Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук
(повна назва)

Кафедра програмної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

Програмна система для подорожей визначними місцями. Back-end.

(тема)

Виконав:
студент 4 курсу, групи ПЗПІ-20-7

Піняєв Є.В.
(прізвище, ініціали)

Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва спеціальності)
Тип програми освітньо-професійна
Освітня програма Програмна інженерія
(повна назва освітньої програми)

Керівник ст.викл. кафедри ПІ Онищенко К.Г.
(посада, прізвище, ініціали)

Допускається до захисту
Зав. кафедри _____ З.В.Дудар
(підпис) (прізвище, ініціали)

2024 р.

Харківський національний університет радіоелектроніки

Факультет комп’ютерних наук
Кафедра програмної інженерії
Рівень вищої освіти перший (бакалаврський)
Спеціальність 121 – Інженерія програмного забезпечення
Тип програми Освітньо-професійна
Освітня програма Програмна Інженерія
(шифр і назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«___» _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Піняєву Євгенію Володимировичу
(прізвище, ім’я, по батькові)

1. Тема роботи Програмна система для подорожей визначними місцями. Back-end.

Затверджена наказом по університету від 20.05.2024 № 471 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 18.06.2024

3. Вихідні дані до роботи Розробити серверну частину програмної системи для планування та організації подорожей визначними місцями за допомогою Node.js, Express.js, MongoDB та CMS Contentful.

4. Перелік питань, що потрібно опрацювати в роботі

Вступ, аналіз предметної галузі, формування вимог до програмної системи, архітектура та проектування програмного забезпечення, опис прийнятих програмних рішень, тестування розробленого програмного забезпечення, висновки, додатки.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної галузі	15.04.2024	*виконано*
2	Формування вимог до ПЗ	20.04.2024	*виконано*
3	Проектування ПЗ	22.04.2024	*виконано*
4	Розробка ПЗ	20.05.2024	*виконано*
5	Тестування ПЗ	22.05.2024	*виконано*
6	Оформлення пояснювальної записки	03.06.2024	*виконано*
7	Перевірка роботи на антиплагіат та проходження нормоконтролю	04.06.2024	*виконано*
8	Оцінка роботи рецензентом, отримання відгуку від керівника кваліфікаційної роботи	14.06.2024	*виконано*
9	Попередій захист роботи	14.06.2024	*виконано*
10	Здача роботи до електронного архіву, допуск роботи до захисту завідувачем кафедри	15.06.2024	*виконано*
11	Захист кваліфікаційної роботи	18.06.2024	

Дата видачі завдання 08 квітня 2024р.

Студент _____ Піняєв Є.В.
(підпис)

Керівник роботи _____ ст.викл. кафедри ПІ Онищенко К.Г.

(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ / ABSTRACT

Пояснювальна записка до кваліфікаційної роботи бакалавра, 79 стор., 10 рис., 3 табл., 8 джерел.

ПОДОРОЖІ, ПРОГРАМНА СИСТЕМА, БЕКЕНД, NODE.JS, EXPRESS.JS, MONGODB, CONTENTFUL, HEADLESS CMS

Об'єктом розробки є серверна частина програмної системи для подорожей визначними місцями.

Мета розробки – створити надійний, масштабований та легко інтегрований бекенд для підтримки функціоналу програмної системи, такого як реєстрація користувачів, зберігання та пошук інформації про визначні місця, управління подорожами та їх планування.

Методи вирішення – Node.js у поєднанні з Express.js для створення API, MongoDB як база даних для зберігання інформації про користувачів та місця, Contentful як headless CMS для керування вмістом.

У результаті розробки буде створено серверну частину програмної системи, яка забезпечуватиме функціональність для мобільного додатку та веб-сайту і дозволятиме користувачам ефективно планувати та організовувати свої подорожі.

TRAVEL, SOFTWARE SYSTEM, BACKEND, NODE.JS, EXPRESS.JS, MONGODB, CONTENTFUL, HEADLESS CMS

The object of development is a software system for sightseeing travels.

The purpose of the development is to create a reliable, scalable, and easily integrated backend to support the functionality of the software system, such as user registration, storage and search of information about places of interest, travel management and planning.

Solution methods – Node.js combined with Express.js for creating an API, MongoDB as a database for storing user and location information, Contentful as a headless CMS for content management.

As a result of the development, the server-side of the software system will be created, providing functionality for the mobile app and website, allowing users to effectively plan and organize their travels.

Я, Піняєв Євгеній Володимирович, студент гр. ПЗПІ-20-7, здобувач вищої освіти на першому (бакалаврському) рівні кафедри «Програмна інженерія», заявляю: моя кваліфікаційна робота на тему «Програмна система для подорожей визначними місцями. Back-end.», що буде представлена до екзаменаційної комісії для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в електронному архіві відкритого доступу ElAr KhNURE. Усі запозичення з друкованих та електронних джерел мають відповідні посилання.

Я ознайомлений з діючим положенням «Про протидію академічному плагіату в ХНУРЕ», згідно з яким виявлення плагіату є підставою для відмови до допуску кваліфікаційної роботи до захисту та застосування дисциплінарних заходів.

ЗМІСТ

ВСТУП

У сучасному світі, на тлі постійного розвитку технологій та швидкого темпу життя, подорожі стають не лише засобом відпочинку, але й важливим елементом саморозвитку та культурного обміну. Прагнення відкривати нові горизонти, знайомитися з різноманітністю світу та збагачувати себе неповторними враженнями населює сучасного мандрівника. Однак, разом з романтикою відкриттів приходить і складність організації подорожі.

Планування маршруту, вибір відповідних місць для відвідування, оптимізація бюджету – це лише частина завдань, які стоять перед подорожуючим. Спроба впоратися з усіма цими аспектами може перетворитися на виклик, особливо для тих, хто прагне самостійно планувати свої подорожі.

Саме тому розробка програмної системи для планування подорожей до визначних місць набуває великого значення в сучасному світі. Ця система має на меті спростити процес планування, забезпечуючи користувачам доступ до комплексного інструментарію для організації своїх подорожей.

Метою даної роботи є створення серверної частини програмної системи для подорожей визначними місцями, яка забезпечить надійну та масштабовану основу для реалізації необхідної функціональності. Основними завданнями є розробка API для взаємодії з клієнтськими додатками, створення бази даних для зберігання інформації про користувачів та місця відвідування, інтеграція з headless CMS для керування вмістом, а також забезпечення належного рівня безпеки для зберігання та обробки даних користувачів.

Серверна частина, яка буде розроблена в рамках даної роботи, стане основою для створення повноцінної програмної системи для подорожей визначними місцями. Вона буде інтегрована з клієнтською частиною, що включатиме веб-сайт та мобільний додаток, надаючи мандрівникам різного рівня досвіду – від новачків до досвідчених туристів – потужний інструмент для ефективного та зручного самостійного планування подорожей.

Розроблена програмна система матиме широке коло застосування в туристичній галузі, а також буде корисною для індивідуальних мандрівників. Вона

допоможе спростити процес планування подорожей, зробивши його більш організованим та персоналізованим відповідно до індивідуальних потреб та уподобань користувачів. Завдяки комплексному підходу та інтеграції різноманітних функцій, система стане надійним помічником для мандрівників на всіх етапах підготовки до подорожі – від пошуку визначних місць до формування маршрутів, складання бюджету та збереження спогадів.

Результати даної роботи можуть бути застосовані в туристичній індустрії для підвищення ефективності планування подорожей та покращення досвіду клієнтів. Крім того, система буде корисною для індивідуальних мандрівників, які прагнуть самостійно організовувати свої подорожі, забезпечуючи їм зручні інструменти для пошуку, планування та зберігання інформації про визначні місця та маршрути.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Аналіз предметної галузі

Подорожі – це не лише відпочинок, а й важливий елемент розвитку особистості, пізнання культурного та історичного надбання світу. У сучасному глобалізованому світі, коли кордони стають все більш прозорими, а доступність різних куточків планети зростає, мандрівки набувають особливого значення. Вони дозволяють не тільки відкривати нові горизонти, а й поглиблювати власні знання, розширювати світогляд та збагачувати свій досвід.

З розвитком сучасних технологій та інтернет-ресурсів, сфера туризму переживає значний розквіт, пропонуючи різноманітні сервіси для мандрівників усіх категорій. Від пошуку авіаквитків та бронювання готелів до отримання рекомендацій щодо визначних пам'яток та розваг – цифрові інструменти стають невід'ємним супутником сучасного туриста.

Аналізуючи предметну галузь подорожей, було виявлено ряд ключових аспектів, які потребують уваги та вдосконалення. По-перше, це потреба у зручному та ефективному способі планування подорожей, що враховує індивідуальні вподобання, інтереси та бюджет користувача. Від вибору маршруту до бронювання готелю, кожен етап подорожі потребує уважного розгляду та виважених рішень, щоб забезпечити максимальний комфорт та задоволення від мандрівки.

По-друге, важливим аспектом є доступ до достовірної, актуальної та повної інформації про визначні місця, пам'ятки та культурні спадщини кожного регіону. Інформованість мандрівника про потенційні маршрути, історичне та культурне значення різних об'єктів, а також рекомендації щодо заходів та розваг забезпечують якісне планування та незабутні враження від подорожі.

По-третє, з урахуванням постійного зростання популярності туризму, виникає проблема збереження навколишнього середовища та збалансованого використання туристичних ресурсів. Екологічно стійкий туризм стає важливим завданням не лише для туристичних компаній, а й для суспільства в цілому. Необхідно знаходити шляхи для мінімізації негативного впливу на місцеві екосистеми та культури, а також сприяти розвитку відповідального туризму.

Аналізуючи всі ці аспекти, було визначено ключову проблему – відсутність цілісного та комплексного підходу до планування подорожей, який би задовольняв усі потреби мандрівників. У відповідь на зростаючий інтерес до подорожей та їх планування з'явилося значна кількість веб-додатків та мобільних застосунків, які пропонують різноманітні сервіси для туристів. Проте, варто відзначити, що більшість із них фокусуються лише на певних аспектах подорожі, таких як пошук готелів чи ресторанів, і не надають комплексного рішення, яке б охоплювало всі етапи планування та враховувало індивідуальні потреби користувачів.

Розглянемо деякі з існуючих рішень для планування подорожей. Tripadvisor [1], один з найпопулярніших туристичних веб-сайтів, надає широкий спектр інформації про готелі, ресторани, та визначні пам'ятки (рисунок 1.1). Цей ресурс є надійним джерелом відгуків та рекомендацій від інших мандрівників. Однак, хоча Tripadvisor добре підходить для пошуку інформації про визначні місця, він не забезпечує можливості створення персоналізованих маршрутів та планів подорожей, а також не дозволяє планувати бюджет.

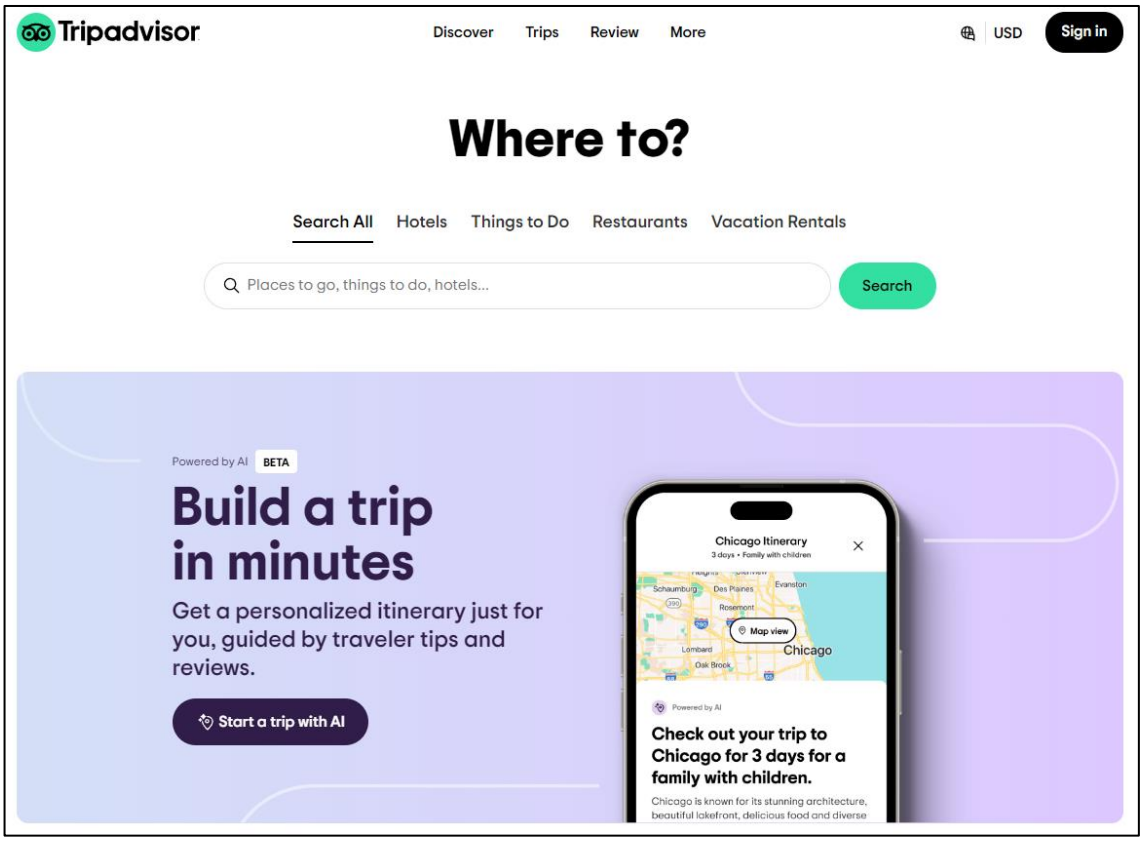


Рисунок 1.1 – Інтерфейс сайту та мобільного застосунку

Ще одним рішенням є Google Trips [2], мобільний додаток від компанії Google, який допомагає організувати інформацію про подорож у одному місці. Він автоматично збирає дані про резервування готелів та авіаквитків з облікового запису Google користувача і надає рекомендації щодо визначних пам'яток та заходів. Однак, також як і Tripadvisor, Google Trips обмежується в можливостях створення персоналізованих маршрутів та планів подорожей та не дозволяє планувати бюджет подорожі.

Розглянемо ще одне рішення у сфері планування подорожей – Airbnb [3] (рисунок 1.2). Цей сервіс, відомий своєю платформою для пошуку та бронювання житла, також надає деякі інструменти для планування мандрівок.

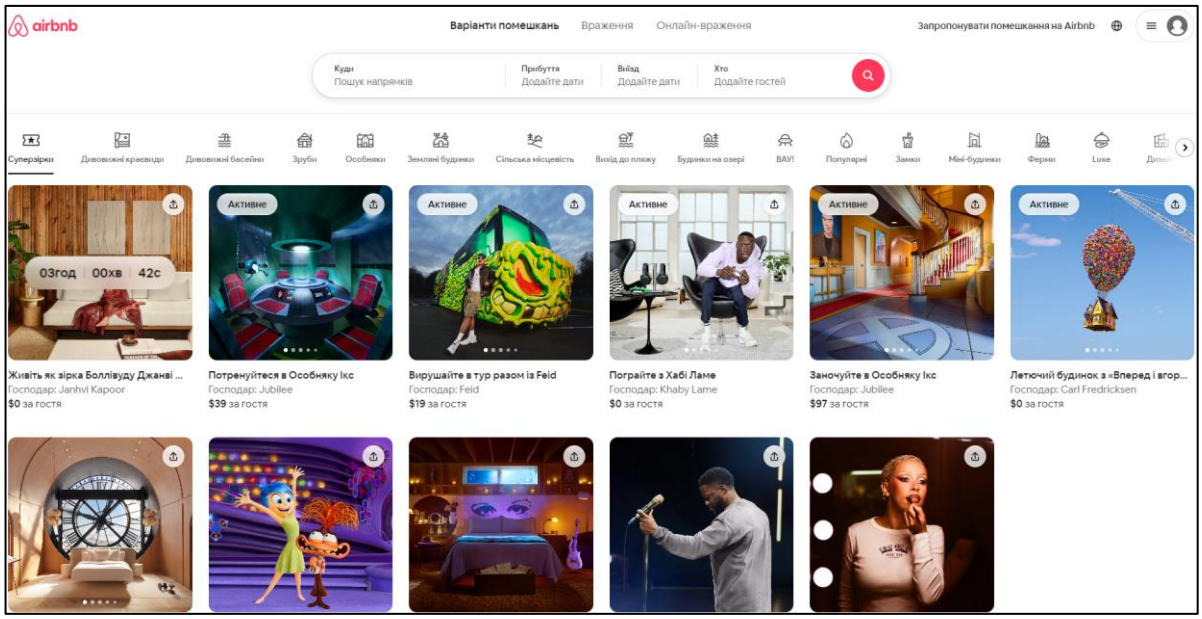


Рисунок 1.2 – Інтерфейс сайту Airbnb

Airbnb дозволяє користувачам знаходити та бронювати різноманітне житло у будь-якій точці світу, від звичайних кімнат до ексклюзивних будинків та апартаментів. Однак, на відміну від традиційних готелів, Airbnb також може забезпечити користувачам більш аутентичне та місцеве переживання. Крім того, сервіс Airbnb Experiences дозволяє знаходити та бронювати унікальні та цікаві враження, організовані місцевими жителями.

Хоча Airbnb зосереджується переважно на житлі та місцевих враженнях, він також надає деякі інструменти для планування мандрівок. Однак, ці інструменти можуть бути обмеженими у порівнянні з іншими сервісами, оскільки Airbnb зазвичай не забезпечує можливості створення докладних маршрутів та планів подорожей, а також не дозволяє планувати бюджет подорожі.

Хоча Airbnb є популярним сервісом для пошуку та бронювання житла та місцевих вражень, він не завжди відповідає потребам користувачів щодо комплексного планування подорожей. Бажаною є розробка програмної системи, яка забезпечить повний спектр сервісів для ефективного та зручного планування мандрівок.

Отже, аналізуючи існуючі рішення, можна зробити висновок, що вони не забезпечують комплексного підходу до планування подорожей, оскільки не мають можливості створювати персоналізовані маршрути, планувати бюджет та зберігати спогади від поїздок. Таким чином, існує потреба у розробці програмної системи, яка задовольнятиме всі ці потреби мандрівників та забезпечить повний спектр сервісів для ефективного та зручного планування подорожей.

