**Architecture & Design Principles**

**Angular**

front-end

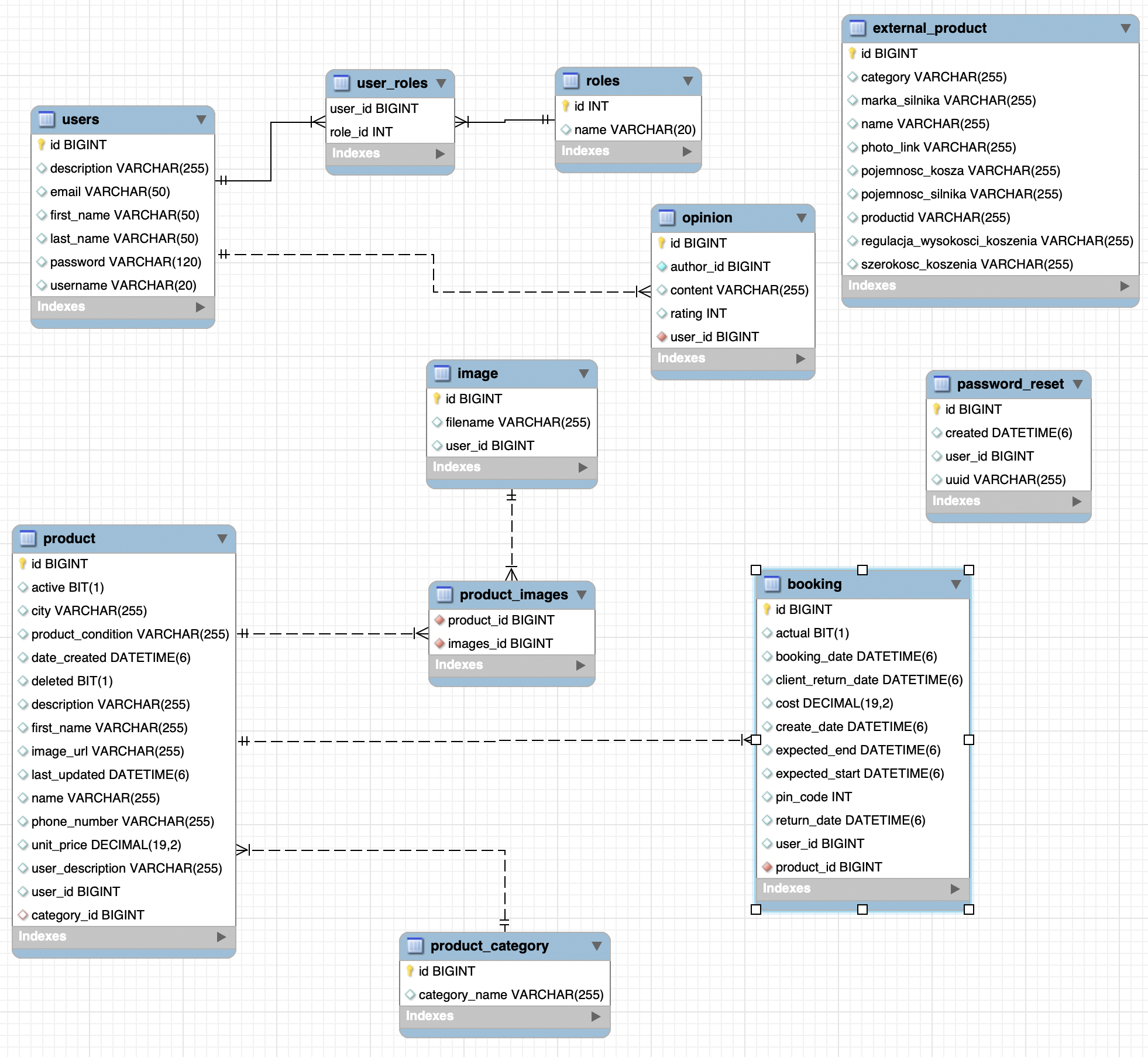
**Spring Boot**

back-end

Database

**Rest API**

In our project as foundation we are using the **Model-View-Controller (MVC)** software design pattern.

We have Angular on the front end, Spring Boot on the back end, and they are communicating with each other using a REST API. And in the background we have a database, which have full CRUD support.

**Database**

The first version of our database structure was very simple. Where we were having only 3 tables in it. So here we had a table product, which had contained information about the product which user as an owner offers for renting.

Nowadays our database structure is more complex. Some short description of tables:

**‘product’** - contains information about the product, which user is offering. Product can contain one or more images of it.

