

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ
СІКОРСЬКОГО»**

Навчально-науковий інститут атомної та теплової енергетики

Кафедра цифрових технологій в енергетиці

«До захисту допущено»

Завідувач кафедри

_____ Наталія АУШЕВА
“ ” _____ 2026 р.

**Дипломна робота
на здобуття ступеня бакалавр**

За освітньою програмою “Цифрові технології в енергетиці”

Спеціальності 122 “Комп’ютерні науки”

на тему: “Серверна частина системи планування групових подорожей”

Виконав: студент 4 курсу, групи ТР-25

Кондратенко Сергій Сергійович

(прізвище, ім’я, по батькові)

_____ (підпис)

Керівник професор каф. ЦТЕ, д. т. н., Сергій ОТРОХ

(посада, науковий ступінь, вчене звання, ім’я, ПРІЗВИЩЕ)

_____ (підпис)

Рецензент професор каф. ІПЗЕ, д. т. н., Олексій НЕДАШКІВСЬКИЙ

(посада, науковий ступінь, вчене звання, ім’я, ПРІЗВИЩЕ)

_____ (підпис)

Н. контроль асистент каф. ЦТЕ, Костянтин ЗДОР

(посада, ім’я, ПРІЗВИЩЕ)

_____ (підпис)

Засвідчую, що у цій дипломній роботі немає
запозичень з праць інших авторів без
відповідних посилань.

Студент _____

Київ – 2026

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”**

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ АТОМНОЇ ТА ТЕПЛОВОЇ ЕНЕРГЕТИКИ

Кафедра ЦИФРОВИХ ТЕХНОЛОГІЙ В ЕНЕРГЕТИЦІ

Рівень вищої освіти – перший (бакалаврський)

Спеціальність 122 “Комп’ютерні науки”

Освітньо-професійна програма “Цифрові технології в енергетиці”

ЗАТВЕРДЖУЮ

Завідувач кафедри ЦТЕ

Наталія АУШЕВА

(підпис)

“ ” 2026 р.

ЗАВДАННЯ

на дипломну роботу студенту

Кондратенку Сергію Сергійовичу

(прізвище, ім’я, по батькові)

1. Тема роботи “Серверна частина системи планування групових подорожей”

Науковий керівник Отрох Сергій Іванович, д. т. н., професор

(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від “28” травня 2026 року № 1992-с

2. Термін подання студентом роботи 04.06.2026

3. Вихідні дані до роботи мова програмування TypeScript, середовище виконання Node.js, фреймворк NestJS, реляційна СУБД PostgreSQL, ORM-система Prisma, платіжний шлюз Stripe API, бібліотека Socket.IO (для WebSocket), середовище розробки WebStorm, засіб для тестування API Postman

4. Перелік питань, які потрібно розробити:

1) провести аналіз наявних аналогів систем для планування подорожей, управління спільними витратами, комунікації учасників і фінансової взаємодії в групових поїздках;

2) визначити основні функції серверної частини системи планування групових подорожей, зокрема автентифікацію користувачів, керування профілями, подорожами, заявками, запрошеннями, ролями учасників, сповіщеннями та фінансовими операціями;

3) обґрунтувати вибір архітектури, технологій, інструментів і алгоритмів для реалізації серверної частини програмної системи;

4) спроектувати архітектуру серверної частини, REST API та структуру реляційної бази даних для зберігання користувачів, подорожей, учасників, заявок, платежів, звітності, повідомлень, сповіщень і медіафайлів;

5) реалізувати модулі серверної частини: автентифікації та авторизації, управління користувачами, створення й редагування подорожей, обробки заявок і запрошень, фінансового обліку, завантаження файлів, сповіщень і фонових задач;

6) реалізувати механізми фінансової взаємодії між організатором і учасниками подорожі, зокрема гарантійні внески, цільові витрати, завантаження чеків, голосування учасників та облік станів оплат;

7) провести тестування REST API за допомогою Postman, перевірити коректність роботи основних endpoint-ів і виконати інтеграційну перевірку серверної частини разом із клієнтською частиною системи;

8) представити процес роботи серверної частини системи на прикладі основних сценаріїв: реєстрація та вхід користувача, створення подорожі, подання і прийняття заявки, обробка платежу, створення сповіщення та збереження медіафайлів.

5. Орієнтований перелік ілюстративного матеріалу схема загальної архітектури програмної системи GTPLANNER; схема клієнт-серверної взаємодії між веб-інтерфейсом, мобільним застосунком, backend API, базою даних і зовнішніми сервісами; схема модульної структури серверної частини системи; ER-діаграма бази даних системи планування групових подорожей; діаграма прецедентів із

ролями користувачів системи; схема обробки REST API-запиту на серверній частині; схема фінансового workflow: гарантійний внесок, оплата, цільова витрата, чек, голосування учасників, підтвердження або відхилення; діаграма послідовності для сценарію подання та прийняття заявки на участь у подорожі; діаграма послідовності для сценарію обробки платежу та створення сповіщення; фрагменти структури бази даних / Prisma-моделей; скріншоти тестування REST API у Postman; скріншоти роботи основних сценаріїв системи, що демонструють результат взаємодії клієнтської та серверної частин.

6. Дата видачі завдання 15.09.2026

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1.	Вибір теми роботи	15.09.2025	виконано
2.	Аналіз методів та засобів розв’язання задачі	20.04.2026-22.04.2026	виконано
3.	Розробка архітектури та загальної структури системи	23.04.2026-24.04.2026	виконано
4.	Проектування бази даних, REST API та серверних модулів	25.04.2026-26.04.2026	виконано
5.	Розробка окремих підсистем: автентифікація, користувачі, подорожі, заявки, ролі	27.04.2026-03.05.2026	виконано
6.	Реалізація фінансового обліку, завантаження чеків, голосування, сповіщень, медіафайлів і Cron-задач	04.05.2026-10.05.2026	виконано
7.	Тестування REST API, налагодження серверної частини та інтеграційна перевірка з клієнтською частиною	11.05.2026-17.05.2026	виконано
8.	Оформлення пояснювальної записки	18.05.2026-31.05.2026	виконано
9.	Захист програмного продукту	13.05.2026	виконано
10.	Передзахист дипломної роботи	03.06.2026	виконано
11.	Захист дипломної роботи	16.06.2026	виконано

Студент

(підпис)

Кондратенко С. С.
(прізвище та ініціали)

Керівник

(підпис)

Отрох С. І.
(прізвище та ініціали)

АНОТАЦІЯ

Дипломна робота виконана на 62 сторінках, містить 13 ілюстрацій, одну таблицю, два додатки, 30 джерел в переліку посилань.

Мета роботи: розробка надійної та масштабованої серверної частини програмної системи GTPLANNER для автоматизації бізнес-процесів планування, фінансового обліку, безпеки платежів та координації учасників групових подорожей.

Використані методи та засоби: клієнт-серверна архітектура, проєктування реляційних баз даних, розробка REST API, JWT-авторизація. Програмні засоби: мова програмування TypeScript, фреймворк NestJS, СУБД PostgreSQL, інструмент об'єктно-реляційного відображення Prisma ORM, платіжний шлюз Stripe API, технологія WebSocket (Socket.IO).

Основний зміст дипломної роботи: спроектовано комплексну структуру бази даних та архітектуру серверної частини. Реалізовано модулі автентифікації, управління користувачами, ролями (організатор, учасник) та життєвим циклом заявок на участь. Розроблено фінансовий модуль для обробки гарантійних внесків, платежів та звітності через інтеграцію зі Stripe. Налаштовано механізми обміну повідомленнями в реальному часі (чат), систему сповіщень, завантаження медіафайлів для альбому спогадів та автоматизовані фонові задачі (Cron).

Рекомендації щодо використання та результати впровадження: розроблена серверна частина може бути використана як незалежне ядро для платформ організації туризму, туристичних агентств або самостійних груп мандрівників. Результати роботи пройшли апробацію під час переддипломної практики в ТОВ "ПРІОКОМ" та успішно протестовані під час інтеграції з клієнтським веб- і мобільним застосунком.

Ключові слова: планування подорожей, серверна частина, база даних, фінансовий облік, інтеграція, маршрутизація, сповіщення, веб-застосунок.

ABSTRACT

The thesis is executed on 62 pages, contains 13 illustrations, one table, two appendices, 30 references.

Purpose of the work: development of a reliable and scalable backend for the GTPLANNER software system to automate the business processes of planning, financial accounting, payment security, and coordination of group travel participants.

Methods and tools used: client-server architecture, relational database design, REST API development, JWT authorization. Software tools: TypeScript programming language, NestJS framework, PostgreSQL database management system, Prisma ORM tool, Stripe API payment gateway, WebSocket technology (Socket.IO).

Main content of the thesis: a comprehensive database structure and backend architecture were designed. Modules for authentication, managing users, roles (organizer, participant), and the life cycle of join requests were implemented. A financial module was developed to process guarantee deposits, payments, and reporting via integration with Stripe. Real-time messaging mechanisms (chat), a notification system, media file uploading for the memory album, and automated background tasks (Cron) were set up.

Recommendations for use and implementation results: the developed backend can be used as an independent core for tourism organization platforms, travel agencies, or independent groups of travelers. The results of the work were tested during pre-diploma practice at "PRIOKOM" LLC and successfully tested during integration with the client web and mobile application.

Keywords: travel planning, backend, database, financial accounting, integration, routing, notifications, web application.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ	10
ВСТУП.....	11
1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ, ПОСТАНОВКА ЗАДАЧІ ТА ОГЛЯД МЕТОДІВ ЇЇ РОЗВ'ЯЗАННЯ.....	15
1.1 Особливості та проблеми координації бізнес-процесів у групових поїздках	15
1.2 Огляд та порівняльний аналіз існуючих програмних сервісів-аналогів	17
1.3 Обґрунтування актуальності та декомпозиція вимог до серверної логіки	20
2 ПРОЄКТУВАННЯ АРХІТЕКТУРИ ТА СТРУКТУРИ ДАНИХ СИСТЕМИ	23
2.1 Обґрунтування вибору клієнт-серверної архітектури та модульного підходу.....	24
2.1.1 Організація серверного ядра та життєвий цикл запиту	26
2.1.2 Проєктування системи прецедентів та рольової ієрархії.....	28
2.2 Проєктування реляційної бази даних та розробка ER-моделі.....	31
2.2.1. Нормалізація та логічна структура даних	31
2.2.2 Забезпечення цілісності даних через Prisma ORM	33
2.3 Проєктування фінансового алгоритму та життєвого циклу бізнес- процесів	35
2.3.1. Математична модель фінансової взаємодії	36
2.3.2 Діаграма послідовності для сценарію подання та прийняття заявки	38
2.3.3. Діаграма послідовності обробки платежу та інтеграція зі Stripe	39
2.3.4 Процес цільового витрачання коштів та система голосування...	41
3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ СИСТЕМИ.....	43
3.1 Розробка серверних модулів та інтеграція бізнес-логіки	44
3.2 Програмна реалізація фінансового модуля та підсистеми реального часу	46
3.3 Тестування REST API та перевірка сценаріїв роботи	48
4 РОЗГОРТАННЯ ТА ДЕМОНСТРАЦІЯ РОБОТИ ПРОГРАМНОЇ СИСТЕМИ ..	53

4.1 Інсталяція програмного забезпечення та вимоги до обчислювальної техніки	54
4.2 Демонстрація функціоналу та представлення сценаріїв роботи програмної системи за тестами.....	55
4.3 Постановка та результати обчислювальних експериментів. Оцінка ефективності	60
ВИСНОВКИ	62
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	63
ДОДАТОК А РЕАЛІЗАЦІЯ ЛОГІКИ ОБРОБКИ ПЛАТЕЖІВ, УПРАВЛІННЯ МАРШРУТАМИ ТА ВЗАЄМОДІЇ З БАЗОЮ ДАНИХ СЕРВЕРНОЇ ЧАСТИНИ СИСТЕМИ GTPLANNER.....	66
ДОДАТОК Б МАТЕРІАЛИ АПРОБАЦІЇ РЕЗУЛЬТАТІВ ДИПЛОМНОЇ РОБОТИ	72

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ

БД — база даних.

СУБД — система управління базами даних.

API (Application Programming Interface) — прикладний програмний інтерфейс, набір інструментів для взаємодії між різними програмними системами.

CRUD (Create, Read, Update, Delete) — чотири базові функції управління даними: створення, читання, оновлення та видалення записів у базі даних.

Cron — система для автоматичного запуску програм і скриптів на сервері у заданий час або з певною періодичністю.

DTO (Data Transfer Object) — патерн проєктування, що використовується для інкапсуляції даних та їх відправлення від одного компонента системи до іншого.

HTTP (HyperText Transfer Protocol) — протокол передачі даних, що використовується для обміну повідомленнями між клієнтом і сервером.

JSON (JavaScript Object Notation) — текстовий формат обміну даними, який легко читається машинами та людьми.

JWT (JSON Web Token) — відкритий стандарт, який визначає компактний і автономний спосіб безпечної передачі інформації між сторонами у вигляді зашифрованого JSON-об'єкта.

ORM (Object-Relational Mapping) — об'єктно-реляційне відображення, технологія, яка пов'язує таблиці реляційних баз даних з концепціями об'єктно-орієнтованих мов програмування.

REST (Representational State Transfer) — архітектурний стиль взаємодії компонентів розподіленого застосунку в мережі.

Webhook — механізм сповіщення сторонніх сервісів або застосунків про події, що відбуваються в системі, за допомогою автоматичних HTTP-запитів.

WebSocket — протокол зв'язку поверх TCP-з'єднання, призначений для обміну повідомленнями між клієнтом і сервером у режимі реального часу.

ВСТУП

Актуальність теми. Сучасний етап розвитку глобальної індустрії туризму та сфери інформаційних технологій характеризується стрімкою цифровізацією та зростанням попиту на спеціалізовані програмні рішення для організації спільного дозвілля. Планування та координація групових подорожей є складним багатофакторним процесом, який охоплює узгодження маршрутів, управління заявками та запрошеннями учасників, оперативне сповіщення про зміни, а також ведення спільного фінансового обліку.

На сьогодні більшість груп мандрівників змушені використовувати фрагментований набір інструментів: загальні месенджери для обговорень, окремі картографічні сервіси для побудови маршрутів та сторонні табличні редактори або банківські застосунки для фіксації витрат. Такий підхід призводить до втрати важливих домовленостей у загальному потоці повідомлень, неузгодженості даних та виникнення фінансових ризиків чи суперечок через складність верифікації витрат і гарантійних внесків.

Оскільки сучасні користувачі взаємодіють із системами через різноманітні клієнтські інтерфейси (вебплатформи та мобільні застосунки), виникає критична потреба у створенні централізованого, безпечного та масштабованого інженерного ядра. Клієнтські застосунки не повинні і не можуть через міркування безпеки та продуктивності взаємодіяти з базою даних напряму або самотійно обробляти чутливу фінансову логіку. Саме тому розробка надійної серверної частини (backend), яка централізовано ізолює бізнес-правила, забезпечує цілісність даних, керує ролями користувачів, обробляє асинхронні платіжні події та підтримує комунікацію в реальному часі, є надзвичайно актуальним науково-практичним завданням [1, 2].

Розробка серверної частини є ключовою складовою комплексної програмної системи GTPLANNER, що покликана об'єднати всі етапи супроводу групових подорожей в єдиному захищеному середовищі.

Мета роботи — розробка надійної, масштабованої та безпечної серверної частини системи GTPLANNER для забезпечення цілісності реляційних даних, автоматизації фінансового обліку, координації взаємодії учасників у реальному часі та розмежування прав доступу користувачів.

