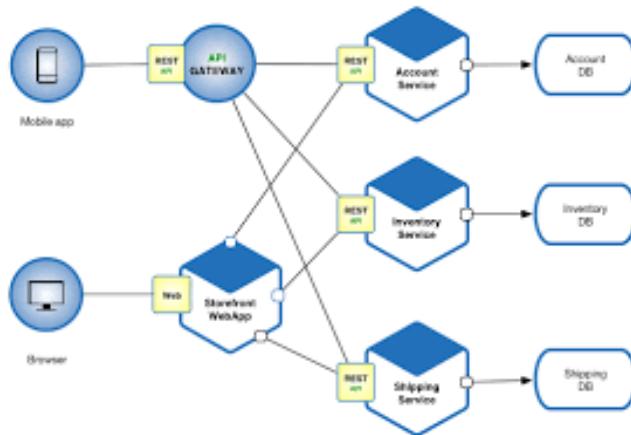




Subject :

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

Abstract	2
Acknowledgement	3
Table of Contents	4
List of figures	6
1 Introduction.....	19
2 Specification requirements.....	20
Introduction.....	20
2.1 Specification of requirements	20
2.1.1 Functional needs :	20
2.1.2 Non-functional needs :	21
Global Use Case Diagram:.....	22
Conclusion	22
3 Tools	23
Introduction.....	23
Conclusion	25
4 Realisation.....	26
Introduction.....	26
4.1 Backend	26
4.1.1 Money transfer Microservice Implementation.....	26
4.1.2 Water/Electricity Bills Microservice Implementation	43
4.1.3 Phone Top Up (recharge) Microservice Implementation	73
4.1.4 Registration/Login Microservice Implementation	97
4.1.5 Configuration Microservice Implementation.....	157

4.1.6	Registration Microservice Implementation.....	172
4.1.7	Proxy Microservice Implementation.....	177
4.2	Frontend.....	183
4.2.1	Login Page	183
4.2.2	Registration	185
4.2.3	Index page	187
4.2.4	Money Transfer.....	190
4.2.5	Phone Recharge.....	200
4.2.6	Water & electricity Payment.....	205
4.2.7	Visitor Counter feature.....	210
	Conclusion	211
5	Conclusion	212

List of figures

Figure 1:Use case diagram.....	22
Figure 2:Initial creation of the application with Spring Initializr	27
Figure 3:Download the project and extract it.....	27
Figure 4:The project after extraction	28
Figure 5:make the project an eclipse project	28
Figure 6:Import it under Eclipse.....	29
Figure 7:Import it under Eclipse.....	30
Figure 8:Import it under Eclipse.....	31
Figure 9:Project structure.....	32
Figure 10:The content of pom.xml	32
Figure 11:application.properties	33
Figure 12:In the application.properties file, configure the database connection parameters.....	34
Figure 13>Create the "MoneyTransferDb" database under MySQL.....	35
Figure 14>Create the “Transfer” class in a sub-package entity of the main package.....	35
Figure 15:@SpringBootApplication	36
Figure 16:mvn spring-boot:run command	36
Figure 17:Check if the "transfer" table is created.....	36
Figure 18:The structure of the table “transfer”	37
Figure 19>Create the "TransferRepository"	38
Figure 20:In the interface TransferRepository add the methods.....	39
Figure 21:Add CRUD, receiveMoney and sendMoney web services to the TransferController	40
Figure 22:Add CRUD, receiveMoney and sendMoney web services to the TransferController	40
Figure 23:sendMoney service test.....	41

Figure 24:The money is sent successfully	41
Figure 25>List of transfer's requests	42
Figure 26:list of money's transfers	42
Figure 27:Receive money by code request	42
Figure 28:The money is received successfully	43
Figure 29:Initial creation of the application with Spring Initializr	44
Figure 30:Download the project and extract it.....	44
Figure 31:The project under the file explorer	45
Figure 32:make the project an eclipse project	46
Figure 33:Import the project under Eclipse.	46
Figure 34:Project structure.....	47
Figure 35: pom.xml dependencies	47
Figure 36.java.version.....	48
Figure 37:@SpringBootApplication	49
Figure 38:application.properties.	49
Figure 39:application.properties	50
Figure 40>Create the "waterelectricity" database under MySQL.....	50
Figure 41>Create the "WaterElectricity" class	51
Figure 42: "WaterElectricity" class.....	52
Figure 43:WaterElectricity entity.....	52
Figure 44:Attributes	53
Figure 45:Generate getters and setters	53
Figure 46:Select all attributes	54
Figure 47:Getters and setters are generated	54
Figure 48:Generate the constructor.....	55
Figure 49:Constructors.....	56

Figure 50:Generate the <code>toString()</code> method	56
Figure 51: <code>toString()</code>	57
Figure 52:We add the validator data dependency in the pom.xml.....	57
Figure 53:We add some persistence annotations.....	58
Figure 54:To start the application, use mvn spring-boot:run command.....	58
Figure 55:We can see now in the cmd Started WePaymentApplication	58
Figure 56:check if the "waterelectricity" table is created.	59
Figure 57:"waterelectricity" table structure	59
Figure 58:Insert a new row to the table waterelectricity.....	60
Figure 59:The new row was added successfully.....	60
Figure 60:Create the "WeRepository" interface	61
Figure 61:WeRepository	62
Figure 62:Create the "WeController" class.....	63
Figure 63:WeController	64
Figure 64:Add CRUD web services to the WeController.....	65
Figure 65:Launch the project again	65
Figure 66>List of water and electricity payments.....	66
Figure 67:200 OK	66
Figure 68:Creation(Add some water and electricity payments)	67
Figure 69:Check if payments were added successfully to the database	68
Figure 70:Update the payment with id 4(agency from RADEEJ to ONE).....	68
Figure 71:The update is well passed.....	69
Figure 72:Delete payment with id 3.....	70
Figure 73:The payment with id 3 was deleted successfully.....	70
Figure 74:Get payments by contract number.....	71
Figure 75:Get payments by contract number method	71

Figure 76:Run the app again	71
Figure 77:Spring Initializr.....	73
Figure 78:Initializr configured	74
Figure 79:Download the project and extract it.....	74
Figure 80:Project in the folder explorer.....	75
Figure 81:Run the command : mvn eclipse :eclipse to make the project an eclipse project	75
Figure 82:Build success	76
Figure 83:Import it under Eclipse	76
Figure 84:Project structure.....	77
Figure 85:The contents of pom.xml.....	77
Figure 86:java.version.....	78
Figure 87:@SpringBootApplication	79
Figure 88:application.properties	79
Figure 89:configure the database connection parameters and some parameters related to the Persistence Framework	80
Figure 90:"recharge" database under MySQL	81
Figure 91:create the package beans under the main package	81
Figure 92>Create the "Recharge" class.....	82
Figure 93:Class diagram of Recharge entity	82
Figure 94:Implementation of Recharge class.....	83
Figure 95:To start the application, use mvn spring-boot:run command.....	83
Figure 96:We can see now in the cmd Started RechargeApplication.....	84
Figure 97:Check if the "recharge" table is created.	84
Figure 98:Insert a new row to the table recharge.....	85
Figure 99:Create the "RechargeRepository" interface.....	86
Figure 100: "RechargeRepository"	86

Figure 101:In the interface add the method findById()	87
Figure 102:Create the "RechargeController" class	88
Figure 103:RechargeController class	88
Figure 104:@RestController and @RequestMapping("recharges")	89
Figure 105:Add CRUD web services to the RechargeController	90
Figure 106>List of phone top up(recharge)	91
Figure 107:Creation(Add some recharge)	91
Figure 108:Check if recharges were added successfully to the database	92
Figure 109:Update the recharge with id 4	92
Figure 110:The update is well achieved	93
Figure 111:Count test	93
Figure 112>Delete recharge with id 3	94
Figure 113:The recharge with id 3 was deleted successfully	94
Figure 114:Get recharge by telephone number	95
Figure 115:Get recharge by telephone number test	95
Figure 116:Get recharge by telephone number result's test	96
Figure 117:Spring Initializr	97
Figure 118:"Download the project and extract it	98
Figure 119:Run the command : mvn eclipse:eclipse to make the project an eclipse project	98
Figure 120:Import it under Eclipse	99
Figure 121:@SpringBootApplication	100
Figure 122:The content of pom.xml	100
Figure 123:java.version property	101
Figure 124:configure the database connection parameters and some parameters related to the Persistence Framework	102
Figure 125>Create the "register" database under MySQL	103

Figure 126:Create the "Role" class	103
Figure 127:Role class.....	104
Figure 128:Attributes of Role class	104
Figure 129:Generate constructors following our needs and wants	105
Figure 130:Select name field	105
Figure 131:Generate getters and setters	106
Figure 132:Add some annotations:@Entity,@Table,@Id,@GeneratedValue	106
Figure 133:Create the "User" class	107
Figure 134:User's class attributes	107
Figure 135:Generate constructors following our needs and wants	108
Figure 136:Generate getters and setters	109
Figure 137:User class.....	110
Figure 138:Add @Entity , @Table and (@UniqueConstraints for email) annotations.....	110
Figure 139:Adding @Id and @GeneratedValue annotations for Id field.....	111
Figure 140:@Column for column names.....	111
Figure 141:Adding @ManyToMany annotation for the relation between the User and the Role classes.....	112
Figure 142:Let's run the app: To start the application, use mvn spring-boot:run command.....	112
Figure 143:We can see now in the cmd Started RegistrationLoginApplication.....	113
Figure 144:Check if the "user","role" tables are created.	113
Figure 145:The structure of User table	114
Figure 146:The structure of Role table	114
Figure 147:The structure of Users_roles table(Association-class)	115
Figure 148:Create the "UserRepository" interface	116
Figure 149: "UserRepository" interface.....	116
Figure 150:the interface UserRepository extends the JpaRepository	117

Figure 151:Creation of UserService class under the sub package service of the main package	117
Figure 152:Creation of UserRegistrationDto class	118
Figure 153:UserRegistrationDto class attributes	119
Figure 154:UserRegistrationDto class after adding constructors	119
Figure 155:Adding the save method's signature to the UserService interface	120
Figure 156:Creation of UserServiceImpl class	121
Figure 157:UserServiceImpl class is created	121
Figure 158:Add @Service annotation to the class and add UserRepository reference	122
Figure 159:The implementation of the save method that returns a new user	122
Figure 160:Creation of the class UserRegistrationController under the sub package controller of the main package.....	123
Figure 161:UserRegistrationController class.....	124
Figure 162:Add annotations @Controller and @RequestMapping which is used to map web requests to Spring Controller methods.....	124
Figure 163>Create a new html file under the templates folder called registration	125
Figure 164:Add showRegistrationForm method with @GetMapping annotation to the controller that return the page registration	126
Figure 165:Adding the userRegistrationDto method with @ModelAttribute annotation	126
Figure 166:Searching for bootstrap 3.3.7 cdn link in Google.....	127
Figure 167:Copy bootstrap 3.3.7 cdn's link.....	127
Figure 168:Paste bootstrap 3.3.7 cdn's link in the head section of our html page	128
Figure 169:Copy paste nav section from the official documentation of bootstrap and personalize it following our needs and wants	128
Figure 170>Show success message using thymeleaf syntax	129
Figure 171:Different fields for the registration form with prebuilt HTML5 validation features	129
Figure 172:Preview of the registration form in eclipse.....	130
Figure 173:Run the app again	130

Figure 174:The registration form in the browser	131
Figure 175:Filling the form with valid data.....	131
Figure 176:After Register button click the success message is displayed to the user	132
Figure 177:Let's check the database if the user was added successfully.....	133
Figure 178:In table role the user has default role.....	133
Figure 179:In the association table	134
Figure 180:Form validation example.....	134
Figure 181:default login page	135
Figure 182:Default generated security password.....	135
Figure 183>Create SecurityConfiguration class under the sub package config of the main package	136
Figure 184:Notice : The presence of the security dependency in the pom.xml.....	136
Figure 185:Adding annotations @Configuration and @EnableWebSecurity	137
Figure 186:Add @Bean annotation to tell JPA to create an entity and @Autowired for the reference userService.....	138
Figure 187:In the configure method we add all permissions pages such as “/login” and “logout”.	138
Figure 188:UserDetailsService	139
Figure 189:Let's implement this method loadUserByUsername.....	140
Figure 190:loadUserByUsername Implementation	140
Figure 191:mapRolesToAuthorities method.....	141
Figure 192:Creation of the MainController class under the web.dto sub package	141
Figure 193:Add login method with @GetMapping annotation that return the login page that we will develop later.....	142
Figure 194:Add BcryptPasswordEncoder reference to the UserServiceImpl class.....	143
Figure 195:Using encode() method in order to encode/crypt the password	143
Figure 196>Create new Html file called login as we did for registration.....	144
Figure 197:Nav bar navigation code.....	144

Figure 198:User Login form we use the post method and as action the view /login.....	145
Figure 199:Continuation code of user login form(username and password inputs ,and the login button)	145
Figure 200:Add a link to the registration page	146
Figure 201:The preview page of login page in eclipse	146
Figure 202:Run the app again to test if the login feature is working well.....	147
Figure 203:Login Page in the browser.....	147
Figure 204:Register new user	148
Figure 205:Confirmation of success registration is displayed	148
Figure 206:Try to fill the login form with the information of the new user	149
Figure 207:After logging button click, we get this error message of type Not Found 404 because there is no homepage.....	149
Figure 208:For this, First add home method that returns index page in the MainController	150
Figure 209:As we did before for the registration and login pages. Let's create a new file called index.html	151
Figure 210:The navigation bar that's common for all views	151
Figure 211:Let's run the app and test the welcome page if is working well after logging.....	152
Figure 212:Notice: the presence of Thymeleaf dependency in the pom.,xml	152
Figure 213:use security authentication to retrieve the email from the database	153
Figure 214:Displaying the email of the user is working well	153
Figure 215:We want to add the logout link in the navbar so we add the selected code	154
Figure 216:Notice : In the configuration security file specifically in the configure method we observe the presence of /logout.....	155
Figure 217:We run the app and test if the Logout link was successfully added to the navbar.....	155
Figure 218:After clicking on the logout link, we are redirected to the login page and a confirmation message is displayed	156
Figure 219:Spring Initializr.....	157

Figure 220:make the project an eclipse project	158
Figure 221:Project Structure	158
Figure 222:@EnableConfigServer annotation.....	159
Figure 223:Configuration Server	159
Figure 224:initialize the git repository with git init command.	160
Figure 225:Create all microservices' .properties files in VS code IDE.....	160
Figure 226:Configuration of the database and the Persistence Framework.....	161
Figure 227:We set datasource url and the server.port:8081 for moneyTransfer service.	161
Figure 228:We set datasource url and the server.port:8082 for phoneRecharge service.....	161
Figure 229:We set datasource url and the server.port:8083 for wePayment service.....	162
Figure 230:Adding files to git repository cloud-config with the command git add ..	162
Figure 231:To start the application, use mvn spring-boot:run command.....	163
Figure 232:Test	164
Figure 233:Test service-money.....	164
Figure 234:Test service-phoneRecharge.....	165
Figure 235:Test service-wePayment.....	165
Figure 236:service-moneyTransfer\pom.xml.....	166
Figure 237:service-phoneRecharge\pom.xml	166
Figure 238:service-wePayment\pom.xml	167
Figure 239:Add bootstrap.properties file under the folder src/main/ressources for the three services	168
Figure 240:bootstrap.properties	168
Figure 241:bootstrap.properties service-recharge.....	169
Figure 242:bootstrap.properties service-transfer	169
Figure 243:Adding the spring-boot-starter-actuator dependency	169
Figure 244:Set the server port for moneyTransfer service equal to 8085.....	170

Figure 245:Test the service actuator refresh in ARC.....	170
Figure 246:The response of the request with the code message 200 OK	170
Figure 247:We notice that the service money transfer runs on the server port 8085	172
Figure 248:Spring Initializr.....	173
Figure 249:@EnableConfigServer annotation.....	173
Figure 250:Creation of service-registration.properties file under VS code IDE.	174
Figure 251:We add these lines of code in service-registration.properties	174
Figure 252:In bootstrap.properties we set these properties: application name, spring-cloud.config.uri and spring-cloud.config.profile.....	175
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.	175
Figure 254:The presence of spring-cloud-starter-netflix-eureka-server dependency at the pom.xml.	176
Figure 255:add the annotation @EnableDiscoveryClient to the main application file	176
Figure 256:Spring Eureka	176
Figure 257:Spring Initializr.....	177
Figure 258:@EnableZuulProxy annotation on my main Spring Boot application.....	178
Figure 259:Creation of service-proxy.properties file under VS code IDE	178
Figure 260:We set the port number to 8080	178
Figure 261:We add the service-proxy.properties file to the cloud-config directory.....	178
Figure 262:In bootstrap.properties we set these properties: application name, spring-cloud.config.uri and spring-cloud.config.profile.....	179
Figure 263:@EnableDiscoveryClient	180
Figure 264:Spring Eureka	180
Figure 265:Open the browser and test get all money transfers list.....	181
Figure 266:Test service-recharge. The list is empty	181
Figure 267:Test service-payment. The list is empty	182

Figure 268:login.html.....	183
Figure 269:login page in the browser	184
Figure 270:registration.html.....	185
Figure 271:registration page in the browser	186
Figure 272:index.html	187
Figure 273:index.html continuation code	188
Figure 274:index.html in the browser.....	189
Figure 275:money_transfer.html.....	190
Figure 276:money transfer service in the browser.....	191
Figure 277:send_money.html	192
Figure 278:send_money.html continuation code.....	193
Figure 279:Fill the form of send money	194
Figure 280:Generation of the code of the transfer	194
Figure 281:receive_money.html	195
Figure 282:receive_money.js	196
Figure 283:receive_money.js code continuation	197
Figure 284:Fill the form of receive money	198
Figure 285:receive money successfully	198
Figure 286:Case of data validation-Confirmation message.....	199
Figure 287:phone_recharge.html	200
Figure 288:phone_recharge.html	201
Figure 289:phone_recharge.js.....	202
Figure 290:phone_recharge.js code continuation	202
Figure 291:phone recharge in the browser.....	203
Figure 292:phone recharge done successfully	203
Figure 293:phone recharge data validation-error message	204

Figure 294:Water & electricity Payment html	205
Figure 295:Water & electricity Payment html continuation.....	205
Figure 296:we_payment.js	206
Figure 297:we_payment.js continuation.....	207
Figure 298:remplirTableau() method.....	207
Figure 299:W/E Payment in the browser.....	208
Figure 300:Table w/e payment	208
Figure 301:w/e payment done successfully	209
Figure 302:W/E payment in the browser	209
Figure 303>Create Counter class under the package beans:.....	210
Figure 304:Add @Bean annotation for counter method in the class ClientAppApplication.java...	210
Figure 305:Visitor counter in the browser	211

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:

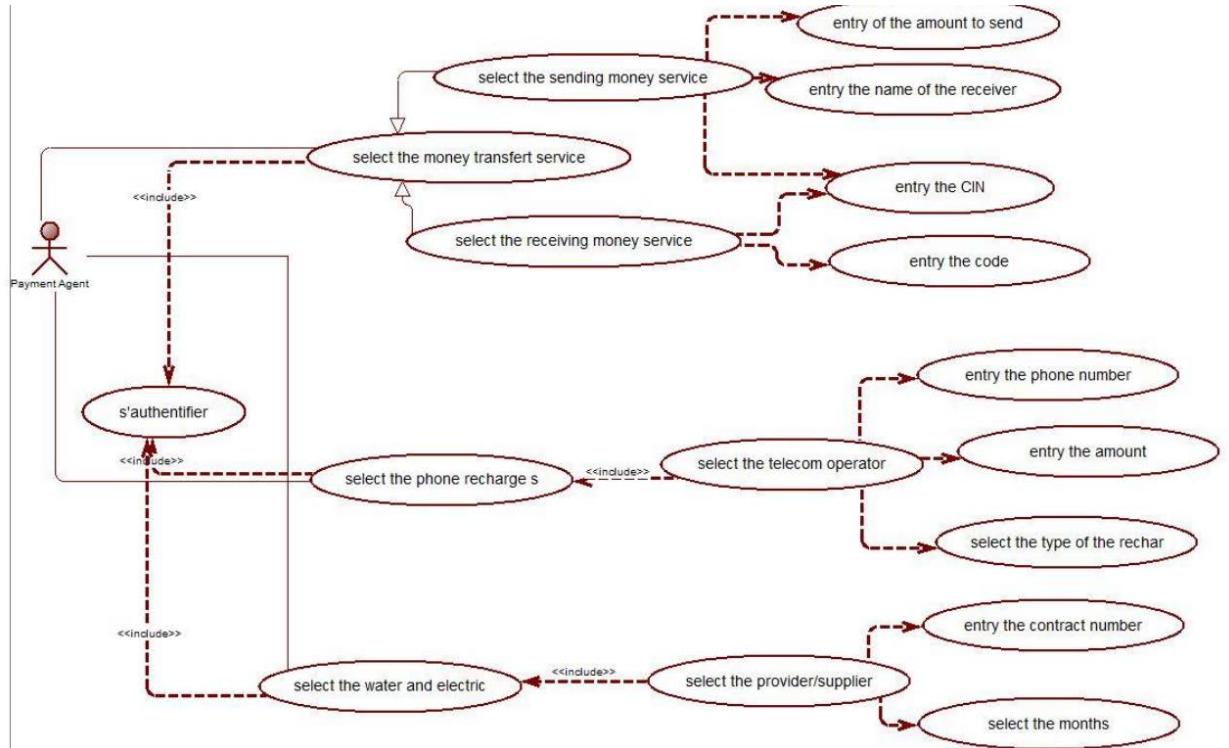


Figure 1:Use case diagram

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:



Project

Maven Project
 Gradle Project

Language

Java Kotlin Groovy

Dependencies

Spring Boot DevTools DEVELOPER TOOLS
Provides fast application restarts, LiveReload, and configurations for enhanced development experience.

Spring Data JPA SQL
Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.

Spring Web WEB
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

MySQL Driver SQL
MySQL JDBC and R2DBC driver.

Rest Repositories WEB
Exposing Spring Data repositories over REST via Spring Data REST.

Project Metadata

Group: ma.ensaj

Artifact: MoneyTransferMicroservice

Name: MoneyTransferMicroservice

Description: A microservice to manage money transfers

Package name: ma.ensaj.MoneyTransferMicroservice

Packaging: Jar War

Java: 15 11 8

Buttons:
GENERATE CTRL + ⌘ **EXPLORE** CTRL + SPACE **SHARE...**

Figure 2:Initial creation of the application with Spring Initializr

3. Download the project and extract it

Name: MoneyTransferMicroservice

Description: A microservice to manage money transfers

MySQL Driver SQL
MySQL JDBC and R2DBC driver.

Buttons:
GENERATE CTRL + ⌘ **EXPLORE** CTRL + SPACE **SHARE...**

MoneyTransferMic...zip ^

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 command...	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
[INFO] 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\MoneyTransferMicroservice.
[INFO]
[INFO]
[INFO] BUILD SUCCESS
[INFO]
[INFO] -----Total time: 9.877 s
[INFO] Finished at: 2021-01-08T16:31:20+01:00
[INFO] -----
```

Figure 5:make the project an eclipse project

5. Import it under Eclipse.

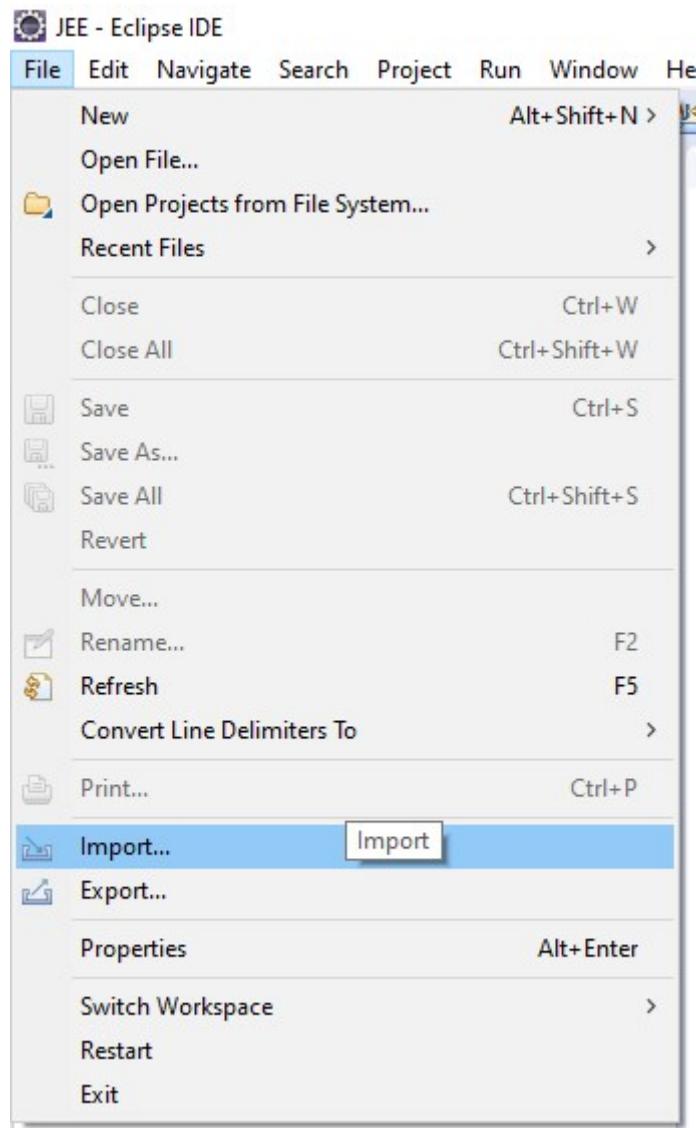


Figure 6: Import it under Eclipse.

Import it under Eclipse.

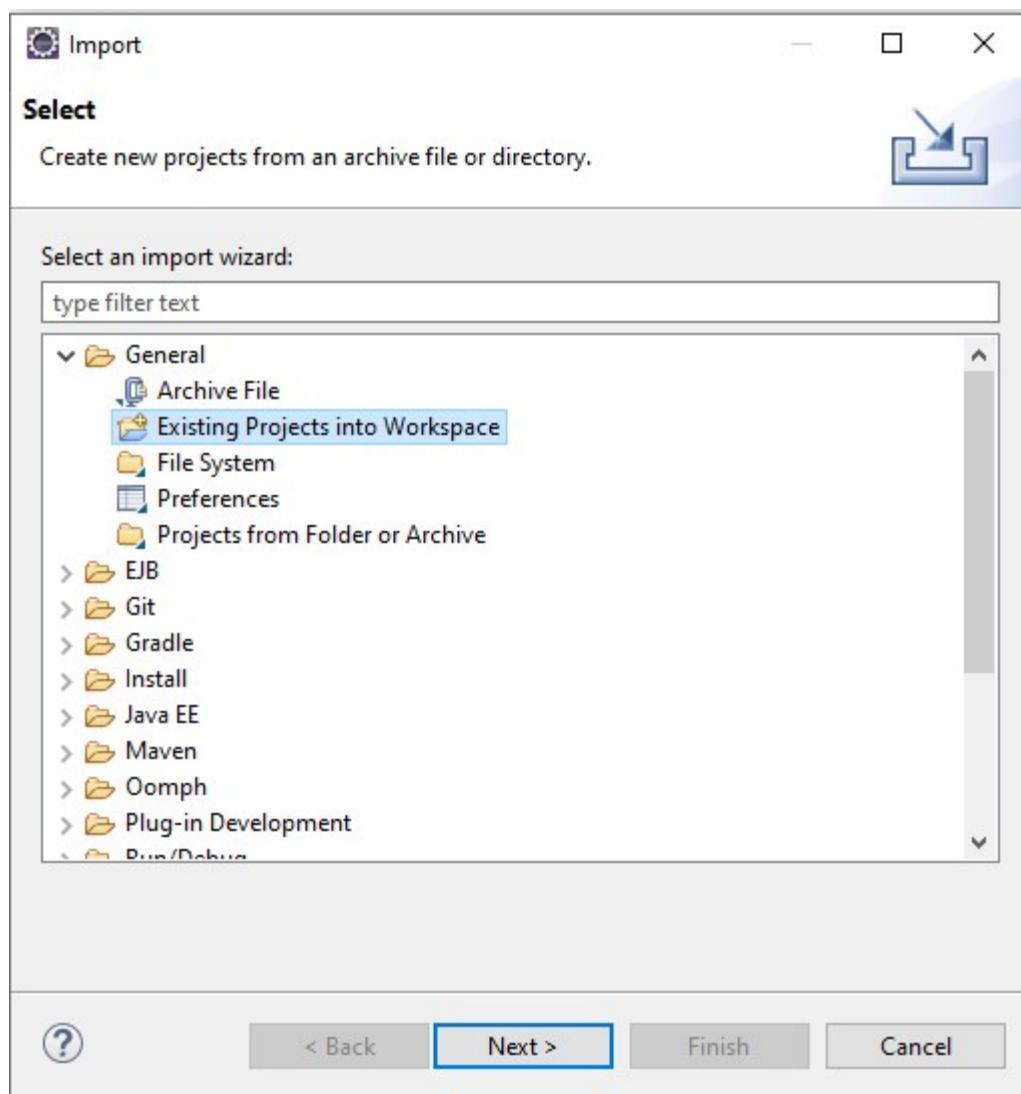


Figure 7:Import it under Eclipse.

Import it under Eclipse.

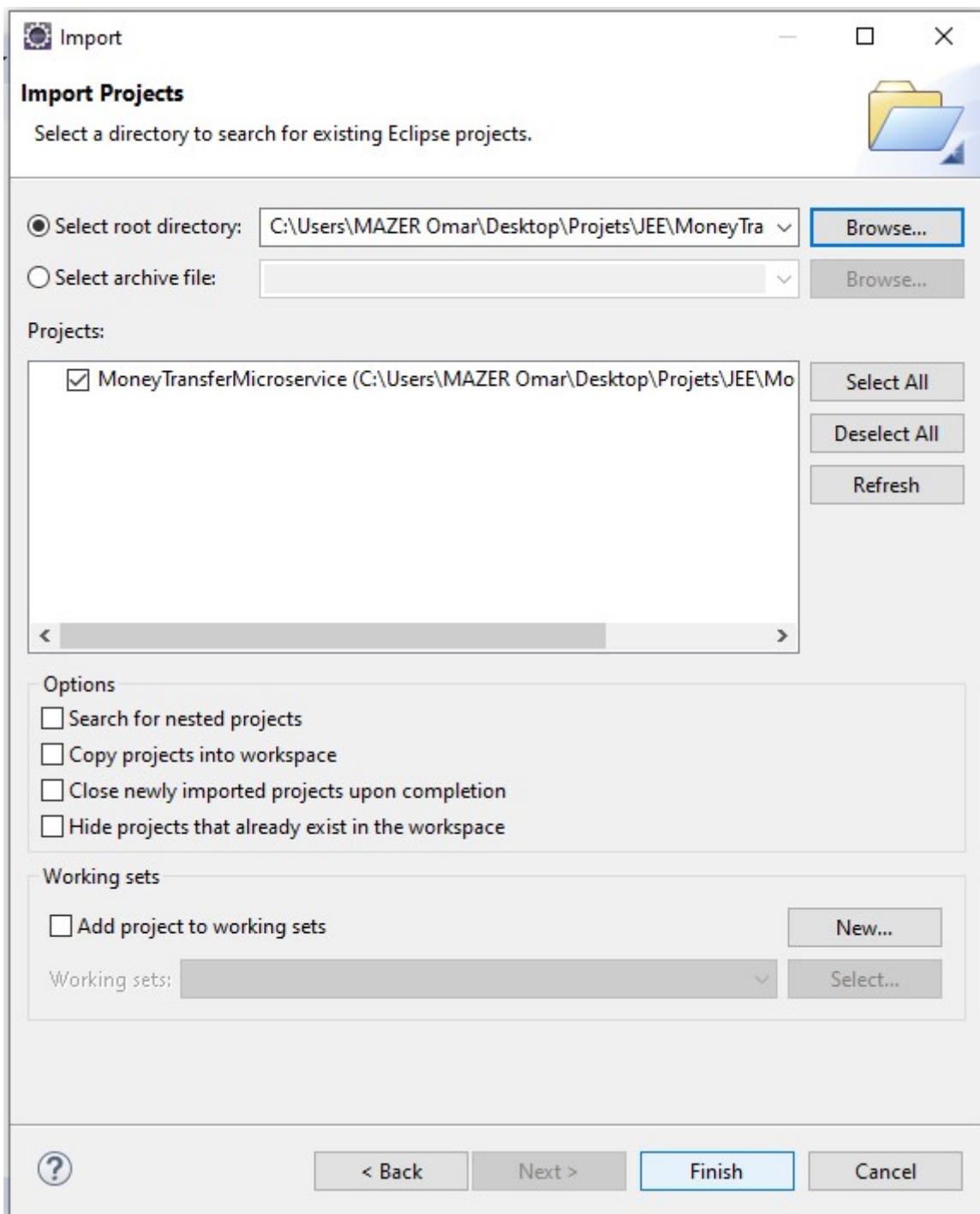


Figure 8: Import it under Eclipse.

6. Project structure

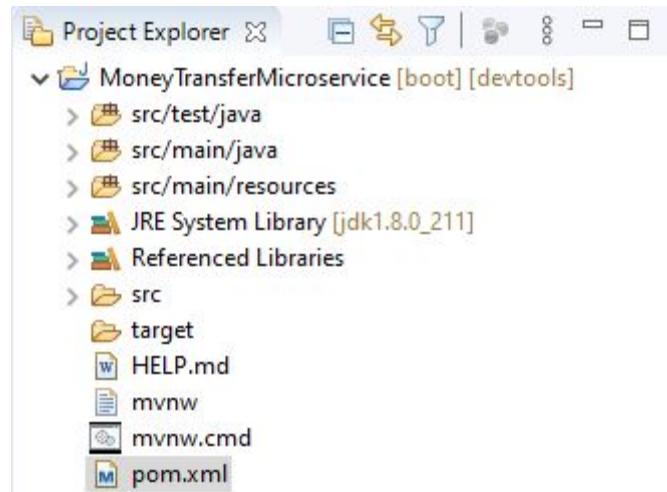


Figure 9:Project structure

7. The content of pom.xml :

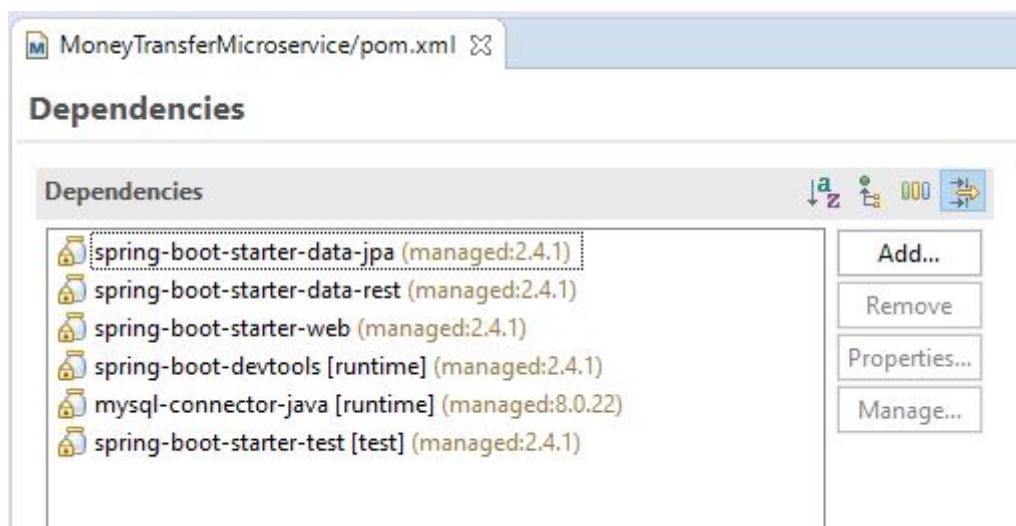


Figure 10:The content of pom.xml

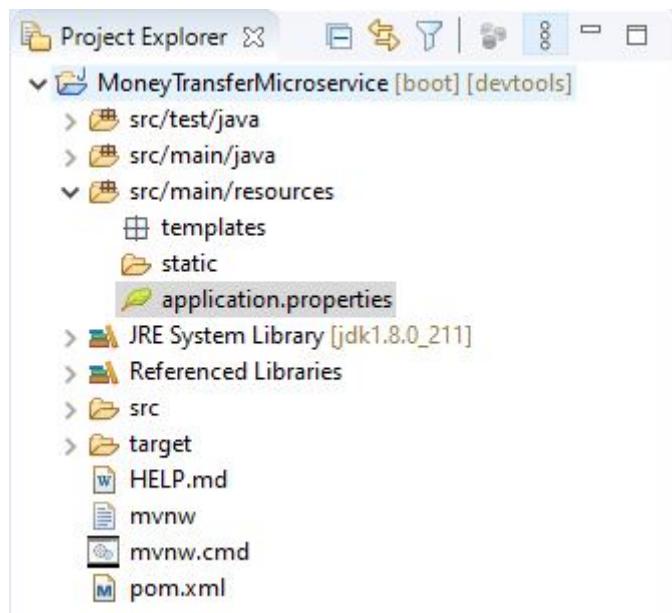
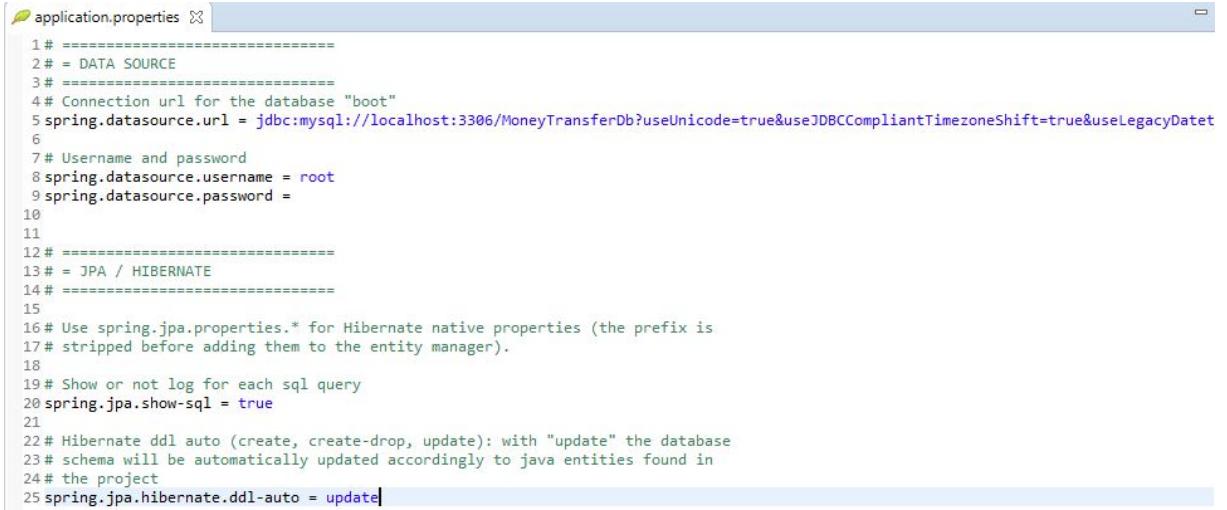


Figure 11:application.properties

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 :



```
application.properties
```

```
1 # =====
2 # = DATA SOURCE
3 # =====
4 # Connection url for the database "boot"
5 spring.datasource.url = jdbc:mysql://localhost:3306/MoneyTransferDb?useUnicode=true&useJDBCCompliantTimezoneShift=true&useLegacyDatetimeCode=false&serverTimezone=UTC
6
7 # Username and password
8 spring.datasource.username = root
9 spring.datasource.password =
10
11
12 # =====
13 # = JPA / HIBERNATE
14 # =====
15
16 # Use spring.jpa.properties.* for Hibernate native properties (the prefix is
17 # stripped before adding them to the entity manager).
18
19 # Show or not log for each sql query
20 spring.jpa.show-sql = true
21
22 # Hibernate ddl auto (create, create-drop, update): with "update" the database
23 # schema will be automatically updated accordingly to java entities found in
24 # the project
25 spring.jpa.hibernate.ddl-auto = update|
```

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

9. Create the "MoneyTransferDb" database under MySQL.

Bases de données



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

```

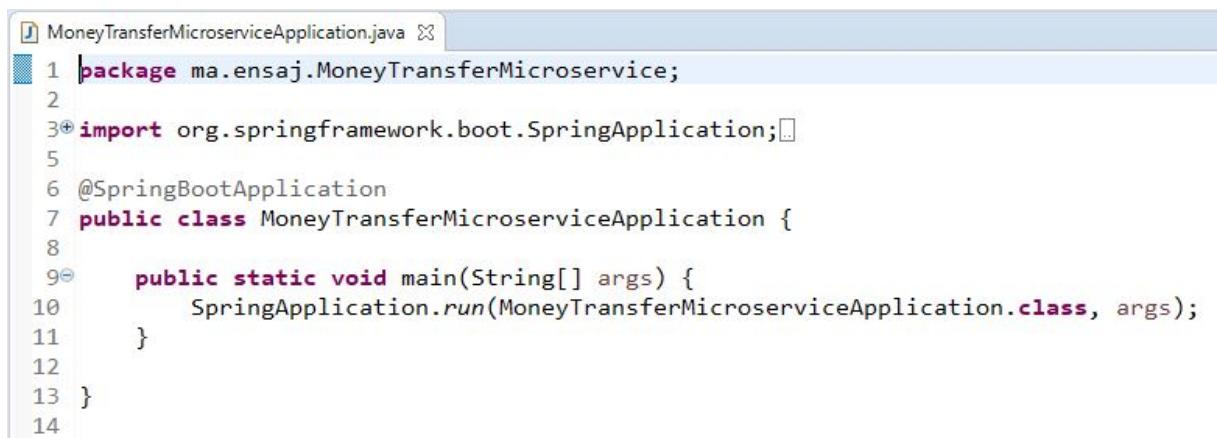
Transfer.java ✘
1 package ma.ensaj.MoneyTransferMicroservice.entity;
2 import java.util.Date;
3
4
5 @Entity
6 public class Transfer {
7     @Id
8     @GeneratedValue(strategy=GenerationType.IDENTITY)
9     private int id;
10    private String cinSender;
11    private int amount;
12    private String nameReceiver;
13    private String cinReceiver;
14    private long code;
15
16    @JsonFormat(shape = JsonFormat.Shape.STRING, pattern = "yyyy-MM-dd HH:mm:ss")
17    @Temporal(TemporalType.TIMESTAMP)
18    private Date dateSending;
19
20    @JsonFormat(shape = JsonFormat.Shape.STRING, pattern = "yyyy-MM-dd HH:mm:ss")
21    @Temporal(TemporalType.TIMESTAMP)
22    private Date dateReceiving;
23
24    @Column(columnDefinition="BOOLEAN DEFAULT false")
25    private Boolean isReceived;
26
27    public Transfer() {
28        super();
29        this.code = new Random().nextLong() % (9999999999L - 1000000000L) + 9999999999L;
30    }
31}

```

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).



```
1 package ma.ensaj.MoneyTransferMicroservice;
2
3 import org.springframework.boot.SpringApplication;
4
5 @SpringBootApplication
6 public class MoneyTransferMicroserviceApplication {
7
8     public static void main(String[] args) {
9         SpringApplication.run(MoneyTransferMicroserviceApplication.class, args);
10    }
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 :

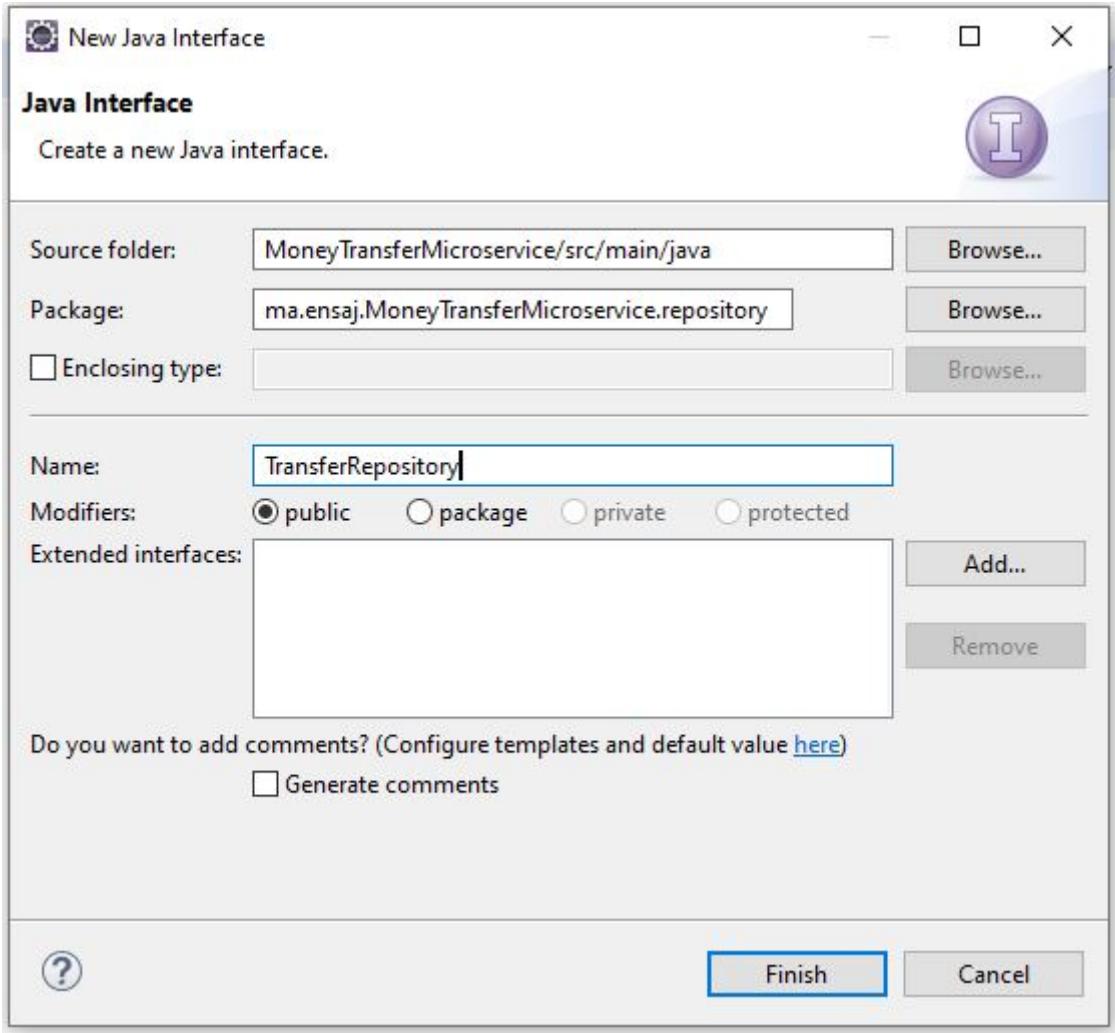


Figure 19:Create the "TransferRepository"

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

Transfer findById(int id);

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

```

1 package ma.ensaj.MoneyTransferMicroservice.repository;
2
3 import org.springframework.data.jpa.repository.JpaRepository;
4 import org.springframework.data.jpa.repository.Query;
5 import org.springframework.data.repository.query.Param;
6
7 import ma.ensaj.MoneyTransferMicroservice.entity.Transfer;
8
9 public interface TransferRepository extends JpaRepository<Transfer, Integer> {
10     Transfer findById(int id);
11
12     @Query("SELECT t FROM Transfer t WHERE t.code= :code AND t.isReceived= false")
13     Transfer findByCode(@Param("code") long code);
14 }

```

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**

```

1 package ma.ensaj.MoneyTransferMicroservice.controller;
2
3@import java.util.Calendar;
4
5 @RestController
6 @RequestMapping("transfers")
7 public class TransferController {
8
9     @Autowired
10    private TransferRepository transferRepository;
11
12    @PostMapping("/sendMoney")
13    public long sendMoney(@RequestBody Transfer transfer){
14        transfer.setDateSending(Calendar.getInstance().getTime());
15        transfer.setIsReceived(false);
16        transferRepository.save(transfer);
17        return transfer.getCode();
18    }
19
20    @PostMapping("/receiveMoney/{code}/{cin}")
21    public Transfer receiveMoney(@PathVariable (required = true) long code,@PathVariable (required = true) String cin){
22        Transfer transfer = transferRepository.findById(code);
23        if(transfer != null) {
24            transfer.setCinReceiver(cin);
25            transfer.setDateReceiving(Calendar.getInstance().getTime());
26            transfer.setIsReceived(true);
27            transferRepository.save(transfer);
28        }
29        return transfer;
30    }
31
32
33}
34
35}
36
37}
38
39}
40
41}
42
43}

```

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

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

```

44
45@DeleteMapping("/{id}")
46    public void delete(@PathVariable (required = true) String id){
47        Transfer s = transferRepository.findById(Integer.parseInt(id));
48        transferRepository.delete(s);
49    }
50
51@GetMapping("/{id}")
52    public Transfer findById(@PathVariable (required = true) String id){
53        return transferRepository.findById(Integer.parseInt(id));
54    }
55
56@GetMapping("/findByCode/{code}")
57    public Transfer findByCode(@PathVariable (required = true) String code){
58        return transferRepository.findByCode(Long.parseLong(code));
59    }
60
61@GetMapping("/all")
62    public List<Transfer>findAll(){
63        return transferRepository.findAll();
64    }
65 }

```

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

22. Testing controllers with ARC(Advanced Rest Client)

sendMoney service test

The screenshot shows the ARC Advanced Rest Client interface. At the top, it displays the method as 'POST' and the request URL as 'http://localhost:8080/transfers/sendMoney'. Below the URL, there is a 'SEND' button and a more options menu. Under the URL, the 'Request parameters' section is expanded, showing the following JSON body content type:

```
application/json
```

Below the body content type, there are three buttons: 'FORMAT JSON' (selected), 'MINIFY JSON', and 'COPY'. A code editor window shows the following JSON payload:1 {
2 "amount": 2500,
3 "nameReceiver": "Omar Mazer",
4 "cinSender": "DK894256"
5 }

Figure 23:sendMoney service test

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

The screenshot shows the ARC Advanced Rest Client interface after sending the request. It displays a green '200 OK' status bar with the text '138.23 ms'. To the right of the status bar is a 'DETAILS' button with a dropdown arrow. Below the status bar, there are four buttons: 'COPY', 'SAVE', 'SOURCE VIEW', and 'DATA TABLE'. A large text area below these buttons contains the response ID: '15845477639'.

Figure 24:The money is sent successfully

24. List of transfer's requests

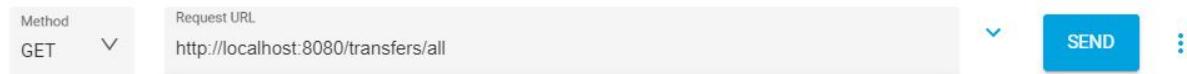


Figure 25:List of transfer's requests

25. Get list of money's transfers

A screenshot of a REST client interface showing a successful response. The status bar at the top indicates '200 OK' and '17.61 ms'. To the right is a 'DETAILS' button with a dropdown arrow. Below the status bar are four buttons: 'COPY', 'SAVE', 'SOURCE VIEW', and 'DATA TABLE'. The main area displays a JSON array with one element:

```
[Array[1]
-0: {
  "id": 1,
  "cinSender": "DK894256",
  "amount": 2500,
  "nameReceiver": "Omar Mazer",
  "cinReceiver": null,
  "code": 15845477639,
  "dateSending": "2021-01-08 17:43:47",
  "dateReceiving": null,
  "isReceived": false
}]
```

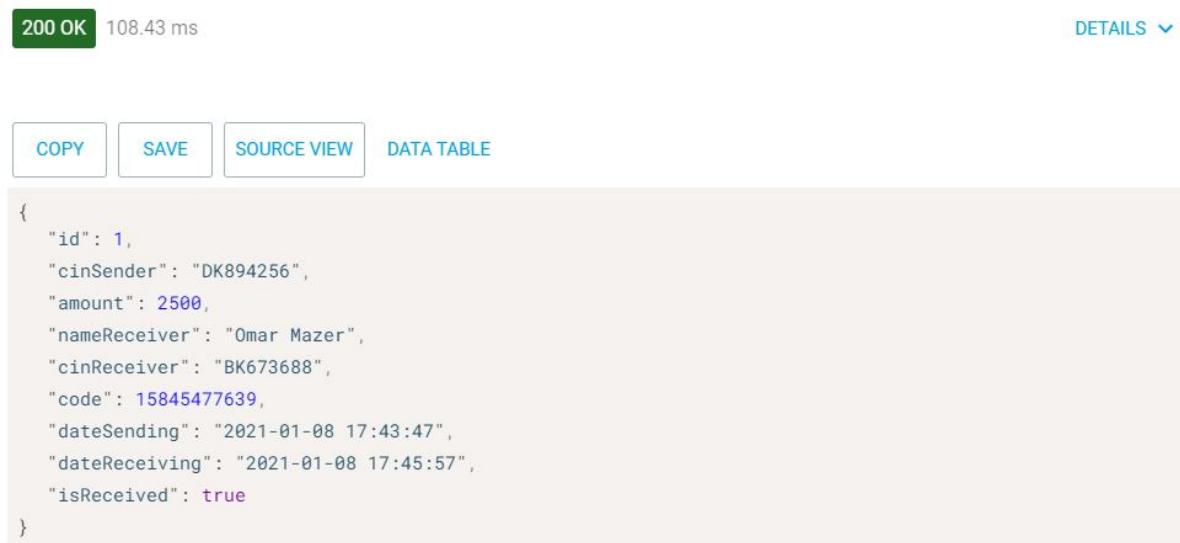
Figure 26:list of money's transfers

26. Receive money by code request



Figure 27:Receive money by code request

27. The money is received successfully



The screenshot shows a successful API response with a green "200 OK" button and a timestamp of 108.43 ms. Below the status bar are four buttons: COPY, SAVE, SOURCE VIEW, and DATA TABLE. The main area displays a JSON object representing a transaction:

```
{  
  "id": 1,  
  "cinSender": "DK894256",  
  "amount": 2500,  
  "nameReceiver": "Omar Mazer",  
  "cinReceiver": "BK673688",  
  "code": 15845477639,  
  "dateSending": "2021-01-08 17:43:47",  
  "dateReceiving": "2021-01-08 17:45:57",  
  "isReceived": true  
}
```

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:

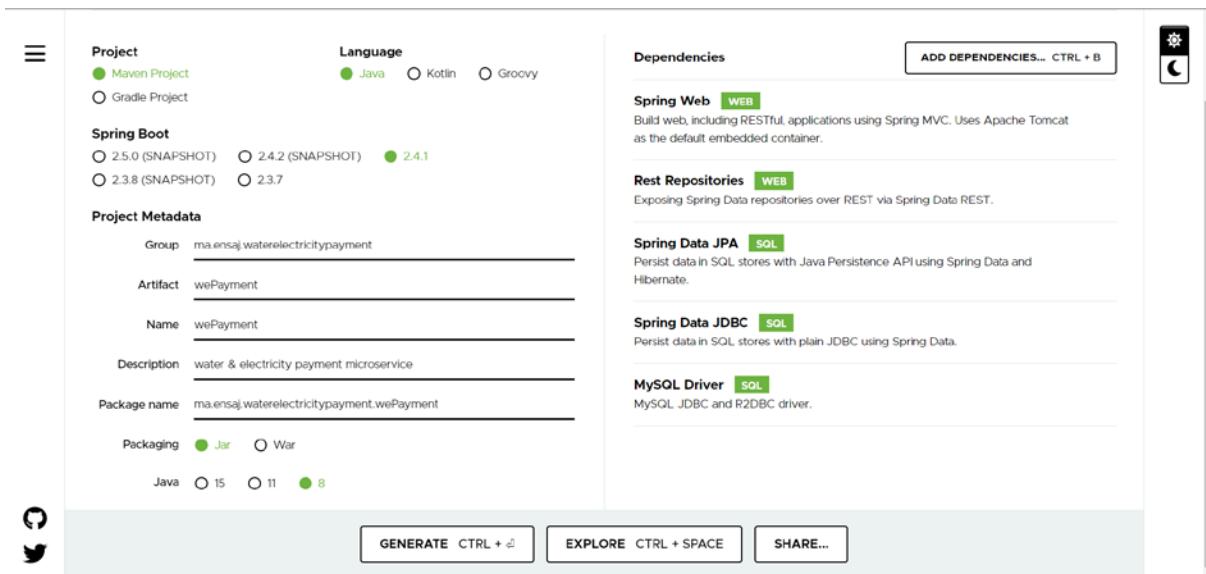


Figure 29:Initial creation of the application with Spring Initializr

2. Download the project and extract it



Figure 30:Download the project and extract it

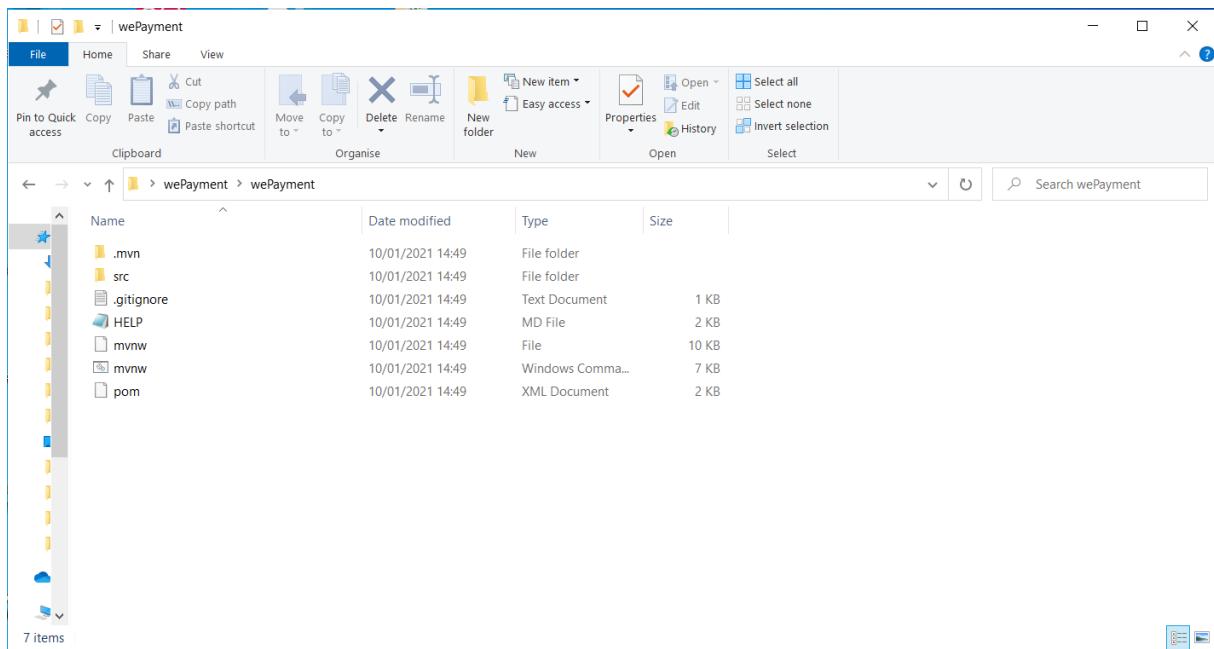


Figure 31:The project under the file explorer

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] -----
[INFO] Total time: 3.473 s
[INFO] Finished at: 2021-01-10T15:54:11+01:00
[INFO] -----

```

Figure 32:make the project an eclipse project

4. Import the project under Eclipse.

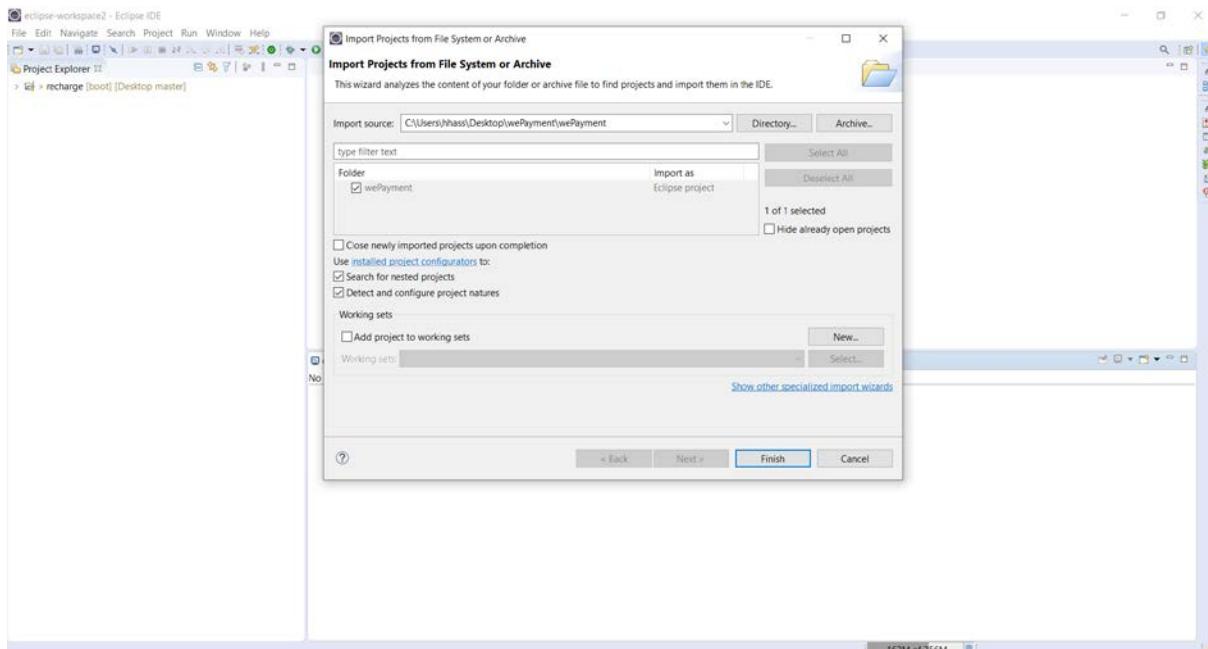


Figure 33:Import the project under Eclipse.

5. Project structure

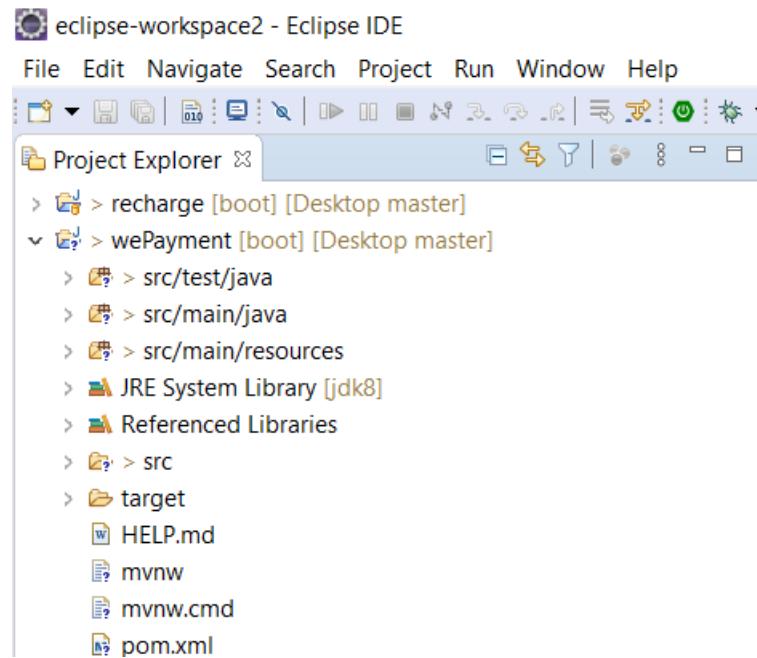


Figure 34:Project structure

6. The content of pom.xml :

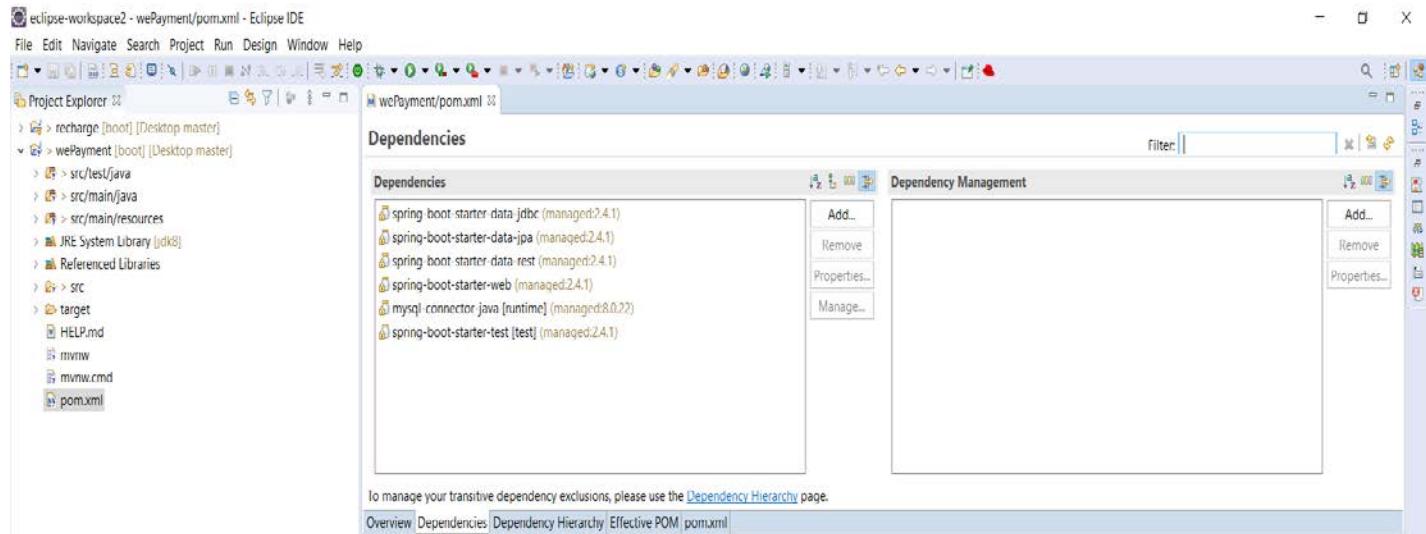


Figure 35: pom.xml dependencies

- 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:

The screenshot shows the Eclipse IDE interface with the title bar "eclipse-workspace2 - wePayment/pom.xml - Eclipse IDE". The menu bar includes File, Edit, Source, Navigate, Search, Project, Run, Window, Help. Below the menu is a toolbar with various icons. The main area displays the content of the pom.xml file:

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
4     <modelVersion>4.0.0</modelVersion>
5     <parent>
6       <groupId>org.springframework.boot</groupId>
7       <artifactId>spring-boot-starter-parent</artifactId>
8       <version>2.4.1</version>
9       <relativePath/> <!-- lookup parent from repository -->
10    </parent>
11    <groupId>ma.ensa1.waterelectricitypayment</groupId>
12    <artifactId>wePayment</artifactId>
13    <version>0.0.1-SNAPSHOT</version>
14    <name>wePayment</name>
15    <description>water & electricity payment microservice</description>
16
17    <properties>
18      <java.version>1.8</java.version>
19    </properties>
20
21    <dependencies>
22      <dependency>
23        <groupId>org.springframework.boot</groupId>
24        <artifactId>spring-boot-starter-data-jdbc</artifactId>
25      </dependency>
26      <dependency>
27        <groupId>org.springframework.boot</groupId>
28        <artifactId>spring-boot-starter-data-jpa</artifactId>
29      </dependency>
30      <dependency>
31        <groupId>org.springframework.boot</groupId>
```

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).

