НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«До захисту допущено»
Завідувач кафедри
_____ Євгенія СУЛЕМА
«___» _____ 2025 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інженерія програмного забезпечення мультимедійних та інформаційно-пошукових систем»

спеціальності 121 Інженерія програмного забезпечення

на тему: «Мобільний застосунок для організації пасажирських перевезень»

Виконав:
студент IV курсу, групи КП-11
Маковецький Павло Віталійович _____

Керівник:
доцент кафедри ПЗКС, к.т.н.,
Рибачок Наталія Антонівна _____

Консультант з нормоконтролю:
доцент кафедри ПЗКС, к.т.н., доцент,
Онай Микола Володимирович _____

Рецензент:
доцент кафедри ЕІ ФЕЛ, к.т.н.,
Вунтесмері Юрій Володимирович _____

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2025 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет прикладної математики
Кафедра програмного забезпечення комп'ютерних систем

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 121 «Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення мультимедійних та інформаційно-пошукових систем»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Євгенія СУЛЕМА
«___»_____2024 р.

ЗАВДАННЯ

на дипломний проєкт студенту

Маковецькому Павлу Віталійовичу

1. Тема проєкту «Мобільний застосунок для організації пасажирських перевезень», керівник проєкту Рибачок Наталія Антонівна, к.т.н., доцент, затверджені наказом по університету № 1808-С від «29» травня 2025 р.

2. Термін подання студентом проєкту «13» червня 2025 р.

3. Вихідні дані до проєкту: див. Технічне завдання.

4. Зміст пояснювальної записки:
   - аналіз предметної області та існуючих рішень;
   - аналіз мов програмування та технологій розроблення мобільних застосунків;
   - розроблення мобільного застосунку для організації пасажирських перевезень;
   - аналіз розроблених алгоритмів та підпрограм;
   - аналіз розробленого застосунку.

5. Перелік обов'язкового графічного матеріалу:
   - структура бази даних (креслення);
   - блок-схема алгоритму роботи застосунку (креслення);
   - архітектура застосунку (плакат);
   - діаграма варіантів використання (плакат).

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Онай М.В., доцент		

7. Дата видачі завдання «31» жовтня 2024 р.

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1.	Вивчення літератури за тематикою проєкту	13.11.2024	
2.	Розроблення та узгодження технічного завдання	22.11.2024	
3.	Розроблення структури застосунку	15.12.2024	
4.	Підготовка матеріалів першого розділу дипломного проєкту	28.12.2024	
5.	Розроблення дизайну сторінок та графічних елементів	01.02.2025	
6.	Підготовка матеріалів другого розділу дипломного проєкту	19.02.2025	
7.	Програмна реалізація мобільного застосунку	11.03.2025	
8.	Тестування застосунку	18.03.2025	
9.	Підготовка матеріалів третього розділу дипломного проєкту	27.03.2025	
10.	Підготовка матеріалів четвертого розділу дипломного проєкту	15.04.2025	
11.	Підготовка графічної частини дипломного проєкту	22.04.2025	
12.	Оформлення документації дипломного проєкту	27.05.2025	

Студент Павло МАКОВЕЦЬКИЙ

Керівник проєкту Наталія РИБАЧОК

АНОТАЦІЯ

Даний дипломний проект присвячений розробленню мобільного застосунку для організації пасажирських перевезень.

У роботі виконано порівняльний аналіз існуючих рішень для планування та підготовки до подорожей, організації пасажирських перевезень, сформульовано вимоги до ПЗ, що повністю задовільнить цільову аудиторію, обґрунтовано вибір технологій та допоміжних засобів серверної та клієнтської частин для реалізації даного рішення.

Розроблений застосунок надає мандрівникам можливість розпланувати свою подорож, контролювати свій бюджет, зберігати всі необхідні файли в одному місці, та вести персоналізовані списки речей. Застосунок підтримує різні типи подій, автоматичний пошук локацій та прогноз погоди

В даному проекті розроблено та досліджено: архітектуру серверної та клієнтської частини мобільного застосунку, алгоритми функціональності, спроектовано модель даних, формалізовані вимоги, а також графічні елементи та дизайн вікон застосунку.

ABSTRACT

This diploma project is dedicated to the development of a mobile application for the organization of passenger transportation.

The work includes a comparative analysis of existing solutions for planning and preparing for travel, organizing passenger transportation, formulating requirements for software that will fully satisfy the target audience, justifying the choice of technologies and auxiliary tools for the server and client parts to implement this solution.

The developed application provides travelers with the ability to plan their trip, control their budget, store all the necessary files in one place, and keep personalized lists of things. The application supports various types of events, automatic location search and weather forecast

In this project, the following were developed and studied: the architecture of the server and client parts of the mobile application, functionality algorithms, a data model, formalized requirements, as well as graphic elements and design of application windows.

ДП.045440-01-90 Мобільний застосунок для організації пасажирських перевезень.
Відомість проєкту

Позначення	Найменування	Кіл-ть	Примітка
	Документація проєкту		
ДП.045440-02-91	Мобільний застосунок	5	
	для організації		
	пасажирських перевезень.		
	Технічне завдання		
ДП.045440-03-81	Мобільний застосунок	60	
	для організації		
	пасажирських перевезень.		
	Пояснювальна записка		
ДП.045440-04-51	Мобільний застосунок	4	
	для організації		
	пасажирських перевезень.		
	Програма та методика		
	тестування		
ДП.045440-05-34	Мобільний застосунок	10	
	для організації		
	пасажирських перевезень.		
	Керівництво користувача		
ДП.045440-06-99	Мобільний застосунок	1	
	для організації		
	пасажирських перевезень.		
	Структура бази даних.		
	ER-діаграма		

Позначення	Найменування	Кіл-ть	Примітка
ДП.045440-07-99	Мобільний застосунок	1	
	для організації		
	пасажирських перевезень.		
	Алгоритм роботи		
	застосунку. Блок-схема		
	алгоритму		
ДП.045440-08-98	Мобільний застосунок	1	
	для організації		
	пасажирських перевезень.		
	Компакт-диск		

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

"ЗАТВЕРДЖЕНО"

Завідувач кафедри

_____ Євгенія СУЛЕМА

"___" _____ 2024 р.

МОБІЛЬНИЙ ЗАСТОСУНОК ДЛЯ ОРГАНІЗАЦІЇ ПАСАЖИРСЬКИХ ПЕРЕВЕЗЕНЬ

Технічне завдання

ДП.045440-02-91

"ПОГОДЖЕНО"

Керівник проєкту:

_____ Наталія РИБАЧОК

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Павло МАКОВЕЦЬКИЙ

2024

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: мобільний застосунок для організації пасажирських перевезень.

Галузь застосування: інформаційні технології.

2. ПІДСТАВА ДЛЯ РОЗРОБЛЕННЯ

Підставою для розроблення є завдання на дипломне проєктування, затверджене кафедрою програмного забезпечення комп'ютерних систем Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського» (КПІ ім. Ігоря Сікорського).

3. ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для складання розкладу подій, контролю бюджету, зберігання документів, та ведення списків речей з метою спрощення процесу планування і підготовки до подорожей для пасажирських перевезень.

4. ВИМОГИ ДО ПРОГРАМНОГО ПРОДУКТУ

Мобільний застосунок має забезпечувати такі основні функції:

1. Система повинна підтримувати наступні ролі: Мандрівник, Адміністратор.
2. Мандрівник має можливість керування подорожами (створення, перегляд, видалення).
3. Мандрівник має можливість керувати подіями на кожен день подорожі (створення, перегляд, редагування, видалення).

4. Мандрівник має можливість складати та заповнювати персоналізовані списки речей.
5. Мандрівник має можливість отримати статистику по витратам на основі коштів, затрачених на події та витратах, доданих власноруч (створення, редагування, видалення) у вигляді діаграми.
6. Мандрівник має можливість прикладання файлів до окремих подій та подорожей загалом.
7. Адміністратор має можливість перегляду статистики за Мандрівниками.
8. Адміністратор має можливість керування акаунтами Мандрівників (створення, перегляд, видалення).

Системні вимоги:

1. Система повинна мати дизайн інтерфейсу з використанням у якості базових чорного та білого кольорів.
2. Система повинна підтримувати можливість автентифікації за допомогою OAuth 2 (Google).
3. Система повинна мати браузерну версію застосунку.
4. Система має бути здатна до підтримки та адаптації екранів різної роздільної здатності та розмірів.

Додаткові вимоги:

1. Розробка повинна бути виконана на мові програмування JavaScript.
2. Мобільний застосунок повинен працювати на мобільних пристроях з операційною системою iOS версії 13.0 або вище, та Android версії 10.0 або вище.
3. Застосунок повинен працювати на мобільних пристроях, які мають щонайменше 1 ГБ оперативної пам’яті.
4. Застосунок повинен працювати на всіх сучасних браузерах із рушіями Blink, SpiderMonkey, WebKit.

5. ВИМОГИ ДО ПРОЄКТНОЇ ДОКУМЕНТАЦІЇ

У процесі виконання проєкту повинна бути розроблена наступна документація:

1) пояснювальна записка;

2) програма та методика тестування;

3) керівництво користувача;

4) креслення:

– «Структура бази даних»;

– «Блок-схема алгоритму роботи застосунку».

5) плакати:

– «Архітектура застосунку»;

– «Діаграма варіантів використання застосунку».

6. ЕТАПИ ПРОЄКТУВАННЯ

Вивчення літератури за тематикою роботи..... 13.12.2024

Розроблення та узгодження технічного завдання 25.12.2024

Розроблення структури застосунку..... 15.01.2025

Розроблення дизайну сторінок та графічних елементів 15.03.2025

Програмна реалізація застосунку..... 25.04.2025

Тестування застосунку 27.04.2025

Підготовка матеріалів текстової частини проєкту 28.04.2025

Підготовка матеріалів графічної частини проєкту..... 28.05.2025

Оформлення технічної документації проєкту..... 28.05.2025

7. ПОРЯДОК ТЕСТУВАННЯ РОЗРОБКИ

Тестування розробленого програмного продукту виконується відповідно до “Програми та методики тестування”.

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

"ЗАТВЕРДЖЕНО"

Завідувач кафедри

_____ Євгенія СУЛЕМА

"___" _____ 2025 р.

МОБІЛЬНИЙ ЗАСТОСУНОК ДЛЯ ОРГАНІЗАЦІЇ ПАСАЖИРСЬКИХ ПЕРЕВЕЗЕНЬ

Пояснювальна записка

ДП.045440-03-81

"ПОГОДЖЕНО"

Керівник проєкту:

_____ Наталія РИБАЧОК

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Павло МАКОВЕЦЬКИЙ

2025

ЗМІСТ

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

ПЗ – програмне забезпечення.

Імпорт – процес перенесення зовнішніх даних або контенту в пристрої та програмне забезпечення.

Офлайн – робота пристрою чи програмного забезпечення не за допомогою Інтернету або іншої комп'ютерної мереж.;

Кастомізація – процес адаптації та налаштування продукту під окрему аудиторію, об'єднану певними особливостями.

Персоналізація – те саме, що й кастомізація.

Бібліотека – набір готових функцій, класів або модулів, які можна повторно використовувати в програмах.

Фреймворк – програмна платформа, яка полегшує розробку застосунків завдяки вже готовій архітектурі, структурі та наборам інструментів.

Повний стек (Full-stack) – набір технологій для розробки як серверної (backend), так і клієнтської (frontend) частин застосунку.

JVM (Java Virtual Machine) – віртуальна машина Java, яка виконує байт-код незалежно від операційної системи.

JRE (Java Runtime Environment) – середовище для запуску Java-програм, яке включає JVM та бібліотеки.

JSON (JavaScript Object Notation) – це легкий формат обміну даними, заснований на підмножині синтаксису JavaScript

JDK (Java Development Kit) – комплект для розробки на Java, що містить компілятор, JRE та додаткові інструменти.

Байт-код – проміжний машинно-незалежний код, який компілюється з Java-програми для виконання на JVM.

Garbage Collector (GC) – автоматизований механізм звільнення пам’яті, який видаляє непотрібні об’єкти.

Інтерпретатор – програма, яка виконує код рядок за рядком без попередньої компіляції.

PIP – стандартний менеджер пакетів для Python.

Duck Typing – концепція, коли тип об'єкта визначається за його поведінкою, а не за декларацією типу.

ECMAScript – офіційний стандарт, на якому базується JavaScript.

NPM (Node Package Manager) – менеджер пакетів для JavaScript.

Callback – функція, яка передається як аргумент до іншої функції для її виклику після завершення певної операції.

Event Loop – механізм виконання асинхронного коду в JavaScript.

SPA (Single Page Application) – вебзастосунок, який завантажує одну HTML-сторінку та динамічно оновлює її контент без повного перезавантаження.

DOM (Document Object Model) – об’єктна модель документа HTML або XML, яка дозволяє змінювати його структуру та вміст за допомогою JavaScript.

Virtual DOM – віртуальне представлення DOM у вигляді дерева JavaScript-об’єктів, яке використовується для оптимізації оновлень реального DOM.

Reactive Binding (реактивне зв'язування) – автоматичне оновлення DOM при зміні стану даних.

Two-way Data Binding (двостороннє зв'язування) – автоматичне синхронізування даних між моделлю і представленням.

Dependency Injection (DI) – шаблон передавання залежностей (сервісів, даних) в компоненти через конструктори або ін’єктори.

CLI (Command Line Interface, інтерфейс командного рядка) – засіб взаємодії з програмним забезпеченням за допомогою команд, кожна з яких відформатована у вигляді рядка тексту.

JSX (JavaScript XML) – синтаксис для опису інтерфейсів у React, що дозволяє писати HTML-подібний код усередині JavaScript.

Unidirectional Data Flow – односпрямований потік даних, коли інформація передається від батьківських компонентів до дочірніх.

ACID – набір властивостей для забезпечення надійності транзакцій у базах даних: A (Atomicity) – атомарність, незворотність кожної транзакції, C (Consistency) – консистентність, переведення бази з одного валідного стану в інший, I (Isolation) – ізоляція, транзакції не заважають одна одній, D (Durability) – надійність, дані зберігаються навіть після збою.

Реплікація – механізм копіювання даних між кількома серверами для забезпечення доступності та резервування.

Індексація – створення спеціальних структур для прискорення пошуку даних у базі.

Транзакція – група операцій з базою даних, що виконується як єдине ціле.

SQL (Structured Query Language) – мова структурованих запитів для роботи з реляційними базами.

NoSQL (Not Only SQL) – узагальнена назва для нереляційних систем зберігання даних.

BSON – двійковий формат для зберігання документів, розширена версія JSON.

JSONB – бінарне зберігання JSON у PostgreSQL з можливістю індексації.

MVCC (Multi-Version Concurrency Control) – механізм багатоверсійного контролю одночасного доступу, що дозволяє виконувати паралельні транзакції без блокувань.

CTE (Common Table Expressions) – проміжна таблиця, яка створюється у рамках одного запиту.

Автентифікація – перевірка автентичності користувача шляхом порівняння введених ним облікових даних (або іншого ідентифікатора), збережених у базі даних.

Сесія – обмежений у часі двосторонній зв'язок, що забезпечує інтерактивне вираження та обмін інформацією між двома або більше комунікаційними пристроями/

Авторизація – процес надання суб'єкту доступу до чогось або виконання певних дій у комп'ютерній системі.

API (Application Programming Interface) – набір правил і протоколів, що дозволяє програмам взаємодіяти одна з одною. API визначає, як програмні компоненти повинні спілкуватися.

ORM (Object Relational Model) – технологія, яка дозволяє розробникам взаємодіяти з базою даних за допомогою об’єктів у коді замість SQL-запитів.

Хешування (hashing) – це процес перетворення будь-яких вхідних даних у фіксовану послідовність символів (хеш), яка унікально представляє ці дані.

Salting – додавання випадкових даних (солі) до пароля перед хешуванням, щоб захистити його від атак типу перебору (brute force) і атак із заздалегідь обчисленими хешами (rainbow tables).

UUID (Universally Unique Identifier) – 128-бітний ідентифікатор, який використовується для однозначної ідентифікації об'єктів у системах.

HTML (HyperText Markup Language) – мова розмітки, яка використовується для створення структури та змісту вебсторінок. Вона визначає елементи, такі як заголовки, абзаци, посилання, зображення та таблиці.

CSS (Cascading Style Sheets) – мова стилів, яка використовується для оформлення зовнішнього вигляду HTML-елементів на вебсторінці. Вона визначає кольори, шрифти, розміри, розташування та інші візуальні властивості елементів.

Відмальовування (рендеринг) – процес перетворення HTML, CSS і JavaScript у візуальне зображення вебсторінки, яке бачить користувач у браузері. Існують клієнтський та серверний типи рендерингу.

REST (Representational State Transfer) – архітектурний стиль для створення вебсервісів, який використовує стандартні HTTP-методи (GET, POST, PUT, DELETE) для взаємодії з ресурсами, представленими у вигляді URL.

URL (Uniform Resource Locator) – уніфікована адреса, яка вказує розташування ресурсу в Інтернеті (наприклад, вебсторінки, зображення або файлу) та спосіб доступу до нього.

Шардінг – метод поділу і зберігання єдиного логічного набору даних у вигляді безлічі баз даних.

ВСТУП

Подорожі займають важливе місце в житті сучасних людей. За даними Всесвітньої туристичної організації (UNWTO), у 2022 році понад 900 мільйонів туристів здійснили міжнародні подорожі, що вдвічі більше, ніж у 2021 році, проте це тільки 63% від рівня, який був до пандемії [1].

Ділові поїздки також залишаються невід'ємною частиною професійної діяльності, адже компанії активно розширюють міжнародні зв'язки та співпрацю. Разом із зростанням мобільності людей збільшується й потреба в ефективних цифрових інструментах для планування та організації подорожей.

На ринку існує велика кількість застосунків, що допомагають мандрівникам вирішувати окремі завдання: бронювати квитки та житло, створювати маршрути, контролювати бюджет, складати списки необхідних речей тощо. Проте ці сервіси переважно спеціалізуються на одному аспекті подорожі, змушуючи користувачів перемикатися між різними платформами. Відсутність єдиного комплексного рішення ускладнює процес підготовки, підвищує ризик втрати важливих даних і створює додаткові незручності під час поїздки.

З огляду на це, актуальність розроблення програмного продукту, який дозволяє вирішувати ключові завдання планування подорожі в одному місці, є надзвичайно високою. У рамках дипломного проєкту буде створено мобільний застосунок, що надасть мандрівникам зручний інструмент для організації поїздок. Він дозволить складати списки необхідних речей, контролювати витрати, створювати розклад подорожі, зберігати важливі документи в електронному вигляді та швидко отримувати до них доступ. Завдяки цьому користувачі зможуть спростити процес підготовки, зменшити ризики, пов'язані з втратою важливих даних, та зробити свою подорож більш комфортною та впорядкованою.

1. ОГЛЯД ТА АНАЛІЗ ІСНУЮЧИХ ПРОГРАМНИХ РІШЕНЬ

У цьому розділі пояснювальної записки представлено інформацію про проблеми, з якими можуть зіткнутися мандрівники під час підготовки до подорожі, а також проведено аналіз існуючих програмних рішень, що надають подібну функціональність.

Нижче подано перелік завдань, що будуть виконані в межах цього розділу:

1. Аналіз ринку та визначення основних проблем, з якими стикаються подорожуючі під час організації та підготовки до поїздки.
2. Дослідження найбільш популярного програмного забезпечення, що використовується мандрівниками для планування подорожей та підготовки до перевезень, проведення порівняльного аналізу цих продуктів та визначення їхніх переваг.
3. Створення порівняльної таблиці з результатами аналізу програмних рішень.
4. Формулювання необхідної функціональності, яку має забезпечувати розроблюване програмне забезпечення.

1.1. Огляд проблеми, яка вирішується ПЗ

Подорожі – як туристичні, так і ділові – зазвичай охоплюють кілька ключових етапів, серед яких особливе значення мають планування та підготовка. На етапі підготовки формується загальний маршрут, визначаються пункти призначення, місця для відпочинку, пересадок тощо. Важливою складовою цього процесу є складання детального розкладу, що дозволяє ефективно розподілити час і забезпечити реалізацію всіх запланованих заходів.

Окрему увагу під час підготовки слід приділяти формуванню списку необхідних речей. Комплектація багажу залежить від цілей подорожі, її тривалості, кліматичних умов у пункті призначення, а також характеру

запланованої активності. До обов’язкових категорій можуть належати: одяг, засоби особистої гігієни, ліки, електроніка, дорожні аксесуари тощо. Відсутність важливих речей може значно ускладнити подорож, зробити її незручною або навіть небезпечною.

Ще одним критично важливим аспектом є зберігання супровідних документів, зокрема: квитків, підтверджень бронювання житла та транспорту, страхових полісів, візової інформації. У практиці користувачів ці документи часто зберігаються у різних джерелах – електронних листах, скриньках пошти, нотатках тощо, що ускладнює їх пошук і використання в момент потреби.

Усі вищезазначені компоненти організації подорожі тісно пов’язані з фінансовими витратами, які можуть суттєво варіюватися залежно від типу подорожі та її тривалості. Тому контроль і прогнозування бюджету є невід’ємною частиною підготовки. Завчасне планування витрат дозволяє уникнути перевитрат, оптимізувати використання фінансових ресурсів і знизити ризики непередбачених ситуацій.

Недостатньо ретельне планування або помилки в організації подорожі можуть призвести до значних ускладнень – від зіпсованих вражень до зриву важливих ділових зустрічей і фінансових збитків. Щоб уникнути таких наслідків, мандрівник повинен мати інструмент, який дозволяє відстежувати необхідні та вже зібрані речі, складати структурований розклад з активностями, зберігати й швидко знаходити усі необхідні документи, контролювати та планувати витрати.

Метою даного дипломного проєкту є спрощення процесу підготовки до поїздки для мандрівників шляхом надання мобільного застосунку, що забезпечить комплексну підтримку на даному етапі. Застосунок об’єднуватиме функціональність для складання розкладу, ведення списків речей, управління бюджетом, а також централізованого зберігання електронних документів, що дозволить мінімізувати організаційні труднощі та пришвидшити планування поїздок.

Для досягнення мети складено список задач, які потрібно виконати:

1. Аналіз ринку ПЗ для подорожування та організації перевезень.
2. Порівняння конкурентів, виявлення їхніх переваг та недоліків.
3. Визначення функціональних можливостей системи, які будуть потрібні потенційним користувачам.
4. Аналіз засобів для розробки ПЗ та обрання стеку технологій.
5. Програмна реалізація застосунку.
6. Тестування ПЗ.

1.2. Аналіз існуючих рішень

Аналіз існуючих рішень дозволяє глибше зрозуміти предметну галузь, оцінити сильні та слабкі сторони конкурентів, а також визначити стратегічні напрями розвитку власного програмного продукту.

Першим етапом цього процесу є підготовка, зокрема пошук застосунків, що відповідають основним потребам потенційних користувачів. Розглянемо наступні рішення: PackPoint [2], TripIt [3], Notion [4].

1.2.1. PackPoint

PackPoint – це мобільний органайзер для складання списку речей, призначений для мандрівників. Застосунок допомагає користувачам визначити, які саме речі необхідно взяти із собою в подорож, з урахуванням її тривалості, погодних умов у пункті призначення та запланованих активностей [2]. Рішення реалізоване у вигляді мобільного застосунку, доступного для платформ iOS та Android, і розповсюджується у двох версіях – безоплатній та платній. Платна версія відрізняється розширеною функціональністю, що включає додаткові можливості персоналізації та збереження списків [2]. Процес створення списку починається з введення користувачем основних даних про подорож: пункту призначення, дати виїзду, тривалості поїздки, а також її типу (ділова, туристична тощо). Далі

користувач обирає види активностей, які планується здійснювати під час подорожі (наприклад, фотографія, альпінізм, ділові зустрічі).

На основі вказаних параметрів, а також прогнозу погодних умов у заданий період, застосунок автоматично генерує список рекомендованих речей, розділених за категоріями. До кожної категорії (одяг, гігієнічні засоби, техніка тощо) можуть бути додані власні позиції, що дозволяє адаптувати список відповідно до індивідуальних потреб користувача.

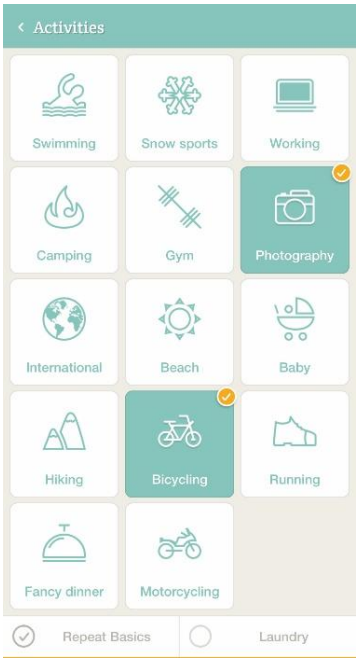


Рис. 1.1. Екран вибору занять

У межах функціональності застосунку реалізовано можливість інтерактивного керування списком речей, необхідних для подорожі. Користувач має змогу відмічати вже зібрані предмети простим натисканням на відповідний пункт у списку. Після позначення елемент автоматично приховується з активного переліку, що дозволяє уникнути зайвої візуальної перевантаженості та зосередитися на речах, які ще не були упаковані.

Керування кількісними показниками реалізовано за допомогою інтуїтивно зрозумілого інтерфейсу: праворуч від кожної позиції розташовані кнопки «+» та «−», що відповідають за збільшення або зменшення кількості

відповідного елементу в списку. Такий підхід забезпечує швидке коригування без потреби вводити дані вручну.

Рис 1.2. Переліки речей

Сильними сторонами PackPoint можна назвати простоту використання; завдяки інтуїтивно зрозумілому інтерфейсу, користувач легко розбереться у програмі та скористається її функціями. Також застосунок пропонує досить широку кастомізацію у вигляді власних позицій, шаблонів та списків, що дозволяє створювати списки для будь-яких потреб.

Недоліками застосунку є його обмежена функціональність загалом; на відміну від багатьох інших рішень, він не надає можливостей по складанню розкладу подорожі, зберігання файлів з документами, тощо. Також слабкою стороною є обмежена кількість платформ, через що використання застосунку є неможливою на деяких платформах. Та не варто забувати про малу кількість оновлень даного рішення через що воно може піддатися загрозам безпеки.

1.2.2. TripIt

TripIt – це мобільний та вебзастосунок для планування подорожей, який дозволяє організовувати маршрути, бронювання та інші важливі аспекти подорожі в одному місці. Він допомагає користувачам зберігати важливу інформацію про поїздку, таку як квитки, бронювання готелів, оренду автомобілів, трансфери та багато іншого [3].

Рішення представлене у вигляді мобільного застосунку для платформ на iOS та Android, та версії для веббраузерів [3].

Ключовим елементом функціонування застосунку є створення об’єктів типу «подорож», які користувач ініціює через головне меню. Для кожної подорожі необхідно вказати основні параметри: назву, пункт призначення, дату початку та завершення поїздки, а також за потреби – додатковий опис. Уся основна функціональність застосунку реалізується в межах конкретної подорожі, що дозволяє забезпечити логічну ізоляцію даних та зручну навігацію.

Планування подорожі розпочинається з додавання «планів» – окремих активностей або подій, які можуть мати різний характер: від ділових зустрічей до розважальних заходів, таких як екскурсії, відвідування концертів, поселення в готель тощо. Застосунок пропонує шаблони найпоширеніших типів активностей, які дозволяють спростити введення інформації, актуальної саме для вибраного типу плану.

Під час створення активності користувач заповнює ключові параметри, серед яких: дата й час початку та завершення події, адреса проведення, контактна інформація організаторів, додаткові примітки тощо. Інтерфейс адаптується відповідно до типу активності, дозволяючи вводити лише релевантні поля.

Після створення плану користувач отримує доступ до детального перегляду: включаючи часові межі виконання, географічне розташування на інтегрованій мапі, контактні дані та додаткову інформацію, що полегшує орієнтування та комунікацію. До кожної активності можна додати

супровідну документацію, зокрема квитки, підтвердження бронювання, маршрути, візові документи тощо. Завдяки цьому усі необхідні матеріали зберігаються у структурованому вигляді без потреби в пошуку сторонніми засобами.

Таким чином, структура подорожі в межах застосунку забезпечує централізоване керування усіма аспектами планування, сприяючи зручності користувача та підвищенню ефективності організації поїздки.

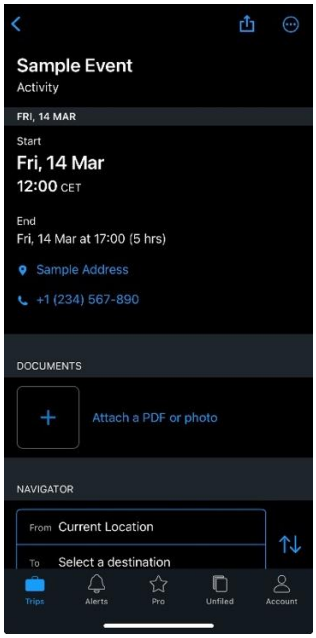


Рис. 1.3. Вікно події

Увесь розклад подорожі відображається на головній сторінці відповідної події у вигляді вертикальної послідовності дат, під якими згруповано заплановані на кожен день активності. Такий формат подання дозволяє користувачеві швидко зорієнтуватися у плані поїздки, переглянувши всі заходи, що мають відбутися впродовж кожного конкретного дня.

Події візуалізуються у вигляді структурованих блоків, які містять ключову інформацію: час початку, місце проведення та стислі деталі щодо самої активності. Кожен блок супроводжується інтуїтивно зрозумілою іконкою, а також кольоровим маркером відповідно до типу події (наприклад,

транспорт, харчування, відпочинок, екскурсія тощо), що дає змогу користувачеві оперативно ідентифікувати категорію активності без потреби відкривати її.

Після натискання на блок події відкривається розширене вікно з докладною інформацією: повний опис, контактні дані організаторів, посилання на електронні квитки, інтеграцію з картою маршруту тощо. При необхідності користувач має змогу змінювати дату чи час проведення події шляхом перетягування елементів у межах інтерфейсу. Усі зміни автоматично синхронізуються з персональним календарем користувача, що забезпечує актуальність розкладу на всіх пристроях.

