NANYANG TECHNOLOGICAL UNIVERSITY

SINGAPORE

Wee Kim Wee School of Communication and Information

CI6225 – Enterprise Application Development

Individual Project -2

Online Banking Application with Spring boot and Microservices

Submitted by

Rashmi Vishwanath(G1801881J)

Introduction

An Online Banking system is a model Internet Banking Site. It enables the customers to perform basic banking transactions providing right service at the right time through right channel. Being “electronic”, not only provides its customers with faster and better facilities, it even reduces the manual overhead of accounts maintenance. It provides enormous benefits to consumers in terms of ease and cost of transactions. With Internet Banking, the brick and mortar structure of the traditional banking gets converted into a click and portal model, thereby giving a concept of virtual banking a real shape.

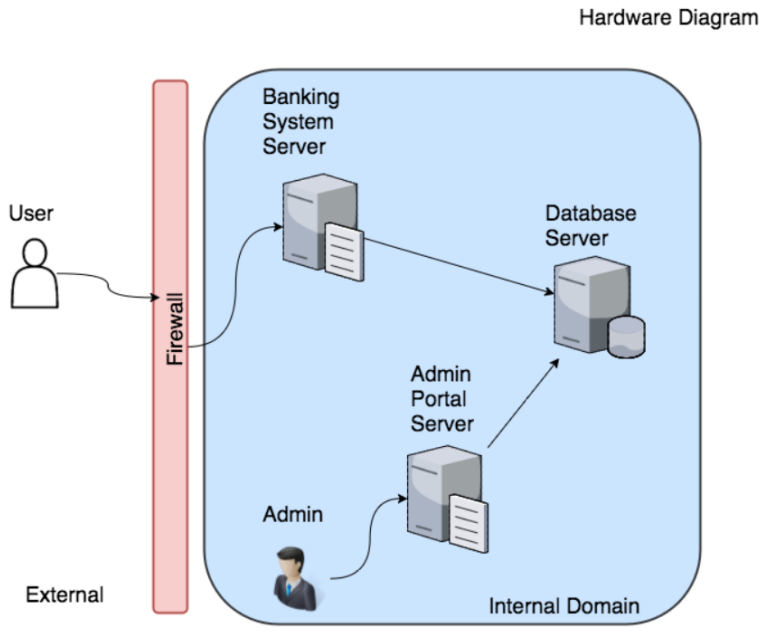
The primary aim of this application is to provide an improved design methodology, which envisages the future expansion, modification which is necessary for banking sector. The System provides the access to the customer to create an account, transfer the amount between current and savings accounts, add the beneficiary and transfer amount to beneficiary account. Anybody who is an account holder in this bank can become a member of Online Banking System.

Objective

Customer Satisfaction: Clients can do their operations comfortably without any risk or loosing of his privacy

Saving Customer Time: Client doesn’t need to go to banks to do small operations

Architecture



System Requirements

Software Requirements	*Hardware Requirements*
Operating System: Windows or Linux or MAC Processor	Processor: Any
User Interface: HTML, CSS, JavaScript, Bootstrap, Ajax	Hard Disk: 10 GB minimum
Back end: Java, Spring boot, Maven, Rest API Tools Used: Eclipse (STS) and Postman	RAM: 256MB or more
Database: MYSQL	Any Screen

Data Dictionary

Table Name: USERDB
Description: This table is used to store the User details

Key	**Field Name**	**Data Type**	**Nullable**
PK	User ID	INT	No
	Username	VARCHAR	No
	Password	VARCHAR	No
	First Name	VARCHAR	No
	Last Name	VARCHAR	No
	Email	VARCHAR	No
	Phone Number	VARCHAR	No
	Savings Account Number	INT	No
	Current Account Number	INT	No
	SB Account Balance	DECIMAL	No
	Current Account Balance	DECIMAL	No

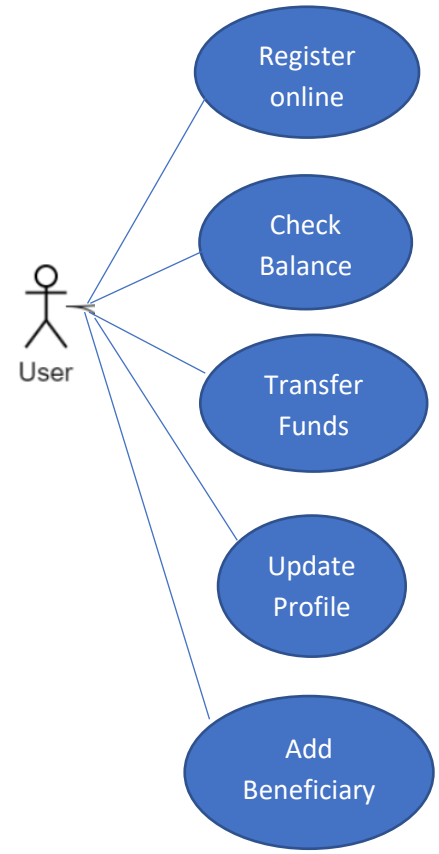
Table Name: RECIPIENT
Description: This table is used to store the Recipient details

