Computer Science and Engineering

Happy Budget
Software Design Description (SDD)

Version 1.0

Document Number SDD-001

Project Team Number: B27
Project Team Members: Amanda Lin (al5885), Gordon Lei (gl1776), Jason Li (jl8466), Jay Kang (yjk44)

REVIEW AND APPROVALS

<Team Members>	Function (Author, Reviewer, Approval)	Date	Signature
Amanda Lin	Author, Poster	03/08/2021	On File
Gordon Lei	Author	03/08/2021	On File
Jason Li	Author	03/08/2021	On File
Jay Kang	Author	03/08/2021	On File

REVISION LEVEL

Date	Revision Number	Purpose
03/08/2021	Version 1.0	Initial Release

Table of Contents

1. INTRODUCTION

1.1 Purpose

The purpose of this document is to define the contents of the Software Design Description (SDD) for the Happy Budget system. The SDD documents the System Architecture and the Detailed Design.

The document is used to communicate the overall quantitative and qualitative system characteristics to stakeholders, operations management, technical support, training, software quality group, and operators.

1.2 Scope

The Happy Budget System is an interactive web-based application for personal budget planning and personal finance education. The system is intended for individual use by adolescents and young adults to provide an outlet to alleviate the learning gap for personal finances. The system will include the following functions:

1. Balance Tracking

    1.1 A user shall be able to link and access their bank account through the application.

    1.2 A user shall be able to input their own data to keep track of their finance and transactions in case the user is unwilling or unable to connect their bank account.

    1.3 A user shall be able to view their balance of their connected bank account.

    1.4 A user shall be able to add to their connected savings account balance if one is connected.

2. Statistics Visualization

    2.1 The application shall display various statistics and trends of the user and other consenting users in easy-to-understand and informative data visualizations.

3. Personal Goals

3.1 A user shall be able to create and keep track of their personal goals.

3.2 The application shall display various statistics and trends regarding the user's personal goals and their investments into those personal goals.

4. Information Delivery

4.1 The application should provide informative financial tips to help the user with their money management.

4.2 The application should provide information regarding terms and concepts used in economics.

5. Interactive Play

5.1 The application shall allow the user to have interactive play when investing in their set personal goals.

The system will take advantage of existing digital financial bank APIs to gather information about the user's standing balance and allow the user to access and use their money. In the case where this is not possible, the system will allow for user input of personal finance data. The system will not run on a new, different software, but on at least one type of web browser. Existing web application development languages such as Python will be used to develop the application. Additionally, existing visual statistics APIs will be used to provide visual statistics and existing database services to store financial tips and user information and data.

1.3 Identification

Happy Budget Software Design Description (SDD), Document Number SDD-001, Version 1.0

1.4 Document Summary

This document contains the following sections:

- System wide design decisions: details the design decisions of the system
- Software item detailed design: details the design of the software units
- Deployment architecture: details the hardware architecture with software, network, and hardware details

- Software item computer resource utilization: details computer resource allocation for the system
- Requirements traceability: details requirements traceability
- System design testing: details software quality group testing and product and acceptance testing
- Appendices: contains dictionaries, UML diagrams not included in the body of the document, requirements descriptions and diagrams, schedule tracking, defect tracking and the project schedule

1.5 System Overview

The system will allow a connection to a personal bank account so that users will be able to see their total balance through this connection and interact with their money. Existing APIs for financial bank connections will be utilized and associated rules and guidelines of the institution will be abided by.

The application must track the user's spending to provide quantitative data and inform the user about their spending habits visually. The user's spending will be tracked through the financial bank API and the visual quantitative data will be displayed using a visual statistics API.

Interactive play to maintain the user's interest and allow the user to gain first-hand experience will take the form of interactive pet interactions scattered throughout the system such as when the user puts money into their goal. These interactive pet interactions must be developed.

The application will support the addition, removal, and priority setting of personal goals that will provide first-hand experience in learning to save money towards a goal. A visual statistics API will be utilized to provide visual quantitative data on the goals progress.

Personal financing tips should also be provided throughout the system's lifetime. These tips should be stored in a database for use throughout the system.

The system will not run on a new, different software, but on at least one type of web browser. Existing web application development languages such as Python will be used to develop the application.

2. REFERENCE DOCUMENTS

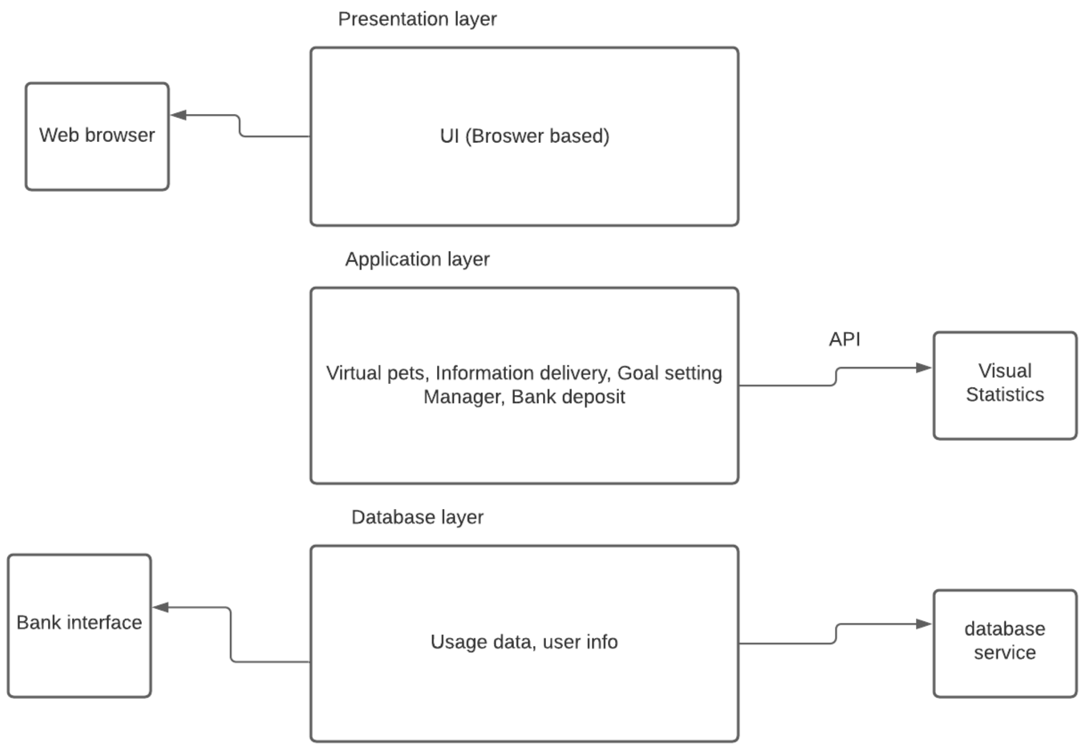
1. Team B27, Project Proposal for Happy Budget, Document Number 001, Version 1.1, 02/07/2021.

2. Team B27, Happy Budget System Requirements and Analysis Specification (SRS), Document Number SRS-004, Version 4.0, 02/17/2020.

3. Team B27, Happy Budget Project Management Plan (SPMP), Document Number SPMP-001, Version 1.0, 11/05/2020.

3. SYSTEM WIDE DESIGN DECISIONS

3.1 Software Component Architectural Design



3.2 Software Architectural General Description

Presentation layer- The application will have a web-browser based UI that the user will use to interact with the features and data of the application.

Application layer- The application layer includes features of the application the user can utilize through the UI. The features include user-side actions such as goal setting and depositing money into their bank account, but also application-side actions such as financial tips and visual statistics of their activities, which is managed by an external API.

Database layer- The database layer manages databases necessary to securely store and access data for users. The layer also includes connection to external API and services to access and handle bank transactions and information. The database also stores user's usage data that is used to calculate trends.

3.3 Software Item Components

Browser based user interface: The user interface will be browser-based. The user will be able to log onto their accounts and access the various features of the applications.

Virtual pets: To provide a sense of interaction, the application will feature a virtual pet that a user can interact with when setting goals and investing into their savings. (5.1)

Information delivery: Based on personal usage data and data from other users, the application will deliver useful information on savings that will be beneficial to the user. The application will also present general advice about finance and economics. (4.3) (4.4)

Goal setting manager: The user will be able to set goals, which will store into the user info database. (3.3)

Statistics visualizer: The user will be able to see a summary of how they and other users have been spending or saving money. The summary will be presented in a visual format that is easy to understand. The visual will be provided by an external reusable component when given the input data. (2.2)

Usage data storage: A database that will store various (anonymous) information about user activity. The database will be managed by an external database management system. (2.2) (3.4)

User info storage: A database that will store user's information. These information range from login detail, personal information (names, bank account), and user input such as goals, savings plans, and tags to determine what info to show the user will also be stored in this database. The database will be managed by an external database management system. (3.3)

Bank deposit: The user will be able to connect to their bank account from the application. When the user makes deposits or bank transactions from the app, the application will send a secure, and quick request to the bank to carry out those transactions. (1.5) (1.7)

3.4 Component Interface Identification

The following interfaces will be used in the system:

- Web Server
- Bank API
- Visual Statistics API
- Database Service
- Application to UI / UI to Application Interface
- Application to Database / Database to Application Interface

3.5 Software Component Concept of Execution

The system is executed as follows:

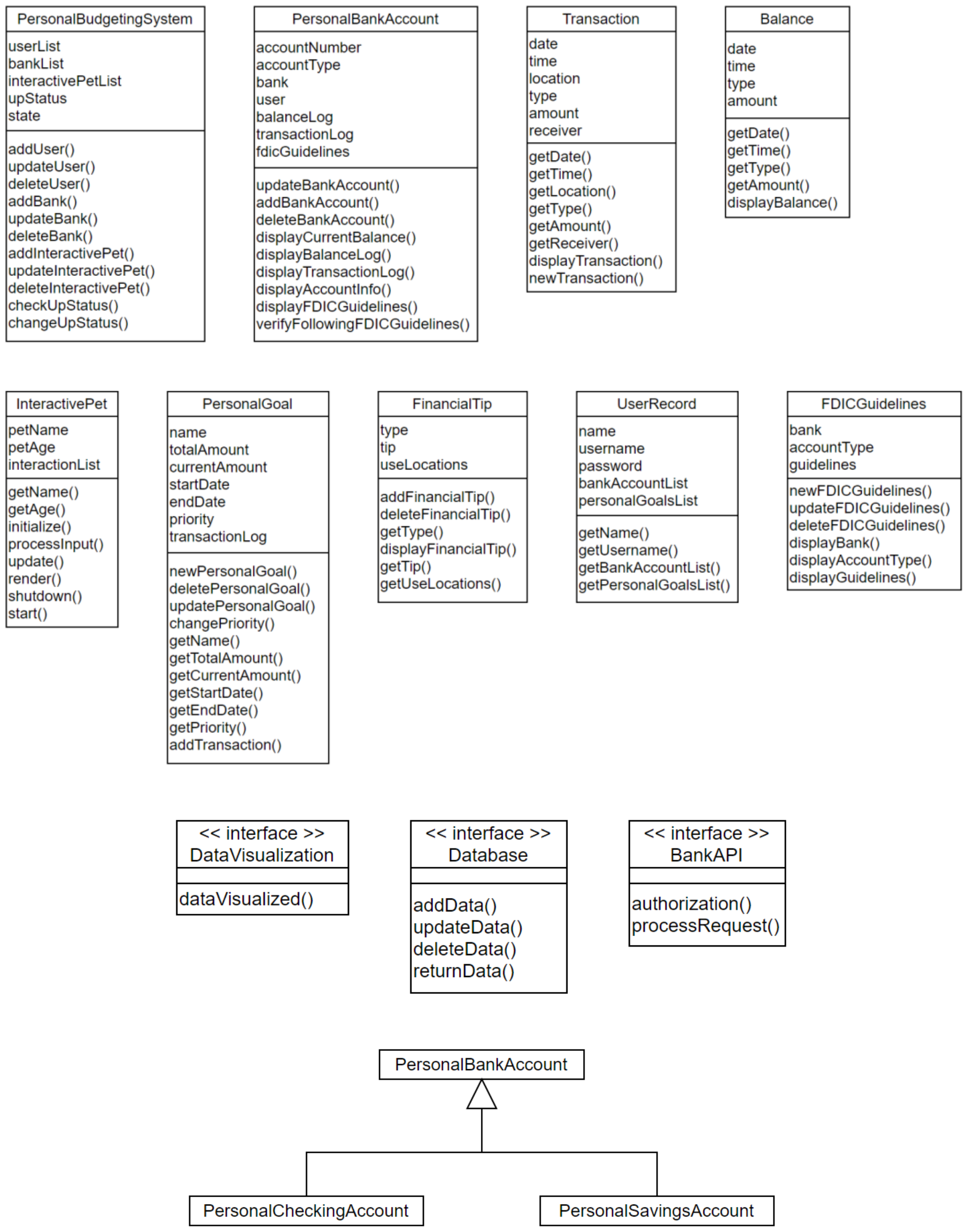
1. Launch the application server
2. Launch the database server
3. Connect to the database service
4. Connect to the bank API
5. Send information from the database server to the application server
6. Connect to visual statistics API
7. Launch the user interface
8. Connect to the web browser
9. Send information from the user interface to the application server

4. SOFTWARE ITEM DETAILED DESIGN

4.1 Structure

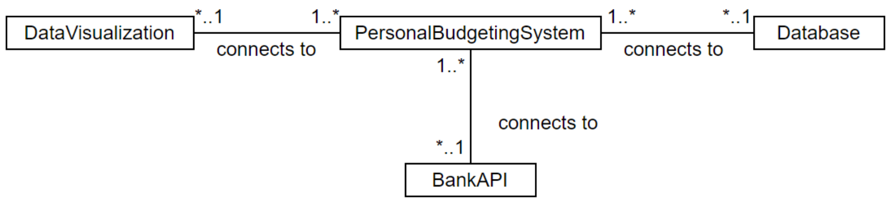
4.1.1 Software Unit Detailed Design

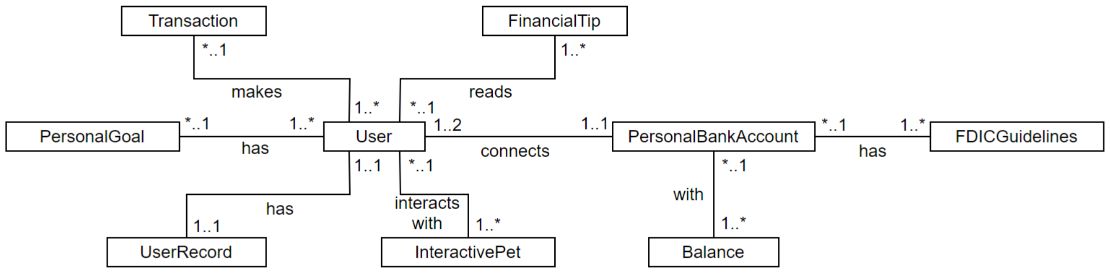
Class diagrams:



4.2 Static Relationship of Software Unit

Class interaction diagram:





4.2.1 Run-time Object Instances

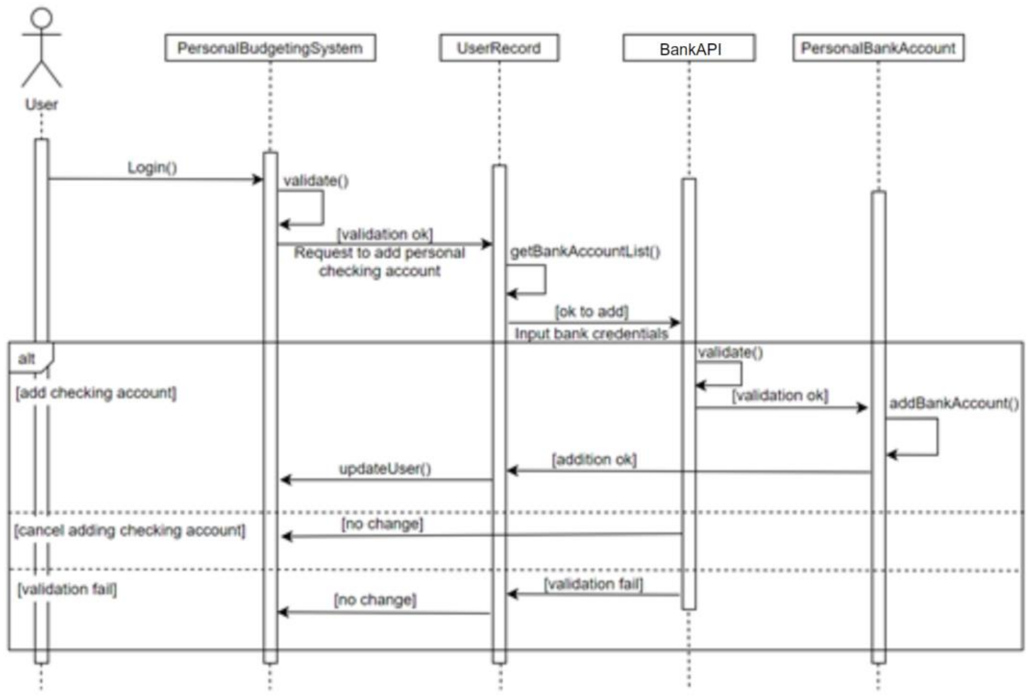
When the application is launched, an Apache server process is made to host the web application and a Postgres server process is made to handle the database. The application will be single threaded as a single user will not be conducting multiple actions at the same time. All object classes will be instantiated within the application’s single threaded process.

Data sent to AWS servers will be queued using fair queueing; in other words, data that has the largest impact to the database will be prioritized first. Examples of such data include the editing of account data, the creation of accounts, interactions between connected bank accounts, etc.

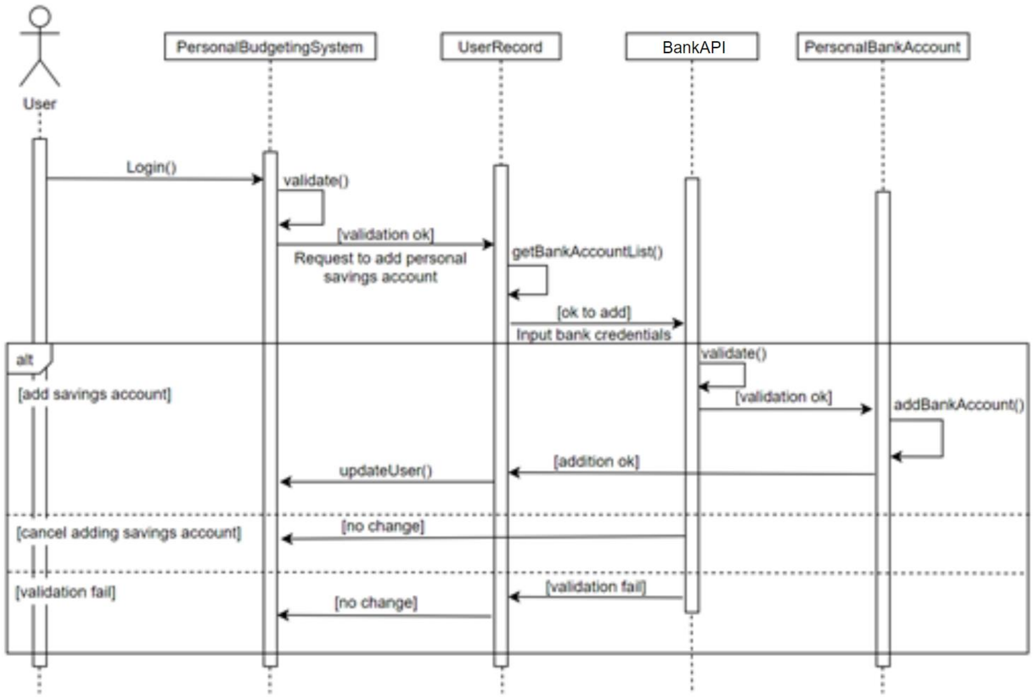
4.3 Behavior

4.3.1 Sequence Interaction Diagrams

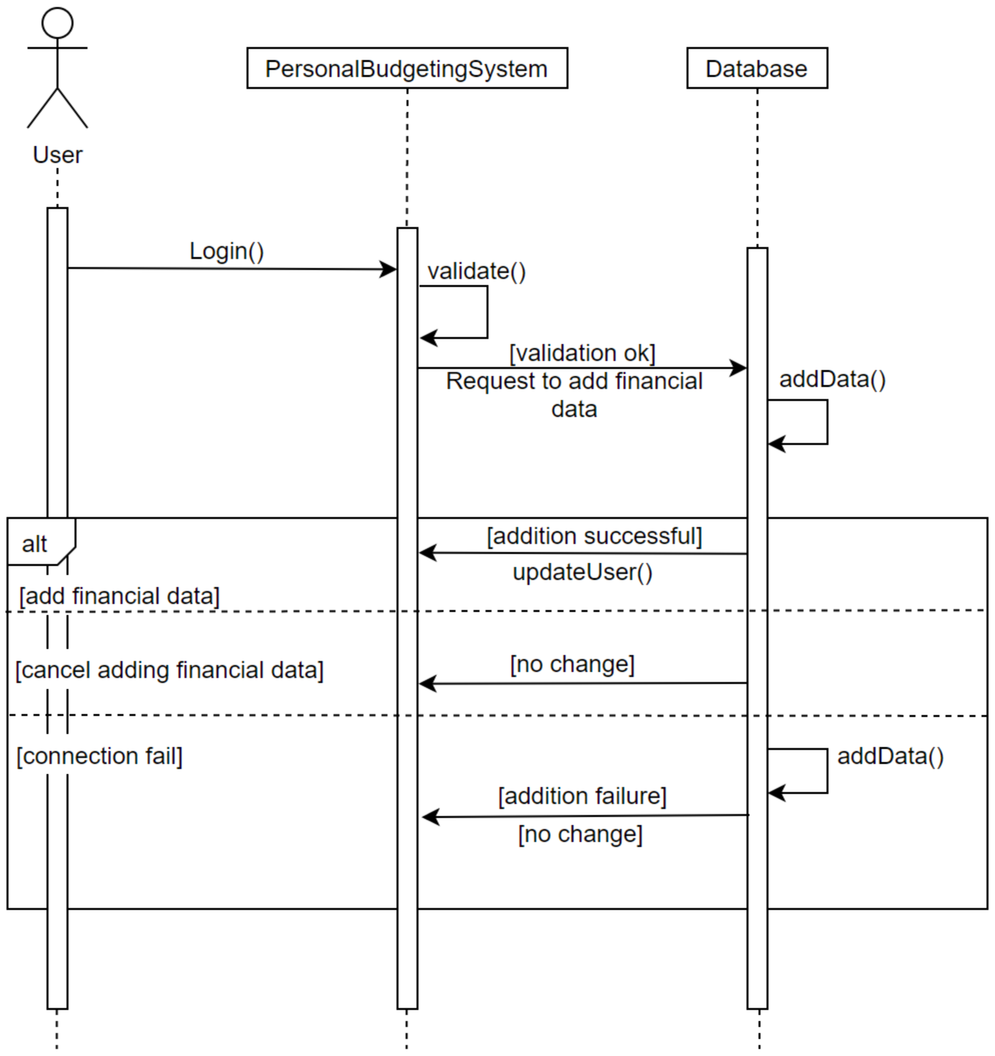
Add Checking Account (1.1)



