Travel Planner	Version: 1.0
Documentation template	Date: <26/02/2022>

<Travel Planner>

Project documentation

Student(s):

- Moisa Oana Miruna

Group: 30432

MINISTRY OF EDUCATION

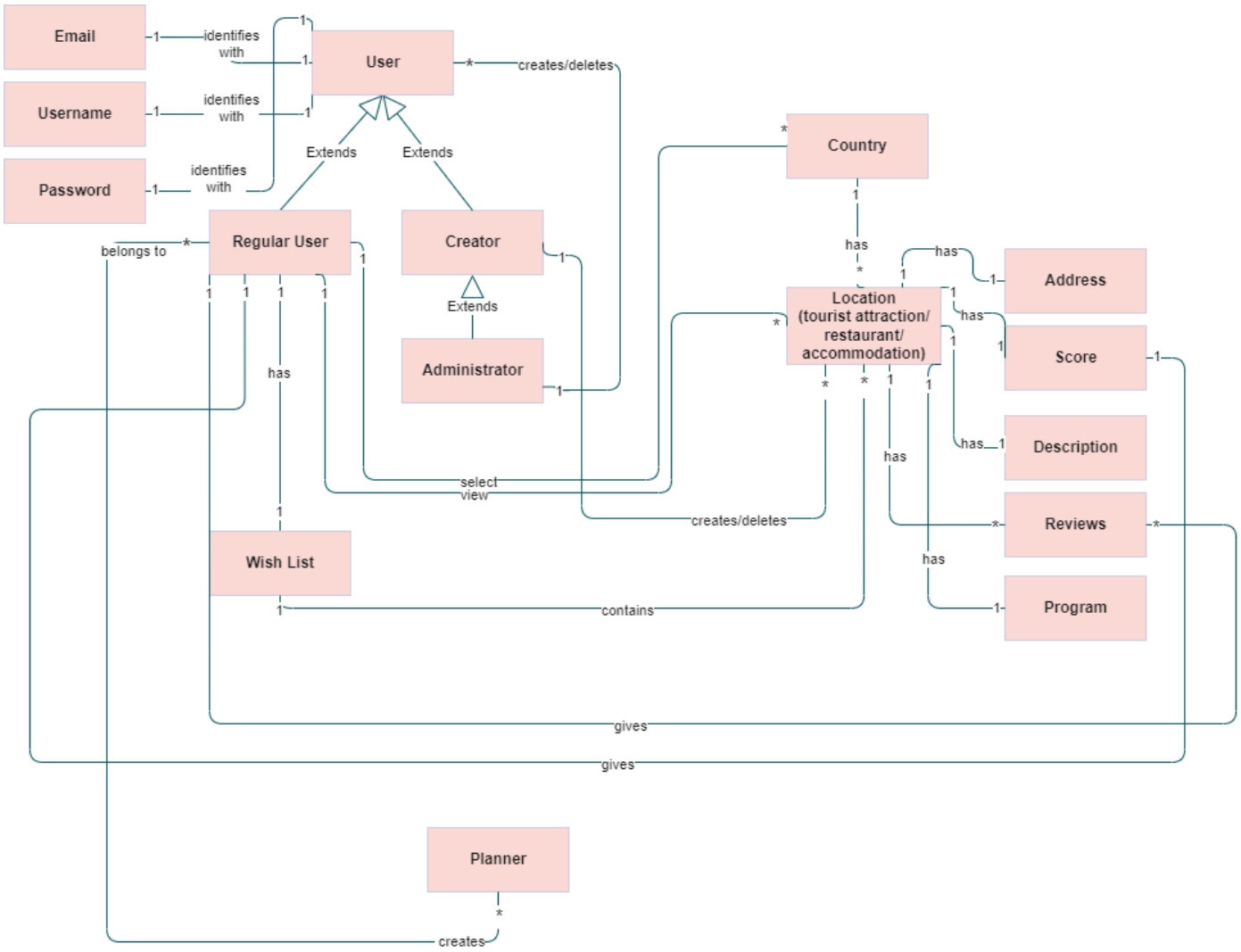
TECHNICAL UNIVERSITY
OF CLUJ-NAPOCA, ROMANIA

Contents

I Project specification

The project I want to develop is called Map Out and it is a travel planning application. The idea of creating such an application came from my experiences of travelling with my friends. We always used a simple Google Docs that had to be shared between us in order to plan out our trip. But this process can be more fun and a lot easier. This application supports 3 kinds of users: regular users, creators and administrators. To become a user of any kind, you have to create an account by specifying your email address, a username, a password and the type of user you want to become. If you already have an account, you can simply login in the application with your credentials. The regular user can do multiple things: explore, create a planner and view their profile. The explore functionality lets the users get inspiration for their next trip and see what options are available. They can select a country, then select what they want to view: tourist attractions, restaurants or accommodation in that country. Each such place has a page with information: address, stars, description, photos, hours, reviews, etc. The user can add a review to that place, give it a score or add it to their wish list. To create a planner, the user must give a name to the trip, and then create a plan for each day of the journey. To create a plan means to add hour interval-activity pairs to that day. Moreover, the user can invite other users of the application to the plan, such that everyone can see/edit/modify it. Besides adding/deleting things from the plan, the users can cross off activities after they have done them. The profile of each regular user contains, apart from their information, all the planners they have created/have been invited to and their wish list. The creator type user can add a tourist attraction/restaurant/accommodation to the application in order to promote it, by providing all the required information for it. They can also view their profile, where they will have a list of all their featured places. The last type of user is the administrator, which is usually just one or a few. The admin can do all the CRUD operations to maintain the application: delete/add users, delete/add locations.