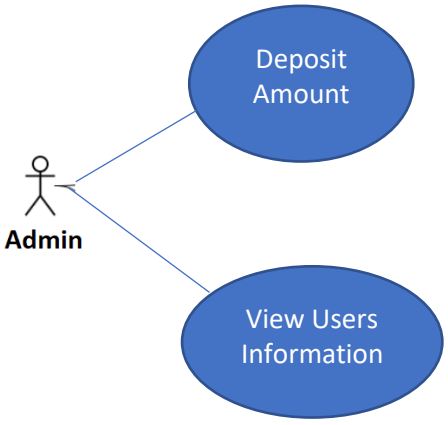
Key	Field Name	Data Type	Nullable
FK	User ID	INT	No
PK	Recipient ID	INT	No
	First Name	VARCHAR	No
	Last Name	VARCHAR	No
	Email	VARCHAR	No
	Phone Number	VARCHAR	No
	Account Type	VARCHAR	No
	Account Number	DECIMAL	No

Table Name: ADMIN
Description: This table is used to store the Admin details

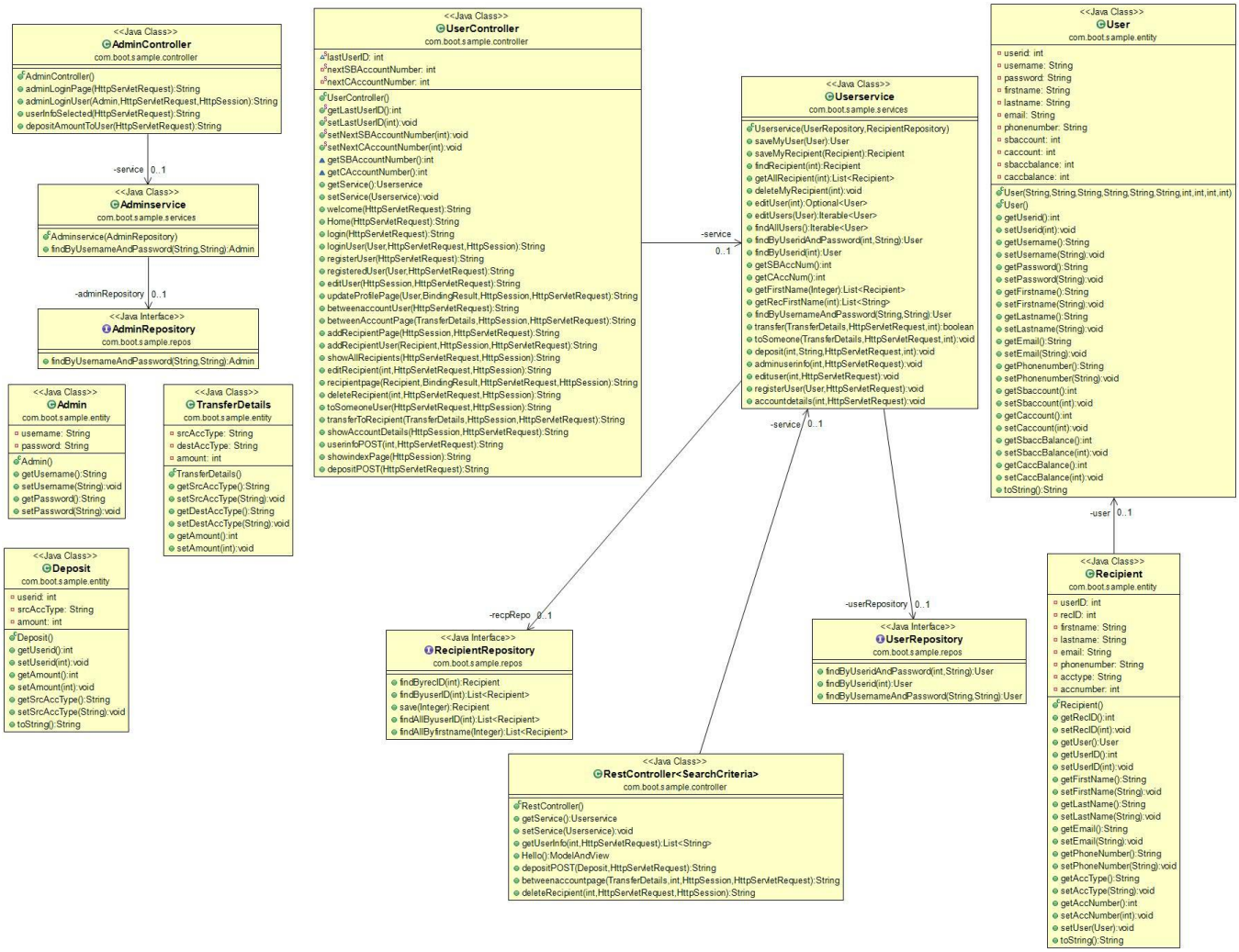
Key	Field Name	Data Type	Nullable
PK	username	VARCHAR	No
	password	VARCHAR	No

Use Case Diagram :

