CS3354 Software Engineering
Final Project Deliverable 2

Eruces

Jonah Getz
Daniel Jun
Yongming Mai
Jun Ren
Shivani Pattni
Jason Ta
Grace Zhou

1. Delegation of Tasks

Member	**Deliverable 2 Task**
Yongming Mai	Estimated cost of hardware products
Jun Ren	Estimated cost of personnel
Grace Zhou	A test plan for the software
Shivani Pattni	Project Scheduling
Jason Ta	Comparison of your work with similar designs
Daniel Jun	Cost, Effort and Pricing Estimation Method
Jonah Getz	Estimated cost of software products

2. Project Deliverable 1 Contents

1. Project Draft Description

What we're doing
We're doing an online banking application that will allow users to deposit and withdraw money, check financial history, and more.

Motivation and expectation of real-life use
We chose to do an online banking software because many of us use online banking software on a day to day basis to manage our finances. Banking software is efficient and secure, and its features allow users to be more in touch with their finances. It is convenient and easy to access anywhere, providing customers with continuous access to their accounts and information. The convenience of online banking software motivates us to model our own. In real life, we expect our clients to use the software to manage their assets and transactions. Users can view how much money they have in their accounts, how much money they owe, and what transactions they've made in the recent past.

Tasks

Member	Deliverable 1	Deliverable 2
Yongming Mai	Class diagram	Estimated cost of hardware products
Jun Ren	Software process model	Estimated cost of personnel
Grace Zhou	Use case diagram	A test plan for the software
Shivani Pattni	Sequence diagram	Project Scheduling
Jason Ta	Software requirements	Comparison of your work with similar designs
Daniel Jun	Address proposal feedback and task delegation	Cost, Effort and Pricing Estimation Method
Jonah Getz	Architectural Design	Estimated cost of software products

Feedback:

- In the final report, please make sure to include comparison with similar applications -if any-, make sure that you differentiate your design from those, and explicitly specify how.
- Fair delegation of tasks.

Other online banking applications are generally the same with basic features like deposit/withdrawal, transfers, etc, so what differentiates our banking app is ai chat. With this feature, users can get assistance at any time with enhanced response quality from machine learning.
Tasks were fairly delegated and agreed upon by volunteer basis.

2. GitHub repo: https://github.com/JasonT124/3354-Eruces

3. Deliverable 1 Task Delegation

Daniel Jun	Address proposal feedback and task delegation
Grace Zhou	Use case diagram
Jason Ta	Software requirements
Shivani Pattni	Sequence diagram
Jun Ren	Software process model
Yongming Mai	Class diagram
Jonah Getz	Architectural design

4: Software Process Model: **Spiral Model**

-Online banking has high complexity and sensitive data direct links to the market, each function should minimize the possibility of bugs. The core principle of the Spiral Model is to comprehensively manage risks throughout the project, which can help our team detect any flow timely and repair it with less price. The design process of the spiral model will be repeatedly evaluated and adjusted so that it can be constantly updated to adapt to the market environment and rules.

Iteration 1 - User Authentication Module:

Objective: Develop a secure user authentication module for the online banking system.

Activities:

Implement a robust login page.

Integrate multi-factor authentication for enhanced security.

Feedback:

Collect user feedback on the login process.

Address any security concerns or usability issues.

Iteration 2 - Account Overview and Dashboard:

Objective: Provide users with a comprehensive overview of their accounts and a user-friendly dashboard.

Activities:

Develop account summary functionalities.

Create an intuitive dashboard for easy navigation.

Feedback:

Gather user feedback on the accessibility and clarity of account information.

Adjust the interface based on user preferences.

Iteration 3 - Fund Transfer Module:

Objective: Enable users to initiate secure fund transfers between accounts.

Activities:

Implement fund transfer functionalities.

Integrate additional security layers for financial transactions.

Feedback:

Solicit feedback on the fund transfer process.

Address any concerns related to transaction security.

Iteration 4 - Transaction History and Statements:

Objective: Provide users with detailed transaction history and e-statements.

Activities:

Develop a transaction history log.

Enable users to generate and download e-statements.

Feedback:

Obtain feedback on the completeness and accuracy of transaction records.

Enhance features based on user suggestions.

Iteration 5 - Customer Support Integration:

Objective: Integrate customer support features for user assistance.

Activities:

Implement live chat or support ticket functionalities.

Ensure quick response times and issue resolution.

Feedback:

Obtain feedback on the effectiveness of customer support features.

Make improvements based on user experiences.

1. **Flexibility:**

The online banking software environment and customer demands can change rapidly. The flexibility of the Spiral Model allows teams to customize and adjust according to evolving requirements, better-meeting customer needs and enhancing the flexibility and quality of the software development process.

2. **Transparency:**

The Spiral Model undergoes planning, risk assessment, and other steps after each prototype, which requires constant negotiation between the requirement member and the client to ensure continuous transparency of the project's progress. This helps minimize miscommunication and misunderstanding, increasing client trust and satisfaction.

3. **Iterative development:**

Online banking software needs to adapt to market demands and new technology trends. The Spiral Model's iterative development allows for quick user feedback and adjustments in each iteration, facilitating the rapid identification and correction of issues to ensure the timely delivery of a robust software system.

4. **Quality assurance:**

The spiral model requires quality assessment at each stage to ensure project quality. Repeat the steps to analyze potential defects and supplemental features, which helps improve the reliability and stability of online banking software.

5. **Cost Control:**

Developing online banking software can incur significant costs related to security and compliance. Spiral modeling helps to manage project costs better and avoid unnecessary waste in evaluation steps. Accurate cost control ensures that the project stays within a reasonable budget.

5: Software Requirements

Functional requirements:

- The system shall be able to authenticate users and store and cache their identification for later usage.
- A user shall be able to deposit, withdraw, and transfer money through the application interface.
- A user shall be able to open and close cards.
- The system shall present user account overviews, transactions, debit cards, and account statements through the user interface.
- The system shall generate account statements, both electronic and paper, upon user request.
- The system shall log all user activity for use in later presentations.
- The system shall provide customer support on user demand, both from customer service employees and an AI chat.

Nonfunctional requirements:

- Product requirements
    - **Usability requirements**: The application shall have an easy-to-use interface and accessibility options for those with disabilities.
    - Efficiency requirements
        - **Performance requirements**: The application shall respond to user interaction within one second. Requests made to the server shall take less than five seconds from request to the display of results application-side.
        - **Space requirements**: The application shall use less than 3MB of storage per webpage. If a mobile app is developed, the app shall use less than 500MB of user storage on-device.
    - **Dependability requirements**: The application shall have a minimum of five nines of uptime (99.999%).
    - **Security requirements**: The application shall encrypt user data and communicate with the bank using secure communication protocols.
- Organizational requirements
    - **Environmental requirements**: The application shall make efficient queries to the bank's servers in order to reduce server usage and electricity consumption.
    - **Operational requirements**: The application shall be patched within one day of a discovered vulnerability. The underlying system shall be monitored by system engineers.
    - **Development requirements**: The application shall be developed using up-to-date IDEs and linting tools to catch common errors.
- External requirements
    - **Regulatory requirements**: The application shall put forth measures to comply with the Financial Modernization Act of 1999 to protect user privacy.[1]

    - **Ethical requirements**: The application shall notify users of potential data breaches and shall not store unneeded user data for sale to advertising companies.
    - Legislative requirements
        - **Accounting requirements**: The application shall not charge exorbitant fees for user requests and actions.[2]
        - **Safety/security requirements**: The application shall implement features to comply with FDIC laws and regulations.[1]

[1]: We are assuming the application is going to be used primarily in the United States.
[2]: We are assuming that the US has laws in place to prevent large fees in banking applications.

6: Use Case Diagram

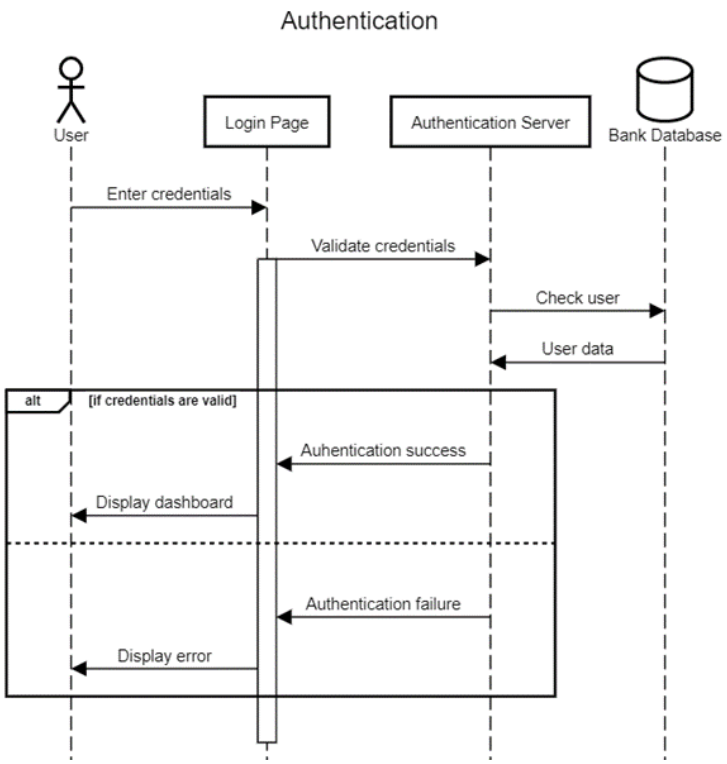
7: Sequence Diagrams

Created using https://sequencediagram.org/

---

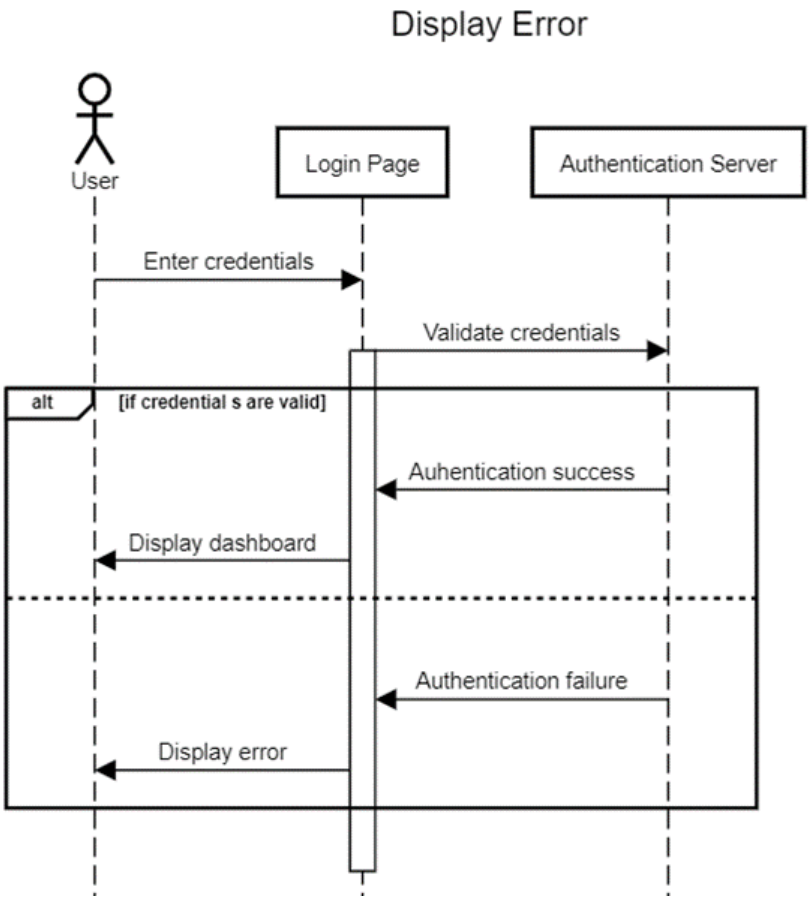
title **Authentication**

actor User
User->Login Page: Enter credentials
Login Page->Authentication Server: Validate credentials
activate Login Page
database Bank Database
Authentication Server->Bank Database: Check user
Bank Database->Authentication Server: User data
alt if credentials are valid
Authentication Server->Login Page: Authentication success
Login Page->User: Display dashboard
else
Authentication Server->Login Page: Authentication failure
Login Page->User: Display error
end