Для підвищення зручності взаємодії реалізовано систему фільтрації, яка дозволяє відображати лише події певного типу або на вибрані дати, що сприяє кращій навігації в розкладі навіть за наявності великої кількості активностей. Додатково користувач може скористатися функцією пошуку за ключовими словами, що дозволяє швидко знаходити потрібні події за назвою, описом або місцем проведення. Таким чином, інтерфейс забезпечує не лише візуальну привабливість, а й високу ефективність у щоденному використанні, що особливо важливо в умовах насиченого графіка подорожі.

Рис. 1.4. Екран розкладу

Серед основних переваг TripIt варто відзначити низку функціональних рішень, які значно спрощують процес планування подорожей. Застосунок підтримує автоматичний імпорт підтверджень бронювань з електронної пошти, що дозволяє швидко сформувати маршрут без потреби у ручному введенні даних. Усі відомості про поїздку зберігаються в одному місці, що забезпечує зручний доступ до розкладу. Крім того, TripIt інтегрується з популярними календарями (наприклад, Google Calendar), підтримує офлайн-доступ до збереженої інформації та надає можливість ділитися маршрутом з іншими користувачами.

Для користувачів, які потребують розширеної функціональності, доступна платна версія TripIt Pro, яка включає додаткові можливості: сповіщення про зміни у розкладі рейсів, моніторинг статусу рейсів у реальному часі, а також інші інструменти для підвищення зручності подорожі [3].

Водночас застосунок має певні недоліки. Безкоштовна версія є функціонально обмеженою, що знижує її привабливість для користувачів, які очікують повного набору можливостей без додаткових витрат. Персоналізація інтерфейсу є мінімальною, що ускладнює адаптацію застосунку до індивідуальних потреб. Прив’язка до електронної пошти як основного джерела даних може створювати ризики у разі несанкціонованого доступу до поштової скриньки, оскільки це може призвести до компрометації конфіденційної інформації. Крім того, TripIt має обмежену інтеграцію з іншими сервісами та більше орієнтований на класичні туристичні подорожі, що робить його менш придатним для спеціалізованих поїздок (наприклад, експедицій, бізнес-відряджень з нестандартним графіком тощо).

1.2.3. Notion

Notion – це застосунок для створення систем управління, ведення нотаток, організації даних, проєктів та планування життя. Його

функціональність охоплює створення записів, канбан-дошок, вікі, календарів та нагадувань. Компанія позиціонує свій продукт як універсальний простір для організації нотаток, управління проєктами та завданнями [4].

Для планування подорожей у Notion користувач може створити окрему сторінку – одну з основних концепцій застосунку; платформу, на якій розміщується зміст. Далі вказується її заголовок, опис та важливі дати, такі як початок і кінець поїздки. Користувач може додати календар для відображення ключових подій, наприклад, бронювання квитків, зустрічей чи інших важливих моментів. Замість того, щоб створювати один великий список завдань, можна організувати завдання за допомогою канбан-дошки або чек-листів, що дає змогу чітко розподілити відповідальність і дедлайни [4].

Notion має кілька переваг для планування подорожей. Однією з основних переваг є гнучкість і можливість налаштувати структуру сторінки для будь-якої подорожі. Користувачі можуть додавати різні елементи, такі як таблиці, канбан-дошки та календарі, що дозволяє організувати планування відповідно до своїх потреб. Інтерактивність інтерфейсу також є великою перевагою, адже всі дані про подорож можна відображати в одному місці, легко оновлювати і переглядати.

Незважаючи на численні переваги, Notion має й кілька недоліків. Для повноцінного використання необхідно мати стабільне підключення до інтернету, оскільки програма не підтримує офлайн-доступ до даних [5]. Також новим користувачам може бути складно швидко освоїти всі можливості Notion через його велику кількість функцій і налаштувань, що створює певну криву навчання. Крім того, для більш складних задач, таких як інтеграція з іншими сервісами або автоматизація процесів, Notion може бути менш потужним, ніж спеціалізовані додатки для подорожей. Як і решта застосунків, Notion не може запрононувати весь спектр функцій, що вимагаються.

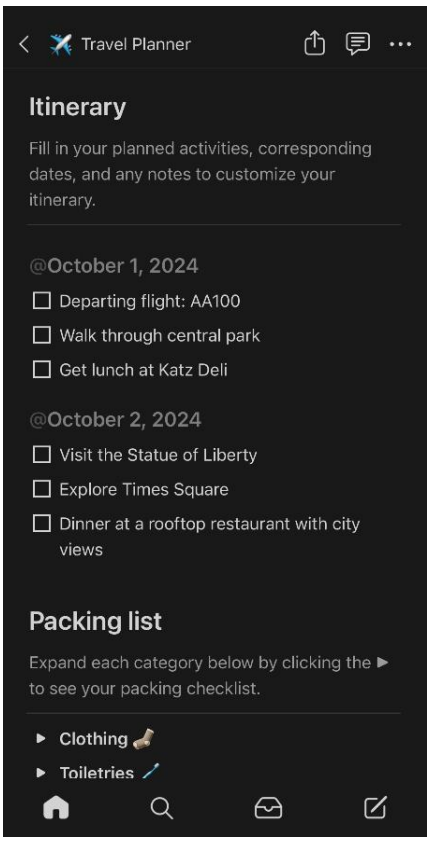


Рис. 1.5. Вікно сторінки

1.3. Результати аналізу

Після розглядання кількох популярних конкурентів, було вирішено сформувати критерії порівняння на основі вимог до програмного продукту даного дипломного проєкту:

1. Збирання речей. Система надає певну функціональність для створення персоналізованих списків речей для різних занять та відслідковування вже зібраних.
2. Управління бюджетом. Система надає функціональність для оцінювання грошових витрат по різним категоріям, наприклад, статті витрат.
3. Організація маршруту. Система дозволяє створити розклад подорожі із зазначенням усіх майбутніх заходів, їхнього порядку та часових рамок.
4. Зберігання документів. Система надає сховище для зберігання документів на різні заходи для їх подальшого використання.

5. Робота в офлайні. Система надає користувачу свою повну функціональність, або більшу її частину, без під’єднання до Інтернету.

Результат порівняння вище описаних програмних рішень за визначеними критеріями наведено у табл. 1.1.

Таблиця 1.1

Порівняльна характеристика існуючих рішень

Критерій	Аналоги		
	PackPoint	TripIt	Notion
Збирання речей	+	–	+
Управління бюджетом	–	–	–
Організація маршруту	–	+	+
Зберігання документів	–	+	–

На основі проведеного порівняльного аналізу трьох застосунків для планування подорожей можна зробити висновок, що кожен із них має специфічні функціональні особливості, що робить їх корисними для вирішення окремих завдань, але недостатніми у контексті комплексного підходу до організації подорожі.

PackPoint орієнтований передусім на створення списків речей для подорожі. Він враховує тривалість перебування, погодні умови у пункті призначення та заплановані види активності. Застосунок дозволяє персоналізувати список, додаючи власні елементи, що є зручним для індивідуального користувача. Проте його функціональність обмежена: відсутні засоби для планування маршруту, управління бюджетом чи зберігання документів, що знижує його придатність для повноцінного використання в процесі комплексного планування подорожі.

TripIt забезпечує зручну організацію маршруту, дає змогу зберігати квитки, бронювання, документи, а також переглядати ці дані в офлайн-режимі – це є суттєвою перевагою під час подорожей у місцевості з нестабільним або відсутнім інтернет-з'єднанням. Проте TripIt не надає можливостей для формування пакувального списку чи контролю витрат, що обмежує його функціональність у частині персонального планування.

Notion відзначається високим рівнем гнучкості та налаштувань. У ньому можна створювати власні бази даних, сторінки, списки речей і таблиці для маршруту. Однак, цей застосунок не має вбудованої підтримки офлайн-режиму, складний у налаштуванні для користувачів без попереднього досвіду, і не передбачає автоматичної інтеграції з туристичними сервісами, що ускладнює процес імпорту квитків або бронювань. Крім того, Notion не містить спеціалізованих функцій для управління бюджетом або пакування речей.

У підсумку, аналіз продемонстрував відсутність універсального рішення, яке б одночасно охоплювало всі аспекти планування подорожі – від організації маршруту до складання пакувальних списків і обліку витрат. Це стало підґрунтям для прийняття рішення про розроблення власного застосунку, який би задовольняв поставлені вимоги дипломного проєкту та поєднував найкращі функціональні риси аналізованих рішень, усуваючи їхні недоліки.

1.4. Висновки до першого розділу

Під час підготовки матеріалів до першого розділу дипломного проєкту було виконано наступні задачі:

1. Проаналізовано ринок ПЗ для подорожування.
2. Встановлено основні проблеми, з якими стикаються користувачі даного забезпечення.
3. Проаналізовано популярні серед подорожуючих системи для планування та підготовки до подорожей.

4. Сформовано перелік критеріїв порівняння систем у вигляді функціональних та нефункціональних вимог.
5. Побудовано порівняльну таблицю результатів аналізу рішень.
6. Виконано порівняльний аналіз існуючих програмних рішень із вказанням на переваги та недоліки кожного з них.

2. ВИБІР ЗАСОБІВ РЕАЛІЗАЦІЇ

Розроблюваний застосунок базується на клієнт-серверній архітектурі, яка передбачає розподіл програмного забезпечення на дві функціональні частини: клієнтську та серверну.

Клієнтська частина відповідає за взаємодію з користувачем, зокрема за відображення інтерфейсу, приймання введених даних та оновлення відображення відповідно до отриманих від сервера відповідей. Серверна частина виконує обробку запитів, що надходять від клієнта, реалізуючи логіку відповідно до функціональних вимог застосунку. У процесі обробки сервер має змогу звертатися до системи керування базами даних, що дозволяє здійснювати необхідні операції над даними. СКБД забезпечує керування даними, обробку запитів з боку сервера та повернення відповідних результатів.

Такий підхід до організації архітектури забезпечує гнучкість системи, оскільки клієнтська й серверна частини є технологічно незалежними. Кожен компонент може змінюватися або доповнюватися без необхідності модифікації іншого, за умови збереження узгодженого формату обміну даними між ними. Крім того, поділ на окремі компоненти підвищує загальний рівень безпеки системи: у разі компрометації одного з компонентів це не призведе до порушення функціонування решти системи.

З урахуванням обраного архітектурного підходу, під час розробки необхідно визначити мову програмування для реалізації серверної частини, засіб створення клієнтського інтерфейсу, а також відповідну систему керування базами даних.

2.1. Мова програмування для серверної частини

Для мови програмування було сформовано наступні вимоги:

1) підтримка паралельного програмування;

2) великий вибір бібліотек;

3) можливість використання для повного стеку;

4) зручне керування залежностями;

5) підтримка модульності.

2.1.1. Java

Java – це одна з найпопулярніших і найпоширеніших мов програмування, яку розробила компанія Sun Microsystems у 1995 році [5]. Головна ідея, що лежала в основі створення Java, – забезпечити платформонезалежність програм: тобто можливість писати програму один раз і запускати її на будь-якому пристрої без потреби вносити зміни в код. Цю властивість забезпечує спеціальне середовище виконання – Java Virtual Machine (JVM), яка дозволяє запускати байт-код Java на будь-якій операційній системі [6].

Мова має класичну об’єктно-орієнтовану модель, у якій усе будується на поняттях класів, об’єктів, спадкування та поліморфізму. Java є сильно типізованою мовою, що підвищує безпеку виконання програм і дозволяє уникати типових помилок на етапі компіляції. Особливістю Java є автоматичне управління пам’яттю – так званий збирач сміття (Garbage Collector), який самостійно видаляє непотрібні об’єкти з оперативної пам’яті, що суттєво зменшує ризик витоків пам’яті [6].

Завдяки своїй універсальності та стабільності, Java активно використовується у великих корпоративних системах, банківських платформах, серверних вебдодатках, вбудованих рішеннях, а також у мобільній розробці, особливо для Android [7]. Для розробки веб- і серверних застосунків найчастіше застосовуються фреймворки Spring, Hibernate, Jakarta EE та інші [7]. Java підтримує багатопотокове програмування, що дозволяє ефективно працювати з паралельними процесами, що особливо важливо для високонавантажених систем.

Попри всі переваги, Java має свої недоліки. Вона вважається досить багатослівною, що призводить до написання великої кількості коду для

вирішення відносно простих задач. Також продуктивність Java зазвичай поступається мовам, які компілюються безпосередньо у машинний код, як-от C чи C++. Проте завдяки постійним оновленням мова залишається актуальною: сучасні версії, починаючи з Java 17 та Java 21, отримали низку зручних нововведень, серед яких record-класи, sealed-класи, зіставлення зі шаблоном та віртуальні потоки (virtual threads) [6].

2.1.2. Python

Python – це високорівнева мова програмування загального призначення, яку розробив Гвідо ван Россум у кінці 1980-х років, а офіційно представив у 1991 році [8]. Метою створення Python було забезпечення максимальної простоти синтаксису та читабельності коду, щоб навіть початківці могли легко засвоїти програмування.

Python є інтерпретованою мовою, тобто програми на ній виконуються без попередньої компіляції в машинний код, а безпосередньо за допомогою інтерпретатора. Це дає змогу швидко запускати і тестувати програми, спрощує процес розробки та усуває необхідність у тривалих етапах компіляції [9]. Python підтримує кілька парадигм програмування: об’єктно-орієнтовану, процедурну та функціональну, що робить її надзвичайно гнучкою та універсальною для різних завдань.

Однією з найсильніших сторін Python є його простий і зрозумілий синтаксис, максимально наближений до природної англійської мови. Це робить мову ідеальним вибором для новачків, а також для проєктів, де важлива швидкість розробки та легкість супроводу коду. Завдяки багатій екосистемі бібліотек і фреймворків, Python широко використовується у веброзробці (Django, Flask), аналізі даних (Pandas, NumPy), машинному навчанні (TensorFlow, scikit-learn), автоматизації, написанні скриптів та тестуванні програмного забезпечення [10].

Попри численні переваги, Python має і свої недоліки. Основний із них це помірна продуктивність у порівнянні з компільованими мовами, такими

як C чи Java, через особливості інтерпретації коду та наявність глобальної блокування потоків (Global Interpreter Lock, GIL) [11], що обмежує ефективну багатопотоковість. Водночас завдяки можливості інтегрувати модулі на C/C++ та використанню асинхронного програмування, ці недоліки в багатьох випадках можна компенсувати.

2.1.3. JavaScript

JavaScript – це високорівнева мова програмування, створена для розробки інтерактивних вебсторінок. Її автор – Брендан Айк, який розробив першу версію JavaScript у 1995 році всього за 10 днів у компанії Netscape Communications. Спочатку мова мала назву Mocha, згодом її перейменували у LiveScript, а після маркетингового ходу – в JavaScript, хоча з Java ця мова має спільного лише деякі синтаксичні елементи [12]. Основна мета мови – забезпечення інтерактивності у браузері без потреби перезавантаження сторінки.

JavaScript є інтерпретованою мовою, яка виконується безпосередньо в браузері за допомогою JavaScript-движка (наприклад, V8 у Chrome, SpiderMonkey у Firefox чи WebKit у Safari) [12]. Згодом, завдяки технологіям на кшталт Node.js, JavaScript вийшов за межі браузера та почав активно використовуватися і на стороні сервера. Це зробило його унікальною мовою, яка дозволяє писати повноцінні клієнт-серверні додатки єдиною мовою. На сьогодні JavaScript є де-факто стандартом для фронтенду, а завдяки Node.js і важливим інструментом бекенд-розробки [13].

Основними перевагами JavaScript є гнучкість синтаксису, підтримка об’єктно-орієнтованого, функціонального та подієвого програмування, а також можливість створювати динамічні, швидкі та інтерактивні інтерфейси. Завдяки величезній кількості бібліотек і фреймворків, таких як React, Vue.js, Angular та Svelte [13], розробники можуть швидко створювати складні вебзастосунки. Також популярні серверні рішення на

JavaScript: Node.js, Express, NestJS [13], що дозволяють створювати API, вебсервери та мікросервіси.

Попри всі переваги, JavaScript має і свої недоліки. Через динамічну типізацію можливі неочевидні помилки під час виконання коду, які складно виявити на етапі розробки. Історично JavaScript мав неоднозначну репутацію через відсутність суворих стандартів, однак із появою ECMAScript (ES6 і вище) мова суттєво покращилася [14]. Ще однією проблемою є асинхронна природа виконання JavaScript, яка потребує особливого підходу до роботи з подіями та мережевими запитами.

На основі детального аналізу та всебічного порівняння ключових характеристик кожної з розглянутих мов програмування – включаючи їх синтаксис, модель управління пам’яттю, продуктивність у різних сценаріях, набір стандартних і сторонніх бібліотек, наявність розвиненої екосистеми інструментів – ми підготуємо вичерпну порівняльну таблицю.

Таблиця 2.1

Порівняльна таблиця мов програмування для серверної частини

Критерій	Мови програмування		
	Java	Python	JavaScript
Підтримка одночасного програмування	+	+/–	+
Великий вибір бібліотек	+	+	+
Можливість використання для повного стеку	–	–	+
Зручне керування залежностями	+/–	+	+
Підтримка модульності	+	+	+

Проаналізувавши можливості трьох популярних мов програмування – Java, Python та JavaScript, можна зробити висновок, що кожна з них має свої сильні та слабкі сторони залежно від конкретних потреб проєкту.

Java залишається надійним вибором для систем із високими вимогами до паралельної обробки даних і суворої структуризації коду. Вона добре підтримує багатопотоковість і модульність, має велику екосистему бібліотек. Проте для повного стеку вона не підходить, тому переважно використовується на серверній стороні. Керування залежностями через Maven або Gradle хоч і надійне, проте дещо громіздке порівняно з сучасними менеджерами інших мов.

Python вирізняється максимальною універсальністю та простотою. Він має велику екосистему бібліотек і добру підтримку модульності. Його слабким місцем залишається багатопотоковість через обмеження GIL, хоча асинхронне програмування компенсує цей недолік у багатьох сценаріях. Для повного стеку Python не використовується, а інтеграція з фронтендом є доволі експериментальною. Натомість керування залежностями – зручне та легке завдяки pip, virtualenv, poetry.

JavaScript демонструє найбільшу гнучкість і універсальність у контексті веброзроблення. Це єдина мова, яка повноцінно використовується як на клієнтській, так і на серверній стороні. Вона добре підтримує одночасне виконання завдань завдяки подієво-орієнтованій моделі та механізмам асинхронності. JavaScript має зручне керування залежностями через npm/yarn/pnpm і відмінну підтримку модульності. Ця мова ідеально підходить для створення повного технологічного стеку.

2.2. Засіб створення клієнтського інтерфейсу

Для засобу розроблення клієнтської частини було сформовано наступні вимоги:

1) підтримка компонентної архітектури;

2) читабельність коду;

3) гнучкість підходу до розроблення;

4) великий вибір допоміжних засобів та бібліотек.

2.2.1. Vue.js

Vue.js – це прогресивний JavaScript-фреймворк для створення користувацьких інтерфейсів, який з'явився у 2014 році завдяки Евану Ю, колишньому розробнику Google [15]. Ідея Vue полягала в тому, щоб поєднати найкращі рішення з існуючих фреймворків на кшталт Angular і React, зберігаючи при цьому простоту та гнучкість. Однією з головних переваг Vue є його прогресивність: його можна легко впровадити у вже існуючий проєкт, поступово збільшуючи кількість інтерактивних компонентів, або ж побудувати на його основі повноцінний односторінковий застосунок [16].

Фреймворк базується на концепції реактивності. Його система реактивних даних автоматично відслідковує зміни у стані застосунку та оновлює лише ті частини DOM, де відбулися зміни [16]. Починаючи з Vue 3, ця система побудована на базі обгортання даних у Proxy-об’єкти, що дозволяє підвищити продуктивність та уникнути обмежень попереднього підходу через Object.defineProperty [17]. Vue реалізує архітектурний шаблон MVVM (Model-View-ViewModel) і пропонує простий декларативний синтаксис шаблонів, підтримку власних директив, двосторонню прив'язку даних (data binding) та гнучкі механізми керування компонентами [16].

З технічного боку Vue вирізняється продуманою структурою. Кожен компонент у ньому має свій життєвий цикл, який включає етапи створення, рендерингу, оновлення та знищення. Компоненти можуть містити власні дані, обчислювані властивості (computed properties), спостерігачі (watchers) та обробники подій [16]. З виходом Vue 3 було представлено Composition API – новий підхід до організації компонентної логіки, який дозволяє об’єднувати функціональні частини застосунку за призначенням, а не за роллю, що особливо зручно для масштабних проєктів [17].

В екосистемі Vue є кілька ключових офіційних інструментів. Для маршрутизації використовується Vue Router, для централізованого сховища стану – Pinia (нова альтернатива Vuex), а для збірки застосунків – надшвидкий Vite, який замінив Vue CLI у сучасних проєктах. Крім цього, Vue має багату екосистему UI-фреймворків, серед яких популярні Vuetify, Element Plus, PrimeVue та Quasar [16].

Основними перевагами Vue є його гнучкість, простий синтаксис, висока продуктивність і низький поріг входу для початківців. Vue дозволяє інтегруватися в існуючі системи без кардинальної перебудови архітектури, що зручно для проєктів з поступовим переходом на сучасні фреймворки. Також Vue 3 має офіційну підтримку TypeScript, що робить його придатним для великих проєктів із суворими вимогами до типізації.

До недоліків можна віднести меншу популярність у корпоративному середовищі порівняно з React, а також відсутність жорстко стандартизованої структури великих проєктів – Vue дозволяє реалізовувати різні підходи, що може ускладнити масштабування командної роботи без чітко прописаних внутрішніх правил. Крім того, для реалізації серверного рендерингу зазвичай доводиться використовувати Nuxt.js, оскільки вбудованих рішень у Vue для цього немає.

2.2.2. Angular

Angular – це повноцінний фронтенд-фреймворк для створення масштабованих вебдодатків, який підтримується та розвивається компанією Google [18]. Варто зазначити, що сучасний Angular (починаючи з версії 2.0) суттєво відрізняється від AngularJS (версії 1.x) як за архітектурою, так і за підходами до побудови застосунків [18]. Angular створений для розробки великих корпоративних систем, де важливі сувора типізація, стандартизована структура проєкту, чітка модульність і ефективне керування залежностями.

Фреймворк базується на архітектурному шаблоні Component-Based Architecture, де весь застосунок складається з ієрархії компонентів. Усі компоненти в Angular суворо структуровані: кожен має власний HTML-шаблон, CSS-стилі та TypeScript-клас, який містить логіку. Angular активно використовує Dependency Injection (DI) – одну з ключових особливостей, що дозволяє зручно керувати залежностями між компонентами, сервісами та іншими модулями [19].

Однією з технічних особливостей Angular є його компілятор – Ahead-of-Time (AOT) Compilation. Ця технологія компілює TypeScript-код і шаблони компонентів ще на етапі збірки, до завантаження застосунку в браузері. Це дозволяє пришвидшити рендеринг, зменшити розмір застосунку та забезпечити вищу безпеку, оскільки шаблони вже не містять інтерпретованого коду під час виконання [18].

Angular надає двосторонню прив’язку даних (two-way data binding) через директиву [(ngModel)], яка синхронізує дані між шаблоном і компонентом. Система реактивних форм дозволяє будувати складні інтерактивні форми з валідацією. Ще одним важливим технічним інструментом є RxJS – бібліотека для роботи з потоками даних, що лежить в основі реактивного програмування в Angular. Вона дозволяє обробляти асинхронні події, HTTP-запити, а також реалізовувати багатопотокову взаємодію всередині застосунку [19].

Фреймворк має суворо визначену модульну архітектуру: проєкти розділяються на окремі Angular-модулі (NgModules), які логічно групують компоненти, сервіси, маршрути та інші об’єкти. Angular CLI – потужний офіційний інструмент для генерації коду, керування проєктами, налаштування збірки та розгортання. Angular також має офіційну систему маршрутизації, що підтримує lazy loading (ліниве завантаження модулів) та захист роутів через guard-и [19].

Екосистема Angular є дуже розвиненою та орієнтованою на корпоративний сектор. Популярні UI-фреймворки, такі як Angular Material і

NG Bootstrap, забезпечують готові компоненти для створення адаптивного дизайну [20]. Angular підтримує TypeScript за замовчуванням, що дозволяє створювати типобезпечний код, що особливо важливо для великих проєктів із великою кількістю командних учасників.

До переваг Angular належать висока продуктивність (завдяки AOT-компіляції), стабільна структура великих проєктів, розвинений механізм DI, реактивне управління даними через RxJS та сувора модульність. Angular також має чудову підтримку масштабованості та можливість створення серверно-рендерингових застосунків за допомогою Angular Universal.

Серед недоліків варто зазначити високу складність порівняно з Vue або React, особливо для початківців. Велика кількість концепцій, обов'язкова типізація та сувора структура проєкту потребують часу для опанування. Окрім того, через розвинену екосистему та наявність великої кількості пакетів і стандартів підтримка проєктів може бути громіздкою без чітких командних правил.

2.2.3. React

React – це бібліотека для створення користувацьких інтерфейсів, розроблена компанією Facebook. Вперше представлена в 2013 році, React швидко здобула популярність завдяки своєму простому, декларативному підходу до розробки UI [21]. Вона дозволяє створювати швидкі та інтерактивні вебдодатки, розділяючи інтерфейс на невеликі, незалежні компоненти. React фокусується лише на відображенні інтерфейсу (view) і не намагається охопити всі аспекти розробки вебдодатків, тому часто використовується разом з іншими бібліотеками для керування станом, маршрутизації тощо.

Однією з основних характеристик React є концепція компонентного підходу. Кожен компонент відповідає за свою частину інтерфейсу та має власну логіку й стиль, що дозволяє повторно використовувати їх у різних частинах застосунку [21]. Компоненти можуть бути функціональними або

класовими. Починаючи з React 16.8, основним способом створення компонентів є функціональні компоненти з використанням Hooks – нової концепції для управління станом, побічними ефектами та іншими функціональностями, яка зменшує необхідність у класах [21].

React використовує механізм віртуального DOM. Замість того, щоб безпосередньо оновлювати реальний DOM після кожної зміни, React спочатку оновлює віртуальний DOM, порівнює його з попереднім станом і лише потім мінімізує реальні зміни, які потрібно внести в DOM. Це дозволяє значно покращити продуктивність, особливо у великих застосунках з численними оновленнями інтерфейсу [21].

JSX (JavaScript XML) – це ще одна важлива частина React. JSX дозволяє писати HTML-подібний код у JavaScript-функціях, що робить синтаксис чистим і зрозумілим. Завдяки JSX ви можете легко маніпулювати компонентами та відображенням даних, при цьому React під час збірки перетворює цей JSX на звичайний JavaScript-код [21].

React підтримує одностороннє керування потоком даних (one-way data flow), де стан (state) зберігається у компонентах і передається вниз через властивості (props). Це дозволяє чітко відслідковувати, як і де відбуваються зміни, і робить відлагодження коду простішим. Для складних застосунків, де потрібно управління глобальним станом, часто використовуються додаткові бібліотеки, такі як Redux або Context API [21].

React Router – це основний інструмент для організації маршрутизації в односторінкових застосунках [22]. Для керування асинхронними запитами, зокрема, часто використовуються Axios або Fetch API. Для управління станом на клієнтській стороні через Redux або новіші підходи, такі як Recoil або Zustand, додається додатковий рівень абстракції, що допомагає зберігати та передавати дані між компонентами [22].

Екосистема React дуже велика та активна. Окрім основного пакету, існує безліч популярних бібліотек для створення компонентів UI, таких як Material-UI, Ant Design, React Bootstrap та інші [23]. Для повноцінного стеку

застосунків існують рішення, як-от Next.js (для серверного рендерингу та статичних сайтів) і Gatsby (для створення швидких статичних сайтів) [24].

Ще однією великою перевагою React є його гібридність. Замість того, щоб бути монолітним фреймворком, React дає можливість використовувати лише частини бібліотеки, інтегруючи її в проєкти разом з іншими технологіями. Це дозволяє легко комбінувати React з іншими фреймворками та бібліотеками, що дає велику свободу у виборі інструментів для проєкту.

Основними перевагами React є гнучкість, висока продуктивність завдяки віртуальному DOM, велика екосистема і здатність ефективно працювати як з невеликими проєктами, так і з великими застосунками. React популярний серед розробників завдяки простоті навчання, можливості використовувати функціональні компоненти та hooks, а також великій кількості ресурсів і підтримки від спільноти.

Проте React має й деякі недоліки. Одним із них є відсутність власного рішення для маршрутизації або управління станом, тому вам доведеться обирати сторонні бібліотеки, що збільшує складність інтеграції. Іншим мінусом є потреба в зовнішніх бібліотеках для роботи з більш складними функціями (наприклад, серверний рендеринг або статичні сайти), хоча існують інструменти, такі як Next.js, які полегшують ці завдання. Крім того, через широке використання JSX у React може бути трохи складно почати для тих, хто не знайомий з сучасним JavaScript.

Таблиця 2.2

Порівняльна таблиця засобів розроблення клієнтської частини

Критерій	Засоби		
	React	Vue	Angular
Підтримка компонентної архітектури	+	+	+
Читабельність коду	+	+	+/–

Продовження табл 2.2

Гнучкість підходу до розроблення	+	+	–
Великий набір допоміжних засобів та бібліотек	+	+/–	+

Усі три технології – React, Vue та Angular – мають свої унікальні переваги та підходи до розробки вебдодатків, що робить їх підходящими для різних типів проєктів. React пропонує велику гнучкість, дозволяючи вибирати найкращі бібліотеки для кожного конкретного завдання. Це робить його ідеальним для розробників, які хочуть повного контролю над кожним аспектом проєкту та готові інтегрувати сторонні рішення для маршрутизації, керування станом чи роботи з даними. React є зручним для невеликих та середніх проєктів, а також для створення односторінкових застосунків, де важливі швидка розробка та масштабованість.

