Avocado Toast: An Online Banking Platform

Spring 2019

Group 1

Alex Petros • Anandita Dubey • Danh Pham

Carlos Deleon • Flaviu Tamas

Planning and Scheduling

Assignment 1

Assignee	Email	Task	Dur. (h)	Dependency	Due Date	Evaluation
Alex Petros	apetros1	System architecture	2	GitHub, GroupMe	2019-01-31	Great job, did 100%
Anandita Dubey	adubey2	User Requirements (1-5)	2	GitHub, GroupMe	2019-01-31	Great job, did 100%
Danh Pham	dpham16	User requirements (6-10)	2	GitHub, GroupMe	2019-01-31	Great job, did 100%
Carlos Deleon	cdeleon1	Teamwork basics	2	GitHub, GroupMe, System architecture	2019-01-31	Great job, did 100%
Flaviu Tamas	ftamas1	Create GitHub & GroupMe, report	2	None	2019-02-01	Great job, did 100%

Assignment 2

Assignee	Email	Task	Dur. (h)	Dependency	Due Date	Evaluation
Alex Petros	apetros1	System model	2	None	2019-02-14	Great job, did 100%
Anandita Dubey	adubey2	Refine problem statement	2	None	2019-02-14	Great job, did 100%
Danh Pham	dpham16	Use case requirements	2	Use cases	2019-02-14	Great job, did 100%
Carlos Deleon	cdeleon1	Use case diagram	2	Use cases	2019-02-14	Great job, did 100%
Flaviu Tamas	ftamas1	Use cases	2	None	2019-02-11	Great job, did 100%

Assignment 3

Assignee	Email	Task	Dur. (h)	Dependency	Due Date	Evaluation
Alex Petros	apetros1	Implementation (Backend)	2	Use cases	2019-03-03	Great job, did 100%
Anandita Dubey	adubey2	Refine System Architecture Model	2	Implementation (Frontend)	2019-03-03	Great job, did 100%

Danh Pham	dpham16	Testing	2	Use cases	2019-03-02	Great job, did 100%
Carlos Deleon	cdeleon1	Behavioral Model	2	Implementation	2019-03-02	Great job, did 100%
Flaviu Tamas	ftamas1	Implementation (Frontend)	2	None	2019-03-02	Great job, did 100%

Assignment 4

Assignee	**Email**	**Task**	**Dur. (h)**	**Dependency**	**Due Date**	**Evaluation**
Alex Petros	apetros1	Revise and refine your system	2	System requirements, system modeling	2019-03-15	Great job, did 100%
Anandita Dubey	adubey2	System modeling	2	Use cases, class diagrams	2019-03-15	Great job, did 100%
Danh Pham	dpham16	Testing	2	Implementation	2019-03-15	Great job, did 100%
Carlos Deleon	cdeleon1	Planning, Scheduling, Collaboration	2	None	2019-03-15	Great job, did 100%
Flaviu Tamas	ftamas1	Implementation	2	Use cases, previous implementation	2019-03-15	Great job, did 100%

Assignment 5

Assignee	**Email**	**Task**	**Dur. (h)**	**Dependency**	**Due Date**	**Evaluation**
Alex Petros	apetros1	Write unit tests	2	Previous implementation	2019-04-13	Great job, did 100%
Anandita Dubey	adubey2	Determine cohesion & coupling, document design patterns	2	Previous implementation	2019-04-13	Great job, did 100%
Danh Pham	dpham16	Write test documentation	2	Previous implementation	2019-04-13	Great job, did 100%
Carlos Deleon	cdeleon1	Improve database schema	2	Previous implementation	2019-04-13	Great job, did 100%
Flaviu Tamas	ftamas1	Coordinate, write report	2	All other tasks	2019-04-13	Great job, did 100%

Problem Statement

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.

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 to visit a physical bank during working hours or make a phone call.

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 enough to do any task in a single 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.

What alternatives are available?

- Services provided by 3rd party application, like Mint
- A phone app
- In person banking services
- Phone calls

- ATMs

Why is this project compelling and worth developing?

- It would reduce costs for banks by reducing the amount of human labor required.
- 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 WiFi.

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.
- 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.

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. Additionally, our platform is customer focused and will be fully transparent. We won’t sell customer data or charge hidden fees.

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.

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.

Changes

There have been no significant changes to our problem statement since it was written.

Requirements

User Requirements

1: Register Customer

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

2: Deposit Cash

- 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: Withdraw Cash

- 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

4: View Transactions

- 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

5: Transfer Funds

- Actors: Customer
- Description: Send someone else money electronically through the website
- Alternate Path: None
- Pre-Condition: Logged in, account has sufficient balance

6: Analyze Transaction Volume

- 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

7: Log in

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

Use case requirements

1: Register Customer

- Introduction: 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

2: Deposit Cash

- Introduction: 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: Withdraw Cash

- Introduction: 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

4: View Transactions

- Introduction: 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

5: Transfer Funds

- Introduction: 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

6: Analyze Transaction Volume

- Introduction: 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

7: Log in

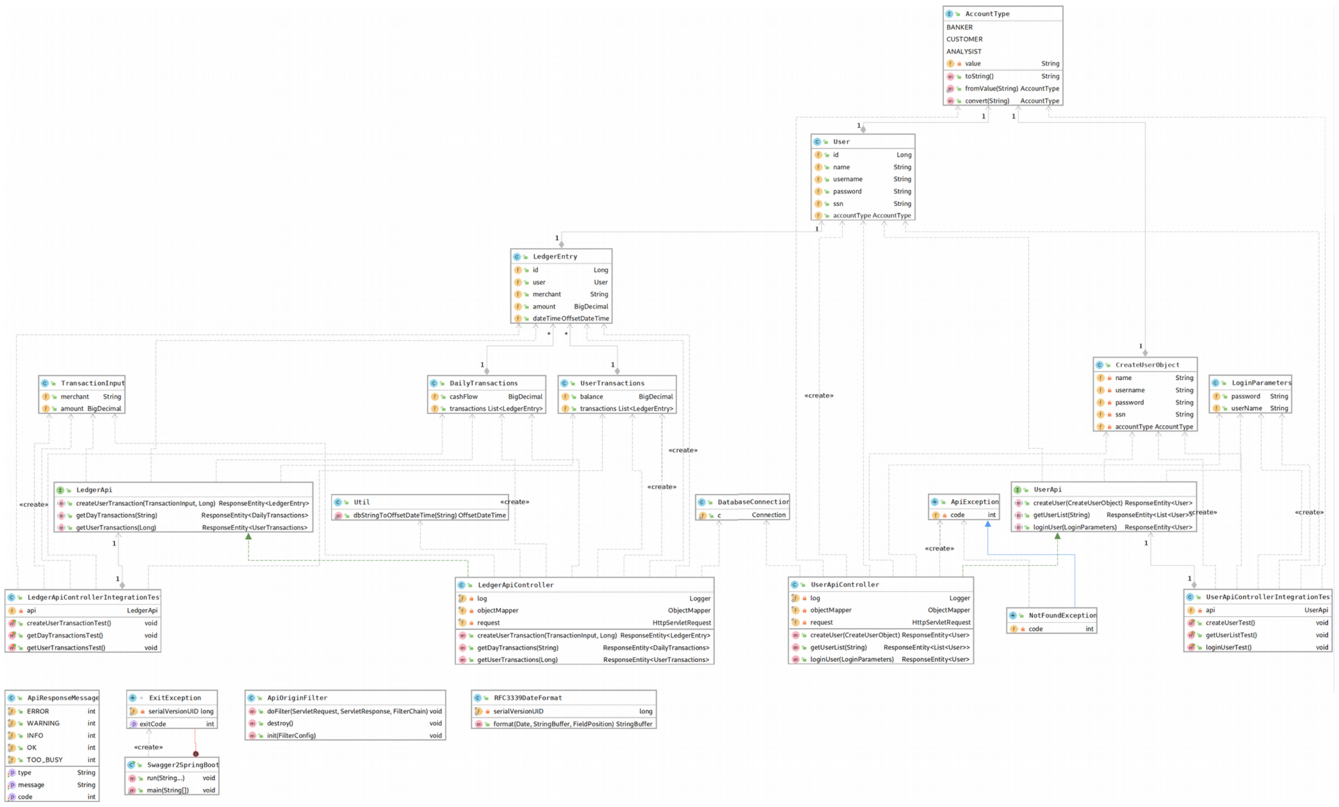
- Introduction: this system interacts with the user and authentication utility
- Inputs: User ID and password
- Requirements Description: Authenticate a user to their account
- Outputs: User is logged into the system and given access to use their account
- Alternate Path: None

