ДИПЛОМНА РОБОТА

ДР.ПІс – 03.00.000 ПЗ

Група ПІс-2015

Бернацький О.М.

2019

ПВНЗ Університет Короля Данила

Кафедра Інформаційних технологій та програмної інженерії

УДК 004

ДИПЛОМНА РОБОТА

Тема ***Розробка бази даних та серверної частини Web-ресурсу з надання туристичних послуг***

Напрям підготовки ***6.050103 «Програмна інженерія»***
(код і назва спеціальності)

ПОЯСНЮВАЛЬНА ЗАПИСКА

ДР.ПІс – 03.00.000 ПЗ
(позначення)

Студент
Бернацький О.М.
(підпис) (дата) (розшифрування підпису)

Керівник проекту
к.т.н *Пашкевич О.П.*
(посада) (підпис) (дата) (розшифрування підпису)

Нормоконтроль
к.т.н *Мануляк І.З.*
(посада) (підпис) (дата) (розшифрування підпису)

Допускається до захисту
Завідувач кафедри
д.т.н.,доц. *Мельничук С.І.*
(посада) (підпис) (дата) (розшифрування підпису)

2019

ПВНЗ УНІВЕРСИТЕТ КОРОЛЯ ДАНИЛА

Факультет *Інформаційних технологій*
Кафедра *Інформаційних технологій та програмної інженерії*
Напрям підготовки *6.050103 «Програмна інженерія»*

ЗАТВЕРДЖУЮ:
Завідувач кафедри ІТПІ
_____ *С.І. Мельничук*
"____" _____ 2019 р.

ЗАВДАННЯ
НА ДИПЛОМНИЙ ПРОЕКТ (РОБОТУ)

Студенту *Бернацькому Олександру Мирославовичу*

1. Тема проекту (роботи) *Розробка бази даних та серверної частини web-ресурсу з надання туристичних послуг*

Затверджена наказом ректора Університету Короля Данила від *15.11.2018 р. № 20/4*
2. Термін здачі студентом закінченого проекту (роботи) *10.06.2019 р.*
3. Вихідні дані до проекту (роботи) *Java, Spring Framework, MySQL,*

4. Зміст пояснювальної записки (перелік питань, що їх належить розробити)
1. Опис туристичних послуг та існуючих рішень з їх надання. 2. Розробка алгоритмів та взаємозв'язків між функціональними об'єктами. 3. Back-end частина проекту.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
1. Розгляд аналогів. 2. Структура бази даних. 3. Демонстрація проекту.

6. Консультанти з проекту (роботи), із зазначенням розділів проекту, що стосуються

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 30.10.18

Керівник Пашкевич О.П.

Завдання прийняв до виконання Бернацький О.М.

КАЛЕНДАРНИЙ ПЛАН

Пор №	Назва етапів дипломного проекту (роботи)	Термін виконання етапів проекту (роботи)	Примітка
1.	Огляд існуючих сайтів з надання туристичних послуг	02.12.2018	
2.	Вибір та обґрунтування технологій для виконання завдання	15.12.2018	
3.	Розробка функціоналу	08.01.2019	
4.	Оформлення пояснювальної записки	26.03.2019	
5.	Оформлення презентації	12.04.2019	
6.	Підготовка до захисту	16.05.2019	
7.			

Студент-дипломник *Бернацький О.М.*
(підпис) (розшифровка підпису)

Керівник проекту *Пашкевич О.П.*
(підпис) (розшифровка підпису)

АНОТАЦІЯ

Відповідно до поставленого завдання у дипломній роботі проводиться та описується процес розробки та тестування API, метою якого є створення бек-енд частини сайту. У процесі виконання буде розроблено API, який буде виконувати всі команди з клієнту, яке дозволить гнучко використовувати всі розроблені функції. API може бути використаний також окремо від користувацької частини, так як всі функції будуть розроблені таким чином, щоб їх можна було використати на різних реалізаціях веб-сайтів.

ANNOTATION

In accordance with the task set in the thesis, the development and testing process of ARI, the purpose of which is the creation of the backend of the site, is described and described. In the process of implementation, an API will be developed, which will execute all the commands from the client, which will allow flexible use of all developed functions. API can also be used separately from the user's part, since all functions will be designed in such a way that they can be used on different website implementations.

РЕФЕРАТ

Розрахунково-пояснювальна записка:

- 77 сторінки;
- 34 рисунків.

Завданням на дипломну роботу є розробка бази даних та серверної частини web-ресурсу з надання туристичних послуг. В першому розділі проекту відбувається дослідження туристичної сфери та описується існуючі сайти з своїми API які є доступні для перегляду. Також в першому розділі відбувається обґрунтування використаних технологій .

У другому розділі описана структура бази даних яка розроблялась для даного проекту, реалізація ентіті класів та REST інтерфейсу .

Третій розділ описує API та наочно демонструє його функціонал, також в цьому розділі наведені приклад кодів і їх короткий опис.

Об’єктом дослідження є три сайти із своїми API для туристичних агенцій України. Метою проекту є створення API, яке буде підключатись до Front-end частини проекту – сайт туристичних путівок, з функціоналом бронювання путівок, а також бронювання готелів.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ І СКОРОЧЕНЬ

БД – база даних

СКБД – система керування базами даних

JWT – JSON Web Token

API –Application Programming Interface

REST - Representational State Transfer

JS – Java Script

XML – eXtensible Markup Language

SOAP – Simple Object Access Protocol

AMF – Action Message Format

RPC - Remote Procedure Call

ORM – Object-Relational Mapping

JSON – JavaScript Object Notation

ОС – операційна система

ВСТУП

На сьогоднішній день існують досить багато різних API які є посередниками між онлайн сервісами і програмами, які розробляють додатки на телефони, планшети, ПК.

API – це набір визначень підпрограм, протоколів взаємодії та засобів для створення програмного забезпечення. Спрощено - це набір чітко визначених методів для взаємодії різних компонентів. API надає розробнику засоби для швидкої розробки програмного забезпечення. API може бути для веб-базованих систем, операційних систем, баз даних, апаратного забезпечення, програмних бібліотек.

Одним з найпоширеніших призначень API є надання набору широко використовуваних функцій, наприклад для малювання вікна чи іконок на екрані. Програмісти використовують переваги API у функціональності, таким чином їм не доводиться розробляти все з нуля. API є абстрактним поняттям — програмне забезпечення, що пропонує деякий API, часто називають реалізацією (англ. implementation) даного API. У багатьох випадках API є частиною набору розробки програмного забезпечення, водночас, набір розробки може включати як API, так і інші інструменти/апаратне забезпечення, отже ці два терміни не є взаємозамінювані.

Високорівневі API часто програють у гнучкості. Виконання деяких функцій нижчого рівня стає набагато складнішим, або навіть неможливим.

В даному дипломному проекті необхідним було розробити API яке буде підключено до сервісів туристичних фірм

1 ОПИС ТУРИСТИЧНИХ ПОСЛУГ ТА ІСНУЮЧИХ РІШЕНЬ З ЇХ НАДАННЯ

1.1 Дослідження туристичних послуг

Туристична послуга є не що інше, як корисна (доцільна) діяльність турпідприємства по задоволенню потреб туриста. Оскільки турпослуги виявляються, як правило, не в однині, правомірно говорити про турпродукту як про комплекс туристичних послуг. І тільки придбавши цей комплекс послуг, турист отримує право на здійснення туру.

М.Б. Біржаков дає наступне визначення: «Туристична послуга - це сукупність цілеспрямованих дій у сфері обслуговування, яка орієнтована на забезпечення і задоволення потреб туриста і екскурсанта, що відповідає цілям туризму, характеру та спрямованості туристичної послуги, туру і не суперечить загальнолюдським принципам моралі і доброго порядку».

Туристичні послуги можуть бути розділені на основні (придбані в пакеті, що гарантує їх обов'язкове використання в місці відпочинку), додаткові (цільові і інфраструктурні послуги, що не входять у вартість туру і придбання яких здійснюється за додаткову плату) і супутні (послуги місцевого інфраструктурного комплексу, якими поряд з місцевими жителями користуються і туристи).

У складі туристичних послуг головними вважаються перевезення і розміщення. У першій розрізняють основний етап перевезення і трансфер. Розміщення передбачає надання туристам спеціально обладнаних місць для ночівлі (готелі, індивідуальні засоби розміщення, мотелі, кемпінги та ін.). Харчування передує розважально-пізнавальним заходам. Далі виділяються екскурсійне обслуговування та анімаційно-дозвільні заходи. І, нарешті, банківська та фінансова сфери обслуговування, системи зв'язку, прокат

автомобілів, ремонтні майстерні, медичне обслуговування, охоронні заходи, фотопослуги - невід'ємні складові організації туристичного відпочинку.

Тур – це певна конкретними характеристиками: географічним маршрутом і порядком його проходження, часовим проміжком, термінами надання, складом і якістю послуг, ціною - цілеспрямована і впорядкована організатором туризму сукупність необхідних і достатніх туристичних послуг, робіт, товарів, туристичних продуктів, що містить принаймні два різних компонента туристичного продукту (наприклад, розміщення і перевезення та ін.), що включає належне забезпечення та всі необхідні і достатні елемент и для задоволення потреб туриста в процесі його здійснення, і пропонована для реалізації на туристичному ринку, як єдине ціле [1].

Отже, що купує турист, приходячи в турагентство? Він купляє не путівку, не тур, не послуги з бронювання, він купляє «туристичний продукт». Саме так записано в спеціальному галузевому законі «Про туризм». Продукт цей являє собою «комплекс послуг з перевезення і розміщення, що надаються за загальну ціну». Що це означає? Купили пакетний тур «переліт + готель + трансфер + страховка» - купили турпродукт, потрапляєте під дію закону. Купили окремо квитки, окремо готель, самі застрахувалися - все, це ні разу не турпродукт, відповідно, закон не діє.

Туристичний продукт формується туроператором. Стати їм може тільки юридична особа (ФОП не може), при дотриманні певних вимог: отримати ліцензію, застрахувати свою діяльність і так далі, всі деталі не важливі. Туроператорів не дуже багато, масових, що охоплюють основні напрямки, і зовсім всього кілька десятків. Вони у всіх на слуху і формують більшу частину потоку по масових напрямках, типу Туреччини, Єгипту, Таїланду і так далі. Туроператор найчастіше сам тур не продає. Він займається тим, що укладає договори з готелями, авіакомпаніями, страховиками, збирає «пакет», призначає на нього єдину ціну і пускає в продаж.

Сформований турпродукт оператор може продавати сам, але в 99% робить це через мережу турагентств. Численні офіси продажів з брендами туроператорів, які з'явилося безліч в останні кілька років, частіше за все турагентства працюють по франчайзингу. Іноді прямі продажі ведуться через операторські сайти, але це не завжди така покупка є вигідною для туриста [2].

1.2 Огляд аналогів Web-сервісів з надання туристичних послуг

Першим сайтом який я вирішив дослідити є otpusk.com (рис. 1.1). Для розробки використана мова програмування JS. Даний сайт містить стандартний функціонал такий як:

– перегляд турів;
– перегляд екскурсійних турів;
– вибір країни туристичної подорожі.

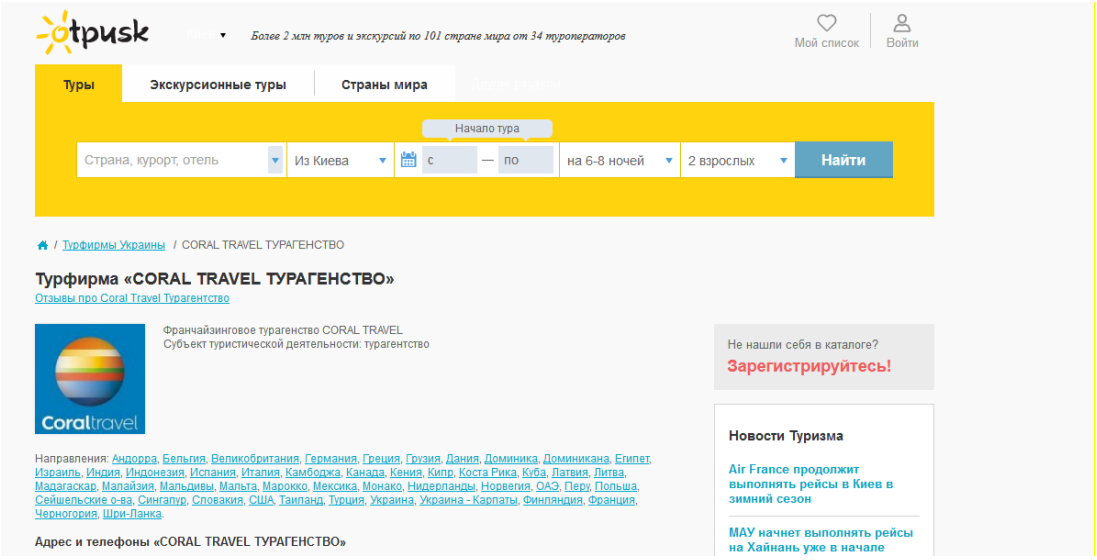

Рисунок 1.1 – Головна сторінка otposk.ua

В вкладці перегляду турів ми можемо переглянути усі доступні тури. Як показано на зображенню (рис. 1.2) нижче – усі тури розміщені у списку,

який має короткі відомості про тур, а саме:

– назвав туру (місце поїздки);
– час туру;
– кількість зірок готелю;
– спосіб прибуття до місця призначення;
– ціна на одну особу.

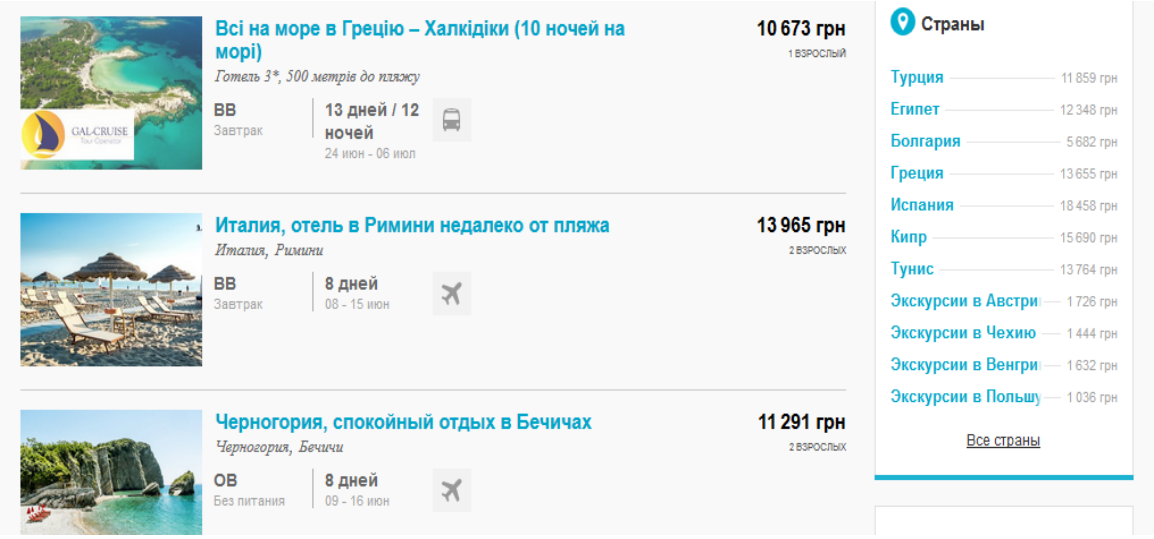

Рисунок 1.2 – Тури otposk.ua

На вкладці про екскурсійні тури нам описуються туристичні путівки які проводяться з гідом. Даний тип туру передбачає відвідування декількох міст, з екскурсіями про історичні пам’ятки, та цікаві місця.

Недоліком цього сайту є те, шо не вказані гарячі тури, які необхідні на кожному сайті. Також одним з головних недоліків є те що на кожну людину замовляється 1 тур і неможливо замовити на декількох людей (сім’ю) один тур. Також не вказано ціну туру для дитини.

Наступний сайт який я розглянув я tat.ua (рис. 1.3). Даний сайт містить такий функціонал:

– рекомендації;
– гарячі тури;
– типи турів;

– замовлення онлайн турів;

– екскурсійні тури.