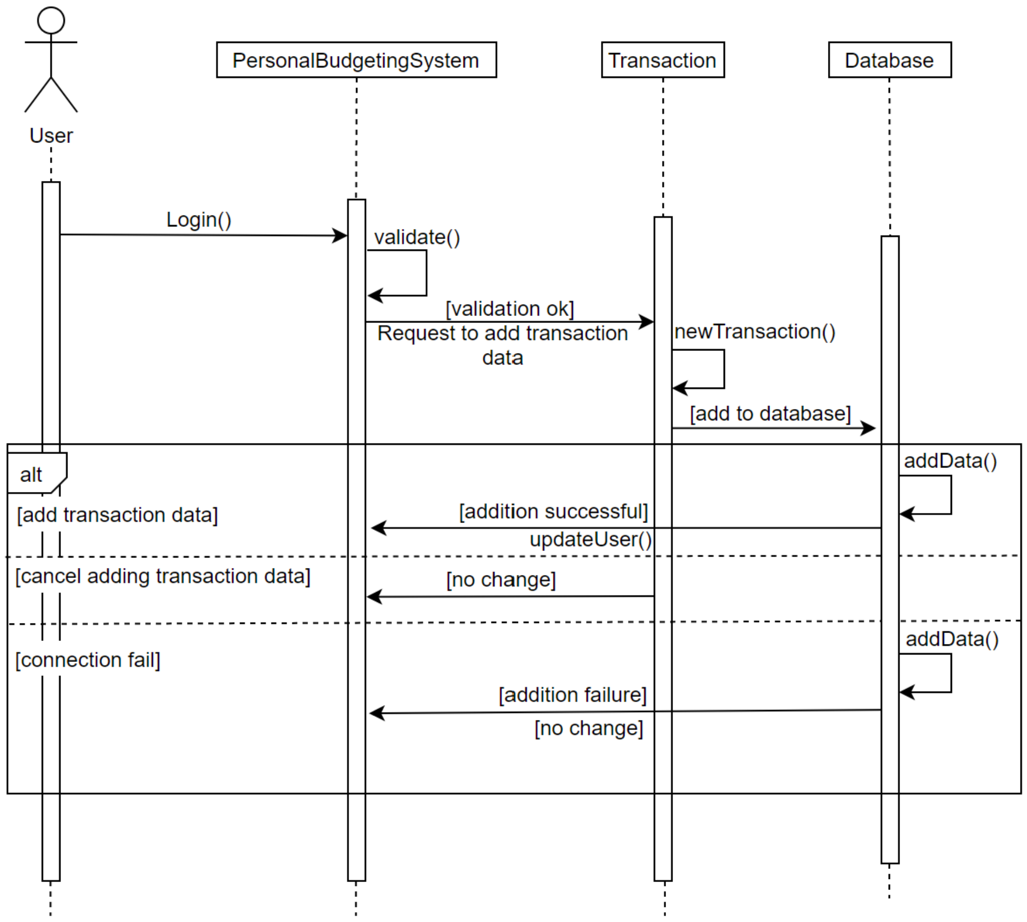
Add Savings Account (1.1)



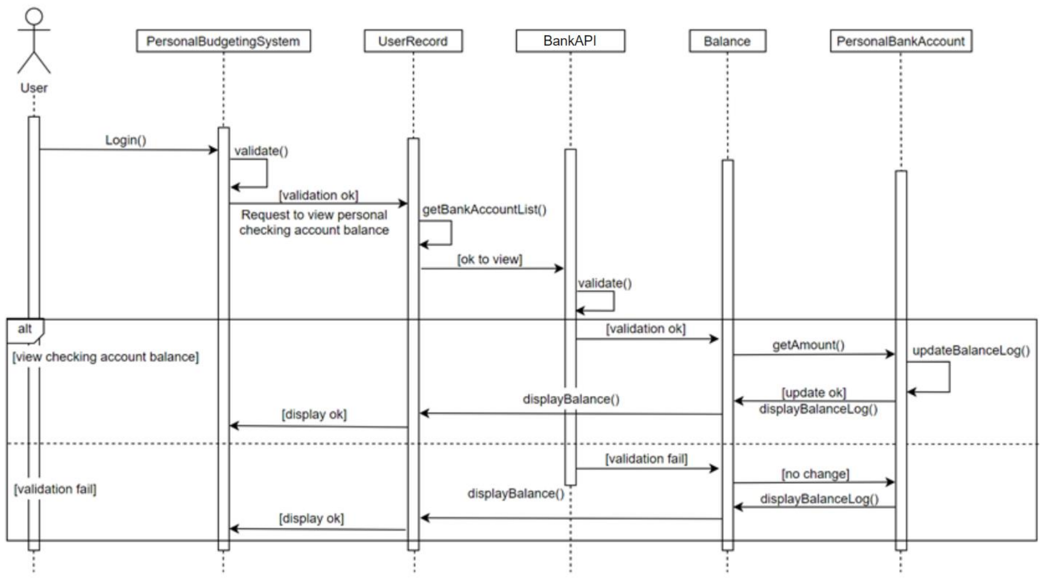
Input Financial Data (1.2)



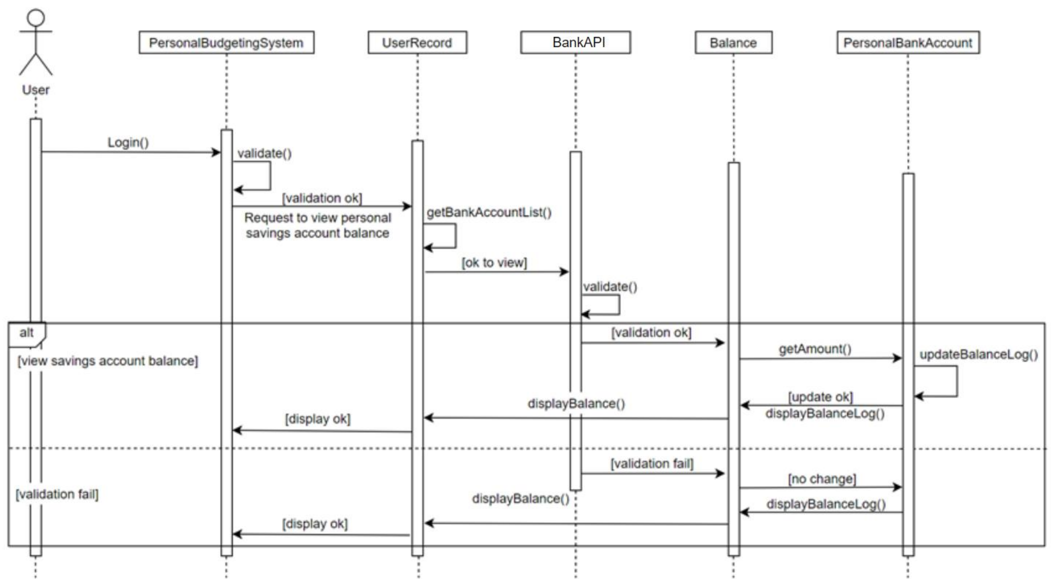
Input Transaction Data (1.2)



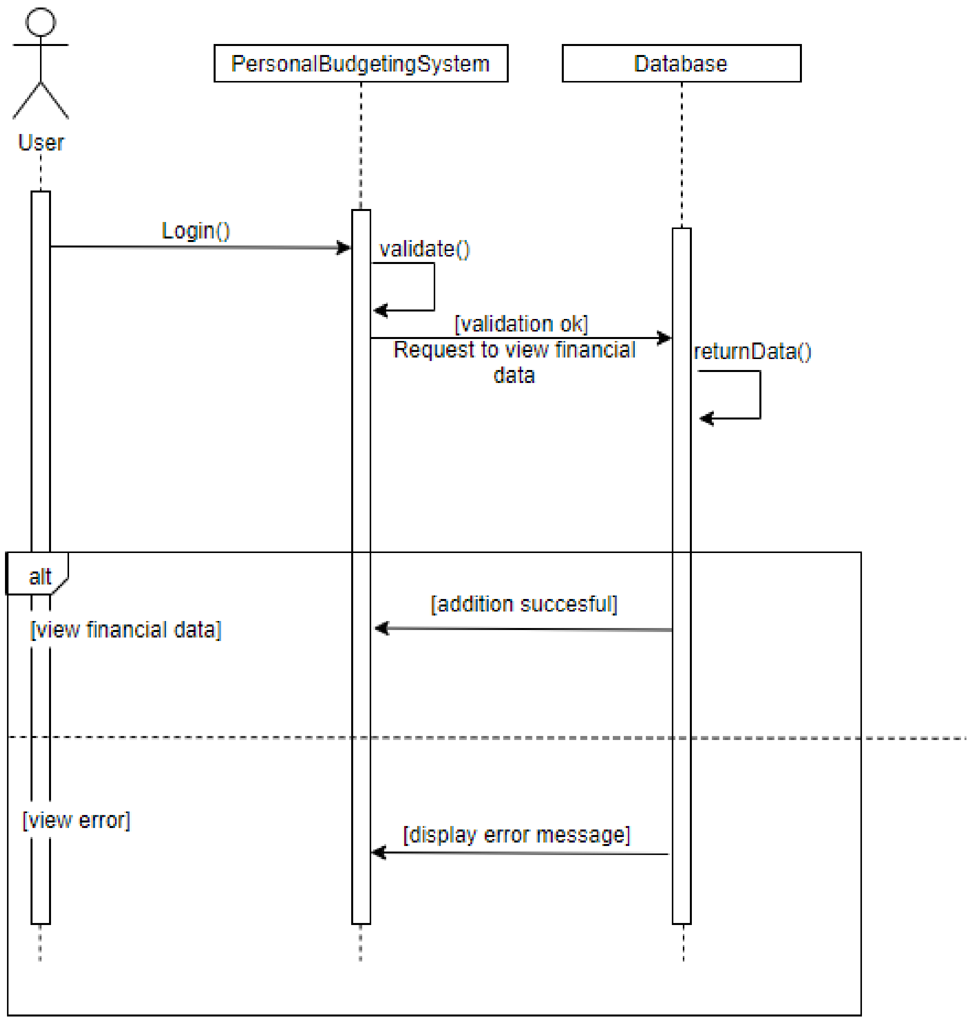
View Checking Account Balance (1.3)



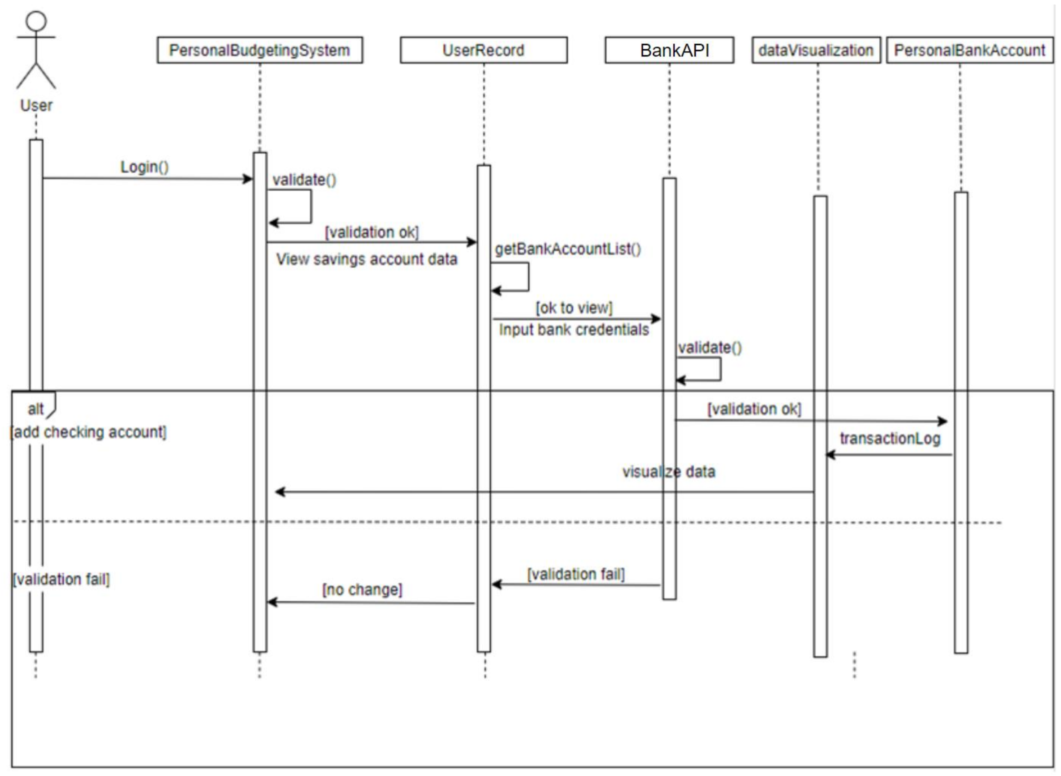
View Savings Account Balance (1.3)



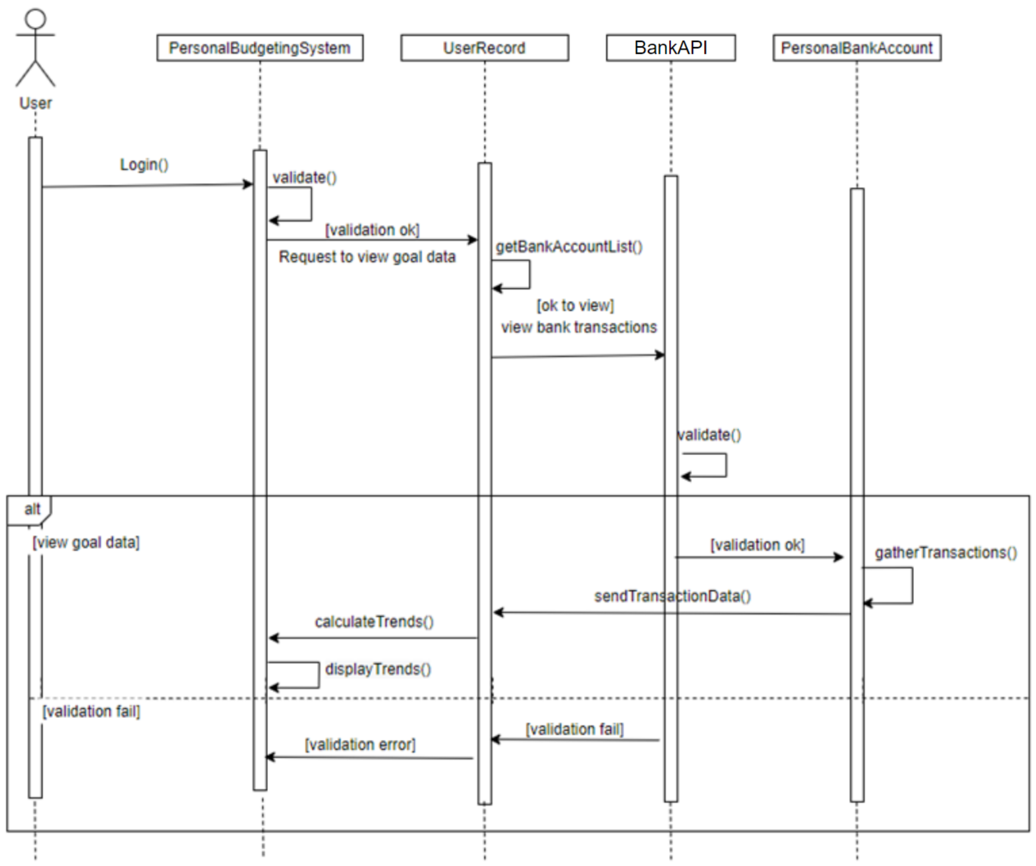
Financial Data (2.1)



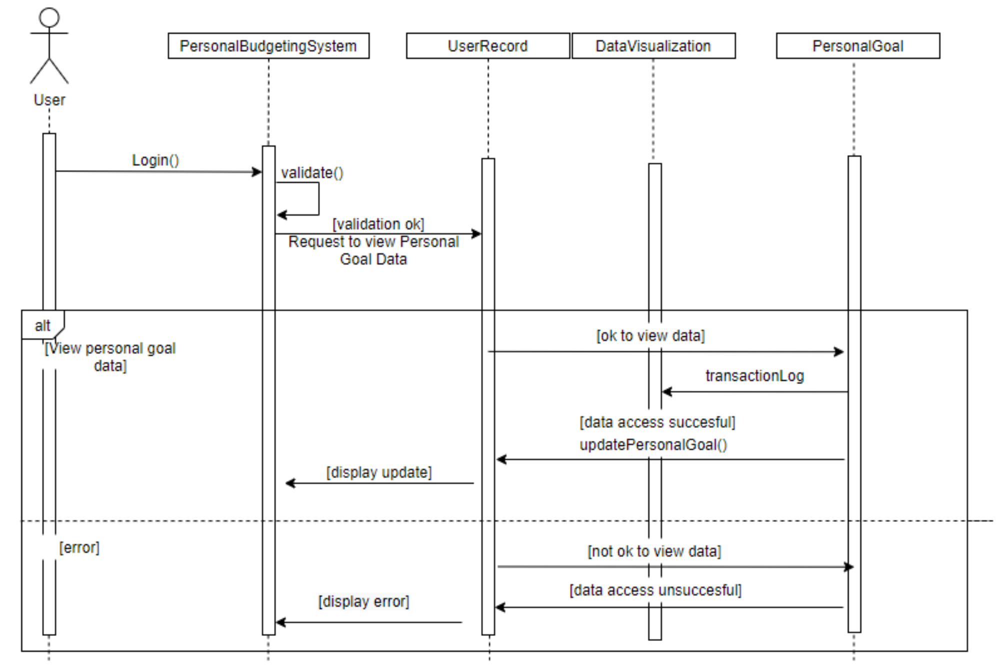
View Savings Account Data (2.1)



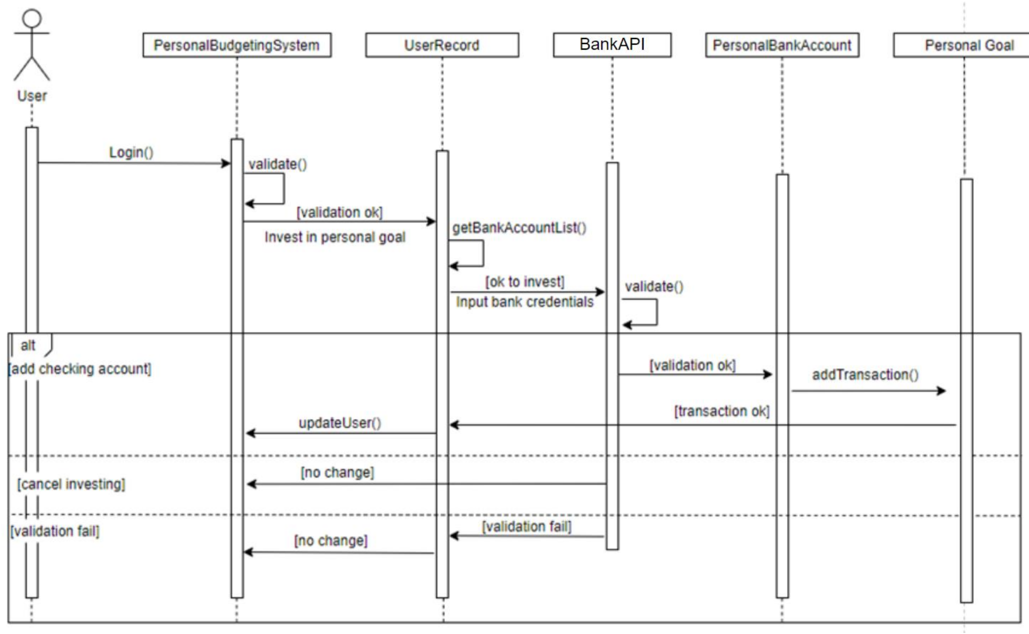
Add/Remove Personal Goals (3.1)



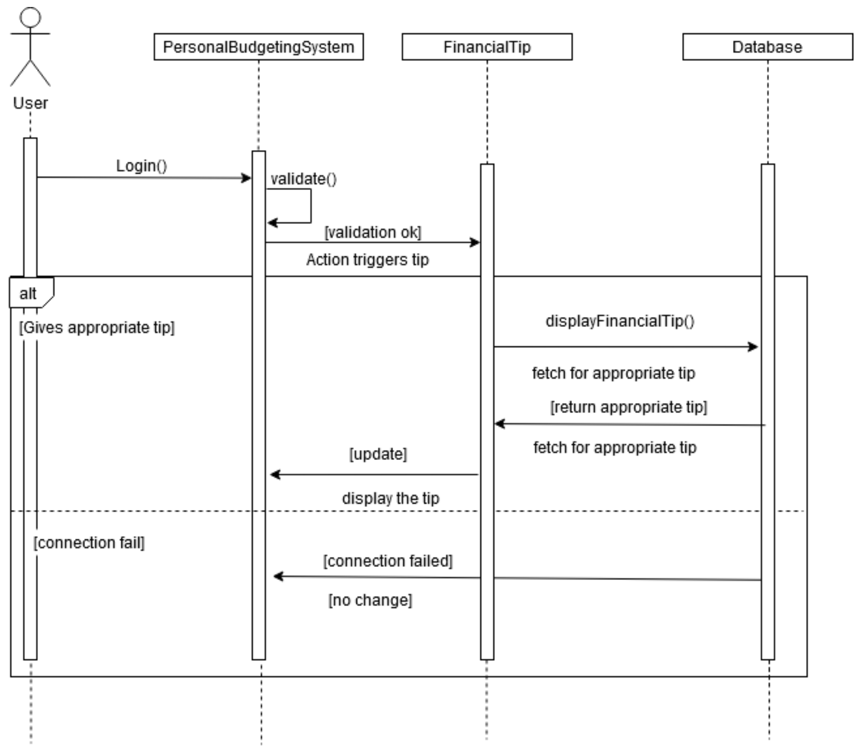
View Personal Goals Data (3.3)