1.2 Виявлення та вирішення проблем

Під час ретельного аналізу предметної галузі подорожей та існуючих рішень для планування мандрівок було виявлено низку суттєвих проблем, які значно ускладнюють та роблять менш ефективним процес організації поїздок до визначних місць. Однією з найбільш нагальних проблем є відсутність єдиної універсальної платформи, яка б могла забезпечити всі необхідні етапи планування подорожі в комплексі.

Натомість, більшість наявних на ринку веб-сайтів та додатків спеціалізуються лише на окремих аспектах процесу підготовки до мандрівки, таких як пошук та бронювання авіаквитків чи резервація готелів. Це змушує потенційних мандрівників використовувати одночасно декілька різних ресурсів, що є значно менш зручним та ефективним, аніж наявність єдиного комплексного рішення. Саме тому розробка програмної системи, яка б об'єднала в собі всі необхідні функції для

планування подорожей, стане важливим кроком у вирішенні цієї актуальної проблеми.

Ще однією вагомою проблемою, з якою стикаються мандрівники, є обмежені можливості для створення власних, персоналізованих маршрутів та планів подорожей. Більшість існуючих на даний момент рішень пропонують лише стандартні, заздалегідь визначені маршрути або рекомендації, не надаючи користувачам можливості самостійно формувати індивідуальні плани відповідно до їхніх особистих уподобань, інтересів та пріоритетів. Саме тому впровадження функціоналу для створення власних маршрутів та персоналізованих планів подорожей, з можливістю вибору дат, визначних місць, заходів та інших складових, стане вкрай важливим етапом у вирішенні цієї проблеми та задоволенні потреб сучасних мандрівників.

Також одним невирішеним на даний момент питанням у сфері планування подорожей є відсутність зручних та ефективних інструментів для планування бюджету майбутньої поїздки. Більшість наявних рішень не надають користувачам можливості попередньо спланувати бюджет своєї подорожі, розрахувати та контролювати очікувані витрати на проживання, транспорт, харчування, розваги та інші складові. Це може призвести до фінансових несподіванок та незапланованих витрат під час самої мандрівки. Тому розробка зручних інструментів для ретельного планування бюджету подорожі, які дозволять встановлювати ліміти витрат та контролювати фінансові аспекти мандрівки, стане важливою складовою нової програмної системи.

Після повернення з подорожі мандрівники часто стикаються з проблемою збереження та подальшого перегляду своїх спогадів, фотографій, нотаток та інших матеріалів, пов'язаних з їхньою поїздкою. Впровадження спеціального функціоналу для зручного зберігання та перегляду спогадів від минулих подорожей безпосередньо в межах однієї програмної системи допоможе вирішити цю проблему та забезпечити користувачам можливість зберігати та переглядати всі свої враження та матеріали в одному місці.

Нарешті, більшість існуючих на даний момент рішень для планування подорожей не враховують індивідуальні уподобання, інтереси та пріоритети конкретних користувачів, що призводить до надання стандартизованих, неперсоналізованих рекомендацій. Ця проблема є вкрай суттєвою, адже кожен мандрівник має свої унікальні вподобання, захоплення та бажання, які необхідно враховувати під час планування подорожі. Надання загальних, універсальних рекомендацій може призвести до того, що запропоновані маршрути, визначні місця чи заходи не відповідатимуть особистим інтересам користувача, що зменшить задоволення від поїздки.

Тому впровадження функціоналу для врахування індивідуальних уподобань користувача та надання персоналізованих рекомендацій на основі їхніх інтересів стане невід'ємною складовою нової програмної системи для планування подорожей до визначних місць. Завдяки такому підходу кожен мандрівник зможе отримувати рекомендації, спеціально підібрані відповідно до його особистих вподобань, захоплень та пріоритетів під час подорожі. Це забезпечить більш індивідуалізований та персоналізований досвід планування мандрівки, що дозволить максимально задовольнити потреби користувачів та зробити їхню подорож незабутньою.

Комплексне вирішення зазначених проблем, таких як відсутність єдиної платформи, обмежені можливості для створення персоналізованих маршрутів, брак інструментів для планування бюджету, складнощі зі збереженням спогадів та неврахування індивідуальних вподобань користувачів, стане вагомим кроком у розвитку програмної системи для планування подорожей до визначних місць. Така система дозволить мандрівникам отримати більш зручний, персоналізований та ефективний досвід підготовки та проведення своїх поїздок, максимально задовольняючи їхні індивідуальні потреби та забезпечуючи незабутні враження.

1.3 Постановка задачі

Метою даної кваліфікаційної роботи є розробка високопродуктивного та надійного бекенду для програмної системи планування подорожей визначними місцями. Ґрунтуючись на ретельному аналізі предметної галузі подорожей та виявлених під час нього ключових проблем, бекенд системи відіграватиме фундаментальну роль, забезпечуючи всю серверну логіку, обробку даних, взаємодію з базою даних, а також надаючи необхідний програмний інтерфейс (API) для безперебійної інтеграції з фронтенд та мобільною частинами системи.

Одним з головних завдань, які необхідно буде вирішити в рамках розробки бекенду цієї програмної системи, є реалізація потужної серверної частини з використанням передових технологій, таких як Node.js [4], Express.js [5] та MongoDB [6]. Комбінація цих інструментів забезпечить високу швидкодію, масштабованість та надійність функціонування серверної частини, що є вкрай важливим для забезпечення належного рівня користувацького досвіду та оперативного опрацювання великих обсягів даних.

Крім того, вагомим аспектом розробки бекенду стане створення ефективної системи управління контентом (CMS) на базі платформи Contentful. Ця система буде відповідати за зручне зберігання, структурування та управління всім необхідним для подорожей контентом, таким як докладна інформація про визначні місця, маршрути, фотографії, відомості про користувачів та їхні персональні дані.

Наступним ключовим завданням у процесі розробки бекенду є створення потужного, гнучкого та зручного програмного інтерфейсу (API), який забезпечить безперебійну взаємодію з клієнтською та мобільною частинами системи. Цей API надаватиме доступ до всіх необхідних функцій, таких як пошук місць відвідування, створення та редагування маршрутів, управління бюджетом подорожі, зберігання та перегляд спогадів від минулих поїздок тощо.

Ще одним важливим аспектом, який необхідно буде врахувати в процесі розробки бекенду, є питання безпеки даних та конфіденційності інформації користувачів. Для цього планується впровадити надійні заходи захисту, такі як шифрування даних, механізми автентифікації та авторизації, а також інші необхідні

заходи для забезпечення цілісності, конфіденційності та доступності персональних даних користувачів.

Нарешті, невід'ємною частиною успішної розробки бекенду є оптимізація його роботи для досягнення мінімального часу відповіді на запити та ефективного використання ресурсів сервера. Це дозволить забезпечити швидку та безперебійну роботу системи навіть під час пікових навантажень, гарантуючи високий рівень продуктивності та задоволеності користувачів.

Успішне вирішення всіх зазначених завдань у процесі розробки бекенду програмної системи для подорожей визначними місцями дозволить створити надійну, високопродуктивну та функціональну серверну частину, яка стане серцевиною всієї системи. Цей потужний бекенд забезпечуватиме ефективну обробку та зберігання даних, безперебійну взаємодію з іншими компонентами системи, а також надаватиме користувачам зручний, безпечний та надійний інструмент для планування, організації та проведення незабутніх подорожей.

1.3.1 Цільова аудиторія

Програмна система для планування подорожей визначними місцями, розроблена з урахуванням потреб широкого кола користувачів, стане незамінним помічником для любителів мандрів. Ця універсальна платформа орієнтована як на досвідчених туристів, так і на новачків, пропонуючи різноманітні інструменти та функціонал, що дозволять зробити процес планування захопливих подорожей максимально зручним та ефективним.

Основними категоріями цільової аудиторії є:

– Індивідуальні мандрівники: Самостійні туристи, які прагнуть самостійно планувати кожен аспект своєї подорожі, зможуть скористатися широким спектром можливостей платформи. Незалежно від рівня досвіду, вони отримають зручний інструмент для вибору визначних місць, складання маршрутів, контролю бюджету, а також персоналізовані рекомендації та поради від системи.

– Молоді мандрівники: Для тих, хто вважає подорожі невід'ємною частиною свого способу життя та можливістю для саморозвитку, ця система стане справжнім

відкриттям. Завдяки інноваційним рішенням та персоналізованим рекомендаціям, процес планування подорожей стане більш захопливим та креативним. Крім того, користувачі зможуть ділитися своїми враженнями та досвідом з іншими мандрівниками в рамках платформи.

– Сімейні пари або групи друзів: Для тих, хто планує подорожувати компанією, система пропонує зручні інструменти для спільного створення планів подорожей. Учасники зможуть додавати нотатки, ділитися спогадами, обговорювати маршрути та узгоджувати бюджет, враховуючи інтереси кожного члена групи.

– Любителі історії, культури та визначних пам'яток: Ця категорія охоплює користувачів будь-якого віку, які захоплюються вивченням історії, культури та відвідуванням визначних місць у різних куточках світу. Для них платформа надасть детальну інформацію про пам'ятки, можливість створювати тематичні маршрути та отримувати рекомендації щодо цікавих заходів та подій.

– Туристичні агентства та організатори подорожей: Хоча система орієнтована переважно на індивідуальних мандрівників, вона також може стати корисним інструментом для професіоналів туристичної галузі. Туристичні агентства та організатори подорожей зможуть використовувати платформу для створення персоналізованих турів, надання клієнтам додаткової інформації про визначні місця та підвищення рівня обслуговування загалом.

Врахування потреб різних категорій користувачів під час розробки цієї програмної системи забезпечить її універсальність, зручність та корисність для широкого кола мандрівників. Завдяки своїй функціональності та орієнтованості на задоволення індивідуальних потреб користувачів, ця платформа має всі шанси стати популярним та затребуваним інструментом на ринку туристичних послуг.

2 ФОРМУВАННЯ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ

Бекенд частина є невід'ємною та надзвичайно важливою складовою програмної системи для подорожей визначними місцями. Вона виступає в ролі потужного фундаменту, на якому базується функціонування всієї системи та від якого залежить її ефективність, продуктивність та надійність.

Основною відповідальністю бекенду є забезпечення безперебійної обробки запитів користувачів, управління та зберігання даних, а також налагодження взаємодії з базою даних та іншими зовнішніми сервісами. Саме завдяки злагодженій роботі бекенд-компонентів користувачі можуть отримувати доступ до необхідної інформації, здійснювати пошук визначних місць, створювати маршрути подорожей та виконувати інші дії в межах системи.

Правильно спроектований та ретельно реалізований бекенд є запорукою стабільної роботи всієї системи. Він має бути розроблений з урахуванням найкращих практик та новітніх технологій, щоб забезпечити високу швидкодію, безпеку та масштабованість рішення. Крім того, бекенд повинен бути гнучким та легко адаптуватися до нових вимог чи змін, що можуть виникнути в майбутньому.

2.1 Технології та інструменти

Розробка бекенду програмної системи для подорожей визначними місцями вимагає ретельного вибору технологічного стеку, який забезпечить високу продуктивність, масштабованість та надійність рішення. У цьому контексті необхідно зосередитись на інструментах та фреймворках, які дозволять побудувати потужний та гнучкий серверний компонент, здатний ефективно обробляти запити користувачів, взаємодіяти з базою даних та іншими сервісами.

Для розробки фронтенд частини будуть використані такі технології та інструменти:

– Node.js. Node.js є платформою з відкритим вихідним кодом, побудованою на JavaScript-двигуні V8 (розробленим Google для Chrome). Вона призначена для створення масштабованих мережевих додатків і забезпечує асинхронну модель

введення/виводу, що підвищує продуктивність та ефективність обробки великої кількості підключень. Node.js є ідеальним вибором для розробки серверної частини веб-додатків, оскільки дозволяє використовувати єдину мову програмування (JavaScript) як на клієнтській, так і на серверній стороні;

– Express.js. Express.js – це мінімалістичний та гнучкий веб-фреймворк для Node.js, який забезпечує зручний інтерфейс для створення веб-додатків та API. Він надає широкий спектр функцій для обробки HTTP-запитів, маршрутизації, управління сесіями, обробки помилок та багато іншого. Express.js дозволяє розробникам швидко розгортати робочі рішення та ефективно організовувати структуру проекту;

– MongoDB. MongoDB є масштабованою, розподіленою, документ-орієнтованою NoSQL базою даних, яка ідеально підходить для зберігання та управління структурованими даними програмної системи для подорожей. Вона дозволяє зберігати дані у вигляді гнучких JSON-подібних документів, що полегшує їх маніпуляцію та забезпечує горизонтальну масштабованість. MongoDB також підтримує репліки та шардінг, що гарантує високу доступність та продуктивність системи;

– Contentful. Contentful є хмарною системою управління контентом, яка забезпечує зручний та гнучкий спосіб створення, редагування та публікації різноманітного контенту (текстів, зображень, відео тощо). Вона пропонує потужний API для інтеграції контенту з будь-якими пристроями та платформами, що дозволяє легко інтегрувати її з бекендом програмної системи для подорожей. Contentful також надає можливості для контролю версій, співпраці та забезпечення безпеки контенту.

Комбінація цих технологій та інструментів забезпечить створення ефективного, масштабованого та надійного бекенду для програмної системи подорожей визначними місцями. Node.js та Express.js дозволять розробити високопродуктивний серверний компонент, MongoDB забезпечить гнучке та масштабоване зберігання даних, а Contentful полегшить управління контентом

системи. Такий технологічний стек відповідає сучасним вимогам та практикам веб-розробки, що гарантує високу якість та стабільність кінцевого рішення.

2.2 Структура та архітектура

Для забезпечення ефективної розробки, масштабованості та підтримки бекенду програмної системи для подорожей визначними місцями буде обрано RESTful API архітектуру [7], шаблон проектування Model-View-Controller (MVC) та використання JSON Web Token (JWT) [8] для автентифікації та авторизації.

RESTful API архітектура є стандартним підходом для створення веб-сервісів та API, який базуватиметься на принципах REST (Representational State Transfer). Вона визначатиме, як клієнтські додатки повинні взаємодіяти з серверними ресурсами за допомогою HTTP-протоколу. Основними перевагами цієї архітектури будуть чітке відокремлення клієнтської та серверної частин, що підвищить гнучкість та масштабованість системи, використання стандартних HTTP-методів (GET, POST, PUT, DELETE) для виконання операцій над ресурсами, застосування уніфікованих URL-адрес для ідентифікації ресурсів, а також безстатевий протокол, що спростить кешування та розподілене навантаження.

Для організації коду та забезпечення чіткого розділення обов'язків у бекенді використовуватиметься шаблон проектування Model-View-Controller (MVC). Цей шаблон передбачатиме поділ додатку на три основні компоненти: Model (Модель), який відповідатиме за взаємодію з базою даних та маніпуляції з даними, View (Представлення), яке відповідатиме за відображення даних для користувача у вигляді JSON-відповідей, та Controller (Контролер), який оброблятиме запити користувачів, взаємодіятиме з моделями для отримання/оновлення даних та передаватиме дані до представлень. Використання MVC забезпечить модульність, гнучкість та можливість багаторазового використання коду, а також полегшить тестування та підтримку системи.

Для забезпечення безпеки та автентифікації користувачів у бекенді використовуватиметься JSON Web Token (JWT). JWT є відкритим стандартом (RFC 7519), який визначає компактний та самодостатній спосіб безпечної передачі

інформації між сторонами у вигляді JSON-об'єкта. Він складатиметься з трьох частин: заголовка, вмісту (payload) та цифрового підпису. Основними перевагами використання JWT будуть безстатевий характер, що полегшить масштабування системи, компактний розмір, що зменшить трафік між клієнтом та сервером, можливість передавати додаткову інформацію в payload, наприклад, привілеї користувача, та перевірка цілісності токена за допомогою цифрового підпису. У програмній системі для подорожей JWT використовуватиметься для автентифікації користувачів після успішного входу в систему. Клієнтський додаток зберігатиме отриманий JWT і включатиме його в заголовок Authorization для подальших захищених запитів до API бекенду.

Така архітектура та структура бекенду забезпечить дотримання найкращих практик веб-розробки, гнучкість, масштабованість та безпеку системи, що є критично важливим для успішного функціонування програмної системи для подорожей визначними місцями.

2.3 Дизайн та користувацький інтерфейс

Для забезпечення ефективного адміністрування та керування програмною системою для подорожей визначними місцями буде розроблено адміністративну панель, призначену для використання адміністраторами системи. Ця панель буде побудована з використанням HTML та CSS, що забезпечить простий та зручний доступ до ключових функцій управління.

Дизайн адміністративної панелі матиме чітку та структуровану архітектуру, орієнтовану на забезпечення зручності використання, ефективності та продуктивності керування системою. Основними компонентами дизайну будуть:

– Навігаційне меню: Для забезпечення легкої навігації між різними розділами системи буде реалізовано навігаційне меню, розташоване у верхній частині сторінки. Воно міститиме посилання на ключові функціональні області, такі як управління користувачами, керування визначними місцями, налаштування системних параметрів тощо.

– Панель інструментів: Для надання адміністраторам зручного доступу до основних операцій управління буде передбачено панель інструментів. Вона може бути розміщена збоку або в нижній частині сторінки та містити кнопки, випадаючі списки та інші елементи керування, що дозволять виконувати такі дії, як створення нових записів, редагування або видалення існуючих.

– Інтерфейс для відображення даних: Центральна частина адміністративної панелі буде відведена під відображення різноманітних даних системи, таких як списки користувачів, переліки визначних місць, статистичні дані тощо. Ця інформація може бути представлена у вигляді таблиць, списків, діаграм або інших візуальних компонентів, залежно від типу даних та контексту використання.

– Форми для введення даних: Для полегшення процесу внесення нових записів або редагування існуючих даних в адміністративній панелі будуть присутні форми введення даних. Ці форми будуть розроблені з дотриманням принципів юзабіліті та простоти використання, забезпечуючи інтуїтивно зрозумілий інтерфейс для взаємодії адміністраторів із системою.

