<Formula 1 ticketing manager>

Project documentation

Student(s):

* S1
* S2

**Group:**

**Contents**

[I Project specification](#_heading=h.gjdgxs) **3**

[1.1 Domain Model Diagram](#_heading=h.30j0zll) 3

[II Use-Case model](#_heading=h.1fob9te) **3**

[2.1 Users and stakeholders](#_heading=h.f2irg1azp7rm) 4

[2.2 Use-Case identification](#_heading=h.3znysh7) 4

[2.3 UML Use-Case diagram](#_heading=h.2et92p0) 4

[III Architectural design](#_heading=h.tyjcwt) **5**

[3.1 Conceptual architecture](#_heading=h.3dy6vkm) 5

[3.2 Package diagram](#_heading=h.1t3h5sf) 5

[3.3 Class diagram](#_heading=h.4d34og8) 5

[3.4 Database (E-R/Data model) diagram](#_heading=h.2s8eyo1) 6

[3.5 Sequence diagram](#_heading=h.17dp8vu) 6

[3.6 Activity diagram](#_heading=h.3rdcrjn) 6

[IV Supplementary specifications](#_heading=h.26in1rg) **6**

[4.1 Non-functional requirements](#_heading=h.lnxbz9) 6

[4.2 Design constraints](#_heading=h.35nkun2) 6

[V Testing](#_heading=h.1ksv4uv) **7**

[5.1 Testing methods/frameworks](#_heading=h.44sinio) 7

[5.2 Future improvements](#_heading=h.2jxsxqh) 7

[VI Bibliography](#_heading=h.z337ya) **7**

# I Project specification

*<Aici de obicei se prezinta pe scurt o idee principal a proiectului si niste specificatii in mare. Scris cu Times New Roman font 12. Minimul recomandat ar fii pana la tagul de incheiere rosu*

This project represents a Formula 1 race ticket purchasing website, where fans from all over the world will be able to buy tickets for the upcoming races, for any of the circuits, and in various packages.

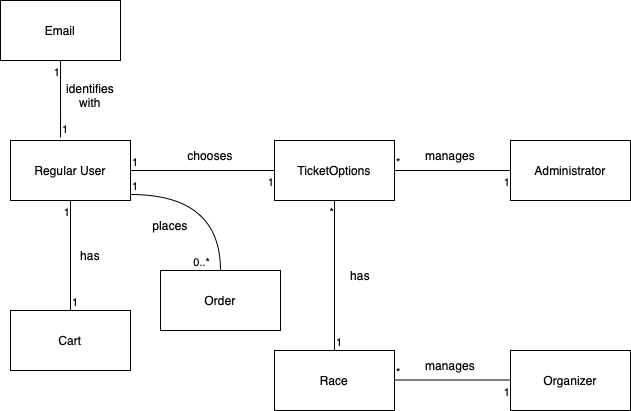
The project will be a web application (a website), accessible for any user on the World Wide Web. Proposed functionalities include displaying all available races to the user, with dates and location, adding such races to the cart, and finally purchasing the tickets. The website administrator will be able to edit existing races, to match any changes incoming in the current season.

There should be three types of user: regular user, administrator and race organizer. As mentioned above, the administrator should be able to create, edit or delete ticketing options for the races, according with the changes in the ongoing Formula 1 season (some F1 calendar changes are made during the season), the race organizer is able to create, update, delete races, according to the same calendar changes, while the user should be able to watch the entire calendar of the ongoing F1 season, along with the weekend programme and purchase tickets for their preferred events.

*/>*

## 1.1 Domain Model Diagram

*< Schema Domain Model a aplicatiei. Se va discuta la laborator. />*

**

# II Use-Case model

*< Se va scrie o mica introducere./>*

Further we will observe the use cases available for this application, tailored to the needs of the various users we have. They will be provided in a detailed manner, to further discover the way the application should work in certain situations.

## 2.1 Users and stakeholders

Among the users we can mention the regular user, who uses the application to obtain F1 race tickets and the administrator and the race organizer, who both have to manage certain aspects of the race and ticket options availability.

Notable stakeholders are the project manager, overseeing the development of the application, the software architect, providing the proper architecture for the program, the developers, involved in actually designing and implementing the project.

