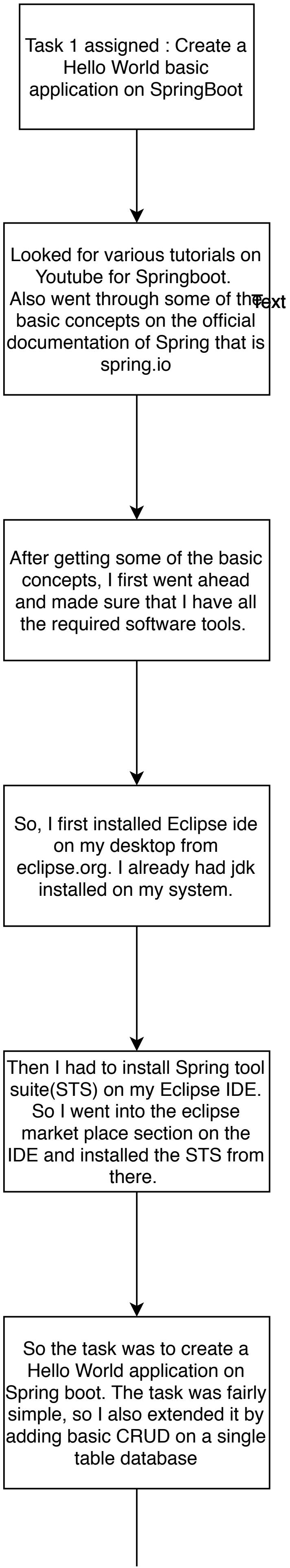
# How I created my first Spring boot Application and completed the first assignment



Task 1 assigned : Create a Hello World basic application on SpringBoot

Looked for various tutorials on Youtube for Springboot.
Also went through some of the basic concepts on the official documentation of Spring that is spring.io

Text

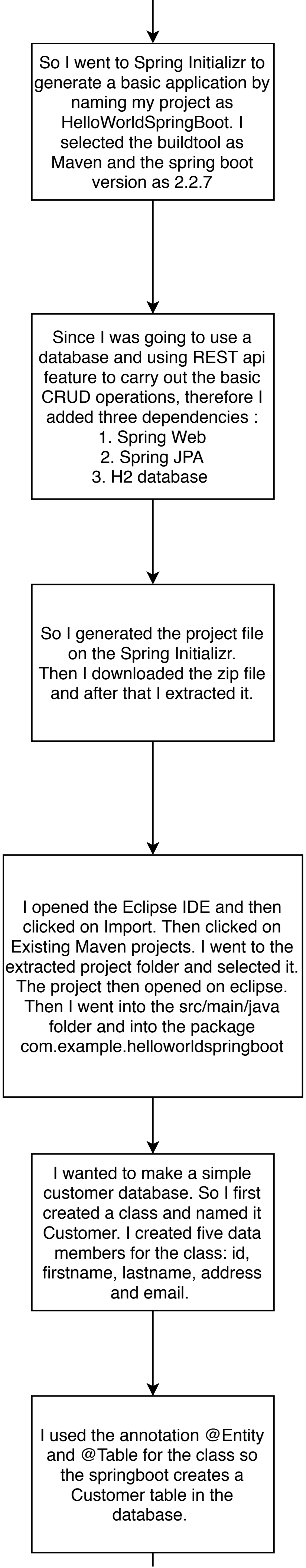
After getting some of the basic concepts, I first went ahead and made sure that I have all the required software tools.

So, I first installed Eclipse ide on my desktop from eclipse.org. I already had jdk installed on my system.

Then I had to install Spring tool suite(STS) on my Eclipse IDE. So I went into the eclipse market place section on the IDE and installed the STS from there.

So the task was to create a Hello World application on Spring boot. The task was fairly simple, so I also extended it by adding basic CRUD on a single table database

**TECH STACK**
**1. Eclipse IDE**
**2. STS 4**
**3. Spring boot 2.2.7**
**4. Build Automation system - MAVEN**
**5. Java 8**
**6. H2 database**
**7. Spring web**
**8. JPA**
**9. SQL**
**10.POSTMAN APP**



So I went to Spring Initializr to generate a basic application by naming my project as HelloWorldSpringBoot. I selected the buildtool as Maven and the spring boot version as 2.2.7