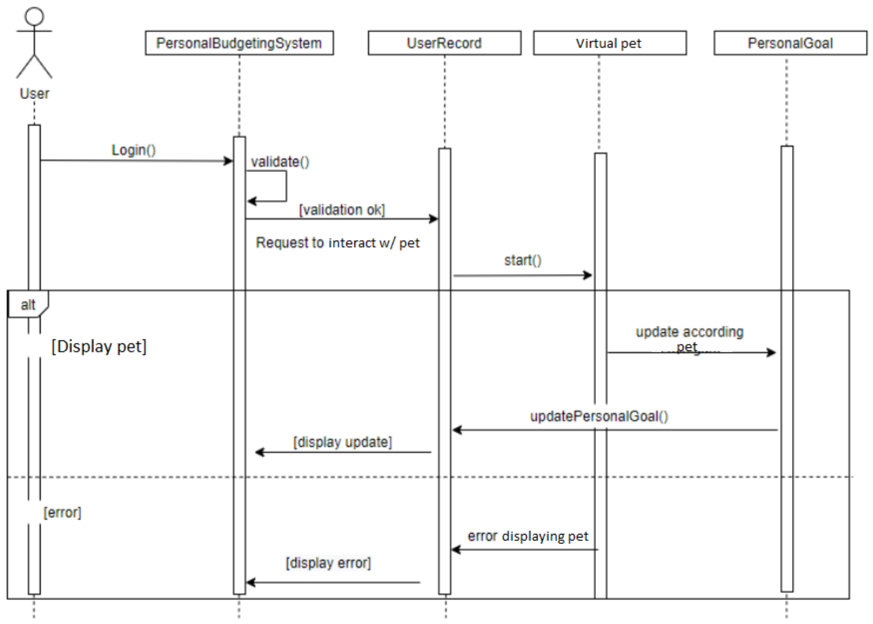
Invest in Personal Goals (3.2)



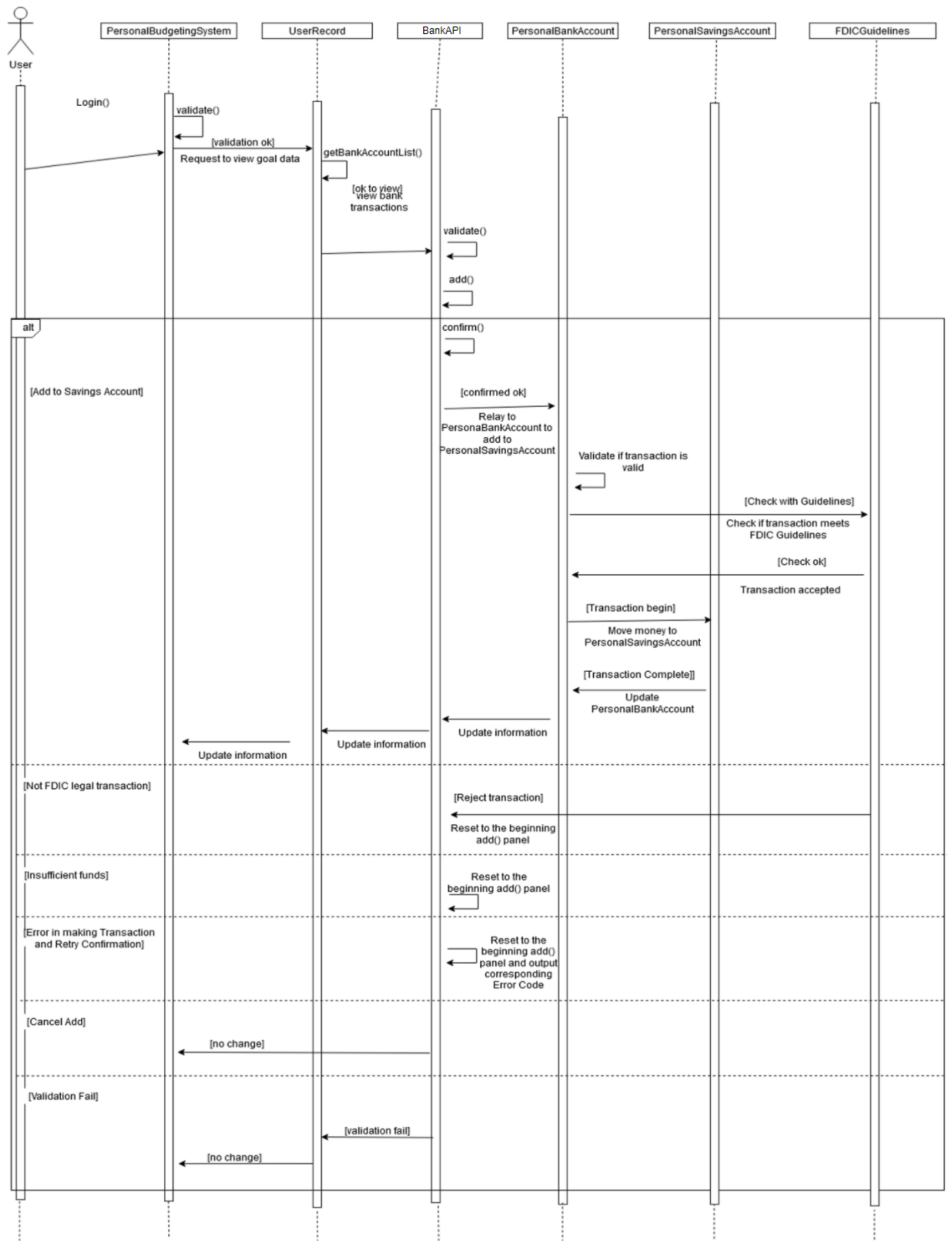
Read Finance Tips (4)



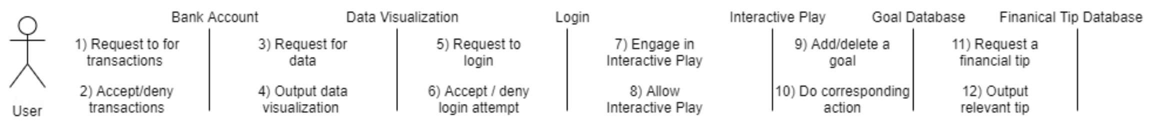
Interact with Interactive Pet (5.1)

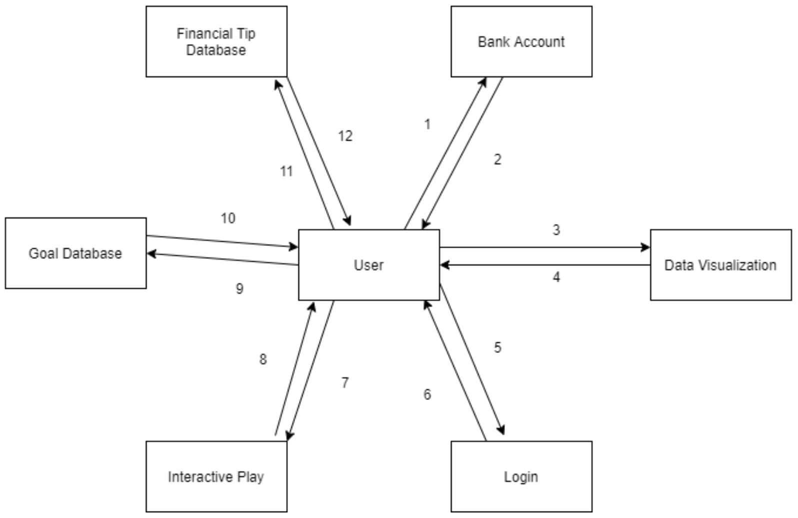


Add to Savings Account (1.4)



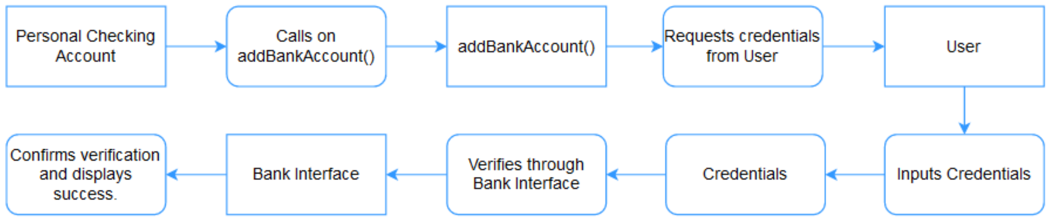
4.3.2 Collaboration Diagrams


User
Bank Account
1) Request to for transactions
2) Accept/deny transactions
Data Visualization
3) Request for data
4) Output data visualization
Login
5) Request to login
6) Accept / deny login attempt
Interactive Play
7) Engage in Interactive Play
8) Allow Interactive Play
Goal Database
9) Add/delete a goal
10) Do corresponding action
Finanical Tip Database
11) Request a financial tip
12) Output relevant tip

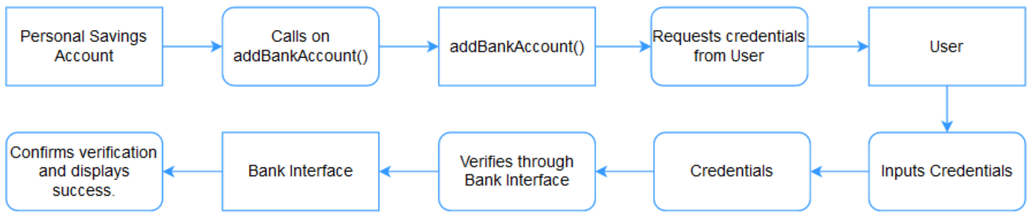


4.3.3 Activity Diagrams

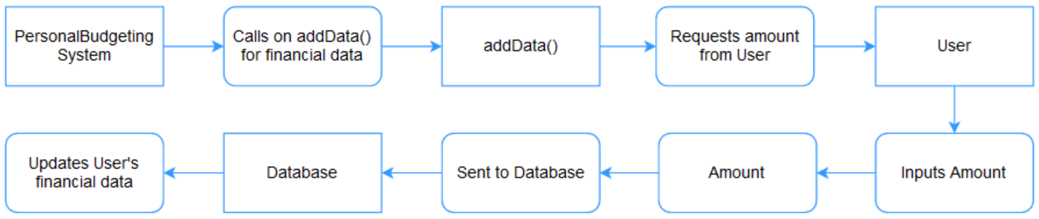
Add Checking Account (1.1)


Personal Checking Account
Calls on addBankAccount()
addBankAccount()
Requests credentials from User
User
Inputs Credentials
Credentials
Verifies through Bank Interface
Bank Interface
Confirms verification and displays success.

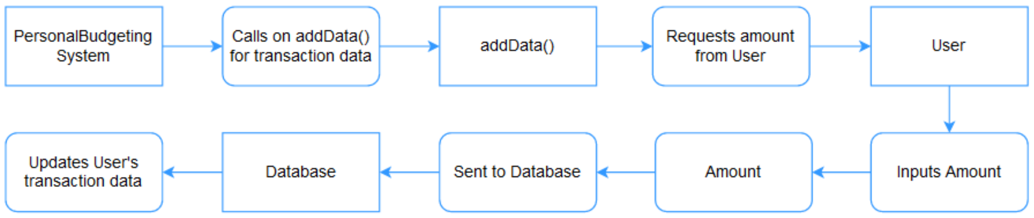
Add Savings Account (1.1)


Personal Savings Account
Calls on addBankAccount()
addBankAccount()
Requests credentials from User
User
Inputs Credentials
Credentials
Verifies through Bank Interface
Bank Interface
Confirms verification and displays success.

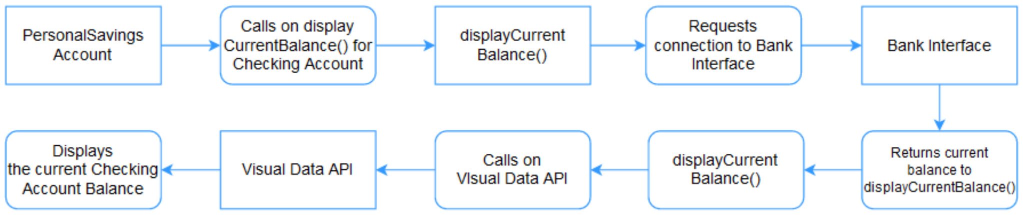
Input Financial Data (1.2)


PersonalBudgeting System
Calls on addData() for financial data
addData()
Requests amount from User
User
Inputs Amount
Amount
Sent to Database
Database
Updates User's financial data

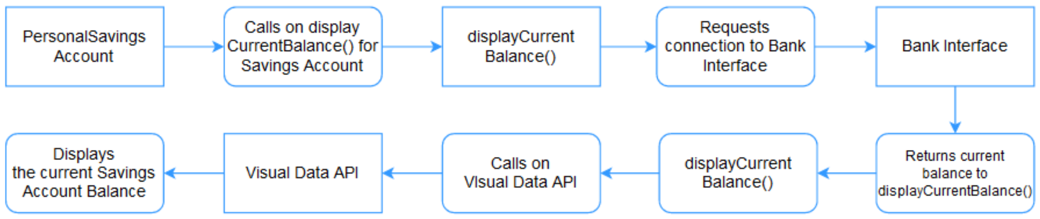
Input Transaction Data (1.2)


PersonalBudgeting System
Calls on addData() for transaction data
addData()
Requests amount from User
User
Inputs Amount
Amount
Sent to Database
Database
Updates User's transaction data

View Checking Account Balance (1.3)

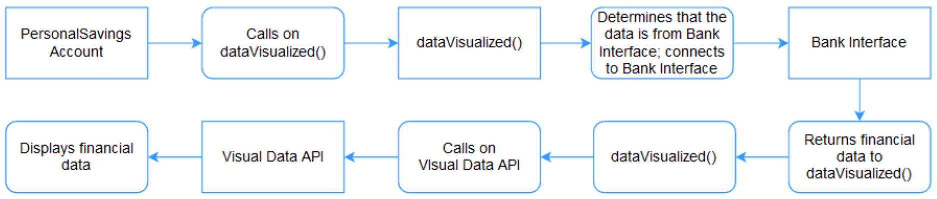

PersonalSavings Account
Calls on display CurrentBalance() for Checking Account
displayCurrent Balance()
Requests connection to Bank Interface
Bank Interface
Returns current balance to displayCurrentBalance()
displayCurrent Balance()
Calls on Visual Data API
Visual Data API
Displays the current Checking Account Balance

View Savings Account Balance (1.3)

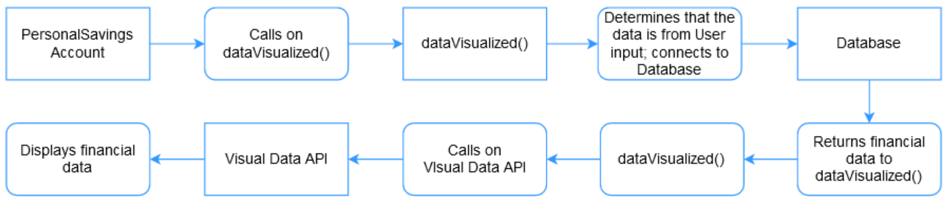

PersonalSavings Account
Calls on display CurrentBalance() for Savings Account
displayCurrent Balance()
Requests connection to Bank Interface
Bank Interface
Returns current balance to displayCurrentBalance()
displayCurrent Balance()
Calls on Visual Data API
Visual Data API
Displays the current Savings Account Balance

View Financial Data (2.1)

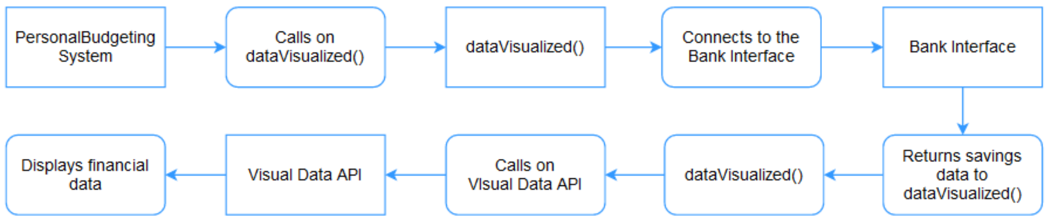
a. If data is from Bank API:


PersonalSavings Account
Calls on dataVisualized()
dataVisualized()
Determines that the data is from Bank Interface; connects to Bank Interface
Bank Interface
Returns financial data to dataVisualized()
dataVisualized()
Calls on Visual Data API
Visual Data API
Displays financial data

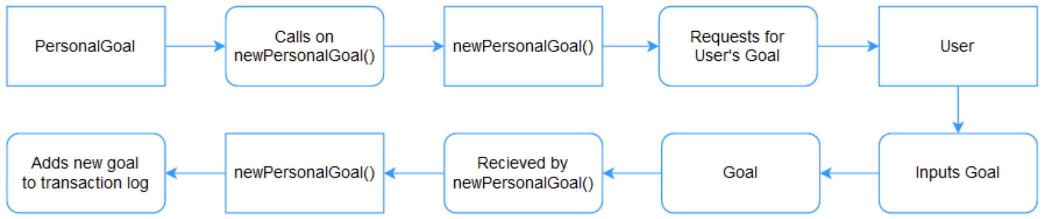
b. If data is from User Interface:


PersonalSavings Account
Calls on dataVisualized()
dataVisualized()
Determines that the data is from User input; connects to Database
Database
Returns financial data to dataVisualized()
dataVisualized()
Calls on Visual Data API
Visual Data API
Displays financial data

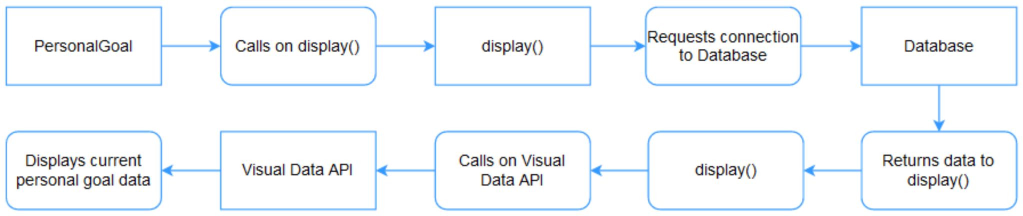
View Savings Account Data (2.1)



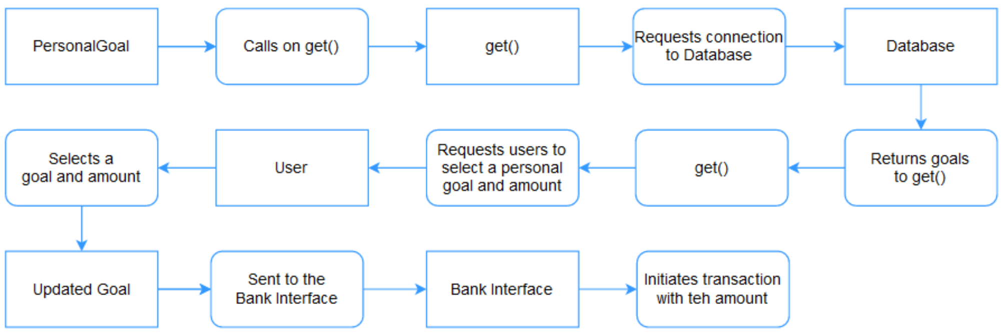
Add/Remove Personal Goals (3.1)



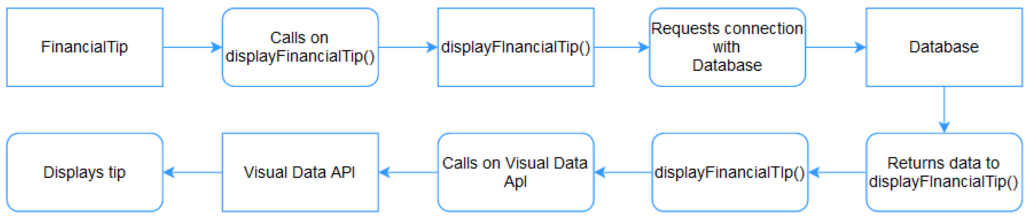
View Personal Goals Data (3.3)



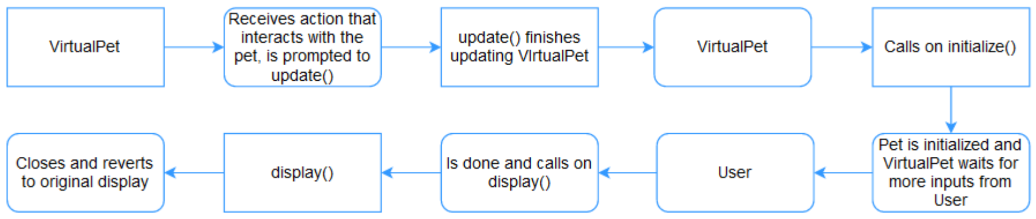
Invest in Personal Goals (3.2)


PersonalGoal
Calls on get()
get()
Requests connection to Database
Database
Returns goals to get()
get()
Requests users to select a personal goal and amount
User
Selects a goal and amount
Updated Goal
Sent to the Bank Interface
Bank Interface
Initiates transaction with teh amount

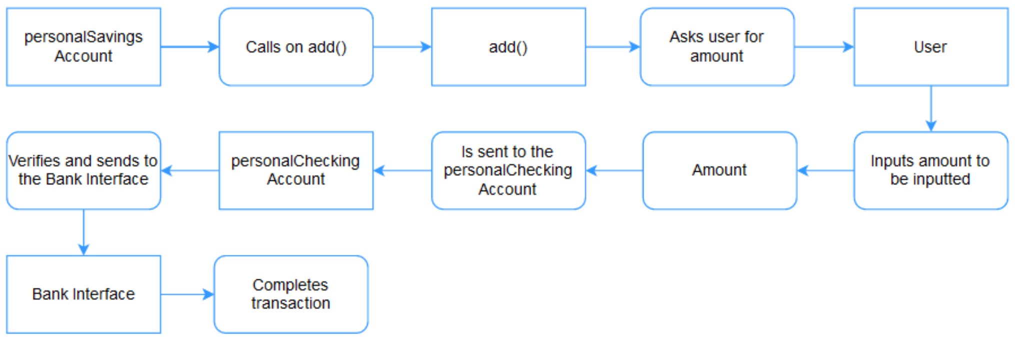
Read Finance Tips (4)



