TQS: Product specification report

Filipe Gonçalves [98083], Gonçalo Machado [98359], Catarina Oliveira [98292]

v2022-05-24

1 Introduction

1.1 Overview of the project

The objectives of this assignment, in the scope of the TQS course, are to develop a viable software product and the specification and enforcement of a Software Quality Assurance (SQA) strategy, applied throughout the software engineering process.

For the general terms of the product as well as our specific product, it was defined that the project should implement a digital marketplace for a “last-mile” delivery , similar to Uber Eats or Glovo.

Our product named LegoLiveries is a digital marketplace for selling Lego sets, which are interlocking plastic bricks manufactured by the Lego Group [1].

1.2 Limitations

In the multiple sprints we had, there were some limitations and some problems we went through.

Firstly, having a group with three elements instead of four, made each element of the team more responsible and more occupied, as they had to do a little more work than supposed to.

Secondly, we thought of creating a favorite tab where the client could have their favorite products, but we didn't have the time to implement it.

Finally, we had some problems during the Functional Testing, as we were working with modals and the GitHub Actions. For the modals, we had some problems because they took too long to close, or they didn't close when asked too, and we had to do some extra steps for the test to continue. For the GitHub Actions, as we register, or change pages, we need to create a little bit of time between some steps, so as to let the redirect work, or the button we were trying to click was out of scroll view, and had to be brought back up to view.

2 Product concept

2.1 Vision statement

Legoliveries will be used to sell and deliver Lego sets. Users will be able to choose from a variety of Lego sets, as well as mark sets as favorites and schedule the deliveries, which could be ASAP (as soon as possible) or at a specific time defined by the user. After the purchase, the user could see information about the rider that would deliver the Lego set as well as follow the rider through GPS.

The concept of delivering Lego sets was suggested by one of our group members, Filipe, who thought it was a good idea for people that were stuck at home and did not have much to do. After some brainstorming, we all agreed with his idea and understood that it could also be useful for people that did not have much free time to go shopping or for an unexpected event, like a friends' gathering or a birthday party, where one could not leave to go buy a Lego set.

We also thought of improving our idea, and depending on the lego set, the user could pay an extra fee and have the set built by the rider at home. As this is an extra and we didn't have much time to implement this feature, we focused on the other use cases.

2.2 Personas

João is a 27 year old man that lives alone in an apartment. João works as a human resources manager and is usually working from early in the morning to late in the afternoon. Due to his work schedule, João does not have much time at home and when he does, he likes to spend it peacefully, by reading a book or doing puzzles or Lego sets.

Motivation: João would like a way to get Lego sets delivered to his home whenever he wanted.

Margarida is a 45 year old woman that lives with her husband and her 9 year old son. Margarida works as a real estate agent and when she is not working she is at home taking care of her son or going out with friends. Margarida's son is a very energetic kid with a very active imagination that really likes to build and design whatever he is thinking, which he can do by drawing, painting or with Legos.

Motivation: Margarida would like a way to search, take note, buy and get Lego sets delivered to her.

Tiago is a 22 year old man that lives with his parents in the suburbs. Tiago has just finished university and is looking for jobs in his area, but in the meantime he wants to earn some money, so he starts to work as a rider.

Motivation: Tiago would like to set himself has available or unavailable, select and order and check his reputation

2.3 Main scenarios

Scenario 1:

- João has coronavirus and can’t leave his house, and is in the mood to build a Lego set
- João can’t go to the store and buy a lego set by himself
- João will use our application to buy and order a lego set
- João can now play with his lego set at home

Scenario 2:

- Margarida needs to buy a lego set for her son, as its his birthday and she didn't buy it sooner
- Margarida can't leave her house, as the party is starting
- Margarida will use our application to buy and order a lego set
- Margarida now has a lego set that she can gift her son for his birthday

Scenario 3:

- João works all day and does not have any way to enjoy the night, as he does not like to go clubbing
- João doesn't have time or will to go to a store and buy some legos
- João will use our application to buy and order a lego set
- João can now enjoy his night every once in a while

Scenario 4:

- Margarida is thinking of buying a lego set for her son
- Margarida knows her son likes superheroes from Marvel
- Margarida will use our application to filter the lego sets from Marvel and to buy and order the lego set
- Margarida now has a lego set of Marvel superheroes for her son

Scenario 5:

- Tiago was at a job interview all morning and just finished lunch
- Tiago has nothing planned for the rest of the day, so he wants to work
- Tiago will use the rider application to set himself as available for deliveries
- Tiago is now ready to start working and will wait for their orders

Scenario 6:

- Tiago has been working has a rider for a few weeks
- Tiago wants to check if the clients are satisfied with his deliveries
- Tiago will use the rider application to check his reputation
- Tiago can now see his reputation and comments made about his deliveries

Scenario 7:

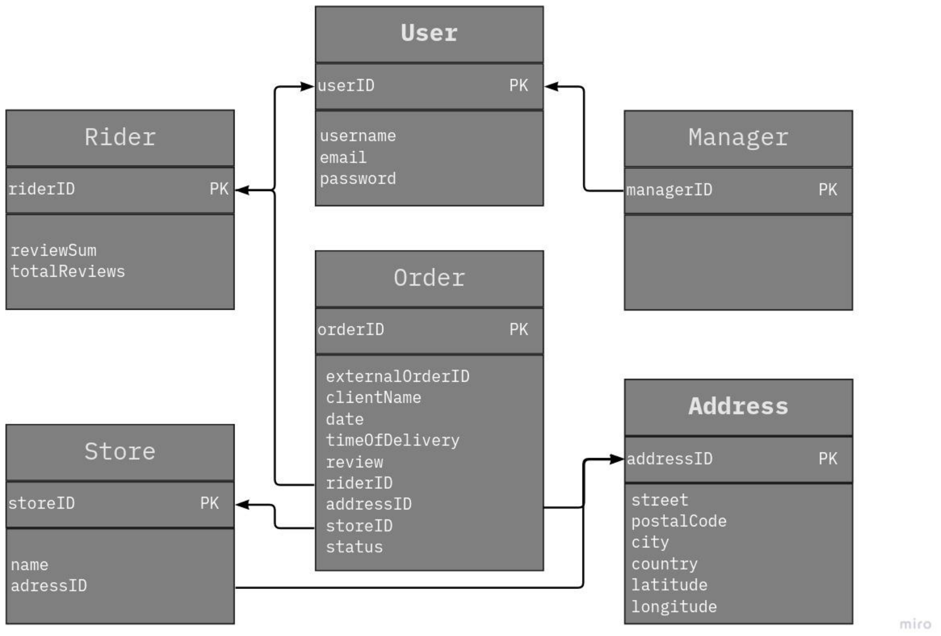
- Tiago is unemployed and wants to work
- Tiago will register in the riders application
- Tiago will start receiving orders and start working

2.4 Project epics and priorities

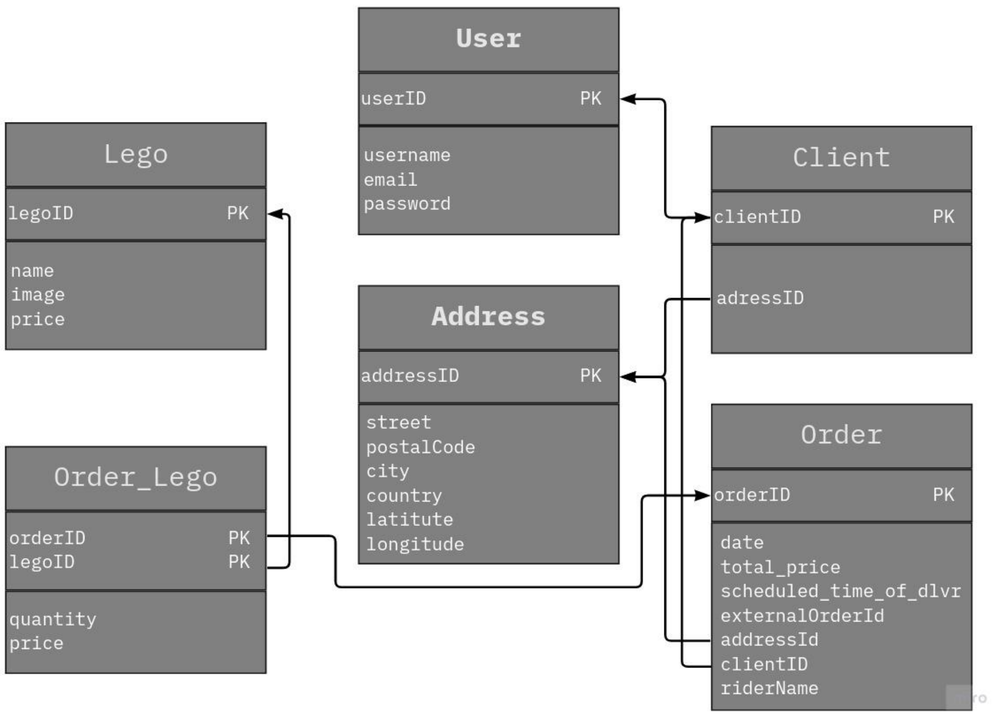
The following epics are in decreasing order from most urgent to least urgent:
- Product concept and planning
- Functional engine backend
- Functional Legoliveries backend
- Functional Legoliveries web app
- Functional engine management board
- Fully implemented engine backend
- Fully implemented Legoliveries backend
- Fully implemented Legoliveries web app
- Fully implemented engine management board
- Minimal riders app
- Functional riders app
- Fully implemented riders app