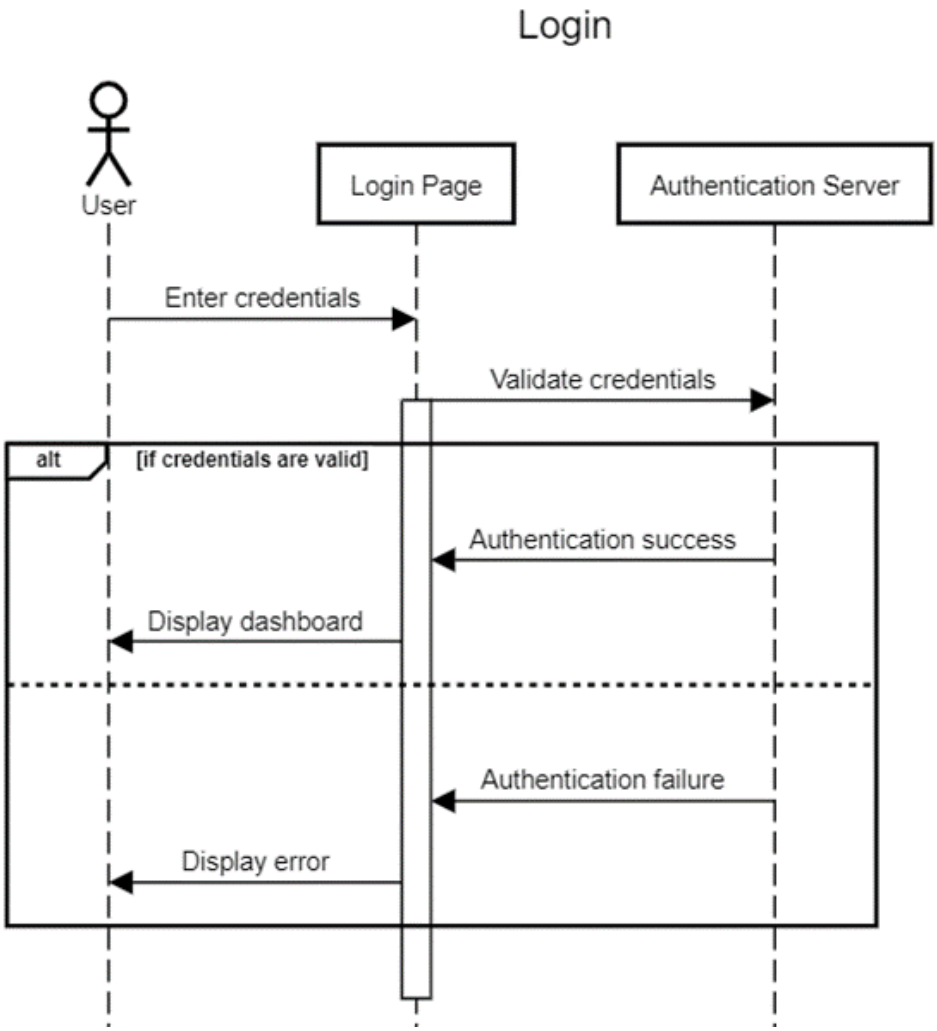
title **Display Error**

actor User
User->Login Page: Enter credentials
Login Page->Authentication Server: Validate credentials
activate Login Page
database Bank Database
alt if credentials are valid
Authentication Server->Login Page: Authentication success
Login Page->User: Display dashboard
else
Authentication Server->Login Page: Authentication failure
Login Page->User: Display error
end



title **Login**

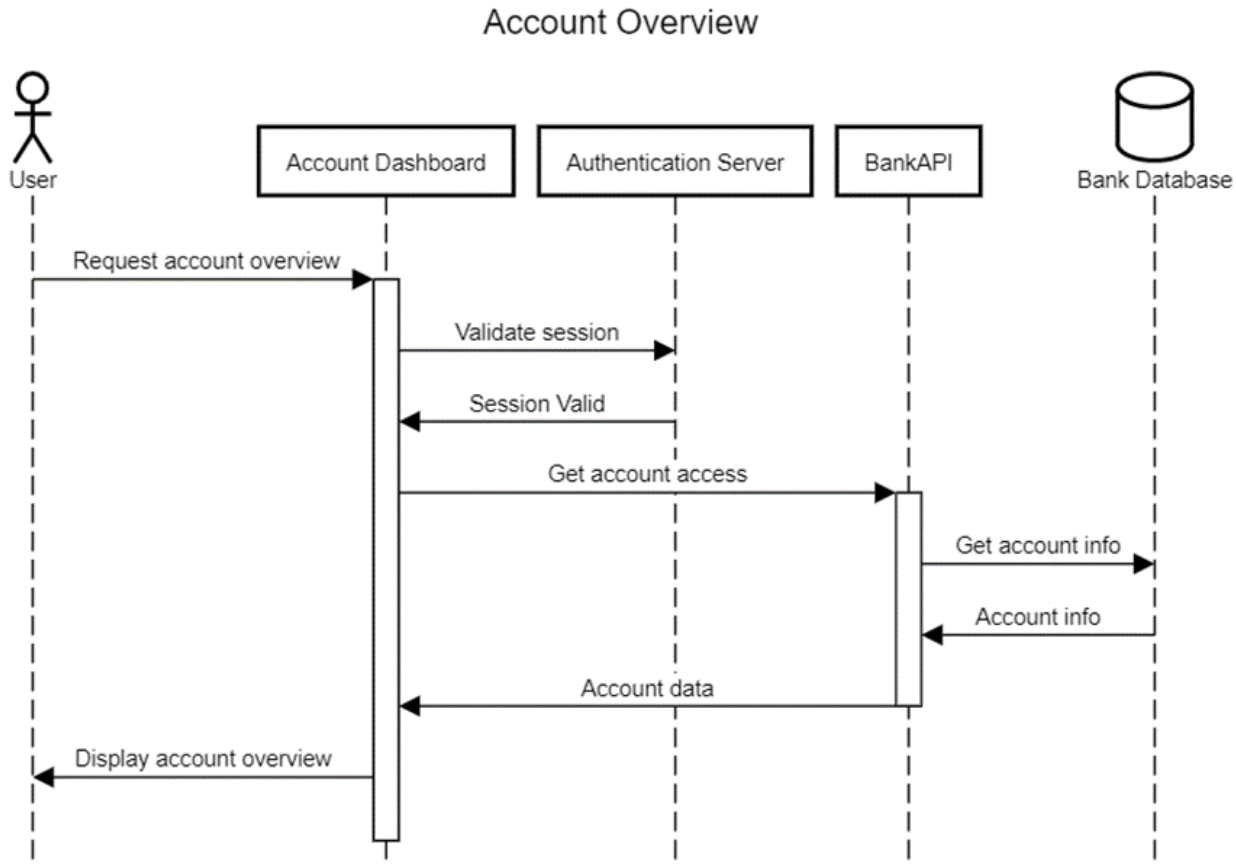
actor User
User->Login Page: Enter credentials
Login Page->Authentication Server: Validate credentials
activate Login Page
database Bank Database
alt if credentials are valid
Authentication Server->Login Page: Authentication success
Login Page->User: Display dashboard
else
Authentication Server->Login Page: Authentication failure
Login Page->User: Display error
end



---

title **Account Overview**

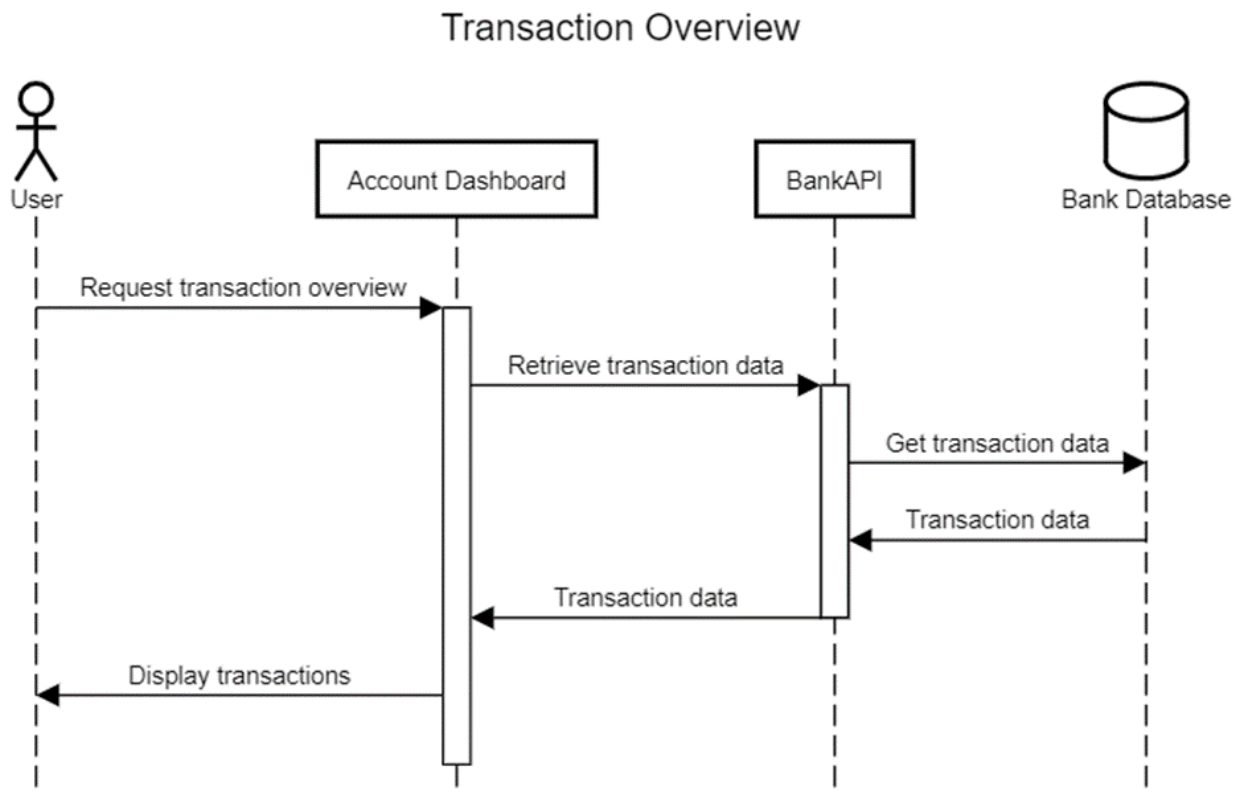
actor User
User->Account Dashboard: Request account overview
activate Account Dashboard
Account Dashboard->Authentication Server: Validate session
Authentication Server->Account Dashboard: Session Valid
Account Dashboard->BankAPI: Get account access
activate BankAPI
database Bank Database
BankAPI->Bank Database: Get account info
Bank Database->BankAPI: Account info
BankAPI->Account Dashboard: Account data
deactivate BankAPI
Account Dashboard->User: Display account overview



---

title **Transaction Overview**

actor User
User->Account Dashboard: Request transaction overview
activate Account Dashboard
Account Dashboard->BankAPI: Retrieve transaction data
activate BankAPI
database Bank Database
BankAPI->Bank Database: Get transaction data
Bank Database->BankAPI: Transaction data
BankAPI->Account Dashboard: Transaction data
deactivate BankAPI
Account Dashboard->User: Display transactions