Для досягнення поставленої мети необхідно вирішити такі завдання:

- опрацювати теоретичні засади побудови RESTful API, сучасні патерни проектування серверної архітектури та принципи моделювання реляційних баз даних;
- спроектувати комплексну логічну структуру бази даних та модульну архітектуру backend-застосунку;
- реалізувати систему автентифікації користувачів із захистом приватних маршрутів за допомогою механізму JWT та безпечного хешування паролів;
- розробити логіку управління подорожами (CRUD-операції), життєвим циклом заявок на участь та динамічним розподілом рольових моделей користувачів (організатор, учасник, гість);
- створити фінансовий модуль для забезпечення обліку цільових витрат, завантаження чеків, проведення внутрішнього голосування та інтеграції платіжного інструментарію Stripe API;
- реалізувати підсистеми обміну повідомленнями в реальному часі на базі протоколу WebSocket, черги сповіщень та автоматизовані фонові задачі за розкладом (Cron);
- здійснити комплексне тестування розроблених REST API маршрутів у середовищі Postman та провести інтеграційну перевірку взаємодії із клієнтськими підсистемами.

Об’єкт розробки — процес автоматизації планування, фінансового супроводу, адміністрування та координації групових подорожей.

Предмет розробки — архітектурні рішення, моделі даних, алгоритми обробки фінансових подій та програмні засоби реалізації серверної частини системи планування групових подорожей.

Методи розробки. Для реалізації програмного комплексу використано архітектурний стиль REST для стандартизованої взаємодії компонентів. Розробку бізнес-логіки виконано на базі компонентно-модульного фреймворку NestJS із застосуванням мови програмування TypeScript. Моделювання та взаємодію з даними реалізовано за допомогою об'єктно-реляційного відображення Prisma ORM та реляційної СУБД PostgreSQL. Обробку платежів побудовано на базі Stripe API та технології асинхронних патернів Webhooks. Обмін даними в реальному часі реалізовано за допомогою протоколу WebSocket (бібліотека Socket.IO).

Практичне значення отриманих результатів. Розроблено та повністю протестовано працездатне інженерне ядро серверної частини системи GTPLANNER. Створена модульна архітектура дозволяє використовувати серверний застосунок як незалежну scalable-платформу, що легко піддається розширенню функціоналу та може виступати базовою інфраструктурою для туристичних компаній, комерційних організаторів турів або некомерційних спільнот мандрівників. Розроблений API успішно інтегровано з клієнтськими веб-та мобільними підсистемами комплексного проекту.

Апробація результатів роботи. Основні технічні рішення та алгоритми серверної частини пройшли практичну перевірку та тестування під час виконання завдань переддипломної практики на базі ІТ-підприємства ТОВ "ПРІОКОМ".

Наукові положення та результати комплексної розробки представлені у формі доповіді та тез на VIII Міжнародній науково-практичній конференції «Information Technology for Education, Science, and Technics» (ITEST-2026, м. Черкаси) у межах спільної публікації «Проектування клієнт-серверної системи для планування групових подорожей» (див. Додаток Б). Повнотекстову англomовну статтю «Concept and Development of an Integrated Client-server System for Group Travel Planning with Web and Mobile Interfaces and Coordinated Financial Workflows» прийнято до подальшого розгляду для публікації у науковій серії Springer Lecture Notes on Data Engineering and Communications Technologies (LNDECT), що індексується міжнародною наукометричною базою Scopus.

Структура та обсяг роботи. Дипломна робота складається з переліку скорочень, вступу, чотирьох розділів, висновків, списку використаних джерел із 30 найменувань та двох додатків. Загальний обсяг текстової частини становить 62 сторінки комп'ютерного тексту. Робота містить 13 рисунків та одну таблицю.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ, ПОСТАНОВКА ЗАДАЧІ ТА ОГЛЯД МЕТОДІВ ЇЇ РОЗВ'ЯЗАННЯ

Метою першого розділу є всебічний аналіз предметної області, вивчення існуючих підходів до автоматизації туристичних послуг та формалізація технічних вимог до розробки серверної частини системи планування групових подорожей GTPLANNER.

У розділі детально розглядаються ключові аспекти та проблеми координації бізнес-процесів, що виникають під час організації спільних поїздок, зокрема питання фінансового обліку, розподілу ролей та управління комунікаціями (підрозділ 1.1). Здійснюється огляд та порівняльний аналіз наявних на ринку програмних сервісів-аналогів для виявлення їхніх функціональних обмежень, архітектурних недоліків та обґрунтування необхідності створення нового спеціалізованого рішення (підрозділ 1.2). На основі результатів проведеного аналізу формулюється чітка постановка прикладної задачі, проводиться декомпозиція функціональних та нефункціональних вимог до серверного ядра, а також визначаються базові архітектурні обмеження майбутньої програмної системи (підрозділ 1.3).

1.1 Особливості та проблеми координації бізнес-процесів у групових поїздках

Сучасний процес організації групових подорожей являє собою складний багаторівневий комплекс бізнес-процесів, який вимагає постійної синхронізації дій між усіма учасниками. На відміну від індивідуального туризму, де планування здійснюється однією особою, груповий формат передбачає наявність ініціатора (організатора) та групи учасників, кожен з яких має власні потреби, фінансові можливості та рівень залученості.

Життєвий цикл групової подорожі можна концептуально поділити на кілька ключових етапів: ініціація та планування маршруту, збір групи та обробка заявок, фінансове планування та збір коштів, безпосередньо подорож, а також етап завершення (підбиття підсумків, розподіл залишкових витрат, обмін медіафайлами). На кожному з цих етапів виникають специфічні організаційні та технічні проблеми, які суттєво ускладнюють координацію.

Проблема децентралізації комунікацій та втрати даних. На сьогодні більшість груп використовують для узгодження деталей загальні месенджери (Telegram, Viber, WhatsApp). Проте архітектура стандартних месенджерів орієнтована на лінійний потік повідомлень. У результаті важлива організаційна інформація (точний час збору, зміни в розкладі, реквізити для оплати) швидко губиться в масиві неструктурованого тексту. Це призводить до необхідності багаторазового дублювання інформації організатором, що знижує ефективність комунікації та підвищує ризик того, що хтось із учасників пропустить критично важливе сповіщення.

Проблема фінансового обліку та непрозорості витрат. Одним із найбільш вразливих аспектів спільного відпочинку є фінансовий облік. Процес передбачає збір гарантійних внесків, оплату бронювань житла та транспорту, а також розподіл поточних витрат (наприклад, спільні чеки в ресторанах або купівля квитків на екскурсії). Здійснення цих операцій через звичайні банківські перекази без спеціалізованої системи обліку часто призводить до суперечок, оскільки важко відстежити, хто з учасників здав кошти, яку суму було витрачено на конкретну потребу і як справедливо розділити залишок. Відсутність єдиного механізму завантаження та верифікації чеків створює непрозорість фінансових потоків.

Управління ролями та правами доступу. У груповій поїздки існує чітка ієрархія: організатор несе відповідальність за маршрут та фінанси, тому він повинен мати ексклюзивні права на редагування програми, підтвердження або відхилення заявок нових учасників, а також ініціацію фінансових зборів. Звичайні учасники повинні мати доступ до перегляду інформації, участі в обговореннях та завантаження власних матеріалів. Відсутність спеціалізованого програмного

середовища з жорстким розмежуванням прав (Role-Based Access Control) робить неможливим безпечно спільне управління подорожжю [3].

Необхідність автоматизації рутинних процесів. Організатор подорожі змушений тримати в пам'яті велику кількість дедлайнів: терміни внесення передоплати, дедлайни подачі документів тощо. Ручний контроль цих подій є неефективним. Існує об'єктивна потреба у впровадженні автоматизованих фонових задач (наприклад, на базі планувальника Cron), які б самостійно перевіряли статуси оплат і автоматично генерували сповіщення для боржників або нагадування про наближення дати виїзду.

Аналіз вищезазначених проблем демонструє, що координація групових поїздок вимагає переходу від використання розрізнених інструментів до єдиної спеціалізованої платформи. Така платформа повинна базуватися на потужній серверній частині, яка здатна забезпечити транзакційну цілісність бази даних, взяти на себе відповідальність за обробку платежів, управління ролями, автоматизацію сповіщень та підтримку узгодженого стану даних для всіх клієнтських пристроїв у режимі реального часу [4].

1.2 Огляд та порівняльний аналіз існуючих програмних сервісів-аналогів

Щоб обґрунтувати доцільність створення платформи GTPLANNER, варто спочатку проаналізувати вже існуючі цифрові інструменти. Оскільки продукт розробляється відразу для вебу та мобільних пристроїв, до порівняння логічно залучити кілька різних класів програм. Мова йде про туристичні планувальники, вузькопрофільні фінансові утиліти та звичні комбінації базових додатків.

До першої групи можна віднести системи Wanderlog та TripIt. Вони працюють як у браузері, так і на смартфонах. Серед їхніх сильних сторін — інтерактивні карти та вміння автоматично підтягувати квитки з електронної пошти користувача. Формат мобільного додатка також додає зручності безпосередньо в

дорозі. Проте є один суттєвий нюанс: архітектура таких платформ орієнтована переважно на одного мандрівника. У випадку групових поїздок решта людей зазвичай отримує лише права на читання. До того ж тут дуже слабо реалізована фінансова частина. Спільного бюджету немає взагалі, або ж він нагадує простий текстовий блокнот. Їхні сервери технічно не адаптовані під безпечний збір коштів через платіжні шлюзи чи повноцінний живий чат.

Зовсім інший підхід пропонують фінансові трекери на кшталт Splitwise або Tricount. Вони створені суто для того, щоб рахувати, хто і скільки кому винен. Ці програми вміють працювати з різними валютами та генерувати миттєві сповіщення. Але їхня головна вада полягає у відсутності будь-якого туристичного контексту. Тут не можна побудувати маршрут, прив'язати витрату до локації на карті чи зберегти пам'ятне фото. Грошові питання існують абсолютно відірвано від реальних подій. Як наслідок, учасникам поїздки доводиться постійно перемикатися між месенджером і фінансовим трекером, дублюючи цифри вручну.

На практиці ж найчастіше туристи вдаються до "народного" методу: створюють групу в Telegram, скидають точки з Google Maps і ведуть облік витрат у таблицях. З одного боку, це безкоштовно і таку зв'язку можна налаштувати під будь-які потреби. З іншого боку, рівень фрагментації даних тут максимальний. Логістика, гроші та комунікація розірвані між трьома різними екосистемами. Жодної мови про серверну автоматизацію навіть не йде. Ніхто не нагадає про дедлайн оплати, система не перевірить коректність введеної суми, і тим більше неможливо організувати захищений платіж через банківський API просто з чату.

Щоб структурувати ці відмінності, архітектурні та функціональні можливості перелічених сервісів зведено у таблицю 1.1. При цьому основна увага приділяється саме бекенд-складовій та її здатності підтримувати потрібну бізнес-логіку.

Таблиця 1.1 – Порівняльний аналіз функціональних можливостей серверної частини систем-аналогів та GTPLANNER

Критерії порівняння (Можливості серверної частини)	Wanderlog / TripIt	Splitwise	Комбінація загальних інструментів	GTPLANNER (Backend)
Централізований REST API для Web та Mobile	Так	Так	Ні	Так
Динамічний контроль ролей (Організатор / Учасник)	Частково	Ні	Ні	Так
Комунікація в реальному часі (WebSocket)	Ні	Ні	Так	Так
Інтеграція платіжного шлюзу (Stripe / Webhooks)	Ні	Частково	Ні	Так
Автоматизовані задачі за розкладом (Cron)	Ні	Ні	Ні	Так
Централізоване збереження медіа та чеків у БД	Частково	Ні	Ні	Так

Зібрані у таблиці 1.1 дані наочно демонструють, що ринок досі не пропонує єдиного комплексного інструменту для туристичних груп. Завжди доводиться обирати між плануванням логістики та обліком грошей, оскільки вони не інтегровані на рівні бази даних. Цінність проєкту GTPLANNER якраз і полягає у створенні централізованого бекенду. Завдяки комбінації REST API та WebSocket сервер зможе паралельно працювати з веб-версією та мобільним застосунком. Це дозволить об'єднати платежі, маршрути та комунікацію в єдину транзакційну систему, мінімізуючи хаос розрізнених додатків.

1.3 Обґрунтування актуальності та декомпозиція вимог до серверної логіки

Вивчення предметної області та слабких місць наявних рішень підтверджує потребу у власному надійному бекенді для платформи GTPLANNER. Саме серверне ядро візьме на себе роль головного координатора. Воно відповідатиме за цілісність даних у базі, безпеку під час фінансових розрахунків і постійну синхронізацію інформації між вебсайтом та мобільним додатком. Щоб перевести цю загальну ідею в площину інженерної задачі, її розбили на перелік конкретних архітектурних та функціональних вимог.

Якщо говорити про управління профілями та подорожами, логіка виглядає наступним чином. Платформа має реєструвати та авторизувати людей через традиційну зв'язку електронної пошти й пароля. Приватні маршрути та чутливі дані захищатимуться за допомогою JWT-токенів. Стосовно самих поїздок, бекенд повинен вміти створювати, читати, редагувати та видаляти відповідні записи. Важливим моментом є робота із заявками. Користувачі матимуть змогу відправити запит на приєднання, а організатор — погодити його або відхилити. До того ж система змушена постійно перевіряти статус людини у кожній окремій поїздці, щоб звичайний учасник раптом не отримав доступу до налаштувань адміністратора групи.

Робота з фінансами формує чи не найважливіший пул завдань. Сервер повинен створювати запити на збір коштів та фіксувати поточні витрати. Безпосередні транзакції проходитимуть через інтегрований платіжний шлюз Stripe API. Проте банківські перекази рідко проходять миттєво. Через це бекенд має ловити асинхронні сповіщення від платіжної системи через механізм Webhooks і лише після отримання підтвердження змінювати статус оплати в базі. Додатково потрібно реалізувати функцію завантаження фотографій чеків. Разом із нею запрацює система внутрішнього голосування: група зможе переглянути чек і підтвердити, що організатор дійсно витратив ці гроші на спільні потреби.

Організація спілкування та розсилка нагадувань теж лягають на серверну частину. Щоб чат подорожі працював без ручного оновлення сторінки, знадобиться двосторонній канал зв'язку на базі WebSocket. Поруч із чатом працюватиме генератор сповіщень. Він сигналізуватиме про будь-які зміни в маршруті, появу боржників чи нові запити на участь. Аби учасники не забували про фінансові дедлайни, у систему вбудується планувальник задач. Ці фонові Cron-процеси регулярно перевірятимуть стан бази і самостійно надсилатимуть нагадування тим, хто затримує оплату або ще не проголосував за завантажений чек.

Звісно, крім бізнес-логіки існують і суворі вимоги до безпеки та архітектури. Зберігати паролі у відкритому вигляді неприпустимо, тому вони перетворюватимуться на криптографічні хеші за допомогою алгоритму bcrypt. Доступ до грошових операцій перевірятиметься на кількох рівнях одночасно. Сервер щоразу аналізуватиме не лише наявність валідного токена, а й поточні права користувача. Для надійного збереження складної структури зв'язків та запобігання конфліктам інформації як основну базу даних обрано PostgreSQL [5]. Усі дії з фінансами загортатимуться в єдину транзакцію: якщо хоч один проміжний етап дасть збій, автоматично скасується вся операція. Сам код писатиметься на базі NestJS. Модульна природа цього фреймворку дозволить у майбутньому легко додавати новий функціонал, не ламаючи вже написані частини проєкту.

