CEN 4021 Software Engineering 2 U01 1201

MIDAS

Team 1:

Manuel Toledo

Gabriel Alfonso

Edward Gil

Dr. Peter Clarke

3/3/2020

Abstract

The following document will go over the feasibility study, project plan, requirements elicitation, and requirements analysis of Team 1's MIDAS project. The feasibility study section will include the limitations and constraints of MIDAS, its purpose as a system, its user requirements, alternative solutions and recommendations. Project plan will include the work breakdown and personnel organization of the project. The requirements elicitation will contain the the system requirements and the use case diagrams. The requirements analysis portion will contain most of the uml diagrams including the sequence, class and object diagrams.

Table of Contents

1. Introduction

This chapter introduces the project by describing the problem that it is trying to solve, with some information on why the problem exists. This chapter will also provide a brief overview of what the rest of this document entails.

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. Background

Technology is rapidly advancing as the years go by. With these advancements, users are provided with a means of easily opening and managing financial accounts online. Some of these financial accounts include bank accounts and investment accounts. There are various applications that are used to track and manage these accounts, but many are very specialized. For example, there may be a budgeting application that can track bank account transactions to provide the user with the ability to create a monthly budget, and there may be an application that allows users to make investments in the stock market. As previously mentioned, with the ability for users to easily open multiple financial accounts, a user may find it more convenient to have an application that can provide both budgeting and investments functionality, as well as additional means of tracking financial information. Without this sort of all-in-one application, it becomes tedious for users to track their accounts and finances through multiple applications, as users may have to manually input duplicate information into each application.

1.3. 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
CAPTCHA	A challenge-response test used to determine whether a user is a human.
API	Application Programming Interface - a set of functions that can be used to access the features or data of an application, or other service.

1.4. Overview of document

This document contains the results of a feasibility study conducted to find a preferred alternative solution for the proposed system. The study analyzed three alternatives and compared them by multiple feasibility criteria. The project plan chapter of the document goes over the work breakdown schedule for the project, as well as the project personnel and hardware/software resources. A Gantt chart was created to visualize the tasks and milestones that will be completed throughout the semester. The requirements elicitation chapter takes a look at the functional and nonfunctional requirements for the project, as well as shows the use case diagram. The requirements analysis chapter will showcase 10 use cases that are planned on being implemented. A scenario, object diagram, and sequence diagram are created for each of the 10 use cases. The document also includes an appendix which contains the project schedule figure, the feasibility analysis and cost matrices, the use cases and the meeting minutes recorded during team meetings.

2. Feasibility Study

This chapter takes a deeper look at the proposed system to assess the feasibility of the project, the value that it can bring, and the costs that the solution requires. Various systems already in existence are reviewed and compared to the proposed new system, to highlight the new system's purpose. At the end of the chapter, three candidate solutions are listed and analyzed, to recommend a preferred alternative.

2.1. Description of Current System (Limitations and Constraints)

This subsection looks at three current finance systems and identifies their limitations and constraints, to detail what can be improved upon with the new system. These three systems were chosen to be studied because they each provide users with personal financial information that will be required in the new system.

2.1.1 Mint

Mint is a popular budget tool that allows users to track their spending and investments. With Mint, a user can link their bank accounts to view balances and transactions. A major constraint on this system is that Mint is a read-only service, meaning users cannot perform any sort of transactions using the accounts that they have linked. While this provides users with peace of mind for not having to worry about unauthorized transactions, the downside is that it can inconvenience users who are looking for a service that can provide additional functionality that may include performing transactions.

2.1.2 Robinhood

Robinhood is an investing app that allows users to link bank accounts to fund their Robinhood account and begin trading stocks, funds, options, and even cryptocurrencies. A limitation of Robinhood is that it specializes in investments, and does not provide the user with any other functionality that can be used to track finances that do not involve investments.

2.1.3 Credit Karma

Credit Karma is another well-known financial tool that lets users view their credit score and credit reports, using information pulled from two of the three major credit bureaus. This application can be very useful for users looking to view their credit score for free, to get a quick overview of their financial health. However, similar to Robinhood, Credit Karma is very specialized, dealing specifically with credit-related information. A user cannot link any bank accounts with Credit Karma to track more financial information.

2.2. Purpose of New System

The main problem with the previously mentioned systems is that on their own, they provide the user with a limited amount of functionality regarding tracking finances. The purpose of the new system is to merge many of the main features that are provided by the systems that were described. This allows users to conveniently keep track of their spending, investments, payments, and credit score, all from one place. With most people having multiple checking and savings accounts, it can be very inconvenient for a user to have multiple applications where each requires the user to link their bank accounts. If the user wants to open a new bank account, the user will have to go and link their account to each of their finance apps. Not only can this be tedious for users, but it introduces a security risk because the user has their financial information in various places.

2.3. High-level Definition of User Requirements

Users will be able to create an account on the system, of which they can log into and out of, both on the website and the mobile apps. A login session will be created when a user signs into their account from a web browser; the session will be stored locally and can be timed out after a long period of inactivity, for security purposes.

The main functionality that users will require are purchasing stocks, linking bank accounts, creating a monthly budget, and viewing their credit score. The system will provide the user with an investment portfolio, from which they can keep track of stock

investments that they can make. The user will also be able to link their bank accounts so they can easily view balances and transactions. The system will allow the user to create a monthly budget to keep track of expenses. The user must manually categorize their transactions under the appropriate budget categories. Additionally, the user will be provided with a means of viewing their credit score, to view their overall financial health.

Security requirements include password obfuscation, so that passwords are hidden while being inputted for a login attempt. Encryption will also be used when transmitting sensitive data such as passwords over a network to the server, or vice versa. User privacy regarding financial information will be accessible only by the user that the information belongs to. A user on the system will have no way to access another user's financial information. Additionally, user information will not be sold to third parties for data collection or any other purpose.

2.4. Alternative Solutions

This section will describe and analyze three candidate solutions for the proposed system.

2.4.1. Description of Alternatives

The three alternatives will function the same way, with the difference between them being the platforms that are supported. The first candidate is a web-based application and an Android application. The mobile app will support Android operating system versions 5.0 and up. The second alternative is the same as the first alternative, but with the addition of a mobile iOS application, which will support iOS versions 8 and up. The third alternative includes the same as the second, but with the addition of a Windows desktop application, which will support Windows versions 8.1 and up.

2.4.2. Selection Criteria

There are four feasibility criteria that were used to analyze the alternatives: operational, technical, economic, and schedule feasibility. The operational feasibility portion of the study looked at how well the proposed

system solves the problem, and whether the proposed system will be used if developed and implemented. Technical feasibility considers the technical resources and expertise that will be required for the proposed system. Economic feasibility takes a look at the costs required to implement the system. Schedule feasibility looks at the time required to design the project.

2.4.3. Analysis of Alternatives (refer to Appendix C – Feasibility)

This section analyzes the three alternative solutions, based on the feasibility criteria described. Under each feasibility criteria, the three alternatives are analyzed to find their advantages or disadvantages. The feasibility analysis matrix that contains the scores and ranking for each candidate is shown in Appendix B.

2.4.3.1 Operational Feasibility

The three alternatives will fully support the user requirements. They differ in the platforms that will be supported for release. The first candidate will only have a website and Android application. This alternative is advantageous because the product only requires focus on two platforms. The disadvantage comes from leaving out some of the market share that comes from iPhones. While Android has a fairly large global market share of 74% for smartphones, the 23% of market share that iPhones have is still large enough to leave out many potential customers [1]. With candidate 2, an iOS app will also be produced, covering 97% of the mobile phone users. This solution will likely be very well received by users, considering that mobile phones make up more than half of all web traffic, and that mobile phone usage is likely to continue increasing in the coming years [2]. The third alternative builds upon the second by including a Windows desktop application. This solution can be used by users who prefer a desktop application, although it may not be worth the extra production efforts considering mobile usage is increasing far greater than desktop usage. For this reason, the second

candidate received the highest score of 95 for operational feasibility, shown in the feasibility analysis matrix in Appendix B.

2.4.3.2 Technical Feasibility

All three candidates will require the same technological resources for building the system, which include a development server, database management software, and development frameworks. The required technologies are mature and should not pose any problems. The difference in the technical feasibility of the alternatives comes from the lack of technical expertise. The first candidate requires knowledge of the Java language, which the developers will already know. The second alternative requires the developers to be trained in iOS development. The third alternative requires both iOS and Windows development training. Therefore, the first candidate is the most technically feasible alternative.

2.4.3.3 Economic Feasibility

The cost of each alternative was estimated, and the detailed calculations are shown in Appendix C. The first alternative has the lowest cost of development, making it the most economically feasible alternative of the three. The second alternative is 23% more expensive than the first alternative, and the third alternative is 12% more expensive than the second alternative, or 42% more expensive than the first alternative.

2.4.3.4 Schedule Feasibility

All three candidates will have a web and mobile design created, as well as the system architecture defined within three months, so they all received an equal score of 100 for schedule feasibility.

2.5. Recommendations

The feasibility analysis matrix shows the final ranking for each alternative. Candidate 2 received the highest ranking of the three. Alternative 1 had the cheapest development costs, and required the least amount of training for development, but candidate 2 provided the greatest benefits in terms of usability, as it provides users with the option of an iOS application. The extra development effort required for alternative 3 is not worth the costs, considering that the desktop application will provide marginal benefits in terms of usability, as users would likely prefer the mobile application over the desktop application. Because of these analyses, candidate 2 is recommended as the most feasible solution for the proposed system.

3. Project Plan

3.1. Project Goals

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.

3.2. Project Organization

3.2.1. Project Personnel Organization

Roles:

- Manuel Toledo: Team Leader, Requirements Engineer
- Edward Gil: Minute Taker, Business Analyst
- Gabriel Alfonso: Time Keeper, Quality Analyst

3.2.2. Hardware and Software Resources

Hardware

- Device to run planning software and UML software (computer)

Software

- Support on Chrome and Firefox
- Mobile support on iOS and Android
- React

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

Task ID	Descript.	Duration
1	Define project scope	3 days
2	Gather requirements	2 days
3	Define Project Idea	2 days
4	Kickoff meeting	1 day
5	Feasibility Study	2 weeks
6	Project Planning	8 days
7	Requirements Elicitation	6 days
8	Requirements Analysis	6 days
9	Write Paper	1 month
10	Make Presentation	3 days

Milestones	Deliverables	Dependencies
Kickoff meeting	Deliverable 1	Task 2 on task 1
Write Paper		Task 3 on task 2
		Task 4 on task 3
		Task 5 on task 4
		Task 8 on task 7
		Task 10 on task 9

4. Requirements Elicitation

This chapter shows the requirements elicitation section of the project. More specifically the functional requirements which describe the high-level functionality for all the project's use cases, and non-functional requirements which describe the user level requirements not directly related to functionality will be present. The use case diagram for the project will also be shown

4.1. Functional Requirements

MIDAS-1-Captcha

The system shall display a captcha system to the User and authenticate User's login

MIDAS-2-Obfuscation

The system shall change the displayed user password to dots and shorten the password.

MIDAS-3-View User Stocks

The system shall change the displayed screen to the view user page and pull stock state from the database.

MIDAS-4-Add Stock

The system shall change the displayed screen to the add stock page.

The system shall accept input name of the stock and amount from the user.

MIDAS-5-Create New Account

The system shall change the displayed screen to the create new account page.

The system shall accept input from the user, name, user id, and password.

MIDAS-6-Encryption

The system shall provide Admin with encryption keys to decrypt data sent from User

MIDAS-7-Timeout

The system shall logout the user if the user is inactive for 10 minutes.

MIDAS-8-Add_Bank

The system shall display the add bank page.

The system shall accept input from User, bank name, routing number, account number.

MIDAS-9-View_Balance

The system shall display the view balance page.

The system shall load balance data from the database.

MIDAS-10-Credit_Score

The system shall display the credit score page.

The system shall accept input from the User.

MIDAS-11-Web-Create_Budget

The system shall accept input from the user: budget name, amount.

MIDAS-12-View_Budget

The system shall display the view budget page.

The system shall load the budget information from the database.

MIDAS-13-Categorize_Transaction

The system shall display the categorization page.

The system shall accept input from the user pertaining to categorizing the users transactions.

MIDAS-14-Login

The system shall display a login screen.

The system shall accept input from the user: user id, pw.

The system shall authenticate the inputted values.

The system shall update the users online value in the database to True..

The system shall display the dashboard screen.

MIDAS-15-Logout

The system shall update the users online value in the database to False.

The system shall display the login screen.

4.2. Non-functional requirements.

The user level requirements are a Mozilla Firefox browser, or a Google Chrome Browser. The user can also access the application via their smartphone using the MIDAS app. The MIDAS app is available on both Android and IOS.

Usability: Very little training should be needed to be able to access MIDAS.

Performance: Most of MIDAS pages and functions should be viewable and complete within a second of being called.

Supportability: MIDAS should be supported by Mozilla Firefox and Chrome browsers. MIDAS should work properly on Android and IOS.

Implementation: MIDAS will be developed using React.

