Customer Account Tracker

The objective of this sample case study is to recap your J2EE skills.

A Leading private bank looking for solution to track customers and their account details. As part of requirements, the below mentioned specification has been given to the partner to implement.

Requirements Specification:

1. Able to create new account for a customer (only one account type / customer)
    a. Account type may be savings (individual/joint) & current etc.,
2. Able to edit customer personal details
3. Able to fetch one or more customer personal details including account details too
4. Customers can transfer funds from one account to another account. If enough fund exists.
5. Refer Appendix for more details

Expected Deliverables

1. Maven project - solution code base, unit test scripts, pom.xml & properties file(s)
2. Read Me file (read.txt) – Explaining end points
3. Test cases execution log report – Unit Test for End Points & Services (Maven test report)

Note: Find below recommended naming conventions to consider

1. groupId: <ADID>.<phase>.<project>
   Example: avitepa.foundation.bank
2. artifactId: <ADID>_<casestudy>
   Example: AVITEPA_bank

Steps to follow / Check point for self-review (indicative only)

A. Set up Dev environment

Note: You may refer the WASP portal **https://wasp.wipro.com/esd** to get the required software/tools

- **During Phase 2 & 3**
    1. Git Version 2.9
    2. OpenJDK Version 11
    3. Maven Version 3.6.0
    4. Spring Tool Suite (OR You may use any alternative IDE)
    5. You may use H2 DB (or MySQL Workbench Version 8.0.CE)
- **During Phase 4**
    1. Jenkins Version 2.84
    2. Docker

B. **Getting started with Creating Spring Boot Application**
1. Configure pom.xml with all required dependencies
2. Configure application.properties (server port, DB details and any other)
3. Configure application-integrationtest.properties (for unit testing)

C. **Build your solution with suitable design / sequence of steps with your plan /assumptions**
1. Identify Model(s) and configure attributes with JPA

2. Create Repository interface and test sample CRUD operations for identified Model(s)
    i. Test for Empty records
    ii. Test for saving
    iii. Test for findAll
    iv. Test for findById
    v. Test for findBy<AnyField>
    vi. Test for deleteById
    vii. Test for deleteAll
    viii. Test for update <using serialized field>
    ix. Test for update <using non-serialized field>

    Note: if required append/define customized method with Query
3. Create "@RestController" and test for all identified end points
    i. Create methods for all identified end points
    ii. Test all end points with hard coded Response body
        a) Test for GetMapping
            a. for String
            b. Object
            c. List
            d. ResponseEntity<HttpStatus>
        b) Test for PostMapping
        c) Test for PutMapping
        d) Test for DeleteMapping
4. Create "@Service" and test for all identified business requirements
    i. Create interface and declare all required methods
    ii. Implement a class with business logic
    iii. Test for identified services
5. Integrate "@RestController", "@Service" and "@Repository"
    i. Replace hardcoded data in "@RestController" with service(s) execution
    ii. Re-Test your end point execution
    iii. Run the application
        a) Test with Postman
        b) Rest client (optional)
        c) UI... (optional)
    iv. Build package with maven
        a) Check/review your test log
        b) Run jar file and validate completeness & correctness of solution
6. Share your code base with below options
    i. If you were able to connect wipro network
        a) Login to https://topgear-training-gitlab.wipto
        b) Create new project
        c) Push your code
        d) Give permission to "AVITEPA"
    ii. Share code base using OneDrive

Appendix

- **Services Implementation**

Class	Method and Variables	Description
AccountServiceImpl		
	`public String addAccount(Account acc);`	This method is expected to receive Account object and creates account and customer details by invoking appropriate DAO
	`public List<Account> getAllAccounts();`	This method is expected to return all Accounts including customer profile
	`public List<Customer> getAllCustomers();`	This method is expected to return all Customers details including account details
	`public String transferFunds(int from,int to,double amount)`	This method is expected to return transfer status like "ID MISATCH" or "INSUFFICIENT FUNDS" or "SUCCESS" **only**. It iterates through "accounts" to find existence id's for both payer and beneficiary, if both found and if payer has sufficient funds then updates the balance for both accounts suitably.
	`public Account getBalanceOf(int accountNumber);`	This method is expected to return account details by mapping account number. It iterates through "accounts" to find existence of received account number, if account number found it will return account object (details). Otherwise "null" to be returned.