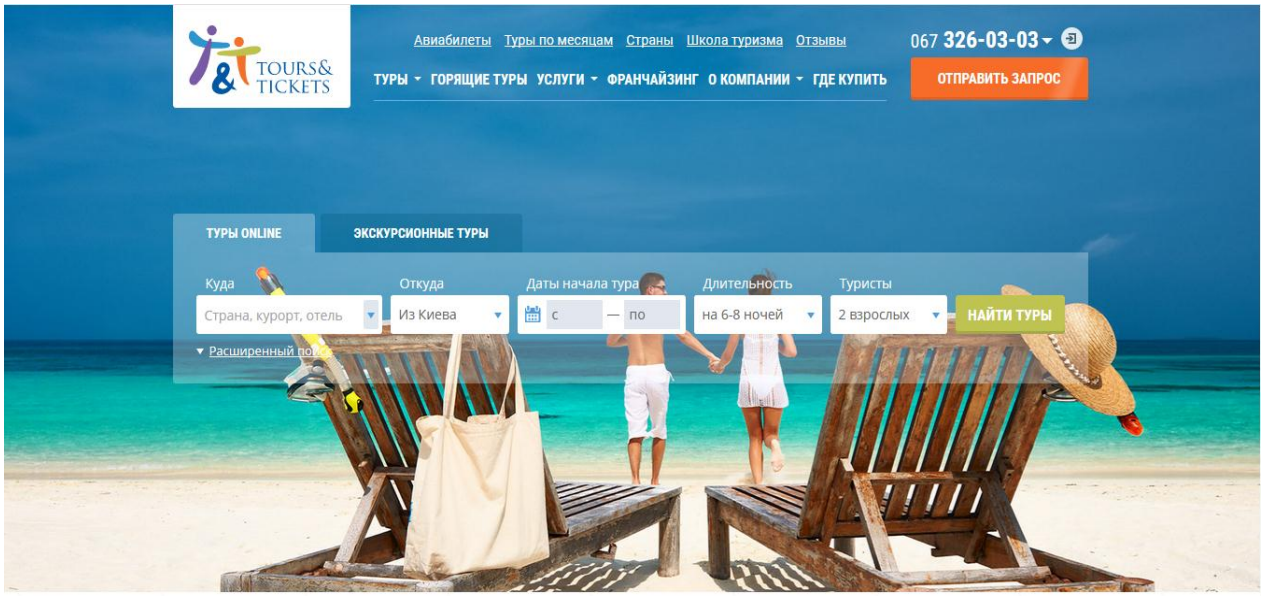

Рисунок 1.3 – Головна сторінка tat.ua

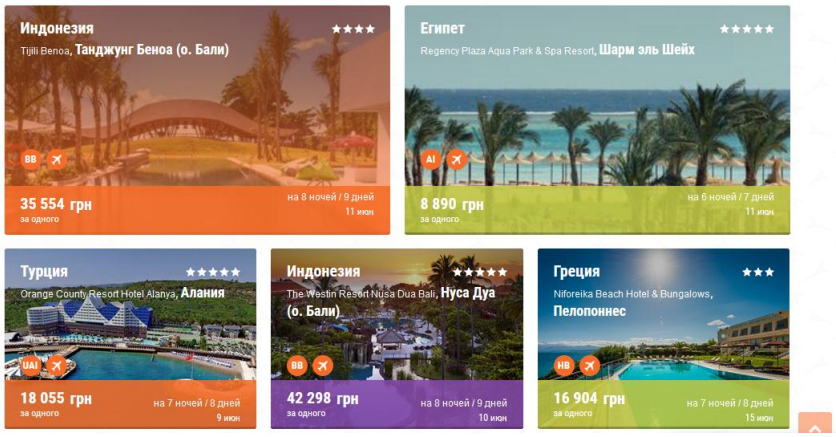

Рисунок 1.4 – Тури tat.ua

На зображенні (рис. 1.4) вище вказані рекомендаційні тури, на яких вказана коротка інформація про тур (місце поїздки), час і ціну на одну людину.

На рисунку (рис. 1.5) нижче в типі відображення списком показують гарячі тури, а саме в них зазначено країну, у яку дають тур, ціну та короткі відомості про нього.

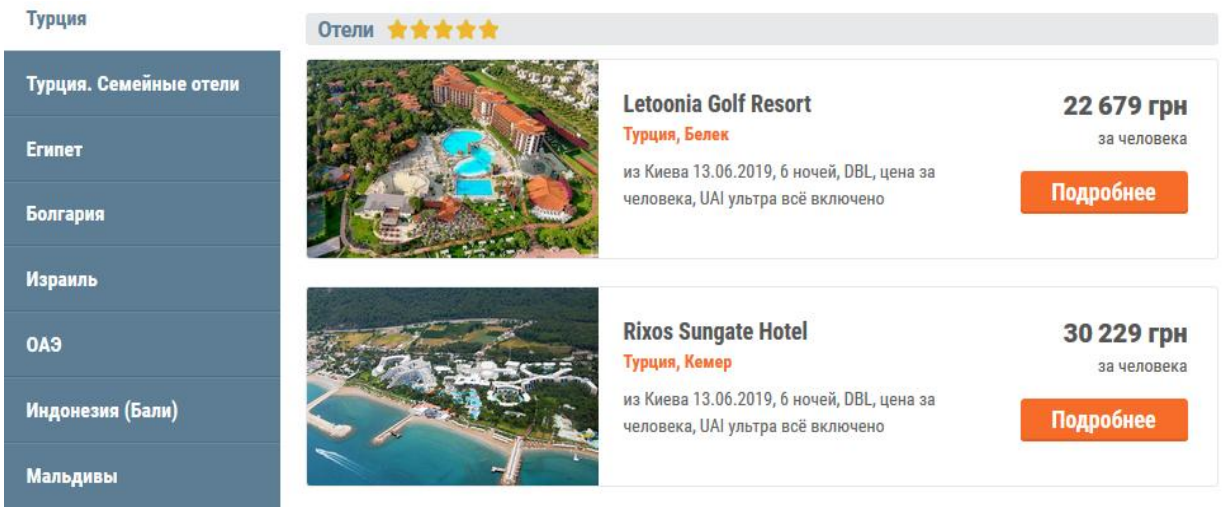

Рисунок 1.5 – Рекомендаційні тури tat.ua

Вкладка ж типи турів (рис. 1.6) дозволяє вказати опцію туру, яку ми хочемо, і відповідно програма підбере із міток інших турів опцію, яку ми вибрали.

Рисунок 1.6 – Турів tat.ua

До недоліків даного сайту можна віднести занадто короткі відомості про тури, вказану ціну, яка записана тільки на 1 дорослу людина, а також не

вказано про відомості гарячих турів а саме – час, скільки ще буде даний гарячий тур і на скільки було знижену відповідно ціну на тури.

І останнім сайтом з яким я ознайомився є сайт sam.ua (рис. 1.7). Даний сайт містить порівняно невеликий функціонал, проте у ньому записані самі головні, а саме потрібні частини:

- пошук по країнах;
- види турів;
- рекомендовані країни;
- наші популярні тури.

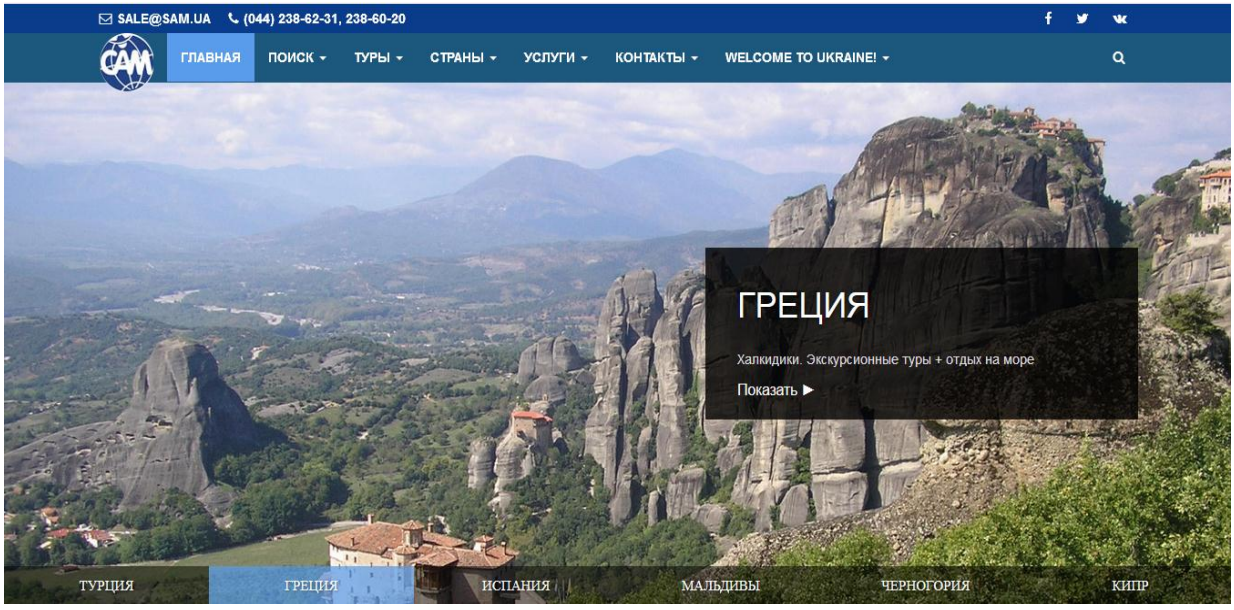

Рисунок 1.7 – Головна сторінка sam.ua

В вкладці Види турів (рис. 1.8) нам пропонують вибрати тури по типу Гірськолижні тури, Лікувальні тури, відпочинок на морі, Екскурсійні тури чи Тури вихідного дня. При виборі певного виду туру нам відкриється сторінка з переліком турів, які підпадають під тип категорії, яку ми вибрали.

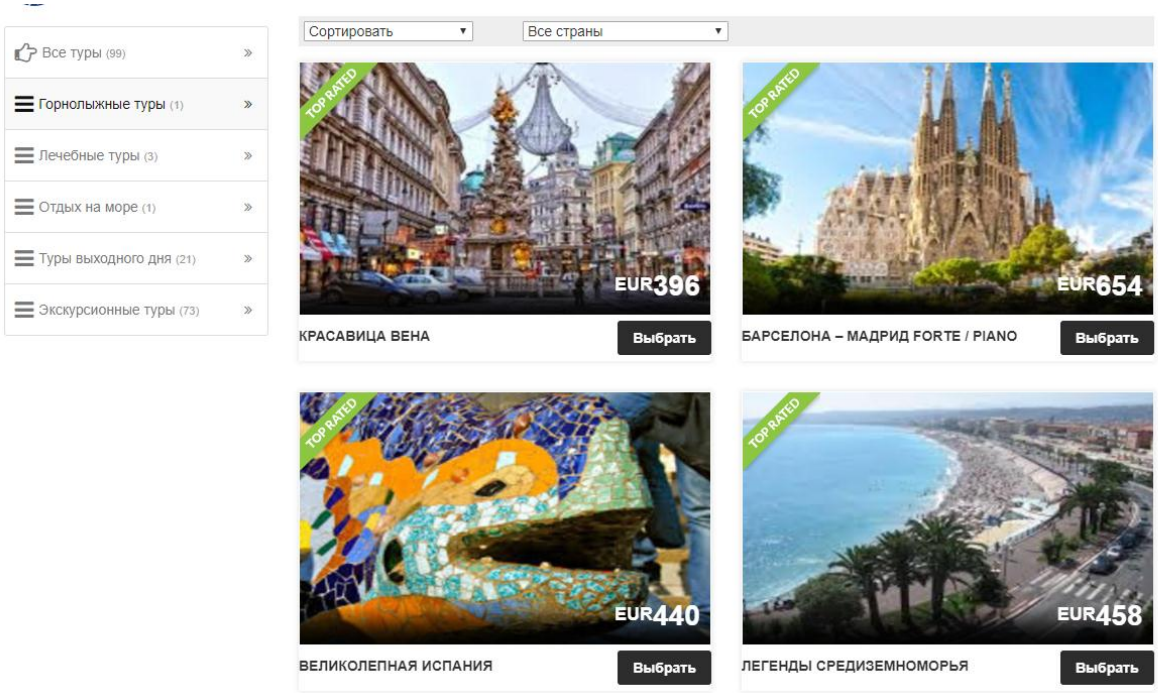

Рисунок 1.8 – Види турів

Також є можливість вибрати рекомендовані сайтом країни (рис.1.9), які покажуть нам тури, які є тільки по даній країні. Хороший вибір, якщо людина хотіла давно відвідати певну країну.

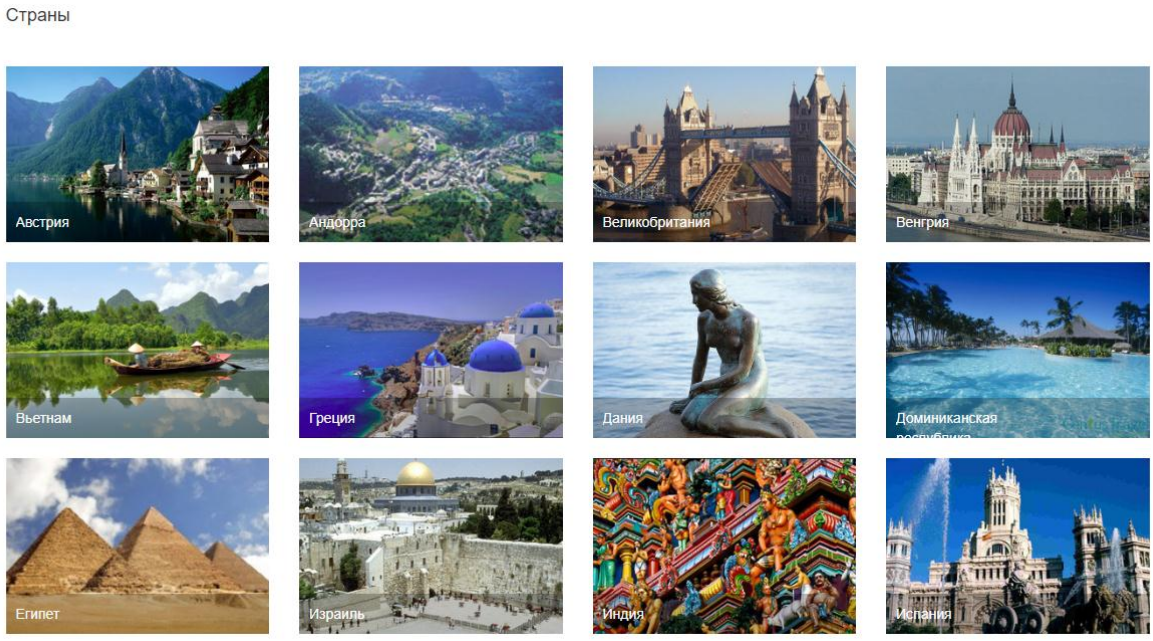

Рисунок 1.9 – Вибір країн

В вкладці популярні тури відображаються тури, які користуються популярністю на даному сайті.

В пошуку туру (рис 1.10) ми можемо вибрати або всі країни, в які доступні тури або певний перелік країн, які хочемо відвідати і які є доступні.

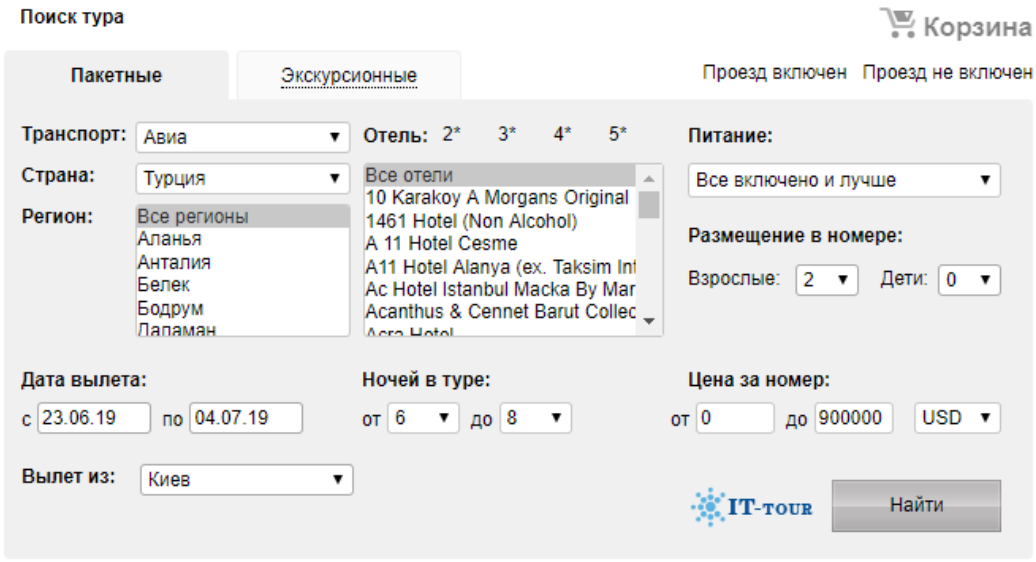

Рисунок 1.10 – Пошук по сайту

Ознайомившись з трьома сайтам я прийшов до висновку що будь який Web-ресурс з надання туристичних послуг повинен мати в собі такі функції як перегляд турів, бронювання, вибір по країнам та типам турів.

1.3 Постановка завдання та вибір технологій

Основним призначенням проекту є створення API для сайту який надає туристичні послуги. Тому в першу чергу я визначив який функціонал повинен бути доступний користувачу на даному ресурсі. До нього можна віднести:

– перегляд доступних турів;

– бронювання турів;

– бронювання готелів;
– можливість зареєструватись.