## 2.2 Use-Case identification

*< Aici se vor prezenta 3-4 use-case-uri mai importante din applicatie dupa urmatorul model*

***Nume Use case/Use case name: < Nume use-case>.***

***Nivel/Level: < User-Goal, Subfunction, Summary > .***

***Actor principal/Main actor: < Actorul scenariului> .***

***Scenariul principal de success/Main success scenario: <Descriere detaliata a scenariului>.***

***Extensie/Extension: <Un caz particular al scenariului, fie pozitiv sau negativ>***

*Se va discuta la laborator mai detaliat.*

Use case: Overview calendar, race and ticket options

Level: Subfunction

Main Actor: Regular user

Main success scenario: The user logs into the application and is able to see all races of the ongoing season. By selecting a particular race the user is able to see the ticket options for that race, along with the weekend plan (other special events during the race weekend)

Extension: If the race has already taken place, the user will not have any ticket purchase options.

Use case: Manage races in the calendar

Level: User-goal

Main Actor: Race organizer

Main success scenario: The race organizer is able to log into the application and manage currently existing races (created by themselves), or create a new race.

Extension: If the organizer has no races organized, he will not be able to manage any of the other existing races, as they are not their own.

Use case: Manage ticketing/weekend planning for races

Level: User-goal

Main Actor: Administrator

Main success scenario: The administrator is able to log into the app and manage the ticketing options and weekend planning for the races, in order to show the user the actual available spots on the circuit, as well as the other activities held during the race weekend.

Extension:

Use case: Purchasing race tickets

Level: User-goal

Main Actor: Regular User

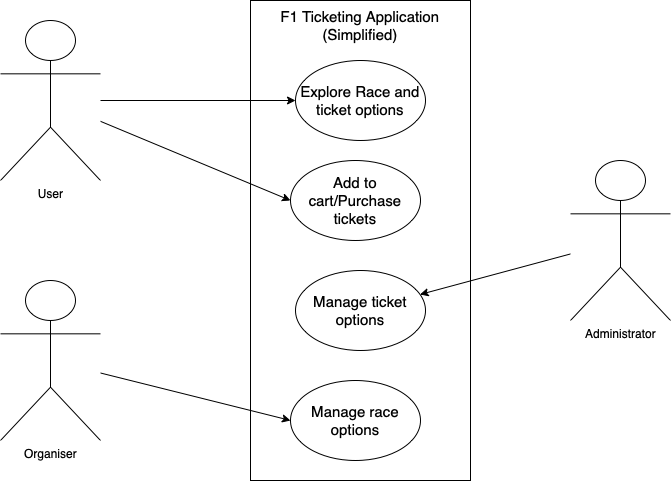
Main success scenario: The user is able to add to cart and then purchase the selected tickets from an available race. By doing that, less spots will be available for that particular race for the particular area chosen by the user.

Extension: If all the races are fully booked, the user will not be able to purchase any tickets, but they can still get information about the race weekend activities if interested to watch on TV.

*/>*

## 2.3 UML Use-Case diagram

*< Diagrama UML cu toate Use-Case-urile aplicatiei/>*



# III Architectural design

*< Se va scrie o mica introducere./>*

## 3.1 Conceptual architecture

*<In acest capitol se vor prezenta arhitectura proiectului. Este o aplicatie web sau desktop. Are baze de date? Cate componente are? Client-Server? Etc*

*8-10 randuri*

*/>*

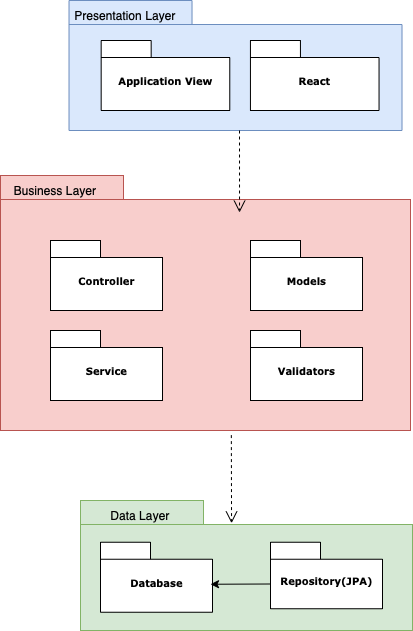
The F1 ticketing application will be a web application, available online for all F1 fans who wish to participate in any of the available racing weekends during the current year.