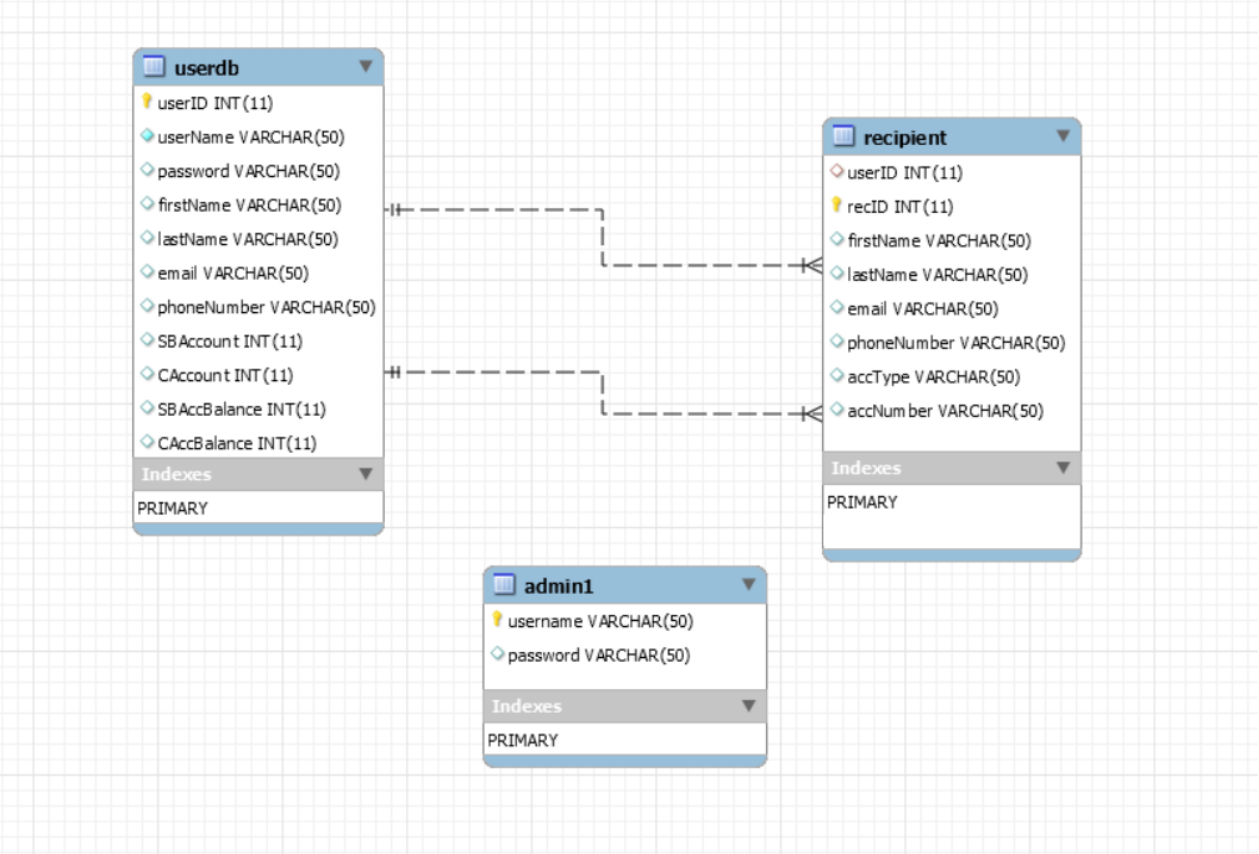




Class Diagram: It describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects.



Entity-Relationship Model: It shows the relationships of entity sets stored in a database



Phase I:

- ✓ Application was developed using Spring MVC,HTML,CSS
- ✓ User Features:
  - User Registration
  - Login
  - Update Profile
  - Transfer Amount between accounts (Savings to Current and vice versa)
  - Transfer Amount to other beneficiaries
  - Add Recipient
  - View Recipient
  - Edit and Delete Recipient details

Phase II:

- ✓ Application was developed using Spring Boot, Restful Web Services, Ajax, jQuery were used in JSP Pages to access Rest Service URL
- ✓ User Features:
  - Same as Phase I(Converted from Spring MVC to Spring boot)
- ✓ Admin Features:
  - Deposit Amount in Customer account
  - View Customer details like Account Numbers and Balances

Implementation:

Phase II: In the second phase, whole project is built based on Spring boot, Rest API as a microservice, JPA repository for database interactions as backend. HTML5, CSS, Bootstrap are used for UI (Front end)

Spring boot*:* Spring Boot is an extension to the Spring framework which eliminates the boilerplate configurations required for setting up a Spring application. Some of the features which make Spring boot more efficient are:

- Embedded server to avoid complexity in application deployment
- Automatic config for Spring functionality – whenever possible
- It provides embedded container support
- Absolutely no code generation and no requirement for XML configuration
- We have different annotations supported by Spring boot

Spring boot mainly requires only one dependency other than Spring:

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
    <version>2.0.5.RELEASE</version>
</dependency>
```

All other dependencies are added automatically to the final archive during build time.

Another good example is testing libraries. We usually use the set of Spring Test, JUnit and Mockito libraries. In a Spring project, we should add all these libraries as dependencies.

But in Spring Boot, we only need the starter dependency for testing to automatically include these libraries.

Spring Boot provides several starter dependencies for different Spring modules. Some of them used in this project are:

- spring-boot-starter-data-jpa
- spring-boot-starter-tomcat
- spring-boot-starter-test
- spring-boot-starter-web

JPA Repository:

CrudRepository is a **Spring Data interface for generic CRUD operations on a repository of a specific type.** It provides several methods to interact with a database. Most of the existing methods are used to perform CRUD operations.

By extending from the Spring CrudRepository, we can implement few methods, which reduces boilerplate code. Some of the methods used in our project are:

- <S extends T> S save (S entity); - Saves all given entities
- <S extends T> Iterable<S> saveAll (Iterable<S> entities); - Retrieves an entity by its id
- Optional<T> findById (ID id); - Returns whether an entity with the given id exists
- Iterable<T> findAll (); - Returns all instances of the type with the given IDs
- void deleteById (ID id); - Deletes a given entity

Microservice:

I have used two microservices in the entire application at the Admin Module:

There are primarily four types of request which can be made using REST Services: (a) GET, (b) POST, (c) PUT, and (d) DELETE

GET is used to retrieve the information. Whenever the request is made, it sends the response depending upon the user passed parameters, it passes the response in JSON

POST is used to create a new resource. It is also used to create subordinate resources. The main advantage of a POST call is that we can send the request in headers and bind the data

PUT is used to update any resource rather than creating new resource. It contains the data of the resource which is going to be updated.

Delete is used to delete any resource or record, we will pass the resource id which needs to be deleted.

I have mainly used two microservices in the Admin Module operations such as Depositing Amount and Viewing User Information.

- Deposit amount uses POST method i.e. saving the values entered in the UI
- Viewing User information uses GET method to fetch the values saved in DB

Deposit Amount Flow as a microservice: The following code is written in UserController

```
/*****************************************************************************
 *
 * @param request   - HttpServletRequest Object
 * @return          - Returns to deposit.jsp
 * Description     - This function will return to deposit.jsp page where ajax code
 *                     for Rest API is executed and update the user with deposited
 *                     amount.
 *
 *****************************************************************************/
```

```
@GetMapping("/deposit-page")
public String depositPOST(HttpServletRequest request)
{
    request.setAttribute("mode","MODE_DEPOSIT");
    return "deposit";
}
```

The following ajax code will be executed which directs to the rest controller URL mentioned here in the code:

```
<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.5.1/jquery.min.js"></script>
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.0/js/bootstrap.min.js"></script>

<script>
$(document).ready(function () {
    });

function fire_ajax_submit() {

    var data = {};
    data.userid = $("#userid").val();
    data.srcAccType = $("#srcAccType").val();
    data.amount = $("#amount").val();

    $.ajax({
                type : "POST",
                contentType: "application/json",
                url : "http://localhost:8080/bank/deposit-amounts",
                data : JSON.stringify(data),
                /* data: value, */
                async: true,
                success :function(response){
                    console.log(response);

                    alert("Amount has been deposited successfully");
                },

            });
    }

</script>
```

This function from the Rest Controller is called which saves the values in database

```
@PostMapping("/deposit-amounts")
public String depositPOST(@RequestBody Deposit deposit,HttpServletRequest request)
{
    service.deposit(deposit.getUserid(),deposit.getSrcAccType(),request,deposit.getAmount());
    return "Ok";
}
```

Viewing User information as a microservice: The following code is written in UserController

```
/**********************************************************************************************************************
 *
 * @param userid    - UserID contents to display in UI
 * @param request   - HttpServletRequest object
 * @return          - Returns to userinfo.jsp page with the information of this user
 * Description      - This method is using Microservices, returns the jsp page which has
 *                      ajax with rest controller URL
 *
 **********************************************************************************************************************/

@PostMapping("/adminuserinfo")
public String userinfoPOST(@RequestParam("userid") int userid,HttpServletRequest request )
{
    service.adminuserinfo(userid,request);
    return "userinfo";
}
```

The following ajax code will be executed which directs to the rest controller URL mentioned here in the code:

```
<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.5.1/jquery.min.js"></script>
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.0/js/bootstrap.min.js"></script>

<script>
$(document).ready(function(){
$.ajax({
    type:"GET",
    url :"http://localhost:8080/bank/db/users/"+${userid},
    async: true,
    /*data:value, */
    success :function(response){
    //  console.log(response);
    //  console.log (response[0]);
            var user_data =  '';
            user_data += '<tr>';
            user_data +='<td>'+response[0]+'</td>';
            user_data +='<td>'+response[1]+'</td>';
            user_data +='<td>'+response[2]+'</td>';
            user_data +='<td>'+response[3]+'</td>';
            user_data +='<td>'+response[4]+'</td>';
            user_data +='<td>'+response[5]+'</td>';
            user_data +='</tr>';
        //console.log(user_data);
        $('table').append(JSON.stringify(user_data));
    },
    error: function(jqXHR, textStatus, errorThrown) {
        alert(jqXHR.status);
    }
    });
});
```

This function from the Rest Controller is called which fetches values from the database

```
    @GetMapping("/db/users/{userid}")
    public List<String> getUserInfo( @PathVariable("userid") int userid, HttpServletRequest request )
    {
        User _cUser = service.findByUserid( userid );
        List<String> _lUserList = new ArrayList<String>();
        if( null != _cUser )
        {
            Integer _iUserID = _cUser.getUserid();
            _lUserList.add( _iUserID.toString() );

            _lUserList.add( _cUser.getUsername() );

            Integer _iSBAccount = _cUser.getSbaccount();
            _lUserList.add( _iSBAccount.toString() );