Оскільки на сайт потрібно додавати різного роду інформацію, для адміністратора був визначений наступний функціонал:

– Додавання турів;
– Додавання готелів;
– Скасування бронювання;
– Зміну даних в турах та готелях.

У даному завданню потрібно створити базу даних, для цього я вибрав СКБД MySQL — компактний багатопотоковий сервер баз даних. Характеризується високою швидкістю, стійкістю і простотою використання . MySQL вважається гарним рішенням для малих і середніх застосувань. Сирцеві коди сервера компілюються на багатьох платформах. До основних переваг можна віднести: простота у встановленні та використанні, підтримується необмежена кількість користувачів, що одночасно працюють із БД, кількість рядків у таблицях може досягати 50 млн, висока швидкість виконання команд, наявність простої і ефективної системи безпеки [3].

Для написання серверної частини була вибрана мова програмування Java. Вона має багато переваг основними з яких є багатоплатфо́рмність, це є плюсом оскільки сайт без переробок може працювати як на Windows, так і Unix сервері.

Також враховуючи те, що розробляється API, був використаний Spring Framework. Основні його особливості можуть бути використані будь-яким додатком Java, але є розширення для створення веб-додатків на платформі Java EE. Незважаючи на це, він не нав'язує якоїсь конкретної моделі програмування, Spring Framework став популярним в спільноті Java як альтернатива, або навіть доповнення моделі Enterprise JavaBean [4].

Для взаємодії з базою даних і правильної реалізації функціоналу мого проекту, я застосував REST, його головною перевагою є відсутність

додаткових внутрішніх прошарків, що означає передачу даних в тому ж вигляді, що і самі дані. Тобто дані не обертаються в XML, як це робить SOAP і XML-RPC, не використовується AMF, як це робить Flash і т.д. Просто віддаються самі дані.

2 РОЗРОБКА АЛГОРИТМІВ ТА ВЗАЄМОЗВ'ЯЗКІВ МІЖ ФУНКЦІОНАЛЬНИМИ ОБ'ЄКТАМИ

2.1 Розробка структури бази даних

Для зберігання всіх даних, за допомогою Системи Управління Базами Даних MySQL була створена база даних у яку увійшли 7 таблиць (рис. 2.1): hotel, tour, tour_resrvation, user, user_role, flyway_schema_history, hibernate_sequence.

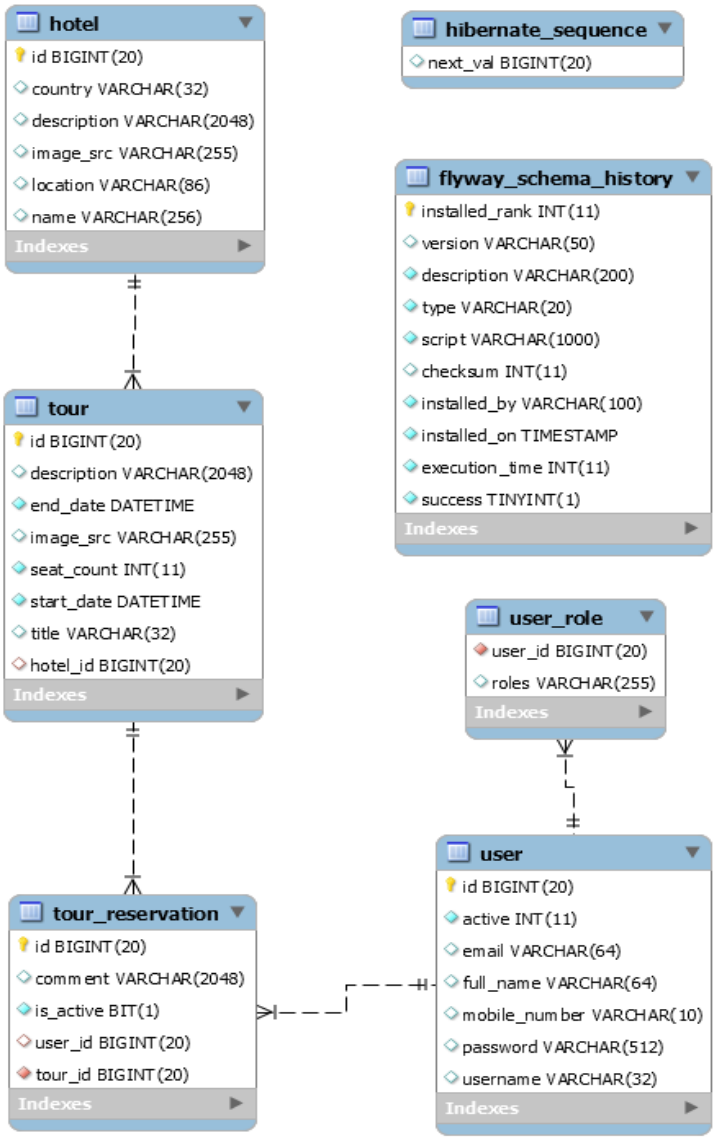

Рисунок 2.1 – Схема бази даних

Таблиця hotel – складається з 6 колонок а саме: id, country, description, img src, location, name. В дану таблиці адміністратор може додавати готелі, вказувати їх розташування, назву, додавати картинку та опис. Нижче наведений код створення цієї таблиці:

```
create table hotel
(
    id          bigint not null,
    country     varchar(32),
    description varchar(2048),
    image_src   varchar(255),
    location    varchar(86),
    name        varchar(256),
    primary key (id)
) engine = InnoDB;
```

Таблиця tour – складається з 8 колонок. В неї увійшли: id, description, end_date, image src, seat_count, start_date, title, hotel_id. Дана таблиця потрібна, щоб користувач міг ознайомитись та замовити потрібний йому тур. Сюди адміністратор вказує назву тура, опис, дату початку та закінчення, зображення, кількість місць в турі, та додає готель з таблиці hotel.

```
create table tour
(
    id          bigint   not null,
    description varchar(2048),
    end_date    datetime not null,
    image_src   varchar(255),
    seat_count  integer  not null,
    start_date  datetime not null,
    title       varchar(32),
    hotel_id    bigint,
    primary key (id)
) engine = InnoDB;
```

Таблиця tour_reservation – в ній я створив 5 колонок: id, comment, is_active, user id, tour_id. Дана таблиця відповідає за резервування турів, сюди входять дані про користувача (його id) який замовив тур, статус бронювання (чи досі воно активне), дані про тур (id), коментар користувача.

```
create table tour_reservation
(
    id        bigint not null,
    comment   varchar(2048),
    is_active bit    not null,
    user_id   bigint,
    tour_id   bigint not null,
    primary key (id)
) engine = InnoDB;
```

Таблиця user – складається з 7 колонок серед них є: id, active, email, full_name, mobile_number, password, username. В таблиці зберігаються дані про користувача, його повне ім’я, логін, мобільний номер, електронна пошта, пароль та статус (чи знаходиться користувач на сайті) .

```
create table user
(
    id            bigint  not null,
    active        INTEGER not null,
    email         varchar(64),
    full_name     varchar(64),
    mobile_number varchar(10),
    password      varchar(512),
    username      varchar(32),
    primary key (id)
) engine = InnoDB;
```

Таблиця user_role – В неї ввійшли наступні колонки: user_id та roles. В ній зберігається інформація про користувача, тобто є він звичайним користувачем чи адміністратором, в залежності від цієї інформації визначаються можливості користувача на сайті.

```
create table user_role
(
    user_id bigint not null,
    roles   varchar(255)
) engine = InnoDB;
```

Таблиця flyway_schema_history – складається з 10 колонок: instlled_rank, version, description, type, script, checksum, installed_by, execution_time, success. Ця таблиця відповідає за відстежування послідовності міграцій, тобто зберігає в собі всі дані міграції, такі як: назву,

скрипт, версію, опис. Міграція схеми виконується над базою коли необхідно оновити або повернути схему бази даних до новішої або старішої версії. Міграція виконується програмно за допомогою інструменту міграцій (англ. Schema migration tool). При виклику інструменту міграції з вказуванням бажаної версії, інструмент автоматично застосовує або відкатує міграції в правильній послідовності аж поки не приведе базу даних до бажаного стану.

Таблиця hibernate_sequence – складається тільки з однієї колонки next_val і відповідає за генерацію id базою даних.

2.2 Розробка ентіті-класів для представлення даних

Для коректної роботи Spring Framework та бази даних була застосована технологія ORM. Це можливість без великих затрат зберігати чи зчитувати об'єкти в БД, оскільки ORM використовує шаблони коду, які мають відмінний дизайн і слідують шаблонами проектування. Це дуже скорочує час на розробку і тестування, простіше супроводжувати, менше коду, немає необхідності самостійно писати запити, створювати об'єкти, тому як можна працювати з наявними об'єктами. Не потрібно перетворювати вручну об'єкти з або в базу даних і тестувати то, чого немає - такий код вже написаний і протестований.

Клас Hotel

Анотація позначає що даний клас є сутністю і для нього буде створена таблиця в базі даних.

```
@Entity
public class Hotel {@Entity
public class Hotel {
```

Поле позначене анотацією @id є ключовим

```
@Id
```

Анотація вказує що поле є автогенерованим (кожного разу при додаванні нового запису в бд, буде генеруватись id)

```
@GeneratedValue(strategy = GenerationType.AUTO)
```

Ідентифікатор готелю

```
private Long id;
```

Спеціальна анотація для полів типу String, означає що дане поле не може бути пустить і не може складатись з пробілів.

```
@NotBlank
```

Анотація @Column з параметром unique вказує що колонка є обов'язковою.

```
@Column(unique = true)
```

Анотація @Size вказує обмеження по розміру колонки (мінімальний та максимальний розмір).

```
@Size(min = 3, max = 256)
```

Ім'я готелю

```
private String name;
```

Наступні рядки відповідають за колонку з описом готелю.

```
@NotBlank
@Size(min = 32, max = 2048)
private String description;
```

Країна розміщення готелю.

```
@NotBlank
@Size(min = 2, max = 32)
private String country;
```

Місце розташування готелю, сюди відноситься місто та вулиця.

```
@NotBlank
@Size(min = 2, max = 86)
private String location;
```

Анотація @Column з параметром updatable визначає те, чи колонка підлягає для оновлення

```
@Column(updatable = false)
```

Назва фото готелю

```
private String imageSrc;
```

Клас Tour

Наступний код відповідає за індифікатор готелю.

```
@Id
@GeneratedValue(strategy = GenerationType.AUTO)
private Long id;
```

Назва туру

```
@NotBlank
@Size(min = 4, max = 32)
private String title;
```

Опис туру

```
@NotBlank
@Size(min = 32, max = 2048)
private String description;
```

Наступна анотація вказує, що поле як і колонка в базі даних не можуть мати значення null.

```
@NotNull
```

Анотація вказує що дане поле є датою, в ній можна вказати що зберігати (дату, час чи те і інше)

```
@Temporal(TemporalType.TIMESTAMP)
```

Дата початку туру

```
private Date startDate;
@NotNull
@Temporal(TemporalType.TIMESTAMP)
```

Дата закінчення туру

```
private Date endDate;
@NotNull
```

Кількість місць у турі

```
private Integer seatCount;
```

Анотація @OneToOne вказує що один запис в цій таблиці відноситься лише до одного запису у таблиці hotel. FetchType.EAGER - означає що при завантаженні з бд, будуть завантажені і всі вкладені об'єкти з інших таблиць.

```
@OneToOne(fetch = FetchType.EAGER)
```

@JoinColumn(name = "hotel_id") - вказує що колонка яка буде створена в цій таблиці повинна мати задану назву.

```
@JoinColumn(name = "hotel_id")
```

Готель, до якого належить тур

```
private Hotel hotel;
```

Клас TourReservation

Наступна анотація означає, що один користувач може мати багато резервувань, FetchType.LAZY означає, що з бази даних буде братись не весь об'єкт а лише основні дані, і тільки якщо буде потрібне якесь поле - дані будуть вибрані з бд.

```
@ManyToOne(fetch = FetchType.LAZY)
```

Вказує що колонка, яка буде посилатись на запис з іншої таблиці буде називатись user_id

```
@JoinColumn(name = "user_id")
```

Означає що дане поле не буде перетворюватись у JSON при отриманні всіх резервувань з бд.

```
@JsonIgnore
```

Об'єкт класу User який відноситься до резервування

```
private User user;
```

Виводить лише id користувача у JSON

```
@Column(name = "user_id", insertable = false, updatable = false)
private Long userId;
```

Клас User

Колонка яка відповідає за ID

```
@Id
@GeneratedValue(strategy = GenerationType.AUTO)
private Long id;
```

Колонка для логіна

```
@NotBlank
@Column(unique = true)
@Size(min= 3,max = 32)
private String username;
```

В цій колонці зберігається емейл. @Email анотація яка автоматично валідує емейл (запис в бд не може бути доданим поки дане поле не буде містити емейл)

```
@NotBlank
@Column(unique = true)
```

```
@Size(max = 64)
@Email
private String email;
```

Колонка для паролю. @JsonProperty означає що даний параметр може бути лише отриманим в JSON, але при цьому при перетворенні об'єкта в JSON дане поле не буде враховано

```
@NotBlank
@Size(min = 6, max = 512)

@JsonProperty(access = JsonProperty.Access.WRITE_ONLY)
private String password;
```

Колонка яка зберігає повне ім’я користувача

```
@NotBlank
@Size(min = 3, max = 64)
private String fullName;
```

Наступна колонка відповідає за номер телефона. Digits - вказує що дане поле типу String може містити лише 10 символів (10 цифр мобільного)

```
@NotBlank
@Size(min = 10, max = 10)
@Digits(integer=10, fraction=0)
private String mobileNumber;
```

Колонка в якій зберігається інформація про стан користувача (Знаходиться він на нашому сайті чи ні)

```
@NotNull
```

Поле не додається в JSON

```
@JsonIgnore
```

columnDefinition = "INTEGER" - вказує примусовий тип колонки

```
@Column(columnDefinition = "INTEGER")
private boolean active;
```

2.3 Реалізація REST інтерфейсу

В даному підрозділі описуються контролери та їхні методи. Контролер забезпечує «зв'язок» між користувачем і системою. Контролює і направляє

дані від користувача до системи і навпаки. Використовує модель і уявлення для реалізації необхідного дії.

AdminController

Метод GET, /api/admin/all-users. Використовується для отримання списку всіх користувачів, номер сторінки передається у URL як параметр

```
    @GetMapping("all-users")
public Iterable<User> allUsers(@RequestParam(name = "p") int page) {
    return userService.getAllUsers(page);
}
```

Метод POST, /api/admin//users/set-admin/{id}. Використовується для надання прав адміністратора користувачеві, id користувача передається як path параметр

```
@PostMapping("/users/set-admin/{id}")
public void setAdmin(@PathVariable Long id) {
    userService.addAdminRole(id);
```

Метод POST, /api/admin/users/remove-admin/{id}. Використовується для відкликання прав адміністратора у користувача, id користувача передається як URL параметр

```
@PostMapping("/users/remove-admin/{id}")
public void removeAdmin(@PathVariable Long id) {
    userService.removeAdminRole(id);
```

Метод GET, /api/admin//all-reservations. Використовується для отримання всіх резервувань користувачів, номер сторінки передається як URL параметр

```
@GetMapping("/all-reservations")
public Iterable<TourReservation> getAllReservations(@RequestParam(name = "p")
int page) {
    return reservationService.getAllReservations(page);
}
```

AuthController

Метод POST, /api/auth/sign-up. Використовується для реєстрації нових користувачів, приймає дані у тілі запиту у форматі JSON (об'єкт класу User)

```
@PostMapping("/sign-up")
public void signUp(@Valid @RequestBody User user) throws
UserRegistrationFailedException {
```

```
        user.setPassword(passwordEncoder.encode(user.getPassword()));
    userService.createUser(user);
}
```

Метод POST, /api/auth/is-registered. Використовується для перевірки чи користувач існує за допомогою імені, ім'я передається у тілі запиту у форматі JSON

```
@PostMapping("/is-registered")
public Map<String, Boolean> checkUsername(@RequestBody Map<String, String> payload) {
    return Collections.singletonMap("isRegistered", userService.isRegistered(payload.get("username")));
}
```

Метод POST, /api/auth/is-email-exist. Використовується для перевірки чи емейл користувача існує, емейл передається у тілі запиту у форматі JSON

```
@PostMapping("/is-email-exist")
public Map<String, Boolean> checkEmail(@RequestBody Map<String, String> payload) {
    return Collections.singletonMap("isExist", userService.isEmailExist(payload.get("email")));
}
```

Метод POST, /api/auth/change-password. Метод для зміни паролю, приймає у тілі запиту старий і новий пароль у форматі JSON

```
@PostMapping("/change-password")
public void changePassword(@RequestBody Map<String, String> passwordData)
throws AccessDeniedException {
    userService.changePassword(
            passwordData.get("oldPassword"),
            passwordEncoder.encode(passwordData.get("newPassword")));
}
```