Загалом, дизайн адміністративної панелі буде зосереджений на забезпеченні ефективного керування та адміністрування програмної системи для подорожей, уникаючи надмірної складності та зберігаючи простоту та зрозумілість інтерфейсу. Використання HTML та CSS як базових технологій дозволить швидко розробити функціональний та зручний у використанні адміністративний інтерфейс, який відповідатиме вимогам та потребам адміністраторів системи.

2.4 Взаємодія з мобільним застосунком та фронтенд

Взаємодія між бекендом, мобільним застосунком та веб-фронтендом є критично важливим аспектом для забезпечення повної функціональності та безперебійної роботи програмної системи для подорожей визначними місцями. Ця взаємодія полягає в ефективному обміні даними між компонентами системи та надає користувачам доступ до необхідної інформації в режимі реального часу, незалежно від використовуваного пристрою. Для досягнення цієї мети буде реалізовано наступні ключові елементи:

– RESTful API для мобільного застосунку та фронтенд. Бекенд системи буде оснащений RESTful API, який забезпечить інтерфейс для взаємодії з мобільним застосунком та фронтенд. Через цей API мобільний додаток зможе надсилати різноманітні запити до серверної частини, такі як отримання списку визначних місць, збереження нових подорожей, оновлення інформації про користувача тощо. RESTful API дотримуватиметься принципів REST, використовуючи стандартні HTTP-методи (GET, POST, PUT, DELETE) для маніпуляції ресурсами та уніфіковані URL-адреси для їх ідентифікації;

– Автентифікація та авторизація з використанням JWT: Для забезпечення безпеки та конфіденційності даних при взаємодії між бекендом та мобільним застосунком буде використано JSON Web Token (JWT). Після успішної аутентифікації користувача мобільний додаток отримає JWT-токен, який міститиме закодовану інформацію про користувача та його привілеї. Цей токен повинен надсилатися в заголовку Authorization при кожному наступному захищеному запиті до бекенду, що дозволить серверу перевірити автентичність та авторизацію користувача;

– Обмін даними в форматі JSON: Для спрощення обміну даними між компонентами системи та забезпечення сумісності з різними платформами і технологіями буде використовуватися формат JSON. Дані, отримані з бекенду, будуть представлені у вигляді JSON-об'єктів, які легко конвертуються в об'єкти JavaScript на клієнтській стороні (веб-фронтенд або мобільний застосунок). Аналогічно, дані, що надсилаються з клієнтської сторони на бекенд, також будуть упаковані у JSON-формат для подальшої обробки на сервері;

– Кешування даних на клієнтській стороні: Для підвищення продуктивності та зменшення навантаження на бекенд буде реалізовано кешування даних на клієнтській стороні (веб-фронтенд та мобільний застосунок). Часто використовувані дані зберігатимуться локально на пристрої користувача. Це дозволить зменшити кількість запитів до бекенду та забезпечить швидший доступ до цих даних, підвищуючи загальну швидкодію системи.

2.5 Продуктивність та оптимізація

Забезпечення високої продуктивності та ефективної роботи програмної системи для подорожей визначними місцями є одним з ключових завдань розробки. Для досягнення цієї мети буде застосовано низку стратегій та методів оптимізації, спрямованих на підвищення швидкодії системи та забезпечення плавного та комфортного досвіду користування. Основні аспекти продуктивності та їх вплив на роботу системи включають наступне:

– Оптимізація запитів до бази даних: Ефективне використання бази даних MongoDB є критично важливим чинником для забезпечення загальної продуктивності системи. Будуть застосовані оптимізовані запити до бази даних, що дозволить мінімізувати час відповіді та забезпечити швидкий доступ до необхідної інформації. Це включатиме використання індексів, оптимізацію запитів з урахуванням специфіки даних, а також застосування методів кешування результатів запитів для подальшого повторного використання;

– Кешування даних: Для прискорення доступу до часто використовуваних даних та зменшення навантаження на базу даних буде реалізовано механізм кешування даних на стороні сервера. Це дозволить зберігати часто запитувані дані в оперативній пам'яті, забезпечуючи майже миттєвий доступ до них без необхідності звертатися до бази даних. Кешування буде застосовано для таких даних, як списки визначних місць, інформація про користувачів та інші статичні або повільно змінювані дані;

– Паралельна обробка запитів: Для оптимізації обробки запитів та підвищення швидкості відповіді сервера буде використано підхід паралельного програмування та асинхронної обробки. Це дозволить ефективно розподіляти навантаження між декількома процесами або потоками, уникаючи блокування та затримок під час виконання довготривалих операцій, таких як взаємодія з базою даних або зовнішніми сервісами.

3 АРХІТЕКТУРА ТА ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 UML проєктування ПЗ

У цьому підрозділі буде представлена серія уніфікованих діаграм моделювання (UML), які візуально ілюструють структуру та поведінку бекенд компонентів системи. UML є стандартизованою нотацією та мовою моделювання, яка забезпечує загальну візуальну репрезентацію різноманітних аспектів системи, включаючи її статичну структуру, динамічну поведінку, архітектуру та взаємодію компонентів.

Використання UML діаграм є ефективним інструментом для документування, візуалізації та комунікації архітектури, дизайну та функціональності бекенд частини системи.

3.1.1 Діаграма класів

Діаграма класів є одним з найважливіших видів діаграм в уніфікованій мові моделювання (UML), яка використовується для візуального представлення структури системи з точки зору об'єктно-орієнтованого дизайну. Ця діаграма забезпечує статичний погляд на систему, відображаючи класи, їхні атрибути, методи та відносини між ними.

Крім того, діаграма класів є важливим артефактом у процесі розробки програмного забезпечення, оскільки вона служить основою для генерації коду, документації та тестів. Вона також допомагає виявляти потенційні проблеми дизайну на ранніх стадіях розробки, дозволяючи внести необхідні зміни до того, як буде написаний великий обсяг коду.

У контексті цього проекту, діаграма класів відображає (рисунок 3.1) структуру та взаємозв'язки між різними сутностями, що беруть участь у системі подорожей та планування відпусток. Вона включає класи, такі як User, Place, Vacation, Quiz та інші, що представляють різні аспекти системи. Детальний опис та аналіз цієї діаграми класів наведено нижче.

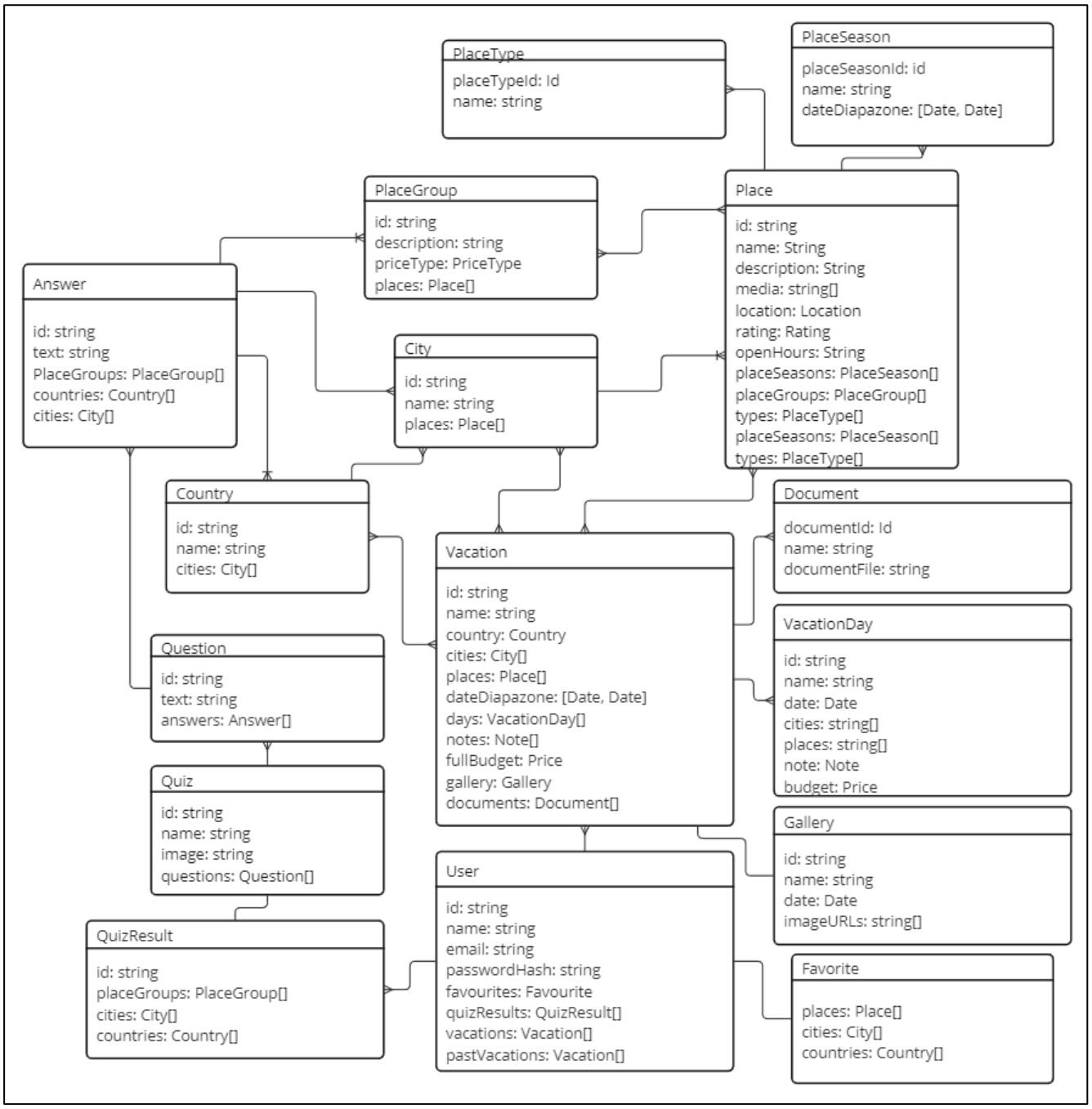


Рисунок 3.1– Діаграма класів (Виконана самостійно)

Основні класи:

– Place: Цей клас представляє собою визначне місце, яке може відвідати користувач. Він містить атрибути, такі як назва, опис, розташування, фотографії, відгуки та рейтинги;

– PlaceType: Цей клас описує тип визначного місця, наприклад, музей, пам'ятка, парк або ресторан;

- PlaceSeason: Цей клас представляє собою сезон, протягом якого визначне місце доступне для відвідування. Він містить атрибути, такі як дата початку та закінчення сезону, інформація про ціни та години роботи;
- PlaceGroup: Цей клас описує групу визначних місць, які можна об'єднати в один маршрут. Він містить атрибути, такі як назва групи, опис та список визначних місць, що входять до неї;
- City: Цей клас представляє собою місто, в якому розташовані визначні місця. Він містить атрибути, такі як назва, опис, розташування, фотографії та список визначних місць, що знаходяться в ньому;
- Country: Цей клас представляє собою країну, в якій розташовані міста. Він містить атрибути, такі як назва, опис, прапор, столиця та список міст, що входять до неї;
- Vacation: Цей клас представляє собою заплановану подорож користувача. Він містить атрибути, такі як назва подорожі, опис, дата початку та закінчення, список визначних місць, що планується відвідати, список міст, що планується відвідати, список місць проживання, список транспортних засобів, бюджет подорожі та інші атрибути, пов'язані з плануванням подорожі;
- VacationDay: Цей клас представляє собою день подорожі. Він містить атрибути, такі як дата, список визначних місць, що планується відвідати в цей день, список місць проживання, список транспортних засобів, бюджет дня подорожі та інші атрибути, пов'язані з плануванням дня подорожі;
- Document: Цей клас представляє собою документ, пов'язаний з подорожжю, наприклад, паспорт, віза або квиток. Він містить атрибути, такі як тип документа, номер документа, дата видачі та дата закінчення терміну дії;
- User: Цей клас представляє собою користувача системи. Він містить атрибути, такі як ім'я, прізвище, email, пароль, список улюблених подорожей, список минулих подорожей та інші атрибути, пов'язані з користувачем;
- Quiz: Цей клас представляє собою тест, який може пройти користувач, щоб отримати персональні рекомендації щодо подорожей. Він містить атрибути, такі як

назва тесту, опис, список запитань та відповідей, та інші атрибути, пов'язані з тестом;

- QuizResult: Цей клас представляє собою результат проходження тесту користувачем. Він містить атрибути, такі як дата проходження тесту, список відповідей користувача, персоналізовані рекомендації щодо подорожей та інші атрибути, пов'язані з результатом тесту;
- Gallery: Цей клас представляє собою галерею фотографій, пов'язаних з визначними місцями або подорожами. Він містить атрибути, такі як назва галереї, опис, список фотографій та інші атрибути, пов'язані з галереєю;
- ImageURLs: Цей клас представляє собою список URL-адрес зображень, пов'язаних з визначними місцями або подорожами;

3.1.2 Діаграма розгортання

Діаграма розгортання (рисунок 3.2) зображує фізичне розміщення різних компонентів програмної системи та їх взаємозв'язки у середовищі розгортання. Для розглянутої системи діаграма розгортання містить такі елементи:

– Сервер бази даних (DB SERVER), компонент: MongoDB

MongoDB є документо-орієнтованою базою даних, яка використовується як сховище даних для цієї системи.

– Сервер додатків (Node), компонент: Server

Це серверна частина додатку, яка виконує обробку запитів та взаємодіє з базою даних.

– Сервер системи керування контентом (CMS SERVER), компонент: CMS Contentful

Цей компонент відповідає за керування контентом веб-сайту та надає інтерфейс для редагування вмісту.

– Веб-браузер (Web Browser), компонент: index.js

Це клієнтська частина веб-додатку, яка виконується у веб-браузері користувача.

– Мобільний додаток (Mobile App), компонент: main.dart

Це мобільний додаток, розроблений з використанням мови програмування Dart, який дозволяє користувачам взаємодіяти з системою через мобільні пристрої.

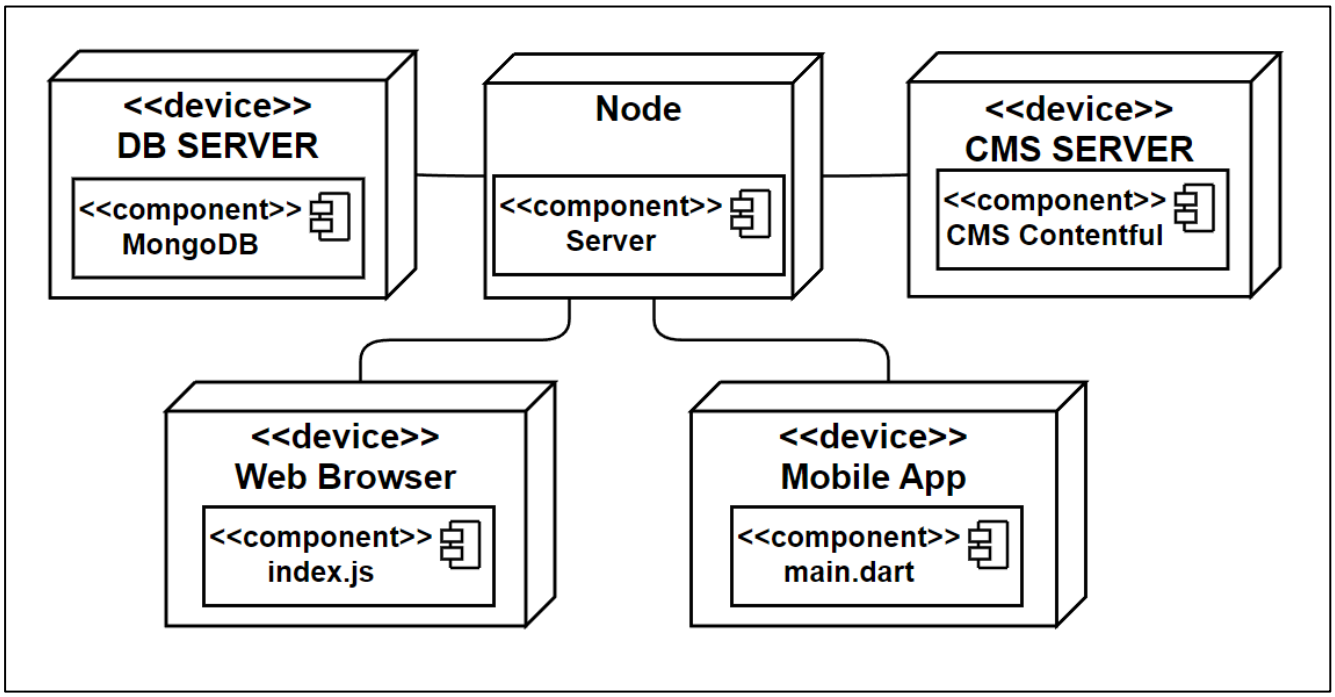


Рисунок 3.2 – Діаграма розгортання (Виконана самостійно)

Взаємозв'язки між компонентами можна описати таким чином:

– Веб-браузер та мобільний додаток взаємодіють з сервером додатків, надсилаючи запити та отримуючи відповіді.

– Сервер додатків взаємодіє з сервером бази даних MongoDB для зберігання та отримання даних.

– Сервер системи керування контентом CMS Contentful забезпечує інтерфейс для редагування та керування контентом веб-сайту, який потім може бути доступний через веб-браузер або мобільний додаток.

Розглянута діаграма розгортання ілюструє високорівневе представлення фізичної архітектури системи та допомагає зрозуміти, як різні компоненти взаємодіють між собою та розміщені у середовищі розгортання.

3.1.3 Діаграма послідовностей

Діаграма послідовностей, зображена на рисунку 3.3, ілюструє послідовність взаємодії між основними компонентами системи під час виконання запиту на отримання даних від користувача. Ключовими учасниками цього процесу є клієнтський додаток (представлений користувачем у веб- або мобільному середовищі), бекенд-сервер та база даних, які задіяні у наступній послідовності кроків:

– Ініціалізація запиту: Клієнтський додаток генерує запит на отримання певного набору даних та надсилає його до бекенд-серверу через відповідний інтерфейс взаємодії (напр., API).

– Перевірка аутентифікації та авторизації: Бекенд-сервер отримує вхідний запит і здійснює перевірку достовірності користувача та його прав доступу до запитуваних даних. Цей крок може включати валідацію облікових даних користувача, токенів автентифікації або інших механізмів контролю доступу.

