Bank Server Application

To be developed individually

Bank server application with simple functionalities like getting the balance of the account, doing routine transactions as withdraw, deposit, fund transfer.

# Overview

Application should allow users to check the balance in the account; withdraw/deposit money; transfer funds from one account to another; display last 10 transactions on the account.

All accounts to be created while the application boots. Application must have at least 10 accounts. Account is owned by a customer.

* Milestone #1: Class design
* Milestone #2: Write test cases
* Milestone #3: Display balance of a given account.
* Milestone #4: Withdraw/deposit money in the account.
* Milestone #5: Transfer money from one account to another.
* Milestone #6: Display last 10 transactions of an account.

## Points to consider

* #1: Appropriate error messages if user tries to do an operation on invalid account.
* #2: Account balance cannot be negative.
* #3: One customer can have only one account, there is only one type of account.

# Expected LEarning

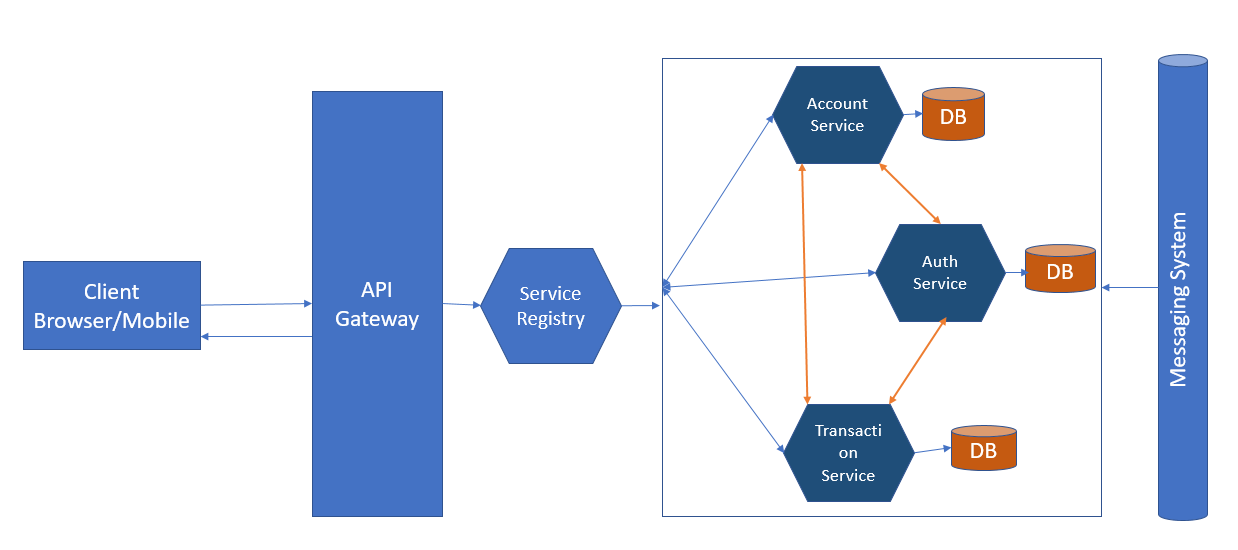
Participants will learn following concepts from this assignment

* Basic programing building blocks like classes, interfaces, flow control statements, arrays, exception handling, programming to interface, one-to-one and one-to-many relationship.
* Writing test cases using Junit
* Creating Github repository and committing code to the repository, branching.
* Sonar for code quality

## Project Phases

|  |  |
| --- | --- |
| Phases | Description |
| Phase #1 | Develop application using arrays |
| Phase #2 | Change array to appropriate collection |
| Phase #3 | Implement JDBC |
| Phase #4 | Implement JPA |
| **Phase #5** | **Implement Spring Boot Data JPA– Microservices** |
| Step 1 | Account Service + Transaction Service + Auth Service |
| Step 2 | Service Registry + API gateway Impl+ SAGA/CQRS Distributed Transaction |
| Step 3 | Containerize the app, Upload Image to docker Hub |
| Step 4 | Image will be deployed to EC2 |

# IMPLEMENT SPRING DATA JPA WITH Spring BOOT - MICROSERVICES

****

**Step1:**

1. All Rest API request and response format must be in JSON.
2. Use in-memory (like H2) database to design all microservices.
3. 200, 201, 400, 404, 401, 500, 505 – use appropriate http status code in your response object.
4. Handle the Rest exceptions globally using Rest exception handler for all endpoints.
5. Test your Rest Api using Junit and Mockito.

**REST API Endpoints:**

|  |  |  |
| --- | --- | --- |
| Endpoints | HTTP Verb/Method | Description |
| /api/v1/accounts | POST | Add new account object in DB |
| /api/v1/accounts/{account\_No} | PUT | Update account by account Number. |
| /api/v1/accounts/{account\_No} | DELETE | Delete account by account Number |
| /api/v1/accounts/{account\_No} | GET | Find Account by account Number. |
| /api/v1/accounts/{account\_No}/balance | GET | Check balance of the Account-by-account Number. |
| **/api/v1/auth/isValidCustomer** | **POST** | **Authenticate customer details (username & password) from DB.** |
| **/api/v1/auth/isActive/{customer\_Id}** | **GET** | **Check the status of Customer** |
| /api/v1/transactions | POST | Deposit/withdraw/ Fund transfer to any other account. Specify transaction type, account No and amount in request body. |
| /api/v1/transactions/summary/{from\_date}/{to\_date}/  {account\_no} | GET | Get the summary of transaction details for a particular account. |

**Step 2:**

1. Use Server-side discovery (Service Registry) to register all services, this would restrict in exposing the service URL to client.
2. All requests must be filtered by API gateway. Configure the appropriate router for each microservices.
3. Use SAGA pattern to handle the distributed transactions via any messaging system.

**Step 3:**

Each microservice must be containerized and then upload all images to docker hub repository.

* 1. Create docker hub <https://hub.docker.com/> login.
  2. Upload docker images into your repository.
  3. Ensure that your docker images accessible anytime to deploy into appropriate environment.

**Step 4:**

1. Create AWS free tier/ AWS sandbox.
2. Create one EC2 instance under any one of the AWS regions.
3. Install docker in EC2 instance.
4. Deploy all docker images into EC2 instance.
5. Use Public IP of EC2, to access microservice endpoints.
6. Validate your results on browser.

|  |  |
| --- | --- |
|  |  |

## Timeline for Execution

Key project dates can be outlined below. Phase duration is best guess estimate and may vary based on performance and caliber of the participants.

|  |  |  |  |
| --- | --- | --- | --- |
| Description | Start Date | End Date | Duration |
| Phase 1 |  |  | 3 days |
| Phase 2 |  |  | 1 day |
| Phase 3 |  |  | 1 day |
| Phase 4 |  |  | 1 day |
| Phase 5 – Step 1 |  |  | 1 day |
| Phase 5 – Step 2 |  |  | 1 day |
| Phase 5 – Step 3 |  |  | 1 day |
| Phase 5 – Step 4 |  |  | 1 day |