Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Інфокомунікацій
(повна назва)

Кафедра Інформаційно-вимірювальних технологій
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Використання сучасних інструментів тестування та забезпечення якості при створенні вебзастосунку
(тема)

Виконав:
здобувач II року навчання,
групи ЗЯм-23-1

Портянніков М.В.
(прізвище, ініціали)

Спеціальність 175 – Інформаційно-вимірювальні технології
(код і повна назва спеціальності)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Забезпечення якості
(повна назва освітньої програми)

Керівник проф. Скляров В.В.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ Захаров І.П.
(підпис) (прізвище, ініціали)

2025 р.

Харківський національний університет радіоелектроніки

Факультет Інфокомунікацій
(повна назва)

Кафедра Інформаційно-вимірювальних технологій

Рівень вищої освіти другий (магістерський)

Спеціальність 175 – Інформаційно-вимірювальні технології
(код і повна назва)

Тип програми освітньо-професійна

Освітня програма Забезпечення якості
(повна назва)

ЗАТВЕРДЖУЮ

Зав. кафедри _____

"_____"_____2025 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Портяннікову Максиму Васильовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Використання сучасних інструментів тестування та забезпечення якості при створенні вебзастосунку
затверджена наказом по університету від "12" листопада 2024р. № 1202 Ст

2. Термін подання здобувачем роботи до екзаменаційної комісії 10 січня 2025 р.

3. Вихідні дані до роботи Розробка веб-застосунку для забезпечення додатку необхідним захистом від різних типів атак. Визначення та створення застосунку, що буде протестовано з використання сторонніх застосунків та завантажених пакетів NodeJS. Програмне забезпечення, що було використано: ОС Microsoft Windows 11, MongoDB Compass, Postman, Visual Studio Code, програмна платформа Node. Технічне забезпечення: IBM-сумісний ПК з ЦП Intel Core i3 та вище.

4. Перелік питань, що потрібно опрацювати в роботі
4.1 Аналіз предметної області.
4.2 Визначення вимог забезпечення якості розробкиі тестування.
4.3 Опис рішень та використання їх на практиці.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) 5.1 Життєвий цикл програмного забезпечення 5.2 Функціональна модель IDEF0 5.3 Декомпозиція функціональної моделі 5.4 Декомпозиція бізнес-процесу «Аутентифікація та авторизація» 5.5 Діаграма дерева вузлів 5.6 Графічне представлення MVC 5.7 Діаграма класів 5.8 Діаграма варіантів використання 5.9 Послідовність дій прецеденту «Бронювання турів» 5.10 Діаграма відносин між моделями 5.11 Успішна відповідь розробленого API. 5.12 Результат тестування ресурсу Tour. 5.13 Результат тестування ресурсу User 5.14 Інтерфейс головної сторінки.

КАЛЕНДАРНИЙ ПЛАН

Пор. №	Назва етапів атестаційної роботи	Термін виконання етапів роботи	Примітка
1.	Отримання завдання кваліфікаційної роботи	25.11.2024	Виконано
2.	Аналіз предметної області та літератури	25.11.2024-	Виконано
		30.11.2024	
3.	Вибір мови програмування та системи управління	01.12.2024	Виконано
	базами даних для розробки функціоналу додатка		
4.	Розробка вимог до інформаційної системи	02.12.2024-	Виконано
		05.12.2024	
5.	Розробка програмного коду інформаційної системи	06.12.2024-	Виконано
		15.12.2024	
7.	Забезпечення якості інформаційної системи	16.12.2024-	Виконано
		20.12.2024	
8	Тестування розробленого API системи	21.12.2024-	
		31.12.2024	
9.	Оформлення пояснювальної записки та	01.01.2025-	Виконано
	документація програмного коду	05.01.2025	
10.	Оформлення графічної частини презентаційних	за 4 дні	Виконано
	матеріалів, комп'ютерних матеріалів для захисту		
	атестаційної роботи		
11.	Представлення на рецензування	за 3 дні	Виконано
	Представлення атестаційної роботи в ДЕК	за 2 дні	Виконано

Дата видачі завдання 25.11.2024

Студент _____ Портянніков М.В.
(підпис)

Керівник роботи _____ проф. Скляров В.В.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи містить:95 с., 3 табл., 20 рис., 1 додаток, 19 джерел.

NODEJS, POSTMAN, МАРШРУТ, МАРШРУТИЗАТОР, HELMET, CORS, HTTP, МЕТОД, БАЗА ДАНИХ, СЕРВЕРНИЙ ЗАСТОСУНОК, ЗАБЕЗПЕЧЕННЯ ЯКОСТІ, ТЕСТУВАННЯ, ОБМЕЖЕННЯ ДОСТУПУ, СТВОРЕННЯ API, ДОКУМЕНТАЦІЯ, СЕРЕДОВИЩЕ, МОДУЛЬ

Об’єктом дослідження є проєктування вебзастосунку для впровадження методів забезпечення якості з проведенням тестування всіх кінцевих маршрутів для кожного із створених ресурсів інформаційної системи. Що буде включати інструменти автоматизації, таких як використання додаткових пакетів Node.js, що дозволять не тільки зменшити навантаження, але і надати захист системи.

Предметом дослідження є проведення аналізу і провадження різних методів і інструментів, що забезпечать застосунок ефективністю та надійністю від можливих атак на систему.

Методи дослідження включають в себе аналіз і порівняння інструментів тестування та забезпечення якості, оцінку ефективного моделювання даних, що дозволить досягти максимальної швидкості застосунку.

Результатом кваліфікаційної роботи є створена серверна частина застосунку, для якої проведено тестування всіх маршрутів, виконано моделювання даних з найефективнішим методом побудови та реалізація веб-документації для створених API.

ABSTRACT

The explanatory note on the qualification work contains: 95 pages, 3 tables, 20 figures, 1 appendices, 19 sources.

NODEJS, POSTMAN, ROUTE, ROUTER, HELMET, CORS, HTTP, METHOD, DATABASE, SERVER, QUALITY ASSURANCE, TESTING, ACCESS RESTRICTIONS, API CREATION, DOCUMENTATION, ENVIRONMENT, MODULE

The object of the research is the design of a web application for the implementation of quality assurance methods with testing of all final routes for each of the created resources of the information system. Which will include automation tools, such as the use of additional Node.js packages, which will allow not only to reduce the load, but also to provide system protection.

The subject of the research is the analysis and implementation of various methods and tools that will ensure the application's effectiveness and reliability against possible attacks on the system.

Research methods will include analysis and comparison of testing and quality assurance tools, evaluation of effective data modeling that will allow for maximum application speed.

The result of the qualification work is the created server part of the application, for which all routes will be tested, data modeling will be performed with the most effective construction method, and web documentation will be implemented for the created APIs.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

БД – база даних

ІС – інформаційна система

ПЗ – програмне забезпечення

СУБД – система управління базами даних

API – Application Programming Interface

JS – JavaScript

QA – Quality Assurance

QC – Quality Control

Npm – Node Package Manager

SDLC – Software Development Lifecycle

UML – Unified Modeling Language

XSS – Cross-Site Scripting

ВСТУП

Використання сучасних інструментів тестування та забезпечення якості інформаційної системи дозволяє створити ефективний та швидкий додаток. Важливим етапом є визначення архітектури проєктування, для якої обирається мова програмування, програмне забезпечення, додаткові пакети та технології, що будуть спрощувати розробку та надання необхідної якості додатку.

Обрані методи та методології для забезпечення якості та тестування додатку є важливим кроком для створення надійного, безпечного та функціонального продукту, що буде відповідати очікування користувачів та вимогам бізнесу [1-5]. Використання верифікації та валідації дозволяє гарантувати, що всі етапи розробки проходять відповідно поставленим вимогам, щодо надання захисту і якості додатку.

Для проєктування якісного веб-застосунку було обрано архітектуру MVC (Model – View – Controller), завдяки структурованому підходу до розробки, розподіляючи застосунок на три основних компоненти, кожен з яких має свою певну роль. Саме використання такого підходу дозволяє надати порядок у кодовій базі та покращує процес розробки, тестування та підтримки застосунку.

Таким чином, використовуючи сучасні технології та методології для побудови якісного вебзастосунку, було досягнуто поставленої мети роботи. Створивши та протестувавши необхідний функціонал додатку обов’язковим кроком буде реалізація документації для створених кінцевих точок маршрутів.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз предметної області. Створення інформаційної системи

У сучасному світі, коли цифрові технології швидко розвиваються, вебзастосунки стають одним із ключових компонентів для бізнесу, організацій та окремих користувачів. Домінуючий вплив на успіх діяльності здійснює розроблений вебзастосунок, що було реалізовано з використанням сучасних методів забезпечення якості системи та проведення тестування.

Вебзастосунки працюють у середовищі інтернету, що дозволяє отримувати доступ до інформаційних систем, з наданням стабільної роботи та високої продуктивності з метою забезпечення конкурентоспроможності серед аналогічних системи, де розроблений вебзастосунок повинен відповідати всім поставленим критеріям для забезпечення якості та надійності даних користувачів.

Основними вимогами до інформаційної системи, що визначаються на етапі планування та аналізу кінцевого результату, є:

– функціональність – вебзастосунок має повністю забезпечувати реалізацію заявленого функціоналу;

– продуктивність – розробити та оптимізувати застосунок з використанням правильно розробленої архітектури додатку та моделювання даних, що надають можливість швидко оброблювати запити до серверу;

– захист даних – інформаційна система повинна бути захищена від зовнішніх атак, витоків інформації та інших кіберзагроз з використанням сучасних методів;

– ефективне використання – у вебдодатку повинна бути присутня зручність та легкість взаємодії з системою, що буде оптимізована на основі проведеного тестування;

– масштабованість – необхідно структурувати весь розроблений функціонал системи, щоб він був придатній до збільшення та розширення функціональності.

Саме аналіз та визначення всіх ключових аспектів предметної області для розробки вебдодатку є невід'ємною частиною процесу розробки. Використання сучасних інструментів, що автоматизують процеси тестування та захисту застосунку, дозволяють підвищити ефективність розробки, забезпечивши високу якість кінцевого продукту з мінімізованою кількістю помилок.

1.2 Забезпечення якості як основний аспект під час реалізації

Надання якості інформаційній системі є важливим елементом на будь-якій стадії розробки вебзастосунку, адже від цього залежить як кінцевий результат, так і його сприйняття користувачами. На всіх етапах розробки, починаючи від планування та проєктування до впровадження та підтримки, саме забезпечення якості має найбільший пріоритет для всіх учасників команди [6]. На рисунку 1.1 представлено графічне зображення життєвого циклу програмного забезпечення.

Рисунок 1.1 – Життєвий цикл програмного забезпечення

Важливість забезпечення якості на різних етапах життєвого циклу програмного забезпечення:

– етап планування. Цей етап важливий через побудову фундаменту для майбутньої інформаційної системи, визначення вимог, вибір архітектури та визначення бізнес-логіки. Стадія планування повинна враховувати можливі ризики, що будуть пов’язані з якістю та безпосередньо можливості оцінити масштабованість. Створення якості на цьому етапі полягає у ретельному аналізі вимог та чіткому визначенні критеріїв для успіху бізнесу;

– етап проєктування та розробки. Під час розробки інформаційної системи присутні технічні виклики, що пов’язані з вибором технологій розробки, архітектурою та компонентами. Успішне застосування інструментів для забезпечення якості на етапі розробки дозволить виявити та усунути помилки до їх реального виникнення під час роботи додатку. Використання інструментів для аналізу статичного коду, такі як SonarQube та ESLint дозволять встановлювати кращі стандарти для написання коду;

– етап тестування. Тестування – це ключовий етап забезпечення якості, коли система проходить через різні перевірки на відповідність вимог, швидкодію, стабільність та безпеку. Тестування може бути як ручним, так і автоматизованим, залежно від типу вебдодатку та вимог. Забезпечення якості на етапі тестування дозволяє охопити велику кількість різноманітних перевірок;

– етап впровадження. Навіть після успішного тестування на етапі, коли продукт вже готовий до виходу на ринок, важливо бути впевненим, що на даному етапі не виникають помилки. Наприклад, можна використовувати інструменти для автоматизованого тестування перед кожним випуском нової версії;

– етап підтримки та розвитку. Після впровадження етап забезпечення якості має продовження. Система повинна весь час залишатися стабільною під час експлуатації та масштабування. Необхідний постійний моніторинг

продуктивності, логування, створення регулярних оновлень та оптимізації функцій.

Підсумовуючи, можна сказати, що надання та забезпечення якості є ключовим аспектом для будь-якої інформаційної системи. Саме використання сучасних інструментів тестування та забезпечення якості дозволять створити продуктивний та безпечний застосунок як для користувачів, так і для бізнесу.

1.3 Сутність тестування інформаційної системи та основні методи забезпечення якості

Тестування інформаційної системи є ключовим етапом у розробці та впровадженні ПЗ, що акцентує увагу на усуненні помилок, недоліків чи невідповідності поставленим вимогам. Це дозволить оцінити якість роботи системи, забезпечивши її надійністю, безпекою та стабільністю під час експлуатації програмного забезпечення користувачами та бізнесом.

Сутність тестування лежить в основі систематичного аналізу роботи інформаційної системи з метою виявленням потенційних помилок в роботі створених API, а також в перевірці фактичних результатів та очікуваних.

Основні методи забезпечення якості, що використовуються під час розробки ПЗ:

– Quality Assurance (Забезпечення якості). QA – це технологічний підхід, який перевіряє, чи є процес розробки продукту правильним та відповідає певним стандартам та гарантує, що процес поставлений правильно і дає передбачуваний результат;

– Quality Control (Контроль якості). QC – це продуктовий підхід. Перевіряє, чи відповідає розроблений продукт всім зазначеним вимогам. Включає в себе різні типи тестування, такі як функціональне тестування, дослідницьке тестування, тестування продуктивності, usability тестування та інші. Процес QC повинен передбачати знаходження помилок в розробленому продукті, з метою їх подальшого виправлення.

Побудуємо порівняльну таблицю для основних методів забезпечення якості. Різниця між Quality Assurance, Quality Control та тестування подана в таблиці 1.1.

Таблиця 1.1 – Основні методи забезпечення якості

Quality Assurance	Quality Control	Тестування
Комплекс заходів, який охоплює всі технологічні аспекти на всіх етапах розробки, випуску та введення в експлуатацію програмних систем для забезпечення необхідного рівня якості програмного продукту	Процес контролю відповідності розроблювальної системи пред'явленим до неї вимогам	Процес, який відповідає безпосередньо за складання та проходження тест-кейсів, знаходження і локалізацію дефектів тощо
Фокус в більшій мірі на процеси і засоби, ніж на безпосередньо виконання тестування системи	Фокус на виконання тестування шляхом виконання програми з метою визначення дефектів з використанням затверджених процесів і засобів	Фокус на виконання тестування як такого
Процесно-орієнтованих підхід	Продуктно-орієнтованих підхід	Продуктно-орієнтованих підхід
Превентивні (випереджувальні заходи)	Коригуючий процес	Превентивний процес
Підмножина процесів Software Test Life Cycle – циклу тестування ПЗ	Підмножина процесів QA	Підмножина процесів QC

1.4 Найкраща практика для створення безпеки

Створення безпеки в вебзастосунку є критичним аспектом, що вимагає використання комплексного підходу. Використання інструментів тестування та забезпечення якості системи дозволить запобігти вразливостям та можливим ризикам в системі. В ході роботи для створення вебзастосунку на платформі Node.js та надання безпеки використовувались такі інструменти, що представляють собою найкращі практики.

Компрометація бази даних:

– шифрування паролів із використанням хешування. Використовували пакет bcrypt для хешування паролів, щоб надати додатковий рівень безпеки;

– шифрування токенів для скидання пароля. Використовували алгоритм SHA-256 для надійного шифрування токенів.

Захист від атак грубої сили (Brute Force):

– використання bcrypt для уповільнення запитів на вхід. Пакет забезпечить систему уповільненням процесу розшифровки пароля, що ускладнює проведення атак [11];

– лімітування запитів. Використання бібліотеки express-rate-limit дозволить обмежити кількість запитів до серверу за певний період часу. Необхідно для зменшення ризику атак грубої сили;

– максимальна кількість спроб входу. Необхідно встановити обмеження на кількість невдалих спроб входу до системи.

Захист від атак типу міжсайтового скриптингу (XSS – Cross-Site Scripting):

– зберігання JWT у HTTPOnly cookies. Це запобігає доступу до токенів через JavaScript знижуючи ризик XSS атак;

– очищення даних користувачів. Необхідно завжди очищувати і валідувати дані, що надходять від клієнтської частини, щоб уникнути введення шкідливого коду;

– встановлення спеціальних HTTP-заголовків. Необхідно використовувати пакет helmet для встановлення додаткових заголовків, що підвищують безпеку системи.

Захист від атак типу відмова в обслуговуванні (DoS):

– лімітування запитів. Використання бібліотеки express-rate-limit дозволить обмежити кількість запитів до серверу за певний період часу;

– обмеження розміру тіла запиту. Необхідно використовувати пакет body-parser для контролю розміру даних;

– небезпечні регулярні вирази. Необхідно використовувати прості регулярні вирази, їх використання не буде навантажувати систему.