The screenshot shows the Eclipse IDE interface. The title bar reads "eclipse-workspace2 - wePayment/src/main/java/ma/ensaj/waterelectricitypayment/wePayment/WePaymentApplication.java - Eclipse IDE". The Project Explorer view on the left shows the project structure with files like recharge, wePayment, pom.xml, and application.properties. The code editor on the right displays the following Java code:

```

1 package ma.ensaj.waterelectricitypayment.wePayment;
2
3 import org.springframework.boot.SpringApplication;
4
5 @SpringBootApplication
6 public class WePaymentApplication {
7
8     public static void main(String[] args) {
9         SpringApplication.run(WePaymentApplication.class, args);
10    }
11 }
12
13 }
14

```

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.

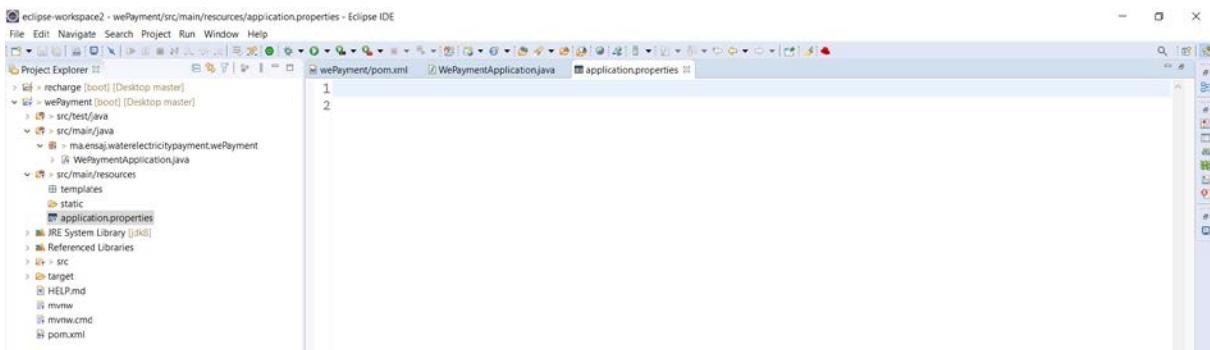
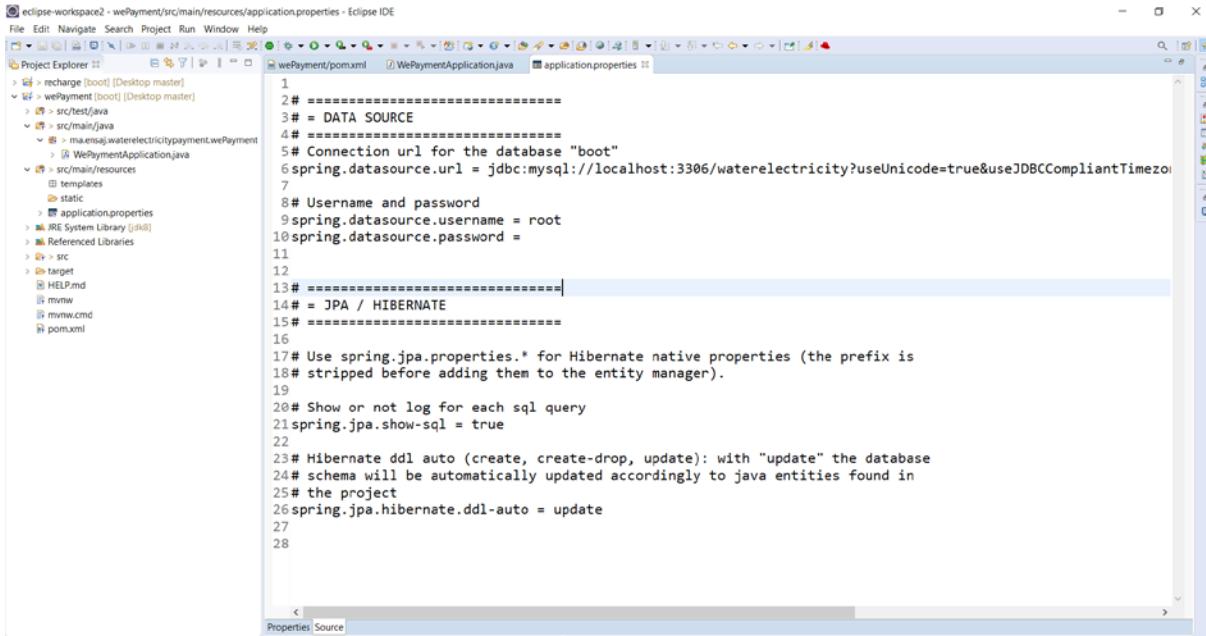


Figure 38:application.properties.

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



The screenshot shows the Eclipse IDE interface with the application.properties file open in the editor. The code in the file is as follows:

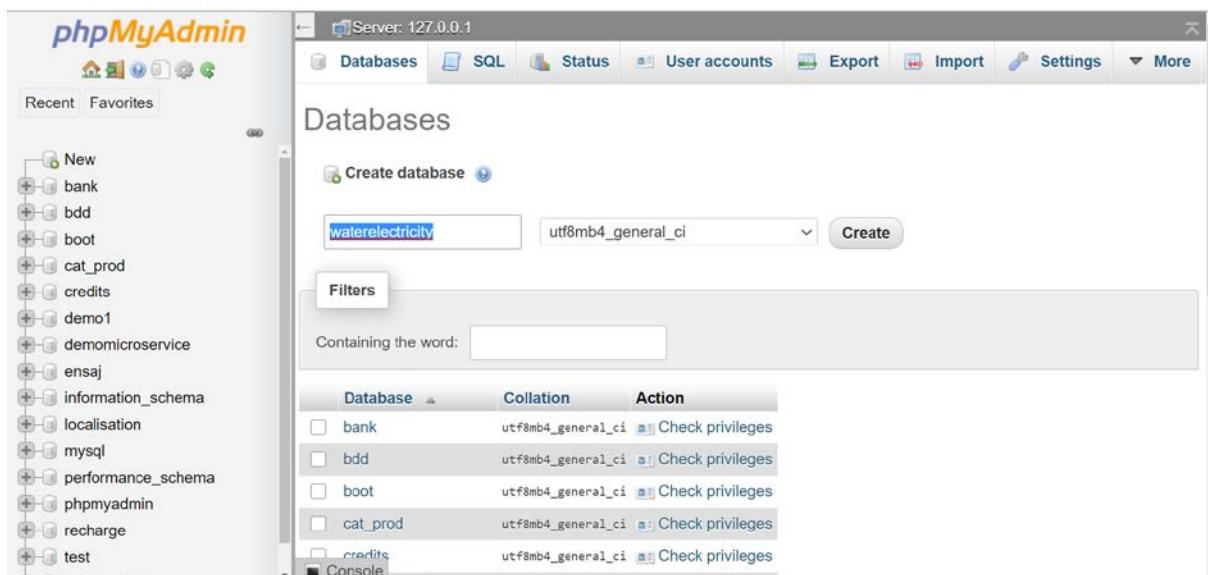
```

1  # =====
2 # = DATA SOURCE
3 # =====
4 # Connection url for the database "boot"
5 spring.datasource.url = jdbc:mysql://localhost:3306/waterelectricity?useUnicode=true&useJDBCCompliantTimezo
6
7 8# Username and password
9spring.datasource.username = root
10spring.datasource.password =
11
12
13# =====
14# = JPA / HIBERNATE
15# =====
16
17# Use spring.jpa.properties.* for Hibernate native properties (the prefix is
18# stripped before adding them to the entity manager).
19
20# Show or not log for each sql query
21spring.jpa.show-sql = true
22
23# Hibernate ddl auto (create, create-drop, update): with "update" the database
24# schema will be automatically updated accordingly to java entities found in
25# the project
26spring.jpa.hibernate.ddl-auto = update
27
28

```

Figure 39:application.properties

11. Create the "waterelectricity" database under MySQL.



The screenshot shows the phpMyAdmin interface with the "Databases" page selected. A new database named "waterelectricity" is being created. The "Create database" button is highlighted, and the database name is entered. The table lists existing databases and their collation and action details.

Database	Collation	Action
bank	utf8mb4_general_ci	Check privileges
bdd	utf8mb4_general_ci	Check privileges
boot	utf8mb4_general_ci	Check privileges
cat_prod	utf8mb4_general_ci	Check privileges
credits	utf8mb4_general_ci	Check privileges
demo1	utf8mb4_general_ci	Check privileges
demomicroservice	utf8mb4_general_ci	Check privileges
ensaj	utf8mb4_general_ci	Check privileges
information_schema	utf8mb4_general_ci	Check privileges
localisation	utf8mb4_general_ci	Check privileges
mysql	utf8mb4_general_ci	Check privileges
performance_schema	utf8mb4_general_ci	Check privileges
phpmyadmin	utf8mb4_general_ci	Check privileges
recharge	utf8mb4_general_ci	Check privileges
test	utf8mb4_general_ci	Check privileges
Console	utf8mb4_general_ci	Check privileges

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`):

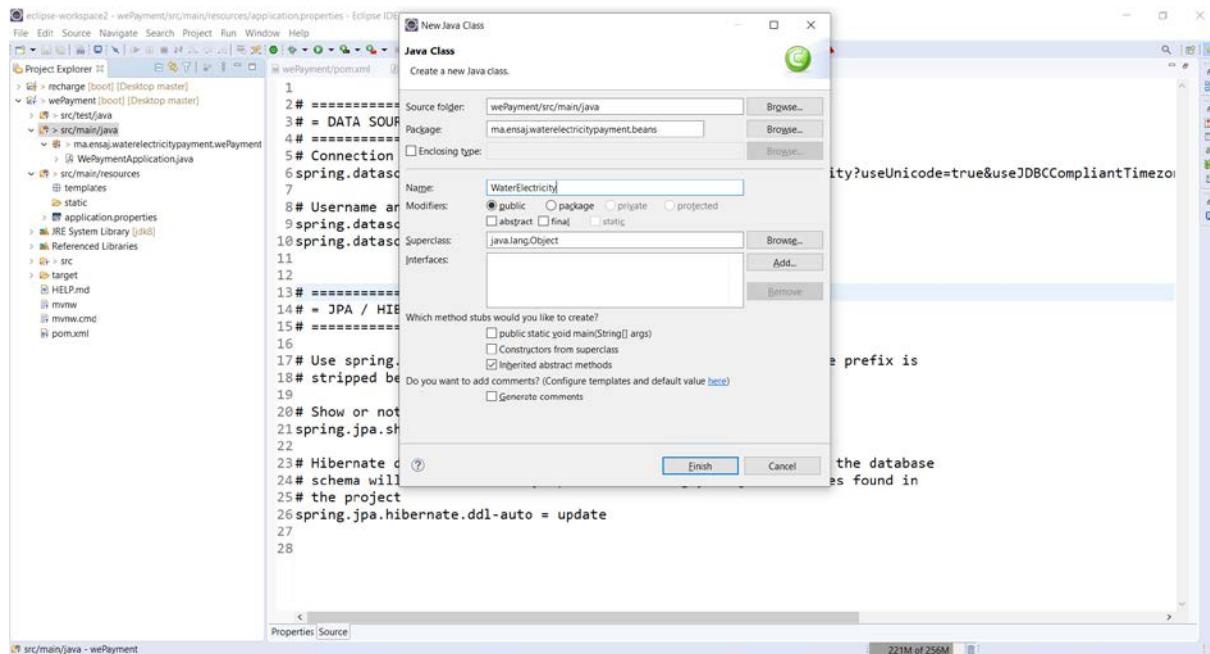


Figure 41:Create the "WaterElectricity" class

The screenshot shows the Eclipse IDE interface with the 'WaterElectricity.java' file open in the editor. The code defines a class named 'WaterElectricity' with a single method. The project structure on the left shows various Java files and resources.

```
1 package ma.ensaj.waterelectricitypayment.beans;
2
3 public class WaterElectricity {
4
5 }
```

Figure 42: "WaterElectricity" class

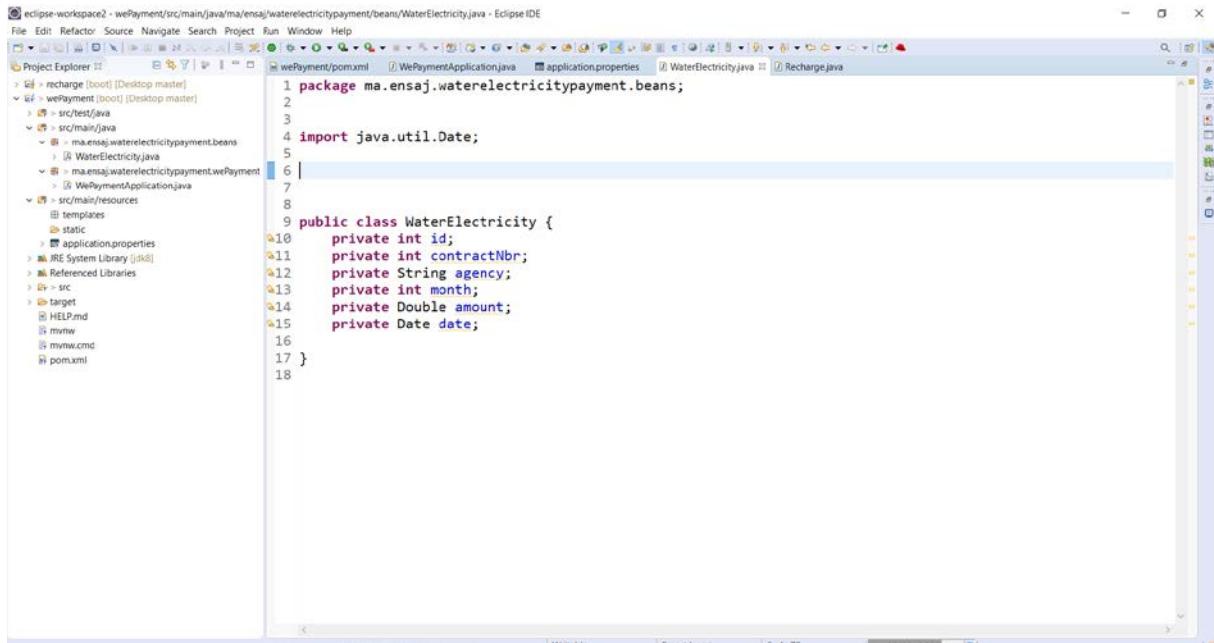
13. Class diagram of **WaterElectricity** entity

water_electricity	
id	int(11) <pk>
agency	varchar(255)
amount	double
contract_nbr	int(11)
date	datetime
month	int(11)

Figure 43: WaterElectricity entity

14. Implementation of WaterElectricity class

15. Attributes

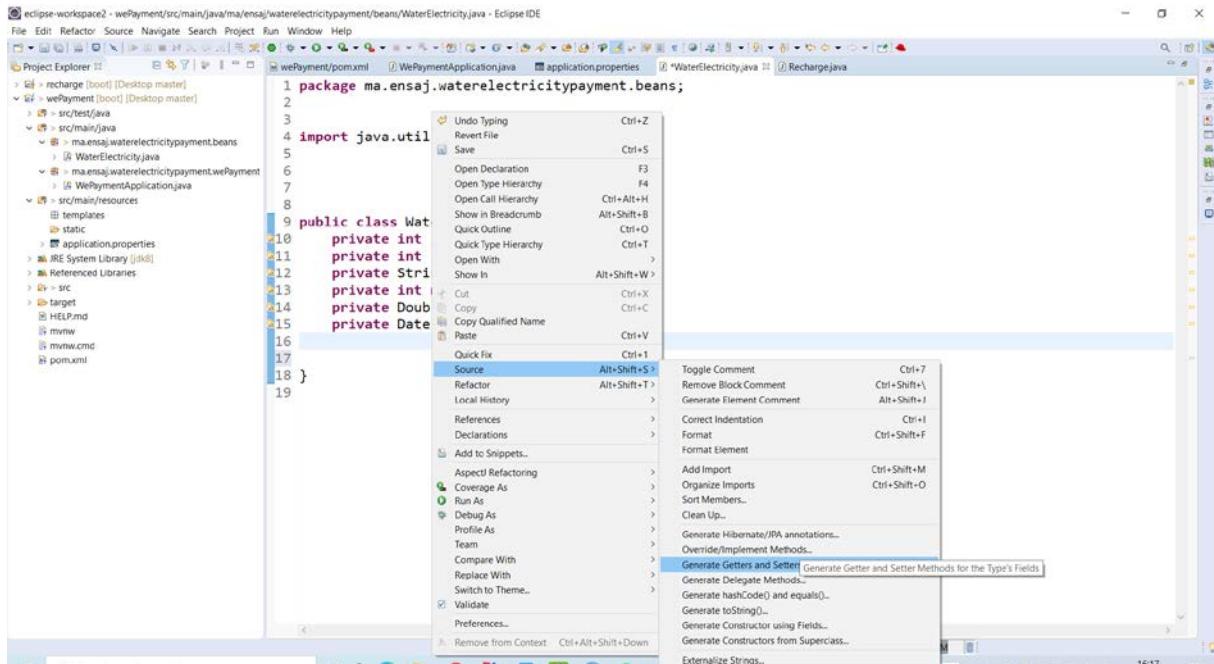


The screenshot shows the Eclipse IDE interface with the WaterElectricity.java file open in the editor. The code defines a class WaterElectricity with private fields: id, contractNbr, agency, month, amount, and date. The code is color-coded for syntax.

```
1 package ma.ensaj.waterelectricitypayment.beans;
2
3
4 import java.util.Date;
5
6
7
8
9 public class WaterElectricity {
10     private int id;
11     private int contractNbr;
12     private String agency;
13     private int month;
14     private Double amount;
15     private Date date;
16
17 }
```

Figure 44:Attributes

16. Generate getters and setters



The screenshot shows the Eclipse IDE interface with the WaterElectricity.java file open. A context menu is displayed over the class definition, with the "Source" option selected. The submenu shows various refactoring options, and at the bottom, the "Generate Getters and Setters" option is highlighted.

```
1 package ma.ensaj.waterelectricitypayment.beans;
2
3
4 import java.util.
5
6
7
8
9 public class Wat {
10     private int
11     private int
12     private Stri
13     private int
14     private Doub
15     private Date
16
17 }
18 }
```

Figure 45:Generate getters and setters

17. Select all attributes

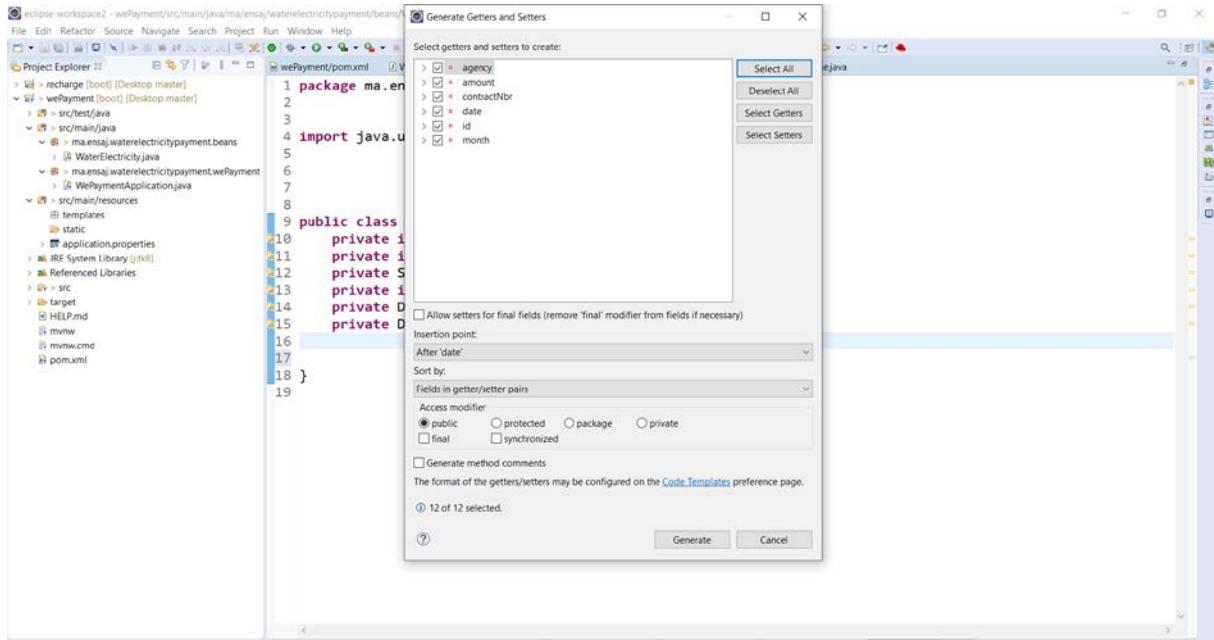


Figure 46:Select all attributes

18. Getters and setters are generated

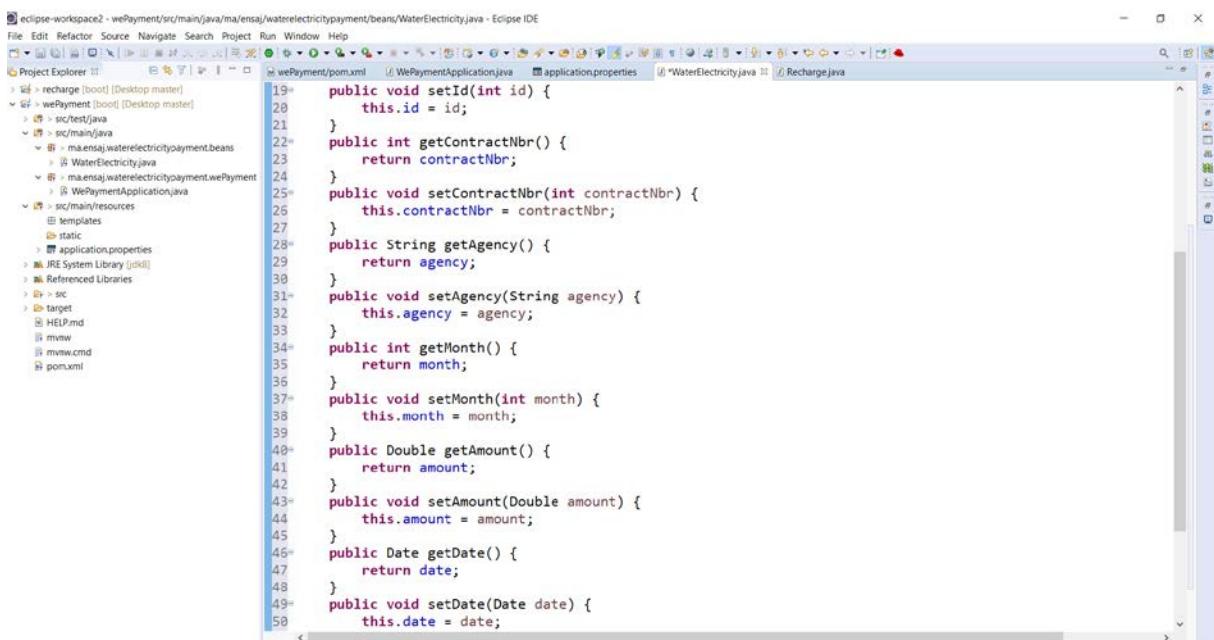


Figure 47:Getters and setters are generated

19. Generate the constructor

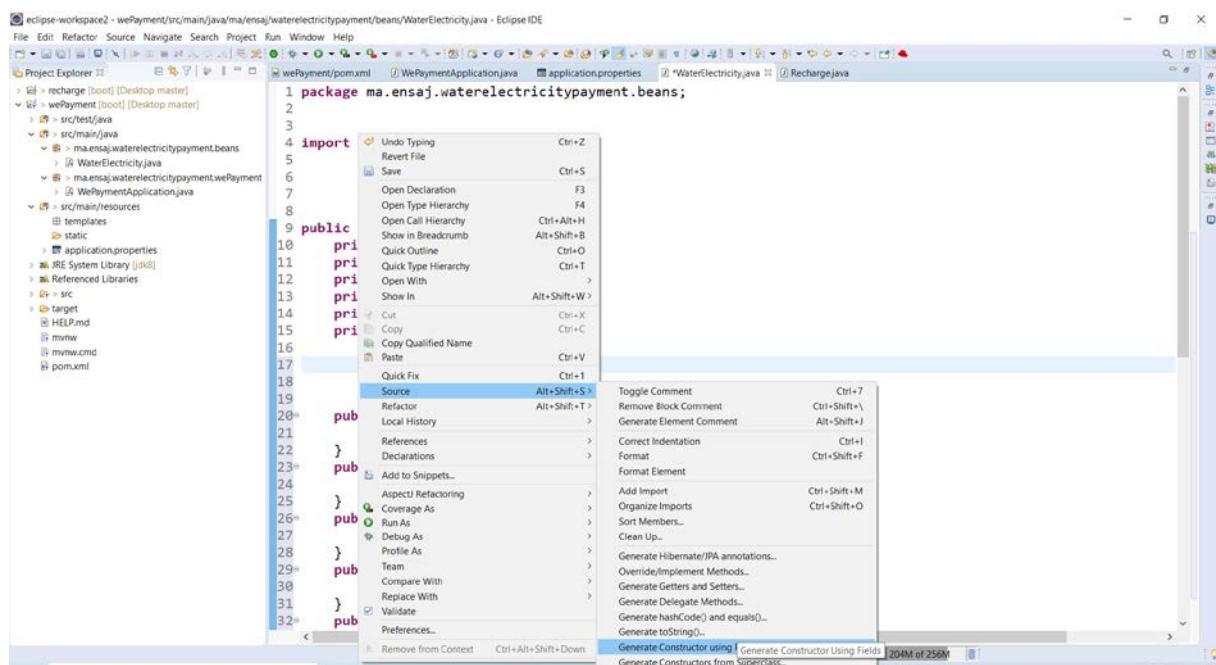


Figure 48:Generate the constructor

20. Constructors

```

1 package ma.ensaj.waterelectricitypayment.beans;
2
3
4 import java.util.Date;
5
6
7
8
9 public class WaterElectricity {
10     private int id;
11     private int contractNbr;
12     private String agency;
13     private int month;
14     private Double amount;
15     private Date date;
16
17     public WaterElectricity() {
18         super();
19     }
20     public WaterElectricity(int id, int contractNbr, String agency, int month, Double amount, Date date) {
21         super();
22         this.id = id;
23         this.contractNbr = contractNbr;
24         this.agency = agency;
25         this.month = month;
26         this.amount = amount;
27         this.date = date;
28     }
29
30
31     public int getId() {
32         return id;
33     }

```

Figure 49:Constructors

21. Generate the `toString()` method

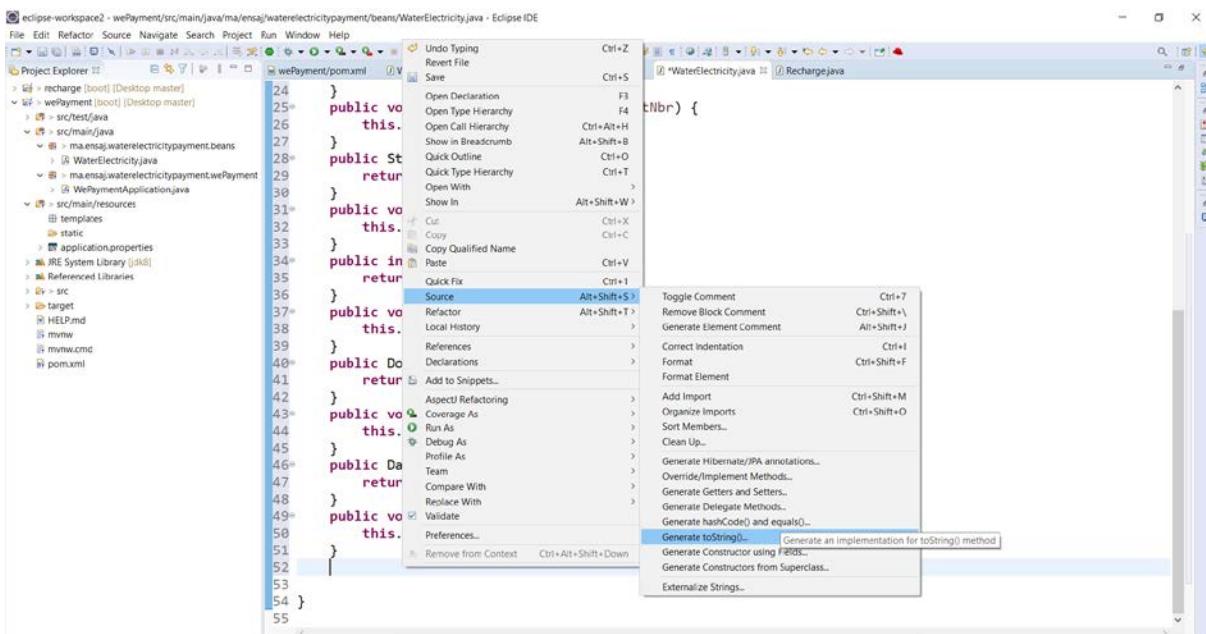


Figure 50:Generate the `toString()` method

22. `toString()`:

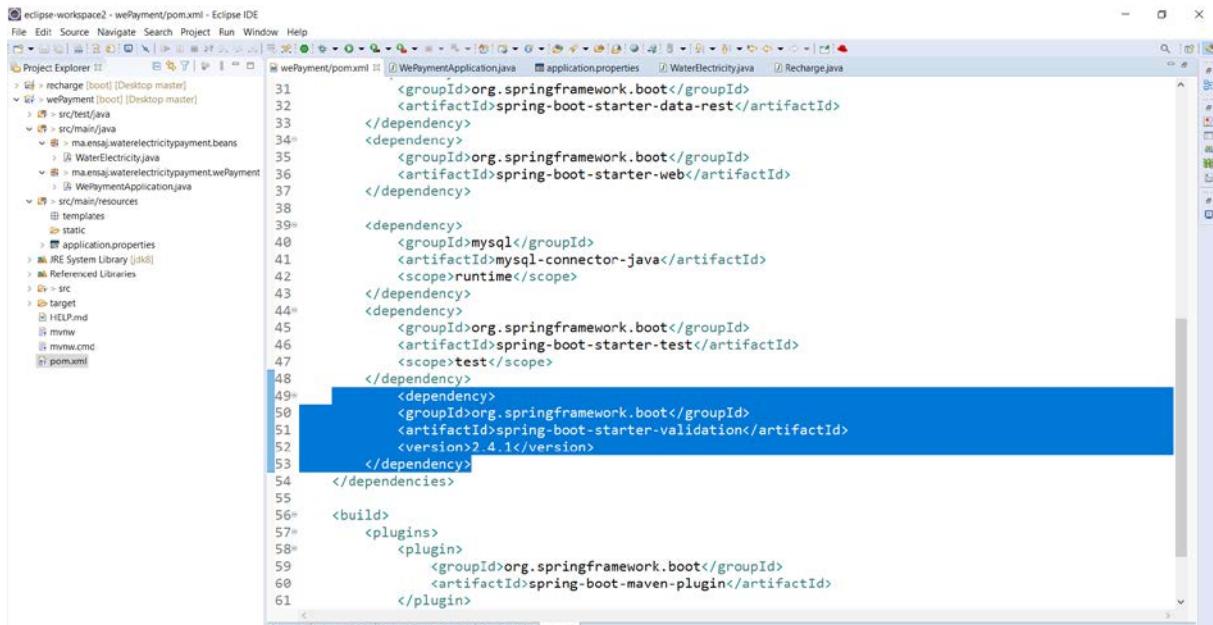
```

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

```

Figure 51:toString()

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



```

<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-rest</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <scope>runtime</scope>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-validation</artifactId>
    <version>2.4.1</version>
</dependency>
</dependencies>
<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
    </plugins>

```

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

24. We add some persistence annotations.

```

eclipse-workspace - wePayment/src/main/java/ma/ensaj/waterelectricitypayment/beans/WaterElectricity.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
wePayment/pom.xml WePaymentApplication.java application.properties WaterElectricity.java Recharge.java
4 import java.util.Date;
5
6 import javax.persistence.Entity;
7 import javax.persistence.GeneratedValue;
8 import javax.persistence.GenerationType;
9 import javax.persistence.Id;
10 import javax.persistence.Temporal;
11 import javax.persistence.TemporalType;
12 import javax.validation.constraints.Max;
13 import javax.validation.constraints.Min;
14
15 |
16
17 @Entity
18 public class WaterElectricity {
19     @Id
20     @GeneratedValue(strategy=GenerationType.IDENTITY)
21     private int id;
22     private int contractNbr;
23     private String agency;
24     @Min(value = 1) @Max(value=12)
25     private int month;
26     private Double amount;
27     @Temporal(TemporalType.DATE)
28     private Date date;
29
30     public WaterElectricity() {
31         super();
32     }
33     public WaterElectricity(int id, int contractNbr, String agency, int month, Double amount, Date date) {
34         super();
35         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>mvn spring-boot:run
[INFO] Scanning for projects...

```

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

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

```

Select C:\Windows\System32\cmd.exe - mvn spring-boot:run
2021-01-10 16:25:49.545 INFO 5492 --- [      main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.41]
2021-01-10 16:25:49.731 INFO 5492 --- [      main] o.a.c.c.C.[Tomcat].[localhost].[/]      : Initializing Spring embedded WebApplicationContext
2021-01-10 16:25:49.731 INFO 5492 --- [      main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialized [name: default]
2021-01-10 16:25:49.988 INFO 5492 --- [      main] o.hibernate.jpa.internal.util.LogHelper : HHH000204: Processing PersistenceUnitInfo [name: default]
2021-01-10 16:25:50.040 INFO 5492 --- [      main] org.hibernate.Version                   : HHH000412: Hibernate ORM core version 5.4.25.Final
2021-01-10 16:25:50.197 INFO 5492 --- [      main] o.hibernate.annotations.common.Version : HCANN000001: Hibernate Commons Annotations [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.hibernate.dialect.MySQL55Dialect
2021-01-10 16:25:51.131 INFO 5492 --- [      main] o.h.e.t.j.p.i.JtaPlatformInitiator  : HHH000490: Using JtaPlatform implementation: [org.hibernate.engine.transaction.jta.platform.internal.NoJtaPlatform]
2021-01-10 16:25:51.157 INFO 5492 --- [      main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory 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 'applicationTaskExecutor'
2021-01-10 16:25:52.689 INFO 5492 --- [      main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context path ''
2021-01-10 16:25:52.707 INFO 5492 --- [      main] m.e.w.wePayment.WePaymentApplication : Started WePaymentApplication in 7.811 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.

The screenshot shows the phpMyAdmin interface for the 'waterelectricity' database. The left sidebar lists databases and tables, with 'water_electricity' selected. The main panel displays the results of a query: 'MySQL returned an empty result set (i.e. zero rows). (Query took 0.0015 seconds.)'. Below this, a SQL query is shown: 'SELECT * FROM `water_electricity`'. The results table header includes columns: id, agency, amount, contract_nbr, date, month. A 'Query results operations' section contains a 'Create view' button. A 'Bookmark this SQL query' section allows labeling the query and includes a 'Label:' input field and a 'Bookmark this SQL query' button.

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

The screenshot shows the phpMyAdmin interface for the 'waterelectricity' database. The left sidebar lists databases and tables, with 'water_electricity' selected. The main panel displays a message: 'Table water_electricity has been altered successfully.' Below this, a SQL command is shown: 'ALTER TABLE `water_electricity` CHANGE `date` `date` DATETIME NULL DEFAULT NULL;'. The 'Table structure' tab is active, showing the table's schema. The columns are listed with their names, types, collations, attributes, and actions. The columns are:

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	id	int(11)			No	None		AUTO_INCREMENT	Change Drop More
2	agency	varchar(255)	utf8mb4_general_ci		Yes	NULL			Change Drop More
3	amount	double			Yes	NULL			Change Drop More
4	contract_nbr	int(11)			No	None			Change Drop More
5	date	datetime			Yes	NULL			Change Drop More
6	month	int(11)			No	None			Change Drop More

Below the table structure, there are buttons for 'Check all', 'With selected:', 'Browse', 'Change', 'Drop', 'Primary', 'Unique', 'Index', 'Fulltext', 'Add to central columns', and 'Remove from central columns'. At the bottom, there are buttons for 'Print', 'Propose table structure', 'Track table', 'Move columns', 'Normalize', 'Console', 'column(s)', 'after month', and 'Go'.

Figure 57:"waterelectricity" table structure

28. Insert a new row to the table waterelectricity

The screenshot shows the 'Insert' page for the 'water_electricity' table in phpMyAdmin. The table structure is displayed with columns: id, agency, amount, contract_nbr, date, and month. The 'id' field is set to 'RADEEJ'. The 'agency' field contains 'RADEEJ'. The 'amount' field is set to '400'. The 'contract_nbr' field contains '1234567'. The 'date' field is set to '2021-01-04 14:52:29'. The 'month' field is set to '2'. A checkbox for 'Ignore' is checked. The 'Go' button is visible at the bottom right.

Figure 58:Insert a new row to the table waterelectricity

The screenshot shows the results of a SQL query: 'SELECT * FROM water_electricity'. The results table displays one row with the following data: id (RADEEJ), agency (RADEEJ), amount (400), contract_nbr (1234567), date (2021-01-04 14:52:29), and month (2). The 'Edit', 'Copy', 'Delete', and other table operations are shown below the results table.

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 :

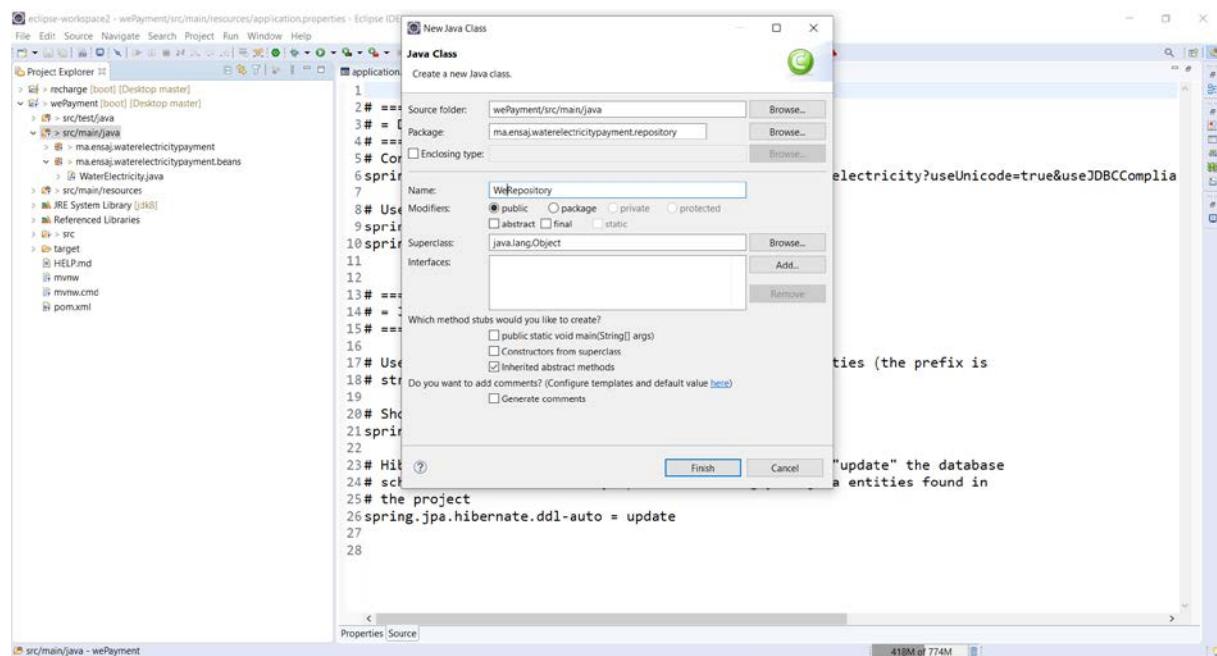
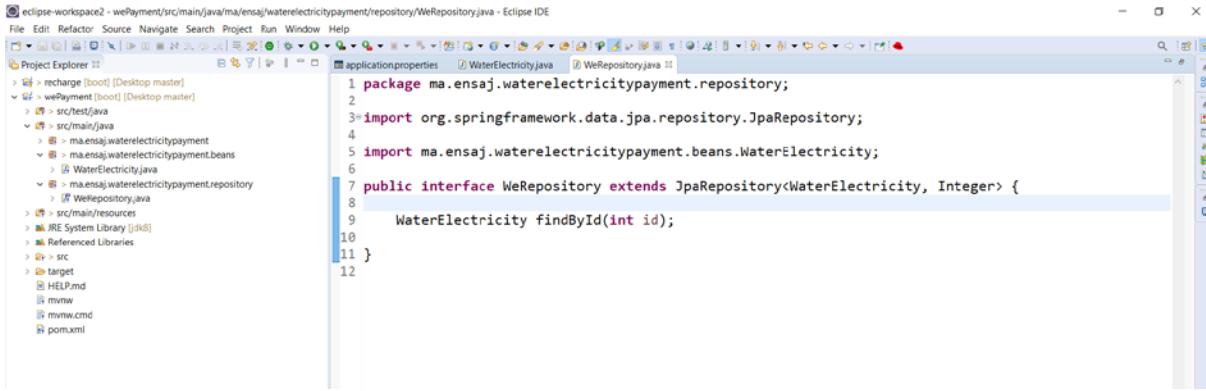


Figure 60:Create the "WeRepository" interface

30. In the interface add the method :

WaterElectricity findById(int id);



The screenshot shows the Eclipse IDE interface with the following details:

- Project Explorer:** Shows the project structure with packages like recharge, wePayment, and src/main/java.
- Code Editor:** Displays the `WeRepository.java` file content.
- Content Area:** Shows the Java code for the `WeRepository` interface:

```
1 package ma.ensaj.waterelectricitypayment.repository;
2
3 import org.springframework.data.jpa.repository.JpaRepository;
4
5 import ma.ensaj.waterelectricitypayment.beans.WaterElectricity;
6
7 public interface WeRepository extends JpaRepository<WaterElectricity, Integer> {
8
9     WaterElectricity findById(int id);
10
11 }
```

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:

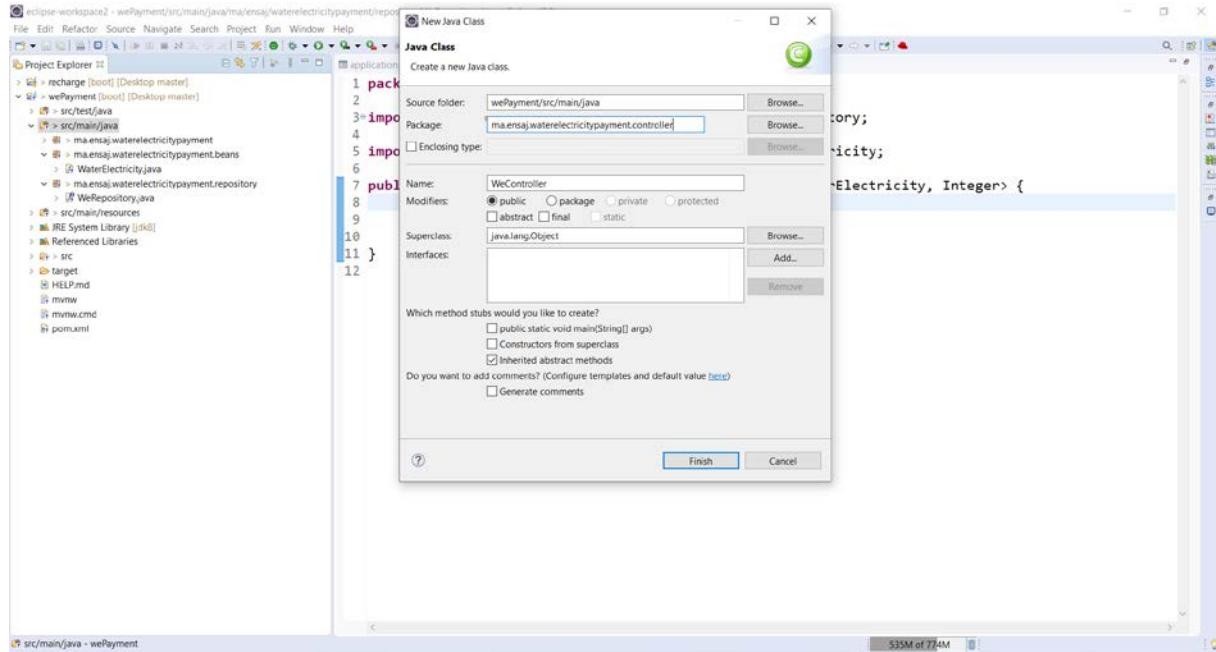


Figure 62:Create the "WeController" class

33. WeController

```

eclipse-workspace2 - wePayment/src/main/java/ma/ensaj/waterelectricitypayment/controller/WeController.java - Eclipse IDE
File Edit Refactor Source Navigate Search Project Run Window Help
Project Explorer applicationproperties WaterElectricity.java WeRepository.java WeController.java
1 package ma.ensaj.waterelectricitypayment.controller;
2
3 public class WeController {
4
5 }
6

```

Figure 63:WeController

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

N.B:

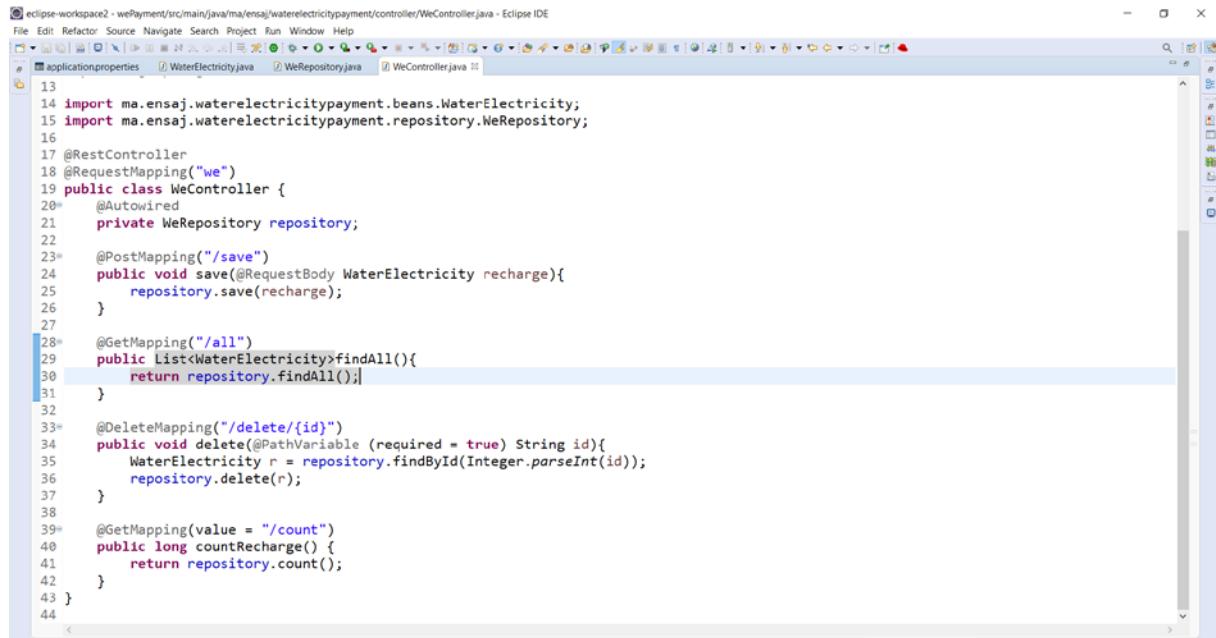
- The **@RestController** annotation was introduced in Spring 4.0 to simplify the creation of RESTful web services. It's a convenience annotation that combines **@Controller** and **@ResponseBody** – which eliminates the need to annotate every request handling method of the controller class with the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

```

eclipse-workspace2 - wePayment/src/main/java/ma/ensaj/waterelectricitypayment/controller/WeController.java - Eclipse IDE
File Edit Refactor Source Navigate Search Project Run Window Help
Project Explorer applicationproperties WaterElectricity.java WeRepository.java WeController.java
1 package ma.ensaj.waterelectricitypayment.controller;
2
3 import org.springframework.web.bind.annotation.RequestMapping;
4 import org.springframework.web.bind.annotation.RestController;
5
6 @RestController
7 @RequestMapping("we")
8 public class WeController {
9
10 }
11

```

35. Add CRUD web services to the WeController



The screenshot shows the Eclipse IDE interface with the title bar "eclipse-workspace2 - wePayment/src/main/java/ma/ensaj/waterelectricitypayment/controller/WeController.java - Eclipse IDE". The code editor displays the WeController.java file, which contains the following Java code:

```
13 import ma.ensaj.waterelectricitypayment.beans.WaterElectricity;
14 import ma.ensaj.waterelectricitypayment.repository.WeRepository;
15
16
17 @RestController
18 @RequestMapping("we")
19 public class WeController {
20     @Autowired
21     private WeRepository repository;
22
23     @PostMapping("/save")
24     public void save(@RequestBody WaterElectricity recharge){
25         repository.save(recharge);
26     }
27
28     @GetMapping("/all")
29     public List<WaterElectricity>findAll(){
30         return repository.findAll();
31     }
32
33     @DeleteMapping("/delete/{id}")
34     public void delete(@PathVariable (required = true) String id){
35         WaterElectricity r = repository.findById(Integer.parseInt(id));
36         repository.delete(r);
37     }
38
39     @GetMapping(value = "/count")
40     public long countRecharge() {
41         return repository.count();
42     }
43 }
44
```

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>mvn spring-boot:run
```

Figure 65: Launch the project again

38. List of water and electricity payments

The screenshot shows the Advanced REST client interface. The left sidebar displays a history of requests, including several GET requests to various endpoints like /we/all, /ws/all, and /recharges/. The main panel shows a request configuration for a GET method to http://localhost:8080/we/all. The response status is 200 Not Found, with a duration of 451.05 ms. The response body is an array containing one element: { "id": 1, "contractNbr": "1234567", "agency": "RADEEJ" }.

Figure 66:List of water and electricity payments

The screenshot shows the Advanced REST client interface. The left sidebar displays a history of requests, including several GET requests to various endpoints like /we/all, /ws/all, and /recharges/. The main panel shows a request configuration for a GET method to http://localhost:8080/we/all. The response status is 200 OK, with a duration of 451.05 ms. The response body is an array containing one element: { "id": 1, "contractNbr": "1234567", "agency": "RADEEJ", "month": 2, "amount": 400, "date": "2021-01-04" }.

Figure 67:200 OK

39. Creation(Add some water and electricity payments)

The screenshot shows the Advanced REST client interface. On the left, there is a history panel titled 'Today' listing various API calls. On the right, a main panel shows a POST request to 'http://localhost:8080/we/save'. The request parameters section includes 'Method: POST', 'Request URL: http://localhost:8080/we/save', 'Body content type: application/json', and a JSON payload:

```
1 {  
2 "contractNbr": 1234567,  
3 "agency": "RADEEJ",  
4 "month": 3,  
5 "amount": 489.90,  
6 "date": "2021-02-06"  
7 }
```

The response status is '200 Not Found' with a duration of '192.62 ms'.

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

The screenshot shows the phpMyAdmin interface for a MySQL database named 'waterelectricity'. The left sidebar lists various databases and tables, including 'water_electricity'. The main panel displays the results of a SELECT query: 'SELECT * FROM `water_electricity`'. The results show four rows of data:

	id	agency	amount	contract_nbr	date	month
<input type="checkbox"/>	1	RADEEJ	400	1234567	2021-01-04 00:00:00	2
<input type="checkbox"/>	2	RADEEJ	489.9	1234567	2021-02-06 00:00:00	3
<input type="checkbox"/>	3	RADEEJ	459.7	1234567	2021-03-06 00:00:00	4
<input type="checkbox"/>	4	ONE	489.9	1234567	2021-02-06 00:00:00	3

Below the table, there are buttons for 'Edit', 'Copy', 'Delete', 'Check all', and 'With selected'. The bottom navigation bar includes 'Console', 'Copy to clipboard', 'Export', 'Display chart', and 'Create view'.

Figure 71:The update is well passed

43. Delete payment with id 3

The Advanced REST client interface shows a history of API requests. A specific DELETE request to `http://localhost:8080/we/delete/3` is selected. The response status is `200 OK` with a duration of `584.13 ms`. The interface includes tabs for Headers, Body, Authorization, Actions, Config, and Code, with `SOURCE VIEW` currently active.

Figure 72:Delete payment with id 3

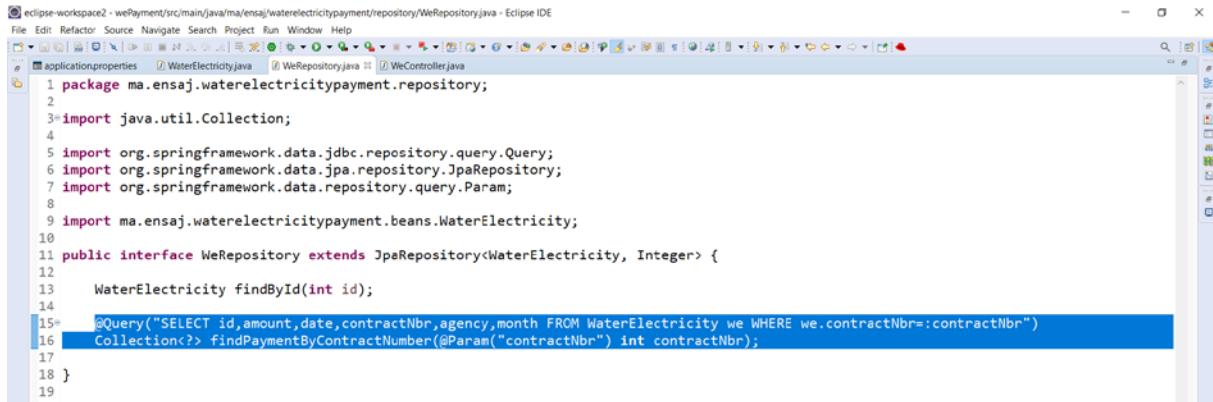
44. The payment with id 3 was deleted successfully

The phpMyAdmin interface displays the `water_electricity` table. The table structure includes columns: `id`, `agency`, `amount`, `contract_nbr`, `date`, and `month`. The current view shows rows 0-2. The first two rows have `id` values 1 and 2 respectively, while the third row has `id` value 4. The interface also shows options for editing, copying, deleting, and exporting data.