3 Domain model

As we can see from the next picture, the general service is supported with a database by **MySQL**. The database model has 6 different entities: **User**, **Manager**, **Address**, **Rider**, **Order** and **Store**. We can create new stores, as this is a service used by different specific platforms, we can see our orders and then the different types of users. The idea was to make the most generic entities possible, for the engine to be able to include other services.



As we can see from the next picture, the specific service is supported with a database by **MySQL**. The



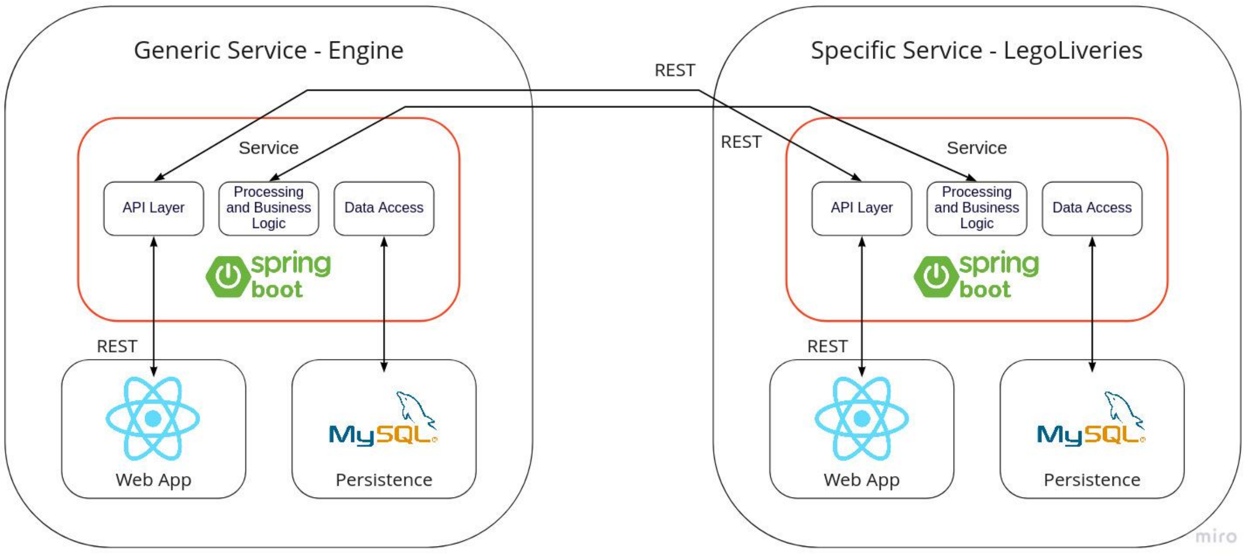
database model has 6 different entities: **User**, **Client**, **Address**, **Order**, **Order_Lego** and **Lego**.

4 Architecture notebook

4.1 Key requirements and constraints

While choosing an architecture, we wanted to make sure that some key points would be served:

- The system cannot let non-authorized users to access personal information
- The system must be deployed in an external service, using a Virtual Machine
- The system must have a platform for a client to place an order
- The system must have a platform for managers, for administration purposes, where can be seen statistical data
- The system must be equipped with a robust implementation as to not happen anything malicious or unusual
- The platforms the system must have, must be for both web surfing and app usage



4.2 Architectural view

Data persistence can be guaranteed using the **MySQL** database, both in the specific service, LegoLiveries, and generic service, Engine. We can access the data directly from **SpringBoot**, using the respective Models, Repositories and Controllers, process it and send it to the front end using a **Rest Controller**. The choice of using **MySQL** was done by the whole team, while evaluating the usage of other technologies, like **Postgres**.