Метод POST, /api/auth/update-user-data. Метод для оновлення даних облікового запису користувача, приймає набір даних у тілі запиту, у форматі JSON

```
@PostMapping("/update-user-data")
public void updateUserData(@RequestBody Map<String, String> newUserData) {
    userService.updateUserData(
            newUserData.get("username"),
            newUserData.get("email"),
            newUserData.get("fullName"),
```

```
            newUserData.get("mobileNumber"));
}
```

Метод GET, /api/auth/user-data. Метод дозволяє отримати дані облікового запису користувача

```
        @GetMapping("/user-data")
    public User getUserData(Principal principal) {
        return (User) userService.loadUserByUsername(principal.getName());
    }
}
```

HotelController

Метод GET, /api/hotel. Метод дозволяє отримати список всіх готелів, приймає номер сторінки як URL параметр

```
@GetMapping
public Page<Hotel> getAllHotels(@RequestParam(name = "p") int page) {
    return hotelService.getAllHotels(page);
}
```

Метод GET, /api/hotel/{id}. Метод повертає готель по id, яке передається як URL параметр

```
@GetMapping("/{id}")
public Hotel getHotelById(@PathVariable Long id) throws NotFoundException {
    return hotelService.getHotelById(id);
}
```

Метод PUT, /api/hotel. Метод дозволяє оновити дані готелю, приймає готель у тілі запиту, у форматі JSON

```
@PutMapping
public void updateHotel(@Valid @RequestBody Hotel hotel) throws
NotFoundException {
    hotelService.updateHotel(hotel);
}
```

Метод DELETE, /api/hotel/{id}. Метод дозволяє видалити готель по ID яке передається як path параметр у URL

```
@DeleteMapping("/{id}")
public void deleteHotel(@PathVariable Long id) throws NotFoundException,
DeletionException, FileNotFoundException {
    hotelService.deleteHotel(id);
}
```

Метод POST, /api/hotel. Використовується для додавання нового готелю, приймає об'єкт готелю у тілі запиту у форматі JSON, а також файл (фото)

```
@PostMapping
public Map<String, Long> addHotel(
        @Valid @RequestPart(name = "hotel") Hotel hotel,
        @RequestPart(name = "image")MultipartFile file) throws IOException {

    return Collections.singletonMap("hotelId", hotelService.addHotel(hotel,
file));
}
```

ReservationController

Метод POST, /api/reservations. Використовується для створення нового резервування, об'єкт отримується з тіла запиту у форматі JSON

```
@PostMapping
public void createReservation(@Valid @RequestBody TourReservation
reservation, Principal principal) throws NotFoundException,
SeatsCountOverflowException {
    reservationService.addReservation(reservation, principal);
}
```

Метод GET, /api/reservations. Використовується для отримання всіх резервувань користувача, номер сторінки для виводу передається як URL параметр

```
@GetMapping
public Page<TourReservation> getAllUserReservations(@RequestParam(name = "p")
int page, Principal principal) {
    return reservationService.getAllUserReservations(page, principal);
}
```

Метод POST, /api/reservations/cancel/{id}. Використовується для скасування резервування, id резервування передається як path параметр у URL

```
@PostMapping("/cancel/{id}")
public void cancelReservation(@PathVariable Long id) throws NotFoundException
{
    reservationService.cancelReservation(id);
}
```

TourController

Метод GET, /api/tours. Використовується для отримання списку всіх турів, номер сторінки передається як URL параметр

```
@GetMapping
public Page<Tour> getAllTours(@RequestParam(name = "p") int page) {
    return tourService.getAllTours(page);
}
```

Метод DELETE, /api/tours/{id}. Використовується для видалення туру, id туру, який потрібно видалити, передається як path параметр у URL

```
@DeleteMapping("/{id}")
public void deleteTour(@PathVariable Long id) throws
ReservationsExistException, DeletionException, FileNotFoundException {
    tourService.removeTour(id);
}
```

Метод PUT, /api/tours. Використовується для оновлення даних туру, об'єкт з новими даними передається у тілі запиту у форматі JSON

```
@PutMapping
public void updateTour(@Valid @RequestBody Tour tour) throws
NotFoundException {
    tourService.updateTour(tour);
}
```

Метод POST, /api/tours. Використовуєть для додавання туру, сформований об'єкт передається у тілі запиту у форматі JSON, також у тілі запиту передається файл (фото) туру

```
@PostMapping
public void createTour(
        @Valid @RequestPart(name = "tour") Tour tour,
        @RequestPart(name = "image") MultipartFile file) throws
NotFoundException, IOException {
    tourService.createNewTour(tour, file);
}
```

3 BACK-END ЧАСТИНА ПРОЕКТУ

3.1 Класи проекту

В даному проекті були створені каталоги (рис 3.1), в яких знаходяться відповідні класи і відповідають за наступне:

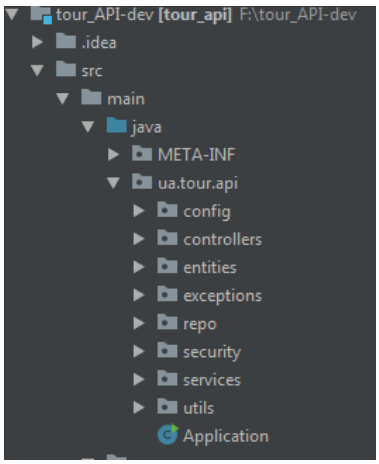

Рисунок 3.1 – Модель проекту

– Config – відповідає за конфігурацію API (рис 3.2);

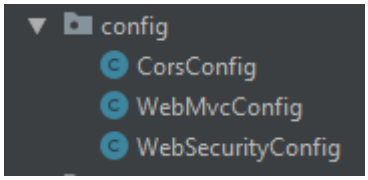

Рисунок 3.2 – Каталог config

– Controller – відповідає за контролери, які пов’язані з базою даних (рис.3.3);

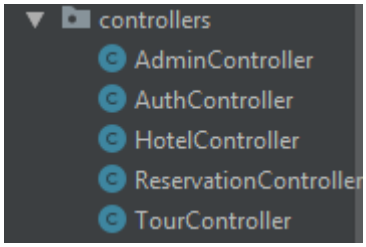

Рисунок 3.3 – Каталог Controller

– Entities – відповідає за побудову моделей (рис.3.4) ;

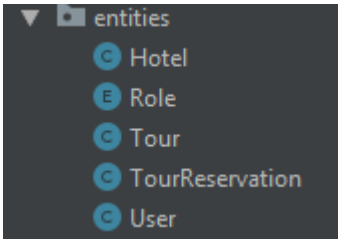

Рисунок 3.4 – Каталог Entities

– Exception – відповідає за перевірку на помилки (рис. 3.4);

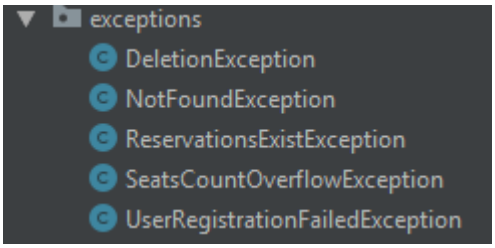

Рисунок 3.4 – Каталог Entities

– Security – відповідає за безпечне з’єднання та передачу данних між API та сайтом (рис. 3.5);

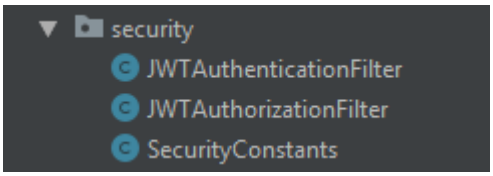

Рисунок 3.5 – Каталог Entities

3.2 Клас HotelService

Даний метод створений для додавання готелю, приймає екземпляр класу Hotel і файл (фотографію), також перевіряє чи готель з таким іменем вже існує. Якщо такого готелю немає отриманому вище екземпляру вставляється id із значенням null, для того, щоб уникнути перезапису. За

допомогою imageService зберігається отримане зображення (у папку на диску), після цього назва зображення записується у екземпляр класу готелю. Після вставки всіх даних, готель зберігається у базу даних. А у випадку, якщо готель з таким іменем вже існує, повертається виключення з відповідним повідомленням. Якщо не виникло ніяких помилок, метод повертає id доданого готелю. Приклад методу наведений нижче:

```
        public long addHotel(Hotel hotel, MultipartFile file) throws
IOException {

            if (!hotelRepository.existsByName(hotel.getName())) {
                hotel.setId(null);
                hotel.setImageSrc(imageService.saveImage(file));
                hotelRepository.save(hotel);
            } else throw new IllegalArgumentException("Hotel with name: " +
hotel.getName() + " already exists");
            //
            return hotel.getId();
        }
```

Метод для видалення готелю, приймає id готелю, який потрібно видалити. Спочатку відбувається спроба отримати готель з бази даних по id і перевірка чи готель, який потрібно видалити, існує. У випадку якщо готель існує, проводиться перевірка чи цей готель присутній у якомусь турі (якщо так, тоді його не можна видаляти). Далі відбувається перевірка чи тур існує і якщо тур існує, і даний готель використовується, повертається виключення з відповідним повідомленням але якщо готель не використовується, спочатку видаляється зображення. Після видалення зображення, готель видаляється з бази даних. А якщо готель, який потрібно видалити, відсутній - повертається виключення з повідомленням

Приклад наведений нижче:

```
        public void deleteHotel(Long hotelId) throws NotFoundException,
DeletionException, FileNotFoundException {
            Optional<Hotel> opHotel = hotelRepository.findById(hotelId);
            if (opHotel.isPresent()) {
                Optional<Tour> opTour =
tourRepository.findFirstByHotel(opHotel.get());
                if (opTour.isPresent()) {
```

```
                        throw new DeletionException("Unable to delete hotel
   because there are tours to this hotel.");
                    } else {

   imageService.deleteImage(opHotel.get().getImageSrc());
                        hotelRepository.deleteById(hotelId);
                    }
                } else throw new NotFoundException("Unable to delete non-
   existent hotel.");
        }
```

Наступний метод був створений для редагування даних готелю, він приймає екземпляр класу готелю, з даними, які потрібно оновити далі відбувається перевірка чи готель, який буде редагуватись, існує та перевірка на те, чи ім'я відредагованого готелю не співпадає з вже існуючими, окрім поточного. Якщо ім'я не співпадає ні з одним з інших готелів - зберігає відредагований готель. Виключення повертається, якщо ім’я відредагованого готелю співало з вже існуючим також виключення повертається, якщо готель,

який потрібно відредагувати, не існує.

```
    public void updateHotel(Hotel hotel) throws NotFoundException {

        if (hotelRepository.existsById(hotel.getId())) {
            if (hotelRepository.countByNameAndNotId(hotel.getName(),
hotel.getId()) == 0) {
                                hotelRepository.save(hotel);

            } else throw new IllegalArgumentException("Cannot update hotel.
Hotel with name: " + hotel.getName() + "already exists");
                } else throw new NotFoundException("Unable to update non-
existent hotel with id: " + hotel.getId());

    }
}
```

3.3 Клас ReservationService

Метод для бронювання місця, приймає сформований екземпляр класу TourReservation.

```
public void addReservation(TourReservation reservation, Principal
principal) throws SeatsCountOverflowException, NotFoundException {
```

Спроба знайти у базі даних тур, місце в якому, потрібно зарезервувати.

```
Optional<Tour> opTour =
tourRepository.findById(reservation.getTour().getId());
```

Перевірка чи тур, місце в якому потрібно забронювати, існує

```
if (opTour.isPresent()) {
```

Якщо тур існує, посилання на нього передається у новий екземпляр класу.

```
Tour = opTour.get();
```

Підрахунок всіх бронювань на даний тур у базі даних

```
Long reservationCount = tourReservationRepository.countByTour(tour);
```

Перевірка чи місце у турі можна зарезервувати (чи кількість можливих резервувань не максимальна)

```
if (reservationCount < tour.getSeatCount()) {
```

Отримання поточного користувача з бази даних, за допомогою об'єкту principal та імені користувача

```
User currentUser = (User)
userService.loadUserByUsername(principal.getName());
```

Перевірка чи користувач існує

```
if (currentUser != null) {
```

Встановлення id резервування у значення null щоб уникнути перезапису вже існуючих даних

```
reservation.setId(null);
```

Встанновлення активності резервування у значення true

```
reservation.setActive(true);
```

Присвоєння посилання на користувача у резервуванні

```
reservation.setUser(currentUser);
```

Збереження резервування у базу даних

```
tourReservationRepository.save(reservation);
}
```

Виключення повертається якщо перевищена кількість доступних місць у турі

```
            } else throw new SeatsCountOverflowException("Seats for the selected tour have expired");
```

Виключення повертається, якщо проходить спроба зарезервувати місце у неіснуючому турі

```
        } else throw new NotFoundException("Tour with id: " + reservation.getTour().getId() + " not found");

}
```

Метод для виводу списку всіх резервувань (по сторінці) в метод передається номер сторінки та об'єкт Pricipal.

```
public Page<TourReservation> getAllUserReservations(int page, Principal principal) throws UsernameNotFoundException {
```

Спроба завантажити поточного користувача з бази даних.

```
        User currentUser = (User) userService.loadUserByUsername(principal.getName());
```

Перевірка чи користувач існує

```
    if (currentUser != null) {
```

Якщо користувач існує, метод повертає список резервувань користувача, з вказаною сторінкою.

```
            return tourReservationRepository.findAllByUser(currentUser, PageRequest.of(page, 15));
```

Якщо користувача не буде знайдено – буде повернено відповідне повідомлення

```
        } else throw new UsernameNotFoundException(«User with username « + principal.getName() + « not found»);
}
```

Наступний метод відповідає за скасування бронювання, метод приймає id бронювання, яке потрібно скасувати

```
    public void cancelReservation(Long reservationId) throws NotFoundException {
```

Спроба отримати резервування з бази даних

```
        Optional<TourReservation> opReservation =
tourReservationRepository.findById(reservationId);
```

Перевірка чи резервування існує

```
        if (opReservation.isPresent()) {
```

Присвоєння посилання на об'єкт новому екземпляру класу

```
            TourReservation reservation = opReservation.get();
```

Встановлення статусу активності резервування у значення false

```
            reservation.setActive(false);
```

Оновлення даних у бд

```
            tourReservationRepository.save(reservation);
```

Виключення повертається, в разі відсутності резервування у бд

```
        } else throw new NotFoundException("Reservation with id: " +
reservationId + " not found");
    }
```

Вивід всіх резервувань (для адміністратора), приймає номер сторінки в якості параметра

```
    public Page<TourReservation> getAllReservations(int page) {
        return tourReservationRepository.findAll(PageRequest.of(page, 16));
    }
}
```

3.4 Опис Функціональної частини

Для наочного прикладу розглянемо роботу API використовуючи середовище Postman для перевірки роботи. Перше що нам потрібно це встановити Postman на нашу ОС і імпортувати у нього проект, після чого настроїти API і токени (рис. 3.7):

