COMPUTER SCIENCE AND ENGINEERING

Happy Budget

System Requirements and Analysis Specification (SRS)

Version 4.0

Document Number SRS-004

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	10/08/2020	On File
Gordon Lei	Author	10/08/2020	On File
Jason Li	Author	10/08/2020	On File
Jay Kang	Author	10/08/2020	On File
Amanda Lin	Author, Poster	10/22/2020	On File
Gordon Lei	Author	10/22/2020	On File
Jason Li	Author	10/22/2020	On File
Jay Kang	Author	10/22/2020	On File
Amanda Lin	Author, Poster	11/19/2020	On File
Gordon Lei	Author	11/19/2020	On File
Jason Li	Author	11/19/2020	On File
Jay Kang	Author	11/19/2020	On File
Amanda Lin	Author, Poster	02/17/2021	On File
Gordon Lei	Author	02/17/2021	On File
Jason Li	Author	02/17/2021	On File
Jay Kang	Author	02/17/2021	On File

REVISION LEVEL

Date	Revision Number	Purpose
10/08/2020	Version 1.0	Initial Release
10/22/2020	Version 2.0	Added sections 6, 7, and 8
11/19/2020	Version 3.0	Added sections 9 and 13
02/17/2021	Version 4.0	Team name change, adjusted document for Senior Design

Table of Contents

1. INTRODUCTION

Purpose

The purpose of this document is to provide the system requirements and analysis specification of Happy Budget, an educational personal budgeting application, by clearly defining the domain, requirements, and analysis of the system. This document describes the scope, overall system, business requirements, functional and non-functional requirements, and system model of the application. Additionally, this document specifies the system test plan requirements, qualification provisions, requirements traceability, and how this document may evolve. The intended audience of this document includes mobile application store owners and teachers. Other intended audience include developers of the system to reference in its design and implementation, the software quality group, and the project management team.

2. SCOPE

Happy Budget is an educational personal budgeting application that is intended to be an educational, interactive personal finance web application that will provide an outlet to alleviate the learning gap for personal finances. The application aims to teach children and young adults the importance of having healthy spending habits and help prevent them from developing financially unhealthy habits while fostering sound financial decisions through an interactive, intuitive, and enticing way. The application is intended for individual use by adolescents and young adults. The standalone web application will include the following functions:

- Connection to a personal bank account or allow for user input of personal finance data
- Tracking and statistical analysis of money usage
- Interactive play
- Addition/Deletion, tracking, and statistical analysis of personal goals
- Provide finance tips

The system will, if possible, take advantage of existing digital financial bank interfaces to gather information about the user's standing balance and allow the user to access and use their money. Otherwise, the system will allow for user input of personal finance data. The

system will not run on a new, different device, but on at least one existing 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.

2.1 Identification

Happy Budget System Requirements and Analysis Specification (SRS), Version 4.0

2.2 Bounds

The application 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 interfaces for financial bank connections will be utilized and associated rules and guidelines of the institution will be abided by. Alternatively, if this connection is not supported, the application must allow for user input of personal finance data.

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 interfaces or user inputted finance data, 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 an interactive pet. The interactive pet will be available when the user puts money into their goal. Any amount of money put into the user's goal will allow the user to interact with the pet, such as giving the pet a treat.

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.

2.3 Objectives

Software Analysis/Requirements Specification (SRS) - 02/17/2021

Software Project Management Plan (SPMP) - 03/01/2021

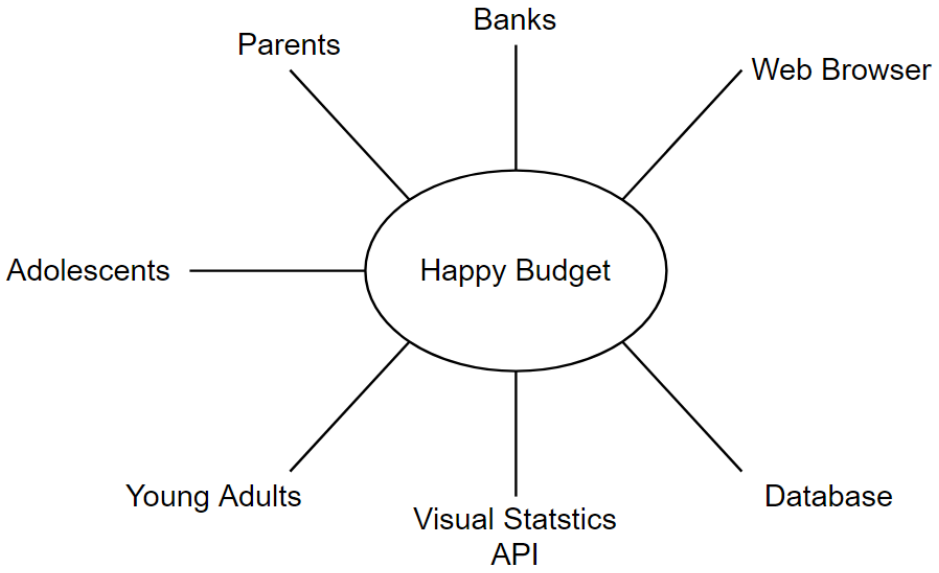
Software Design Document (SDD) - 03/08/2021

Implementation/Demonstration (code and full documentation) - 05/05/2021

Presentation – 05/10/2021

3. OVERALL SYSTEM OVERVIEW

3.1 Context Diagram



The application will interface with financial banks to allow users to see their total balance and interact with their money. The application must abide by and display the rules associated with the personal bank account. If this connection is not supported, the application must allow for user input of personal finance data.

The application will be able to interface with at least one type of browser (Google Chrome).

The application will incorporate a visual statistics API to allow users to visually see their personal data on the user's spending habits, what they spend their money on, and their goals progress. Additional visual statistics may include comparisons between the current week's spending and the past week's spending, and future savings.

A database service will be utilized to store sensitive user information and data such as a user's bank account information and a user's spending. Personal financing tips will also be gathered into a database.

3.2 Additional Descriptive Items

A list of major functions includes, but is not limited to:

1. Linking and accessing bank accounts (if supported): Users will be given the option to link their bank account with the application. By doing so, they can interact with their bank account and perform transactions such as withdrawing money. This also gives the application more accurate data corresponding to their financial situation so the application can give better suggestions or data visualization to the user.

2. Input user data: In the case where the user does not link a bank account or a connection between bank accounts and the application is not supported, users can add imaginary accounts to the application to represent their actual bank account and then input in data or numbers to represent their transactions.

3. Data visualization: The application will represent spending trends of the users, estimated amount of money saved, and other such statistical data in various graphs.

4. Interactive play and creating personal goals: Users will be able to create goals to incentivize themselves to save money to reach the goal. There will also be an interactive pet feature that activates if the user puts money toward their goal.

5. Financial Tips: Throughout the application's lifetime, financial tips will appear when appropriate, such as when a user spends more than they are saving.

The intended users of this product are young adults, but the application may be used by anyone. Young adults should typically begin saving money to build up a habit of having enough money to pay for bills or potential loans (such as college tuition). However, young adults are often not taught about personal finance as there are usually no classes for these topics in high school or college. The users do not need to have vast technical expertise; at most they will need to know how to use a phone or navigate through a website as the application will be hosted online.

Constraints that can limit the developer's options include:

1. Developers must follow FDIC guidelines in order to make sure they are not doing illegal actions.

2. Developers may not be able to connect bank accounts to the application as banks may not give permission to do so.

Assumptions and dependencies that can affect the requirements include:

1. The application is built to run on Google Chrome.

Requirements subsets may be further developed in later releases. Requirement subsets include:

1. Define required operating systems and other such dependencies on the application
2. Outline the requirements and key features
3. Create a “skeleton” project where user can input and delete own data
4. Be able to see trends and data in graphs and other such data visualization
5. User will be able to see tips on how to effectively use and save their money depending on their situation
6. Add concept of interactive play
7. Add bank connectivity and allow transactions to be made related to those bank accounts
8. Make sure bank connectivity can do the features mentioned above to substitute inputting own data
9. Add glossary section for relevant terminology

4. DOCUMENT OVERVIEW

This document specifies the system requirements and analysis for Happy Budget, a web-based interactive personal budgeting application. This document begins with a general overview of the system and leads into a more in-depth description and analysis of the system’s requirements and architecture. At the end, this document specifies organizational requirements such as test plans, qualification provisions, traceability, and evolution.

The following sections contain an in-depth description of the related business requirements, risks, and alternatives, functional and non-functional requirements, analysis of the system component architecture, system test plan requirements, and qualification provisions, requirements traceability, and evolution.

5. REFERENCE DOCUMENTS

1. Team B27, Project Proposal for Happy Budget, Document Number 001, Version 1.0, 02/06/2021.
2. Team B27, Project Management Plan for Happy Budget, Document Number 002, Version 1.0, 11/05/2020.

6. BUSINESS REQUIREMENTS

6.1 Technology

Technology allows this application to reach people in areas without proper facilities to receive quality financial education. Additionally, machine learning and artificial intelligence will be useful in making sure each user's experience is tailored and personalized. By observing how the user is using the application and tracking their spending and savings, the application can offer more relevant information and advice on how the user can save money and spend money wisely.

6.2 Economics

The economic driver for the application is in the form of ad revenue or subscription based. The application will be free to use but will feature unobtrusive ads that will generate revenue so that the application can be constantly updated and improved upon. Users may choose to instead pay a small fee to get rid of ads if they wish so.

6.3 Regulatory and Legal

As the software connects with the user's bank account, there can be dangers of sensitive information being leaked. The application must never compromise the privacy of the user, for ethical reasons as well as the legal complications it can ensue. The application must also make sure to follow the rules and guidelines of the financial institutions.

6.4 Market Considerations

The market audience of this application is primarily towards young adolescents and their parents who wish to develop good financial skills. The market the application can indirectly support is any market that deals with more expensive, long-use products, such as electronics, as the application encourages users to control constant, cheap spending for a long-term goal.

6.5 Risks and Alternatives

Risk Table:

Business Risk: "Going out of business"

Description: Members of the group might need to delay taking the software development class due to unforeseen circumstances, dismantling the team.

Probability: Very unlikely

How discovered: Communication between team members

Responsible Party: Any team members

Status: Not present

Mitigation Plan: Acquire new group members, join a new project

Operational Risk: GitHub Merge Conflicts

Description: When working on the same GitHub repo, it is possible that the repos on our local machine are not synced as people have pushed changes onto the repo. If anyone else tries to push their changes, a merge conflict may occur.

How discovered: Merge problems are always a consideration when working with GitHub

Probability: Likely

Responsible Party: Developers

Status: Not present

Mitigation Plan: Pull the repo before adding or committing to the repo or manually fix the merge conflict.

Technology Risk: Bank Account conflict

Description: Users may not able to connect their bank account to the application because the banks do not allow this or have a usable API that will allow them to do so.

Probability: Very likely

How discovered: Could be a potential problem as banks are usually hesitant to give out access to account information

Responsible Party: Banks

Status: Not present

Mitigation Plan: Users can input their own data into the system to simulate their bank account

Technological Risk: Database Security

Description: Because users will be making accounts, user account data must be stored and secured so their data will not get stolen.

Probability: Likely

How discovered: User data should always be protected.

Responsible Party: Developers

Status: Not present

Mitigation Plan: Developers can create a system of 2FA or OAUTH to make accounts more protected, as well as using salting user information.

6.6 Human Resources and Training

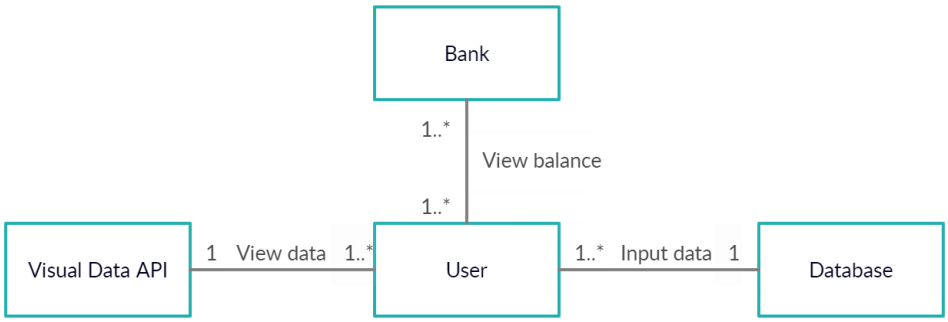
A training session for the technician who takes calls and inquiries from users having trouble with the application will help both the customer experience and the technician.

7. SPECIFIC REQUIREMENTS (DESCRIPTIVE FUNCTIONAL REQUIREMENTS)

The following principles should be applied:

1. Specific requirements should be stated in conformance will all the characteristics described in earlier documents (for example, the proposal) or earlier sections of the document
2. Specific requirements should be cross-referenced (traceable) to earlier documents that they relate
3. All requirements should be uniquely identifiable
4. Careful attention should be given to organizing the requirements for readability and understandability

7.1 Functional Descriptive Detailed Requirements



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.