Vue, з іншого боку, поєднує в собі елементи гнучкості та простоти. Завдяки зручному синтаксису, швидкому початку роботи і меншій складності в порівнянні з Angular, Vue є чудовим вибором для проєктів середнього масштабу. Vue дозволяє легко інтегруватися в існуючі проєкти і пропонує чудові можливості для створення інтерактивних інтерфейсів, не вимагаючи занадто багато налаштувань. Хоча Vue менш популярний у корпоративному секторі, його екосистема швидко розвивається, і він забезпечує ефективний процес розробки без великих витрат часу на налаштування.

Angular ж є найбільш обмеженим з трьох фреймворків в плані гнучкості, але це також є його великою перевагою в певних контекстах. Angular має чітко визначену структуру і архітектуру, що робить його ідеальним для великих, масштабованих проєктів, де важлива стандартизованість та керованість залежностями. Його використання в корпоративних рішеннях з багатьма командами дозволяє створювати складні та високопродуктивні додатки. Однак для менш досвідчених

розробників Angular може здатися занадто складним і важким для освоєння через велику кількість концепцій і правил, які необхідно враховувати під час розробки.

Загалом, React є найгнучкішим і найбільш популярним серед розробників завдяки своїй простоті, можливості інтеграції з іншими інструментами та швидкості розробки, та він краще всього підходить як засіб для розроблення клієнтської частини програмного продукту.

2.3. Вибір системи керування базою даних

Системи керування базами даних поділяються на два основні типи: реляційні та нереляційні. Реляційні СКБД організовують дані у вигляді таблиць (відношень), де кожен запис складається з набору атрибутів відповідно до визначених доменів. Дані в таких системах пов’язуються між собою за допомогою зовнішніх ключів, що забезпечує логічну цілісність інформації.

Для роботи з реляційними базами даних використовується мова структурованих запитів SQL, яка дозволяє виконувати основні операції над даними: створення, читання, оновлення та видалення (CRUD-операції).

Нереляційні СКБД, у свою чергу, реалізують альтернативні підходи до зберігання та обробки даних, які відрізняються від реляційної моделі. До цього типу належать, зокрема, документо-орієнтовані СКБД (наприклад, MongoDB), графові бази даних (наприклад, Neo4j), а також системи типу ключ-значення (наприклад, Redis). Такі системи часто позначають терміном NoSQL, що підкреслює відсутність використання традиційної мови SQL для виконання запитів і маніпулювання даними.

Враховуючи вимоги, поставлені до програмного продукту, СКБД обиратиметься за наступними критеріями:

1) захищеність даних;

2) підтримка реплікації;

3) підтримка індексації;

4) можливість використання неструктурованих типів даних;

5) відповідність вимогам ACID.

2.3.1. MongoDB

MongoDB – це популярна документо-орієнтована NoSQL база даних, яка зберігає дані у форматі BSON (Binary JSON), що є розширенням JSON. MongoDB була розроблена для зберігання великих обсягів структурованих і неструктурованих даних, а також для забезпечення високої масштабованості та продуктивності. Вона широко використовується для розробки вебдодатків, мобільних застосунків, а також для обробки великих даних в умовах високої навантаженості [25].

Основною характеристикою MongoDB є її гнучка схема. Оскільки дані зберігаються у вигляді документів, що представляють собою колекції JSON-подібних об'єктів, структура кожного документа може бути різною. Це дозволяє зберігати дані без жорсткої схеми, що особливо корисно при швидкій розробці, коли структура даних може змінюватися з часом. У порівнянні з реляційними базами даних, де таблиці мають фіксовану структуру, MongoDB дозволяє значно спростити і пришвидшити процес зберігання і маніпулювання даними [25].

Масштабованість MongoDB також є важливою перевагою. Вона підтримує горизонтальне масштабування, що дозволяє додавати нові сервери для обробки більшої кількості даних і запитів. Завдяки цьому MongoDB чудово підходить для застосунків з великими обсягами даних або для тих, що потребують високої доступності та швидкої обробки запитів. Вона підтримує автоматичне шардінгування даних, що розподіляє дані по кількох серверах, забезпечуючи рівномірне навантаження і високу доступність [25].

Запити в MongoDB можуть бути виконані через Mongo Query Language (MQL), що дозволяє виконувати різноманітні операції, такі як пошук, оновлення, видалення, агрегація та сортування. MongoDB також

підтримує потужну агрегацію через оператори, що дозволяють виконувати складні обчислення без необхідності витягувати дані в програму. Це дає можливість ефективно обробляти дані безпосередньо на рівні бази [25].

MongoDB забезпечує гнучкість у роботі з різними типами даних, такими як текст, числа, масиви та навіть файли через GridFS – спеціальну файлову систему для зберігання великих файлів. Вона добре підходить для проєктів, де потрібно зберігати не лише структуровані дані, але й файли, зображення, відео тощо [26].

Продуктивність MongoDB забезпечує високу швидкість завдяки використанню індексів для прискорення пошукових операцій. Вона також дозволяє зберігати дані в пам'яті для забезпечення швидкого доступу. Проте, на відміну від реляційних баз, MongoDB не підтримує складні транзакції, що може бути обмеженням для деяких типів застосунків, де важлива строгість і консистентність даних [26].

Велику роль у популярності MongoDB відіграє її проста інтеграція з іншими технологіями. Вона є чудовим вибором для стеків технологій, таких як MEAN (MongoDB, Express, Angular, Node.js) та MERN (MongoDB, Express, React, Node.js), де MongoDB використовує JavaScript як основну мову для запитів, що дозволяє зручно працювати з базою даних без необхідності вивчати нові мови чи інтерфейси.

Одним із недоліків MongoDB є відсутність строгих зв'язків між даними, як у реляційних базах даних. Це означає, що при складних запитах, які потребують об'єднання кількох колекцій, можуть виникати складнощі з виконанням операцій, оскільки MongoDB не підтримує JOIN операції так, як це роблять реляційні бази.

2.3.2. MySQL

MySQL – це одна з найпопулярніших реляційних баз даних, яка використовує Structured Query Language (SQL) для управління і маніпулювання даними. MySQL є частиною більш широкої екосистеми з

відкритим кодом і широко використовується для зберігання даних у вебдодатках, особливо в поєднанні з такими технологіями, як PHP або Python. MySQL відома своєю стабільністю, простотою у використанні та високою швидкістю для малих та середніх за розмірами застосунків [27].

Однією з основних характеристик MySQL є її реляційна структура, яка забезпечує чітке визначення таблиць з фіксованими схемами і зв'язками між ними. Це дозволяє організувати дані в таблиці, визначаючи стовпці з певними типами даних, що забезпечує цілісність і структуру даних. Такі БД особливо підходять для застосунків, які вимагають строгих правил збереження даних і підтримки транзакцій.

Масштабованість MySQL забезпечується за рахунок підтримки вертикального масштабування через додавання більше ресурсів на сервер або горизонтального масштабування через реплікацію та шардінг. Реплікація дозволяє створювати копії бази даних на декількох серверах для покращення доступності та продуктивності. Проте в порівнянні з деякими NoSQL базами даних, такими як MongoDB, горизонтальне масштабування MySQL може бути складнішим і вимагати додаткових налаштувань.

Однією з найбільших переваг MySQL є її підтримка транзакцій. Базуючись на ACID-принципах, MySQL гарантує, що дані будуть зберігатися цілісними навіть при відмовах системи, що робить її ідеальною для застосунків, де важливі консистентність і цілісність даних, наприклад, в онлайн-банкінгу чи електронній комерції. Вона також підтримує важливі механізми блокування даних для запобігання конкурентним оновленням.

Запити в MySQL виконуються через SQL, що робить роботу з базою даних зрозумілою і стандартною для більшості розробників. SQL – це потужний інструмент для маніпулювання даними, що дозволяє виконувати складні операції, такі як злиття таблиць (JOIN), фільтрація, сортування та агрегація. Зручність і потужність SQL дозволяють легко реалізовувати складні логічні запити для отримання і обробки даних, а також підтримують оптимізацію запитів для покращення продуктивності.

Продуктивність MySQL також є важливою характеристикою цієї СУБД. Вона може ефективно обробляти величезні обсяги даних, якщо правильно налаштовані індекси і виконується оптимізація запитів. MySQL підтримує різноманітні механізми зберігання даних, зокрема InnoDB, який забезпечує підтримку транзакцій і зовнішніх ключів, а також MyISAM, що оптимізує швидкість читання даних для більш легких завдань без транзакцій [28].

Великою перевагою є також спільнота MySQL, оскільки це одна з найбільш популярних баз даних у світі. Завдяки великій кількості ресурсів, документації та обговорень на форумах, початківці можуть швидко вирішувати проблеми, а досвідчені розробники можуть знайти оптимальні рішення для складних задач. Крім того, MySQL широко підтримується хмарними провайдерами, такими як Amazon Web Services (AWS), Google Cloud і Microsoft Azure, що дозволяє безперешкодно інтегрувати її в хмарні рішення.

2.3.3. PostgreSQL

PostgreSQL – це потужна, об'єктно-реляційна база даних з відкритим кодом, яка підтримує складні SQL-запити, транзакції та високу масштабованість. PostgreSQL відома своєю стабільністю, гнучкістю та багатофункціональністю, і є однією з найпопулярніших реляційних СУБД для застосунків, де важлива цілісність даних і підтримка складних запитів [29].

Гнучкість схеми та підтримка складних даних – це одна з головних переваг PostgreSQL. Ця база даних підтримує як традиційні реляційні дані, так і об'єктно-орієнтовані типи даних, такі як масиви, JSON, HSTORE, що дозволяє зберігати і маніпулювати різноманітними типами даних. PostgreSQL також підтримує розширення типів даних, що дає змогу створювати індивідуальні типи для специфічних потреб застосунку. Завдяки

цьому вона є дуже гнучкою і може використовуватися в багатьох різних контекстах [29].

Підтримка транзакцій та ACID-принципів – PostgreSQL дотримується стандарту ACID, що гарантує цілісність та надійність даних, навіть у разі системних збоїв чи помилок. Це робить PostgreSQL відмінним вибором для проєктів, де важливо забезпечити строгий контроль над даними, таких як фінансові системи або інші високонавантажені застосунки, де транзакції та консистентність мають критичне значення.

Однією з ключових особливостей PostgreSQL є її підтримка складних SQL-запитів. Вона підтримує всі стандартні SQL-операції, включаючи складні злиття (JOIN), агрегацію, підзапити, а також індекси для оптимізації запитів. Крім того, PostgreSQL підтримує потужну агрегацію даних та можливість створення складних запитів за допомогою функцій та процедур, що дає можливість вирішувати найрізноманітніші задачі. Це робить PostgreSQL ідеальним вибором для проєктів, де потрібно працювати з великими обсягами даних і виконувати складну аналітику.

Масштабованість та продуктивність PostgreSQL також на високому рівні. Вона підтримує горизонтальне масштабування через реплікацію та шардінг, що дозволяє розподіляти навантаження між кількома серверами для забезпечення високої доступності та покращення продуктивності. Завдяки можливості реплікації в реальному часі PostgreSQL може використовуватися в розподілених системах, що є важливим для великих корпоративних застосунків.

Ще однією важливою особливістю PostgreSQL є підтримка розширень. PostgreSQL має велику кількість офіційних і неофіційних розширень, таких як PostGIS для роботи з геопросторовими даними, повнотекстовий пошук для пошукових систем і hstore для зберігання пар ключ-значення, що значно розширює функціональні можливості бази даних. Крім того, PostgreSQL підтримує створення користувацьких функцій та

індексів, що дозволяє налаштувати базу даних під специфічні вимоги проєкту [30].

Продуктивність PostgreSQL забезпечується її підтримкою різних типів індексів, таких як B-tree, Hash, GIN, GiST та інших, що дозволяє оптимізувати пошук, оновлення та агрегацію даних. Вона також підтримує кластеризацію таблиць для забезпечення ефективного доступу до даних, що дозволяє досягти високої продуктивності навіть при роботі з великими наборами даних [31].

Безпека є ще одним важливим аспектом PostgreSQL. Вона підтримує SSL-шифрування, а також дозволяє здійснювати управління доступом через рольову модель з багаторівневими правами доступу. Це дозволяє детально налаштовувати рівні доступу для різних користувачів і забезпечувати надійний захист даних.

Таблиця 2.3

Порівняльна таблиця систем керування базою даних

Критерій	СКБД		
	MongoDB	MySQL	PostgreSQL
Захищеність даних	+/–	+	+
Підтримка реплікації	+	+	+
Підтримка індексації	+	+	+
Можливість використання неструктурованих типів даних	+	–	+
Відповідність вимогам ACID	+/–	+	+

PostgreSQL демонструє стабільно високі показники за всіма критеріями. Ця СКБД повністю відповідає вимогам ACID, пропонує надійну систему безпеки, чудову підтримку реплікації та індексації. До того ж, на відміну від класичних реляційних баз, PostgreSQL підтримує

неструктуровані типи даних – JSON, hstore, XML – що робить її надзвичайно універсальною для сучасних застосунків.

MySQL також показує хороші результати. Вона добре справляється з реплікацією, індексацією та захистом даних, а також цілком дотримується ACID-вимог. Однак її слабке місце – це відсутність повноцінної підтримки неструктурованих типів даних. Це може стати обмеженням для проєктів, які активно працюють із JSON чи іншими гнучкими форматами.

MongoDB, у свою чергу, найкраще себе проявляє у роботі з неструктурованими даними та масштабованою реплікацією. Вона поступається класичним реляційним СКБД у питаннях захисту та суворого дотримання ACID (хоч останні версії вже підтримують транзакції, але з деякими обмеженнями). MongoDB забезпечує гнучкість та швидкість для динамічних застосунків, але для критичних систем, де важлива повна цілісність даних, може бути менш бажаною.

2.4. Висновки до другого розділу

Під час підготовки матеріалів до другого розділу дипломного проєкту було виконано наступні задачі:

1) сформовано перелік основних вимог до засобів реалізації;
2) обрано найбільш популярні рішення, проведено аналіз, виділено їхні слабкі та сильні сторони;
3) побудовано порівняльні таблиці за сформованими критеріями;
4) обрано набір технологій за результатами порівняння.

3. СТРУКТУРНО-АЛГОРИТМІЧНА ОРГАНІЗАЦІЯ СИСТЕМИ

3.1. Опис варіантів використання

Для визначення функціональних вимог існує низка методів, зокрема користувацькі історії, сценарії використання, UML-діаграми тощо. Кожен із них має свої переваги та обмеження. Наприклад, сценарії описують послідовність дій, але не забезпечують візуалізації, що може ускладнити сприйняття. У межах даного проєкту було обрано діаграму варіантів використання (діаграму прецедентів), яка розробляється на етапі проєктування системи. Вона дозволяє наочно представити функціональність, узгодити технічне завдання, а також слугує основою для подальшого документування та тестування.

Передусім необхідно визначити прецеденти (варіанти використання системи) для кожної з ролей користувачів. У межах системи передбачено дві ролі: «Мандрівник» і «Адміністратор». Кожна з них має чітко окреслений набір функціональних можливостей, які не перетинаються між собою.

Розглянемо варіанти використання для обох ролей. Почнемо з ролі «Мандрівник», оскільки саме вона охоплює найширший спектр функціональних можливостей у системі:

1. Керування подорожами (створення, перегляд, видалення).
2. Керування подіями на кожен день подорожі (створення, перегляд, редагування, видалення).
3. Складання та заповнення персоналізованих списків речей.
4. Отримання статистики по витратам на основі коштів, затрачених на події та витратах, доданих власноруч (створення, редагування, видалення).
5. Можливість прикладання файлів до окремих подій та подорожей загалом.

Тепер розглянемо вимоги для користувача ролі «Адміністратор»; його функціональність є обмеженішою, але важливою:

1. Можливість перегляду статистики за Мандрівниками.
2. Можливість керування обліковими записами Мандрівників (створення, перегляд, видалення).

На основі наведених варіантів використання було побудовано діаграму прецедентів, представлену на рисунку нижче.

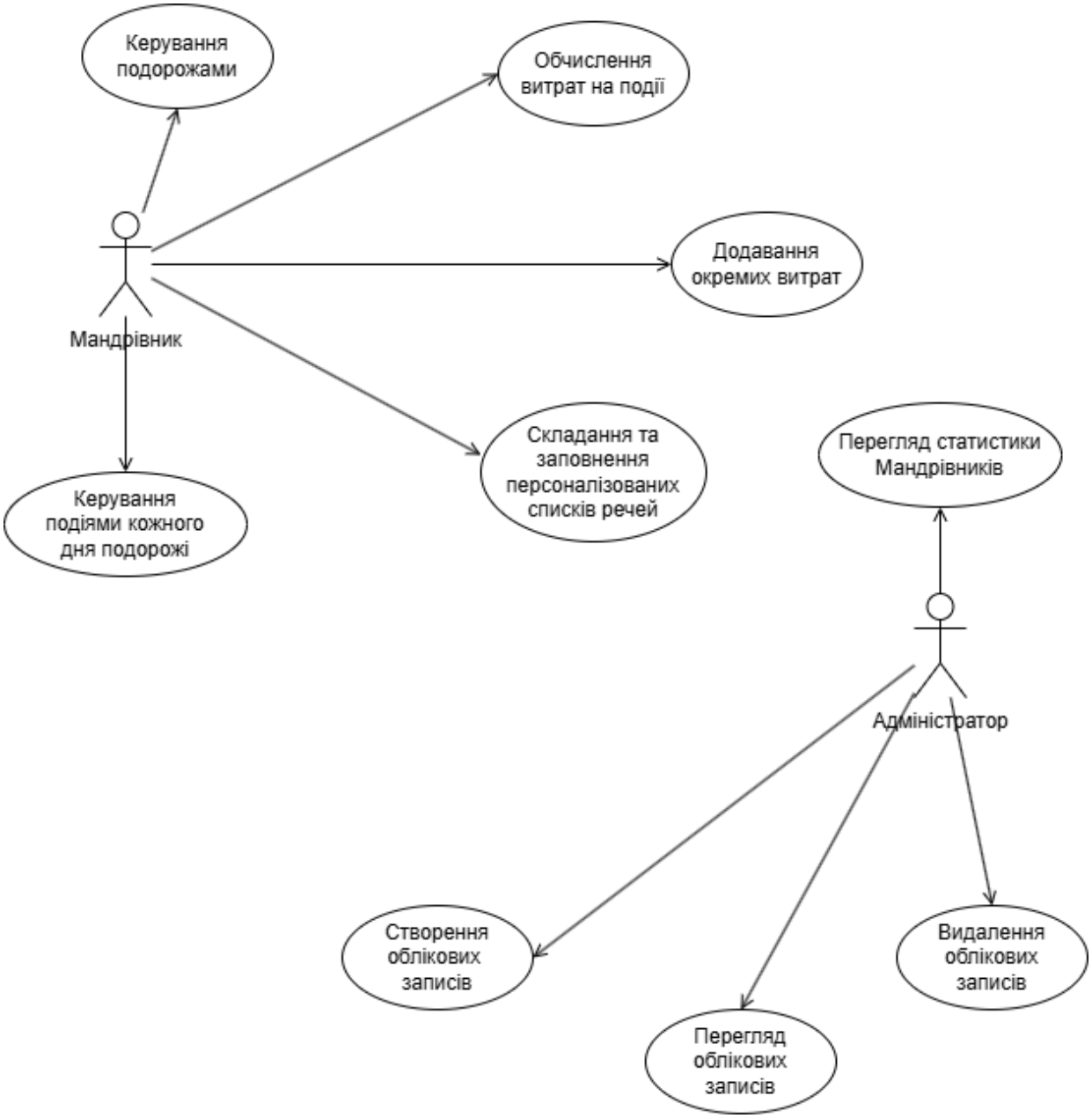


Рис. 3.1. Діаграма варіантів використання

3.2. Опис структури системи

3.2.1. Модуль мандрівника

Модуль мандрівника є основним функціональним компонентом системи, оскільки охоплює більшість сценаріїв використання. Його структура містить наступні підсистеми:

1. Система автентифікації та авторизації. Реалізує механізми реєстрації нового користувача, входу до системи та валідації облікових даних. Забезпечує захищений доступ до персоналізованого контенту.
2. Панель керування подорожами. Центральний компонент, що забезпечує створення, редагування, перегляд та видалення подорожей. Дані зберігаються у базі даних із підтримкою історичної інформації.
3. Модуль управління подіями. Відповідає за створення хронологій подорожей. Включає календарну прив’язку подій, визначення їх типів, тривалості та супровідних даних.
4. Підсистема обліку витрат. Реалізована у вигляді аналітичного модуля з інтерактивною візуалізацією витрат користувача (графіки, категорії, динаміка витрат). Підтримується додавання витрат вручну або через прив’язки до подій.
5. Файлове сховище. Призначене для збереження та керування файлами, що супроводжують подорож. Забезпечує операції перегляду, завантаження, перейменування та видалення.
6. Модуль списків речей. Забезпечує створення та ведення списків необхідних речей. Підтримуються шаблони, індивідуальні записи, відмітка зібраного, оновлення прогресу та збереження користувацьких налаштувань.

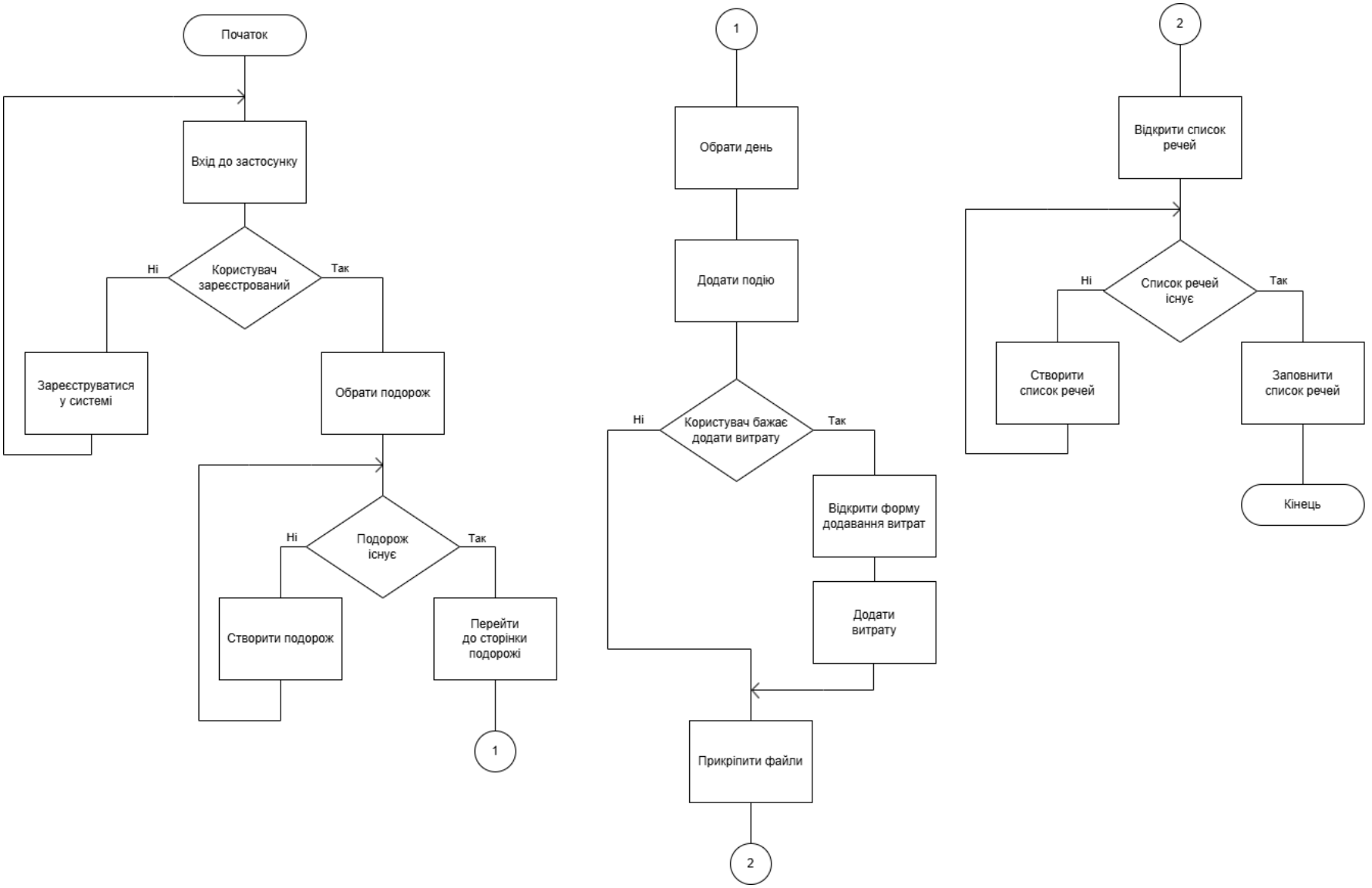


Рис. 3.2. Блок-схема алгоритму роботи застосунку

3.2.2. Модуль адміністратора

Модуль адміністратора має обмежену, переважно наглядову функціональність. Вона містить: перегляд зареєстрованих користувачів, відстежування їхньої активності, статистики використання системи. Також адміністратор може вільно видаляти користувачів із системи та створювати нових.

Отже, архітектура розроблюваного застосунку побудована за модульним принципом, що забезпечує гнучкість у реалізації, розширюваність функціональності та зручність у подальшому супроводі системи.

3.3. Проєктування моделі даних

Оскільки на етапі вибору засобів розробки було обрано реляційну СКБД, сутності системи представлені у вигляді таблиць, а їхні властивості – у вигляді атрибутів із зазначенням типів даних та обмежень. Структура бази даних нормалізована до третьої нормальної форми. Зв’язки між таблицями реалізовано за допомогою первинних і зовнішніх ключів.

1. Users (користувачі):
   - id (uuid) – унікальний ідентифікатор користувача;
   - email (citext) – електронна адреса, нечутлива до регістру, унікальна;
   - password_hash (text) – хеш пароля;
   - full_name (text) – повне ім’я, необов’язкове;
   - created_at (timestamptz) – дата та час створення запису;
   - updated_at (timestamptz) – дата та час останнього оновлення.
2. Trips (Подорожі):
   - id (uuid) – унікальний ідентифікатор подорожі;
   - user_id (uuid) – зовнішній ключ на користувача-власника;
   - name (text) – назва подорожі;
   - destination (text) – пункт призначення;

- start_date (date) – дата початку;
- end_date (date) – дата завершення;
- description (text) – додатковий опис, необов’язково;
- created_at (timestamptz) – дата та час створення подорожі;
- updated_at (timestamptz) – дата та час останнього оновлення подорожі.

3. Events (події):
   - id (uuid) – унікальний ідентифікатор події;
   - trip_id (uuid) – зовнішній ключ на подорож;
   - category (event_category) – категорія (hotel, flight …);
   - name (text) – заголовок події;
   - address (text) – адреса, необов’язково;
   - phone (text) – телефон, необов’язково;
   - start_ts (timestamptz) – час початку події;
   - end_ts (timestamptz) – час завершення, необов’язково;
   - confirmation (text) – № бронювання або PNR, необов’язково;
   - price (numeric(12,2)) – вартість, необов’язково;
   - currency (char(3)) – код валюти (USD за замовчуванням);
   - details (jsonb) – довільні деталі;
   - created_at (timestamptz) – дата та час створення події;
   - updated_at (timestamptz) – дата та час редагування події.
4. Files (файли):
   - id (uuid) – унікальний ідентифікатор файлу;
   - trip_id (uuid) – FK на подорож;
   - storage_key (text) – шлях у сховищі або S3-ключ;
   - filename (text) – оригінальна назва файлу;
   - mime_type (text) – MIME-тип, необов’язково;
   - byte_size (int) – розмір у байтах, необов’язково;
   - uploaded_at (timestamptz) – коли завантажено.

5. Event files (файли подій):
   - event_id (uuid) – зовнішній ключ на подію (частина первинного);
   - file_id (uuid) – зовнішній ключ на файл (частина первинного).
6. Expenses (витрати):
   - id (uuid) – унікальний ідентифікатор витрати;
   - trip_id (uuid) – зовнішній ключ на подорож;
   - event_id (uuid) – зовнішній ключ на подію, необов'язково;
   - category (expense_category) – категорія витрати;
   - name (text) – опис («Dinner at …»);
   - expense_date (date) – дата витрати;
   - amount (numeric(12,2)) – сума (невід'ємна);
   - currency (char(3)) – ISO-код валюти (USD за замовчуванням);
   - created_at (timestamptz) – коли створена витрата.
7. Checklist templates (шаблони списків речей):
   - id (uuid) – унікальний ідентифікатор шаблону;
   - name (text) – назва шаблону;
   - description (text) – опис, необов'язково.
8. Checklist template items (елементи шаблонів):
   - id (uuid) – унікальний ідентифікатор елемента;
   - template_id (uuid) – зовнішній ключ на шаблон;
   - item_name (text) – назва речі;
   - default_quantity (int) – кількість за замовчуванням (невід'ємна).
9. Checklists (списки речей):
   - id (uuid) – унікальний ідентифікатор списку;
   - trip_id (uuid) – зовнішній ключ на подорож;
   - created_at (timestamptz) – коли створено.
10. Checklist items (елементи списків речей):
    - id (uuid) – унікальний ідентифікатор елемента;
    - checklist_id (uuid) – зовнішній ключ на список;

- item_name (text) – назва речі;
- quantity (int) – кількість (невід’ємна);
- collected (boolean) – уже зібрано (false за замовчуванням);
- template_item_id (uuid) – зовнішній ключ на елемент шаблону.

Таблиця 3.4

Таблиця зв’язків між сутностями в моделі даних

Зв’язок	Кардинальність	Дія при видаленні
users – trips	1 : N	Каскадне
trips – events	1 : N	Каскадне
trips – files	1 : N	Каскадне
trips – expenses	1 : N	Каскадне
events – expenses	1 : 0–N	Встановити NULL
events – event_files – files	M : N	Каскадне
checklist_templates – checklist_template_items	1 : N	Каскадне
trips – checklists	1 : 1	Каскадне
checklists – checklist_items	1 : N	Каскадне
checklist_template_items – checklist_items	1 : 0–N	Встановити NULL

Одним із ефективних способів візуалізації моделі даних є ER-діаграма (діаграма «сутність–зв’язок»). Вона відображає сутності системи та взаємозв’язки між ними у графічному вигляді, що сприяє кращому розумінню структури бази даних.