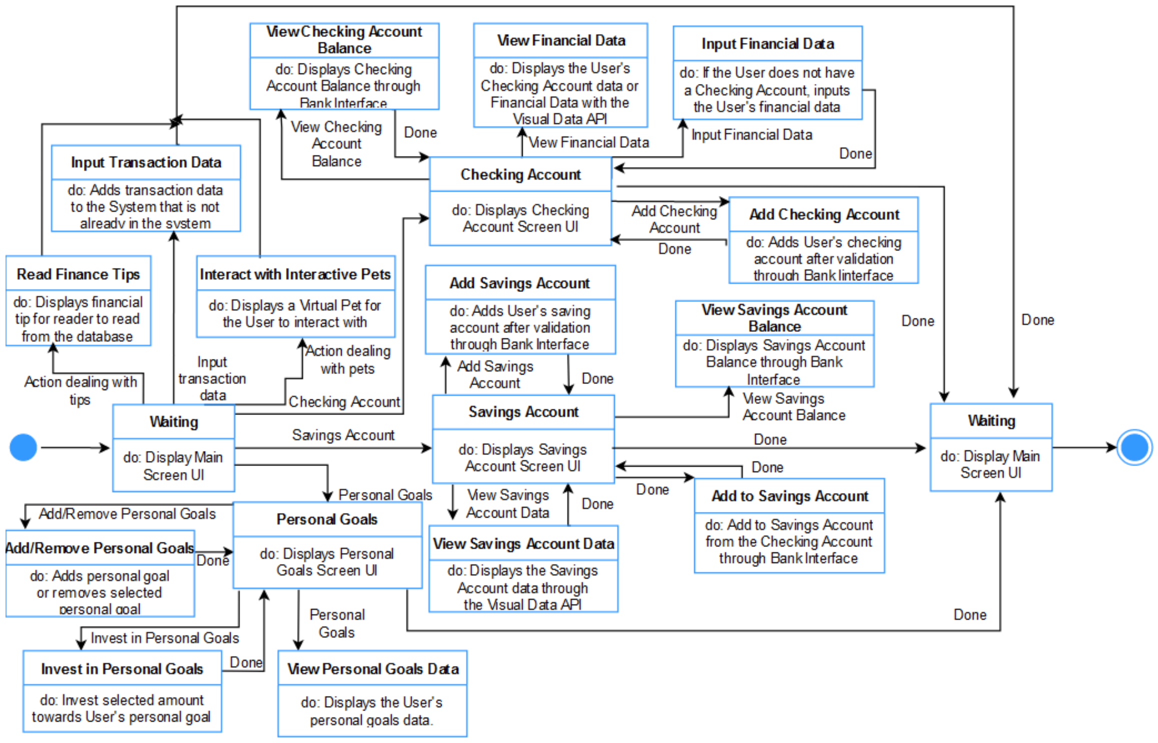
Interact with Interactive Pet (5)


VirtualPet
Receives action that interacts with the pet, is prompted to update()
update() finishes updating VirtualPet
VirtualPet
Calls on initialize()
Pet is initialized and VirtualPet waits for more inputs from User
User
Is done and calls on display()
display()
Closes and reverts to original display

Add to Savings Account (1.4)

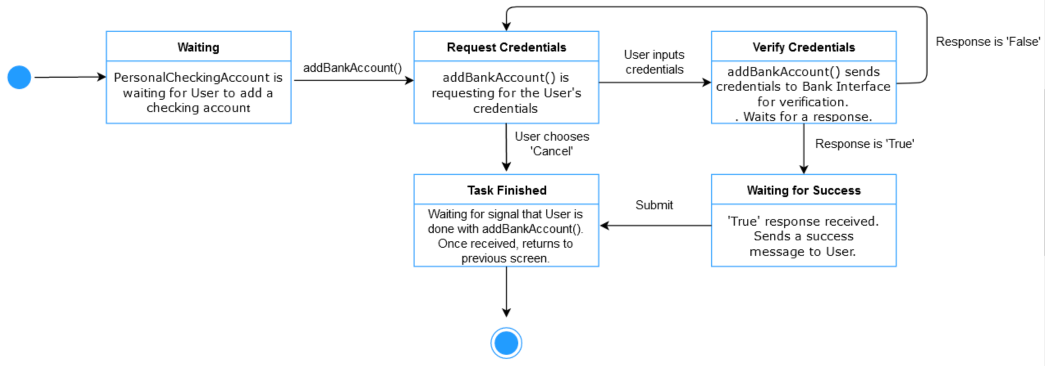


4.3.4 State Diagrams

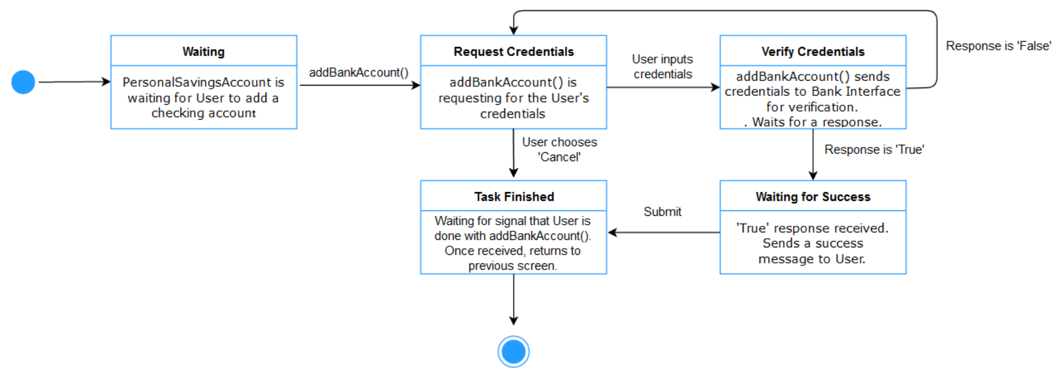

View Checking Account Balance
do: Displays Checking Account Balance through Bank Interface
View Financial Data
do: Displays the User's Checking Account data or Financial Data with the Visual Data API
Input Financial Data
do: If the User does not have a Checking Account, inputs the User's financial data
Input Transaction Data
do: Adds transaction data to the System that is not already in the system
Checking Account
do: Displays Checking Account Screen UI
Add Checking Account
do: Adds User's checking account after validation through Bank Interface
Read Finance Tips
do: Displays financial tip for reader to read from the database
Interact with Interactive Pets
do: Displays a Virtual Pet for the User to interact with
Add Savings Account
do: Adds User's saving account after validation through Bank Interface
View Savings Account Balance
do: Displays Savings Account Balance through Bank Interface
Waiting
do: Display Main Screen UI
Savings Account
do: Displays Savings Account Screen UI
Add to Savings Account
do: Add to Savings Account from the Checking Account through Bank Interface
Personal Goals
do: Displays Personal Goals Screen UI
Add/Remove Personal Goals
do: Adds personal goal or removes selected personal goal
View Savings Account Data
do: Displays the Savings Account data through the Visual Data API
Invest in Personal Goals
do: Invest selected amount towards User's personal goal
View Personal Goals Data
do: Displays the User's personal goals data.
Done
Action dealing with tips
Input transaction data
Action dealing with pets
Checking Account
Savings Account
Personal Goals
Add/Remove Personal Goals
Invest in Personal Goals
View Savings Account Data
View Savings Account Balance
Add Savings Account
Add Checking Account
View Financial Data
Input Financial Data

4.3.5 Event Diagrams

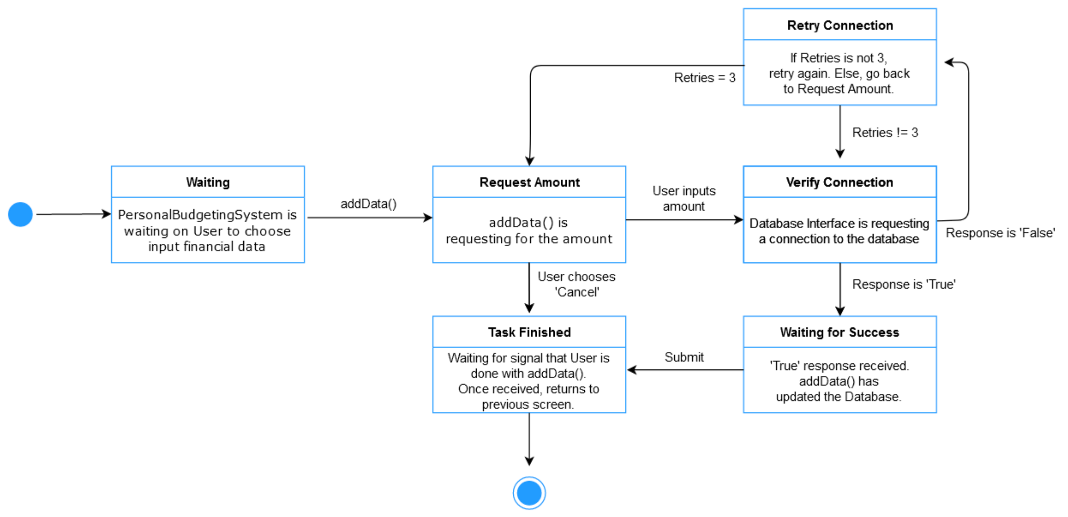
Add Checking Account (1.1)


Waiting
PersonalCheckingAccount is waiting for User to add a checking account
addBankAccount()
Request Credentials
addBankAccount() is requesting for the User's credentials
User inputs credentials
Verify Credentials
addBankAccount() sends credentials to Bank Interface for verification. Waits for a response.
Response is 'False'
User chooses 'Cancel'
Response is 'True'
Task Finished
Waiting for signal that User is done with addBankAccount(). Once received, returns to previous screen.
Submit
Waiting for Success
'True' response received. Sends a success message to User.

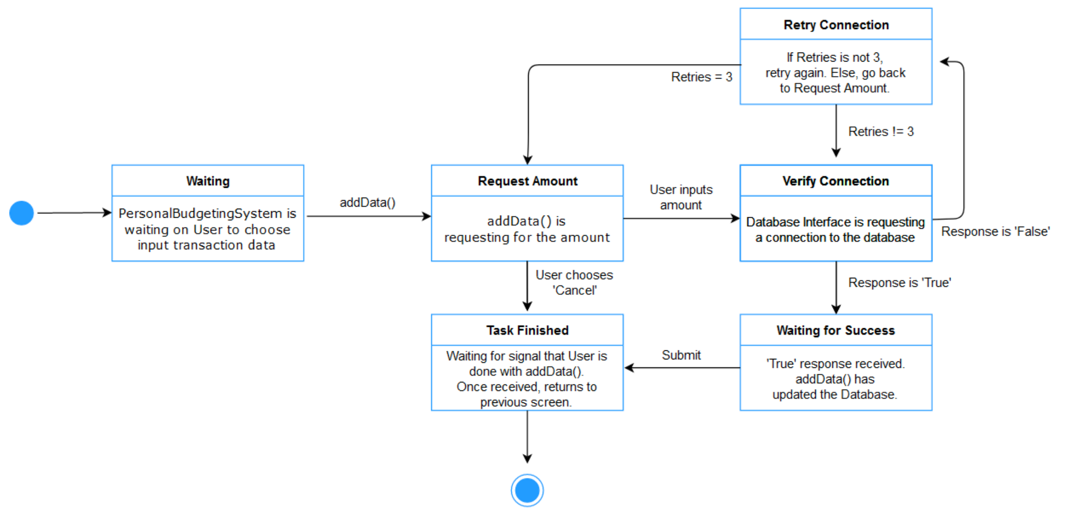
Add Savings Account (1.1)


Waiting
PersonalSavingsAccount is waiting for User to add a checking account
addBankAccount()
Request Credentials
addBankAccount() is requesting for the User's credentials
User inputs credentials
Verify Credentials
addBankAccount() sends credentials to Bank Interface for verification. Waits for a response.
Response is 'False'
Response is 'True'
User chooses 'Cancel'
Task Finished
Waiting for signal that User is done with addBankAccount(). Once received, returns to previous screen.
Submit
Waiting for Success
'True' response received. Sends a success message to User.

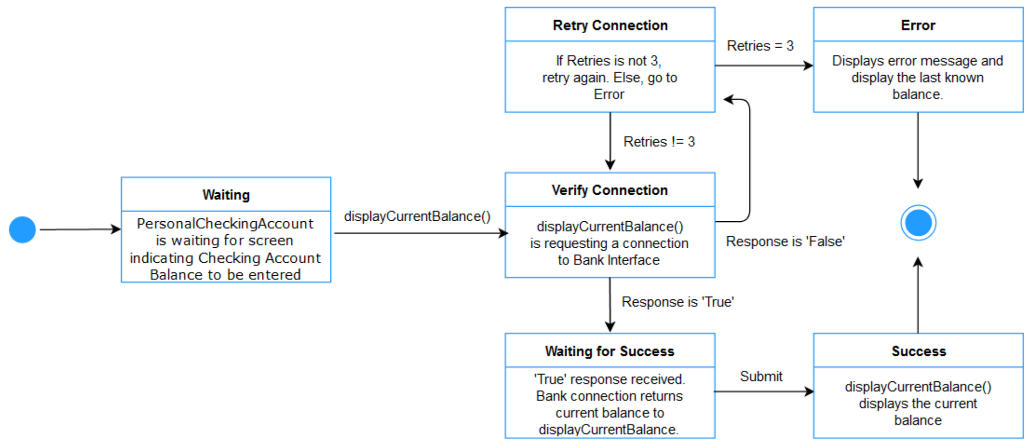
Input Financial Data (1.2)


Retry Connection
If Retries is not 3, retry again. Else, go back to Request Amount.
Retries = 3
Retries != 3
Waiting
PersonalBudgetingSystem is waiting on User to choose input financial data
addData()
Request Amount
addData() is requesting for the amount
User inputs amount
Verify Connection
Database Interface is requesting a connection to the database
Response is 'False'
User chooses 'Cancel'
Response is 'True'
Task Finished
Waiting for signal that User is done with addData(). Once received, returns to previous screen.
Submit
Waiting for Success
'True' response received. addData() has updated the Database.

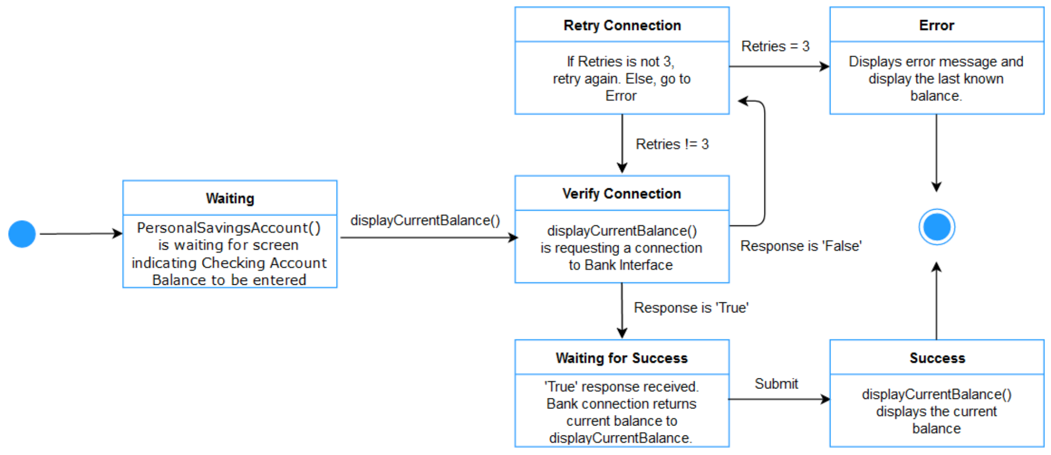
Input Transaction Data (1.2)


Retry Connection
If Retries is not 3, retry again. Else, go back to Request Amount.
Retries = 3
Retries != 3
Waiting
PersonalBudgetingSystem is waiting on User to choose input transaction data
addData()
Request Amount
addData() is requesting for the amount
User inputs amount
Verify Connection
Database Interface is requesting a connection to the database
Response is 'False'
User chooses 'Cancel'
Response is 'True'
Task Finished
Waiting for signal that User is done with addData(). Once received, returns to previous screen.
Submit
Waiting for Success
'True' response received. addData() has updated the Database.

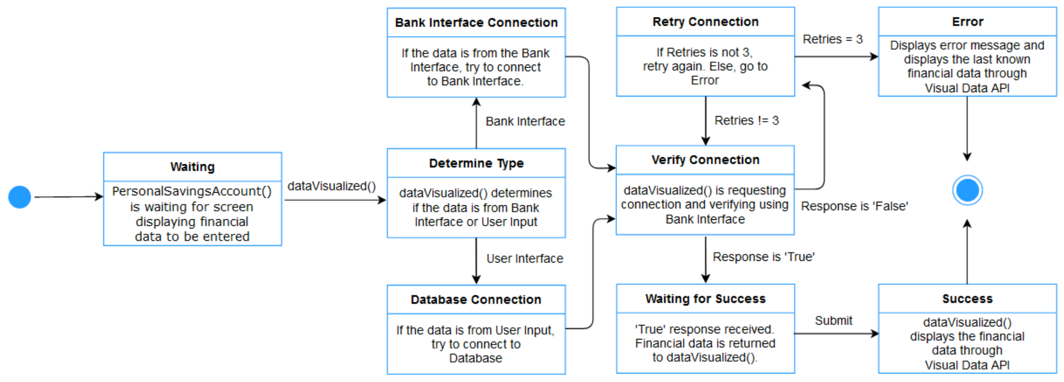
View Checking Account Balance (1.3)


Waiting
PersonalCheckingAccount is waiting for screen indicating Checking Account Balance to be entered
displayCurrentBalance()
Verify Connection
displayCurrentBalance() is requesting a connection to Bank Interface
Response is 'False'
Retry Connection
If Retries is not 3, retry again. Else, go to Error
Retries != 3
Retries = 3
Error
Displays error message and display the last known balance.
Response is 'True'
Waiting for Success
'True' response received. Bank connection returns current balance to displayCurrentBalance.
Submit
Success
displayCurrentBalance() displays the current balance

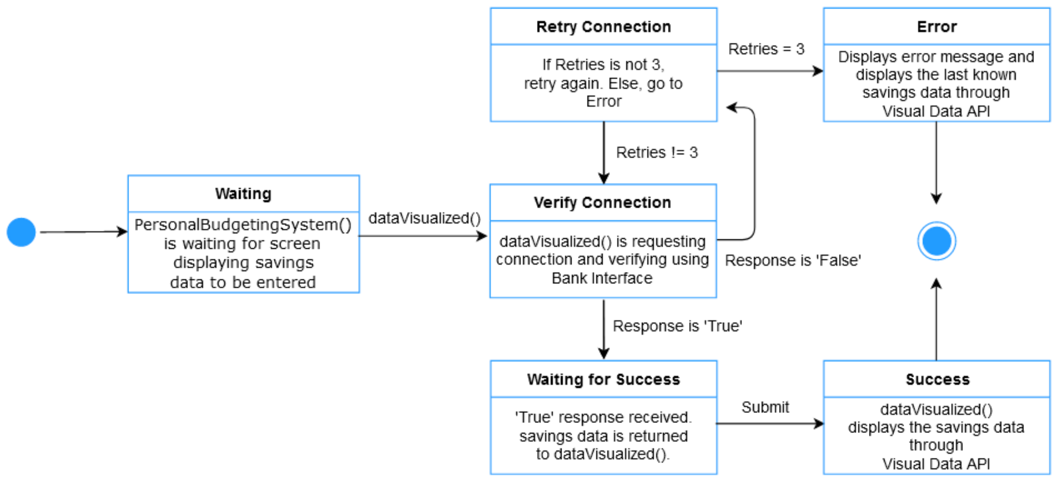
View Savings Account Balance (1.3)


Waiting
PersonalSavingsAccount() is waiting for screen indicating Checking Account Balance to be entered
displayCurrentBalance()
Verify Connection
displayCurrentBalance() is requesting a connection to Bank Interface
Response is 'False'
Retry Connection
If Retries is not 3, retry again. Else, go to Error
Retries != 3
Retries = 3
Error
Displays error message and display the last known balance.
Response is 'True'
Waiting for Success
'True' response received. Bank connection returns current balance to displayCurrentBalance.
Submit
Success
displayCurrentBalance() displays the current balance

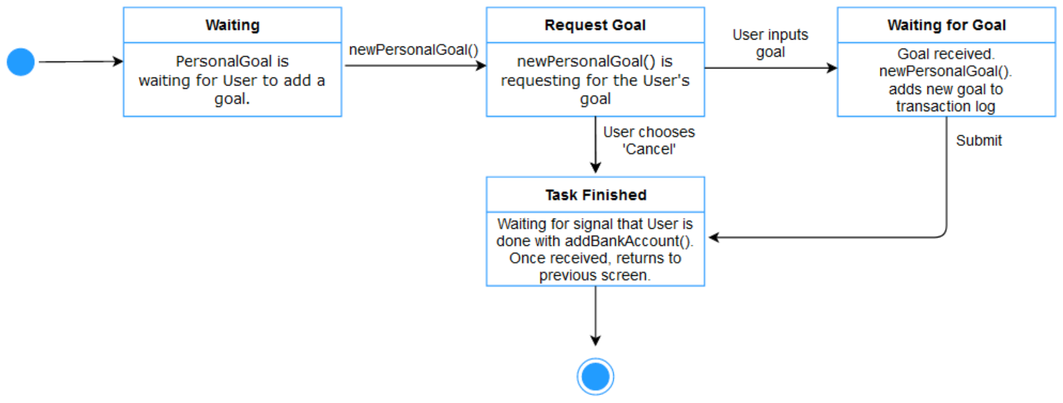
View Financial Data (2.1)


Waiting
PersonalSavingsAccount() is waiting for screen displaying financial data to be entered
dataVisualized()
Determine Type
dataVisualized() determines if the data is from Bank Interface or User Input
Bank Interface
Bank Interface Connection
If the data is from the Bank Interface, try to connect to Bank Interface.
User Interface
Database Connection
If the data is from User Input, try to connect to Database
Verify Connection
dataVisualized() is requesting connection and verifying using Bank Interface
Response is 'False'
Retry Connection
If Retries is not 3, retry again. Else, go to Error
Retries != 3
Retries = 3
Error
Displays error message and displays the last known financial data through Visual Data API
Response is 'True'
Waiting for Success
'True' response received. Financial data is returned to dataVisualized().
Submit
Success
dataVisualized() displays the financial data through Visual Data API