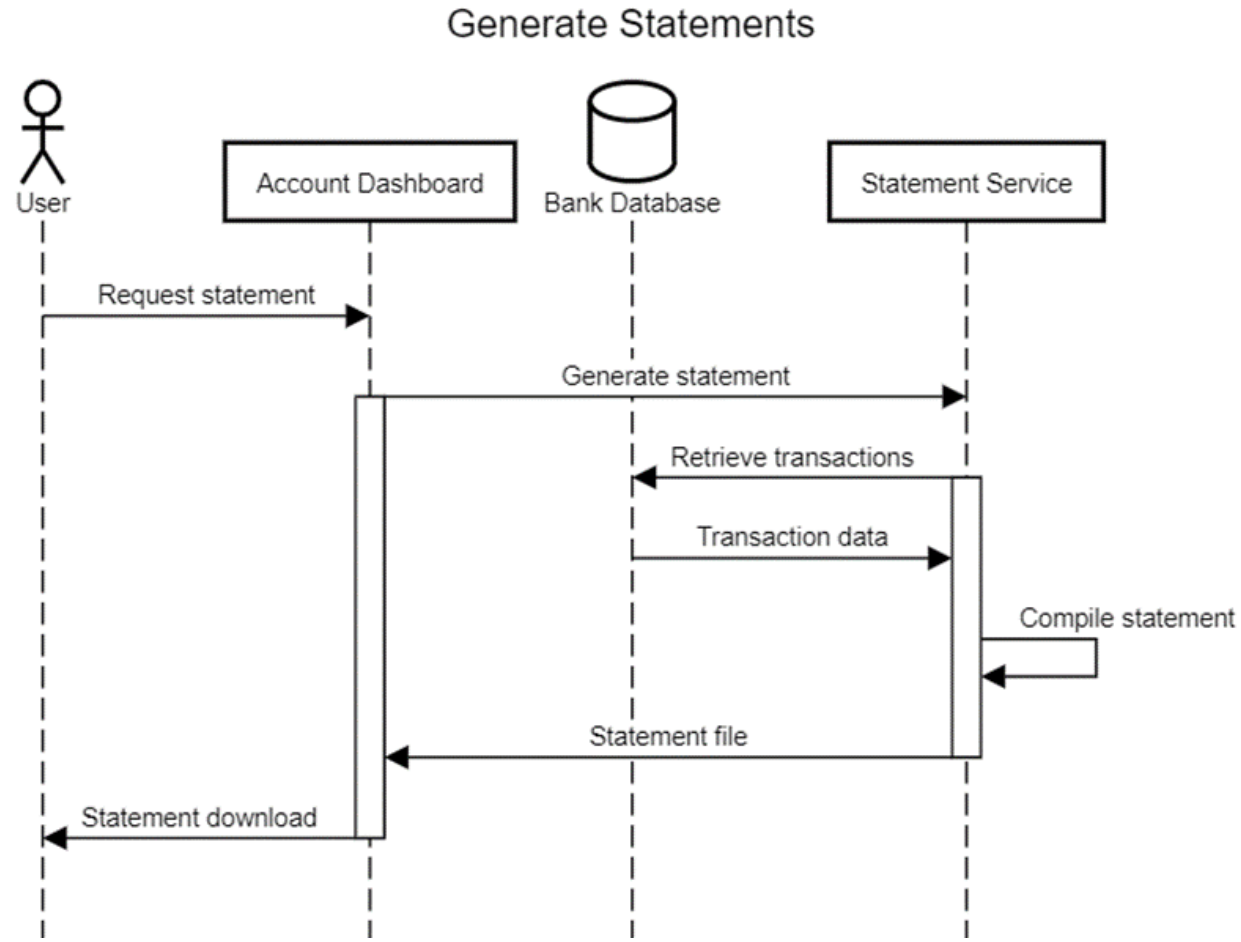


---

title **Generate Statements**

actor User
User->Account Dashboard: Request statement
database Bank Database
Account Dashboard->Statement Service: Generate statement
activate Account Dashboard
Statement Service->Bank Database: Retrieve transactions
activate Statement Service
Bank Database->Statement Service: Transaction data

Statement Service->Statement Service: Compile statement
Statement Service->Account Dashboard: Statement file
deactivate Statement Service
Account Dashboard->User: Statement download
deactivate Account Dashboard

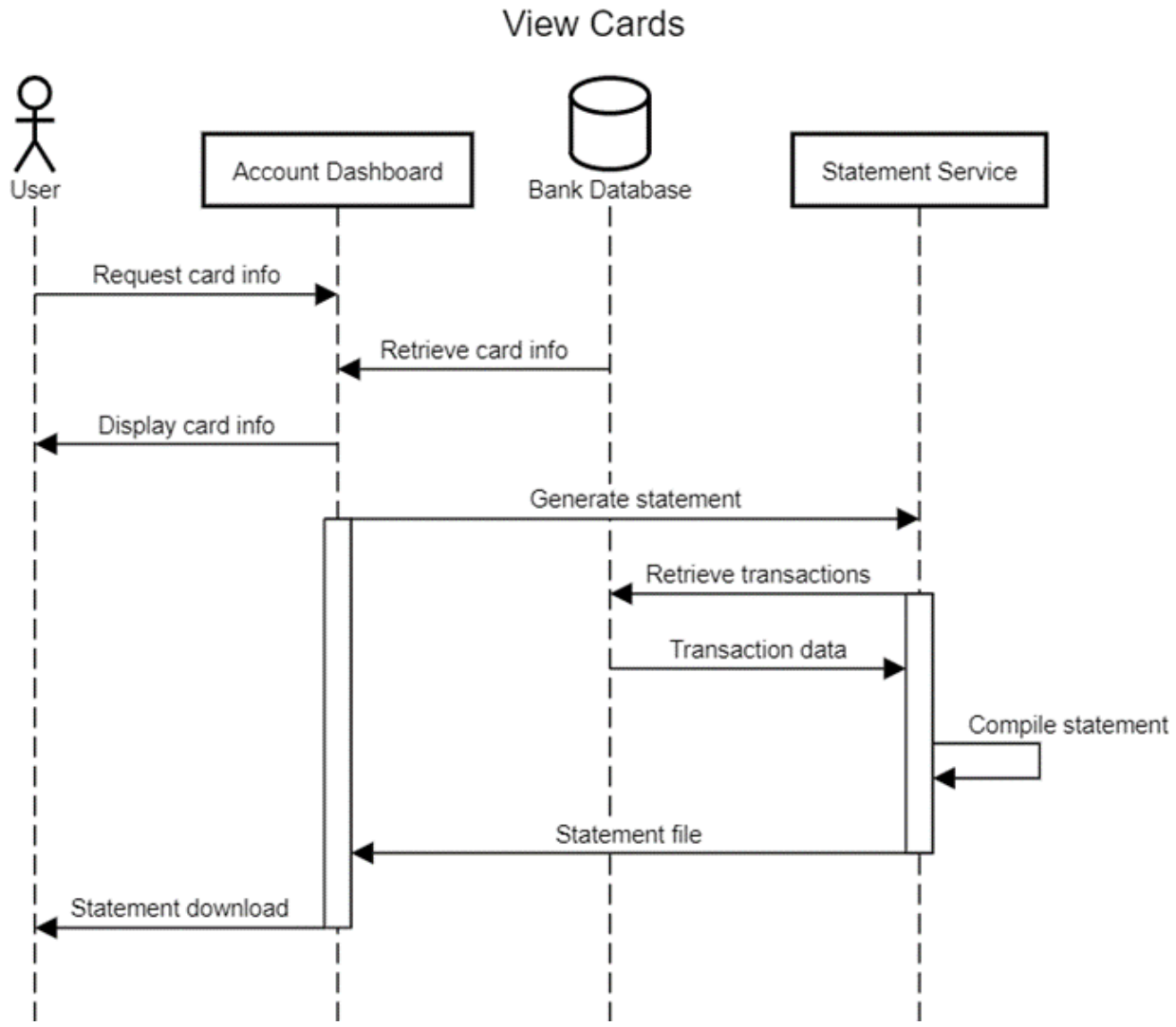


---

title **View Cards**

actor User
User->Account Dashboard: Request card info
database Bank Database
Bank Database->Account Dashboard: Retrieve card info
Account Dashboard->User: Display card info
Account Dashboard->Statement Service: Generate statement
activate Account Dashboard
Statement Service->Bank Database: Retrieve transactions
activate Statement Service
Bank Database->Statement Service: Transaction data
Statement Service->Statement Service: Compile statement

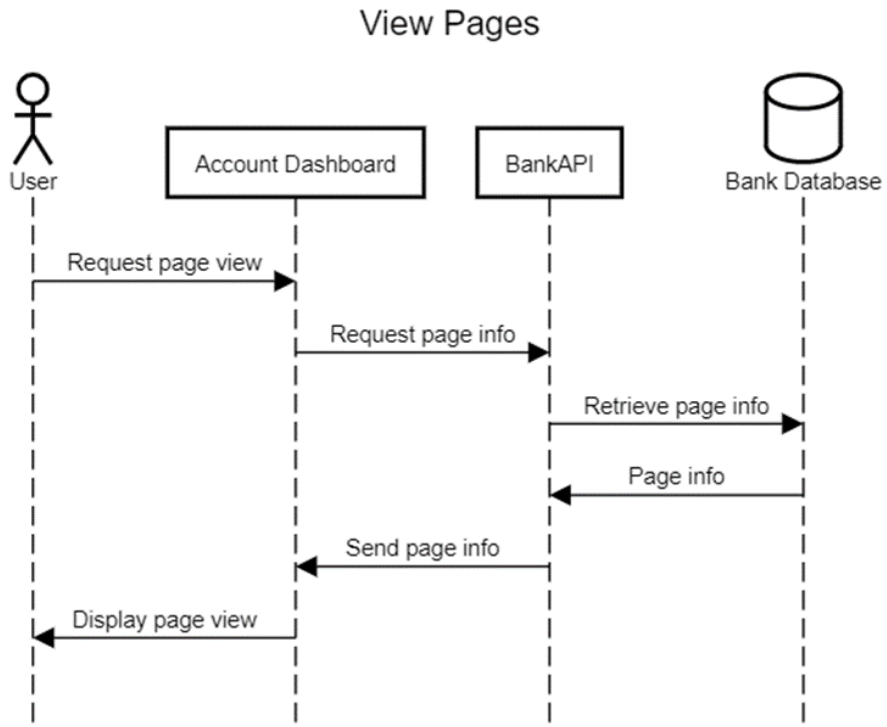
Statement Service->Account Dashboard: Statement file
deactivate Statement Service
Account Dashboard->User: Statement download
deactivate Account Dashboard