            Integer _iSBBalance = _cUser.getSbaccBalance();
            _lUserList.add( _iSBBalance.toString() );

            Integer _iCAccount = _cUser.getCaccount();
            _lUserList.add( _iCAccount.toString() );

            Integer _iCAccBalance = _cUser.getCaccBalance();
            _lUserList.add( _iCAccBalance.toString() );
        }

        request.setAttribute("mode","MODE_REGISTER");
        return _lUserList;
    }
```

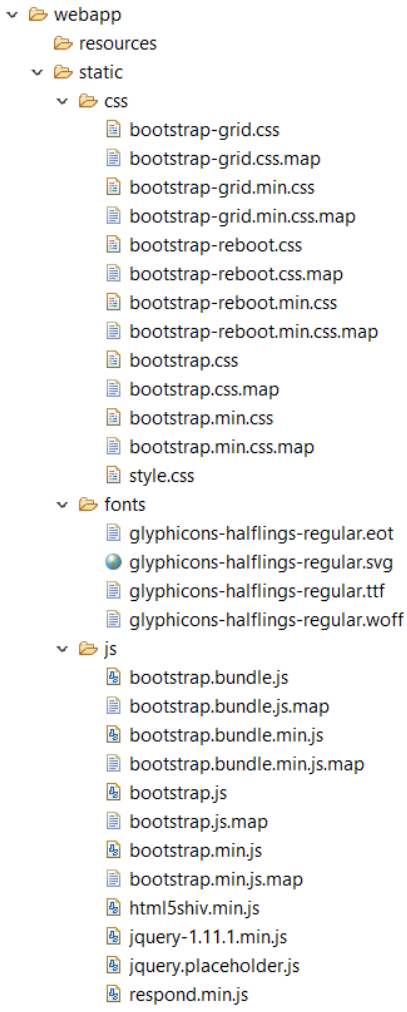
Also, few of the functions are made as microservice and which can be executed via postman tool –

```java
/* Through Postman it works */
@PutMapping("/TransferBetweenAccount/{userid}")
public String betweenaccountpage( @RequestBody TransferDetails transferdetails,@PathVariable("userid") int userid,HttpSession session, HttpServletRequest request )
{
    if( true == service.transfer(transferdetails,request,userid) )
    {
        return "Success";
    }
    else
    {
        return "fail";
    }
}