1.1 Domain Model Diagram



II Use-Case model

In this chapter, I will present a more in-depth description of the use cases of the application.

In the first section, “Users and stakeholders”, I will list all the possible users, meaning people who use the application directly, and also some stakeholders, meaning the individuals that are actively involved in the project or whose interests might be affected by it.

In the next section, “Use case identification”, I will thoroughly present five main use-cases, two of which are general for every type of user, two are specific to regular users and one is specific to the creators and administrators.

In the last section, “UML use-case diagram”, I will present the use-case diagram with all the functionalities of the application.

2.1 Users and stakeholders

- **USERS:**

The application supports three kinds of users:

- regular users
- creators
- administrators

To become a user of the application, a person has to create an account by specifying their email address, a username, a password and the type of user they want to become out of the three. After they have created an account, they can simply login using the username and password. The three kinds of users have these attributes in common: username, email address, password, type of user and these actions: create account, login, view profile. After this point, they support different functionalities:

- Regular User:
  - Explore: See the tourist attractions, the restaurants and the accommodation available in a country. Sort them by score, filter them by name or city.
  - Review: Give a review to a place.
  - Give a score: Give a score out of 5 stars to a place.
  - Add to wish list: Add a place to your wish list.
  - Create planner: Create a planner for your trip, plan out what to do every day with hour-activity details.

  - Share planner with friends: Invite other users to the planner so that they can also contribute to it.
- Creator:
  - Add tourist attraction/restaurant/accommodation: Promote a place by creating a page for it in the application so that the regular users can see it.
- Administrator:
  - CRUD operations: add/delete user, add/delete tourist attraction/restaurant/accommodation

➢ **STAKEHOLDERS:**

- **Internal stakeholders:**
  - Application developers
  - Managers
- **External stakeholders:**
  - Regular people who enjoy travelling
  - Restaurant/Accommodation owners in touristic places
  - The community of a touristic city/country

2.2 Use-Case identification

1. **Use case 1**

Use case name: Create Account

Level: User-Goal

Main actor: Regular User, Creator, Administrator

Main success scenario: The user enters his/her personal information in the designated textboxes: email address, username, password, confirm password and selects the type of account they want to create (regular user, creator, admin). They can click the Create button to create the account or click the Reset button to reset the page and clear the textboxes.

Extension: A bad flow of the program would be if the user forgets to fill in one of the fields or if the provided password is not the same as the one from the 'confirm password' field (they don't coincide). In both cases the user is warned and the account cannot be created if the mistake is not corrected.

2. **Use case 2**

Use case name: Login

Level: User-Goal

Main actor: Regular User, Creator, Administrator

Main success scenario: The user has already created an account. The users enter their username and password. If they click the 'Login' button and the credentials are correct, they will be redirected to their account. If they click the 'Reset' button, the page will be refreshed and the textboxes will be cleared.

Extension: A bad flow of the program would be if the provided username doesn't exist or if the password is not correct. Either way, the user will be warned and they will not be able to enter their account until they provide the correct information.

3. **Use case 3**

Use case name: Add tourist attraction/restaurant/accommodation

Level: User-Goal

Main actor: Creator, Administrator

Main success scenario: The creator/administrator clicks the 'add tourist attraction' button from their profile and is redirected to a page for filling out all the required information. The creator/administrator has to provide a name for the place, the country and city where it is located, the full address, a description, the opening hours and a phone number. To add this place in the application, the user clicks the 'Add' button. A reset button is also available.

Extension: A bad flow of the program would be if the user forgets to fill in one of the fields because all of them are mandatory. Also, the location must not already exist in the application. Either way, the creator gets a warning.

4. **Use case 4**

Use case name: Explore

Level: User-Goal

Main actor: Regular user

Main success scenario: The user clicks the 'Explore' button from the main page and is redirected to a page which contains a list of countries. The user selects a country then a category: tourist attraction, restaurant or accommodation. The user will be provided with a list of all available places of that type. The user can sort the placed by their score or filter them by name/city. The user simply clicks on the place they want to view and will be redirected to their respective place.

Extension: A bad flow of the program would be if the user wants to filter the locations by a city/name that doesn't exist. Then there will be no search results.

