Хмельницький національний університет
Факультет інформаційних технологій
Кафедра інженерії програмного забезпечення

ДИПЛОМНИЙ ПРОЕКТ

Інтернет-платформа «Агентство подорожей»

Назва теми

Рівень вищої освіти Перший(бакалаврський)

Галузь знань 12 «Інформаційні технології»

Спеціальність 121«Інженерія програмного забезпечення»

Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

Шифр ДПІПЗ.190160.19.11.ПЗ

Виконав студент III курсу група ІПЗс-19-1 _____ В.В. Собко
Підпис Ініціали, прізвище

Керівник канд. пед. наук, доцент _____ Н.І. Праворська
Науковий ступінь, звання Підпис Ініціали, прізвище

Нормоконтролер ст. викладач _____ Г.І. Бедратюк
Підпис Ініціали, прізвище

До захисту допускаю:
Завідувач кафедри інженерії
програмного забезпечення _____ Л. П. Бедратюк
Підпис Ініціали, прізвище

1 червня 2022 р.

Хмельницький 2022

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій
Кафедра Інженерії програмного забезпечення
Рівень вищої освіти Перший (бакалаврський)
Галузь знань 12 «Інформаційні технології»
Спеціальність 121 «Інженерія програмного забезпечення»
Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Л. П. Бедратюк
05 02 2022 р.

ЗАВДАННЯ
НА ДИПЛОМНИЙ ПРОЕКТ (РОБОТУ)

Собку Владиславу Вадимовичу
Прізвище, ім'я, по батькові студента

1. Тема проекту (роботи) Інтернет-платформа «Агентство подорожей»

Керівник проекту (роботи) Праворська Наталія Іванівна, канд. техн. наук, доцент
Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 05.02.2022 р. № 11
2. Строк подання студентом проекту (роботи) на кафедру 01.06.2022 р.
3. Вихідні дані до проекту (роботи) Матеріали переддипломної практики
4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)
Дослідження предметної області та постановка задачі, проектування програмного забезпечення, програмна реалізація, тестування програмного забезпечення
5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень)
Презентаційні матеріали (слайди, 20шт.)

6. Консультанти розділів дипломного проекту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Бедратюк Г.І., ст. викладач кафедри ІПЗ		
Антиплагіат	Гурман І. В., доцент кафедри ІПЗ		

7. Дата видачі завдання « 05 » лютого 2022р.

КАЛЕНДАРНИЙ ПЛАН

Назва етапів (розділів) дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1 Ознайомлення з тематикою дипломного проектування (ДП), визначення та узгодження індивідуальних там ДП	01.12 – 30.12.2021	
2 Дослідження предметної області, в якій планується використання програмного засобу (ПЗ), визначення задач та вимог, розробка технічного завдання	02.01 – 31.01.2022	
3 Проектування програмного забезпечення	01.02 – 28.02.2022	
4 Програмна реалізація	01.03 – 10.04.2022	
5 Тестування програмного забезпечення	11.04 – 30.04.2022	
6 Написання вступу, загальних висновків, оформлення джерел посилання та додатків. Оформлення пояснювальної записки ДП згідно вимог стандартів	01.05 – 25.05.2022	
7 Попередній захист ДП	Травень 2022 (згідно графіка)	
8 Перевірка ДП на плагіат, нормконтроль, отримання відгуків та рецензій. Брошурування (зшиття) пояснювальної записки	26.05 – 30.05.2022	
9 Підготовка до захисту та захист ДП	з 01.06.2022	

Студент _____ В.В. Собко
Підпис Ініціали, прізвище

Керівник проекту (роботи) _____ Н.І. Праворська
Підпис Ініціали, прізвище

АНОТАЦІЯ

Тема дипломного проекту: Інтернет-платформа «Агентство подорожей».

Автор проекту: Собко Владислав Вадимович.

Керівник проекту: Праворська Наталія Іванівна.

Пояснювальна записка: 117 с.,43 рис.,5 табл., 4дод., 19 джерел.

Графічна частина: 20 слайдів.

ПОДОРОЖІ,E-COMMERCE, JAVASCRIPTREST, MONGODB, NODEJS, REACT, REDUX, MERN, MVC,

Метою проекту є розробка програмного забезпечення, яке дозволятиме його користувачам автоматизувати процес пошуку та замовлення подорожей і турів, а також полегшить облік замовлень для адміністратора системи. Система має сучасний, інтуїтивно-зрозумілий користувачам інтерфейс.

У дипломному проекті було проаналізовано предметну область, з'ясовані причини частого виникнення проблем при замовленні подібних послуг не користуючись он-лайн-сервісами, проведено аналіз існуючих альтернативних програмних продуктів, а також були порівняні архітектурні рішення та патерни для розробки веб-додатків, визначені модулі системи та здійснена їх програмна реалізація.

Для розробки веб-додатку була використана мова програмування JavaScript, фреймворки React та Node.jsта база даних MongoDB.

В результаті було розроблено інтернет-платформу для замовлення турів та подорожей.

30.05.2022
Дата

Підпис

ВІДОМІСТЬ ДОКУМЕНТІВ

№ рядка	Формат	Позначення документа	Найменування документа	К-сть аркушів	№ екз.	Примітка
			Текстові документи			
1	А4	ДПІПЗ.190160.19.11.ВД	Пояснювальна записка	117		
2	А4		Завдання на дипломний	1		
			проект			
3	А4		Анотація	1		
			Графічні документи			
4	А4		Презентаційні матеріали	20		

					ДПІПЗ.190160.19.11.ВД			
Змн.	Арк.	№ докум.	Підпис	Дата				
Виконав		Собко В.В.		30.05	Інтернет-платформа «Агентство подорожей» Відомість документів	Літ.	Арк.	Акрушів
Керівник		Праворська Н.І.		1.06			1	117
Н. Контр.		Бедратюк Г.І.		1.06		ХНУ, ІПЗс-19-1		
Зав. Каф.		Бедратюк Л.П.		1.06				

ЗМІСТ

ВСТУП

В сучасному світі галузь туризму є однією з найбільш динамічних й прибуткових, поступаючись прибутками, на думку експертів, лише видобутку та переробці нафти. За даними міжнародної міжурядової організації в галузі туризму ЮНВТО, саме туризм забезпечує десять відсотків обороту світового виробничо-сервісного ринку, на цю сферу припадає шість відсотків світового ВВП, сім відсотків світових інвестицій, кожне шістнадцяте робоче місце, одинадцять відсотків світових споживчих витрат. Епоха глобалізації, інформатизації та комунікації і виникнення нового інформаційного суспільства у XXI столітті зумовили особливо інтенсивний розвиток туризму, в тому числі, й унаслідок нових умов взаємодії між країнами та регіонами щодо потужного обміну потоками інформації, технологій, фінансів, людських ресурсів, послуг, товарів. Урізноманітнився, власне, перелік різновидів туризму за рахунок нових напрямків: туризм корпоративний, діловий, конгрес-туризм, шопінг-туризм, паломницький, релігійно-пізнавальний, еко-туризм, сіті-туризм, пригодницький, освітній, екстремальний, круїзи і таке інше. Туроператори розширюють перелік споживачів турпродукту, пропонуючи варіанти подорожей, приміром, для літніх людей, людей з обмеженими можливостями, нудистів, сімейних пар з дітьми тощо.

Задовольнити такий попит і зростаючу кількість замовників туристичного продукту допомагає невпинний розвиток інформаційних технологій. Сучасний туризм є глобальною комп'ютеризованою індустрією, в якій задіяні великі авіакомпанії, комплекси готелів і турфірми всього світу. Діджиталізація лежить не лише в основі інформаційних систем управління, оперування та обміну базами даних, діяльності інтегрованих мереж та терміналів, розташованих в різних точках планети, а й забезпечує глобальні й національні системи продажів та бронювання туристичних послуг. В роботі з клієнтами – отримувачами послуг в галузі туризму ІТ-технології дозволяють

використовувати ефективні комунікаційні та маркетингові інструменти, надавати цілодобовий доступ до послуг, сучасних систем комунікації, надають можливості створення й застосування внутрішньо-корпоративних систем бронювання, нових маркетингових каналів просування турпослуг, отримання довідкових даних, запровадження он-лайн мультимедійних технологій, зокрема, каталогів, довідників.

Розвиток туристичної індустрії вимагає створення нових тревел-інструментів – інтерактивних додатків для зручного пошуку он-лайн, з комп'ютерів та смартфонів, до прикладу, квитків на літак, оформлення віз, страховок, бронювання готелів. Нормою життя стало автоматизоване обслуговування, і вже нині людина, маючи доступ до мережі Інтернет, може забронювати туристичний тур та заплатити за нього.

Актуальність даної теми полягає в тому, що споживач в таких умовах потребуватиме точної і повної інформації про туристичний продукт і доступні технологічні можливості отримання, обробки, комбінування та передачі інформації. Найкращим рішенням для цього була б проста і доступна для користувачів інтернет-платформа.

Мета проекту – розробка інтернет-платформи для агентства організації подорожей, вирішити проблеми з пошуком та замовленням путівок для звичайних користувачів, а також розроблене ПЗ стане чудовим бізнес-рішенням для обліку даних про продані путівки для агентств, які займаються організацією туризму та подорожей. Для досягнення поставленої мети були сформовані завдання на дипломне проектування та переддипломну практику:

– проаналізувати специфіку та особливості предметної області туристичного бізнесу;

– визначити та довести актуальність даної теми;

– дослідити існуюче програмне забезпечення, яке стосується даної предметної області, виявити його переваги та недоліки;

– скласти список функціональних та нефункціональних вимог до програмного забезпечення;

– провести аналіз інструментів та методів розробки ПЗ;

– розробити оригінальний, цікавий та зручний інтерфейс користувача;

– провести аналіз використання реляційних та нереляційних баз даних та розробити серверну частину інтернет-платформи;

– визначити специфіки фронт-енд рішень та розробити клієнтську частину інтернет-платформи;

– провести різні види тестування програмного забезпечення.

1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Змістовний аналіз предметної області, її структурних та функціональних особливостей

Предметною областю розробки веб-додатку є інформаційна система туристичної компанії.

В нашій країні чимало компаній займаються туристичним бізнесом. Ці компанії можна умовно поділити на турагентів та туроператорів. Туроператори здебільшого пропонують потенційному клієнтові-туристу сукупність туристичних послуг – певний перелік, який вони складають, купуючи окремі послуги у багатьох учасників сфери цього бізнесу, зокрема, у транспортних організацій, готельних комплексів, розважальних закладів та закладів культури тощо. Зазвичай вони пропонують унікальний туристичний продукт, складають договори з перевізниками та іншими представниками сервісу, приміром, готелями.

Крім цього, туроператори займаються організацією самих подорожей і обслуговують клієнтів в процесі подорожі. Тобто вони безпосередньо реалізовують путівку.

Фірми-турагенти, на відміну від туроператорів, є посередниками. Вони отримують дохід за рахунок комісійних коштів від продажу туристичних турів. На підставі агентського договору між туроператором і турагентом останній здійснює продаж путівок та займається реалізацією туристичних турів. Турагент, крім того, займається просуванням реклами, відповідає за збір, відправку, якість обслуговування клієнтів, перерахування грошей туроператору.

Туроператор дозволяє турагентові користуватися своїм каталогом туристичних пропозицій, рекламою та інформацією, і, крім того, гарантує виконання програми туру. Туроператор бере на себе зобов’язання повернути кошти клієнту в разі непередбачуваних обставин.

Оформлювати необхідну супутню документацію може як турагент, так і туроператор, відповідно до домовленості між ними. Згідно із законодавством, одна й та ж туристична фірма може бути і турагентом, і туроператором одночасно.

В свою чергу, серед фірм-туроператорів існує розмежування на компанії масового ринку і спеціалізовані фірми. Компанії масового ринку здійснюють послуги для всіх клієнтів, незалежно від типу подорожі, аудиторії, та інших критеріїв.

А, до прикладу, спеціалізовані оператори організовують подорожі лише в певні країни, або можуть орієнтуватися на певні цільові категорії споживачів туристичних послуг, як-от люди з особливими потребами і таке інше.

Власне, структуру кожної такої компанії визначає саме така спеціалізація. Адже мета діяльності зазначених фірм – максимально задовольнити потреби замовників, надаючи послуги, заявлені в путівці, екскурсії, розваги та інші послуги туризму й отримати при цьому прибуток. Таким чином, справа організації цього бізнесу, вимагає значних ресурсів: інформаційних, трудових, та фінансових.

На відміну від умов під час самостійного здійснення подорожей, яке обирає певна частина туристів чи мандрівників, користування послугами туристичних компаній дає їхнім клієнтам чимало переваг. Зокрема, такі подорожуючі мають можливість повністю запланувати деталі і маршрути своїх путівок і не хвилюватися за розподіл часу, коштів, логістику, комфортні умови, харчування, безпеку, медичну страховку або інші деталі чи непередбачувані обставини.

Отже, об’єктом розробки даної роботи є діяльність туристичної фірми та її робота з клієнтами.

Таким чином, було проведено детальний аналіз обраної предметної області, визначено значення таких понять, як туризм, туристична компанія, і виділено типи організацій, які надають туристичні послуги. Визначено і частково описано об’єкт розробки.

1.2 Аналіз наявного програмно-технічного забезпечення предметної області

Після проведення детального аналізу існуючого ПЗ, на його основі можна скласти такий список веб-сайтів, що користуються популярністю серед користувачів в нашій країні:

– «Ksamil Travel»;

– «Business Visit»;

– «Там тур»;

– «Відвідай»;

– «Країна ЮА: Тури за кордон».

Нижче описано властивості всіх зазначених веб-застосунків, з врахуванням їх функціональних та нефункціональних характеристик, особливостей, зовнішнього вигляду, переваг та недоліків, а також з досвіду особистого користування та відгуків користувачів.

Ksamil Travel – одна з найпопулярніших платформ для замовлення путівок.

З-поміж інших, цей додаток примітний своїм сучасним мінімалістичним інтерфейсом та притягальними для ока світлими кольорами (рисунки 1.1 та 1.2). Під час зміни розмірів екрану, структура елементів змінюється, але дизайн все ж залишається хорошим, елементи зручно розташовані як на екрані персонального комп’ютера, так і в смартфоні.

Сайт насичений розмаїтою пізнавальною інформацією для користувача. Усі тури та путівки детально розписано. Безперечно, покращує враження від відвідин вказаного сервісу наявність невеликої кількості анімації. Зручність полягає й у тому, що можна вибирати дату відправлення самостійно, не прив’язуючись до попередньо запланованих турів. Також є форма, де користувач може залишити відгук, зв’язатись з адміністрацією і навіть переглянути розташування офісу управління на карті.

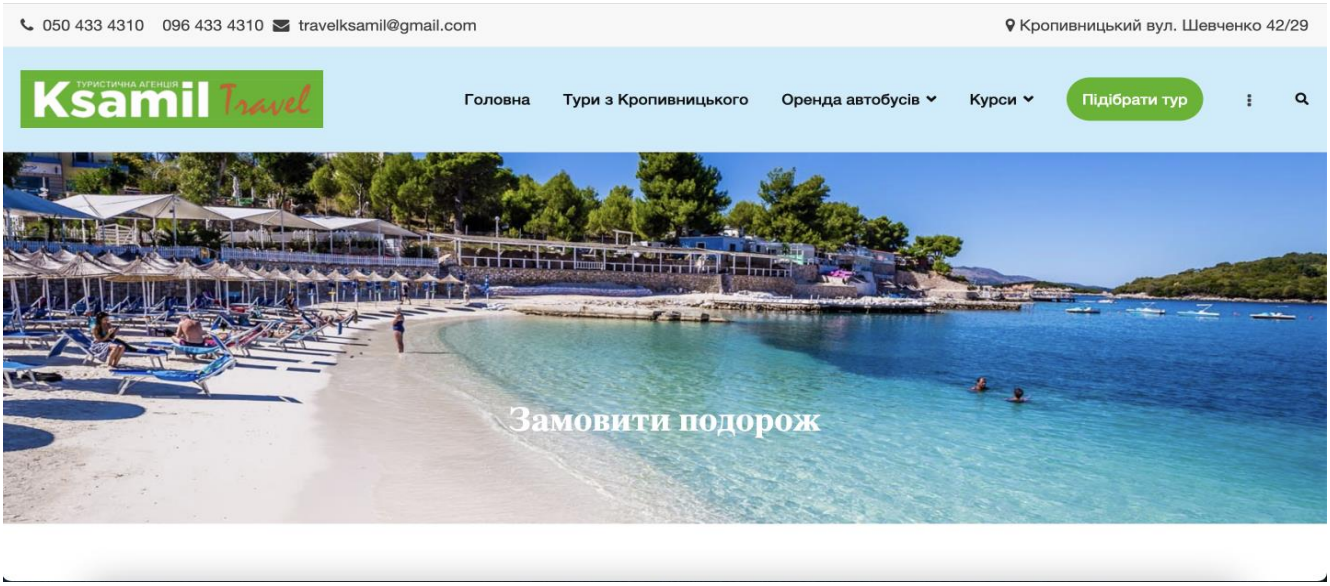


Рисунок 1.1 - інтерфейс сайту «Ksamil Travel»

Крім того, можна виділити наявність різних категорій, наприклад, тури по Україні, авіатури, морські та гірськолижні курорти, тури автобусом та інше. Проте з точки зору користувача є певні недоліки. Головною хибою, на нашу думку, є те, що весь контент на сайті доступний лише однією мовою.

До того ж, приміром, основний функціонал зосереджений у верхньому меню, і при виборі певної опції меню контент змінюється далеко внизу, а скролінг сторінки в цей момент не відбувається.

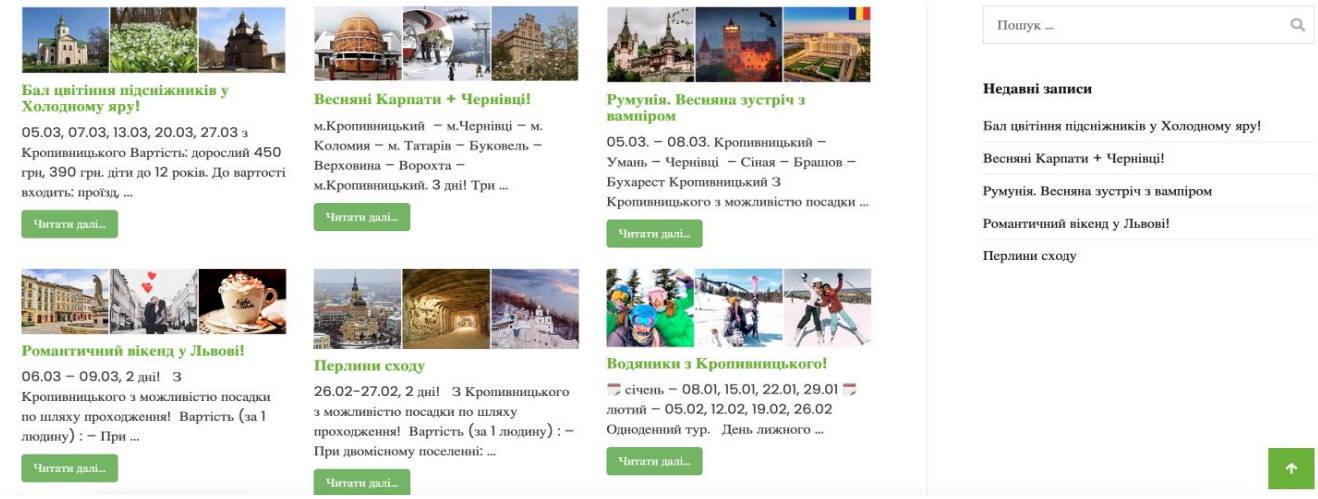


Рисунок 1.2 - інтерфейс сайту «Ksamil Travel»

Наступним розглянемо веб-сайт «Business Visit». До переваг цього ресурсу можна віднести вибір мов (наявні 3 мови), а також посилання на соціальні мережі (Facebook, Instagram та Youtube), де можна знайти більше інформації, переглянути більше фотографій з подорожей та відгуки користувачів. Головною перевагою і прикметою, що вирізняє цей сайт з-поміж інших, є те, що можна побачити графік зміни цін на конкретний тур, який посприяє користувачу визначитися з вибором.

Головним недоліком цього сайту (який виявлено в ході аналізу і порівнянь з іншими сервісами) є його інтерфейс (рисунки 1.3 та 1.4). Незважаючи на вдало підібрані кольори, шрифт не виглядає сучасним. Також багато зайвого тексту, який потрібно довго читати. Ще одна незручність полягає в тому, що внизу постійно з'являється реклама, яка перекриває контент, якщо її не закривати.

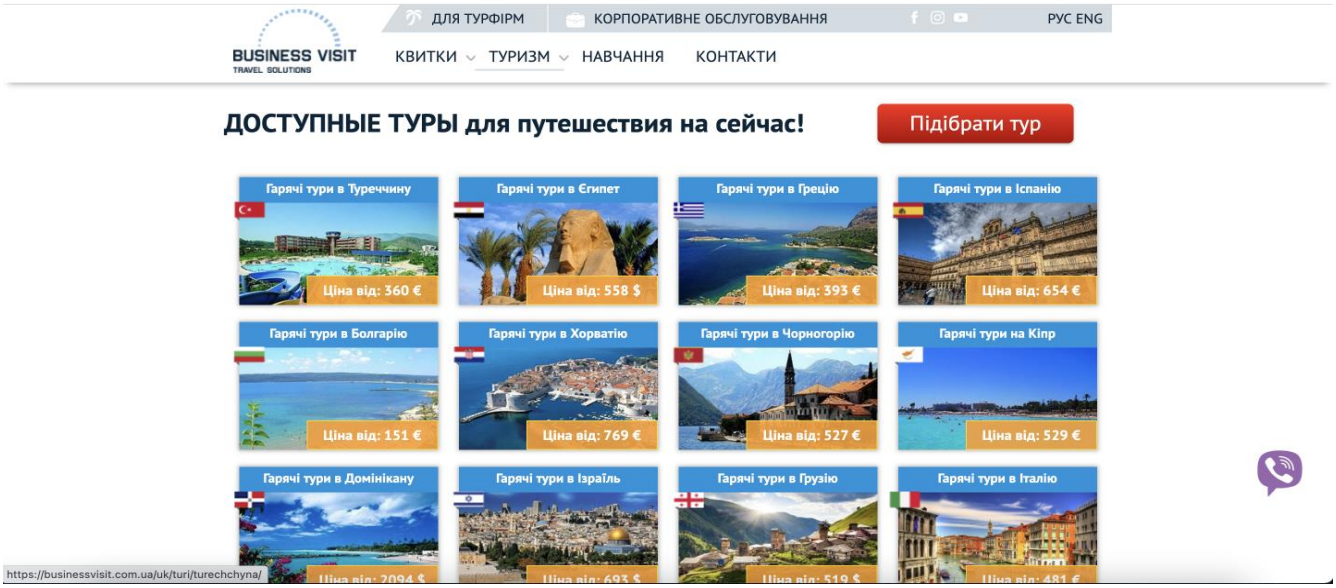


Рисунок 1.3 - Інтерфейс сайту «Business Visit»

Також істотним недоліком є відсутність пошуку та фільтрування турів. Користувач буде довго шукати потрібний йому тур. Суттєвим недоопрацюванням є те, що сайт іноді видає помилки. Наприклад, повідомлення з кодом 404, про те, що сторінку не знайдено, а отже, містить посилання на сторінки, яких не існує, що є негативним в плані досвіду для користувача.

Рисунок 1.4 - Інтерфейс сайту «Business Visit»

Наступним розглянемо веб-додаток «Там Тур».

Інтерфейс веб-сайту (рисунки 1.5 та 1.6) виконаний в спокійних зелено-салатових тонах, є достатньо приємним для очей.

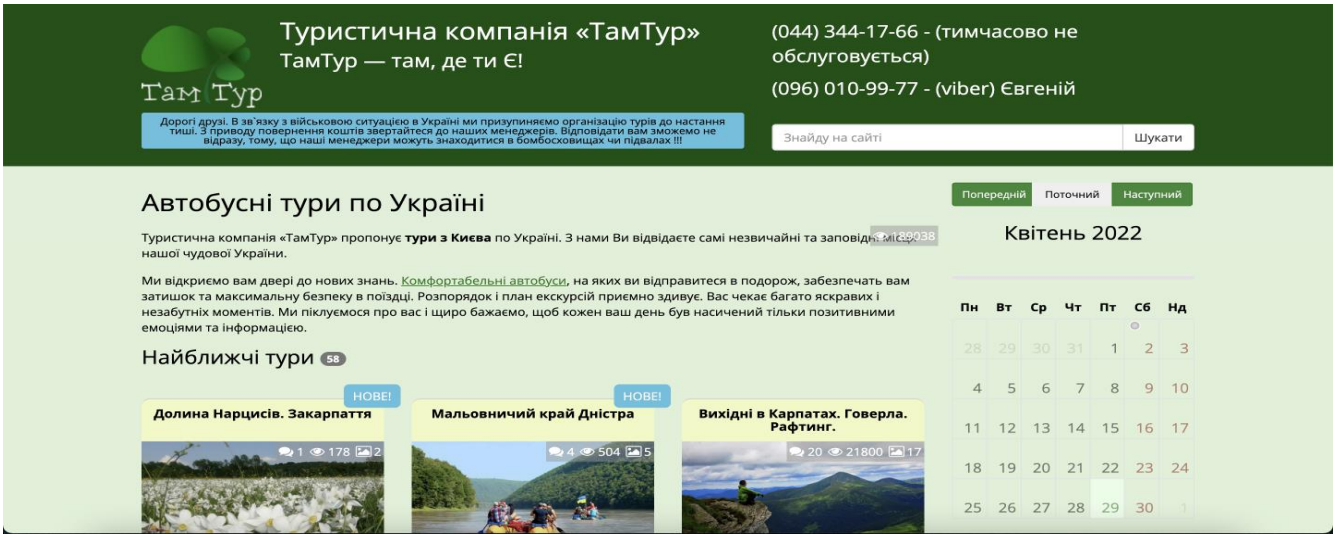


Рисунок 1.5 - Інтерфейс додатку «Там Тур»

Серед головних переваг – детально розписана програма усіх турів, наявність відгуків інших користувачів одразу під інформацією про вибрану подорож, відсутність реклами. У правій частині сайту розміщений календар, на якому можна одразу побачити, на які дні плануються тури. Головною особливістю вказаного сайту є те, що користувач може одразу роздрукувати

усю інформацію про тур, а також напряму зв’язатись з людиною, яка відповідальна за цей тур, адже є вказаний її номер телефону.

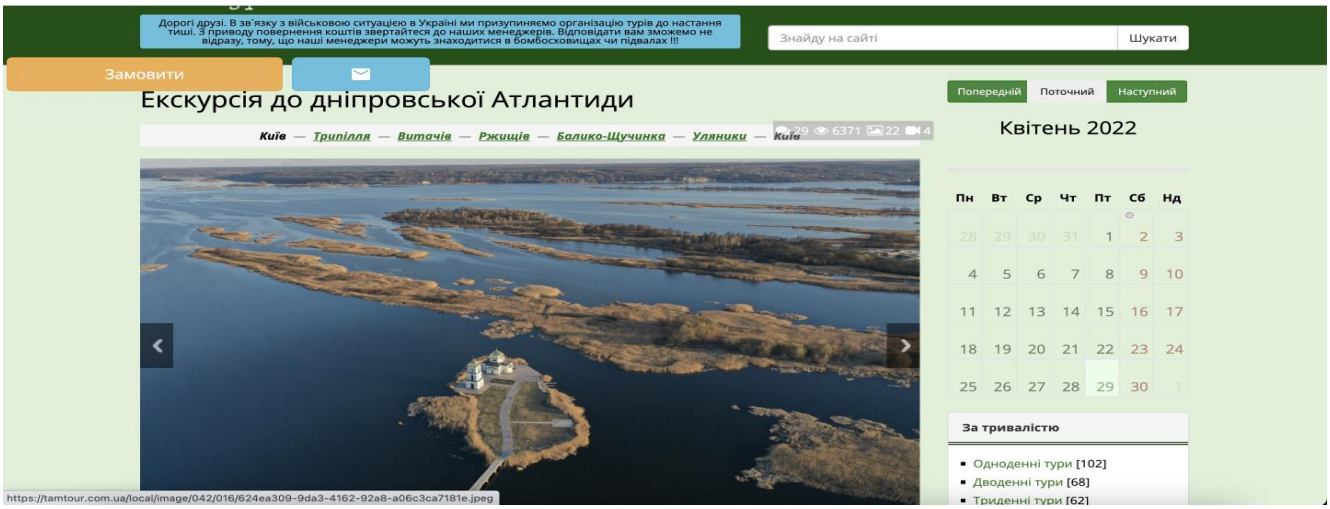


Рисунок 1.6 - Інтерфейс додатку «Там Тур»

Основними недоліками є розміщені у верхній частині сайту кнопки «Зробити замовлення» та «Надіслати відгук», які не скроляться, і тим самим перекривають контент. Також у верхньому блоці частина інформації відображається не достатньо великим шрифтом, її не дуже зручно читати. Що стосується функціоналу, не зручним фактором є відсутність фільтрації, хоча це й частково компенсується наявністю пошуку. Також на цьому сайті немає системи оплати.

