**CEN 4021 Software Engineering 2 U01 1201**

# MIDAS

Team 1:

Manuel Toledo

Gabriel Alfonso

Edward Gil

Dr. Peter Clarke

Deliverable 2

4/21/2020

# Abstract

The following document will go over the project plan, monitoring, system design and Detailed Design of Team 1’s MIDAS project. The project plan section will include the effort estimation of the Project along with a well proposed Risk management plan. Monitoring will include all the processes we used to monitor the Project such as Schedule integrity, completeness of function, quality monitoring and budget monitoring. The system design will contain the diagrams for the system design of the project. The Detailed Design portion will contain most of the uml diagrams including the minimal class diagram, sequence diagrams and state machines.

# Table of Contents

# 1. Introduction

Briefly introduce the contents of the chapter.

## 1.1. Problem definition

There are many financial applications in the market today that are created to help users keep track of their financial information. With so many similar applications available, the user is tasked with deciding which to use. Additionally, these are usually niche apps, so users must utilize multiple applications to have greater coverage regarding tracking their finances. The problem then is clear, users do not have access to a single application that can be used to track multiple parts of the user's finances, which is what this project will focus on.

## 1.2. Purpose, scope and goals of project

### Purpose

With this project we set up to design a budgeting application that would be straightforward to use. We planned that such design would appeal to the casual mobile and web user to aid them manage their finances in an intuitive manner. Another goal was to allow the user to be highly aware of their balance from their different bank accounts and the balance of their credit cards, so we decided to display that information at the top of the homepage. And while our design was very tempting for beginners in financial budgeting, we included stocks to attract folks more experienced with finances and encourage beginners to start their investing journey.

The Scope of this Project will include the formalization of the design of up to 10 use cases of our system. It will not include the implementation of such use cases and it will include the function definition of the other use cases apart from the 10 chosen.

## 1.3. Assumptions and constraints

Assumptions for this Project are that the Personnel is knowledgeable in UML design and React Native. We assume that the development process is not covered in the scope of this project. Some constraints of the Project are the small size of the team (3). Also, that the goals are going to be hard to measure and track.

### 1.4. Definitions, Acronyms, and Abbreviations

| Term | Meaning |
|---|---|
| UML | Unified Modeling Language - general-purpose, developmental, modeling language that is intended to provide a standard way to visualize the design of a system |
| StarUML | An open-source modeling software used to create UML diagrams. |
| OS | Operating System |
| UC | Use Case |
| COCOMO | A model for estimating cost for Projects |

### 1.5. Overview of document

This document will cover the second half of the Software design Project called MIDAS. In the Project Plan chapter we will be talking about the general organization and plan for the project along with Risk Management and Effort Estimation. In the Monitoring chapter we will cover the process of Monitoring the project by analysing the schedule integrity, completeness of function, quality and budget. Then we will talk about the general design stage of the project, and more specifically System Design and Detailed Design of the Project.

# 2. Project Plan

## 2.1.Project Organization

### 2.1.1. Project Personnel Organization

| Personnel | Team Role | Technical Role |
|---|---|---|
| Gabriel Alfonso | Team Leader | Quality Analyst |
| Manuel Toledo | Minute Taker | UML Engineer |
| Edward Gil | Time Keeper | Business Analyst |

### 2.1.2. Hardware and Software Resources

| Hardware Resources | Specs |
|---|---|
| ThinkPad X280 (Laptop x3) | CPU: Intel i5-7300U 2.60 GHz, RAM: 8GB, SDD: 256GB |
| Web Server | CPU: Quad Core 2GHz+, RAM: 6GB, Storage: 1 TB+ |
| Iphone | Model 7S |
| Android OnePlus | Model 6T |

| Software Resources |
|---|
| StarUML v3.2.2 |
| MS Project 2016 Edition |
| MongoDB v4.2.6 |
| Visual Studio Code v1.44 |

| |
|---|
| React Native v0.62 |

## 2.2. Effort estimation summary (focus on duration and cost)

The effort required for this project was estimated using the COCOMO cost estimation model [2]. For the 15 implemented use cases that are listed in Appendix B, the total estimated effort for the project is 16.8 person-months, and the estimated cost is $84018. A detailed description of the effort estimation is shown in Appendix C.

## 2.3. Identification of Tasks, Milestones and Deliverables (work breakdown)

| No. | Name | Tasks | Milestones | Deliverable |
|---|---|---|---|---|
| 1 | Project Definition | ☑ | ☑ | Deliverable 1 |
| 2 | Project Planning | ☑ | | |
| 3 | Feasibility Study | ☑ | ☑ | |
| 4 | Requirements Elicitation | ☑ | | |
| 5 | Requirements Analysis | ☑ | | |
| 6 | Deliverable 1 Finish | | ☑ | |
| 7 | Effort Estimation | ☑ | | Deliverable 2 |
| 8 | Risk Management Plan | ☑ | | |
| 9 | Monitoring | ☑ | | |
| 10 | System Design | ☑ | ☑ | |
| 11 | Detailed Design | ☑ | | |

| | | | | |
|---|---|---|---|---|
| 12 | Deliverable 2 Finish | | ☑ | |
| 13 | Use cases 1-3 (Implementation) | ☑ | | Implementation |
| 14 | Use Cases 4-6 | ☑ | | |
| 15 | Half UC Finish | | ☑ | |
| 16 | Use Cases 7-10 | ☑ | | |
| 17 | Testing UC 1-3 | ☑ | | |
| 18 | Testing UC 4-6 | ☑ | | |
| 19 | Testing UC 7-10 | ☑ | | |

## 2.4.Risk management plan (should include a list of possible risks)

Table 2.4.a is a risk register that contains some risks identified that can occur throughout the life of the project. Each risk has an estimated impact and probability, along with a possible contingency plan as a reaction to the risk if it should occur.

| No. | Name | Summary | Category | Impact | Probability | Contingency |
|---|---|---|---|---|---|---|
| 1 | Requirements change | Requirements are changed: use cases are added or modified. | Business | 1 | 50% | Meet with developers and project managers to decide if requirements can be changed without significant impact to schedule and cost, or if the requirement change should be made in a later release. |

| | | | | | | |
|---|---|---|---|---|---|---|
| 2 | Software version updates | A major version update occurs to one of the used pieces of software: React, etc. | External | 2 | 40% | If no harmful changes to product occur (i.e. deprecated functions), then update to the new version. |
| 3 | Insufficient technical knowledge | Development team does not have enough knowledge to implement requirements. | Technical | 2 | 25% | Acquire training resources to train staff, e.g. workshops. |
| 4 | Staff turnover | A team member with a lot of expertise tied into the project is leaving the company. | Business | 3 | 40% | Offer a reasonable raise in the team member's salary as an incentive to keep them on board. |
| 5 | Other products released | A different company releases a similar product to ours. | Business | 4 | 20% | Add new use cases to the product that allows it to be distinguished from the competition. |

| | | | | | | |
|---|---|---|---|---|---|---|
| 6 | Neglecting design | Developers neglect UML design specifications. | Technical | 4 | 10% | Go through the documentation again, to make sure design specifications are being adhered to. |
| 7 | Cannot acquire brokers license | Failure to acquire a brokers license to be allowed to purchase/sell stocks on behalf of users. | Legal | 4 | 20% | Attempt to reapply for a license, taking into account the reasons for the previous application rejection. |

Table 2.4.a - Risk Register of Identified Project Risks

A risk matrix is presented in Figure 2.4.a, which is used to visualize the impact and probability of the risks identified. Table 2.4.b is used to convert impacts and probabilities between text and number values. The risk matrix is divided into colored zones. The green zone indicates risk that is considered acceptable. The yellow zone indicates risk that requires special treatment and monitoring. The red zone indicates risk that is not acceptable for the project.

| Probability | | | Impact | | |
|---|---|---|---|---|---|
| As Text | As a Real Number | As an Integer | As Text | As a Real Number | As an Integer |
| Very Low | 0.2 | 1 | Negligible | 0.2 | 1 |

| | | | | | |
|---|---|---|---|---|---|
| Low | 0.4 | 2 | Small | 0.4 | 2 |
| Normal | 0.6 | 3 | Significant | 0.6 | 3 |
| High | 0.8 | 4 | Severe | 0.8 | 4 |
| Very High | 1.0 | 5 | Catastrophic | 1.0 | 5 |

Table 2.4.b - conversions for the risk matrix

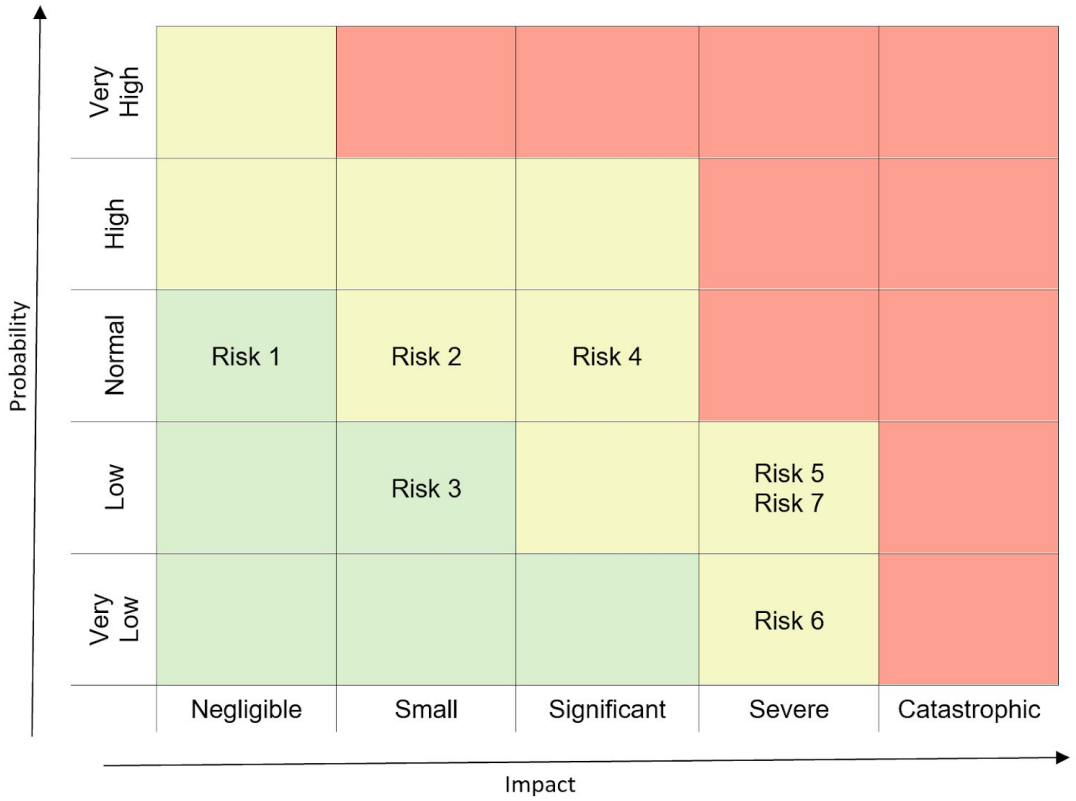



Figure 2.4.a - Risk Matrix

A risk management plan for the project is defined to identify the procedures to monitor and update risks and who is in charge of operations:

- Who is responsible for raising a warning?
    - The business analyst, quality analyst, and all of the developers.
- Who is to be warned?

    - The team leader.
- Who is responsible for approving activation of contingency plans?
    - The project manager.
- Formal steps:
    - The business analyst, quality analyst, and developers are responsible for monitoring for risks and raising a warning to the team leader if a risk may be occurring.
    - The team leader then gathers more information on the risk that may be playing out, including possible contingency plans to manage the risk.
    - The team leader hands over the information and contingency plans to the project manager, who will choose to approve one of the contingency plans.

# **3.** Monitoring

## 3.1.Schedule Integrity

| Milestone Activities | Expected Completion | Actual Completion | Delta |
|---|---|---|---|
| Project Definition | 1/16/20 | 1/16/20 | 0 |
| Feasibility Study | 2/4/20 | 2/5/20 | +1 |
| Deliverable 1 Finish | 2/28/20 | 2/28/20 | 0 |
| System Design | 4/6/20 | 4/7/20 | +1 |
| Deliverable 2 Finish | 4/16/20 | 4/21/20 | +5 |

## 3.2.Completeness of function

| Activities | UC-2 | UC-3 | UC-4 | UC-5 | UC-7 | UC-8 | UC-9 | UC-11 | UC-14 | UC-15 | Total |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Requirements Defined | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | 10 |
| Function Designed | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | 10 |
| Code Implemented | N | N | N | N | N | N | N | N | N | N | 0 |
| Function Tested | N | N | N | N | N | N | N | N | N | N | 0 |
| Function Integrated | N | N | N | N | N | N | N | N | N | N | 0 |

## 3.3.Quality

Quality monitoring was performed to detect any potential defects in the project's milestones and attempt to resolve them. The quality monitoring process was activity-based, so the milestone activities were each inspected to find defects that were occurring during their completion. While not all defects were resolved, a large portion of defects were brought to light and were able to be resolved.

| Milestone Activities | Defects Detected | Defects Resolved | Delta |
|---|---|---|---|
| Project Definition | 1 | 1 | 0 |

| | | | |
|---|---|---|---|
| Feasibility Study | 3 | 2 | -1 |
| Project Plan | 5 | 3 | -2 |
| Requirements Elicitation | 6 | 4 | -2 |
| Requirements Analysis | 7 | 5 | -2 |
| System Design | 5 | 5 | 0 |

Table 3.3.a - Defect Removal Activity-Based Quality Monitoring

Table 3.3.a lists the milestone activities along with the number of defects that were not resolved. During the feasibility study, one of the defects were not resolved, which was that the cost matrix did not account for hardware/software resources that were used in the management of the project itself, as opposed to the development of the product. The project planning, requirements elicitation, and requirements analysis phases each had to unresolved defects. The project plan was not specific about the hardware and software resources that would be used throughout the project, and the project personnel were only defined for a single phase, when it should have been defined for both phases of the project. The defects during the requirements elicitation phase included not quantifying non-functional requirements, and missing specifics about the input to some use cases. In the requirements analysis, some scenarios were not consistent with the use cases, and some object diagrams did not capture all the information in the scenarios.

Overall, 20 out of the 27 total defects discovered in the major milestones were resolved. That is a total of 74% resolved known defects.

### 3.4.Budget

| Milestone Activities | Expected | Actual | Delta |
|---|---|---|---|
| Project Definition | 0 | 0 | 0 |
| Feasibility Study | 1.5 | 1.5 | 0 |
| Project Plan | 0.5 | 0.5 | 0 |
| Requirements Elicitation | 1 | 1 | 0 |
| Requirements Analysis | 2 | 2 | 0 |
| System Design | 3.5 | 4.5 | 1 |

Table 3.4.a - Activity-Based Budget Monitoring

The budget of the project was monitored using an activity-based monitoring process, taking a look at the major milestones and comparing the expected budget with the activity's actual budget. Table 3.4.a shows the expected and actual budget for each milestone, with the budget expressed as a number of thousands of dollars, e.g. a value of 1.5 for expected budget means $1500.