Наостанок варто окреслити загальний рух інформації в системі. Мобільний додаток та вебсайт спілкуватимуться із сервером через HTTP-запити, передаючи

всі параметри у форматі JSON. Щоб уникнути потрапляння на сервер некоректних даних, на вході працюватимуть спеціальні DTO-класи [6]. Вони відповідатимуть за сувору фільтрацію та перевірку типів кожної змінної. Вхідним потоком виступатимуть параметри нових поїздок, тексти повідомлень, файли з чеками та системні виклики від платіжного шлюзу. У відповідь бекенд повертатиме структуровані об'єкти з бази. Паралельно він генеруватиме події через WebSocket, змушуючи клієнтські інтерфейси миттєво оновлюватися на екранах смартфонів чи комп'ютерів.

2 ПРОЄКТУВАННЯ АРХІТЕКТУРИ ТА СТРУКТУРИ ДАНИХ СИСТЕМИ

Другий розділ присвячено безпосередньому проєктуванню програмного ядра платформи GTPLANNER. Після того як попередній аналіз дозволив сформулювати чіткий перелік технічних вимог, наступним логічним етапом стає перетворення цих абстрактних правил на конкретну інженерну архітектуру. Головне завдання цієї частини роботи — закласти надійний фундамент, який об'єднає серверну логіку, базу даних, клієнтські інтерфейси та зовнішні платіжні сервіси в єдиний злагоджений механізм.

Спочатку увага зосереджується на виборі загальної моделі взаємодії компонентів. Детально розглядаються принципи побудови клієнт-серверної архітектури, де бекенд виступає суворо ізольованим центром прийняття рішень для вебсайту та мобільного застосунку. На цьому ж етапі формалізуються ключові ролі користувачів і сценарії їхньої поведінки, що знайде своє візуальне відображення у відповідних діаграмах прецедентів та схемах структурної організації серверної частини.

Наступним, не менш критичним кроком є проєктування єдиного сховища інформації. Оскільки платформа працює зі складною мережею зв'язків між профілями туристів, туристичними маршрутами, фінансовими транзакціями та медіафайлами, необхідно ретельно продумати структуру реляційної бази даних. У розділі описується послідовний перехід від концептуального бачення до логічної моделі. Окрема увага приділяється специфікації кожної сутності, підбору оптимальних типів даних та налаштуванню обмежень цілісності (зовнішніх ключів) у середовищі PostgreSQL. Усі спроектовані зв'язки закріплюються за допомогою ER-діаграм, що дозволяє наочно оцінити надійність, безпеку та здатність майбутньої системи до масштабування.

2.1 Обґрунтування вибору клієнт-серверної архітектури та модульного підходу

Сучасні програмні продукти вкрай рідко обмежуються лише одним середовищем взаємодії з користувачем. Вимоги ринку диктують необхідність одночасної підтримки щонайменше веб-версії для персональних комп'ютерів та нативного застосунку для мобільних пристроїв. У контексті системи планування групових подорожей GTPLANNER ця вимога є критичною: етап попереднього планування та фінансових розрахунків зручніше виконувати з великого екрана комп'ютера, тоді як під час самої поїздки учасникам потрібен миттєвий доступ до маршруту чи чату виключно зі смартфона.

Традиційний монолітний підхід, за якого сервер не лише виконує розрахунки, але й самостійно генерує HTML-сторінки (Server-Side Rendering), виявився технічно непридатним для такого сценарію. Замість нього концептуальним фундаментом платформи стала строга розподілена клієнт-серверна архітектура на базі стандарту REST. У цій парадигмі відбувається чітке розділення зон відповідальності: клієнтські застосунки відповідають виключно за презентаційний шар, а серверне ядро бере на себе всю бізнес-логіку.

Загальну візуалізацію компонентів розробленої програмної системи та їхнього взаємозв'язку наведено на рисунку 2.1.

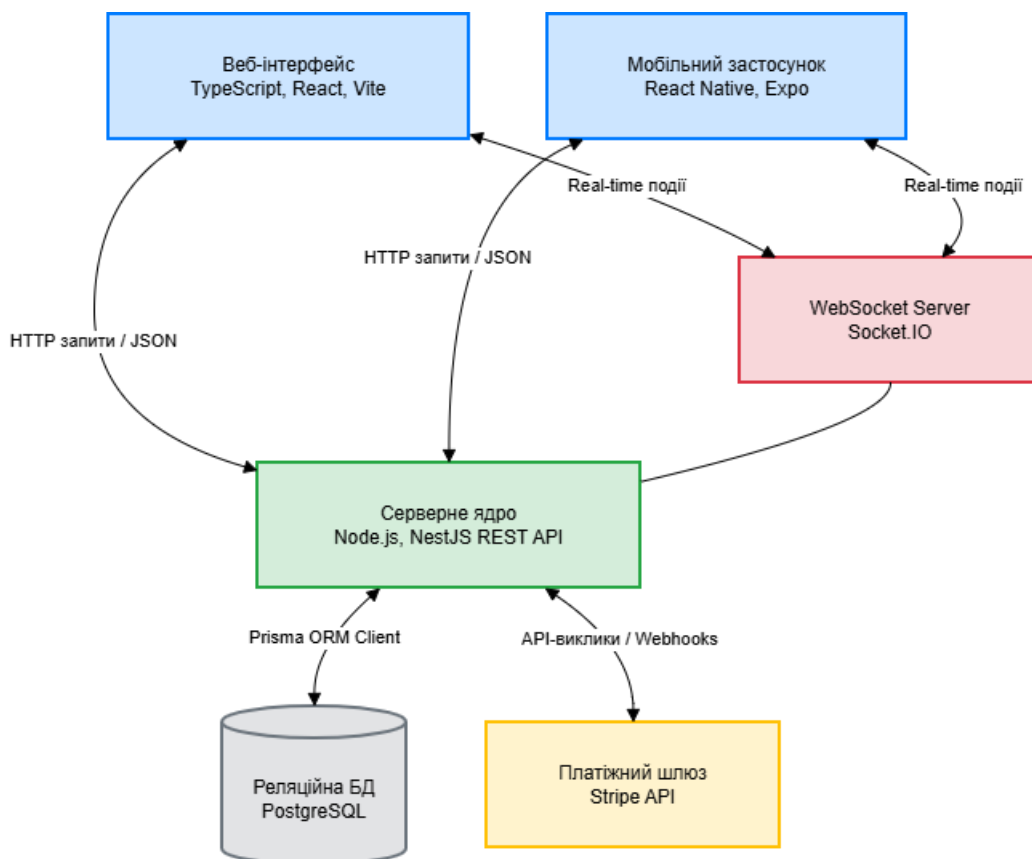


Рисунок 2.1 – Схема загальної архітектури програмної системи
GTPLANNER

Як видно з архітектурної схеми, фронтенд (на базі React) та мобільний застосунок (React Native) виступають у ролі так званих «тонких клієнтів». Їхнє завдання зводиться до відображення інтерфейсу, збору даних із форм та формування мережевого запиту до сервера. Вони не містять жодних алгоритмів розрахунку фінансів чи перевірки прав доступу, що виключає можливість компрометації даних через вразливості на пристрої користувача. Будь-яка спроба підмінити суму платежу або обійти блокування інтерфейсу заздалегідь приречена на невдачу, оскільки фінальне рішення завжди залишається за бекендом.

2.1.1 Організація серверного ядра та життєвий цикл запиту

Серверна частина побудована на базі середовища Node.js [7, 8, 9] із застосуванням фреймворку NestJS. Ключовим архітектурним рішенням стала відмова від складних мікросервісів на користь надійного модульного моноліту [10]. Такий підхід дозволяє уникнути надмірних витрат ресурсів на мережеву взаємодію між сервісами, але при цьому зберігає сувору організацію коду. Внутрішня структура застосунку жорстко розбита на незалежні предметні домени: модуль користувачів, модуль маршрутів, фінансовий модуль та модуль комунікацій. Завдяки застосуванню патерну інверсії управління (Inversion of Control) та механізму впровадження залежностей (Dependency Injection), ці модулі можуть ефективно обмінюватися даними, не утворюючи сильної зв'язності (tight coupling).

Для забезпечення надійності процесу обробки інформації архітектура передбачає багатошаровий життєвий цикл мережевого запиту. Жоден HTTP-виклик не потрапляє безпосередньо до бази даних. Детальний маршрут проходження інформації всередині бекенду проілюстровано на рисунку 2.2.

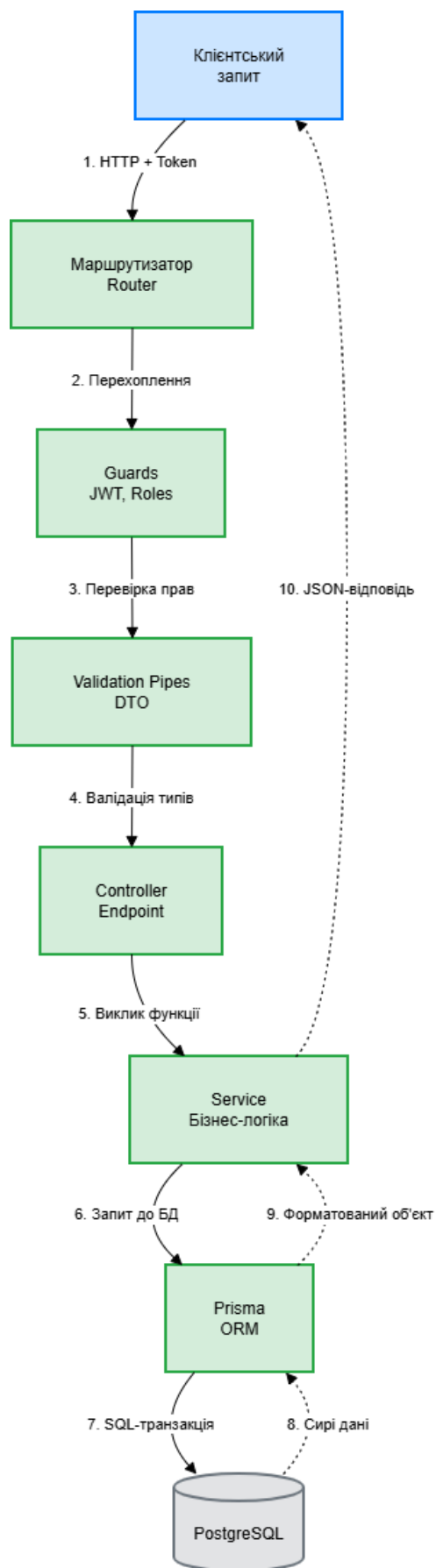


Рисунок 2.2 – Схема клієнт-серверної взаємодії та обробки REST API запиту

Коли клієнт ініціює дію (наприклад, подання заявки на участь у подорожі), запит разом із криптографічним JWT-токеном надходить на сервер. Першою лінією оборони виступають захисники (Guards). Їхнє завдання — розшифрувати токен, підтвердити особу користувача та з'ясувати, чи має він необхідну роль для виконання цієї дії. Якщо перевірка пройдена, запит передається до шару трансформації та валідації (Pipes). Тут вхідний JSON-об'єкт автоматично перевіряється на відповідність заздалегідь визначеним класам передачі даних (DTO). Наприклад, якщо замість числа для суми платежу надіслано текстовий рядок, Pipe миттєво відхилить запит із помилкою валідації, запобігаючи збоєм на рівні бази даних.

Лише після успішного проходження цих перевірок контролер (Controller) перехоплює управління та делегує його відповідному сервісу (Service), де зосереджена безпосередня бізнес-логіка. Сервіс виконує необхідні розрахунки, після чого звертається до інструменту об'єктно-реляційного відображення Prisma ORM. Саме Prisma перетворює методи TypeScript [11, 12, 13] на безпечні SQL-запити до бази даних PostgreSQL, запобігаючи ризикам SQL-ін'єкцій. Отримані з бази сирі дані сервіс пакує у стандартизований JSON-формат і через контролер повертає клієнту. Така багат шаровість робить архітектуру максимально стійкою до некоректних або шкідливих вхідних даних.

2.1.2 Проєктування системи прецедентів та рольової ієрархії

Комплексність предметної області вимагає динамічного підходу до розподілу прав доступу. Архітектура GTPLANNER оперує дворівневою системою авторизації. Перший рівень — це глобальні ролі системи. Неавторизований відвідувач (гість) обмежений у можливостях і має доступ лише до реєстрації, авторизації та перегляду списку відкритих (публічних) подорожей. Після введення правильних облікових даних гість переходить у статус авторизованого

користувача, отримуючи доступ до особистого кабінету та інструментів створення власних маршрутів.

Справжня диференціація прав відбувається на другому рівні — всередині контексту конкретної подорожі. Тут архітектура передбачає суворе розмежування повноважень між організатором та учасником. Організатор є власником маршруту, що наділяє його адміністративними привілеями: він затверджує фінансові збори, додає нові локації, змінює дати та вирішує долю поданих заявок. Учасник, своєю чергою, отримує доступ до приватного простору поїздки, але з обмеженими правами запису: він може оплачувати власні рахунки, спілкуватися в чаті, завантажувати фотографії чеків або брати участь у голосуваннях. Гнучкість архітектури полягає в тому, що ролі не прив'язуються до профілю жорстко — одна й та сама особа може виступати організатором у власній поїздки та звичайним учасником у поїздки своїх друзів.

Повний перелік сценаріїв взаємодії різних категорій користувачів із серверною частиною системи наведено на діаграмі прецедентів (рисунок 2.3).



Рисунок 2.3 – Діаграма прецедентів (Use Case) із ролями користувачів системи

Закладена логіка прецедентів гарантує цілісність бізнес-процесів. Завдяки реалізації JWT-захисту та спеціалізованих Role Guards на рівні кожного маршруту маршрутизатора (endpoint), серверне ядро здатне автономно відхиляти

несанкціоновані запити. Наприклад, якщо звичайний учасник спробує надіслати HTTP-запит на видалення подорожі (навіть обійшовши приховування кнопки на фронтенді), бекенд розпізнає невідповідність ролей і поверне помилку 403 Forbidden. Це підтверджує правильність вибору архітектурного стилю і дозволяє перейти до наступного етапу — проєктування надійної структури реляційної бази даних, яка зберігатиме всі описані взаємозв'язки.

2.2 Проєктування реляційної бази даних та розробка ER-моделі

Фундаментом будь-якої інформаційної системи є її база даних. Для платформи GTPLANNER як основне сховище інформації було обрано PostgreSQL [14, 15, 16]. Специфіка розроблюваного проєкту вимагає безпомилкової обробки фінансових операцій, збереження чіткої ієрархії ролей та миттєвої реакції на голосування учасників. Реляційна модель баз даних ідеально підходить для таких завдань, оскільки вона на рівні архітектури гарантує виконання вимог ACID (атомарність, узгодженість, ізольованість та довговічність транзакцій). Якщо під час складного переказу коштів між віртуальними рахунками хоча б один проміжний крок завершиться з помилкою, база даних автоматично відкотить усі зміни, запобігаючи втраті фінансової цілісності.

2.2.1. Нормалізація та логічна структура даних

Процес проєктування розпочався з виділення ключових сутностей предметної області та встановлення зв'язків між ними. Базовими об'єктами системи виступають користувач (User) та подорож (Trip). Очевидно, що один турист може брати участь у кількох поїздках одночасно, а одна подорож завжди містить групу користувачів. Такий класичний зв'язок типу «багато до багатьох» (Many-to-Many) неможливо реалізувати у реляційній базі безпосередньо. Щоб уникнути дублювання інформації та аномалій оновлення, логічну схему привели до третьої

нормальної форми (ЗНФ). Для цього було створено проміжну сутність учасника (TripMember), яка пов'язує конкретного користувача з конкретною поїздкою. Ця ж таблиця взяла на себе роль зберігача контекстної інформації: саме в ній фіксується, чи є людина звичайним учасником, чи організатором. Окремо винесено таблицю JoinRequest для обробки заявок на приєднання до публічних подорожей.