Наступним веб-сайтом, який вибрано для аналізу, виступає «Відвідай ЮА». Цей веб-застосунок дає змогу переглядати контент аж на чотирьох мовах. Інтерфейс додатку (рисунки 1.7 та 1.8) не містить зайвого, мінімалістичний, оформлений у градієнтних кольорах світлої палітри.

З переваг можна виокремити наявність категорій та підкатегорій, що робить цей веб-додаток унікальним в даному списку рішень. Також є зручний пошук, можливість завантажити на пристрій текстовий опис в форматі docx і переглянути коротке відео про головні місцевості даної подорожі. Весь контент доступний для перегляду без обмежень, навіть для користувачів без облікового.

Реклама майже не зустрічається. Розміщені посилання на соціальні мережі, де можна побачити більше фото та відео і почитати відгуки користувачів.

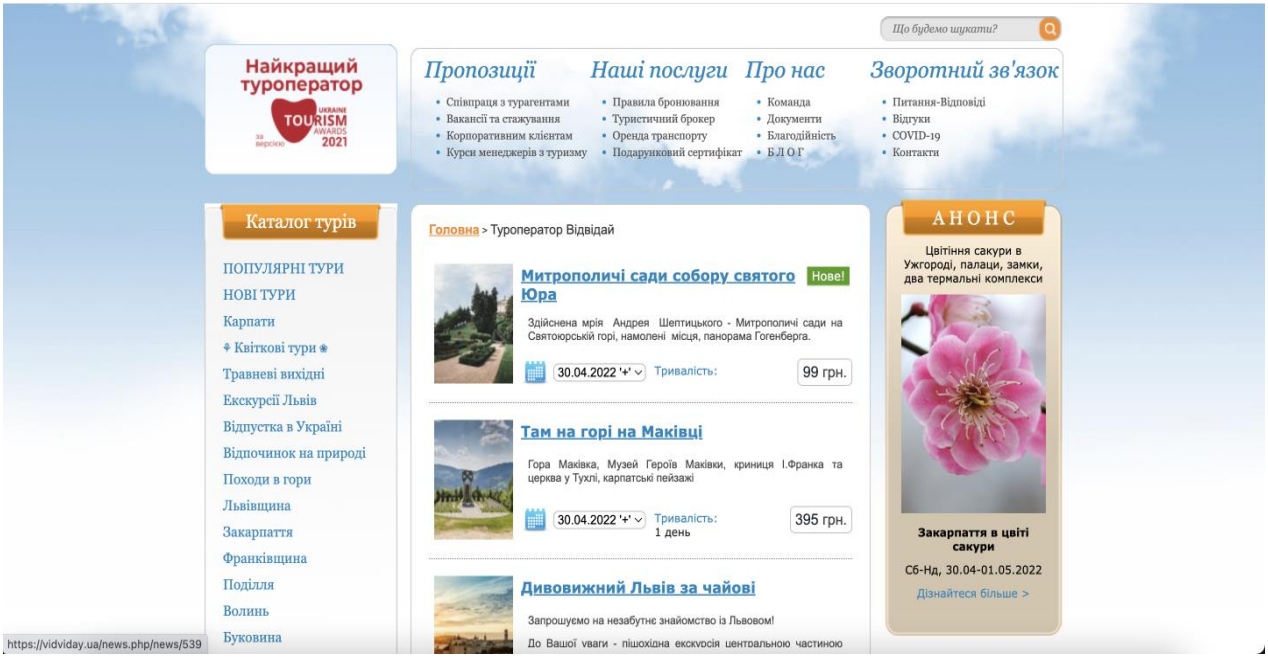


Рисунок 1.7 - Інтерфейс додатку «Відвідай ЮА»

Головний недолік вбачається у тому, що на мобільних пристроях розмітка сайту не змінюється, а лише пропорційно зменшується, що робить розмір шрифту та зображень надто малим для перегляду та читання. Також відсутня система оплати і фільтрування даних.

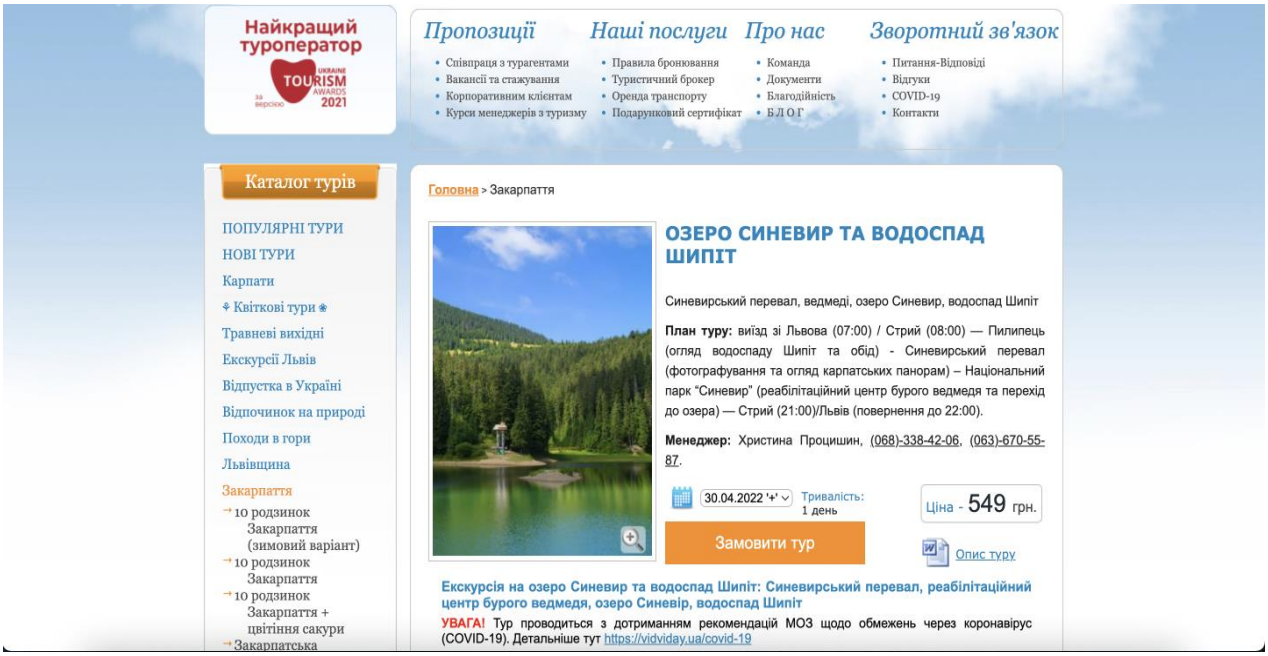


Рисунок 1.8 - Інтерфейс додатку «Відвідай ЮА»

Наостанок, проведено аналіз веб-сайту «Країна ЮА».

До переваг цього веб-додатку можна зарахувати можливість вибрати та забронювати дату подорожі і те, що під кожним туром користувачеві запропоновані схожі тури, що є чудовим рішенням для бізнесу, адже надає користувачам додаткові преференції. Ключовою особливістю є розміщена пропозиція, щодо замовлення подарункового сертифікату. Серед переваг також слід назвати наявність інструкції щодо користування сайтом, відгуки користувачів, посилання на соціальні мережі та багато іншого. Є форма для зворотного зв'язку, де можна зв'язатись з керівництвом або надіслати відгук. Присутня можливість підписатися на розсилку повідомлень, щоб інформація про всі нові тури надходила на електронну адресу.

Головним недоліком є відсутність фільтрації даних та категорій, хоча пошук – присутній. Також проблемою є те, що на кожну дату в календарі зазначена кількість турів, проте перевірити, які саме це тури, неможливо, адже перехід з календаря не працює. Інтерфейс цього сайту (рисунок 1.9 та 1.10) є застарілим, а гамма кольорів нецікавою і непривабливою.

Крім того, для замовлення подорожей, наявність облікового запису є обов'язковою.

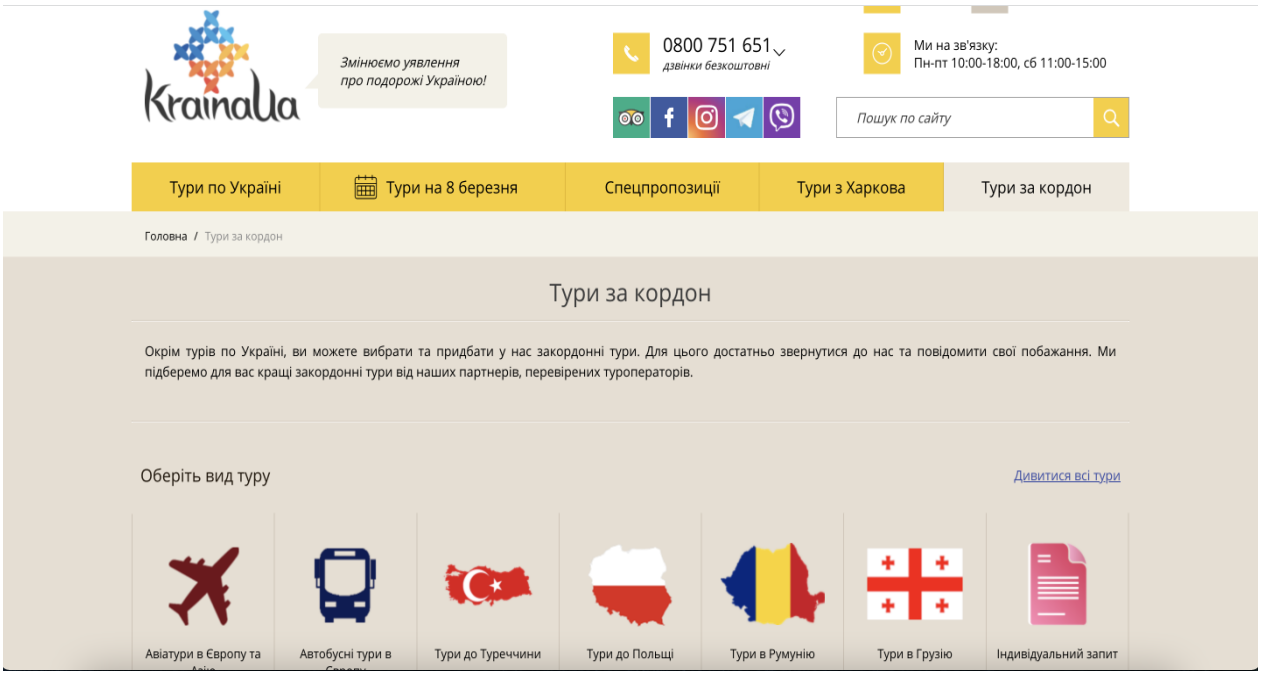


Рисунок 1.9 - Інтерфейс сайту «Країна ЮА»

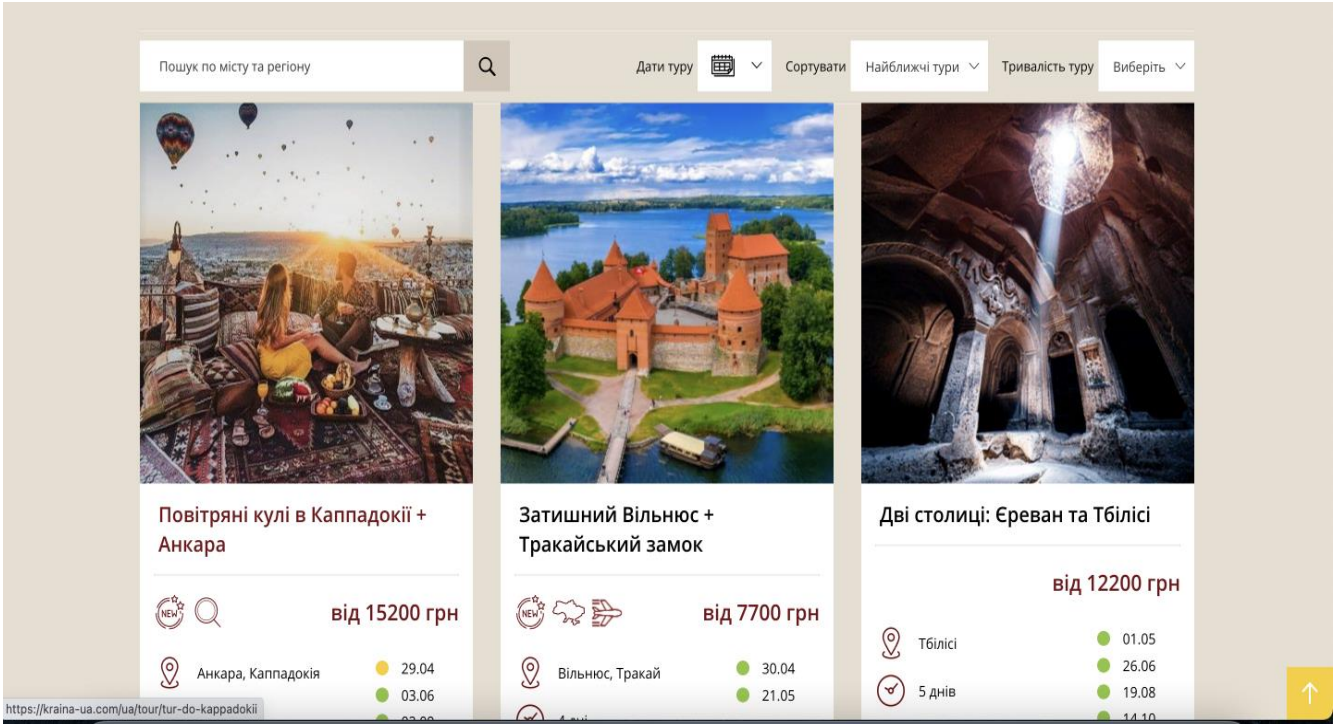


Рисунок 1.10 - Інтерфейс сайту «Країна ЮА»

Здійснивши детальний розбір всіх схожих веб-застосунків можна здійснити порівняння всіх розглянутих сайтів за різними властивостями. Головною властивістю сайтів є їх зручність, а саме, можливість швидко знайти потрібний продукт в переліку. Далі, надважливою, є оцінка інтерфейсу, адже це перше, на що користувач звертає увагу, а ще те, скільки на сайті є реклами і наскільки вона нормальна працює. Також дуже важливо порівняти такі функції, як наявність пошуку та присутність фільтрування, можливість самостійно вибирати дні подорожі, та наявність системи оплати, для того, щоб людина, яка користується системою, могла одразу оплатити он-лайн своє замовлення.

В таблиці 1.1 зроблено порівняння сайтів за цими властивостями.

Таким чином, провівши детальний аналіз ринку існуючих рішень для фірм, які займаються продажем подорожей, можна зробити висновки. Багато з цих веб-застосунків мають зрозумілий та простий для користувачів, однак не зовсім сучасний інтерфейс, який в наш час вже потребує оновлення. Також, як з’ясувалося, абсолютна більшість сайтів дозволяє користувачу виконувати фільтрацію та пошук турів.

Таблиця 1.1 – Порівняльна таблиця додатків

Назва додатку	Пошук та фільтрація	Зручність інтерфейсу	Реклама	Можливість вибрати дату	Наявність системи оплати	Обов'язкова авторизація
Ksamil Travel	Присутні	6/10	Відсутня	Присутня	Присутня	Так
Business Visit	Відсутні	5/10	Присутня	Відсутня	Відсутня	Ні
Там Тур	Присутні	7/10	Відсутня	Присутня	Відсутня	Ні
Відвідай ЮА	Присутні	8/10	Присутня	Присутня	Присутня	Ні
Країна ЮА	Присутні	5/10	Відсутня	Відсутня	Присутня	Так

З негативних чинників можна зазначити те, що у більшості рішень зустрічається реклама, як пов'язана з подорожами, так і ні. Крім того, на більшості застосунків відсутня оплата, користувач може тільки надіслати свої контактні дані, після чого оператор або дзвонить йому, або пише на електронну пошту. Отже, проведений детальний аналіз існуючих схожих рішень та їх властивостей допоміг визначити основні характеристики, функціональні та нефункціональні вимоги до розроблюваного проекту.

1.3 Визначення вимог до програмного продукту

За підсумками детального дослідження предметної області та існуючих схожих програмних систем, було розроблено технічне завдання та список функціональних та нефункціональних вимог.

Створюваний веб-сайт для замовлення подорожей мусить мати декілька особливостей порівняно з його аналогами. Головною відмінністю від інших

подібних рішень є те, що платформа повинна бути універсальною, складатися з двох частин: клієнтської та адміністраторської. Також має бути передбачена авторизація та реєстрація для ідентифікації користувачів. Не авторизованим користувачам має бути створено можливість побачити каталог подорожей. Додати в корзину та замовити подорож, повинні мати змогу лише авторизовані користувачі. Ще, має бути можливість як додати замовлення в корзину, так і видалити з неї, на випадок, якщо користувач передумає замовляти. Перед оформленням замовлення має відбуватись оплата з використанням надійної системи для обробки платежів PayPal.

Крім цього, користувачеві слід надати можливість переглядати власні попередні замовлення. Для адміністратора буде передбачено змогу переглянути замовлення всіх користувачів. В каталозі товарів повинен бути пошук за назвою подорожі, або її опису, а також фільтр по континентах та за ціною.

Мова веб-додатку має бути англійською, адже ним можуть послуговуватися користувачі з різних країн.

Беручи до уваги всі зазначені вище вимоги, можна скласти перелік функціональних можливостей, які має надавати програма:

Для звичайного користувача:

- реєстрація;
- авторизація;
- перегляд списку турів;
- фільтрація за ціною та континентом;
- пошук за назвою та описом;
- перегляд галереї фотографій вибраного місця;
- детальний перегляд вибраної путівки;
- додавання в корзину;
- видалення з корзини;
- замовлення путівки з оплатою;
- перегляд списку своїх замовлень;

Для адміністратора:

– авторизація;

– перегляд всіх замовлень всіх користувачів;

– додавання нової подорожі з завантаженням фото;

– перегляд списку турів;

– пошук за назвою та описом;

– фільтрація за ціною та континентом;

– детальний перегляд вибраної путівки;

Також сформовано наступні вимоги до зовнішнього інтерфейсу користувача та інші вимоги:

– адаптивність до мобільних пристроїв;

– основні кольори – темні, світлий текст для зручного користування при будь-якому освітленні;

– початковою при користуванні має бути сторінка авторизації;

– на головній сторінці повинні бути зручні фільтри та пошук;

– при кліку на фотографію, повинна відкриватися галерея фотографій зі слайдером;

– пошук повинен відбуватись як за описом путівки, так і за назвою, залежно від того, що введе користувач в полі пошуку;

– все повинно розміщуватися зрозуміло інтуїтивно для користувачів, щоб вони могли вільно користуватися платформою навіть без керівництва користувача.

Для того, щоб чітко сформувати вимоги до програмного забезпечення використовується уніфікована мова моделювання UML.

Діаграма UML – це діаграма, створена на основі UML (Unified Modeling Language) з ціллю візуального зображення системи, з її сутностями, такими, як ролі, дії актори, ролями, діями, класи, для цілісного розуміння або документування інформації про системи. Вона складається з схематичних представлень елементів програми.

Складемо описи акторів (користувачів системи) та усіх можливих їхніх дій (варіантів використання).

У таблиці 1.2 наведені актори розроблюваного програмного засобу.

Таблиця 1.2 – Опис акторів розроблюваного ПЗ

Актор	Короткий опис
Незареєстрований користувач (гість)	Може зареєструватись або авторизуватись, має доступ до головної сторінки сайту, може використовувати пошук та фільтри, дивитись детальну інформацію про тури та подорожі, а також переглядати галерею світлин.
Зареєстрований користувач	Має змогу додавати вибрані тури до корзини, робити замовлення та оплачувати їх, переглядати список своїх замовлень, а також все те, що й гість
Адміністратор	Може додавати нові тури, переглядати список замовлень всіх користувачів

В таблиці 1.3 наведені варіанти використання розроблюваного ПЗ.

Таблиця 1.3 – Опис варіантів використання розроблюваного ПЗ

Актор	Найменування ВВ	Опис ВВ
1	2	3
Гість	Реєстрація	Користувач має можливість зареєструватися в системі.
	Перегляд турів	Користувач може бачити всю інформацію про тур, включно з детальним описом та галереєю фотографій
	Пошук та фільтрація	Користувач має змогу застосовувати фільтри за континентом та ціною, і робити пошук за назвою та описом

Кінець таблиці 1.3

Зареєстрований користувач	Авторизація	Зареєстрований користувач має можливість увійти в систему.
	Перегляд турів	Користувач може бачити всю інформацію про тур, включно з детальним описом та галереєю фотографій
	Пошук та фільтрація	Користувач може застосовувати фільтри за континентом та ціною, і робити пошук за назвою та описом.
	Додавання в корзину	Користувач має можливість додавати тури в корзину.
	Замовлення та оплата	Користувач має можливість замовити додані до корзини тури та оплатити замовлення.
	Перегляд списку своїх замовлень	Користувач може переглянути свої попередні замовлення.
Адміністратор	Авторизація	Адміністратор має можливість увійти в систему.
	Додавання нового туру	Адміністратор має змогу створити новий тур.
	Перегляд списку замовлень	Адміністратор може переглядати замовлення усіх користувачів

Визначивши акторів системи та варіанти використання, було побудовано діаграму варіантів використання (рисунок 1.11).

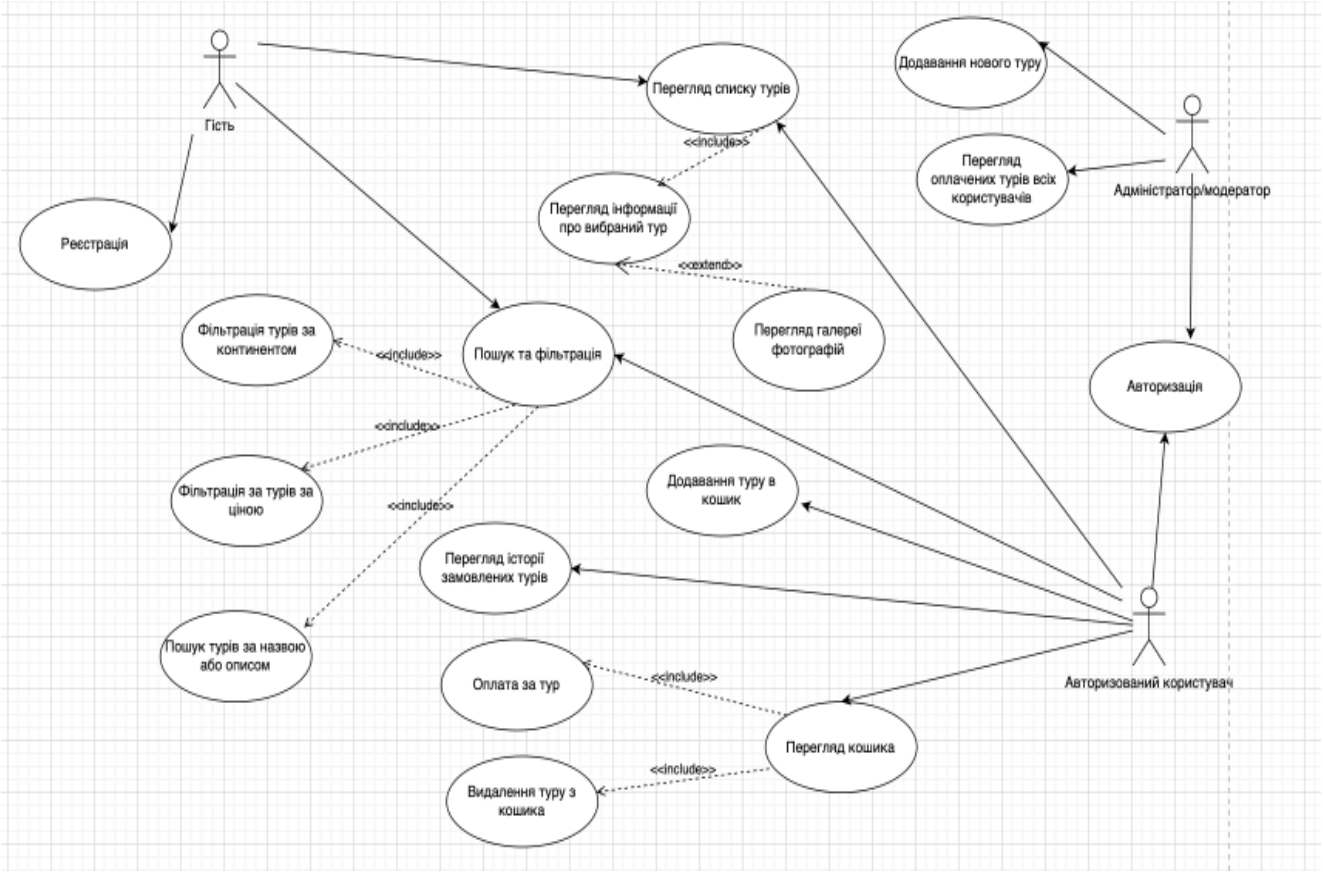


Рисунок 1.11 - Діаграма варіантів використання

Закінчивши аналіз вимог до програмного продукту, було створено технічне завдання, яке представлене у додатку А і взяте за основу для подільшої розробки сервісу для турфірми.

В даному розділі було розглянуто особливості предметної області, знайдено проблеми, що існують в даній предметній області та шляхи їх вирішення. Проаналізовано існуючі програмні засоби, виявлено їх переваги та недоліки. Складено список функціональних та нефункціональних вимог до розроблюваного проекту.

2 ПРОЕКТУВАННЯ ВЕБ-ДОДАТКА

2.1 Аналіз та вибір архітектури веб-додатка

Більшість веб-сайтів, на які щодня є можливість знайти в мережі Інтернет, побудовані з використанням клієнт-серверної архітектури. Даний тип архітектури включає в себе в два основні компоненти – клієнт та сервер. Основним принципом та головною перевагою даного підходу є те, що більшість інформації розташовано на сервері, що дає змогу швидше працювати клієнту. Під клієнтом мається на увазі будь-який програмний засіб, який має зовнішній інтерфейс та може надсилати запити для передачі та отримання даних від сервера – комп'ютера, або програмного забезпечення, яке перебуває віддалено.

Сервер – це апаратне забезпечення, яке слугує для отримання запитів від клієнтів, їх обробки та відправки даних у потрібному для клієнта форматі, а також він виконує багато різноманітних задач, пов'язаних з проведенням складних обчислень та інших логічних завдань. Сервер зазвичай є дуже потужною комп'ютерною системою, адже він мусить бути спроможним приймати та обробляти велику кількість інформації одночасно, яка надходить від різних клієнтів. Ці запити обробляються по черзі і клієнтам надсилаються відповіді, тому це повинно працювати якнайшвидше. Як правило, сервери працюють постійно, адже клієнти можуть надсилати запити хоч цілодобово.

Існують випадки, коли програма-клієнт та програма-сервер розміщені на одному комп'ютері, такий підхід називається використанням локального сервера. Його доцільно використовувати в процесі розробки програмного забезпечення. Та все ж, в більшості випадків клієнт та сервер знаходяться на різних комп'ютерах та можуть бути навіть в різних куточках світу, а для комунікації між собою використовують всесвітню мережу Інтернет та протокол передачі даних HTTP - HyperTextTransferProtocol – це протокол, який призначений для передачі даних в мережі. Ці дані являють собою гіпертекст, який прийнято називати HTML, повна назва - Hyper Text Markup Language.

Отже, обробивши запит клієнта, сервер відправляє йому у відповідь HTML, а клієнт, в свою чергу зберігає його в себе локально та показує на екрані користувачу.

Пізніше був винайдений розширений протокол передачі даних, який називається HTTPS (HyperTextTransferProtocolSecure). Цей протокол використовує SSL (Secure Sockets Layer) для шифрування даних, які клієнт надсилає серверу, а сервер – клієнту. Це шифрування забезпечує захист конфіденційної інформації, створюючи, таким чином, захищене та надійне з'єднання. На даний момент саме він використовується для більшості веб-сайтів, хоча раніше він був доступний тільки для банківських систем та інших, які містять персональну інформацію користувачів.

Зі зростанням кількості розроблених додатків, які використовують клієнт-серверну архітектуру, почали з'являтись різні архітектурні рішення для створення серверних прикладних програмних інтерфейсів (API) та керування ними. Найпоширенішим серед них став шаблон REST (Representational State Transfer) – передача стану представлення. Його особливістю є те, що він дозволяє виконувати обробку запитів відповідно до того, яким методом було надіслано дані. Метод вказує, що саме потрібно клієнту від сервера – отримання, оновлення, видалення, або ж створення певних даних. Таких чином сервер розуміє, що конкретно йому треба зробити і не виконує зайвих дій, наприклад, оновлення даних, якщо користувачу потрібно лише отримати їх. Крім цього, архітектура REST дає змогу надсилати клієнтам інформацію про статус виконання запиту у вигляді коду із трьох цифр. Наприклад, у разі успішного виконання запиту користувач отримає код «200», а якщо з якихось причин запит не дасть очікуваний результат, сервер поверне клієнту код «500» або «404» в залежності від типу помилки при виконанні цього запиту.

