Product: PiggyBank

Deliverable 2
SE 3354.008

Project Name

Group #4

Names of Team Members

Tobenna Nwosu

Hyungjin Jeong

Alejandro Alfaro

Nicky Sah

Proshun Saha

Dominic Nguyen

Alexander Montesino

Table of Contents

DELIVERABLE 1

Introduction to Document

- **Purpose of Product**

The purpose of the Online Banking Software is to provide users with a simple, efficient, and user-friendly platform to manage their banking needs. It aims to cater to a broad demographic by offering accessible and responsive design. The software will enable users to perform essential banking tasks such as transferring funds, depositing/withdrawing funds, and tracking transaction history

- **Scope of Product**

The scope of the project involves developing an online banking software application. The core functionalities will include:

1. Transferring funds between accounts.
2. Depositing and withdrawing funds.
3. Transaction history tracking.

The software will be designed to be user-friendly, responsive, and accessible.

❖ Acronyms, Abbreviations, and Definitions

API - Application Programming Interface: A set of rules and specifications that software programs can follow to communicate with each other.

GUI - Graphical User Interface: A visual way for users to interact with a computer program, using elements like windows, icons, and menus.

ACH - Automated Clearing House: A nationwide electronic funds transfer system that facilitates transactions between participating financial institutions. ACH is used for a variety of credit and debit transfers, including direct deposit of paychecks, recurring bill payments, and business-to-business payments.

KYC - Know Your Customer: A set of standards and procedures used by financial institutions and other regulated companies to verify the identity of their customers and assess potential risks. KYC processes are designed to prevent fraud, money laundering, and other illegal activities.

❖ References

1. Software Engineering Principles and Practices

- Chase, Well Fargo, Bank of America Mobile Application

2. Web Development Technologies

- React, Angular, Spring,...
- Coding languages: HTML, CSS, JavaScript.

3. Database Technologies

- MySQL

4. Security Practices

- NIST Cybersecurity Framework.

❖ Outline of the Rest of the SRS

Motivation for the project, including goals such as enhancing user-friendliness, promoting financial understanding, addressing software development challenges, ensuring security, fostering team collaboration, facilitating skill development, and preventing AI from negatively impacting customer decisions.

General Description of Banking Software

- **Context of Product -** Our product is designed with the goal to ensure customer satisfaction. While many established banks offer their own unique interfaces, some exploit customers' lack of financial knowledge, leading to unfavorable banking experiences. We uphold our core values of security, trust, and financial counseling. By striving to follow ethical guidelines, we strive to be a trusted partner in the well-being of our customers.
- **Product Function**
    - **Bill payments**

        - Efficient payment processing for expenses
    - **Transactions**
        - Deposits - customers can add funds to their available accounts
        - Withdrawals - customers can remove funds at any ATM
        - Transfer - smooth transfer of funds between accounts
    - **Multi-factor authentication**
        - One-time passcode - temporary codes sent via SMS or email
        - Password Resets - best way to recover accounts
        - Email or Phone Verification - adds more layer of security for our customers
    - **Bank Statements -** users can either receive monthly statements by mail or even go paperless via emails.
    - **Unique User Interface -** designed to promote accessibility, modernization, responsiveness, and an ease of navigation
        - Implemented to support several devices (PCs, laptops, phones, tablets, etc)
- **User Characteristics**
    - **Customers**
        - Individuals who are 18 and below will be required to have a parent or guardian as the primary account holder.
        - Customers must verify their identity with valid U.S. identification.
        - Open to any individual or business owner
    - **Employees**
        - Must be 18 years or older
        - Verified identity to work in the US.
        - **Bank Tellers**
            - No college degree is required, however we do require a high school diploma or an equivalent.
            - Responsible for handling the transactions and assistance of customers
        - **Loan Officers**
            - Preferred candidates hold a college degree in finance, business, or a related field.
            - Handles reviewing loan applications, review credit score/history, and easing the way through the loan process for customers
        - **Customer Support Representatives**
            - Must be fluent in English; bilingual in other languages is preferred
            - Excellent communication skills are required to assist customers and provide general banking support
        - **Administrators**
            - Handles cybersecurity

- **Constraints**
    - Users must have access to the internet to our services
    - Our platform is limited to the web and applications on supported computers, phones, and tablets
    - **Regulations**
        - Gramm-Leach-Bliley Act (GLBA): required to product the financial information of our users via security measures
        - PCI DSS - put requirements on companies that process and store cardholder data
        - Bank Secrecy Act (BSA) - require banks to form programs against fraudulent activities and money laundromat
    - High traffic on the platform may impact system responsiveness.
- **Assumptions and Dependencies**
    - **Assumptions**
        - Software will follow ethical and banking regulations
        - Users will have access to the internet for our services
        - Customers will compare features such as accessibility, security, and many more between banking software and others such as Chase.
    - **Dependencies**
        - Software will utilize third-party APIs for identity verification and security
        - Strong and reliable servers
        - Constant maintenance

Architectural Design

Clients: The users accessing the online banking platform through web browsers or mobile applications act as clients. They initiate requests for various banking services.

