

Online Banking System Project

By

**Amer Mamoun Aziz Gasimov**

Supervised by:

**Dr. Kovásznai Gergely**

**Contents**

[ Project Management Plan: 2](#_Toc10667617)

[ Over view: 2](#_Toc10667618)

[ Project scope and objectives: 2](#_Toc10667619)

[ Project Schedule: 2](#_Toc10667620)

[ Requirements Specifications: 3](#_Toc10667621)

[ Accounts 3](#_Toc10667622)

[ Transfer 3](#_Toc10667623)

[ Transaction 3](#_Toc10667624)

[ Menu 3](#_Toc10667625)

[ Registration and login system 3](#_Toc10667626)

[ Design: 4](#_Toc10667627)

[ Database: 4](#_Toc10667628)

[ Implementation: 5](#_Toc10667629)

[ Backend: 5](#_Toc10667630)

[- MVC: Model view controller 5](#_Toc10667631)

[- DAO: Data Access Object 6](#_Toc10667632)

[- Services Components: 7](#_Toc10667633)

[ Frontend: 8](#_Toc10667634)

[- Thymeleaf: 8](#_Toc10667635)

[ Testing: 9](#_Toc10667636)

[Figures 11](#_Toc10667637)

[References 12](#_Toc10667638)

# Project Management Plan:

## Over view:

The purpose of this project to build a simple online banking system using Java and Spring framework.

The main purpose of this project is to learn the proper way to develop an application which is similar in a way to the real project in the labor market.

Also, to deal with the database:

What kind of database we should use, how to make the perfect connection,

How to manage the entities and tables, and the columns...and so on.

## Project scope and objectives:

The project team will focus on the futures of the Spring framework especially in the front-end, also to implement the controllers. And to make sure to use programming technology patterns like Adapter, Inheritance…ect.

In Spring framework, we will deal with the security configuration which luckily it has a quite good documentation to help us to develop the security functions of the application.

Also, the project team will implement the necessary procedures for testing.

## Project Schedule:

The project deadline 5th of June, the project will take few weeks to be completed with all the expected functionalities, also we have a Trello page to divide the work between the team members.

For each phase of the project there is a deadline, also an organized card to manage the “To do” list, “in progress” list and “Finished tasks” list.

This is so handy tool to manage our work, and to be organized as we are proceeding with developing the project.

**GitHub repository:** with installation instructions

<https://github.com/AZIZGASIMOV94/userFront>

# Requirements Specifications:

This project will have the following functionalities:

## Accounts

* Primary
  + View Transaction list
  + Search Transaction
* Savings
  + View Transaction list
  + Search Transaction

## Transfer

* Between Accounts
* Person-To-Person
* Add/Edit Recipients

## Transaction

* Deposit
* Withdrawal

## Menu

* Profile Settings
* Schedule Appointment

## Registration and login system

# Design:

## Database:

We extract the entities from our requirements, we will have entities as the following:

Each User will have only one primary account and only one saving account and then each account type has an entity for a transaction so we will have for each type many primary transection and many saving transection. So the relationship here is one-to-many.

Also, a User can have many Recipients entity and many Appointment entity.  
so the relationship here is one-to-many.  
the Entity relationship diagram will look like the following figure.

Figure 1 Database diagram

# Implementation:

## Backend:

We are using the most recent technologies for (Spring framework , n.d.)

(Spring\_Security\_4, n.d.), Spring Data and Spring framework with (Hibernate, n.d.)which can be used to configure the entities of our database and define the fields and the relations between entities.

Also, we used Mysql server to manipulate the accounts and the records.

One example for a dependency that have been used is:

<dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-security</artifactId>  
</dependency>

Figure 2 Security dependency

This dependency will download the required files for Spring security which will automatically will requires basic authentication for all HTTP endpoints.

### MVC: Model view controller

One of the architecture patters we have used in our project is (MVC, n.d.) this pattern, will specifies that an application consist of a data model, presentation information, and control information. The pattern requires that each of these be separated into different objects.



Figure 3 MVC Diagram

The Model contains only the pure application data, it contains no logic describing how to present the data to a user.

* The View presents the model’s data to the user. The view knows how to access the model’s data, but it does not know what this data means or what the user can do to manipulate it.
* The Controller exists between the view and the model. It listens to events triggered by the view (or another external source) and executes the appropriate reaction to these events. In most cases, the reaction is to call a method on the model. Since the view and the model are connected through a notification mechanism, the result of this action is then automatically reflected in the view.

### DAO: Data Access Object

(DAO, n.d.)stands for Data Access Object. It’s a design pattern in which a data access object (DAO) is an object that provides an abstract interface to some type of database or other persistence mechanisms. By mapping application calls to the persistence layer, DAOs provide some specific data operations without exposing details of the database. This isolation supports the Single responsibility principle. It separates what data accesses the application needs, in terms of domain-specific objects and data types (the public interface of the DAO), and how these needs can be satisfied with a specific DBMS, database schema.