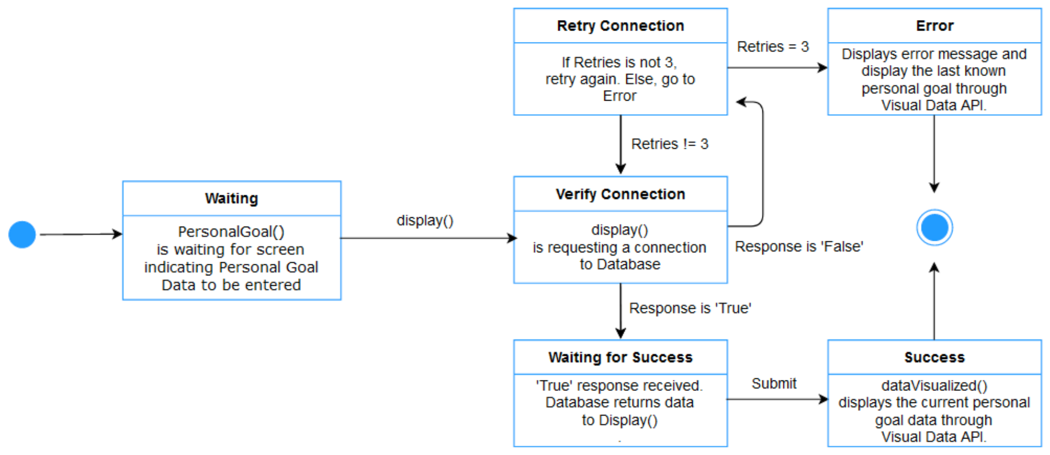
View Savings Account Data (2.1)


Retry Connection
If Retries is not 3, retry again. Else, go to Error
Retries = 3
Error
Displays error message and displays the last known savings data through Visual Data API
Retries != 3
Waiting
PersonalBudgetingSystem() is waiting for screen displaying savings data to be entered
dataVisualized()
Verify Connection
dataVisualized() is requesting connection and verifying using Bank Interface
Response is 'False'
Response is 'True'
Waiting for Success
'True' response received. savings data is returned to dataVisualized().
Submit
Success
dataVisualized() displays the savings data through Visual Data API

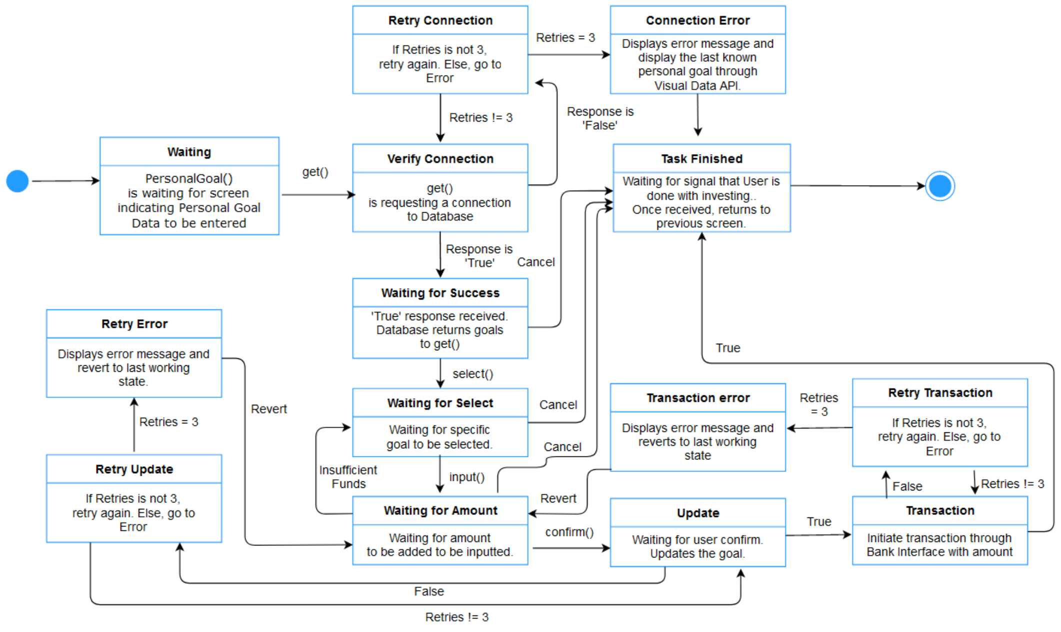
Add/Remove Personal Goals (3.1)


Waiting
PersonalGoal is waiting for User to add a goal.
newPersonalGoal()
Request Goal
newPersonalGoal() is requesting for the User's goal
User inputs goal
Waiting for Goal
Goal received. newPersonalGoal(). adds new goal to transaction log
User chooses 'Cancel'
Submit
Task Finished
Waiting for signal that User is done with addBankAccount(). Once received, returns to previous screen.

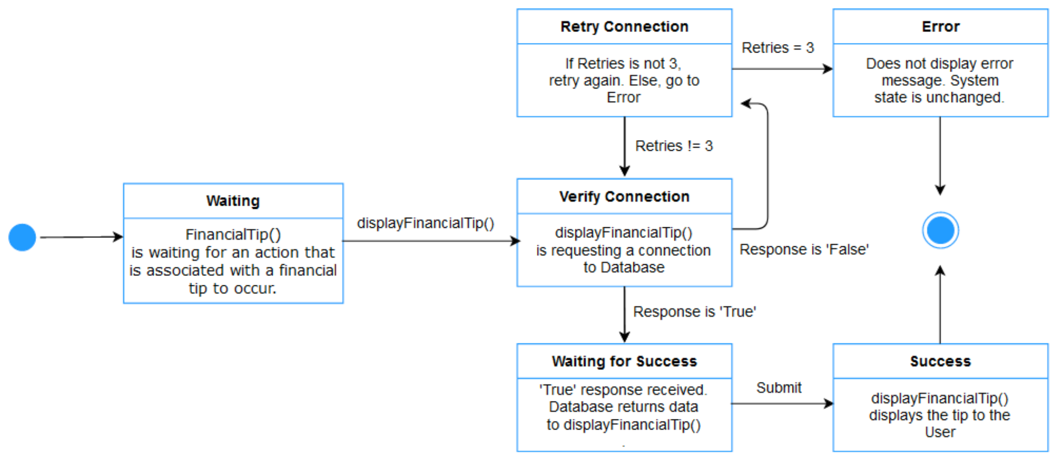
View Personal Goals Data (3.3)


Waiting
PersonalGoal() is waiting for screen indicating Personal Goal Data to be entered
display()
Retry Connection
If Retries is not 3, retry again. Else, go to Error
Retries = 3
Error
Displays error message and display the last known personal goal through Visual Data API.
Retries != 3
Verify Connection
display() is requesting a connection to Database
Response is 'False'
Response is 'True'
Waiting for Success
'True' response received. Database returns data to Display()
Submit
Success
dataVisualized() displays the current personal goal data through Visual Data API.

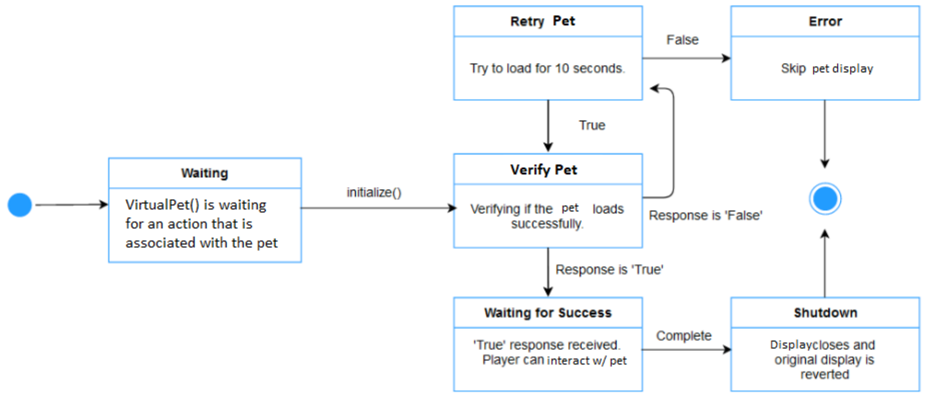
Invest in Personal Goals (3.2)


Waiting
PersonalGoal() is waiting for screen indicating Personal Goal Data to be entered
get()
Retry Connection
If Retries is not 3, retry again. Else, go to Error
Retries = 3
Connection Error
Displays error message and display the last known personal goal through Visual Data API.
Retries != 3
Response is 'False'
Verify Connection
get() is requesting a connection to Database
Task Finished
Waiting for signal that User is done with investing.. Once received, returns to previous screen.
Response is 'True'
Cancel
Waiting for Success
'True' response received. Database returns goals to get()
True
Retry Error
Displays error message and revert to last working state.
Revert
select()
Waiting for Select
Waiting for specific goal to be selected.
Cancel
Transaction error
Displays error message and reverts to last working state
Retries = 3
Retry Transaction
If Retries is not 3, retry again. Else, go to Error
Retries = 3
Retry Update
If Retries is not 3, retry again. Else, go to Error
Insufficient Funds
input()
Cancel
False
Retries != 3
Revert
Waiting for Amount
Waiting for amount to be added to be inputted.
confirm()
Update
Waiting for user confirm. Updates the goal.
True
Transaction
Initiate transaction through Bank Interface with amount
False
Retries != 3

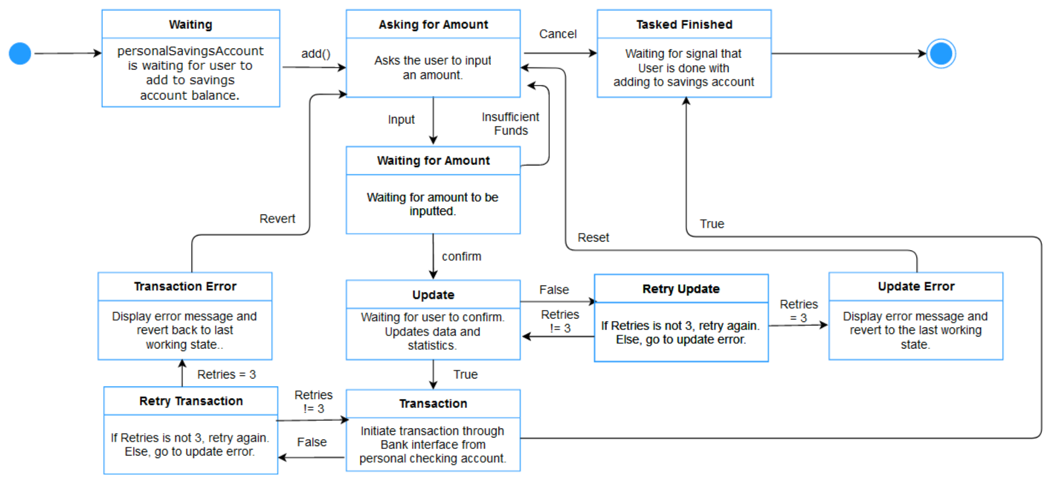
Display Finance Tips (4)


Retry Connection
If Retries is not 3, retry again. Else, go to Error
Retries = 3
Error
Does not display error message. System state is unchanged.
Retries != 3
Waiting
FinancialTip() is waiting for an action that is associated with a financial tip to occur.
displayFinancialTip()
Verify Connection
displayFinancialTip() is requesting a connection to Database
Response is 'False'
Response is 'True'
Waiting for Success
'True' response received. Database returns data to displayFinancialTip()
Submit
Success
displayFinancialTip() displays the tip to the User

Interact with Interactive Pet (5.1)


Retry Pet
Try to load for 10 seconds.
False
Error
Skip pet display
True
Waiting
VirtualPet() is waiting for an action that is associated with the pet
initialize()
Verify Pet
Verifying if the pet loads successfully.
Response is 'False'
Response is 'True'
Waiting for Success
'True' response received. Player can interact w/ pet
Complete
Shutdown
Displaycloses and original display is reverted

Add to Savings Account (1.4)


Waiting
personalSavingsAccount is waiting for user to add to savings account balance.
add()
Asking for Amount
Asks the user to input an amount.
Cancel
Tasked Finished
Waiting for signal that User is done with adding to savings account
Input
Insufficient Funds
Waiting for Amount
Waiting for amount to be inputted.
Revert
Reset
True
confirm
Transaction Error
Display error message and revert back to last working state..
Update
Waiting for user to confirm. Updates data and statistics.
False
Retries != 3
Retry Update
If Retries is not 3, retry again. Else, go to update error.
Retries = 3
Update Error
Display error message and revert to the last working state.
Retries = 3
True
Retry Transaction
If Retries is not 3, retry again. Else, go to update error.
Retries != 3
False
Transaction
Initiate transaction through Bank interface from personal checking account.

4.4 Concept of Execution

Upon the launch of the application through a connection to the web server, the flows through the system are as follows:

1. User login (8.1)
2. Add checking account (1.1) and/or input financial data (1.2)
    a. View checking account balance (1.3)
    b. View financial data (2.1)
3. Input transaction data (1.2)
    a. View financial data (2.1)
4. Add savings account (1.1)
    a. View savings account balance (1.3)
    b. View savings account data (2.1)
    c. Add to savings account (1.4)
5. Add/Remove personal goals (3.1)
    a. Invest in person goals (3.2)
    b. View personal goals data (3.3)

6. Interact with interactive pet (5.1)
7. Read finance tips (4)

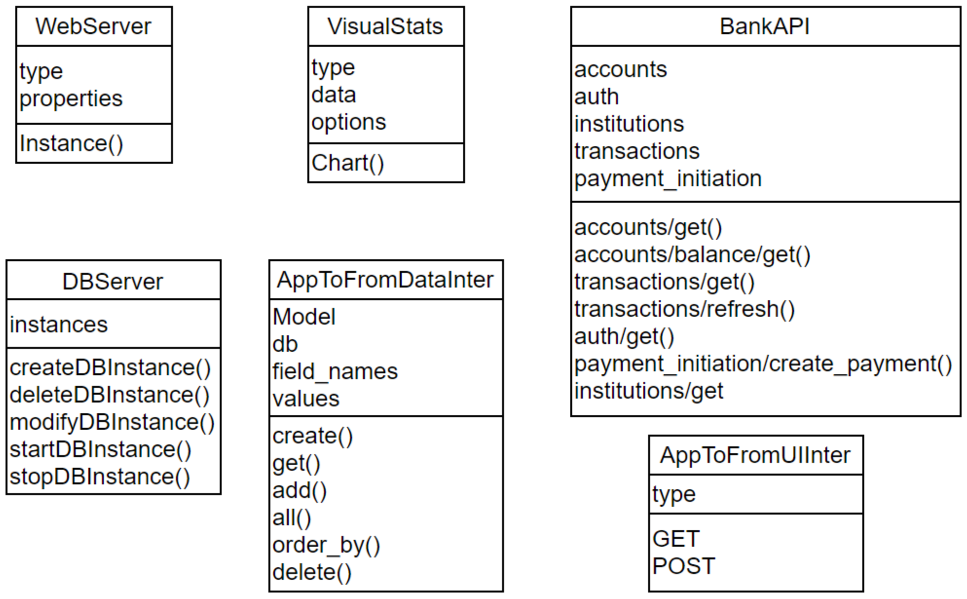
4.5 Interface Design

4.5.1 Unique Identifier of Interface

The following interfaces will be used in the system:

- Web Server as WebServer
- Bank API as BankAPI
- Visual Statistics API as VisualStats
- Database Service as DBService
- Application to/from UI Interface as AppToFromUIInter
- Application to/from Database Interface as AppToFromDataInter

4.5.2 Interface Diagrams



5. IMPLEMENTATION ARCHITECTURE

5.1 All Active and Passive Classes Assigned to Components

Not required.

5.2 Diagrams of Physical Packaging of Logical Components

Not required.

6. DEPLOYMENT ARCHITECTURE

6.1 Physical Deployment Architecture Diagram

Amazon Web Service (AWS) will be used to deploy the web application. As such, the physical architecture/hardware will be under the management of Amazon; the team will have no say in how Amazon configures its physical infrastructure.

In particular, AWS Elastic Compute Cloud (EC2) will be used as the web browser interface and Amazon Relational Database Service (RDS) will be used for the database server. Furthermore, Django Chart.js will be used as the visual statistics API and Plaid API will be used for the Bank API.

7. DICTIONARIES

See section 13.1.

8. SOFTWARE ITEM COMPUTER RESOURCE UTILIZATION

Memory: Utilization should not exceed 80%. If exceeded, a notification should be broadcasted to Operations saying that memory is being exhausted and to add more memory.

Disk Storage: Utilization should not exceed 80%. If exceeded, a notification should be broadcasted to Operations saying that memory is being exhausted and to add more memory.

Network: Utilization should not exceed 40% to prevent collisions on the network.

CPU: Utilization should not exceed 40% for the active system, 40% for the backup, 10% for the operating system, and 10% for the resources so as to not run out of CPU power.

9. REQUIREMENTS TRACEABILITY

9.1 Software Component-Level Requirements Traceability

The requirements will be kept track by use of a unique identifier and a specification. For example, a requirement will have the tag 1, and its specification would be labeled as 1.1, 1.2, etc. The requirement tag should be traceable through the development and backwards.

Any code written for the software will be labeled to trace what requirement the code satisfies. All requirements should be mentioned alongside with its identifier in the document to allow for easy traceability.

10. SYSTEM DESIGN TESTING

The SQA testing process will initially have developers non-statically test their code by using the desk check. Developers will manually and personally check if their portion of code works and meets the needs of the consumer. This process will also involve the deliberation of verification and validation of the product, of which the entire development team will answer the questions of whether we made the product right and is this the right product to be made for the user. Throughout the development process, teammates not responsible for a specific portion of the code will inspect other teammate’s work in order to help find problems with the logic or other such defects. Inspections will include looking to see whether the work matches with the requirements specification, software architecture, and UML design models while being neat code and does not damage any part of the system, such as the database schema. The program will also be checked to see if it is incomplete or complies with standards and is maintainable; the former will be tested with special tests to be determined and the latter will be tested by asking the consumer whether it suits their needs.

For the testing process of the software program, test cases will be designed to test the objective of the test. For example, if account creation is to be tested, the test cases will include having passwords with invalid characters or trying to tamper with the account databases through means such as SQL injection. Test data for the cases will be prepared and the program will run with those test data. The results will be compared with the test cases and any desired output from the data to see if there is a match. A test report will be made either formally through documentation or informally and a conclusion will be reached regarding the test.

The product testing phase will involve testing the use cases and other such scenarios. The login system will be tested to make sure users can create their own accounts and by inputting their username and password will have access to only their account and no other account. After accessing their accounts, users will be able to view data corresponding to only their account. Tests will be conducted so that the correct information and other such as any forms of data visualization can be viewable to the user. Being able to add bank account information or data to the account will be tested so that only their bank account or data will be reflected on the application. The interactive play will be tested for being fully playable and fun, Defects for any of the scenarios above will be reported to the development team to fix and pushed to a future prototype.

The acceptance testing process will start by developers as a team defining the acceptance criteria. The criteria are then tested, and developers will plan the acceptance testing. The plan will be tested, and the next step will be serving acceptance tests. The acceptance tests will be run, and the test results will be examined, and a report will be made. Developers will then analyze the report and either accept or reject the system.

The quality assurance team will fully understand the software process, the software product as specified in the requirements specification, the software project plan, supporting plans, and any standards, policies, or guidelines to which the process or the product must adhere. Based on these understandings, the quality assurance team will conduct periodic analysis, inspections, reviews, audits, and assessments on all teams. For example, to ensure that the system conforms to its specification and meets expectations of the system customer, the quality assurance team will conduct milestone reviews at each stage of the software process with the development team. Additionally, the quality assurance team will create a test plan, test scenarios with expected outputs, execute the tests, and report any defects back to the development team for fixes.

Project reviews and audits will be conducted on a periodic basis and involve checking the quality of project deliverables by checking software, its documentation, and records of the process to find violations of standards as well as errors and omissions. The different types of reviews as well as the schedule, resources, methods, and procedures in conducting them are as follows:

1. Program inspections and developer peer reviews involve a variety of team members with different backgrounds working together to find bugs in the program. Tasks such as incomplete versions of the system and representations such as UML models will be delegated. A line-by-line review of the program source code will be performed, and a checklist of common programming errors will be referenced to help find bugs.

2. Formal project technical reviews are conducted when there is an imminent problem with the project and negotiation, or when risk mitigation actions fail. In this scenario, an alternative approach to allow for the continuation of the project will be found—which includes whether or not the new approach is still aligned with the customer's goals. If a solution can not be found, the ultimate decision of this review may be to cancel the project.

3. Walkthroughs are a part of the review meeting phase and consists of the author of the program or document will go over it with the review team, with one team member sharing the review, and another formally recording all decisions made. Additionally, all actions agreed during the review should be signed by the review chair.

4. Management reviews are a part of the post-review activities phase and will be conducted if it is determined that the problems discovered need more resources.

Audits will be performed through the use of auditing systems placed strategically in code to detect any terms of a license.

11. RATIONALE

None.

12. NOTES

None.

