AVOCADO TOAST: AN ONLINE BANKING PLATFORM

Spring-2019

DANH PHAM ANANDITA DUBEY FLAVIU TAMAS

CARLOS DELEON ALEX PETROS

CONTENTS

1. Planning and Scheduling

Assignee	Email	Task	Duration	Depends on	Due date	Evaluation
Anandita	Adubey2	Design Pattern	2h	Class Diagram, Architecture Modeling	04/13/19	100% completed the task
Alex	Apetros1	Unit Testing	2h	System Implementation	04/13/19	100% completed the task
Carlos	Cdeleon1	Refine Implementation	2h	Previous Implementation	04/13/19	100% completed the task
Danh	Dpham16	Unit Testing Documentation	2h	Unit Testing, System Implementation	04/13/19	100% completed the task
Flaviu	Ftamas1	Refine Implementation	2h	Previous implementation	04/13/19	100% completed the task

2. Problem Statement

2.1 What is your product, on a high level?

- Our product is an online banking application, which will allow the customers of a bank or other another financial institution to perform money-management without physically visiting the bank.
- It will allow the customers to open their accounts, manage them electronically, to monitor them, to make transactions, pay their bills, transfer money, make deposits, and so on.

2.2 Whom is it for?

- An online banking application is beneficial to everyone but is especially useful for those people with a stringent work schedule.
- It will help them to manage their accounts and keep track of their activities in a quick manner with minimal costs without the need to visit a physical bank during working hours or make a phone call.

2.3 What problem does it solve?

- 24/7 availability, saving the customer from rushing the banks during working hours. With an online banking system, people can perform their tasks at a time that suits their work schedule.

- Stringent schedules, where customers with strict working hours to perform their banking activities effectively and conveniently.

- Centralized source of information, where instead of visiting different officials specialized in different tasks, the customer can use an online banking application flexible single enough to do any task in a click. Human bankers are not always available, but the application always will be, meaning there is no need to rush to the bank to get things done.

- Remain informed: With the online banking system, customers can easily receive up-to-date information regarding their upcoming deadlines or dues through notifications, emails, or text messages.

- Easy bill payments, so that there is no need to rush to the bank. Everything can be done at home instead.

2.4 What alternatives are available?

- Services provided by 3rd party application, like

  Mint A phone app

- In person banking services
- Phone calls
- ATMs

2.5 Why is this project compelling and worth developing?

- It would reduce costs for banks by reducing the amount of human labor re-quired.
- An electronic banking system would enable banks to keep stringent records
- A computerized ledger would reduce the number of mistakes when calculating interests, making transactions, and so on.
- It would increase customer satisfaction because they would have access to our banks from any place with Wi-Fi.
- Providing flexibility to the customers to get their work done at minimal or no cost.
- Saving time of the customers (customer need not necessarily visit the bank physically)

2.6 Describe the top-level objectives, differentiators, target customers, and scope of your product.

- Objective: To create a product that performs to our requirements and have it sustainable for multiple lifecycles.

- Differentiators: Our system won't sell data to any third parties, have no hidden transaction fees, and will not participate in predatory lending practices

- Target customers: Our target customer is the average citizen, anyone that currently uses or will use a bank for their transaction needs. (Basically useful to everyone, no matter what)

- Scope: The scope of our project is building an online business ledger, making a database to store our information, developing a web app and making a UI for our customers.

2.7 What are the competitors and what is novel in your approach?

- Our competitors are Finacle, nCino, Oracle, etc. What we bring new to the table is that our system will be entirely online.
- Our platform is customer focused and will be fully transparent. We won't sell customer data or charge hidden fees. Additionally, providing the customers with the flexibility to perform the tasks at the minimal or no cost.

2.8 Make it clear that the system can be built, making good use of the available resources and technology.

- Our system is possible because it will be very simplistic and direct. Clients will connect to our application software where they will be greeted by a user-friendly interface, which will primarily be coded in HTML, with CSS used to create a beautiful design for our clients.
- Finally, the frontend of our application will be connected to a backend SQL database. The database will be responsible for recording transactions such as withdrawals, deposits, bill pays, and more.

2.9 What is interesting about this project from a technical point of view?

The project will use a client-server architecture in order to be accessible to clients from anywhere. Emphasis will also be placed on graphical design, using our creativity to design a nice interface for our clients. Providing flexibility to the clients to perform the non-transactional tasks through online banking. That might include:

- Viewing account balances
- Viewing recent transactions
- Downloading bank statements
- Downloading periodic account statements
- Funds transfer between the customer's linked accounts Paying third parties, including bill payments
- Credit card applications
- Register utility billers

3. Requirements

3.1 User Requirements

Use Case:

3.1.1 *Use Case Number: 1 Use Case Name: Register Customer*

- Actors: Banker
- Description: Add a new customer, with their own username and password
- Alternate Path: None
- Pre-Condition: Logged in

3.1.2 *Use Case Number: 2 Use Case Name: Register Customer*

- Actors: Banker
- Description: Take a certain amount of cash from the customer, adding the amount to their balance
- Alternate Path: None
- Pre-Condition: Logged in

3.1.3 *Use Case Number: 3 Use Case Name: Register Customer*

- Actors: Banker
- Description: Give the customer a certain amount of cash, subtracting the amount from their balance
- Alternate Path: None
- Pre-Condition: Logged in, account has sufficient balance

3.1.4 *Use Case Number: 4 Use Case Name: Register Customer*

- Actors: Banker, Customer
- Description: View a list of all the transactions in the account, as well as the overall account balance
- Alternate Path: None
- Pre-Condition: Logged in

3.1.5 *Use Case Number: 5 Use Case Name: Register Customer*

- Actors: Customer
- Description: Send someone else money electronically through the website

- Alternate Path: None
- Pre-Condition: Logged in, account has sufficient balance

3.1.6 *Use Case Number: 6 Use Case Name: Register Customer*

- Actors: Analyst
- Description: View the amount of money that was deposited and withdrawn in a single day, for the entire bank
- Alternate Path: Manual SQL access
- Pre-Condition: Logged in

3.1.7 *Use Case Number: 7 Use Case Name: Register Customer*

- Actors: Banker, Customer, Analyst
- Description: Authenticate themselves to the system
- Alternate Path: None
- Pre-Condition: None

Use Case Requirements:

3.1.8 *Use Case Number: 1*

- Introduction: Register Customer, this system interacts with the user, authentication utility, SQL database
- Inputs: Username Password
- Requirements Description: Registers a new customer, they must input a user-name and password for their account
- Outputs: Their login information is logged into the system to be recalled later
- Alternate Path: None

3.1.9 *Use Case Number: 2*

- Introduction: Deposit Cash, this system interacts with the user SQL database
- Inputs: Cash
- Requirements Description: Insert cash from the customer, the amount is added into the balance
- Outputs: Cash amount is added into their balance
- Alternate Path: None

3.1.10 *Use Case Number: 3*

- Introduction: Withdraw cash, this system interacts with the user SQL database
- Inputs: Cash amount
- Requirements Description: Withdraw cash from the bank, the amount is de-ducted from the balance
- Outputs: Cash is given to the customer and cash amount is deducted from balance
- Alternate Path: None

3.1.11 *Use Case Number: 4*

- Introduction: View Ledger, this system interacts with the user SQL database
- Inputs: User PIN number
- Requirements Description: View a list of all the transactions in the account, as well as the overall account balance
- Outputs: Transaction record is displayed on the screen, with their account balance
- Alternate Path: None

