<Online banking>

Analysis and Design Document

Student: Anda Papita

**Group: 30431**

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| <dd/mmm/yy> | <x.x> | <details> | <name> |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Table of Contents

I. Project Specification 4

II. Elaboration – Iteration 1.1 4

1. Domain Model 4

2. Architectural Design 4

2.1 Conceptual Architecture 4

2.2 Package Design 4

2.3 Component and Deployment Diagrams 4

III. Elaboration – Iteration 1.2 4

1. Design Model 4

1.1 Dynamic Behavior 4

1.2 Class Design 4

2. Data Model 4

3. Unit Testing 4

IV. Elaboration – Iteration 2 4

1. Architectural Design Refinement 4

2. Design Model Refinement 4

V. Construction and Transition 5

1. System Testing 5

2. Future improvements 5

VI. Bibliography 5

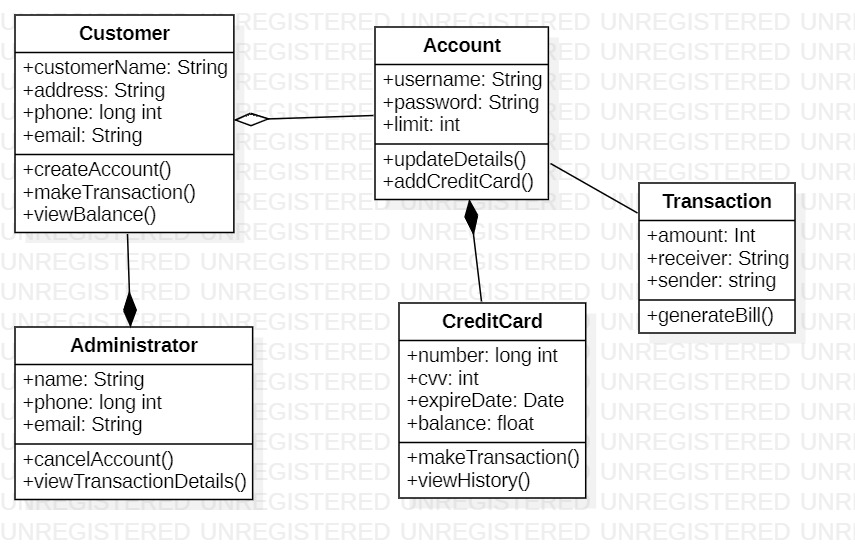
# Project Specification

Online banking is a term used for performing balance checks, account transactions, payments etc. via an online website. Online banking today is most often performed on the Internet but can also use special programs downloaded to an online device.

This is an application for maintaining a person's account regarding funds. This project shows the working of a banking account system and cover the basic functionality of an online banking application. The purpose was to develop a project for solving financial applications of a customer in banking environment in order to nurture the needs of an end banking user by providing various ways to perform banking tasks. It also allows to enable the user’s work space, to have additional functionalities which are not provided under a conventional banking project. The online banking application undertaken as a project is based on relevant technologies.

# Elaboration – Iteration 1.1

# Domain Model



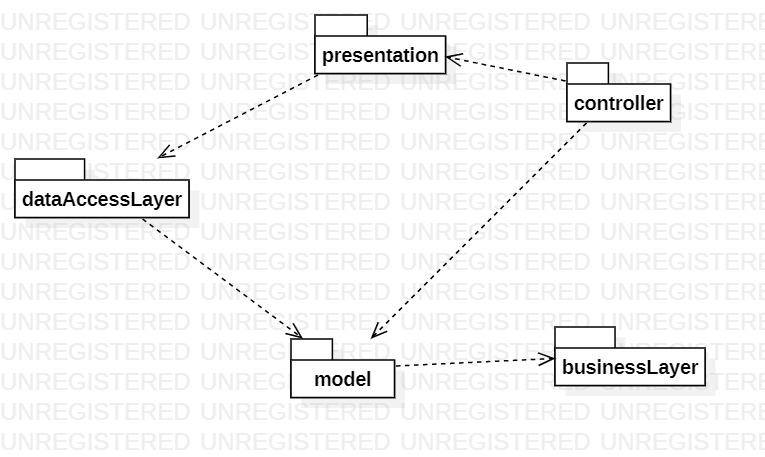
# Architectural Design

## Conceptual Architecture

The design of this app follows a layered design architecture. The 4 packages are: model, presentation(view), dataAccessLayer and controller. Therefore, the model package is represented by the classes Customer and Administrator. These classes are the ones which contain as attributes all the fields that can be found in the corresponding tables from the database. The dataAccessLayer is the package that contains the classes that handle the connectivity between the database and the model classes. The presentation package is represented by the classes View and Controller. Furthermore, I decided to use the **Factory design pattern** which is applied to the classes from the dataAccessLayer, which assure the connectivity with the data base.