‘**image’** - contains an information about the unique file name and an Id of user, who is the owner of the product. Images are stored on the server.

**‘product\_category’** - is used in order to sort products by category.

**‘booking’** - stores information about booking process between two users.

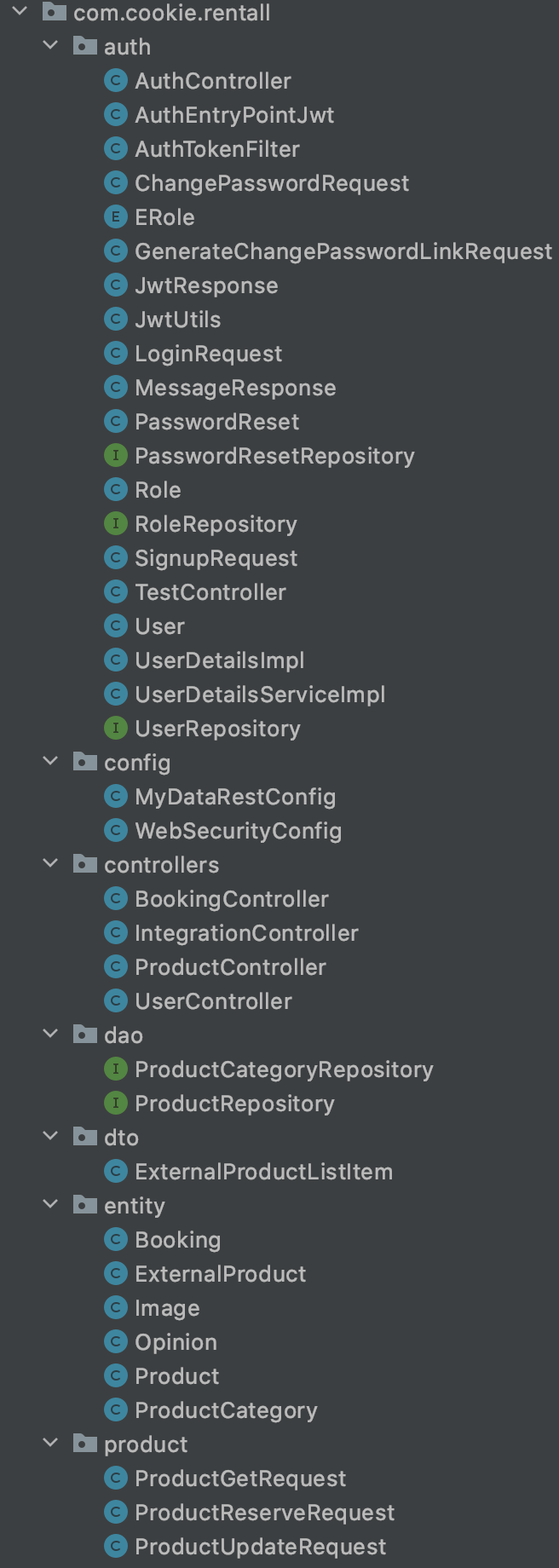
**‘external\_product’** - table, which is the part of integration with MediaExpert.

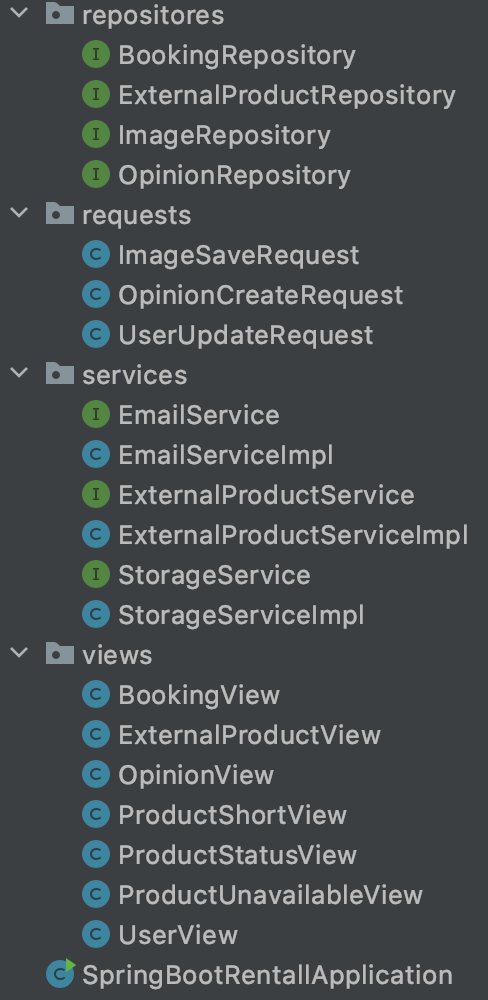
**‘users’** - is used for storing data about the user, such as name, email, description.