Найбільшій увазі вимагало проектування фінансового блоку, який є критичною частиною системи. Його структурували через п'ять взаємопов'язаних таблиць. Таблиця TripFinance зберігає глобальні фінансові налаштування поїздки, зокрема базову суму внеску та дедлайни оплат. Таблиці Payment та PaymentTransaction відповідають за фіксацію гарантійних внесків туристів та містять ідентифікатори сесій зовнішнього платіжного шлюзу Stripe. Таблиця цільових витрат (TripExpense) описує ініціативи організатора щодо зняття коштів: вона зберігає суму, мету витрати та масив посилань на завантажені файли-підтвердження. Нарешті, таблиця ExpenseVote акумулює рішення кожного учасника щодо конкретної заявки організатора. Завдяки такому розділенню система може безпечно агрегувати голоси і застосовувати математичні формули для прийняття рішень про переказ коштів.

Взаємодію користувачів поза фінансами забезпечує розгалужена мережа з майже двох десятків допоміжних таблиць. Для реалізації картографічного планування спроектовано таблицю локацій (Waypoint), яка зберігає координати та порядок відвідування. Повноцінний модуль спілкування реалізовано через таблиці повідомлень (TripChatMessage), відстеження прочитання (TripChatReadState), згадок користувачів (TripChatMention) та реакцій (TripChatReaction). Харчування групи координується через зв'язку таблиць FoodItem та FoodChoice, а для збереження медіафайлів після поїздки передбачено сутності TripMemory та TripMemoryDone.

Повну структуру бази даних, яка складається з 24 таблиць, атрибути кожної сутності та характер зв'язків між ними проілюстровано на детальній ER-діаграмі (рисунок 2.4). Для збереження читабельності на схемі відображено первинні та зовнішні ключі, а також найбільш критичні поля бізнес-логіки.

Для глибшого розуміння інженерного підходу до формування сутностей доцільно розглянути фрагмент коду. Нижче наведено лістинг моделі цільової витрати організатора, описаної мовою Prisma, яка після компіляції та міграції перетворюється на відповідну таблицю в PostgreSQL.

Лістинг 2.1 – Фрагмент структури бази даних (модель цільової витрати у Prisma)

```
model TripExpense {
  id          String @id @default(uuid())
  tripId      String
  organizerId String
  amount      Float
  description String

  proofUrl  String?
  proofUrls String[] @default([])

  status  ExpenseStatus @default(PENDING)
  attempt Int            @default(1)

  createdAt DateTime @default(now())
  updatedAt DateTime @updatedAt

  trip Trip @relation(fields: [tripId], references:
[id], onDelete: Cascade)
  votes ExpenseVote[]

  stripeTransferId String?
}

enum ExpenseStatus {
  PENDING
  APPROVED
  REJECTED
}
```

```

    CLARIFICATION
    SUPPORT_NEEDED
}

```

Як видно з лістингу, кожна колонка має чітко визначений тип даних. Для ідентифікаторів використовуються стійкі до колізій рядки формату UUID (@default(uuid())), що є стандартом безпеки для розподілених систем. Статус заявки жорстко обмежений переліком допустимих значень (enum ExpenseStatus), що унеможливорює запис некоректного тексту в цю колонку та дозволяє реалізувати логіку голосування, описану в математичній моделі системи. Атрибут stripeTransferId забезпечує надійний зв'язок між внутрішнім записом бази даних та реальною транзакцією у зовнішньому платіжному шлюзі.

Таким чином, спроектована реляційна модель забезпечує надійний фундамент для зберігання даних, дозволяючи перейти до безпосереднього опису алгоритмів фінансової взаємодії, голосування та обробки платежів.

2.3 Проєктування фінансового алгоритму та життєвого циклу бізнес-процесів

Найскладнішим інженерним викликом під час розробки туристичної платформи є реалізація надійного механізму координації спільних грошових потоків. Аналіз предметної області показав, що традиційний підхід, за якого учасники просто переказують кошти на особисту картку організатора, базується виключно на міжособистісній довірі. У разі виникнення конфліктів, скасування бронювань або відмови когось із туристів від поїздки, такий підхід миттєво призводить до фінансових втрат. Щоб розв'язати цю проблему на рівні програмної архітектури, було спроектовано багатокроковий алгоритм поетапного фінансування. Його суть полягає у використанні зовнішнього платіжного шлюзу як незалежного середовища для тимчасового утримання коштів. Гроші переказуються

організатору не одразу всією сумою, а лише частинами і тільки після колективного погодження конкретної витрати.

2.3.1. Математична модель фінансової взаємодії

Для формалізації правил поведінки системи під час збору коштів, голосування та розрахунку компенсацій було розроблено відповідну математичну модель. На початковому етапі планування, коли формується загальний бюджет подорожі, базова сума гарантійного внеску розраховується за принципом рівного розподілу. Ця залежність має такий вигляд:

$$C = \frac{S}{N}, \quad (2.1)$$

де C – базова сума гарантійного внеску одного учасника,

S – загальний запланований бюджет подорожі,

N – загальна кількість підтверджених учасників групи.

Процес витрачання зібраних коштів підпорядковується суворим обмеженням. Організатор створює цільові запити на оплату (наприклад, для бронювання житла), кожен з яких обов'язково супроводжується електронною копією чека. Програмний алгоритм на рівні бази даних гарантує, що сума всіх ініційованих запитів ніколи не перевищить зібраний бюджет. Це математично виражається як умова:

$$\sum_{j=1}^m e_j \leq S, e_j > 0, \quad (2.2)$$

де e_j – сума j -го цільового запиту на витрату,

m – загальна кількість ініційованих запитів,

S – загальний запланований бюджет подорожі.

Рішення щодо переказу коштів за створеним запитом приймається колективно. Кожен голос учасника може набувати одного з трьох значень:

позитивне (схвалення), негативне (відхилення) або нейтральне (запит на уточнення деталей). Для ухвалення фінального рішення сервер підраховує кількість голосів кожної категорії та застосовує таке правило:

$$Decision(j) = \begin{cases} \text{Approved}, A_j > R_j \wedge Q_j = 0 \\ \text{Rejected}, R_j \geq \frac{N}{2} \\ \text{Clarification}, Q_j > 0 \end{cases}, \quad (2.3)$$

де $Decision(j)$ – кінцевий статус j -го запиту на виплату,

A_j – кількість учасників, що проголосували за схвалення,

R_j – кількість учасників, що проголосували проти,

Q_j – кількість запитів на уточнення деталей,

N – загальна кількість учасників групи, які мають право голосу.

Математичний апарат також враховує складні життєві ситуації, зокрема вихід людини з групи до завершення поїздки. Система не може повернути користувачу всю суму початкового внеску, оскільки частина грошей вже могла бути витрачена на незворотні бронювання. Сума повернення обчислюється як різниця між внеском та часткою вже затверджених колективних витрат:

$$Refund(u) = \begin{cases} C, \text{ подорож скасовано до початку} \\ \max\left(0, C - \frac{E^{approved}}{N}\right), \text{ вихід до початку подорожі,} \\ 0, \text{ вихід після початку подорожі} \end{cases}, \quad (2.4)$$

де $Refund(u)$ – сума коштів, що підлягає поверненню учаснику u ,

C – початковий гарантійний внесок учасника,

$E^{approved}$ – сумарна вартість усіх уже затверджених витрат на момент виходу,

N – загальна кількість учасників.

2.3.2 Діаграма послідовності для сценарію подання та прийняття заявки

Кожен бізнес-процес у системі являє собою ланцюжок мережових взаємодій між клієнтським застосунком, сервером та базою даних. Розглянемо сценарій приєднання нового користувача до публічної подорожі. Цей процес починається з того, що турист знаходить цікавий маршрут і натискає кнопку подання заявки. Клієнтська програма формує HTTP-запит, долучаючи до нього JWT-токен для підтвердження особи [19, 20]. Сервер перевіряє токен, переконується, що користувач ще не є учасником цієї поїздки, і створює у базі даних PostgreSQL запис у таблиці JoinRequest зі статусом очікування.

Одночасно з цим вмикається модуль комунікації в реальному часі. Через WebSocket-з'єднання сервер миттєво надсилає організатору подорожі сповіщення про нову заявку. Організатор, отримавши сигнал, переглядає профіль кандидата та ініціює зустрічний HTTP-запит на підтвердження. Бекенд валідує права організатора, змінює статус заявки на схвалений та автоматично генерує новий запис у таблиці TripMember, офіційно надаючи кандидату права учасника. Детальну візуалізацію цього процесу наведено на рисунку 2.5.

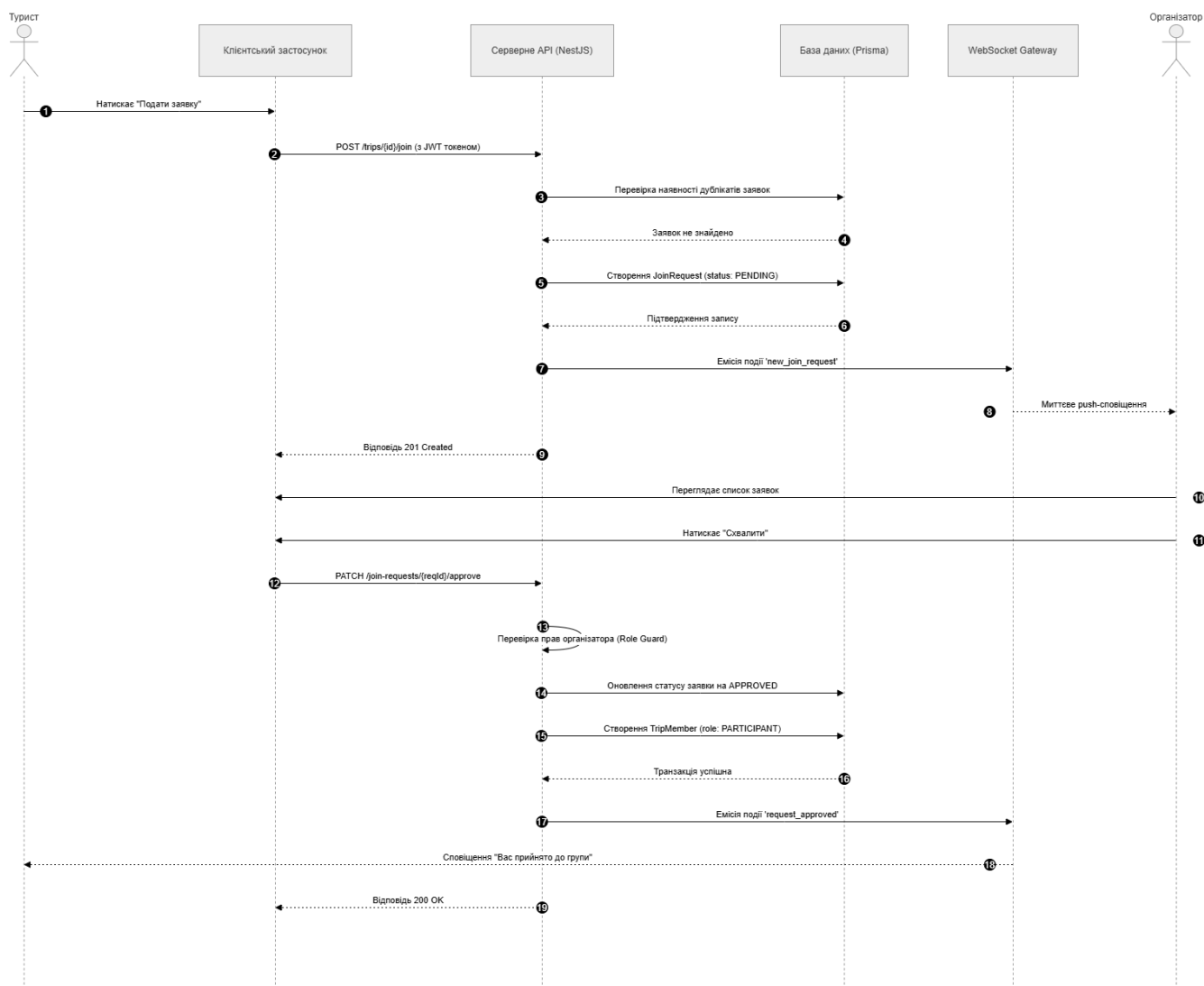


Рисунок 2.5 – Діаграма послідовності обробки заявки на приєднання до подорожі

2.3.3. Діаграма послідовності обробки платежу та інтеграція зі Stripe

Після успішного приєднання до групи турист зобов'язаний сплатити свою частку гарантійного внеску. Робота з реальними фінансами вимагає абсолютної ізоляції платіжних даних від внутрішньої бази системи для дотримання стандартів безпеки PCI DSS. Тому інтеграція побудована на базі API платіжного провайдера Stripe. Сервер ніколи не отримує номерів кредитних карток, натомість він оперує абстрактними ідентифікаторами сесій.

Коли учасник ініціює оплату, серверне ядро звертається до Stripe API із запитом на створення захищеної платіжної сесії (Checkout Session), передаючи суму та спеціальні метадані. Згенероване посилання повертається клієнту, який перенаправляє туриста на безпечну сторінку еквайрингу. Оскільки введення реквізитів і підтвердження від банку може тривати кілька хвилин, з'єднання між клієнтом і сервером розривається. Коли банк остаточно списує кошти, Stripe самостійно надсилає асинхронний серверний запит — Webhook — на спеціально відкритий маршрут нашого бекенду. Сервер перевіряє криптографічний підпис вебхука, переконується у відсутності підміни даних, знаходить відповідний запис у таблиці Payment та змінює його статус на оплачений. Опис інформаційного обміну наведено на рисунку 2.6.