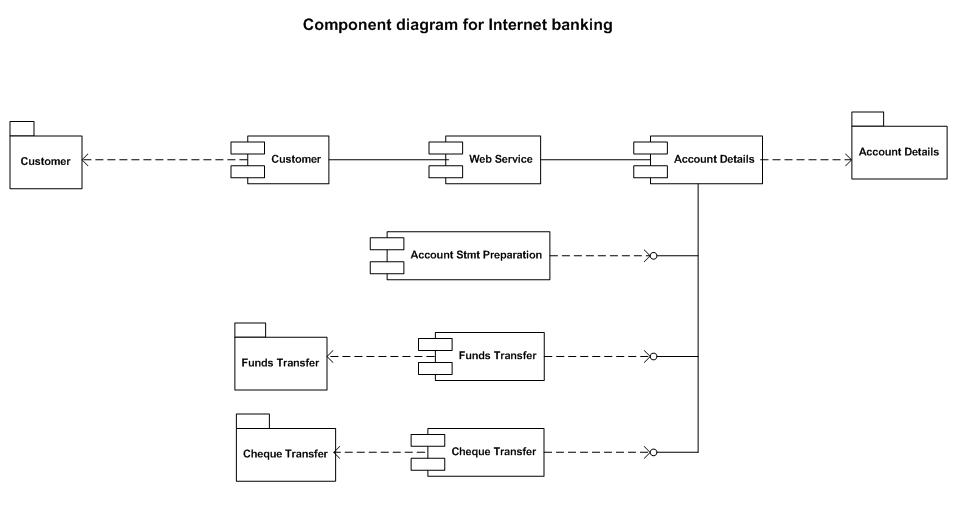
My application is a web application created with **Java Servlet Pages (Jsp)** and **Tomcat** Server, in the Eclipse framework. The database was created with **MySql.**

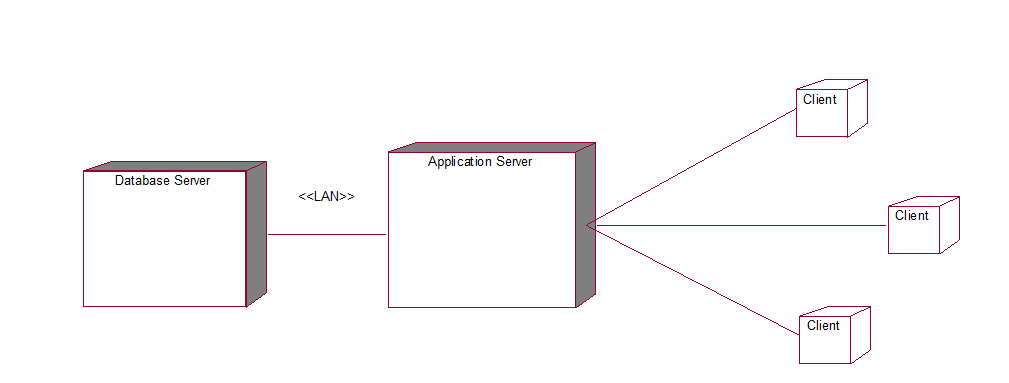
## Package Design



## Component and Deployment Diagrams

The component and deployment diagrams:

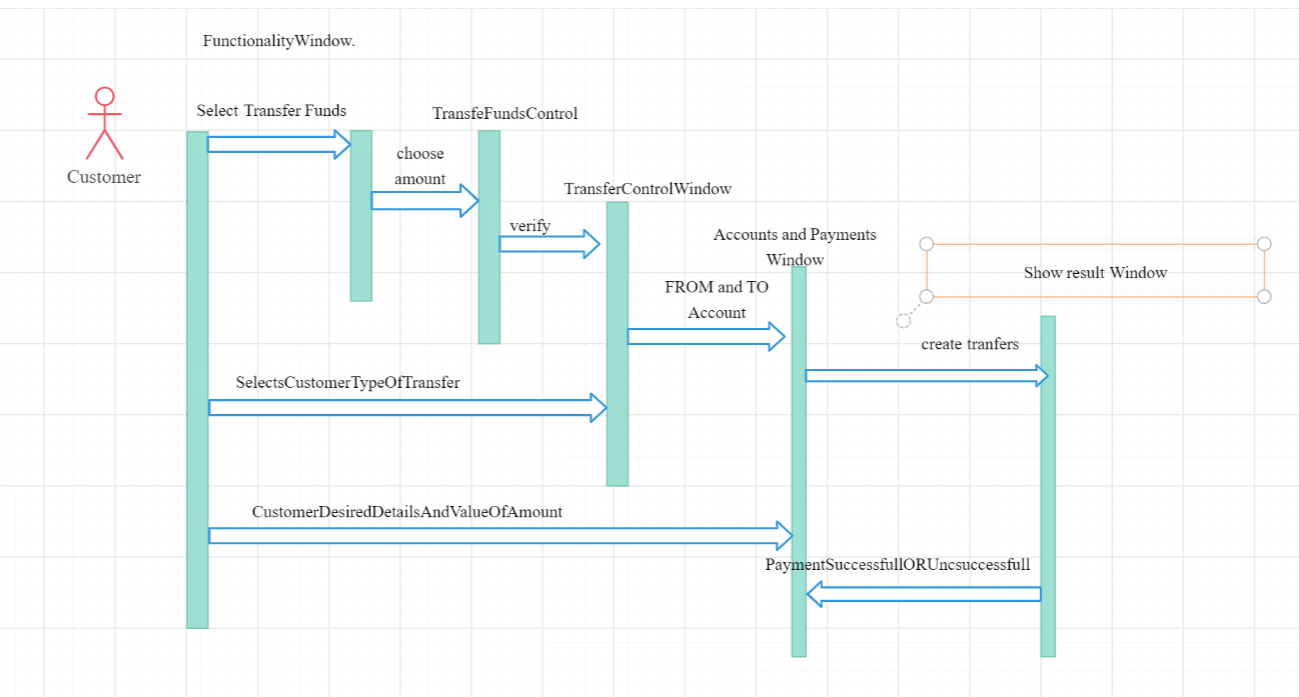




# Elaboration – Iteration 1.2

# Design Model

## Dynamic Behavior

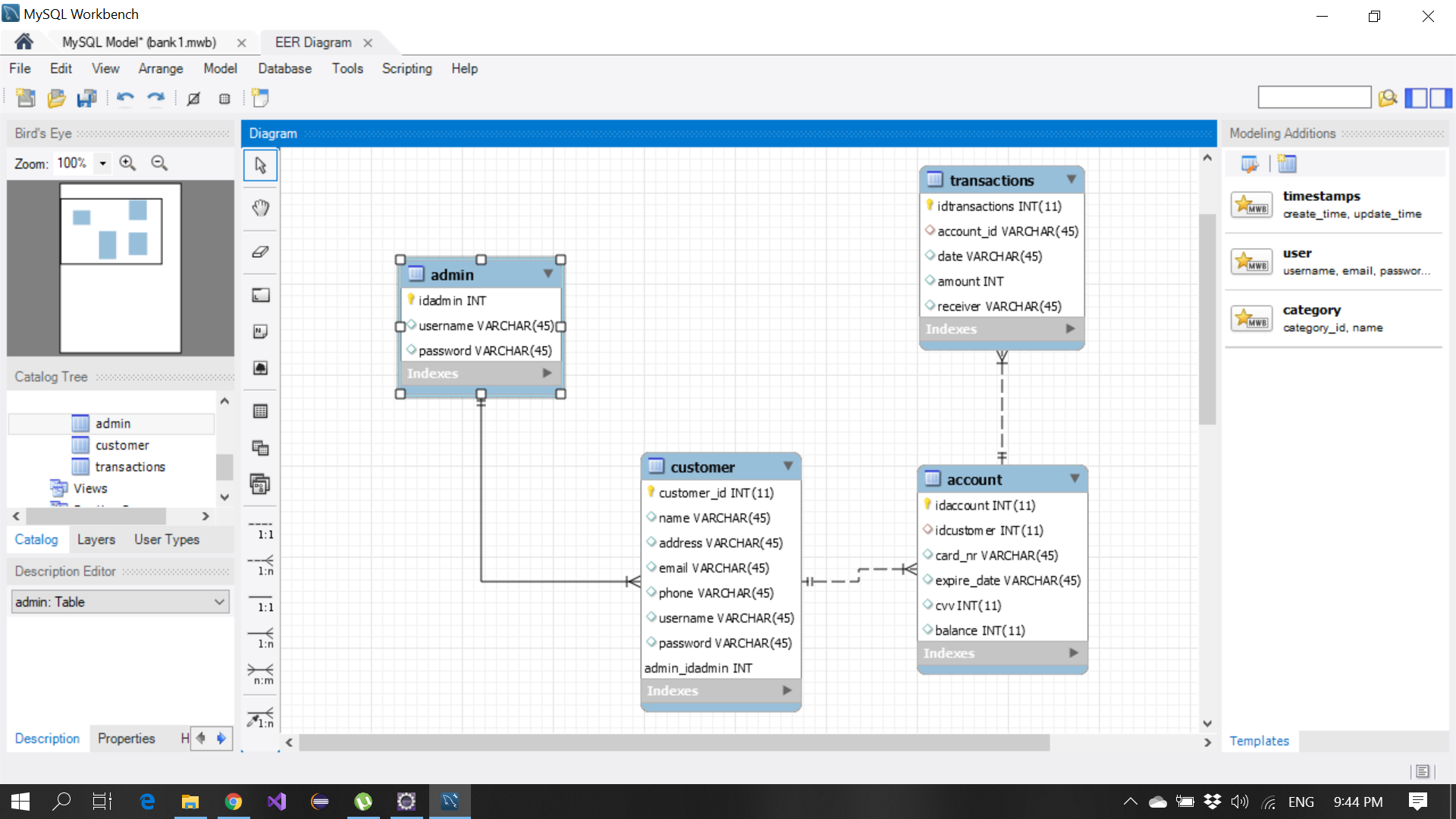


## Class Design

See II.1. Domain model for the initial class diagram (this is still in the works as I have yet to choose the design patterns that will be used).

# Data Model

The diagram for the database, in which we can see the data model, is:



# Unit Testing

See V.1. System testing chapter.

# Elaboration – Iteration 2

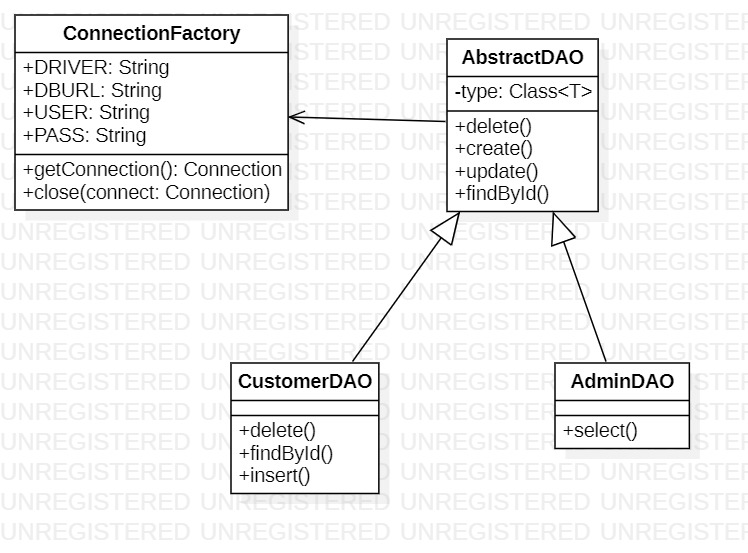
# Architectural Design Refinement

I have chosen to user the Abstract Factory Design Pattern. which is applied to the classes from the dataAccessLayer, which assure the connectivity with the data base.

