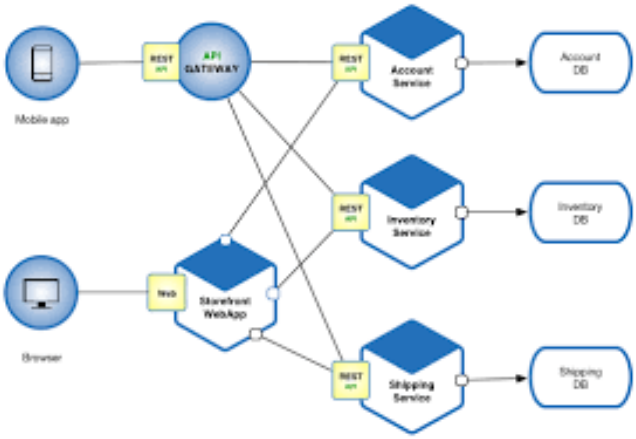
Université Chouaib Doukkali
Ecole Nationale des Sciences Appliquées d'El Jadida
Département Télécommunications, Réseaux et Informatique

Subject :

Final Exam Project- Web Development using JEE



Realized by :

HASSAR Hassane
MAZER Omar

Framed by :
Dr.Ing.F.KHARROUBI

University Year: 2020/2021

Abstract

Microservices architecture has been experiencing a meteoric rise in recent years. Giants such as **Amazon, Uber, Ebay, Groupon** and **Netflix** have redesigned their applications and information systems to be based on this architecture.

The resulting applications are of unprecedented robustness and scalability. The complexity of the application is broken down into small, easily affordable problems. This increases the resilience of the application tenfold.

Since Java EE allows us to build Java **REST APIs** quickly and easily, the main purpose of this project is developing an online **Client-Server (distributed) JEE web application** using java **REST APIs** and respecting the architectural style of micro services. In fact, via this project, we should build **CRUD REST API** using Java JEE to build 3 services:

1. **Money transfer**
2. **Water/Electricity Bills**
3. **Phone Top Up (recharge)**

Keywords:

Micro-Service, Client-Server Architecture, Web Application, JEE, Frameworks, CRUD

Acknowledgement

It is a pleasure and a very pleasant moment to pay tribute and express thanks to all those who, in one way or another, have supported and contributed to the finalisation of this work.

First of all, we would like to express our sincere gratitude to our supervisor, **Dr.Ing.Fouad KHARROUBI**, for the advice he has always given us, his understanding and the confidence he has always shown in us.

We would also like to express our deep gratitude and respect for the assistance and advice provided by all for the missions mentioned in this report. Of course, we thank our friends and families who throughout our studies have not ceased to support and accompany us in all these moments.

Thank you all!

Table of Contents

List of figures

1
Introduction

Since Java EE allows us to build Java **REST APIs** quickly and easily, the main purpose of this project is developing an online **Client-Server (distributed) JEE web application** using java **REST APIs** and respecting the architectural style of micro services. In fact, via this project, we should build **CRUD REST API** using Java JEE to build 3 services:

1. **Money transfer**
2. **Water/Electricity Bills**
3. **Phone Top Up (recharge)**

This report is organised in **four** chapters as follows:

- **Design and modelling**: in this chapter, we will detail the various diagrams produced.
- **Tools**: in this chapter, we will detail the various tools used.
- **Realisation**: in this chapter, we will detail all steps (screenshots and comments) like we did during lab's classes all step by step.
- **Conclusion and perceptions**: in this part we will summarise the study and the work carried out in this project, and draw up the perspectives and the remaining work in order to relatively perfect our project.

2
Specification requirements

Introduction

As seen in the software engineering course, the requirements specification phase is mainly aimed at:

- Understanding the customer's needs.
- Establish a clear description of what the software must do (detailed functionalities, quality requirements, interface...).

In this chapter, we will try to specify as much as possible the different expected functionalities as discussed with the project owner (MOA); knowing that: "**The requirements are never complete or definitive**". Then, we will detail the **global use case diagram**.

2.1 Specification of requirements

2.1.1 Functional needs :

These are the functionalities of the system. These are the requirements specifying an input/output behaviour of the system which must allow for:

Payment Agent:

- Authenticate
- Select services
- Create new service
- Read services
- Edit/Update existing services
- Delete services

2.1.2 Non-functional needs :

These are the needs that characterise the system. These are needs in terms of performance, type of equipment or type of design. These needs may concern implementation constraints (programming language, DBMS type, operating system, etc.).

- **Ease of use**: handling and robustness .
- **Performance**: response time, throughput, fluidity...etc.
- **Reliability**: fault tolerance .
- **Maintainability**: easy to correct and upgrade the software .
- **Portability** : adaptability to other hardware or software environments .
- **Security** : data integrity and access protection. For example, authentication is mandatory for all users of the application, using a login and password.

Global Use Case Diagram:

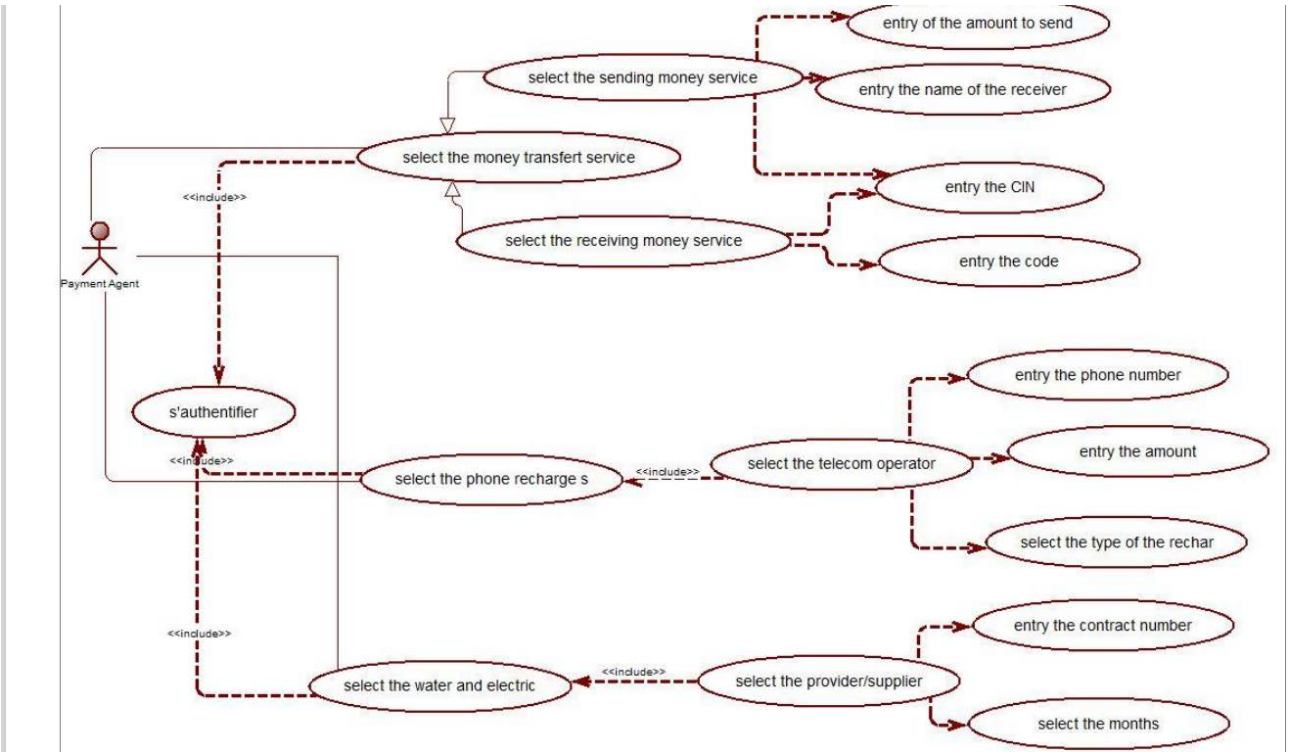


Figure 1:Use case diagram

Conclusion

After specifying and analysing the needs of our application, both functional and non-functional. In the next chapter, we will discuss the tools part.

3 Tools

Introduction

In this chapter we present the technical choice we chose for the development of our application.

Table 1:Technical choice for the development of our application

Technology	Type	Where did we use it?	How did we use it?	Why?
Spring-boot	Framework	Backend development	Backend (microservice's code)	To create microservices
Spring-cloud	Framework	Backend development	Configuration	Build some of the common patterns in distributed systems
Spring-Initializr	web application	Project creation (Backend development)	Configuration + dependencies	extract all the dependencies you need for an application and does much of the configuration for you
Spring Security	Java/Java EE framework	Backend	In Logging and sign-up features	provides authentication, authorization and other security features for enterprise applications.

Eureka	**Eureka Server** is an application that holds the information about all client-service applications.	Backend	We use it to set the registration service	Every Micro service will register into the Eureka server and Eureka server knows all the client applications running on each port and IP address.
Zuul	edge service that proxies requests to multiple backing services	Backend + proxy	We use it to set the proxy service	handles all the requests and does the dynamic routing of microservice applications.
PhpMyAdmin	Software	Backend development(Database side)	MySQL administration tools	free and open source
Hibernate	Framework	Backend development	object-relational mapping	object-relational mapping
Thymeleaf	XML/XHTML/HTML5 template engine	Frontend development	Web pages development	It provides full Spring Framework integration.
Bootstrap	Framework	Frontend development	Frontend development	For frontend
AJAX	web development techniques	Frontend development	Frontend development	For communication between API Rest and frontend
ARC	API testing tool	Test	Test	Testing microservices method's
Git	System software	Backend development	We use it in the repository cloud-config	To create local git repository

Conclusion

We are convinced of the choice of our tools. Each time a technology is used, it is associated with a small definition. In the following chapter, we tackle the part of the realization step by step.

4 Realisation

Introduction

The realisation phase is a very decisive stage in the life cycle of our application. This phase allows us to concretise our project through the development of interfaces and through concrete realisations of the system's functionalities.

4.1 Backend

4.1.1 Money transfer Microservice Implementation

1. Initial creation of the application with Spring Initializr
2. Initializr offers a quick way to extract all the dependencies you need for an application and does much of the configuration for you. This example only needs the **Web, JPA, REST Repository, Devtools and MySQL Driver** dependencies. The following image shows the Initializr configured for this example project:

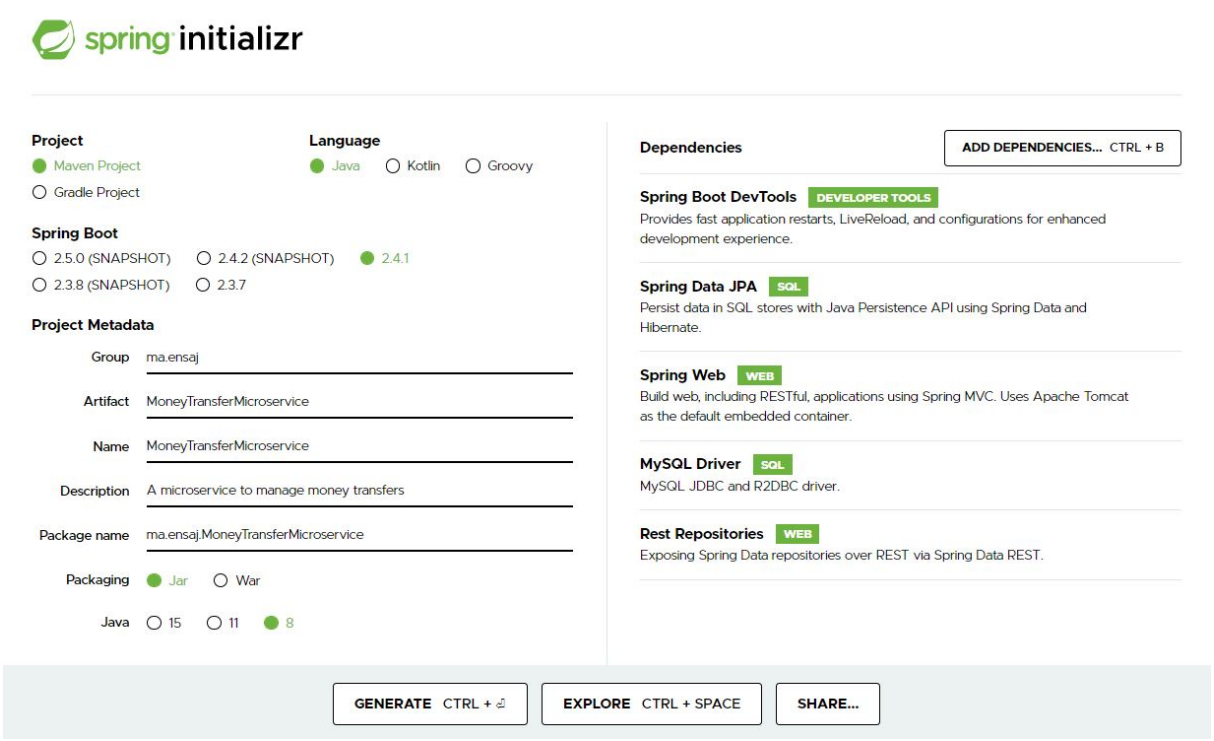


Figure 2:Initial creation of the application with Spring Initializr

3. Download the project and extract it

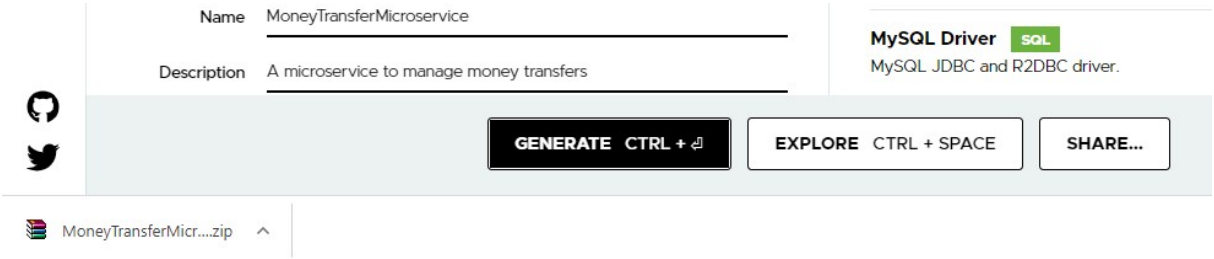
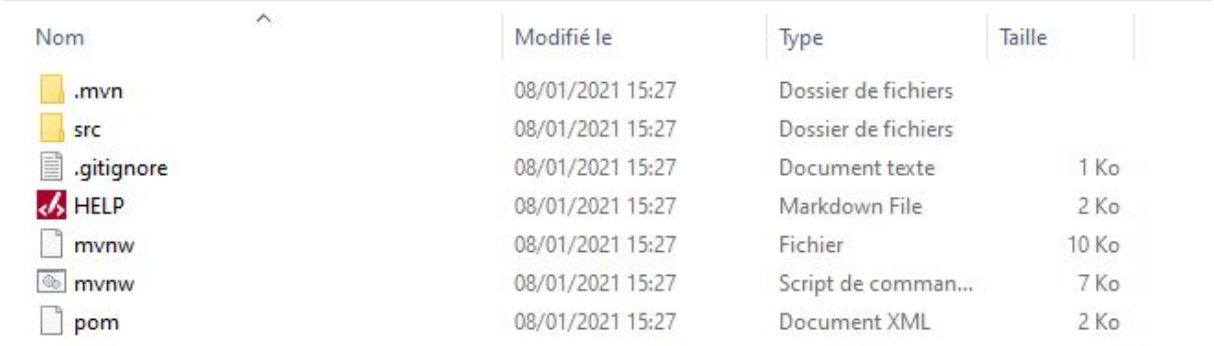


Figure 3:Download the project and extract it



Nom	Modifié le	Type	Taille
.mvn	08/01/2021 15:27	Dossier de fichiers	
src	08/01/2021 15:27	Dossier de fichiers	
.gitignore	08/01/2021 15:27	Document texte	1 Ko
HELP	08/01/2021 15:27	Markdown File	2 Ko
mvnw	08/01/2021 15:27	Fichier	10 Ko
mvnw	08/01/2021 15:27	Script de comman...	7 Ko
pom	08/01/2021 15:27	Document XML	2 Ko

Figure 4:The project after extraction

4. Run the command : **mvn eclipse :eclipse** to make the project an eclipse project

```
C:\Users\MAZER Omar\Desktop\Projets\JEE\MoneyTransferMicroservice>mvn eclipse:eclipse
[INFO] Scanning for projects...
[INFO]
[INFO] -----------------< ma.ensaj:MoneyTransferMicroservice >-----------------
[INFO] Building MoneyTransferMicroservice 0.0.1-SNAPSHOT
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] >>> maven-eclipse-plugin:2.10:eclipse (default-cli) > generate-resources @ MoneyTransferMicroservice >>>
[INFO]
[INFO] <<< maven-eclipse-plugin:2.10:eclipse (default-cli) < generate-resources @ MoneyTransferMicroservice <<<
[INFO]
[INFO]
[INFO] --- maven-eclipse-plugin:2.10:eclipse (default-cli) @ MoneyTransferMicroservice ---
[INFO] Using Eclipse Workspace: null
[INFO] Adding default classpath container: org.eclipse.jdt.launching.JRE_CONTAINER
[INFO] Resource directory's path matches an existing source directory but "test", "filtering" or "output" were different
.The resulting eclipse configuration may not accurately reflect the project configuration for src/main/resources
[INFO] Not writing settings - defaults suffice
[INFO] Wrote Eclipse project for "MoneyTransferMicroservice" to C:\Users\MAZER Omar\Desktop\Projets\JEE\MoneyTransferMic
roservice.
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  9.877 s
[INFO] Finished at: 2021-01-08T16:31:20+01:00
[INFO] ------------------------------------------------------------------------

C:\Users\MAZER Omar\Desktop\Projets\JEE\MoneyTransferMicroservice>
```

Figure 5:make the project an eclipse project

5. Import it under Eclipse.

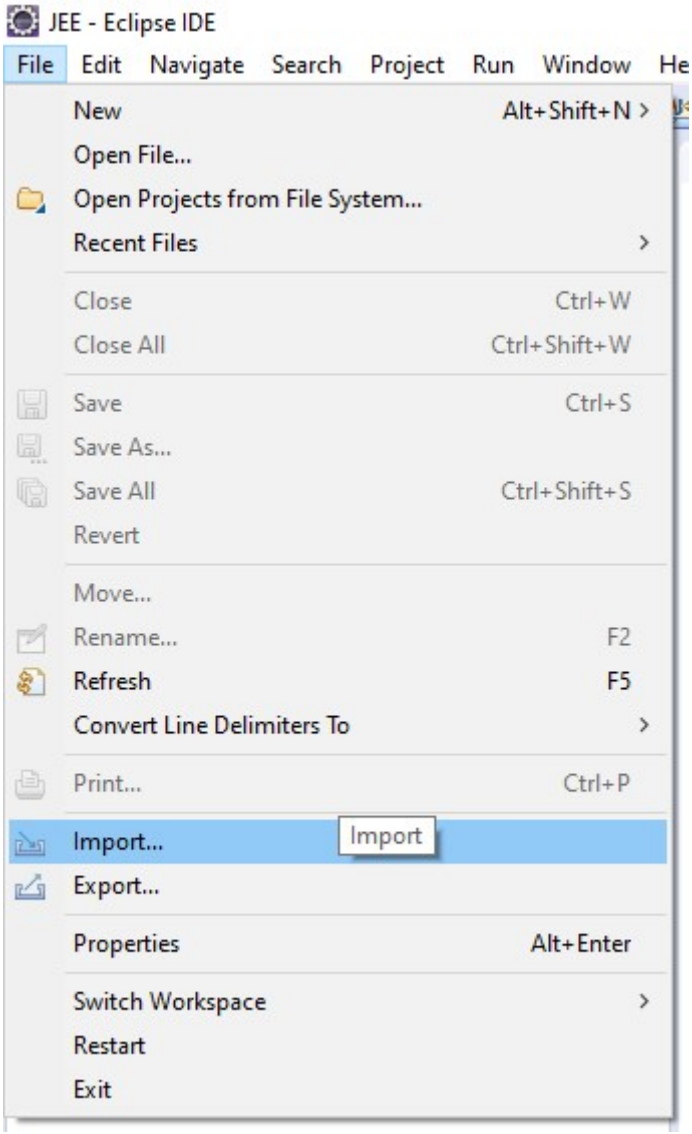


Figure 6:Import it under Eclipse.

Import it under Eclipse.

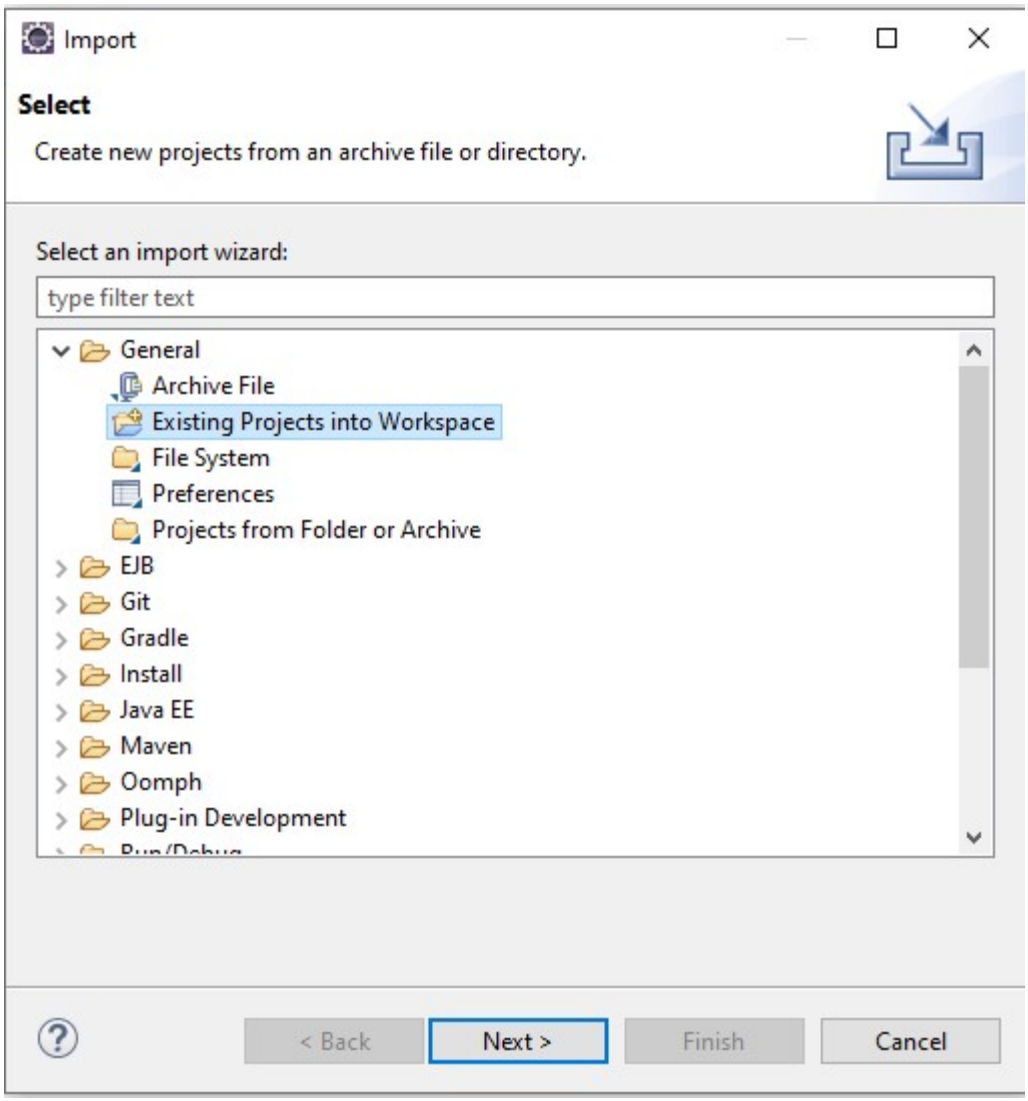


Figure 7:Import it under Eclipse.

Import it under Eclipse.

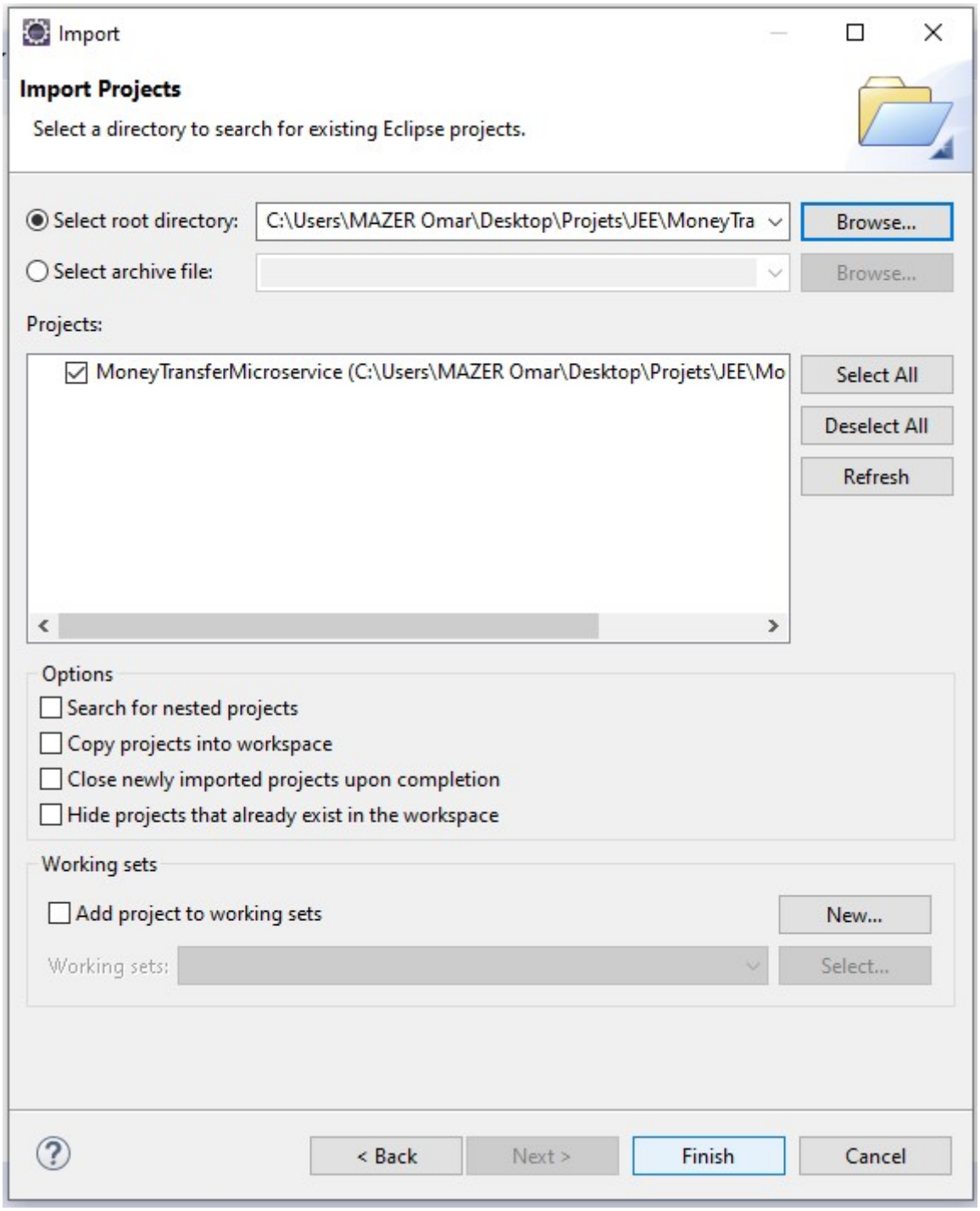


Figure 8:Import it under Eclipse.

6. Project structure

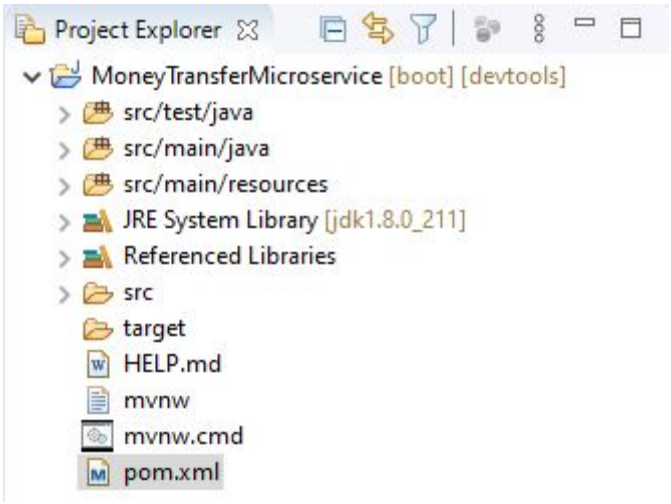


Figure 9:Project structure

7. The content of pom.xml :

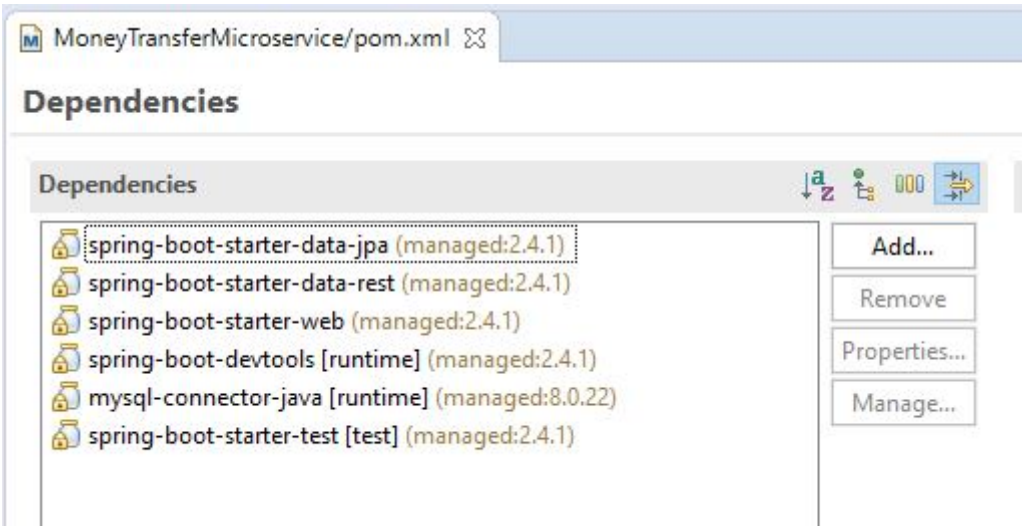


Figure 10:The content of pom.xml

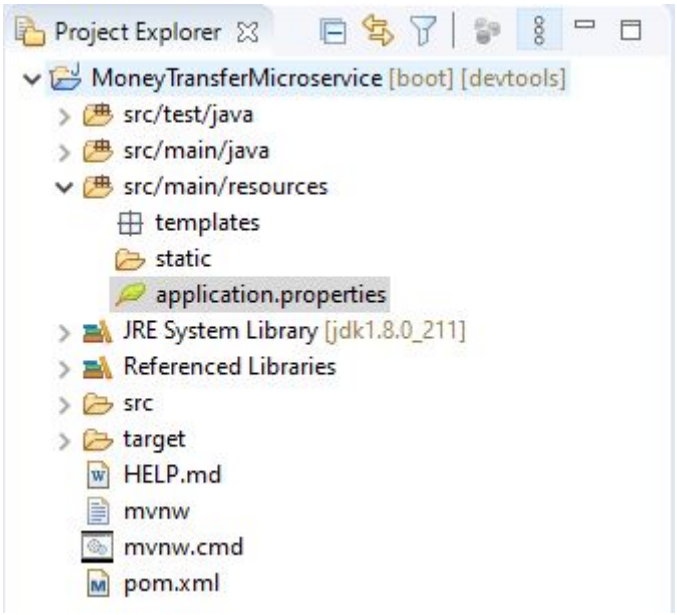


Figure 11:application.properties

Configuration of the database and the Persistence Framework

Spring Boot comes with a built-in mechanism for configuring applications using a file called **application.properties.** It is located in the **src / main / resources** folder, as shown in the following figure.

8. In the **application.properties** file, configure the database connection parameters and some parameters related to the **Persistence** Framework :

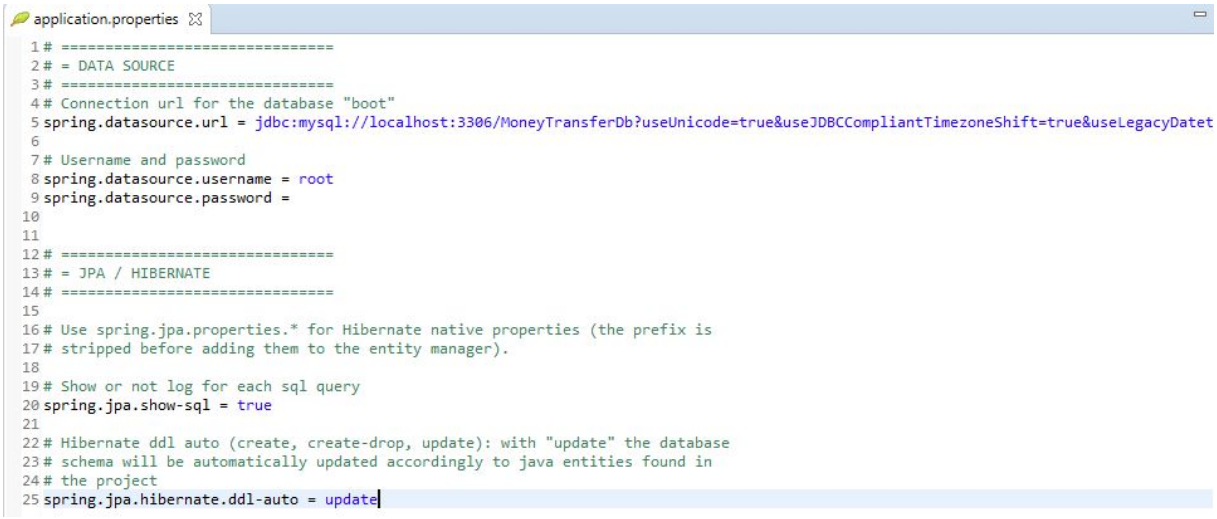


Figure 12:In the application.properties file, configure the database connection parameters

9. Create the "**MoneyTransferDb**" database under MySQL.

Figure 13:Create the "MoneyTransferDb" database under MySQL.

Creation of business classes

First create the package **entity** under the main package

10. Create the "**Transfer**" class in a sub-package **entity** of the main package

Transfer.java

```
1  package ma.ensaj.MoneyTransferMicroservice.entity;
2  import java.util.Date;
14
15 @Entity
16 public class Transfer {
17     @Id
18     @GeneratedValue(strategy=GenerationType.IDENTITY)
19     private int id;
20     private String cinSender;
21     private int amount;
22     private String nameReceiver;
23     private String cinReceiver;
24     private long code;
25
26     @JsonFormat(shape = JsonFormat.Shape.STRING,pattern = "yyyy-MM-dd HH:mm:ss")
27     @Temporal(TemporalType.TIMESTAMP)
28     private Date dateSending;
29
30     @JsonFormat(shape = JsonFormat.Shape.STRING,pattern = "yyyy-MM-dd HH:mm:ss")
31     @Temporal(TemporalType.TIMESTAMP)
32     private Date dateReceiving;
33
34     @Column(columnDefinition="BOOLEAN DEFAULT false")
35     private Boolean isReceived;
36
37     public Transfer() {
38         super();
39         this.code = new Random().nextLong() % (9999999999L - 1000000000L) + 9999999999L;
40     }
```

Figure 14:Create the "Transfer" class in a sub-package entity of the main package

A Spring Boot application is a normal application, with main and annotations :

11. The **@SpringBootApplication** annotation is a meta-annotation that triggers auto-configuration and component scanning (in the classic Spring sense).

MoneyTransferMicroserviceApplication.java

```
1  package ma.ensaj.MoneyTransferMicroservice;
2
3  import org.springframework.boot.SpringApplication;
5
6  @SpringBootApplication
7  public class MoneyTransferMicroserviceApplication {
8
9      public static void main(String[] args) {
10         SpringApplication.run(MoneyTransferMicroserviceApplication.class, args);
11     }
12
13 }
14
```

Figure 15:@SpringBootApplication

12. To start the application, use **mvn spring-boot:run** command

```
C:\Users\MAZER Omar\Desktop\Projets\JEE\MoneyTransferMicroservice>mvn spring-boot:run
```

Figure 16:mvn spring-boot:run command

13. Check if the "**transfer**" table is created.

Table	Action	Lignes	Type	Interclassement	Taille	Perte
transfer	Parcourir Structure Rechercher Insérer Vider Supprimer	0	InnoDB	utf8mb4_general_ci	16,0 kio	-
1 table	Somme	0	InnoDB	utf8mb4_general_ci	16,0 kio	0 o

Figure 17:Check if the "transfer" table is created.

14. The structure of the table "**transfer**"

	#	Nom	Type	Interclassement	Attributs	Null	Valeur par défaut	Commentaires	Extra
☐	1	id	int(11)			Non	*Aucun(e)*		AUTO_INCREMENT
☐	2	amount	int(11)			Non	*Aucun(e)*		
☐	3	cin_receiver	varchar(255)	utf8mb4_general_ci		Oui	*NULL*		
☐	4	cin_sender	varchar(255)	utf8mb4_general_ci		Oui	*NULL*		
☐	5	code	bigint(20)			Non	*Aucun(e)*		
☐	6	date_receiving	datetime			Oui	*NULL*		
☐	7	date_sending	datetime			Oui	*NULL*		
☐	8	is_received	bit(1)			Non	*Aucun(e)*		
☐	9	name_receiver	varchar(255)	utf8mb4_general_ci		Oui	*NULL*		

Figure 18:The structure of the table “transfer”

15. Creation of repositories

Spring Data JPA provides a data access layer implementation for a Spring application.

It is a very convenient brick because it allows not reinventing the data access wheel with each new application, and thus allowing you to focus on the business side.

Create the "**TransferRepository**" interface in a sub-package -called repository- of the main package :

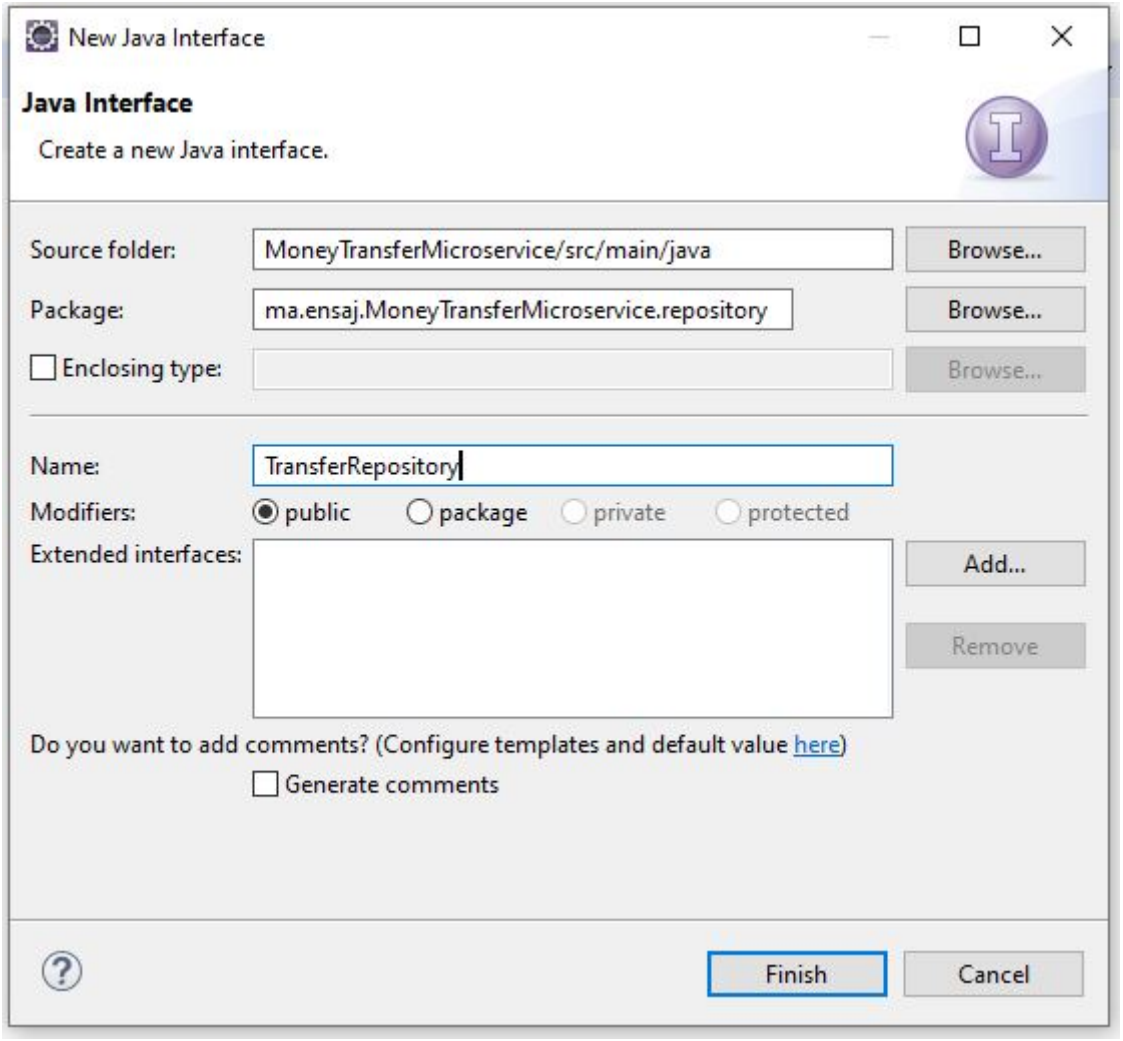


Figure 19:Create the "TransferRepository"

16. In the interface **TransferRepository** add the methods :

Transfer findById(int id);

Transfer findByCode(@Param (« «code » lopng code) ;

TransferRepository.java

```java
package ma.ensaj.MoneyTransferMicroservice.repository;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;

import ma.ensaj.MoneyTransferMicroservice.entity.Transfer;

public interface TransferRepository extends JpaRepository<Transfer, Integer> {
    Transfer findById(int id);

    @Query("SELECT t FROM Transfer t WHERE t.code= :code AND t.isReceived= false")
    Transfer findByCode(@Param("code") long code);
}
```

Figure 20:In the interface TransferRepository add the methods

17. Creation of REST controllers

REST stands for REpresentational State Transfer. It is developed by Roy Thomas Fielding, who also developed HTTP. The main goal of RESTful Web Services is to make Web services more efficient. RESTful Web Services attempt to define services using the various concepts already present in HTTP. REST is an architectural approach, not a protocol.

It does not define the standard message exchange format. We can create REST services with XML and JSON. JSON is a more popular format with REST. Key abstraction is a resource in REST. A resource can be anything. It is accessible via a Uniform Resource Identifier (URI).

18. Create the "**TransferController**" class in a sub-package-called controller- of the main package:

19. Add annotations **@RestController** and **@RequestMapping(“transfers”)**

N.B:

- The **@RestController** annotation was introduced in Spring 4.0 to simplify the creation of RESTful web services. It's a convenience annotation that combines @Controller and @ResponseBody – which eliminates the need to annotate every request handling method of the controller class with the @ResponseBody annotation.
- **@RequestMapping** is used to map web requests to Spring Controller methods.

20. Add **CRUD, receiveMoney and sendMoney** web services to the **TransferController**

TransferController.java

```java
package ma.ensaj.MoneyTransferMicroservice.controller;

import java.util.Calendar;

@RestController
@RequestMapping("transfers")
public class TransferController {

    @Autowired
    private TransferRepository transferRepository;

    @PostMapping("/sendMoney")
    public long sendMoney(@RequestBody Transfer transfer){
        transfer.setDateSending(Calendar.getInstance().getTime());
        transfer.setIsReceived(false);
        transferRepository.save(transfer);
        return transfer.getCode();
    }

    @PostMapping("/receiveMoney/{code}/{cin}")
    public Transfer receiveMoney(@PathVariable (required = true) long code,@PathVariable (required = true) String cin){
        Transfer transfer = transferRepository.findByCode(code);
        if(transfer != null) {
            transfer.setCinReceiver(cin);
            transfer.setDateReceiving(Calendar.getInstance().getTime());
            transfer.setIsReceived(true);
            transferRepository.save(transfer);
        }
        return transfer;
    }
```

Figure 21:Add CRUD, receiveMoney and sendMoney web services to the TransferController

21. Add **CRUD, receiveMoney and sendMoney** web services to the **TransferController**

```java

    @DeleteMapping("/{id}")
    public void delete(@PathVariable (required = true) String id){
        Transfer s = transferRepository.findById(Integer.parseInt(id));
        transferRepository.delete(s);
    }

    @GetMapping("/{id}")
    public Transfer findById(@PathVariable (required = true) String id){
        return transferRepository.findById(Integer.parseInt(id));
    }

    @GetMapping("/findByCode/{code}")
    public Transfer findByCode(@PathVariable (required = true) String code){
        return transferRepository.findByCode(Long.parseLong(code));
    }

    @GetMapping("/all")
    public List<Transfer>findAll(){
        return transferRepository.findAll();
    }
}
```

Figure 22:Add CRUD, receiveMoney and sendMoney web services to the TransferController

22. Testing controllers with ARC(Advanced Rest Client)

sendMoney service test

Figure 23:sendMoney service test

23. The money is sent successfully. The message code **200 OK.**

Figure 24:The money is sent successfully

24. List of transfer’s requests

Figure 25:List of transfer’s requests

25. Get list of money’s transfers

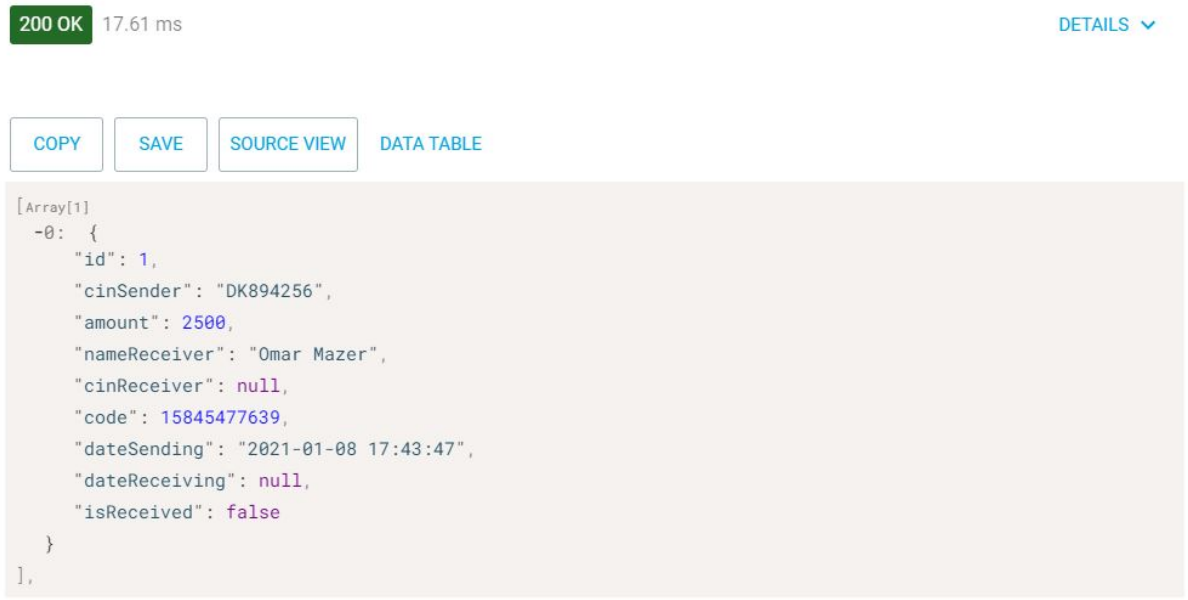


Figure 26:list of money’s transfers

26. Receive money by code request

Figure 27:Receive money by code request

27. The money is received successfully

Figure 28:The money is received successfully

4.1.2 Water/Electricity Bills Microservice Implementation

1. Initial creation of the application with Spring Initializr

Initializr offers a quick way to extract all the dependencies you need for an application and does much of the configuration for you. This example only needs the **Web, JPA, REST Repository, Devtools and MySQL** Driver dependencies. The following image shows the Initializr configured for this example project:

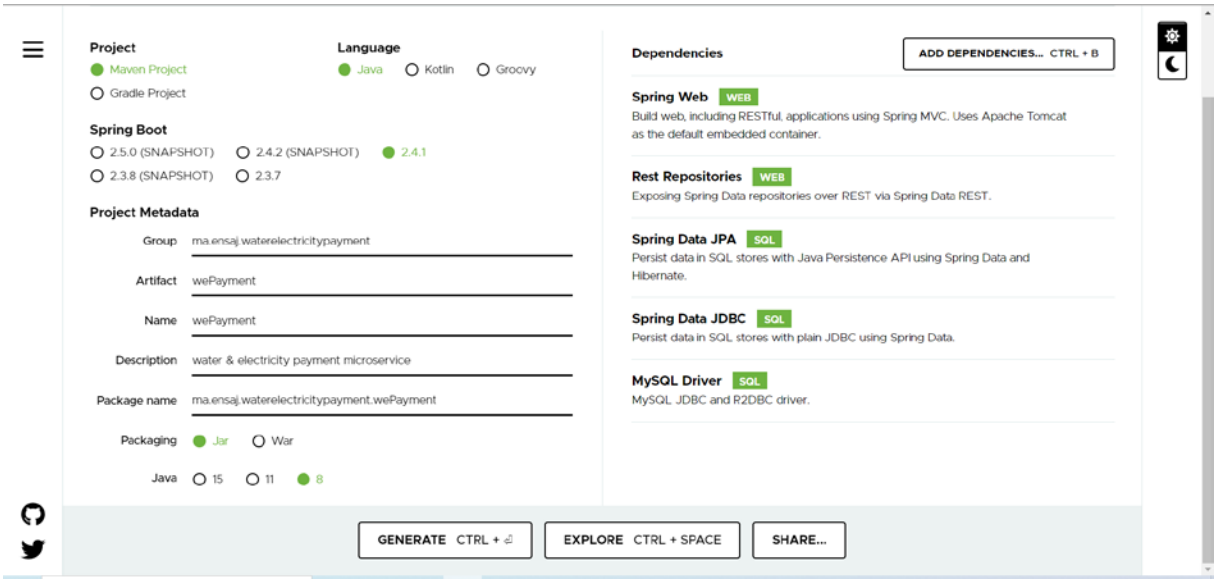


Figure 29:Initial creation of the application with Spring Initializr

2. Download the project and extract it

Figure 30:Download the project and extract it

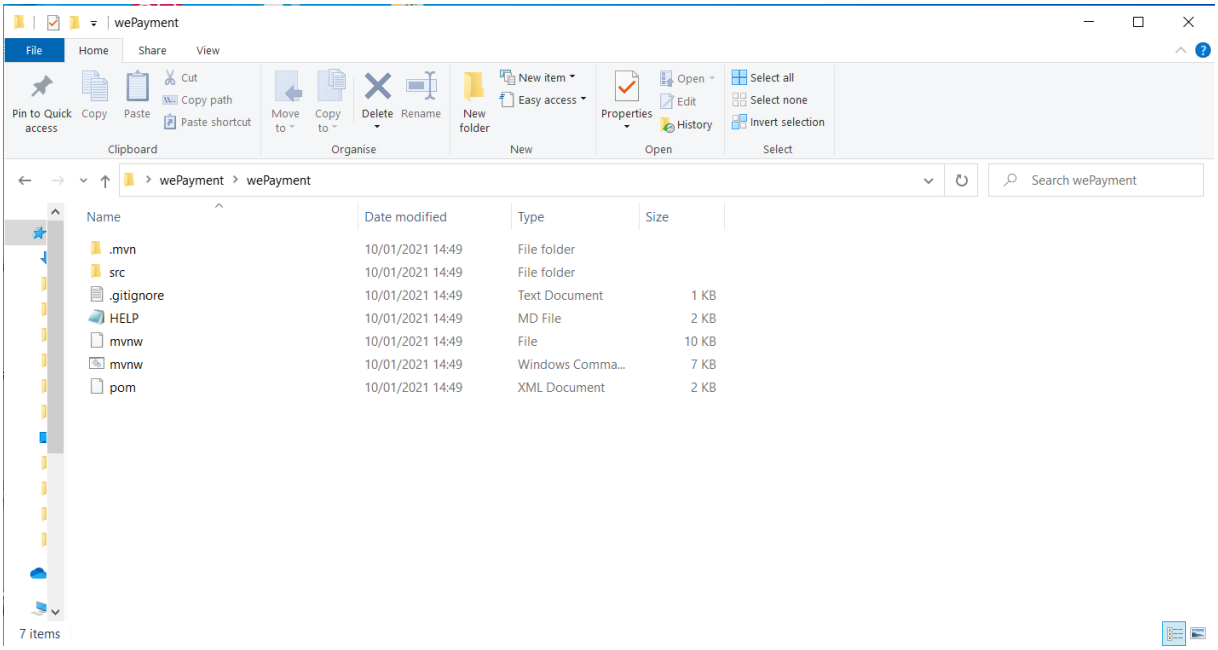


Figure 31:The project under the file explorer

3. Run the command : **mvn eclipse :eclipse** to make the project an eclipse project

```
C:\Users\hhass\Desktop\wePayment\wePayment>mvn eclipse:eclipse
[INFO] Scanning for projects...
[INFO]
[INFO] -------------< ma.ensaj.waterelectricitypayment:wePayment >-------------
[INFO] Building wePayment 0.0.1-SNAPSHOT
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] >>> maven-eclipse-plugin:2.10:eclipse (default-cli) > generate-resources @ wePayment >>>
[INFO]
[INFO] <<< maven-eclipse-plugin:2.10:eclipse (default-cli) < generate-resources @ wePayment <<<
[INFO]
[INFO]
[INFO] --- maven-eclipse-plugin:2.10:eclipse (default-cli) @ wePayment ---
[INFO] Using Eclipse Workspace: null
[INFO] Adding default classpath container: org.eclipse.jdt.launching.JRE_CONTAINER
[INFO] Resource directory's path matches an existing source directory but "test", "filtering" or "output" were different
.The resulting eclipse configuration may not accurately reflect the project configuration for src/main/resources
[INFO] Not writing settings - defaults suffice
[INFO] Wrote Eclipse project for "wePayment" to C:\Users\hhass\Desktop\wePayment\wePayment.
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  3.473 s
[INFO] Finished at: 2021-01-10T15:54:11+01:00
[INFO] ------------------------------------------------------------------------

C:\Users\hhass\Desktop\wePayment\wePayment>
```

Figure 32:make the project an eclipse project

4. Import the project under Eclipse.

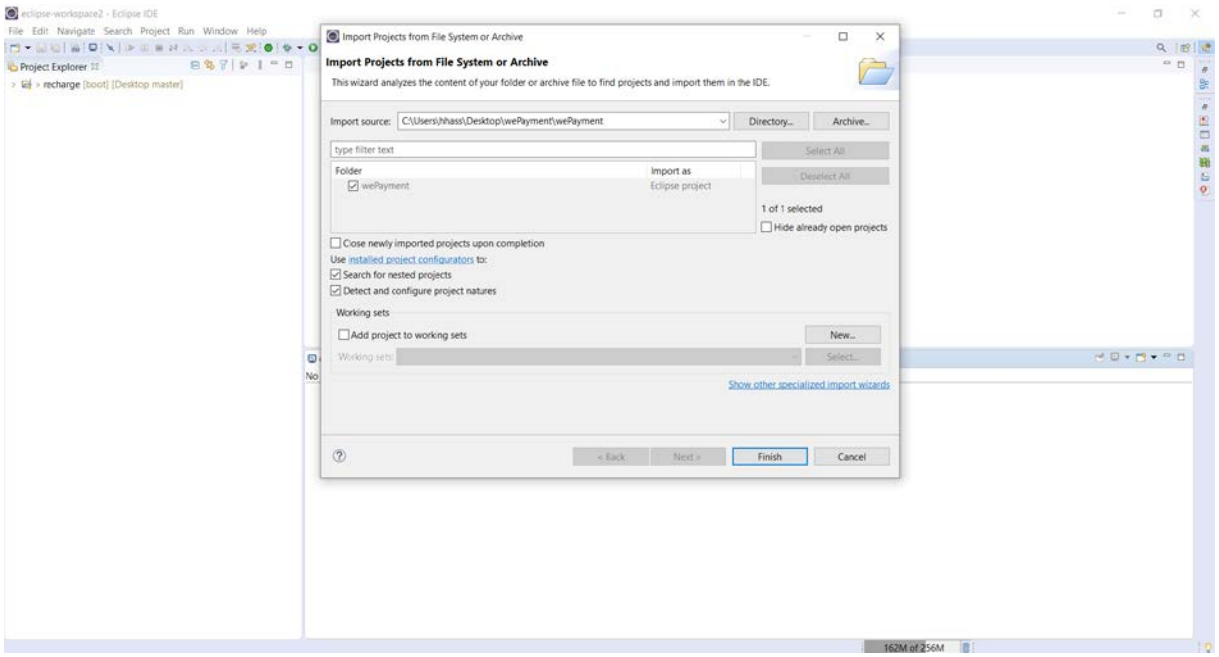


Figure 33:Import the project under Eclipse.

5. Project structure

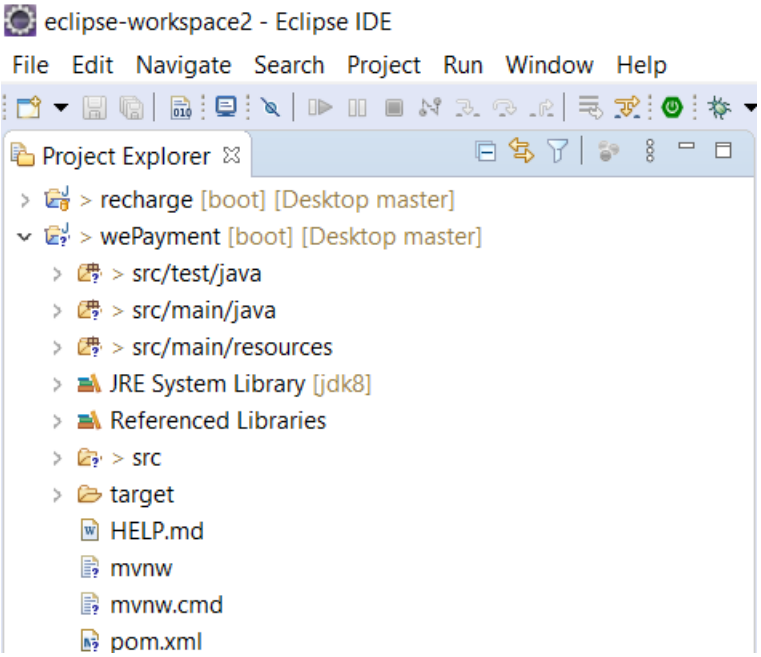


Figure 34:Project structure

6. The content of pom.xml :

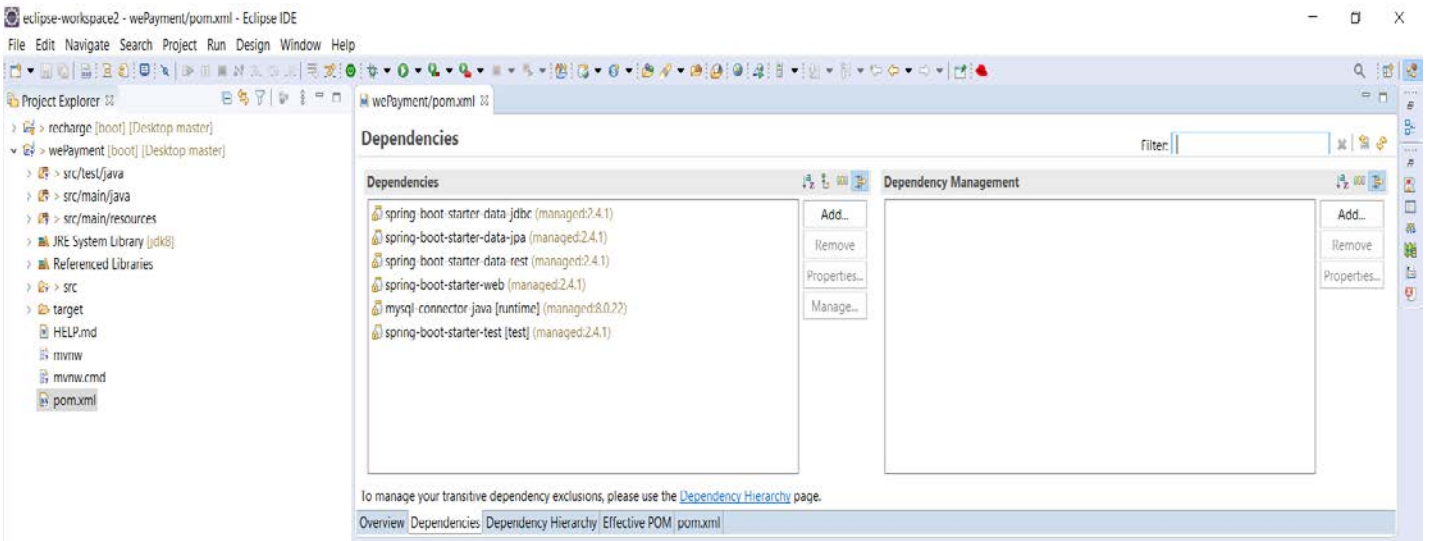


Figure 35: pom.xml dependencies

7. To indicate that we are using Java 8, it is usually necessary to specify it in several places in the pom (source version, target version). Here, you just have to set the **java.version** property:

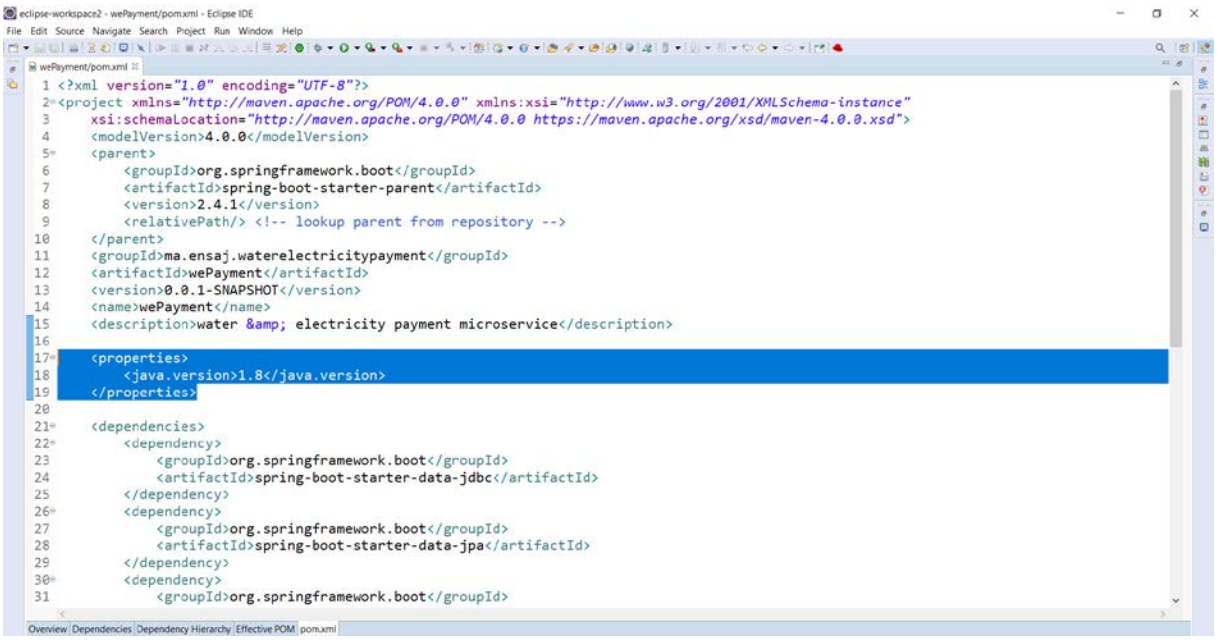


Figure 36:java.version

8. A Spring Boot application is a normal application, with main and annotations :

The **@SpringBootApplication** annotation is a meta-annotation that triggers auto-configuration and component scanning (in the classic Spring sense).

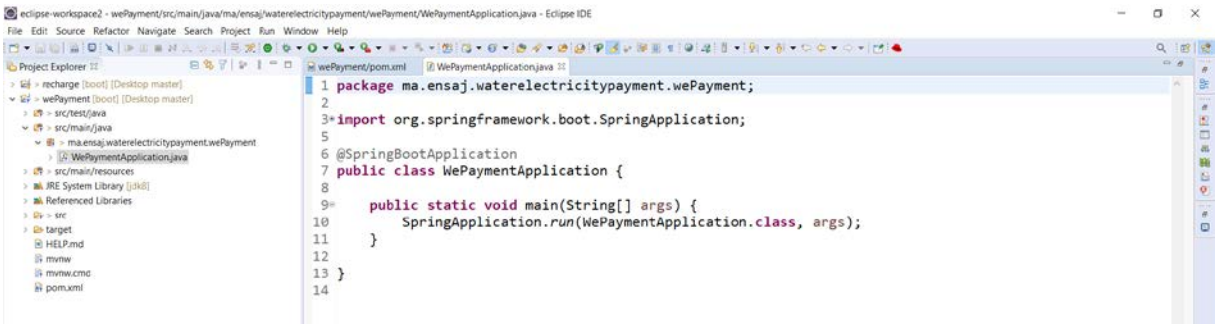


Figure 37:@SpringBootApplication

9. Configuration of the database and the Persistence Framework

Spring Boot comes with a built-in mechanism for configuring applications using a file called **application.properties.** It is located in the **src / main / resources** folder, as shown in the following figure.

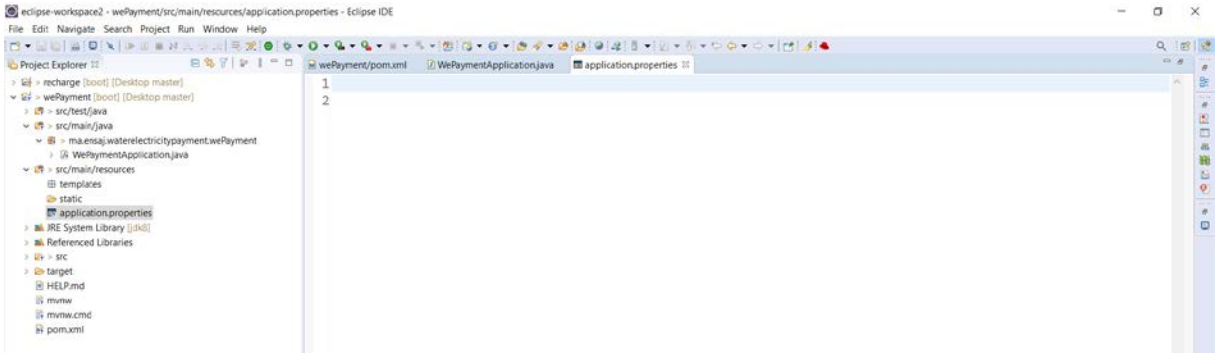


Figure 38:application.properties.

10. In the **application.properties** file, configure the database connection parameters and some parameters related to the **Persistence** Framework :

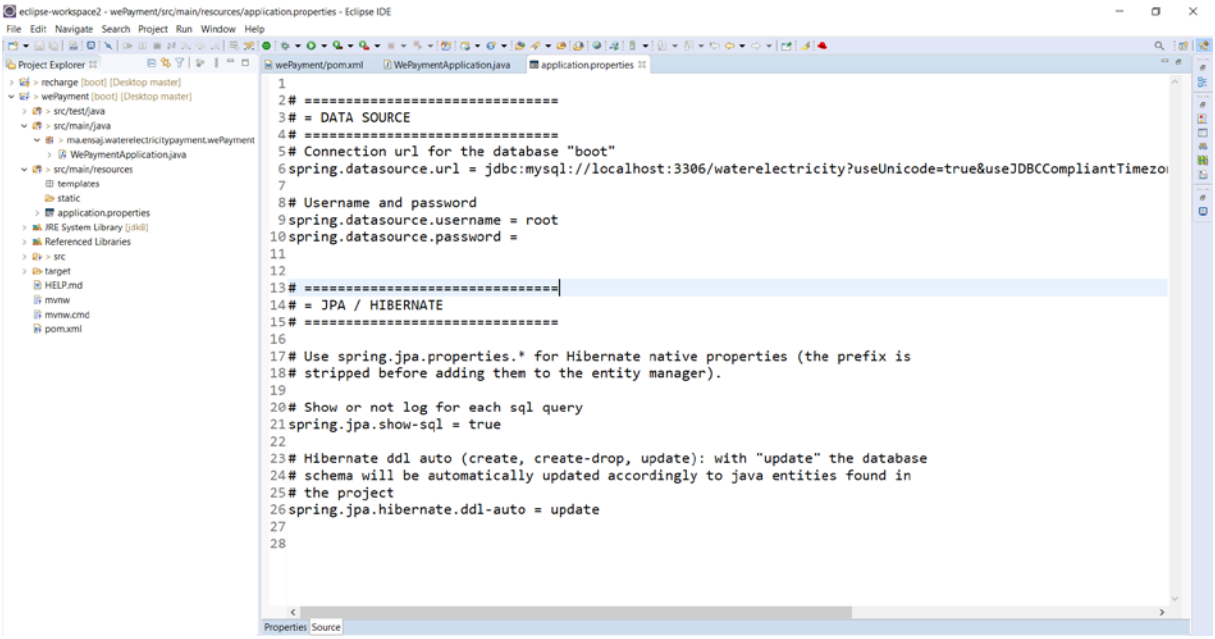


Figure 39:application.properties

11. Create the "**waterelectricity**" database under MySQL.

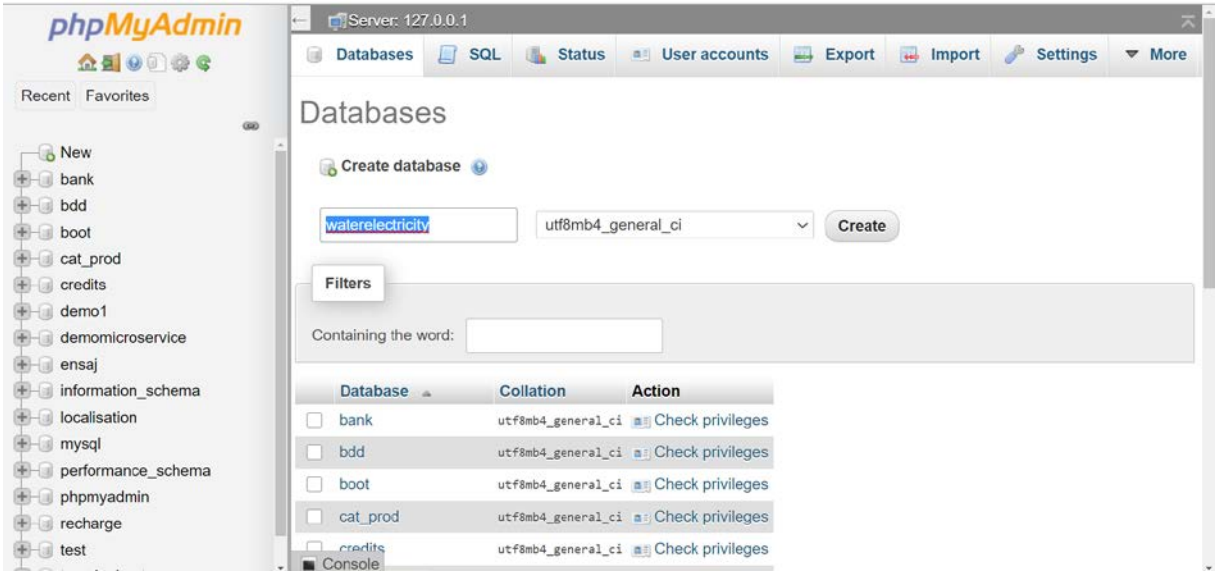


Figure 40:Create the "waterelectricity" database under MySQL.

