**GIT:**

[**https://github.com/ilholbea/assignmentService**](https://github.com/ilholbea/assignmentService)

**How to run:**

In order to run the application you need to install the following tools:

* JAVA JDK - <http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>
  + Add JAVA to Environment Variables -> [[1]](https://warwick.ac.uk/fac/sci/dcs/people/research/csrcbc/teaching/howto/javapath/)
* NODEJS - <https://nodejs.org/en/>
  + Go to AssignmentService folder
  + Open a cmd and run: npm install -g @angular/cli
  + Go to frontend folder
  + Run: npm update

Go into AssignmentService folder. Here you will find 2 .bat files for Windows and 2 .sh files for Linux.

Assuming you are using Windows, the steps to run the application are:

1. Run startBackEnd.bat
2. Run startFrontEnd.bat
3. Open a browser and access: localhost:4200
4. You will be redirected to a login page. Insert some inputs for username and password
5. You are now on the application

For Linux you need to run the same steps, but instead of calling .bat files, you run the .sh files. Also, for Linux, you don’t need environment variables.

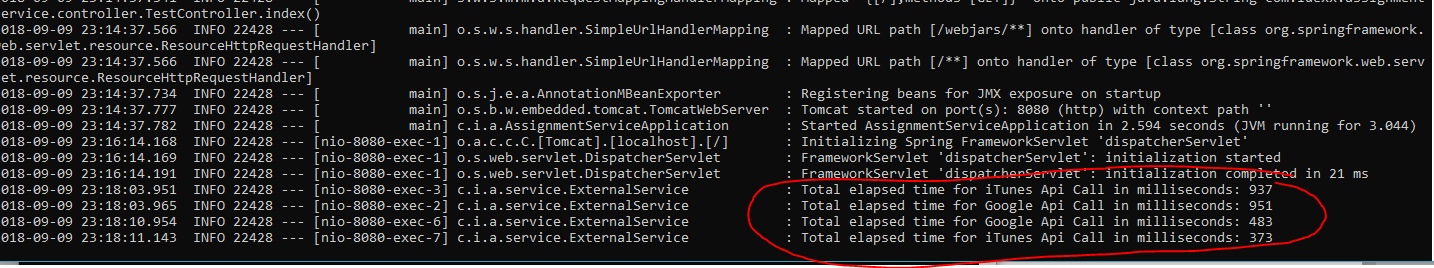
You can logout using the Logout button in the top right corner of the page.

You can select whether you want to search using GoogleApi/iTunesApi/Both.

Insert an input text in the search field and click on the Search button

The results will be returned under the search bar. They will be tabbed, for each service one tab. If you disable the service unchecking the checkbox, the call to that API won’t be made, no result will be shown and the tab will be hidden.

The metrics on response times for the upstream services will be shown in the cmd window that started the backend(startBackEnd.bat).



The results are sorted alphabetically in the backend using streams (Java 8 functionality) and then sent to frontend.

The limit of results on the upstream services is configured per environment. I’ve provided two configuration files with different number of results for each upstream service. The files are application.properties and application-30results.properties. In order to use the second configuration file you need to change the startBackEnd.bat to read from the file:

java -jar jar/assignment.jar --spring.config.location=jar/application-30results.properties

Also, as a scalability option, for iTunes you can selected whether the search should be performed on albums, artists etc. This option can also be changed in the configuration file.

Google API can return a maximum of 40 results.

**Justification of architecture/technologies:**

I’ve selected these technologies, mainly because, I believe they are really good for this type of application (webapp) and because I feel comfortable and I’ve got experience with them.

Even though it wasn’t specified I’ve added users to the application, because, later on, we might add new user functionality like recent searches, top 10 searches, etc.  
Given the fact that it would’ve been more complicated to provide the application if I would’ve added persistence to it (a database), I chose to simulate the behavior of a user based application. The user sends a username and password to the backend and from there it gets information like last name, first name, email (this info would be stored in a databse) and a token. Based on this token, the authentication is realized and the user is stored in the localStorage. Having users offers a bit more scalability options to the application and I believe this is pretty important.

I’ve added the option for the user to select on which upstream services to search. More upstream services could be added. Each upstream call is independent and if one fails the other still returns the results.

I’ve done that by having separate GET methods in the ApiController.

In the ApiController I then call the ExternalService in which there are two methods for each upstream service. Each method is different and it parses the JSON resulted from calling the API.

The GoogleAPI returns inconsistent JSON structure, that’s why I’ve had to check whether the attributes were there before taking the info from them.

The iTunesAPI returned a JSON with constant structure and it was easier to use.

Given the fact that the information that we needed was pretty similar, I’ve decided to create a model ExternalResponse in which I would put data parsed from the JSON. I then sent this list of ExternalResponse objects to the interface and displayed them.

For the UI framework, I chose to work with Angular Material, because it offers a minimalist look with subtle animations. It also integrates pretty well with Angular overall.