@DeleteMapping("/delete-recipient/{recID}")
public String deleteRecipient(@PathVariable int recID, HttpServletRequest request, HttpSession session)
{
    service.deleteMyRecipient(recID);
    return "Deleted";
}
```

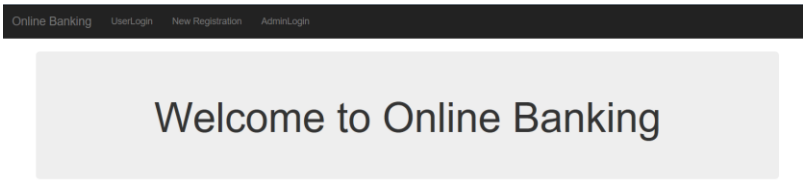
User Interface

HTML5 and CSS are used to create UI templates. Bootstrap template used as follows:

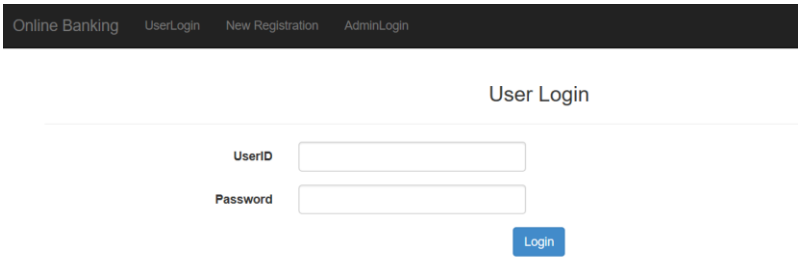


Sample Snapshots:

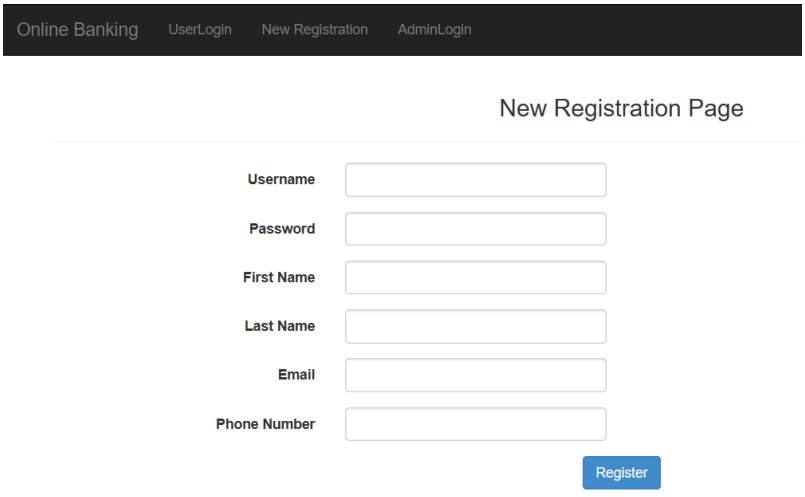
Index Page



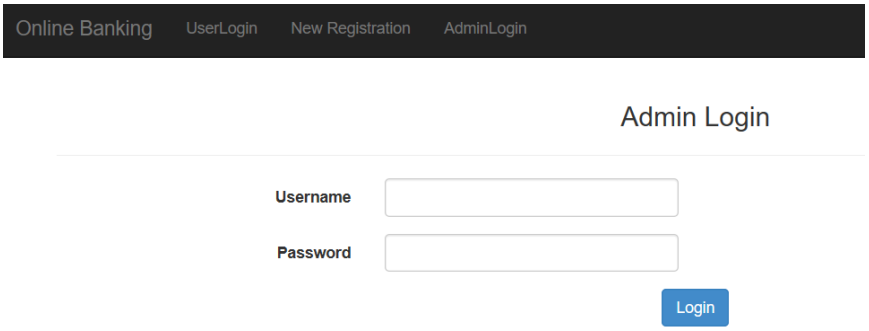
User Login



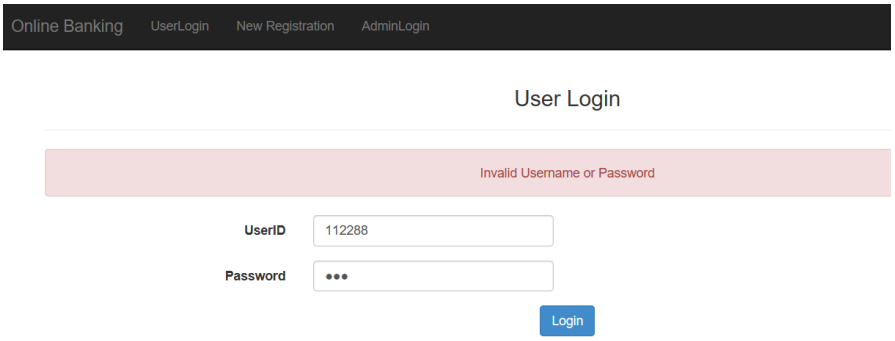
User Registration



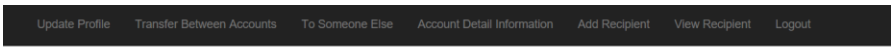
Admin



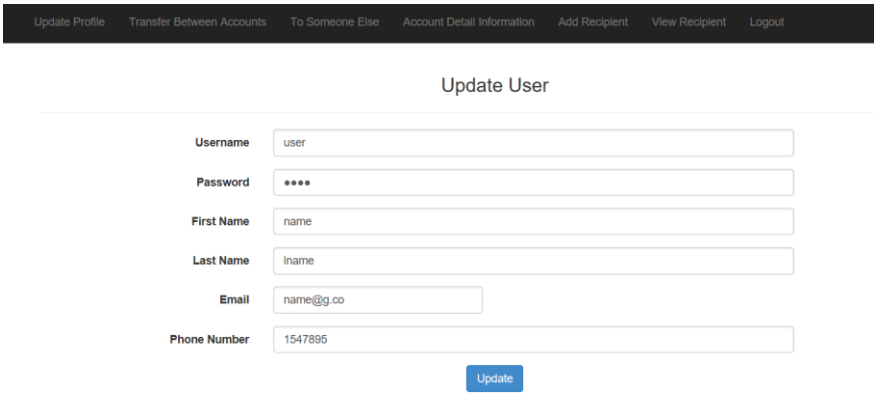
Invalid Details on User Login



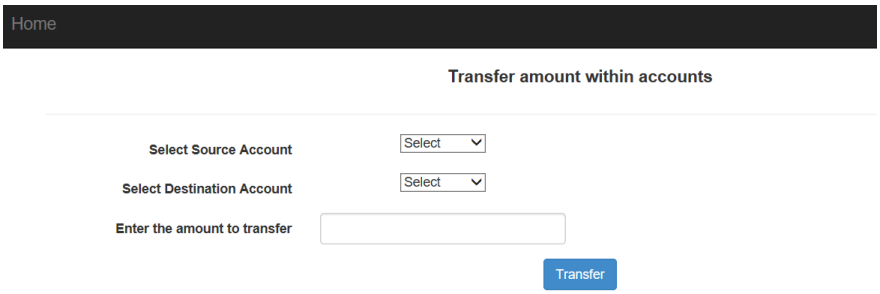
Home Page



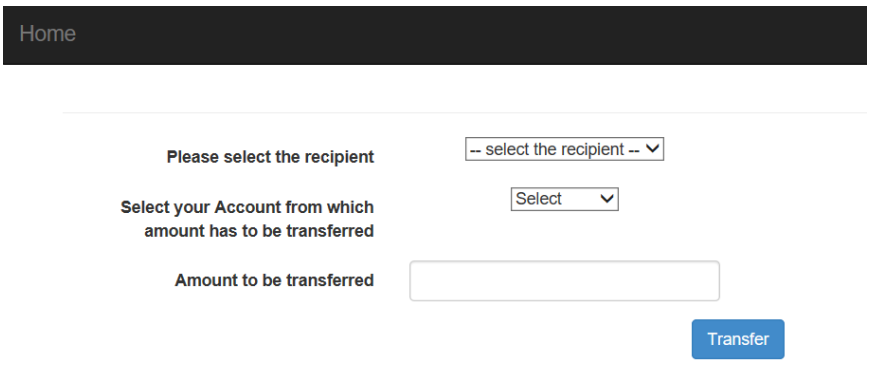
Update User



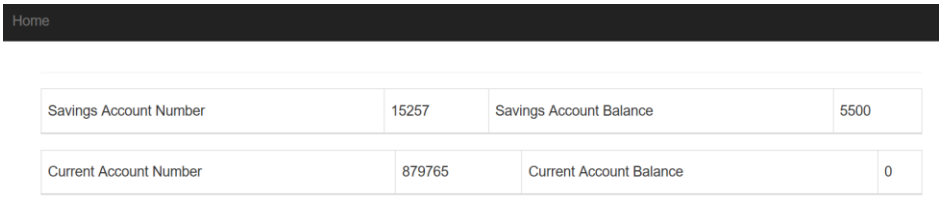
Transfer Between Accounts



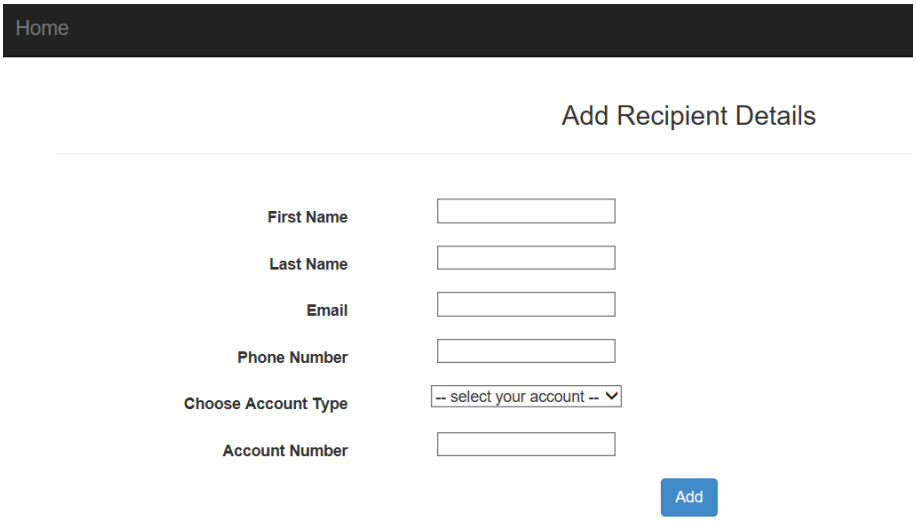
Transfer to Beneficiary



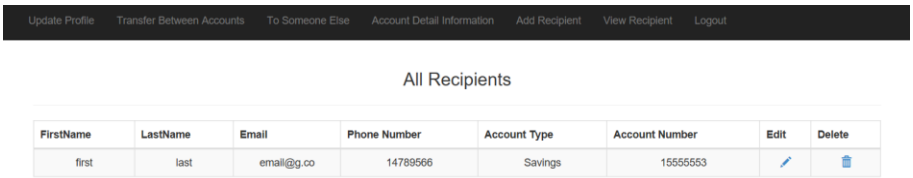
Account Detail Information



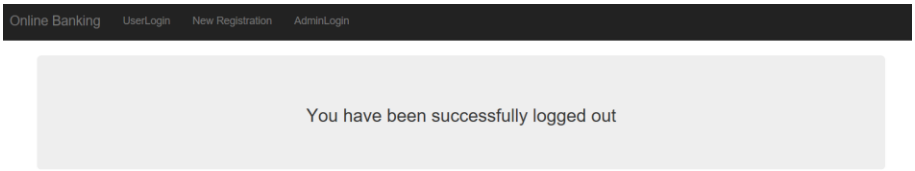
Add Recipients



View Recipients



Logout – Included session validation after logout



Conclusion

This project is developed to nurture the needs of a user in a banking sector by embedding all the tasks of transactions taking place in a bank. Online banking is an innovative tool that is fast becoming a necessity. The website is made friendlier from where the first-time customers can directly make and access their accounts. By using the mentioned technologies, we can improve the performance of the application. Rest Services and Spring boot ensures the application to be modular. Thus, the Online Banking System is developed and executed successfully using Spring boot and rest API.

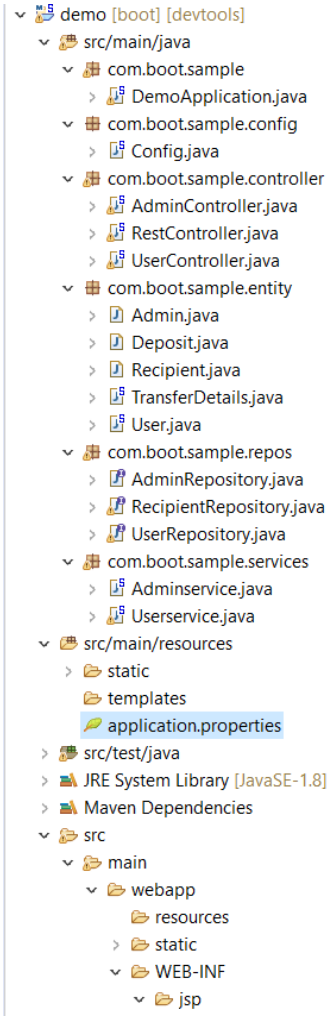
User Guide

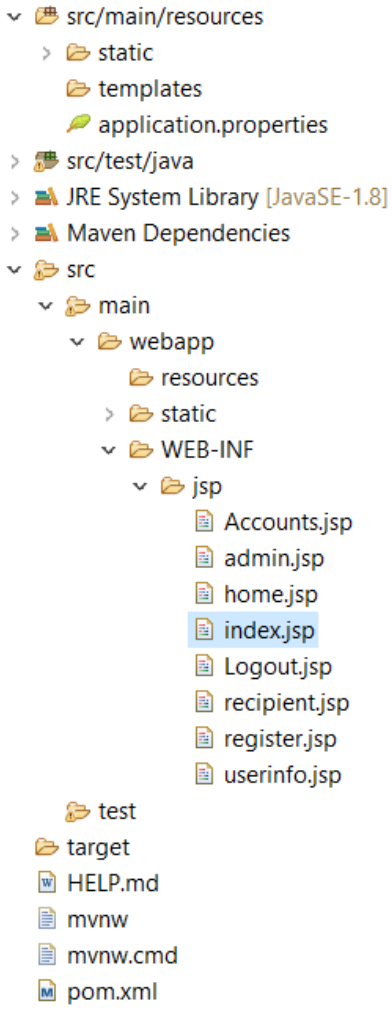
Getting Started –

To start the application, we need to start the UI and the database. We need Spring Tool Suite (STS) and MySQL to be installed

The first page you see is the Index page which contains several features for the Online Banking System.

Project Structure looks as -





Few details about the structure –