12. Creation of business classes

Create the "**WaterElectricity**" class in a sub-package of the main package (here: ma.ensaj.waterelectricitypayment.beans):

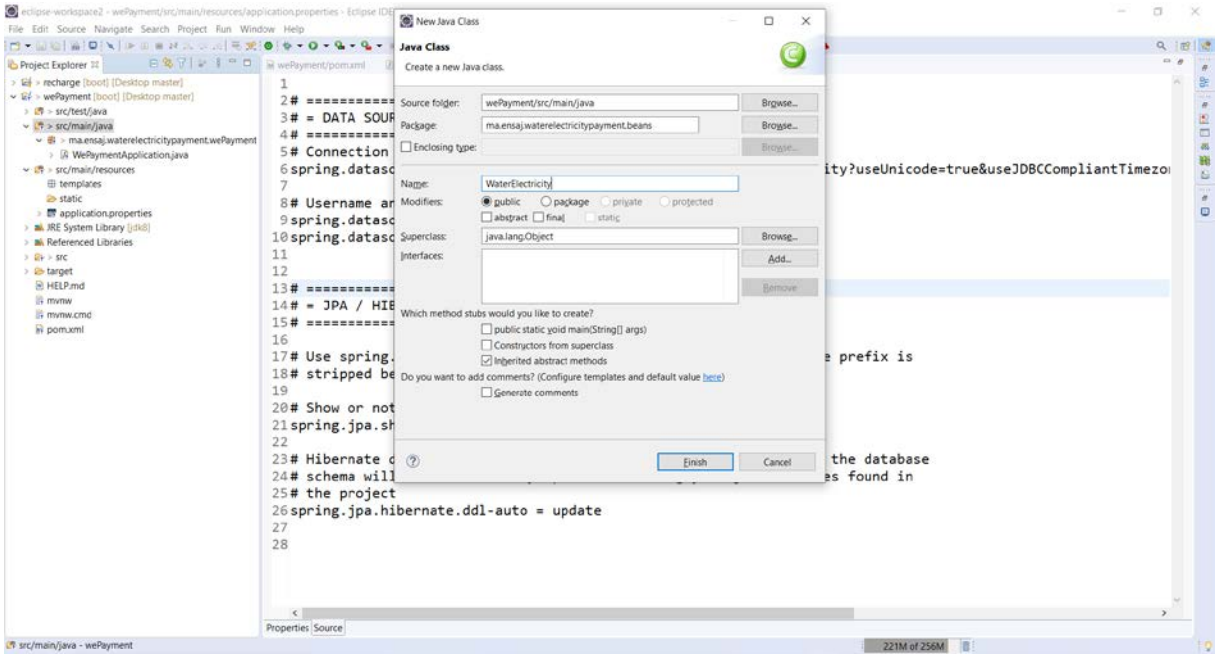


Figure 41:Create the "WaterElectricity" class

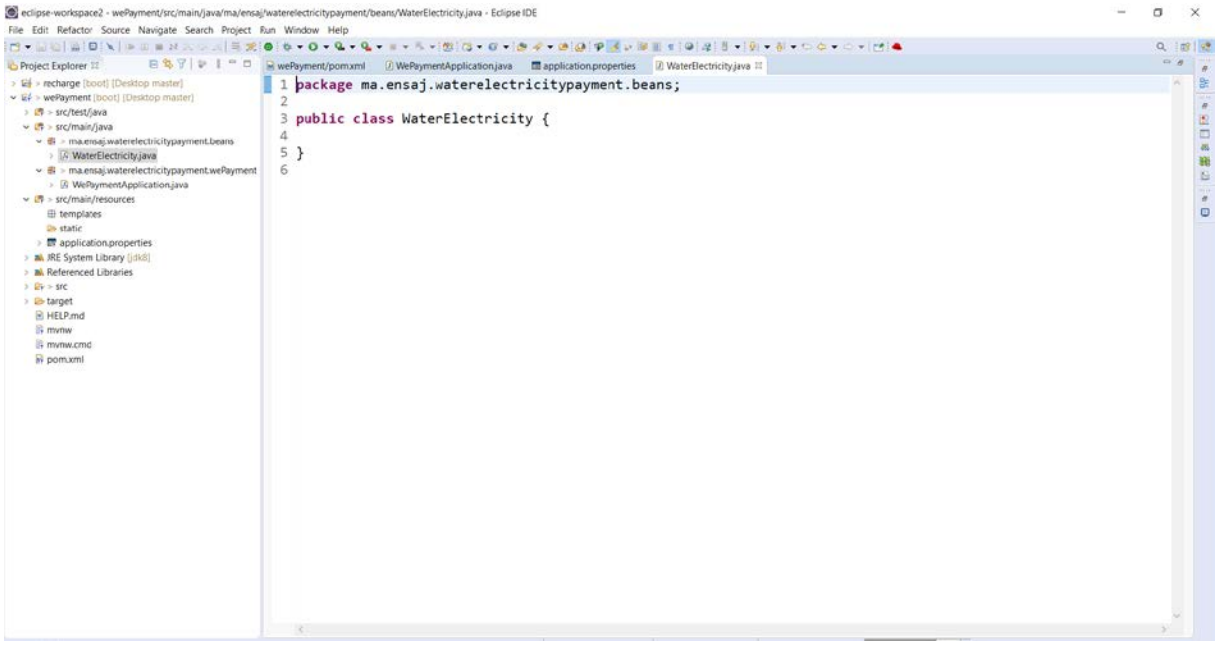


Figure 42: "WaterElectricity" class

13. Class diagram of **WaterElectricity** entity

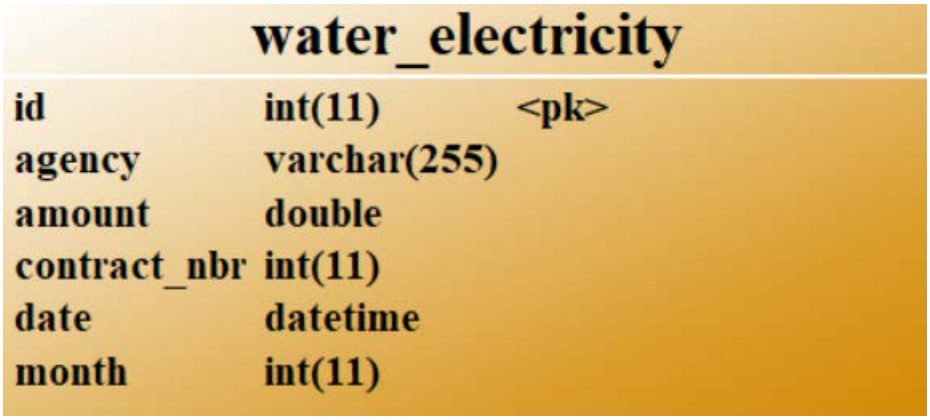


Figure 43:WaterElectricity entity

14. Implementation of WaterElectricity class

15. Attributes

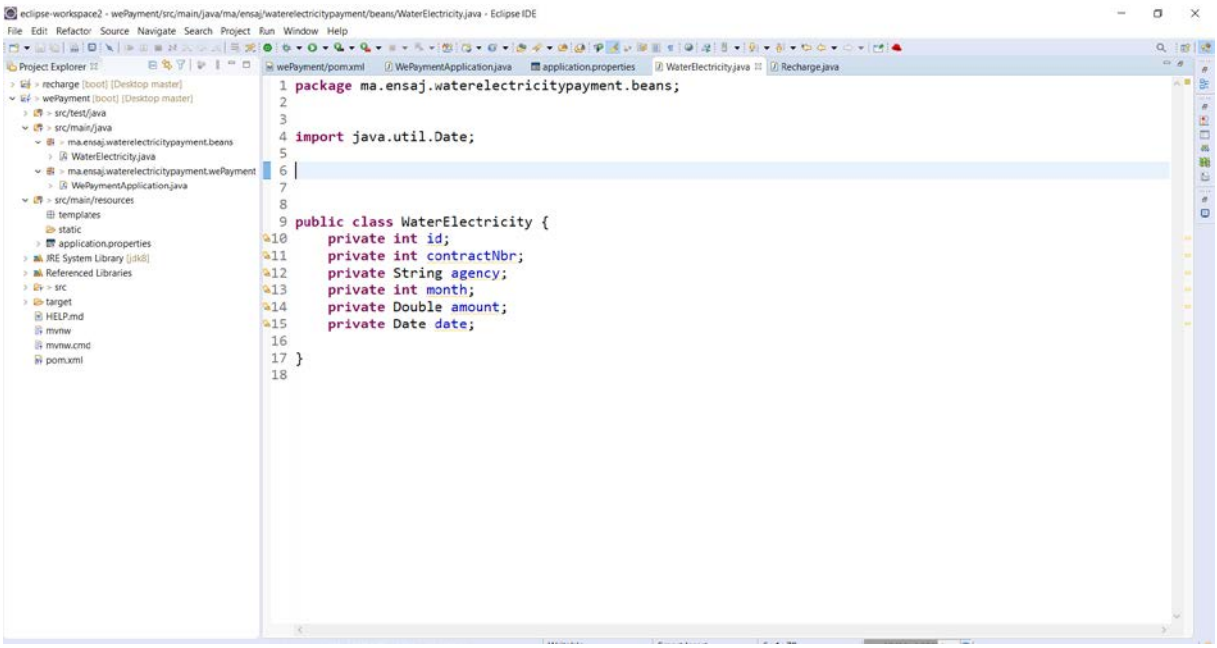


Figure 44:Attributes

16. Generate getters and setters

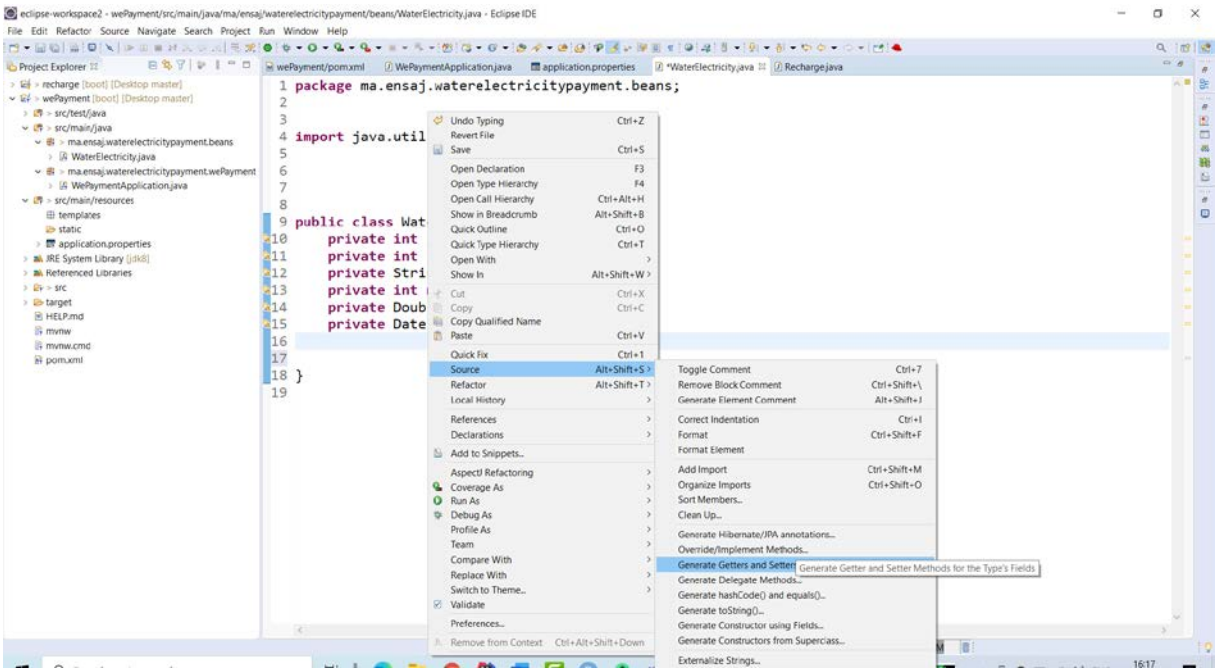


Figure 45:Generate getters and setters

17. Select all attributes

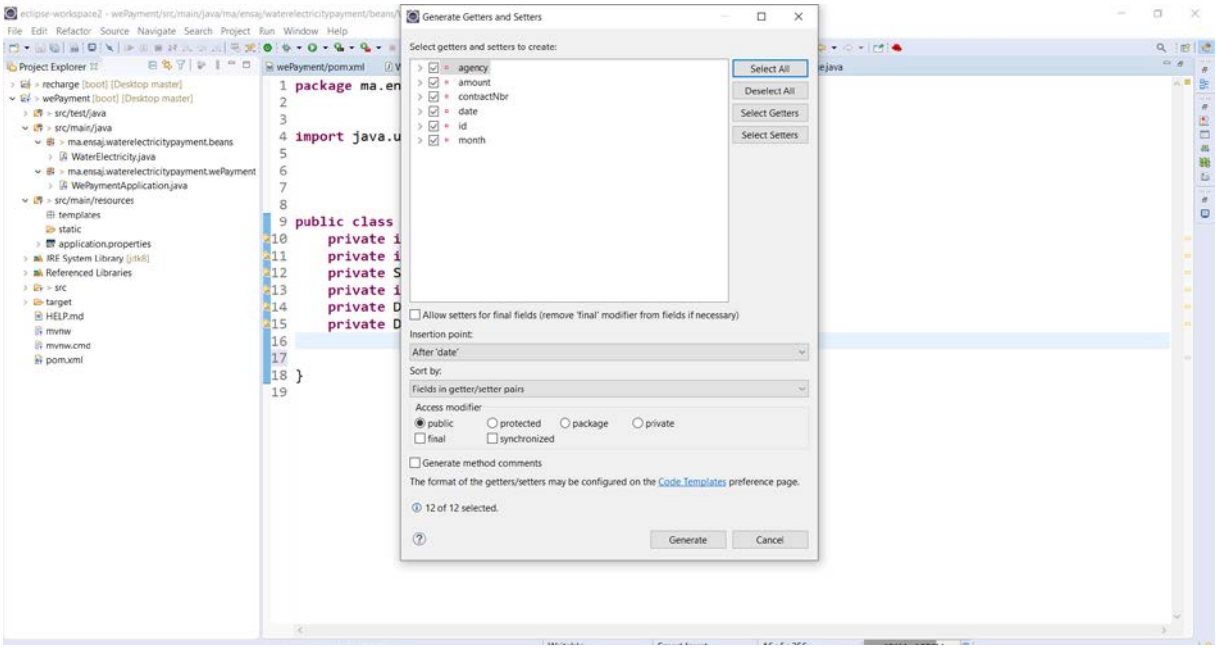


Figure 46:Select all attributes

18. Getters and setters are generated

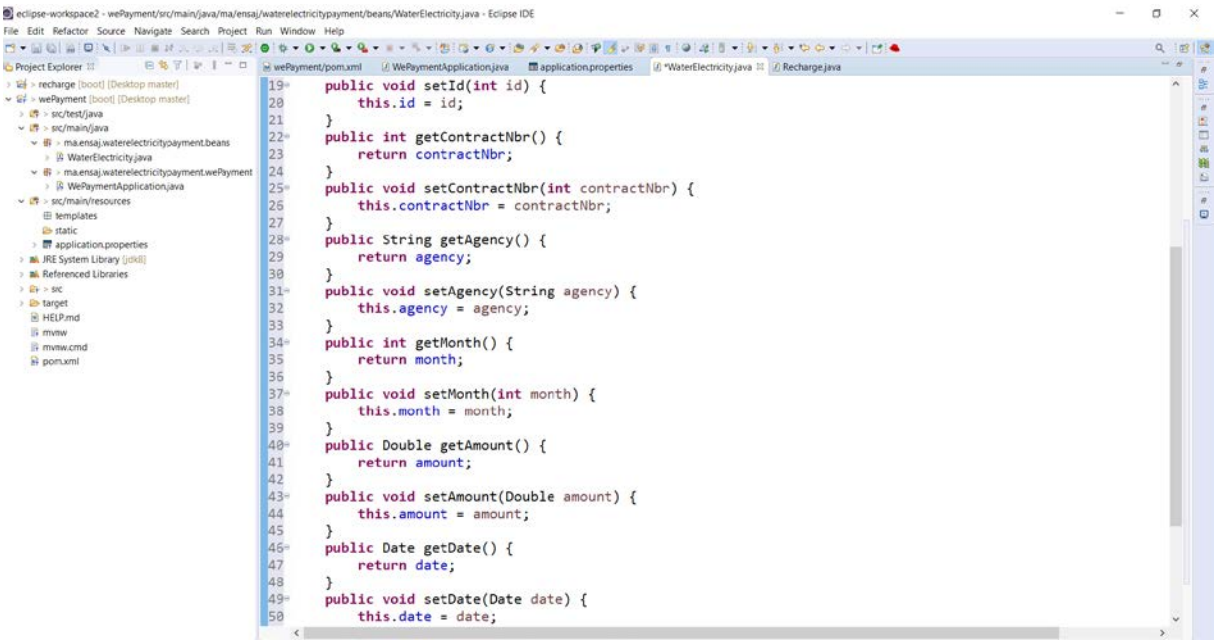


Figure 47:Getters and setters are generated

19. Generate the constructor

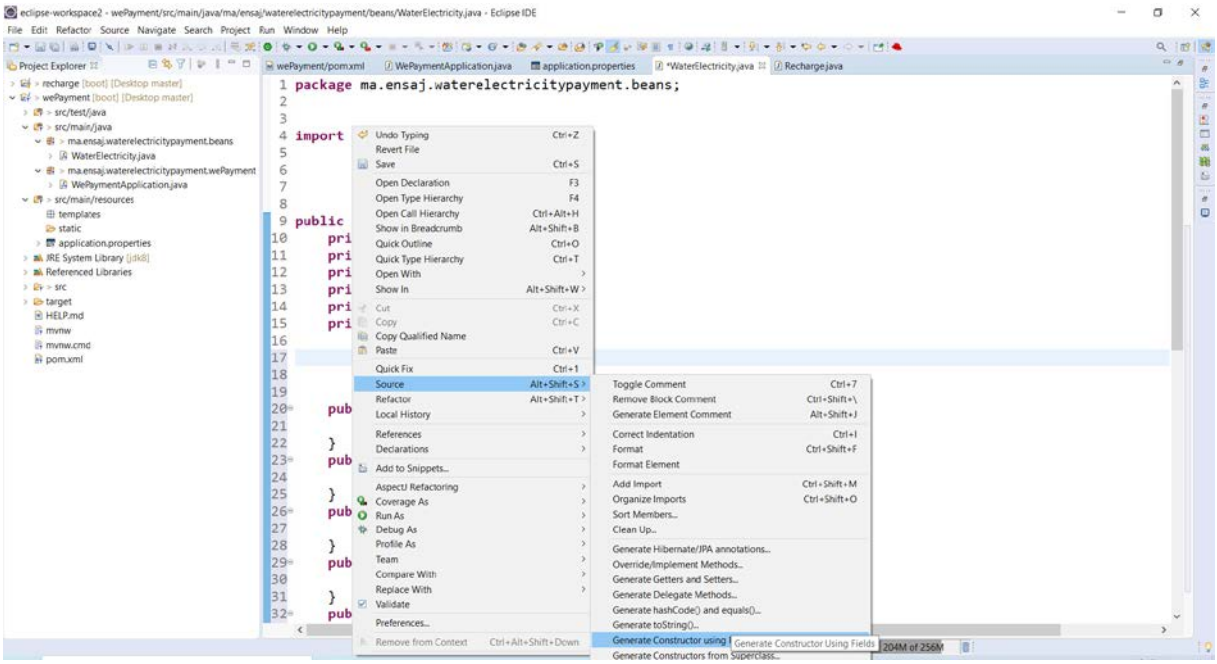


Figure 48:Generate the constructor

20. Constructors

```java
package ma.ensaj.waterelectricitypayment.beans;


import java.util.Date;



public class WaterElectricity {
    private int id;
    private int contractNbr;
    private String agency;
    private int month;
    private Double amount;
    private Date date;

    public WaterElectricity() {
        super();
    }
    public WaterElectricity(int id, int contractNbr, String agency, int month, Double amount, Date date) {
        super();
        this.id = id;
        this.contractNbr = contractNbr;
        this.agency = agency;
        this.month = month;
        this.amount = amount;
        this.date = date;
    }


    public int getId() {
        return id;
```

Figure 49:Constructors

21. Generate the toString() method

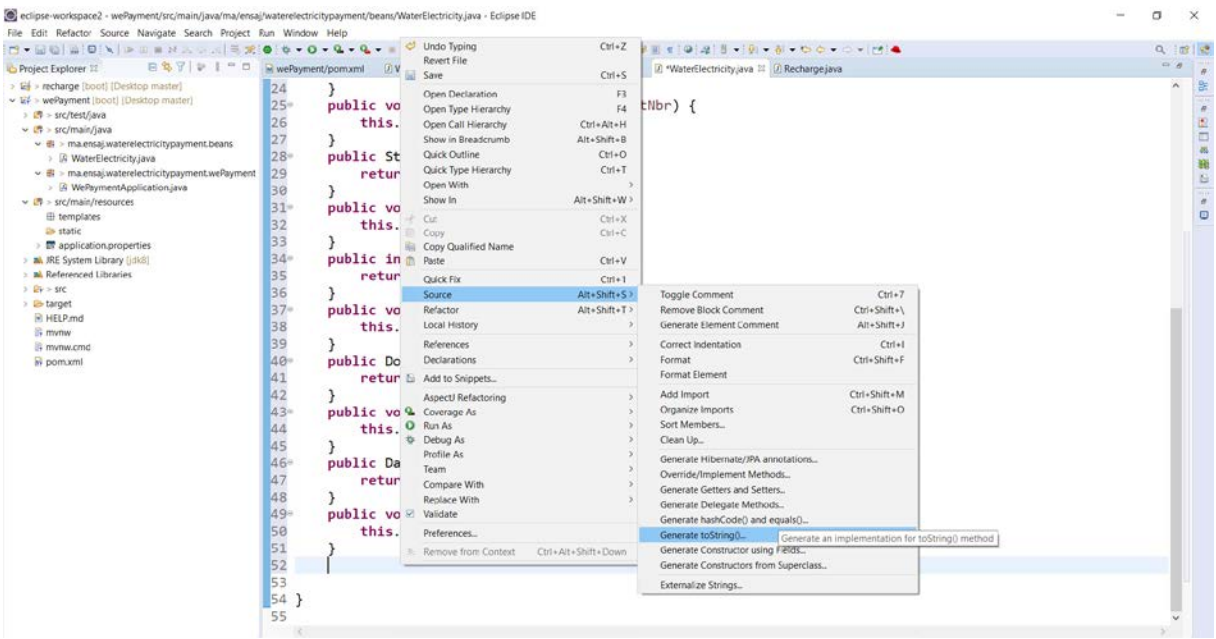


Figure 50:Generate the toString() method

22. toString():

```
    this.date = date;
}
@Override
public String toString() {
    return "WaterElectricity [id=" + id + ", contractNbr=" + contractNbr + ", agency=" + agency + ", mo
            + ", amount=" + amount + ", date=" + date + "]";
}
```

Figure 51:toString()

23. We add the validator data dependency in the pom.xml

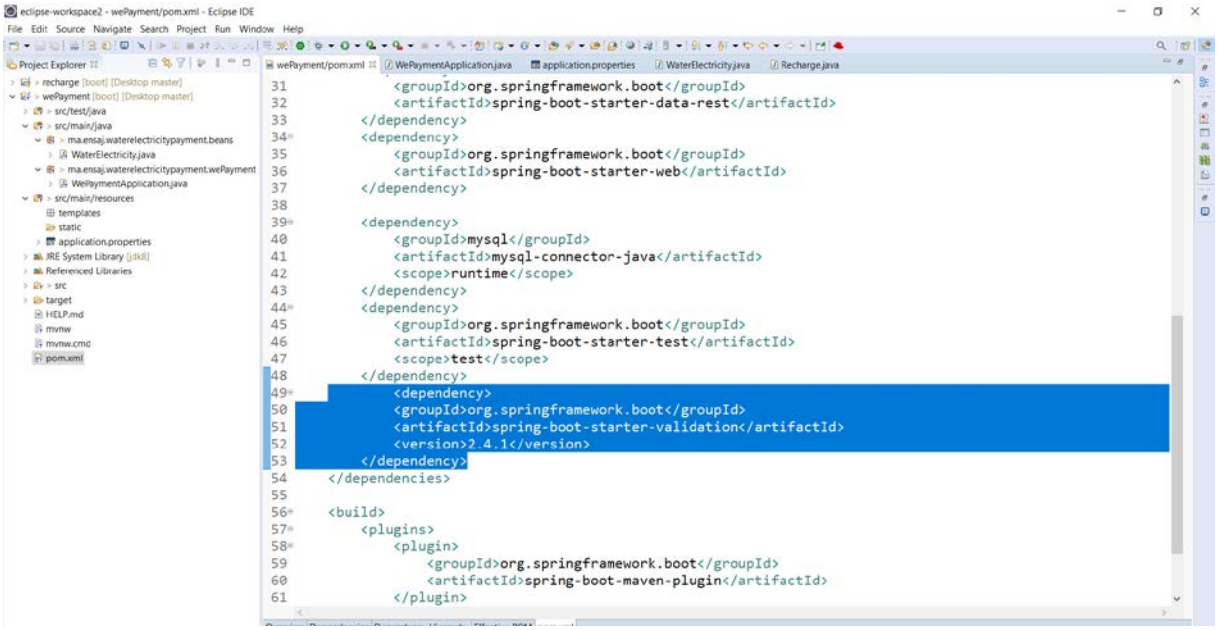
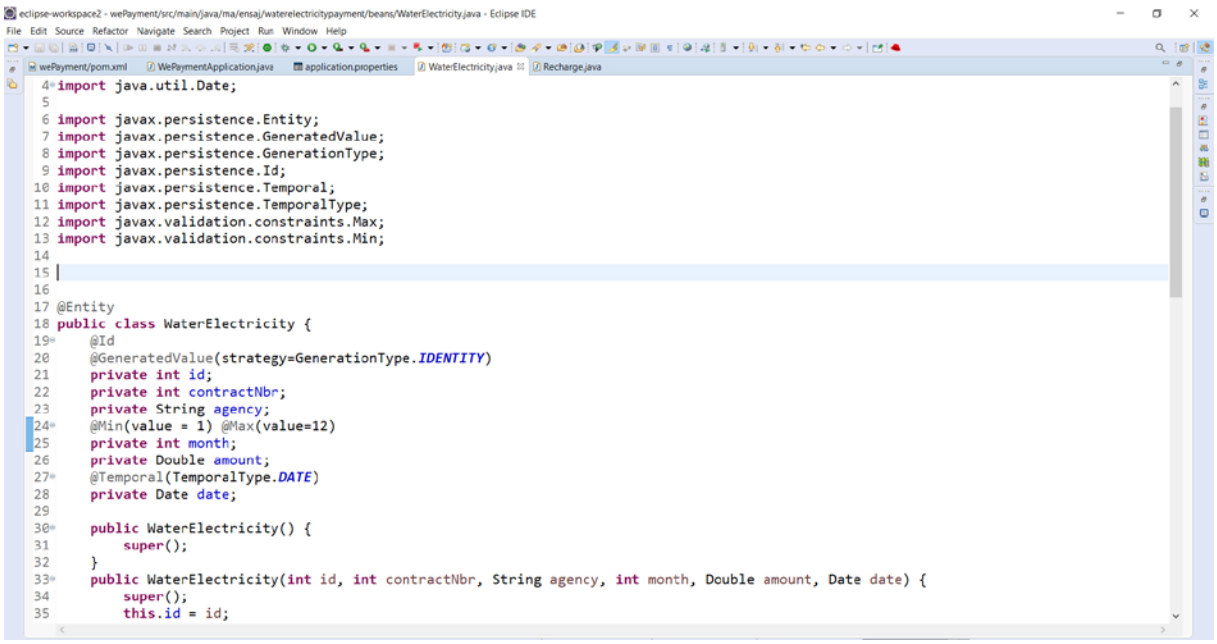


Figure 52:We add the validator data dependency in the pom.xml

24. We add some persistence annotations.



```java
import java.util.Date;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Temporal;
import javax.persistence.TemporalType;
import javax.validation.constraints.Max;
import javax.validation.constraints.Min;


@Entity
public class WaterElectricity {
    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private int id;
    private int contractNbr;
    private String agency;
    @Min(value = 1) @Max(value=12)
    private int month;
    private Double amount;
    @Temporal(TemporalType.DATE)
    private Date date;

    public WaterElectricity() {
        super();
    }
    public WaterElectricity(int id, int contractNbr, String agency, int month, Double amount, Date date) {
        super();
        this.id = id;
```

Figure 53:We add some persistence annotations.

25. To start the application, use **mvn spring-boot:run** command

```
C:\Users\hhass\Desktop\wePayment\wePayment>mvn spring-boot:run
```

Figure 54:To start the application, use mvn spring-boot:run command

26. We can see now in the cmd **Started WePaymentApplication**

```
2021-01-10 16:25:49.545  INFO 5492 --- [           main] org.apache.catalina.core.StandardEngine  : Starting Servlet engine: [Apache To
mcat/9.0.41]
2021-01-10 16:25:49.731  INFO 5492 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApp
licationContext
2021-01-10 16:25:49.731  INFO 5492 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initial
ization completed in 3850 ms
2021-01-10 16:25:49.988  INFO 5492 --- [           main] o.hibernate.jpa.internal.util.LogHelper  : HHH000204: Processing PersistenceUn
itInfo [name: default]
2021-01-10 16:25:50.040  INFO 5492 --- [           main] org.hibernate.Version                    : HHH000412: Hibernate ORM core versi
on 5.4.25.Final
2021-01-10 16:25:50.197  INFO 5492 --- [           main] o.hibernate.annotations.common.Version   : HCANN000001: Hibernate Commons Anno
tations {5.1.2.Final}
2021-01-10 16:25:50.356  INFO 5492 --- [           main] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Starting...
2021-01-10 16:25:50.823  INFO 5492 --- [           main] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Start completed.
2021-01-10 16:25:50.855  INFO 5492 --- [           main] org.hibernate.dialect.Dialect            : HHH000400: Using dialect: org.hiber
nate.dialect.MySQL55Dialect
2021-01-10 16:25:51.131  INFO 5492 --- [           main] o.h.e.t.j.p.i.JtaPlatformInitiator       : HHH000490: Using JtaPlatform implem
entation: [org.hibernate.engine.transaction.jta.platform.internal.NoJtaPlatform]
2021-01-10 16:25:51.157  INFO 5492 --- [           main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactor
y for persistence unit 'default'
2021-01-10 16:25:51.317  WARN 5492 --- [           main] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled
by default. Therefore, database queries may be performed during view rendering. Explicitly configure spring.jpa.open-in-view to disable
 this warning
2021-01-10 16:25:51.843  INFO 5492 --- [           main] o.s.s.concurrent.ThreadPoolTaskExecutor  : Initializing ExecutorService 'appli
cationTaskExecutor'
2021-01-10 16:25:52.689  INFO 5492 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8080 (ht
tp) with context path ''
2021-01-10 16:25:52.707  INFO 5492 --- [           main] m.e.w.wePayment.WePaymentApplication     : Started WePaymentApplication in 7.8
11 seconds (JVM running for 8.703)
```

Figure 55:We can see now in the cmd Started WePaymentApplication

27. check if the "**waterelectricity**" table is created.

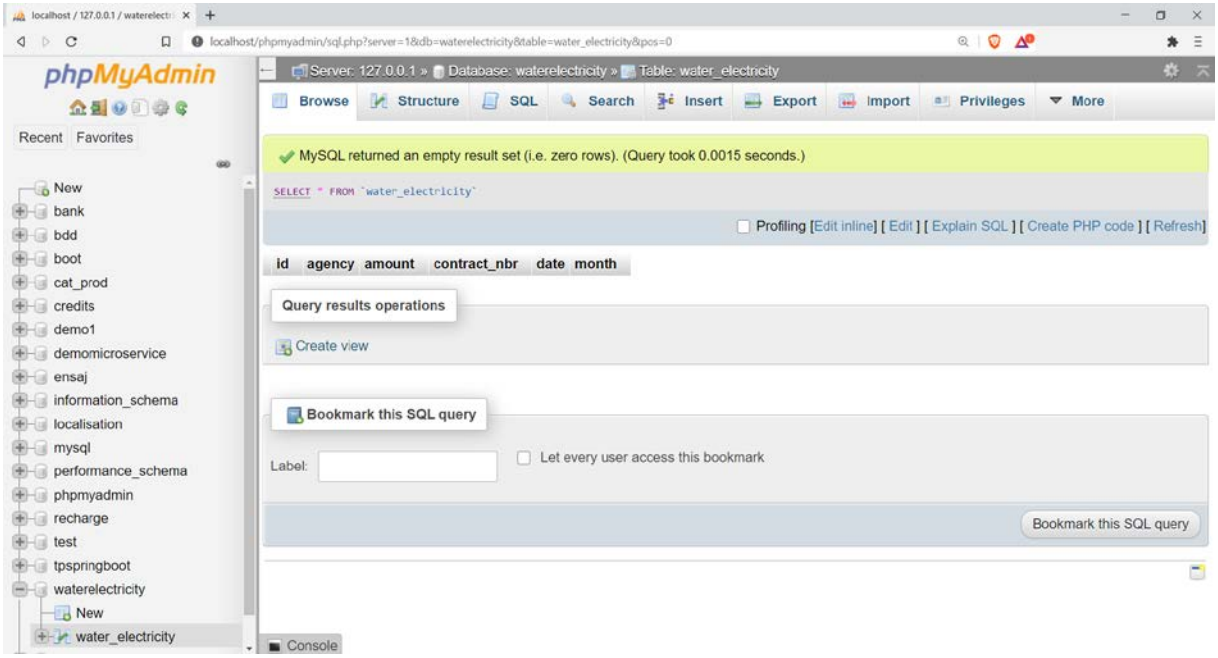


Figure 56:check if the "waterelectricity" table is created.

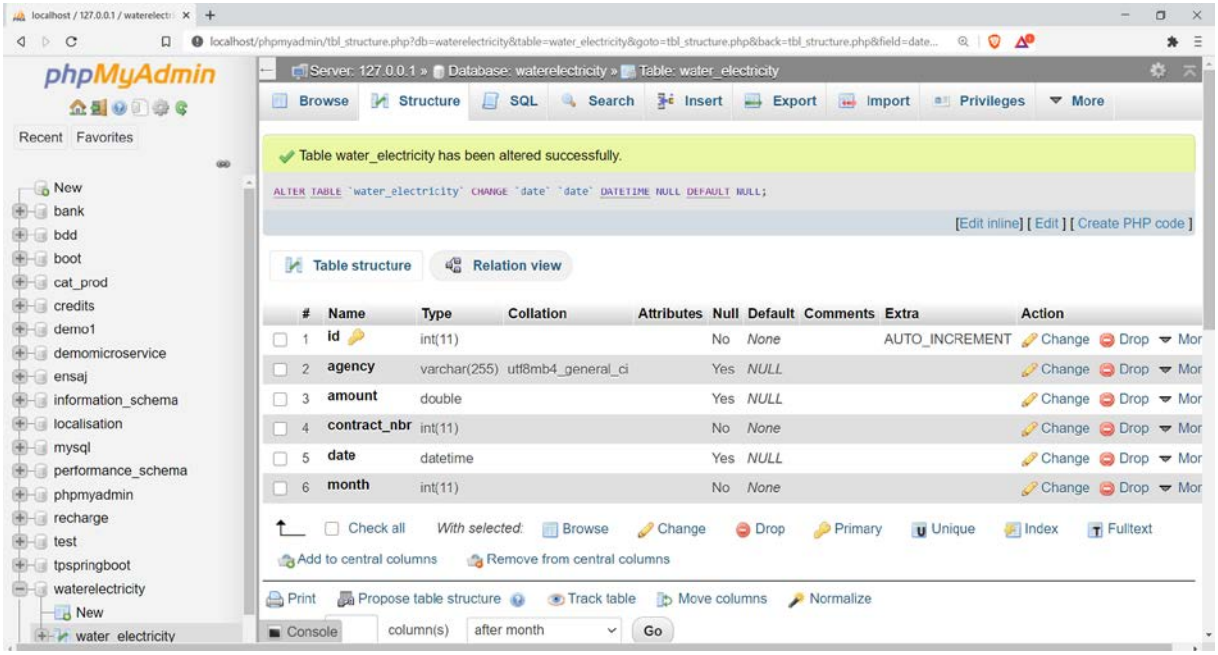


Figure 57:"waterelectricity" table structure

28. Insert a new row to the table waterelectricity

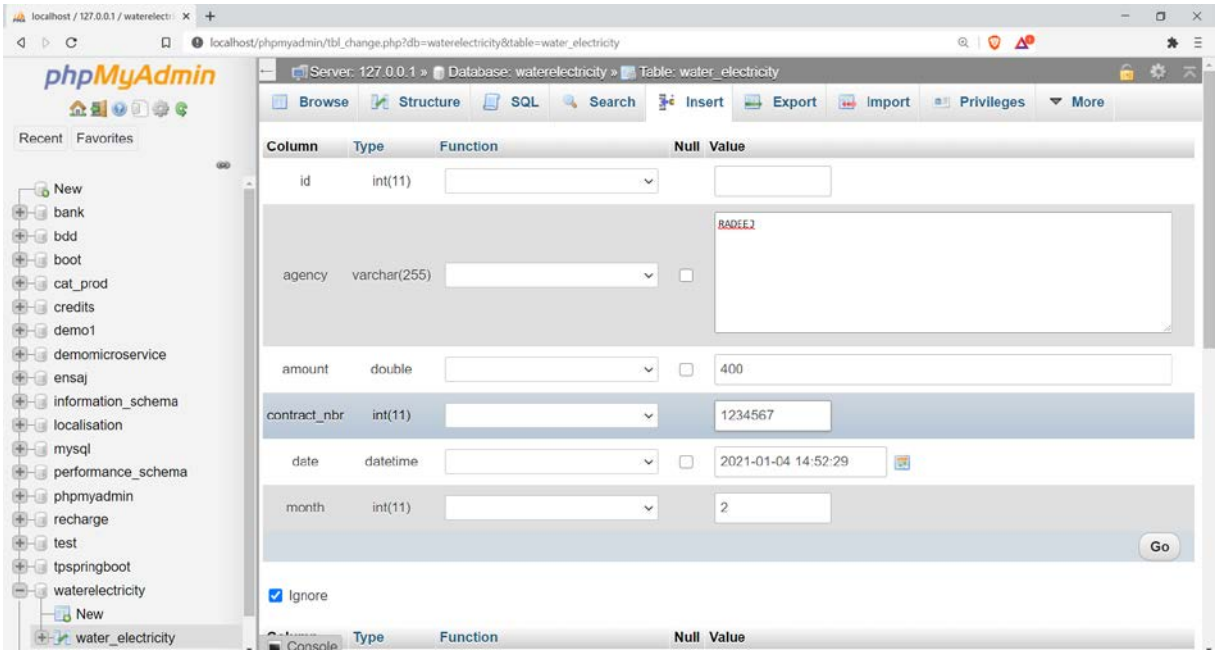


Figure 58:Insert a new row to the table waterelectricity

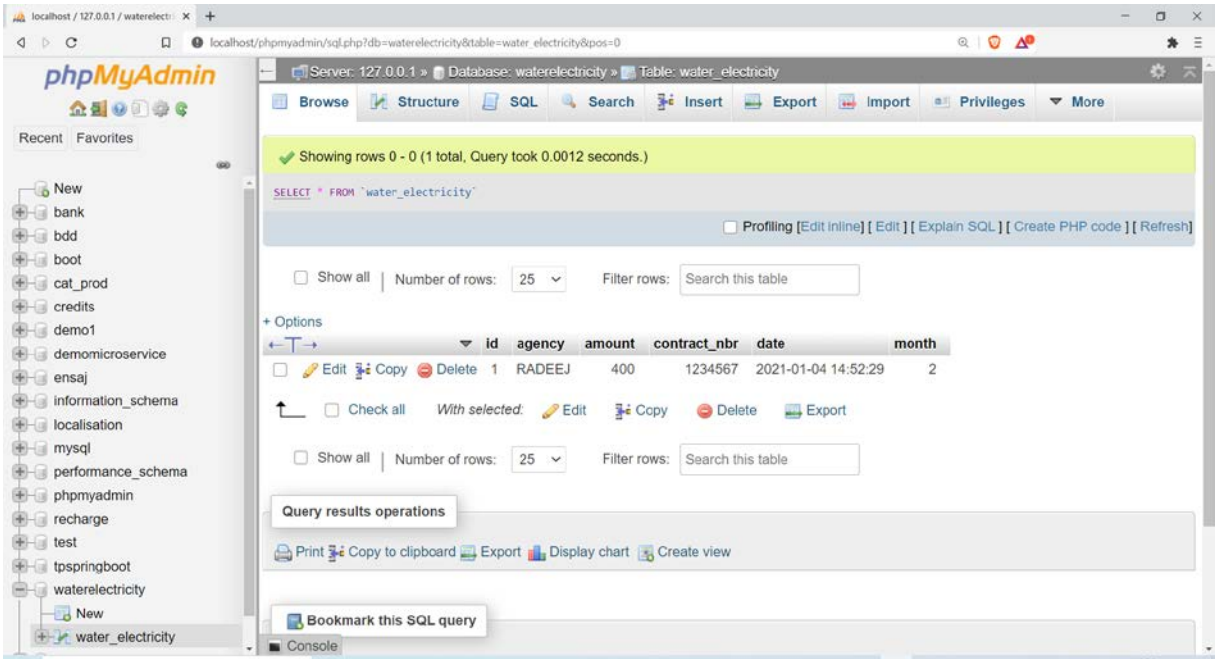


Figure 59:The new row was added successfully

29. Creation of repositories

Spring Data JPA provides a data access layer implementation for a Spring application.

It is a very convenient brick because it allows not reinventing the data access wheel with each new application, and thus allowing you to focus on the business side.

Create the "**WeRepository**" interface in a sub-package -called repository- of the main package :

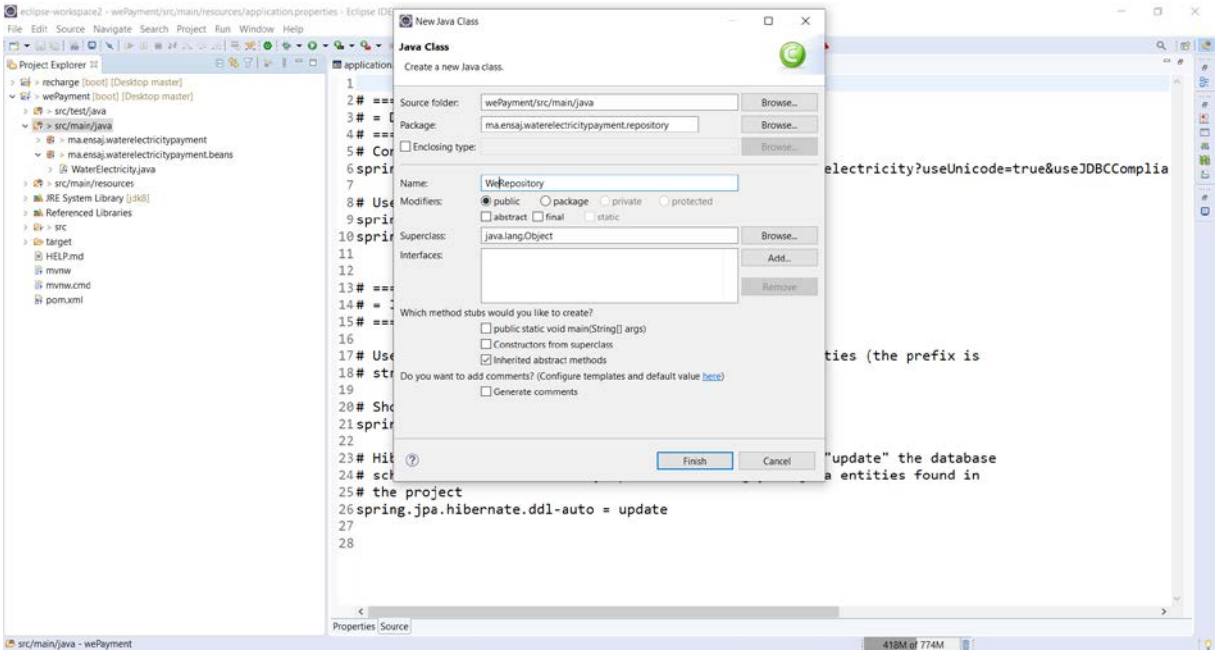


Figure 60:Create the "WeRepository" interface

30. In the interface add the method :

WaterElectricity findById(int id);

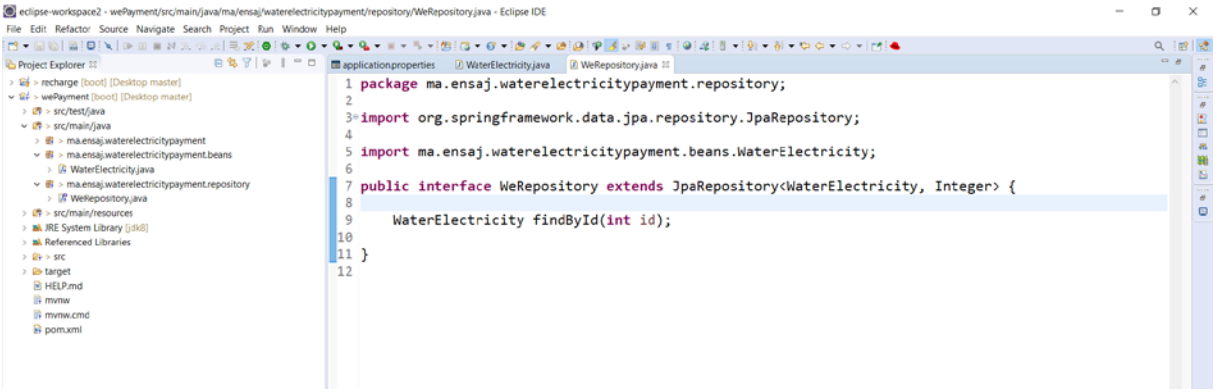


Figure 61:WeRepository

31. Creation of REST controllers

REST stands for REpresentational State Transfer. It is developed by Roy Thomas Fielding, who also developed HTTP. The main goal of RESTful Web Services is to make Web services more efficient. RESTful Web Services attempt to define services using the various concepts already present in HTTP. REST is an architectural approach, not a protocol.

It does not define the standard message exchange format. We can create REST services with XML and JSON. JSON is a more popular format with REST. Key abstraction is a resource in REST. A resource can be anything. It is accessible via a Uniform Resource Identifier (URI).

32. Create the "**WeController**" class in a sub-package-called controller- of the main package:

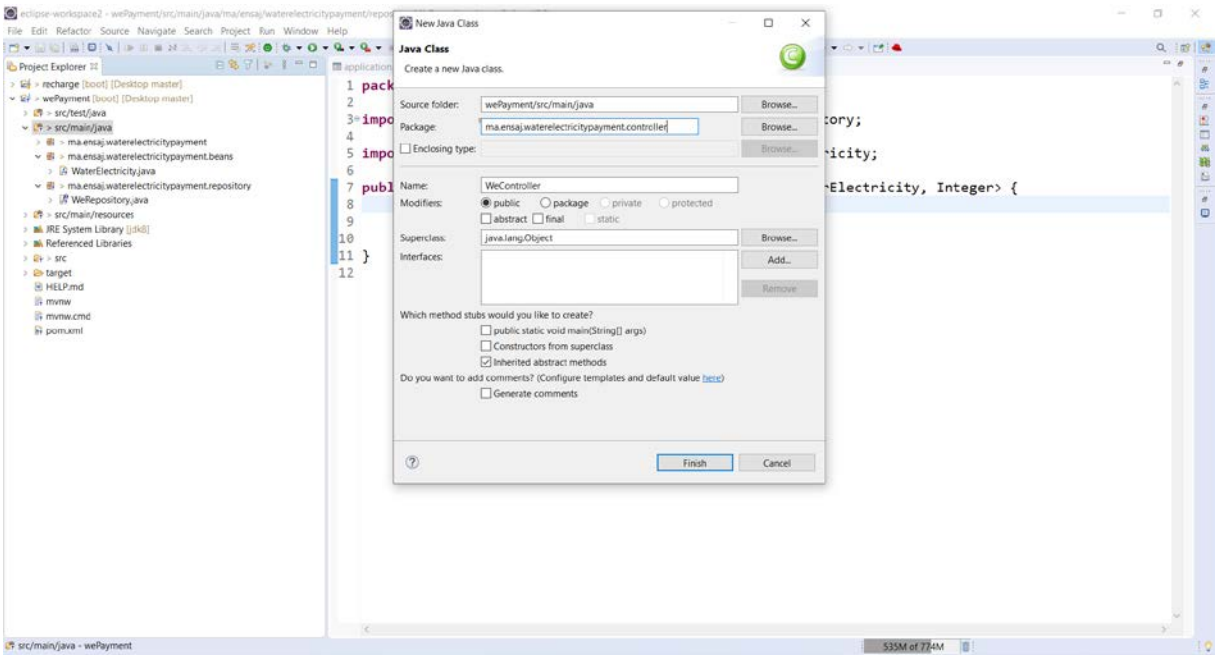


Figure 62:Create the "WeController" class

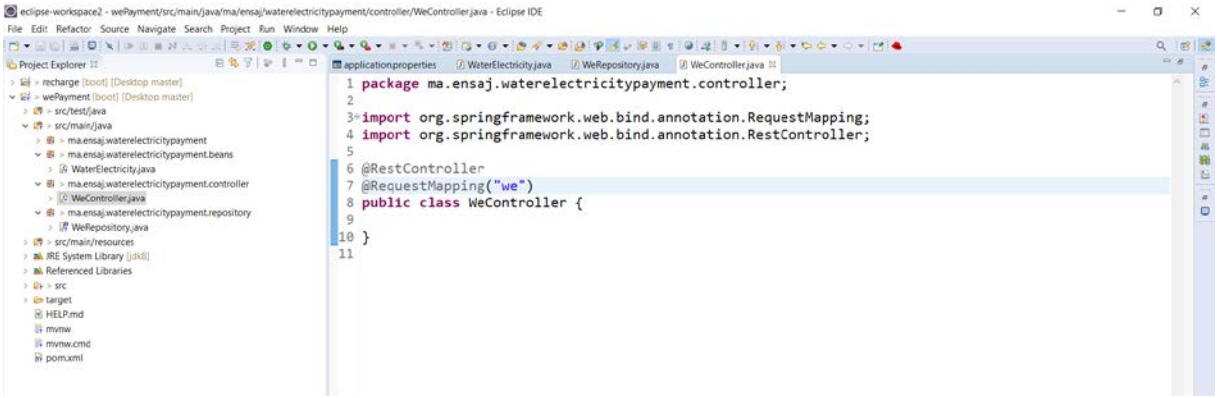
33. WeController

Figure 63:WeController

34. Add annotations **@RestController** and **@RequestMapping("we")**

N.B:

- The **@RestController** annotation was introduced in Spring 4.0 to simplify the creation of RESTful web services. It's a convenience annotation that combines @Controller and @ResponseBody – which eliminates the need to annotate every request handling method of the controller class with the @ResponseBody annotation.
- **@RequestMapping** is used to map web requests to Spring Controller methods.



35. Add **CRUD** web services to the **WeController**

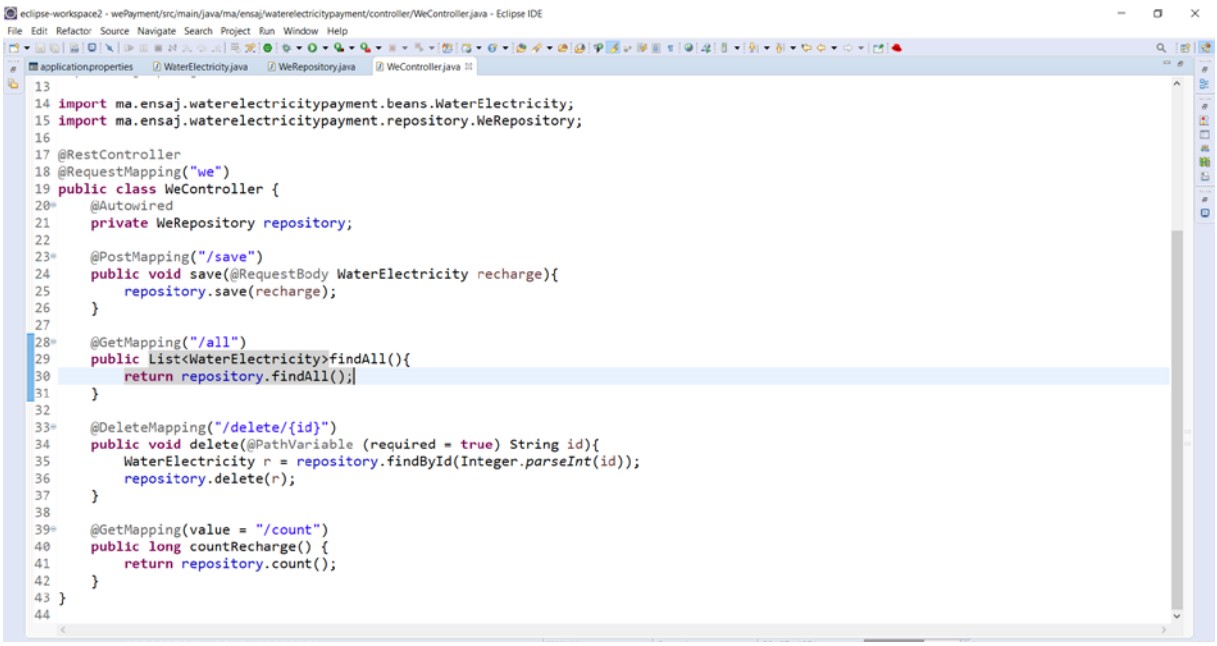


Figure 64:Add CRUD web services to the WeController

36. Testing controllers with ARC(Advanced Rest Client)

37. Launch the project again

```
C:\Users\hhass\Desktop\wePayment\wePayment>mvn spring-boot:run
```

Figure 65:Launch the project again

38. List of water and electricity payments

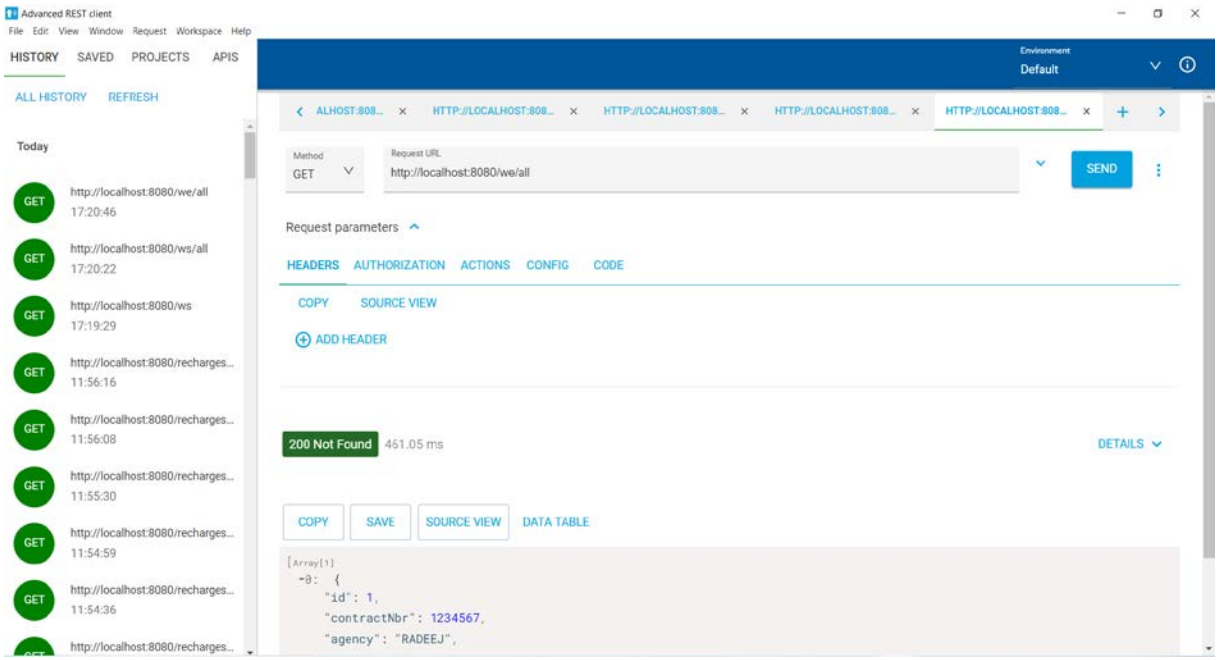


Figure 66:List of water and electricity payments

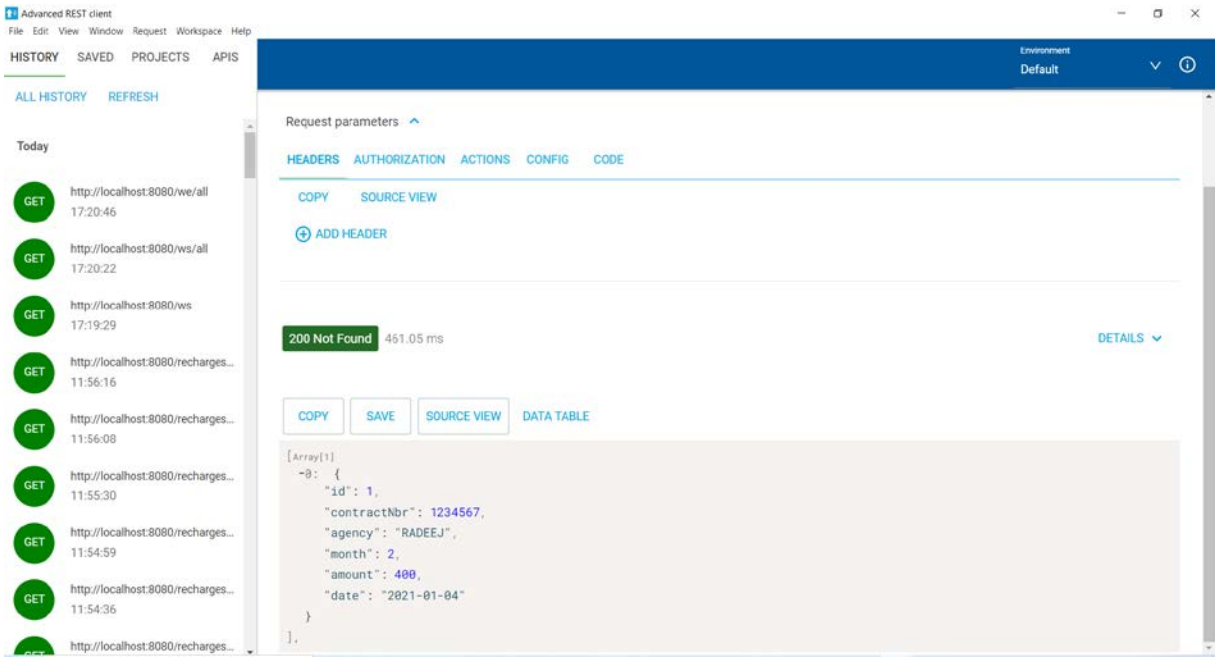


Figure 67:200 OK

39. Creation(Add some water and electricity payments)

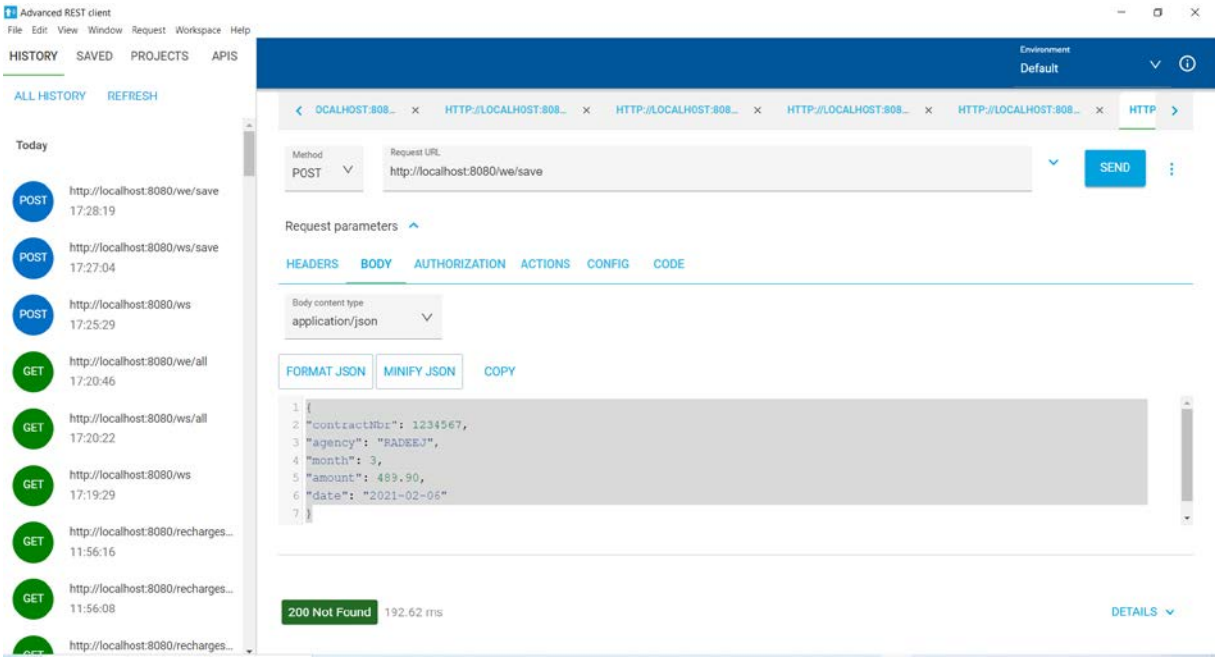


Figure 68:Creation(Add some water and electricity payments)

40. Check if payments were added successfully to the database

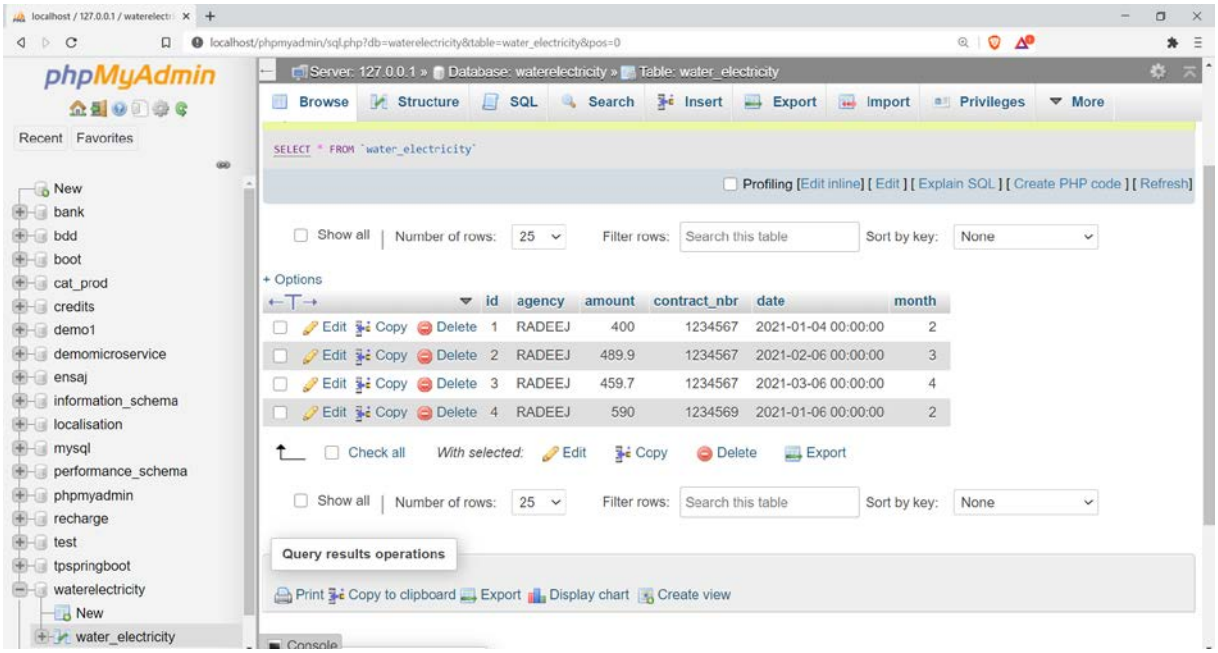


Figure 69:Check if payments were added successfully to the database

41. Update the payment with id 4(agency from RADEEJ to ONE)

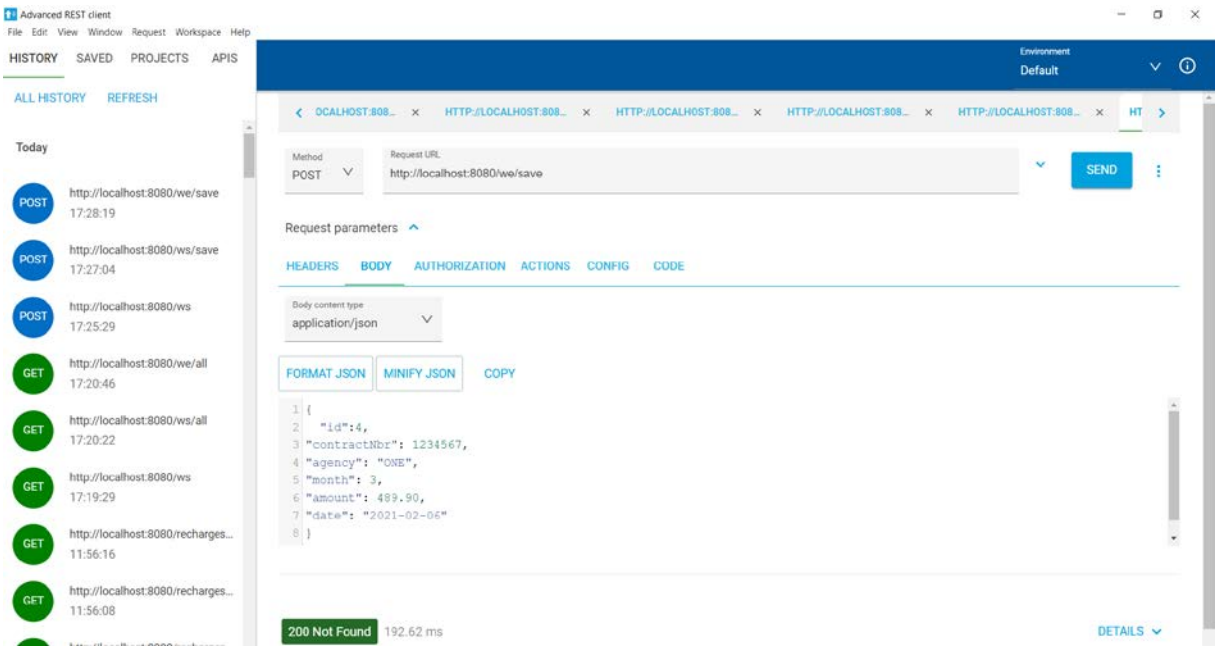


Figure 70:Update the payment with id 4(agency from RADEEJ to ONE)

42. The update is well passed

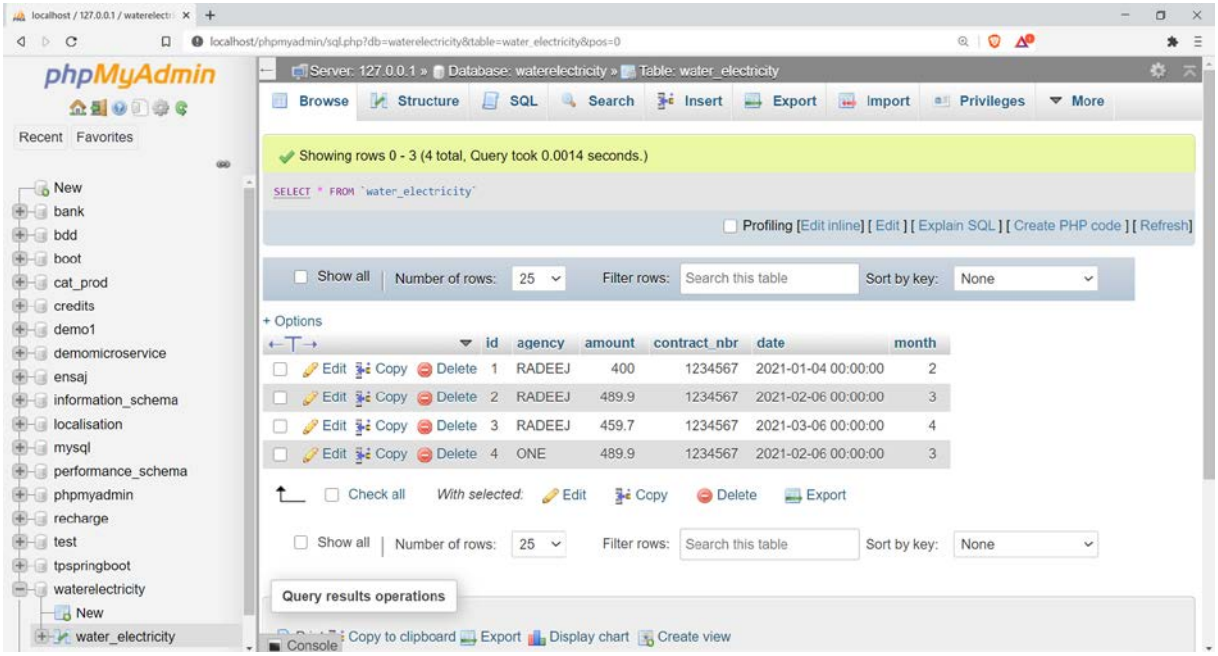


Figure 71:The update is well passed

43. Delete payment with id 3

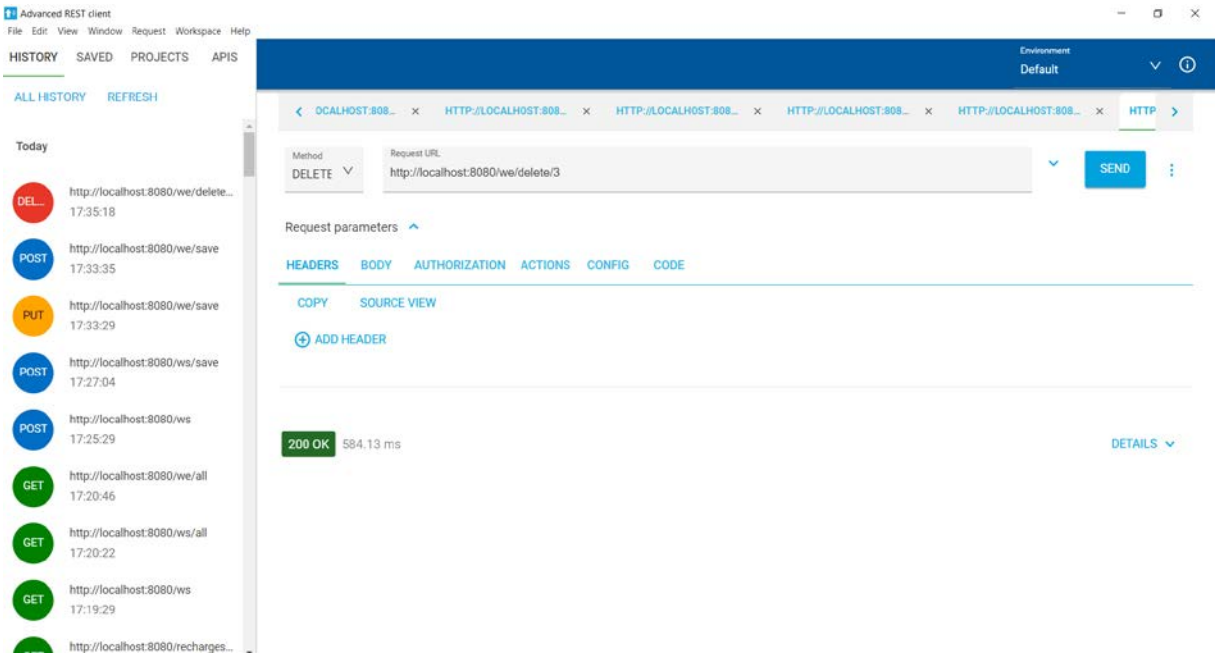


Figure 72:Delete payment with id 3

44. The payment with id 3 was deleted successfully

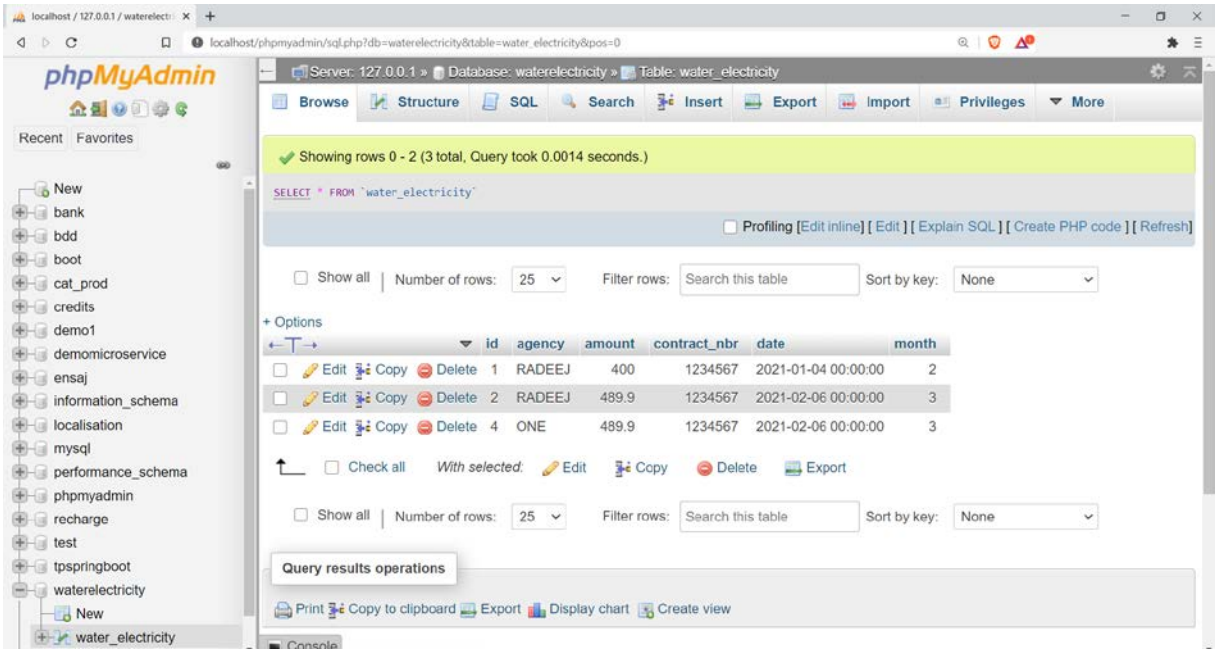
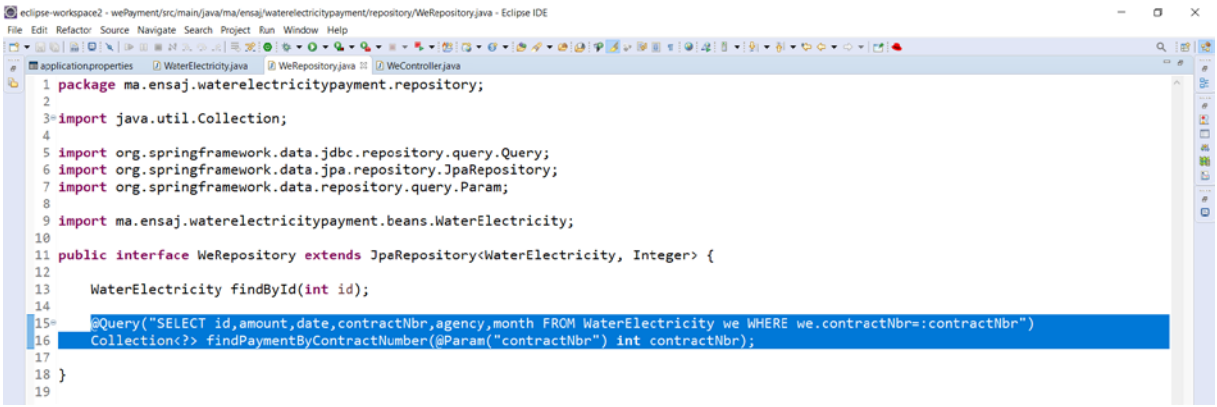


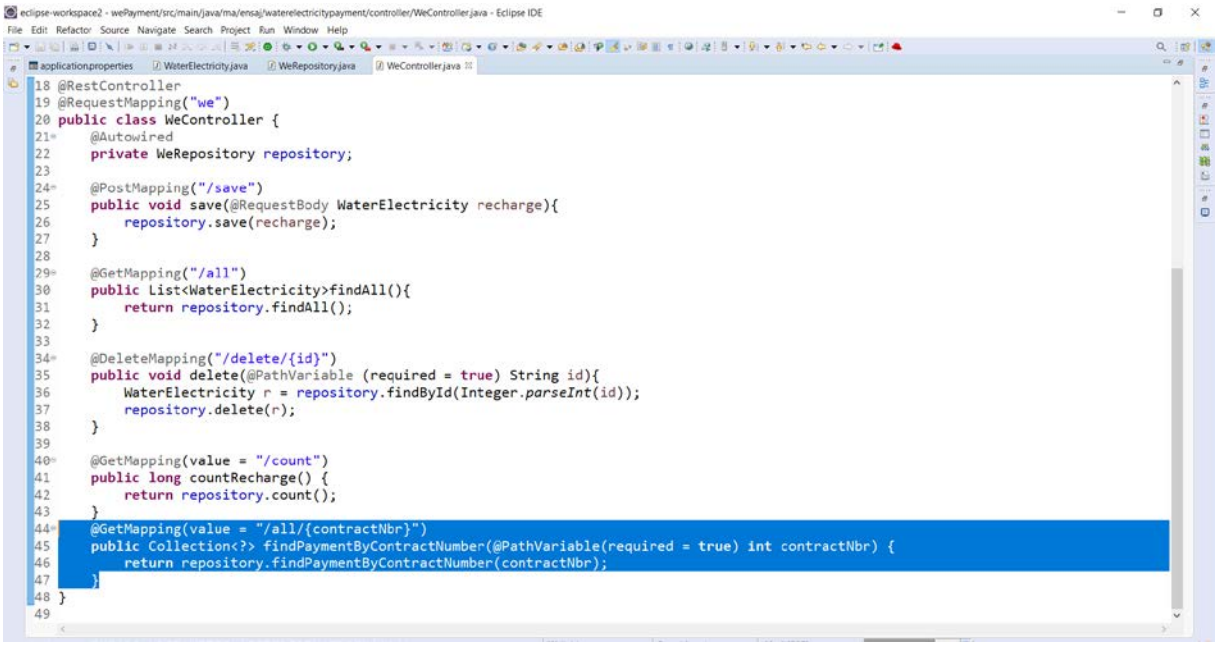
Figure 73:The payment with id 3 was deleted successfully

45. Get payments by contract number



```
package ma.ensaj.waterelectricitypayment.repository;

import java.util.Collection;

import org.springframework.data.jdbc.repository.query.Query;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.repository.query.Param;

import ma.ensaj.waterelectricitypayment.beans.WaterElectricity;

public interface WeRepository extends JpaRepository<WaterElectricity, Integer> {

    WaterElectricity findById(int id);

    @Query("SELECT id,amount,date,contractNbr,agency,month FROM WaterElectricity we WHERE we.contractNbr=:contractNbr")
    Collection<?> findPaymentByContractNumber(@Param("contractNbr") int contractNbr);

}
```

Figure 74:Get payments by contract number



```
@RestController
@RequestMapping("we")
public class WeController {
    @Autowired
    private WeRepository repository;

    @PostMapping("/save")
    public void save(@RequestBody WaterElectricity recharge){
        repository.save(recharge);
    }

    @GetMapping("/all")
    public List<WaterElectricity>findAll(){
        return repository.findAll();
    }

    @DeleteMapping("/delete/{id}")
    public void delete(@PathVariable (required = true) String id){
        WaterElectricity r = repository.findById(Integer.parseInt(id));
        repository.delete(r);
    }

    @GetMapping(value = "/count")
    public long countRecharge() {
        return repository.count();
    }
    @GetMapping(value = "/all/{contractNbr}")
    public Collection<?> findPaymentByContractNumber(@PathVariable(required = true) int contractNbr) {
        return repository.findPaymentByContractNumber(contractNbr);
    }
}
```

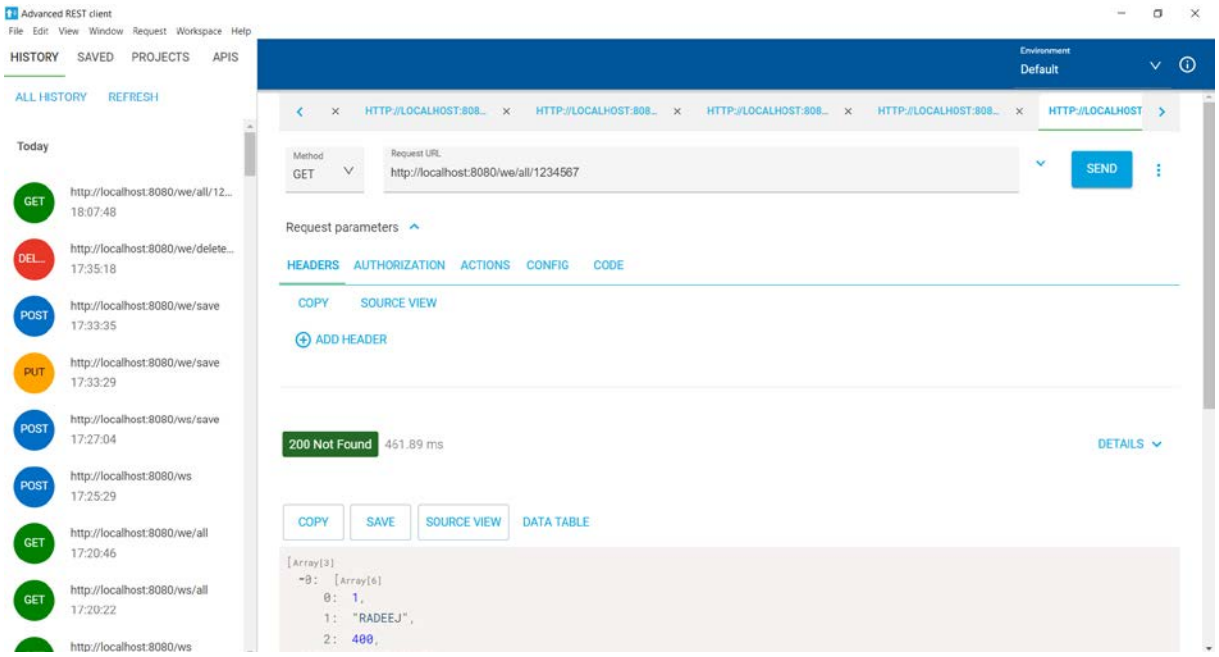
Figure 75:Get payments by contract number method

46. Run the app again

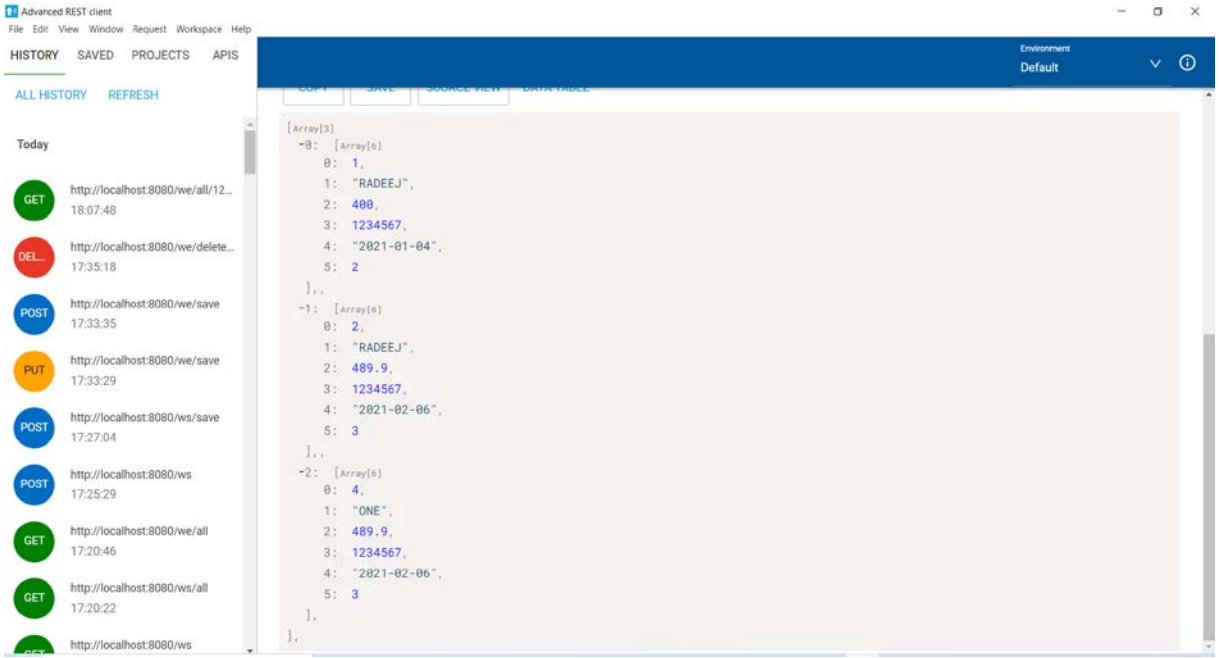
```
C:\Users\hhass\Desktop\wePayment\wePayment>mvn spring-boot:run
```

Figure 76:Run the app again

47. Test of Get payments by contract number service



48. Get payments by contract number service results of the request



4.1.3 Phone Top Up (recharge) Microservice Implementation

1. Initial creation of the application with Spring Initializr

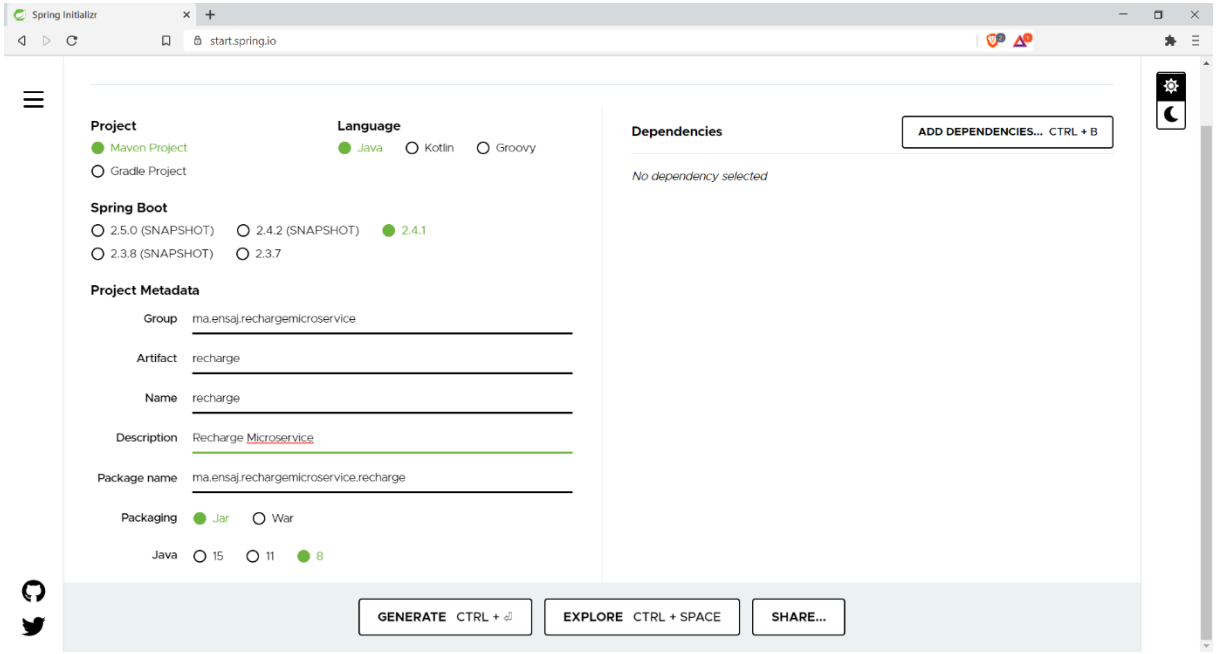


Figure 77:Spring Initializr

2. Initializr offers a quick way to extract all the dependencies you need for an application and does much of the configuration for you. This example only needs the **Web, JPA, REST Repository, Devtools and MySQL Driver dependencies**. The following image shows the Initializr configured for this example project:

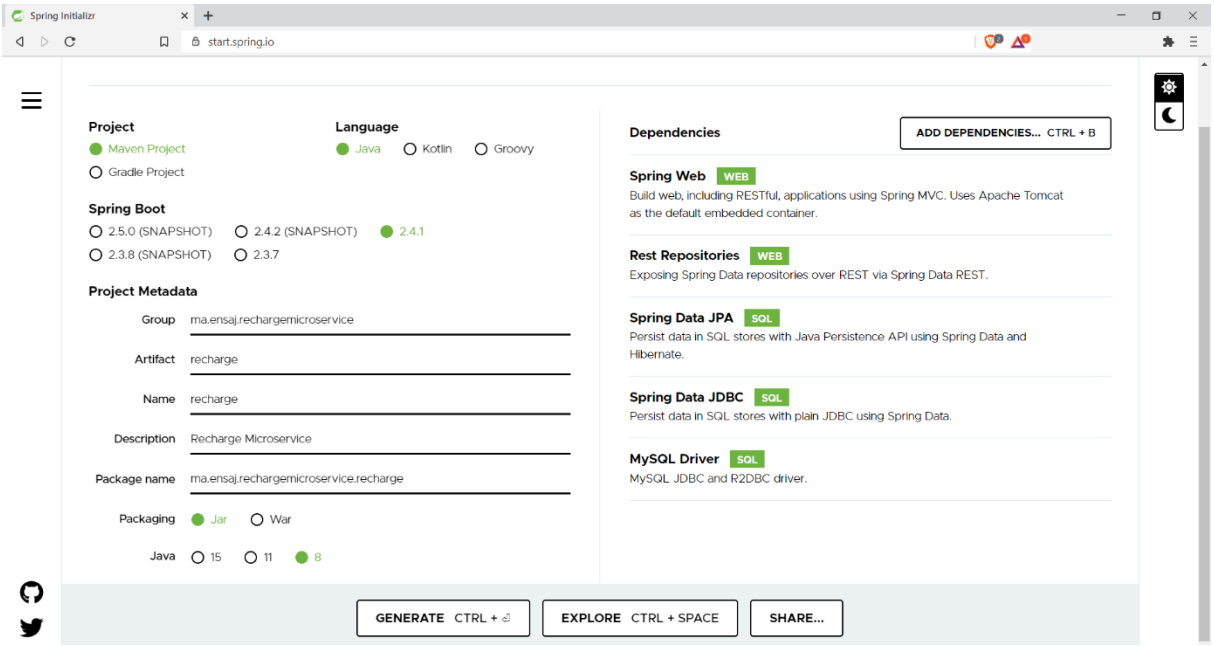


Figure 78:Initializr configured

3. Download the project and extract it

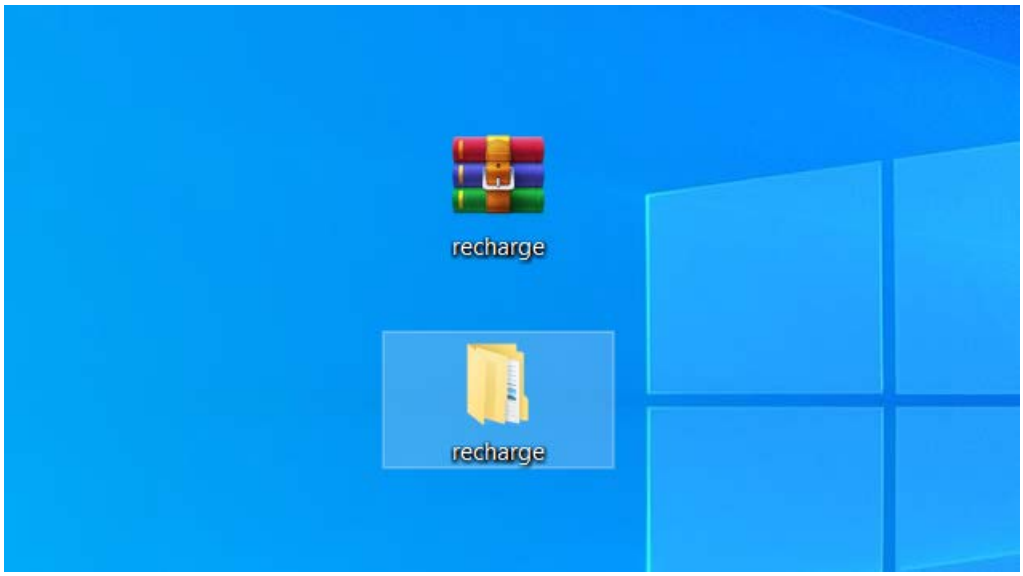


Figure 79:Download the project and extract it

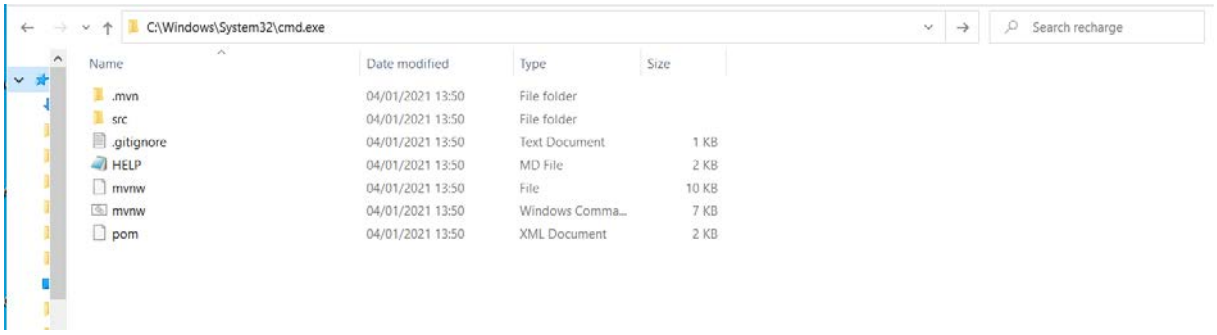


Figure 80:Project in the folder explorer

4. Run the command : **mvn eclipse :eclipse** to make the project an eclipse project

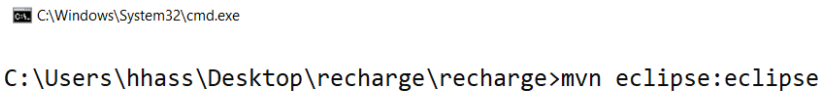


Figure 81:Run the command : mvn eclipse :eclipse to make the project an eclipse project

5. Run the command : **mvn eclipse :eclipse** to make the project an eclipse project

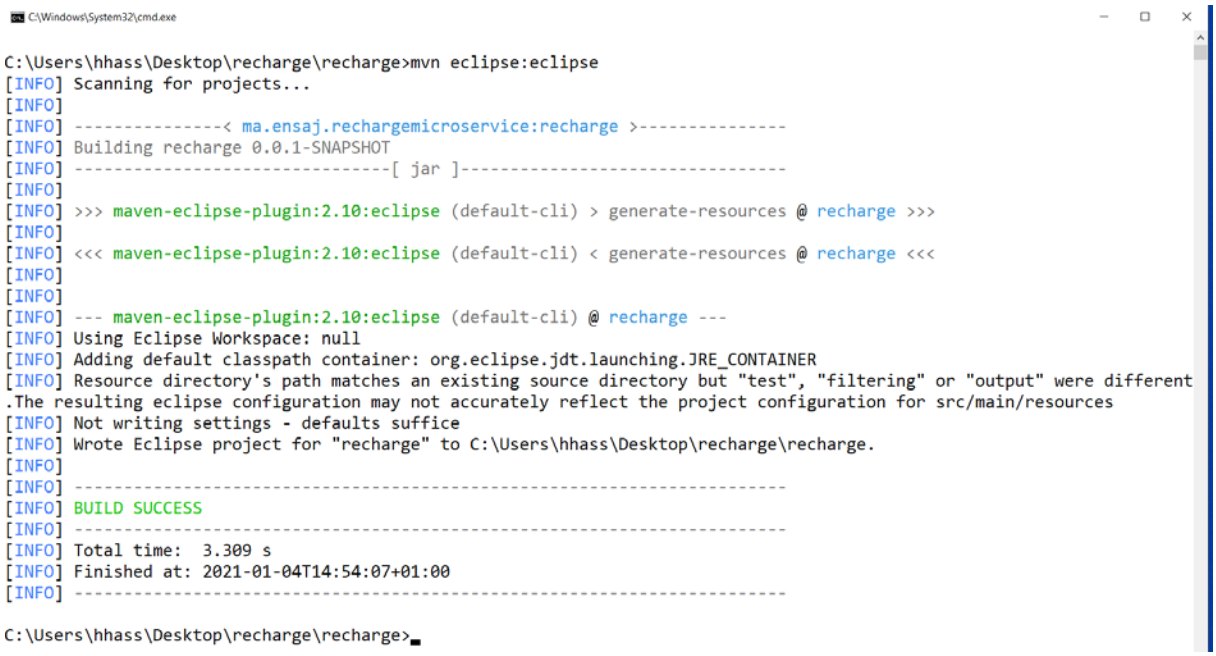


Figure 82:Build success

6. Import it under Eclipse

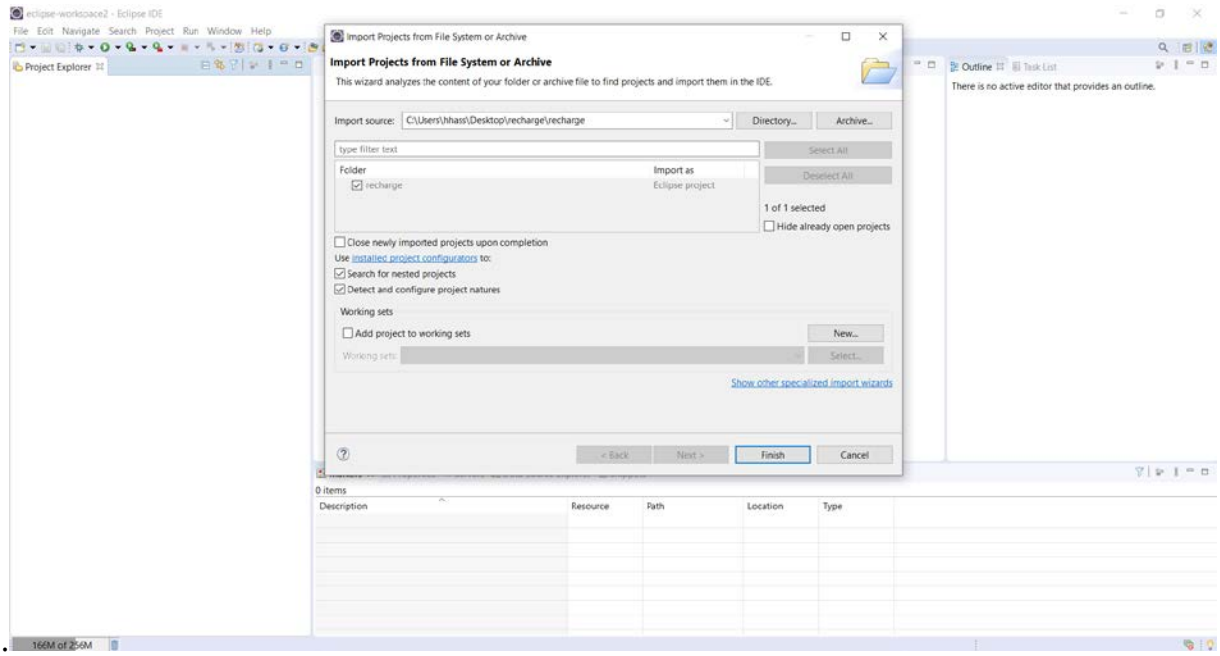


Figure 83:Import it under Eclipse

7. Project structure

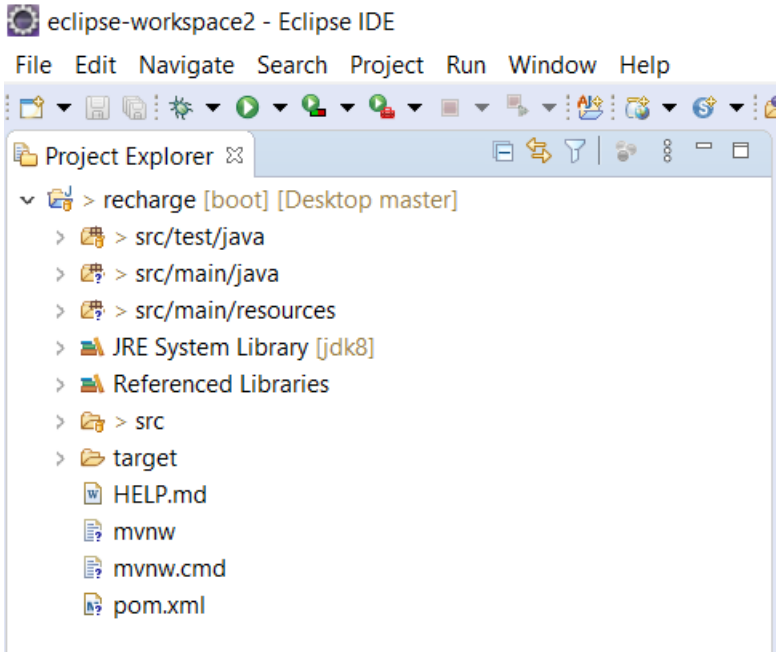


Figure 84:Project structure

8. The contents of pom.xml :

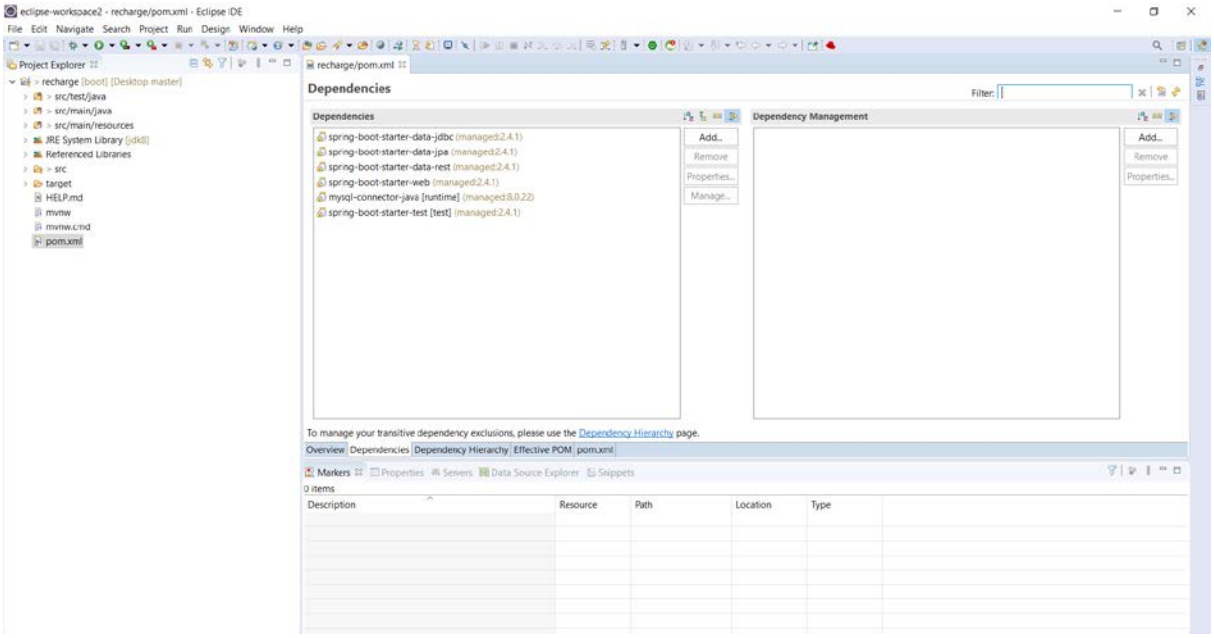


Figure 85:The contents of pom.xml

9. To indicate that we are using Java 8, it is usually necessary to specify it in several places in the pom (source version, target version). Here, you just have to set the **java.version** property:

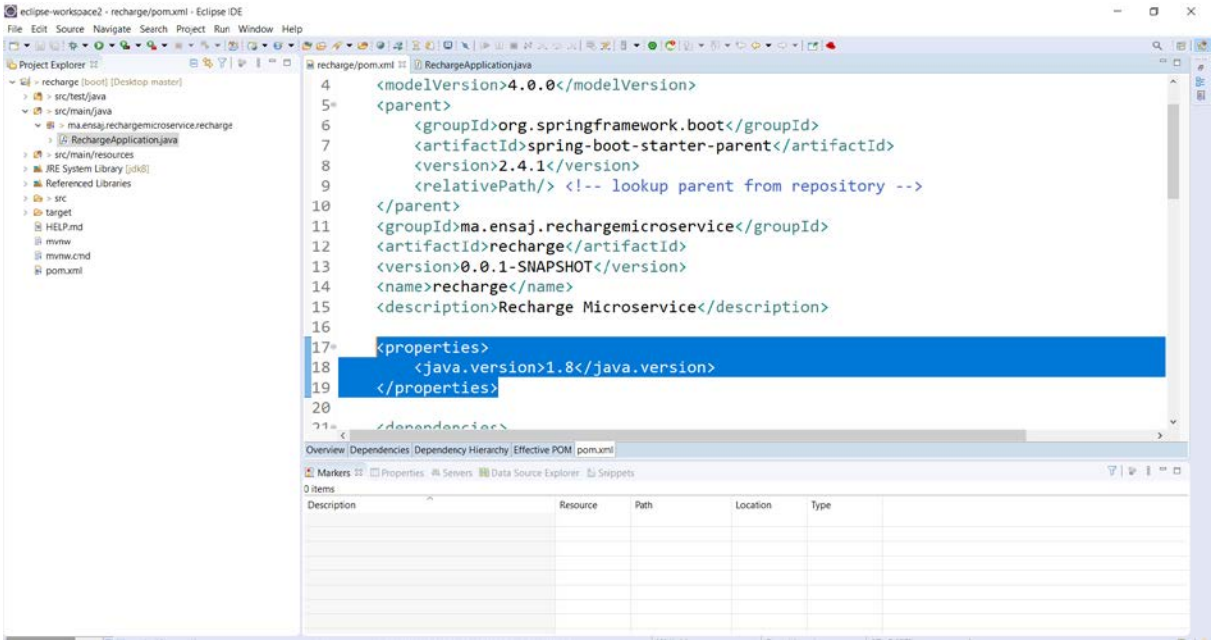


Figure 86:java.version

10. A Spring Boot application is a normal application, with main and annotations :

The **@SpringBootApplication** annotation is a meta-annotation that triggers auto-configuration and component scanning (in the classic Spring sense).

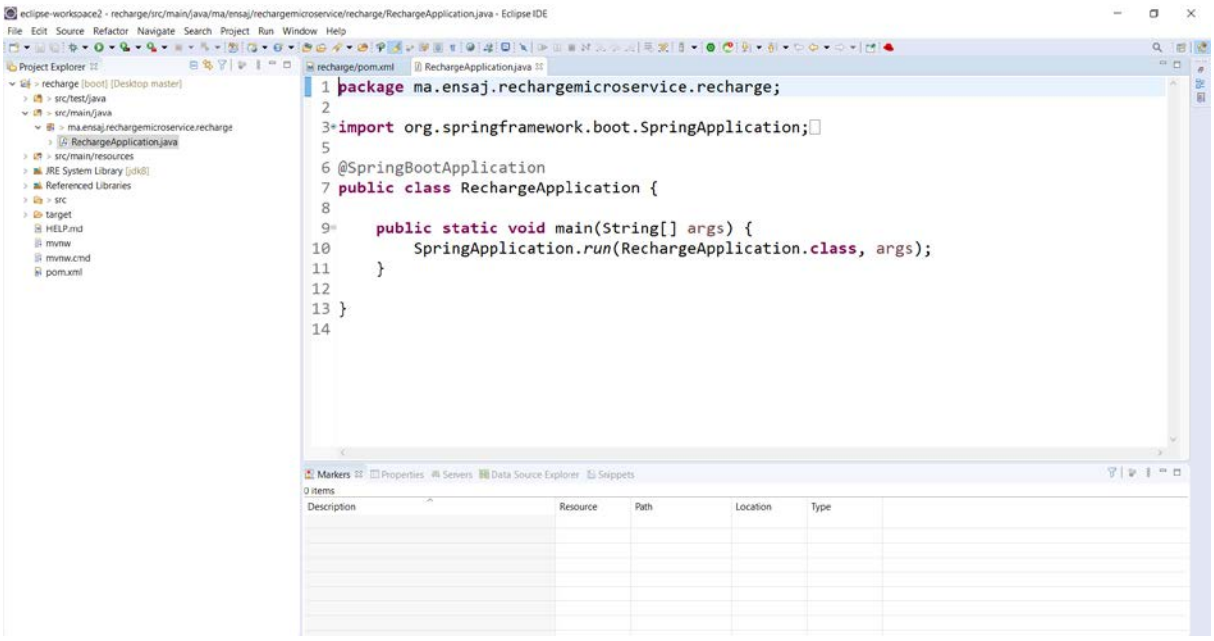


Figure 87:@SpringBootApplication

11. Configuration of the database and the Persistence Framework

Spring Boot comes with a built-in mechanism for configuring applications using a file called **application.properties.** It is located in the **src / main / resources** folder, as shown in the following figure.

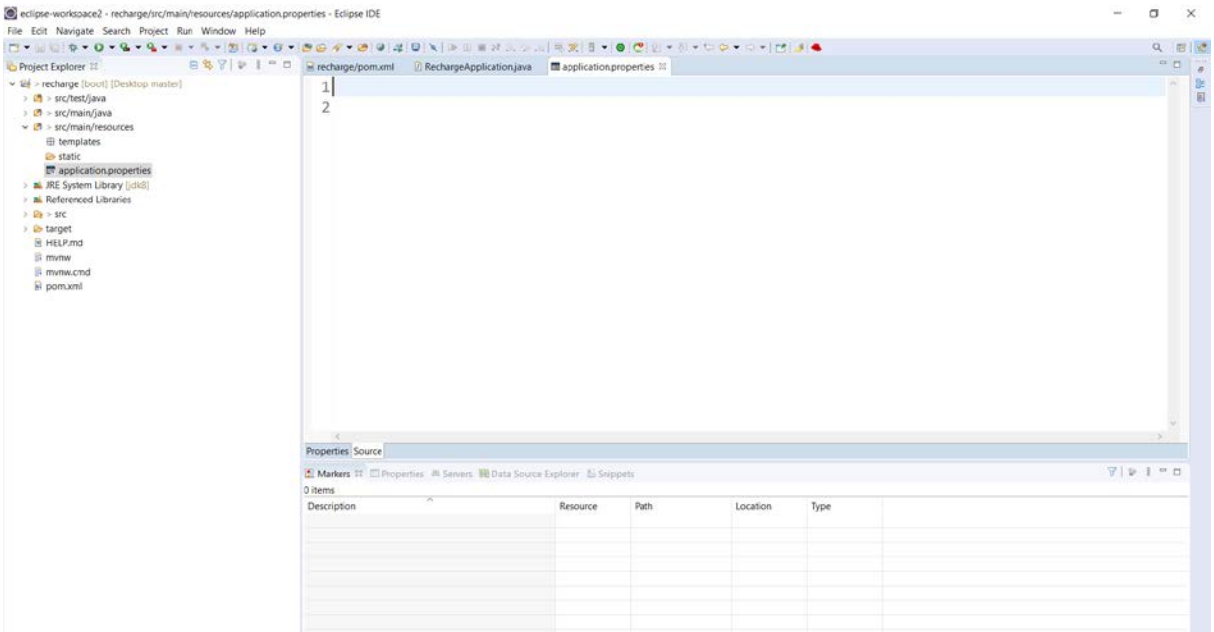


Figure 88:application.properties

12. In the **application.properties** file, configure the database connection parameters and some parameters related to the **Persistence** Framework

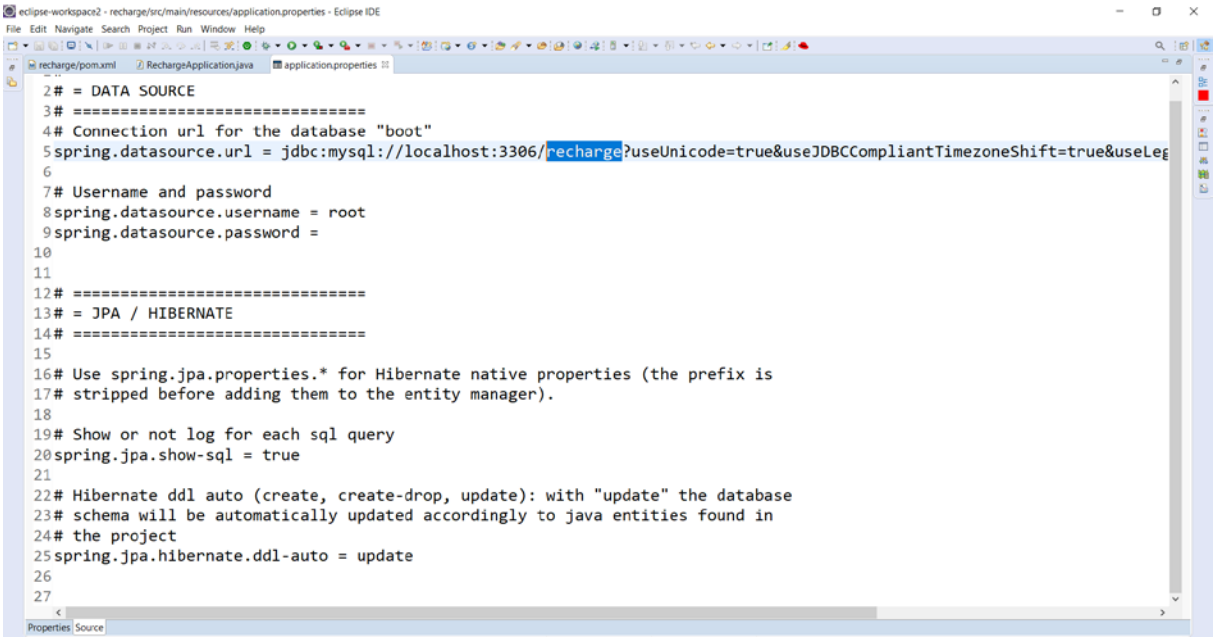


Figure 89:configure the database connection parameters and some parameters related to the Persistence Framework

13. Create the "recharge" database under MySQL

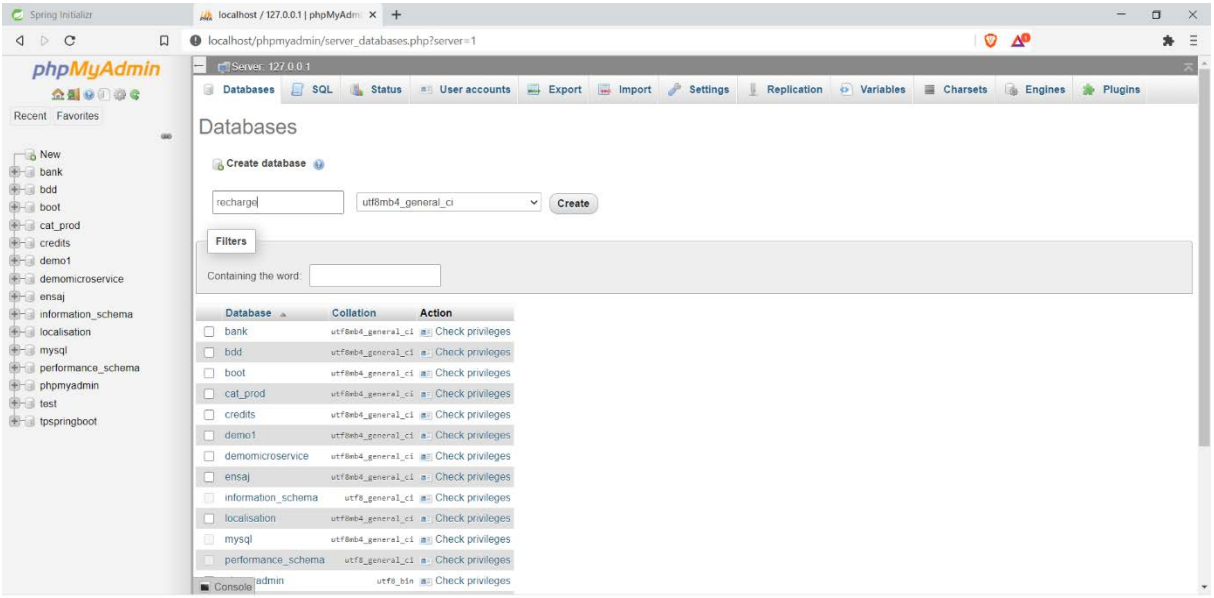


Figure 90:"recharge" database under MySQL

14. Creation of business classes

15. First create the package **beans** under the main package

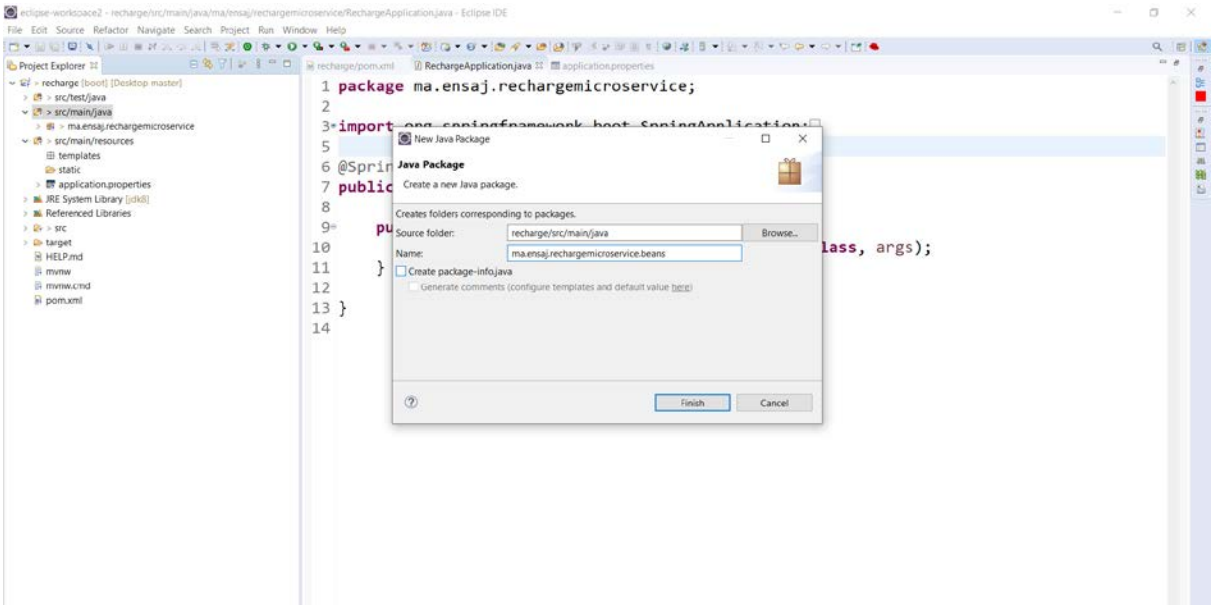


Figure 91:create the package beans under the main package

16. Create the "**Recharge**" class in a sub-package of the main package (here: ma.ensaj.rechargemicroservice.beans):

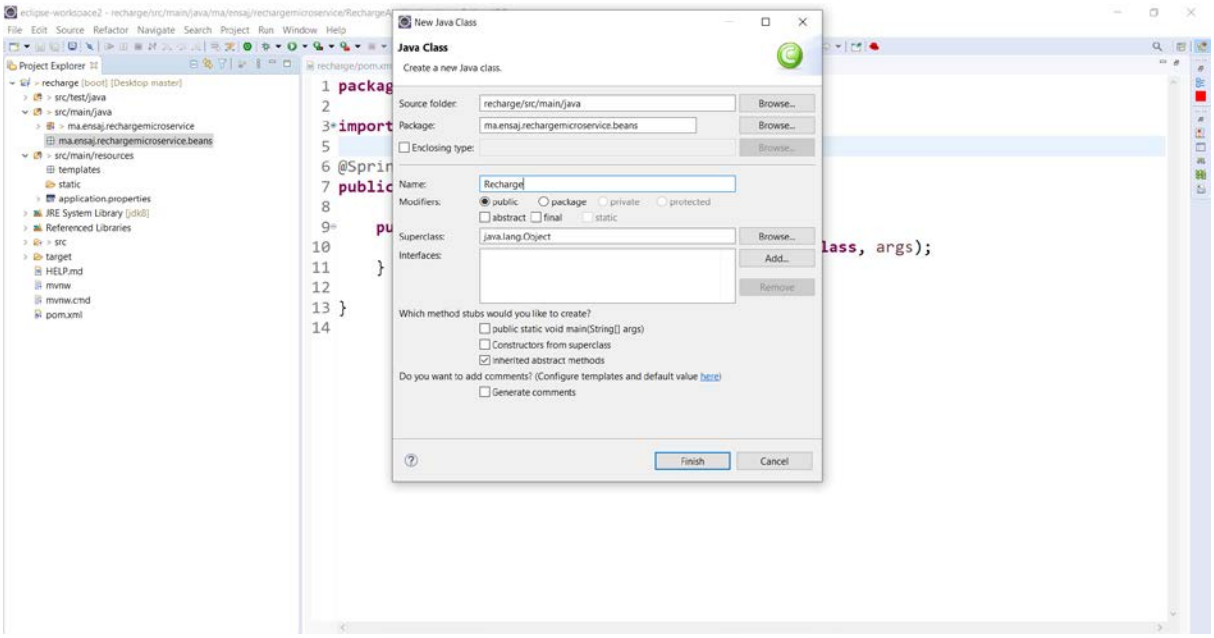


Figure 92:Create the "Recharge" class

17. Class diagram of Recharge entity

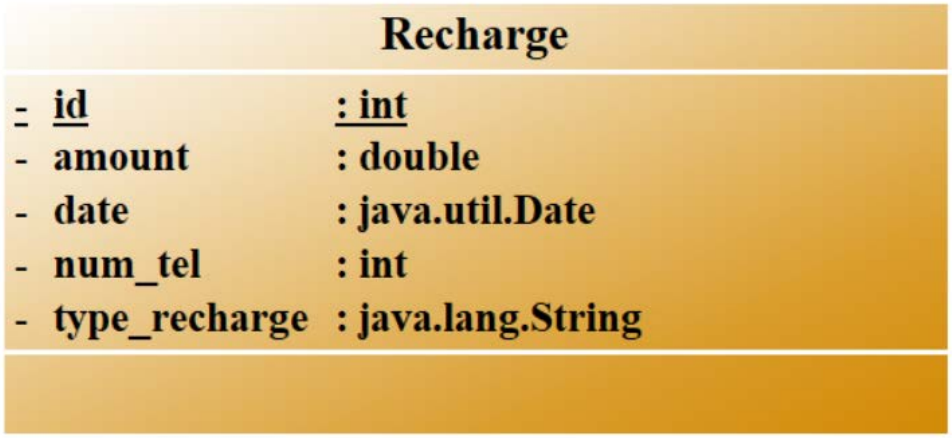


Figure 93:Class diagram of Recharge entity

18. Implementation of Recharge class

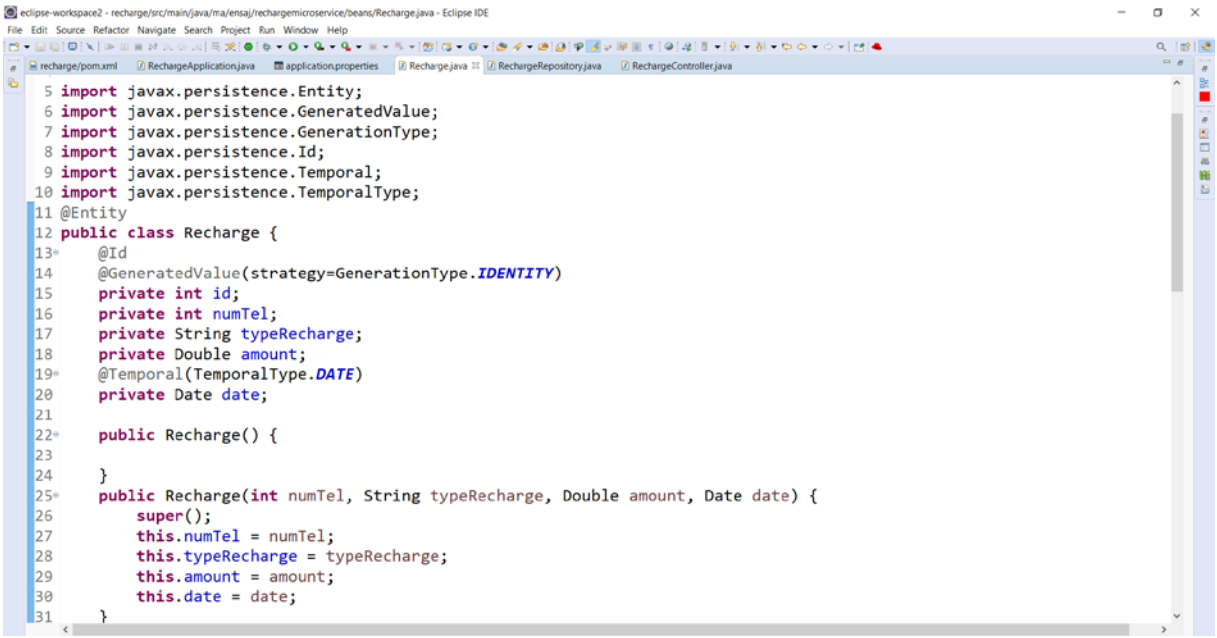


Figure 94:Implementation of Recharge class

19. To start the application, use **mvn spring-boot:run** command

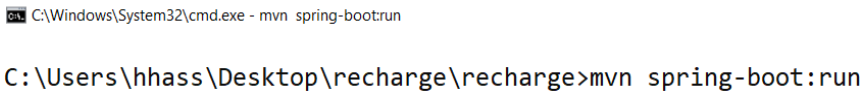


Figure 95:To start the application, use mvn spring-boot:run command

20. We can see now in the cmd **Started RechargeApplication**

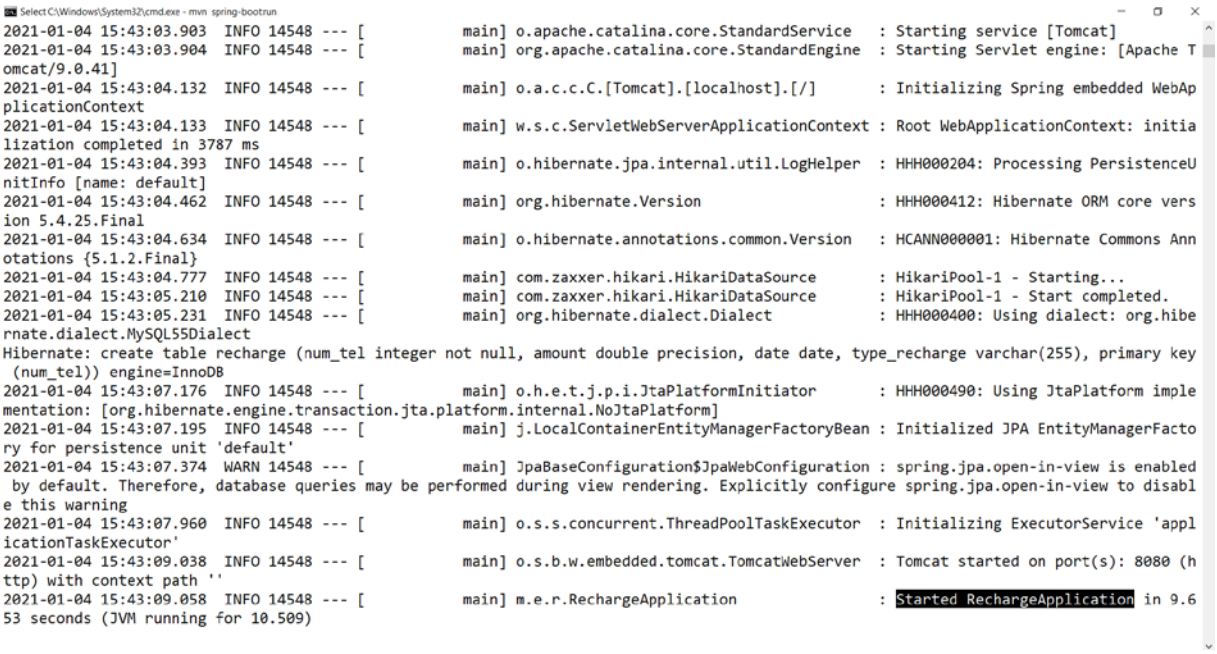


Figure 96:We can see now in the cmd Started RechargeApplication

21. Check if the "recharge" table is created.

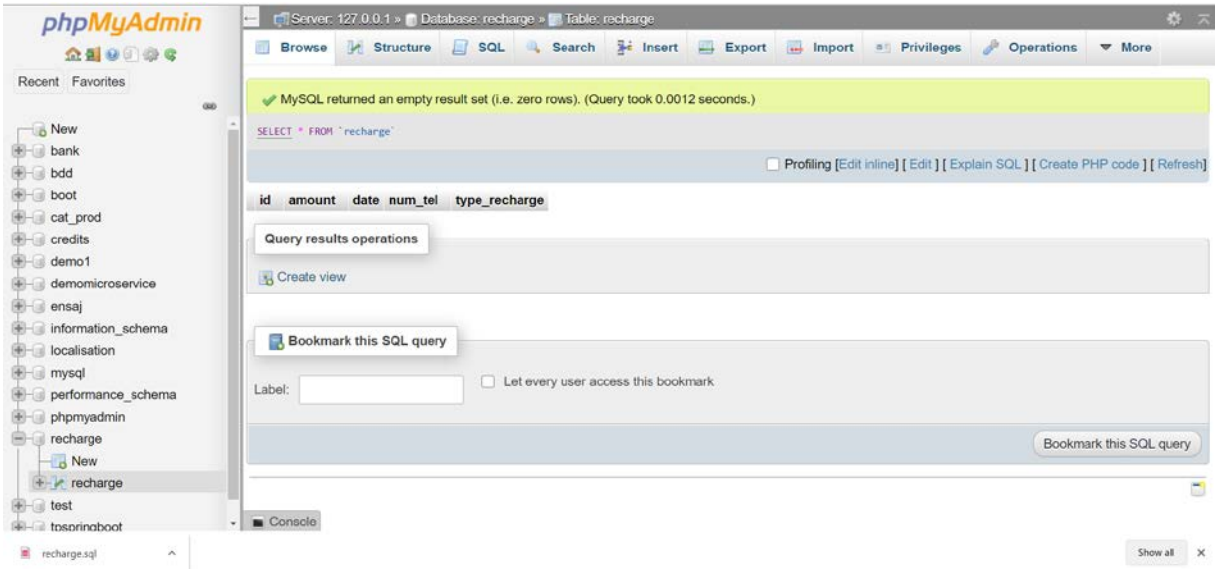


Figure 97:Check if the "recharge" table is created.

22. Insert a new row to the table recharge

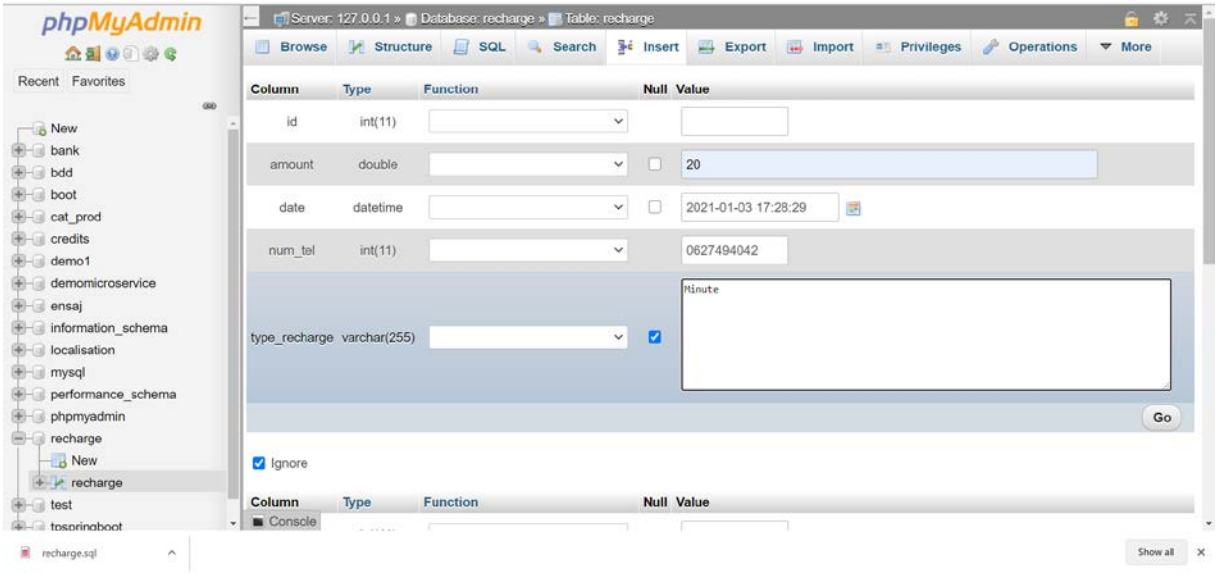


Figure 98:Insert a new row to the table recharge

23. Creation of repositories

Spring Data JPA provides a data access layer implementation for a Spring application.

It is a very convenient brick because it allows not reinventing the data access wheel with each new application, and thus allowing you to focus on the business side.

24. Create the "**RechargeRepository**" interface in a sub-package -called repository- of the main package :

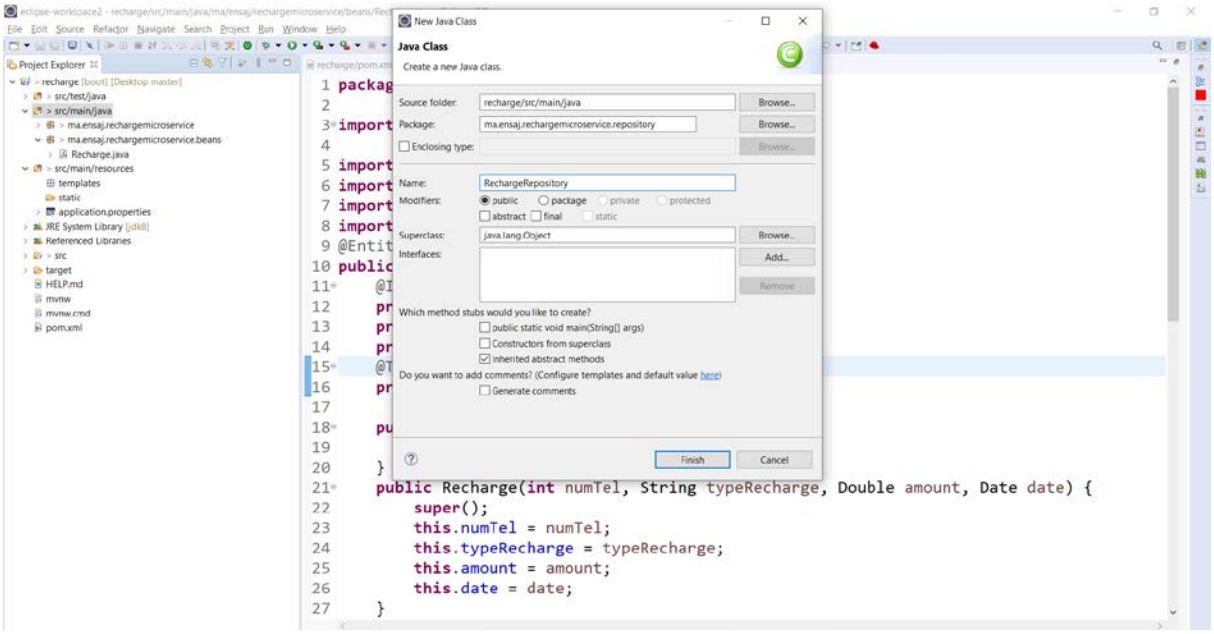


Figure 99:Create the "RechargeRepository" interface

25. **RechargeRepository** extends **JpaRepository<Recharge,Integer>**

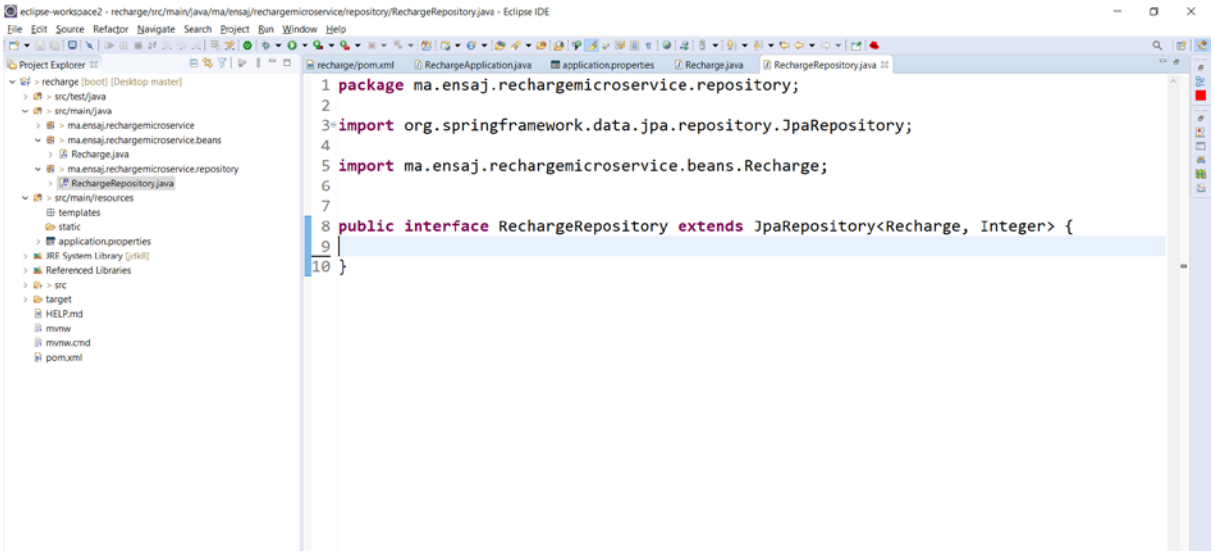


Figure 100: "RechargeRepository"

26. In the interface add the method :

Recharge findById(int id);

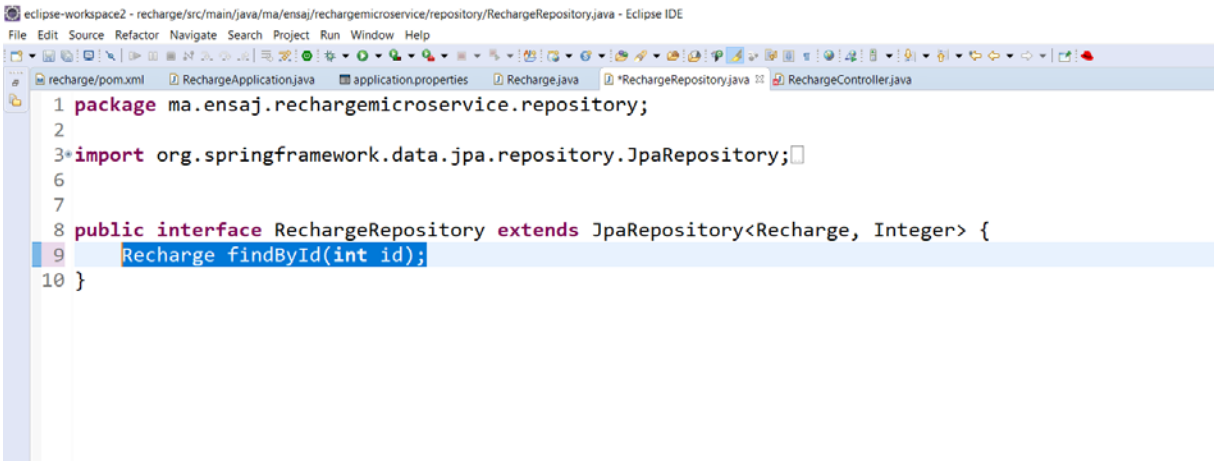


Figure 101:In the interface add the method findById()

27. Creation of REST controllers

REST stands for **REpresentational State Transfer**. It is developed by Roy Thomas Fielding, who also developed HTTP. The main goal of RESTful Web Services is to make Web services more efficient. RESTful Web Services attempt to define services using the various concepts already present in HTTP. REST is an architectural approach, not a protocol.

It does not define the standard message exchange format. We can create REST services with XML and JSON. JSON is a more popular format with REST. Key abstraction is a resource in REST. A resource can be anything. It is accessible via a Uniform Resource Identifier (URI).

28. Create the "**RechargeController**" class in a sub-package-called controller- of the main package:

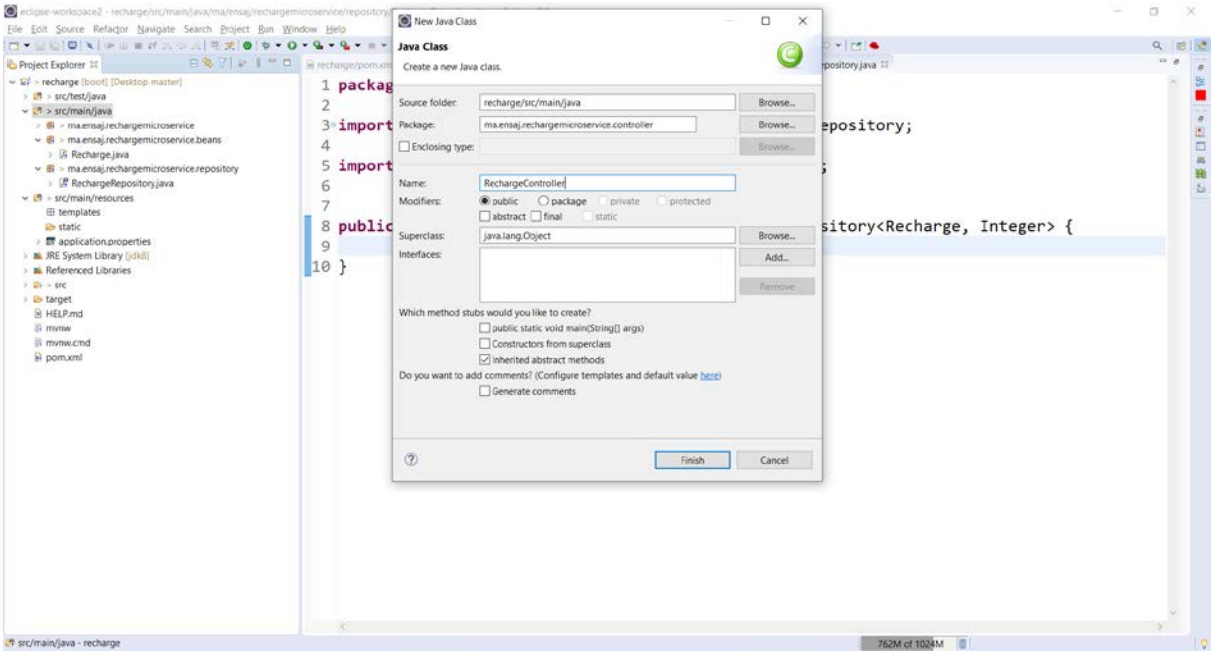


Figure 102:Create the "RechargeController" class

29. RechargeController class

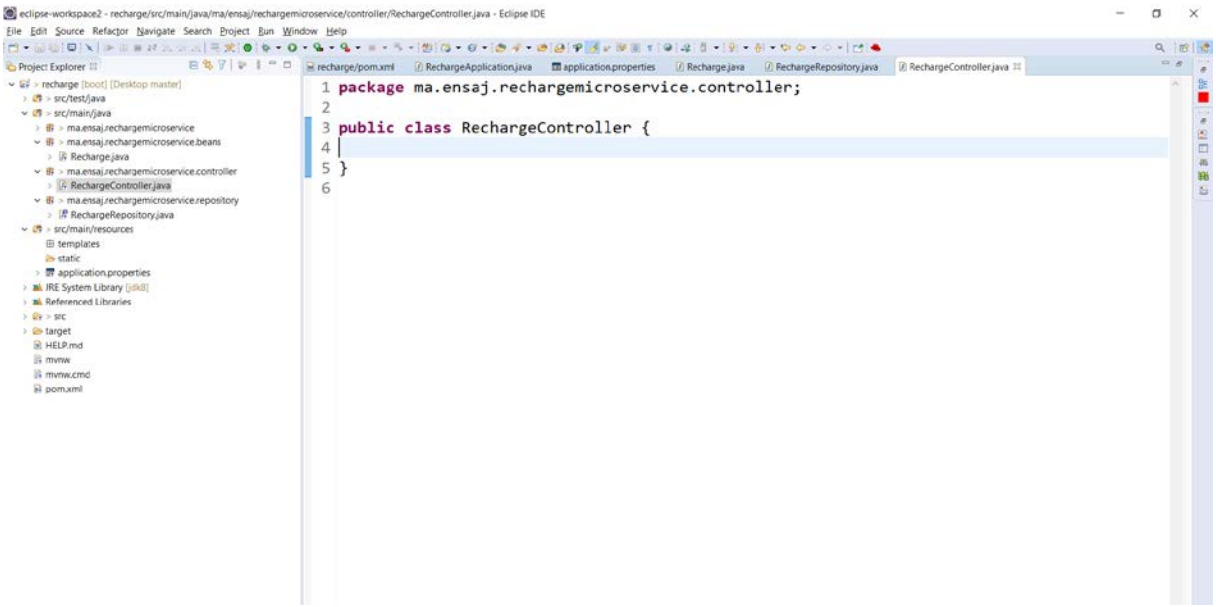


Figure 103:RechargeController class

30. Add annotations **@RestController** and **@RequestMapping(“recharges”)**

N.B:

- The **@RestController** annotation was introduced in Spring 4.0 to simplify the creation of RESTful web services. It's a convenience annotation that combines @Controller and @ResponseBody – which eliminates the need to annotate every request handling method of the controller class with the @ResponseBody annotation.
- **@RequestMapping** is used to map web requests to Spring Controller methods.

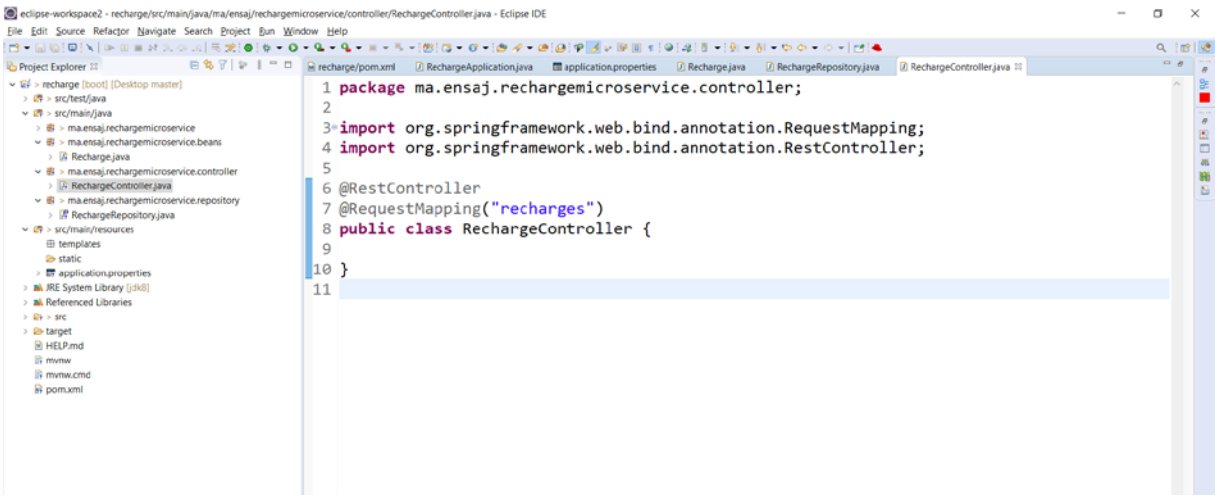


Figure 104:@RestController and @RequestMapping(“recharges”)

31. Add **CRUD** web services to the **RechargeController**

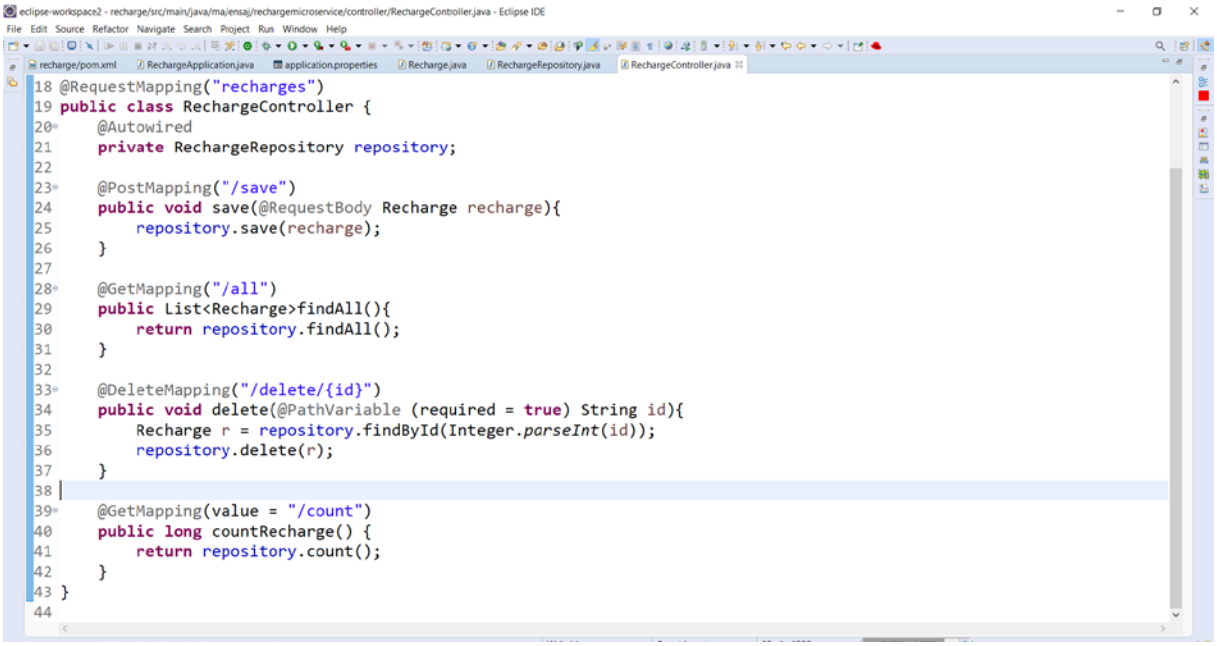


Figure 105:Add CRUD web services to the RechargeController

32. Testing controllers with ARC(Advanced Rest Client)

33. List of phone top up(recharge)

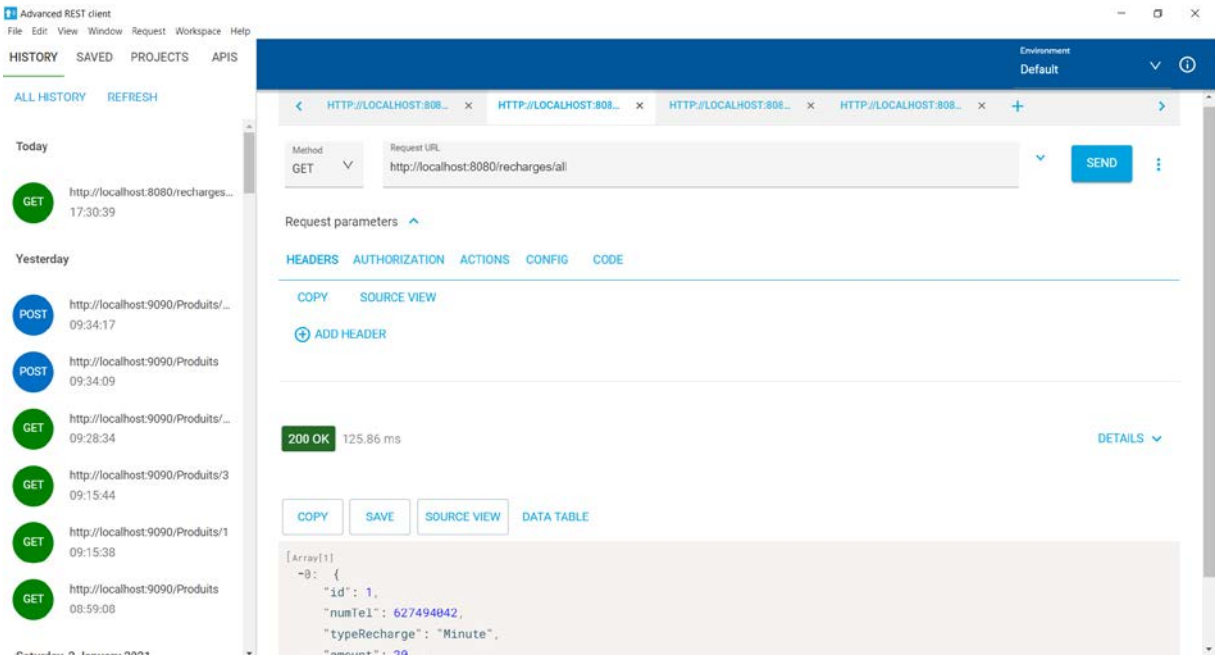


Figure 106:List of phone top up(recharge)

34. Creation(Add some recharge)

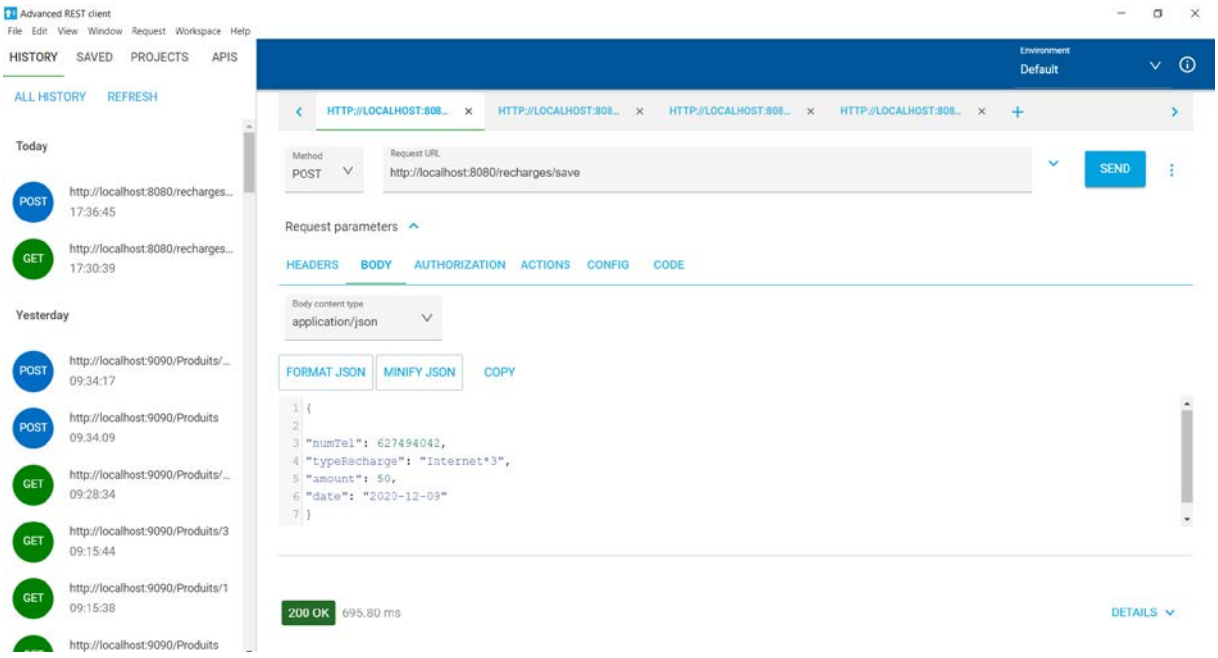


Figure 107:Creation(Add some recharge)

35. Check if recharges were added successfully to the database

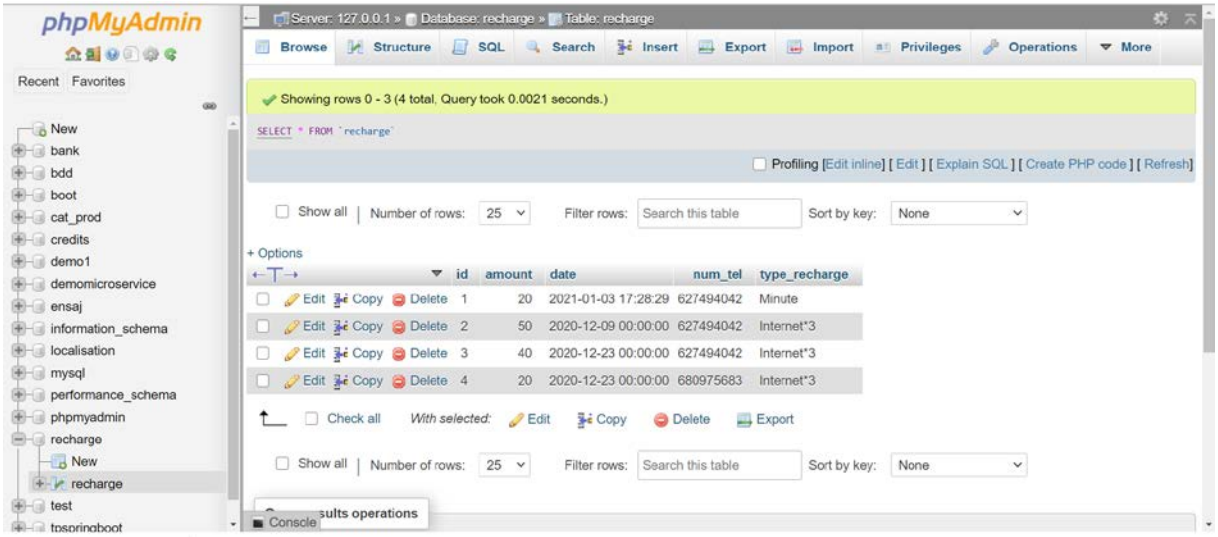


Figure 108:Check if recharges were added successfully to the database

36. Update the recharge with id 4

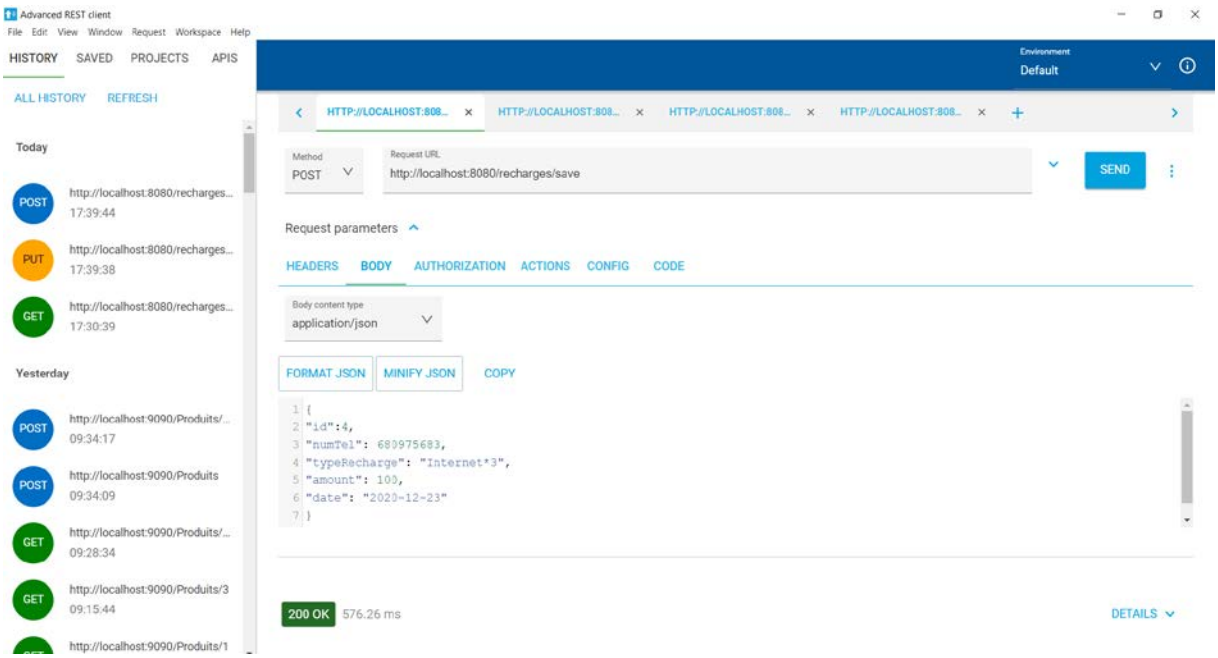


Figure 109:Update the recharge with id 4

37. The update is well achieved

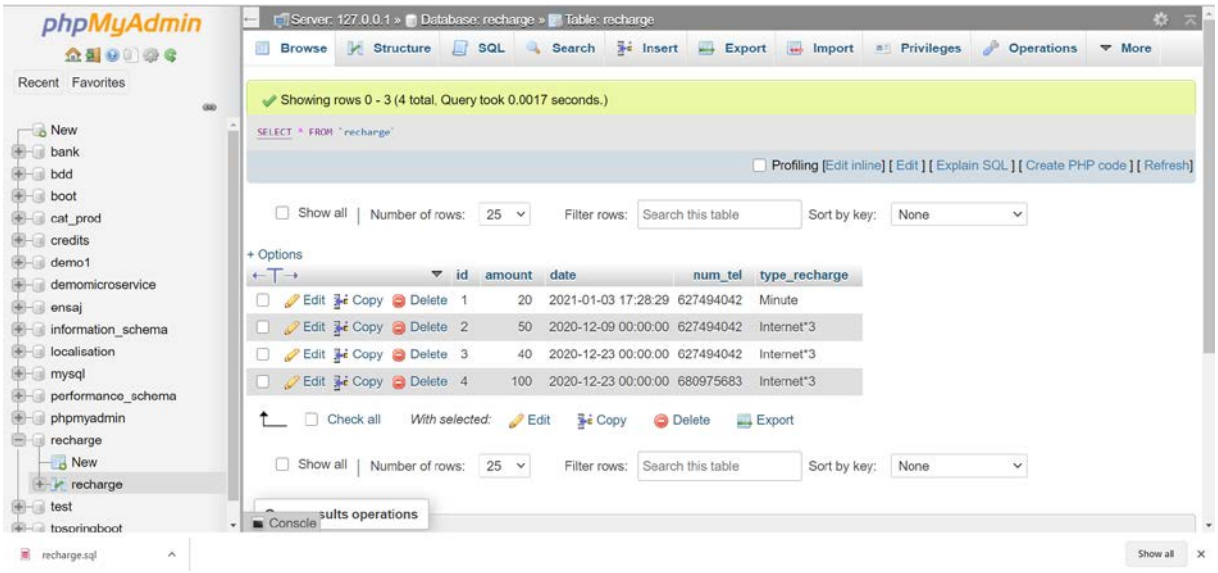


Figure 110:The update is well achieved

38. Count test

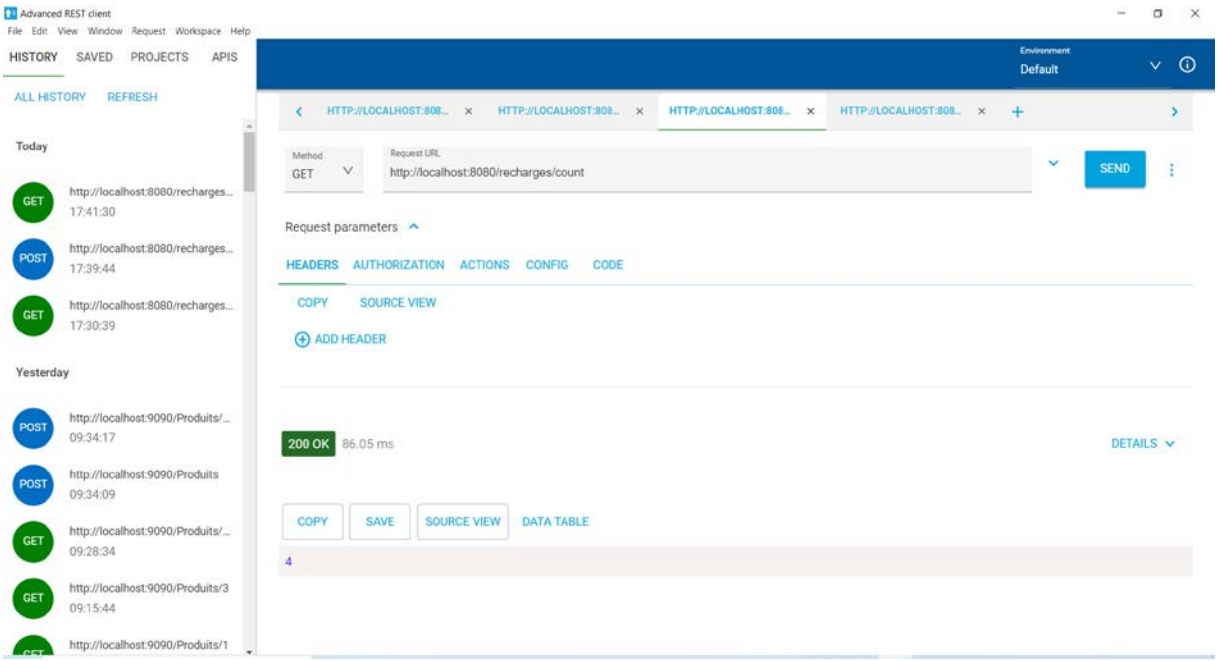


Figure 111:Count test

39. Delete recharge with id 3

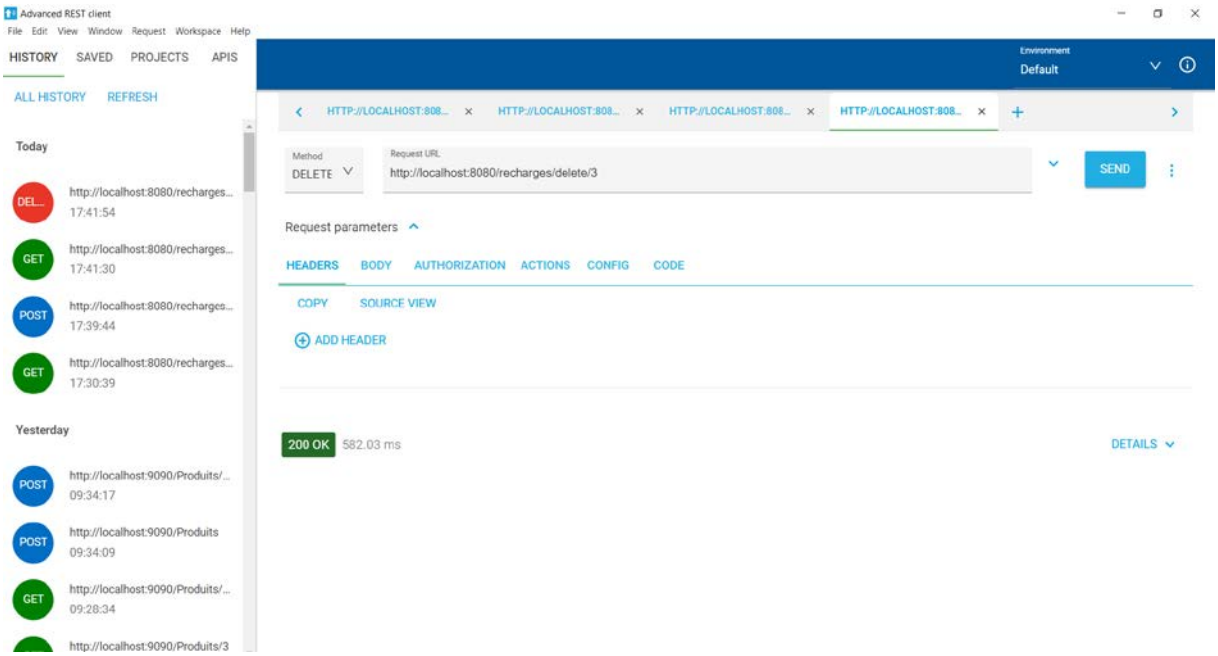


Figure 112:Delete recharge with id 3

40. The recharge with id 3 was deleted successfully

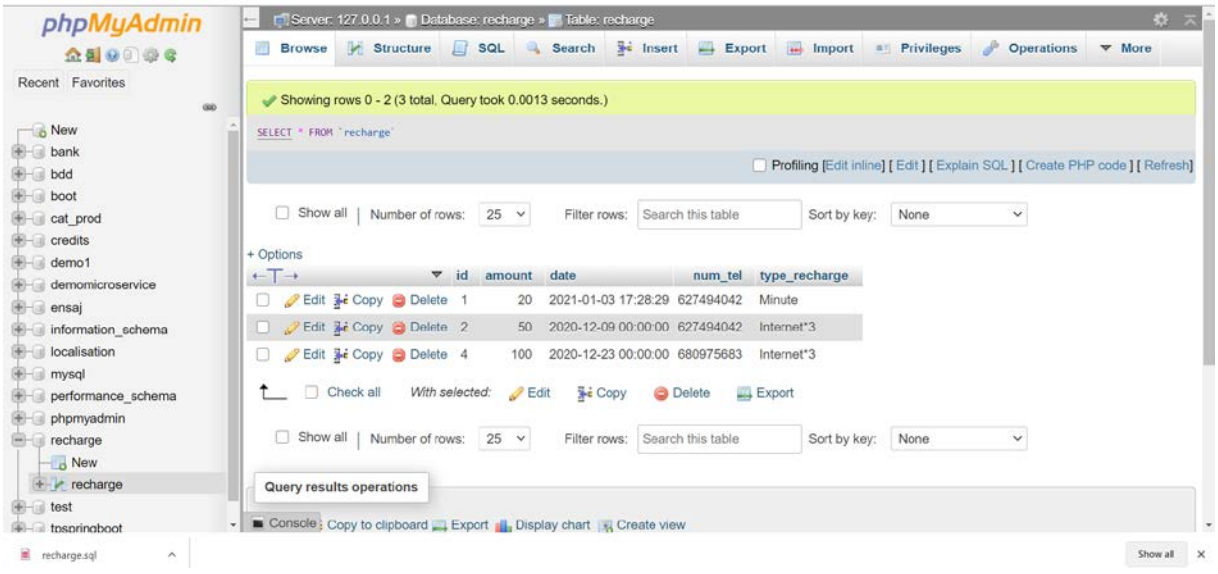


Figure 113:The recharge with id 3 was deleted successfully

41. Get recharge by telephone number

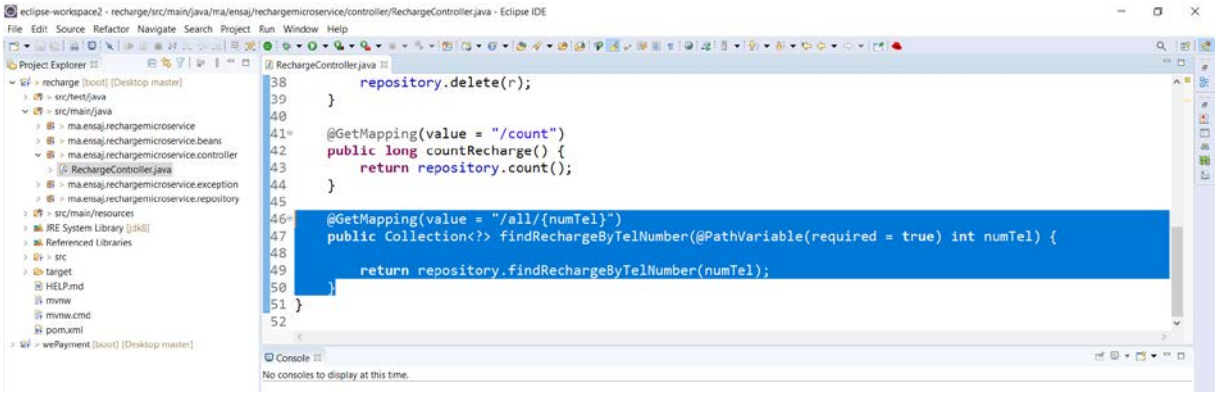


Figure 114:Get recharge by telephone number

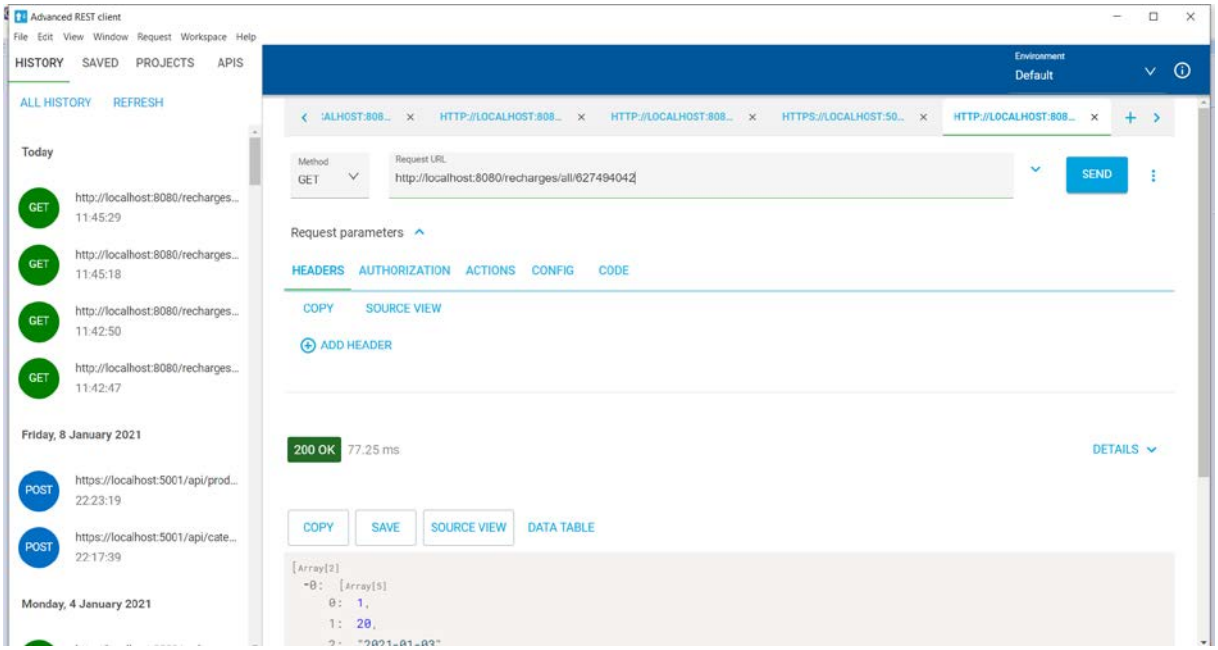


Figure 115:Get recharge by telephone number test

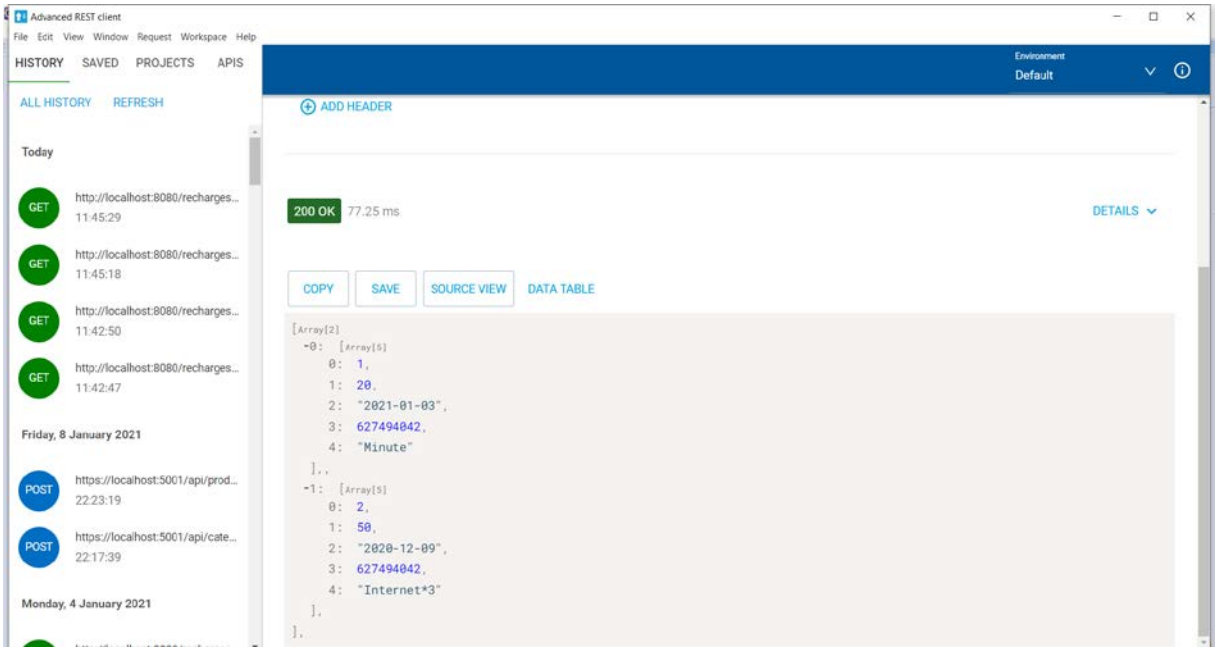


Figure 116:Get recharge by telephone number result's test

4.1.4 Registration/Login Microservice Implementation

Creation and set up of the Spring Boot project

1. Initial creation of the application with **Spring Initializr**

Initializr offers a quick way to extract all the dependencies you need for an application and does much of the configuration for you. This example only needs the **Web, JPA, Devtools, MySQL Driver, Spring Security and Thymeleaf** dependencies. The following image shows the Initializr configured for this example project:

N.B: **Thymeleaf** is a modern server-side Java template engine for both web and standalone environments. Thymeleaf's main goal is to bring elegant natural templates to your development workflow — HTML that can be correctly displayed in browsers and also work as static prototypes, allowing for stronger collaboration in development teams.