**‘opinion’** - stores opinions of other users regarding particular user.

**Spring Boot Back End**

The back-end server uses Spring Boot with Spring Security for JWT Authentication and Spring Data JPA for interacting with database.

**Project Structure**



**Auth package**

Client

Server

Check existing

Save User to database

Post api/auth/signup

{ username, email, role, password}

**User**

**Registration**

return Message(«Registered successfully!»)

Post api/auth/signin

Authenticate

{ username, password}

Create JWT string with a secret

{ username, password }

**User**

**Login**

return JWTResponse

{ token, type, user info, authorities }

Request data with

Check JWT Signature

Get user info & authenticate

Authorize using user’s Authorities

JWT on **Authorization Header**

**Access**

**Resource**

return Response based on **Authorities**

The diagram shows flow of how we implemented User Registration, User Login and Authorization process. Classes in auth package:

* **AuthController** handles signup/login requests
* **AuthEntryPointJwt** implements **AuthenticationEntryPoint**(catch unauthorised error and return a 401 when Clients access protected resources without authentication)

-**AuthTokenFilter** extends **OncePerRequestFilter**(makes a single execution for each request to our API. It provides a **doFilterInternal()** method that we implement parsing & validating JWT, loading User details(using **UserDetailsService**), checking Authorization(using **UsernamePasswordAuthenticationToken**))

-**ChangePasswordRequest** class with password and uuid field with getters/setters.

-**ERole** enum with defined roles.

* **GenerateChangePasswordLinkRequest** model with email field.

- **JwtResponse** { token, type, id, username, email, roles }

* **JwtUtils** provides methods for generating, parsing validating JWT
* **LoginRequest** { username, password }
* **MessageResponse** { message }
* **PasswordReset** model with fields { id, uuid, userId, createdDate }
* **PasswordResetRepository** extends JpaRepository<PasswordReset, Long>
* **Role** model with fields { id, name }
* **RoleRepository** extends JpaRepository<Role, Long>
* **SignupRequest** { username, email, password }

-**TestController** has accessing protected resource methods with role based validations.

* **User** model, it has 5 fields: {id, username, email, password, roles }
* **UserDetails** contains necessary information ( such as: username, password, authorities) to build an Authentication object.

- **UserDetailsImpl** implements **UserDetails**

* **UserDetailsService** interface has a method to load User by username and returns a UserDetails object that Spring Security can use for authentication and validation.

-**UserDetailsServiceImpl** implements **UserDetailsService**

**- UserRepository** extends **JpaRepository<User, Long>**

**Config package**

* **MyDataRestConfig** implements **RepositoryRestConfigurer**
* **WebSecurityConfig** extends **WebSecurityConfigurerAdapter**(We override the **configure(HttpSecurity http)** method from **WebSecurityConfigurerAdapter** interface. It tells Spring Security how we configure CORS and CSRF, when we want to require all users to be authenticated or not, which filter and when we want it to work, which Exception Handler is chosen)

**Controllers package**

* **BookingController** rest controller, defines all the methods used for booking purposes
* **IntegrationController** rest controller, contains method, which returns a data about searched product in MediaExpert.
* **ProductController** rest controller, contains such methods, which checks the authentication process, searching for products by it’s status, and method for adding new product.

-**UserController** stores methods such as cancelReservation, createOpinion.

**Dao package**

* **ProductCategoryRepository** extends **JpaRepository<ProductCategory, Long>**
* **ProductRepository** extends **JpaRepository<Product, Long>**

**Dto package**

-**ExternalProductListItem** model which contains 2 fields { productID, url }

**Entity package**

A package which contains some of the models.

Such as **Booking, ExternalProduct, Image, Opinion, Product, ProductCategory**

**Product package**

* **ProductReserveRequest** a model which contains information about starting and ending reservation dates.

-**ProductUpdateRequest** a model which mainly updates information about the status of the product while booking process

**Repositories package**

Contains repositories, all of them extends CrudRepository: BookingRepository, ExternalProductRepository, ImageRepository, OpinionRepository.

**Requests package**

-**ImageSaveRequest** a model which contains a single field { url }

-**OpinionCreateRequest** a model which contains three fields { userId, content, rating }

-**UserUpdateRequest** a model which contains four fields { email, firstName, lastName, description }

**Services package**

* **ExternalProductService** interface gets External product list
* **ExternalProductServiceImpl** class which extends **ExternalProductService** , is responsible fo parsing data using Regular Expressions from MediaExpert. In order to get a description of the product.
* **StorageService** interface for storing images.
* **StorageServiceImpl** defines methods for storing images.