Com.boot.sample.controller -This package contains the controller classes for User and Admin. Rest Controller is explicitly written for microservice

Com.boot.sample.entity - This package contains all the POJO classes

Com.boot.sample.repos – This package contains Interfaces for Admin, Recipient and User extending CRUD Repository respectively

Com.boot.sample.service – This package contains Services for User and Admin with business logic

Application.properties – This file has DB connection details, path to JSP pages

Pom.xml – This file has all dependencies related to Database, web, spring boot, tomcat server etc.

User Actions

Task 1: Register

- ✓ Click the New Registration on the index page
- ✓ Fill up all the fields along with the password and submit
- ✓ A unique userID will be generated for newly registered users and displayed on the UI. This is like providing Customer id to the new customers

Task 2: User Login

- ✓ Login with your userID (displayed on the UI while you registered) and password
- ✓ Home page will be displayed with
    - Transfer Between Accounts
    - To Someone Else
    - Account Detail Information
    - Add Recipient
    - View Recipient
    - Logout

Task 3: Transfer Between Accounts

This allows customer to transfer the amount between his own accounts. It has 'Home' button to go back to the Home page. Steps to transfer amount between accounts:

- ✓ Select Source Account Type, Destination Account Type
- ✓ Enter the amount you want to transfer
- ✓ Click Transfer to send the amount

Task 4: To Someone Else

This allows customer to transfer the amount to added beneficiary (Recipients). It has 'Home' button to go back to the Home page. Steps to transfer amount to recipients in the list:

- ✓ Choose the recipient from the dropdown list
- ✓ Select the account you would like to transfer from
- ✓ Enter the amount you want to transfer
- ✓ Click Transfer to send the amount

Task 5: Account Details Information

- ✓ It displays both Savings and Current account numbers and their respective balances

Task 6: Update User Profile

- ✓ Logged in User details will be auto populated when clicked on update profile button