4.3. Use case diagram

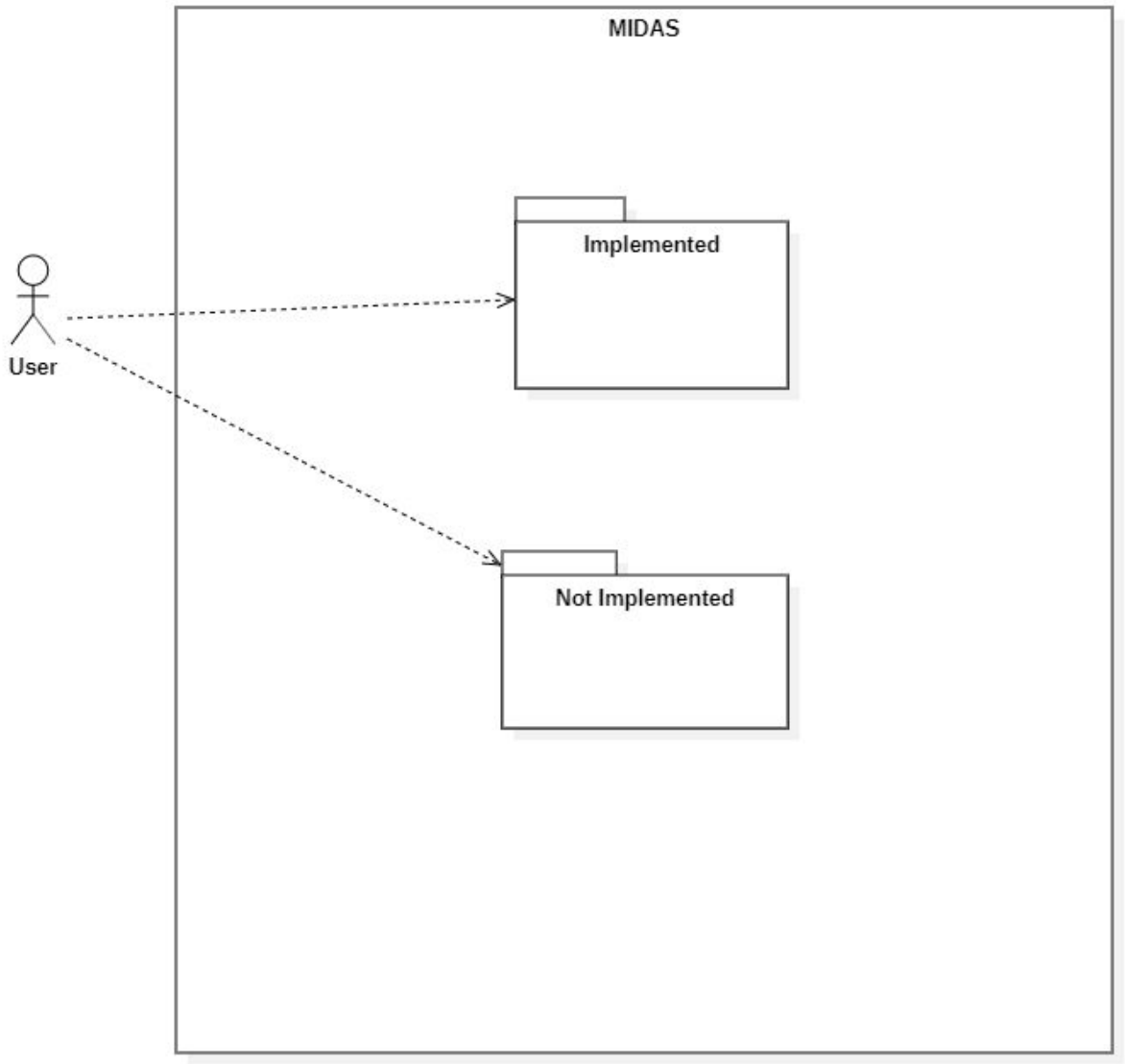


Figure 4.3.a MIDAS use case diagram containing the packages *Implemented* and *Not Implemented*

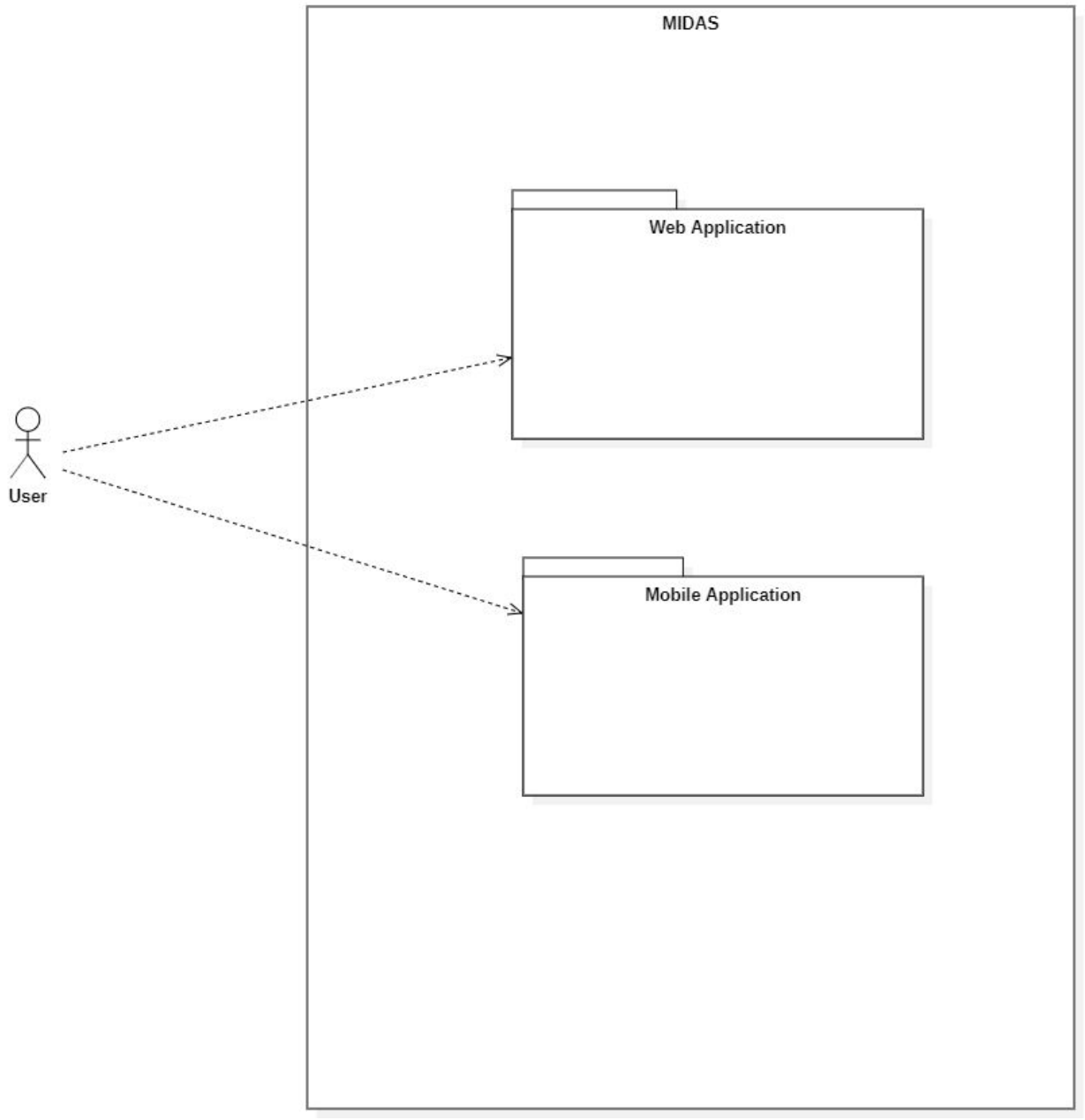


Figure 4.3.b MIDAS implemented use case diagram containing the packages *Web Application* and *Mobile Application*

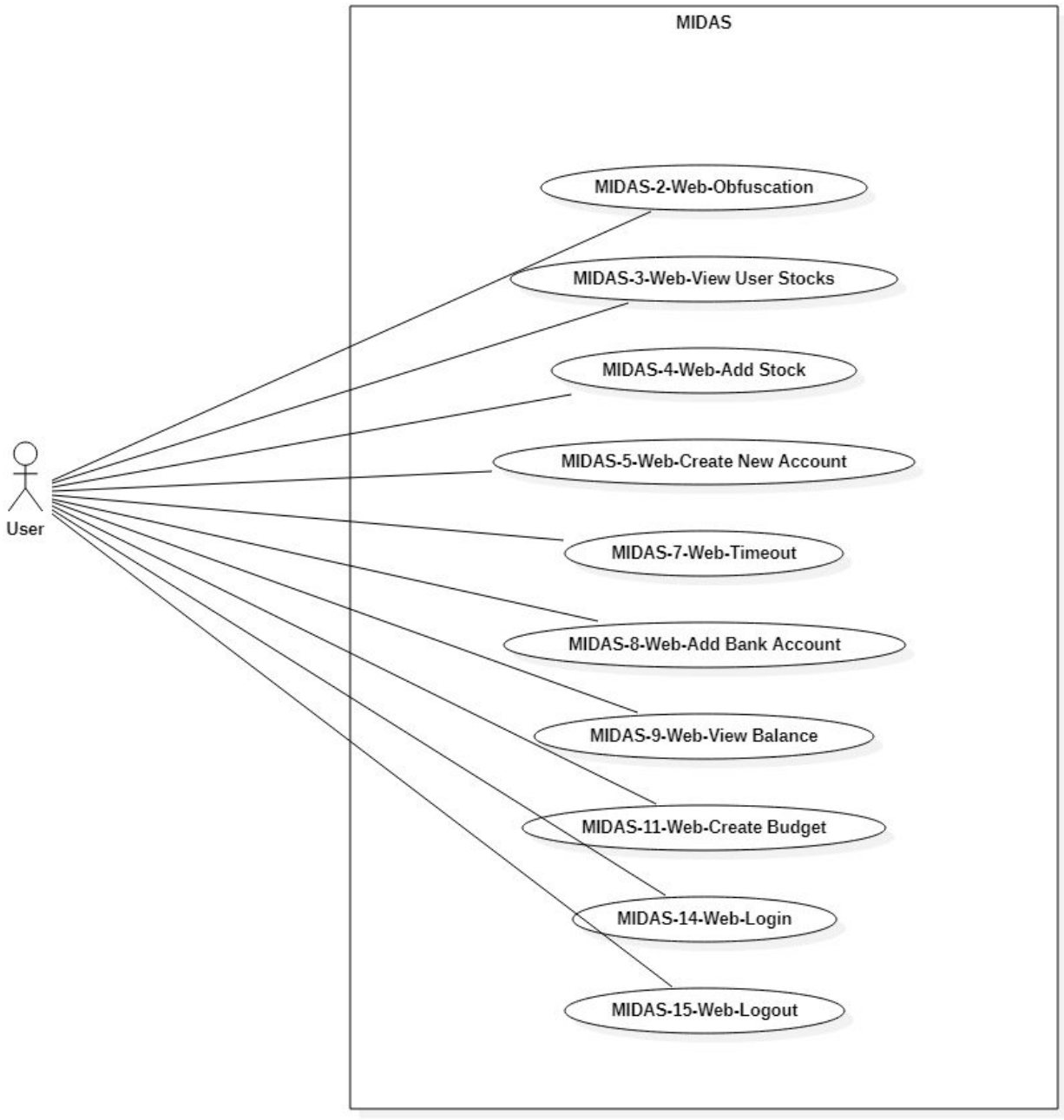


Figure 4.3.c MIDAS Web Application use case diagram containing the *Web Application* use cases

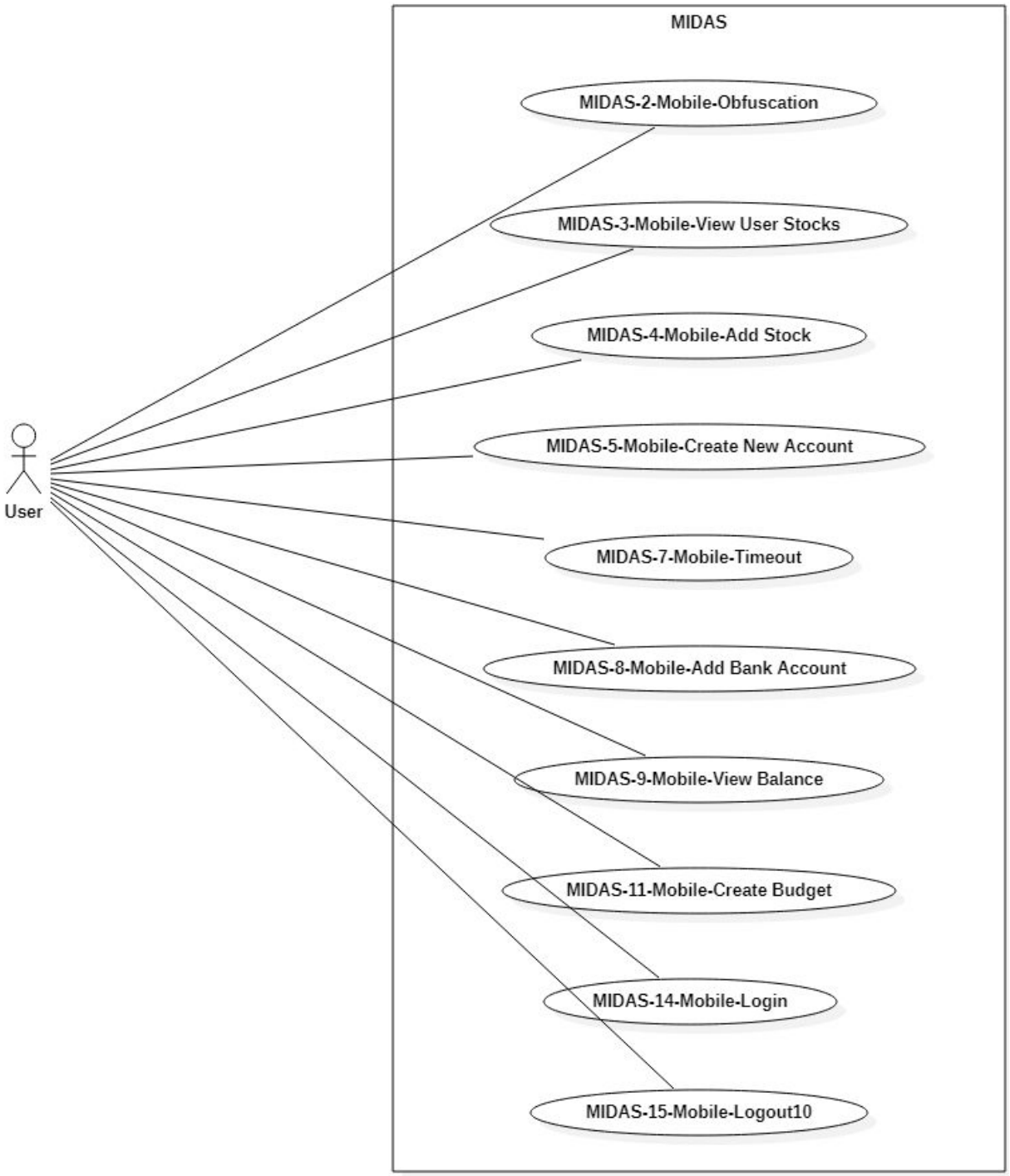


Figure 4.3.d MIDAS Mobile Application use case diagram containing the *Mobile Application* use cases