Також, використовуючи REST, не виникає ніяких обмежень на розмір та формат представлення ресурсів – документів, які передаються через мережу.

Отже, сервер – це сервіс, який призначений для обробки, зберігання та відправки даних, які отримуються та відправляються за допомогою HTTP та HTTPS запитів. Клієнт – це будь-яка програмна система, яка може створювати, відправляти такі запити та отримувати від них відповіді.

Така архітектура дозволить отримувати клієнту будь-які дані та ресурси, розміщені на сервері.

Таким чином, для реалізації завдання дипломного проекту, було обрано клієнт-серверну архітектуру та шаблон REST, який забезпечить створення та використання протоколу передачі даних через мережу.

2.2 Опис структури даних та моделі бази даних

Переважна більшість сучасних веб-сайтів зберігають інформацію про користувачів та іншу важливу інформацію у сховищах різних типів. Такі сховища називаються базами даних. Бази даних дають можливість забезпечити надійне збереження даних та доступ до них у разі потреби. Дуже важливо на етапі проектування програмного забезпечення вибрати базу даних, яка буде найкраще підходити для вирішення поставленої задачі, адже це вплине на багато важливих факторів, таких як, швидкість роботи системи, можливість безперешкодної підтримки програмного продукту в майбутньому та багато іншого.

На теперішній час системи управління даними діляться на два основні види – реляційні та нереляційні бази даних.

Реляційні бази даних є найбільш поширеними, адже збережена в них інформація структурується в набір таблиць. Кожна таблиця складається з рядків, в яких знаходяться поля – ключі, по яких можна отримати дані про певний об’єкт або сутність. Ці поля зберігають в собі атрибути об’єкта. Стовпці таблиці описують характеристики цих атрибутів. Кожне поле має строго

визначений тип даних та описує лише одну властивість об’єкта, наприклад, електронна пошта, назва книги, номер телефону, тощо.

Кожна таблиця в реляційній базі даних мусить мати ключовий атрибут, який буде її однозначно ідентифікувати. Такий атрибут називають первинним ключем.

Для реляційних баз даних притаманні свої переваги та недоліки. Перевагами вважають такі чинники:

− легке розширення бази даних при збільшенні кількості даних;

− наявність великої кількості спеціальних програм для доступу до баз даних, які називають СКБД – системи керування базами даних, які надають адміністратору бази даних різні корисні функції, такі як очищення таблиць, сортування та пошук даних та багато іншого;

− чітко визначена структура бази даних, яка дає цілком визначене уявлення про предметну область та сутності;

− зрозумілий для користувачів спосіб представлення даних за допомогою таблиць та зв’язків.

Серед недоліків реляційних баз даних виділяють наступні фактори:

− багато таблиць та зв’язків у великих проектах можуть заплутати та спричинити труднощі з розумінням структури бази даних;

− не найвища швидкість запису та витягування даних;

− дані бази даних займають велику кількість фізичної пам’яті, при чому чим більше в базу додається даних, тим більше пам’яті буде зайнято, що й спричиняє низьку швидкість роботи;

− предметна область не завжди може буде описана належним чином за допомогою таблиць.

Реляційні БД мають відповідати ряду вимог:

− атомарність – кожен атрибут повинен мати лише одне значення, а не множину значень;

− цілісність – пов’язаність всіх частин бази в єдине ціле;

− ізольованість – сутності бази даних не повинні залежати одні від інших;

− довговічність – властивість, яка гарантує, що після додавання запису в базу, або його зміни, дані будуть збережені та відновлені у випадку виникнення збоїв у роботі.

Для того, щоб полегшити роботу з базами даних, використовують різні СКБД: SQLLite, MySQL, PostgreSQL, MariaDB, тощо. Вони надають можливість працювати з даними безпосередньо в базі, а саме, створювати, знаходити, редагувати та видаляти конкретні дані в БД, забезпечуючи безпечний доступ.

Крім реляційних баз даних, існують ще нереляційні. Їх також називають NoSQL. Вони пропонують зовсім іншу концепцію до створення бази даних та доступу до неї. Їх особливістю є те, що дані не структуровані в табличному вигляді, що дає більш гнучкі можливості управління даними. Це в основному корисно для роботи з величезними наборами розподілених даних. Бази даних NoSQL мають масштабований, високопродуктивний та гнучкий характер. NoSQL дедалі більше набирає популярності, оскільки використовується в додатках з великою кількістю даних, яка постійно зростає та в реальному часі.

Існують різні типи баз даних NoSql, серед яких виділяють 4 основних:

− бази даних документів зберігають дані в документах, подібних до об’єктів JSON (JavaScript Object Notation). Кожен документ містить пари полів і значень. Зазвичай значення можуть бути різних типів, включаючи рядки, числа, логічні значення, масиви або об’єкти. Така система ієрархічно зберігає дані, адже має структуру дерева. Це дає змогу здійснювати доволі швидкий пошук по всій базі даних. Цей тип буде корисним для зберігання впорядкованих даних та у випадку, коли немає великих кількості зв’язків між даними та відсутня потреба збору статистики. Прикладами СУБД даного типу є: MongoDB, eXist, CouchDB, MarkLogic;

− бази даних "ключ-значення" – це простіший тип бази даних, де кожен елемент містить ключі та значення. При цьому зв'язок між значеннями відсутній і немає ніяких обмежень щодо кількості ключів та значень. Така властивість стала причиною для обрання цього типу бази даних для розробки хмарних

сервісів. Поширеними СУБД даного прикладу є DynamoDB, MemcacheDB, Berkeley DB та Riak;

− сховища з BigTable – зберігають дані в таблицях, рядках і динамічних стовпцях у вигляді розрідженої матриці. Як правило, зазначений тип БД використовується здебільшого у величезних веб-ресурсах або ж в інших платформах, які працюють з великим обсягом даних. Прикладами таких СУБД є: Hypertable, Cassandra, Hbase;

− графові бази даних зберігають дані у вузлах і ребрах. Вузли зазвичай зберігають інформацію про людей, місця та речі, тоді як ребра зберігають інформацію про відносини між вузлами. Відомими графовими СУБД є: ArangoDB, Giraph, Neo4j, HyperGraphDB.

MongoDB – це документно-орієнтована база даних NoSQL, яка використовується для зберігання великих обсягів даних. Замість використання таблиць і рядків, як у традиційних реляційних базах даних, MongoDB використовує колекції та документи. Документи складаються з пар ключ-значення, які є основною одиницею даних у MongoDB. Колекції містять набори документів і функцій, що є еквівалентом таблиць реляційної бази даних. MongoDB з’явилася приблизно в середині 2000-х років.

Особливості MongoDB:

– кожна база даних містить колекції, які в свою чергу містять документи. Кожен документ може відрізнятися різною кількістю полів. Розмір і зміст кожного документа можуть відрізнятися один від одного;

– структура документа більше відповідає тому, як розробники будують свої класи та об’єкти на відповідних мовах програмування. Розробники часто кажуть, що їхні класи не є рядками і стовпцями, а мають чітку структуру з парами ключ-значення;

– рядки (або документи, які називаються в MongoDB) не повинні мати заздалегідь визначену схему. Натомість поля можна створювати на льоту;

– модель даних, доступна в MongoDB, дозволяє розробнику легше представляти ієрархічні зв’язки, зберігати масиви та інші складніші структури;

– масштабованість. Середовища MongoDB дуже масштабовані. Компанії по всьому світу визначили кластери, деякі з яких працюють із понад 100 вузлами з близько мільйонами документів у базі даних.

Проаналізувавши переваги і недоліки різних видів баз даних, було вирішено розробляти завдання дипломного проекту з використанням нереляційної бази даних MongoDB. Це забезпечить швидкий доступ до даних, незалежно від їх кількості, та відкине потребу детально проектувати таблиці та створювати зв'язки між ними, а також, скоротить терміни розробки та проектування.

Беручи за основу створену раніше діаграму варіантів використання (див. рисунок 1.11), було визначено дані, які будемо зберігати в базу даних, а саме дані про користувача, ролі користувачів, тури, корзина, замовлення та оплати. Таким чином було створено колекції даних «User», «Role», «Product», «OrderHistory», «Cart» та «Payments».

User-колекція користувачів, які створили власний обліковий запис в розроблюваній програмній системі. Має наступні поля:

– Id – унікальний ідентифікатор користувача;

– Name – ім'я користувача;

– LastName – прізвище користувача;

– Email – електронна адреса користувача, на яку зареєстрований обліковий запис;

– Password – хеш (закодована послідовність символів) паролю користувача, який використовується для входу в обліковий запис;

– CreatedAt – дата реєстрації користувача;

– RoleId – ідентифікатор ролі користувача, це може бути або звичайний користувач або ж адміністратор.

Role – дана колекція вказує ролі, які існують в системі, а саме адміністратор, або звичайний користувач, та містить такі поля:

– Id – унікальний ідентифікатор ролі;

– Title – назва ролі;

Product – дана колекція позначає подорожі та тури, які додає адміністратор, а користувач може замовити. Ця колекція містить наступні поля:

– Id – унікальний ідентифікатор подорожі;

– Title – назва подорожі;

– Description – опис подорожі;

– Price – ціна подорожі;

– Images – масив з фотографіями місць подорожі;

– Continents – континент, на якому розташоване місце подорожі;

– SoldCount – кількість замовлень даної подорожі;

– CreatedAt – дата створення подорожі.

Cart – дана колекція вказує тури, які певний користувач додав собі в корзину з метою подальшого їх замовлення, та містить такі поля:

– Id – унікальний ідентифікатор створеної корзини;

– UsedId – унікальний ідентифікатор користувача, який додав товар до корзини;

– ProductId – унікальний ідентифікатор туру в корзині;

– Quantity – кількість путівок в замовленому турі;

– Date – дата додавання туру в корзину;

Payments – дана колекція описує дані про оплату користувачем подорожей при здійсненні замовлення, та містить наступні поля:

– Id – унікальний ідентифікатор оплати користувачем подорожі;

– UsedId – унікальний ідентифікатор користувача, який оплачує путівку;

– ProductId – унікальний ідентифікатор путівки, яка оплачується;

– Paid – логічне поле, що позначає чи була здійснена оплата;

– - Canceled – логічне поле, яке позначає чи була відмінена оплата;

– PaymentAdress – місце звідки здійснювалась оплата;

– CreatedAt – дата оплати подорожі користувачем;

OrdersHistory – дана колекція відповідає за історію замовлень турів всіма користувачами, та містить такі поля:

– Id – унікальний ідентифікатор замовлення;

– UsedId – унікальний ідентифікатор користувача, який замовив путівку;

– ProductId – унікальний ідентифікатор путівки, яка була замовлена;

– PaymentId – унікальний ідентифікатор оплати путівки користувачем;

– Price – загальна ціна замовлення;

– Quantity – загальна кількість замовлених путівок;

– DateOfPurchase – дата оплати;

– CreatedAt – дата створення замовлення;

Таким чином було отримано достатньо велику кількість полів, проте всі ці поля є необхідними для коректної роботи розроблюваного веб-сайту.

Варто зазначити, що в деяких колекціях повторюються поля UserId та ProductId. Це зроблено для того, щоб за допомогою цих полів пов’язати з певними колекціями користувача та інформацію про тури відповідно. Також є доцільним створення допоміжної таблиці Role, адже досить зручно використовувати в таблиці User лише поле RoleId і за допомогою нього при потребі дістати всю інформацію про сутність Role.

Отже, було створено логічну модель бази даних, і керуючись нею можемо перейти до проектування фізичної моделі.

2.3 Проектування серверної частини веб-додатка

Серверна частина є важливою складовою частиною програмного продукту, адже відповідає за більшість функціоналу системи. Сервер повинен пов’язати між собою базу даних та те, що буде показуватись користувачу на екрані, тобто клієнтську частину.

Для створення запитів та управління ними повинен використовуватись прикладний програмний інтерфейс RESTAPI.

RESTAPI (RepresentationalStateTransfer) – передача стану представлення. Цей патерн надає змогу компонентам клієнта та сервера взаємодіяти в мережі.

Щоб надіслати на сервер інформацію, або отримати її з сервера на клієнті використовують запити чотирьох типів:

– GET – для отримання даних;

– POST – для створення нових даних;

– PUT – для редагування існуючих даних;

– DELETE – для видалення даних.

Інформація з цих запитів передається на сервер у тілі запиту, а сервер, в свою робить та зберігає зміни у базі даних. Формат даних, які отримуються, та відповідей, які надсилаються визначений за стандартними протоколами HTTP та HTTPS.

Важливим є формат, у якому подаються ці запити та відповіді. Ці формати є визначеними завдяки стандартному протоколу HTTP.

На сьогодні виділяють два основні та найпоширеніші підходи до створення серверів: монолітний та мікросервісний. Розглянемо переваги та недоліки кожного з них.

Монолітна архітектура найбільш поширена при розробці веб-додатків. В основі цієї архітектури лежить єдиний пакет, в якому знаходяться модулі, які взаємодіють між собою, утворюючи багаторівневу архітектуру. Структура монолітної архітектури складається з чотирьох рівнів, які представлено на рисунку 2.1.

Рисунок 2.1 - Монолітна архітектура

Рівень бізнес-логіки – створює правила та спосіб організації моделей, які описують сутності до предметної області. Як правило, це є ієрархічно поєднані між собою класи, або за допомогою композиції, агрегації чи інших відношень. Така модель повинна чітко описувати сутності відповідно їх аналогів у реальному житті.

Рівень доступу до сховища даних – цей рівень відповідає за доступ до даних. На ньому відбувається робота з базою даних за допомогою, які називають CRUD [10].

– C - Create – додавання даних;

– R - Read – отримання даних;

– U – Update – оновлення даних;

– D - Delete – видалення даних.

Рівень представлення – це інтерфейс користувача, який надсилає запити та отримує відповіді, після чого виводить інформацію на екран у зручному для користувача вигляді [10].

Переваги монолітної архітектури:

– просте розгортання: оскільки це повна структура, її можна розвернути безпосередньо на сервері;

– єдина технологія: так як монолітний додаток є єдиним, набагато швидше і легше проводиться тестування;

– низька вартість найму: один програміст може завершити весь процес бізнес-інтерфейсу з базою даних.

Недоліки монолітної архітектури:

– повільний запуск системи: процес містить всю бізнес-логіку та задіяно занадто багато модулів запуску, що призводить до тривалого періоду запуску системи;

– погана ізоляція системних помилок: низька доступність, помилки в одному модулі можуть викликати простої всієї системи;

– тривалий цикл усунення неполадок в інтерактивному режимі: для усунення несправностей в Інтернеті потрібне комплексне оновлення всієї системи програм.

Мікросервісна архітектура є іншим підходом до розробки сервера. Архітектурний стиль мікросервісів – це спосіб перетворення однієї програми на групу невеликих сервісів. Кожна служба працює у власному процесі. Для зв'язку між сервісами використовується полегшений механізм зв'язку (зазвичай із використанням API ресурсів HTTP). Сервіси побудовані навколо бізнес-можливостей та можуть бути розгорнуті незалежно за допомогою повністю автоматизованого механізму розгортання. Ці послуги мають мінімальне централізоване керування. Сервіси можуть розроблятися різними мовами та використовувати різні технології зберігання даних. Структура мікросервісної архітектури показана на рисунку 2.2.

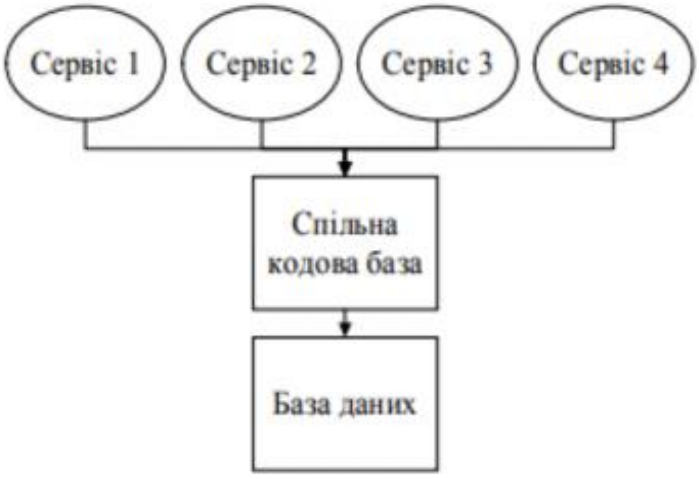


Рисунок 2.2 - Мікросервісна архітектура

Переваги мікросервісної архітектури:

− легко розробляти та підтримувати: мікросервіс буде орієнтований тільки на конкретну бізнес-функцію, тому його логіка зрозуміла, обсяг коду невеликий, досить просто розробити та підтримувати один мікросервіс, а вся програма побудована декількома мікросервісами, тому вся програма також підтримується в контрольованому стані;

− швидкий запуск: одна мікрослужба містить менше коду, тому вона запускатиметься та працюватиме швидше;

− необмежений стек технологій: можна використовувати різні фреймворки і навіть мови програмування для створення різних мікросервісів однієї системи;

− може витримувати високий рівень паралелізму: багато мікросервісів можуть виконувати будь-які дії одночасно, не впливаючи один на одного.

Недоліки мікросервісної архітектури:

− високі вимоги до експлуатації та обслуговування: більше послуг означає більше інвестицій в експлуатацію та обслуговування. У монолітній архітектурі лише один додаток має нормально працювати, тоді як у мікросервісах необхідно забезпечити нормальне функціонування десятків або навіть сотень сервісів;

− складність, властива розподіленому: використання мікросервісів для побудови розподіленої системи, мережна затримка і т. д. створять величезні проблеми;

− складність тестування: для кожного мікросервісу потрібен свій підхід тестування, а також велика кількість витрачених ресурсів для того щоб запустити всі мікросервіси одночасно;

− вартість налаштування інтерфейсу висока: мікросервіси обмінюються даними через інтерфейси. Якщо певний API мікросервісу змінено, всі мікросервіси, які використовують цей інтерфейс, мають бути переналаштовані.

Отже, можна зробити висновок, що монолітну архітектуру доцільно використовувати при розробці простих та невеликих програмних систем, які не обробляють велику кількість ресурсів. Мікросервісна архітектура повинна використовуватись для більш складних систем, де існує багато не пов’язаних між собою модулів. Тому для даного дипломного проекту було вибрано монолітну архітектуру, враховуючи його середню складність та стислі терміни розробки. Модулі даної системи будуть доволі взаємопов’язані, тому робити

декомпозицію їх на окремі сервіси є недоцільним і затратним по часу розгортання та тестування.

2.4 Проектування інтерфейсу користувача

Для того, щоб створити дизайн проекту було вирішено використати метод прототипізації. Прототипізація – це етап розробки ПЗ, на якому створюється макет або прототип програмного засобу, що дає змогу візуалізувати інтерфейс користувача на ранніх етапах розробки, для того щоб продемонструвати майбутнє ПЗ замовнику. Прототипування дає змогу уникнути більшості ризиків на етапі проектування.

Спочатку необхідно описати верхню частину сторінки, яка називається «хедер». Він повинен в собі містити логотип, та кнопка для повернення на головну сторінку, а також кнопки авторизації та реєстрації (рисунок 2.3).

Рисунок 2.3 - Хедер для не авторизованого користувача

У авторизованого користувача в хедері кнопки зміняться. Замість кнопок авторизації та реєстрації буде кнопка для виходу з облікового запису, а також з’являться кнопки переходу на корзину та на список попередніх замовлень (рисунок 2.4).

Рисунок 2.4 - Хедер для не авторизованого користувача

В адміністратора хедер буде схожий з хедером користувача, тільки в нього з’явиться ще кнопка, для переходу на сторінку додавання нової подорожі (рисунок 2.5).

Рисунок 2.5 - Хедер для не авторизованого користувача

Далі необхідно провести проектування екрану розроблюваного програмного продукту.

Розпочнемо з головної сторінки з каталогом турів. Тут будуть розміщені наступні елементи:

– заголовок;

– випадаючі списки для фільтрів по континентах та ціні у вигляді елементів, на які можна натиснути, щоб застосувати фільтр;

– поле для пошуку;

– каталог турів, кожен тур, в якому буде складатись з зображення, назви та ціни.

Макет головної сторінки представлено на рисунку 2.6.

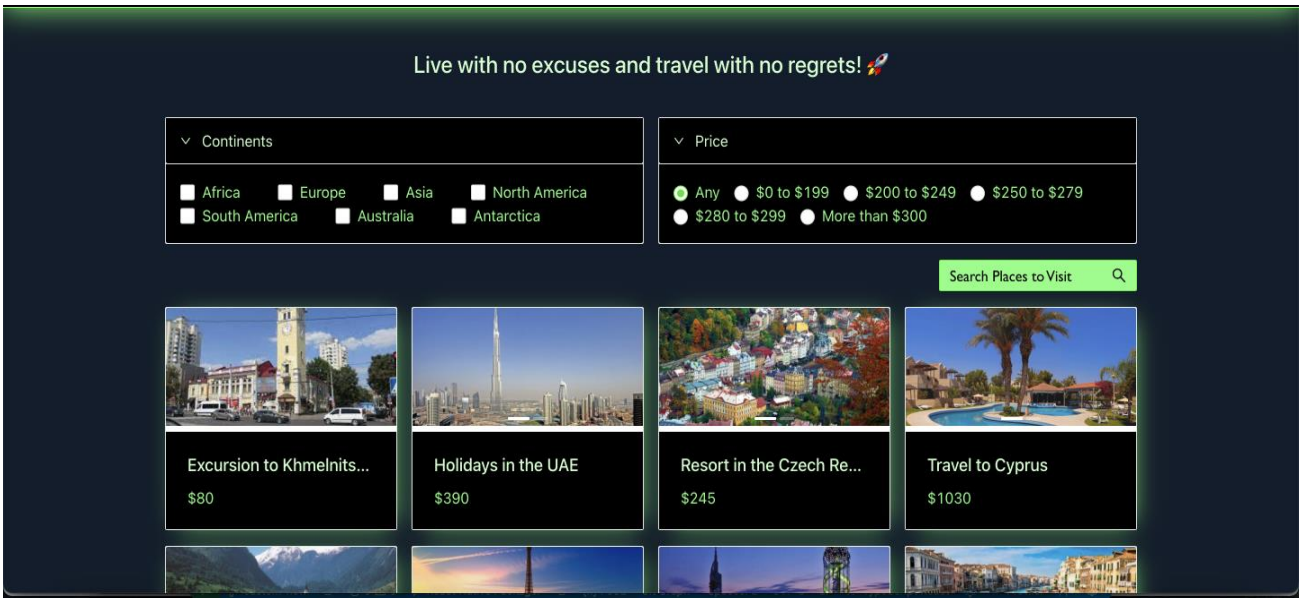


Рисунок 2.6 - Макет головної сторінки

Наступним кроком буде проектування макету для детального перегляду інформації про тур. В лівій частині буде розташований слайдер для перегляду фотографій. В правій частині буде виводитись таблиця з детальною інформацією про тур: описом, ціною та кількість замовлень цієї подорожі. Внизу знаходитиметься кнопка додавання туру в корзину. Макет сторінки з детальної інформацією представлений на рисунку 2.7.

Рисунок 2.7 - Макет сторінки з детальною інформацією про тур.

Далі змоделюємо сторінку корзини. Тут має знаходитись заголовок, таблиця з інформацією про вибраний тур: фотографією, кількістю путівок та ціною, а також присутня кнопка для того, щоб була змога видалити тур з корзини. Внизу розташована кнопка для оплати замовлення. Макет сторінки корзини зображений на рисунку 2.8.

Рисунок 2.8 - Макет сторінки корзини.

Наступними проведено моделювання макетів сторінок для перегляду замовлень для користувача та адміністратора (рисунки 2.9 та 2.10).

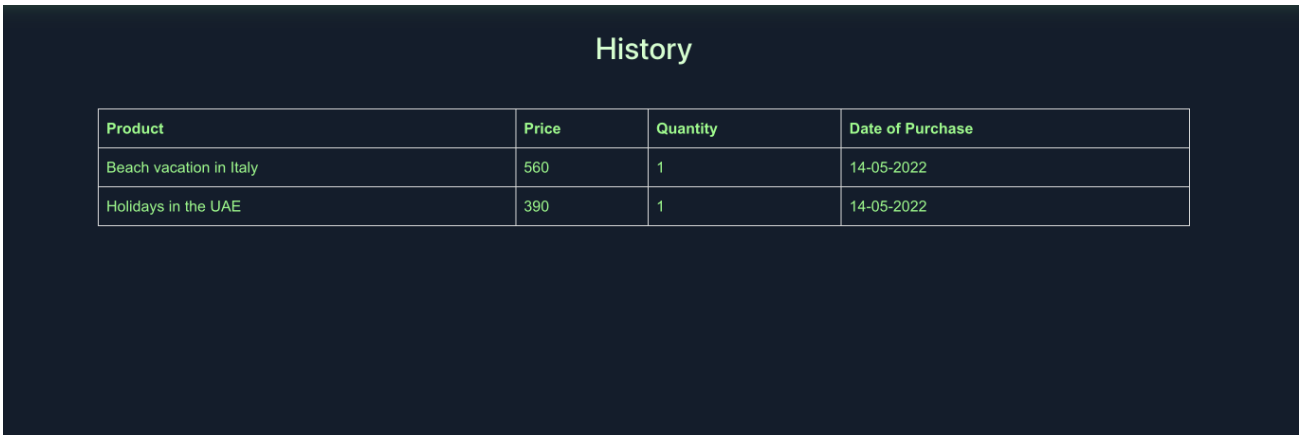


Рисунок 2.9 - Макет сторінки замовлень для звичайного користувача.

Вони складаються з заголовка та таблиці, в якій знаходиться інформація щодо замовлень: назва туру, ціна, кількість путівок та дата замовлення. В користувача лише одна таблиця – його власні замовлення. В адміністратора ж їх багато – для кожного користувача окрема таблиця замовлень.

Рисунок 2.10 - Макет сторінки замовлень для адміністратора.

Також в адміністратора повинен бути екран для додавання нового туру. На цьому екрані будуть розміщені поля для заповнення інформації про товар: назва, опис, ціна та континент. При цьому континент має вибиратись з випадаючого списку. Тут має знаходитися сама кнопка для додавання. Інтерфейс сторінки додавання туру можна побачити на рисунку 2.11.

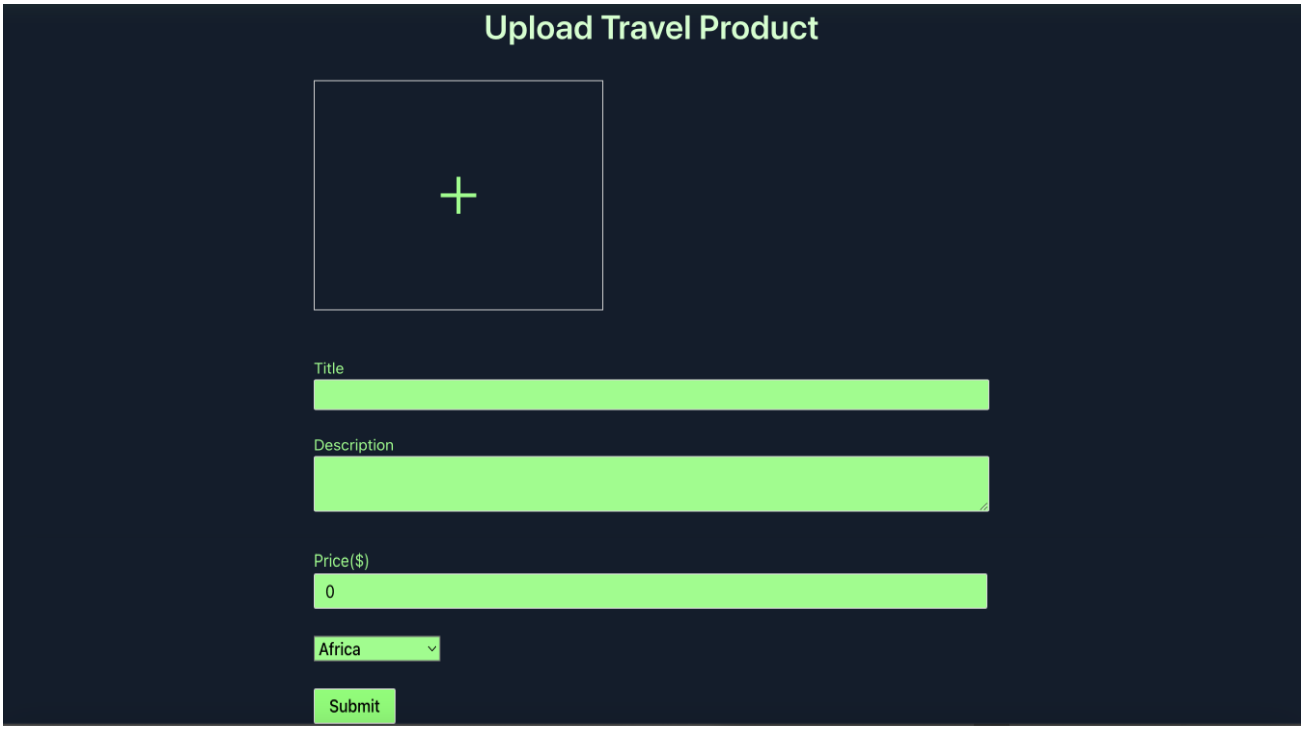


Рисунок 2.11 - Макет сторінки для додавання турів

Останніми спроектуємо сторінки для реєстрації та авторизації, які представлено на рисунку 2.12 та 2.13. На них розташовані поля для вводу даних та кнопка, для відправки цих даних.

