Bank Management System

* **Problem statement**

The purpose of the requirements document is to systematically capture requirements for the project and the system “**Bank Management System**” to be developed. Both functional and non-functional requirements are captured in this document. It also serves as the input for the project scoping.

**About the System**

* The client would like to develop an independent application Bank Management System (BMS) application in order to automate the process of managing the activities of bank like opening an account, transaction etc.
* The following section will cover aspects related to Bank Management System.
* Customer Registration
* Apply Loan
* Update Account Details

**Scope of the System**

The scope of the system is explained through its modules as follows

* Customer Registration – used by customers to register the details of self-information into the system. The system stores the details of the customer in the system along with the account details.
* Apply Loan - will be used by registered customers to apply loan into the system. The system stores the loan details in the system along with the account details.
* Update Profile - will be used by registered customers to update the details of account holder into the system. The system should update the details of the account holder in the system along with the account details.
* **Skill Tower develop the project**

List the Technology based on your respective technology stack, that will be used to development the project.

Associate will choose any one of the technology stack and develop the application.

|  |  |
| --- | --- |
| Batch Name | Skill Names |
| Backend Java Microservices | Java  Web API (Spring Boot)  Microservices  Junit, Mockito  Git  MySQL  Kafka  Swagger |

* **Use Case/Architecture Diagram**

**Use Case Diagram**



**Sample Architecture Diagram**



**Flow Diagram**



* **User Stories**

|  |  |  |
| --- | --- | --- |
| **User Story #** | **User Story Name** | **User Story** |
| US\_01 | Customer Registration and Login | As a Customer, I should be able to register my details in the system so that I can login into the system  Acceptance criteria:  Customer should be able register the details in the system and it should be saved in the database.  Customer should be able to login with a User Id and Password that exists in database. On clicking logout the session should be invalidated and login page must be displayed  Capture the details like Name, Username, Password, Address, State, Country, Email Address, PAN, Contact No, DOB, Account Type etc |
| US\_02 | Apply Loan Details | As a Customer, I should be able to login so that I can apply and view the loan details  Acceptance criteria:  Customer should be able to apply loan in the system and it should be saved in the database.  Capture fields like Loan Type, Loan Amount, Date, Rate of Interest and Duration of Loan. |
| US\_03 | Update Account Details | As a Customer, I should be able to login so that I can update my account details in the system  Acceptance criteria:  When the Customer logs in, he should be able to update his account details. |

* **Expected Deliverables**

The following deliverables are expected as outcomes

* Application Code base
* API for each service
* Readme document on the complete application
* Setup of the application (Since there is no UI skills, either UI can be kept minimum or you can use Post man to test your APIs)
* High level steps used to convert to server less architecture
* How to run the application
* Any inference
* Snapshot of any implementation
* Reports:
* Functional Test Report
* Code Profiling Report
* Other Reports as applicable
* **Milestone and duration**

As per project requirement modification can be done in the below table.

|  |  |
| --- | --- |
| Milestone | Topic |
| Milestone -1 | Developing the API using Java microservices |

* **Implementation Notes**

As per the project requirement modification can be done in the below table.

|  |  |
| --- | --- |
| Milestone -1 | * Use Rest APIs to develop the services * Use Microservice Architecture * Use Domain Driven Design * Use Maven Build * Implement repository pattern. * Implement 12 Factor principles * Use browser / POST Man to invoke APIs. * Use ORM to work with database, under repository pattern * User access security microservice to allow/disallow CRUD operations * Use browser / POST Man to invoke APIs * For messaging use either Kafka/ Active MQ/ Rabbit MQ * Use Kafka for messaging * Implement API Versioning * Implement API Gateway * Use Swagger API * Message input/output format should be in JSON (Read the values from the property/input files, wherever applicable). Input/output format can be designed as per the discretion of the participant * Any error message or exception should be logged to the user should be user-readable (and not technical) * Database connections and web service URLs should be configurable. * Web service URLs should be configurable. * Implement External Configurable Solution * Unit testing should be implemented using Junit /Mockito * Use GIT for source code repository * Database can be any of the mentioned - SQL/ Oracle/ MySQL/ Mongo DB/ Cassandra |

* **Evaluation rubrics**

|  |  |
| --- | --- |
| Core Java 8 | * Associate should have used appropriate Base class Libraries, Control Statements and Operators, File Handling and I/O Operations for implementing the functionalities. * Use Core Java 8 features like Lambdas, Streams |
| Microservices | * Follow the below basic structure * API - Controllers * Domain - Model, Events, Business Services Integration * Services – API Implementation * Infrastructure Project * Associate must have designed/developed individual Microservices for each functionality. * Each of the Microservices need to comprise below functionality, which need to be developed   a. REST Controllers  b. Services  c. Entity & Model classes, including appropriate relationship (like One-One, Many-One, etc…) between Entity Classes. (Entity and Model classes have been developed in the Previous Phase)  d. In case specific Entity or Model classes are required across multiple Microservices, it is recommended to maintain separate copy of Entity or Model classes for each Microservices.  e. Microservices should interact with corresponding DB tables or Databases it owns.  f. Microservice need to interact with other Microservice  h. Usage of Postman to test the Microservices by directly passing requests to each REST end Point, of each Microservice  i. Unit Testing code should be developed using NUnit/Xunit and perform Unit Testing  j. Circuit Breaker, Service Registry, Service Discovery should be implemented   * Use Swagger UI and test each public method in the service * Implementation of Repository pattern |
| Rest API | * Associate must have used REST API for exposing resources * Associate must have used HTTP GET/PUT/POST request method designators for the business methods which is to be exposed * Associate must have customized the request and response formats according to the requirement * Associates must have used appropriate RETURN CODES based on the service outcome * Associates must have extracted query/form/header parameters from the input * Associate must have built a custom response based on the input |
| Spring Boot | * Develop RESTful web services with Spring Boot * Make BEST USE of Spring Boot Actuator and Spring Boot Developer Tools * Externalize application configuration using Spring Boot Profiles and Dynamic Configuration |
| Kafka | * Create your Producers and Consumers in Java to interact with your messaging Queue * Include Brokers * Build Kafka based messaging applications with spring kafka |
| MySQL | * Use MySQL to retrieve, filter, analyze, format and present information for your APIs |
| Unit Testing | * Test cases covers the functionality of API with custom inputs * Good test Coverage using Junit/Mockito |
| Common | * Code Smell * Technical Debt * Secured Coding * Coding Standards |