	<code>id</code>	<code>agency</code>	<code>amount</code>	<code>contract_nbr</code>	<code>date</code>	<code>month</code>
1	1	RADEEJ	400	1234567	2021-01-04 00:00:00	2
2	2	RADEEJ	489.9	1234567	2021-02-06 00:00:00	3
3	4	ONE	489.9	1234567	2021-02-06 00:00:00	3

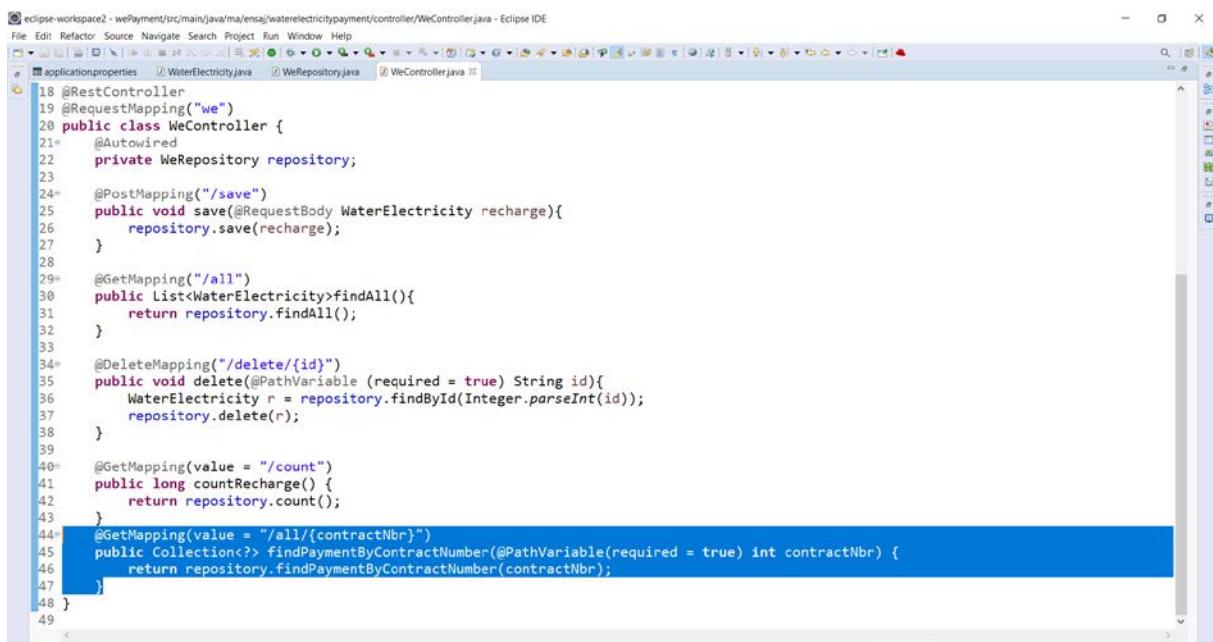
Figure 73:The payment with id 3 was deleted successfully

45. Get payments by contract number



```
eclipse-workspace2 - wePayment/src/main/java/ma/ensaj/waterelectricitypayment/repository/WeRepository.java - Eclipse IDE
File Edit Refactor Source Navigate Search Project Run Window Help
application.properties WaterElectricity.java WeRepository.java WeController.java
1 package ma.ensaj.waterelectricitypayment.repository;
2
3 import java.util.Collection;
4
5 import org.springframework.data.jdbc.repository.query.Query;
6 import org.springframework.data.jpa.repository.JpaRepository;
7 import org.springframework.data.repository.query.Param;
8
9 import ma.ensaj.waterelectricitypayment.beans.WaterElectricity;
10
11 public interface WeRepository extends JpaRepository<WaterElectricity, Integer> {
12
13     WaterElectricity findById(int id);
14
15     @Query("SELECT id,amount,date,contractNbr,agency,month FROM WaterElectricity we WHERE we.contractNbr=:contractNbr")
16     Collection<?> findPaymentByContractNumber(@Param("contractNbr") int contractNbr);
17
18 }
19
```

Figure 74: Get payments by contract number



```
eclipse-workspace2 - wePayment/src/main/java/ma/ensaj/waterelectricitypayment/controller/WeController.java - Eclipse IDE
File Edit Refactor Source Navigate Search Project Run Window Help
application.properties WaterElectricity.java WeRepository.java WeController.java
18 @RestController
19 @RequestMapping("we")
20 public class WeController {
21     @Autowired
22     private WeRepository repository;
23
24     @PostMapping("/save")
25     public void save(@RequestBody WaterElectricity recharge){
26         repository.save(recharge);
27     }
28
29     @GetMapping("/all")
30     public List<WaterElectricity>findAll(){
31         return repository.findAll();
32     }
33
34     @DeleteMapping("/delete/{id}")
35     public void delete(@PathVariable (required = true) String id){
36         WaterElectricity r = repository.findById(Integer.parseInt(id));
37         repository.delete(r);
38     }
39
40     @GetMapping(value = "/count")
41     public long countRecharge() {
42         return repository.count();
43     }
44     @GetMapping(value = "/all/{contractNbr}")
45     public Collection<?> findPaymentByContractNumber(@PathVariable(required = true) int contractNbr) {
46         return repository.findPaymentByContractNumber(contractNbr);
47     }
48 }
```

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

The screenshot shows the Advanced REST client interface. On the left, a sidebar displays a history of API requests. In the main panel, a request is being configured for `HTTP://localhost:8080/we/all/1234567` using the `GET` method. The response status is `200 Not Found` with a duration of `451.89 ms`. The response body is a JSON array with two elements:

```
[Array(2)
  -0: [Array(6)
    0: 1,
    1: "RADEEJ",
    2: 400,
    3: 1234567,
    4: "2021-01-04",
    5: 2
  ],
  -1: [Array(6)
    0: 2,
    1: "RADEEJ",
    2: 489.9,
    3: 1234567,
    4: "2021-02-06",
    5: 3
  ],
  -2: [Array(6)
    0: 4,
    1: "ONE",
    2: 489.9,
    3: 1234567,
    4: "2021-02-06",
    5: 3
  ],
]
```

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

The screenshot shows the Advanced REST client interface. On the left, a sidebar displays a history of API requests. In the main panel, a request is being configured for `HTTP://localhost:8080/we/all/1234567` using the `GET` method. The response status is `200 OK` with a duration of `451.89 ms`. The response body is a JSON array with three elements, each containing an array of six items:

```
[Array(3)
  -0: [Array(6)
    0: 1,
    1: "RADEEJ",
    2: 400,
    3: 1234567,
    4: "2021-01-04",
    5: 2
  ],
  -1: [Array(6)
    0: 2,
    1: "RADEEJ",
    2: 489.9,
    3: 1234567,
    4: "2021-02-06",
    5: 3
  ],
  -2: [Array(6)
    0: 4,
    1: "ONE",
    2: 489.9,
    3: 1234567,
    4: "2021-02-06",
    5: 3
  ],
]
```

4.1.3 Phone Top Up (recharge) Microservice Implementation

1. Initial creation of the application with Spring Initializr

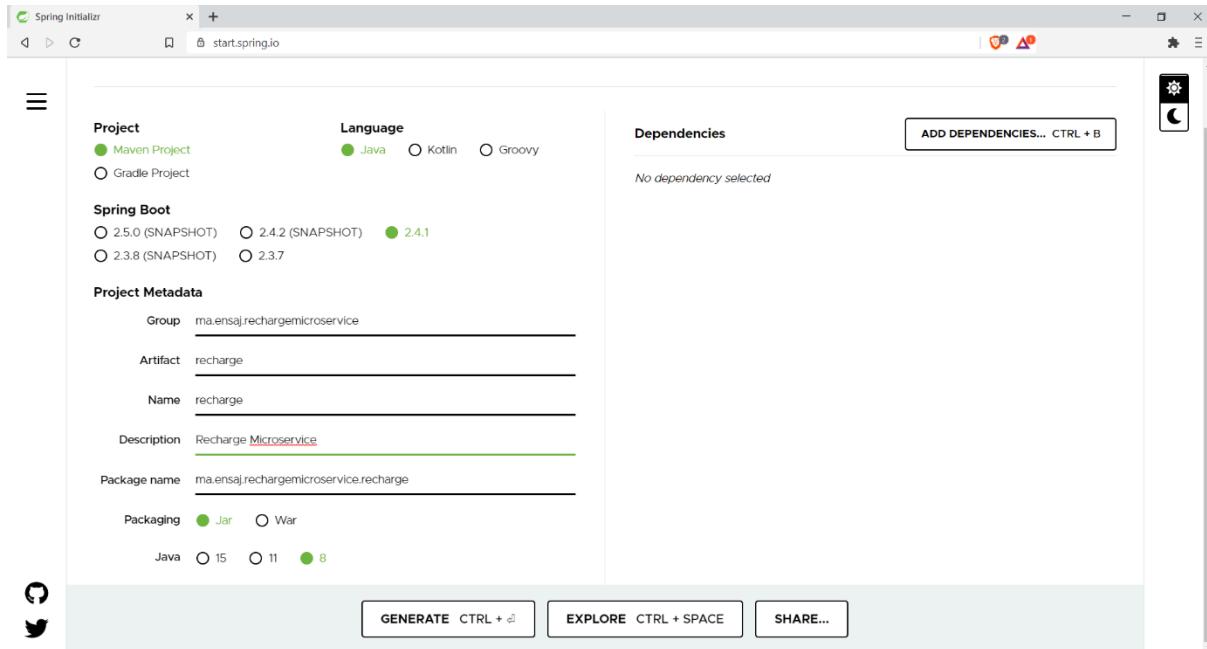


Figure 77:Spring Initializr

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:

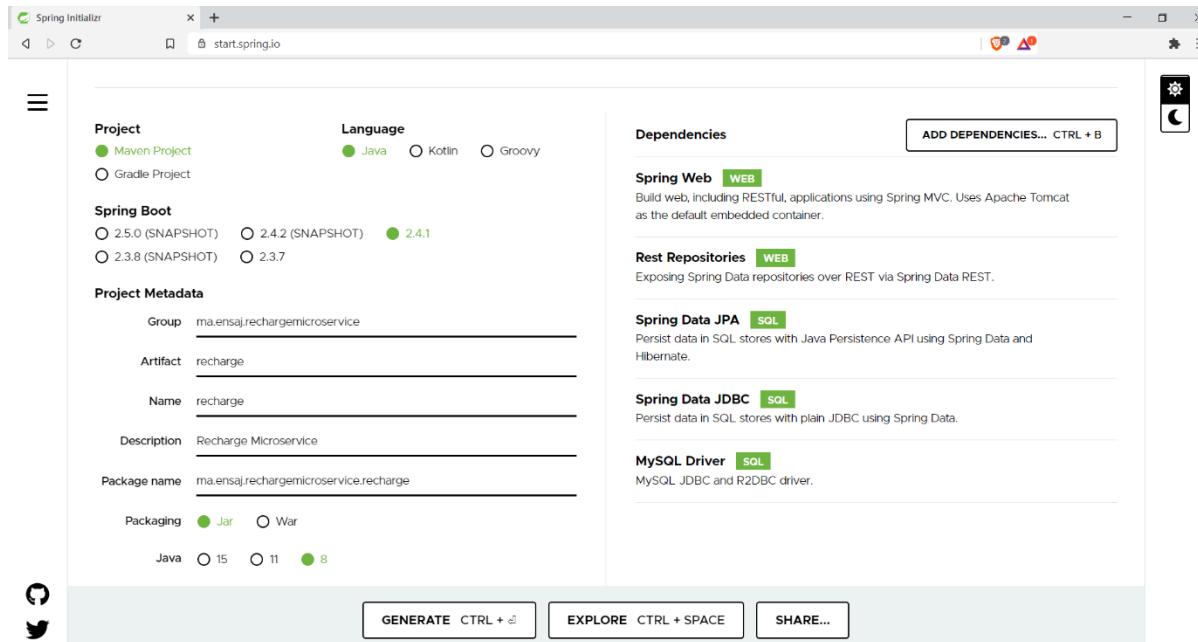


Figure 78:Initializr configured

3. Download the project and extract it

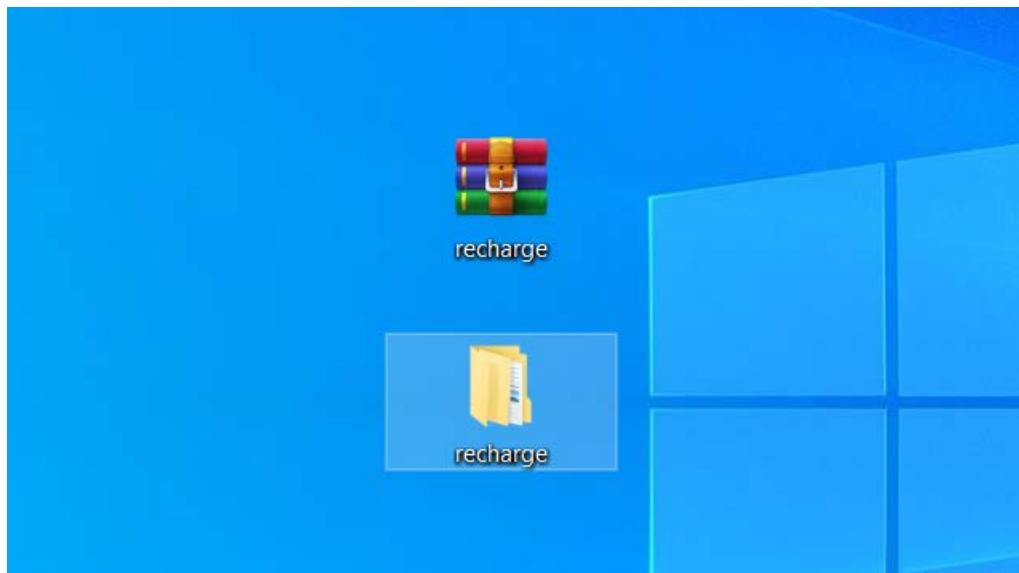


Figure 79:Download the project and extract it

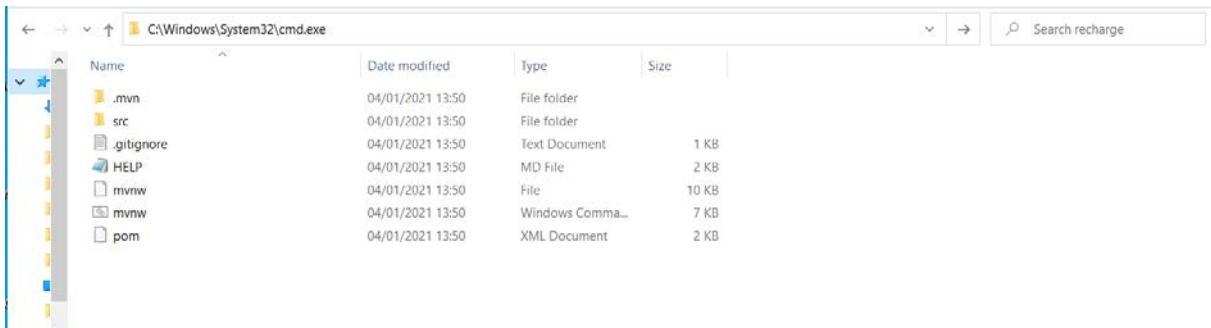


Figure 80:Project in the folder explorer

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

```
C:\Windows\System32\cmd.exe
C:\Users\hhass\Desktop\recharge\recharge>mvn eclipse:eclipse
```

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

```
C:\Windows\System32\cmd.exe
C:\Users\hhass\Desktop\recharge\recharge>mvn eclipse:eclipse
[INFO] Scanning for projects...
[INFO]
[INFO] -----< ma.ensaj.rechargemicroservice:recharge >-----
[INFO] Building recharge 0.0.1-SNAPSHOT
[INFO] -----[ jar ]-----
[INFO]
[INFO] >>> maven-eclipse-plugin:2.10:eclipse (default-cli) > generate-resources @ recharge >>
[INFO]
[INFO] <<< maven-eclipse-plugin:2.10:eclipse (default-cli) < generate-resources @ recharge <<<
[INFO]
[INFO]
[INFO] --- maven-eclipse-plugin:2.10:eclipse (default-cli) @ recharge ---
[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 "recharge" to C:\Users\hhass\Desktop\recharge\recharge.
[INFO]
[INFO] -----
[INFO] BUILD SUCCESS
[INFO]
[INFO] Total time: 3.309 s
[INFO] Finished at: 2021-01-04T14:54:07+01:00
[INFO] -----
```

C:\Users\hhass\Desktop\recharge\recharge>

Figure 82:Build success

6. Import it under Eclipse

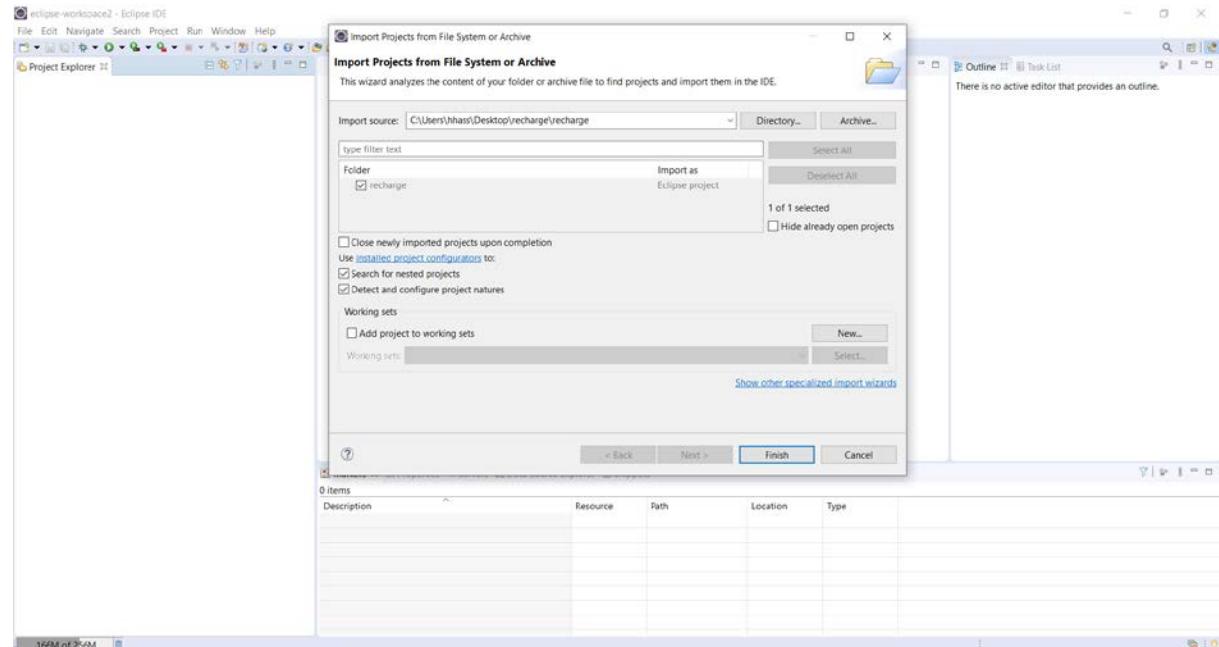


Figure 83:Import it under Eclipse

7. Project structure

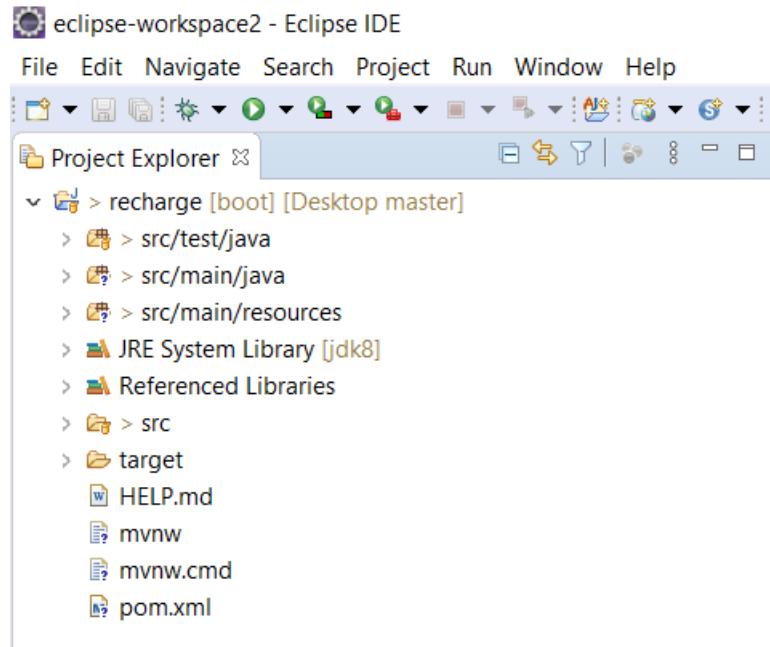


Figure 84:Project structure

8. The contents of pom.xml :

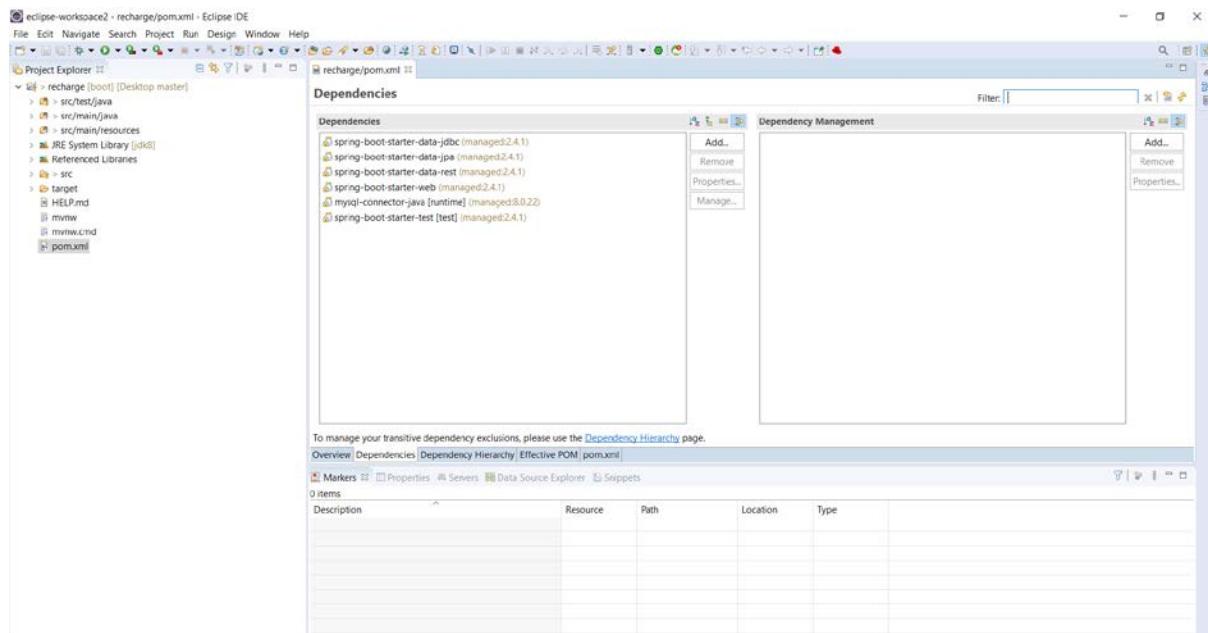
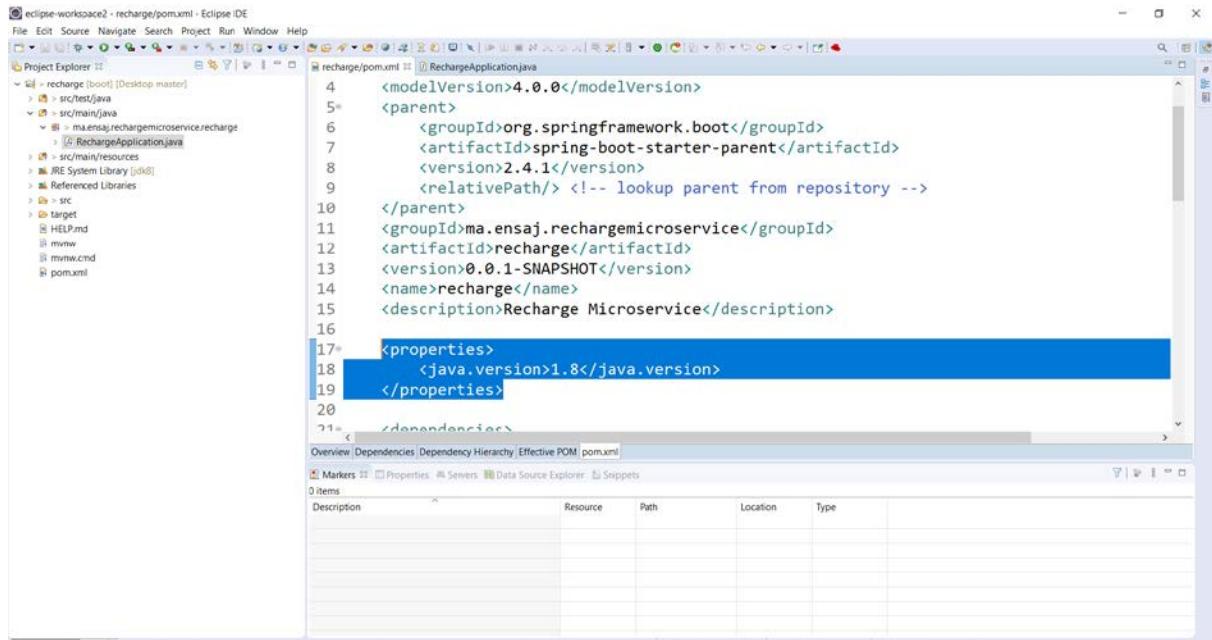


Figure 85:The contents of pom.xml

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:



```

eclipse-workspace2 - recharge/pom.xml - Eclipse IDE
File Edit Source Navigate Project Run Window Help
Project Explorer [recharge [boot] [Desktop master]]
  > src/test/java
  > src/main/java
    > ma.ensaj.rechargemicroservice.recharge
      > RechargeApplication.java
  > src/main/resources
  > .RE System Library [jdk]
  > Referenced Libraries
  > src
  > target
  > HELP.MD
  > minw
  > minw.cmd
  > pom.xml

4  <modelVersion>4.0.0</modelVersion>
5  <parent>
6    <groupId>org.springframework.boot</groupId>
7    <artifactId>spring-boot-starter-parent</artifactId>
8    <version>2.4.1</version>
9    <relativePath/> <!-- lookup parent from repository -->
10   </parent>
11   <groupId>ma.ensaj.rechargemicroservice</groupId>
12   <artifactId>recharge</artifactId>
13   <version>0.0.1-SNAPSHOT</version>
14   <name>recharge</name>
15   <description>Recharge Microservice</description>
16
17   <properties>
18     <java.version>1.8</java.version>
19   </properties>
20
21</dependencies>

```

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).

```

eclipse-workspace2 - recharge/src/main/java/ma/ensaj/rechargemicroservice/recharge/RechargeApplication.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer Recharge [boot] [Desktop master]
src/test/java
src/main/java
ma.ensaj.rechargemicroservice.recharge
RechargeApplication.java
src/main/resources
JRE System Library [jdk8]
Referenced Libraries
src
target
HELP.md
mnw
mnw.cmd
pom.xml
recharge/pom.xml RechargeApplication.java
1 package ma.ensaj.rechargemicroservice.recharge;
2
3 import org.springframework.boot.SpringApplication;
4
5 @SpringBootApplication
6 public class RechargeApplication {
7
8     public static void main(String[] args) {
9         SpringApplication.run(RechargeApplication.class, args);
10    }
11 }
12
13 }
14
Markers Properties Servers Data Source Explorer Snippets
0 items
Description Resource Path Location Type
```

Figure 87:@SpringBootApplication

11. Configuration of the database and the Persistence Framework

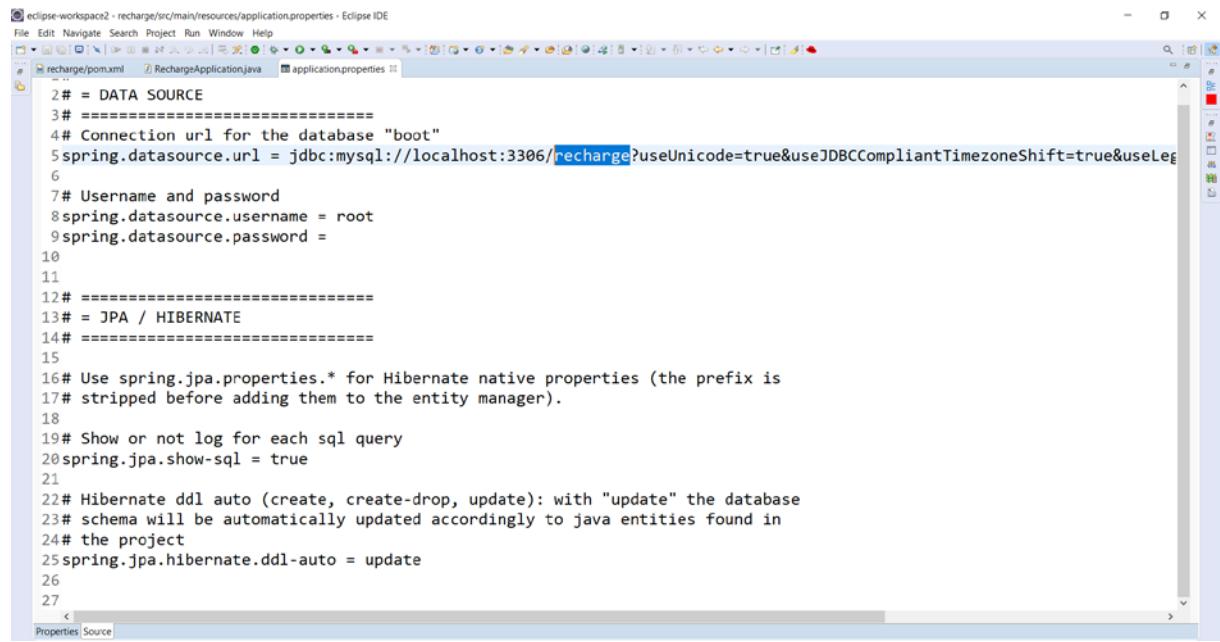
Spring Boot comes with a built-in mechanism for configuring applications using a file called **application.properties**. It is located in the **src / main / resources** folder, as shown in the following figure.

```

eclipse-workspace2 - recharge/src/main/resources/application.properties - Eclipse IDE
File Edit Navigate Search Project Run Window Help
Project Explorer recharge [boot] [Desktop master]
src/test/java
src/main/java
src/main/resources
templates
static
application.properties
JRE System Library [jdk8]
Referenced Libraries
src
target
HELP.md
mnw
mnw.cmd
pom.xml
recharge/pom.xml RechargeApplication.java application.properties
1
2
Properties Source
Markers Properties Servers Data Source Explorer Snippets
0 items
Description Resource Path Location Type
```

Figure 88:application.properties

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



The screenshot shows the Eclipse IDE interface with the title bar "eclipse-workspace2 - recharge/src/main/resources/application.properties - Eclipse IDE". The application.properties file is open in the central editor area. The code content is as follows:

```
2# = DATA SOURCE
3# =====
4# Connection url for the database "boot"
5spring.datasource.url = jdbc:mysql://localhost:3306/recharge?useUnicode=true&useJDBCCompliantTimezoneShift=true&useLegacyDatetimeCode=false&serverTimezone=UTC
6
7# Username and password
8spring.datasource.username = root
9spring.datasource.password =
10
11
12# =====
13# = JPA / HIBERNATE
14# =====
15
16# Use spring.jpa.properties.* for Hibernate native properties (the prefix is
17# stripped before adding them to the entity manager).
18
19# Show or not log for each sql query
20spring.jpa.show-sql = true
21
22# Hibernate ddl auto (create, create-drop, update): with "update" the database
23# schema will be automatically updated accordingly to java entities found in
24# the project
25spring.jpa.hibernate.ddl-auto = update
26
27
```

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

13. Create the "recharge" database under MySQL

The screenshot shows the phpMyAdmin interface on a local server at 127.0.0.1. In the top navigation bar, there are tabs for Databases, SQL, Status, User accounts, Export, Import, Settings, Replication, Variables,Charsets, Engines, and Plugins. The current tab is 'Databases'. Below the tabs, a search bar contains 'recharge' and a dropdown menu shows 'utf8mb4_general_ci'. A 'Create' button is visible. Under the 'Create database' section, there is a 'Filters' button and a 'Containing the word:' input field. The main area displays a table titled 'Databases' with columns 'Database', 'Collation', and 'Action'. The table lists various databases, each with a checkbox and a 'Check privileges' link. The newly created 'recharge' database is listed at the bottom of the table.

Figure 90:"recharge" database under MySQL

14. Creation of business classes

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

The screenshot shows the Eclipse IDE interface with a project named 'recharge' open in the Project Explorer. The code editor shows a Java file named 'RechargeApplication.java' with the following code:

```
1 package ma.ensaj.rechargemicroservice;
2
3 import org.springframework.boot.SpringApplication;
4
5 @SpringBootApplication
6 public class RechargeApplication {
7     public static void main(String[] args) {
8         SpringApplication.run(RechargeApplication.class, args);
9     }
10 }
```

A 'New Java Package' dialog box is displayed over the code editor. It shows the 'Source folder:' as 'recharge/src/main/java' and the 'Name:' as 'ma.ensaj.rechargemicroservice.beans'. There are checkboxes for 'Create package-info.java' and 'Generate comments (configure templates and default value here)'. At the bottom of the dialog are 'Finish' and 'Cancel' buttons.

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):

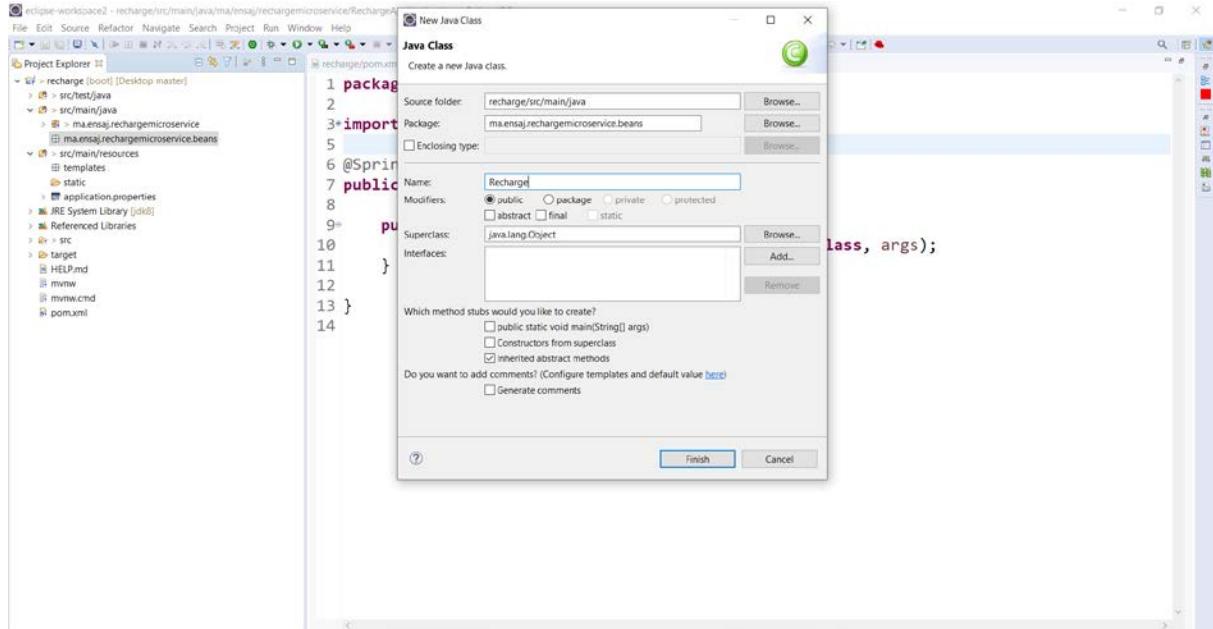


Figure 92:Create the "Recharge" class

17. Class diagram of Recharge entity

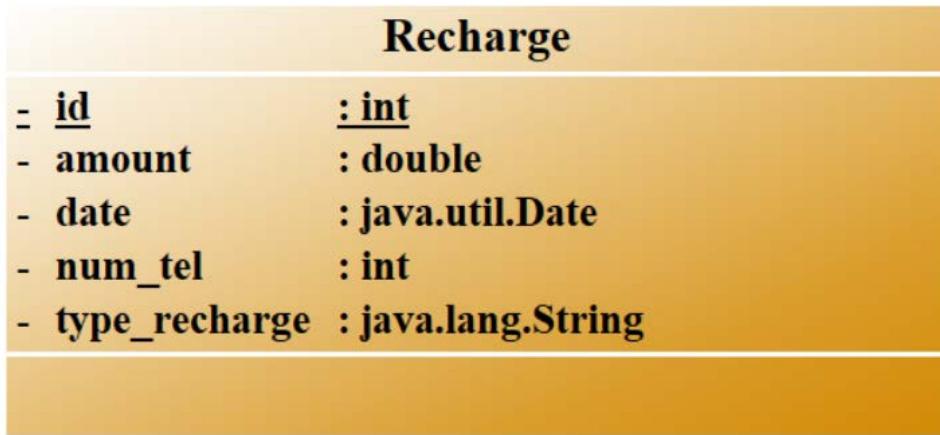
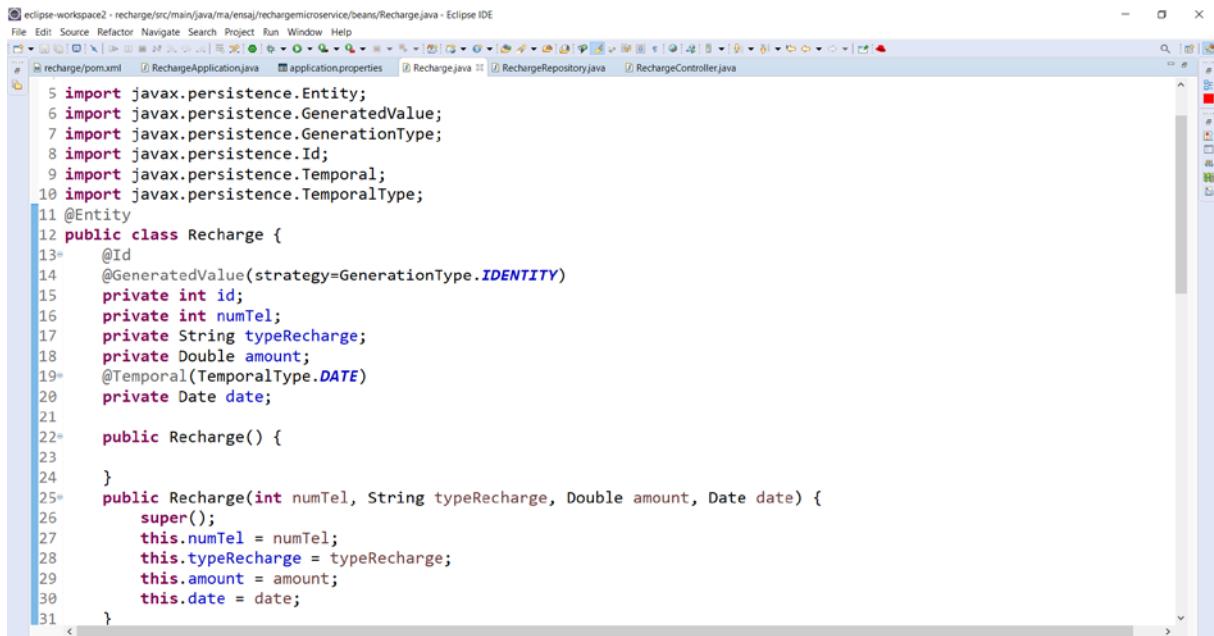


Figure 93:Class diagram of Recharge entity

18. Implementation of Recharge class



The screenshot shows the Eclipse IDE interface with the title bar "eclipse-workspace2 - recharge/src/main/java/ma/ensa/rechargemicroservice/beans/Recharge.java - Eclipse IDE". The code editor displays the following Java code:

```
5 import javax.persistence.Entity;
6 import javax.persistence.GeneratedValue;
7 import javax.persistence.GenerationType;
8 import javax.persistence.Id;
9 import javax.persistence.Temporal;
10 import javax.persistence.TemporalType;
11 @Entity
12 public class Recharge {
13     @Id
14     @GeneratedValue(strategy=GenerationType.IDENTITY)
15     private int id;
16     private int numTel;
17     private String typeRecharge;
18     private Double amount;
19     @Temporal(TemporalType.DATE)
20     private Date date;
21
22     public Recharge() {
23
24     }
25     public Recharge(int numTel, String typeRecharge, Double amount, Date date) {
26         super();
27         this.numTel = numTel;
28         this.typeRecharge = typeRecharge;
29         this.amount = amount;
30         this.date = date;
31     }
}
```

Figure 94:Implementation of Recharge class

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

```
C:\Windows\System32\cmd.exe - mvn spring-boot:run
```

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

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

20. We can see now in the cmd Started RechargeApplication

```

Select C:\Windows\System32\cmd.exe - mvn spring-boot:run
2021-01-04 15:43:03.903 INFO 14548 --- [           main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2021-01-04 15:43:03.904 INFO 14548 --- [           main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.41]
2021-01-04 15:43:04.132 INFO 14548 --- [          applicationContext
lization completed in 3787 ms
2021-01-04 15:43:04.393 INFO 14548 --- [       mainInfo [name: default]
2021-01-04 15:43:04.462 INFO 14548 --- [ion 5.4.25.Final
2021-01-04 15:43:04.634 INFO 14548 --- [otations {5.1.2.Final}
2021-01-04 15:43:04.777 INFO 14548 --- [       main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Starting...
2021-01-04 15:43:05.210 INFO 14548 --- [       main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Start completed.
2021-01-04 15:43:05.231 INFO 14548 --- [       main] org.hibernate.dialect.Dialect      : HHH000400: Using dialect: org.hibernate.dialect.MySQL55Dialect
Hibernate: create table recharge (num_tel integer not null, amount double precision, date date, type_recharge varchar(255), primary key (num_tel)) engine=InnoDB
2021-01-04 15:43:07.176 INFO 14548 --- [           main] o.h.e.t.j.p.i.JtaPlatformInitiator      : HHH000490: Using JtaPlatform implementation: [org.hibernate.engine.transaction.jta.platform.internal.NoJtaPlatform]
2021-01-04 15:43:07.195 INFO 14548 --- [           main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
2021-01-04 15:43:07.374 WARN 14548 --- [           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
2021-01-04 15:43:07.960 INFO 14548 --- [           main] o.s.s.concurrent.ThreadPoolTaskExecutor : Initializing ExecutorService 'applicationTaskExecutor'
2021-01-04 15:43:09.038 INFO 14548 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context path ''
2021-01-04 15:43:09.058 INFO 14548 --- [           main] m.e.r.RechargeApplication            : Started RechargeApplication in 9.653 seconds (JVM running for 10.509)

```

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

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

The screenshot shows the phpMyAdmin interface. On the left, the database structure is visible with the 'recharge' database selected. Inside 'recharge', there is a table named 'recharge'. The table has the following structure:

	id	amount	date	num_tel	type_recharge

Below the table, a SQL query is run: `SELECT * FROM `recharge``. The results pane shows: "MySQL returned an empty result set (i.e. zero rows). (Query took 0.0012 seconds.)".

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

22. Insert a new row to the table recharge

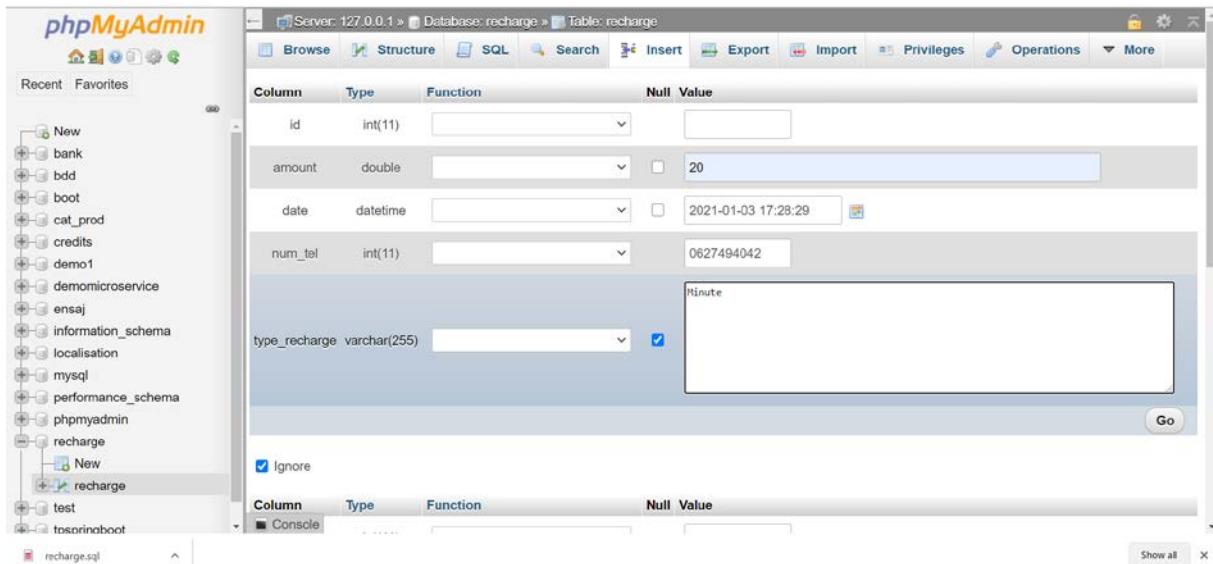


Figure 98:Insert a new row to the table recharge

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 :

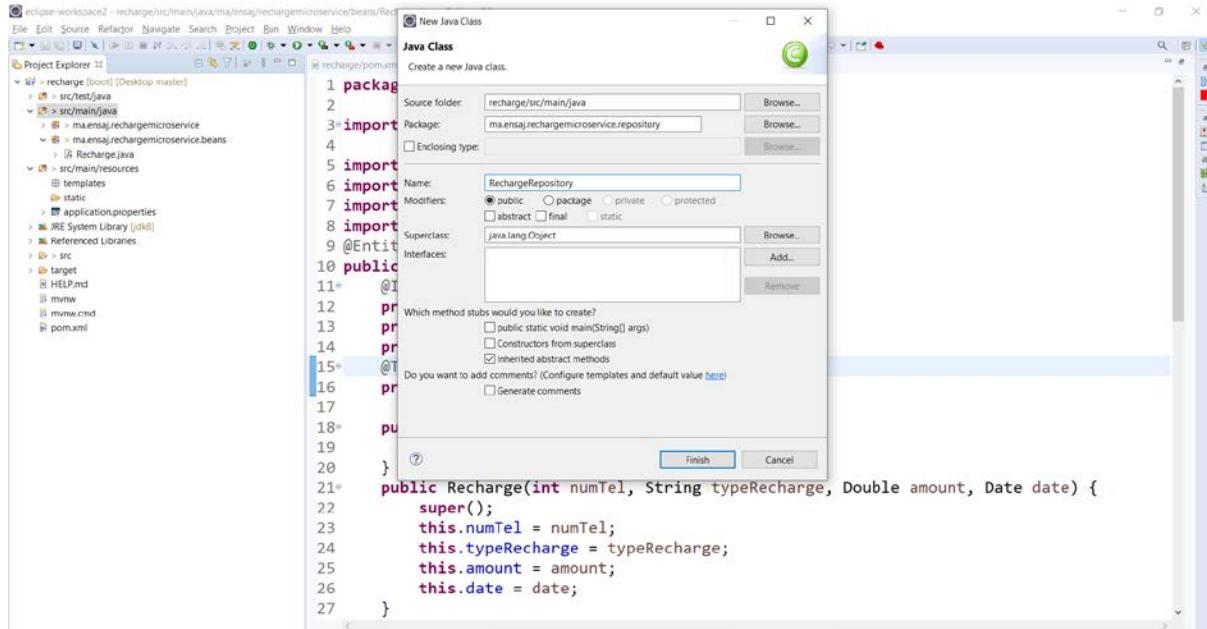


Figure 99:Create the "RechargeRepository" interface

25. RechargeRepository extends JpaRepository<Recharge, Integer>

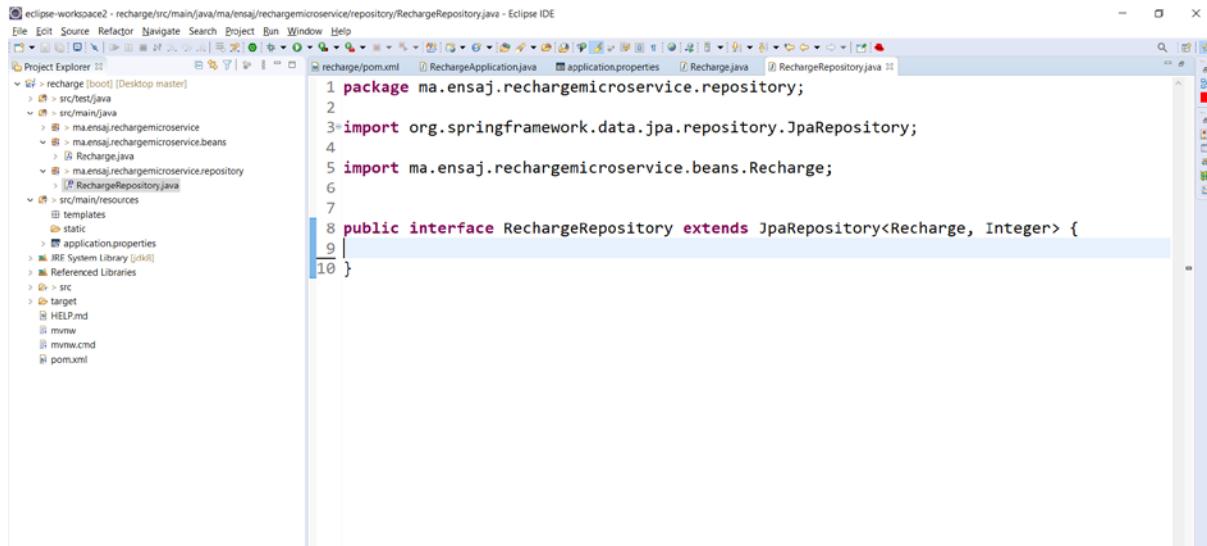
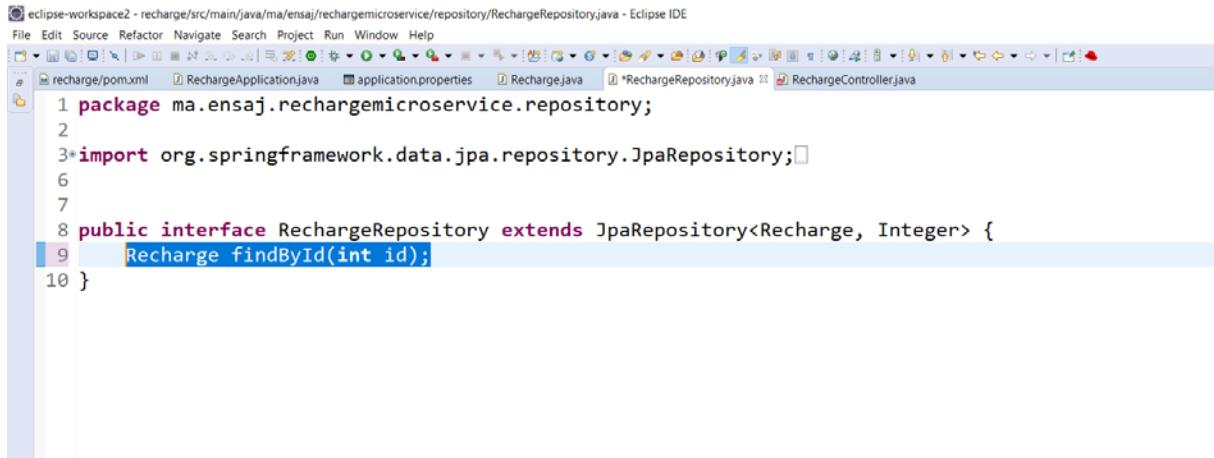


Figure 100: "RechargeRepository"

26. In the interface add the method :

Recharge findById(int id);



The screenshot shows the Eclipse IDE interface with the title bar "eclipse-workspace2 - recharge/src/main/java/ma/ensaj/rechargemicroservice/repository/RechargeRepository.java - Eclipse IDE". The menu bar includes File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, and Help. The toolbar has various icons for file operations like Open, Save, Cut, Copy, Paste, Find, and Run. The left sidebar shows the project structure with files like recharge/pom.xml, RechargeApplication.java, application.properties, Recharge.java, *RechargeRepository.java, and RechargeController.java. The main editor area displays the Java code for the RechargeRepository interface:

```
1 package ma.ensaj.rechargemicroservice.repository;
2
3 import org.springframework.data.jpa.repository.JpaRepository;
4
5
6 public interface RechargeRepository extends JpaRepository<Recharge, Integer> {
7     Recharge findById(int id);
8 }
9
10 }
```

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

27. Creation of REST controllers

REST stands for **R**Epresentational **S**tate **T**ransfer. 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:

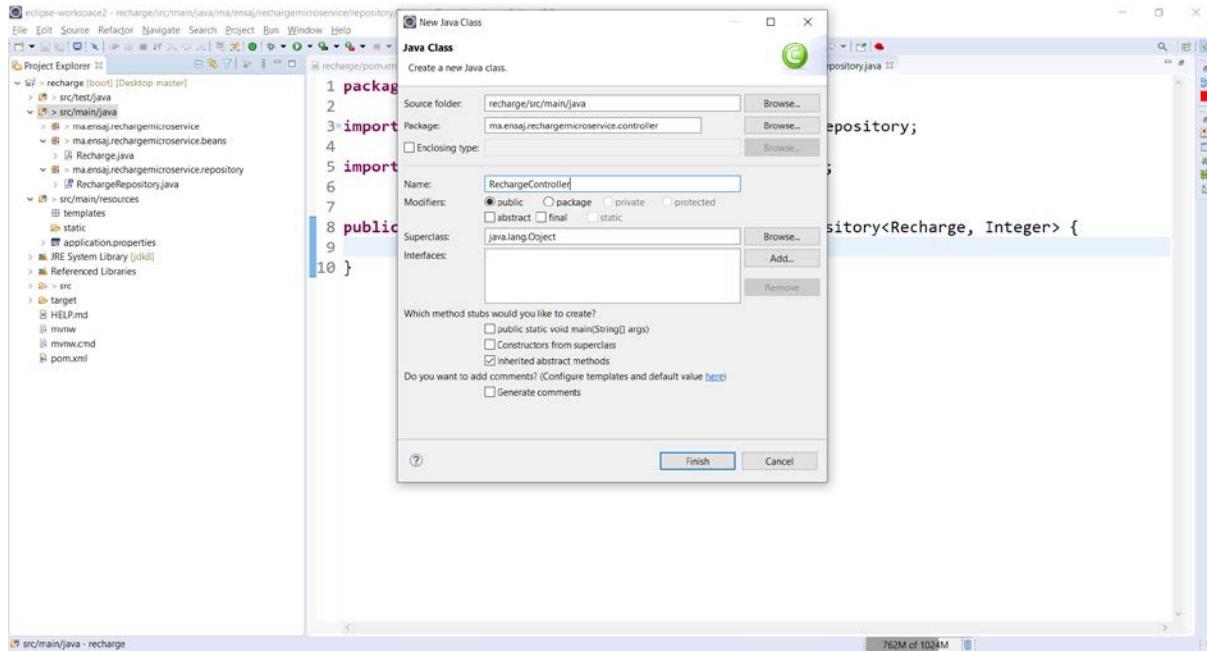


Figure 102:Create the "RechargeController" class

29. RechargeController class

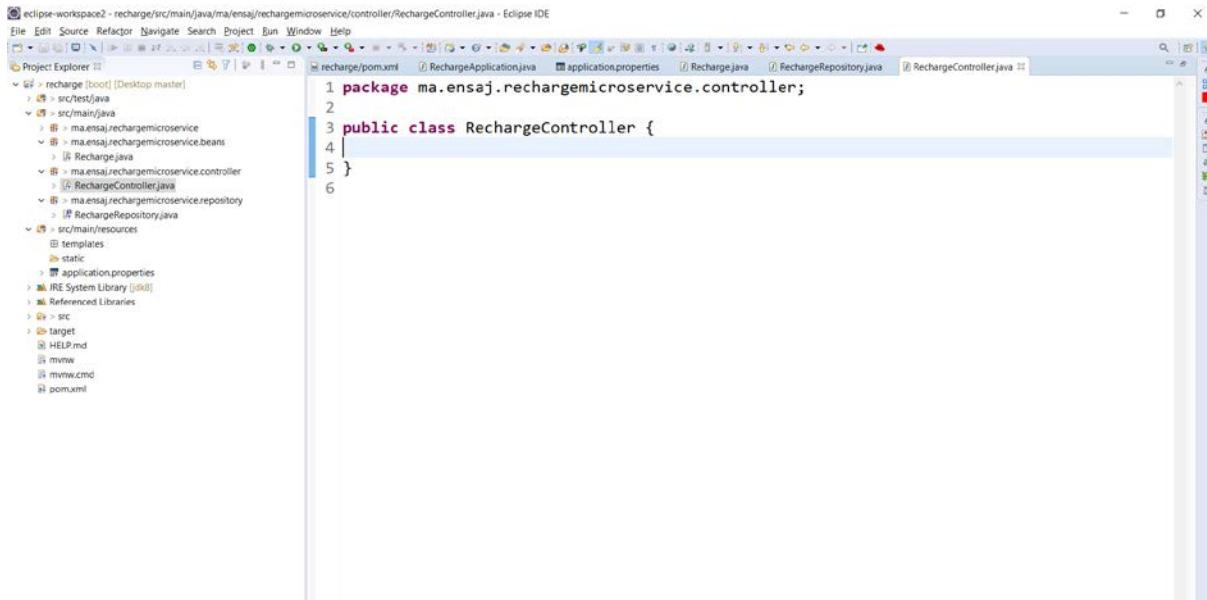
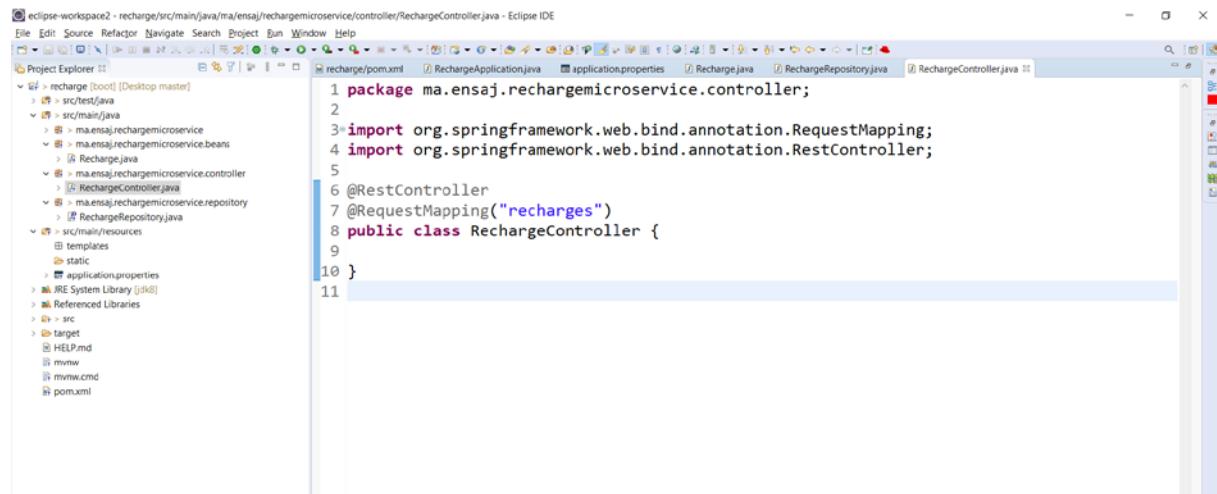


Figure 103:RechargeController class

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.

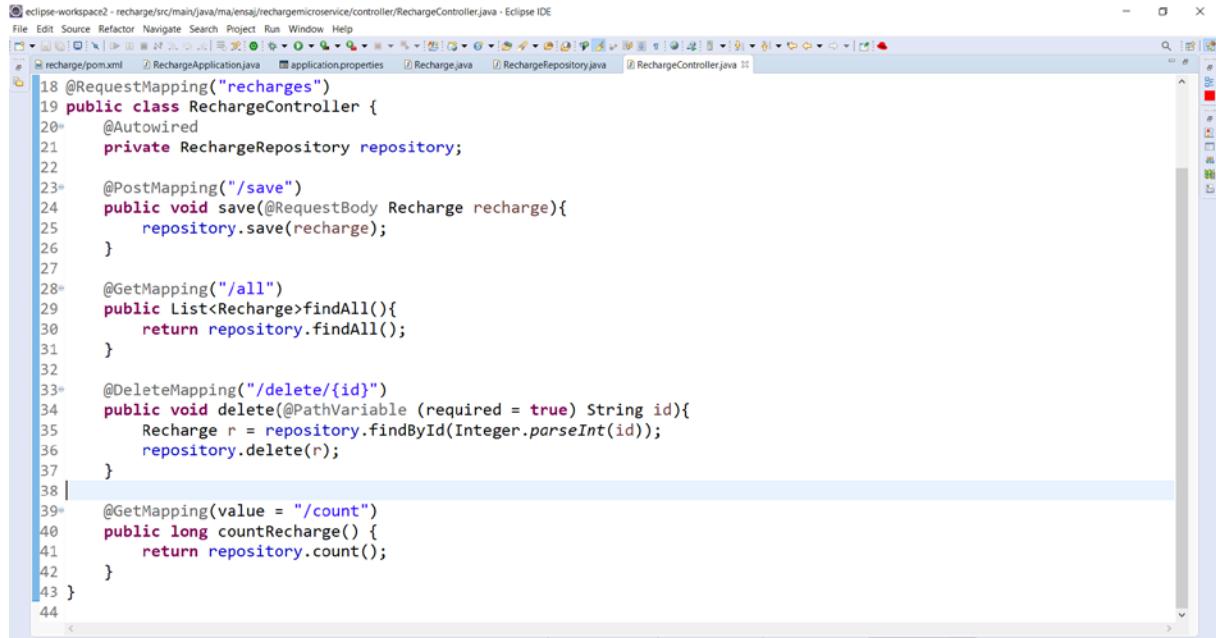


The screenshot shows the Eclipse IDE interface with the 'RechargeController.java' file open in the editor. The code is as follows:

```
1 package ma.ensaj.rechargemicroservice.controller;
2
3 import org.springframework.web.bind.annotation.RequestMapping;
4 import org.springframework.web.bind.annotation.RestController;
5
6 @RestController
7 @RequestMapping("recharges")
8 public class RechargeController {
9
10 }
```

Figure 104:@RestController and @RequestMapping("recharges")

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



The screenshot shows the Eclipse IDE interface with the file 'RechargeController.java' open in the central editor window. The code implements a CRUD controller for a 'Recharge' entity. It includes methods for saving a new recharge, getting all recharges, deleting a recharge by ID, and counting the total number of recharges.

```
18 @RequestMapping("recharges")
19 public class RechargeController {
20     @Autowired
21     private RechargeRepository repository;
22
23     @PostMapping("/save")
24     public void save(@RequestBody Recharge recharge){
25         repository.save(recharge);
26     }
27
28     @GetMapping("/all")
29     public List<Recharge>findAll(){
30         return repository.findAll();
31     }
32
33     @DeleteMapping("/delete/{id}")
34     public void delete(@PathVariable (required = true) String id){
35         Recharge r = repository.findById(Integer.parseInt(id));
36         repository.delete(r);
37     }
38
39     @GetMapping(value = "/count")
40     public long countRecharge() {
41         return repository.count();
42     }
43 }
44
```

Figure 105: Add CRUD web services to the RechargeController

32. Testing controllers with ARC(Advanced Rest Client)

33. List of phone top up(recharge)

The screenshot shows the Advanced REST client interface. On the left, there is a history panel with several recent requests. In the main central area, a new request is being configured: Method is set to GET, Request URL is http://localhost:8080/recharges/all. Below the URL, under 'Request parameters', there is a section for 'HEADERS', 'AUTHORIZATION', 'ACTIONS', 'CONFIG', and 'CODE'. Under 'CODE', there is a 'COPY' button and a 'SOURCE VIEW' tab. The 'SOURCE VIEW' tab displays the JSON response: [Array[1]] -> 0: { "id": 1, "numTel": 627494842, "typeRecharge": "Minute" }.

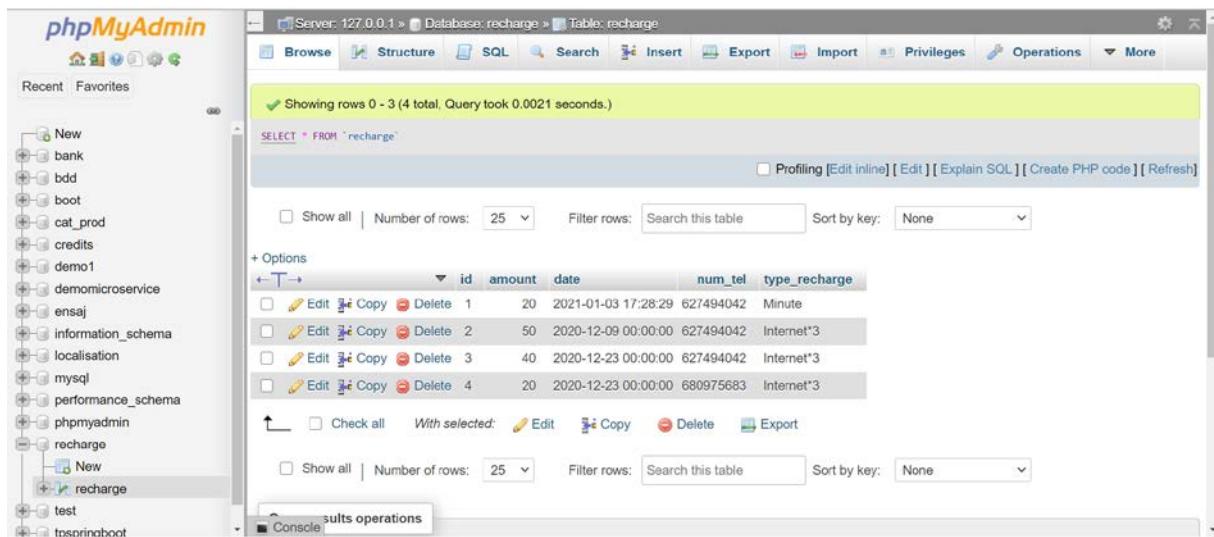
Figure 106>List of phone top up(recharge)

34. Creation(Add some recharge)

The screenshot shows the Advanced REST client interface. On the left, there is a history panel with several recent requests. In the main central area, a new request is being configured: Method is set to POST, Request URL is http://localhost:8080/recharges/save. Below the URL, under 'Request parameters', there is a section for 'HEADERS', 'BODY', 'AUTHORIZATION', 'ACTIONS', 'CONFIG', and 'CODE'. Under 'BODY', the 'Body content type' is set to 'application/json'. The 'BODY' tab displays the JSON request body: { "id": 1, "numTel": 627494042, "typeRecharge": "Internet*3", "amount": 50, "date": "2020-12-09" }. Below the body, under 'FORMAT JSON', there is a 'COPY' button and a 'MINIFY JSON' tab. The 'FORMAT JSON' tab displays the JSON response: { "id": 1, "numTel": 627494042, "typeRecharge": "Internet*3", "amount": 50, "date": "2020-12-09" }.

Figure 107:Creation(Add some recharge)

35. Check if recharges were added successfully to the database

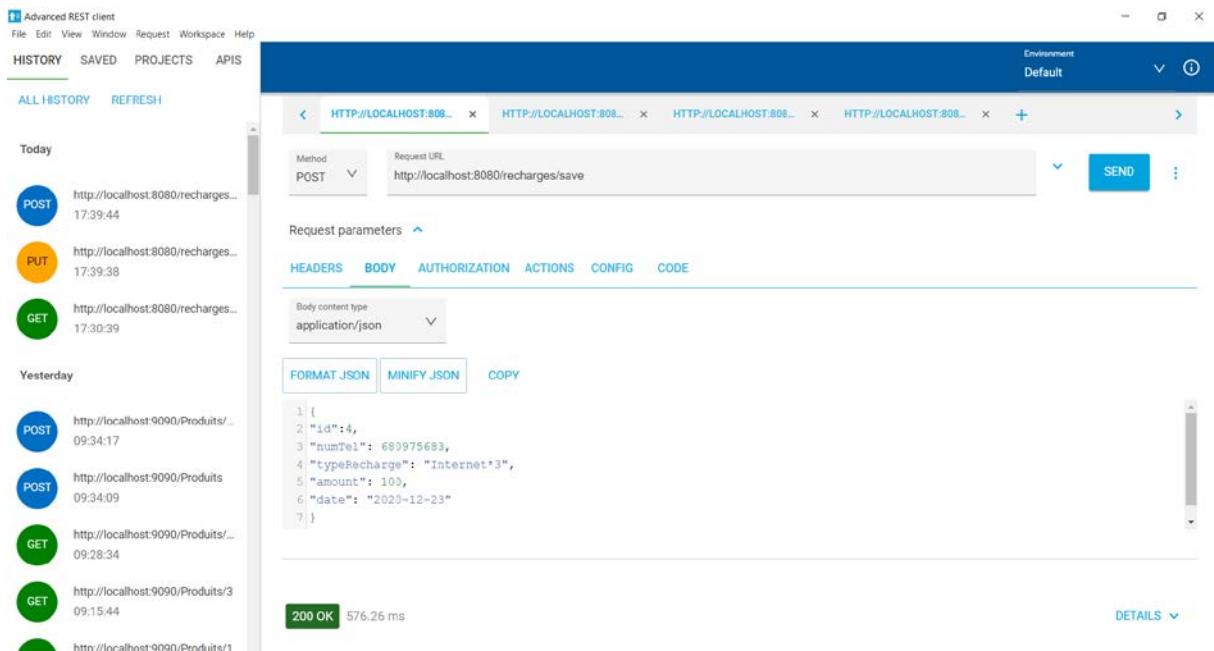


The screenshot shows the phpMyAdmin interface connected to a MySQL database named 'recharge'. The left sidebar lists various databases and tables. The main area displays the 'recharge' table with the following data:

	id	amount	date	num_tel	type_recharge
<input type="checkbox"/>	1	20	2021-01-03 17:28:29	627494042	Minute
<input type="checkbox"/>	2	50	2020-12-09 00:00:00	627494042	Internet*3
<input type="checkbox"/>	3	40	2020-12-23 00:00:00	627494042	Internet*3
<input type="checkbox"/>	4	20	2020-12-23 00:00:00	680975683	Internet*3

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

36. Update the recharge with id 4



The screenshot shows the Advanced REST client interface. The history panel on the left lists several requests. The main panel shows a POST request to 'http://localhost:8080/recharges/save' with the following JSON body:

```
1 {
2 "id":4,
3 "numTel": "680975683",
4 "typeRecharge": "Internet*3",
5 "amount": 100,
6 "date": "2020-12-23"
7 }
```

The response at the bottom indicates a successful 200 OK status with a response time of 576.26 ms.

Figure 109:Update the recharge with id 4

37. The update is well achieved

The screenshot shows the phpMyAdmin interface for the 'recharge' database. The left sidebar lists various databases and tables, with 'recharge' selected. The main area displays the 'recharge' table with the following data:

	id	amount	date	num_tel	type_recharge
<input type="checkbox"/>	1	20	2021-01-03 17:28:29	627494042	Minute
<input type="checkbox"/>	2	50	2020-12-09 00:00:00	627494042	Internet'3
<input type="checkbox"/>	3	40	2020-12-23 00:00:00	627494042	Internet'3
<input type="checkbox"/>	4	100	2020-12-23 00:00:00	680975683	Internet'3

Figure 110:The update is well achieved

38. Count test

The screenshot shows the Advanced REST client interface. On the left, a history panel lists API requests. The main panel shows a request configuration for a GET method to the URL `http://localhost:8080/recharges/count`. The response section shows a 200 OK status with a response body containing the number 4.

Figure 111:Count test

39. Delete recharge with id 3

The screenshot shows the Advanced REST client interface. On the left, there's a history panel with several log entries. In the main area, a new request is being configured: Method is set to DELETE, and the Request URL is http://localhost:8080/recharges/delete/3. Below this, the response status is shown as 200 OK with a time of 582.03 ms.

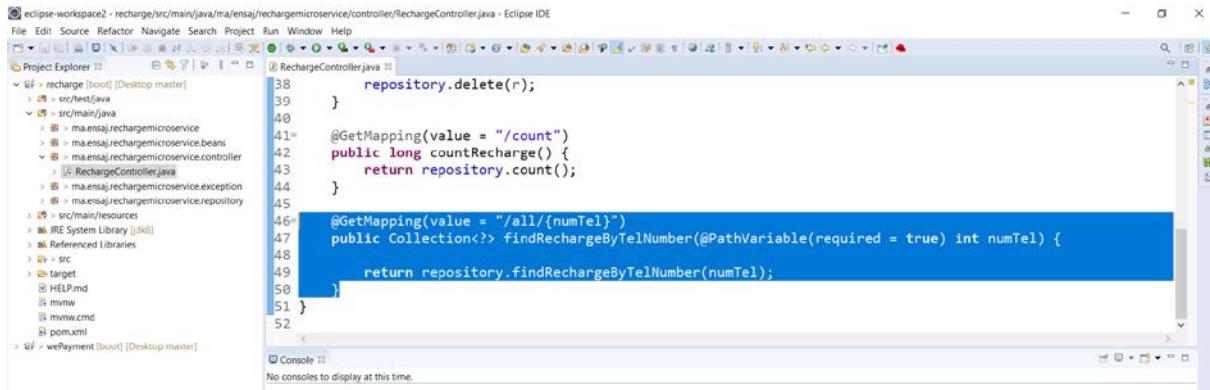
Figure 112:Delete recharge with id 3

40. The recharge with id 3 was deleted successfully

The screenshot shows the phpMyAdmin interface for the recharge database. The 'Structure' tab is selected. In the main pane, the recharge table is displayed with three rows of data. The row with id 3 has been highlighted, indicating it is selected for deletion. The 'Operations' toolbar at the bottom right shows the delete icon.