13. APPENDICES

13.1 Dictionaries

Class

Name	**Description**	**Methods**	**Attributes**

PersonalBudgetingSystem	Represents the entire personal budgeting system application.	addUser(), updateUser(), deleteUser(), addBank(), updateBank(), deleteBank(), addMinigame(), updateMinigame(), deleteMinigame(), checkUpStatus(), changeUpStatus()	userList, bankList, minigamesList, upStatus, state
PersonalBankAccount	Represents a personal bank account	updateBankAccount(), addBankAccount(), deleteBankAccount(), displayCurrentBalance(), displayBalanceLog(), displayTransactionLog(), displayAccountInfo(), displayFDICGuidelines(), verifyFollowingFDICGuidelines()	accountNumber, accountType, accountName, bank, user, balanceLog, transactionLog, fdicGuidelines
Transaction	An exchange of money between two parties	getDate(), getTime(), getLocation(), getType(), getAmount(), getReceiver(), displayTransaction(), newTransaction()	date, time, location, type, amount, receiver
Balance	Amount of money in User's possession	getDate(), getTime(), getType(), getAmount(), displayBalance()	Date, time, type, amount
InteractivePet	Facilitates the interactive pet to reward the User for good practices.	getName(), getAge(), initialize(), processInput(), update(), render(), shutdown(), start()	petName, petAge, interactionList

Personal Goal	Facilitates User's access to interact with their personal financial goals.	newPersonalGoal(), deletePersonalGoal, updatePersonalGoal(), changePriority(), getName(), getTotalAmount(), GetStartDate(), getEndDate(), getPriority(), addTransaction()	name, totalAmount, currentAmount, startDate, endDate, priority, transactionLog
FinancialTip	Catalogue of financial tips	addFinancialTip(), deleteFinancialTip(), getType(). getUseLocation()	type, tip, useLocations
User	A representation of the user and their respective information	getName(), getUserName(), getBankAccountList(), getPersonalGoalList()	name, username, password, bankAccountList, personalGoalList
FDICGuidelines	Contains a list of rules and guidelines outlined by the FDIC regarding bank accounts and transactions	newFDICGuidelines(), updateFDICGuidelines(), deleteFDICGuidelines(), displayBank(), displayAccountTypes(), displayGuidelines()	bank, accountType, guidelines

Methods

Name	**Description**	**Class**	**Arguments**
addUser()	Add a user to the system database.	PersonalBudgetingSystem	username, password, email
updateUser()	Update a user’s personal information in the system database.	PersonalBudgetingSystem	username
deleteUser()	Remove a user from the system database.	PersonalBudgetingSystem	username
addBank()	Add a bank to the system database.	PersonalBudgetingSystem	name, id
updateBank()	Update a bank’s information in the system database.	PersonalBudgetingSystem	id, changeType, changeInput
deleteBank()	Delete a bank’s information from the system database.	PersonalBudgetingSystem	id
addInteractivePet()	Add an interactive pet to the system database.	PersonalBudgetingSystem	name, age, id, runLocation
updateInteractivePet ()	Update an interactive pet in the system database.	PersonalBudgetingSystem	id
deleteInteractivePet ()	Delete an interactive pet from the system database.	PersonalBudgetingSystem	id
checkUpStatus()	Check if the system is up and available for use.	PersonalBudgetingSystem	null

changeUpStatus()	Change the system’s availability.	PersonalBudgetingSystem	status
updateBankAccount()	Update the bank account information	PersonalBankAccount	null
addBankAccount()	Add new bank account	PersonalBankAccount	name, id
deleteBankAccount()	Delete bank account	PersonalBankAccount	null
displayCurrentBalance()	Display current bank account balance	PersonalBankAccount	null
displayBalanceLog()	Display bank account balance log	PersonalBankAccount	null
displayTransactionLog()	Display bank account transaction log	PersonalBankAccount	null
displayAccountInfo()	Display bank account information	PersonalBankAccount	null
displayFDICGuidelines()	Display FDIC guidelines for the bank account	PersonalBankAccount	null
verifyFollowingFDICGuidelines()	Verify FDIC guidelines are followed	PersonalBankAccount	null
getDate()	Return date of the transaction	Transaction	null
getTime()	Return exact time of transaction (hours/minutes)	Transaction	null
getLocation()	Return location transaction took place	Transaction	null
getType()	Return type of transaction(withdrawal, deposit)	Transaction	null

getAmount()	Return amount transferred	Transaction	null
getReceiver()	Return who received the money	Transaction	null
displayTransaction()	Display attributes of transaction	Transaction	null
newTransaction()	Create a new transaction	Transaction	null
getDate()	Return date of access	Balance	null
getTime()	Return time of access	Balance	null
getType()	Return type of balance(savings, checking, etc)	Balance	null
getAmount()	Return amount of balance	Balance	null
displayBalance()	Display attributes of balance	Balance	null
getName()	Return name of interactive pet	InteractivePet	null
getAge()	Return age of interactive pet	InteractivePet	null
initialize()	Initialize the game	InteractivePet	null
processInput()	Process input from User	InteractivePet	null
update()	Update game state after every tick	InteractivePet	null
render()	Render changed graphics after updates	InteractivePet	null
shutdown()	Close the game	InteractivePet	null

start()	Begin the game	InteractivePet	null
newPersonalGoal()	Adds new personal goal	PersonalGoal	null
deletePersonalGoal ()	Delete a selected personal goal	PersonalGoal	null
updatePersonalGoal()	Update personal goals after change	PersonalGoal	null
changePriority()	Change the priority of the selected goal	PersonalGoal	null
getName()	Return name of selected personal goal	PersonalGoal	null
getTotalAmount()	Returns total amount invested in goal	PersonalGoal	int
getCurrentAmount ()	Returns current amount invested in goal	PersonalGoal	null
getStartDate()	Returns start date of selected personal goal	PersonalGoal	null
getEndDate()	Returns end date of selected personal goal	PersonalGoal	null
getPriority()	Returns the priority of selected personal goal	PersonalGoal	null
addTransaction()	Invest in the selected personal goal	PersonalGoal	null
addFinancialTip()	Add a new financial tip	FinancialTip	String (unknown amount for this release)
deleteFinancialTip()	Delete an existing financial tip	FinancialTip	null

getType()	Get the type of the financial tip	FinancialTip	null
displayFinancialTip()	Display the financial tip	FinancialTip	null
getTip()	Return the financial tip as an object	FinancialTip	null
getUseLocation()	Return when/where the financial tip will be displayed	FinancialTip	null
getName()	Return the name of the user	User	null
getUserName()	Return the username of the user	User	null
getBankAccountList()	Return a list of the user's bank accounts	User	null
getPersonalGoalsList()	Return a list of the user's personal goals	User	null
newFDICGuidelines()	Create a new guideline	FDICGuidelines	String (unknown amount for this release)
updateFDICGuidelines()	Update a guideline	FDICGuidelines	String (unknown amount for this release)
deleteFDICGuidelines()	Delete the existing guideline	FDICGuidelines	null
displayBank()	Display the bank and its corresponding information	FDICGuidelines	null

displayAccountType()	Display the type of account	FDICGuidelines	null
displayGuidelines()	Display the guidelines	FDICGuidelines	null

Attributes

Name	**Description**	**C/S**				**R/W**
userList	List of users in the system	Simple	userRecord pointer list	64 bytes		R
bankList	List of banks in the system	Simple	Institution pointer list	64 bytes		R
interactivePetList	List of interactive pets in the system	Simple	InteractivePet pointer list	64 bytes		R
upStatus	Up status of system	Simple	boolean	1 bit		R/W
state	Current system state	Complex	currentUserRecord			R/W
accountNumber	Unique account number	Simple	Integer	4 bytes		R/W
accountType	Account type	Simple	String	8 bytes		R/W
accountName	Account name		String	8 bytes		R/W
user	Pointer to the current user	Simple	Pointer	4 bytes		R
balanceLog	List of balances with	Simple	List	64 bytes		R

	dates and times					
date	Current Date	Complex	Month	Day	Year	R/W
Month	Current Month	Simple	Int	4 bytes		R/W
Day	Current Day	Simple	Int	4 bytes		R/W
Year	Current Year	Simple	Int	4 bytes		R/W
time	Current Time	Complex	Hour	Minute	Second	R/W
Hour	Current hour	Simple	Int	4 byte		R/W
Minute	Current minute	Simple	Int	4 byte		R/W
Second	Current second	Simple	Int	4 byte		R/W
location	Location of the action taken	Complex	Address	Zipcode	Country	R/W
type	Type of action	Simple	String	Depends		R/W
amount	Amount of action	Simple	Float	4 byte		R/W
receiver	Who received the action	Simple	String	Depends		R/W
petName	Name of interactive pet	Simple	String	8 bytes		R/W
petAge	Age of interactive pet	Complex	Day	Month	Year	R
interactionList	List of possible interactions	Simple	String list	64 bytes		R

name	Name of personal goal	Simple	String	Depends		R/W
totalAmount	Total amount invested	Simple	Float	4 byte		R/W
currentAmount	Current amount invested	Simple	Float	4 byte		R/W
startDate	Date goal started	Complex	Year	Month	Day	R/W
endDate	Date goal ended	Complex	Year	Month	Day	R/W
priority	Priority of goal	Simple	Int	4 byte		R/W
transactionLog	History of investment	Complex	String	Depends		R/W
type	Type of Financial tip	Simple	String	Depends		R/W
tip	The financial tip	Simple	String	Depends		R/W
useLocations	A list of use cases	Complex	useLocation			R/W
useLocation	A use case	Simple	String	Depends		
name	The name of the user	Simple	String	Depends		R/W
username	The username of the user	Simple	String	Depends		R/W
password	The password of the user	Simple	String	Depends		R/W

bankAccountList	A list of all bank accounts the user has	Complex	PersonalBankAccount			R/W
personalGoalList	A list of all the personal goals the user has	Complex	PersonalGoal			R/W
personalGoalName	Name of personal goal	Simple	String	Depends		R/W
personalGoalAmount	Amount of personal goal	Simple	Float	4 bytes		R/W
bank	The bank and any corresponding information	Complex	bankName	bankPolicy		R/W
bankName	Name of the bank	Simple	String	Depends		R/W
bankPolicy	Any policies of the bank	Simple	String	Depends		R/W
accountType	The type of bank account	Simple	String	Depends		R/W
fdicGuidelines	The FDIC guidelines	Complex	guideline			R/W
guideline	Individual guidelines	Simple	String	Depends		R/W

Relationship

Name	**Description**	**From class**	**To class**	**Optional/ mandatory**	**Cardinality**

Personal Budgeting System class connects to the Data Visualization interface.	Establish a connection to Data Visualization interface.	PersonalBudgeting System	Data Visualization	Mandatory	1..*
DataVisualization interface connects to the Personal BudgetingSystem class.	Establish a connection to the Personal Budgeting System class.	Data Visualization	Personal Budgeting System	Mandatory	*..1
Personal Budgeting System connects to the Database interface.	Establish a connection to Database interface.	PersonalBudgeting System	Database	Mandatory	1..*
Database interface connects to the Personal BudgetingSystem class.	Establish a connection to the Personal Budgeting System class.	Database	Personal Budgeting System	Mandatory	*..1
Personal Budgeting	Establish a connection to the BankAPI.	PersonalBudgeting System	BankAPI	Optional	1..*

System connects to the BankAPI.					
BankAPI connects to the Personal BudgetingSystem class.	Establish a connection to the Personal Budgeting System class.	BankAPI	Personal Budgeting System	Mandatory	*..1
PersonalBudgetingSystem connects to DataVisualization	Establish a connection to DataVisualization	PersonalBudgetingSystem	DataVisualization	Mandatory	1..*
PersonalBudgetingSystem connects to Database	Establish a connection to Database	PersonalBudgetingSystem	Database	Mandatory	1..*
PersonalBudgetingSystem Connects to BankAPI	Establish a connection to Bank API	PersonalBudgetingSystem	BankAPI	Optional	1..*
User connects to Personal Bank Account class	Establish a connection to Personal Bank Account class	User	Personal Bank Account	Mandatory	1..2

Personal Bank Account class connected to user	Establish a connection to user	Personal Bank Account	User	Mandatory	1..1
Personal Bank Account class has a Balance	Personal Bank Account class contains a Balance class object	Personal Bank Account	Balance	Mandatory	*..1
Balance class is associated to Personal Bank Account class	Balance class is associated with Personal Bank Account class	Balance	Personal Bank Account	Optional	1..*
Personal Bank Account class has FDICGuidelines	Personal Bank Account class contains an FDICGuidelines class object	Personal Bank Account	FDICGuidelines	Mandatory	*..1
FDICGuidelines class is associated to Personal Bank Account class	FDICGuidelines class is associated with Personal Bank Account class	FDICGuidelines	Personal Bank Account	Mandatory	1..*
User makes transaction	Make a transaction	User	Transaction	Optional	1..*

User reads FinancialTip	Read a financial tip	User	FinancialTip	Optional	*..1
User has PersonalGoal	Have a list/or a single personal goal	User	PersonalGoal	Optional	1..*
User connects to PersonalBankAccount	Establish connection to a personal bank account	User	PersonalBankAccount	Optional	1..2
User has UserRecord	Have a user record	User	UserRecord	Mandatory	1..1
User plays minigame	Play a minigame	User	minigame	Optional	***..1**
PersonalBankAccount has FDICGuidelines	Have a FDIC Guidelines	PersonalBankAccount	FDICGuidelines	Mandatory	***..1**
PersonalBankAccount with Balance	Have a record of the balance	PersonalBankAccount	Balance	Mandatory	***..1**

Key Events

Name	**Description**	**Motive**	**Action**	**Pre-conditions**	**Post-conditions**	**State Change**
Add Checking Account	The user adds their	To add a checkin	addBankAccount()	The User must have a checking	If the validation was successful, the	The new checking account

	checking account	g account		account to add that has not yet been added.	User's checking account is now added into the system. Otherwise, the system state is unchanged.	is added to the user's list of bank accounts or remains unchanged in case of connection or validation error.
Add Savings Account	The User adds their savings account to the system after successful validation through the Bank API.	To add a savings account	addBankAccount()	The User must have a savings account to add that has not yet been added.	If the validation was successful, the User's Savings Account is now added to the system. Otherwise, the system date is unchanged.	The new savings account is added to the user's list of bank accounts or remains unchanged in case of connection or validation error.
Input Financial Data	The User inputs their financial data.	To input financial data.	addData()	None.	If the connection was successful, the	The database will include

					User's financial data is added into the system Database. Otherwise, the system Database is unchanged.	the new financial data or be unchanged.
Input Transaction Data	The user inputs their transaction data	To input transaction data	addData()	None	If the connection was successful, the User's transaction data is added into the system Database. Otherwise, the system Database state is unchanged	Database state will include the added transaction data or unchanged
View Savings Account Data	The user can view their savings account data displayed with the Visual Data API	To view savings account data	updateUser()	User has a savings account added to the system	If the validation was successful, the User's visual savings data is updated in the system. Otherwise, the system state is unchanged	Personal Budgeting System state is updated or unchanged
Add personal goals	The user adds their personal goal to the	To add a new personal goal	newPersonalGoal()	None	If the connection was successful, the	The personal goal list will

	list of personal goals				user has successfully added a new goal to the list of goals in the PersonalGoal class.	include the new goal or remain unchanged.
Remove personal goals	The user removes their personal goal from the list of personal goals	To remove an existing personal goal	removePersonalGoal()	User has at least one existing personal goal	If the connection was successful, the user has successfully removed a goal from the list of goals in the PersonalGoal class.	The personal goal list will not include the removed goal or remain unchanged.
View personal goals data	The user can view data corresponding to respective personal goals	To view data about personal goals	getPersonalGoalsList()	None. If the user has no goals it will just display an empty list	If the connection is successful, the user will view their list of personal goals	As the user is simply viewing their list, the system remains unchanged.
Invest in personal goals	The user invests in their personal goals	To help user start or continue to reach a personal goal	addTransaction()	User has at least a single personal goal	If the connection is successful, the user will perform a transaction to put money	A new transaction will be added to the transaction Log, keeping

					into their personal goals	track of the transactions that took place that put money towards the personal goal, or remain unchanged in case of errors.
Read finance tips	The user reads a generated financial tip	To show the user related tips	getTip()	None	If the connection is successful, the user will see a message with the financial tip.	After the user views the message, the financial tip will not appear again, unless the user actively looks for it, or an error message will display indicatin

						g a connection error.
Interact with interactive pet	The user can interact with an interactive pet as a reward for reaching the goal	Interactive play may incentive them to get a reward for saving	start()	None	If the connection is successful, the user will interact with an interactive pet. If not, the system state will not change.	The interactive pet will start and update while the user plays, but shutdown after the user finishes.
Add to savings account	The user can add funds to their savings account	To allow the user to add money to their savings	initialize()	None	If the interaction with the interactive pet is completed, the system will display its original display before the interactive pet was displayed. Otherwise, the system state is unchanged.	The savings account balance is updated depending on the amount selected.

13.2 UML Diagrams

None outside the body of the document.

13.3 Requirements Diagrams

Function Descriptive Detailed Requirements:

1. Balance Tracking

   1.5 A user shall be able to link and access their bank account through the application.

   1.6 A user shall be able to input their own data to keep track of their finance and transactions in case the user is unwilling or unable to connect their bank account.

   1.7 A user shall be able to view their balance of their connected bank account.

   1.8 A user shall be able to add to their connected savings account balance if one is connected.

2. Statistics Visualization

   2.2 The application shall display various statistics and trends of the user and other consenting users in easy-to-understand and informative data visualizations.

3. Personal Goals

   3.3 A user shall be able to create and keep track of their personal goals.

   3.4 The application shall display various statistics and trends regarding the user's personal goals and their investments into those personal goals.

4. Information Delivery

   4.3 The application should provide informative financial tips to help the user with their money management.

   4.4 The application should provide information regarding terms and concepts used in economics.

5. Interactive Play

   5.1 The application shall allow the user to have interactive play when investing in their set personal goals.

Requirement Use Cases:

The actors in the use cases for this system are:

- User, the user using the system
- Bank API, the API for a bank or financial institution
- Visual Data API, the API to display data visually
- Database System, the database for the system
- Interactive Pet, the interactive pet the user can interact with

The generalized functional requirement use cases between actors for this system are:

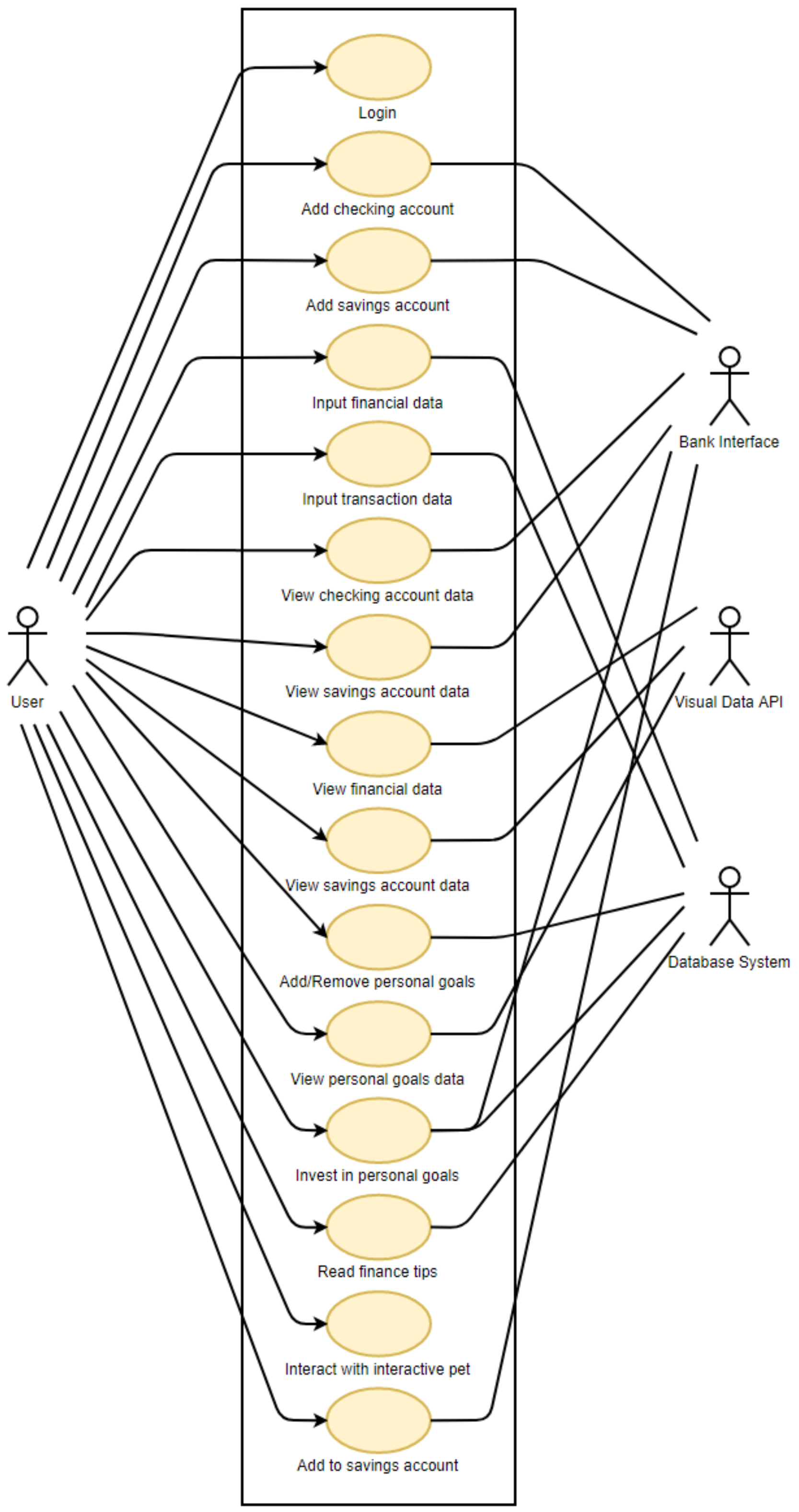
- Add a checking account between the User and the system, and the system and the Bank API (1.1)
- Add a savings account between the User and the system, and the system and the Bank API (1.1)
- Input financial data between the User and the system, and the system and the Database System (1.2)
- Input transaction data between the User and the system, and the system and the Database System (1.2)
- View checking account balance between the User and the system, and the system and the Bank API (1.3)
- View savings account balance between the User and the system, and the system and the Bank API (1.3)
- View financial data between the User and the system, and the system and the Database System, and the system and the Visual Data API (2.1)
- View savings account data between the User and the system, and the system and the Database System, and the system and the Visual Data API (2.1)

- Add/Remove personal goals between the User and the system (3.1)
- View personal goals data between the User and the system, and the system and the Database System, and the system and the Visual Data API (3.3)
- Invest in personal goals data between the User and the system, and the system and the Database System, and the system and the Bank API (3.2)
- Display finance tips and terms/concepts definitions between the User and the system, and the system and the Database System (4)
- Interact with an interactive pet between the User and the system (5.1)
- Add to savings account between the user and the system, and the system and the Bank API (1.4)

The generalized functional requirement use cases between actors for this system are:

- Login between the User and the system (8.1)

Use Case Diagrams:


Login
Add checking account
Add savings account
Input financial data
Input transaction data
View checking account data
View savings account data
View financial data
View savings account data
Add/Remove personal goals
View personal goals data
Invest in personal goals
Read finance tips
Interact with interactive pet
Add to savings account
User
Bank Interface
Visual Data API
Database System

Use Case Descriptions:

Login (8.1)		
Description	The User logs into the system.	
Pre-Conditions	None	
Flows	Basic or Normal Flows	1. Request Username and Password The system requests that the User enter their username and password. 2. Enter Username and Password The User enters their username and password. 3. Validate The system validates the entered username and password and logs the User into the system.
	Alternative Flows	A1. Incorrect Username and/or Password If in Step 3, Validate, the username and password combination is rejected, an error message will be displayed, and a username and password is requested again.
Post Conditions	If the validation was successful, the User is now logged into the system. Otherwise, the system state is unchanged.	
Special Requirements	The password input should be hidden for security reasons.	
Extension Points	None	

Add Checking Account (1.1)		
Description	The User adds their checking account to the system after successful validation through the Bank API.	
Pre-Conditions	The User must have a checking account to add that has not yet been added.	
Flows	Basic or Normal Flows	1. Request Checking Account Information The system requests that the User enter their checking account information. 2. Enter Checking Account Information The User enters their checking account information. 3. Validate The system validates the entered checking account information through the Bank API and adds the checking account to the system.
	Alternative Flows	A1. Incorrect Checking Account Information If in Step 3, Validate, the checking account information is rejected, an error message will be displayed, and the checking account information is requested again. A2. Cancel Adding Checking Account Information If in Step 2, Enter Checking Account Information, the User cancels the action, the system will return to the previous screen.
Post Conditions	If the validation was successful, the User's checking account is now added into the system. Otherwise, the system state is unchanged.	