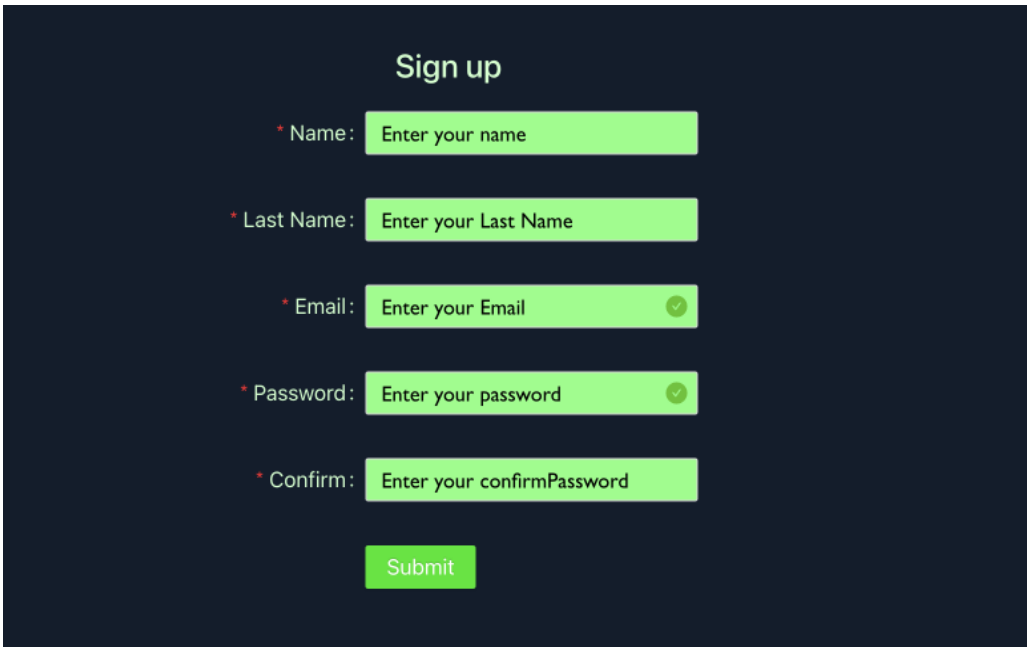


Рисунок 2.12 - Макет сторінки реєстрації.

На сторінці авторизації також є посилання для переходу на сторінку реєстрації та опція для запам’ятовування системою даного користувача.

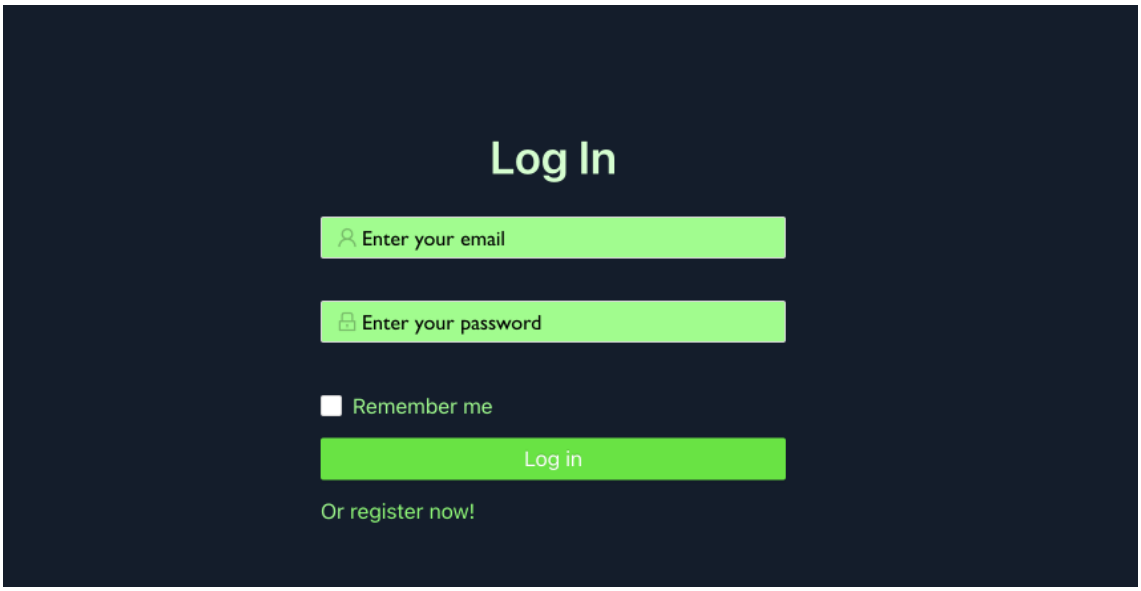


Рисунок 2.13 - Макет сторінки авторизації.

2.5 Аналіз та вибір технологій і методів реалізації веб-додатка

Сучасний веб-застосунок вимагає сучасних методів розробки програмного забезпечення. Основною мовою програмування для розробки вибрана об'єктно-орієнтована мова програмування JavaScript. Ця мова розробки програмного забезпечення є найпоширенішою в світі. JavaScript-код можна знайти майже на усіх платформах з відкритим кодом. Будь який веб-сайт створений з використанням декількох технологій. Це можуть бути фреймворки, бібліотеки, плагіни, патерни, тощо. Такі поєднання технологій називаються стеками. Для розробки даного дипломного проекту вибрано MERN стек.

Стек MERN – це стек, створений на основі мови програмування JavaScript та розроблений для спрощення процесу розробки. MERN включає чотири компоненти з відкритим вихідним кодом: MongoDB, Express, React і Node.js. Ці компоненти забезпечують комплексне середовище для розробників. Він є одним з найпопулярніших стеків, які нині використовуються розробниками у провідних IT-компаніях. MERN набагато полегшує процес взаємодії клієнтською частини з серверною та базою даних, а також робить цей процес більш контрольованим.

Розглянемо всі компоненти, з яких складається стек MERN.

1) MongoDB - кросплатформна база даних документів.

MongoDB – це NoSQL (нереляційна) документно-орієнтована база даних. У той час як традиційні реляційні бази даних мають типовий дизайн схеми, що базується на стовпцях і таблицях, MongoDB не вимагає схем. Дані зберігаються в гнучких документах за допомогою мови запитів на основі JSON (JavaScript Object Notation). Зміст, розмір та кількість полів у документах можуть відрізнятися від одного до іншого. Це означає, що структура даних змінюватиметься з часом. MongoDB відомий своєю гнучкістю та простотою масштабування. дозволяє своїм користувачам створювати схеми, бази даних, таблиці тощо. Документи, які можна ідентифікувати за первинним ключем, становлять основний блок MongoDB. Після встановлення MongoDB

користувачі також можуть використовувати оболонку Mongo. Оболонка Mongo надає інтерфейс JavaScript, за допомогою якого користувачі можуть взаємодіяти та виконувати операції (наприклад: запити, оновлення записів, видалення записів).

2) Express: базова платформа веб-додатків.

Express є структурою веб-програми для Node.js, іншого компонента MERN. Замість того, щоб писати повний код веб-сервера вручну на Node.js, розробники використовують Express для спрощення завдання написання коду сервера. Немає необхідності повторювати той самий код знову і знову, як це було б з HTTP-модулем Node.js.

Платформа Express призначена для створення надійних веб-додатків та API. Він відомий своєю швидкою швидкістю та мінімалістською структурою, а багато функцій доступні у вигляді плагінів, що робить код коротшим і легшим для написання.

Express має наступні переваги:

- асинхронність та однопотоковість;
- ефективність, швидкість і масштабованість;
- має найбільшу спільноту для Node.js;
- Express сприяє повторному використанню коду за допомогою вбудованого маршрутизатора;
- надійний API.

3) React: бібліотека JavaScript для створення інтерфейсів користувача.

React використовується для розробки односторінкових і мобільних додатків через його здатність обробляти дані, які швидко змінюються. React дозволяє користувачам кодувати на JavaScript і створювати компоненти інтерфейсу користувача. React створює структуру віртуальної сторінки (Virtual DOM), змінює лише ті елементи, на стан яких впливають дії користувача, а потім накладає зміни на реальну структуру сторінки.

React має низький бар’єр входу та велике ком’юніті, яке створило багато готових до використання компонентів, і до нього можна звернутися за допомогою, якщо щось піде не так.

React надає розробнику інструмент, який називається JSX – означає JavaScript XML. Це розширення JavaScript, яке дозволяє поєднувати логіку та верстку інтерфейсу користувача в одному й тому ж самому компоненті. Це значно спрощує написання компонентів React. Компоненти в React – це складові блоки інтерфейсу користувача, в якому кожен компонент має логіку та вносить свій внесок у загальний інтерфейс користувача. Ці компоненти також сприяють повторному використанню коду та полегшують розуміння веб-програми в цілому.

Такі функції, як Virtual DOM та JSX роблять React набагато швидшим за інші бібліотеки та фреймворки.

4) Node.JS: Середовище виконання JS.

Середовище Node.js створене за допомогою движка JavaScript V8 від Google надає середовище JavaScript, яке дозволяє користувачеві запускати свій код на сервері (за межами браузера). Менеджер пакетів вузлів, тобто npm, дозволяє користувачеві вибирати з тисячі безкоштовних пакетів (модулів вузлів) для завантаження. Цей компонент є найбільш важливим для стека, оскільки він значною мірою включає в себе технологію Websockets.

Npm-компонент, дозволяє виконувати завдання паралельно (або багатопотоково). Щоб дозволити серверу обробляти декілька паралельних запитів, Node js створює цикл подій, у якому на обробку кожного запиту відводиться час; якщо запитів немає, час не відводиться.

В результаті однопотоковий JavaScript стає умовно багатопоточним, і програма може витримувати великі навантаження та легко розширюватися в майбутньому.

У клієнтській частині не можна використовувати лише React, адже це бібліотека тільки для створення зовнішнього інтерфейсу користувача.

Потрібен також інструмент для керування станом веб-додатку. Для цього зазвичай використовують технологію Redux.

Redux є контейнером для управління станом програми. Redux не прив'язаний безпосередньо до React і може використовуватися з іншими js-бібліотеками та фреймворками.

Ключові моменти Redux:

− сховище (store): зберігає стан програми. У кожному додатку може бути лише одне сховище.

− дії (actions): деякий набір інформації, що походить від додатка до сховища і який вказує, що саме потрібно зробити. Для передачі інформації у сховища викликається метод dispatch().

− творці дій (actioncreators): функції, що створюють дію;

− reducer: функція (або кілька функцій), яка отримує дію і відповідно до цієї дії змінює стан сховища.

Працює це таким чином: З тобто компонентів представлення React ми посилаємо дію, цю дію отримує функція reducer, який відповідно до типу дії оновлює стан сховища. Потім компоненти React отримують і застосовують оновлений стан зі сховища.

При реалізації клієнтської частини, є також обов’язковим створення розмітки та стилів. Для створення розмітки даного проекту безальтренативною є мова розмітки HTML, а для привабливого зовнішнього інтерфейсу користувача – мова стилів CSS.

HTML (Hyper Text Markup Language) – це код, який використовується для структурування та відображення веб-сторінки та її вмісту. Наприклад, вміст може бути структурований всередині множини параграфів, маркованих списків або з використанням зображень і таблиць даних.

HTML не є мовою програмування; це мова розмітки яка використовується, щоб повідомляти вашому браузеру, як відображати веб-сторінки, які відвідує користувач. HTML складається з ряду елементів, які використовуються, щоб вкладати або обертати різні частини контенту, щоб

змусити контент відображатись або діяти певним чином. Огороджувальні теги мають змогу зробити слова або зображення посиланням на щось, зробити слова курсивом, зробити шрифт більше або менше і так далі.

Елементи HTML можуть також мати атрибути. Атрибути містять додаткову інформацію про елементи. Наприклад існує атрибут class, який дозволяє дати елементу ідентифікаційне ім'я, яке може використовуватися пізніше, щоб звертатися до елемента з інформацією про стиль та інші речі.

Атрибут завжди повинен мати:

− пробіл між ним та ім'ям елемента (або попереднім атрибутом, якщо елемент вже має один або кілька атрибутів);

− ім'я атрибута, за яким слідує знак рівності;

− значення атрибута, укладене з двох сторін у лапки.

CSS (Cascading Style Sheets) – це код, який ви використовується для стилізації вашої веб-сторінки. На сьогодні, майже всі сайти використовують CSS. Як і HTML, CSS насправді є мовою програмування. Це – мова таблиці стилів. Це означає, що ця технологія дозволяє використовувати стилі вибірково до елементів у документах HTML.

Практично будь-які "зовнішні прояви" сайту створюються за допомогою CSS. Мова CSS швидко стала стандартом у веб-розробці, тому що дозволяє швидко змінити візуальне оформлення сайту, не вдаючись до використання більш складних мов програмування.

Поступово розробники відмовилися від стандартного оформлення сторінок за допомогою таблиць (це вбудований у HTML синтаксис, який не вимагає додаткових інструментів для оформлення), тому що це ускладнювало структуру сторінок. HTML-файли сильно розростались, ними було складно керувати, а способів оформити хоча б текст більше не ставало. Але всіх урятував CSS.

Щоправда, навіть CSS може помітно збільшити розміри HTML і зробити його важко читабельним, якщо застосовувати каскадні таблиці прямо в основному документі. Тому за правило взято оформлення CSS-розмітки в

окремому файлі, який підключається до умовного index.html, а той вже підтягує потрібні стилі.

Для розміщення елементів на «полотні» сайту використовуються дві основні методики:

− Flex – дозволяє автоматично розподілити об'єкти в блоці за рахунок створення блоків-обгорток з властивістю flex;

− Grid – дозволяє відмовитись від обгорток та розміщувати об'єкти по сітці.

Обидва методи дозволяють створювати сайти, елементи яких завжди займають коректну позицію і адаптуються під роздільну здатність екрану.

В даному розділі було проведено проектування системи та обрано оптимальний варіант його архітектури відповідно до предметної області. На основі специфікації вимог спроектовано користувацький інтерфейс. Спроектовано структуру бази даних. Розроблена архітектура системи та її складові компоненти.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Розробка бази даних

Node.js дає змогу легко взаємодіяти з базами даних MongoDB. Для цього знадобиться драйвер MongoDB Node.js, щоб підключитися до бази даних і виконати запити. Його можна встановити його за допомогою наступної команди:

```
npm install mongodb
```

Далі, необхідно створити кластер MongoDB Atlas і завантажити зразки даних.

Після виконаних дій, знадобиться база даних MongoDB. Найпростіший спосіб розпочати роботу з MongoDB – це використовувати Atlas – зручний хмарний сервіс для керування базами даних MongoDB.

Необхідно перейти до Atlas і створити новий кластер. Кластер – це набір вузлів, де будуть зберігатися копії бази даних. Після створення потрібно завантажити зразок даних.

Модуль MongoDB містить в собі бібліотеку MongoClient. Можна використовувати екземпляр MongoClient для підключення до кластера та доступу до бази даних у цьому кластері.

Приклад підключення до кластера:

```
const {MongoClient} = require('mongodb');
```

Далі необхідно створити асинхронну функцію, де буде підключення до кластера MongoDB. Всередині цієї функції треба створити константу для URI підключення. URI підключення – це рядок, який розробник копіює з Atlas в серверну частину проекту. Коли вставляється рядок підключення, необхідно не забути оновити <ім’я користувача> і <пароль>, на відповідні облікові дані користувача. Рядок підключення виглядає таким чином:

```
consturi = "mongodb+srv://<username>:<password>@<your-cluster-url>/test?retryWrites=true&w=majority";
```

Потім настає черга для створення екземпляру MongoClient.

```
const client = new MongoClient(uri);
```

Тепер можна використовувати MongoClient для підключення до кластера. Необхідно використовувати ключове слово await під час виклику client.connect(), щоб вказати, що потрібно заблокувати подальше виконання, доки ця операція не завершиться.

3.2 Розробка програмних модулів

Структура проекту зображена на рисунку 3.1:

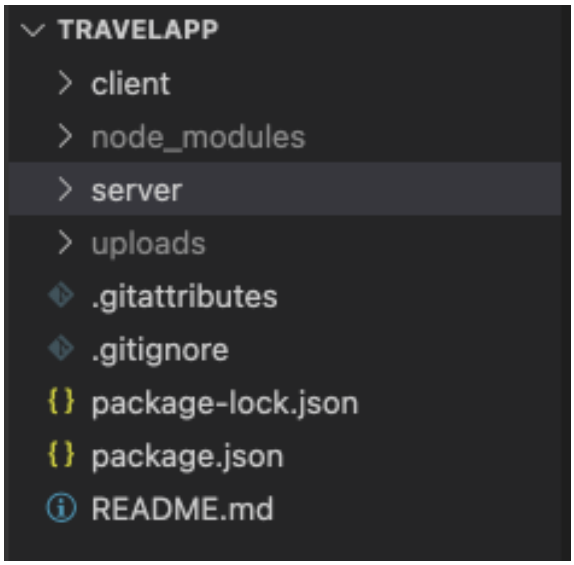


Рисунок 3.1 - Структура проекту

Проект містить дві основні папки:

– client – містить файли клієнтської частини проекту;

– server – містить серверної клієнтської частини проекту;

Розглянемо папку server (рисунок 3.2).

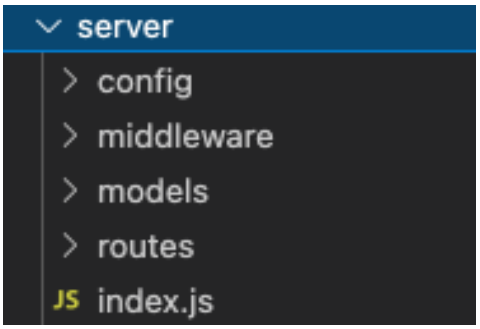


Рисунок 3.2 - Структура папки server

Вона складається з головного файлу index.js, в якому відбувається підключення до бази даних та запускається сервер на порту 5001.

Папка config містить ключі та посилання, такі як рядок підключення до бази даних та .env файли.

Папка middleware містить файл auth.js, в якому знаходиться допоміжна функція, за допомогою якої визначають користувача за його токеном.

Папка models описує структуру об’єкта бази даних. В проекті є три моделі: User, ProducttaPayment. Їх представлено на рисунку 3.3.

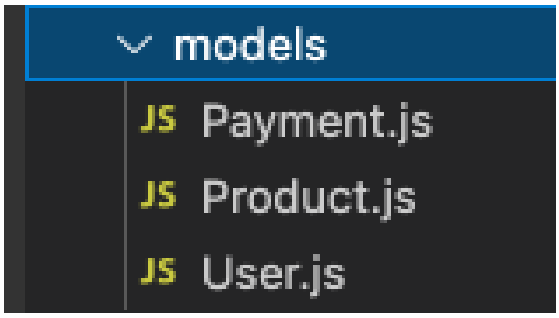


Рисунок 3.3 - Моделі проекту

Приклад моделі:

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;

const productSchema = mongoose.Schema({
    writer: {
        type: Schema.Types.ObjectId,
        ref: 'User'
    },
```

```
        title: {
            type: String,
            maxlength: 50
        },
        description: {
            type: String
        },
        price: {
            type: Number,
            default: 0
        },
        images: {
            type: Array,
            default: []
        },
        continents: {
            type: Number,
            default: 1
        },
        sold: {
            type: Number,
            maxlength: 100,
            default: 0
        },
        views: {
            type: Number,
            default: 0
        }
    }, { timestamps: true })

    productSchema.index({
        title:'text',
        description: 'text',
    }, {
        weights: {
            name: 5,
            description: 1,
        }
    })

    const Product = mongoose.model('Product', productSchema);

    module.exports = { Product }
```

Наступною розглянемо папку routes (рисунок 3.4). В ній знаходяться функції, які дозволяють з певним маршрутом обробити запит, який надходить з клієнтської частини, за потреби зберегти дані та надіслати відповідь у коректному форматі.

Рисунок 3.4 - Папка routes

Приклад такої функції:

```
router.get("/getHistory", auth, (req, res) => {
  User.findOne({ _id: req.user._id }, (err, doc) => {
    let history = doc.history;
    if (err) return res.status(400).send(err);
    return res.status(200).json({ success: true, history });
  });
});
```

Далі перейдемо до папки client (рисунок 3.5)

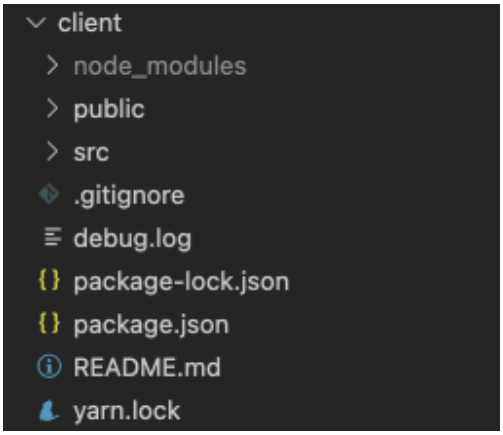


Рисунок 3.5 - Мапка client

Більшість файлів тут слугують утилітами, а 99% вихідного коду збережено в папці src, яка розміщена на рисунку 3.6.

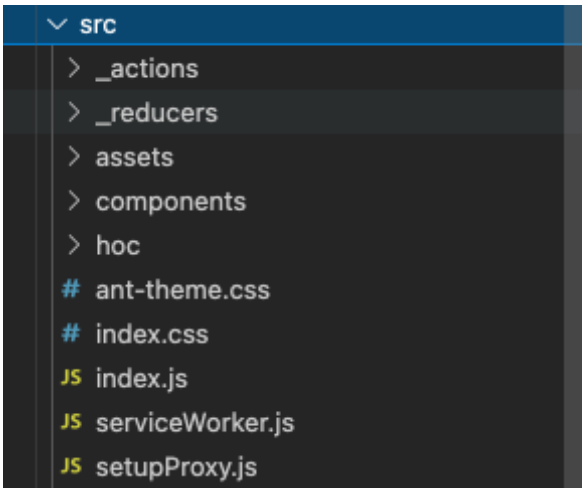


Рисунок 3.6 - Папка src

Файл index.js є точкою входу додатку. В ньому знаходиться кореневий компонент проекту (root), який є обгорткою для всіх інших компонентів:

```
ReactDOM.render(
<Provider
    store={createStoreWithMiddleware(
      Reducer,
      window.__REDUX_DEVTOOLS_EXTENSION__ &&
        window.__REDUX_DEVTOOLS_EXTENSION__()
    )}
>
<BrowserRouter>
<App />
</BrowserRouter>
</Provider>,
  document.getElementById("root")
);
```

В файлах .css знаходяться стилі проекту, які покращують зовнішній інтерфейс користувача. Приклад стилів:

```
.app {
  flex-direction: column;
  display: flex;
  justify-content: center;
  align-items: center;
  height: 100vh;
}

input.error {
  border-color: red;
}
```

В папці actions знаходяться файли, в яких описані дії, які може виконувати користувач. В папці reducers знаходяться редюсери – функції, які здійснюють управління станом даних проекту залежно від дії користувача. Структура папок actions та reducers зображена на рисунку 3.7.

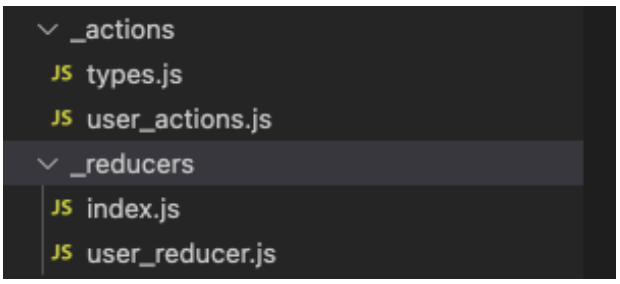


Рисунок 3.7 - Папки actions та reducers

Приклад action:

```
export function loginUser(dataToSubmit) {
  const request = axios
    .post(`${USER_SERVER}/login`, dataToSubmit)
    .then((response) => response.data);

  return {
    type: LOGIN_USER,
    payload: request,
  };
}
```

Приклад редюсера:

```
export default function (state = {}, action) {
  switch (action.type) {
    case REGISTER_USER:
      return { ...state, register: action.payload };
    case LOGIN_USER:
      return { ...state, loginSuccess: action.payload };

    default:
      return state;
  }
}
```

Папка assets містить зображення, використанні на сторінках веб-додатку (рисунок 3.8).

Рисунок 3.8 - Папка assets

Папка components складається з компонентів проекту (рисунок 3.9). Компоненти проекту створені за допомогою технології ReactJSX, яка дозволяє в одному й тому ж файлі поєднати логіку та розмітку.

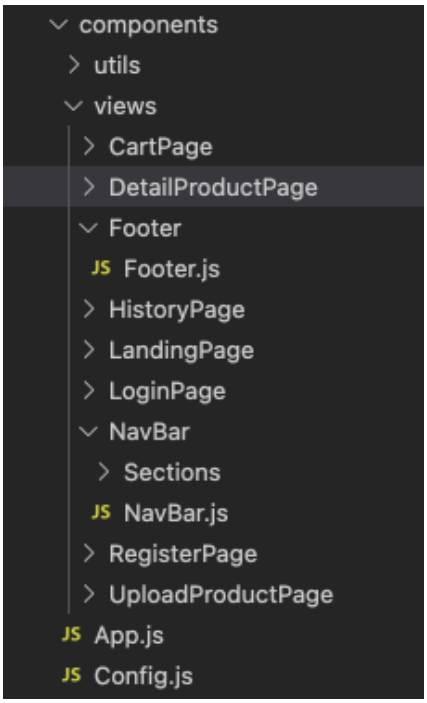


Рисунок 3.9 - Структура папки components

Приклад компоненту:

```
import React, { useEffect, useState } from 'react'
import Axios from 'axios'
import { Row, Col } from 'antd';

import ProductImage from './Sections/ProductImage';
import ProductInfo from './Sections/ProductInfo';
import { addToCart } from '../../../_actions/user_actions';
import { useDispatch } from 'react-redux';


function DetailProductPage(props) {
    const dispatch = useDispatch();
    const productId = props.match.params.productId
    const [Product, setProduct] = useState([])

    useEffect(() => {

Axios.get(`/api/product/products_by_id?id=${productId}&type=single`)
            .then(response => {
                setProduct(response.data[0])
            })
```

```
    }, [])

    const addToCartHandler = (productId) => {
        dispatch(addToCart(productId))
    }

    return (
<div className="postPage" style={{ width: '100%', padding: '3rem 4rem'
}}>

<div style={{ display: 'flex', justifyContent: 'center' }}>
<h1>{Product.title}</h1>
</div>

