**Finished Implementation:**

* System Design, UML Class diagrams.
* GitHub/Version Control setup, test GitHub repo, create branches.
* Setup DB connection, test CRUD operations on PostgreSQL.
* Tables in DB and assigning relationships to the tables.
* Relationships between classes (composition, association, aggregation).
* Backend code setup – Model(class), Services(methods) for models, Controller(endpoints/routes).
* Frontend code setup – React.js Components/Views (JSX + Bootstrap).
* Setup React application – SPA routes and views, to render the views + data.
* Connect backend + frontend for CRUD operations and persisting data to DB.
* Testing local application and making it ready for production, push production code to GitHub.

**Unfinished Implementation:**

* Laying out inheritance hierarchy among OOP classes to database relations.
* Deployment of backend server and frontend application to a deployment service like Heroku.

**Bugs fixed:**

* Negative balance being allowed when transferring or withdrawing the balance.
* Retyped password input field wouldn’t match when user typed the same password.
* Hooking up front & backend – for example, while making login & sign-up requests, from frontend to backend API the HTTP methods (for instance, using GET method on a POST method route) used were incorrect.
* CORS issue – CORS denial error due to unset headers at the time of making requests to the backend server.
* Security problems – Sensitive information leak fixed using JWT where the response sent by the backend service possesses an access token which can only be used for a specific amount of time while user is making requests. User needs to get a renewed token when an access token gets expired.
* Transactions not being recorded – Foreign keys not being properly created, which caused the table to not contain any data whenever a transaction occurred.
* Relationships between OOP classes & DB tables – Data not being persisted into a table for any of the classes whenever a database operation happens.

**Unfixed bugs:**

* User navigating manually to a route, logs them out of the session and is redirected to the login page.
* User can put any amount of value while depositing money.
* User can put any date as their DOB while signing up for their account.

**Java Technologies Used:**

* + Java J2EE SDK 17.0.1
    - Java SDK for enterprise grade APIs.
  + PostgreSQL 14.2
    - Relational database for storing large amounts of information.
  + Spring Boot 2.6.6

Java Framework for automated application packaging.

* + - JPA Package
      * ORM (object relational mapper) module for storing POJO’s as database tables.
    - Web Package
      * Enables internet features and allows server to run on socket.
    - Security Package
      * Authentication, Authorization, and Permission handler package.
    - Mail Package
      * Mail sender package to allow email communication from server.
  + 0Auth Java JWT 3.19.1
    - JWT token provider and encryptor, handles the token lifecycle.
  + Project Lombok 1.18.20
    - Annotations for code shortening and reducing boiler plate code overall.
  + MailDev 2.0.1
    - Mail server to receive registration emails from the spring boot server in development environment

**JavaScript Technologies Used:**

* + Node.js 16.14.2
    - Environment for running JavaScript.
  + ReactJS 17.0.2

Reactive JavaScript Framework for modern UI.

* + - React DOM Router
      * Router module to allow secure shared state across the application.
    - React Bootstrap
      * Prebuilt React components from the Bootstrap library.
  + ViteJS 2.8.0
    - JavaScript compiler and application builder.
  + Bootstrap CSS 5.1.3
    - CSS styling library with prebuilt components.
  + SASS CSS compiler 7.0.1
    - CSS extension compiler for reducing redundancy and used styles.
  + Axios request handler 0.26.1
    - HTTP request handler to help making code shorter for requests.
  + Yup JS validation 0.32.11
    - Validation module to perform frontend checks on data before it is being sent.
  + Formik 2.2.9
    - Form handler module to help with data handling and validation with reduced boiler plate code.

**My Contributions:**

* Worked on creating the Bank Account class and all its related components like BankAccontController that takes care of the routes for backend API when user communicates with the service for any transactions after they are logged in; BankAccountService that defines methods which make calls to the JPARepository from where we can perform CRUD operations on the relations; Unit tests to test the BankAccount related operations like creating a bank account, finding all bank accounts and their balances for a specific user, finding balance for a specific bank account, deleting a bank account, updating balance on an account where all these operations persisted the data into the database to make sure that data is properly being persisted into a database.
* Besides, I worked on frontend React components, namely, Customer, Not Found, Deposit, Withdraw, Transfer and Balance. Handled making API calls to the backend server, managed the response returned by the API, updated the state of the respective component accordingly based on the returned response from the API, updated the views accordingly and structured the frontend properly to add styling to all the frontend pages.
* Also, I worked on pushing my part of the application on a central repository hosted on GitHub from where everyone can pull the latest code whenever I push my code to the repository so the team can get updated code every time I make changes to my code.

**Teammate Contributions:**

**Omar** – CORS Management, Authentication/Authorization/Login/Registration, UserAccount class and all its related components, Frontend – Login Component.

**Philippe** – Application/System design, Class relationships, Database management, Transaction class and all its related components.

**Peer review**:

Soham Thaker: 10/10

Philippe Cormier: 10/10

Omar Hussein: 10/10