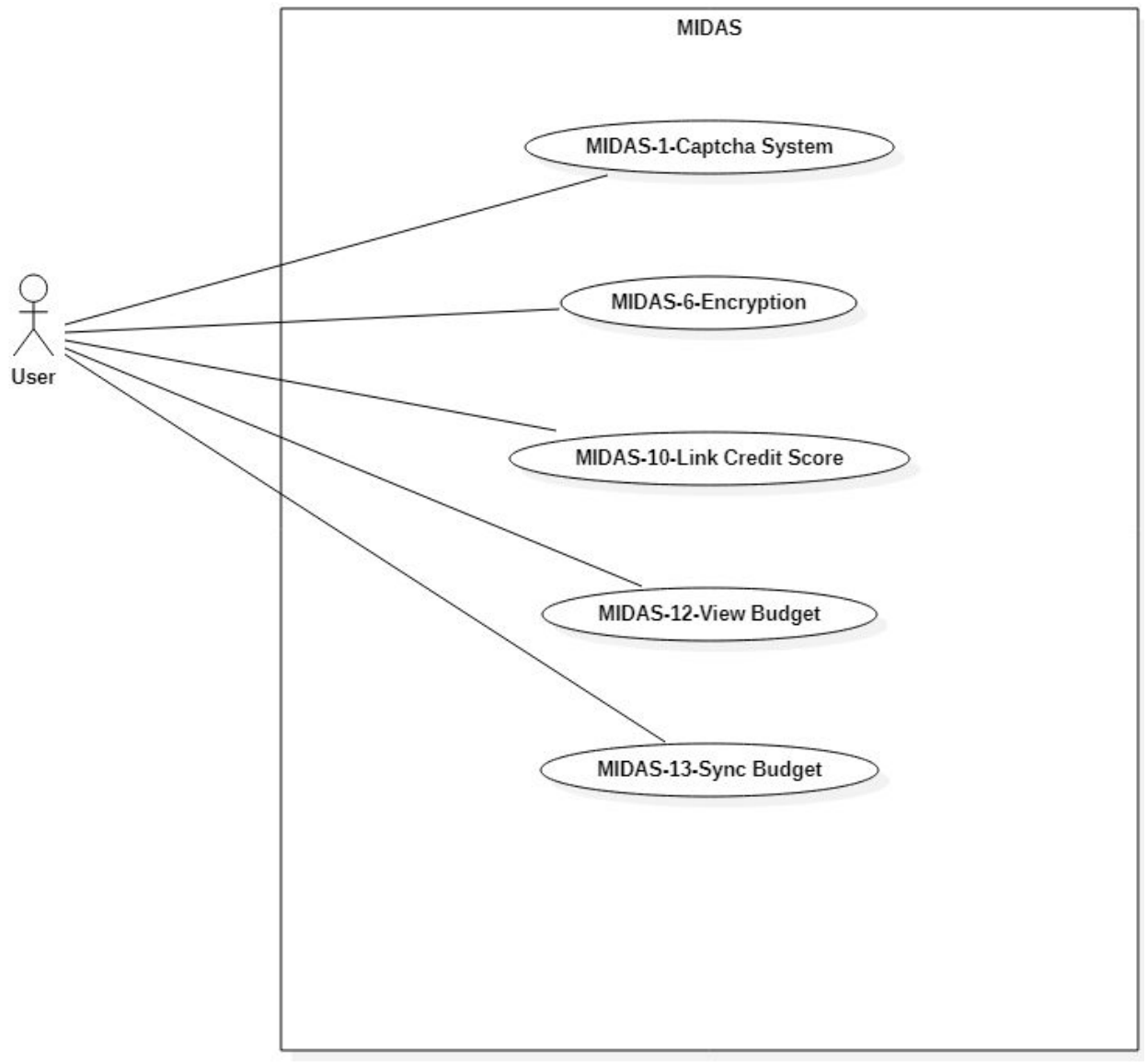


Figure 4.3.e MIDAS not implemented use case diagram containing the not implemented use cases

5. Requirements analysis

This chapter describes the requirements analysis section of the project. It includes the analysis models for 10 use cases. Included in the analysis model portion there are scenarios and a class diagram for the project's use cases, also present are object diagrams, and sequence diagrams for ten use cases which are split into Web and Model applications.

5.1. Scenarios.

MIDAS-2-Obfuscation

John Doe inputs id: "john" and pw: "123" in the login page,

System obfuscates pw in the display to "*****",

MIDAS-3-View User Stocks

John Doe clicks on the view user stocks button in the dashboard.

System displays the view user stocks page with "FB"

MIDAS-4-Add Stock

John Doe clicks on the add stocks button in the dashboard.

System displays the add stocks page.

John Doe inputs name: "fb", and amount: "1"

John Doe clicks the submit button.

System stores the inputted values into the database.

MIDAS-5-Create New Account

John Doe clicks on the create new account button in the login page.

System displays the create new account page.

John Doe inputs id: “john01”, name: “John Doe”, pw: “123”

John Doe clicks the submit button

System stores the inputted values to the database.

MIDAS-7-Timeout

John Doe is signed into the system.

John Doe is idle for 10 minutes.

System updates John Doe’s account to be set to Offline.

System logs John Doe out

System displays the login page.

MIDAS-8-Add_Bank

John Doe is in the Dashboard

John Doe clicks the view balance button.

System displays the view balance page.

System loads balance information from the database.

John Doe clicks on the add bank account button.

System displays the add bank account page.

John Doe inputs name: “chase”, routingnum: 067014822, accountnum: 12245679

System stores inputted items in the database.

MIDAS-9-View_Balance

John Doe is in the Dashboard

John Doe clicks the view balance button.

System displays the view balance page.

System loads balance information from the database.

MIDAS-11-Create_Budget

John Doe is in the dashboard.

John Doe clicks the add budget button.

System displays the add budget page.

John Doe inputs name: “Groceries”, amount: “165”

John Doe clicks the submit button.

System stores the inputted values into the database.

MIDAS-14-Login

John Doe is in the login screen.

John Doe inputs id: “john01”, pw: “123”

John Doe clicks the submit button.

System loads John Does account from the database.

System authenticates stored password with inputted password.

System updates John Does user profile to be online

System displays the dashboard.

MIDAS-15-Logout

John Doe clicks the logout button.

System updates John Does profile to be offline.

System displays the dashboard.