For most of the project, the budget was as expected, except for the system design phase. This phase required more time and effort than was originally expected, so more resources in terms of time and people had to be assigned to work on the system design.

Overall, most of the budget was estimated correctly, with only the system design phase requiring an extra $1000 of resources that were not planned.

# **4.** System Design

This section will go into detail on MIDAS' system design. The system's architecture

patterns, subsystem decomposition, hardware and mapping, persistent data management and security management will be introduced and explained..

## 4.1.Overview

The architecture patterns chosen were 4-tier and microservices. The system was split into 4 major subsystems, Client which focuses on the pages provided to the user, Presentation Logic which determines whether to render a web page or a mobile page, System Logic which handles system operations using the microservices architecture pattern, and Data which handles the operations dealing with the data handling.

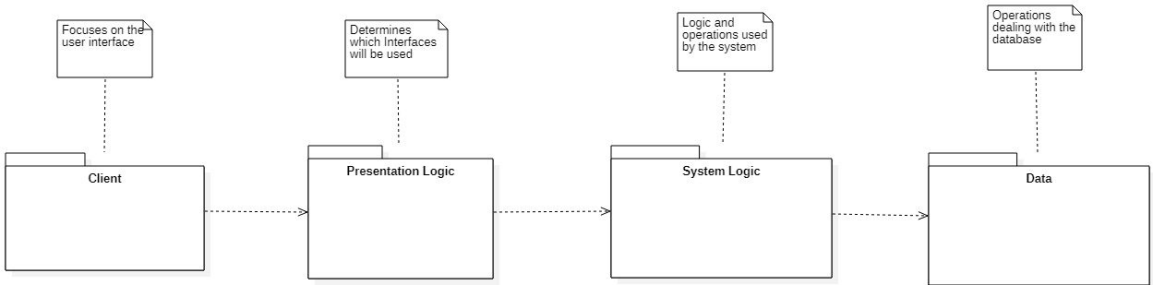



Figure 4.1.a *System Package Diagram*

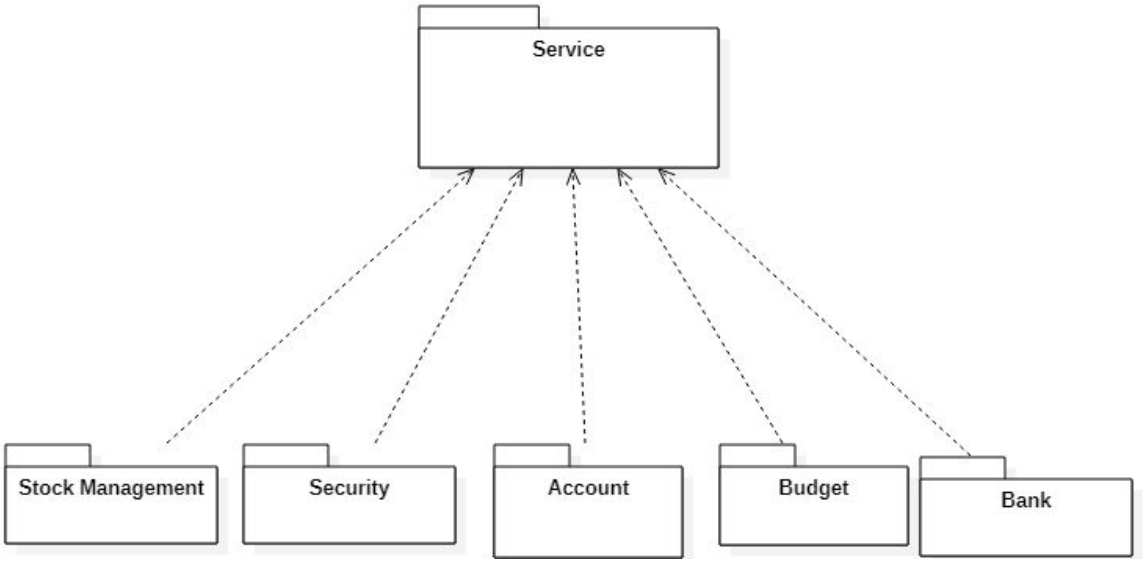



Figure 4.1.b *Microservices Package*

## 4.2.Subsystem Decomposition

Client which focuses on the pages provided to the user, Presentation Logic which determines whether to render a web page or a mobile page, System Logic

which handles system operations using the microservices architecture pattern, and Data which handles the operations dealing with the data handling.

*Client Layer*: Allows the system to communicate with the user by providing properly formatted data and information to the user while also being able to retrieve user input. Effectively allows us to generate GUI pages for the user to access and operate. Use cases: MIDAS-2-Obfuscation, MIDAS-3-View User Stocks, MIDAS-4-Add Stock, MIDAS-5-Create New Account, MIDAS-7-Timeout, MIDAS-8-Add Bank Account, MIDAS-9-View Balance, MIDAS-11-Create Budget, MIDAS-14-Login, MIDAS-15-Logout.

*Presentation Logic Layer*: Allows the system to determine which type of frontend to render for the user. Use cases: MIDAS-3-View User Stocks, MIDAS-4-Add Stock, MIDAS-5-Create New Account, MIDAS-8-Add Bank Account, MIDAS-9-View Balance, MIDAS-11-Create Budget, MIDAS-14-Login, MIDAS-15-Logout.

*System Logic Layer*: Allows the system to do its basic operations and logic involving handling user data, and accessing data. Use cases: MIDAS-2-Obfuscation, MIDAS-3-View User Stocks, MIDAS-4-Add Stock, MIDAS-5-Create New Account, MIDAS-7-Timeout, MIDAS-8-Add Bank Account, MIDAS-9-View Balance, MIDAS-11-Create Budget, MIDAS-14-Login, MIDAS-15-Logout.

Data Logic Layer: This layer allows the system to encapsulate data and provide ways to store data in persistent ways. Use cases: MIDAS-3-View User Stocks, MIDAS-4-Add Stock, MIDAS-5-Create New Account, MIDAS-7-Timeout, MIDAS-8-Add Bank Account, MIDAS-9-View Balance, MIDAS-11-Create Budget, MIDAS-14-Login, MIDAS-15-Logout.

## 4.3.Hardware and Software Mapping

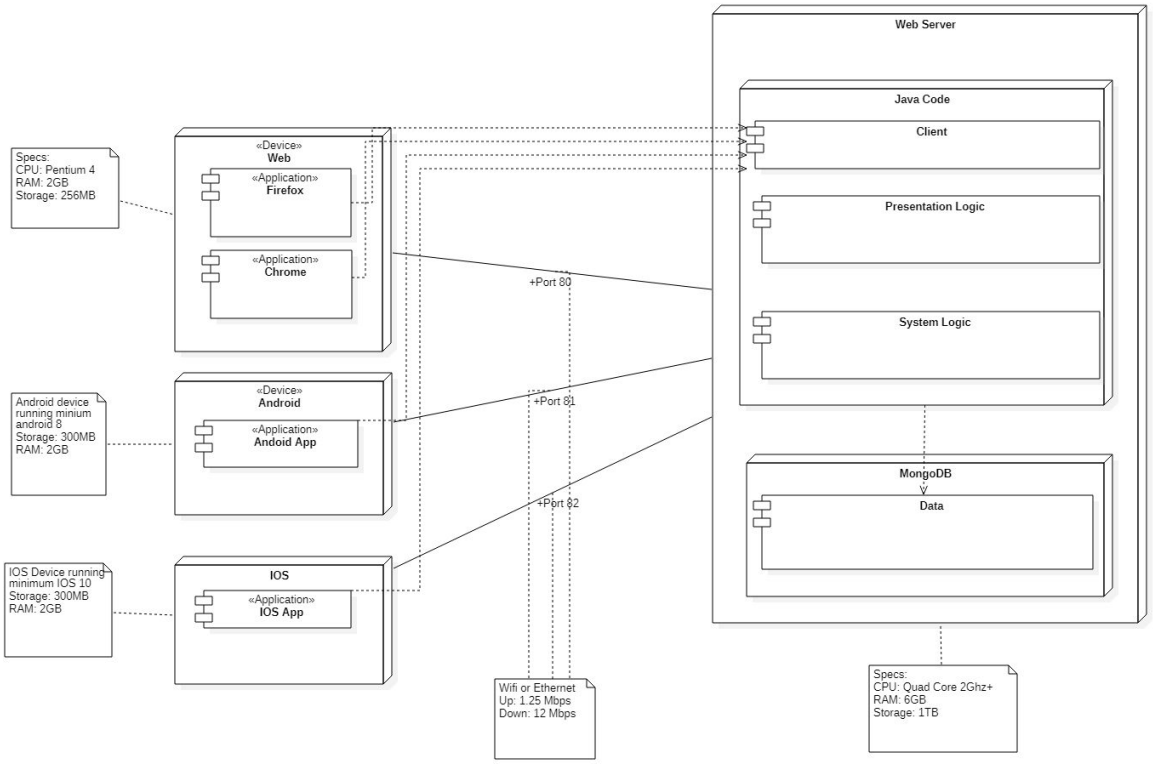



Figure 4.3.a *Deployment Diagram*

Hardware Specification for Server:

- CPU: Quad Core 2Ghz+
- RAM: 6GB
- Storage: 1TB

Hardware Specification for Web Client:

- CPU: Pentium 4
- RAM: 2GB
- Storage: 256MB

Hardware Specification for Android Client:

- Android device running minium android 8
- Storage: 300MB
- RAM: 2GB

Hardware Specification for IOS Client:

- IOS Device running minimum IOS 10

- Storage: 300MB
- RAM: 2GB

Connection Specification from Server to Client:

- Wifi or Ethernet
- Up: 1.25 Mbps
- Down: 12 Mbps

## 4.4. Persistent Data Management

For persistent data management, there are four objects that must be saved. The objects are User information, Budget information, Bank Account information, and Stock information.

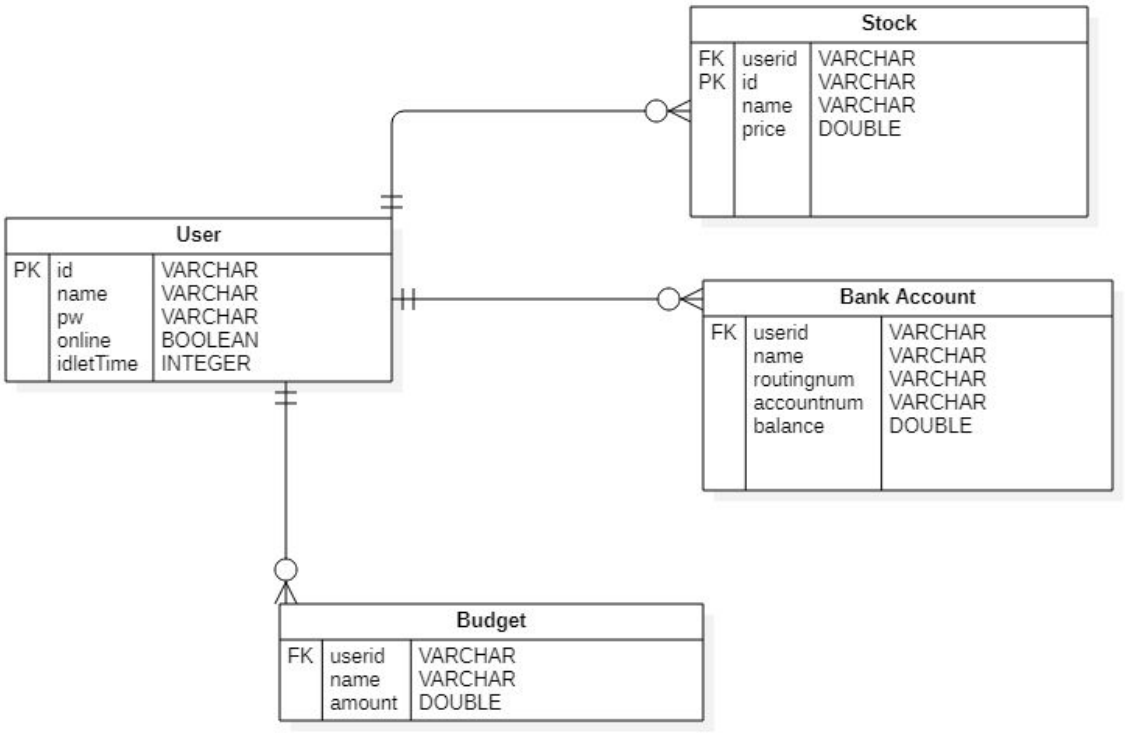



Figure 4.4.a *ER Diagram* in *Data* package

## 4.5. Security Management

For system security, a system timeout is being used to automatically log out an idle user who has been idle for at least 5 minutes. The user will be safely logged out without any problems. Password obfuscation will also be employed to mask the users password when they are in the process of inputting it in the login screen. For password encryption SHA hashing will be used to protect the user's

password from being visible to both an intruder and the system. SHA works by retrieving the inputted password string, hashing it then storing the hashed string into the database where it will later be compared to the user inputted and hashed password. Finally, for connection security HTTPS will be used.

# **5.** Detailed Design

This section will go into detail on MIDAS' system organization. The various subsystems, sequence diagrams, state machine models will be introduced and explained as well as the design patterns that are going to be used in the class designs.

## 5.1. Overview - minimal class diagram for the subsystem(s).

Client Subsystem: Having all the boundary page classes in this subsystem enables the system to be organized so no client side classes will be lost in the other system levels.

Presentation Subsystem: Having all the classes relating to page presentation here allows the system to separate control classes that manipulate the client pages.

System Logic Subsystem: Classes that deal with system logic and system operations.

Data Subsystem: Classes that deal with the operation of data and actual data objects.