5. **Use case 5**

Use case name: Create a planner

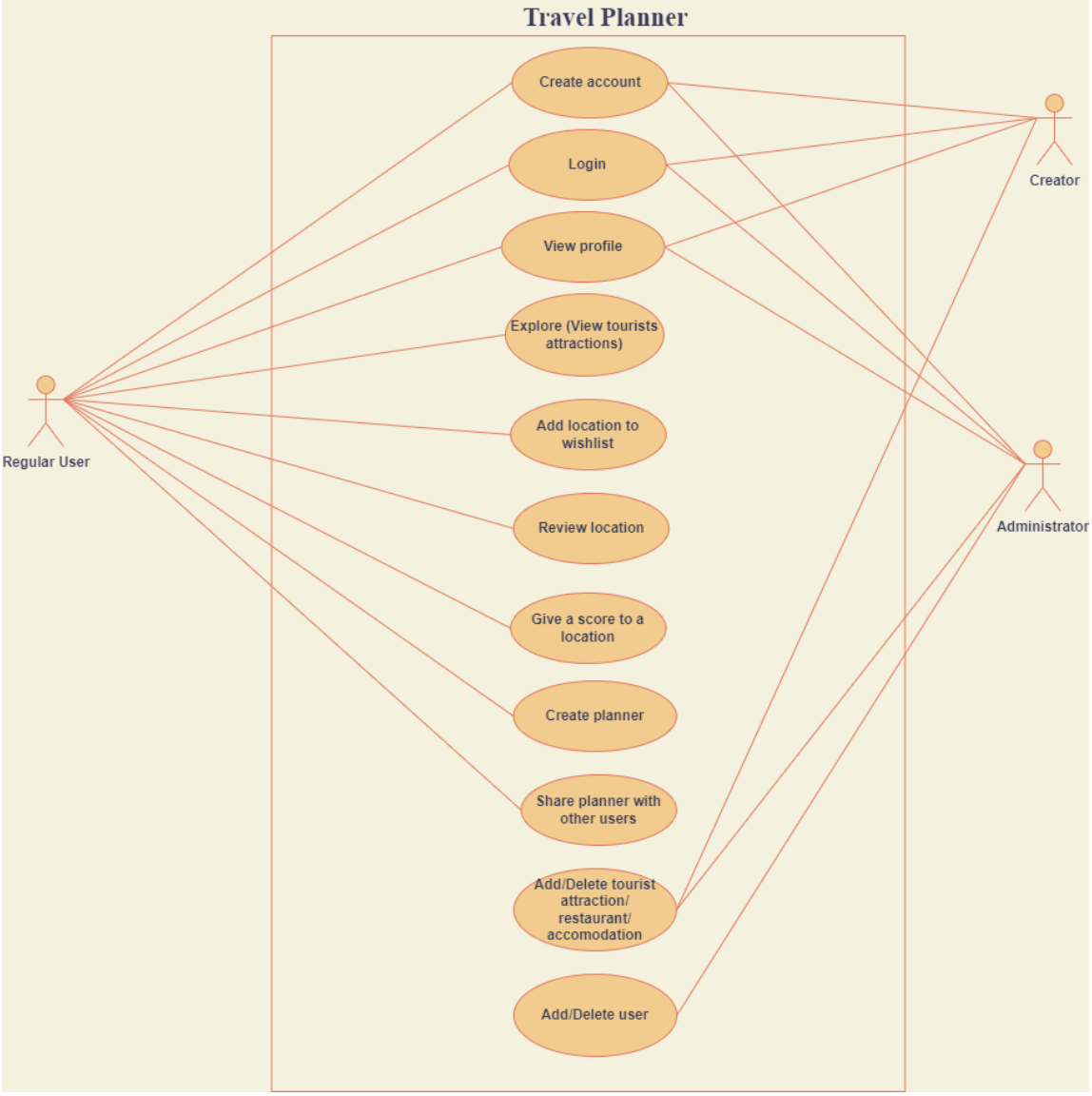
Level: User-Goal

Main actor: Regular user

Main success scenario: The user clicks the 'Create planner' button from the main page. The user will have to add a separate section to the planner for every day of the trip. To each day the user can add hour interval-activity pairs for every thing they plan to do that day.

Extension: A bad flow of the program would be if the user doesn't fill all the required info for a planner. If this is the case, a warning will be generated.

2.3 UML Use-Case diagram



III Architectural design

3.1 Conceptual architecture

My travel planning application will be a **web application**. The architecture I will use is a **Layered Architecture** because it is the one that fits the structure of my application, which uses a database for data storage. This means that I will have 5 packages: **Controller Package**

(representing the Presentation Layer), **Service Package** (representing the Business Layer), **Repository Package** (representing the Persistence Layer) and the **Model Package** (representing the Database Layer).

Model Package: it is the lowest layer form the hierarchy and it represents the actual tables from the database. I will use Hibernate ORM Framework that will map the classes that I create in this package to tables in the database.

Repository Package: this layer can access data from the database and send it to the business layer. In these classes we will perform basic requests to the database like insert, delete, retrieve or modify data.

Service Package: this layer can only access the persistence layer below it. In this layer we will do the validations to the data we want to insert in the tables and other functionalities and then we will call methods from the persistence layer to do the actual operations to the database. Each class in the Service package will have a corresponding class in the Repository package.