– Формування та виконання запиту до бази даних: Після успішної аутентифікації та авторизації користувача бекенд-сервер формує відповідний запит до бази даних, використовуючи визначену мову запитів (напр., SQL для реляційних баз даних або запити NoSQL для нереляційних сховищ даних).

– Обробка запиту та повернення результатів: База даних обробляє отриманий запит, виконує необхідні операції пошуку, фільтрації та вибірки даних, а також формує результуючий набір даних відповідно до специфікації запиту. Сформований набір даних повертається бекенд-серверу.

– Обробка та форматування результатів: Бекенд-сервер отримує результати запиту від бази даних та, у разі необхідності, здійснює додаткову обробку або форматування даних відповідно до вимог клієнтського додатку.

– Повернення результатів користувачу: Після завершення необхідної обробки бекенд-сервер повертає відформатований набір даних клієнтському додатку через відповідний інтерфейс взаємодії.

– Представлення результатів: Клієнтський додаток отримує відповідь від бекенд-серверу та відображає або оброблює отримані дані відповідно до своєї логіки роботи та інтерфейсу користувача.

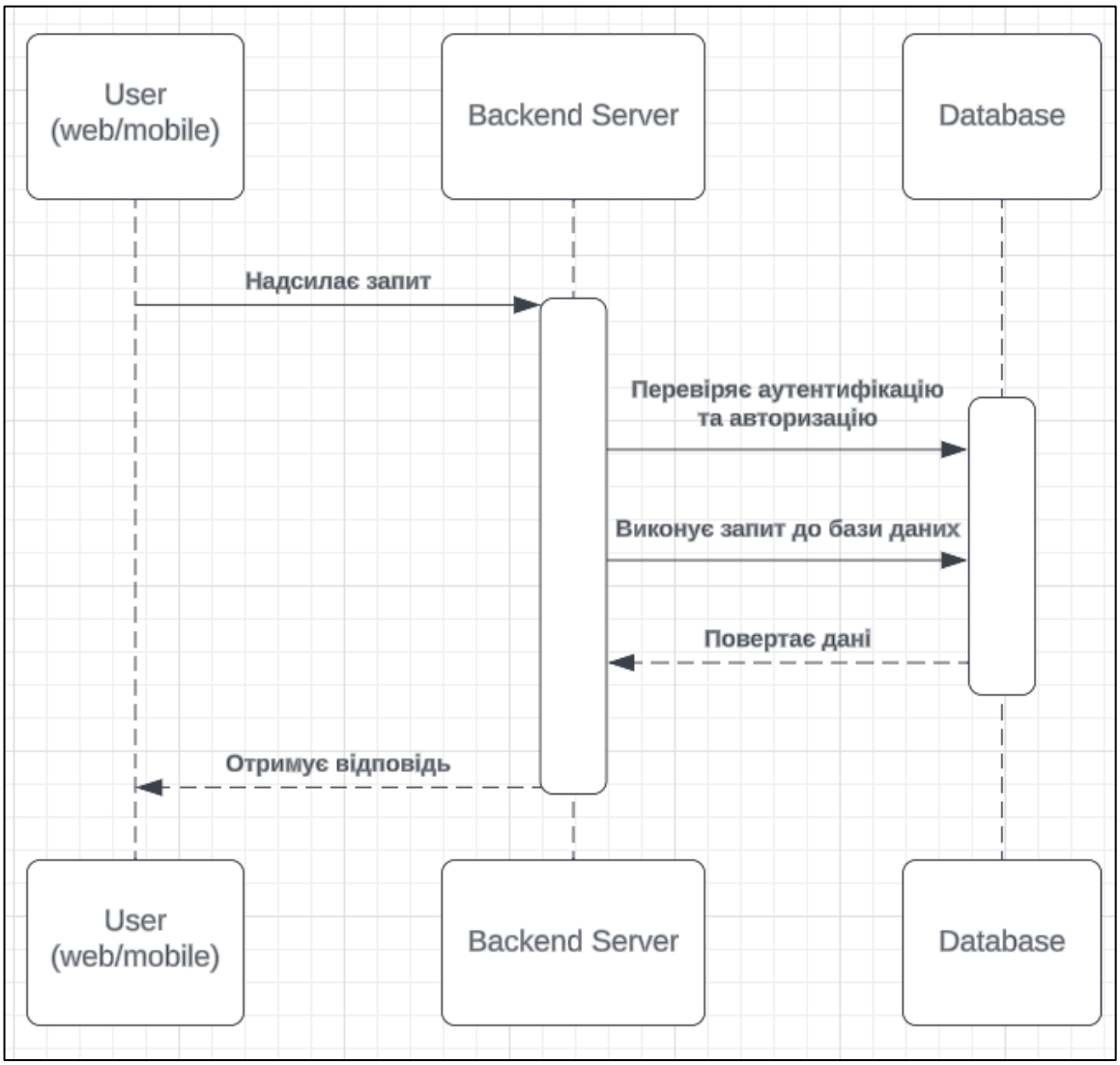


Рисунок 3.3 – Діаграма послідовностей (Виконана самостійно)

Ця послідовність взаємодії забезпечує безпечний та контрольований доступ до даних системи, а також гарантує цілісність та конфіденційність інформації завдяки використанню механізмів аутентифікації та авторизації. Крім того, розподіл обов'язків між клієнтським додатком, бекенд-сервером та базою даних сприяє модульності та масштабованості системи.

3.2 Проєктування архітектури ПЗ

3.2.1 Архітектурний патерн

Бекенд програмної системи для подорожей визначними місцями буде розроблений за допомогою архітектурного патерну Model-View-Controller (MVC). Цей патерн є широко застосовуваним та ефективним підходом для побудови веб-додатків, оскільки він забезпечує чітке розмежування між компонентами, що відповідають за управління даними, представлення та обробку запитів користувача.

– Модельний компонент (Model): Модель є центральним елементом, який відповідає за управління даними та реалізацію бізнес-логіки системи. У контексті нашого проекту, модель буде взаємодіяти з базою даних MongoDB, виконуючи операції зберігання, оновлення, видалення та валідації даних. Модельний компонент забезпечить абстракцію над даними та реалізуватиме правила бізнес-логіки, що стосуються обробки та маніпуляції даними;

– Представницький компонент (View): Представницький компонент відповідає за візуалізацію даних та відображення інформації для користувача. У нашій системі представницький компонент буде реалізований у вигляді адміністративної панелі, яка буде створена за допомогою HTML та CSS. Цей компонент буде відповідати за генерування користувацького інтерфейсу на основі даних, отриманих від моделі, та забезпечення належного відображення інформації для взаємодії з користувачем;

– Контролерний компонент (Controller): Контролерний компонент виступає в ролі проміжного шару між представницьким компонентом та модельним компонентом. Він відповідає за обробку запитів від користувача, взаємодію з модельним компонентом для отримання або маніпуляції даними, та взаємодію з представницьким компонентом для відображення відповідної інформації на адміністративній панелі. Контролерний компонент буде обробляти запити на створення, оновлення та видалення даних, а також забезпечувати відображення необхідної інформації у потрібному вигляді.

Використання архітектурного патерну MVC забезпечить модульність та гнучкість системи, полегшуючи розробку, тестування та подальшу підтримку програмного забезпечення. Така архітектура дозволить ефективно розділити відповідальності між компонентами, що сприятиме масштабованості системи та спростить реагування на зміни вимог і розширення функціональності програмного продукту в майбутньому.

3.2.2 Шаблони проєктування

У розробці бекенд-системи для подорожей визначними місцями буде застосовано низку шаблонів проєктування, які забезпечать модульність, гнучкість та розширюваність коду, а також полегшать подальшу підтримку та розвиток системи. Використання загальноприйнятих шаблонів проєктування є ефективним підходом для вирішення часто повторюваних проблем у розробці програмного забезпечення та забезпечує дотримання принципів об'єктно-орієнтованого програмування.

– Шаблон Singleton: Шаблон Singleton буде застосований для класів, які відповідають за встановлення та управління з'єднанням з базою даних. Цей шаблон гарантує існування лише одного екземпляра певного класу у всьому додатку, надаючи глобальну точку доступу до нього. Використання Singleton дозволить уникнути неефективного використання ресурсів та потенційних конфліктів, які можуть виникнути при створенні декількох екземплярів класу для доступу до бази даних.

– Шаблон Factory Method: Шаблон Factory Method буде використаний для створення об'єктів, які відповідають за обробку різних типів запитів. Цей шаблон надає абстрактний інтерфейс для створення об'єктів, делегуючи їх фактичне створення підкласам. Застосування Factory Method забезпечить гнучкість та розширюваність системи, оскільки додавання нових типів запитів не вимагатиме модифікації існуючого коду, а лише реалізації нових підкласів.

– Шаблон Data Access Object (DAO): Шаблон DAO буде використаний для забезпечення абстракції між бізнес-логікою та механізмом доступу до джерела

даних, яким є база даних. Цей шаблон інкапсулює логіку доступу до даних у окремих об'єктах, що дозволяє змінювати джерело даних (наприклад, перейти від однієї бази даних до іншої або використовувати зовнішній сервіс) без необхідності модифікації основної бізнес-логіки.

– Шаблон Dependency Injection (DI): Шаблон DI буде застосований для впровадження залежностей між компонентами системи. Цей шаблон сприяє низькій зв'язності між об'єктами та полегшує їх тестування, оскільки залежності можуть бути замінені на макети (mock) або фіктивні реалізації під час виконання тестів. Використання DI також полегшує модифікацію та підтримку коду, оскільки зміни в одному компоненті не вимагають змін в інших компонентах, що залежать від нього.

Впровадження цих шаблонів проєктування забезпечить дотримання принципів об'єктно-орієнтованого програмування, таких як інкапсуляція, абстракція та розділення відповідальностей. Це сприятиме створенню модульної, гнучкої та масштабованої системи, яка легко підтримуватиметься та розвиватиметься в майбутньому. Крім того, використання загальноприйнятих шаблонів полегшить розуміння коду іншими розробниками та спростить інтеграцію їх роботи.

3.3 Проєктування структури зберігання, управління даними

3.3.1 Аналіз вимог до бази даних

Система планування подорожей буде зберігати широкий спектр даних, пов'язаних з подорожами, користувачами та налаштуваннями системи. До основних категорій даних належать:

– Місця: Інформація про місця, які користувачі можуть відвідати, включаючи назву, опис, розташування, фотографії, відгуки, ціни та сезонність.

– Типи місць: Категорії місць, такі як готелі, ресторани, музеї, пам'ятки тощо.

– Сезони місць: Інформація про сезонність місць, включаючи дати пікового сезону, ціни та доступність.

– Групи місць: Колекції місць, які користувачі можуть створити та персоналізувати.

– Міста: Інформація про міста, де розташовані місця, включаючи назву, країну, регіон, фотографії та опис.

– Країни: Інформація про країни, де розташовані міста, включаючи назву, прапор, столицю, валюту та мову.

– Відпустки: План подорожей користувача, включаючи інформацію про дати подорожі, місця призначення, транспорт, проживання, бюджет, нотатки та документи.

– Дні відпустки: Детальний план кожного дня відпустки, включаючи інформацію про місця, які потрібно відвідати, транспорт, бюджет та нотатки.

– Документи: Документи, пов'язані з відпусткою, такі як паспорти, візи, квитки, бронювання та страхові поліси.

– Користувачі: Інформація про користувачів системи, включаючи ім'я, адресу електронної пошти, пароль та налаштування.

База даних повинна підтримувати широкий спектр функціональних можливостей, щоб відповідати потребам системи планування подорожей. До цих можливостей належать:

– Зберігання та пошук даних: База даних повинна ефективно зберігати та отримувати доступ до всіх типів даних, перелічених у розділі 4.3.1.2.

– Фільтрація та сортування даних: Користувачі повинні мати можливість фільтрувати та сортувати дані за різними критеріями, щоб легко знаходити потрібну інформацію.

– Зв'язки між даними: База даних повинна підтримувати зв'язки між різними типами даних, щоб представляти реальні стосунки між ними.

– Оновлення та видалення даних: Користувачі повинні мати можливість оновлювати та видаляти дані, як це необхідно.

– Безпека даних: База даних повинна забезпечувати захист даних від несанкціонованого доступу, зміни або видалення.

– Масштабованість: База даних повинна бути масштабованою, щоб вміщувати зростаючий обсяг даних та кількість користувачів.

– Ефективність: База даних повинна бути ефективною, щоб забезпечувати швидкий доступ до даних та мінімізувати час очікування.

Аналіз вимог до бази даних показав, що система планування подорожей потребуватиме потужної та гнучкої бази даних, яка може зберігати та управляти великою кількістю даних, підтримувати складні зв'язки між даними та забезпечувати широкий спектр функціональних можливостей.

3.3.2 Вибір системи управління базами даних

У розробці бекенд-системи для подорожей визначними місцями будуть використані дві різні системи зберігання даних: MongoDB та Contentful. Вибір кожної з них обґрунтований специфічними вимогами до даних та функціональності, які повинна забезпечувати система.

MongoDB є популярною неструктурованою базою даних, яка використовує модель даних документо-орієнтованого зберігання. Вибір MongoDB для цього проекту обумовлений наступними перевагами:

– Гнучка схема даних: MongoDB дозволяє зберігати дані в форматі JSON-подібних документів, які не вимагають жорсткої схеми. Це забезпечує гнучкість у моделюванні даних про визначні місця, які можуть мати різні атрибути та структури.

– Масштабованість: MongoDB забезпечує горизонтальну масштабованість, що дозволяє розподіляти дані на декількох серверах для підвищення продуктивності та доступності системи за зростаючого навантаження.

– Підтримка геопросторових даних: MongoDB має вбудовану підтримку геопросторових даних, що дозволяє ефективно зберігати та обробляти інформацію про розташування визначних місць на мапі.

– Індексація та агрегація: MongoDB забезпечує потужні можливості індексації та агрегації даних, що дозволяє ефективно виконувати складні запити та аналітичні операції над даними.

Contentful є хмарною Content Management System (CMS), яка забезпечує зручний та гнучкий спосіб управління контентом. Використання Contentful у цьому проекті обґрунтоване наступними факторами:

– Зосередження на контенті: Contentful дозволяє відокремити управління контентом від коду додатку, забезпечуючи зручний інтерфейс для редагування та публікації контенту.

– Гнучка модель контенту: Contentful пропонує гнучку модель для структурування контенту, що дозволяє легко створювати різноманітні типи контенту та налаштовувати їх поля.

– Масштабованість та доступність: Як хмарний сервіс, Contentful забезпечує масштабованість та високу доступність для зберігання та доставки контенту.

– Багатомовна підтримка: Contentful має вбудовану підтримку багатомовного контенту, що важливо для системи, орієнтованої на міжнародних користувачів.

Поєднання MongoDB та Contentful у цьому проекті забезпечить гнучкість та масштабованість системи. MongoDB буде використовуватися для зберігання структурованих даних про визначні місця, їх атрибутів та геопросторової інформації, в той час як Contentful буде відповідати за управління неструктурованим контентом, таким як описи, зображення та мультимедійні матеріали.

3.3.3 Проєктування структури бази даних

На основі створеної концептуальної ER-діаграми (рисунок 3.4) було спроектовано логічну модель реляційної бази даних для програмної системи. Ця модель складається з низки взаємопов'язаних сутностей, що забезпечують зберігання та організацію даних, необхідних для забезпечення функціональності системи.

Географічна локалізація місць забезпечується через зв'язки з сутностями City та Country, які відповідно представляють міста та країни. Ці сутності містять

ідентифікаційні та назвові атрибути, а також зв'язки з іншими сутностями, такими як Place для міст та City для країн.

Для забезпечення функціональності квізів та тестування знань користувачів передбачено сутності Question, Quiz та QuizResult. Question зберігає текст запитання та пов'язані з ним відповіді, тоді як Quiz об'єднує набори запитань у тематичні квізи, які мають назву та зображення. QuizResult фіксує результати проходження квізів користувачами, зберігаючи зв'язки з відповідними групами місць, містами та країнами.

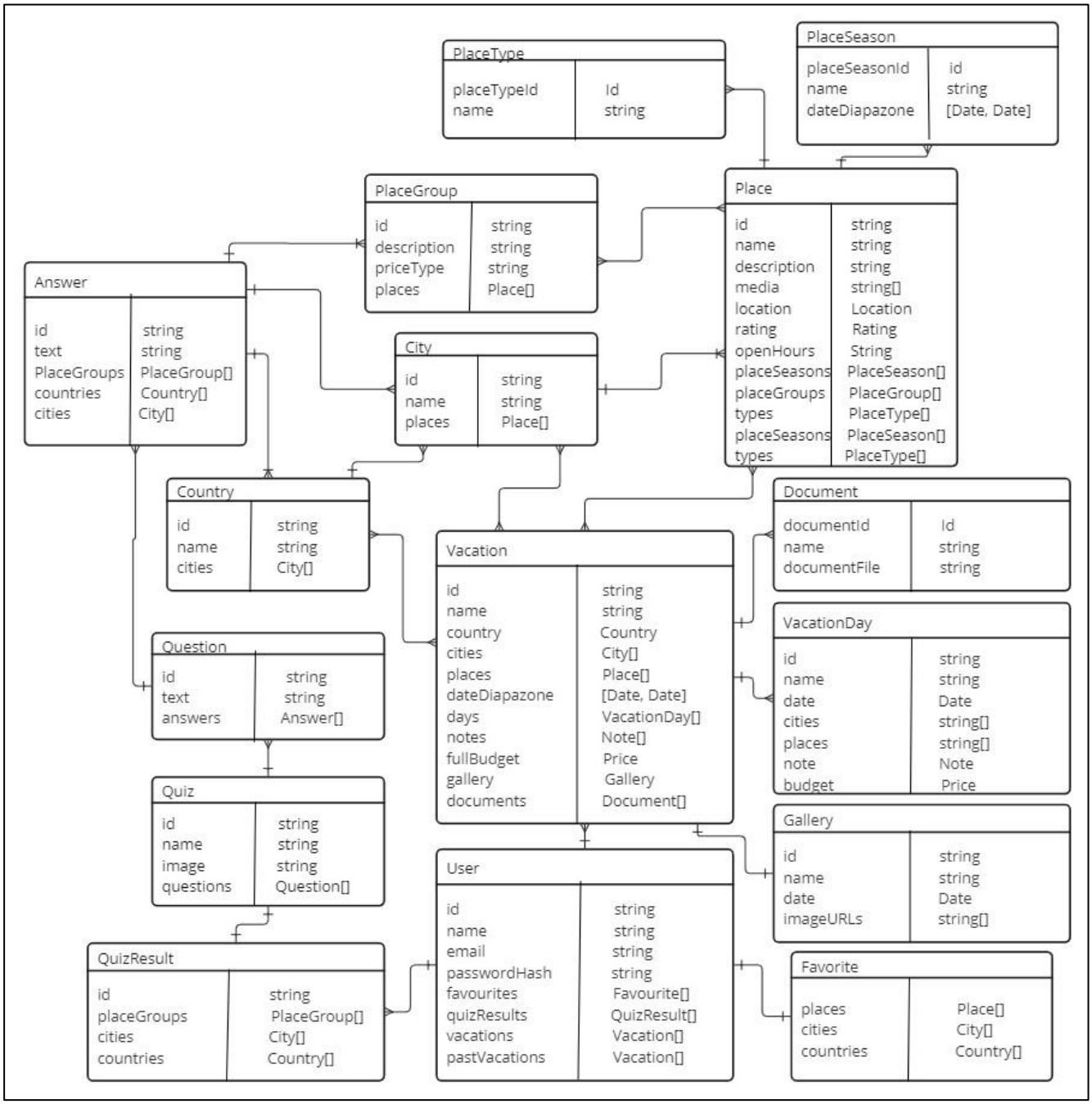


Рисунок 3.4 – Діаграма класів (Виконана самостійно)

Планування та організація подорожей реалізовано через сутності Vacation та VacationDay. Vacation представляє загальну інформацію про запланований тур, включно з назвою, країною, містами, місцями, датами, нотатками, бюджетом та галереєю. VacationDay забезпечує деталізацію на рівні окремих днів подорожі, містячи інформацію про дату, відвідувані міста та місця, нотатки та витрати.

