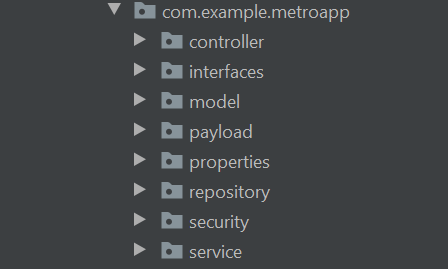
# Implementation

Using Spring Boot framework which has been built by some design patterns like (Dependency injection and MVC Design Pattern) for developing our Back end allows our system structure to be more clarified and organized with low coupling and high cohesion within its subsystems.

As shown below:

Our system contains of some packages , each package contain some classes with the same role and each one of them apply this role in its different subsystem.

Each subsystem contains from a Controller which be called by user in order to use this subsystem, An Interface for the services done by this subsystem and be called by the subsystem controller, A Service class which implelemnt the interface and overwrite its functions, A Model represents an Entity in the database and A Repository responsible for this model in the database.

And there is also other packages used in the whole system (security and payload packages).

## Subsystems and their main functions:

1. Account and user:

* **signUP() :**

This function is responsible for creating user account, it takes a signUpRequest object which contains user data needed to signup as a parameter and check if username or email is already exist and in this case it returns BAD\_REQUEST, it also creates a new customer in stripe (Payment Gateway) and store its customerID which will be used if user want to charge his wallet, then it save this account in the Data Base and returns HttpStatus.OK.

* **login()**:

This function is responsible for user login, it takes a loginRequest object which contains username and password then it authenticate this user and returns user JWT token for this user or return BAD\_REQUEST if there is error with login (wrong username or password).

* **changeUserPassword() – changeUserEmail() – changeUserPhoneNum**() :

Those functions are responsible for change user data and require authentication so they take user current password and the new data user wants to change , and if it is the correct password they return HttpStatus.OK , else : they return BAD\_REQUEST.

1. Payment:

* **createCharge():**

This function is responsible for charging user wallet, it takes chargeRequest object which contains credit card data (cardNum-Date-CVV-Amount) and create a payment using the payment gateway (Stripe is used in our project as a simulation method) and if payment is created successfully it update the user wallet and returns HttpStatus.OK , else it returns BAD\_REQUEST.

1. Ticket:

* **buyTicket():**

This function is responsible for buying a new ticket for user, it takes ticket price as a parameter it checks if user balance is greater or equal the ticket price and then create a new ticket for this user, insert it into Data Base , update user balance and then returns HttpStatus.OK , or returns BAD\_REQUEST if something wrong.

1. Subscription:

* **addSubscripe():**

This function is responsible for adding a new subscription for user, it takes subscription needed data as a parameter it checks if user balance is greater or equal the subscription price and if user have not any other subscriptions and then create a new Subscription for this user, insert it into Data Base , update user balance and then returns HttpStatus.OK , or returns BAD\_REQUEST if something wrong.

* **GetSubscriptionType():**

This function is responsible for get the suitable subscription type based on source and destination stations and the period selected by user, it takes those as parameters and check how many regions in the path between two stations in order to return the suitable subscription type.

1. Station:

* **getClosestStation():**

This function is responsible for getting the closest station for user, it takes the latitude and longitude as parameters then check the Data Base for all the stations and return the closest one for him.

1. Trip:

* **initializeAndBuildGraph():**

This function is responsible for creating a graph of stations (metro map) in order to get shortest path and so on, it access the stations,line and station\_line tabels in Data Base and make the graph based on it.

* **getTripPath():**

This function is responsible for getting the shortest path between two stations, it takes two stations as parameters and return a Map<String, Bool> with the station name and Boolean to check if this station is a change one (from line to other).

* **getNumberOfRegions():**

This function is responsible for getting the number of regions for the shortest path between two stations, it takes two stations as parameters, calculate the shortest path and then return the total number of different regions in this path.

1. Machine:

* **Pass():**

This function is responsible for Metro Machines requests, it needs an ApiKey in order to authorize this request and takes machineRequest object contains the request type (Ticket-Subscription), this request ID (ticketID-subID) and a station name where this machine exist then it call one of two functions:

1. **ValidateTicket():**

This function is responsible for checking ticket request, it takes ticketID and stationName as parameters it checks if ticket id is right and exist in Data Base, its check this is the first request for this ticket (user getting into source station) then add current station into this ticket record and return true, if this is the second request for this ticket (user getting out destination station) then checks if the shortest path between source station and current one is greater than or equal to the maximum stations allowed then it delete the ticket record and return true, Otherwise it returns false.

1. **ValidateSub():**

This function is responsible for checking subscription request, it takes subscriptionID and stationName as parameters it check if subscription id is right and exist in Data Base, it checks if the current station exist in the shortest path between subSourceStation and subDestinationStation, it checks if the endDate for this subscription is after current Date (subscription is not expired), it checks if the number of remaining trips is greater than zero then it checks if this is the first request for this sub (user getting into source station) then update this sub record (in\_use = true) and return true, if this is the second request for this sub (user getting out destination station) then update this sub record (in\_use = false) and decrease the number of remaining trips and return true, otherwise it returns false.

If the called function returns true: then the machine will let user to pass

Otherwise: the machine won’t let user to pass.

## Subsystems for admin WebApp and their main functions

1. Admin:

Responsible for Admin login and authorization before performing any function.

1. Basic Ticket:

Responsible for adding, update and delete ticket types (Ticket Price, Max number of Trips).

1. Basic Subscription:

Responsible for adding, update and delete subscription type (Subscription Price, Max number of Trips, Duration, Number Of Regions).

1. Station:

Responsible for adding, update and delete stations.

## User Mobile Application

A mobile application front end developed by flutter which enables developing an app for both operating systems using the same codebase, makes the development process faster and more efficient and perform at a level compatible with native apps and are winning over other cross-platform technologies.

The mobile application receives user actions and send requests to Back End Rest APIs after authorizing this user through a login request, it saves the returned JWT Token and then pass it as a header for any other request to the APIs in order to perform high security.