Figure 113:The recharge with id 3 was deleted successfully

41. Get recharge by telephone number



The screenshot shows the Eclipse IDE interface with the 'RechargeController.java' file open in the editor. The code implements a controller for managing recharges. It includes methods for deleting a recharge, counting all recharges, and finding recharges by telephone number.

```
38     repository.delete(r);
39   }
40
41= @GetMapping(value = "/count")
42  public long countRecharge() {
43    return repository.count();
44  }
45
46= @GetMapping(value = "/all/{numTel}")
47  public Collection<?> findRechargeByTelNumber(@PathVariable(required = true) int numTel) {
48
49    return repository.findRechargeByTelNumber(numTel);
50  }
51 }
52
```

Figure 114: Get recharge by telephone number

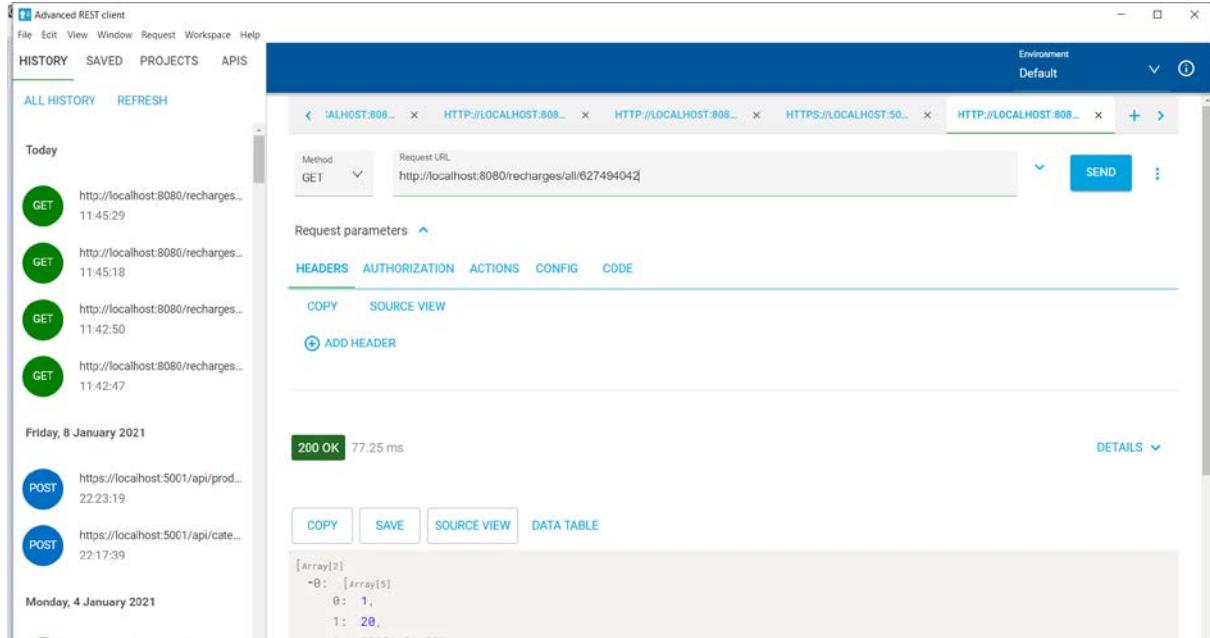


Figure 115: Get recharge by telephone number test

The screenshot shows the Advanced REST client interface. The top navigation bar includes File, Edit, View, Window, Request, Workspace, Help, HISTORY, SAVED, PROJECTS, APIS, Environment (Default), and a refresh button. The main area displays a list of API requests under 'HISTORY'. A specific request is selected, showing a green '200 OK' status box with '77.25 ms' response time. Below the status are buttons for COPY, SAVE, SOURCE VIEW, and DATA TABLE. The DATA TABLE tab is active, displaying the following JSON response:

```
[Array[2]
  ↪0: [Array[5]
    0: 1,
    1: 20,
    2: "2021-01-03",
    3: 627494042,
    4: "Minute"
  ],
  ↪1: [Array[5]
    0: 2,
    1: 50,
    2: "2020-12-09",
    3: 627494042,
    4: "Internet*3"
  ],
],
]
```

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/>]

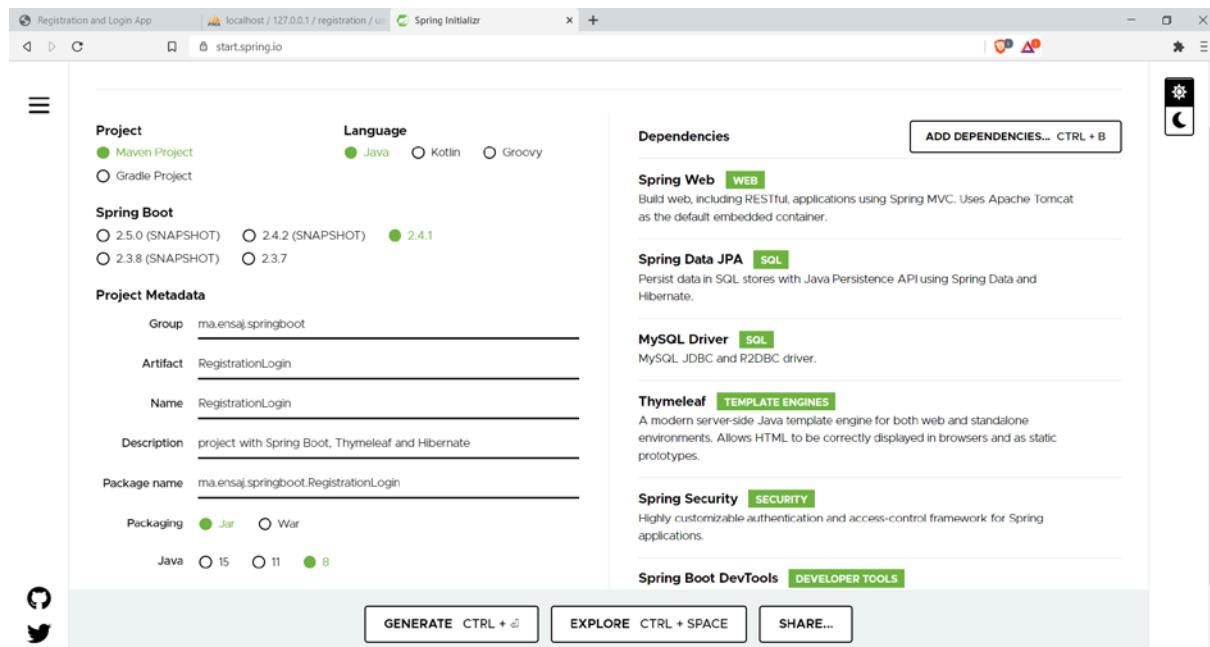


Figure 117:Spring Initializr

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

```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.18363.1256]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\hhass\Desktop\RegistrationLogin\RegistrationLogin>mvn eclipse:eclipse
```

A screenshot of a Windows Command Prompt window. The title bar says "C:\Windows\System32\cmd.exe". The window displays the command "mvn eclipse:eclipse" being typed at the prompt. The command has been partially entered, with the cursor at the end of "eclipse:eclipse".

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

4. Import it under Eclipse.

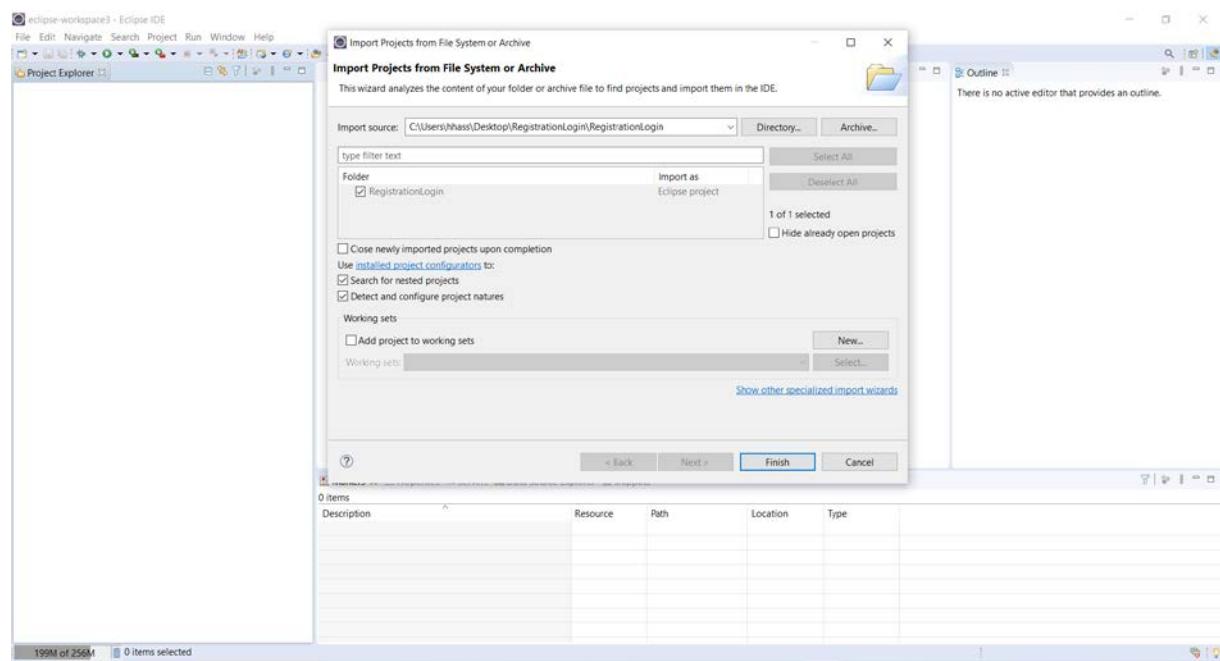


Figure 120:Import it under Eclipse.

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).

```

1 package ma.ensaj.springboot.RegistrationLogin;
2
3 import org.springframework.boot.SpringApplication;
4
5 @SpringBootApplication
6 public class RegistrationLoginApplication {
7
8     public static void main(String[] args) {
9         SpringApplication.run(RegistrationLoginApplication.class, args);
10    }
11 }
12
13 }
14

```

Figure 121:@SpringBootApplication

6. The content of pom.xml :

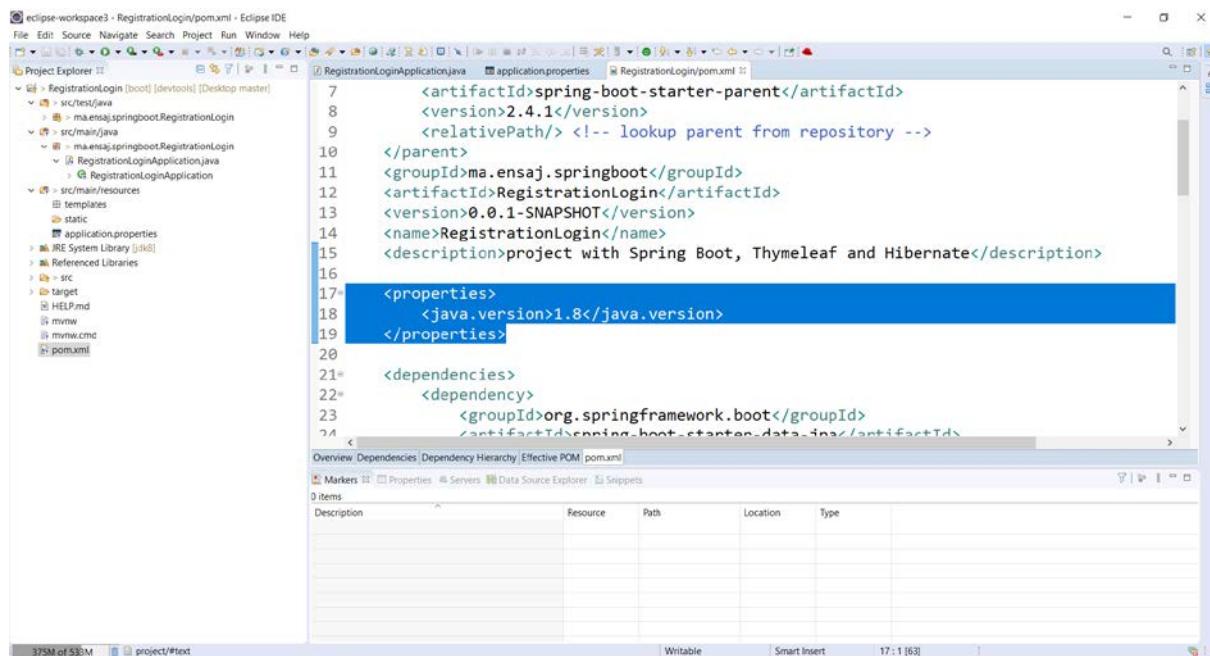
The screenshot shows the Eclipse IDE interface with the 'pom.xml' file open. The 'Dependencies' tab is selected, displaying a list of managed dependencies:

- spring-boot-starter-data-jpa (managed 2.3.0.RELEASE)
- spring-boot-starter-security (managed 2.3.0.RELEASE)
- spring-boot-starter-thymeleaf (managed 2.3.0.RELEASE)
- spring-boot-starter-web (managed 2.3.0.RELEASE)
- thymeleaf-extras-springsecurity5 (managed 3.0.4.RELEASE)
- spring-boot-devtools (runtime) (managed 2.3.0.RELEASE)
- mysql-connector-java [runtime] (managed 8.0.20)
- spring-boot-starter-test [test] (managed 2.3.0.RELEASE)
- spring-security-test [test] (managed 5.3.2.RELEASE)

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:



```

eclipse-workspace3 - RegistrationLogin/pom.xml - Eclipse IDE
File Edit Source Navigate Search Project Run Window Help
Project Explorer [1] RegistrationLogin [boot] [devtools] [Desktop master]
src/test/java ma.ensaj.springboot.RegistrationLogin
src/main/java ma.ensaj.springboot.RegistrationLogin
src/main/resources templates static application.properties
src/main/resources pom.xml
I RE System Library [jdk8]
Referenced Libraries
target HELD.md minw minw.cmd
pom.xml

RegistrationLoginApplication.java application.properties RegistrationLogin/pom.xml

7 <artifactId>spring-boot-starter-parent</artifactId>
8 <version>2.4.1</version>
9 <relativePath/> <!-- lookup parent from repository -->
10 </parent>
11 <groupId>ma.ensaj.springboot</groupId>
12 <artifactId>RegistrationLogin</artifactId>
13 <version>0.0.1-SNAPSHOT</version>
14 <name>RegistrationLogin</name>
15 <description>project with Spring Boot, Thymeleaf and Hibernate</description>
16
17 <properties>
18   <java.version>1.8</java.version>
19 </properties>
20
21 <dependencies>
22   <dependency>
23     <groupId>org.springframework.boot</groupId>
24     <artifactId>spring-boot-starter-data-jpa</artifactId>

```

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** :

The screenshot shows the Eclipse IDE interface with the title bar "eclipse-workspace3 - RegistrationLogin/src/main/resources/application.properties - Eclipse IDE". The left side features the "Project Explorer" view, which lists the project structure: RegistrationLogin [boot] [devtools] [Desktop master], src/test/java, src/main/java, src/main/resources, JRE System Library [jdk8], Referenced Libraries, src, target, mvnw, mvnw.cmd, and pom.xml. The right side is the "application.properties" file editor. The code in the editor is:

```
## Spring DATASOURCE (DataSourceAutoConfiguration & DataSourceProperties)
spring.datasource.url = jdbc:mysql://localhost:3306/registration?useSSL=false
spring.datasource.username = root
spring.datasource.password =
5
## Hibernate Properties
# The SQL dialect makes Hibernate generate better SQL for the chosen database
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL5InnoDBDialect
9
10# Hibernate ddl auto (create, create-drop, validate, update)
11spring.jpa.hibernate.ddl-auto = update
12
13logging.level.org.hibernate.SQL=DEBUG
14logging.level.org.hibernate.type=TRACE
15
16
17
```

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

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

Database	Collation	Action
bank	utf8mb4_general_ci	Check privileges
bdd	utf8mb4_general_ci	Check privileges
boot	utf8mb4_general_ci	Check privileges
cat_prod	utf8mb4_general_ci	Check privileges
credits	utf8mb4_general_ci	Check privileges
demo1	utf8mb4_general_ci	Check privileges
demomicroservice	utf8mb4_general_ci	Check privileges
ensaj	utf8mb4_general_ci	Check privileges
information_schema	utf8_general_ci	Check privileges
localisation	utf8mb4_general_ci	Check privileges
mysql	utf8mb4_general_ci	Check privileges
performance_schema	utf8_general_ci	Check privileges
phpmyadmin	utf8_bin	Check privileges
recharge	utf8mb4_general_ci	Check privileges
registration	utf8mb4_general_ci	Check privileges
test	latin1_swedish_ci	Check privileges
Console gboot	utf8mb4_general_ci	Check privileges

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`):

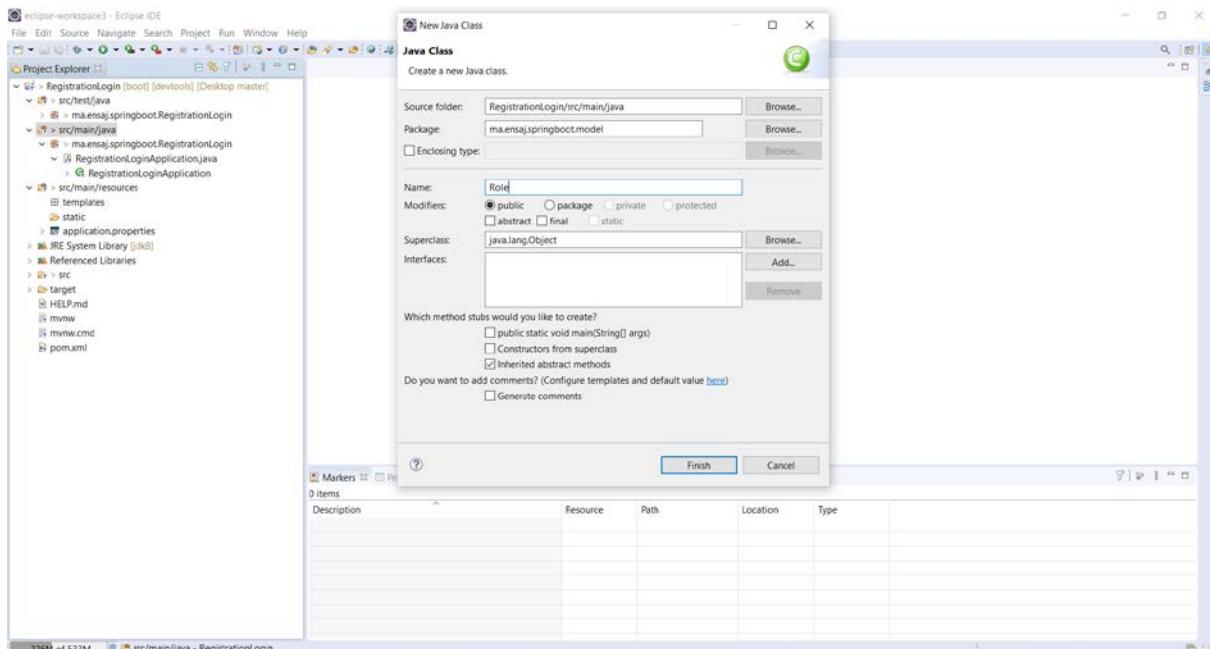


Figure 126:Create the "Role" class

12. Role class

The screenshot shows the Eclipse IDE interface with the 'Role.java' file open in the editor. The code defines a single class 'Role' with no attributes or methods.

```
1 package ma.ensaj.springboot.model;
2
3 public class Role {
4
5 }
```

The Project Explorer view on the left shows the project structure, including 'src/main/java' and 'src/main/resources' directories. The 'Markers' view at the bottom is empty.

Figure 127:Role class

13. Attributes of Role class

The screenshot shows the Eclipse IDE interface with the 'Role.java' file open in the editor. The code now includes two private attributes: 'id' and 'name'.

```
1 package ma.ensaj.springboot.model;
2
3 public class Role {
4     private Long id;
5     private String name;
6
7 }
```

The Project Explorer view on the left shows the project structure. The 'Markers' view at the bottom is empty.

Figure 128:Attributes of Role class

14. Generate constructors following our needs and wants

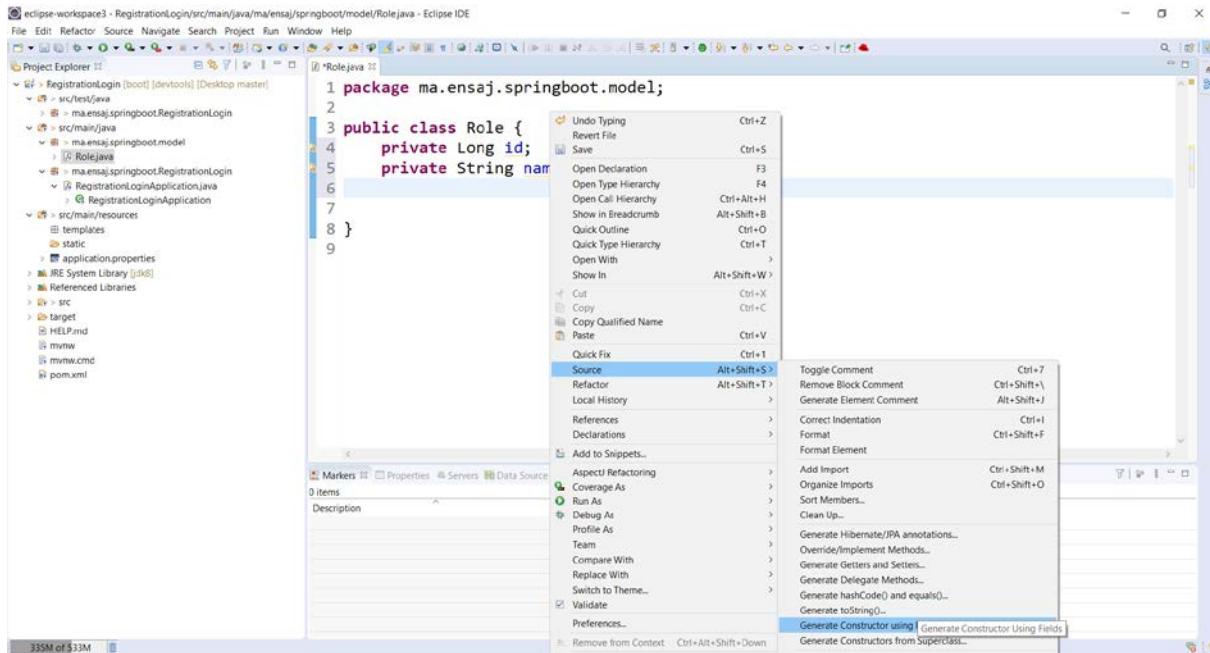


Figure 129:Generate constructors following our needs and wants

15. Select name field

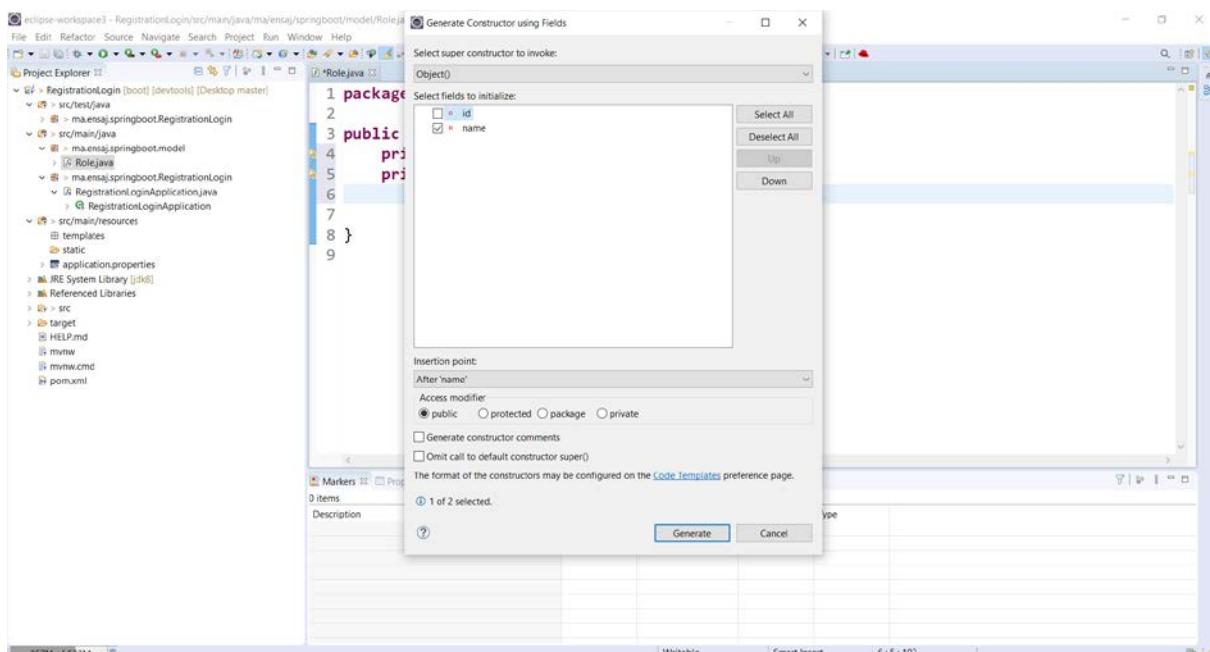


Figure 130:Select name field

16. Generate getters and setters

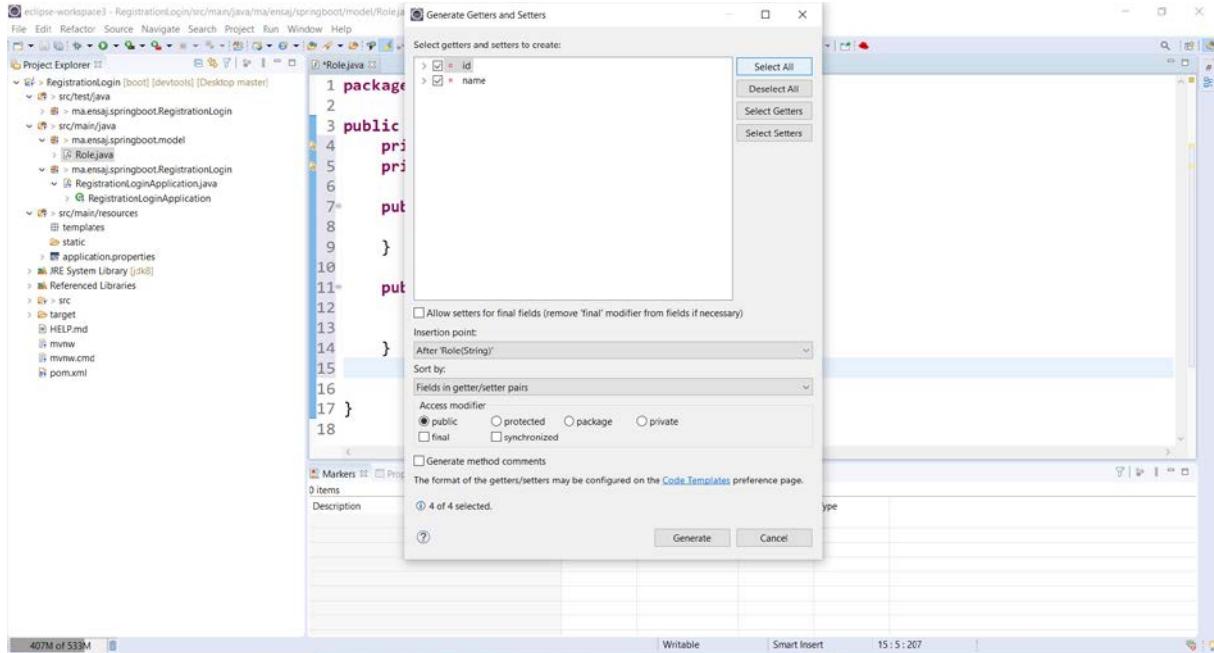


Figure 131:Generate getters and setters

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

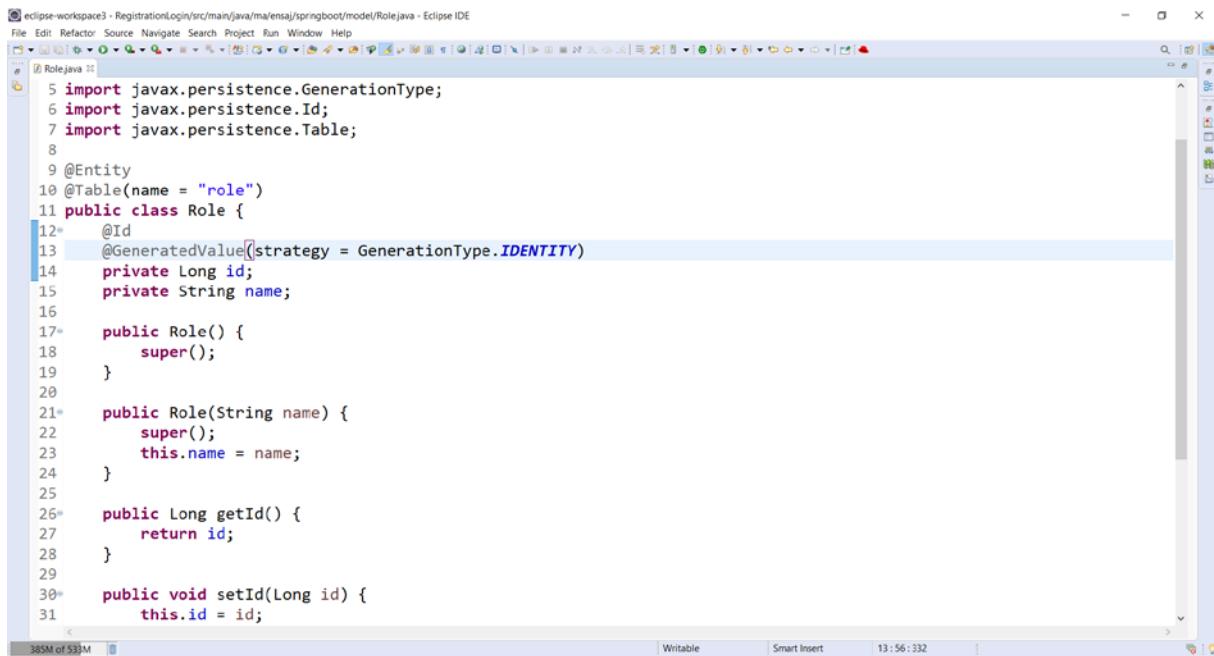


Figure 132:Add some annotations:@Entity,@Table,@Id,@GeneratedValue

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

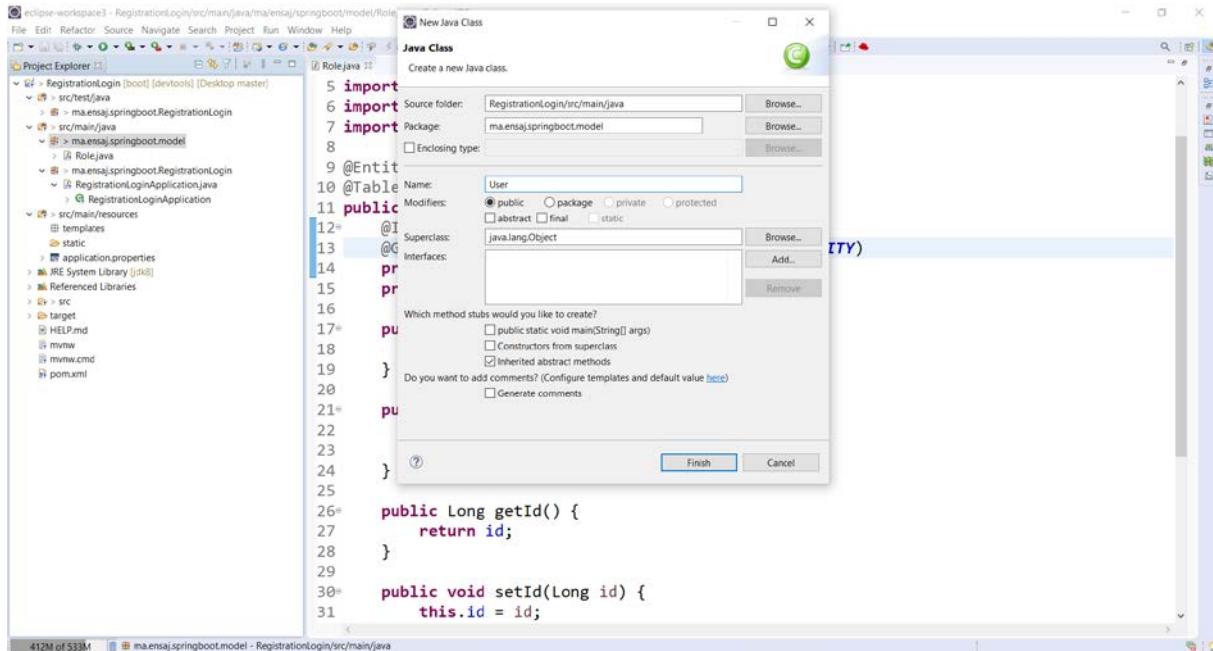


Figure 133:Create the "User" class

19. User's class attributes

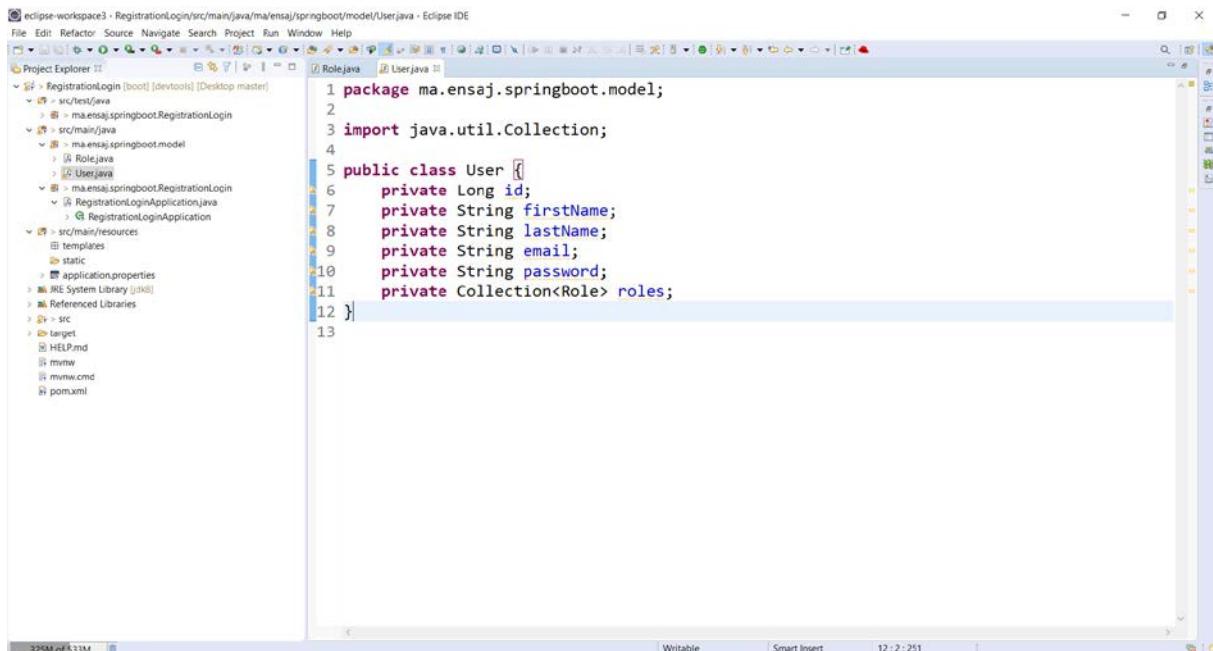


Figure 134:User's class attributes

20. Generate constructors following our needs and wants

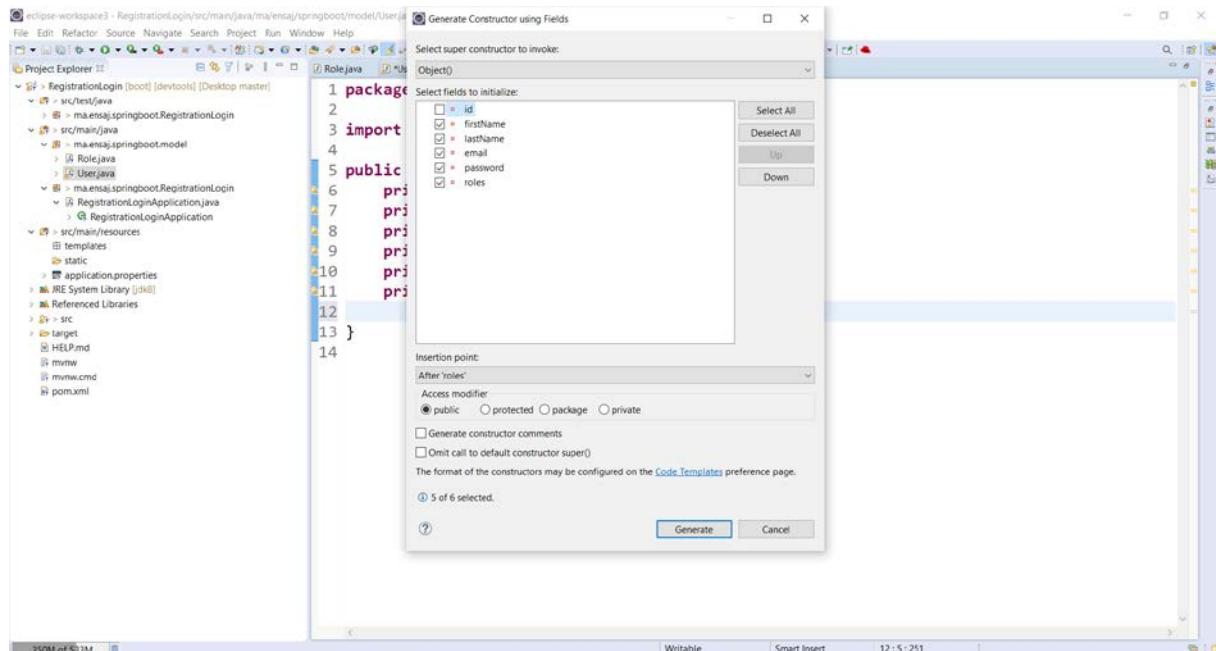


Figure 135:Generate constructors following our needs and wants

21. Generate getters and setters

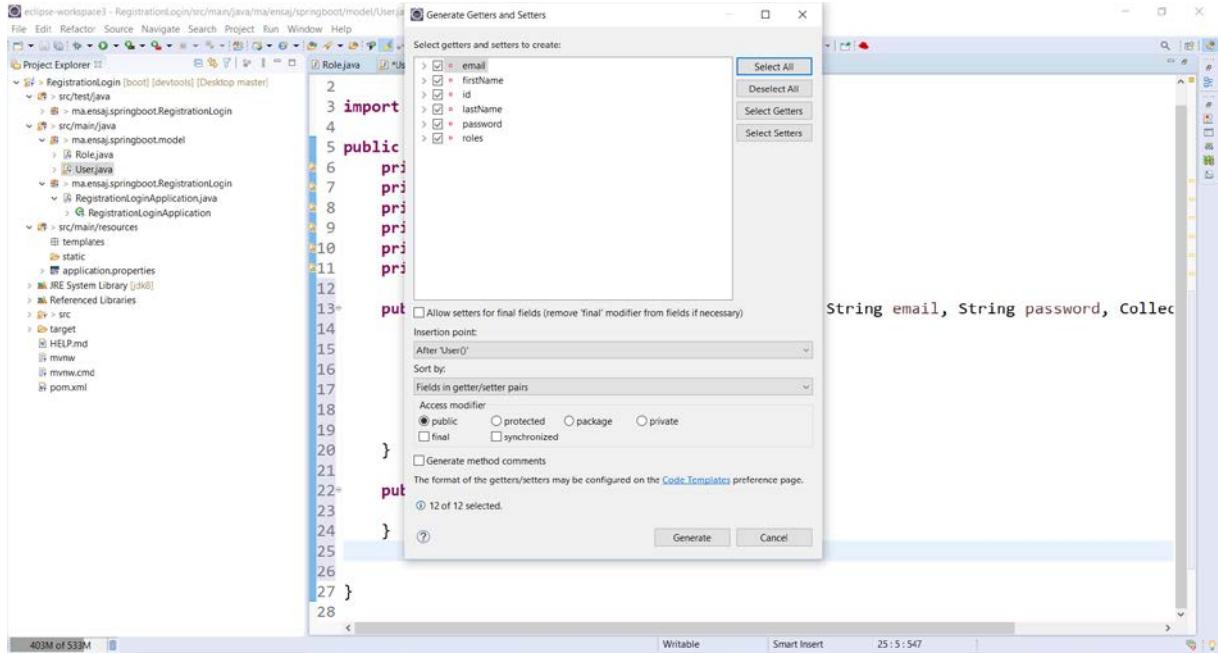
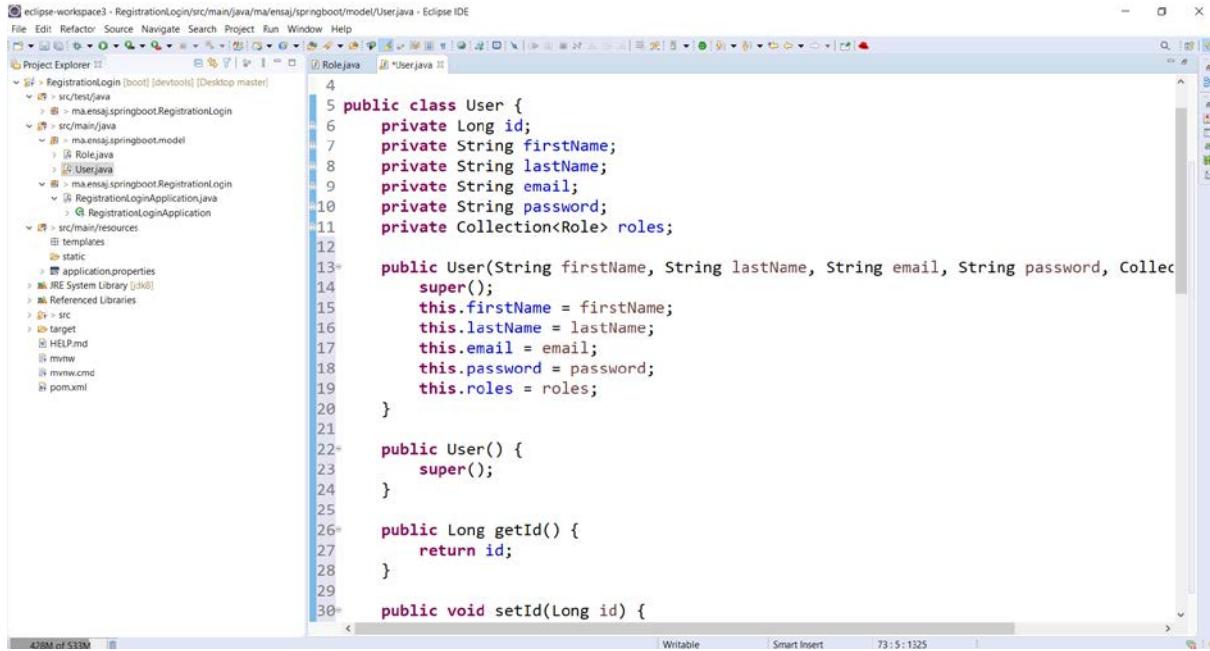


Figure 136:Generate getters and setters

22. User class

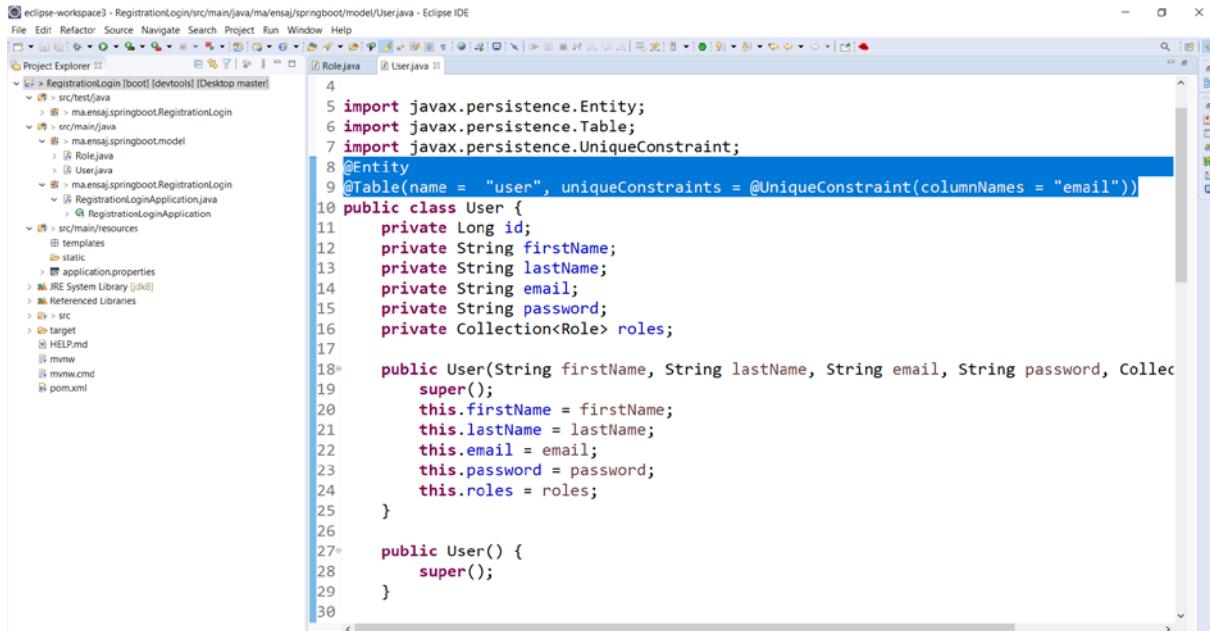


The screenshot shows the Eclipse IDE interface with the User.java file open in the editor. The code defines a User class with fields for id, firstName, lastName, email, password, and roles. It includes constructors, getters, and setters.

```
4 public class User {
5     private Long id;
6     private String firstName;
7     private String lastName;
8     private String email;
9     private String password;
10    private Collection<Role> roles;
11
12
13    public User(String firstName, String lastName, String email, String password, Collection<Role> roles) {
14        super();
15        this.firstName = firstName;
16        this.lastName = lastName;
17        this.email = email;
18        this.password = password;
19        this.roles = roles;
20    }
21
22    public User() {
23        super();
24    }
25
26    public Long getId() {
27        return id;
28    }
29
30    public void setId(Long id) {
```

Figure 137:User class

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

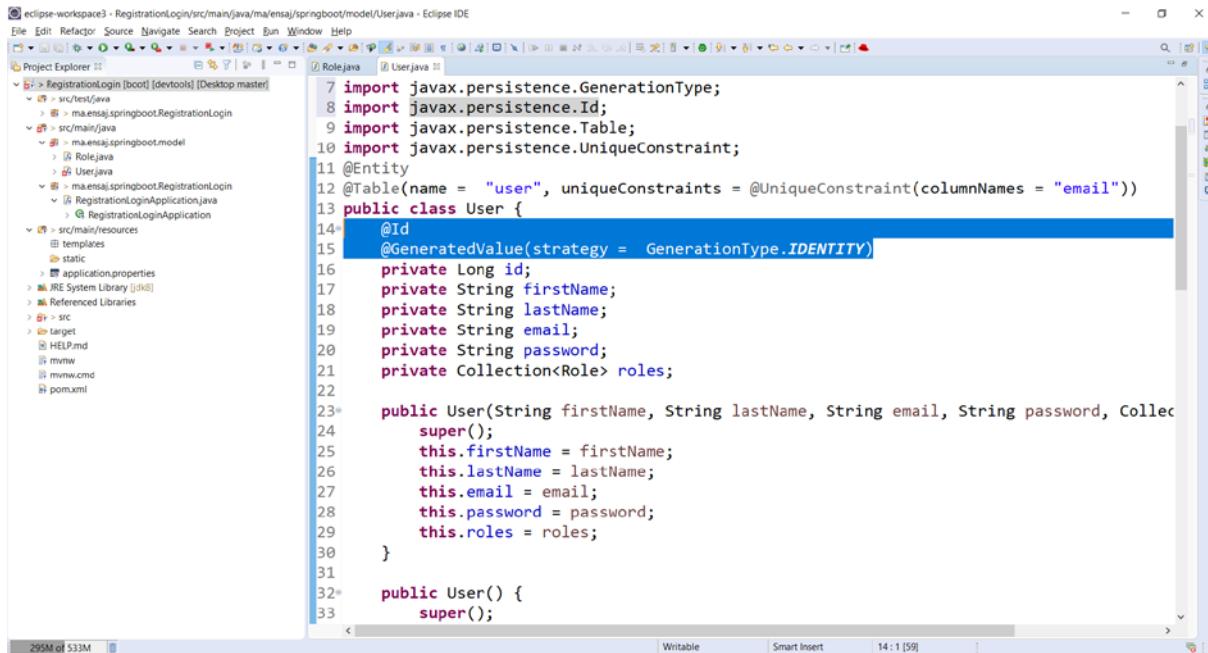


The screenshot shows the Eclipse IDE interface with the User.java file open in the editor. Annotations have been added to the User class: @Entity, @Table(name = "user", uniqueConstraints = @UniqueConstraint(columnNames = "email")), and @UniqueConstraint(columnNames = "email").

```
4 import javax.persistence.Entity;
5 import javax.persistence.Table;
6 import javax.persistence.UniqueConstraint;
8 @Entity
9 @Table(name = "user", uniqueConstraints = @UniqueConstraint(columnNames = "email"))
10 public class User {
11     private Long id;
12     private String firstName;
13     private String lastName;
14     private String email;
15     private String password;
16     private Collection<Role> roles;
17
18    public User(String firstName, String lastName, String email, String password, Collection<Role> roles) {
19        super();
20        this.firstName = firstName;
21        this.lastName = lastName;
22        this.email = email;
23        this.password = password;
24        this.roles = roles;
25    }
26
27    public User() {
28        super();
29    }
30}
```

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

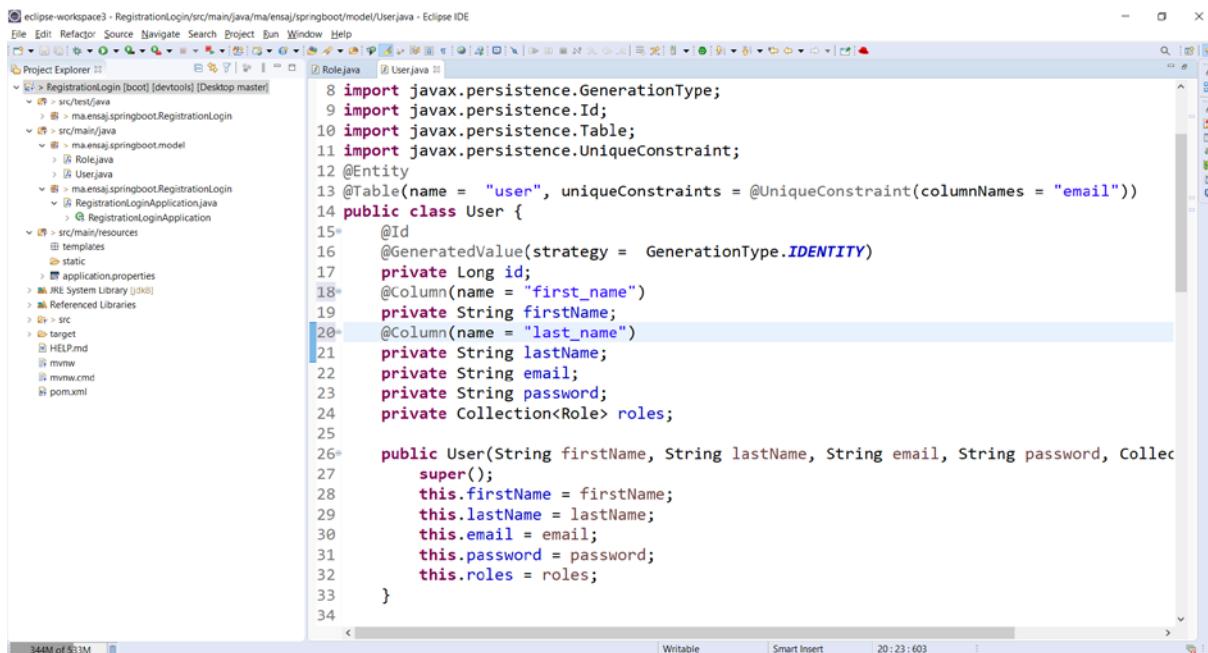
24. Adding @Id and @GeneratedValue annotations for Id field



```
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;
import javax.persistence.UniqueConstraint;
@Entity
@Table(name = "user", uniqueConstraints = @UniqueConstraint(columnNames = "email"))
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    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, Collection<Role> roles) {
        super();
        this.firstName = firstName;
        this.lastName = lastName;
        this.email = email;
        this.password = password;
        this.roles = roles;
    }
    public User() {
        super();
    }
}
```

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

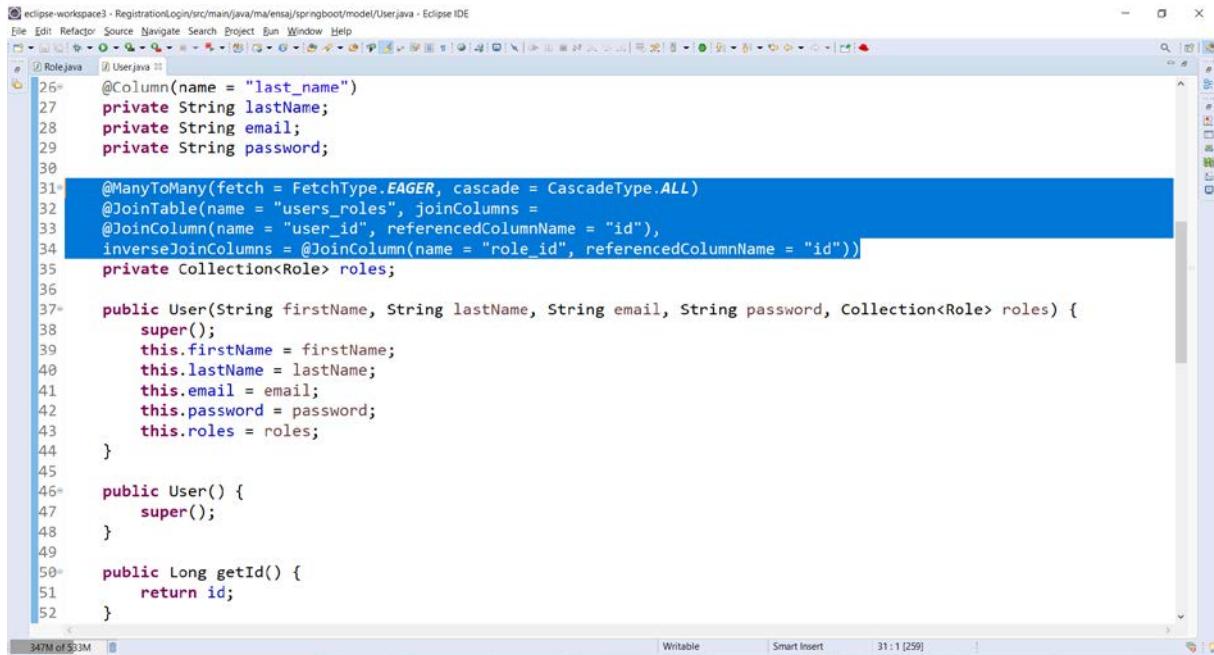
25. @Column for column names



```
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;
import javax.persistence.UniqueConstraint;
@Entity
@Table(name = "user", uniqueConstraints = @UniqueConstraint(columnNames = "email"))
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @Column(name = "first_name")
    private String firstName;
    @Column(name = "last_name")
    private String lastName;
    private String email;
    private String password;
    private Collection<Role> roles;
    public User(String firstName, String lastName, String email, String password, Collection<Role> roles) {
        super();
        this.firstName = firstName;
        this.lastName = lastName;
        this.email = email;
        this.password = password;
        this.roles = roles;
    }
    public User() {
        super();
    }
}
```

Figure 140:@Column for column names

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



```
26  @Column(name = "last_name")
27  private String lastName;
28  private String email;
29  private String password;
30
31  @ManyToMany(fetch = FetchType.EAGER, cascade = CascadeType.ALL)
32  @JoinTable(name = "users_roles", joinColumns =
33  @JoinColumn(name = "user_id", referencedColumnName = "id"),
34  inverseJoinColumns = @JoinColumn(name = "role_id", referencedColumnName = "id"))
35  private Collection<Role> roles;
36
37  public User(String firstName, String lastName, String email, String password, Collection<Role> roles) {
38      super();
39      this.firstName = firstName;
40      this.lastName = lastName;
41      this.email = email;
42      this.password = password;
43      this.roles = roles;
44  }
45
46  public User() {
47      super();
48  }
49
50  public Long getId() {
51      return id;
52  }
```

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

```
Select C:\Windows\System32\cmd.exe - mvn spring-boot:run
[...]
2021-01-12 18:50:52.656 INFO 11212 --- [           task-1] com.zaxxer.hikari.HikariDataSource      : HikariPool-1 - Start completed.
2021-01-12 18:50:52.680 INFO 11212 --- [restartedMain] o.s.b.d.a.OptionalLiveReloadServer    : LiveReload server is running on port 35729
2021-01-12 18:50:52.683 INFO 11212 --- [           task-1] org.hibernate.dialect.Dialect        : HHH000400: Using dialect: org.hibernate.dialect.MySQL5InnoDBDialect
2021-01-12 18:50:52.747 INFO 11212 --- [restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context path ''
2021-01-12 18:50:52.752 INFO 11212 --- [restartedMain] DeferredRepositoryInitializationListener : Triggering deferred initialization of Spring Data repositories
2021-01-12 18:50:52.753 INFO 11212 --- [restartedMain] DeferredRepositoryInitializationListener : Spring Data repositories initialized!
2021-01-12 18:50:52.787 INFO 11212 --- [restartedMain] m.e.s.R.RegistrationLoginApplication : Started RegistrationLoginApplication in 5.584 seconds (JVM running for 6.234)
2021-01-12 18:50:52.808 DEBUG 11212 --- [           task-1] o.h.type.spi.TypeConfiguration$Scope   : Scoping TypeConfiguration [org.hibernate.type.spi.TypeConfiguration@5c74befb] to MetadataBuildingContext [org.hibernate.boot.internal.MetadataBuildingContextRootImpl@41727666]
2021-01-12 18:50:52.961 DEBUG 11212 --- [           task-1] o.h.type.spi.TypeConfiguration$Scope   : Scoping TypeConfiguration [org.hibernate.type.spi.TypeConfiguration@5c74befb] to SessionFactoryImpl [org.hibernate.internal.SessionFactoryImpl@7aae29a]
2021-01-12 18:50:53.023 INFO 11212 --- [           task-1] o.h.e.t.j.p.i.JtaPlatformInitiator    : HHH000490: Using JtaPlatform implementation: [org.hibernate.engine.transaction.jta.platform.internal.NoJtaPlatform]
2021-01-12 18:50:53.033 TRACE 11212 --- [           task-1] o.h.type.spi.TypeConfiguration$Scope   : Handling #sessionFactoryCreated from [org.hibernate.internal.SessionFactoryImpl@7aae29a] for TypeConfiguration
2021-01-12 18:50:53.037 INFO 11212 --- [           task-1] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
```

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

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

Table	Action	Rows	Type	Collation	Size	Overhead
role		3	InnoDB	utf8mb4_general_ci	16.0 Kib	
user		3	InnoDB	utf8mb4_general_ci	32.0 Kib	
users_roles		9	InnoDB	utf8mb4_general_ci	96.0 Kib	
3 tables	Sum					

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

30. The structure of User table

The screenshot shows the phpMyAdmin interface for the 'user' table in the 'registration' database. The left sidebar lists various databases and tables, with 'user' selected. The main panel displays the table structure with 5 columns: Id, email, first_name, last_name, and password. It also shows the primary key constraint and two indexes: 'PRIMARY' and 'UKob8kqqqgmefl0aco34akdtp'. Action buttons for each row and index are visible.

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	Id	bigint(20)	utf8mb4_general_ci		No	None		AUTO_INCREMENT	Change Drop More
2	email	varchar(255)	utf8mb4_general_ci		Yes	NULL			Change Drop More
3	first_name	varchar(255)	utf8mb4_general_ci		Yes	NULL			Change Drop More
4	last_name	varchar(255)	utf8mb4_general_ci		Yes	NULL			Change Drop More
5	password	varchar(255)	utf8mb4_general_ci		Yes	NULL			Change Drop More

Action	Keyname	Type	Unique	Packed	Column	Cardinality	Collation	Null	Comment
Edit Drop	PRIMARY	BTREE	Yes	No	id	0	A	No	
Edit Drop	UKob8kqqqgmefl0aco34akdtp	BTREE	Yes	No	email	0	A	Yes	

Figure 145:The structure of User table

31. The structure of Role table

The screenshot shows the phpMyAdmin interface for the 'role' table in the 'registration' database. The left sidebar lists various databases and tables, with 'role' selected. The main panel displays the table structure with 2 columns: id and name. It shows the primary key constraint and one index: 'PRIMARY'. Action buttons for each row and index are visible.

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	id	bigint(20)	utf8mb4_general_ci		No	None		AUTO_INCREMENT	Change Drop More
2	name	varchar(255)	utf8mb4_general_ci		Yes	NULL			Change Drop More

Action	Keyname	Type	Unique	Packed	Column	Cardinality	Collation	Null	Comment
Edit Drop	PRIMARY	BTREE	Yes	No	id	0	A	No	

Figure 146:The structure of Role table

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

The screenshot shows the phpMyAdmin interface for a MySQL database named 'registration'. The left sidebar lists various databases, and the 'users_roles' table is selected under the 'registration' database. The main panel displays the structure of the 'users_roles' table, which has two columns: 'user_id' and 'role_id'. A SQL query window shows the command 'SELECT * FROM `users_roles`'. Below the query results, there are options for 'Query results operations' like 'Create view' and 'Bookmark this SQL query'. The status bar at the bottom indicates 'MySQL returned an empty result set (i.e. zero rows). (Query took 0.0018 seconds.)'.