System Requirements

- CPU: Any x86_64 or ARM processor
- RAM: 200MiB min
- OS: Window 10, macOS, GNU/Linux

System Modeling

Class Diagram


AccountType
BANKER
CUSTOMER
ANALYSIST
value String
toString() String
fromValue(String) AccountType
convert(String) AccountType
User
id Long
name String
username String
password String
ssn String
accountType AccountType
LedgerEntry
id Long
user User
merchant String
amount BigDecimal
dateTime OffsetDateTime
TransactionInput
merchant String
amount BigDecimal
DailyTransactions
cashFlow BigDecimal
transactions List<LedgerEntry>
UserTransactions
balance BigDecimal
transactions List<LedgerEntry>
CreateUserObject
name String
username String
password String
ssn String
accountType AccountType
LoginParameters
password String
userName String
LedgerApi
createUserTransaction(TransactionInput, Long) ResponseEntity<LedgerEntry>
getDayTransactions(String) ResponseEntity<DailyTransactions>
getUserTransactions(Long) ResponseEntity<UserTransactions>
Util
dbStringToOffsetDateTime(String) OffsetDateTime
DatabaseConnectior
c Connection
ApiException
code int
UserApi
createUser(CreateUserObject) ResponseEntity<User>
getUserList(String) ResponseEntity<List<User>>
loginUser(LoginParameters) ResponseEntity<User>
LedgerApiControllerIntegrationTes
api LedgerApi
createUserTransactionTest() void
getDayTransactionsTest() void
getUserTransactionsTest() void
LedgerApiController
log Logger
objectMapper ObjectMapper
request HttpServletRequest
createUserTransaction(TransactionInput, Long) ResponseEntity<LedgerEntry>
getDayTransactions(String) ResponseEntity<DailyTransactions>
getUserTransactions(Long) ResponseEntity<UserTransactions>
UserApiController
log Logger
objectMapper ObjectMapper
request HttpServletRequest
createUser(CreateUserObject) ResponseEntity<User>
getUserList(String) ResponseEntity<List<User>>
loginUser(LoginParameters) ResponseEntity<User>
NotFoundExceptior
code int
UserApiControllerIntegrationTes
api UserApi
createUserTest() void
getUserListTest() void
loginUserTest() void
ApiResponseMessage
ERROR int
WARNING int
INFO int
OK int
TOO_BUSY int
type String
message String
code int
ExitException
serialVersionUID long
exitCode int
Swagger2SpringBoot
run(String...) void
main(String[]) void
ApiOriginFilter
doFilter(ServletRequest, ServletResponse, FilterChain) void
destroy() void
init(FilterConfig) void
RFC3339DateFormat
serialVersionUID long
format(Date, StringBuffer, FieldPosition) StringBuffer
«create»
1
*

Database Specification

Tables

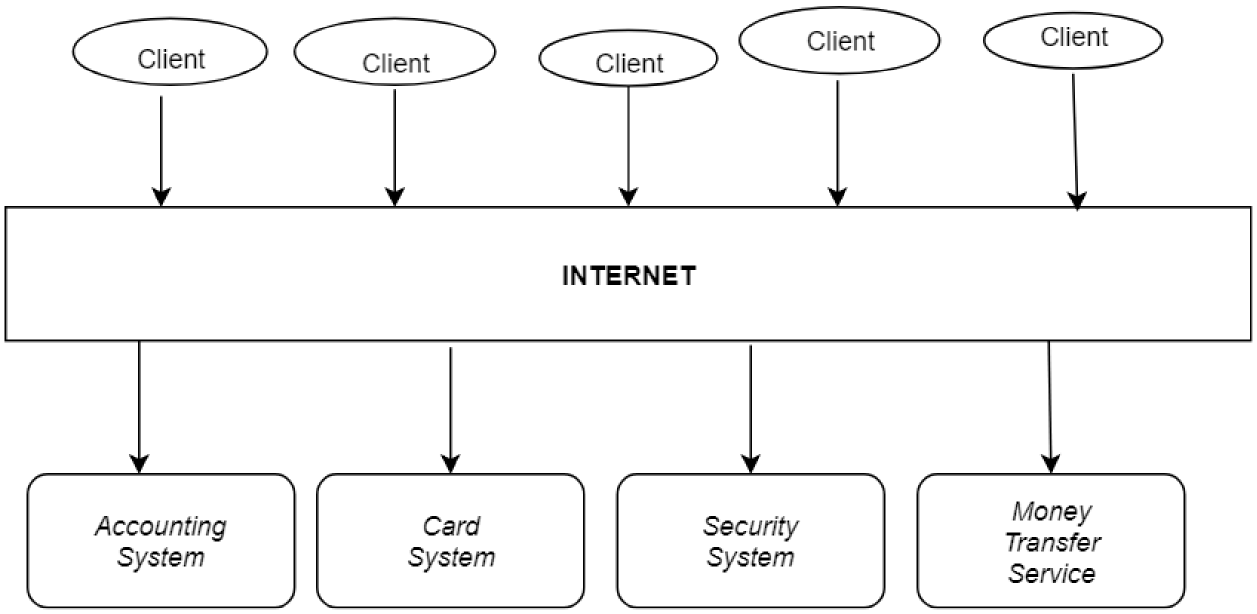
```
create table Analyst (
     id INTEGER primary key,
     username TEXT not null unique,
     password TEXT not null,
     name TEXT not null,
     account_type TEXT not null
);
create table Banker (
     id INTEGER primary key,
     username TEXT not null unique,
     password TEXT not null,
     name TEXT not null,
     account_type TEXT not null
);
create table Customer (
     id INTEGER primary key,
     username TEXT not null unique,
     password TEXT not null,
     name TEXT not null,
     ssn INTEGER not null,
     account_type TEXT not null
);
create table Transactions (
     id INTEGER primary key,
     customer_user_id INTEGER not null references Customer,
     merchant TEXT not null,
     amount INTEGER not null,
     date_time STRING not null
);
```