- ✓ User can update the values and click update
- ✓ Profile will be updated

Task 7: Add Recipient

This module enables Customers to add the beneficiary details. It has 'Home' button to go back to the Home page

Input

- ✓ Enter all the required details like First Name, Last Name, Email, Phone Number, Account Type, Account Number
- ✓ Click Add button

Output

Recipient would be added successfully, and you can view the recipient list in To Someone Else page while transferring amount

Task 8: View Recipient

This section displays all the recipients added for that specific Customer. Each Recipient details can be updated and deleted by the Customer

- ✓ Clicking on Edit icon, loads the page with recipient details and Customer can update the details
- ✓ Clicking on Delete icon, deletes the recipient and page will be refreshed displaying rest of the recipients.

Task 9: Logout

- ✓ Clicking on Logout button, invalidates the session and user will be logged out of the application

Admin Actions

It is mainly used by Admin. Clicking on ***Admin Login*** from index page, loads the login page for the admin to login with username and password. It involves -

Task 1: Deposit Amount

Admin can deposit the amount for the newly registered customers.

Input

- ✓ Enter the user id of a customer to whom the amount must be deposited
- ✓ Select the account to which the amount must be deposited (Savings or Current)
- ✓ Enter the Amount to be deposited

Output
Amount will be successfully deposited to the specified customer. User can login to check his balance for the deposited amount.

Task 2: User Information

Admin can view the Customer's information

Input

- ✓ Enter the user id of a customer to view the details

Output
Customer's information like User id, Username, Account Number and Balance are displayed for the specific user

Open the below file to install all the necessary software and configurations required to run this application


5. Preparing the Spring Web Developr


STS tool installation.docx

Open the code and Run the application as – Right Click on the Project and Run as Spring boot app– You will be able to navigate to the mentioned pages.