<br />

<Row gutter={[16, 16]} >
<Col lg={12} xs={24}>
<ProductImage detail={Product} />
</Col>
<Col lg={12} xs={24}>
<ProductInfo
                        addToCart={addToCartHandler}
                        detail={Product} />
</Col>
</Row>
</div>
    )
}

export default DetailProductPage
```

3.3 Керівництво користувача

Коли користувач заходить на розроблений сайт, першою сторінкою, яка йому відкриється, буде головна сторінка веб-застосунку (рисунок 3.1). На цій сторінці знаходиться каталог подорожей, кожна подорож в якому містить наступні дані:

– назва подорожі;

– фотографії, які самостійно переключаються за допомогою слайдера;

– ціна подорожі.

Над каталогом розташовані фільтри. За допомогою них можна відфільтрувати тури з каталогу за континентом та ціною. Увімкнені користувачем фільтри одразу застосуються. Також можна для зручності

скористатись пошуком, який також спрацьовує кожного разу, коли користувач починає щось вводити.

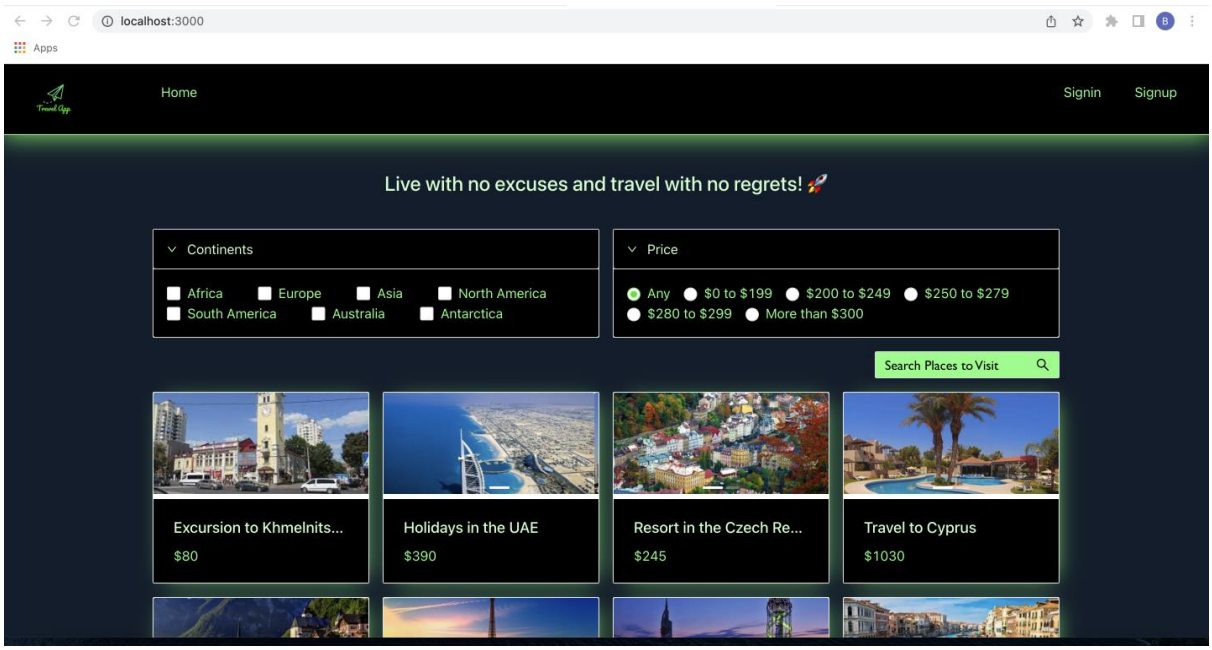


Рисунок 3.10 - Головна сторінка

Під каталогом знаходиться кнопка «LoadMore» (рисунок 3.2). Вона призначена для пагінації. За замовчуванням користувач бачить тільки 8 елементів каталогу. Натиснувши на цю кнопку можна буде побачити більше турів.

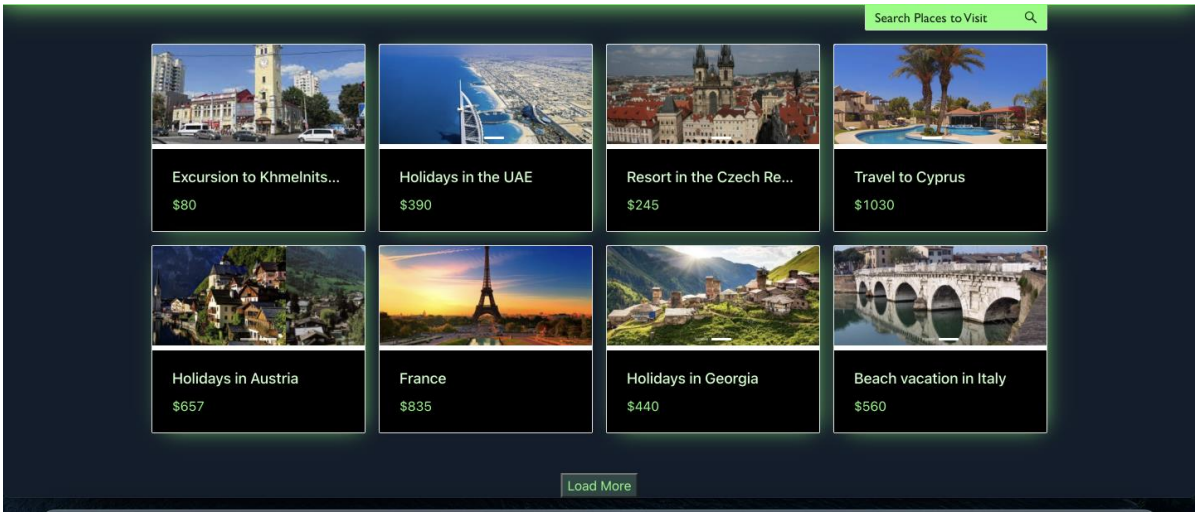


Рисунок 3.11 - Пагінація на головній сторінці

Натиснувши на фото будь-якої з подорожей у списку, можна потрапити на сторінку детальної інформації про тур, яка зображена на рисунку 3.12. На цій сторінці користувачу показана назва туру, галерея всіх фотографій, детальний опис туру, а також ціна і кількість людей, що замовили цей тур раніше. Внизу сторінки є кнопка «AddtoCart». За допомогою неї можна додати тур в корзину, щоб у подальшому його замовити.

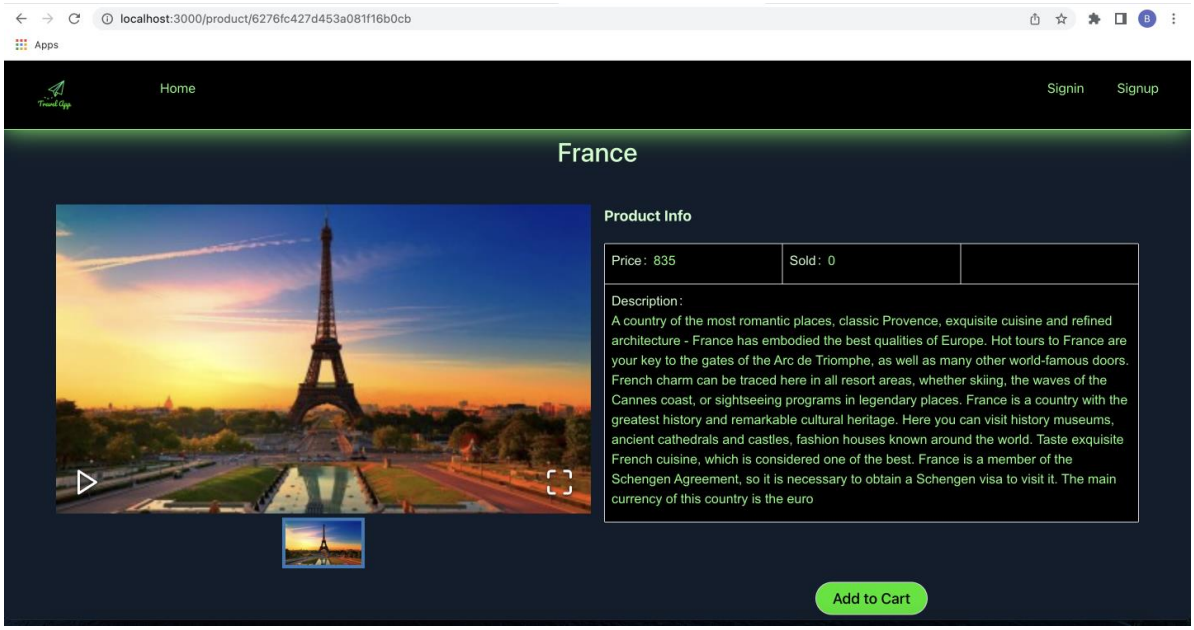


Рисунок 3.12 - Сторінка детальної інформації

Щоб додати подорож до корзини необхідна авторизація. Якщо ж у користувача немає власного облікового запису, він мусить зареєструватись. Для цього потрібно натиснути кнопку «Signup» на верхній панелі. Після цього відкриється сторінка реєстрації (рисунок 3.13). Для того щоб зареєструватись, користувач має ввести наступну інформацію:

– ім’я;
– прізвище;
– електронна адреса;
– пароль;
– повторення паролю;

Після цього потрібно натиснути кнопку «Submit». Якщо всі поля заповнені вірно, процес реєстрації пройде успішно. У разі невдачі – спрацює валідація і система видасть повідомлення про помилку.

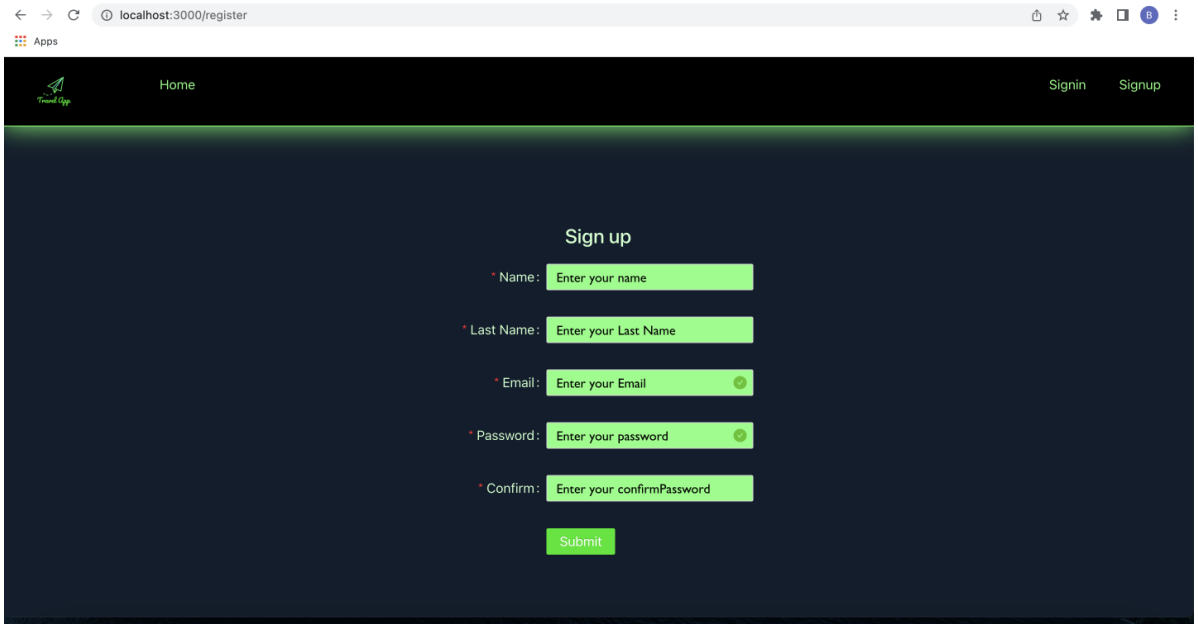


Рисунок 3.13 - Сторінка реєстрації

Зареєструвавшись, користувач може авторизуватись (рисунок 3.14). Для цього потрібно у верхній панелі вибрати «Signin». Користувач потрапить на сторінку логіну, де в нього теж буде форма з полями: електронна адреса та пароль.

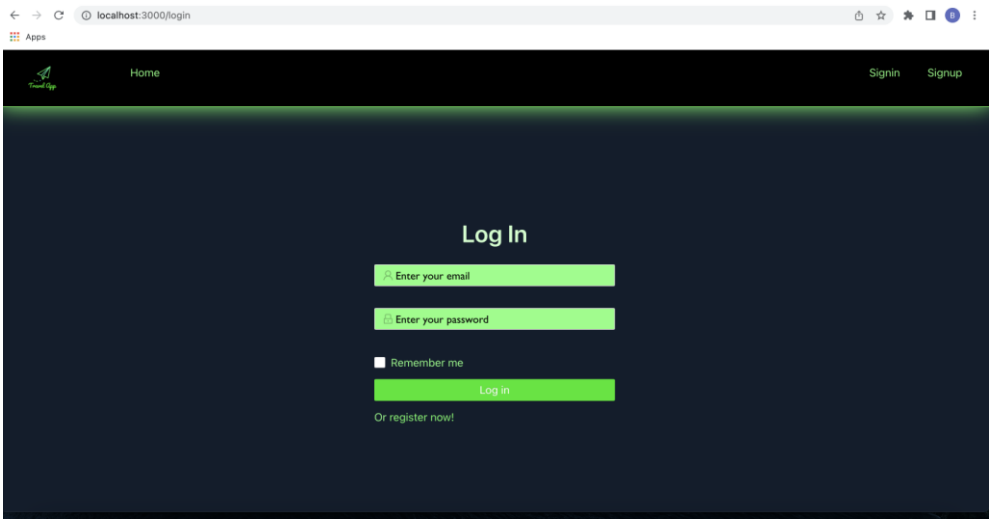


Рисунок 3.14 - Сторінка авторизації

Заповнивши ці поля своїми даними, користувачу треба натиснути кнопку «Login». При бажанні можна поставити галочку у полі «Rememberme». У такому разі, при наступному вході в обліковий запис, система автоматично заповнить поле «Email» відповідними даними користувача.

Авторизувавшись на сайті, користувач отримає додаткові функції. У верхній панелі, з'являться кнопки для переходу на корзину та історію своїх замовлень. Сторінка корзини зображена на рисунку 3.15. На ній користувач може переглянути тури, які він додав у корзину, у вигляді таблиці, а також видалити певний тур з корзини, якщо передумає його замовляти. Також тут з кнопка оплати за допомогою платіжної системи PayPal. Після оплати замовлення буде вважатись оформленим.

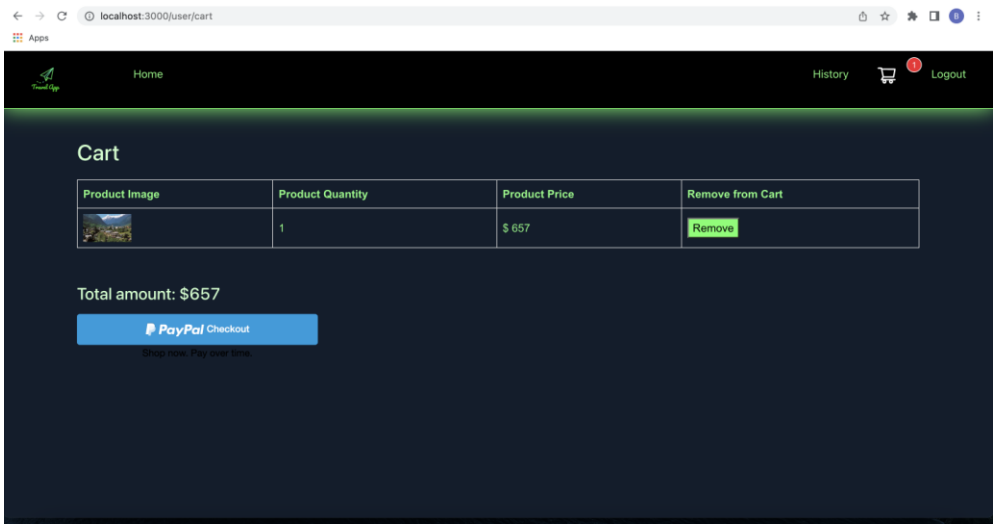


Рисунок 3.15 - Сторінка корзини користувача

Після оплати замовлення потрапляє у список замовлень. Його також можна переглянути у вигляді таблиці (рисунок 3.16). У ній показана така інформація, як назва туру, дата замовлення, ціна та кількість путівок.

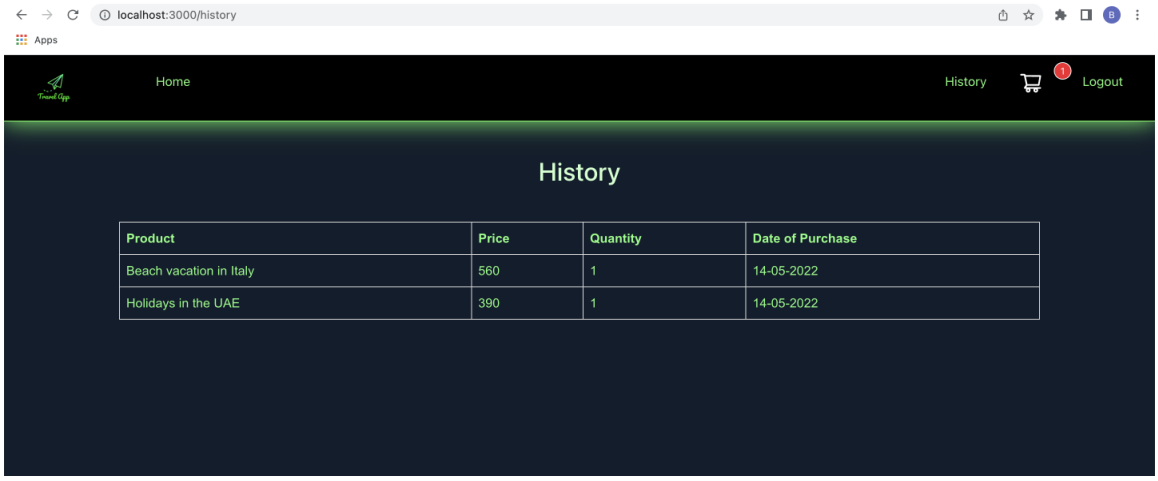


Рисунок 3.16 - Сторінка історії замовлень користувача

Якщо користувач авторизується як адміністратор, в нього буде ще більше додаткових можливостей. На верхній панелі з’являться опції додавання нового туру та перегляду всіх замовлень. Щоб додати новий тур, потрібно натиснути кнопку Upload. На сторінці яка відкриється (рисунок 3.17), потрібно ввести інформацію про тур, а саме:

– назву;

– опис;

– ціну (у доларах);

– вибрати континент;

– завантажити фотографію.

Після цього потрібно натиснути кнопку «Submit» і доданий тур з’явиться у каталогах всіх користувачів.

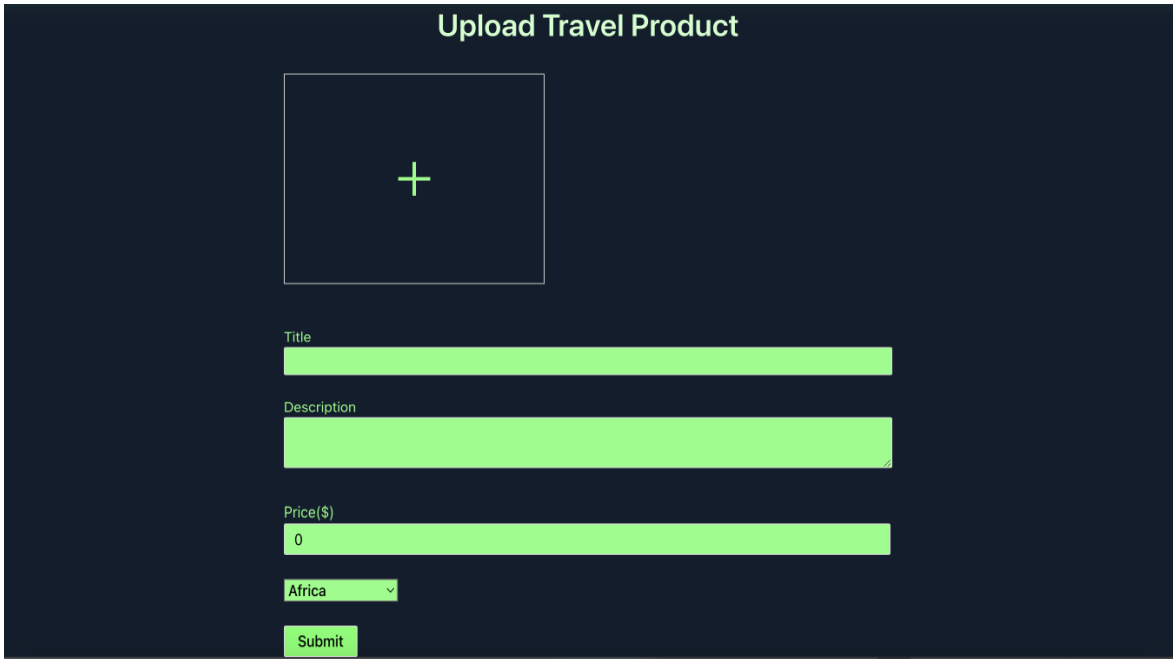


Рисунок 3.17 – Сторінка додавання туру

Вибравши на верхній панелі пункт History, адміністратору відкриється список всіх замовлень, згрупований по користувачах. В кожному замовленні є інформація про назву туру, дату замовлення, ціну та кількість путівок. Таблиця історії замовлень всіх користувачів зображена на рисунку 3.18.

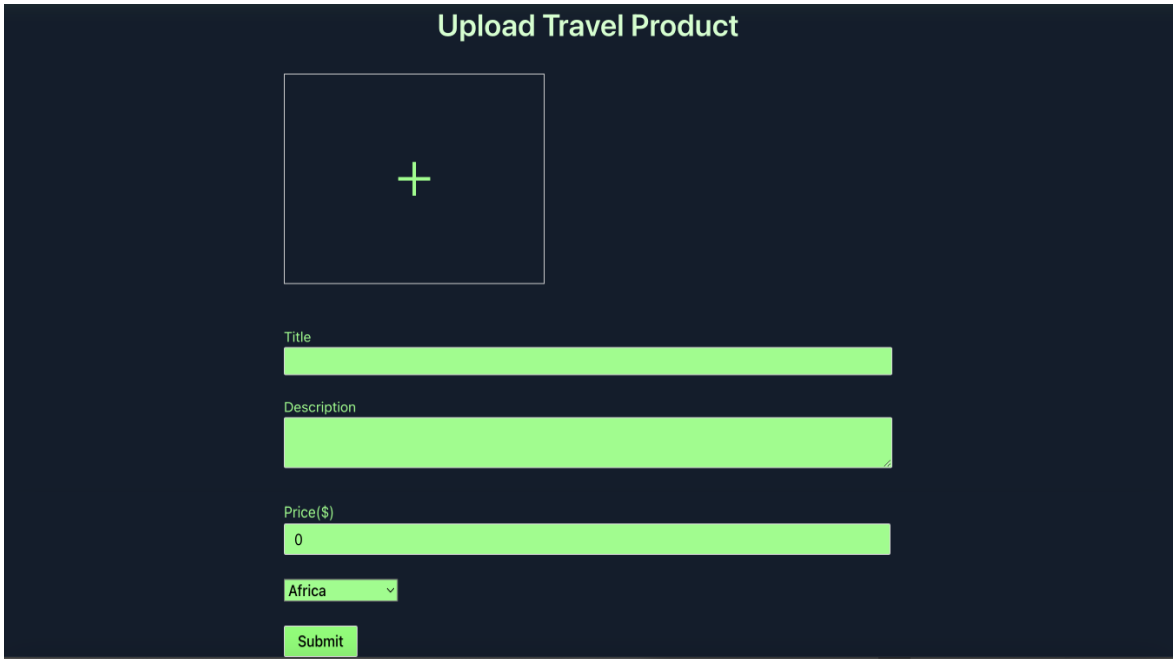


Рисунок 3.18 – Сторінка замовлень всіх користувачів

3.4 Технічні характеристики інтернет-платформи

Для роботи платформи без збоїв, потрібно, щоб пристрій, на якому вона запускається задовольняв наступні мінімальні вимоги:

– 512 Мб внутрішньої пам’яті;

– 2 Гб оперативної пам’яті;

– 2-ядерний процесор;

– доступ до мережі «Інтернет» зі швидкістю мінімум5Мбіт/с.

3.5 Розгортання та встановлення системи

Для того, щоб розгорнути даний проект, потрібно мати встановленими на своєму комп’ютері такі засоби, як Node.JSта MongoDB.

Відкривши папку з проектом, необхідно запустити два термінали.

Для запуску клієнтської частини потрібно виконати такі команди:

– cd client;

– npm install;

– npm start;

Щоб запустити серверну частину, треба ввести таку послідовність команд:

– cd server;

– npm install;

– node index.js;

В цьому розділі було описано процес створення бази даних та процес підключення її до веб-додатку. Також було проведено декомпозицію системи на модулі та розглянуто детально кожен модуль. Визначено технічні характеристики інтернет-платформи. Розглянуто алгоритм інсталяції та експлуатації розробленого програмного продукту.

4 ТЕСТУВАННЯ ВЕБ-ДОДАТКА

4.1 Аналіз методів тестування веб-додатка

Завершити програмний проект недостатньо, також потрібно провести його тестування. Ця дія полягає в перевірці програмного забезпечення, чи працює розроблений програмний продукт належним чином і чи відповідає він вимогам технічного завдання.

Основними цілями тестування програмного забезпечення є усунення помилок і покращення різних аспектів програмного забезпечення, таких як продуктивність, користувацький досвід, безпека тощо. Велика кількість тестів може покращити загальну якість програмного забезпечення, що призведе до більшого задоволення клієнтів. Запуск програми без ретельного тестування спричинить багато дрібних або великих проблем для користувачів.

Тестування програмного забезпечення, як правило, класифікується на дві основні великі категорії: функціональне тестування та нефункціональне тестування.

Функціональне тестування.

Функціональне тестування включає тестування функціональних аспектів програмного додатка. Коли виконуються функціональні тести, є потреба перевірити весь функціонал додатку. Необхідно побачити, будуть отримані бажані результати чи ні.

Існує кілька типів функціонального тестування, наприклад:

- юніт-тестування;
- інтеграційне тестування;
- наскрізне тестування;
- регресійне тестування;
- тестування білої скриньки;
- тестування чорної скриньки;
- тестування інтерфейсу;

Функціональні тести виконуються як вручну, так і за допомогою засобів автоматизації. Для такого типу тестування провести ручне тестування легко, але за потреби слід використовувати інструменти.

Нефункціональне тестування.

Нефункціональне тестування – це тестування нефункціональних аспектів програми, таких як продуктивність, надійність, зручність використання, безпека тощо. Нефункціональні тести проводяться після функціональних проб.

За допомогою нефункціонального тестування можна значно покращити якість робробленого програмного забезпечення. Нефункціональне тестування дозволяє відшліфувати програмне забезпечення.

Нефункціональні тести, як правило, не виконуються вручну. Насправді, виконати такі тести вручну важко. Тому ці тести, зазвичай, виконуються за допомогою інструментів.

Існує кілька типів нефункціонального тестування, наприклад:

- тестування продуктивності;
- тестування безпеки;
- тестування навантаженням;
- тестування на сумісність;
- тестування юзабіліті;
- тестування масштабованості;
- стрес-тестування;
- тестування на відповідність;
- тестування ефективності;
- тестування аварійного відновлення;
- тестування локалізації.

4.2 Розробка тестів

Проведемо функціональне тестування веб-додатка. Завданням функціонального тестування є перевірка відповідності програми своїм специфікаціям. Тестові сценарії показано у таблиці 4.1.

Таблиця 4.1 - Тестовий випадок №1.

Тестовий випадок призначений для перевірки коректності програми для повного циклу роботи в частині адміністратора.			
Steps (Кроки/дії)		Expected Results (Очікувані результати)	Actual results (Реальні результати):
1.	Відкрити веб-сайт	Повинна з’явитися головна сторінка.	З’явилася головна сторінка з каталогом турів.
2.	Натиснути кнопку «Signin»	Повинна з’явитися сторінка авторизації.	З’явилася сторінка з формою для авторизації.
3.	Авторизація	Після введення даних та їх відправки повинен відбутись перехід на головну сторінку, вигляд меню має змінитись	Відбувається перехід на головну сторінку зі списом турів. У меню з’явились пункти «History» та «Upload» та «Logout».
4.	Додавання туру	Повинна з’явитися сторінка із полями для введення нової подорожі. Заповнивши поля, натиснувши на кнопку, новий об’єкт повинен з’явитись на головній сторінці в каталозі та в базі даних.	З’явилась форма введення даних. На основі введених даних, в базу даних записався відповідний об’єкт та відображається на головній активності.

Таблиця 4.1 - Тестовий випадок №1 (продовження)

5.	Натиснути кнопку «History»	Повинна з’явитися сторінка зі списком усіх замовлених подорожей.	З’явилася активність зі списком усіх замовлень. Замовлені подорожі згруповані по користувачах.
6.	Вихід з облікового запису	Повинні зникнути пункти меню «History» та «Upload» та з’явитись «Signin» та «Signup».	Користувач вийшов з облікового запису. Більше не має можливість додавати подорожі та переглядати історію замовлень, натомість може зареєструватись або авторизуватись.

Таблиця 4.2 - Тестовий випадок №2.

Тестовий випадок призначений для перевірки коректності програми для повного циклу роботи в частині звичайного користувача.			
Steps (Кроки/дії)		Expected Results (Очікувані результати)	Actual results (Реальні результати):
1.	Відкрити веб-сайт	Повинна з’явитися головна сторінка.	З’явилася головна сторінка з каталогом турів.
2.	Натиснути кнопку «Signup»	Повинна з’явитися сторінка реєстрації.	З’явилася сторінка з формою для реєстрації.
3.	Реєстрація.	Після введення даних та натиснення кнопки «Submit» повинен створитись новий	Відбувається перехід на сторінку авторизації. Обліковий запис користувача створено та

Таблиця 4.2 - Тестовий випадок №2 (продовження)

		обліковий запис користувача та відбутись перехід на сторінку авторизації.	відбувається перехід на сторінку авторизації.
4.	Натиснути кнопку «Signin»	Повинна з’явитися сторінка авторизації.	З’явилася сторінка з формою для авторизації.
5.	Авторизація	Після введення даних та натиснення кнопки «Submit» повинен відбутись перехід на головну активність.	Відбувається перехід на головну сторінку додатку. У меню з’явились пункти «Checkout» та «History» та «Logout» .
6.	Натиснути на вибраний в каталозі тур.	Повинна з’явитися сторінка з детальною інформацією про вибраний тур приладдя.	Відкривається сторінка з детальним описом про вибране тур. Також можна переглянути фотографії.
7.	Натиснути кнопку «Add to Cart».	Вибрана путівка має додатись до корзини.	Відкривається сторінка з полями, які необхідно заповнити для оформлення замовлення.
8.	Натиснути на значок корзини у меню	Повинен відбутись перехід на сторінку за списком турів, доданих до корзини.	Відкривається сторінка з списком турів, які знаходяться у корзині. Список зображений у вигляді таблиці.

Таблиця 4.2 - Тестовий випадок №2 (продовження)

9.	Натиснути на кнопку «Remove» навпроти вибраної путівки у корзині	Вибраний тур повинен зникнути з корзини.	Видалена путівка зникає з корзини поточного користувача.
10.	Натиснути кнопку «Замовити».	Повинно з’явитись вікно платіжної системи PayPal. Після успішної оплати, замовлення повинно з’явитись у списку замовлень клієнта.	З’явилось вікно PayPal. Оплата проведена. Замовлена путівка з’явилась у списку замовлень клієнта.
11.	Натиснути кнопку «History»	Повинна з’явитися сторінка зі списком замовлених подорожей користувача.	З’явилася активність зі списком усіх замовлень поточного користувача.
12.	Вихід з облікового запису	Повинні зникнути пункти меню «History» та значок та з’явитись «Signin» та «Signup».	Користувач вийшов з облікового запису. Більше не має можливості додавати тури в корзину та переглядати історію замовлень, натомість може зареєструватись або авторизуватись.

Модульне тестування – це вид функціонального тестування, при якому перевіряється на працездатність необхідних модулів додатку. Модуль – це найменша частина програмного забезпечення, яка може бути протестована. Для даного програмного забезпечення необхідно перевірити усі методи дій та дані які вони повертають.

Для модульного тестування використаємо бібліотеки Jest та Enzyme. Наступний фрагмент коду реалізує тестування головної сторінки, перевіряючи чи відрендерилась вона в браузері без помилок:

```
importReactfrom'react';
importReactDOMfrom'react-dom';
importAppfrom'./App';


it('renders without crashing',()=>{
constdiv=document.createElement('div');
ReactDOM.render(<App/>,div);
ReactDOM.unmountComponentAtNode(div);
});
```

Результат виконання тесту представлено на рисунку 4.1:

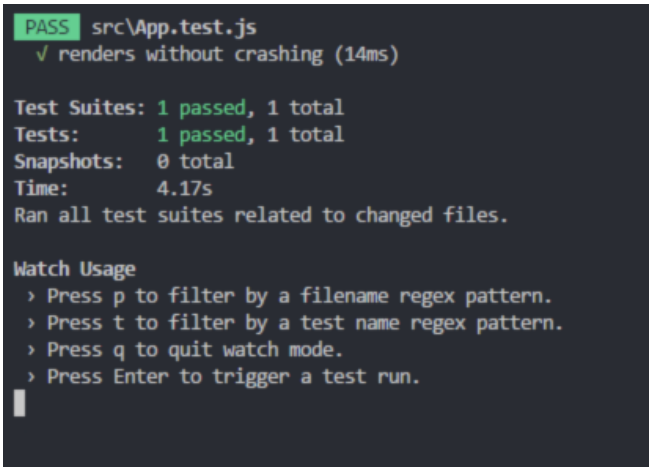


Рисунок 4.1 - Результат тестування рендеру головної сторінки

Таким чином, у цьому розділі було розглянуто та описано основні методи здійснення тестування веб-додатків.

Як видно з результатів модульного тестування, все працює саме так, як і передбачалося.

Згідно проведеного функціонального тестування можна зробити висновок, що програмний продукт пройшов тестування на 100%, результат відповідає

очікуваним вимогам, поставленим у технічному завданні і проект готовий до експлуатації.

ВИСНОВКИ

Під час виконання дипломного проекту було проаналізовано предметну область, доведена актуальність розробки даного проекту, визначені причини, через які можуть виникати проблеми у людей, які не користуються веб-додатками у сфері надання послуг, таких як туризм та бізнес-подорожі, а також досліджені існуючі рішення у даній сфері. На підставі аналізу схожих програмних засобів, було складено список їх головних переваг та недоліків та визначено функціональні і нефункціональні вимоги до майбутнього програмного продукту. На основі цих вимог було сформовано технічне завдання і варіанти використання додатку.

Наступним етапом стало проектування програмного забезпечення. На цьому кроці було проведено аналіз недоліків та переваг поширених архітектур, які зазвичай використовуються для розробки веб-сайтів. Таким чином, встановлено. що для даної програмної системи найкраще підходить клієнт-серверна архітектура з використання патерну MVC (Model–View - Controller).

Після цього було спроектовано архітектуру бази даних, визначено, які саме мають бути поля, таблиці та зв’язки між таблицями.

Потім, спираючись на спроектовану архітектуру програмного продукту та бази даних, прийнято рішення про вибір технології та інструментів для розробки системи, які найкраще підходять для даного типу архітектури, а саме: мова програмування JavaScript, фреймворки React та Node.js інереляційна база даних MongoDB. Основним фактором при виборі стала швидкість роботи, яку забезпечують ці технології.

Наступним кроком стало розбиття програмної системи на модулі і проведене проектування цих модулів, визначено, як вони будуть взамодіяти один з одним. Також на цьому етапів було спроектовано інтерфейс розроблюваного веб-додатку.

На основі спроектованої архітектури та інтерфейсу було створено базу даних, а також серверну та клієнтську частини системи і механізм для їхньої взаємодії.

Завершальним етапом стало тестування, під час якого були виявлені та виправлені всі несправності функціоналу та зовнішнього вигляду. Для цього проводилось модульне та функціональне тестування, сформовано звіт щодо результатів тестових випробувань. Здійснені тести показали, що система виконує усі функції, які були описані у технічному завданні, має зручний для користувача зовнішній вигляд, у якому немає нічого зайвого.

Таким чином, поставлене в процесі дипломного проектування завдання є повністю виконаним. Розроблена програмна система для замовлення турів та подорожей є цілком справною та працездатною. Завдяки розробці даного проекту, було отримано та закріплено практичні та теоретичні вміння та знання всіх життєвих циклів розробки програмного забезпечення.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Л. П. Бедратюк. Дипломний проект: методичні вказівки щодо його виконання для студентів спеціальності 121 «Інженерія програмного забезпечення» / Л. П. Бедратюк, Г. І. Радельчук, Ю. В. Форкун. О. М. Яшина – Хмельницький: ХНУ, 2020. – 77с . (дата звернення – 07.05.2022).
2. Wieruch, Robin. Road to React: Your journey to master plain yet pragmatic React.js. – New York, 2020. – 226с. (дата звернення – 14.04.2022).
3. Brown, Ethan. Web Development with Node and Express – London: O'Relly Media, 2019. – 340с.(дата звернення – 01.04.2022).
4. TompsonLaura. Creating SPAWEB-Applications – London.: DiaSoft. 2016. – 672 с.(дата звернення – 28.03.2022).
5. Lindstorn, Steve. CSS Refactoring – New York: O'Relly Media, 2016. – 159с. . (дата звернення – 02.04.2022).
6. React [Електронний ресурс] – Режим доступу до ресурсу: https://uk.reactjs.org/ . (дата звернення – 10.04.2022).
7. Node.js чи Java [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/webbdev/js-9ca13173577b/ . (дата звернення – 12.04.2022).
8. Redux Fundamentals, Part 1: Redux Overview [Електронний ресурс] – Режим доступу до ресурсу: https://redux.js.org/tutorials/fundamentals/part-1-overview/ . (дата звернення – 14.04.2022).
9. Redux-Saga: Beginner Tutorial [Електронний ресурс] – Режим доступу до ресурсу: https://redux-saga.js.org/docs/introduction/BeginnerTutorial/ . (дата звернення – 15.04.2022).
10. Сервер. Створення сервера [Електронний ресурс] – Режим доступу до ресурсу: https://metanit.com/web/nodejs/3.1.php/ . (дата звернення – 17.04.2022).
11. Туроператор «Відвідай» [Електронний ресурс] – Режим доступу до ресурсу: https://vidviday.ua/. (дата звернення – 19. 04.2022).

12. Туристична компанія «Там тур» [Електронний ресурс] – Режим доступу до ресурсу: https://tamtour.com.ua/tury-po-ukraini. (дата звернення – 18.04.2022).

13. Business Visit [Електронний ресурс] – Режим доступу до ресурсу: https://businessvisit.com.ua/uk/turi/ . (дата звернення – 18.03.2022).

14. «Країна ЮА: Тури за кордон» [Електронний ресурс] – Режим доступу до ресурсу: https://kraina-ua.com/ua/tours/tours-abroad . (дата звернення – 19.04.2022).

15. Ksamil Travel [Електронний ресурс] – Режим доступу до ресурсу: https://ksamiltravel.com.ua/on-line-zamov/ . (дата звернення – 22.04.2022).

16. Модульне тестування [Електронний ресурс] / QaLight. – Режим доступу до ресурсу: https://qalight.ua/baza-znaniy/modulne-testuvannya/ . (дата звернення – 08.05.2022).

17. Уніфікована мова програмування UML [Електронний ресурс] / Портал знань. – Режим доступу до ресурсу: http://www.znannya.org/?view=uml . (дата звернення – 11.05.2022).

18. Unit Testing Tutorial: What is, Types, Tools & Test EXAMPLE [Електронний ресурс] – Режим доступу до ресурсу: https://www.guru99.com/unit-testing-guide.html

19. NOSQL – переваги та недоліки нереляційних баз даних [Електронний ресурс] / QAinfo. – Режим доступу до ресурсу: https://www.quality-assurance-group.com/nosql-perevagy-ta-nedoliky-nerelyatsijnyh-baz-danyh/ . (дата звернення – 04.05.2022).

ДОДАТОК А
(обов’язковий)

ТЕХНІЧНЕ ЗАВДАННЯ

Введення

Робота виконується в рамках проекту розробки інтернет-платформи «Агентство подорожей». Технічне завдання розроблено у відповідності до стандарту ГОСТ 19.201–78.

1 Підстава для розробки

Підставою для розробки є «Завдання на дипломний проект», затверджене завідувачем кафедри інженерії програмного забезпечення. Найменування розробки: Інтернет-платформа «Агентство подорожей».

2 Призначення розробки

2.1 Функціональне призначення

Функціональним призначенням додатку пошук та замовлення подорожей для звичайного користувача, а також облік турів та замовлень для адміністратора веб-додатку.

2.2 Експлуатаційне призначення

Програма повинна експлуатуватися на будь-яких пристроях, на яких є браузер. Кінцевим користувачем додатку може виступати будь-яка особа.

3 Вимоги до програми

3.1 Вимоги до функціональних характеристик

Для звичайного користувача:

- реєстрація;
- авторизація;
- перегляд списку турів;
- фільтрація за ціною та континентом;
- пошук за назвою та описом;
- перегляд галереї фотографій вибраного місця;
- детальний перегляд вибраної путівки;
- додавання в корзину;
- видалення з корзини;
- замовлення путівки з оплатою;
- перегляд списку своїх замовлень;

Для адміністратора:

– авторизація;

– перегляд всіх замовлень всіх користувачів;

– додавання нової подорожі з завантаженням фото;

– перегляд списку турів;

– пошук за назвою та описом;

– фільтрація за ціною та континентом;

– детальний перегляд вибраної путівки;

3.2 Вимоги до надійності

Веб-додаток повинен забезпечувати такі вимоги до надійності:

– обробляти невірні дії користувача і попереджати його про можливі наслідки;

– можливість самостійно відновлюватись у разі збою;

– можливість резервного копіювання бази даних.

3.3 Умови експлуатації та вимоги до технічних засобів

Веб-додаток повинен працювати на всіх пристроях, які мають браузер та стабільний доступ до мережі «Інтернет»: смартфонах, планшетах та комп’ютерах. Браузер може бути будь-яким: Safari, GoogleChrome, Opera, MozillaFirefox, тощо.

Для роботи платформи без збоїв, потрібно, щоб пристрій, на якому вона запускається задовольняв наступні мінімальні вимоги:

– 512 Мб внутрішньої пам’яті;

– 2 Гб оперативної пам’яті;

– 2-ядерний процесор;

– доступ до мережі «Інтернет» зі швидкістю мінімум5Мбіт/с.

3.4 Вимоги до інформаційної та програмної сумісності

При розробцівеб-додатку буде використовуватись високорівнева об’єктно-орієнтована мова JavaScript. Для розробки серверної частини використовуватиметься платформа Node.js та фреймворк Express. Для розробки клієнтської частини була використана бібліотека інтерфейсу користувача React

та бібліотека управління станом даних Redux. В якості СКБД буде використовуватись нереляційна база даних MongoDB.

3.5 Спеціальні вимоги

Програма повинна мати зручний, гарний та зрозумілий зовнішній інтерфейс користувача.

4 Вимоги до програмної документації

У момент здачі проекту замовнику надається наступний набір документів:

– текст програми;

– опис програми;

– технічне завдання;

– керівництво користувача.

5 Стадії та етапи розробки

Стадії та етапи розробки інтернет-платформи «Агентство подорожей»подані у таблиці А.1.

Таблиця А.1 – Стадії та етапи розробки проекту

Стадія розробки	Етапи робіт	Зміст робіт
1	2	3
Технічне завдання 02.01.22 – 31.01.22	Обґрунтування необхідності розробки програми	Коротка характеристика програмного забезпечення; підстава і призначення розробки; вимоги до програмної системи і документація; стадії і етапи розробки програми; порядок контролю і приймання
Ескізний проект 01.02.22 – 26.02.22	Розробка ескізного проекту	Попередня розробка структури вхідних і вихідних даних; уточнення середовища програмування; розробка і опис загальної алгоритмічної структури

Кінець таблиці А.1

1	2	3
Технічний проект 29.02.22 – 19.03.22	Розробка технічного проекту	Уточнення структури вхідних і вихідних даних; розробка докладного алгоритму; розробка структури програми
Робочий проект 20.03.22 – 15.04.22	Розробка програмного забезпечення	Реалізація програмного забезпечення; відлагодження; проведення попереднього тестування
Розробка програмної документації 16.04.22 – 22.04.22	Розробка документації до програмного забезпечення	Розробка необхідної документації, передбаченої технічним завданням
Тестування системи 23.04.22 – 30.04.22	Проведення тестування програмного забезпечення	Розробка методики тестування; проведення основних тестів; коректування програмного забезпечення
Впровадження	Підготовка і передача програми	Підготовка і розгортання програмного забезпечення
Розробка програмної документації 16.04.22 – 22.04.22	Розробка документації до програмного забезпечення	Розробка необхідної документації, передбаченої технічним завданням

6 Порядок контролю та приймання

Контроль здійснюється кінцевими користувачами системи, підключеними на етапі тестування додатку.

ДОДАТОК Б
(обов’язковий)

ДІАГРАМИ

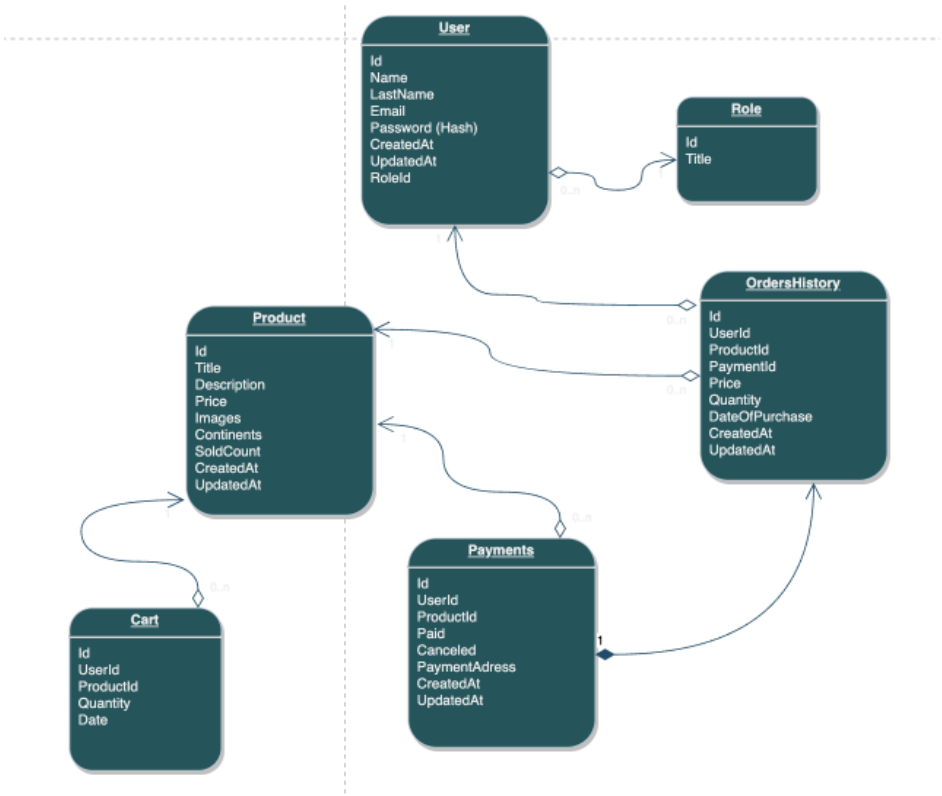


Рисунок Б.1 - Схема бази даних

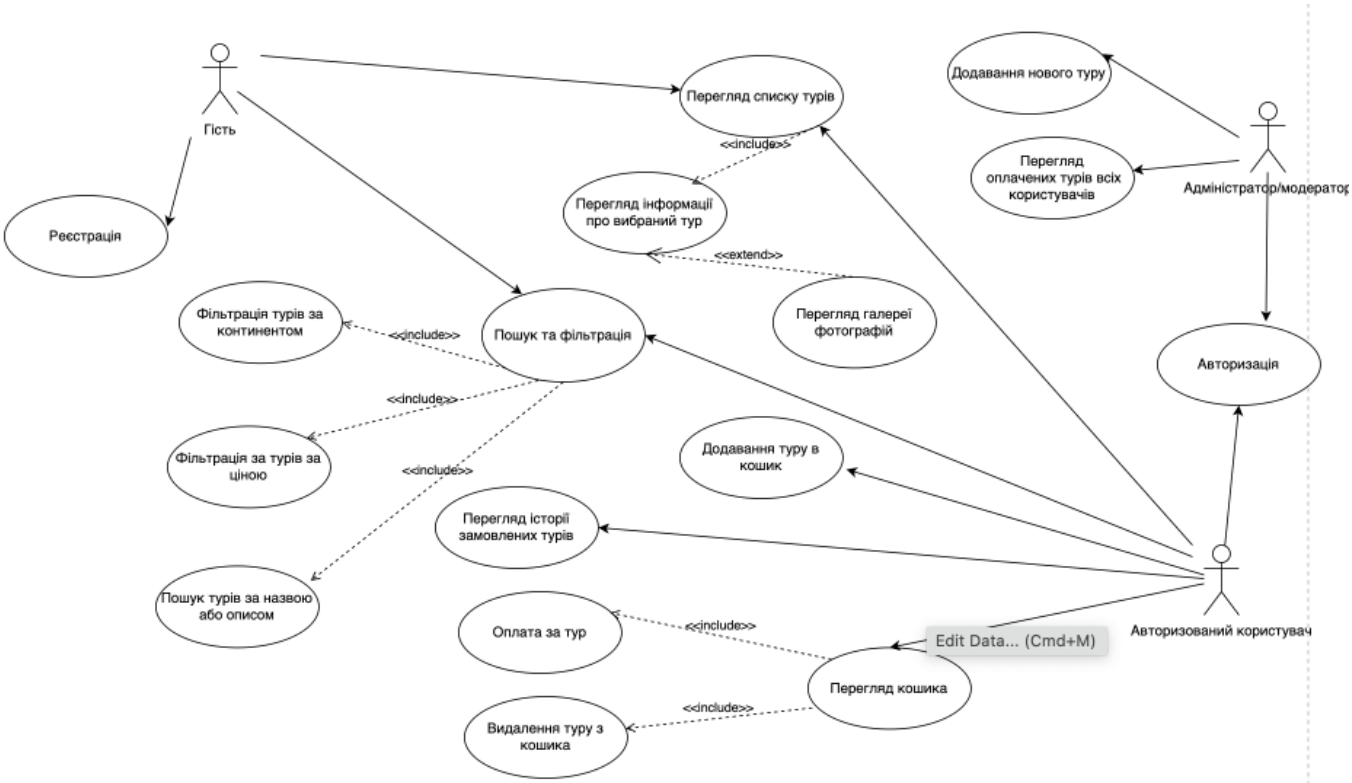


Рисунок Б.2 - Діаграма варіантів використання

ДОДАТОК В
(обов’язковий)

КОД (ЛІСТИНГ) ПРОГРАМИ

index.js

```
const express = require("express");
const app = express();
const path = require("path");
const cors = require('cors')

const bodyParser = require("body-parser");
const cookieParser = require("cookie-parser");

const config = require("./config/key");

const mongoose = require("mongoose");
const connect = mongoose.connect(config.mongoURI, { useNewUrlParser: true, useUnifiedTopology: true })
  .then(() => console.log('MongoDB Connected...'))
  .catch(err => console.log(err));

app.use(cors())

app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
app.use(cookieParser());

app.use('/api/users', require('./routes/users'));
app.use('/api/product', require('./routes/product'));

app.use('/uploads', express.static('uploads'));

if (process.env.NODE_ENV === "production") {

  app.use(express.static("client/build"));

  app.get("*", (req, res) => {
    res.sendFile(path.resolve(__dirname, "../client", "build", "index.html"));
  });
}

const port = process.env.PORT || 5001

app.listen(port, () => {
  console.log(`Server Running at ${port}`)
});
```

product.js

```
const express = require('express');
const router = express.Router();
const { Product } = require("../models/Product");
const multer = require('multer');

const { auth } = require("../middleware/auth");

var storage = multer.diskStorage({
    destination: (req, file, cb) => {
```

```
        cb(null, 'uploads/')
    },
    filename: (req, file, cb) => {
        cb(null, `${Date.now()}_${file.originalname}`)
    },
    fileFilter: (req, file, cb) => {
        const ext = path.extname(file.originalname)
        if (ext !== '.jpg' || ext !== '.png') {
            return cb(res.status(400).end('only jpg, png are allowed'), false);
        }
        cb(null, true)
    }
})

var upload = multer({ storage: storage }).single("file")

router.post("/uploadImage", auth, (req, res) => {

    upload(req, res, err => {
        if (err) {
            return res.json({ success: false, err })
        }
        return res.json({ success: true, image: res.req.file.path, fileName: res.req.file.filename })
    })

});


router.post("/uploadProduct", auth, (req, res) => {

    const product = new Product(req.body)

    product.save((err) => {
        if (err) return res.status(400).json({ success: false, err })
        return res.status(200).json({ success: true })
    })

});


router.post("/getProducts", (req, res) => {

    let order = req.body.order ? req.body.order : "desc";
    let sortBy = req.body.sortBy ? req.body.sortBy : "_id";
    let limit = req.body.limit ? parseInt(req.body.limit) : 100;
    let skip = parseInt(req.body.skip);

    let findArgs = {};
    let term = req.body.searchTerm;

    for (let key in req.body.filters) {

        if (req.body.filters[key].length > 0) {
            if (key === "price") {
                findArgs[key] = {
```

```
                $gte: req.body.filters[key][0],
                $lte: req.body.filters[key][1]
            }
        } else {
            findArgs[key] = req.body.filters[key];
        }
    }
}

console.log(findArgs)

if (term) {
    Product.find(findArgs)
        .find({ $text: { $search: `\"${term}\"` } })
        .populate("writer")
        .sort([[sortBy, order]])
        .skip(skip)
        .limit(limit)
        .exec((err, products) => {
            if (err) return res.status(400).json({ success: false, err })
            res.status(200).json({ success: true, products, postSize: products.length })
        })
} else {
    Product.find(findArgs)
        .populate("writer")
        .sort([[sortBy, order]])
        .skip(skip)
        .limit(limit)
        .exec((err, products) => {
            if (err) return res.status(400).json({ success: false, err })
            res.status(200).json({ success: true, products, postSize: products.length })
        })
}

});

router.get("/products_by_id", (req, res) => {
    let type = req.query.type
    let productIds = req.query.id

    console.log("req.query.id", req.query.id)

    if (type === "array") {
        let ids = req.query.id.split(',');
        productIds = [];
        productIds = ids.map(item => {
            return item
        })
    }

    console.log("productIds", productIds)


    Product.find({ '_id': { $in: productIds } })
        .populate('writer')
```

```
        .exec((err, product) => {
            if (err) return res.status(400).send(err)
            return res.status(200).send(product)
        })
});



module.exports = router;
```

users.js

```
const express = require("express");
const router = express.Router();
const { User } = require("../models/User");
const { Product } = require("../models/Product");
const { auth } = require("../middleware/auth");
const { Payment } = require("../models/Payment");

const async = require("async");

router.get("/auth", auth, (req, res) => {
  res.status(200).json({
    _id: req.user._id,
    isAdmin: req.user.email === "admin@admin.com" ? true : false,
    isAuth: true,
    email: req.user.email,
    name: req.user.name,
    lastname: req.user.lastname,
    role: req.user.role,
    image: req.user.image,
    cart: req.user.cart,
    history: req.user.history,
  });
});

router.post("/register", (req, res) => {
  const user = new User(req.body);

  user.save((err, doc) => {
    if (err) return res.json({ success: false, err });
    return res.status(200).json({
      success: true,
    });
  });
});

router.post("/login", (req, res) => {
  User.findOne({ email: req.body.email }, (err, user) => {
    if (!user)
      return res.json({
        loginSuccess: false,
        message: "Auth failed, email not found",
      });
```

```
    user.comparePassword(req.body.password, (err, isMatch) => {
      if (!isMatch)
        return res.json({ loginSuccess: false, message: "Wrong password" });

      user.generateToken((err, user) => {
        if (err) return res.status(400).send(err);
        res.cookie("w_authExp", user.tokenExp);
        res.cookie("w_auth", user.token).status(200).json({
          loginSuccess: true,
          userId: user._id,
        });
      });
    });
  });
});

router.get("/logout", auth, (req, res) => {
  User.findOneAndUpdate(
    { _id: req.user._id },
    { token: "", tokenExp: "" },
    (err, doc) => {
      if (err) return res.json({ success: false, err });
      return res.status(200).send({
        success: true,
      });
    }
  );
});

router.get("/addToCart", auth, (req, res) => {
  User.findOne({ _id: req.user._id }, (err, userInfo) => {
    let duplicate = false;

    console.log(userInfo);

    userInfo.cart.forEach((item) => {
      if (item.id == req.query.productId) {
        duplicate = true;
      }
    });

    if (duplicate) {
      User.findOneAndUpdate(
        { _id: req.user._id, "cart.id": req.query.productId },
        { $inc: { "cart.$.quantity": 1 } },
        { new: true },
        (err, userInfo) => {
          if (err) return res.json({ success: false, err });
          res.status(200).json(userInfo.cart);
        }
      );
    } else {
      User.findOneAndUpdate(
        { _id: req.user._id },
```

```
      {
        $push: {
          cart: {
            id: req.query.productId,
            quantity: 1,
            date: Date.now(),
          },
        },
      },
      { new: true },
      (err, userInfo) => {
        if (err) return res.json({ success: false, err });
        res.status(200).json(userInfo.cart);
      }
    );
  }
 });
});

router.get("/removeFromCart", auth, (req, res) => {
  User.findOneAndUpdate(
    { _id: req.user._id },
    {
      $pull: { cart: { id: req.query._id } },
    },
    { new: true },
    (err, userInfo) => {
      let cart = userInfo.cart;
      let array = cart.map((item) => {
        return item.id;
      });

      Product.find({ _id: { $in: array } })
        .populate("writer")
        .exec((err, cartDetail) => {
          return res.status(200).json({
            cartDetail,
            cart,
          });
        });
    }
  );
});

router.get("/userCartInfo", auth, (req, res) => {
  User.findOne({ _id: req.user._id }, (err, userInfo) => {
    let cart = userInfo.cart;
    let array = cart.map((item) => {
      return item.id;
    });

    Product.find({ _id: { $in: array } })
      .populate("writer")
      .exec((err, cartDetail) => {
        if (err) return res.status(400).send(err);
```

```
        return res.status(200).json({ success: true, cartDetail, cart });
      });
  });
});

router.post("/successBuy", auth, (req, res) => {
  let history = [];
  let transactionData = {};

  req.body.cartDetail.forEach((item) => {
    history.push({
      dateOfPurchase: Date.now(),
      name: item.title,
      id: item._id,
      price: item.price,
      quantity: item.quantity,
      paymentId: req.body.paymentData.paymentID,
    });
  });

  transactionData.user = {
    id: req.user._id,
    name: req.user.name,
    lastname: req.user.lastname,
    email: req.user.email,
  };

  transactionData.data = req.body.paymentData;
  transactionData.product = history;