Захист від ін’єкцій у NoSQL-запитах:

– використання mongoose для MongoDB. Використання SchemaTypes дозволить захистити запити;

– очищення вхідних даних. Необхідно перевіряти і очищувати дані користувачів перед виконанням запитів.

Інші практики, що необхідно використовувати:

– використання HTTPS. Необхідно надавати шифрування даних між клієнтською і серверною частиною через HTTPS;

– створення випадкових токенів скидання пароля з датами їх закінчення. Дозволить завжди отримувати нові токени без можливості використання старих;

– скасування доступу після оновлення паролю. Гарантує, що скомпроментовані токени будуть недійсними одразу після оновлення паролю;

– не зберігайте конфіденційні дані в Git;

– не надсилайте конфіденційну інформацію користувачам. Необхідно обмежувати дані, що надсилаються;

– захист від підробки міжсайтових запитів (CSRF). Необхідне використання csrf пакета для запобігання атак;

– підтвердження через електронну пошту під час створення нового профілю. Дозволить перевірити дійсність електронної пошти;

– реалізація двофакторної автентифікації. Додатково підвищує безпеку профілів користувачів;

– очищення параметрів. Необхідно додати обробку для очищення надлишкових параметрів під час створення запитів до сервера.

Використання сучасних інструментів для забезпечення якості, проведення тестування, а також інших додаткових засобів дозволить створити надійну та ефективну систему як з точки зору продуктивності, так і захисту даних.

1.5 Важливість документації для кінцевого продукту

Документація є невід'ємною частиною процесу розробки веб-застосунку. Перегляд документацій та додаткових матеріалів дозволяє швидко ознайомитись з цільовою системою. Саме вона відіграє важливу роль у створені самого проєкту, через її чіткість і прозорість процесу розробки, можливості проводити ефективне тестування на основі перегляду моделей та можливість постійно проводити її актуалізацію.

Розглянемо всі аспекти, що заносяться в документацію для кінцевого продукту:

– чіткість і прозорість процесу розробки. Розробка документації дозволить учасникам проєкту розуміти функціональність системи, архітектуру та всі процеси. Перегляд документації забезпечить легку координацію в команді та дозволить легко знаходити можливі помилки та виправити їх на ранніх стадіях;

– ефективне тестування. Створення опису для всіх модулів та функцій у вигляді тестових сценаріїв може стати важливим критерієм для проведення тестування. Документація полегшить роботу, надавши розуміння як слід перевіряти, щоб забезпечити високу надійність продукту;

– легкість підтримки і розширення. Розширення функціоналу і виправлення існуючого – це базова практика під час розробки систем. Отже, документація полегшує процес підтримки та оновлення продукту, оскільки нові члени команди зможуть швидко зорієнтуватися в коді, завдяки детальному опису архітектури та створених API;

– забезпечення відповідності стандартам. Інструменти забезпечення якості і тестування в будь-якому випадку присутні під час розробки інформаційної системи, отже, документація дозволить перевірити, чи відповідає система всім стандартам та вимогам, що були впровадженні в проєкті;

– захист даних і безпека. Створення документації дозволить описати всі заходи безпеки, що були прийняті, включаючи шифрування, керування автентифікацією та авторизацією, а також логікою збереження даних.

Отже, аналіз документації дозволить швидко адаптуватися в коді та описі проєкту, що стане фундаментальним інструментом для розробки, тестування та забезпечення якості веб-застосунку. Її використання не лише полегшить проєктування але й забезпечує гнучкість у процесі його розвитку, що стає важливим для успішної роботи розробленого продукту на ринку.

1.6 Постановка задачі

Визначення вимог та завдань для розробки, тестування, забезпечення якості і безпеки інформаційної системи потребує детального аналізу бізнесу та представлення кінцевого результату. Поставлені задачі необхідно в повному обсязі виконати для досягнення мети.

Для досягнення мети кваліфікаційної роботи, визначимо задачі, що необхідно виконати:

– визначитись з набором технологій, що будуть використовуватись для розробки вебзастосунку;

– проаналізувати та обрати архітектуру;

– розробити інтерфейс, що буде взаємодіяти та надавати необхідний результат;

– провести тестування розроблених API;

– надати правильну структуру проєкту відповідно обраної архітектури;

– забезпечити вебзастосунок захистом від різних типів атак;

– розробити документацію застосунку;

– розробити динамічний інтерфейс користувача.

В ході роботи, поставлені задачі будуть поділені на більш менші частини, що дозволить детально проаналізувати вимоги користувачів і бізнесу. Саме дотримання вимог дозволить створити ефективну та швидку систему.

2 ВИЗНАЧЕННЯ ВИМОГ ЗАБЕЗПЕЧЕННЯ ЯКОСТІ РОЗРОБКИ І ТЕСТУВАННЯ

2.1 Основні етапи життєвого циклу програмного забезпечення (SDLC)

SDLC (Software Development Lifecycle) – це процес, що використовується для під час проєктування програмного забезпечення, розробки, тестування та забезпечення якістю розроблювальної інформаційної системи [18].

SDLC націлений на виробництво ПЗ, яке відповідає вимогам та очікуванням користувачів та найкоротші терміни завершує роботи і оцінює витрати, що були використанні на проєктування інформаційної системи.

Розглянемо основі етапи життєвого циклу розробки програмного забезпечення, що подані на рисунку 2.1. Вона складається з шести етапів, що формують цикл і забезпечують послідовний процес створення, впровадження та підтримки ПЗ:

– планування. На цьому етапі визначаються цілі проєкту, основні вимоги та бюджет. Створюється загальний план робіт, розподіляються завдання та ресурси. Це фундаментальний етап, що задає напрямок подальшої розробки;

– аналіз. Проводиться детальне дослідження та визначення вимог до програмного забезпечення. Учасники проєкту збирають інформацію про потреби кінцевих користувачів і замовника. Це допомагає уникнути неточностей і закласти правильні вимоги для наступних етапів;

– проєктування. Цей етап включає в себе розробку архітектури системи та дизайн системи. Спроєктований дизайн буде використаний як основа для побудови інтерфейсу користувача та визначення взаємодій компонентів між собою і структурою БД;

– реалізація. Відбувається написання коду для програмного забезпечення відповідно до створеного дизайну. Розробники створюють

функціональні компоненти, інтегрують їх та готують до тестування. Це етап активної роботи над програмним продуктом;

– тестування та інтеграція. Програмне забезпечення тестується на наявність помилок та перевіряється на відповідність вимогам. Інтегруються всі модулі в єдину систему, після чого виконуються різні види тестування, такі як функціональне, інтеграційне, приймальне тощо;

– підтримка. Розгорнувши програмне забезпечення одразу буде проведено перехід до етапу підтримки. Даний етап включає виправлення знайдений помилок, оновлення версій системи, при необхідності проводити рефакторинг коду та вдосконалення функціоналу у відповідь на змінення вимог та потреб користувачів.

Цей цикл може повторюватися, коли потрібно оновити або вдосконалити програмне забезпечення, роблячи процес постійно гнучким і адаптивним до нових змін.

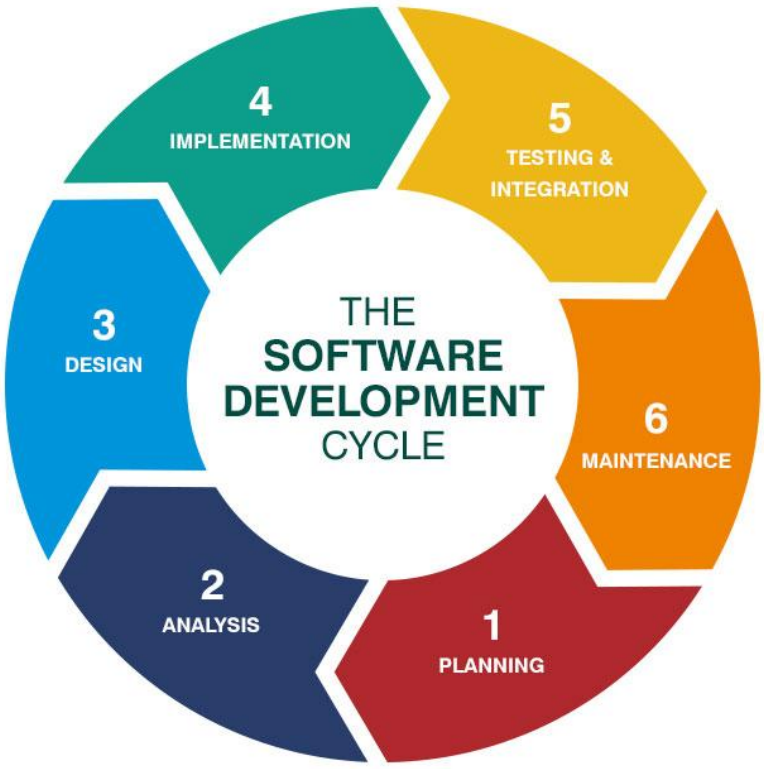

Рисунок 2.1 – Етапи життєвого циклу програмного забезпечення

2.2 Системне проєктування вебзастосунку. Розробка функціональної моделі IDEF0

Системне проєктування – це процес створення та розробки складних інформаційних систем, що включає в себе різні компоненти та забезпечує ефективну взаємодію для досягнення мети [3]. Під час системного проєктування ми можемо визначити оптимальну архітектуру, створити покроковий план реалізації системи та провести аналіз різноманітних аспектів, таких як, майбутня масштабованість, гнучкість, безпека даних, послідовність дій та багато іншого [6].

Починаючи етап проєктування функціональних і нефункціональних вимог до вебзастосунку було поставлено запитання, яка мета проєктування функціональної моделі? Проаналізувавши сутність IDEF0, можна сказати, що її основною метою є створення структурованого опису бізнес-процесів з можливістю їх деталізації та зменшення рівня абстракції.

Для розробки моделі було сформовано вхідні, вихідні дані, механізми та управління, що тісно взаємодіють із системою [8]. Нижче наведено всі компоненти, що використовуються системою та їх короткий опис.

Вхідні дані:

– забронювати. Користувачеві надається можливість оформити бронювання на обраний тур;

– дані користувача. Надання відповідних даних для проведення реєстрації та авторизації в системі;

– набір турів. Список доступний турів для бронювання;

– тур. Отримання детальної інформації про переглянутий тур;

– створення туру. Адміністратор або головний гід має можливість створювати нові тури;

– написання відгуку. Тільки клієнт має можливість створювати один відгук для одного туру.

Вихідні дані:

– заброньований тур. Автоматичне оновлення інформації, щодо забронюваного туру та відображення оновлень на веб-сайті;

– користувач. Після реєстрації в системі необхідно зберегти інформацію нового профілю;

– створений відгук. Оновлення даних туру та відображення нового відгуку на сторінці туру;

– оновлені дані. Оновлена інформація може стосуватися як користувача так і тура;

– оновлений список групи. Оновлення даних про тур. Зміна кількості вільних місць на певну дату;

– отримані дані. Користувач переходить на сторінку турів і автоматично створюється запит на отримання всіх даних після чого виконання запиту повертає набір даних.

Механізми:

– клієнт. Особа, що оформлює бронювання обраного туру на веб-сторінці;

– гід. Особа, що проводить експедицію для клієнтів по заданим локаціям туру;

– ПЗ. Функціональність системи для обробки всіх запитів та коректної роботи вебзастосунку;

– адміністратор. Отримує доступ до всієї інформації в системі та можливостям її оновлення.

Управління:

– правила користування. В даному документі виставлені вимоги користування системою;

– нормативні документи. Документ, що встановлює правила, настанови чи характеристики щодо діяльності або її результатів.

Підсумовуючи створення першого рівня функціональної моделі, можна сказати, що нам необхідно визначити всі важливі компоненти, які взаємодіють із системою. Розроблену діаграму подано на рисунку 2.2.

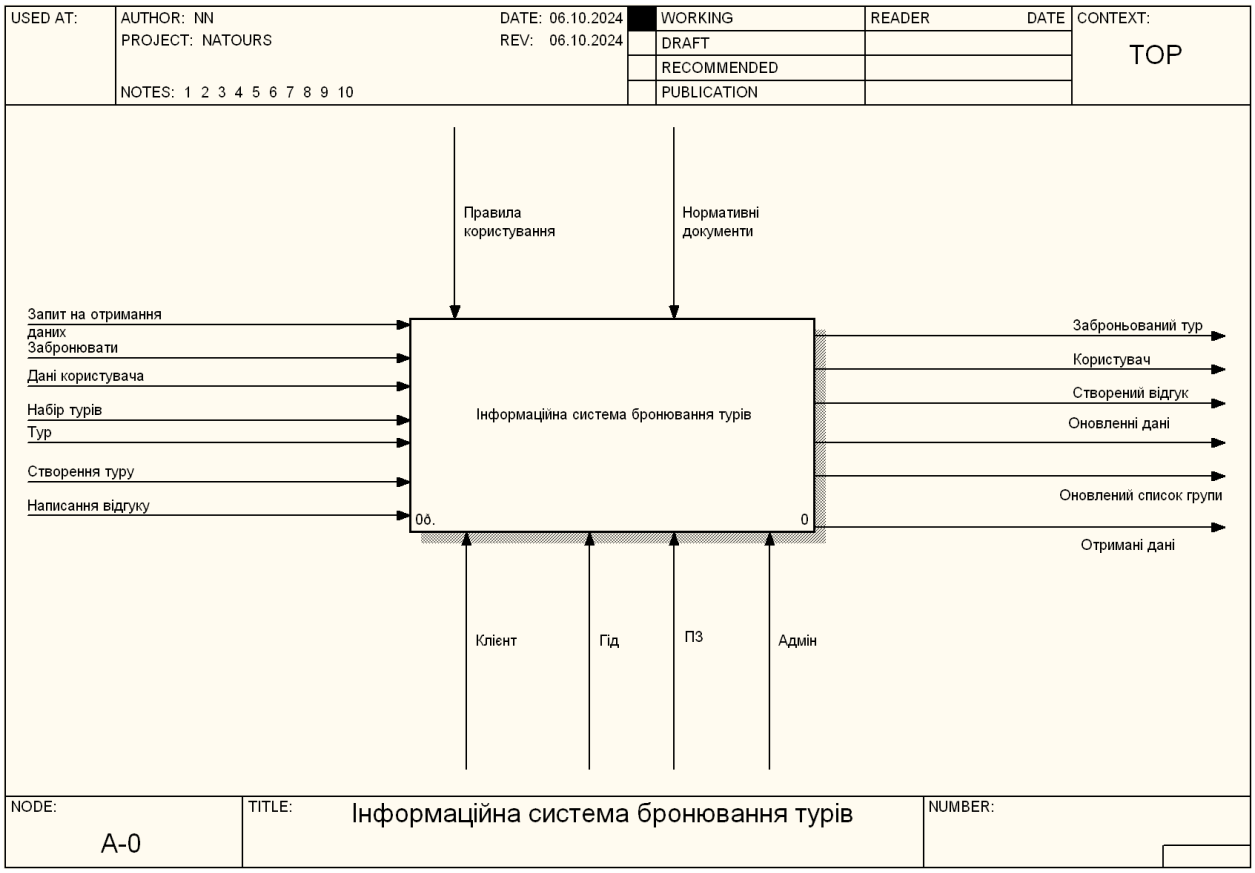

Рисунок 2.2 – Функціональна модель IDEF0

Для більшого розуміння роботи інформаційної системи було прийняте рішення зробити декомпозицію функціональної моделі. На діаграмі декомпозиції, яка подана на рисунку 2.3, можна побачити 3 головних бізнес-функції системи:

– аутентифікація та авторизація. Одна з найголовніших бізнес-функцій в будь-якій системі, що забезпечує контроль доступу. Мета аутентифікації полягає в підтвердженні особи через перевірку облікових даних. Наприклад, пароль та електронна адреса. Авторизація, у свою чергу, надає доступ до ресурсів після успішної аутентифікації користувача, тобто дозволяє системі контролювати, які дії доступні користувачеві. Розробка цього функціоналу є ключовим критерієм для сучасних систем, оскільки вони забезпечують захист від несанкціонованого доступу, зберігаючи конфіденційність та цілісність даних;

– вибір та бронювання турів. Для конкретно обраної системи, вибір і бронювання турів – це важливі процеси, які лежать в основі туристичних платформ, забезпечуючи користувача легшим та швидким способом бронювання місця в експедиції. Вибір туру включає в себе перегляд та порівняння схожих подорожей. Користувачеві надається можливість фільтрувати варіанти відповідно до своїх критеріїв, що дозволить обрати кращу подорож. Бронювання туру передбачає резервування обраної подорожі. Цей процес включає надання особистих даних та здійснення платежу. Без даного функціоналу туристичні платформи не будуть повноцінними, оскільки не зможуть забезпечити ефективне планування та вибір турів;

– взаємодія з наборами даних. Це важливий процес, що забезпечує отримання, оновлення та видалення даних, виконуючи деякі запити в автоматизованому режимі. Процес взаємодії починається з доступу до набору даних, що зберігаються в базі даних, диску або в хмарних сховищах. Користувачі в більшості випадків можуть тільки отримувати дані без можливості їх оновлення чи видалення. Наприклад, фільтрація турів за певними критеріями. Можливості адміністратора набагато більші, аніж користувачів. Основна задача адміністратора і гіда полягає в створенні, оновленні та видаленні ресурсів. Отже, ефективна взаємодія за наборами даних є основою в багатьох сучасних інформаційних системах і бізнес-процесах, що дозволяє створювати повноцінну інформацію для аналітики, прогнозування та оптимізації процесів.