Special Requirements	None	
Extension Points	None	

Add Savings Account (1.1)		
Description	The User adds their savings account to the system after successful validation through the Bank API.	
Pre-Conditions	The User must have a savings account to add that has not yet been added.	
Flows	Basic or Normal Flows	1. Request Savings Account Information The system requests that the User enter their savings account information. 2. Enter Savings Account Information The User enters their savings account information. 3. Validate The system validates the entered savings account information through the Bank API and adds the checking account to the system.
	Alternative Flows	A1. Incorrect Checking Account Information If in Step 3, Validate, the checking account information is rejected, an error message will be displayed, and the checking account information is requested again. A2. Cancel Adding Checking Account Information If in Step 2, Enter Checking Account Information, the User cancels the action, the system will return to the previous screen.
Post Conditions	If the validation was successful, the User's checking account is now added into the system. Otherwise, the system state is unchanged.	
Special Requirements	None	
Extension Points	None	

Input Financial Data (1.2)		
Description	The User inputs their financial data.	
Pre-Conditions	The User does not have a checking account added into the system.	
Flows	Basic or Normal Flows	1. Request Financial Data The system requests the User's financial data (the total amount of money they have at the current time). 2. Input Financial Data The User inputs their financial data. 3. Connect to Database The system makes a connection with the Database System. 4. Add Financial Data The system adds the User's financial data into the Database System.

	Alternative Flows	A1. Invalidated Connection If in Step 3, Connect to Database, the Database rejects the connection, the system will request for a connection an additional three times. If the connection is still rejected, it will display an error message and request the User's financial data again. A2. Cancel Adding Financial Data If in Step 2, Input Financial Data, the User cancels the action, the system will return to the previous screen.
Post Conditions	If the connection was successful, the User's financial data is added into the system Database. Otherwise, the system Database state is unchanged.	
Special Requirements	The User must not have already added their checking account to the system.	
Extension Points	None	

Input Transaction Data (1.2)		
Description	The User inputs their transaction data.	
Pre-Conditions	None	
Flows	Basic or Normal Flows	1. Request Transaction Data The system requests the User's transaction data. 2. Input Transaction Data The User inputs their transaction data. 3. Connect to Database The system makes a connection with the Database. 4. Add Transaction Data The system adds the User's transaction data into the Database.
	Alternative Flows	A1. Invalidated Connection If in Step 3, Connect to Database, the Database rejects the connection, the system will request for a connection an additional three times. If the connection is still rejected, it will display an error message and request the User's transaction data again. A2. Cancel Adding Transaction Data If in Step 2, Input Transaction Data, the User cancels the action, the system will return to the previous screen.
Post Conditions	If the connection was successful, the User's transaction data is added into the system Database. Otherwise, the system Database state is unchanged.	
Special Requirements	The User must not have already added their checking account to the system.	
Extension Points	None	

View Checking Account Balance (1.3)	
Description	The User can view their checking account balance.
Pre-Conditions	The User must have a checking account added to the system.

Flows	Basic or Normal Flows	1. Request Checking Account Information Upon entering the screen indicating the User's checking account balance, the system requests the User's checking account balance from the Bank API. 2. Validate Connection The Bank API will validate the connection between the system and the Bank API and will return the User's total checking account balance. 3. Display Checking Account Balance The system displays the User's total checking account balance acquired from the Bank API connection.
	Alternative Flows	A1. Invalidated Connection If in Step 2, Validate Connection, the Bank API rejects the connection, the system will request for a connection an additional three times. If the connection is still rejected, it will display an error indication and display the last known balance.
Post Conditions	If the validation was successful, the User's checking account balance is updated in the system. Otherwise, the system state is unchanged.	
Special Requirements	None	
Extension Points	None	

View Savings Account Balance (1.3)		
Description	The User can view their savings account balance.	
Pre-Conditions	The User must have a savings account added to the system.	
Flows	Basic or Normal Flows	1. Request Savings Account Information Upon entering the screen indicating the User's savings account balance, the system requests the User's savings account balance from the Bank API. 2. Validate Connection The Bank Interface will validate the connection between the system and the Bank API and will return the User's total savings account balance. 3. Display Savings Account Balance The system displays the User's total savings account balance acquired from the Bank API connection.
	Alternative Flows	A1. Invalidated Connection If in Step 2, Validate Connection, the Bank API rejects the connection, the system will request for a connection an additional three times. If the connection is still rejected, it will display an error indication and display the last known balance.
Post Conditions	If the validation was successful, the User's savings account balance is updated in the system. Otherwise, the system state is unchanged.	
Special Requirements	None	

Extension Points	None	

View Financial Data (2.1)		
Description	The User can view their checking account data or financial data displayed with the Visual Data API.	
Pre-Conditions	The User must have either a checking account added to the system or have personally inputted their financial data.	
Flows	Basic or Normal Flows	1. Determine Data Type Upon entering the screen displaying the User's financial data, the system will determine if the User's data is from the Bank API or from the User's Input. 2.A. Request Checking Account Data If the User's data is from the Bank API, the system will request a connection to the corresponding Bank API. 2.B. Connect to Database If the User's data is from the User's input, the system will request a connection to the Database. 3. Validate Connection The Bank API will validate the connection from the system to either the Bank API or the Database and will return the User's financial data. 4. Display Financial Data The system displays the User's financial data through the Visual Data API.
	Alternative Flows	A1. Invalidated Connection If in Step 3, Validate Connection, the Bank API or the Database rejects the connection, the system will request for a connection an additional three times. If the connection is still rejected, it will display an error indication and display the last known financial data through the Visual Data API.
Post Conditions	If the validation was successful, the User's visual financial data is updated in the system. Otherwise, the system state is unchanged.	
Special Requirements	None	
Extension Points	None	

View Savings Account Data (2.1)		
Description	The User can view their savings account data displayed with the Visual Data API.	
Pre-Conditions	The User must have a savings account added to the system.	
Flows	Basic or Normal Flows	1. Request Savings Account Data Upon entering the screen displaying the User's savings data, the system will request a connection to the corresponding Bank API. 2. Validate Connection

		The Bank API will validate the connection from the system to the Bank API and will return the User's savings data. 3. Display Savings Data The system displays the User's savings data through the Visual Data API.
	Alternative Flows	A1. Invalidated Connection If in Step 3, Validate Connection, the Bank API rejects the connection, the system will request for a connection an additional three times. If the connection is still rejected, it will display an error indication and display the last known savings through the Visual Data API.
Post Conditions	If the validation was successful, the User's visual savings data is updated in the system. Otherwise, the system state is unchanged.	
Special Requirements	None	
Extension Points	None	

Add/Remove Personal Goals (3.1)		
Description	The User can add or remove their personal goals to the system.	
Pre-Conditions	None	
Flows	Basic or Normal Flows	1. Request Personal Goal Information Upon entering the screen displaying the User's personal goals, the system will request the User's personal goal information. 2. Input Personal Goal Information The User will input the information for their personal goal. 3. Add Personal Goal The system will add the User's personal goal to the system.
	Alternative Flows	A1. Cancel If in Step 2, Input Personal Goal Information, the User cancels the action, the system will return to the previous screen.
Post Conditions	If the action is completed, the User's personal goal is added to the system. Otherwise, the system state is unchanged.	
Special Requirements	None	
Extension Points	None	

View Personal Goals Data (3.3)	
Description	The User can view data related to their personal goals in the system.
Pre-Conditions	The User must have at least one personal goal added into the system.

Flows	Basic or Normal Flows	1. Request Personal Goal Data Upon entering the screen displaying the User's personal goal data, the system will request the User's personal goal data. 2. Connect to Database The User will request a connection to the Database and return the User's personal goal data. 3. Display Personal Goal Data The system displays the User's personal goal data through the Visual Data API.
	Alternative Flows	A1. Invalidated Connection If in Step 2, Connect to Database, the Database rejects the connection, the system will request for a connection an additional three times. If the connection is still rejected, it will display an error indication and display the last known personal goal data through the Visual Data API.
Post Conditions	If the action is completed, the User's visual personal goal data is updated in the system. Otherwise, the system state is unchanged.	
Special Requirements	None	
Extension Points	None	

Invest in Personal Goals (3.2)		
Description	The User can invest in their personal goals added into the system.	
Pre-Conditions	The User must have at least one personal goal added into the system and have instigated the investment into the personal goal.	
Flows	Basic or Normal Flows	1. Request Personal Goal Data Upon entering the screen displaying the User's personal goal data, the system will request the User's personal goal data. 2. Connect to Database The system will request a connection to the Database and return the User's personal goal data. 3. Request Personal Goal Selection Upon selecting to add to a personal goal, the system will request the User to select the personal goal they want to invest into. 4. Select Personal Goal The User will select the personal goal they want to invest in. 5. Request Investment Amount The system will request the User to input the amount they want to invest into the personal goal. 6. Input Investment Amount

		The User will input the amount they want to invest into the personal goal. 7. Update The system will add the inputted amount into the total amount already invested into the goal and update associated data and statistics. 8. Complete Transaction The system will initiate a transaction through the Bank API for the inputted amount from the user.
	Alternative Flows	A1. Invalidated Connection If in Step 2, Connect to Database, the Database rejects the connection, the system will request for a connection an additional three times. If the connection is still rejected, it will display an error indication and return to the previous screen. A2. Cancel If in Step 4 to 6, Select Personal Goal to Input Investment Amount, the User cancels their action, the system will return to the previous screen and will not carry out the following steps. A3. Insufficient Funds If in Step 6, Input Investment Amount, the User inputs an amount greater than what they have in their checking account, the system will display an indication and request an amount from the User again. A4. Update Failure If in Step 7, Update, the system is unable to update the information, the system will attempt to update the information an additional three time. If failure continues, revert back to the last working state, display an error message, and do not move on to the following step. A5. Transaction Failure If in Step 8, Complete Transaction, the system fails to complete the transaction through the Bank API, the system will attempt to complete the transaction through the Bank API an additional three times. If the failure continues, revert back to the last working state before the update and display an error message.
Post Conditions	If the action is completed, the User's visual personal goal data is updated in the system. Otherwise, the system state is unchanged.	
Special Requirements	None	
Extension Points	None	

Display Finance Tips and Terms/Concepts Definitions (4)

Description	The User can read finance tips.	
Pre-Conditions	The User must do an action that is associated with a finance tip or a term/concept definition.	
Flows	Basic or Normal Flows	1. Connect to Database Upon completing an action that is associated with a finance tip, the system will request a connection to the Database and return the associated finance tip. 2. Display Finance Tip or Term/Concept The system will display the finance tip or the term/concept definition for the User to read.
	Alternative Flows	A1. Invalidated Connection If in Step 1, Connect to Database, the Database rejects the connection, the system will request for a connection an additional three times. If the connection is still rejected, it will not do the following step.
Post Conditions	If the action is completed, the finance tip or the term/concept definition will be displayed for the User in the system. Otherwise, the system state is unchanged.	
Special Requirements	None	
Extension Points	None	

Interact with Interactive Pet (5.1)		
Description	The User can interact with an interactive pet.	
Pre-Conditions	The User must do an action that is associated with the interactive pet.	
Flows	Basic or Normal Flows	1. Display Interactive Pet Upon completing an action that is associated with the interactive pet, the system will display the interactive pet. 2. User Interaction The user can interact with the interactive pet.
	Alternative Flows	A1. Failure to Display Interactive Pet If in Step 1, Display Interactive Pet, the interactive pet fails to load after 10 seconds, the system will skip the interactive pet entirely.
Post Conditions	If the interaction is completed, the system will display its original display before the interactive pet was displayed. Otherwise, the system state is unchanged.	
Special Requirements	None	
Extension Points	None	

Add to Savings Account (1.4)	
Description	The User can add money into their savings account.
Pre-Conditions	The User must have a savings account and a checking account already added into the system.

Flows	Basic or Normal Flows	1. Request Amount Upon the User selecting to add to their savings account balance, the system requests the User to input the amount they want to add into the savings account from their checking account. 2. Input Amount The User will input the amount they want to add to their savings account balance from their checking account balance. 3. Update The system will add the inputted amount into the User's total savings balance and update associated data and statistics. 4. Complete Transaction The system will instigate the transaction through the Bank API for the inputted amount from the User.
	Alternative Flows	A1. Insufficient Funds If in Step 2, Input Amount, the User inputs an amount that is greater than the balance they currently have in their checking account, the system will display an error indication and request an amount again from the User. A2. Cancel If in Step 2, Input Amount, the User cancels their action, the system will return to the previous screen and will not carry out the following steps. A2. Update Failure If in Step 3, Update, the system is unable to update the data and statistics, the system will attempt to update the data and statistics an additional three times. If failure continues, revert back to the last working state, display an error message, and do not move on to the following step. A3. Transaction Failure If in Step 4, Complete Transaction, the system fails to complete the transaction through the Bank API, the system will attempt to complete the transaction through the Bank API an additional three times. If the failure continues, revert back to the last working state before the update and display an error message.
Post Conditions	If the mini game is completed, the system will display its original display before the mini game was displayed. Otherwise, the system state is unchanged.	
Special Requirements	Must follow the rules and regulations of the associated financial institution.	
Extension Points	None	

13.4 Schedule Tracking

Artifact or Deliverable	Who (Individual and Team)	Estimated	Actual	Difference
SPMP	Amanda Lin	10 hours		
	Gordon Lei	5 hours		
	Jason Li	5 hours		
	Jay Kang	5 hours		
	Summary for entire team	25 hours		

Artifact or Deliverable	Who (Individual and Team)	Estimated	Actual	Difference
SRS - Final	Amanda Lin	10 hours	10 hours	0 hours
	Gordon Lei	5 hours	5 hours	0 hours
	Jason Li	5 hours	5 hours	0 hours
	Jay Kang	5 hours	5 hours	0 hours
	Summary for entire team	25 hours	25 hours	0 hours

Artifact or Deliverable	Who (Individual and Team)	Estimated	Actual	Difference
SDD - Initial	Amanda Lin	5 hours	10 hours	+5 hours
	Gordon Lei	5 hours	3 hours	-2 hours
	Jason Li	5 hours	3 hours	-2 hours
	Jay Kang	5 hours	7 hours	+2 hours
	Summary for entire team	20 hours	23 hours	+3 hours

Artifact or Deliverable	Who (Individual and Team)	Estimated	Actual	Difference
SDD - Final	Amanda Lin	5 hours		
	Gordon Lei	5 hours		
	Jason Li	5 hours		
	Jay Kang	5 hours		
	Summary for entire team	20 hours		

Cumulative

Who (Individual or Team)	Estimated	Actual	Difference
Amanda Lin	30 hours		
Gordon Lei	20 hours		
Jason Li	20 hours		
Jay Kang	20 hours		
Summary for entire team	90 hours		

13.5 Defect Tracking

Artifact or Deliverable	Who (Individual and Team)	Estimated	Actual	Difference
SPMP	Amanda Lin	40		
	Gordon Lei	20		
	Jason Li	20		
	Jay Kang	20		
	Summary for entire team	100		

Artifact or Deliverable	Who (Individual and Team)	Estimated	Actual	Difference
SRS - Final	Amanda Lin	40	16	-24
	Gordon Lei	20	2	-18
	Jason Li	20	10	-10
	Jay Kang	20	1	-19
	Summary for entire team	100	29	-71

Artifact or Deliverable	Who (Individual and Team)	Estimated	Actual	Difference
SDD - Initial	Amanda Lin	20	17	-3
	Gordon Lei	20	3	-17
	Jason Li	20	12	-8
	Jay Kang	20	12	-8

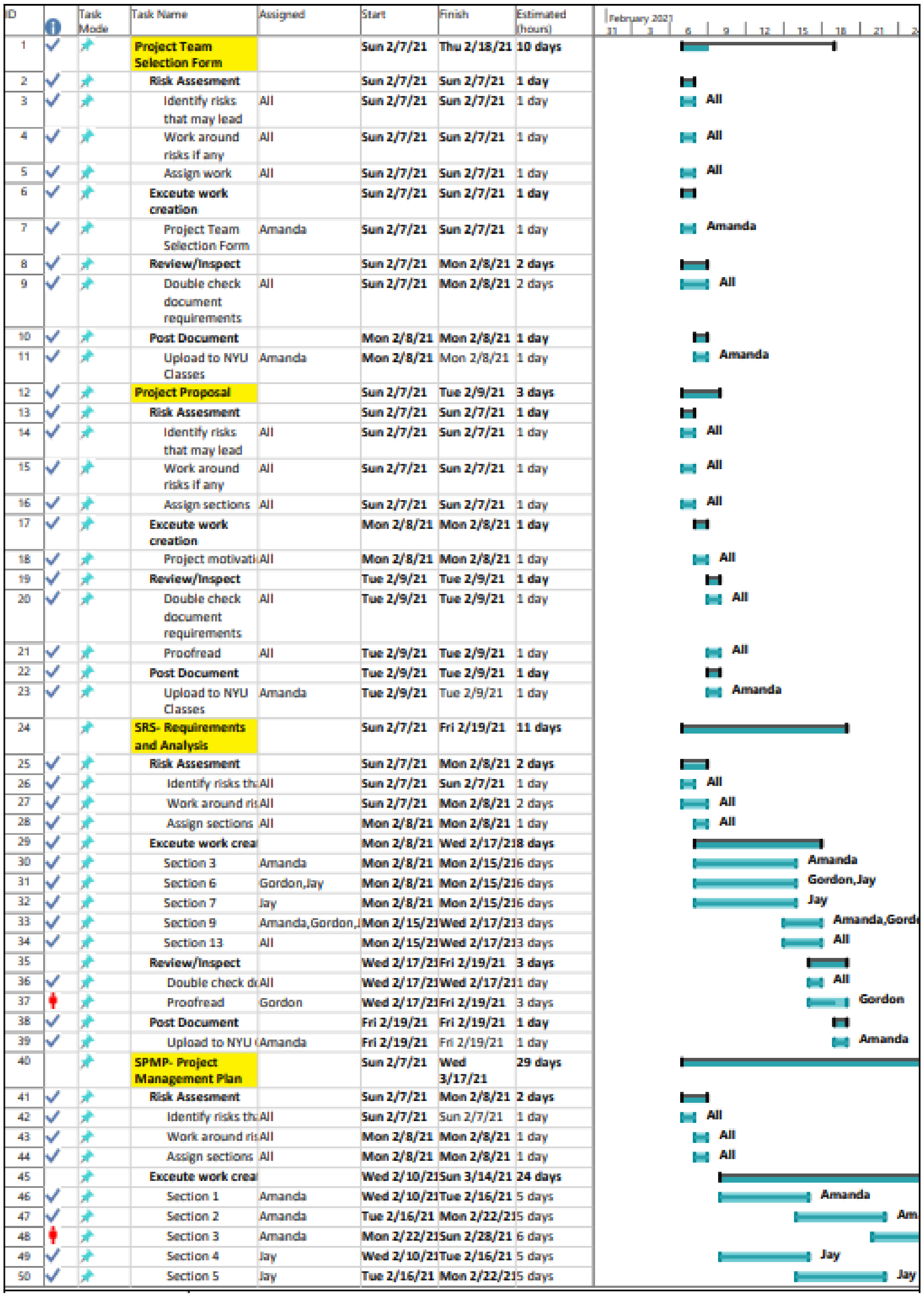
	Summary for entire team	80	41	-39

Artifact or Deliverable	Who (Individual and Team)	Estimated	Actual	Difference
SDD - Final	Amanda Lin	20		
	Gordon Lei	20		
	Jason Li	20		
	Jay Kang	20		
	Summary for entire team	80		

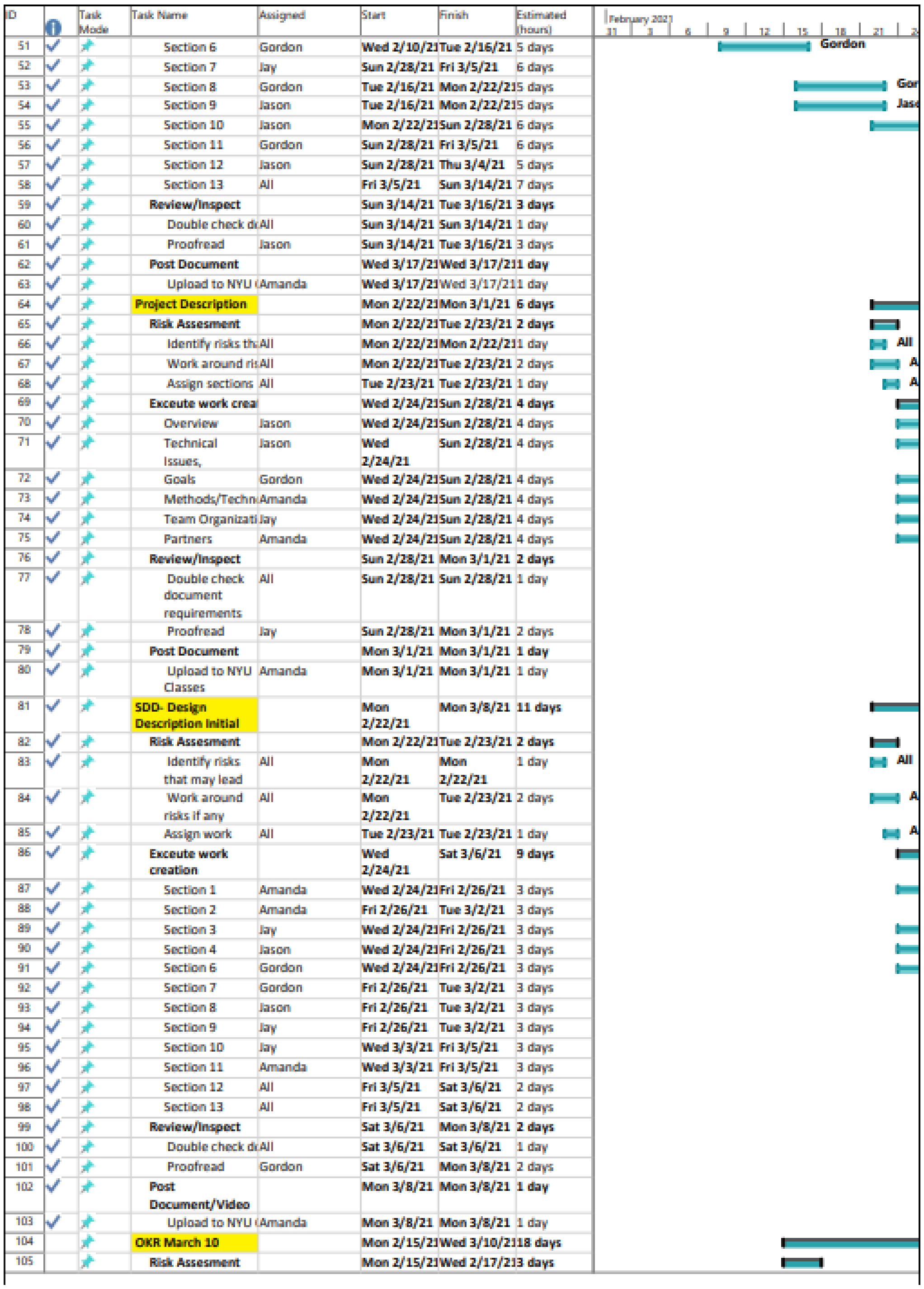
Cumulative

Who (Individual or Team)	Estimated	Actual	Difference
Amanda Lin	120		
Gordon Lei	80		
Jason Li	80		
Jay Kang	80		
Summary for entire team	360		