Servers: The system will include backend servers responsible for:

- **Web Server:** Hosting the user interface and handling communication with the clients
- **Application Server:** Implementing the core banking logic, such as processing fund transfers, deposits, withdrawals, and managing transaction history.
- **Database Server:** Storing and managing the banking data, including account information and transaction records.

When a user wants to transfer funds, their client application sends a request to the application server. The application server then interacts with the database server to update the account balances and may respond back to the client through the web server to confirm the transaction.

When Used:

The Client-server pattern is particularly useful in the following scenarios, all of which are relevant to our Online Banking Software:

- **Shared Data Access:** When data needs to be accessed and modified from multiple locations (by various users). Our online banking system requires numerous users to access their account information and perform transactions from different devices and locations.
- **Variable System Load:** When the load on the system is expected to fluctuate. The ability to replicate servers allows the system to handle peak usage times in online banking.
- **Centralized Functionality:** To provide common functionalities (like transaction processing, security protocols, and data management) to all clients without needing to implement them on each client device.

Advantages:

- **Distributed Architecture:** Servers can be distributed across a network, allowing for scalability and potentially better performance for geographically dispersed users.
- **Centralized Resource Management:** Common functionalities and data can be managed centrally on servers, simplifying administration and ensuring consistency. For our banking software, security protocols and the main database will be managed on the server-side.
- **Modularity and Specialization:** Different servers can be dedicated to specific tasks (e.g., web serving, application logic, database management), leading to a more organized and maintainable system. Our project reflects this with the planned separation of frontend, backend, and database responsibilities among team members.

Disadvantages:

- **Single Point of Failure:** Each server providing a specific service can be a single point of failure. If the application server or database server goes down, significant parts of the online banking system will be unavailable.
- **Network Dependency:** Performance can be unpredictable as it relies on the network connection between clients and servers. Slow or unreliable internet connections can negatively impact the user experience.
- **Security Vulnerabilities:** Servers can be susceptible to denial-of-service attacks and other security threats, requiring robust security measures. Our project proposal explicitly mentions "Security (secure money, avoid fraudulent activity)" as a key motivation,

highlighting the importance of addressing this disadvantage.

- **Management Complexity:** Managing multiple servers, especially if owned by different entities (less relevant for our project but a general concern), can be complex.

Specific Requirements

- **Functional**
  - **Bill Payments**
    - **Introduction**
      - The system should provide efficient and secure payment processing for various expenses, ensuring timely and reliable transactions. Users should be able to make payments directly from their accounts to designated recipients.
    - **Inputs**
      - User authentication (login credentials, multi-factor authentication)
      - Payment details (recipient, amount, payment method)
      - User confirmation (optional verification steps)
    - **Processing**
      - Validate user identity
      - Verify sufficient funds
      - Process the payment through a secure gateway
      - Update transaction history
    - **Outputs**
      - Payment confirmation message
      - Updated account balance
      - Transaction receipt (email or downloadable PDF)
  - **Transactions**
    - **Introduction**
      - The system should facilitate seamless financial transactions, including deposits, withdrawals, and transfers between accounts.
    - **Inputs**
      - User authentication
      - Transaction type selection (deposit, withdrawal, transfer)
      - Transaction details (amount, target account, source account)
    - **Processing**
      - Verify account details and user authorization
      - Validate sufficient funds for withdrawals and transfers
      - Execute transaction securely
      - Update account balances accordingly
    - **Outputs**
      - Updated account balances
      - Transaction confirmation (real-time update)
      - Transaction receipt (email or downloadable PDF)

- **Multi-Factor Authentication**
    - **Introduction**
        - The system should enhance security by requiring multiple forms of authentication.
    - **Inputs**
        - User login credentials
        - One-time passcode request
        - Password reset request
        - Email verification request
    - **Processing**
        - Validate primary login credentials
        - Generate and send one-time passcode
        - Verify identity through additional authentication
    - **Outputs**
        - Successful authentication
        - Account access granted or denied message
- **Bank Statements**
    - **Introduction**
        - Users should be able to access their monthly bank statements in electronic formats.
    - **Inputs**
        - User authentication
        - Statement choice
    - **Processing**
        - Retrieve transaction records for the month
        - Format the data into a readable statement
        - Generate PDF or print version
    - **Outputs**
        - Monthly bank statement via email or mail
        - Downloadable statement option

- **Non-functional**
    - **Multi-Level Security** - the software must protect sensitive data regarding our customers. This includes having secured authentication such as multi-factor authentication and biometric verification. Our software should include a role-based access control (RBAC) to ensure limited permissions for customers, customer support, bank tellers, and administrators. All of the sensitive data must be encrypted and far out of reach using standard protocols. Furthermore, the software must follow certain regulations such as the PCI-DSS. Lastly, our software should be able to detect any signs of unauthorized access and fraudulent/suspicious activities.