Для зберігання персональної інформації користувачів, їх вподобань, результатів квізів та історії подорожей передбачено сутність User. Вона містить атрибути електронної пошти, хешованого пароля, списку улюблених місць, пройдених квізів, запланованих та минулих подорожей.

Додатково, сутність Document забезпечує можливість зберігання та керування документами, пов'язаними з подорожами, такими як квитки або інструкції.

Галереї зображень, пов'язаних з певними місцями або подорожами, реалізовано через сутність Gallery, яка містить назву, дату та посилання на зображення.

Нарешті, сутність Favorite дозволяє користувачам зберігати свої вподобання щодо місць, міст та країн для подальшого використання та персоналізації.

Така структура бази даних забезпечує цілісне та скоординоване зберігання всіх необхідних даних для ефективного функціонування програмної системи, включаючи планування подорожей, проходження квізів, перегляд інформації про місця, міста та країни, а також керування користувацькими даними та мультимедійним вмістом.

3.4 Приклади найцікавіших алгоритмів та методів

У процесі розробки бекенд-компонентів для майбутньої програмної системи будуть застосовані різноманітні алгоритмічні рішення та методології, що забезпечать ефективну та надійну роботу системи. Розглянемо детальніше деякі ключові прийоми, які планується реалізувати:

– Алгоритм рекомендаційної системи: Для надання користувачам персоналізованих рекомендацій щодо потенційно цікавих визначних пам'яток, маршрутів чи туристичних дестинацій буде розроблений та інтегрований спеціалізований рекомендаційний алгоритм. Цей алгоритм здійснюватиме аналіз історії взаємодії користувача із системою, його персональних уподобань та преференцій, а також використовуватиме методи машинного навчання та обробки великих даних для формування релевантних пропозицій у контексті індивідуальних потреб кожного окремого користувача;

– Методологія кешування даних: З метою підвищення швидкодії та оптимізації використання обчислювальних ресурсів у бекенд-компонентах буде впроваджено ефективну стратегію кешування. Ця методологія передбачатиме тимчасове зберігання результатів попередньо оброблених запитів або обчислень у високошвидкісній проміжній пам'яті (кеші), що дозволить уникнути повторних ресурсномістких обчислень для ідентичних запитів у майбутньому. Це забезпечить скорочення часу відповіді сервера та покращення загальної продуктивності системи;

– Алгоритми криптографічного шифрування: Для гарантування належного рівня захисту конфіденційних даних користувачів, таких як облікові записи, паролі та особиста інформація, у бекенд-компонентах будуть імплементовані новітні алгоритми криптографічного шифрування. Ці алгоритми забезпечуватимуть надійне кодування інформації під час її передачі через мережу та зберігання на серверних ресурсах, ефективно захищаючи її від несанкціонованого доступу або компрометації.

Синергія застосування вищезазначених та інших передових алгоритмічних рішень і методологій дозволить забезпечити високу якість, ефективність та безпеку функціонування бекенд-компонентів майбутньої програмної системи, гарантуючи задоволення потреб користувачів у плануванні подорожей на найвищому рівні.

3.5 Створення дизайну системи.

Інтерфейс адміністратора є ключовим компонентом будь-якої комплексної програмної системи, надаючи користувачам з відповідними правами доступ до функціональних можливостей для управління системою та її даними. Цей розділ описує принципи та рекомендації щодо створення дизайну для інтерфейсу адміністратора.

3.5.1 Цільова аудиторія

Інтерфейс адміністратора орієнтований на користувачів з різними рівнями досвіду та ролями:

– Системні адміністратори: Ці фахівці відповідають за загальне адміністрування та налаштування системи, забезпечуючи її безперебійну роботу. Інтерфейс для системних адміністраторів має надавати чіткий контроль над ключовими параметрами, системними ресурсами та доступом до них;

– Менеджери користувачів: Ці користувачі керують доступом до системи, створюють та редагують облікові записи, присвоюють ролі та права, а також контролюють активність користувачів. Інтерфейс для менеджерів користувачів повинен бути зручним для управління користувачами та моніторингу їх активності;

– Менеджери контенту: Ці фахівці відповідають за створення та управління інформаційним контентом, зокрема додавання, редагування та видалення контенту, завантаження та обробку мультимедійних файлів, а також модерацію та публікацію відгуків користувачів. Інтерфейс для менеджерів контенту повинен забезпечувати легкість виконання цих завдань;

– Менеджери відгуків: Ці фахівці займаються збором та аналізом відгуків користувачів з метою вдосконалення системи. Інтерфейс для менеджерів відгуків повинен дозволяти легко переглядати, модерувати та аналізувати відгуки, виявляючи проблемні моменти та покращуючи загальний користувацький досвід.

3.5.2 Функціональні можливості

Інтерфейс адміністратора має бути не просто набором інструментів, а й багатофункціональним центром керування, що охоплює всі аспекти системи:

Управління користувачами:

– Створення, редагування та видалення користувачів, немов реєстрація нових жителів віртуального міста.

– Надання та налаштування ролей та прав доступу, немов розподіл повноважень та відповідальності.

– Перегляд інформації про користувачів та їх активність, немов аналіз статистичних даних про жителів.

Управління контентом:

– Додавання, редагування та видалення місць, міст, країн та інших даних, немов створення та оновлення інформаційної бази знань.

– Завантаження та обробка зображень, відео та інших мультимедійних файлів, немов додавання візуальних барв до інформації.

– Модерування та публікація відгуків користувачів, немов контроль за громадською думкою.

Управління опитуваннями:

– Створення, редагування та видалення опитувань, немов проведення соціологічних досліджень серед користувачів.

– Додавання та редагування запитань та варіантів відповідей, немов формування анкет для збору інформації.

– Перегляд результатів опитувань та аналітика, немов вивчення думок та побажань користувачів.

Контроль та моніторинг:

– Налаштування параметрів системи та оновлення програмного забезпечення: Інтерфейс адміністратора має надавати доступ до налаштування ключових параметрів системи, таких як мова, часовий пояс, формат даних, рівень

доступу та інші. Це дозволить адміністраторам налаштувати систему відповідно до потреб та вподобань.

– Створення та налаштування звітів: Інтерфейс має надавати можливість створення та налаштування звітів, що містять інформацію про використання системи, активність користувачів, помилки та інші дані. Це допоможе адміністраторам відстежувати роботу системи, виявляти проблемні моменти та приймати обґрунтовані рішення щодо покращення.

3.5.3 Дизайн інтерфейсу

Інтерфейс адміністратора має бути не лише функціональним, але й приємним у використанні, візуально привабливим та інтуїтивно зрозумілим. Для цього важливо дотримуватися наступних принципів дизайну:

– Простота: Інтерфейс має бути простим та зрозумілим, щоб адміністратори могли легко знайти потрібні функції та виконувати необхідні дії.

– Функціональність: Всі необхідні функції мають бути доступні та зручно розташовані, щоб адміністратори могли швидко та ефективно виконувати свою роботу.

– Гнучкість: Інтерфейс має бути налаштовуваним, щоб адміністратори могли пристосувати його до своїх потреб та вподобань.

– Відгук: Користувачі мають отримувати чіткий візуальний та вербальний відгук на свої дії, щоб вони знали, що відбувається в системі.

– Доступність: Інтерфейс має бути доступним для людей з обмеженими можливостями, щоб всі могли комфортно та безперешкодно використовувати систему.

4 ОПИС ПРИЙНЯТИХ ПРОГРАМНИХ РІШЕНЬ

У цьому розділі представлено детальний опис прийнятих програмних рішень для реалізації системи управління подорожами визначними місцями. Основна мета системи – надати користувачам зручний інструмент для планування та організації своїх подорожей, включаючи вибір місць для відвідування, зберігання улюблених локацій, а також отримання персоналізованих рекомендацій на основі результатів опитувань.

Система побудована на основі сучасних технологій, які забезпечують високу продуктивність, масштабованість та безпеку. Зокрема, для створення API використовується Node.js у поєднанні з Express.js, що дозволяє розробити гнучкий і надійний серверний додаток. Для зберігання даних обрано MongoDB – документно-орієнтовану базу даних, яка підтримує роботу з великими обсягами неструктурованої інформації.

Керування вмістом здійснюється за допомогою Contentful – headless CMS, яка надає зручний інтерфейс для адміністраторів і редакторів вмісту, а також дозволяє легко інтегрувати дані з різних джерел у наш додаток. Використання цієї платформи забезпечує ефективне управління контентом та швидке його оновлення.

4.1 Реєстрація та авторизація користувачів

Реєстрація користувачів є одним із ключових аспектів будь-якої програмної системи, яка працює з користувачами. У нашій системі реалізація цього процесу здійснюється за допомогою Node.js, з використанням фреймворку Express.js для створення API та MongoDB для зберігання даних. Для забезпечення безпеки використовується bcrypt для хешування паролів і jwt для генерації токенів.

Для моделювання користувача було використано Mongoose – об'єктно-документний маппер (ODM) для MongoDB. Це дозволяє нам визначити схему користувача, яка включає наступні ключові поля:

- userId: Унікальний ідентифікатор користувача, який генерується за допомогою бібліотеки uuid. Це поле є обов’язковим та унікальним;
- name: Ім'я користувача. Це поле також є обов’язковим;

- email: Адреса електронної пошти користувача, яка повинна бути унікальною. Використовується для ідентифікації користувача під час входу в систему;
- passwordHash: Хеш паролю користувача, зберігається для забезпечення безпеки. Паролі не зберігаються у відкритому вигляді;
- avatarUrl: URL аватару користувача, яке є опціональним полем;
- favourites: Структура для зберігання улюблених місць, міст та країн користувача;
- quizResults: Містить результати опитувань користувача, включаючи ідентифікатори опитувань, груп місць, міст та країн;
- vacations та pastVacations: Поля для зберігання інформації про поточні та минулі відпустки користувача. Використовуються зв'язки з моделлю Vacation;
- isAdmin: Поле, яке позначає, чи є користувач адміністратором;
- createdAt: Дата створення облікового запису;
- lastLoginAt: Дата останнього входу в систему.

Модель користувача також включає методи та middleware для роботи з паролями і оновленням дати останнього входу.

Процес реєстрації включає декілька важливих кроків:

– Перевірка існування користувача: Спочатку перевіряється, чи існує вже користувач з вказаним email. Якщо такий користувач існує, повертається відповідь з кодом 400 та повідомленням про помилку;

– Генерація унікального ідентифікатора: Якщо користувач не знайдений, генерується унікальний ідентифікатор за допомогою бібліотеки uuid;

– Хешування паролю: Пароль хешується за допомогою bcrypt для забезпечення безпеки;

– Збереження користувача: Новий користувач зберігається у базі даних MongoDB;

– Генерація токена: Після успішного збереження користувача генерується JWT токен для автентифікації користувача.

Процес входу включає наступні кроки:

– Пошук користувача: Пошук користувача за email у базі даних. Якщо користувач не знайдений, повертається відповідь з кодом 401 та повідомленням про помилку;

– Перевірка паролю: Перевірка правильності паролю за допомогою методу comparePassword. Якщо пароль неправильний, повертається відповідь з кодом 401;

– Генерація токена: Після успішної перевірки паролю генерується JWT токен для автентифікації користувача.

Middleware для автентифікації відповідає за перевірку валідності JWT токена у запитах. Воно виконує наступні дії:

– Витягування токена: Токен витягується з заголовку Authorization.

– Перевірка токена: Валідація токена за допомогою jwt.verify. Якщо токен недійсний, повертається відповідь з кодом 401.

– Додавання ідентифікатора користувача: Якщо токен валідний, ідентифікатор користувача додається до об'єкту запиту, що дозволяє ідентифікувати користувача у подальших запитах.

4.2 Робота з Contentful

Contentful є сучасною headless CMS (Content Management System), яка дозволяє зберігати та керувати контентом, відокремлюючи його від інтерфейсу користувача. Це забезпечує гнучкість і масштабованість при розробці веб-додатків. У розроблюваній системі Contentful використовується для зберігання інформації про міста, країни та місця, які можуть бути цікавими для подорожуючих. Інтеграція з Contentful здійснюється за допомогою GraphQL-запитів, що дозволяє ефективно отримувати потрібні дані.

Контролери для роботи з Contentful відповідають за обробку запитів від користувачів, отримання даних з Contentful та повернення цих даних у форматі, зручному для подальшого використання. Розглянемо основні контролери: cityController, countryController та placeController.

Контролер cityController (наведено у Додатку А) включає два методи: swowAllCities та showOneCity.

– swowAllCities: Цей метод відповідає за отримання списку всіх міст. Для цього використовується GraphQL-запит, який витягує ідентифікатори, назви та координати всіх міст з колекції міст у Contentful. Отримані дані повертаються у форматі JSON.

– showOneCity: Цей метод відповідає за отримання даних про конкретне місто за його ідентифікатором. GraphQL-запит витягує назву та координати міста. Якщо під час виконання запиту виникає помилка, метод повертає повідомлення про помилку з відповідним статусом.

Контролер countryController також включає два методи: swowAllCountries та showOneCountry.

– swowAllCountries: Метод отримує список всіх країн, включаючи їх ідентифікатори, назви та координати. Використовується GraphQL-запит, аналогічний до запиту для міст.

– showOneCountry: Цей метод отримує дані про конкретну країну за її ідентифікатором. Запит витягує назву та координати країни. У разі помилки метод повертає відповідне повідомлення про помилку.

Контролер placeController включає методи swowAllPlaces та showOnePlace.

– swowAllPlaces: Метод отримує список всіх місць з детальною інформацією про кожне місце, включаючи ідентифікатори, назви, координати, описи, колекції медіа, рейтинги, години роботи, сезони, групи місць та типи. Використовується складний GraphQL-запит, який дозволяє отримати всі необхідні дані за один запит.

– showOnePlace: Метод отримує дані про конкретне місце за його ідентифікатором. Запит аналогічний до запиту для всіх місць, але обмежений одним конкретним місцем. У разі помилки метод повертає повідомлення про помилку.

Для інтеграції з Contentful використовується бібліотека graphql-request, яка спрощує роботу з GraphQL-запитами. Кожен запит формулюється за допомогою gql-тегу та передається клієнту GraphQL, який виконує запит і повертає результат.

У разі успішного виконання запиту результат повертається у форматі JSON. У разі помилки відбувається обробка винятків, і користувачу повертається повідомлення про помилку з відповідним статусом.

4.3 Вакації

Вакації – це важлива частина програмної системи, яка дозволяє користувачам планувати, організовувати та зберігати інформацію про свої подорожі. Вони включають детальну інформацію про маршрут, дні подорожі, відвідувані міста та місця, бюджет, галерею фото та документи. Для реалізації функціоналу вакацій використовується декілька моделей: Vacation, VacationDay та User. Інтеграція з Contentful здійснюється за допомогою GraphQL-запитів для отримання додаткової інформації про країни, міста та місця.

Модель Vacation представляє собою загальний план подорожі. Основні поля включають:

- name: Назва подорожі;
- country: Країна, яку планується відвідати;
- cities: Список міст, які планується відвідати;
- places: Список місць, які планується відвідати;
- dateDiapazone: Діапазон дат подорожі, включаючи початкову та кінцеву дати;
- days: Список днів подорожі, представлений об'єктами VacationDay;
- fullBudget: Загальний бюджет подорожі;
- gallery: Галерея фото, пов'язана з подорожжю;
- documents: Список документів, пов'язаних з подорожжю;
- user: Ідентифікатор користувача, якому належить подорож.

Модель VacationDay представляє собою окремий день подорожі. Основні поля включають:

- name: Назва дня;
- date: Дата дня подорожі;
- cities: Список міст, відвідуваних протягом дня;

- places: Список місць, відвідуваних протягом дня;
- notes: Заметки, пов'язані з днем подорожі;
- budget: Бюджет, виділений на цей день;
- vacation: Ідентифікатор подорожі, до якої належить день.

Для роботи з моделями використовуються відповідні контролери, що забезпечують функціонал додавання, видалення, оновлення та отримання даних про ваканції та дні подорожей. Розглянемо основні методи контролерів.