Розробка функціональної моделі полягає у створенні структурованого опису системи або процесу з метою чіткого визначення його основних функцій і взаємодій між компонентами [4]. Така модель допомагає візуалізувати роботу системи, показати послідовність виконання завдань і оптимізувати робочі процеси. Використання функціональної моделі спрощує комунікацію між розробниками та зацікавленими сторонами, дозволяючи краще розуміти вимоги до системи.

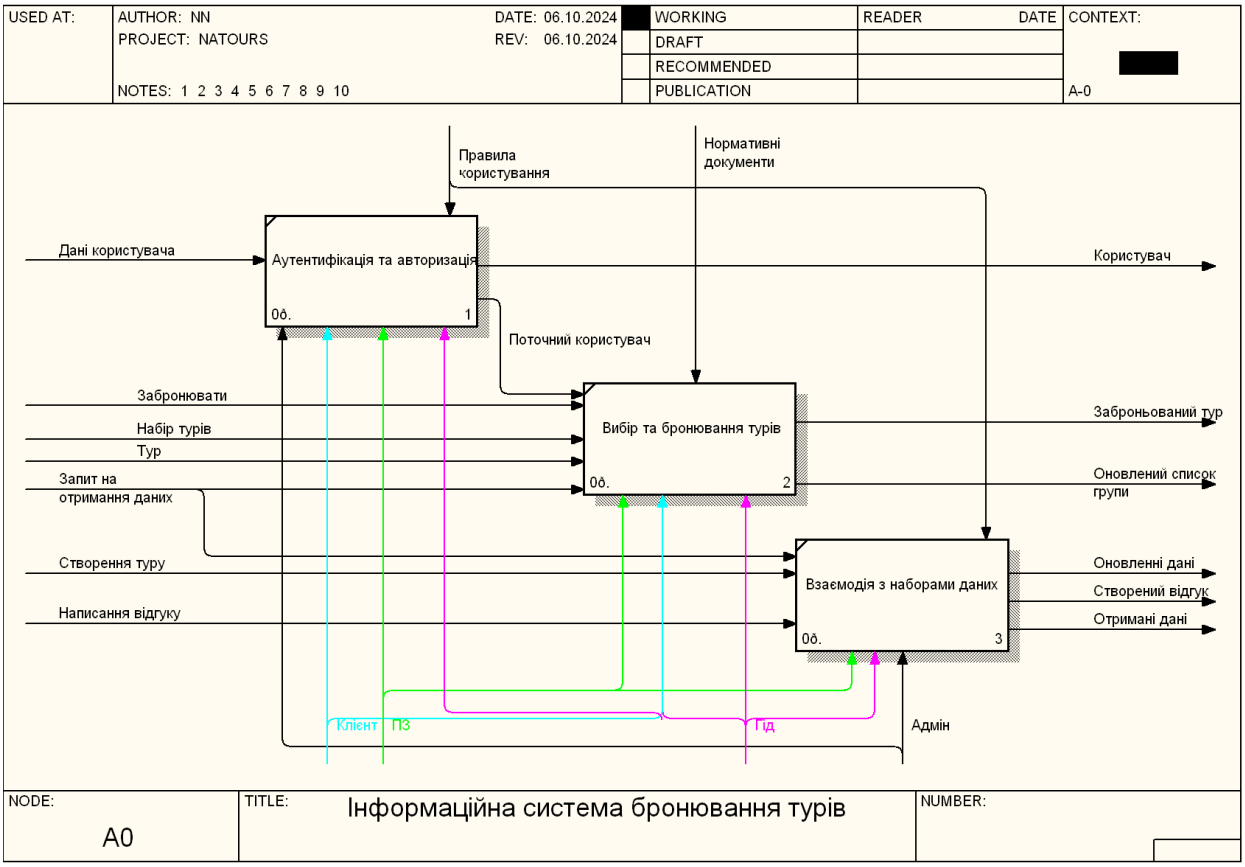

Рисунок 2.3 – Декомпозиція функціональної моделі

Проведемо декомпозицію бізнес-процесів для уточнення взаємодії компонентів між собою та уточнення функціональності системи. Зниження рівня абстракції бізнес-процесу «Аутентифікація та авторизація» дозволить зрозуміти як відбувається процес входу до системи та створення нового профілю користувача.

Бізнес-процес «Аутентифікація та авторизація» було поділено на три головних функціональні блоки, що відповідають за даний процес:

– реєстрація. Якщо користувач відвідує вперше вебсторінку даного сервісу, то першим кроком для отримання повного функціоналу системи та нових можливосте й в ролі авторизованого користувача необхідно створити обліковий запис. Для цього користувачеві необхідно надати особисту інформацію для реєстрації в системі, яку в майбутньому буде використовувати для авторизації;

– аутентифікація. Функція системи, яка отримує запит на перевірку даних під час авторизації в системі. Для цього користувач надає інформацію для входу, якщо ідентифікація користувача пройшло успішно, то його автоматично буде проведено до авторизації;

– авторизація. Після підтвердження користувача буде автоматично авторизованого в системі та надано доступ до функцій і можливостей системи, що доступні тільки при авторизації.

На рисунку 2.4 подано декомпозиції бізнес-процесу «Аутентифікація та авторизація».

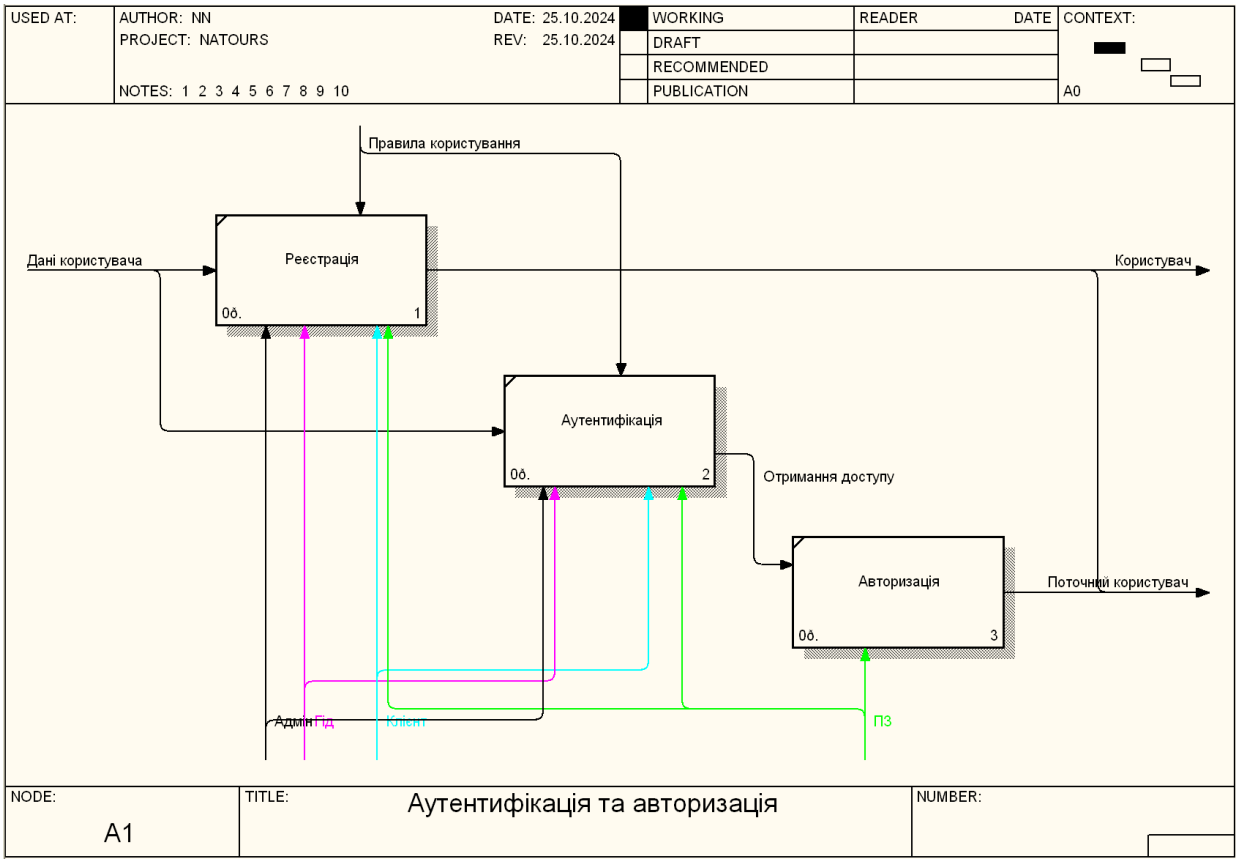

Рисунок 2.4 – Декомпозиція бізнес-процесу «Аутентифікація та авторизація»

2.3 Візуалізація функціональної моделі. Використання діаграми дерева вузлів

Візуалізація функціональної моделі за допомогою діаграми дерева вузлів дозволить показати структуру та ієрархію елементів системи. Даний підхід особливо важливий під час розробки систем різної складності, оскільки можна побачити взаємозв’язок між компонентами.

Основні переваги використання діаграми дерева вузлів:

– візуалізації ієрархії. Діаграма дерева вузлів представляє інформацію у вигляді вузлів, де кожен вузол має свої дочірні елементи. Така структура дозволить чітко відобразити ієрархічні зв’язки між різними функціями та блоками системи;

– зрозуміла структура. Завдяки візуалізації стає зрозуміло, які елементи пов’язані між собою. Наприклад, для сайту з продажу взуття можна побудувати дерево, яке покаже категорії товарів, такі як "Чоловіче взуття", "Жіноче взуття", "Дитяче взуття", та підкатегорії всередині кожної з них;

– навігація та пошук. Діаграма дерева вузлів дозволяє легко орієнтуватися та знаходити необхідні елементи в системи;

– аналіз залежностей. Діаграма дозволяє аналізувати залежності між блоками, що дозволить визначити недоліки та ключові вузли системи.

Отже, використання такої візуалізації, чітко показує взаємозв’язки між компонентами та функціональними блоками системи. Розроблена діаграма дерева вузлів для системи подана на рисунку 2.5.

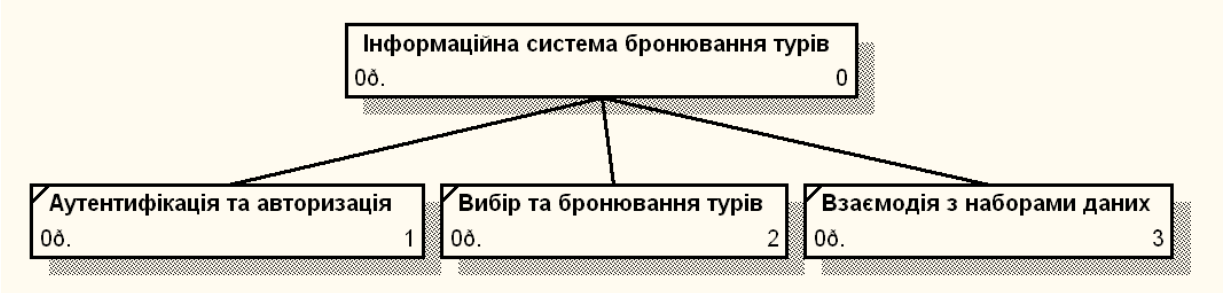

Рисунок 2.5 – Діаграма дерева вузлів

2.4 Вибір та переваги використання MVC архітектури

Архітектурний патерн MVC став класичним підходом для побудови інформаційних систем з графічним інтерфейсом користувача різної складності. Його основною метою є розподіл програми на три основних компоненти: View, Model, Controller. Саме цей розподіл має значний вплив на масштабованість, гнучкість та ефективність розробки системи:

– Model (Модель). Відповідає за управління даними та бізнес-логікою застосунку. Модель взаємодіє з наборами даних із БД або інших джерел та визначає, як дані оброблюються в системі;

– View (Представлення). Відповідає за відображення графічного інтерфейсу користувачу системи;

– Controller (Контролер). Виступає посередником між Model та View, приймаючи запити користувача через View та направляючи їх до Model. Після даних дій, контролер отримує результати та відповідно до запиту буде відправлено необхідне представлення користувачу [13].

Архітектура Model-View-Controller є популярним рішенням для проєктування програмного забезпечення. Аналізуючи вимоги до структури коду та переваги шаблону, можна сказати, що обраний патерн став кращим рішенням для побудови вебзастосунку з використанням надійних інструментів для тестування, забезпечення якості та створення безпеки [2].

Розглянемо переваги обраної архітектури та створимо графічне уявлення використанням MVC:

– розподіл відповідальності. Кожен компонент відповідає за свій функціонал, що зменшує залежність та полегшує його підтримку. Розробка представлень відбувається окремо від розробки бізнес-логіки, що спрошує процес реалізації системи;

– легкість тестування. Створені моделі та контролери можна легко протестувати незалежно від графічного інтерфейсу, оскільки їх логіка

ізольована. Таке тестування дозволить легко знаходити та виправляти помилки в системі;

– повторне використання коду. Певна модель може використовуватись повторно. Що означає, що одну бізнес-логіку можна використовувати одразу до декількох інтерфейсів;

– масштабованість. MVC полегшує масштабування програмного забезпечення, додаючи новий функціонал або проводити зміни для існуючого.

Отже, патерн MVC підходить для розробки багатьох типів додатків, особливо вебзастосунків, де розділення інтерфейсів і бізнес-логіки є важливим аспектом. Дана архітектура є однією із найпоширеніших рішень завдяки структурованості коду, що робить його придатним для складних та масштабованих застосунків.

Графічне представлення MVC архітектури подане на рисунку 2.6.

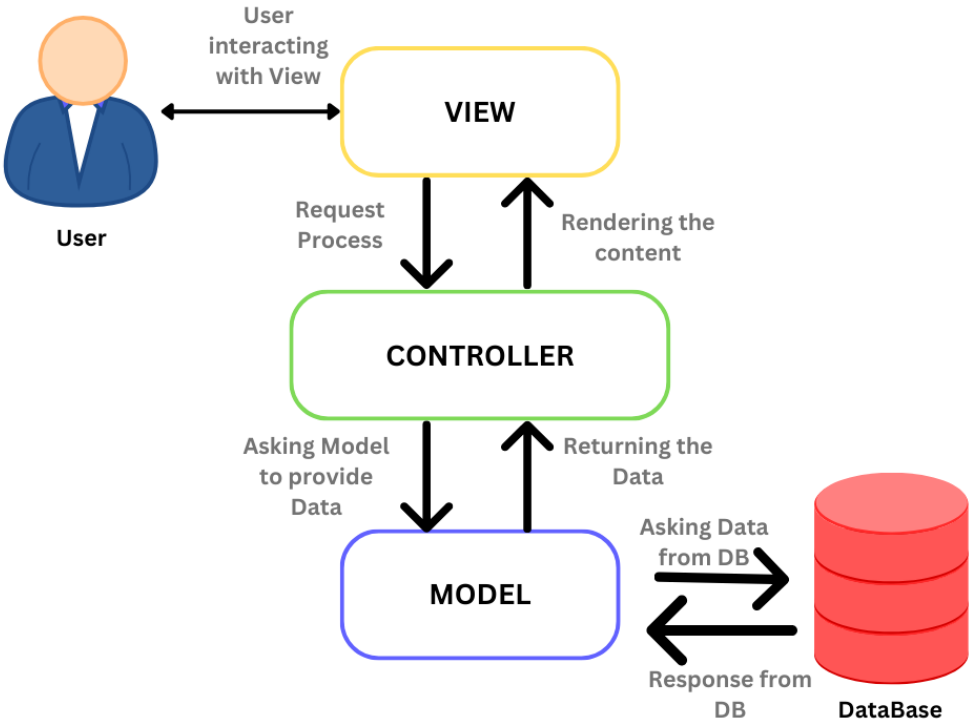

Рисунок 2.6 – Графічне представлення MVC

2.5 Моделювання діаграми класів для реалізації API сервера

Діаграма класів – це структурна діаграма в мові UML (Unified Modeling Language), яка дозволяє візуалізувати структуру системи та компоненти, що вона використовує. Сутність її використання полягає в можливості моделювання об’єктно-орієнтованих систем.

