Project: ONLINE BANKING SYSTEM

Back End Development

|  |  |  |
| --- | --- | --- |
|  | **FUNCTIONAL SPECIFICATION** | |
|  |  |  |
| **Project Code:** |  |  |
| **Project Name:** |  | ONLINE BANKING SYSTEM |

**FUNCTIONAL SPECIFICATION**

**Table of Contents**

1. INTRODUCTION 3

2. SYSTEM OVERVIEW 3

3. SUB-SYSTEM DETAILS 4

4. DATA ORGANIZATION 5

5. REST APIs to be Built 6

6. ASSUMPTIONS 9

7. EXPECTATIONS 9

8. ACCEPTANCE CRITERIA 9

9. TRACEABILITY TO REQUIREMENTS 9



**FUNCTIONAL SPECIFICATION**

**1 Introduction**

XYZ Ltd is a company which builds a software system which is responsible for making transactions online.

XYZ Ltd plans to develop "ONLINE BANKING SYSTEM" - web application [J2EE Batches - Web Application], where users can register, login, add payee , transfer funds, manage bank accounts.

**Scope and Overview:**

The scope of the “ONLINE BANKING SYSTEM” will be to provide the functionality as described below. The system will be developed on a Windows operating system using Java/J2EE, Hibernate, Spring, Angular, oracle database.

**2 System Overview**

The “ONLINE BANKING SYSTEM” should support basic functionalities (explained in section 2.1) for all below listed users.

* Administrator (A)
* User(U)

***2.1 Authentication & Authorization***

***2.1.1 Authentication****:*

Any end-user should be authenticated using a unique username and password.

***2.1.2*** ***Authorization***

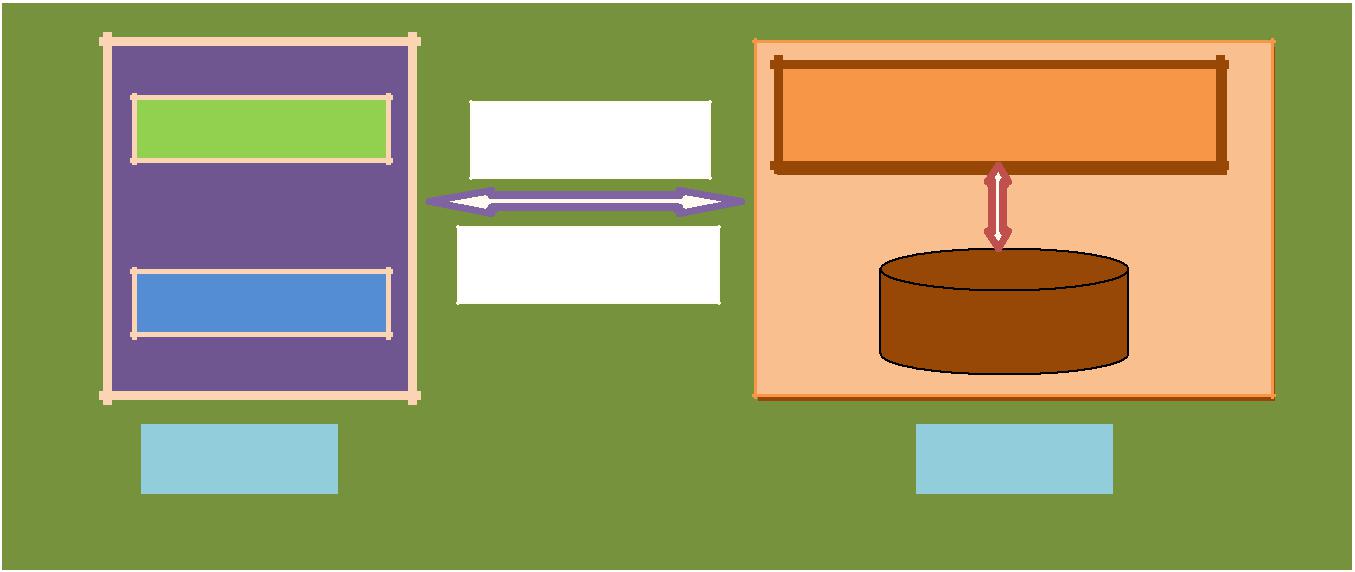
The operations supported and allowed would be based on the user type. For example, Administrator has the rights to view all transactions and can perform crud operations. He can also add and remove bank branches if necessary.

Whereas User has a right to Add payee ,transfer amount ,balance checking.

***2.2 Functional Flow***

The functional flow of the messages across different application components is shown below.

Ex. - Web Application.



|  |  |  |
| --- | --- | --- |
| **Administrator** | **OB Application** |  |
| HTTP/GET |  |
|  | HTTP/Response |  |
| **User** | Database |  |
|  |  |
| Client GUI | Server |  |
|  | **ONLINE BANKING SYSTEM** |  |
|  |  |  |
| ***2.3 Environment*** |  |  |

The system will be developed on any Windows OS machine using J2EE, Hibernate and Angular, Oracle database, Spring.

