# Documentation on the project

## Technologies and basic structure

The project consists of backend and frontend apps.

Backend is built with Spring Boot + Maven and uses Junit for running tests. I used good old Java 8, but it also runs fine on Java 11 JVM.

Frontend makes use of latest ReactJS with some additional 3rd party libraries. Mocha is used for tests, together with BabelJS.

## How to run it

I invested quite a lot of time to make sure it works not depending on the IDE you use, so each part can be activated by a console command:

1. Backend:

- for simple run: mvn spring-boot:run

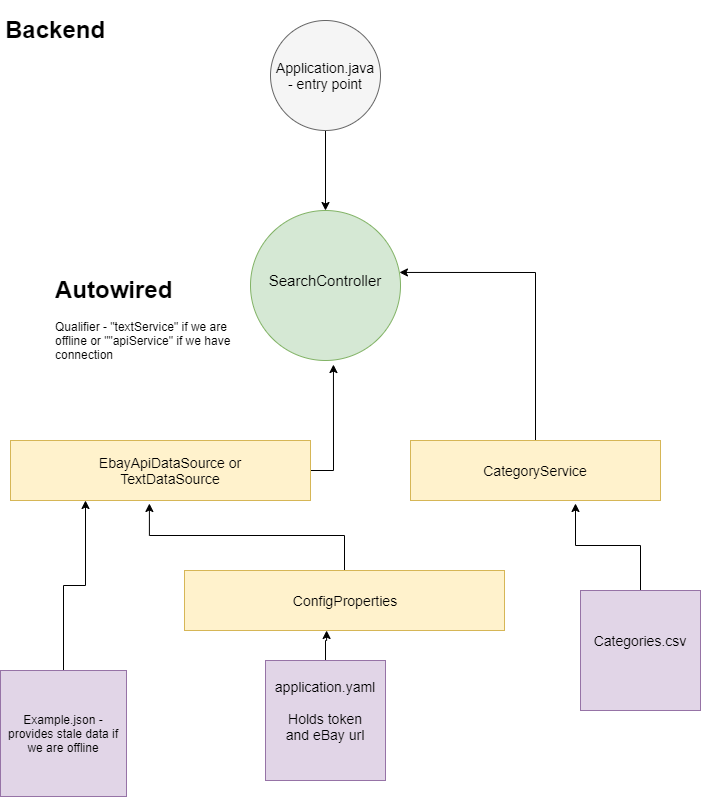
- with debug: mvn spring-boot:run -Dspring-boot.run.jvmArguments="-Xdebug -Xrunjdwp:transport=dt\_socket,server=y,suspend=y,address=5005" and then connect with debugger.

- In the second case after debugger is connected, the context will be initialized

1. Tests run with “mvn test”, and locally I ran the from Intellij as well
2. Frontend runs with “npm start” or “yarn start”, whichever you prefer. NB: make sure you run “npm install” command before you run the app
3. For tests I used Mocha in this case, though usually I use it mostly for backend. Works with “npm test” command.

NB: make sure you have a valid token in application.yaml

## Architecture



The figure above shows basic backend architecture. For simplicity I did not include data structures – I used an online tool to map JSON fields into Java classes, though later in work didn’t really use the mapped objects. Later on, I used them for testing to check if eBay API didn’t change, because the frontend would expect a certain set of object fields.

@EbayApiDataSource and @TextDataSource

Backend has a set of services autowired into one controller. Since most of the time I had to use stale data, I made a workaround: if I can connect to API, then EbayApiDataSource is used in SearchController which is the default way to work, but if we are offline and can’t use the token, then TextDataSource is used. I made them extend one abstract class and an interface, then changed the @Qualifier before launch.

Resources folder contains application.yml, which holds relevant token and API URL, Categories.csv file which I found among the API docs and example.json with stale data.

If we are offline, TextDataSource in init() method takes the json and mocks API call. In offline mode it mimics everything very well but can’t load more data to mimic pagination.

@CategoryService

To get some categories for advanced search on frontend, I needed to provide user with categories for filtering. I didn’t want to hardcode them completely, so I decided to parse a csv file, which can be replaces if category IDs change (this is basically the doc with changes).

@ConfigProperties

Provides access to props, a layer of abstraction to keep access to props in one place.

@SearchController

Has @Autowired services and 4 endpoints. In real project I would probably use Swagger with a microservice, but I thought in this case Swagger is a bit too much.

The endpoints are all for GET methods:

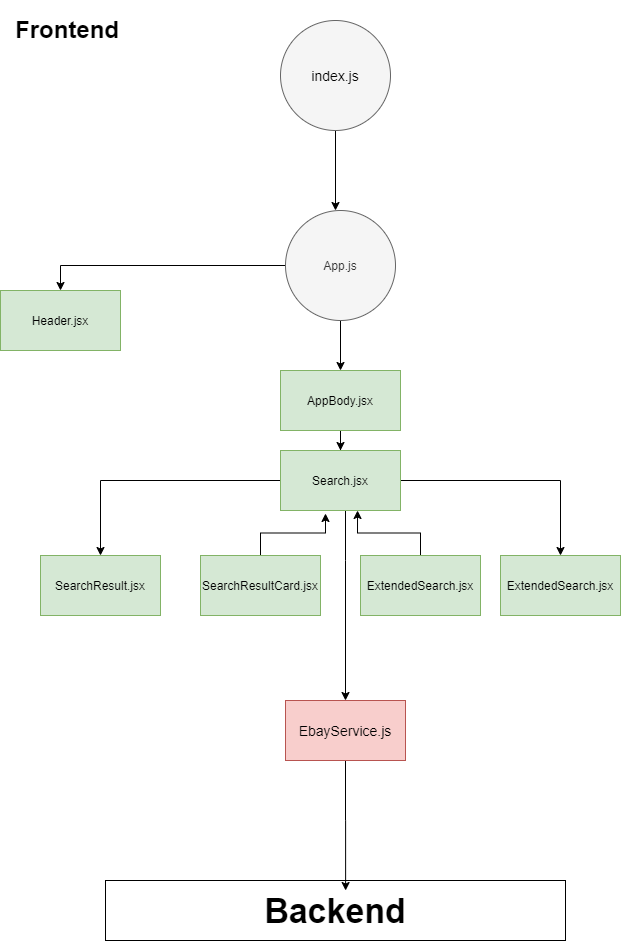
/get\_examples – returns a set of image URLs, just for UI landing page to look less empty. Wanted to make them clickable, but I think I don’t have time for it

/search?query=shoes&categoryIds=111,222 – search. User can choose several categories in advanced search on frontend, the array of categories in formed on client side.

/categories -provides categories for advanced search

/load\_more – accepts URL as param, taken by frontend from “next” field API result. Brings additional results

NB: Controller endpoints are available by CORS from origin localhost:3000.



Frontend is a bit more primitive, because I lost a lot of time trying to show how smart I am and adding Protractor testing framework, which I have in my current project. It’s a mighty testing tool which includes Selenium and can run automated tests, but I got into a pure dependency hell and finally gave up and installed Mocha + Chai + Babel using versions that I know work fine together. Mostly I use Mocha for Node.js backend testing, and heavier tools for automated frontend tests.

The UI has: search field, which remembers user’s last searches, advanced search with checkboxes for adding categories and search results. On click a popup is open and user can go to Ebay to buy the item.

Below there is a button to load additional results.

REST calls are made via simple Axios requests.