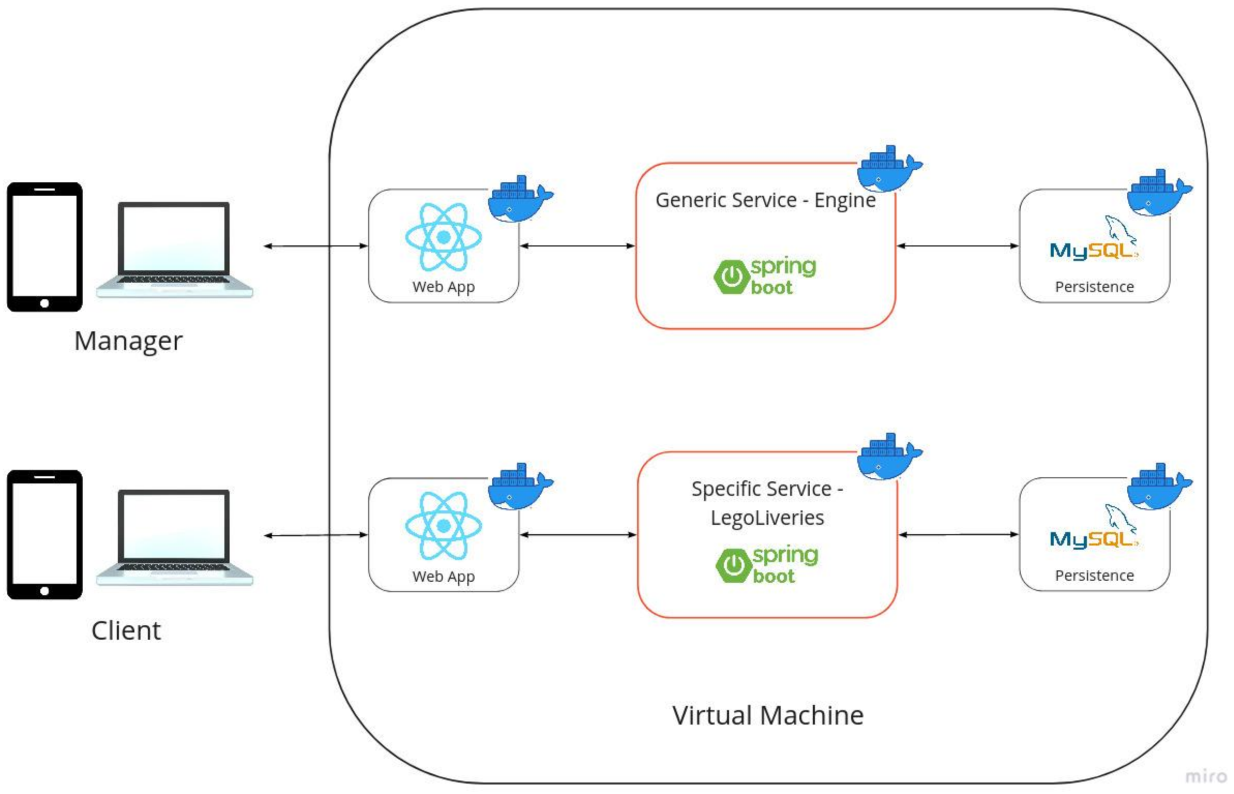
For the Web Applications, we decided to use **React** for both the management dashboard and the specific service frontend. From the different options, we chose **React** for its simplicity and organization, as well as for being very easy in creating a **Progressive Web App**, a web application that when deployed can turn into a mobile app on request of the client.

We used **SpringBoot** for the backend of both services, as it was mandated by the project.

These different components will interact with each other as seen in the diagram, using HTTP requests.