**Views package**

-**BookingView** model which mainly stores fields, in which contains information about the booking process.

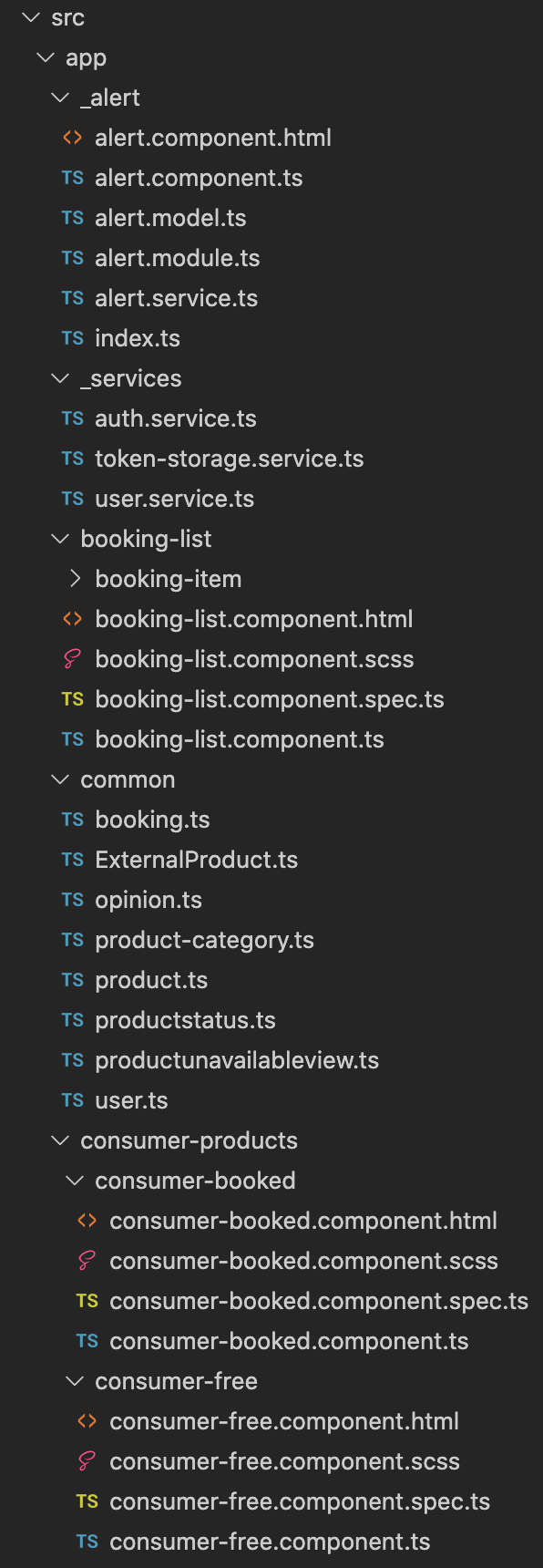
-**ExternalProductView** model which mainly stores fields, in which contains description of the external product. We are assigning to this fields data parsed from mediaExpert.

-**OpinionView** a model with fields { authorId, content, rating }

-**ProductStatusView** contains only one field about the status of the product(If the product if Reserved, Free,Booked).

-**ProductUnavailableView** a model which stores data about from what period the product is booked for the purposes that 2 users cannot order the same product for the same time.

-**UserView** a model which stores information about users personal information such as name, username, opinions about that user and it’s rating.