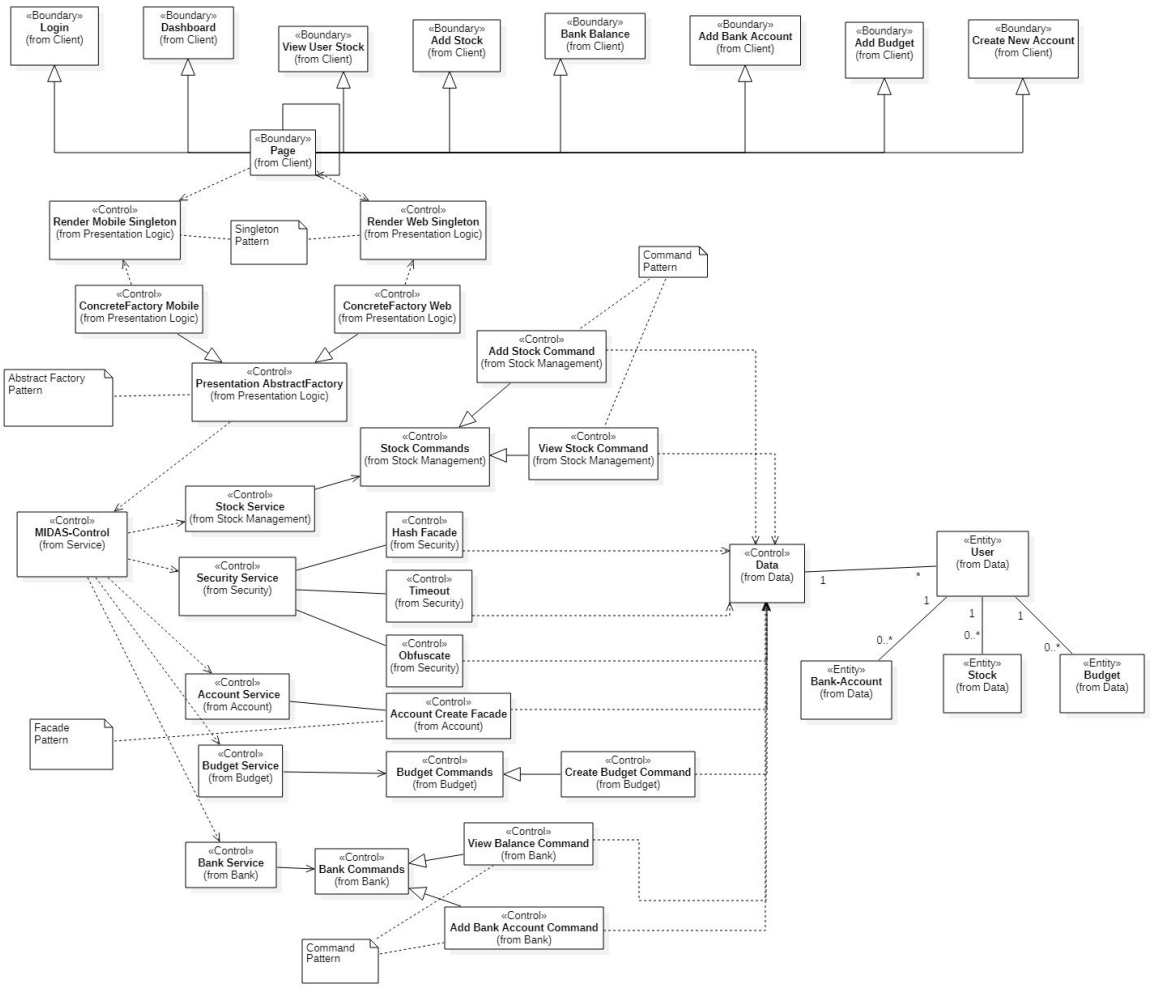



Figure 5.1.a *Minimal class Diagram*

## 5.2. Brief description of all sequence diagrams for the use cases. Refer to Appendix 6.4

*MIDAS-2-Obfuscation Sequence Diagram*: Obfuscates the user's password by changing the length and dotting the displayed password.

*MIDAS-3-View User Stocks Sequence Diagram*: Displays all of the user's owned stocks.

*MIDAS-4-Add Stock Sequence Diagram*: Displays the add stock page and allows the user to add a stock to his profile.

*MIDAS-5-Create New Account Sequence Diagram*: Displays the create new account page from the login page and allows the user to make a MIDAS account.

*MIDAS-7-Timeout Sequence Diagram*: Is a security measure that will logout the user if he is idle for 5 - 10 minutes.

*MIDAS-8-Add Bank Account Sequence Diagram*: Displays the add bank account page from the balance page and allows the user to add their bank account to their profile.

*MIDAS-9-View Balance Sequence Diagram*: Displays the balance page to the user so they can see their bank balance.

*MIDAS-11-Create Budget Sequence Diagram*: Displays the create budget page and allows the user to create a personal budget in their profile.

*MIDAS-14-Login Sequence Diagram*: Effectively checks the user's profile with the inputted password and logs the user into the dashboard page.

*MIDAS-15-Logout Sequence Diagram*: Effectively changes the user's online status in the server and redirects them to the login page.

## 5.3. State machine model

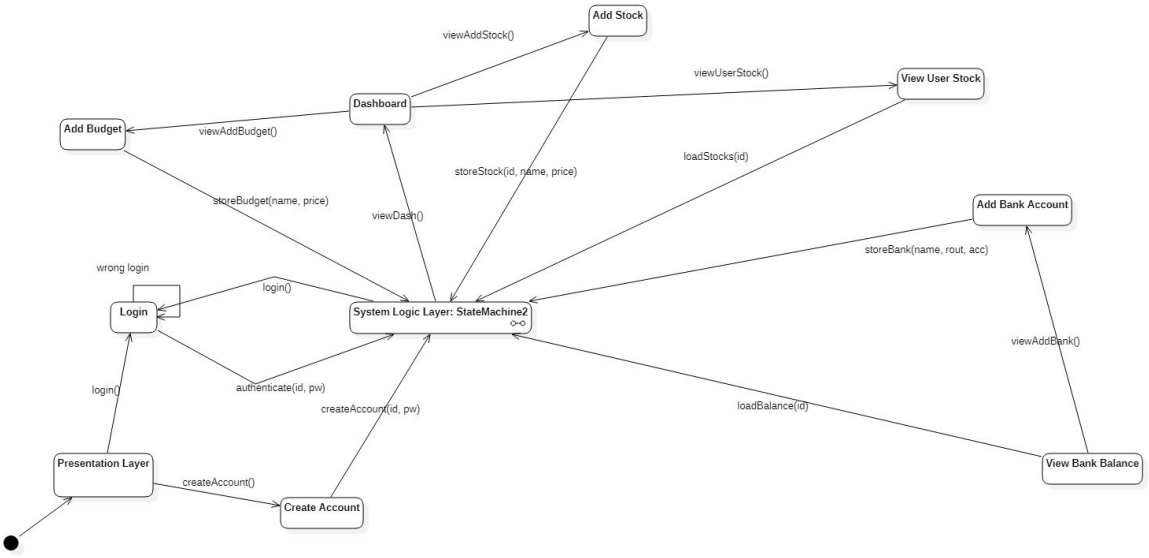



Figure 5.3.a ***Top Level State Machine*** : deals with the overview of the system and mostly the client layer.

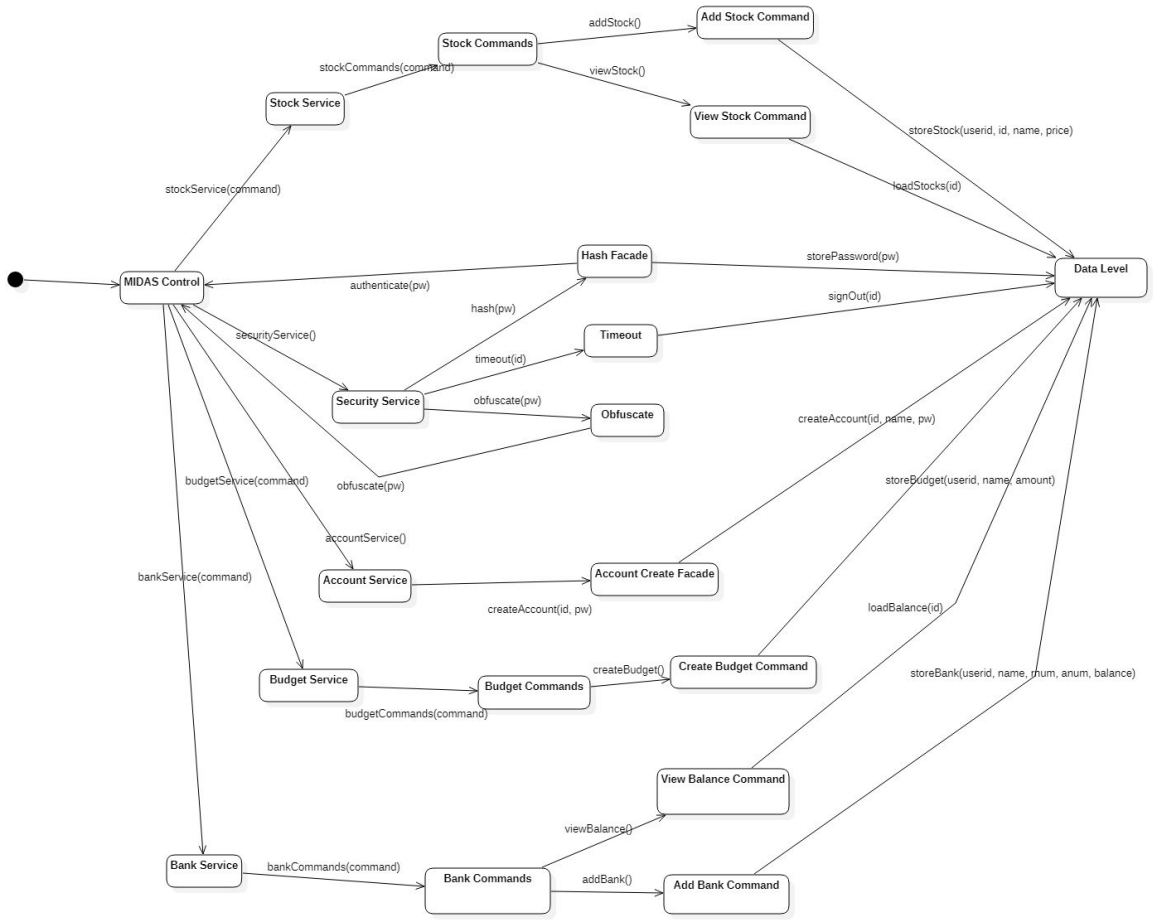



Figure 5.3.b ***System Logic Level State Machine:*** deals with the important system logic and operations in the system logic layer.

## 5.4. Description of detailed class design. Refer to Appendix 6.3.

Figure 6.4.a ***Client Class Diagram:*** is used to separate the boundary objects that the user will interact with.

Figure 6.4.b ***Presentation Logic Class Diagram:*** is used to render the specific type of pages (web or mobile) to the user and making sure that only one instance of them are rendered per user.

Figure 6.4.c ***Service Class Diagram:*** is used to connect the main system control class to the different microservices and the presentation layer.

Figure 6.4.d ***Stock Management Class Diagram:*** is in charge of dealing with the stock manipulation ie, add / view commands.

Figure 6.4.e ***Security Class Diagram:*** is in charge with dealing with system

security ie, timeout, hashing, and obfuscation.

Figure 6.4.f ***Account Class Diagram:*** is in charge of dealing with account creation manipulation.

Figure 6.4.g ***Budget Class Diagram:*** is in charge of dealing with budget manipulation commands such as add and view.

Figure 6.4.h **Bank Class Diagram:** is in charge of dealing with bank account manipulation and finding out the user's balance.

Figure 6.4.i ***Data Class Diagram:*** is in charge with data manipulation operations and entities that are used in the system such as User.

**Command**: commands were used in the *Stock Management Class Diagram, Budget Class Diagram,* and *Bank Class Diagram* to separate essential system operations into manageable commands..

**Singleton**: singletons were used in the *Presentation Logic Class Diagram* specifically to make sure there is only one active instance of a page per user in either the web or mobile application.

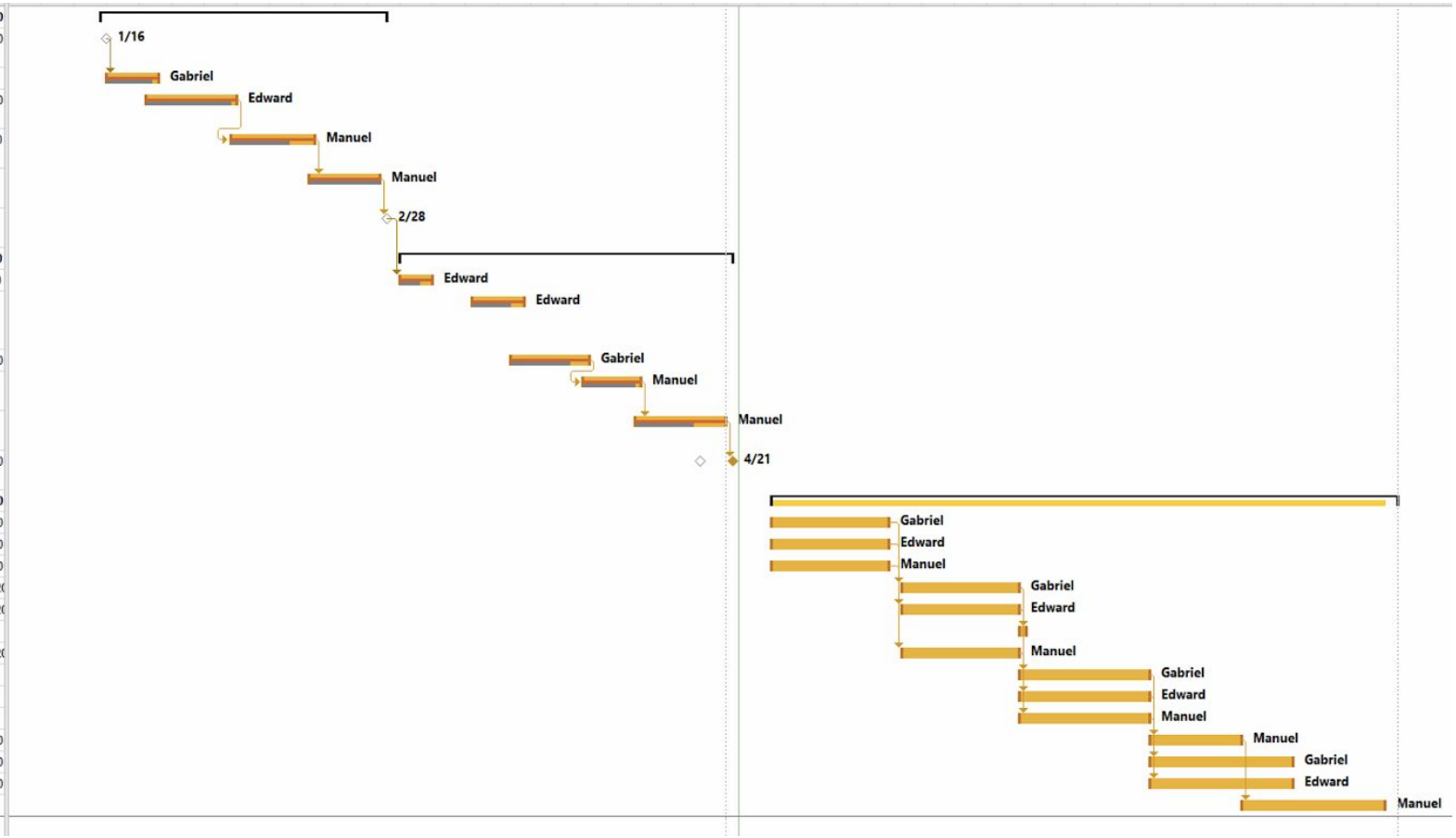
**Facade**: a facade was used to simplify the account creation in the *Account Class Diagram*.

**AbstractFactory**: an abstract factory was used in the *Presentation Logic Class Diagram* to differentiate which type of page (web or mobile) was being rendered to the user.