With modules for Spring Framework, a host of integrations with your favourite tools, and the ability to plug in your own functionality, Thymeleaf is ideal for modern-day HTML5 JVM web development.[https://www.thymeleaf.org/]

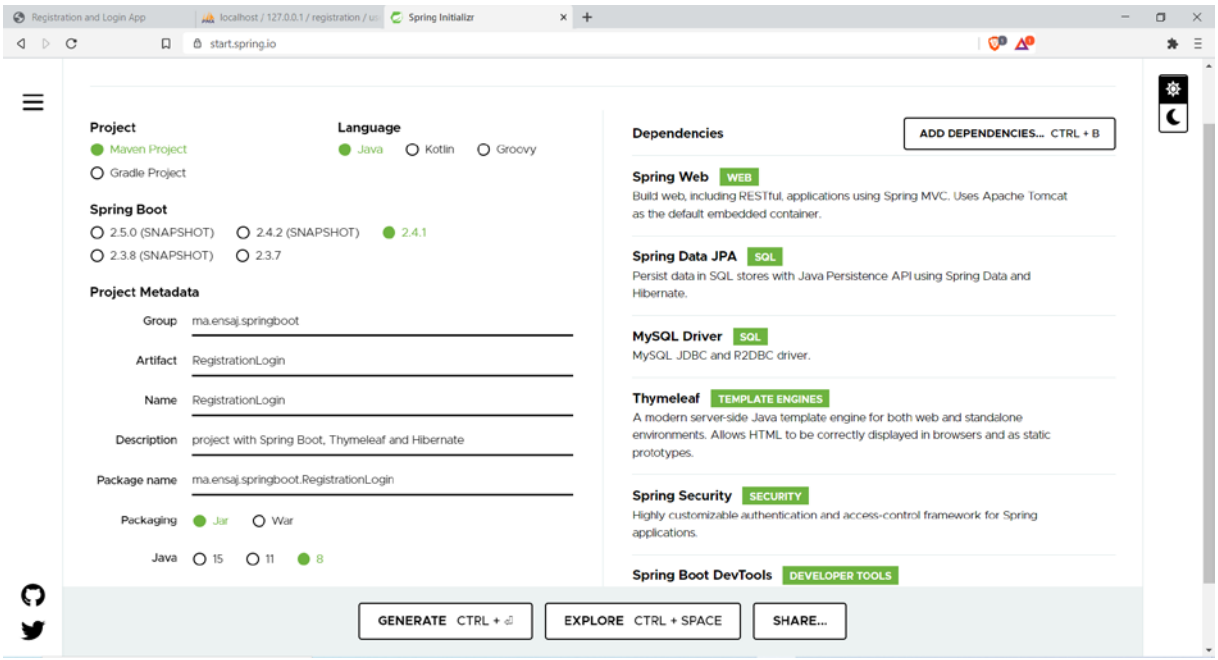


Figure 117:Spring Initializr

2. Download the project and extract it

Figure 118:"Download the project and extract it

3. Run the command : **mvn eclipse :eclipse** to make the project an eclipse project

Figure 119:Run the command : mvn eclipse :eclipse to make the project an eclipse project

4. Import it under Eclipse.

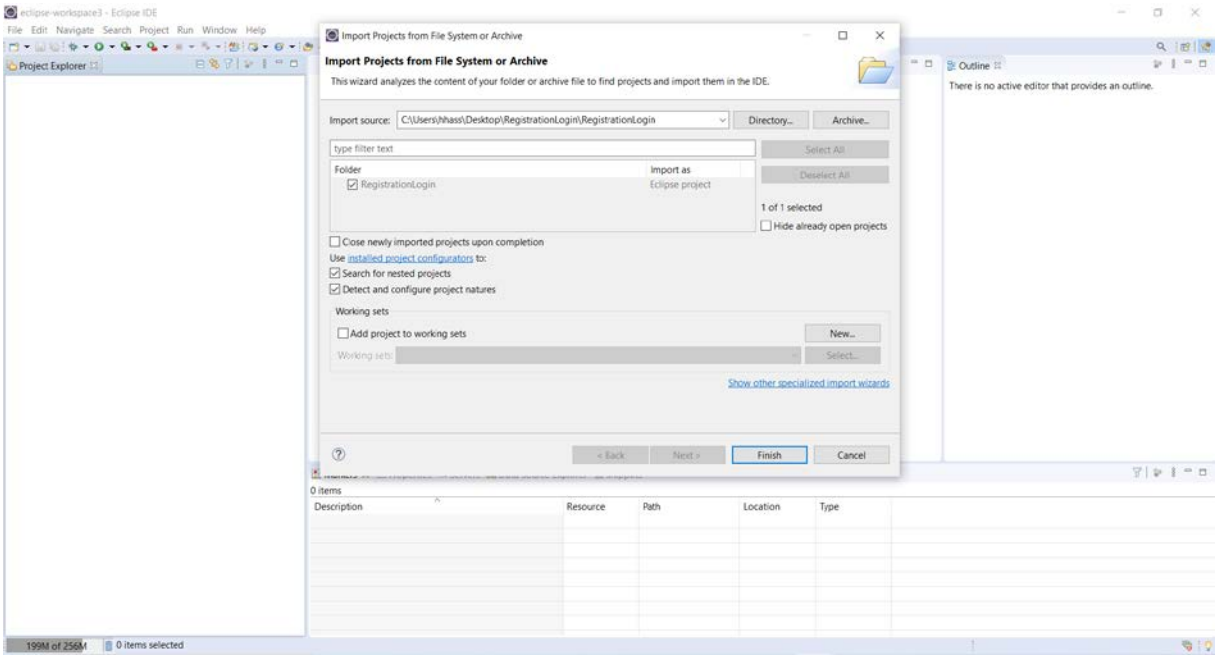


Figure 120:Import it under Eclipse.

5. A Spring Boot application is a normal application, with main and annotations :

The **@SpringBootApplication** annotation is a meta-annotation that triggers auto-configuration and component scanning (in the classic Spring sense).

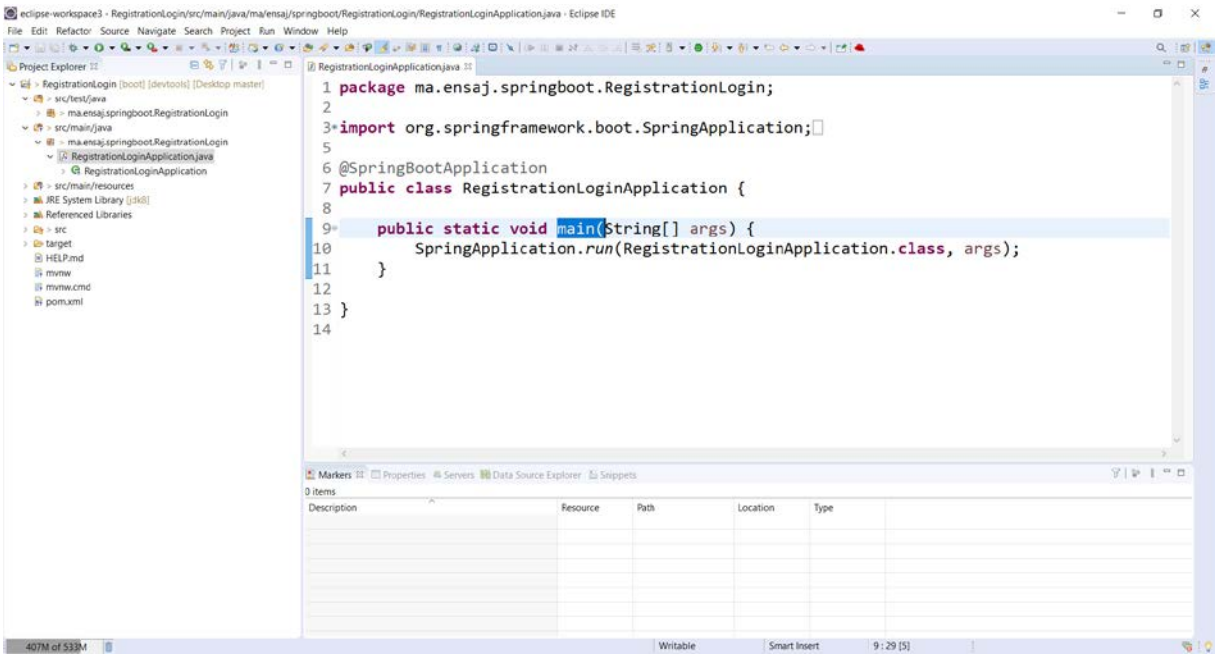


Figure 121:@SpringBootApplication

6. The content of pom.xml :

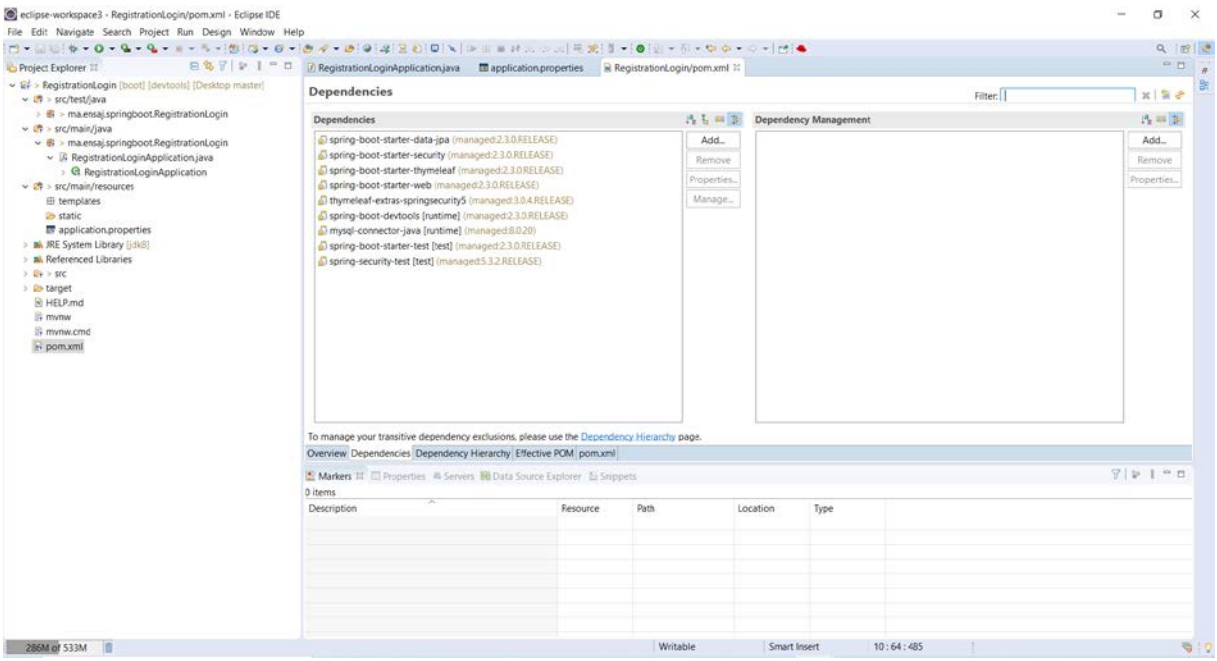


Figure 122:The content of pom.xml

7. To indicate that we are using Java 8, it is usually necessary to specify it in several places in

the pom (source version, target version). Here, you just have to set the **java.version** property:

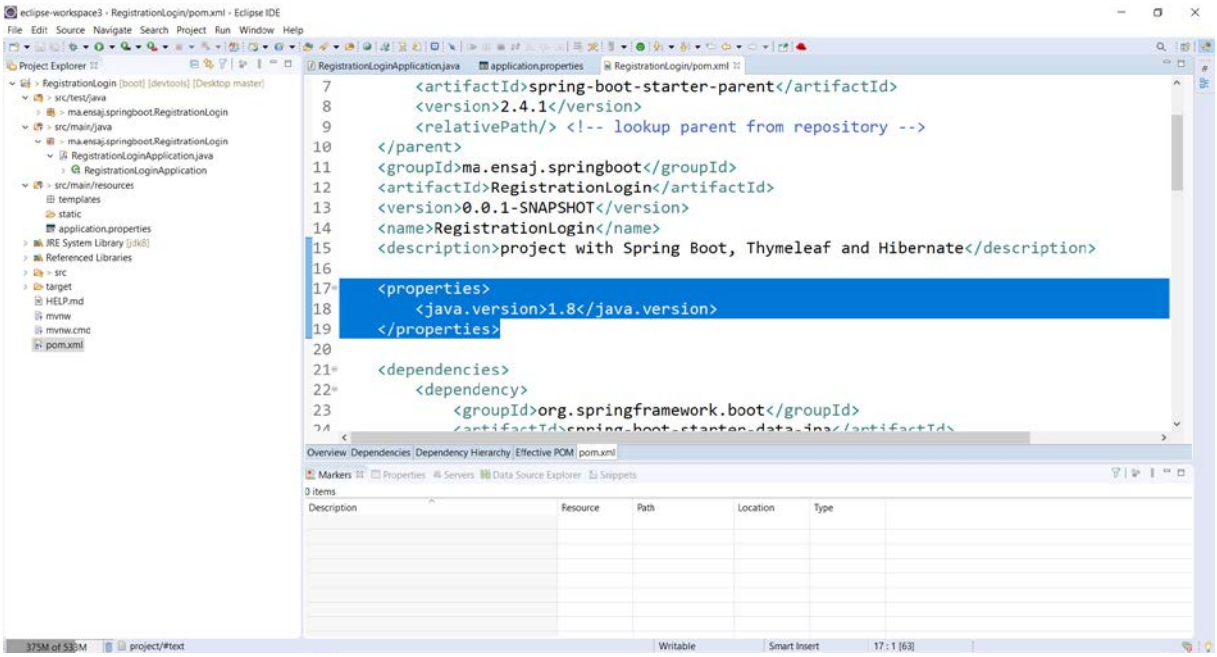


Figure 123:java.version property

Database setup

8. Configuration of the database and the Persistence Framework

Spring Boot comes with a built-in mechanism for configuring applications using a file called **application.properties.** It is located in the **src / main / resources** folder, as shown in the following figure.

9. In the **application.properties** file, configure the database connection parameters and some parameters related to the **Persistence** Framework :

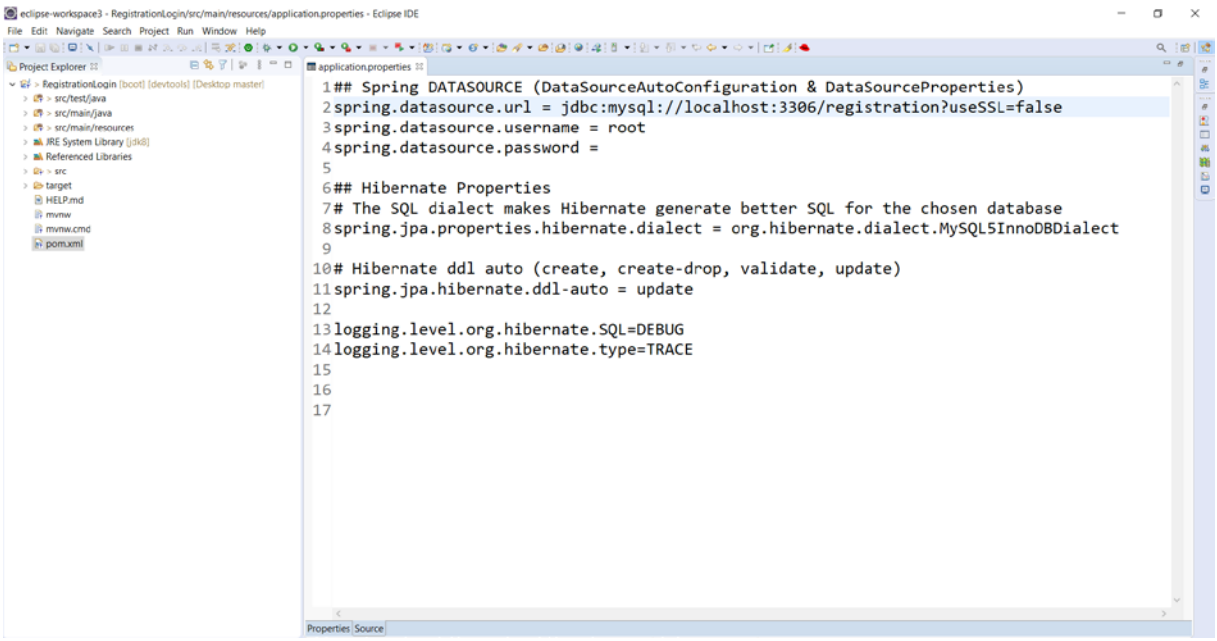


Figure 124:configure the database connection parameters and some parameters related to the Persistence Framework

10. Create the "**register**" database under MySQL.

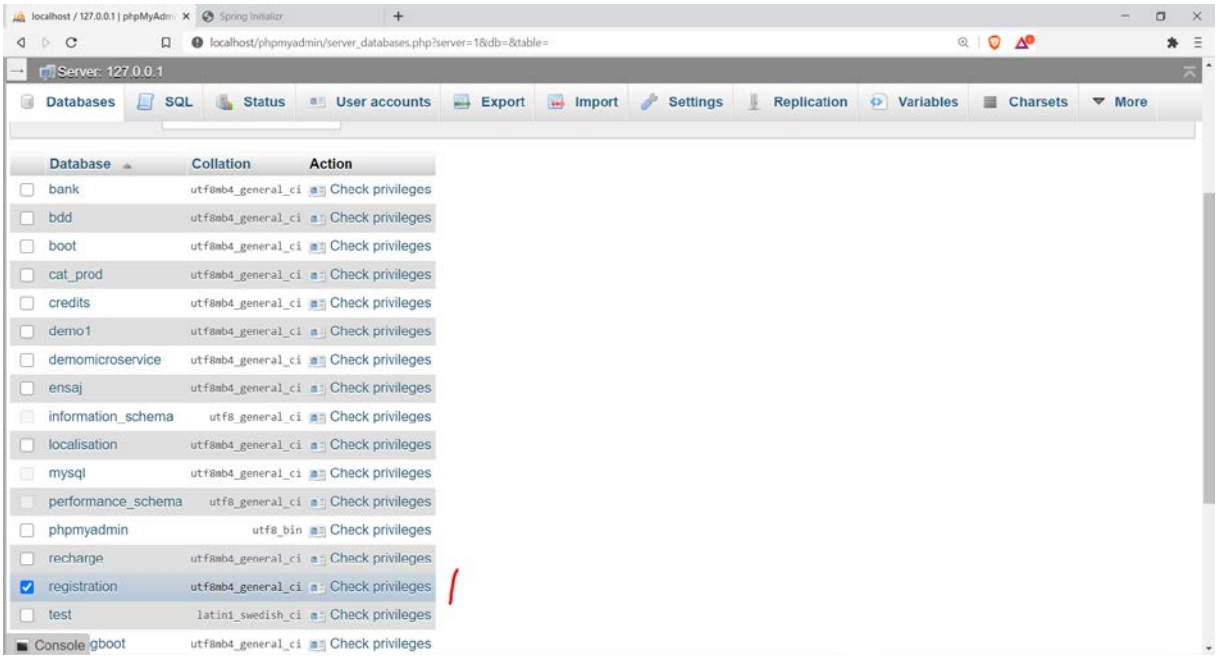


Figure 125:Create the "register" database under MySQL.

Creation of JPA entities: User and Role (Many to Many mapping)

11. Create the "**Role**" class in a sub-package of the main package (here: ma.ensaj.springboot.model):

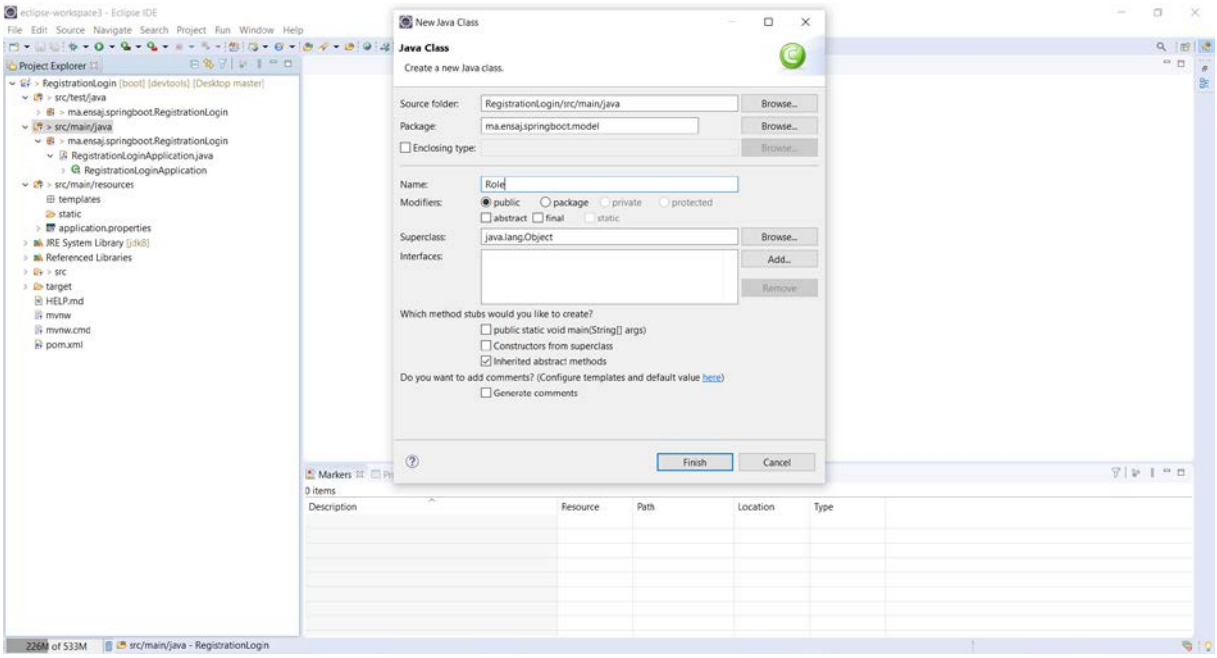


Figure 126:Create the "Role" class

12. Role class

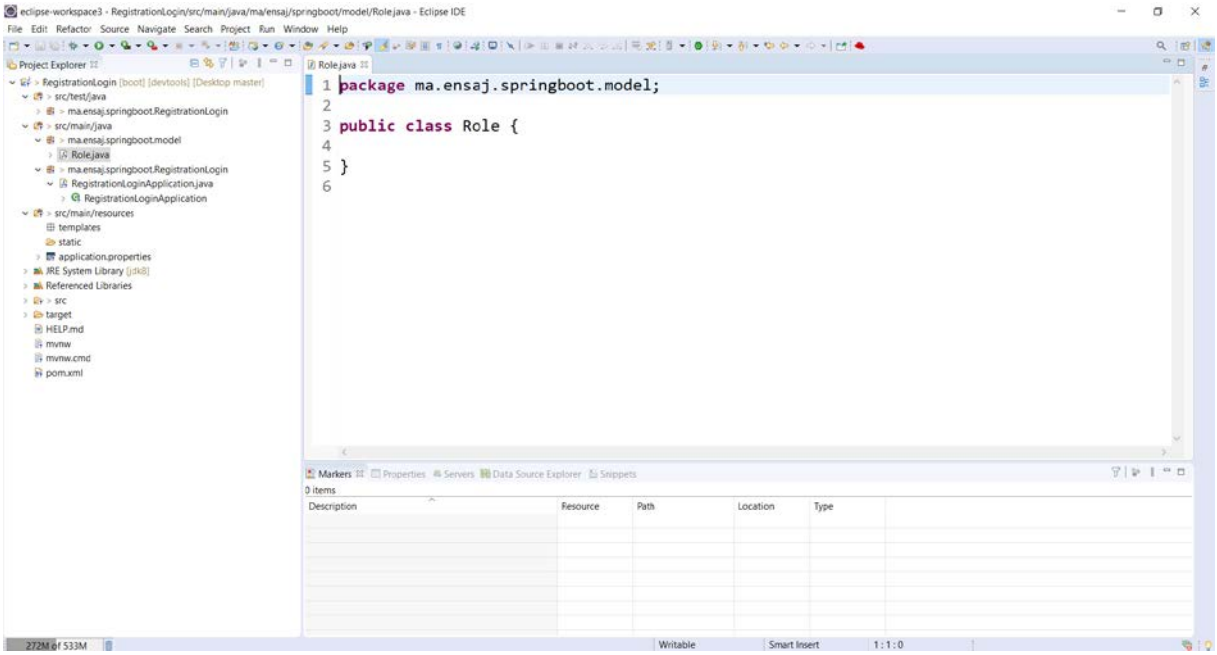


Figure 127:Role class

13. Attributes of Role class

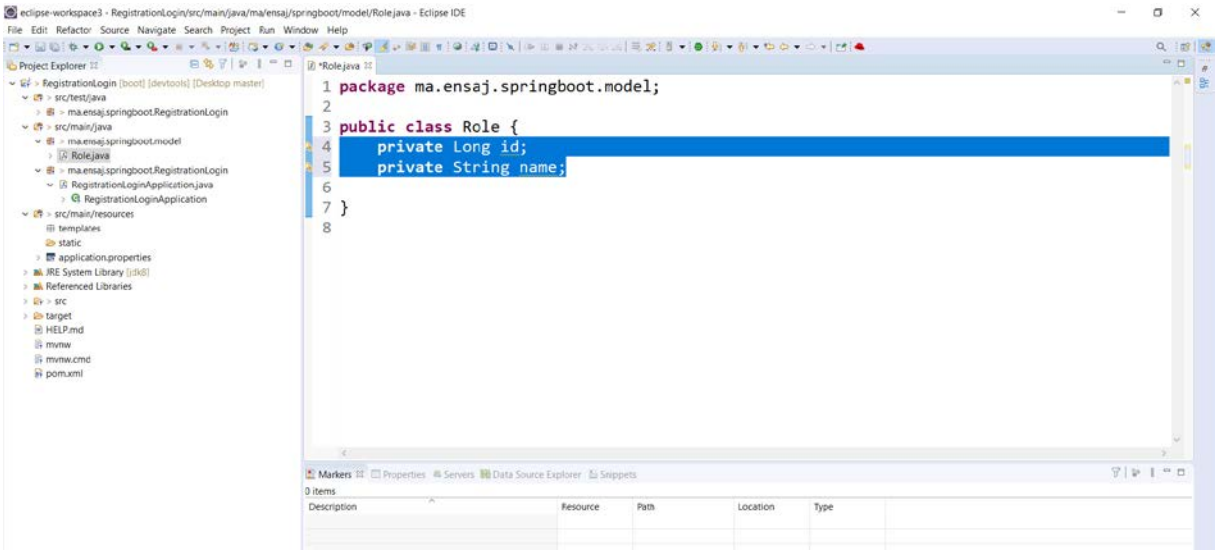


Figure 128:Attributes of Role class

14. Generate constructors following our needs and wants

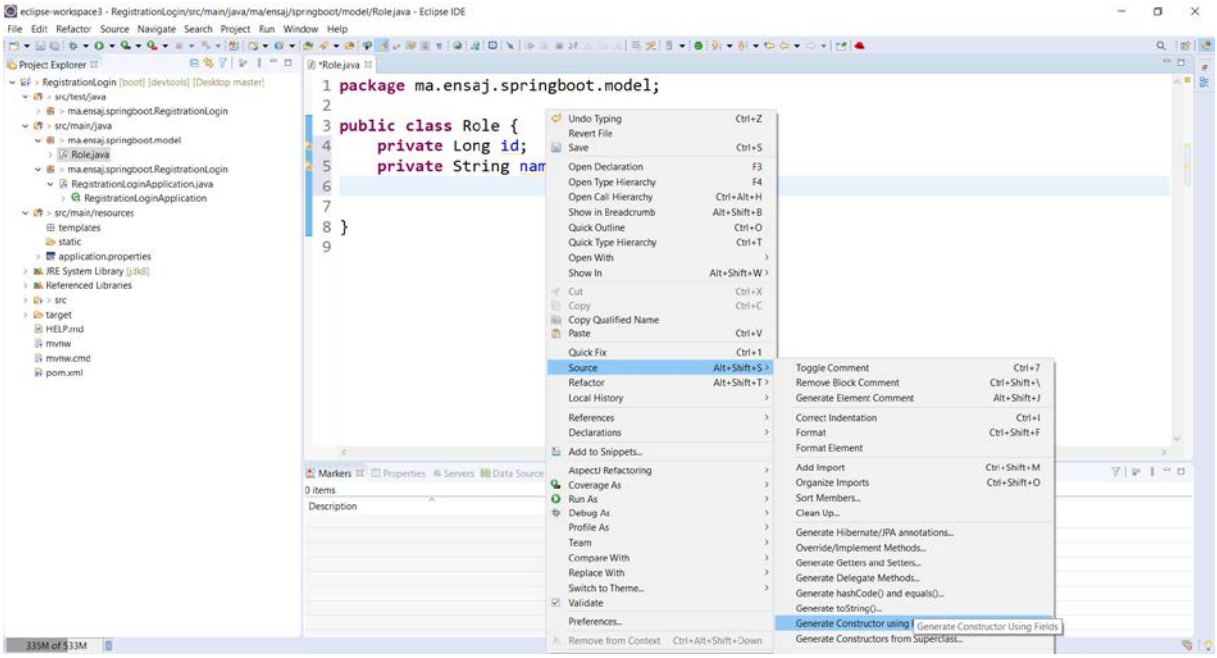


Figure 129:Generate constructors following our needs and wants

15. Select name field

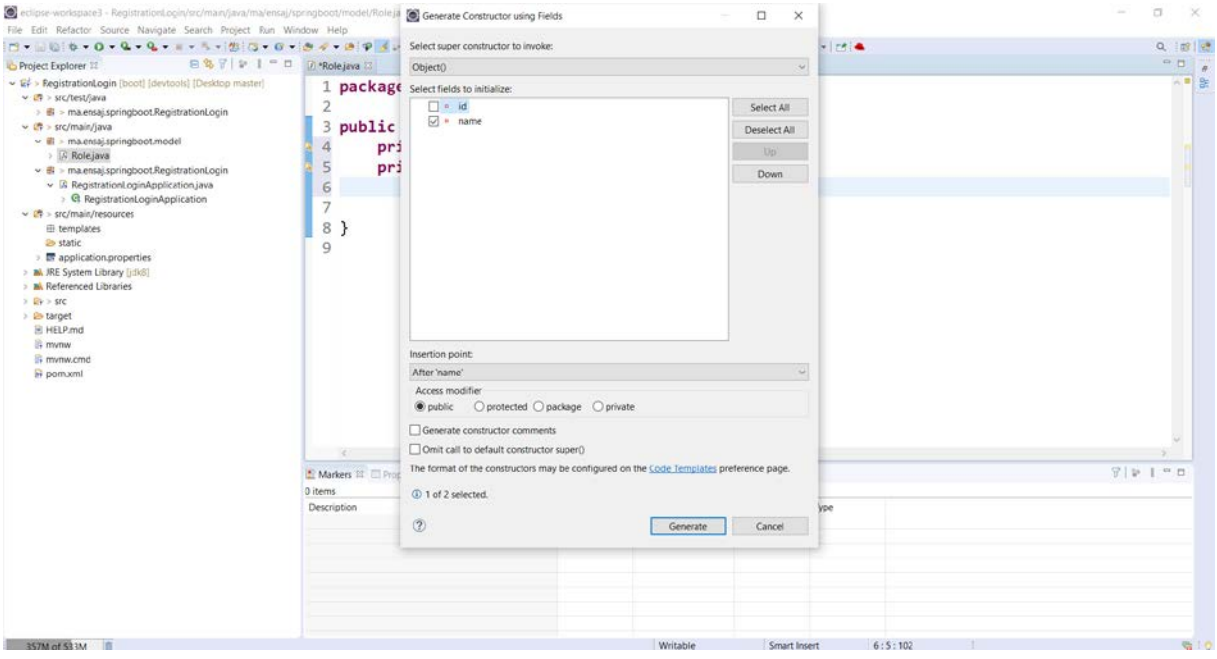


Figure 130:Select name field

16. Generate getters and setters

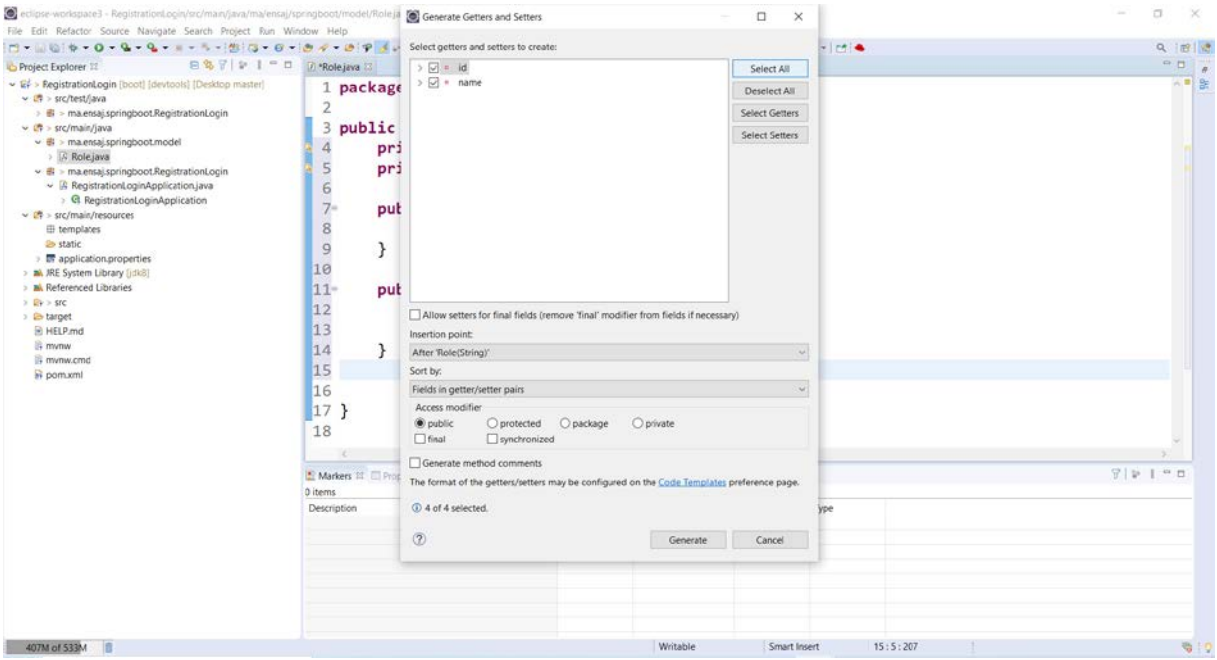
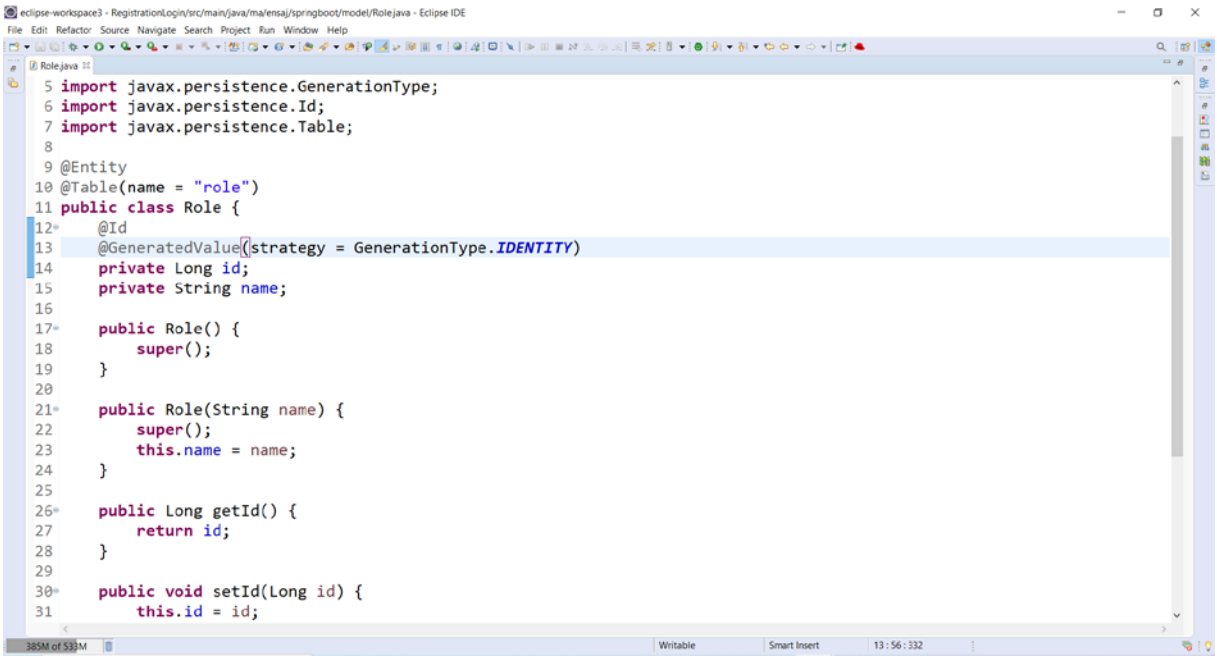


Figure 131:Generate getters and setters

17. Add some annotations**:@Entity,@Table,@Id and @GeneratedValue**



```
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name = "role")
public class Role {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;

    public Role() {
        super();
    }

    public Role(String name) {
        super();
        this.name = name;
    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
```

Figure 132:Add some annotations:@Entity,@Table,@Id,@GeneratedValue

18. Create the "**User**" class in a sub-package of the main package (here: ma.ensaj.springboot.model):

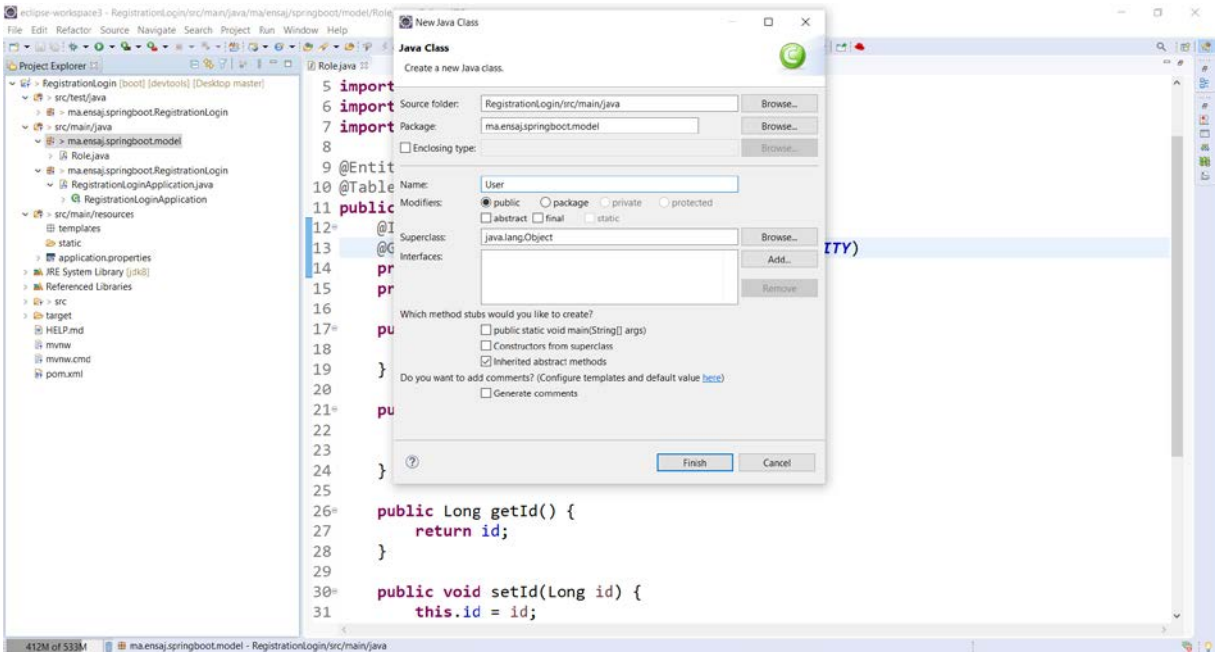


Figure 133:Create the "User" class

19. User's class attributes

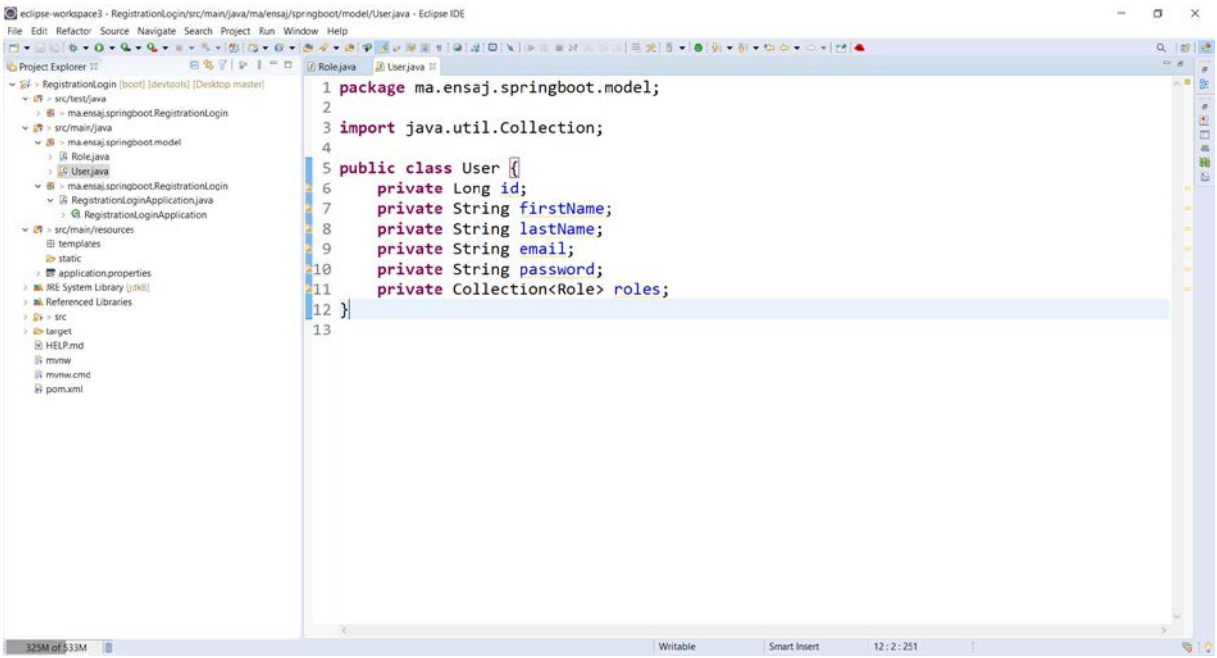


Figure 134:User's class attributes

20. Generate constructors following our needs and wants

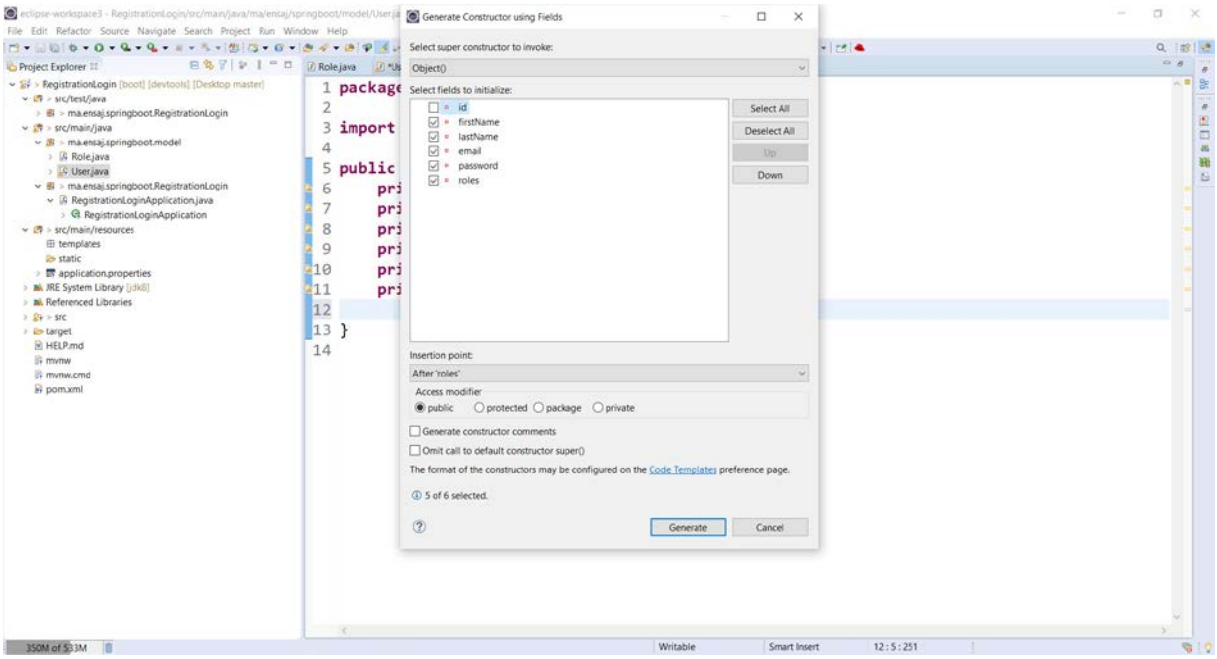


Figure 135:Generate constructors following our needs and wants

21. Generate getters and setters

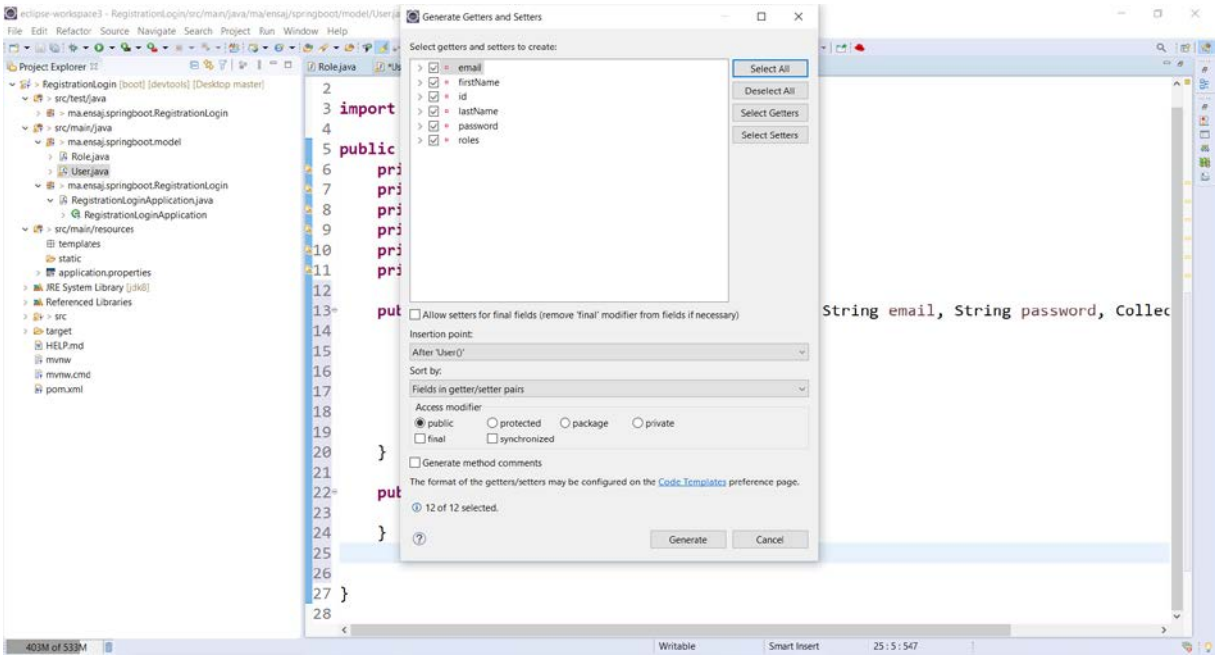
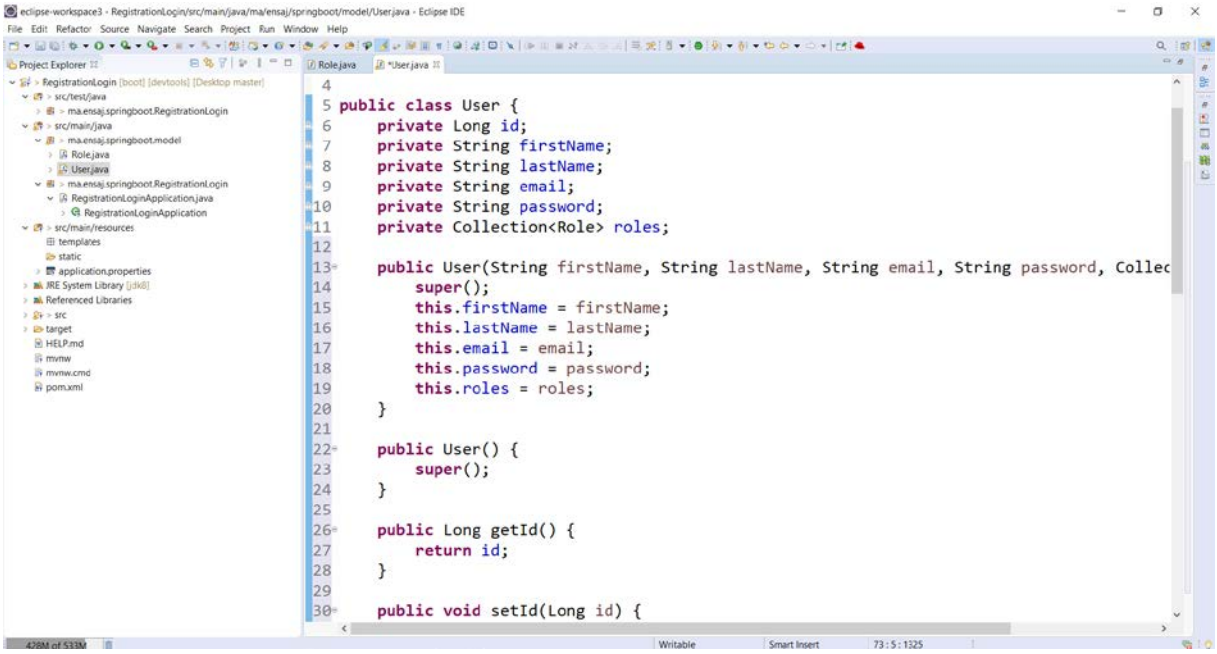


Figure 136:Generate getters and setters

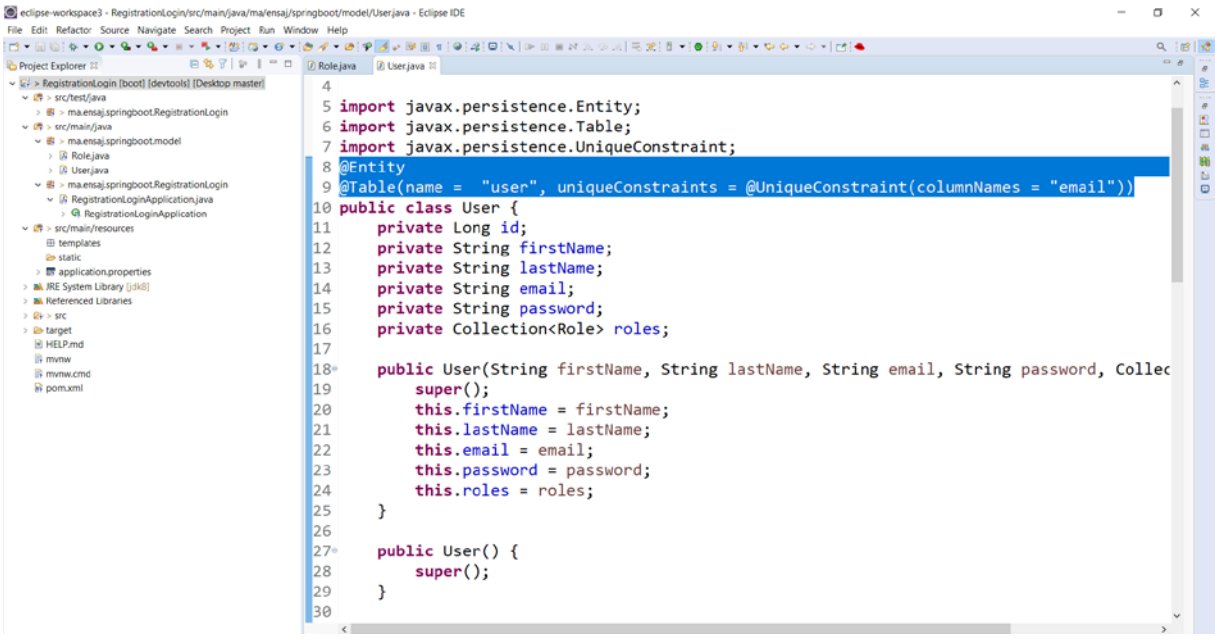
22. User class



```java
public class User {
    private Long id;
    private String firstName;
    private String lastName;
    private String email;
    private String password;
    private Collection<Role> roles;

    public User(String firstName, String lastName, String email, String password, Collec
        super();
        this.firstName = firstName;
        this.lastName = lastName;
        this.email = email;
        this.password = password;
        this.roles = roles;
    }

    public User() {
        super();
    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
```

Figure 137:User class

23. Add **@Entity , @Table and (@UniqueConstraints** for email) annotations



```java
import javax.persistence.Entity;
import javax.persistence.Table;
import javax.persistence.UniqueConstraint;
@Entity
@Table(name =  "user", uniqueConstraints = @UniqueConstraint(columnNames = "email"))
public class User {
    private Long id;
    private String firstName;
    private String lastName;
    private String email;
    private String password;
    private Collection<Role> roles;

    public User(String firstName, String lastName, String email, String password, Collec
        super();
        this.firstName = firstName;
        this.lastName = lastName;
        this.email = email;
        this.password = password;
        this.roles = roles;
    }

    public User() {
        super();
    }
```

Figure 138:Add @Entity , @Table and (@UniqueConstraints for email) annotations

24. Adding @Id and @GeneratedValue annotations for Id field

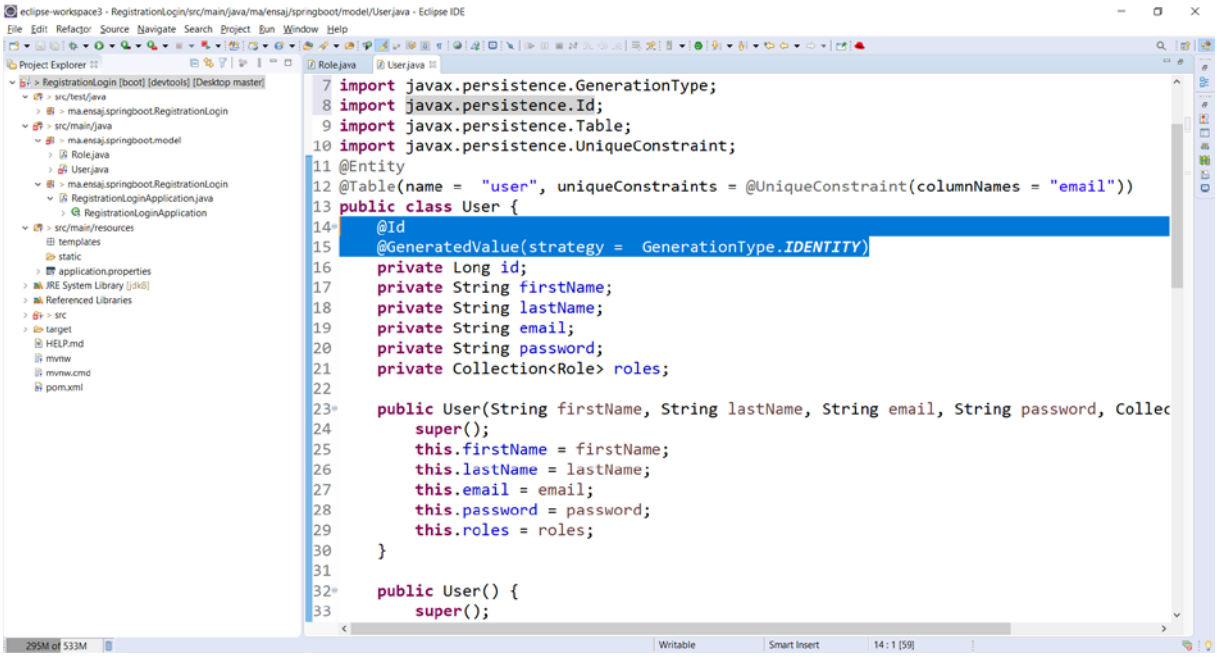


Figure 139:Adding @Id and @GeneratedValue annotations for Id field

25. **@Column** for column names

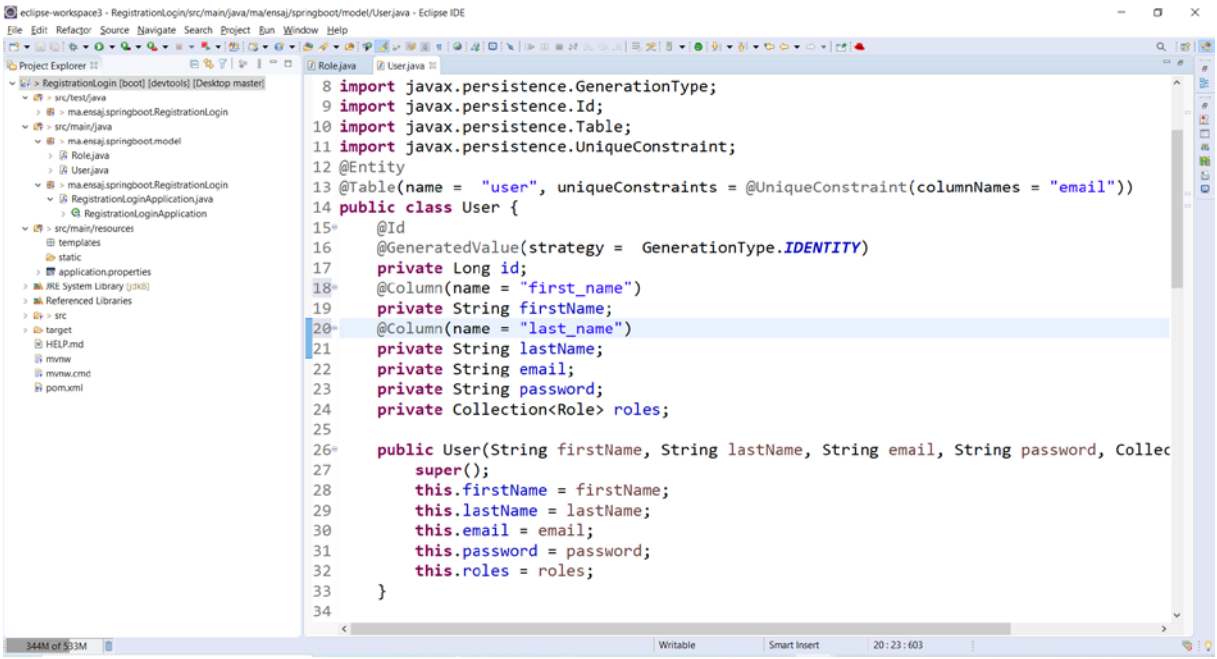


Figure 140:@Column for column names

26. Adding **@ManyToMany** annotation for the relation between the User and the Role classes.

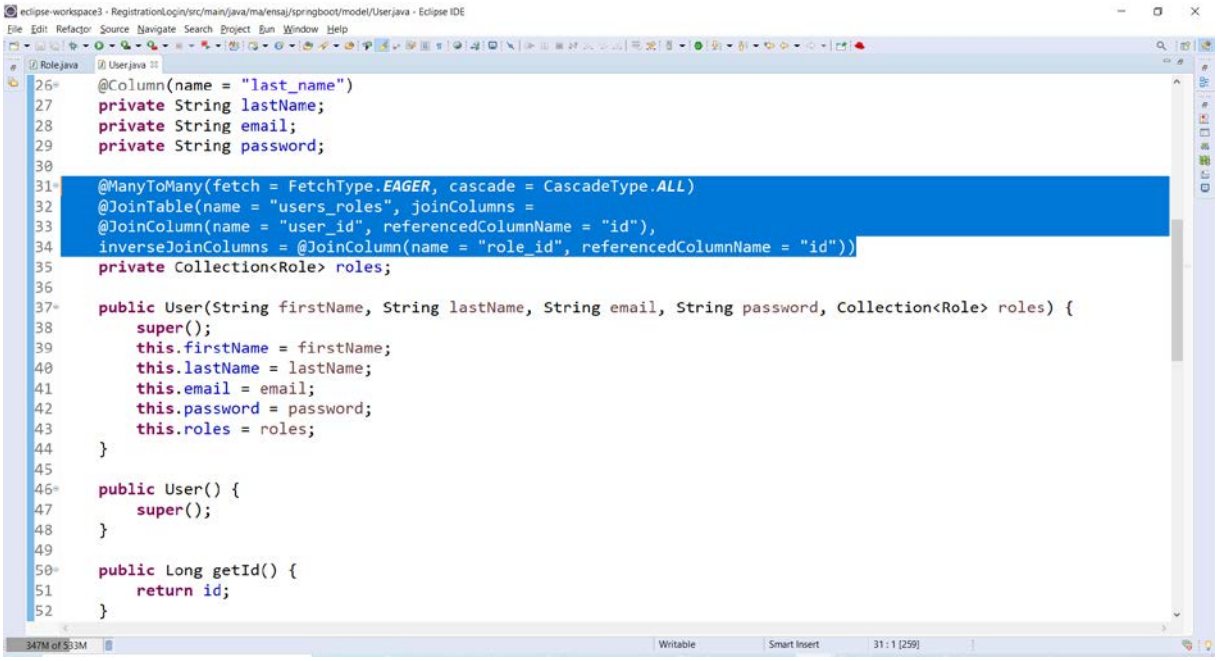


Figure 141:Adding @ManyToMany annotation for the relation between the User and the Role classes.

27. Let’s run the app: To start the application, use **mvn spring-boot:run** command

```
C:\Users\hhass\Desktop\RegistrationLogin\RegistrationLogin>mvn spring-boot:run
```

Figure 142:Let’s run the app: To start the application, use mvn spring-boot:run command

28. We can see now in the cmd **Started RegistrationLoginApplication**

Figure 143:We can see now in the cmd Started RegistrationLoginApplication

29. Check if the "user",”role” tables are created.

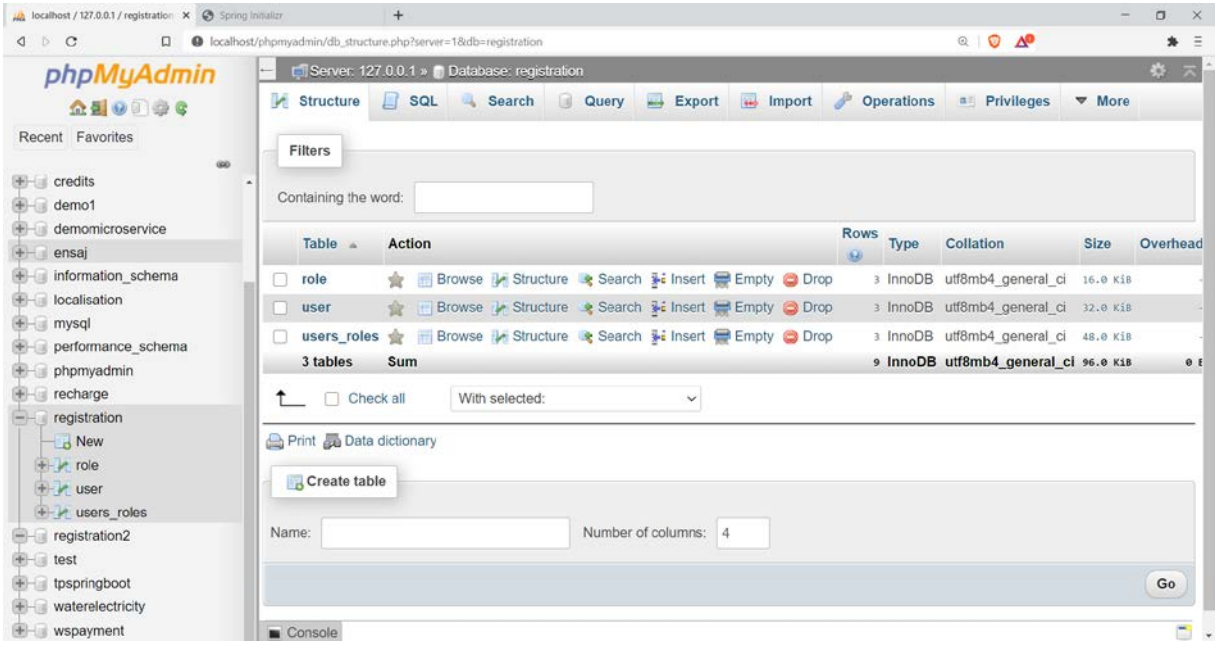


Figure 144:Check if the "user",”role” tables are created.

30. The structure of **User** table

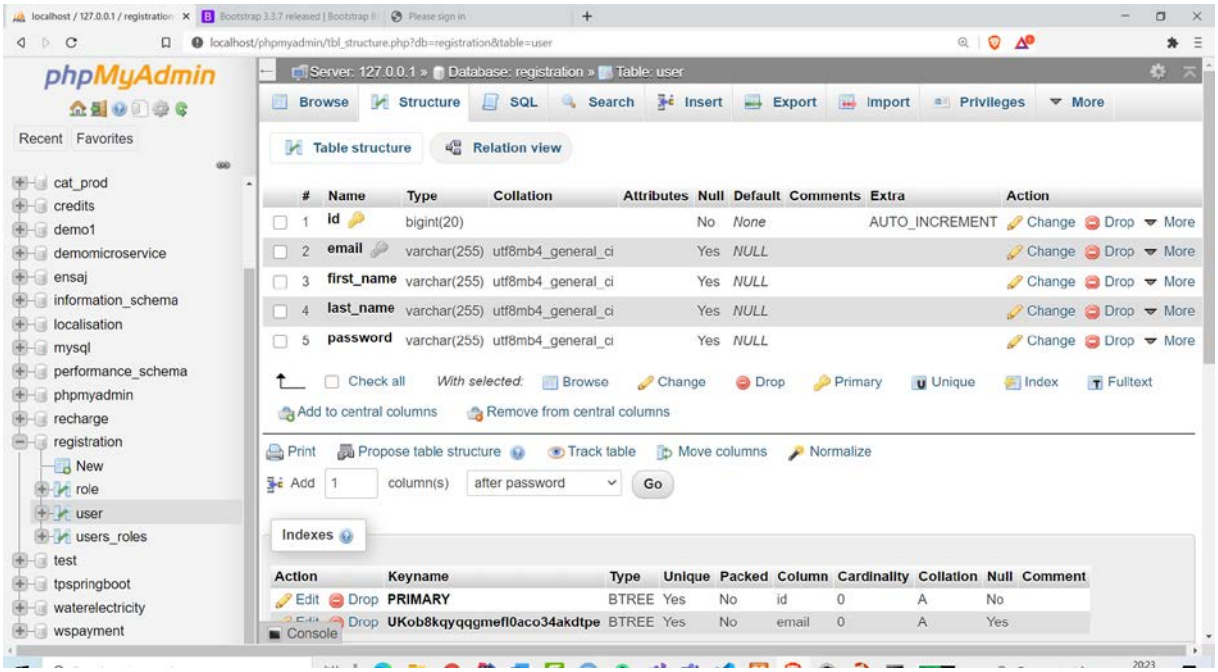


Figure 145:The structure of User table

31. The structure of **Role** table

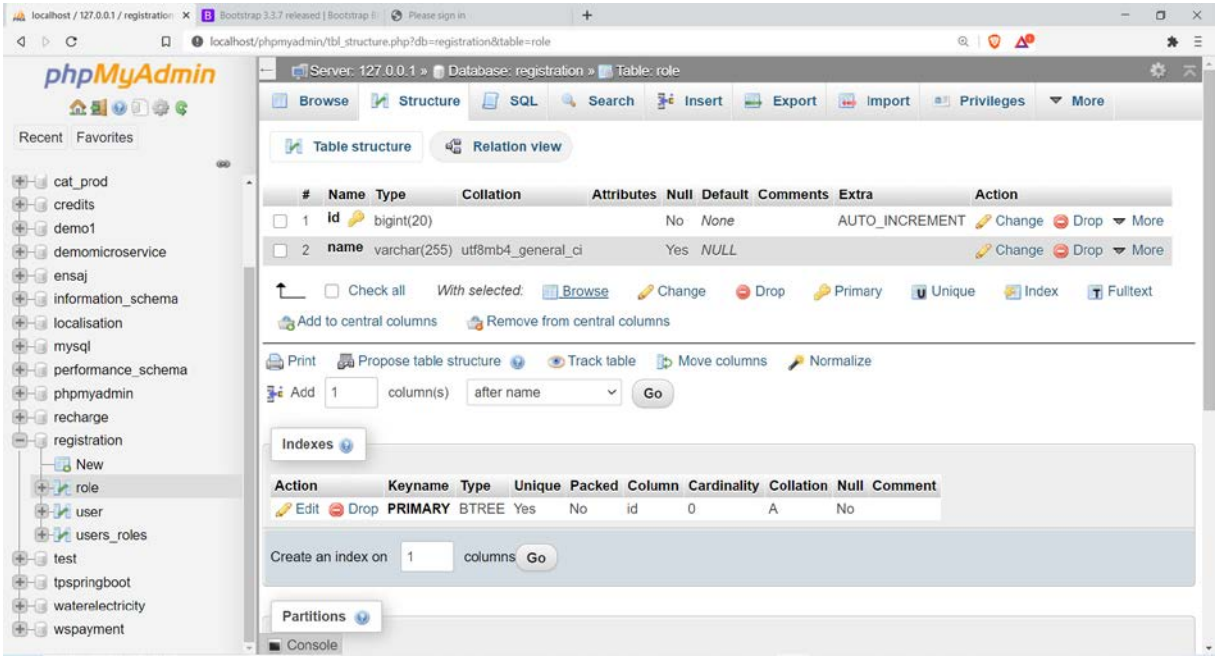


Figure 146:The structure of Role table

32. The structure of **Users_roles** table(Association-class)

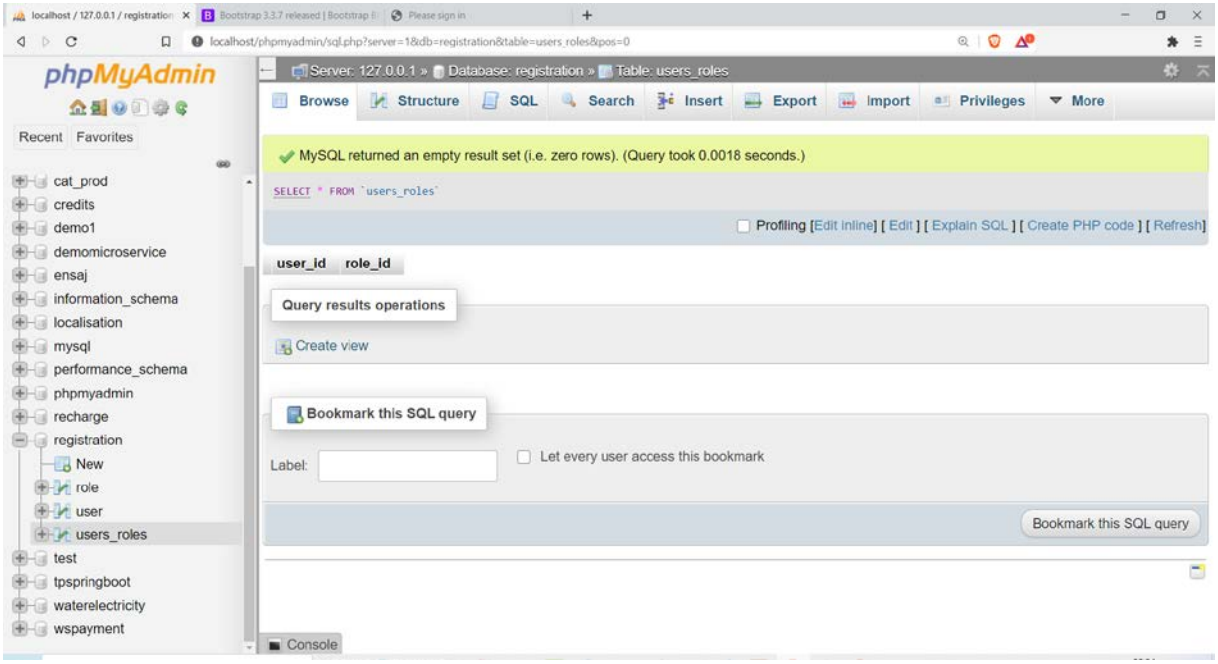


Figure 147:The structure of Users_roles table(Association-class)

Backend implementation for registration feature(Repository, Service and Controller)

33. Creation of repositories

Spring Data JPA provides a data access layer implementation for a Spring application.