7.2 Requirement Use Cases

The actors in the use cases for this system are:

- User, the user using the system
- Bank Interface, the interface 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:

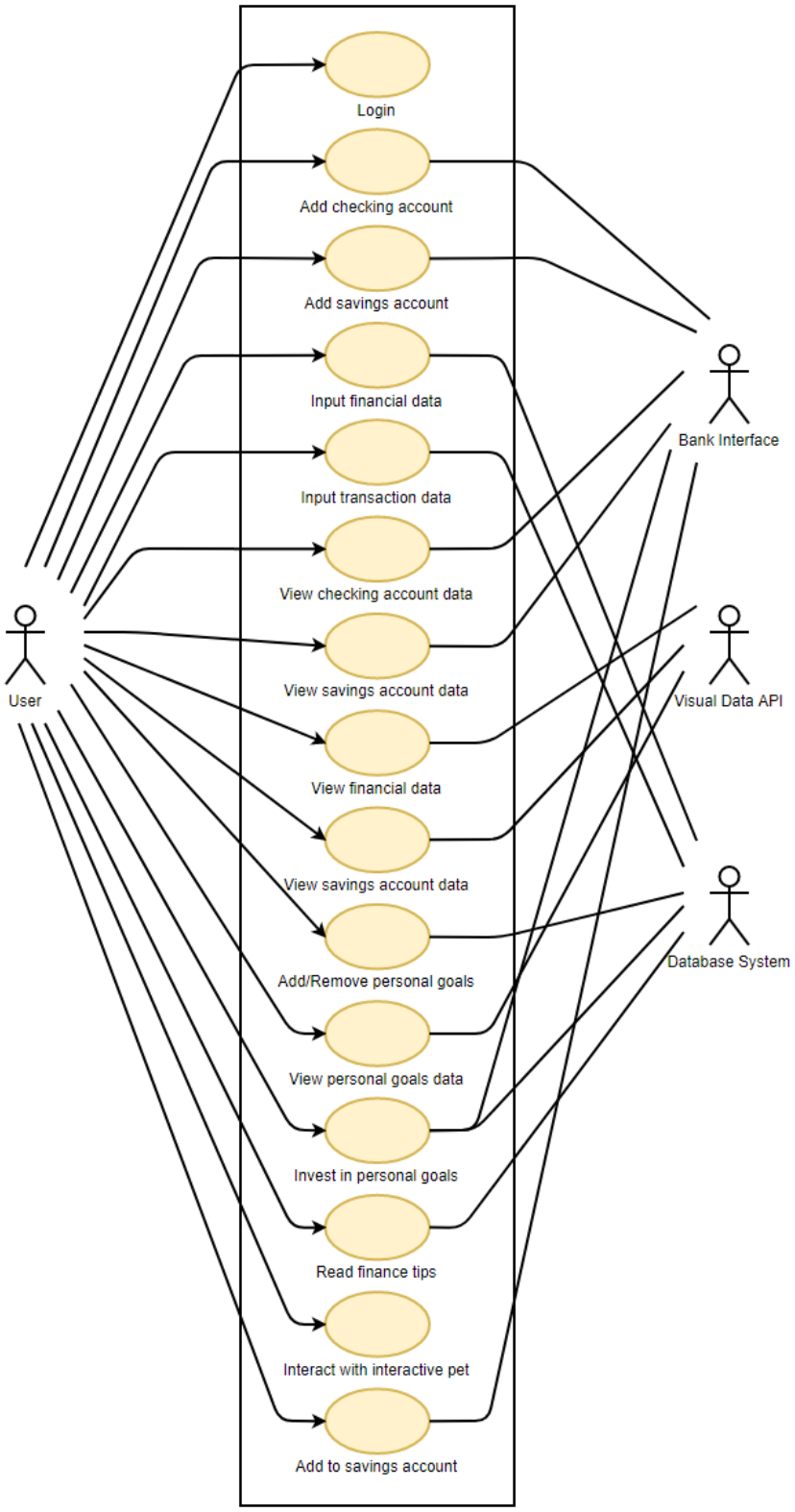
- Add a checking account between the User and the system, and the system and the Bank Interface (1.1)
- Add a savings account between the User and the system, and the system and the Bank Interface (1.1)
- Input financial data between the User and the system, and the system and the Database System (1.2)
- 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 Interface (1.3)
- View savings account balance between the User and the system, and the system and the Bank Interface (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 Interface (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 Interface (1.4)

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

- Login between the User and the system (8.1)

7.2.1 Use Case Diagrams



7.2.2 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 Interface.	
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 Interface 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 Interface.	
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 Interface 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 Interface. 2. Validate Connection

		The Bank Interface will validate the connection between the system and the Bank Interface 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 Interface connection.
	Alternative Flows	A1. Invalidated Connection If in Step 2, Validate Connection, the Bank Interface 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 Interface. 2. Validate Connection The Bank Interface will validate the connection between the system and the Bank Interface 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 Interface connection.
	Alternative Flows	A1. Invalidated Connection If in Step 2, Validate Connection, the Bank Interface 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 Interface or from the User's Input. 2.A. Request Checking Account Data If the User's data is from the Bank Interface, the system will request a connection to the corresponding Bank Interface. 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 Interface will validate the connection from the system to either the Bank Interface 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 Interface 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 Interface. 2. Validate Connection The Bank Interface will validate the connection from the system to the Bank Interface 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 Interface 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 Interface 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 Interface, the system will attempt to complete the transaction through the Bank Interface 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 Interface 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 Interface, the system will attempt to complete the transaction through the Bank Interface 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	

8. NON-FUNCTIONAL DESCRIPTIVE DETAILED REQUIREMENTS

6. Product requirements

6.1 The application shall be available at all times. In the event that a maintenance or a downtime of the server is necessary, an announcement should be made at least a week before the downtime.

6.2 The application shall maintain privacy of the user's usage data if the user opts out of sharing their usage data.

6.3 The amount of memory required for this application will be finalized in a later release.

7. Organizational requirements

7.1 People shall be identified by their account names.

7.2 Programming language(s) to be determined in a future release.

7.3 The operating system(s) that the application will require will be determined in a future release.

8. External requirements

8.1 The application shall allow the user a secure way to create and access their personal account.

8.2 The application shall keep the user's bank account information secure in accordance to law.

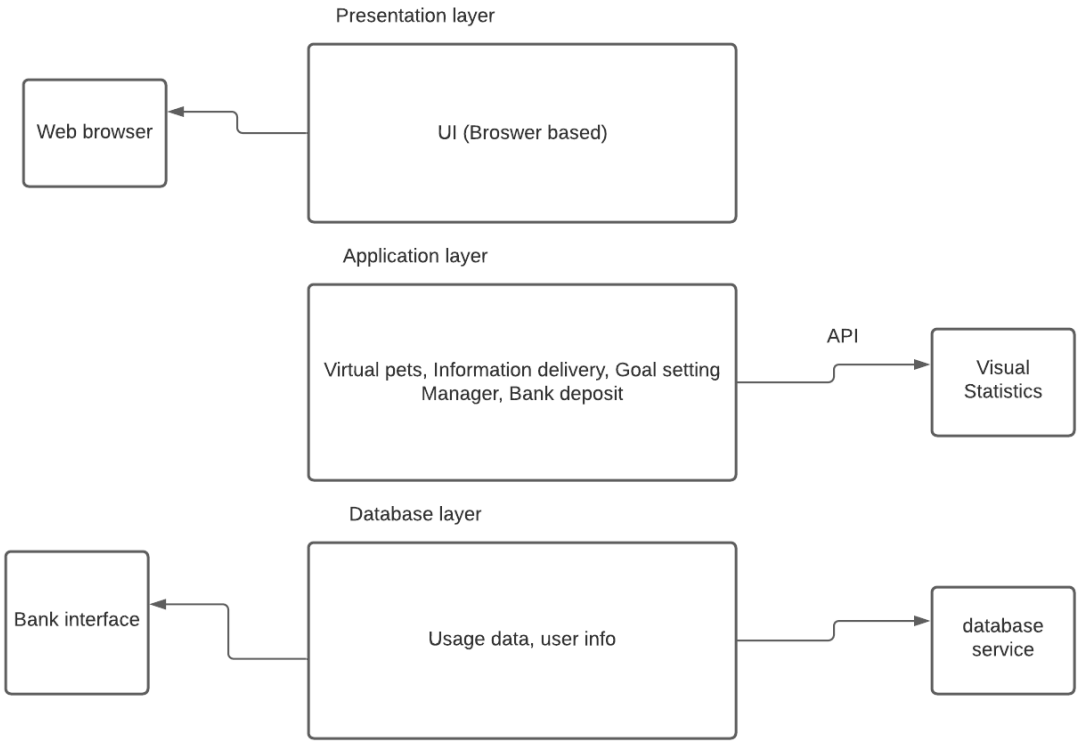
9. SYSTEM MODEL (UML)

9.1 Component (Component/Package/Subsystem) Architecture

This system's component architecture follows a three-tiered architecture consisting of a user interface or presentation layer, an application layer, and a database layer.

Reusable components in the system include the visual statistics API, the database service, the web browser, and the bank interface.

9.2 Component Architecture Diagram



9.3 Component Descriptions

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.

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.

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

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

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.

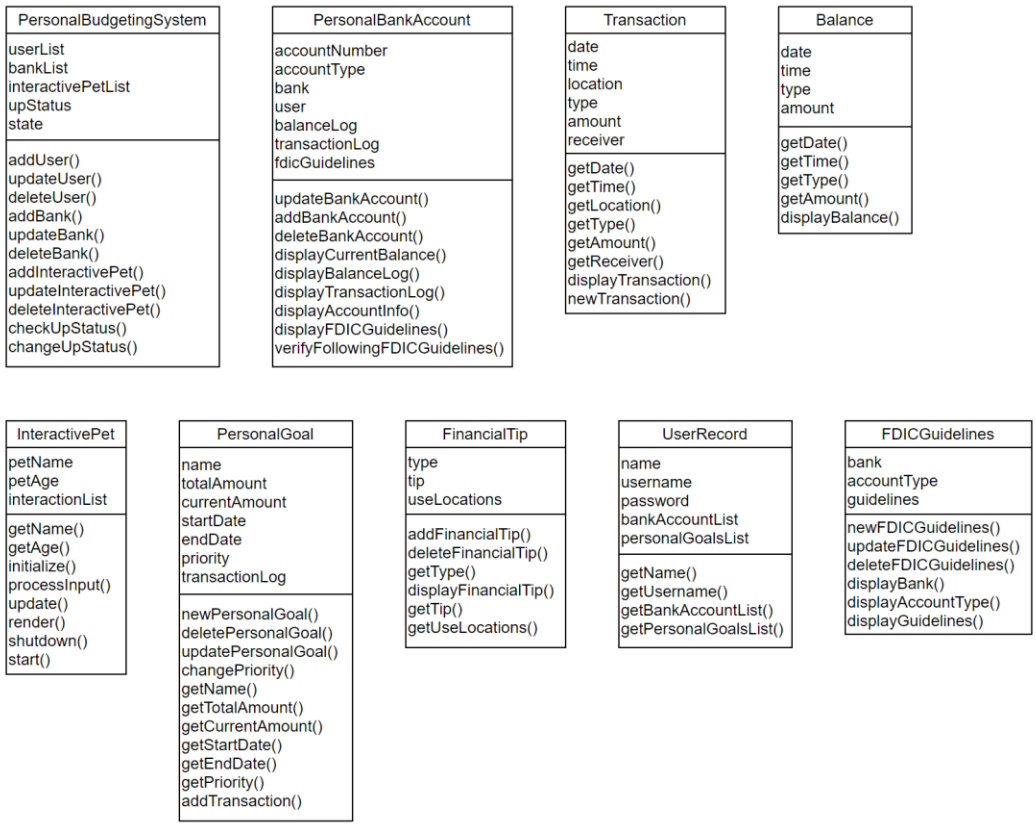
User info storage: A database that will store user's information. These information range from login detail, personal information (names, bank account), and user input such as goals, savings plans, and tags to determine what info to show the user will also be stored in this database. The database will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

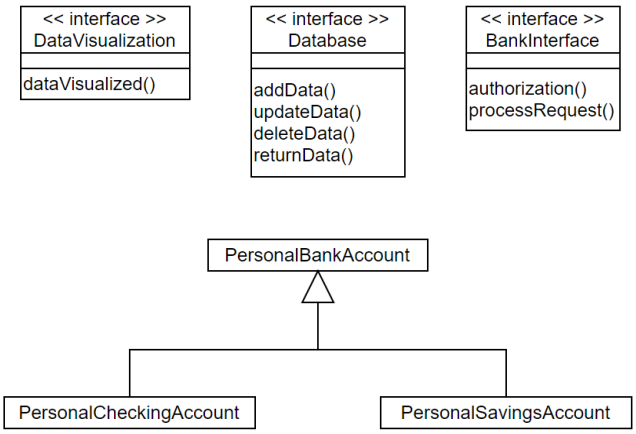
Bank deposit: The user will be able to connect to their bank account from the application. When the user makes deposits or bank transactions from the app, the application will send a secure, and quick request to the bank to carry out those transactions.

9.4 Class Diagrams

9.4.1 Individual Class Diagrams

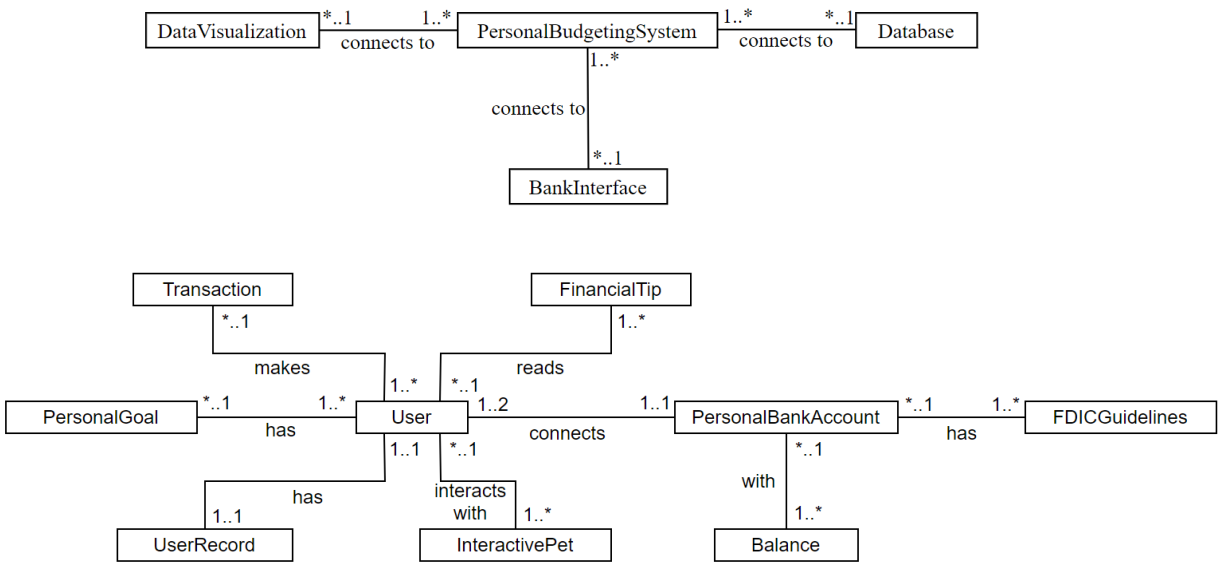
Class diagrams:





9.4.2 Class Relationship/Interaction Diagrams

Class interaction diagram:



9.5 Events

The system responds to the following events:

- 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)
- View 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)
- Display finance tips (4)
- Interact with interactive pet (5)
- Add to savings account (1.4)

9.5.1 Motives

Event: Add checking account (1.1)

Motive: The user wants to add their checking account to the system.