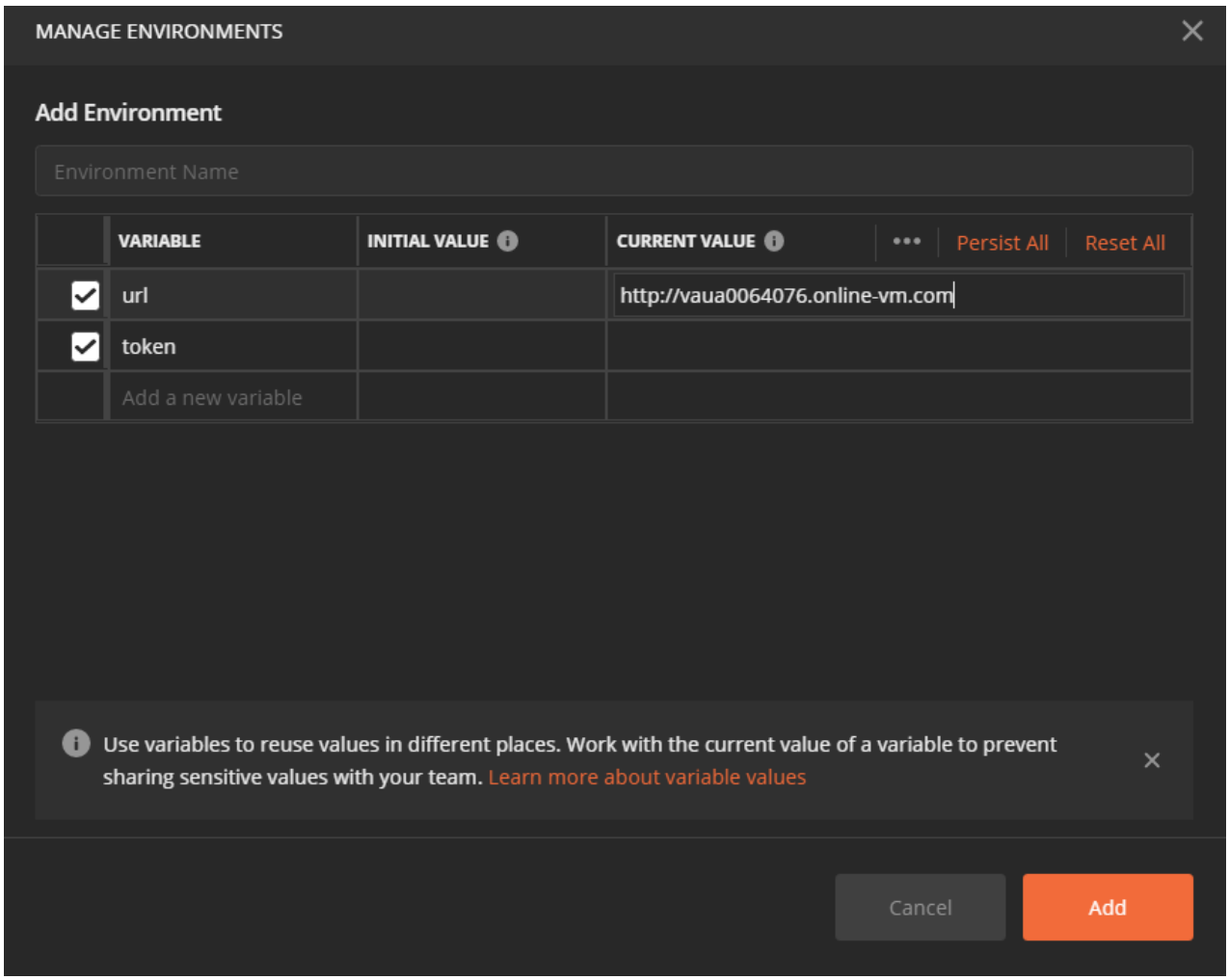

Рисунок 3.7 – Настройка підключення

Но для коректної роботи і можливості працювати необхідно отримати токен, без якого наш проект не буде працювати. Що б отримати даний токен необхідно зайти в папку користувачі до запиту вхід і зареєструвати (рис. 3.8)користувача:

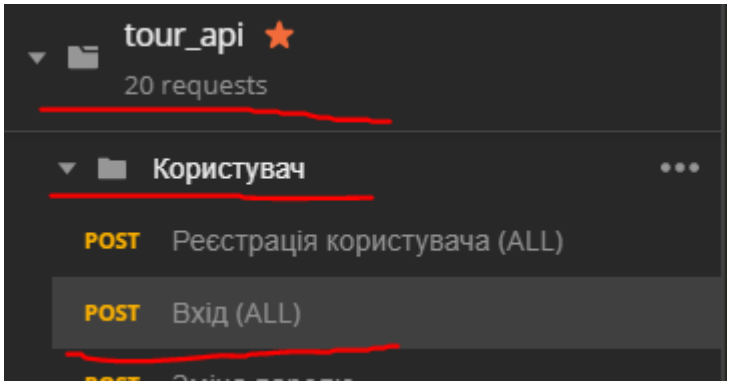

Рисунок 3.8 – Реєстрація користувача

Після чого ми переходимо у вкладку body(Рис. 3.9) і реєструємо користувача з наступними даними:

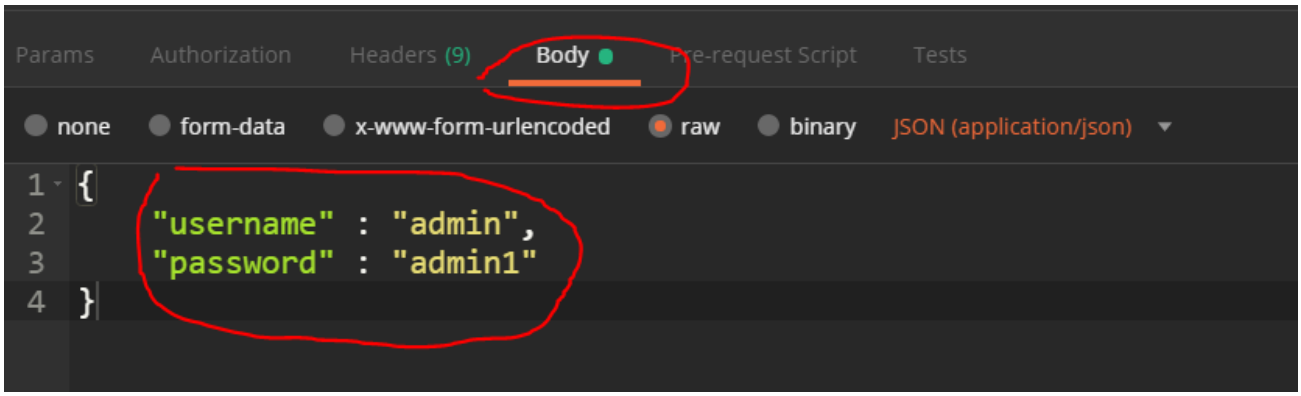

Рисунок 3.9 – Данні користувача

Після чого необхідно натиснути кнопку Send, запит повинен завершитись (рис. 3.10) з HTTP кодом 200,після чого перейти у вкладку headers і скопіювати рядок який починається зі слова Bearer, і до кінця рядку

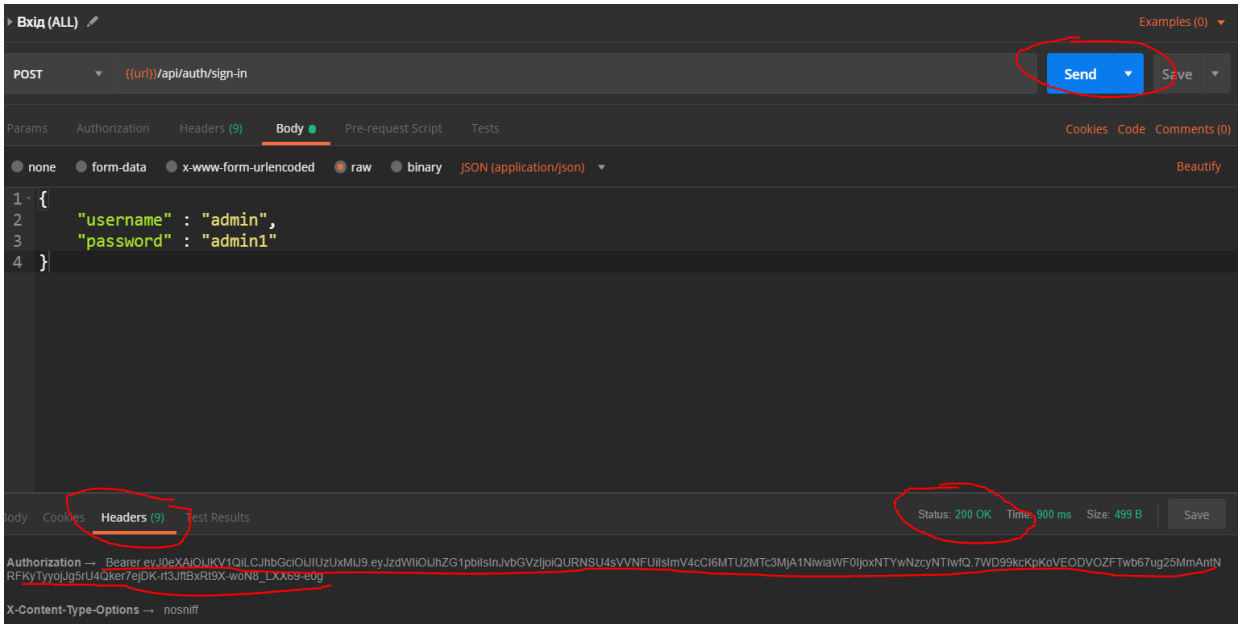

Рисунок 3.10 – Отримання токена

Перейти в налаштування середовища і вставити у поле token (рис. 3.11) у колонку INITIAL VALUE і вставити в поле token значення, яке ми отримали:

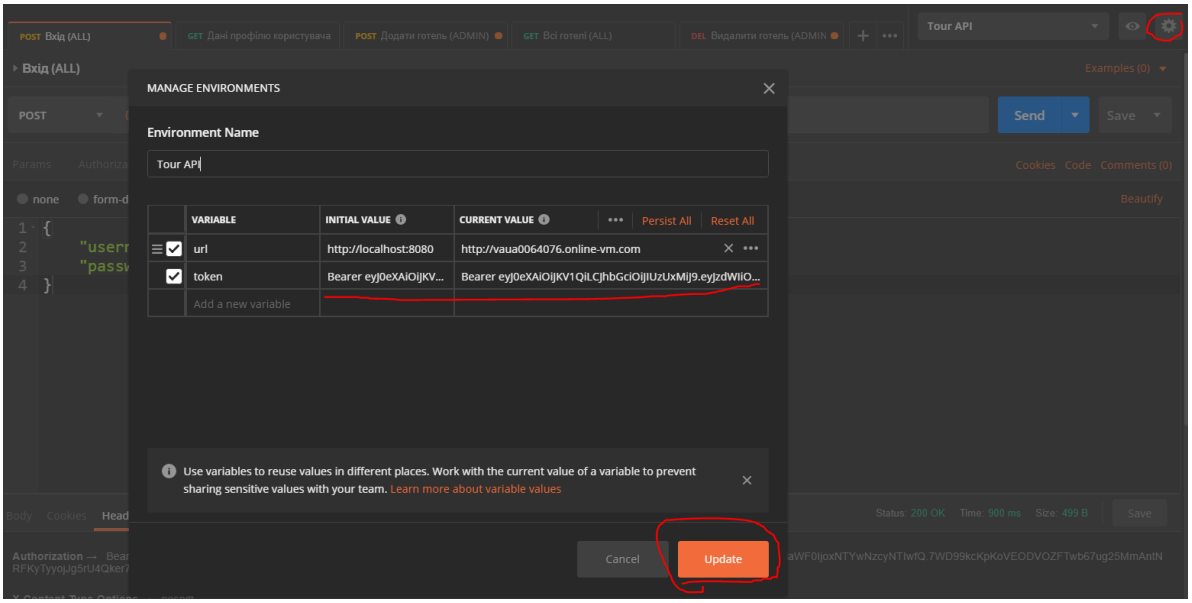

Рисунок 3.11 – Запис токена

Після чого для перевірки можна спробувати додати готель (потрібно вибрати відповідний запит, і у вкладці body вибрати іншу картинку з диску комп'ютера).

Якщо все зроблено правильно статус запиту буде з кодом 200 (рис. 3.12) і готель можна буде побачити у списку готелів.

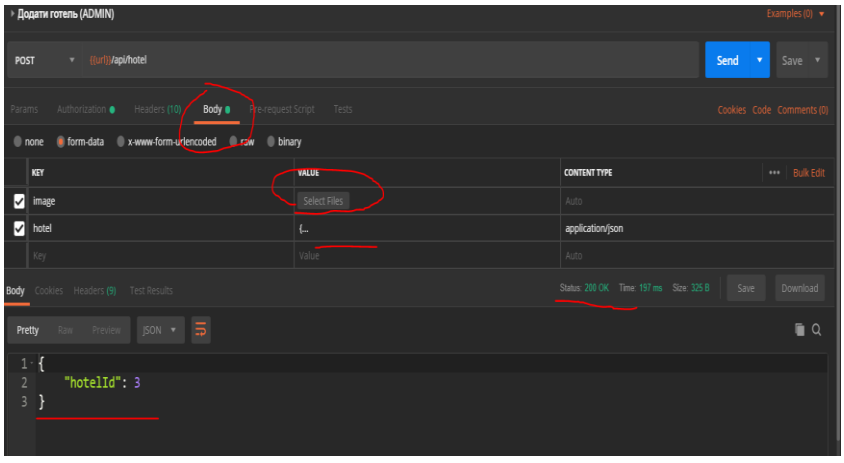

Рисунок 3.12 – Запит з готелями

Далі, для демонстрації роботи функціоналу API допустимо, що ми забули свій пароль і нам необхідно змінити його. Для цього використовується функція зміни паролю (рис. 3.13), яка має наступний вигляд:

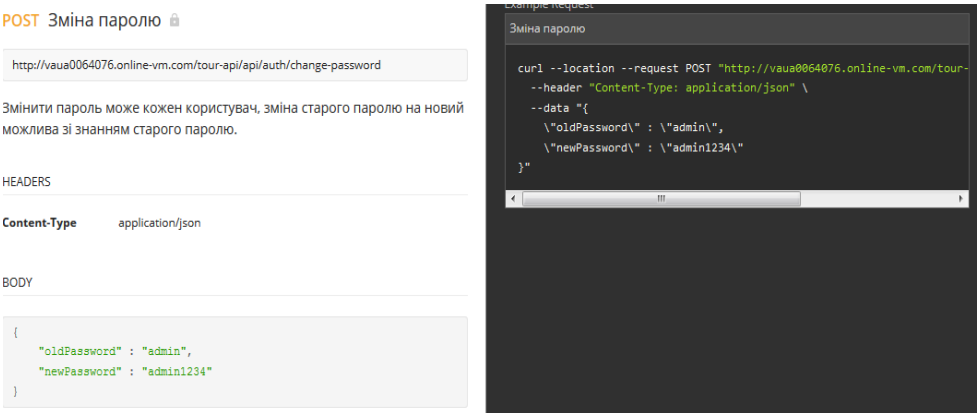

Рисунок 3.13 – Запит на зміну паролю

Також функціоналом розробленого API є зміна даних профілю (рис. 3.14). Для цього використовується POST зміни даних користувача. Якщо було змінено ім’я користувача, то для підтвердження потрібно увійти за допомогою нового імені користувача

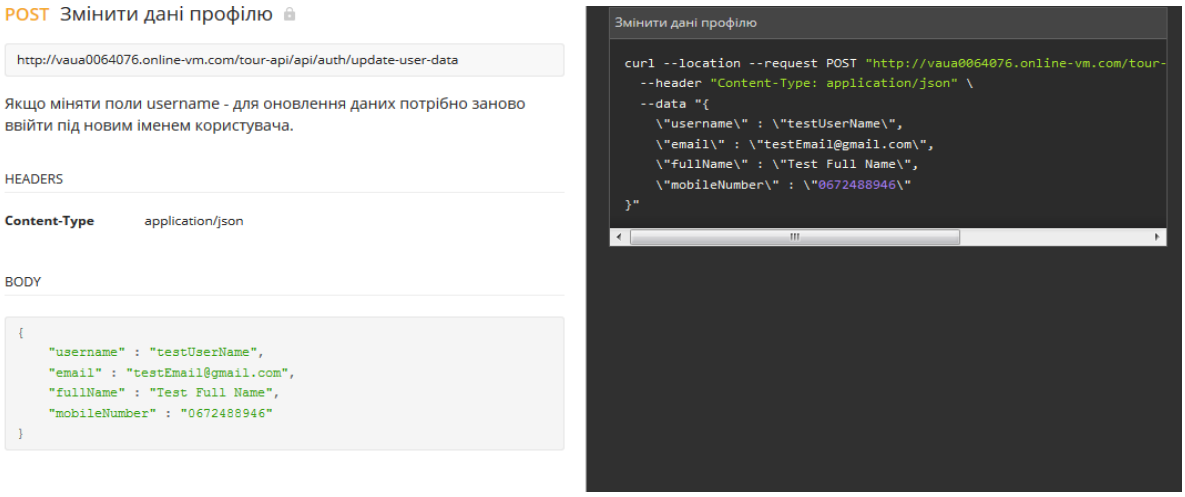

Рисунок 3.14 – Запит на даних користувача

В даному API також реалізована функція на додавання турів (рис. 3.15), яку може виконувати тільки модератор сайту. В даній функції є такі поля як назва самого туру, короткий опис, дата початку і кінця туру, а також фотографія:

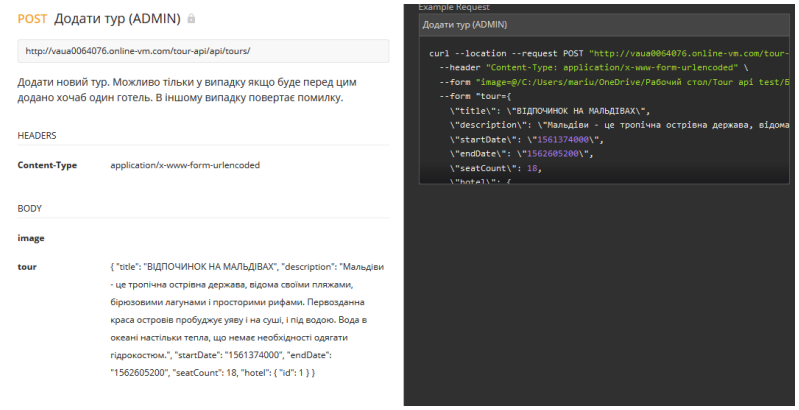

Рисунок 3.15 – Додавання туру

Також до кожного туру є підв’язка готелів. Під’вязка виконується по ID готелю з використанням зв’язку «один до багатьох» у БД, оскільки в одному турі може бути запропоновано декілька готелів, в яких може поселитись замовник туру.