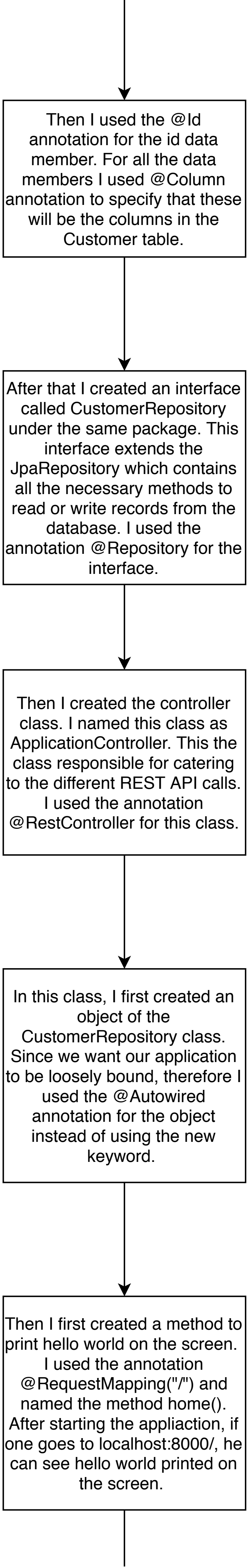
Since I was going to use a database and using REST api feature to carry out the basic CRUD operations, therefore I added three dependencies :
1. Spring Web
2. Spring JPA
3. H2 database

So I generated the project file on the Spring Initializr.
Then I downloaded the zip file and after that I extracted it.

I opened the Eclipse IDE and then clicked on Import. Then clicked on Existing Maven projects. I went to the extracted project folder and selected it. The project then opened on eclipse. Then I went into the src/main/java folder and into the package com.example.helloworldspringboot

I wanted to make a simple customer database. So I first created a class and named it Customer. I created five data members for the class: id, firstname, lastname, address and email.

I used the annotation @Entity and @Table for the class so the springboot creates a Customer table in the database.



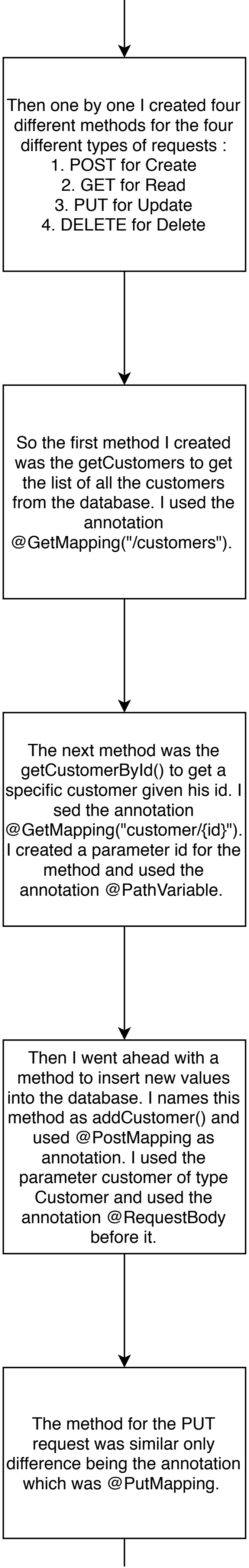
Then I used the @Id annotation for the id data member. For all the data members I used @Column annotation to specify that these will be the columns in the Customer table.

After that I created an interface called CustomerRepository under the same package. This interface extends the JpaRepository which contains all the necessary methods to read or write records from the database. I used the annotation @Repository for the interface.

Then I created the controller class. I named this class as ApplicationController. This the class responsible for catering to the different REST API calls. I used the annotation @RestController for this class.

In this class, I first created an object of the CustomerRepository class. Since we want our application to be loosely bound, therefore I used the @Autowired annotation for the object instead of using the new keyword.

Then I first created a method to print hello world on the screen. I used the annotation @RequestMapping("/") and named the method home(). After starting the appliaction, if one goes to localhost:8000/, he can see hello world printed on the screen.



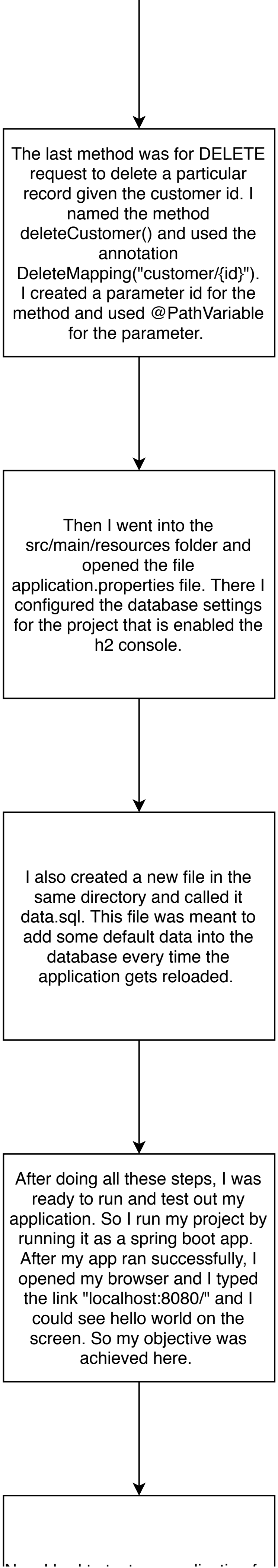
Then one by one I created four different methods for the four different types of requests :
1. POST for Create
2. GET for Read
3. PUT for Update
4. DELETE for Delete