# 6. Appendix

## 6.1. Appendix A - Project schedule (Gantt chart or PERT Chart)

| | | |
|---|---|---|
| ◢ **Deliverable 1** | **32 days** | **Thu 1/16/20** |
| Project Definition (Kickoff Meeting) | 1 day | Thu 1/16/20 |
| Project Planning | 6 days | Fri 1/17/20 |
| Feasibility Study | 10 days | Thu 1/23/20 |
| Requirements Elicitation | 9 days | Wed 2/5/20 |
| Requirements Analysis | 9 days | Mon 2/17/20 |
| Deliverable 1 Finish | 1 day | Fri 2/28/20 |
| ◢ **Deliverable 2** | **37 days** | **Mon 3/2/20** |
| Effort Estimation | 5 days | Mon 3/2/20 |
| Risk Management Plan | 6 days | Fri 3/13/20 |
| Monitoring | 8 days | Thu 3/19/20 |
| System Design | 7 days | Mon 3/30/20 |
| Detailed Design | 10 days | Tue 4/7/20 |
| Deliverable 2 Finish | 1 day | Tue 4/21/20 |
| ◢ **Implementation** | **70 days** | **Tue 4/28/20** |
| Use case 1 | 14 days | Tue 4/28/20 |
| Use case 2 | 14 days | Tue 4/28/20 |
| Use case 3 | 14 days | Tue 4/28/20 |
| Use case 4 | 14 days | Mon 5/18/2( |
| Use case 5 | 14 days | Mon 5/18/2( |
| Half Use Cases | 1 day | Fri 6/5/20 |
| Use case 6 | 14 days | Mon 5/18/2( |
| Use case 7 | 14 days | Fri 6/5/20 |
| Use case 8 | 14 days | Fri 6/5/20 |
| Use case 9 | 14 days | Fri 6/5/20 |
| Use case 10 | 10 days | Thu 6/25/20 |
| Testing (1-3) | 16 days | Thu 6/25/20 |
| Testing (4-6) | 16 days | Thu 6/25/20 |
| Testing (7-10) | 16 days | Thu 7/9/20 |



## 6.2.Appendix B – All use cases

**Use Case ID:** MIDAS-1-Captcha

**Use Case Level:** System-level end-to-end

**Details:**

**Actors:** User

**Pre-conditions:**

1. User Must have an account
2. User is on the *login page*
3. The account login request is sent into the system.
4. The system determines the email and password combination is incorrect.
5. The system responds by outputting an Email/Password incorrect screen.

**Description:** System will render a captcha system in the login screen for the user to complete

**Trigger:** Captcha is shown.

The system responds by... .

1. User inputs captcha text.
2. Input is sent to the system for verification.
3. System authenticates User's password and username.
4. System.signs User in

CHA

**Post-conditions:**

1. System displays the dashboard.

**Relevant Requirements:** Google's reCAPTCHA

**Alternative courses of action:** None

**Extensions:** None

**Exceptions:**

- The user attempts to login without the captcha being completed. The system will tell the user to complete the captcha to be able to login.
- System failed to authenticate User information.

**Concurrent Uses:** None

**Related use cases:** T01-21-Web-Login

---

**Decision Support**

**Frequency:** 50 requests a month.

**Criticality:** High, ensures the account is safe from login brute forcing.

**Risk:** Medium, requires learning how to implement a captcha system into a login page.

---

**Constraints**

**Usability:**

- The average user should be able to complete a captcha in under 1 minute..

**Reliability:**

- Mean Time to Failure - 5 failures every 48 working hours is acceptable.

**Performance:**

- The captcha image should take no longer than 2 seconds to appear.

**Supportability:**

- The captcha system will be displayed correctly in Mozilla Firefox and Google Chrome.

**Implementation:**

- The use case will be implemented using React and will be available on browsers and in the mobile application.

---

**Modification History**

**Owner:** Manuel Toledo

**Initiation Day:** 1/15/2020

**Date Last Modified:** 1/22/2020

**Use Case ID:** MIDAS-2-Obfuscation

**Use Case Level:** System-level end-to-end

**Details:**

**Actors:** User

**Pre-conditions:**

1. User is in the Login Page

**Description:** System will obfuscate user inputted password into dots and shorten the password length.

**Trigger:** User inputs password into password text field in login page.

The system responds by...

1. System will visually change the inputted password into dots.
2. System will visually change the length of the password.

**Relevant requirements:** None

**Post-conditions:**

1. The text in the password field is shortened and shown as several black dots.

**Alternative courses of action:** None

**Extensions:** None

**Exceptions:**

- System does not shorten the password

**Concurrent Uses:** None

**Related use cases:** None

---

**Decision Support:**

**Frequency:** High - performed every time a user logs into the system.

**Criticality:** High - removes the possibility of the users password being stolen by someone looking over their shoulder.

**Risk:** Low

---

**Constraints:**

**Usability:**

- No previous training is needed as it is straightforward.

**Reliability:**

- Mean Time to Failure - 1 failure for every 48 hours of operation is acceptable.

**Performance:**

- Inputted password should be obfuscated within 1 second.

**Supportability:**

- Obfuscation should display properly in Mozilla Firefox and Google Chrome.

**Implementation:**

- The use case will be implemented using React and will be available on browsers and in the mobile application.

---

**Modification History**

**Owner:** Manuel Toledo

**Initiation Day:** 2/28/2020

**Date Last Modified:** 4/23/2020

**Use Case ID:** MIDAS-3-View User Stocks

**Use Case Level:** system-level end-to-end

**Details:**

**Actors:** User

**Pre-conditions:**

1. User is logged into system.
2. User is in the dashboard.

**Description:** System will show all the user's stocks to the user.

**Trigger:** User clicks on View Stocks button on dashboard

The system responds by...

1. System will change the displayed page to the View User Stock page.
2. System will load user stock data from the database.

**Relevant requirements:** None

**Post-conditions:**

1. The user will be in the View User Stock page.
2. The user will be able to view his/her stocks.

**Alternative courses of action:**

1. At steps 1 User can click the back button to return to the dashboard..

**Extensions:** None

**Exceptions:**

- System does not display the View User Stock page.
- System does not load user stock data.

**Concurrent Uses:** None

**Related use cases:**

1. MIDAS-4-Web-Add Stock

---

**Decision Support:**

**Frequency:** Medium - performed every time a user wishes to view his/her stocks.

**Criticality:** High - is a fundamental part of MIDAS

**Risk:** Low - needs little investment.

---

**Constraints:**

**Usability:**

- No previous training is needed as it is straightforward.

**Reliability:**

- Mean Time to Failure - 10 failure for every 48 hours of operation is acceptable.

**Performance:**

- View User Page should be shown and functional within a second of being called.

**Supportability:**

- Should be displayed properly in Mozilla Firefox and Google Chrome.

**Implementation:**

- The use case will be implemented using React and will be available on browsers.

---

**Modification History:**

**Owner:** Manuel Toledo

**Initiation Day:** 2/28/2020

**Date Last Modified:** 4/23/2020

**Use Case ID:** MIDAS-4-Add Stock

**Use Case Level:** system-level end-to-end

**Details:**

**Actors:** User

**Pre-conditions:**

1. User is in the dashboard.

**Description:** The use can add a stock to his/her account.

**Trigger:** The user clicks on the add stock button in the dashboard.

The system responds by...

1. System displays the add stock page to the user.
2. The User inputs the name of the stock.
3. The user inputs the amount owned of the stock.
4. The user clicks the submit button.
5. System stores the stock data into the database.

**Relevant requirements:** None

**Post-conditions:**

1. System displays the add stock page..

**Alternative courses of action:**

1. At step 2 to 3, User can click the back button.

**Extensions:** None

**Exceptions:**

- System does not display the add stock page.
- System does not store the stock information.

**Concurrent Uses:** None

**Related use cases:**

1. MIDAS-3-Web-View User Stocks

---

**Decision Support:**

**Frequency:** High - performed every time a user wishes to add a stock to their account.

**Criticality:** High - is a fundamental part of MIDAS.

**Risk:** medium - requires writing data to database.

---

**Constraints:**

**Usability:**

- Straight forward, should take no more than 20 seconds for a user to complete it.

**Reliability:**

- Mean Time to Failure - 3 failures for every 48 hours of operation is acceptable.

**Performance:**

- Stock should be added to the database within a second.
- Add Stock page should be displayed within a second.

**Supportability:**

- Should display properly in Mozilla Firefox and Google Chrome.

**Implementation:**

- The use case will be implemented using React

---

**Modification History:**

**Owner:** Manuel Toledo

**Initiation Day:** 2/28/2020

**Date Last Modified:** 4/23/2020

**Use Case ID:** MIDAS-5-Create New Account

**Use Case Level:** system-level end-to-end

**Details:**

**Actors:** User

**Pre-conditions:**

1. User is in the Login page

**Description:** User creates a MIDAS account.

**Trigger:** User clicks the create account button in the login screen.

The system responds by...

1. System displays the Create New Account page.
2. User inputs name.
3. User inputs user id.
4. User inputs password.
5. User clicks the submit button.
6. System stores inputted values into the database.
7. System displays the Login page.

**Relevant requirements:**

**Post-conditions:**

1. System displays the Login page.

**Alternative courses of action:**

1. In steps 2 through 4, User can click the back button.

**Extensions:** None

**Exceptions:**

- System fails to display the Create New Account page.
- System fails to store User inputted values.
- System fails to display the Login page.

**Concurrent Uses:** None

**Related use cases:** None

---

**Decision Support:**

**Frequency:** Medium - Use case only happens at most once per user visit.

**Criticality:** High - Use case is needed for users to make a MIDAS account.

**Risk:** Medium - requires database management.

---

**Constraints:**

**Usability:**

- User usability should be straightforward and easy.

**Reliability:**

- Mean Time to Failure - 1 failure for every 30 account creation is acceptable.

**Performance:**

- New user profile should be added to the database within a second.
- Create a New Account page should be displayed within a second of being called.

**Supportability:**

- Obfuscation should display properly in Mozilla Firefox and Google Chrome.

**Implementation:**

- The use case will be implemented using React.

---

**Modification History**

**Owner:** Manuel Toledo

**Initiation Day:** 2/28/2020

**Date Last Modified:** 4/23/2020

**Use Case ID:** MIDAS-6-Encryption

**Use Case Level:** High-Level

**Details:**

**Actors:** Admin

**Pre-conditions:**

1. The admin is logged into the system

**Description:** End-to-end encryption to protect against attacks and data leaks.

**Trigger:** The Admin clicks on encryption keys

The system responds by...

1. The system presents the Admin with the keys used to decrypt data sent from a user and the keys used to encrypt data being sent to a user.

**Relevant requirements:** None.

**Post-conditions:**

1. The admin can check all the encryption keys as proof that encryption is in place.

**Alternative courses of action:** None.

**Extensions:** None.

**Exceptions:**

- The Admin's credentials are invalid to see the encryption keys.

**Concurrent Uses:** None.

**Related use cases:** None.

---

**Decision Support:**

**Frequency:** Encryption occurs continuously.

**Criticality:** Medium. Although encryption offers a great deal of protection, it is not vital for the functionality of the app.

**Risk:** Medium risk. Implementing an end-to-end encryption system can be tricky especially for beginner software developers.

---

**Constraints:**

**Usability:**

- Encryption should be seamless so as to protect the user without them knowing it is there.

**Reliability:**

- Occasional encryption and decryption errors are acceptable. About 1 error per 10,000 encryption is accepted.

**Performance:**

- The encryption should strongly protect the user data by making it near impossible for a middle-man to crack it.

**Supportability:**

- Encryption will be supported in Mozilla Firefox and Chrome. The mobile application will be supported in Android and IOS.

**Implementation:**

- This use case will be implemented using React and will be available on browsers and in the mobile application.

---

**Modification History:**

**Owner: Gabriel Alfonso**

**Initiation Day: 02/29/2020**

**Date Last Modified: 03/03/2020**

**Use Case ID:** MIDAS-7-Timeout

**Use Case Level:** System-level end-to-end

**Details:**

**Actors:** User

**Pre-conditions:**

1. The user must have an account.
2. The user must be logged in.

**Description:** After a certain amount of time of inactivity, the system logs out the user for security.

**Trigger:**

The user has been inactive for 10 minutes.

The system responds by...

1. The system displays a countdown saying the user will be logged out if the user doesn't confirm that they are still there.
2. Once the countdown reaches zero the system logs the user out.
3. The system displays a message saying the user was logged out for inactivity.

**Relevant requirements:** None.

**Post-conditions:**

1. The user is no longer logged in, and would have to login again to use the application.

**Alternative courses of action:** If after the countdown starts and before it reaches zero the user confirms they are still there, then the countdown is canceled and the user remains logged in.

**Extensions:** None.

**Exceptions:**

- The countdown dialog box does not allow the user to cancel the countdown, so the user is logged out even though the user is still there.

**Concurrent Uses:** None

**Related use cases:** MIDAS-14-Web-Login, MIDAS-15-Web-Logout.

---

**Decision Support:**

**Frequency:** For 10,000 users, this use is expected to occur 1500 times per hour.

**Criticality:** Medium Criticality. Even though this use case does provide some security for the user, it is not utterly necessary for the functionality of the app.

**Risk:** Low Risk. This use case is simple to implement, just forcefully log out the user after some time of inactivity.

---

**Constraints:**

**Usability:**

- The countdown dialog box should be simple and straightforward, so the user knows what to do. The user should understand what happened after they got logged out through a simple logout message.

**Reliability:**

- This use case is allowed to fail 1 in 7500 times.

**Performance:**

- The user should be logged out within 2 seconds of the countdown running off.

**Supportability:**

- This use case will be supported in Mozilla Firefox and Chrome. The mobile application will be supported in Android and IOS.

**Implementation:**

- The use case will be implemented using React and will be available on browsers and in the mobile application.

---

**Modification History:**

**Owner:** Gabriel Alfonso

**Initiation Day:** 02/26/2020

**Date Last Modified:** 04/23/2020

**Use Case ID:** MIDAS-8-Add_Bank

**Use Case Level:** System-level end-to-end

**Details:**

**Actors:** User

**Pre-conditions:**

1. User must have an account.
2. User must be logged in.
3. The user is on the homepage.

**Description:** The user adds a Bank Account to their personal account to manage it.

**Trigger:**

The user clicks on the Add Account button.

The system responds by…

1. Providing most common banks, billers, and investment accounts, along with a search bar to look up any account by name
2. After the user chooses a bank account to add, the system outputs a 'connect your account' page, whereby after accepting, the user is taken to the native website of said bank.
3. Once the bank authentication is complete, the system uploads the information to its database and displays the proper information from the bank account.

Providing most commonly added banks, billers, and investments accounts, and providing a search bar to search for any other account by name.

{The user chooses their bank account they want to add}

**Relevant requirements:** None.

**Post-conditions:**

1. A bank account has now been added into the user' account, from which you can see recent transactions and total balance.

**Alternative courses of action:** The user chooses the wrong account to add and proceeds to click on *back* to return, then finally chooses and adds the correct account.