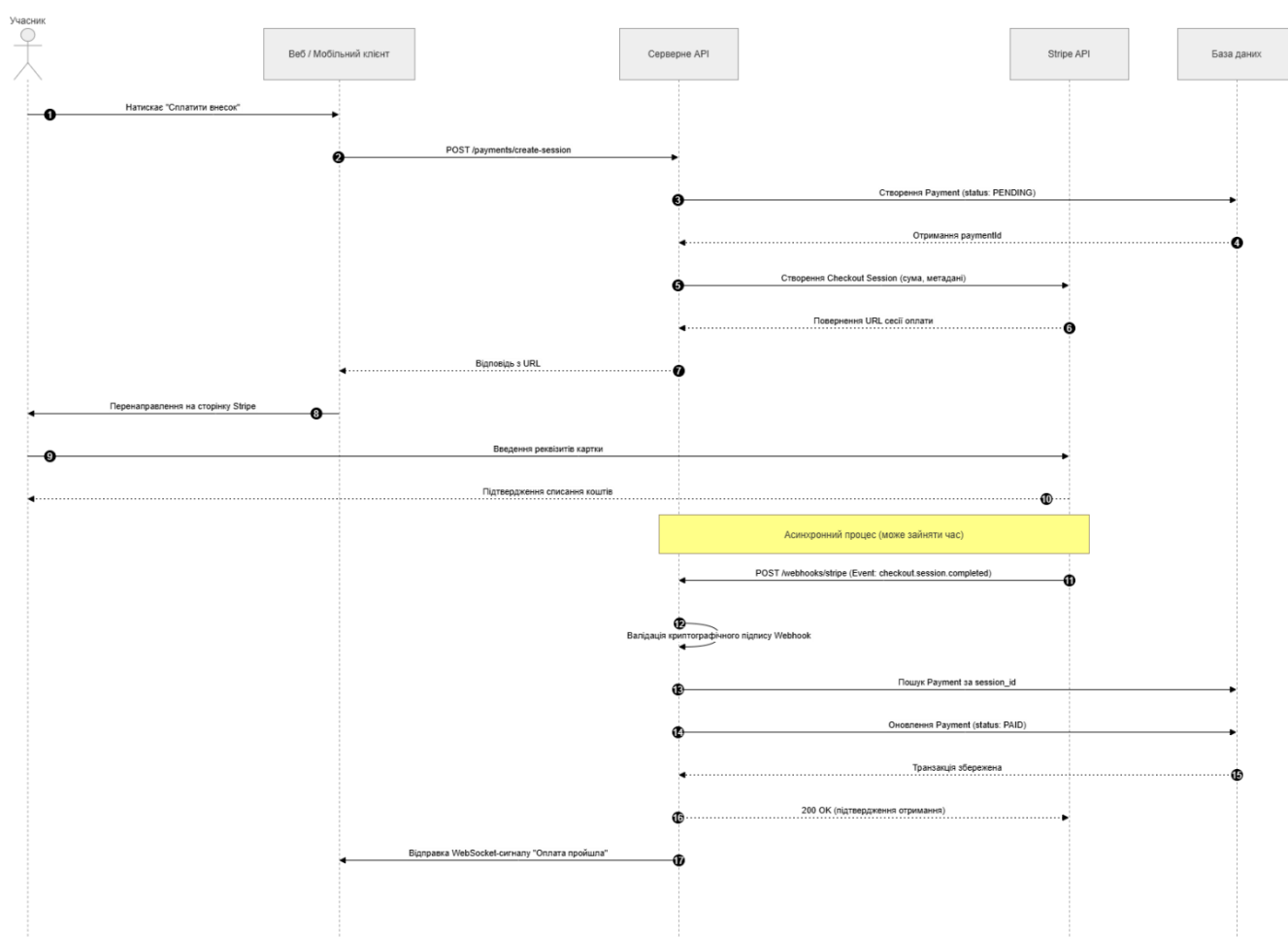


Рисунок 2.6 – Діаграма послідовності створення платіжної сесії та обробки Webhook

2.3.4 Процес цільового витрачання коштів та система голосування

Коли організатор фізично сплачує за послугу (наприклад, купує квитки на потяг для всієї групи), він має отримати компенсацію із загального фонду, що утримується на балансі системи. Цей процес перетворюється на структурований життєвий цикл цільової витрати. Організатор заповнює форму в застосунку, вказуючи витрачену суму, і обов'язково прикріплює фотографію фіскального чека. Файл завантажується на сервер, зберігається у хмарному сховищі, а його URL-адреса записується у таблицю TripExpense.

Після збереження заявки система блокує можливість її редагування та розпочинає процедуру голосування. Учасники отримують повідомлення із проханням перевірити чек. Кожен турист надсилає свій голос (підтвердити або відхилити), який зберігається у таблиці ExpenseVote. Щоразу, коли додається новий голос, серверна бізнес-логіка перераховує загальний баланс прийнятих рішень. Як тільки кількість погоджень досягає математичної більшості, сервер ініціює API-запит до Stripe на переказ відповідної суми з віртуального рахунку поїздки безпосередньо на банківську картку організатора (Stripe Transfer). Хронологію цих дій проілюстровано на рисунку 2.7.

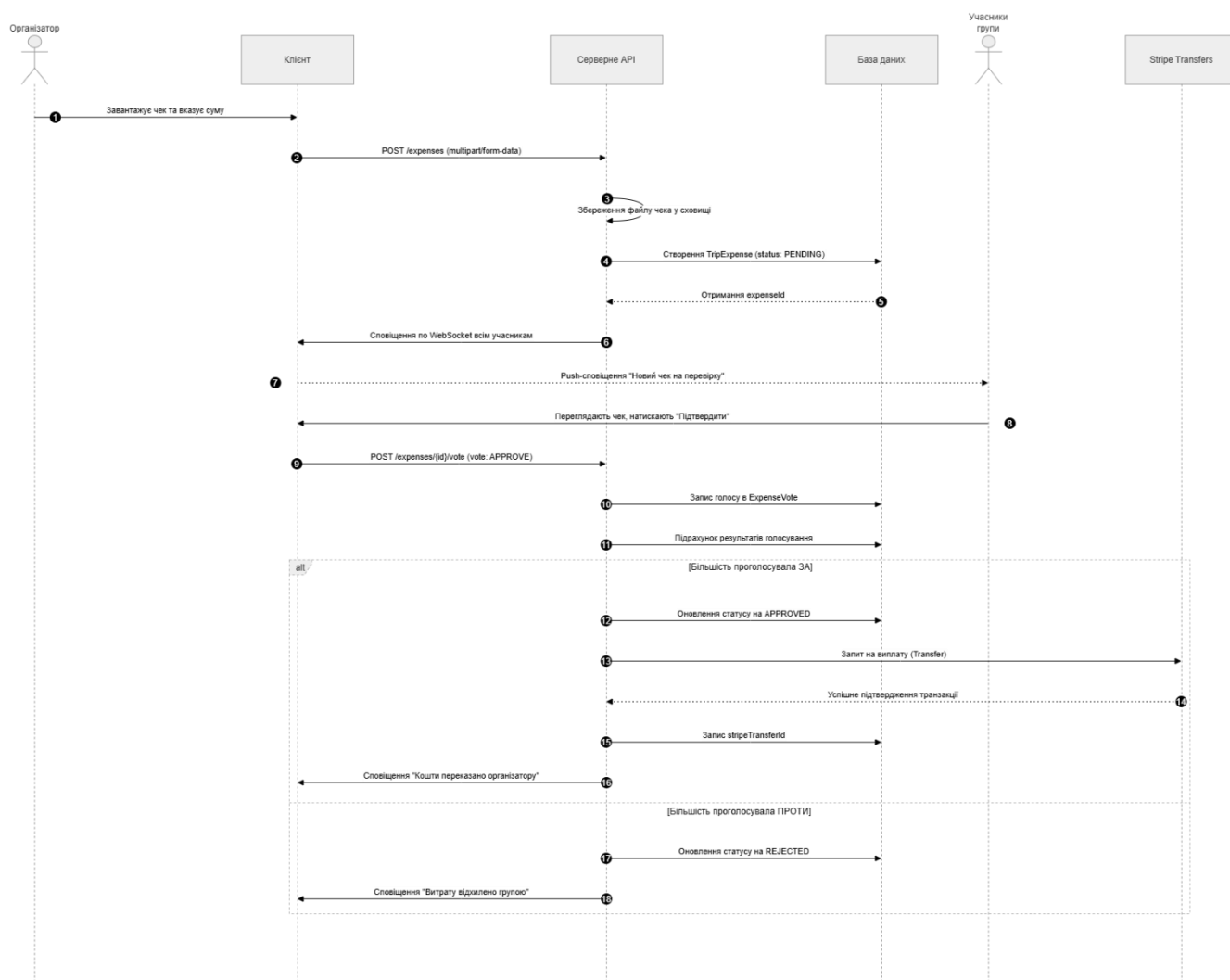


Рисунок 2.7 – Діаграма послідовності фінансового workflow: створення заявки, голосування та виплата

Такий підхід до проєктування послідовностей повністю ліквідує хаос, характерний для планування в звичайних месенджерах. Усі фінансові дії стають алгоритмізованими, прозорими та юридично прив'язаними до конкретних користувацьких рішень у базі даних, що повністю виконує поставлену перед дипломним проєктом мету.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ СИСТЕМИ

У третьому розділі висвітлено процес перетворення розроблених теоретичних моделей на готовий програмний продукт. Попередні етапи дозволили сформулювати архітектуру, спроектувати базу даних та визначити математичні алгоритми. Натомість зараз основний фокус зміщується безпосередньо на написання коду. Завдання полягає у створенні надійного серверного бекенду для платформи GTPLANNER. Ця частина системи має безперебійно працювати одночасно з вебсайтом та мобільним додатком.

Матеріали розділу деталізують хід розробки серверного ядра на базі середовища Node.js та фреймворку NestJS. Описано кроки зі створення окремих програмних модулів і налаштування маршрутів. Також показано механізм обробки вхідних запитів та специфіку роботи з базою даних PostgreSQL через інструментарій Prisma. Значне місце відведено інтеграції зовнішніх сервісів. Йдеться насамперед про підключення платіжної платформи Stripe для безпечного управління грошима групи. Крім того, розкрито технічні нюанси налаштування WebSocket-з'єднання, яке забезпечує роботу внутрішнього чату без затримок.

Логічним завершенням етапу розробки є експериментальна перевірка написаного коду. Необхідно було впевнитися, що бекенд правильно обробляє дані, контролює рівні доступу та формує коректні відповіді для клієнтських пристроїв. Для цього проведено тестування основних API-маршрутів у середовищі Postman. Отримані результати підтверджують правильність роботи закладених фінансових алгоритмів і доводять готовність створеного програмного ядра до подальшої експлуатації.

3.1 Розробка серверних модулів та інтеграція бізнес-логіки

Процес програмної реалізації серверної частини розпочався з ініціалізації робочого середовища на базі платформи Node.js та розгортання каркасу застосунку за допомогою інструментарію фреймворку NestJS. Відповідно до спроектованої архітектури модульного моноліту, кодову базу було жорстко сегментовано на незалежні предметні домени. У кореневій директорії проєкту створено окремі пакети для ключових функціональних блоків: модуля автентифікації, управління профілями, подорожами, модуля комунікацій та фінансового процесора. Кожен такий блок містить власний контролер, сервіси бізнес-логіки та класи об'єктів передачі даних (див. лістинг коду в Додатку А). Зв'язок між модулями забезпечується виключно через механізм інверсії управління (Dependency Injection). Це дозволяє, наприклад, модулю повідомлень безпечно звертатися до сервісу користувачів для отримання імен авторів тексту, не порушуючи при цьому загальної ізоляції програмного коду.

Наступним критичним етапом стала організація взаємодії з базою даних PostgreSQL. Для цього розроблену раніше декларативну схему з 24 таблиць було скомпільовано у типізований клієнт Prisma Client. Щоб інтегрувати цей інструмент у загальну екосистему застосунку, було написано спеціальний клас-обгортку, який наслідує базові інтерфейси NestJS для управління життєвим циклом модуля. Під час запуску сервера цей сервіс автоматично встановлює пул з'єднань із базою даних, а під час зупинки — коректно їх закриває. Завдяки такому підходу всі інші сервіси застосунку отримують безпечний та повністю типізований доступ до реляційних даних. Розробник у середовищі написання коду автоматично бачить усі доступні колонки та зв'язки, що звело до мінімуму ризик виникнення синтаксичних помилок під час формування складних запитів на вибірку інформації.

Маршрутизація вхідних HTTP-запитів та їх первинна обробка реалізована на рівні контролерів. Кожен метод контролера жорстко прив'язаний до конкретного REST-ендпоінту за допомогою відповідних декораторів шляху та методу. Проте

перед тим, як передати управління сервісу, система здійснює сувору валідацію вхідної інформації. Для цього застосовано механізм глобальних перехоплювачів (Validation Pipes) у комбінації з бібліотеками перевірки та трансформації класів. Під час створення нової заявки на цільову витрату клієнт відправляє JSON-об'єкт. Сервер автоматично перевіряє, чи є сума витрати додатним числом, чи відповідає ідентифікатор подорожі формату UUID і чи присутнє посилання на фотографію чека у вигляді валідної URL-адреси. Якщо хоча б одне поле не задовольняє встановлені правила, запит негайно відхиляється з помилкою формату вхідних даних, надійно захищаючи бізнес-логіку від некоректних параметрів.

Безпосередня бізнес-логіка платформи зосереджена всередині класів-сервісів. Саме тут реалізовані фінансові та координаційні алгоритми, описані в математичній моделі системи. Характерним прикладом є сервіс обробки подорожей, який керує процесом створення поїздок та базовим розподілом ролей. Коли користувач надсилає запит на планування нового маршруту, сервіс не просто записує рядок у таблицю Trip. Він ініціює комплексну транзакцію бази даних, яка гарантує атомарність операції. Спочатку створюється базовий запис про поїздки, потім формуються дефолтні налаштування фінансового модуля у таблиці TripFinance, і зрештою система автоматично створює зв'язок у таблиці TripMember, закріплюючи за творцем маршруту ексклюзивні права організатора. У разі виникнення програмного збою на будь-якому з цих етапів транзакція відкочується, залишаючи базу даних у первісному узгодженому стані.

Окремий масив програмного коду присвячено гарантуванню безпеки та динамічному розмежуванню доступу. Для захисту приватних маршрутів розроблено систему кастомних охоронців (Guards). Базовий рівень безпеки забезпечує охоронець авторизації, який перехоплює запит, витягує криптографічний токен із заголовка, розшифровує його та додає ідентифікатор користувача до системного об'єкта. Однак для туристичної платформи зі складною ієрархією цього недостатньо, тому додатково було створено спеціалізований охоронець ролей. Цей клас аналізує параметри запиту, знаходить ідентифікатор цільової подорожі та виконує швидкий запит до бази даних, щоб з'ясувати

поточний статус користувача у цій конкретній групі. Завдяки впровадженню такого механізму спроба звичайного учасника ініціювати видалення чужого маршруту або змінити фінансові дедлайни фізично блокується бекендом ще до етапу виконання основної логіки контролера.

3.2 Програмна реалізація фінансового модуля та підсистеми реального часу

Забезпечення повноцінного функціонування платформи вимагало виходу за межі базових CRUD-операцій та залучення зовнішніх інструментів для обробки специфічних бізнес-процесів. Оскільки система GTPLANNER оперує реальними коштами користувачів, архітектура повинна відповідати суворим міжнародним стандартам безпеки індустрії платіжних карток (PCI DSS) [21]. Зберігання чи безпосередня обробка номерів банківських карток на власному сервері є неприпустимою. З огляду на це, логіку грошових переказів було делеговано зовнішньому платіжному провайдеру Stripe [22, 23], тоді як внутрішній бекенд узяв на себе виключно координаційну та облікову роль.

Програмна інтеграція зі Stripe реалізована за принципом ізольованих сесій. Коли учасник подорожі ініціює сплату гарантійного внеску, клієнтський застосунок звертається до відповідного контролера NestJS. Сервер за допомогою офіційного SDK генерує унікальну платіжну сесію (Checkout Session), передаючи до шлюзу інформацію про суму, валюту та ідентифікатори користувача. У відповідь бекенд отримує тимчасове посилання, за яким клієнт перенаправляється на захищену сторінку еквайрингу. Оскільки процес введення реквізитів і банківського підтвердження відбувається асинхронно і може тривати певний час, сервер не утримує відкрите HTTP-з'єднання. Натомість реалізовано механізм прослуховування вебхуків (Webhooks). Спеціальний контролер очікує вхідних POST-запитів від серверів Stripe. При отриманні події про успішне списання коштів система спочатку валідує криптографічний підпис запиту, щоб унеможливити

підробку даних, а потім оновлює статус відповідного запису в таблиці бази даних, використовуючи унікальний ідентифікатор наміру платежу.

Другий етап фінансового циклу передбачає виплату коштів організатору після успішного голосування групи за завантажений чек. Для вирішення цього завдання застосовано інструментарій Stripe Connect. Під час реєстрації на платформі організатор прив'язує свій банківський рахунок до системи, створюючи пов'язаний обліковий запис. Як тільки математичний алгоритм бекенду фіксує достатню кількість позитивних голосів учасників у базі даних Prisma, спеціальний сервіс ініціює програмний переказ (Stripe Transfer) із віртуального балансу подорожі безпосередньо на картку організатора. Ідентифікатор цієї транзакції зберігається у колонці цільової витрати, що забезпечує повний аудит руху коштів та унеможливлює повторні виплати за одним і тим самим чеком.