---

title **View Pages**

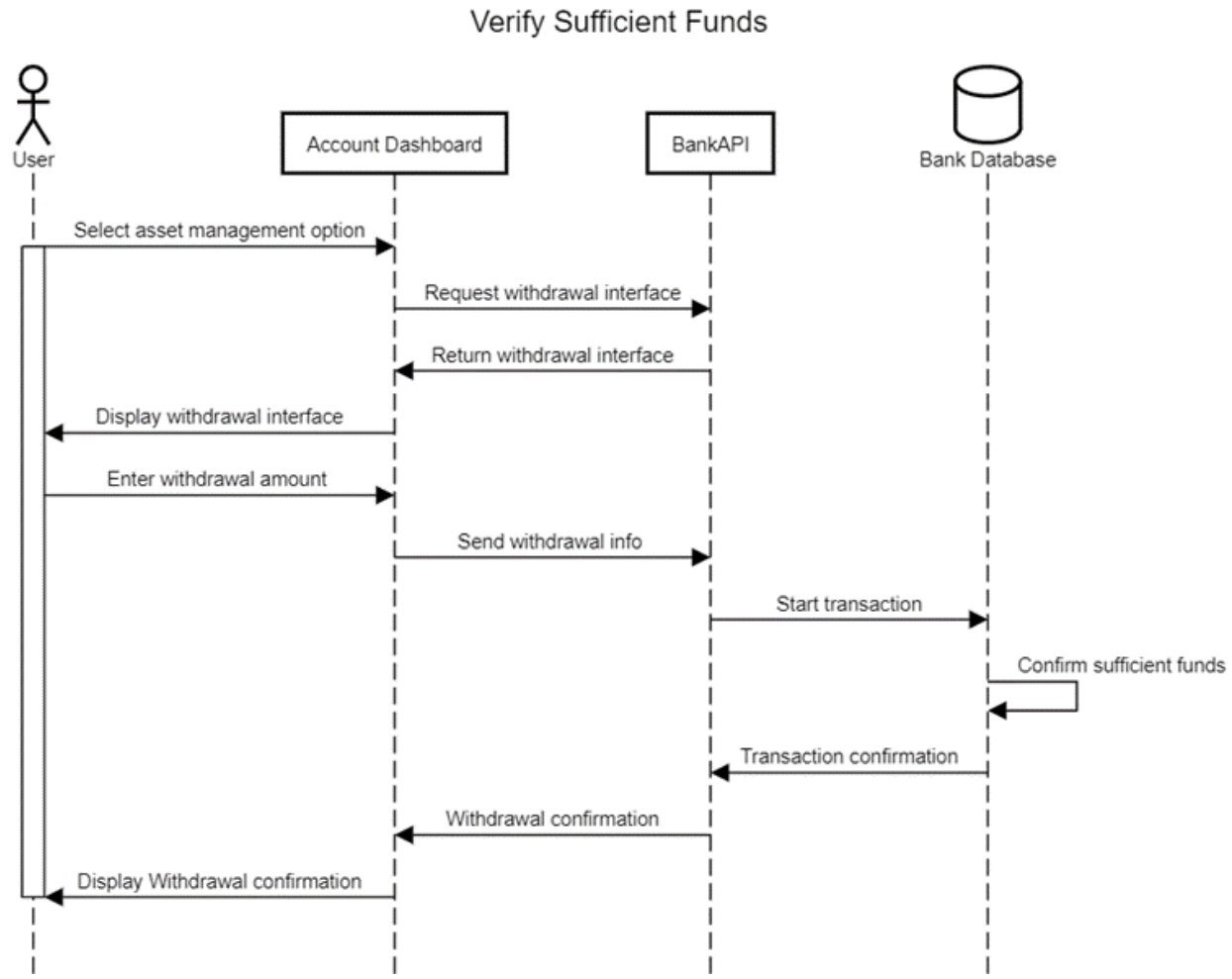
actor User
User->Account Dashboard: Request page view
Account Dashboard->BankAPI: Request page info
database Bank Database
BankAPI->Bank Database: Retrieve page info
Bank Database->BankAPI: Page info
BankAPI->Account Dashboard: Send page info
Account Dashboard->User: Display page view



---

title **Verify Sufficient Funds**

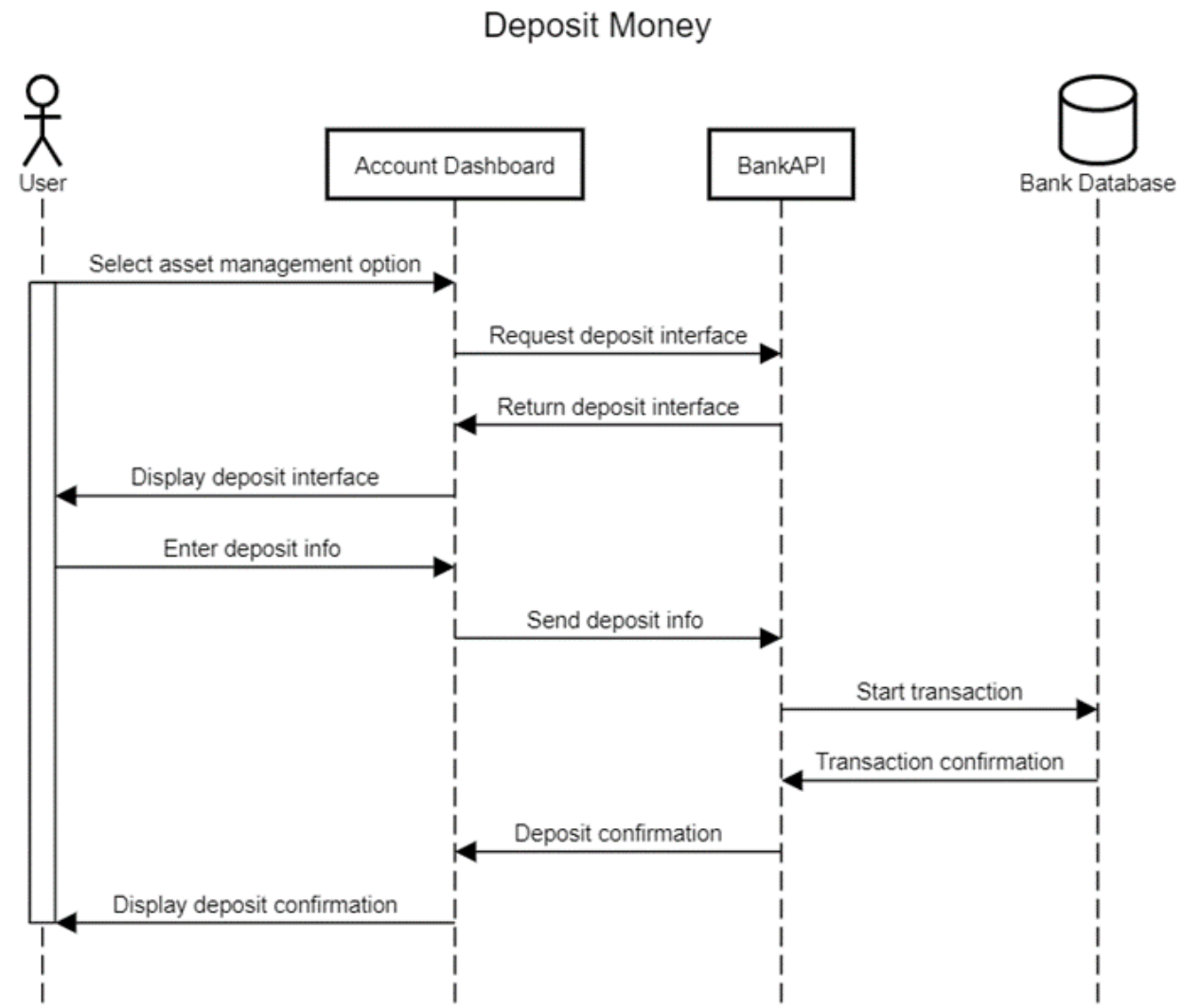
actor User
User->Account Dashboard: Select asset management option
activate User
Account Dashboard->BankAPI: Request withdrawal interface
BankAPI->Account Dashboard: Return withdrawal interface
Account Dashboard->User: Display withdrawal interface
User->Account Dashboard: Enter withdrawal amount
Account Dashboard->BankAPI: Send withdrawal info
database Bank Database
BankAPI->Bank Database: Start transaction
Bank Database->Bank Database: Confirm sufficient funds
Bank Database->BankAPI: Transaction confirmation
BankAPI->Account Dashboard: Withdrawal confirmation
Account Dashboard->User: Display Withdrawal confirmation
deactivate User



---

title **Deposit Money**

actor User
User->Account Dashboard: Select asset management option
activate User
Account Dashboard->BankAPI: Request deposit interface
BankAPI->Account Dashboard: Return deposit interface
Account Dashboard->User: Display deposit interface
User->Account Dashboard: Enter deposit info
Account Dashboard->BankAPI: Send deposit info
database Bank Database
BankAPI->Bank Database: Start transaction
Bank Database->BankAPI: Transaction confirmation
BankAPI->Account Dashboard: Deposit confirmation
Account Dashboard->User: Display deposit confirmation
deactivate User

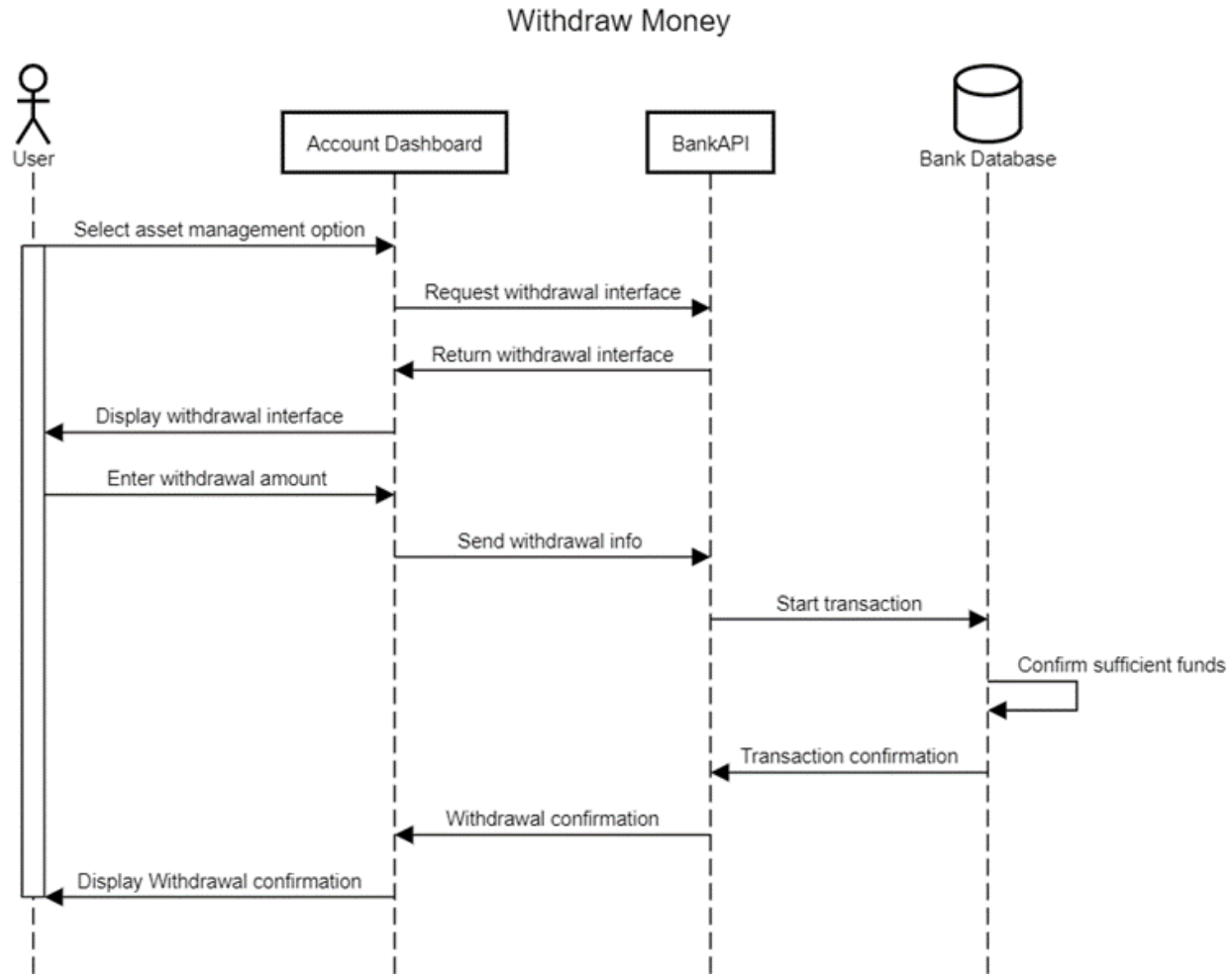


---

title **Withdraw Money**

actor User
User->Account Dashboard: Select asset management option
activate User
Account Dashboard->BankAPI: Request withdrawal interface
BankAPI->Account Dashboard: Return withdrawal interface
Account Dashboard->User: Display withdrawal interface
User->Account Dashboard: Enter withdrawal amount
Account Dashboard->BankAPI: Send withdrawal info
database Bank Database
BankAPI->Bank Database: Start transaction
Bank Database->Bank Database: Confirm sufficient funds
Bank Database->BankAPI: Transaction confirmation