* Intel hardware machine (PC P4-2.26 GHz, 512 MB RAM, 40 GB HDD)
* Visual studio code
* Server – Apache Tomcat 8 or higher
* Database – Oracle 11g or higher
* JRE 8
* Eclipse IDE or Spring Tool Suite



**FUNCTIONAL SPECIFICATION**

**3 Sub-system Details**

The Online Banking System is defined, wherein all users need to login successfully before performing any of their respective operations.

Find below (section 3.1 & 3.2) tables that provides functionality descriptions for each type of user / sub-system. Against each requirement, indicative data is listed in column ‘Data to include’. Further, suggested to add/modify more details wherever required with an approval from customer/faculty.

***3.1 Administrator***

The administrator as a user is defined to perform below listed operations after successful login.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | Objects | Operations | Data to include | Remarks |
| AD-001  To  AD-004 | Branch | Add  View  Delete  Modify | Branch id, Branch Name ,Branch City |  |
| AD-005  To  AD-0010 | Transactions | View | Transaction Id, Amount, DOT, Transaction Type, Account Number,Payee Id |  |

***3.2 User***

The user is defined to perform below listed operations after successful login.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ID | Objects | Operations | Data to include |  | Remarks | | | | |
| US-001 | User | Signup | Username, Password, Confirm password, Fname, Lname , Mobile num, City, Email, Role, OTP |  |  |  |  |  |  |
| US-002 | Payee | Add payee | Payee id, Bank name, Payee Account Number, Payee Name, User Name |  |  |  |  |  |  |
| US-003 | Transfer | Fund Transfer | From Account, Payee id , Transaction Type, Transaction date, amount |  |  |  |  |  |  |

**FUNCTIONAL SPECIFICATION**

***3.3 Login | Logout***

**[Web Application - J2EE, Hibernate, Spring]**

* Go to Login screen when you click on Register link
* If the user is new he/she will register in the sign up form
* After successfully registering then user will be redirected to login page
* when you login successfully after entering valid username & password, user will be redirected to home page.
* User will not login if the credentials fetched in the database are not matched
* Implement Session tracking for all logged in users before allowing access to application features. Anonymous users should be checked, unless explicitly mentioned.

**4 Data Organization**

This section explains the data storage requirements of the Online Banking System and **indicative** data description along with suggested table (database) structure. The following section explains few of the tables (fields) with description. However, in similar approach need to be considered for all other tables.

***4.1 Table: User\_Signup\_Details***

The user specific details such as username, email, phone etc. Authentication, and authorization / privileges should be kept in one or more tables, as necessary and applicable.

|  |  |
| --- | --- |
| **Field Name** | **Description** |
| username | Username of the user |
|  |  |
| *Password* | Password of the user |
| *Confirm password* | User Password confirmation |
| *First name* | User First Name |
| *Last Name* | User Last Name |
| *Mobile Number* | Users Mobile Number |
| *City* | City |
| *Email* | Users Email Address |
| *Role* | User or Admin roles |
| OTP | OTP validation |

***4.2 Table: Account\_Details***

This table contains information related to a Account

|  |  |
| --- | --- |
| **Field Name** | **Description** |
| *Account Number* | Unique account number for every user |
| *Cust Id* | Unique id for each customer |
| *Branch Id* | Unique id for every bank branch |
| *Balance* | Balance in a bank account |
| *Account Type* | Savings or primary account |



***4.3 Table:Payee\_Details***

This table contains information related to Payee Details

|  |  |
| --- | --- |
| **Field Name** | **Description** |
| Payee Id | Unique id of the payee |
| Bank name | Bank name of the payee |
| *Payee Account Number* | Payees Account number |
| *Payee Name* | Payee name |
| *User Name* | Payer name |

***4.4 Table: Fund\_Transfer***

This table contains information related to Fund transfer

|  |  |
| --- | --- |
| **Field Name** | **Description** |
| From Account | Account number of the payer |
| Payee Id | Unique id of the payee |
| Transaction type | Transaction type |
| Transaction date | Date of fund transfer |
| *Transaction Amount* | Amount that is transfered |



**FUNCTIONAL SPECIFICATION**

1. **REST APIs to be Built**

Create following REST resources which are required in the application,

1. Creating **User** Entity: Create Spring Boot with Microservices Application with Spring Data JPA

**Technology stack:**

* Spring Boot
* Spring REST
* Spring Data JPA

Here will have multiple layers into the application:

1. Create an Entity: User
2. Create a UserRepository interface and will make use of Spring Data JPA
3. Will have findByUserName method
4. Add the User details
5. Create a UserService class and will expose all these services
6. Finally, create a UserRestController will have the following Uri’s:

|  |  |  |  |
| --- | --- | --- | --- |
| URI | METHODS | Description | Format |
| /user/username | GET | Give a single user description searched based on username | JSON |
| /user | POST | Add the user details | JSON |
| /user | PUT | Update the user details | JSON |
| /user/username | DELETE | Delete user by Username | String |