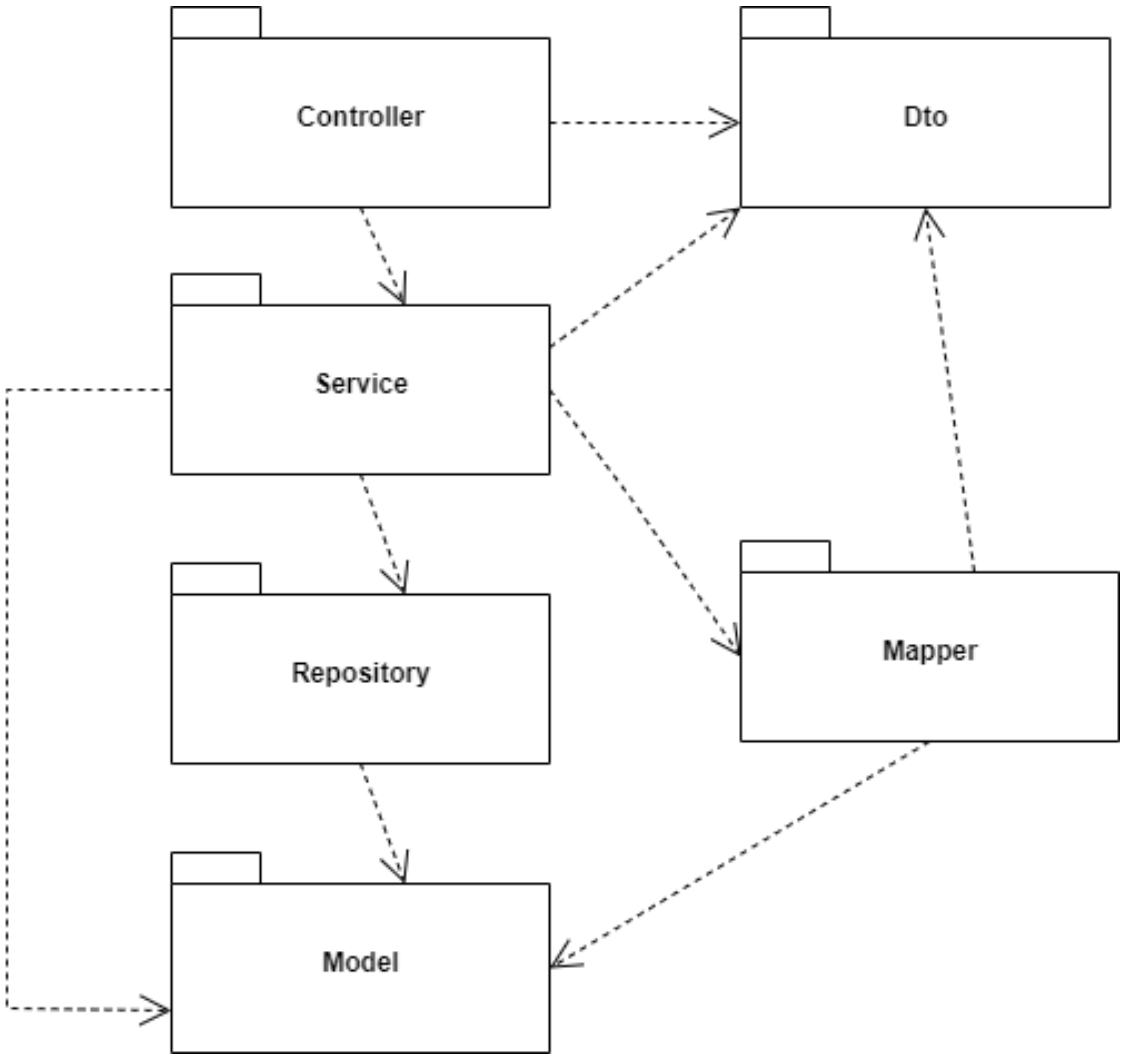
Controller Package: this layer will hold the classes that deal with the user interface. Each web page will have one corresponding controller class in this package.