I order to connect the project with the database it’s mandatory to use the JDBC, JPA and mysql-connector-java dependencies as the following:

<dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-jdbc</artifactId>  
</dependency>  
<dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-data-jpa</artifactId>  
</dependency>  
<dependency>  
 <groupId>mysql</groupId>  
 <artifactId>mysql-connector-java</artifactId>  
</dependency>

Figure 4 Dependencies

### Services Components:

Service Components are the class file which contains @Service annotation. These class files are used to write business logic in a different layer, separated from @AccountController class file. The logic for creating a service component class file is shown here:

public interface AccountService {  
 PrimaryAccount createPrimaryAccount();  
 SavingsAccount createSavingsAccount();  
 void deposit(String accountType, double amount, Principal principal);  
  
 void withdraw(String accountType, double amount, Principal principal);  
}

Figure 5 E.g. AccountService

The class that implements the Interface with @Service annotation is as shown:

@Service  
public class AccountServiceImpl implements AccountService {  
  
 private static int *nextAccountNumber* = 11223145;  
  
 @Autowired  
 private PrimaryAccountDao primaryAccountDao;  
  
 @Autowired  
 private SavingsAccountDao savingsAccountDao;

@Autowired  
 private UserService userService;  
  
 @Autowired  
 private TransactionService transactionService;

…

…

}

Figure 6 E.g. AccoutnServiceImpl

## Frontend:

### Thymeleaf:

(Thymeleaf, n.d.)is a Java-based library used to create a web application. It provides a good support for serving a XHTML/HTML5 in web applications, and it’s the default template engine for spring currently.

#### \_Thymeleaf Templates:

Thymeleaf converts the files into well-formed XML files. It contains 6 types of templates as given below −

* XML
* Valid XML
* XHTML
* Valid XHTML
* HTML5
* Legacy HTML5

All templates, except Legacy HTML5, are referring to well-formed valid XML files. Legacy HTML5 allows us to render the HTML5 tags in web page including not closed tags.

In Maven we need to add the following dependency into the pom.xml file:

<dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-thymeleaf</artifactId>  
</dependency>

Figure 7 Thymeleaf dependency

An example for using thymeleaf as the following:

<head th:replace="common/header :: common-header"/>

Also,

<form th:action="@{/appointment/create}" method="post" id="appointmentForm">  
 <input type="hidden" name="id" th:value="${appointment.id}"/>

Figure 8 E.g. using of thymeleaf

We also used Bootstrap, Javascript/JQuery and many other libraries…

#### Javascript/JQuery have been used for representing the transaction table in the Primary Account as the following:

<script>  
 $(document).ready(function() {  
 $('#example').DataTable();  
 } );  
</script>

Figure 9 E.g. using Javascript/JQuery

# Testing:

Spring Boot (testing, n.d.) provides a number of utilities and annotations to help when testing the application. (test, n.d.) support is provided by two modules: spring-boot-test contains core items, and spring-boot-test-autoconfigure supports auto-configuration for tests.

I order to be able to test some of the functions we need to add the required dependency into pom.XML file:

<dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-test</artifactId>  
 <scope>test</scope>  
</dependency>

Figure 10 Testing dependency

One example for testing method as the following:

@InjectMocks  
private UserServiceImpl userService;  
  
private User userDoc;  
private Set<UserRole> userRole;  
  
@Before  
public void init(){  
 User user = new User();  
 user.setUserId(*USER\_ONE\_ID*);  
 user.setEmail("gasimovaziz@gmail.com");  
  
 user.setEmail("abc@gmail.com");  
}  
  
@Test(expected = ApiException.class)  
public void createUserAndUserExists(){  
 //data prep  
 Mockito.*when*(userDaoMock.findByEmail("gasimovaziz@gmail.com")).thenReturn(userDoc);  
  
 //Method call  
 User user = userService.createUser(userDoc, userRole);  
  
 //verification  
 Assert.*assertNull*(user);  
 Mockito.*verify*(userDaoMock,Mockito.*times*(1)).findByEmail(Mockito.*anyString*());  
 Mockito.*verifyNoMoreInteractions*(userDaoMock);

Figure 11 E.g. Testing

# Figures

[Figure 1 Database diagram 4](#_Toc10667675)

[Figure 2 Security dependency 5](#_Toc10667676)

[Figure 3 MVC Diagram 5](https://d.docs.live.net/e0869a9ac3b80448/My%20Projects/Intellij%20IDEA/E-Banking%20Web%20Application/Documentation.docx#_Toc10667677)

[Figure 4 Dependencies 6](#_Toc10667678)

[Figure 5 E.g. AccountService 7](#_Toc10667679)

[Figure 6 E.g. AccoutnServiceImpl 7](#_Toc10667680)

[Figure 7 Thymeleaf dependency 8](#_Toc10667681)

[Figure 8 E.g. using of thymeleaf 8](#_Toc10667682)

[Figure 9 E.g. using Javascript/JQuery 9](#_Toc10667683)

[Figure 10 Testing dependency 9](#_Toc10667684)

[Figure 11 E.g. Testing 10](#_Toc10667685)

# References

DAO. (n.d.). Retrieved from https://www.baeldung.com/java-dao-pattern

Hibernate. (n.d.). Retrieved from https://www.baeldung.com/spring-boot-hibernate

MVC. (n.d.). Retrieved from https://www.geeksforgeeks.org/mvc-design-pattern/

Spring framework . (n.d.). Retrieved from https://www.tutorialspoint.com/spring\_boot/spring\_boot\_introduction.htm

Spring\_Security\_4. (n.d.). Retrieved from https://docs.spring.io/spring-security/site/docs/4.2.13.BUILD-SNAPSHOT/reference/htmlsingle/

test. (n.d.). Retrieved from https://www.baeldung.com/spring-boot-testing

testing. (n.d.). Retrieved from https://docs.spring.io/spring-boot/docs/current/reference/html/boot-features-testing.html

Thymeleaf. (n.d.). Retrieved from https://www.thymeleaf.org/documentation.html