**Extensions:** Bank Authentication

**Exceptions:**

- The user fails to log in to their third-party bank website and is therefore locked from their bank account, not allowing the bank to be added.

**Concurrent Uses:** None

**Related use cases:** T01-23-Bank Authentication, T01-24-Encryption

---

**Decision Support:**

**Frequency:** For 10,000 users, this use is expected to occur 80 times per hour.

**Criticality:** High Criticality. This use case unlocks critical features of the application, such as budget management and managing your total balance and recent transactions.

**Risk:** Medium to High risk. Even though this use case does not require extensive technical expertise in order to implement. It requires implementing a search bar, prompting authorization from third-party banks, and storing and displaying the proper information from the bank account. Which, if not properly implemented, can be exploited by third-parties to steal critical information of the user.

---

**Constraints:**

**Usability:**

- The average user should be able to add a bank account in under 2 minutes given that they have the correct credentials.

**Reliability:**

- A 10% failure per week is to be expected given that the system has to interact with third-party websites, which require additional credentials.

**Performance:**

- The user should be connected to the respective third-party bank website in under 4 seconds.

**Supportability:**

- Adding a bank account will be supported in Mozilla Firefox and Chrome. The mobile application will be supported in Android and IOS.

**Implementation:**

- The use case will be implemented using React and will be available on browsers and in the mobile application.

---

**Modification History:**

**Owner:** Gabriel Alfonso

**Initiation Day:** 01/16/2020

**Date Last Modified:** 04/23/2020

**Use Case ID:** MIDAS-9-View_Balance

**Use Case Level:** System-level end-to-end

**Details:**

**Actors:** User

**Pre-conditions:**

1. The user must have an account.
2. The user must be logged in.
3. The user must have added a bank account.
4. The user is on the homepage.

**Description:** The user can see the balance on their different bank accounts.

**Trigger:**

The user clicks on the Balance button. (Total net-worth can be seen before clicking the button)

The system responds by...

1. The system takes the user to the user's balance page.
2. The system uses the user's data to load their bank accounts.
3. The system displays the balance for each bank account.

**Relevant requirements:** None.

**Post-conditions:**

1. In the user's balance page, the user can see the balance for each bank account and the total net-worth from every bank account.

**Alternative courses of action:** If the user only has one bank account, the total net-worth is the same as the balance of that bank account. Therefore, the user can see the balance from the homepage without clicking the button. (In the balance button the user would see a total net-worth preview)

**Extensions:** None.

**Exceptions:**

- After clicking the balance button nothing shows up in the balance page because the user's bank account could not be loaded or the user hadn't added any yet.

**Concurrent Uses:** None.

**Related use cases:** MIDAS-8-Web-Add_Bank, MIDAS-14-Web-Login, MIDAS-5-Web-Create New_Account

---

**Decision Support:**

**Frequency:** For 10,000 users, this use is expected to occur 500 times per hour.

**Criticality:** High Criticality. Knowing the total balance for the different bank accounts is an important aspect of our budgeting app.

**Risk:** Low risk. This use case is relatively easy to implement because it is a simple fetch and display functionality.

---

**Constraints:**

**Usability:**

- The user should understand the data at first glance  and the structure of the conveyed information should be intuitive.

**Reliability:**

- Mean time to failure. - 1 failure to fetch balances out of 5000 requests is acceptable.

**Performance:**

- The balances should be fetched and displayed in less than 3 seconds.

**Supportability:**

- This use case will be supported in Mozilla Firefox and Chrome. The mobile application will be supported in Android and IOS.

**Implementation:**

- The use case will be implemented using React and will be available on browsers and in the mobile application.

---

**Modification History:**

**Owner:** Gabriel Alfonso

**Initiation Day:** 02/25/2020

**Date Last Modified:** 04/23/2020

**Use Case ID:** MIDAS-10-Credit_Score

**Use Case Level:** system-level end-to-end

**Details:**

**Actors:** User

**Pre-conditions:**

1. User must have an account.
2. User must be logged in.
3. The user is on the homepage.

**Description:** The user can link their credit score to plan their finances accordingly

**Trigger:** The user clicks on the link credit score button.

The system responds by...

1. The system takes the user to a new page
2. The system displays fields to be filled with information to verify the user identity and check if they have a credit score.
3. After filling the information, the user clicks on the submit button and the data is sent to the back-end for verification.
4. Once the system verifies the user, they are taken back to the homepage.

**Relevant requirements:** None.

**Post-conditions:**

1. While on the homepage the user can now see their credit score.

**Alternative courses of action:** The user clicks the back button to go back from the credit score linking page.

The information entered is incorrect and the system displays an error message.

**Extensions:** None.

**Exceptions:**

- The credit score for the user is not present in the database even though it should be, therefore the credit score cannot be linked to the user account.

**Concurrent Uses:** None.

**Related use cases:** None.

---

**Decision Support:**

**Frequency:** For 10,000 users, this use is expected to occur 60 times per hour.

**Criticality:** Low criticality. Linking the credit score is not a high priority for the functionality of our budgeting application.

**Risk:** Medium Risk. This use case would involve communicating with a Credit Bureau which contains the credit score of the users.

---

**Constraints:**

**Usability:**

- With the right information at hand the user should be able to link their credit score in under 3 minutes.

**Reliability:**

- Given the communication with a Credit Bureau, a 20% weekly failure rate is to be expected because of problems of record discrepancy.

**Performance:**

- After the info has been verified, the credit score should be linked to the account in under 2 seconds.

**Supportability:**

- Encryption will be supported in Mozilla Firefox and Chrome. The mobile application will be supported in Android and IOS.

**Implementation:**

- This use case will be implemented using React and will be available on browsers and in the mobile application.

---

**Modification History:**

**Owner: Gabriel Alfonso**

**Initiation Day: 02/29/2020**

**Date Last Modified: 03/03/2020**

**Use Case ID:** MIDAS-11-Create_Budget

**Use Case Level:** System-Level End-to-End

**Details:**

**Actors:** User

**Pre-conditions:**

1. The user must be logged into their account.
2. The user must be on the budget page.
3. Internet service must be available.

**Description:** The user can add a new category of expenses to track in their monthly budget.

**Trigger:** The user clicks the Add Budget Category button on the home page.

The system responds by...

1. The system prompts the user to select a budget category and displays a drop-down list with the available budget types.
2. The user clicks the drop-down list and selects one of the following categories: Rent, Shopping, Food, Travel, Transport, Entertainment, Health, Education, Family, Bills.
3. The system displays a new text field and prompts the user to specify an expense limit for their selected budget category.
4. The user types their preferred spending limit in dollars for the selected budget category.
5. The user presses the Finish button.
6. The system sends the user's username and the user's selected budget category and expense amount to the server.
7. The system saves the budget category and expense amount to the user's budget in the datastore, along with an initial value of 0 for the current month expenses under this category.
8. The system updates the budget page to display the new category and alerts the user that the new budget category was successfully added.

**Relevant requirements:** None

**Post-conditions:**

1. The user's selected budget category and expense amount is added to the list of the user's budget categories in the database.
2. The user's current month expenses for the new category is set to 0 dollars.
3. The budget page is updated and displays the recently added budget category.

**Alternative courses of action:**

1. At steps 2, 4, and 5, the user can cancel the creation of a new budget category.
2. At step 4, the user can change their budget category selection.
3. At step 5, the user can change their budget category selection and/or their specified expense limit.

**Extensions:** None

**Exceptions:**

1. The user selects a budget category that already exists in their budget.
2. The user enters an invalid dollar amount for the expense limit, i.e. a zero or negative value.
3. The user's session could timeout while they are filling out the budget creation details.
4. Error of communication to server.
5. Internet connection gets lost.

**Concurrent Uses:** None

**Related use cases:**

1. MIDAS-12-View_Budget

---

**Decision Support**

**Frequency:** The user may create a budget category at any moment. The creation for a specific category will only be done once.

**Criticality:** High, viewing budget is one of the main features of the application that is provided to the user, so the user should be able to add categories to track in their budget.

**Risk:** Low, no new skills are required to implement this use case.

---

**Constraints**

**Usability:**

- No previous training needed. Category creation will be intuitive: budget categories will show examples of types of expenses for that category, and expense limit will ask for a dollar amount to be inputted.

**Reliability:**

- Mean time to failure – 1 failure for every 30,000 category creation requests sent to the server is acceptable.

**Performance:**

- When the user clicks the Finish button, the new budget category should be created and stored within 1 second.

**Supportability:**

- The user should be able to add a budget category on the Mozilla Firefox and Google Chrome web browsers, as well as on the iOS and Android application.

**Implementation:**

- The use case will be implemented using React and will be available on browsers and in the mobile application.

---

**Modification History**

**Owner:** Edward Gil

**Initiation Day:** 2/25/2020

**Date Last Modified:** 4/23/2020

**Use Case ID:** MIDAS-12-View_Budget

**Use Case Level:** System-Level End-to-End

**Details:**

**Actors:** User

**Pre-conditions:**

1. The user must be logged into their account.
2. The user must be on the home page.
3. Internet service must be available.

**Description:** The user can view their monthly budget for all of their added budget categories. The spending limit and current month expenses will be shown for each category.

**Trigger:** The user clicks the View Budget button on the home page.

The system responds by...

1. The system redirects the user to the budget page.
2. The system displays a message saying that their budget is being loaded.
3. The system uses the user's username to fetch their list of budget categories, which includes the set monthly expense limit and the current month expenses for each category.
4. The system displays the fetched budget, divided into the categories that were retrieved from the server.
5. The system displays the user's monthly expense limit and current month expenses under each category.

**Relevant requirements:** None

**Post-conditions:**

1. The user's budget for the current month is displayed.

**Alternative courses of action:**

1. At step 2, the user can go back to the home page, before the budget loads.

**Extensions:** None

**Exceptions:**

1. The user's session could be timed out when they press the View Budget button.
2. The user has not added any categories to track in their budget.
3. Error of communication to server.
4. Internet connection gets lost.

**Concurrent Uses:**

1. MIDAS-6-Encryption: encrypt the budget data being sent from the server to protect the user's privacy regarding spending habits.

**Related use cases:**

1. MIDAS-11-Web-Create_Budget
2. MIDAS-11-Mobile-Create_Budget
3. MIDAS-13-Categorize_Transaction

---

**Decision Support**

**Frequency:** The user may want to view their budget for the month frequently. 2 times per day is expected.

**Criticality:** High, viewing budget is one of the main features of the application that is provided to the user, so that the user can track their spending and be cautious not to exceed their allowed limit.

**Risk:** Low, no new skills are required to implement this use case.

---

**Constraints**

**Usability:**

- Must be easy to understand what the user's budget for the current month is, and how it is divided into different categories. The user's current spending for each category should be intuitively displayed under the respective category.

**Reliability:**

- Mean time to failure – 1 failure for every 20,000 requests sent to the server is acceptable.

**Performance:**

- The user's budget and current month expenses should be retrieved in less than 2 seconds

**Supportability:**

- The user should be able to view their budget on the Mozilla Firefox and Google Chrome web browsers, as well as on the iOS and Android application.

**Implementation:**

- The use case will be implemented using React and will be available on browsers and in the mobile application.

---

**Modification History**

**Owner:** Edward Gil

**Initiation Day:** 2/26/2020

**Date Last Modified:** 3/3/2020

**Use Case ID:** MIDAS-13-Categorize_Transaction

**Use Case Level:** System-Level End-to-End

**Details:**

**Actors:** User

**Pre-conditions:**

1. The user must be logged into their account.
2. The user must be on the budget page.
3. Internet service must be available.
4. The user must have at least one bank account linked.

**Description:** Search through user's current month transaction history of their linked bank accounts, to show transactions which have not been categorized into the user's budget. The user can then choose to categorize a transaction under a budget category.

**Trigger:** The user clicks the Categorize Purchases button on the home page.

The system responds by...

1. The system redirects the user to the categorization page.
2. The system goes through the user's linked bank accounts to get a list of purchases that were made in the current month.
3. The system sends the user's username to the server to get the list available budget categories for the user, and also the list of user purchases that have already been categorized into the user's budget.
4. The system removes the purchases that have already been categorized, from the list in step 2.
5. The system displays the list of uncategorized purchases, with a drop-down menu to the right of each purchase, with each menu containing all the user's available budget categories.
6. The user categorizes a purchase by selecting a budget category from a drop-down menu of any of the shown purchases.
7. The system sends the user's username and the selected purchase item and budget category to the server.
8. The system saves the newly categorized purchase into the datastore and adds the purchase amount of it to the user's current expenses for that budget category.
9. The system removes the newly categorized purchase item from the user's display.
10. The system displays a message to the user saying the purchase was successfully categorized.

**Relevant requirements:**

1. A service from the third-party payment method that can be used to view the user's transaction history.

**Post-conditions:**

1. The user is on the categorization page, which shows the remaining purchases that have yet to be categorized.
2. The user's current month expenses for the budget category that the purchase item was categorized under is updated to reflect that purchase's expense amount.

**Alternative courses of action:**

1. At step 5, the user can cancel the operation of categorizing a purchase by going back to the budget page.

**Extensions:**

1. MIDAS-12-View_Budget: Fetch and display the user's updated budget expenses.

**Exceptions:**

1. The user's session times out while choosing a purchase to categorize.
2. The user attempts to categorize a purchase that has already been categorized.
3. The user has not created the budget category that a purchase falls under.
4. The system fails to receive the transaction history for one of the user's linked bank accounts.
5. Error of communication to server.
6. Internet service gets lost.

**Concurrent Uses:**

1. MIDAS-6-Encryption: encrypt purchase information to protect the user's privacy.

**Related use cases:**

1. MIDAS-11-Web-Create_Budget
2. MIDAS-11-Mobile-Create_Budget
3. MIDAS-12-View_Budget

---

**Decision Support**

**Frequency:** The user may want to categorize multiple purchases per day. Expected 10 times per user per day.

**Criticality:** High, user purchases must be categorized so that the user can see how much they have currently spent for the month.

**Risk:** High, requires learning how to interact with API's for the different available payment methods to detect recent purchases.

---

**Constraints**

**Usability:**

- No previous training needed. Selecting a budget category for a purchase will be intuitive.

**Reliability:**

- Mean time to failure – 1 failure for every 10,000 requests sent to the server is acceptable.

**Performance:**

- When the user categorizes a transaction, it should be saved to the datastore and removed from the user's current view of uncategorized purchases within 1 second.

**Supportability:**

- The user should be able to categorize transaction on the Mozilla Firefox and Google Chrome web browsers, as well as on the iOS and Android application.

**Implementation:**

- The use case will be implemented using third-party payment method APIs and (backend language), and a custom categorization engine using (machine learning service).

---

**Modification History**

**Owner:** Edward Gil

**Initiation Day:** 2/27/2020

**Date Last Modified:** 3/3/2020

**Use Case ID:** MIDAS-14-Login

**Use Case Level:** System-Level End-to-End

**Details:**

**Actors:** User

**Pre-conditions:**

1. The user should already have an account created.
2. Internet service must be available.

**Description:** The user can login to their account.

**Trigger:** The user types their account username or email, and their password and presses the Login button.

The system responds by...