5.2. Static model

5.2.1. Object Diagrams

Figure 5.2.1.a MIDAS-2-Web-Obfuscation Object

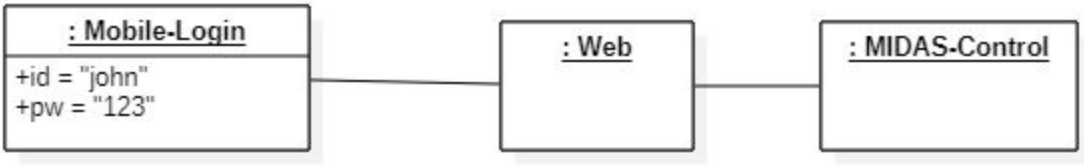


Figure 5.2.1.b MIDAS-2-Mobile-Obfuscation Object

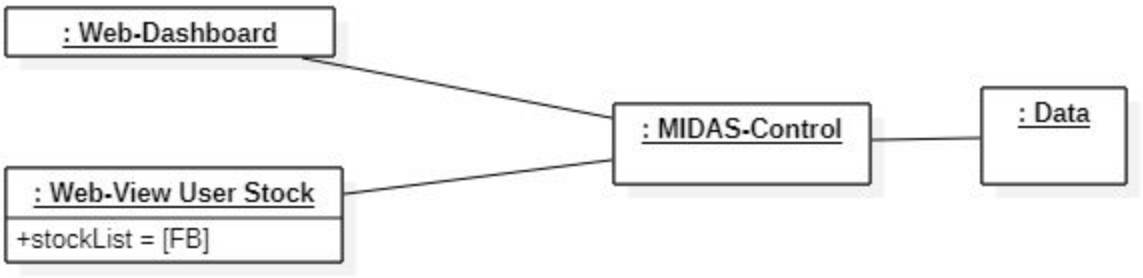


Figure 5.2.1.c MIDAS-3-Web-View User Stocks Object

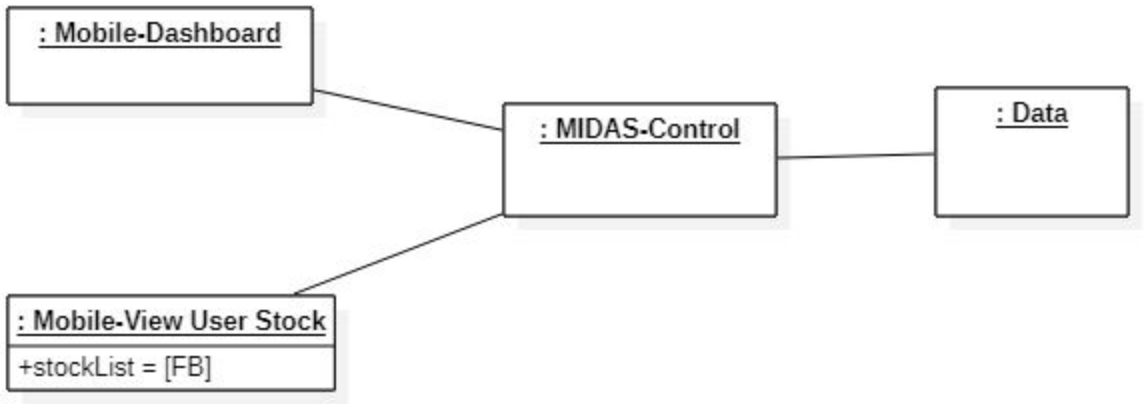


Figure 5.2.1.d MIDAS-3-Mobile-View User Stocks Object

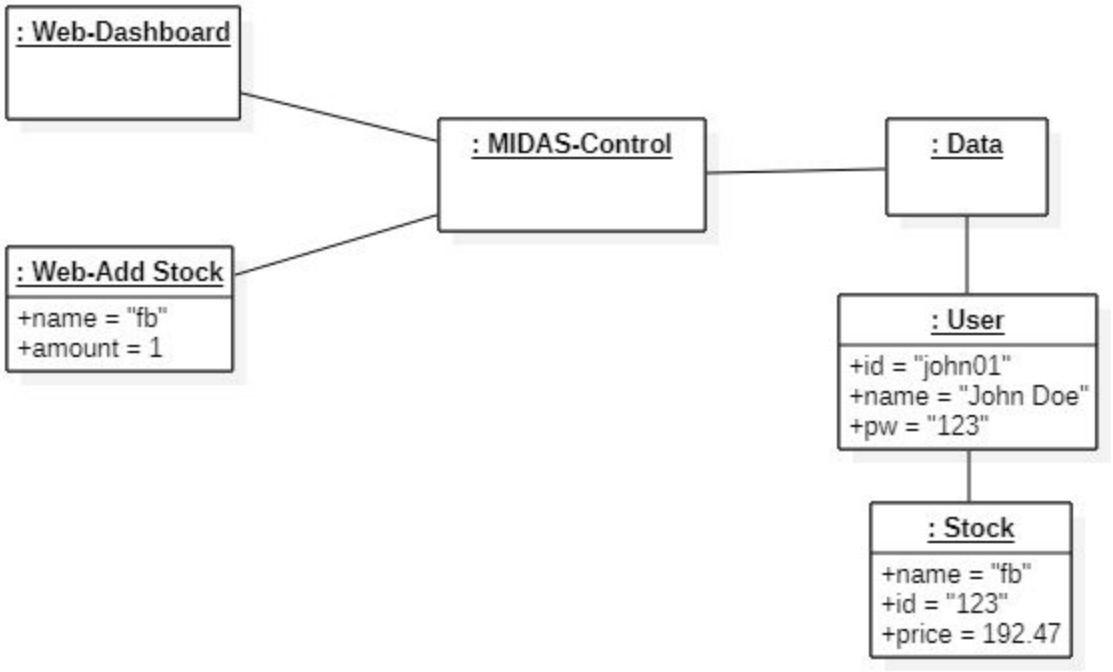


Figure 5.2.1.e MIDAS-4-Web-Add Stock Object

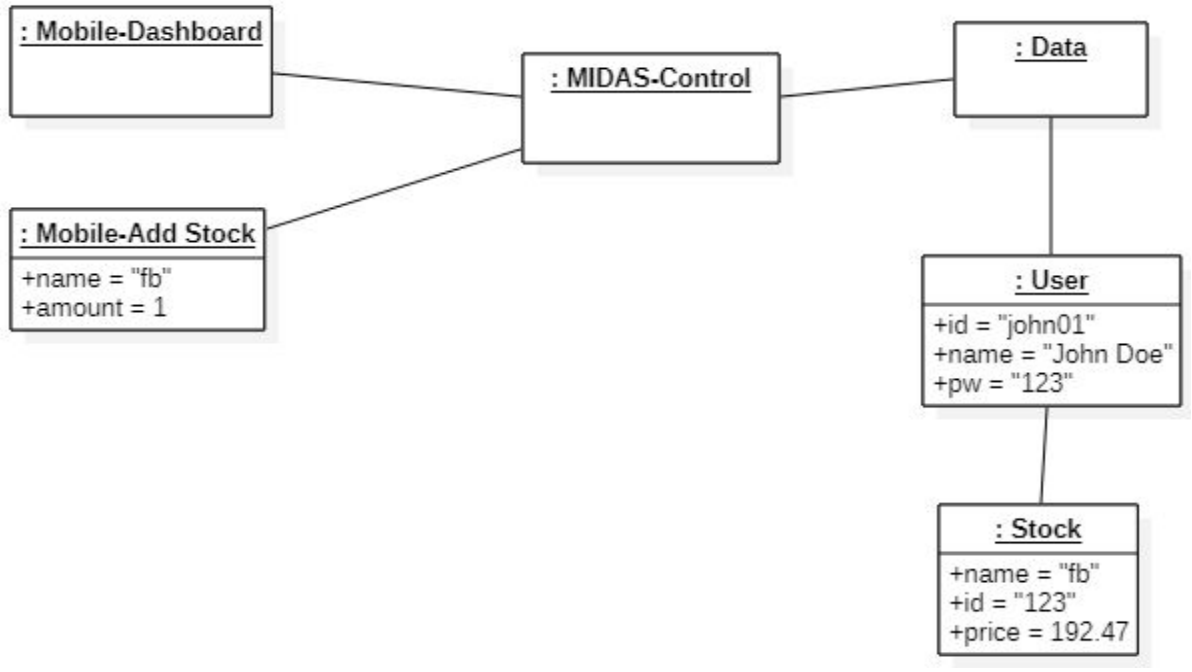


Figure 5.2.1.f MIDAS-4-Mobile-Add Stock Object

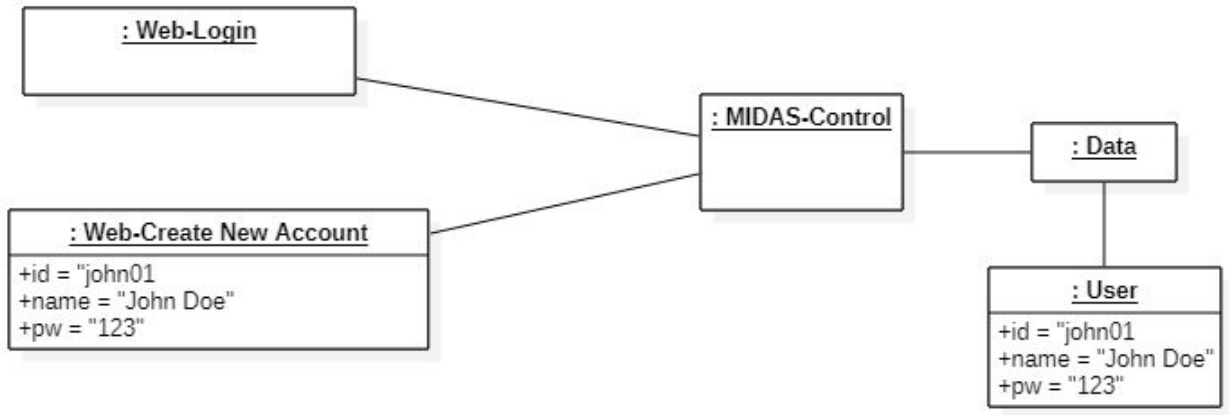


Figure 5.2.1.g MIDAS-5-Web-Create New Account Object

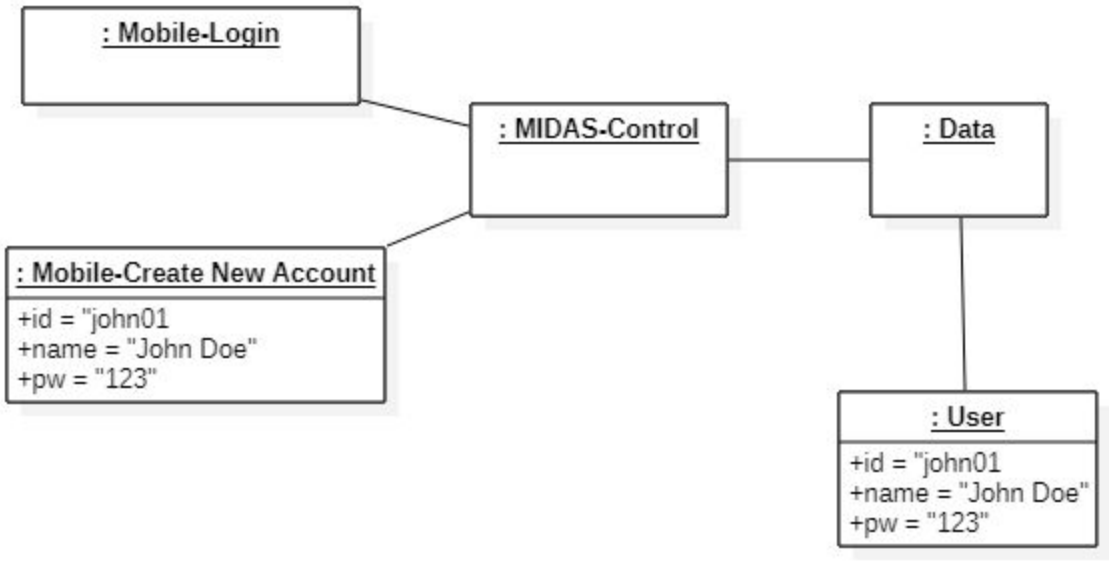


Figure 5.2.1.h MIDAS-5-Mobile-Create New Account Object