Backend: Java & Spring

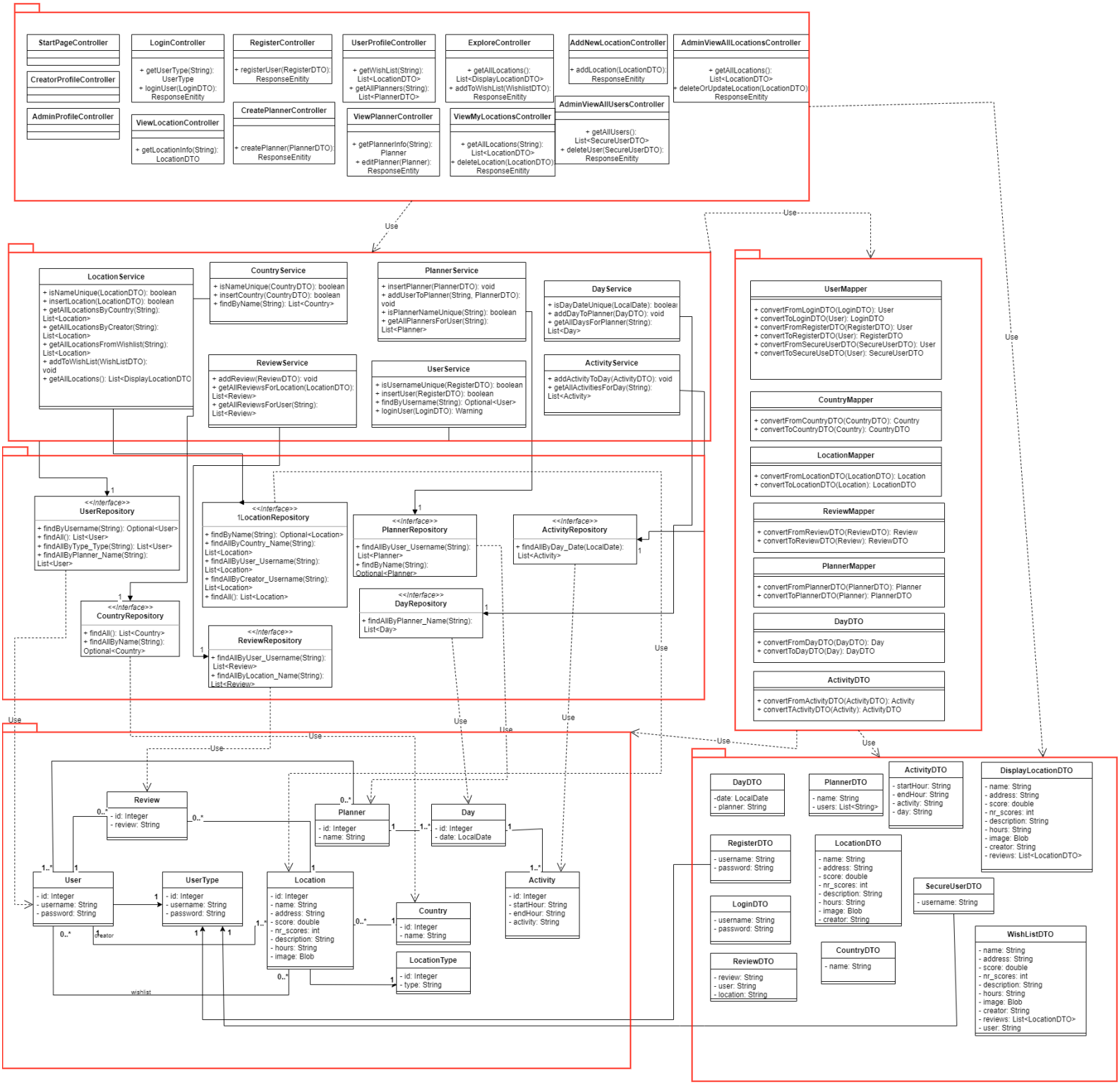
Frontend: React

Type: Client-Server

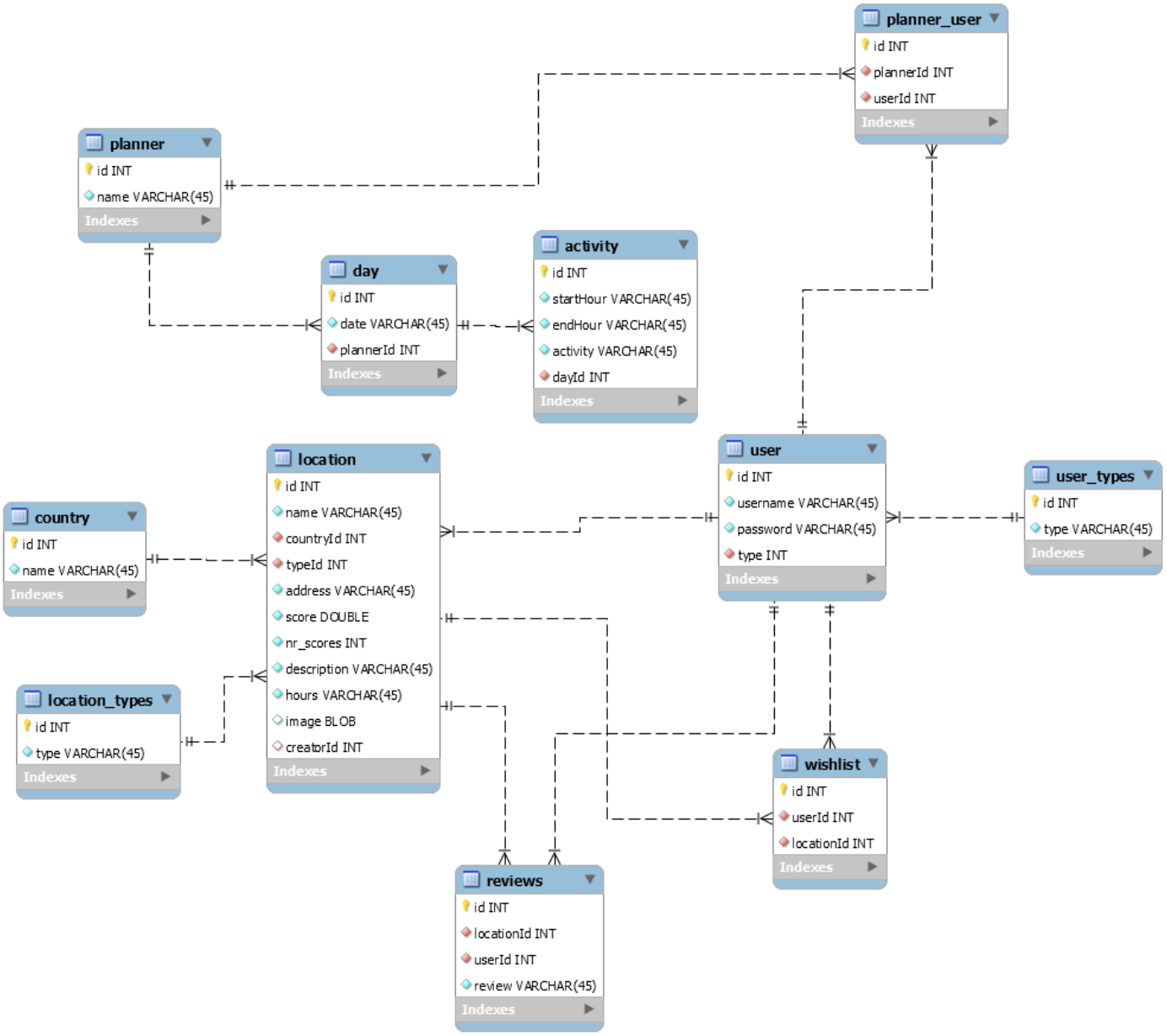
3.2 Package diagram



3.3 Class diagram

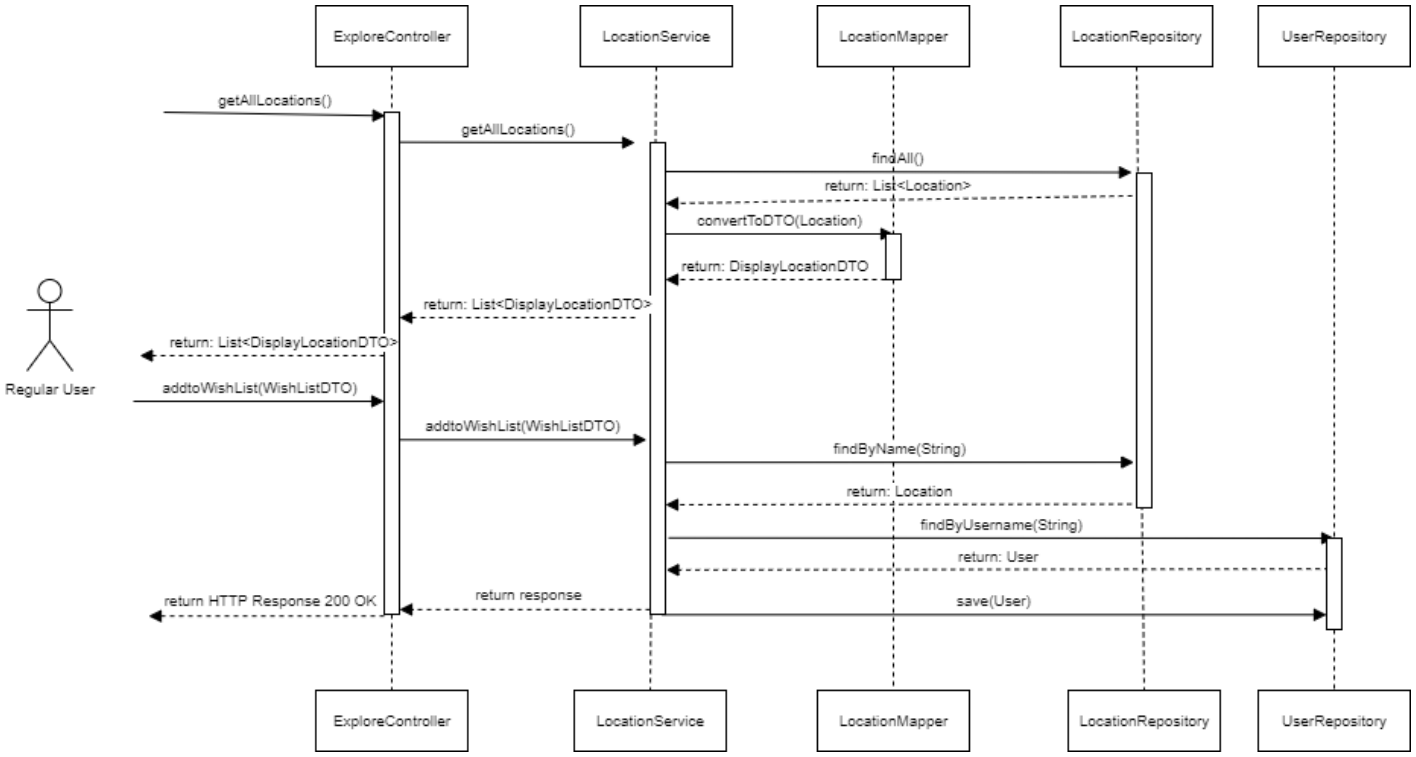


3.4 Database (E-R/Data model) diagram

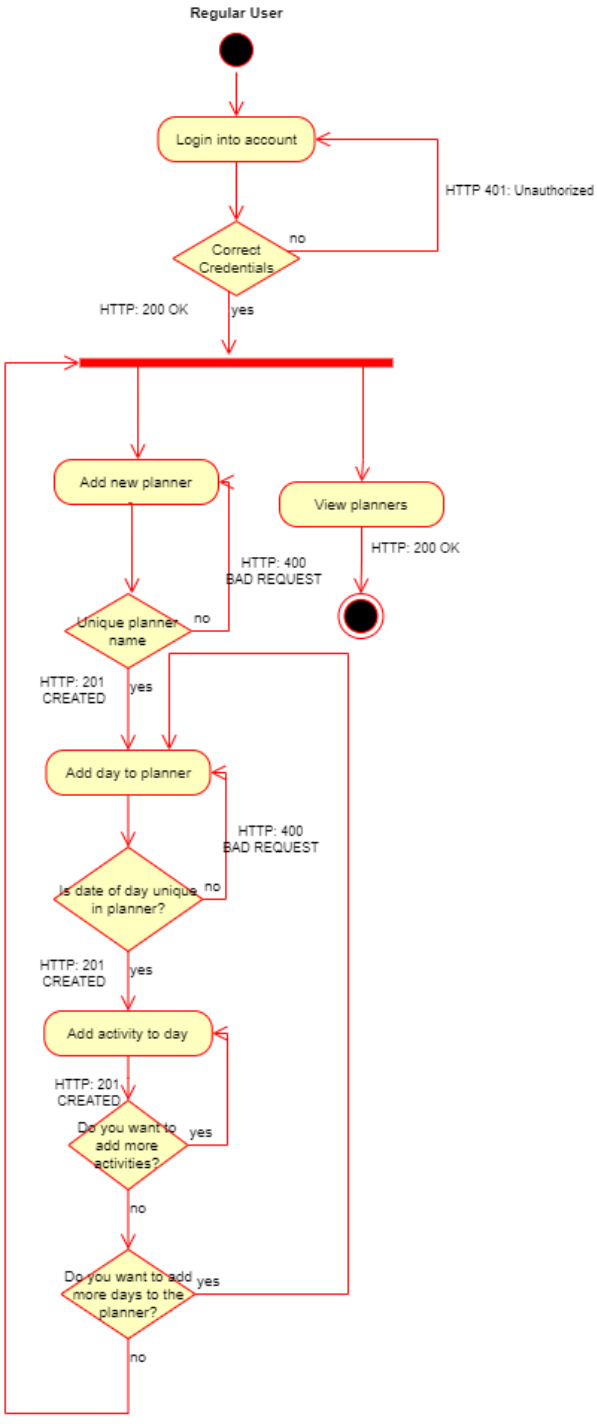


3.5 Sequence diagram

Flow presented in the sequence diagram: Add location to wish list by regular user



3.6 Activity diagram



Flow presented in the activity diagram: regular user login, view all planners, add new planner

IV Supplementary specifications

< Se va scrie o mica introducere./>

4.1 Non-functional requirements

The non-functional requirements of my application are:

1. **Security**

This non-functional requirement refers to the protection of data inside the system against unauthorized access. I have chosen this NFR because I believe it is of utmost importance that the data from the database is always valid and never compromised, because the whole application relies on it.

This requirement is materialized through the login and create account functionality. Each user (regular user, creator, admin) has to create an account with a password in order to use the application. Also, the CRUD operations on the database can be performed only by the administrator, so the regular user for example does not have edit access in the user or destinations tables.

