Readme

Table of Content

About the technologies used in this task

There are many technologies, tools, frameworks were used during the development. Some are given while others were chosen freely. I tried to minimize using 3rd party tools, but this section will give a good view about it.

Environment:

- JDK: OpenJDK version 11. (release date: 2018-09-25)
- OS: Windows 10 Pro
- IDE: IntelliJ IDEA 2018.3.3 x64
- Maven: integrated with IDEA (version 3.3.9)
- git: 2.20.1.windows.1
- sourcetree: 3.0.15
- docker: docker desktop (2.2.0.5)
- npm: 6.4.1

Technologies:

- JAVA: BE programming language
- Angular: FE programming language (with version 9)
- Spring-boot: Spring platform (2.2.6)

- SpringFox: java library for API writing (2.9.2)
- Junit5 and Mockito: for junits and integration tests

Installation and test

To make it easier, there are several possibilities to run the application.

Compile and run

Download the sources from github.

Frontend client can be found here: https://github.com/nemesys119/prime-client

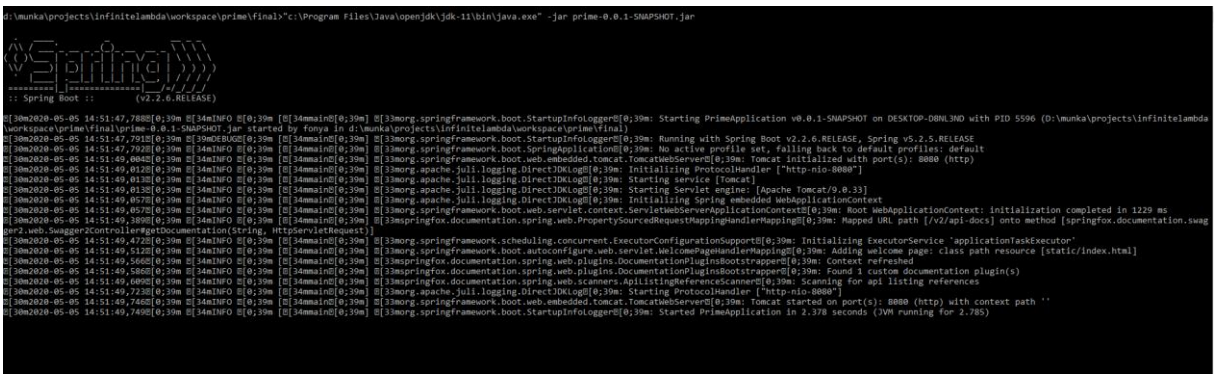
Backend application here: https://github.com/nemesys119/prime

Currently there is no coded any automatic tools or scripts which copy client's code from dist to backend's src/main/resources/static folder, so it should be done manually (There are several ways to do it, for example make a multi module maven project with a parent). Frontend application shouldn't be built, because the latest code has already copied into the destination folder in the backend project. Backend application only needs a JDK 11 and maven to build.

If you don't want to download JDK or maven, you try it as the latest compiled version was also placed on github (https://github.com/nemesys119/prime/blob/master/final/prime-0.0.1-SNAPSHOT.jar). After download it, the following command will start it:

java -jar prime-0.0.1-SNAPSHOT.jar

The following screen will be appeared, showing the success start of the app:



Using Docker

It was written above how you can download the application and where you can find the jar file if you don't want to compile it. as the project contains a dockerfile as well, you can make an image and start

it using docker. First of all you have to open a command line in the project's root folder (prime). Using the following command the docker image will be created:

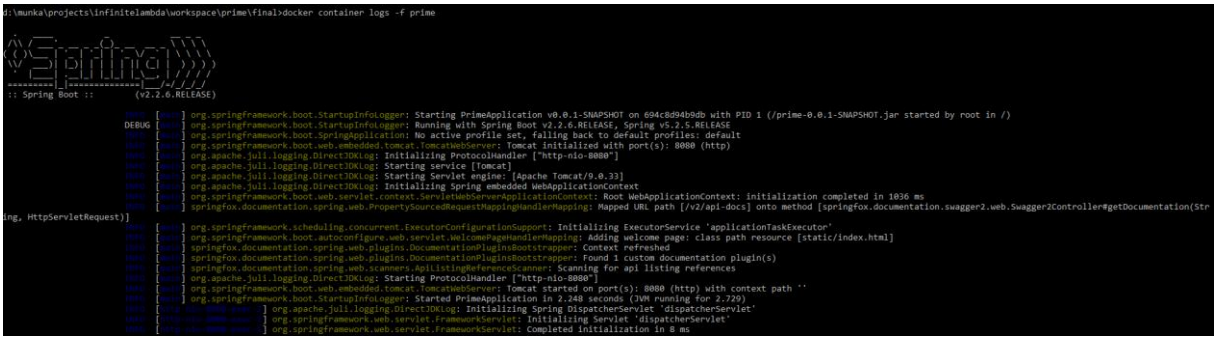
docker image build -t prime-example .