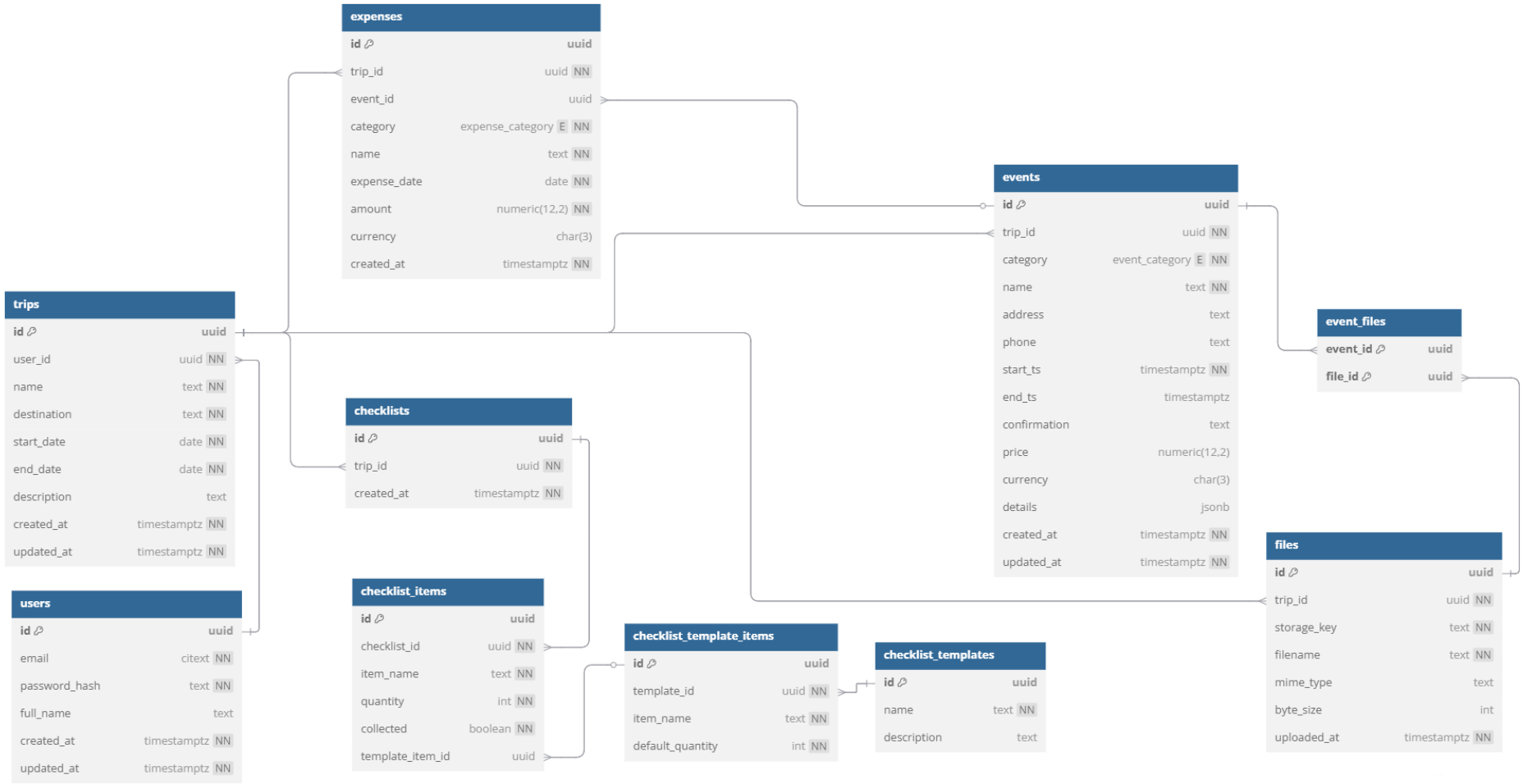


Рис. 3.8. ER-діаграма моделі даних

3.4. Архітектура застосунку

Застосунок побудований на основі багатошарової архітектури з дотриманням принципу розподілу відповідальностей, відповідно до якого кожен модуль виконує лише одну чітко визначену функцію. Для взаємодії між модулями використовуються відкриті інтерфейси, також відомі як API, що забезпечує слабке зв'язування між компонентами та сприяє гнучкості системи.

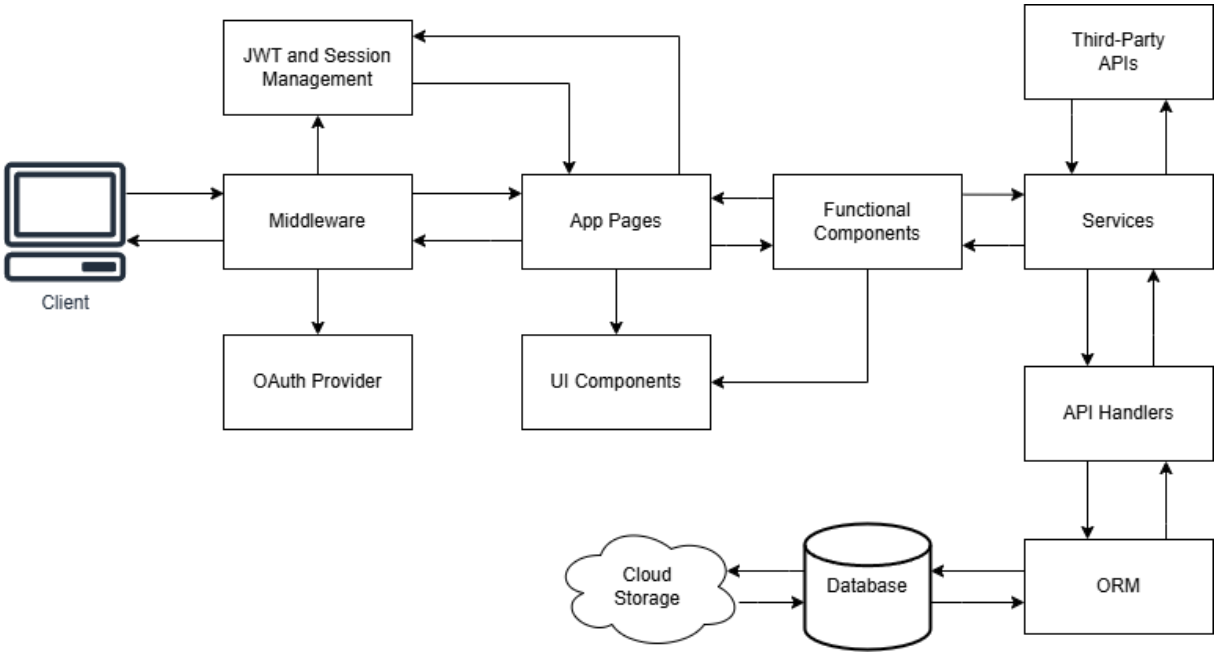


Рис. 3.9. Діаграма архітектури застосунку

Робота застосунку розпочинається зі звернення користувача до інтерфейсу – як через взаємодію з елементами UI, так і шляхом переходу за визначеною URL-адресою. Кожен запит, який надходить від клієнтської частини, спочатку обробляється проміжним програмним забезпеченням, яке виконує перевірку автентичності та авторизації користувача, а також підтримку сесії.

На цьому етапі проміжне програмне забезпечення перевіряє наявність дійсного JWT-токена або ідентифікатора провайдера автентифікації у разі використання протоколу OAuth. Якщо дані є коректними, запит

пропускається до відповідного маршруту, інакше – блокується, а користувач отримує повідомлення про відмову в доступі.

У разі успішної перевірки користувач перенаправляється до відповідного представлення інтерфейсу застосунку – динамічного вікна, яке оновлюється у відповідь на дії користувача та наповнюється актуальними даними з бази даних. Це вікно формується з компонентів – окремих функціональних елементів, кожен з яких відповідає за певну частину інтерфейсу.

Компоненти поділяються на два основні типи:

- UI-компоненти – базові елементи інтерфейсу (кнопки, випадаючі списки, текстові поля тощо).
- Функціональні компоненти – складніші елементи, які складаються з кількох UI-компонентів і реалізують більш комплексну логіку (наприклад, відображення календаря або формування звіту про витрати).

Функціональні компоненти отримують необхідні дані через сервіси – програмні модулі, що містять в собі бізнес-логіку застосунку. Вони виконують основні операції, такі як обробка даних, валідація, трансформація форматів і виконання CRUD-операцій. Для отримання необхідної інформації сервіси звертаються до API-обробників – модулів, що формують HTTP-запити до серверної частини.

API-обробники взаємодіють із ORM-рівнем – програмним модулем, який абстрагує взаємодію з системою керування базами даних, забезпечуючи отримання, зміну або видалення даних. Відповідь від ORM передається назад до сервісу, де дані можуть бути додатково оброблені та передані функціональним компонентам, змінюючи їхній стан.

У разі потреби сервіси також можуть взаємодіяти з зовнішніми API, наприклад, для отримання географічних координат, прогнозу погоди чи іншої інформації від сторонніх постачальників.

Крім того, частина функціональних компонентів може використовувати модулі автентифікації для виконання певних дій, таких як вихід із системи або оновлення токена доступу.

Після завершення усіх етапів обробки даних відбувається формування кінцевої сторінки, яка надсилається у відповідь клієнту у вигляді HTML-, CSS- та JavaScript-файлів. Веббраузер користувача обробляє ці файли та відображає інтерфейс, відповідно до поточного стану застосунку.

3.5. Висновки до третього розділу

У даному розділі було виконано ряд задач, а саме:

1) було здійснено детальне визначення та формалізацію вимог;
2) було надано загальний опис системи та її функціональності;
3) було спроєктовано модель даних та побудовано відповідну діаграму;
4) було визначено та побудовано архітектуру застосунку.

4. ОСОБЛИВОСТІ ТЕХНІЧНОЇ РЕАЛІЗАЦІЇ

4.1. Загальний опис реалізації

Технічна реалізація застосунку представляє собою сучасний вебзастосунок, орієнтований на мобільні пристрої, побудований на потужному стеку JavaScript-технологій з використанням TypeScript для забезпечення типобезпеки. Основою проєкту виступає фреймворк Next.js, який надає оптимальні можливості як для відмальовування на серверній стороні, так і для створення кінцевих точок API. Інтерфейсна частина реалізована на React 19, найновішій версії бібліотеки, що забезпечує оптимальну продуктивність та сучасні можливості для розробки компонентів.

Архітектура проєкту дотримується принципів RESTful API, з ендпоінтами, організованими в директорії /api. Використання React Server Components дозволяє оптимізувати продуктивність за рахунок виконання частини логіки на сервері. Структура бази даних включає моделі для користувачів, подорожей, подій, файлів, витрат та списків речей, з чітко визначеними зв'язками між ними. Така архітектура забезпечує гнучкість та масштабованість системи, дозволяючи легко розширювати функціональність у майбутньому.

4.1.1. Опис особливостей реалізації серверної частини

Для забезпечення зручної та надійної роботи з даними використовується PostgreSQL у поєднанні з ORM Prisma. Це дозволяє моделювати складні зв'язки між сутностями системи – користувачами, подорожами, подіями, витратами та чеклістами. Prisma надає типобезпечний доступ до бази даних і спрощує виконання запитів, міграцій та управління схемою. Особливістю проєкту є інтеграція з Neon Database – безсерверним PostgreSQL-сервісом, оптимізованим для застосунків з варіативним навантаженням.

Аутентифікація у системі реалізована за допомогою NextAuth.js, що забезпечує як зовнішню авторизацію через Google OAuth, так і класичну реєстрацію з логіном та паролем [32]. Для роботи з паролями використовується bcrypt, що забезпечує безпечне хешування. JSON Web Tokens (JWT) застосовуються для підтримки сесій користувачів, а бібліотеки jsonwebtoken та jose надають необхідні інструменти для роботи з токенами. Валідація вхідних даних при реєстрації та логіні здійснюється за допомогою бібліотеки Zod, що встановлює суворі правила для паролів, електронної пошти та інших користувацьких даних [33].

Зберігання файлів, пов'язаних з подорожами, реалізовано через інтеграцію з Google Cloud Storage, що забезпечує надійне та масштабоване рішення для зберігання зображень, документів та інших файлів. У системі ведеться облік метаданих файлів, таких як ім'я, MIME-тип та розмір, що дозволяє ефективно управляти контентом.

Для роботи з географічними даними використовується інтеграція з Google Maps через React Google Maps, що дозволяє користувачам обирати місця призначення, та використовувати автозаповнення адрес.

Для підвищення інформативності та зручності планування подорожей у систему інтегровано Open Meteo API – безкоштовний сервіс прогнозу погоди з підтримкою геолокації. Це дозволяє користувачам отримувати актуальний прогноз на вибрані дати подорожі безпосередньо у застосунку. Завдяки цьому рішенням користувачі можуть краще підготуватися до умов перебування в певній місцевості, обираючи одяг, спорядження чи плануючи активності відповідно до погодних умов.

4.1.2. Опис особливостей реалізації клієнтської частини

Візуальний інтерфейс застосунку розроблений з використанням компонентної бібліотеки shadcn/ui, яка базується на примітивах Radix UI. Цей підхід забезпечує доступність, кастомізованість та послідовність дизайну. Серед використовуваних компонентів – діалогові вікна, форми,

вкладки, тости для сповіщень та картки. Стилізація компонентів здійснюється за допомогою Tailwind CSS.

Для роботи з формами застосовується React Hook Form, що забезпечує ефективну обробку введених даних, валідацію та управління станом форм. Інтеграція з Zod дозволяє створювати типобезпечні схеми валідації для форм. Особливу увагу приділено валідації даних користувача – реалізовано ретельну перевірку електронної пошти з використанням регулярних виразів, перевірку складності паролів (включаючи наявність великих та малих літер, цифр, спеціальних символів), а також валідацію телефонних номерів з підтримкою різних міжнародних форматів.

Робота з датами та часом реалізована за допомогою бібліотек date-fns (маніпуляції з датами), react-day-picker (компонент для вибору дат). Для відображення аналітичних даних, таких як статистика подорожей та витрат, використовується бібліотека recharts, а для правильної роботи з грошовими значеннями – currency.js.

4.2. Тестування застосунку

У межах розробки було здійснено ручне тестування відповідно до принципів димного тестування – методики, яка передбачає виконання мінімального набору тестів, спрямованих на виявлення критичних помилок у базовій функціональності системи. Зазвичай димне тестування виконується безпосередньо розробником перед передачею програмного забезпечення на етап повноцінного тестування, оскільки версія, яка не проходить базових перевірок, не підлягає подальшому тестуванню.

У рамках тестування було перевірено наступні випадки:

1) реєстрація користувача;

2) вхід до системи;

3) створення подорожі;

4) створення події;

5) додавання власної витрати;

6) прикріплення файлу;

7) створення та заповнення списку речей.

4.3. Рекомендації щодо подальшого вдосконалення

Даний застосунок відповідає усім поставленим до нього вимогам, однак має потенціал для подальшого розширення функціональності та покращення користувацького досвіду.

У першу чергу доцільно реалізувати можливість взаємодії з іншими користувачами. Запровадження функцій обміну планами подорожей, перегляду маршрутів інших користувачів і спільного редагування маршрутів значно підвищить зручність використання системи, особливо для групових поїздок.

Наступним кроком до вдосконалення є впровадження локалізації інтерфейсу. Оскільки застосунок може використовуватися людьми з різних країн, важливо передбачити підтримку декількох мов, валют і форматів дат. Це дозволить зробити систему більш гнучкою та зручною для широкого кола користувачів.

Окрім цього, варто розглянути можливість інтеграції з зовнішніми API, зокрема API для бронювання готелів або авіаквитків. Така інтеграція дозволить користувачам здійснювати замовлення безпосередньо з інтерфейсу застосунку, що підвищить його практичну цінність та привабливість для мандрівників.

4.4. Висновки до четвертого розділу

У даному розділі було виконано ряд задач, а саме:

1) описано загальні деталі технічної реалізації застосунку;
2) описано особливості реалізації серверної та клієнтської частини;
3) описано тестування застосунку;
4) наведено ряд рекомендацій щодо розширення функціональності застосунку.

ВИСНОВКИ

Метою даного дипломного проєкту є полегшення підготовки до подорожі шляхом надання мобільного застосунку, який надаватиме всебічну підтримку користувачеві на всіх етапах підготовки.

Перед початком розроблення було досліджено предметну область, проаналізовано існуючі аналоги, представлено ідею вирішення поставленої задачі. Аналіз різних типів програмного забезпечення та засобів їх розроблення, попередньо виконаний в дипломному проєкті, показав доцільність створення застосунку з використанням мови програмування JavaScript та фреймворку Next.js на стороні сервера, JavaScript-фреймворку React на стороні клієнта. В якості системи керування базами даних було обрано PostgreSQL.

Розроблений застосунок:

1. Забезпечує розмежування доступу до функціональності, реалізуючи процедури реєстрації та авторизації.
2. Надає функціональність для складання розкладу подорожі, стеженням за витратами, складанням списків речей та зберігання електронних документів.
3. Взаємодіє з сторонніми API для від Google та Open Meteo для автозаповнення при пошуку пунктів призначення та надання прогнозу погоди.

Розробка виконана у повному обсязі згідно з положенням Технічного завдання, тестування продукту проведено відповідно до затвердженої програми та методики тестування. Під час тестування особливу увагу було приділено перевірці критично важливих функцій системи. Представлені напрямки подальшого вдосконалення застосунку.

Розроблений застосунок забезпечує централізоване та зручне управління усіма аспектами планування подорожі – від складання розкладу до контролю витрат і підготовки речей.

СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. Рушії розвитку туристичної індустрії в умовах сучасності [Електронний ресурс], – Режим доступу: https://www.researchgate.net/publication/371384606_RUSII_ROZVITKU_TURISTICNOI_INDUSTRII_V_UMOVAH_SUCASNOSTI. – Дата доступу: 26.02.2025 р.
2. PackPoint travel packing list app. [Електронний ресурс], – Режим доступу: https://www.packpnt.com/. – Дата доступу: 01.03.2025 р.
3. How it works. [Електронний ресурс] – Режим доступу: https://www.tripit.com/web/free/how-it-works. – Дата доступу: 01.03.2025 р.
4. Notion goes where Evernote dares not: what you need to know. [Електронний ресурс], – Режим доступу: https://androidcommunity.com/notion-goes-where-evernote-dares-not-what-you-need-to-know-20180608/. – Дата доступу: 03.03.2025 р.
5. Java | Definition & Facts | Britannica. [Електронний ресурс], – Режим доступу: https://www.britannica.com/technology/Java-computer-programming-language. – Дата доступу: 15.03.2025 р.
6. How JVM Works - JVM Architecture – GeeksforGeeks, [Електронний ресурс], – Режим доступу: https://www.geeksforgeeks.org/how-jvm-works-jvm-architecture/. – Дата доступу: 18.03.2025 р.
7. Where is Java used and why is it needed? [Електронний ресурс], – Режим доступу: https://javarush.com/en/groups/posts/en.1079.where-is-java-used-and-why-is-it-needed. – Дата доступу: 20.03.2025 р.
8. History of the Python development language. [Електронний ресурс], – Режим доступу: https://www.bocasay.com/history-python-programming/. – Дата доступу: 23.03.2025 р.
9. What is Python Interpreter – GeeksforGeeks. [Електронний ресурс], – Режим доступу: https://www.geeksforgeeks.org/what-is-python-interpreter/. – Дата доступу: 25.03.2025 р.

10. Where is Python used in the real world? [Електронний ресурс], – Режим доступу: https://academy.nit-institute.com/where-is-python-used-real-world/, – Дата доступу: 30.03.2025 р.
11. What Is the Python Global Interpreter Lock (GIL)? – Real Python. [Електронний ресурс], – Режим доступу: https://realpython.com/python-gil/, – Дата доступу: 02.04.2025 р
12. History of JavaScript - read our article to find out! [Електронний ресурс], – Режим доступу: https://softteco.com/blog/history-of-javascript, – Дата доступу: 05.04.2025 р.
13. JavaScript frameworks and libraries - Learn web development | MDN. [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/Frameworks_libraries, – Дата доступу: 07.04.2025 р.
14. ECMA-262 - Ecma International. [Електронний ресурс], – Режим доступу: https://ecma-international.org/publications-and-standards/standards/ecma-262/, – Дата доступу: 09.04.2025 р.
15. Vue.js history. [Електронний ресурс], – Режим доступу: https://madushaprasad21.medium.com/vue-js-history-1a6b8567198f. – Дата доступу: 12.04.2025 р.
16. Vue.js - The Progressive JavaScript Framework | Vue.js. [Електронний ресурс], – Режим доступу: https://vuejs.org/glossary/. – Дата доступу: 15.04.2025 р.
17. What is Vue.js? Overview of the Versatile JavaScript Framework – Glossary. [Електронний ресурс], – Режим доступу: https://www.sanity.io/glossary/vue-js. – Дата доступу: 15.04.2025 р.
18. What is Angular? [Електронний ресурс], – Режим доступу: https://angular.dev/overview. – Дата доступу: 17.04.2025 р.
19. Top 20 Key Features Of Angular. [Електронний ресурс], – Режим доступу: https://medium.com/@aqeelabbas3972/modular-development-angular-uses-a-modular-approach-to-development-

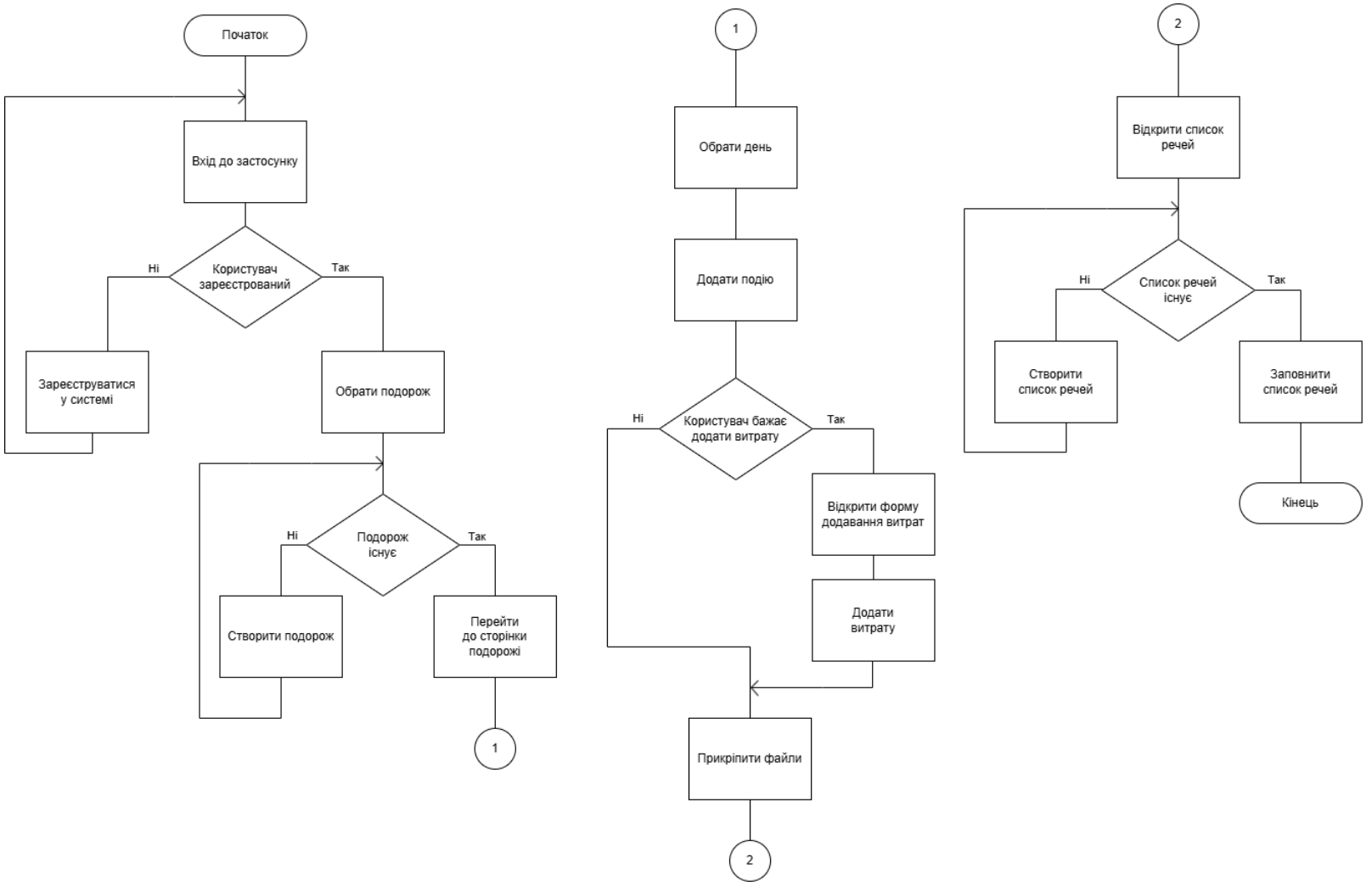
which-helps-developers-create-7e9686a51eb4. – Дата доступу: 18.04.2025 р.
20. Ten Best Angular UI Libraries. [Електронний ресурс], – Режим доступу: https://dreamix.eu/insights/10-best-angular-ui-libraries/. – Дата доступу: 19.04.2025 р.
21. React Reference – React. [Електронний ресурс], – Режим доступу: https://react.dev/reference/react. – Дата доступу: 21.04.2025 р.
22. Build a React app from Scratch – React. [Електронний ресурс], – Режим доступу: https://react.dev/learn/build-a-react-app-from-scratch. – Дата доступу: 22.04.2025 р.
23. Best 19 React UI Component Libraries in 2025. [Електронний ресурс], – Режим доступу: https://prismic.io/blog/react-component-libraries. – Дата доступу: 22.04.2025 р.
24. Best React Frameworks | Next.js, Gatsby, CRA, Remix, Blitz.js. [Електронний ресурс], – Режим доступу: https://www.webdevstory.com/best-react-frameworks/. – Дата доступу: 23.04.2025 р.
25. MongoDB: Flexible database for the agile age. [Електронний ресурс], – Режим доступу: https://www.oracle.com/ua/database/mongodb/. – Дата доступу: 23.04.2025 р.
26. MySQL Explained: Your Guide to Mastering This Powerful Database. [Електронний ресурс], – Режим доступу: https://www.oracle.com/mysql/what-is-mysql/. – Дата доступу: 24.04.2025 р.
27. Understanding Indexes in MySQL. [Електронний ресурс], – Режим доступу: https://shlokashah.medium.com/understanding-indexes-in-mysql-ed4535998863. – Дата доступу: 24.04.2025 р.
28. Optimizing MySQL for Peak Performance – A Comprehensive Guide. [Електронний ресурс], – Режим доступу: https://medium.com/relee m/optimizing-mysql-for-peak-performance-a-comprehensive-guide-881ba1ad3ee1. – Дата доступу: 24.04.2025 р.

29. PostgreSQL Documentation. [Електронний ресурс], – Режим доступу: https://www.postgresql.org/docs/current/. – Дата доступу: 25.04.2025 р.
30. A Developers Guide to PostgreSQL Extensions. [Електронний ресурс], – Режим доступу: https://dev.to/timescale/postgresql-extensions-what-they-are-and-how-to-use-them-4i76. – Дата доступу: 25.04.2025 р.
31. Advanced Indexing Strategies in PostgreSQL. [Електронний ресурс], – Режим доступу: https://www.freecodecamp.org/news/postgresql-indexing-strategies/. – Дата доступу: 25.04.2025 р.
32. Introduction | NextAuth.js. [Електронний ресурс], – Режим доступу: https://next-auth.js.org/getting-started/introduction. – Дата доступу: 26.04.2025 р.
33. Intro | Zod. [Електронний ресурс], – Режим доступу: https://zod.dev./ – Дата доступу: 26.04.2025 р.