Event: Add savings account (1.1)

Motive: The user wants to add their savings account to the system.

Event: Input financial data (1.2)

Motive: The user wants to add financial data to the system that is not already in the system.

Event: Input transaction data (1.2)

Motive: The user wants to add transaction data to the system that is not already in the system.

Event: View checking account balance (1.3)

Motive: The user wants to check the balance in their checking account.

Event: View savings account balance (1.3)

Motive: The user wants to check the balance in their savings account.

Event: View financial data (2.1)

Motive: The user wants to view their financial data in the form of visual statistics.

Event: View savings account data (2.1)

Motive: The user wants to view their savings account data in the form of visual statistics.

Event: Add/Remove personal goals (3.1)

Motive: The user wants to add or remove personal goals to or from their list of personal goals.

Event: View personal goals data (3.3)

Motive: The user wants to view financial data about their personal goals in the form of visual statistics.

Event: Invest in personal goals (3.2)

Motive: The user wants to invest in their personal goals.

Event: Display finance tip (4)

Motive: The user does an action that results in a finance tip appearing to educate the user.

Event: Interact with interactive pet (5)

Motive: The user does an action that results in a chance to interact with an interactive pet to incentivize the user to perform more actions that are good for their personal finances.

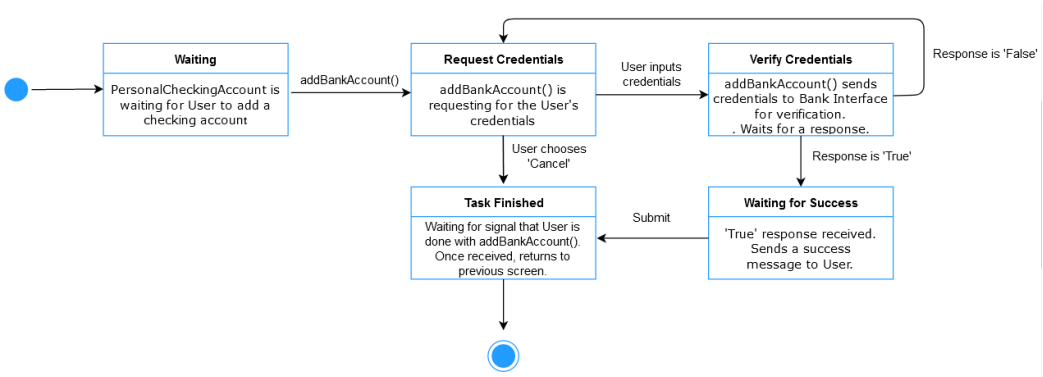
Event: Add to savings account (1.4)

Motive: The user wants to add money to their savings account.

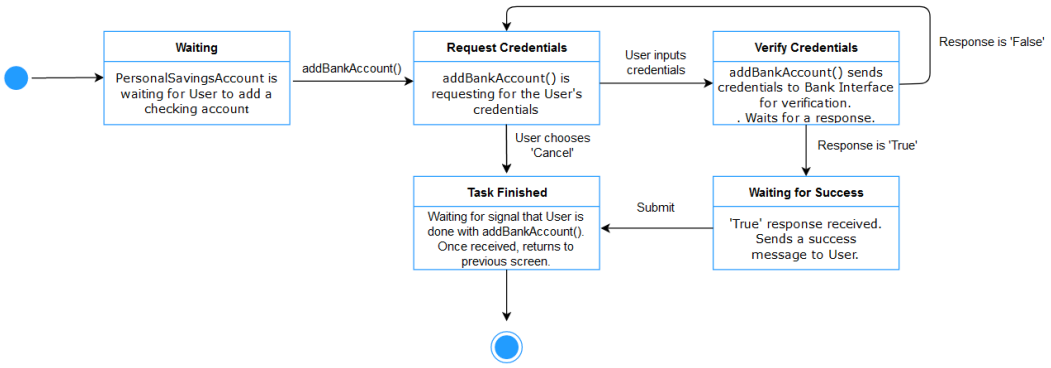
9.5.2 Event Diagrams

Event Diagrams:

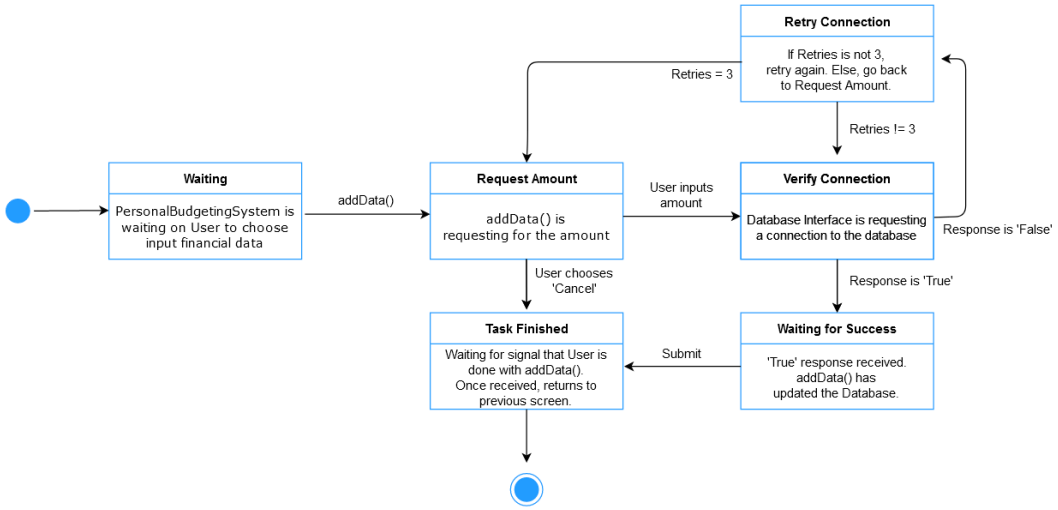
Add Checking Account (1.1)



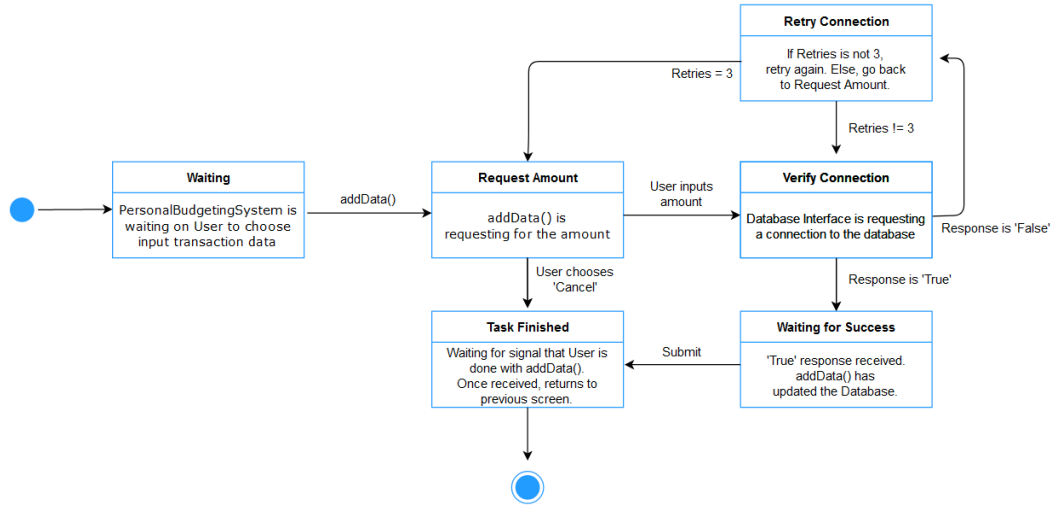
Add Savings Account (1.1)



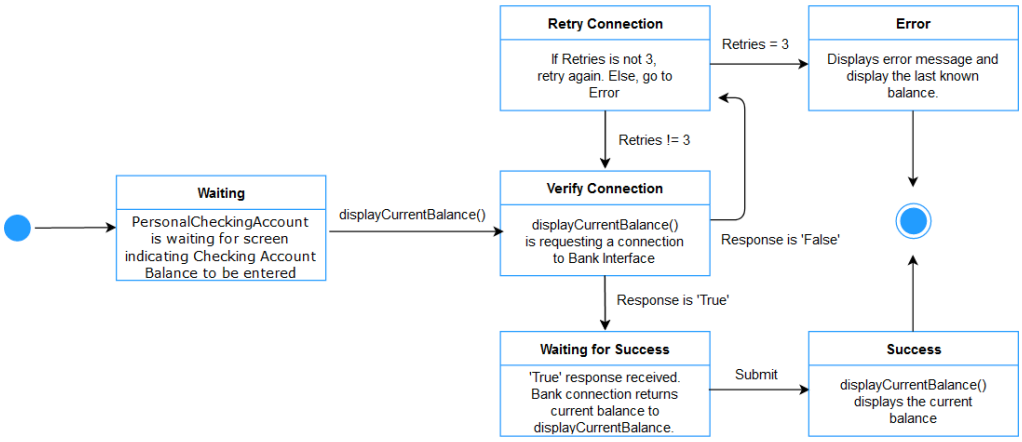
Input Financial Data (1.2)



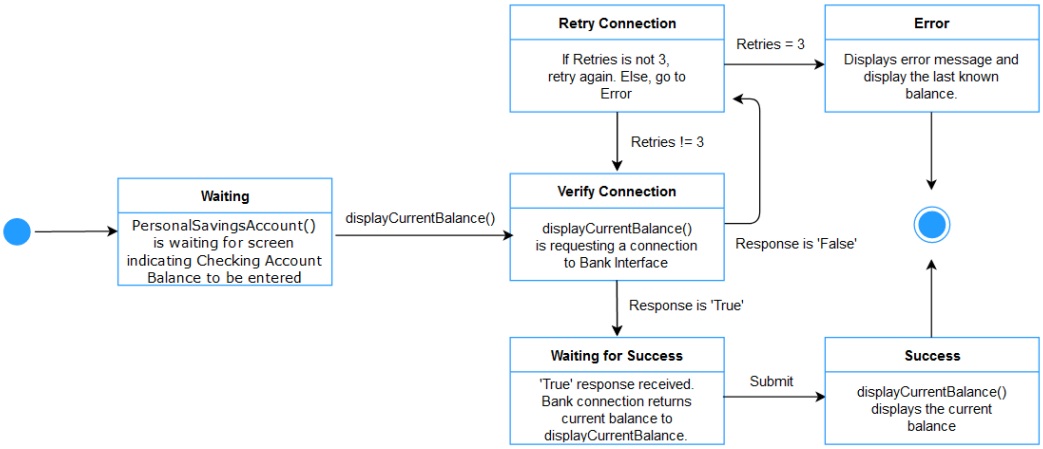
Input Transaction Data (1.2)



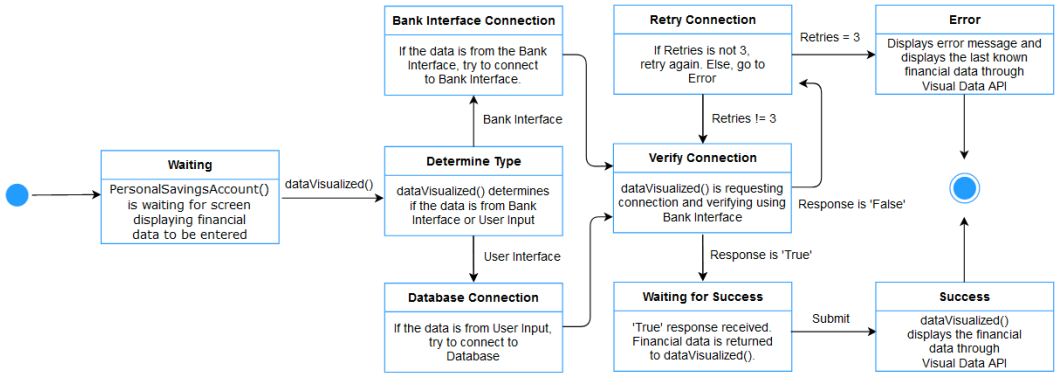
View Checking Account Balance (1.3)



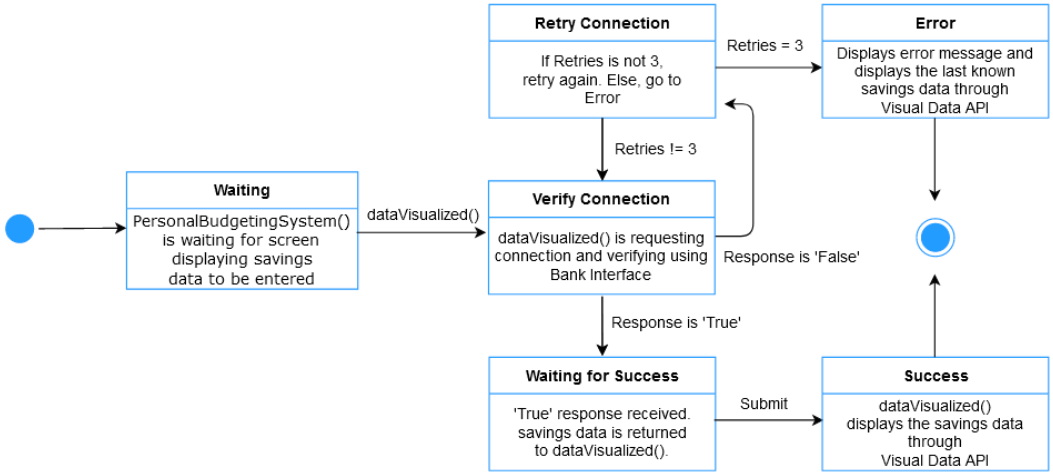
View Savings Account Balance (1.3)