BankAPI->Account Dashboard: Withdrawal confirmation
Account Dashboard->User: Display Withdrawal confirmation
deactivate User

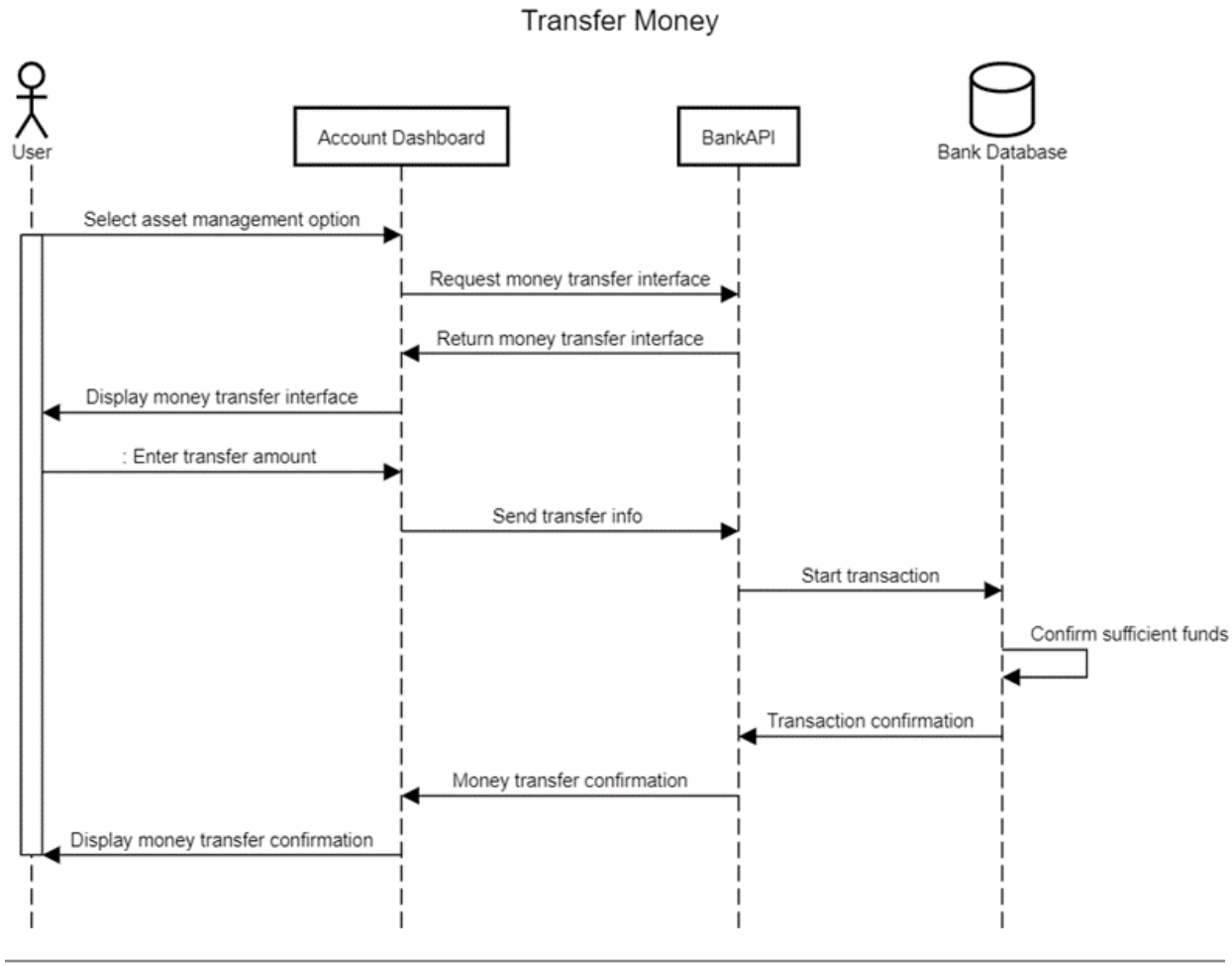


---

title **Transfer Money**

actor User
User->Account Dashboard: Select asset management option
activate User
Account Dashboard->BankAPI: Request money transfer interface
BankAPI->Account Dashboard: Return money transfer interface
Account Dashboard->User: Display money transfer interface
User->Account Dashboard:: Enter transfer amount
Account Dashboard->BankAPI: Send transfer info
database Bank Database
BankAPI->Bank Database: Start transaction

Bank Database->Bank Database: Confirm sufficient funds
Bank Database->BankAPI: Transaction confirmation
BankAPI->Account Dashboard: Money transfer confirmation
Account Dashboard->User: Display money transfer confirmation
deactivate User

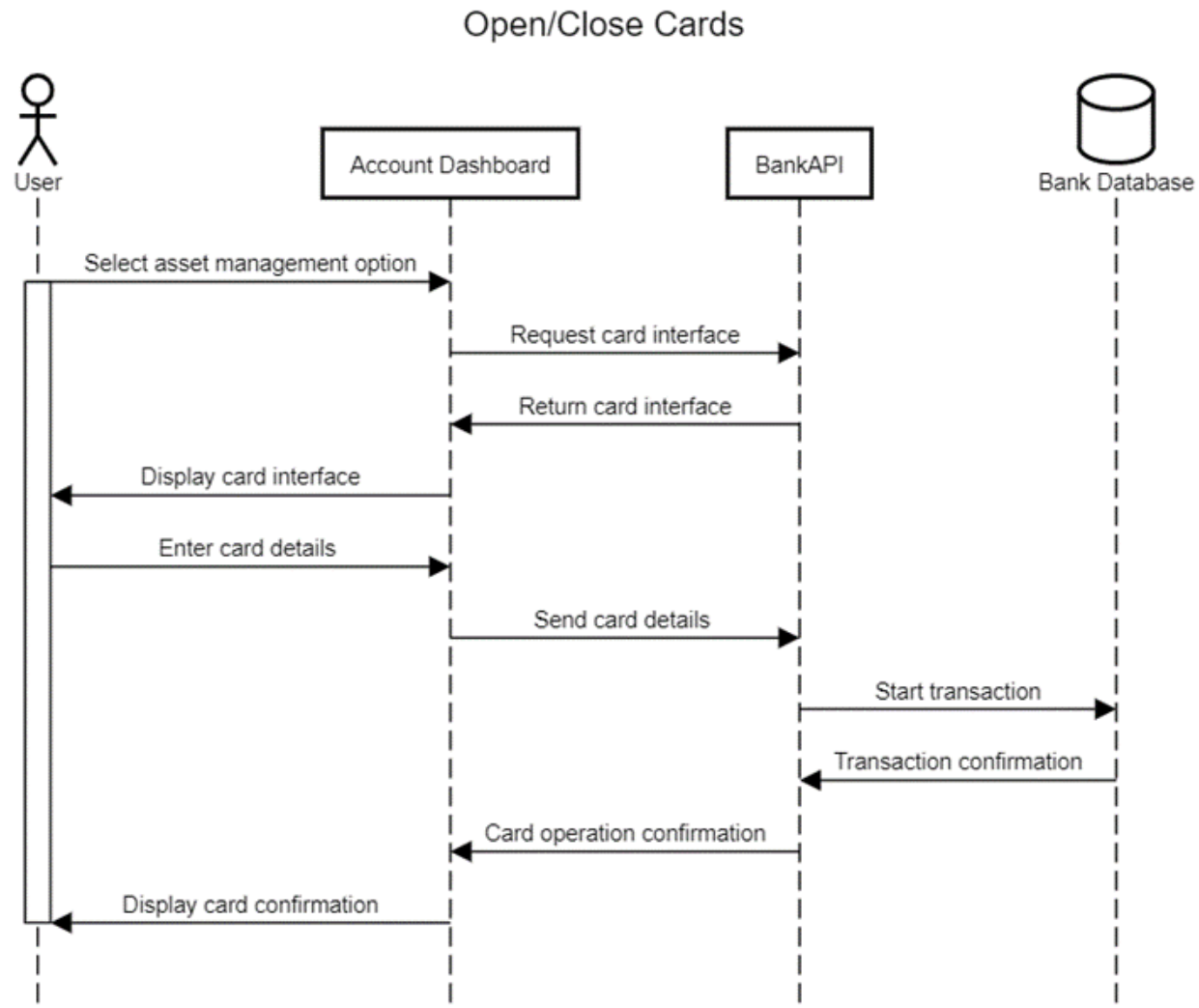


---

title **Open/Close Cards**

actor User
User->Account Dashboard: Select asset management option
activate User
Account Dashboard->BankAPI: Request card interface
BankAPI->Account Dashboard: Return card interface
Account Dashboard->User: Display card interface
User->Account Dashboard: Enter card details
Account Dashboard->BankAPI: Send card details
database Bank Database
BankAPI->Bank Database: Start transaction
Bank Database->BankAPI: Transaction confirmation

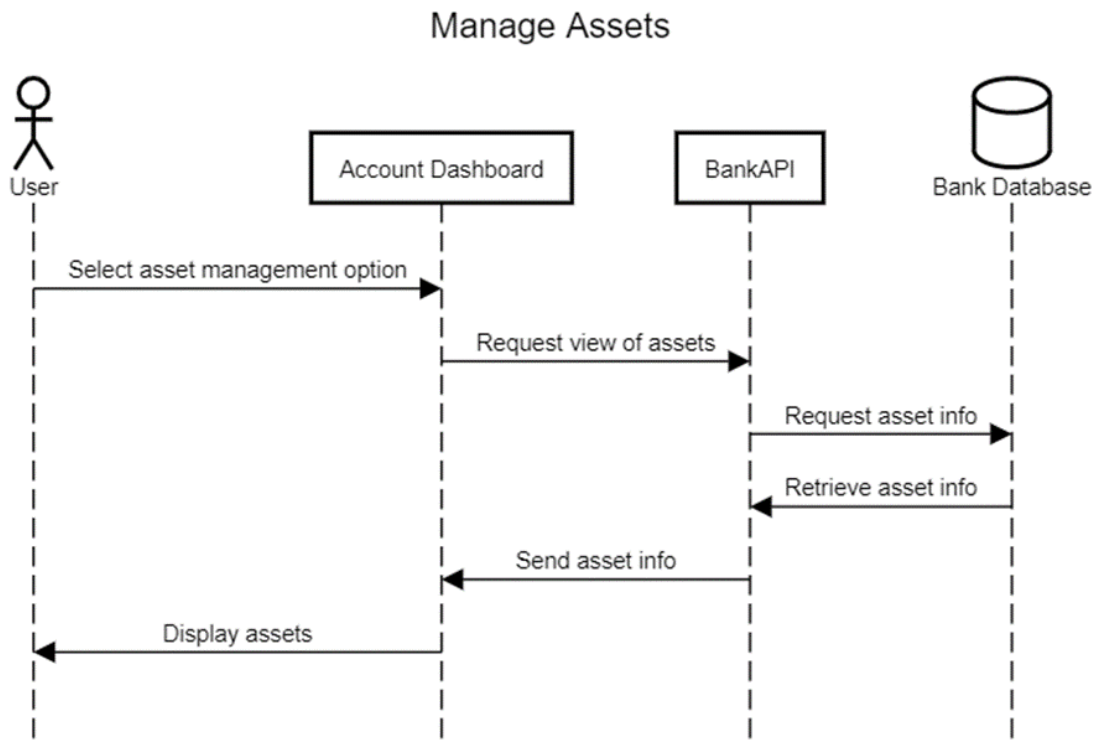
BankAPI->Account Dashboard: Card operation confirmation
Account Dashboard->User: Display card confirmation
deactivate User