1. The system displays a loading message saying that the user is being logged in.
2. The system uses the inputted username or email to fetch the user's profile.
3. The system verifies that the inputted password matches the password stored in the user's profile.
4. The system creates a new login session for the user and returns the session id.
5. The system stores the user's username and session id locally on the client side.
6. The system redirects the user to their account home page.

**Relevant requirements:** None

**Post-conditions:**

1. The user is logged into their account.

**Alternative courses of action:** None

**Extensions:** None

**Exceptions:**

1. The inputted username or email does not belong to any existing account in the system.
2. The inputted password does match the password stored in the user's profile.
3. Error of communication to server.
4. Internet connection gets lost.

**Concurrent Uses:**

1. MIDAS-6-Encryption: the user profile is encrypted when being retrieved.

**Related use cases:**

1. MIDAS-15-Web-Logout: User account logout.

---

**Decision Support**

**Frequency:** Every time the user is logged out and wants to access their account. Usually once per day.

**Criticality:** High, required for the user to have access to their account and therefore application functionality.

**Risk:** Medium, requires learning how to create a user login session and store the session id on the client side, so that the user can later be logged out of the session.

---

**Constraints**

**Usability:**

- The user should immediately be able to figure out where to input their account username/email and password and then simply click the login button.

**Reliability:**

- Mean time to failure – 1 failure for every 40,000 login attempts sent to the server is acceptable.

**Performance:**

- The user should be logged into the system within 2 seconds of pressing the login button.

**Supportability:**

- The user should be able to login to their account on the Mozilla Firefox and Google Chrome web browsers, as well as on the iOS and Android application.

**Implementation:**

- The use case will be implemented using React and will be available on browsers and in the mobile application.

---

**Modification History**

**Owner:** Edward Gil

**Initiation Day:** 2/26/2020

**Date Last Modified:** 4/23/2020

**Use Case ID:** MIDAS-15-Logout

**Use Case Level:** System-Level End-to-End

**Details:**

**Actors:** User

**Pre-conditions:**

1. The user should have an account created and be logged in.
2. Internet service must be available.

**Description:** The user can logout of their account.

**Trigger:** The user presses the logout button on the top-right of the navigation bar.

The system responds by...

1. The system displays a loading message saying that the user is being logged out.
2. The system sends the session id to the server.
3. The system removes the session id from the list of active sessions.
4. The system redirects the user to the login page.
5. The system displays a message saying that the user has been logged out successfully.

**Relevant requirements:** None

**Post-conditions:**

1. The user is logged out of their account.
2. The session id is now invalid and should be removed from the client side.

**Alternative courses of action:** None

**Extensions:** None

**Exceptions:**

1. The session was already timed out before the user presses the logout button.
2. Error of communication to server.
3. Internet connection gets lost.

**Concurrent Uses:** None

**Related use cases:**

1. MIDAS-14-Web-Login: User account login.

---

**Decision Support**

**Frequency:** Every time the user is logged in and wants to log out. Usually once per day.

**Criticality:** High, required to restrict access to account when it is not currently in use by the user.

**Risk:** Low, no new skills required to implement this use case.

---

**Constraints**

**Usability:**

- Logging out is intuitive, since there will be a log out button accessible at the top-right of the application, on a navigation bar.

**Reliability:**

- Mean time to failure – 1 failure for every 50,000 logout attempts sent to the server is acceptable.

**Performance:**

- The user should be logged out of the system within 2 seconds of pressing the logout button.

**Supportability:**

- The user should be able to logout of their account on the Mozilla Firefox and Google Chrome web browsers, as well as on the iOS and Android application.

**Implementation:**

- The use case will be implemented using React and will be available on browsers and in the mobile application.

---

**Modification History**

**Owner:** Edward Gil

**Initiation Day:** 2/26/2020

**Date Last Modified:** 4/23/2020

## 6.3. Appendix C – Detail description of effort estimation (include appropriate figures and diagrams)

To estimate the effort required for the project, function points were first derived. The function points are derived from estimations of the following 5 functional measures:

1. User Inputs – The number of inputs from the user that is required by the system.
2. User Outputs – The number of elements in the system that produce an output.
3. User Inquiries – The number of user inputs that generate a software response
4. Internal Logical Files – The number of files created and used by the system.
5. External Interfaces – The number of external elements that the system communicates with.

| | Functional Measures | | | | | |
|---|---|---|---|---|---|---|
| Use Cases | User Inputs | User Outputs | User Inquiries | Internal Logical Files | External Interfaces | Total |
| MIDAS-1-Captcha | 1 | 1 | 0 | 1 | 1 | **4** |
| MIDAS-2-Obfuscation | 1 | 1 | 0 | 0 | 0 | **2** |
| MIDAS-3-View User Stocks | 1 | 1 | 0 | 0 | 1 | **3** |
| MIDAS-4-Add Stock | 2 | 1 | 1 | 0 | 2 | **6** |
| MIDAS-5-Create New Account | 3 | 1 | 0 | 0 | 1 | **5** |

| | | | | | | |
|---|---|---|---|---|---|---|
| MIDAS-6-Encryption | 1 | 1 | 0 | 1 | 1 | **4** |
| MIDAS-7-Timeout | 0 | 1 | 0 | 0 | 1 | **2** |
| MIDAS-8-Add Bank | 1 | 1 | 1 | 0 | 2 | **5** |
| MIDAS-9-View Balance | 1 | 1 | 0 | 0 | 2 | **4** |
| MIDAS-10-Credit Score | 2 | 1 | 0 | 0 | 2 | **5** |
| MIDAS-11-Create Budget | 3 | 1 | 1 | 0 | 1 | **6** |
| MIDAS-12-View Budget | 1 | 1 | 0 | 0 | 1 | **3** |
| MIDAS-13-Categorize Transaction | 2 | 2 | 0 | 0 | 1 | **5** |
| MIDAS-14-Login | 2 | 1 | 0 | 0 | 1 | **4** |
| MIDAS-15-Logout | 1 | 1 | 0 | 0 | 1 | **3** |
| Table 6.2.a - Estimated Function Points | | | | | | **61** |

For the 15 implemented use cases, an estimate of 61 total function points is derived, as seen in Table 6.2.a. These function points were then input into a calculator that uses the COCOMO model to compute the effort required in person-months[3]. For the scale and cost drivers, nominal values were used. Figure 6.2.a displays the

computed effort, along with the estimated software cost of $84018, based on an estimated $5000 cost per person-month.

**Software Development (Elaboration and Construction)**

Effort = 16.8 Person-months
Schedule = 9.3 Months
Cost = $84018

Total Equivalent Size = 4880 SLOC

**Acquisition Phase Distribution**

| Phase | Effort (Person-months) | Schedule (Months) | Average Staff | Cost (Dollars) |
|---|---|---|---|---|
| Inception | 1.0 | 1.2 | 0.9 | $5041 |
| Elaboration | 4.0 | 3.5 | 1.2 | $20164 |
| Construction | 12.8 | 5.8 | 2.2 | $63854 |
| Transition | 2.0 | 1.2 | 1.7 | $10082 |