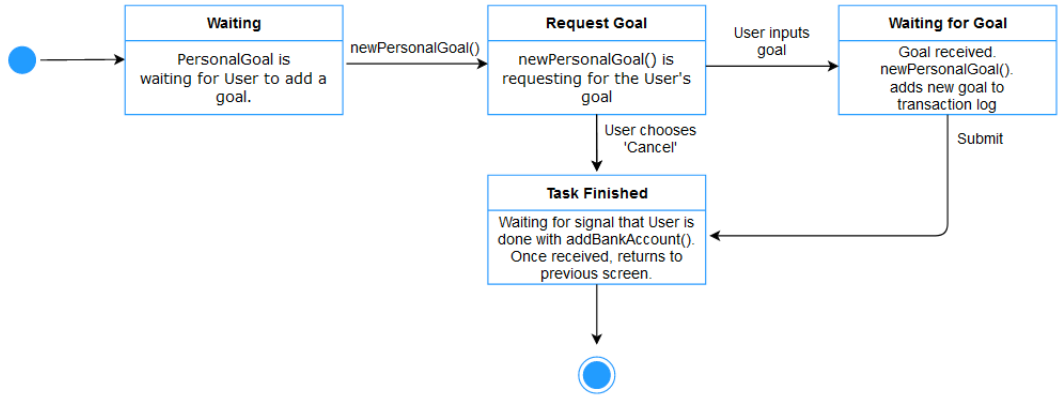
View Financial Data (2.1)



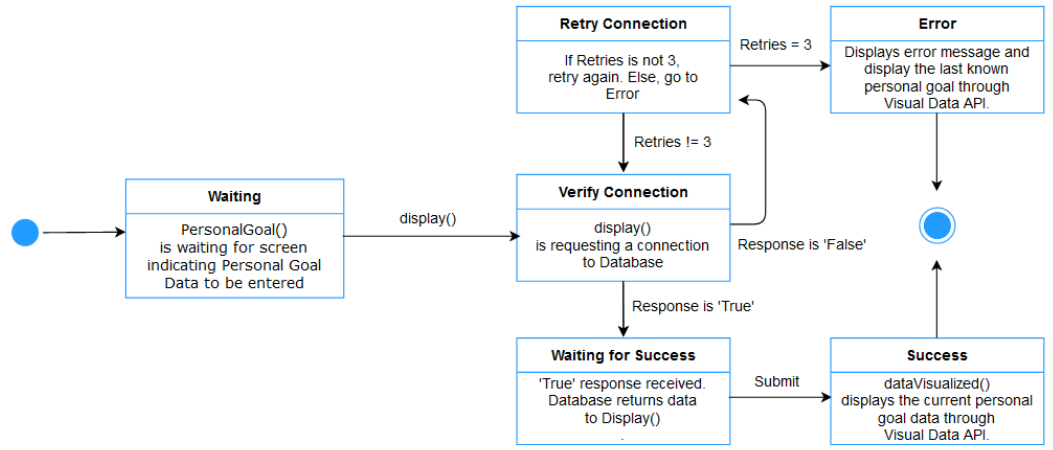
View Savings Account Data (2.1)



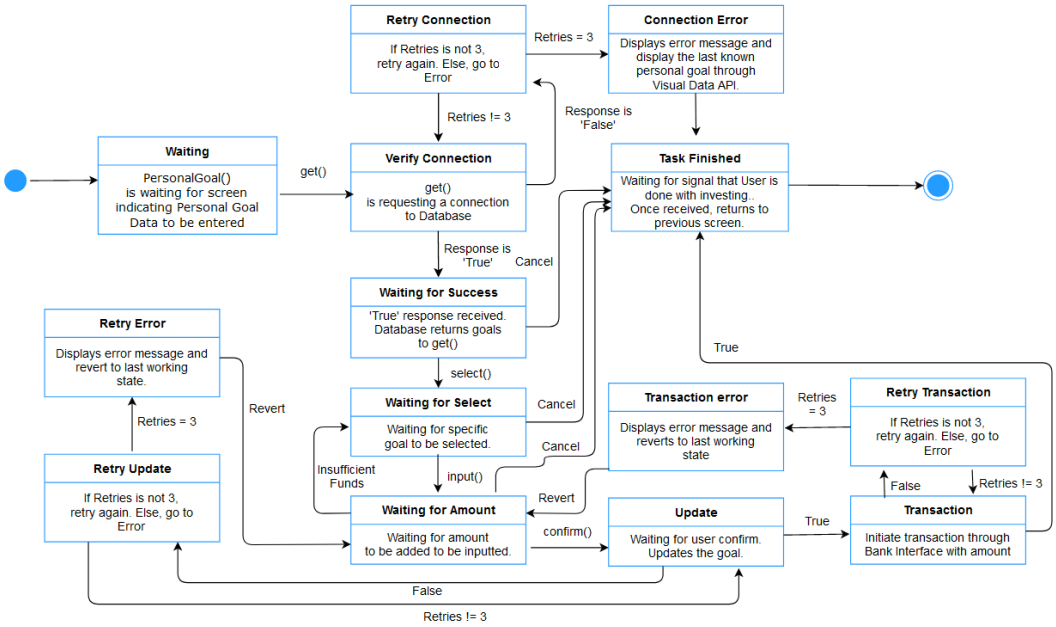
Add/Remove Personal Goals (3.1)



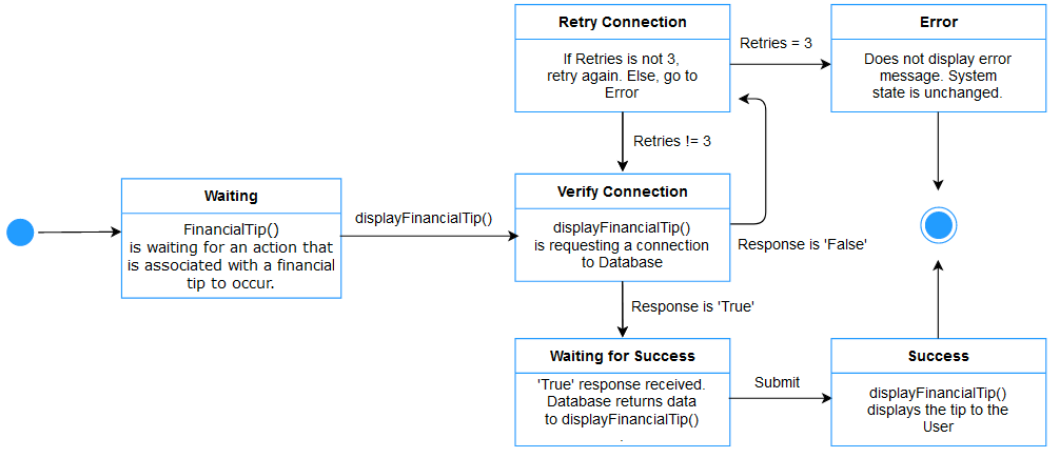
View Personal Goals Data (3.3)



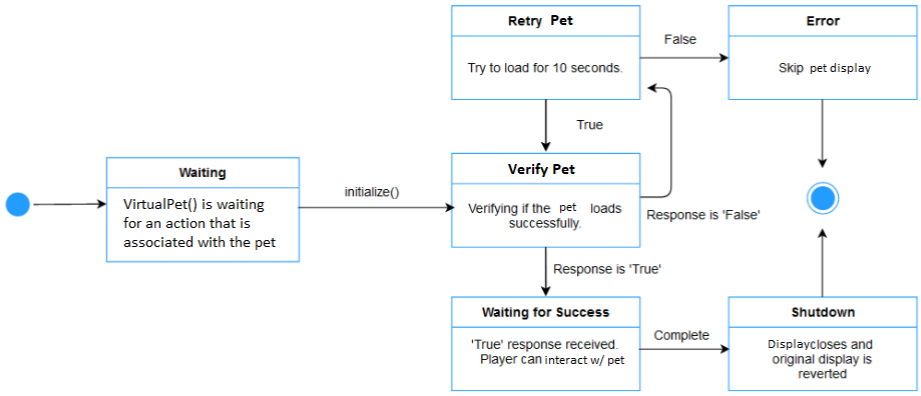
Invest in Personal Goals (3.2)



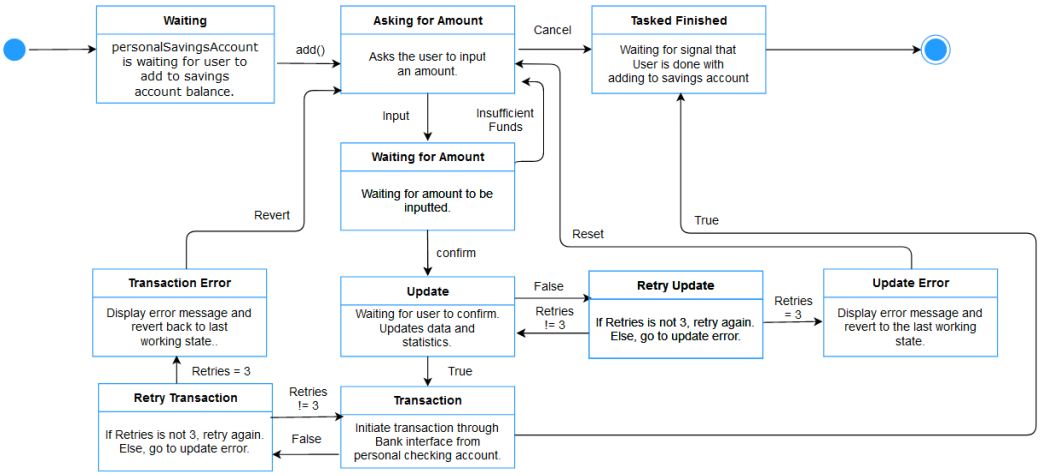
Display Finance Tips (4)



Interact with Interactive Pet (5.1)



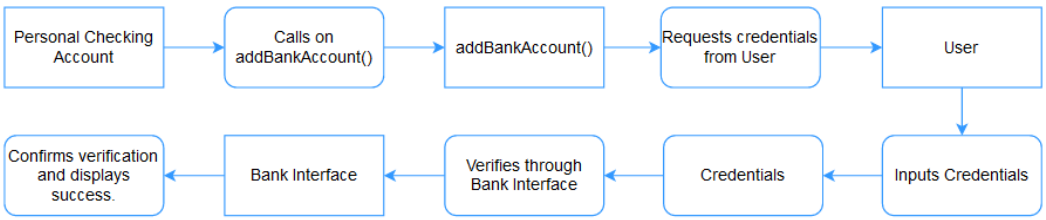
Add to Savings Account (1.4)



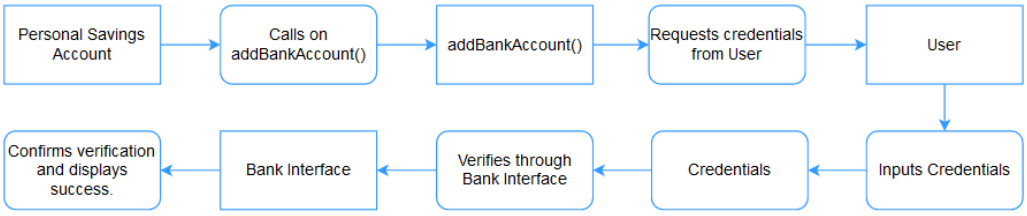
9.6 Activity/State (Scenario) Sectional

Activity Diagrams:

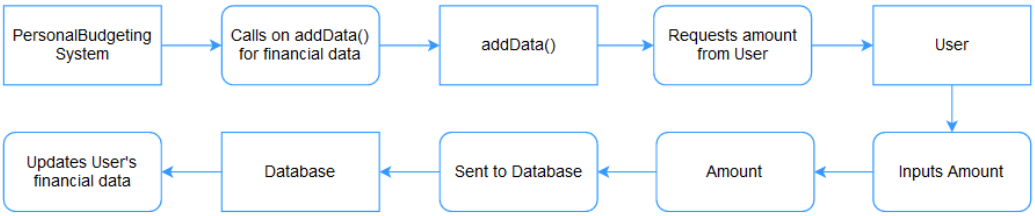
Add Checking Account (1.1)



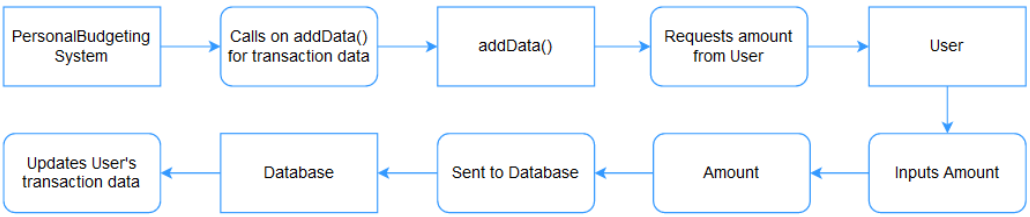
Add Savings Account (1.1)



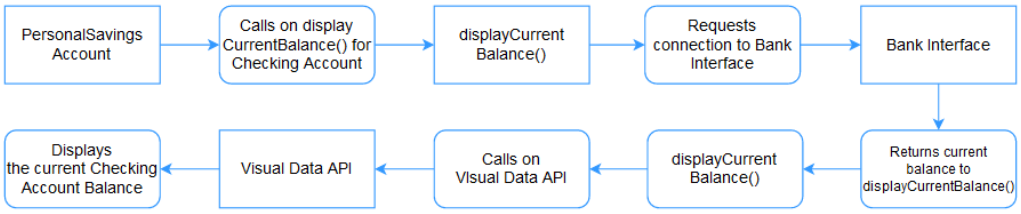
Input Financial Data (1.2)



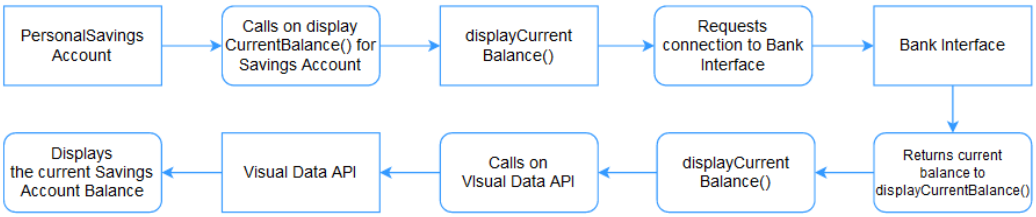
Input Transaction Data (1.2)



View Checking Account Balance (1.3)