---

title **Manage Assets**

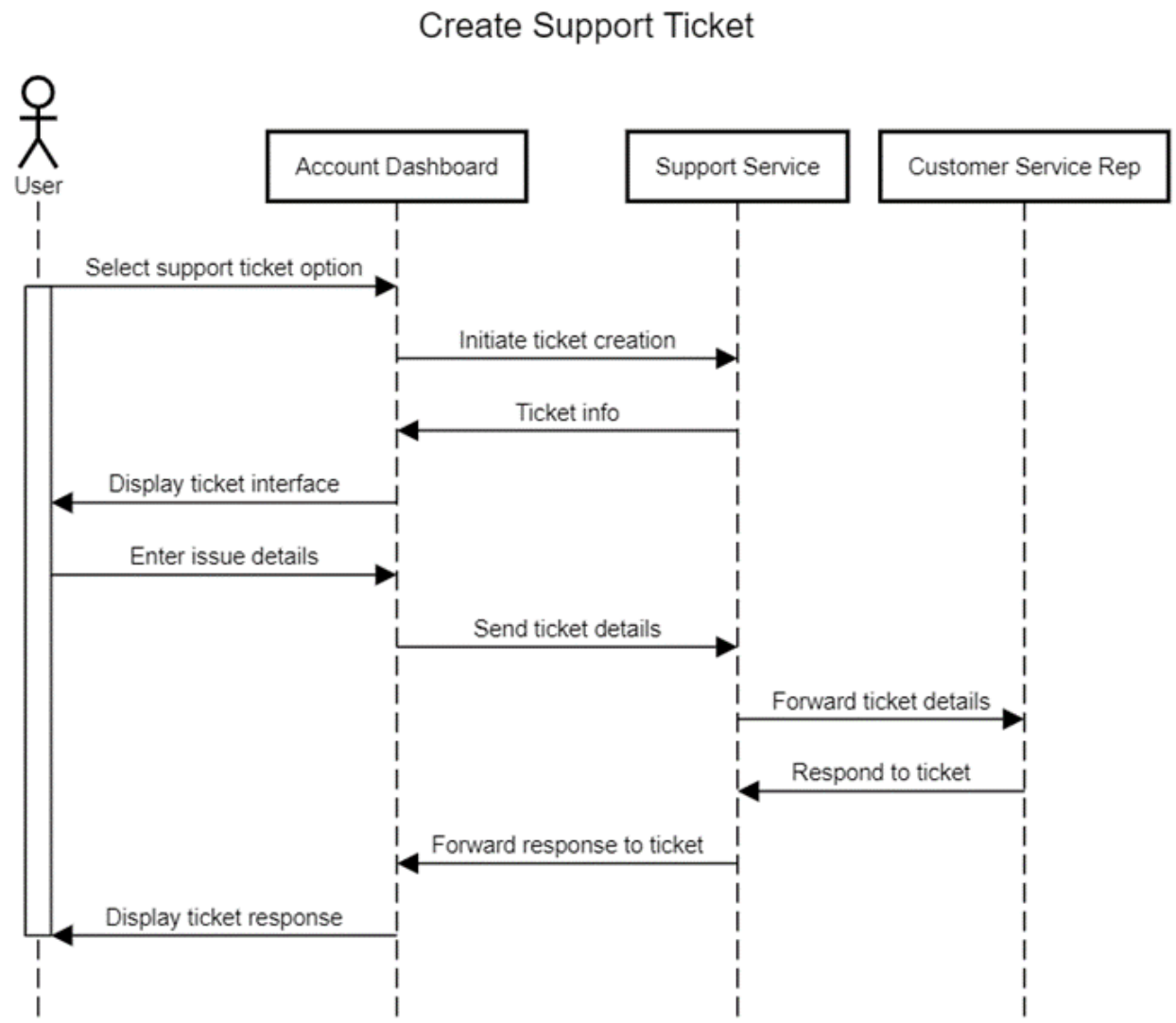
actor User
User->Account Dashboard: Select asset management option
Account Dashboard->BankAPI: Request view of assets
database Bank Database
BankAPI->Bank Database: Request asset info
Bank Database->BankAPI: Retrieve asset info
BankAPI->Account Dashboard: Send asset info
Account Dashboard->User: Display assets



---

title **Create Support Ticket**

actor User
User->Account Dashboard: Select support ticket option
activate User
Account Dashboard->Support Service: Initiate ticket creation
Support Service->Account Dashboard: Ticket info
Account Dashboard->User: Display ticket interface
User->Account Dashboard: Enter issue details
Account Dashboard->Support Service: Send ticket details
Support Service->Customer Service Rep: Forward ticket details
Customer Service Rep->Support Service: Respond to ticket
Support Service->Account Dashboard: Forward response to ticket
Account Dashboard->User: Display ticket response
deactivate User

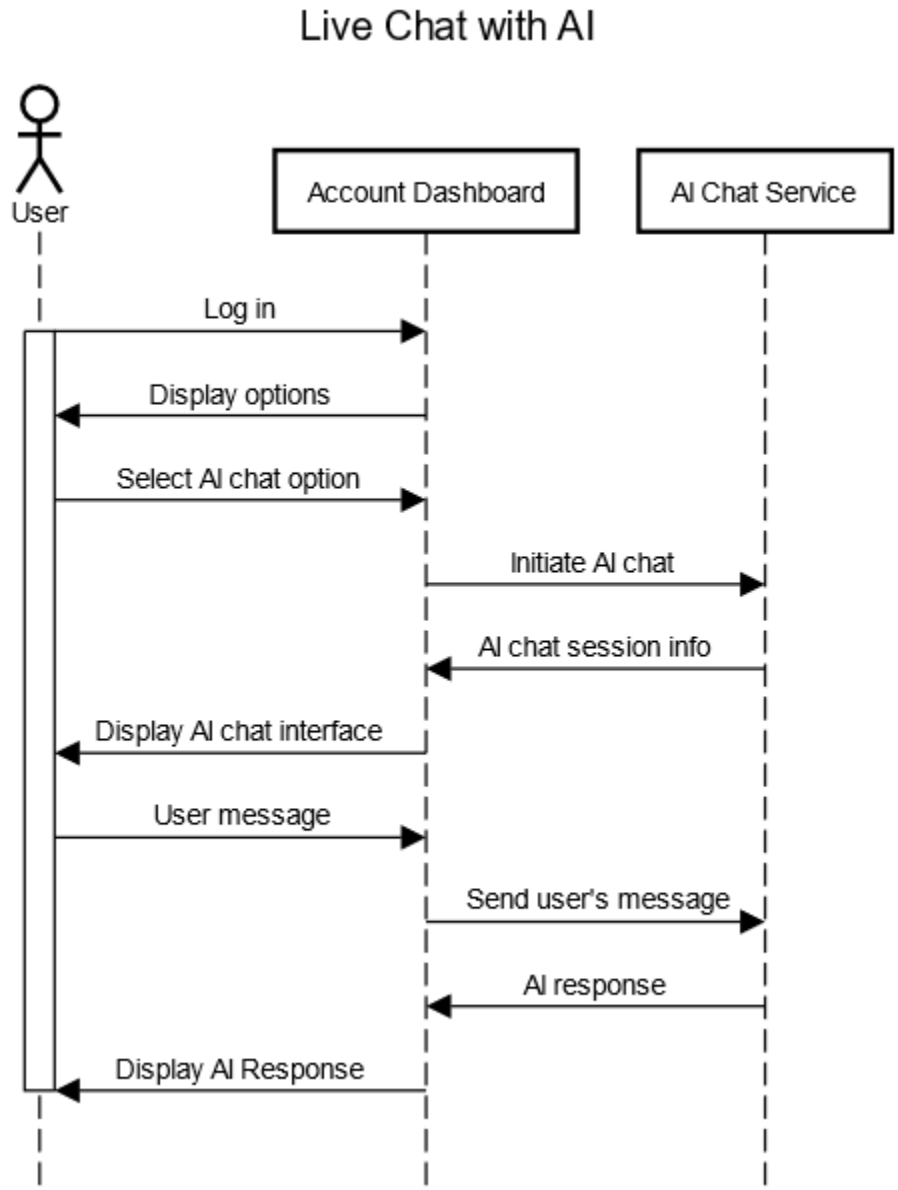


---

title **Live Chat with AI**

actor User
User->Account Dashboard: Log in
activate User
Account Dashboard->User: Display options
else if AI chat option is chosen
User->Account Dashboard: Select AI chat option
Account Dashboard->AI Chat Service: Initiate AI chat
AI Chat Service->Account Dashboard: AI chat session info
Account Dashboard->User: Display AI chat interface
User->Account Dashboard: User message
Account Dashboard->AI Chat Service: Send user's message
AI Chat Service->Account Dashboard: AI response
Account Dashboard->User: Display AI Response