This application will use a database in order to store the various information, such as user accounts, user tickets purchased, administrator and organizer data, calendar information and also race-specific information, which will further be presented to the user. The database has a multitude of tables, designed to hold the specific information for the main components of the application.

The architecture chosen for this application is a layered architecture, as it fits the structure and use-cases of the application best. We will need to access a database, perform various operations and also provide a user interface, thus making a layered architecture a perfect choice for a clean and easy to understand implementation of the project.

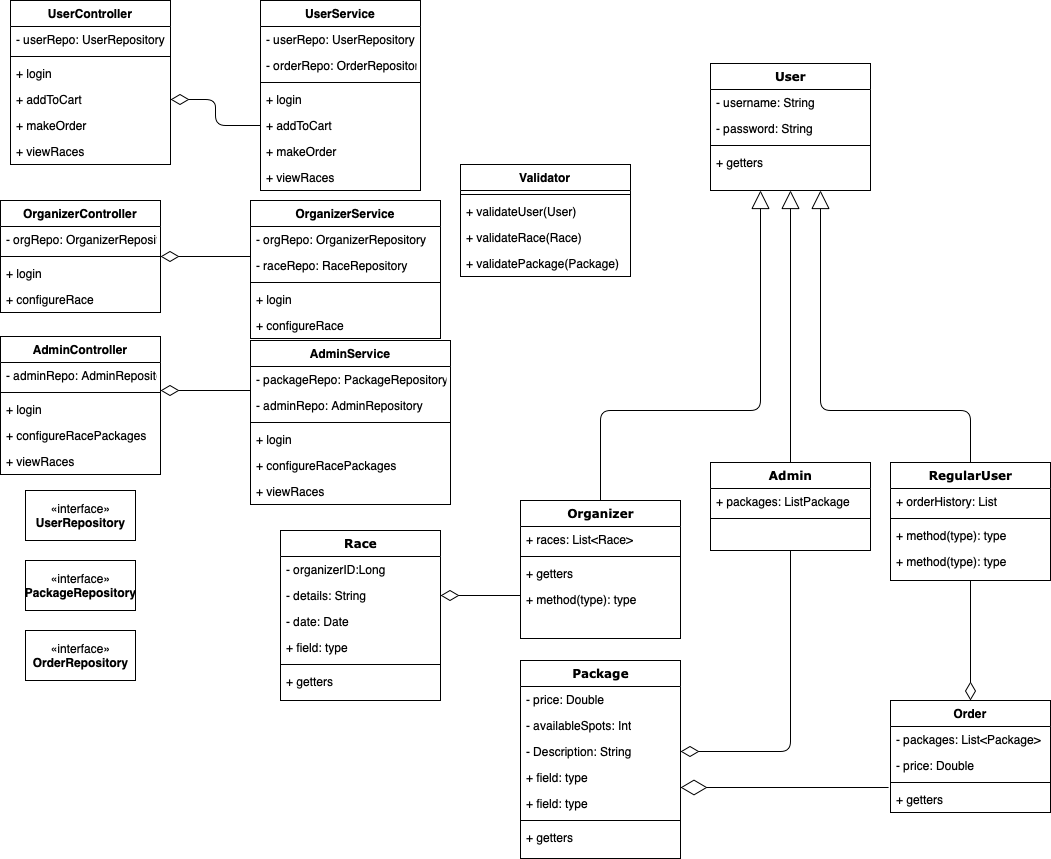
## 3.2 Package diagram

*< (Package Diagram)/>*

**

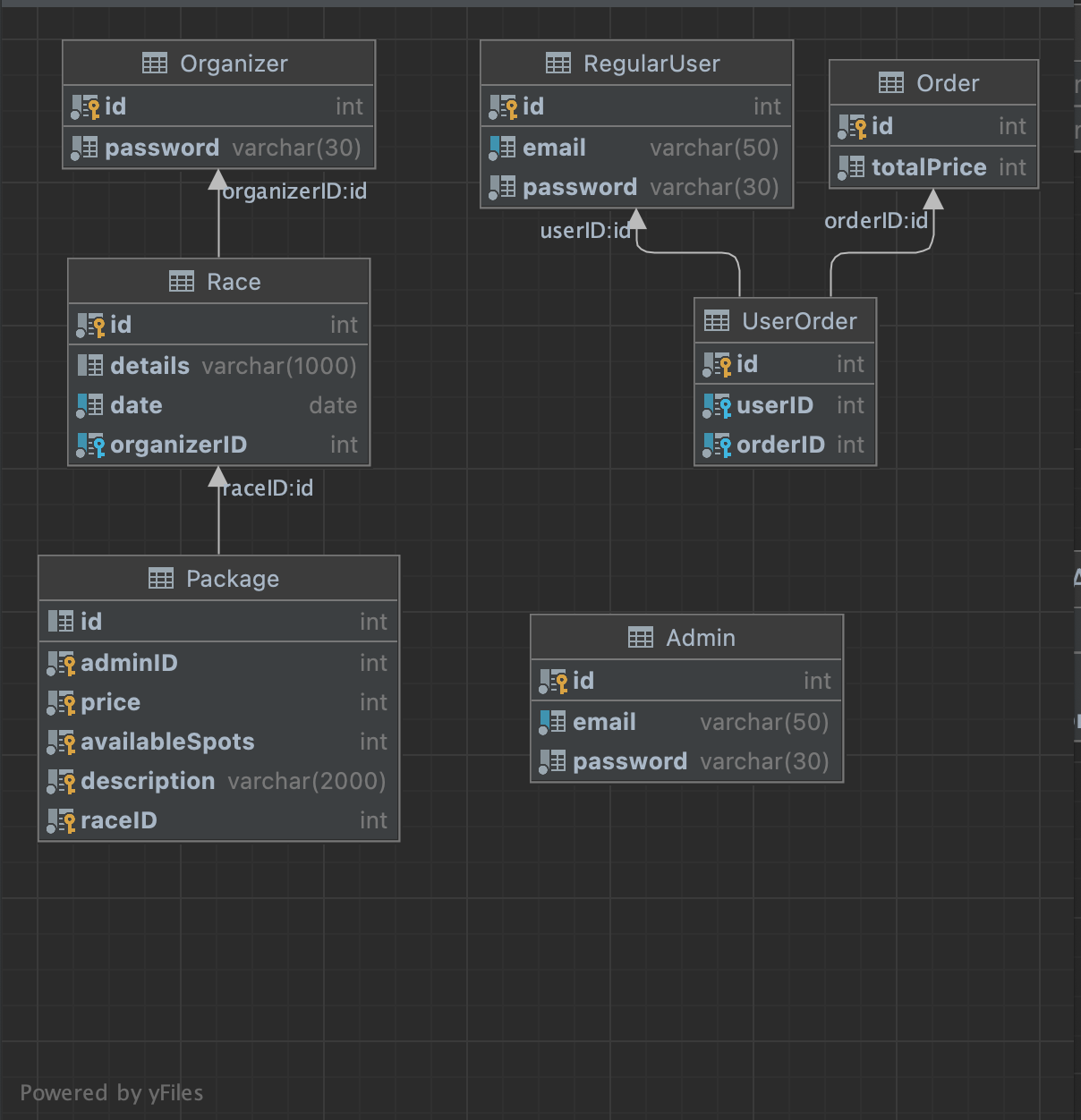
## 3.3 Class diagram

*< (Class Diagram)/>*



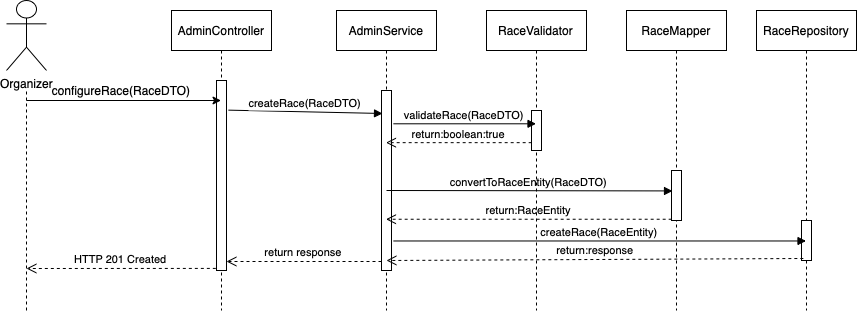
## 3.4 Database (E-R/Data model) diagram