Однією із важливих функцій, яка має бути передбаченою в API такого типу це необхідність оновити час туру або інші його данні з деяких причин. Однак в даному API не можна змінити зображення туру. Приклад зміни наведений нижче (рис. 3.16):

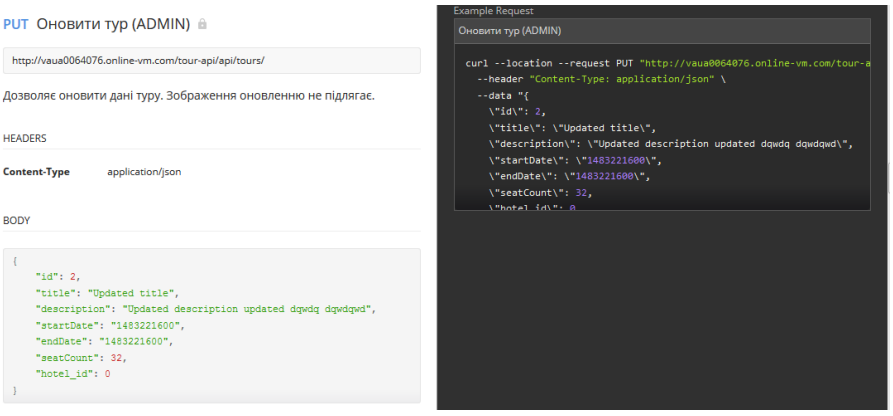

Рисунок 3.16 – Зміна туру

Що до готелів і функціоналу, яке передбачає наше API – це:

– додавання готелю (Адміністратор);
– перегляд всіх готелів (Всі користувачі);
– видалення готелів (Адміністратор);
– оновлення готелів (Адміністратор).

Функція додавання готелю (рис. 3.17) передбачає додавання назви готелю, короткого опису і місця розташування (країна, місто):

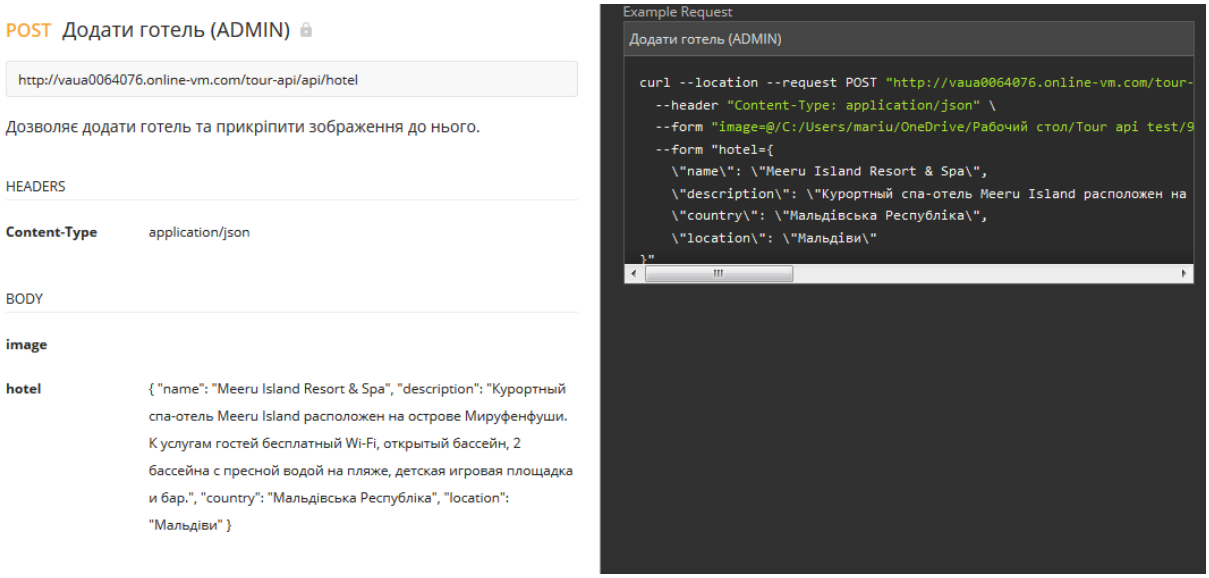

Рисунок 3.17 – Додавання даних про готель

Перегляд як і видалення (рис. 3.18) виконується за допомогою запитів, реалізованих на JSON:

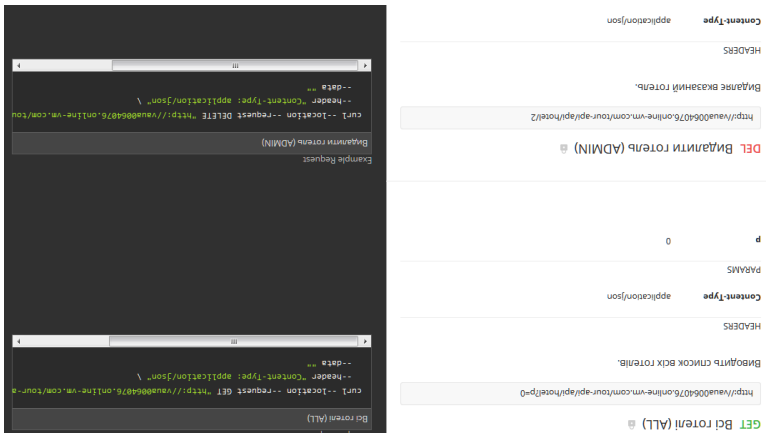

Рисунок 3.18 – Перегляд та видалення даних про готелі

Для редагування даних про готель в API використовуємо функцію для редагування (рис. 3.19). Пошук готелю, який потрібно редагувати відбувається по полі ID готелю:

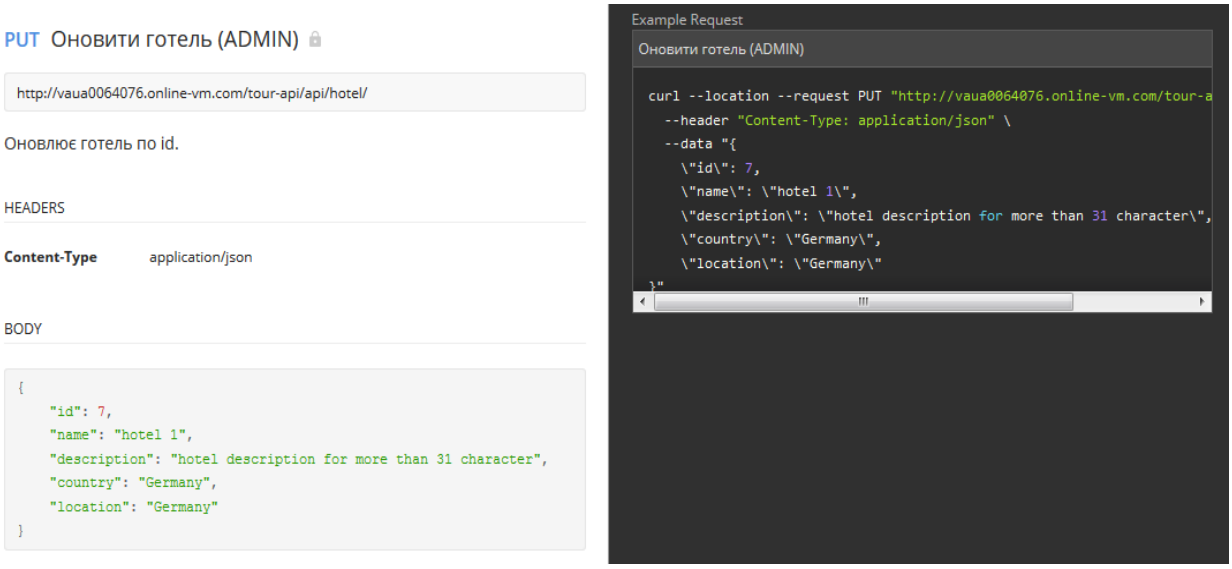

Рисунок 3.19 – Оновлення даних про готель

Наступною частиною функціоналу API є резервування турів. Реалізовано наступні функції резервування:

– заразервувати тур;
– всі резервації;
– скасувати резервацію.

Зарезервувати тур може звичайний користувач. Для цього необхідно вибрати тур (рис. 3.20), який цікавить:

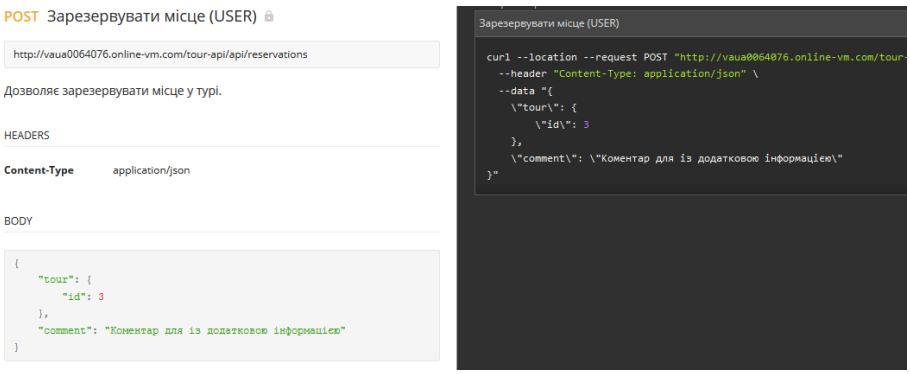

Рисунок 3.20 – Резервація туру

Перегляд усіх резервованих турів здійснюється модератором, як і скасування (рис. 3.21). Скасування реалізовано через вибір туру і користувача який вибрав даний тур.

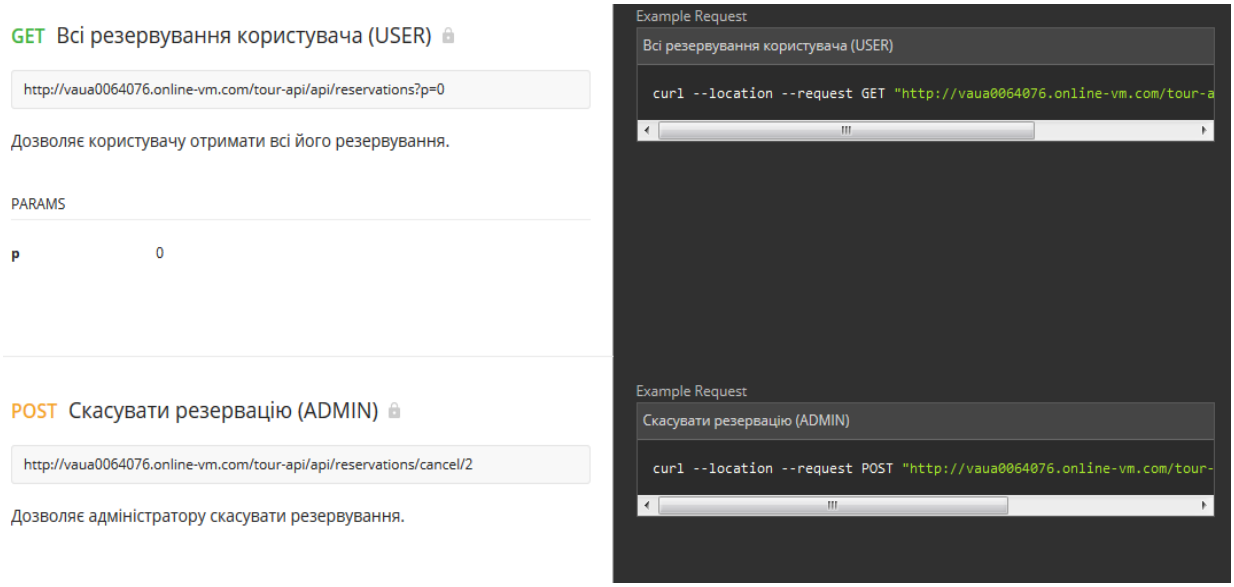

Рисунок 3.21 – Перегляд і скасування резервації туру

Наступним блоком функціоналу API є функції адміністратора. До функціоналу адміністратора відноситься:

– перегляд всіх резервувань всіх користувачів;
– перегляд усіх користувачів;
– надання прав адміністратора;
– видалення прав адміністратора.

Перегляд всіх резервувань (рис 3.22) відбувається по даті резервування.

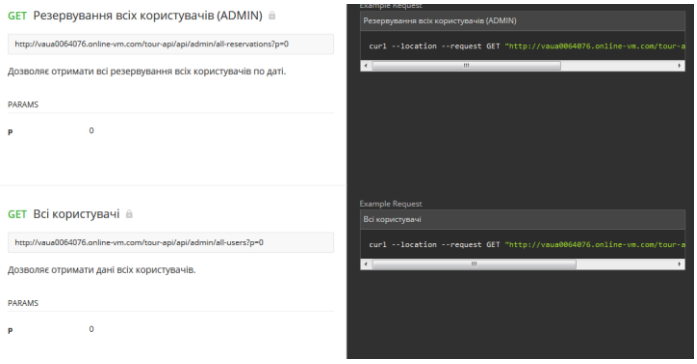

Рисунок 3.22 – Перегляд всіх резервувань і користувачів

Надання і забирання прав адміністратора (рис. 3.23) відбувається через ID користувача:

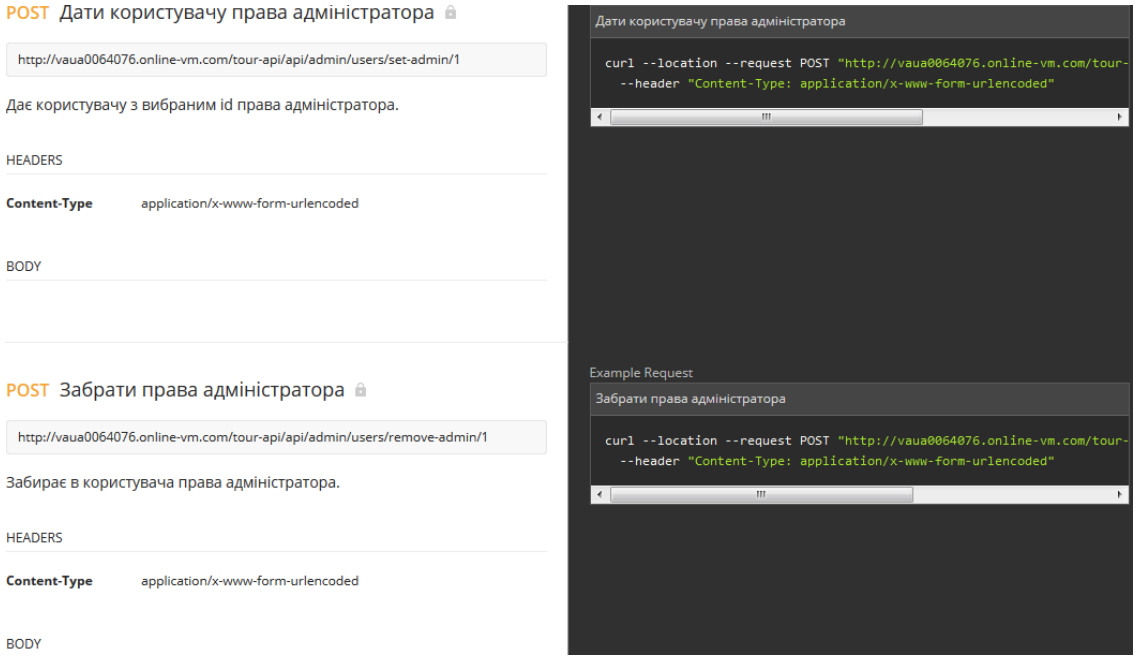

Рисунок 3.23 – Права адміністратора

ВИСНОВКИ

Під час виконання дипломного проекту було спроектовано та розроблено API- back end частини сайту туристичних путівок реалізований за допомогою мови програмування Java та фреймворку Spring, який сильно спрощує та розширює функціонал мови, в якості бази даних використовувалась MySQL з одноіменною СУБД.

Тестування API проводилось в декілька етапів, та відбувалось від створення перших функціонуючих алгоритмів до повного написання та відладки.

Підчас написання кожної функції, вона тестувалась на правильність роботи. Після написання функціональної частини (наприклад контролеру авторизації та автентифікації) проводилось тестування всіх методів які входять до даної частини. Наступним проходом по всім функціях було тестування API в цілому, на відповідність до поставлених задач і коректній роботі.

Дипломний проект виконано у відповідності до завдання і всіма вимогами. Розроблене API дозволяє виконувати бек-енд роботу всіх функції типового сайту з туристичними путівками.