It is a very convenient brick because it allows not reinventing the data access wheel with each new application, and thus allowing you to focus on the business side.

34. Create the "**UserRepository**" interface in a sub-package -called repository- of the main package:

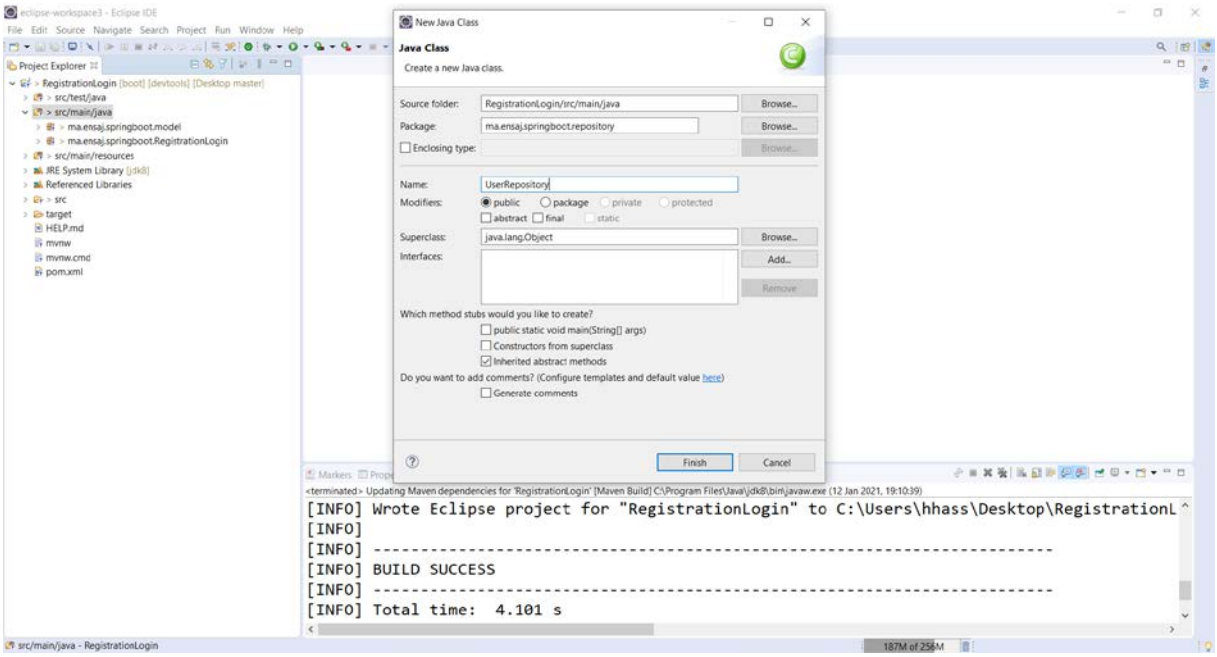


Figure 148:Create the "UserRepository" interface

35. **UserRepository** interface

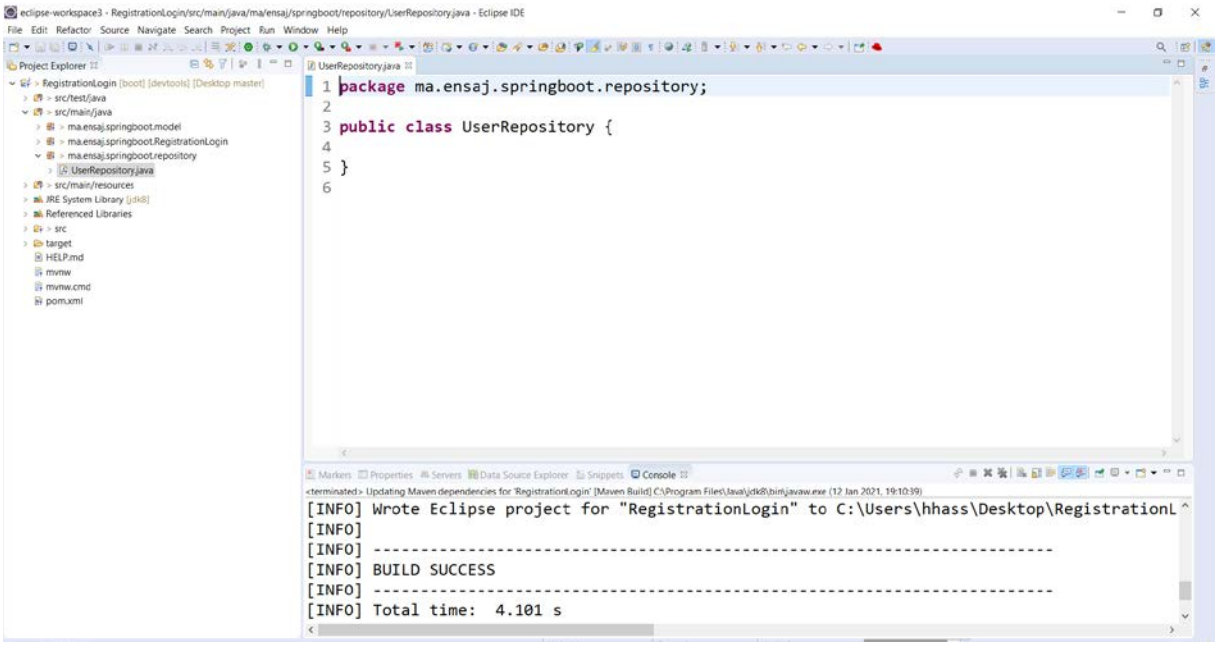


Figure 149: "UserRepository" interface

36. In the interface **UserRepository** that **extends** the **JpaRepository** add the method **findByEmail** that return an user

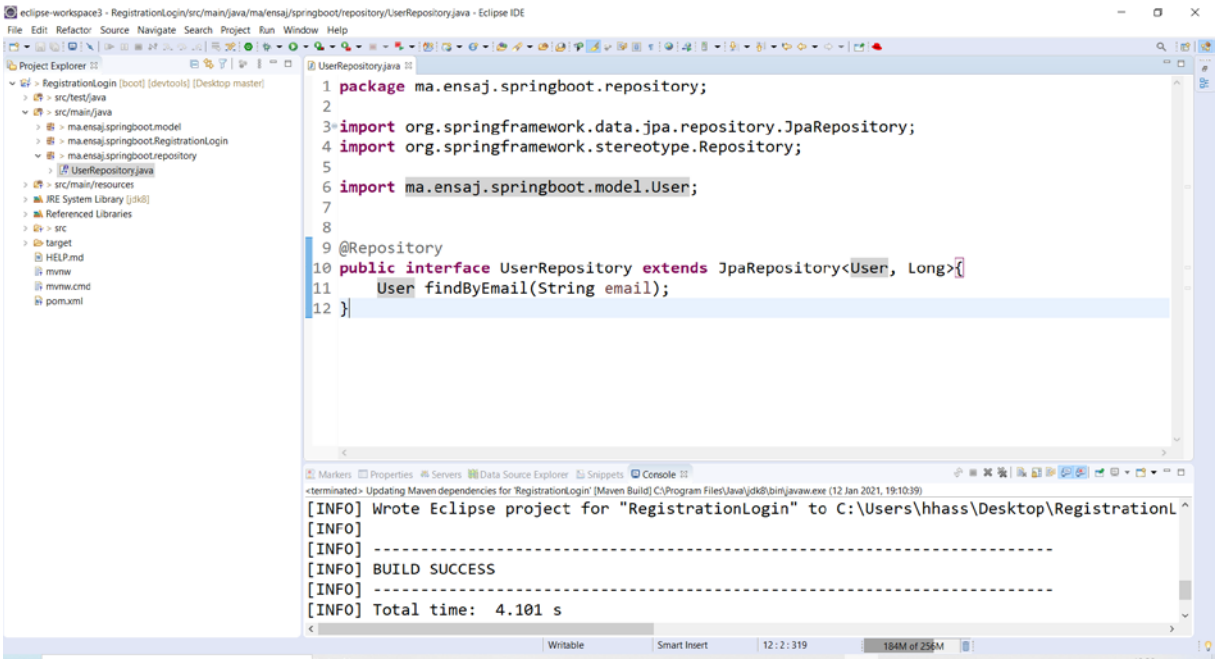


Figure 150:the interface UserRepository extends the JpaRepository

37. Creation of **UserService** class under the sub package **service** of the main package

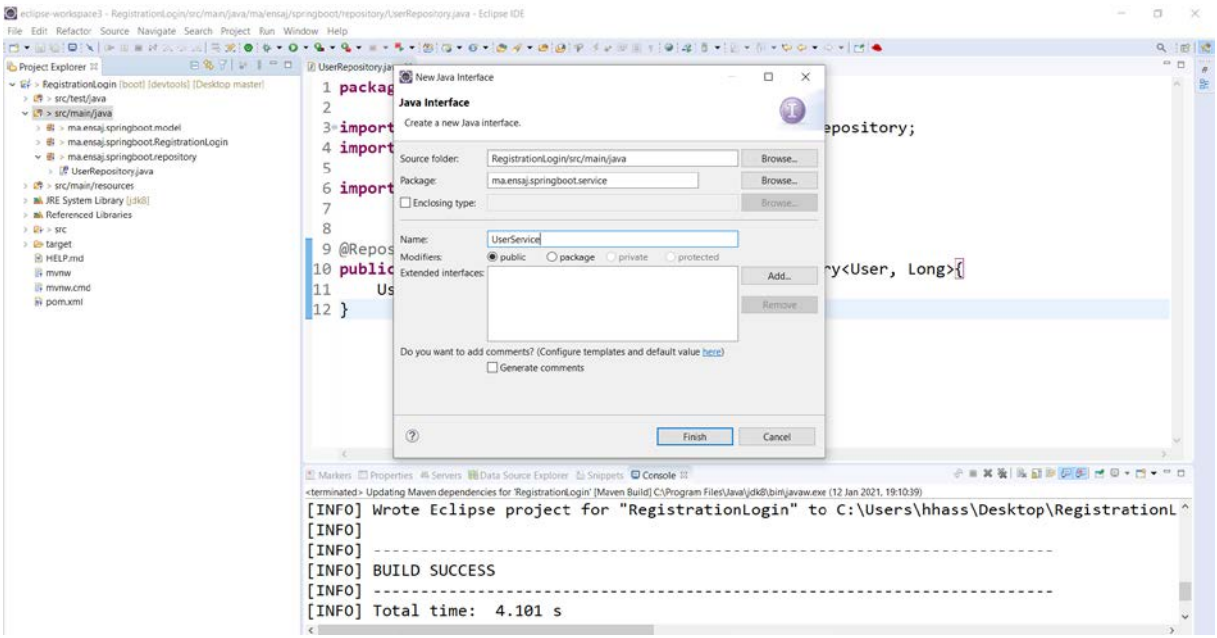


Figure 151:Creation of UserService class under the sub package service of the main package

38. Creation of **UserRegistrationDto class**

N.B:DTO: The Transfer Object pattern is used when we want to pass data with multiple attributes in one shot from client to server.

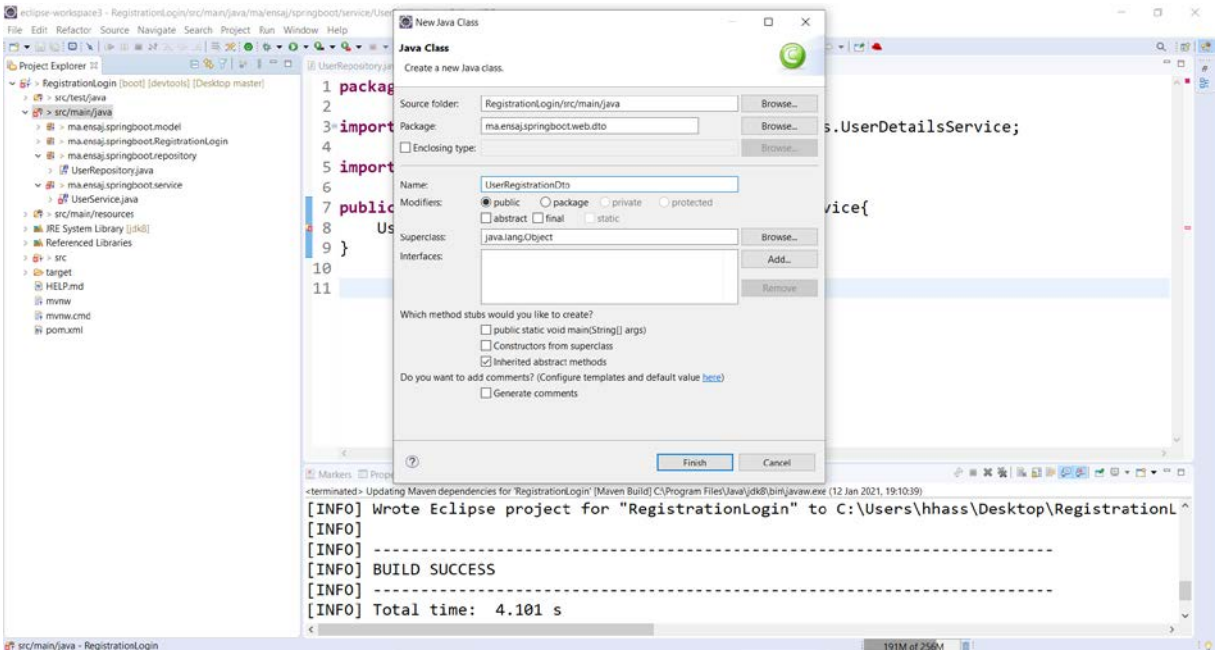


Figure 152:Creation of UserRegistrationDto class

39. **UserRegistrationDto** class attributes

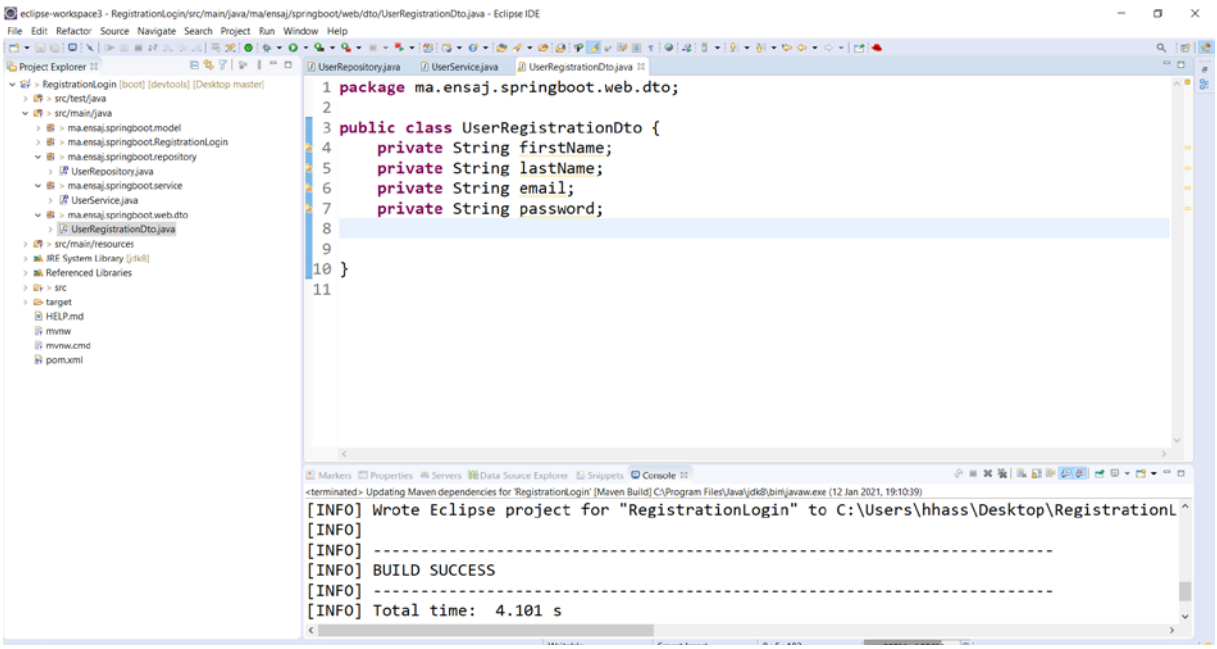


Figure 153:UserRegistrationDto class attributes

40. **UserRegistrationDto class** after adding default constructor and other one with parameters and getters and setters

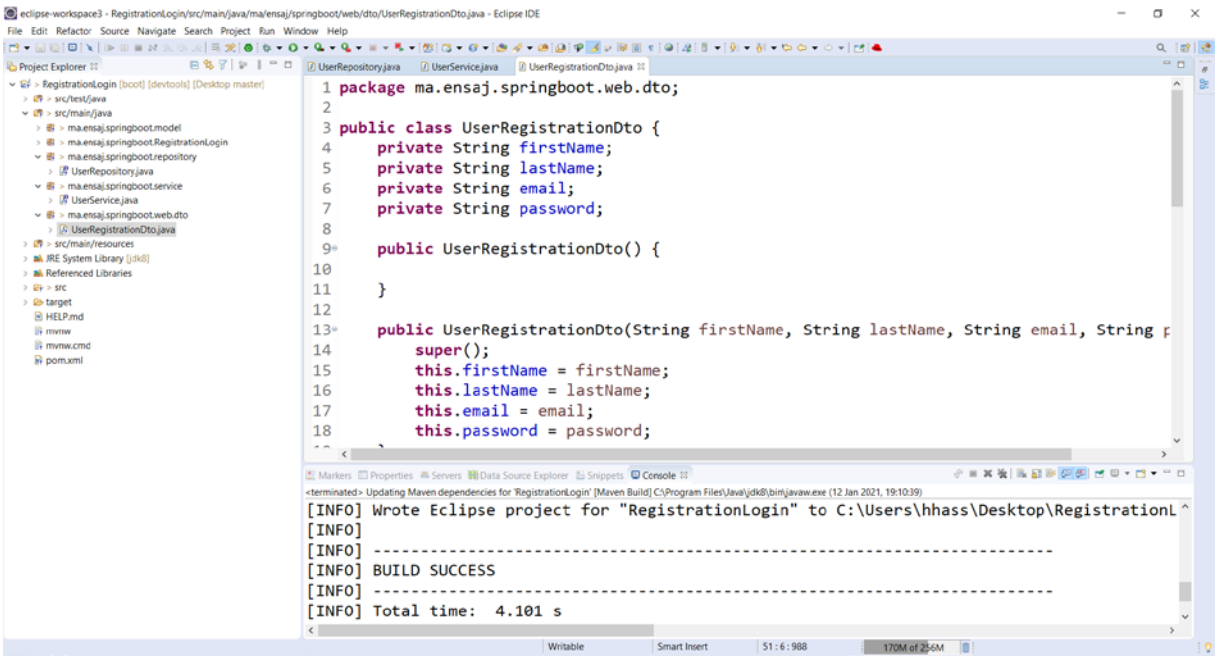


Figure 154:UserRegistrationDto class after adding constructors

41. Adding the save method's signature to the **UserService** interface

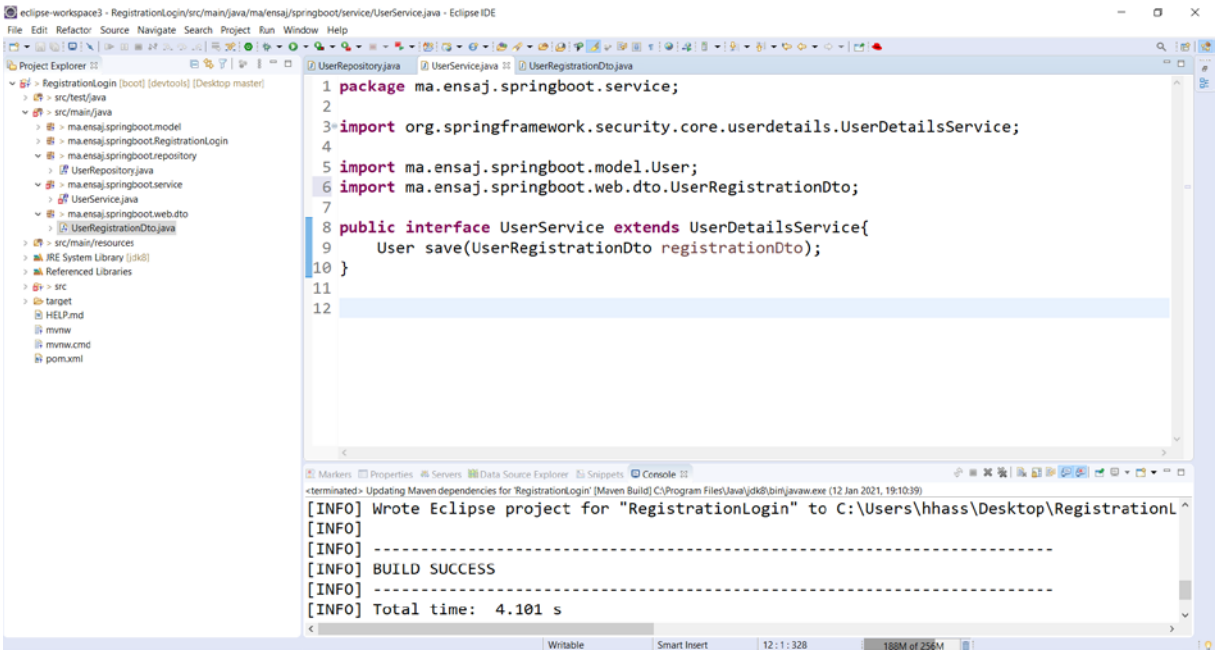


Figure 155:Adding the save method's signature to the UserService interface

42. Creation of **UserServiceImpl** class

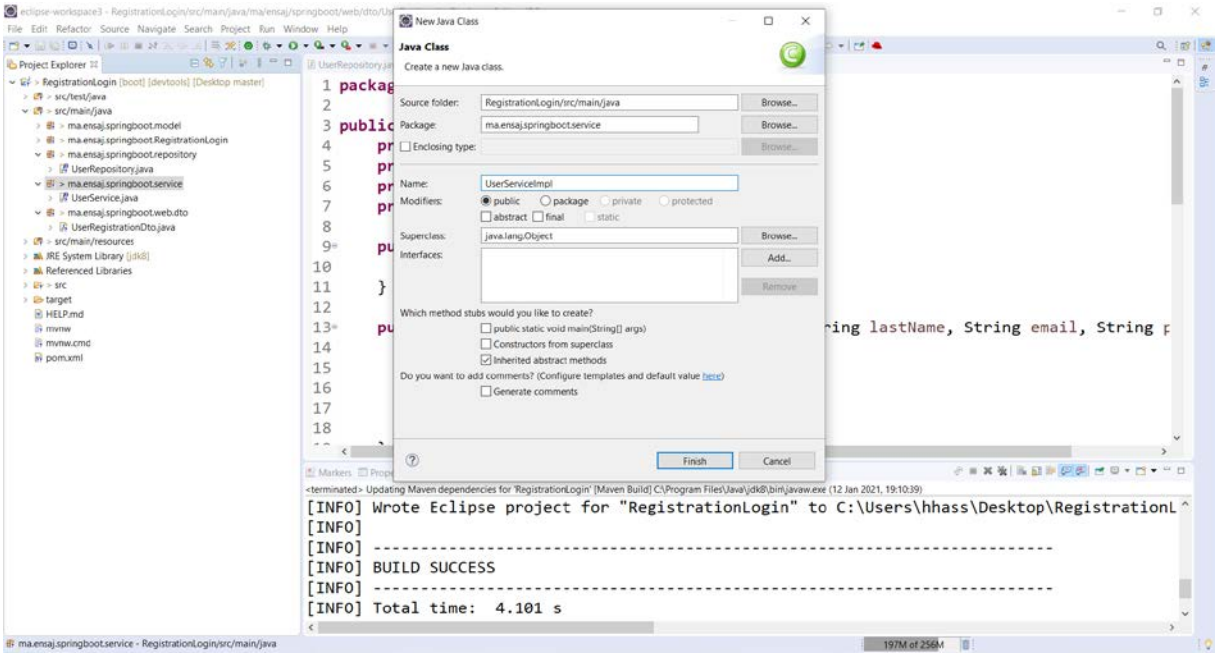


Figure 156:Creation of UserServiceImpl class

43. **UserServiceImpl** class is created

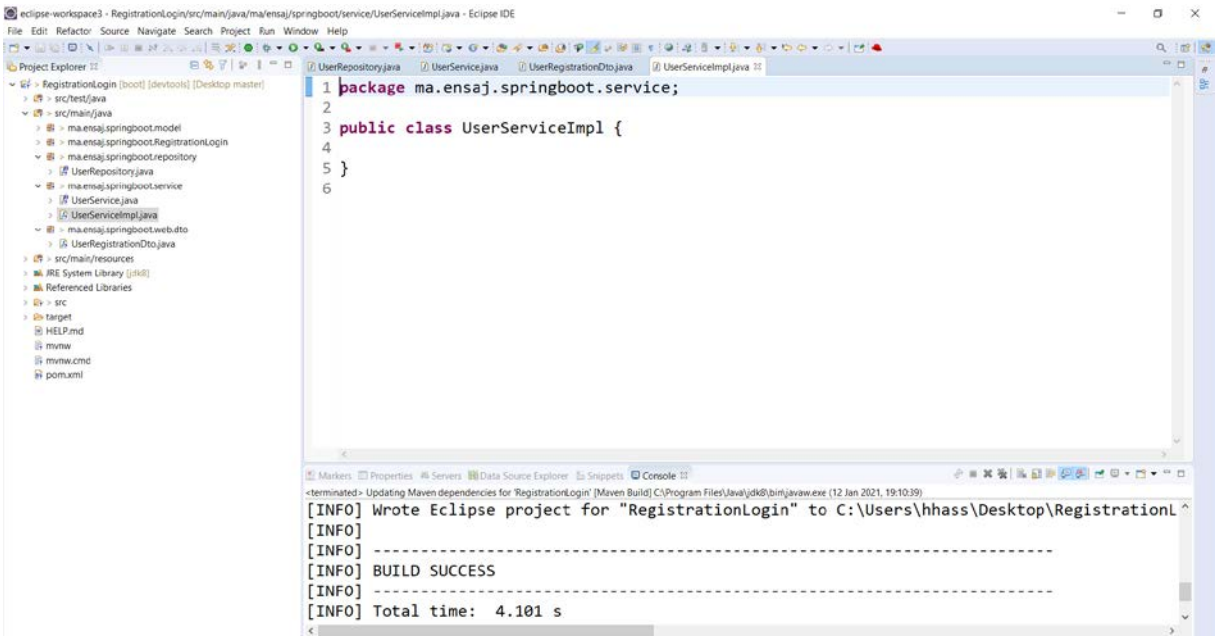
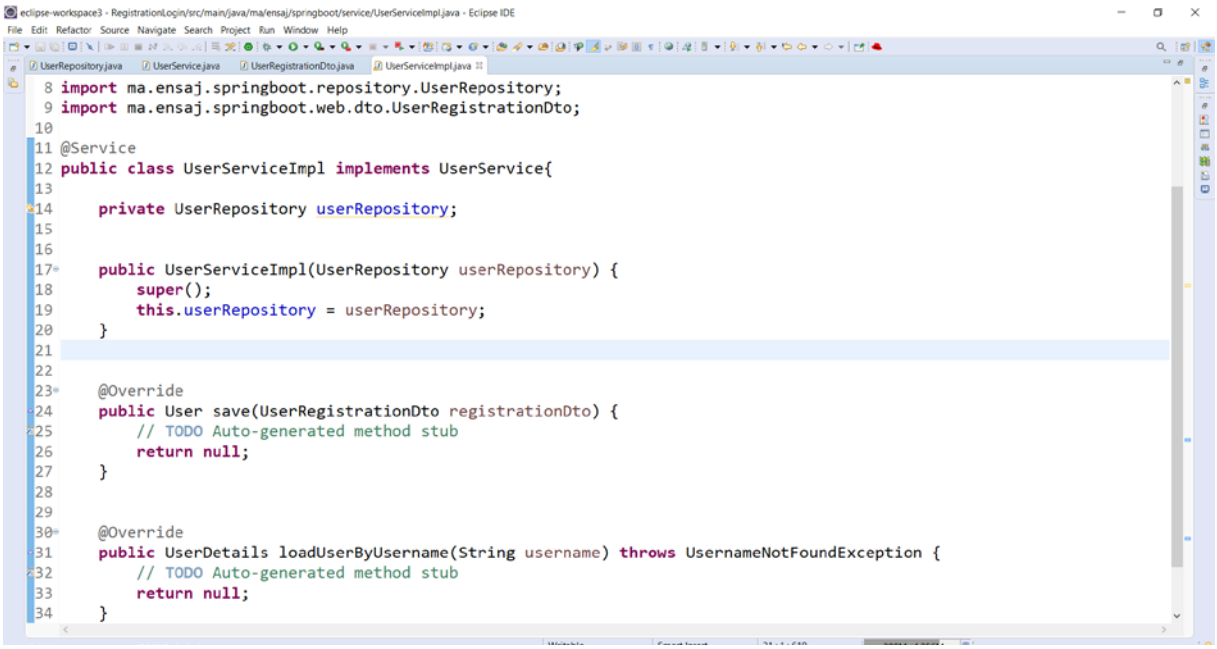


Figure 157:UserServiceImpl class is created

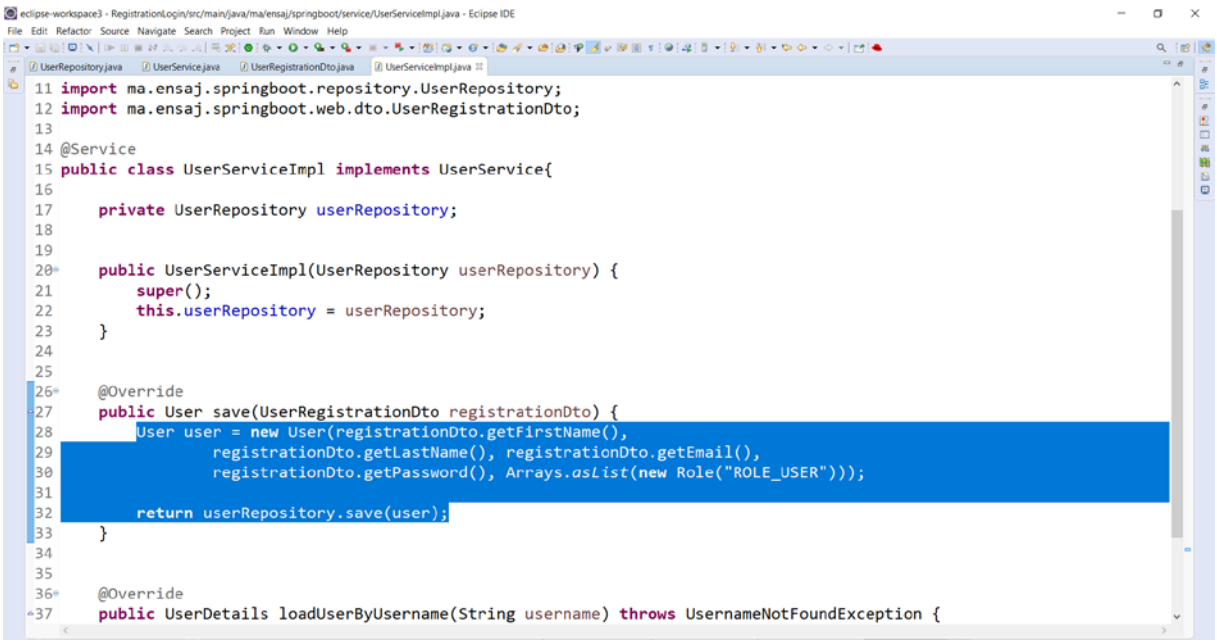
44. Add **@Service** annotation to the class and add **UserRepository** reference. After that generation of the constructor and implement the methods of **UserService** interface



```
import ma.ensaj.springboot.repository.UserRepository;
import ma.ensaj.springboot.web.dto.UserRegistrationDto;

@Service
public class UserServiceImpl implements UserService{

    private UserRepository userRepository;


    public UserServiceImpl(UserRepository userRepository) {
        super();
        this.userRepository = userRepository;
    }


    @Override
    public User save(UserRegistrationDto registrationDto) {
        // TODO Auto-generated method stub
        return null;
    }


    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
        // TODO Auto-generated method stub
        return null;
    }
```

Figure 158:Add @Service annotation to the class and add UserRepository reference

45. The implementation of the save method that returns a new user



```
import ma.ensaj.springboot.repository.UserRepository;
import ma.ensaj.springboot.web.dto.UserRegistrationDto;

@Service
public class UserServiceImpl implements UserService{

    private UserRepository userRepository;


    public UserServiceImpl(UserRepository userRepository) {
        super();
        this.userRepository = userRepository;
    }


    @Override
    public User save(UserRegistrationDto registrationDto) {
        User user = new User(registrationDto.getFirstName(),
                registrationDto.getLastName(), registrationDto.getEmail(),
                registrationDto.getPassword(), Arrays.asList(new Role("ROLE_USER")));

        return userRepository.save(user);
    }


    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
```

Figure 159:The implementation of the save method that returns a new user

Controller

46. Creation of the class **UserRegistrationController** under the sub package **controller** of the main package

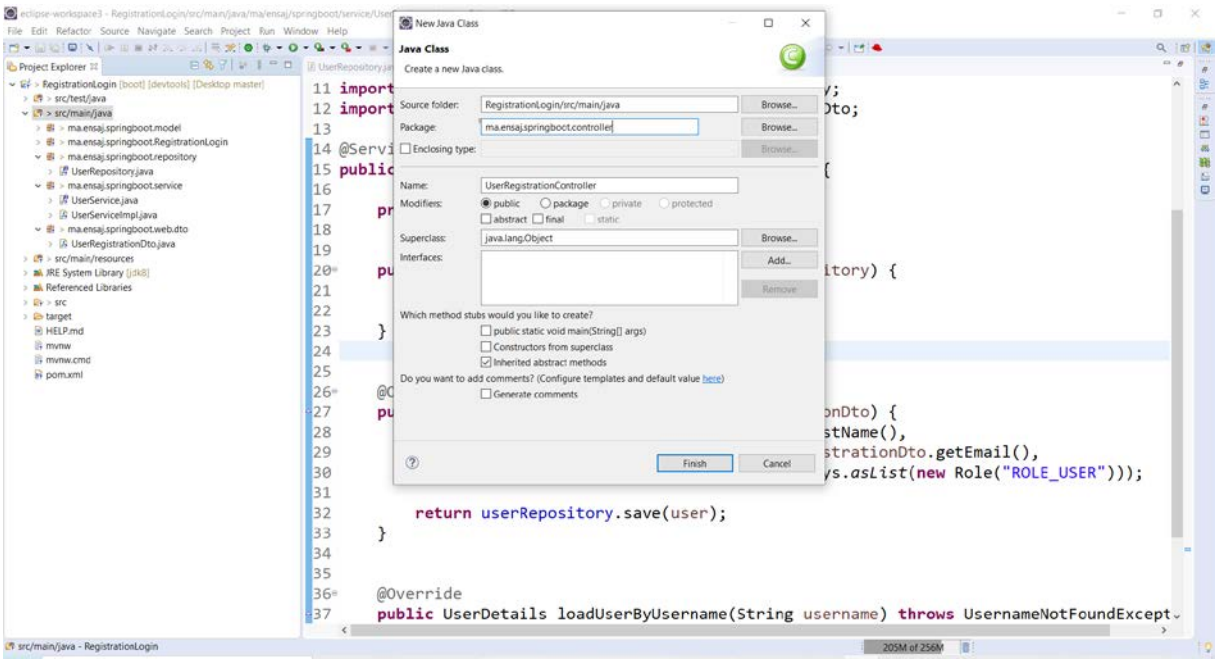


Figure 160:Creation of the class UserRegistrationController under the sub package controller of the main package

47. **UserRegistrationController class**

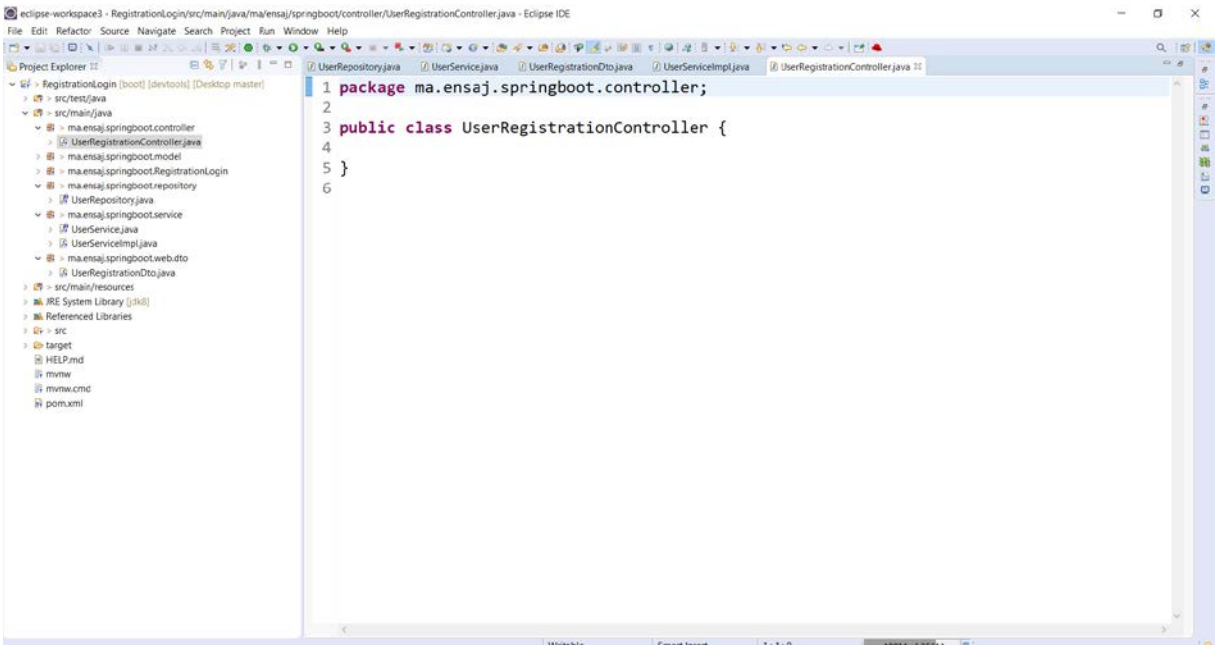


Figure 161:UserRegistrationController class

48. Add annotations **@Controller** and **@RequestMapping** which is used to map web requests to Spring Controller methods.

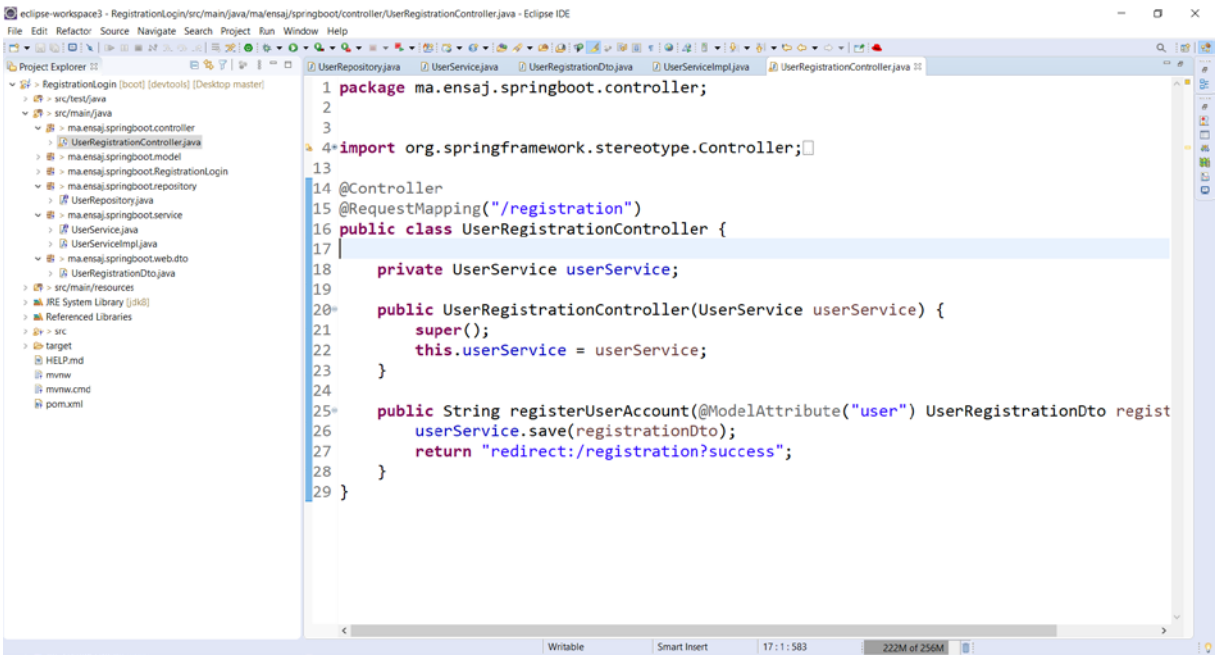


Figure 162:Add annotations @Controller and @RequestMapping which is used to map web requests to Spring Controller methods.

Frontend implementation of the Registration feature

49. Create a new html file under the **templates** folder called **registration**

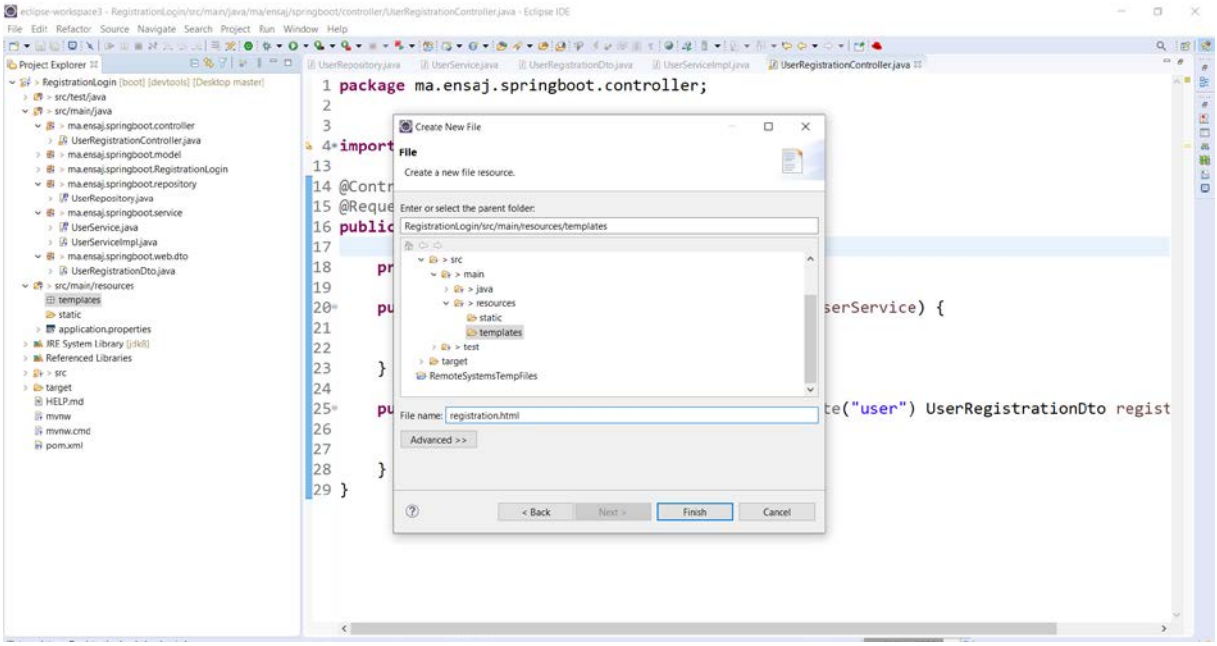


Figure 163:Create a new html file under the templates folder called registration

50. Add **showRegistraionForm** method with **@GetMapping** annotation to the **controller** that return the **page registration**

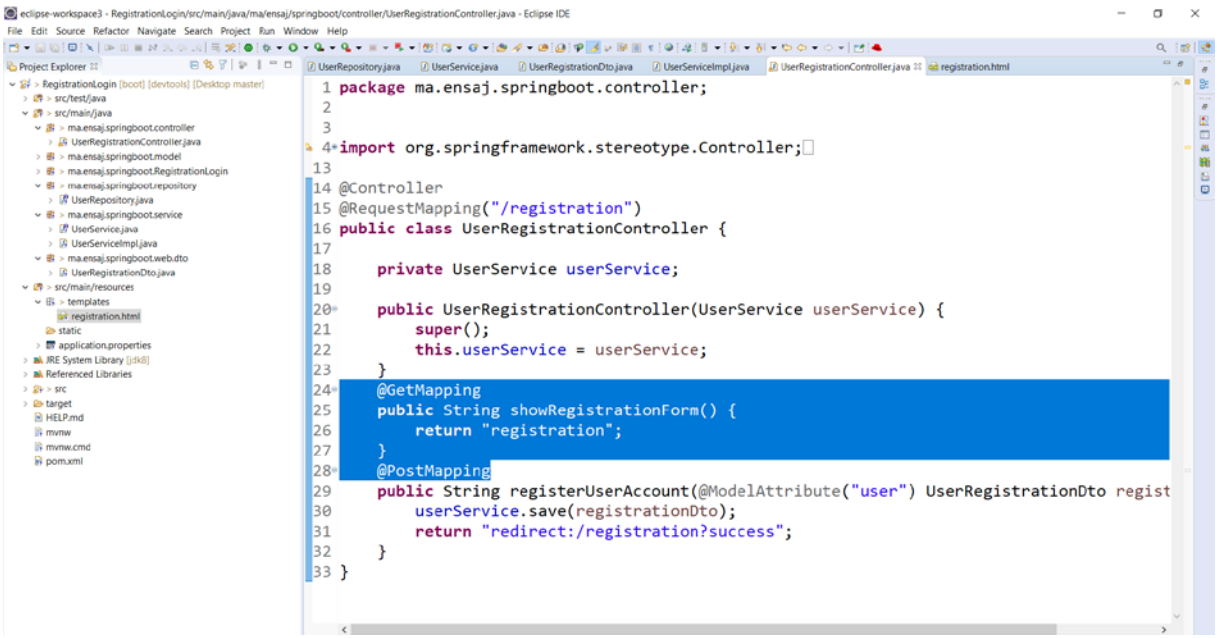


Figure 164:Add showRegistraionForm method with @GetMapping annotation to the controller that return the page registration

51. Adding the **userRegistrationDto** method with **@ModelAttribute** annotation

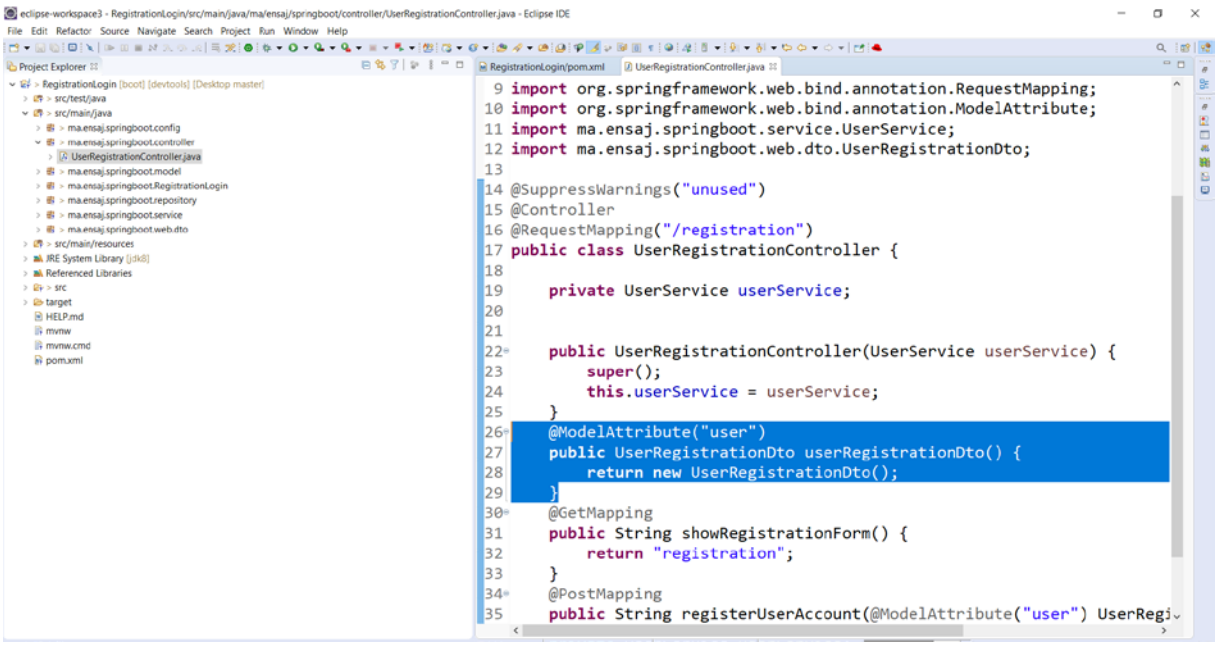


Figure 165:Adding the userRegistrationDto method with @ModelAttribute annotation

Here we go

Let’s develop our page

First, I need to use bootstrap for responsive application

52. Searching for bootstrap 3.3.7 cdn link in Google.

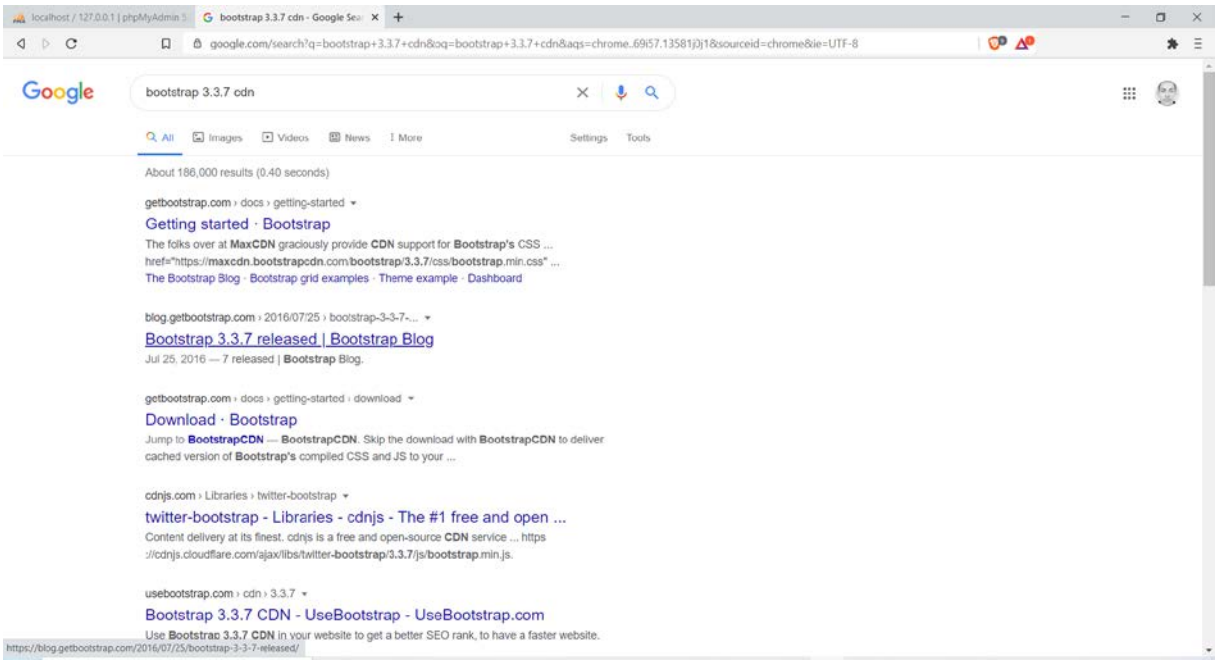


Figure 166:Searching for bootstrap 3.3.7 cdn link in Google.

53. Copy bootstrap 3.3.7 cdn’s link

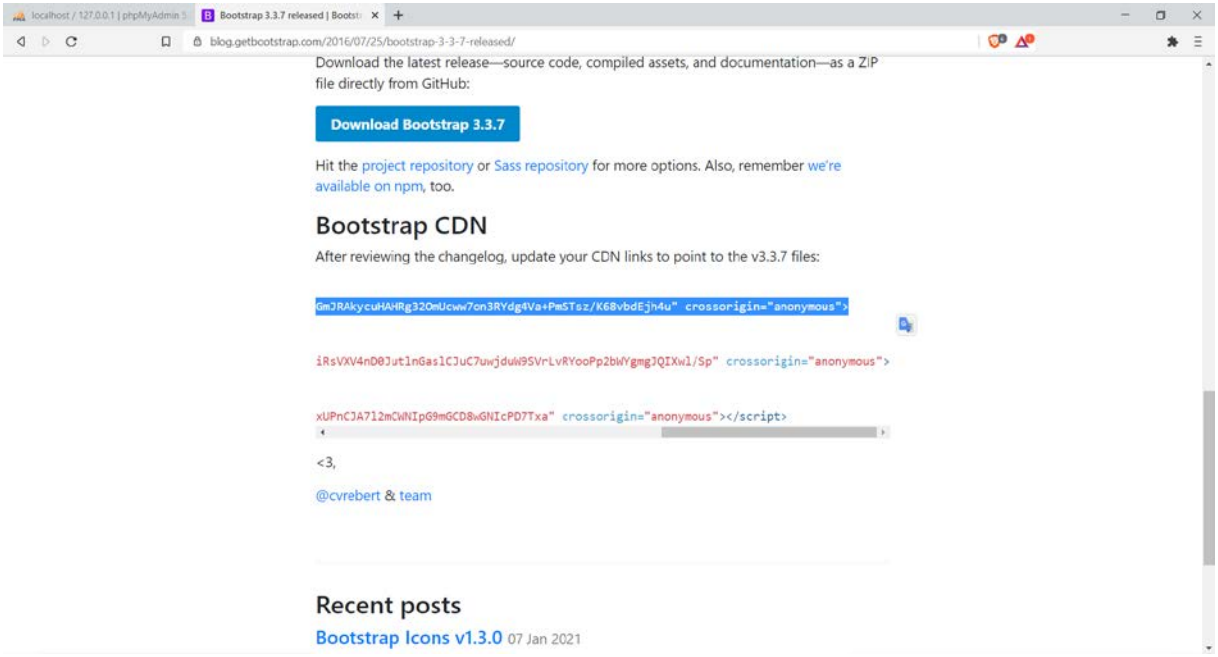


Figure 167:Copy bootstrap 3.3.7 cdn’s link

54. Paste bootstrap 3.3.7 cdn's link in the head section of our html page

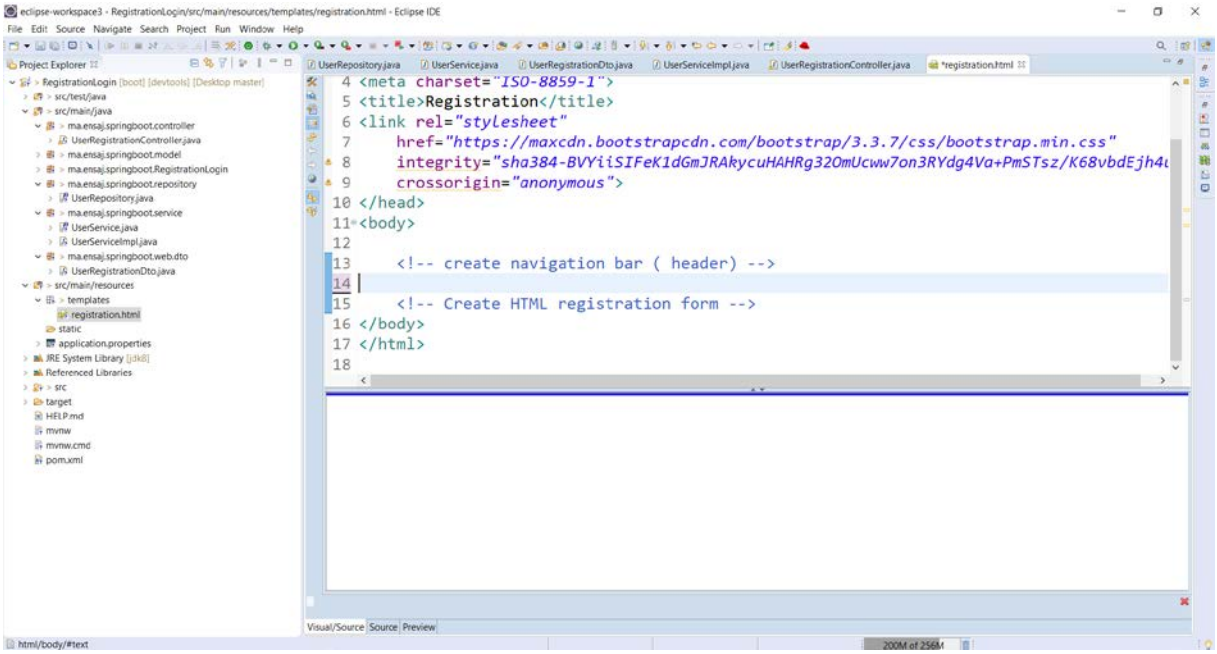


Figure 168:Paste bootstrap 3.3.7 cdn's link in the head section of our html page

55. Copy paste nav section from the official documentation of bootstrap and personalize it following our needs and wants

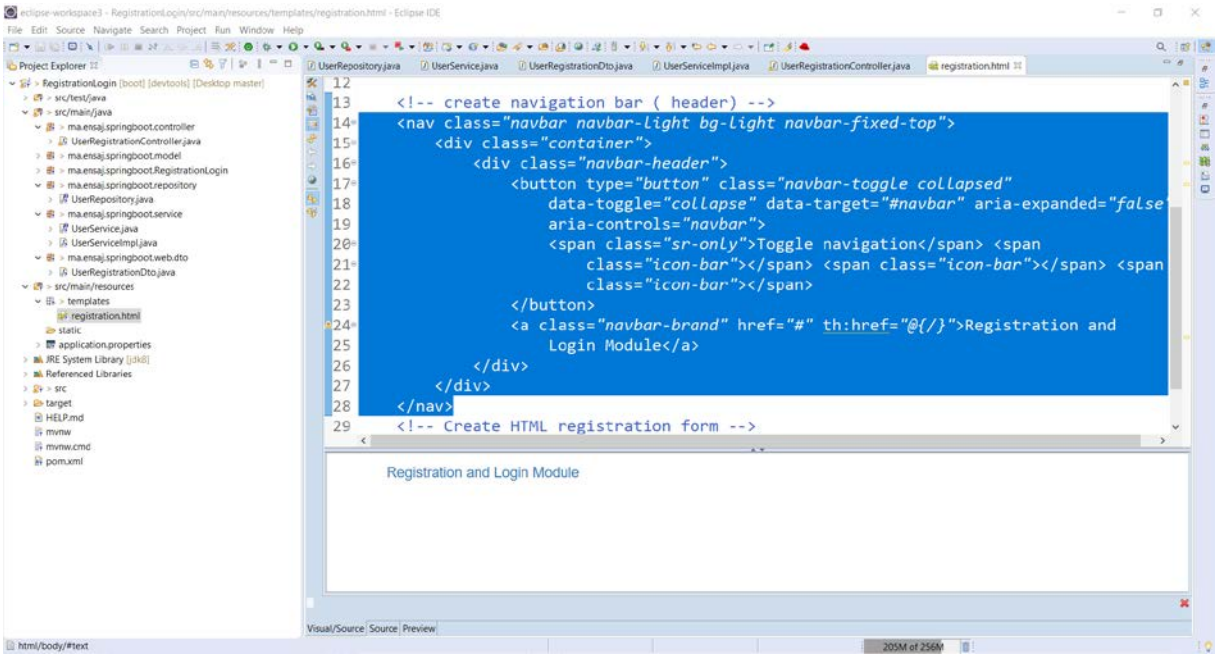


Figure 169:Copy paste nav section from the official documentation of bootstrap and personalize it following our needs and wants

56. Show success message using thymeleaf syntax

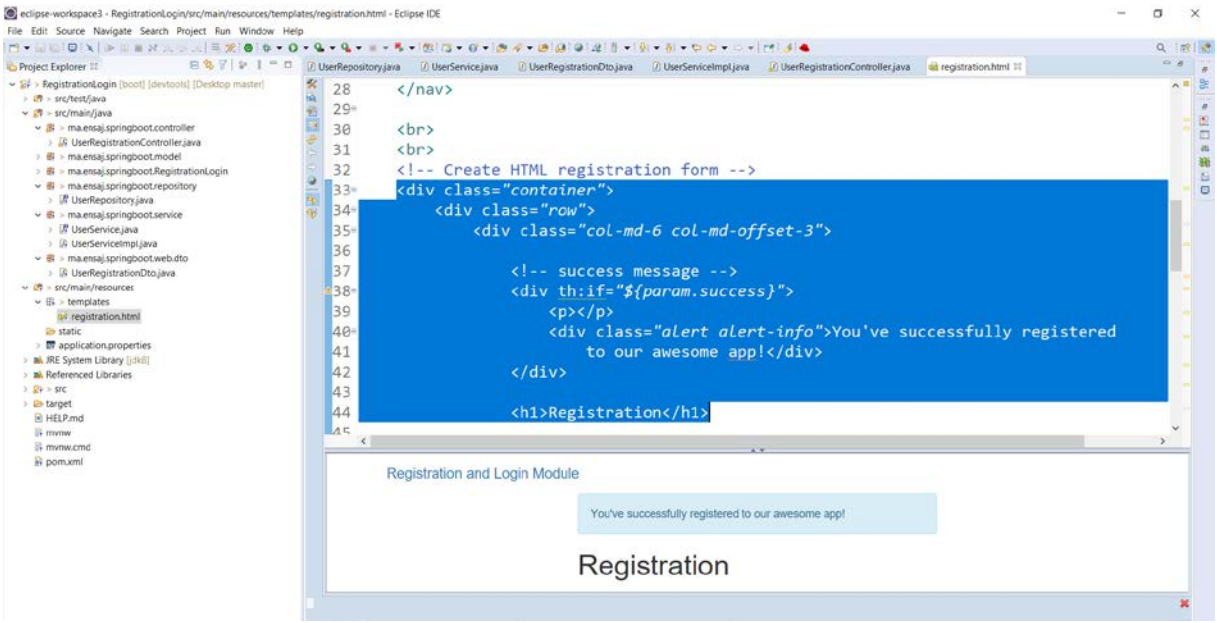


Figure 170:Show success message using thymeleaf syntax

57. Different fields for the registration form with prebuilt HTML5 validation features

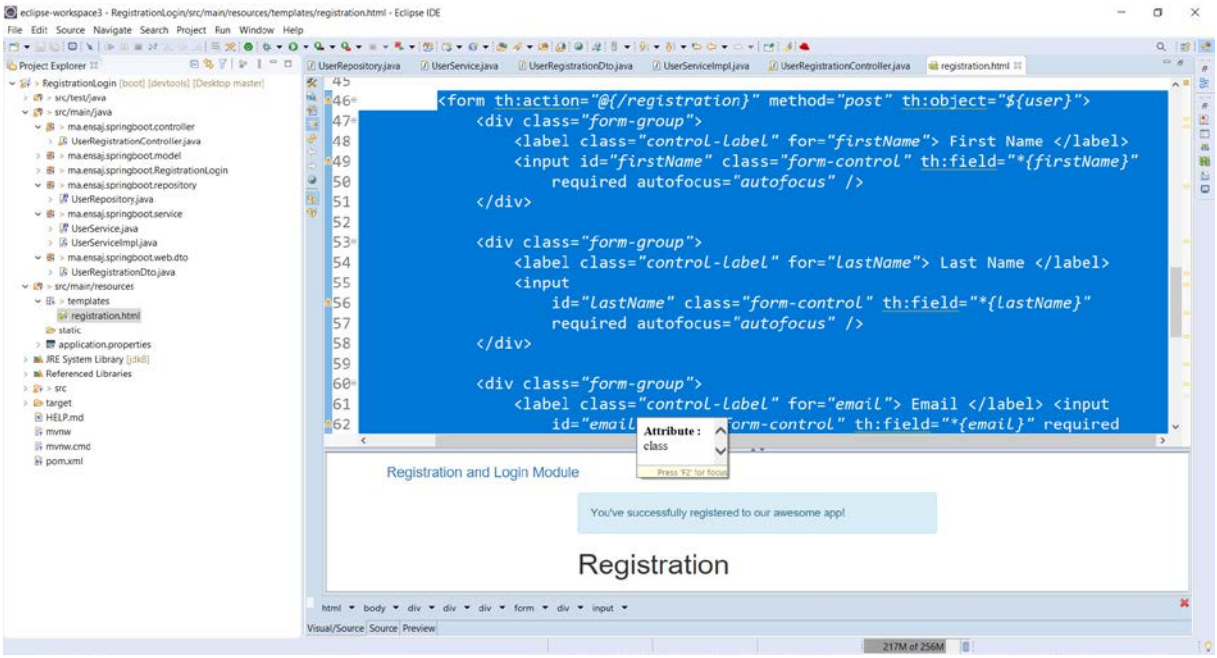


Figure 171:Different fields for the registration form with prebuilt HTML5 validation features

58. Preview of the registration form in eclipse

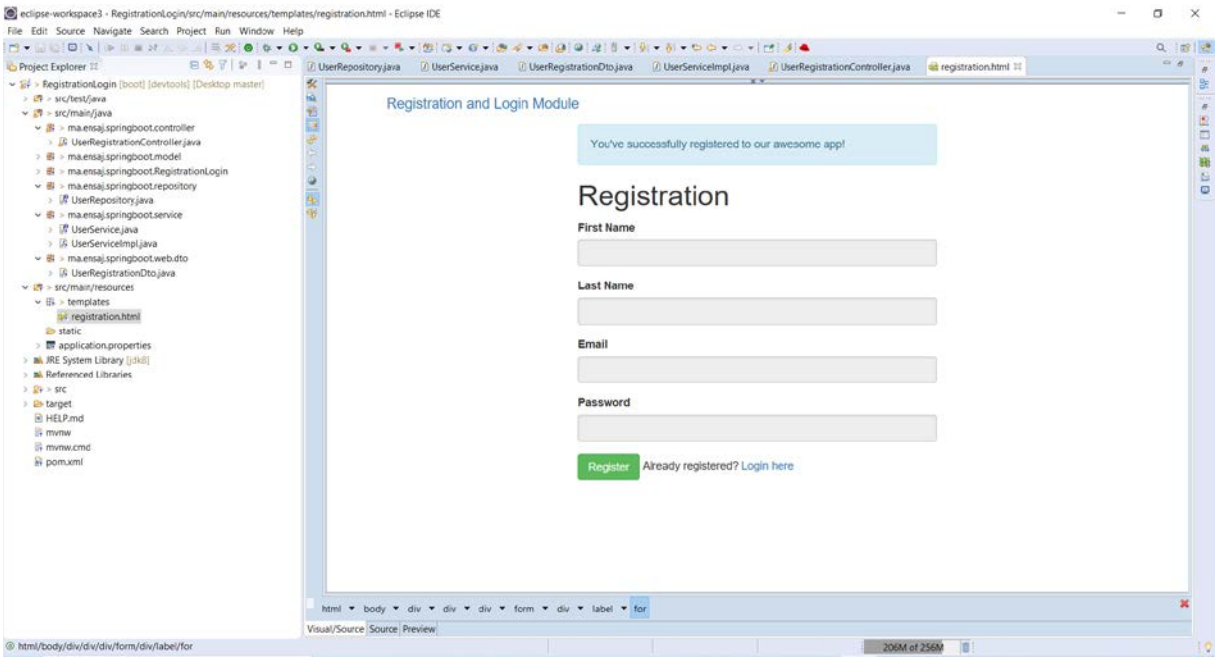


Figure 172:Preview of the registration form in eclipse

59. Run the app again

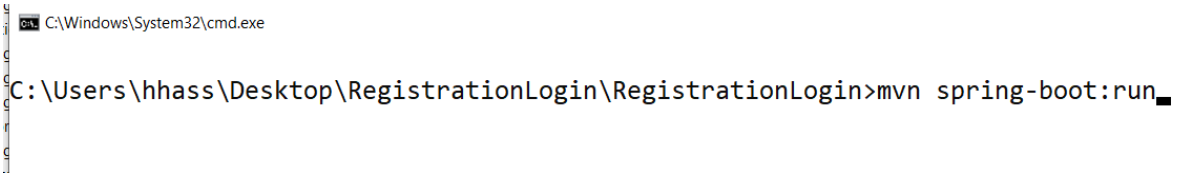


Figure 173:Run the app again

60. The registration form in the browser

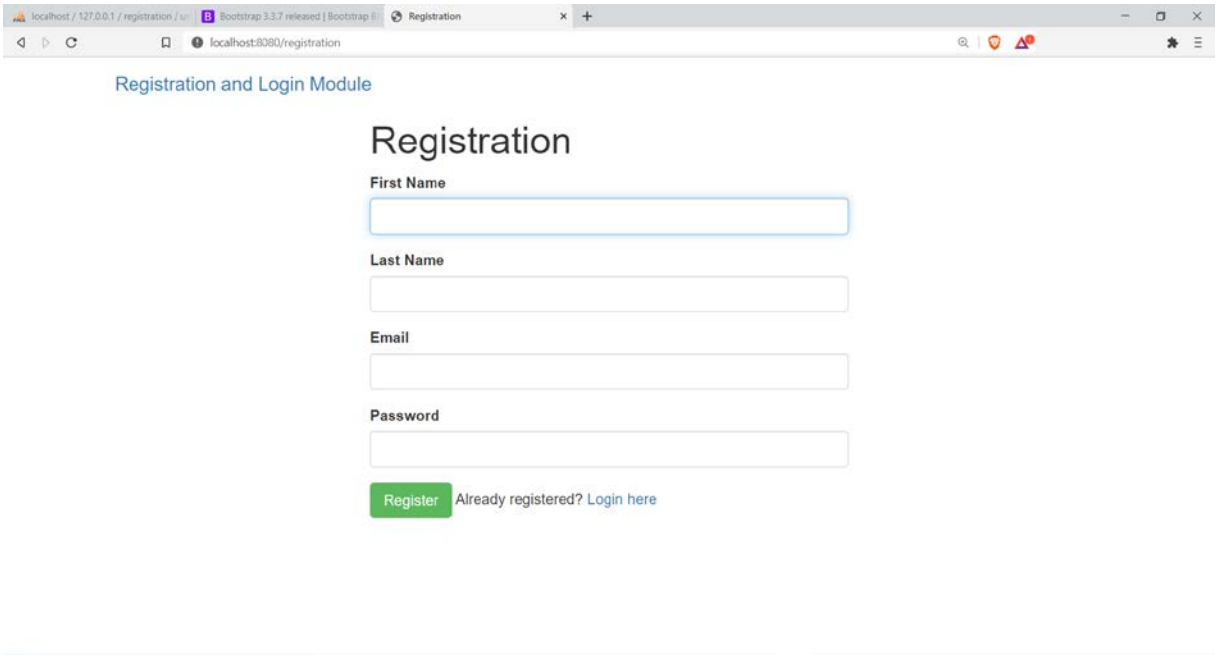


Figure 174:The registration form in the browser

61. Filling the form with valid data

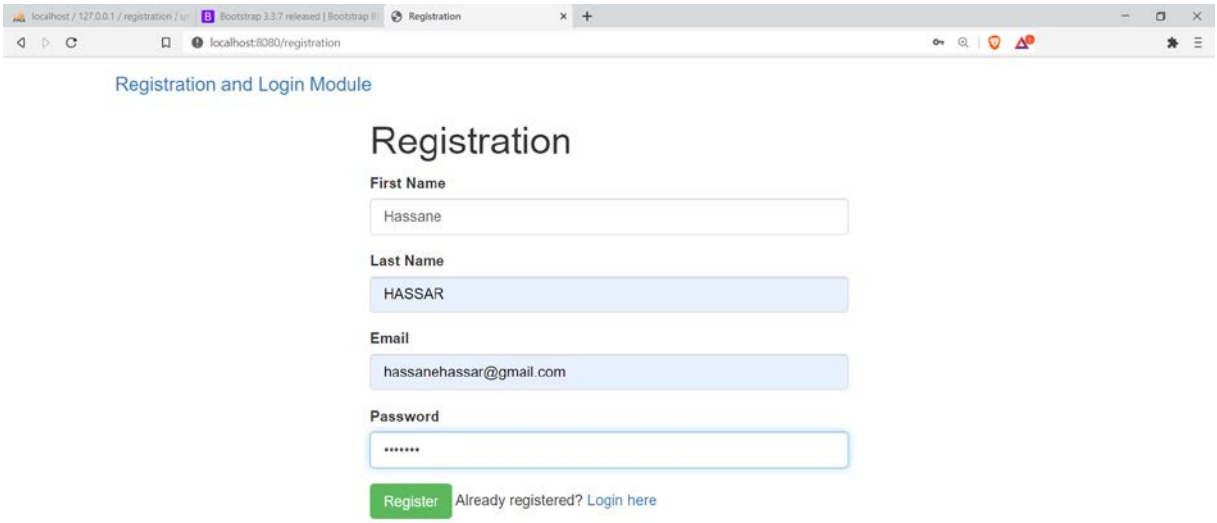


Figure 175:Filling the form with valid data

62. After Register button click the success message is displayed to the user

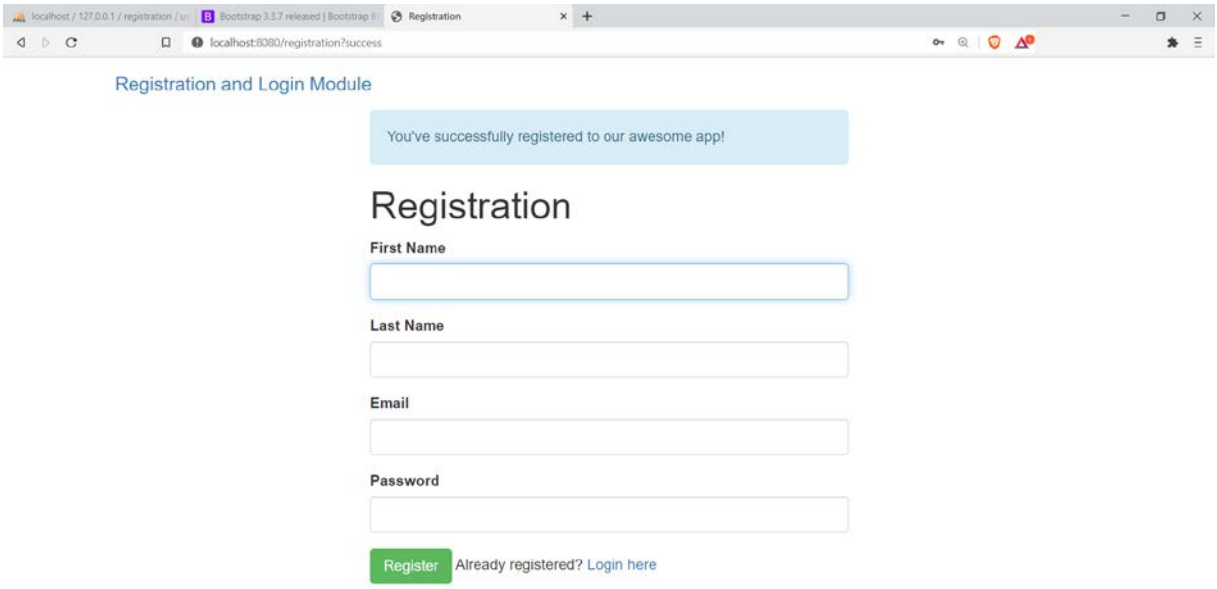


Figure 176:After Register button click the success message is displayed to the user