deactivate User

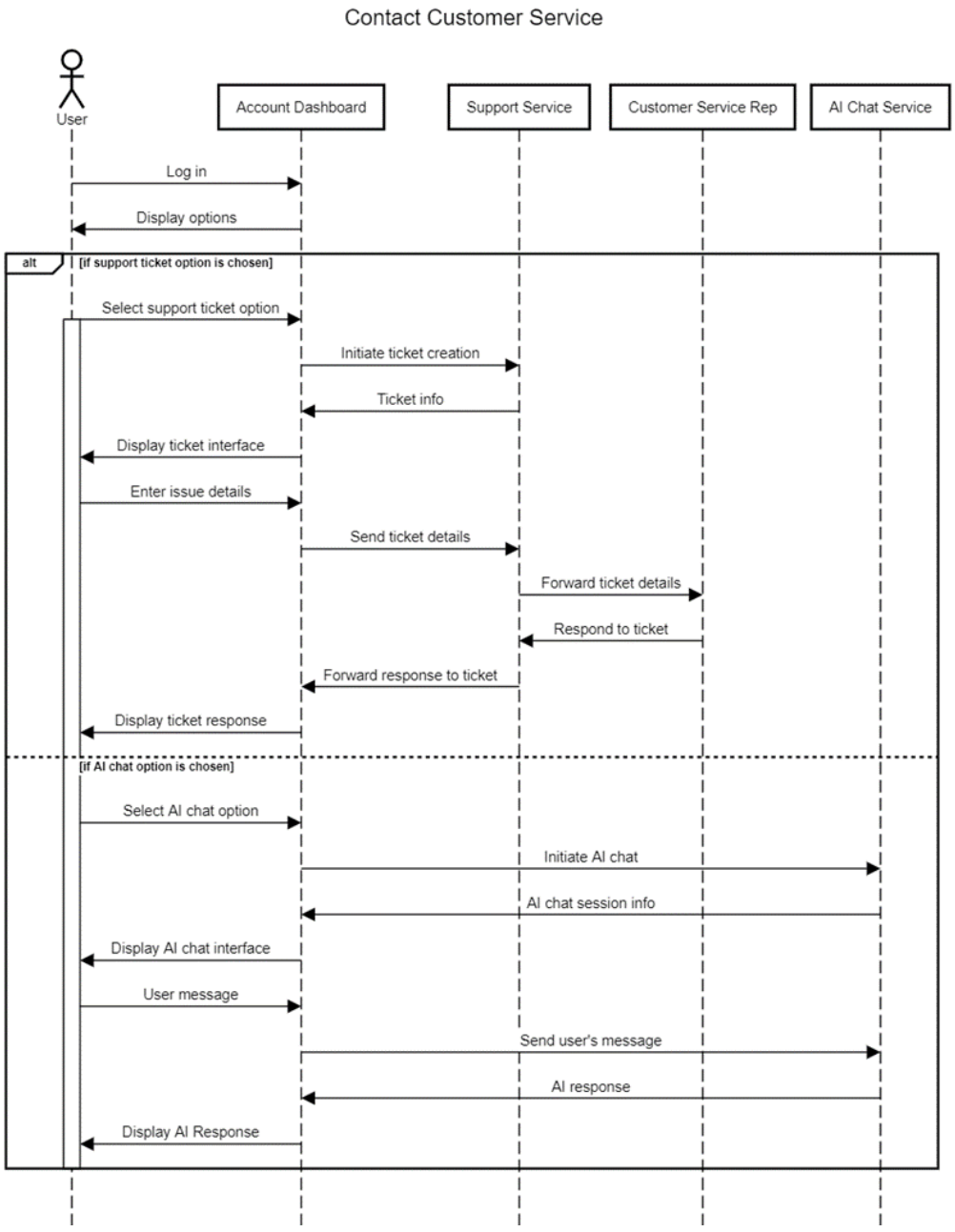


---

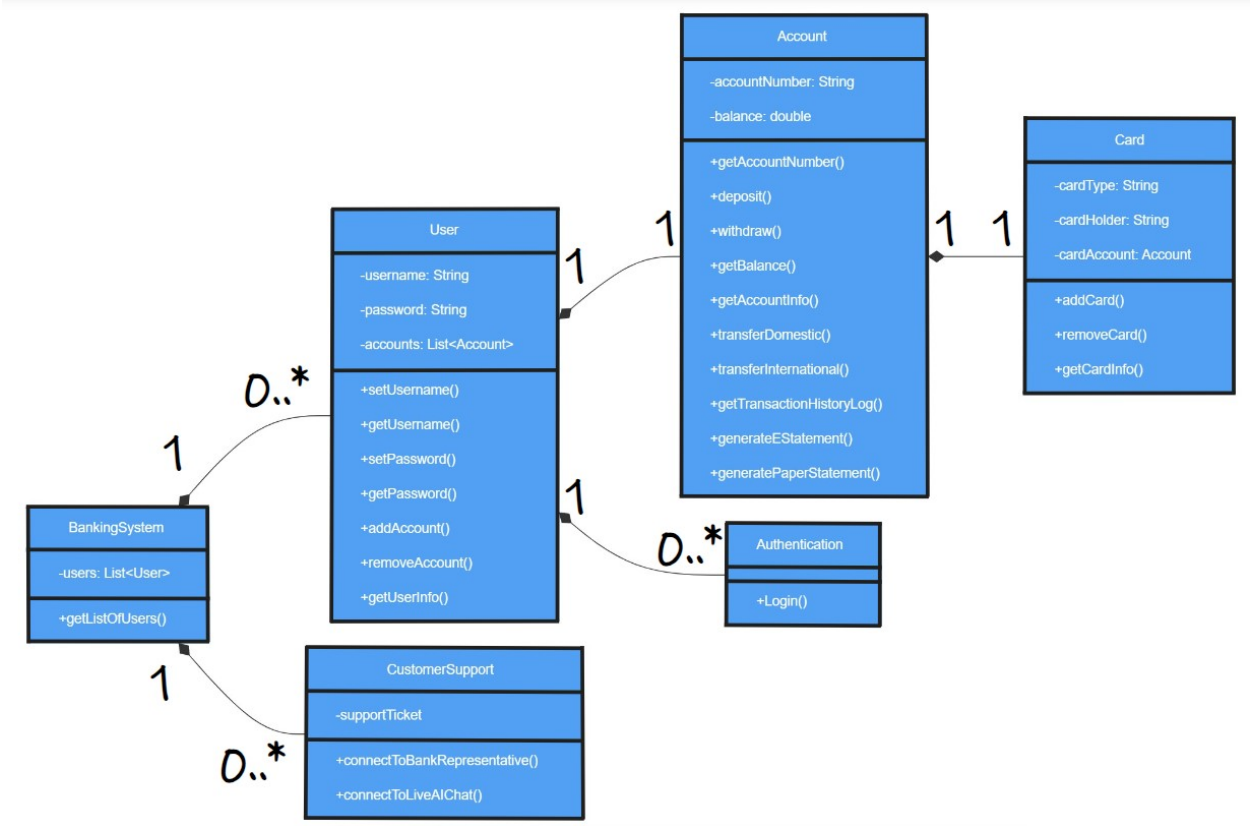
title **Contact Customer Service**

actor User
User->Account Dashboard: Log in
Account Dashboard->User: Display options
alt if support ticket option is chosen
User->Account Dashboard: Select support ticket option
activate User
Account Dashboard->Support Service: Initiate ticket creation
Support Service->Account Dashboard: Ticket info
Account Dashboard->User: Display ticket interface

User->Account Dashboard: Enter issue details
Account Dashboard->Support Service: Send ticket details
Support Service->Customer Service Rep: Forward ticket details
Customer Service Rep->Support Service: Respond to ticket
Support Service->Account Dashboard: Forward response to ticket
Account Dashboard->User: Display ticket response
else if AI chat option is chosen
User->Account Dashboard: Select AI chat option
Account Dashboard->AI Chat Service: Initiate AI chat
AI Chat Service->Account Dashboard: AI chat session info
Account Dashboard->User: Display AI chat interface
User->Account Dashboard: User message
Account Dashboard->AI Chat Service: Send user's message
AI Chat Service->Account Dashboard: AI response
Account Dashboard->User: Display AI Response
end
deactivate User


Contact Customer Service
User
Account Dashboard
Support Service
Customer Service Rep
AI Chat Service
Log in
Display options
alt
[if support ticket option is chosen]
Select support ticket option
Initiate ticket creation
Ticket info
Display ticket interface
Enter issue details
Send ticket details
Forward ticket details
Respond to ticket
Forward response to ticket
Display ticket response
[if AI chat option is chosen]
Select AI chat option
Initiate AI chat
AI chat session info
Display AI chat interface
User message
Send user's message
AI response
Display AI Response

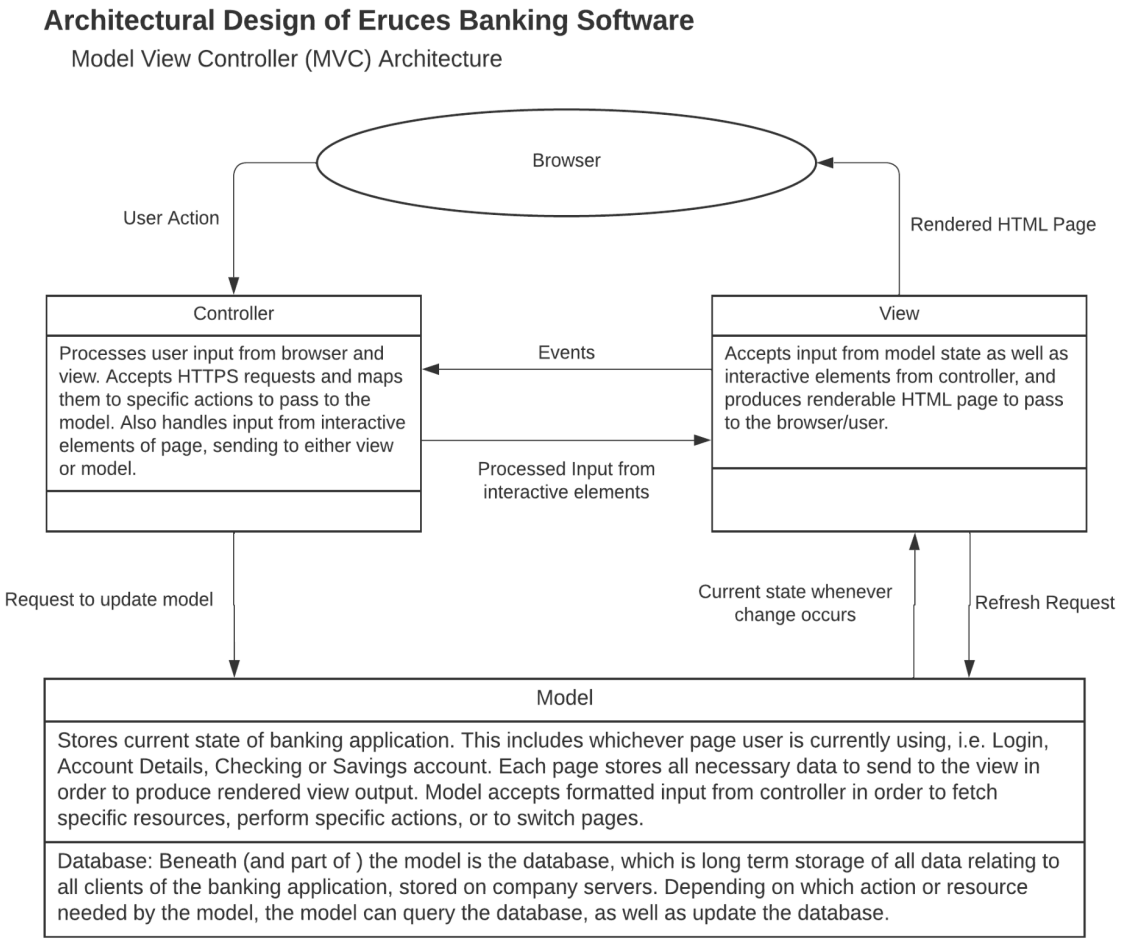
8: Class Diagram



9: Architectural Design

The Eruces online banking application will use the Model View Controller (MVC) architectural system. This system has been chosen because of several advantages that it offers:

1. Wide adoption: MVC has been an industry standard for years in the development of web applications, and is thus well documented and with mature specifications
2. Existing tools: tools such as Microsoft's ASP.NET MVC Pattern can speed development of MVC applications.
3. Flexibility of Data Representation: because of the separation between data and views in MVC, the representation of customer data can be changed independently of the actual data. Also allows for platform flexibility, such as an app vs. desktop browser based application.
4. Requirements Flexibility: MVC allows a high degree of change in application requirements, which fits the Spiral development model used by Eruces. The architectural design can easily accommodate any feedback in each iteration.



End of Project Deliverable 1 contents.

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

3.1. Project Scheduling

Based on our Function Point Analysis and productivity rates,we plan for the project duration to be approximately 4 months with a project start date of December 1, 2023 and a project end date of March 31, 2024. The schedule accounts for 5 working days a week, which does not include weekends, and an effective 7-hour workday taking into account an 8-hour workday with a 1-hour lunch break. We scheduled the project in this way due to the complexity of an online banking application, especially its security features, and to give time for testing and feedback. Additionally, the schedule includes a buffer for unexpected delays that may arise during the project lifetime.

3.2. Cost, Effort, and Pricing Estimation

We will employ the Function Point Analysis to estimate the expenditure, labor intensity, and pricing of the Online Banking Software Project. Function Points offer a standardized metric to assess the functional dimensions of software development. This measurement aids in quantifying the various features and user interactions the application will facilitate, thereby allowing us to establish a proportional relationship between the functional scope of the project and the effort involved in its realization.

I. Determine Function Category Count

A. Number of User Inputs:
1. Username and password entry
2. Verification process
3. Deposit money operation
4. Withdrawal money operation
5. Domestic transfer operation
6. International transfer operation
7. Open and close cards operation
*Total Inputs: **7***

B. Number of User Outputs:
1. Display account number
2. Show balance
3. View transactions
4. View cards
5. Display history log
6. Generate e-statements
7. Generate paper statements
8. SupportTicket information
9. Live AI chat interaction
*Total Outputs: **9***

C. Number of User Queries:
1. Query account overview
2. Query transactions
3. Query cards
4. Query account statements
5. Query history log
6. Query statement generation
*Total Queries: **6***

D. Number of Data Files and Relational Tables:
1. Account information table
2. Transaction history table
3. Card information table
4. Statements table
5. Support history table
*Total Tables: **5***

E. Number of External Interfaces:
1. Interface with SupportTicket system
2. Interface with Live AI chat system
3. Interface for Other bank or platform
*Total Interfaces: **3***

II. Determine the Complexity

A. User Input (Simple):
User input operations are considered basic data entry tasks, such as entering usernames, passwords, and transaction amounts. These straightforward tasks do not involve complex logic or multi-step verification processes.

