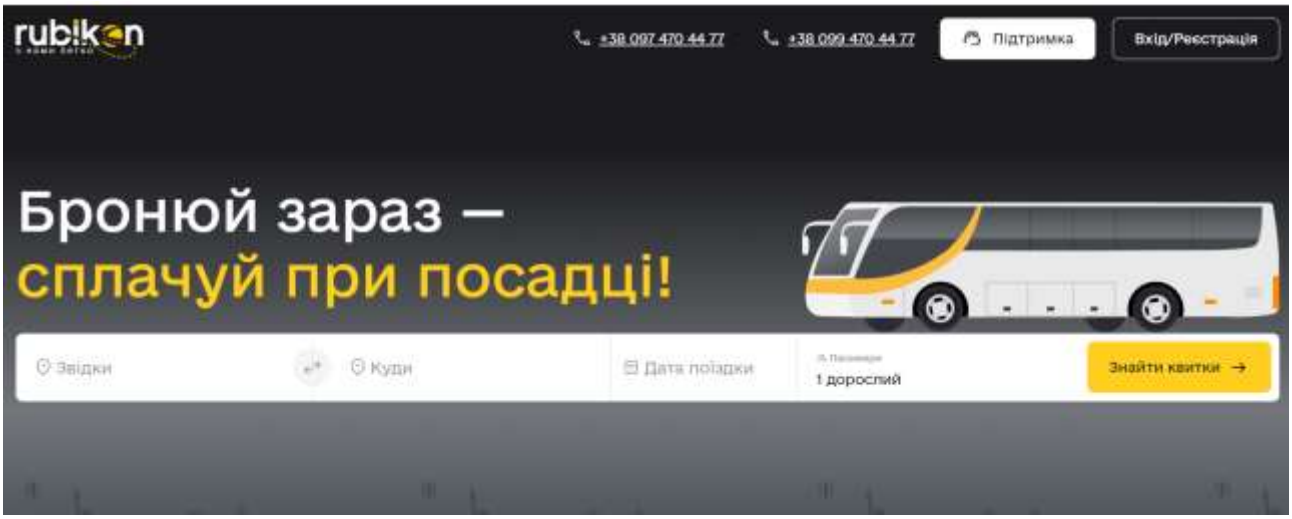

Рисунок 3.25 – Відкриття головної сторінки

Робимо пошук по полю «Звідки» та вводимо Одесу, подивимось на коректність пошуку:

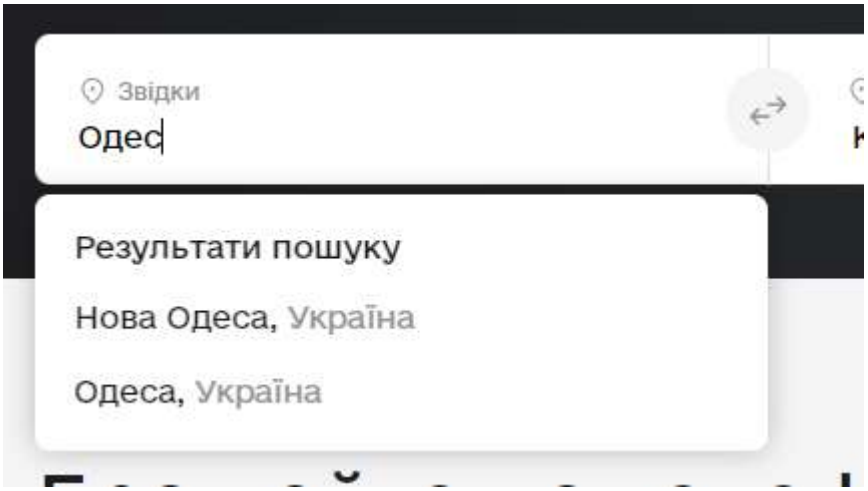

Рисунок 3.26 – Перевірка роботи пошуку за полем «Звідки»

З результату, бачимо що пошук працює, запит на пошук летить та відповідь із серверу прилітає.

Спробуємо відправити форму, не заповнивши усі поля:

Рисунок 3.27 – Спроба знаходження квитків без успішної валідації

З результату видно, що валідація у формі пошуку працює бездоганно, помилки виводяться. Тепер введемо коректні дані:

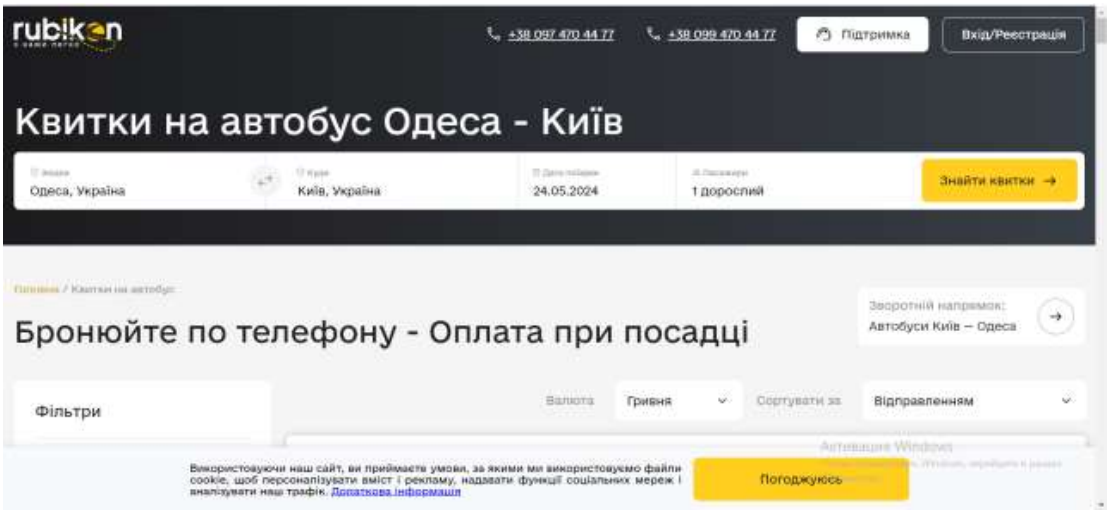

Рисунок 3.28 – Спроба пошуку квитків з успішною валідацією

Після натискання на кнопку «Знайти квитки», відбулось перенаправлення на сторінку з результатами пошуку

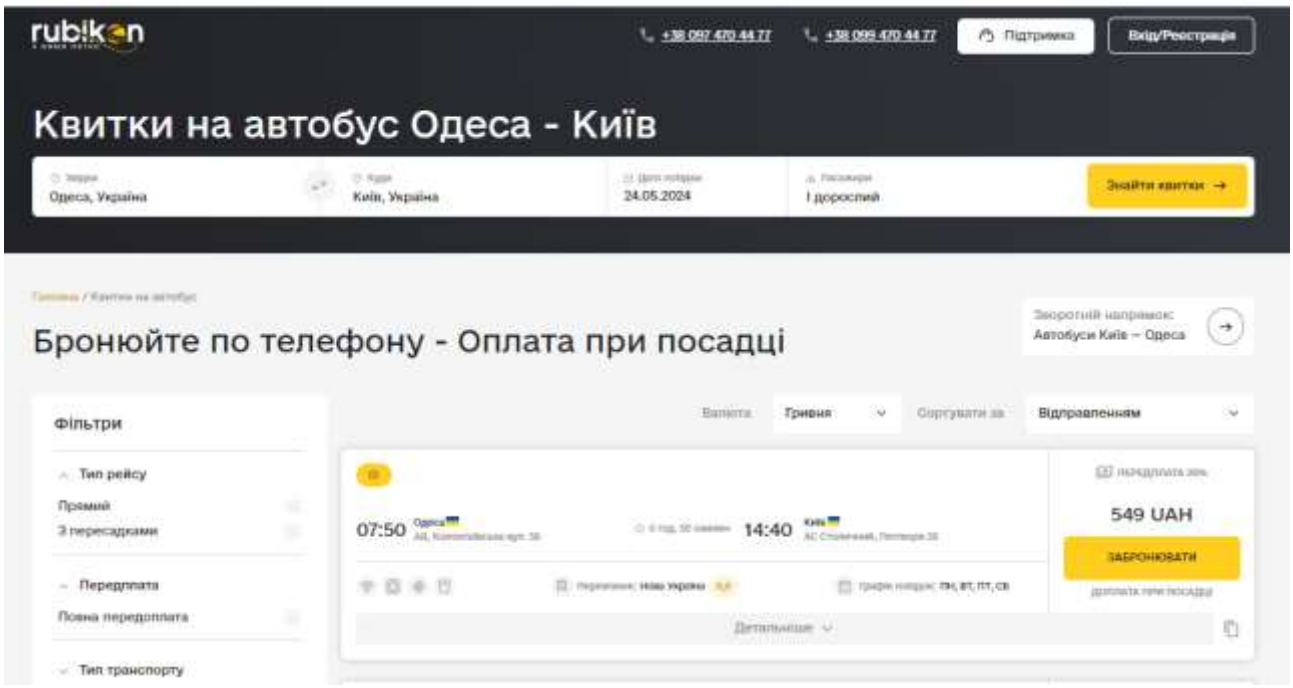

Рисунок 3.29 – Сторінка з результатами пошуку

Оберемо іншу дату поїздки:

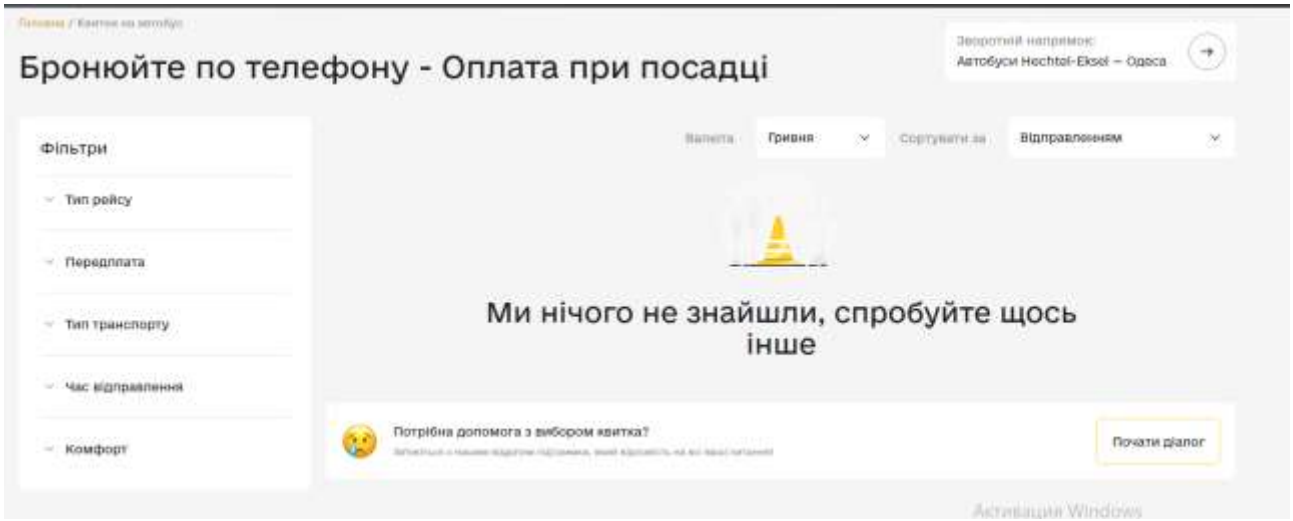

Рисунок 3.30 – Нічого не знайдено

Виводиться повідомлення, що на обрану дату нічого не знайдено.