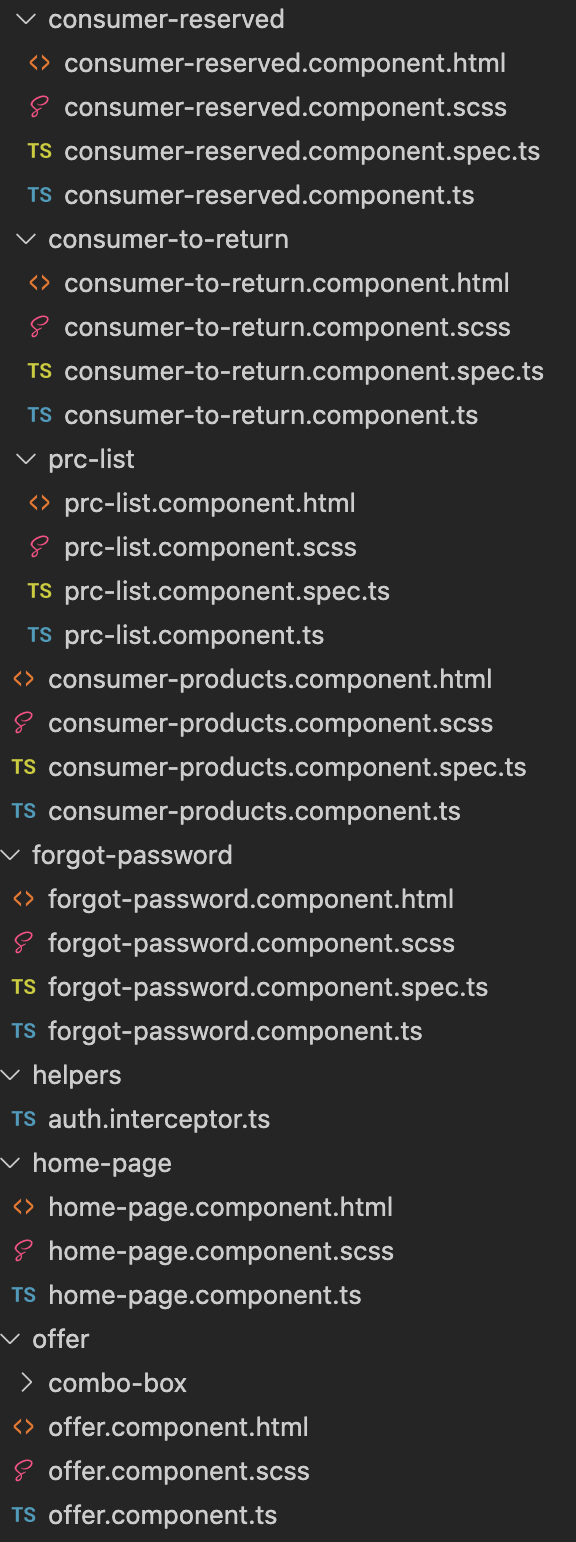
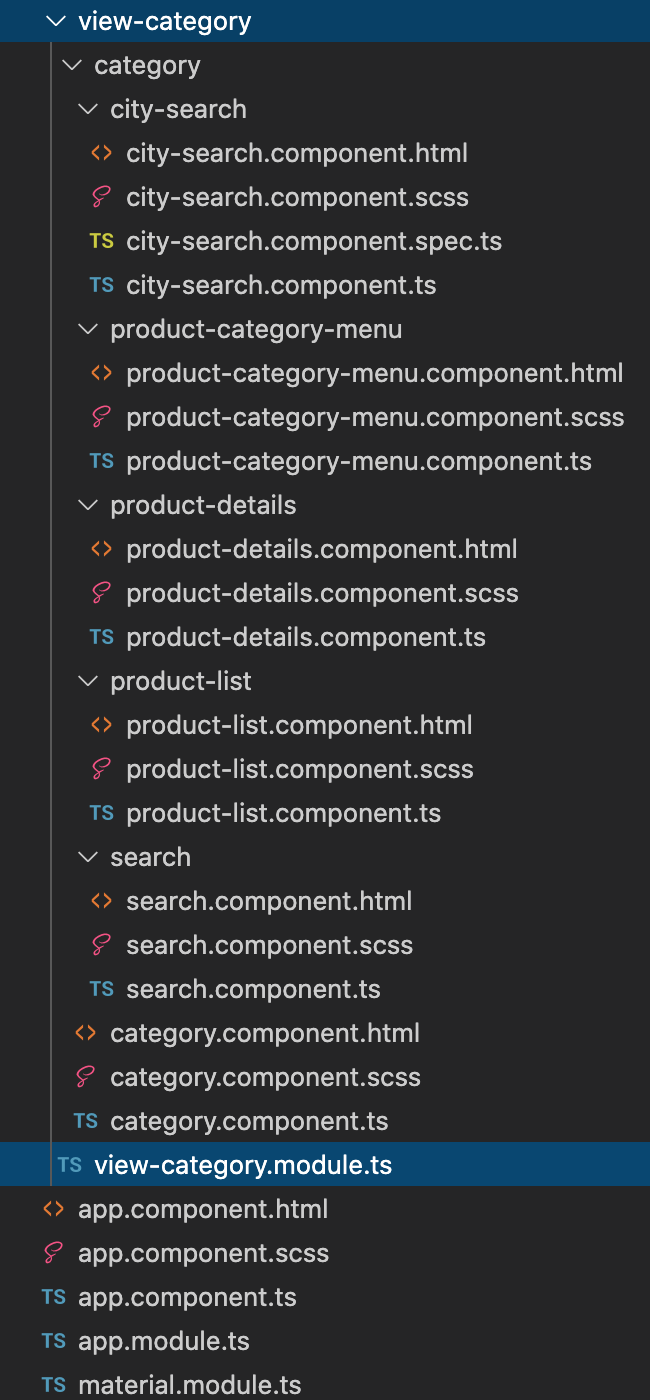