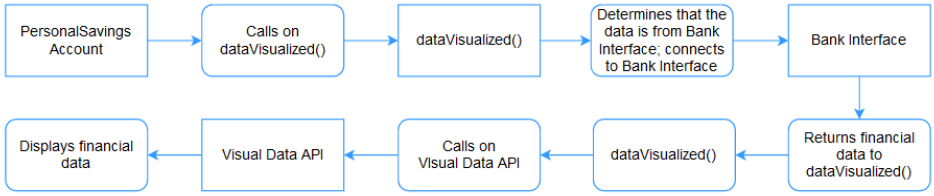


View Savings Account Balance (1.3)

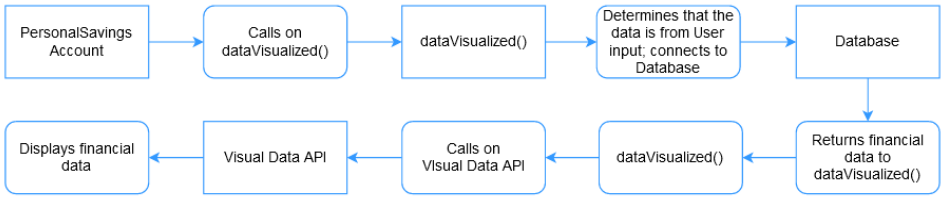


View Financial Data (2.1)

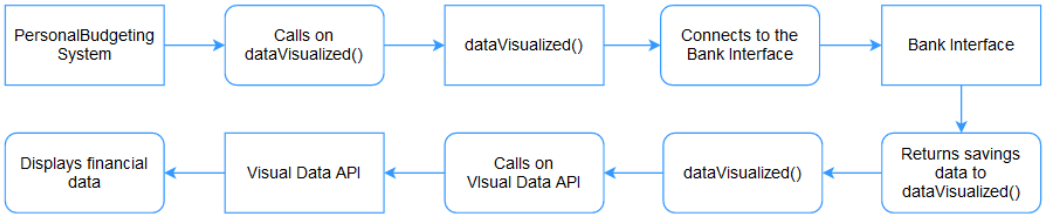
a. If data is from Bank Interface:



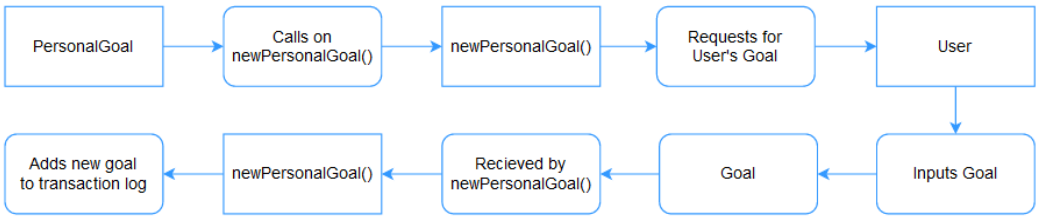
b. If data is from User Interface:



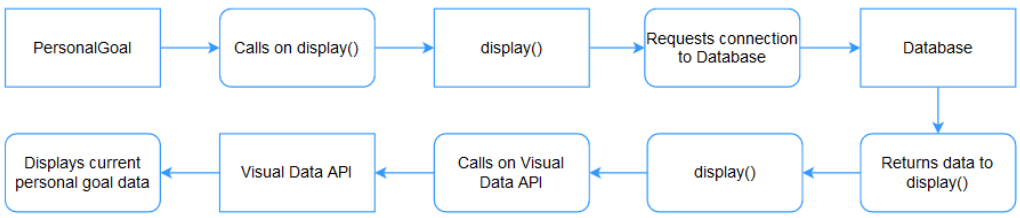
View Savings Account Data (2.1)



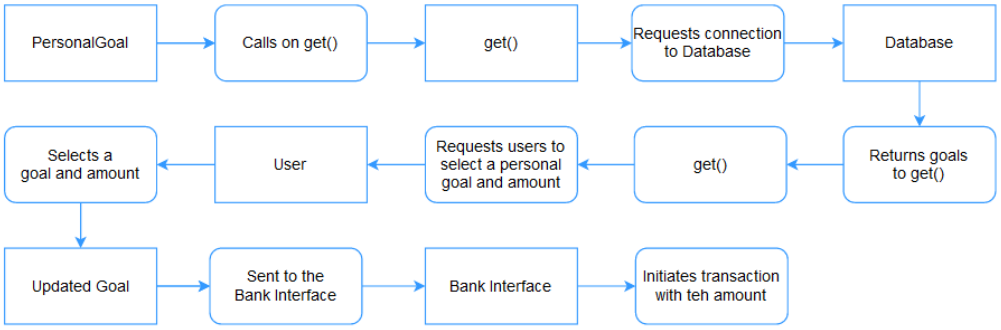
Add/Remove Personal Goals (3.1)



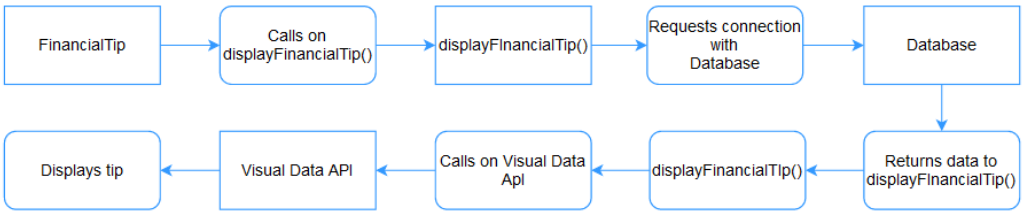
View Personal Goals Data (3.3)



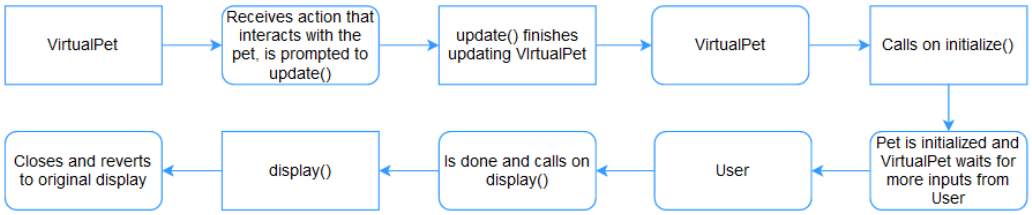
Invest in Personal Goals (3.2)



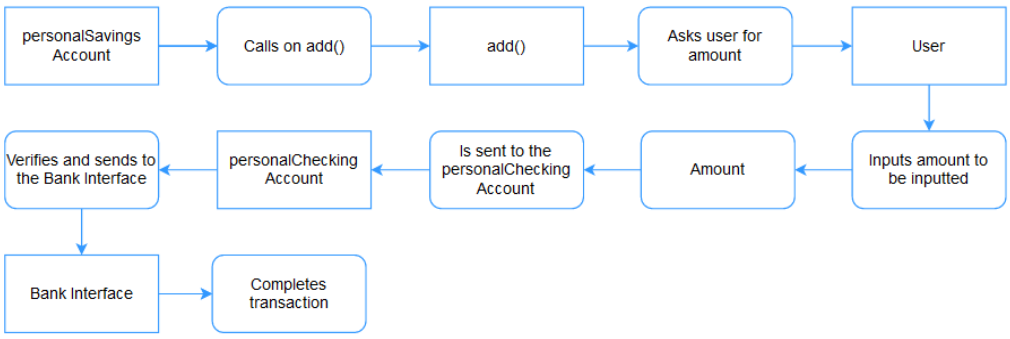
Read Finance Tips (4)



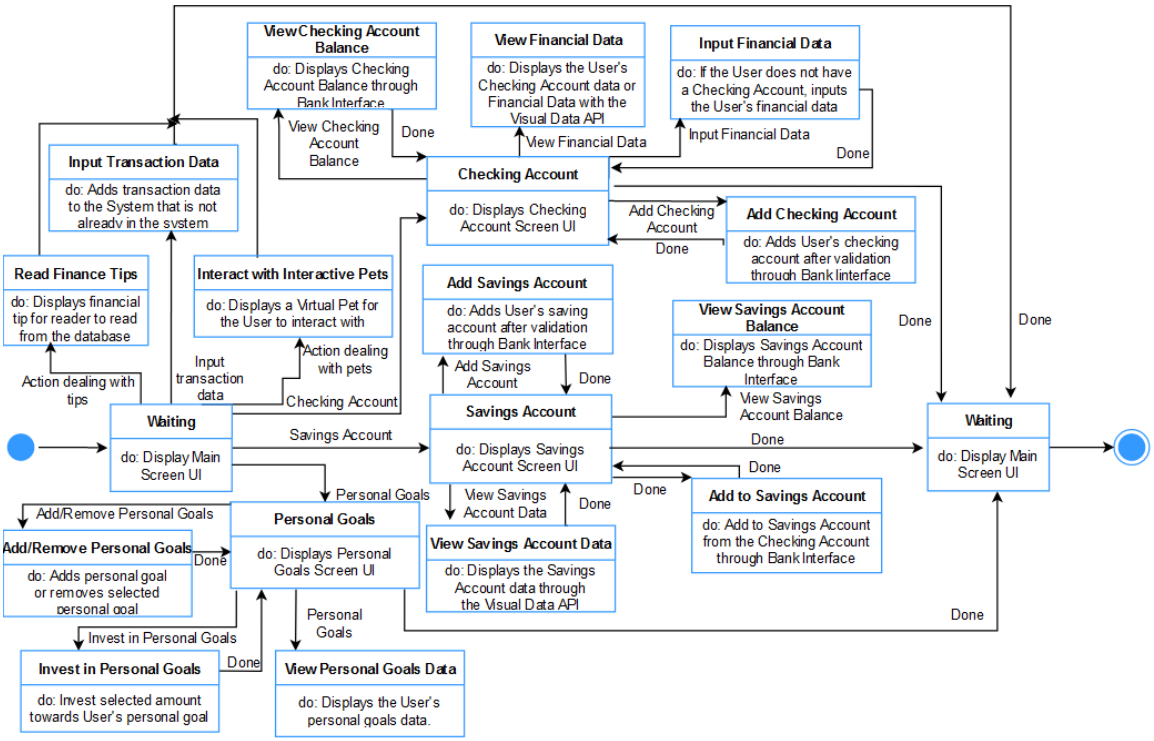
Interact with Interactive Pet (5)



Add to Savings Account (1.4)



State Diagram:



9.7 State Logic

State	Description
Waiting	The app is waiting for input. The display shows the main screen UI, with options to go to the Checking Account page, Savings Account page, Personal Goals page, and to Input Financial Data.
Checking Account	The UI is switched to the Checking Account page. The display shows options to View Checking Account Balance, View Financial Data, Input Financial Data, Add Checking Account, and to go back to the main screen UI.
View Checking Account Balance	The checking account balance is retrieved through the Bank Interface. The display shows the checking account balance.
View Financial Data	The financial/checking account data is retrieved through the Bank Interface and sent to the Visual Data API. The display shows the financial/checking account data through the Visual Data API.

Input Financial Data	The financial data is received from the User and is stored into the Database. The display shows an Input Box when the User is prompted and a Success message when the financial data is stored in the database.
Add Checking Account	The checking account credentials are received from the User before it is validated through the Bank Interface. The display shows an Input Box when the User is prompted and a Success message when the checking account is added.
Savings Account	The UI is switched to the Savings Account page. The display shows options to View Savings Account Balance, View Savings Account Data, Add to Savings Account, Add Savings Account, and to go back to the main screen UI.
View Savings Account Balance	The savings account balance is retrieved through the Bank Interface. The display shows the checking account balance.
Add Savings Account	The savings account credentials are received from the User before it is validated through the Bank Interface. The display shows an Input Box when the User is prompted and a Success message when the savings account is added.
Add to Savings Account	The amount is received from the User before it is added to the Savings Account from the Checking Account through the Bank Interface. The display shows an Input Box when the User is prompted and a Success message when the savings account is added to.
View Savings Account Data	The savings account data is retrieved through the Bank Interface and sent to the Visual Data API. The display shows the savings account data through the Visual Data API.
Personal Goals	The UI is switched to the Personal Goals page. The display shows options to Add/Remove PersonalGoals, Invest in Personal Goals, View Personal Goals Data, and to go back to the main screen UI.
Add/Remove Personal Goals	The personal goal is received from the User before it is added or the selected personal goal is received from the User before it is deleted. The display shows an Input Box when the User is prompted.
Invest in Personal Goals	The selected personal goal and amount is received from the User before it is invested into the Personal goal. The display shows an Input Box when the User is prompted.

View Personal Goals Data	The personal goals data is retrieved through the Database and sent to the Visual Data API. The display shows the personal goals data through the Visual Data API.
Read Finance Tips	The corresponding finance tip to the action is retrieved from the database. The display shows the finance tip.
Interact with Interactive Pets	The virtual pet is interacted with and reacts to the User's corresponding action. The display shows the virtual pet.
Input Transaction Data	The transaction data is received from the User and is stored into the Database. The display shows an Input Box when the User is prompted and a Success message when the transaction data is stored in the database.