2. **Performance**

This non-functional requirement describes how fast the application responds to user requests, so how much the user has to wait for the application to provide the desired result. For my application, the response time for all operations has to be of maximum 1 second. Because the operations needed for this application are not necessarily very costly, I believe this is a realistic requirement.

3. **Portability**

This non-functional requirement refers to the accessibility of the application across devices and across different environments. The application must be cross-platform and run on Microsoft Windows, Linux, and macOS. This is achieved because the application is written in Java, which is cross-platform by default. The java compiler transforms the source code in bytecode that is then executed by the JVM specific to that platform.

4. **Usability**

This non-functional requirement refers to how hard is to use the application. The interface of the application has to be user-friendly and easy to use. The user must be able to learn how to use the application with not much difficulty (learnability) and to use it right away when they return to it (memorability). The design must be creative and unique. Also, the interface must be explicit enough such that the error rate of the user input is not very high.

5. **Availability**

This non-functional requirement refers to how accessible is the application for the user in a moment in time. The application must be accessible by all users in any moment of the day. If for some reason the system becomes unavailable, the issue must be resolved in a couple of hours.

4.2 Design constraints

In this section, I will describe the technical constraints around which my application is designed.

- The web application will be developed in **Java,** using the **Spring framework**
- It will use **React** for the frontend design
- **MySQL Workbench** for database administration
- **Hibernate** framework for communication with the database

V Testing

< Se va discuta la laborator./>

5.1 Testing methods/frameworks

5.2 Future improvements

VI Documentation of APIs

POST/register

Request:

● body (JSON):

{

“firstName”: “SomeFirstName”,

“lastName”: “SomeLastName”,

“username”: “someUsername”,

“password”: “somePassword”,

“email”: “email@gmail.com”,

“type”: “UserType.USER”,

}

Responses:

● 201 CREATED – User was successfully created.