3.1.12 *Use Case Number: 5*

- Introduction: Pay Bill, this system interacts with the user, SQL database, and the recipient
- Inputs: Cash amount and information of the recipient
- Requirements Description: Send a person or company money electronically through the website
- Outputs: Deducts cash amount from overall balance and cash is sent to recipient
- Alternate Path: None

3.1.13 *Use Case Number: 6*

- Introduction: View Transaction Volume, this system interacts with the user SQL database
- Inputs: User PIN number

- Requirements Description: Views the amount of money deposited and with-drawn in a single day
- Outputs: Shows deposit withdraw amount in specified day
- Alternate Path: Manual SQL Access

3.1.14 *Use Case Number: 7*

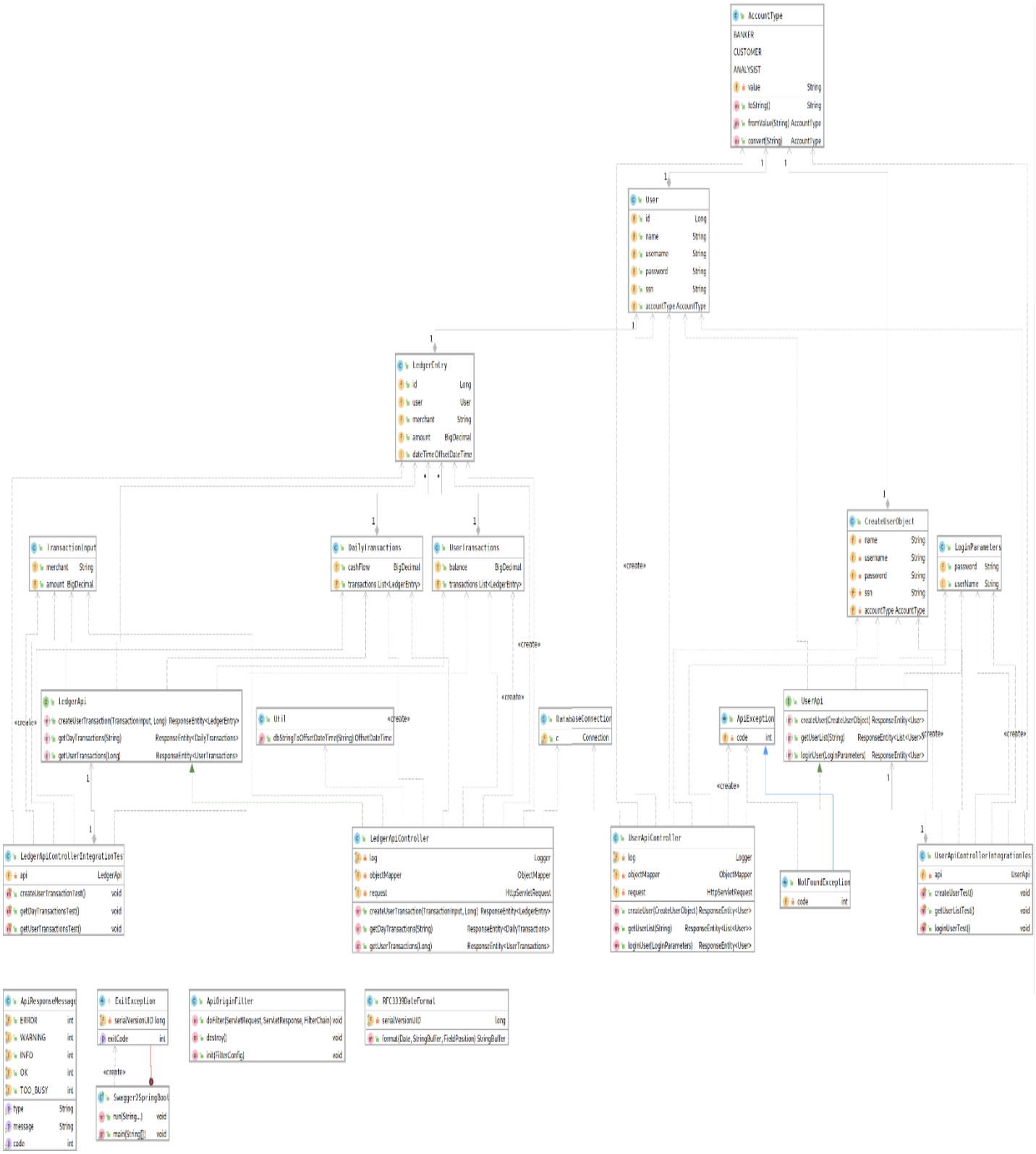
- Introduction: Log in, this system interacts with the user and authentication utility
- Inputs: User ID and password
- Requirements Description: Authenticate a user to their account
- Outputs: User is logged into the system and given access to use their account
- Alternate Path: None

3.2 System Requirements

- CPU: Intel/AMD processor
- RAM: 2 GB(Recommended)
- OS: 64-bit Windows 7, Windows 8.1, Windows 10

4. System Modeling

4.1 Class Diagram



4.2 Database Specification and Analysis

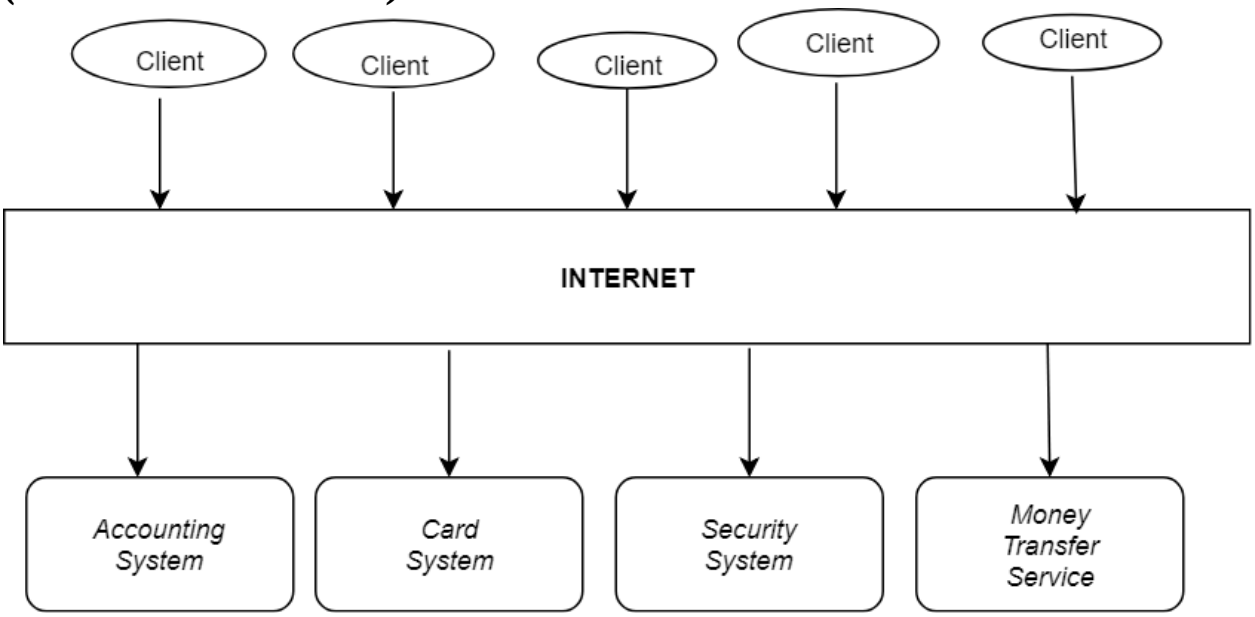
4.2.1 Specify your system database tables (data attributes and their types) and relationship between them (Primary Keys and Foreign Keys)

- The first table would have 3 data types which include INT, DATE, and VARCHAR.
- INT Attributes: social security number, account number, account balance
- DATE Attributes: date of birth
- VARCHAR Attributes: first name, last name, email address, username, password, account type
- The primary key would be the account number.
- The second table would have 2 data types which include INT and DATE.
- INT Attributes: transaction number, transaction amount, and account number
- DATE Attributes: transaction date
- The primary key is the transaction number.