  User.findOneAndUpdate(
    { _id: req.user._id },
    { $push: { history: history }, $set: { cart: [] } },
    { new: true },
    (err, user) => {
      if (err) return res.json({ success: false, err });

      const payment = new Payment(transactionData);
      payment.save((err, doc) => {
        if (err) return res.json({ success: false, err });

        let products = [];
        doc.product.forEach((item) => {
          products.push({ id: item.id, quantity: item.quantity });
        });

        async.eachSeries(
          products,
          (item, callback) => {
            Product.update(
              { _id: item.id },
              {
                $inc: {
                  sold: item.quantity,
                },
```

```
            },
            { new: false },
            callback
          );
        },
        (err) => {
          if (err) return res.json({ success: false, err });
          res.status(200).json({
            success: true,
            cart: user.cart,
            cartDetail: [],
          });
        }
      );
    });
  }
 );
});

router.get("/getHistory", auth, (req, res) => {
  User.findOne({ _id: req.user._id }, (err, doc) => {
    let history = doc.history;
    if (err) return res.status(400).send(err);
    return res.status(200).json({ success: true, history });
  });
});

router.get("/usersHistory", function (req, res) {
  User.find({}, function (err, users) {
    res.send([users]);
  });
});

module.exports = router;
```

Payment.js

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

const paymentSchema = mongoose.Schema({
    user: {
        type: Array,
        default: []
    },
    data: {
        type: Array,
        default: []
    },
    product: {
        type: Array,
        default: []
    }
```

```
}, { timestamps: true })


const Payment = mongoose.model('Payment', paymentSchema);

module.exports = { Payment }
```

Product.js

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;

const productSchema = mongoose.Schema({
    writer: {
        type: Schema.Types.ObjectId,
        ref: 'User'
    },
    title: {
        type: String,
        maxlength: 50
    },
    description: {
        type: String
    },
    price: {
        type: Number,
        default: 0
    },
    images: {
        type: Array,
        default: []
    },
    continents: {
        type: Number,
        default: 1
    },
    sold: {
        type: Number,
        maxlength: 100,
        default: 0
    },
    views: {
        type: Number,
        default: 0
    }
}, { timestamps: true })


productSchema.index({
    title:'text',
    description: 'text',
}, {
    weights: {
        name: 5,
```

```
        description: 1,
    }
})

const Product = mongoose.model('Product', productSchema);

module.exports = { Product }
```

User.js

```
const mongoose = require('mongoose');
const bcrypt = require('bcrypt');
const saltRounds = 10;
const jwt = require('jsonwebtoken');
const moment = require('moment');

const userSchema = mongoose.Schema({
    name: {
        type: String,
        maxlength: 50
    },
    email: {
        type: String,
        trim: true,
        unique: 1
    },
    password: {
        type: String,
        minglength: 5
    },
    lastname: {
        type: String,
        maxlength: 50
    },
    role: {
        type: Number,
        default: 0
    },
    cart: {
        type: Array,
        default: []
    },
    history: {
        type: Array,
        default: []
    },
    image: String,
    token: {
        type: String,
    },
    tokenExp: {
        type: Number
    }
})
```

```
userSchema.pre('save', function (next) {
    var user = this;

    if (user.isModified('password')) {
        console.log('password changed')
        bcrypt.genSalt(saltRounds, function (err, salt) {
            if (err) return next(err);

            bcrypt.hash(user.password, salt, function (err, hash) {
                if (err) return next(err);
                user.password = hash
                next()
            })
        })
    } else {
        next()
    }
});

userSchema.methods.comparePassword = function (plainPassword, cb) {
    bcrypt.compare(plainPassword, this.password, function (err, isMatch) {
        if (err) return cb(err);
        cb(null, isMatch)
    })
}

userSchema.methods.generateToken = function (cb) {
    var user = this;
    var token = jwt.sign(user._id.toHexString(), 'secret')
    var oneHour = moment().add(1, 'hour').valueOf();

    user.tokenExp = oneHour;
    user.token = token;
    user.save(function (err, user) {
        if (err) return cb(err)
        cb(null, user);
    })
}

userSchema.statics.findByToken = function (token, cb) {
    var user = this;

    jwt.verify(token, 'secret', function (err, decode) {
        user.findOne({ "_id": decode, "token": token }, function (err, user) {
            if (err) return cb(err);
            cb(null, user);
        })
    })
}

const User = mongoose.model('User', userSchema);

module.exports = { User }
```

auth.js

```
const { User } = require('../models/User');

let auth = (req, res, next) => {
  let token = req.cookies.w_auth;

  User.findByToken(token, (err, user) => {
    if (err) throw err;
    if (!user)
      return res.json({
        isAuth: false,
        error: true
      });

    req.token = token;
    req.user = user;
    next();
  });
};

module.exports = { auth };
```

App.js

```
import React, { Suspense } from "react";
import { Route, Switch } from "react-router-dom";
import Auth from "../hoc/auth";
import LandingPage from "./views/LandingPage/LandingPage.js";
import LoginPage from "./views/LoginPage/LoginPage.js";
import RegisterPage from "./views/RegisterPage/RegisterPage.js";
import NavBar from "./views/NavBar/NavBar";
// import Footer from "./views/Footer/Footer";
import UploadProductPage from "./views/UploadProductPage/UploadProductPage";
import DetailProductPage from "./views/DetailProductPage/DetailProductPage";
import CartPage from "./views/CartPage/CartPage";
import HistoryPage from "./views/HistoryPage/HistoryPage";

function App() {
  return (
<Suspense fallback={<div>Loading...</div>}>
<NavBar />
<div style={{ paddingTop: "75px", minHeight: "calc(100vh - 80px)" }}>
<Switch>
<Route exact path="/" component={Auth(LandingPage, null)} />
<Route exact path="/login" component={Auth(LoginPage, false)} />
<Route exact path="/register" component={Auth(RegisterPage, false)} />
<Route
          exact
          path="/product/upload"
          component={Auth(UploadProductPage, true)}
```

```
        />
<Route
          exact
          path="/product/:productId"
          component={Auth(DetailProductPage, null)}
        />
<Route exact path="/user/cart" component={Auth(CartPage, true)} />
<Route exact path="/history" component={Auth(HistoryPage, true)} />
</Switch>
</div>
    {/* <Footer /> */}
</Suspense>
  );
}

export default App;
```

FileUpload.js

```
import React, { useState } from 'react'
import Dropzone from 'react-dropzone';
import { Icon } from 'antd';
import Axios from 'axios';
function FileUpload(props) {

    const [Images, setImages] = useState([])

    const onDrop = (files) => {

        let formData = new FormData();
        const config = {
            header: { 'content-type': 'multipart/form-data' }
        }
        formData.append("file", files[0])
        //save the Image we chose inside the Node Server
        Axios.post('/api/product/uploadImage', formData, config)
            .then(response => {
                if (response.data.success) {

                    setImages([...Images, response.data.image])
                    props.refreshFunction([...Images, response.data.image])

                } else {
                    alert('Failed to save the Image in Server')
                }
            })
    }


    const onDelete = (image) => {
        const currentIndex = Images.indexOf(image);

        let newImages = [...Images]
        newImages.splice(currentIndex, 1)
```

```
        setImages(newImages)
        props.refreshFunction(newImages)
    }

    return (
<div style={{ display: 'flex', justifyContent: 'space-between' }}>
<Dropzone
            onDrop={onDrop}
            multiple={false}
            maxSize={800000000}
>
            {({ getRootProps, getInputProps }) => (
<div style={{
                width: '300px', height: '240px', border: '1px solid lightgray',
                display: 'flex', alignItems: 'center', justifyContent: 'center'
              }}
                {...getRootProps()}
>
                {console.log('getRootProps', { ...getRootProps() })}
                {console.log('getInputProps', { ...getInputProps() })}
<input {...getInputProps()} />
<Icon type="plus" style={{ fontSize: '3rem' }} />

</div>
            )}
</Dropzone>

<div style={{ display: 'flex', width: '350px', height: '240px', overflowX: 'scroll' }}>

            {Images.map((image, index) => (
<div onClick={() => onDelete(image)}>
<img style={{ minWidth: '300px', width: '300px', height: '240px' }} src={`http://localhost:5001/${image}`}
alt={`productImg-${index}`} />
</div>
            ))}


</div>

</div>
    )
}

export default FileUpload
```

ImageSlider.js

```
import React from "react";
import { Carousel } from "antd";
import { object } from "yup";

function ImageSlider(props) {
```

```
  return (
<div>
<Carousel autoplay>
      {props.images.map((image, index) => (
<div key={index}>
<img
            style={{
              width: "100%",
              objectFit: "cover",
              maxHeight: "120px",
            }}
            src={`http://localhost:5001/${image}`}
            alt="productImage"
          />
</div>
      ))}
</Carousel>
</div>
  );
}

export default ImageSlider;
```

user_actions.js

```
import axios from "axios";
import {
  LOGIN_USER,
  REGISTER_USER,
  AUTH_USER,
  LOGOUT_USER,
  ADD_TO_CART_USER,
  GET_CART_ITEMS_USER,
  REMOVE_CART_ITEM_USER,
  ON_SUCCESS_BUY_USER,
  GET_ALL_USERS_HISTORY,
} from "./types";
import { USER_SERVER } from "../components/Config.js";

export function registerUser(dataToSubmit) {
  const request = axios
    .post(`${USER_SERVER}/register`, dataToSubmit)
    .then((response) => response.data);

  return {
    type: REGISTER_USER,
    payload: request,
  };
}

export function loginUser(dataToSubmit) {
  const request = axios
    .post(`${USER_SERVER}/login`, dataToSubmit)
    .then((response) => response.data);
```

```
  return {
    type: LOGIN_USER,
    payload: request,
  };
}

export function auth() {
  const request = axios
    .get(`${USER_SERVER}/auth`)
    .then((response) => response.data);

  return {
    type: AUTH_USER,
    payload: request,
  };
}

export function logoutUser() {
  const request = axios
    .get(`${USER_SERVER}/logout`)
    .then((response) => response.data);

  return {
    type: LOGOUT_USER,
    payload: request,
  };
}

export function addToCart(_id) {
  const request = axios
    .get(`${USER_SERVER}/addToCart?productId=${_id}`)
    .then((response) => response.data);

  return {
    type: ADD_TO_CART_USER,
    payload: request,
  };
}

export function getCartItems(cartItems, userCart) {
  const request = axios
    .get(`/api/product/products_by_id?id=${cartItems}&type=array`)
    .then((response) => {
      //Make CartDetail inside Redux Store
      // We need to add quantity data to Product Information that come from Product Collection.

      userCart.forEach((cartItem) => {
        response.data.forEach((productDetail, i) => {
          if (cartItem.id === productDetail._id) {
            response.data[i].quantity = cartItem.quantity;
          }
        });
      });
```

```
      return response.data;
    });

  return {
    type: GET_CART_ITEMS_USER,
    payload: request,
  };
}

export function removeCartItem(id) {
  const request = axios
    .get(`/api/users/removeFromCart?_id=${id}`)
    .then((response) => {
      response.data.cart.forEach((item) => {
        response.data.cartDetail.forEach((k, i) => {
          if (item.id === k._id) {
            response.data.cartDetail[i].quantity = item.quantity;
          }
        });
      });
      return response.data;
    });

  return {
    type: REMOVE_CART_ITEM_USER,
    payload: request,
  };
}

export function onSuccessBuy(data) {
  const request = axios
    .post(`${USER_SERVER}/successBuy`, data)
    .then((response) => response.data);

  return {
    type: ON_SUCCESS_BUY_USER,
    payload: request,
  };
}

export function getAllUsersHistory(data) {
  const request = axios
    .get(`${USER_SERVER}/usersHistory`, data)
    .then((response) => response.data);

  return {
    type: GET_ALL_USERS_HISTORY,
    payload: request,
  };
}
```

user_reducer.js

```
import {
  LOGIN_USER,
  REGISTER_USER,
  AUTH_USER,
  LOGOUT_USER,
  ADD_TO_CART_USER,
  GET_CART_ITEMS_USER,
  REMOVE_CART_ITEM_USER,
  ON_SUCCESS_BUY_USER,
  GET_ALL_USERS_HISTORY,
} from "../_actions/types";

export default function (state = {}, action) {
  switch (action.type) {
    case REGISTER_USER:
      return { ...state, register: action.payload };
    case LOGIN_USER:
      return { ...state, loginSucces: action.payload };
    case AUTH_USER:
      return { ...state, userData: action.payload };
    case LOGOUT_USER:
      return { ...state };
    case ADD_TO_CART_USER:
      return {
        ...state,
        userData: {
          ...state.userData,
          cart: action.payload,
        },
      };
    case GET_CART_ITEMS_USER:
      return {
        ...state,
        cartDetail: action.payload,
      };
    case REMOVE_CART_ITEM_USER:
      return {
        ...state,
        cartDetail: action.payload.cartDetail,
        userData: {
          ...state.userData,
          cart: action.payload.cart,
        },
      };
    case ON_SUCCESS_BUY_USER:
      return {
        ...state,
        userData: {
          ...state.userData,
          cart: action.payload.cart,
        },
        cartDetail: action.payload.cartDetail,
      };

    case GET_ALL_USERS_HISTORY:
```

```
      return {
        ...state,
        usersHistoryData: action.payload,
      };

    default:
      return state;
  }
}
```

LoginPage.js

```
import React, { useState } from "react";
import { withRouter } from "react-router-dom";
import { loginUser } from "../../../_actions/user_actions";
import { Formik } from "formik";
import * as Yup from "yup";
import { Form, Icon, Input, Button, Checkbox, Typography } from "antd";
import { useDispatch } from "react-redux";

const { Title } = Typography;

function LoginPage(props) {
  const dispatch = useDispatch();
  const rememberMeChecked = localStorage.getItem("rememberMe") ? true : false;

  const [formErrorMessage, setFormErrorMessage] = useState("");
  const [rememberMe, setRememberMe] = useState(rememberMeChecked);

  const handleRememberMe = () => {
    setRememberMe(!rememberMe);
  };

  const initialEmail = localStorage.getItem("rememberMe")
    ? localStorage.getItem("rememberMe")
    : "";

  return (
<Formik
      initialValues={{
        email: initialEmail,
        password: "",
      }}
      validationSchema={Yup.object().shape({
        email: Yup.string()
          .email("Email is invalid")
          .required("Email is required"),
        password: Yup.string()
          .min(6, "Password must be at least 6 characters")
          .required("Password is required"),
      })}
      onSubmit={(values, { setSubmitting }) => {
        setTimeout(() => {
          let dataToSubmit = {
```

```
            email: values.email,
            password: values.password,
          };

          dispatch(loginUser(dataToSubmit))
            .then((response) => {
              if (response.payload.loginSuccess) {
                window.localStorage.setItem("userId", response.payload.userId);
                if (rememberMe === true) {
                  window.localStorage.setItem("rememberMe", values.email);
                } else {
                  localStorage.removeItem("rememberMe");
                }
                props.history.push("/");
              } else {
                setFormErrorMessage("Check out your Account or Password again");
              }
            })
            .catch((err) => {
              setFormErrorMessage("Check out your Account or Password again");
              setTimeout(() => {
                setFormErrorMessage("");
              }, 3000);
            });
          setSubmitting(false);
        }, 500);
      }}
>
      {(props) => {
        const {
          values,
          touched,
          errors,
          dirty,
          isSubmitting,
          handleChange,
          handleBlur,
          handleSubmit,
          handleReset,
        } = props;
        return (
<div className="app">
<Title level={2}>Log In</Title>
<form onSubmit={handleSubmit} style={{ width: "350px" }}>
<Form.Item required>
<Input
                id="email"
                prefix={
<Icon type="user" style={{ color: "rgba(0,0,0,.25)" }} />
                }
                placeholder="Enter your email"
                type="email"
                value={values.email}
                onChange={handleChange}
                onBlur={handleBlur}
```

```
              className={
                errors.email && touched.email
                  ? "text-input error"
                  : "text-input"
              }
            />
            {errors.email && touched.email && (
<div className="input-feedback">{errors.email}</div>
            )}
</Form.Item>

<Form.Item required>
<Input
              id="password"
              prefix={
<Icon type="lock" style={{ color: "rgba(0,0,0,.25)" }} />
              }
              placeholder="Enter your password"
              type="password"
              value={values.password}
              onChange={handleChange}
              onBlur={handleBlur}
              className={
                errors.password && touched.password
                  ? "text-input error"
                  : "text-input"
              }
            />
            {errors.password && touched.password && (
<div className="input-feedback">{errors.password}</div>
            )}
</Form.Item>

          {formErrorMessage && (
<label>
<p
                style={{
                  color: "#ff0000bf",
                  fontSize: "0.7rem",
                  border: "1px solid",
                  padding: "1rem",
                  borderRadius: "10px",
                }}
>
                {formErrorMessage}
</p>
</label>
          )}

<Form.Item>
<Checkbox
              id="rememberMe"
              onChange={handleRememberMe}
              checked={rememberMe}
>
```

```
                Remember me
</Checkbox>
<div>
<Button
                type="primary"
                htmlType="submit"
                className="login-form-button"
                style={{ minWidth: "100%" }}
                disabled={isSubmitting}
                onSubmit={handleSubmit}
>
                Log in
</Button>
</div>
              Or <a href="/register">register now!</a>
</Form.Item>
</form>
</div>
       );
     }}
</Formik>
  );
}

export default withRouter(LoginPage);
```

Navbar.js

```
import React, { useState } from 'react';
import LeftMenu from './Sections/LeftMenu';
import RightMenu from './Sections/RightMenu';
import { Drawer, Button, Icon } from 'antd';

import logo from '../../../assets/logo.png'
import './Sections/Navbar.css';

function NavBar() {
  const [visible, setVisible] = useState(false)

  const showDrawer = () => {
    setVisible(true)
  };

  const onClose = () => {
    setVisible(false)
  };

  return (
<nav className="menu" style={{ position: 'fixed', zIndex: 5, width: '100%' }}>
<div className="menu__logo">
<a href="/">
<img src={logo} alt="logo" style={{width: '45px'}}/>
</a>
</div>
```

```
<div className="menu__container">
<div className="menu_left">
<LeftMenu mode="horizontal" />
</div>
<div className="menu_rigth">
<RightMenu mode="horizontal" />
</div>
<Button
        className="menu__mobile-button"
        type="primary"
        onClick={showDrawer}
>
<Icon type="align-right" />
</Button>
<Drawer
        placement="right"
        className="menu_drawer"
        closable={true}
        onClose={onClose}
        visible={visible}
>
<LeftMenu mode="inline" />
<RightMenu mode="inline" />
</Drawer>
</div>
</nav>
  )
}