- **Availability** - must be available at all times with a minimal downtime. Downtime should be planned only for maintenance and have no effect on our services to our customers. It should include certain triggers and alerts in case our services are rendered unavailable in the case of maintenance, software failures, etc. These alerts must be sent to the administrators for an investigation to start as soon as possible. Our banking software should guarantee 99% uptime and recovery time.
- **Performance** - The software must be able to handle high traffic on any day of the week, especially during the holidays. The system should perform operations on transactions in less than a second to ensure a smooth experience for customers. Database queries must follow algorithms that are the most efficient and quickest to improve the user experience and response time.

Traceability Matrix for Banking Software.

1. FR vs. NFR Traceability Matrix

This matrix maps Functional Requirements (FR) to Non-Functional Requirements (NFR) to ensure that all functional aspects of the system align with performance, security, and usability constraints.

FR ID	**Functional Requirement (FR)**	**Mapped NFR ID(s)**	**Non-Functional Requirement (NFR)**
FR-01	Users can log in to their accounts.	NFR-01, NFR-03	Secure authentication, fast response time.
FR-02	Users can check their account balance.	NFR-02, NFR-04	High availability, real-time data updates.
FR-03	Users can transfer funds to another account.	NFR-01, NFR-05	Secure transaction, regulatory compliance.
FR-01	Users can view transaction history.	NFR-02, NFR-06	Data integrity, audit logging.
FR-01	Users can reset their passwords.	NFR-03, NFR-07	Multi-factor authentication, ease of use.

