**SIMPLE CRUD**

This is an exercise that has for goal to do a mini project that does all the CRUD operations (Create, Read, Update, Delete).

In this exercise, you will using languages we are using for the new projects to do a **REST** Application that shows a number of Employees and their main skills. We need to be able to see details about them, Modify their data, add a new employee and delete an existing employee. All the data will be stored in a CSV file.

Languages :

* Backend **: Java Spring Boot** (Latest version)
* Frontend : **Angular TS 14**

The mini project will be divided in multiple tasks that will allow you to focus on your code and gain time. All the analysis is done for you.

You can (if your Scrum Master agrees) create task tickets in your team Backlog for each corresponding tasks and put them into your Scrum team Active sprint in the correct categories (TODO, IN PROGRESS, ON HOLD, DONE).

**Reminder** : only one ticket at a time can be in the IN PROGRESS column (its logic to do one task, and then another, doing multiple tasks at the same time is, generally, a bad idea).  
  
Your main focus will be to write **CLEAN** and **MAINTAINABLE** code. A code that another developer can understand and use to complete/fix a bug.

Your **code will be reviewed** by one of your Scrum team member. Documentation is not really necessary, unless there is a really complex part of code you want to explain explicitly.

**Time : 2 weeks** is a good range of time to complete the mini project. If you didn’t finish it in time, no worries, the quality of your code is the most important.

**BIGGEST TIP OF ALL TIME: DO NOT HESITATE TO ASK QUESTIONS TO THE OTHER DEVS**

Lets Get Started !

1. Create the backend

Before anything, we need our Backend. The Backend will be a Spring boot Application that you will run separately from your Frontend that will be in AngularTS 14. Your Backend will be your Server, the app that will ONLY give to the frontend the **necessary** information to the frontend.   
  
Try installing the Spring boot app, you can use this link <https://start.spring.io/>

And add this dependency to your pom.xml file :

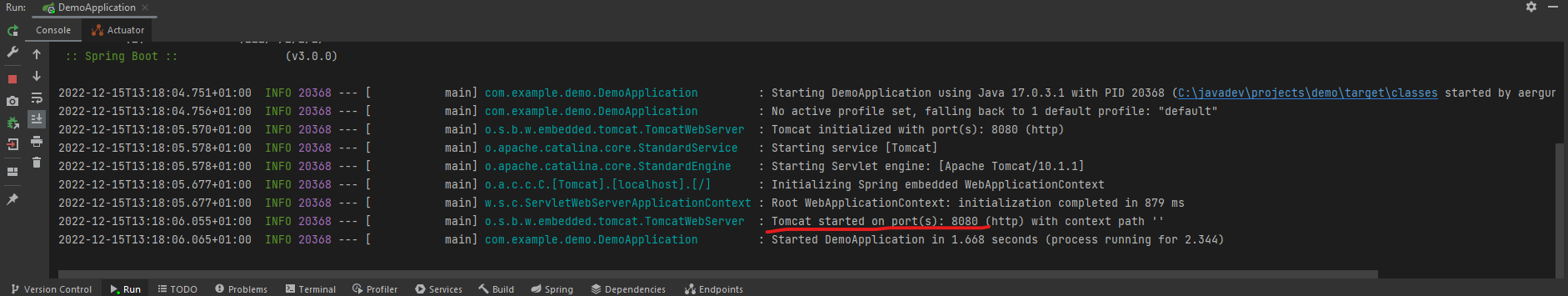
<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

Then run your project, it will stay open and you will see this in the logs :



Your server will start and use the port 8080.

1. Create Your Model

Simple, Create the model in your backend with the model needed for :

**Employee :**

* id : int NOT NULL
* name : string NOT NULL
* phoneNbr : string NOT NULL
* skills : string[]
* notes : string

Note : if you are Familiar with OpenAPI skip this part and use OpenAPI instead.

1. Create your Repository / Service / Controller class

Repository is where you will be adding your methods to get the data from your data source (for this exercise, in the CSV file).