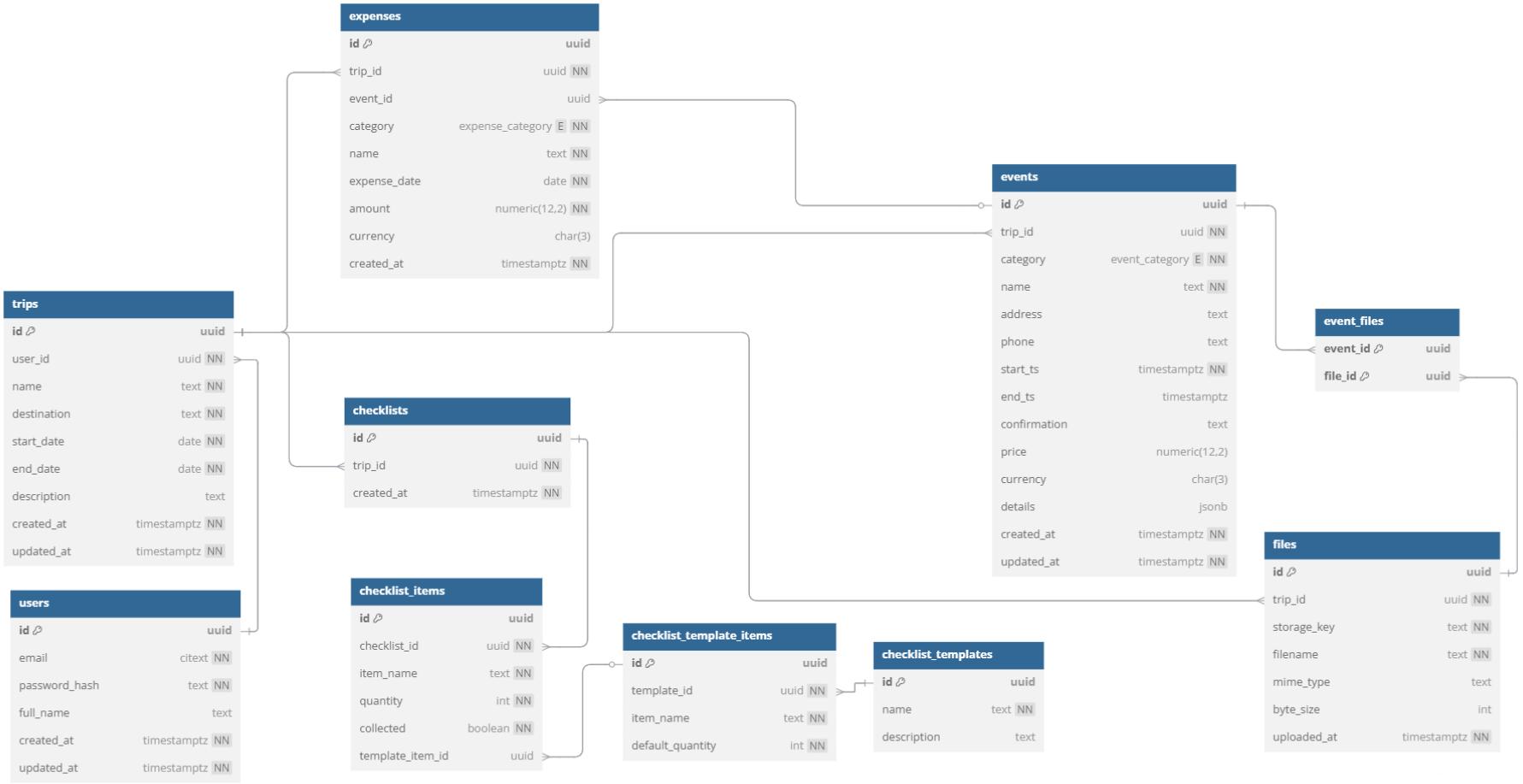
ДОДАТКИ

Додаток 1

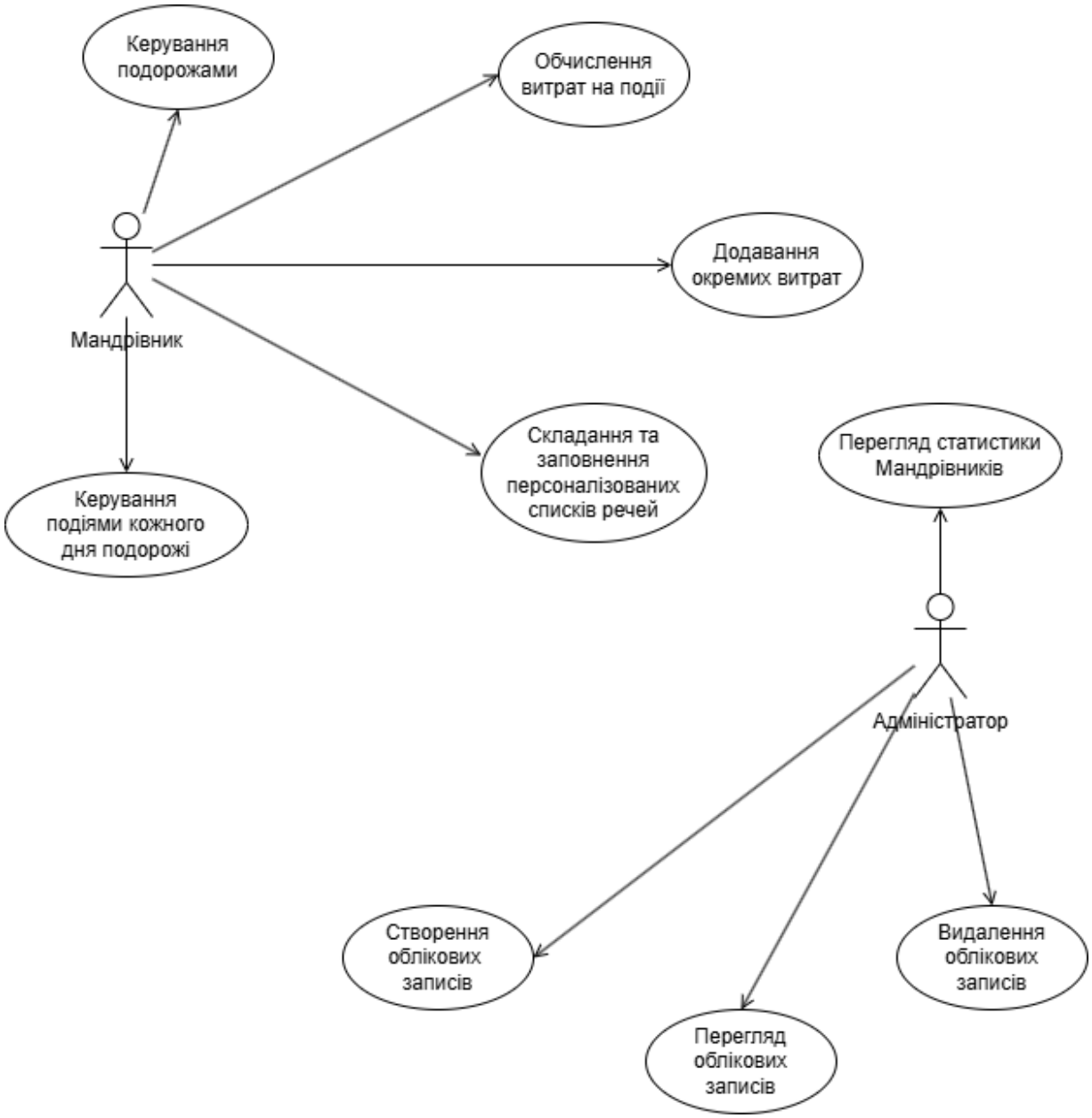
Копії графічних матеріалів



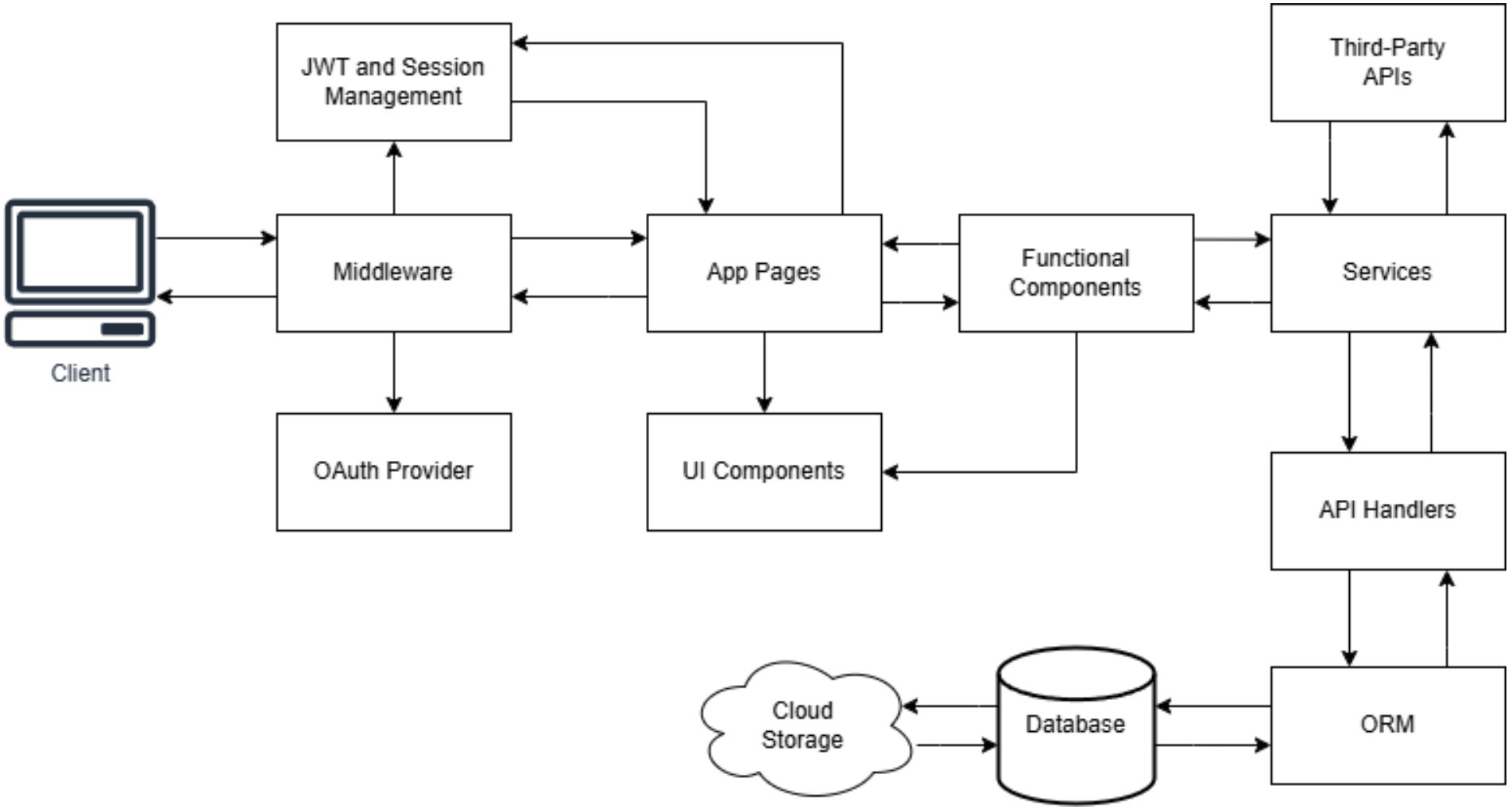
ДП.045440-06-99. Мобільний застосунок для організації пасажирських перевезень. Алгоритм роботи застосунку. Блок-схема алгоритму



ДП.045440-07-99. Мобільний застосунок для організації пасажирських перевезень. Структура бази даних. ER-діаграма



Діаграма варіантів використання.
Маковецький П.В., група КП-11



Архітектура застосунку.
Маковецький П.В., група КП-11

Додаток 2

Лістинг програми

Фрагменти коду вікна подорожі (./app/trips/[id]/page.tsx)

```
"use client";

import { TripFiles } from "@/components/TripFiles";
import { Button } from "@/components/ui/button";
import {
  Card,
  CardContent,
  CardDescription,
  CardHeader,
  CardTitle,
} from "@/components/ui/card";
import { ChecklistPanel } from "@/components/ui/checklist-panel";
import { EventDialog } from "@/components/ui/event-dialog";
import { Tabs, TabsContent, TabsList, TabsTrigger } from
"@/components/ui/tabs";
import { Toaster } from "@/components/ui/toaster";
import { EventCategory } from "@prisma/client";
import currency from "currency.js";
import {
  CalendarDaysIcon,
  CalendarIcon,
  CheckSquareIcon,
  ChevronDown,
  ChevronLeft,
  DollarSignIcon,
  FolderIcon,
  MapPinIcon,
} from "lucide-react";
import Link from "next/link";
import { useRouter } from "next/navigation";
import React, { useEffect, useState } from "react";

import {
  BudgetItem,
  ChecklistItem,
  DayWeather,
  Event,
  Trip,
  TripParams,
} from "@/lib/types/trip";

import { formatDate, getDateRange } from "@/lib/utils/date-utils";
import { mapEventToFormData } from "@/lib/utils/event-utils";

import { budgetService, weatherService } from "@/lib/services";

import { BudgetPanel } from "@/components/trip/BudgetPanel";
import { DayView } from "@/components/trip/DayView";
import { Timeline } from "@/components/trip/Timeline";

export default function TripDetailPage({
  params,
}: {
  params: Promise<TripParams>;
}) {
  const unwrappedParams = React.use(params);
  const tripId = unwrappedParams.id;

  const router = useRouter();
  const [trip, setTrip] = useState<Trip | null>(null);
```

```
        const [events, setEvents] = useState<Event[]>([]);
        const [weather, setWeather] = useState<DayWeather | null>(null);
        const [budgetItems, setBudgetItems] = useState<BudgetItem[]>([]);
        const [checklist, setChecklist] = useState<ChecklistItem[]>([]);
        const [isLoading, setIsLoading] = useState(true);
        const [error, setError] = useState<string | null>(null);
        const [selectedDate, setSelectedDate] = useState<string |
null>(null);
        const [showEventDialog, setShowEventDialog] = useState(false);
        const [selectedEvent, setSelectedEvent] = useState<Event |
null>(null);
        const [selectedEventType, setSelectedEventType] =
          useState<EventCategory>("hotel");
        const [isEditing, setIsEditing] = useState(false);
        const [activeTab, setActiveTab] = useState<string>("timeline");
        const [showDescription, setShowDescription] = useState(false);

        useEffect(() => {
          const fetchTripData = async () => {
            setIsLoading(true);
            try {
              const tripResponse = await fetch(`/api/trips/${tripId}`);
              if (!tripResponse.ok) {
                if (tripResponse.status === 404) {
                  router.push("/trips");
                  return;
                }
                throw new Error("Failed to fetch trip");
              }
              const tripData = await tripResponse.json();
              setTrip(tripData.trip);

              await fetchEvents();

              if (tripData.trip) {
                generateMockDataExceptEventsAndChecklist(tripData.trip);
              }
            } catch (err) {
              console.error("Error fetching trip:", err);
              setError("Failed to load trip details. Please try again
later.");
            } finally {
              setIsLoading(false);
            }
          };

          fetchTripData();
        }, [tripId, router]);

        const fetchEvents = async () => {
          try {
            const response = await fetch(`/api/trips/${tripId}/events`, {
              cache: "no-store",
            });

            if (!response.ok) {
              throw new Error(
                `Failed to fetch events: ${response.status}
${response.statusText}`
              );
            }

            const data = await response.json();
```

```
            if (Array.isArray(data.events)) {
              setEvents(data.events);
              const budgetFromEvents =
budgetService.generateBudgetFromEvents(
                data.events
              );
              setBudgetItems(budgetFromEvents);
            } else {
              console.error("API returned non-array events:", data);
              setEvents([]);
              setBudgetItems([]);
            }

            return data.events || [];
          } catch (error) {
            console.error("Error fetching events:", error);
            setEvents([]);
            setBudgetItems([]);
            return [];
          }
        };

        const generateMockDataExceptEventsAndChecklist = (trip: Trip) => {
          if (!trip) return;

          const startDate = new Date(trip.startDate);
          const endDate = new Date(trip.endDate);
          const dayDiff =
            Math.ceil(
              (endDate.getTime() - startDate.getTime()) / (1000 * 60 * 60 *
24)
            ) + 1;

          setSelectedDate(startDate.toISOString().split("T")[0]);
        };

        useEffect(() => {
          if (selectedDate) {
            setWeather(weatherService.generateMockWeather(selectedDate));
          }
        }, [selectedDate]);

        const getEventsForDay = (date: string) => {
          if (!events || events.length === 0) {
            return [];
          }

          const formattedSelectedDate = new
Date(date).toISOString().split("T")[0];

          const filteredEvents = events.filter((event) => {
            try {
              if (!event.startTs) {
                console.warn("Event missing startTs:", event);
                return false;
              }

              const eventStartDateObj = new Date(event.startTs);
              const eventStartDate =
eventStartDateObj.toISOString().split("T")[0];

              if (eventStartDate === formattedSelectedDate) {
                return true;
              }
```

```
              if (event.endTs) {
                const eventEndDateObj = new Date(event.endTs);
                const eventEndDate =
eventEndDateObj.toISOString().split("T")[0];

                if (
                  eventStartDate !== eventEndDate &&
                  formattedSelectedDate === eventEndDate
                ) {
                  return true;
                }
              }

              return false;
            } catch (error) {
              console.error("Error parsing event date:", error, event);
              return false;
            }
          });

          return filteredEvents.sort(
            (a, b) => new Date(a.startTs).getTime() - new
Date(b.startTs).getTime()
          );
        };

        const getChecklistProgress = () => {
          const collected = checklist.filter((item) =>
item.collected).length;
          return { collected, total: checklist.length };
        };

        const handleEditEvent = (event: Event) => {
          setSelectedEvent(event);
          setSelectedEventType(event.category as EventCategory);
          setIsEditing(true);
          setShowEventDialog(true);
        };

        const handleDeleteEvent = async (eventId: string) => {
          if (!confirm("Are you sure you want to delete this event?"))
return;

          try {
            const response = await
fetch(`/api/trips/${tripId}/events/${eventId}`, {
              method: "DELETE",
            });

            if (!response.ok) throw new Error("Failed to delete event");

            fetchEvents();
          } catch (error) {
            console.error("Error deleting event:", error);
            alert("Failed to delete event. Please try again.");
          }
        };

        const handleAddEvent = (eventType: EventCategory) => {
          setSelectedEventType(eventType);
          setIsEditing(false);
          setSelectedEvent(null);
          setShowEventDialog(true);
```

```
        };

        const handleEventSubmit = async (data: any) => {
          try {
            const currentSelectedDate =
              selectedDate || new Date().toISOString().split("T")[0];

            let details: any = {};

            if (selectedEventType === "hotel") {
              const checkInDate =
                isEditing && data.checkInDate
                  ? data.checkInDate.toISOString().split("T")[0]
                  : currentSelectedDate;

              let checkOutDate;
              if (isEditing && data.checkOutDate) {
                checkOutDate =
data.checkOutDate.toISOString().split("T")[0];
              } else {
                const nextDay = new Date(checkInDate);
                nextDay.setDate(nextDay.getDate() + 1);
                checkOutDate = nextDay.toISOString().split("T")[0];
              }

              details = {
                checkInDate,
                checkInTime: data.checkInTime || "14:00",
                checkOutDate,
                checkOutTime: data.checkOutTime || "12:00",
              };
            } else if (selectedEventType === "restaurant") {
              details = {
                reservationDate:
                  isEditing && data.reservationDate
                    ? data.reservationDate.toISOString().split("T")[0]
                    : currentSelectedDate,
                reservationTime: data.reservationTime || "19:00",
              };
            } else if (selectedEventType === "flight") {
              const departureDate =
                isEditing && data.departureDate
                  ? data.departureDate.toISOString().split("T")[0]
                  : currentSelectedDate;
              const departureTime = data.departureTime || "10:00";

              let arrivalDate;
              let arrivalTime;

              if (isEditing && data.arrivalDate) {
                arrivalDate = data.arrivalDate.toISOString().split("T")[0];
                arrivalTime = data.arrivalTime || "12:00";
              } else {
                arrivalDate = departureDate;
                const [depHours, depMinutes] =
departureTime.split(":").map(Number);
                let arrHours = depHours + 2;
                const arrMinutes = depMinutes;

                if (arrHours >= 24) {
                  arrHours -= 24;
                  const nextDay = new Date(departureDate);
                  nextDay.setDate(nextDay.getDate() + 1);
                  arrivalDate = nextDay.toISOString().split("T")[0];
```

```
                }

                arrivalTime = `${arrHours.toString().padStart(2,
"0")}:${arrMinutes
                  .toString()
                  .padStart(2, "0")}`;
              }

              details = {
                departureDate,
                departureTime,
                arrivalDate,
                arrivalTime,
                flightNumber: data.flightNumber || "",
              };
            } else {
              const startTime = data.details?.startTime || "09:00";
              let endTime = data.details?.endTime || "11:00";

              if (endTime <= startTime) {
                const [hours, minutes] = startTime.split(":").map(Number);
                const endHours = (hours + 2) % 24;
                endTime = `${endHours.toString().padStart(2,
"0")}:${minutes
                  .toString()
                  .padStart(2, "0")}`;
              }

              const { endTime: _, ...otherDetails } = data.details || {};

              details = {
                date: currentSelectedDate,
                startTime,
                endTime,
                ...otherDetails,
              };
            }

            const eventData = {
              name: data.name,
              address: data.address,
              phone: data.phone || null,
              confirmationNumber: data.confirmationNumber || null,
              price: data.price || null,
              category: selectedEventType,
              details,
            };

            if (data.locationDetails) {
              if (selectedEventType === "transport") {
                if (data.locationDetails.pickup?.location) {
                  details.pickupCoordinates =
`${data.locationDetails.pickup.location.lat},${data.locationDetails.pickup.
location.lng}`;
                }

                if (data.locationDetails.dropoff?.location) {
                  details.dropoffCoordinates =
`${data.locationDetails.dropoff.location.lat},${data.locationDetails.dropof
f.location.lng}`;
                }
              } else if (data.locationDetails.address?.location) {
```

```
              details.coordinates =
`${data.locationDetails.address.location.lat},${data.locationDetails.addres
s.location.lng}`;
            }

            eventData.details = details;
          }

          let response;

          if (isEditing && selectedEvent) {
            response = await fetch(
              `/api/trips/${tripId}/events/${selectedEvent.id}`,
              {
                method: "PUT",
                headers: { "Content-Type": "application/json" },
                body: JSON.stringify(eventData),
                cache: "no-store",
              }
            );
          } else {
            response = await fetch(`/api/trips/${tripId}/events`, {
              method: "POST",
              headers: { "Content-Type": "application/json" },
              body: JSON.stringify(eventData),
              cache: "no-store",
            });
          }

          if (!response.ok) {
            const errorData = await response.json();
            throw new Error(errorData.error || "Failed to save event");
          }

          setShowEventDialog(false);
          setSelectedEvent(null);
          setIsEditing(false);

          const currentDate = selectedDate;
          await fetchEvents();
          if (currentDate) {
            setSelectedDate(currentDate);
          }
        } catch (error) {
          console.error("Error saving event:", error);
          alert("Failed to save event. Please try again.");
        }
      };

      useEffect(() => {
        if (selectedDate) {
          if (getEventsForDay(selectedDate).length > 0) {
            setActiveTab("timeline");
          }
        }
      }, [selectedDate]);

      const goToTab = (tabName: string) => {
        setActiveTab(tabName);
      };

      const handleAddCustomExpense = (expense: BudgetItem) => {
        if (!expense) return;
```

```
          setBudgetItems((prevItems) => [...prevItems, expense]);
        };

        const handleEditCustomExpense = (id: string, updatedExpense:
BudgetItem) => {
          if (!id || !updatedExpense) return;

          setBudgetItems((prevItems) =>
            prevItems.map((item) => (item.id === id ? updatedExpense :
item))
          );
        };

        const handleDeleteCustomExpense = (id: string) => {
          if (!id) return;

          setBudgetItems((prevItems) => prevItems.filter((item) => item.id
!== id));
        };

        if (isLoading) {
          return (
            <div className="container mx-auto px-4 py-8 flex justify-center
items-center min-h-[50vh]">
              <div className="h-12 w-12 rounded-full border-4 border-
primary border-t-transparent animate-spin"></div>
            </div>
          );
        }

        if (error || !trip) {
          return (
            <div className="container mx-auto px-4 py-8">
              <Card className="bg-destructive/10 border-destructive/20">
                <CardContent className="p-6">
                  <p className="text-destructive">{error || "Trip not
found"}</p>
                  <Button asChild variant="outline" className="mt-4">
                    <Link href="/trips">Back to trips</Link>
                  </Button>
                </CardContent>
              </Card>
            </div>
          );
        }

        const dateRange = getDateRange(trip.startDate, trip.endDate);
        const checklistProgress = getChecklistProgress();

        return (
          <div className="container mx-auto px-4 py-4 sm:py-8">
            <div className="flex justify-between items-center mb-4 sm:mb-
6">
              <Button
                asChild
                variant="link"
                size="sm"
                className="gap-1 sm:gap-2 -ml-2 text-muted-foreground
hover:text-foreground"
              >
                <Link href="/trips">
                  <ChevronLeft className="h-4 w-4" />
                  <span className="hidden sm:inline">Back to trips</span>
                </Link>
```

```
              </Button>
            </div>

            <Card className="mb-4 sm:mb-6">
              <div className="flex flex-col sm:flex-row justify-between">
                <div className="flex-1">
                  <CardHeader className="py-2 sm:py-3">
                    <div className="flex justify-between items-start gap-
2">
                      <CardTitle className="text-xl sm:text-2xl pr-2">
                        {trip.name}
                      </CardTitle>
                      <div className="sm:hidden flex items-center shrink-
0">
                        <div className="text-xs flex flex-wrap justify-
end">
                          <span className="font-medium">

{currency(budgetService.getTotalBudget(budgetItems), {
                                symbol: "$",
                                precision: 2,
                              }).format()}
                          </span>
                          <span className="text-muted-foreground mx-
1">•</span>
                          <span>{events.length} events</span>
                        </div>
                      </div>
                    </div>
                    <CardDescription className="flex items-center gap-1
text-foreground/70 text-sm mt-1">
                      <MapPinIcon className="h-3 w-3 sm:h-4 sm:w-4 shrink-
0" />
                      {trip.destination}
                    </CardDescription>
                  </CardHeader>
                  <CardContent className="py-1 sm:py-2 pt-0">
                    <div className="flex items-center gap-1 text-xs
sm:text-sm text-muted-foreground">
                      <CalendarIcon className="h-3 w-3 sm:h-4 sm:w-4
shrink-0" />
                      <span>
                        {formatDate(trip.startDate)} -
{formatDate(trip.endDate)}
                      </span>
                    </div>

                    {trip.description && (
                      <div className="mt-3 pt-2 border-t border-border/40">
                        <Button
                          variant="ghost"
                          size="sm"
                          className="p-0 h-auto w-full flex justify-between
items-center text-muted-foreground text-xs sm:text-sm hover:bg-transparent
hover:text-foreground transition-colors"
                          onClick={() =>
setShowDescription(!showDescription)}
                        >
                          <span className="font-medium">Description</span>
                          <span
                            className={`transform transition-transform
duration-200 ${
                              showDescription ? "rotate-180" : "rotate-0"
                            }`}
```

```
                        >
                          <ChevronDown className="h-3 w-3 sm:h-4 sm:w-4"
/>
                        </span>
                      </Button>

                      {showDescription && (
                        <div className="text-xs sm:text-sm text-muted-
foreground mt-1.5 pl-0.5 animate-in fade-in-50 slide-in-from-top-1
duration-200">
                          {trip.description}
                        </div>
                      )}
                    </div>
                  )}
                </CardContent>
              </div>

              <div className="border-l pl-4 mr-4 hidden sm:flex items-
center justify-end gap-6">
                <div className="text-center">
                  <div className="text-sm text-muted-
foreground">Duration</div>
                  <div className="text-xl font-bold">
                    {(() => {
                      const duration =
                        Math.ceil(
                          (new Date(trip.endDate).getTime() -
                            new Date(trip.startDate).getTime()) /
                            (1000 * 60 * 60 * 24)
                        ) + 1;
                      return `${duration} ${duration === 1 ? "day" :
"days"}`;
                    })()}
                  </div>
                </div>

                <div className="text-center">
                  <div className="text-sm text-muted-
foreground">Budget</div>
                  <div className="text-xl font-bold">
                    {currency(budgetService.getTotalBudget(budgetItems),
{
                      symbol: "$",
                      precision: 2,
                    }).format()}
                  </div>
                </div>

                <div className="text-center">
                  <div className="text-sm text-muted-
foreground">Events</div>
                  <div className="text-xl font-
bold">{events.length}</div>
                </div>
              </div>
            </div>
          </Card>

          <div className="lg:hidden mb-4">
            <Tabs
              defaultValue="timeline"
              value={activeTab}
              onValueChange={setActiveTab}
```

```
    className="w-full"
  >
    <TabsList className="grid grid-cols-4 w-full mb-1 h-auto">
      <TabsTrigger
        value="timeline"
        className="text-xs py-2 flex flex-col items-center"
      >
        <CalendarDaysIcon className="h-4 w-4 mb-1" />
        <span>Timeline</span>
      </TabsTrigger>
      <TabsTrigger
        value="budget"
        className="text-xs py-2 flex flex-col items-center"
      >
        <DollarSignIcon className="h-4 w-4 mb-1" />
        <span>Budget</span>
      </TabsTrigger>
      <TabsTrigger
        value="files"
        className="text-xs py-2 flex flex-col items-center"
      >
        <FolderIcon className="h-4 w-4 mb-1" />
        <span>Files</span>
      </TabsTrigger>
      <TabsTrigger
        value="checklist"
        className="text-xs py-2 flex flex-col items-center"
      >
        <CheckSquareIcon className="h-4 w-4 mb-1" />
        <span>Checklist</span>
      </TabsTrigger>
    </TabsList>

    <TabsContent value="timeline" className="mt-0 p-0">
      <div className="space-y-4">
        <Timeline
          dateRange={dateRange}
          selectedDate={selectedDate}
          onSelectDate={setSelectedDate}
          getEventsForDay={getEventsForDay}
        />

        {selectedDate && (
          <DayView
            selectedDate={selectedDate}
            events={events}
            weather={weather}
            tripId={tripId}
            onAddEvent={handleAddEvent}
            onEditEvent={handleEditEvent}
            onDeleteEvent={handleDeleteEvent}
            getEventsForDay={getEventsForDay}
          />
        )}
      </div>
    </TabsContent>

    <TabsContent value="budget" className="mt-0 p-0">
      <BudgetPanel
        budgetItems={budgetItems}
        dateRange={dateRange}
        onAddExpense={handleAddCustomExpense}
        onEditExpense={handleEditCustomExpense}
        onDeleteExpense={handleDeleteCustomExpense}
```

```
                />
              </TabsContent>

              <TabsContent value="files" className="mt-0 p-0">
                <TripFiles tripId={tripId} key={`trip-files-${tripId}`}
/>
              </TabsContent>

              <TabsContent value="checklist" className="mt-0 p-0">
                <ChecklistPanel tripId={tripId} />
              </TabsContent>
            </Tabs>
          </div>

          <div className="hidden lg:grid grid-cols-3 gap-8">
            <div className="lg:col-span-2 space-y-4 sm:space-y-6
lg:space-y-8">
              <Timeline
                dateRange={dateRange}
                selectedDate={selectedDate}
                onSelectDate={setSelectedDate}
                getEventsForDay={getEventsForDay}
              />

              {selectedDate && (
                <DayView
                  selectedDate={selectedDate}
                  events={events}
                  weather={weather}
                  tripId={tripId}
                  onAddEvent={handleAddEvent}
                  onEditEvent={handleEditEvent}
                  onDeleteEvent={handleDeleteEvent}
                  getEventsForDay={getEventsForDay}
                />
              )}
            </div>

            <div className="space-y-4 sm:space-y-6 lg:space-y-8">
              <BudgetPanel
                budgetItems={budgetItems}
                dateRange={dateRange}
                onAddExpense={handleAddCustomExpense}
                onEditExpense={handleEditCustomExpense}
                onDeleteExpense={handleDeleteCustomExpense}
              />

              <TripFiles tripId={tripId} key={`trip-files-${tripId}`} />

              <ChecklistPanel tripId={tripId} />
            </div>
          </div>

          {showEventDialog && (
            <EventDialog
              open={showEventDialog}
              onClose={() => setShowEventDialog(false)}
              eventType={selectedEventType}
              isEditing={isEditing}
              initialData={
                isEditing && selectedEvent
                  ? mapEventToFormData(selectedEvent)
                  : undefined
              }
```

```
              onSubmit={handleEventSubmit}
              tripStartDate={trip ? new Date(trip.startDate) : undefined}
              tripEndDate={trip ? new Date(trip.endDate) : undefined}
            />
          )}

          <Toaster />
        </div>
      );
    }
```