export default NavBar
```

ДОДАТОК Г
(обов’язковий)

ПРЕЗЕНТАЦІЙНІ МАТЕРІАЛИ

Хмельницький Національний університет

Факультет інформаційних технологій
Кафедра інженерії програмного забезпечення

Дипломний проект на тему: Інтернет-платформа «Агентство подорожей»

Виконав

студент групи ІПЗс19-1

Собко Владислав Вадимович

Керівник:

Праворська Наталія Іванівна

кандидат педагогічних наук, доцент

2022 р.

Актуальність теми

Актуальність даного проекту полягає в потребі простої і доступної для користувачів програмної системи і надійної передачі даних, що надавало б перевагу над іншими подібними рішеннями в галузі туризму, адже зараз саме інформаційні технології відіграють провідну роль в цій справі.

Споживач в таких умовах потребуває точної і повної інформації про туристичний продукт. Найкращим рішенням для цього була б проста і доступна для користувачів інтернет-платформа.

Розробивши даний програмний продукт, користувачі зможуть уникнути великої кількості витраченого часу на пошук найбільш бажаної путівки, а адміністратор витрачатиме менше часу на облік замовлень

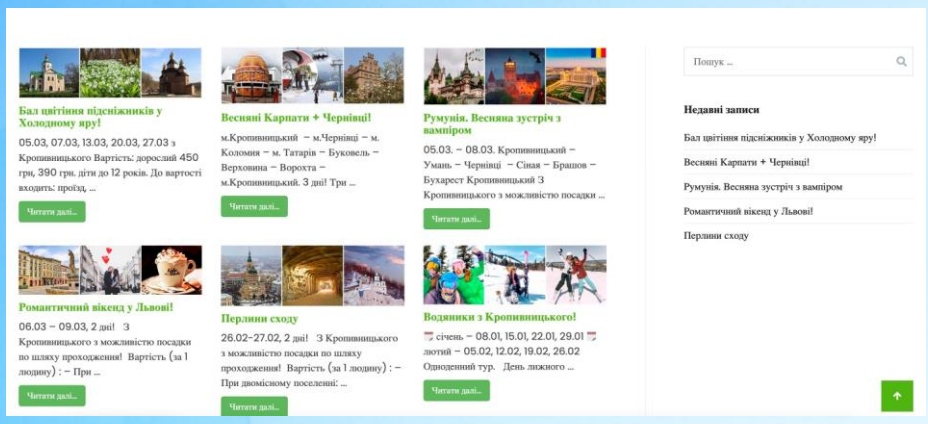
Мета та завдання проекту

Мета проекту - розробка інтернет-платформи для агентства з організації подорожей, вирішити проблеми з пошуком та замовленням путівок для звичайних користувачів, а також буде чудовим бізнес-рішенням для обліку даних про продані путівки для агентств, які займаються організацією туризму та подорожей.

Для реалізації програмного забезпечення необхідно виконати наступні завдання

- дослідити предметну область туристичного бізнесу та виявити потреби потенційних користувачів програмного продукту;
- провести детальний аналіз існуючих рішень
- розробити технічне завдання
- розробити архітектуру веб-додатку та бази даних
- вибрати технології для розробки;
- розробити зручний та привабливий для користувачів інтерфейс
- розробити програмний продукт
- протестувати готову інтернет-платформу

Ksamil Travel

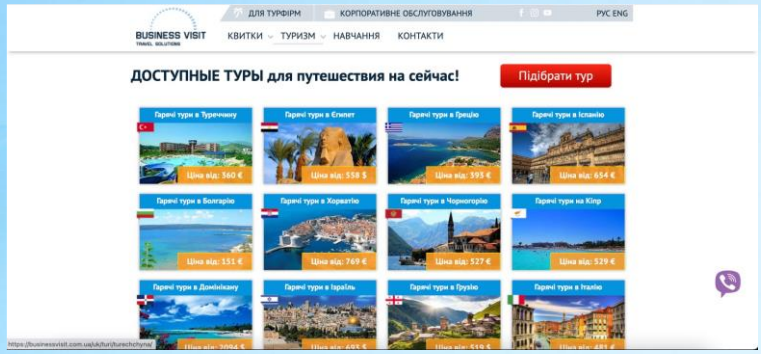


Один з найпопулярніших веб-сайтів для замовлення путівок.

Вирізняється поміж інших своїм сучасним мінімалістичним інтерфейсом та приємними оку світлими кольорами.

Недоліком є те, що основний функціонал зосереджений у верхньому меню, і вибравши щось в ньому контент змінюється далеко внизу, а скролінг сторінки в цей момент не відбувається.
А також те, що весь контент на сайті доступний лише однією мовою.

Business Visit

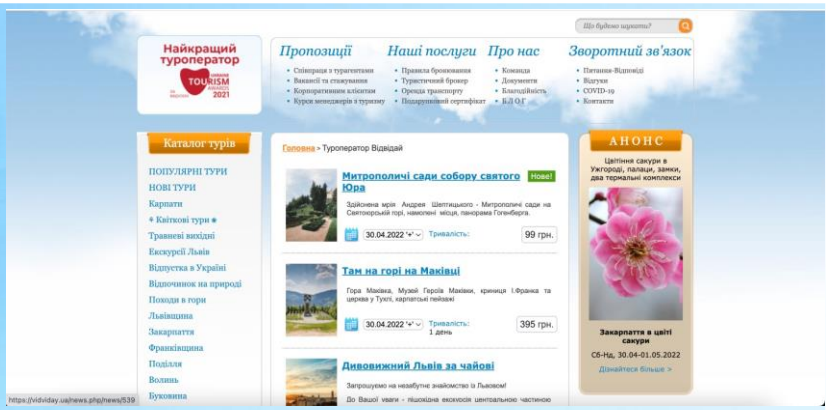


Основним недоліком цього сайту є його інтерфейс Незважаючи на вдало підібрані кольори, шрифт не виглядає сучасним. Також багато зайвого тексту, який потрібно довго читати. Крім того внизу постійно з`являється реклама, яка перекриває контент, поки її не закрити. Головним недоліком є те, що сайт іноді видає помилки, наприклад повідомлення з кодом 404, про те, що сторінку не знайдено, а отже містить посилання на неіснуючі сторінки, що є негативним досвідом для користувача.

Серед переваг можна виділити вибір мов (присутні 3 мови) а також посилання на соціальні мережі (Facebook, Instagram та Youtube), де можна дізнатись більше інформації, переглянути більше фотографій з подорожей та почитати відгуки користувачів. Головною перевагою і тим, що вирізняє цей сайт поміж інших, є те, що можна побачити графік зміни цін на певний тур, що може допомогти користувачу визначитись з вибором.

Відвідай UA

Інтерфейс додатку простий, мінімалістичний, містить градієнтні кольори та має світлу кольорову гамму.



З переваг можна виділити наявність категорій та підкатегорій, що робить цей веб-додаток унікальним в даному списку рішень. Також є зручний пошук і можливість завантажити текстовий опис в форматі .doc і переглянути коротке відео про головні локації даної подорожі. Реклама майже не зустрічається

Головним недоліком є те, що на мобільних пристроях вигляд сайту не змінюється, а лише пропорційно зменшується, що робить розмір шрифту та зображень надто малим для читання та перегляду. Також відсутня фільтрація даних і система оплати.

Порівняння наявного ПЗ

Назва додатку	Пошук та фільтрація	Зручність інтерфейсу	Реклама	Наявність системи оплати	Обов'язкова авторизація
Ksamil Travel	Присутні	6/10	Відсутня	Присутня	Так
Business Visit	Відсутні	5/10	Присутня	Відсутня	Ні
Відвідай ЮА	Присутні	8/10	Присутня	Присутня	Ні

Функціональні вимоги

Для звичайного користувача:

- реєстрація
- авторизація
- перегляд списку турів
- пошук за назвою та описом
- фільтрація за ціною та континентом
- детальний перегляд вибраної путівки
- перегляд галереї фотографій вибраного місця
- додавання в корзину
- видалення з корзини
- замовлення путівки з оплатою
- перегляд історії своїх замовлень

Для адміністратора:

- авторизація
- перегляд списку турів
- додавання нової подорожі з завантаженням фото
- перегляд всіх замовлень всіх користувачів

Діагарама варіантів використання

Використані технології

node.js + express

HTML + CSS

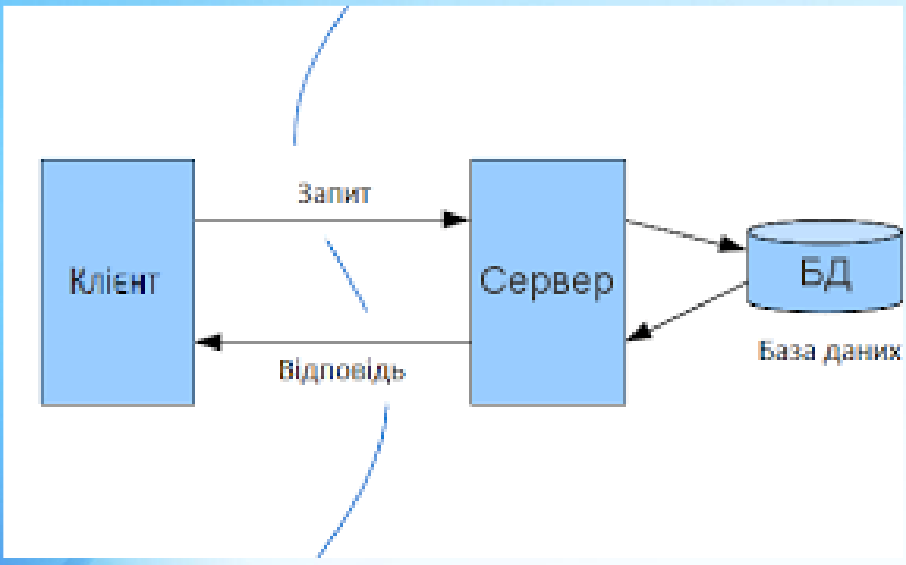
React

Redux

MongoDB.

Архітектура рівня доступа до даних

Клієнт-серверна архітектура



Архітектура рівня представлення:

MVC
Model View Controller

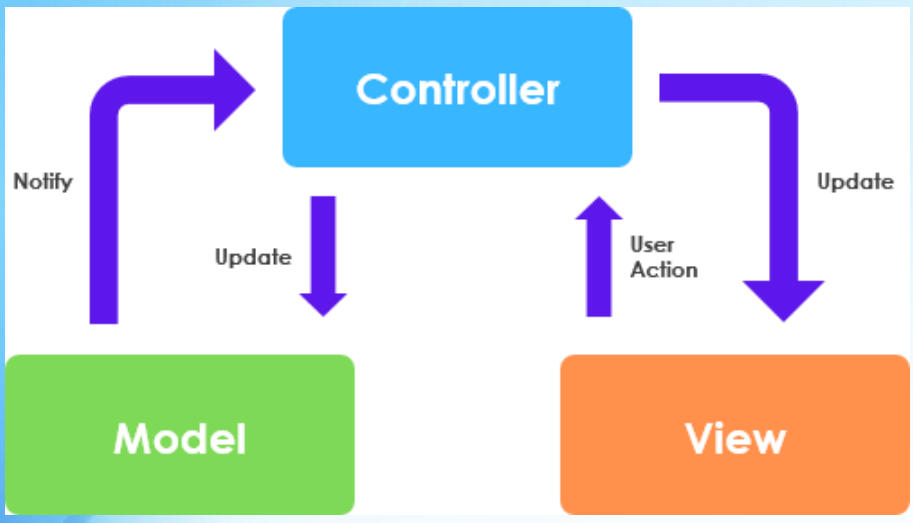
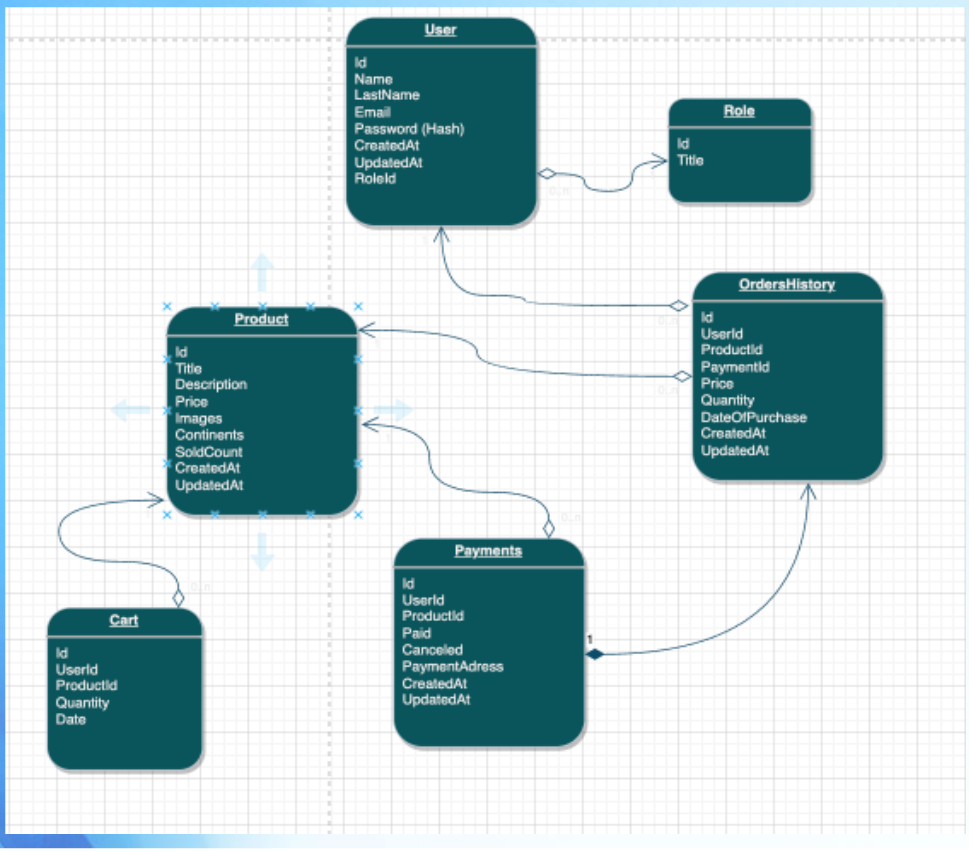
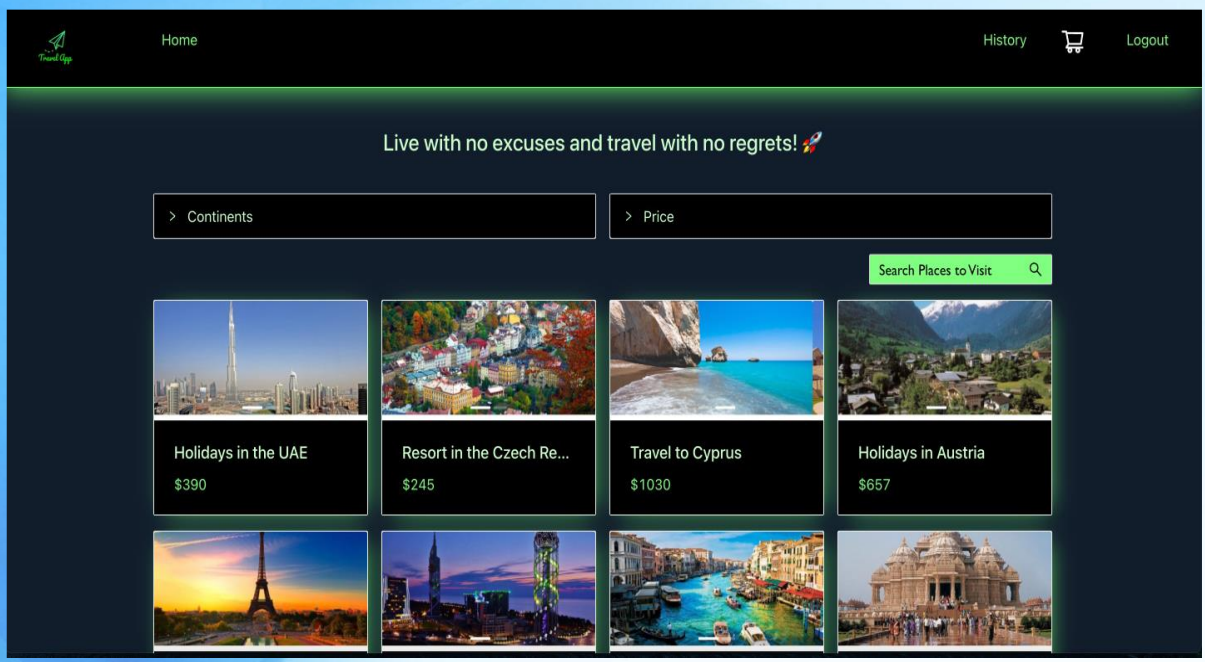


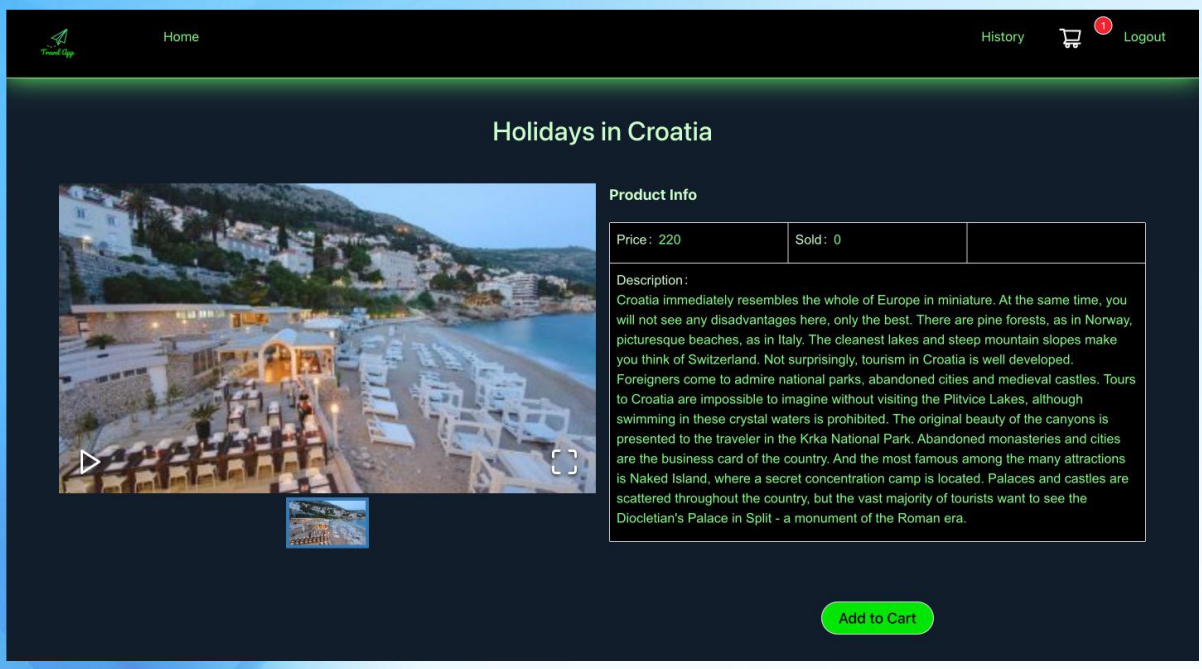
Схема бази даних



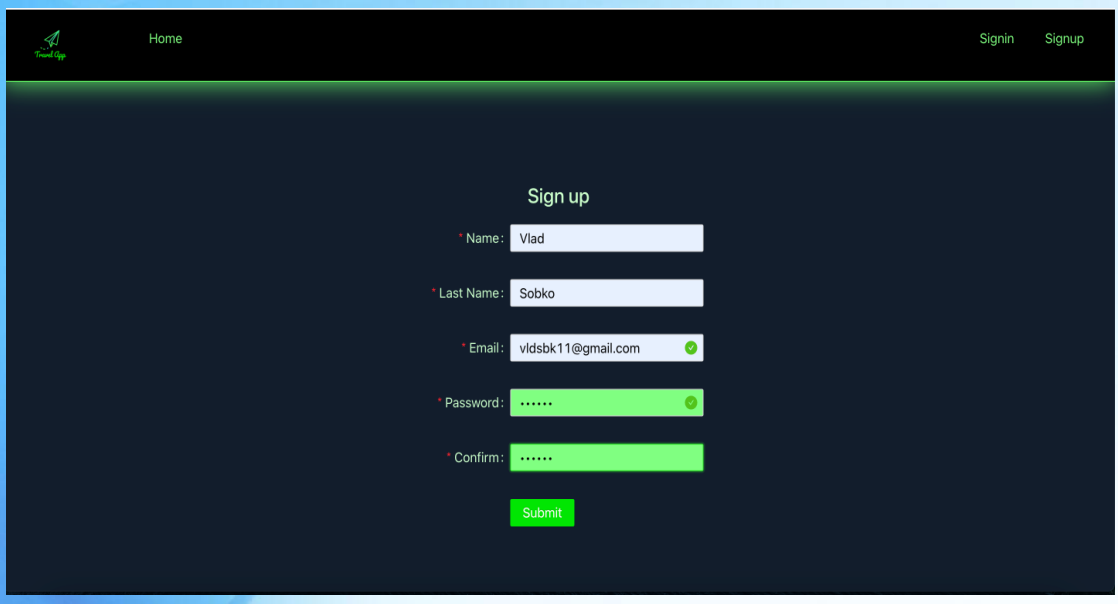
Реалізація проекту. Головна сторінка



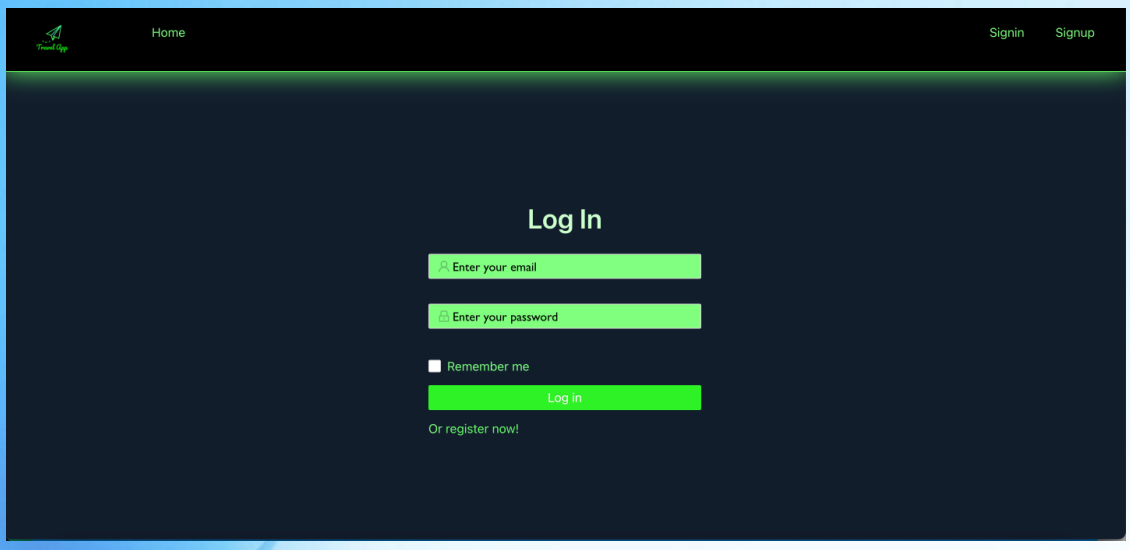
Реалізація проекту. Сторінка деталей туру



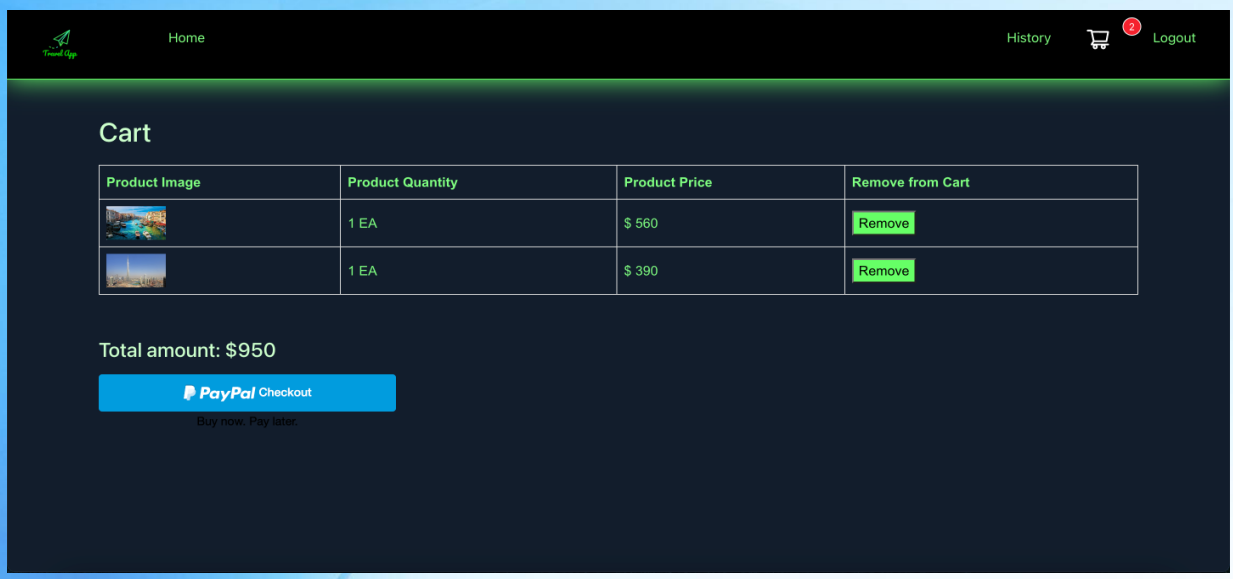
Реалізація проекту. Сторінка реєстрації



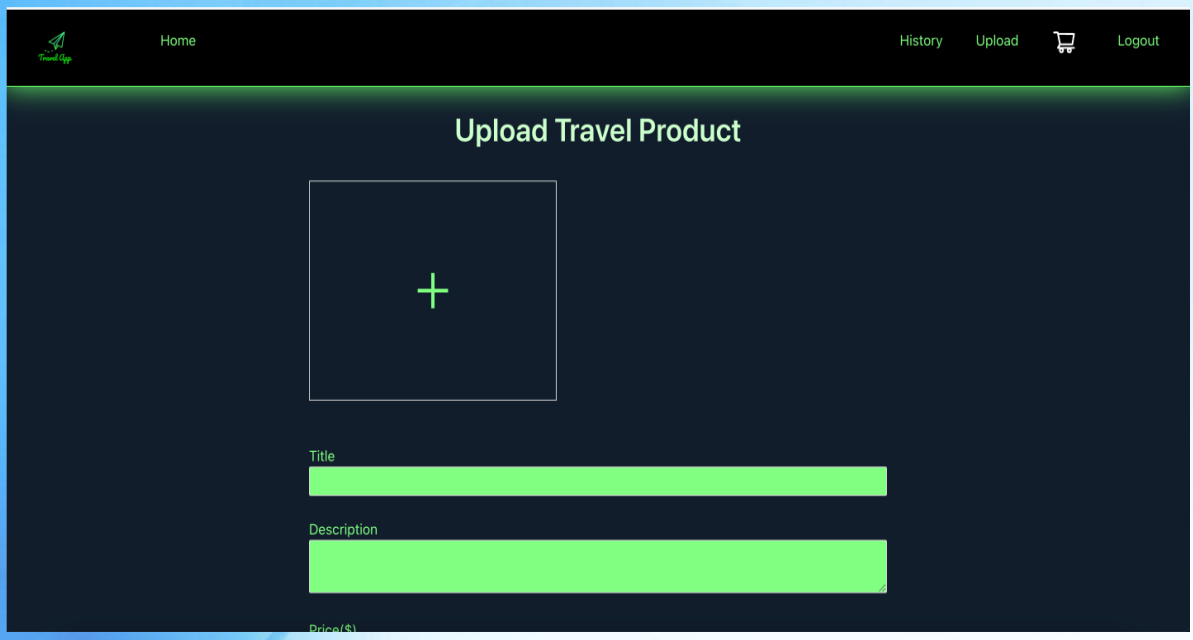
Реалізація проекту. Сторінка авторизації



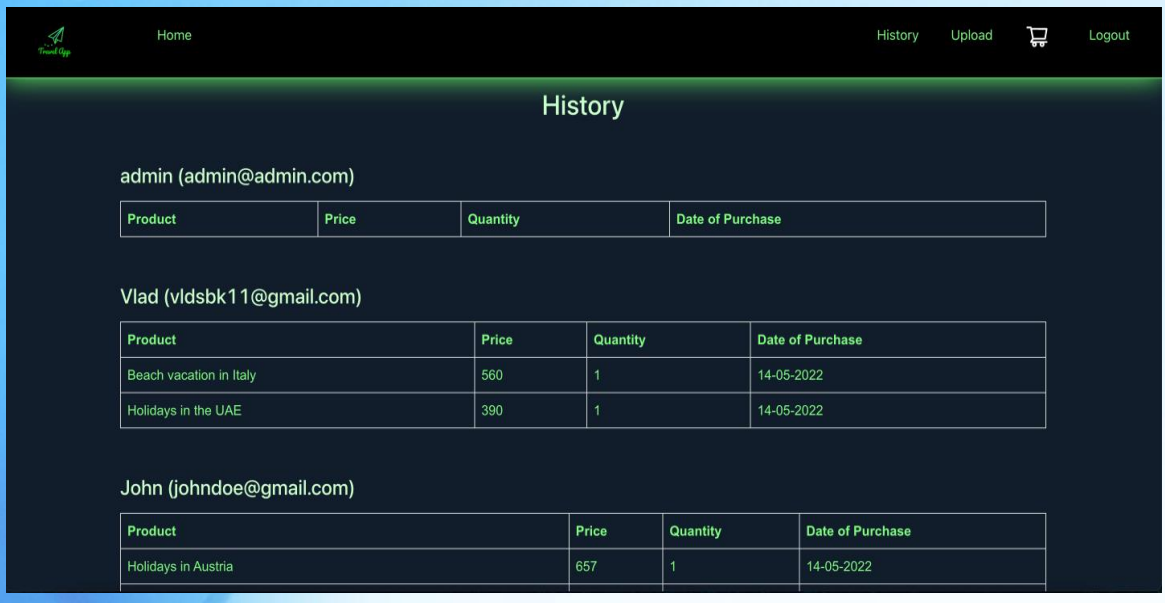
Реалізація проекту. Сторінка корзини



Реалізація проекту. Панель адміністратора. Додавання нового туру



Реалізація проекту. Панель адміністратора. Перегляд замовлень користувачів



Висновки

- Спроектовано інтерфейс, логіку та архітектуру веб-застосунку, базу даних і метод роботи API.
- Створений веб-сайт відповідає поставленому завданню та усім визначеним вимогам.
- Програмний продукт успішно сконструйований та готовий до використання. Судячи з цього, можна зробити висновок, що даний програмний продукт є цілком працездатним та справним.

Дякую за увагу!

Завідувачу кафедри інженерії програмного забезпечення проф. Бедратюку Л. П.

здобувача вищої освіти
Собка Владислава Вадимовича
Прізвище, ініціали

факультет ІТ, 3 курс, група ІПЗс-19-1

ЗАЯВА

З правилами чинного Положення «Про дотримання академічної доброчесності в Хмельницькому національному університеті» від 26.09.2020 (зі змінами від 26.11.2020), згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування заходів дисциплінарної та академічної відповідальності, ознайомлений. Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на плагіат оповіщений та надаю свою згоду на обробку та збереження університетом моєї роботи в інституційному репозитарії університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-технічних засобів (Unicheck та Anti-Plagiarism) та використання роботи для виявлення плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення плагіату в текстах робіт.

Робота для перевірки університетом надається в друкованому та електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

30.05
дата

підпис

Завідувачу кафедри
інженерії програмного забезпечення
проф. Бедратюку Л. П.
студента групи ІПЗс-19-1
Собка Владислава Вадимовича
Прізвище, ініціали

ЗАЯВА

Прошу закріпити за мною тему дипломного проекту освітнього ступеня «бакалавр» за спеціальністю 121 «Інженерія програмного забезпечення»: Інтернет-платформа «Агентство подорожей» .

(керівник дипломного проекту – Праворська Наталія Іванівна)
Прізвище, ім'я, по батькові

30.05
Дата


Підпис студента

Fri May 27 10:19:43 EEST 2022, Хіврич Володимир Русланович, Хмельницький національний університет, ХНУ

Anti-Plagiarism v-15.257

Максимальне співпадіння з одним документом 4.0%

Словники перевірки: en_US, ru_RU, ua_UA. **Помилок в документах: 11%**

ID: 104085 Назва: Інтернет-платформа «Агентство подорожей» Додано в БД: 2022-05-27 Автора: В.В. Собко Керівники: Н.І. Праворська Консультанти: Опоненти:	Документ		Сумарний збіг по Базі Даних	
	Символи	Лексеми	Символи	Лексеми
	78765	780	7352 (9%)	97 (12%)

Джерело плагіату

ID	Опис	Наявність плагіату в документі	
		Символи	Лексеми

UNICHECK

Ім'я користувача:
Кафедра ІПЗ

Дата перевірки:
27.05.2022 11:16:22 EEST

Дата звіту:
27.05.2022 11:18:49 EEST

ID перевірки:
1011347892

Тип перевірки:
Doc vs Internet + Library

ID користувача:
100005589

Назва документа: Собко_ІПЗс-19-1_Диплом_без додатків

Кількість сторінок: 79 Кількість слів: 12443 Кількість символів: 97995 Розмір файлу: 8.79 MB ID файлу: 1011233803

Виявлено модифікації тексту (можуть впливати на відсоток схожості)

8.37%
Схожість

Найбільша схожість: 3.63% з джерелом з Бібліотеки (ID файлу: 1008408523)

2.84% Джерела з Інтернету 239 ..... Сторінка 81

6.49% Джерела з Бібліотеки 125 ..... Сторінка 82

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 10

Підозріле форматування 16 сторінок

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА ДИПЛОМНИЙ ПРОЕКТ
освітнього ступеня «Бакалавр»

Дипломник Собко Владислав Вадимович

Тема Інтернет-платформа «Агентство подорожей».

Спеціальність 121 – Інженерія програмного забезпечення

Обсяг дипломного проекту:

Кількість листів креслень 0; кількість сторінок записки 117

1.Короткий зміст пояснювальної записки та прийнятих рішень У дипломному проекті було досліджено і проаналізовано предметну область туристичного бізнесу та виявлено потреби потенційних користувачів програмного продукту. Був проведений детальний аналіз існуючих рішень, визначено їх переваги і недоліки, та доведено актуальність розробки нового програмного забезпечення. Було розробити технічне завдання, архітектуру веб-додатку та бази даних, вибрано технології для розробки, розроблено зручний та привабливий для користувачів інтерфейс, розроблено програмний продукт, а також протестовано готову інтернет-платформу.

2. Висновок про відповідність проекту поставленому завданню Дипломний проект виконаний відповідно до поставленого завдання та з дотриманням всіх вимог.

3. Характеристика виконання кожного розділу проекту, ступінь використання останніх досягнень науки і техніки та передових методів роботи У вступі доведено актуальність теми, визначено мету та завдання дипломного проектування. У першому розділі проведено аналіз предметної області, розглянуто існуючі рішення та визначені функціональні і нефункціональні вимоги до розроблюваного програмного забезпечення. У другому розділі проведено аналіз сучасних архітектур, розглянуто їх переваги і недоліки та визначено, що система буде відповідати монолітній архітектурі та моделі клієнт-сервер. У третьому розділі підготовлено всі залежності для написання коду та виконано практичну розробку програмних модулів і описано їх особливості, в результаті чого створено програмний продукт. В четвертому розділі було виконано тестування системи та проведено його у відповідності до функціональних вимог, в результаті було підтверджено правильну роботу програми.

4. Позитивні сторони проекту Тематика дипломного проекту є актуальною, оскільки користувачі потребують простої і доступної програмної системи, що надавала б перевагу над іншими подібними рішеннями в галузі туризму, адже зараз саме інформаційні технології відіграють провідну роль в цій справі. Також було застосовано новітні технології для побудови програмного продукту та актуальні архітектурні рішеня.

5. Негативні сторони проекту Проект реалізований лише з однією мовою, без можливості зміни. Було б доцільно додати ще одну-дві мови з можливістю вибору.

6. Оцінка графічного оформлення та пояснювальної записки проекту Графічне оформлення виконано відповідно до теми дипломного проекту та подано у вигляді діаграм і рисунків. Пояснювальна записка оформлена згідно вимог чинних стандартів.

7. Відгук про дипломний проект в цілому Дипломний проект заслуговує позитивної оцінки. Матеріал пояснювальної записки структурований, послідовний, чіткий та простий, що дозволяє чітко зрозуміти викладений матеріал у рамках тематики дипломного проекту. Графічний матеріал дає можливість наочно побачити деталі проектування системи.

8. Інші зауваження

9. Оцінка дипломного проекту Дипломний проект виконаний у повному обсязі, відповідає поставленій задачі та заслуговує на оцінку «відмінно».

РЕЦЕНЗЕНТ Говорущенко Тетяна Олександрівна, доктор технічних наук, завідувач кафедри комп'ютерної інженерії та інформаційних систем (КІС).

“ 30 ” травня 2022 р. _____
(підпис)

РІШЕННЯ ЕКСПЕРНОЇ КОМІСІЇ
КАФЕДРИ ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ
ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатом звіту подібності щодо роботи, генерованого системою виявлення текстових збігів/ідентичності/схожості:

Назва: Інтернет-платформа «Агентство подорожей»

Автор: Собко Владислав Вадимович
Спеціальність: 121 – Інженерія програмного забезпечення
Освітня програма: Освітньо-професійна програма «Інженерія програмного забезпечення»
Науковий керівник: Праворська Наталія Іванівна, кандидат педагогічних наук, доцент
Після аналізу звіту подібності зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі, є законними і не є плагіатом. Робота приймається до захисту.	**відповідає**
2	Виявлені запозичення не є плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи. Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за 2 дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої та електронної версії роботи	
3	Виявлені запозичення не є плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнені. Робота може бути допущена до захисту (наступного року) після того як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	
5	Інше:	

Підтвердження:

Запозичення, виявлені в роботі, є законними і не є плагіатом, оскільки:

1) у тексті дипломної роботи системами перевірки на плагіат виявлено схожість з деякими документами в частині загальновживаних обов'язкових словосполучень у стандартних бланках (титулка, бланк завдання), у структурі змісту, назвах розділів/підрозділів тощо та в назвах публікацій у переліку джерел посилання;

2) в якості запозичень системою було зафіксовано деякі послідовності вихідного коду і посилання на бібліотеки, які є стандартними мовними конструкціями програмування та не можуть розглядатися як об'єкт авторських прав і, відповідно, їх порушення;

3) усі запозичення є фрагментарними або мають належним чином оформленні посилання;

4) виявлені модифікації тексту не впливають на відсоток схожості.

Сумарний обсяг всіх запозичень, визначений системою виявлення збігів ідентичності/ схожості, складає 8,37 % і адресується до 364 джерел, що, з урахуванням наведених обґрунтувань, відповідає характеру теми і свідчить на користь дипломного проекту.

Керівник _____ Н.І. Праворська

Гарант ОП _____ Л. П. Бедратюк

Завідувач кафедри _____ Л. П. Бедратюк