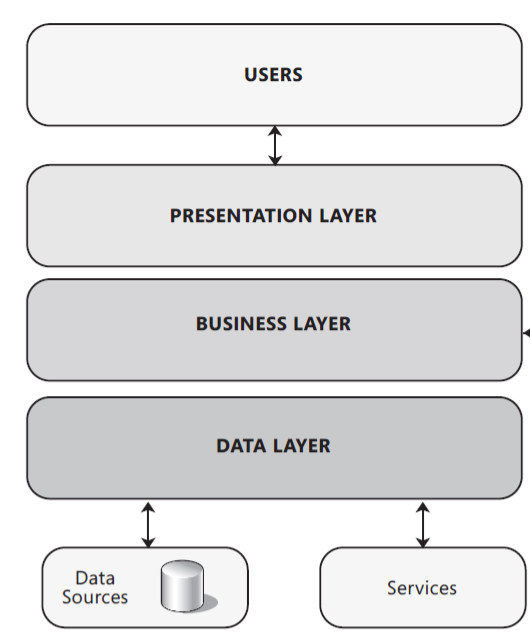
Factory method is a [creational design pattern](https://www.geeksforgeeks.org/design-patterns-set-1-introduction/), i.e., related to object creation. In Factory pattern, we create object without exposing the creation logic to client and the client use the same common interface to create new type of object.  
The idea is to use a static member-function (static factory method) which creates & returns instances, hiding the details of class modules from user.

A factory pattern is one of the core design principles to create an object, allowing clients to create objects of a library in a way such that it doesn’t have tight coupling with the class hierarchy of the library.

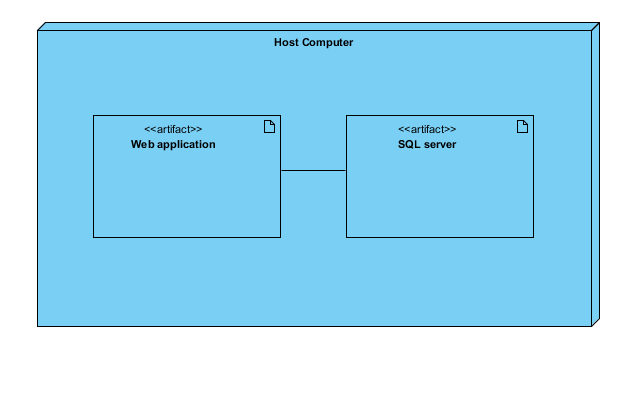
Thus, we can observe the design in the following diagram relative to my project:



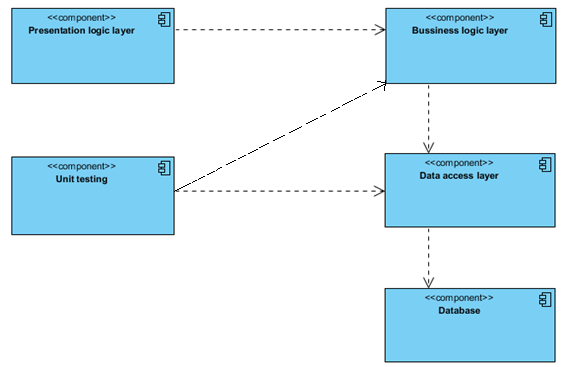
**Conceptual architecture diagram:**



**Deployment diagram:**



**Component diagram:**



# Construction and Transition

# System Testing

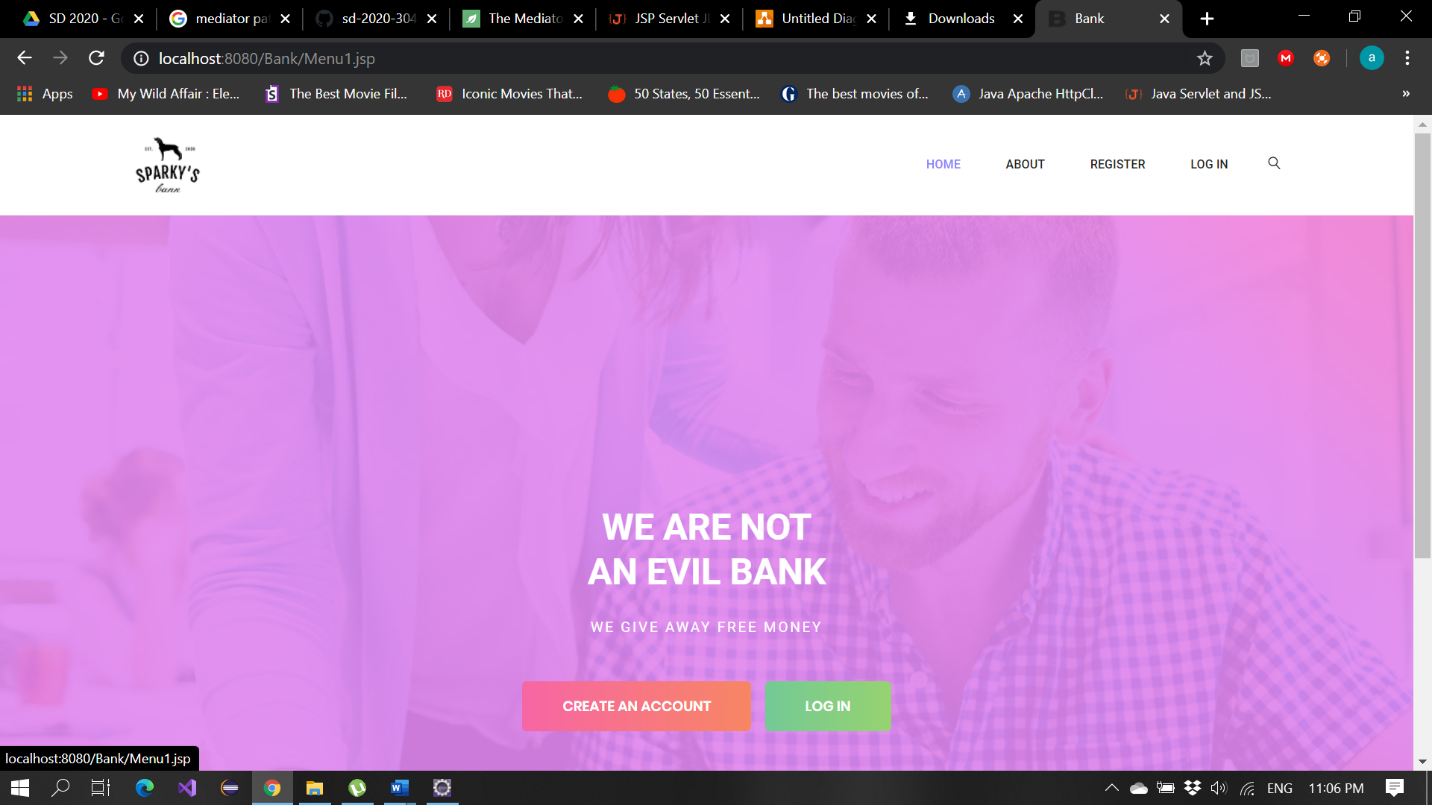
Test cases considered:

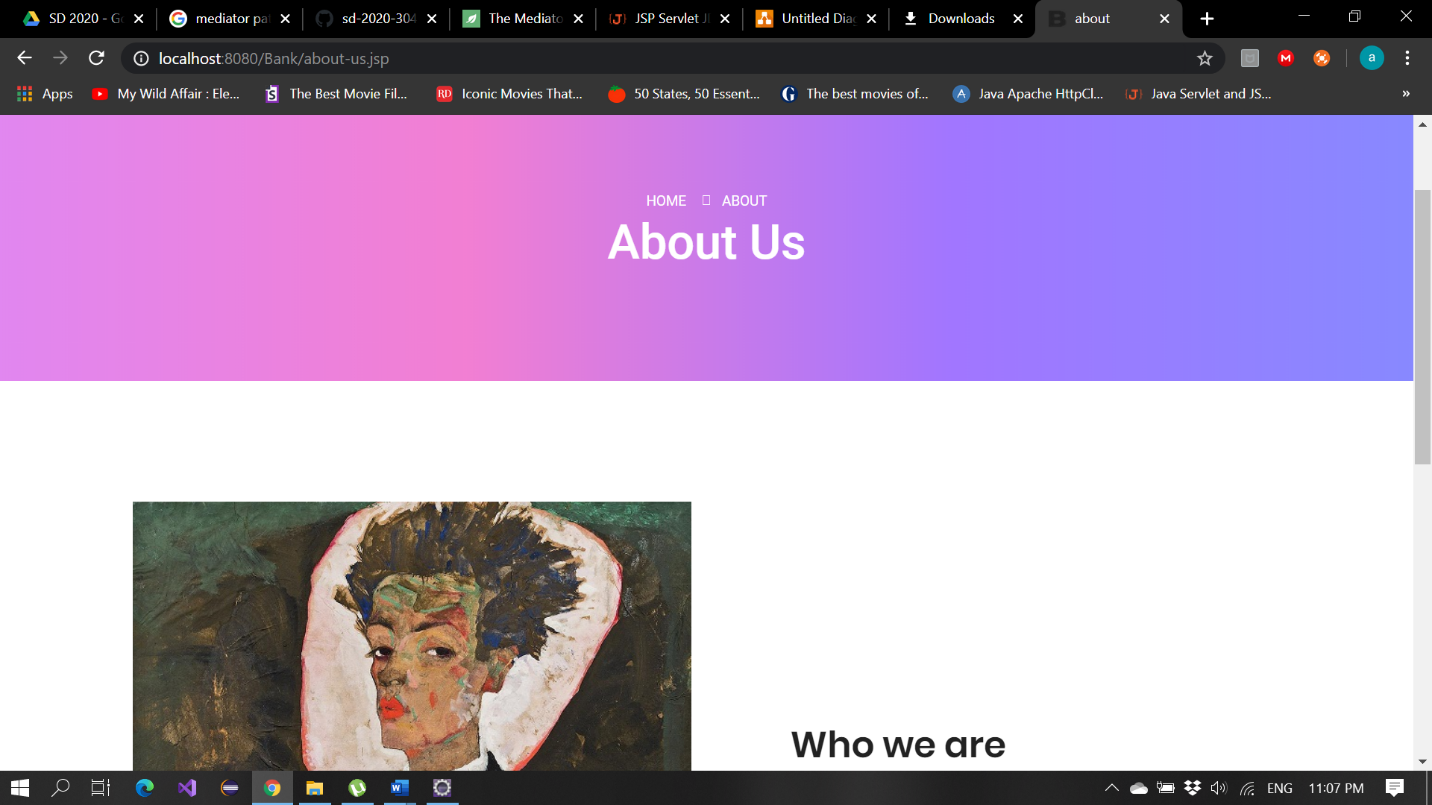
1. Testing the functionality of customer operations:
   * Whether the user can see the transactions history
   * Whether the user can enquire his account balance
   * Whether the user can transfer money to other accounts
   * Whether the user can navigate through the home page and choose what he wants to do next
2. Testing the functionality of the register page
   * Whether the user can fill in the data
   * Whether when submitting the data the validators prevent entering wrong data
   * If the user submits valid data, whether it is registered