Окрім фінансової логіки, критично важливим аспектом роботи системи є координація користувачів під час самої подорожі. Традиційна архітектура REST API виявилася неефективною для функціонування внутрішнього чату та системи сповіщень, оскільки вона вимагає від клієнтських пристроїв постійно надсилати повторні запити для перевірки нових даних. Цю проблему було розв'язано шляхом впровадження протоколу WebSocket [24]. У середовищі NestJS для цього застосовано бібліотеку Socket.IO, яка дозволяє підтримувати постійний дуплексний канал зв'язку між клієнтом і сервером із мінімальними затримками.

Програмна реалізація підсистеми реального часу побудована на концепції шлюзів (Gateways). Під час встановлення первинного з'єднання WebSocket-шлюз автоматично перехоплює процес рукоштовування (handshake) та витягує JWT-токен із заголовків авторизації. Якщо токен недійсний, з'єднання миттєво обривається на рівні сокета. Для оптимізації мережевого трафіку застосовано механізм ізольованих кімнат. Як тільки користувач відкриває екран конкретної поїздки, його клієнтський застосунок надсилає подію приєднання, і сервер підписує цей сокет на відповідну кімнату маршруту. Відтепер будь-яке нове повідомлення, інформація про додавання локації на карту або сповіщення про нову заявку на участь маршрутизується виключно тим користувачам, які перебувають у цій віртуальній

кімнаті, замість того щоб розсилати події всім активним клієнтам платформи. Таке архітектурне рішення значно знижує навантаження на центральний процесор сервера та заощаджує заряд батареї на мобільних пристроях учасників.

3.3 Тестування REST API та перевірка сценаріїв роботи

Після завершення етапу програмування виникла необхідність експериментальної перевірки працездатності створених механізмів. Оскільки розроблене серверне ядро функціонує як незалежний REST API і не має власного графічного інтерфейсу, для перевірки HTTP-статусів [25] та імітації поведінки клієнтських застосунків застосовувалися базові методи тестування [26] у програмному середовищі Postman [27]. Цей інструмент дозволяє конструювати будь-які типи мережевих запитів, гнучко налаштовувати HTTP-заголовки та детально аналізувати структуру отриманих відповідей від сервера. Головною метою тестування стало підтвердження того, що бекенд коректно обробляє інформацію, правильно застосовує рольові обмеження та безпомилково виконує фінансові розрахунки у базі даних PostgreSQL.

Першочергової перевірки вимагав модуль безпеки та автентифікації. У середовищі Postman було створено колекцію запитів для симуляції процесу входу в систему. При відправленні POST-запиту з коректною електронною поштою та паролем сервер успішно генерував відповідь зі статусом 200 OK, повертаючи зашифрований JWT-токен доступу. Далі перевірялася надійність системи захисту маршрутів (Guards). Було виконано спробу звернутися до захищеного ендпоінту отримання списку приватних подорожей без передачі токена в заголовок Authorization. Як і передбачалося архітектурою, сервер негайно перервав обробку такого виклику та повернув помилку 401 Unauthorized. Наступним кроком стала перевірка диференціації прав доступу всередині конкретної поїздки. Для цього від імені звичайного учасника (з відповідним токеном) було надіслано PATCH-запит на зміну фінансових дедлайнів, що є виключною компетенцією організатора.

Бекенд успішно розпізнав невідповідність ролей та заблокував дію, повернувши статус 403 Forbidden.

Наступний масив експериментів стосувався перевірки основної бізнес-логіки управління маршрутами та обробки заявок. Було протестовано сценарій подання запиту на приєднання до публічної подорожі. При відправленні відповідного POST-запиту в базі даних успішно створювався запис зі статусом очікування, а сервер повертав код 201 Created. Особливу увагу приділили перевірці реакції системи на некоректні вхідні дані, щоб протестувати роботу механізму валідації (Validation Pipes). Коли у тілі запиту на створення нової цільової витрати замість числового значення суми було навмисно передано текстовий рядок, DTO-класи NestJS миттєво відхилили цей запит. Сервер повернув деталізовану помилку 400 Bad Request із чітким описом того, яке саме поле не пройшло перевірку типів, що гарантує захист бази даних від потрапляння невалідного контенту. Скріншот успішного виконання комплексного запиту на створення подорожі наведено на рисунку 3.1.

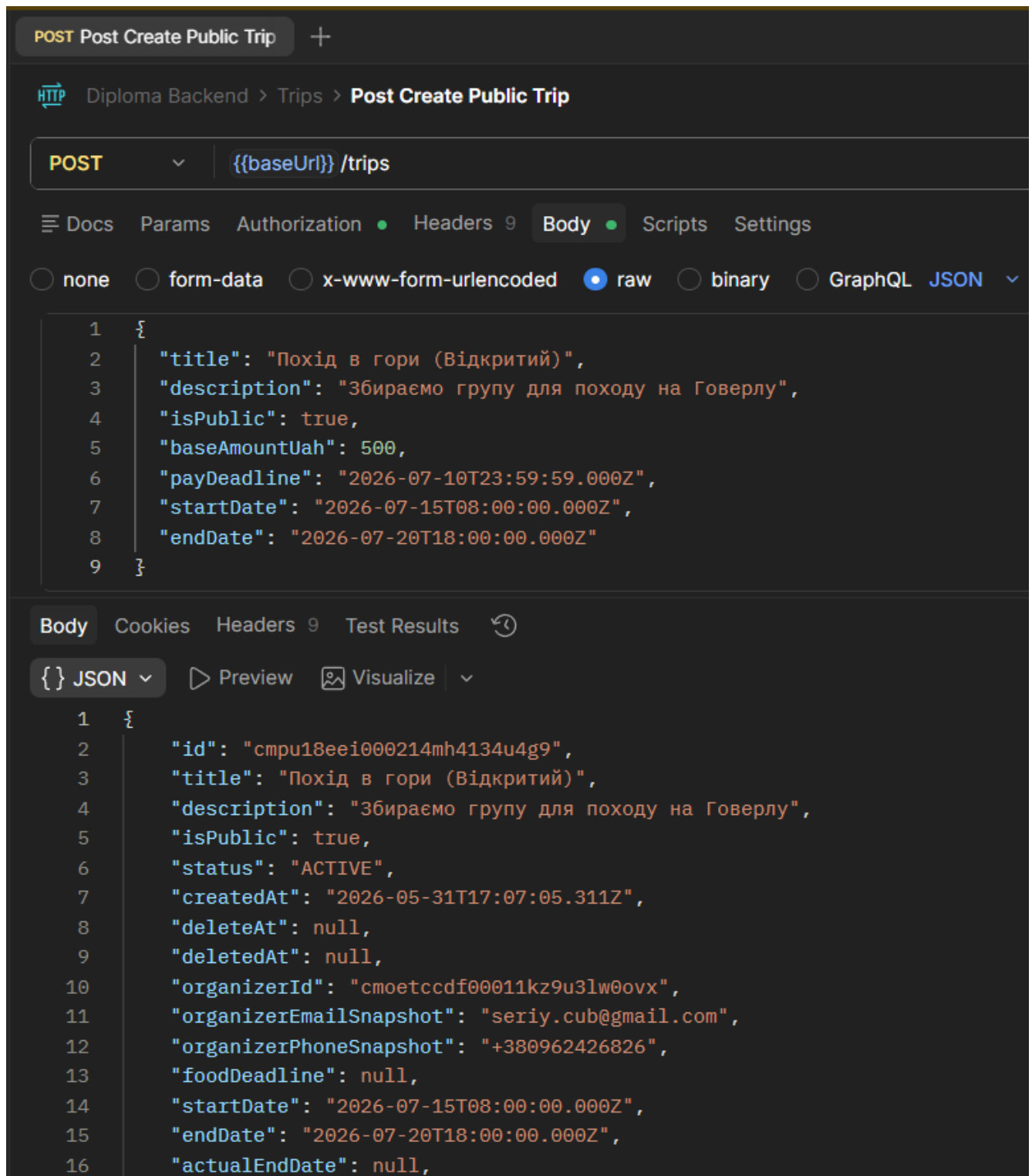


Рисунок 3.1 – Створення нової подорожі

Найбільш складною і водночас найважливішою частиною тестування стала симуляція фінансових процесів. Оскільки реальне списання коштів відбувається на стороні Stripe, у Postman було змодельовано процес отримання асинхронного

Webhook-запиту. До спеціально відкритого маршруту сервера було надіслано згенерований POST-запит, який імітував подію checkout.session.completed (успішна оплата гарантійного внеску). Серверна логіка успішно пройшла етап перевірки цифрового підпису, знайшла відповідну транзакцію в базі даних та автоматично змінила її статус з PENDING на PAID. Після цього тестувалася система голосування за цільові витрати. Шляхом відправлення серії запитів від різних користувачів імітувалося колективне ухвалення рішення. Щойно кількість позитивних голосів досягла математичної більшості, закладеної в алгоритмі, статус чека автоматично змінився на APPROVED, що підтвердило правильність роботи спроектованої фінансової моделі. Візуалізацію результатів тестування алгоритму голосування представлено на рисунку 3.2.

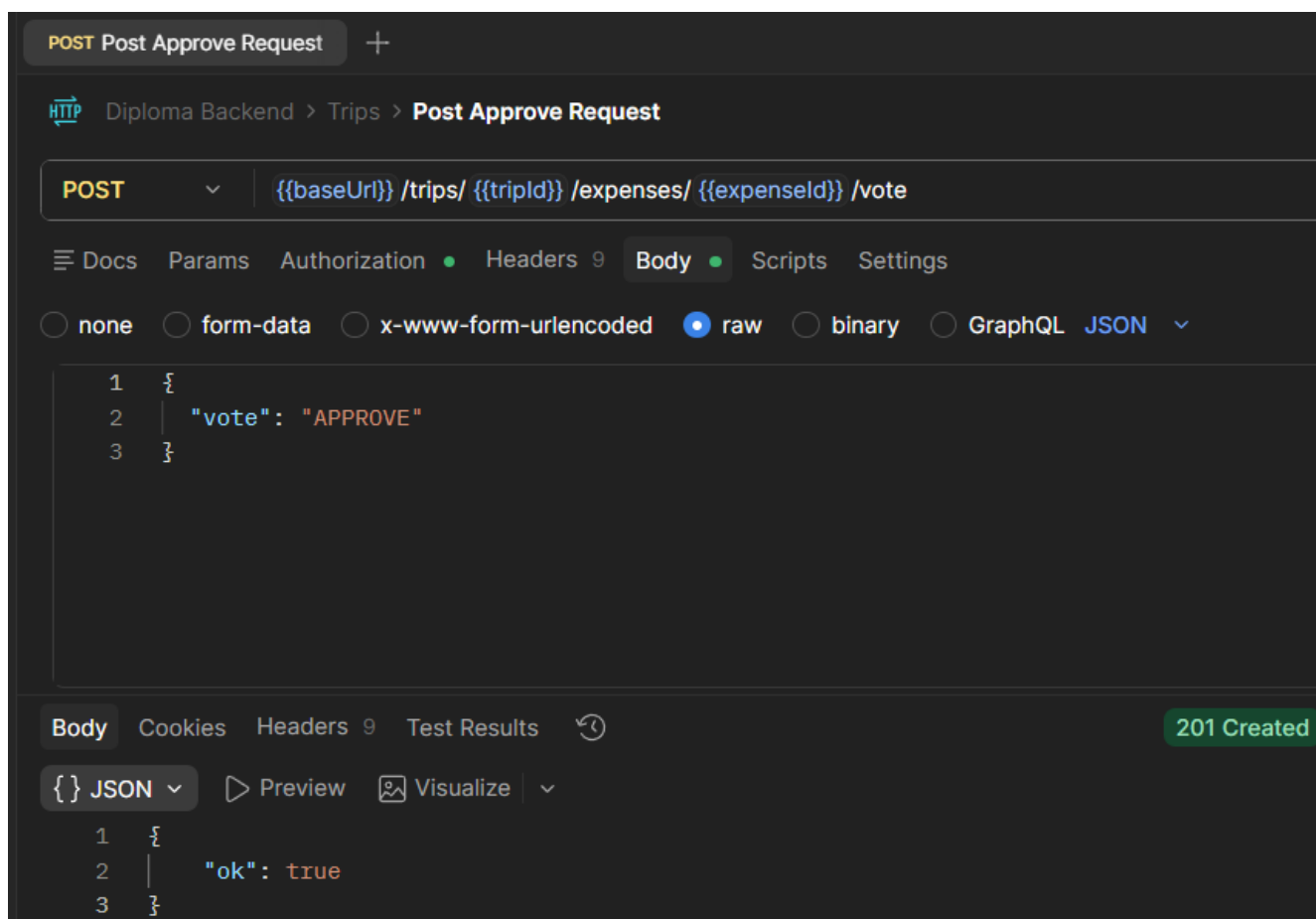


Рисунок 3.2 – Голосування за цільову витрату (чек)

Завершальним етапом стала перевірка підсистеми реального часу. Оскільки сучасні версії Postman підтримують протокол WebSocket, було створено пряме підключення до розгорнутого шлюзу (Gateway). Після успішного проходження рукописання та передачі токена, клієнт приєднався до віртуальної кімнати конкретної подорожі. Коли через паралельне вікно відправлявся звичайний HTTP-запит на додавання нового повідомлення в чат, сервер миттєво, без додаткових запитів, генерував WebSocket-подію і надсилав її до відкритого з'єднання. Отримані під час тестування результати переконливо доводять, що серверне ядро повністю відповідає висунутим функціональним вимогам, стабільно витримує спроби несанкціонованого доступу та готове до інтеграції з реальними клієнтськими інтерфейсами вебсайту та мобільного застосунку.

4 РОЗГОРТАННЯ ТА ДЕМОНСТРАЦІЯ РОБОТИ ПРОГРАМНОЇ СИСТЕМИ

У четвертому розділі описано завершальний етап роботи над дипломним проектом. Оскільки програмний код уже написаний, систему потрібно перенести на робочий сервер. Основна мета цієї частини — підготувати розроблену платформу GTPLANNER до повноцінного запуску.

Спочатку визначено апаратні вимоги до серверного обладнання. Вони потрібні для того, щоб бекенд працював швидко і без збоїв під навантаженням. Далі показано кроки з розгортання проєкту. Інструкція пояснює нюанси встановлення платформи Node.js та підключення реляційної бази даних PostgreSQL. Окрему увагу приділено конфігурації змінних оточення. Це обов'язкова умова для безпечної роботи зовнішніх платіжних систем.

Крім технічного розгортання, роботу платформи необхідно показати на практиці. Для цього наведено детальні користувацькі сценарії. Вони ілюструють весь шлях туриста в застосунку. Описано кроки від створення маршруту і додавання учасників до складних операцій, таких як голосування за чеки та переказ коштів. Результати цих тестів підтверджують, що закладені бізнес-процеси виконуються без помилок. Вони також яскраво демонструють, чому спеціалізована платформа набагато зручніша за звичайні чати під час планування групових поїздок.

4.1 Інсталяція програмного забезпечення та вимоги до обчислювальної техніки

Для забезпечення стабільної та безперебійної роботи розробленої серверної частини платформи необхідно підготувати відповідне апаратне та програмне середовище. Оскільки архітектура бекенду базується на фреймворку NestJS та використовує інструментарій Prisma для роботи з базою даних, система висуває певні вимоги до обчислювальних ресурсів. Для розгортання базового вузла, здатного обслуговувати одночасні підключення користувачів по протоколу WebSocket та обробляти фінансові транзакції, рекомендовано використовувати віртуальний або фізичний виділений сервер із мінімум двома процесорними ядрами та чотирма гігабайтами оперативної пам'яті. Для зберігання реляційних даних, медіафайлів подорожі та резервних копій системи доцільно виділити твердотільний накопичувач (SSD) об'ємом від 20 гігабайтів, що забезпечить високу швидкість виконання операцій читання та запису під час складних вибірок.