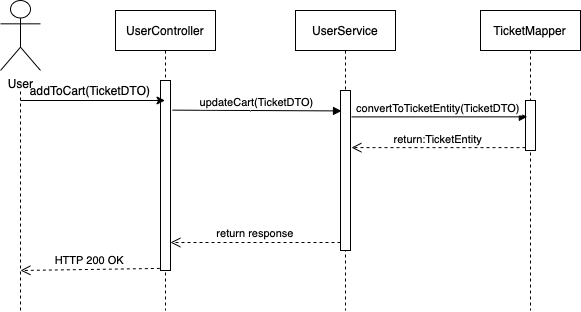
*< (Data Model)/>*

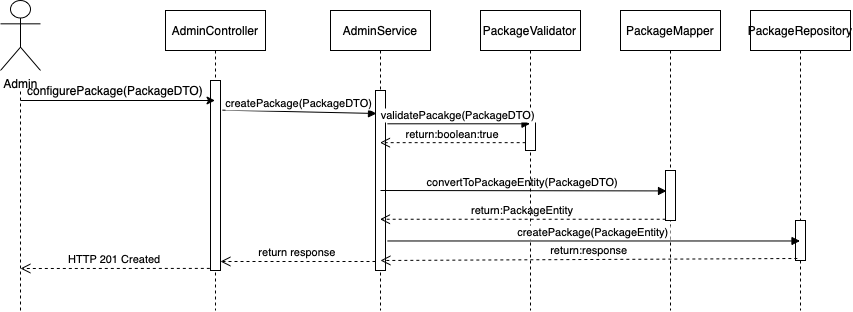
**

Here we can see a general view of the database diagram.

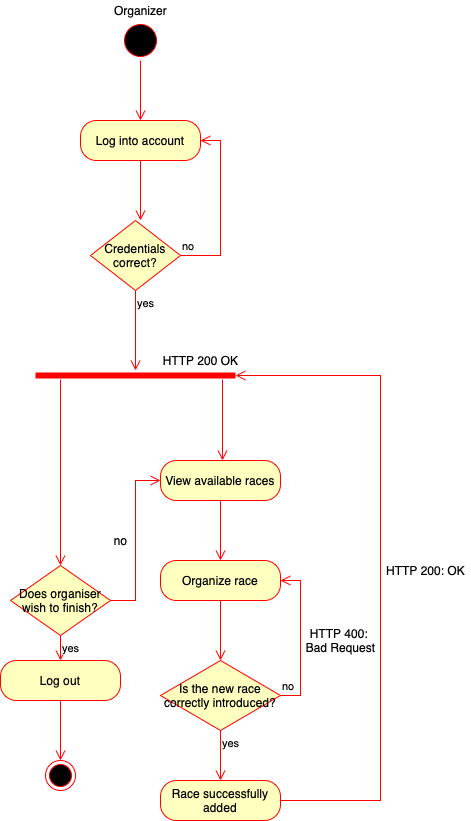
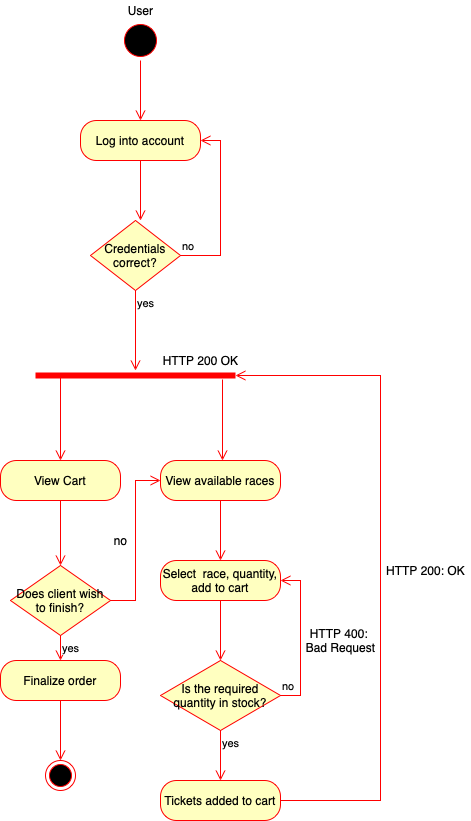
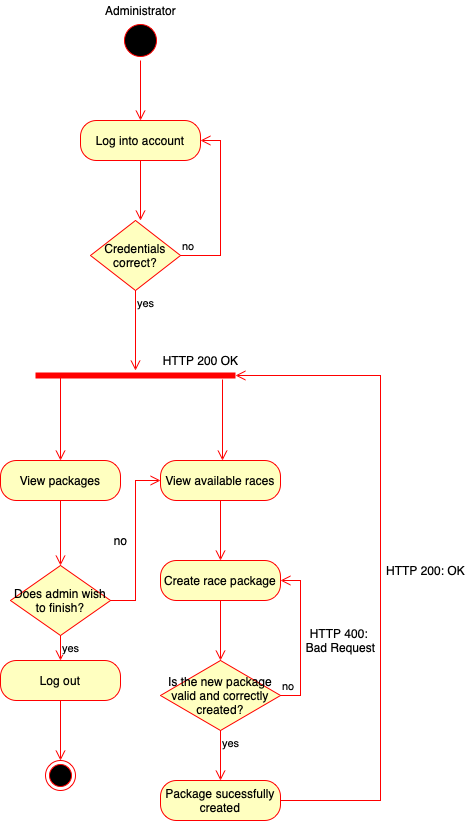
## 3.5 Sequence diagram