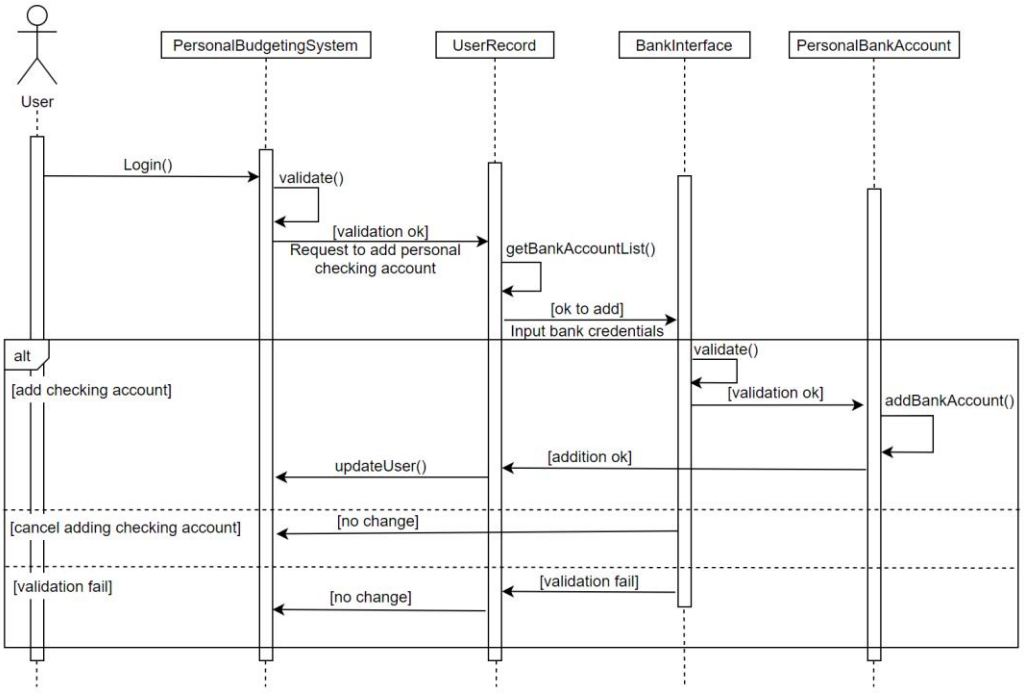
Stimulus	**Description**
Done	The User is done with the state or chooses the option to go back to the previous page.
Checking Account	The User chooses 'Checking Account' (from the main screen UI)
View Checking Account Balance	The User chooses 'View Checking Account Balance' (from the 'Checking Account' page)
View Financial Data	The User chooses 'View Financial Data' (from the 'Checking Account' page)
Input Financial Data	The User chooses 'Input Financial Data'' (from the 'Checking Account' page)
Add Checking Account	The User chooses 'Add Checking Account' (from the 'Checking Account' page)
Savings Account	The User chooses Savings Account' (from the main screen UI)
View Savings Account Balance	The User chooses 'View Savings Account' (from the 'Savings Account' page)
Add Savings Account	The User chooses 'Add Savings Account' (from the 'Savings Account' page)
Add to Savings Account	The User chooses 'Add to Savings Account' (from the 'Savings Account' page)

View Savings Account Data	The User chooses ‘View Savings Account Data’ (from the ‘Savings Account’ page)
Personal Goals	The User chooses ‘Personal Goals’ (from the main screen UI)
Add/Remove Personal Goals	The User chooses ‘Add/Remove Personal Goals’ (from the ‘Personal Goals’ page)
Invest in Personal Goals	The User chooses ‘Invest in Personal Goals’ (from the ‘Personal Goals’ page)
View Personal Goals Data	The User chooses ‘View Personal Goals’ (from the ‘Personal Goals’ page)
Action Dealing with Tips	The User performs an action that will trigger a finance tip.
Action Dealing with Pets	The User performs an action that will trigger an interaction with the virtual pet.
Input Transaction Data	The User chooses ‘Input Transaction Data’ (from the main screen UI)

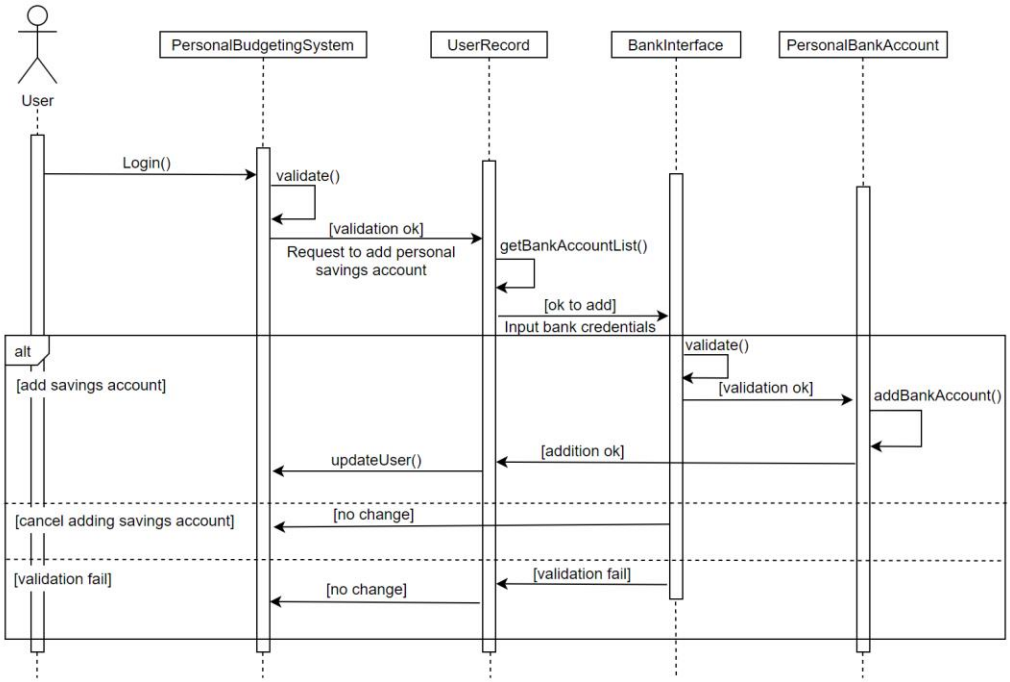
9.8 Behavior

9.8.1 Sequence Diagrams

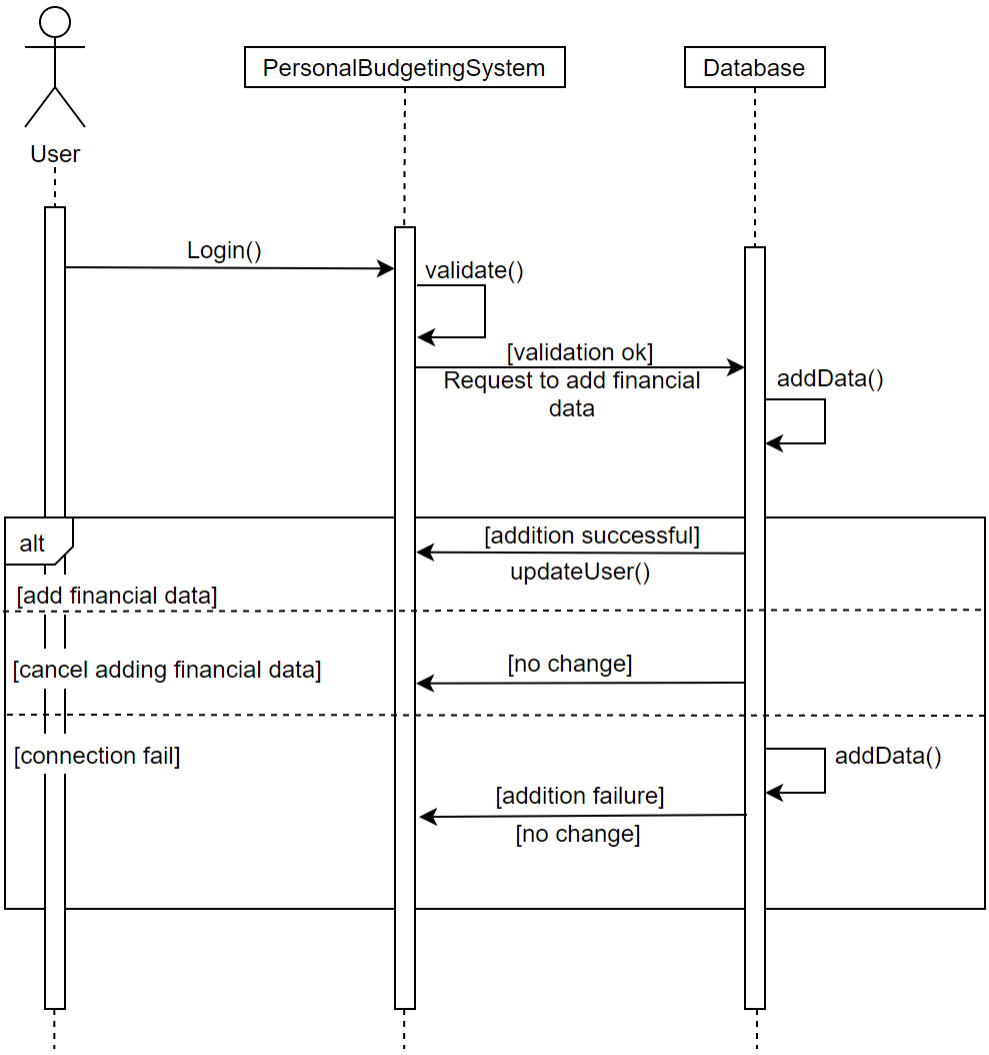
Add Checking Account (1.1)



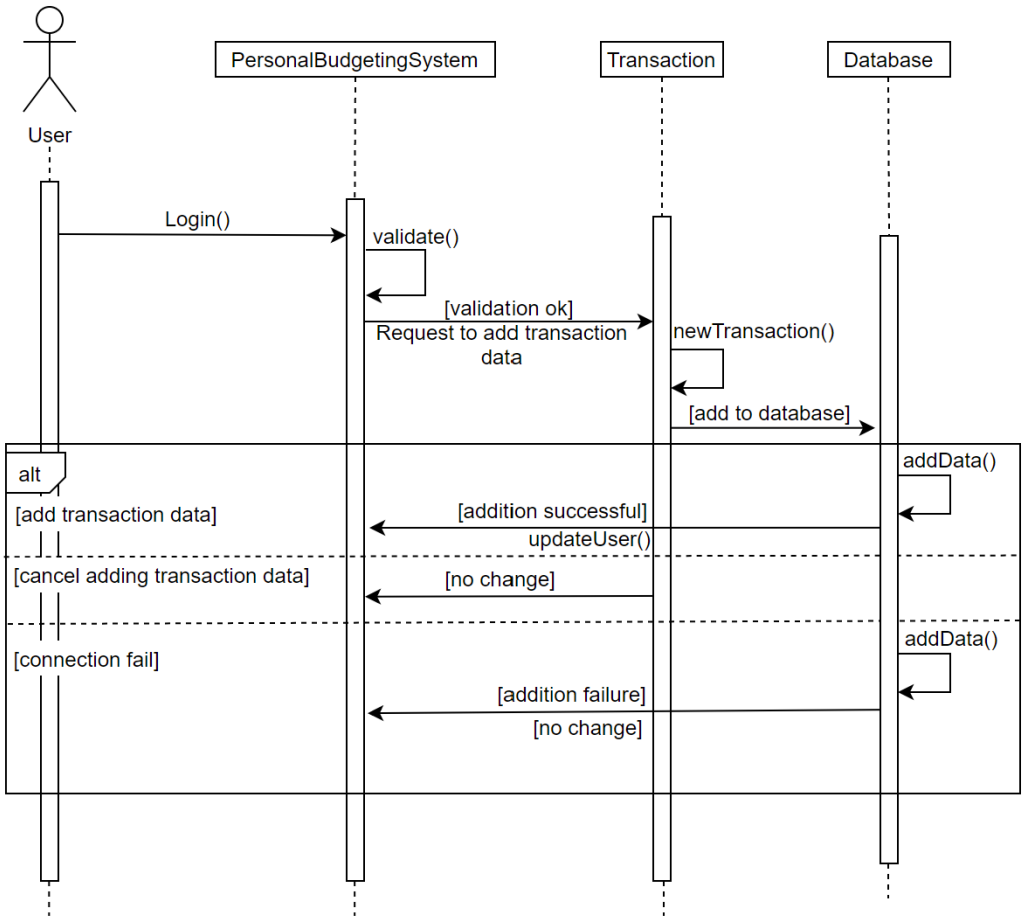
Add Savings Account (1.1)



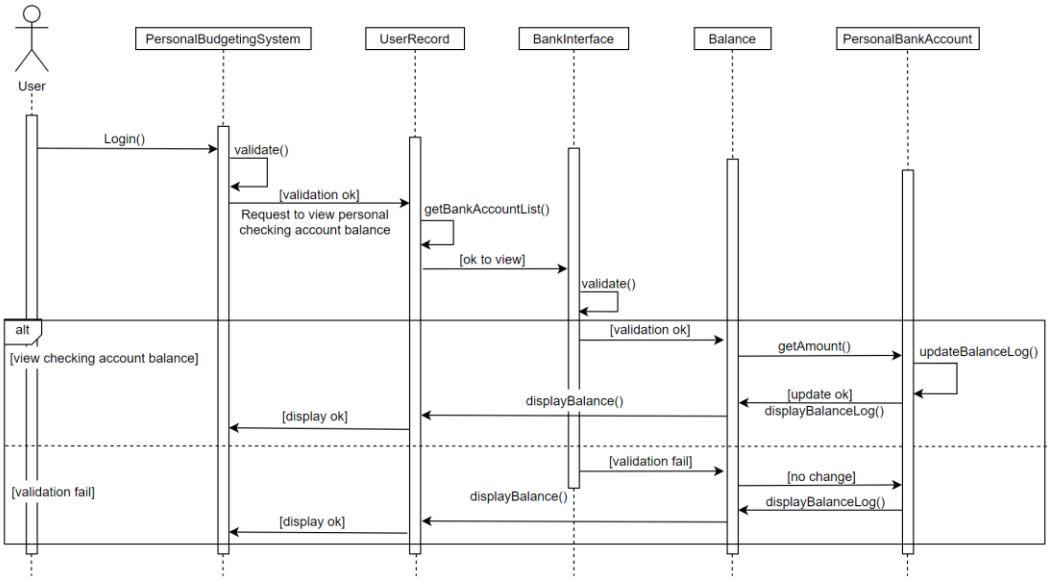
Input Financial Data (1.2)