Контролер для вакацій включає наступні методи:

– addVacation: Метод для додавання нової подорожі. Він отримує дані про подорож та дні подорожі з запиту, створює нові об'єкти Vacation та VacationDay, зберігає їх у базі даних, а також оновлює дані користувача, додаючи нову подорож до його списку подорожей;

– deleteVacation: Метод для видалення подорожі за її ідентифікатором. Він видаляє подорож з бази даних та оновлює дані користувача, видаляючи подорож з його списку подорожей;

– updateVacation: Метод для оновлення даних про подорож за її ідентифікатором. Він оновлює дані подорожі у базі даних та повертає оновлену інформацію;

– getVacations: Метод для отримання списку всіх подорожей користувача. Він отримує подорожі з бази даних та деталізує їх, використовуючи GraphQL-запити до Contentful для отримання додаткової інформації про країни, міста та місця;

– getVacationById: Метод для отримання даних про конкретну подорож за її ідентифікатором. Він деталізує інформацію про подорож, використовуючи GraphQL-запити до Contentful.

4.4 Збереження файлів на прикладі галереї та сховища документів

У програмній системі для подорожей візначними місцями однією з ключових функцій є можливість збереження та керування різноманітними файлами, такими як зображення, документи та інші мультимедійні дані. Для реалізації цієї функції

було прийнято рішення використовувати хмарне сховище Firebase Storage від Google.

Firebase Storage є масштабованим і надійним хмарним сховищем, яке забезпечує безпечне зберігання та швидкий доступ до файлів. Інтеграція Firebase Storage у програмне рішення дозволяє уникнути проблем, пов'язаних з обмеженим місцем на серверах, та забезпечує простий спосіб керування файлами.

Для взаємодії з Firebase Storage було використано бібліотеку firebase-admin. Конфігурація Firebase Storage виконується в окремому модулі firebaseConfig.js, де ініціалізується з'єднання з Firebase та експортується об'єкт bucket, що представляє сховище.

Для завантаження файлів використовується проміжний обробник upload з бібліотеки multer. Він дозволяє зручно обробляти вхідні файли та передавати їх до контролерів.

Для керування галереєю зображень було створено окрему модель Gallery, яка містить масив URL-адрес завантажених зображень (imageURLs) та посилання на відповідну подорож (vacation).

Контролер galleryController містить кілька методів для роботи з галереєю:

а) addImageToGallery: Цей метод дозволяє завантажити одне або кілька зображень до галереї. Спочатку перевіряється наявність існуючої галереї для відповідної подорожі. Якщо галереї не існує, створюється нова. Потім для кожного файлу виконується наступна послідовність дій:

1) Створюється новий файл в Firebase Storage з унікальною назвою.
2) Файл завантажується в сховище Firebase Storage за допомогою потоку запису.
3) Після завершення завантаження встановлюється публічний доступ до файлу.
4) URL-адреса файлу додається до масиву imageURLs у моделі галереї.
5) Зберігається зв'язок між моделями Vacation та Gallery.

б) getGalleryById: Цей метод повертає галерею за її ідентифікатором.

в) getUserGalleries: Метод повертає всі галереї, пов'язані з користувачем, шляхом пошуку подорожей користувача та відповідних їм галерей.

г) deleteGallery: Видаляє галерею та всі пов'язані з нею зображення з Firebase Storage. Спочатку видаляються всі зображення з Firebase Storage, а потім сама галерея з бази даних. Також видаляється зв'язок між подорожжю та галереєю.

д) updateGallery: Дозволяє видалити одне або кілька зображень з існуючої галереї. Спочатку перевіряється наявність галереї. Потім масив imageURLs в моделі галереї фільтрується, видаляючи URL-адреси, які необхідно видалити. Після оновлення галереї видаляються відповідні файли з Firebase Storage.

е) addImagesToGalleryById: Аналогічно до addImageToGallery, але замість створення нової галереї, зображення додаються до існуючої галереї за її ідентифікатором.

Для керування документами, пов'язаними з подорожами, була створена модель Document, яка містить назву документа (name), URL-адресу документа (documentURL) та посилання на відповідну подорож (vacation).

Контролер documentController містить два методи для роботи з документами:

а) addDocument: Цей метод дозволяє завантажити новий документ та пов'язати його з існуючою подорожжю. Спочатку перевіряється наявність файлу в запиті. Потім виконуються наступні кроки:

1) Створюється новий файл в Firebase Storage з унікальною назвою.
2) Файл завантажується в сховище Firebase Storage за допомогою потоку запису.
3) Після завершення завантаження встановлюється публічний доступ до файлу.
4) Створюється нова модель Document з назвою, URL-адресою файлу та посиланням на подорож.
5) Зберігається зв'язок між моделями Document та Vacation.

б) getDocumentById: Цей метод повертає документ за його ідентифікатором.

Це рішення дозволяє зберігати файли в хмарному сховищі Firebase Storage, що забезпечує масштабованість та надійність, а також зручно керувати галереями зображень та сховищем документів, пов'язаних з подорожами користувачів.

4.5 Адміністративна панель

У розробленій програмній системі для подорожей визначними місцями важливим аспектом є наявність адміністративної панелі, яка забезпечує зручний та ефективний спосіб керування веб-сайтом, контентом та налаштуваннями. Адміністративна панель складається з кількох веб-сторінок, кожна з яких має свої унікальні функції та призначення. Доступ до адміністративної панелі обмежений лише для авторизованих адміністраторів, що гарантує захист системи від несанкціонованих змін та забезпечує належний рівень безпеки.

а) Сторінка входу адміністратора

Першим кроком для доступу до адміністративної панелі є процес авторизації адміністратора на спеціальній сторінці входу (рисунок 4.1). На цій сторінці адміністратор повинен ввести свої облікові дані, а саме електронну пошту та пароль, у відповідні поля форми.

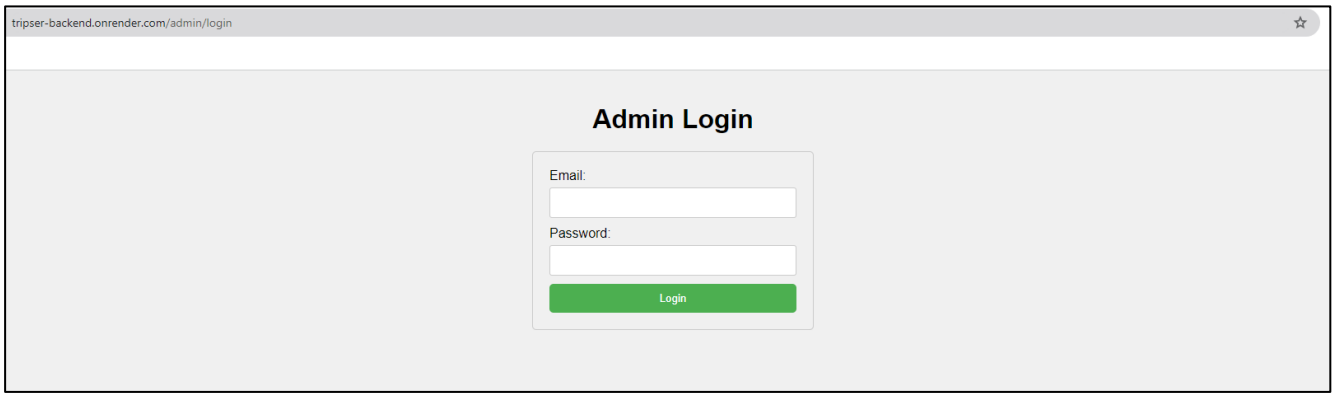


Рисунок 4.1 – Сторінка входу адміністратора

Після успішної перевірки облікових даних адміністратор отримує JWT-токен (JSON Web Token), який зберігається у cookies браузера. Цей токен використовується для перевірки автентифікації адміністратора при доступі до різних розділів адміністративної панелі та виконанні певних дій.

1) Сторінка входу адміністратора реалізована з використанням HTML, CSS та JavaScript. HTML-розмітка визначає структуру сторінки та форму входу з полями для введення електронної пошти та пароля. CSS-стилі забезпечують привабливий зовнішній вигляд та адаптивність для різних розмірів екрану, гарантуючи належний користувацький досвід незалежно від пристрою, який використовується.

2) Функція login, реалізована на JavaScript, відповідає за виконання асинхронного запиту до серверної частини за допомогою fetch API. Цей запит передає введені користувачем облікові дані на сервер для перевірки їх дійсності та, у разі успішної автентифікації, отримує JWT-токен у відповіді.

б) Адміністративна панель

Після успішного входу адміністратор переходить на головну сторінку адміністративної панелі (рисунок 4.2).

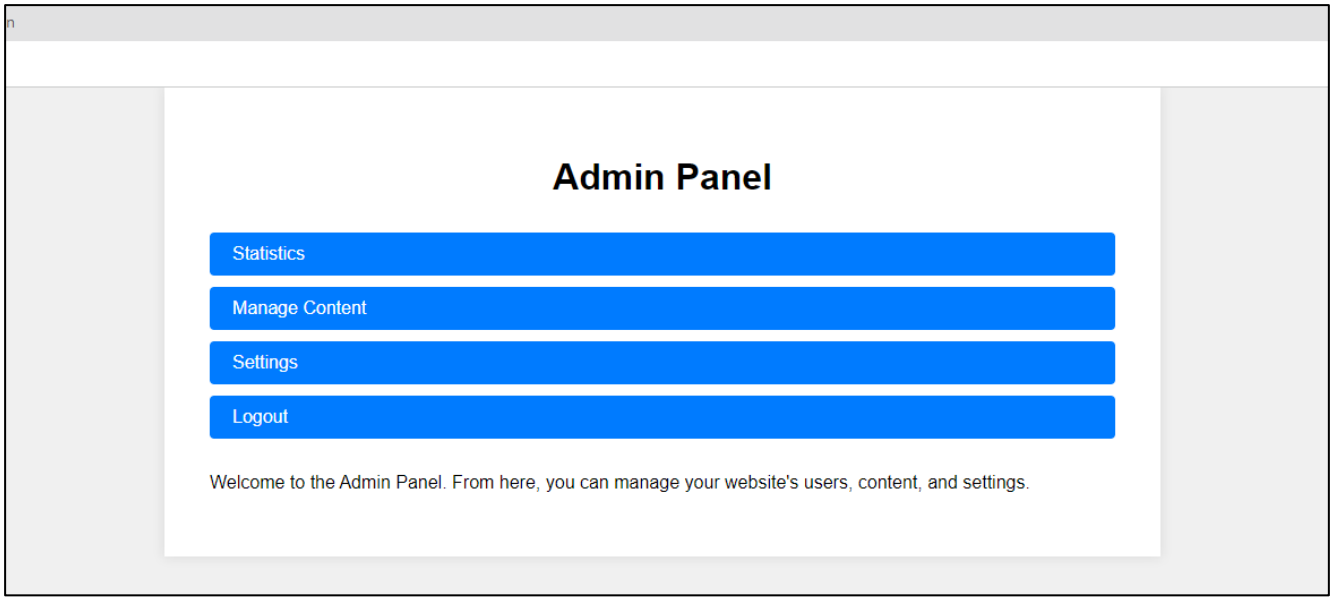


Рисунок 4.2 – Адміністративна панель

На цій сторінці адміністратор має доступ до різноманітних функцій керування, таких як:

1) Перегляд статистики користувачів та реєстрацій. Ця функція дозволяє адміністратору відстежувати активність користувачів у системі та

аналізувати тенденції реєстрацій, що сприяє прийняттю обґрунтованих рішень щодо розвитку та вдосконалення веб-ресурсу.

2) Керування вмістом через інтеграцію з Contentful, який є потужною headless CMS-системою. Завдяки інтеграції з Contentful адміністратор може легко створювати, редагувати та видаляти різні типи контенту, такі як опис місць, квізи, події тощо, забезпечуючи актуальність та релевантність інформації на веб-сайті.

3) Налаштування системи. Ця функція передбачає можливість налаштування різноманітних параметрів та опцій програмної системи згідно з потребами та вимогами адміністратора.

4) Вихід з облікового запису адміністратора. Після завершення роботи з адміністративною панеллю адміністратор може безпечно вийти зі свого облікового запису, очистивши JWT-токен з cookies браузера.

Головна сторінка адміністративної панелі реалізована за допомогою HTML та CSS. HTML-розмітка визначає структуру сторінки та меню з посиланнями на різні функції. CSS-стилі забезпечують привабливий зовнішній вигляд та відповідне розташування елементів, створюючи зручний та інтуїтивно зрозумілий інтерфейс для адміністратора. JavaScript-функції, такі як logout(), statistics() та settings(), виконують відповідні дії при натисканні на посилання, забезпечуючи необхідну функціональність для ефективного керування системою.

в) Сторінка статистики

На сторінці статистики (рисунок 4.3) адміністратор може переглядати дані про реєстрацію користувачів та активних користувачів системи. Ця інформація є важливою для аналізу та моніторингу активності користувачів, а також для прийняття рішень щодо подальшого розвитку та оптимізації веб-ресурсу.

1) Сторінка статистики реалізована за допомогою HTML, CSS та JavaScript, а також використовує бібліотеку Chart.js для відображення даних у вигляді наочних графіків та діаграм.

2) HTML-розмітка визначає структуру сторінки та місце для відображення графіків та статистичних даних.

3) CSS-стилі забезпечують привабливий зовнішній вигляд та належне розташування елементів на сторінці, створюючи зручний та зрозумілий інтерфейс для адміністратора.

4) JavaScript-функції getRegistrationStatsAndDrawChart() та getActiveUsersStats() виконують асинхронні запити до серверної частини для отримання статистичних даних про реєстрацію користувачів та активних користувачів відповідно. Отримані дані обробляються та відображаються на сторінці за допомогою графіків, створених з використанням бібліотеки Chart.js.

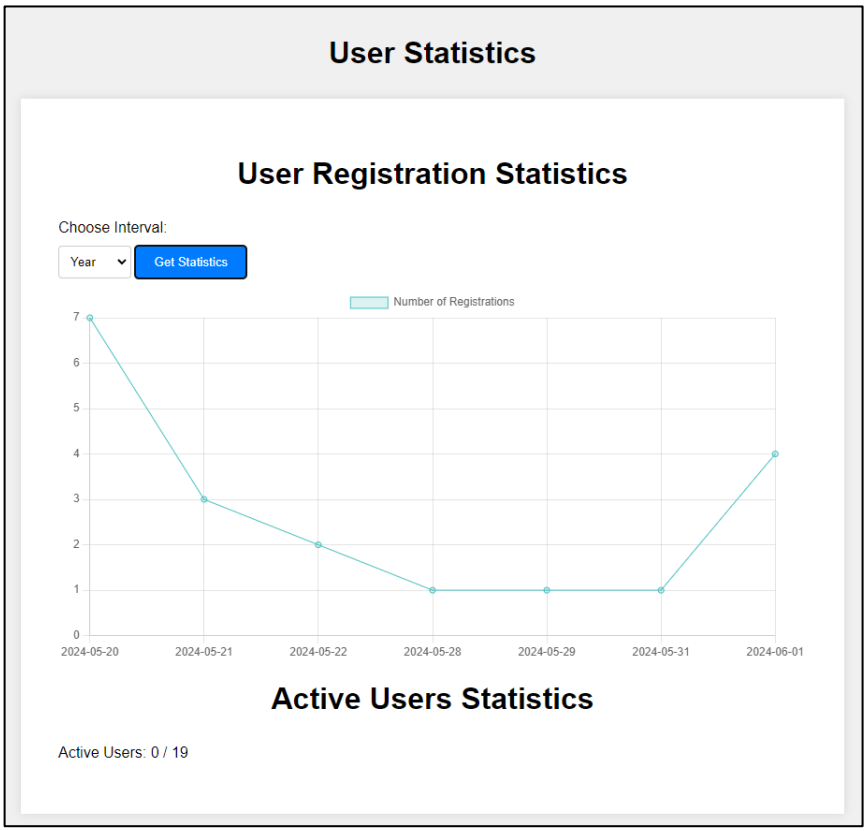


Рисунок 4.3 – Сторінка статистики

г) Сторінка налаштувань

На сторінці налаштувань (рисунок 4.4) адміністратор має можливість додавати нових адміністраторів, видаляти існуючих адміністраторів та виконувати резервне копіювання даних системи. Ці функції є критично важливими для забезпечення належного управління та контролю над веб-ресурсом.

1) Сторінка налаштувань реалізована за допомогою HTML, CSS та JavaScript.

2) HTML-розмітка визначає структуру сторінки, форми для додавання та видалення адміністраторів, а також кнопку для резервного копіювання даних.

3) CSS-стилі забезпечують привабливий зовнішній вигляд та належне розташування елементів на сторінці, створюючи зручний та інтуїтивно зрозумілий інтерфейс для адміністратора.

4) JavaScript-функції addAdmin, deleteAdmin та backupData обробляють відповідні події форм та виконують асинхронні запити до серверної частини для додавання, видалення адміністраторів та резервного копіювання даних відповідно.

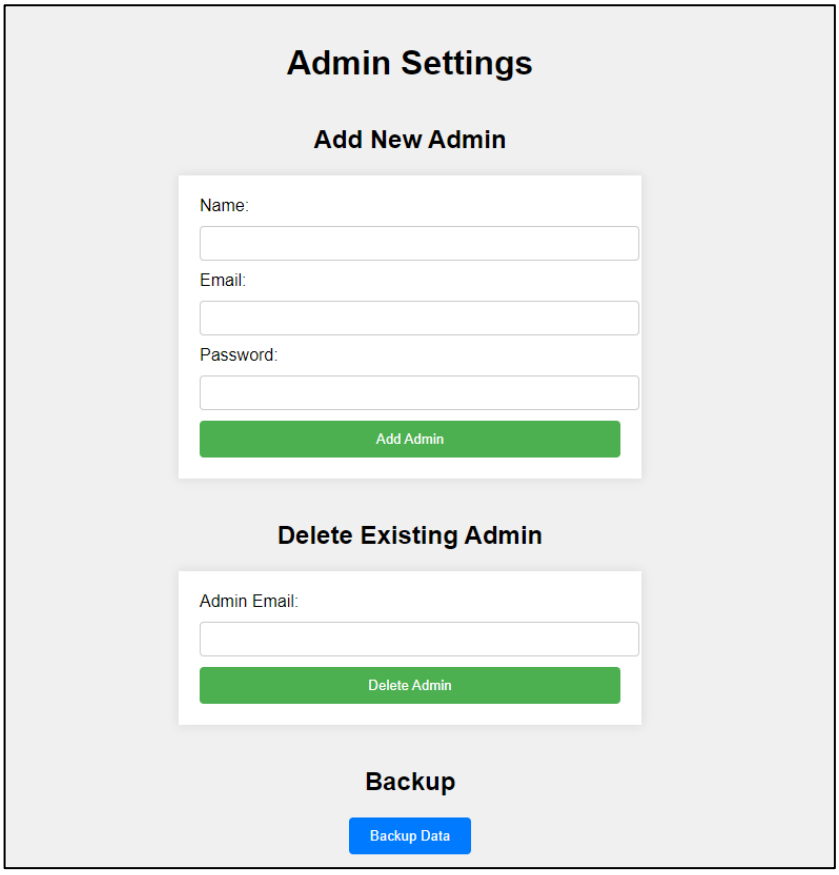


Рисунок 4.4 – Сторінка налаштувань

Ці функції забезпечують необхідну функціональність для адміністративного керування користувачами та даними системи, дозволяючи

адміністраторам ефективно контролювати доступ до веб-ресурсу та зберігати важливі дані в безпечному місці. Використання асинхронних запитів fetch гарантує плавну роботу інтерфейсу та уникнення блокування основного потоку виконання JavaScript, забезпечуючи належний користувацький досвід.