Database system

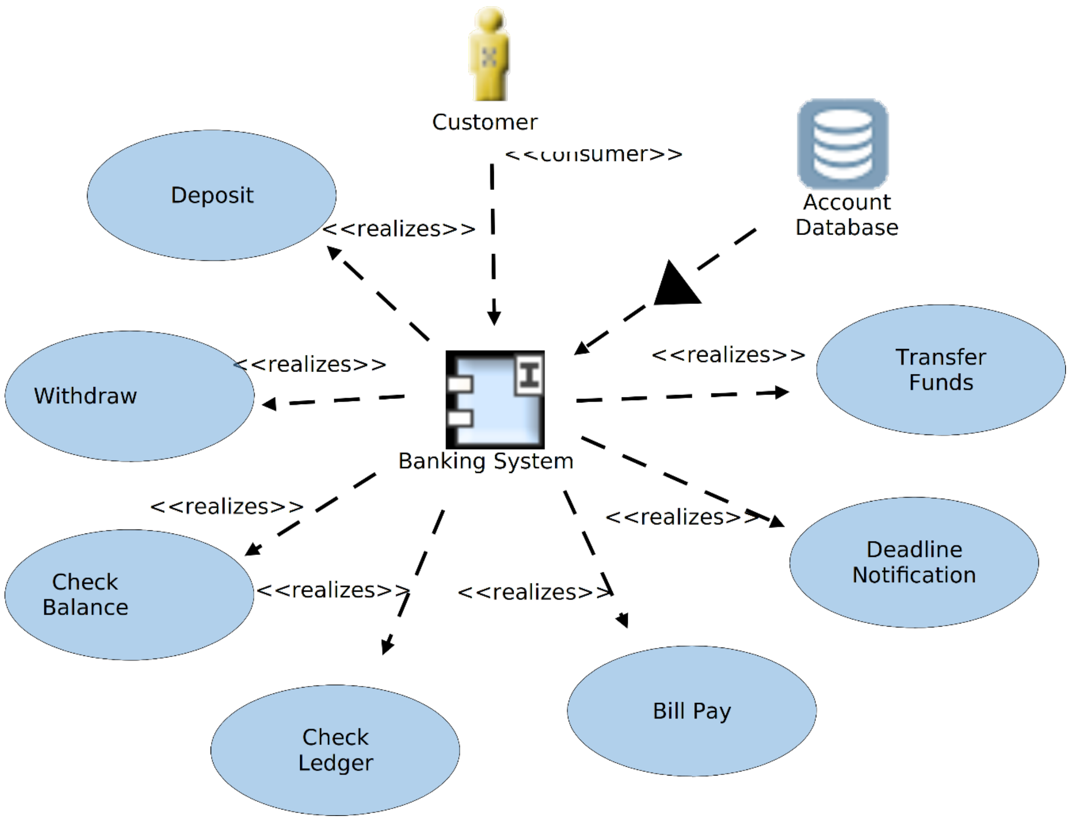
This project uses SQLite for the database.