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:

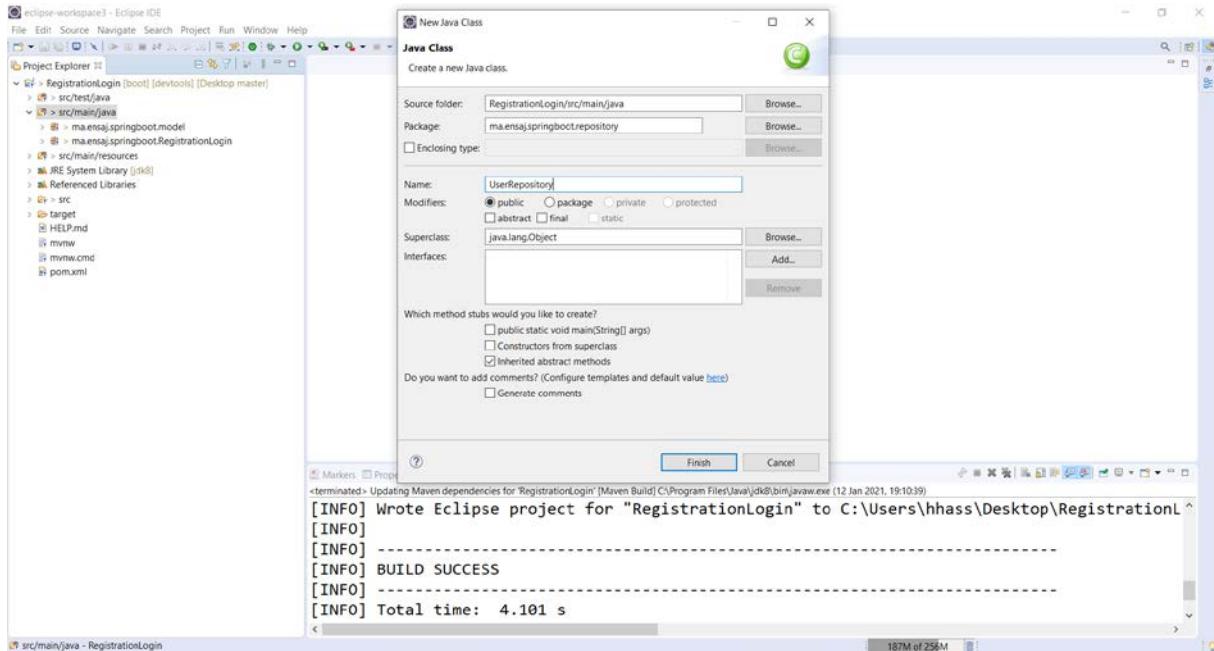


Figure 148:Create the "UserRepository" interface

35. UserRepository interface

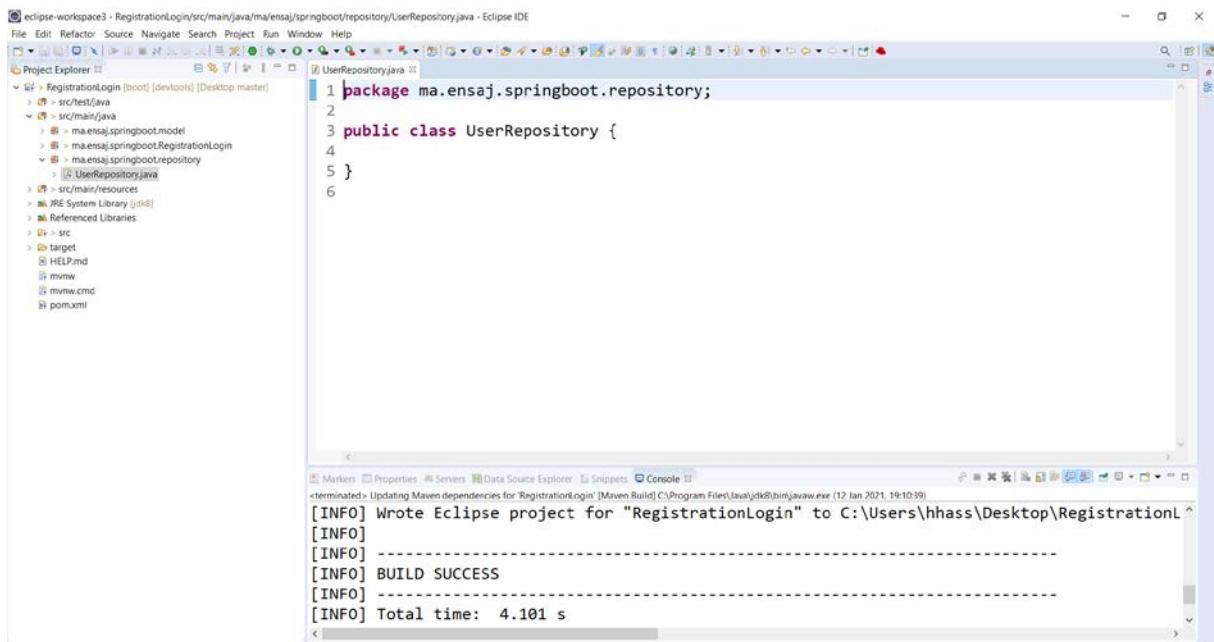
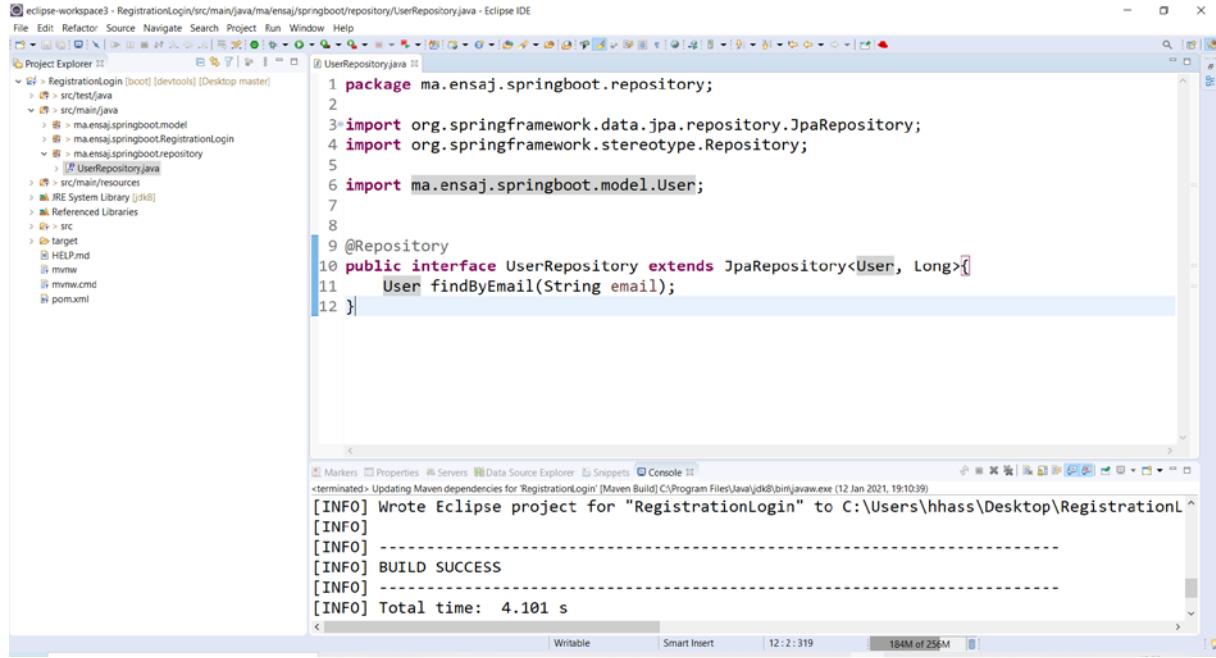


Figure 149: "UserRepository" interface

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



```

eclipse-workspace3 - RegistrationLogin/src/main/java/ma/ensaj/springboot/repository/UserRepository.java - Eclipse IDE
File Edit Refactor Source Navigate Search Project Run Window Help
Project Explorer [1] UserRepository.java [2]
1 package ma.ensaj.springboot.repository;
2
3 import org.springframework.data.jpa.repository.JpaRepository;
4 import org.springframework.stereotype.Repository;
5
6 import ma.ensaj.springboot.model.User;
7
8
9 @Repository
10 public interface UserRepository extends JpaRepository<User, Long>{
11     User findByEmail(String email);
12 }

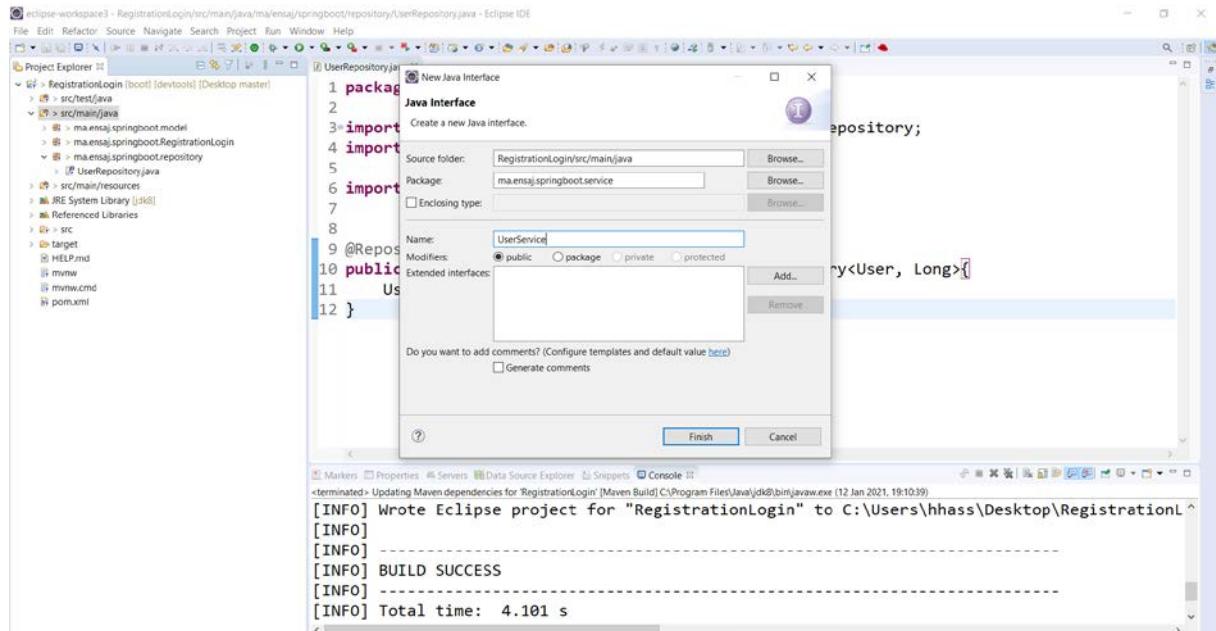
```

Markers Properties Servers Data Source Explorer Snippets Console

[terminated] Updating Maven dependencies for 'RegistrationLogin' [Maven Build] C:\Program Files\Java\jdk8\bin\javaw.exe (12 Jan 2021, 19:10:39)
[INFO] Wrote Eclipse project for "RegistrationLogin" to C:\Users\hhass\Desktop\RegistrationL^
[INFO]
[INFO]
[INFO] -----
[INFO] BUILD SUCCESS
[INFO]
[INFO] -----
[INFO] Total time: 4.101 s

Figure 150:the interface UserRepository extends the JpaRepository

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



New Java Interface

Create a new Java interface.

Source folder: RegistrationLogin/src/main/java

Package: ma.ensaj.springboot.service

Modifies: public package private protected

Name: UserService

Extended interfaces:

Add... Remove

Do you want to add comments? (Configure templates and default value [here](#)) Generate comments

Finish Cancel

Markers Properties Servers Data Source Explorer Snippets Console

[terminated] Updating Maven dependencies for 'RegistrationLogin' [Maven Build] C:\Program Files\Java\jdk8\bin\javaw.exe (12 Jan 2021, 19:10:39)
[INFO] Wrote Eclipse project for "RegistrationLogin" to C:\Users\hhass\Desktop\RegistrationL^
[INFO]
[INFO]
[INFO] -----
[INFO] BUILD SUCCESS
[INFO]
[INFO] -----
[INFO] Total time: 4.101 s

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.

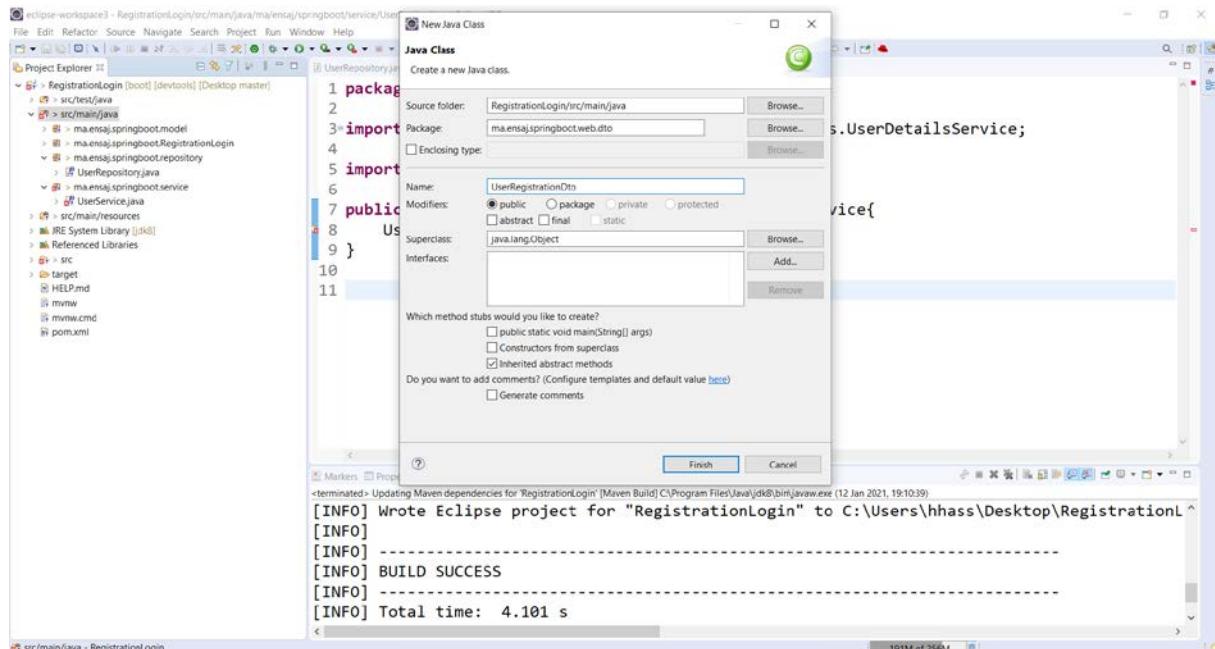


Figure 152:Creation of UserRegistrationDto class

39. UserRegistrationDto class attributes

```

eclipse-workspace3 - RegistrationLogin/src/main/java/ma/ensaj/springboot/web/dto/UserRegistrationDto.java - Eclipse IDE
File Edit Refactor Source Navigate Search Project Run Window Help
Project Explorer UserRepositoryJava UserServiceJava UserRegistrationDto.java
1 package ma.ensaj.springboot.web.dto;
2
3 public class UserRegistrationDto {
4     private String firstName;
5     private String lastName;
6     private String email;
7     private String password;
8
9
10 }
11

```

```

Markers Properties Servers Data Source Explorer Snippets Console
<terminated> Updating Maven dependencies for 'RegistrationLogin' [Maven Build] C:\Program Files\Java\jdk8\bin\javaw.exe (12 Jan 2021, 19:10:39)
[INFO] Wrote Eclipse project for "RegistrationLogin" to C:\Users\hhass\Desktop\RegistrationL^
[INFO]
[INFO] -----
[INFO] BUILD SUCCESS
[INFO]
[INFO] Total time:  4.101 s

```

Figure 153:UserRegistrationDto class attributes

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

```

eclipse-workspace3 - RegistrationLogin/src/main/java/ma/ensaj/springboot/web/dto/UserRegistrationDto.java - Eclipse IDE
File Edit Refactor Source Navigate Search Project Run Window Help
Project Explorer UserRepositoryJava UserServiceJava UserRegistrationDto.java
1 package ma.ensaj.springboot.web.dto;
2
3 public class UserRegistrationDto {
4     private String firstName;
5     private String lastName;
6     private String email;
7     private String password;
8
9     public UserRegistrationDto() {
10
11     }
12
13     public UserRegistrationDto(String firstName, String lastName, String email, String password) {
14         super();
15         this.firstName = firstName;
16         this.lastName = lastName;
17         this.email = email;
18         this.password = password;
19     }
20
21     public String getFirstName() {
22         return firstName;
23     }
24
25     public void setFirstName(String firstName) {
26         this.firstName = firstName;
27     }
28
29     public String getLastName() {
30         return lastName;
31     }
32
33     public void setLastName(String lastName) {
34         this.lastName = lastName;
35     }
36
37     public String getEmail() {
38         return email;
39     }
40
41     public void setEmail(String email) {
42         this.email = email;
43     }
44
45     public String getPassword() {
46         return password;
47     }
48
49     public void setPassword(String password) {
50         this.password = password;
51     }
52 }

```

```

Markers Properties Servers Data Source Explorer Snippets Console
<terminated> Updating Maven dependencies for 'RegistrationLogin' [Maven Build] C:\Program Files\Java\jdk8\bin\javaw.exe (12 Jan 2021, 19:10:39)
[INFO] Wrote Eclipse project for "RegistrationLogin" to C:\Users\hhass\Desktop\RegistrationL^
[INFO]
[INFO] -----
[INFO] BUILD SUCCESS
[INFO]
[INFO] Total time:  4.101 s

```

Figure 154:UserRegistrationDto class after adding constructors

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

The screenshot shows the Eclipse IDE interface. The top bar has menu items: File, Edit, Refactor, Source, Navigate, Search, Project, Run, Window, Help. Below the menu is the Project Explorer view, which lists the project structure: RegistrationLogin (boot) [devtools] [Desktop master]. It includes src/test/java, src/main/java (with subfolders like ma.ensaj.springboot.model, ma.ensaj.springboot.registrationLogin, ma.ensaj.springboot.repository, ma.ensaj.springboot.service, ma.ensaj.springboot.web.dto), src/main/resources, IRE System Library [jdk8], Referenced Libraries, src, target, and pom.xml. The main workspace shows three tabs: UserRepository.java, UserService.java (the current active tab), and UserRegistrationDto.java. The UserService.java code is as follows:

```
1 package ma.ensaj.springboot.service;
2
3 import org.springframework.security.core.userdetails.UserDetailsService;
4
5 import ma.ensaj.springboot.model.User;
6 import ma.ensaj.springboot.web.dto.UserRegistrationDto;
7
8 public interface UserService extends UserDetailsService{
9     User save(UserRegistrationDto registrationDto);
10 }
```

Below the workspace is the Console tab, which displays Maven build logs:

```
<terminated> Updating Maven dependencies for 'RegistrationLogin' [Maven Build] C:\Program Files\Java\jdk8\bin\javaw.exe (12 Jan 2021, 19:10:39)
[INFO] Wrote Eclipse project for "RegistrationLogin" to C:\Users\hhass\Desktop\RegistrationL^
[INFO]
[INFO]
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 4.101 s
```

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

42. Creation of **UserServiceImpl** class

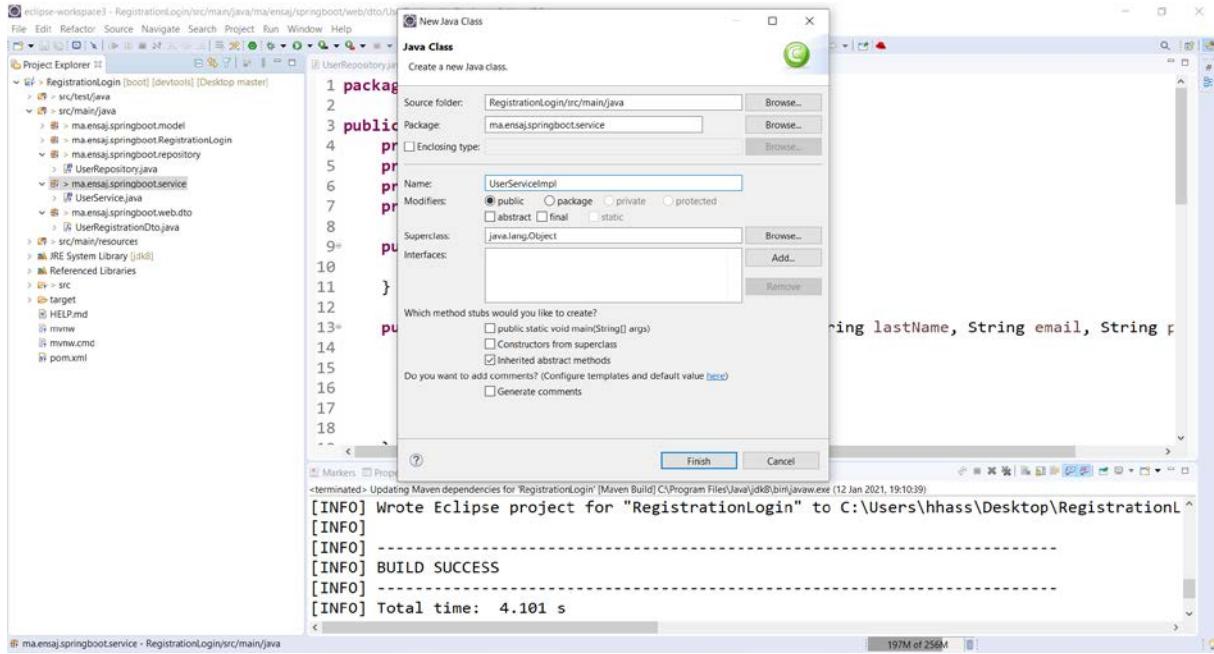


Figure 156:Creation of UserServiceImpl class

43. UserServiceImpl class is created

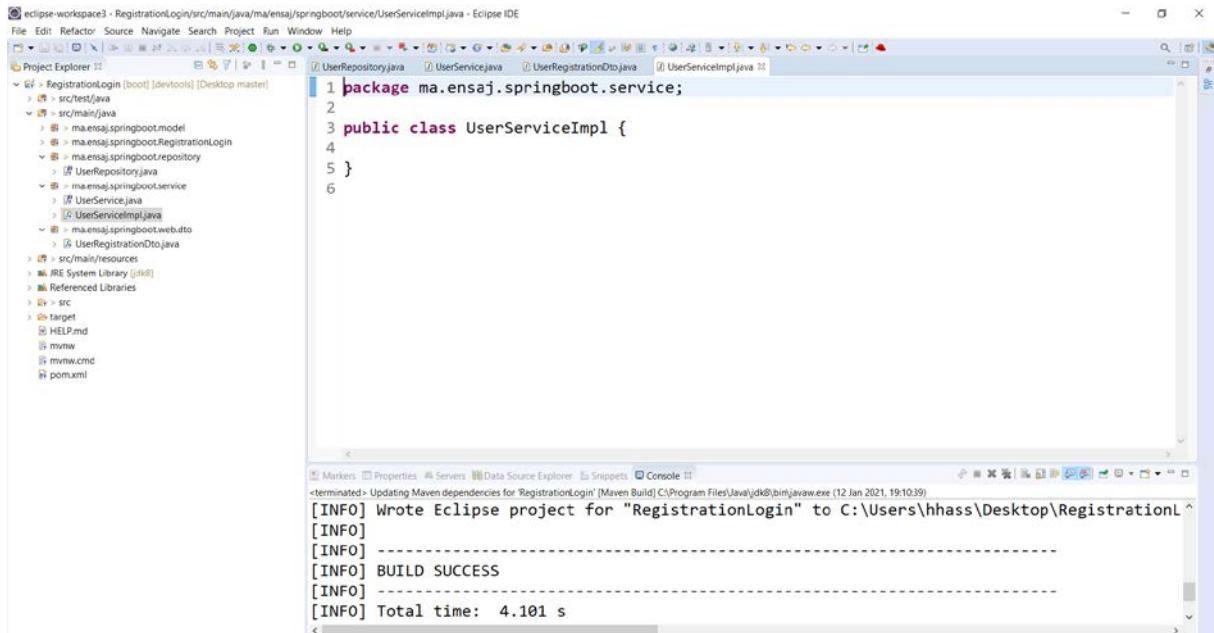
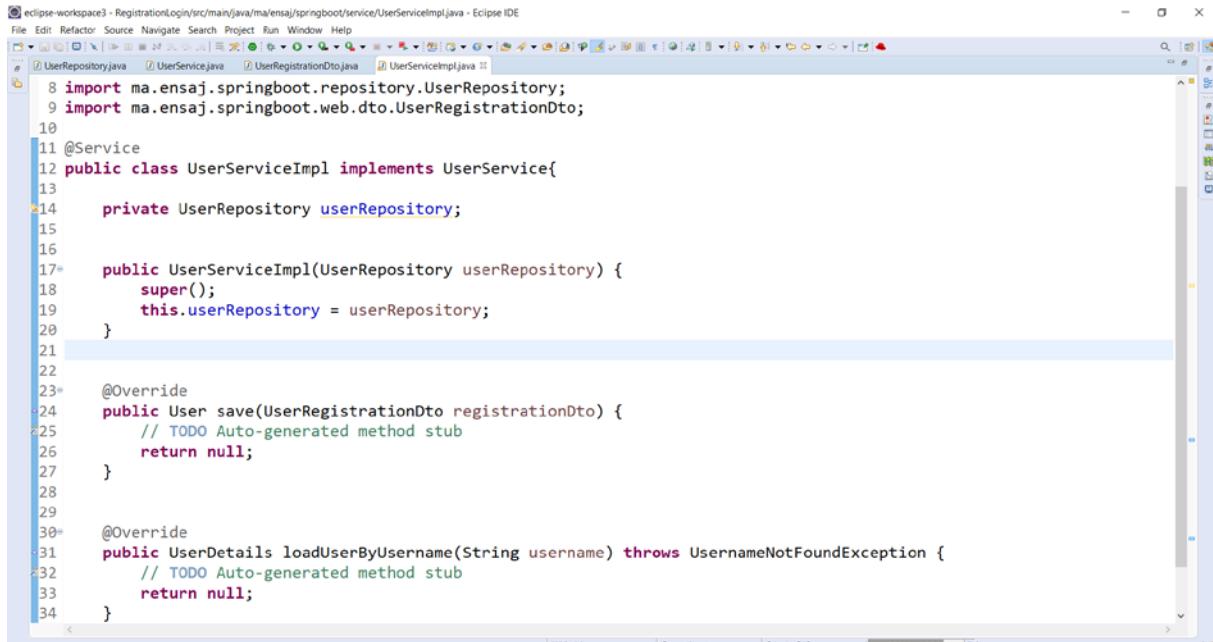


Figure 157:UserServiceImpl class is created

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



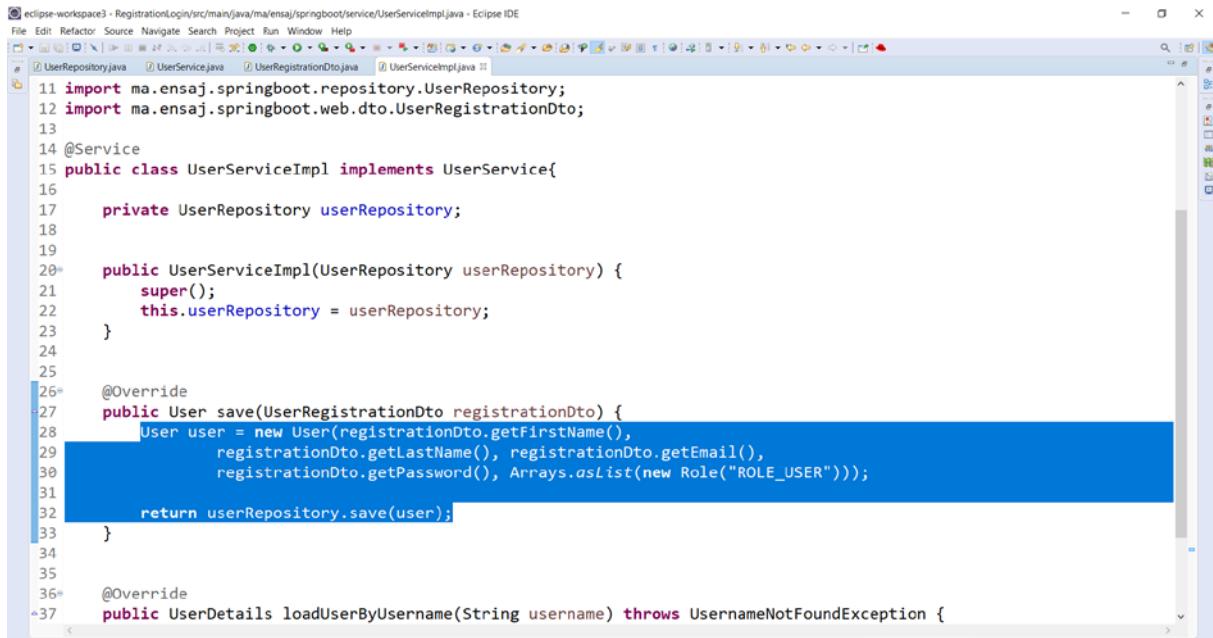
```

eclipse-workspace3 - RegistrationLogin/src/main/java/ma/ensaj/springboot/service/UserServiceImpl.java - Eclipse IDE
File Edit Refactor Source Navigate Search Project Run Window Help
UserRepository.java UserService.java UserRegistrationDto.java UserServiceImpl.java
8 import ma.ensaj.springboot.repository.UserRepository;
9 import ma.ensaj.springboot.web.dto.UserRegistrationDto;
10
11 @Service
12 public class UserServiceImpl implements UserService{
13
14     private UserRepository userRepository;
15
16
17     public UserServiceImpl(UserRepository userRepository) {
18         super();
19         this.userRepository = userRepository;
20     }
21
22
23     @Override
24     public User save(UserRegistrationDto registrationDto) {
25         // TODO Auto-generated method stub
26         return null;
27     }
28
29
30     @Override
31     public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
32         // TODO Auto-generated method stub
33         return null;
34     }

```

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

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



```

eclipse-workspace3 - RegistrationLogin/src/main/java/ma/ensaj/springboot/service/UserServiceImpl.java - Eclipse IDE
File Edit Refactor Source Navigate Search Project Run Window Help
UserRepository.java UserService.java UserRegistrationDto.java UserServiceImpl.java
11 import ma.ensaj.springboot.repository.UserRepository;
12 import ma.ensaj.springboot.web.dto.UserRegistrationDto;
13
14 @Service
15 public class UserServiceImpl implements UserService{
16
17     private UserRepository userRepository;
18
19
20     public UserServiceImpl(UserRepository userRepository) {
21         super();
22         this.userRepository = userRepository;
23     }
24
25
26     @Override
27     public User save(UserRegistrationDto registrationDto) {
28         User user = new User(registrationDto.getFirstName(),
29             registrationDto.getLastName(), registrationDto.getEmail(),
30             registrationDto.getPassword(), Arrays.asList(new Role("ROLE_USER")));
31
32         return userRepository.save(user);
33     }
34
35
36     @Override
37     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

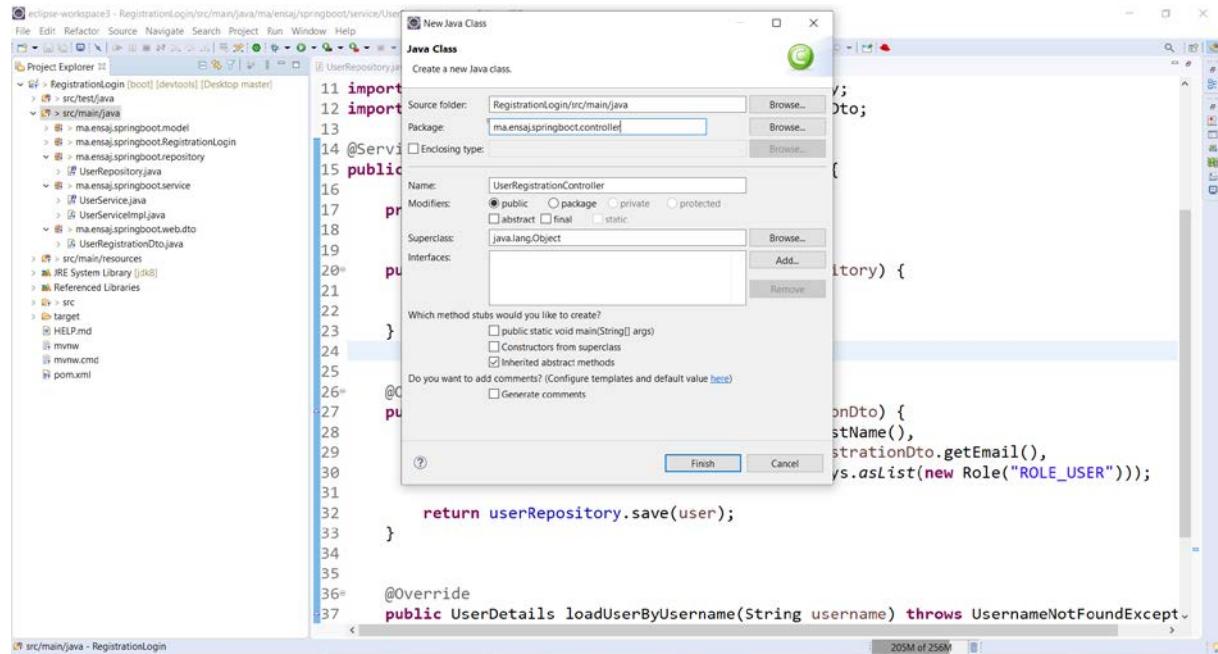


Figure 160:Creation of the class UserRegistrationController under the sub package controller of the main package

47. **UserRegistrationController** class

The screenshot shows the Eclipse IDE interface with the 'UserRegistrationController.java' file open in the editor. The code defines a single class:

```
1 package ma.ensaj.springboot.controller;
2
3 public class UserRegistrationController {
4
5 }
```

Figure 161:UserRegistrationController class

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

The screenshot shows the Eclipse IDE interface with the 'UserRegistrationController.java' file open in the editor. The code now includes annotations:

```
1 package ma.ensaj.springboot.controller;
2
3
4 *import org.springframework.stereotype.Controller;*
5
6 @Controller
7 @RequestMapping("/registration")
8 public class UserRegistrationController {
9
10     private UserService userService;
11
12     public UserRegistrationController(UserService userService) {
13         super();
14         this.userService = userService;
15     }
16
17     public String registerUserAccount(@ModelAttribute("user") UserRegistrationDto registrationDto) {
18         userService.save(registrationDto);
19         return "redirect:/registration?success";
20     }
21 }
```

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**

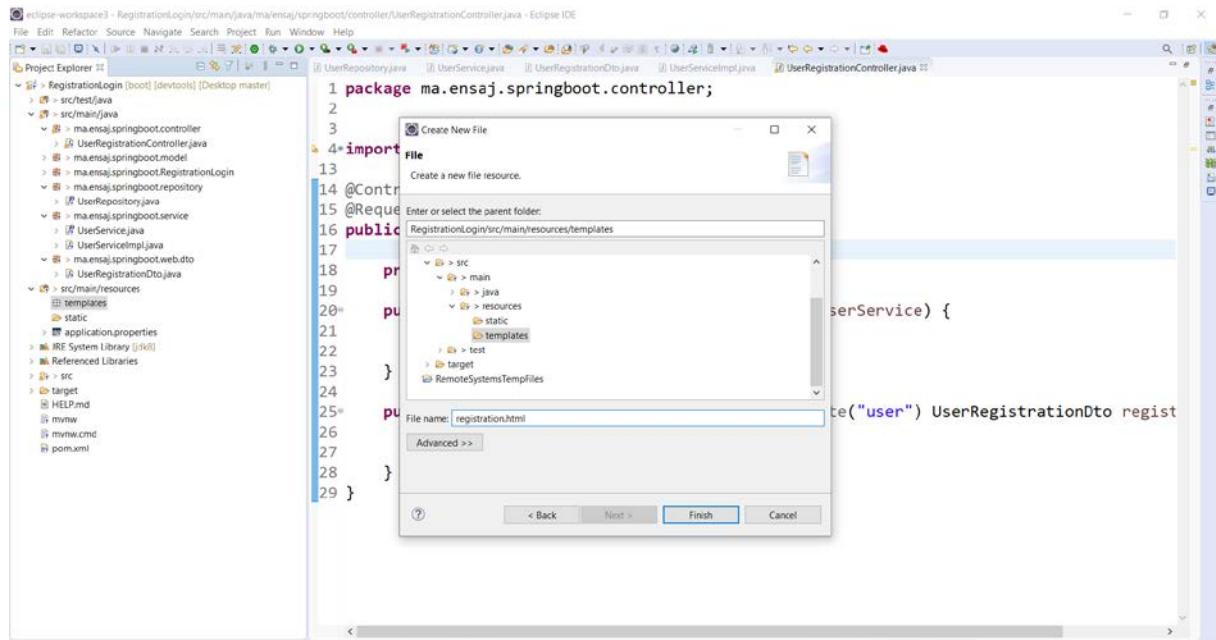


Figure 163:Create a new html file under the templates folder called registration

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

```

1 package ma.ensaj.springboot.controller;
2
3
4 import org.springframework.stereotype.Controller;
5
6 @Controller
7 @RequestMapping("/registration")
8 public class UserRegistrationController {
9
10     private UserService userService;
11
12     public UserRegistrationController(UserService userService) {
13         super();
14         this.userService = userService;
15     }
16
17     @GetMapping
18     public String showRegistrationForm() {
19         return "registration";
20     }
21
22     @PostMapping
23     public String registerUserAccount(@ModelAttribute("user") UserRegistrationDto registrationDto) {
24         userService.save(registrationDto);
25         return "redirect:/registration?success";
26     }
27
28 }
29
30
31
32
33

```

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

51. Adding the userRegistrationDto method with @ModelAttribute annotation

```

1 import org.springframework.web.bind.annotation.RequestMapping;
2 import org.springframework.web.bind.annotation.ModelAttribute;
3 import ma.ensaj.springboot.service.UserService;
4 import ma.ensaj.springboot.web.dto.UserRegistrationDto;
5
6 @SuppressWarnings("unused")
7 @Controller
8 @RequestMapping("/registration")
9 public class UserRegistrationController {
10
11     private UserService userService;
12
13     public UserRegistrationController(UserService userService) {
14         super();
15         this.userService = userService;
16     }
17
18     @ModelAttribute("user")
19     public UserRegistrationDto userRegistrationDto() {
20         return new UserRegistrationDto();
21     }
22
23     @GetMapping
24     public String showRegistrationForm() {
25         return "registration";
26     }
27
28     @PostMapping
29     public String registerUserAccount(@ModelAttribute("user") UserRegis...

```

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.

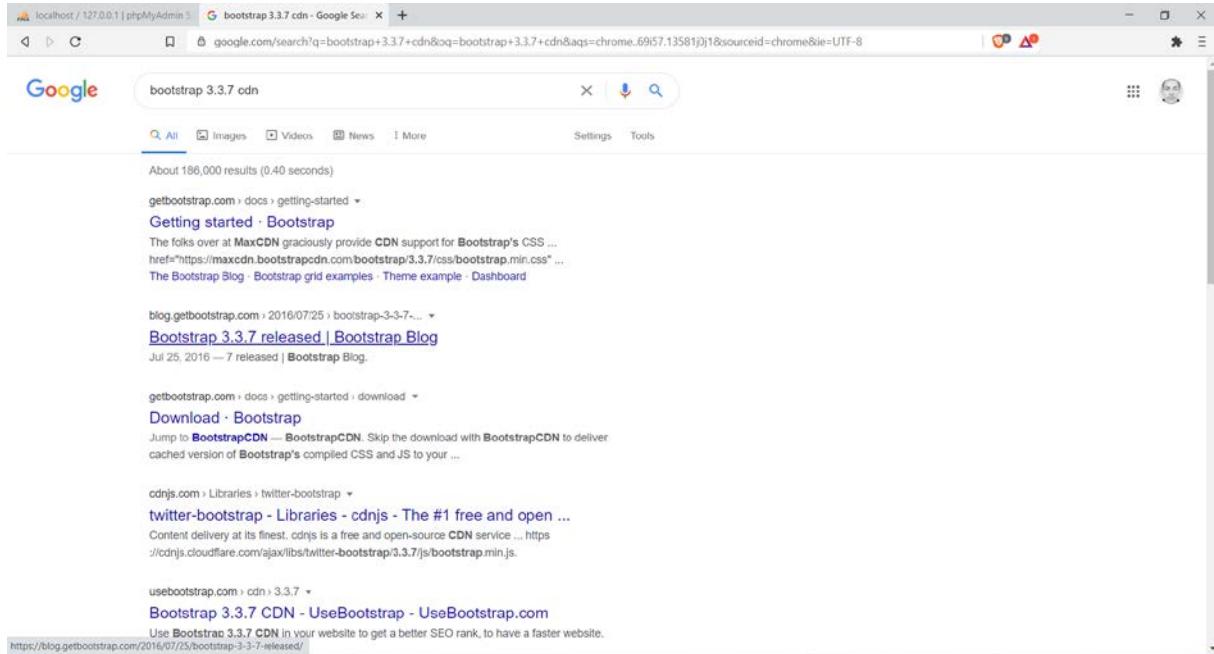


Figure 166:Searching for bootstrap 3.3.7 cdn link in Google.

53. Copy bootstrap 3.3.7 cdn's link

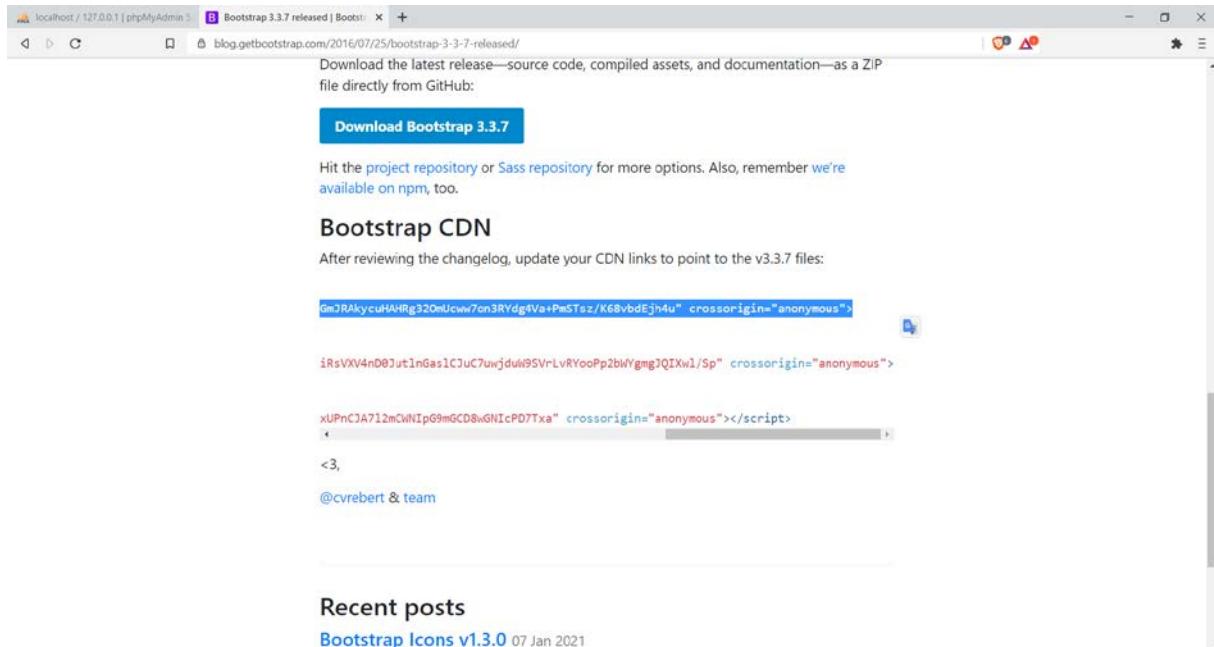
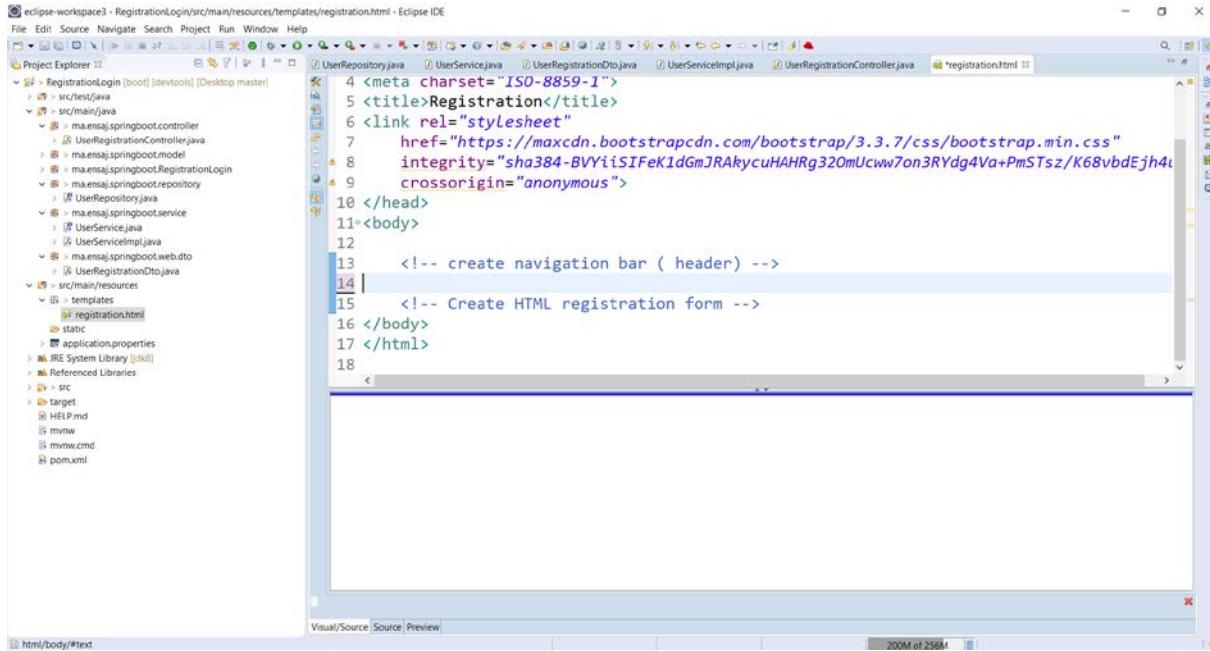


Figure 167:Copy bootstrap 3.3.7 cdn's link

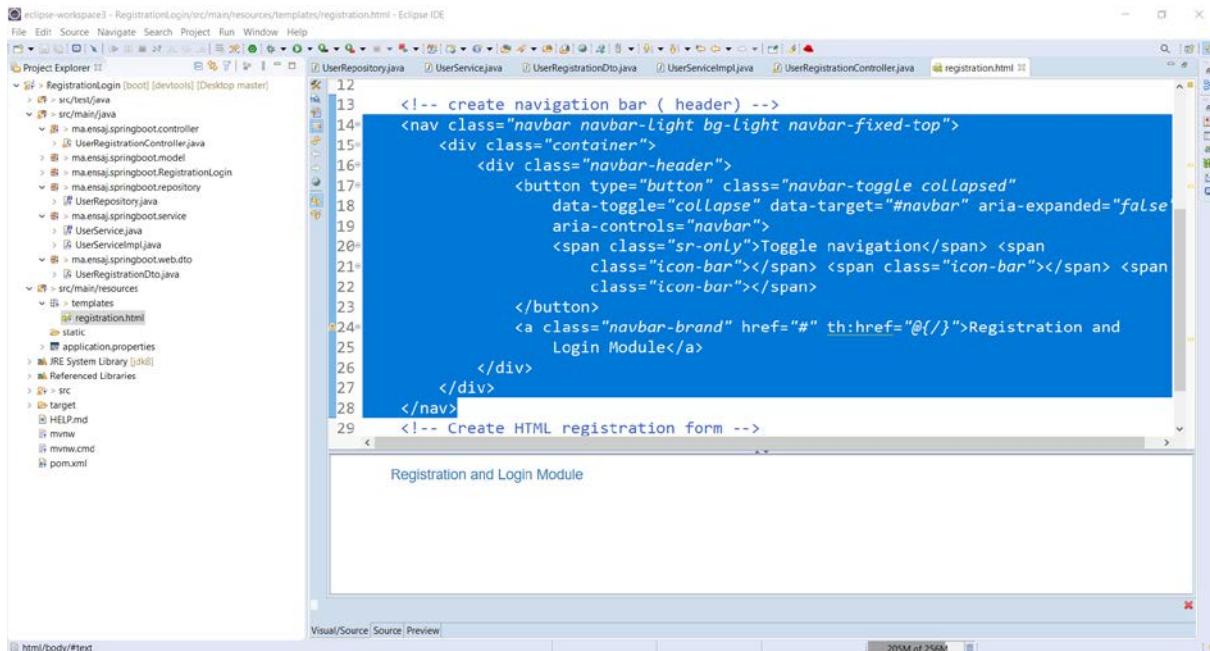
54. Paste bootstrap 3.3.7 cdn's link in the head section of our html page



```
4 <meta charset="ISO-8859-1">
5 <title>Registration</title>
6 <link rel="stylesheet"
7 href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
8 integrity="sha384-BVYiiSIFeK1dGmJRAkycuHAHRg32OmUcww7on3RYdg4Va+PmSTsz/K68vbdEjh4c
9 crossorigin="anonymous">
10 </head>
11 <body>
12
13     <!-- create navigation bar ( header ) -->
14
15     <!-- Create HTML registration form -->
16 </body>
17 </html>
18
```

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

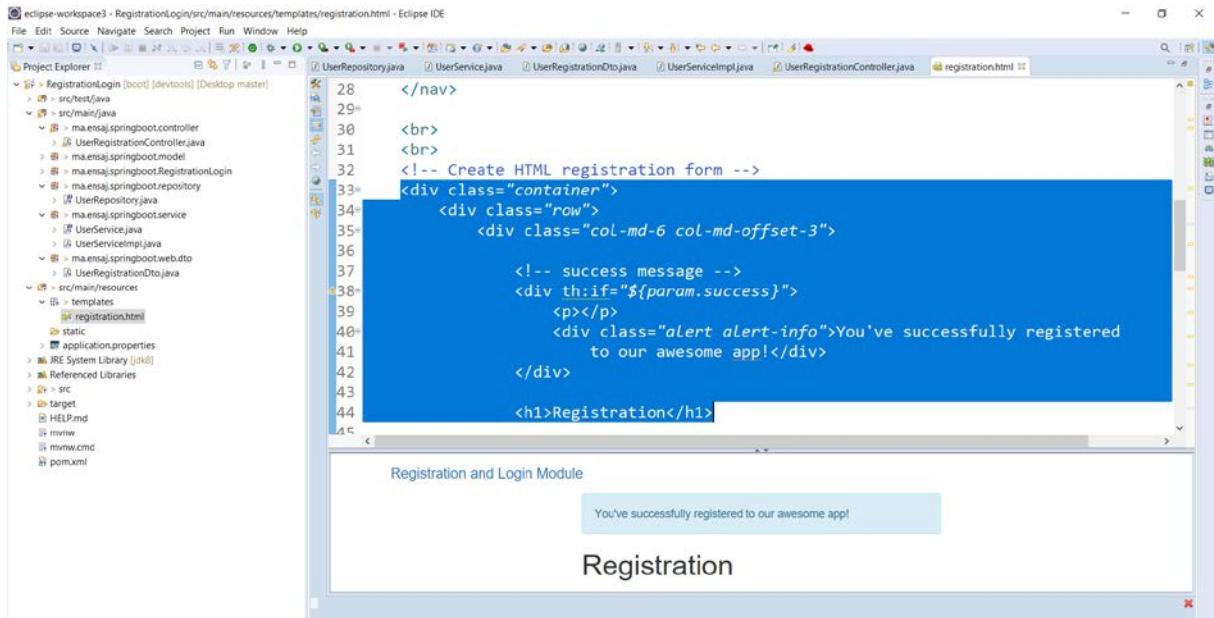
55. Copy paste nav section from the official documentation of bootstrap and personalize it following our needs and wants



```
12
13     <!-- create navigation bar ( header ) -->
14     <nav class="navbar navbar-light bg-light navbar-fixed-top">
15         <div class="container">
16             <div class="navbar-header">
17                 <button type="button" class="navbar-toggle collapsed"
18                     data-toggle="collapse" data-target="#navbar" aria-expanded="false"
19                     aria-controls="navbar">
20                     <span class="sr-only">Toggle navigation</span> <span
21                         class="icon-bar"></span> <span class="icon-bar"></span> <span
22                         class="icon-bar"></span>
23                 </button>
24                 <a class="navbar-brand" href="#" th:href="@{/}">Registration and
25                     Login Module</a>
26             </div>
27         </div>
28     </nav>
29     <!-- Create HTML registration form -->
```

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

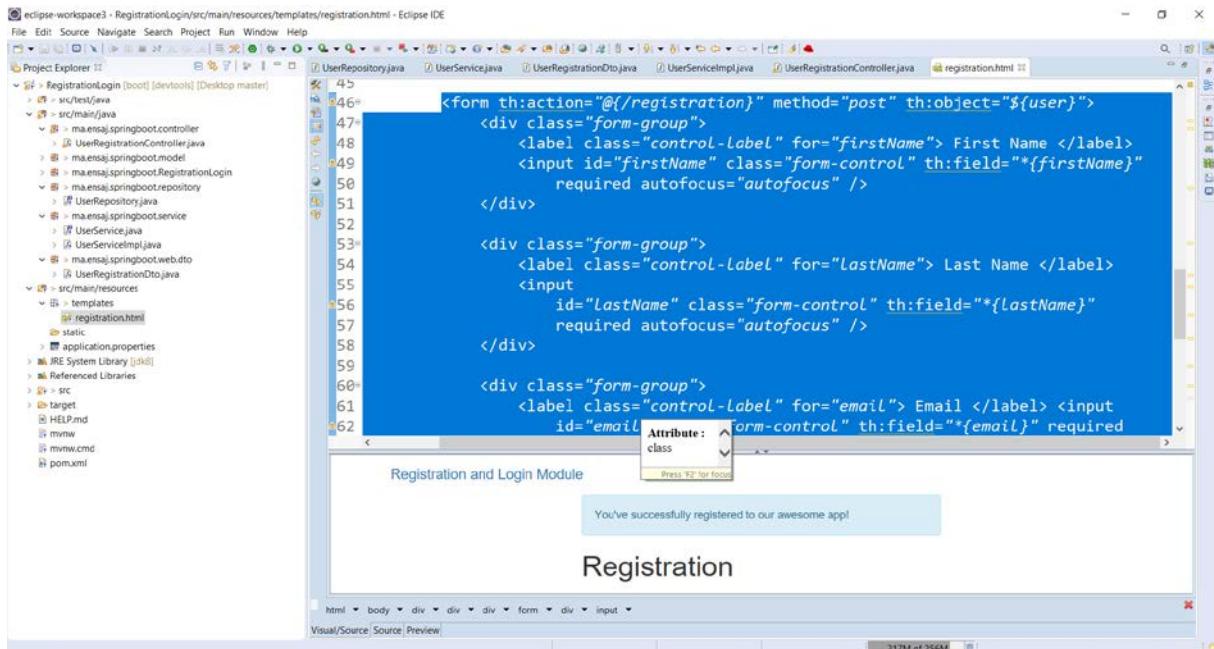


The screenshot shows the Eclipse IDE interface with the registration.html file open in the editor. The code uses Thymeleaf syntax to display a success message if the registration was successful. The message is contained within a div that has a class of "alert alert-info". The message text is "You've successfully registered to our awesome app!". Below the code, a preview window shows the registration page with the message displayed.

```
</nav>
<br>
<br>
<!-- Create HTML registration form --&gt;
<div class="container">
    <div class="row">
        <div class="col-md-6 col-md-offset-3">
            <!-- success message --&gt;
            &lt;div th:if="${param.success}"&gt;
                &lt;p&gt;&lt;/p&gt;
                &lt;div class="alert alert-info"&gt;You've successfully registered to our awesome app!&lt;/div&gt;
            &lt;/div&gt;
        &lt;/div&gt;
    &lt;/div&gt;
&lt;h1&gt;Registration&lt;/h1&gt;</pre>
```

Figure 170:Show success message using thymeleaf syntax

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



The screenshot shows the Eclipse IDE interface with the registration.html file open in the editor. The code demonstrates the use of Thymeleaf and HTML5 validation features for a registration form. It includes three form groups: one for the first name, one for the last name, and one for the email. Each group contains a label, an input field with the Thymeleaf attribute `th:field`, and an `autofocus` attribute. A tooltip for the email input field indicates it is required. Below the code, a preview window shows the registration page with the message "You've successfully registered to our awesome app!".

```
<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 />
    </div>
```

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

58. Preview of the registration form in eclipse

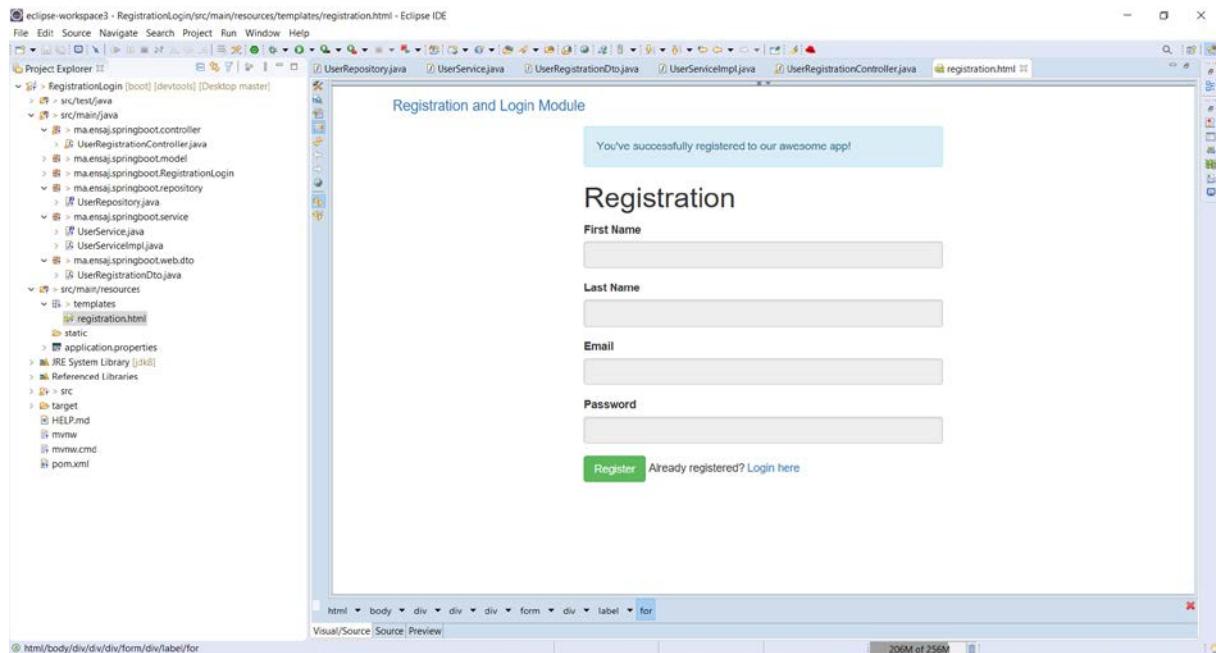


Figure 172:Preview of the registration form in eclipse

59. Run the app again

```
C:\Windows\System32\cmd.exe
C:\Users\hhass\Desktop\RegistrationLogin\RegistrationLogin>mvn spring-boot:run
```

Figure 173:Run the app again

60. The registration form in the browser

The screenshot shows a web browser window with the title bar "localhost / 127.0.0.1 / registration / un" and "Bootstrap 3.3.7 released | Bootstrap B". The main content area is titled "Registration and Login Module" and "Registration". The form contains four input fields: "First Name" (empty), "Last Name" (empty), "Email" (empty), and "Password" (empty). Below the form is a green "Register" button and a link "Already registered? Login here".

Figure 174:The registration form in the browser

61. Filling the form with valid data

The screenshot shows the same web browser window as Figure 174, but the form fields now contain data: "First Name" is "Hassane", "Last Name" is "HASSAR", "Email" is "hassanehassar@gmail.com", and "Password" is "*****". The "Register" button is visible at the bottom.

Figure 175:Filling the form with valid data

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

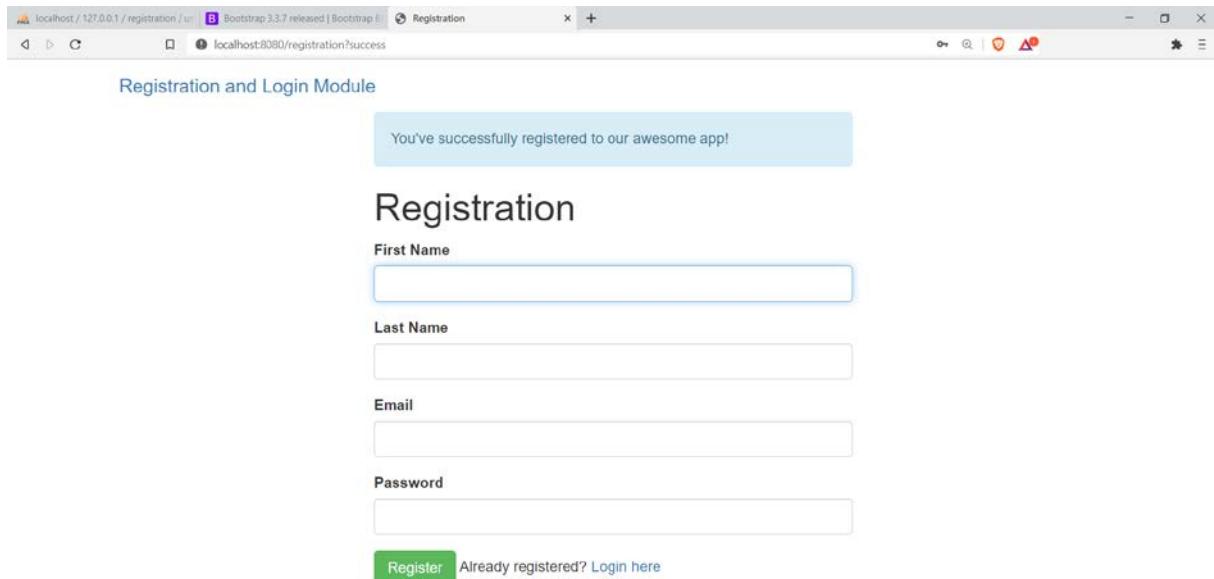


Figure 176:After Register button click the success message is displayed to the user

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

Showing rows 0 - 0 (1 total, Query took 0.0020 seconds.)

	Edit	Copy	Delete	Insert	Export	Import	Privileges	Operations	Tracking	Triggers
	Edit	Copy	Delete	Insert	Export	Import	Privileges	Operations	Tracking	Triggers
6	Edit	Copy	Delete							
	Edit	Copy	Delete							

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

64. In table role the user has default role

Showing rows 0 - 0 (1 total, Query took 0.0021 seconds.)

	Edit	Copy	Delete	Insert	Export	Import	Privileges	Operations	Tracking	Triggers
	Edit	Copy	Delete	Insert	Export	Import	Privileges	Operations	Tracking	Triggers
4	Edit	Copy	Delete							
	Edit	Copy	Delete							

Figure 178:In table role the user has default role

65. In the association table:

The screenshot shows the phpMyAdmin interface for the 'registration' database. The left sidebar lists various databases and tables, including 'users_roles'. The main area displays the 'users_roles' table with one row: user_id 6 and role_id 4. Below the table, there are sections for 'Query results operations' (Print, Copy to clipboard, Export, Display chart, Create view) and 'Bookmark this SQL query'.

Figure 179:In the association table

66. Form validation example

The screenshot shows a web browser displaying a registration form. The page title is 'Registration and Login Module'. A success message at the top says 'You've successfully registered to our awesome app!'. The registration form fields are: First Name (Hassane), Last Name (empty), Email (empty), and Password (empty). The 'Email' field has a validation error message: 'Please fill out this field.' A 'Register' button is at the bottom.

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

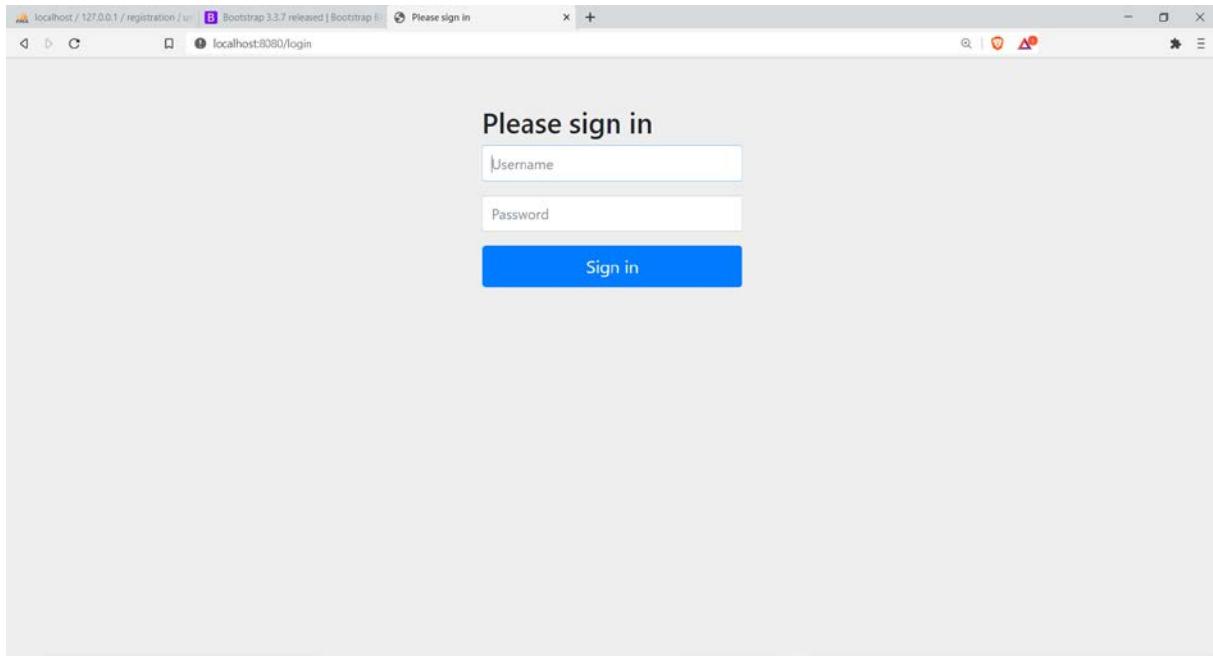


Figure 181:default login page

68. Default generated security password

```
PS C:\Windows\System32\cmd.exe - mvn spring-boot:run
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@3f2f60a09, org.springframework.security.web.header.HeaderWriterFilter@3399b743, org.springframew
ork.security.web.csrf.CsrfFilter@49f310f2, org.springframework.security.web.authentication.logout.LogoutFilter@3b8b9ec2, org.springframew
ork.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.spring
framework.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.MySQLInnoDBDialect
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 repositories
2021-01-12 20:39:06.172 INFO 13856 --- [restartedMain] DeferredRepositoryInitializationListener : Spring Data repositories initialized!
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

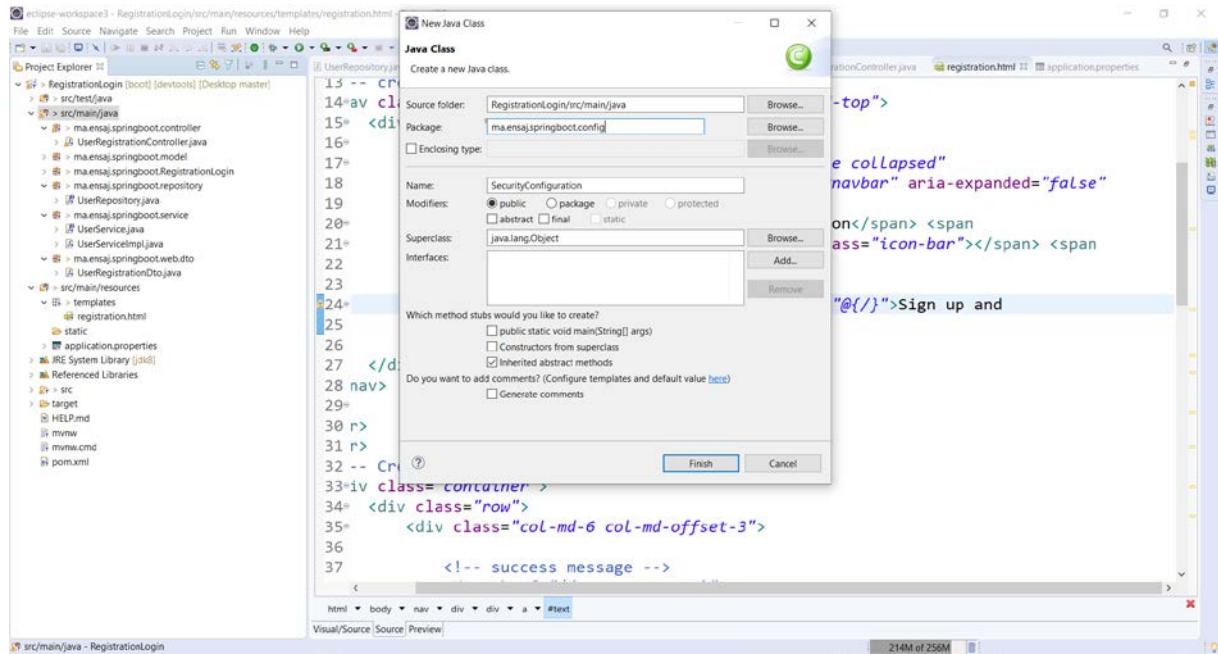


Figure 183:Create SecurityConfiguration class under the sub package config of the main package

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

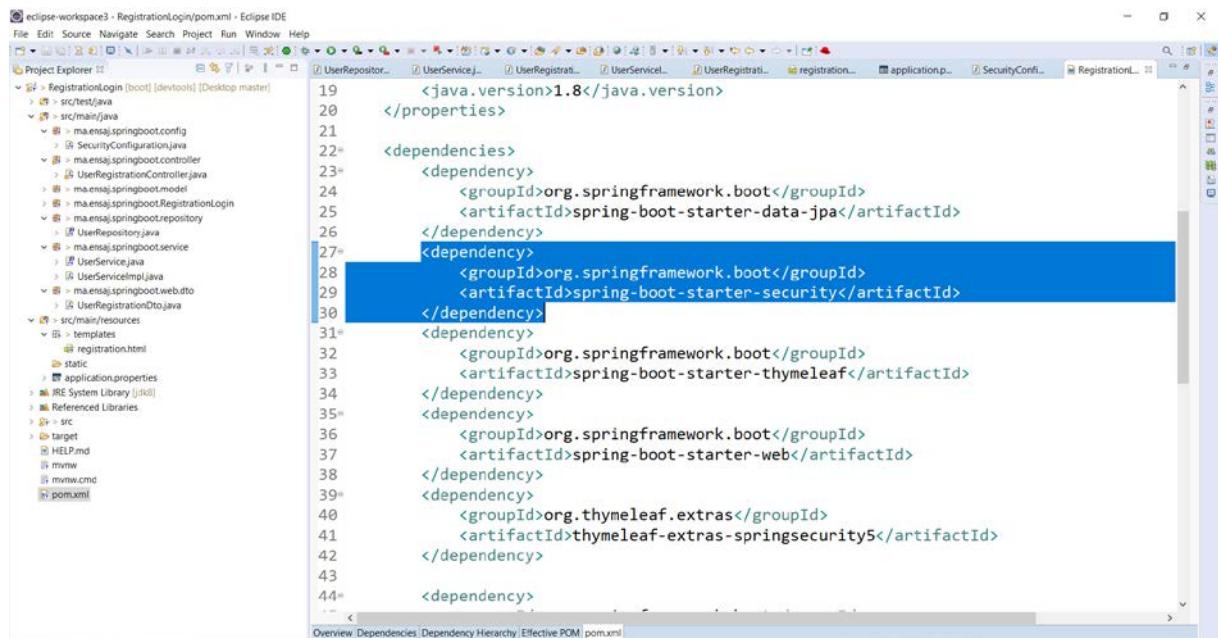
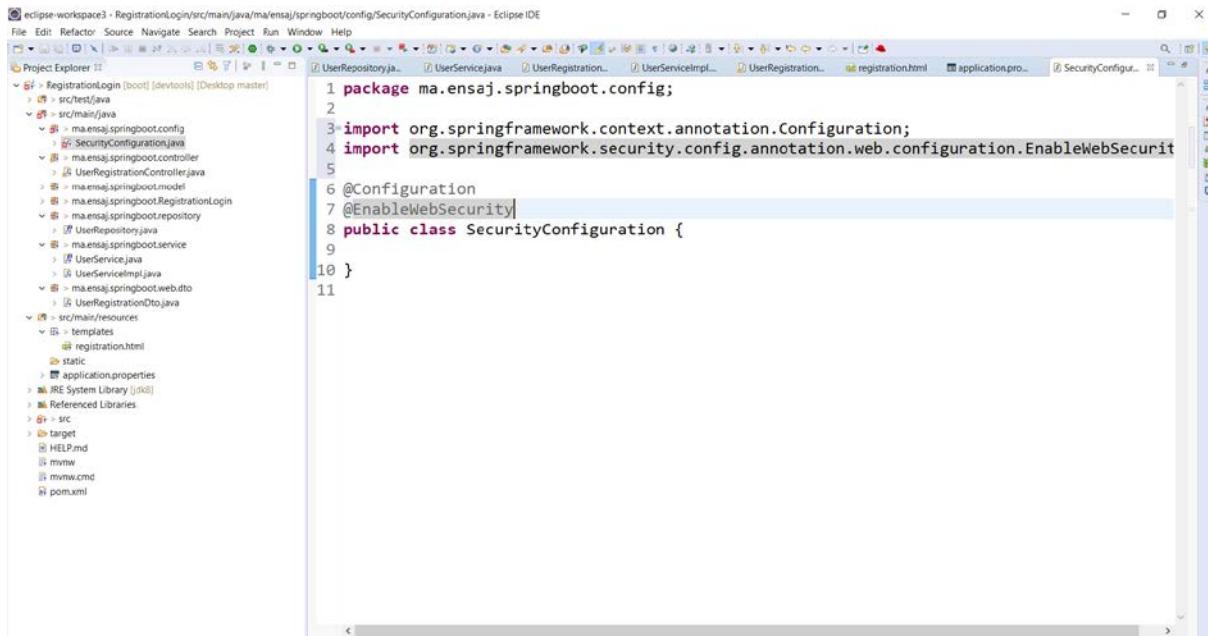


Figure 184:Notice : The presence of the security dependency in the pom.xml

71. Adding annotations @Configuration and @EnableWebSecurity



The screenshot shows the Eclipse IDE interface with the following details:

- Project Explorer:** Shows the project structure for "RegistrationLogin [boot] [devtools] [Desktop master]". It includes packages like "ma.ensaj.springboot.config" and "ma.ensaj.springboot.controller", and files like "SecurityConfiguration.java", "UserRegistrationController.java", "UserRegistrationServiceImpl.java", "UserRegistrationDto.java", "UserRepository.java", "UserService.java", and "UserServiceImpl.java".
- Code Editor:** Displays the "SecurityConfiguration.java" file with the following code:

```
1 package ma.ensaj.springboot.config;
2
3 import org.springframework.context.annotation.Configuration;
4 import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
5
6 @Configuration
7 @EnableWebSecurity
8 public class SecurityConfiguration {
9
10 }
```
- Toolbar:** Standard Eclipse IDE toolbar with various icons for file operations, search, and navigation.
- Bottom Status Bar:** Shows the status "registration.html" and "application.pro...".