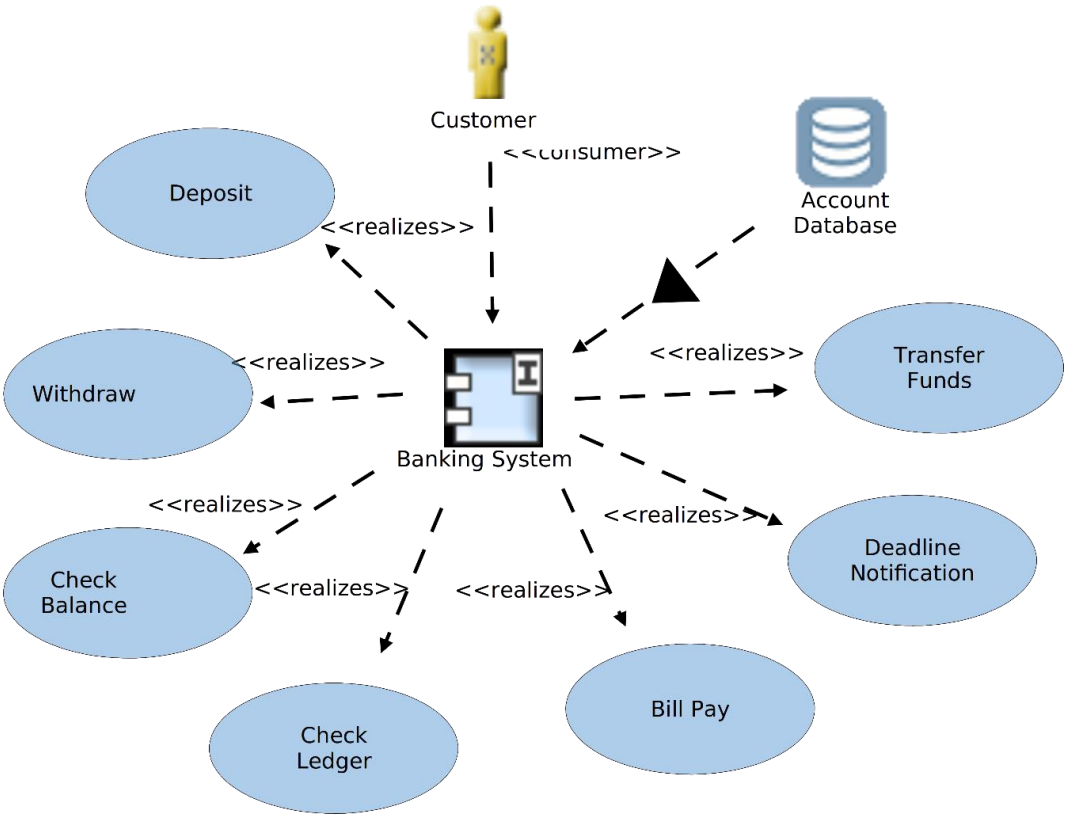
4.2.2 Specify the type of database management system (MySQL, MS-SQL server, Oracle, etc.) you will use in your project

- The database management system that will be used in the project will be MySQL.