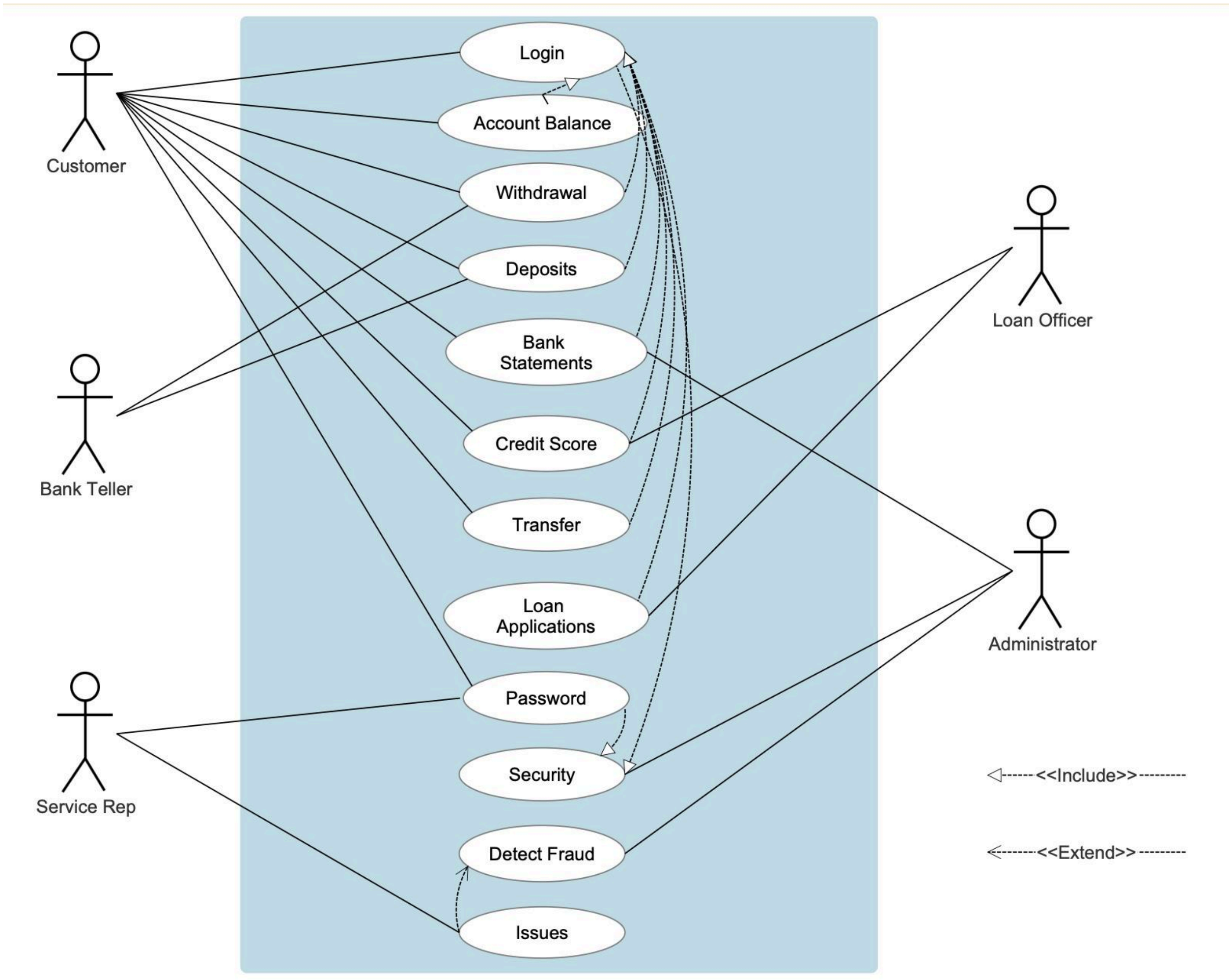
2. FR vs. Use Cases Traceability Matrix

This matrix maps Functional Requirements (FR) to Use Cases, ensuring that each FR is implemented through at least one use case.

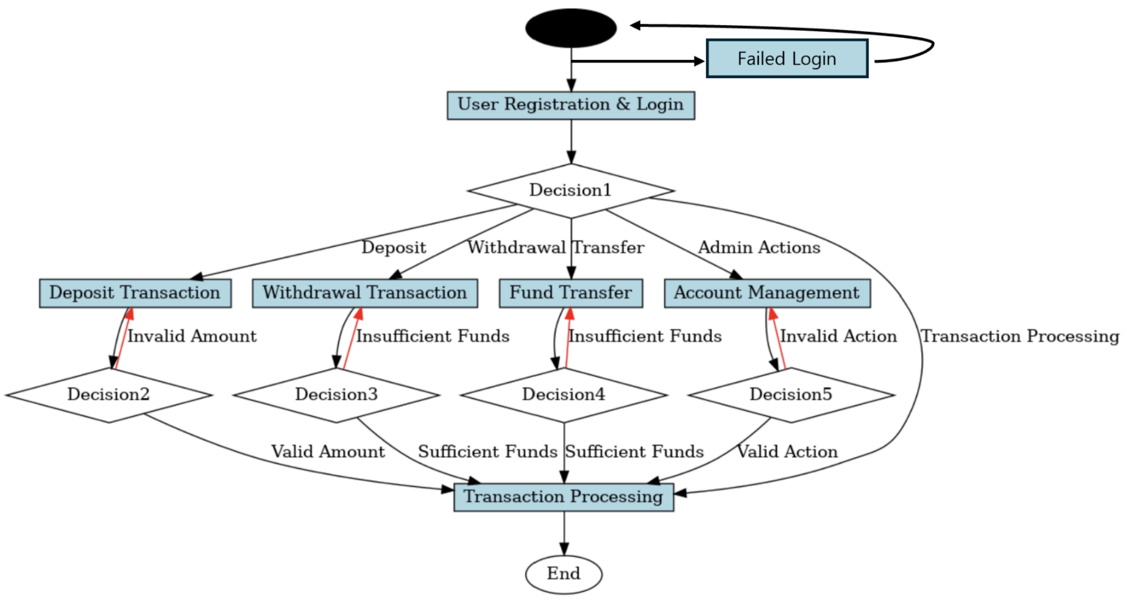
FR ID	**Functional Requirement (FR)**	**Use Case ID**	**Use Case Description**
FR-01	Users can log in to their accounts.	UC-01	User enters credentials and gains access.
FR-02	Users can check their account balance.	UC-02	User requests balance, system retrieves and displays.
FR-03	Users can transfer funds to another account.	UC-03	User enters details, system verifies and processes the transfer.
FR-04	Users can view transaction history.	UC-04	User selects history option; system fetches and displays.
FR-05	Users can reset their passwords.	UC-05	User requests reset, verifies identity, and sets a new password.