63. Let’s check the database if the user was added successfully

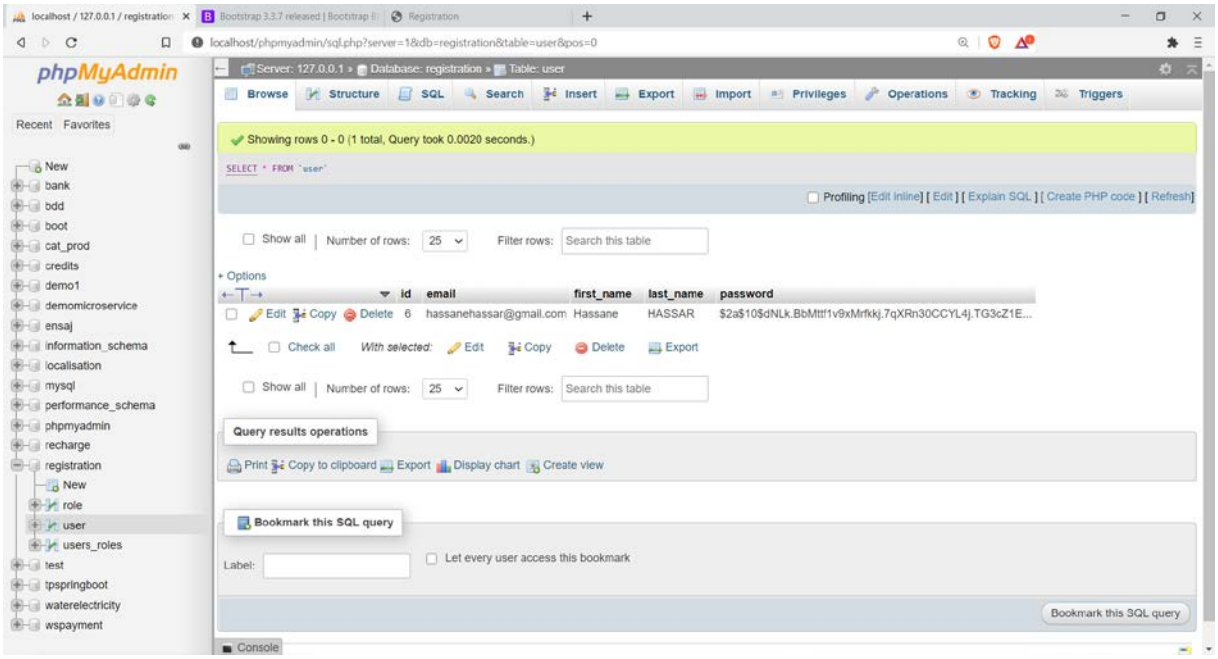


Figure 177:Let’s check the database if the user was added successfully

64. In table role the user has default role

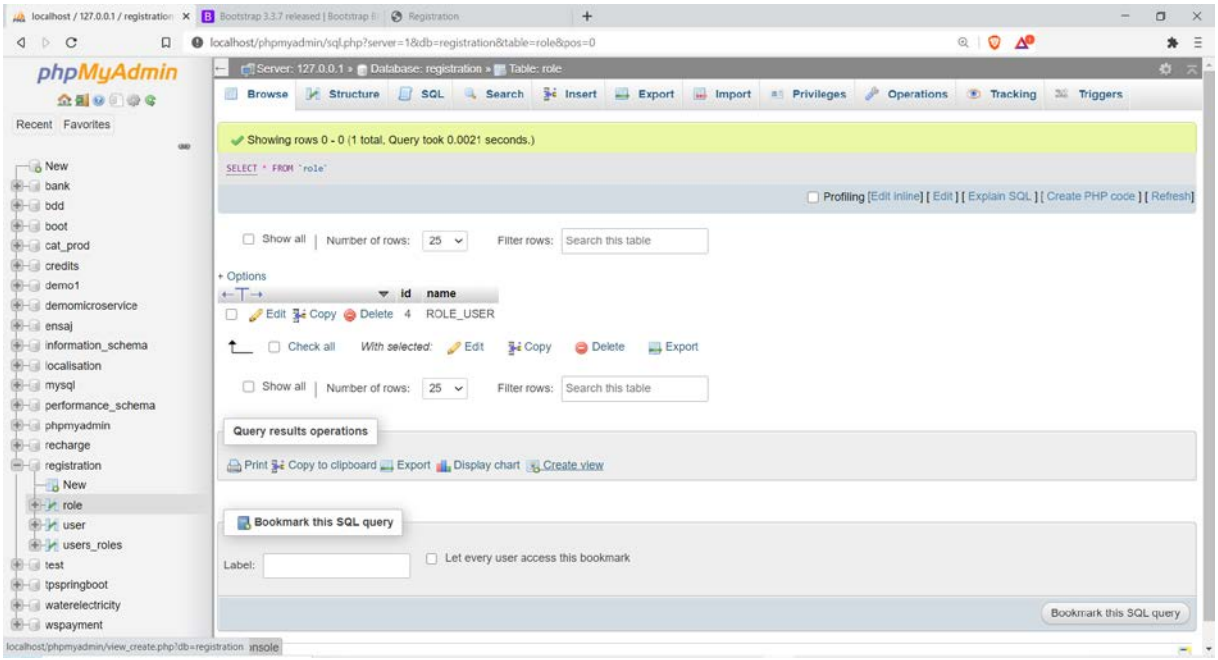


Figure 178:In table role the user has default role

65. In the association table:

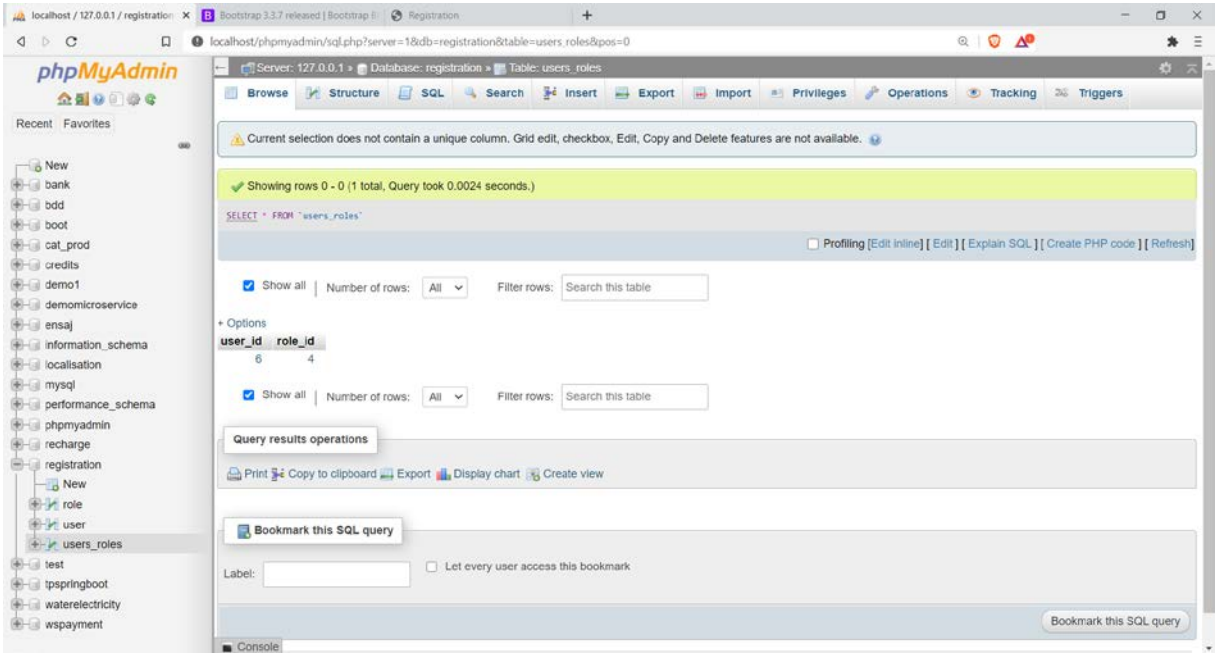


Figure 179:In the association table

66. Form validation example

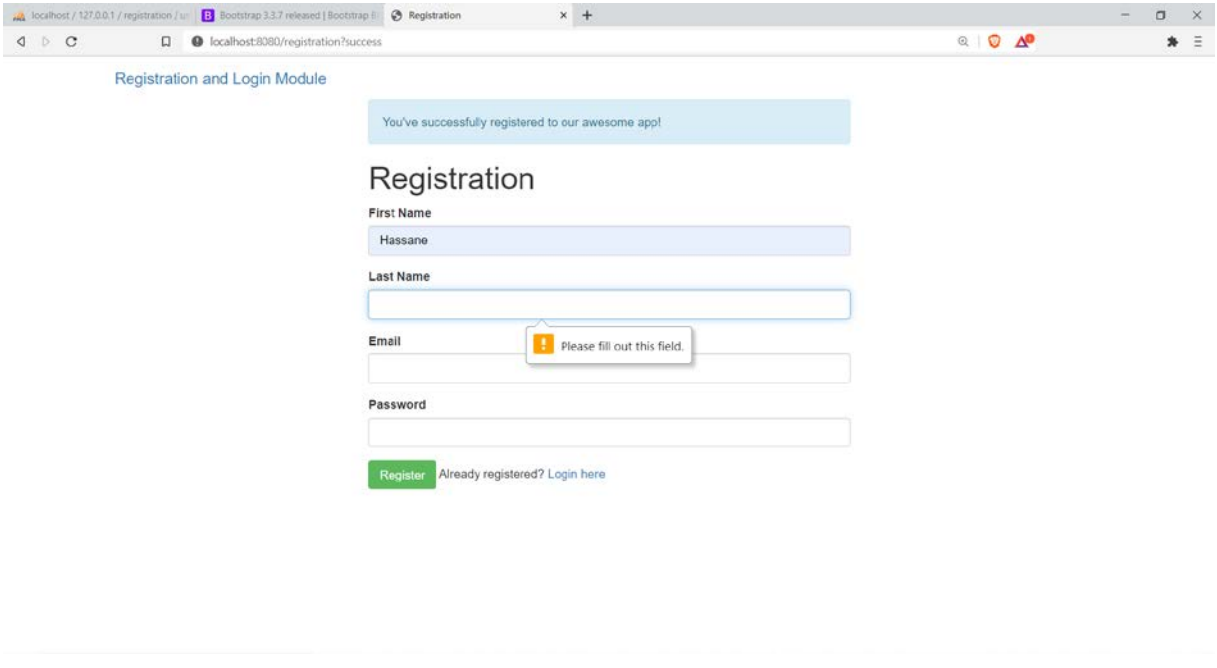


Figure 180:Form validation example

Spring Security Configuration

67. **N.B:** Spring Security offers default login page but we are going to custom our us

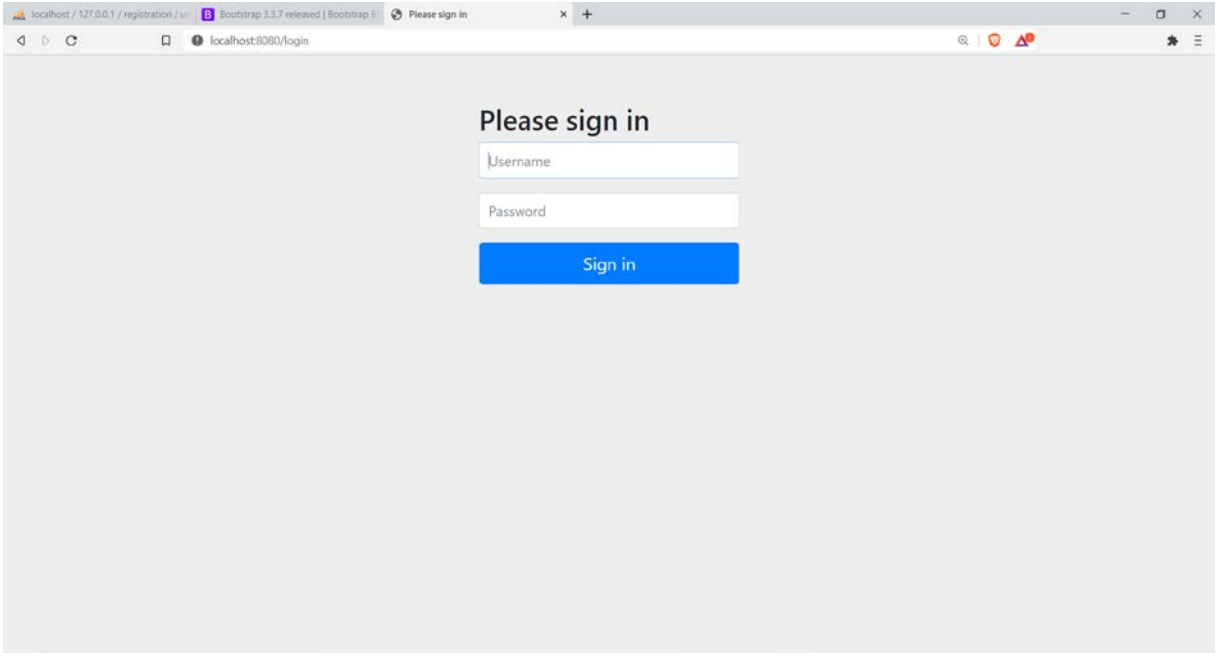


Figure 181:default login page

68. Default generated security password

```
y -> org.hibernate.type.AdaptedImmutableType@7bd70126
2021-01-12 20:39:05.575 DEBUG 13856 --- [         task-1] org.hibernate.type.BasicTypeRegistry     : Adding type registration imm_seria
lizable -> org.hibernate.type.AdaptedImmutableType@124f7c64
2021-01-12 20:39:05.669  INFO 13856 --- [         task-1] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Starting...
2021-01-12 20:39:05.783  INFO 13856 --- [  restartedMain] .s.s.UserDetailsServiceAutoConfiguration :

Using generated security password: 73b08259-a4da-4b77-b65f-4925620b8a83

2021-01-12 20:39:06.043  INFO 13856 --- [  restartedMain] o.s.s.web.DefaultSecurityFilterChain     : Creating filter chain: any request
, [org.springframework.security.web.context.request.async.WebAsyncManagerIntegrationFilter@2253194a, org.springframework.security.web.c
ontext.SecurityContextPersistenceFilter@3f260a09, org.springframework.security.web.header.HeaderWriterFilter@3399b743, org.springframew
ork.security.web.csrf.CsrfFilter@49f310f2, org.springframework.security.web.authentication.logout.LogoutFilter@3b8b9ec2, org.springfram
ework.security.web.authentication.UsernamePasswordAuthenticationFilter@445093de, org.springframework.security.web.authentication.ui.Def
aultLoginPageGeneratingFilter@272e16bc, org.springframework.security.web.authentication.ui.DefaultLogoutPageGeneratingFilter@4b8d607e,
org.springframework.security.web.authentication.www.BasicAuthenticationFilter@1a9db6de, org.springframework.security.web.savedrequest.R
equestCacheAwareFilter@3e5410b9, org.springframework.security.web.servletapi.SecurityContextHolderAwareRequestFilter@61565825, org.spri
ngframework.security.web.authentication.AnonymousAuthenticationFilter@368d1cf7, org.springframework.security.web.session.SessionManagem
entFilter@4734d426, org.springframework.security.web.access.ExceptionTranslationFilter@48a06e5e, org.springframework.security.web.acces
s.intercept.FilterSecurityInterceptor@6abfaee5]
2021-01-12 20:39:06.045  INFO 13856 --- [         task-1] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Start completed.
2021-01-12 20:39:06.075  INFO 13856 --- [         task-1] org.hibernate.dialect.Dialect            : HHH000400: Using dialect: org.hibe
rnate.dialect.MySQL5InnoDBDialect
2021-01-12 20:39:06.102  INFO 13856 --- [  restartedMain] o.s.b.d.a.OptionalLiveReloadServer       : LiveReload server is running on po
rt 35729
2021-01-12 20:39:06.165  INFO 13856 --- [  restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8080 (h
ttp) with context path ''
2021-01-12 20:39:06.171  INFO 13856 --- [  restartedMain] DeferredRepositoryInitializationListener : Triggering deferred initialization
 of Spring Data repositoriesů
2021-01-12 20:39:06.172  INFO 13856 --- [  restartedMain] DeferredRepositoryInitializationListener : Spring Data repositories initializ
ed!
2021-01-12 20:39:06.185 DEBUG 13856 --- [         task-1] o.h.type.spi.TypeConfiguration$Scope     : Scoping TypeConfiguration [org.hib
ernate.type.spi.TypeConfiguration@71601c82] to MetadataBuildingContext [org.hibernate.boot.internal.MetadataBuildingContextRootImpl@d5a
ae7e]
```

Figure 182:Default generated security password

Here we go

69. Create **SecurityConfiguration** class under the sub package **config** of the main package

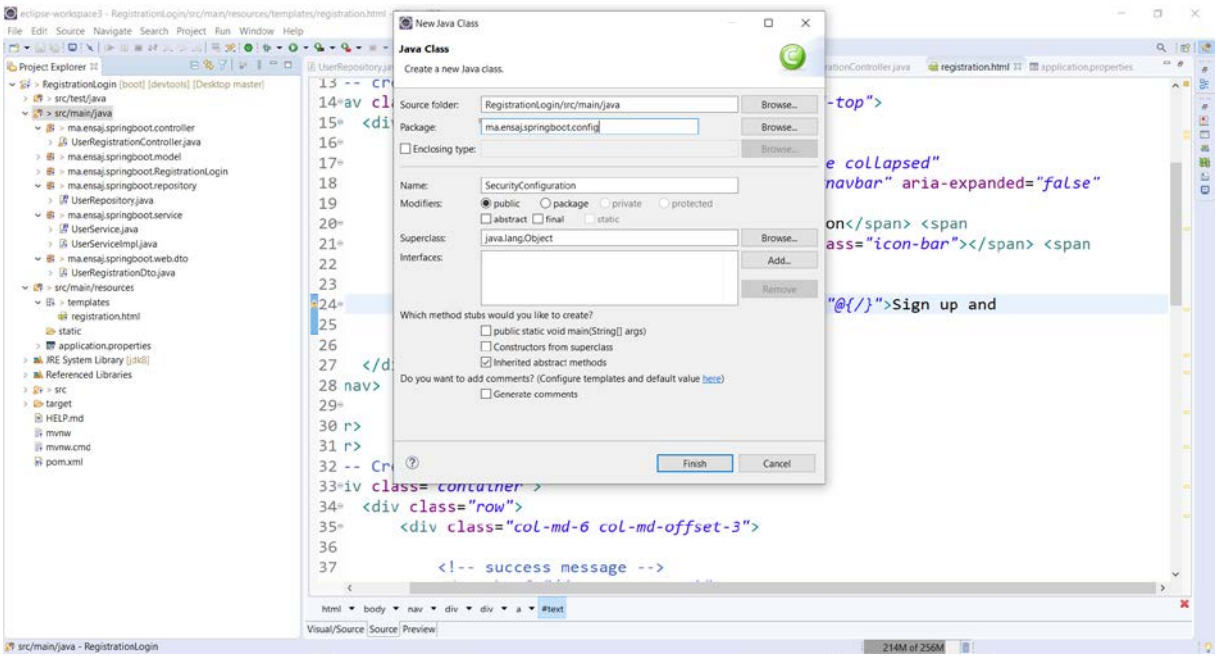


Figure 183:Create SecurityConfiguration class under the sub package config of the main package

70. Notice : The presence of the security dependency in the pom.xml

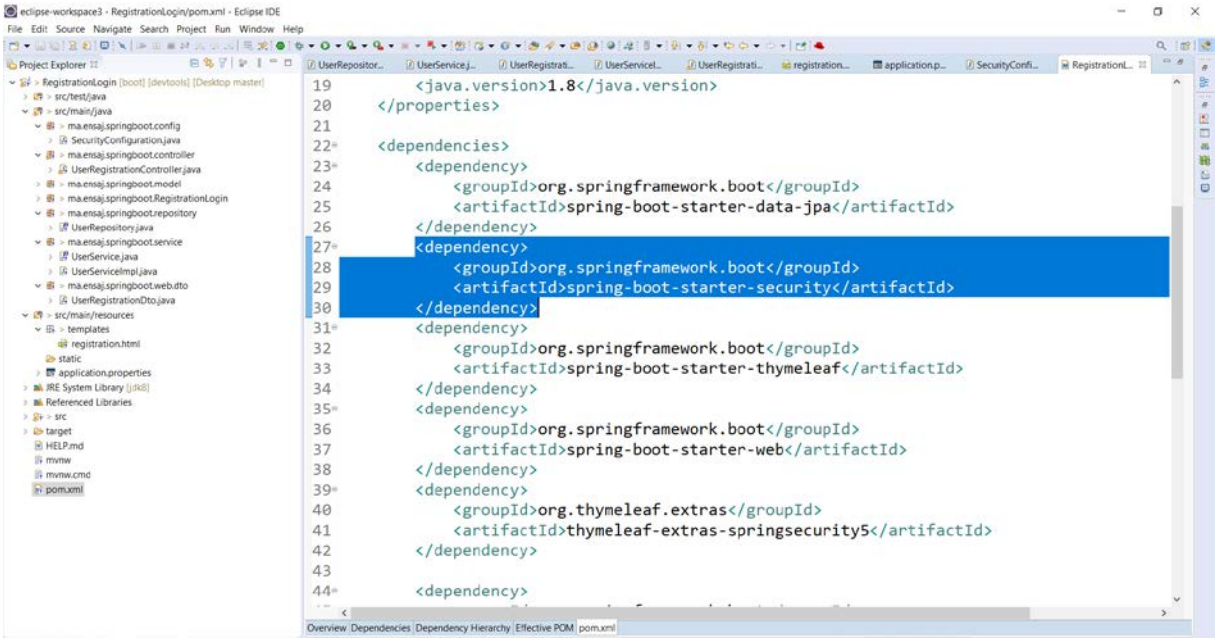


Figure 184:Notice : The presence of the security dependency in the pom.xml

71. Adding annotations **@Configuration and @EnableWebSecurity**

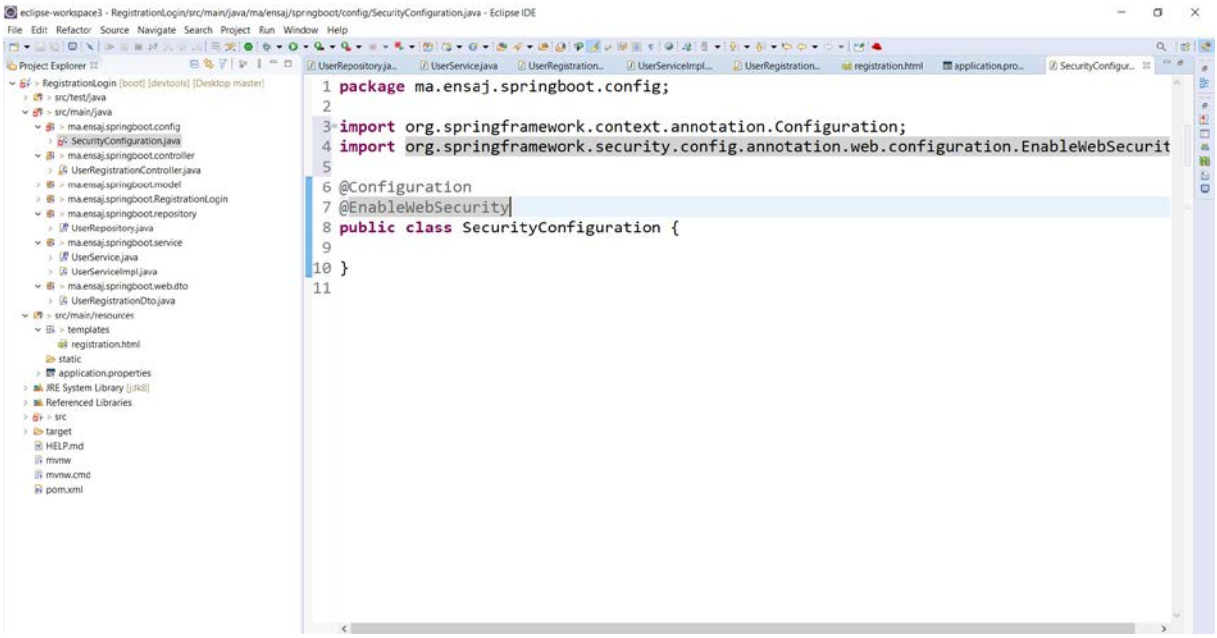
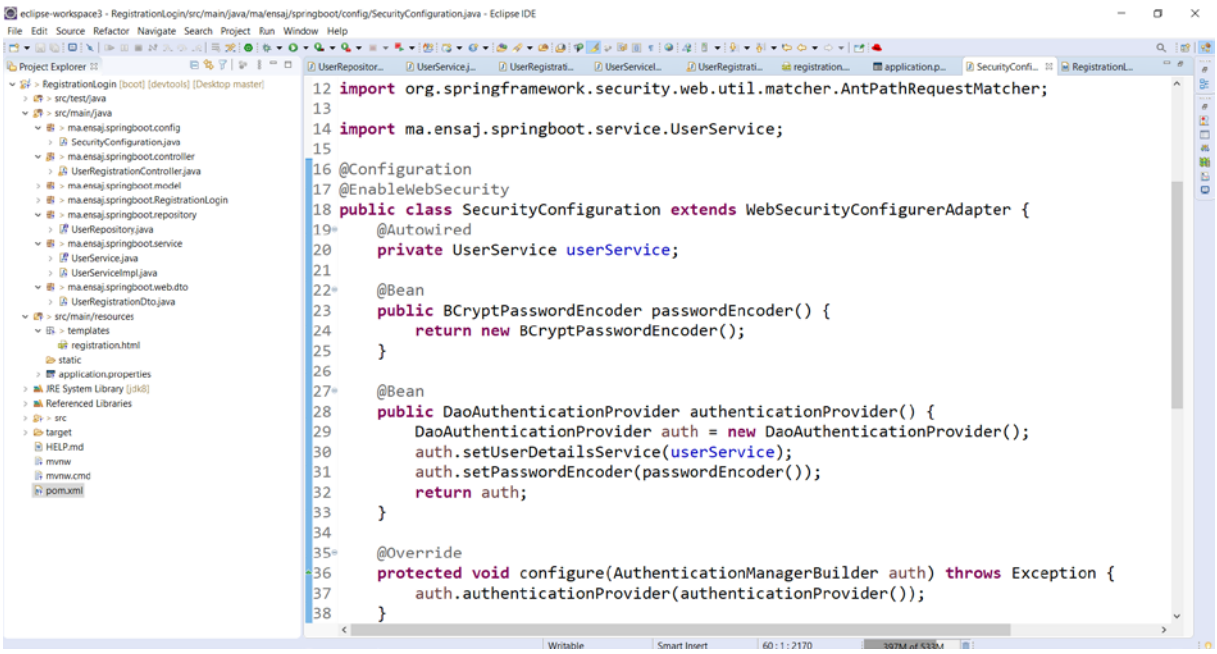


Figure 185:Adding annotations @Configuration and @EnableWebSecurity

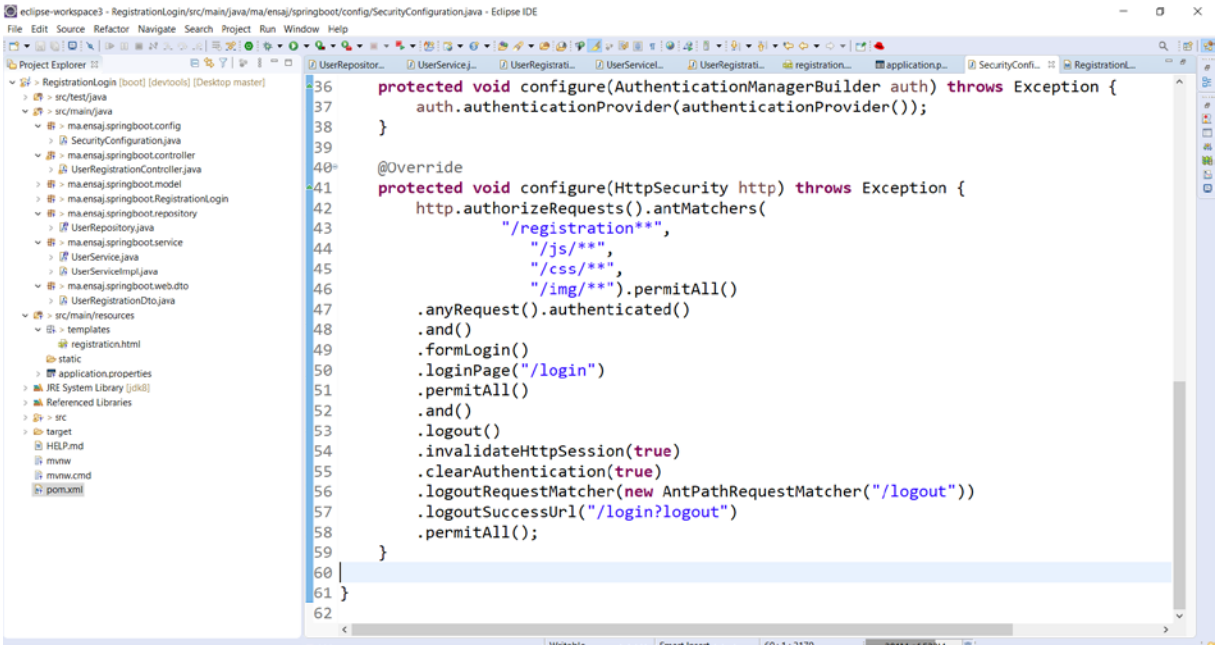
72. Add **@Bean** annotation to tell JPA to create an entity and @Autowired for the reference userService.



```
12 import org.springframework.security.web.util.matcher.AntPathRequestMatcher;
13
14 import ma.ensaj.springboot.service.UserService;
15
16 @Configuration
17 @EnableWebSecurity
18 public class SecurityConfiguration extends WebSecurityConfigurerAdapter {
19     @Autowired
20     private UserService userService;
21
22     @Bean
23     public BCryptPasswordEncoder passwordEncoder() {
24         return new BCryptPasswordEncoder();
25     }
26
27     @Bean
28     public DaoAuthenticationProvider authenticationProvider() {
29         DaoAuthenticationProvider auth = new DaoAuthenticationProvider();
30         auth.setUserDetailsService(userService);
31         auth.setPasswordEncoder(passwordEncoder());
32         return auth;
33     }
34
35     @Override
36     protected void configure(AuthenticationManagerBuilder auth) throws Exception {
37         auth.authenticationProvider(authenticationProvider());
38     }
```

Figure 186:Add @Bean annotation to tell JPA to create an entity and @Autowired for the reference userService.

73. In the configure method we add all permissions pages such as “/login” and “logout”.



```
36     protected void configure(AuthenticationManagerBuilder auth) throws Exception {
37         auth.authenticationProvider(authenticationProvider());
38     }
39
40     @Override
41     protected void configure(HttpSecurity http) throws Exception {
42         http.authorizeRequests().antMatchers(
43                 "/registration**",
44                    "/js/**",
45                    "/css/**",
46                    "/img/**").permitAll()
47         .anyRequest().authenticated()
48         .and()
49         .formLogin()
50         .loginPage("/login")
51         .permitAll()
52         .and()
53         .logout()
54         .invalidateHttpSession(true)
55         .clearAuthentication(true)
56         .logoutRequestMatcher(new AntPathRequestMatcher("/logout"))
57         .logoutSuccessUrl("/login?logout")
58         .permitAll();
59     }
60
61 }
62
```

Figure 187:In the configure method we add all permissions pages such as “/login” and “logout”.

Backend for Login feature

74. Notice :The **UserService** interface that we have already declare it extends the class **UserDetailsService** that defined the method **loadUserByUsername**

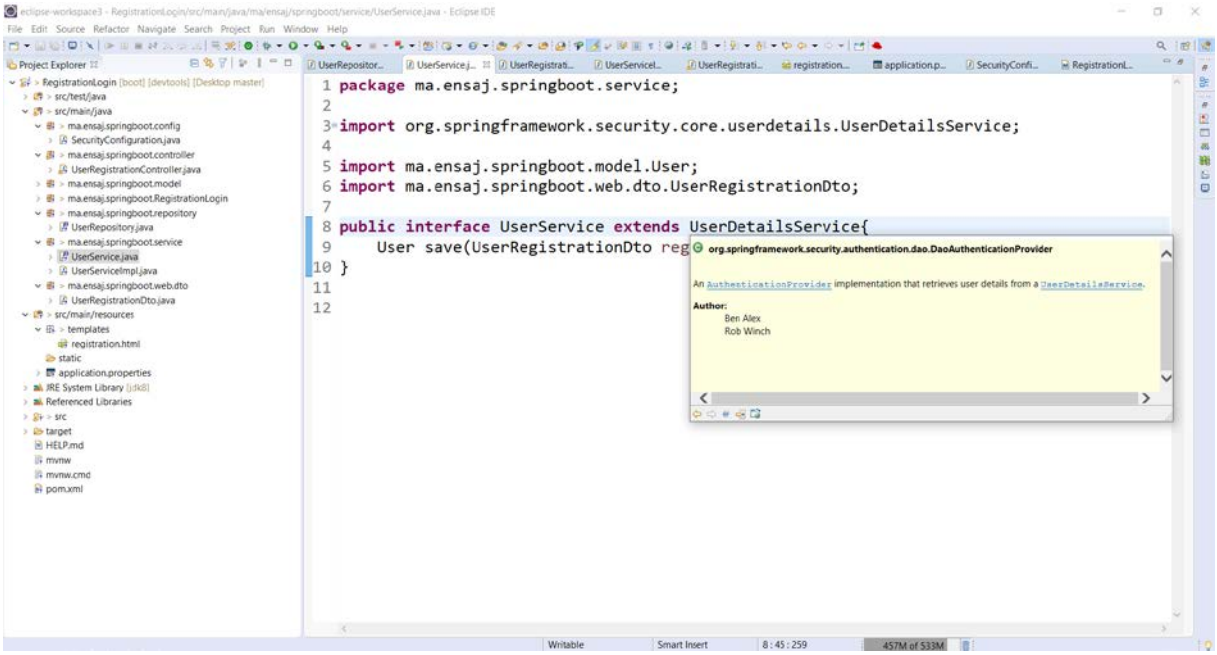


Figure 188:UserDetailsService

75. Let's implement this method **loadUserByUsername**

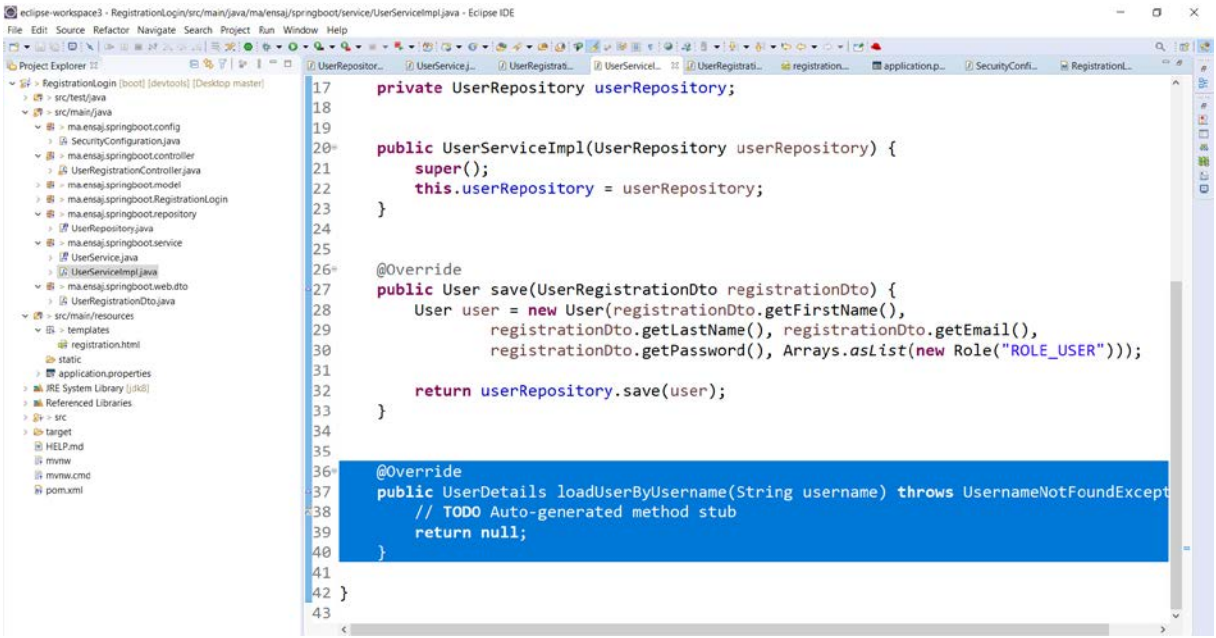


Figure 189:Let's implement this method loadUserByUsername

76. The algorithm is simple. We retrieve the user from the database by its username. If it doesn't exist, we throw a new exception otherwise we return a new user with email and password from user details of spring boot security

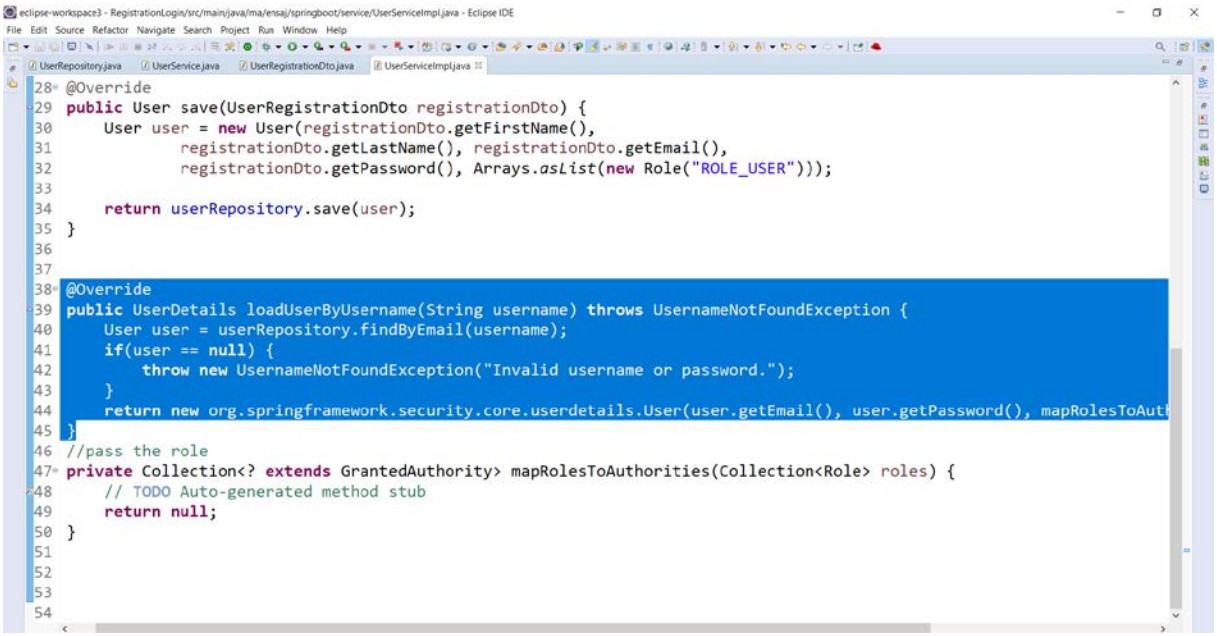
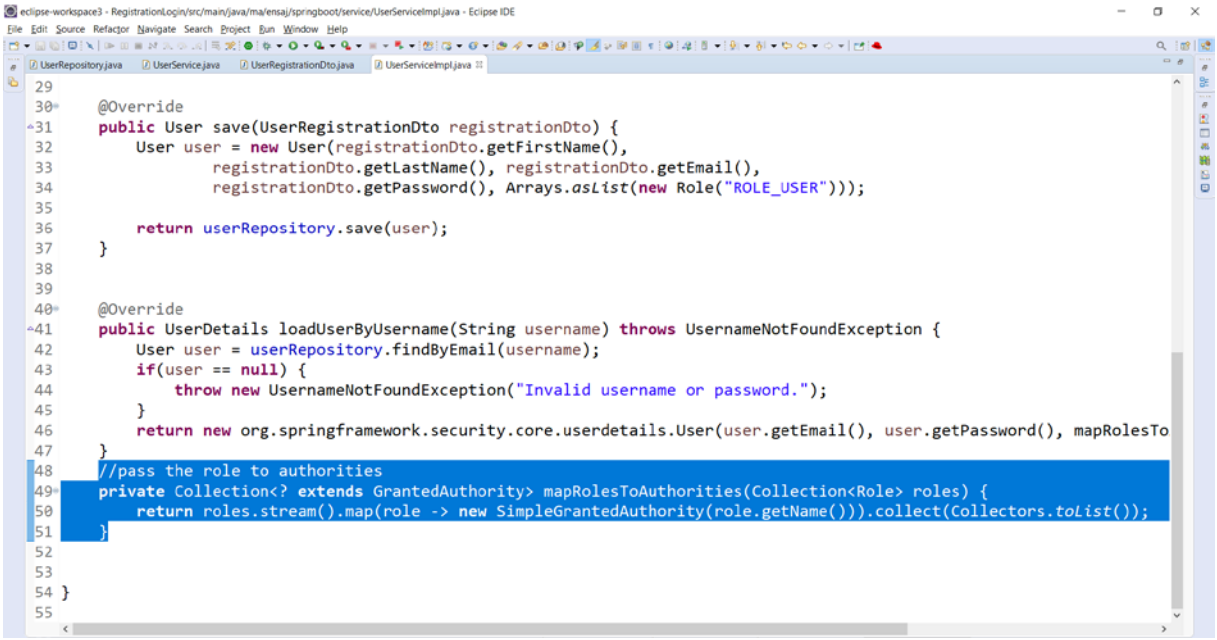


Figure 190:loadUserByUsername Implementation

77. **mapRolesToAuthorities** method



```
    @Override
    public User save(UserRegistrationDto registrationDto) {
        User user = new User(registrationDto.getFirstName(),
                registrationDto.getLastName(), registrationDto.getEmail(),
                registrationDto.getPassword(), Arrays.asList(new Role("ROLE_USER")));

        return userRepository.save(user);
    }


    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
        User user = userRepository.findByEmail(username);
        if(user == null) {
            throw new UsernameNotFoundException("Invalid username or password.");
        }
        return new org.springframework.security.core.userdetails.User(user.getEmail(), user.getPassword(), mapRolesTo
    }
    //pass the role to authorities
    private Collection<? extends GrantedAuthority> mapRolesToAuthorities(Collection<Role> roles) {
        return roles.stream().map(role -> new SimpleGrantedAuthority(role.getName())).collect(Collectors.toList());
    }
}
```

Figure 191:mapRolesToAuthorities method

78. Creation of the **MainController** class under the **web.dto** sub package

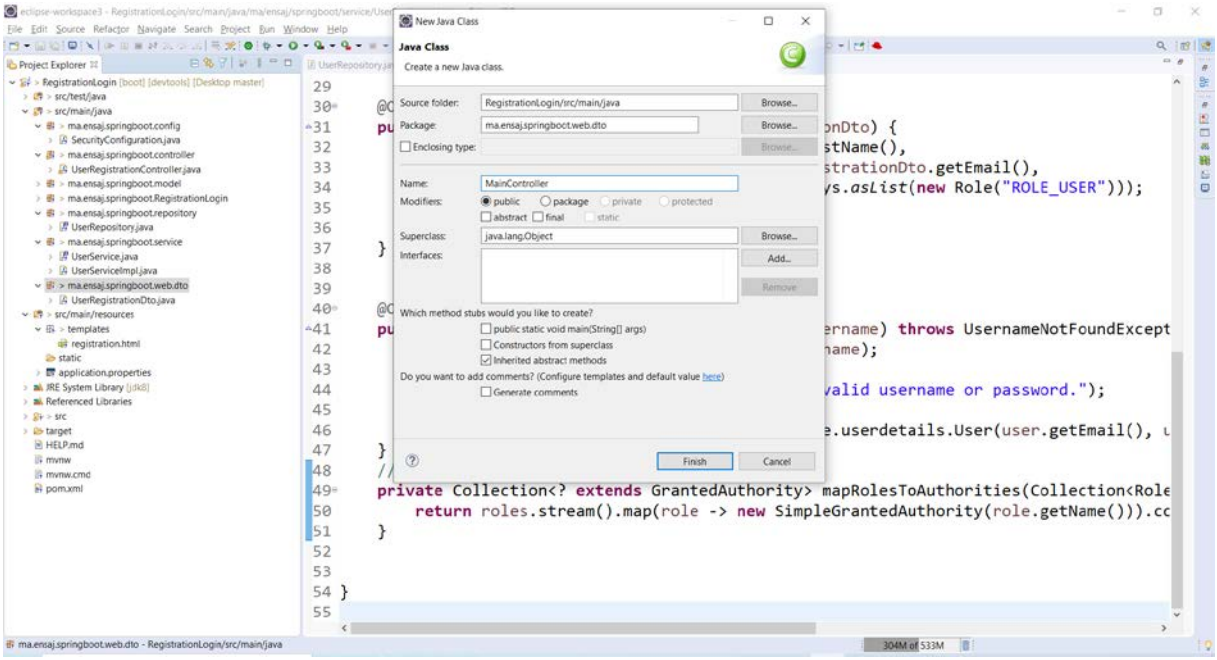


Figure 192:Creation of the MainController class under the web.dto sub package

79. Add **login** method with **@GetMapping** annotation that return the **login page** that we will develop later

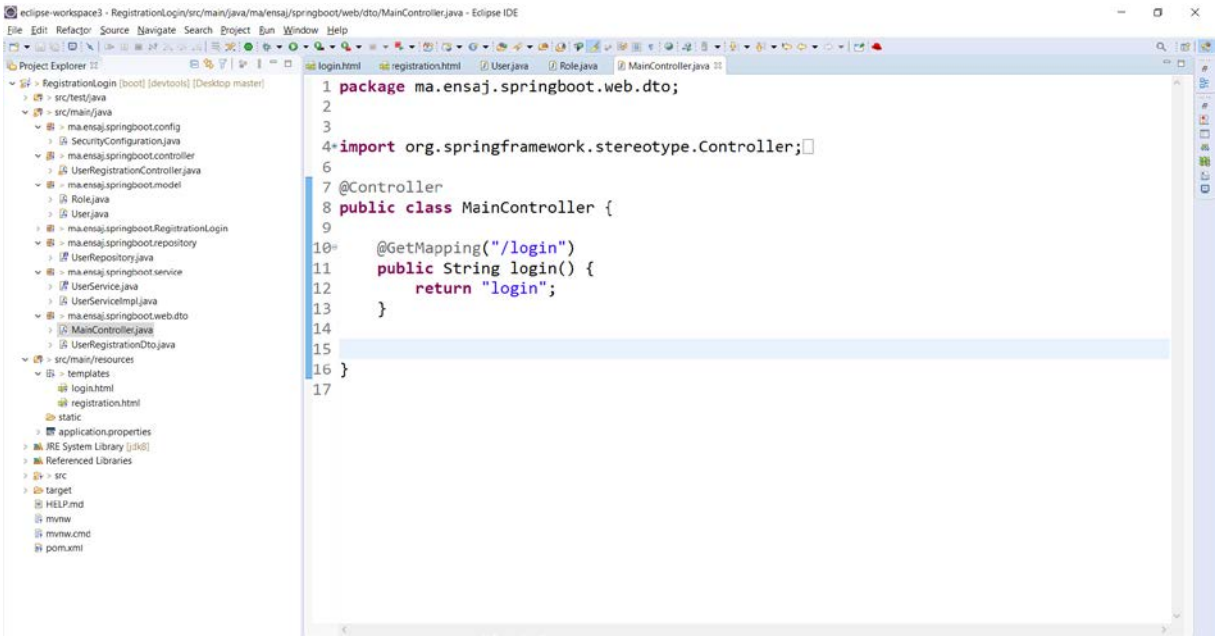


Figure 193:Add login method with @GetMapping annotation that return the login page that we will develop later

80. Add **BcryptPasswordEncoder** reference to the **UserServiceImpl** class

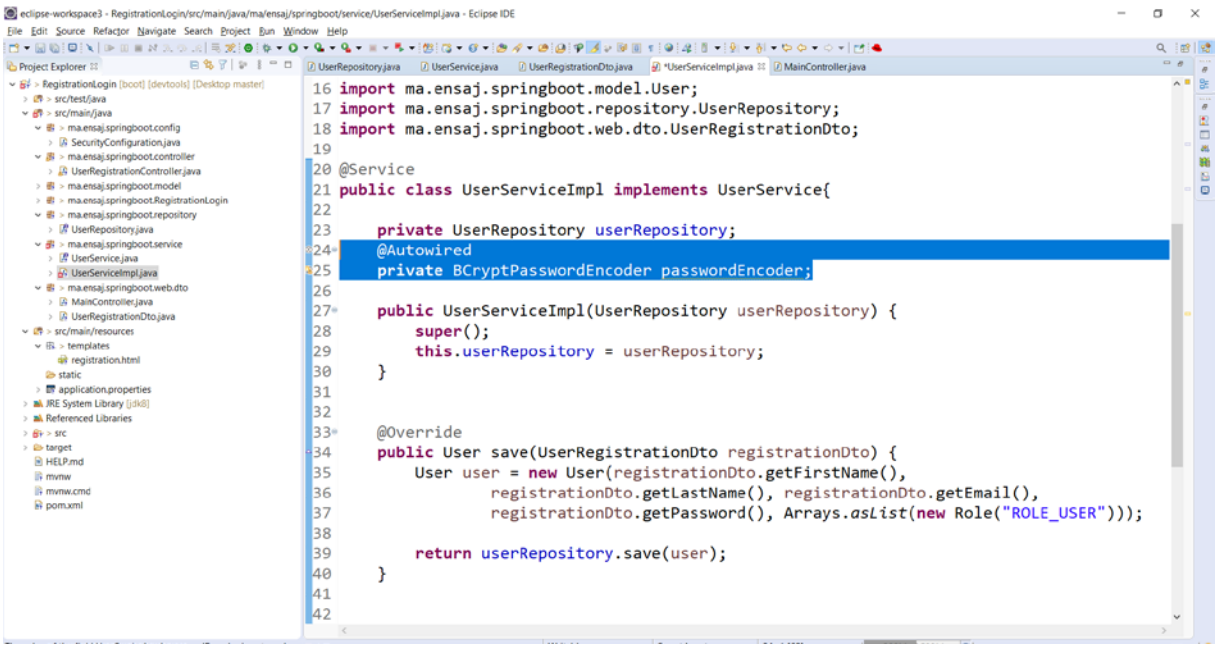


Figure 194:Add BcryptPasswordEncoder reference to the UserServiceImpl class

81. Using encode() method in order to encode/crypt the password

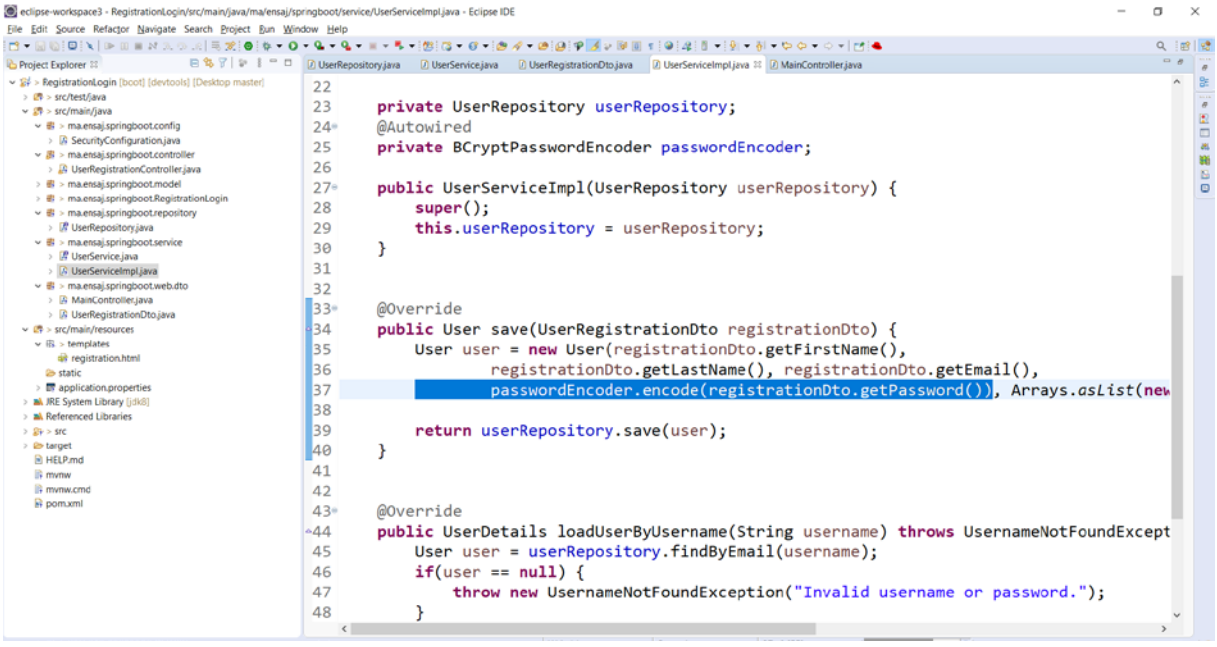


Figure 195:Using encode() method in order to encode/crypt the password

Frontend for Login feature

82. Create new Html file called login as we did for registration.

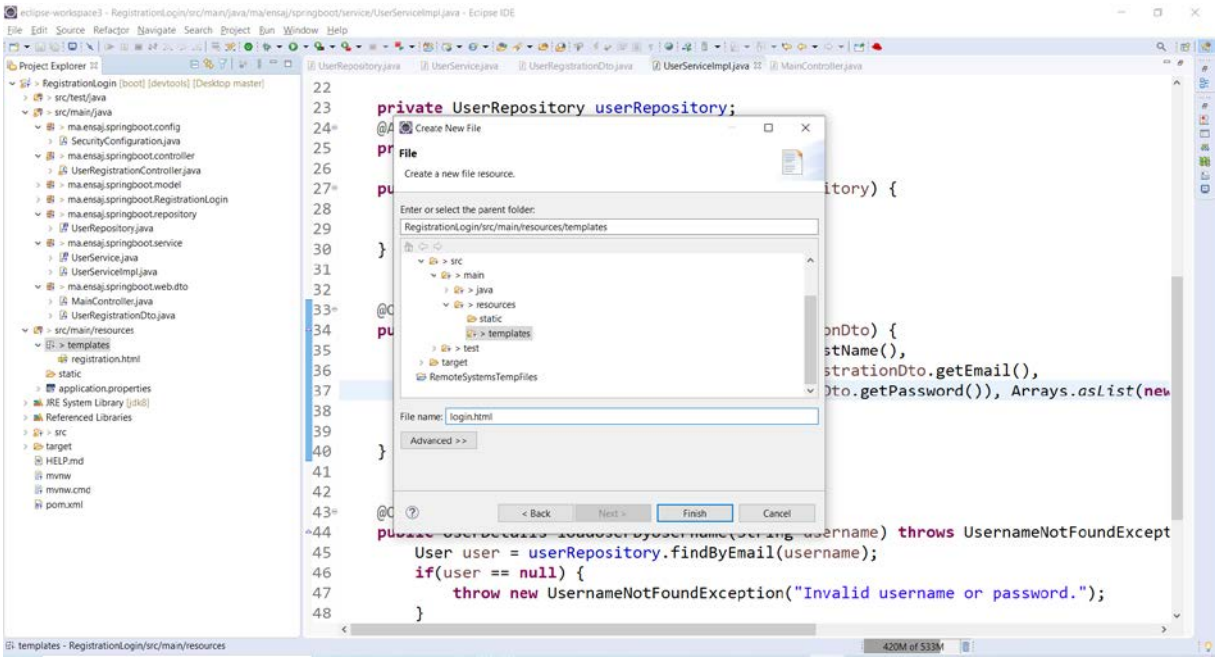


Figure 196:Create new Html file called login as we did for registration.

83. Nav bar navigation code

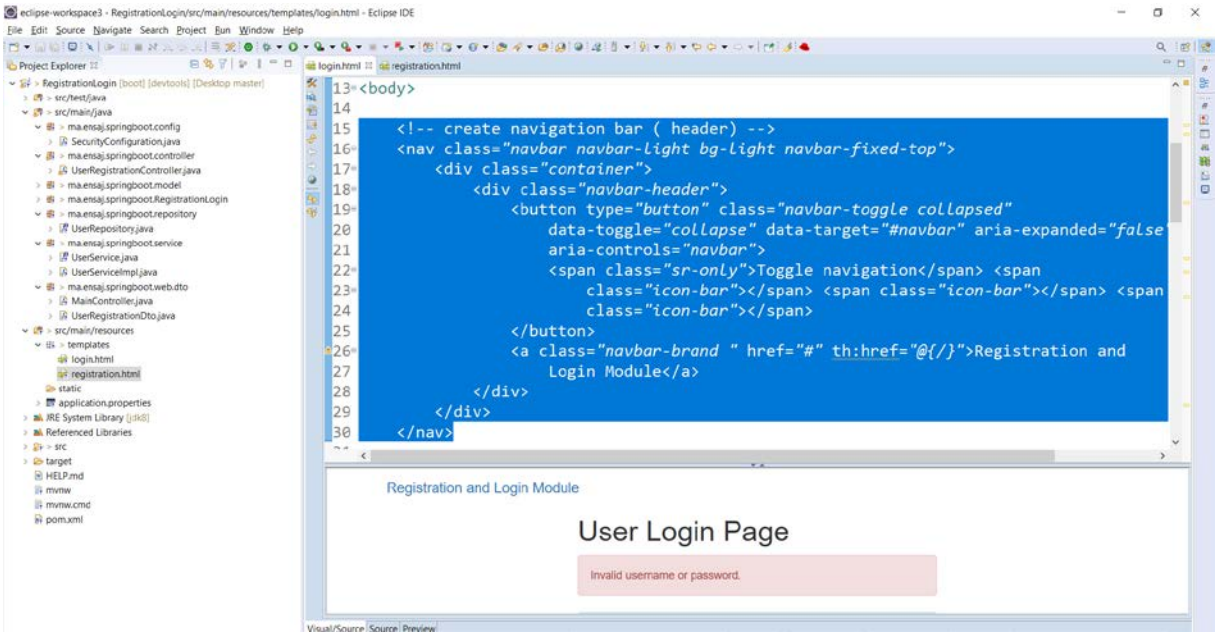


Figure 197:Nav bar navigation code

84. User Login form we use the post method and as action the view /login

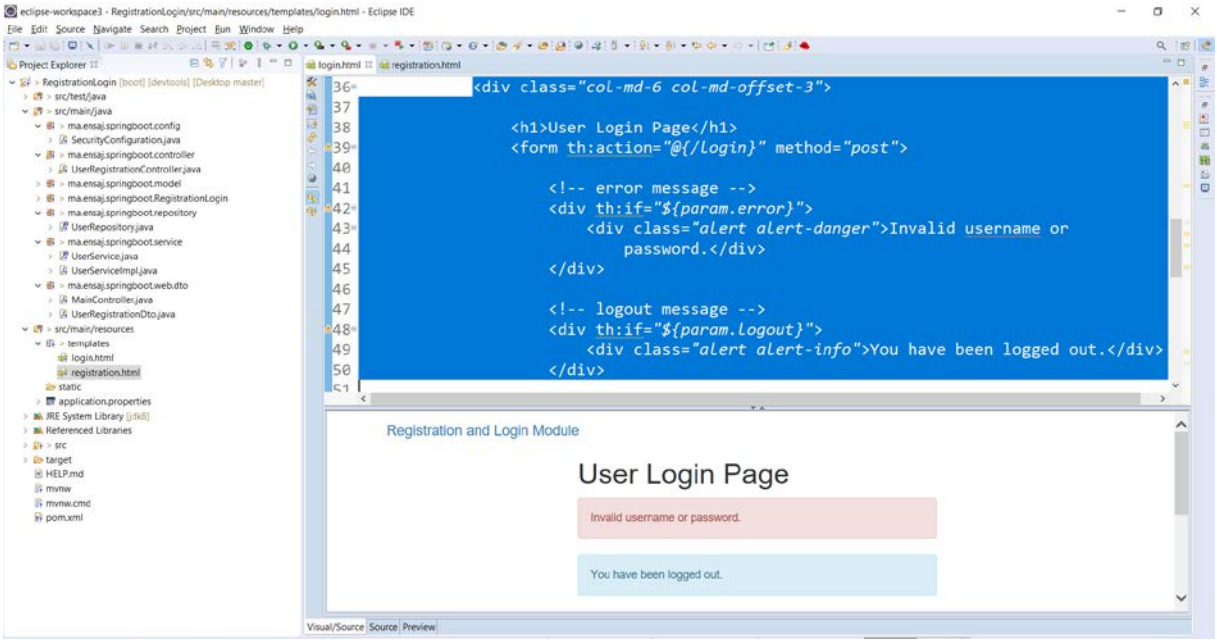


Figure 198:User Login form we use the post method and as action the view /login

85. Continuation code of user login form(username and password inputs ,and the login button)

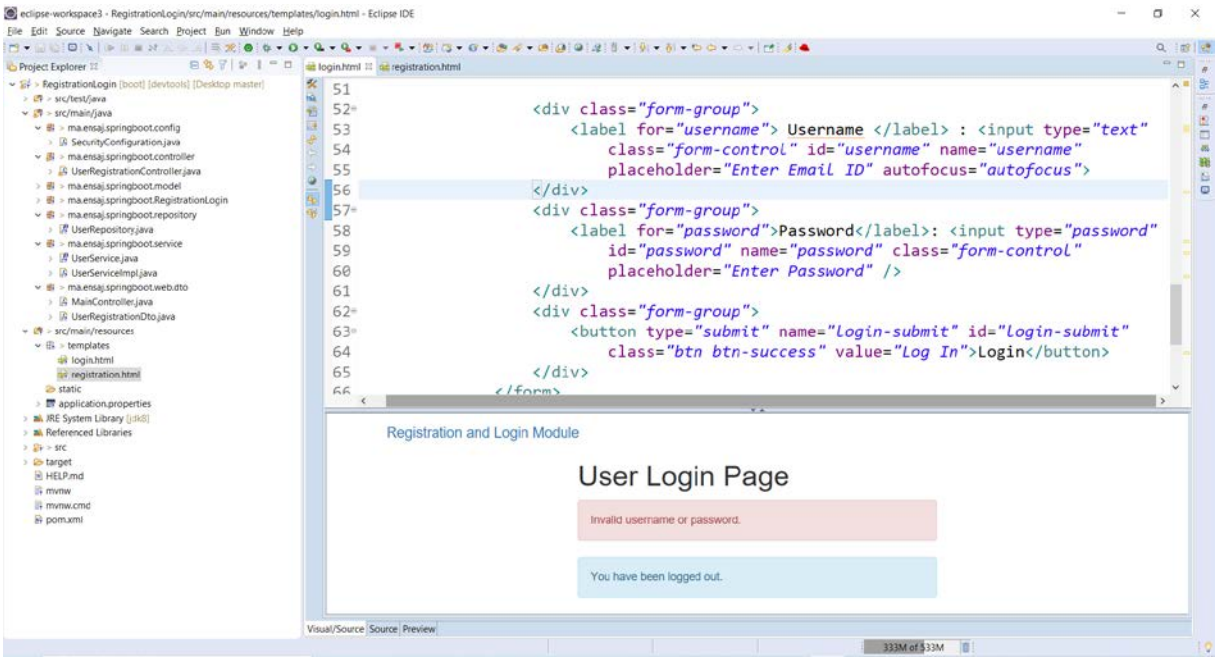


Figure 199:Continuation code of user login form(username and password inputs ,and the login button)

86. Add a link to the registration page

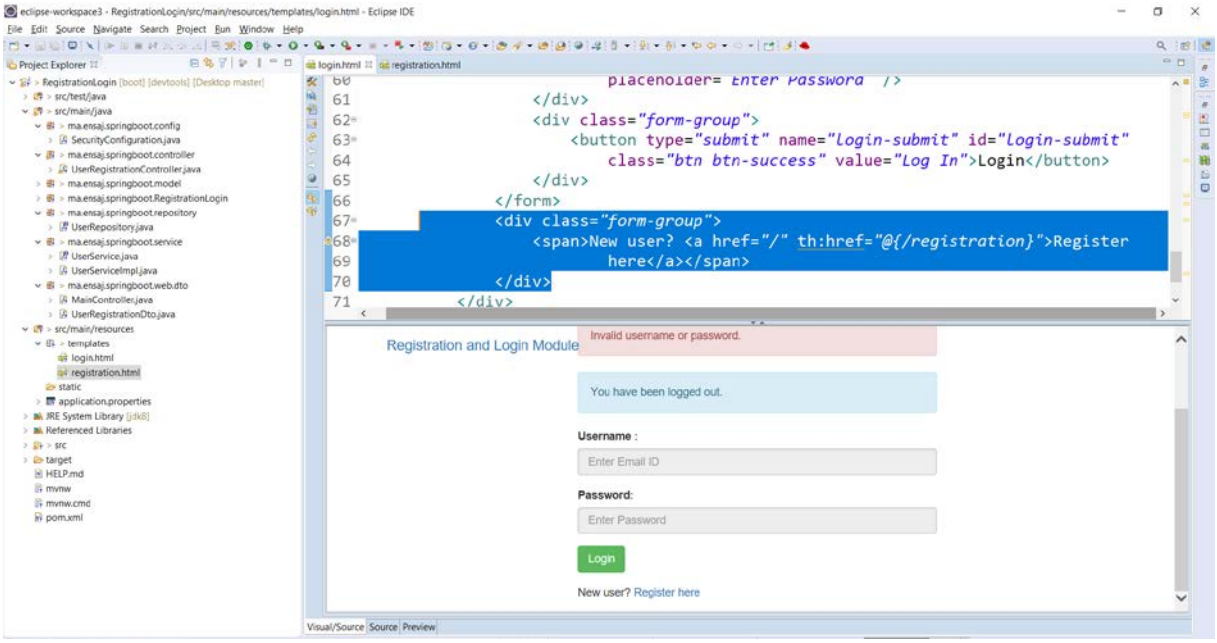


Figure 200:Add a link to the registration page

87. The preview page of login page in eclipse

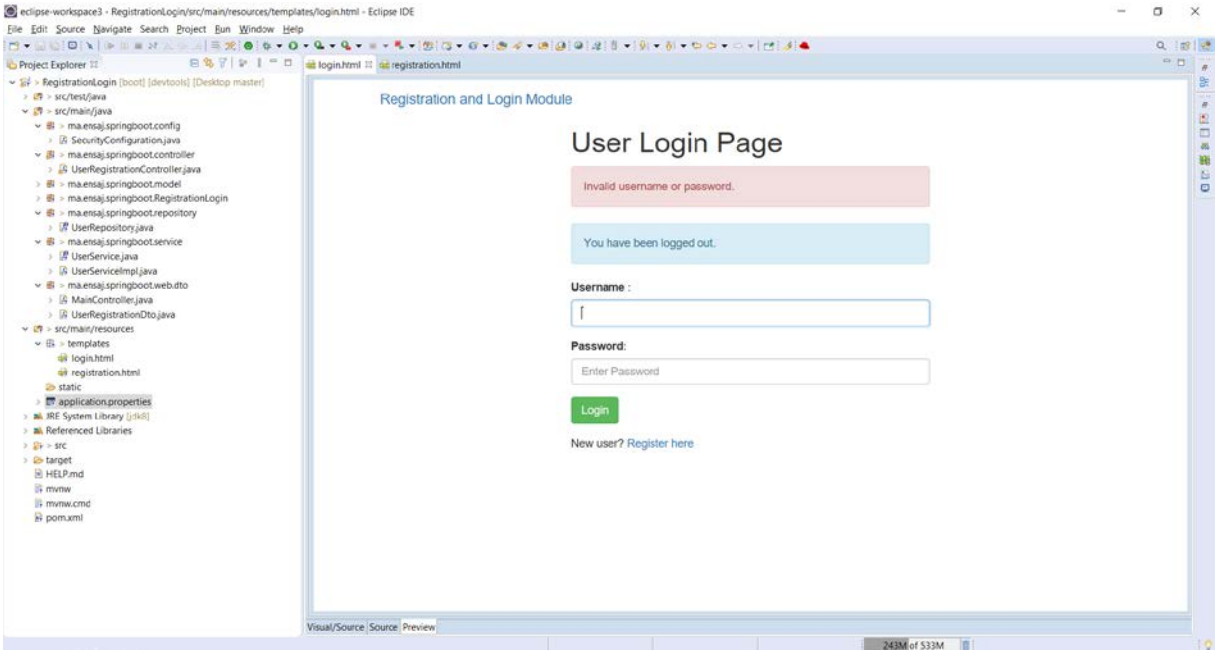


Figure 201:The preview page of login page in eclipse

88. Run the app again to test if the login feature is working well

```
C:\Windows\System32\cmd.exe

C:\Users\hhass\Desktop\RegistrationLogin\RegistrationLogin>mvn spring-boot:run
```

Figure 202:Run the app again to test if the login feature is working well

89. Login Page in the browser

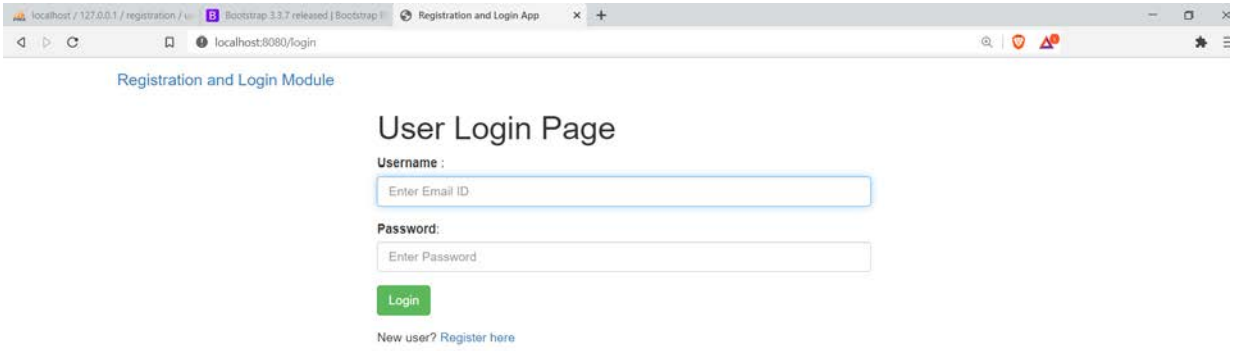


Figure 203:Login Page in the browser

90. Register new user

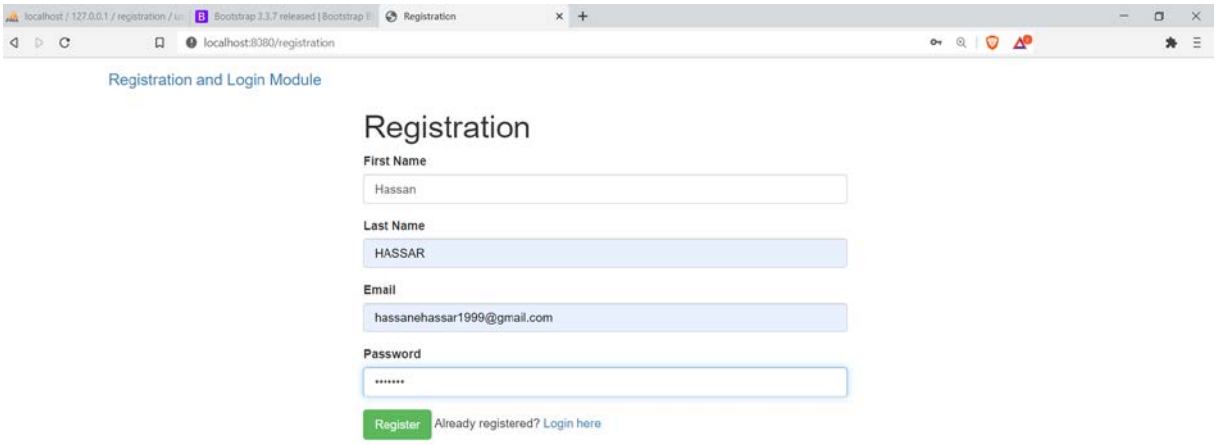


Figure 204:Register new user

91. Confirmation of success registration is displayed

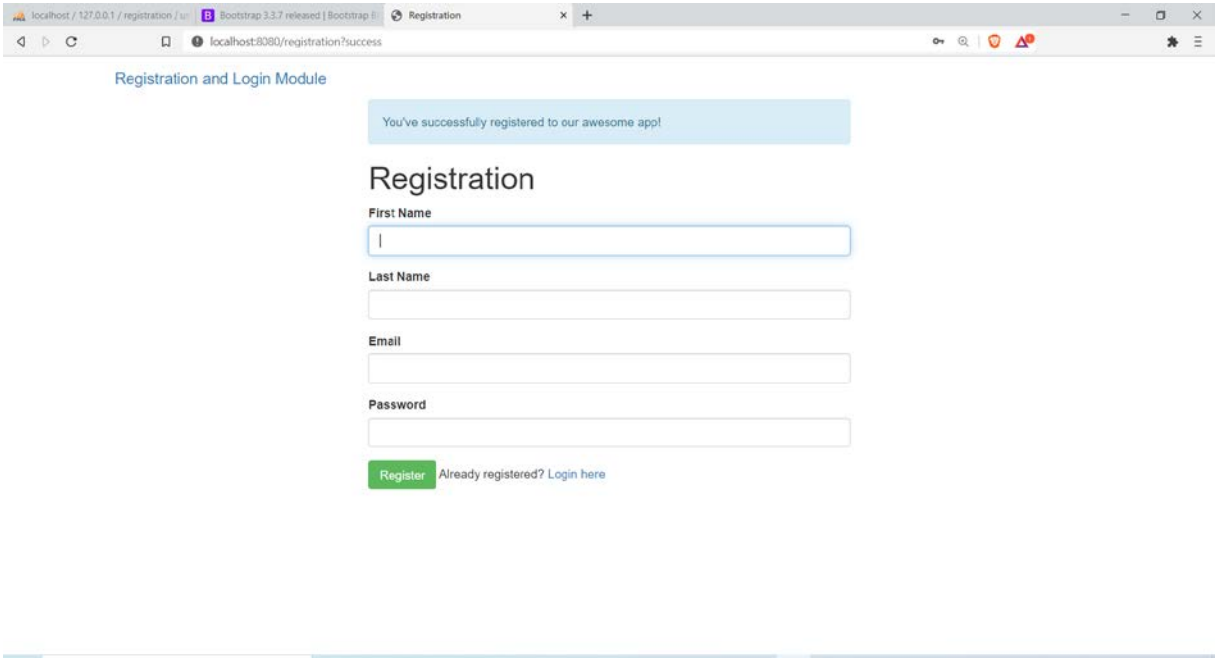


Figure 205:Confirmation of success registration is displayed

92. Try to fill the login form with the information of the new user

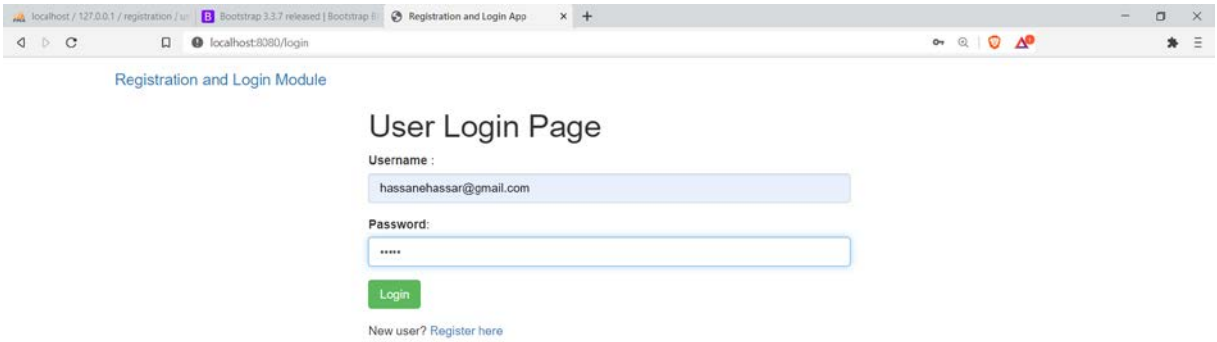


Figure 206:Try to fill the login form with the information of the new user

93. After logging button click, we get this error message of type Not Found 404 because there is no homepage.

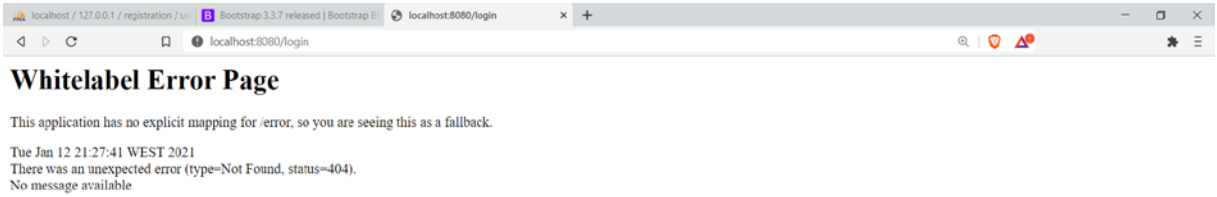


Figure 207:After logging button click, we get this error message of type Not Found 404 because

there is no homepage

94. Adding a home page to our app

95. For this, First add home method that returns index page in the MainController

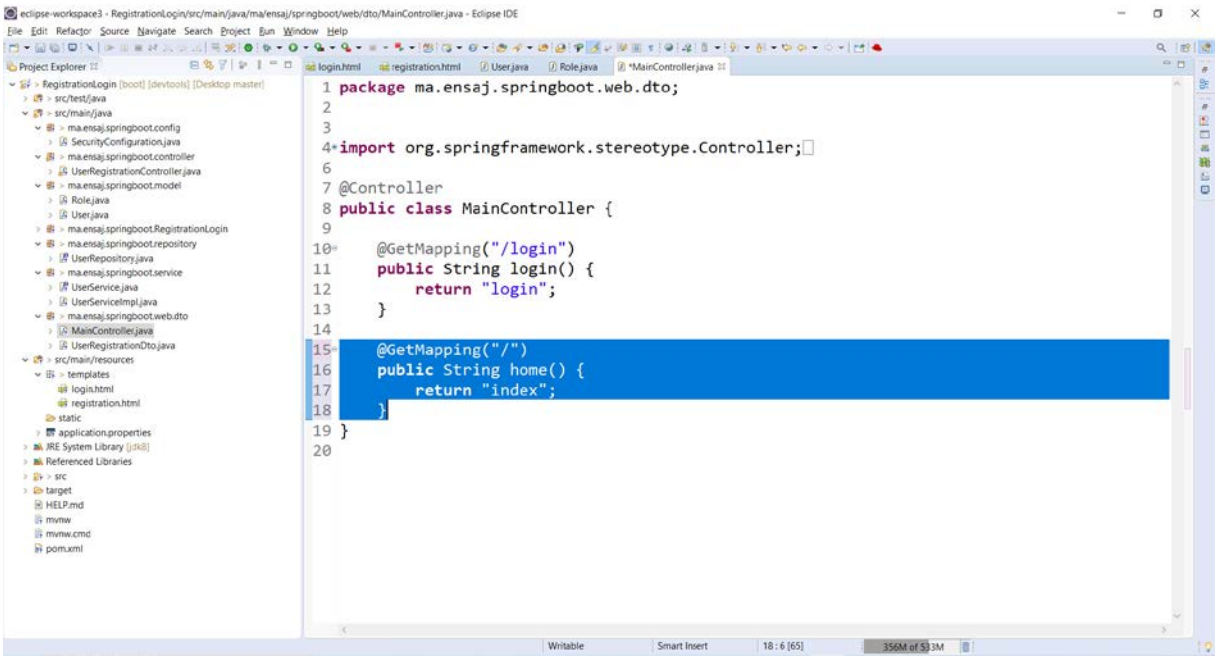


Figure 208:For this, First add home method that returns index page in the MainController

96. As we did before for the registration and login pages. Let’s create a new file called index.html

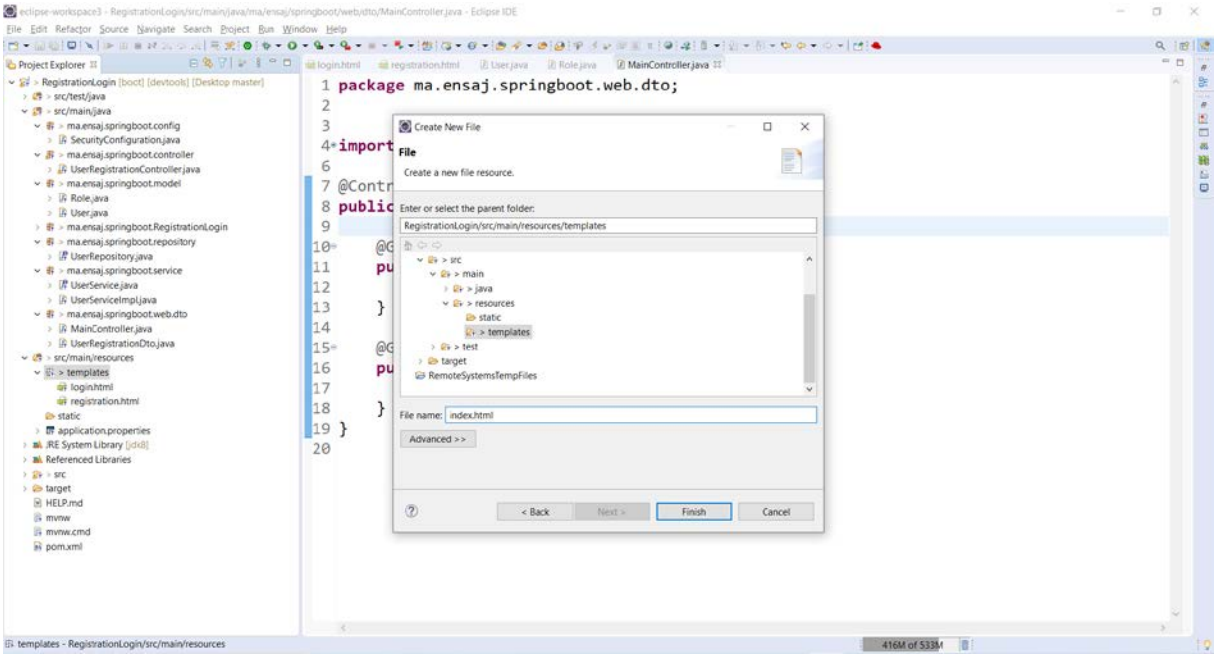
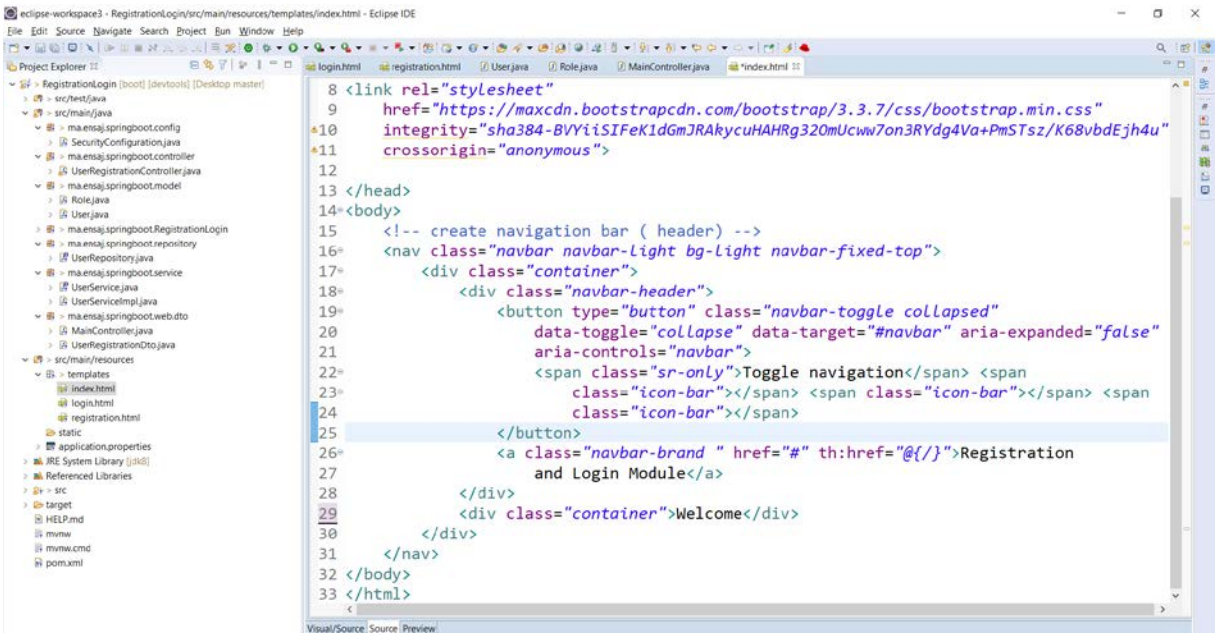