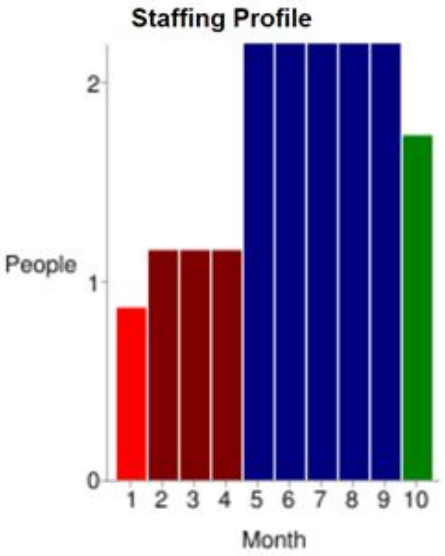



Figure 6.2.a - Effort Estimation/Cost Summary

## 6.4.Appendix D

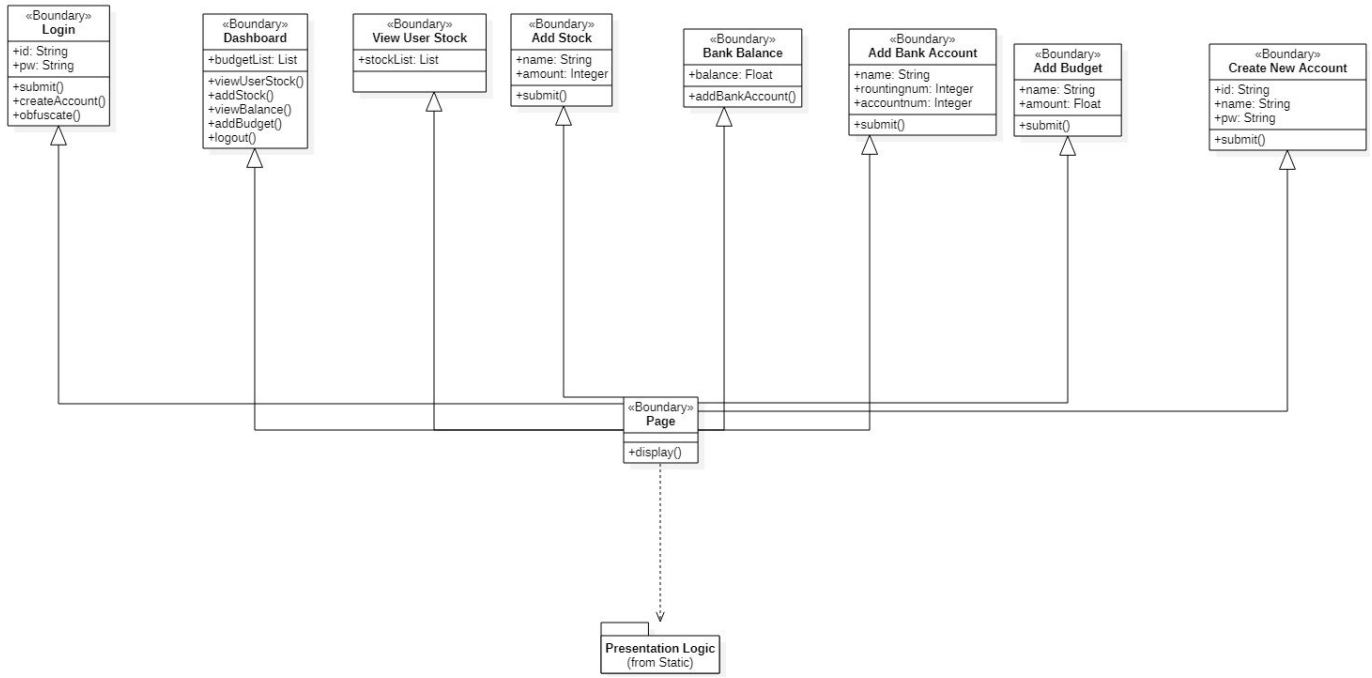


Figure 6.4.a ***Client Class Diagram***

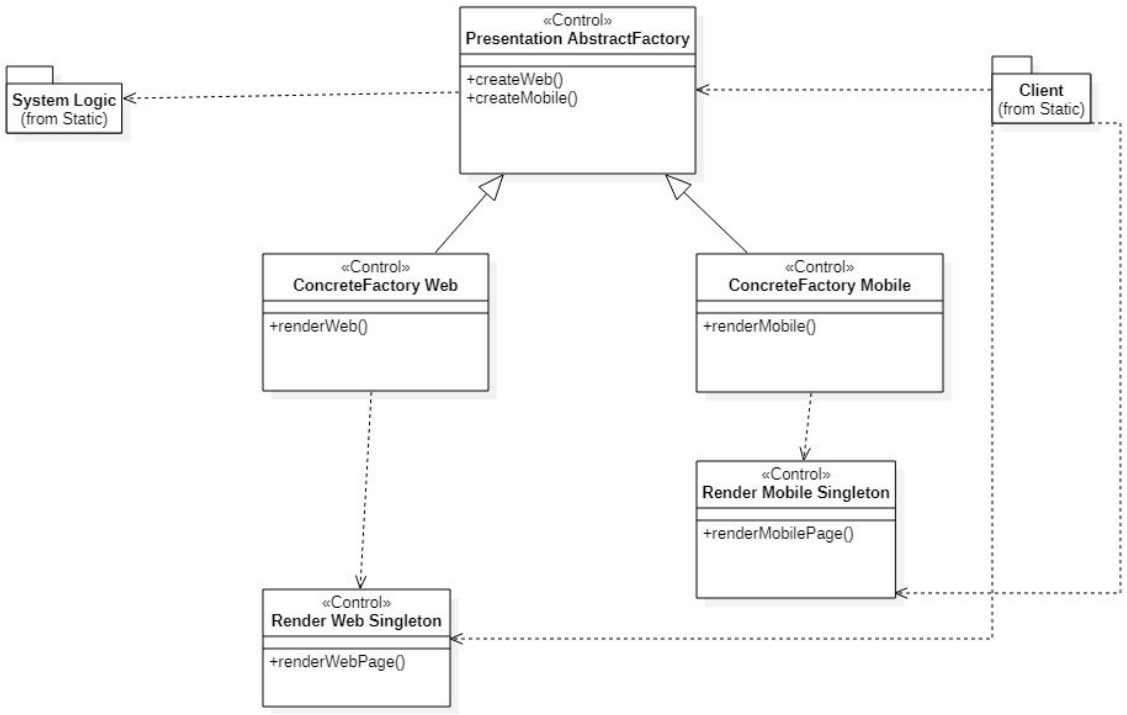



Figure 6.4.b ***Presentation Logic Class Diagram***

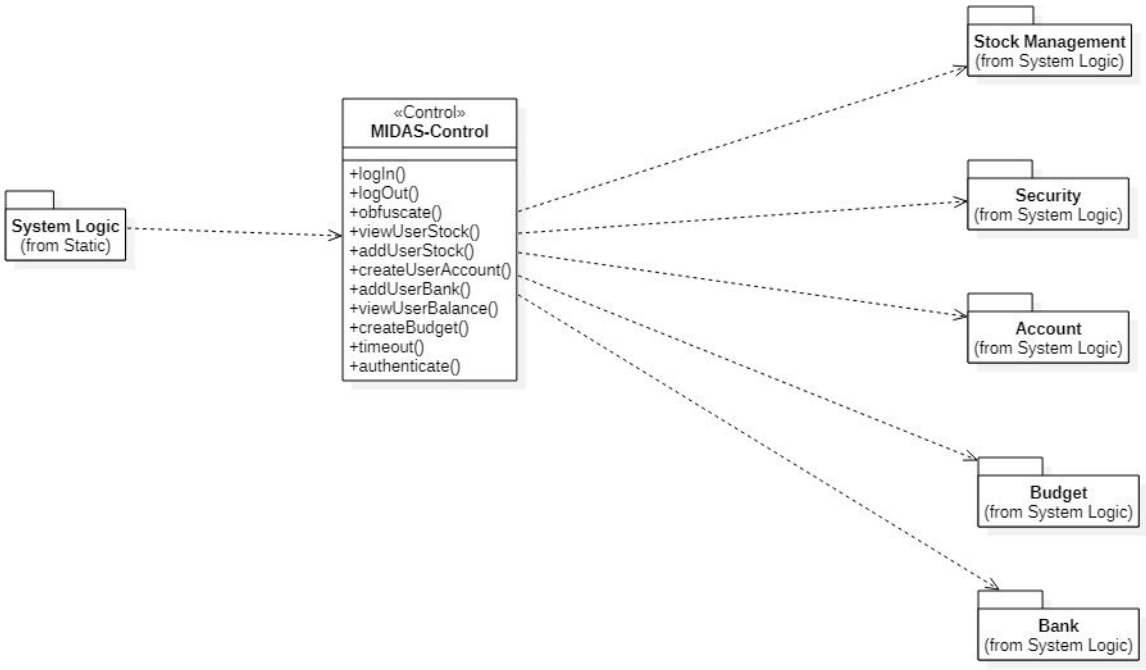



Figure 6.4.c ***Service Class Diagram***

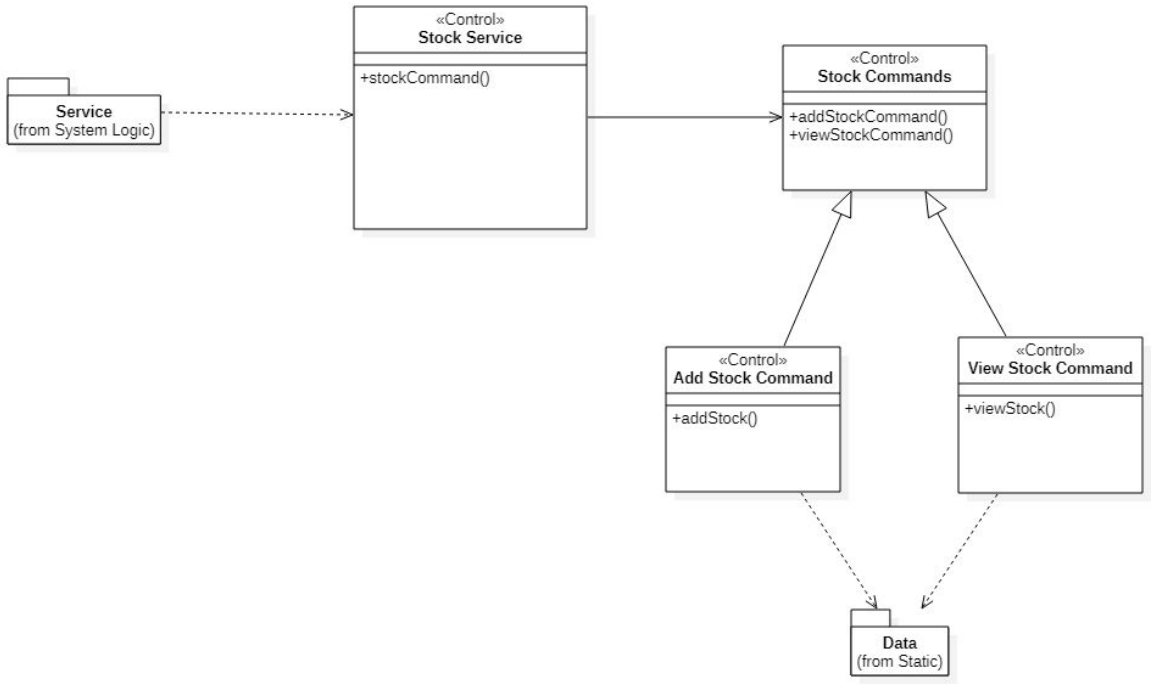



Figure 6.4.d ***Stock Management Class Diagram***

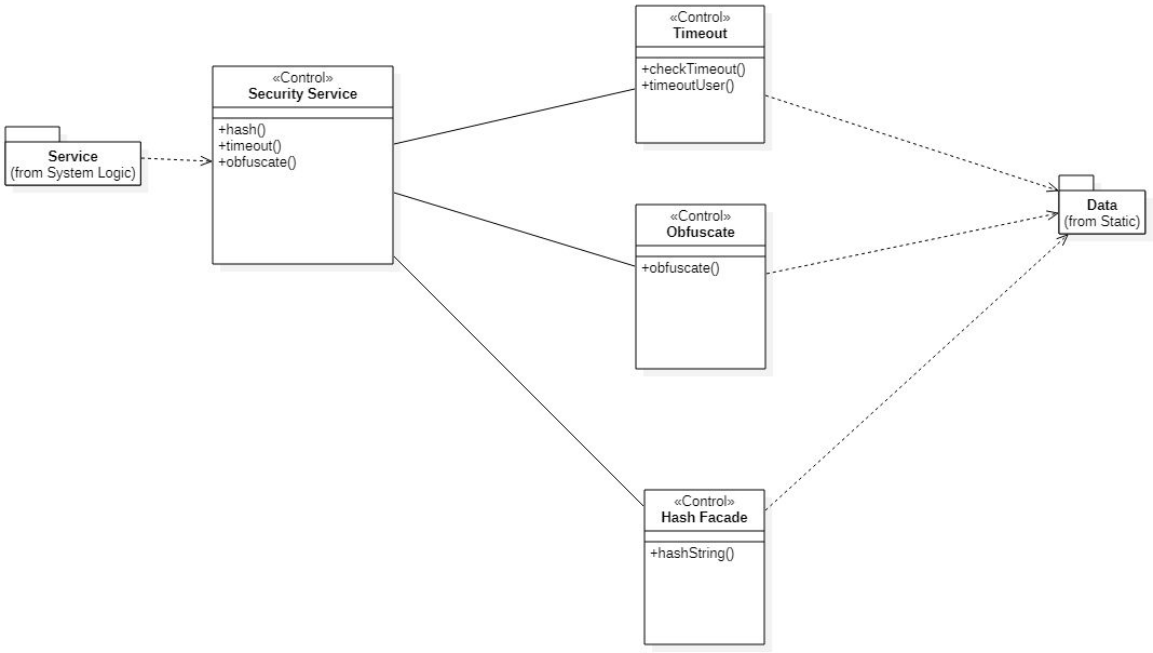



Figure 6.4.e ***Security Class Diagram***

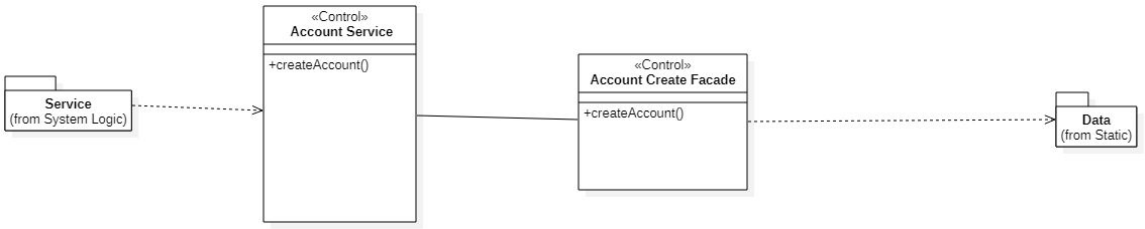



Figure 6.4.f ***Account Class Diagram***

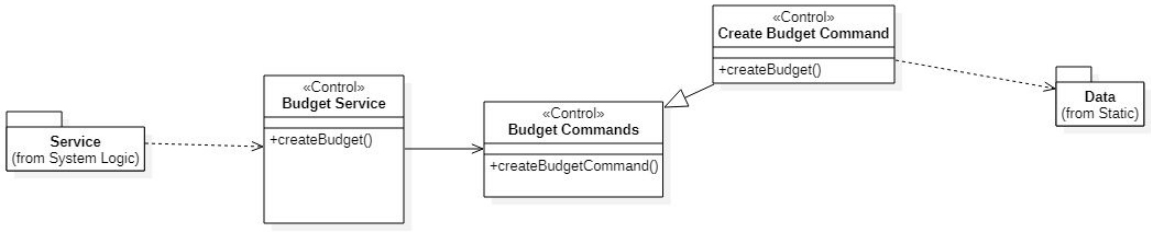



Figure 6.4.g ***Budget Class Diagram***

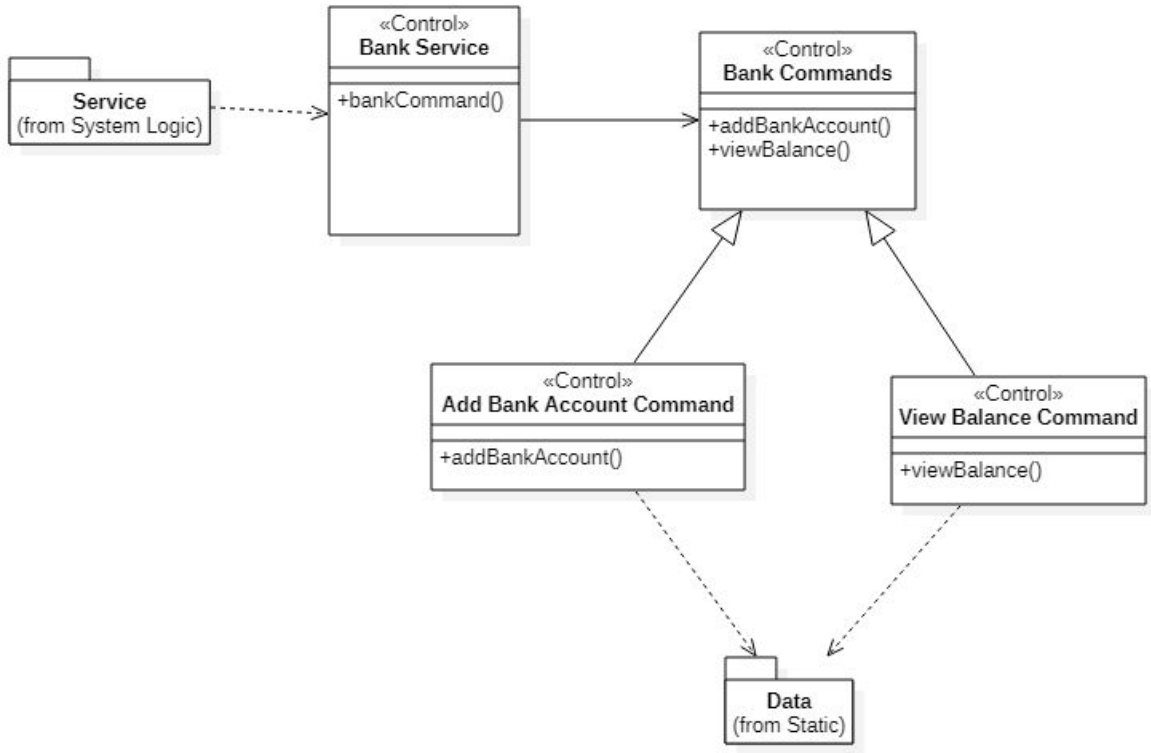



Figure 6.4.h **Bank  Class Diagram**

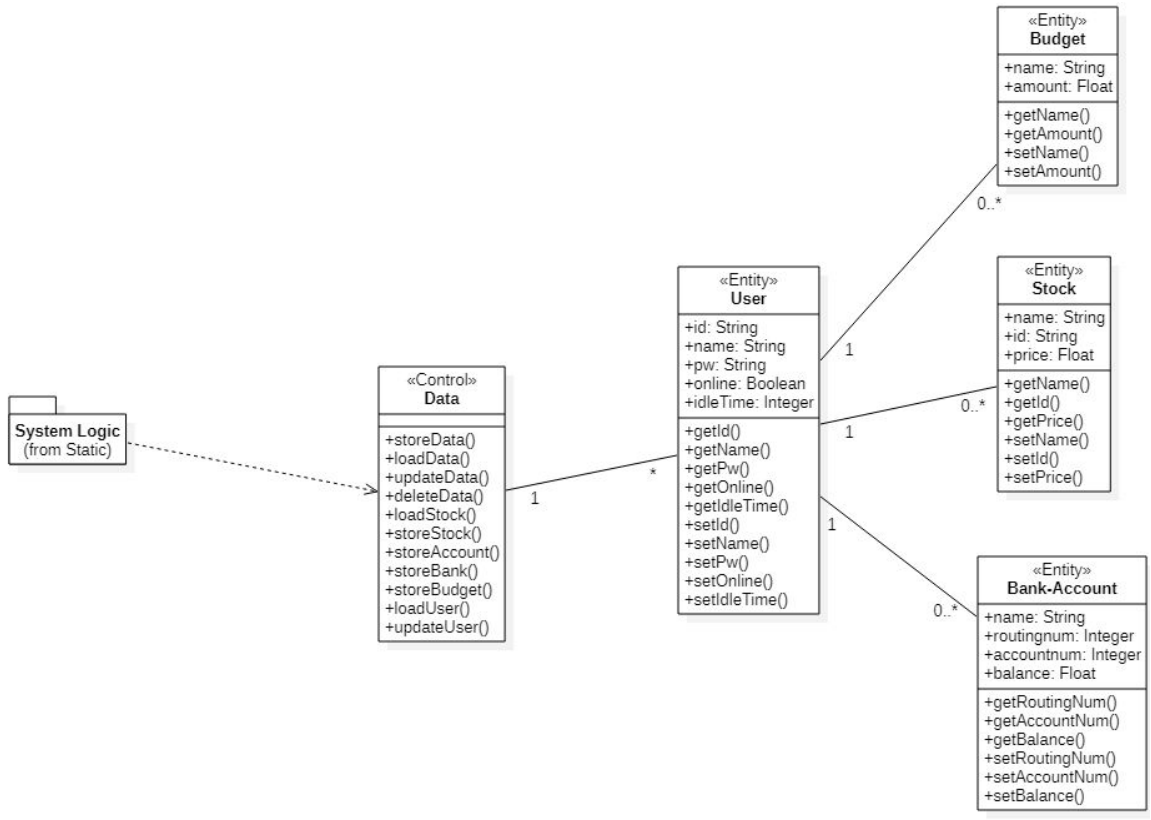



Figure 6.4.i ***Data Class Diagram***

## 6.5.Appendix E – Refined sequence diagrams

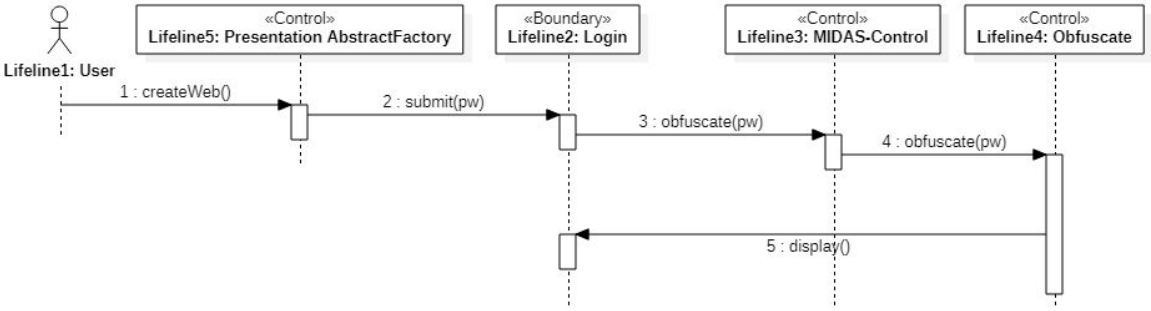



Figure 6.5.a ***MIDAS-2-Obfuscation Sequence Diagram***

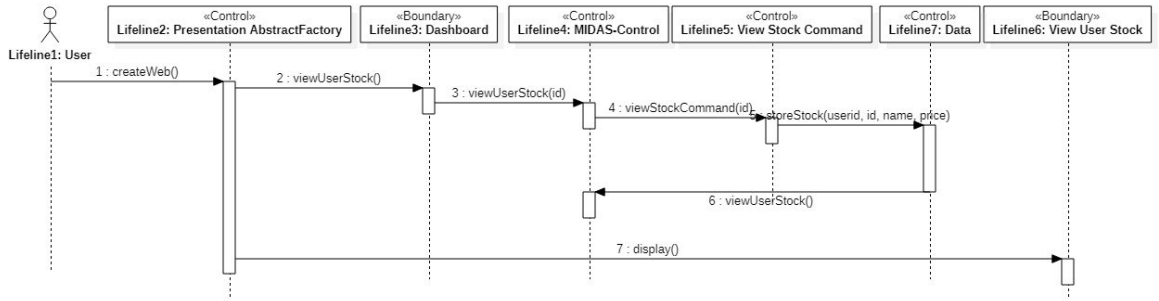



Figure 6.5.b ***MIDAS-3-View User Stocks Sequence Diagram***

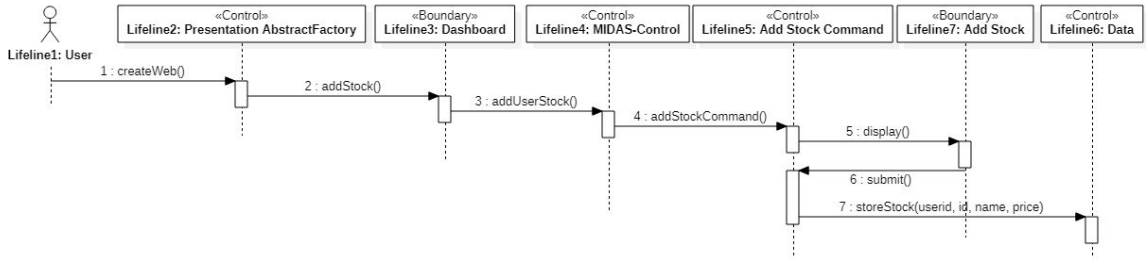



Figure 6.5.c ***MIDAS-4-Add Stock Sequence Diagram***

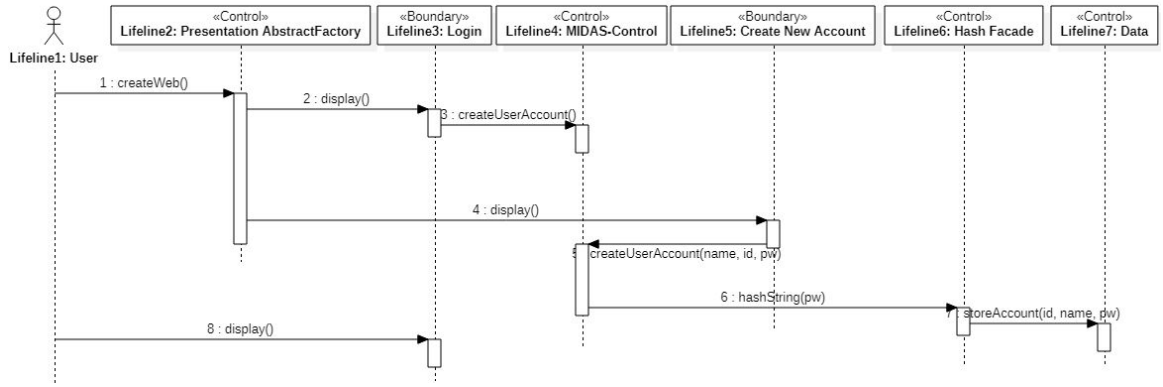



Figure 6.5.d ***MIDAS-5-Create New Account Sequence Diagram***

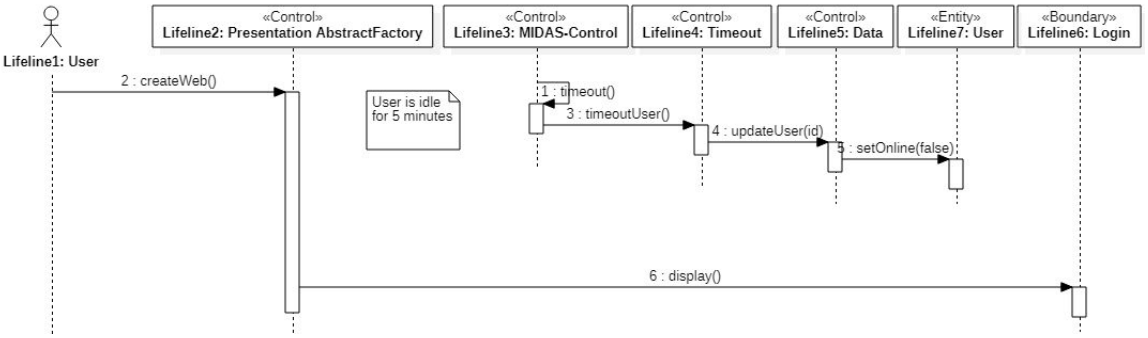



Figure 6.5.e ***MIDAS-7-Timeout Sequence Diagram***

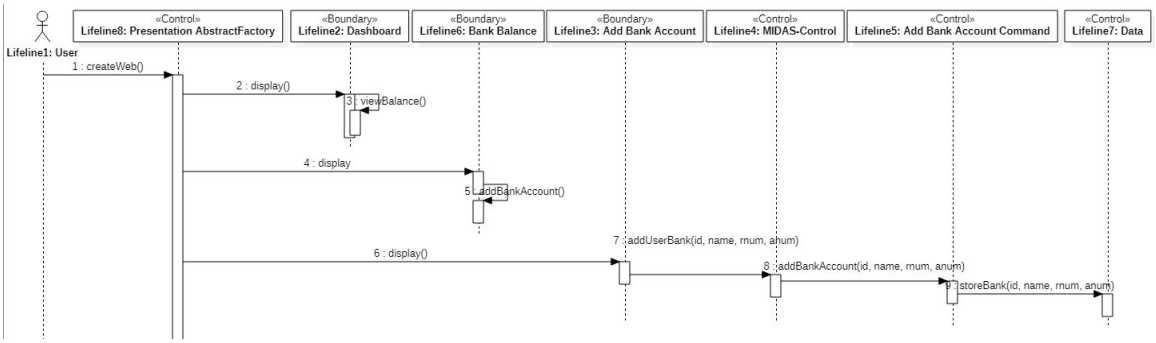



Figure 6.5.f ***MIDAS-8-Add Bank Account Sequence Diagram***

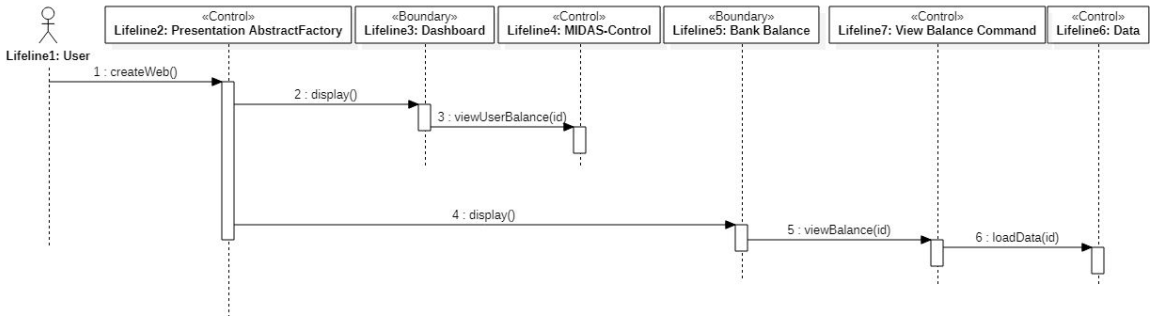



Figure 6.5.g ***MIDAS-9-View Balance Sequence Diagram***

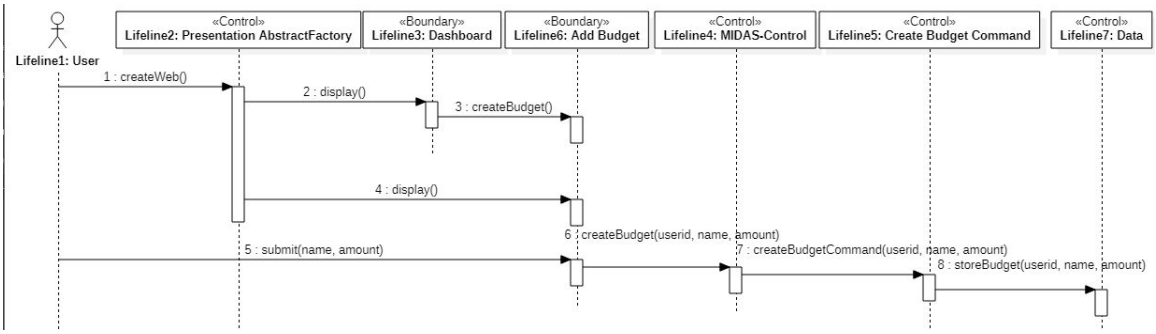



Figure 6.5.h ***MIDAS-11-Create Budget Sequence Diagram***

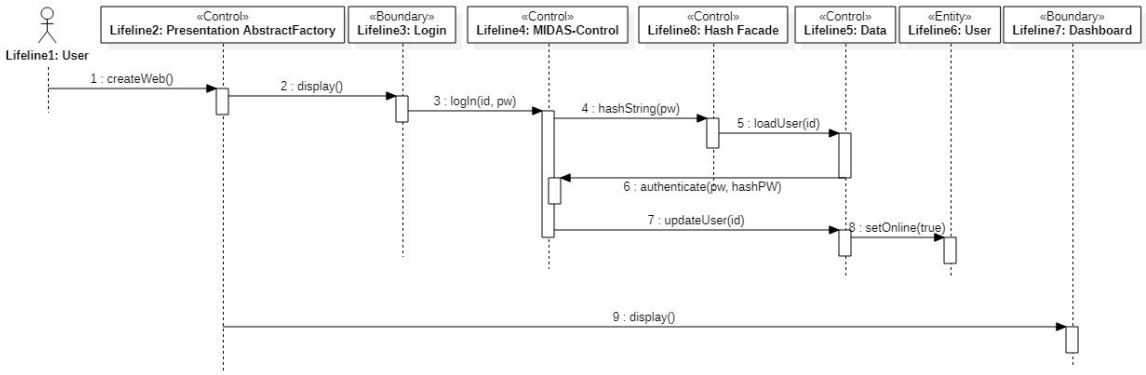



Figure 6.5.i ***MIDAS-14-Login Sequence Diagram***

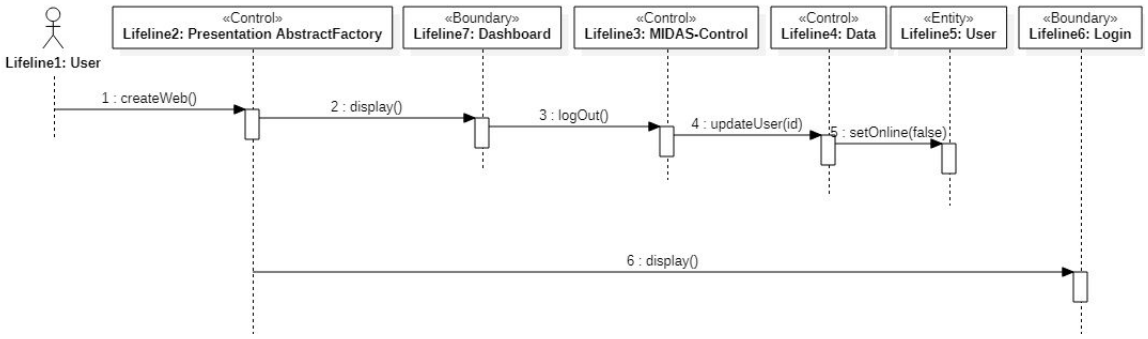



Figure 6.5.j ***MIDAS-15-Logout Sequence Diagram***

## 6.6.Appendix F - Diary of Meetings

**Date**: March 19, 2020

**Time**: 9:11 P.M. – 9:17 P.M.

**Location**: Zoom virtual meeting

**Persons in Attendance**:

- Gabriel Alonso – Team Leader