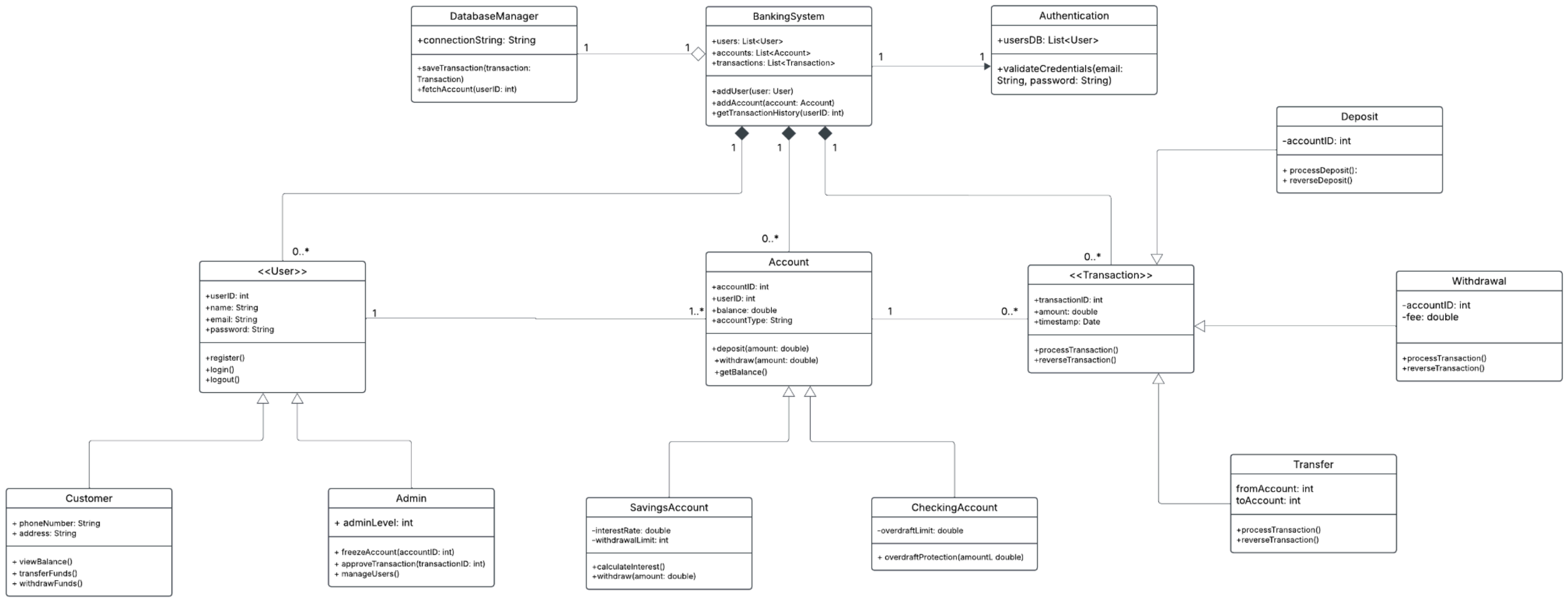
UML Diagrams

Use-case Diagram



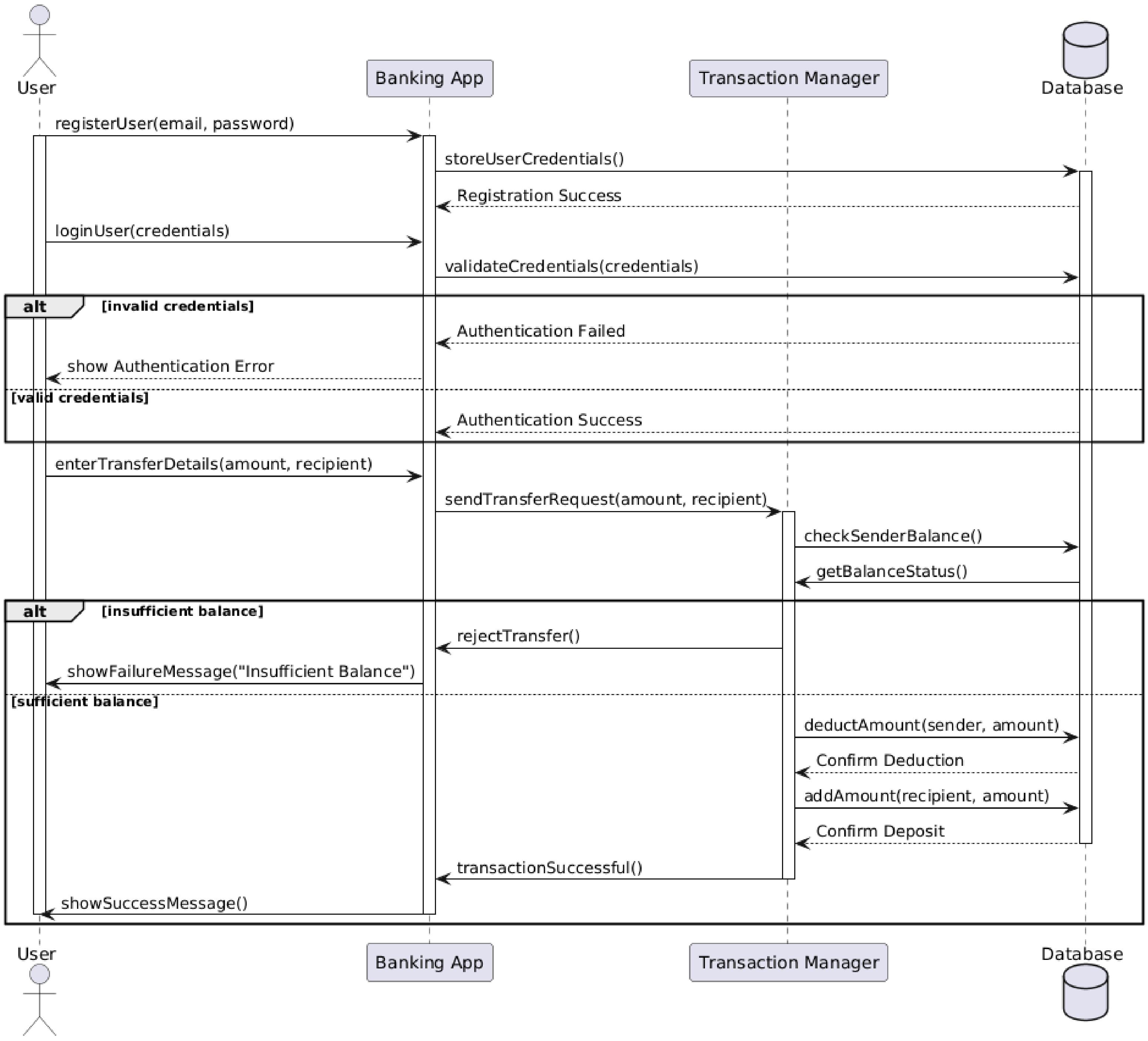
Activity Diagrams



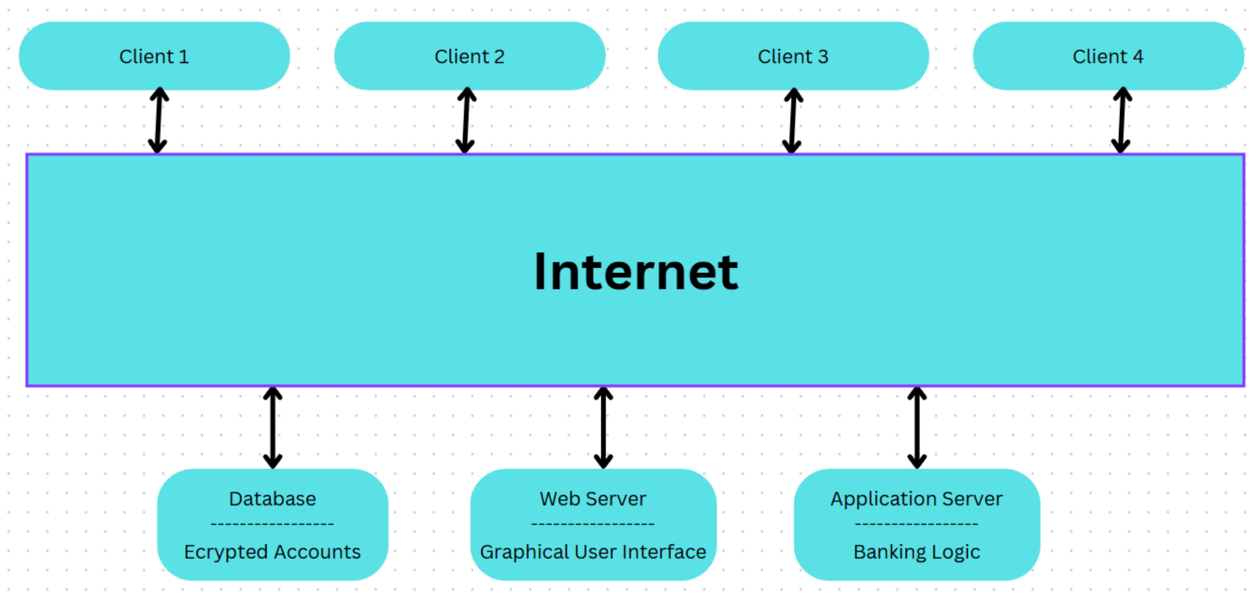


Class Diagram

Sequence Diagram



Architectural Diagram



Software Development Model: Plan-Driven

Repository:

https://github.com/The-Chicken-Nugget/PiggyBank

DELIVERABLE 2

3. Project Scheduling, Cost, Effort and Pricing Estimation, Project Duration and Staffing

3.1 Project Scheduling

3.2 Cost, Effort and Pricing Estimation (Functional Point)

3.3 Estimated cost of hardware products (Pricing Strategies)

- **Under Pricing**
  - **The project intends to enter a competitive online banking market by offering undermarket hardware prices to win contracts as well as build initial client relationships.**
- **Increased Pricing**
  - **A contingency margin serves as a protective measure to prevent financial loss since uncertainty persists regarding hardware availability particularly for security equipment including firewalls and redundant database servers.**
- **Price to Win**
  - **A contingency margin serves as a protective measure to prevent financial loss since uncertainty persists regarding hardware availability particularly for security equipment including firewalls and redundant database servers.**