Розглянемо основні призначення діаграми класів та опишемо їх:

– моделювання структури системи. Допомагає технічним спеціалістам моделювати статичну структуру системи, включаючи класи, контролери, моделі та багато іншого. Демонструє взаємодію компонентів між собою;

– документування архітектури системи. Дозволяє описати та документувати архітектуру, яка показує як класи пов’язані між собою та їх взаємодію;

– планування і проєктування системи. Дозволяє визначити ключові компоненти системи, їх властивості та функції, а також спосіб взаємодії різних частин системи;

– підтримка розробки. Використання діаграми класів необхідне під час розробки та написання коду, оскільки демонструють чітке уявлення компонентів та структури;

– обговорення проєкту. Використовується для комунікації з менеджерами, замовниками та іншими учасниками проєкту. Вона спрощує пояснення архітектури та взаємодії між об'єктами системи.

Діаграма класів для створеної серверної частини застосунку подана на рисунку 2.7.

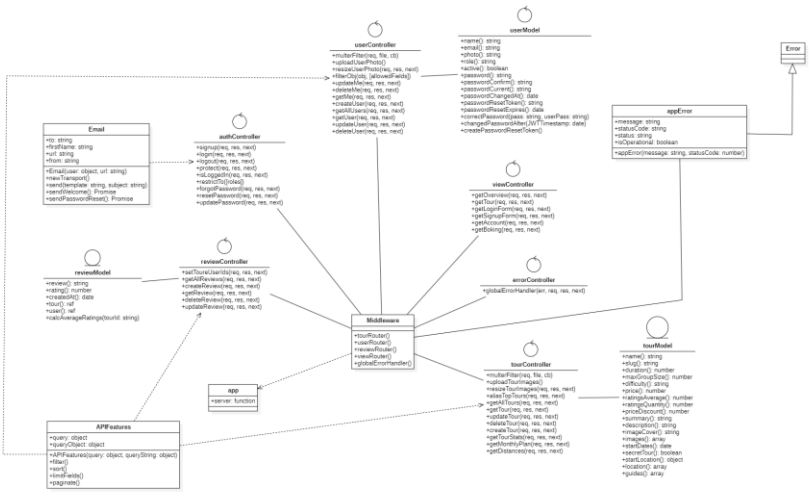

Рисунок 2.7 – Діаграма класів

Розглянемо деякі компоненти системи та створимо для них опис у вигляді таблиці 2.1.

Таблиця 2.1 – Опис компонентів діаграми класів

Компонент	Атрибути	Опис	Методи	Опис
Email	to	Електронна пошта користувача	Email(user, url)	Конструктор класу
	firstName	Ім’я користувача	newTransport()	Конфігурація відправки листів
	url	Посилання, що буде додане	send(template, subject)	Метод для обробки шаблону листів
	from	Відправник листа	sendWelcome()	Відправка та рендер шаблону Welcome
			sendPasswordReset()	Відправка та рендер шаблону скидання паролю
APIFeatures	query	Запит, що виконуються до БД	APIFeatures(query, queryObject)	Конструктор класу
			filter()	Фільтрація параметрів
	queryObject	Параметр запиту	sort()	Сортування результатів
			limitFields()	Запит необхідних полів

Продовження таблиці 2.1

Компонент	Атрибути	Опис	Методи	Опис
			paginate()	Пагінація сторінок
Error	Розширює клас appError, для створення особистих помилок			
appError	message	Повідомлення помилки	appError(message, statusCode)	Конструктор класу
	statusCode	Статус код		
	status	Статус помилки		
	isOperational	Тип помилки		
tourController			multerFilter(req, file,cb)	Допоміжна функція для middleware
			uploadTourImages()	Middleware для завантаження зображень турів
			resizeTourImages(req, res, next)	Зміна розмірів зображень
			aliasTopTours(req, res, next)	Отримання топ найкращих турів
			getAllTours(req, res, next)	Отримання всіх турів
			getTour(req, res, next)	Отримати 1 тур
			updateTour(req, res, next)	Оновлення турів
			deleteTour(req, res, next)	Видалення тура
			createTour(req, res, next)	Створення нового туру
			getTourStats(req, res, next)	Отримання статистики про тури
			getMonthlyPlan(req, res, next)	Отримання помісячного плану для турів обраного року
			getDistances(req, res, next)	Пошук турів в межах вказаної дистанції
Middleware			tourRouter()	Middleware маршрутів
			userRouter()	Middleware маршрутів
			reviewRouter()	Middleware маршрутів
			viewRouter()	Middleware маршрутів

Кінець таблиці 2.1

Компонент	Атрибути	Опис	Методи	Опис
			globalErrorHandler()	Глобальна обробка помилок
app	server	Екземпляр для запуску сервера		

Підсумовуючи моделювання діаграми класів, можна сказати, що вона є ключовим інструментом для демонстрації об’єктно-орієнтованих систем. Вона дозволяє чітко визначити компоненти системи, їхні взаємозв'язки та спрощує процес розробки, планування та підтримки програмного забезпечення. Її головні переваги — це можливість наочного представлення архітектури, полегшення комунікації та підтримка структурованого підходу до розробки системи.

2.6 Діаграма варіантів використання. Забезпечення якості застосунку

В ході визначення вимог та аналізу інформаційної системи було прийняте рішення розробити діаграму варіантів використання для уточнення вимог до розроблювального додатку. Діаграма варіантів використання (Use Case Diagram) є важливим інструментом у розробці програмного забезпечення, оскільки допомагає забезпечити якість застосунку на ранніх етапах проєктування. Основною метою є виявлення всіх можливих сценаріїв, що доступні акторам системи та представлення їх у легкому форматі для розуміння розробників, користувачів та замовників.

Розглянемо як діаграма варіантів використання може впливати на забезпечення якості програмного забезпечення:

– чітке визначення вимог. Діаграма варіантів використання допомагає визначати, як користувачі можуть взаємодіяти із системою. Це дозволить

краще зрозуміти всі можливі функції програми та зменшити ризик виникнення проблем і недоліків функцій;

– виявлення та усунення помилок у системі. Під час створення діаграми, можна легко побачити, які сценарії системи необхідно використати під час розробки;

– підвищення якості. Діаграма дозволяє визначити основну функціональність системи та їх взаємодію з користувачами. Візуалізація прецедентів дозволяє визначити процес планування архітектури та розробку системи;

– тестування функціоналу. Діаграма варіантів використання є базою для створення сценаріїв тестування. Кожен варіант використання може стати тестовим випадком, що дозволить переконатися, що всі можливі сценарії роботи програми реалізовані правильно та працюють як очікувалося.

Підсумовуючи, створення діаграми варіантів використання – це важливий етап для забезпечення якості в застосунку, що буде відповідати вимогам користувачів та дозволить уникнути потенційних проблем в майбутньому.

Розроблена діаграма варіантів використання для системи бронювання турів подана на рисунку 2.8.

Розробивши діаграму варіантів використання було визначено прецеденти системи, що доступні кожному із типів акторів, та будуть реалізовані в системі бронювання турів. Бізнес-функції, що підлягають реалізації:

– взаємодія за даними;

– створення туру;

– отримання статистики;

– введення даних;

– реєстрація;

– авторизація;

– перегляд турів;

– бронювання туру;

– оплата туру;

– оновлення даних;

– перегляд особових даних;

– забронювані тури.

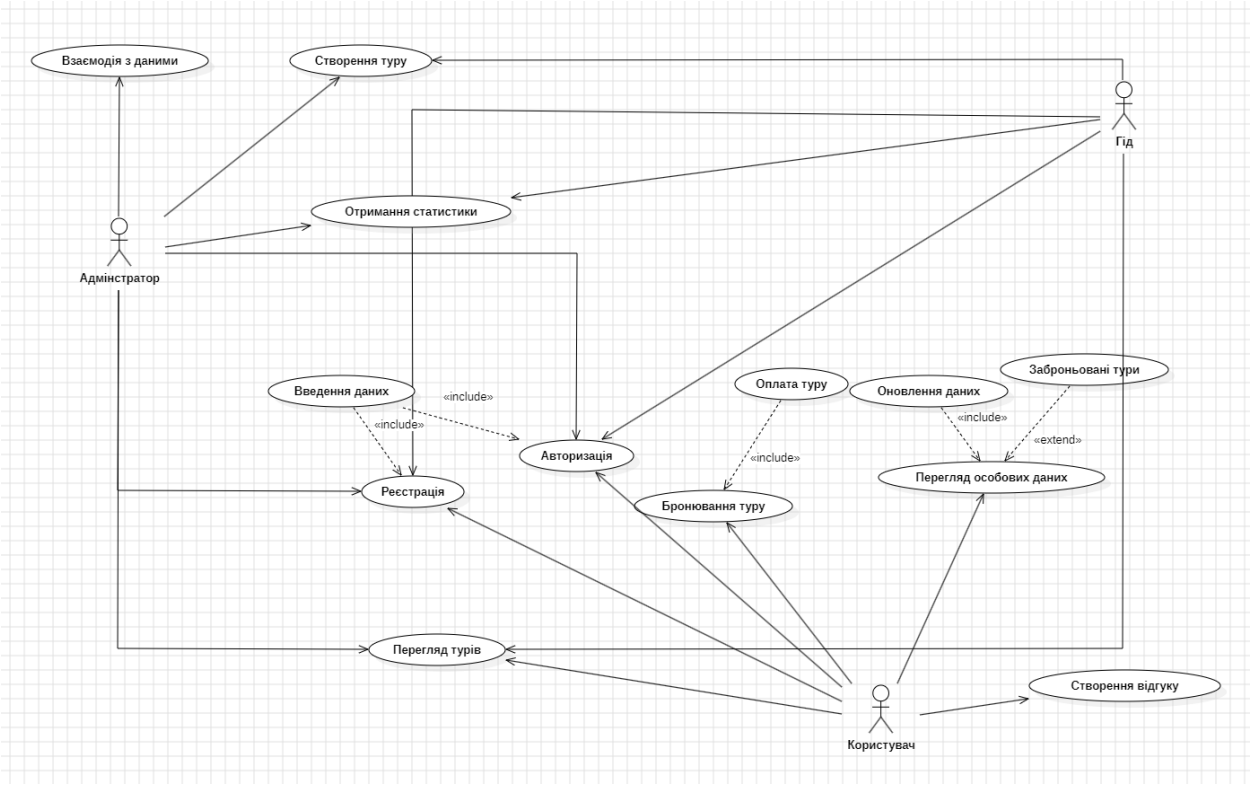

Рисунок 2.8 – Діаграма варіантів використання

В наступному розділі буде описано розробку системи з визначеними бізнес-функціями та забезпечення якості інформаційної системи бронювання турів.

2.7 Проєктування діаграм послідовності дій для визначених прецедентів системи

Діаграма послідовності дій – це один із можливих видів діаграм UML, який чітко ілюструє взаємодію об’єктів системи у послідовному порядку

протягом деякого часу. Головна сутність діаграми послідовності дій є показати порядок обміну повідомленнями між об’єктами і компонентами системи для обраного процесу. Така візуалізація дозволить деталізувати послідовність взаємодії компонентів системи.

Розглянемо головні цілі діаграми послідовності дій:

– візуалізація дій. Показує, як елементи інформаційної системи або процесу обмінюються повідомленнями;

– визначення послідовності. Дозволяє чітко розуміти, в якому порядку відбуваються події у системи, що дозволяє зрозуміти динаміку взаємодіє елементів;

– аналіз логіки і поведінки. Допомагає розробникам і аналітикам у аналізу та перевірці логіки процесу, розглядаючи кожен етап окремо.

Для будь-якої інформаційної системи необхідно розроблювати діаграму послідовності дій для візуалізації процесів. Діаграма послідовності дій для прецеденту «Бронювання турів» подано на рисунку 2.9.

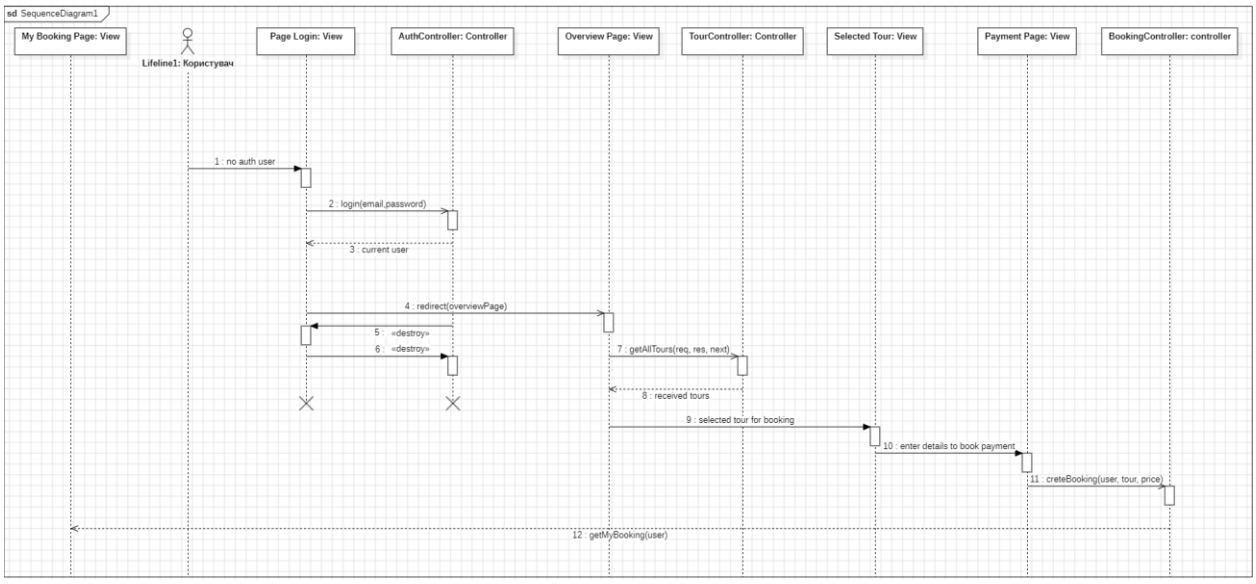

Рисунок 2.9 – Послідовність успішних дій прецеденту «Бронювання турів»

На даній діаграмі послідовності дій подано послідовність успішних кроків, що виконуються під час процесу бронювання турів. Розглянемо всі етапи та суть:

– no auth user. Користувач відвідує вебсайт на якому йому потрібно авторизуватись в системі для бронювання туру та його оплати;

– login(email, password). Після введення особистих даних для авторизації буде відправлено POST-запит до серверу, на якому буде проведено етап аутентифікації користувача. Аутентифікація – це етап перевірки користувача;

– current user. Дана відповідь включає в себе успішну аутентифікацію та авторизацію користувача, отримання json web token, що необхідний для авторизації та створення файлу cookie. Файл cookie використовується для авторизації з використанням клієнтського інтерфейсу;

– redirect(overviewPage). Авторизувавшись в системі, користувача буде автоматично перенаправлено на головну сторінку системи;

– getAllTours(req, res, next). Для перегляду всіх турів, на які можна створити бронювання необхідно отримати інформації про них. Даний запит працює автоматично та відправляє GET-запит для отримання турів;

– received tours. Після успішного виконання запиту, на сторінці відображаються доступні тури для бронювання;

– selected tour for booking. Переглядаючи список турів, користувач може детально ознайомитись із вмістом туру та одразу створити бронювання;

– enter details to book payment. Система перенаправляє користувача на сторінку оплати бронювання туру;

– createBooking(user, tour, price). Після успішної оплати на основі даних із попередніх кроків надсилається POST-запит, який створить дане бронювання та збереже в базі даних;

– getMyBooking(user). На сторінці «My Booking» користувач може переглянути всю інформацію про заброньований тур.

Отже, таким чинок, діаграма послідовності дій дозволила візуалізувати процес «Бронювання туру» з елементами як серверної так і клієнтської частини застосунку.

2.8 Задачі системи, що підлягають реалізації, для створення якості та безпеки

Під час роботи над проєктом необхідно особливу увагу приділити безпеці даних користувачів, щоб захистити їх від можливих загроз та несанкціонованого доступу. Це включає реалізацію надійного захисту на всіх рівнях розробки та дотримання найкращих практик для забезпечення конфіденційності, цілісності та доступності даних.

Ключові аспекти, що допоможуть створити ефективну систему:

– шифрування даних;

– аутентифікація та авторизація;

– захист від атак на сервері та в мережі;

– постійне оновлення системи;

– видання мінімальної кількості прав;

– зберігати тільки необхідні дані;

– визначення правильних залежностей між моделями бази даних;

– створення чіткої структури проєкту;

– створення API документації.

Таким чином, дотримання високих стандартів безпеки та використання найкращих практик розробки сприяють зниженню ризиків, пов'язаних із безпекою, та забезпечують надійний захист даних користувачів.