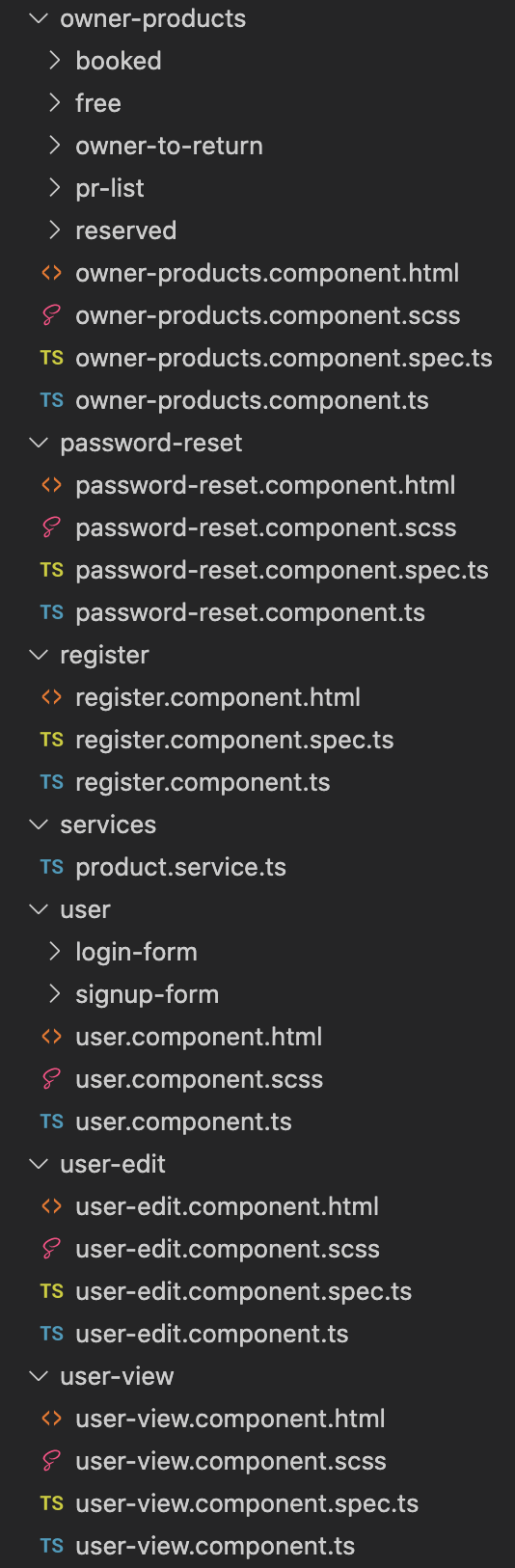
**Angular Front End**