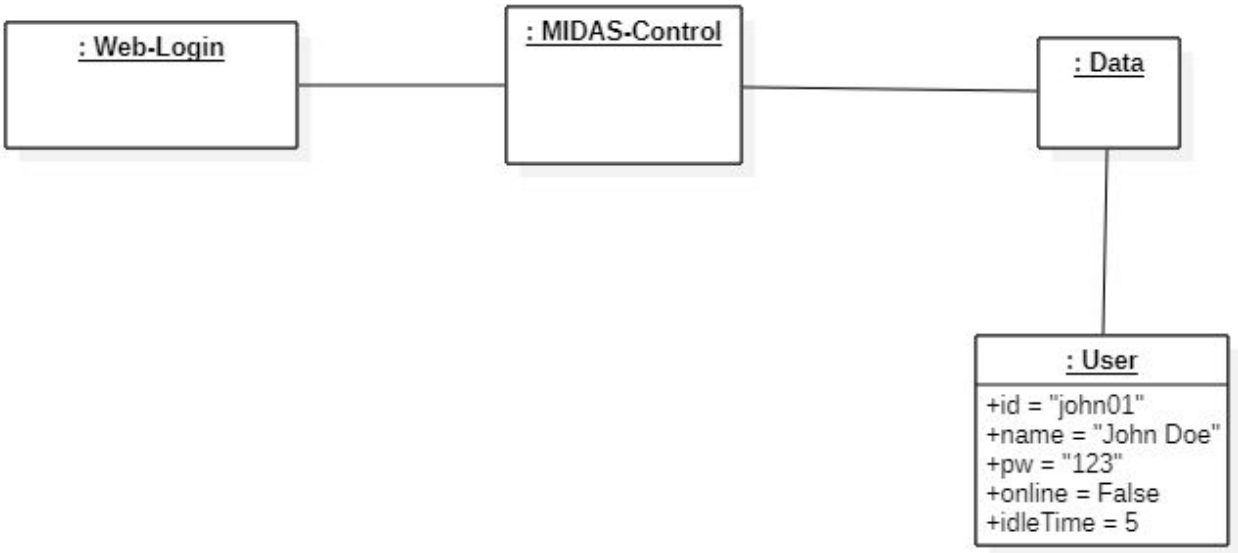


Figure 5.2.1.i MIDAS-7-Web-Timeout Object

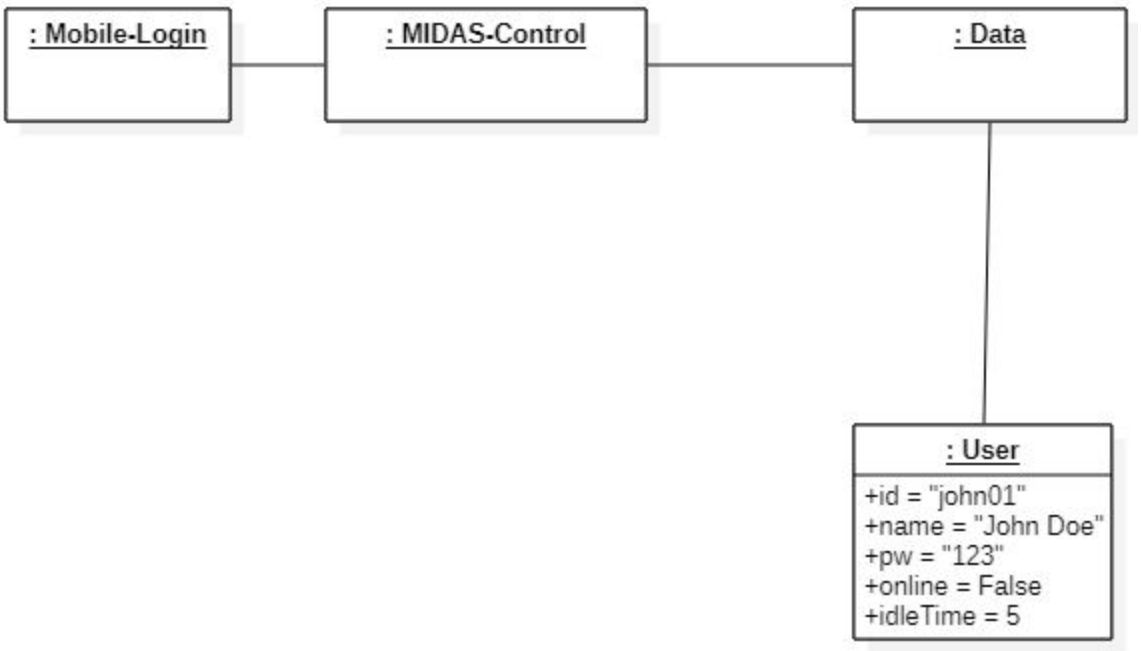


Figure 5.2.1.j MIDAS-7-Mobile-Timeout Object

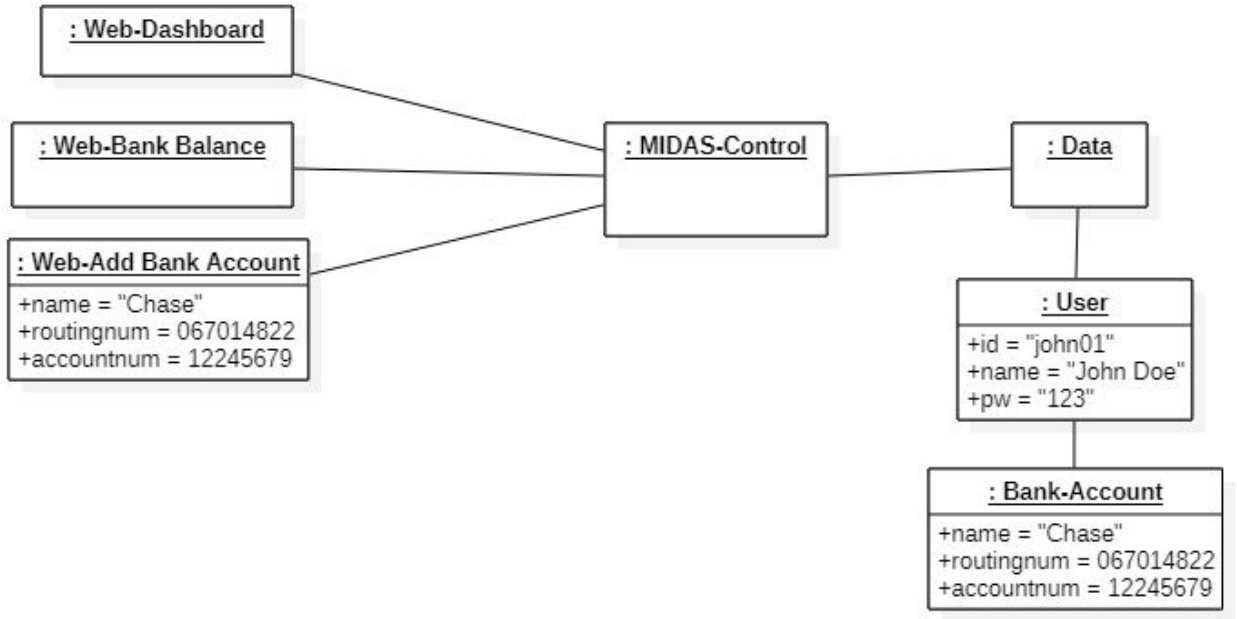


Figure 5.2.1.k MIDAS-8-Web-Add Bank Account Object

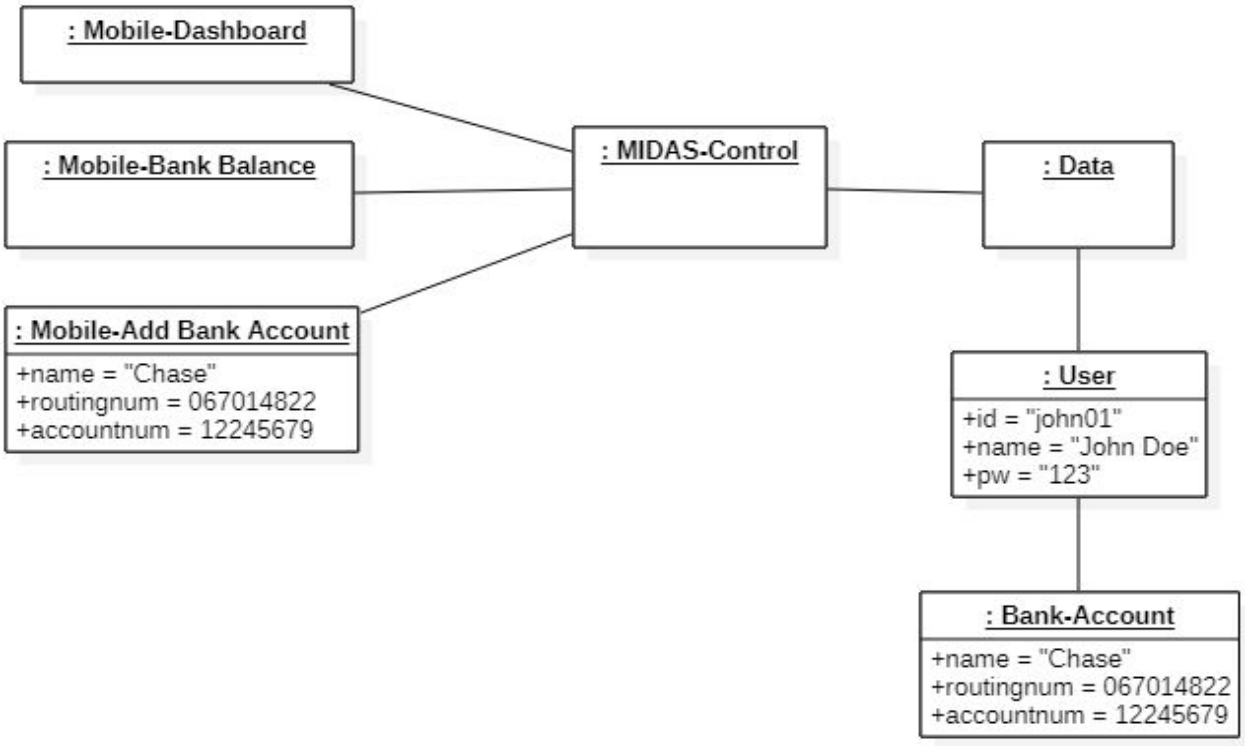


Figure 5.2.1.l MIDAS-8-Mobile-Add Bank Account Object

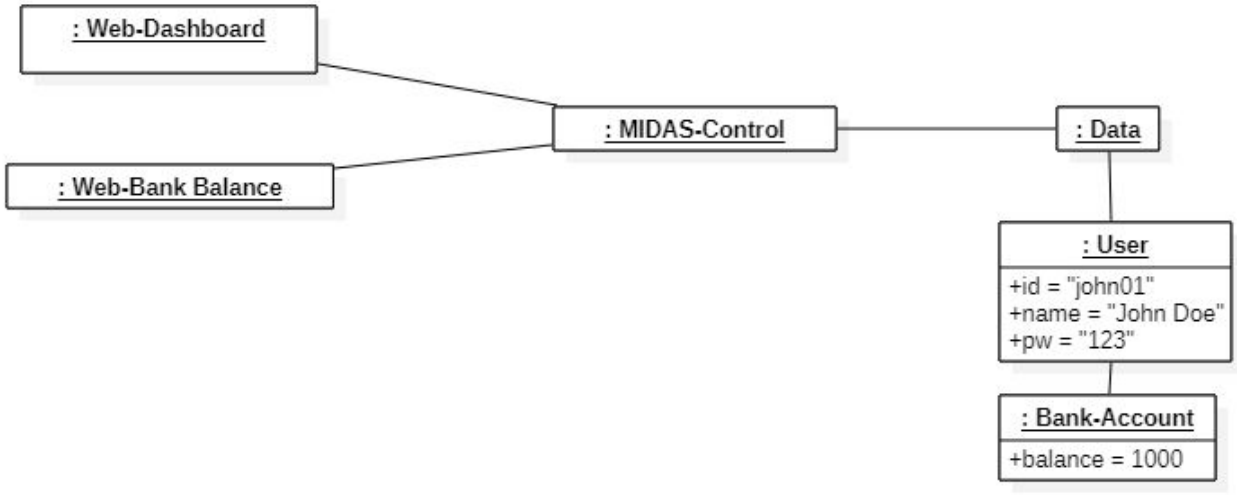


Figure 5.2.1.m MIDAS-9-Web-View Balance Object

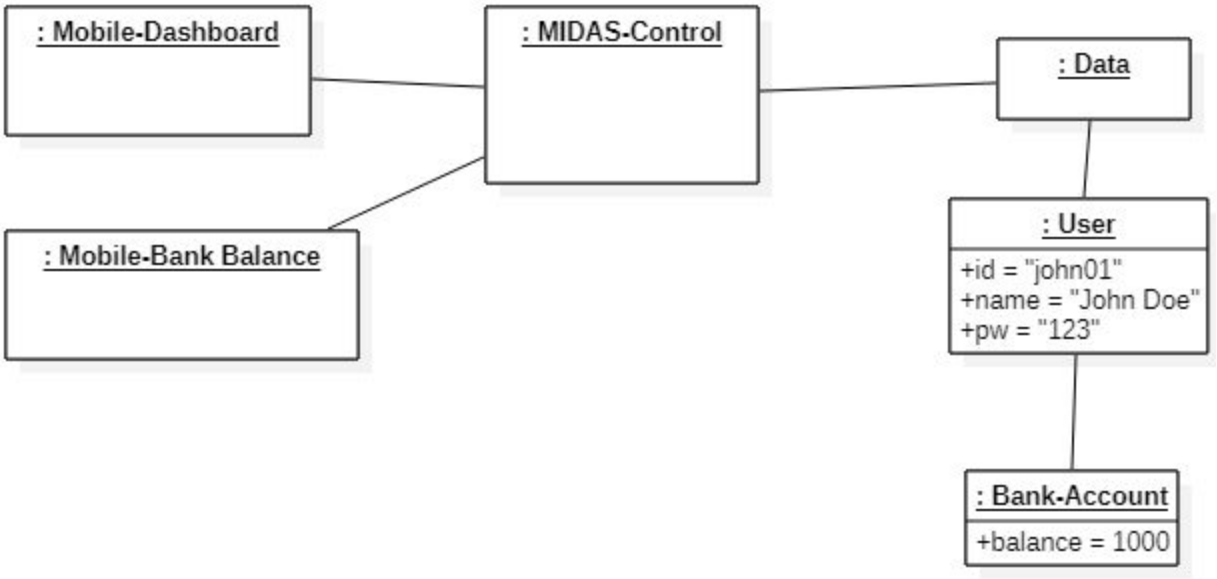


Figure 5.2.1.n MIDAS-9-Mobile-View Balance Object

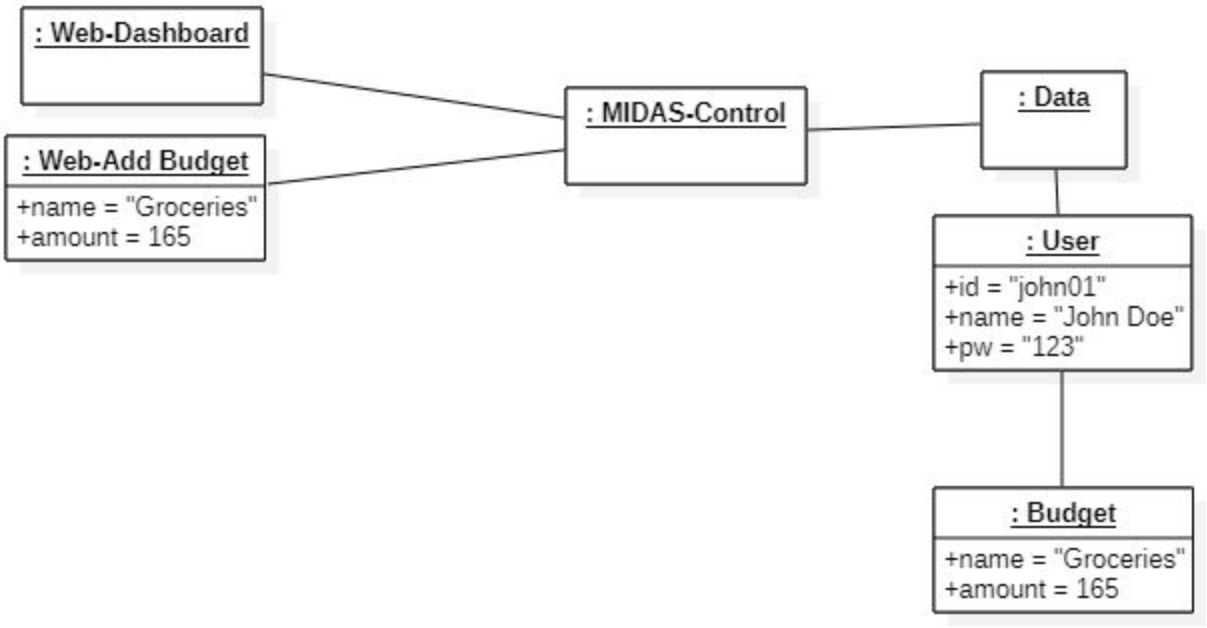


Figure 5.2.1.o MIDAS-11-Web-Create Budget Object

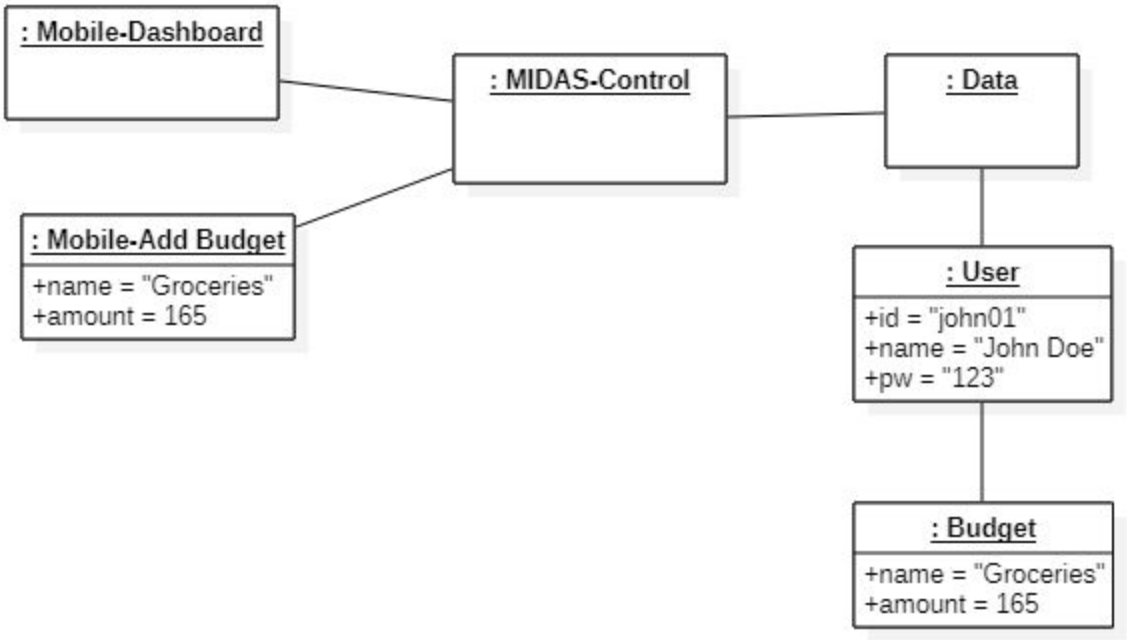