4.3 Architecture Modeling (Client Server Pattern)

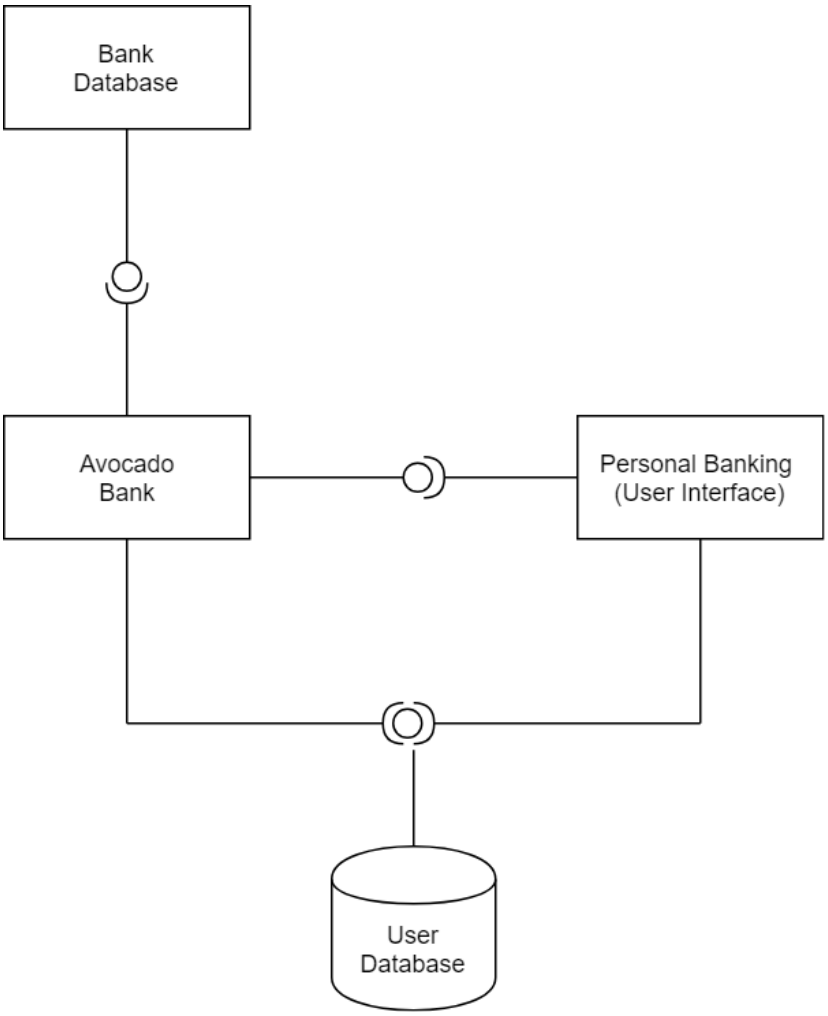


4.3.1 Context Model

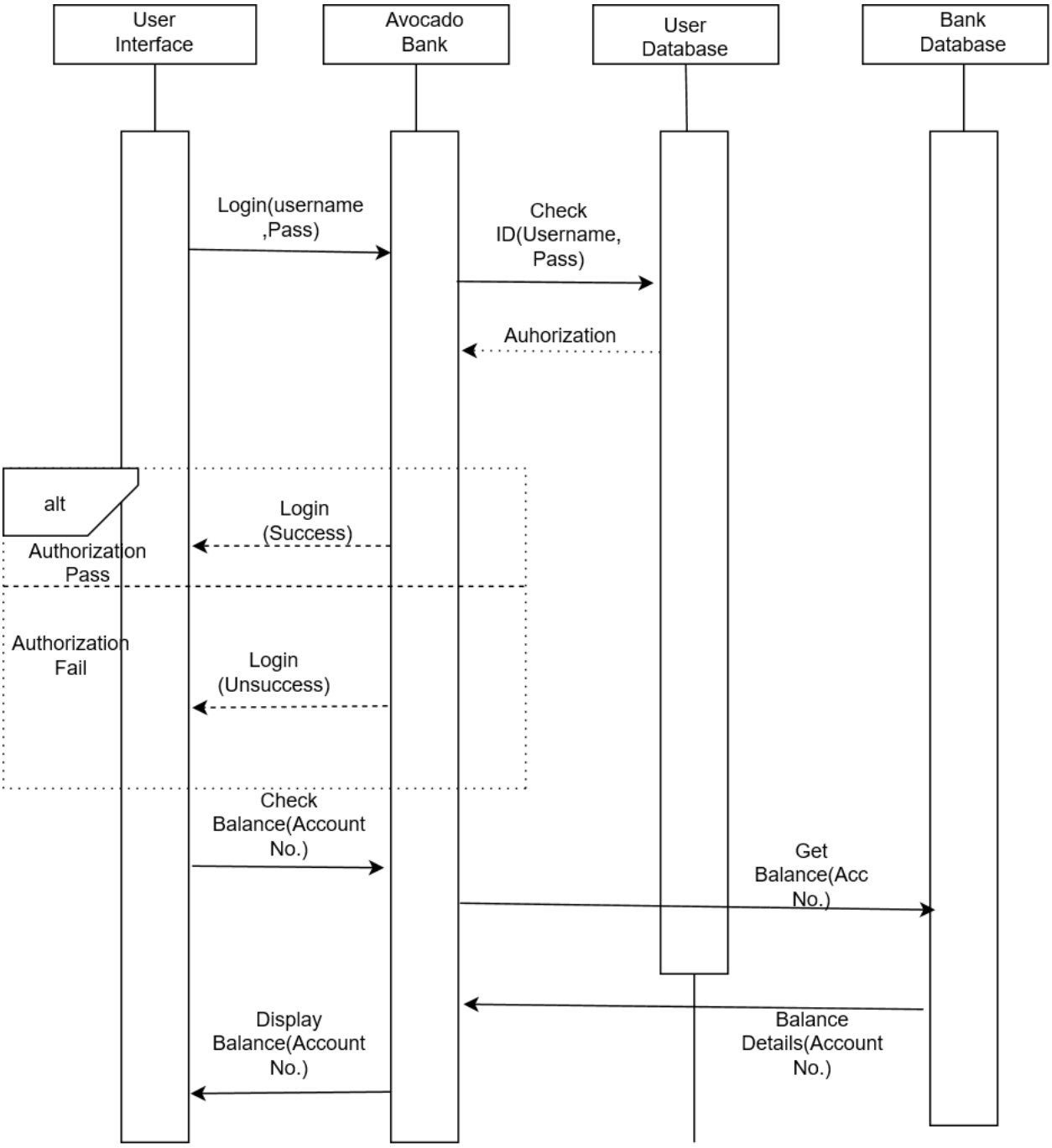


4.3.2 4 Views

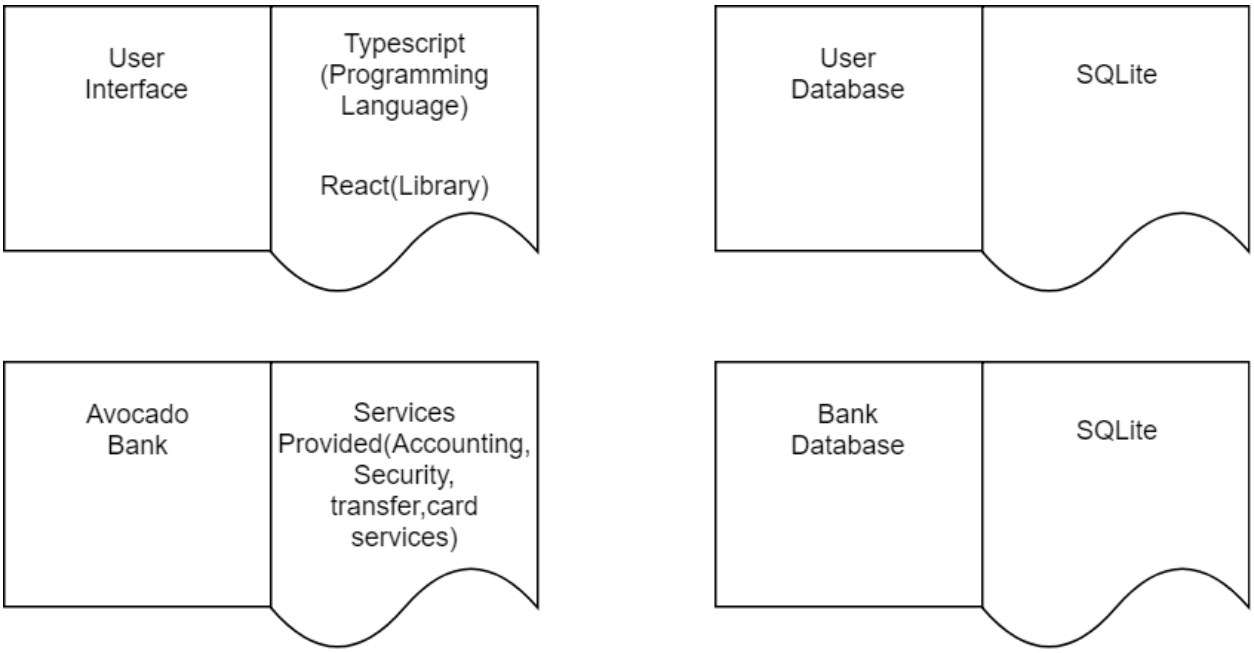
Logical View:



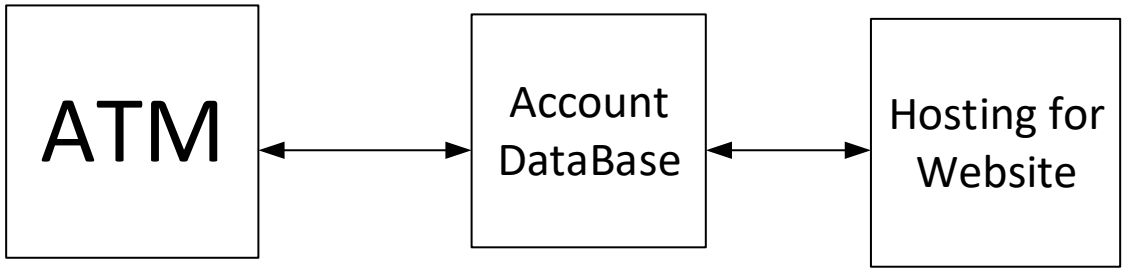
Process View:



Development View:



Physical view:



4.3.3 System Design Pattern

Measurement of Class Diagram's Coupling and Cohesion:

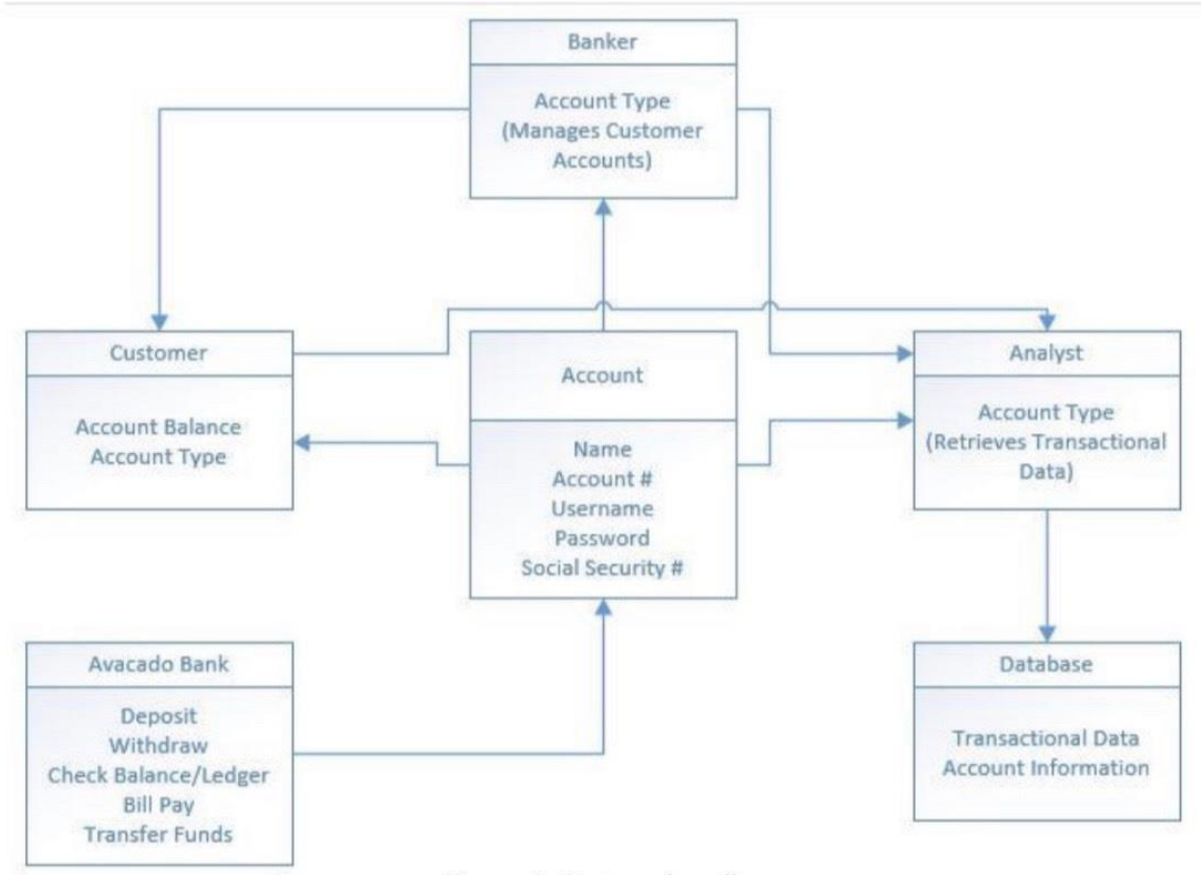


Figure 3: System class diagram

Calculations based on above diagram:

Formula- **Cohesion for a class(n) = Classes on which n directly-indirectly potentially depends/Nc**

Where Nc-total no. of classes

- For Banker class:

Cohesion = 3/6=0.5

- For Customer class:

Cohesion= 2/6= 0.34

- For Avocado Bank:

Cohesion = 5/6=0.83

- For Analyst class:

Cohesion = 1/6=0.16

- For Database class:

Cohesion = 0/6=0

- For Account class:

Cohesion = 4/6=0.67

Classes	Directly or indirectly depends on:	Cohesion Values:
Banker	Customer, Analyst, Database	0.5
Customer	Analyst, Database	0.34
Avocado Bank	Account, customer, banker, analyst, Database	0.83
Analyst	Database	0.16
Database	-	0
Account	Customer, Analyst, Database, Banker	0.67

COHESION VALUES

Coupling :

A coupling measure between classes, which class directly coupled with other class

Flow P (C2, C1) = Slice (P, c1, V c1) | N (c2) / N (C1)

Flow *P* (*c*2, *c*1) implies that information flow from class *c*2 to class *c*1 in a modular *P*, employee information modular.

Coupling P (C1, C2) = Flow p (C1, C2) +Flow p (C2, C1)/N(C1) N(C2)

Note: The instructor did not want us to calculate coupling values

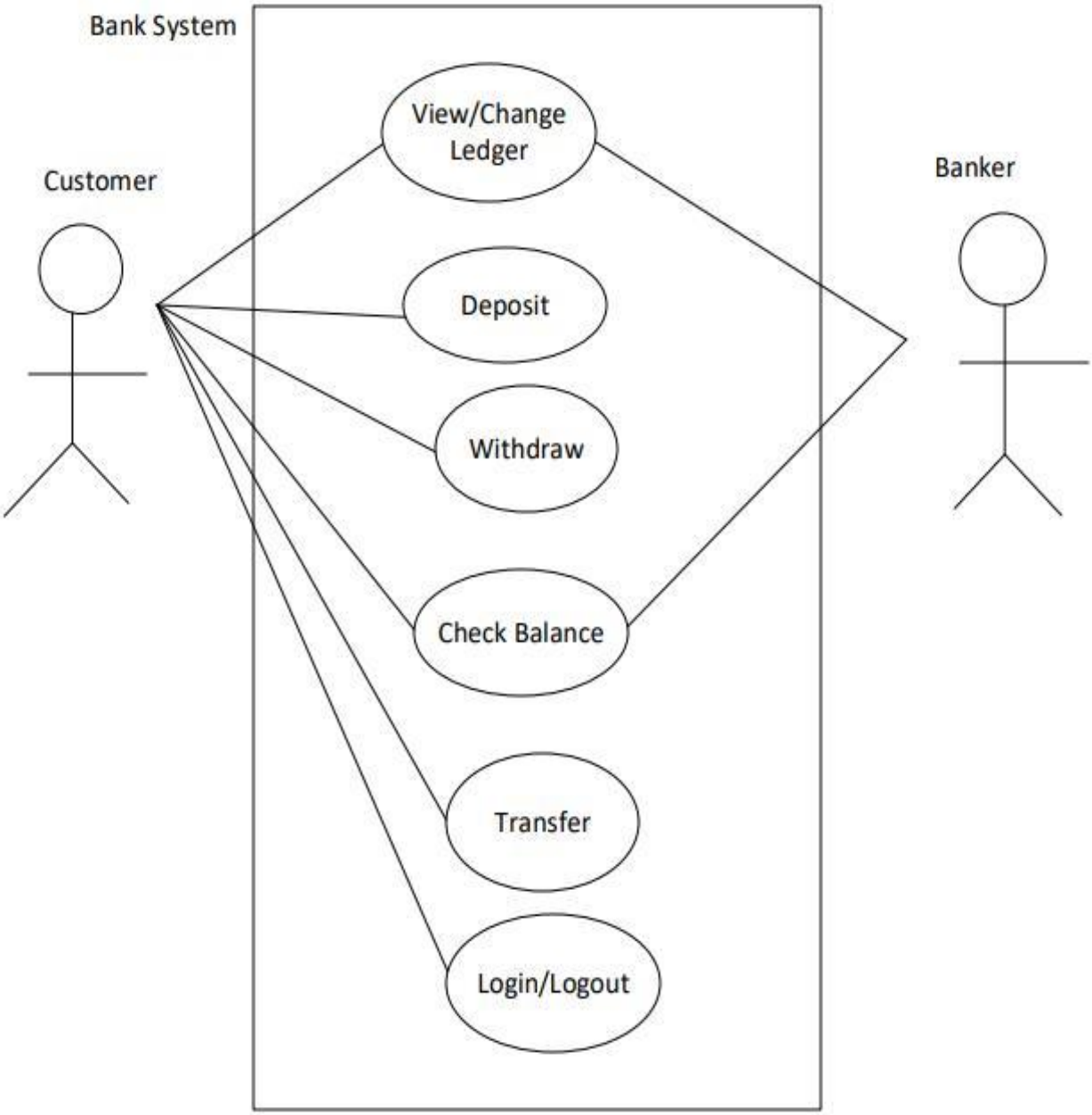
System Design Pattern:

Major Issues During Implementation	Design Patterns used to rectify the issues	Reasons for the Design patterns chosen
Database Connectivity	Singleton Pattern (Creational Design Pattern)	It is used for the database connection so that it can be easily accessed from anywhere in the program.
Implementation of HTTP API	Strategy Pattern (Behavioral Design Pattern)	Since it lets the algorithm vary independently from clients that use it.
To provide controller with Object Mapper (to translate between JSON and Java objects	Dependency Injection pattern	It is used since we did not create the Object Mapper but instead ask the provider to create one for us.

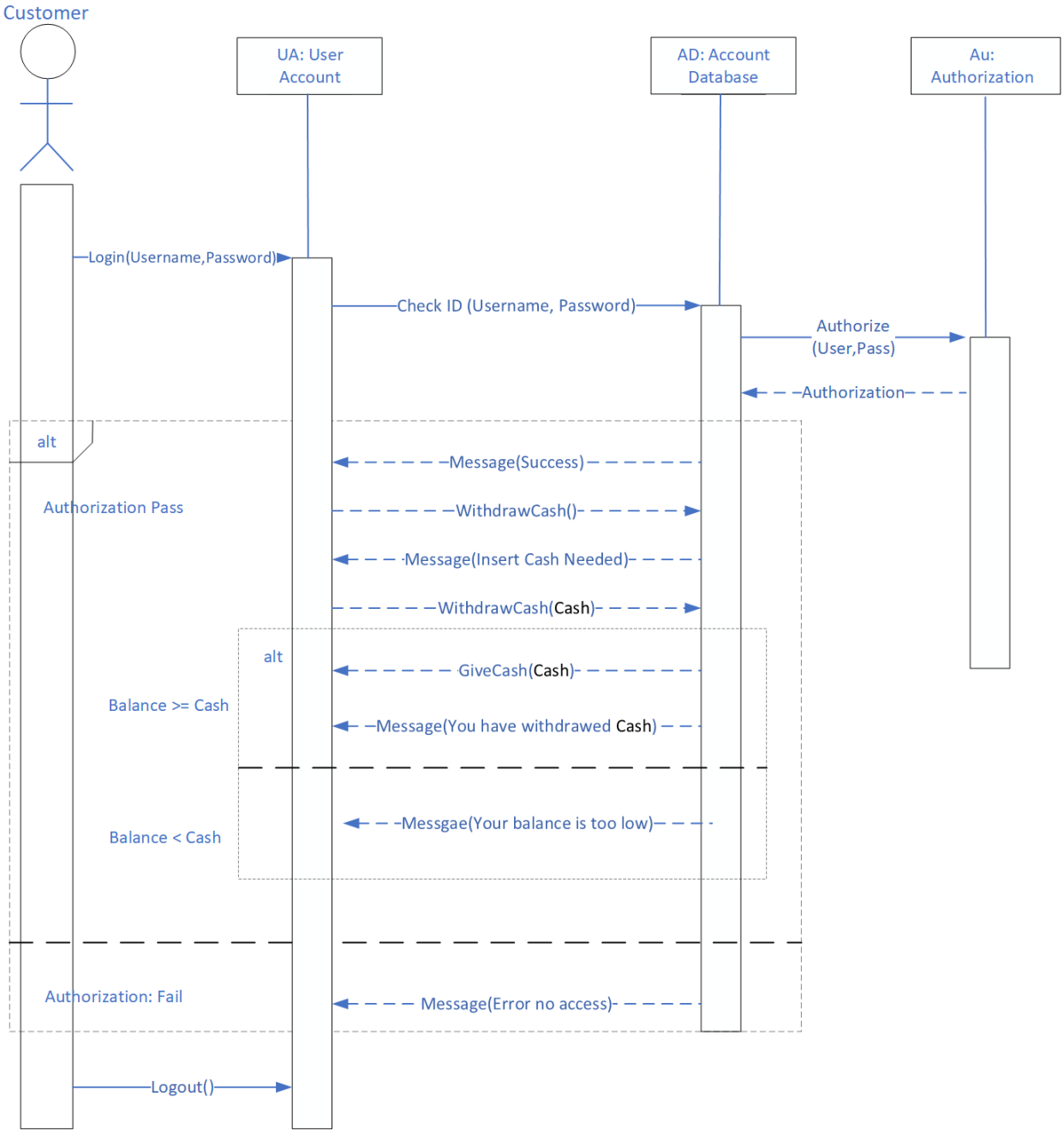
Note: The design patterns that has been used in order to improve the system that might change the way the classes interact with each other or the relationships between the classes. (See Class Diagram, 4.1)

4.4 Behavioral Modeling

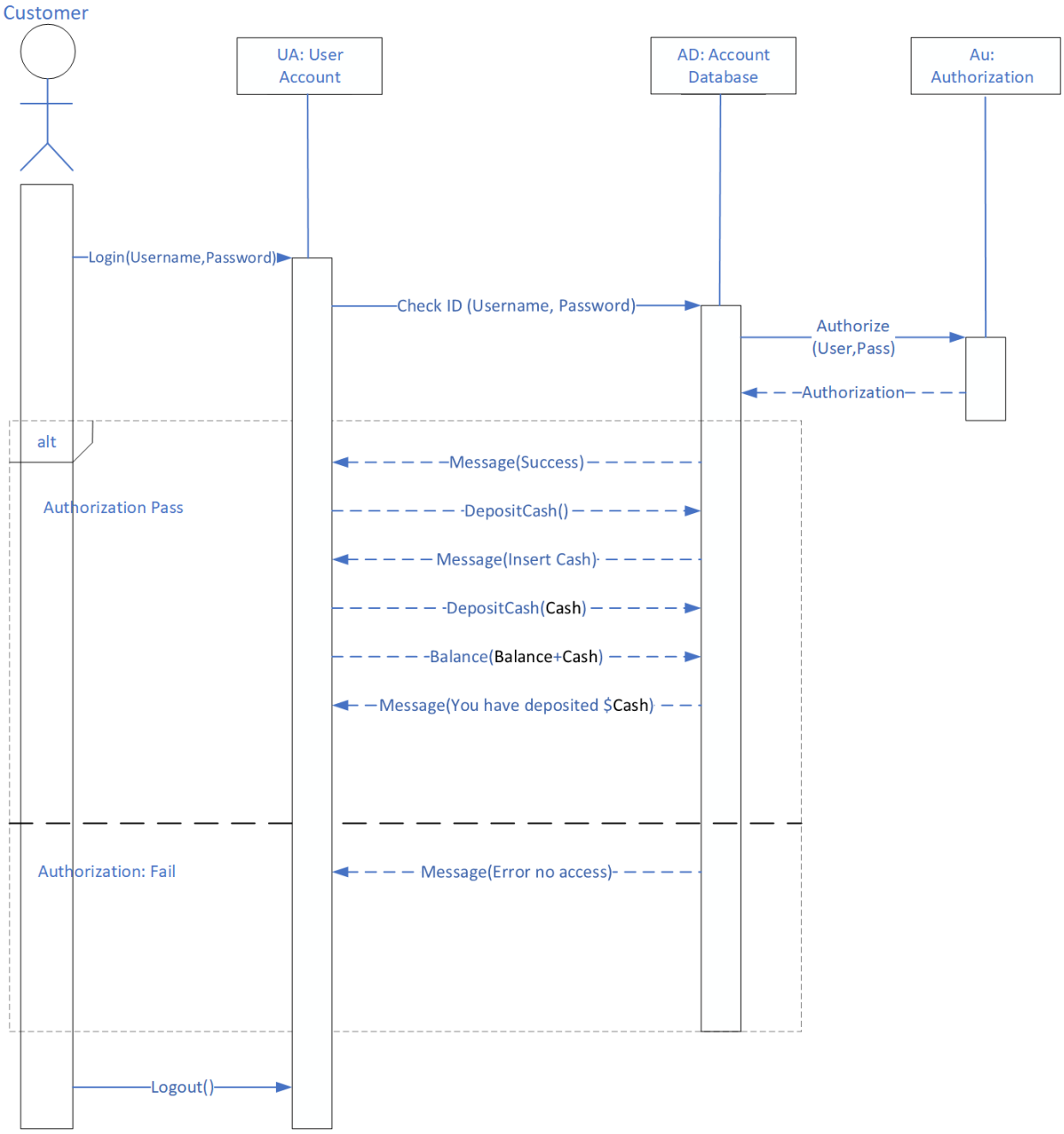
Use Case Diagram:



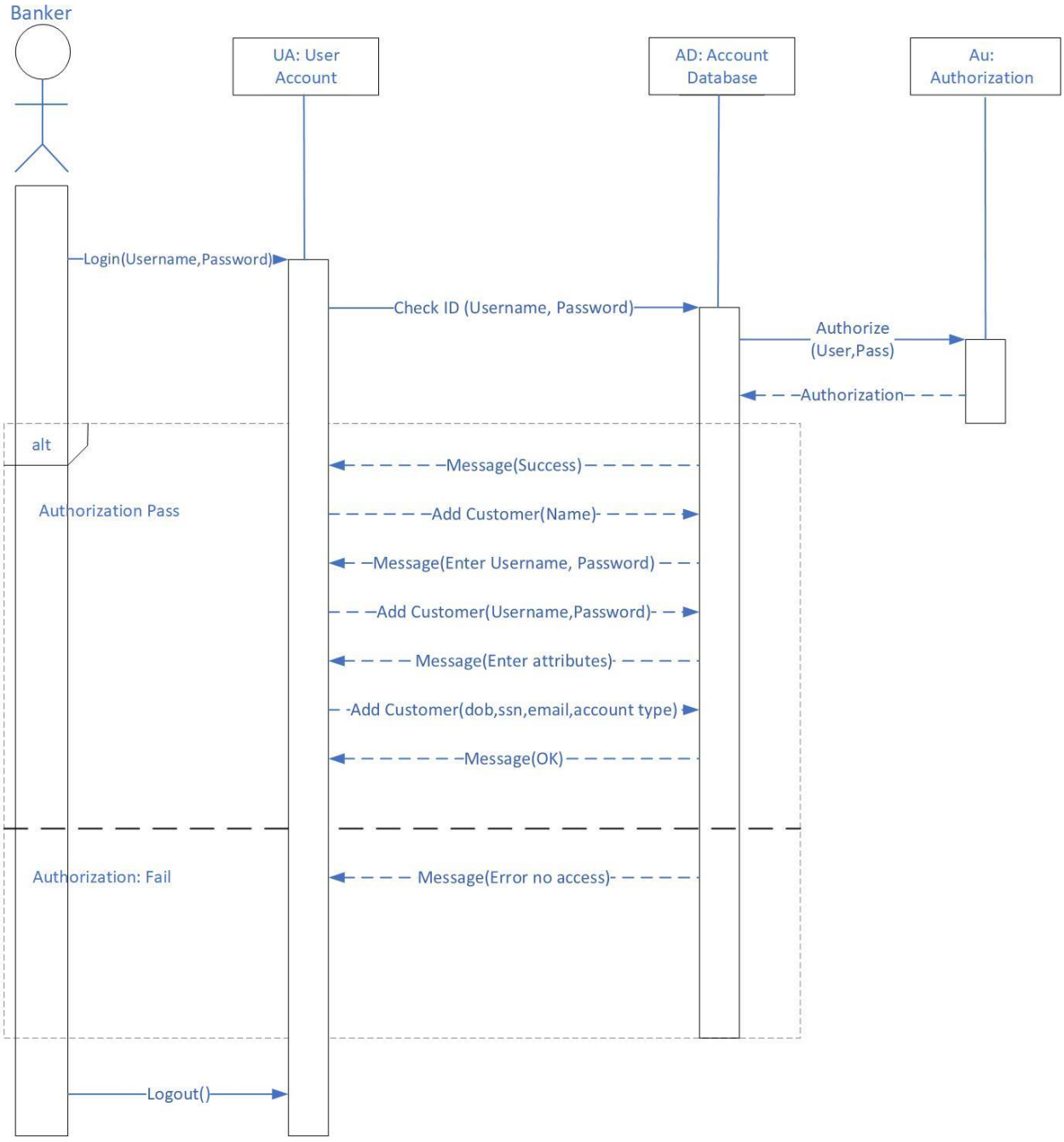
Sequence Diagram for Withdrawing Cash (Use Case 1):