**Technology**

-Angular 9

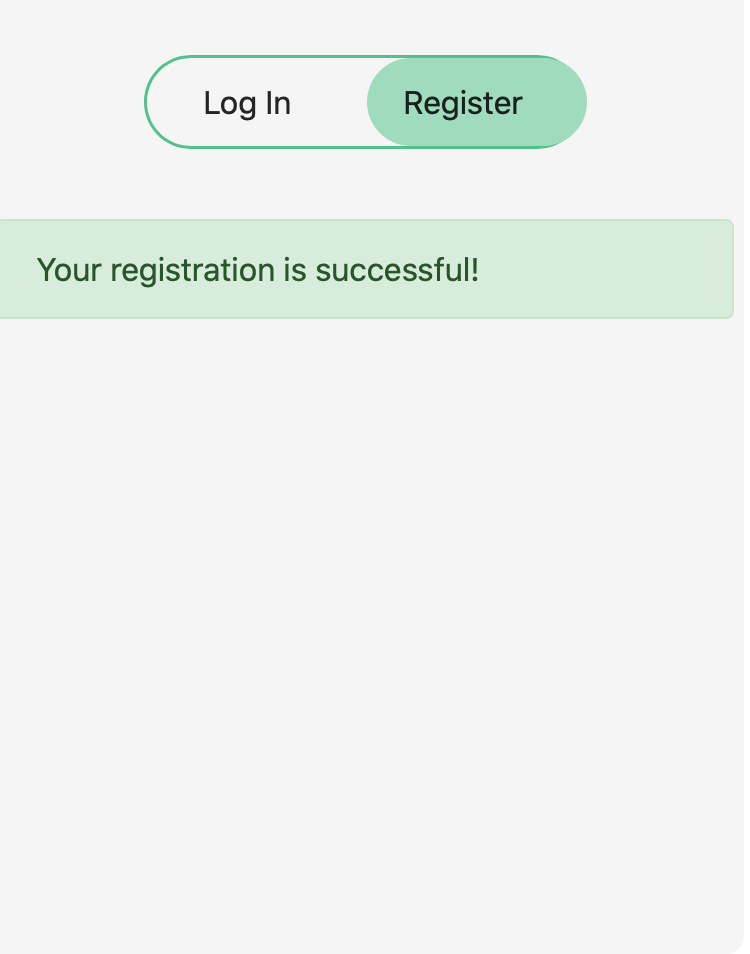
-RxJS 6

Angular is a framework for building modern single-page application.

**Project Structure**

In this section Front End will be described by components.

**\_Alert Package**

Component which is responsible for alerts. Such in order that user gets notification about the booking process, successful or not login attempt and if the offer was successfully posted.

**\_Services Package**

* **auth.service** uses Angular HttpClient to make authentication requests.

- The App component is a container using Router. It gets user token & user information from Browser Session Storage via **token-storage.service**.

- we use **user.service** to get protected resources from API.

**Booking-list Package**

This component is responsible for getting and displaying the information about the booked products from the user.

**Common Package**

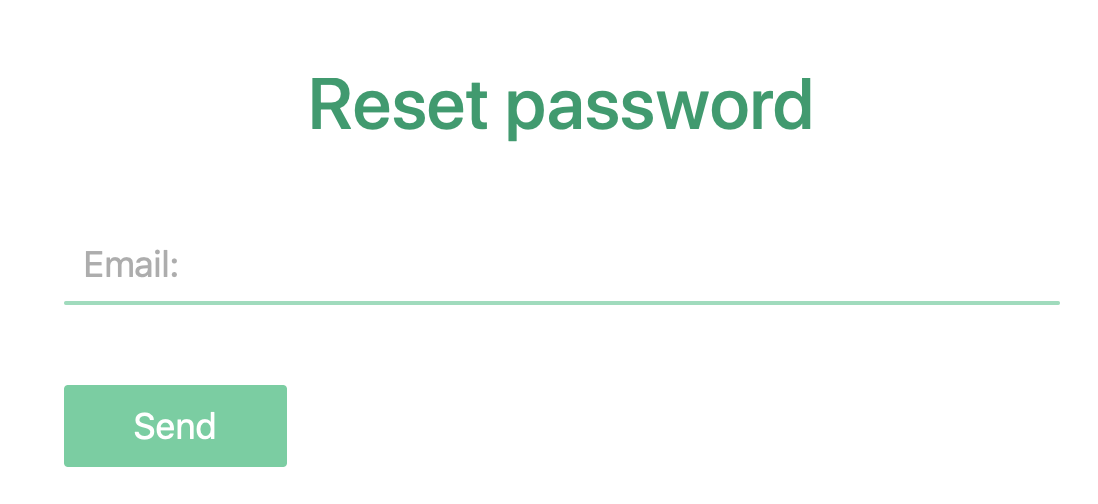
Contains models with fields such as booking.ts, ExternalProduct.ts, opinion.ts, product-category.ts, product.ts, productstatus.ts, productunavailableview.ts, user.ts.