3 ОПИС РІШЕНЬ ТА ВИКОРИСТАННЯ ЇХ НА ПРАКТИЦІ

3.1 Обґрунтування вибору мови програмування

Написання коду є одним із самих важливих етапів під час проєктування інформаційної системи. Вибір мови програмування є важливим рішенням, яке буде впливати на ефективність розробки, продуктивність системи, підтримку та можливість масштабування. Обираючи мову програмування, необхідно враховувати ключові фактори:

– продуктивність і ефективність. Вибір мови залежить від того, наскільки швидко і ефективно необхідно виконувати задачі. Для проєктування вебдодатків можна використовувати JavaScript, Python, PHP;

– тип програми. Якщо розроблюється вебзастосунок, то кращим рішенням буде використовувати JavaScript та платформу Node.js для написання серверної частини ІС;

– масштабованість. Для великих і масштабованих систем важливо обирати мову, що підтримує високу продуктивність;

– доступність бібліотек і фреймворків. Кількість різних пакетів, що можна використати для спрощення проєктування системи, є важливим елементом під час вибору мови програмування;

– досвід і знання команди. Кваліфікація команди також відіграє значну роль. Якщо команда має великий досвід у JavaScript, це може бути вагомою причиною для вибору Node.js на сервері.

Оскільки команда включає одну особу із досвідом розробки серверу з використанням платформи Node.js, було прийняте рішення обрати її та впровадити кращі практики для створення надійного та безпечного застосунку. Node.js є гарним кандидатом, що підпадає під всі критерії для

вибору мови програмування. Розглянемо переваги платформи для розробки серверної частини застосунку:

– висока продуктивність та швидкість. Node.js працює на двигуну V8 від компанії Google, який інтерпретує JavaScript в машинний код, що забезпечує швидку обробку запитів;

– асинхронна та неблокуюча модель вводу-виводу. Node.js побудований на асинхронних операціях, що дозволяють виконувати велику кількість задач одночасно, не блокуючи єдиний потік. Якщо, використовувати синхронний код на сервері, то ми отримаємо блокування виконання операцій. Таким чином, користувачі будуть очікувати виконання даного коду для певного користувача. Кращою практикою для Node.js є використання асинхронного коду для операцій, що мають деяке навантаження;

– єдність мови на клієнтській та серверній частині. Оскільки JavaScript використовується як для клієнтської, так і для серверної частини, це спрощує розробку та дозволяє команді розробників працювати з єдиним стеком технологій;

– модульність. Модулі Node.js являють собою набір функцій, які групуються в один або декілька файлів JavaScript. Кожен створений модуль має свій власний контекст, який не впливає на роботу інших створюваних модулів. Модулі надають можливість повторно використовувати код, що збільшує ефективність його використання;

– використання гнучкого фреймворку Express. Express забезпечує мінімальну структуру та простоту налаштування, що дозволяє легко створювати маршрути, обробляти запити та інтегруватися з базами даних. Він добре підходить для побудови RESTful API та спрощує розробку серверних застосунків [12].

3.2 Обґрунтування вибору бази даних. .Переваги NoSQL

Під час вибору мови програмування стає також питання, яку обрати СУБД для інформаційної системи. Головною перевагою бази даних повинно бути легке отримання доступу до збереженої інформації.

Таким чином, було прийняте рішення обрати MongoDB, яка відмінно підходить для розробки сучасних вебдодатків завдяки своїй гнучкості, масштабованості та легкій інтеграції з Node.js [10]. Саме ця СУБД дозволить легко взаємодіяти з даними розроблювального додатку:

– легке управління складними даними. Динамічний формат документів MongoDB підходить для структурування даних, які часто змінюються. Наприклад, у вашому інтернет-магазині дані про товари можуть мати різні атрибути, що легко зберігати в MongoDB без необхідності зміни схеми бази даних;

– підтримка високої продуктивності та масштабованості. MongoDB швидко обробляє операції з великим обсягом даних і має вбудовану підтримку для збереження записів даних на декількох машинах, що є суттєвою перевагою для масштабування веб-застосунків;

– зручна інтеграція з JavaScript. Оскільки MongoDB працює з документами у форматі BSON (схожий на JSON), інтеграція з Node.js відбувається природно і без зайвих перетворень форматів даних, що значно спрощує обробку запитів.

3.3 Моделювання даних інформаційної системи. Аналіз зв’язкім між моделями

Під час проєктування моделей на серверній частині додатку автоматично буде створено моделі, що додані в СУБД MongoDB. Головна різниця MongoDB від реляційних СУБД – це те, що дані зберігаються у вигляді

документу, а не запису [13]. Документ представляє формат даних ключ-значення, що дозволяє легко отримувати необхідні дані.

Перед початком побудови моделей, необхідно проаналізувати які типи зв'язків необхідно використати для моделей, щоб створити ефективну структуру бази даних.

Існує три основних типи відношень:

– один до одного. Це відношення між даними, коли одне поле може зберігати одне значення;

– один до багатьох. Один документ може бути зв'язаний з іншими документами. Наприклад, може відноситись один до тисячі;

– багато до багатьох. Можна сказати, що відношення йдуть в двох напрямках, наприклад, один фільм має багато акторів, в той же час, один актор грає роль в багатьох фільмах.

На основі даних відношень було побудовано діаграму взаємозв'язків моделей між собою. Діаграма подана на рисунку 3.1.

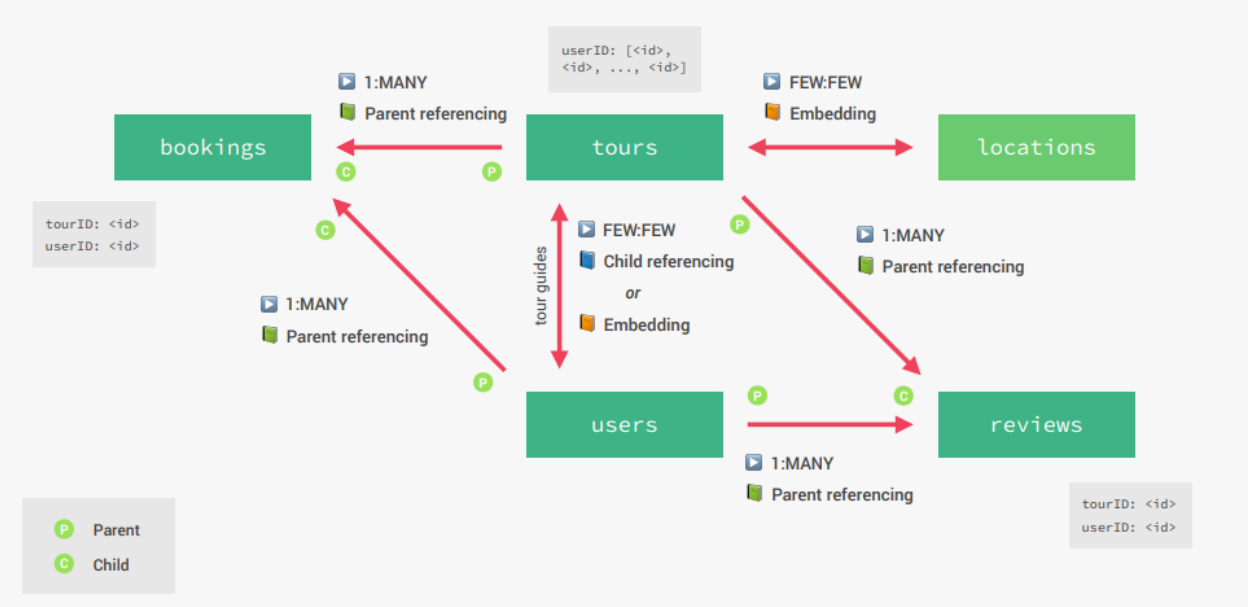

Рисунок 3.1 – Діаграма відношень між моделями системи

Розробивши візуальне представлення зв'язків між моделями системи, тепер необхідно проаналізувати та визначити, які ключі повинні бути присутні

в кожній моделі, і провести їх моделювання за допомогою пакету Mongoose [16].

Точне визначення ключових полів для кожної моделі є критично важливим етапом проєктування структури даних, оскільки саме ці ключі забезпечують організоване та повноцінне зберігання інформації в кожному документі. Завдяки цьому всі створювані документи будуть мати необхідний набір даних, що підвищує цілісність і зручність обробки інформації в системі, а також полегшує подальшу роботу з нею під час розробки, тестування та масштабування додатка.

Опис визначених ключових полів для моделей описано в таблиці 3.1.

Таблиця 3.1 – Опис документів та їх полів

№	Найменування колекції	Ключі документа	Тип даних	Призначення
1	users	_id	ObjectId	Унікальний ідентифікатор
		name	String	Ім’я
		email	String	Електронна адреса
		photo	String	Фото користувача
		role	String	Роль в системі
		active	Boolean	Статус видалення профілю
		password	String	Встановлений пароль
		passwordResetExpires	Date	Встановлюється автоматично при скиданні паролю
		passwordResetToken	String	Встановлюється автоматично при скиданні паролю
2	bookings	_id	ObjectId	Унікальний ідентифікатор
		user	ObjectId	Користувач, що створив бронювання туру
		tour	ObjectId	Заброньований тур користувачем
		price	Number	Ціна
		createdAt	Date	
		paid	Boolean	Статус оплати туру
3	reviews	_id	ObjectId	
		review	String	Написаний відгук
		rating	Number	Рейтинг за 5 бальною шкалою

Кінець таблиці 3.1

№	Найменування колекції	Ключі документа	Тип даних	Призначення
3	reviews	createdAt	String	
		tour	ObjectId	Тур, до якого написаний відгук
		user	ObjectId	Користувач, що написав відгук
4	tours	_id	ObjectId	Унікальний ідентифікатор
		name	String	Назва
		duration	Number	Тривалість
		maxGroupSize	Number	Максимальна група
		difficulty	String	Складність
		price	Number	Ціна
		ratingsAverage	Number	Середній рейтинг
		ratingsQuantity	Number	Кількість відгуків
		summary	String	Короткий опис
		description	String	Детальний опис
		imageCover	String	Головне зображення
		images	Array[String]	Додаткові зображення
		createdAt	Date	
		startDates	Array[Date]	Дати початку
		secretTour	Boolean	Статус секретного туру, наприклад для VIP клієнтів
		startLocation	Object	Початкова локація
		locations	Array[Object]	Всі локації, що будуть відвідані
		guides	Array[ObjectId]	Гіди даного туру
		slug	String	Рядок, що використовується для створення url

Маючи розроблену структуру моделей бази даних, необхідно створити відповідні моделі, використовуючи Mongoose, щоб забезпечити коректне відображення даних у MongoDB. Це включає визначення схем для кожної моделі з вказівкою всіх необхідних полів і їхніх типів, а також налаштування валідації, зв’язків і базових значень.

Модель Tour необхідна для того, щоб дані зберігались в СУБД в тому вигляді в якому вони будуть спроєктовані. В лістингу 3.1 подано програмний код для реалізації моделі Tour.

Лістинг 3.1 – Програмний код моделі Tour

```
const mongoose = require('mongoose');
const slugify = require('slugify');

const tourSchema = new mongoose.Schema(
  {
    name: {
      type: String,
      unique: true,
      required: [true, 'A tour must have a name'],
      trim: true,
      maxlength: [40, 'A tour name must have less or equal then 40
characters'],
      minlength: [10, 'A tour name must have more or equal 10
characters'],
    },
    slug: String,
    duration: {
      type: Number,
      required: [true, 'A tour must have a duration'],
    },
    maxGroupSize: {
      type: Number,
      required: [true, 'A tour must have a group size'],
    },
    difficulty: {
      type: String,
      required: [true, 'A tour must have a difficulty'],
      enum: {
        values: ['easy', 'medium', 'difficult'],
        message: 'Difficulty is either: easy, medium, difficult',
      },
    },
    price: {
      type: Number,
      required: [true, 'A tour must have a name'],
    },
    ratingsAverage: {
      type: Number,
      default: 0,
      min: [0, 'Rating must be above 0'],
      max: [5, 'Rating must be below 5.0'],
      set: (val) => Math.round(val * 10) / 10, //4.666 => 46.666 => 47
=> 4.7
    },
    ratingsQuantity: {
      type: Number,
      default: 0,
    },
    priceDiscount: {
```

```
      type: Number,
      validate: {
        validator: function (val) {
          return val < this.price;
        },
        message: 'Discount price ({VALUE}) should be below regular
price',
      },
    },
    summary: {
      type: String,
      trim: true,
      required: [true, 'A tour must have a description'],
    },
    description: {
      type: String,
      required: true,
      trim: true,
    },
    imageCover: {
      type: String,
      required: [true, 'A tour must have a cover image'],
    },
    images: [String],
    createdAt: {
      type: Date,
      default: Date.now(),
      select: false,
    },
    startDates: [Date],
    secretTour: {
      type: Boolean,
      default: false,
    },
    startLocation: {
      //GeoJSON
      type: {
        type: String,
        default: 'Point',
        enum: ['Point'],
      },
      coordinates: [Number],
      address: String,
      description: String,
    },
    locations: [
      {
        type: {
          type: String,
          default: 'Point',
          enum: ['Point'],
```

```
        },
        coordinates: [Number],
        address: String,
        description: String,
      },
    ],
    guides: [
      {
        type: mongoose.Schema.ObjectId,
        ref: 'User',
      },
    ],
  },
  {
    toJSON: { virtuals: true },
    toObject: { virtuals: true },
  },
);
```

Для кожної моделі було прийняте рішення добавляти проміжне програмне забезпечення (middleware) та методи для документа, яке буде налаштовувати взаємодію моделі з даними, наприклад, хешування паролю користувача, перевірка правильного введеного паролю та багато іншого. В даному випадку розглянемо middleware та методи, що взаємодіють з даними користувача. В лістингу 3.2 подано методи та проміжне програмне забезпечення.

Лістинг 3.2 – Middleware та методи для моделі userModel

```
userSchema.pre('save', async function (next) {
  //only run this func if pass was actually modified
  if (!this.isModified('password')) return next();

  //hash the pass with cost of 12
  this.password = await bcrypt.hash(this.password, 14);

  //delete passwordConfirm field
  this.passwordConfirm = undefined;
  next();
});

userSchema.pre('save', async function (next) {
  if (!this.isModified('password') || this.isNew) return next();
```

```
  this.changedPasswordAt = Date.now();
  next();
});

userSchema.pre(/^find/, function (next) {
  this.find({ active: { $ne: false } });

  next();
});

userSchema.methods.correctPassword = async function (
  candidatePassword,
  userPassword,
) {
  return await bcrypt.compare(candidatePassword, userPassword);
};

userSchema.methods.changedPasswordAfter = function (JWTTimestamp) {
  if (this.passwordChangedAt) {
    const changedTimestamp = parseInt(
      this.passwordChangedAt.getTime() / 1000,
      10,
    );

    return JWTTimestamp < changedTimestamp; //100 < 200
  }

  //false means not changed
  return false;
};

userSchema.methods.createPasswordResetToken = function () {
  const resetToken = crypto.randomBytes(32).toString('hex');
  this.passwordResetToken = crypto
    .createHash('sha256')
    .update(resetToken)
    .digest('hex');
  this.passwordResetExpires = Date.now() + 10 * 60 * 1000;

  return resetToken;
};
```

Таким чином, при проєктуванні моделі необхідно не тільки визначити структуру для зберігання в базі даних, але й розробити middleware для автоматичної обробки запитів до БД, а також створити методи, що виступають як бізнес-логіка додатку.

3.4 Захист маршрутів. Використання проміжного програмного забезпечення

Створення захисту та надання тільки мінімально необхідного доступу є важливим критерієм під час проєктування інформаційних систем. Для захисту маршрутів з використанням проміжного програмного забезпечення (middleware) можна створювати спеціальні функції, що взаємодіють з об’єктами запиту та відповіді та мають вбудовану функцію next(), яка дозволяє переходити з однієї функції до іншої. Таким чином перехід функцій називається набором проміжних функцій програмного забезпечення, що використовуються для визначених маршрутів системи. Наприклад, таким чином ми можемо перевіряти чи авторизований користувач, перевірити роль користувача та багато інших.

В лістингу 3.3 подано програмний код однієї із таких функцій, що буде перевіряти чи авторизований користувач, якщо так, то ми зможемо отримати доступ до іншого функціоналу.

Лістинг 3.3 – Middleware функція для перевірки авторизації

```
const protect = catchAsync(async function (req, res, next) {
  //1)getting token and check of it's there
  let token;
  if (
    req.headers.authorization &&
    req.headers.authorization.startsWith('Bearer')
  ) {
    token = req.headers.authorization.split(' ')[1];
  } else if (req.cookies.jwt) {
    token = req.cookies.jwt;
  }

  if (!token) {
    return next(
      new AppError('Your not logged in! Please log in to get access.',
401),
    );
  }

  //2)varification token
```

```
  const decoded = await promisify(jwt.verify)(token,
process.env.SECRET);
  // console.log(decoded);

  //3)check if user still exists
  const currentUser = await User.findById(decoded.id);
  if (!currentUser) {
    return next(new AppError('User does not have exist!'));
  }
  //4)check if user changed password after the token was issued
  if (currentUser.changedPasswordAfter(decoded.iat)) {
    return next(
      new AppError('User recently changed password! Log in again!',
401),
    );
  }

  //grant access to protected route
  req.user = currentUser;
  res.locals.user = currentUser;
  next();
});
```

Алгоритм перевірки користувача включає перевірку наявного JWT-токена, який надається у відповідь лише після введення правильних даних. Отримавши токен, ми перевіряємо його, щоб визначити, чи не було внесено користувачем змін. Після проходження цих етапів передаємо користувача в об'єкт запиту, щоб наступні middleware-функції могли взаємодіяти з авторизованим користувачем.