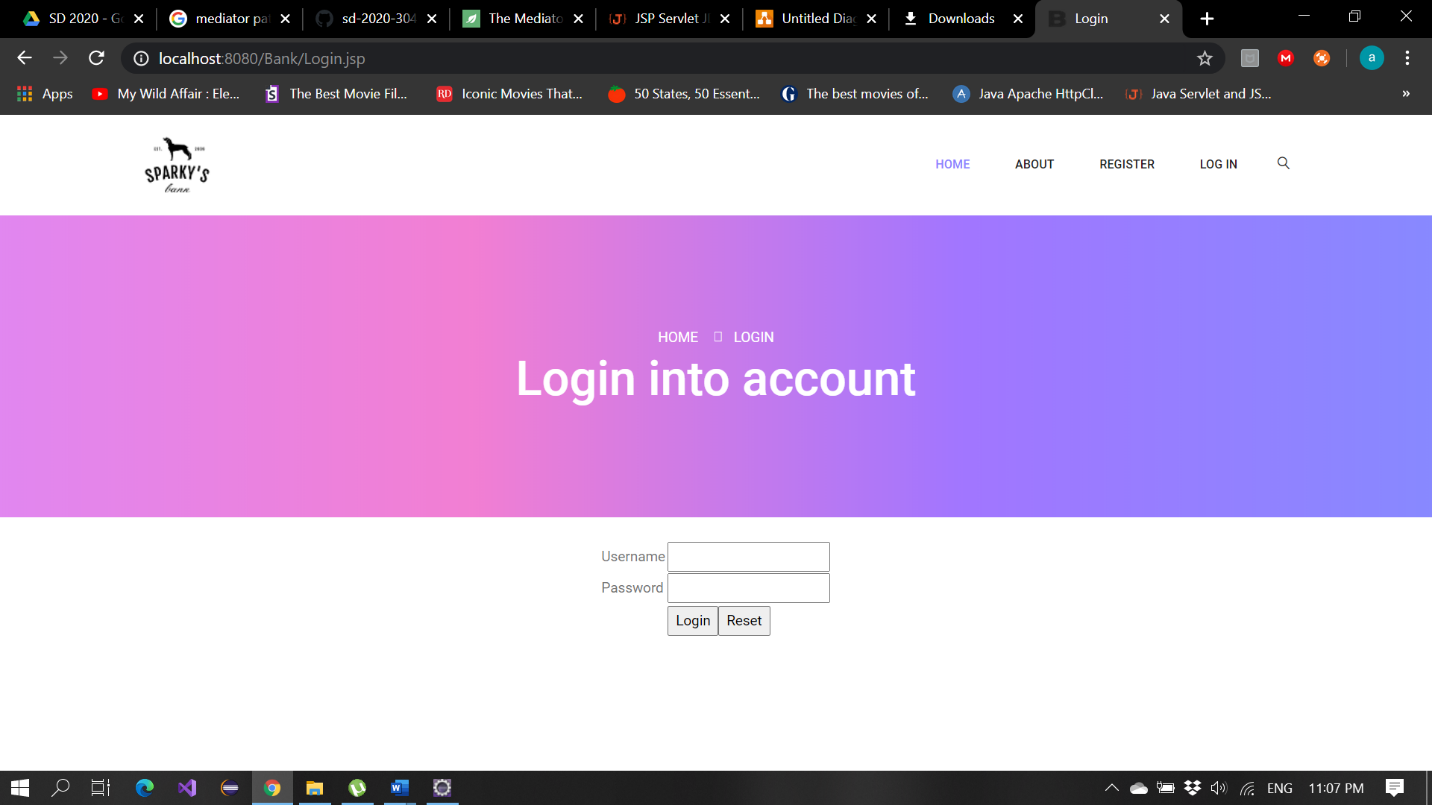
* This can be verified by the admin.