Figure 209:As we did before for the registration and login pages. Let’s create a new file called index.html

97. The navigation bar that’s common for all views



```
<link rel="stylesheet"
    href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
    integrity="sha384-BVYiiSIFeK1dGmJRAkycuHAHRg32OmUcww7on3RYdg4Va+PmSTsz/K68vbdEjh4u"
    crossorigin="anonymous">

</head>
<body>
    <!-- create navigation bar ( header) -->
    <nav class="navbar navbar-light bg-light navbar-fixed-top">
        <div class="container">
            <div class="navbar-header">
                <button type="button" class="navbar-toggle collapsed"
                    data-toggle="collapse" data-target="#navbar" aria-expanded="false"
                    aria-controls="navbar">
                    <span class="sr-only">Toggle navigation</span> <span
                        class="icon-bar"></span> <span class="icon-bar"></span> <span
                        class="icon-bar"></span>
                </button>
                <a class="navbar-brand " href="#" th:href="@{/}">Registration
                    and Login Module</a>
            </div>
            <div class="container">Welcome</div>
        </div>
    </nav>
</body>
</html>
```

Figure 210:The navigation bar that’s common for all views

98. Let’s run the app and test the welcome page if is working well after logging

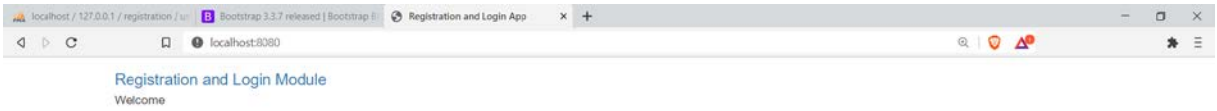


Figure 211:Let’s run the app and test the welcome page if is working well after logging

99. Notice: the presence of Thymeleaf dependency in the pom.,xml

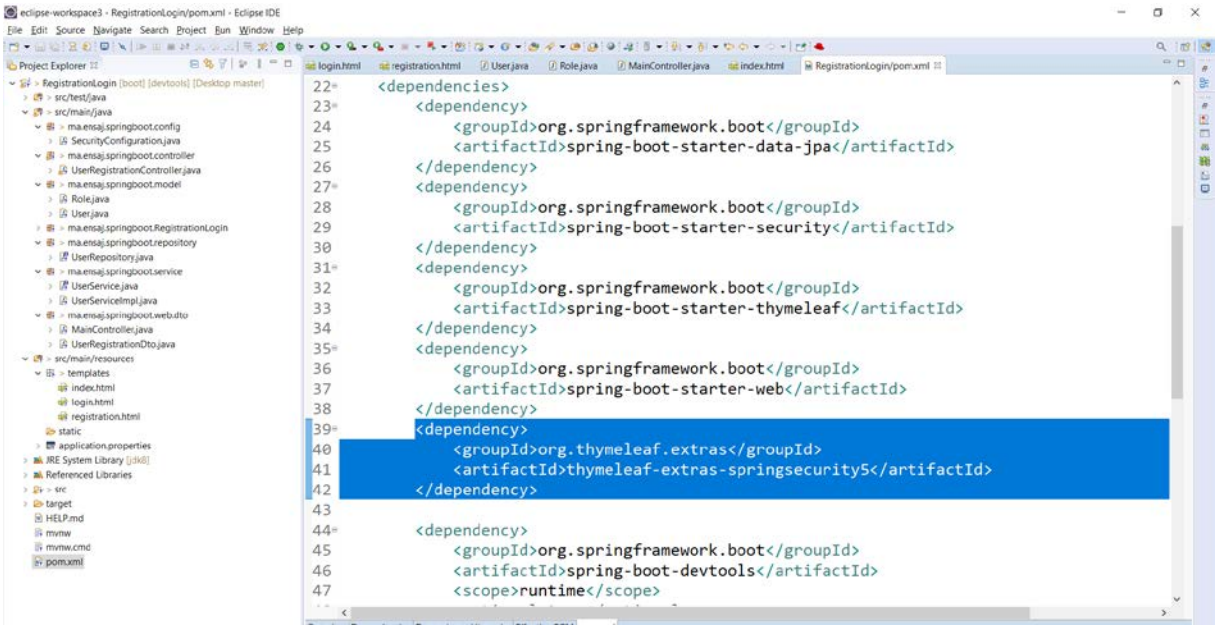


Figure 212:Notice: the presence of Thymeleaf dependency in the pom.,xml

100. We want to display the email of the logging user after login

For example: Welcome hhassar99@gmail.com

101. So, for this we use security authentication to retrieve the email from the database

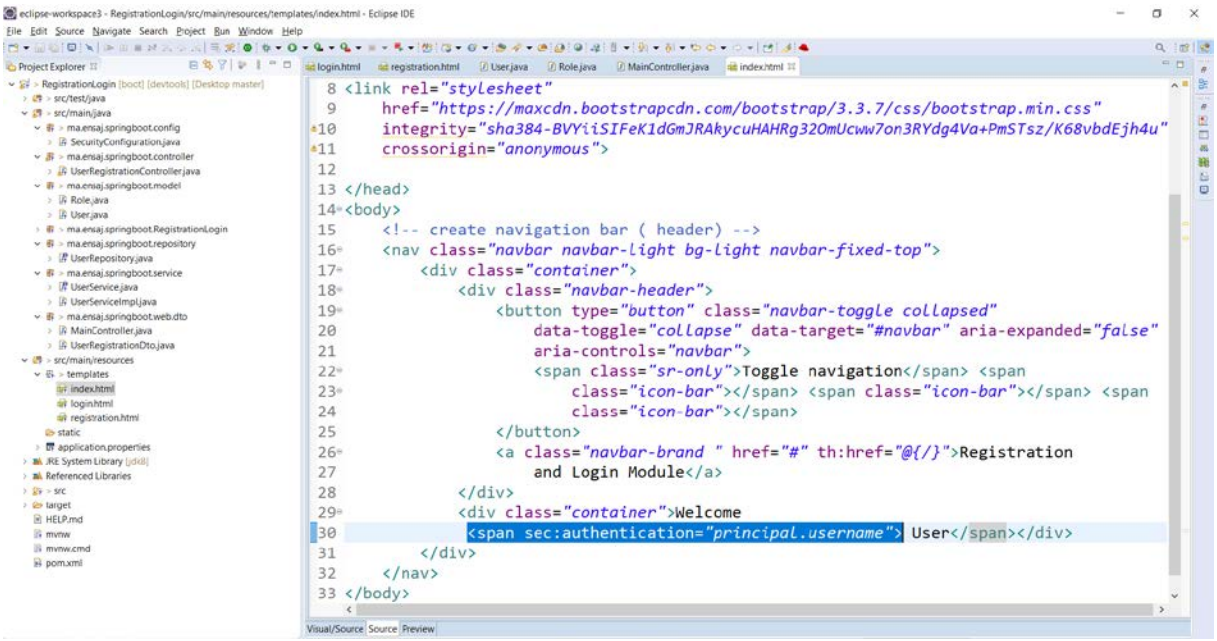


Figure 213:use security authentication to retrieve the email from the database

102. Displaying the email of the user is working well

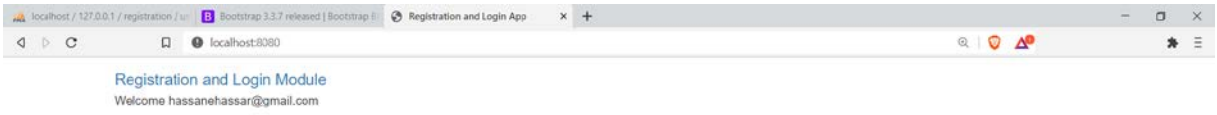


Figure 214:Displaying the email of the user is working well

103. We want to add the logout link in the navbar so we add the selected code

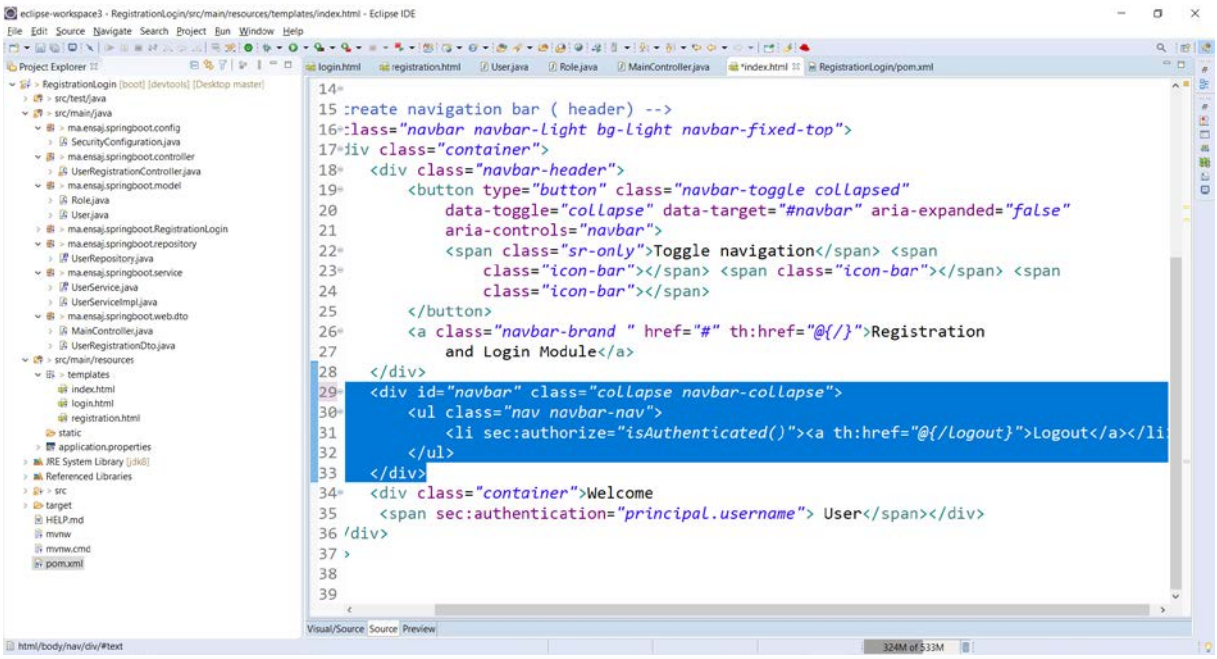


Figure 215:We want to add the logout link in the navbar so we add the selected code

104. **Notice :** In the configuration security file specifically in the configure method we observe the presence of /logout

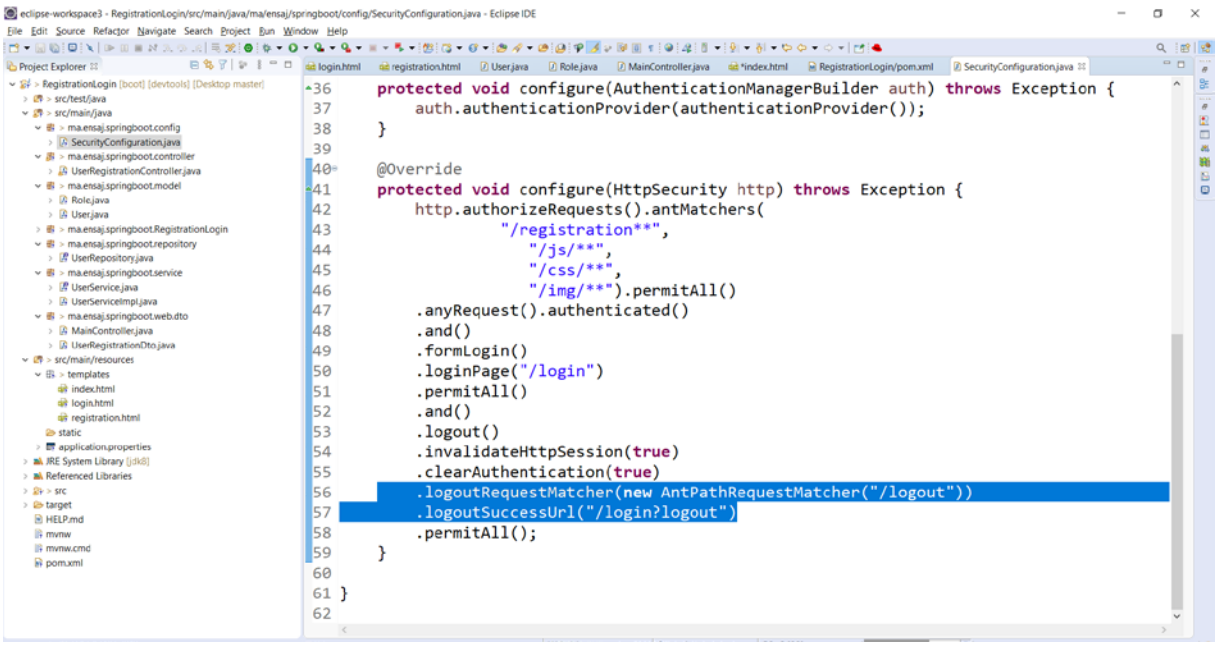


Figure 216:Notice : In the configuration security file specifically in the configure method we observe the presence of /logout

105. We run the app and test if the Logout link was successfully added to the navbar

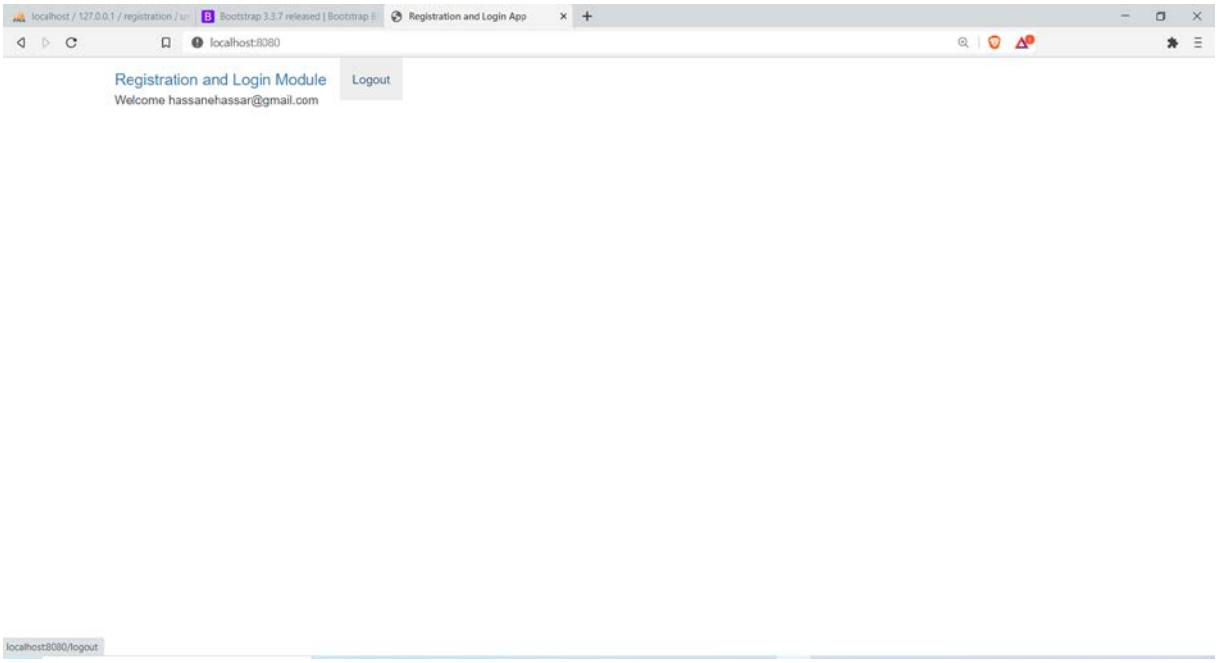


Figure 217:We run the app and test if the Logout link was successfully added to the navbar

106. After clicking on the logout link, we are redirected to the login page and a confirmation message is displayed

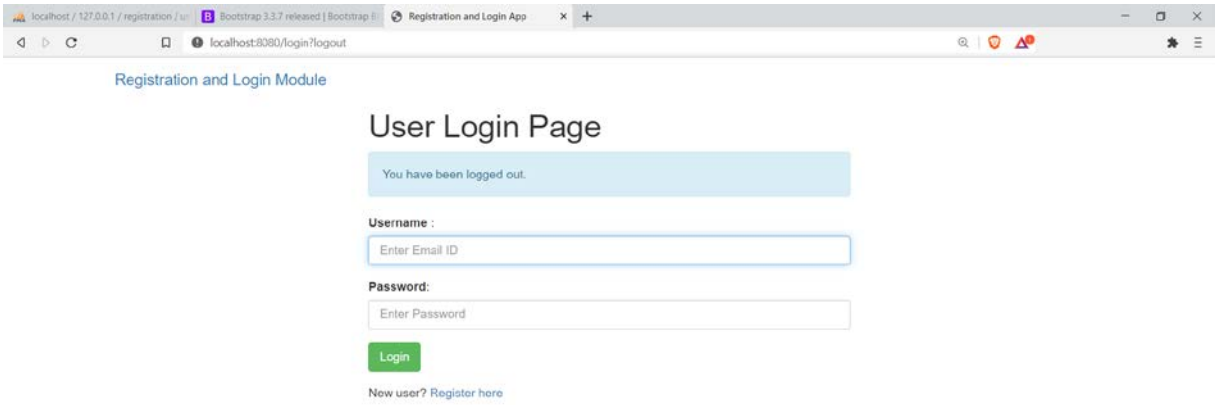


Figure 218:After clicking on the logout link, we are redirected to the login page and a confirmation message is displayed

4.1.5 Configuration Microservice Implementation

1. Initial creation of the application with **Spring Initializr**

Initializr offers a quick way to extract all the dependencies you need for an application and does much of the configuration for you. This example only needs the **Config Server** dependencies. The following image shows the Initializr configured for this example project:

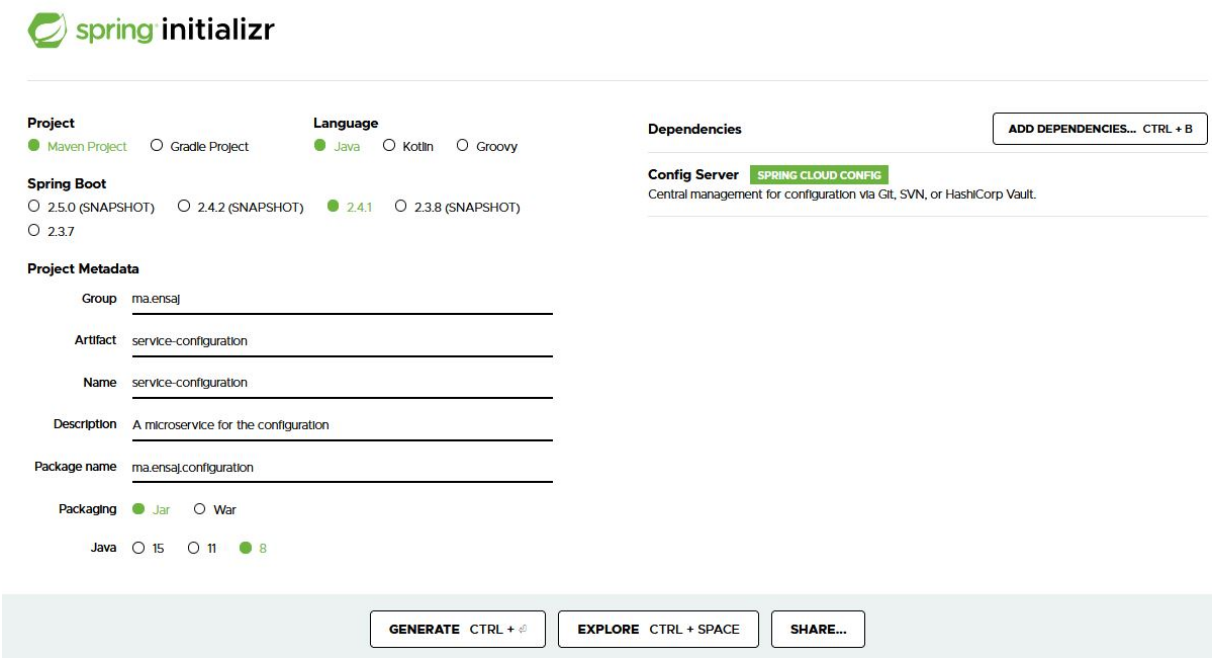


Figure 219:Spring Initializr

2. After download the project and extract it we run the command : **mvn eclipse :eclipse** to make the project an eclipse project

```
C:\Users\MAZER Omar\Desktop\Projets\JEE\service-configuration>mvn eclipse:eclipse
[INFO] Scanning for projects...
Downloading from spring-milestones: https://repo.spring.io/milestone/org/springframework/cloud/spring-cloud-dependencies
/2020.0.0/spring-cloud-dependencies-2020.0.0.pom
Downloading from central: https://repo.maven.apache.org/maven2/org/springframework/cloud/spring-cloud-dependencies/2020.
0.0/spring-cloud-dependencies-2020.0.0.pom
Downloaded from central: https://repo.maven.apache.org/maven2/org/springframework/cloud/spring-cloud-dependencies/2020.0
.0/spring-cloud-dependencies-2020.0.0.pom (11 kB at 21 kB/s)
Downloading from spring-milestones: https://repo.spring.io/milestone/org/springframework/cloud/spring-cloud-dependencies
-parent/3.0.0/spring-cloud-dependencies-parent-3.0.0.pom
Downloading from central: https://repo.maven.apache.org/maven2/org/springframework/cloud/spring-cloud-dependencies-paren
t/3.0.0/spring-cloud-dependencies-parent-3.0.0.pom
Downloaded from central: https://repo.maven.apache.org/maven2/org/springframework/cloud/spring-cloud-dependencies-parent
/3.0.0/spring-cloud-dependencies-parent-3.0.0.pom (8.5 kB at 46 kB/s)
Downloading from spring-milestones: https://repo.spring.io/milestone/org/springframework/cloud/spring-cloud-commons-depe
ndencies/3.0.0/spring-cloud-commons-dependencies-3.0.0.pom
Downloading from central: https://repo.maven.apache.org/maven2/org/springframework/cloud/spring-cloud-commons-dependenci
es/3.0.0/spring-cloud-commons-dependencies-3.0.0.pom
Downloaded from central: https://repo.maven.apache.org/maven2/org/springframework/cloud/spring-cloud-commons-dependencie
s/3.0.0/spring-cloud-commons-dependencies-3.0.0.pom (7.1 kB at 27 kB/s)
Downloading from spring-milestones: https://repo.spring.io/milestone/org/springframework/cloud/spring-cloud-netflix-depe
ndencies/3.0.0/spring-cloud-netflix-dependencies-3.0.0.pom
Downloading from central: https://repo.maven.apache.org/maven2/org/springframework/cloud/spring-cloud-netflix-dependenci
es/3.0.0/spring-cloud-netflix-dependencies-3.0.0.pom
Downloaded from central: https://repo.maven.apache.org/maven2/org/springframework/cloud/spring-cloud-netflix-dependencie
```

Figure 220:make the project an eclipse project

3. After importing the project under Eclipse. Here is the Project Structure

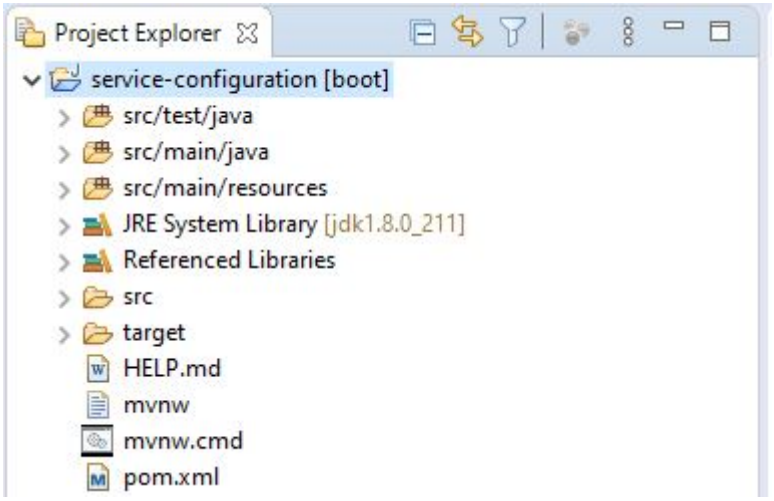


Figure 221:Project Structure

4. A Spring Boot application is a normal application, with main and annotations :

The **@SpringBootApplication** annotation is a meta-annotation that triggers auto-configuration and component scanning (in the classic Spring sense).

5. We add the @EnableConfigServer annotation

N.B: The **@EnableConfigServer** annotation makes your Spring Boot application act as a Configuration Server. Configuration Server runs on the Tomcat port 8888 and application configuration properties are loaded from native search locations.

ServiceConfigurationApplication.java

```
1  package ma.ensaj.configuration;
2
3  import org.springframework.boot.SpringApplication;
6  @EnableConfigServer
7  @SpringBootApplication
8  public class ServiceConfigurationApplication {
9
10     public static void main(String[] args) {
11         SpringApplication.run(ServiceConfigurationApplication.class, args);
12     }
13
14 }
```

Figure 222:@EnableConfigServer annotation

6. Configuration Server runs on the **Tomcat port 8888** and **application configuration properties** are loaded from native search locations. where **${user.home}/config-repo** is a git repository containing YAML and properties files.

application.properties

```
1 server.port=8888
2 spring.cloud.config.server.git.uri=file://${user.home}/cloud-config
```

Figure 223:Configuration Server

7. Here under the main user directory for example here :**C:/Users/MAZER OMAR**, we create a folder could **cloud-config** and under this folder we initialize the git repository with **git init** command.

Figure 224:initialize the git repository with git init command.

8. Create all microservices' **.properties** files in VS code IDE.

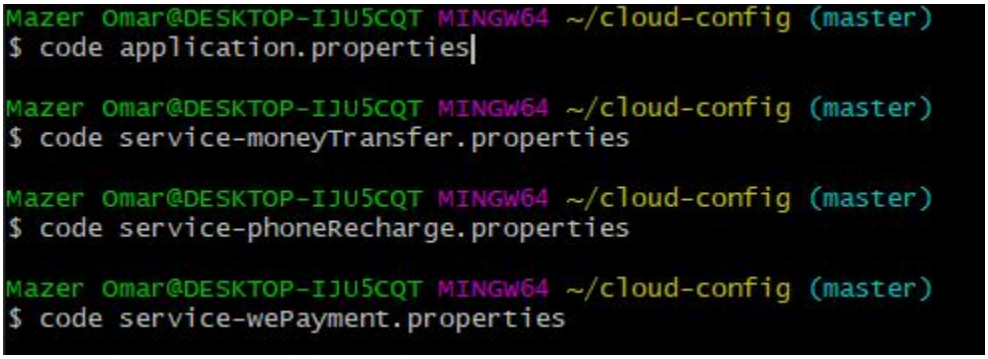


Figure 225:Create all microservices' .properties files in VS code IDE.

9. In **applications.properties** file we add these lines of code.(Configuration of the database and the Persistence Framework). In the **application.properties** file, configure the database connection parameters and some parameters related to the **Persistence** Framework :

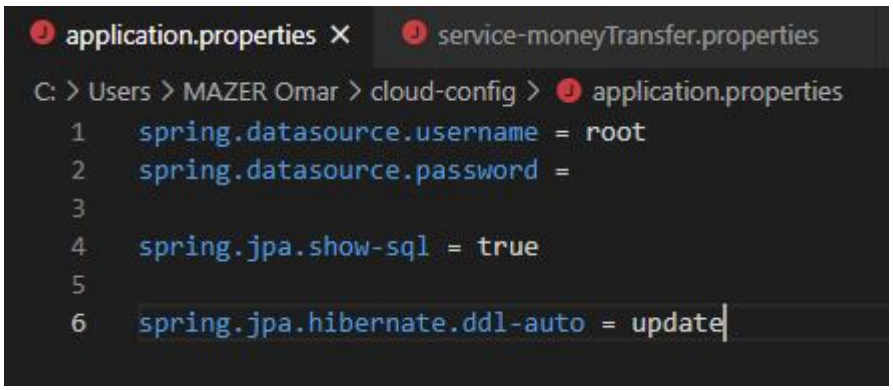


Figure 226:Configuration of the database and the Persistence Framework

10. We set **datasource url** and the **server.port:8081** for **moneyTransfer** service.

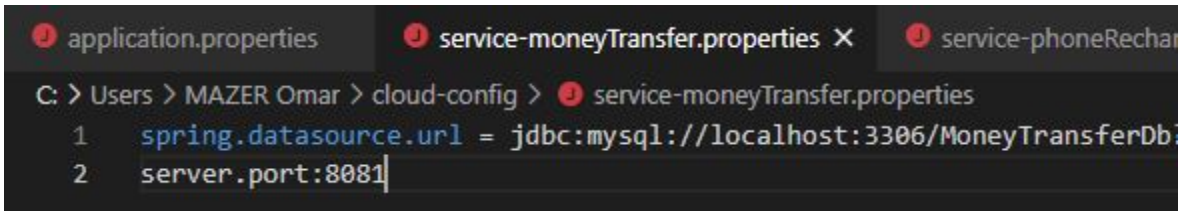


Figure 227:We set datasource url and the server.port:8081 for moneyTransfer service.

11. We set **datasource url** and the **server.port:8082** for **phoneRecharge** service.

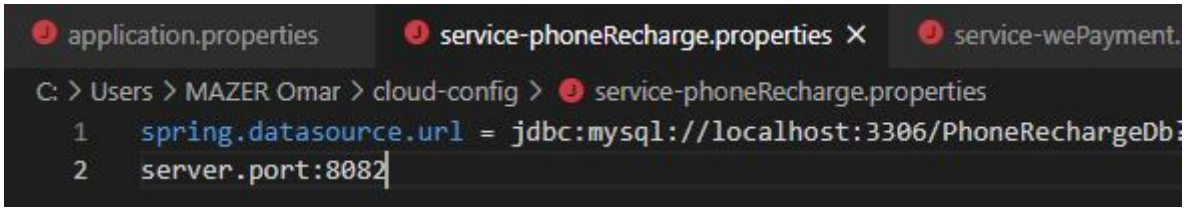


Figure 228:We set datasource url and the server.port:8082 for phoneRecharge service.

12. We set **datasource url** and the **server.port:8083** for **wePayment** service.

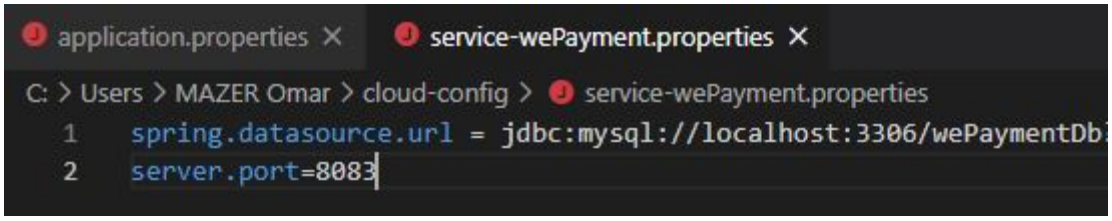


Figure 229:We set datasource url and the server.port:8083 for wePayment service.

13. Adding files to **git repository** cloud-config with the command **git add .**

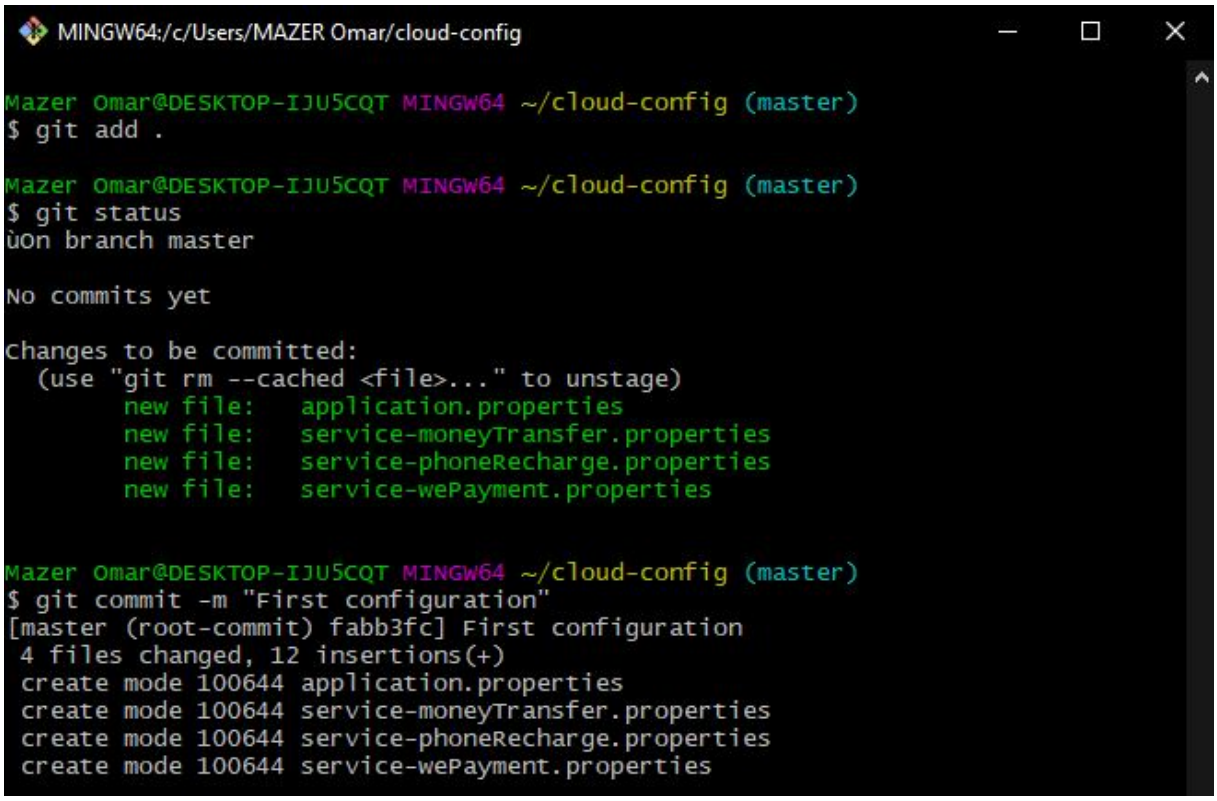


Figure 230:Adding files to git repository cloud-config with the command git add .

14. To start the application, use **mvn spring-boot:run** command

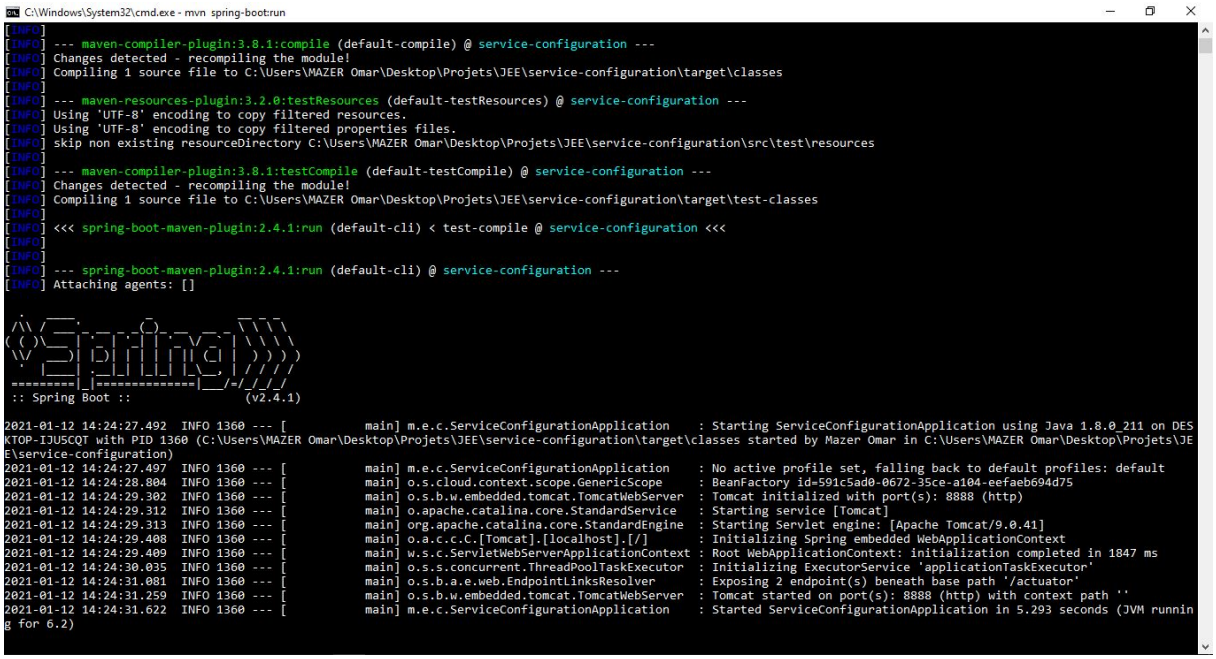


Figure 231:To start the application, use mvn spring-boot:run command

15. Let’s test it in the browser

Figure 232:Test

Figure 233:Test service-money

Figure 234:Test service-phoneRecharge

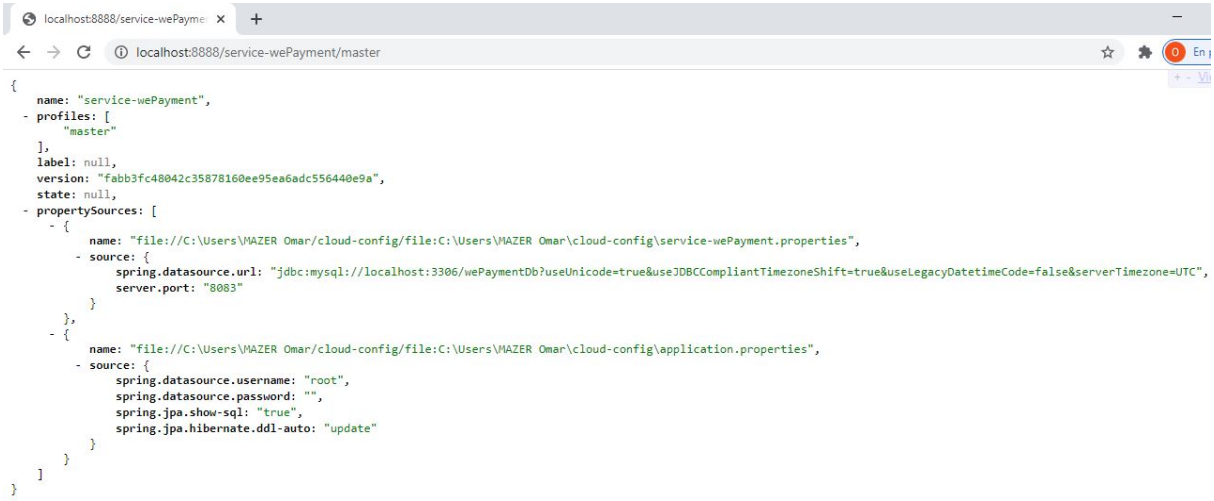


Figure 235:Test service-wePayment

16. Adding the **spring-cloud-starter-config** dependency in the **pom.xml** of each service**:**

1. **service-moneyTransfer\pom.xml**

service-moneyTransfer/pom.xml

```
28              <artifactId>spring-boot-starter-data-rest</artifactId>
29          </dependency>
30          <dependency>
31              <groupId>org.springframework.boot</groupId>
32              <artifactId>spring-boot-starter-web</artifactId>
33          </dependency>
34          <dependency>
35              <groupId>org.springframework.cloud</groupId>
36              <artifactId>spring-cloud-starter-config</artifactId>
37          </dependency>
```

Figure 236:service-moneyTransfer\pom.xml

2. **service-phoneRecharge\pom.xml**

service-phoneRecharge/pom.xml

```
31          </dependency>
32          <dependency>
33              <groupId>org.springframework.boot</groupId>
34              <artifactId>spring-boot-starter-web</artifactId>
35          </dependency>
36          <dependency>
37              <groupId>org.springframework.cloud</groupId>
38              <artifactId>spring-cloud-starter-config</artifactId>
39          </dependency>
```

Figure 237:service-phoneRecharge\pom.xml

3. **service-wePayment\pom.xml**

service-wePayment/pom.xml

```
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-config</artifactId>
        </dependency>
```

Figure 238:service-wePayment\pom.xml

17. Add **bootstrap.properties** file under the folder **src/main/ressources** for the three services

N.B: **bootstrap.properties** is responsible for loading configuration properties from the external sources and for decrypting properties in the local external configuration files. When the Spring Cloud application starts, it creates a bootstrap context.

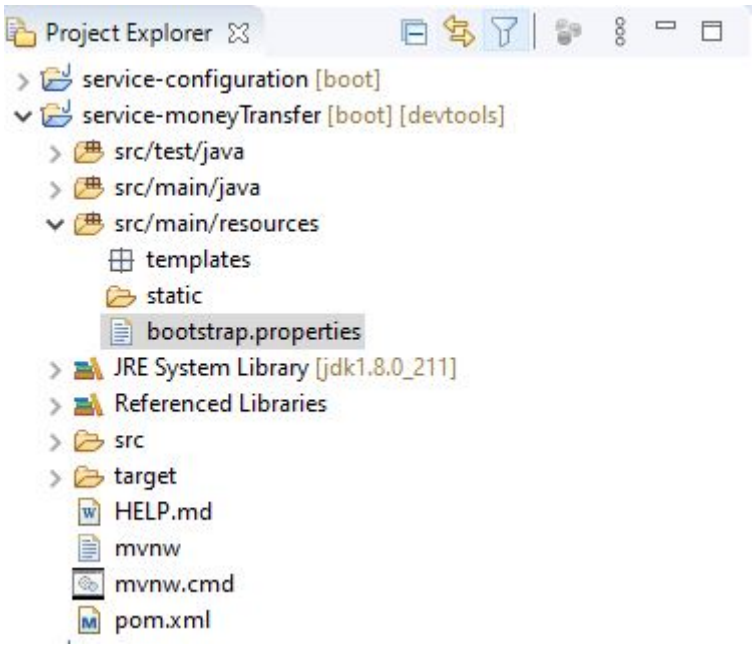


Figure 239:Add bootstrap.properties file under the folder src/main/ressources for the three services

18. We set these lines of code to all **bootstrap.properties** files :

spring.cloud.config.uri= http://localhost:8888 and

spring.cloud.config.profile=production

management.endpoints.web.exposure.include=*

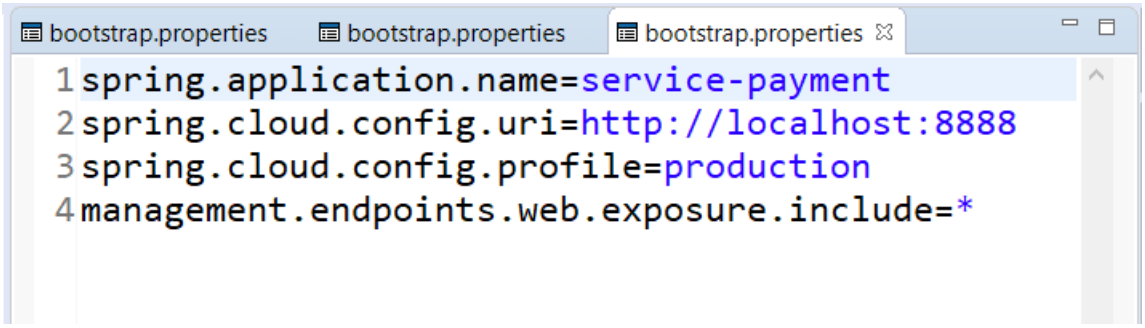


Figure 240:bootstrap.properties

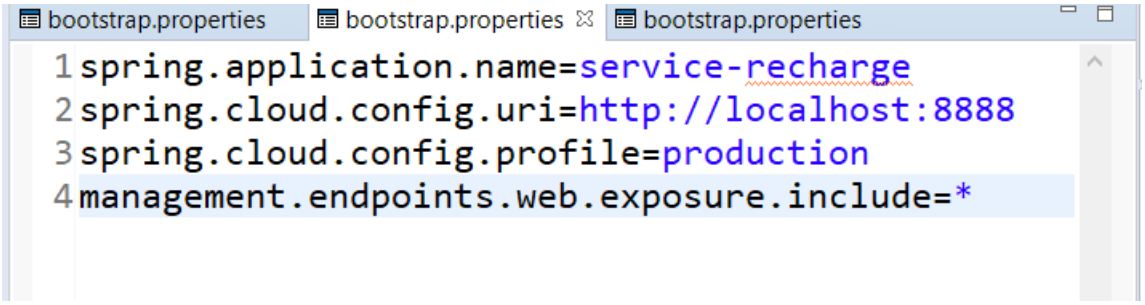


Figure 241:bootstrap.properties service-recharge

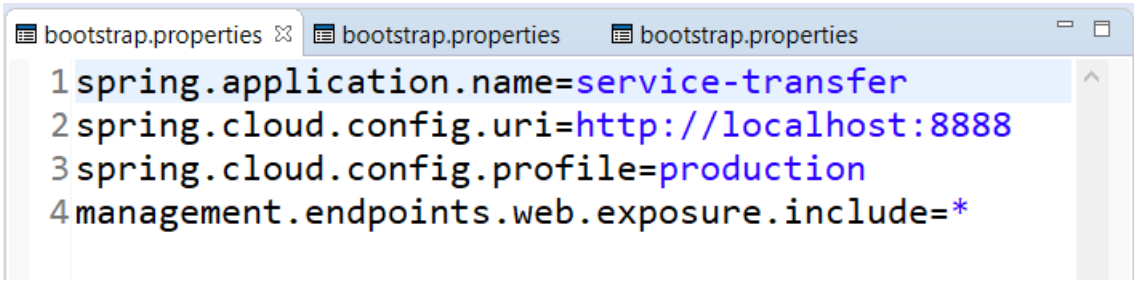


Figure 242:bootstrap.properties service-transfer

19. Adding the **spring-boot-starter-actuator dependency** under the **pom.xml** for **all services**

N.B: Spring Boot **Actuator** is a sub-project of the Spring Boot Framework. It includes a number of additional features that help us to monitor and manage the Spring Boot application. It contains the actuator endpoints (the place where the resources live).

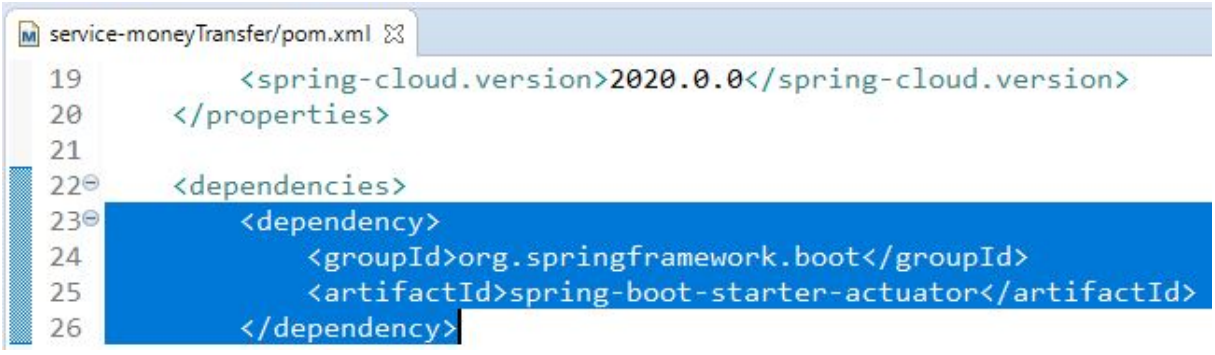


Figure 243:Adding the spring-boot-starter-actuator dependency

20. Set the server port for moneyTransfer service equal to **8085**

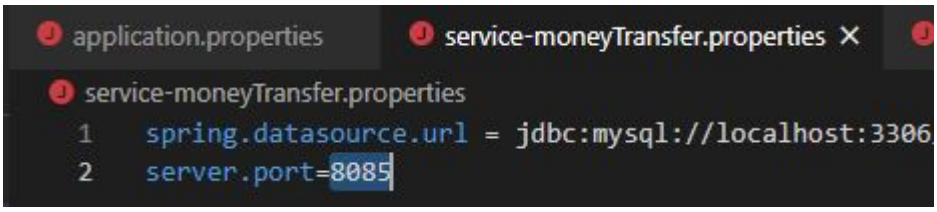


Figure 244:Set the server port for moneyTransfer service equal to 8085

21. Test the service actuator refresh in ARC

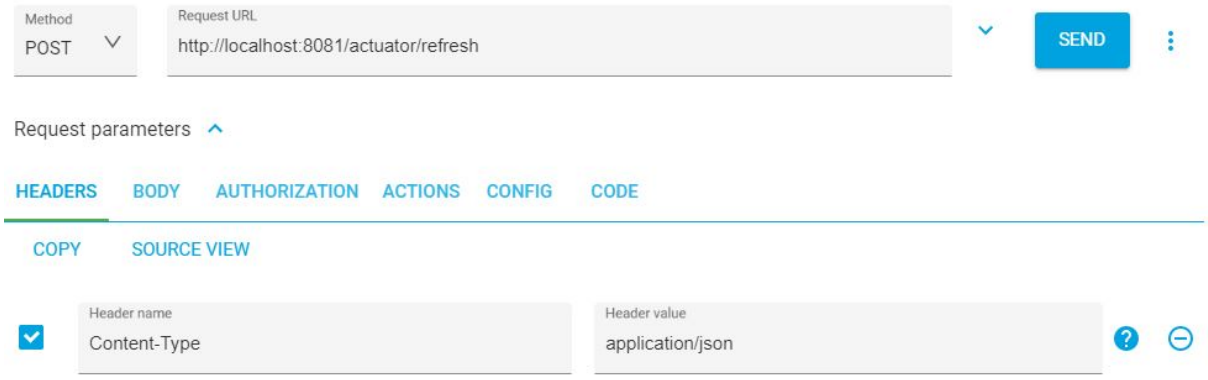


Figure 245:Test the service actuator refresh in ARC

22. The response of the request with the code message 200 OK

Figure 246:The response of the request with the code message 200 OK

23. We notice that the service money transfer runs on the server port 8085

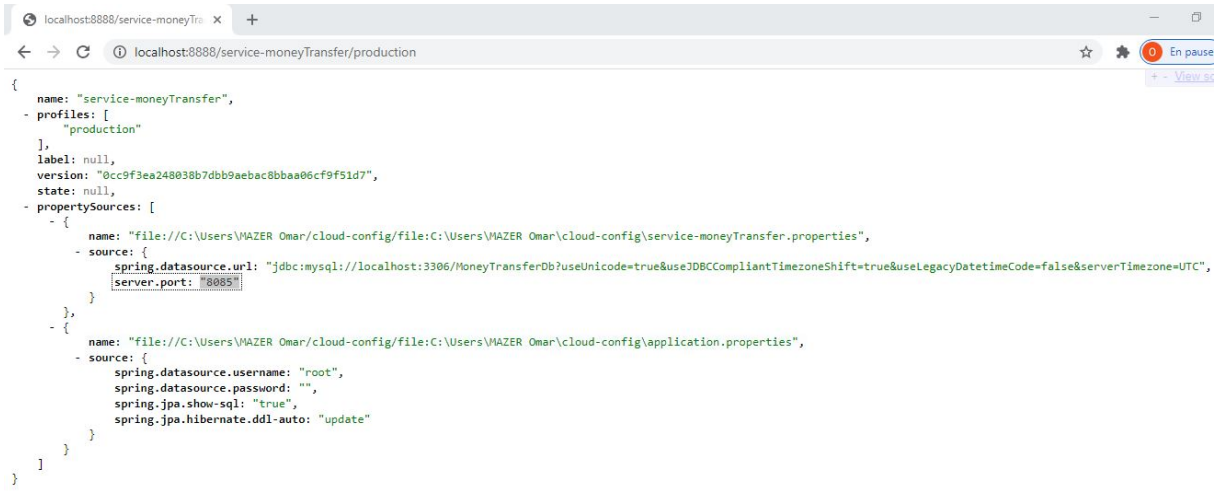


Figure 247:We notice that the service money transfer runs on the server port 8085

4.1.6 Regisration Microservice Implementation

1. Initial creation of the application with **Spring Initializr**

Initializr offers a quick way to extract all the dependencies you need for an application and does much of the configuration for you. This example only needs the **Config Client and Eureka Server** dependencies. The following image shows the Initializr configured for this example project:

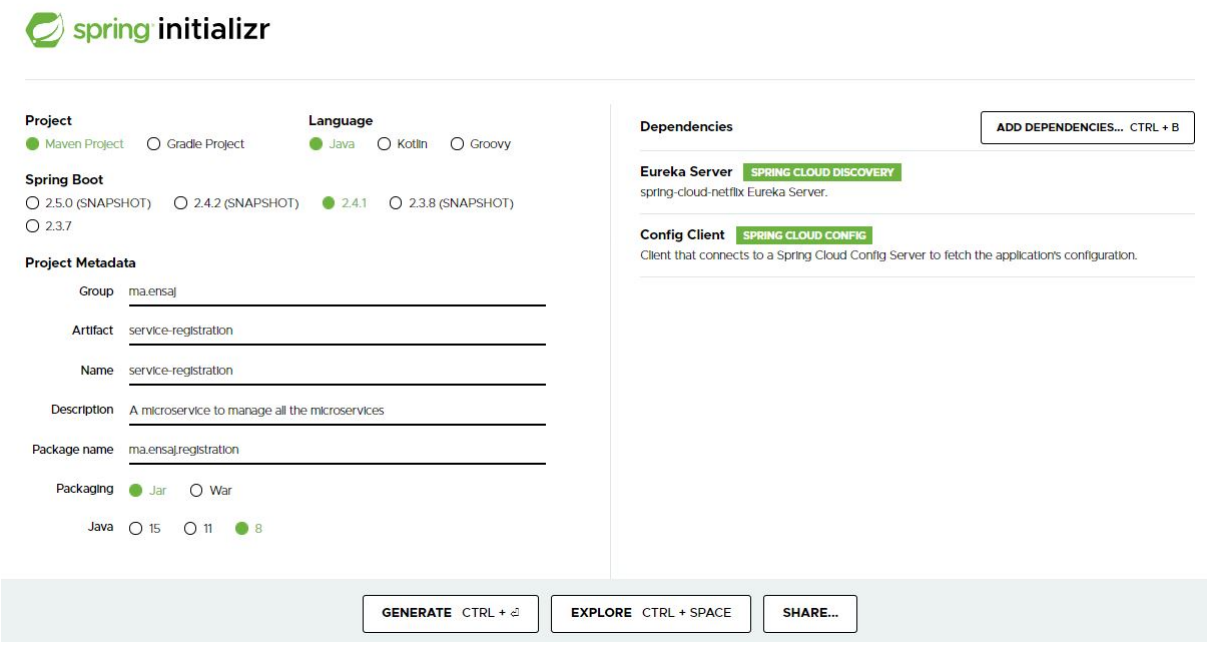


Figure 248:Spring Initializr

2. A Spring Boot application is a normal application, with main and annotations :

The **@SpringBootApplication** annotation is a meta-annotation that triggers auto-configuration and component scanning (in the classic Spring sense).

3. We add the @EnableConfigServer annotation

N.B: The **@EnableConfigServer** annotation makes your Spring Boot application act as a Configuration Server. Configuration Server runs on the Tomcat port 8888 and application configuration properties are loaded from native search locations.

ServiceRegistrationApplication.java

```
1  package ma.ensaj.registration;
2
3  import org.springframework.boot.SpringApplication;
6  @EnableEurekaServer
7  @SpringBootApplication
8  public class ServiceRegistrationApplication {
9
10     public static void main(String[] args) {
11         SpringApplication.run(ServiceRegistrationApplication.class, args);
12     }
13
14 }
15
```

Figure 249:@EnableConfigServer annotation

4. Creation of **service-registration.properties** file under VS code IDE.

```
Mazer Omar@DESKTOP-IJU5CQT MINGW64 ~/cloud-config (master)
$ code service-registration.properties
```

Figure 250:Creation of service-registration.properties file under VS code IDE.

5. We add these lines of code in **service-registration.properties**

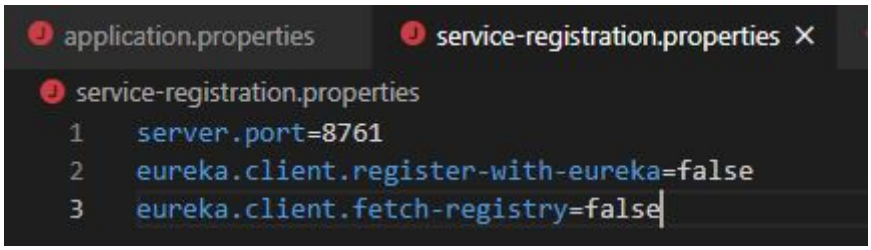


Figure 251:We add these lines of code in service-registration.properties

6. In **bootstrap.properties** we set these properties: **application name**, **spring-cloud.config.uri** and **spring-cloud.config.profile**.

Figure 252:In bootstrap.properties we set these properties: application name, spring-cloud.config.uri and spring-cloud.config.profile.

7. We go to the browser and we tap in the **url** the address **localhost:8761**.And we get the page of **spring Eureka.**

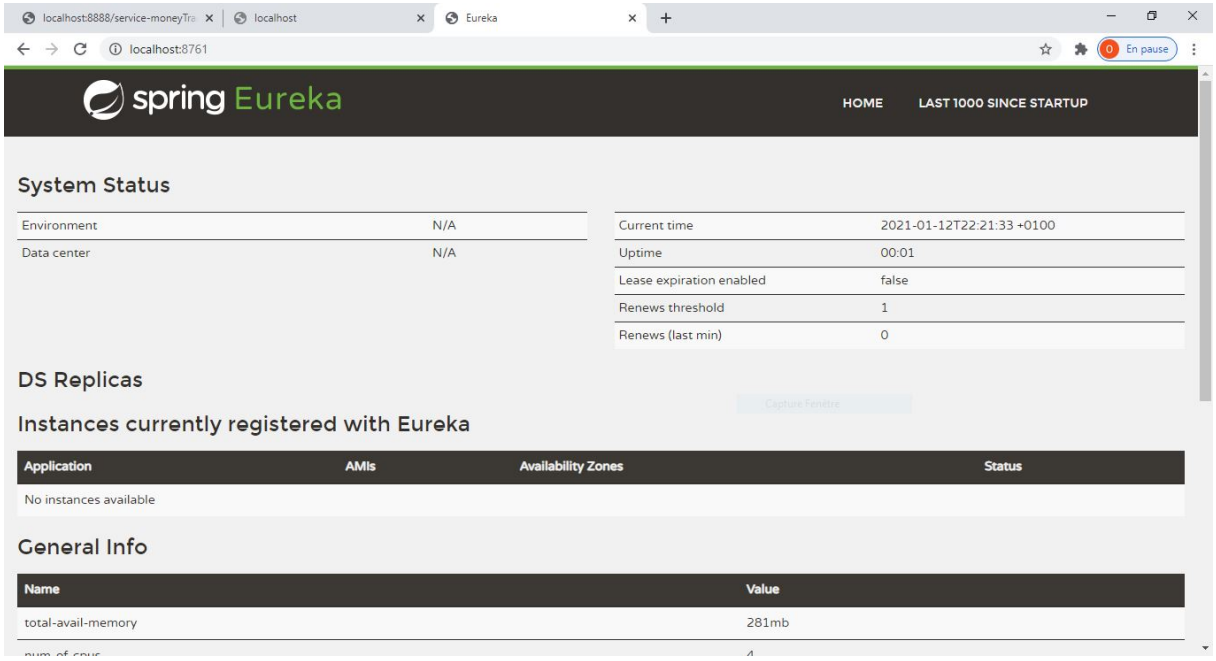


Figure 253:We go to the browser and we tap in the url the address localhost:8761.And we get the page of spring Eureka.

8. **Notice :** The presence of **spring-cloud-starter-netflix-eureka-server dependency** at the **pom.xml**.

service-moneyTransfer/pom.xml

```
49        <dependency>
50            <groupId>org.springframework.cloud</groupId>
51            <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
52        </dependency>
```

Figure 254:The presence of spring-cloud-starter-netflix-eureka-server dependency at the pom.xml.

9. We add the annotation **@EnableDiscoveryClient** to the main application file.

N.B: Spring Cloud Commons provides the **@EnableDiscoveryClient** annotation. This looks for implementations of the **DiscoveryClient** interface with **META-INF/spring.factories**. Implementations of the Discovery Client add a configuration class to **spring.factories** under the **org.springframework.cloud.client.discovery.EnableDiscoveryClient** key. Examples of **DiscoveryClient** implementations include Spring Cloud Netflix Eureka, Spring Cloud Consul Discovery, and Spring Cloud Zookeeper Discovery.

MoneyTransferApplication.java

```
1  package ma.ensaj.moneyTransfer;
2
3  import org.springframework.boot.SpringApplication;
6  @EnableDiscoveryClient
7  @SpringBootApplication
8  public class MoneyTransferApplication {
9
10     public static void main(String[] args) {
11         SpringApplication.run(MoneyTransferApplication.class, args);
12     }
13
14 }
```

Figure 255:add the annotation @EnableDiscoveryClient to the main application file

10. We notice now at **Spring Eureka** that all services are now running in their pre-specified port's numbers

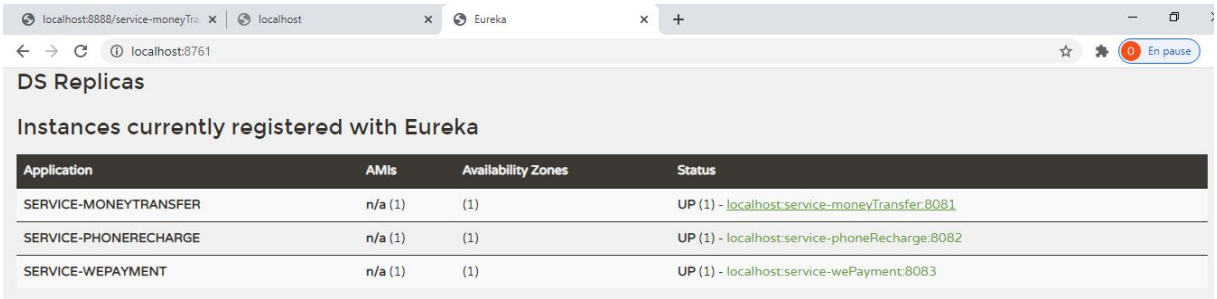


Figure 256:Spring Eureka

4.1.7 Proxy Microservice Implementation

1. Initial creation of the application with **Spring Initializr**

Initializr offers a quick way to extract all the dependencies you need for an application and does much of the configuration for you. This example only needs the **Config Server** dependencies. The following image shows the Initializr configured for this example project:

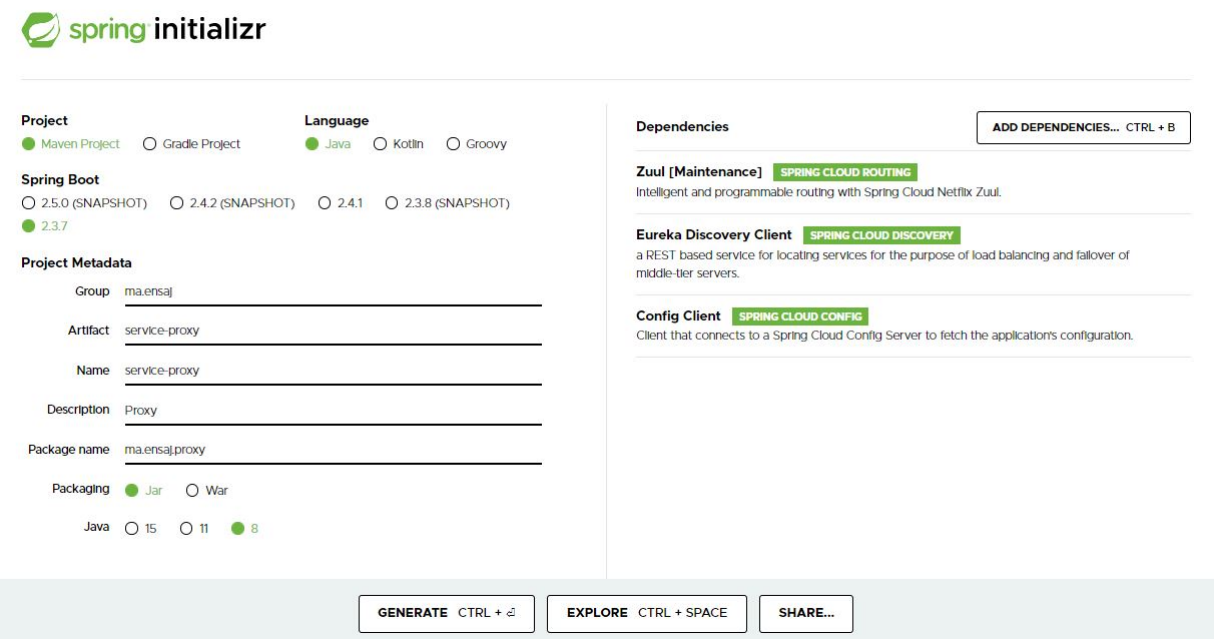


Figure 257:Spring Initializr

2. Add the **@EnableZuulProxy** annotation on my main Spring Boot application. The **@EnableZuulProxy** annotation is used to make my Spring Boot application act as a **Zuul Proxy** server.

ServiceProxyApplication.java

```java
package ma.ensaj.proxy;

import org.springframework.boot.SpringApplication;
@EnableZuulProxy
@SpringBootApplication
public class ServiceProxyApplication {

    public static void main(String[] args) {
        SpringApplication.run(ServiceProxyApplication.class, args);
    }

}
```

Figure 258:@EnableZuulProxy annotation on my main Spring Boot application

3. Creation of **service-proxy.properties file** under VS code IDE

```
Mazer Omar@DESKTOP-IJU5CQT MINGW64 ~/cloud-config (master)
$ code service-proxy.properties
```

Figure 259:Creation of service-proxy.properties file under VS code IDE

4. We set the port number to 8080

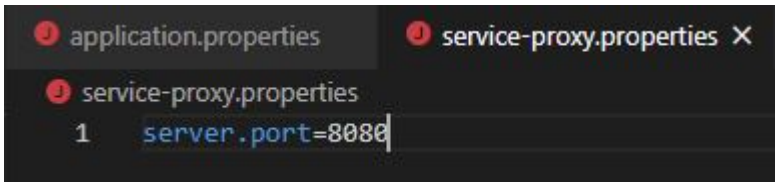


Figure 260:We set the port number to 8080

5. We add the **service-proxy.properties file** to the **cloud-config directory**

```
Mazer Omar@DESKTOP-IJU5CQT MINGW64 ~/cloud-config (master)
$ git add .

Mazer Omar@DESKTOP-IJU5CQT MINGW64 ~/cloud-config (master)
$ git commit -m "adding service-proxy.propeties"
[master b0295b6] adding service-proxy.propeties
 1 file changed, 1 insertion(+)
 create mode 100644 service-proxy.properties
```

Figure 261:We add the service-proxy.properties file to the cloud-config directory

6. In **bootstrap.properties** we set these properties: **application name**, **spring-**

cloud.config.uri and **spring-cloud.config.profile**.

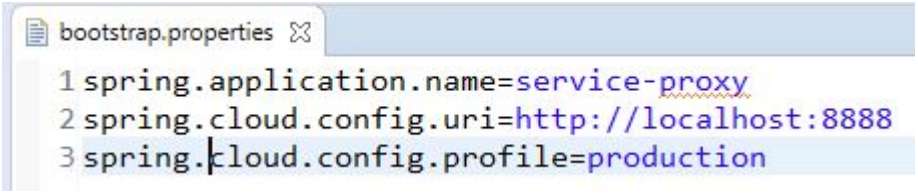


Figure 262:In bootstrap.properties we set these properties: application name, spring-cloud.config.uri and spring-cloud.config.profile.

7. We add the annotation **@EnableDiscoveryClient** to the main application file.

N.B: Spring Cloud Commons provides the **@EnableDiscoveryClient** annotation. This looks for implementations of the **DiscoveryClient** interface with **META-INF/spring.factories**. Implementations of the Discovery Client add a configuration class to **spring.factories** under the **org.springframework.cloud.client.discovery.EnableDiscoveryClient** key. Examples of **DiscoveryClient** implementations include Spring Cloud Netflix Eureka, Spring Cloud Consul Discovery, and Spring Cloud Zookeeper Discovery.

ServiceProxyApplication.java

```
1  package ma.ensaj.proxy;
2
3  import org.springframework.boot.SpringApplication;
7  @EnableZuulProxy
8  @EnableDiscoveryClient
9  @SpringBootApplication
10 public class ServiceProxyApplication {
11
12     public static void main(String[] args) {
13         SpringApplication.run(ServiceProxyApplication.class, args);
14     }
15
16 }
```

Figure 263:@EnableDiscoveryClient

8. We notice now at **Spring Eureka** that all services are now running in their pre-specified port's numbers and **service-proxy** is well added.

