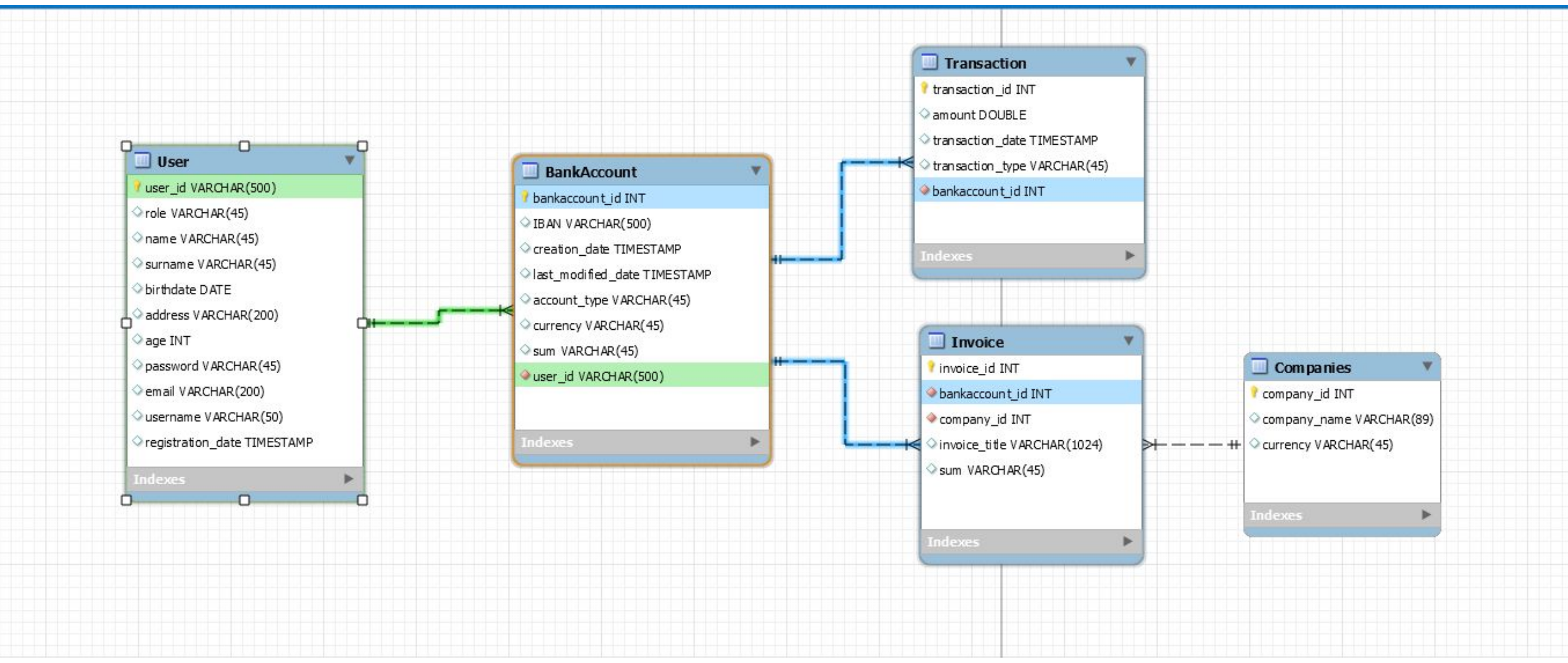
Assignment 2 - Banking Application using Spring Framework and Thymeleaf

For this assignment I have implemented an Online Banking Application for Hai-En-Gi Bank.
The database used to achieve this task is MySQL database.
Below you can see the database schema of the application:



The application has been developed using Spring Framework and the Thymeleaf Template engine for the interface design.

Its main functionalities are the creation of an admin account and many user accounts. The administrator is able to login, to see all the users, add new companies in the portofolio , send emails to users when their account balance is negative and to generate radom bills. Users are only able to register ,so far.

Spring is a framework for dependency-injection which is a pattern that allows to build very decoupled systems, using objects which are called “beans”.Spring requires defining the dispatcher servlet, mappings, and other supporting configurations. We can do this using either the web.xml file or an Initializer class.

The dependency-injection is achieved through the annotations like @Autowired, @Bean, @Component. @Repository etc. and this allows for moving the responsibility of manageing the objects and components to the container.

But ,to make things easier, we used Spring Boot which is an extension of the Spring framework and requires only the ***application.properties*** file to be configured for the application to run smoothly. This is also where we defined the datasource which was our MySQL database and ,also, we defined the ports and protocols for the email sending.