Price strategy	Applied To	Reason	Example
Under Pricing	Core servers, initial database hardware	Enter online bank software market quickly	Discount for basic server setting
Increased Pricing	Secure storage, Cybersecurity hardware	Rick of cost uncertainty and fixed cost	Extra cost added to firewall equipment to cover unexpected price increasing
Price to Win	Security enhancement hardware	Requirements like changing post-contract	Higher prices additional hardware later

3.4 Estimated cost of software products (with Algorithmic Cost Modelling)

- **Function Point Method**

	Function Category		Complexity			Complexity
		Count	Simple	Average	Complex	
1	Number of user input	8	3	4	6	24
2	Number of user output	20	4	5	7	80

3	Number of user queries	6	3	4	6	18
4	Number of data files and relational tables	3	7	10	15	21
5	Number of external interfaces	1	5	7	10	5
					GFP	148

- **Processing complexity:**

PC#	Processing Complexity	PC
1	Does the system require reliable backup and recovery?	5
2	Are data communications required?	4
3	Are there distributed processing functions?	4
4	Is performance critical?	5
5	Will the system run in an existing, heavily utilized operational environment?	4
6	Does the system require online data entry?	5
7	Does the online data entry require the input transaction to be built over multiple screens or operations?	4
8	Are the master files updated online?	3
9	Are the inputs, outputs, files, or inquiries complex?	3
10	Is the internal processing complex?	3
11	Is the code designed to be reusable?	4
12	Are conversion and installation included in the design?	3
13	Is the system designed for multiple installations in different organizations?	4
14	Is the application designed to facilitate change and ease of use by the user?	5

- **Processing Complexity Adjustment (PCA):**
  - **0.65 + 0.01(PC1 +... + PC14) = 1.21**

- **Function Point (FP): GFP * PCA → 148 * 1.21 = 179.08**
- **Estimated Effort (E): FP / productivity → 179.08 / 60 = 2.985 → 3 person-week**
- **Project Duration (D): E / team size → 3 / 7 = 0.42 week (1 week)**

3.5 Estimated cost of Personnel

The personnel cost estimation for the project is based on typical market rates for software development roles at an entry to mid-level. The table below summarizes the expected labor categories, hourly rates, estimated work hours, and the corresponding cost:

Role	Hourly Rate ($/hr)	Estimated Hours	Cost ($)
Project Manager	$55	20 hours	$1,100
Backend Developer	$45	40 hours	$1,800
Frontend Developer	$45	40 hours	$1,800
Database Administrator	$50	25 hours	$1,250
Security Analyst	$50	15 hours	$750
QA Engineer (Testing)	$40	20 hours	$800
UX/UI Engineer	$45	15 hours	$675
Technical Writer	$35	10 hours	$350
Total Estimated Personal Cost			$8525

Assumptions:

- **Roles and responsibilities are distributed among the project team according to project needs.**
- **The hourly rates are estimated based on typical entry-level to mid-level software industry averages.**
- **Work hours reflect the expected effort over the full project timeline, including planning, development, testing, and documentation phases.**
- **Personnel may fulfill multiple roles depending on their skillsets and project requirements.**

The estimated personnel cost for the PiggyBank project is approximately $8,525. This includes costs for project management, software development (frontend, backend, and database), security, testing, and documentation. We assumed part-time work distributed over one academic semester, with team members playing multiple roles as necessary.

4. Software Testing

- **Testing Chart:**

White Box Testing			
#	**Description**	**Input**	**Expected Output**
1	**Test for a deposit for $50**	**An int value of 50**	**Int value of current balance + 50**
2	**Test for a withdrawal of $50 with sufficient funds**	**An int value of 50**	**Int value of current balance - 50**
3	**Test for a withdrawal of 50 with insufficient funds**	**An int value of 50**	**String as “Insufficient funds”**
4	**Test for a proper password as a string and contains the proper format (number of digits, special characters, lowercase and uppercase letters)**	**A string**	**String as “Password valid”**
5	**Test for an improper password as a string and doesn’t contains the proper format (number of digits, special characters, lowercase and uppercase letters)**	**A string**	**String as “Password invalid”**
6	**Test for a valid passcode for verification of Identity**	**An int value**	**String as “Passcode valid”**
7	**Test for an invalid passcode for verification of Identity**	**An int value**	**String as “Passcode invalid”**
8	**Test for valid login**	**An username and password as string,**	**String as “Login successful”**

		with delimiter of ‘,’	
9	**Test for invalid login**	**An username and password as string, with delimiter of ‘,’**	**String as “Login failed”**
10	**Test for transfer to an existing account with sufficient funds**	**An int value**	**Int value of recipient’s balance value + transfer** **Int value of sender’s balance value - transfer**
11	**Test for transfer to an existing account with insufficient funds**	**An int value**	**String as “Insufficient funds”**
12	**Test for transfer to an non-existing account with insufficient funds**	**An int value**	**String as “Recipient Account not found”**
13	**Test for transfer to an non-existing account with sufficient funds**	**An int value**	**String as “Recipient Account not found”**