Service is where you will be making the modifications necessary to the objects (mapping, change values, format, etc …) before sending them to the frontend.

controller is where you will be defining your REST API.

Create an interface and implementation class for your Repository and Service. Create a simple class for the Controller.

For now don’t implement anything, you will come back later.

1. Add the CSV File to your project

The CSV file is your Database for this project. It will be given to you as **DATABASE\_SIMPLE\_CRUD\_EMPLOYEES.csv**

You will read and write in this file. CSV file format is not that complex, no worries. You can put it in your resources.

You can use the library of your choice to map from CSV to java Object (f.e. : OpenCSV).

1. Read one employee from the file

First try to read the first employee from the csv file and map it to the Java object Employee.

Then print the result with all the values and check if everything corresponds.  
If it does, start creating all the implementation necessary to get the list of all the employees in JSON from your REST API.

You should be able to see the list of employees in JSON when you go to the url : localhost/8080/employees

1. Create your frontend Application

Now you will create your frontend, you can use this tutorial to create your frontend app : <https://angular.io/tutorial/tour-of-heroes/toh-pt0>

use the command **ng serve** to start your frontend app.

1. Create the frontend model and first component

Create the model in your angular app, be careful to name all the class attributes the same way you did for your backend, the mapping will be done automatically by name when you’re receiving information from your backend.

Then, create a simple component that shows all the data of the first employee (do it flat, it is just to test if you get the data correctly)

Finally, create a Service that will call your Backend Rest API and get a list of employees. And try to use them in your component and display the first employee of the list.

Tip : use Resolvers and routes for the first component, it will display all the employees.

1. Table of Employees

Now that you’ve been this far, you are ready to create a table (using classic angular materials) to display the list of all employees. Don’t do anything fancy, just center the table and display the id, name and phone number of the employees.

1. Employee details

Now, we want to click on a table row and see on another page the details of the employee showing us the skills and the notes. You can use the angular material MatCard to make it beautiful.

You can now create a get api call that return an employee by id.

Tip : use the url employee/{id} for your backend and frontend (the port will be different of course)

1. Add new Employee

Now, we want to add a new employee to the list, we want to add a button to the main screen (the screen that shows the list of employees) and when we click on it we have a form displayed (as a pop up or a new page) where we have to enter the name, phoneNbr, skills and notes. Skills and notes are not mandatory to add a new employee.

Then below the form we want a button to submit it and add it to the list of employees. You will need an post call in your api.

When the form is submitted, we come back to the main screen and see the table being updated without refresh (if possible). Don’t forget, the Database (CSV file) needs to be updated with the new employee.

1. Delete an Employee

Now, we want a new column in our table, this column will contain a delete button. When we press it, it deletes the employee displayed on the row and updates the database. You will need a delete call in your api.

BONUS: you can add a delete button on the top right corner of the card of the employee details. When we press it, it deletes the employee and we return to the main screen  
  
BONUS 2: add a confirmation pop up when we want to delete an employee

1. Update an Employee

Finally, we want to update the information about an employee. Same as the delete, create a new column, where you add that button and when we click it, we go to a new page or pop up with a form displaying us all the current data of the employee that can be updated. From there we can directly change the data that we want and resubmit them.  
  
When submitted the data will be updated in the DB and in the table.

BONUS: don’t create a new column, but instead, use the same column as delete button to place the edit button.

BONUS 2 : make the name field unmodifiable.

BONUS 3 : also put the edit button on the detail page of the employee on the top right corner of the card. Delete and Edit button will be next to each other.

1. BONUS : Add validations

Add validations for each field of the form when using the add or edit button.

* Name field cannot exceed 20 characters
* Phone number field cannot exceed 10 characters and use a mask to display in this format: +99 XXX/XXX XX
* There can be a maximum of 5 skills.

1. BONUS : add a snackbar

Make a snackbar for each CRUD action you do on your frontend to confirm that the action has been correctly done.