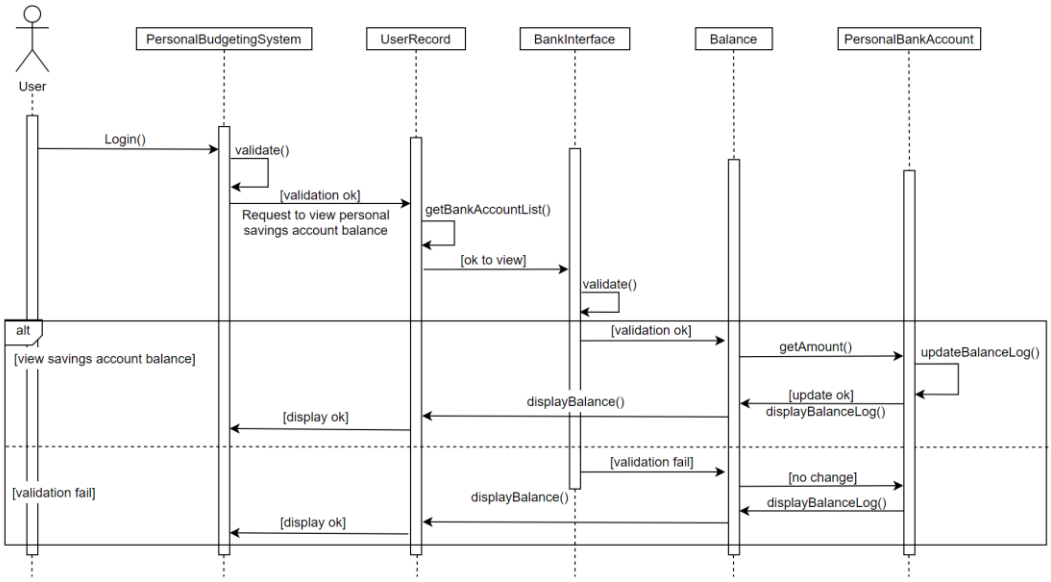
Input Transaction Data (1.2)



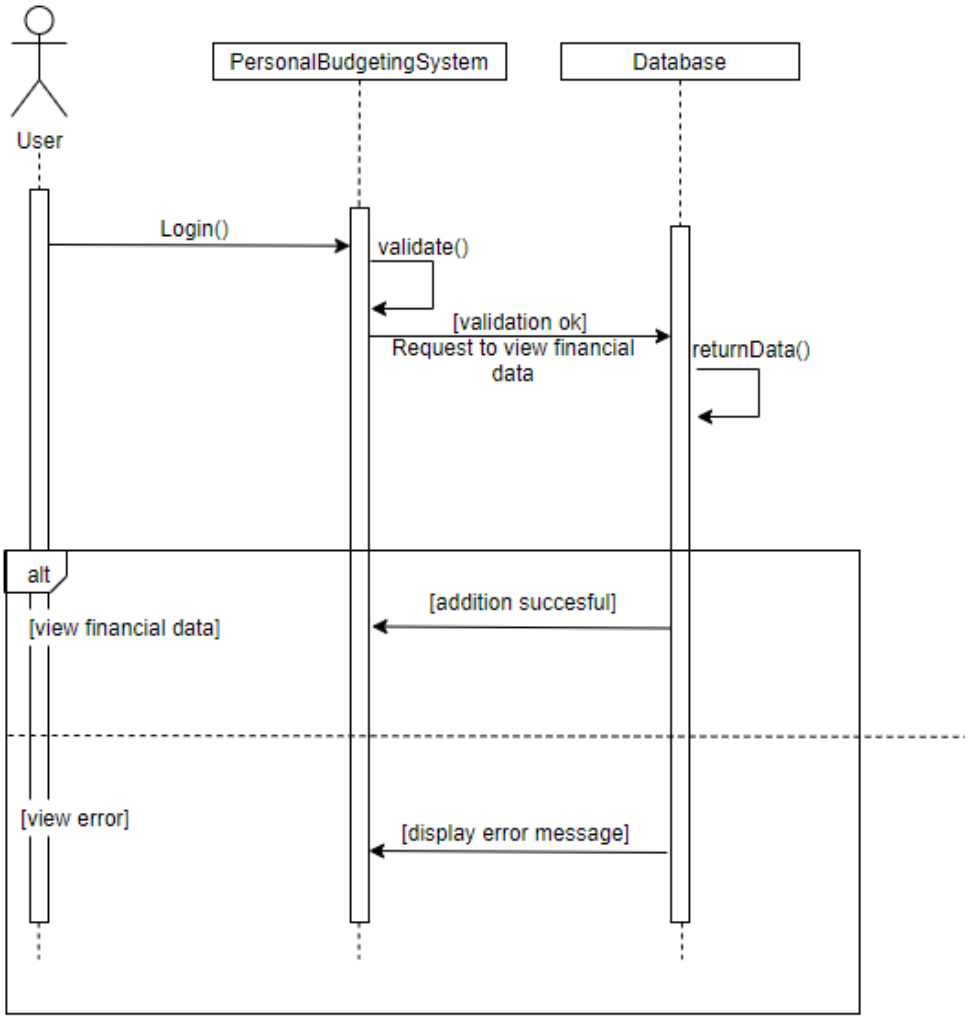
View Checking Account Balance (1.3)



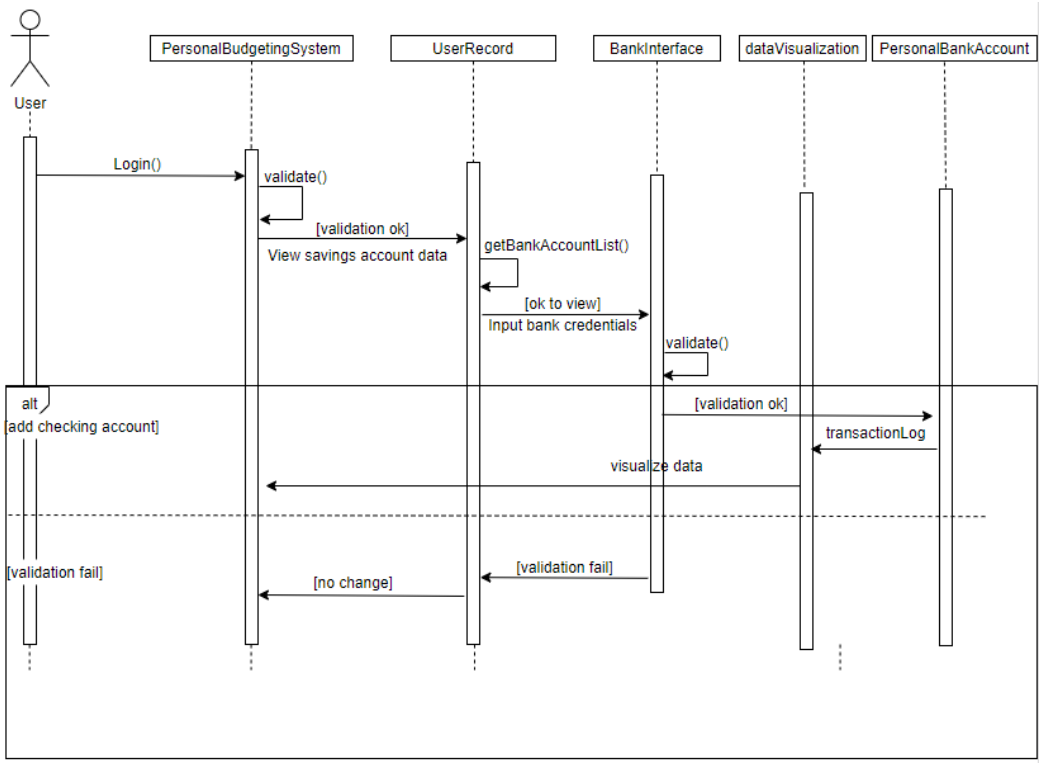
View Savings Account Balance (1.3)



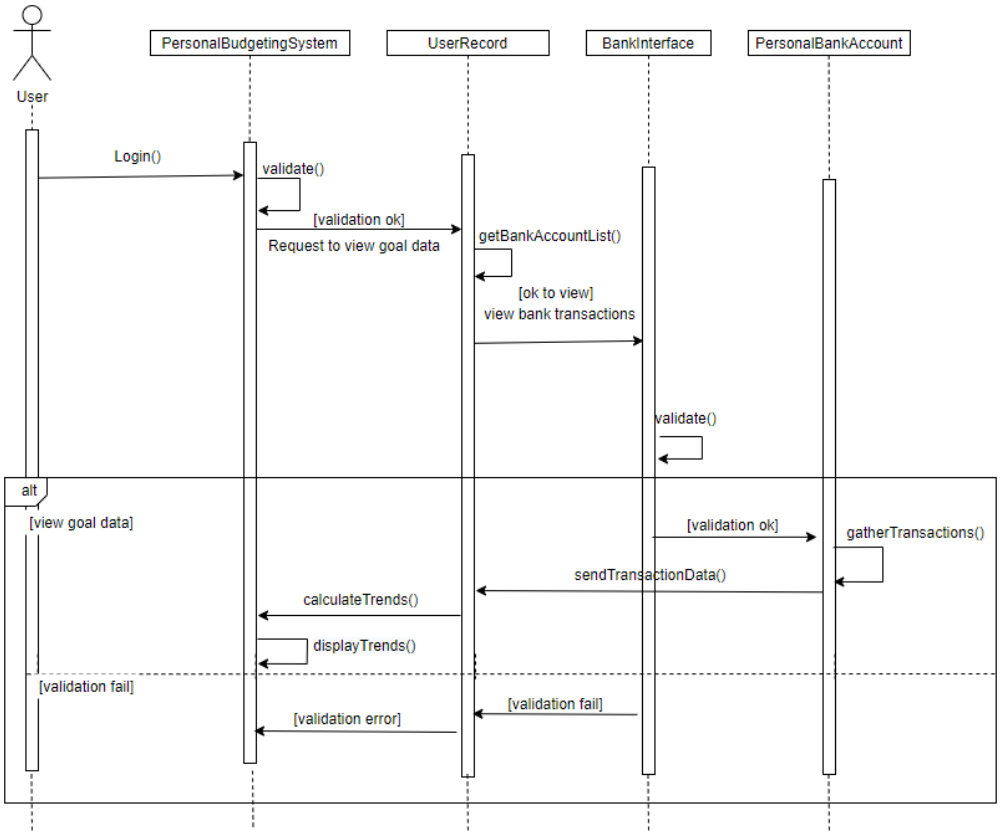
Financial Data (2.1)



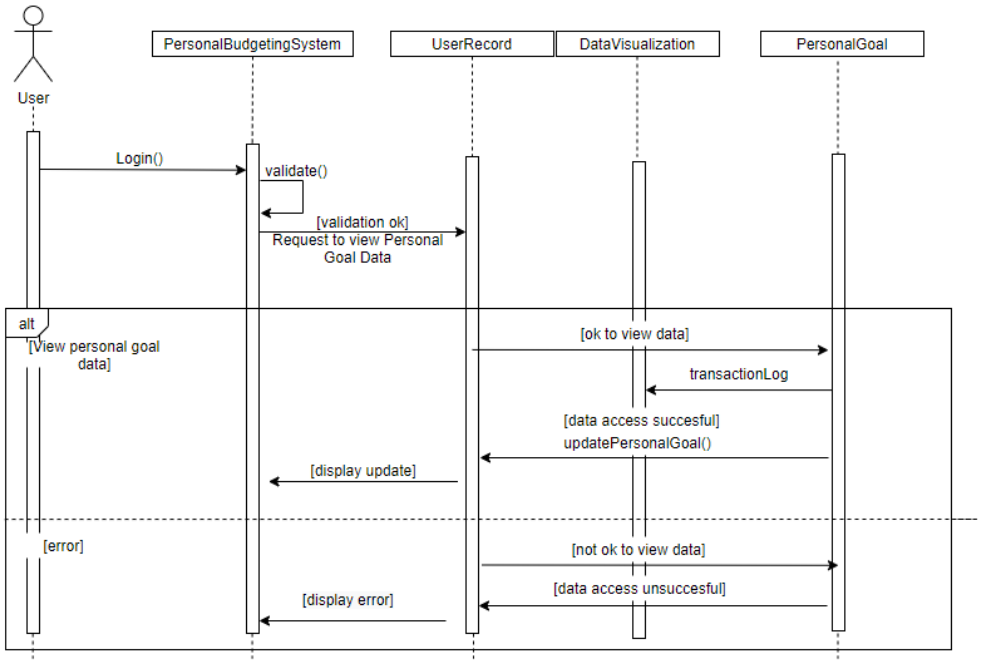
View Savings Account Data (2.1)



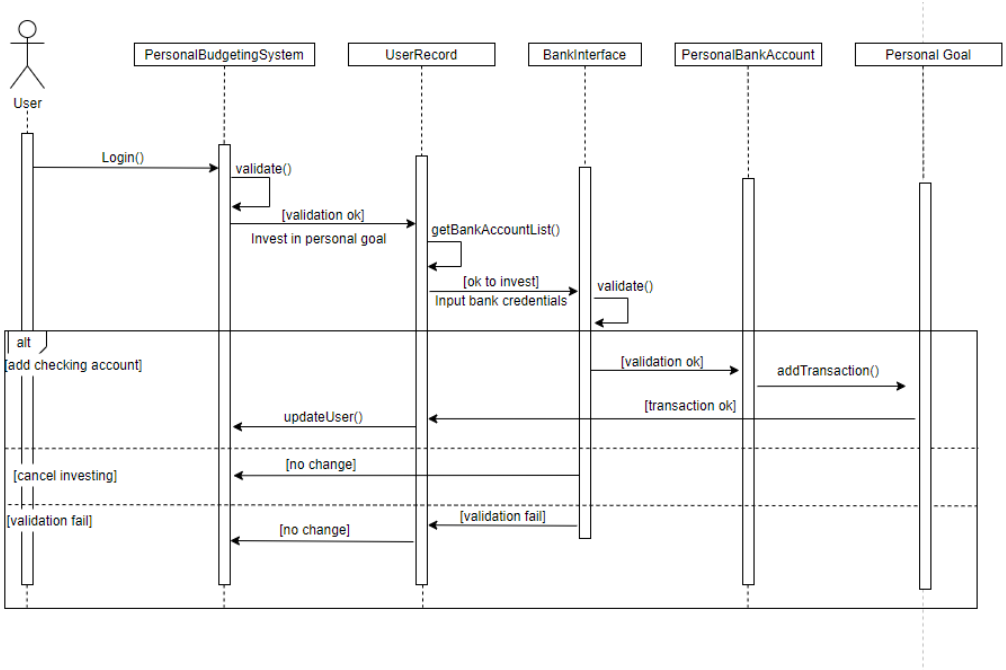
Add/Remove Personal Goals (3.1)