1. Creating Account\_Details Entity:

Build a RESTful resource for **Account\_Details** manipulations, where CRUD operations to be carried out. Here will have multiple layers into the application:

1. Create an Entity: Account\_Details
2. Create a Account\_DetailsRepository interface and will make use of Spring Data JPA
3. Will have findByAccountNumber method
4. Add the addAccount method
5. Will have modifyAccount method
6. Will have deleteBranch method
7. Create a AccountDetailsService class and will expose all these services
8. Finally, create a AccountDetailsRestController will have the following Uri’s:

|  |  |  |  |
| --- | --- | --- | --- |
| URI | METHODS | Description | Format |
| /accountdetails/{Acc\_number} | GET | Get account details | JSON |
| / accountdetails | POST | Add new account into database | JSON |
| / accountdetails | PUT | Modify account details | JSON |
| / accountdetails /{Acc\_number} | DELETE | Delete an Branch | JSON |

1. Creating **Payee** Entity:

Build a RESTful resource for **Payee** manipulations, where following operations to be carried out. Here will have multiple layers into the application:

1. Create an Entity: Payee
2. Create a PayeeRepository interface and will make use of Spring Data JPA
   1. Add the addPayee details to add payees to the data base
   2. Will have getAllPayees method to get all the payees by the admin
   3. Will have findPayeeById method to find the payee details
   4. Will have modifyPayee method to modify details
   5. Will have removePayee method to remove payees
3. Create a PayeeService class and will expose all these services

Finally, create a PayeeRestController will have the following Uri’s:

|  |  |  |  |
| --- | --- | --- | --- |
| URI | METHODS | Description | Format |
| /payee | POST | Add new payees to the database by the user | JSON |
| /payee | GET | Find all payees of the user | JSON |
| /payee/{payeeId} | GET | Find payee by ID | JSON |
| /payee | PUT | Modify payee and update by the user | JSON |
| /payee/{payeeId} | DELETE | Delete payee by payeeid by the user | JSON |

1. Creating Transaction\_Details Entity:

Build a RESTful resource for **Checkout** manipulations, where following operations to be carried out. Here will have multiple layers into the application:

1. Create an Entity: Transaction\_Details
2. Create a TransactionDetailsRepository interface and will make use of Spring Data JPA
   1. Will have getAllPayees method
   2. Will have getAllTransactions method
   3. Will have fundTransfer method
   4. Will have modifyTransactions method
3. Create a TransactionDetailsService class and will expose all these services
4. Finally, create a TransactionDetailsRestController will have the following Uri’s:

|  |  |  |  |
| --- | --- | --- | --- |
| URI | METHODS | Description | Format |
| /transactiondetails/{payeeId} | GET | Find all payees In the database | JSON |
| / transactiondetails | GET | Find all transactions in the database | JSON |
| / transactiondetails | POST | Transfer amount from one account to another | JSON |
| / transactiondetails | PUT | Modify transaction details by the admin | JSON |

**FUNCTIONAL SPECIFICATION**

1. **Assumptions**

* User Interface: The type of client interface (front-end) to be supported - Angular based
* The administrator can View transactions and also perform crud operations on braches
* When the user transfers the amount ,the that should be deducted from his account and also reflected in the database.
* If user removed any payee then that will be deleted from the database
* User can add any number of payees and can transfer amount until amount becomes zero.
* User can perform crud operation on payees and admin can perform crud operations on branches

1. **General Expectations**

* Participants must create the **ER Diagram**.
* Participants must do **Functional Testing using POSTMAN tool**
* Integration of Angular and Spring Boot with Microservices should be done.
  + The server should be a concurrent server servicing multiple client.
  + Database can be implemented using Oracle 11g or above.
  + To begin with, the application should support at least 1 dmin and 2 Users.
  + Compilation and Build should be done using Eclipse IDE or STS
  + Source-code and all documents must be maintained (checked-in) in configuration management system (subversion)
  + Coding standards (for Java) should be followed

NOTE:

1. **Validation of user Data**

* Spring MVC using JSR-303 annotations
* ANGULAR form validations
* JavaScript validation (if necessary)

1. **UI Design – (for Web Application) Use DIV/CSS or Semantic Elements to control the style and layout**
2. **Create at least one SQL DML-statement inside PL/SQL blocks**

**8 Acceptance Criteria**

All P1 requirements must be mandatorily implemented

**9 Acronyms and Glossary**

Acronym and glossary for this document mentioned in the below table.

|  |  |
| --- | --- |
| **Abbreviation** | **Remark** |
| OBS | Online banking system |
| RS | Requirement Specification |
| FS | Functional Specification |

1. Validations should be performed at all levels of application appropriately.