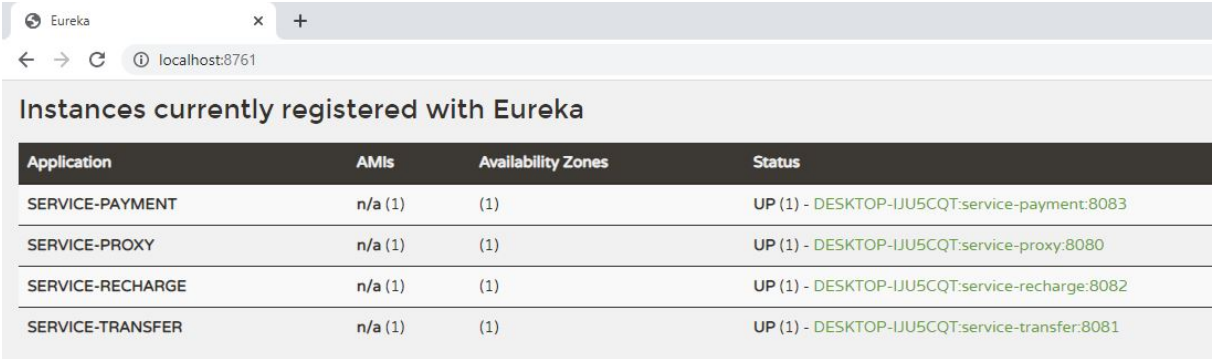


Figure 264:Spring Eureka

9. Open the browser and test get all money transfers list

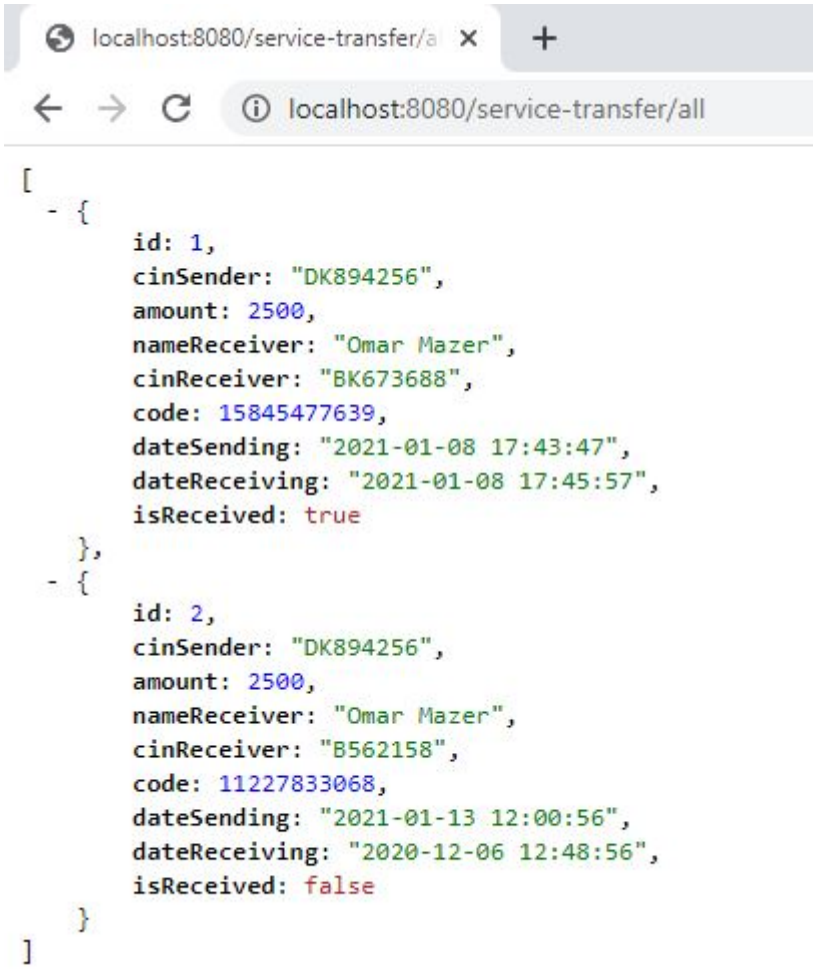


Figure 265:Open the browser and test get all money transfers list

10. Test service-recharge. The list is empty

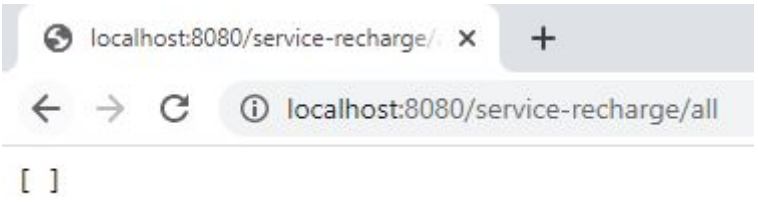


Figure 266:Test service-recharge. The list is empty

11. Test service-payment. The list is empty

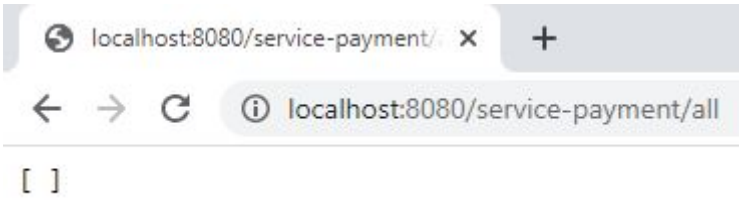


Figure 267:Test service-payment. The list is empty

4.2 Frontend

N.B: We have implemented notification feature for all microservices

4.2.1 Login Page

login.html

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="ISO-8859-1">
    <title>Login</title>
    <link rel="stylesheet" th:href="@{/css/bootstrap.min.css}" type="text/css">
    <link rel="stylesheet" th:href="@{/css/custom.css}" type="text/css">
</head>

<body style="background-image: url('/img/bgimg.PNG')">
    <nav class="navbar navbar-expand-lg navbar-light bg-light mb-4 shadow-sm ">
        <a class="navbar-brand" th:href="@{/}">TASHILATE-ENSAJ</a>
    </nav>
    <div class="col-lg-4 formStyle p-4">
        <h1> Login </h1>
        <form th:action="@{/login}" method="post">
            <!-- error message -->
            <div th:if="${param.error}">
                <div class="alert alert-danger">Invalid username or
                    password.</div>
            </div>
            <!-- logout message -->
            <div th:if="${param.logout}">
                <div class="alert alert-info">You have been logged out.</div>
            </div>
            <div class="form-group">
                <label for="username"> Username </label> :
                <input type="text" class="form-control" id="username" name="username" placeholder="Enter Email ID"
                    autofocus="autofocus">
            </div>
            <div class="form-group">
                <label for="password">Password</label>: <input type="password" id="password" name="password"
                    class="form-control" placeholder="Enter Password" />
            </div>
            <div class="form-group">

                <button type="submit" name="login-submit" id="login-submit"
                    class="btn btn-outline-success btn-block mb-4" value="Log In">Login</button>
            </div>
        </form>
        <div class="form-group">
            <span>New user? <a href="/" th:href="@{/registration}">Register
                    here</a></span>
        </div>
    </div>
```

Figure 268:login.html

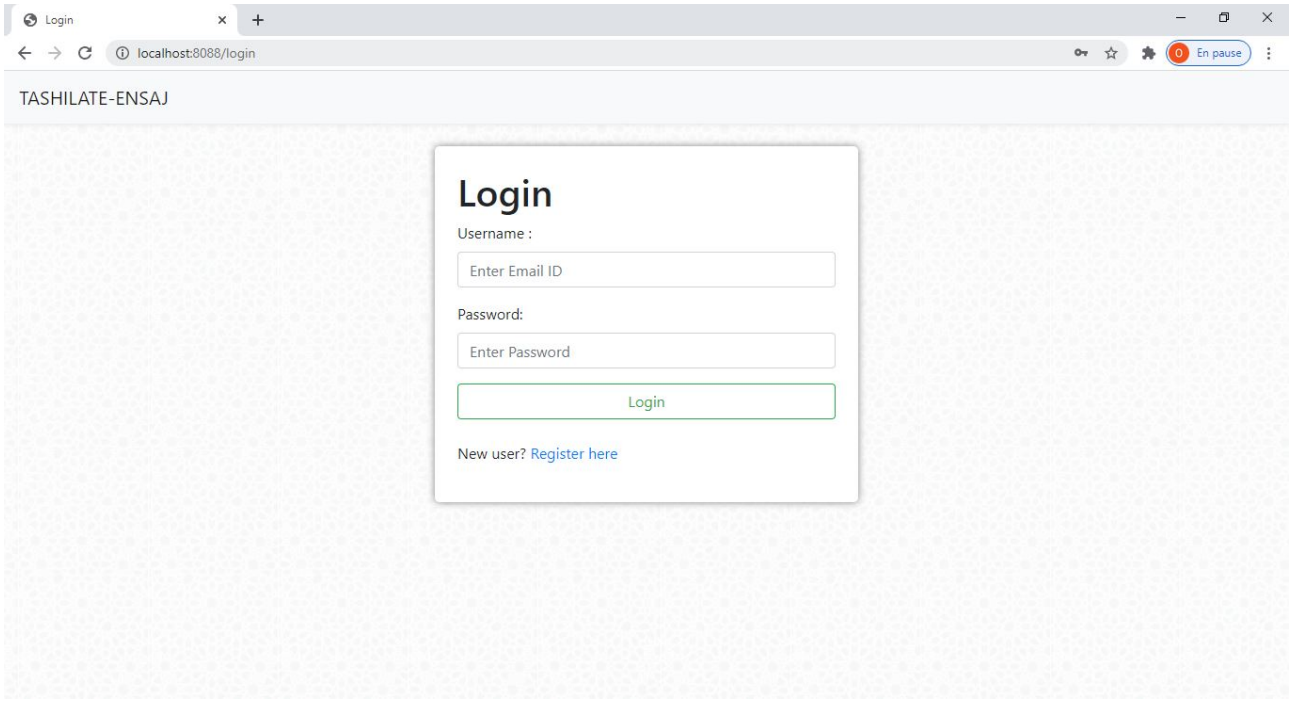


Figure 269:login page in the browser

4.2.2 Registration

registration.html

```
<head>
    <meta charset="ISO-8859-1">
    <title>Registration</title>
    <link rel="stylesheet" th:href="@{/css/bootstrap.min.css}" type="text/css">
    <link rel="stylesheet" th:href="@{/css/custom.css}" type="text/css">
</head>

<body style="background-image: url('/img/bgimg.PNG')">
    <nav class="navbar navbar-expand-lg navbar-light bg-light mb-4 shadow-sm ">
        <a class="navbar-brand" th:href="@{/}">TASHILATE-ENSAJ</a>
    </nav>
    <div class="col-lg-4 formStyle p-4">
        <!-- success message -->
        <div th:if="${param.success}">
            <br>
            <div class="alert alert-info">You've successfully registered
                to our awesome app!</div>
        </div>
        <h1>Registration</h1>
        <form th:action="@{/registration}" method="post" th:object="${user}">
            <div class="form-group">
                <label class="control-label" for="firstName"> First Name </label>
                <input id="firstName" class="form-control" th:field="*{firstName}" required autofocus="autofocus" />
            </div>
            <div class="form-group">
                <label class="control-label" for="lastName"> Last Name </label> <input id="lastName"
                    class="form-control" th:field="*{lastName}" required autofocus="autofocus" />
            </div>
            <div class="form-group">
                <label class="control-label" for="email"> Email </label> <input id="email" class="form-control"
                    th:field="*{email}" required autofocus="autofocus" />
            </div>
            <div class="form-group">
                <label class="control-label" for="password"> Password </label> <input id="password" class="form-control"
                    type="password" th:field="*{password}" required autofocus="autofocus" />
            </div>
            <div class="form-group">
                <button type="submit" class="btn btn-success btn-block mb-4">Register</button>
                <span>Already registered? <a href="/" th:href="@{/login}">Login
                        here</a></span>
            </div>
        </form>
    </div>
</body>
```

Figure 270:registration.html

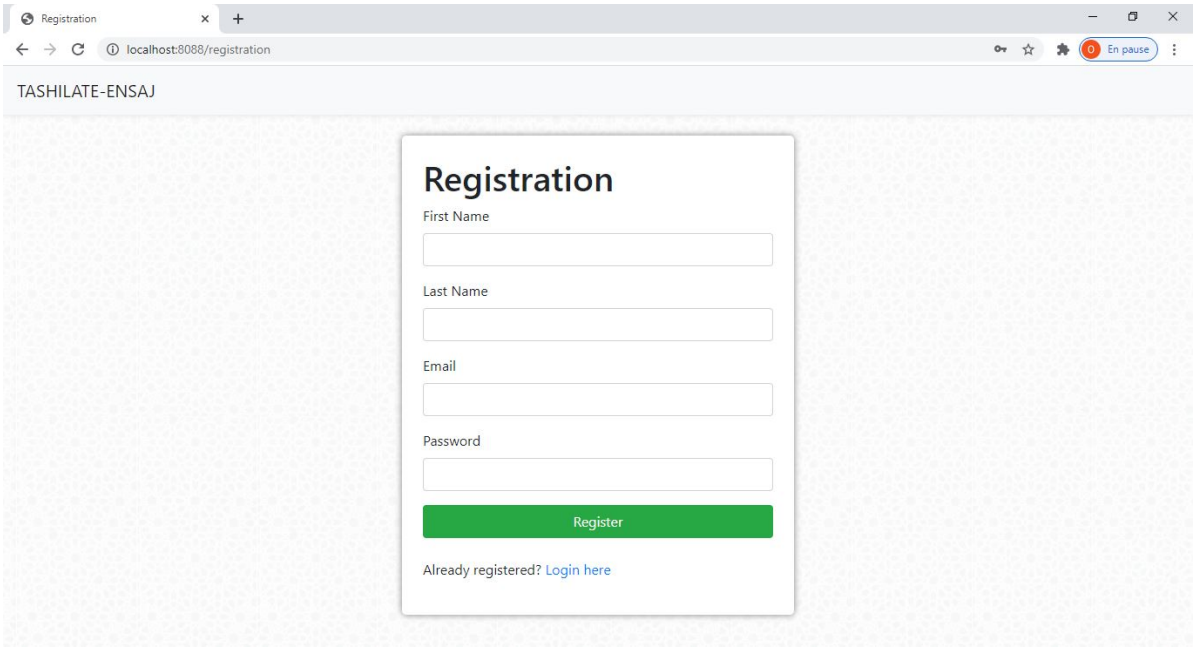


Figure 271:registration page in the browser

4.2.3 Index page

index.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org" xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-springsecurity3">

<head>
    <meta charset="ISO-8859-1">
    <title>TASHILATE-ENSAJ</title>
    <link rel="stylesheet" th:href="@{/css/bootstrap.min.css}" type="text/css">
    <link rel="stylesheet" th:href="@{/css/custom.css}" type="text/css">
</head>

<body style="background-image: url('/img/bgimg.PNG')">
    <nav class="navbar navbar-expand-lg navbar-light bg-light mb-4 shadow-sm ">
        <a class="navbar-brand" th:href="@{/}">TASHILATE-ENSAJ</a>
        <ul class="navbar-nav ml-auto">
            <li class="nav-item">
                <span sec:authentication="principal.username"> User</span>
            </li>
            <li sec:authorize="isAuthenticated()" class="nav-item ml-4 mr-4">
                <a th:href="@{/logout}">Logout</a>
            </li>
        </ul>
    </nav>
```

Figure 272:index.html

index.html

```
            </li>
        </ul>
    </nav>
<br><br><br><br>

    <div class="row d-flex justify-content-center">
        <div class="col-sm-12 col-lg-3 col-md-6 mb-5">
            <a th:href="@{/money_transfer}">
                <div class="card p-4 cardStyle">
                    <img src="/img/money_transfer.png" class="card-img-top serviceImg" alt="money_transfer">
                    <div class="card-body">
                        <h4 class="card-title text-center">Money Transfer</h4>
                    </div>
                </div>
            </a>
        </div>
        <div class="col-sm-12 col-lg-3 col-md-6 mb-5">
            <a th:href="@{/phone_recharge}">
                <div class="card p-4 cardStyle">
                    <img src="/img/phone_recharge.png" class="card-img-top serviceImg" alt="phone_recharge">
                    <div class="card-body">
                        <h4 class="card-title text-center">Phone Recharge</h4>
                    </div>
                </div>
            </a>
        </div>
        <div class="col-sm-12 col-lg-3 col-md-6 mb-5">
            <a th:href="@{/we_payment}">
                <div class="card p-4 cardStyle">
                    <img src="/img/we_payment.png" class="card-img-top serviceImg" alt="we_payment">
                    <div class="card-body">
                        <h4 class="card-title text-center">W/E Payment</h4>
                    </div>
                </div>
            </a>
        </div>
    </div>
</body>
</html>
```

Figure 273:index.html continuation code

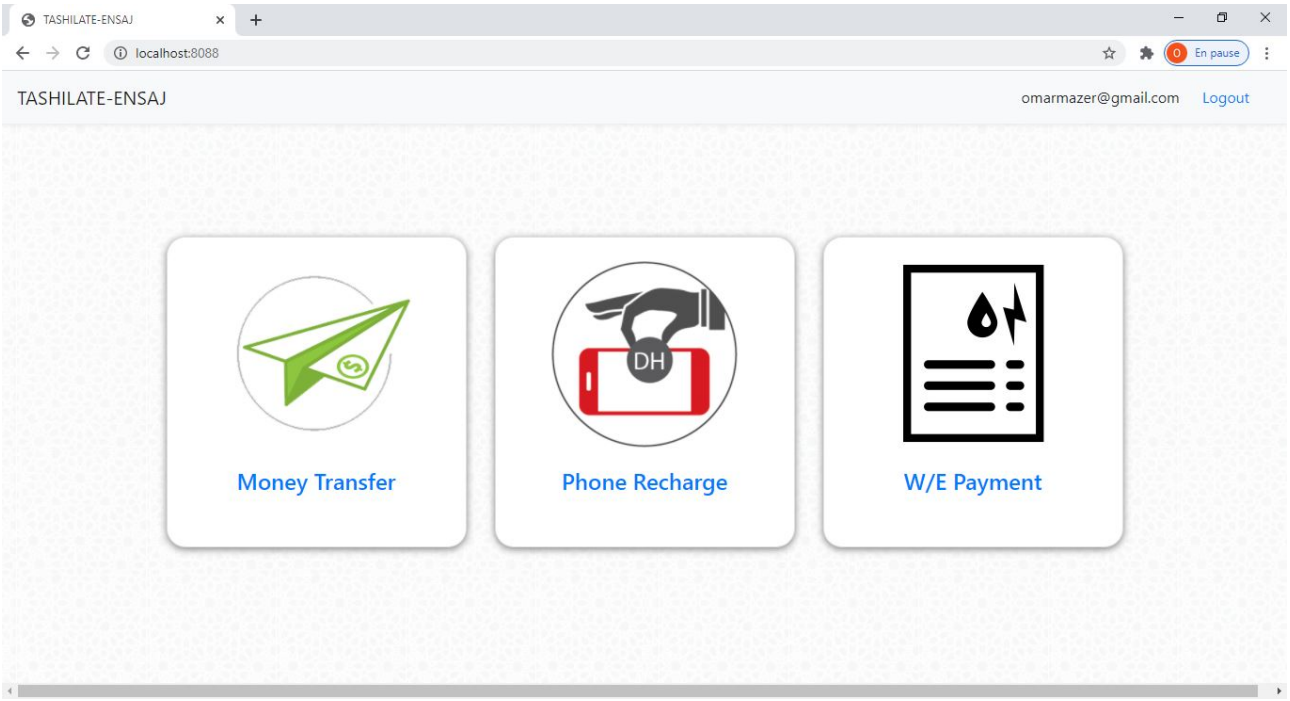


Figure 274:index.html in the browser

4.2.4 Money Transfer

money_transfer.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org" xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-springsecurity3">

<head>
    <meta charset="ISO-8859-1">
    <title>TASHILATE-ENSAJ</title>

    <link rel="stylesheet" th:href="@{/css/bootstrap.min.css}" type="text/css">
    <link rel="stylesheet" th:href="@{/css/custom.css}" type="text/css">
</head>
<body style="background-image: url('/img/bgimg.PNG')">
    <nav class="navbar navbar-expand-lg navbar-light bg-light mb-4 shadow-sm ">
        <a class="navbar-brand" th:href="@{/}">TASHILATE-ENSAJ</a>
        <ul class="navbar-nav ml-auto">
            <li class="nav-item">
                <span sec:authentication="principal.username"> User</span>
            </li>
            <li sec:authorize="isAuthenticated()" class="nav-item ml-4 mr-4">
                <a th:href="@{/logout}">Logout</a>
            </li>
        </ul>
    </nav>
<br><br><br><br>
    <div class="row d-flex justify-content-center">
        <div class="col-sm-12 col-lg-3 col-md-6 mb-5">
            <a th:href="@{/money_transfer/send_money}">
                <div class="card p-4 cardStyle">
                    <img src="/img/send_money.png" class="card-img-top serviceImg" alt="send_money">
                    <div class="card-body">
                        <h4 class="card-title text-center">Send Money</h4>
                    </div>
                </div>
            </a>
        </div>
        <div class="col-sm-12 col-lg-3 col-md-6 mb-5">
            <a th:href="@{/money_transfer/receive_money}">
                <div class="card p-4 cardStyle">
                    <img src="/img/receive_money.png" class="card-img-top serviceImg" alt="receive_money">
                    <div class="card-body">
                        <h4 class="card-title text-center">Receive Money</h4>
                    </div>
                </div>
            </a>
        </div>
    </div>
```

Figure 275:money_transfer.html

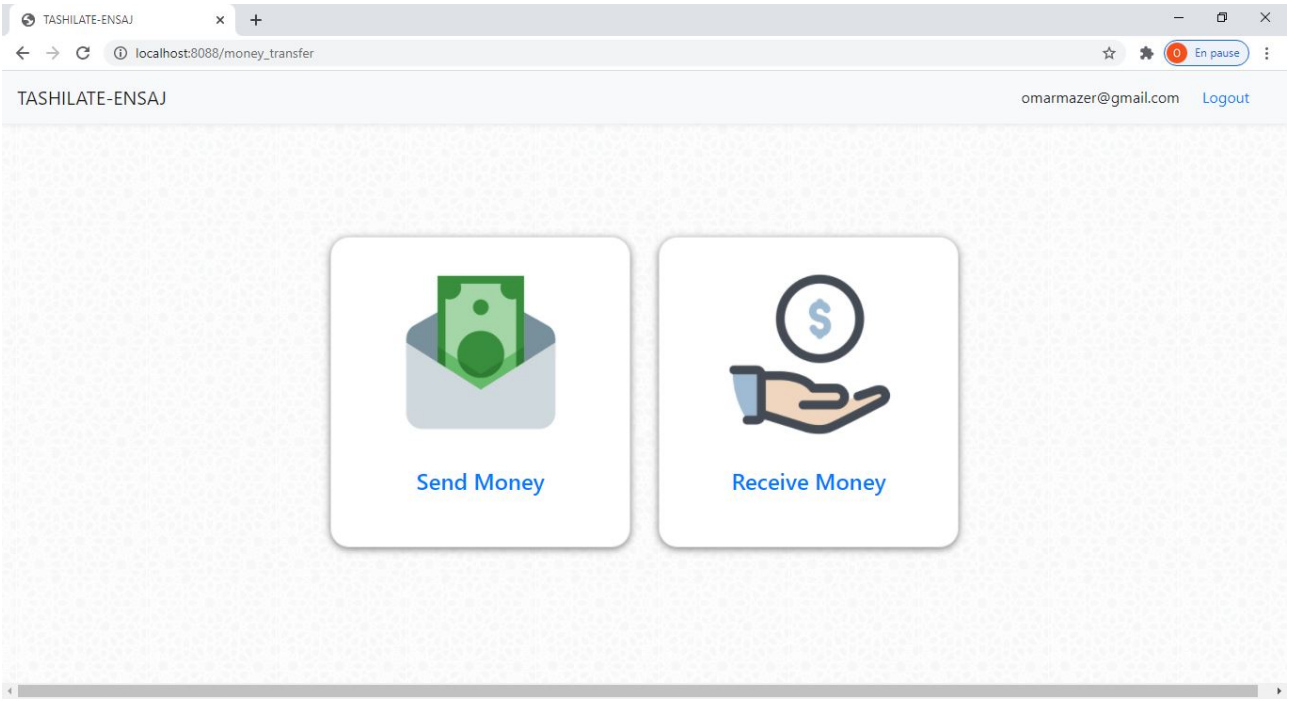


Figure 276:money transfer service in the browser

Send Money

send_money.html

```
<head>
    <meta charset="ISO-8859-1">
    <title>TASHILATE-ENSAJ</title>
    <link rel="stylesheet" th:href="@{/css/bootstrap.min.css}" type="text/css">
    <link rel="stylesheet" th:href="@{/css/custom.css}" type="text/css">
</head>
<body style="background-image: url('/img/bgimg.PNG')">
    <nav class="navbar navbar-expand-lg navbar-light bg-light mb-4 shadow-sm ">
        <a class="navbar-brand" th:href="@{/}">TASHILATE-ENSAJ</a>
        <ul class="navbar-nav ml-auto">
            <li class="nav-item">
                <span sec:authentication="principal.username"> User</span>
            </li>
            <li sec:authorize="isAuthenticated()" class="nav-item ml-4 mr-4">
                <a th:href="@{/logout}">Logout</a>
            </li>
        </ul>
    </nav>
    <div class="col-lg-4 formStyle p-4" id="form">
        <h1> Send Money </h1>
        <div class="form-group">
            <label for="cinSender">The CIN of the sender </label> :
            <input type="text" class="form-control" id="cinSender" name="cinSender" autofocus="autofocus" required>
        </div>
        <div class="form-group">
            <label for="nameReceiver">The name of the receiver</label>:
            <input type="text" id="nameReceiver" name="nameReceiver" class="form-control" required/>
        </div>
        <div class="form-group">
            <label for="amount">The amount of money to send</label>:
            <div class="input-group">
                <input type="text" class="form-control" id="amount" name="amount" required/>
                <div class="input-group-append">
                    <span class="input-group-text">DH</span>
                </div>
            </div>
        </div>
        <div class="form-group">
            <button type="submit" name="send" id="send" class="btn btn-outline-success btn-block mb-4" value="Send">Send</button>
        </div>
    </div>
    <div class="col-lg-4 formStyle p-4 text-center" id="code" >
    </div>
    <script src="/js/jquery-3.3.1.min.js"></script>
    <script src="/js/send_money.js"></script>
```

Figure 277:send_money.html

send_money.js

```
$(document).ready(function(){
    var cinSender = $("#cinSender");
    var nameReceiver = $("#nameReceiver");
    var amount = $("#amount");
    $("#code").hide();
    $('#send').click(function() {
        $.ajax({
            url: 'http://localhost:8080/service-transfer/sendMoney',
            data: JSON.stringify({cinSender: cinSender.val(), nameReceiver: nameReceiver.val(), amount: amount.val()}),
            headers: { 'Access-Control-Allow-Origin': '*' },
            type: 'POST',
            dataType: "json",
            contentType:"application/json; charset=utf-8",
            async: false,
            success: function(data, textStatus, jqXHR) {
                remplir(data);
                if (!("Notification" in window)) {
                    alert("This browser does not support desktop notification");
                  }
                  else if (Notification.permission === "granted") {
                    var notification = new Notification("MONEY SENT SUCCESSFULLY!");
                  }
                  else if (Notification.permission !== "denied") {
                    Notification.requestPermission().then(function (permission) {
                      if (permission === "granted") {
                        var notification = new Notification("MONEY SENT SUCCESSFULLY!");
                      }
                    });
                  }
            },
            error: function(jqXHR, textStatus, errorThrown) {
                console.log(textStatus);
            }
        });
    });
    function remplir(data) {
        var ligne="";
        ligne+='<img src="/img/success.png" height="200" width="200"/>';
        ligne+='<h3 color="green">DONE SUCCESSFULLY!!</h3>';
        ligne+='<div class="alert alert-info">The code of the transfer: <strong>'+data+'</strong></div>';
        $('#code').html(ligne);
        $("#code").show();
        $("#form").hide();
    }
```

Figure 278:send_money.html continuation code

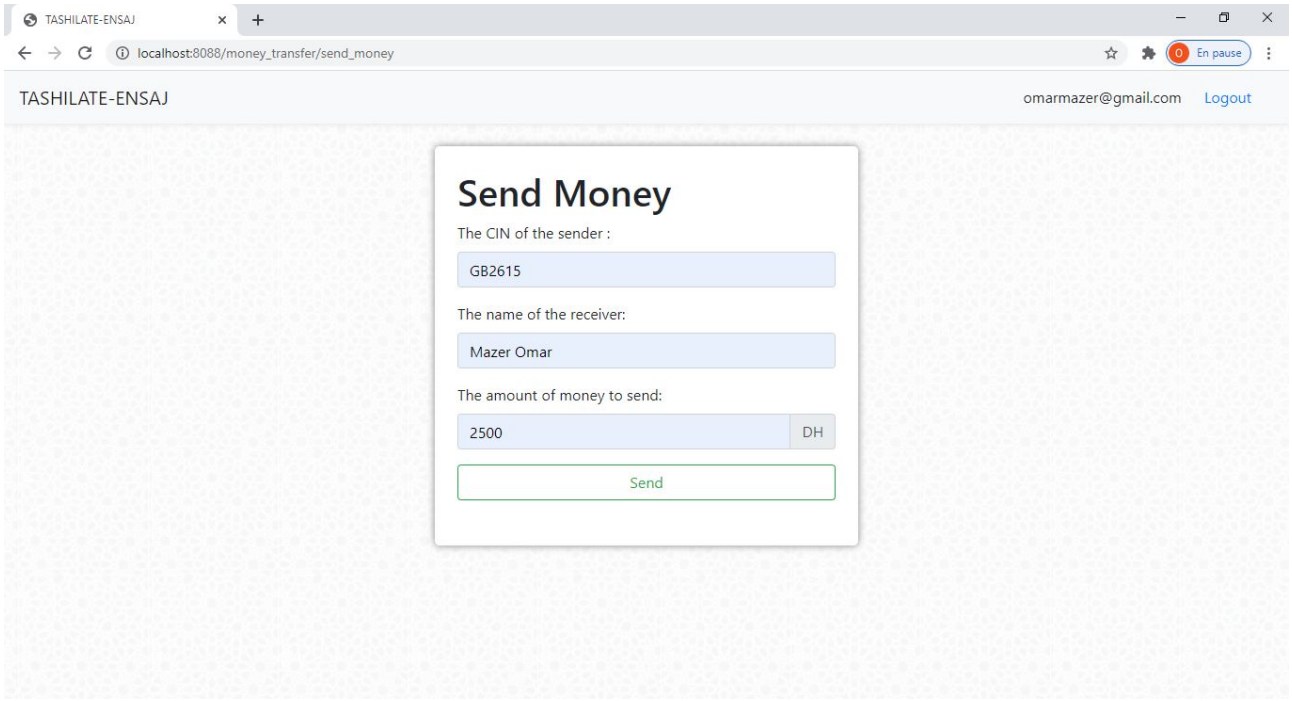


Figure 279:Fill the form of send money

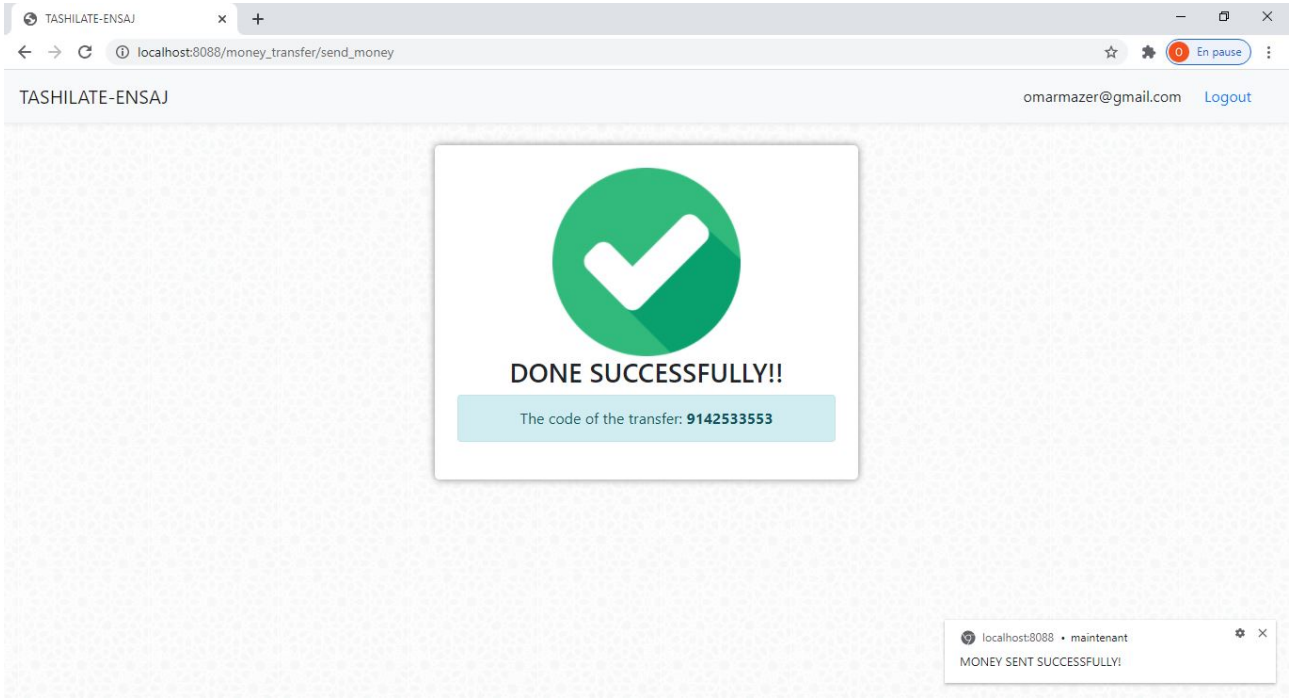


Figure 280:Generation of the code of the transfer

Receive Money

receive_money.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org" xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-springsecurity3">
<head>
    <meta charset="ISO-8859-1">
    <title>TASHILATE-ENSAJ</title>
    <link rel="stylesheet" th:href="@{/css/bootstrap.min.css}" type="text/css">
    <link rel="stylesheet" th:href="@{/css/custom.css}" type="text/css">
</head>
<body style="background-image: url('/img/bgimg.PNG')">
    <nav class="navbar navbar-expand-lg navbar-light bg-light mb-4 shadow-sm ">
        <a class="navbar-brand" th:href="@{/}">TASHILATE-ENSAJ</a>
        <ul class="navbar-nav ml-auto">
            <li class="nav-item">
                <span sec:authentication="principal.username"> User</span>
            </li>
            <li sec:authorize="isAuthenticated()" class="nav-item ml-4 mr-4">
                <a th:href="@{/Logout}">Logout</a>
            </li>
        </ul>
    </nav>
    <div class="col-lg-4 formStyle p-4" id="form">
        <h1> Receive Money </h1>
        <div class="form-group">
            <label for="code">The code of the transfer</label>:
            <input type="text" id="code" name="code" class="form-control" autofocus="autofocus" required/>
        </div>
        <div class="form-group">
            <label for="cinReceiver">The CIN of the receiver </label> :
            <input type="text" class="form-control" id="cinReceiver" name="cinReceiver" required>
        </div>
        <div class="form-group">
            <button type="submit" name="receive" id="receive"   class="btn btn-outline-success btn-block mb-4" value="Receive">Receive</button>
        </div>
        <div id="erreur">
        </div>
    </div>
    <div class="col-lg-4 formStyle p-4 text-center" id="result" >
    </div>
    <script src="/js/jquery-3.3.1.min.js"></script>
    <script src="/js/receive_money.js"></script>
</body>
</html>
```

Figure 281:receive_money.html

receive_money.js

```
$(document).ready(function(){
    var code = $("#code");
    var cinReceiver = $("#cinReceiver");
    $("#result").hide();
    $("#erreur").hide();
    $('#receive').click(function() {
        $.ajax({
            url: 'http://localhost:8080/service-transfer/receiveMoney',
            data: JSON.stringify({cinReceiver: cinReceiver.val(), code: code.val()}),
            headers: { 'Access-Control-Allow-Origin': '*' },
            type: 'POST',
            dataType: "json",
            contentType:"application/json; charset=utf-8",
            async: false,
            success: function(data, textStatus, jqXHR) {
                remplir(data);
                if (!("Notification" in window)) {
                    alert("This browser does not support desktop notification");
                  }
                  else if (Notification.permission === "granted") {
                    var notification = new Notification("MONEY RECEIVED SUCCESSFULLY!");
                  }
                  else if (Notification.permission !== "denied") {
                    Notification.requestPermission().then(function (permission) {
                      if (permission === "granted") {
                        var notification = new Notification("MONEY RECEIVED SUCCESSFULLY!");
                      }
                    });
                  }
            },
            error: function(jqXHR, textStatus, errorThrown) {
                console.log(textStatus);
                remplirErreur();
            }
        });
    });
```

Figure 282:receive_money.js

```
37      function remplirErreur() {
38          $('#erreur').html('<div class="alert alert-danger text-center">This Transfer does not exist!!</div>');
39          $("#erreur").show();
40      }
41      function remplir(data) {
42          var ligne="";
43          ligne+='<img src="/img/success.png" height="200" width="200"/>';
44          ligne+='<h3 color="green">DONE SUCCESSFULLY!!</h3>';
45          ligne+='<div class="alert alert-info text-left">The name of the receiver: <strong>'+data.nameReceiver+'</strong><br/>';
46          ligne+='The CIN of the receiver: <strong>'+data.cinReceiver+'</strong><br/>';
47          ligne+='The CIN of the sender: <strong>'+data.cinSender+'</strong><br/>';
48          ligne+='The amount received: <strong>'+data.amount+' DH</strong><br/>';
49          ligne+='The date of sending money: <strong>'+data.dateSending+'</strong><br/>';
50          ligne+='The date of the receiving: <strong>'+data.dateReceiving+'</strong><br/>';
51          ligne+='</div>';
52          $('#result').html(ligne);
53          $("#result").show();
54          $("#form").hide();
55      }
56  });
```

Figure 283:receive_money.js code continuation

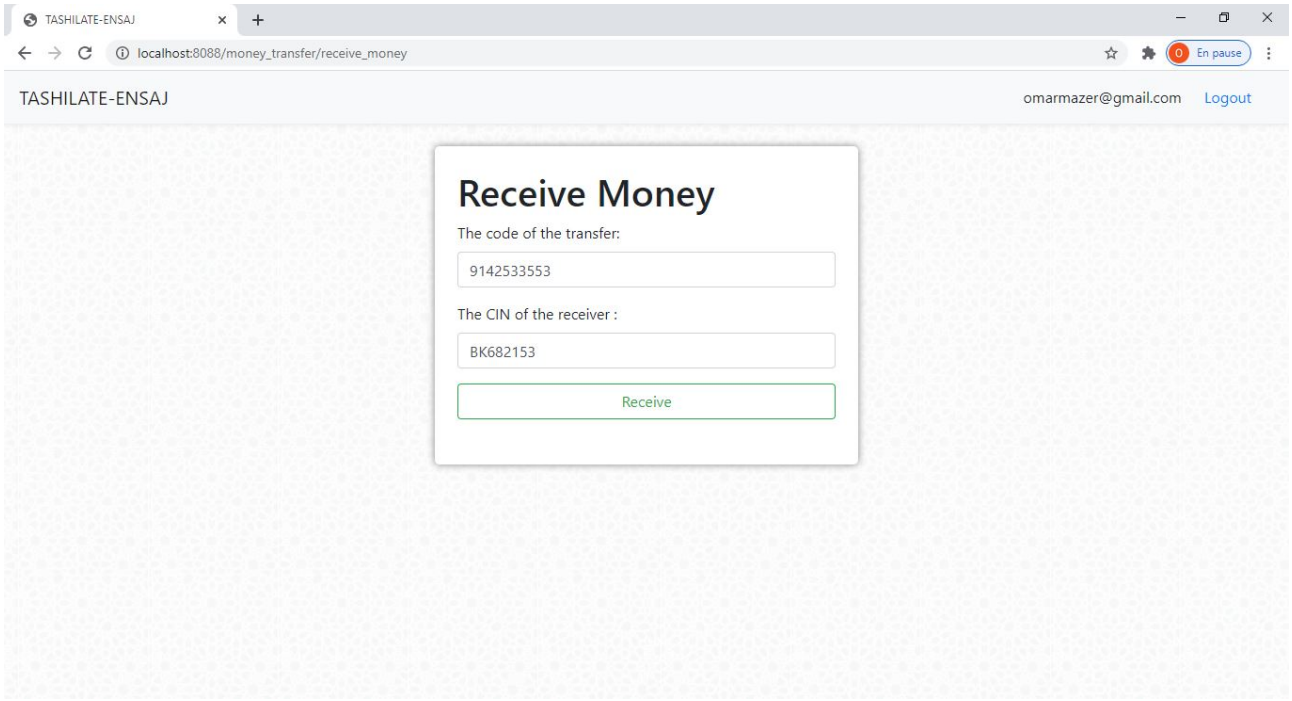


Figure 284:Fill the form of receive money

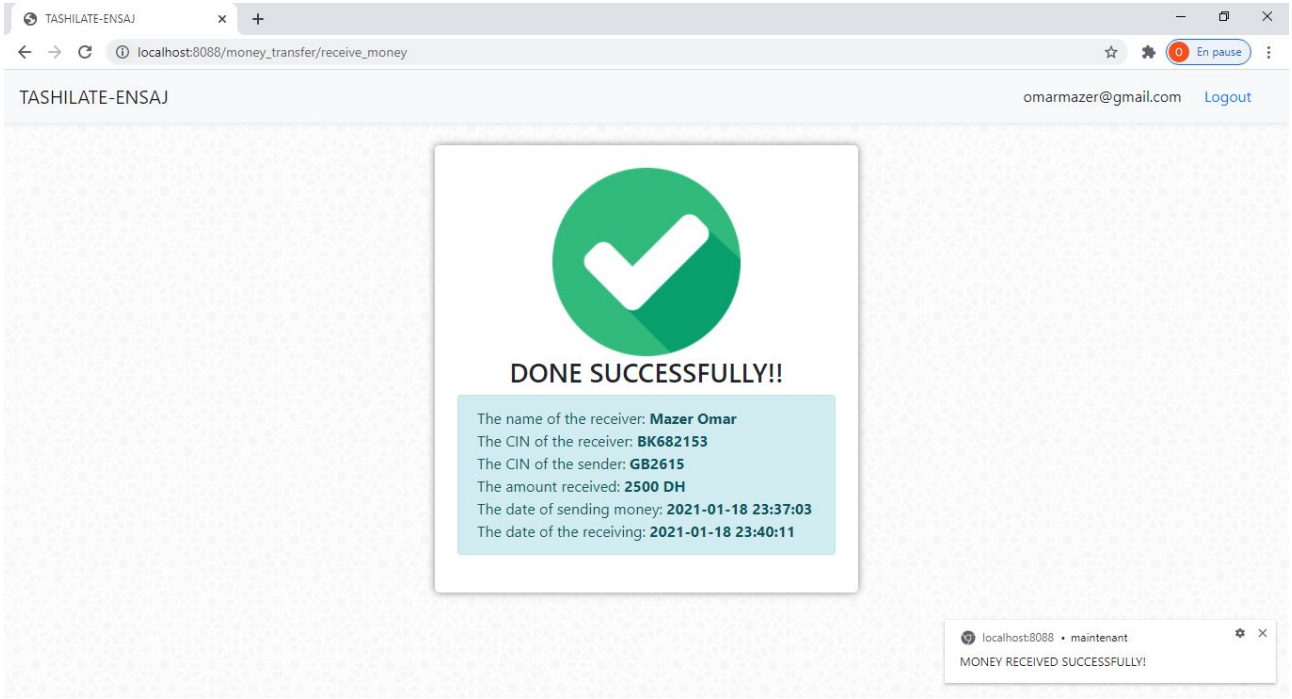


Figure 285:receive money successfully

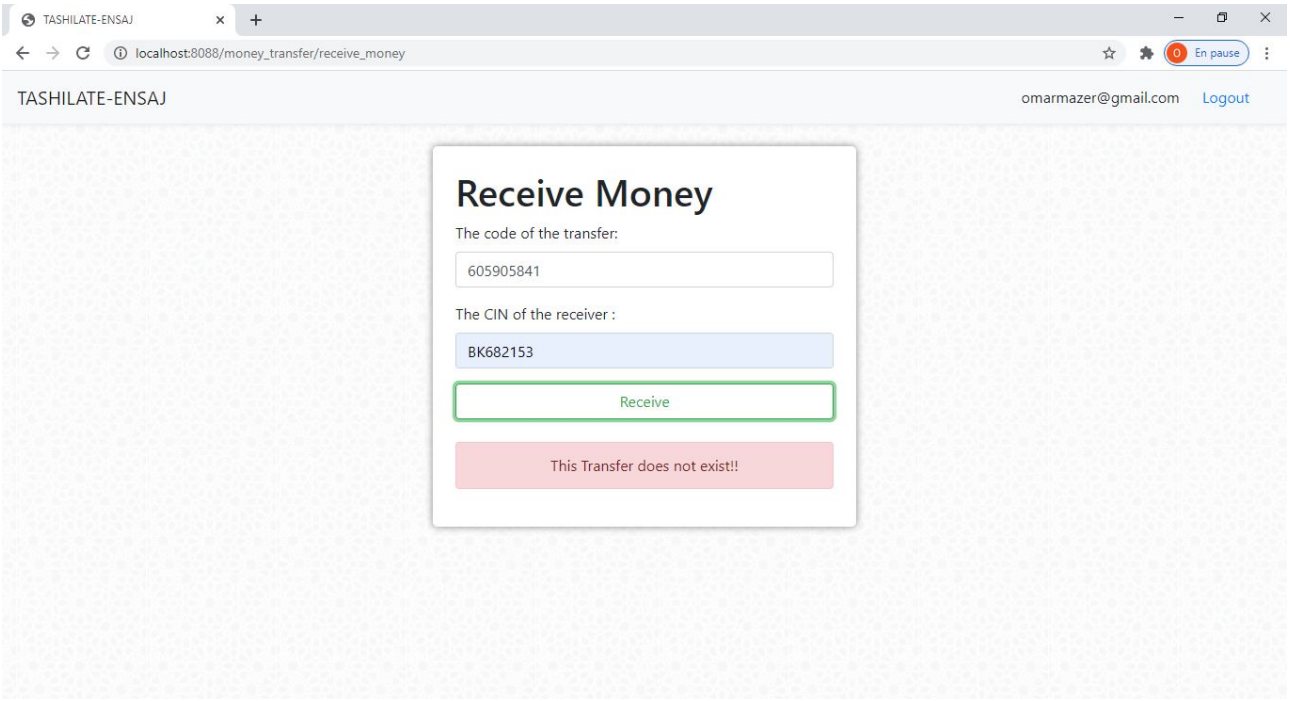


Figure 286:Case of data validation-Confirmation message

4.2.5 Phone Recharge

phone_recharge.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org" xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-springsecurity3">
<head>
    <meta charset="ISO-8859-1">
    <title>TASHILATE-ENSAJ</title>
    <link rel="stylesheet" th:href="@{/css/bootstrap.min.css}" type="text/css">
    <link rel="stylesheet" th:href="@{/css/custom.css}" type="text/css">
</head>
<body style="background-image: url('/img/bgimg.PNG')">
    <nav class="navbar navbar-expand-lg navbar-light bg-light mb-4 shadow-sm ">
        <a class="navbar-brand" th:href="@{/}">TASHILATE-ENSAJ</a>
        <ul class="navbar-nav ml-auto">
            <li class="nav-item">
                <span sec:authentication="principal.username"> User</span>
            </li>
            <li sec:authorize="isAuthenticated()" class="nav-item ml-4 mr-4">
                <a th:href="@{/logout}">Logout</a>
            </li>
        </ul>
    </nav>
    <div class="col-lg-4 formStyle p-4" id="form">
        <h1> Phone Recharge </h1>
        <div class="form-group">
            <table class="text-center d-flex justify-content-center p-2">
                <tr>
                    <td><img class="m-1" src="/img/iam.jpg" height="100" width="150"/></td>
                    <td><img class="m-1" src="/img/orange.png" height="100" width="100"/></td>
                    <td><img class="m-1" src="/img/inwi.jpeg" height="100" width="100"/></td>
                </tr>
                <tr>
                    <td><input class="form-check-input" type="radio" name="operator" id="iam" value="Maroc Telecom" checked></td>
                    <td><input class="form-check-input" type="radio" name="operator" id="orange" value="Orange"></td>
                    <td><input class="form-check-input" type="radio" name="operator" id="inwi" value="Inwi"></td>
                </tr>
            </table>
        </div>
        <div class="form-group">
            <label for="numTel">The phone number</label>:
            <input type="text" id="numTel" name="numTel" class="form-control" placeholder="0612345678" autofocus="autofocus" required/>
        </div>
```

Figure 287:phone_recharge.html

phone_recharge.html

```
            <input type="text" id="numTel" name="numTel" class="form-control" placeholder="0612345678" autofocus="autofocus" required/>
        </div>
        <div class="form-group">
            <label for="amount">The amount of money </label> :
            <div class="input-group">
                <select class="form-control" id="amount" name="amount">
                  <option selected value="5">5</option>
                  <option value="10">10</option>
                  <option value="20">20</option>
                  <option value="25">25</option>
                  <option value="50">50</option>
                  <option value="100">100</option>
                  <option value="200">200</option>
                </select>
                <div class="input-group-append">
                    <span class="input-group-text">DH</span>
                </div>
            </div>
        </div>
        <div class="form-group d-flex justify-content-center p-2">
            <div class="form-check form-check-inline">
              <input class="form-check-input" type="radio" name="typeRecharge" id="phone" value="Phone" checked>
              <label class="form-check-label" for="typeRecharge">Phone</label>
            </div>
            <div class="form-check form-check-inline">
              <input class="form-check-input" type="radio" name="typeRecharge" id="internet" value="Internet">
              <label class="form-check-label" for="typeRecharge">Internet</label>
            </div>
            <div class="form-check form-check-inline">
              <input class="form-check-input" type="radio" name="typeRecharge" id="special_offer" value="Special offer">
              <label class="form-check-label" for="typeRecharge">Special offer</label>
            </div>
        </div>
        <div class="form-group">
            <button type="submit" name="recharge" id="recharge" class="btn btn-outline-success btn-block mb-4" value="Recharge">Recharge</button>
        </div>
        <div id="erreur">
        </div>
    </div>
    <div class="col-lg-4 formStyle p-4 text-center" id="result">
    </div>
    <script src="/js/jquery-3.3.1.min.js"></script>
    <script src="/js/phone_recharge.js"></script>
</body>
</html>
```

Figure 288:phone_recharge.html

phone_recharge.js

```
$(document).ready(function(){
    var numTel = $("#numTel");
    var amount = $("#amount");
    $("#result").hide();
    $("#erreur").hide();
    $('#recharge').click(function() {
        $.ajax({
            url: 'http://localhost:8080/service-recharge/recharge',
            data: JSON.stringify({numTel: numTel.val(), typeRecharge: $("input[name='typeRecharge']:checked").val(),amount: amount.val(),
            headers: { 'Access-Control-Allow-Origin': '*' },
            type: 'POST',
            dataType: "json",
            contentType:"application/json; charset=utf-8",
            async: false,
            success: function(data, textStatus, jqXHR) {
                remplir(data);
                if (!("Notification" in window)) {
                    alert("This browser does not support desktop notification");
                  }
                  else if (Notification.permission === "granted") {
                    var notification = new Notification("PHONE RECHARGED SUCCESSFULLY!");
                  }
                  else if (Notification.permission !== "denied") {
                    Notification.requestPermission().then(function (permission) {
                      if (permission === "granted") {
                        var notification = new Notification("PHONE RECHARGED SUCCESSFULLY!");
                      }
                    });
                  }
            },
            error: function(jqXHR, textStatus, errorThrown) {
                console.log(textStatus);
                remplirErreur();
            }
        });
    });
```

Figure 289:phone_recharge.js

```
    function remplirErreur() {
        $('#erreur').html('<div class="alert alert-danger text-center">This phone number is not correct!!</div>');
        $("#erreur").show();
    }
    function remplir(data) {
        var ligne="";
        ligne+='<img src="/img/success.png" height="200" width="200"/>';
        ligne+='<h3 color="green">DONE SUCCESSFULLY!!</h3>';
        ligne+='<div class="alert alert-info text-left">The phone numberr: <strong>'+data.numTel+'</strong><br/>';
        ligne+='The amount: <strong>'+data.amount+' DH</strong><br/>';
        ligne+='The operator: <strong>'+data.operator+'</strong><br/>';
        ligne+='The type of the recharge: <strong>'+data.typeRecharge+'</strong><br/>';
        ligne+='The date of the recharge: <strong>'+data.date+'</strong><br/>';
        ligne+='</div>';
        $('#result').html(ligne);
        $("#result").show();
        $("#form").hide();
    }
});
```

Figure 290:phone_recharge.js code continuation

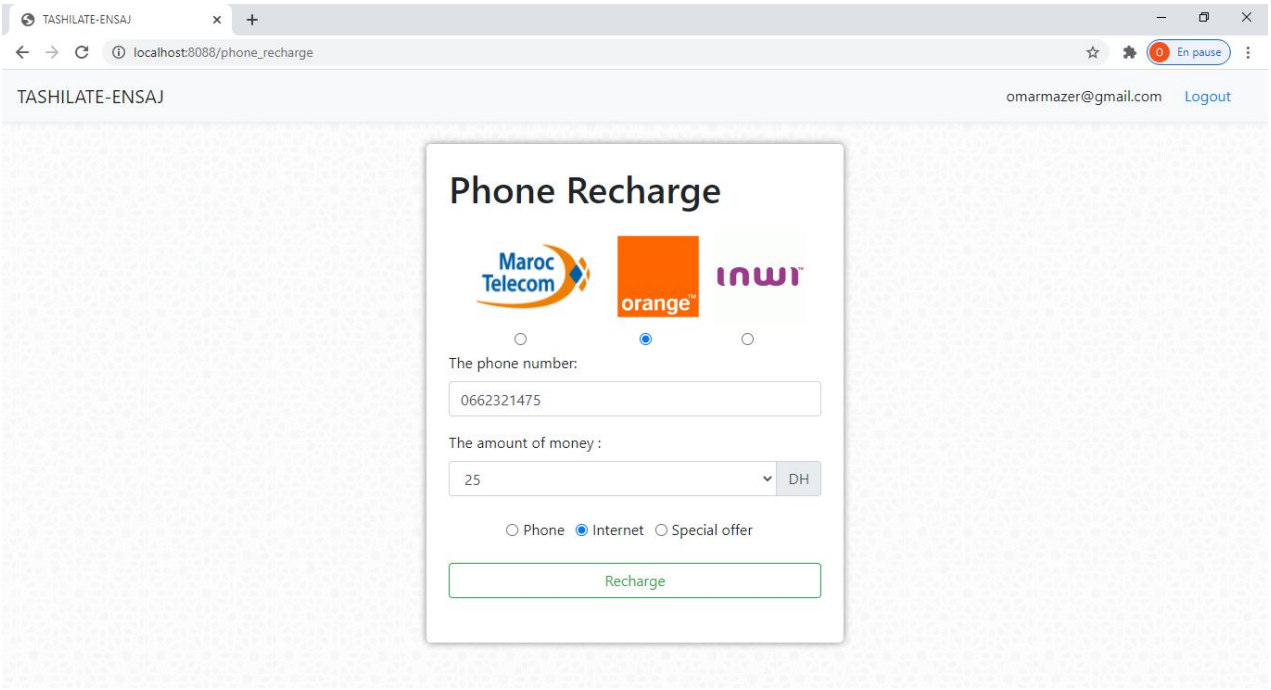


Figure 291:phone recharge in the browser

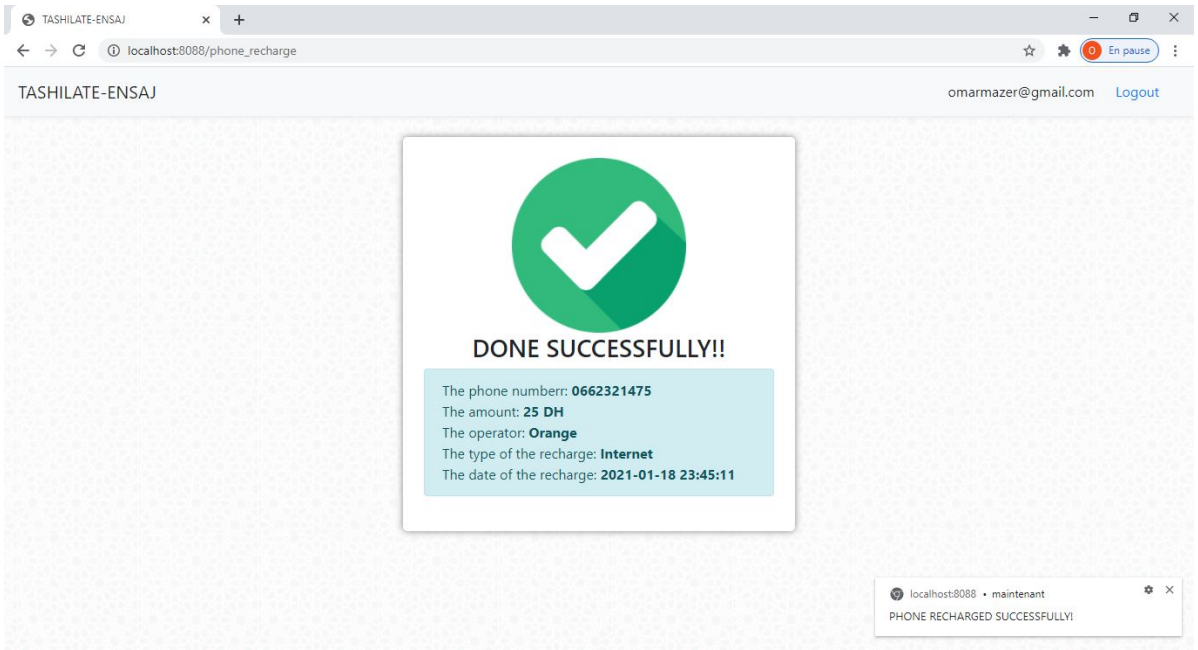


Figure 292:phone recharge done successfully

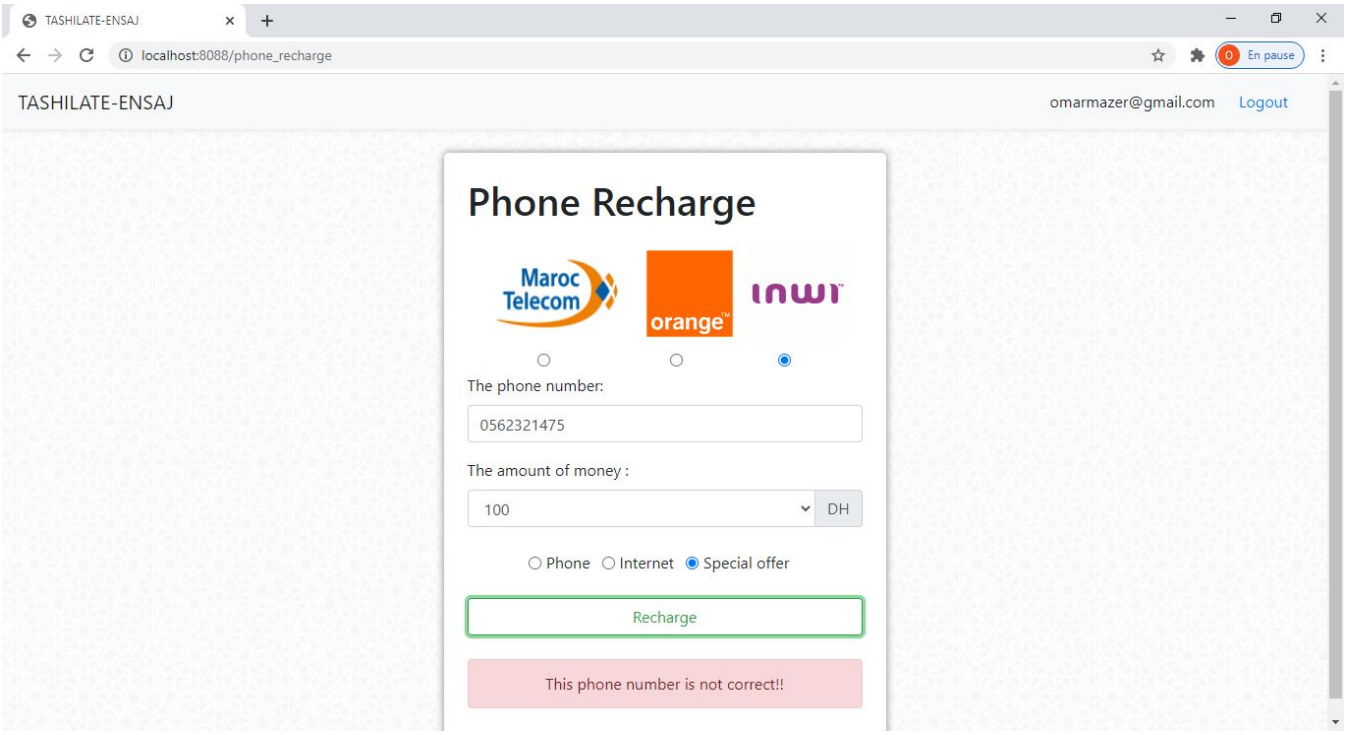


Figure 293:phone recharge data validation-error message

4.2.6 Water & electricity Payment

we_payment.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org" xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-springsecurity3">
<head>
    <meta charset="ISO-8859-1">
    <title>TASHILATE-ENSAJ</title>
    <link rel="stylesheet" th:href="@{/css/bootstrap.min.css}" type="text/css">
    <link rel="stylesheet" th:href="@{/css/custom.css}" type="text/css">
</head>
<body style="background-image: url('/img/bgimg.PNG')">
    <nav class="navbar navbar-expand-lg navbar-light bg-light mb-4 shadow-sm ">
        <a class="navbar-brand" th:href="@{/}">TASHILATE-ENSAJ</a>
        <ul class="navbar-nav ml-auto">
            <li class="nav-item">
                <span sec:authentication="principal.username"> User</span>
            </li>
            <li sec:authorize="isAuthenticated()" class="nav-item ml-4 mr-4">
                <a th:href="@{/logout}">Logout</a>
            </li>
        </ul>
    </nav>
    <div class="col-lg-4 formStyle p-4" id="form">
        <h1> W/E Payment </h1>
        <div class="form-group">
            <label for="contractNbr">The contract number</label>:
            <input type="text" id="contractNbr" name="contractNbr" class="form-control" autofocus="autofocus" required/>
        </div>
        <div class="form-group">
            <label for="agency">The W/E provider </label> :
            <select class="form-control" id="agency" name="agency">
                <option selected value="LYDEC">LYDEC</option>
                <option value="RADEEJ">RADEEJ</option>
                <option value="REDAL">REDAL</option>
                <option value="RADEEC">RADEEC</option>
                <option value="RADEEO">RADEEO</option>
                <option value="RADEET">RADEET</option>
            </select>
        </div>
        <div class="form-group">
            <button type="submit" name="search" id="search" class="btn btn-outline-success btn-block mb-4" value="Search">Search</button>
        </div>
```

Figure 294:Water & electricity Payment html

```
        <div id="erreur">
        </div>
    </div>
    <div class="col-lg-4 formStyle p-4 text-center" id="bills">
    </div>
    <div class="col-lg-4 formStyle p-4 text-center" id="result">
        <img src="/img/success.png" height="200" width="200"/>
        <h3 color="green">DONE SUCCESSFULLY!!</h3>
    </div>
    <script src="/js/jquery-3.3.1.min.js"></script>
    <script src="/js/we_payment.js"></script>
</body>
</html>
```

Figure 295:Water & electricity Payment html continuation

we_payment.js

```
$(document).ready(function(){
    var contractNbr = $("#contractNbr");
    var agency = $("#agency");
    $("#bills").hide();
    $("#result").hide();
    $("#erreur").hide();
    $('#search').click(function() {
        $.ajax({
            url: 'http://localhost:8080/service-payment/allBills',
            data: JSON.stringify({contractNbr: contractNbr.val(),agency: agency.val()}),
            headers: { 'Access-Control-Allow-Origin': '*' },
            type: 'POST',
            dataType: "json",
            contentType:"application/json; charset=utf-8",
            async: false,
            success: function(data, textStatus, jqXHR) {
                remplirTableau(data);
            },
            error: function(jqXHR, textStatus, errorThrown) {
                console.log(textStatus);
                remplirErreur();
            }
        });
    });
    function remplirErreur() {
        $('#erreur').html('<div class="alert alert-danger text-center">There is no bill for this contract number!!</div>');
        $("#erreur").show();
    }
    function remplir(data) {
        var ligne="";
        ligne+='<div class="alert alert-info text-left mb-2">';
        ligne+='The contract number: <strong>'+data.contractNbr+'</strong><br/>';
        ligne+='The W/E provider: <strong>'+data.agency+'</strong><br/>';
        ligne+='The month: <strong>'+data.month +'\/' +data.year+'</strong><br/>';
        ligne+='The amount: <strong>'+data.amount+' DH</strong><br/>';
        ligne+='The date: <strong>'+data.date+'</strong><br/>';
        ligne+='</div>';
        $('#result').html($('#result').html()+ligne);
        $("#result").show();
        $("#bills").hide();
    }
```

Figure 296:we_payment.js

```
41      }
42      function pay(id){
43          $.ajax({
44              url: 'http://localhost:8080/service-payment/pay',
45              data: JSON.stringify({id: id}),
46              headers: { 'Access-Control-Allow-Origin': '*' },
47              type: 'POST',
48              dataType: "json",
49              contentType:"application/json; charset=utf-8",
50              async: false,
51              success: function(data, textStatus, jqXHR) {
52                  remplir(data);
53                  if (!("Notification" in window)) {
54                      alert("This browser does not support desktop notification");
55                    }
56                    else if (Notification.permission === "granted") {
57                      var notification = new Notification("PAYMENT DONE SUCCESSFULLY!");
58                    }
59                    else if (Notification.permission !== "denied") {
60                      Notification.requestPermission().then(function (permission) {
61                        if (permission === "granted") {
62                          var notification = new Notification("PAYMENT DONE SUCCESSFULLY!");
63                        }
64                      });
65                    }
66              },
67              error: function(jqXHR, textStatus, errorThrown) {
68                  console.log(textStatus);
69              }
70          });
71      }
```

Figure 297:we_payment.js continuation

```
72      function remplirTableau(data) {
73          var ligne="";
74          ligne+='<table class="table table-striped table-bordered"><thead><tr><th scope="col"></th><th scope="col">Amount (DH)</th><th scope="col">Month</th></tr></thead><tbody>';
75          for (i = 0; i < data.length; i++) {
76              ligne += '<tr class="text-right"><th scope="row"><input class="form-check-input id" type="checkbox" name="id" value="'+data[i].id +'"></th>';
77              ligne += '<td>' + data[i].amount + '</td>';
78              ligne += '<td>' + data[i].month +'\/' +data[i].year + '</tr>';
79          }
80          ligne+='</tbody></table>';
81          ligne+='<div class="alert alert-info">The total to pay : <strong><span id="total">0.00</span>DH</strong></div>';
82          ligne+='<div class="form-group"><button type="submit" name="pay" id="pay" class="btn btn-primary btn-block mb-4" value="Pay">Pay</button></div>';
83          $('#bills').html(ligne);
84          $("#bills").show();
85          $("#form").hide();
86          $(document).on('change', '.id', function() {
87              if($(this).is(':checked') == true){
88                  var montant = parseFloat($(this).closest('tr').find('td').eq(0).text());
89                  $("#total").html((parseFloat($("#total").html())+montant).toFixed(2));
90
91              }else{
92                  var montant = parseFloat($(this).closest('tr').find('td').eq(0).text());
93                  $("#total").html((parseFloat($("#total").html())-montant).toFixed(2));
94              }
95          })
96          $(document).on('click', '#pay', function() {
97
98              $('input[name="id"]:checked').each(function() {
99                  pay(this.value);
00              });
01          })
02      }
03  });
```

Figure 298:remplirTableau() method

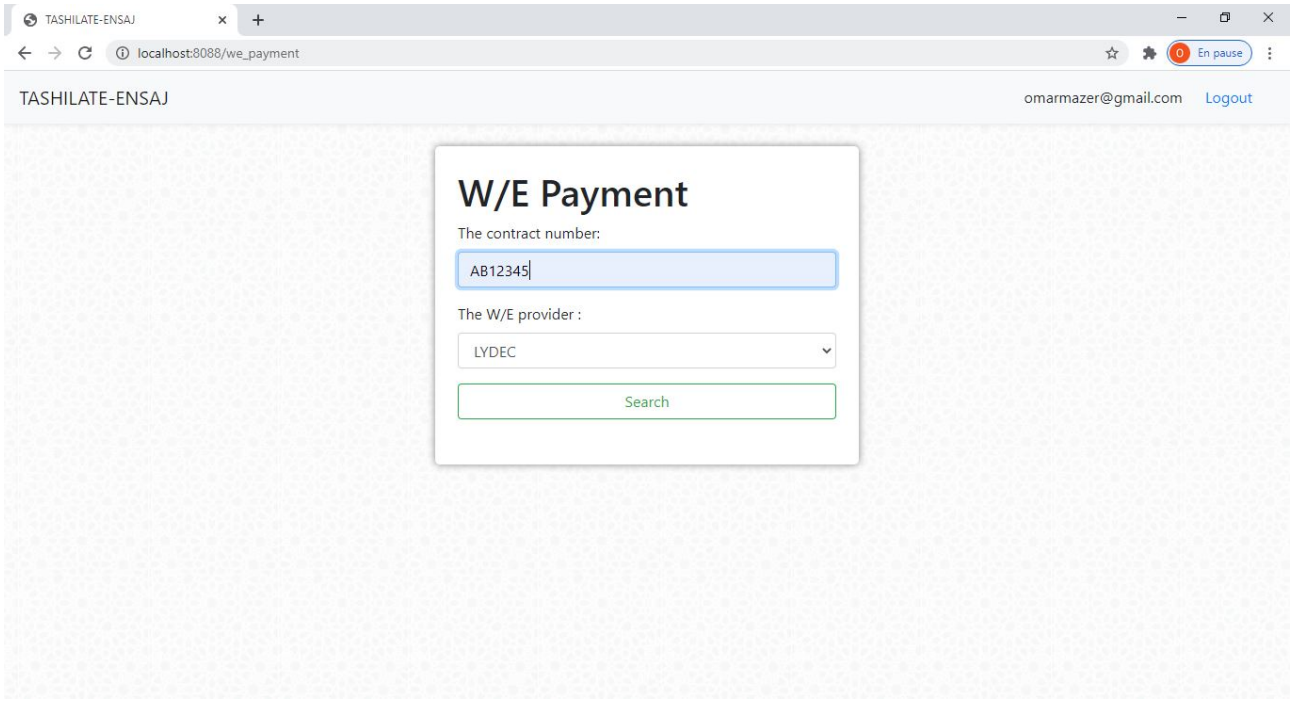


Figure 299:W/E Payment in the browser

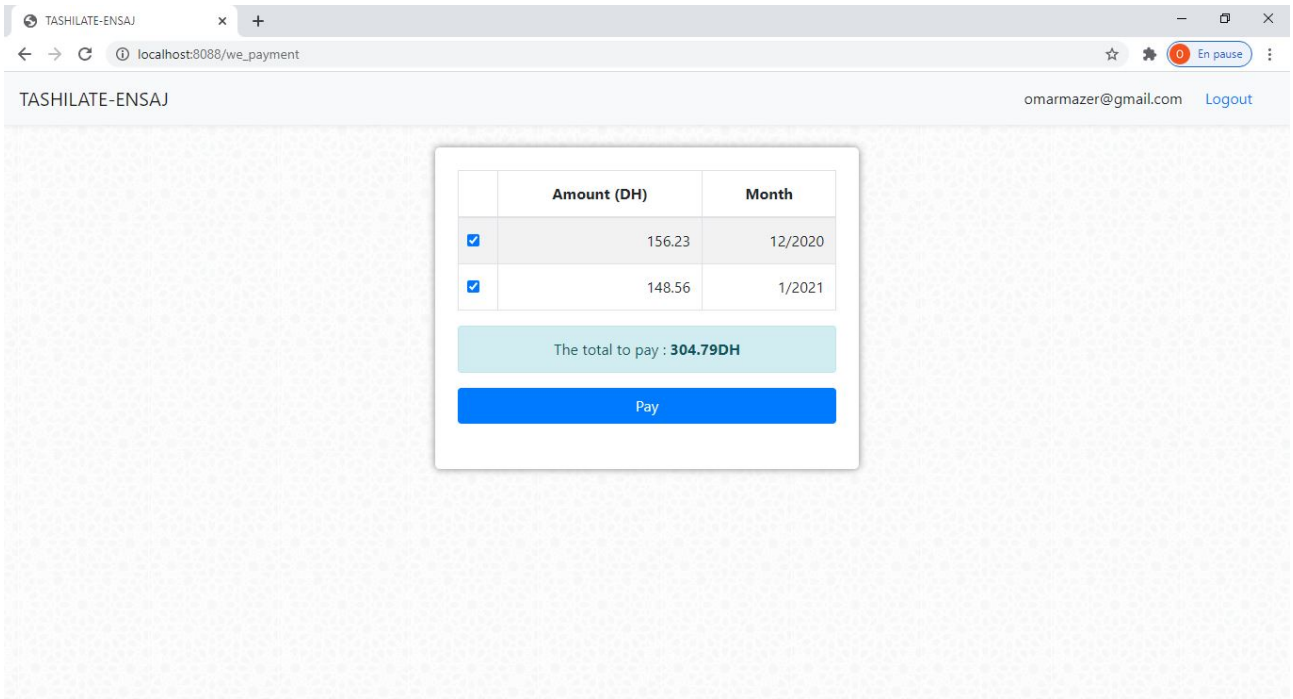


Figure 300:Table w/e payment

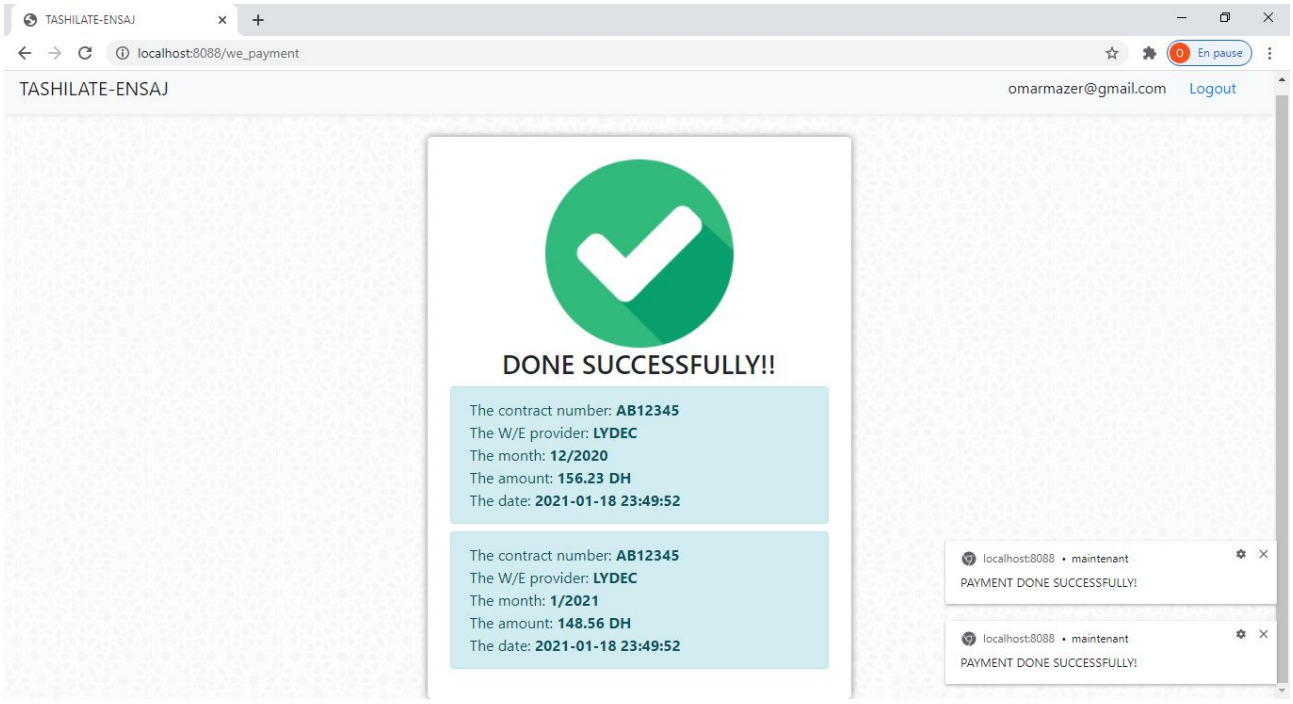


Figure 301:w/e payment done successfully

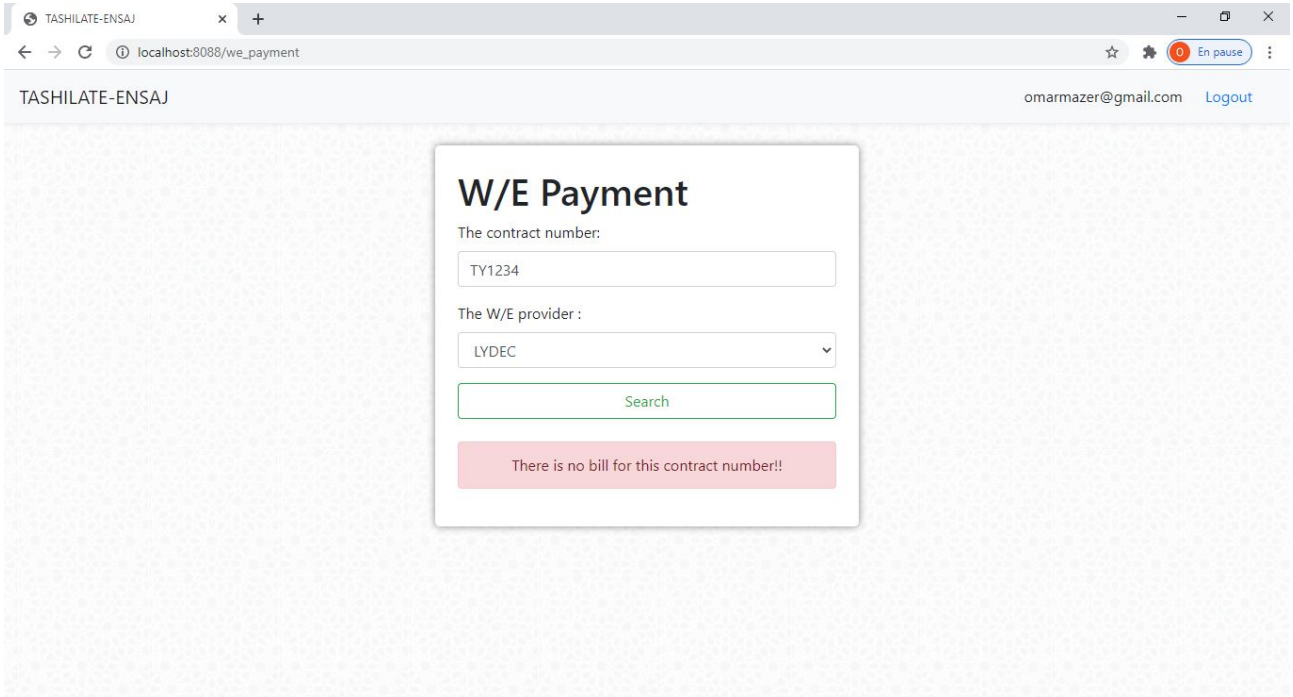


Figure 302:W/E payment in the browser

4.2.7 Visitor Counter feature

1. Create **Counter** class under the package **beans**

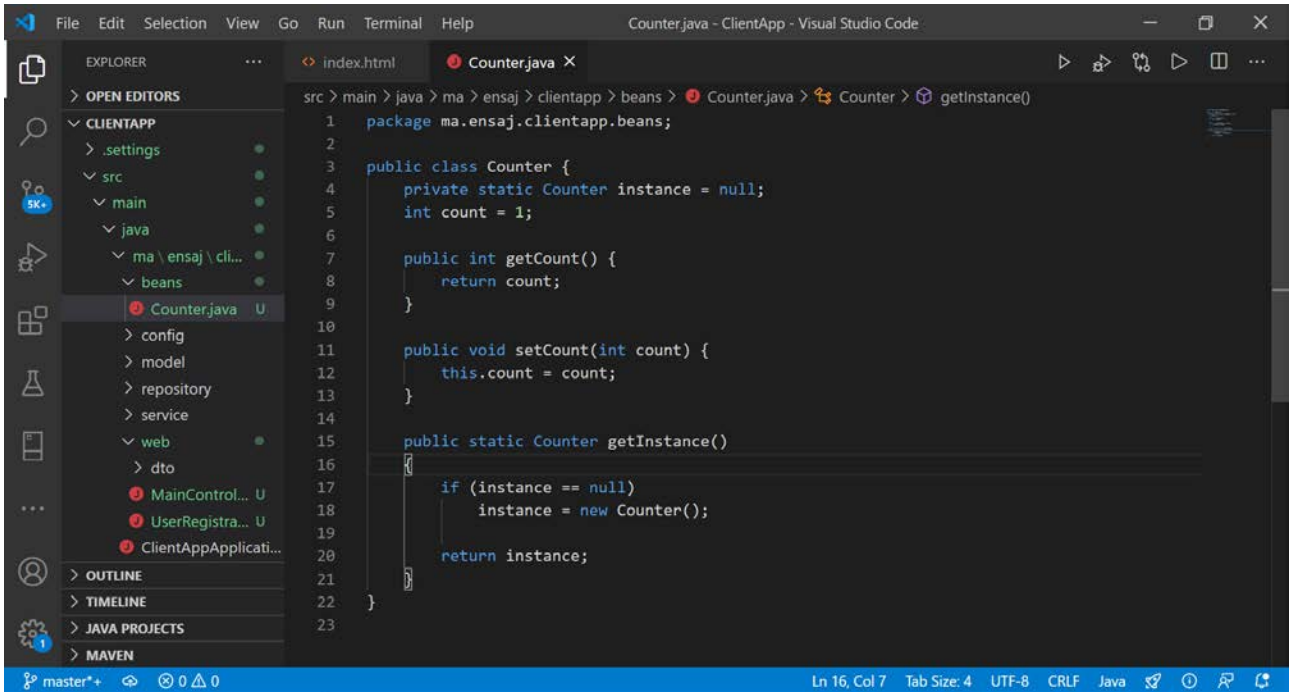


Figure 303:Create Counter class under the package beans:

2. Add **@Bean** annotation for counter method in the class **ClientAppApplication.java**

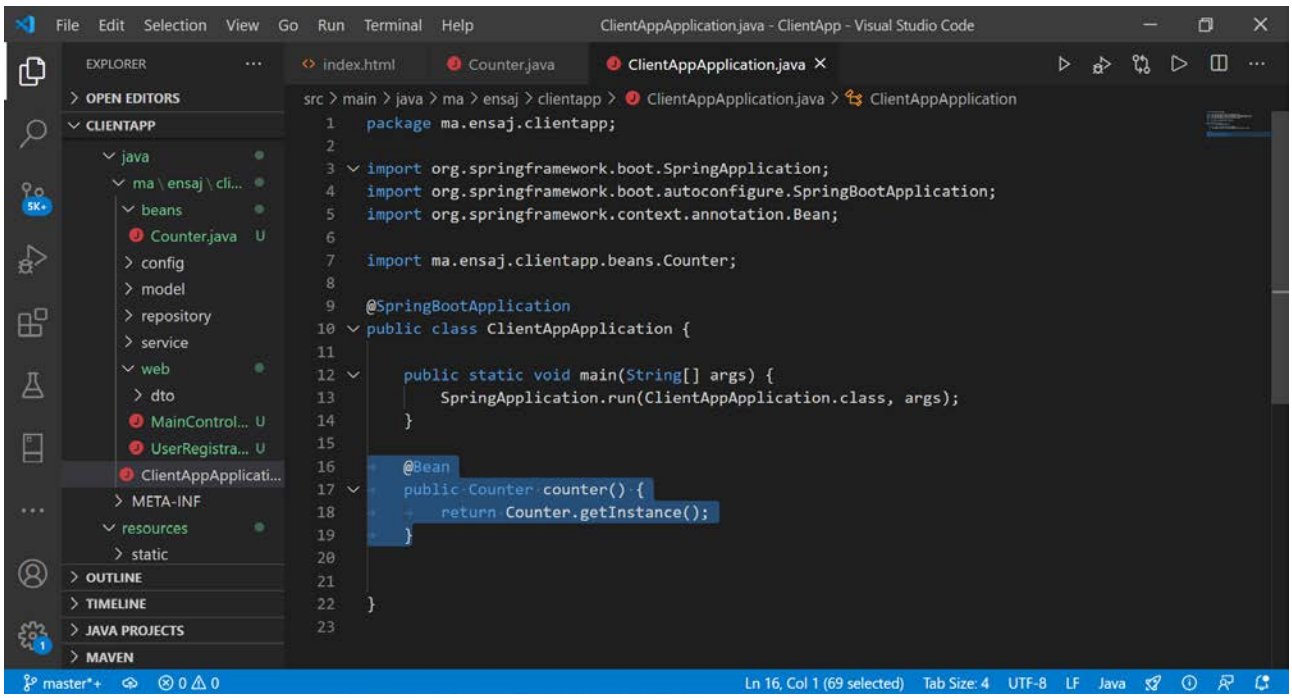


Figure 304:Add @Bean annotation for counter method in the class ClientAppApplication.java

3. Let’s run our app to check if the visitor counter feature works well

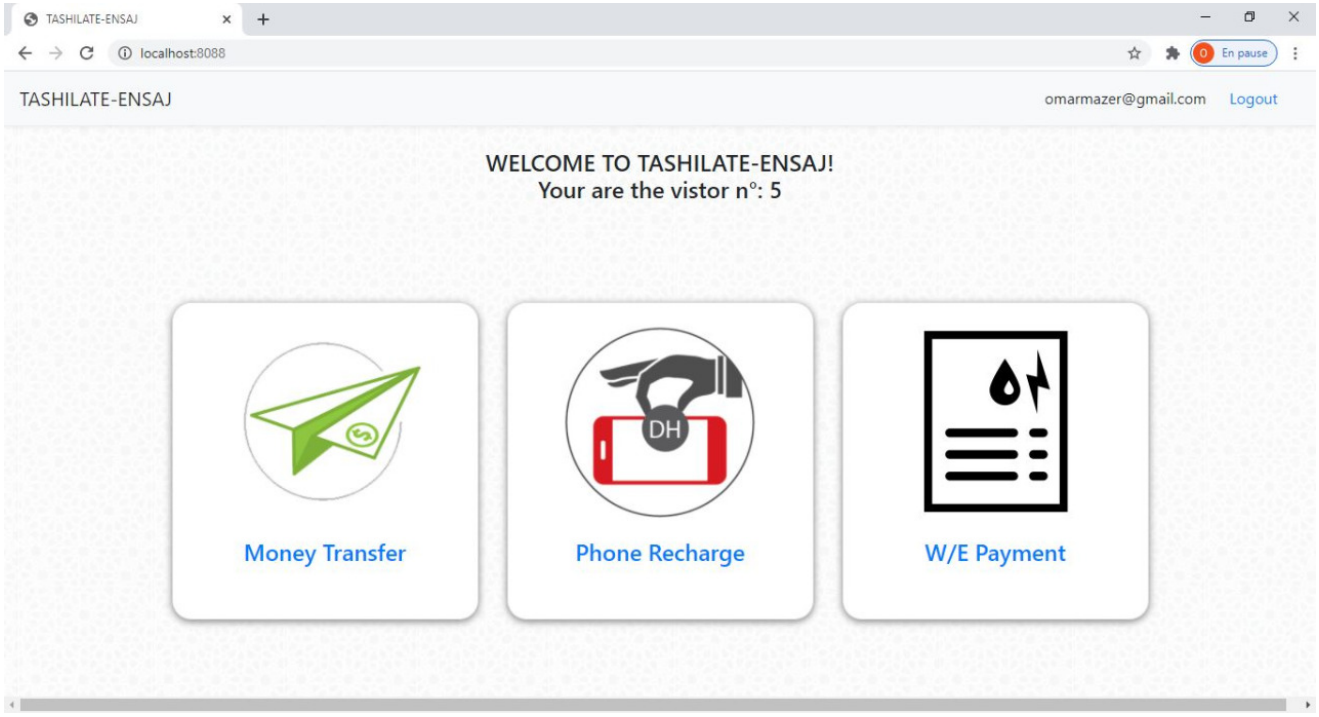


Figure 305:Visitor counter in the browser

Conclusion

In this chapter we have presented in detail the development of our system frontend and backend step by step with explanation.

5 Conclusion

In this project we learned how to create an application based on micro-service architecture which we are now more than ever aware of its extreme usefulness. We are very proud to reach this level in a few weeks.

We are very proud that our application meet, first of all, the **functional requirements** of the enterprise as well as **technical requirements** such as **performance** (response time, avoid the problem of load build-up and fault tolerance), as well as, **maintenance** our application must be **closed to modification and open to extensions**. In addition, **security** and **distribution.**

In the future we want to develop more our applications by adding some charts in the dashboard for statistical purposes that helps a lot in decision making.