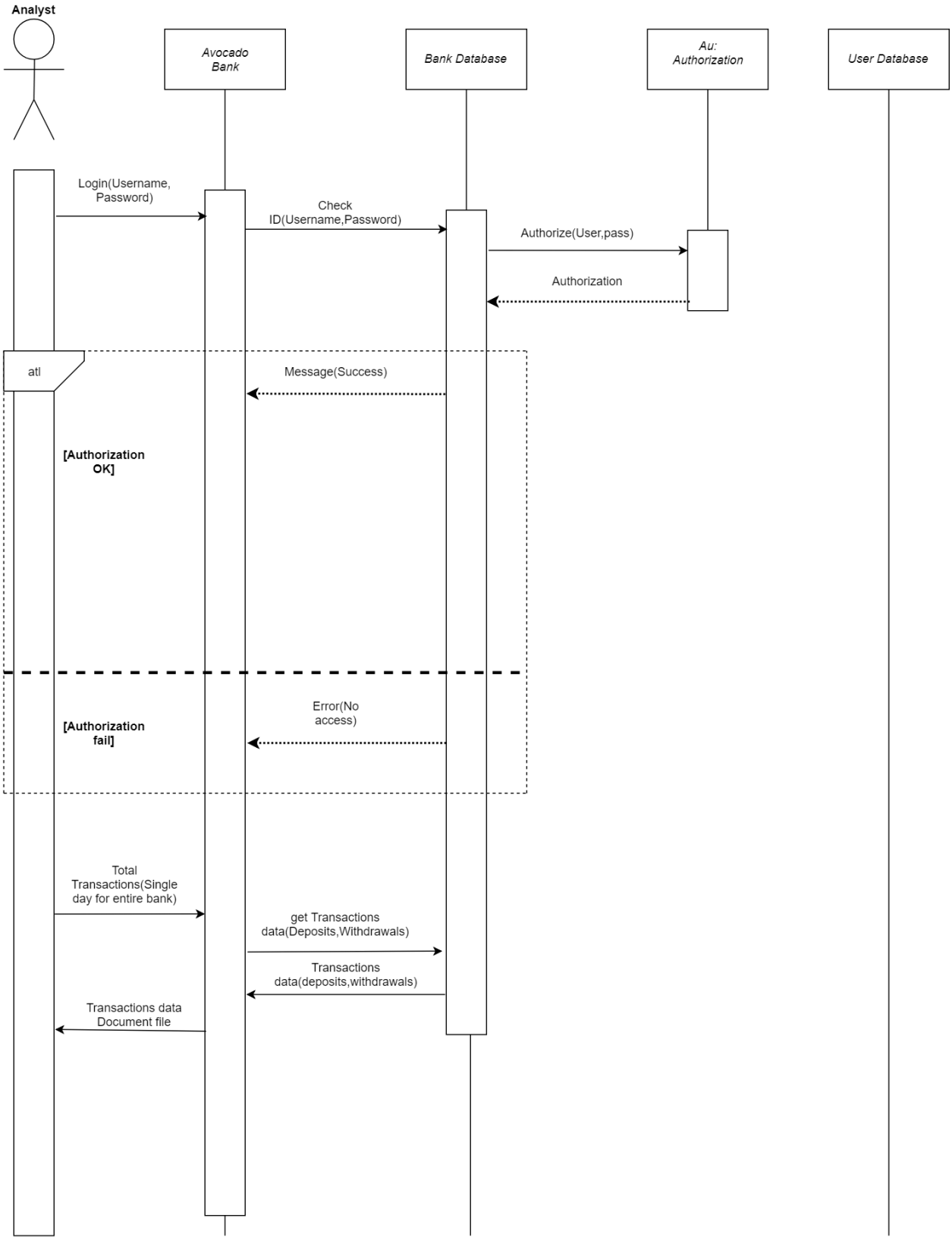
Sequence Diagram for Depositing Cash (Use Case 2):



Sequence Diagram for Adding Customer (Use Case 3):



Sequence Diagram for Get Transactional Data Analyst (Use Case 4):



5. Implementation

Clone project repository (See GitHub in Appendix):

1. For the backend, open up a console in 'backend' and run 'gradlew.bat run'
2. For the front end open up a console in frontend and run 'npm install; npm start'
3. The actual site is https://avocado-toast.wp6.pw/
4. The username is admin and password is admin

6. Testing

6.1 Functionality Testing:

Feature One, Login:

Partition Input**:** login(String U, string P)

- String U is a string for the user's Username
    - One possible partition is string with length < 0, string with length = 0, string with length > 0
- String P is a string for the user's Password
    - One possible partition is string with length < 0, string with length = 0, string with length > 0

Test Specification:

- Username: <String a-z, A-Z, 0-9, special characters>
- Password: <String a-z, A-Z, 0-9, special characters>

Test Case:

#1:

- Inputs:
    - Username: admin
    - Password: admin
- Outputs:
    - Logs in

#2:

- Inputs:
    - Username:
    - Password:

- Outputs:
    - Please input your username! Please input your Password!

Feature Two, Send Money:

Partition Input: sendMoney(String ID, int N)

- String ID is a string for the recipient's Username
    - One possible partition is string with length < 0, string with length = 0, string with length > 0
- double N is an integer between 0 and the user's maximum balance
    - One possible partition is a number: < -∞ - 0.00,0.00-∞, 0.00-…*>, …* is the user's maximum balance

Test Specification:

- Recipient's ID: <String a-Z, A-Z, 0-9, special characters>
- Amount: <0…*>, * is the user' maximum balance

Test Case:

Assume user has $10 in balance

#1:

- Inputs:
    - ID: bobross
    - Amount: $3.01
- Outputs:
    - Successfully sent $3.01 to bobross

#2:

- Inputs:
    - ID:
    - Amount: $3.01
- Outputs:
    - Please add a destination

#3:

- Inputs:

  - ID: bobross
  - Amount:$ -3.01
- Outputs:
  - You can only transfer money out of your account

#4:

- Inputs:
  - ID: bobross
  - Amount: $500
- Outputs:
  - You can't transfer more money than you have

6.2 Unit Testing Documentation:

Test ID	*createUserTest()*
Purpose of Test	*To test if a user object can be created and inserted into database successfully*
Test Environment	*The environment used is JUnit because we wrote our unit tests in IntelliJ IDE Java platform with Windows OS*
Test Steps	*Tester must import the code from our github directory : backend/src/test/java/pw/wp6/avocado_toast/* *api/UserApiControllerIntegrationTest.java* *into a Java IDE with JUnit to proceed running the test*

Test Input	*CreateUserObject body = new CreateUserObject();* *body.setName("John Doe");* *body.setUsername("jdoe1");* *body.setAccountType(AccountType.CUSTOMER);* *body.setPassword("pass123");* *body.setSsn("123456");* *Name: John Doe* *Username: jdoe1* *Account Type: Customer* *Password: pass123* *Ssn: 123456*
Expected Result	*Test passed: 1* *The test uses an .assertEquals() methods that checks if two objects are equals or not. If they are not, an AssertionError without a message is thrown. It checks if the createUser() method returns "HttpStatus.NOT_IMPLEMENTED", meaning the server does not support the functionality required to fulfill the request.*
Likely Problems/Bugs Revealed	*Vulnerability to SQL injections in username and password field to bypass user authentication* *Vulnerability to SQL injections in username and password field to data manipulation (inserting, deleting, changing data, etc)*

Test ID	*loginUserTest()*
Purpose of Test	*To test if a user object's username and password can be used to access system functionalities.*
Test Environment	*The environment used is JUnit because we wrote our unit tests in IntelliJ IDE Java platform with Windows OS*
Test Steps	*Tester must import the code from our github directory : backend/src/test/java/pw/wp6/avocado_toast/* *api/UserApiControllerIntegrationTest.java* *into a Java IDE with JUnit to proceed running the test*
Test Input	*LoginParameters body = new LoginParameters();* *body.setUserName("jdoe1");* *body.setPassword("pass123");* *Username: jdoe1* *Password: pass123*
Expected Result	*Test passed: 1* *The test uses an .assertEquals() methods that checks if two objects are equals or not. If they are not, an AssertionError without a message is thrown. It checks if the loginUser() method returns "HttpStatus.NOT_IMPLEMENTED", meaning the server does not support the functionality required to fulfill the request.*

Likely Problems/Bugs Revealed	*Vulnerability to SQL injections in username and password field to bypass user authentication* *Vulnerability to SQL injections in username and password field to data manipulation (inserting, deleting, changing data, etc)*

6.3 Bug Documentation:

Bug	The test uncovered the bug	Description of the bug	Action was taken to fix the bug
Vulnerability to SQL injections to bypass user authentication	loginUserTest()	A user could bypass our authentication software and enter the website without a valid username and password	Change the permission and privileges so user does not have privilege to login if its invalid
Vulnerability to SQL injections in username, password, and SSN ID field to data manipulation (inserting, deleting, changing data, etc)	loginUserTest() createUserTest()	A user could bypass manipulate our data and insert, delete, or update the data in the username, password, and SSN ID field	Change permission and privilege so user does not have privilege to change data

7. Appendix

GitHub: https://github.com/gsu-swe-s2019-group-1

Website: https://avocado-toast.wp6.pw/

Project Page: