**Business Requirement**

The project is mainly based on the idea of developing a banking solution on micro services API based solution, which would replace the existing manual banking solutions. All the operations that are carried out in the bank manually (like new account registration, deposit, withdrawal, cheque book issue etc.) would be performed automatically and easily by the Core Banking Solution.

**Use cases to be implemented:**

1. Customer Registration
2. Update Customer
3. Retrieve customer information which should have all linked account details
4. Account Creation
5. Update Account
6. Close Account
7. Withdrawal from an account
8. Deposit to an account
9. Fund transfer between two accounts
10. Validate From account exist and transfer amount is < current balance.
11. Validate To account exist
12. Debit “FROM” Customer account and Credit “TO” customer account by transit amount
13. Generate e-statement for a particular time period for example : 01/07/2018 to 01/08/2018
14. Each page should have maximum 10 records and if total records in a statement exceeds 10 than implement pagination at API level.
15. Assume after the first release of the application, there is a change in Customer registration process for some of the countries (not all the countries), figure out the best approach to deal with this scenario and implement it.

**Expectations:**

1. Design logical architecture diagram and activity diagram for the solution.
2. The solution should be built based on microservices architecture, figure out the microservices to be built based on the domain model.
3. Implement Rest APIs for all the use cases.
4. Each API should have proper swagger documentation with swagger UI implementation, which can be used to **test the API**.
5. The API specification should be as per Best practices.
6. Implement input field validations and return proper error message for invalid input.
7. The API should return proper HTTP error code based on different application specific exception/error scenario, along with informative error message.
8. Junit test cases with 70% code coverage. (Mockito can be used for mocking)
9. Integrate with local sonarqube so that the code coverage can be viewed. (sonarqube eclipse plugin can be used)
10. Identify and implement the best way of communication between two microservices, keeping in mind performance, failover and resilency.
11. Handle failover scenario for example: when app-A is calling app-B and app-B fails.
12. Enable logging in each microservice.
13. Tech stack should be used to develop the solution : **Java**, **Spring boot**, **Netflix Zuul**, **Netflix Eureka**, **Spring data with JPA and Rabbit MQ/Active MQ**
14. Address pagination and selective attributes (facilitate browser and mobile view).
15. Add caching as well to improve performance.
16. Prepare a document which can be followed to run and test the application. For testing the APIs swagger UI should be exposed.
17. Create a folder with your name under the box location: <https://lion.app.box.com/folder/52025587228> and upload the source code with the document.

Note : For persistent data , in-memory database like H2 can be used.