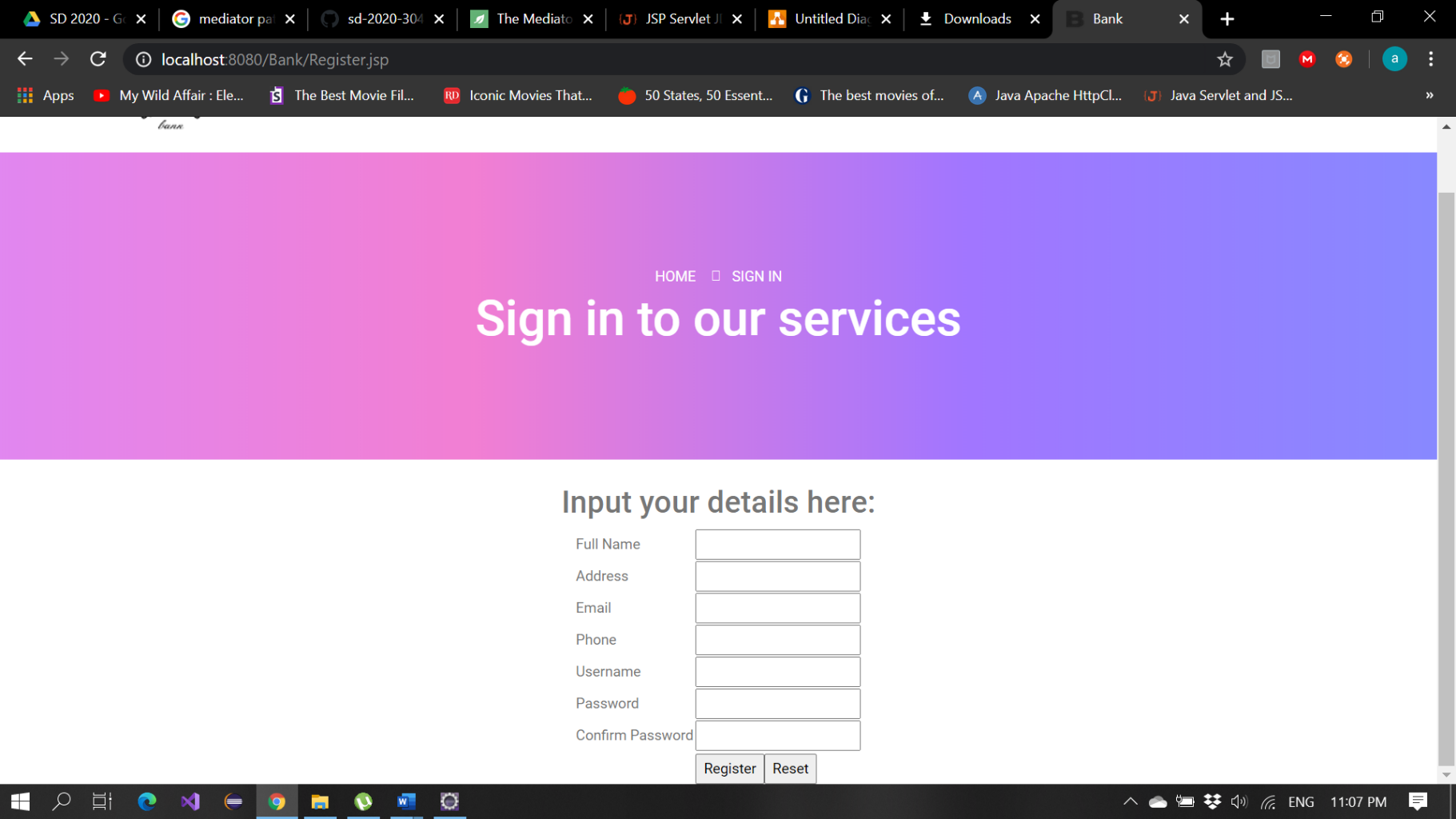
1. Testing the functionality of the login page
   * If the user fills in the password that corresponds to the password in the database tied to that user account
   * If the user fills in the username that corresponds to the username in the database tied to that user account
   * He can login into his account
2. Testing the functionality of the admin page
   1. Whether the admin can see all the transactions that took place over the webpage
   2. Whether the admin can register a new customer to the application based on request
   3. Whether the admin can delete a customer from the database in case he didn’t respect the policy or based on demand

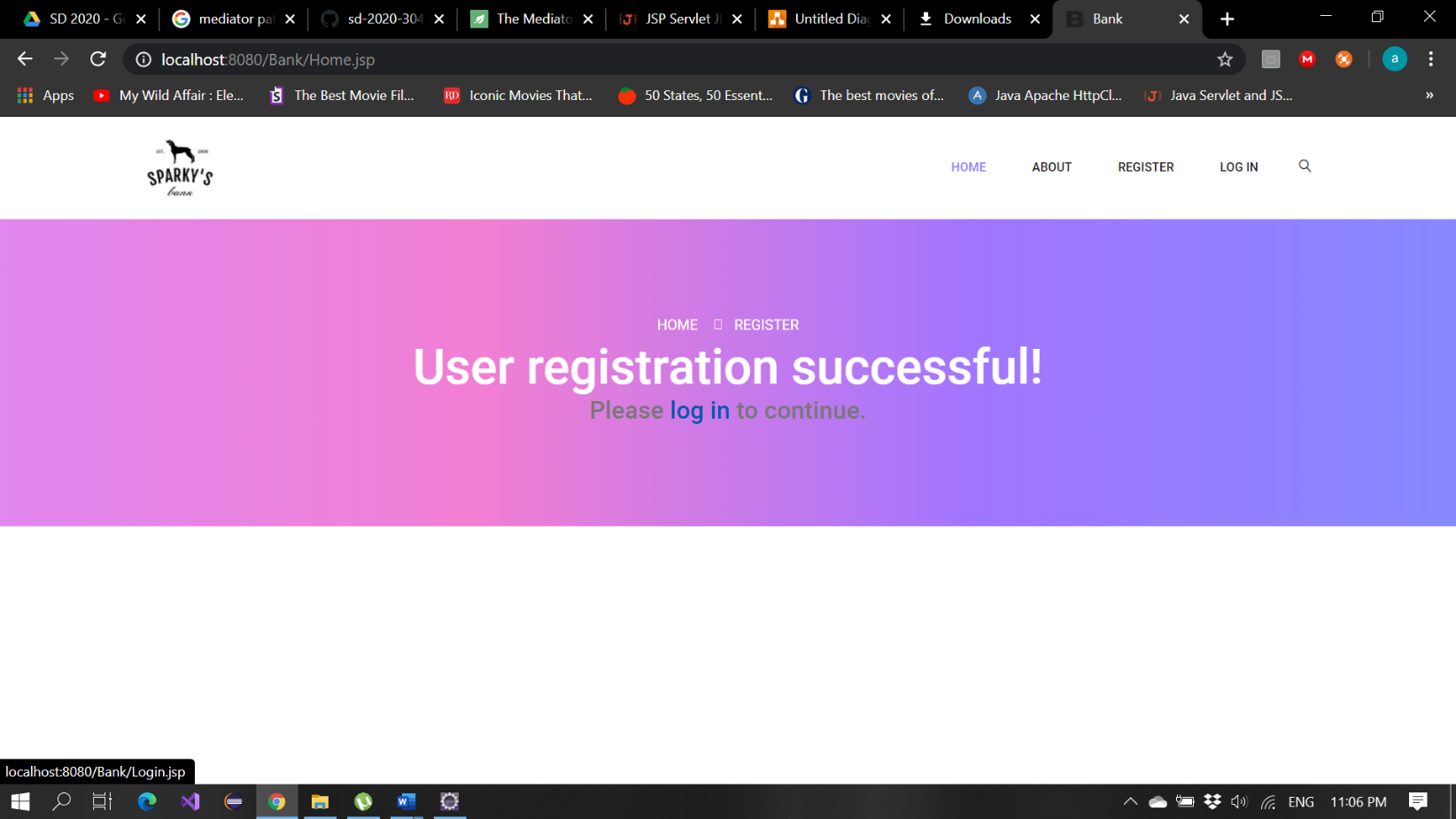
Here are some screenshots with the web application interface:

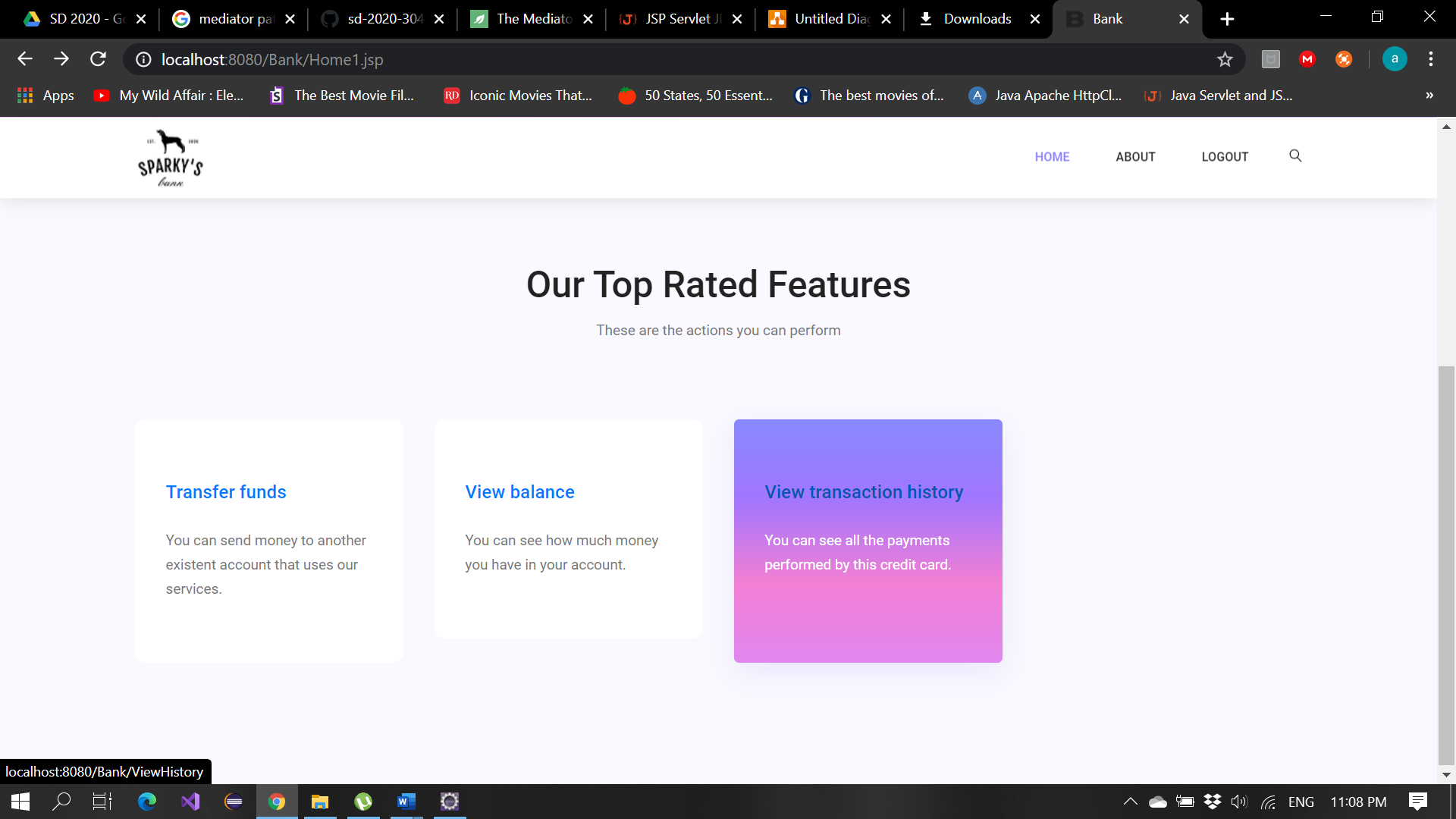


**

**

**

**

**

# Future improvements

The system can be further improved by offering EUR (and other currencies) accounts, free-free currency exchange and cryptocurrency exchange, features that will make the application useable at an international scale and available globally.

# Bibliography

* <https://mail.codejava.net/coding/java-servlet-and-jsp-hello-world-tutorial-with-eclipse-maven-and-apache-tomcat>
* <https://mail.codejava.net/coding/jsp-servlet-jdbc-mysql-create-read-update-delete-crud-example>