Black Box Testing						
#	**Description**	**Input**	**Expected Output (EO)**	**Actual Output (AO)**	**Passing Criteria**	**Test Result**
1	**Test for a deposit for $50 with balance of $100**	**50**	**150**	**150**	**EO = AO**	**TBD**
2	**Test for a withdrawal of $50 with sufficient funds (balance of $100)**	**50**	**50**	**50**	**EO = AO**	**TBD**
3	**Test for a withdrawal of 50 with insufficient funds (balance of 20)**	**50**	**“Insufficient funds”**	**“Insufficient funds”**	**EO = AO**	**TBD**
4	**Test for a proper password as a string and contains the proper format (at least 2 digits, 1 special character, 1 lowercase and 1 uppercase letters)**	**“America10@”**	**“Password valid”**	**“Password valid”**	**EO = AO**	**TBD**

5	**Test for an improper password as a string and doesn’t contain the proper format (at least 2 digits, 1 special character, 1 lowercase and 1 uppercase letters)**	**“america”**	**“Password invalid”**	**“Password invalid”**	**EO = AO**	**TBD**
6	**Test for a valid passcode (int) for verification of Identity**	**21**	**“Passcode Valid”**	**“Passcode Valid”**	**EO = AO**	**TBD**
7	**Test for an invalid passcode (anything data type other than int) for verification of Identity**	**goofy**	**“Passcode invalid”**	**“Passcode invalid”**	**EO = AO**	**TBD**
8	**Test for valid login (username, password)**	**“ChrisRock10, America10@”**	**“Login successful”**	**“Login successful”**	**EO = AO**	**TBD**
9	**Test for invalid login (username, password)**	**“ChrisRock10, america”**	**“Login failed”**	**“Login failed”**	**EO = AO**	**TBD**
10	**Test for transfer to an existing account with sufficient funds (balance of $100)**	**50**	**“Transfer successful”**	**“Transfer successful”**	**EO = AO**	**TBD**
11	**Test for transfer to an existing account with insufficient funds (balance of $20)**	**50**	**“Transfer failed”**	**“Transfer failed”**	**EO = AO**	**TBD**
12	**Test for transfer to an non-existing account with insufficient funds ($20)**	**50**	**“Transfer failed”**	**“Transfer failed”**	**EO = AO**	**TBD**
13	**Test for transfer to an non-existing account with sufficient funds ($100)**	**50**	**“Transfer failed”**	**“Transfer failed”**	**EO = AO**	**TBD**

5. Similar Design Comparisons

Our project was built to offer simple, secure, and easy-to-use online banking. Compared to apps from big banks like Chase, Wells Fargo, and Bank of America, we focused more on transparency and helping users better understand their finances instead of overwhelming them with tons of features. PiggyBank also takes inspiration from newer apps such as Chime and

SoFi, with a clean and responsive design, but sticks closer to traditional banking functions with stronger security. Unlike Chime, which is mobile-only, PiggyBank is available on both web and mobile. We also combined all our features into one platform, while traditional banks often split services across multiple apps. Our focus on multi-factor authentication, role-based access control, and ethical standards gives users a safer and more straightforward banking experience.

6. Conclusion

Conclusion Statement:

Challenges and Changes in Project Management, in Software Planning etc.

During the project, we ran into a few challenges especially with project management and planning. Due to our conflicting schedules, we weren't able to meet in person until the lecture dedicated to working on Deliverable 2. Other than that, we communicated with each other virtually. We kept clear communication, divided the tasks, and made sure everything got done on time. Software planning was also a challenge, especially when it came to estimating effort, cost, and project duration because things kept changing. Overall, we all did our part, planned, and communicated to ensure the project was complete.

7. Repository Link: https://github.com/The-Chicken-Nugget/PiggyBank

8. References (MLA or APA format)

1. *PCI Security Standards Council*. “PCI DSS v4.0.” *PCI Security Standards*, 2022, https://www.pcisecuritystandards.org/pci_security/.
2. Wells Fargo. “Online Banking Features.” *Wells Fargo*, https://www.wellsfargo.com.
3. “Cybersecurity Framework.” *National Institute of Standards and Technology*, U.S. Department of Commerce, https://www.nist.gov/cyberframework.
4. Freeman, Eric, and Elisabeth Robson. *Head First Design Patterns: A Brain-Friendly Guide*. 2nd ed., O'Reilly Media, 2021.
5. Pressman, Roger S., and Bruce R. Maxim. *Software Engineering: A Practitioner’s Approach*. 8th ed., McGraw-Hill Education, 2014.
6. Elmasri, Ramez, and Shamkant B. Navathe. *Fundamentals of Database Systems*. 7th ed., Pearson, 2015.