Для того, щоб мати впровадити розроблену функцію protect для маршруту, необхідно використати її в наборі функцій для певного маршруту. Реалізація використання подана в лістингу 3.4.

Лістинг 3.4. – Використання middleware функції для маршруту

```
router.use(protect);
router.patch('/updateMyPassword', updatePassword);
router.patch('/updateMe', uploadUserPhoto, resizeUserPhoto, updateMe);
router.delete('/deleteMe', deleteMe);
router.get('/me', getMe, getUser);
```

Отже, таке використання middleware-функцій є дуже ефективним і важливим рішенням під час проєктування інформаційних систем, оскільки воно дозволяє централізовано контролювати доступ до ресурсів і реалізовувати авторизацію на рівні маршрутизатора. Це підвищує безпеку додатка, зменшуючи ризик несанкціонованого доступу та дозволяючи автоматизувати обробку запитів залежно від ролі користувача або інших критеріїв.

Крім того, middleware-функції можуть виконувати додаткові завдання, такі як обробка фотографій користувачів, перевірка правильності введеного пароля, а також налаштування ідентифікації користувача, що сприяє спрощенню коду та забезпечує модульність. Такий підхід дозволяє легше підтримувати і масштабувати систему в майбутньому, оскільки всі важливі перевірки та операції винесені в окремі функції, які можуть повторно використовуватися для різних маршрутів та компонентів.

3.5 Використання додаткових пакетів для створення безпеки та якості застосунку

Використання додаткових пакетів для створення вебзастосунку є фундаментальним принципом побудови якісного та безпечного застосунку. Для розробки вебдодатку бронювання турів було обрано декілька критично важливих пакетів, що позитивно вплинуть на захист та якість системи:

– cors. Дозволяє налаштувати політику доступу до ресурсів серверу або інших доменів. Використання cors необхідно для доступу до API з різних клієнтів, наприклад веб-сайт або мобільний додаток. Додаючи cors, ви захищаєте сервер від несанкціонованих запитів з інших джерел, забезпечуючи безпечний доступ тільки для дозволених доменів;

– helmet. Встановлює стандартні заголовки безпеки HTTP для захисту від атак, таких як cross-site scripting (XSS) та іншими вразливостями, що пов’язані з відправкою і обробкою заголовків;

– morgan. HTTP-логер, який використовується для введення логів вхідних запитів на сервер. У різних режимах виводить різну деталізацію запиту, що дозволить відстежувати роботу додатку;

– express-rate-limit. Обмежує кількість запитів до API від одного IP-адресу за вказаний проміжок часу. Це знижує ризик атак типу brute-force і DDoS (distributed denial-of-service), підвищуючи стійкість додатку. Наприклад, у конфігурації max: 200 дозволяє максимум 200 запитів на годину від одного IP, що запобігає перевантаженню сервера через велику кількість запитів.

– express-mongo-sanitize. Надає захист додатку від ін’єкцій в запити до бази даних MongoDB, що можуть бути виконані через запити з використанням спеціального символу $;

– xss-clean. Надає захист від атак XSS (cross-site scripting) шляхом очищення вхідних даних у скриптах;

– hpp. Захищає додаток від атак типу HTTP Parameter Pollution, які можуть викликати небажані конфлікти в значеннях параметрів. whitelist дозволяє обійти захист для певних параметрів (наприклад, duration, price), щоб уникнути непередбачуваних обмежень при передачі параметрів;

– compression. Стискає відповіді сервера, використовуючи спеціальні алгоритми. Стискання дозволяє зменшити навантаження на сервер, що в свою чергу підвищить продуктивність додатку.

Звичайно, у додатку на платформі Node.js можна інтегрувати більшу кількість пакетів для підвищення якості, безпеки та продуктивності інформаційної системи. Використані пакети, що були проваджені в систему подані в лістингу 3.5.

Лістинг 3.5 – Впровадження пакетів до інформаційної системи

```
app.use(express.static(path.join(__dirname, 'public')));

app.use(cors());

//set security http headers
app.use(helmet());
```

```
//development logging
if (process.env.NODE_ENV === 'development') {
  app.use(morgan('dev'));
}

//limit requests from same API
const limiter = rateLimit({
  max: 200,
  windowMs: 60 * 60 * 1000,
  message: 'To many requests from this IP, please try again in an
hour!',
});
app.use('/api', limiter);

//body parser, reading data from body into req.body
app.use(express.json({ limit: '10kb' }));
app.use(cookieParser());

//data sanitization against NoSQL query injection
app.use(mongoSanitize());

//data sanitization against XSS
app.use(xss());

//prevent HTTP parameter pollution
app.use(
  hpp({
    whitelist: [
      'duration',
      'price',
      'difficulty',
      'ratingsAverage',
      'ratingsQuantity',
      'maxGroupSize',
    ],
  }),
);

app.use(compression());

app.use((req, res, next) => {
  req.requestTime = new Date().toISOString();
  // console.log(req.cookies);
  next();
});

app.use((req, res, next) => {
  res.setHeader(
    'Content-Security-Policy',
```

```
    "img-src 'self' data: https://a.tile.openstreetmap.org
https://b.tile.openstreetmap.org https://c.tile.openstreetmap.org",
  );
  next();
});
```

3.6 Тестування розроблених API з використанням програмного застосунку Postman

Програмний застосунок для тестування розроблених API є критично важливим інструментом для розробників програмного забезпечення. Використання Postman допомагає перевіряти результати розробленого інтерфейсу без використання клієнтської частини, що в свою чергу допоможе забезпечити якість та надійність компонентів системи. Створення запитів до серверної частини застосунку дозволить автоматизувати процес перевірки отримання відповідей API, виявлення помилок та перевіряти, чи відповідають відповіді очікуваним результатам [9]. Таким чином кожен розроблений інтерфейс для ресурсів інформаційної системи необхідно протестувати та впевнитись в правильності роботи.

Протестуємо API з різними методами, такими як, GET, POST, PATCH, DELETE. Спочатку відправимо запит для реєстрації в системі за маршрутом {{url}}/api/v1/users/signup і методом POST, ввівши електронну адресу, ім’я, пароль та підтвердження введеного паролю.

Ввівши необхідні дані та відправивши запит до серверу було отримано відповідь про успішну реєстрацію в системі. Відповідь представлена у вигляді об’єкта з полями статусу, токена авторизації, та детальної інформації про користувача. Успішна відповідь API для реєстрації нового користувача подана на рисунку 3.2.

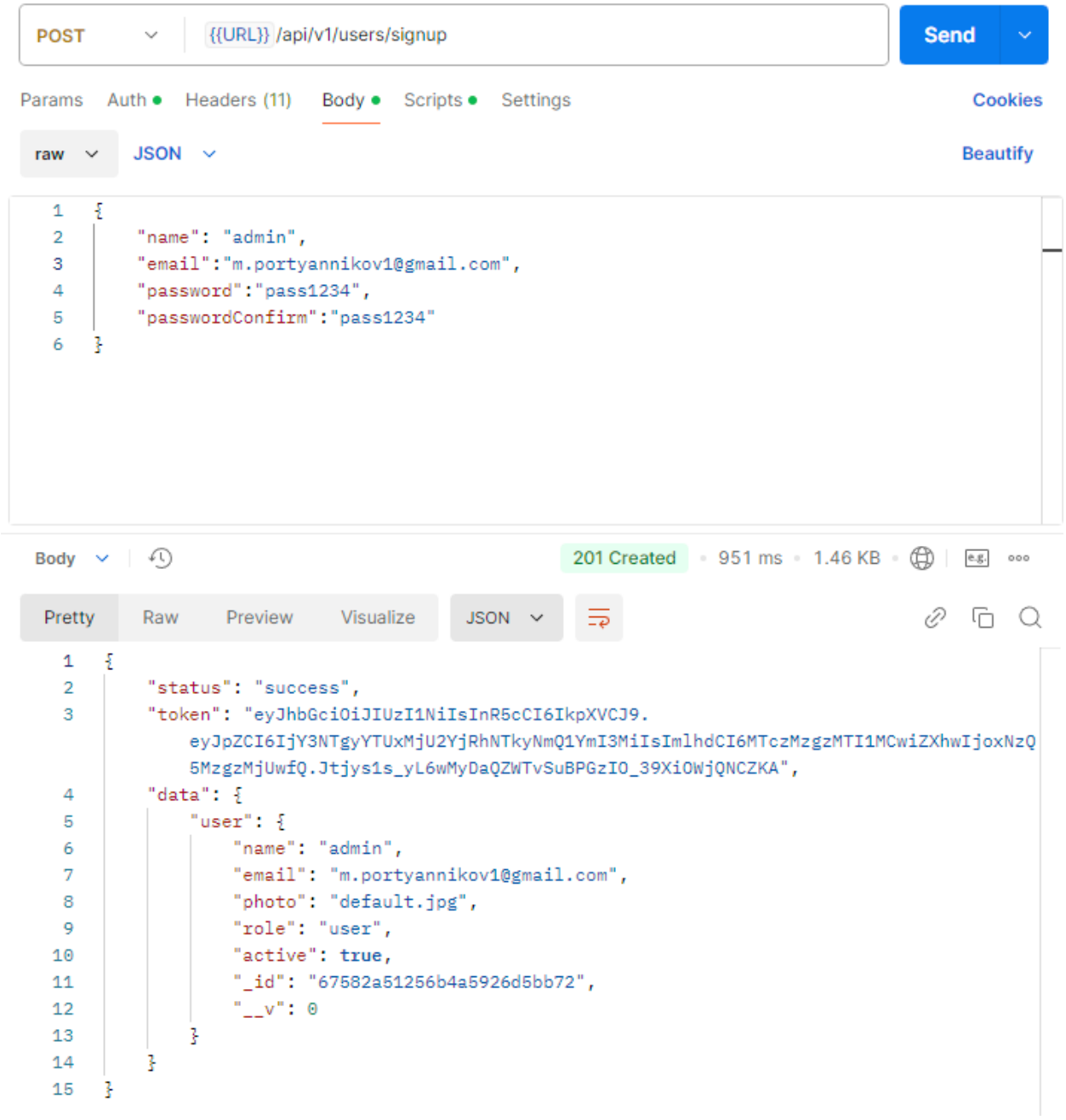

Рисунок 3.2 – Успішна відповідь API для реєстрації нового користувача

Також спробуємо відправити аналогічний запит, але з деякою помилкою, для того щоб перевірити очікуваний результат роботи. Для цього відправимо запит з різними паролями. Результатом запиту повинні отримати повідомлення про те, що паролі не співпадають. Результат відповіді подано на рисунку 3.3.

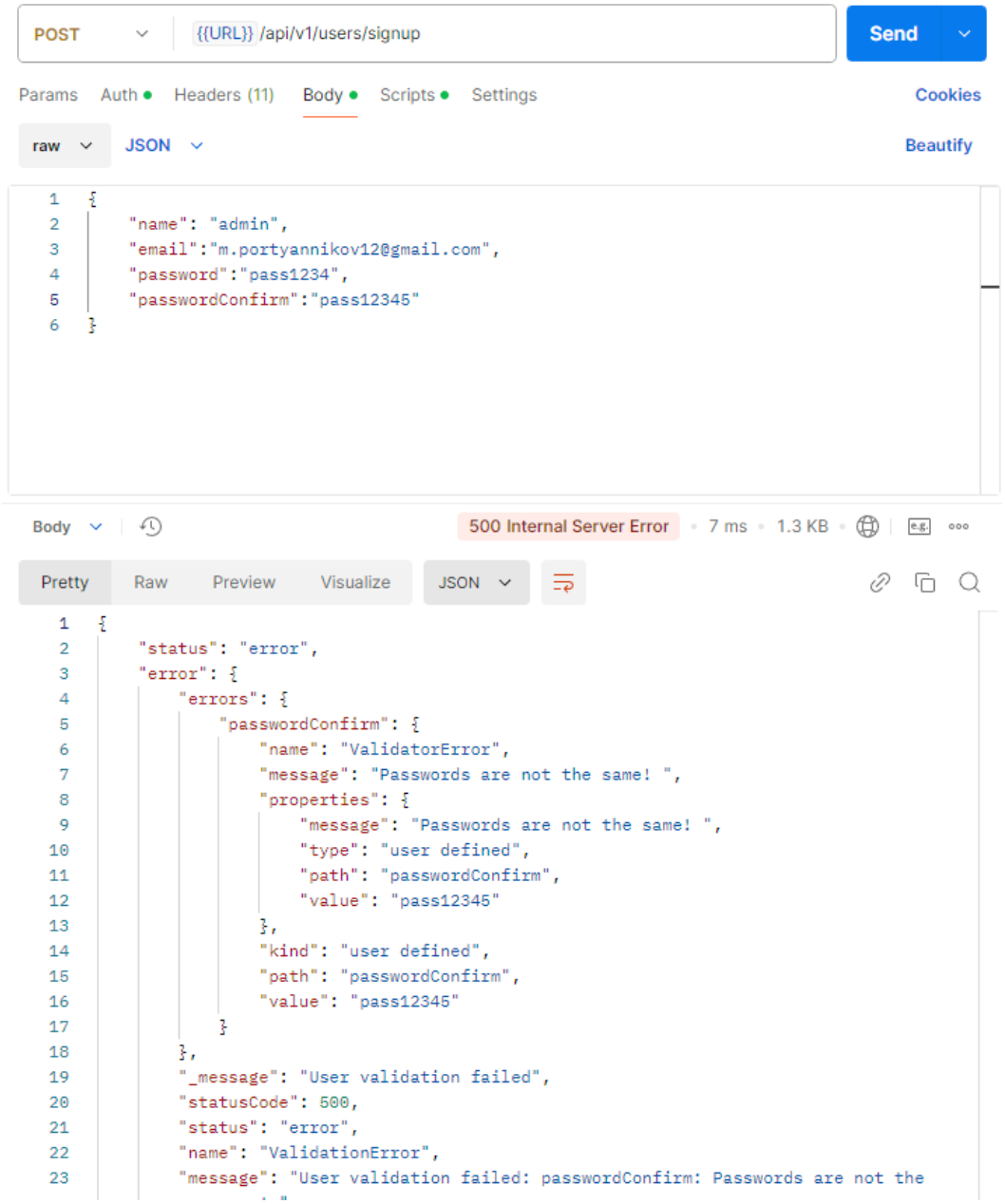

Рисунок 3.3 – Помилка введення даних

Наступним методом для тестування було обрано GET. Методи використовується для отримання набору даних певного ресурсу. Для цього створимо запит на отримання всіх турів. Результат методу GET для ресурсу турів подано на рисунку 3.4.

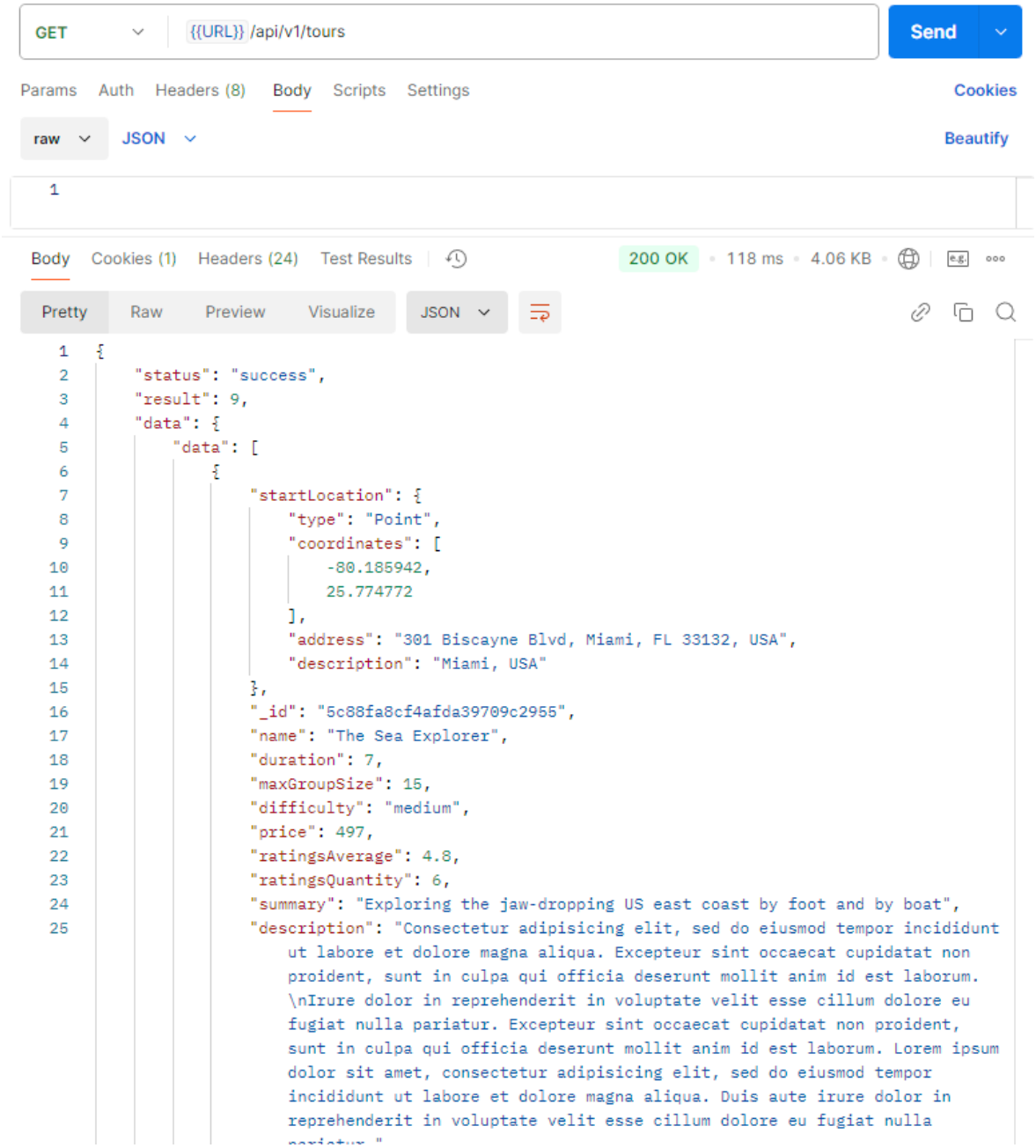

Рисунок 3.4 – Результат відповіді на отримання всіх турів

Важливо обов’язково протестувати метод PATCH. Його мета полягає в частковому оновленні ресурсу. Це означає, що не потрібно оновлювати весь всі ключі ресурсу, можна обрати окремі поля та надати інформацію про них. Прикладом тестування методу PATCH буде оновлення паролю користувача. Поданий URL на рисунку 3.5 відповідає за оновлення паролю поточного користувача, для цього потрібно вказати поточний пароль та новий пароль користувача.

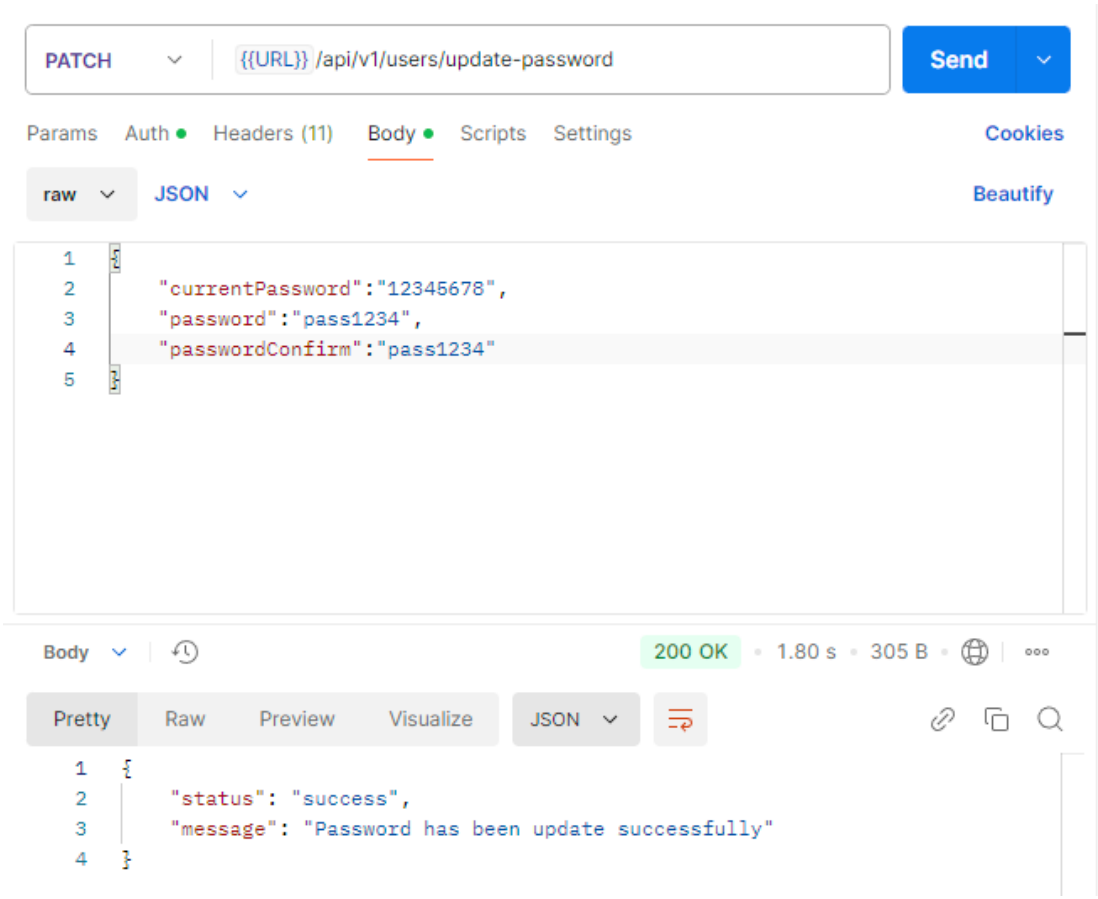

Рисунок 3.5 – Результат оновлення паролю користувача

Останнім методом HTTP-запитів буде розглянуто DELETE. Його основна мета – забезпечити видалення даних, які більше не потрібні, або з метою оптимізації роботи системи. Прикладом роботи методу DELETE буде видалення туру із системи за його унікальним ідентифікатором. Оскільки звичайний користувач не може видалити тур, для цього було прийняте рішення забезпечити доступ до цього маршруту тільки авторизованим адміністраторам та гідам. Таким чином, вказавши Bearer Token можна отримати поточного користувача та визначити його роль в системі. Після успішного проходження цих етапів буде відправлено запит на видалення туру і отримано відповідь із статусом 204.

Інформацію про метод DELETE подано на рисунку 3.6.

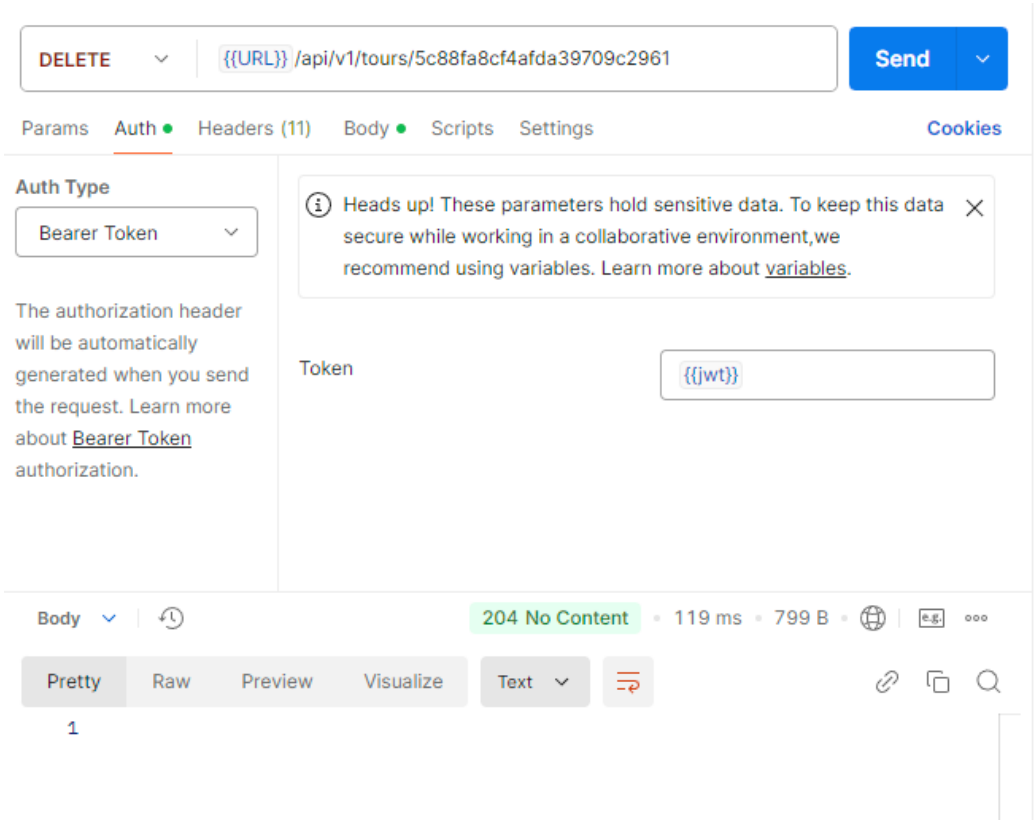

Рисунок 3.6 – Результат видалення туру за його унікальним ідентифікатором

Розглянемо випадок, коли авторизований користувач не має прав доступу до надсилання запиту для видалення туру. Спочатку авторизуємось, де роль користувача буде user. Виконавши аналогічний запит, але іншим користувачем, отримає повідомлення про те, що немає доступу для виконання цієї дії. Результат тестування подано на рисунку 3.7.

Отже, таким чином ми можемо проводити тестування розроблених обробників маршрутів для того, щоб переконатися у їхній коректній роботі, відповідності вимогам та точності передачі даних між клієнтом і сервером. Postman є одним із найпоширеніших інструментів для тестування API, який дозволяє не лише перевіряти функціональність маршрутів, але й аналізувати відповіді, відстежувати помилки та симулювати різні сценарії використання.

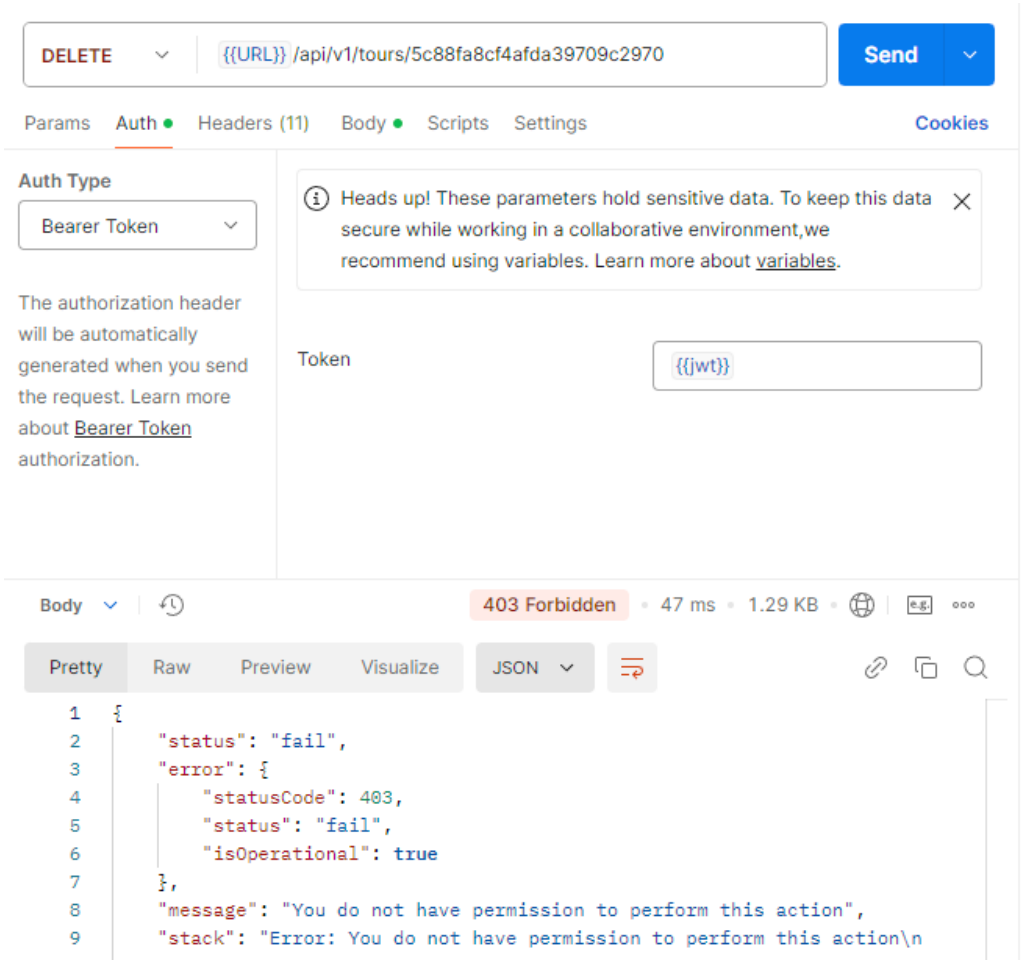

Рисунок 3.7 – Видалення ресурсу без отриманих прав доступу

3.7 Автоматизоване тестування програмного забезпечення. Створення тестів

Автоматизоване тестування програмного забезпечення є невід'ємною частиною процесу розробки для надання якості і безпеки вебдодатку. Автоматизоване тестування – це процес використання спеціальних інструментів або програмного забезпечення для автоматизації виконання сценаріїв [19]. Основною метою є зменшення часу і ресурсів, що необхідні для перевірки роботи програмного забезпечення. Завдяки цій меті можна визначити основні переваги та причини для використання автоматизованого тестування:

– швидкість виконання. Використання автоматизованих тестів значно пришвидшує перевірку сценаріїв, оскільки їх обсяг може перевищувати десятки варіантів;

– підвищена точність. Оскільки під час ручного тестування можна допустити помилки в ході перевірки сценарію. Автоматизовані тести виконують ті самі дії з тією ж самою послідовністю;

– економія ресурсів. Створення автоматизованих тестів потребує часу та ресурсів, але якщо вони проводяться регулярно, то витрата ресурсів не буде марною;

– повторюваність. Тести можна використовувати необмежену кількість разів і на будь-якому етапі розробки;

– паралельне виконання. Автоматизовані тести можуть виконуватись паралельно, що знизить час на проведення тестування;

– масштабованість. Автоматизоване тестування легко масштабувати відповідно до збільшення проєкту;

– підвищення якості. Автоматизоване тестування дозволяє швидше знаходити помилки, знижуючи ризик випуску дефектного продукту.

Таким чином, можна сказати, що будь-яка інформаційна система повинна буди протестована і результати кожного сценарію повинні відповідати очікуваним результатам. Тільки після отримання успішних результатів тесту можна сказати, що сценарій відповідає вимогам та може бути допущений до наступного етапу.

В ході роботи, було розроблено велику кількість автоматизованих тестів з використанням бібліотек supertest та jest. Їх використання дозволило створити тестові сценарії для перевірки розроблених API системи.

Для ресурсу турів було розроблено набір тестів, що тестують отримання даних, перевірку введених параметрів та створення нового туру. Дані сценарії було протестовані з використанням тестової бази даних, для того щоб залишилась цілісність даних. Реалізація тестових сценаріїв подана в лістингу 3.6.

Лістинг 3.6 – Програмний код тестових сценаріїв для ресурсу Tour

```
beforeAll(async () => {
  await mongoose.connect(process.env.MONGO_URL);

  console.log('MongoDB Connected:', mongoose.connection.readyState);
// Должно быть "1"
});

afterAll(async () => {
  await mongoose.disconnect();
});

afterEach(async () => {
  await Tour.deleteMany({ name: 'Secret Tour' });
});
describe('GET /api/v1/tours', () => {
  it('должен возвращать все документы с корректным статусом и
результатом', async () => {
    const response = await
request(app).get('/api/v1/tours').expect(200);

    expect(response.body.status).toBe('success');
    expect(response.body.result).toBe(9);
    expect(response.body.data.data.length).toBe(9);
    expect(response.body.data.data[0]).toHaveProperty(
      'name',
      'The Sea Explorer',
    );
    expect(response.body.data.data[1]).toHaveProperty(
      'name',
      'The Park Camper',
    );
  });

  it('должен корректно фильтровать данные по query параметрам', async
() => {
    const response = await request(app)
      .get('/api/v1/tours?price[gte]=500')
      .expect(200);

    expect(response.body.status).toBe('success');
    expect(response.body.result).toBe(7);
    expect(response.body.data.data[0]).toHaveProperty(
      'name',
      'The Snow Adventurer',
    );
  });

  it('', async () => {
```

```
    const response = await
request(app).get('/api/v1/tours?limit=1&page=2');

    expect(response.body.status).toBe('success');
    expect(response.body.data.data.length).toBe(1);
  });
});

describe('POST /api/v1/tours', () => {
  it('create new tour', async () => {
    const response = await request(app).post('/api/v1/tours').send(t);

    expect(response.body.status).toBe('success');
    expect(response.body.data.data.name).toBe(t.name);
  });
});
```

Запустивши тестові сценарії, отримаємо інформацію про успішне виконання всіх тестів, що підтверджує коректну роботу окремих модулів та взаємодію компонентів. Результат тестування подано на рисунку 3.8.

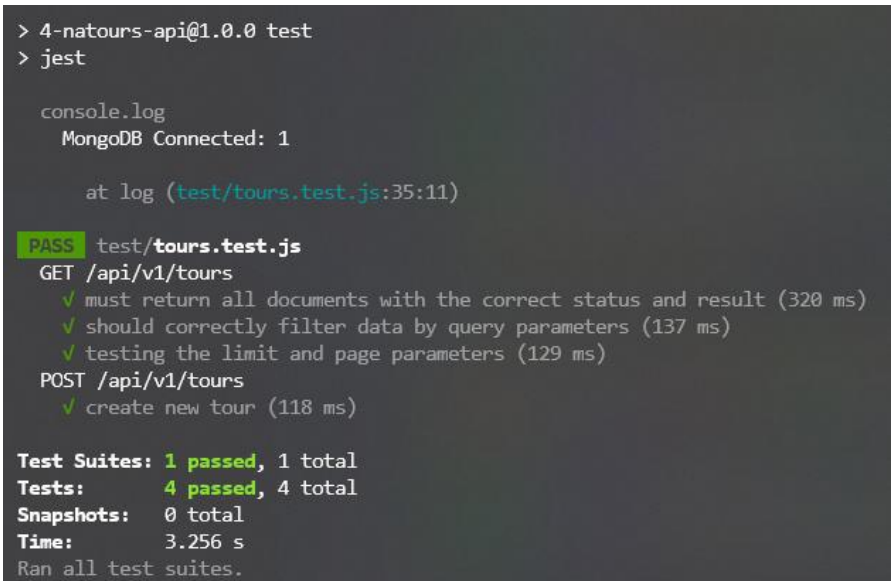

Рисунок 3.8 – Результат тестування ресурсу Tour

Наступним ресурсом для тестування було обрано User. Основою ресурсу є авторизація та реєстрація в системі. Додатково було створено сценарій для пошуку адміністраторів та створення нового користувача. Програмний код тестових сценаріїв подано в лістингу 3.7.

Лістинг 3.7 – Програмний код тестових сценаріїв для ресурсу User

```
beforeAll(async () => {
  await mongoose.connect(process.env.MONGO_URL);

  console.log('MongoDB Connected:', mongoose.connection.readyState);
// Должно быть "1"
});

afterAll(async () => {
  await mongoose.disconnect();
});

afterEach(async () => {
  await User.deleteMany({ email: user.email });
});
describe('GET /api/v1/users', () => {
  it('should get all users with the admin role', async () => {
    const response = await
request(app).get('/api/v1/users?role=admin');

    expect(response.body.status).toBe('success');
    expect(response.body.result).toBe(1);
    expect(response.body.data.data[0]).toHaveProperty('role',
'admin');
  });
});

describe('POST /api/v1/users', () => {
  it('Create new user', async () => {
    const response = await request(app)
      .post('/api/v1/users')
      .send(user)
      .expect(201);

    expect(response.body.status).toBe('success');
    expect(response.body.data.data.name).toBe(user.name);
  });

  it('signup', async () => {
    const response = await request(app)
      .post('/api/v1/users/signup')
      .send(user)
      .expect(201);

    expect(response.body.status).toBe('success');
    expect(response.body).toHaveProperty('token');
    expect(response.body.data.user.email).toBe(user.email);
  });

  it('login', async () => {
```

```
    const response = await request(app)
      .post('/api/v1/users/login')
      .send({ email: 'm.portyannikov@gmail.com', password: 'pass1234'
})
      .expect(200);

    expect(response.body.status).toBe('success');
    expect(response.body).toHaveProperty('token');

expect(response.body.data.user.email).toBe('m.portyannikov@gmail.com')
;
  });
});
```

Запустивши тестові сценарії, було отримано інформацію про успішне виконання тестів для ресурсу User, що подано на рисунку 3.9. У випадку помилок, тести допоможуть виявити проблемні місця, що дозволить ефективно вирішити їх на ранніх етапах.

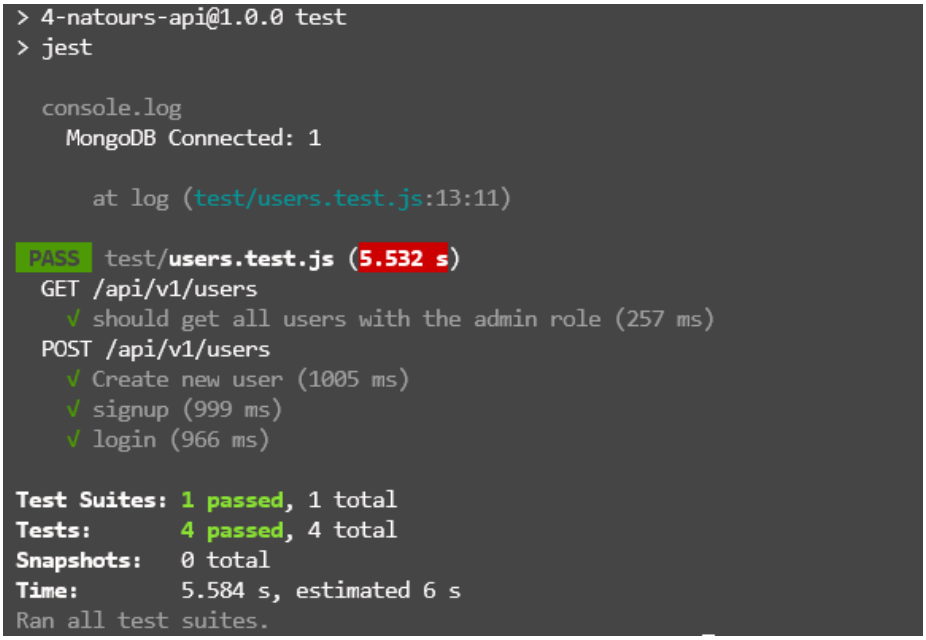

Рисунок 3.9 – Результат тестування ресурсу User

3.8 Створення графічного інтерфейсу користувача

Створення графічного інтерфейсу користувача – це процес проєктування та реалізації інтерфейсу, що забезпечить можливість взаємодії

користувача з інформаційною системою з використанням візуальних елементів. Основна мета розробки графічного інтерфейсу полягає в створенні легкої та зрозумілої взаємодії, зменшуючи витрати та потреби вивчати складні команди та інструкції.

Процес розробки інтерфейсу полягає у визначені цілей системи та аудиторії, що буде її використовувати. На етапі проєктування розроблюються прототипи інтерфейсу та схеми, що можуть бути використані при розробці. Обов'язковим критерієм для графічного інтерфейсу є:

– зручність використання. Зручність досягається шляхом створення логічної структури сайту та використанням звичних елементів інтерфейсу;

– адаптивність. Інтерфейс користувача повинен коректно відображатись на різних пристроях;

– доступність. Інтерфейс повинен враховувати потреби користувачів з обмеженими можливостями.

На рисунку 3.10 подано графічний інтерфейс головної сторінки.

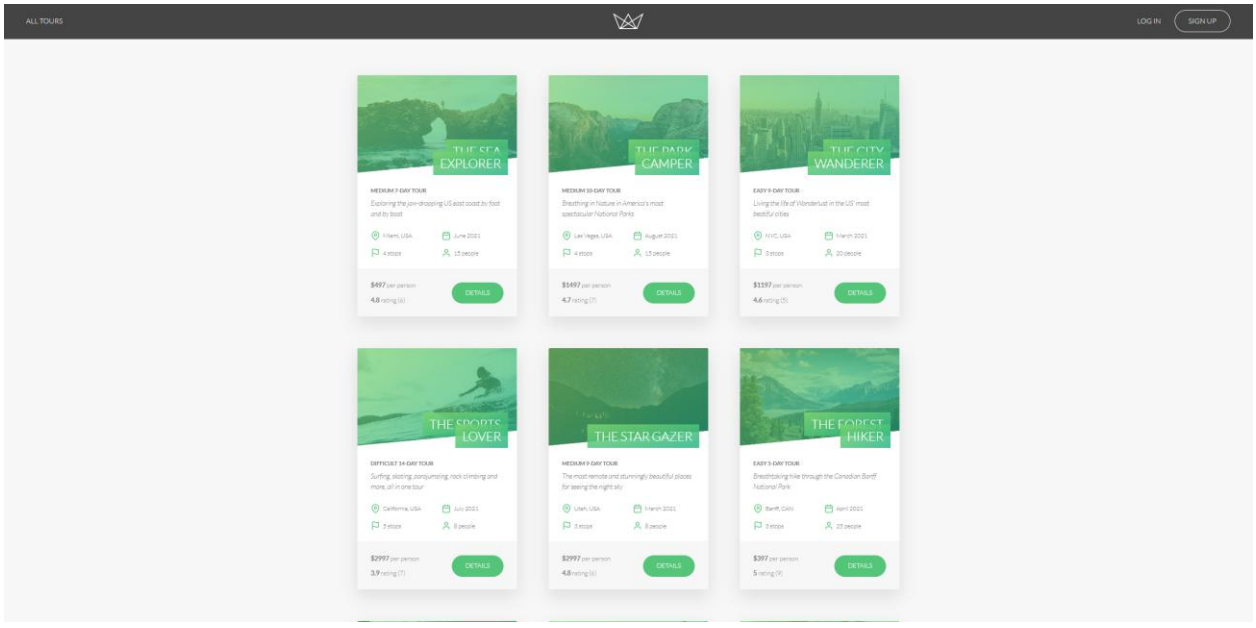

Рисунок 3.10 – Інтерфейс головної сторінки

Для реалізації інтерфейсу користувача було використано шаблонізатор pug. Шаблонізатор – це програмне забезпечення, що дозволяє

використовувати HTML-шаблони для генерації динамічних сторінок [17]. Програмний код головної сторінки подано в лістингу 3.8.

Лістинг 3.10 – Програмний код головної сторінки

```
extends base

block content
  main.main
    .card-container
      //-Перебираем массив туров и для каждого тура создаем свою
карточку
      each tour in tours
        .card
          .card__header
            .card__picture
              .card__picture-overlay  
              img.card__picture-
img(src=`/img/tours/${tour.imageCover}`, alt=`${tour.name}`)
            h3.heading-tertirary
              span= tour.name

          .card__details
            h4.card__sub-heading= `${tour.difficulty}
${tour.duration}-day tour`
            p.card__text= tour.summary
            .card__data
              svg.card__icon
                use(xlink:href='/img/icons.svg#icon-map-pin')
              span= tour.startLocation.description
            .card__data
              svg.card__icon
                use(xlink:href='/img/icons.svg#icon-calendar')
              span= tour.startDates[0].toLocaleString('en-us', {month:
'long', year:'numeric'})
            .card__data
              svg.card__icon
                use(xlink:href='/img/icons.svg#icon-flag')
              span= `${tour.locations.length} stops`
            .card__data
              svg.card__icon
                use(xlink:href='/img/icons.svg#icon-user')
              span= `${tour.maxGroupSize} people`

          .card__footer
            p
              span.card__footer-value= `$${tour.price}`
              |
              span.card__footer-text per person
```

```
            p.card__ratings
              span.card__footer-value= tour.ratingsAverage
              |
              span.card__footer-text= `rating
(${tour.ratingsQuantity})`
            a.btn.btn--green.btn--small(href=`/tour/${tour.slug}`)
Details
```

Таким чином, створення графічного інтерфейсу користувача – це один із найголовніших процесів, що поєднує в собі знання в галузях дизайну, проєктування та написання програмного коду інформаційної системи. Результатом проєктування системи є створена серверна частина застосунку, яка використовує архітектуру MVC. На стороні сервера забезпечується обробка запитів, взаємодія з базою даних і генерація сторінок веб-додатку для відображення інтерфейсу. Генерація сторінок може виконуватись як статично, так і динамічно з використанням технології Node.js. Окрім серверної частини, важливу роль відіграє фронтенд-розробка, яка включає створення інтерактивних компонентів, забезпечення адаптивності інтерфейсу для різних пристроїв та інтеграцію з бекендом через API.

Таким чином, створення інтерфейсу не обмежується лише графічним дизайном – це комплексна задача, що охоплює проєктування, технічну реалізацію та оптимізацію для забезпечення високої продуктивності, доступності та зручності використання.

ВИСНОВКИ

В ході роботи було проведено аналіз предметної області для визначення якості та безпеки розроблюваної інформаційної системи. Для уточнення взаємодії елементів системи між собою та їх взаємозв’язків було створено функціональну модель для деталізації процесів. Отримання загальної інформації про систему та створення UML діаграм дозволило деталізувати основні процеси системи.

Визначення вимог і потреб користувачів дозволило обрати оптимальних підхід до розробки програмного забезпечення. Створення моделей бази даних, контролерів та представлень стали ключовими частинами для ефективної роботи системи.

Велика частина уваги була приділена для розподілу бізнес-логіки та логіки додатка, що в свою чергу дозволило забезпечити користувачів надійністю системи. Хешування паролів, встановлення термінів дії токенів та токенів скидання паролю захищають особистий профіль користувача від несанкціонованого доступу.

Таким чином, розроблена система забезпечує користувачів необхідними функціями і можливостями для ефективної взаємодії, дозволяючи виконувати завдання з мінімальними витратами часу та зусиль. Вона сприяє підвищенню продуктивності, забезпечує інтуїтивний досвід користувача та адаптується до потреб і запитів цільової аудиторії, що є ключовим для успішного функціонування сучасних інформаційних систем.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Петров Е.Г., Новожилова М.В., Гребеннік І.В., Соколова Н.А. Методи та засоби прийняття рішень у соціально-економічних та технічних системах: Херсон: Олді – плюс, 2003. 380 с.

2. Петров Е.Г., Новожилова М.В., Гребеннік І.В. Методи і засоби прийняття рішень у соціально-економічних системах, Київ: Техніка, 2004. 256 с.

3. Гребеннік І.В., Вишняк М.Ю., Іванов В.Г., Імангулова З.А., Калита Н.І. Елементи системного проєктування (за редакцією І.В.Гребенніка): навч. посібник. Харків: ХНУРЕ, 2016. 322 с.

4. Наконечний О. Г., Гребеннік І. В., Романова Т. Є., Тевяшев А. Д., Методи прийняття рішень: навч. посібник. Харків: ХНУРЕ, 2016. 132 с.

5. Гребеннік І.В., Коваленко А.І., Міщеряков Ю.В., Решетнік В.М., Титов С.В. Системне програмування. Харків: ХНУРЕ, 2017. 374 с.

6. Нефьодов Л. І., Невлюдов І. Ш., Безкоровайний В. В. CALS-технології і системи: навч. посібник. Харків: ХНУРЕ, 2021. 272 с.

7. Системне програмування. Підручник для студентів спеціальностей 122 – «Комп'ютерні науки», 151 – «Автоматизація та комп'ютерно-інтегровані технології» / І.В. Гребеннік, А.І.Коваленко, С.В.Тітов, Ю.В.Міщеряков, В.М. Решетнік. Харків: ХНУРЕ, 2017. 376 с.

8. Основи Інтернет-технологій : навч. посіб. / В.М. Бредіхін, В.В. Карасюк, О.В. Карпухін, Ю.В. Міщеряков; під ред. О.В. Карпухіна. Харків: СМІТ, 2010. 394 с.

9. Password hashing in Node.js with bcrypt - LogRocket Blog. LogRocket Blog. URL: https://blog.logrocket.com/password-hashing-node-js-bcrypt/ (дата звернення: 17.011.2024).

10. MongoDB - Database, Collection, and Document - GeeksforGeeks. GeeksforGeeks. URL: https://www.geeksforgeeks.org/mongodb-database-collection-and-document/ (дата звернення: 13.10.2024).

11. Express Tutorial Part 4: Routes and controllers - Learn web development | MDN. MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/routes (дата звернення: 13.06.2023).

12. Controllers in NodeJS | CodeForGeek. CodeForGeek. URL: https://codeforgeek.com/controllers-in-nodejs/ (дата звернення: 19.11.2024).

13. Databases and Collections — MongoDB Manual. MongoDB: The Developer Data Platform | MongoDB. URL: https://www.mongodb.com/docs/manual/core/databases-and-collections/ (дата звернення: 11.11.2024).

14. Express 4.x - API Reference. Express - Node.js web application framework. URL: https://expressjs.com/en/api.html (дата звернення: 13.11.2024).

15. Mongoose v7.2.4: Documents. Mongoose ODM v7.2.4. URL: https://mongoosejs.com/docs/documents.html (дата звернення: 13.11.2024).

16. Getting Started – Pug. URL: https://pugjs.org/api/getting-started.html (дата звернення: 17.11.2024).

17. Стадии цикла разработки ПО - QALight. QALight. URL: https://qalight.ua/ru/baza-znaniy/stadii-tsikla-razrabotki-po/ (дата звернення: 17.11.2024).

18. Тестування фреймворків · Jest. Jest Delightful JavaScript Testing. URL: https://jestjs.io/uk/docs/testing-frameworks (дата звернення: 17.11.2024).

19. Автоматизоване тестування - QALight. QALight. URL: https://qalight.ua/baza-znaniy/avtomatizovane-testuvannya/ (дата звернення: 17.11.2024).