Architecture modeling

Client Server pattern

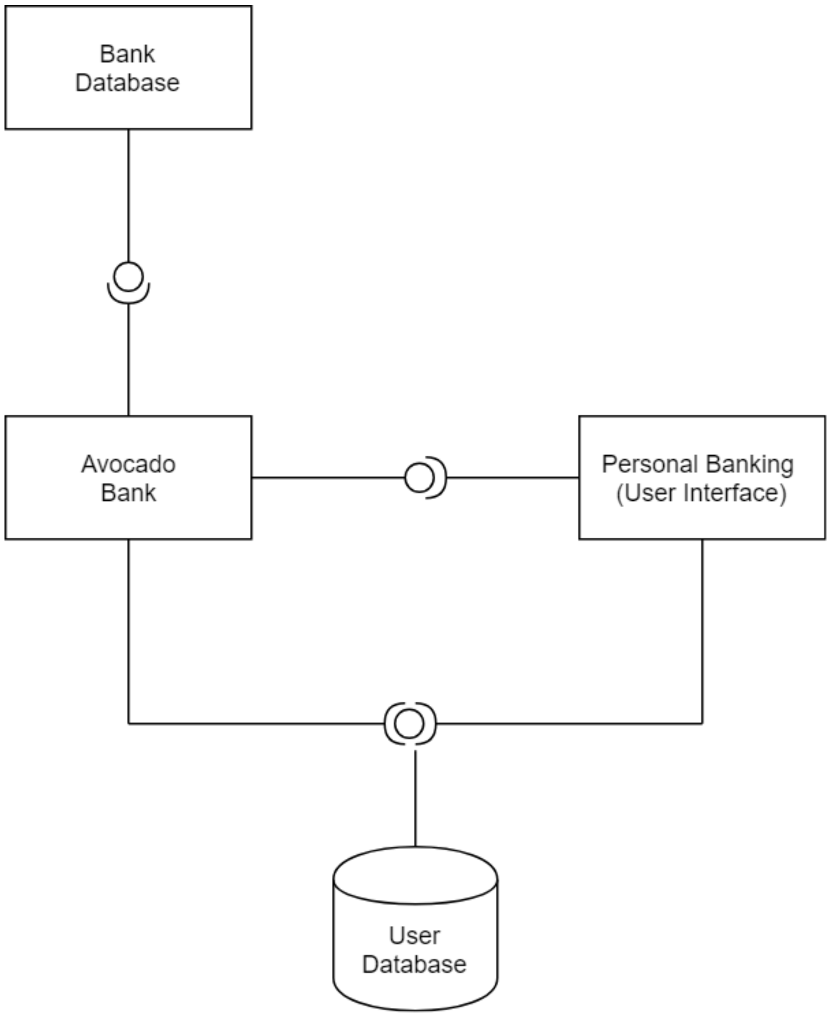


Context Model

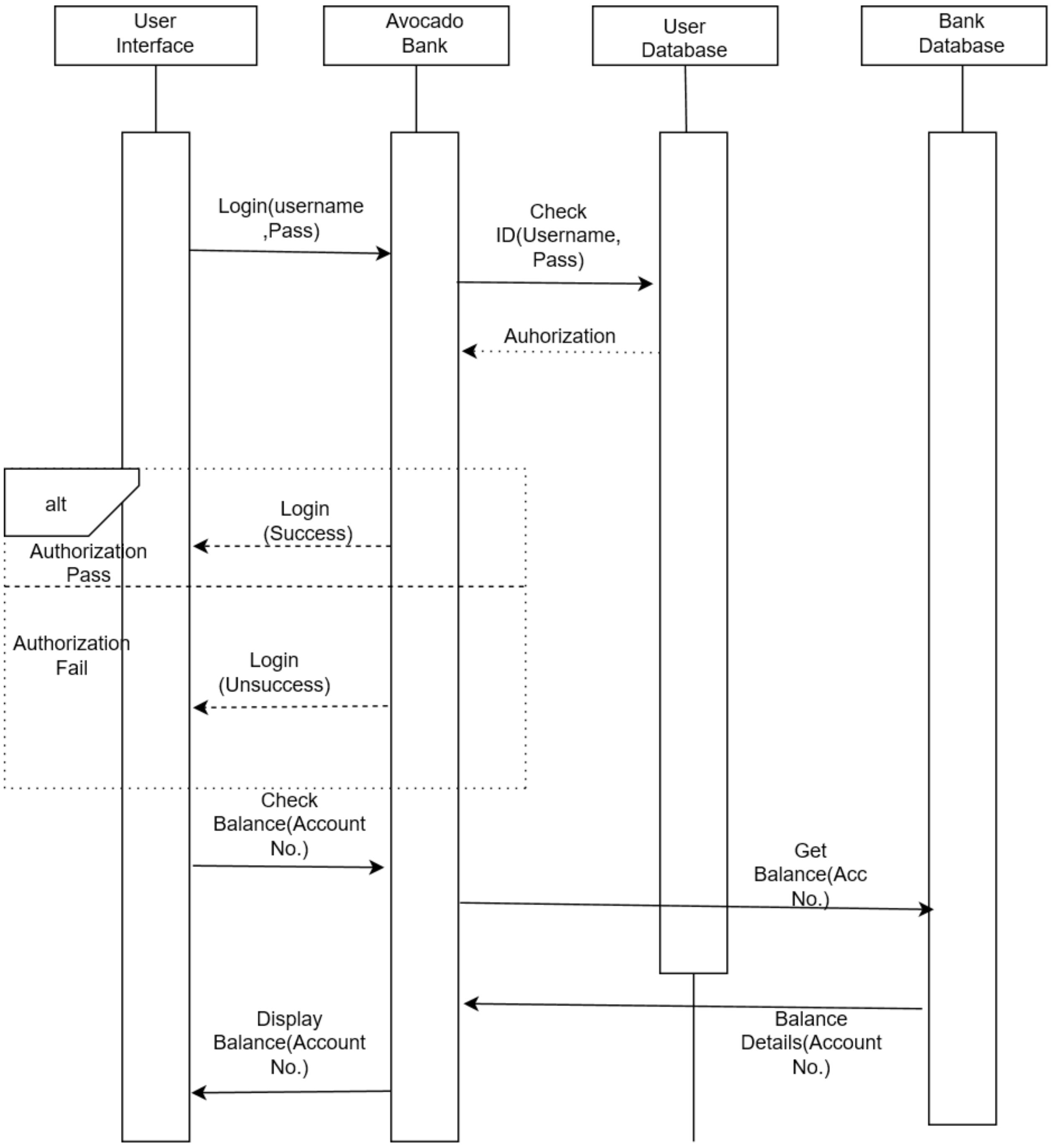


Views

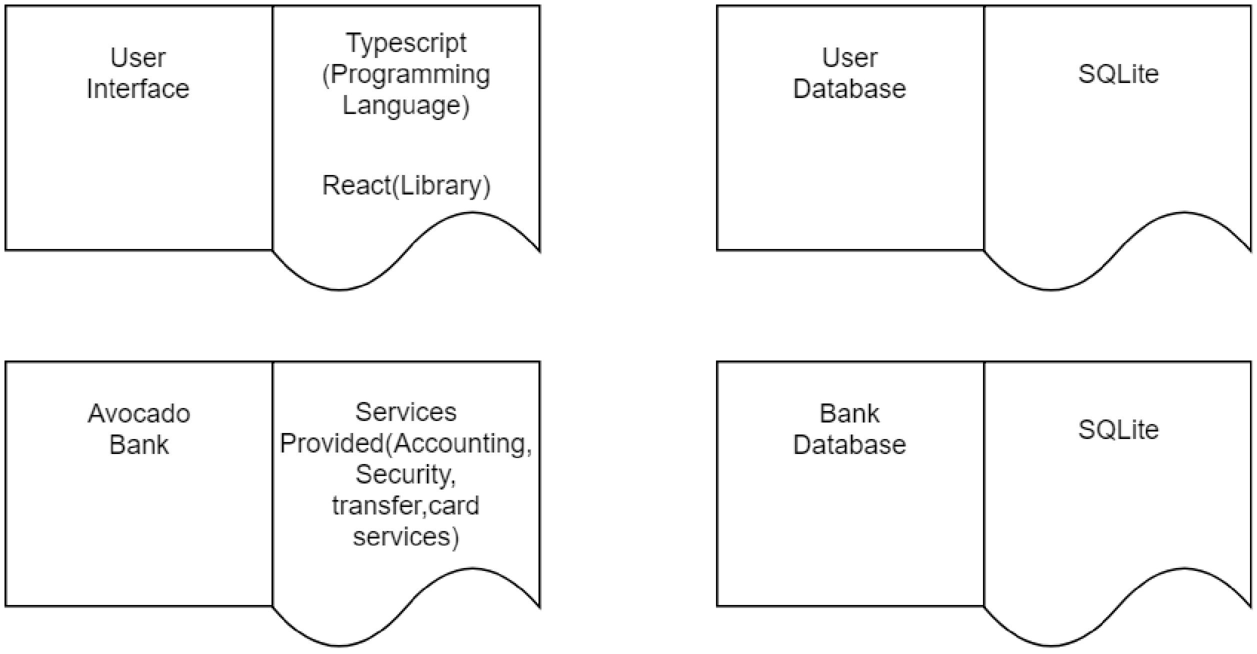
Logical



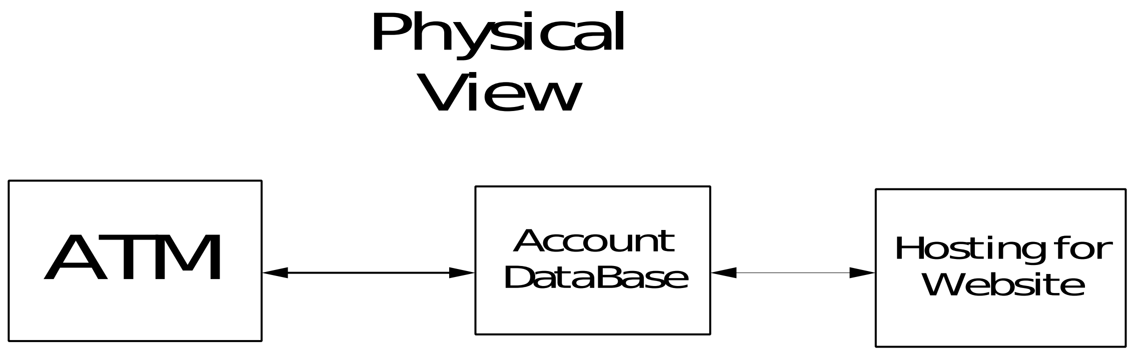
Process


User Interface
Avocado Bank
User Database
Bank Database
Login(username ,Pass)
Check ID(Username, Pass)
Auhorization
alt
Authorization Pass
Login (Success)
Authorization Fail
Login (Unsuccess)
Check Balance(Account No.)
Get Balance(Acc No.)
Balance Details(Account No.)
Display Balance(Account No.)

Development


User Interface
Typescript (Programming Language)
React(Library)
User Database
SQLite
Avocado Bank
Services Provided(Accounting, Security, transfer,card services)
Bank Database
SQLite