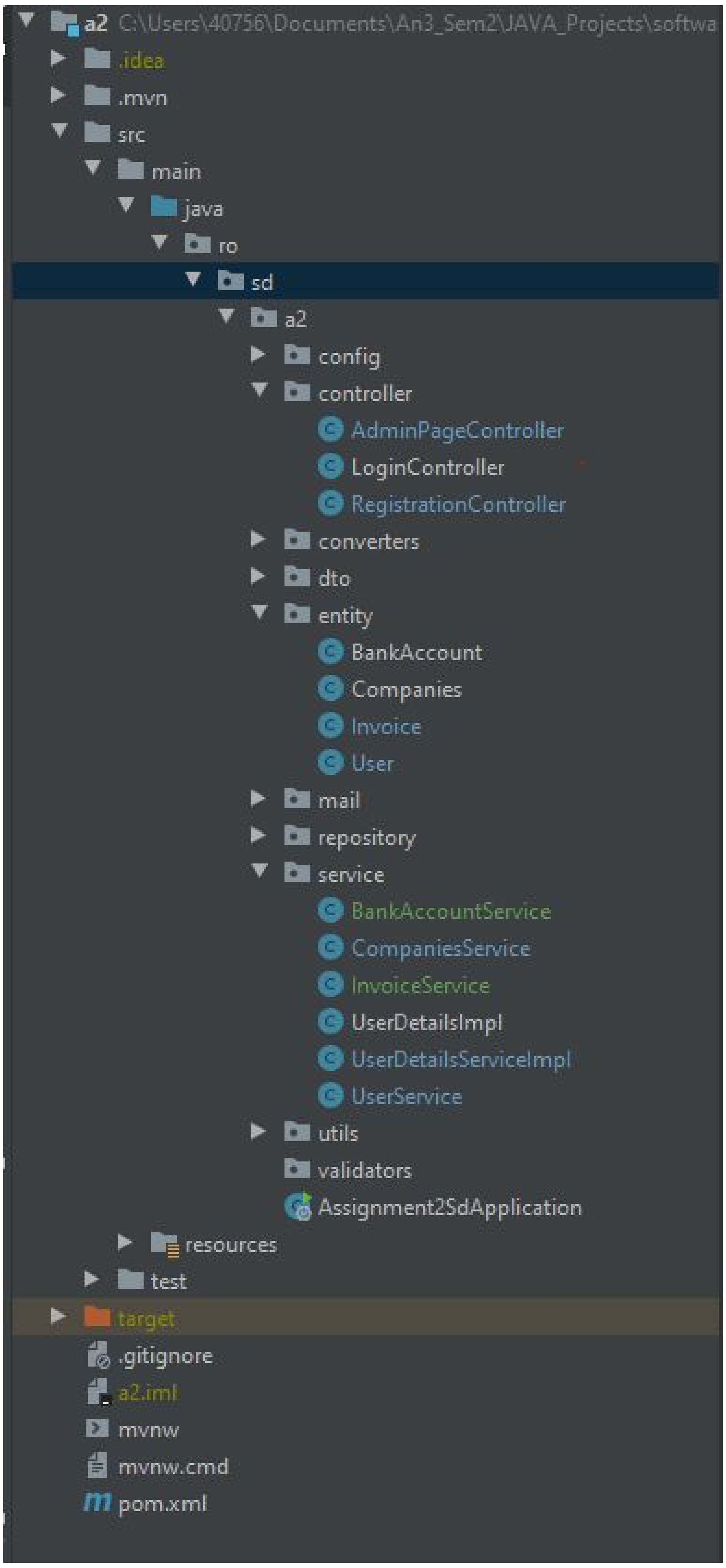
The architectural style chosen for the implementation is MVC and the CRUD operations with the database are managed by Hibernate which is an object-relational mapping tool . As you will see below, the structure of the packages clearly defines the layers of Model-View-Controller architectural pattern.

The model was implemented through JPA repositories which were tasked with the persistency of objects and data retrieval from database.

The controllers and services were all annotated accordingly to their roles with the annotations provided by the Spring Framework (@Controller, @Services).

For security reasons I worked with DataTransferObjects (DTOs) in order to hide sensitive information stored inside entities. As such, DTOs retrieve only essential information from the database for the client.

Also the templates for the web pages are found in the "resources/templates" folder. The templates are HTML pages injected with Thymeleaf attributes which allowed for the retrieval of data form input forms.



```
▼ a2 C:\Users\40756\Documents\An3_Sem2\JAVA_Projects\softwa
  ▶ .idea
  ▶ .mvn
  ▼ src
    ▼ main
      ▼ java
        ▼ ro
          ▼ sd
            ▼ a2
              ▶ config
              ▼ controller
                  AdminPageController
                  LoginController
                  RegistrationController
              ▶ converters
              ▶ dto
              ▼ entity
                  BankAccount
                  Companies
                  Invoice
                  User
              ▶ mail
              ▶ repository
              ▼ service
                  BankAccountService
                  CompaniesService
                  InvoiceService
                  UserDetailsImpl
                  UserDetailsServiceImpl
                  UserService
              ▶ utils
                validators
                Assignment2SdApplication
      ▶ resources
    ▶ test
  ▶ target
    .gitignore
    a2.iml
    mvnw
    mvnw.cmd
    pom.xml
```

The Security Class

Last but not least, I worked with Spring Security to ensure a more secure authentication process for the user.
The WebSecurityConfig class is annotated with @EnableWebSecurity to enable Spring Security's web security support and provide the Spring MVC integration. It also extends WebSecurityConfigurerAdapter and overrides a couple of its methods to set some specifics of the web security configuration.

The configure(HttpSecurity) method defines which URL paths should be secured and which should not. Specifically, the / , /welcome,/signup, /generate, /admin_dashboard , /user_dashboard paths are configured to not require any authentication. All other paths must be authenticated.

When a user successfully logs in, they are redirected to the previously requested page that required authentication. There is a custom /login page (which is specified by loginPage()), and everyone is allowed to view it. In the case when authentication succeds the .deffaullSuccessUrl , which is the page that users are redirected to, is the /admin_Dashboard. In case of error, the User is prompted with a message which will tell him the credentials he entered were wrong.

The userDetailsServiceImpl is my version of service which loads a UserDetails objects needed for the authentication.

Also here I have defined the AuthenticationManager which returns an Authentication of it veirifies id the input represed a valid object.

Lastly, I have defined a bean for the BCryptPasswordEncoder which uses adaptive hash algorithm to encode and store the passwords.

References :
https://www.javaguides.net/2018/10/user-registration-module-using-springboot-springmvc-springsecurity-hibernate5-thymeleaf-mysql.html
https://medium.com/@grokwich/spring-boot-thymeleaf-html-form-handling-762ef0d51327
https://www.jackrutorial.com/2018/04/spring-boot-user-registration-login.html