Figure 5.2.1.p MIDAS-11-Mobile-Create Budget Object

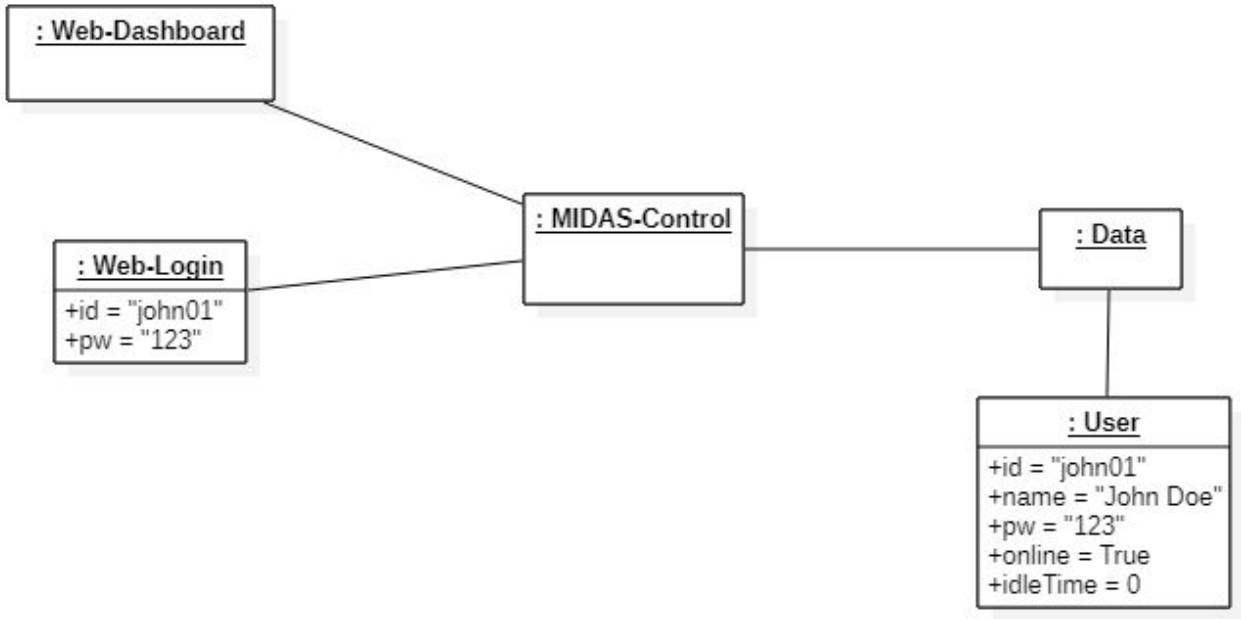


Figure 5.2.1.q MIDAS-14-Web-Login Object

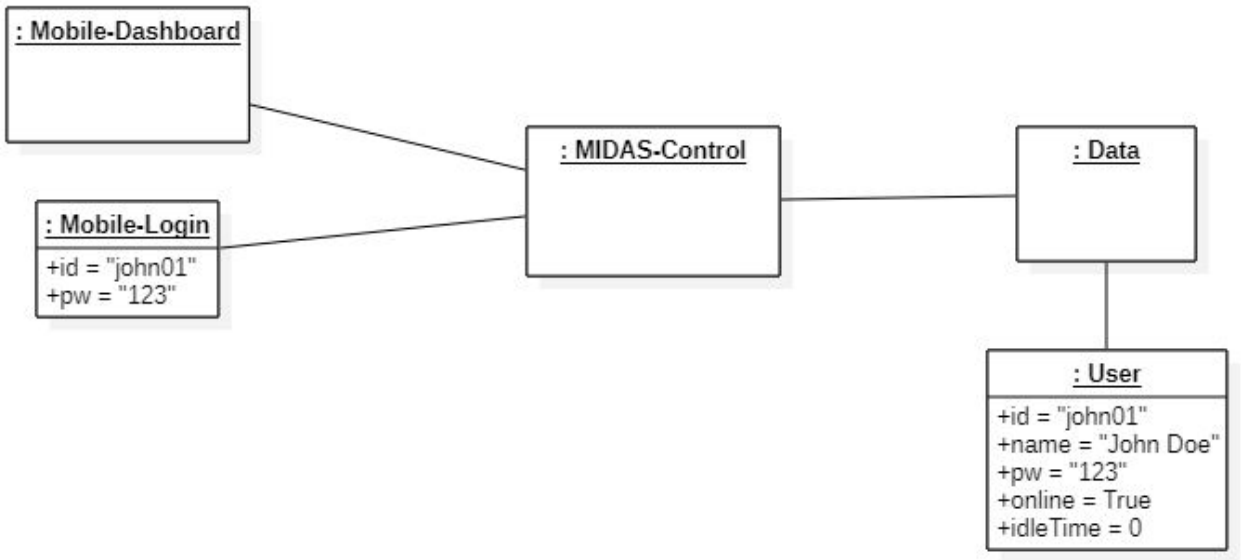


Figure 5.2.1.r MIDAS-14-Mobile-Login Object

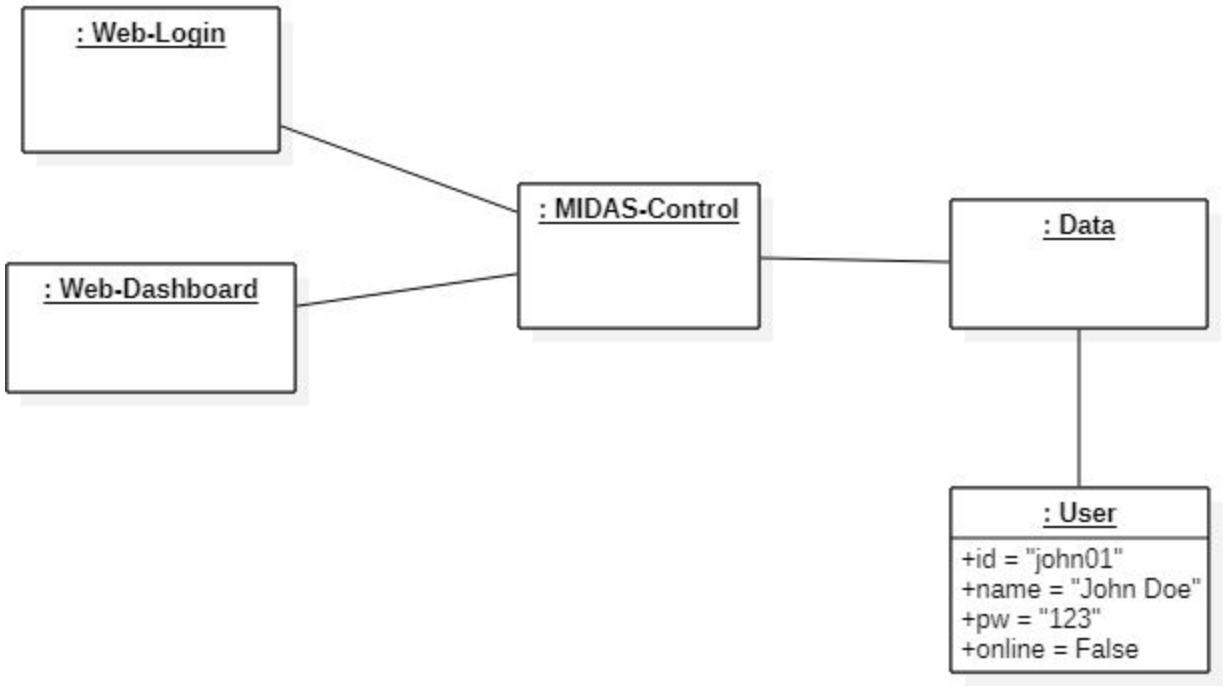


Figure 5.2.1.s MIDAS-15-Web-Logout Object

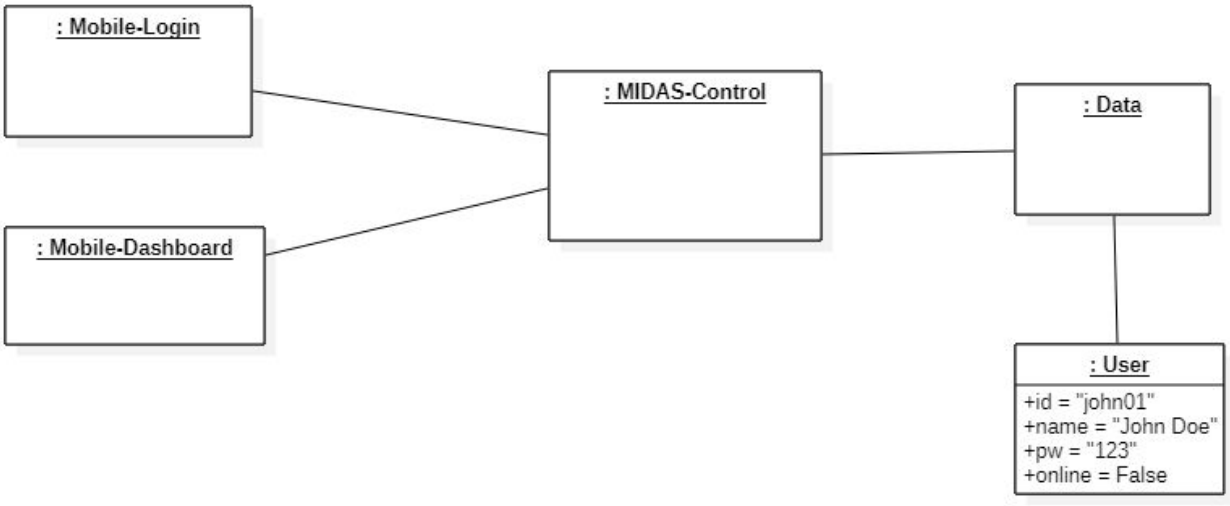


Figure 5.2.1.t MIDAS-15-Mobile-Logout Object

5.2.2. Class diagram

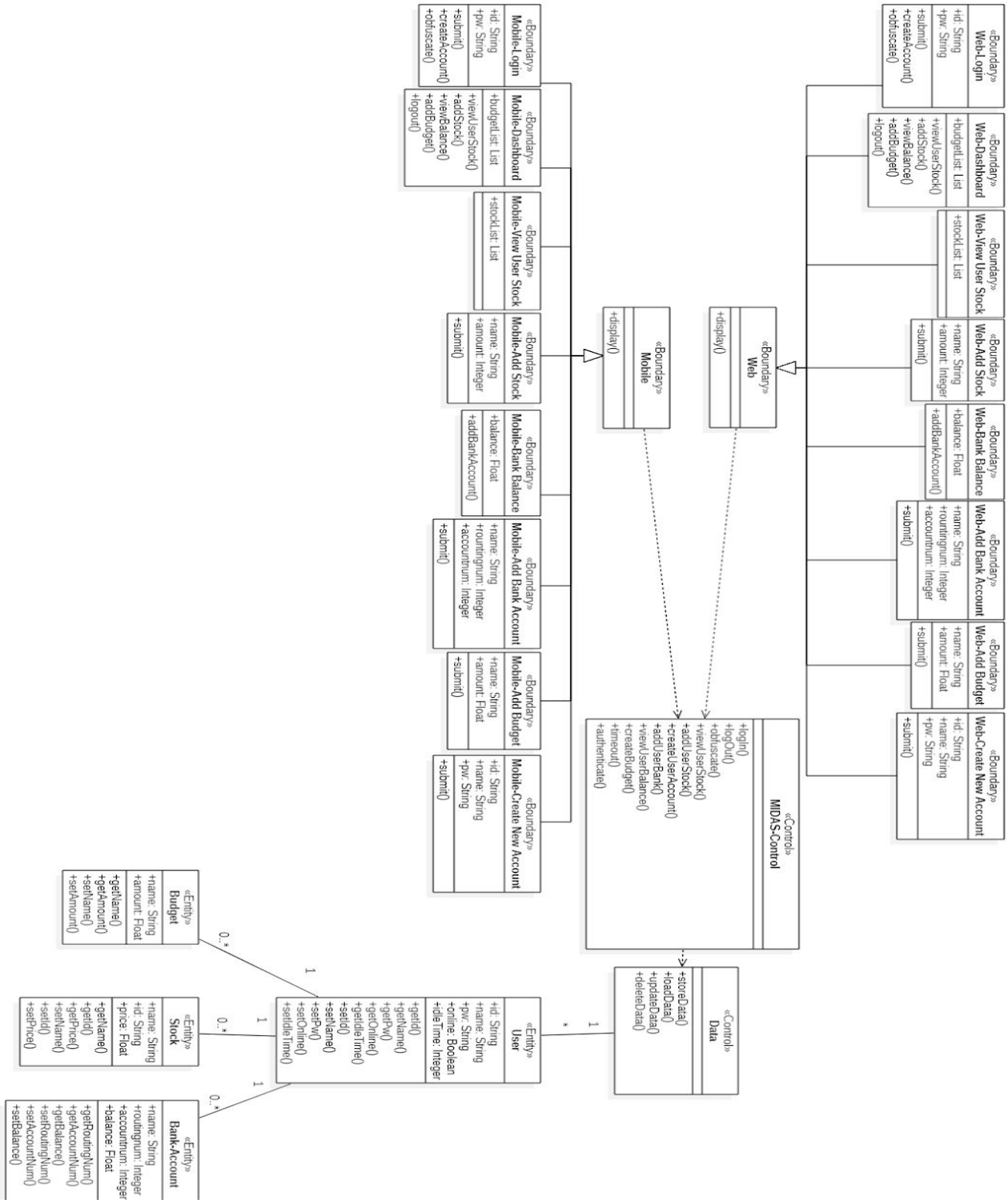


Figure 5.2.2.a MIDAS class diagram

5.3. Dynamic model

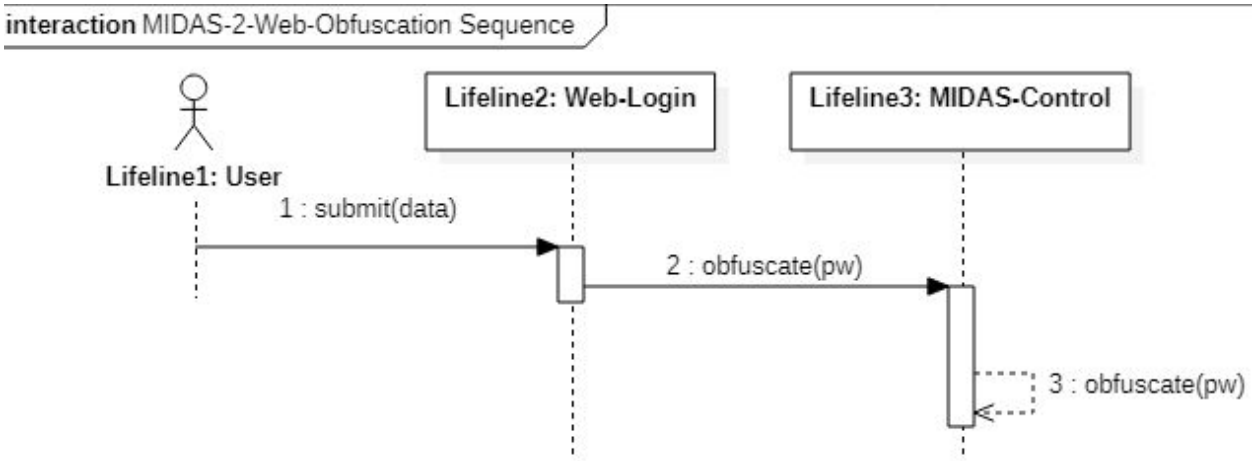


Figure 5.3.1.a MIDAS-2-Web-Obfuscation Sequence

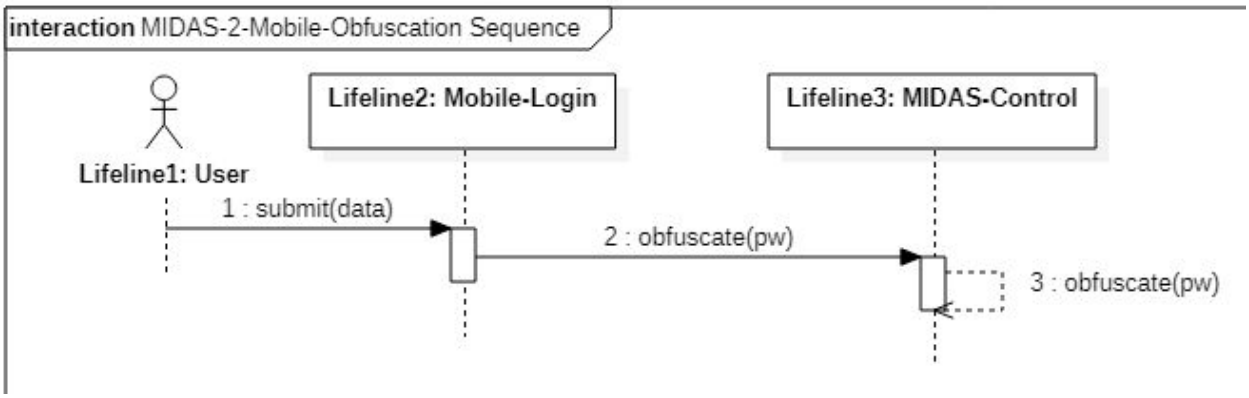


Figure 5.3.1.b MIDAS-2-Mobile-Obfuscation Sequence

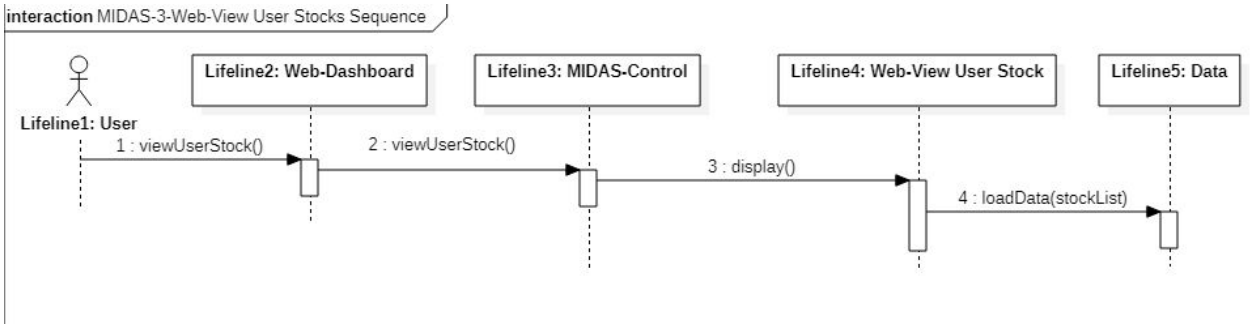