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”.

```

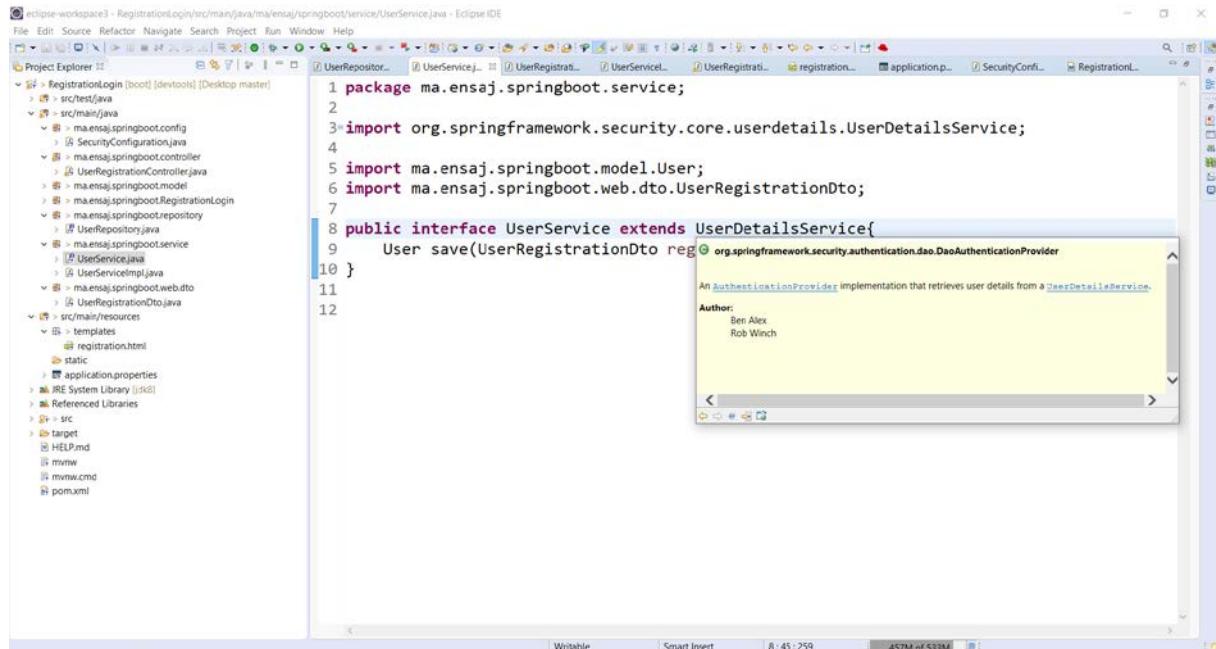
36     protected void configure(AuthenticationManagerBuilder auth) throws Exception {
37         auth.authenticationProvider(authenticationProvider());
38     }
39
40     @Override
41     protected void configure(HttpSecurity http) throws Exception {
42         http.authorizeRequests().antMatchers(
43             "/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**



The screenshot shows the Eclipse IDE interface with the project 'RegistrationLogin' open. The 'UserService.java' file is selected in the editor. The code defines a public interface 'UserService' that extends 'UserDetailsService'. The interface has one method, 'User save(UserRegistrationDto reg)'. A tooltip for 'UserDetailsService' is displayed, stating: 'An AuthenticationProvider implementation that retrieves user details from a UserDetailsService.' It also lists authors: Ben Alex and Rob Winch.

```
1 package ma.ensaj.springboot.service;
2
3 import org.springframework.security.core.userdetails.UserDetailsService;
4
5 import ma.ensaj.springboot.model.User;
6 import ma.ensaj.springboot.web.dto.UserRegistrationDto;
7
8 public interface UserService extends UserDetailsService{
9     User save(UserRegistrationDto reg)
10 }
```

Figure 188:UserService

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

```
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 {
    // TODO Auto-generated method stub
    return null;
}
```

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

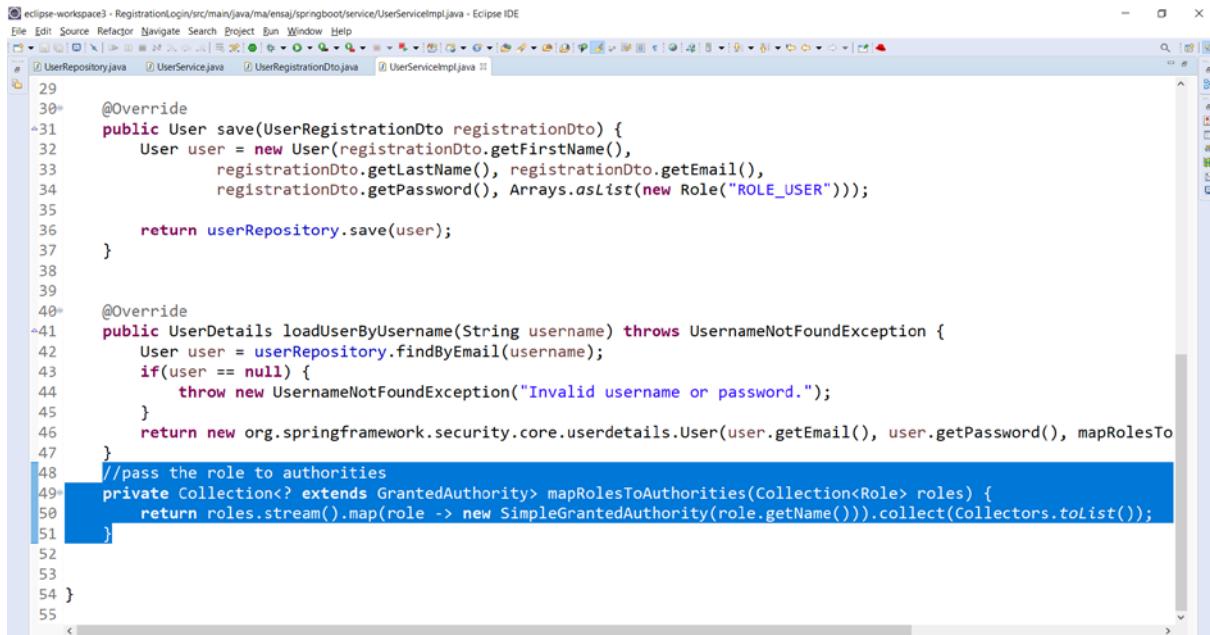
```
@Override
public User save(UserRegistrationDto registrationDto) {
    User user = new User(registrationDto.getFirstName(),
        registrationDto.getLastName(), registrationDto.getEmail(),
        registrationDto.getPassword(), Arrays.asList(new Role("ROLE_USER")));
    return userRepository.save(user);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

@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(), mapRolesToAuthorities());
}

//pass the role
private Collection<? extends GrantedAuthority> mapRolesToAuthorities(Collection<Role> roles) {
    // TODO Auto-generated method stub
    return null;
}
```

Figure 190:loadUserByUsername Implementation

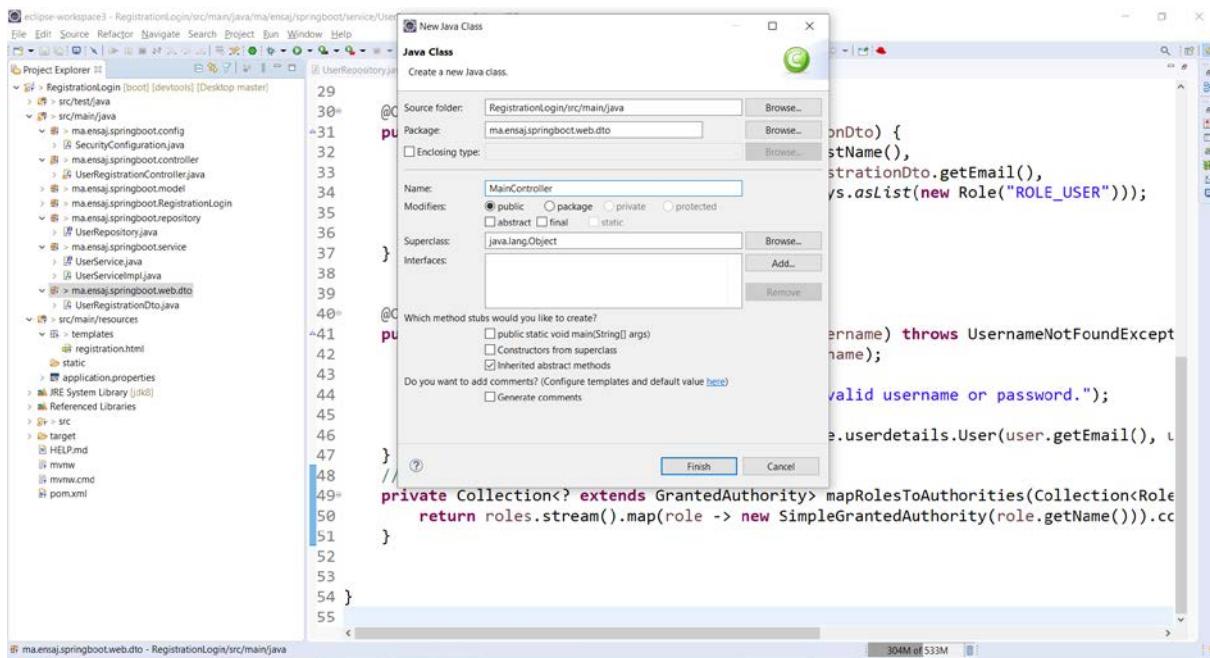
77. mapRolesToAuthorities method



```
29
30  @Override
31  public User save(UserRegistrationDto registrationDto) {
32      User user = new User(registrationDto.getFirstName(),
33                          registrationDto.getLastName(), registrationDto.getEmail(),
34                          registrationDto.getPassword(), Arrays.asList(new Role("ROLE_USER")));
35
36      return userRepository.save(user);
37  }
38
39
40  @Override
41  public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
42      User user = userRepository.findByEmail(username);
43      if(user == null) {
44          throw new UsernameNotFoundException("Invalid username or password.");
45      }
46      return new org.springframework.security.core.userdetails.User(user.getEmail(), user.getPassword(), mapRolesTo
47 )
48  //pass the role to authorities
49  private Collection<? extends GrantedAuthority> mapRolesToAuthorities(Collection<Role> roles) {
50      return roles.stream().map(role -> new SimpleGrantedAuthority(role.getName())).collect(Collectors.toList());
51  }
52
53
54 }
```

Figure 191:mapRolesToAuthorities method

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



New Java Class

Create a new Java class.

Source folder: RegistrationLogin/src/main/java

Package: ma.ensaj.springboot.web.dto

Enclosing type:

Name: MainController

Modifiers: public

Superclass: java.lang.Object

Interfaces:

Which method stubs would you like to create?

public static void main(String[] args)

Constructors from superclass

Inherited abstract methods

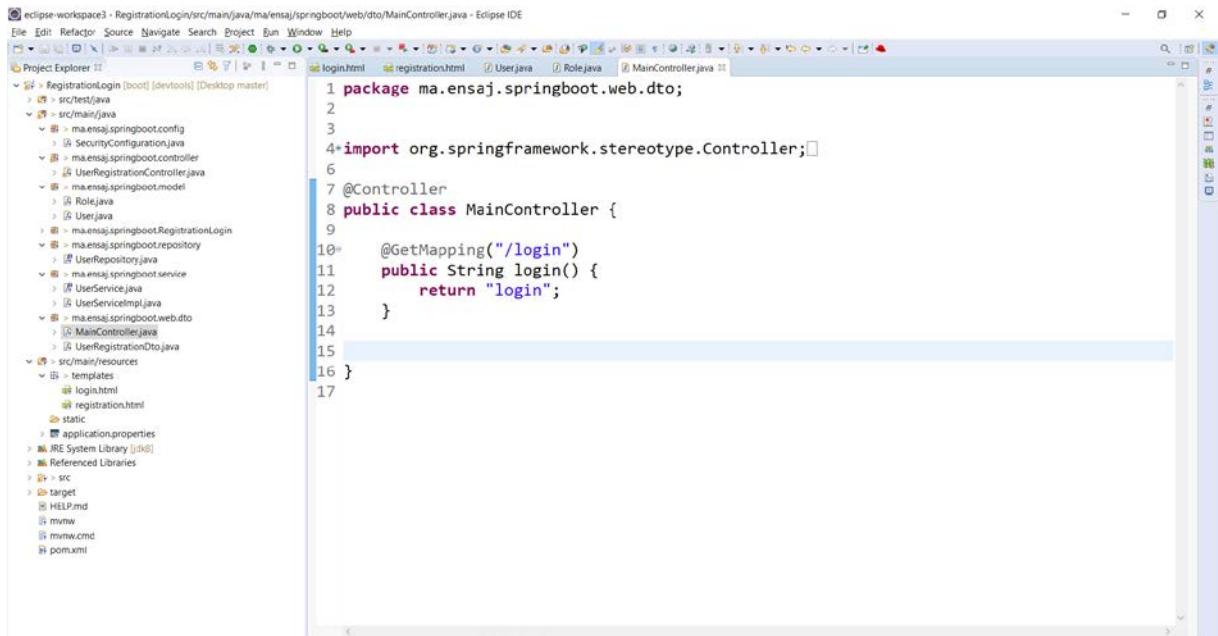
Do you want to add comments? (Configure templates and default value [see])

Finish Cancel

```
29
30  @Override
31  public User save(UserRegistrationDto registrationDto) {
32      User user = new User(registrationDto.getFirstName(),
33                          registrationDto.getLastName(), registrationDto.getEmail(),
34                          registrationDto.getPassword(), Arrays.asList(new Role("ROLE_USER")));
35
36      return userRepository.save(user);
37  }
38
39
40  @Override
41  public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
42      User user = userRepository.findByEmail(username);
43      if(user == null) {
44          throw new UsernameNotFoundException("Invalid username or password.");
45      }
46      return new org.springframework.security.core.userdetails.User(user.getEmail(), user.getPassword(), mapRolesTo
47 )
48  //pass the role to authorities
49  private Collection<? extends GrantedAuthority> mapRolesToAuthorities(Collection<Role> roles) {
50      return roles.stream().map(role -> new SimpleGrantedAuthority(role.getName())).collect(Collectors.toList());
51  }
52
53
54 }
```

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



The screenshot shows the Eclipse IDE interface with the MainController.java file open in the editor. The code defines a MainController class with a login() method annotated with @GetMapping("/login"). The code is as follows:

```
1 package ma.ensaj.springboot.web.dto;
2
3
4 import org.springframework.stereotype.Controller;
5
6
7 @Controller
8 public class MainController {
9
10    @GetMapping("/login")
11    public String login() {
12        return "login";
13    }
14
15
16 }
17
```

The Project Explorer view on the left shows the project structure, including src/main/java, src/test/java, and various configuration and service files.

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

```

16 import ma.ensaj.springboot.model.User;
17 import ma.ensaj.springboot.repository.UserRepository;
18 import ma.ensaj.springboot.web.dto.UserRegistrationDto;
19
20 @Service
21 public class UserServiceImpl implements UserService{
22
23     private UserRepository userRepository;
24     @Autowired
25     private BCryptPasswordEncoder passwordEncoder;
26
27     public UserServiceImpl(UserRepository userRepository) {
28         super();
29         this.userRepository = userRepository;
30     }
31
32
33     @Override
34     public User save(UserRegistrationDto registrationDto) {
35         User user = new User(registrationDto.getFirstName(),
36             registrationDto.getLastName(), registrationDto.getEmail(),
37             registrationDto.getPassword(), Arrays.asList(new Role("ROLE_USER")));
38
39         return userRepository.save(user);
40     }
41
42

```

Figure 194: Add BcryptPasswordEncoder reference to the UserServiceImpl class

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

```

22
23     private UserRepository userRepository;
24     @Autowired
25     private BCryptPasswordEncoder passwordEncoder;
26
27     public UserServiceImpl(UserRepository userRepository) {
28         super();
29         this.userRepository = userRepository;
30     }
31
32
33     @Override
34     public User save(UserRegistrationDto registrationDto) {
35         User user = new User(registrationDto.getFirstName(),
36             registrationDto.getLastName(), registrationDto.getEmail(),
37             passwordEncoder.encode(registrationDto.getPassword()), Arrays.asList(new
38
39         return userRepository.save(user);
40     }
41
42
43     @Override
44     public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
45         User user = userRepository.findByEmail(username);
46         if(user == null) {
47             throw new UsernameNotFoundException("Invalid username or password.");
48         }

```

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.

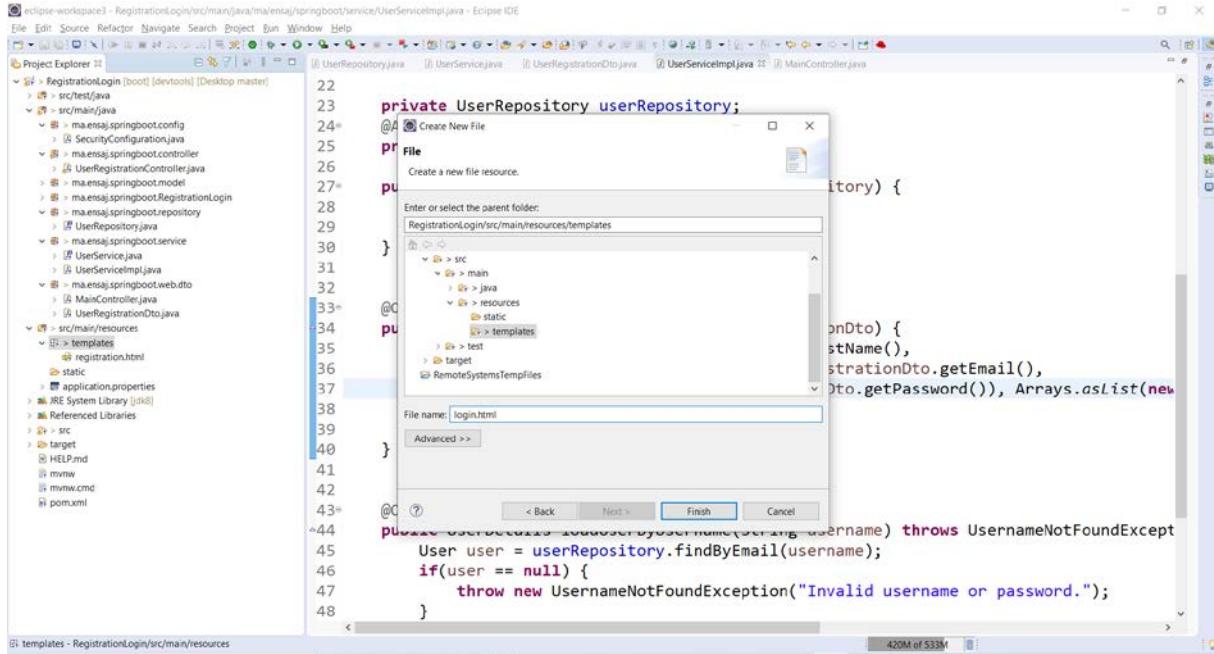


Figure 196:Create new Html file called login as we did for registration.

83. Nav bar navigation code

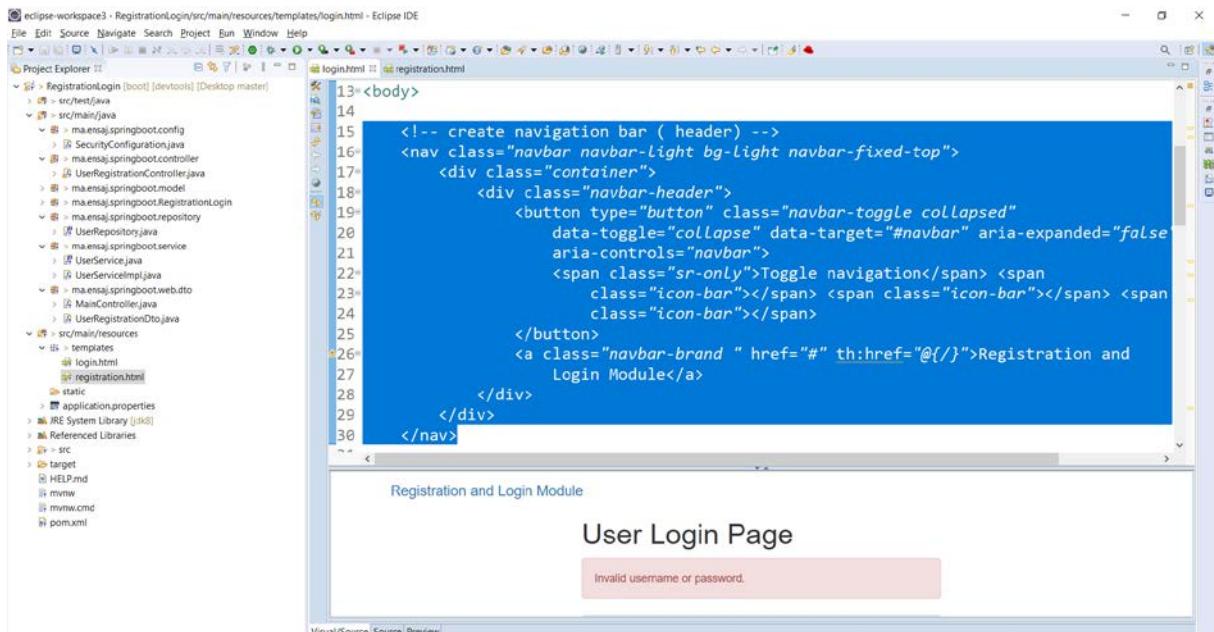
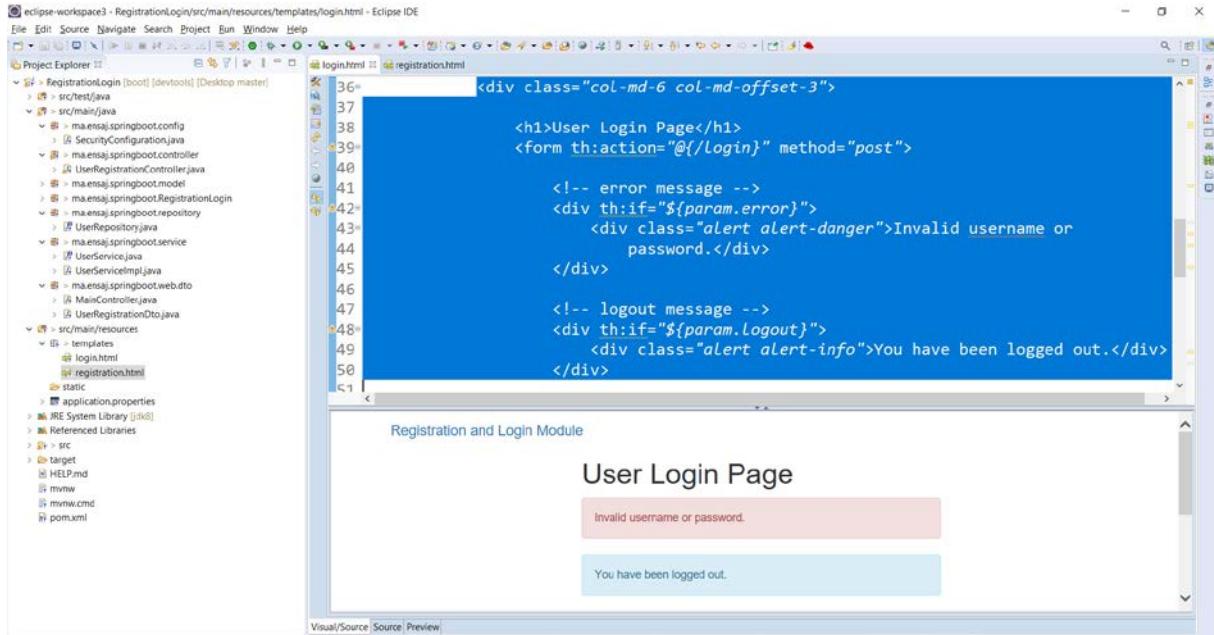


Figure 197:Nav bar navigation code

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



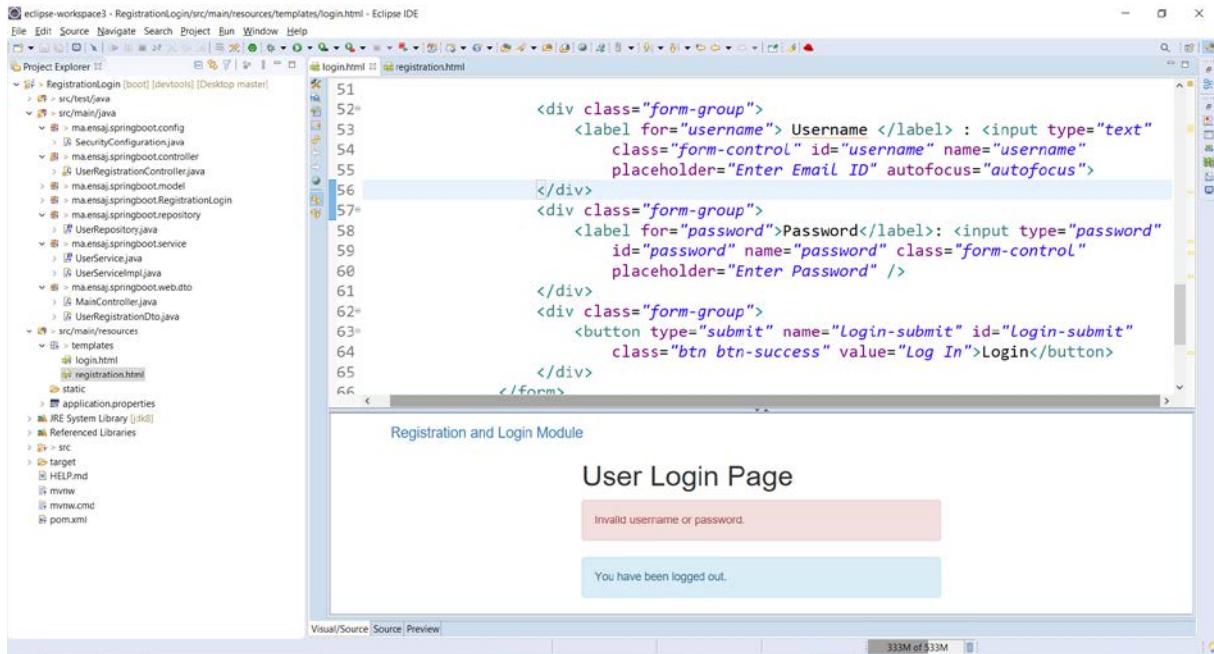
```

<div class="col-md-6 col-md-offset-3">
    <h1>User Login Page</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>
    </form>

```

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)



```

<div class="form-group">
    <label for="username"> Username </label> : <input type="text" 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-success" value="Log In"> Log In </button>
</div>

```

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

86. Add a link to the registration page

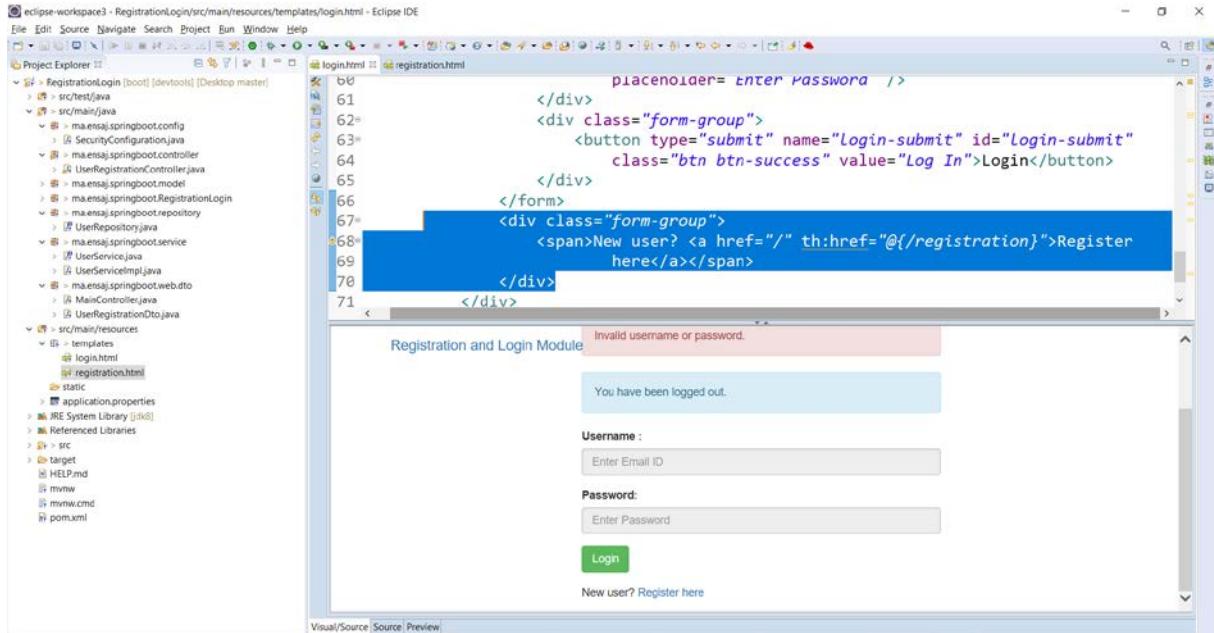


Figure 200: Add a link to the registration page

87. The preview page of login page in eclipse

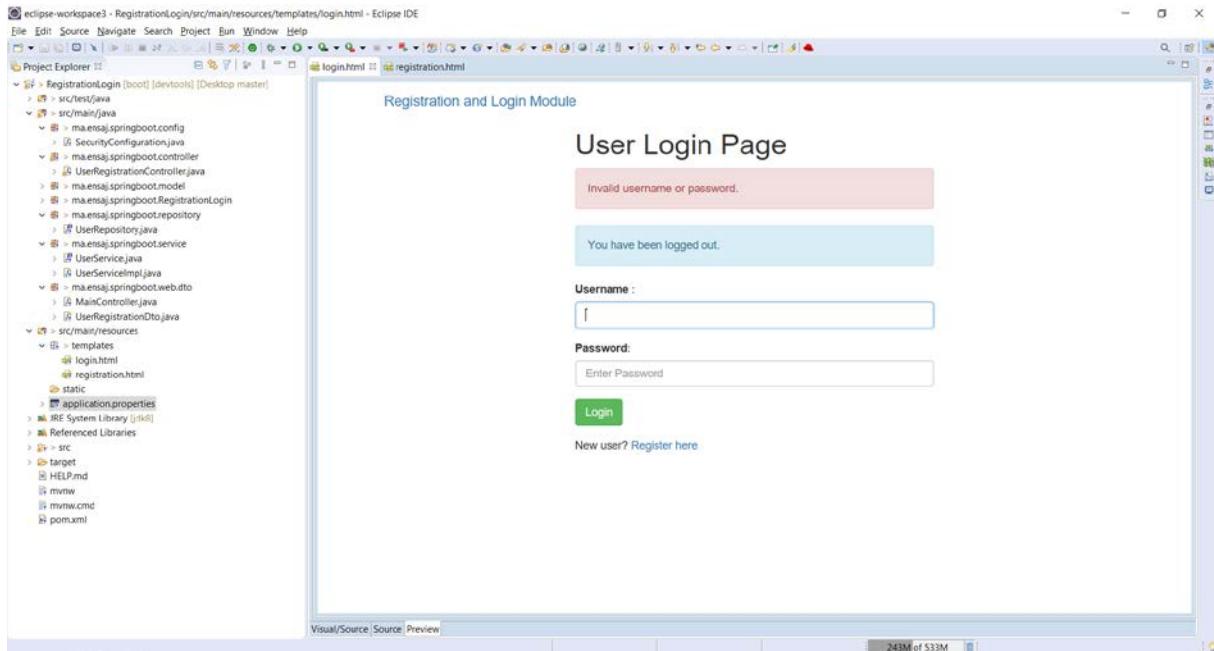


Figure 201: The preview page of login page in eclipse

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

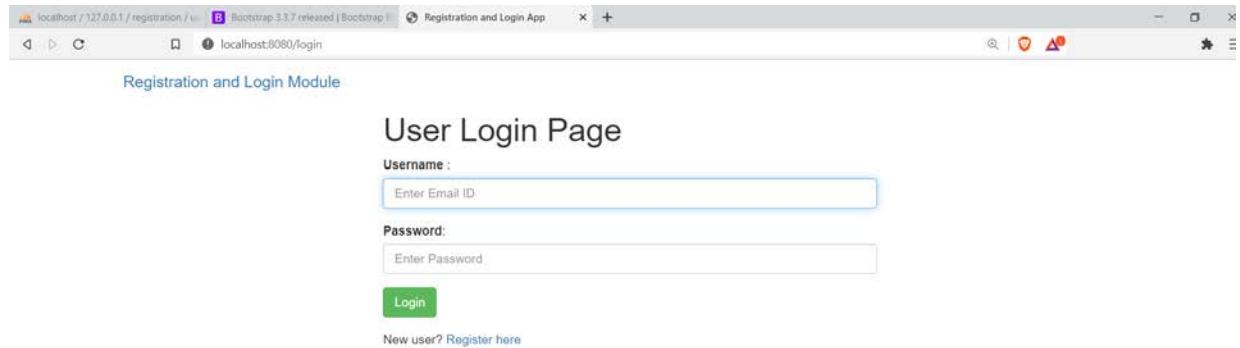


Figure 203:Login Page in the browser

90. Register new user

Registration and Login Module

Registration

First Name
Hassan

Last Name
HASSAR

Email
hassanehassar1999@gmail.com

Password

[Register](#) Already registered? [Login here](#)

Figure 204:Register new user

91. Confirmation of success registration is displayed

Registration and Login Module

You've successfully registered to our awesome app!

Registration

First Name
|

Last Name
|

Email
|

Password
|

[Register](#) Already registered? [Login here](#)

Figure 205:Confirmation of success registration is displayed

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

Registration and Login Module

User Login Page

Username :

Password:

New user? [Register here](#)

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.

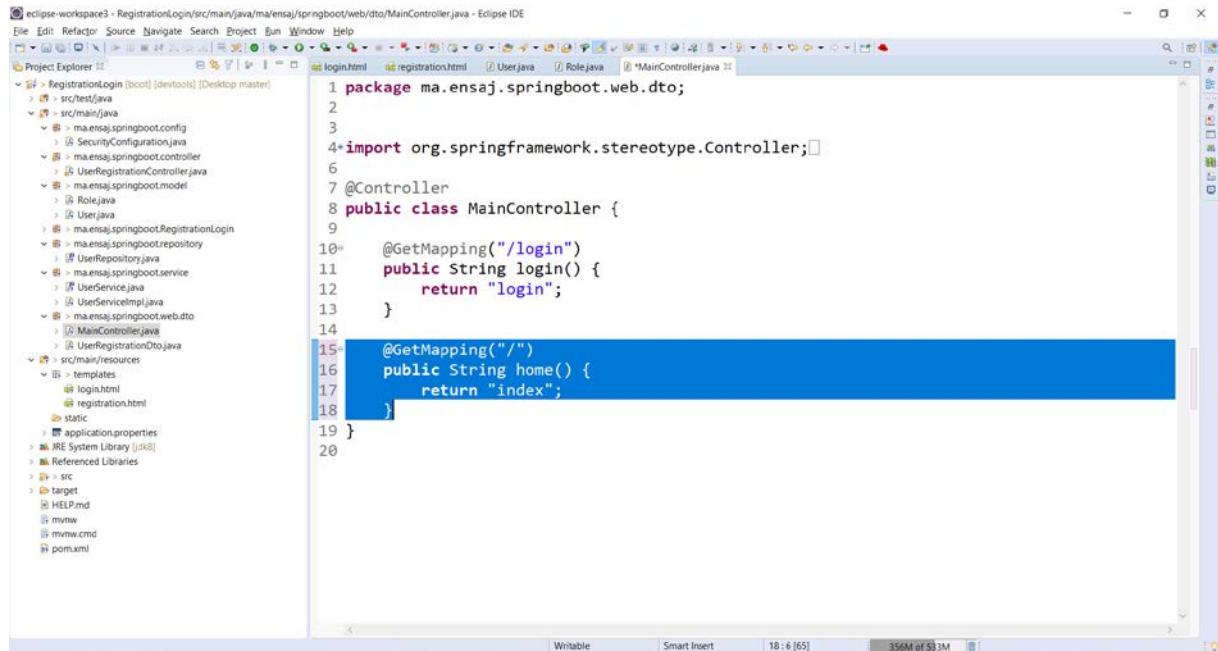


Figure 207:After logging button click, we get this error message of type Not Found 404 because

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



The screenshot shows the Eclipse IDE interface with the MainController.java file open in the editor. The code defines a MainController class with two methods: login() and home(). The home() method is highlighted with a blue selection bar.

```
1 package ma.ensaj.springboot.web.dto;
2
3
4 import org.springframework.stereotype.Controller;
5
6 @Controller
7 public class MainController {
8
9     @GetMapping("/login")
10    public String login() {
11        return "login";
12    }
13
14    @GetMapping("/")
15    public String home() {
16        return "index";
17    }
18
19 }
```

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

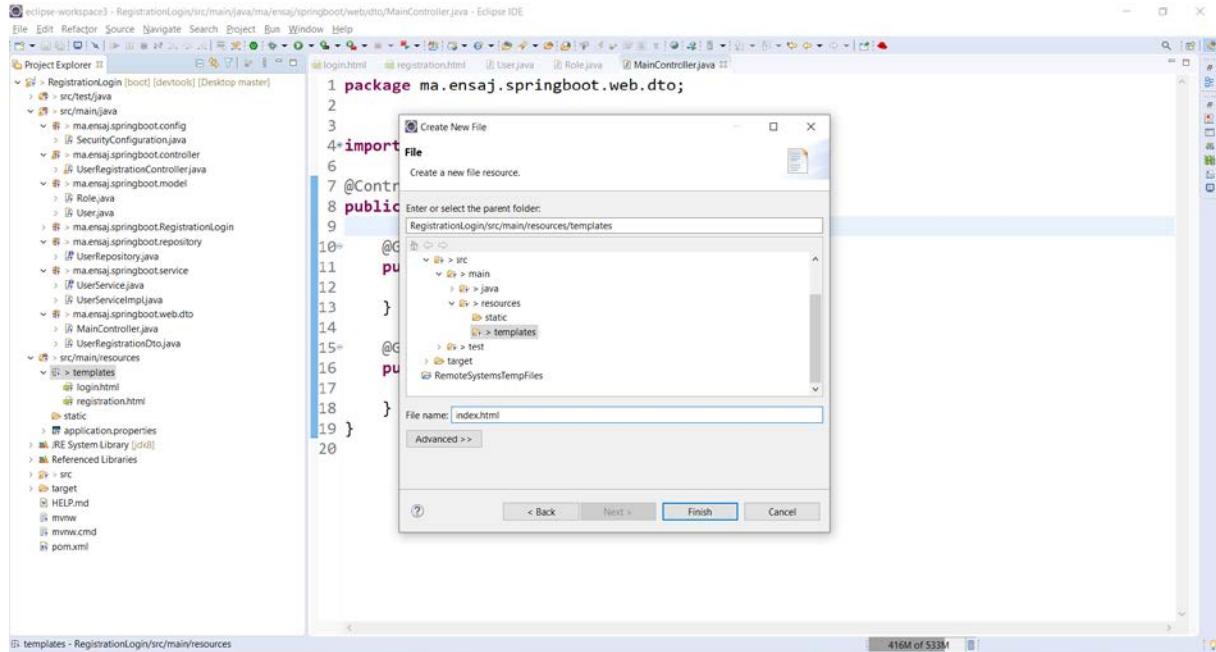


Figure 209: As we did before for the registration and login pages. Let's create a new file called index.html

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

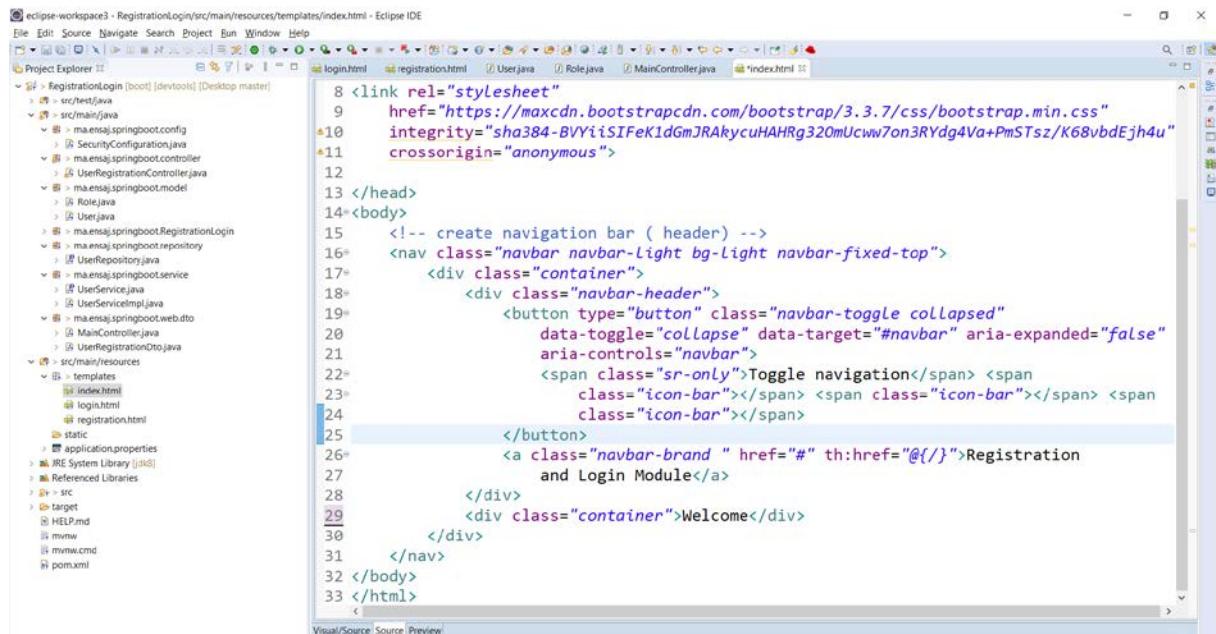


Figure 210: The navigation bar that's common for all views

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



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

A screenshot of the Eclipse IDE interface. The title bar says "eclipse-workspace3 - RegistrationLogin/pom.xml - Eclipse IDE". The left side shows the "Project Explorer" view with various Java files and resources. The right side shows the content of the "pom.xml" file. A blue rectangular highlight is placed over the Thymeleaf dependency section.

```
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-security</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-thymeleaf</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>org.thymeleaf.extras</groupId>
        <artifactId>thymeleaf-extras-springsecurity5</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-devtools</artifactId>
        <scope>runtime</scope>
    </dependency>
```

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

```
8 <link rel="stylesheet"
9   href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
10  integrity="sha384-BVYii5IfEKifJRAkycuHAHRg32OmUcw7on3RYdg4Va+PmSTsz/K68vbdEjh4u"
11  crossorigin="anonymous">
12
13 </head>
14<body>
15  <!-- create navigation bar ( header ) -->
16  <nav class="navbar navbar-light bg-light navbar-fixed-top">
17    <div class="container">
18      <div class="navbar-header">
19        <button type="button" class="navbar-toggle collapsed"
20          data-toggle="collapse" data-target="#navbar" aria-expanded="false"
21          aria-controls="navbar">
22          <span class="sr-only">Toggle navigation</span> <span
23            class="icon-bar"></span> <span class="icon-bar"></span> <span
24            class="icon-bar"></span>
25        </button>
26        <a class="navbar-brand " href="#" th:href="@{/}">Registration
27          and Login Module</a>
28      </div>
29      <div class="container">Welcome
30        <span sec:authentication="principal.username"> User</span>
31      </div>
32    </nav>
33 </body>
```

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

102. Displaying the email of the user is working well

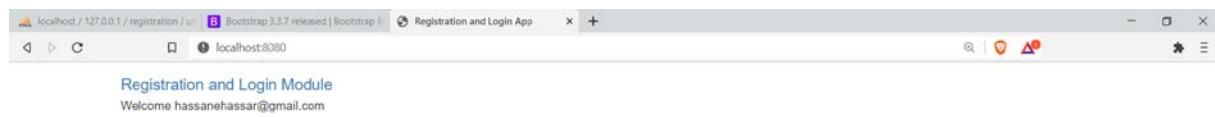


Figure 214:Displaying the email of the user is working well

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

```
14<
15 :create navigation bar ( header ) -->
16<class="navbar navbar-Light bg-light navbar-fixed-top">
17<jiv class="container">
18<div class="navbar-header">
19<button type="button" class="navbar-toggle collapsed"
20 data-toggle="collapse" data-target="#navbar" aria-expanded="false"
21 aria-controls="navbar">
22<span class="sr-only">Toggle navigation</span> <span
23 class="icon-bar"></span> <span class="icon-bar"></span> <span
24 class="icon-bar"></span>
25</button>
26<a class="navbar-brand " href="#" th:href="@{/}">Registration
27 and Login Module</a>
28</div>
29<div id="navbar" class="collapse navbar-collapse">
30<ul class="nav navbar-nav">
31<li sec:authorize="isAuthenticated()"><a th:href="@{/logout}">Logout</a></li>
32</ul>
33</div>
34<div class="container">Welcome
35<span sec:authentication="principal.username"> User</span></div>
36</div>
37>
38
39
```

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

```
protected void configure(AuthenticationManagerBuilder auth) throws Exception {
    auth.authenticationProvider(authenticationProvider());
}

@Override
protected void configure(HttpSecurity http) throws Exception {
    http.authorizeRequests().antMatchers(
        "/registration**",
        "/js/**",
        "/css/**",
        "/img/**").permitAll()
    .anyRequest().authenticated()
    .and()
    .formLogin()
    .loginPage("/login")
    .permitAll()
    .and()
    .logout()
    .invalidateHttpSession(true)
    .clearAuthentication(true)
    .logoutRequestMatcher(new AntPathRequestMatcher("/logout"))
    .logoutSuccessUrl("/login?logout")
    .permitAll();
}
```

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

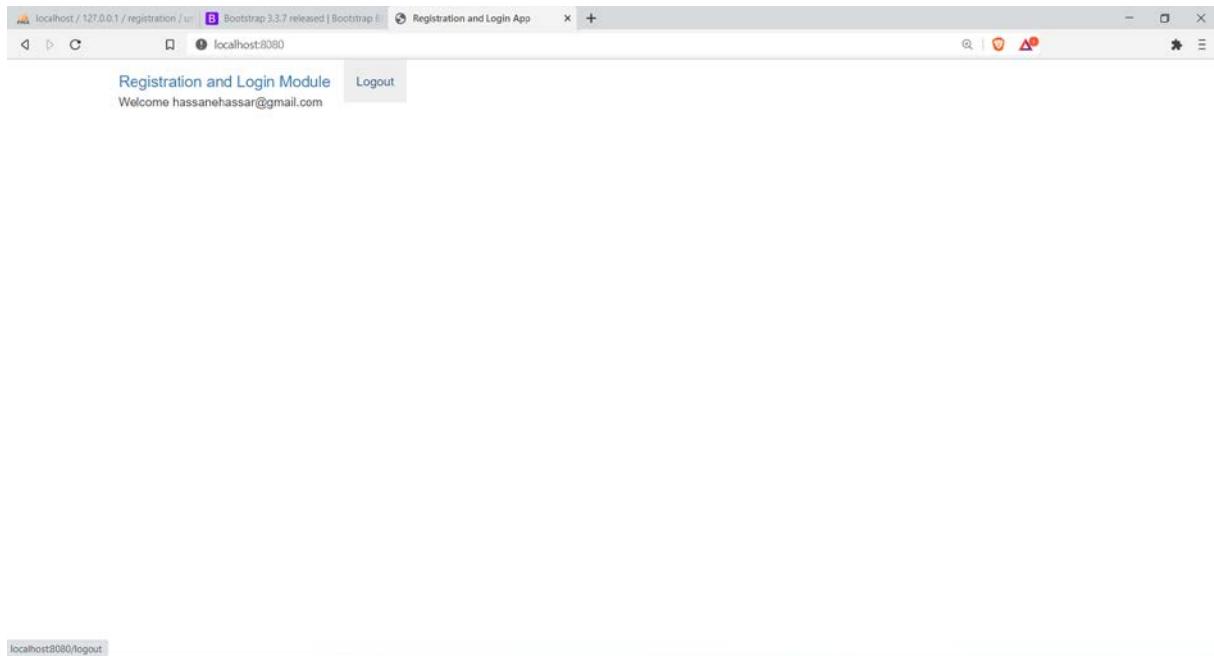


Figure 217:We run the app and test if the Logout link was successfully added to the navbar

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

The screenshot shows a web browser window with the title "Registration and Login App". The URL in the address bar is "localhost:8080/login?logout". The page content is titled "User Login Page". A blue banner at the top says "You have been logged out.". Below the banner is a form with two input fields: "Username :" and "Password:". The "Username :" field has the placeholder "Enter Email ID" and the "Password:" field has the placeholder "Enter Password". There is a green "Login" button below the fields. At the bottom of the page, there is a link "New user? Register here".

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:

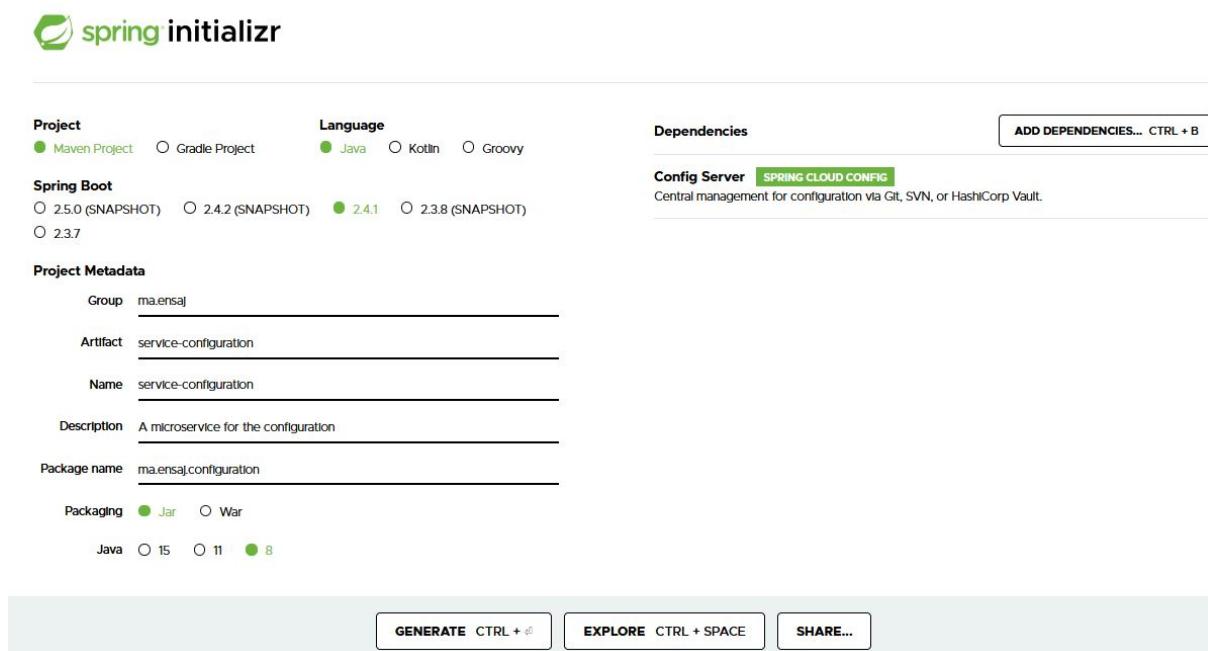


Figure 219:Spring Initializr

- 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\Projects\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-parent/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-dependencies/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-dependencies/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-dependencies/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-dependencies/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-dependencies/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-dependencies/3.0.0/spring-cloud-netflix-dependencies-3.0.0.pom (8.8 kB at 39 kB/s)
```

Figure 220:make the project an eclipse project

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

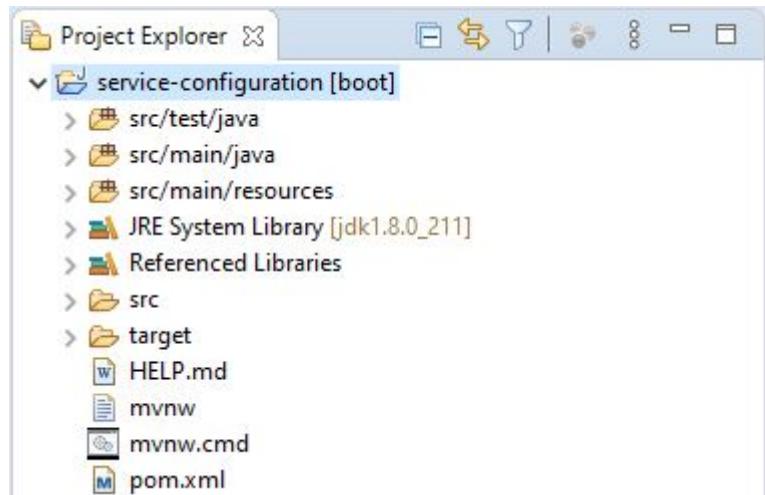


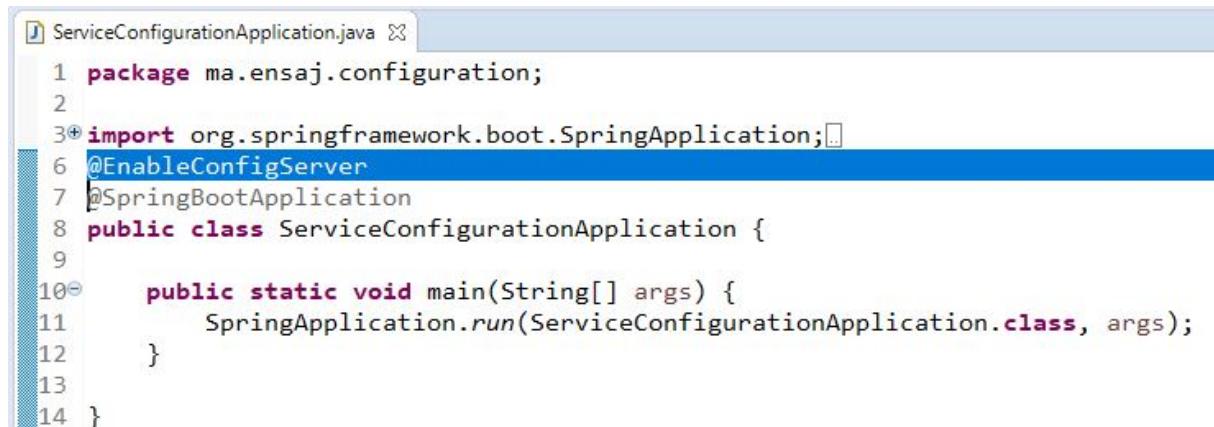
Figure 221:Project Structure

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.



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



```
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.

```
MINGW64:/c/Users/MAZER Omar/cloud-config
Mazer Omar@DESKTOP-IJU5CQT MINGW64 ~ (master)
$ mkdir cloud-config

Mazer Omar@DESKTOP-IJU5CQT MINGW64 ~ (master)
$ cd cloud-config

Mazer Omar@DESKTOP-IJU5CQT MINGW64 ~/cloud-config (master)
$ git init
Initialized empty Git repository in c:/users/MAZER Omar/cloud-config/.git/
```

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

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

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

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

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

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

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** :

```
application.properties X service-moneyTransfer.properties  
C: > Users > MAZER Omar > cloud-config > application.properties  
1 spring.datasource.username = root  
2 spring.datasource.password =  
3  
4 spring.jpa.show-sql = true  
5  
6 spring.jpa.hibernate.ddl-auto = update
```

Figure 226:Configuration of the database and the Persistence Framework

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

```
application.properties X service-moneyTransfer.properties X service-phoneRecharge.properties  
C: > Users > MAZER Omar > cloud-config > service-moneyTransfer.properties  
1 spring.datasource.url = jdbc:mysql://localhost:3306/MoneyTransferDb?  
2 server.port:8081
```

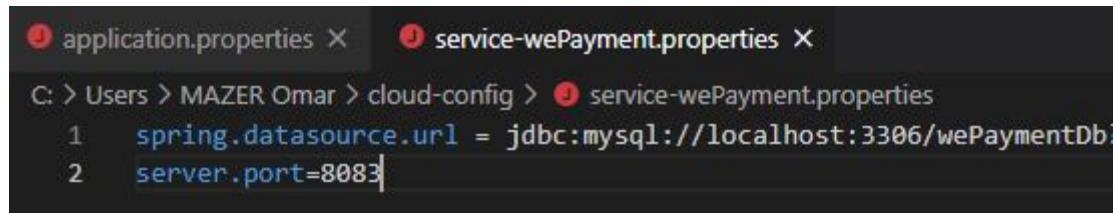
Figure 227:We set datasource url and the server.port:8081 for moneyTransfer service.

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

```
application.properties X service-phoneRecharge.properties X service-wePayment.properties  
C: > Users > MAZER Omar > cloud-config > service-phoneRecharge.properties  
1 spring.datasource.url = jdbc:mysql://localhost:3306/PhoneRechargeDb?  
2 server.port:8082
```

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

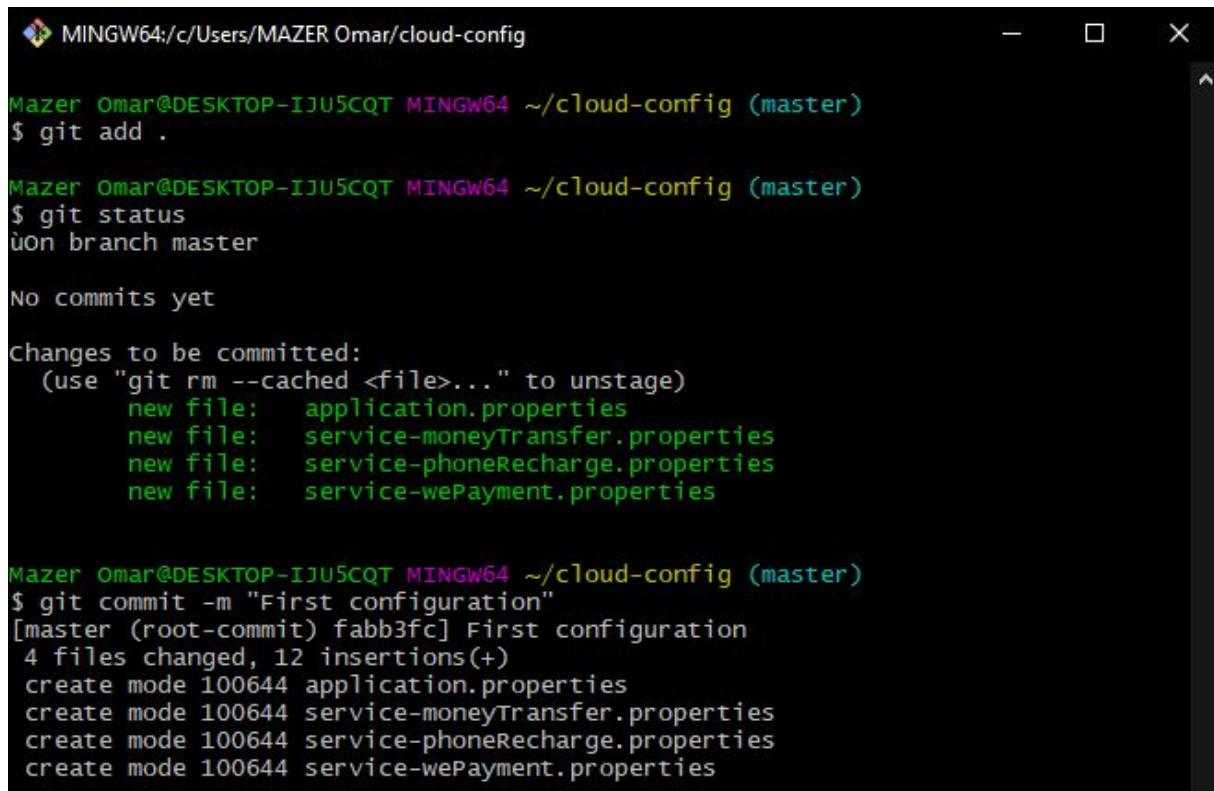
12. We set **datasource url** and the **server.port:8083** for **wePayment** service.