ПЕРЕЛІК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Туристичні послуги. Поняття туристичного продукту. Матеріал з Студопедії. URL: https://studopedia.org/5-117541.html (дата звернення: 25.11.2018).
2. Як працює турагентство. Основні поняття.. Матеріал з Livejournal URL: https://madam-shazly.livejournal.com/69616.html (дата звернення:28.11.2018).
3. MySQL. Матеріал з Вікіпедії — вільної енциклопедії URL: https://uk.wikipedia.org/wiki/MySQL (Дата звернення: 05.12.2018).
4. Глушаков С. Программирование на Java 2. Харьков: Фолио, 2003. 536 с
5. Герберт Шилдт.Java. Полное руководство. Java SE 7 - Java 7: The Complete Reference. 8-е изд. М.: Вильямс, 2012. 1104 с.
6. Барри БердПрограммирование на Java для чайников, 3-е издание Beginning Programming with Java For Dummies, 3rd Edition. М.:«Диалектика», 2013. 384 с.
7. Аллен Дж. Тейлор.SQL для чайников SQL for Dummies. 7-е изд. М.:Диалектика, 2010. 416 с.
8. Соломон М.К., Oracle. Программированиенаязыке Java. М.К. Соломон, Н. Мориссо-Леруа, Д. Басу. М.: Лори, 2010. 512 с.
9. Johnson R. Professional Java Development with the Spring Framework / R. Johnson, J. Hoeller, A. Arendsen, et al. – Wrox Press, 2005. 343 с.
10. Мафтик С.М. Механизмы защиты в сетях ЭВМ. М.: Мир, 1993. 256 с.
11. Гаффин А. Руководство по глобальной компьютерной сети Internet. -Network World, 1998. 500 с.
12. Капор М. Ява для всех. - Санкт-Петербург, 1997. 200 с.

13. Мильвидский А. М. Введение в Java. 1998. 250 с.

14. Монахов Вадим Мова програмування Java і середовище NetBeans. - СПб.: «БХВ-Петербург», 2008. С. 640.

15. ru.sun.com/java/j2ee/book/platform_Java.pdf - Платформа Java ™. Офіційне видання. Дуглас Крамер (Douglas Kramer)

16. Джамса Кріс. JAVA. Бібліотека програміста Попурі. - Мінськ, 1996.

17. Мейнджер Джейсон. JAVA: Основи програмування / Пер. з англ. С. Бойко під ред. Я. Шмідского. К.: BNV, 1997.

18. Девід Фленаган Java в прикладах - довідник

19. Х.м. Дейтел, П.Дж. Дейтел, С.І. Сантрі - Технології програмування на Java

20. Холл М. Сервлеты и JavaServer Pages. Библиотека програмиста. - СПб.:Питер, 2001. 496с.

21. JavaServer Pages™ Specification version 2.0. - Sun Microsystems, 2003. 765ст.

22. Грофф Дж., Вайнберг П. SQL: Полное руководство: Пер.с англ. - К BHV, 2001. 816с.

23. Вебер Д. Технология Java™ в подлиннике: Пер. с англ. - СПб.: BHV, 2000. 1104с.

24. Эккель Б. Философия Java. Библиотека програмиста. - СПб: Питер, 2001. 880с.

25. Кей Хорстманн «Java 2. Тонкости программирования. Том 2» Киев, 2008 г. 469с

26. Патрик Ноутон «Java 2. Наиболее полное руководство» Киев, 2010 г. 796с.

ДОДАТОК А

Клас CorsConfig

```
package ua.tour.api.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.cors.CorsConfiguration;
import org.springframework.web.cors.UrlBasedCorsConfigurationSource;
import org.springframework.web.filter.CorsFilter;

import java.util.Arrays;

@Configuration
public class CorsConfig {

    @Bean
    public CorsFilter corsFilter() {
        UrlBasedCorsConfigurationSource source = new
UrlBasedCorsConfigurationSource();
        CorsConfiguration config = new CorsConfiguration();
        config.setAllowCredentials(true);
        config.addAllowedOrigin("*");
        config.addAllowedHeader("*");
        config.addAllowedMethod("OPTIONS");
        config.addAllowedMethod("GET");
        config.addAllowedMethod("POST");
        config.addAllowedMethod("PUT");
        config.addAllowedMethod("DELETE");
        config.setExposedHeaders(Arrays.asList("Access-Control-Allow-Headers",
"Authorization, x-xsrf-token, Access-Control-Allow-Headers, Origin, Accept, X-
Requested-With, " +
                "Content-Type, Access-Control-Request-Method, Access-Control-
Request-Headers"));
        source.registerCorsConfiguration("/**", config);
        return new CorsFilter(source);
    }
}
```

Клас WebMvcConfig

```
package ua.tour.api.config;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.ResourceHandlerRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

@Configuration
public class WebMvcConfig implements WebMvcConfigurer {

    @Value(("${images.upload.path}"))
    private String uploadPath;

    @Override
```

```
    public void addResourceHandlers(ResourceHandlerRegistry registry) {
        registry.addResourceHandler("/img/**")
                .addResourceLocations("file:///"+ uploadPath + "/");
    }
}
```

Клас WebSecurityConfig

```
package ua.tour.api.config;


import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.http.HttpMethod;
import
org.springframework.security.config.annotation.authentication.builders.Authentica
tionManagerBuilder;
import
org.springframework.security.config.annotation.method.configuration.EnableGlobalM
ethodSecurity;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.configuration.EnableWebSecurit
y;
import
org.springframework.security.config.annotation.web.configuration.WebSecurityConfi
gurerAdapter;
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.web.cors.CorsConfiguration;
import org.springframework.web.cors.CorsConfigurationSource;
import org.springframework.web.cors.UrlBasedCorsConfigurationSource;
import ua.tour.api.entities.Role;
import ua.tour.api.security.JWTAuthenticationFilter;
import ua.tour.api.security.JWTAuthorizationFilter;
import ua.tour.api.services.UserService;

import static ua.tour.api.security.SecurityConstants.SIGN_IN_URL;

@Configuration
@EnableWebSecurity
@EnableGlobalMethodSecurity(prePostEnabled = true)
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

    private final UserService userService;

    @Autowired
    public WebSecurityConfig(UserService userService) {
        this.userService = userService;
    }

    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder(11);
    }

    private JWTAuthenticationFilter configuredJwtAuthenticationFilter() throws
Exception {
        JWTAuthenticationFilter filter = new
```

```
JWTAuthenticationFilter(authenticationManager());
        filter.setFilterProcessesUrl(SIGN_IN_URL);
        return filter;
    }

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
                    .cors()
                .and()
                    .csrf().disable()
                    .authorizeRequests()
                        .antMatchers("/api/auth/**").permitAll()

.antMatchers(HttpMethod.GET,"/api/reservations/*").hasAuthority(Role.USER.getAuth
ority())
                        .antMatchers(HttpMethod.GET, "/img/**",
"/api/tours/*","/api/hotel", "/api/hotel/*").permitAll()

.antMatchers(HttpMethod.POST,"/api/reservations").hasAuthority(Role.USER.getAutho
rity())

.antMatchers(HttpMethod.POST,"/api/reservations/cancel/{id}").hasAuthority(Role.A
DMIN.getAuthority())
                        .antMatchers("/api/tours/**", "/api/hotel/**",
"/api/reservations/**", "/api/admin/**").hasAuthority(Role.ADMIN.getAuthority())
                        .anyRequest().authenticated()
                .and()
                    .addFilter(new
JWTAuthorizationFilter(authenticationManager()))
                    .addFilter(configuredJwtAuthenticationFilter())

.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS);

    }


    @Autowired
    public void configAuthentication(AuthenticationManagerBuilder auth) throws
Exception {
        auth.userDetailsService(userService)
                .passwordEncoder(passwordEncoder());

    }

    @Bean
    CorsConfigurationSource corsConfigurationSource() {
        final UrlBasedCorsConfigurationSource source = new
UrlBasedCorsConfigurationSource();
        source.registerCorsConfiguration("/**", new
CorsConfiguration().applyPermitDefaultValues());
        return source;
    }
}
```

Клас HotelService

```
package ua.tour.api.services;

import org.springframework.data.domain.Page;
import org.springframework.data.domain.PageRequest;
```

```
import org.springframework.stereotype.Service;
import org.springframework.web.multipart.MultipartFile;
import ua.tour.api.entities.Hotel;
import ua.tour.api.entities.Tour;
import ua.tour.api.exceptions.DeletionException;
import ua.tour.api.exceptions.NotFoundException;
import ua.tour.api.repo.HotelRepository;
import ua.tour.api.repo.TourRepository;

import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.Optional;

@Service
public class HotelService {

    private HotelRepository hotelRepository;
    private TourRepository tourRepository;
    private ImageService imageService;

    public HotelService(HotelRepository hotelRepository, TourRepository
tourRepository, ImageService imageService) {
        this.hotelRepository = hotelRepository;
        this.tourRepository = tourRepository;
        this.imageService = imageService;
    }

    public Page<Hotel> getAllHotels(int page) {
        return hotelRepository.findAll(PageRequest.of(page, 15));
    }

    public Hotel getHotelById(Long id) throws NotFoundException {
        Optional opHotel = hotelRepository.findById(id);
        if (opHotel.isPresent()) {
            return (Hotel) opHotel.get();
        } else throw new NotFoundException("Hotel with id: " + id + " not
found");
    }

    public long addHotel(Hotel hotel, MultipartFile file) throws IOException {
        hotel.setId(null);
        hotel.setImageSrc(imageService.saveImage(file));
        hotelRepository.save(hotel);
        return hotel.getId();
    }

    public void deleteHotel(Long hotelId) throws NotFoundException,
DeletionException, FileNotFoundException {
        Optional<Hotel> opHotel = hotelRepository.findById(hotelId);
        if (opHotel.isPresent()) {
            Optional<Tour> opTour =
tourRepository.findFirstByHotel(opHotel.get());
            if (opTour.isPresent()) {
                throw new DeletionException("Unable to delete hotel because there
are tours to this hotel.");
            } else {
                imageService.deleteImage(opHotel.get().getImageSrc());
                hotelRepository.deleteById(hotelId);
            }

        } else throw new NotFoundException("Unable to delete non-existent
hotel.");
```

```
    }


    public void updateHotel(Hotel hotel) throws NotFoundException {
        if (hotelRepository.existsById(hotel.getId())) {
            hotelRepository.save(hotel);
        } else throw new NotFoundException("Unable to update non-existent hotel
with id: " + hotel.getId());

    }
}
```

Клас ImageService

```
package ua.tour.api.services;

import org.apache.tomcat.util.http.fileupload.InvalidFileNameException;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Service;
import org.springframework.web.multipart.MultipartFile;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.Arrays;
import java.util.UUID;

@Service
public class ImageService {

    private final String[] allowedExtensions = {".jpg",".png"};

    @Value(("${images.upload.path}"))
    private String uploadPath;

    public String saveImage(MultipartFile file) throws IOException {
        if(file != null) {
            String name = file.getOriginalFilename();
            if (name != null) {
                File uploadFolder = new File(uploadPath);
                boolean isUploadFolderExists = uploadFolder.exists();

                if (!isUploadFolderExists) {
                    isUploadFolderExists = uploadFolder.mkdir();
                }

                boolean isExstensionValid = false;
                String extension = name.substring(name.lastIndexOf("."));
                isExstensionValid = isImageExtensionValid(extension);

                if (isExstensionValid && isUploadFolderExists) {
                    String uuidFile = UUID.randomUUID().toString();
                    String resultFileName = uuidFile + "." +
file.getOriginalFilename();
                    file.transferTo(new File(uploadPath + '/' + resultFileName));
                    return resultFileName;
                } else throw new IOException("File extension is not valid. Valid
file extensions: " + Arrays.toString(allowedExtensions));
            } else throw new InvalidFileNameException(null, "Invalid file name");
        } else throw new FileNotFoundException("Uploaded file not found");
    }
```

```
    public void deleteImage(String fileName) throws FileNotFoundException {
        File file = new File(uploadPath + "/" + fileName);
        if (!file.delete()) throw new FileNotFoundException("file with name " +
fileName + " not found");
    }

    private boolean isImageExtensionValid(String extension) {
        for (String e : allowedExtensions) {
            if (e.equals(extension)) return true;
        }
        return false;
    }
}
```

Клас ReservationService

```
package ua.tour.api.services;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.PageRequest;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;
import ua.tour.api.entities.Tour;
import ua.tour.api.entities.TourReservation;
import ua.tour.api.entities.User;
import ua.tour.api.exceptions.NotFoundException;
import ua.tour.api.exceptions.SeatsCountOverflowException;
import ua.tour.api.repo.TourRepository;
import ua.tour.api.repo.TourReservationRepository;

import java.security.Principal;
import java.util.Optional;

@Service
public class ReservationService {

    private TourReservationRepository tourReservationRepository;
    private TourRepository tourRepository;
    private UserService userService;

    @Autowired
    public ReservationService(
            TourReservationRepository tourReservationRepository,
            UserService userService,
            TourRepository tourRepository) {

        this.tourReservationRepository = tourReservationRepository;
        this.userService = userService;
        this.tourRepository = tourRepository;
    }

    public void addReservation(TourReservation reservation, Principal principal)
throws SeatsCountOverflowException, NotFoundException {
        Optional<Tour> opTour =
tourRepository.findById(reservation.getTour().getId());
        if (opTour.isPresent()) {
            Tour tour = opTour.get();
            Long reservationCount = tourReservationRepository.countByTour(tour);
            if (reservationCount < tour.getSeatCount()) {
```

```
                User currentUser = (User)
userService.loadUserByUsername(principal.getName());
                if (currentUser != null) {
                    reservation.setId(null);
                    reservation.setActive(true);
                    reservation.setUser(currentUser);
                    tourReservationRepository.save(reservation);
                }
            } else throw new SeatsCountOverflowException("Seats for the selected
tour have expired");
        } else throw new NotFoundException("Tour with id: " +
reservation.getTour().getId() + " not found");

    }

    public Page<TourReservation> getAllUserReservations(int page, Principal
principal) throws UsernameNotFoundException {
        User currentUser = (User)
userService.loadUserByUsername(principal.getName());
        if (currentUser != null) {
            return tourReservationRepository.findAllByUser(currentUser,
PageRequest.of(page, 15));
        } else throw new UsernameNotFoundException("User with username " +
principal.getName() + " not found");
    }

    public void cancelReservation(Long reservationId) throws NotFoundException {
        Optional<TourReservation> opReservation =
tourReservationRepository.findById(reservationId);
        if (opReservation.isPresent()) {
            TourReservation reservation = opReservation.get();
            reservation.setActive(false);
            tourReservationRepository.save(reservation);
        } else throw new NotFoundException("Reservation with id: " +
reservationId + " not found");
    }

    public Page<TourReservation> getAllReservations(int page) {
        return tourReservationRepository.findAll(PageRequest.of(page, 16));
    }
}
```

Клас TourService

```
package ua.tour.api.services;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.PageRequest;
import org.springframework.stereotype.Service;
import org.springframework.web.multipart.MultipartFile;
import ua.tour.api.entities.Tour;
import ua.tour.api.exceptions.DeletionException;
import ua.tour.api.exceptions.NotFoundException;
import ua.tour.api.exceptions.ReservationsExistException;
import ua.tour.api.repo.HotelRepository;
import ua.tour.api.repo.TourRepository;
import ua.tour.api.repo.TourReservationRepository;

import java.io.FileNotFoundException;
import java.io.IOException;
```

```
import java.util.Optional;

@Service
public class TourService {

    private final TourRepository tourRepository;
    private final TourReservationRepository reservationRepository;
    private final HotelRepository hotelRepository;
    private final ImageService imageService;

    @Autowired
    public TourService(TourRepository tourRepository, TourReservationRepository
reservationRepository, HotelRepository hotelRepository, ImageService
imageService) {
        this.tourRepository = tourRepository;
        this.reservationRepository = reservationRepository;
        this.hotelRepository = hotelRepository;
        this.imageService = imageService;
    }

    public void createNewTour(Tour tour, MultipartFile file) throws
NotFoundException, IOException {
        boolean isHotelExist =
hotelRepository.existsById(tour.getHotel().getId());
        boolean isTourTitleOrDescriptionExist =
tourRepository.existsByTitleOrDescription(tour.getTitle(),
tour.getDescription());
        if (isHotelExist && !isTourTitleOrDescriptionExist) {
            tour.setId(null);
            tour.setImageSrc(imageService.saveImage(file));
            tourRepository.save(tour);
        } else throw new NotFoundException("Tour can not be created because hotel
with id: " + tour.getHotel().getId() + " not exist.");
    }

    public void removeTour(Long tourId) throws ReservationsExistException,
DeletionException, FileNotFoundException {
        Optional<Tour> opTour = tourRepository.findById(tourId);
        if (opTour.isPresent()) {
            Long reservationsCount =
reservationRepository.countByTour(opTour.get());
            if (reservationsCount == 0) {
                imageService.deleteImage(opTour.get().getImageSrc());
                tourRepository.deleteById(tourId);
            } else throw new ReservationsExistException("The tour can not be
deleted because reservations have been found.");
        } else throw new DeletionException("Unable to delete tour, because it not
exist");
    }


    public Page<Tour> getAllTours(int page) {
        return tourRepository.findAll(PageRequest.of(page, 10));
    }

    public void updateTour(Tour tour) throws NotFoundException {
        if (tourRepository.existsById(tour.getId())) {
            tourRepository.save(tour);
        } else throw new NotFoundException("Can not update tour, because it does
not exist");
    }
}
```