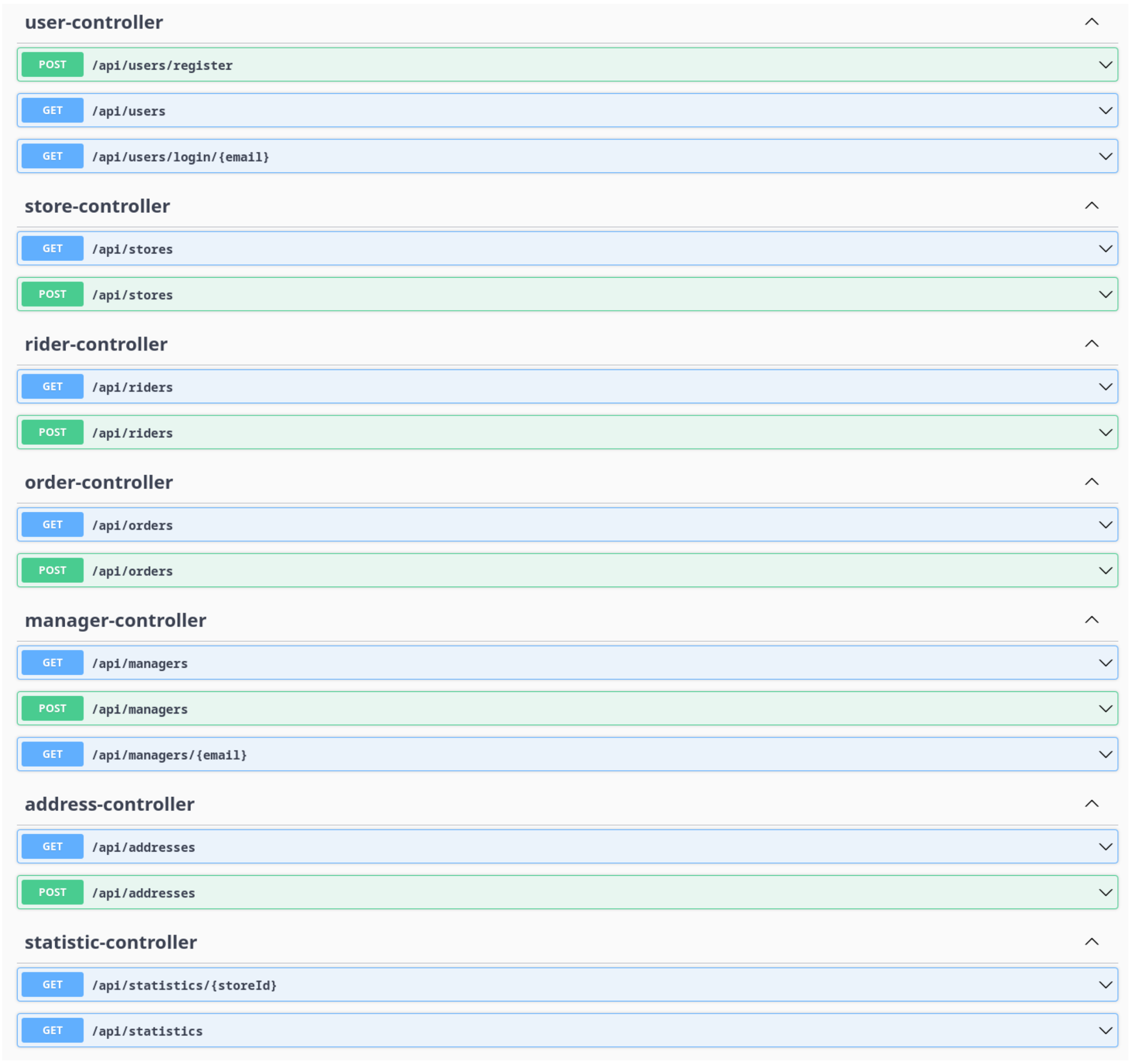
4.3 Deployment architecture

The deployment of the project was done using the Virtual Machine given to us by the professors. Each component was first executed with the help of **Docker**, so both the continuous deployment and the initial deployment was simpler, as it was only needed to reconstruct the containers and build them again.



5 API for developers

The APIs were built using the **@RESTController** annotation from **SpringBoot**. We can see in the next images their endpoints and information about them. We used an extension for the project, **swagger** [2], which turns our APIs into their respective information and statistics.


user-controller

POST /api/users/register

GET /api/users

GET /api/users/login/{email}

store-controller

GET /api/stores

POST /api/stores

rider-controller

GET /api/riders

POST /api/riders

order-controller

GET /api/orders

POST /api/orders

manager-controller

GET /api/managers

POST /api/managers

GET /api/managers/{email}

address-controller

GET /api/addresses

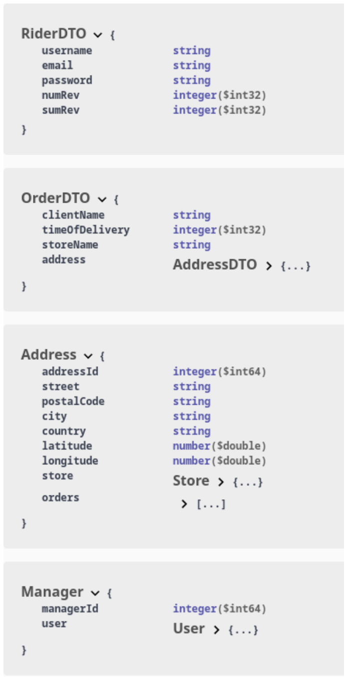
POST /api/addresses

statistic-controller

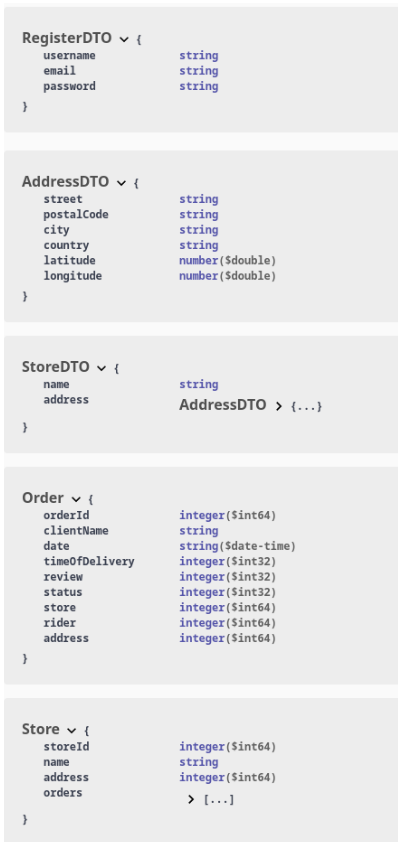
GET /api/statistics/{storeId}

GET /api/statistics

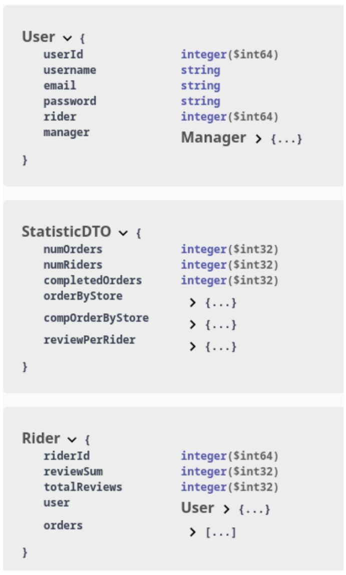
The Engine endpoints were mapped from 7 different REST Controllers, using different Data Object Models and Models.



```
RiderDTO ⌄ {
   username        string
   email           string
   password        string
   numRev          integer($int32)
   sumRev          integer($int32)
}

OrderDTO ⌄ {
   clientName      string
   timeOfDelivery  integer($int32)
   storeName       string
   address         AddressDTO > {...}
}

Address ⌄ {
   addressId       integer($int64)
   street          string
   postalCode      string
   city            string
   country         string
   latitude        number($double)
   longitude       number($double)
   store           Store > {...}
   orders          > [...]
}

Manager ⌄ {
   managerId       integer($int64)
   user            User > {...}
}
```

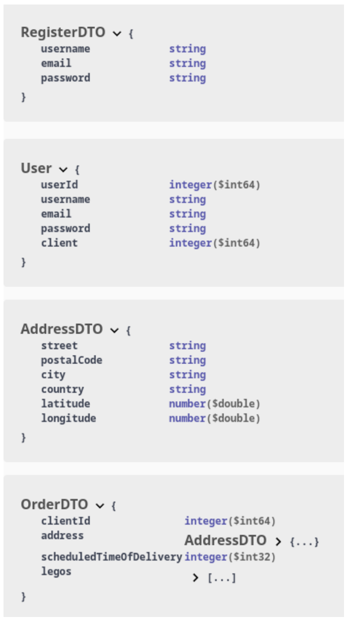


```
RegisterDTO ⌄ {
   username        string
   email           string
   password        string
}

AddressDTO ⌄ {
   street          string
   postalCode      string
   city            string
   country         string
   latitude        number($double)
   longitude       number($double)
}

StoreDTO ⌄ {
   name            string
   address         AddressDTO > {...}
}

Order ⌄ {
   orderId         integer($int64)
   clientName      string
   date            string($date-time)
   timeOfDelivery  integer($int32)
   review          integer($int32)
   status          integer($int32)
   store           integer($int64)
   rider           integer($int64)
   address         integer($int64)
}

Store ⌄ {
   storeId         integer($int64)
   name            string
   address         integer($int64)
   orders          > [...]
}
```

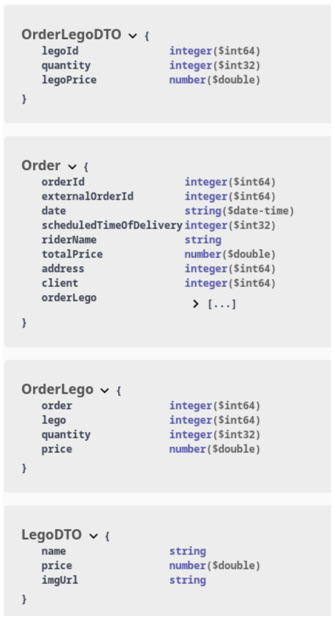


```
User ⌄ {
   userId           integer($int64)
   username         string
   email            string
   password         string
   rider            integer($int64)
   manager          Manager > {...}
}

StatisticDTO ⌄ {
   numOrders        integer($int32)
   numRiders        integer($int32)
   completedOrders  integer($int32)
   orderByStore     > {...}
   compOrderByStore > {...}
   reviewPerRider   > {...}
}

Rider ⌄ {
   riderId          integer($int64)
   reviewSum        integer($int32)
   totalReviews     integer($int32)
   user             User > {...}
   orders           > [...]
}
```

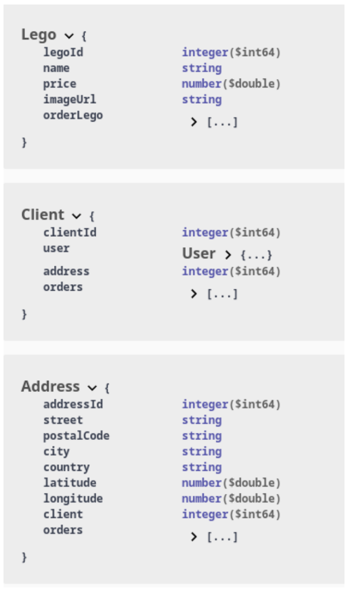
As for the specific service rest API, we have 5 different REST Controllers, and more Data Object Models and Models.



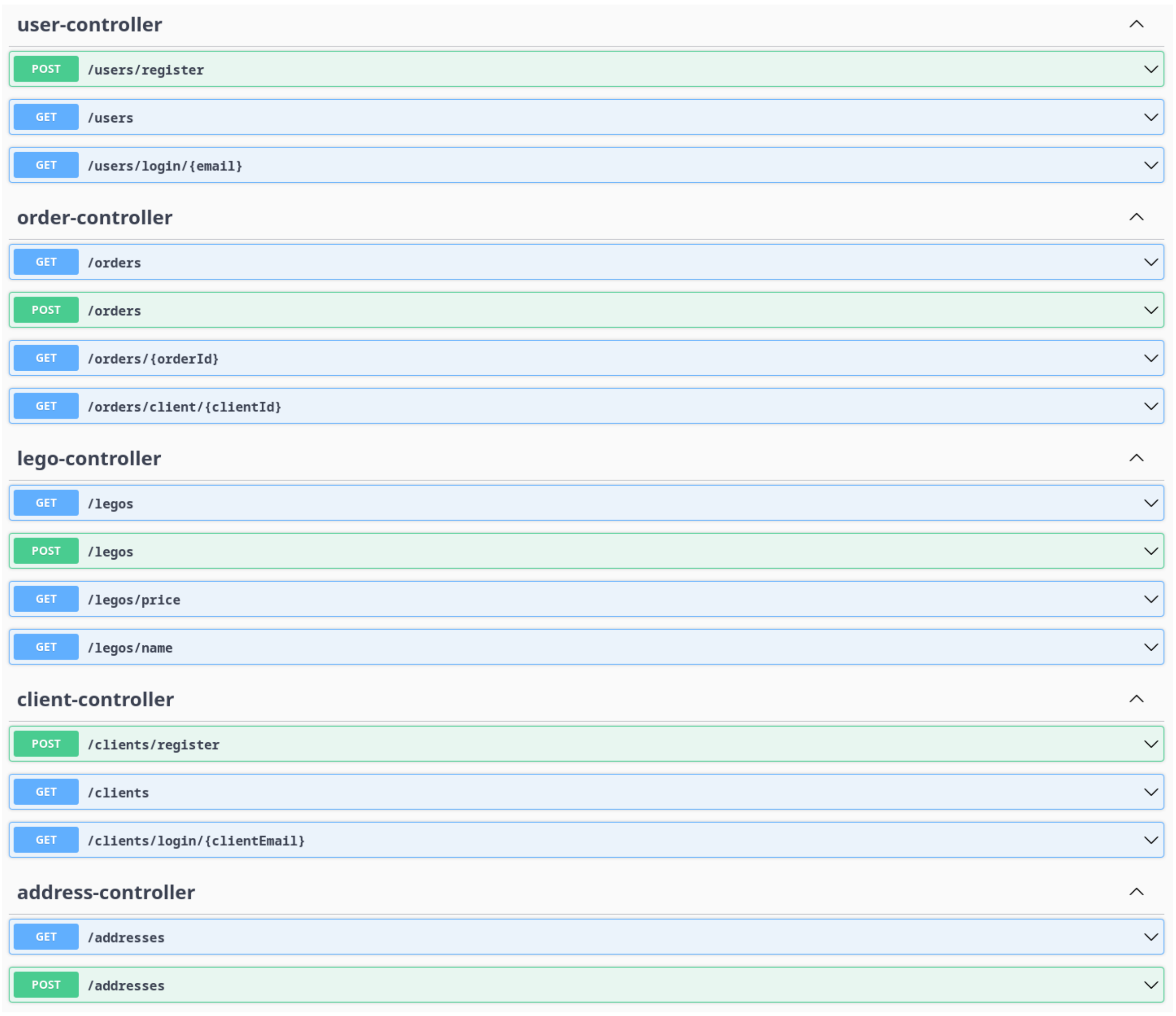
```
RegisterDTO ⌄ {
   username        string
   email           string
   password        string
}

User ⌄ {
   userId          integer($int64)
   username        string
   email           string
   password        string
   client          integer($int64)
}

AddressDTO ⌄ {
   street          string
   postalCode      string
   city            string
   country         string
   latitude        number($double)
   longitude       number($double)
}

OrderDTO ⌄ {
   clientId               integer($int64)
   address                AddressDTO > {...}
   scheduledTimeOfDelivery integer($int32)
   legos                  > [...]
}
```



```
OrderLegoDTO ⌄ {
   legoId          integer($int64)
   quantity        integer($int32)
   legoPrice       number($double)
}

Order ⌄ {
   orderId                 integer($int64)
   externalOrderId         integer($int64)
   date                    string($date-time)
   scheduledTimeOfDelivery integer($int32)
   riderName               string
   totalPrice              number($double)
   address                 integer($int64)
   client                  integer($int64)
   orderLego               > [...]
}

OrderLego ⌄ {
   order           integer($int64)
   lego            integer($int64)
   quantity        integer($int32)
   price           number($double)
}

LegoDTO ⌄ {
   name            string
   price           number($double)
   imgUrl          string
}
```



```
Lego ⌄ {
   legoId          integer($int64)
   name            string
   price           number($double)
   imageUrl        string
   orderLego       > [...]
}

Client ⌄ {
   clientId        integer($int64)
   user            User > {...}
   address         integer($int64)
   orders          > [...]
}

Address ⌄ {
   addressId       integer($int64)
   street          string
   postalCode      string
   city            string
   country         string
   latitude        number($double)
   longitude       number($double)
   client          integer($int64)
   orders          > [...]
}
```


user-controller

POST /users/register

GET /users

GET /users/login/{email}

order-controller

GET /orders

POST /orders

GET /orders/{orderId}

GET /orders/client/{clientId}

lego-controller

GET /legos

POST /legos

GET /legos/price

GET /legos/name

client-controller

POST /clients/register

GET /clients

GET /clients/login/{clientEmail}

address-controller

GET /addresses

POST /addresses

6 References and resources

[1] Lego Home, https://www.lego.com/en-pt
[2] Swagger, https://swagger.io/