```
application.properties X service-wePayment.properties X
C: > Users > MAZER Omar > cloud-config > service-wePayment.properties
1 spring.datasource.url = jdbc:mysql://localhost:3306/wePaymentDb
2 server.port=8083
```

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 .**



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

Mazer Omar@DESKTOP-IJU5CQT MINGW64 ~/cloud-config (master)
$ git status
On branch master

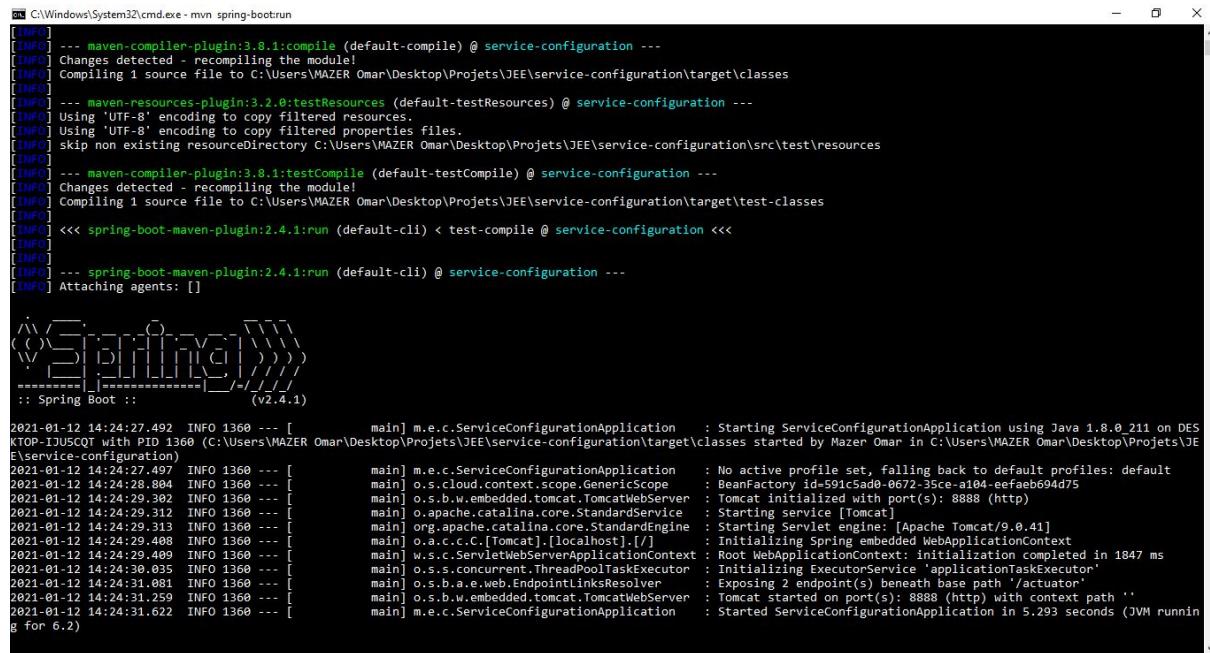
No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:   application.properties
    new file:   service-moneyTransfer.properties
    new file:   service-phoneRecharge.properties
    new file:   service-wePayment.properties

Mazer Omar@DESKTOP-IJU5CQT MINGW64 ~/cloud-config (master)
$ git commit -m "First configuration"
[master (root-commit) fabb3fc] First configuration
 4 files changed, 12 insertions(+)
 create mode 100644 application.properties
 create mode 100644 service-moneyTransfer.properties
 create mode 100644 service-phoneRecharge.properties
 create mode 100644 service-wePayment.properties
```

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



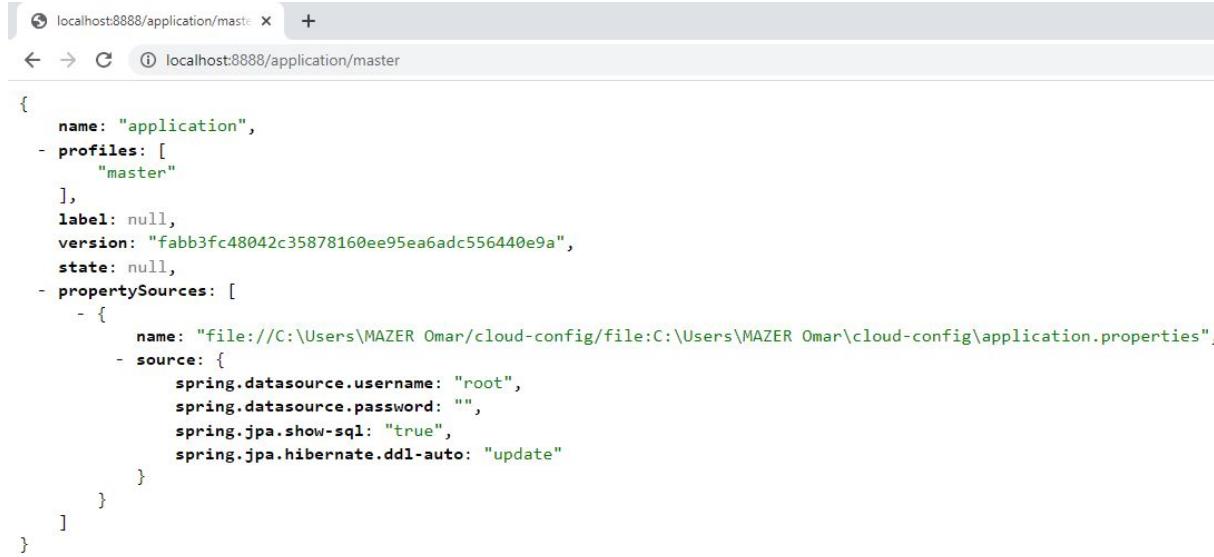
```
[INFO] --- maven-compiler-plugin:3.8.1:compile (default-compile) @ service-configuration ---
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 1 source file to C:\Users\MAZER Omar\Desktop\Projets\JEE\service-configuration\target\classes
[INFO] --- maven-resources-plugin:3.2.0:testResources (default-testResources) @ service-configuration ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Using 'UTF-8' encoding to copy filtered properties files.
[INFO] skip non existing resourceDirectory C:\Users\MAZER Omar\Desktop\Projets\JEE\service-configuration\src\test\resources
[INFO] --- maven-compiler-plugin:3.8.1:testCompile (default-testCompile) @ service-configuration ---
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 1 source file to C:\Users\MAZER Omar\Desktop\Projets\JEE\service-configuration\target\test-classes
[INFO] <<< spring-boot-maven-plugin:2.4.1:run (default-cli) < test-compile @ service-configuration <<<
[INFO]
[INFO] --- spring-boot-maven-plugin:2.4.1:run (default-cli) @ service-configuration ---
[INFO] Attaching agents: []

:: Spring Boot ::          (v2.4.1)

2021-01-12 14:24:27.492 INFO 1360 --- [           main] m.e.c.ServiceConfigurationApplication : Starting ServiceConfigurationApplication using Java 1.8.0_211 on DESKTOP-IJUUSQQT with PID 1360 (C:\Users\MAZER Omar\Desktop\Projets\JEE\service-configuration\target\classes started by Mazer Omar in C:\Users\MAZER Omar\Desktop\Projets\JEE\service-configuration)
2021-01-12 14:24:27.497 INFO 1360 --- [           main] m.e.c.ServiceConfigurationApplication : No active profile set, falling back to default profiles: default
2021-01-12 14:24:28.884 INFO 1360 --- [           main] o.s.cloud.context.scope.GenericScope : BeanFactory id=591c5ad0-0672-35ce-a104-eefae694d75
2021-01-12 14:24:29.382 INFO 1360 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8888 (http)
2021-01-12 14:24:29.312 INFO 1360 --- [           main] o.apache.catalina.core.StandardService : Starting service [tomcat]
2021-01-12 14:24:29.313 INFO 1360 --- [           main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.41]
2021-01-12 14:24:29.408 INFO 1360 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2021-01-12 14:24:29.409 INFO 1360 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 1847 ms
2021-01-12 14:24:29.409 INFO 1360 --- [           main] o.s.s.concurrent.ThreadPoolTaskExecutor : Initializing ExecutorService 'applicationTaskExecutor'
2021-01-12 14:24:31.081 INFO 1360 --- [           main] o.s.b.a.e.web.EndpointLinksResolver : Exposing 2 endpoint(s) beneath base path '/actuator'
2021-01-12 14:24:31.259 INFO 1360 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8888 (http) with context path ''
2021-01-12 14:24:31.622 INFO 1360 --- [           main] m.e.c.ServiceConfigurationApplication : Started ServiceConfigurationApplication in 5.293 seconds (JVM running for 6.2)
```

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

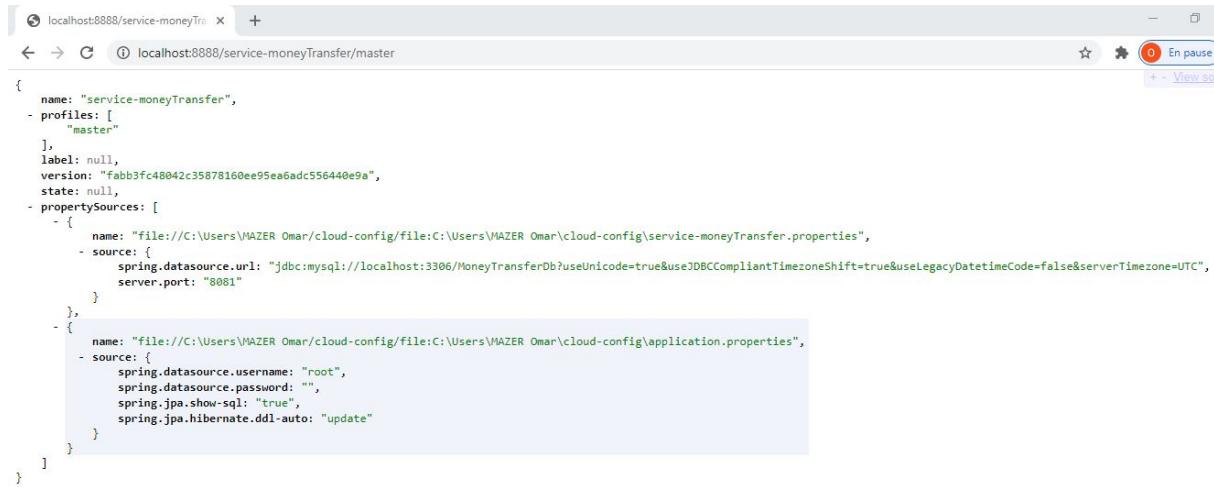
15. Let's test it in the browser



A screenshot of a web browser window titled "localhost:8888/application/master". The page displays a JSON configuration file for the "application" profile. The configuration includes a "label" (null), a "version" ("fabb3fc48042c35878160ee95ea6adc556440e9a"), and a "state" (null). The "propertySources" section contains two entries: one from a file at "C:\Users\MAZER Omar\cloud-config\file:C:\Users\MAZER Omar\cloud-config\application.properties" and another from the same file. Both entries define a "spring.datasource.username" of "root" and a "spring.datasource.password" of "".

```
{
  "name": "application",
  "- profiles": [
    "master"
  ],
  "label": null,
  "version": "fabb3fc48042c35878160ee95ea6adc556440e9a",
  "state": null,
  "- propertySources": [
    {
      "name": "file://C:\Users\MAZER Omar\cloud-config\file:C:\Users\MAZER Omar\cloud-config\application.properties",
      "- source": {
        "spring.datasource.username": "root",
        "spring.datasource.password": "",
        "spring.jpa.show-sql": "true",
        "spring.jpa.hibernate.ddl-auto": "update"
      }
    }
  ]
}
```

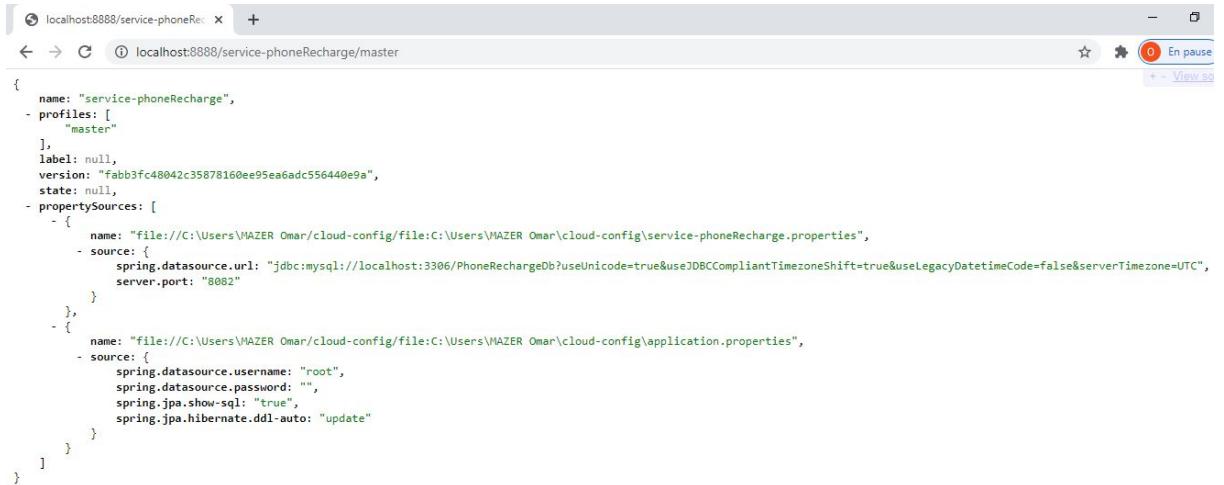
Figure 232:Test



A screenshot of a web browser window titled "localhost:8888/service-moneyTransfer/master". The page displays a JSON configuration file for the "service-moneyTransfer" profile. The configuration includes a "label" (null), a "version" ("fabb3fc48042c35878160ee95ea6adc556440e9a"), and a "state" (null). The "propertySources" section contains two entries: one from a file at "C:\Users\MAZER Omar\cloud-config\file:C:\Users\MAZER Omar\cloud-config\service-moneyTransfer.properties" and another from the same file. The first entry defines a "spring.datasource.url" of "jdbc:mysql://localhost:3306/MoneyTransferDb?useUnicode=true&useJDBCCompliantTimezoneShift=true&useLegacyDatetimeCode=false&serverTimezone=UTC" and a "server.port" of "8081". The second entry defines a "spring.datasource.username" of "root" and a "spring.datasource.password" of "".

```
{
  "name": "service-moneyTransfer",
  "- profiles": [
    "master"
  ],
  "label": null,
  "version": "fabb3fc48042c35878160ee95ea6adc556440e9a",
  "state": null,
  "- propertySources": [
    {
      "name": "file://C:\Users\MAZER Omar\cloud-config\file:C:\Users\MAZER Omar\cloud-config\service-moneyTransfer.properties",
      "- source": {
        "spring.datasource.url": "jdbc:mysql://localhost:3306/MoneyTransferDb?useUnicode=true&useJDBCCompliantTimezoneShift=true&useLegacyDatetimeCode=false&serverTimezone=UTC",
        "server.port": "8081"
      }
    },
    {
      "name": "file://C:\Users\MAZER Omar\cloud-config\file:C:\Users\MAZER Omar\cloud-config\application.properties",
      "- source": {
        "spring.datasource.username": "root",
        "spring.datasource.password": "",
        "spring.jpa.show-sql": "true",
        "spring.jpa.hibernate.ddl-auto": "update"
      }
    }
  ]
}
```

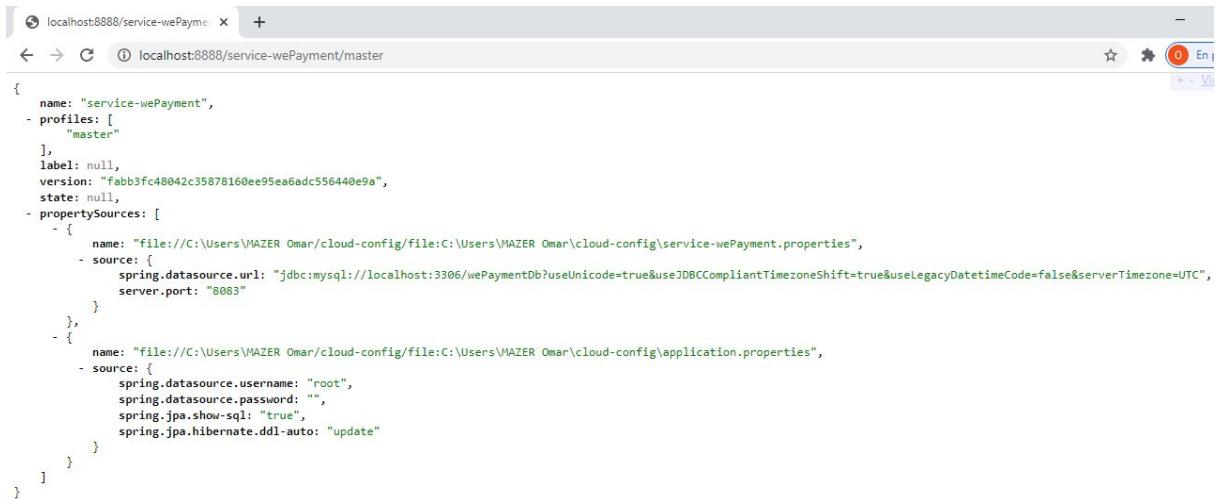
Figure 233:Test service-money



The screenshot shows a browser window with the URL `localhost:8888/service-phoneRecharge/master`. The page displays a JSON configuration file for the `service-phoneRecharge` application. The configuration includes a profile named "master", a database connection with URL `jdbc:mysql://localhost:3306/PhoneRechargeDb?useUnicode=true&useJDBCCompliantTimezoneShift=true&useLegacyDatetimeCode=false&serverTimezone=UTC`, port `8082`, and a second profile section with properties for `application.properties`.

```
{  
  name: "service-phoneRecharge",  
  profiles: [  
    "master"  
  ],  
  label: null,  
  version: "fabbb3fc48042c35878160ee95ea6adc556440e9a",  
  state: null,  
  propertySources: [  
    {  
      name: "file:///C:/Users/MAZER Omar/cloud-config/file:C:/Users/MAZER Omar/cloud-config/service-phoneRecharge.properties",  
      source: {  
        spring.datasource.url: "jdbc:mysql://localhost:3306/PhoneRechargeDb?useUnicode=true&useJDBCCompliantTimezoneShift=true&useLegacyDatetimeCode=false&serverTimezone=UTC",  
        server.port: "8082"  
      }  
    },  
    {  
      name: "file:///C:/Users/MAZER Omar/cloud-config/file:C:/Users/MAZER Omar/cloud-config/application.properties",  
      source: {  
        spring.datasource.username: "root",  
        spring.datasource.password: "",  
        spring.jpa.show-sql: "true",  
        spring.jpa.hibernate.ddl-auto: "update"  
      }  
    }  
  ]  
}
```

Figure 234:Test service-phoneRecharge



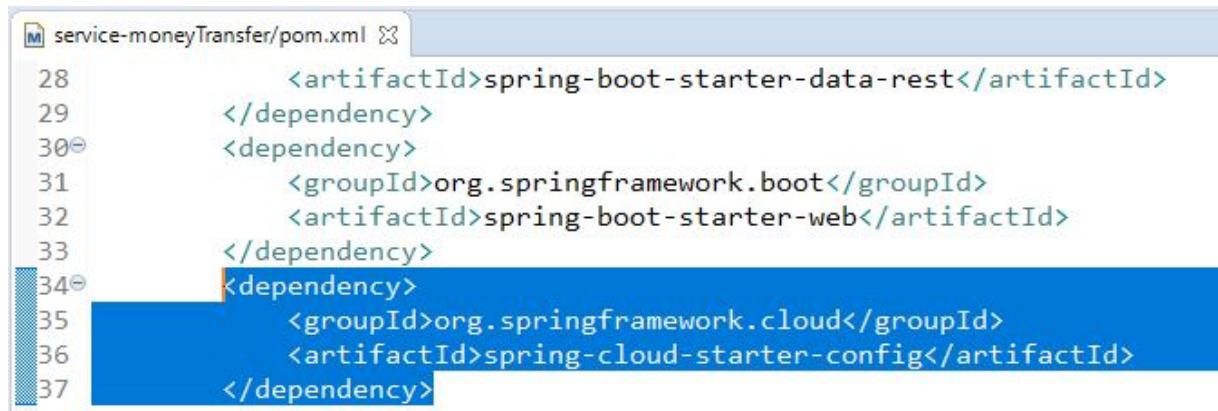
The screenshot shows a browser window with the URL `localhost:8888/service-wePayment/master`. The page displays a JSON configuration file for the `service-wePayment` application. The configuration includes a profile named "master", a database connection with URL `jdbc:mysql://localhost:3306/wePaymentDb?useUnicode=true&useJDBCCompliantTimezoneShift=true&useLegacyDatetimeCode=false&serverTimezone=UTC`, port `8083`, and a second profile section with properties for `application.properties`.

```
{  
  name: "service-wePayment",  
  profiles: [  
    "master"  
  ],  
  label: null,  
  version: "fabbb3fc48042c35878160ee95ea6adc556440e9a",  
  state: null,  
  propertySources: [  
    {  
      name: "file:///C:/Users/MAZER Omar/cloud-config/file:C:/Users/MAZER Omar/cloud-config/service-wePayment.properties",  
      source: {  
        spring.datasource.url: "jdbc:mysql://localhost:3306/wePaymentDb?useUnicode=true&useJDBCCompliantTimezoneShift=true&useLegacyDatetimeCode=false&serverTimezone=UTC",  
        server.port: "8083"  
      }  
    },  
    {  
      name: "file:///C:/Users/MAZER Omar/cloud-config/file:C:/Users/MAZER Omar/cloud-config/application.properties",  
      source: {  
        spring.datasource.username: "root",  
        spring.datasource.password: "",  
        spring.jpa.show-sql: "true",  
        spring.jpa.hibernate.ddl-auto: "update"  
      }  
    }  
  ]  
}
```

Figure 235:Test service-wePayment

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

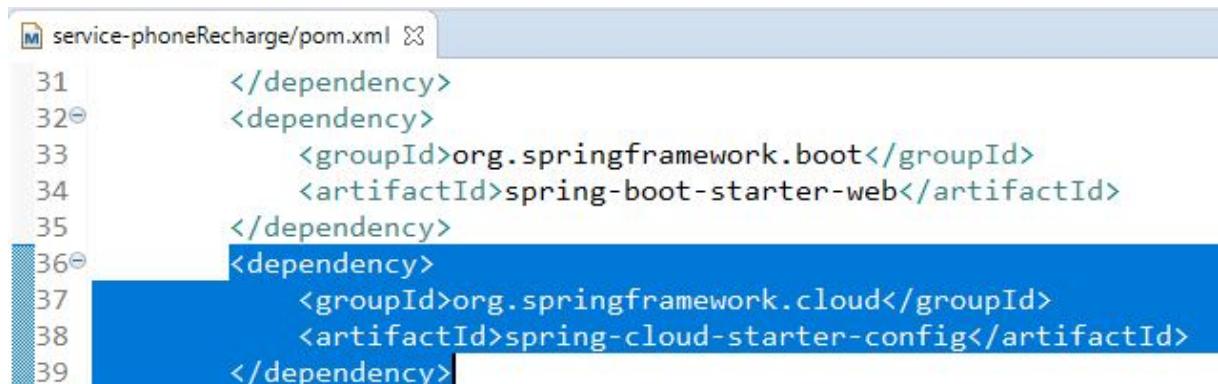
1. **service-moneyTransfer\pom.xml**



```
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

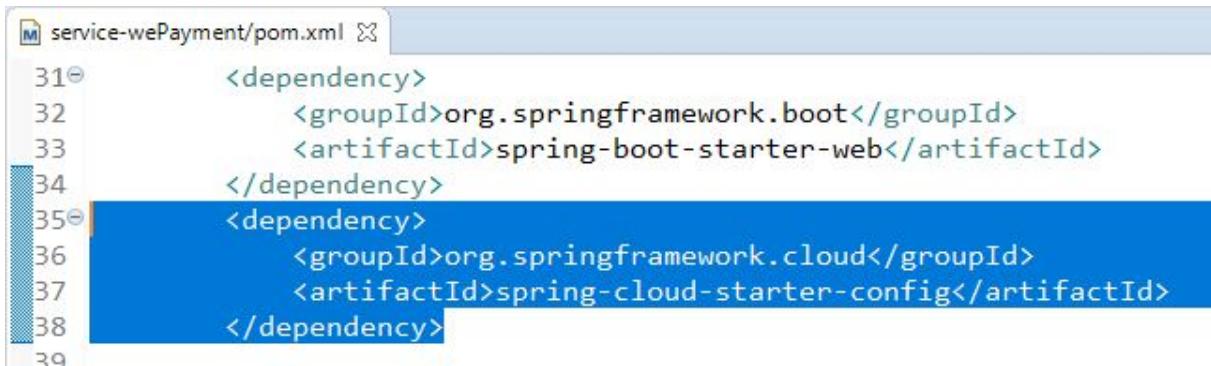
2. **service-phoneRecharge\pom.xml**



```
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**



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

Figure 238:service-wePayment\pom.xml

17. Add **bootstrap.properties** file under the folder **src/main/resources** 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.

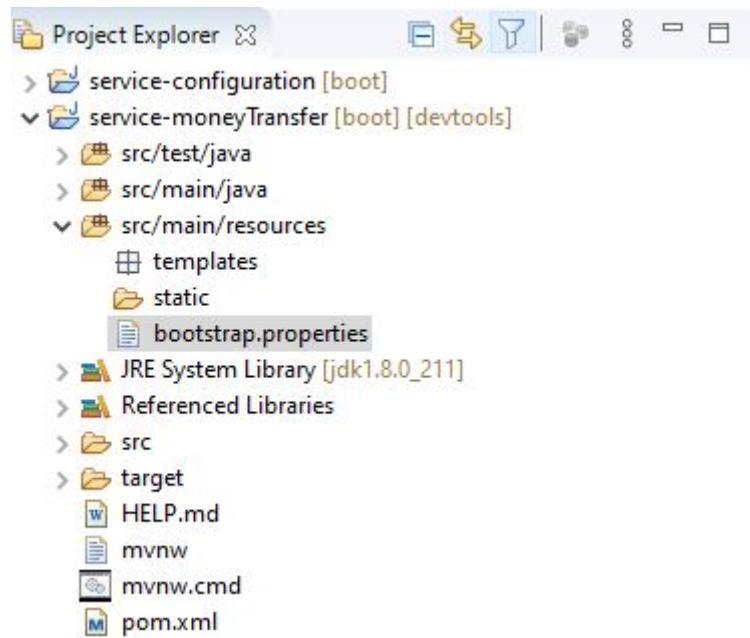


Figure 239: Add bootstrap.properties file under the folder src/main/resources for the three services

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=*



```
1 spring.application.name=service-payment
2 spring.cloud.config.uri=http://localhost:8888
3 spring.cloud.config.profile=production
4 management.endpoints.web.exposure.include=*
```

Figure 240:bootstrap.properties

```
1 spring.application.name=service-recharge
2 spring.cloud.config.uri=http://localhost:8888
3 spring.cloud.config.profile=production
4 management.endpoints.web.exposure.include=*
```

Figure 241:bootstrap.properties service-recharge

```
1 spring.application.name=service-transfer
2 spring.cloud.config.uri=http://localhost:8888
3 spring.cloud.config.profile=production
4 management.endpoints.web.exposure.include=*
```

Figure 242:bootstrap.properties service-transfer

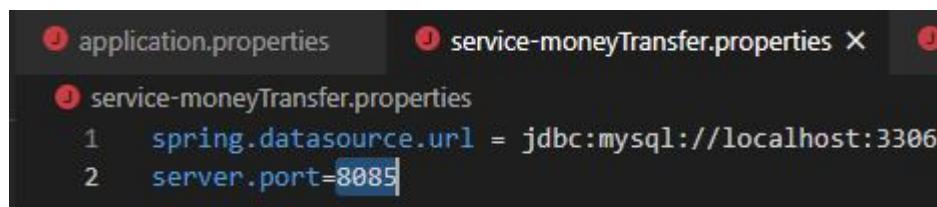
19. Adding the **spring-boot-starter-actuator** dependency under the **pom.xml** for all services

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).

```
19      <spring-cloud.version>2020.0.0</spring-cloud.version>
20  </properties>
21
22  <dependencies>
23    <dependency>
24      <groupId>org.springframework.boot</groupId>
25      <artifactId>spring-boot-starter-actuator</artifactId>
26    </dependency>
```

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

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

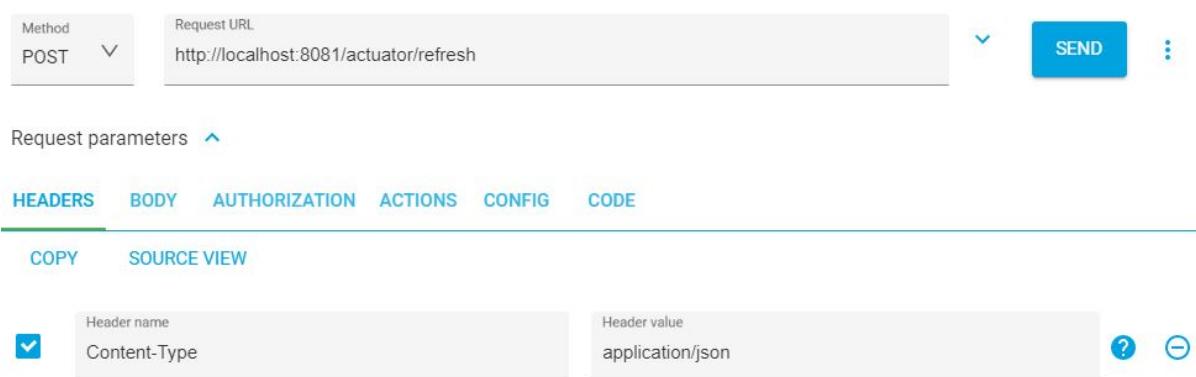


The screenshot shows two files in a file editor:

- application.properties**: Contains the line `spring.datasource.url = jdbc:mysql://localhost:3306`.
- service-moneyTransfer.properties**: Contains the line `server.port=8085`, which is highlighted in blue.

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

21. Test the service actuator refresh in ARC



The screenshot shows the "actuator/refresh" endpoint in the Actuator module of the Application Registry Center (ARC). The request URL is `http://localhost:8081/actuator/refresh`. The method is set to POST. The "HEADERS" tab is selected, showing a single header entry: Content-Type with a value of application/json. The "SEND" button is visible at the top right.

Figure 245: Test the service actuator refresh in ARC

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

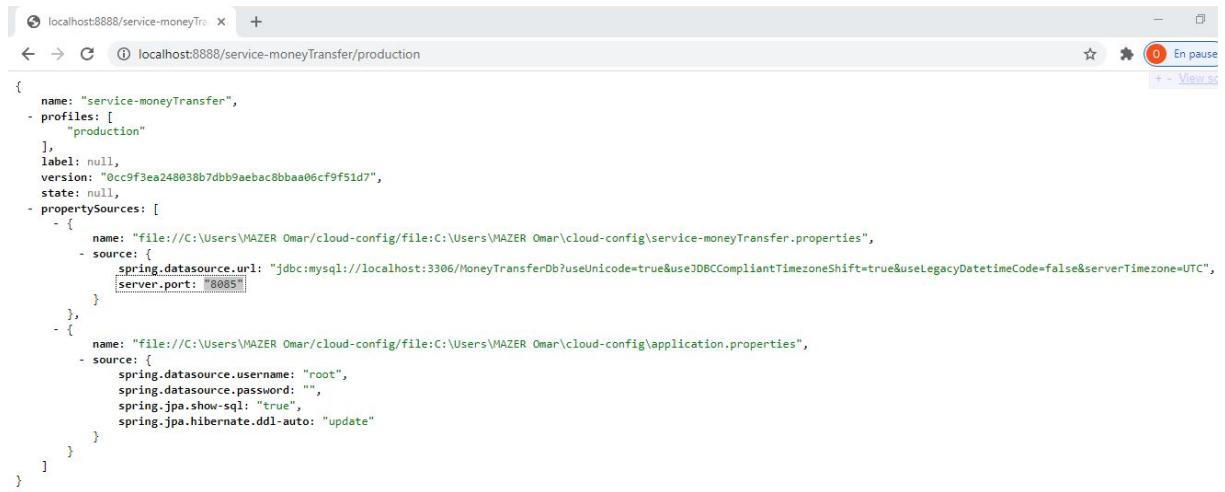


The screenshot shows the response details for a 200 OK status message. The response time is 1708.67 ms. The "DATA TABLE" tab is selected, displaying the following JSON array:

```
[Array[3]
 0: "config.client.version",
 1: "server.port",
 2: "spring.cloud.bootstrap.enabled"
]
```

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



```
{\n    name: "service-moneyTransfer",\n    profiles: [\n        "production"\n    ],\n    label: null,\n    version: "0cc9f3ea248038b7dbb9aebac8bbba06cf9f51d7",\n    state: null,\n    propertySources: [\n        {\n            name: "file:///C:/Users/MAZER_Omar/cloud-config/file:C:/Users/MAZER_Omar/cloud-config/service-moneyTransfer.properties",\n            source: {\n                spring.datasource.url: "jdbc:mysql://localhost:3306/MoneyTransferDb?useUnicode=true&useJDBCCompliantTimezoneShift=true&useLegacyDatetimeCode=false&serverTimezone=UTC",\n                server.port: "8085"\n            }\n        },\n        {\n            name: "file:///C:/Users/MAZER_Omar/cloud-config/file:C:/Users/MAZER_Omar/cloud-config/application.properties",\n            source: {\n                spring.datasource.username: "root",\n                spring.datasource.password: "",\n                spring.jpa.show-sql: "true",\n                spring.jpa.hibernate.ddl-auto: "update"\n            }\n        }\n    ]\n}
```

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

4.1.6 Registration 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:

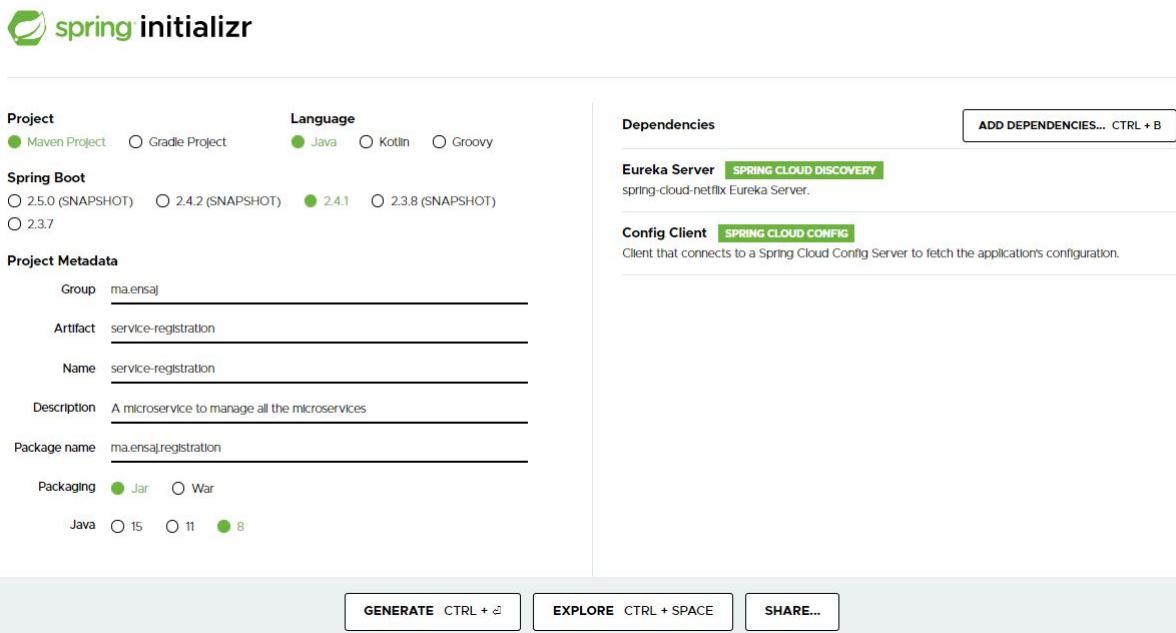


Figure 248:Spring Initializr

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

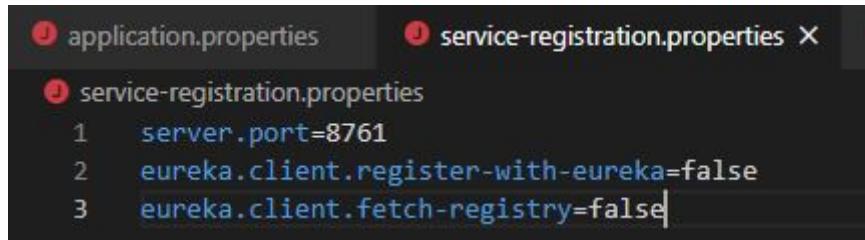
Figure 249:@EnableConfigServer annotation

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**



```
application.properties      service-registration.properties X
service-registration.properties
1 server.port=8761
2 eureka.client.register-with-eureka=false
3 eureka.client.fetch-registry=false|
```

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

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



```
bootstrap.properties
1 spring.application.name=service-registration
2 spring.cloud.config.uri=http://localhost:8888
3 spring.cloud.config.profile=production
```

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

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

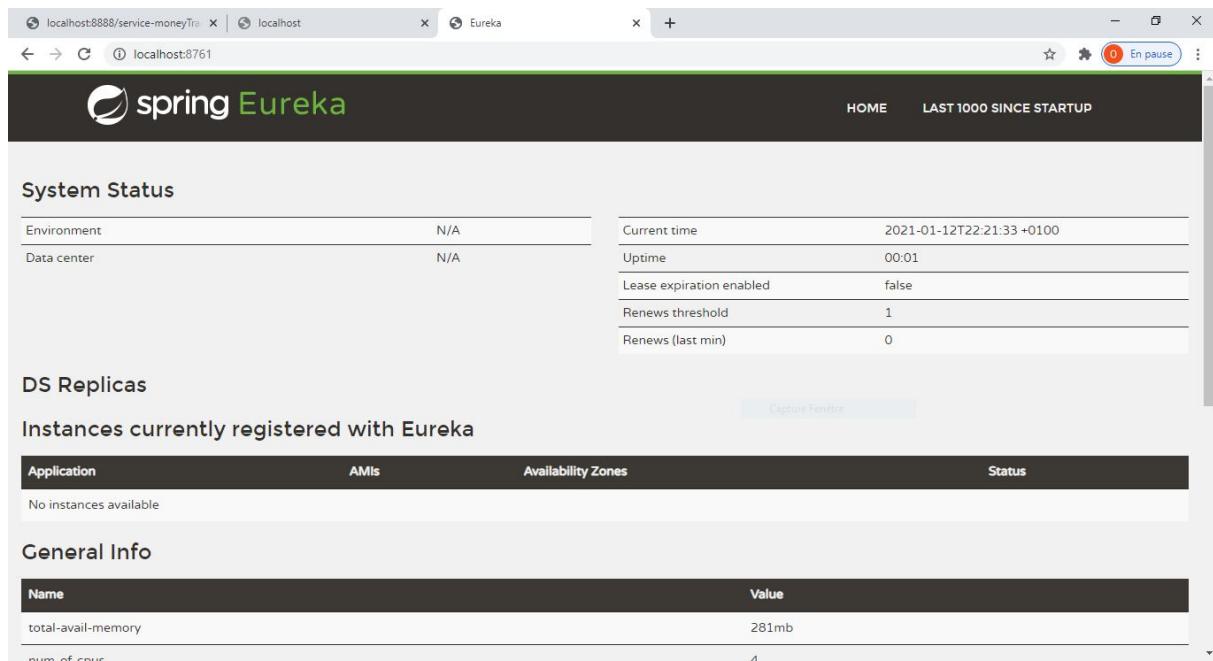


Figure 253:We go to the browser and we tap in the url the address localhost:8761.And we get the page of spring Eureka.

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

```

1 package ma.ensaj.moneyTransfer;
2
3 import org.springframework.boot.SpringApplication;
4
5 @EnableDiscoveryClient
6 @SpringBootApplication
7 public class MoneyTransferApplication {
8
9     public static void main(String[] args) {
10         SpringApplication.run(MoneyTransferApplication.class, args);
11     }
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

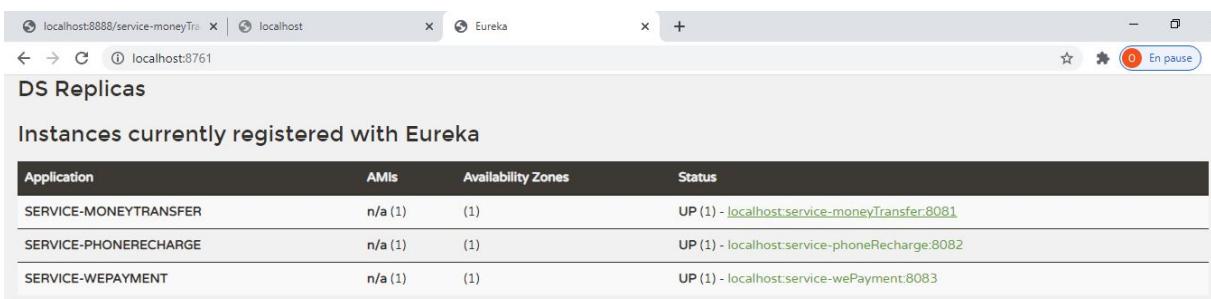


Figure 256:Spring Eureka

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:

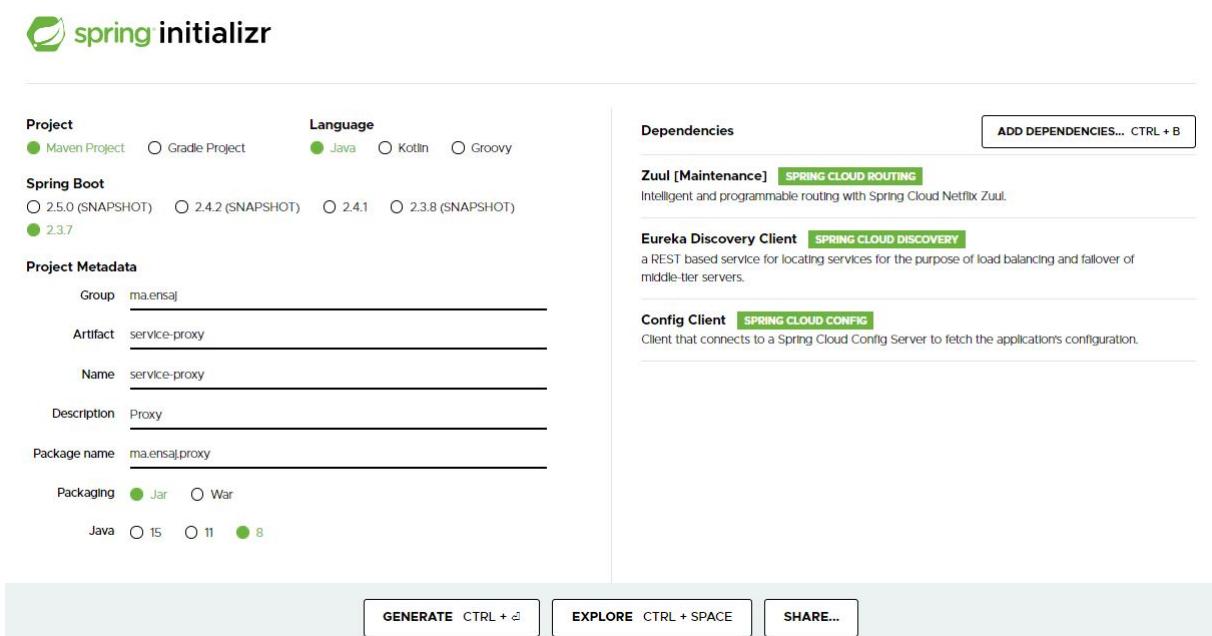


Figure 257:Spring Initializr

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.

```

1 package ma.ensaj.proxy;
2
3+ import org.springframework.boot.SpringApplication;
4
5 @EnableZuulProxy
6 @SpringBootApplication
7 public class ServiceProxyApplication {
8
9
10+ public static void main(String[] args) {
11     SpringApplication.run(ServiceProxyApplication.class, args);
12 }
13
14 }

```

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

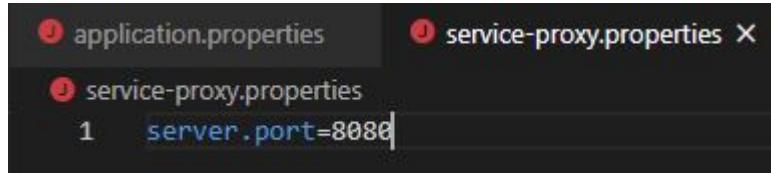


Figure 260:We set the port number to 8080

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**.



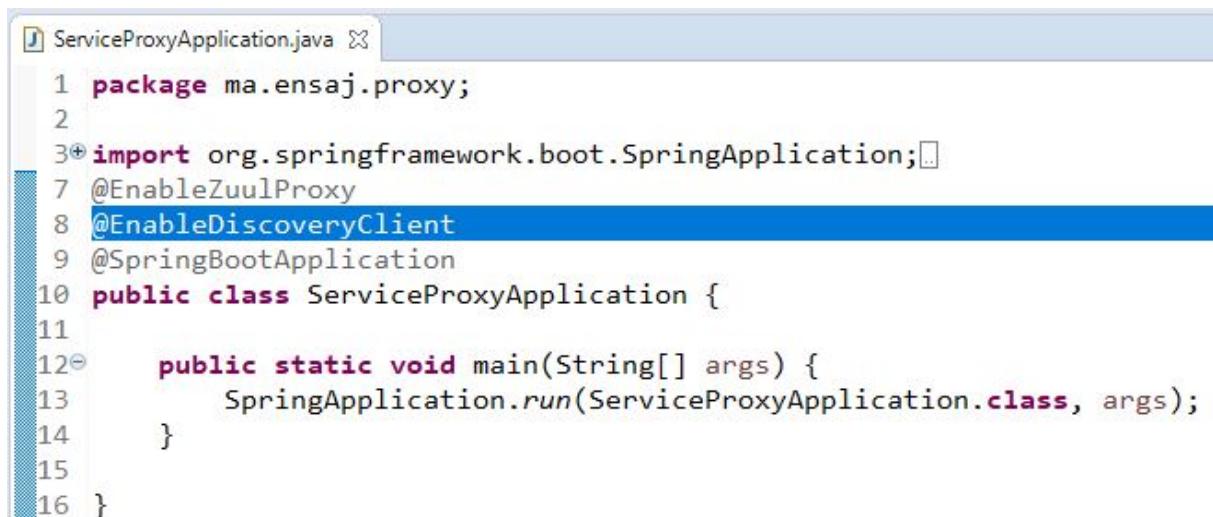
```
bootstrap.properties
1 spring.application.name=service-proxy
2 spring.cloud.config.uri=http://localhost:8888
3 spring.cloud.config.profile=production
```

A screenshot of a code editor window titled "bootstrap.properties". The window contains three lines of configuration properties. Line 1 sets the application name to "service-proxy". Line 2 sets the Spring Cloud Config URI to "http://localhost:8888". Line 3 sets the Spring Cloud Config profile to "production". The code is color-coded, with "spring." in blue, ".application.name" in green, "service-proxy" in brown, ".cloud.config.uri" in green, "http://localhost:8888" in brown, ".cloud.config.profile" in green, and "production" in brown.

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.

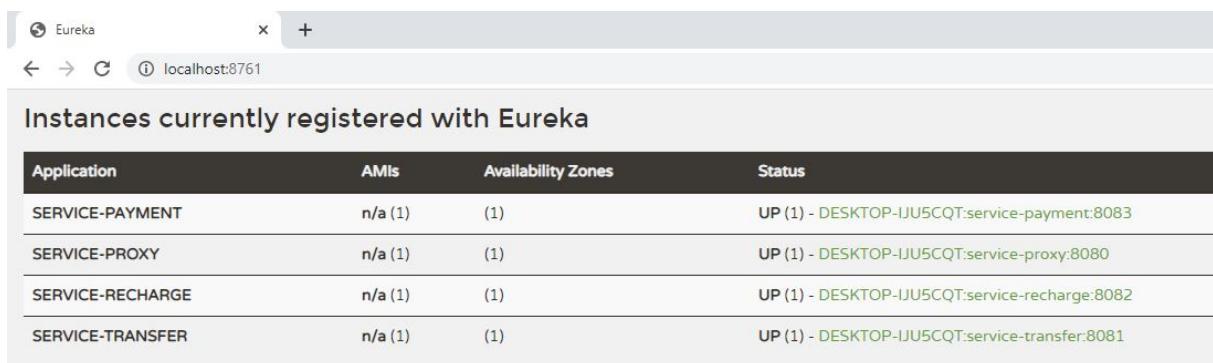


```

1 package ma.ensaj.proxy;
2
3 import org.springframework.boot.SpringApplication;
4
5 @EnableZuulProxy
6 @EnableDiscoveryClient
7 @SpringBootApplication
8 public class ServiceProxyApplication {
9
10     public static void main(String[] args) {
11         SpringApplication.run(ServiceProxyApplication.class, args);
12     }
13
14 }
15
16 }
```

Figure 263:@EnableDiscoveryClient

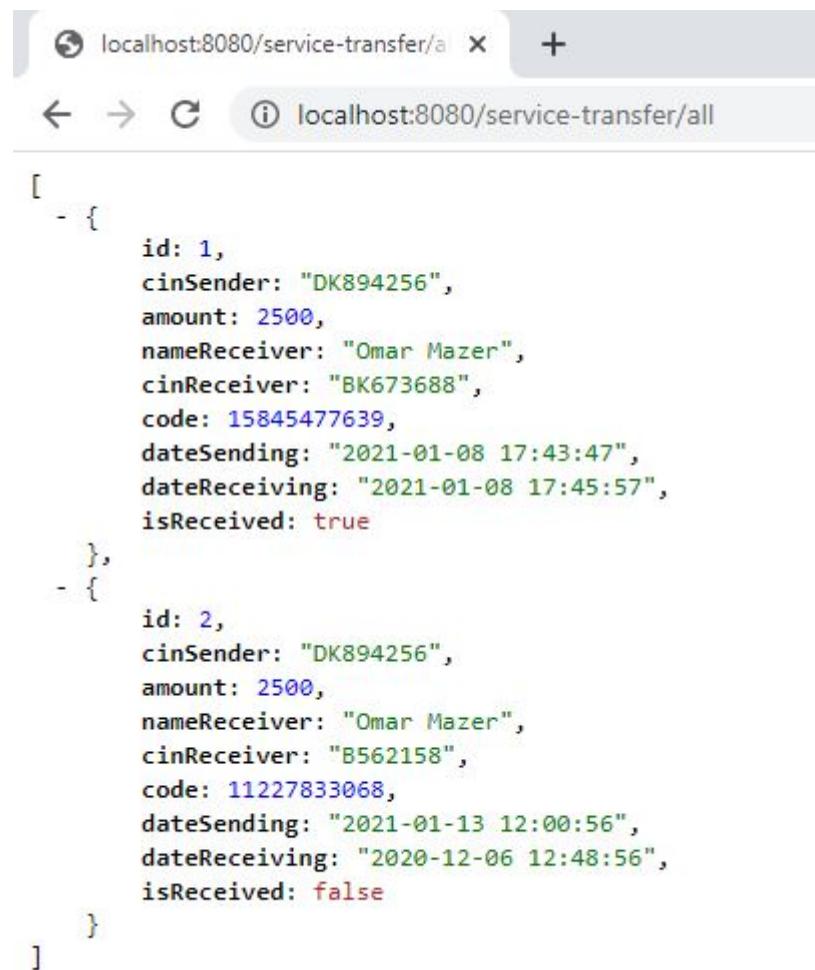
8. We notice now at **Spring Eureka** that all services are now running in their pre-specified port's numbers and **service-proxy** is well added.



Application	AMIs	Availability Zones	Status
SERVICE-PAYMENT	n/a (1)	(1)	UP (1) - DESKTOP-IJU5CQT:service-payment:8083
SERVICE-PROXY	n/a (1)	(1)	UP (1) - DESKTOP-IJU5CQT:service-proxy:8080
SERVICE-RECHARGE	n/a (1)	(1)	UP (1) - DESKTOP-IJU5CQT:service-recharge:8082
SERVICE-TRANSFER	n/a (1)	(1)	UP (1) - DESKTOP-IJU5CQT:service-transfer:8081

Figure 264:Spring Eureka

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

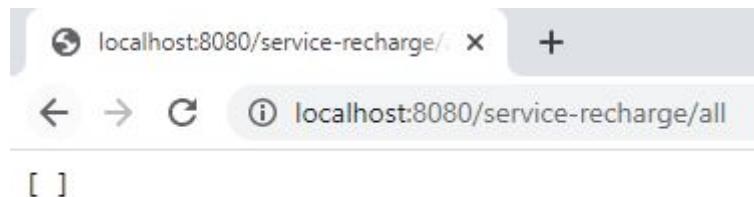


A screenshot of a web browser window. The address bar shows the URL `localhost:8080/service-transfer/all`. The page content displays a JSON array of two objects representing money transfers. Each transfer has an ID, a sender CIN, an amount, receiver details, a code, and sending/receiving dates.

```
[  
  - {  
      id: 1,  
      cinSender: "DK894256",  
      amount: 2500,  
      nameReceiver: "Omar Mazer",  
      cinReceiver: "BK673688",  
      code: 15845477639,  
      dateSending: "2021-01-08 17:43:47",  
      dateReceiving: "2021-01-08 17:45:57",  
      isReceived: true  
    },  
    - {  
      id: 2,  
      cinSender: "DK894256",  
      amount: 2500,  
      nameReceiver: "Omar Mazer",  
      cinReceiver: "B562158",  
      code: 11227833068,  
      dateSending: "2021-01-13 12:00:56",  
      dateReceiving: "2020-12-06 12:48:56",  
      isReceived: false  
    }  
]
```

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

10. Test service-recharge. The list is empty



A screenshot of a web browser window. The address bar shows the URL `localhost:8080/service-recharge/all`. The page content displays an empty JSON array, indicated by the character [followed by a space and a closing bracket].

```
[ ]
```

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

11. Test service-payment. The list is empty

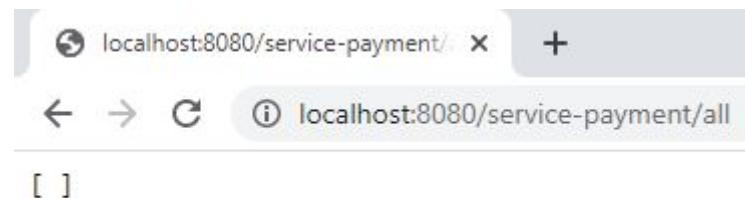


Figure 267:Test service-payment. The list is empty

4.2 Frontend

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

4.2.1 Login Page



```
login.html
1  <!DOCTYPE html>
2  <html>
3  <head>
4      <meta charset="ISO-8859-1">
5      <title>Login</title>
6      <link rel="stylesheet" th:href="@{/css/bootstrap.min.css}" type="text/css">
7      <link rel="stylesheet" th:href="@{/css/custom.css}" type="text/css">
8  </head>
9
10 <body style="background-image: url('/img/bgimg.PNG')">
11     <nav class="navbar navbar-expand-lg navbar-light bg-light mb-4 shadow-sm ">
12         <a class="navbar-brand" th:href="@{/}">TASHILATE-ENSAJ</a>
13     </nav>
14     <div class="col-lg-4 formStyle p-4">
15         <h1> Login </h1>
16         <form th:action="@{/login}" method="post">
17             <!-- error message -->
18             <div th:if="${param.error}">
19                 <div class="alert alert-danger">Invalid username or
20                     password.</div>
21             </div>
22             <!-- logout message -->
23             <div th:if="${param.logout}">
24                 <div class="alert alert-info">You have been logged out.</div>
25             </div>
26             <div class="form-group">
27                 <label for="username"> Username </label> :
28                 <input type="text" class="form-control" id="username" name="username" placeholder="Enter Email ID"
29                     autofocus="autofocus">
30             </div>
31             <div class="form-group">
32                 <label for="password"> Password </label>: <input type="password" id="password" name="password"
33                     class="form-control" placeholder="Enter Password" />
34             </div>
35             <div class="form-group">
36                 <button type="submit" name="Login-submit" id="login-submit"
37                     class="btn btn-outline-success btn-block mb-4" value="Log In">Login</button>
38             </div>
39         </form>
40         <div class="form-group">
41             <span>New user? <a href="/" th:href="@{/registration}">Register
42                 here</a></span>
43         </div>
44     </div>
45 ..
```

Figure 268:login.html

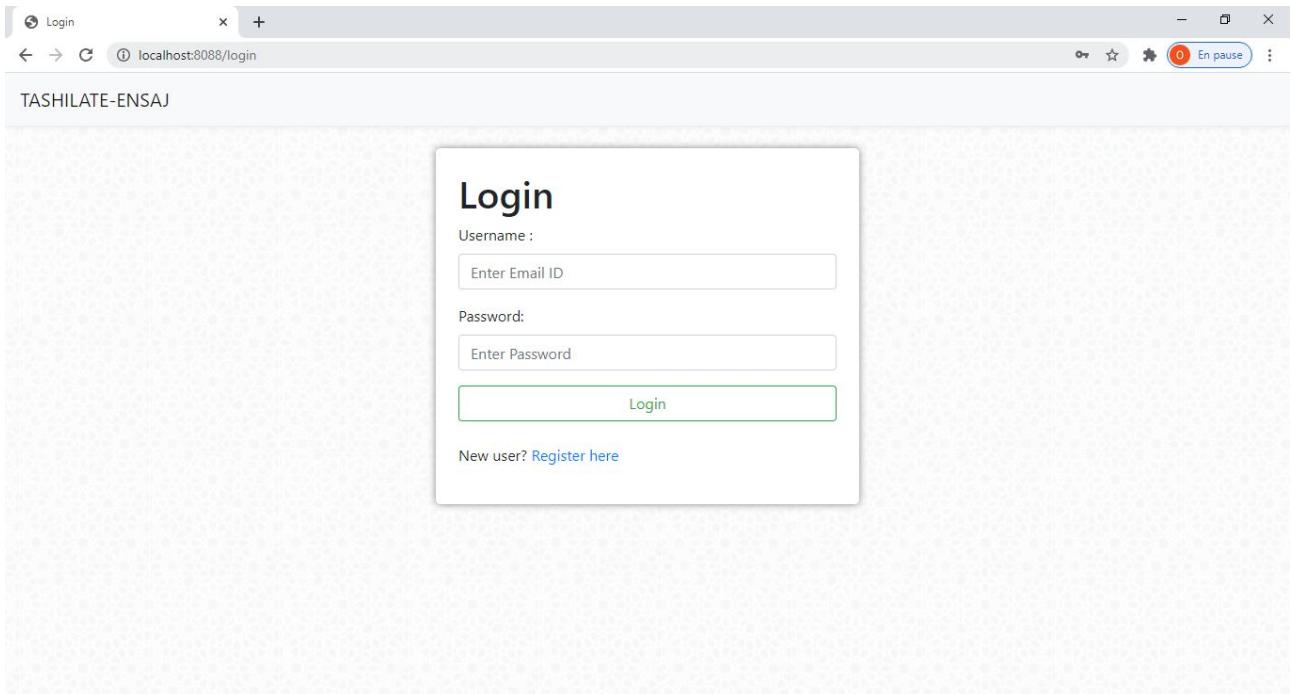
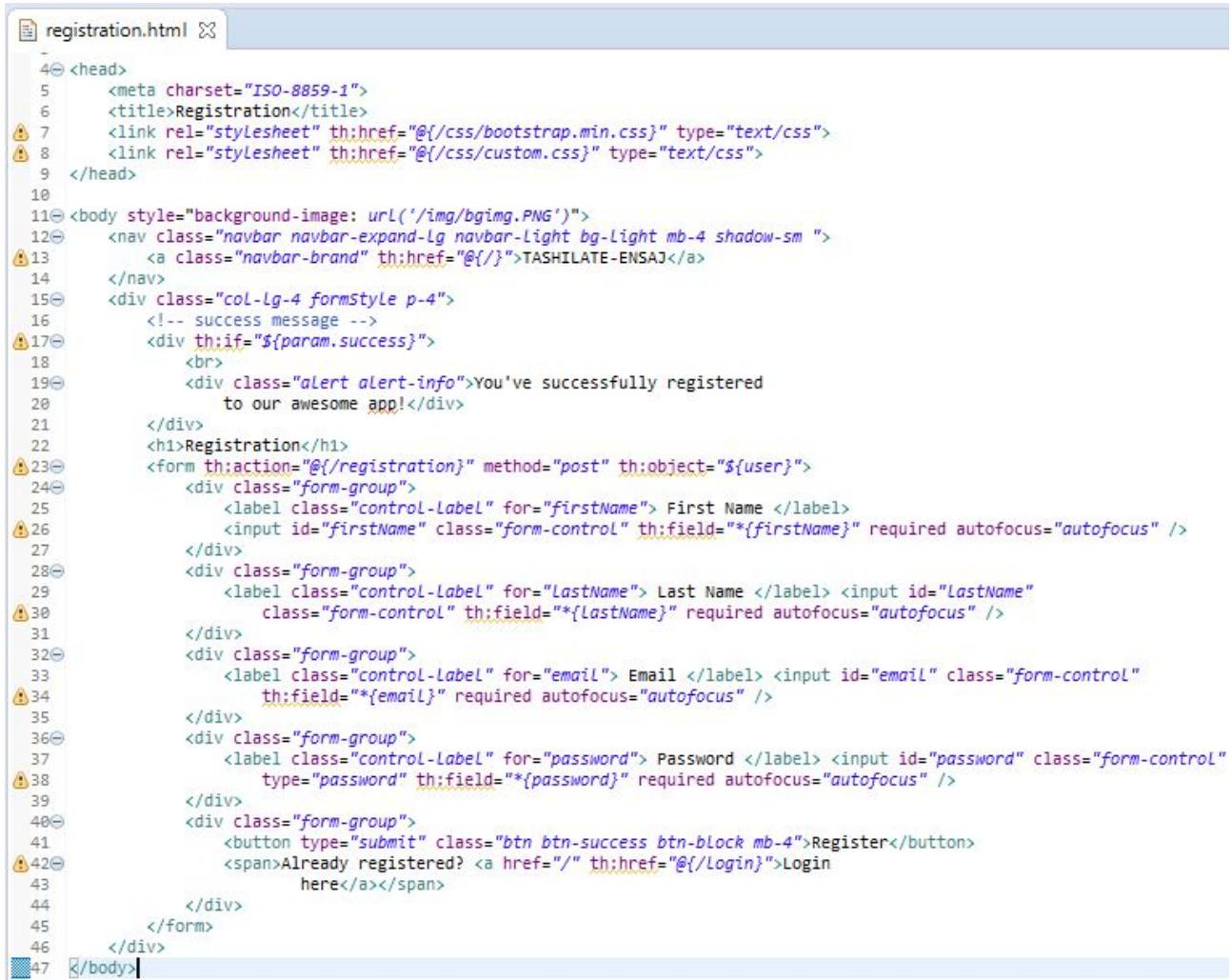


Figure 269:login page in the browser

4.2.2 Registration



The screenshot shows a code editor window with the file 'registration.html' open. The code is a JSP page for user registration. It includes a header with meta tags, a navigation bar, and a success message section. The main form contains fields for First Name, Last Name, Email, and Password, each with its own validation group. A 'Register' button is provided, along with a link to log in if the user is already registered.