Figure 5.3.1.c MIDAS-3-Web-View User Stocks Sequence

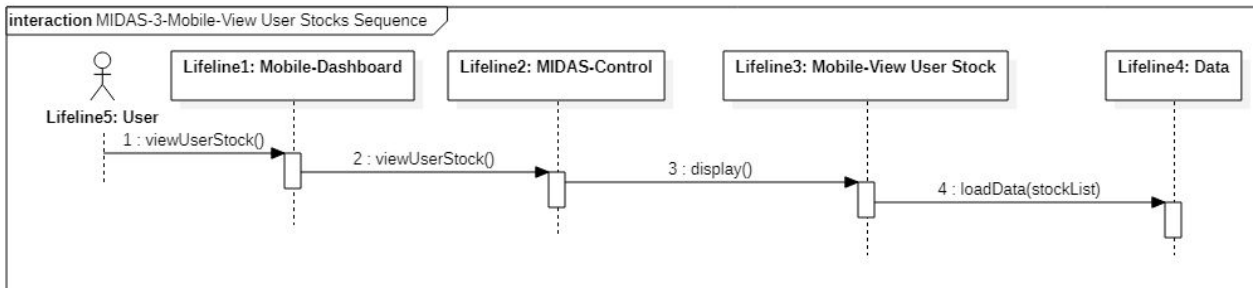


Figure 5.3.1.d MIDAS-3-Mobile-View User Stocks Sequence

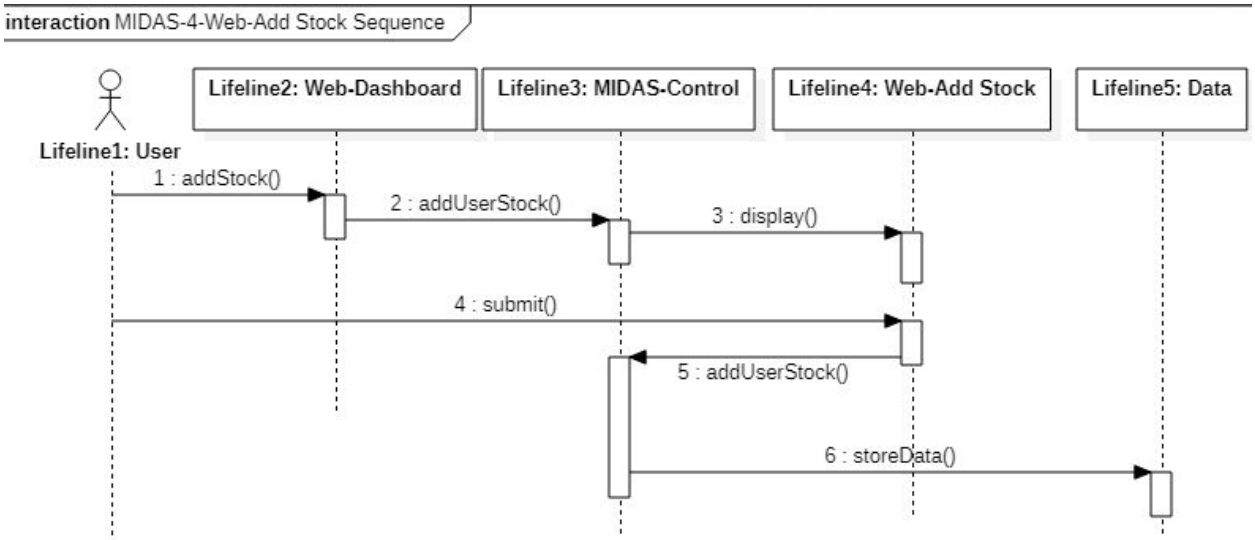


Figure 5.3.1.e MIDAS-4-Web-Add Stock Sequence

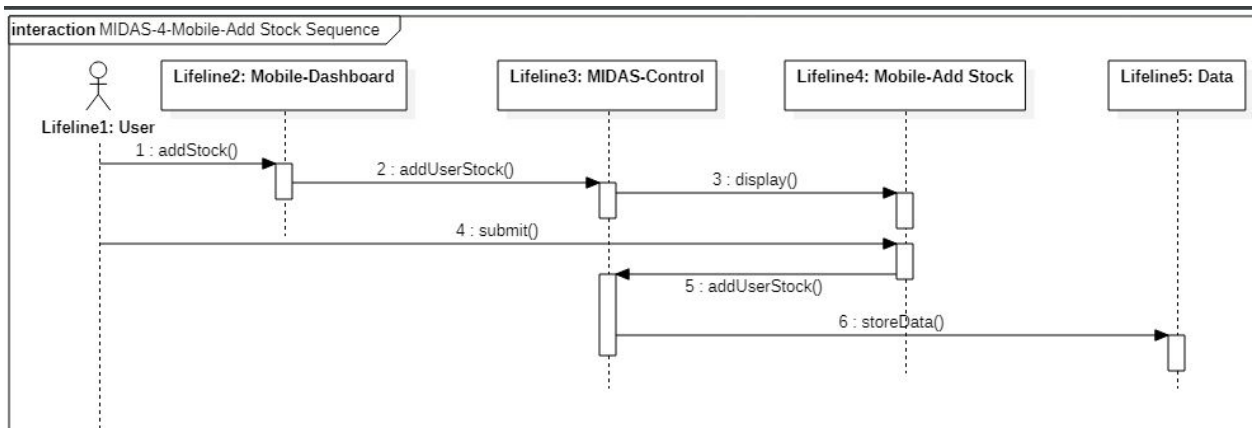


Figure 5.3.1.f MIDAS-4-Mobile-Add Stock Sequence

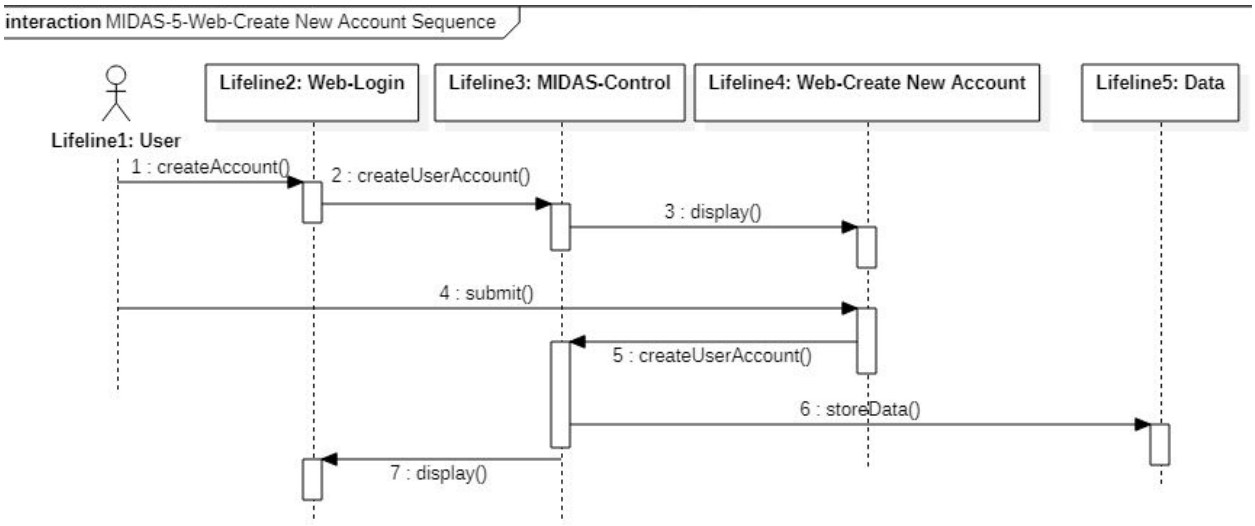


Figure 5.3.1.g MIDAS-5-Web-Create New Account Sequence

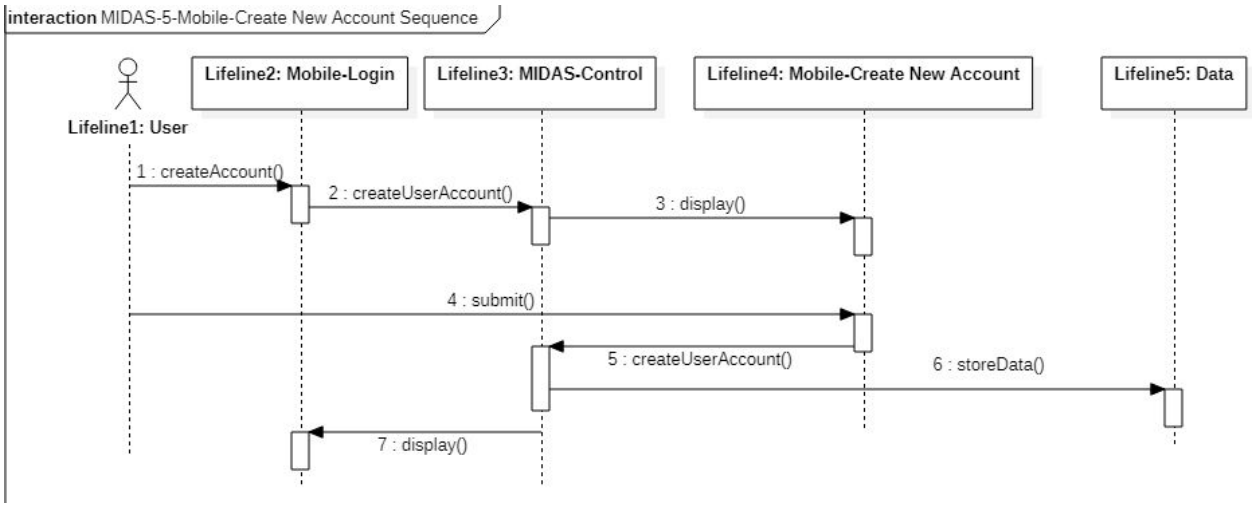


Figure 5.3.1.h MIDAS-5-Mobile-Create New Account Sequence

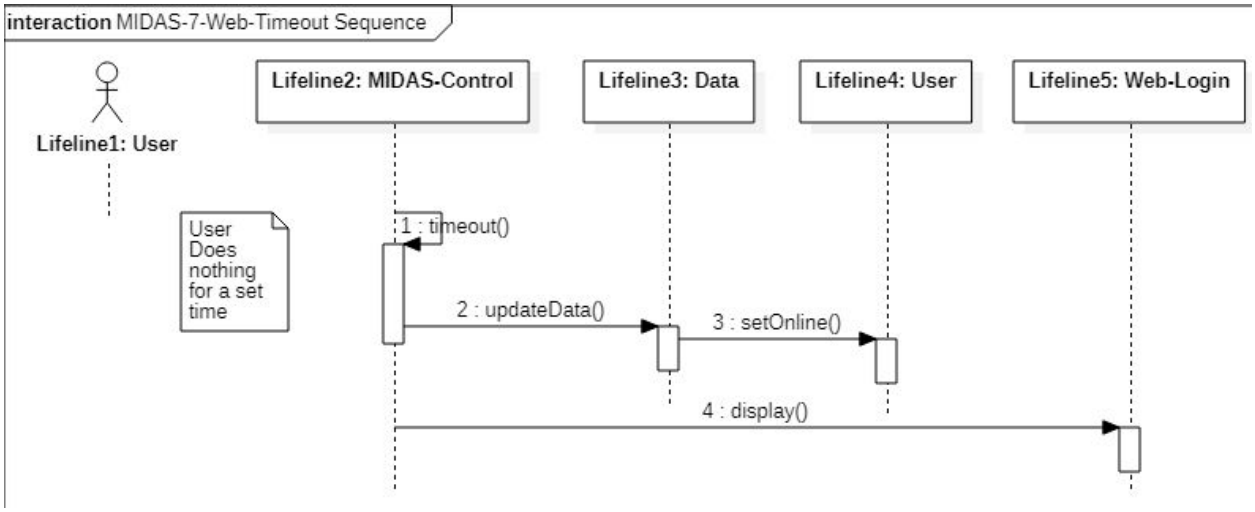


Figure 5.3.1.i MIDAS-7-Web-Timeout Sequence

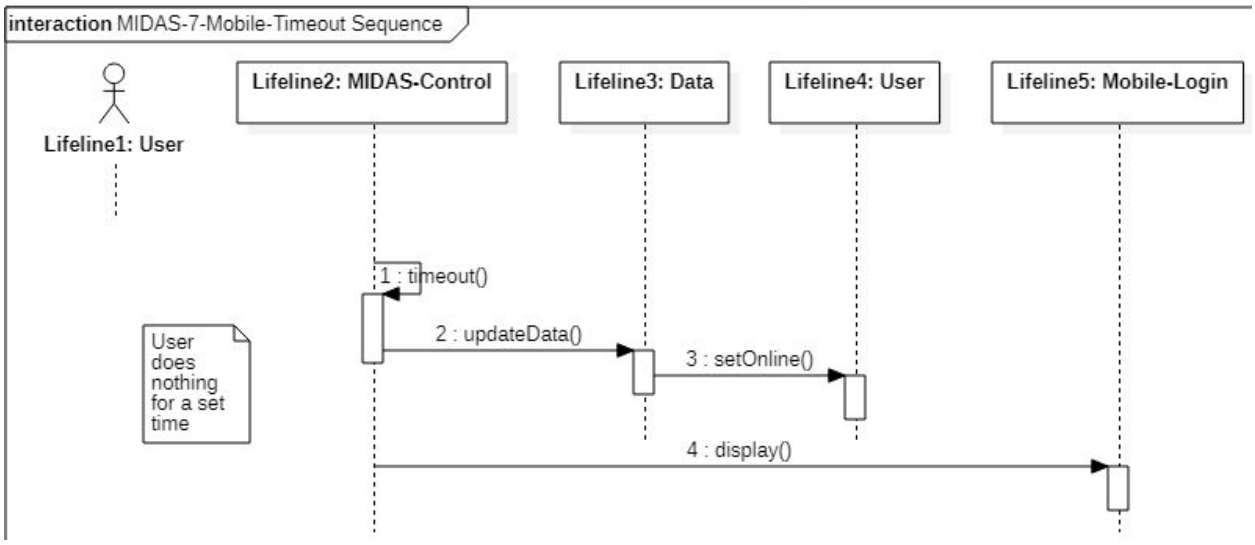


Figure 5.3.1.j MIDAS-7-Mobile-Timeout Sequence

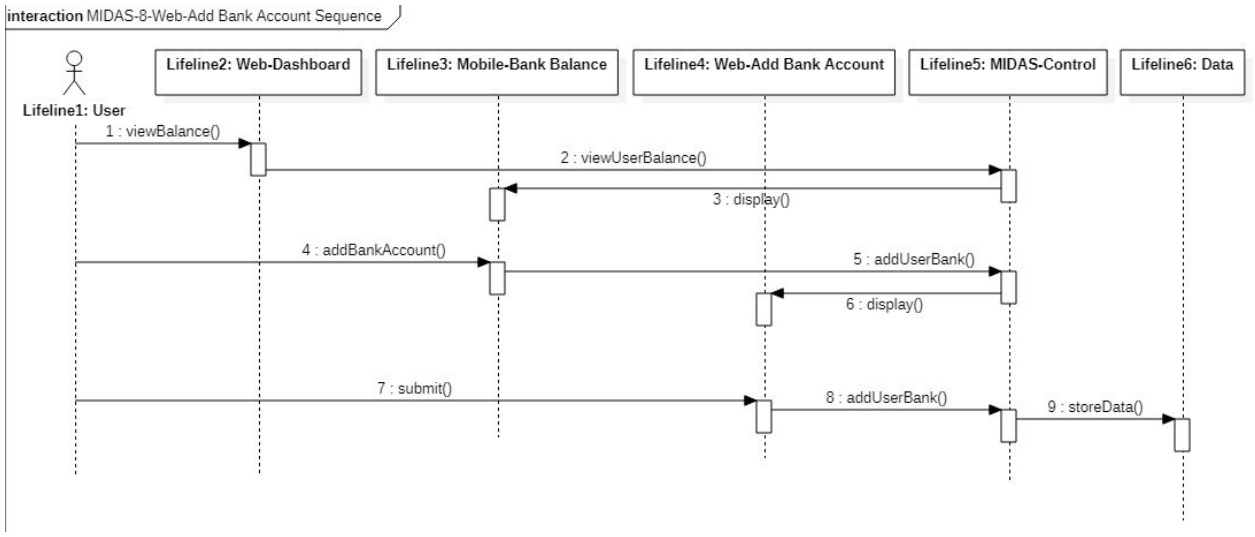