Клас TourService

```
package ua.tour.api.services;

import org.hibernate.HibernateException;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.PageRequest;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;
import ua.tour.api.entities.Role;
import ua.tour.api.entities.User;
import ua.tour.api.exceptions.UserRegistrationFailedException;
import ua.tour.api.repo.UserRepository;

import java.util.Collections;
import java.util.Optional;

@Service
public class UserService implements UserDetailsService {

    private final UserRepository userRepository;


    @Autowired
    public UserService(UserRepository userRepository) {
        this.userRepository = userRepository;
    }

    @Override
    public UserDetails loadUserByUsername(String s) throws
UsernameNotFoundException {
        User applicationUser = userRepository.findByUsername(s);
        if (applicationUser == null) {
            throw new UsernameNotFoundException(s);
        }
        return applicationUser;
    }

    public void createUser(User user) throws UserRegistrationFailedException {
        if (userRepository.findFirstByUsernameOrEmail(user.getUsername(),
user.getEmail()) == null) {
            user.setActive(true);
            user.setRoles(Collections.singleton(Role.USER));
            try {
                user.setId(null);
                userRepository.save(user);
            } catch (HibernateException e) {
                throw new UserRegistrationFailedException("Failed to register new
user.");
            }
        } else throw new UserRegistrationFailedException("Failed to register new
user, because username or email already exist.");

    }

    public Page<User> getAllUsers(int page) {
        return userRepository.findAll(PageRequest.of(page, 15));
```

```
    }

    public void addAdminRole(Long userId) {
        Optional opUser = userRepository.findById(userId);
        if (opUser.isPresent()) {
            User user = (User) opUser.get();
            if (!user.getRoles().contains(Role.ADMIN)) {
                user.addRole(Role.ADMIN);
            }
            userRepository.save(user);
        }
    }

    public void removeAdminRole(Long userId) {
        Optional opUser = userRepository.findById(userId);
        if (opUser.isPresent()) {
            User user = (User) opUser.get();
            if (user.getRoles().contains(Role.ADMIN)) {
                user.removeRole(Role.ADMIN);
            }
            userRepository.save(user);
        }
    }

    public boolean isRegistered(String username) throws HibernateException {
        return userRepository.findByUsername(username) != null;
    }

    public boolean isEmailExist(String email) {
        return userRepository.findByEmail(email) != null;
    }
}
```

Клас AdminController

```
package ua.tour.api.controllers;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;
import ua.tour.api.entities.TourReservation;
import ua.tour.api.entities.User;
import ua.tour.api.services.ReservationService;
import ua.tour.api.services.UserService;

@RestController
@RequestMapping("/api/admin/")
public class AdminController {

    private final UserService userService;
    private final ReservationService reservationService;

    @Autowired
    public AdminController(UserService userService, ReservationService
reservationService) {
        this.userService = userService;
        this.reservationService = reservationService;
    }


    @GetMapping("all-users/{page}")
    public Iterable<User> allUsers(@PathVariable int page) {
        return userService.getAllUsers(page);
```

```
    }

    @PostMapping("/users/set-admin/{id}")
    public void setAdmin(@PathVariable Long id) {
        userService.addAdminRole(id);
    }

    @PostMapping("/users/remove-admin/{id}")
    public void removeAdmin(@PathVariable Long id) {
        userService.removeAdminRole(id);
    }

    @GetMapping("/all-reservations/{page}")
    public Iterable<TourReservation> getAllReservations(@PathVariable int page) {
        return reservationService.getAllReservations(page);
    }
}
```

Клас AuthController

```
package ua.tour.api.controllers;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import ua.tour.api.entities.User;
import ua.tour.api.exceptions.UserRegistrationFailedException;
import ua.tour.api.services.UserService;

import javax.validation.Valid;
import java.util.Collections;
import java.util.Map;

@RestController
@RequestMapping("/api/auth/")
public class AuthController {

    private PasswordEncoder passwordEncoder;
    private UserService userService;

    @Autowired
    public AuthController(UserService userService, PasswordEncoder
passwordEncoder) {
        this.userService = userService;
        this.passwordEncoder = passwordEncoder;
    }

    @PostMapping("/sign-up")
    public void signUp(@Valid @RequestBody User user) throws
UserRegistrationFailedException {
        user.setPassword(passwordEncoder.encode(user.getPassword()));
        userService.createUser(user);
    }

    @PostMapping("/is-registered")
    public Map<String, Boolean> checkUsername(@RequestBody Map<String, String>
payload) {
        return Collections.singletonMap("isRegistered",
```

```
userService.isRegistered(payload.get("username")));
    }

    @PostMapping("/is-email-exist")
    public Map<String, Boolean> checkEmail(@RequestBody Map<String, String>
payload) {
        return Collections.singletonMap("isExist",
userService.isEmailExist(payload.get("email")));
    }
}
```

Клас HotelController

```
package ua.tour.api.controllers;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.Page;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.multipart.MultipartFile;
import ua.tour.api.entities.Hotel;
import ua.tour.api.exceptions.DeletionException;
import ua.tour.api.exceptions.NotFoundException;
import ua.tour.api.services.HotelService;

import javax.validation.Valid;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.Collections;
import java.util.Map;

@RestController
@RequestMapping("/api/hotel")
public class HotelController {

    private HotelService hotelService;

    @Autowired
    public HotelController(HotelService hotelService) {
        this.hotelService = hotelService;
    }

    @GetMapping("/{page}")
    public Page<Hotel> getAllHotels(@PathVariable int page) {
        return hotelService.getAllHotels(page);
    }

    @GetMapping("/{id}")
    public Hotel getHotelById(@PathVariable Long id) throws NotFoundException {
        return hotelService.getHotelById(id);
    }

    @PutMapping
    public void updateHotel(@Valid @RequestBody Hotel hotel) throws
NotFoundException {
        hotelService.updateHotel(hotel);
    }

    @DeleteMapping("/{id}")
    public void deleteHotel(@PathVariable Long id) throws NotFoundException,
DeletionException, FileNotFoundException {
        hotelService.deleteHotel(id);
```

```
    }

    @PostMapping
    public Map<String, Long> addHotel(
            @Valid @RequestPart(name = "hotel") Hotel hotel,
            @RequestPart(name = "image")MultipartFile file) throws IOException {

        return Collections.singletonMap("hotelId", hotelService.addHotel(hotel,
file));
    }
}
```

Клас ReservationController

```
package ua.tour.api.controllers;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.Page;
import org.springframework.web.bind.annotation.*;
import ua.tour.api.entities.TourReservation;
import ua.tour.api.exceptions.NotFoundException;
import ua.tour.api.exceptions.SeatsCountOverflowException;
import ua.tour.api.services.ReservationService;

import javax.validation.Valid;
import java.security.Principal;

@RestController
@RequestMapping("/api/reservations")
public class ReservationController {

    private ReservationService reservationService;

    @Autowired
    public ReservationController(ReservationService reservationService) {
        this.reservationService = reservationService;
    }

    @PostMapping
    public void createReservation(@Valid @RequestBody TourReservation
reservation, Principal principal) throws NotFoundException,
SeatsCountOverflowException {
        reservationService.addReservation(reservation, principal);
    }

    @GetMapping("/{page}")
    public Page<TourReservation> getAllUserReservations(@PathVariable int page,
Principal principal) {
        return reservationService.getAllUserReservations(page, principal);
    }

    @PostMapping("/cancel/{id}")
    public void cancelReservation(@PathVariable Long id) throws NotFoundException
{
        reservationService.cancelReservation(id);
    }

}
```

Клас TourController

```
package ua.tour.api.controllers;

import org.springframework.data.domain.Page;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.multipart.MultipartFile;
import ua.tour.api.entities.Tour;
import ua.tour.api.exceptions.DeletionException;
import ua.tour.api.exceptions.NotFoundException;
import ua.tour.api.exceptions.ReservationsExistException;
import ua.tour.api.services.TourService;

import javax.validation.Valid;
import java.io.FileNotFoundException;
import java.io.IOException;

@RestController
@RequestMapping("/api/tours")
public class TourController {

    private final TourService tourService;

    public TourController(TourService tourService) {
        this.tourService = tourService;
    }

    @GetMapping("/{page}")
    public Page<Tour> getAllTours(@PathVariable int page) {
        return tourService.getAllTours(page);
    }

    @DeleteMapping("/{id}")
    public void deleteTour(@PathVariable Long id) throws
ReservationsExistException, DeletionException, FileNotFoundException {
        tourService.removeTour(id);
    }


    @PutMapping
    public void updateTour(@Valid @RequestBody Tour tour) throws
NotFoundException {
        tourService.updateTour(tour);
    }


    @PostMapping
    public void createTour(
            @Valid @RequestPart(name = "tour") Tour tour,
            @RequestPart(name = "image") MultipartFile file) throws
NotFoundException, IOException {
        tourService.createNewTour(tour, file);
    }
}
```

Клас Hotel

```
package ua.tour.api.entities;

import javax.persistence.*;
import javax.validation.constraints.Size;

@Entity
```

```
public class Hotel {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;

    @Size(min = 3,message = "Hotel name must be longer than 2 characters")
    private String name;

    @Size(min = 32, max = 2048, message = "Hotel description must be longer than
31 but less than 2049")
    private String description;

    @Size(min = 2)
    private String country;

    @Size(min = 2)
    private String location;

    @Column(updatable = false)
    private String imageSrc;

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getDescription() {
        return description;
    }

    public void setDescription(String description) {
        this.description = description;
    }

    public String getCountry() {
        return country;
    }

    public void setCountry(String country) {
        this.country = country;
    }

    public String getLocation() {
        return location;
    }

    public void setLocation(String location) {
        this.location = location;
    }
```

```
    public String getImageSrc() {
        return imageSrc;
    }

    public void setImageSrc(String imageSrc) {
        this.imageSrc = imageSrc;
    }
}
```

Клас Role

```
package ua.tour.api.entities;

import org.springframework.security.core.GrantedAuthority;

public enum  Role implements GrantedAuthority {
    USER, ADMIN;

    @Override
    public String getAuthority() {
        return name();
    }
}
```

Клас Tour

```
package ua.tour.api.entities;

import org.hibernate.validator.constraints.Range;

import javax.persistence.*;
import javax.validation.constraints.Min;
import javax.validation.constraints.NotBlank;
import javax.validation.constraints.NotNull;
import javax.validation.constraints.Size;
import java.util.Date;

@Entity
public class Tour {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;

    @NotBlank
    @Size(min = 4, max = 32, message = "Min title length is 4 characters and max
32")
    private String title;

    @NotBlank
    @Size(min = 32, max = 2048, message = "Description length must be more than
31 characters but less then 2049 ")
    private String description;

    @NotNull
    private Date startDate;

    @NotNull
    private Date endDate;
```

```
@NotNull
private Integer seatCount;

@OneToOne(fetch = FetchType.EAGER)
@JoinColumn(name = "hotel_id")
private Hotel hotel;

@Column(updatable = false)
private String imageSrc;

public Long getId() {
    return id;
}

public void setId(Long id) {
    this.id = id;
}

public String getTitle() {
    return title;
}

public void setTitle(String title) {
    this.title = title;
}

public String getDescription() {
    return description;
}

public void setDescription(String description) {
    this.description = description;
}

public Date getStartDate() {
    return startDate;
}

public void setStartDate(Date startDate) {
    this.startDate = convertToJDate(startDate);
}

public Date getEndDate() {
    return endDate;
}

public void setEndDate(Date endDate) {
    this.endDate = convertToJDate(endDate);
}

public Integer getSeatCount() {
    return seatCount;
}

public void setSeatCount(Integer seatCount) {
    this.seatCount = seatCount;
}

public Hotel getHotel() {
    return hotel;
}
```

```
    public void setHotel(Hotel hotel) {
        this.hotel = hotel;
    }

    private Date convertToJDate(Date date) {
        return new Date(date.getTime()*1000 );
    }

    public String getImageSrc() {
        return imageSrc;
    }

    public void setImageSrc(String imageSrc) {
        this.imageSrc = imageSrc;
    }
}
```

Клас TourReservation

```
package ua.tour.api.entities;

import com.fasterxml.jackson.annotation.JsonIgnore;

import javax.persistence.*;
import javax.validation.constraints.NotNull;
import javax.validation.constraints.Size;

@Entity
public class TourReservation {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "user_id")
    @JsonIgnore
    private User user;

    @Column(name = "user_id", insertable = false, updatable = false)
    private Long user_id;

    @NotNull
    @ManyToOne(fetch = FetchType.EAGER)
    @JoinColumn(name = "tour_id")
    private Tour tour;

    @Size(max = 2048, message = "Max chars in message - 2048")
    private String comment;

    private boolean isActive;

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public User getUser() {
```

```
        return user;
    }

    public void setUser(User userId) {
        this.user = userId;
    }

    public Tour getTour() {
        return tour;
    }

    public void setTour(Tour tour) {
        this.tour = tour;
    }

    public String getComment() {
        return comment;
    }

    public void setComment(String comment) {
        this.comment = comment;
    }

    public Long getUser_id() {
        return user_id;
    }

    public boolean isActive() {
        return isActive;
    }

    public void setActive(boolean active) {
        isActive = active;
    }
}
```

Клас User

```
package ua.tour.api.entities;

import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.userdetails.UserDetails;

import javax.persistence.*;
import javax.validation.constraints.*;
import java.util.Collection;
import java.util.Set;

@Entity
public class User implements UserDetails {

    public User() {
    }

    public User(String username, String password, Set<Role> roles) {
        this.username = username;
        this.password = password;
        this.roles = roles;
    }

    public User(String username, String email, String password, String fullName,
String mobileNumber) {
```

```
        this.username = username;
        this.email = email;
        this.password = password;
        this.fullName = fullName;
        this.mobileNumber = mobileNumber;
    }

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;


    @NotBlank
    @Column(unique = true)
    @Size(min=3, message="Name should have at least 2 characters")
    private String username;

    @NotBlank
    @Column(unique = true)
    @Email(message ="Invalid email format")
    private String email;

    @NotBlank
    @Size(min = 6, message = "Invalid password (minimum 6 and maximum 32
characters)")
    private String password;

    @NotBlank
    @Size(min = 3, message = "Full name must be longer than 2 characters")
    private String fullName;

    @NotBlank
    @Size(min = 13, max = 13, message = "Phone number must consist of 13 digits")
    @Digits(integer=13, fraction=0, message = "Phone number invalid")
    private String mobileNumber;

    @NotNull
    @Column(columnDefinition = "INTEGER")
    private boolean active;

    @ElementCollection(targetClass = Role.class, fetch = FetchType.EAGER)
    @CollectionTable(name = "user_role", joinColumns = @JoinColumn(name =
"user_id"))
    @Enumerated(EnumType.STRING)
    private Set<Role> roles;

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getUsername() {
        return username;
    }

    @Override
    public boolean isAccountNonExpired() {
        return true;
    }
```

```
@Override
public boolean isAccountNonLocked() {
    return true;
}

@Override
public boolean isCredentialsNonExpired() {
    return true;
}

@Override
public boolean isEnabled() {
    return isActive();
}

@Override
public Collection<? extends GrantedAuthority> getAuthorities() {
    return getRoles();
}


public void setUsername(String username) {
    this.username = username;
}

public String getEmail() {
    return email;
}

public void setEmail(String email) {
    this.email = email;
}

public String getPassword() {
    return password;
}

public void setPassword(String password) {
    this.password = password;
}

public String getFullName() {
    return fullName;
}

public void setFullName(String fullName) {
    this.fullName = fullName;
}

public String getMobileNumber() {
    return mobileNumber;
}

public void setMobileNumber(String mobileNumber) {
    this.mobileNumber = mobileNumber;
}

public boolean isActive() {
    return active;
}
```

```
    public void setActive(boolean active) {
        this.active = active;
    }

    public Set<Role> getRoles() {
        return roles;
    }

    public void setRoles(Set<Role> roles) {
        this.roles = roles;
    }

    public void addRole(Role role) {
        this.roles.add(role);
    }

    public void removeRole(Role role) {
        this.roles.remove(role);
    }
}
```

БІБЛІОГРАФІЧНА ДОВІДКА

Тема дипломної роботи: Розробка структури бази даних та серверної частини Web-ресурсу з надання туристичних послуг.

Обсяг пояснювальної записки: 45 аркуш

Перелік графічних матеріалів:

Таблиць 0

рисунків 34

додатків 1 на 22 аркушах.

Дата закінчення дипломного проекту: “10” червня 2019 р.

Студент – дипломник Бернацький О.М

(підпис) (розшифровка підпису)