```
4④ <head>
5      <meta charset="ISO-8859-1">
6      <title>Registration</title>
7      <link rel="stylesheet" th:href="@{/css/bootstrap.min.css}" type="text/css">
8      <link rel="stylesheet" th:href="@{/css/custom.css}" type="text/css">
9  </head>
10
11④ <body style="background-image: url('/img/bgimg.PNG')">
12④   <nav class="navbar navbar-expand-lg navbar-light bg-light mb-4 shadow-sm ">
13     <a class="navbar-brand" th:href="@{}/>TASHILATE-ENSAJ</a>
14   </nav>
15④   <div class="col-lg-4 formStyle p-4">
16     <!-- success message -->
17④     <div th:if="${param.success}">
18       <br>
19       <div class="alert alert-info">You've successfully registered
20         to our awesome app!</div>
21     </div>
22     <h1>Registration</h1>
23④     <form th:action="@{/registration}" method="post" th:object="${user}">
24④       <div class="form-group">
25         <label class="control-label" for="firstName"> First Name </label>
26         <input id="firstName" class="form-control" th:field="*{firstName}" required autofocus="autofocus" />
27       </div>
28④       <div class="form-group">
29         <label class="control-label" for="LastName"> Last Name </label> <input id="LastName"
30           class="form-control" th:field="*{lastName}" required autofocus="autofocus" />
31       </div>
32④       <div class="form-group">
33         <label class="control-label" for="email"> Email </label> <input id="email" class="form-control"
34           th:field="*{email}" required autofocus="autofocus" />
35       </div>
36④       <div class="form-group">
37         <label class="control-label" for="password"> Password </label> <input id="password" class="form-control"
38           type="password" th:field="*{password}" required autofocus="autofocus" />
39       </div>
40④       <div class="form-group">
41         <button type="submit" class="btn btn-success btn-block mb-4">Register</button>
42         <span>Already registered? <a href="/" th:href="@{/login}">Login
43           here</a></span>
44       </div>
45     </form>
46   </div>
47 </body>
```

Figure 270:registration.html

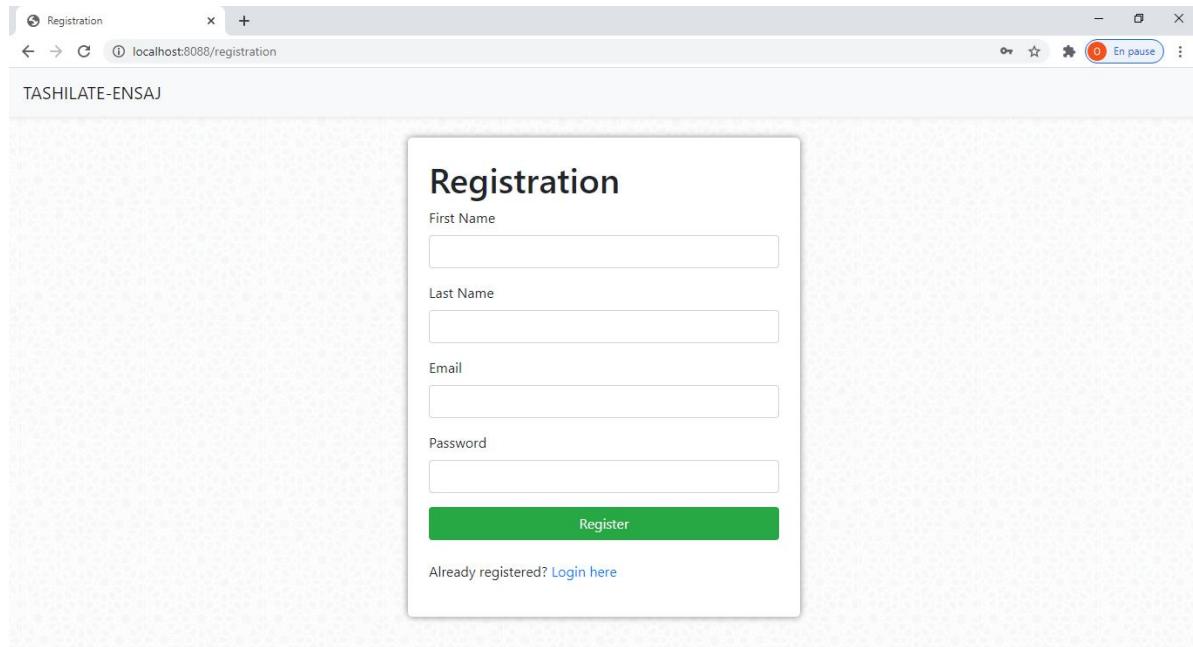


Figure 271:registration page in the browser

4.2.3 Index page

```
index.html ✘
1  <!DOCTYPE html>
2  
```

Figure 272:index.html

```
index.html ✘
20          </li>
21      </ul>
22  </nav>
23  <br><br><br><br>|
24
25@   <div class="row d-flex justify-content-center">
26@     <div class="col-sm-12 col-lg-3 col-md-6 mb-5">
27@       <a th:href="@{/money_transfer}">
28@         <div class="card p-4 cardStyle">
29          
30@          <div class="card-body">
31            <h4 class="card-title text-center">Money Transfer</h4>
32          </div>
33        </div>
34      </a>
35    </div>
36@   <div class="col-sm-12 col-lg-3 col-md-6 mb-5">
37@     <a th:href="@{/phone_recharge}">
38@       <div class="card p-4 cardStyle">
39          
40@          <div class="card-body">
41            <h4 class="card-title text-center">Phone Recharge</h4>
42          </div>
43        </div>
44      </a>
45    </div>
46@   <div class="col-sm-12 col-lg-3 col-md-6 mb-5">
47@     <a th:href="@{/we_payment}">
48@       <div class="card p-4 cardStyle">
49          
50@          <div class="card-body">
51            <h4 class="card-title text-center">W/E Payment</h4>
52          </div>
53        </div>
54      </a>
55    </div>
56  </div>
57 </body>
58 </html>
```

Figure 273:index.html continuation code

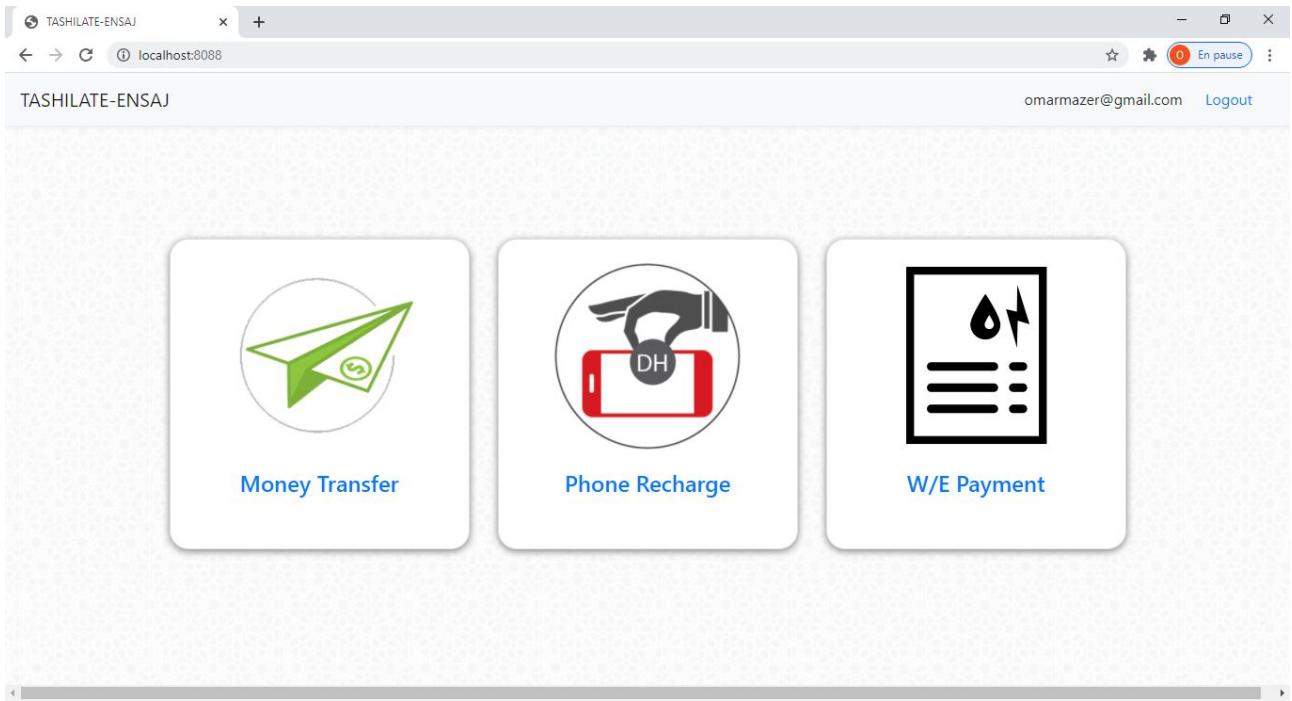


Figure 274:index.html in the browser

4.2.4 Money Transfer

```
money_transfer.html
1 <!DOCTYPE html>
2<html xmlns:th="http://www.thymeleaf.org" xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-springsecurity3">
3
4<head>
5     <meta charset="ISO-8859-1">
6     <title>TASHILATE-ENSAJ</title>
7
8     <link rel="stylesheet" th:href="@{/css/bootstrap.min.css}" type="text/css">
9     <link rel="stylesheet" th:href="@{/css/custom.css}" type="text/css">
10    </head>
11<body style="background-image: url('/img/bgimg.PNG')>
12    <nav class="navbar navbar-expand-lg navbar-light bg-light mb-4 shadow-sm ">
13        <a class="navbar-brand" th:href="@{/}''>TASHILATE-ENSAJ</a>
14        <ul class="navbar-nav ml-auto">
15            <li class="nav-item">
16                <span sec:authentication="principal.username"> User</span>
17            </li>
18            <li sec:authorize="isAuthenticated()" class="nav-item ml-4 mr-4">
19                <a th:href="@{/Logout}''>Logout</a>
20            </li>
21        </ul>
22    </nav>
23    <br><br><br><br>
24    <div class="row d-flex justify-content-center">
25        <div class="col-sm-12 col-lg-3 col-md-6 mb-5">
26            <a th:href="@{/money_transfer/send_money}''>
27                <div class="card p-4 cardStyle">
28                    
29                    <div class="card-body">
30                        <h4 class="card-title text-center">Send Money</h4>
31                    </div>
32                </div>
33            </a>
34        </div>
35        <div class="col-sm-12 col-lg-3 col-md-6 mb-5">
36            <a th:href="@{/money_transfer/receive_money}''>
37                <div class="card p-4 cardstyle">
38                    
39                    <div class="card-body">
40                        <h4 class="card-title text-center">Receive Money</h4>
41                    </div>
42                </div>
43            </a>
44        </div>
45    </div>
```

Figure 275:money_transfer.html

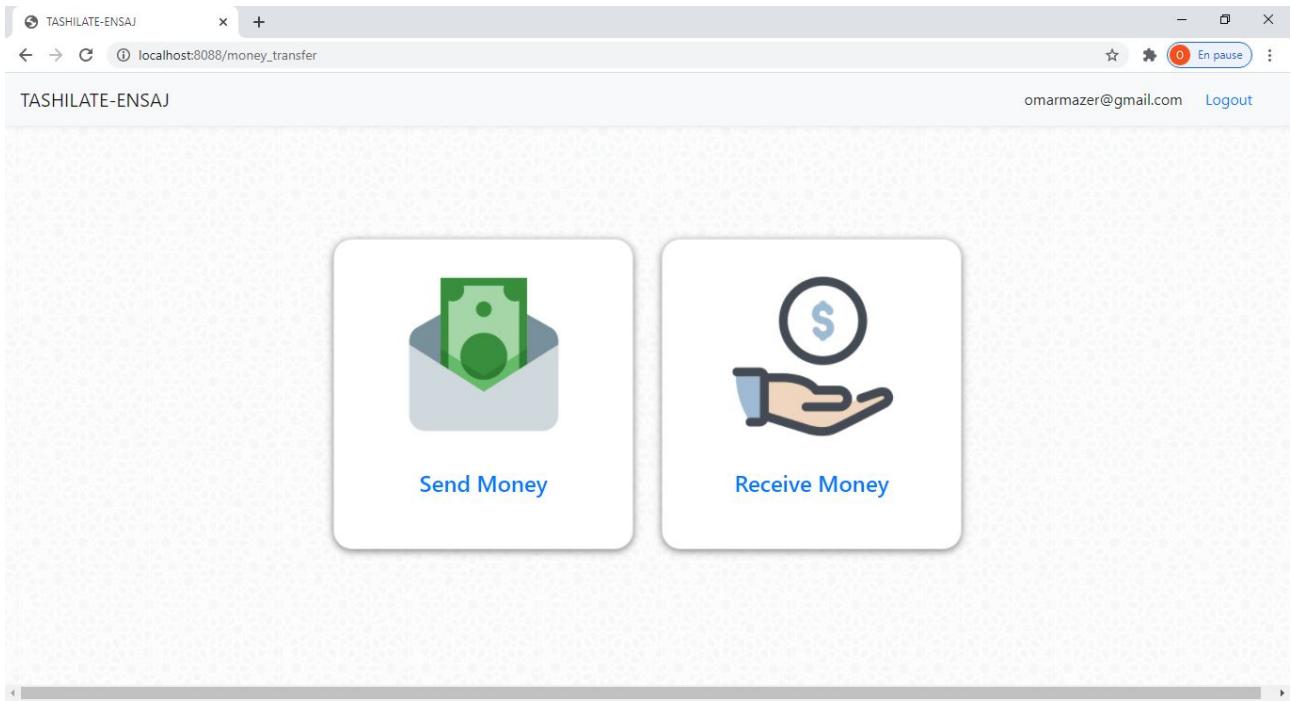


Figure 276:money transfer service in the browser

Send Money

```

send_money.html
3④ <head>
4      <meta charset="ISO-8859-1">
5      <title>TASHILATE-ENSAJ</title>
6      <link rel="stylesheet" th:href="@{/css/bootstrap.min.css}" type="text/css">
7      <link rel="stylesheet" th:href="@{/css/custom.css}" type="text/css">
8  </head>
9④ <body style="background-image: url('/img/bgimg.PNG')">
10④   <nav class="navbar navbar-expand-lg navbar-light bg-light mb-4 shadow-sm ">
11     <a class="navbar-brand" th:href="@{}/>TASHILATE-ENSAJ</a>
12④   <ul class="navbar-nav ml-auto">
13④     <li class="nav-item">
14       <span sec:authentication="principal.username"> User</span>
15     </li>
16④     <li sec:authorize="isAuthenticated()" class="nav-item ml-4 mr-4">
17       <a th:href="/Logout">Logout</a>
18     </li>
19   </ul>
20 </nav>
21④ <div class="col-lg-4 formStyle p-4" id="form">
22   <h1> Send Money </h1>
23④   <div class="form-group">
24     <label for="cinSender">The CIN of the sender </label> :
25     <input type="text" class="form-control" id="cinSender" name="cinSender" autofocus="autofocus" required>
26   </div>
27④   <div class="form-group">
28     <label for="nameReceiver">The name of the receiver</label>;
29     <input type="text" id="nameReceiver" name="nameReceiver" class="form-control" required/>
30   </div>
31④   <div class="form-group">
32     <label for="amount">The amount of money to send</label>;
33④     <div class="input-group">
34       <input type="text" class="form-control" id="amount" name="amount" required/>
35④     <div class="input-group-append">
36       <span class="input-group-text">DH</span>
37     </div>
38   </div>
39   <div class="form-group">
40     <button type="submit" name="send" id="send" class="btn btn-outline-success btn-block mb-4" value="Send">Send</button>
41   </div>
42 </div>
43   <div class="col-lg-4 formstyle p-4 text-center" id="code" >
44   </div>
45   <div>
46     <script src="/js/jquery-3.3.1.min.js"></script>
47     <script src="/js/send_money.js"></script>

```

Figure 277:send_money.html

```

send_money.js
1 $(document).ready(function(){
2     var cinSender = $("#cinSender");
3     var nameReceiver = $("#nameReceiver");
4     var amount = $("#amount");
5     $("#code").hide();
6     $('#send').click(function() {
7         $.ajax({
8             url: 'http://localhost:8080/service-transfer/sendMoney',
9             data: JSON.stringify({cinSender: cinSender.val(), nameReceiver: nameReceiver.val(), amount: amount.val()}),
10            headers: { 'Access-Control-Allow-Origin': '*' },
11            type: 'POST',
12            dataType: "json",
13            contentType:"application/json; charset=utf-8",
14            async: false,
15            success: function(data, textStatus, jqXHR) {
16                remplir(data);
17                if (!("Notification" in window)) {
18                    alert("This browser does not support desktop notification");
19                }
20                else if (Notification.permission === "granted") {
21                    var notification = new Notification("MONEY SENT SUCCESSFULLY!");
22                }
23                else if (Notification.permission !== "denied") {
24                    Notification.requestPermission().then(function (permission) {
25                        if (permission === "granted") {
26                            var notification = new Notification("MONEY SENT SUCCESSFULLY!");
27                        }
28                    });
29                }
30            },
31            error: function(jqXHR, textStatus, errorThrown) {
32                console.log(textStatus);
33            }
34        });
35    });
36    function remplir(data) {
37        var ligne="";
38        ligne+="

```

Figure 278:send_money.html continuation code

The CIN of the sender :
GB2615

The name of the receiver:
Mazer Omar

The amount of money to send:
2500 DH

Send

Figure 279:Fill the form of send money

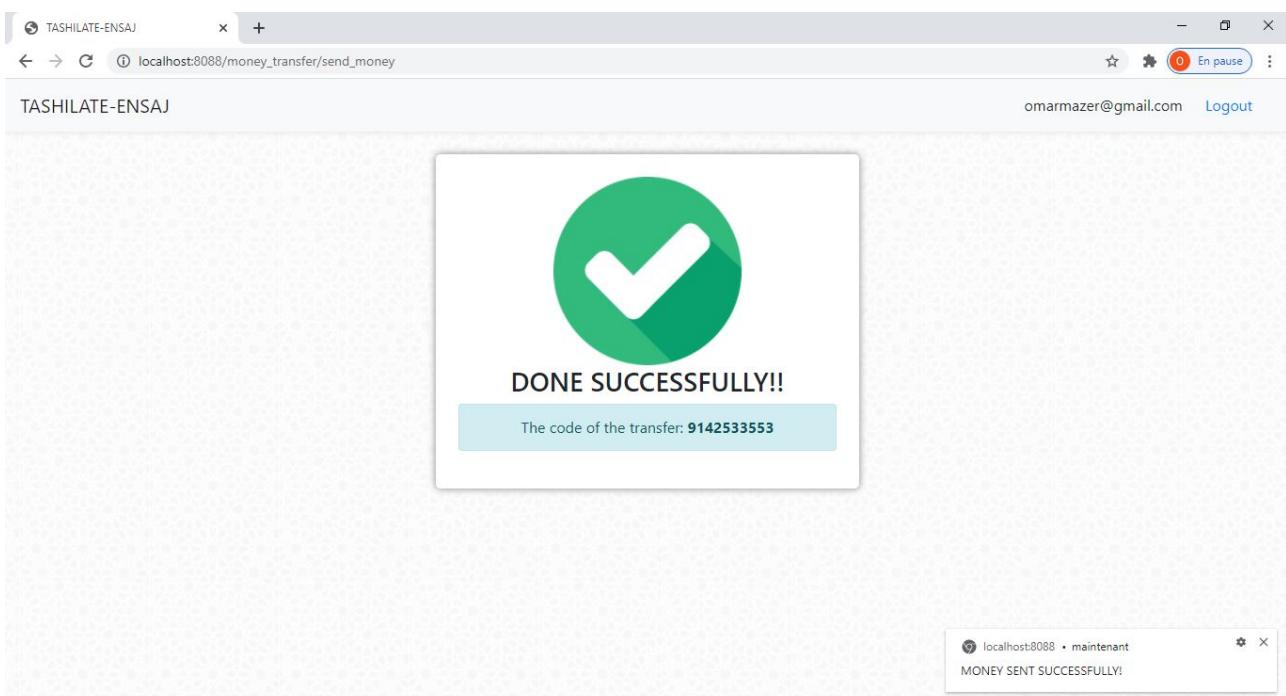


Figure 280:Generation of the code of the transfer

Receive Money

```
receive_money.html
1 <!DOCTYPE html>
2@<html xmlns:th="http://www.thymeleaf.org" xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-springsecurity3">
3@<head>
4    <meta charset="ISO-8859-1">
5    <title>TASHILATE-ENSAI</title>
6    <link rel="stylesheet" th:href="@{/css/bootstrap.min.css}" type="text/css">
7    <link rel="stylesheet" th:href="@{/css/custom.css}" type="text/css">
8 </head>
9@<body style="background-image: url('/img/bgimg.PNG')">
10@    <nav class="navbar navbar-expand-lg navbar-light bg-light mb-4 shadow-sm">
11        <a class="navbar-brand" th:href="@{}>TASHILATE-ENSAI</a>
12@        <ul class="navbar-nav ml-auto">
13@            <li class="nav-item">
14                <span sec:authentication="principal.username"> User</span>
15            </li>
16@            <li sec:authorize="isAuthenticated()" class="nav-item ml-4 mr-4">
17                <a th:href="@{/Logout}>Logout</a>
18            </li>
19        </ul>
20    </nav>
21@    <div class="col-lg-4 formstyle p-4" id="form">
22        <h1> Receive Money </h1>
23@        <div class="form-group">
24            <label for="code">The code of the transfer</label>;
25            <input type="text" id="code" name="code" class="form-control" autofocus="autofocus" required>
26        </div>
27@        <div class="form-group">
28            <label for="cinReceiver">The CIN of the receiver </label> :
29            <input type="text" class="form-control" id="cinReceiver" name="cinReceiver" required>
30        </div>
31@        <div class="form-group">
32            <button type="submit" name="receive" id="receive" class="btn btn-outline-success btn-block mb-4" value="Receive">Receive</button>
33        </div>
34@        <div id="erreur">
35            </div>
36        </div>
37@        <div class="col-lg-4 formstyle p-4 text-center" id="result" >
38        </div>
39        <script src="/js/jquery-3.3.1.min.js"></script>
40        <script src="/js/receive_money.js"></script>
41    </div>
42 </body>
```

Figure 281:receive_money.html

```
receive_money.js ✘
1 $(document).ready(function(){
2     var code = $("#code");
3     var cinReceiver = $("#cinReceiver");
4     $("#result").hide();
5     $("#erreur").hide();
6     $("#receive").click(function() {
7         $.ajax({
8             url: 'http://localhost:8080/service-transfer/receiveMoney',
9             data: JSON.stringify({cinReceiver: cinReceiver.val(), code: code.val()}),
10            headers: { 'Access-Control-Allow-Origin': '*' },
11            type: 'POST',
12            dataType: "json",
13            contentType:"application/json; charset=utf-8",
14            async: false,
15            success: function(data, textStatus, jqXHR) {
16                remplir(data);
17                if (!("Notification" in window)) {
18                    alert("This browser does not support desktop notification");
19                }
20                else if (Notification.permission === "granted") {
21                    var notification = new Notification("MONEY RECEIVED SUCCESSFULLY!");
22                }
23                else if (Notification.permission !== "denied") {
24                    Notification.requestPermission().then(function (permission) {
25                        if (permission === "granted") {
26                            var notification = new Notification("MONEY RECEIVED SUCCESSFULLY!");
27                        }
28                    });
29                }
30            },
31            error: function(jqXHR, textStatus, errorThrown) {
32                console.log(textStatus);
33                remplirErreurs();
34            }
35        });
36    });
37});
```

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+="
```

Figure 283:receive_money.js code continuation

The code of the transfer:
9142533553

The CIN of the receiver :
BK682153

Receive

Figure 284:Fill the form of receive money

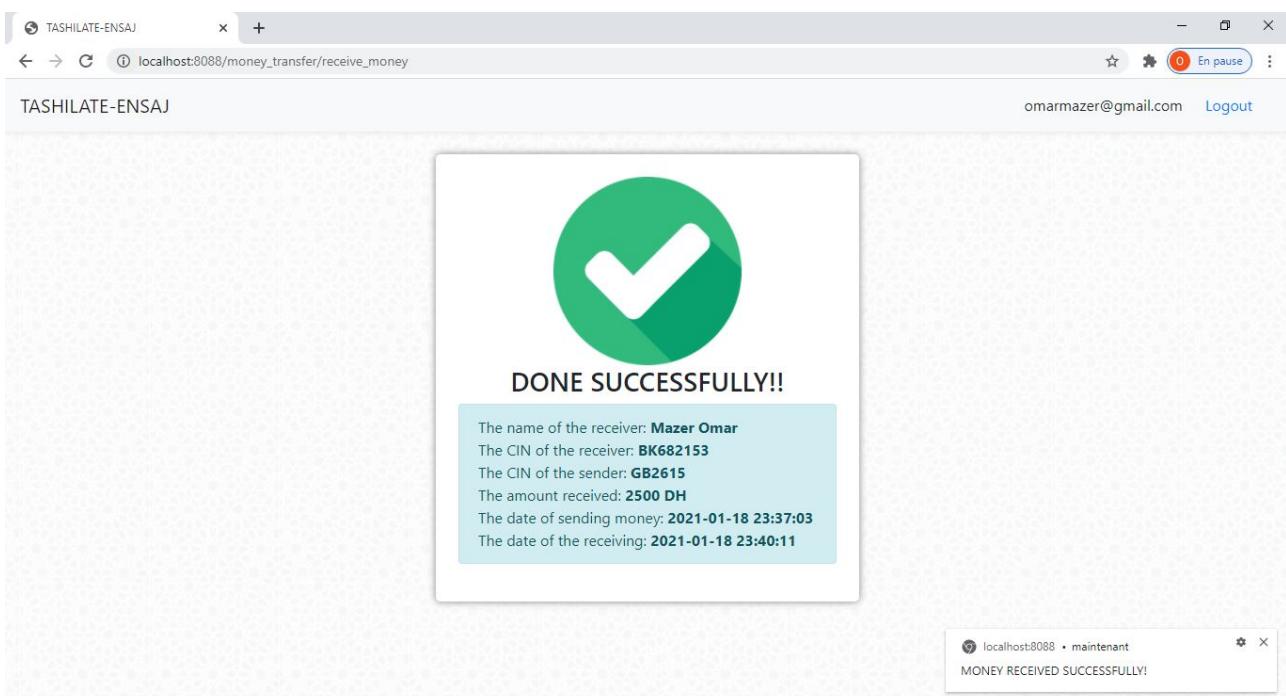


Figure 285:receive money successfully

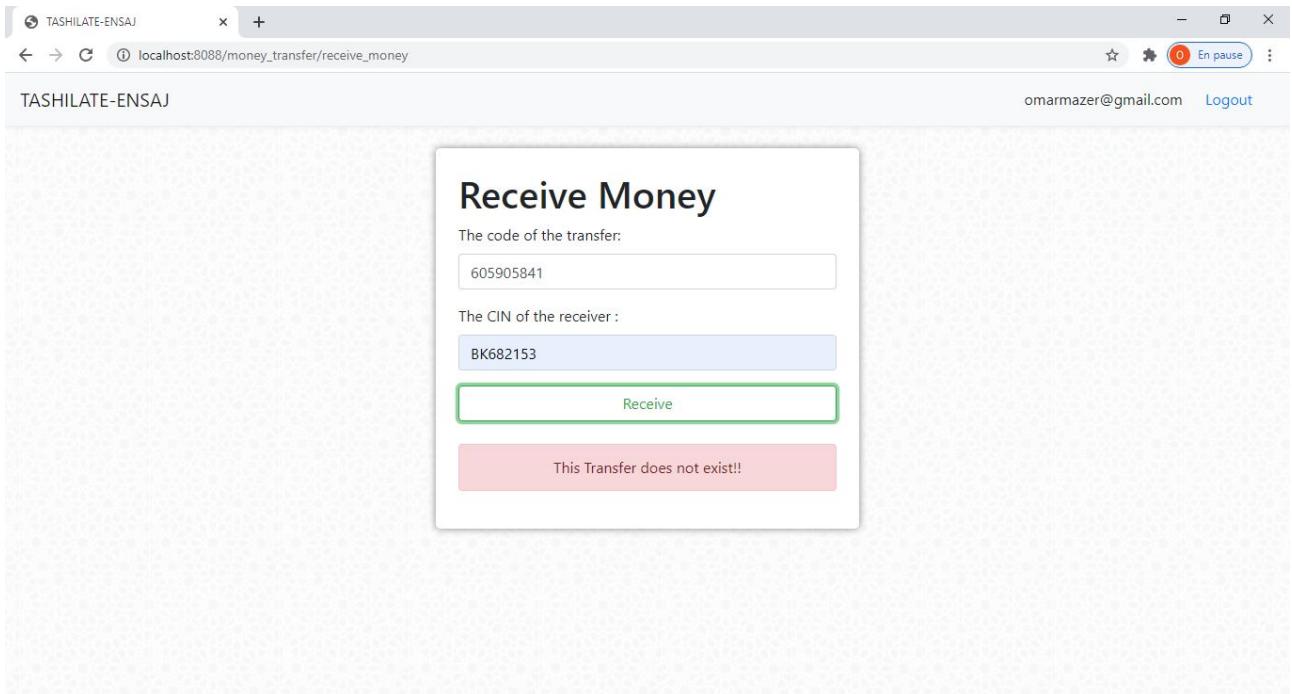


Figure 286:Case of data validation-Confirmation message

4.2.5 Phone Recharge

```
phone_recharge.html
1  <!DOCTYPE html>
2  <html xmlns:th="http://www.thymeleaf.org" xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-springsecurity3">
3  <head>
4      <meta charset="ISO-8859-1">
5      <title>TASHILATE-ENSAJ</title>
6      <link rel="stylesheet" th:href="@{/css/bootstrap.min.css}" type="text/css">
7      <link rel="stylesheet" th:href="@{/css/custom.css}" type="text/css">
8  </head>
9  <body style="background-image: url('/img/bgimg.PNG')">
10     <nav class="navbar navbar-expand-lg navbar-light bg-light mb-4 shadow-sm ">
11         <a class="navbar-brand" th:href="@{}/>TASHILATE-ENSAJ</a>
12         <ul class="navbar-nav ml-auto">
13             <li class="nav-item">
14                 <span sec:authentication="principal.username"> User</span>
15             </li>
16             <li sec:authorize="isAuthenticated()" class="nav-item ml-4 mr-4">
17                 <a th:href="@{/Logout}>Logout</a>
18             </li>
19         </ul>
20     </nav>
21     <div class="col-lg-4 formStyle p-4" id="form">
22         <h1> Phone Recharge </h1>
23         <div class="form-group">
24             <table class="text-center d-flex justify-content-center p-2">
25                 <tr>
26                     <td></td>
27                     <td></td>
28                     <td></td>
29                 </tr>
30                 <tr>
31                     <td><input class="form-check-input" type="radio" name="operator" id="iam" value="Maroc Telecom" checked></td>
32                     <td><input class="form-check-input" type="radio" name="operator" id="orange" value="Orange"></td>
33                     <td><input class="form-check-input" type="radio" name="operator" id="inwi" value="Inwi"></td>
34                 </tr>
35             </table>
36         </div>
37         <div class="form-group">
38             <label for="numTel">The phone number</label>
39             <input type="text" id="numTel" name="numTel" class="form-control" placeholder="0612345678" autofocus="autofocus" required/>
40         </div>
```

Figure 287:phone_recharge.html

```
phone_recharge.html □
39      <input type="text" id="numTel" name="numTel" class="form-control" placeholder="0612345678" autofocus="autofocus" required/>
40  </div>
41  <div class="form-group">
42    <label for="amount">The amount of money </label> :
43  <div class="input-group">
44    <select class="form-control" id="amount" name="amount">
45      <option selected value="5">5</option>
46      <option value="10">10</option>
47      <option value="20">20</option>
48      <option value="25">25</option>
49      <option value="50">50</option>
50      <option value="100">100</option>
51      <option value="200">200</option>
52    </select>
53  <div class="input-group-append">
54    <span class="input-group-text">DH</span>
55  </div>
56 </div>
57 <div class="form-group d-flex justify-content-center p-2">
58  <div class="form-check form-check-inline">
59    <input class="form-check-input" type="radio" name="typeRecharge" id="phone" value="Phone" checked>
60    <label class="form-check-label" for="typeRecharge">Phone</label>
61  </div>
62  <div class="form-check form-check-inline">
63    <input class="form-check-input" type="radio" name="typeRecharge" id="internet" value="Internet">
64    <label class="form-check-label" for="typeRecharge">Internet</label>
65  </div>
66  <div class="form-check form-check-inline">
67    <input class="form-check-input" type="radio" name="typeRecharge" id="special_offer" value="Special offer">
68    <label class="form-check-label" for="typeRecharge">Special offer</label>
69  </div>
70 </div>
71 <div class="form-group">
72  <button type="submit" name="recharge" id="recharge" class="btn btn-outline-success btn-block mb-4" value="Recharge">Recharge</button>
73 </div>
74 <div id="erreur">
75 </div>
76 </div>
77 </div>
78 <div class="col-lg-4 formstyle p-4 text-center" id="result">
79 </div>
80 <script src="/js/jquery-3.3.1.min.js"></script>
81 <script src="/js/phone_recharge.js"></script>
82 </body>
83 </html>
```

Figure 288:phone_recharge.html

```
phone_recharge.js
1 $(document).ready(function(){
2     var numTel = $("#numTel");
3     var amount = $("#amount");
4     $("#result").hide();
5     $("#erreur").hide();
6     $("#recharge").click(function() {
7         $.ajax({
8             url: 'http://localhost:8080/service-recharge/recharge',
9             data: JSON.stringify({numTel: numTel.val(), typeRecharge: $("input[name='typeRecharge']:checked").val(), amount: amount.val()}),
10            headers: { 'Access-Control-Allow-Origin': '*' },
11            type: "POST",
12            dataType: "json",
13            contentType:"application/json; charset=utf-8",
14            async: false,
15            success: function(data, textStatus, jqXHR) {
16                remplir(data);
17                if (!("Notification" in window)) {
18                    alert("This browser does not support desktop notification");
19                }
20                else if (Notification.permission === "granted") {
21                    var notification = new Notification("PHONE RECHARGED SUCCESSFULLY!");
22                }
23                else if (Notification.permission !== "denied") {
24                    Notification.requestPermission().then(function (permission) {
25                        if (permission === "granted") {
26                            var notification = new Notification("PHONE RECHARGED SUCCESSFULLY!");
27                        }
28                    });
29                }
30            },
31            error: function(jqXHR, textStatus, errorThrown) {
32                console.log(textStatus);
33                remplirErreur();
34            }
35        });
36    });
});
```

Figure 289:phone_recharge.js

```
37     function remplirErreurs() {
38         $('#erreur').html('<div class="alert alert-danger text-center">This phone number is not correct!!</div>');
39         $('#erreur').show();
40     }
41     function remplir(data) {
42         var ligne="";
43         ligne+="";
44         ligne+="<h3 color='green'>DONE SUCCESSFULLY!!</h3>";
45         ligne+="<div class='alert alert-info text-left'>The phone number: <strong>' +data.numTel+ '</strong><br/>';
46         ligne+="The amount: <strong>' +data.amount+ ' DH</strong><br/>";
47         ligne+="The operator: <strong>' +data.operator+ '</strong><br/>';
48         ligne+="The type of the recharge: <strong>' +data.typeRecharge+ '</strong><br/>';
49         ligne+="The date of the recharge: <strong>' +data.date+ '</strong><br/>';
50         ligne+="</div>";
51         $('#result').html(ligne);
52         $('#result').show();
53         $('#form').hide();
54     }
55 };
```

Figure 290:phone_recharge.js code continuation

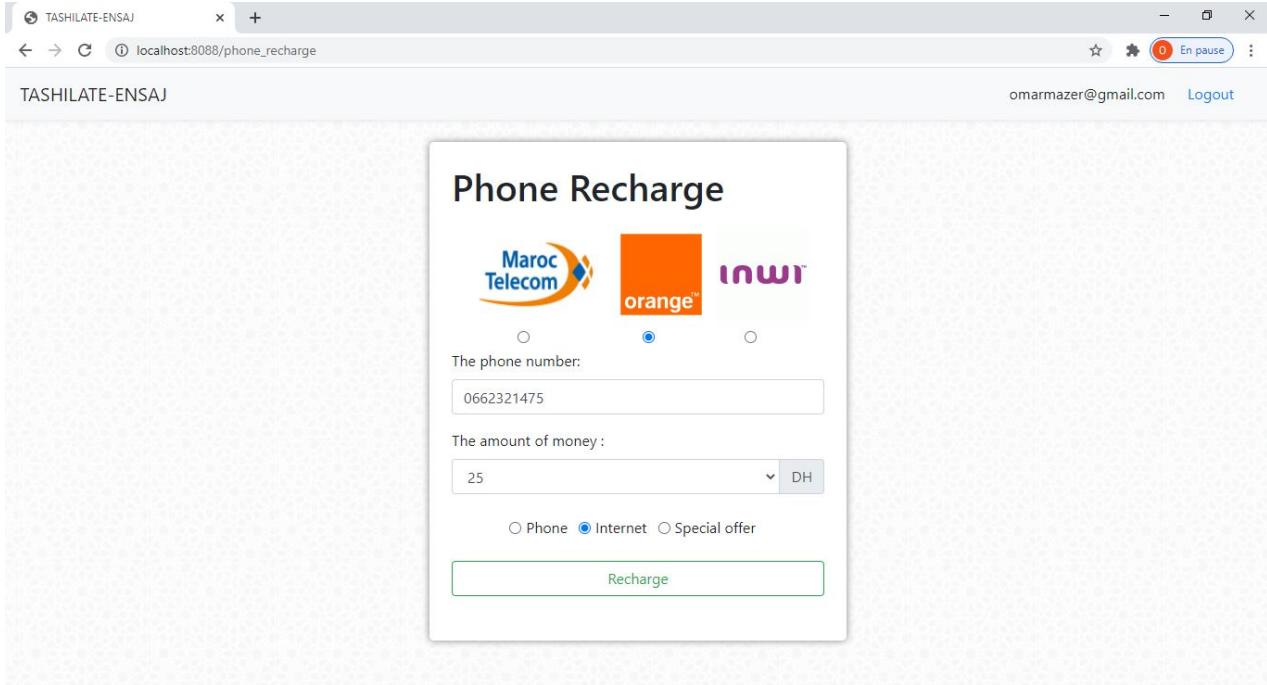


Figure 291:phone recharge in the browser

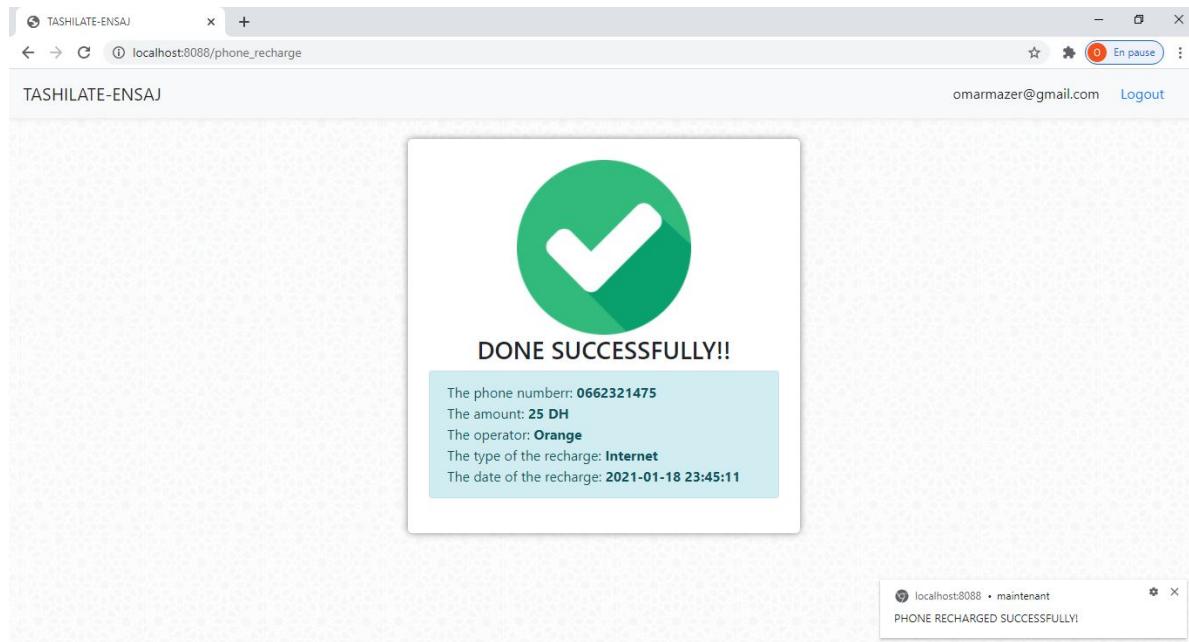


Figure 292:phone recharge done successfully

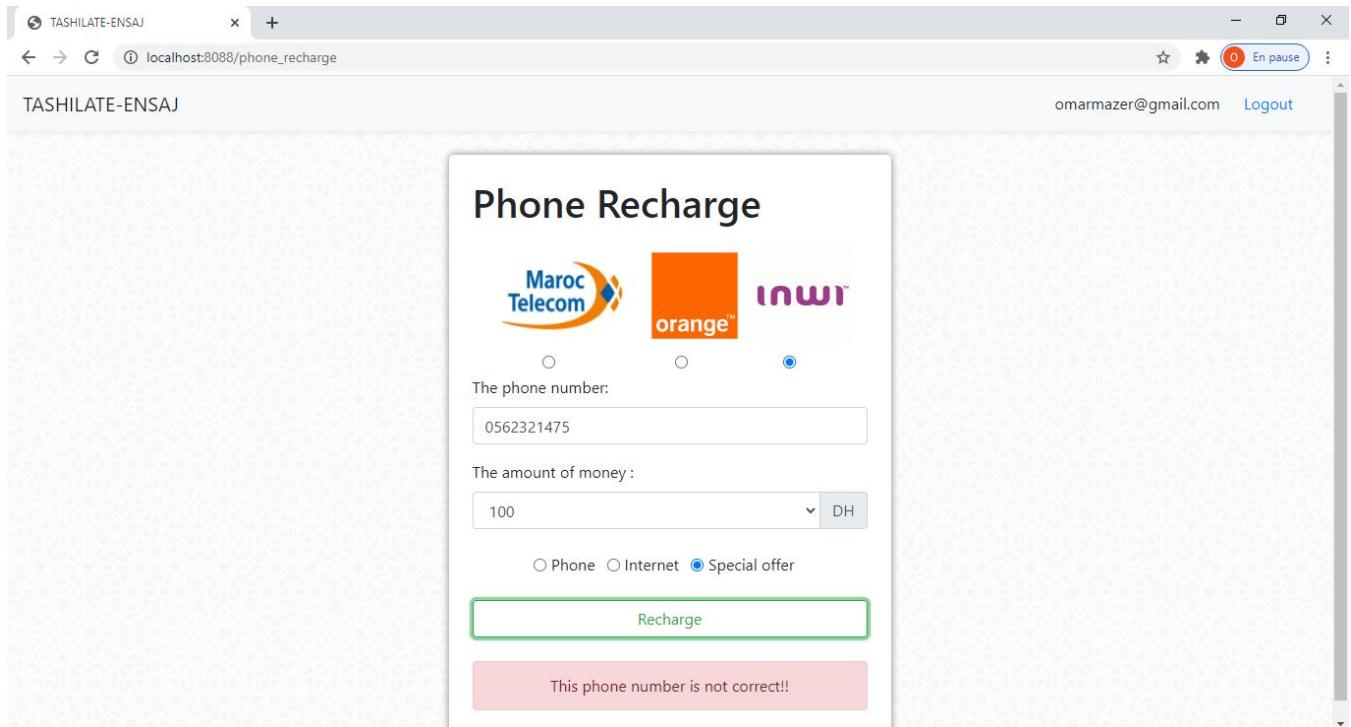


Figure 293:phone recharge data validation-error message

4.2.6 Water & electricity Payment

```
we_payment.html
1  <!DOCTYPE html>
2  
```

Figure 294:Water & electricity Payment html

```
41     <div id="erreur">
42     </div>
43 </div>
44     <div class="col-lg-4 formstyle p-4 text-center" id="bills">
45     </div>
46     <div class="col-lg-4 formstyle p-4 text-center" id="result">
47         
48         <h3 color="green">DONE SUCCESSFULLY!!</h3>
49     </div>
50     <script src="/js/jquery-3.3.1.min.js"></script>
51     <script src="/js/we_payment.js"></script>
52 </body>
53 </html>
```

Figure 295:Water & electricity Payment html continuation

```
we_payment.js
1 $(document).ready(function(){
2     var contractNbr = $("#contractNbr");
3     var agency = $("#agency");
4     $("#bills").hide();
5     $("#result").hide();
6     $("#erreur").hide();
7     $('#search').click(function() {
8         $.ajax({
9             url: 'http://localhost:8080/service-payment/allBills',
10            data: JSON.stringify({contractNbr: contractNbr.val(), agency: agency.val()}),
11            headers: { 'Access-Control-Allow-Origin': '*' },
12            type: 'POST',
13            dataType: "json",
14            contentType:"application/json; charset=utf-8",
15            async: false,
16            success: function(data, textStatus, jqXHR) {
17                remplirTableau(data);
18            },
19            error: function(jqXHR, textStatus, errorThrown) {
20                console.log(textStatus);
21                remplirErreur();
22            }
23        });
24    });
25    function remplirErreur() {
26        $('#erreur').html('<div class="alert alert-danger text-center">There is no bill for this contract number!!</div>');
27        $('#erreur').show();
28    }
29    function remplir(data) {
30        var ligne="";
31        ligne+="

";
32        ligne+="The contract number: <strong>" +data.contractNbr+ "</strong><br/>";
33        ligne+="The W/E provider: <strong>" +data.agency+ "</strong><br/>";
34        ligne+="The month: <strong>" +data.month +'/' +data.year+ "</strong><br/>";
35        ligne+="The amount: <strong>" +data.amount+ " DH</strong><br/>";
36        ligne+="The date: <strong>" +data.date+ "</strong><br/>";
37        ligne+="</div>";
38        $('#result').html($('#result').html()+ligne);
39        $('#result').show();
40        $('#bills').hide();
41    }
}


```

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+="

||
||
||


```

Figure 298:remplirTableau() method

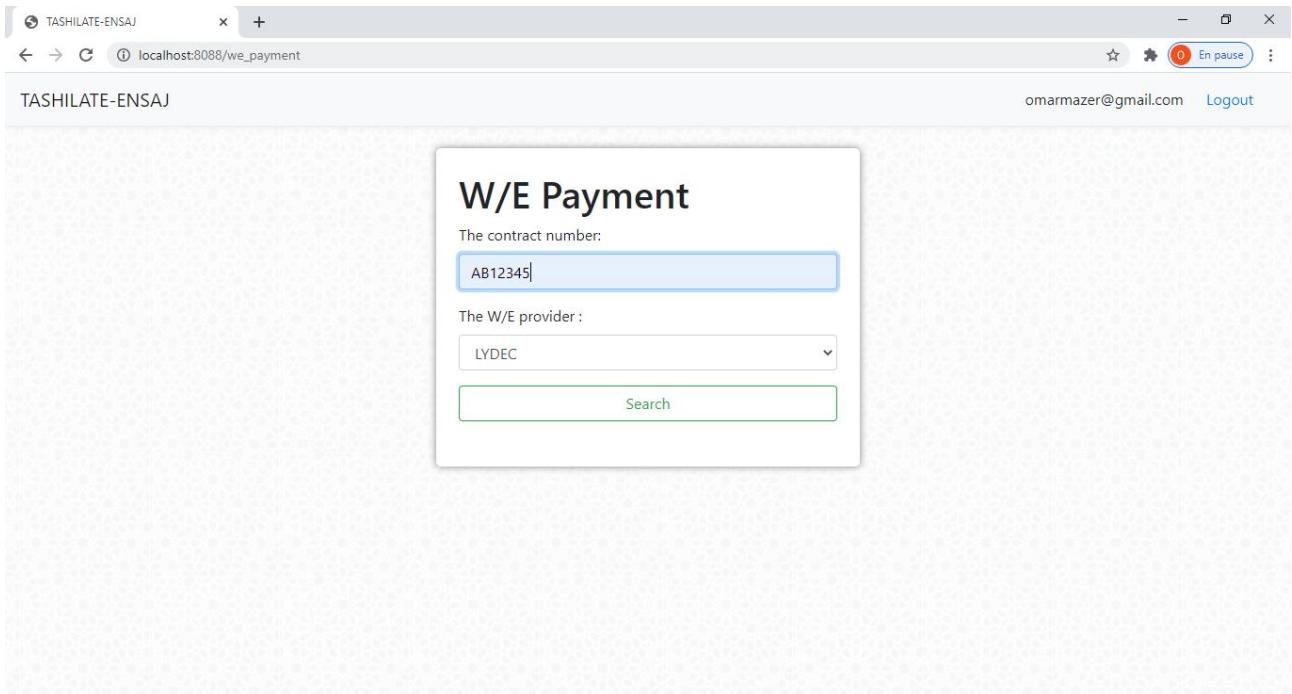


Figure 299:W/E Payment in the browser

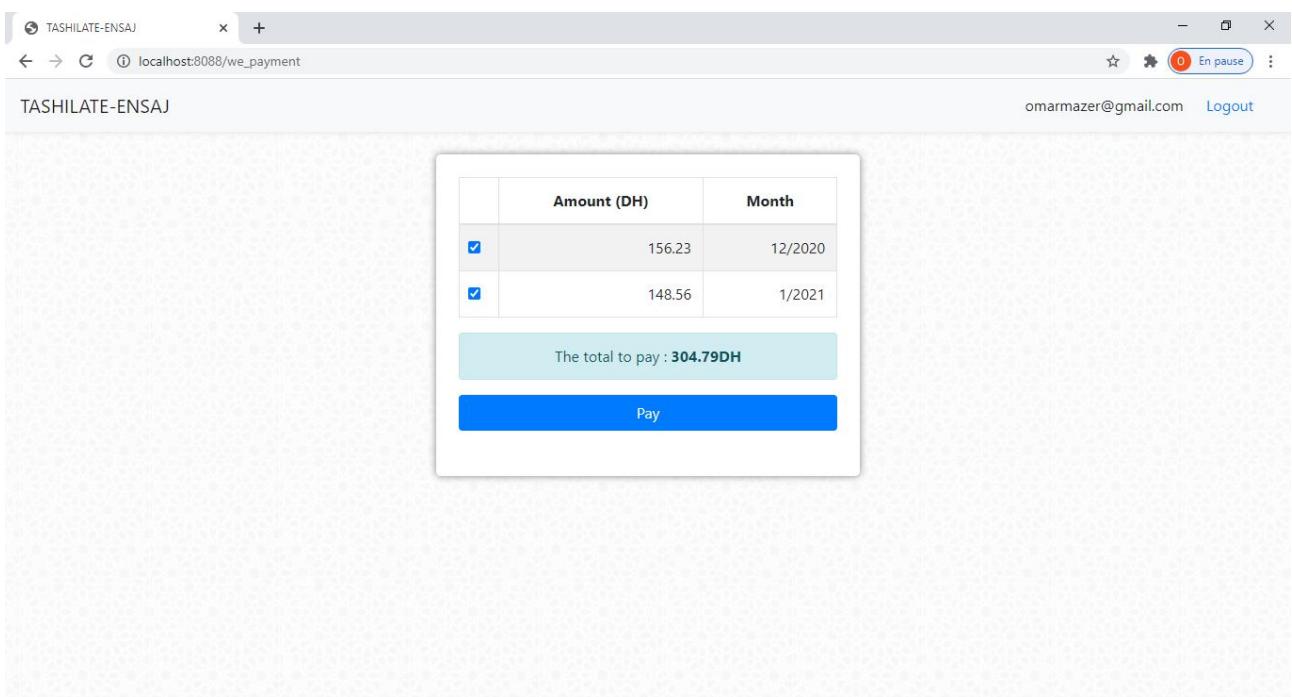


Figure 300:Table w/e payment

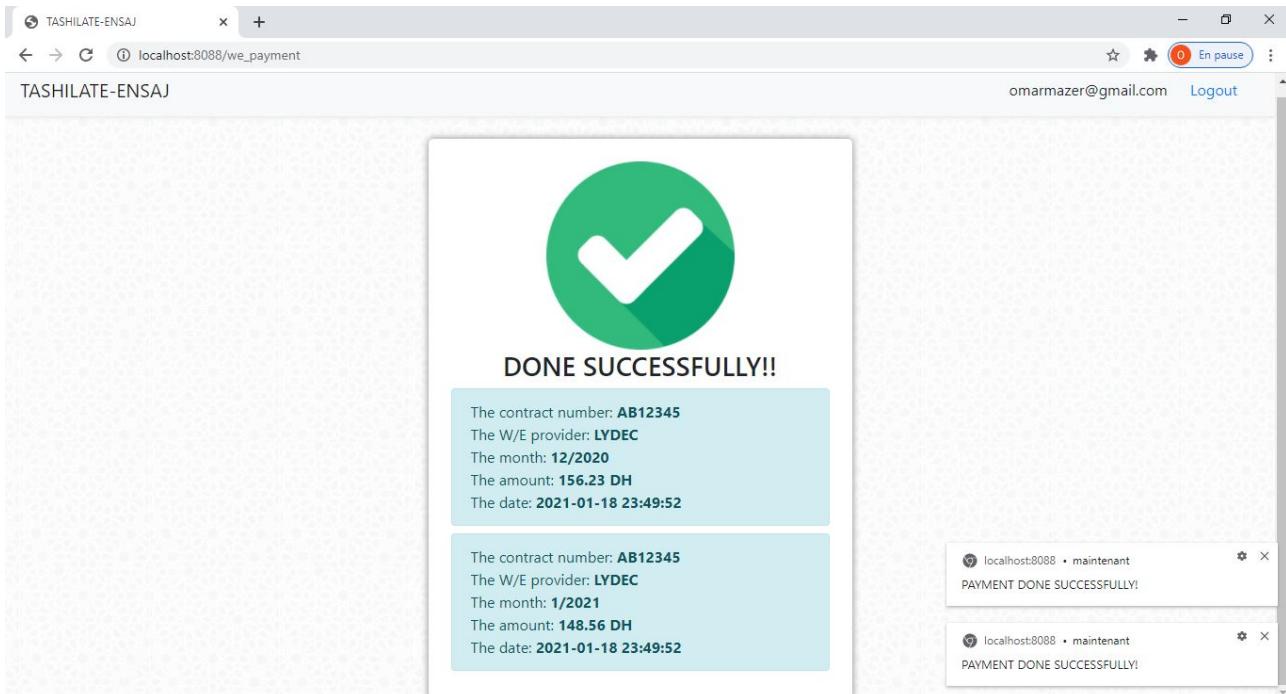


Figure 301:w/e payment done successfully

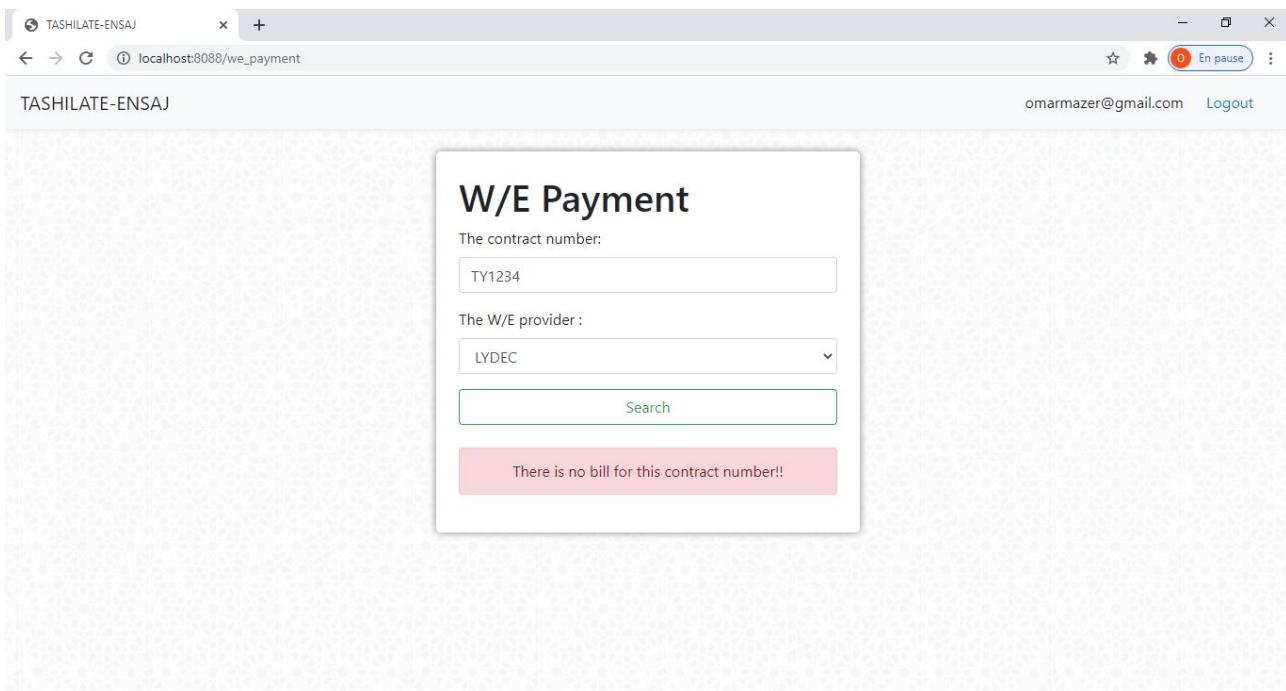
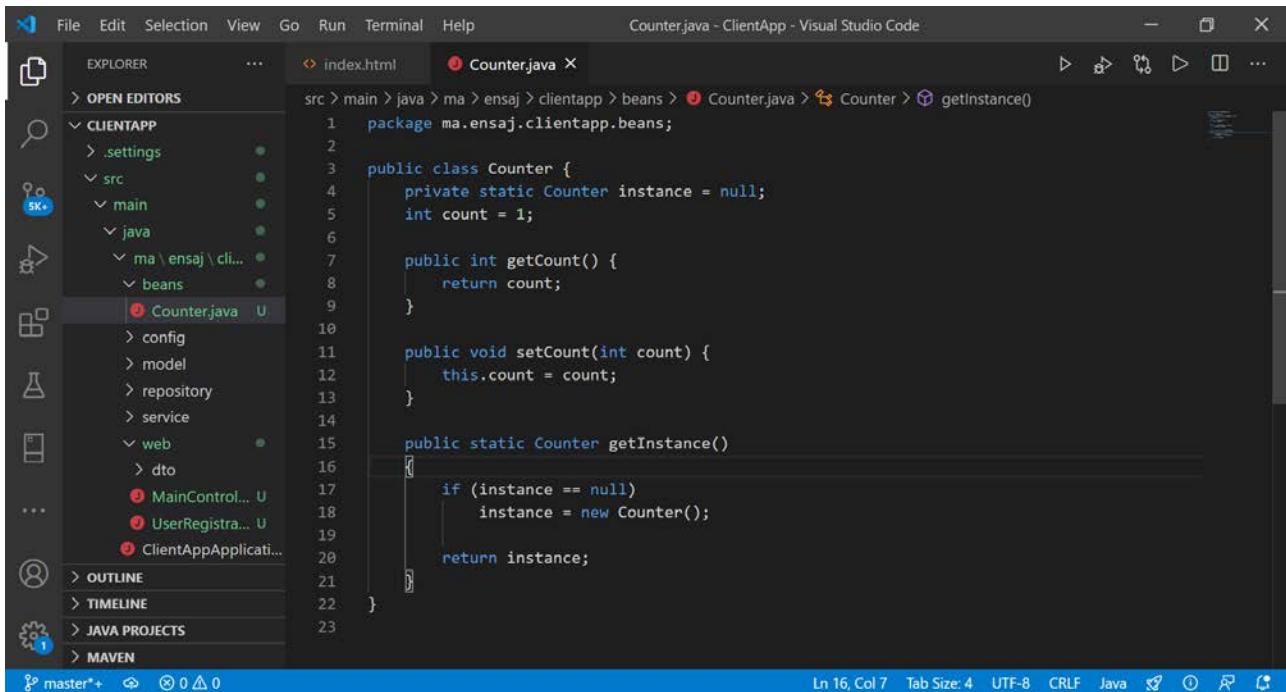


Figure 302:W/E payment in the browser

4.2.7 Visitor Counter feature

1. Create Counter class under the package beans



The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows the project structure under "CLIENTAPP". The "beans" package contains the "Counter.java" file, which is currently selected.
- Editor:** The "Counter.java" file is open. The code defines a static Counter instance and a getCount() and setCount() method. It also includes a static getInstance() method that returns the static instance or creates a new one if null.
- Bottom Status Bar:** Shows "Ln 16, Col 7" and other standard status bar information.

```
package ma.ensaj.clientapp.beans;

public class Counter {
    private static Counter instance = null;
    int count = 1;

    public int getCount() {
        return count;
    }

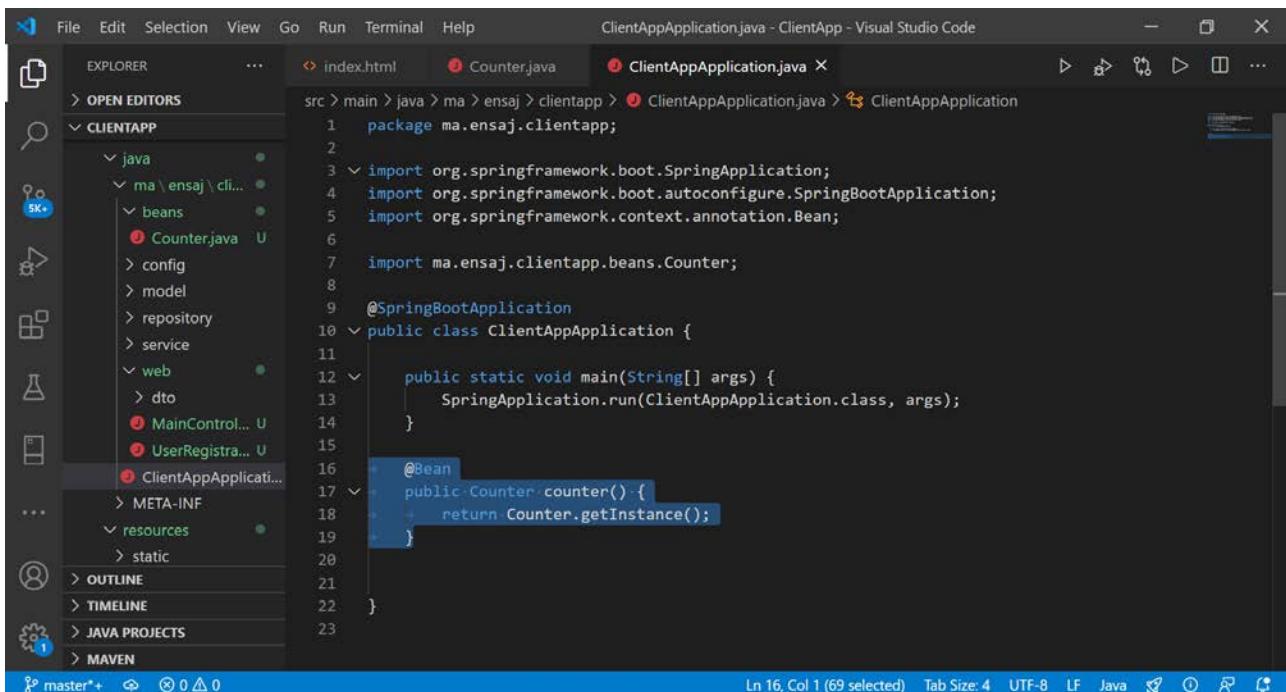
    public void setCount(int count) {
        this.count = count;
    }

    public static Counter getInstance() {
        if (instance == null)
            instance = new Counter();

        return instance;
    }
}
```

Figure 303:Create Counter class under the package beans:

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



The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows the project structure under "CLIENTAPP". The "ClientAppApplication.java" file is selected.
- Editor:** The "ClientAppApplication.java" file is open. It imports SpringBootApplication and Counter from the beans package. The main() method runs the application. A @Bean annotated method named counter() returns the Counter.getInstance() instance.
- Bottom Status Bar:** Shows "Ln 16, Col 1 (69 selected)" and other standard status bar information.

```
package ma.ensaj.clientapp;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import ma.ensaj.clientapp.beans.Counter;

@SpringBootApplication
public class ClientAppApplication {

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

    @Bean
    public Counter counter(){
        return Counter.getInstance();
    }
}
```

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

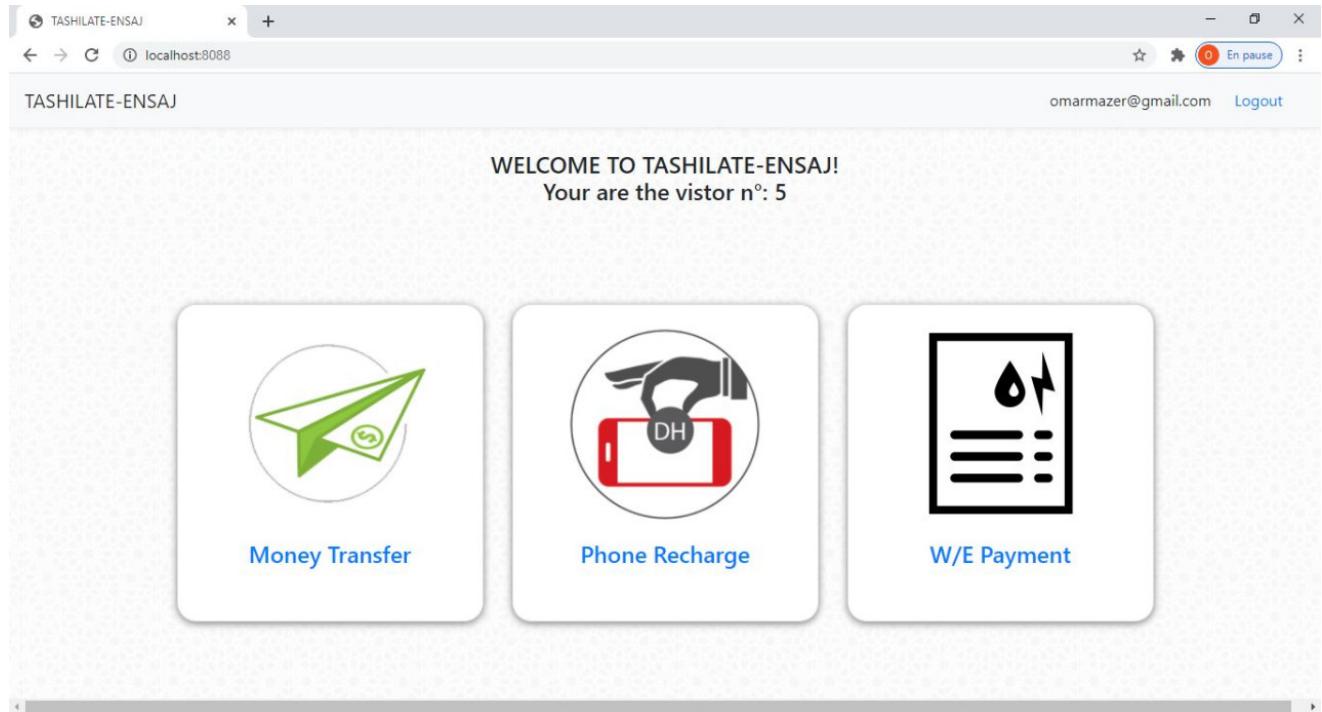


Figure 305:Visitor counter in the browser

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.