B. User Output (Average):
The outputs may involve displaying sensitive data, necessitating encryption during transit and secure display on the client side, therefore considered average.

C. User Queries (Complex):
These queries often span multiple data tables and require sophisticated permissions checks, data encryption, and potentially involve complex joins. The complexity is further heightened by the necessity to prevent unauthorized access and to ensure the integrity and confidentiality of sensitive financial data.

D. Data Files and Relational Tables (Complex):
Given that banking systems typically handle highly sensitive and intricate data and must maintain strict data integrity and security, the complexity of data storage and management is naturally high. This also likely involves complex data relationships and multi-table joins.

E. External Interfaces (Average):
The banking software does not pursue flashy features but instead selects mature and stable technologies for its external interfaces. This preference leads to interfaces that are typically backed by strong support and enriched with comprehensive documentation, greatly simplifying both the development and the maintenance processes.

III. Compute gross function point (GFP)

	Function Category		Complexity	Count x Complexity

		Count	Simple	Average	Complex	
1	Number of user input	7	3	4	6	21
2	Number of user output	9	4	5	7	45
3	Number of user queries	6	3	4	6	36
4	Number of data files and relational tables	5	7	10	15	75
5	Number of external interfaces	3	5	7	10	21
Gross Function Point						198

IV. Assessment of Processing Complexity (PC)

The computation of the Processing Complexity (PC) value is a pivotal step in our estimation process. This value is derived from the responses to a set of factors known as the complexity weighting factors. Each factor is evaluated and assigned a numerical value on a scale from 0 to 5, where the values represent the following degrees of influence:

0: No influence
1: Incidental influence
2: Moderate influence
3: Average influence
4: Significant influence
5: Essential influence

For the Online Banking Software Project, these factors are tailored to our specific operational requirements and potential complexities. Below is a revised table reflecting this adaptation:

No.	Complexity Weighting Factor	Value
1	Reliable Backup and Recovery	5
2	Data Communications	5
3	Distributed Processing Functions	2
4	Performance Critical	4
5	Existing Operational Environment	4
6	Online Data Entry	5
7	Input Transaction Over Multiple Screens	3
8	Master Files Updated Online	4

9	Complexity of Inputs, Outputs, Files	3
10	Internal Processing Complexity	2
11	Code Reusability	3
12	Conversion/Installation Design	3
13	System Designed for Multiple Installations	3
14	Application Designed for Change	5
Total Processing Complexity		51

V. Calculation of Processing Complexity Adjustment (PCA)

The formula for PCA is as follows:
PCA=0.65+0.01×(Total Processing Complexity)
PCA=0.65+0.01×51=**1.16**

VI. Function Point (FP) Calculation

To determine the Function Points (FP):
FP=Gross Function Point (GFP)×PCA
With our GFP at 198, the FP is:
FP = 198 * 1.16 = **229.68**

VII. Effort and Project Duration Estimation

The effort, in person weeks, is estimated as:
E= FP/productivity
Using a productivity factor of 60:
`E = 229.68 / 60 = 3.828 ≈4 person−weeks`
For a team of 7 developers, the project duration, in weeks, is:
`D = E / team sizes = 4 / 7 ≈ 1 week`

3.3. Estimated Cost of Hardware Products

Within the banking sector, the pivotal concerns of security and compliance drive a distinct preference for physical servers over their cloud-based counterparts. The selection of physical servers is contingent upon several factors, including the scale of the user base and the volume of data to be managed.

A typical physical server, estimated at a cost of "$1,476.31 [1]", underscores the financial aspect of deploying and maintaining the necessary infrastructure for a secure banking system. In the initial stages of our online banking app proposal, we will commence with a single server, with a flexible strategy to incrementally add more servers as the project scope expands.

3.4. Estimated Cost of Software Products

In order to develop the Eruces online banking system, no special software will be needed. Employees will be able to use free and open-source IDEs to develop. However, there are two areas where our software will interface with external software services, which will cost money.

The first area is for authentication. In order to provide a high level of security and usability, our application will need to interface with a third party authentication software that is based on a subscription service. This will enable two-factor authentication for all of our users. One company that provides this feature is Duo security, which will be used to provide an estimate for this software service. The middle tier of service is selected for estimation purposes.

- Price: $6 per user per month = $72 per user per year [2]
- Estimating approximately 5000 users for the app at launch
- ~$360000 per year on authentication

The second service is an LLM API to power the customer support chat. The best one currently out is OpenAI's GPT-4:

- Price: $0.01 per 1000 tokens. 1000 tokens is about 750 words or one page of text [3]
- Estimating that each AI customer support ticket will use on average 750 tokens
- With about 5000 users, every day about one percent will have an issue.
- 5000 * 0.01 = 50 per day
- 50 * 0.01 * (750 / 1000) = $0.375 per day, or ~$140 per year on LLM API

Combined, the total estimate for software costs per year is $360140. Almost all of this is on authentication, which means that it may be too expensive to support. Because of this, a third party two-factor authentication solution should only be considered if our own authentication is deemed to be too insecure.

3.5. Estimated Cost of Personnel

This section estimates the personnel costs for the Online Banking Software Project. The cost estimation considers the composition of the development team, their respective experience levels, and the project duration. We have selected a combination of mid-level and senior-level professionals to balance expertise, project requirements, and budget considerations.

Development Team Composition:
Front-end Developers: **2** Mid-Level ($40 - $70 per hour)
Back-end Developers: **2** Senior-Level ($75 - $130 per hour)
Quality Assurance Engineers: **2** Mid-Level ($48 - $57 per hour)
Project Manager: **1** Senior-Level ($49 - $62 per hour)

Estimated Hours:
Project Duration: 1 week (40 hours)
Total Hours (per role): **40 hours**

Cost Calculation:
Front-end Developers: 2 x $55 (average) x 40 hours = **$4,400**
Back-end Developers: 2 x $102.5 (average) x 40 hours = **$8,200**
QA Engineers: 2 x $52.5 (average) x 40 hours = **$4,200**
Project Manager: $55 (average) x 40 hours = **$2,200**

Total Personnel Cost:
Total = $4,400 (Front-end) + $8,200 (Back-end) + $4,200 (QA) + $2200 (PM) = **$19,000**

Training Cost:

Given the intuitive nature of the banking software and the limited number of personnel requiring training (estimated at 20), along with the use of a mix of digital and physical training materials estimated at $500, and without the need for external trainers:

Number of Trainees: 20 employees.
Training Materials and Cost: Approximately $200 total for both digital and physical resources.
Internal Trainers: Assuming the training is conducted by internal staff, there may be minimal to no additional cost for trainers.

Total = 20 x $200 = **4,000$**

$19,000 + $4,000 = **$23,000**

4. Test Plan

This program tests the withdraw() function in our online banking application. The simulation assumes that the user currently has $1000 in their account, but in the real application, the program would provide other methods that allow more generalization.

The withdraw() method takes in a parameter that represents the amount that the user wants to withdraw and returns true or false depending on whether the withdrawal was successful or not.

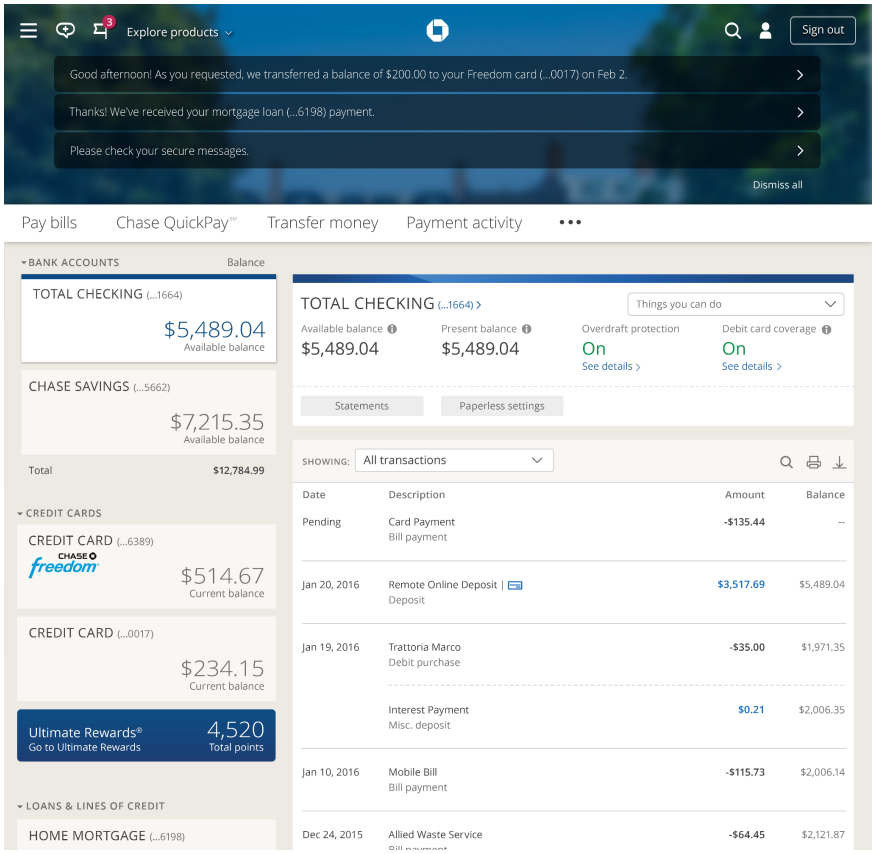
JUnit testing:
- testCorrect() attempts to withdraw $200 from the account, which should be successful.
- testIncorrect() attempts to withdraw $2000 from the account, which should not be successful.

The results of the testing are attached as images in the zip file.

5. Comparison of Work with Similar Designs

Compared to other banking applications of similar designs, our banking application is more streamlined and focused. Most online banking applications, also known as banking portals, provide utilities like “account management, transaction history, fund transfers, bill payments, and online applications for financial products” [4], and our application does all of these. Compared to other banking apps out there, ours also comes with an AI chatbot to augment customer support.

Here, we are comparing our application to Chase’s web application.



Chase Web Portal Front Page [5]

The Chase web app has more activities like QuickPay and rewards, along with several notifications at the top. On the other hand, our app ditches these in favor of a more focused experience.

6. Conclusion

We believe that our work on this banking application has been good. We started off with a banking application in mind and we have followed through with that. We did not need to make any changes.

7. References

[1] Collins, Tom. “Dedicated vs. Cloud: How Much Does a Server Cost?” Atlantech Online, 7 Apr. 2022, www.atlantech.net/blog/cloud-vs.-dedicated-server-cost-which-is-the-better-deal.

[2] R. Mohanakrishnan, “Top 10 Multi-Factor Authentication Software Solutions for 2021,” *Spiceworks*, Jun. 24, 2021. https://www.spiceworks.com/it-security/identity-access-management/articles/top-10-multi-factor-authentication-software-solutions/ (accessed Nov. 17, 2023).

[3] OpenAI, “Pricing,” *openai.com*, 2023. https://openai.com/pricing (accessed Nov. 17, 2023).

[4] businesswire, “Banking on the new chase.com is even easier,” *businesswire*, Mar. 10, 2016. https://www.businesswire.com/news/home/20160310005849/en/Banking-on-the-new-chase.com-is-even-easier (accessed Nov. 17, 2023).

[5] Kvartalnyi, Nazar, “What Software Does Online Banking Use?,” *inoxoft*, Aug. 22, 2023. https://inoxoft.com/blog/what-software-does-online-banking-use (accessed Nov. 17, 2023).

[6] Kaushal, D, “How to estimate custom software development cost in 2024,” *netsolutions*, 2023. https://www.netsolutions.com/insights/custom-software-development-cost.

[7] Salary.com, “Hourly Wage for Senior Project Manager Salary in the United States,” *Salary.com*. https://www.salary.com/research/salary/listing/senior-project-manager-hourly-wages#:~:text=The%20average%20hourly%20wage%20for,falls%20between%20%2449%20and%20%2462.