Натискаємо на кнопку "Детальніше" на будь-якому квитку. Відкривається вікно з детальною інформацією про рейс.

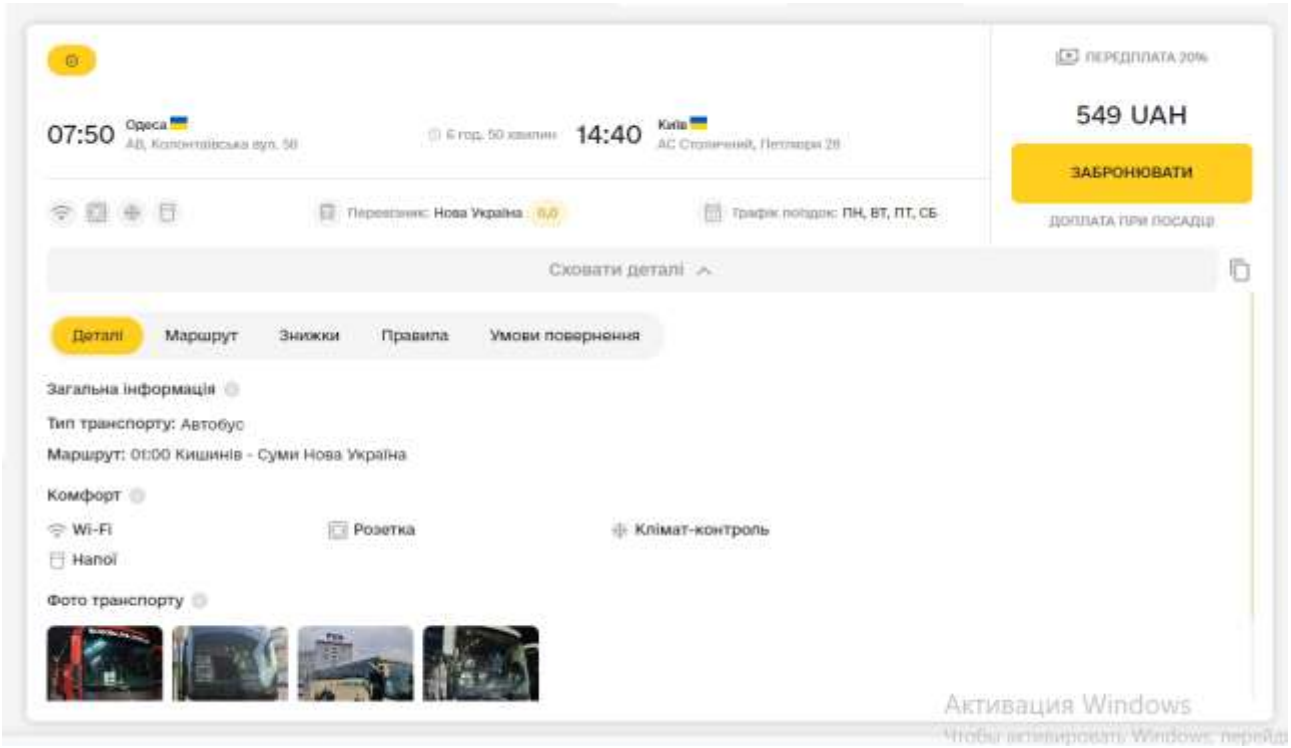

Рисунок 3.31 – Детальна інформація по квитку

Натискаємо "Деталі" відкривається інформація, внизу натиснемо на зображення будь-яке. Відкривається галерея, що дає можливість переглянути зображення у більшому розмірі. Галерея працює успішно.

Рисунок 3.32 – Відкриття галереї

Перевіримо роботу обрання фільтрів. Натискаємо на будь-який фільтр. Бачимо, що фільтр примінився, дані оновились, також з’явилась таба з назвою обраного фільтра. Фільтрація працює успішно.

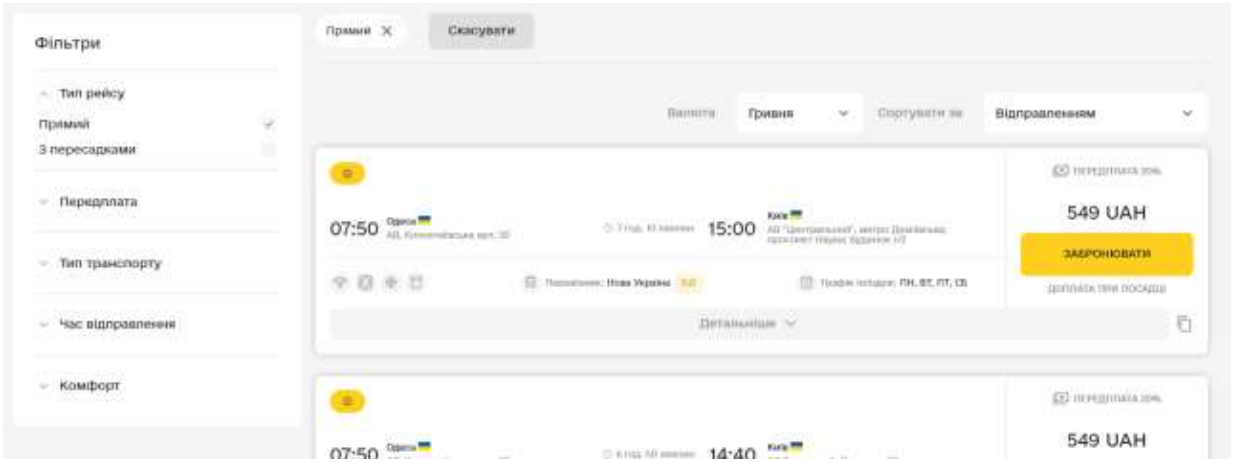

Рисунок 3.33 – Перевірка роботи фільтрів

Скинути фільтр можна натиснувши повторно у боковій панелі з фільтрами на той фільтр або натиснути на табу з фільтром. Натискаємо на табу, та бачимо, що фільтр більше не застосовується.

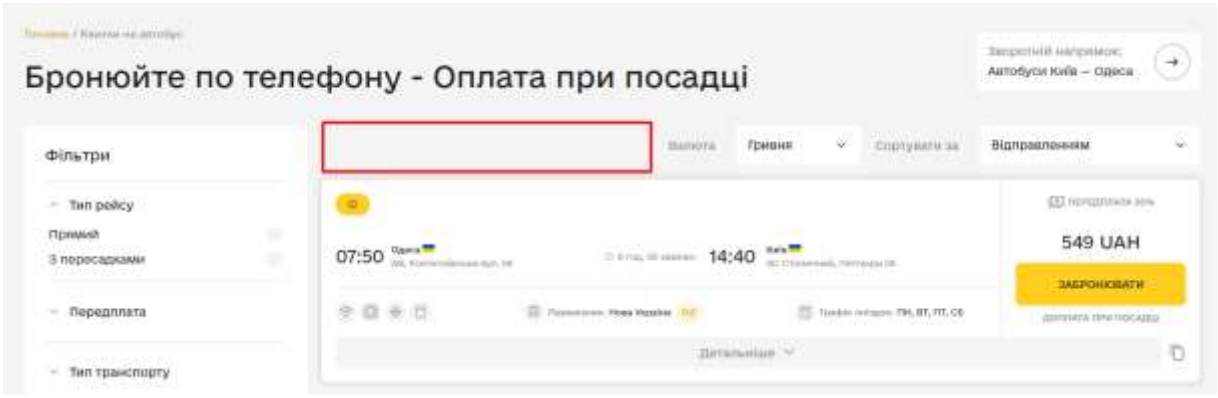

Рисунок 3.34 – Скасування певного фільтра

Оберемо знов кілька фільтрів та скинемо їх за один клік, достатньо натиснути на «Скасувати», сторінка оновиться з актуальними даними та скине усі фільтри.

Рисунок 3.35 – Скасування усіх фільтрів

Перевіримо як застосовується сортування (див. рис. 3.22), натиснемо на будь-який інший вид сортування. Оберемо спочатку сортування за зростанням ціни (див. рис. 3.36), а потім оберемо за спаданням ціни (див. рис. 3.37). Як бачимо з рисунків, сортування працює, дані оновлюються.

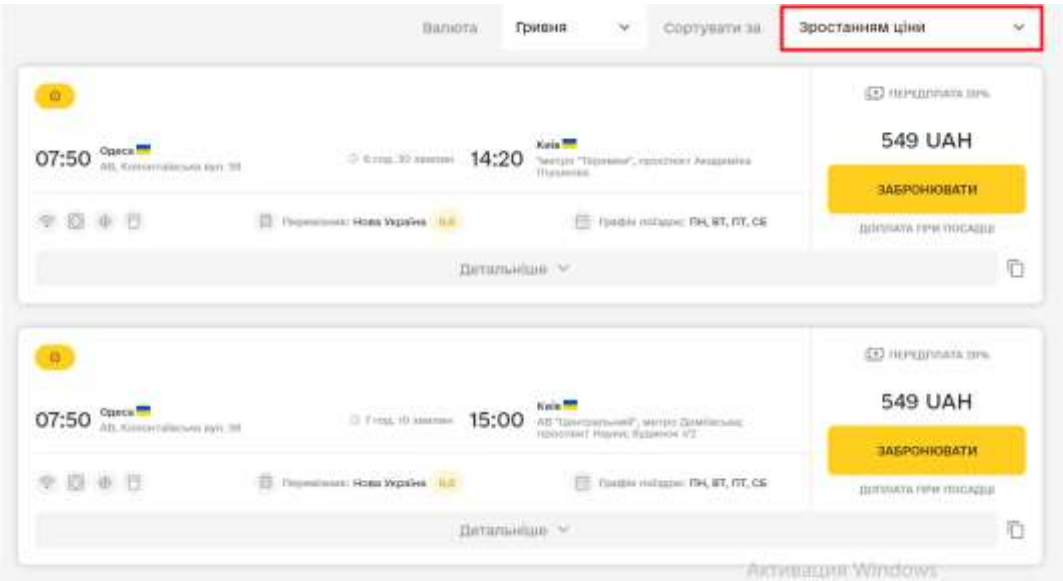

Рисунок 3.36 – Сортування за зростанням ціни

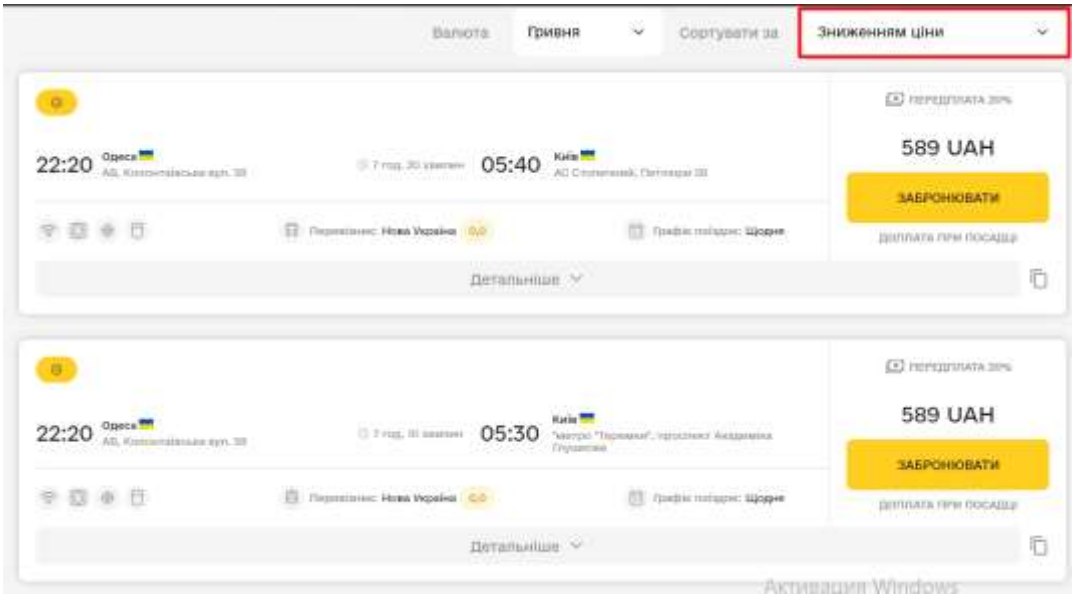

Рисунок 3.37 – Сортування за зниженням ціни

Було проведено тестування веб-застосунку, яке підтвердило її коректну роботу. Всі логічні аспекти функціонування були збережені, всі функціональні завдання виконані, а механізми обробки помилок успішно реалізовані.

3.4. Висновки до третього розділу

Для покращення зручності користування та розширення функціональних можливостей веб-застосунку для он-лайн купівлі квитків на автобус, пропонується впровадити такі вдосконалення:

1) Мульти-мовна підтримка:

– Реалізувати інтерфейс, що підтримує кілька мов, для зручності користувачів з різних країн.

– Автоматично визначати мову браузера користувача та встановлювати її як мову за замовчуванням.

2) Інтерактивні карти:

– Інтегрувати інтерактивну карту для візуалізації маршрутів, що дозволить користувачам бачити точні місця зупинок та маршрути в реальному часі.

– Додати можливість прокладання маршрутів до найближчих зупинок громадського транспорту.

3) Функціональність для частих пасажирів:

– Ввести систему лояльності та знижок для постійних пасажирів, що допоможе залучити більше постійних клієнтів.

– Додати можливість зберігати улюблені маршрути та отримувати персоналізовані пропозиції.

4) Розширені опції оплати:

– Інтегрувати додаткові платіжні системи, включаючи криптовалюти, для зручності користувачів.

– Ввести можливість оплати через мобільні додатки та цифрові гаманці.

Впровадження цих покращень дозволить зробити веб-застосунок більш зручним та функціональним для користувачів, підвищуючи його конкурентоспроможність на ринку транспортних послуг.

ВИСНОВКИ

У ході виконання роботи було розглянуто та вирішено проблему автоматизації процесу купівлі квитків на автобус у системі автобусних перевезень. Метою роботи було створення веб-застосунку, що забезпечує зручний і швидкий спосіб придбання квитків он-лайн, оптимізуючи роботу транспортних компаній та підвищуючи ефективність обслуговування клієнтів.

У першому розділі роботи було проведено детальний аналіз сучасних технологій для розробки клієнтської частини веб-застосунків. Було розглянуто популярні фреймворки, такі як Vue.js, React та Angular, кожен з яких має свої унікальні переваги та недоліки. Після ретельного аналізу було обрано Vue.js завдяки його простоті у вивченні, високій модульності та гнучкості. Vue.js дозволяє швидко створювати як окремі компоненти, так і масштабні односторінкові застосунки. Також було обґрунтовано вибір TypeScript для забезпечення надійності коду та зменшення кількості помилок завдяки суворій типізації. Для ефективного управління станом додатку було обрано бібліотеку Pinia, яка забезпечує зручну та інтуїтивно зрозумілу роботу зі станом. Використання цих технологій дозволило створити структурований та стабільний фундамент для розробки веб-застосунку.

Окрім вибору технологій, було проведено аналіз існуючих методів розробки веб-застосунків, включаючи підходи до проектування архітектури системи, принципи побудови користувацького інтерфейсу та методи забезпечення безпеки. Це дозволило створити комплексне рішення, яке відповідає сучасним стандартам якості та безпеки.

У другому розділі було розглянуто проектування веб-застосунку та розробку інтерфейсу користувача. Було детально описано процес проектування логіки програми, визначення її функцій, розбиття на компоненти, обробку помилок та використання типів даних. Реалізація інтерфейсу включала створення форми пошуку квитків, бокової панелі фільтрації, компонента для

відображення деталей квитка та галереї зображень. Цей етап роботи забезпечив високу функціональність та зручність використання веб-застосунку.

У третьому розділі було детально розглянуто та реалізовано основні функціональні можливості веб-застосунку. Це включало реалізацію пошуку квитків, функціональності сортування та фільтрації результатів пошуку, а також інтеграцію галереї зображень автобусів. Форма пошуку квитків була створена з урахуванням зручності користувачів, включаючи функціональність для вибору місця відправлення та прибуття, дати поїздки та кількості пасажирів. Функціональність сортування та фільтрації дозволяє користувачам налаштовувати результати пошуку відповідно до їхніх потреб. Інтеграція галереї зображень автобусів забезпечує візуальне представлення транспорту, що покращує користувацький досвід.

Крім того, було реалізовано обробку помилок та валідацію введених даних, що забезпечує стабільну роботу застосунку та запобігає некоректним діям з боку користувача. Це включає перевірку правильності введених даних, повідомлення про помилки та їх виправлення в режимі реального часу.Усі етапи розробки веб-застосунку тісно пов'язані з комп'ютерними науками. Використання мов програмування JavaScript та TypeScript, фреймворків Vue.js та бібліотек для управління станом (Pinia) та взаємодії з API (Fetch API) демонструє практичне застосування знань з програмування та розробки програмного забезпечення. Аналіз вимог, проектування архітектури та реалізація компонентів веб-застосунку ілюструють важливість системного аналізу та проектування в комп'ютерних науках. Розробка зручного та інтуїтивного інтерфейсу користувача, тестування та забезпечення якості програмного забезпечення підкреслюють значення людино-комп'ютерної взаємодії та методів забезпечення якості.

В рамках роботи було запропоновано кілька удосконалень для подальшого розвитку веб-застосунку, таких як підтримка кількох мов, інтерактивні карти, система лояльності, розширені опції оплати. Впровадження цих покращень дозволить зробити веб-застосунок більш зручним та функціональним для

користувачів, підвищуючи його конкурентоспроможність на ринку транспортних послуг.

У кваліфікаційній роботі були реалізовані наступні компетентності та результати навчання: здатність до абстрактного мислення, аналізу та синтезу, здатність вчитися і оволодівати сучасними знаннями, здатність до пошуку, оброблення та аналізу інформації з різних джерел, здатність розробляти архітектури, модулі та компоненти програмних систем, здатність ідентифікувати, класифікувати та формулювати вимоги до програмного забезпечення, здатність обґрунтовано обирати та освоювати інструментарій з розробки та супроводження програмного забезпечення, знати та вміти застосовувати інформаційні технології обробки, зберігання та передачі даних, знати та вміти застосовувати методи верифікації та валідації програмного забезпечення, аналізувати, цілеспрямовано шукати і вибирати необхідні для вирішення професійних завдань інформаційно-довідникові ресурси і знання з урахуванням сучасних досягнень науки і техніки, знати та вміти застосовувати методи та засоби управління проектами.

Таким чином, робота охопила всі аспекти розробки сучасного веб-застосунку, від аналізу та проектування до реалізації та тестування. Виконана робота демонструє важливість інтеграції теоретичних знань з комп'ютерних наук з практичними навичками, забезпечуючи створення надійного та ефективного інструменту для користувачів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Busfor. URL: https://busfor.ua/?utm_source=google&utm_medium=cpc&comuto_cmkt=&gad_source=1&gclid=Cj0KCQjwu8uyBhC6ARIsAKwBGpQ2cBsOsde1j8ELUdFrOa2pfi91lEAfShP-F6vhf0RmSiybVFxxS0kaAoLeEALw_wcB

2. Ecolines. URL: https://ecolines.net/ua/uk

3. BlaBlaCar. URL: https://www.blablacar.com.ua/

4. Порівняльний аналіз популярних JavaScript - фреймворків та бібліотек для front-end розробки. URL: https://openarchive.nure.ua/server/api/core/bitstreams/3daaef14-8941-4239-b28a-815c46e2e73c/content

5. Порівнюємо React, Angular і Vue — найпопулярніші бібліотеки й фреймворки у 2022 році. URL: https://dou.ua/forums/topic/39933/

6. Використання TypeScript в розробці веб-сайтів. URL: https://galaktica.io/blog/typescript-tse/

7. Недоліки TypeScript. URL: https://www.gen.tech/post/typescript-vid-hajpu-do-standartu-rozrobki#:~:text=%D0%9D%D0%B5%D0%B4%D0%BE%D0%BB%D1%96%D0%BA%D0%B8%20TypeScript%3A&text=%D0%94%D0%BE%D0%B2%D1%88%D0%B8%D0%B9%20%D1%87%D0%B0%D1%81%20%D1%80%D0%BE%D0%B7%D1%80%D0%BE%D0%B1%D0%BA%D0%B8.,%D1%86%D0%B5%20%D0%BC%D0%BE%D0%B6%D0%B5%20%D1%83%D1%81%D0%BA%D0%BB%D0%B0%D0%B4%D0%BD%D0%B8%D1%82%D0%B8%20%D1%88%D0%B2%D0%B8%D0%B4%D0%BA%D1%83%20%D1%96%D1%82%D0%B5%D1%80%D0%B0%D1%86%D1%96%D1%8E.

8. Управління станом веб-застосунку за допомогою Pinia. URL: https://pinia-ua.netlify.app/introduction.html

9. CRUD. URL: https://highload.today/uk/shho-take-crud-prostimi-slovami-funktsiyi-perevagi-ta-prikladi/

10. Знайомство з Visual Studio Code. URL: https://romul.name/blog/znayomstvo-z-visual-studio-code/

11. Документація з Vue.js. URL: https://ua.vuejs.org/guide/quick-start.html

12. Галерея Fancybox. URL: https://fancyapps.com/fancybox/

13. Лістинг програми. URL: https://gitlab.com/SamiraRadzhabova/diplom_radzhabova

ДОДАТОК А

pages/[buses].vue

```
<script setup async lang="ts">
// Components
import TripsFilter from "~/components/trips/TripsFilter.vue";
import HeroSection from "~/components/main-page/hero/LazyHeroSection.vue";
import EmptyLayout from "~/components/PersonalCabinet/EmptyLayout.vue";
import AppButton from "~/components/UI/AppButton.vue";
import AppSelect from "~/components/UI/AppSelect.vue";
import Trip from "~/components/trips/Trip/Trip.vue";
import Rates from "~/components/rates/index.vue";
// Types
import { ICurrencyOption, ISelectOption } from "~/types";
import {
  ITripsDeparturesDate,
  INearestDateTrip,
  TripModalFlagsKeys,
  TripModalFlags,
  IFilterPayload,
  ITrip,
} from "~/types/trips";
// Stores
import { useTripsStore } from "~/stores/trips";
// Constants
import { QUERY_KEY } from "~/constants/trips";
import { IFetchQuery } from "~/types/api";
import AppBreadcrumb from "~/components/UI/AppBreadcrumb.vue";
import AppBtnBack from "~/components/UI/AppBtnBack.vue";
// Composables
```

```
const tripsStore = useTripsStore();
const route = useRoute();
// Data
const { SORT, CURRENCY_ID } = QUERY_KEY;
// Store actions
const {
  getCurrenciesNameConcat,
  getSelectedCurrency,
  getSelectedFilters,
  getRecountedPrice,
  getSelectedSort,
  getActiveDate,
} = toRefs(tripsStore);
const {
  resetSearchValidationErrors,
  fetchTripsNearest,
  fetchTripDetails,
  fetchSEOTrips,
  parseUrlQuery,
  resetFilters,
  updateQuery,
  fetchTrips,
} = tripsStore;
// Store data
const { sortFields } = toRefs(tripsStore.filters);
const { trips, fetchQuery, pendingTripsFetch, pendingTripsMore, search } =
  toRefs(tripsStore);
// Async data
// -- merging query from the URL to the app store
fetchQuery.value = deepMerge(fetchQuery.value, route.query);
```

```
parseUrlQuery(route.query as IFetchQuery);
const { error } = await fetchSEOTrips(route.params.from, route.params.to);
// Flags
const isActiveModal: TripModalFlags = reactive({
  currency: false,
  filter: false,
  price: false,
  sort: false,
});
const prevWidth = ref(0);
// Computed data
const tripsFounded = computed(() => {
  return trips.value.departures.data.length;
});
const isMoreTrips = computed(() => {
  return !!trips.value.departures.next_page_url;
});
const isNearestDatesTrips = computed(() => {
  let isTrips = false;
  trips.value.nearestDates.forEach((nearestDate) => {
    if (nearestDate.departures.data.length) {
      isTrips = true;
    }
  });
  return isTrips;
});
const onlyOneTripFounded = computed(() => {
  return trips.value.departures.data.length === 1;
});
const emptyLayoutText = computed(() => {
```

```
  return isNearestDatesTrips.value
    ? "Ми не знайшли квитки на цей день, але маємо декілька варіантів у інші
дати:"
    : "Ми нічого не знайшли, спробуйте щось інше";
});
const pageHeadTitle = computed(() => {
  const { toCity, fromCity } = search.value;
  return `Rubikon - Пошук рейсів ${fromCity.selected?.name || ""}-${
    toCity.selected?.name || ""
  }`;
});
const mobileCurrencyBtn = computed(() => {
  return `${getSelectedCurrency.value?.name},
${getSelectedCurrency.value?.code}`;
});
// Methods
function toggleModal(modalFlag: TripModalFlagsKeys) {
  isActiveModal[modalFlag] = !isActiveModal[modalFlag];
  isActiveModal[modalFlag] && window.innerWidth < 1023
    ? lockPageScroll()
    : unlockPageScroll();
}
function closeFilterModal() {
  if (window.innerWidth < 1023) {
    isActiveModal.filter = false;
    unlockPageScroll();
  }
}
function closeModal(modalFlag: TripModalFlagsKeys) {
  isActiveModal[modalFlag] = false;
```

```
  unlockPageScroll();
}
function updateURLQuery(currentQuery: object) {
  const query = JSON.parse(JSON.stringify(currentQuery));
  return navigateTo({
    path: "/trips",
    query,
  });
}
async function updateTrips(filterPayload: IFilterPayload) {
  updateQuery(filterPayload);
  updateURLQuery(fetchQuery.value);
}
async function getTripDetailsById(trip: ITrip) {
  const queryPayload = {
    departureDate: fetchQuery.value.departureDate,
    fromBusStopId: trip.startBusStop.id as number,
    toBusStopId: trip.endBusStop.id as number,
    currencyId: fetchQuery.value.currencyId,
  };
  const { data: tripDetails } = await fetchTripDetails(trip.id, queryPayload);
  if (tripDetails) trip.more = tripDetails.data;
}
async function resetAllFilters() {
  resetFilters();
  await fetchTrips(true, true);
  updateURLQuery(fetchQuery.value);
}
// Need to test!
async function fetchMoreNearestTrips(nearestTrips: INearestDateTrip) {
```

```
  if (!nearestTrips.departures.next_page_url) return;
  fetchQuery.value[QUERY_KEY.DEPARTURE_DATE] = nearestTrips.date.date;
  fetchQuery.value[QUERY_KEY.PAGE] = nearestTrips.departures.current_page +
1;
  await fetchTripsNearest();
}
async function getNearestTripDetails(trip: ITrip, tripDepartureDate: string) {
  const queryPayload = {
    departureDate: tripDepartureDate,
    fromBusStopId: trip.startBusStop.id as number,
    toBusStopId: trip.endBusStop.id as number,
    currencyId: fetchQuery.value.currencyId,
  };
  const { data: tripDetails } = await fetchTripDetails(trip.id, queryPayload);
  if (tripDetails) trip.more = tripDetails.data;
}
// Need to test!
async function fetchMoreTrips() {
  fetchQuery.value.page += 1;
  await fetchTrips(false);
}
async function updateCurrency(selectedSort: ICurrencyOption) {
  closeModal("currency");
  fetchQuery.value[CURRENCY_ID] = selectedSort.id as number;
  await fetchTrips(true, true);
  updateURLQuery(fetchQuery.value);
}
async function updateSort(selectedSort: ISelectOption) {
  closeModal("sort");
  fetchQuery.value[SORT] = selectedSort.id as string;
```

```
  await fetchTrips(true, true);
  updateURLQuery(fetchQuery.value);
}
async function onActiveDateChange(activeDate: ITripsDeparturesDate) {
  if (activeDate.active) return;
  // reset previous active status
  trips.value.dates.forEach((dateObj) => (dateObj.active = false));
  // set new date active status
  activeDate.active = true;
  search.value.departureDate = activeDate.date.date;
  fetchQuery.value[QUERY_KEY.DEPARTURE_DATE] = activeDate.date.date;
  // fetch new trips
  await fetchTrips(true, true);
}
// Lifecycle hooks
onMounted(() => {
  prevWidth.value = window.innerWidth;
  if (window.innerWidth >= 1024) isActiveModal.filter = true;
  window.addEventListener("resize", () => {
    if (prevWidth.value == window.innerWidth) return false;
    prevWidth.value = window.innerWidth;
    if (window.innerWidth >= 1024) {
      unlockPageScroll();
      isActiveModal.filter = true;
    } else if (window.innerWidth < 1023) {
      isActiveModal.filter = false;
    }
  });
});
onUnmounted(() => {
```

```
  resetSearchValidationErrors();
});
// Page meta, etc.
useHead({
  title: pageHeadTitle,
});
</script>
```

PlaceModal.vue

```
<script setup lang="ts">
// ---- TYPES ---- //
import { ISearchCityItem } from "~/types"
// ---- COMPONENTS ---- //
import SearchModalLayout from
"~/components/App/Search/SearchModalLayout.vue"
import AppInput from "~/components/UI/AppInput.vue"
import AppLoaderSmall from "~/components/UI/AppLoaderSmall.vue"
// ---- DATA ---- //
const props = defineProps<{
  popularDirections?: ISearchCityItem[]
  recentlySearched?: ISearchCityItem[]
  selectedCity: ISearchCityItem | null
  searchQuery: string
  inputLabel: string
  pending: boolean
  cities: ISearchCityItem[]
}>()
const emit = defineEmits<{
  (e: "on-city-select", city: ISearchCityItem): void
  (e: "on-search-input", searchQuery: string): void
```

```
}>()
// Computed =>
const searchNotFound = computed(() => {
  return props.searchQuery && !props.cities.length
})
const searchPlaceholderText = computed(() => {
  return searchNotFound.value
    ? "Результатів пошуку не знайдено"
    : "Результати пошуку"
})
// ---- METHODS ---- //
function selectCity(selectedCity: ISearchCityItem) {
  emit("on-city-select", selectedCity)
}
function onSearchInput(searchQuery: string) {
  emit("on-search-input", searchQuery)
}
</script>
```

PassengersModal.vue

```
<script setup lang="ts">
// Components
import SearchModalLayout from
"~/components/App/Search/SearchModalLayout.vue"
import AppButton from "~/components/UI/AppButton.vue"
import AppInput from "~/components/UI/AppInput.vue"
import ModalItem from "~/components/App/Search/PassengersModalItem.vue"
// Constants
import { QUERY_KEY } from "~/constants/trips"
// Store
```

```
import { useTripsStore } from "~/stores/trips"
const tripsStore = useTripsStore()
// Store data
const { adults, animals, children } = toRefs(tripsStore.search.passengers)
const { fetchQuery, validationErrors } = toRefs(tripsStore)
// Local data
const emit = defineEmits<{
  (e: "on-confirm"): void
}>()
// Computed data
const showChildInputs = computed(() => {
  return !!children.value.ages.length
})
const showAnimalsInputs = computed(() => {
  return !!animals.value.weights.length
})
// Methods
function onPassengerDataUpdate() {
  validationErrors.value.passengers = false
}
// -- adults
function onAdultPassengerAdd() {
  adults.value.count += 1
}
function onAdultPassengerRemove() {
  if (adults.value.count === 1) return
  adults.value.count -= 1
}
// -- childrens
function onChildPassengerAdd() {
```

```
  children.value.ages.push({ age: 0 })
}
function onChildPassengerRemove() {
  if (children.value.ages.length === 0) return
  children.value.ages.pop()
  // reset query children
  if (children.value.ages.length === 0) {
    fetchQuery.value[QUERY_KEY.PASSENGERS].children = {
      count: undefined,
      ages: [],
    }
    return
  }
}
// -- animals
function onAnimalPassengerAdd() {
  animals.value.weights.push({ weight: 0 })
}
function onAnimalPassengerRemove() {
  if (animals.value.weights.length === 0) return
  animals.value.weights.pop()
  // reset query animals
  if (animals.value.weights.length === 0) {
    fetchQuery.value[QUERY_KEY.PASSENGERS].animals = {
      count: undefined,
      weights: [],
    }
    return
  }
}
```

```
function onSubmitPassengers() {
  emit("on-confirm")
}
</script>
```

SearchForm.vue

```
<script setup lang="ts">
// ---- TYPES ---- //
import { ISearchCityItem } from "~/types"
import { LocationQueryRaw } from "#vue-router"
// ---- COMPONENTS ---- //
import ModalTransition from "~/components/UI/Transitions/ModalTransition.vue"
import PassengersModal from "~/components/App/Search/PassengersModal.vue"
import AppDatepicker from "~/components/UI/AppDatepicker.vue"
import PlaceModal from "~/components/App/Search/PlaceModal.vue"
import AppButton from "~/components/UI/AppButton.vue"
import AppInput from "~/components/UI/AppInput.vue"
// ---- MODULES ---- //
import { useShowSearchPopup } from "~/composables/useShowSearchPopup"
import { useScreen } from "vue-screen"
// ---- STORES ---- //
import { useTripsStore } from "~/stores/trips"
// ---- CONSTANTS ---- //
import { QUERY_KEY } from "~/constants/trips"
// --- COMPOSABLES --- //
const screen = useScreen()
const modalSearchPlaceFrom = useShowSearchPopup()
const modalSearchPlaceTo = useShowSearchPopup()
const tripsStore = useTripsStore()
// -- stores data reactive
```

```
const {
  pendingTripsFetch,
  validationErrors,
  pendingSearchBar,
  fetchQuery,
  search,
} = toRefs(tripsStore)
const { resetSearchValidationErrors } = tripsStore
// ---- DATA ---- //
defineProps<{
  passengersInfo: string
}>()
const multiCalendars = ref() // Начальное значение для десктопа
const scrollMonth = ref(false)
const closeAuto = ref(false)
const headShow = ref(false)
// Computed data
const hasDirectionError = computed(() => {
  return validationErrors.value.city.to || validationErrors.value.city.from
})
// debounce input handlers =>
const onFromCitySearchDebounced = debounce(onFromCitySearch, 500)
const onToCitySearchDebounced = debounce(onToCitySearch, 500)
// ---- METHODS ---- //
const modalSearchPassenger = {
  ...useShowSearchPopup(),
}
function onFromCityFieldClick(input: Event) {
  const { fromCity } = toRefs(search.value)
  const { from } = toRefs(fetchQuery.value)
```

```
  // reset fields if there is already some value
  if (fromCity.value.query) {
    // reset search
    fromCity.value = {
      searchResults: [],
      selected: null,
      query: "",
    }
    // reset query
    from.value = { table: undefined, id: undefined }
  }
  modalSearchPlaceFrom.openModal(input, true)
}
function onToCityFieldClick(input: Event) {
  // TODO:implement input field reset on first click on the field
  const { toCity } = toRefs(search.value)
  const { to } = toRefs(fetchQuery.value)
  // reset fields if there is already some value
  if (toCity.value.query) {
    // reset search
    toCity.value = {
      searchResults: [],
      selected: null,
      query: "",
    }
    // reset query
    to.value = { table: undefined, id: undefined }
  }
  modalSearchPlaceTo.openModal(input, true)
}
```

```
// Datepicker handlers =>
// Определение количества месяцев в зависимости от ширины экрана
function calculateMultiCalendars() {
  if (screen.width < 600) {
    multiCalendars.value = 13
    scrollMonth.value = false
    closeAuto.value = false
  } else if (screen.width >= 600 && screen.width < 1400) {
    multiCalendars.value = 0
    scrollMonth.value = true
    closeAuto.value = true
  } else if (screen.width >= 1400) {
    multiCalendars.value = 1
    scrollMonth.value = true
    closeAuto.value = true
  }
  return multiCalendars.value
}
function onDepartureDateSelect(date: string | string[]) {
  if (typeof date === "string") {
    search.value.departureDate = date
    fetchQuery.value[QUERY_KEY.DEPARTURE_DATE] = date
    // reset validation
    validationErrors.value.departureDate = false
  }
}
function openDatepicker() {
  if (screen.width < 600) {
    lockPageScroll()
```

```
    scrollMonth.value = false
    closeAuto.value = false
    headShow.value = true
  } else {
    unlockPageScroll()
    modalSearchPlaceFrom.closeModal()
    modalSearchPassenger.closeModal()
    modalSearchPlaceTo.closeModal()
    scrollMonth.value = true
    closeAuto.value = true
    headShow.value = false
  }
}
function closeDatepicker() {
  headShow.value = false
  unlockPageScroll()
}
// Search cities handlers =>
function onFromCitySelect(userSelectedCity: ISearchCityItem) {
  // reset validation flag
  const { from } = toRefs(validationErrors.value.city)
  if (from.value) from.value = false
  const { name, countryName, table, id } = userSelectedCity
  const { selected, query } = toRefs(search.value.fromCity)
  selected.value = userSelectedCity
  // For cities => concate cityName + countryName
  // For countries => take only country name
  query.value = countryName ? `${name}, ${countryName}` : name
  fetchQuery.value.from.table = table
  fetchQuery.value.from.id = id
```

```
  modalSearchPlaceFrom.closeModal()
}
function onToCitySelect(userSelectedCity: ISearchCityItem) {
  // reset validation
  const { to } = toRefs(validationErrors.value.city)
  if (to.value) to.value = false
  const { name, countryName, table, id } = userSelectedCity
  const { selected, query } = toRefs(search.value.toCity)
  selected.value = userSelectedCity
  // For cities => concate cityName + countryName
  // For countries => take only country name
  query.value = countryName ? `${name}, ${countryName}` : name
  fetchQuery.value.to.table = table
  fetchQuery.value.to.id = id
  modalSearchPlaceTo.closeModal()
}
async function onFromCitySearch(searchQuery: string) {
  // get required referrence to the searchArr =>
  const { searchResults, query, selected } = toRefs(search.value.fromCity)
  // update query field in the object =>
  query.value = searchQuery
  // if user cleared search input => break the function
  if (!searchQuery) {
    searchResults.value = []
    selected.value = null
    return
  }
  // handle search response =>
  const { data } = await tripsStore.findCity(searchQuery)
```

```
  // update founded data =>
  if (data) searchResults.value = data.data.data
}
async function onToCitySearch(searchQuery: string) {
  // get required referrence to the searchArr =>
  const { searchResults, query, selected } = toRefs(search.value.toCity)
  // update query field in the object =>
  query.value = searchQuery
  // if user cleared search input => break the function
  if (!searchQuery) {
    searchResults.value = []
    selected.value = null
    return
  }
  // handle search response =>
  const { data } = await tripsStore.findCity(searchQuery)
  // update founded data =>
  if (data) searchResults.value = data.data.data
}
// Form submission =>
async function findTrips() {
  const { data: tripsFetched } = await tripsStore.fetchTrips()
  if (tripsFetched) {
    const currentQuery = JSON.parse(JSON.stringify(fetchQuery.value))
    return navigateTo({
      path: "/trips",
      // params: {
      //   buses: "автобуси",
      //   from: search.value.fromCity.selected?.name,
```

```
      //   to: search.value.toCity.selected?.name,
      // },
      query: currentQuery as LocationQueryRaw,
    })
  }
}
function toggleDirection() {
  const cityFrom = search.value.fromCity
  // update search obj
  search.value.fromCity = search.value.toCity
  search.value.toCity = cityFrom
  // update query obj
  fetchQuery.value.from = {
    table: search.value.fromCity.selected?.table,
    id: search.value.fromCity.selected?.id,
  }
  fetchQuery.value.to = {
    table: cityFrom.selected?.table,
    id: cityFrom.selected?.id,
  }
}
watch(
  () => screen.width,
  () => {
    calculateMultiCalendars()
  },
  { immediate: true },
)
// Lifecycle hooks
onUnmounted(() => {
```

```
  resetSearchValidationErrors()
})
</script>
```

Ticket.vue

```
<script setup lang="ts">
// Components
import AppButton from "~/components/UI/AppButton.vue";
import MarqueeInfinity from "~/components/PersonalCabinet/MarqueeInfinity.vue";
import AppTooltip from "~/components/UI/AppTooltip.vue";
// Modules
import { useScreen } from "vue-screen";
const { width } = toRefs(useScreen());
// Constants
import TripRoute from "~/components/trips/Trip/More/Route.vue";
import TripDiscounts from "~/components/trips/Trip/More/Discounts.vue";
import TripRule from "~/components/trips/Trip/More/Rule.vue";
import TripDetail from "~/components/trips/Trip/More/Detail.vue";
import TripConditions from "~/components/trips/Trip/More/Conditions.vue";
// Store instances
const tripStore = useTripsStore();
const profileStore = useProfileStore();
// Stores data
const { getRecountedPrice, pendingTripsDetails } = toRefs(tripStore);
const { archive } = toRefs(profileStore);
// Data
const openMoreInfo = ref<boolean>(false);
const openTicketPassengers = ref<boolean>(false);
const props = defineProps<{
  ticket: IOrder;
```

```
}>();
const emit = defineEmits<{
  "on-confirm-order": [orderId: number];
  "on-cancel-order": [returnInfo: IModalCancelPay];
  "on-fetch-details": [ticketId: IOrder];
}>();
// Methods
function onCancelOrder() {
  const returnInfo: IModalCancelPay = {
    ticketId: props.ticket.id,
    return: props.ticket.return,
    returnRules: props.ticket.returnRules,
    ticketIsPaid: props.ticket.badges.paid,
  };
  emit("on-cancel-order", returnInfo);
}
function getTicketDetailsById() {
  if (!props.ticket.more) emit("on-fetch-details", props.ticket);
  openMoreInfo.value = !openMoreInfo.value;
  if (openMoreInfo.value && window.innerWidth < 1024) {
    document.documentElement.style.overflow = "hidden";
  } else {
    document.documentElement.style.overflow = "unset";
  }
}
const toggleMobileTicketPassengers = () => {
  if (window.innerWidth < 1024) {
    openTicketPassengers.value = !openTicketPassengers.value;
  }
};
```

```
// Computed
const aboutBtnText = computed(() => {
  return openMoreInfo.value ? "Сховати деталі" : "Детальніше";
});
const aboutBtnPending = computed(() => {
  return pendingTripsDetails.value && openMoreInfo.value
    ? "Завантаження"
    : aboutBtnText.value;
});
//Start Animation openDetails Desktop
const enter = (element: HTMLElement) => {
  if (window.innerWidth >= 1024) {
    element.style.width = getComputedStyle(element).width;
    element.style.position = "absolute";
    element.style.visibility = "hidden";
    element.style.height = "auto";
    const height = getComputedStyle(element).height;
    element.style.width = "";
    element.style.position = "";
    element.style.visibility = "";
    element.style.height = "0";
    getComputedStyle(element).height;
    requestAnimationFrame(() => {
      element.style.height = height;
    });
  }
};
const afterEnter = (element: HTMLElement) => {
  if (window.innerWidth >= 1024) {
    element.style.height = "auto";
```

```
  }
};
const leave = (element: HTMLElement) => {
  if (window.innerWidth >= 1024) {
    element.style.height = getComputedStyle(element).height;
    getComputedStyle(element).height;
    requestAnimationFrame(() => {
      element.style.height = "0";
    });
  }
};
</script>

store/trips.ts
// Types =>
import {
  ITripFilters,
  ITripState,
  ITripDepartures,
  ITripDetails,
  FetchQueryIdT,
  ITripLocation,
  ITripsNearest,
  IFilterPayload,
  ISEOLinkTrips,
} from "~/types/trips"
import { ICurrencyOption, ISearchCityItem } from "~/types"
import {
  IFetchError,
  IFetchSuccess,
```

```
  IFetchSuccessPaginated,
  IFetchQuery,
} from "~/types/api"
// Modules =>
import QueryString from "qs"
import _ from "lodash"
// Constants =>
import { PROMISE_RESULT } from "~/constants/api"
import { QUERY_KEY } from "~/constants/trips"
import { ITripInfoPayload } from "~/types/booking"
export const useTripsStore = defineStore("trips", {
  state: (): ITripState => ({
    pendingTripsDetails: false,
    pendingTripsFetch: false,
    pendingTripsMore: false,
    pendingSearchBar: false,
    trips: {
      dates: [
        {
          date: { date: "", name: "" },
          price: 0,
          busDeparturesCount: 0,
          active: false,
        },
        {
          date: { date: "", name: "" },
          price: 0,
          busDeparturesCount: 0,
          active: false,
        },
```

```
    {
      date: { date: "", name: "" },
      price: 0,
      busDeparturesCount: 0,
      active: false,
    },
  ],
  departures: {
    first_page_url: null,
    next_page_url: null,
    prev_page_url: null,
    current_page: 1,
    per_page: null,
    from: null,
    path: null,
    data: [],
    to: null,
  },
  nearestDates: [],
},
search: {
  departureDate: "",
  fromCity: {
    searchResults: [],
    selected: null,
    query: "",
  },
  toCity: {
    searchResults: [],
    selected: null,
```

```
    query: "",
  },
  passengers: {
    adults: {
      count: 1,
    },
    children: {
      ages: [],
      count: 0,
    },
    animals: {
      weights: [],
      count: 0,
    },
    format: "",
  },
},
filters: {
  departureBusStops: [],
  arrivalBusStops: [],
  departureTypes: [],
  transportTypes: [],
  departureTimes: [],
  prepayments: [],
  busOptions: [],
  currencies: [],
  sortFields: [],
},
fetchQuery: {
  [QUERY_KEY.DEPARTURE_BUS_STOPS]: [],
```

```
  [QUERY_KEY.DEPARTURE_OWN_STOP]: undefined,
  [QUERY_KEY.ARRIVAL_BUS_STOPS]: [],
  [QUERY_KEY.DEPARTURE_TYPE_ID]: [],
  [QUERY_KEY.ARRIVAL_OWN_STOP]: undefined,
  [QUERY_KEY.DEPARTURE_TIMES]: [],
  [QUERY_KEY.TRANSPORT_TYPES]: [],
  [QUERY_KEY.DEPARTURE_DATE]: undefined,
  [QUERY_KEY.PREPAYMENTS]: [],
  [QUERY_KEY.BUS_OPTIONS]: [],
  [QUERY_KEY.TICKET_BACK]: undefined,
  [QUERY_KEY.CURRENCY_ID]: undefined,
  [QUERY_KEY.PASSENGERS]: {
    children: { count: undefined, ages: [] },
    animals: { count: undefined, weights: [] },
    adults: { count: 1 },
  },
  [QUERY_KEY.ORDER_ID]: undefined,
  [QUERY_KEY.PER_PAGE]: undefined,
  [QUERY_KEY.SORT]: undefined,
  [QUERY_KEY.FROM]: { table: undefined, id: undefined },
  [QUERY_KEY.TO]: { table: undefined, id: undefined },
  [QUERY_KEY.PAGE]: 1,
 },
 validationErrors: {
  city: {
    from: false,
    to: false,
  },
  departureDate: false,
  passengers: false,
```

```
  },
}),
actions: {
  async findCity(searchQuery: string) {
    this.pendingSearchBar = true
    const { data, error } = await useMyFetch<
      IFetchSuccessPaginated<ISearchCityItem>,
      IFetchError
    >("/locations", {
      query: { search: searchQuery },
    })
    this.pendingSearchBar = false
    return { data: data.value, error: error.value }
  },
  async fetchFilters() {
    const { data, error } = await useMyFetch<
      IFetchSuccess<ITripFilters>,
      IFetchError
    >("/filters")
    if (data.value) {
      this.filters = data.value.data
      // set basic sort
      if (!this.fetchQuery.sort) {
        this.fetchQuery.sort = this.filters.sortFields[0].id
      }
      // set basic currency
      if (!this.fetchQuery.currencyId) {
        const baseCurrency = this.filters.currencies.find(
          (currency) => currency.code === "UAH"
        )
```

```
      this.fetchQuery.currencyId = baseCurrency?.id
    }
  }
  return { data: data.value, error: error.value }
},
async fetchTrips(initialFetch = true, resetQuery = false) {
  // search validation
  if (process.client) this.validateSearch()
  if (this.hasValidationError) {
    return { data: null, error: "validation error" }
  }
  // show pertinent pending status depending on fetch type
  if (initialFetch) this.pendingTripsFetch = true
  else this.pendingTripsMore = true

  if (resetQuery) this.fetchQuery[QUERY_KEY.PAGE] = 1
  this.transformPassengers()
  const query = QueryString.stringify(this.fetchQuery)
  const { data, error } = await useMyFetch<
    IFetchSuccess<ITripDepartures>,
    IFetchError
  >(`/departures?${query}`)
  if (data.value) {
    if (!initialFetch) {
      // pushing new trips to existing trips
      _.mergeWith(
        this.trips.departures,
        data.value.data.departures,
        this.concatArrays
      )
```

```
      } else {
        // overwrite the whole data structure
        this.trips = data.value.data
      }
    }
    // clean departures data to avoid inconsistency between the data and internal app
state
    // in case of failed request
    if (error.value) this.resetDeparturesData()
    if (initialFetch) this.pendingTripsFetch = false
    else this.pendingTripsMore = false
    return { data: data.value, error: error.value }
  },
  async fetchSEOTrips(from: string | string[], to: string | string[]) {
    const { data, error } = await useMyFetch<
      IFetchSuccess<ISEOLinkTrips>,
      IFetchError
    >("/departures/search/cities", {
      query: { from, to },
    })
    if (data.value) {
      // desctructure response
      const { filter, data: foundedTrips } = data.value.data
      // desctructure app store values
      const {
        fromCity,
        toCity,
        departureDate: searchDepartureDate,
      } = toRefs(this.search)
      const {
```

```
      to,
      from,
      departureDate: queryDepartureDate,
    } = toRefs(this.fetchQuery)

    // save founded trips info to the app store
    this.trips = foundedTrips
    // update app search fields with founded cities data
    fromCity.value.selected = filter.from
    fromCity.value.query = `${filter.from.name}, ${filter.from.countryName}`
    to.value.id = filter.to.id
    to.value.table = filter.to.table
    toCity.value.selected = filter.to
    toCity.value.query = `${filter.to.name}, ${filter.to.countryName}`
    from.value.id = filter.from.id
    from.value.table = filter.from.table
    // update app search datepicker value
    searchDepartureDate.value = filter.departureDate
    queryDepartureDate.value = filter.departureDate
  }
  return { data: data.value, error: error.value }
},
async fetchTripsNearest() {
  this.pendingTripsMore = true
  this.transformPassengers()
  const query = QueryString.stringify(this.fetchQuery)
  const { data, error } = await useMyFetch<
    IFetchSuccess<ITripsNearest>,
    IFetchError
  >(`/departures/nearest?${query}`)
```

```
  if (data.value) {
    const targetObj = this.trips.nearestDates.find((nearestDateObj) => {
      return (
        nearestDateObj.date.date ===
        this.fetchQuery[QUERY_KEY.DEPARTURE_DATE]
      )
    })
    if (targetObj) {
      _.mergeWith(
        targetObj.departures,
        data.value.data.departures,
        this.concatArrays
      )
    }
  }
  this.pendingTripsMore = false
  return { data: data.value, error: error.value }
},
async fetchTripDetails(
  tripId: number | string | string[],
  query: ITripInfoPayload
) {
  this.pendingTripsDetails = true

  const { data, error } = await useMyFetch<
    IFetchSuccess<ITripDetails>,
    IFetchError
  >(`/departures/${tripId}`, { query })
  this.pendingTripsDetails = false
  return { data: data.value, error: error.value }
```

```
},
async getLocation(
  table: string | undefined,
  id: string | number | undefined
) {
  const { data, error } = await useMyFetch<
    IFetchSuccess<ITripLocation>,
    IFetchError
  >(`/locations/${table}/${id}`)
  let result = null
  if (data.value) result = data.value.data
  return { result, error: error.value }
},
async prefetchTrips(currentQuery: IFetchQuery) {
  const promises = []
  if (currentQuery) promises.push(this.fetchTrips())
  // fetch cities from/to if they exist in query
  if (currentQuery.from) {
    promises.push(
      this.getLocation(currentQuery.from.table, currentQuery.from.id)
    )
  }
  if (currentQuery.to) {
    promises.push(
      this.getLocation(currentQuery.to.table, currentQuery.to.id)
    )
  }
  const [trips, cityFrom, cityTo] = await Promise.allSettled(promises)
  // update search bar results with new values
  if (cityFrom.status === PROMISE_RESULT.SUCCESS) {
```

```
    const { result: city } = cityFrom.value
    // For cities => concate cityName + countryName
    // For countries => take only country name
    this.search.fromCity.selected = city
    this.search.fromCity.query = city.countryName
      ? `${city.name}, ${city.countryName}`
      : city.name
    this.validationErrors.city.from = false
  }
  if (cityTo.status === PROMISE_RESULT.SUCCESS) {
    const { result: city } = cityTo.value
    this.search.toCity.selected = city
    this.search.toCity.query = city.countryName
      ? `${city.name}, ${city.countryName}`
      : city.name
    this.validationErrors.city.to = false
  }
},
// helpers
transformPassengers() {
  // reset query values to avoid duplicates
  this.fetchQuery[QUERY_KEY.PASSENGERS] = {
    children: { count: undefined, ages: [] },
    animals: { count: undefined, weights: [] },
    adults: { count: undefined },
  }
  const { fetchQuery, search } = toRefs(this)
  // if adults were added
  const adultsCount = search.value[QUERY_KEY.PASSENGERS].adults
  if (adultsCount) {
```

```
      fetchQuery.value[QUERY_KEY.PASSENGERS].adults.count =
        search.value[QUERY_KEY.PASSENGERS].adults.count
    }
    // if children were added
    const childrenCount =
      search.value[QUERY_KEY.PASSENGERS].children.ages.length
    if (childrenCount) {
      fetchQuery.value[QUERY_KEY.PASSENGERS].children.count =
childrenCount
      search.value[QUERY_KEY.PASSENGERS].children.ages.forEach((child) => {
        return fetchQuery.value[QUERY_KEY.PASSENGERS].children.ages.push(
          child.age
        )
      })
    }
    // if animals were added
    const animalCount =
      search.value[QUERY_KEY.PASSENGERS].animals.weights.length
    if (animalCount) {
      fetchQuery.value[QUERY_KEY.PASSENGERS].animals.count = animalCount

      search.value[QUERY_KEY.PASSENGERS].animals.weights.forEach((animal)
=> {
        return
fetchQuery.value[QUERY_KEY.PASSENGERS].animals.weights.push(
          animal.weight
        )
      })
    }
  },
```

```
validateSearch() {
  this.resetSearchValidationErrors()
  const { fromCity, toCity, departureDate, passengers } = this.search

  if (!fromCity.selected) {
    this.validationErrors.city.from = true
  }
  if (!toCity.selected) {
    this.validationErrors.city.to = true
  }
  if (!departureDate) {
    this.validationErrors.departureDate = true
  }
  if (passengers.animals.weights.length) {
    const invalidWeight = passengers.animals.weights.some((weight) => {
      if (typeof weight.weight !== "number" || weight.weight == 0) {
        return true
      }
    })
    if (invalidWeight) this.validationErrors.passengers = true
  }
},
concatArrays(objValue: [], srcValue: []) {
  if (_.isArray(objValue)) {
    return objValue.concat(srcValue)
  }
},
parseUrlQuery(urlQuery: IFetchQuery) {
  // -- merging query from the URL to the app store
  _.merge(this.fetchQuery, urlQuery)
```

```
  // -- merging passengers to search bar data → transform them to inner logic state
  if (urlQuery.passengers) {
    function transformPassenger(objValue: [], srcValue: [], key: string) {
      if (key === "ages" || key === "weights") {
        return srcValue.map((value) => ({
          [key === "ages" ? "age" : "weight"]: value,
        }))
      }
    }
    _.mergeWith(
      this.search.passengers,
      urlQuery.passengers,
      transformPassenger
    )
    // set search date from query
    if (this.fetchQuery.departureDate) {
      this.search.departureDate = this.fetchQuery.departureDate
    }
  }
},
updateQuery(filterPayload: IFilterPayload) {
  const { filterKey, filterName, filterId } = filterPayload
  // update fetchQuery in the store
  const currentIds = this.fetchQuery[filterKey] as number[]

  if (currentIds.includes(filterId)) {
    this.fetchQuery[filterKey] = currentIds.filter((id) => id !== filterId)
  } else {
    this.fetchQuery[filterKey] = [...currentIds, filterId]
  }
```

```
},
resetFilters() {
  this.fetchQuery[QUERY_KEY.DEPARTURE_BUS_STOPS] = []
  this.fetchQuery[QUERY_KEY.ARRIVAL_BUS_STOPS] = []
  this.fetchQuery[QUERY_KEY.DEPARTURE_TYPE_ID] = []
  this.fetchQuery[QUERY_KEY.DEPARTURE_TIMES] = []
  this.fetchQuery[QUERY_KEY.TRANSPORT_TYPES] = []
  this.fetchQuery[QUERY_KEY.PREPAYMENTS] = []
  this.fetchQuery[QUERY_KEY.BUS_OPTIONS] = []
},
resetFetchQuery() {
  this.fetchQuery = {
    [QUERY_KEY.DEPARTURE_BUS_STOPS]: [],
    [QUERY_KEY.DEPARTURE_OWN_STOP]: undefined,
    [QUERY_KEY.ARRIVAL_BUS_STOPS]: [],
    [QUERY_KEY.DEPARTURE_TYPE_ID]: [],
    [QUERY_KEY.ARRIVAL_OWN_STOP]: undefined,
    [QUERY_KEY.DEPARTURE_TIMES]: [],
    [QUERY_KEY.TRANSPORT_TYPES]: [],
    [QUERY_KEY.DEPARTURE_DATE]: undefined,
    [QUERY_KEY.PREPAYMENTS]: [],
    [QUERY_KEY.BUS_OPTIONS]: [],
    [QUERY_KEY.TICKET_BACK]: undefined,
    [QUERY_KEY.CURRENCY_ID]: undefined,
    [QUERY_KEY.PASSENGERS]: {
      children: { count: undefined, ages: [] },
      animals: { count: undefined, weights: [] },
      adults: { count: 1 },
    },
    [QUERY_KEY.ORDER_ID]: undefined,
```

```
      [QUERY_KEY.PER_PAGE]: undefined,
      [QUERY_KEY.SORT]: undefined,
      [QUERY_KEY.FROM]: { table: undefined, id: undefined },
      [QUERY_KEY.TO]: { table: undefined, id: undefined },
      [QUERY_KEY.PAGE]: 1,
    }
    // set defaults sort and currency
    if (this.filters.sortFields.length > 0) {
      this.fetchQuery[QUERY_KEY.SORT] = this.filters.sortFields[0].id
    }
    if (this.filters.currencies.length > 0) {
      const UAHCurrency = this.filters.currencies.find(
        (currency) => currency.code === "UAH"
      )
      this.fetchQuery[QUERY_KEY.CURRENCY_ID] = UAHCurrency
        ? UAHCurrency.id
        : this.filters.currencies[0].id
      // this.fetchQuery[QUERY_KEY.CURRENCY_ID] =
this.filters.currencies[0].id
    }
  },
  resetSearchValidationErrors() {
    this.validationErrors = {
      city: {
        from: false,
        to: false,
      },
      departureDate: false,
      passengers: false,
    }
```

```
},
resetSearch() {
  this.search = {
    departureDate: "",
    fromCity: {
      searchResults: [],
      selected: null,
      query: "",
    },
    toCity: {
      searchResults: [],
      selected: null,
      query: "",
    },
    passengers: {
      adults: {
        count: 1,
      },
      children: {
        ages: [],
        count: 0,
      },
      animals: {
        weights: [],
        count: 0,
      },
      format: "",
    },
  }
},
```

```
resetDeparturesData() {
  this.trips = {
    dates: [
      {
        date: { date: "", name: "" },
        price: 0,
        busDeparturesCount: 0,
        active: false,
      },
      {
        date: { date: "", name: "" },
        price: 0,
        busDeparturesCount: 0,
        active: false,
      },
      {
        date: { date: "", name: "" },
        price: 0,
        busDeparturesCount: 0,
        active: false,
      },
    ],
    departures: {
      first_page_url: null,
      next_page_url: null,
      prev_page_url: null,
      current_page: 1,
      per_page: null,
      from: null,
      path: null,
```

```
      data: [],
      to: null,
    },
    nearestDates: [],
  }
 },
},
getters: {
 getCurrenciesNameConcat: ({ filters }) => {
  return filters.currencies.map((currency) => ({
   code: currency.code,
   id: currency.id,
   name: `${currency.name}, ${currency.code}`,
  }))
 },
 isSelectedFilter: ({ fetchQuery }) => {
  return (valueKey: FetchQueryIdT, filterKey: QUERY_KEY): boolean => {
   if (Array.isArray(fetchQuery[filterKey])) {
    return fetchQuery[filterKey]?.some(
     (filterId: number) => filterId === valueKey
    )
   } else {
    return fetchQuery[filterKey] === valueKey
   }
  }
 },
 getSelectedSort: ({ filters, fetchQuery }) => {
  return (
   filters.sortFields.find(
    (sortType) => sortType.id === fetchQuery.sort
```

```
    ) || null
  )
},
getSelectedCurrency: ({ filters, fetchQuery }) => {
  return filters.currencies.find(
    (currency: ICurrencyOption) => currency.id === fetchQuery.currencyId
  )
},
getSelectedFilters: ({ fetchQuery, filters }) => {
  const selectedFilters: IFilterPayload[] = []
  const arrayFiltersKeys = new Map([
    [QUERY_KEY.BUS_OPTIONS, "busOptions"],
    [QUERY_KEY.DEPARTURE_BUS_STOPS, "departureBusStops"],
    [QUERY_KEY.ARRIVAL_BUS_STOPS, "arrivalBusStops"],
    [QUERY_KEY.DEPARTURE_TYPE_ID, "departureTypes"],
    [QUERY_KEY.DEPARTURE_TIMES, "departureTimes"],
    [QUERY_KEY.TRANSPORT_TYPES, "transportTypes"],
    [QUERY_KEY.PREPAYMENTS, "prepayments"],
  ])
  for (const [filterKey, filterArrayName] of arrayFiltersKeys) {
    fetchQuery[filterKey].forEach((selectedId: number) => {
      const filterExist = filters[filterArrayName].find(
        (option) => option.id === selectedId
      )

      if (filterExist) {
        selectedFilters.push({
          filterName: filterExist.name,
          filterId: selectedId,
          filterKey,
```

```
      })
    }
  })
 }
 return selectedFilters
},
getActiveDate: ({ trips }) => {
 return trips.dates.find((date) => date.active)
},
getRecountedPrice: () => {
 return (price: number) => {
  return price / 100
 }
},
hasValidationError: ({ validationErrors }) => {
 for (const key in validationErrors) {
  if (typeof validationErrors[key] === "object") {
   for (const subKey in validationErrors[key]) {
    if (validationErrors[key][subKey] === true) {
     return true
    }
   }
  } else if (validationErrors[key] === true) {
   return true
  }
 }
 return false
},
tripDetailsTabHasContent: () => {
 return (data: unknown) => {
```

```
      for (const key in data) {
        const value = data[key]
        if (Array.isArray(value) && value.length) return true
        if (typeof value === "string" && value) return true
        if (typeof value === "boolean" && value) return true
      }
      return false
    }
  },
 },
})
```

nuxt.config.ts

```
// https://nuxt.com/docs/api/configuration/nuxt-config
export default defineNuxtConfig({
 devtools: { enabled: false },
 runtimeConfig: {
  public: {
   API_BASE_URL:
    process.env.NUXT_API_BASE_URL || "https://api.rubikon.staj.bid/api/v1/",
  },
 },
 css: ["@/assets/scss/main.scss"],
 vite: {
  vue: {
   script: {
    propsDestructure: true,
   },
  },
  css: {
```

```
    preprocessorOptions: {
      scss: {
        additionalData: `
          @import "@/assets/scss/util";
          @import "@/assets/scss/vars.scss";
        `,
      },
    },
  },
},
build: {
  transpile: ["@vuepic/vue-datepicker"],
},
modules: [
  "@nuxtjs/eslint-module",
  "@pinia/nuxt",
  "@pinia-plugin-persistedstate/nuxt",
  "@nuxt/image",
  "@vueuse/nuxt",
  "nuxt-lodash",
  "nuxt-icon",
  "nuxt-delay-hydration",
],
postcss: {
  plugins: {
    autoprefixer: {
      overrideBrowserslist: ["last 4 versions"],
      cascade: false,
    },
  },
```

```
  },
  eslint: {
    lintOnStart: false,
  },
  image: {
    inject: true,
  },
  // eslint-disable-next-line @typescript-eslint/ban-ts-comment
  // @ts-ignore
  delayHydration: {
    debug: true,
    mode: "mount",
  },
  typescript: {
    tsConfig: {
      compilerOptions: {
        verbatimModuleSyntax: false,
      },
    },
  },
})
```

ДОДАТОК Б

```
<div class="search__group" :class="{ error: validationErrors.city.from }">
  <AppInput
    class="app-input-box--header app-input-box--header-from"
    :class="{ error: validationErrors.city.from }"
    v-model="search.fromCity.query"
    icon-left="SocialLocal"
    placeholder="Звідки"
    id="placeFrom"
    :read-only="modalSearchPlaceFrom.readOnlySearch.value"
    @update:model-value="onFromCitySearchDebounced"
    @click="onFromCityFieldClick"
  />
  <!-- Mobile only -->
  <ModalTransition>
    <PlaceModal
      v-if="modalSearchPlaceFrom.isOpen.value"
      v-click-outside="modalSearchPlaceFrom.closeModal"
      input-label="Звідки"
      :pending="pendingSearchBar"
      :search-query="search.fromCity.query"
      :selected-city="search.fromCity.selected"
      :cities="search.fromCity.searchResults"
      @modal-close="modalSearchPlaceFrom.closeModal"
      @on-city-select="onFromCitySelect"
      @on-search-input="onFromCitySearchDebounced"
    />
  </ModalTransition>
  <span class="search__error">Необхідно заповнити поле</span>
</div>
```

Рисунок Б1 – Вигляд компоненту для поля форми «Звідки»

```
function toggleDirection() {
  const cityFrom = search.value.fromCity

  // update search obj
  search.value.fromCity = search.value.toCity
  search.value.toCity = cityFrom

  // update query obj
  fetchQuery.value.from = {
    table: search.value.fromCity.selected?.table,
    id: search.value.fromCity.selected?.id,
  }
  fetchQuery.value.to = {
    table: cityFrom.selected?.table,
    id: cityFrom.selected?.id,
  }
}
```

Рисунок Б2 – Функція toggleDirection для зміни місцями значень «Куди» та «Звідки»

```
async function findTrips() {
  const { data: tripsFetched } = await tripsStore.fetchTrips()

  if (tripsFetched) {
    const currentQuery = JSON.parse(JSON.stringify(fetchQuery.value))

    return navigateTo({
      path: "/trips",
      query: currentQuery as LocationQueryRaw,
    })
  }
}
```

Рисунок Б3 – Функція findTrips для пошуку квитків

```
const hasDirectionError = computed(() => {
  return validationErrors.value.city.to || validationErrors.value.city.from
})
```

Рисунок Б4 – Комп’ютед hasDirectionError

```
export enum QUERY_KEY {
  DEPARTURE_BUS_STOPS = "departureBusStopIds",
  DEPARTURE_OWN_STOP = "ownDepartureBusStop",
  ARRIVAL_BUS_STOPS = "arrivalBusStopIds",
  DEPARTURE_TYPE_ID = "departureTypeIds",
  ARRIVAL_OWN_STOP = "ownArrivalBusStop",
  TRANSPORT_TYPES = "transportTypeIds",
  DEPARTURE_TIMES = "departureTimeIds",
  DEPARTURE_DATE = "departureDate",
  PREPAYMENTS = "prepaymentIds",
  BUS_OPTIONS = "busOptionIds",
  TICKET_BACK = "ticketBack",
  CURRENCY_ID = "currencyId",
  PASSENGERS = "passengers",
  ORDER_ID = "orderId",
  PER_PAGE = "perPage",
  SORT = "sort",
  PAGE = "page",
  FROM = "from",
  TO = "to",
}
```

Рисунок Б5 – Створення константи QUERY_KEY

```
async fetchTripDetails(
  tripId: number | string | string[],
  query: ITripInfoPayload
) {
  this.pendingTripsDetails = true

  const { data, error } = await useMyFetch<
    IFetchSuccess<ITripDetails>,
    IFetchError
  >(`/departures/${tripId}`, { query })

  this.pendingTripsDetails = false
  return { data: data.value, error: error.value }
},
```

Рисунок Б6 – Асинхронний метод fetchTripDetails

```
<Fancybox :options="optionsFancy" class="detail-gallery">
  <a
    data-fancybox="gallery"
    v-for="(photo, i) in data.busPhotos"
    :key="i"
    :href="photo"
    class="detail-gallery__link"
    :data-count="`${photo.length - 5}`"
  >
    <NuxtImg :src="photo" :alt="`busPhoto-${i}`" />
  </a>
</Fancybox>
```

Рисунок Б7 – Компонент для галереї

```
const optionsFancy = {
  infinite: false,
  classes: {
    image: "f-thumbs__slide__img",
    isSelected: "is-selected",
    isPrev: "is-prev",
    isNext: "is-next",
  },
}
```

Рисунок Б8 – Опції для компоненту галереї

```
<script setup lang="ts">
import * as FancyboxLibs from "@fancyapps/ui"
import "@fancyapps/ui/dist/fancybox/fancybox.css"

const props = defineProps({
  options: Object,
})
const { Fancybox } = FancyboxLibs
const fancyGallery = ref<HTMLElement | null>(null)

onMounted(() => {
  Fancybox.bind(fancyGallery.value, "[data-fancybox]", {
    ...(props.options || {}),
  })
})

onUpdated(() => {
  Fancybox.unbind(fancyGallery.value)
  Fancybox.close()

  Fancybox.bind(fancyGallery.value, "[data-fancybox]", {
    ...(props.options || {}),
  })
})
                You, 3 months ago • connection Fancybox
onUnmounted(() => {
  Fancybox.destroy()
})
</script>
```

Рисунок Б9 – Компонент Fancybox

```
<div class="trips-select">
  <p>Сортувати за</p>
  <AppSelect
    class="trips-select__component trips-select__component--sort"
    :options="sortFields"
    :selected="getSelectedSort"
    @on-choose="updateSort"
  />
</div>
```

Рисунок Б10 – Компонент сортування

```
async function updateSort(selectedSort: ISelectOption) {
  closeModal("sort")

  fetchQuery.value[SORT] = selectedSort.id as string
  await fetchTrips(true, true)

  updateURLQuery(fetchQuery.value)
}
```

Рисунок Б11 – Функція updateSort

ДОДАТОК В

Раджабова С.Г., студентка
Університету митної справи та фінансів
(науковий керівник – Ульяновська Ю.В., к.т.н., доц.,
завідувач кафедри комп'ютерних наук
та інженерії програмного забезпечення
Університету митної справи та фінансів)

ЗАСТОСУВАННЯ МЕТОДІВ ПРИЙНЯТТЯ РІШЕНЬ ДЛЯ ОПТИМІЗАЦІЇ ПРОЦЕСІВ УПРАВЛІННЯ ІТ-ПРОЄКТАМИ

У сучасному світі, де інформаційні технології швидко розвиваються, ефективне управління ІТ-проєктами стає ключовим фактором успіху багатьох організацій. У зв'язку з цим зростає потреба в оптимізації процесів планування, контролю та моніторингу проєктів. Методи прийняття рішень такі як методи мережевого планування, можуть бути використаними для забезпечення високої продуктивності, ефективності та гнучкості управління проєктами, відслідковування термінів тощо.

Мережеві графіки є одними з ключових компонентів у сфері управління проєктами, який сприяє ефективності планування, виконання та контролю проєктів. Їх важливість може бути підкреслена наступними аспектами:

– забезпечують цілісний і візуальний перегляд всіх завдань та етапів проєкту. Вони допомагають командам та керівництву легко ідентифікувати залежності між завданнями, визначити критичні шляхи та ризики.

– менеджери можуть ефективно розподіляти ресурси, забезпечуючи їх оптимальне використання. Це не лише підвищує продуктивність, але і допомагає уникнути перевантаження або недостатнього використання ресурсів.

– дозволяють точно оцінити тривалість проєкту, виходячи з аналізу тривалості окремих завдань і їх взаємозв'язків. Це сприяє точному плануванню та контролю термінів виконання проєкту.

Розглянемо на прикладі задачі використання мережевих графіків в управлінні IT-проєктами.

В таблиці 1 наведено перелік операцій, які розробив керівник проєкту, які передбачають реалізацію проєкту, також вказано час, необхідний для виконання кожної операції та взаємозв'язок робот.

Таблиця 1

Робота	Безпосередня робота	Час виконання
A	–	4
B	–	6
C	–	5
D	B	2
E	A	9
F	B	4
G	C, D	8
H	B, E	3
I	F, G	5
J	H	7

Обчислимо довжину критичного шляху; кількість робіт, яка знаходиться на критичному шляху; чи можна відкласти виконання роботи F без відстрочки завершення проєкту в цілому.

Для розв'язування задачі необхідно розрахувати такі часові параметри: ранній термін початку операції від початку проєкту (ES) та ранній термін закінчення операції (EF).

В якості дати початку проєкту приймається «нульовий день» і він буде раннім терміном початку операції A. Щоб отримати ES для операції B слід додати тривалість операції A (тобто 4) до 0 і отримаємо значення 4.

EF для операції A дорівнює терміну ES (тобто 0) плюс її тривалість 4. Завдання B також починається з «нульового дня», EF операції B дорівнює терміну ES тобто 6. Та завдання C починається з «нульового дня», EF операції C дорівнює терміну ES тобто 5. Далі рахуємо для кожного завдання, результат представлений на рис 1.

Критичний шлях – це ланцюг послідовно пов'язаних операцій у мережевому графіку з найбільшою тривалістю, він характеризується як шлях з нульовим резервом часу. В даній задачі критичний шлях: A, E, H, J (жирний шрифт на рис. 1). Тривалість критичного шляху 23 дні.

Щоб визначити, чи можна відкласти виконання роботи F, потрібно розглянути, чи входить вона в критичний шлях. Робота F не є частиною критичного шляху, тому її можна відкласти без відстрочки завершення проєкту в цілому.

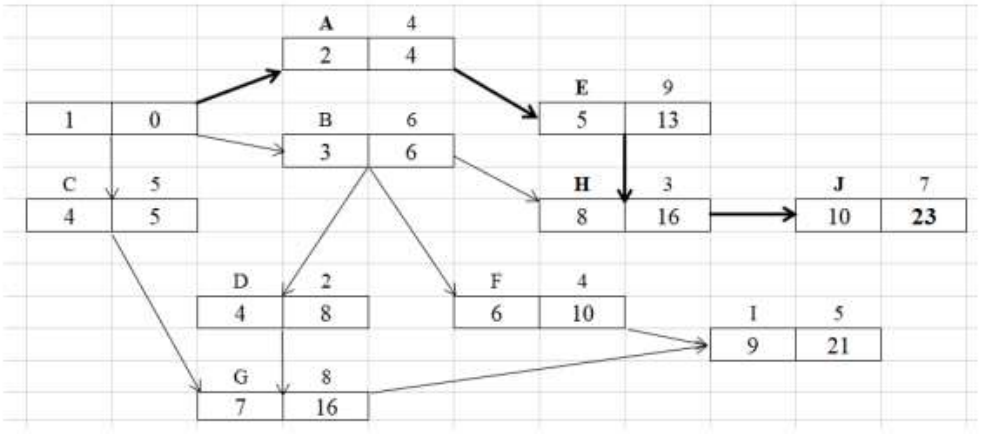

Рис. 1 Мережевий графік

Таким чином, роль мережевих графіків у сучасному IT-менеджменті не можна недооцінювати. Вони не лише сприяють ефективному плануванню та управлінню проєктами, але і забезпечують інструментарій для адаптації до змін та невизначеності, що є невід'ємною частиною сучасного динамічного IT-середовища.