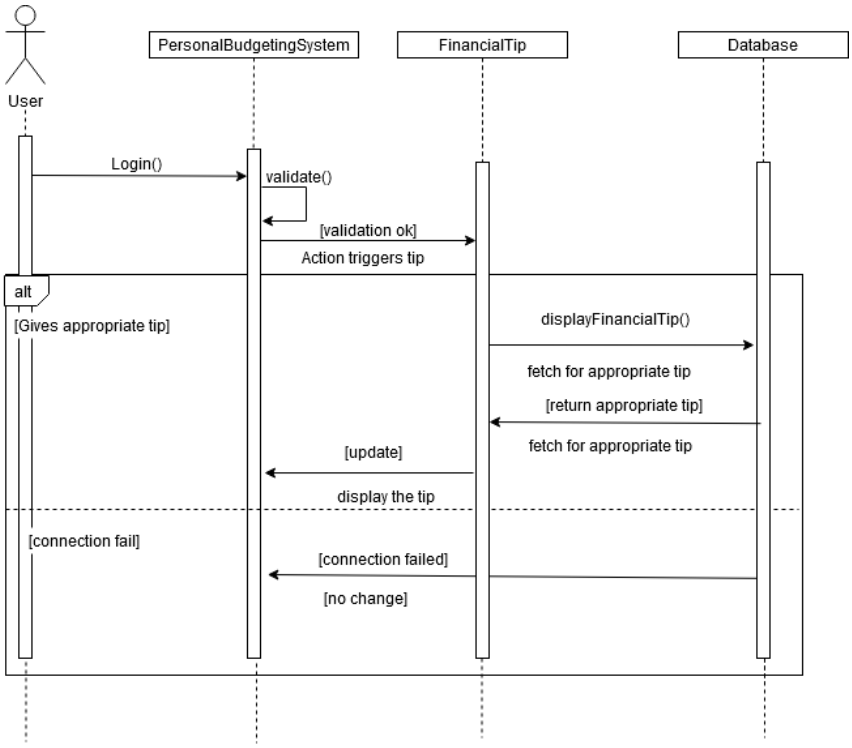
View Personal Goals Data (3.3)



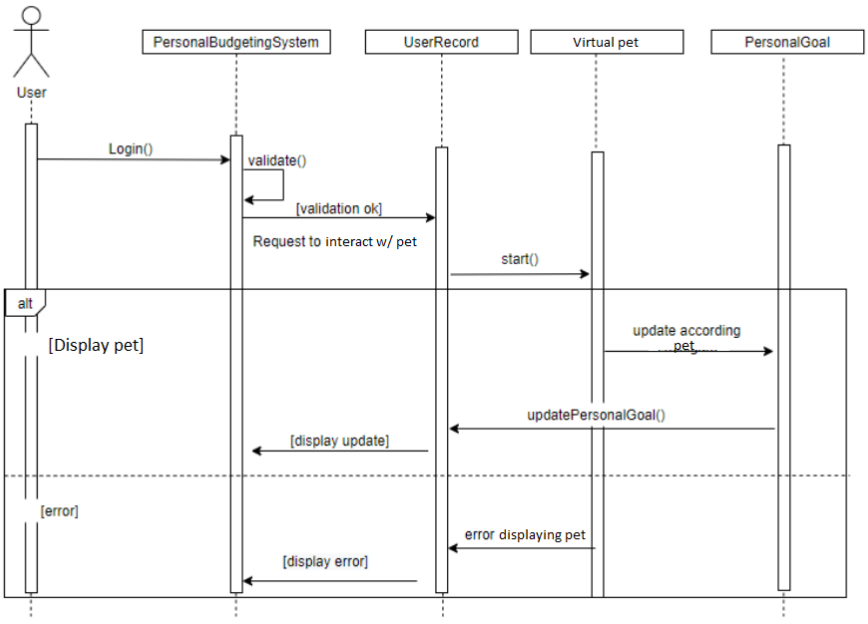
Invest in Personal Goals (3.2)



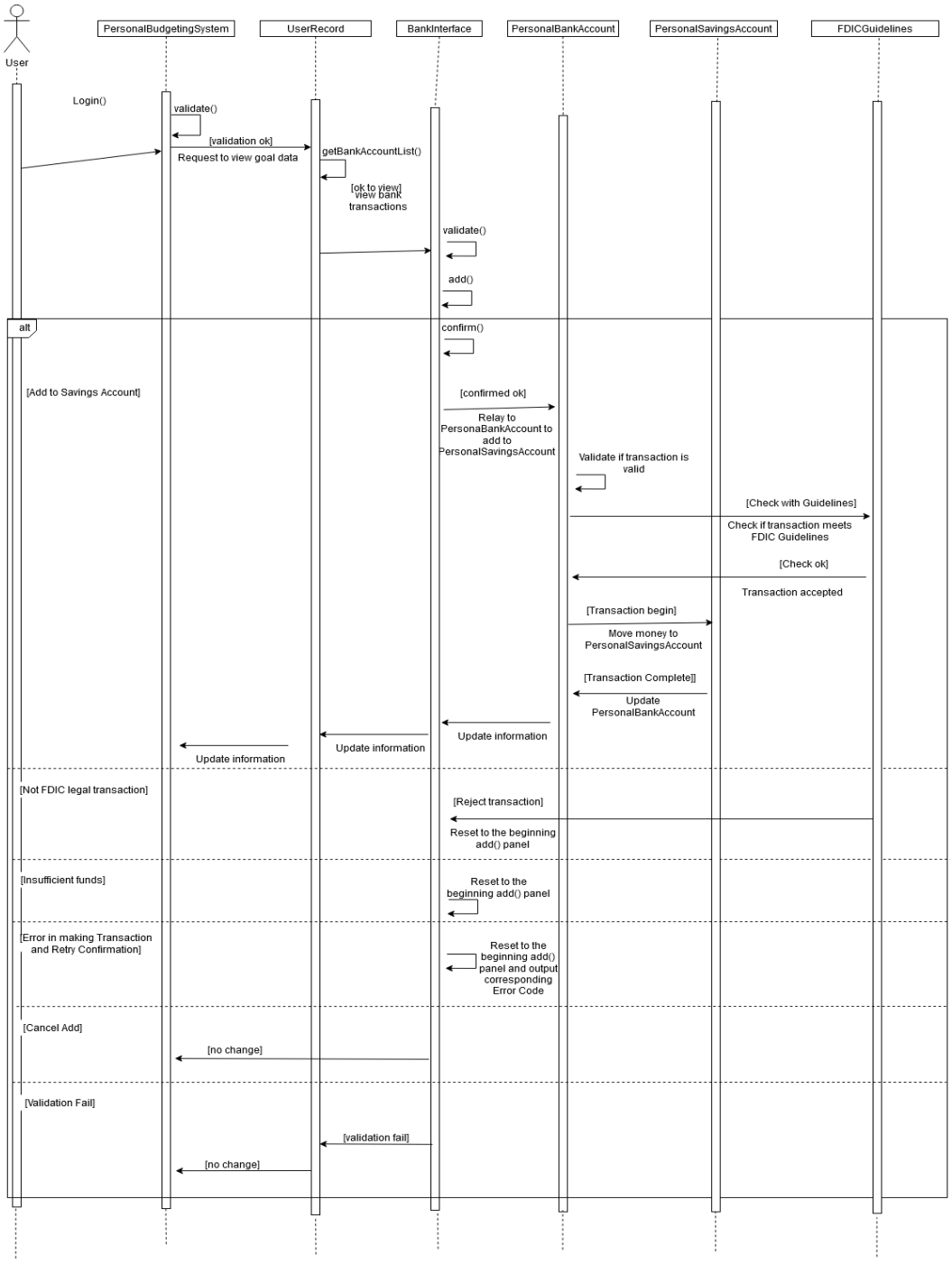
Read Finance Tips (4)



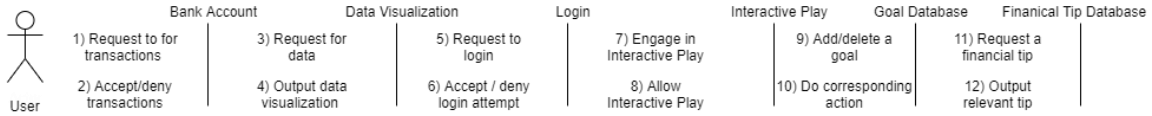
Interact with Interactive Pet (5.1)

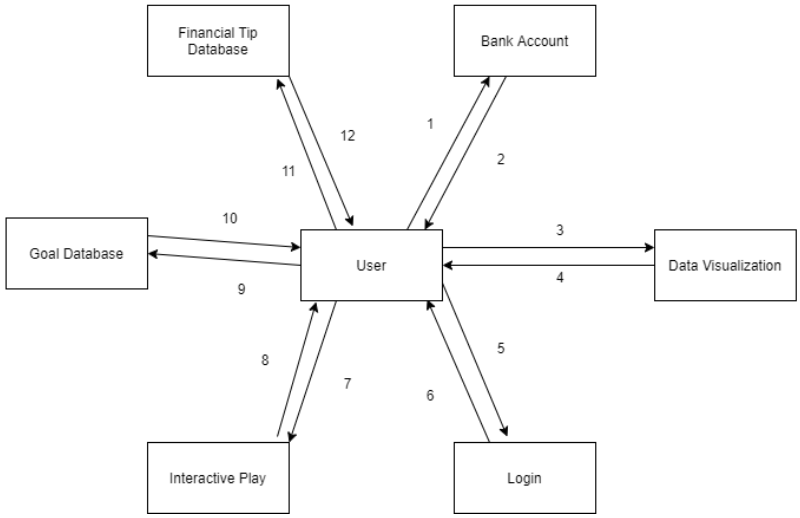


Add to Savings Account (1.4)



9.8.2 Collaboration Diagrams





10. SYSTEM TEST PLAN REQUIREMENTS

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.

11. QUALIFICATION PROVISIONS

The team will go through four general steps to assure the document’s quality. The process taken will be self-review, peer review, walkthrough, and inspection. Each step, the team will assure the document’s quality based on eight characteristics: correctness, unambiguousness, completeness, consistency, stability, verifiable, modifiable, and traceable.

After each addition to the document, the editor will conduct a self-check. First, the individual will make sure that their addition to the document itself satisfies the eight characteristics mentioned above. Then the editor will read over the document to make sure that their addition does not cause any inconsistencies in the whole document.

Each week the team will meet up to review the document. Each member will review works of other team members and catch any defects within the document that the original

author might have not caught. In addition to defect catching, each member should give suggestions to further improve the quality of the document.

If the opportunity arises, the team will also hold walkthroughs. The team will aim to educate and inform the audience about the system and the documentation, while finding any defects within it. After the walkthrough, the team will note any questions or remarks of the audience and utilize it to make the document clearer and more readable.

Upon any major milestone in the system development, the team will employ the help of subject matter experts who will perform an inspection to find and list defects the document has. After the defects are reviewed and corrected, the team will contact the subject matter experts with the revision of the document, to confirm the document's quality.

12. 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.

13. EVOLUTION OF THE SRS

Requirements of the system always change—whether after installation or during development, and through new hardware, legislations, regulations, budgetary constraints, compromises, etc—and as such, the SRS should evolve along with these changes. Requirements evolution pans out as follows:

1. An initial understanding of the problem is developed through a project proposal, where initial requirements are drafted.
2. Getting feedback from select users and clients will change the understanding of the problem which will result in a set of changed requirements.
3. These changed requirements will then be validated and redistributed.

A formal change process to identify control, track, and report projected changes is documented as follows:

1. The client is required to complete an engineering change notice on an uniquely-identified existing requirement whether the change is adaptive, corrective, or perfective.
2. Management process must be addressed as this notice then goes to the development team, where every change will need to be looked at in adequate detail, to factor in its effect on the cost, schedule, dependencies, relation to the other requirements, etc. These considerations will be sent back to the client, who will then approve or disapprove the change.
3. If the change is approved by the client, it will be sent to the change control board, where it will be determined whether the requirement suits the business needs. Negotiations will be done between the development team, the client, and the control board on aspects like the release date and which iteration should the change be implemented in.
4. If the change is approved, it will be sent to the requirement writers, who will update the SRS, validate it, and redistribute it. If it does not pass validation, the change will be sent back to be revised.

Every requirement can be traced using its unique number through analysis, design, implementation, and the code itself. A database tool will be used to track every requirement number and where it is implemented in all the artifacts going forward.

14. RATIONALE

None.

15. NOTES

None.

16. APPENDICES

Schedule Tracking

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

Artifact or Deliverable	Who (Individual or 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		

Defect Tracking

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

Artifact or Deliverable	Who (Individual or 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		

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(),	date, time, location, type, amount, receiver

		displayTransaction(), newTransaction()	
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	newFDICGuidelines(), updateFDICGuidelines(), deleteFDICGuidelines(), displayBank(), displayAccountTypes(), displayGuidelines()	bank, accountType, guidelines

	regarding bank accounts and transactions		

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	BankInterface 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 dates and times	Simple	List	64 bytes		R
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	Establish a connection to the Personal	Data Visualization	Personal Budgeting System	Mandatory	*..1

connects to the Personal BudgetingSystem class.	Budgeting System class.				
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 System connects to the BankInterface interface.	Establish a connection to the BankInterface interface.	PersonalBudgeting System	BankInterface	Optional	1..*
BankInterface interface connects to the Personal BudgetingS	Establish a connection to the Personal Budgeting System class.	BankInterface	Personal Budgeting System	Mandatory	*..1

ystem class.					
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 BankInterface	Establish a connection to Bank Interface	PersonalBudgetingSystem	BankInterface	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 checking account	To add a checking account	addBankAccount()	The User must have a checking account to add that has not yet been added.	If the validation was successful, the User's checking account is now added into the system. Otherwise, the system state is unchanged.	The new checking account 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 Interface.	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 User's financial data is added into the system Database. Otherwise, the system Database is unchanged.	The database will include 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	Database state will include the added

					transaction data is added into the system Database. Otherwise, the system Database state is unchanged	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 list of personal goals	To add a new personal goal	newPersonalGoal()	None	If the connection was successful, the user has successfully added a new goal to the list of goals in the PersonalGoal class.	The personal goal list will 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	The personal goal list will not include the removed goal or

					list of goals in the PersonalGoal class.	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 into their personal goals	A new transaction will be added to the transaction Log, keeping 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 indicating 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 dependin g on the amount selected.