Загалом, адміністративна панель є потужним інструментом для керування веб-сайтом та контентом програмної системи для подорожей визначними місцями. Вона забезпечує належний рівень безпеки, зручний інтерфейс та необхідну функціональність для ефективного адміністрування, дозволяючи адміністраторам контролювати різні аспекти веб-ресурсу та приймати обґрунтовані рішення на основі статистичних даних. Інтеграція з Contentful як headless CMS-системою додає гнучкості та масштабованості в керуванні контентом, забезпечуючи актуальність та релевантність інформації на веб-сайті.

5 ТЕСТУВАННЯ РОЗРОБЛЕНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Тестування програмного забезпечення є невід'ємною частиною процесу розробки і відіграє ключову роль у забезпеченні якості кінцевого продукту. Ефективне тестування дозволяє виявити і виправити дефекти та невідповідності на ранніх стадіях розробки, що в свою чергу знижує вартість виправлення помилок і підвищує загальну якість програмного забезпечення.

Для забезпечення високої якості та надійності розробленого бекенду веб-застосунку було проведено ретельне тестування. Мануальне тестування зосереджувалося на різноманітних аспектах функціонування бекенду, щоб гарантувати його стабільність та відповідність вимогам. Зокрема, перевірка функціональності включала тестування основних операцій, таких як обробка запитів, взаємодія з базою даних, авторизація та автентифікація користувачів.

5.1 Тестування обробки запитів

Тестування процесу обробки запитів було одним із ключових аспектів перевірки бекенду. Це передбачало ретельну перевірку різноманітних HTTP-методів (GET, POST, PUT, DELETE) та їх коректної обробки на рівні контролерів та сервісів. Були протестовані різні типи запитів, включаючи запити з параметрами в URL, тілі запиту та заголовках.

Під час тестування обробки запитів особлива увага приділялася перевірці валідації вхідних даних, обробці помилок та винятків, а також правильності відповідей від бекенду. Були протестовані різноманітні сценарії, включаючи успішні випадки, граничні ситуації та некоректні вхідні дані.

У таблиці 5.1 наведено приклад тест-кейсу для перевірки обробки запиту на створення нового користувача. Під час тестування авторизації та автентифікації особлива увага приділялася перевірці безпеки та надійності реалізованих механізмів. Були протестовані різноманітні сценарії, включаючи успішну реєстрацію та вхід в систему, спроби входу з некоректними даними, перевірка прав доступу до захищених ресурсів та обробка конфіденційних даних.

Таблиця 5.1 – Тест-кейс №1 (таблиця виконана самостійно)

Інформація про тест-кейс			
Ідентифікатор тесту:	Тест-кейс №1		
Опис функції:	Реєстрація нового користувача		
Власник тесту:	Піняєв Євгеній Володимирович		
Дата створення:	01.06.2024		
Мета тесту:	Перевірка процесу реєстрації нового користувача		
Передумова			
№	Опис випадку	Очікуваний результат	Висновок
1	Бекенд запущено та готовий до обробки запитів	Бекенд готовий до прийому запитів	Пройдено
Реєстрація нового користувача			
№	Опис випадку	Очікуваний результат	Висновок
1	Відправити POST-запит з коректними даними користувача (ім'я, електронна пошта, пароль)	Бекенд успішно обробляє запит, створює новий обліковий запис користувача та повертає HTTP-статус 201 (Created) з інформацією про створеного користувача	Пройдено
2	Відправити POST-запит з некоректними даними користувача (наприклад, невалідна електронна пошта)	Бекенд успішно валідує вхідні дані, визначає їх некоректність та повертає HTTP-статус 400 (Bad Request) з описом помилки	Пройдено
3	Відправити POST-запит з відсутніми обов'язковими полями (наприклад, без електронної пошти або паролю)	Бекенд успішно валідує вхідні дані, визначає відсутність обов'язкових полів та повертає HTTP-статус 400 (Bad Request) з описом помилки	Пройдено
Результати тестування			
Тестувальник: Піняєв Є.В.	Дата прогону тесту: 01.06.2024	Результат тесту (P/F/B): ПРОЙДЕНО (P)	

5.2 Тестування взаємодії з базою даних

Важливим аспектом тестування бекенду було перевірка коректної взаємодії з базою даних. Це включало тестування операцій читання, створення, оновлення та видалення даних з використанням різноманітних запитів та умов фільтрації.

Під час тестування взаємодії з базою даних особлива увага приділялася перевірці коректності збереження та отримання даних, обробці помилок та винятків, а також забезпеченню цілісності та узгодженості даних. Були протестовані різноманітні сценарії, включаючи успішні випадки, граничні ситуації та некоректні вхідні дані. У таблиці 5.2 наведено приклад тест-кейсу для перевірки операції отримання списку відпусток користувача з бази даних.

Таблиця 5.2 – Тест-кейс №2 (таблиця виконана самостійно)

Інформація про тест-кейс			
Ідентифікатор тесту:	Тест-кейс №1		
Опис функції:	Отримання списку відпусток користувача		
Власник тесту:	Піняєв Євгеній Володимирович		
Дата створення:	01.06.2024		
Мета тесту:	Перевірка обробки запиту на отримання списку ресурсів з бази даних		
Передумова			
№	Опис випадку	Очікуваний результат	Висновок
1	Бекенд запущено та готовий до обробки запитів. База даних містить деякі ресурси	Бекенд готовий до прийому запитів. База даних містить дані	Пройдено
Отримання списку відпусток користувача			
№	Опис випадку	Очікуваний результат	Висновок
1	Відправити GET-запит без додаткових параметрів	Бекенд успішно обробляє запит та повертає список всіх ресурсів з бази даних. HTTP-статус 200 (OK)	Пройдено
2	Відправити GET-запит з параметром	Бекенд успішно обробляє запит та повертає ресурс	Пройдено
3	Відправити GET-запит з некоректним параметром	Бекенд успішно валідує вхідні дані, визначає некоректність параметра фільтрації та повертає HTTP-статус 400 (Bad Request) з описом помилки.	Пройдено
Результати тестування			
Тестувальник: Піняєв Є.В.	Дата прогону тесту: 01.06.2024	Результат тесту (P/F/B): ПРОЙДЕНО (P)	

5.3 Тестування інтеграції з CMS Contentful

Для забезпечення коректної роботи інтеграції бекенду з системою управління контентом Contentful було проведено ретельне тестування. Під час тестування особлива увага приділялася перевірці правильності взаємодії з API Contentful, забезпеченню належної обробки помилок та валідації отриманих даних.

У таблиці 5.3 наведено приклад тест-кейсу для перевірки операції отримання списку міст з CMS Contentful.

Таблиця 5.3 – Тест-кейс №3 (таблиця виконана самостійно)

Інформація про тест-кейс			
Ідентифікатор тесту:	Тест-кейс №1		
Опис функції:	Отримання списку міст з CMS Contentful		
Власник тесту:	Піняєв Євгеній Володимирович		
Дата створення:	01.06.2024		
Мета тесту:	Перевірка обробки запиту на отримання списку міст з CMS Contentful		
Отримання списку міст з CMS Contentful			
№	Опис випадку	Очікуваний результат	Висновок
1	Бекенд запущено та готовий до обробки запитів. Налаштування з'єднання з Contentful завершено успішно	Бекенд готовий до прийому запитів. З'єднання з Contentful налаштовано	Пройдено
Реєстрація нового користувача			
№	Опис випадку	Очікуваний результат	Висновок
1	Відправити GET-запит на отримання списку міст з Contentful	Бекенд успішно обробляє запит, отримує список міст з Contentful та повертає HTTP-статус 200 (OK) з отриманими даними	Пройдено
2	Симулювати помилку з'єднання з Contentful	Бекенд успішно виявляє помилку з'єднання з Contentful та повертає HTTP-статус 500 (Internal Server Error) з описом помилки	Пройдено
Результати тестування			
Тестувальник: Піняєв Є.В.	Дата прогону тесту: 01.06.2024	Результат тесту (P/F/B): ПРОЙДЕНО (P)	

ВИСНОВКИ

У результаті розробки бекенд частини програмної системи, роботи над дипломним проектом, було досягнуто значних успіхів і реалізовано ключові функціональні вимоги. На початковому етапі проекту було проведено ретельний аналіз потреб користувачів та визначено детальні вимоги до функціональності, продуктивності, масштабованості, безпеки та інших критичних аспектів бекенд системи.

Для забезпечення відповідності вимогам було застосовано сучасні технології та архітектурні підходи, такі як архітектура RESTful API, шаблон проектування Model-View-Controller (MVC) та використання JSON Web Tokens (JWT) для забезпечення безпеки та автентифікації користувачів. Ця комбінація дозволила розробити надійну, масштабовану та безпечну бекенд частину системи.

Основні функціональні можливості бекенду були успішно реалізовані, включаючи обробку запитів користувачів, управління даними про користувачів, визначні місця, подорожі, опитування та інші сутності. Було забезпечено безпечне зберігання конфіденційних даних користувачів, валідацію вхідних даних та ефективну обробку помилок і виключних ситуацій.

Для досягнення високої продуктивності та швидкої реакції системи було впроваджено механізми кешування, оптимізації запитів до бази даних, індексації та агрегації даних. Крім того, було реалізовано алгоритми рекомендацій та персоналізації на основі аналізу вподобань користувачів та результатів опитувань, що забезпечило надання релевантних рекомендацій щодо визначних місць і подорожей.

Після завершення розробки функціональної частини, бекенд система пройшла ретельне тестування. Це дозволило перевірити відповідність бекенду функціональним вимогам, його стабільність та здатність витримувати високе навантаження.

Результатом роботи став стабільний, високопродуктивний та захищений бекенд, який забезпечує надійну підтримку функціонування програмної системи і задовольняє потреби користувачів у швидкій та безпечній роботі з даними.

Незважаючи на досягнуті успіхи, проект має потенціал для подальшого вдосконалення та розширення функціональності. Можливими напрямками розвитку є впровадження додаткових заходів безпеки, інтеграція з зовнішніми сервісами для збагачення даних, використання машинного навчання для поліпшення алгоритмів рекомендацій, розширення можливостей аналітики та моніторингу системи.

Знання та досвід, здобуті під час розробки бекенд частини програмної системи, є цінними для подальшого професійного зростання та реалізації майбутніх проектів у сфері розробки серверних додатків, бекенд систем та мікросервісних архітектур.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Tripadvisor : веб-сайт. URL: https://www.tripadvisor.com (дата звернення: 09.04.2024).

2. Google Trips : веб-сайт. URL: https://www.google.com/travel/ (дата звернення: 09.04.2024).

3. Airbnb: веб-сайт. URL: https://www.airbnb.com.ua/ (дата звернення: 09.04.2024).

4. Node.js : веб-сайт. URL: https://nodejs.org/en (дата звернення: 09.04.2024).

5. Express.js : веб-сайт. URL: https://expressjs.com/ (дата звернення: 09.04.2024).

6. MongoDB : веб-сайт. URL: https://www.mongodb.com/ (дата звернення: 09.04.2024).

7. RESTful API : веб-сайт. URL: https://www.mediawiki.org/wiki/Wikimedia_REST_API (дата звернення: 09.04.2024).

8. JSON Web Token (JWT) : веб-сайт. URL: https://jwt.io/ (дата звернення: 09.04.2024).

ДОДАТОК А

Код контролера cityController

```
const { gql } = require("graphql-request");

const cityController = {
  swowAllCities: async (req, res) => {
    try {
      const query = gql`
        {
          cityCollection {
            items {
              sys {
                id
              }
              name
              location {
                lat
                lon
              }
            }
          }
        }
      `;

      const data = await req.graphQLClient.request(query);
      res.json(data);
    } catch (error) {
      console.error(error);
      res
        .status(500)
        .json({ error: "Ошибка при получении данных из Contentful" });
    }
  },
  showOneCity: async (req, res) => {
    const cityId = req.params.id;
    try {
      const query = gql`
        {
          city(id: "${cityId}") {
            name
            location {
            lat
            lon
            }
          }
        }
      `;

      const data = await req.graphQLClient.request(query);
      res.json(data);
    } catch (error) {
      console.error(error);
      res
        .status(500)
        .json({ error: "Ошибка при получении данных из Contentful" });
    }
  },
};

module.exports = cityController;
```

ДОДАТОК Б

Участь у конференції

Піняєв Євгеній Володимирович
Харківський національний університет радіоелектроніки
студент кафедри Програмної інженерії
Онищенко Костянтин Георгійович
Харківський національний університет радіоелектроніки
старший викладач кафедри Програмної інженерії
м. Харків, Україна
yevhenii.piniaiev@nure.ua

КЛЮЧОВІ СЛОВА: ПОДОРОЖІ, NODE.JS, EXPRESS.JS, CMS, MVC
KEYWORDS: ПОДОРОЖІ, NODE.JS, EXPRESS.JS, CMS, MVC

РОЗРОБКА БЕКЕНД ЧАСТИНИ ПРОГРАМНОЇ СИСТЕМИ ДЛЯ ПЛАНУВАННЯ ПОДОРОЖЕЙ

У сучасному світі подорожі стають все більш популярними та доступними для широких верств населення. Однак процес планування та організації поїздки може бути складним та вимагати значних зусиль від мандрівника. Існуючі рішення часто мають обмежений функціонал або незручний інтерфейс, що ускладнює ефективне планування подорожей. Метою даного проєкту є розробка бекенд частини програмної системи для комплексного планування та організації подорожей, що забезпечить зручний та функціональний сервіс для користувачів.

Для створення бекенду буде використано Node.js [1] та фреймворк Express.js [2], що забезпечать високу продуктивність, масштабованість та швидкодію системи. Основні дані, такі як інформація про користувачів, їхні подорожі та бронювання, будуть зберігатися у нереляційній базі даних MongoDB [3]. Для керування постійними даними, наприклад, описами локацій, готелів та атракцій, буде використано headless CMS Contentful [4], що дозволить зручно управляти цим контентом.

Бекенд система матиме модульну структуру та складатиметься з кількох основних компонентів, таких як авторизація користувачів, керування подорожами, пошук подорожей тощо. Взаємодія з базою даних MongoDB відбуватиметься через спеціалізовані API. Інтеграція з Contentful забезпечить отримання актуальних постійних даних про локації, готелі та інші туристичні об'єкти. Структура програмного коду відповідатиме MVC [5] (рисунок 1).

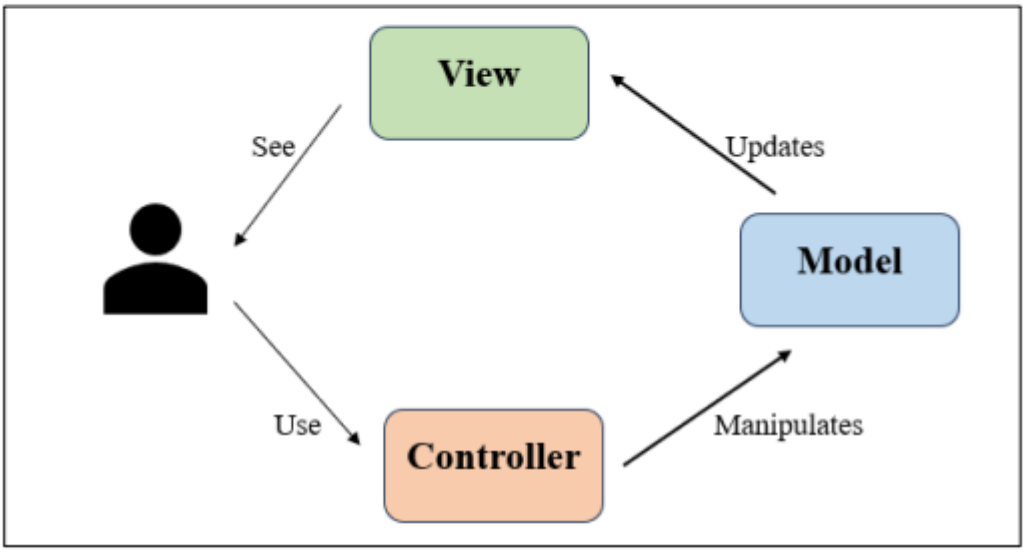


Рисунок 1 – Ілюстрація архітектурного патерну MVC

Шаблон Model-View-Controller (MVC) – це архітектурний патерн, який розділяє додаток на три взаємопов'язані компоненти:

– model (модель): відповідає за керування даними додатку, їх валідацію, доступ до бази даних тощо;

– view (представлення): відповідає за відображення інтерфейсу для користувача та надання даних з моделі;

– controller (контролер): обробляє вхідні запити, взаємодіє з моделлю для оновлення даних та вибирає відповідне представлення для відображення результату.

Така архітектура забезпечує модульність, підвищену підтримуваність і повторне використання коду.

Система матиме такі ключові функції:

– авторизація та керування обліковими записами користувачів з використанням JWT токенів;

– CRUD операції для планування подорожей: створення, редагування та видалення майбутніх поїздок;

– пошук подорожей за різними критеріями, такими як локація, дати, бюджет тощо;

– інтеграція з зовнішніми сервісами бронювання готелів, авіаквитків та інших туристичних послуг.