Physical



Coupling & Cohesion

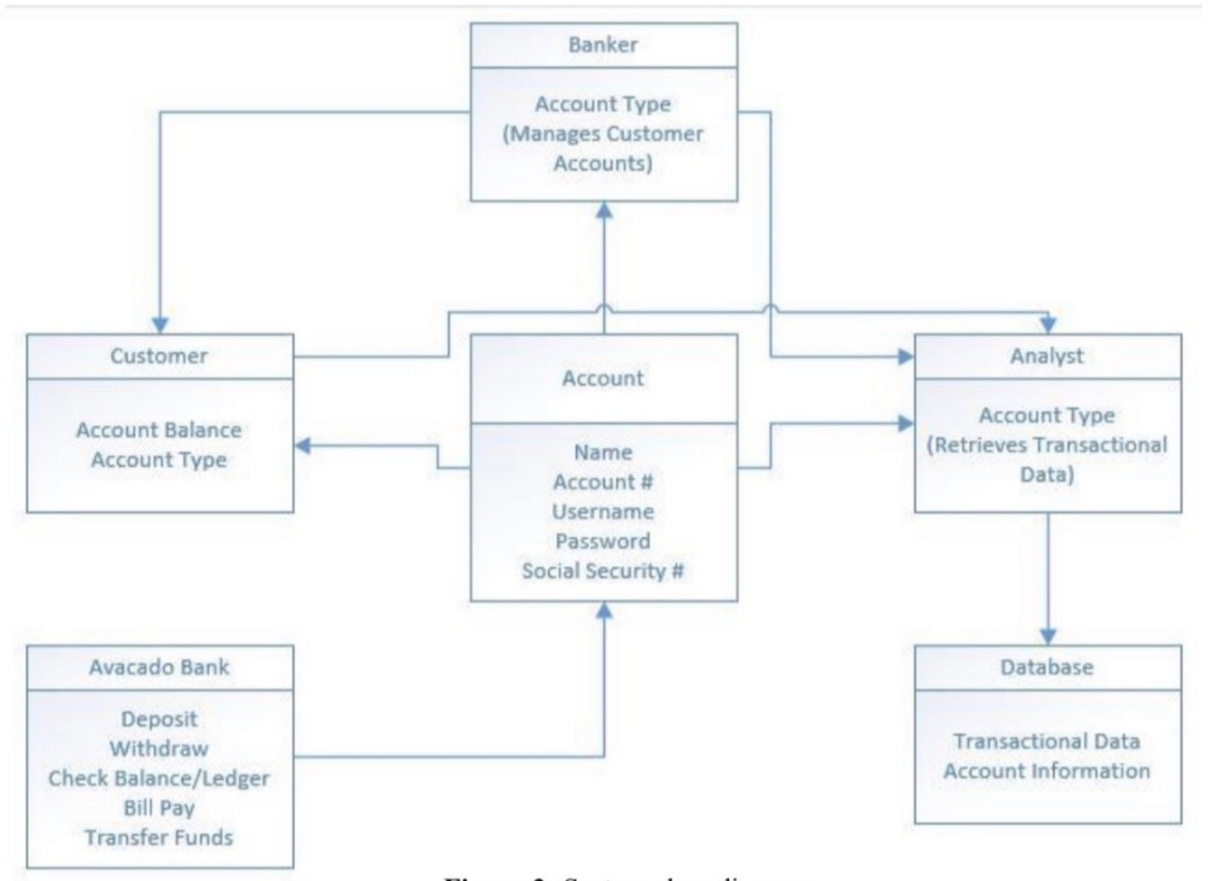


Figure 3: System class diagram

Cohesion Formula

cohesion(Class n) = Classes on which n directly-indirectly potentially depends/total no. of classes

Calculations:

Banker class

Cohesion = 3/6=0.5

Customer class

Cohesion= 2/6= 0.34

Avocado Bank

Cohesion = 5/6=0.83

Analyst class

Cohesion = 1/6=0.16

Database class

Cohesion = 0/6=0

Account class

Cohesion = 4/6=0.67

Classes	Directly or indirectly depends on	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

Coupling Formula

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 (c2, c1) implies that information flow from class c2 to class c1 in a modular P, employee information modular.

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

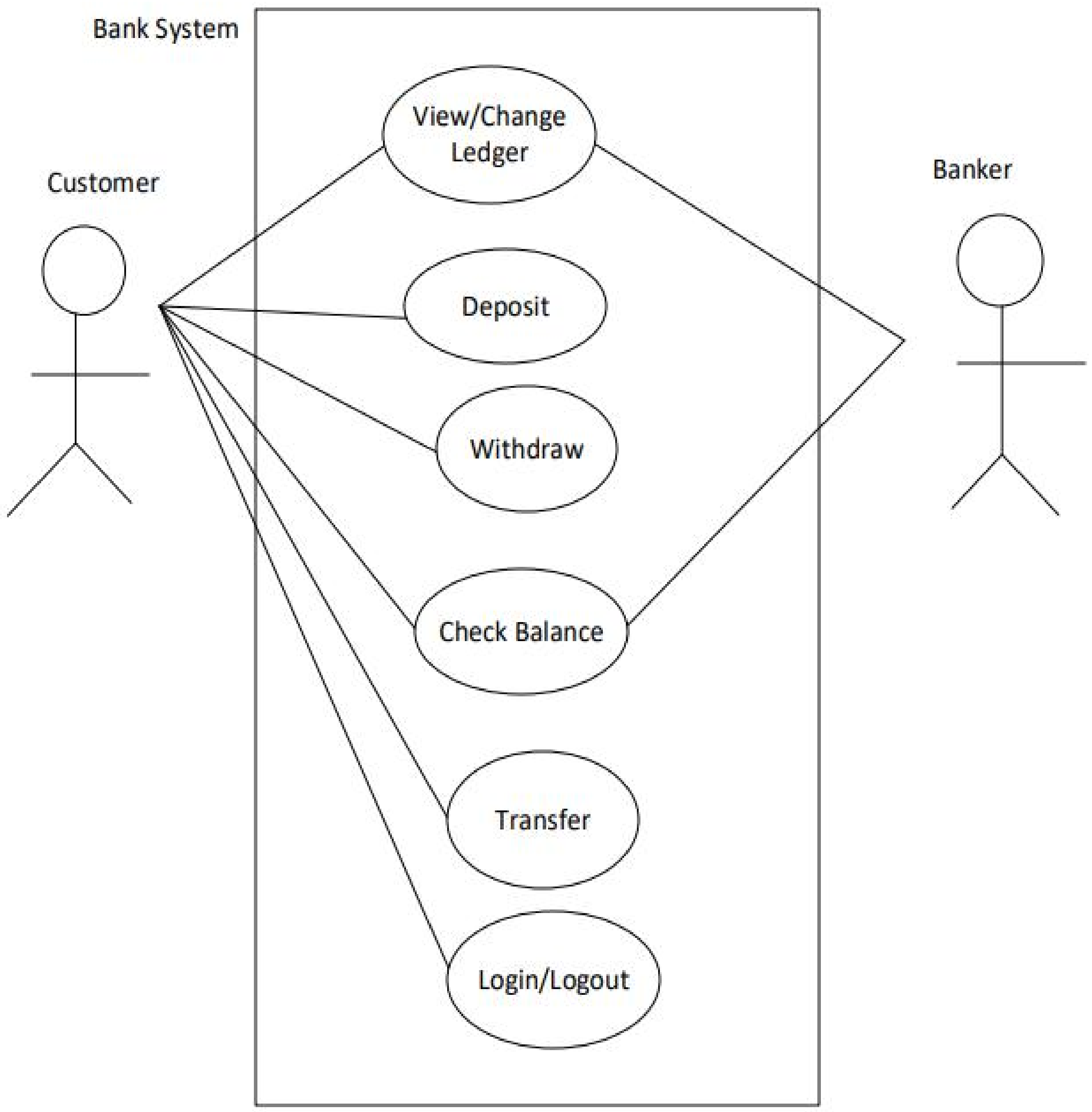
Design Patterns

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)