Базовою операційною системою для серверного середовища є дистрибутив сімейства Linux, наприклад Ubuntu актуальної версії з тривалою підтримкою, що гарантує високу надійність та безпеку мережових з'єднань. Головним програмним забезпеченням, яке необхідно інсталювати на сервер до початку розгортання коду, є середовище виконання Node.js версії 20.x або вище, а також система керування реляційними базами даних PostgreSQL. Крім того, для автоматизованого завантаження необхідних бібліотек та модулів фреймворку використовується стандартний менеджер пакетів Node Package Manager (npm).

Процес інсталяції безпосередньо програмної системи розпочинається з копіювання вихідного коду на сервер та завантаження всіх залежностей. Після цього настає критично важливий етап конфігурації змінного оточення. У кореневій директорії проєкту створюється спеціальний файл конфігурації, куди адміністратор вписує конфіденційні параметри. До цього переліку обов'язково входять рядок підключення до бази даних PostgreSQL, секретний ключ для підпису

криптографічних JWT-токенів автентифікації та закриті ключі доступу до платіжного шлюзу Stripe. Фізичне відокремлення цих налаштувань від програмного коду гарантує, що у разі витоку файлів системи злоумисники не отримають доступу до фінансів та бази користувачів.

Після успішного налаштування середовища ініціюється підготовка структури бази даних. За допомогою командного рядка Prisma виконується процедура розгортання міграцій. Алгоритм автоматично зчитує декларативну схему проєкту та послідовно створює всі необхідні таблиці, налаштовує зовнішні ключі для забезпечення цілісності та генерує переліки (Enums) безпосередньо у рушії PostgreSQL. Наступним кроком компілюється внутрішній клієнт для взаємодії коду з даними. Завершується інсталяція збиранням оптимізованого робочого білда (процес компіляції TypeScript у JavaScript) та запуском головного серверного процесу. Після ініціалізації всіх модулів сервер відкриває вказаний мережевий порт і переходить у режим очікування HTTP-запитів від вебклієнтів та мобільних застосунків.

4.2 Демонстрація функціоналу та представлення сценаріїв роботи програмної системи за тестами

Цей підрозділ присвячено детальній демонстрації роботи системи з позиції кінцевого користувача. Головна мета полягає у практичній перевірці всіх закладених функцій. Тестування проводилося шляхом повного проходження життєвого циклу планування групової поїздки. Базовий сценарій починається з реєстрації нових облікових записів. Користувач відкриває клієнтський застосунок і вводить свої дані. Сервер успішно валідує електронну пошту та пароль. Після входу в систему на екрані відображається головна панель управління. Тут розташований порожній список активних маршрутів, який потрібно наповнити.

Перший ключовий етап роботи – це створення нової подорожі. Організатор натискає відповідну кнопку у своєму профілі. На екрані відкривається спеціальна форма налаштувань (рис. 4.1).

Створення подорожі

Заповніть основну інформацію про подорож, маршрут, склад групи та фінанси.

Назва подорожі *

Похід в гори

12/100

Опис

Відпочинок в Карпатах

☒ Одноденна подорож

Початок * **Кінець ***

01.06.2026 10:00 03.06.2026 14:00

Склад групи та вік

Кого запрошуємо (стать):

☒ Змішана (всі) ☐ Тільки чоловіки ☐ Тільки жінки

Вікові обмеження *

☒ Для будь-якого віку

Рисунок 4.1 – Інтерфейс створення нової подорожі

Турист вводить назву поїздки, обирає дати та вказує локацію. Окремим блоком налаштовуються фінансові правила групи. Система вимагає обов'язково вказати суму базового внеску та кінцевий термін його сплати. Після збереження інформації бекенд генерує унікальне посилання. Організатор надсилає цей код

іншим людям для запрошення. Для перевірки роботи цього механізму застосовувався другий тестовий акаунт. Наприклад, інший користувач вводить отриманий код у своєму застосунку. Сервер миттєво створює нову заявку на приєднання до маршруту. Організатор бачить цей запит в інтерфейсі і схвалює його (рис. 4.2). Після цього новий турист отримує повний доступ до спільного простору подорожі.

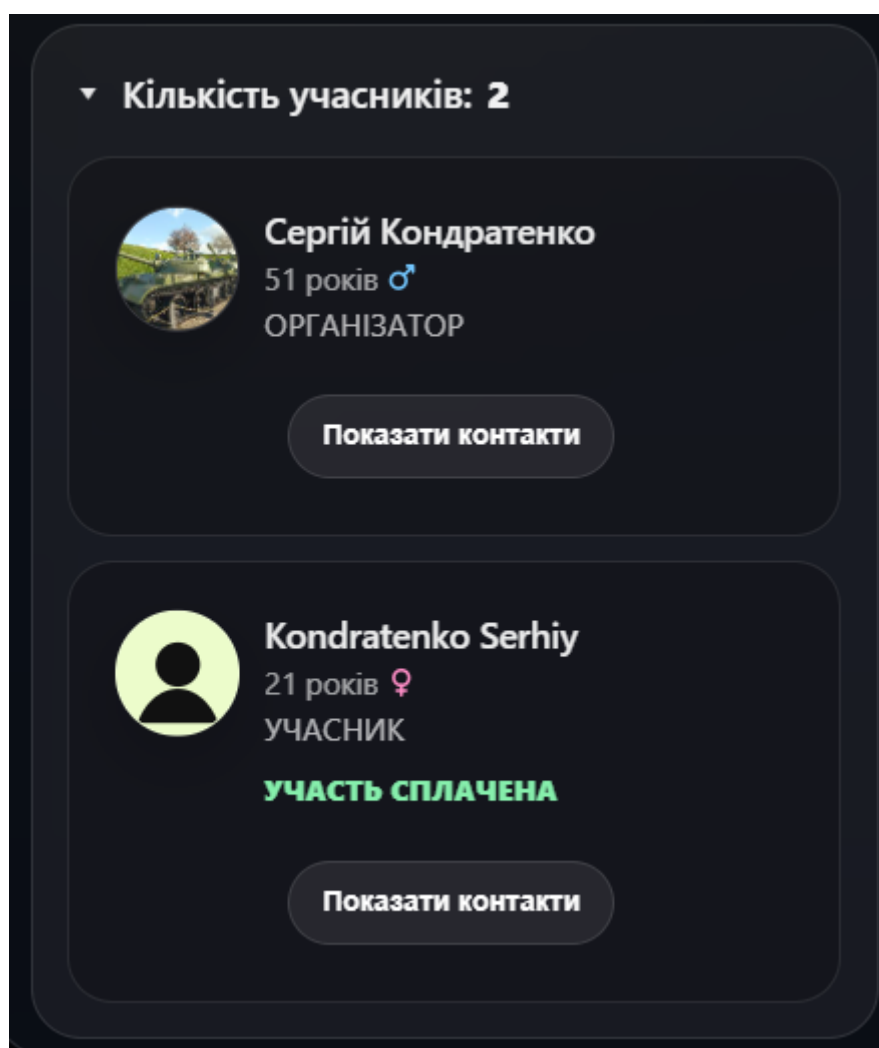


Рисунок 4.2 – Процес запрошення учасників та схвалення заявки

Наступний важливий сценарій перевіряє модуль фінансової безпеки платформи. Система вимагає від кожного учасника підтвердити свою присутність реальними грошима. У профілі туриста з'являється системне повідомлення про

необхідність сплати гарантійного депозиту. Користувач натискає кнопку оплати. Застосунок перенаправляє його на захищену сторінку еквайрингу від платіжного провайдера Stripe (рис. 4.3).

Активация подорожі

Оскільки ця подорож є платною, для її активації ви маєте внести гарантійний внесок на суму **500** грн. Цей внесок зараховується як оплата вашої власної участі в подорожі. Якщо сума гарантійного внеску перевищує базову вартість подорожі, різниця буде автоматично повернута вам після її успішного завершення.

> link | seriy.cub@gmail.com

VISA Visa Credit
.... 4242

Оплатити гарантійний внесок

Рисунок 4.3 – Сторінка оплати гарантійного внеску

Тут учасник вводить реквізити своєї тестової банківської картки. Серверна частина платформи GTPLANNER фізично не зберігає ці конфіденційні дані. Вона лише очікує на підтвердження операції від банку. Після успішної транзакції Stripe надсилає асинхронний Webhook-сигнал. Наш сервер миттєво обробляє його та оновлює статус платежу в базі даних. У загальному чаті поїздки одразу з'являється інформація про успішний внесок.

Третій сценарій ілюструє процес колективних витрат та звітності. Це головна функціональна перевага розробленої платформи. Організатор витрачає власні кошти на спільні потреби групи. Наприклад, він купує квитки на потяг або оплачує бронювання житла. Далі він відкриває застосунок і створює нову заявку на компенсацію. Користувач вказує витрачену суму і прикріплює фотографію фіскального чека (рис. 4.4).

СУМА, ГРН

7000

ОПИС ВИТРАТИ

Бронювання житла

+ Додати чеки

НАДІСЛАТИ ЗВІТ

500 грн

Test

СХВАЛЕНО

Доказ 1

stripe >

Рисунок 4.4 – Інтерфейс подання фінансового звіту

Після публікації цього звіту всі інші учасники отримують push-сповіщення. Це технічно реалізовано завдяки постійному WebSocket-з'єднанню. Учасники відкривають документ на своїх смартфонах і починають голосувати. Вони можуть погодити витрату, відхилити її або попросити додаткові пояснення в коментарях.

Програмний алгоритм у реальному часі підраховує зібрані голоси. Щойно система фіксує необхідну математичну більшість погоджень, відбувається автоматичний розрахунок. Бекенд надсилає внутрішню команду до системи Stripe на виплату компенсації. Гроші автоматично перераховуються з віртуального балансу подорожі на прив'язану картку організатора. Усі ці етапи чітко фіксуються в історії транзакцій. Тестування показало, що сценарій працює абсолютно стабільно. Гроші переказуються без технічних затримок та програмних помилок. Статуси чеків автоматично змінюються на погоджені. Такий алгоритм повністю виключає людський фактор при розрахунках і гарантує абсолютну прозорість спільного бюджету.

4.3 Постановка та результати обчислювальних експериментів.

Оцінка ефективності

Цей підрозділ описує результати обчислювальних експериментів. Головна мета полягала в оцінці ефективності та стабільності серверної частини системи. Для перевірки надійності архітектури проведено навантажувальне тестування. Найбільший інтерес викликала поведінка бази даних під час масового голосування за фінансові звіти. Спеціальний скрипт згенерував сотні одночасних запитів до бекенду. Інструмент Prisma успішно впорався з цим потоком транзакцій. Механізм пулу з'єднань запобіг блокуванню таблиць у PostgreSQL. Сервер обробляв запити без втрати даних і конфліктів запису. Середній час відповіді склав менше двохсот мілісекунд. Це підтверджує високу продуктивність створеного ядра системи під навантаженням.

Наступний етап експериментів стосувався перевірки безпеки та розмежування прав. Результати підтвердили повне виконання вимог технічного завдання. Програмні захисники (Guards) працюють абсолютно коректно. Звичайний учасник фізично не може змінити фінансові дедлайни або видалити маршрут. Сервер миттєво відхиляє такі несанкціоновані запити ще до звернення до бази даних. Окремо перевірено роботу підсистеми реального часу. Повідомлення в чаті та сповіщення про нові чеки доставляються миттєво. Клієнтським застосункам не потрібно постійно відправляти фонові запити для оновлення сторінки. Протокол WebSocket суттєво економить ресурси сервера та зберігає заряд батареї на мобільних пристроях туристів.

Власна розробка має беззаперечні переваги над існуючими універсальними рішеннями. Зазвичай люди планують спільні подорожі у звичайних месенджерах. Вони створюють групи у Telegram або Viber. Фінанси доводиться рахувати вручну в таблицях Excel або нотатках. Такий підхід постійно призводить до втрати важливої інформації в потоці текстових повідомлень. Учасники часто плутаються у розрахунках, що провокує міжособистісні конфлікти. Платформа GTPLANNER повністю вирішує цю проблему.

Система жорстко алгоритмізує всі організаційні процеси. Кожна витрата обов'язково проходить прозору процедуру колективного погодження. Автоматична інтеграція зі Stripe усуває потребу переказувати гроші на особисті банківські картки до завершення перевірки чеків. Розроблений програмний продукт ліквідує хаос у комунікації та робить процес підготовки до групових поїздок безпечним і максимально зручним.

Як перспектива подальшого масштабування платформи та вдосконалення вже існуючого клієнтського застосунку, побудованого на базі фреймворку React Native [28], розглядається розширення фінансового модуля. Зокрема, доцільним є дослідження новітніх архітектур мобільних платежів [29] для впровадження автоматичної конвертації валют, а також інтеграція технологій комп'ютерного зору (OCR) для автоматизованого розпізнавання даних із фотографій фіскальних чеків [30].

ВИСНОВКИ

1. На основі аналізу підходів, методів та алгоритмів розв’язання поставленої задачі обґрунтовано використання клієнт-серверної архітектури. Вона доповнена математичними моделями для чесного розподілу витрат та алгоритмами захисту транзакцій. Такий підхід дав змогу побудувати прозорий механізм. Тепер туристична група керує спільним бюджетом через просте колективне голосування.

2. На основі аналізу програмних засобів обґрунтовано використання платформи Node.js та фреймворку NestJS для розробки бекенду. Усі дані зберігаються в реляційній СУБД PostgreSQL, а за зв'язок із нею відповідає Prisma ORM. Безпеку реальних платежів гарантує провайдер Stripe. Водночас протокол WebSocket миттєво доставляє повідомлення в чати подорожей.

3. Розроблено програмну систему «GTPLANNER». Вона повністю бере на себе весь цикл планування групових поїздок. Застосунок самостійно реєструє учасників, збирає гарантійні внески та фіксує витрати. Це вирішує проблему плутанини в ручних таблицях та хаосу у звичайних месенджерах.

4. На основі тестування доведено коректність роботи розробленого програмного забезпечення. Експерименти з навантаженням та перевірка запитів через Postman підтвердили стабільність сервера. Бекенд без помилок обробляє платежі та одразу блокує будь-які спроби несанкціонованого доступу до чужих даних.

5. Напрямок подальшого вдосконалення системи є впровадження алгоритмів OCR. Це дозволить наявному мобільному застосунку автоматично зчитувати суми прямо з фотографій паперових чеків. Також доцільно додати механізм конвертації валют для закордонних мандрівок. Окрім цього, планується реалізувати офлайн-режим, щоб туристи мали доступ до своїх маршрутів навіть за відсутності інтернету.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Соммервілл І. Інженерія програмного забезпечення: підручник. 10-те вид. Київ: Навчальна книга – Богдан, 2019. 740 с.
2. Фаулер М. Архітектура корпоративних програмних додатків: навч. посіб. Київ: Діалектика, 2020. 544 с.
3. Martin R. C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Boston: Prentice Hall, 2018. 432 p.
4. Мельник А. О., Коваленко В. В. Аналіз сучасних архітектурних рішень для веб-застосунків. *Інформаційні технології*: зб. матеріалів III Міжнар. наук.-практ. конф. Львів: Львівська політехніка, 2023. С. 45–49.
5. Клеппман М. Розробка систем, що працюють з даними. Київ: Фабула, 2021. 608 с.
6. Гамма Е., Хелм Р., Джонсон Р., Вліссідес Дж. Патерни проектування об'єктно-орієнтованого програмного забезпечення. Київ: Діалектика, 2021. 368 с.
7. Casciaro M., Mammino L. Node.js Design Patterns. Birmingham: Packt Publishing, 2020. 664 p.
8. Кантелон М., Гартер М., Головайчук Т. Node.js на практиці. Київ: KIC, 2018. 380 с.
9. Node.js v20.x Documentation: веб-сайт. URL: <https://nodejs.org/docs/latest/api/> (дата звернення: 10.05.2026).
10. NestJS - A progressive Node.js framework: веб-сайт. URL: <https://nestjs.com/> (дата звернення: 12.05.2026).
11. Freeman A. Pro TypeScript: Application-Scale JavaScript Development. New York: Apress, 2019. 300 p.
12. TypeScript: Typed JavaScript at Any Scale: веб-сайт. URL: <https://www.typescriptlang.org/> (дата звернення: 14.05.2026).
13. Васильєв О. Програмування на JavaScript: навч. посіб. Тернопіль: Підручники і посібники, 2021. 416 с.

14. Obe R. O., Hsu L. S. PostgreSQL: Up and Running. Sebastopol: O'Reilly Media, 2017. 280 p.
15. Чіппендейл Д. Основи роботи з PostgreSQL. Київ: Олімп, 2022. 250 с.
16. PostgreSQL: The World's Most Advanced Open Source Relational Database: веб-сайт. URL: <https://www.postgresql.org/> (дата звернення: 16.05.2026).
17. Іванов О. С. Застосування ORM систем у високонавантажених проектах. *Сучасна інженерія*: зб. наук. праць. Київ: КПІ ім. Ігоря Сікорського, 2024. С. 112–117.
18. Prisma - Next-generation ORM for Node.js & TypeScript: веб-сайт. URL: <https://www.prisma.io/> (дата звернення: 17.05.2026).
19. JSON Web Token Introduction - jwt.io: веб-сайт. URL: <https://jwt.io/introduction> (дата звернення: 18.05.2026).
20. The OAuth 2.0 Authorization Framework: RFC 6749. URL: <https://datatracker.ietf.org/doc/html/rfc6749> (дата звернення: 20.05.2026).
21. OWASP Top Ten Web Application Security Risks: веб-сайт. URL: <https://owasp.org/www-project-top-ten/> (дата звернення: 21.05.2026).
22. Петренко І. І., Сидорчук М. М. Порівняльний аналіз платіжних систем для інтеграції у веб-додатки. *Економіка і програмування*: тези доп. Київ: НАУ, 2025. С. 88–91.
23. Stripe API Reference: веб-сайт. URL: <https://stripe.com/docs/api> (дата звернення: 22.05.2026).
24. WebSockets API - Web APIs | MDN: веб-сайт. URL: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API (дата звернення: 24.05.2026).
25. HTTP Status Codes - REST API Tutorial: веб-сайт. URL: <https://restfulapi.net/http-status-codes/> (дата звернення: 25.05.2026).
26. Куликов С. Тестування програмного забезпечення. Базовий курс. Київ: Епіфам, 2021. 312 с.
27. Postman Learning Center: веб-сайт. URL: <https://learning.postman.com/> (дата звернення: 26.05.2026).

28. React Native - A framework for building native apps using React: веб-сайт.
URL: <https://reactnative.dev/> (дата звернення: 27.05.2026).
29. Ткаченко В. М. Системи мобільних платежів та їх архітектура. *Інновації в IT*: зб. матеріалів конф. Харків: ХНУРЕ, 2025. С. 55–58.
30. Літвін В. В. Технології розпізнавання образів та комп'ютерного зору (OCR). Львів: Магнолія, 2023. 248 с.

ДОДАТОК А

РЕАЛІЗАЦІЯ ЛОГІКИ ОБРОБКИ ПЛАТЕЖІВ, УПРАВЛІННЯ МАРШРУТАМИ ТА ВЗАЄМОДІЇ З БАЗОЮ ДАНИХ СЕРВЕРНОЇ ЧАСТИНИ СИСТЕМИ GTPLANNER

Програмні засоби:

- мова програмування – TypeScript;
- серверний фреймворк – NestJS;
- інструмент об’єктно-реляційного відображення – Prisma ORM;
- система керування реляційними базами даних – PostgreSQL;
- інтеграція платіжного шлюзу – Stripe API.

```
// --- 1. Ініціалізація серверного ядра та налаштування Webhooks
(main.ts) ---
import { NestFactory } from '@nestjs/core';
import { AppModule } from '../app.module';
import { ValidationPipe } from '@nestjs/common';
import { NestExpressApplication } from '@nestjs/platform-express';

import { join } from 'path';
import * as express from 'express';

async function bootstrap() {
  const app = await
NestFactory.create<NestExpressApplication>(AppModule, {
  rawBody: true,
});

  app.useGlobalPipes(new ValidationPipe({ whitelist: true,
transform: true }));

  app.use(
```

```

    express.json({
      limit: '20mb',
      verify: (req: any, res, buf) => {
        if (req.originalUrl.startsWith('/stripe/webhook')) {
          req.rawBody = buf; // Збереження сирого тіла для
валідації підпису Stripe
        }
      },
    }),
  );

  app.use(express.urlencoded({ extended: true, limit: '20mb'
})));

  app.enableCors({
    origin: (origin, cb) => {
      if (!origin) return cb(null, true);
      if (origin.startsWith('http://localhost:')) return
cb(null, true);
      return cb(new Error('Not allowed by CORS'), false);
    },
    allowedHeaders: ['Content-Type', 'Authorization', 'stripe-
signature'],
    credentials: true,
  });

  app.useStaticAssets(join(__dirname, '..', 'uploads'), {
prefix: '/uploads' });
  await app.listen(process.env.PORT || 3000);
}
bootstrap();

// --- 2. Фрагмент декларативної схеми бази даних фінансів
(schema.prisma) ---
model TripExpense {

```

```

    id          String @id @default(uuid())
    tripId       String
    organizerId  String
    amount       Float
    description   String

    proofUrl     String?
    proofUrls    String[] @default([])

    status       ExpenseStatus @default(PENDING)
    attempt      Int            @default(1)

    createdAt    DateTime @default(now())
    updatedAt    DateTime @updatedAt

    trip         Trip           @relation(fields: [tripId], references:
[id], onDelete: Cascade)
    votes        ExpenseVote[]

    stripeTransferId String?
}

model ExpenseVote {
    id          String @id @default(uuid())
    expenseId   String
    userId      String
    vote        VoteType

    expense     TripExpense @relation(fields: [expenseId], references:
[id], onDelete: Cascade)
    user        User         @relation(fields: [userId], references:
[id], onDelete: Cascade)

    @@unique([expenseId, userId])
}

```

```

// --- 3. Інтеграція платіжного провайдера та переказ коштів
(stripe.service.ts) ---
import { Injectable, InternalServerErrorException } from
'@nestjs/common';
import Stripe from 'stripe';
import { PrismaService } from '../prisma.service';

@Injectable()
export class StripeService {
  private stripe: Stripe;

  constructor(private prisma: PrismaService) {
    if (!process.env.STRIPE_SECRET_KEY) {
      throw new Error('STRIPE_SECRET_KEY is not defined in
.env');
    }
    this.stripe = new Stripe(process.env.STRIPE_SECRET_KEY, {
      apiVersion: '2026-02-25.clover',
    });
  }

  async refundPayment(paymentIntentId: string, amountUah?:
number) {
    try {
      console.log(`[Stripe] Attempting refund for transaction
${paymentIntentId}...`);
      const refundOptions: Stripe.RefundCreateParams = {
        payment_intent: paymentIntentId,
      };

      if (amountUah && amountUah > 0) {
        refundOptions.amount = Math.round(amountUah * 100);
        console.log(`[Stripe] Initiating partial refund for
amount ${amountUah} UAH.`);

```

```

    } else {
      console.log(`[Stripe] Initiating full refund.`);
    }

    const refund = await
this.stripe.refunds.create(refundOptions);

    console.log(`[Stripe] Refund successful! Refund ID:
${refund.id}`);

    return refund.id;
  } catch (error: any) {
    console.error('[Stripe Refund Error]', error.message);
    throw new
InternalServerErrorException('STRIPE_REFUND_ERROR');
  }
}

// --- 4. Фонова автоматизація бізнес-процесів за розкладом
(trips.service.ts) ---
import { Injectable } from '@nestjs/common';
import { PrismaService } from '../prisma.service';
import { Cron, CronExpression } from '@nestjs/schedule';

@Injectable()
export class TripsService {
  constructor(private prisma: PrismaService) {}

  @Cron(CronExpression.EVERY_HOUR)
  async handleTripStatusChanges() {
    const now = new Date();

    const startingTrips = await this.prisma.trip.findMany({
      where: {
        status: 'PLANNED',

```

```

        startDate: { lte: now },
    },
    include: { members: true },
});

for (const trip of startingTrips) {
    // Автоматичне переведення подорожі в активний статус
    await this.prisma.trip.update({
        where: { id: trip.id },
        data: { status: 'ACTIVE' },
    });

    const userIds = trip.members
        .filter((m) => m.status === 'ACTIVE')
        .map((m) => m.userId);

    console.log(`[Cron] Trip ${trip.id} status changed to
ACTIVE`);
    }
}
}

```

ДОДАТОК Б

МАТЕРІАЛИ АПРОБАЦІЇ РЕЗУЛЬТАТІВ ДИПЛОМНОЇ РОБОТИ

Тези доповіді на VIII Міжнародній науково-практичній конференції «Information Technology for Education, Science, and Technics» (ITEST-2026, м. Черкаси)

ПРОЄКТУВАННЯ КЛІЄНТ-СЕРВЕРНОЇ СИСТЕМИ ДЛЯ ПЛАНУВАННЯ ГРУПОВИХ ПОДОРОЖЕЙ

DESIGNING A CLIENT-SERVER SYSTEM FOR GROUP TRIP PLANNING

Заліський М., Отрох С., Тараненко А., Кондратенко С. *Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського», Київ, Україна*

Анотація. Об'єктом дослідження є процес організації групових подорожей в умовах використання розрізнених цифрових інструментів. Предметом дослідження є архітектура та функціональні модулі клієнт-серверної системи для спільного планування поїздок. Завдання роботи: проаналізувати наявні проблеми координації учасників групових подорожей; визначити вимоги до єдиної платформи; розробити концепцію системи з інтерактивною картою, чатом, сповіщеннями та фінансовим модулем. Використано методи системного аналізу, об'єктно-орієнтованого проєктування та моделювання архітектури програмного забезпечення. Результатом є концепція платформи, що об'єднує планування маршруту, комунікацію та поетапні платежі з голосуванням учасників в одному продукті.

ПРОЄКТУВАННЯ КЛІЄНТ-СЕРВЕРНОЇ СИСТЕМИ ДЛЯ ПЛАНУВАННЯ ГРУПОВИХ ПОДороЖЕЙ

DESIGNING A CLIENT-SERVER SYSTEM FOR GROUP TRIP PLANNING

Заліський М.¹, Отрох С.², Тараненко А.², Кондратенко С.²

¹ Державний університет «Київський авіаційний інститут», Київ, Україна

² Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського», Київ, Україна

Анотація. Об'єктом дослідження є процес організації групових подорожей в умовах використання розрізнених цифрових інструментів. Предметом дослідження є архітектура та функціональні модулі клієнт-серверної системи для спільного планування поїздок. Завдання роботи: проаналізувати наявні проблеми координації учасників групових подорожей; визначити вимоги до єдиної платформи; розробити концепцію системи з інтерактивною картою, чатом, сповіщеннями та фінансовим модулем. Використано методи системного аналізу, об'єктно-орієнтованого проєктування та моделювання архітектури програмного забезпечення. Результатом є концепція платформи, що об'єднує планування маршруту, комунікацію та поетапні платежі з голосуванням учасників в одному продукті.

Ключові слова: Платформа Планування Групових Подорожей, Інтерактивна Карта, Онлайн-чат, Сповіщення, Голосування Учасників, Поетапні Платежі, Депозит Організатора, Архітектура «Клієнт-сервер».

Abstract. The object of this research is the process of organizing group trips when people use many different digital tools at the same time. The subject is the architecture and modules of a client-server system for group trip planning. The main tasks were: to find the problems that appear when people try to coordinate a group trip; to define what such platform needs to have; to design a system with interactive map, chat, notifications and financial module. Methods of system analysis, object-oriented design and software architecture modelling were used. The result is a platform concept that puts route planning, communication and step-by-step payments with participant voting all in one place.

Keywords: Group Trip Planning Platform, Client-server Architecture, Web Interface, Mobile Application, Interactive Map, Online Chat, Notifications, Participant Voting, Staged Payments, Organizer Deposit.

Рисунок Б1 – Тези доповіді ITEST-2026

Повнотекстова стаття, подана до публікації у науковій серії Springer Lecture Notes on Data Engineering and Communications Technologies (LNDECT) для індексації у Scopus

Concept and Development of an Integrated Client-server System for Group Travel Planning with Web and Mobile Interfaces and Coordinated Financial Workflows

Maksym Zaliskyi, Serhii Otrokh, Alina Taranenko, and Serhii Kondratenko
National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", Beresteiskyi Ave., 37, Kyiv, 03056, Ukraine

Abstract. Group trip organization usually uses many separate tools: chats, spreadsheets, maps, notes, and payment services. Because of this, the process becomes fragmented, slow, and sometimes risky for both organizer and participants. This paper presents the concept and implementation approach for an integrated client-server system for group travel planning with web and mobile interfaces. The proposed system combines trip creation, participant coordination, interactive map planning, real-time communication, notifications, shared media, and controlled financial workflows in one software product. A special focus is placed on a staged payment model in which participant funds are held in an external payment environment, while the organizer submits expense requests with proofs and receives money only after participant approval.

Concept and Development of an Integrated Client-server System for Group Travel Planning with Web and Mobile Interfaces and Coordinated Financial Workflows

Maksym Zalisky¹[0000-0002-1535-4384], Serhii Otrokh²[0000-0001-9008-0902], Alina Taranenko²[0009-0001-2472-0219] and Serhii Kondratenko²[0009-0000-3398-1288]

¹State University "Kyiv Aviation Institute", Kyiv, Ukraine

²National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute",
Beresteyskyi Ave., 37, Kyiv, 03056, Ukraine
taranenko.alina@l11.kpi.ua
kondratenko.serhiy@l11.kpi.ua

Abstract. Group trip organization usually uses many separate tools: chats, spreadsheets, maps, notes, and payment services. Because of this, the process becomes fragmented, slow, and sometimes risky for both organizer and participants. This paper presents the concept and implementation approach for an integrated client-server system for group travel planning with web and mobile interfaces. The proposed system combines trip creation, participant coordination, interactive map planning, real-time communication, notifications, shared media, and controlled financial workflows in one software product. A special focus is placed on a staged payment model in which participant funds are held in an external payment environment, while the organizer submits expense requests with proofs and receives money only after participant approval. The paper describes the system architecture, data model, financial logic, client-server interaction, development pipeline, and testing approach. The practical value of the work is a prototype-oriented concept that can reduce coordination mistakes, improve transparency, and increase trust in group travel planning.

Keywords: Group Travel Planning System, Client-server Architecture, Web Application, Mobile Application, Real-time Chat, Interactive Map, Staged Payments, Participant Voting, Notifications, Escrow-like Funds Holding.

Рисунок Б2 – Стаття, подана до публікації у LNDECT (Scopus)