Забезпечення безпеки даних є пріоритетом при розробці системи. Буде використано шифрування конфіденційних даних, захищені протоколи передачі даних (HTTPS) та JWT токени для автентифікації. Для гарантії високої якості коду будуть розроблені юніт-тести для перевірки окремих компонентів, а також проведено інтеграційне тестування всієї бекенд системи.

Розроблена бекенд частина програмної системи для планування подорожей забезпечить комплексне та зручне рішення для самостійної організації мандрівок. Використання сучасних технологій, таких як Node.js, Express.js, MongoDB та Contentful, а також дотримання архітектурного патерну MVC, гарантуватиме високу продуктивність, масштабованість та легкість підтримки системи. Ключові функції, такі як авторизація користувачів, керування подорожами, пошук за критеріями та інтеграція з сервісами бронювання, забезпечать повний комплекс необхідних інструментів для планування подорожей.

Розроблене рішення відрізнятиметься повнотою функціоналу, зручністю використання та високими показниками безпеки. У майбутньому планується розширення системи, зокрема, додавання можливостей спільного планування подорожей для груп користувачів, інтеграції з сервісами навігації та рекомендаційними системами. Це дозволить створити ще більш комплексний та корисний сервіс для мандрівників усього світу.

СПИСОК ДЖЕРЕЛ

1. Node.js [Електронний ресурс]. URL: https://nodejs.org/en/ (Дата звернення 26.03.2024).

2. Express.js [Електронний ресурс]. URL: https://expressjs.com/ (Дата звернення 26.03.2024).

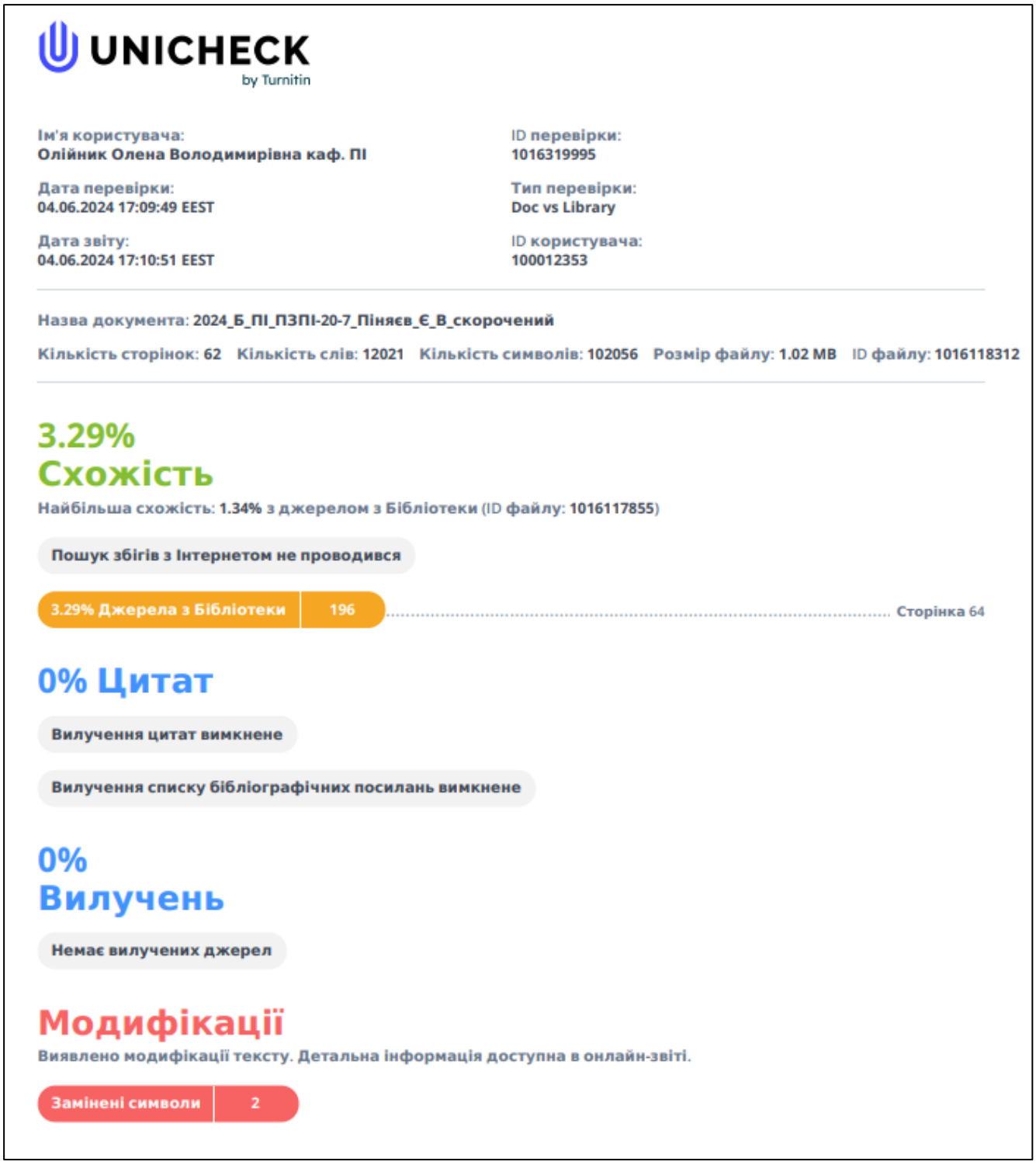
3. MongoDB [Електронний ресурс]. URL: https://www.mongodb.com/ (Дата звернення 26.03.2024).

4. Contentful [Електронний ресурс]. URL: https://www.contentful.com/ (Дата звернення 26.03.2024).

5. Model–view–controller [Електронний ресурс] // Wikipedia. URL: https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller (Дата звернення 26.03.2024).

ДОДАТОК В

Звіт результатів перевірки на унікальність тексту в базі ХНУРЕ



UNICHECK
by Turnitin

Ім'я користувача:
Олійник Олена Володимирівна каф. ПІ

ID перевірки:
1016319995

Дата перевірки:
04.06.2024 17:09:49 EEST

Тип перевірки:
Doc vs Library

Дата звіту:
04.06.2024 17:10:51 EEST

ID користувача:
100012353

Назва документа: 2024_Б_ПІ_ПЗПІ-20-7_Піняєв_Є_В_скорочений

Кількість сторінок: 62 Кількість слів: 12021 Кількість символів: 102056 Розмір файлу: 1.02 MB ID файлу: 1016118312

3.29%
Схожість

Найбільша схожість: 1.34% з джерелом з Бібліотеки (ID файлу: 1016117855)

Пошук збігів з Інтернетом не проводився

3.29% Джерела з Бібліотеки 196 ..... Сторінка 64

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи 2

ДОДАТОК Г

Слайди презентації

Аналоги та актуальність

Tripadvisor

airbnb

В чому потреба?

1. Комплексний підхід до планування
2. Персоналізованість
3. Бюджетний контроль
4. Рефлексія спогадів
5. Усі квитки під рукою
6. Цікаві тестування
7. Персоналізовані підказки

Путівка до ідеального подорожнього досвіду: Постановка Задачі

Використання передових технологій

Головним завданням є реалізація серверної частини з використанням передових технологій, таких як Node.js, Express.js та MongoDB

Ефективна система управління контентом

Використання платформи Contentful для зручного зберігання, структурування та управління всім контентом, пов'язаним з подорожами

Потужний та гнучкий API

Надання API, який забезпечує безперебійну взаємодію з клієнтською та мобільною частинами системи

Моделюваннння ПЗ

Методи та інструменти

Діаграма класів

Діаграма класів є важливим інструментом моделювання програмного забезпечення, що дозволяє візуалізувати структуру системи, зв'язки між класами та їхні взаємодії, що сприяє зрозумінню архітектури та логіки програми під час розробки.

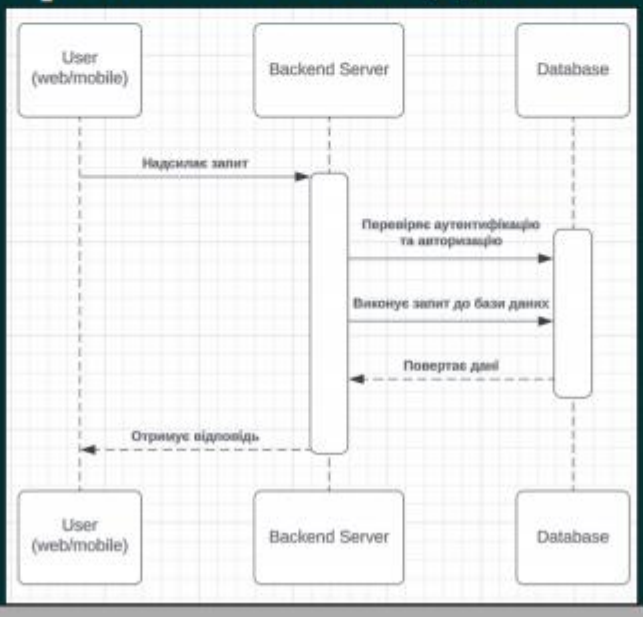
Діаграма розгортання

Діаграма розгортання дозволяє візуалізувати фізичну структуру системи, розташування її компонентів та зв'язки між ними в середовищі реалізації, що сприяє ефективній розробці, впровадженню та управлінню програмним продуктом.

Діаграма послідовностей

Діаграма послідовностей важлива для візуалізації та аналізу взаємодії об'єктів у системі на основі послідовності виконання їхніх операцій та повідомлень, що допомагає у розумінні порядку взаємодії

Діаграма послідовностей



Архітектура серверної частини програмної системи

Архітектурний патерн - MVC

Цей патерн є широко застосовуваним та ефективним підходом для побудови веб-додатків, оскільки він забезпечує чітке розмежування між компонентами, що відповідають за управління даними, представлення та обробку запитів користувача

Структури зберігання, управління даними

MongoDB і Contentful - це два потужних інструменти для зберігання та управління даними. MongoDB пропонує гнучку структуру зберігання, а Contentful - зручний інтерфейс для управління контентом. Разом вони створюють ефективну екосистему для розробки проектів.

Технології та інструменти розробки

Node.js

Node.js - це засіб розробки, який дозволяє використовувати JavaScript для побудови потужних та масштабованих веб-додатків на стороні сервера.

Express.js

Express.js - це мінімалістичний та гнучкий веб-фреймворк для Node.js, який дозволяє швидко розробляти потужні веб-додатки з мінімальними зусиллями.

MongoDB

MongoDB - це документ-орієнтована база даних, яка надає гнучку та масштабовану структуру зберігання даних для різноманітних застосувань.

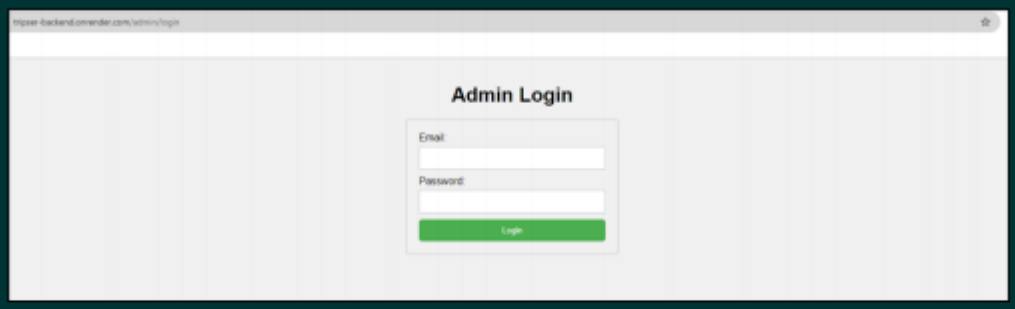
CMS Contentful

Contentful - це сучасна система управління контентом, що надає зручний інтерфейс для створення, редагування та публікації контенту на веб-сайтах і додатках.

Додаткові інструменти

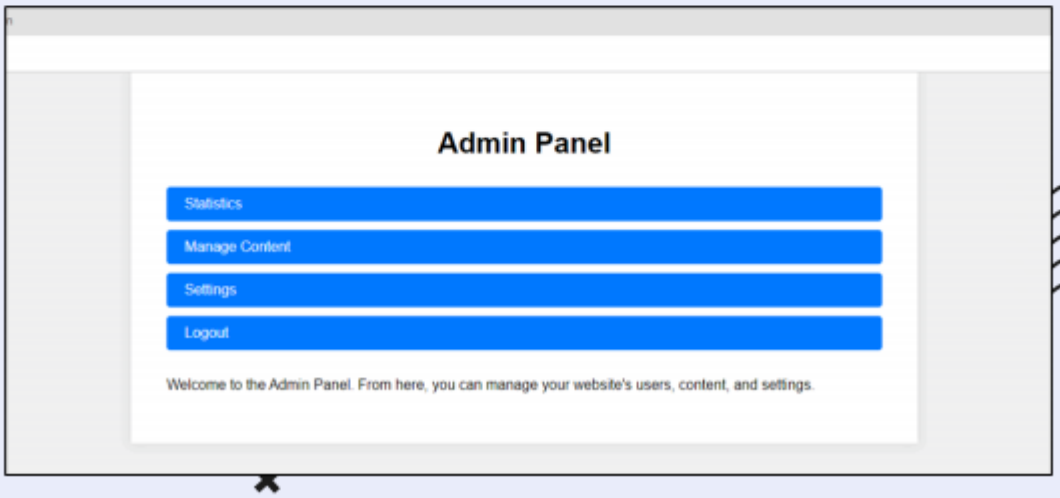
JSON Web Token, VS Code

Робота адміністративної панелі



09

Головна сторінка



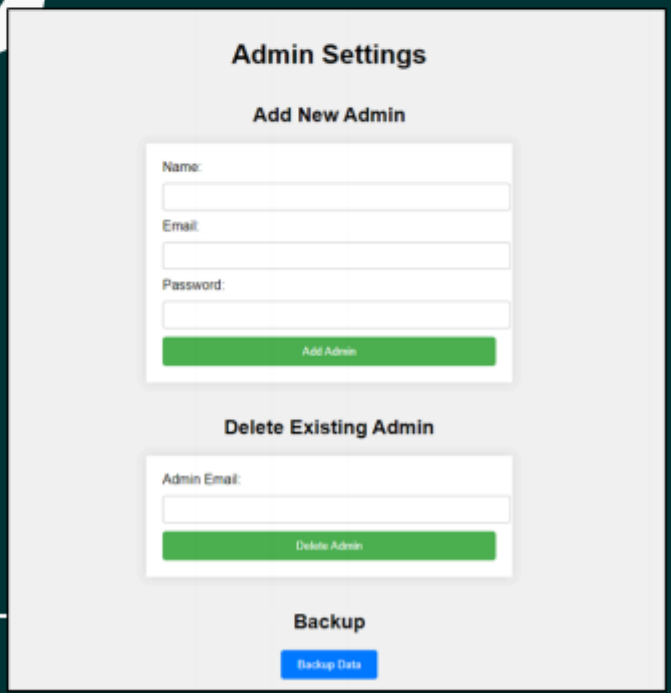
10

Сторінка статистики прогамної системи

11

Пінясв Є.В. ПЗПІ-20-7

Сторінка налаштувань



12

Тестування серверної частини

Тестування обробки запитів

Тестування процесу обробки запитів було одним із ключових аспектів перевірки бекенду. Це передбачало ретельну перевірку різноманітних HTTP-методів (GET, POST, PUT, DELETE) та їх коректної обробки на рівні контролерів та сервісів. Були протестовані різні типи запитів, включаючи запити з параметрами в URL, тілі запиту та заголовках.

Тестування взаємодії з базою даних

В процесі тестування взаємодії з базою даних було здійснено перевірку операцій читання, створення, оновлення та видалення даних з використанням різноманітних запитів та умов фільтрації. Особлива увага була приділена перевірці коректності збереження та отримання даних, обробці помилок та винятків, а також забезпеченню цілісності та узгодженості даних

Тестування інтеграції з CMS Contentful

Для забезпечення коректної роботи інтеграції бекенду з системою управління контентом Contentful було проведено ретельне тестування. Під час тестування особлива увага приділялася перевірці правильності взаємодії з API Contentful, забезпеченню належної обробки помилок та валідації отриманих даних.

Приклад тест-кейсу для перевірки обробки запиту на створення нового користувача

Інформація про тест-кейс	
Ідентифікатор тесту:	Тест-кейс №1
Опис функції:	Реєстрація нового користувача
Власник тесту:	Пінявєв Євгеній Володимирович
Дата створення:	01.06.2024
Мета тесту:	Перевірка процесу реєстрації нового користувача

Передумова

№	Опис випадку	Очікуваний результат	Висновок
1	Бекенд запущено та готовий до обробки запитів	Бекенд готовий до прийому запитів	Пройдено

Реєстрація нового користувача

№	Опис випадку	Очікуваний результат	Висновок
1	Відправити POST-запит з коректними даними користувача (ім'я, електронна пошта, пароль)	Бекенд успішно обробляє запит, створює новий обліковий запис користувача та повертає HTTP-статус 201 (Created) з інформацією про створеного користувача	Пройдено
2	Відправити POST-запит з некоректними даними користувача (наприклад, невалідна електронна пошта)	Бекенд успішно валідує вхідні дані, визначає їх некоректність та повертає HTTP-статус 400 (Bad Request) з описом помилки	Пройдено
3	Відправити POST-запит з відсутніми обов'язковими полями (наприклад, без електронної пошти або паролю)	Бекенд успішно валідує вхідні дані, визначає відсутність обов'язкових полів та повертає HTTP-статус 400 (Bad Request) з описом помилки	Пройдено

Результати тестування

Тестувальник:	Дата прогону тесту:	Результат тесту (П/Н):
Пінявєв Є.В.	01.06.2024	ПРОЙДЕНО (П)

Досягнення у роботі

- Проведено ретельний аналіз потреб користувачів та визначено детальні вимоги до функціональності.
- Застосовано сучасні технології та архітектурні підходи для забезпечення безпеки, масштабованості та продуктивності системи.
- Реалізовано основні функціональні можливості, забезпечено безпечне зберігання та ефективну обробку даних користувачів.
- Впроваджено механізми кешування, оптимізації запитів та алгоритми рекомендацій для підвищення продуктивності та реагування системи.

Публікація