- Manuel Toledo – Minute Taker
- Edward Gil – Time Keeper

**Persons Late**: None

**Summary of Discussion**: Assigned project roles and technical roles. Gabriel is team leader/quality analyst, Manuel is minute taker/uml engineer, Edward is time keeper/business analyst.

**Assigned Tasks**: Edward will begin looking into the COCOMO effort estimation model. Manuel will continue working on the UML diagrams. Gabriel should continue updating the project schedule.

**Date**: March 26, 2020

**Time**: 9:20 P.M. – 9:27 P.M.

**Location**: Zoom virtual meeting

**Persons in Attendance**:

- Gabriel Alonso – Team Leader
- Manuel Toledo – Minute Taker
- Edward Gil – Time Keeper

**Persons Late**: None

**Summary of Discussion**: Created initial assignments of deliverable 2 sections. Manuel will work on system design and detailed design, with help from Edward and Gabriel if needed. Gabriel will continue working on the project plan, schedule integrity, and completeness of function. Edward will work on the COCOMO effort estimation, risk management plan, and quality/budget monitoring.

**Assigned Tasks**: Team members will continue working on their initially assigned parts.

Date: April 14, 2020

Time: 9:29 P.M. – 9:33 P.M.

Location: Zoom virtual meeting

Persons in Attendance:

- Gabriel Alonso – Team Leader
- Manuel Toledo – Minute Taker

- Edward Gil – Time Keeper

Persons Late: None

Summary of Discussion: Talked about the upcoming presentation, and who will do what parts. Manuel will present the UML diagram slides, and demo StarUML. Gabriel will present a use case and talk about the completeness of function, and demo the project schedule. Edward will introduce the problem definition, risk management plan, and present the other use case.

Assigned Tasks: All team members begin to work on the presentation slides and rehearse their parts.

## 7. References

[1] Tsui, F. (2008). Managing software projects. New Delhi.: Viva Books.

[2] Overview of COCOMO. (n.d.). Retrieved from http://www.softstarsystems.com/overview.htm

[3] COCOMO II - Constructive Cost Model. Retrieved from http://softwarecost.org/tools/COCOMO/