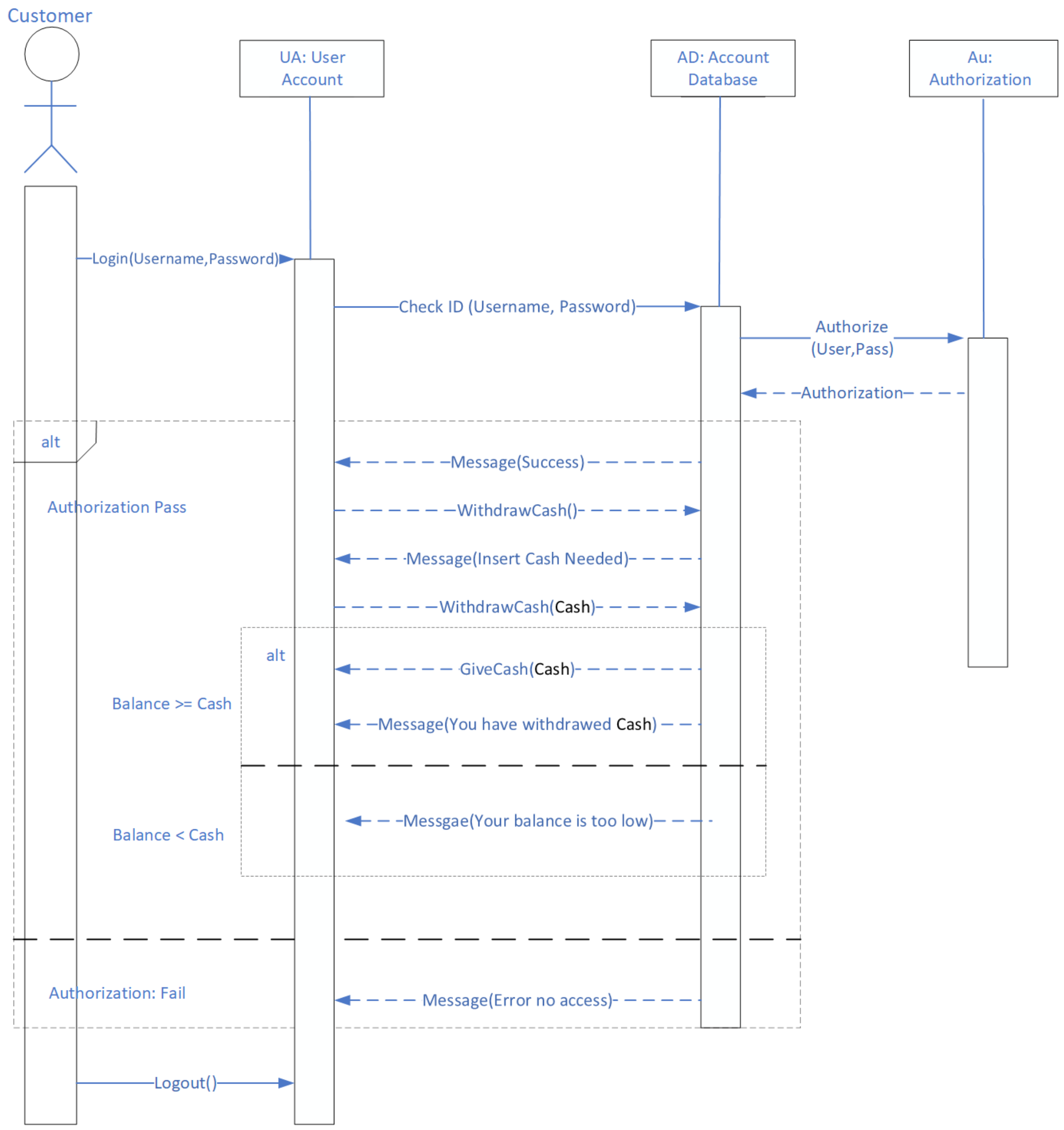
Behavioral modeling

Use Case Diagram

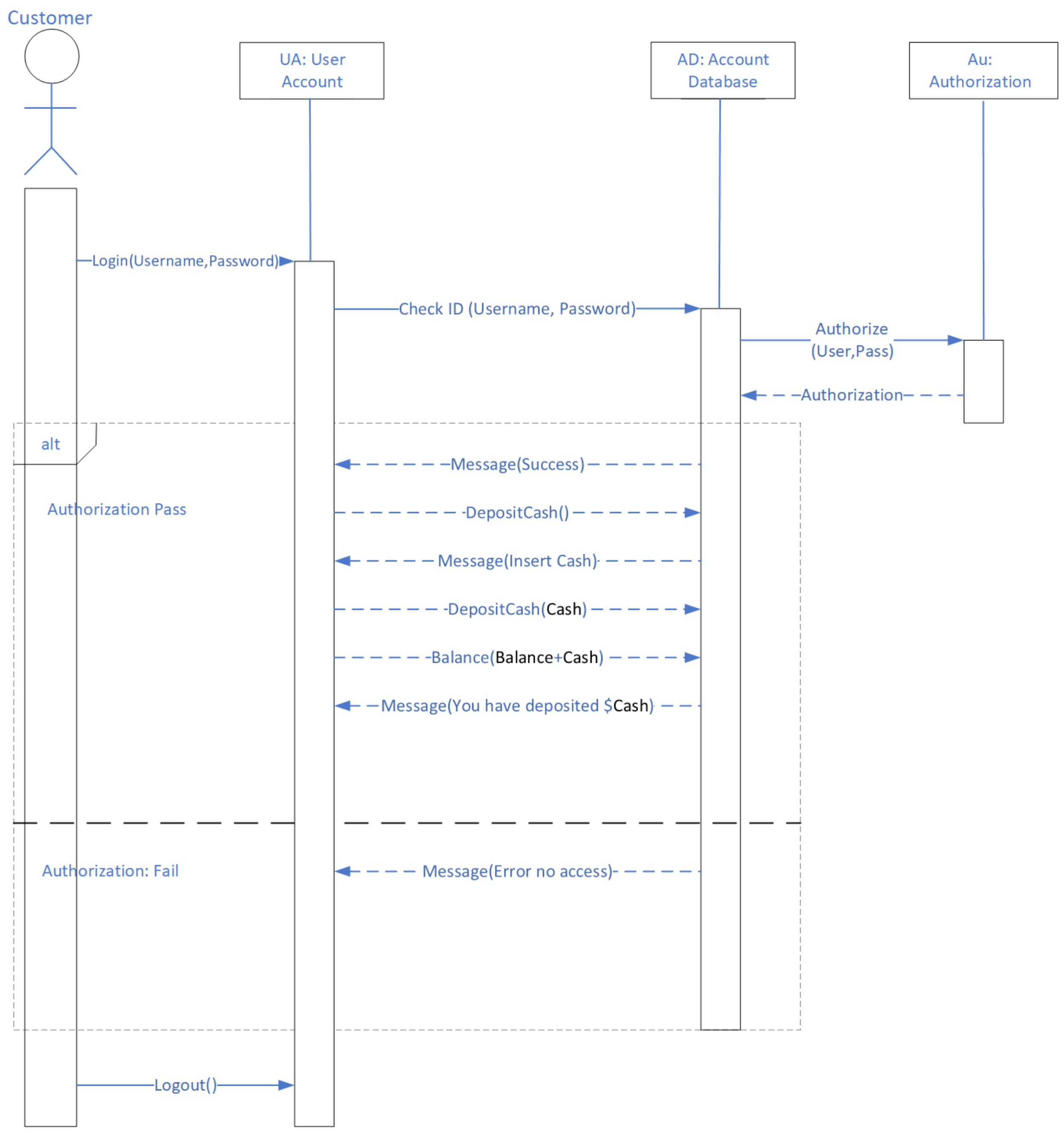


Sequence Diagrams

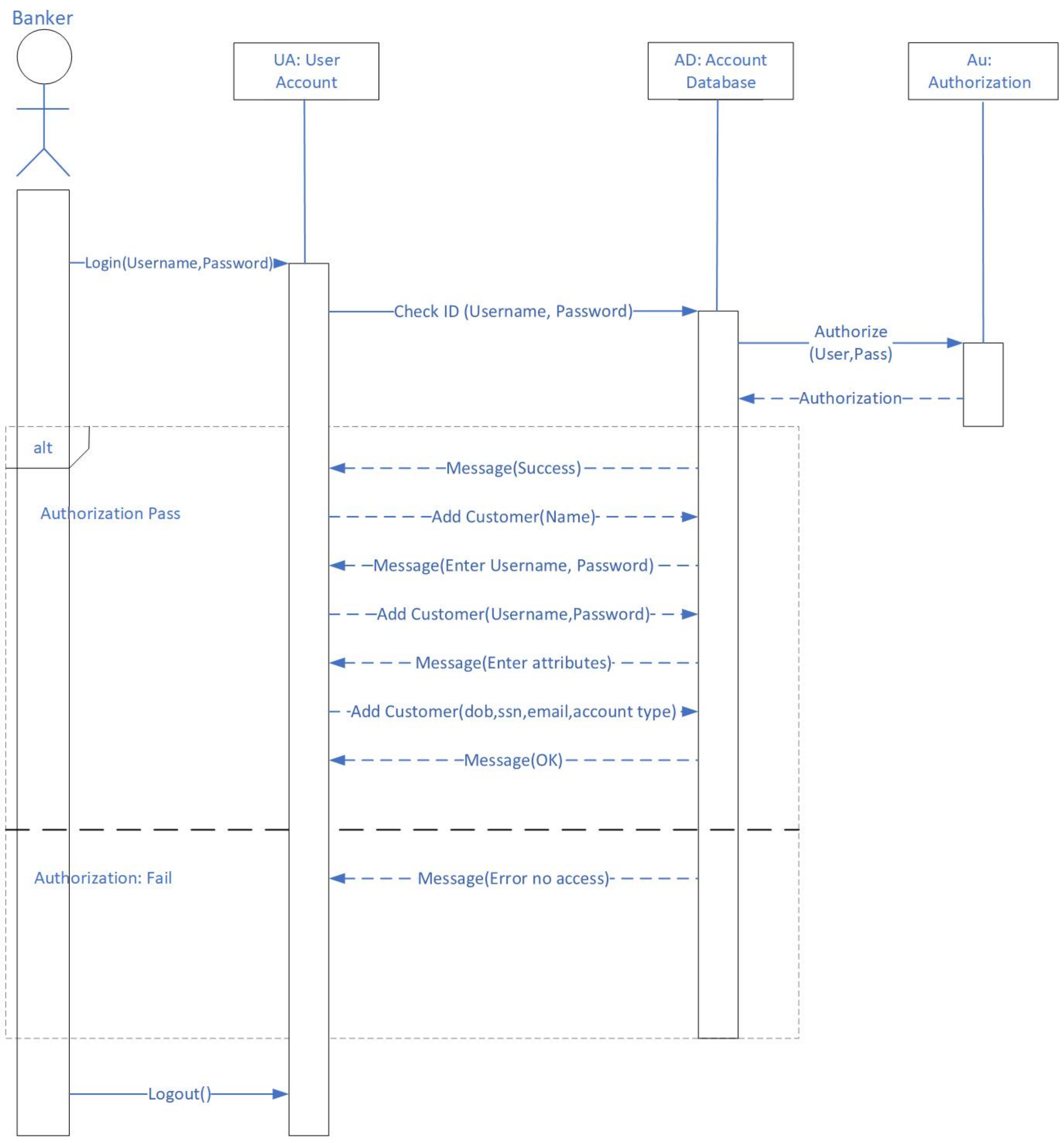
Use case 3, Withdraw cash



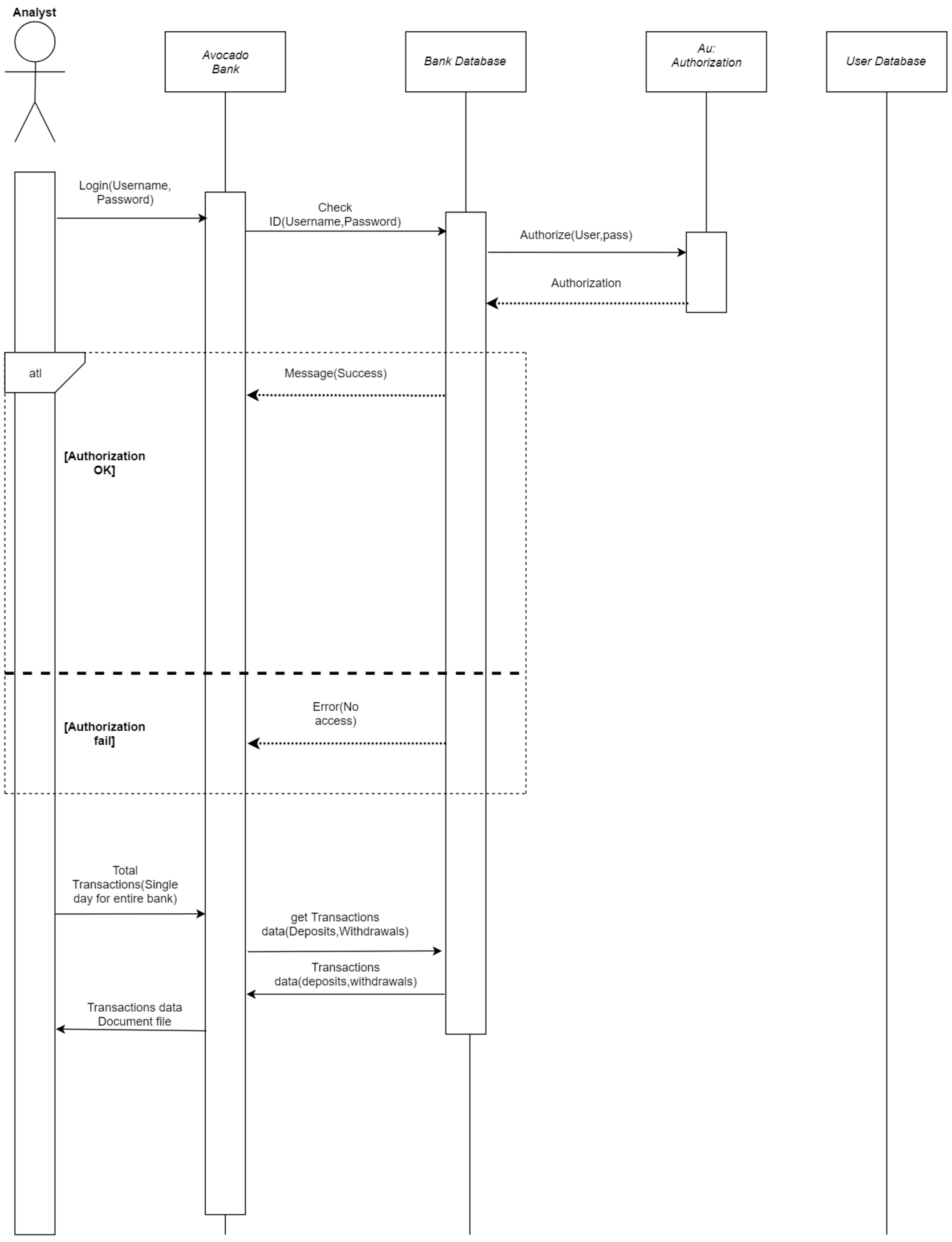
Use Case 2, Deposit cash



Use Case 1, Register customer



Use Case 6, Analyze Transaction Volume


Analyst
Avocado Bank
Bank Database
Au: Authorization
User Database
Login(Username, Password)
Check ID(Username,Password)
Authorize(User,pass)
Authorization
atl
Message(Success)
[Authorization OK]
Error(No access)
[Authorization fail]
Total Transactions(Single day for entire bank)
get Transactions data(Deposits,Withdrawals)
Transactions data(deposits,withdrawals)
Transactions data Document file

Implementation

Backend

The backend is implemented in Java 1.8, using Spring Boot. It communicates between the frontend and the database according contract described by ‘api.yaml’.