**Consumer-Products Package**

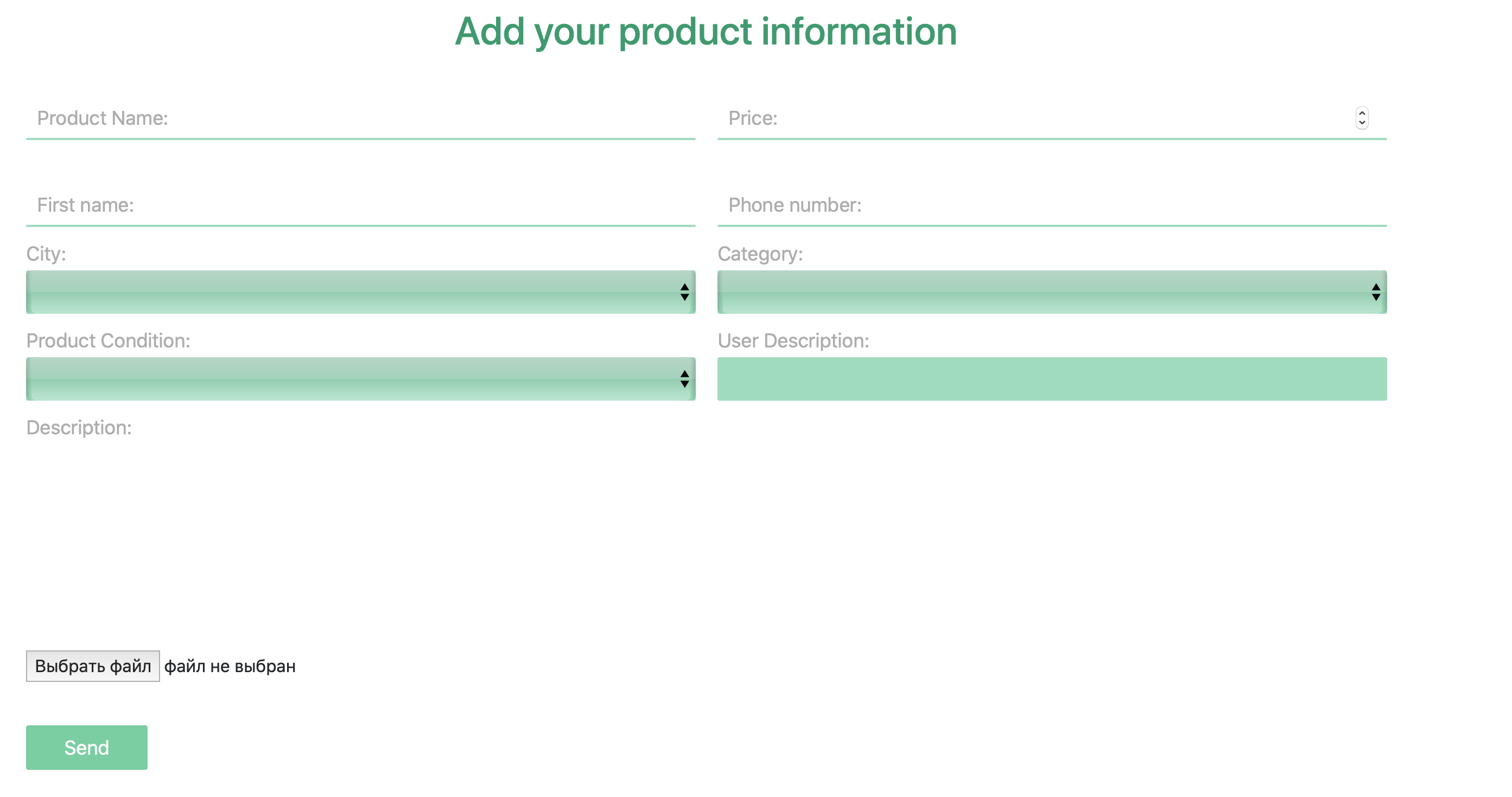
Contains such components as consumer-booked, consumer-free, consumer-reserved, consumer-to-return. For displaying in which status of booking the product is for the user who is renting a product from another user.

**Forgot-password Component**

Mainly component for reseting the password.



**Offer Component**

Component for Creating new offers, where user can fill in the information about the product, and optionally adding one or more images of the product. With the ability of getting description about the product from MediaExpert integration.

**Offer-list Component**

Shows all of the products added by the user.

**Owner-Products Package**

Contains such components as booked, free, owner-to-return, reserved. For displaying in which status of booking the product is for the user who is offering a product from another user.

**Register Component**

The component binds data (username, email, password) from template to AuthService.register() method that returns an Observable object.

**Services Component**

Contains a single service: product.service.ts. Which contains all methods regarding getting and working on REST API data from Spring Boot Server. Such as displaying products by category, by keyword, getting current user information, search external products(Integration with MediaExpert), deleting product, adding offers, reserve product, cancel reservation, book product, confirm return product, create opinion.

**User Package**

**-login-form component** uses **AuthService** to work with **Observable** object. Besides that, it calls **TokenStorageService** methods to check loggedIn status and save Token, User info to Session Storage

**User-edit Component**

Component which enables for user to change personal data such as first name, last name, email, description.

**User-view Component**

Component which shows personal information of the user and also opinions about him from the other users.

**View-category Package**

* **city-search** component for searching and displaying products by it’s city.
* **product-category-menu** component for searching and displaying products by it’s category.
* **Product-details** component, which gives a bigger description of product with the ability to rent it.
* **Product-list component**, which shows list of products with the entered criterias.
* **search component** responsible for searching products by the given keyword.