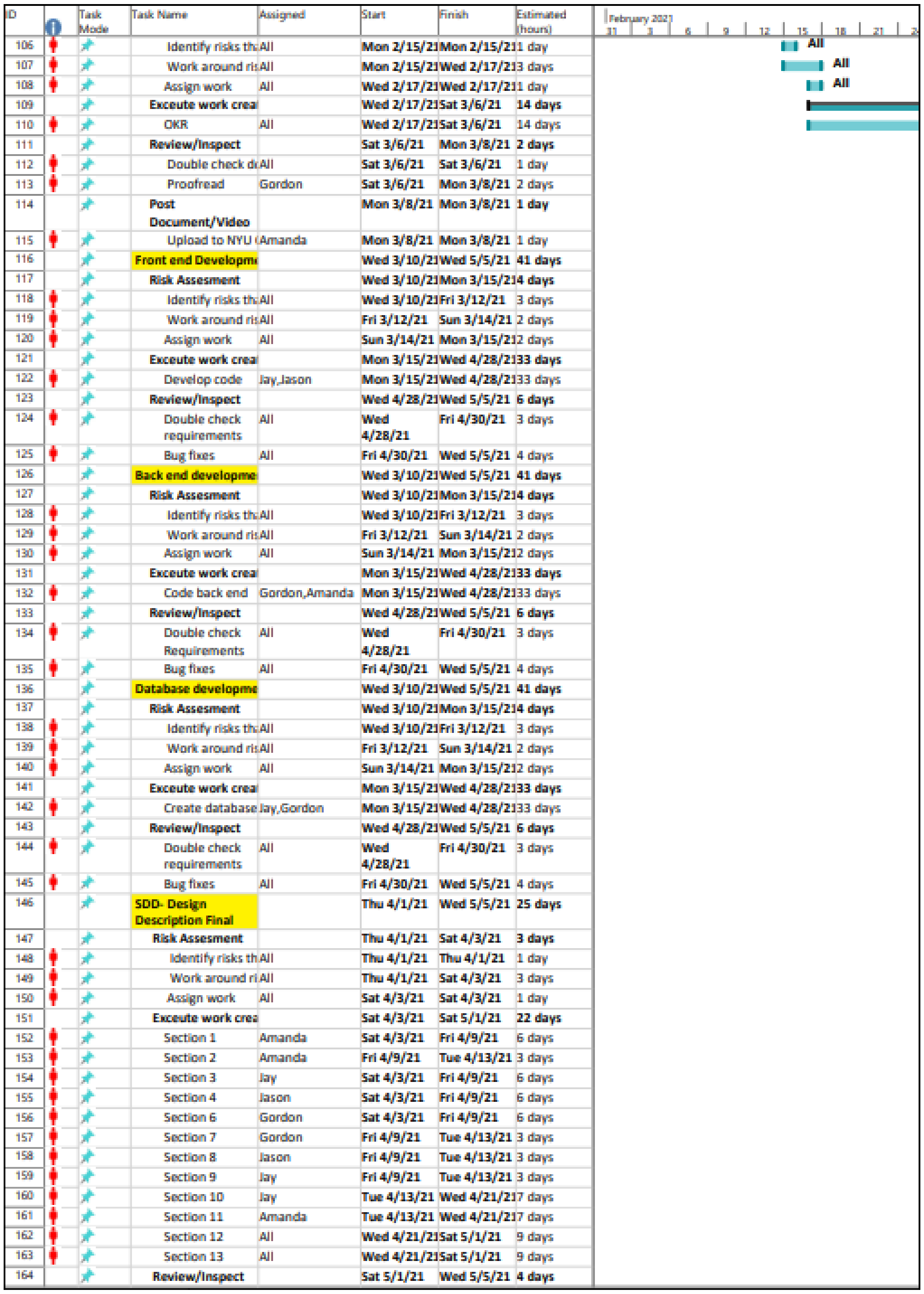
13.5 Project Schedule



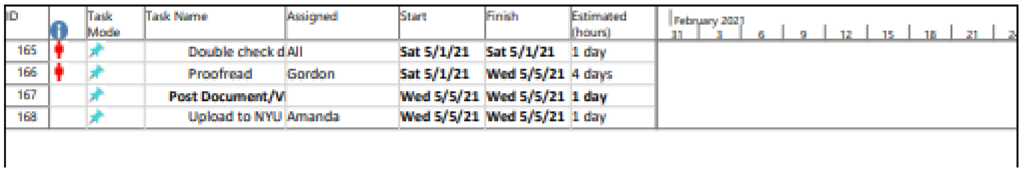
ID	ⓘ	Task Mode	Task Name	Assigned	Start	Finish	Estimated (hours)
1	✓	📌	**Project Team Selection Form**		Sun 2/7/21	Thu 2/18/21	10 days
2	✓	📌	**Risk Assesment**		Sun 2/7/21	Sun 2/7/21	1 day
3	✓	📌	Identify risks that may lead	All	Sun 2/7/21	Sun 2/7/21	1 day
4	✓	📌	Work around risks if any	All	Sun 2/7/21	Sun 2/7/21	1 day
5	✓	📌	Assign work	All	Sun 2/7/21	Sun 2/7/21	1 day
6	✓	📌	**Exceute work creation**		Sun 2/7/21	Sun 2/7/21	1 day
7	✓	📌	Project Team Selection Form	Amanda	Sun 2/7/21	Sun 2/7/21	1 day
8	✓	📌	**Review/Inspect**		Sun 2/7/21	Mon 2/8/21	2 days
9	✓	📌	Double check document requirements	All	Sun 2/7/21	Mon 2/8/21	2 days
10	✓	📌	**Post Document**		Mon 2/8/21	Mon 2/8/21	1 day
11	✓	📌	Upload to NYU Classes	Amanda	Mon 2/8/21	Mon 2/8/21	1 day
12	✓	📌	**Project Proposal**		Sun 2/7/21	Tue 2/9/21	3 days
13	✓	📌	**Risk Assesment**		Sun 2/7/21	Sun 2/7/21	1 day
14	✓	📌	Identify risks that may lead	All	Sun 2/7/21	Sun 2/7/21	1 day
15	✓	📌	Work around risks if any	All	Sun 2/7/21	Sun 2/7/21	1 day
16	✓	📌	Assign sections	All	Sun 2/7/21	Sun 2/7/21	1 day
17	✓	📌	**Exceute work creation**		Mon 2/8/21	Mon 2/8/21	1 day
18	✓	📌	Project motivati	All	Mon 2/8/21	Mon 2/8/21	1 day
19	✓	📌	**Review/Inspect**		Tue 2/9/21	Tue 2/9/21	1 day
20	✓	📌	Double check document requirements	All	Tue 2/9/21	Tue 2/9/21	1 day
21	✓	📌	Proofread	All	Tue 2/9/21	Tue 2/9/21	1 day
22	✓	📌	**Post Document**		Tue 2/9/21	Tue 2/9/21	1 day
23	✓	📌	Upload to NYU Classes	Amanda	Tue 2/9/21	Tue 2/9/21	1 day
24		📌	**SRS- Requirements and Analysis**		Sun 2/7/21	Fri 2/19/21	11 days
25	✓	📌	**Risk Assesment**		Sun 2/7/21	Mon 2/8/21	2 days
26	✓	📌	Identify risks th	All	Sun 2/7/21	Sun 2/7/21	1 day
27	✓	📌	Work around ris	All	Sun 2/7/21	Mon 2/8/21	2 days
28	✓	📌	Assign sections	All	Mon 2/8/21	Mon 2/8/21	1 day
29	✓	📌	**Exceute work crea**		Mon 2/8/21	Wed 2/17/21	8 days
30	✓	📌	Section 3	Amanda	Mon 2/8/21	Mon 2/15/21	6 days
31	✓	📌	Section 6	Gordon,Jay	Mon 2/8/21	Mon 2/15/21	6 days
32	✓	📌	Section 7	Jay	Mon 2/8/21	Mon 2/15/21	6 days
33	✓	📌	Section 9	Amanda,Gordon,J	Mon 2/15/21	Wed 2/17/21	3 days
34	✓	📌	Section 13	All	Mon 2/15/21	Wed 2/17/21	3 days
35		📌	**Review/Inspect**		Wed 2/17/21	Fri 2/19/21	3 days
36	✓	📌	Double check d	All	Wed 2/17/21	Wed 2/17/21	1 day
37	●	📌	Proofread	Gordon	Wed 2/17/21	Fri 2/19/21	3 days
38	✓	📌	**Post Document**		Fri 2/19/21	Fri 2/19/21	1 day
39	✓	📌	Upload to NYU (	Amanda	Fri 2/19/21	Fri 2/19/21	1 day
40		📌	**SPMP- Project Management Plan**		Sun 2/7/21	Wed 3/17/21	29 days
41	✓	📌	**Risk Assesment**		Sun 2/7/21	Mon 2/8/21	2 days
42	✓	📌	Identify risks th	All	Sun 2/7/21	Sun 2/7/21	1 day
43	✓	📌	Work around ris	All	Mon 2/8/21	Mon 2/8/21	1 day
44	✓	📌	Assign sections	All	Mon 2/8/21	Mon 2/8/21	1 day
45		📌	**Exceute work crea**		Wed 2/10/21	Sun 3/14/21	24 days
46	✓	📌	Section 1	Amanda	Wed 2/10/21	Tue 2/16/21	5 days
47	✓	📌	Section 2	Amanda	Tue 2/16/21	Mon 2/22/21	5 days
48	●	📌	Section 3	Amanda	Mon 2/22/21	Sun 2/28/21	6 days
49	✓	📌	Section 4	Jay	Wed 2/10/21	Tue 2/16/21	5 days
50	✓	📌	Section 5	Jay	Tue 2/16/21	Mon 2/22/21	5 days



ID	ℹ	Task Mode	Task Name	Assigned	Start	Finish	Estimated (hours)
51	✓	📌	Section 6	Gordon	Wed 2/10/21	Tue 2/16/21	5 days
52	✓	📌	Section 7	Jay	Sun 2/28/21	Fri 3/5/21	6 days
53	✓	📌	Section 8	Gordon	Tue 2/16/21	Mon 2/22/21	5 days
54	✓	📌	Section 9	Jason	Tue 2/16/21	Mon 2/22/21	5 days
55	✓	📌	Section 10	Jason	Mon 2/22/21	Sun 2/28/21	6 days
56	✓	📌	Section 11	Gordon	Sun 2/28/21	Fri 3/5/21	6 days
57	✓	📌	Section 12	Jason	Sun 2/28/21	Thu 3/4/21	5 days
58	✓	📌	Section 13	All	Fri 3/5/21	Sun 3/14/21	7 days
59	✓	📌	**Review/Inspect**		Sun 3/14/21	Tue 3/16/21	**3 days**
60	✓	📌	Double check d	All	Sun 3/14/21	Sun 3/14/21	1 day
61	✓	📌	Proofread	Jason	Sun 3/14/21	Tue 3/16/21	3 days
62	✓	📌	**Post Document**		Wed 3/17/21	Wed 3/17/21	**1 day**
63	✓	📌	Upload to NYU (	Amanda	Wed 3/17/21	Wed 3/17/21	1 day
64	✓	📌	**Project Description**		Mon 2/22/21	Mon 3/1/21	**6 days**
65	✓	📌	**Risk Assesment**		Mon 2/22/21	Tue 2/23/21	**2 days**
66	✓	📌	Identify risks th	All	Mon 2/22/21	Mon 2/22/21	1 day
67	✓	📌	Work around ris	All	Mon 2/22/21	Tue 2/23/21	2 days
68	✓	📌	Assign sections	All	Tue 2/23/21	Tue 2/23/21	1 day
69	✓	📌	**Exceute work crea**		Wed 2/24/21	Sun 2/28/21	**4 days**
70	✓	📌	Overview	Jason	Wed 2/24/21	Sun 2/28/21	4 days
71	✓	📌	Technical Issues,	Jason	Wed 2/24/21	Sun 2/28/21	4 days
72	✓	📌	Goals	Gordon	Wed 2/24/21	Sun 2/28/21	4 days
73	✓	📌	Methods/Techn	Amanda	Wed 2/24/21	Sun 2/28/21	4 days
74	✓	📌	Team Organizati	Jay	Wed 2/24/21	Sun 2/28/21	4 days
75	✓	📌	Partners	Amanda	Wed 2/24/21	Sun 2/28/21	4 days
76	✓	📌	**Review/Inspect**		Sun 2/28/21	Mon 3/1/21	**2 days**
77	✓	📌	Double check document requirements	All	Sun 2/28/21	Sun 2/28/21	1 day
78	✓	📌	Proofread	Jay	Sun 2/28/21	Mon 3/1/21	2 days
79	✓	📌	**Post Document**		Mon 3/1/21	Mon 3/1/21	**1 day**
80	✓	📌	Upload to NYU Classes	Amanda	Mon 3/1/21	Mon 3/1/21	1 day
81	✓	📌	**SDD- Design Description Initial**		Mon 2/22/21	Mon 3/8/21	**11 days**
82	✓	📌	**Risk Assesment**		Mon 2/22/21	Tue 2/23/21	**2 days**
83	✓	📌	Identify risks that may lead	All	Mon 2/22/21	Mon 2/22/21	1 day
84	✓	📌	Work around risks if any	All	Mon 2/22/21	Tue 2/23/21	2 days
85	✓	📌	Assign work	All	Tue 2/23/21	Tue 2/23/21	1 day
86	✓	📌	**Exceute work creation**		Wed 2/24/21	Sat 3/6/21	**9 days**
87	✓	📌	Section 1	Amanda	Wed 2/24/21	Fri 2/26/21	3 days
88	✓	📌	Section 2	Amanda	Fri 2/26/21	Tue 3/2/21	3 days
89	✓	📌	Section 3	Jay	Wed 2/24/21	Fri 2/26/21	3 days
90	✓	📌	Section 4	Jason	Wed 2/24/21	Fri 2/26/21	3 days
91	✓	📌	Section 6	Gordon	Wed 2/24/21	Fri 2/26/21	3 days
92	✓	📌	Section 7	Gordon	Fri 2/26/21	Tue 3/2/21	3 days
93	✓	📌	Section 8	Jason	Fri 2/26/21	Tue 3/2/21	3 days
94	✓	📌	Section 9	Jay	Fri 2/26/21	Tue 3/2/21	3 days
95	✓	📌	Section 10	Jay	Wed 3/3/21	Fri 3/5/21	3 days
96	✓	📌	Section 11	Amanda	Wed 3/3/21	Fri 3/5/21	3 days
97	✓	📌	Section 12	All	Fri 3/5/21	Sat 3/6/21	2 days
98	✓	📌	Section 13	All	Fri 3/5/21	Sat 3/6/21	2 days
99	✓	📌	**Review/Inspect**		Sat 3/6/21	Mon 3/8/21	**2 days**
100	✓	📌	Double check d	All	Sat 3/6/21	Sat 3/6/21	1 day
101	✓	📌	Proofread	Gordon	Sat 3/6/21	Mon 3/8/21	2 days
102	✓	📌	**Post Document/Video**		Mon 3/8/21	Mon 3/8/21	**1 day**
103	✓	📌	Upload to NYU (	Amanda	Mon 3/8/21	Mon 3/8/21	1 day
104		📌	**OKR March 10**		Mon 2/15/21	Wed 3/10/21	**18 days**
105		📌	**Risk Assesment**		Mon 2/15/21	Wed 2/17/21	**3 days**



ID	ℹ	Task Mode	Task Name	Assigned	Start	Finish	Estimated (hours)
106	●	📌	Identify risks th	All	Mon 2/15/21	Mon 2/15/21	1 day
107	●	📌	Work around ris	All	Mon 2/15/21	Wed 2/17/21	3 days
108	●	📌	Assign work	All	Wed 2/17/21	Wed 2/17/21	1 day
109		📌	**Exceute work crea**		**Wed 2/17/21**	**Sat 3/6/21**	**14 days**
110	●	📌	OKR	All	Wed 2/17/21	Sat 3/6/21	14 days
111		📌	**Review/Inspect**		**Sat 3/6/21**	**Mon 3/8/21**	**2 days**
112	●	📌	Double check d	All	Sat 3/6/21	Sat 3/6/21	1 day
113	●	📌	Proofread	Gordon	Sat 3/6/21	Mon 3/8/21	2 days
114		📌	**Post Document/Video**		**Mon 3/8/21**	**Mon 3/8/21**	**1 day**
115	●	📌	Upload to NYU (	Amanda	Mon 3/8/21	Mon 3/8/21	1 day
116		📌	**Front end Developm**		**Wed 3/10/21**	**Wed 5/5/21**	**41 days**
117		📌	**Risk Assesment**		**Wed 3/10/21**	**Mon 3/15/21**	**4 days**
118	●	📌	Identify risks th	All	Wed 3/10/21	Fri 3/12/21	3 days
119	●	📌	Work around ris	All	Fri 3/12/21	Sun 3/14/21	2 days
120	●	📌	Assign work	All	Sun 3/14/21	Mon 3/15/21	2 days
121		📌	**Exceute work crea**		**Mon 3/15/21**	**Wed 4/28/21**	**33 days**
122	●	📌	Develop code	Jay,Jason	Mon 3/15/21	Wed 4/28/21	33 days
123		📌	**Review/Inspect**		**Wed 4/28/21**	**Wed 5/5/21**	**6 days**
124	●	📌	Double check requirements	All	Wed 4/28/21	Fri 4/30/21	3 days
125	●	📌	Bug fixes	All	Fri 4/30/21	Wed 5/5/21	4 days
126		📌	**Back end developme**		**Wed 3/10/21**	**Wed 5/5/21**	**41 days**
127		📌	**Risk Assesment**		**Wed 3/10/21**	**Mon 3/15/21**	**4 days**
128	●	📌	Identify risks th	All	Wed 3/10/21	Fri 3/12/21	3 days
129	●	📌	Work around ris	All	Fri 3/12/21	Sun 3/14/21	2 days
130	●	📌	Assign work	All	Sun 3/14/21	Mon 3/15/21	2 days
131		📌	**Exceute work crea**		**Mon 3/15/21**	**Wed 4/28/21**	**33 days**
132	●	📌	Code back end	Gordon,Amanda	Mon 3/15/21	Wed 4/28/21	33 days
133		📌	**Review/Inspect**		**Wed 4/28/21**	**Wed 5/5/21**	**6 days**
134	●	📌	Double check Requirements	All	Wed 4/28/21	Fri 4/30/21	3 days
135	●	📌	Bug fixes	All	Fri 4/30/21	Wed 5/5/21	4 days
136		📌	**Database developme**		**Wed 3/10/21**	**Wed 5/5/21**	**41 days**
137		📌	**Risk Assesment**		**Wed 3/10/21**	**Mon 3/15/21**	**4 days**
138	●	📌	Identify risks th	All	Wed 3/10/21	Fri 3/12/21	3 days
139	●	📌	Work around ris	All	Fri 3/12/21	Sun 3/14/21	2 days
140	●	📌	Assign work	All	Sun 3/14/21	Mon 3/15/21	2 days
141		📌	**Exceute work crea**		**Mon 3/15/21**	**Wed 4/28/21**	**33 days**
142	●	📌	Create database	Jay,Gordon	Mon 3/15/21	Wed 4/28/21	33 days
143		📌	**Review/Inspect**		**Wed 4/28/21**	**Wed 5/5/21**	**6 days**
144	●	📌	Double check requirements	All	Wed 4/28/21	Fri 4/30/21	3 days
145	●	📌	Bug fixes	All	Fri 4/30/21	Wed 5/5/21	4 days
146		📌	**SDD- Design Description Final**		**Thu 4/1/21**	**Wed 5/5/21**	**25 days**
147		📌	**Risk Assesment**		**Thu 4/1/21**	**Sat 4/3/21**	**3 days**
148	●	📌	Identify risks th	All	Thu 4/1/21	Thu 4/1/21	1 day
149	●	📌	Work around ri	All	Thu 4/1/21	Sat 4/3/21	3 days
150	●	📌	Assign work	All	Sat 4/3/21	Sat 4/3/21	1 day
151		📌	**Exceute work crea**		**Sat 4/3/21**	**Sat 5/1/21**	**22 days**
152	●	📌	Section 1	Amanda	Sat 4/3/21	Fri 4/9/21	6 days
153	●	📌	Section 2	Amanda	Fri 4/9/21	Tue 4/13/21	3 days
154	●	📌	Section 3	Jay	Sat 4/3/21	Fri 4/9/21	6 days
155	●	📌	Section 4	Jason	Sat 4/3/21	Fri 4/9/21	6 days
156	●	📌	Section 6	Gordon	Sat 4/3/21	Fri 4/9/21	6 days
157	●	📌	Section 7	Gordon	Fri 4/9/21	Tue 4/13/21	3 days
158	●	📌	Section 8	Jason	Fri 4/9/21	Tue 4/13/21	3 days
159	●	📌	Section 9	Jay	Fri 4/9/21	Tue 4/13/21	3 days
160	●	📌	Section 10	Jay	Tue 4/13/21	Wed 4/21/21	7 days
161	●	📌	Section 11	Amanda	Tue 4/13/21	Wed 4/21/21	7 days
162	●	📌	Section 12	All	Wed 4/21/21	Sat 5/1/21	9 days
163	●	📌	Section 13	All	Wed 4/21/21	Sat 5/1/21	9 days
164		📌	**Review/Inspect**		**Sat 5/1/21**	**Wed 5/5/21**	**4 days**



ID	ℹ	Task Mode	Task Name	Assigned	Start	Finish	Estimated (hours)
165	●	📌	Double check d	All	Sat 5/1/21	Sat 5/1/21	1 day
166	●	📌	Proofread	Gordon	Sat 5/1/21	Wed 5/5/21	4 days
167		📌	**Post Document/V**		**Wed 5/5/21**	**Wed 5/5/21**	**1 day**
168		📌	Upload to NYU	Amanda	Wed 5/5/21	Wed 5/5/21	1 day