*< (Sequence Diagram)/> *

**

**

## 3.6 Activity diagram

*< (Activity Diagram)/> *

# IV Supplementary specifications

*< Se va scrie o mica introducere./>*

## 4.1 Non-functional requirements

*< Specificatiile non-functionale ale aplicatiei. Se va discuta la laborator./>*

The application should respect the following non-functional requirements:

* Availability for any internet user – the website should be accessible all over the globe, by any person interested in the services offered.
* Security of user accounts and data – the security of the application is provided by the request of an email and password for logging in; otherwise, user data such as orders previously made are private and only viewed by the logged user.
* Generally good performance of application – the system should run smoothly, without any issues.
* Accessibility across devices – the user should be able to access the web application by phone, tablet or personal computer.

## 4.2 Design constraints

*< Se va discuta la laborator./>*

The programming language used for the implementation of this application is Java, with the help of the Spring framework. The front end of the application will be implemented with the help of HTML and CSS.

The database is created in MySQL and accessed with the help of the aforementioned Spring framework.

## 4.3 API Documentation

POST 8081/user/register

Request: body(JSON): {

“name” : ”someName”,

“password”: “123”

“email” : “a@yahoo.com”

}

Responses:

* 200 OK – New user was successfully registered
* 400 BAD REQUEST
* 500 INTERNAL SERVER ERROR
* 502 BAD GATEWAY

POST 8081/user/LOGIN

Request: body(JSON): {

“name” : ”someName”,

“password”: “123”

}

Responses:

* 200 OK –user was successfully logged in
* 400 BAD REQUEST
* 500 INTERNAL SERVER ERROR
* 502 BAD GATEWAY

POST 8081/user/addToCart/{packID}

Request: PathVariable {packID} – ID of the package to be added to cart

Responses:

* 200 OK – Pack was successfully added to cart
* 400 BAD REQUEST
* 404 PACKAGE NOT FOUND
* 500 INTERNAL SERVER ERROR
* 502 BAD GATEWAY

POST 8081/user/order/{userID} – ID of the user placing the order

Responses:

* 200 OK – Order was successfully placed
* 400 BAD REQUEST
* 404 USER NOT FOUND
* 500 INTERNAL SERVER ERROR
* 502 BAD GATEWAY

GET 8081/user/order/{userID} – ID of the user whose order history we are retrieving

Responses:

* 200 OK – Order History
* 400 BAD REQUEST
* 404 USER NOT FOUND
* 500 INTERNAL SERVER ERROR
* 502 BAD GATEWAY

POST 8081/admin/addPackage

Request: body(JSON): {

“name” : ”someName”,

“price”: “123”,

“raceID” : “1”,

“raceIncluded” : ”y/n”,

“standard”: “DIAMOND”,

“noOfTickets” : “10”,

“id” : “0”

}

Responses:

* 200 OK – New package was successfully added
* 400 BAD REQUEST
* 500 INTERNAL SERVER ERROR
* 502 BAD GATEWAY

POST 8081/admin/addRace

Request: body(JSON): {

“name” : ”someName”,

“location”: “123”

}

Responses:

* 200 OK – New race was successfully added
* 400 BAD REQUEST
* 500 INTERNAL SERVER ERROR
* 502 BAD GATEWAY

# V Testing

*< Se va discuta la laborator./>*

## 5.1 Testing methods/frameworks

## 5.2 Future improvements

# VI Bibliography