Фрагменти коду вікна з подорожами (./app/trips/page.tsx)

```
    "use client";

    import { AuthStatus } from "@/components/AuthStatus";
    import { CreateTripModal } from "@/components/CreateTripModal";
    import { DeleteTripModal } from "@/components/DeleteTripModal";
    import { NavLogo } from "@/components/NavLogo";
    import { TripCard } from "@/components/TripCard";
    import { Button } from "@/components/ui/button";
    import {
      Card,
      CardContent,
      CardDescription,
      CardHeader,
      CardTitle,
    } from "@/components/ui/card";
    import { Separator } from "@/components/ui/separator";
    import { Tabs, TabsContent, TabsList, TabsTrigger } from
"@/components/ui/tabs";
    import {
      CalendarDaysIcon,
      FilterIcon,
      LogOut,
      MapIcon,
      PlaneTakeoffIcon,
      PlusCircle,
    } from "lucide-react";
    import { signOut, useSession } from "next-auth/react";
    import { useRouter } from "next/navigation";
    import { useEffect, useState } from "react";

    interface Trip {
      id: string;
      name: string;
      destination: string;
      startDate: string;
      endDate: string;
      description: string | null;
    }

    export default function TripsPage() {
      const router = useRouter();
      const { data: session } = useSession();
      const [isLoggingOut, setIsLoggingOut] = useState(false);
      const [isModalOpen, setIsModalOpen] = useState(false);
      const [trips, setTrips] = useState<Trip[]>([]);
      const [isLoading, setIsLoading] = useState(true);
```

```
const [error, setError] = useState<string | null>(null);
const [activeTab, setActiveTab] = useState<string>("all");
const [isDeleteModalOpen, setIsDeleteModalOpen] = useState(false);
const [tripToDelete, setTripToDelete] = useState<{
  id: string;
  name: string;
} | null>(null);
const [isDeleting, setIsDeleting] = useState(false);

useEffect(() => {
  const fetchTrips = async () => {
    setIsLoading(true);
    try {
      const response = await fetch("/api/trips");

      if (!response.ok) {
        throw new Error("Failed to fetch trips");
      }

      const data = await response.json();
      setTrips(data.trips);
    } catch (err) {
      console.error("Error fetching trips:", err);
      setError("Failed to load trips. Please try again later.");
    } finally {
      setIsLoading(false);
    }
  };

  fetchTrips();
}, []);

const handleLogout = async () => {
  setIsLoggingOut(true);
  try {
    if (session) {
      await signOut({ redirect: false });
    } else {
      await fetch("/api/auth/logout", {
        method: "POST",
        headers: {
          "Content-Type": "application/json",
        },
      });
    }
    router.push("/login");
  } catch (error) {
    console.error("Logout failed:", error);
  } finally {
    setIsLoggingOut(false);
  }
};

const openModal = () => setIsModalOpen(true);
const closeModal = () => setIsModalOpen(false);

const handleDeleteTrip = (id: string, name: string) => {
  setTripToDelete({ id, name });
  setIsDeleteModalOpen(true);
};

const closeDeleteModal = () => {
  setIsDeleteModalOpen(false);
  setTripToDelete(null);
```

```
};

const confirmDeleteTrip = async () => {
  if (!tripToDelete) return;

  setIsDeleting(true);
  try {
    const response = await fetch(`/api/trips/${tripToDelete.id}`, {
      method: "DELETE",
    });

    if (!response.ok) {
      throw new Error("Failed to delete trip");
    }

    setTrips(trips.filter((trip) => trip.id !== tripToDelete.id));
    closeDeleteModal();
  } catch (err) {
    console.error("Error deleting trip:", err);
    setError("Failed to delete trip. Please try again later.");
  } finally {
    setIsDeleting(false);
  }
};

const getTripsGroupedByMonth = () => {
  const grouped: Record<string, Trip[]> = {};

  trips.forEach((trip) => {
    const date = new Date(trip.startDate);
    const monthYear = date.toLocaleString("default", {
      month: "long",
      year: "numeric",
    });

    if (!grouped[monthYear]) {
      grouped[monthYear] = [];
    }

    grouped[monthYear].push(trip);
  });

  return grouped;
};

const getUpcomingTrips = () => {
  const now = new Date();
  const thirtyDaysFromNow = new Date();
  thirtyDaysFromNow.setDate(now.getDate() + 30);

  return trips.filter((trip) => {
    const startDate = new Date(trip.startDate);
    return startDate >= now && startDate <= thirtyDaysFromNow;
  });
};

const getPastTrips = () => {
  const now = new Date();

  return trips.filter((trip) => {
    const endDate = new Date(trip.endDate);
    return endDate < now;
  });
};
```

```
        const tripsGroupedByMonth = getTripsGroupedByMonth();
        const upcomingTrips = getUpcomingTrips();
        const pastTrips = getPastTrips();

        return (
          <div className="container mx-auto px-4 py-6 sm:py-8">
            <header className="flex flex-col sm:flex-row justify-between
items-start sm:items-center mb-4 sm:mb-8 gap-2 sm:gap-4">
              <div className="flex flex-col">
                <NavLogo />
                <p className="text-muted-foreground text-sm mt-1">
                  Manage your travel plans
                </p>
              </div>

              <div className="hidden sm:flex items-center justify-center">
                <AuthStatus />
              </div>

              <div className="flex gap-2 w-full sm:w-auto mt-2 sm:mt-0">
                <Button
                  onClick={openModal}
                  className="gap-1 sm:gap-2 flex-1 sm:flex-initial justify-
center"
                >
                  <PlusCircle className="h-4 w-4" />
                  <span>New trip</span>
                </Button>
                <Button
                  variant="outline"
                  onClick={handleLogout}
                  disabled={isLoggingOut}
                  className="gap-1 sm:gap-2 flex-1 sm:flex-initial justify-
center"
                >
                  <LogOut className="h-4 w-4" />
                  <span>{isLoggingOut ? "Logging out..." : "Logout"}</span>
                </Button>
              </div>
            </header>

            <Separator className="my-4 sm:my-6" />

            {isLoading ? (
              <div className="flex justify-center items-center h-48 sm:h-
64">
                <div className="h-10 w-10 sm:h-12 sm:w-12 rounded-full
border-4 border-primary border-t-transparent animate-spin"></div>
              </div>
            ) : error ? (
              <Card className="bg-destructive/10 border-destructive/20">
                <CardContent className="p-4 sm:p-6">
                  <p className="text-destructive">{error}</p>
                </CardContent>
              </Card>
            ) : trips.length === 0 ? (
              <div className="flex items-center justify-center h-
[calc(100vh-250px)]">
                <Card className="bg-muted/40 w-full max-w-md">
                  <CardHeader className="py-3 sm:py-6">
                    <CardTitle className="text-xl sm:text-2xl">
                      Get started with your first trip
                    </CardTitle>
```

```
                    <CardDescription className="text-sm sm:text-base">
                      You don't have any trips yet.
                    </CardDescription>
                  </CardHeader>
                  <CardContent className="flex flex-col items-center p-4
sm:p-6">
                    <p className="text-center text-sm sm:text-base mb-4
sm:mb-6">
                      Create your first trip to start planning your
journey!
                    </p>
                    <Button
                      onClick={openModal}
                      size="lg"
                      className="gap-2 w-full sm:w-auto"
                    >
                      <PlusCircle className="h-4 w-4" />
                      Create your first trip
                    </Button>
                  </CardContent>
                </Card>
              </div>
            ) : (
              <>
                {/* Trip tabs - especially useful on mobile */}
                <Tabs
                  defaultValue="all"
                  value={activeTab}
                  onValueChange={setActiveTab}
                  className="w-full mb-6"
                >
                  <TabsList className="grid grid-cols-4 w-full mb-4 h-
auto">
                    <TabsTrigger
                      value="all"
                      className="text-xs py-2 flex flex-col items-center"
                    >
                      <FilterIcon className="h-4 w-4 mb-1" />
                      <span>All</span>
                    </TabsTrigger>
                    <TabsTrigger
                      value="upcoming"
                      className="text-xs py-2 flex flex-col items-center"
                    >
                      <CalendarDaysIcon className="h-4 w-4 mb-1" />
                      <span>Upcoming</span>
                    </TabsTrigger>
                    <TabsTrigger
                      value="past"
                      className="text-xs py-2 flex flex-col items-center"
                    >
                      <PlaneTakeoffIcon className="h-4 w-4 mb-1" />
                      <span>Past</span>
                    </TabsTrigger>
                    <TabsTrigger
                      value="by-month"
                      className="text-xs py-2 flex flex-col items-center"
                    >
                      <MapIcon className="h-4 w-4 mb-1" />
                      <span>By Month</span>
                    </TabsTrigger>
                  </TabsList>

                  <TabsContent value="all" className="mt-0 p-0">
```

```
                    <div className="grid grid-cols-1 sm:grid-cols-2
lg:grid-cols-3 gap-4 sm:gap-6">
                      {trips.map((trip) => (
                        <TripCard
                          key={trip.id}
                          id={trip.id}
                          name={trip.name}
                          destination={trip.destination}
                          startDate={trip.startDate}
                          endDate={trip.endDate}
                          description={trip.description || undefined}
                          onDelete={handleDeleteTrip}
                        />
                      ))}
                    </div>
                  </TabsContent>

                  <TabsContent value="upcoming" className="mt-0 p-0">
                    {upcomingTrips.length > 0 ? (
                      <div className="grid grid-cols-1 sm:grid-cols-2
lg:grid-cols-3 gap-4 sm:gap-6">
                        {upcomingTrips.map((trip) => (
                          <TripCard
                            key={trip.id}
                            id={trip.id}
                            name={trip.name}
                            destination={trip.destination}
                            startDate={trip.startDate}
                            endDate={trip.endDate}
                            description={trip.description || undefined}
                            onDelete={handleDeleteTrip}
                          />
                        ))}
                      </div>
                    ) : (
                      <Card className="bg-muted/20">
                        <CardContent className="p-6 text-center">
                          <p className="text-muted-foreground">
                            No upcoming trips in the next 30 days.
                          </p>
                          <Button
                            onClick={openModal}
                            size="sm"
                            className="mt-4 gap-1"
                          >
                            <PlusCircle className="h-4 w-4" />
                            Plan a new trip
                          </Button>
                        </CardContent>
                      </Card>
                    )}
                  </TabsContent>

                  <TabsContent value="past" className="mt-0 p-0">
                    {pastTrips.length > 0 ? (
                      <div className="grid grid-cols-1 sm:grid-cols-2
lg:grid-cols-3 gap-4 sm:gap-6">
                        {pastTrips.map((trip) => (
                          <TripCard
                            key={trip.id}
                            id={trip.id}
                            name={trip.name}
                            destination={trip.destination}
                            startDate={trip.startDate}
```

```
                            endDate={trip.endDate}
                            description={trip.description || undefined}
                            onDelete={handleDeleteTrip}
                          />
                        ))}
                      </div>
                    ) : (
                      <Card className="bg-muted/20">
                        <CardContent className="p-6 text-center">
                          <p className="text-muted-foreground">No past
trips.</p>
                        </CardContent>
                      </Card>
                    )}
                  </TabsContent>

                  <TabsContent value="by-month" className="mt-0 p-0">
                    <div className="space-y-8">
                      {Object.entries(tripsGroupedByMonth).map(
                        ([monthYear, monthTrips]) => (
                          <div key={monthYear} className="space-y-4">
                            <h2 className="text-xl font-
semibold">{monthYear}</h2>
                            <div className="grid grid-cols-1 sm:grid-cols-2
lg:grid-cols-3 gap-4 sm:gap-6">
                              {monthTrips.map((trip) => (
                                <TripCard
                                  key={trip.id}
                                  id={trip.id}
                                  name={trip.name}
                                  destination={trip.destination}
                                  startDate={trip.startDate}
                                  endDate={trip.endDate}
                                  description={trip.description ||
undefined}
                                  onDelete={handleDeleteTrip}
                                />
                              ))}
                            </div>
                          </div>
                        )
                      )}
                    </div>
                  </TabsContent>
                </Tabs>
              </>
            )}

            <CreateTripModal isOpen={isModalOpen} onClose={closeModal} />

            {tripToDelete && (
              <DeleteTripModal
                isOpen={isDeleteModalOpen}
                onClose={closeDeleteModal}
                onDelete={confirmDeleteTrip}
                tripName={tripToDelete.name}
                isDeleting={isDeleting}
              />
            )}
          </div>
        );
      }
```

Фрагменти коду вікна входу (./app/login/page.tsx)

```
"use client";

import { NavLogo } from "@/components/NavLogo";
import { Button } from "@/components/ui/button";
import {
  Card,
  CardContent,
  CardDescription,
  CardFooter,
  CardHeader,
  CardTitle,
} from "@/components/ui/card";
import { Input } from "@/components/ui/input";
import { Label } from "@/components/ui/label";
import { EyeIcon, EyeOffIcon, Lock } from "lucide-react";
import { signIn } from "next-auth/react";
import Link from "next/link";
import { useRouter, useSearchParams } from "next/navigation";
import React, { Suspense, useEffect, useState } from "react";

function LoginForm() {
  const [email, setEmail] = useState("");
  const [password, setPassword] = useState("");
  const [showPassword, setShowPassword] = useState(false);
  const [error, setError] = useState("");
  const [loading, setLoading] = useState(false);
  const [googleLoading, setGoogleLoading] = useState(false);
  const [showDashboardAccess, setShowDashboardAccess] =
useState(false);
  const [dashboardPassword, setDashboardPassword] = useState("");
  const [showDashboardPassword, setShowDashboardPassword] =
useState(false);
  const [dashboardLoading, setDashboardLoading] = useState(false);
  const [dashboardError, setDashboardError] = useState("");
  const router = useRouter();
  const searchParams = useSearchParams();

  useEffect(() => {
    const message = searchParams.get("message");
    if (message) {
      setError(message);
    }
  }, [searchParams]);

  const isFormValid = email.trim() !== "" && password.trim() !== "";
  const isDashboardFormValid = dashboardPassword.trim() !== "";

  const handleSubmit = async (e: React.FormEvent) => {
    e.preventDefault();

    if (!isFormValid) return;

    setLoading(true);
    setError("");

    try {
      const response = await fetch("/api/auth/login", {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify({ email, password }),
```

```
            credentials: "include",
          });

          if (!response.ok) {
            let errorMessage = "Login failed";
            try {
              const errorData = await response.json();
              errorMessage = errorData.message || errorMessage;
            } catch (parseError) {
              console.error("Error parsing response:", parseError);
              errorMessage = `Error: ${response.status}
${response.statusText}`;
            }
            throw new Error(errorMessage);
          }

          let data;
          try {
            data = await response.json();
            if (!data.success) {
              throw new Error("Server responded but login was
unsuccessful");
            }
          } catch (parseError) {
            console.error("Error parsing success response:", parseError);
            throw new Error("Error processing server response");
          }

          setTimeout(() => {
            router.push("/trips");
          }, 100);
        } catch (err: any) {
          console.error("Login error:", err);
          setError(err.message || "An error occurred during login");
        } finally {
          setLoading(false);
        }
      };

      const handleDashboardAccess = async (e: React.FormEvent) => {
        e.preventDefault();

        if (!isDashboardFormValid) return;

        setDashboardLoading(true);
        setDashboardError("");

        try {
          const response = await fetch("/api/dashboard/access", {
            method: "POST",
            headers: { "Content-Type": "application/json" },
            body: JSON.stringify({ secretKey: dashboardPassword }),
            credentials: "include",
          });

          if (!response.ok) {
            let errorMessage = "Invalid dashboard password";

            try {
              const errorData = await response.json();
              errorMessage = errorData.message || errorMessage;
            } catch (parseError) {
              console.error("Error parsing response:", parseError);
```

```
                errorMessage = `Error: ${response.status}
${response.statusText}`;
              }

              throw new Error(errorMessage);
            }

            let data;
            try {
              data = await response.json();
              if (!data.success) {
                throw new Error(
                  "Server responded but dashboard access was unsuccessful"
                );
              }
            } catch (parseError) {
              console.error("Error parsing success response:", parseError);
              throw new Error("Error processing server response");
            }

            setTimeout(() => {
              router.push("/dashboard");
            }, 100);
          } catch (err: any) {
            console.error("Dashboard access error:", err);
            setDashboardError(
              err.message || "An error occurred while validating dashboard
access"
            );
          } finally {
            setDashboardLoading(false);
          }
        };

        const handleGoogleSignIn = async () => {
          try {
            setGoogleLoading(true);
            setError("");
            await signIn("google", { callbackUrl: "/trips" });
          } catch (err: any) {
            console.error("Google sign-in error:", err);
            setError(err.message || "An error occurred during Google sign-
in");
            setGoogleLoading(false);
          }
        };

        return (
          <div className="flex min-h-screen flex-col items-center justify-
center px-4 py-8 sm:py-12">
            <div className="mb-8">
              <NavLogo />
            </div>
            <Card className="w-full max-w-md">
              <CardHeader className="space-y-1 py-3 sm:py-6">
                <CardTitle className="text-xl sm:text-2xl font-bold text-
center">
                  Sign in to your account
                </CardTitle>
                <CardDescription className="text-center text-sm sm:text-
base">
                  Enter your credentials below
                </CardDescription>
              </CardHeader>
```

```
              <CardContent className="py-3 sm:py-6">
                {error && (
                  <div className="bg-destructive/15 text-destructive text-
sm p-3 rounded-md mb-4">
                    {error}
                  </div>
                )}

                <form onSubmit={handleSubmit} className="space-y-4">
                  <div className="space-y-2">
                    <Label htmlFor="email">Email</Label>
                    <Input
                      id="email"
                      type="email"
                      placeholder="your.email@example.com"
                      value={email}
                      onChange={(e) => setEmail(e.target.value)}
                      disabled={loading}
                      required
                    />
                  </div>
                  <div className="space-y-2">
                    <Label htmlFor="password">Password</Label>
                    <div className="relative">
                      <Input
                        id="password"
                        type={showPassword ? "text" : "password"}
                        value={password}
                        onChange={(e) => setPassword(e.target.value)}
                        disabled={loading}
                        required
                      />
                      <Button
                        type="button"
                        variant="ghost"
                        size="icon"
                        className="absolute right-0 top-0 h-10 w-10"
                        onClick={() => setShowPassword(!showPassword)}
                        aria-label={showPassword ? "Hide password" : "Show
password"}
                        tabIndex={0}
                        onKeyDown={(e) => {
                          if (e.key === "Enter" || e.key === " ") {
                            e.preventDefault();
                            setShowPassword(!showPassword);
                          }
                        }}
                      >
                        {showPassword ? (
                          <EyeOffIcon className="h-4 w-4" />
                        ) : (
                          <EyeIcon className="h-4 w-4" />
                        )}
                      </Button>
                    </div>
                  </div>
                  <Button
                    type="submit"
                    className="w-full"
                    disabled={!isFormValid || loading}
                  >
                    {loading ? (
                      <>
```

```
                      <span className="mr-2">Signing in</span>
                      <div className="h-4 w-4 border-2 border-current
border-t-transparent animate-spin rounded-full"></div>
                    </>
                  ) : (
                    "Sign in"
                  )}
                </Button>
              </form>

              <div className="mt-4 flex items-center gap-2">
                <div className="h-px flex-1 bg-muted"></div>
                <p className="text-xs text-muted-foreground">OR</p>
                <div className="h-px flex-1 bg-muted"></div>
              </div>

              <Button
                variant="outline"
                className="w-full mt-4 flex items-center gap-2"
                onClick={handleGoogleSignIn}
                disabled={googleLoading}
              >
                {googleLoading ? (
                  <>
                    <span className="mr-2">Connecting</span>
                    <div className="h-4 w-4 border-2 border-current
border-t-transparent animate-spin rounded-full"></div>
                  </>
                ) : (
                  <>
                    <svg className="h-5 w-5" viewBox="0 0 24 24">
                      <path
                        d="M22.56 12.25c0-.78-.07-1.53-.2-
2.25H12v4.26h5.92c-.26 1.37-1.04 2.53-2.21 3.31v2.77h3.57c2.08-1.92 3.28-
4.74 3.28-8.09z"
                        fill="#4285F4"
                      />
                      <path
                        d="M12 23c2.97 0 5.46-.98 7.28-2.66l-3.57-2.77c-
.98.66-2.23 1.06-3.71 1.06-2.86 0-5.29-1.93-6.16-4.53H2.18v2.84C3.99 20.53
7.7 23 12 23z"
                        fill="#34A853"
                      />
                      <path
                        d="M5.84 14.09c-.22-.66-.35-1.36-.35-2.09s.13-
1.43.35-2.09V7.07H2.18C1.43 8.55 1 10.22 1 12s.43 3.45 1.18 4.93l2.85-
2.22.81-.62z"
                        fill="#FBBC05"
                      />
                      <path
                        d="M12 5.38c1.62 0 3.06.56 4.21 1.64l3.15-
3.15C17.45 2.09 14.97 1 12 1 7.7 1 3.99 3.47 2.18 7.07l3.66 2.84c.87-2.6
3.3-4.53 6.16-4.53z"
                        fill="#EA4335"
                      />
                      <path d="M1 1h22v22H1z" fill="none" />
                    </svg>
                    Sign in with Google
                  </>
                )}
              </Button>
            </CardContent>

            <CardFooter className="px-6 py-4 flex justify-center">
```

```
                <div className="text-sm text-center">
                  Don&apos;t have an account?{" "}
                  <Link
                    href="/register"
                    className="font-medium text-primary hover:underline"
                  >
                    Sign up
                  </Link>
                </div>
              </CardFooter>
            </Card>

            <div className="mt-8 w-full max-w-md">
              <Button
                variant="outline"
                className="w-full py-3 text-sm text-muted-foreground flex
items-center justify-center gap-2"
                onClick={() =>
setShowDashboardAccess(!showDashboardAccess)}
              >
                <Lock className="h-4 w-4" />
                {showDashboardAccess
                  ? "Hide Dashboard Access"
                  : "Admin Dashboard Access"}
              </Button>

              {showDashboardAccess && (
                <Card className="mt-4">
                  <CardHeader className="pb-0">
                    <CardTitle className="text-xl">Admin
Dashboard</CardTitle>
                  </CardHeader>
                  <CardContent className="pt-6 pb-6 px-6">
                    <form onSubmit={handleDashboardAccess}
className="space-y-6">
                      <div className="space-y-3">
                        <Label
                          htmlFor="dashboard-password"
                          className="text-sm font-medium"
                        >
                          Enter admin password
                        </Label>
                        <div className="relative">
                          <Input
                            id="dashboard-password"
                            type={showDashboardPassword ? "text" :
"password"}
                            value={dashboardPassword}
                            onChange={(e) =>
setDashboardPassword(e.target.value)}
                            disabled={dashboardLoading}
                            required
                            className="h-11"
                          />
                          <Button
                            type="button"
                            variant="ghost"
                            size="icon"
                            className="absolute right-0 top-0 h-11 w-10"
                            onClick={() =>
                              setShowDashboardPassword(!showDashboardPasswo
rd)
                            }
                            aria-label={
```

```
                              showDashboardPassword
                                ? "Hide password"
                                : "Show password"
                            }
                            tabIndex={0}
                            onKeyDown={(e) => {
                              if (e.key === "Enter" || e.key === " ") {
                                e.preventDefault();
                                setShowDashboardPassword(!showDashboardPass
word);
                              }
                            }}
                          >
                            {showDashboardPassword ? (
                              <EyeOffIcon className="h-4 w-4" />
                            ) : (
                              <EyeIcon className="h-4 w-4" />
                            )}
                          </Button>
                        </div>
                      </div>

                      {dashboardError && (
                        <div className="bg-destructive/15 text-destructive
text-sm p-3 rounded-md my-2">
                          {dashboardError}
                          <div className="mt-2 text-xs italic">
                            If you're accessing from a mobile device,
please make sure
                            you're on the same network as the server.
                          </div>
                        </div>
                      )}

                      <Button
                        type="submit"
                        className="w-full h-11 mt-4"
                        disabled={!isDashboardFormValid ||
dashboardLoading}
                      >
                        {dashboardLoading ? (
                          <>
                            <span className="mr-2">Verifying</span>
                            <div className="h-5 w-5 border-2 border-current
border-t-transparent animate-spin rounded-full"></div>
                          </>
                        ) : (
                          "Access Dashboard"
                        )}
                      </Button>
                    </form>
                  </CardContent>
                </Card>
              )}
            </div>
          </div>
        );
      }

      export default function LoginPage() {
        return (
          <Suspense
            fallback={
```

```
              <div className="flex min-h-screen flex-col items-center
justify-center">
                <div className="h-8 w-8 animate-spin rounded-full border-4
border-primary border-t-transparent"></div>
              </div>
            }
          >
            <LoginForm />
          </Suspense>
        );
      }
```

Фрагменти коду компоненту хронології (/components/trip/Timeline.tsx)

```
      "use client";

      import { Button } from "@/components/ui/button";
      import { Card, CardContent, CardHeader, CardTitle } from
"@/components/ui/card";
      import { CalendarDaysIcon } from "lucide-react";
      import { cn } from "@/lib/utils";

      interface TimelineProps {
        dateRange: string[];
        selectedDate: string | null;
        onSelectDate: (date: string) => void;
        getEventsForDay: (date: string) => any[];
      }

      export const Timeline = ({
        dateRange,
        selectedDate,
        onSelectDate,
        getEventsForDay,
      }: TimelineProps) => {
        const getFormattedDate = (dateString: string) => {
          return dateString.split("T")[0];
        };

        return (
          <Card>
            <CardHeader className="py-3 sm:py-4">
              <CardTitle className="flex items-center gap-2 text-lg
sm:text-xl">
                <CalendarDaysIcon className="h-4 w-4 sm:h-5 sm:w-5" />
                Timeline
              </CardTitle>
            </CardHeader>
            <CardContent className="pt-0 sm:pt-4">
              <div className="flex overflow-x-auto pb-2 sm:pb-4 space-x-2
sm:space-x-4 -mx-4 px-4 sm:mx-0 sm:px-0">
                {dateRange.map((date) => {
                  const dayEvents = getEventsForDay(date);
                  const isSelected = date === selectedDate;
                  const dateObj = new Date(date);
                  const dayOfMonth = dateObj.getDate();
                  const month = dateObj.toLocaleString("default", {
                    month: "short",
                  });

                  const endingEvents = dayEvents.filter((event) => {
                    if (!event.endTs) return false;
                    const eventEndDate = getFormattedDate(event.endTs);
                    const eventStartDate = getFormattedDate(event.startTs);
```

```
                    return eventEndDate === date && eventStartDate !==
date;
                  });

                  const oneDayEvents = dayEvents.filter((event) => {
                    if (!event.endTs) return false;
                    const eventEndDate = getFormattedDate(event.endTs);
                    const eventStartDate = getFormattedDate(event.startTs);
                    return eventEndDate === eventStartDate &&
eventStartDate === date;
                  });

                  const startingEvents = dayEvents.filter((event) => {
                    if (!event.endTs) return true; // Events without end
date are treated as starting events
                    const eventEndDate = getFormattedDate(event.endTs);
                    const eventStartDate = getFormattedDate(event.startTs);

                    // Include only events that start on this date AND are
NOT one-day events
                    return eventStartDate === date && eventEndDate !==
eventStartDate;
                  }).length;

                  return (
                    <Button
                      key={date}
                      variant={isSelected ? "default" : "outline"}
                      className={cn(
                        "flex-shrink-0 flex flex-col h-auto py-1 sm:py-2
px-2 sm:px-3 min-h-[4.5rem] min-w-[3.5rem]",
                        isSelected
                          ? "border-2 border-primary"
                          : "border-2 border-transparent",
                        dayEvents.length > 0
                          ? "ring-2 ring-primary/20"
                          : "ring-2 ring-transparent"
                      )}
                      onClick={() => onSelectDate(date)}
                    >
                      <span className="text-xs sm:text-sm">{month}</span>
                      <span className="text-lg sm:text-xl font-bold">
                        {dayOfMonth}
                      </span>
                      <span className="text-[10px] sm:text-xs mt-1 min-h-
[14px] text-center">
                        {dayEvents.length > 0 && (
                          <>
                            {oneDayEvents.length > 0 && (
                              <div className="text-orange-500 dark:text-
orange-400">
                                {oneDayEvents.length}{" "}
                                {oneDayEvents.length === 1 ? "event" :
"events"}
                              </div>
                            )}
                            {startingEvents > 0 && (
                              <div className="text-green-500 dark:text-
green-400">
                                {startingEvents}{" "}
                                {startingEvents === 1 ? "event" : "events"}
start
                                {startingEvents === 1 ? "s" : ""}
                              </div>
```

```
                            )}
                            {endingEvents.length > 0 && (
                              <div className="text-blue-500 dark:text-blue-
400">
                                {endingEvents.length}{" "}
                                {endingEvents.length === 1 ? "event" :
"events"} end
                                {endingEvents.length === 1 ? "s" : ""}
                              </div>
                            )}
                          </>
                        )}
                      </span>
                    </Button>
                  );
                })}
              </div>
            </CardContent>
          </Card>
        );
      };
```