So the first method I created was the getCustomers to get the list of all the customers from the database. I used the annotation @GetMapping("/customers").

The next method was the getCustomerById() to get a specific customer given his id. I sed the annotation @GetMapping("customer/{id}"). I created a parameter id for the method and used the annotation @PathVariable.

Then I went ahead with a method to insert new values into the database. I names this method as addCustomer() and used @PostMapping as annotation. I used the parameter customer of type Customer and used the annotation @RequestBody before it.

The method for the PUT request was similar only difference being the annotation which was @PutMapping.



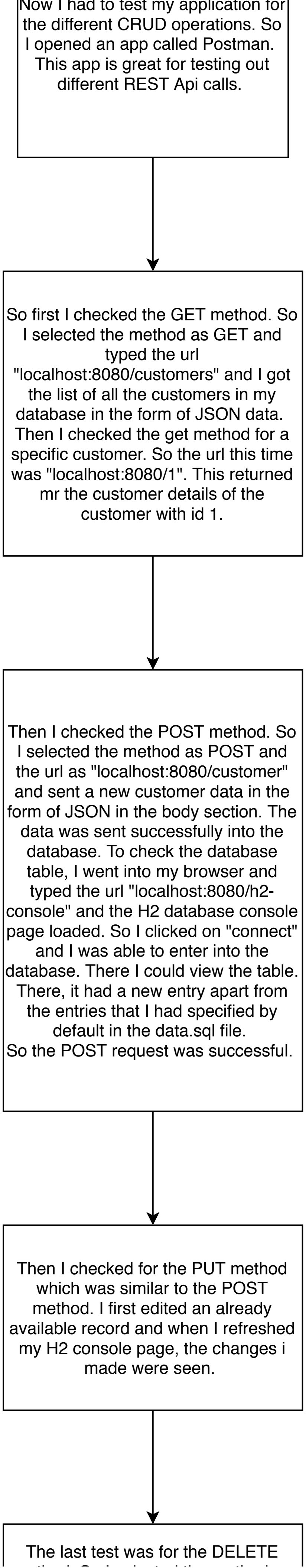
The last method was for DELETE request to delete a particular record given the customer id. I named the method deleteCustomer() and used the annotation DeleteMapping("customer/{id}"). I created a parameter id for the method and used @PathVariable for the parameter.

Then I went into the src/main/resources folder and opened the file application.properties file. There I configured the database settings for the project that is enabled the h2 console.

I also created a new file in the same directory and called it data.sql. This file was meant to add some default data into the database every time the application gets reloaded.

After doing all these steps, I was ready to run and test out my application. So I run my project by running it as a spring boot app. After my app ran successfully, I opened my browser and I typed the link "localhost:8080/" and I could see hello world on the screen. So my objective was achieved here.

Now I had to test my application f



Now I had to test my application for the different CRUD operations. So I opened an app called Postman. This app is great for testing out different REST Api calls.

So first I checked the GET method. So I selected the method as GET and typed the url "localhost:8080/customers" and I got the list of all the customers in my database in the form of JSON data. Then I checked the get method for a specific customer. So the url this time was "localhost:8080/1". This returned mr the customer details of the customer with id 1.

Then I checked the POST method. So I selected the method as POST and the url as "localhost:8080/customer" and sent a new customer data in the form of JSON in the body section. The data was sent successfully into the database. To check the database table, I went into my browser and typed the url "localhost:8080/h2-console" and the H2 database console page loaded. So I clicked on "connect" and I was able to enter into the database. There I could view the table. There, it had a new entry apart from the entries that I had specified by default in the data.sql file.
So the POST request was successful.

Then I checked for the PUT method which was similar to the POST method. I first edited an already available record and when I refreshed my H2 console page, the changes i made were seen.

The last test was for the DELETE

method. So I selected the method as DELETE and then typed the url "localhost:8080/customer/2". After the request was processed, I refreshed the h2 console page and I saw that the record with id = 2 was not there. So I was successful in implementing the CRUD operations on a single table.