After that it should be started using a container:

docker container run --name prime -p 8080:8080 -d prime-example

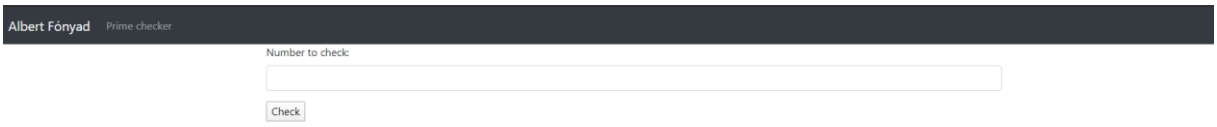
If everything fine, the app started and the log can be seen using the following command:

docker container logs -f prime

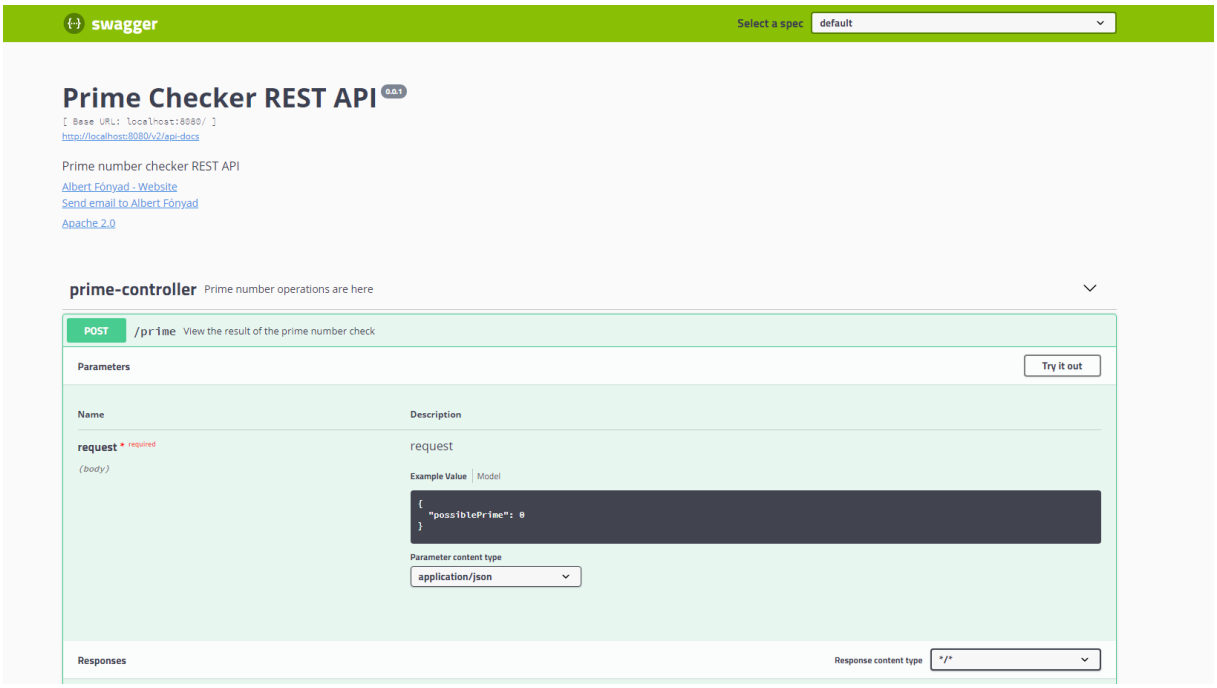


User interface

The application is available using the following URL: http://localhost:8080/



REST API here: http://localhost:8080/swagger-ui.html



Code structure

Prime-client

This is an auto-generated project using angular-cli. There are 2 dependencies which have to be installed locally, bootstrap and jquery. Versions can be found in the package.json. The real logic is implemented in the primechecker component and in the service using the same name. The primechecker service is responsible for the http communication and error handling. There is a request/response model which are used in the http POST method. As I left the target on ES 2015, there is no Bigint support in the client, so I use a little trick to make it possible to use really big number (for example M_{607} or M_{512} from here: https://en.wikipedia.org/wiki/Largest_known_prime_number). There is a conversion between string and number. The primechecker component is responsible for the validation and the html template.

Prime

It is a spring-boot based application, so the entry point is PrimeApplication. There isn't any magic written there, the REST entry point is the PrimeController. It uses the same model object as the frontend client and also make validation (using Hibernate validator as it is the default JSR 380 implementation). If Something went wrong HTTP-400 code will sent back with error message. In other cases the injected PrimeService will be called for the primality test. It alse gives back the next, bigger prime number. There is a reason to set java version to 11 as java 8 haven't got primality test yet. Using version 9 or higher has a big advantage… The Oracle has already implement a Miller-Rabin (https://en.wikipedia.org/wiki/Miller%E2%80%93Rabin_primality_test) and a Lucas-Lehmer (https://en.wikipedia.org/wiki/Lucas%E2%80%93Lehmer_primality_test) algorithm. The certainty parameter of the algorithm is written as a config parameter with default value (50). There are junit

and integration tests as well, can be found in src/test/java directory. Sl4j is used as a logging framework with xml based config.

Assumptions

The biggest question for me was the best algorithm to use. There were 2 solution, first was the real precise way, where the logic started to iterate and divide the number and check the remainder. It only need to go the number's square root and there are som tricks to make it faster, but as soon as a really big number is given, the waiting time starts to be horrible. That's why there are lot of algorithms created by mathematicians and that's why I trusted in the solution of the Oracle. The application accepts really big numbers as well.

for example:

531137992816767098689588206552468627329593117727031923199444138200403559860852242739162502265229285668889329486246501015346579337652707239409519978766587351943831270835393219031728127

In a real, big project the error handling should be centralized, but in this case handlers are written separately. The reason is the deadline (which I don't know) and in the most cases solutions have to follow the size of the project and the time available.