Фрагменти коду компоненту списку подій на день (./components/trip/EventList.tsx)

```
"use client";

import { EventAttachments } from "@/components/EventAttachments";
import { Button } from "@/components/ui/button";
import { Card } from "@/components/ui/card";
import { Event } from "@/lib/types/trip";
import { formatTime } from "@/lib/utils/date-utils";
import {
  BuildingIcon,
  CalendarIcon,
  CarIcon,
  MapPinIcon,
  PencilIcon,
  PlaneIcon,
  PhoneIcon,
  TicketIcon,
  TrashIcon,
  UtensilsIcon,
  ChevronDownIcon,
  ChevronUpIcon,
} from "lucide-react";
import { useState } from "react";
import { cn } from "@/lib/utils";

interface EventListProps {
  events: Event[];
  tripId: string;
  onEditEvent: (event: Event) => void;
  onDeleteEvent: (eventId: string) => void;
  selectedDate?: string;
}

export const EventList = ({
  events,
```

```
        tripId,
        onEditEvent,
        onDeleteEvent,
        selectedDate,
      }: EventListProps) => {
        const [expandedEventId, setExpandedEventId] = useState<string |
null>(null);

        const toggleExpand = (eventId: string) => {
          setExpandedEventId(expandedEventId === eventId ? null : eventId);
        };

        if (events.length === 0) {
          return (
            <div className="flex flex-col items-center justify-center p-4
bg-muted/30 rounded-lg border border-dashed border-muted-foreground/30">
              <CalendarIcon className="h-8 w-8 text-muted-foreground mb-1
opacity-70" />
              <h3 className="font-medium text-sm">No Events Yet</h3>
              <p className="text-xs text-muted-foreground text-center mt-
1">
                No activities or appointments are scheduled for this day.
                <br />
                Use the "Add Event" button to start planning your day.
              </p>
            </div>
          );
        }

        const getFormattedDate = (dateString: string) => {
          return dateString.split("T")[0];
        };

        const formatDateTime = (dateString: string | null) => {
          if (!dateString) return "";
          const date = new Date(dateString);
          return date.toLocaleString(undefined, {
            weekday: "short",
            month: "short",
            day: "numeric",
            hour: "numeric",
            minute: "2-digit",
          });
        };

        return (
          <div className="space-y-2">
            {events.map((event) => {
              const isEndDate =
                selectedDate &&
                event.endTs &&
                getFormattedDate(event.endTs) === selectedDate &&
                getFormattedDate(event.startTs) !== selectedDate;

              const isStartDate =
                selectedDate && getFormattedDate(event.startTs) ===
selectedDate;

              const isMultiDayEvent =
                event.endTs &&
                getFormattedDate(event.startTs) !==
getFormattedDate(event.endTs);

              const isOneDay =
```

```
                event.endTs &&
                getFormattedDate(event.startTs) ===
getFormattedDate(event.endTs);

              const isExpanded = expandedEventId === event.id;

              return (
                <Card
                  key={event.id}
                  className={cn(
                    "p-2 relative transition-all duration-200 ease-in-out",
                    isExpanded ? "shadow-md" : "",
                    "hover:bg-accent/5 cursor-pointer",
                    isMultiDayEvent && isStartDate && !isEndDate
                      ? "bg-green-50/25 dark:bg-green-950/25"
                      : "",
                    isMultiDayEvent && isEndDate
                      ? "bg-blue-50/25 dark:bg-blue-950/25"
                      : "",
                    isOneDay ? "bg-orange-50/25 dark:bg-orange-950/25" : ""
                  )}
                  onClick={(e) => {
                    if (!(e.target as HTMLElement).closest("button")) {
                      toggleExpand(event.id);
                    }
                  }}
                >
                  <div className="flex justify-between items-start">
                    <div className="flex-1 min-w-0 pr-2">
                      <div className="flex items-start gap-1.5 min-w-0">
                        <span className="flex-shrink-0 mt-0.5">
                          {event.category === "hotel" && (
                            <BuildingIcon className="h-3.5 w-3.5" />
                          )}
                          {event.category === "restaurant" && (
                            <UtensilsIcon className="h-3.5 w-3.5" />
                          )}
                          {event.category === "flight" && (
                            <PlaneIcon className="h-3.5 w-3.5" />
                          )}
                          {event.category === "transport" && (
                            <CarIcon className="h-3.5 w-3.5" />
                          )}
                          {event.category === "activity" && (
                            <TicketIcon className="h-3.5 w-3.5" />
                          )}
                          {event.category === "other" && (
                            <CalendarIcon className="h-3.5 w-3.5" />
                          )}
                        </span>
                        <div className="min-w-0 flex-1 flex flex-col">
                          <div className="flex flex-wrap items-center gap-
x-1 gap-y-0.5 w-full">
                            <span className="font-semibold text-sm
truncate">
                              {event.name}
                            </span>
                            {isMultiDayEvent && isEndDate && (
                              <span className="inline-flex text-xs font-
normal bg-blue-100 dark:bg-blue-900 text-blue-700 dark:text-blue-300 px-1
py-0.5 rounded whitespace-nowrap flex-shrink-0">
                                End
                              </span>
                            )}
```

```
                            {isMultiDayEvent && isStartDate && !isEndDate
&& (
                              <span className="inline-flex text-xs font-
normal bg-green-100 dark:bg-green-900 text-green-700 dark:text-green-300
px-1 py-0.5 rounded whitespace-nowrap flex-shrink-0">
                                Start
                              </span>
                            )}
                            {isOneDay && (
                              <span className="inline-flex text-xs font-
normal bg-orange-100 dark:bg-orange-900 text-orange-700 dark:text-orange-
300 px-1 py-0.5 rounded whitespace-nowrap flex-shrink-0">
                                One-day
                              </span>
                            )}
                          </div>
                          <span className="text-xs text-muted-foreground
block truncate">
                            {isMultiDayEvent && isEndDate
                              ? "Ends "
                              : isMultiDayEvent && isStartDate
                              ? "Starts "
                              : ""}
                            {formatTime(
                              isEndDate && event.endTs ? event.endTs :
event.startTs
                            )}{" "}
                            {!isEndDate &&
                              event.endTs &&
                              !isMultiDayEvent &&
                              `- ${formatTime(event.endTs)}`}
                            {isMultiDayEvent && isStartDate && event.endTs
&& (
                              <span className="whitespace-nowrap">
                                {" - "}Ends{" "}
                                {new
Date(event.endTs).toLocaleDateString(undefined, {
                                  month: "short",
                                  day: "numeric",
                                })}{" "}
                                {formatTime(event.endTs)}
                              </span>
                            )}
                          </span>
                        </div>
                      </div>
                    </div>

                    <div className="flex items-center space-x-1 flex-
shrink-0">
                      <Button
                        variant="ghost"
                        size="sm"
                        onClick={(e) => {
                          e.stopPropagation();
                          onEditEvent(event);
                        }}
                        className={cn(
                          "h-6 w-6 p-0",
                          isEndDate ? "opacity-50 cursor-not-allowed" : ""
                        )}
                        disabled={!!isEndDate}
                        title={isEndDate ? "Edit from start date" : "Edit
event"}
```

```
                    >
                      <PencilIcon className="h-3.5 w-3.5" />
                      <span className="sr-only">Edit</span>
                    </Button>
                    <Button
                      variant="ghost"
                      size="sm"
                      onClick={(e) => {
                        e.stopPropagation();
                        onDeleteEvent(event.id);
                      }}
                      className="h-6 w-6 p-0 text-destructive"
                    >
                      <TrashIcon className="h-3.5 w-3.5" />
                      <span className="sr-only">Delete</span>
                    </Button>
                    <EventAttachments
                      tripId={tripId}
                      eventId={event.id}
                      className="ml-1"
                    />
                    <Button
                      variant="ghost"
                      size="sm"
                      onClick={(e) => {
                        e.stopPropagation();
                        toggleExpand(event.id);
                      }}
                      className="h-6 w-6 p-0"
                    >
                      {isExpanded ? (
                        <ChevronUpIcon className="h-3.5 w-3.5" />
                      ) : (
                        <ChevronDownIcon className="h-3.5 w-3.5" />
                      )}
                      <span className="sr-only">
                        {isExpanded ? "Collapse" : "Expand"}
                      </span>
                    </Button>
                  </div>
                </div>

                <div
                  className={cn(
                    "mt-1 text-xs grid grid-cols-1 gap-y-0.5",
                    !isExpanded &&
                      (event.address ||
                        event.phone ||
                        event.price ||
                        event.confirmation)
                      ? "border-t border-border/40 pt-1"
                      : ""
                  )}
                >
                  {event.address && (
                    <div className="flex items-center gap-1 text-muted-
foreground overflow-hidden">
                      <MapPinIcon className="h-3 w-3 flex-shrink-0" />
                      <span
                        className={cn(
                          "truncate",
                          isExpanded && "whitespace-normal break-words"
                        )}
                      >
```

```
                        {event.address}
                      </span>
                    </div>
                  )}

                  {event.phone && (
                    <div className="flex items-center gap-1 text-muted-
foreground overflow-hidden">
                      <PhoneIcon className="h-3 w-3 flex-shrink-0" />
                      <span className="truncate">{event.phone}</span>
                    </div>
                  )}

                  {(event.price || event.confirmation) && (
                    <div className="flex flex-wrap gap-x-3 gap-y-0.5
text-muted-foreground">
                      {event.price && (
                        <span className="flex-shrink-0 truncate">
                          <span className="font-medium">Price:</span>{"
"}
                          {event.currency || "$"}
                          {typeof event.price === "number"
                            ? event.price.toFixed(2)
                            : event.price}
                        </span>
                      )}

                      {event.confirmation && (
                        <span className="truncate max-w-full">
                          <span className="font-medium">Conf:</span>{" "}
                          <span
                            className={cn(
                              "truncate",
                              isExpanded && "whitespace-normal break-all"
                            )}
                          >
                            {event.confirmation}
                          </span>
                        </span>
                      )}
                    </div>
                  )}
                </div>

                {isExpanded && (
                  <div className="mt-3 pt-2 border-t border-border/40
text-xs animate-in fade-in-50 slide-in-from-top-1 duration-200">
                    <div className="grid grid-cols-1 sm:grid-cols-2 gap-
2">
                      <div>
                        <h5 className="font-medium mb-1">Event
Details</h5>
                        <ul className="space-y-1 text-muted-foreground">
                          <li>
                            <span className="font-
medium">Category:</span>{" "}
                            <span
className="capitalize">{event.category}</span>
                          </li>
                          <li>
                            <span className="font-medium">Start:</span>{"
"}
                            {formatDateTime(event.startTs)}
                          </li>
```

```
                            {event.endTs && (
                              <li>
                                <span className="font-medium">End:</span>{"
"}
                                {formatDateTime(event.endTs)}
                              </li>
                            )}
                            {event.details &&
                              Object.keys(event.details).length > 0 && (
                                <li className="mt-2">
                                  <span className="font-medium">
                                    Additional Details:
                                  </span>
                                  <div className="mt-1 pl-2 space-y-1">
                                    {Object.entries(event.details).map(
                                      ([key, value]) => (
                                        <div key={key} className="break-
words">
                                          <span className="font-medium
capitalize">
                                            {key.replace(/([A-Z])/g, "
$1").trim()}:
                                          </span>{" "}
                                          <span>{String(value)}</span>
                                        </div>
                                      )
                                    )}
                                  </div>
                                </li>
                              )}
                          </ul>
                        </div>
                      </div>
                    )}
                  </Card>
                );
              })}
            </div>
          );
        };
```

Фрагменти коду кінцевої точки API для керування подорожами (./app/api/events/route.ts)

```
import { NextResponse } from "next/server";
import { prisma } from "@/lib/db";
import { getCurrentUser } from "@/lib/auth";

export async function GET(
  request: Request,
  { params }: { params: Promise<{ id: string }> }
) {
  try {
    const user = await getCurrentUser();
    if (!user) {
      return NextResponse.json({ error: "Unauthorized" }, { status:
401 });
    }
```

```
      const { id: tripId } = await params;

      const trip = await prisma.trip.findUnique({
        where: {
          id: tripId,
          userId: user.id,
        },
      });

      if (!trip) {
        return NextResponse.json({ error: "Trip not found" }, { status:
404 });
      }

      const events = await prisma.event.findMany({
        where: {
          tripId,
        },
        orderBy: {
          startTs: "asc",
        },
      });

      return NextResponse.json({ events });
    } catch (error) {
      console.error("Error fetching events:", error);
      return NextResponse.json(
        { error: "Failed to fetch events" },
        { status: 500 }
      );
    }
  }

  export async function POST(
    request: Request,
    { params }: { params: Promise<{ id: string }> }
  ) {
    try {
      const user = await getCurrentUser();
      if (!user) {
        return NextResponse.json({ error: "Unauthorized" }, { status:
401 });
      }

      const { id: tripId } = await params;

      const trip = await prisma.trip.findUnique({
        where: {
          id: tripId,
          userId: user.id,
        },
      });

      if (!trip) {
        return NextResponse.json({ error: "Trip not found" }, { status:
404 });
      }

      const tripStartDate = new Date(trip.startDate);
      tripStartDate.setHours(0, 0, 0, 0);

      const tripEndDate = new Date(trip.endDate);
      tripEndDate.setHours(23, 59, 59, 999);
```

```
          const isDateWithinTripRange = (date: Date) => {
            const dateToCheck = new Date(date);
            return dateToCheck >= tripStartDate && dateToCheck <=
tripEndDate;
          };

          const body = await request.json();

          const {
            name,
            address,
            phone,
            confirmationNumber,
            price,
            category,
            details,
          } = body;

          let startTs, endTs;

          try {
            if (category === "hotel") {
              const { checkInDate, checkInTime, checkOutDate, checkOutTime
} =
                details;
              startTs = new Date(`${checkInDate}T${checkInTime}`);
              endTs = new Date(`${checkOutDate}T${checkOutTime}`);
            } else if (category === "restaurant") {
              const { reservationDate, reservationTime } = details;
              startTs = new Date(`${reservationDate}T${reservationTime}`);
              endTs = new Date(startTs);
              endTs.setHours(endTs.getHours() + 2);
            } else if (category === "flight") {
              const { departureDate, departureTime, arrivalDate,
arrivalTime } =
                details;
              startTs = new Date(`${departureDate}T${departureTime}`);
              endTs = new Date(`${arrivalDate}T${arrivalTime}`);
            } else {
              const currentDate = new Date().toISOString().split("T")[0];

              const startDate = details.startDate || details.date ||
currentDate;
              const startTime = details.startTime || "12:00";

              startTs = new Date(`${startDate}T${startTime}`);

              if (details.endDate && details.endTime) {
                endTs = new Date(`${details.endDate}T${details.endTime}`);
              } else if (details.endTime) {
                endTs = new Date(`${startDate}T${details.endTime}`);
              } else {
                endTs = new Date(startTs);
                endTs.setHours(endTs.getHours() + 2);
              }
            }

            if (isNaN(startTs.getTime())) {
              throw new Error("Invalid start date/time format");
            }

            if (endTs && isNaN(endTs.getTime())) {
              throw new Error("Invalid end date/time format");
```

```
            }

            let startDateValid = true;
            let endDateValid = true;

            if (startTs) {
              startDateValid = isDateWithinTripRange(startTs);
            }

            if (endTs) {
              endDateValid = isDateWithinTripRange(endTs);
            }

            if (!startDateValid || !endDateValid) {
              return NextResponse.json(
                {
                  error: "Event dates must be within the trip date range",
                  details: {
                    tripStartDate: trip.startDate,
                    tripEndDate: trip.endDate,
                    eventStartDate: startTs?.toISOString(),
                    eventEndDate: endTs?.toISOString(),
                    startDateValid,
                    endDateValid,
                  },
                },
                { status: 400 }
              );
            }

            if (startTs && endTs) {
              // Compare full dates first
              if (startTs >= endTs) {
                return NextResponse.json(
                  {
                    error: "End date and time must be after start date and
time",
                    details: {
                      startDateTime: startTs.toISOString(),
                      endDateTime: endTs.toISOString(),
                    },
                  },
                  { status: 400 }
                );
              }

              // Additional check specifically for same day events
              const startDate = new
Date(startTs).toISOString().split("T")[0];
              const endDate = new Date(endTs).toISOString().split("T")[0];

              if (startDate === endDate) {
                const startTime = startTs.getHours() * 60 +
startTs.getMinutes();
                const endTime = endTs.getHours() * 60 + endTs.getMinutes();

                if (startTime >= endTime) {
                  return NextResponse.json(
                    {
                      error:
                        "When start and end dates are the same, end time
must be later than start time",
                      details: {
                        startDateTime: startTs.toISOString(),
```

```
                    endDateTime: endTs.toISOString(),
                  },
                },
                { status: 400 }
              );
            }
          }
        }
      } catch (error) {
        console.error("Error parsing dates:", error);
        return NextResponse.json(
          { error: "Invalid date format in request" },
          { status: 400 }
        );
      }

      const eventData = {
        tripId,
        category,
        name,
        address,
        phone,
        confirmation: confirmationNumber,
        price: price ? parseFloat(price) : null,
        startTs,
        endTs,
        details: details || {},
      };

      try {
        const event = await prisma.event.create({
          data: eventData,
        });

        return NextResponse.json({ event }, { status: 201 });
      } catch (dbError) {
        console.error("Database error creating event:", dbError);
        return NextResponse.json(
          { error: "Database error: " + (dbError as Error).message },
          { status: 500 }
        );
      }
    } catch (error) {
      console.error("Error creating event:", error);
      return NextResponse.json(
        { error: "Failed to create event" },
        { status: 500 }
      );
    }
  }
```

Фрагменти коду кінцевої точки API для керування подорожами (./app/api/trips/route.ts)

```
import { NextRequest, NextResponse } from "next/server";
import { prisma } from "@/lib/db";
import { getCurrentUser } from "@/lib/auth";
import { tripSchema } from "@/lib/types";

export async function GET() {
```

```
    try {
      const user = await getCurrentUser();
      if (!user) {
        return NextResponse.json({ error: "Unauthorized" }, { status:
401 });
      }

      const trips = await prisma.trip.findMany({
        where: { userId: user.id },
        orderBy: { startDate: "asc" },
      });

      return NextResponse.json({ trips });
    } catch (error) {
      console.error("Error fetching trips:", error);
      return NextResponse.json(
        { error: "Failed to fetch trips" },
        { status: 500 }
      );
    }
  }

  export async function POST(request: NextRequest) {
    try {
      const user = await getCurrentUser();
      if (!user) {
        return NextResponse.json({ error: "Unauthorized" }, { status:
401 });
      }

      const body = await request.json();

      const validationResult = tripSchema.safeParse(body);

      if (!validationResult.success) {
        return NextResponse.json(
          {
            error: "Validation failed",
            details: validationResult.error.format(),
          },
          { status: 400 }
        );
      }

      const { name, destination, startDate, endDate, description } =
        validationResult.data;

      const trip = await prisma.trip.create({
        data: {
          userId: user.id,
          name,
          destination,
          startDate: new Date(startDate),
          endDate: new Date(endDate),
          description,
        },
      });

      return NextResponse.json({ trip }, { status: 201 });
    } catch (error) {
      console.error("Error creating trip:", error);
      return NextResponse.json(
        { error: "Failed to create trip" },
        { status: 500 }
```

```
        );
    }
}
```

Додаток 3

Копія презентації

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

КАФЕДРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ КОМП’ЮТЕРНИХ СИСТЕМ

МОБІЛЬНИЙ ЗАСТОСУНОК ДЛЯ ОРГАНІЗАЦІЇ ПАСАЖИРСЬКИХ ПЕРЕВЕЗЕНЬ

Виконав: студент групи КП -11 Маковецький Павло Віталійович

Керівник: доцент кафедри ПЗКС, к.т.н. Рибачок Наталія Антонівна

Київ — 2025

ПОСТАНОВКА ЗАДАЧІ

Мета проекту: спрощення процесу підготовки до поїздки для мандрівників шляхом надання мобільного застосунку, що забезпечить комплексну підтримку на даному етапі.

Завдання:

1. Проаналізувати наявні рішення, визначити вимоги до застосунку.
2. Обрати засоби розроблення програмного продукту.
3. Спроєктувати архітектуру застосунку та модель даних .
4. Реалізувати програмне забезпечення згідно вимог.
5. Виконати тестування та налагодження застосунку.
6. Порівняти розробку з аналогами, визначити напрями подальшого покращення.

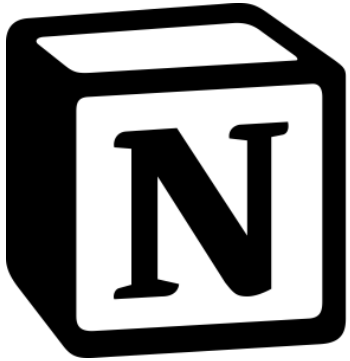
АКТУАЛЬНІСТЬ

1. Ділові та розважальні поїздки – невід'ємна частина життя.
2. Зростання кількості подорожуючих більш ніж вдвічі за останніх два роки.
3. Наявні рішення є вузькоспеціалізованими.
4. Відсутність рішення, що об'єднує всю необхідну функціональність.

ІСНУЮЧІ АНАЛОГИ

TripIt

PackPoint



Notion

ПОРІВНЯННЯ АНАЛОГІВ

Застосунок	PackPoint	TripIt	Notion
Збирання речей	+	–	+
Управління бюджетом	–	–	–
Організація маршруту	–	+	+
Зберігання документів	–	+	–

ФУНКЦІОНАЛЬНІ ВИМОГИ

1. Система повинна підтримувати наступні ролі: Мандрівник, Адміністратор.
2. Мандрівник має можливість керування подорожами (створення, перегляд, видалення).
3. Мандрівник має можливість керувати подіями на кожен день подорожі (створення, перегляд, редагування, видалення).
4. Мандрівник має можливість складати та заповнювати персоналізовані списки речей.
5. Мандрівник має можливість отримати статистику по витратам на основі коштів, затрачених на події та витратах, доданих власноруч (створення, редагування, видалення) у вигляді діаграми.
6. Мандрівник має можливість прикладання файлів до окремих подій та подорожей загалом.
7. Адміністратор має можливість перегляду статистики за Мандрівниками.
8. Адміністратор має можливість керування акаунтами Мандрівників (створення, перегляд, видалення).

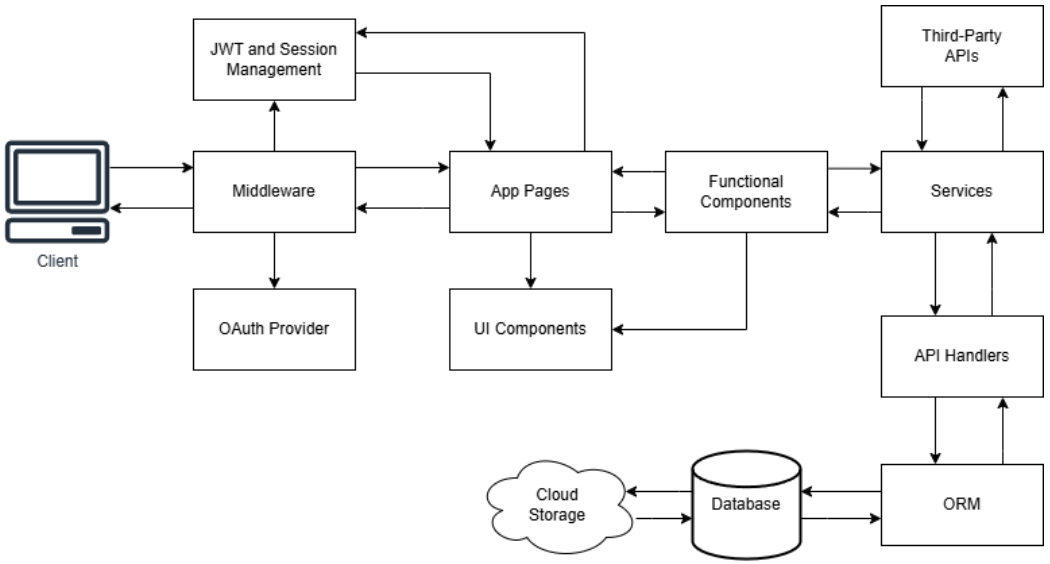
НЕФУНКЦІОНАЛЬНІ ВИМОГИ

1. Система повинна реалізовувати адаптивну розгортку; інтерфейс має підлаштовуватися під екрани різних розмірів та роздільних здатностей.
2. Система повинна коректно працювати на всіх сучасних мобільних браузерах для платформ iOS та Android.
3. Система повинна підтримувати безпарольну автентифікацію на основі протоколу OAuth 2.
4. Система повинна зберігати файли користувачів у захищеному хмарному сховищі.
5. Система повинна реалізовувати надійну валідацію введених даних.

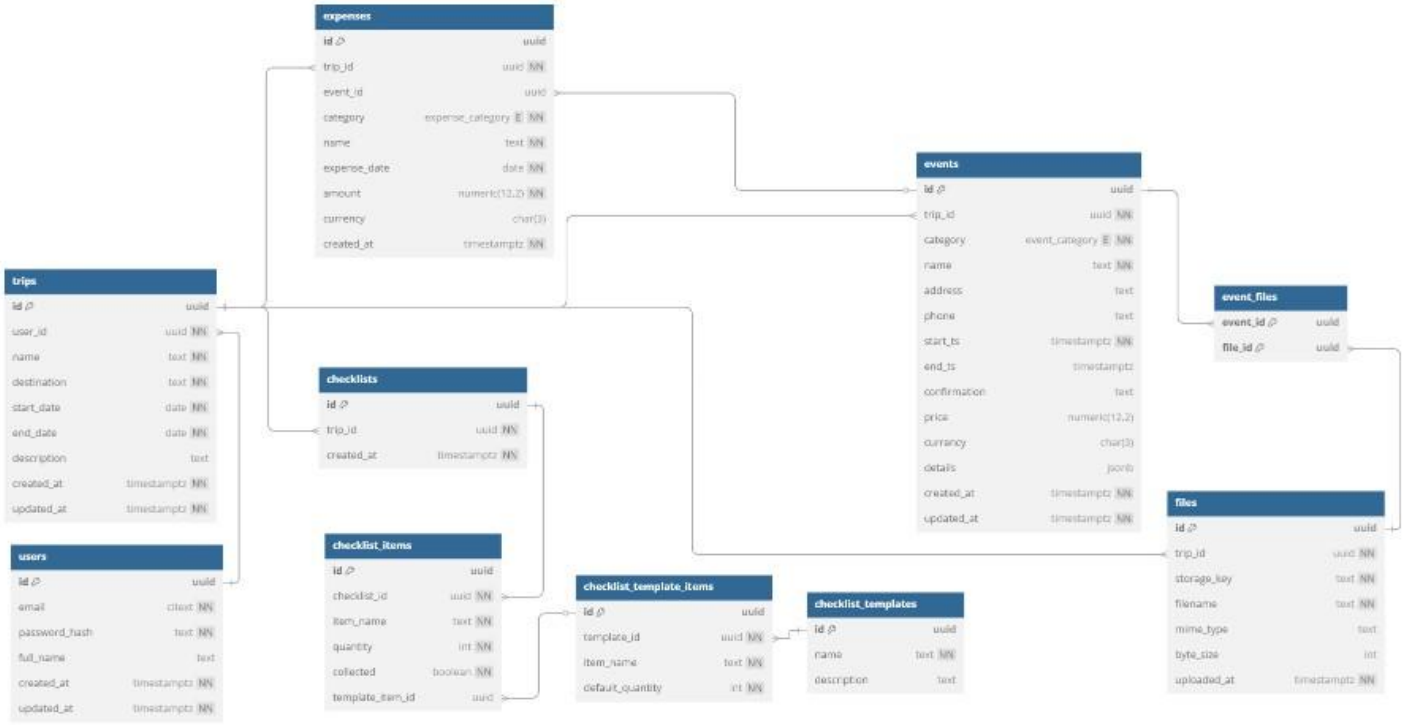
ОБГРУНТУВАННЯ ВИБОРУ ЗАСОБІВ РОЗРОБЛЕННЯ

Тип ПЗ – **мобільний застосунок**
Мова програмування – **TypeScript**,
фреймворк **Next.js.**
База даних – **PostgreSQL**.
Методологія розробки: **водоспадна модель.**

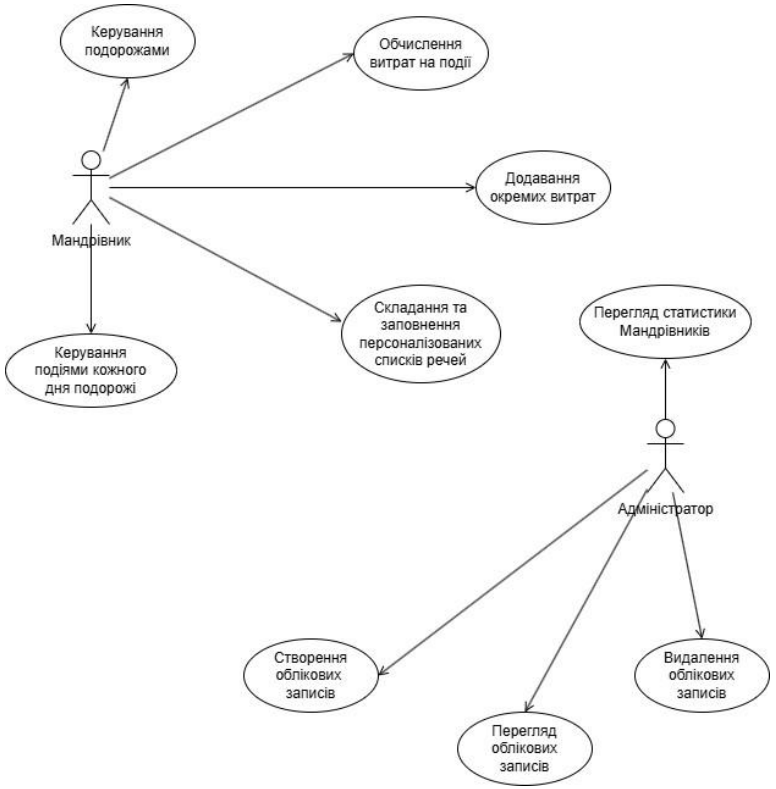
АРХІТЕКТУРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ



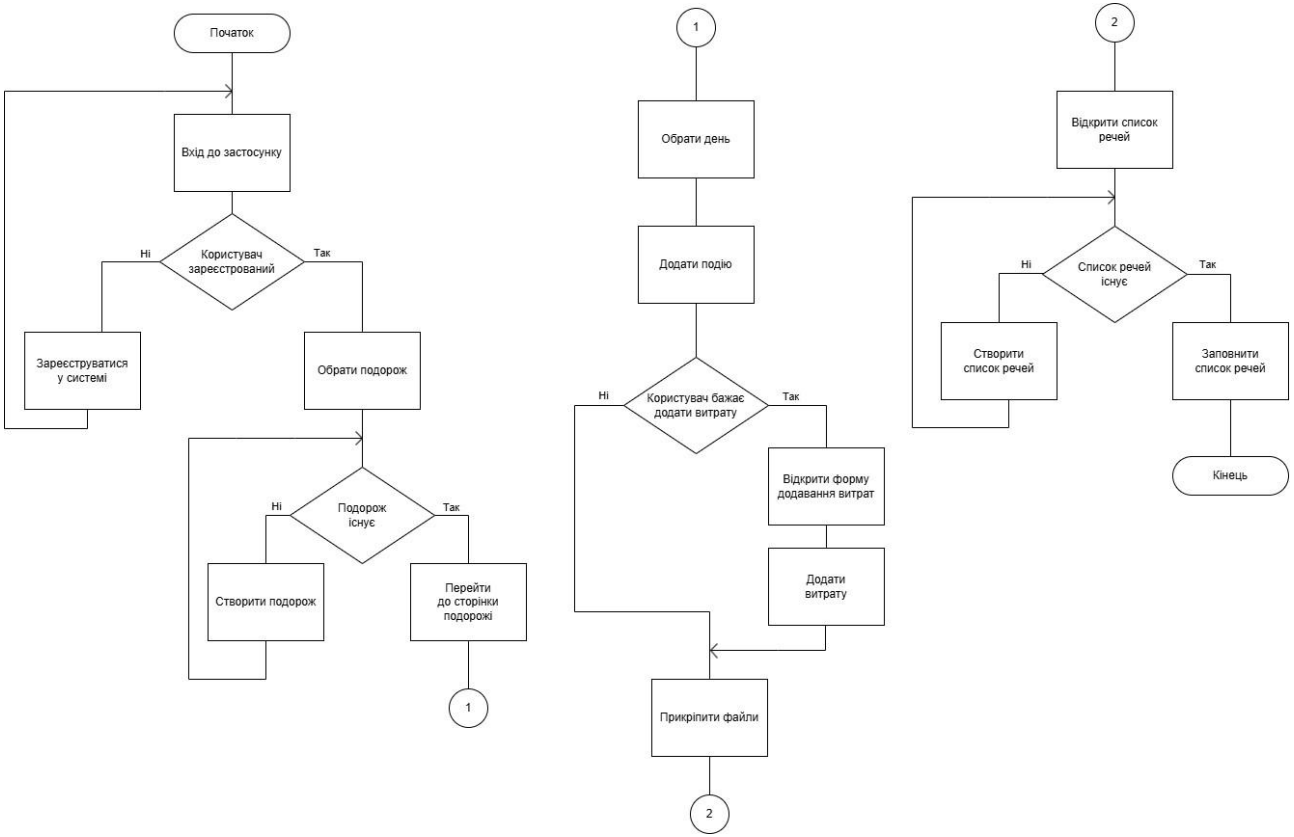
СТРУКТУРА БАЗИ ДАНИХ



ВАРІАНТИ ВИКОРИСТАННЯ



ОСНОВНИЙ СЦЕНАРІЙ



ТЕСТУВАННЯ

Ручне тестування відповідно до принципів димного тестування – методики, яка передбачає виконання мінімального набору тестів, спрямованих на виявлення критичних помилок у базовій функціональності системи. У рамках тестування було перевірено наступні випадки:

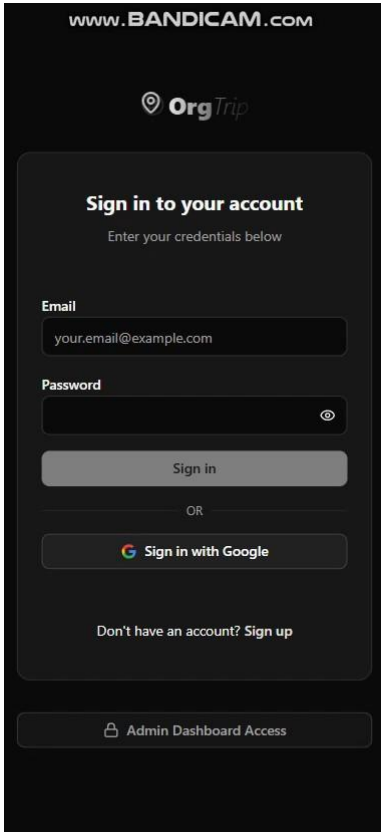
1) реєстрація користувача та вхід до системи;
2) створення подорожі;
3) створення події;
4) додавання власної витрати;
5) прикріплення файлу;
6) створення та заповнення списку речей

Також було забезпечено обробку помилок, наявність підказок в інтерфейсі застосунку, збереження заповнених даних при відкритті форми редагування.

ПОРІВНЯННЯ ПРОГРАМНОГО ПРОДУКТУ З АНАЛОГАМИ

Застосунок	PackPoint	TripIt	Notion	**Створений застосунок**
Збирання речей	+	–	+	+
Управління бюджетом	–	–	–	+
Організація маршруту	–	+	+	+
Зберігання документів	–	+	–	+

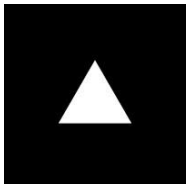
ДЕМОНСТРАЦІЯ РОБОТИ



НАПРЯМКИ ПОДАЛЬШОГО ВДОСКОНАЛЕННЯ

1. Реалізація можливості взаємодії з іншими подорожуючими.
2. Впровадження локалізації.
3. Подальша інтеграція із зовнішніми сервісами.

РОЗГОРТАННЯ



Платформа – **Vercel**.
Постачальник бази даних – **Neon.**
Сховище файлів – **Google Cloud Storage**

17/20

ВИСНОВКИ

1. Проаналізовано існуючі програмні рішення: TripIt, PackPoint, Notion.
2. Визначено вимоги до розроблюваної програмної системи.
3. Визначено основні ролі користувачей у системі та основні сценарії тестування.
4. Обрано програмні засоби для розробки: Next.js, TypeScript, PostgreSQL.
5. Спроєктовано модель даних.
6. Реалізовано багатошарову архітектуру мобільного застосунку.
7. Поставлені до програмної системи вимоги реалізовано в повній мірі.
8. Розроблена програмна система була протестована за методикою димного тестування. Особливу увагу було приділено тестуванню критично важливої функціональності.
9. Запропоновано напрямки подальшого розвитку програмної системи.

УНІКАЛЬНІСТЬ

Звіт подібності

метадані

Назва організації

National Technical University of Ukraine Igor Sikorskyi Kyiv Politech Institute

Заголовок

Бакалавр КП-11 Маковецький Павло Віталійович

Автор — Науковий керівник / Експерт

Маковецький Павло ВіталійовичРибачок Наталія Антонівна

підрозділ

ФПМ, К-ра програмного забезпечення комп'ютерних систем

Обсяг знайдених подібностей

Коефіцієнт подібності визначає, який відсоток тексту по відношенню до загального обсягу тексту було знайдено в різних джерелах. Зверніть увагу, що високі значення коефіцієнта не автоматично означають плагіат. Звіт має аналізувати компетентна / уповноважена особа.

10	11651	93081
Довжина фрази для коефіцієнта подібності 2	Кількість слів	Кількість символів

ДЯКУЮ ЗА УВАГУ!

20/20

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

"ЗАТВЕРДЖЕНО"

Завідувач кафедри

_____ Євгенія СУЛЕМА

"___" _____ 2024 р.

МОБІЛЬНИЙ ЗАСТОСУНОК ДЛЯ ОРГАНІЗАЦІЇ ПАСАЖИРСЬКИХ ПЕРЕВЕЗЕНЬ

Програма та методика тестування

ДП.045440-04-51

"ПОГОДЖЕНО"

Керівник проєкту:

_____ Наталія РИБАЧОК

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Павло МАКОВЕЦЬКИЙ

2024

ЗМІСТ

1. ОБ’ЄКТ ВИПРОБУВАНЬ

Мобільний застосунок для організації пасажирських перевезень, який представляє собою вебзастосунок, розроблений під мобільні пристрої, реалізований за допомогою JavaScript, Next.js, та PostgreSQL.

2. МЕТА ТЕСТУВАННЯ

У процесі тестування має бути перевірено наступне:

1) функціональна працездатність елементів вікон застосунку;

2) наявність доступу до бази даних застосунку та її сутностей;

3) відповідність форматів та протоколів передачі даних;

4) забезпечення належного рівня безпеки даних;

5) зручність роботи із застосунком;

6) відповідність дизайну вимогам Технічного завдання.

3. МЕТОДИ ТЕСТУВАННЯ

Тестування виконується методом «сірого ящику». Перевіряється як код, так і безпосередньо програмний продукт на відповідність функціональним вимогам. Тестування відбувається на рівні «димного тестування».

Використовуються наступні методи:

1) функціональне тестування, зокрема на рівні критичного шляху;

2) тестування продуктивності програмного забезпечення;

3) тестування інтерфейсу.

При виборі сценаріїв тестування використовувався метод «димного» тестування – методики, що передбачає створення мінімального набору тестування на явні помилки, який зазвичай виконується безпосередньо програмістом. Це дозволяє швидко перевірити основну поведінку системи,

оскільки версію програми, що не пройшла ці тести, немає сенсу передавати на подальше тестування.

4. ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Тестування виконується вручну.

Працездатність застосунку перевіряється шляхом:

1) динамічного ручного тестування – введенням граничних та недопустимих значень в поля, які можна редагувати;
2) динамічного ручного тестування на відповідність функціональним вимогам;
3) статичного тестування коду;
4) тестування застосунку в різних веббраузерах на різних пристроях;
5) тестування при великому навантаженні на БД;
6) тестування стабільності роботи при різних умовах;
7) тестування зручності використання;
8) тестування інтерфейсу.

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

"ЗАТВЕРДЖЕНО"

Завідувач кафедри

_____ Євгенія СУЛЕМА

"___" _____ 2025 р.

МОБІЛЬНИЙ ЗАСТОСУНОК ДЛЯ ОРГАНІЗАЦІЇ ПАСАЖИРСЬКИХ ПЕРЕВЕЗЕНЬ

Керівництво користувача

ДП.045440-05-34

"ПОГОДЖЕНО"

Керівник проєкту:

_____ Наталія РИБАЧОК

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Павло МАКОВЕЦЬКИЙ

2025

ЗМІСТ

1. Структура застосунку

Застосунок є мобільним застосунком у вигляді веб-застосунку, адаптованого під мобільні пристрої, що складається із динамічних сторінок із інтерактивними елементами для навігації та взаємодії. Мовою інтерфейсу є англійська.

Вікна системи:

- Вікно входу.
- Вікно реєстрації.
- Вікно подорожей користувача.
- Вкладка хронології.
- Вкладка бюджету.
- Вкладка файлів.
- Вкладка списку речей.

Перемикання між вкладками відбувається без перезавантаження вікна, при цьому застосунок у цей момент динамічно завантажує релевантні дані з БД.

2. Процедури реєстрації та входу

Знайомство мандрівника із застосунком починається з вікна авторизації. Якщо він вже має обліковий запис, достатньо ввести електронну адресу та пароль у відповідні поля й натиснути кнопку входу. Система звіряє введені дані з інформацією в базі даних, і у разі збігу користувач автоматично перенаправляється на сторінку зі своїми подорожами (рис 1). Якщо ж дані виявляться некоректними або відсутніми, з’явиться відповідне повідомлення про помилку, а форма очищується для повторного введення.

Якщо акаунт ще не створено, мандрівник може перейти до реєстрації, натиснувши відповідне посилання. У новому вікні потрібно буде заповнити поля з іменем, електронною адресою, паролем та підтвердженням паролю.

При цьому система перевіряє правильність формату введених даних – наприклад, електронна адреса не повинна містити заборонених символів, а пароль має відповідати вимогам до мінімальної довжини та складності (рис. 2).

Після успішної реєстрації користувач отримає підтвердження та буде автоматично перенаправлений до сторінки входу, де зможе авторизуватися та розпочати планування своєї першої подорожі.

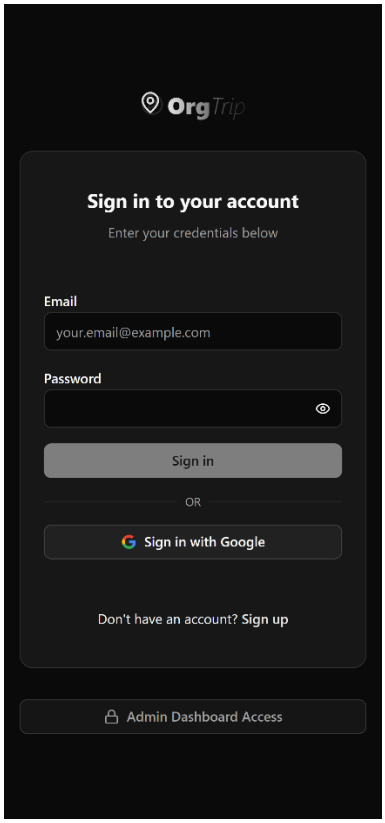


Рис. 1. Вікно входу

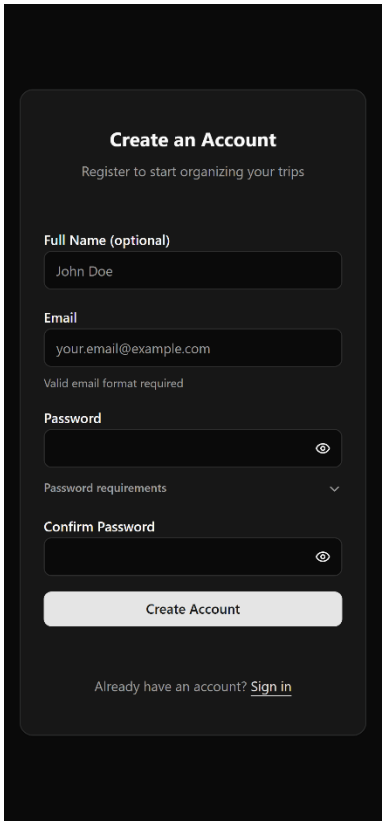


Рис. 2. Вікно реєстрації

3. Подорожі та їх створення

У вікні подорожей користувач побачить перелік своїх мандрівок, поділених на активні та завершені, впорядкованих за датами (рис. 3). Якщо жодної подорожі ще не створено, система запропонує користувачеві розпочати планування, натиснувши кнопку створення нової подорожі. Після натискання відкриється модальне вікно, де потрібно буде ввести основну

інформацію: назву подорожі, її тривалість (із зазначенням дати початку та завершення), пункт призначення, а також за бажанням – короткий опис (рис. 4). Після підтвердження введених даних подорож буде збережена в базі даних, а користувача автоматично перенаправить до вікна з деталями цієї мандрівки.

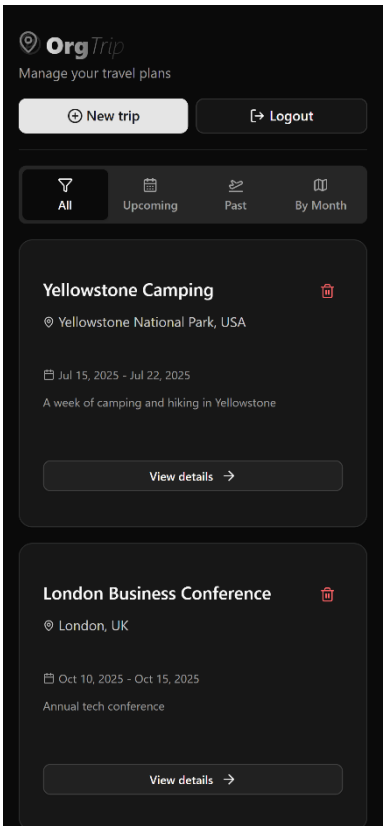


Рис. 3. Вікно подорожей

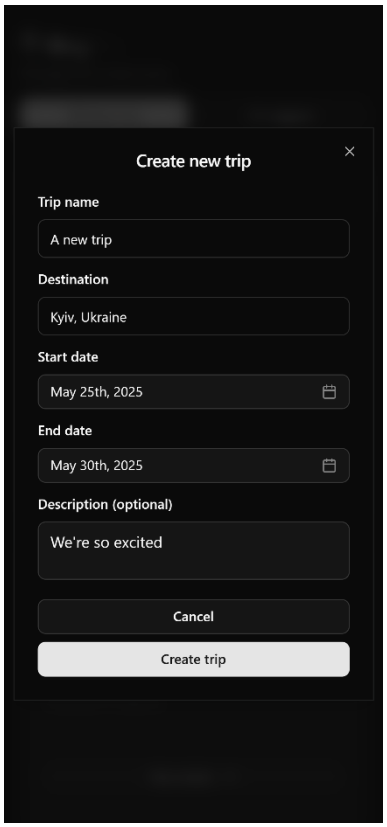


Рис. 4. Форма додавання подорожей

4. Хронологія та додавання подій

На сторінці подорожі користувача зустріне заголовок, який міститиме ключову інформацію: тривалість мандрівки, загальну суму витрат та кількість запланованих подій. Нижче розташовано чотири вкладки, кожна з яких відповідає окремому розділу функціональності застосунку.

У першій вкладці представлено інтерактивна хронологія – календар, що відображає кількість подій на кожен день та їхні типи. Ця інформація буде представлена у вигляді кнопок з відповідними мітками, натискання на

які відкриває перелік активностей, запланованих на обраний день. Список подій з’являється під календарем і містить відомості про тип, деталі та особливості кожної з них (рис. 5).

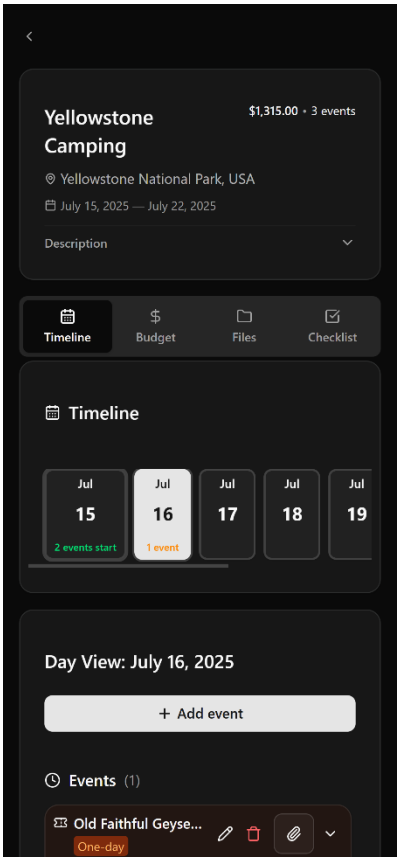


Рис. 5. Вкладка хронології

Користувач має змогу додавати нові події, редагувати існуючі, видаляти непотрібні, а також переглядати розширену інформацію. Крім того, до кожної події можна прикріплювати файли — це дає змогу зручно зберігати документи, необхідні для конкретної активності (рис. 6-7). Завдяки цьому підходу мандрівник отримує централізований доступ до всієї інформації, пов’язаної з подією, що дозволяє краще організовувати час, уникати плутанини та зберігати важливі дані в одному місці.

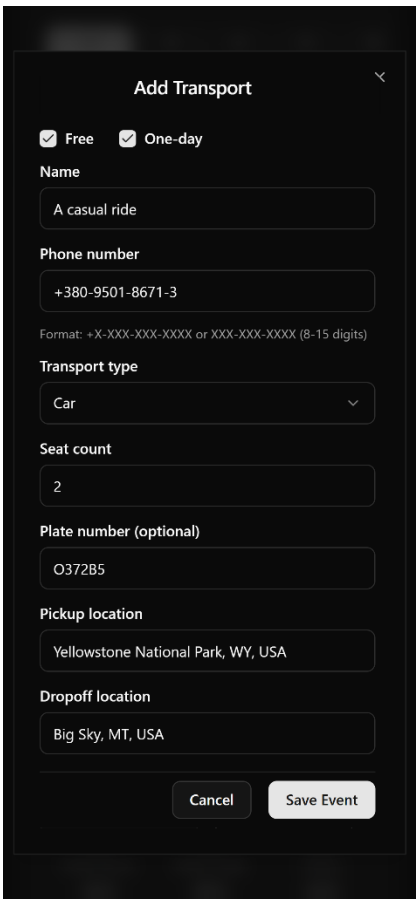


Рис. 6. Форма додавання події

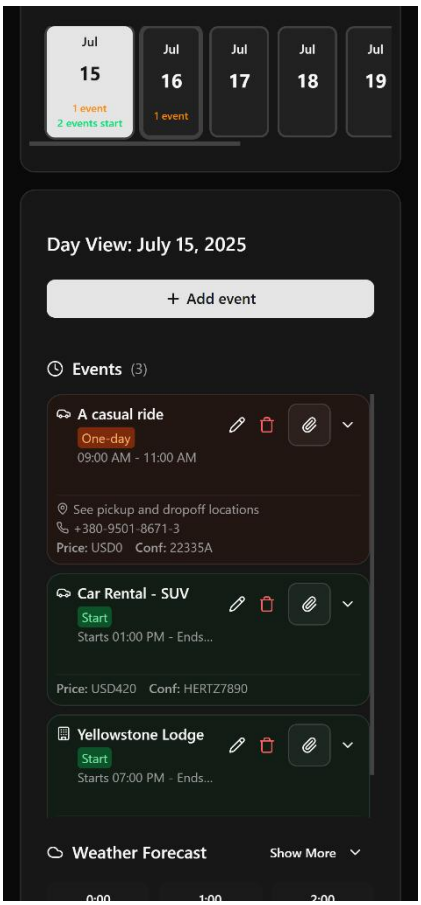


Рис. 7. Результат додавання події

5. Контроль бюджету та додавання витрат

На другій вкладці користувач знайде інструмент для моніторингу бюджету – динамічну інформативну діаграму з легендою. Вона відображатиме щоденні витрати у вигляді кольорових стовпчиків, поділених на сегменти відповідно до типів активностей, що спричинили ці витрати (рис. 8).

Усі зміни, внесені до активностей подорожі, автоматично відображатимуться на графіку. При наведенні курсора на будь-який стовпчик з’являтиметься детальна інформація з точними сумами витрат за кожною категорією.

Крім того, користувач має змогу вручну додавати індивідуальні витрати, які не пов’язані з конкретними подіями. Такі витрати будуть

відображені як окрема категорія на графіку та додатково представлені в окремому підменю для зручного перегляду та контролю (рис. 9).

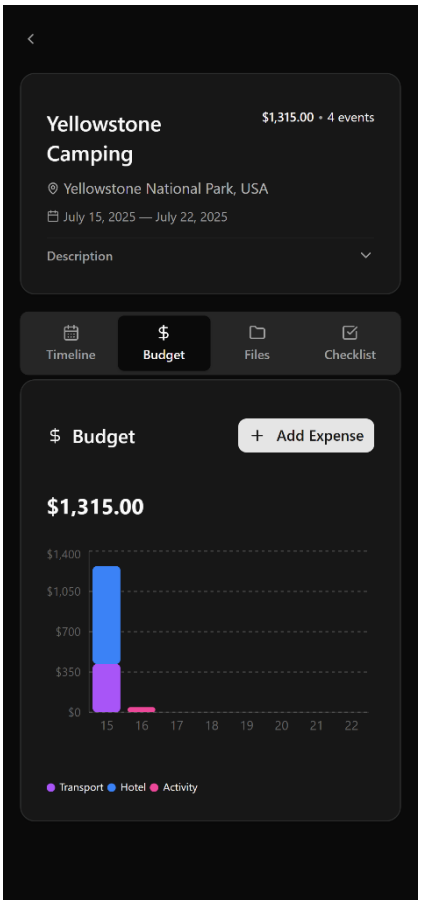


Рис. 8. Вкладка бюджету

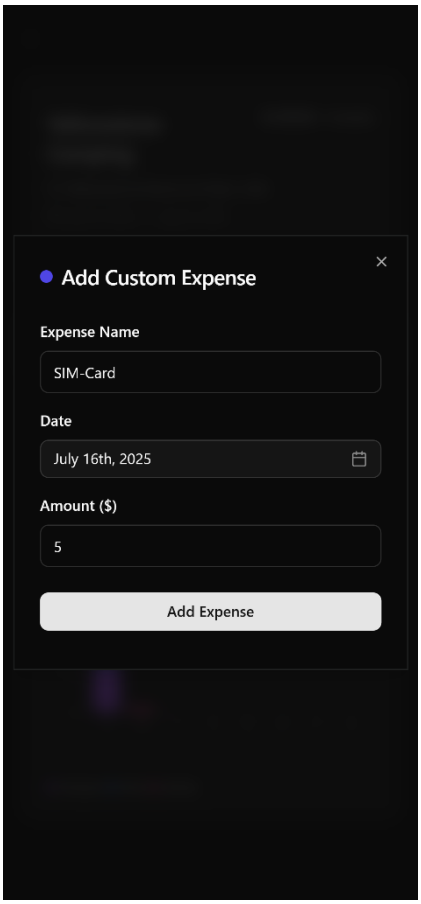


Рис. 9. Додавання власної витрати

6. Сховище файлів подорожі

На третій вкладці користувача зустріне централізоване файлове сховище, призначене для зберігання документів, що стосуються всієї подорожі загалом. Тут мандрівник зможе завантажувати файли, які не прив’язані до конкретних подій – наприклад, сканкопії паспортів, медичні довідки або страхові документи (рис. 10).

Усі завантажені матеріали синхронізуються з зовнішнім хмарним сховищем, що забезпечує їхню безпечність та доступність. Користувач має змогу переглядати ці файли, відкривати їх у новому вікні, перейменовувати, видаляти або завантажувати на власний пристрій.

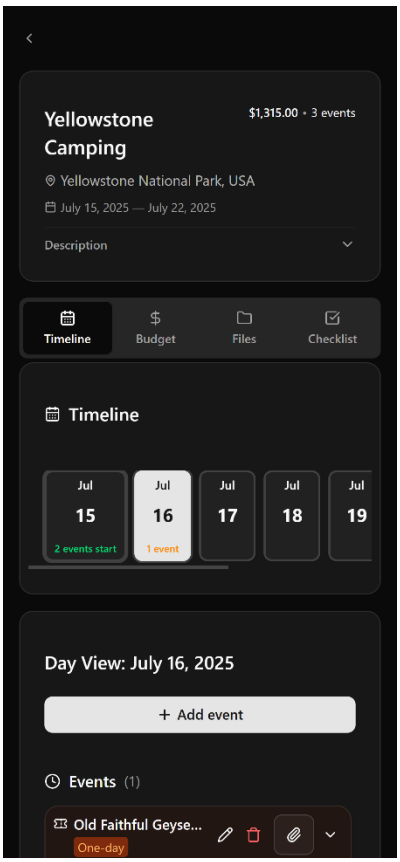


Рис. 10. Вкладка файлів

7. Складання списків речей

Четверта вкладка призначена для створення персоналізованих списків речей, необхідних для подорожі. Якщо такий список ще не створено, користувач отримає запрошення сформувати новий. Після натискання на відповідну кнопку відкриється форма, в якій можна буде обрати один із запропонованих шаблонів, адаптованих до різних типів подорожей – наприклад, гірський похід, ділова поїздка, фототур тощо. Кожен шаблон містить типовий перелік речей із зазначенням рекомендованої кількості для кожної позиції.

Також доступний «Базовий» набір, що включає речі, необхідні для будь-якої мандрівки – такі як одяг, медикаменти тощо. Після вибору шаблону система автоматично згенерує відповідний список.

Користувач побачить перелік речей разом із кількістю, яку потрібно взяти. Біля кожного елемента буде кнопка, що дозволяє позначити річ як зібрану. При цьому у верхній частині екрана поступово заповнюватиметься індикатор прогресу. Якщо користувач змінить рішення або помилково позначить річ, достатньо натиснути на неї ще раз – вона повернеться до статусу «незібраної» (рис. 11).

Коли всі речі буде зібрано, система повідомить користувача про повну готовність до подорожі.

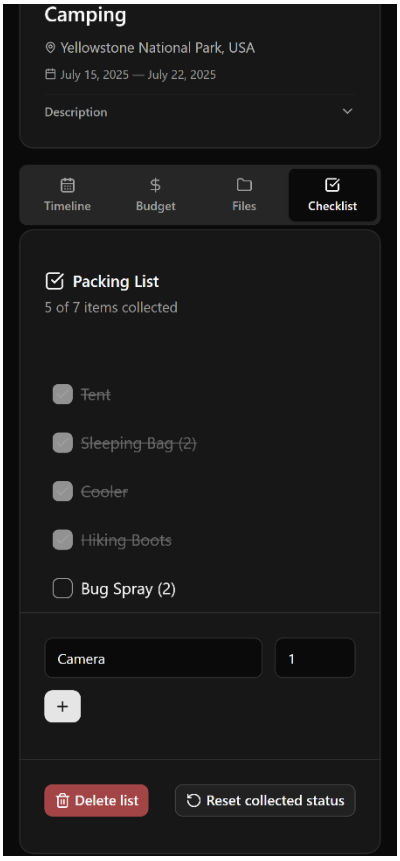


Рис. 11. Вкладка списку речей