● 406 NOT ACCEPTABLE – Invalid username because it already exists.

● 400 BAD REQUEST – Invalid email address provided.

POST/login

Request:

● body (JSON):

{

“username”: “someUsername”,

“password”: “somePassword”,

}

Responses:

● 200 OK – Correct credentials provided!

● 401 UNAUTHORIZED– Invalid username or password.

GET/creatorprofile

Responses:

● 200 OK – Page accessed by a creator.

● 403 FORBIDDEN– Page accessed by another type of user.

GET/viewcreator/{username}

Response:

● body (JSON):

{

0:

{

“address”: "Abercorn, South Queensferry EH30 9SL, United Kingdom"

“country”: "Scotland"

“creator_username”: "ioanaghete"

“description”: "Midhope Castle is a 16th-century tower house in Scotland. "

“hours”: "Open to the public every day from 8 am to 4 pm."

“image”: "midhope.jpg"

“locationType”: "ATTRACTION"

“name”: "Midhope Castle"

“nrScores”: 2

“reviews”: {

0:{

“review”: "The most magical place! <3"

“user”: "oanamoisa"

}

1:{

“review”: "Couldn't wait to visit it after watching Outlander!!"

“user”: "alinamihut"

}

}

“score”: 4

}

}

Responses:

● 200 OK – Page accessed by a creator!

● 403 FORBIDDEN– Page accessed by another type of user.

GET/addlocation

● body (JSON):

{

“countries”:

{

0: {“name”: 'England'}

1: {“name”: 'Hungary'}

2: {“name”: 'Romania'}

3: {“name”: 'Scotland'}

},

“types”:

{

0: "ATTRACTION"

1: "RESTAURANT"

2: "ACCOMMODATION"

}

}

Responses:

● 200 OK – Page accessed by a creator.

● 403 FORBIDDEN– Page accessed by another type of user.

POST/addlocation

Request:

● body (JSON):

{

“name”: “someLocation”,

“address”: “someAddress”,

“hours”: “someHours”,

“description”: “someDescription”,

“country”: “someCountry”,

"locationType": "ATTRACTION",

"image": "image.png",

"creator_username": "someUsername"

}

Responses:

- 200 OK – Location successfully added!
- 400 BAD REQUEST – Location already exists.

GET/viewlocation

Responses:

- 200 OK – Page accessed by a creator or regular user.
- 403 FORBIDDEN– Page accessed by creator.

PUT/viewlocation

Request:

- body (JSON):

{

"score": 5

"location":

{

“name”: “someLocation”,

“address”: “someAddress”,

“hours”: “someHours”,

“description”: “someDescription”,

“country”: “someCountry”,

“locationType”: “ATTRACTION”,

“image”: “image.png”,

“creator_username”: “someUsername”

“score”: 4,

“nrScores”:2,

“reviews”:

{

0:{ “user”: “someUsername”,

“review”: “some review text”

}

}

}

}

Responses:

● 200 OK – Score updated!

POST/viewlocation

Request:

● body (JSON):

{

“review”: “some review text”,

“user”: “someUsername”,

“location”:

{

“name”: “someLocation”,

“address”: “someAddress”,

“hours”: “someHours”,

“description”: “someDescription”,

“country”: “someCountry”,

“locationType”: “ATTRACTION”,

“image”: “image.png”,

“creator_username”: “someUsername”

“score”: 4,

“nrScores”:2,

“reviews”:

{

0:{ “user”: “someUsername”,

"review": "some review text"

}

}

}

}

Responses:

● 201 CREATED – Review created!

POST/wishlist

Request:

● body (JSON):

{

"user": "someUsername",

"location":

{

"name": "someLocation",

"address": "someAddress",

"hours": "someHours",

"description": "someDescription",

"country": "someCountry",

"locationType": "ATTRACTION",

“image”: “image.png”,

“creator_username”: “someUsername”

“score”: 4,

“nrScores”:2,

“reviews”:

{

0:{ “user”: “someUsername”,

“review”: “some review text”

}

}

}

}

Responses:

● 200 OK – Successfully added to wishlist!

● 400 BAD REQUEST – Location already exists in wishlist.

DELETE/deletelocation

Request:

● body (JSON):

{“someString”}

Responses:

● 200 OK – Successful deletion.

● 404 NOT_FOUND– Location with the provided name doesn't exist!

GET/adminprofile

Response:

● body (JSON):

{

"countries":

{

0: {name: 'England'}

1: {name: 'Hungary'}

}

"users":

{

0: "oanamoisa"

1: "ioanaghete"

}

}

Responses:

● 200 OK – Page accessed by an admin.

● 403 FORBIDDEN– Page accessed by other types of users.

POST/addcountry

Request:

● body (JSON):

{“someCountryName”}

Responses:

● 200 OK – Successfully added country!

● 400 BAD REQUEST – Country already exists.

DELETE/deletecountry

Request:

● body (JSON):

{“someCountryName”}

Responses:

● 200 OK – Successful deletion.

● 404 NOT_FOUND -Country with the provided name doesn’t exist!

VI Bibliography