Start your server by running ‘gradle.bat run’ in ‘backend/’.

Run unit tests by running ‘gradle.bat test’ in ‘backend/’.

You can view the api documentation by starting the server and navigating to http://localhost:8000/api/

The source code for the backend can be found at https://github.com/gsu-swe-s2019-group-1/project/tree/master/backend.

Frontend

The frontend is implemented in TypeScript, with React as the framework library, and Ant Design as the component library. It provides the user with a beautiful and easy-to-use interface. Building it requires NodeJS, and running it requires a modern web browser.

Install dependencies using ‘npm install’ in ‘frontend/’.

A live-reloading development server can be started by running ‘npm start’ in ‘frontend/’. Your browser will automatically be opened. The default username & password is ‘admin’ and ‘admin’.

The code can be compiled for deployment using ‘npm run build’

The source code for the frontend can be found at https://github.com/gsu-swe-s2019-group-1/project/tree/master/frontend.

Deployment

Deployment instructions and files required for deployment can be found at https://github.com/gsu-swe-s2019-group-1/project/tree/master/deploy.

Testing

Functionality Testing

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!

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

Unit Testing Documentation

createUserTest()

Test ID	createUserTest()
Purpose of Test	To test if a user object can be created and inserted into database successfully
Test Environment	JUnit
Test Steps	Run ‘gradle.bat test’ in the ‘backend/’ directory of the git repo

Test Input	`CreateUserObject body = new CreateUserObject();`
	`body.setName("John Doe");`
	`body.setUsername("jdoe1");`
	`body.setAccountType(AccountType.CUSTOMER);`
	`body.setPassword("pass123");`
	`body.setSsn("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)

loginUserTest()

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	JUnit
Test Steps	Run 'gradle.bat test' in the 'backend/' directory of the git repo

Test Input	`LoginParameters body = new LoginParameters();`
	`body.setUserName("jdoe1");`
	`body.setPassword("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)

Bug Documentation

Bug	**The test uncovered the bug**	**Description of the bug**	**Action 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	Change the permission and privileges so user does not have

		and password	privilege to login if its invalid

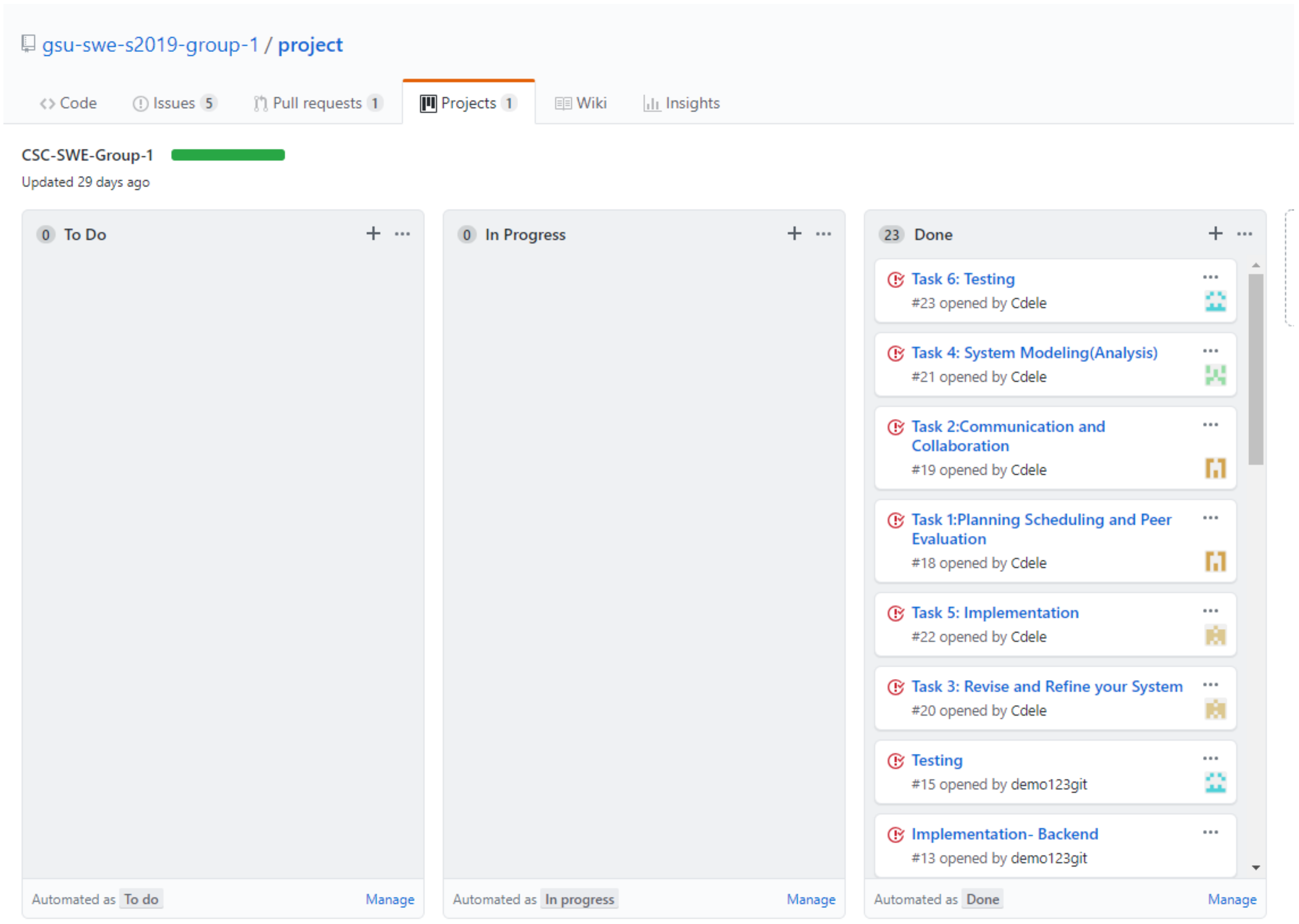
Vulnerability to SQL injections in username, password, and SSN ID field to data manipulation (inserting, deleting, changing data, etc)	loginUserTest()
	createUserTest()
		A user could bypass manipulate our data and insert, delete, or update the data in the username, password, and SSN ID field	Change permission and privilege so user does not have privilege to change data

Appendix A: Links

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

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

Appendix B: Screenshots



CSC-SWE-Group-1

Updated 29 days ago

0 To Do

Automated as To do — Manage

0 In Progress

Automated as In progress — Manage

23 Done

Task 6: Testing
#23 opened by Cdele

Task 4: System Modeling(Analysis)
#21 opened by Cdele

Task 2:Communication and Collaboration
#19 opened by Cdele

Task 1:Planning Scheduling and Peer Evaluation
#18 opened by Cdele

Task 5: Implementation
#22 opened by Cdele

Task 3: Revise and Refine your System
#20 opened by Cdele

Testing
#15 opened by demo123git

Implementation- Backend
#13 opened by demo123git

Automated as Done — Manage