Uberoo - API

Toutain - Fornali - Swiderska - Benza

SOA - October 2018

I - Chosen service style :

Considering that the subject did not require to implement a user interface, it seemed important to us to dissociate our services access from their implementation.
To satisfy all the services required by the MVP, we only needed to use the HTTP verbs POST and GET. We wanted a simple and uniform architecture for our operations and a simple and known semantic. For this reason, we chose to build a REST API.
In this particular case, the defaults of REST weren't significative compared to it's advantages, so it's seemed to be the most adapted choice to meet the given use case.

II - API design :

The objective of the project was not to encounter language-syntax problems, but instead, to learn and understand resource-oriented application functioning by developing one.
Moreover, we found that there are a lot of good, open and easy to understand frameworks built on Java. Thus, we quite naturally selected the Java framework SpringBoot in order to realize the project.

Spring Boot makes it easy to create stand-alone, production-grade Spring based Applications that you can "just run". Along with that we chose to use MongoDB for data storing. We thought that this database management system was enough powerfull to fullfill the objectives of this project and more.
These technologies works well together and thanks to this choice we have been able to create a walking skeleton early in the imparted time.

We thought and chose a certain architecture basis at the start and did not derived from it since. Here it is:

1 - "Controller" and request mapping structure :

The Controller module (which is only a package in our implementation) contains our Web Service entry points. It proposes different entry points, each one being pertinent for the required subject scenario.
We chose to establish the following "logic" for our request URLs construction:
- "DSA" is our request mapping basis which means "Delivery Service API". All procedures request mapping inheritates from it.

- Food retrieving [GET]:

> /DSA/CATALOG to GET the offered catalog (simple dish plus menus)
> /DSA/FOOD to GET the simple dishes catalog
> /DSA/MENU to GET the menus catalog

If you want more information on one particular simple dish or menu, you can use the following:

> /DSA/FOOD/{foodName}
> /DSA/MENU/{menuName}

You also have to add the desired delivery address in order for the system to compute the "Estimated Time before Arrival" and to obtain it in the response.

- Order retrieving [GET]:
The users of these following entry points are:

* The restaurant, which want to see its list of orders.
> /DSA/RESTAURANT/{restaurantName}/ORDERS
* The coursier, which also want to see its assigned orders.
> /DSA/COURSIER/ORDERS/{coursierName}

- Creating a food order [POST]:

A user of our API wants to make a food order.

He has to use this entry point :

> **/DSA/FOOD/{clientName}**

The simple dishes / menus items wanted by the user have to be inquired under JSON format in the request body in order to be understood by the system.

Same as before, the delivery address has to be inquired in order to compute the ETA but of course, also for the coursier assigned to the order to know where to deliver.

- Notify that an order is ready for delivery [POST]:

When a restaurant finished to prepare an order, it can notifies it to the system by using the following entry point:

> **/DSA/RESTAURANT/ORDERS/{orderId}**

Reminder: A restaurant knows each order identifier thanks to the order retrieving entry point.

Of course, we assume only the restaurant can access to and modify its order. No identifying system is proposed on our API.

- Notify that an order has been delivered [POST]:

Once a coursier has delivered an order, he can notify it to the system by using the following entry point:

> **/DSA/COURSIER/{coursierName}/ORDERS/{orderId}**
>
> He has to inquired his name (we assume there is no namesake) and its assigned order identifier.

- Updating the database:

We also did implement a "tool entry point" that fill the database with wrote scenarios (/src/main/resources/databaseScenarios)

> **/DSA/UPDATE**

2 - Models :

In order to realise the API we created our own data model.
SpringBoot system needs it to relate database repository with Java objects.
Naturally, each entity created represents a user or an item involved in Uberoo.
We thus have:

* Client > "Joe", the user who wants to buy food
* Food > "Honey Goat Cheese Pizza by Titeuf Pizza", a simple dish prepared by a certain restaurant
* Menu > "Child's menu by Kumano", one menu prepare by the Kumano restaurant
* Restaurant > "Kumano", a restaurant
* Coursier > "Dimitri", a coursier that has to deliver assigned orders
* Order > "Order X which content is Y assigned to Dimitri" our representation of a food order

We have more entities mainly used for internal functioning. For instance we do have enums to represent the current status of a coursier / an order. (e.g. AVAILABLE / DELIVERED...)

3 - Repositories :

It represents our database architecture. Each repository can be seen as a table in a SQL database.
For instance the order repository is a space where can be stored orders (Order from our data model).

4 - Services :

We also do have one module containing all of our services.
In our implementation, a service X is built on top of one repository X and proposes access to it.
For instance, the Order Service proposes access to the order repository.
Thus, only the service layer can access to the repository one.

III - API in others styles :

1 - Documents :

If we had use the document style for our API we would have provide the same level of services. Below you can see a representation of what we have in mind, under the XML format :

- User consultation of foods and menus :
  - <verb> consult </verb> => Action to realize
    - <subject> foods </subject> => Target of the action
    - <subject> menus </subject>
    - <subject> all </subject>
    -
- User order of food and/or menus :
  - <verb> order </verb>
    - <food> choice1 </food>
    - <food> choice2 </food>
    - …
    - <menu> choice1 </menu>
    - …
- Consultation of orders from the restaurant :
  - <verb> consult </verb>
    - <subject> orders </subject>
    - <restaurantId> id </restaurantId>
- Update of orders from the restaurant :
  - <verb> update </verb>
    - <subject> orders </subjects>
- Update of orders from the coursier :
  - <verb> update </verb>
    - <subject> order </subjects>
    - <coursierId> id </coursierId>