Figure 5.3.1.k MIDAS-8-Web-Add Bank Account Sequence

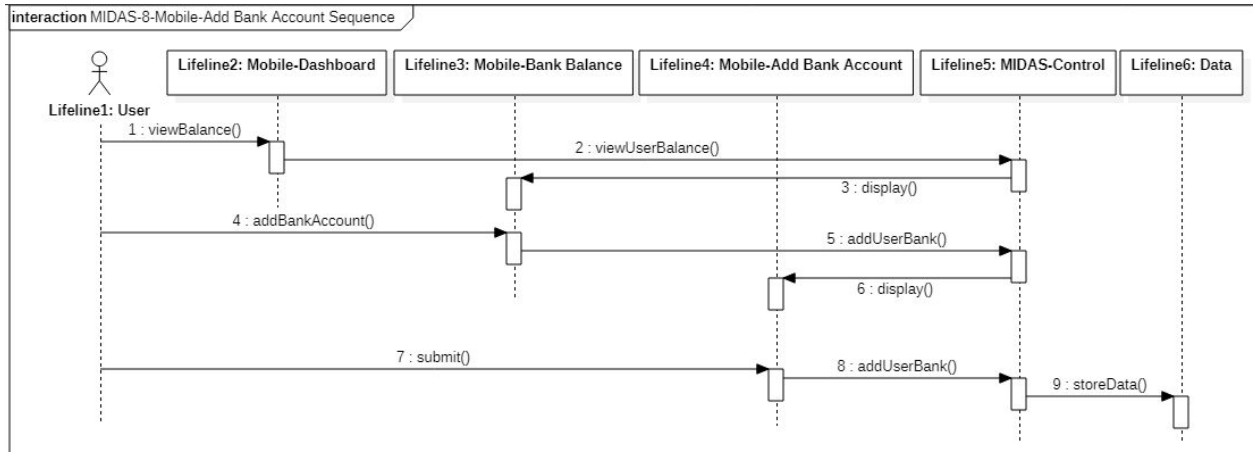


Figure 5.3.1.l MIDAS-8-Mobile-Add Bank Account Sequence

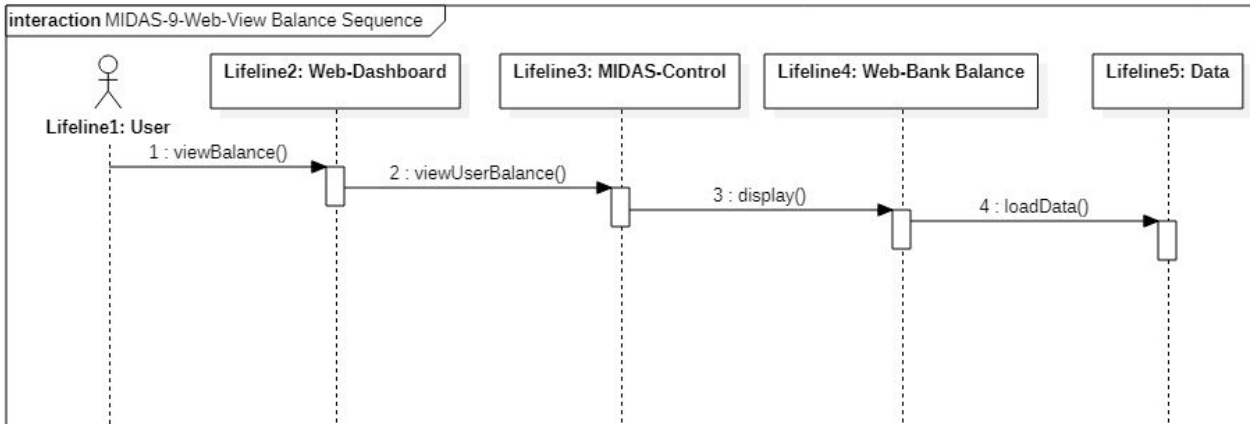


Figure 5.3.1.m MIDAS-9-Web-View Balance Sequence

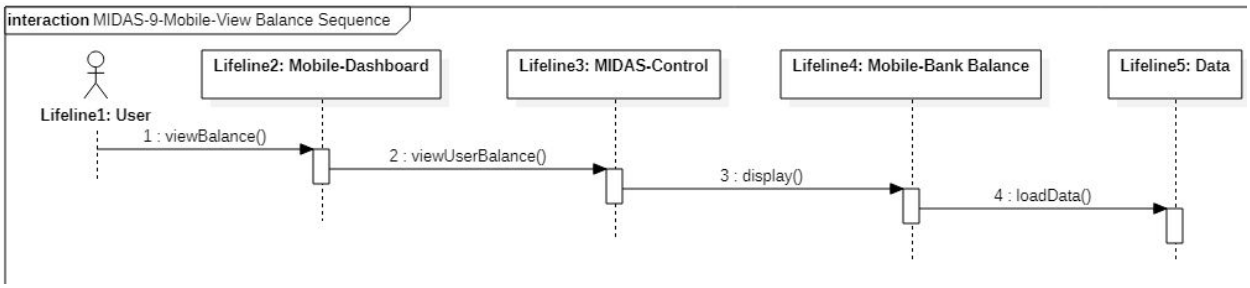


Figure 5.3.1.n MIDAS-9-Mobile-View Balance Sequence

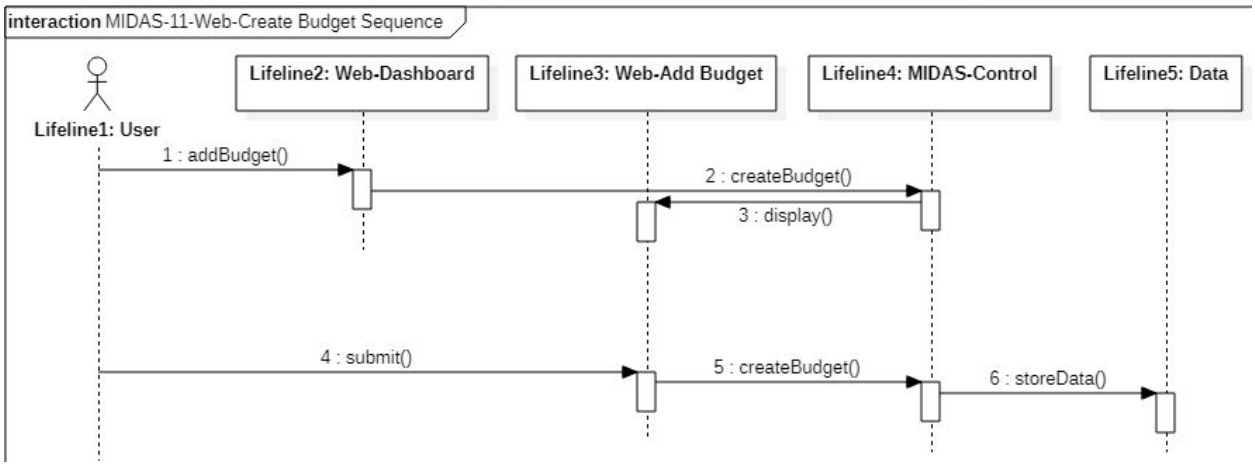


Figure 5.3.1.o MIDAS-11-Web-Create Budget Sequence

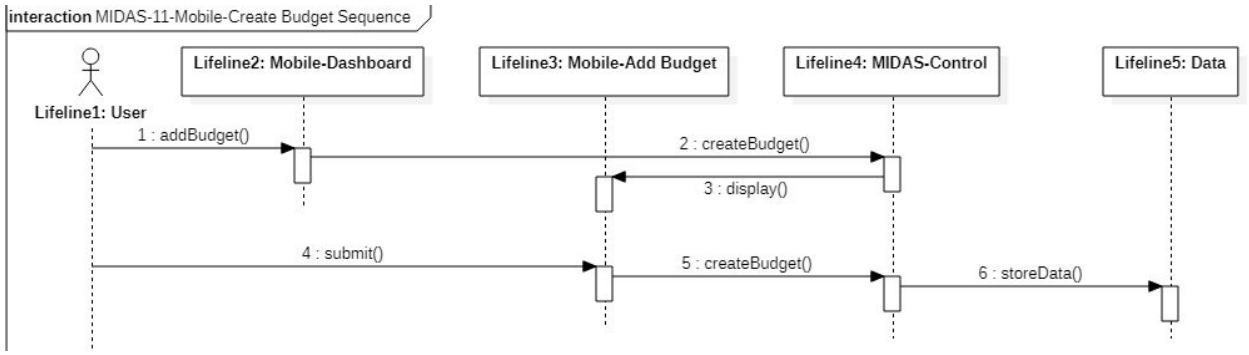


Figure 5.3.1.p MIDAS-11-Mobile-Create Budget Sequence

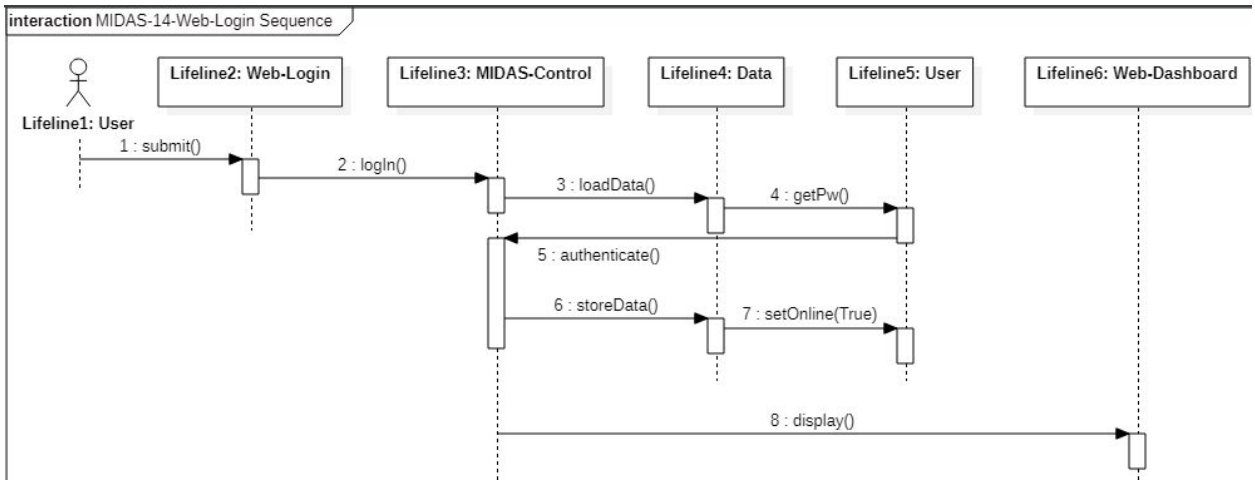


Figure 5.3.1.q MIDAS-14-Web-Login Sequence

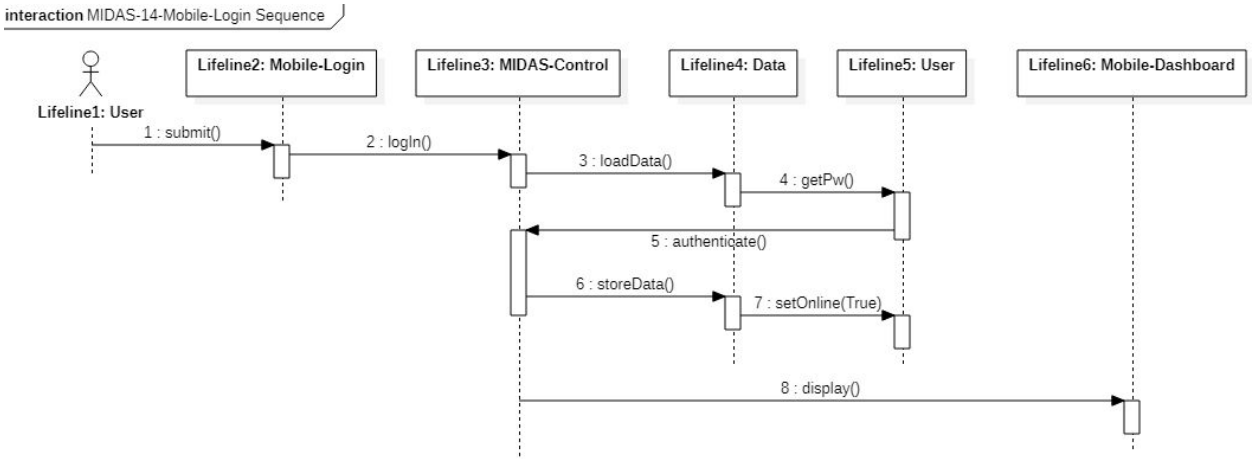


Figure 5.3.1.r MIDAS-14-Mobile-Login Sequence

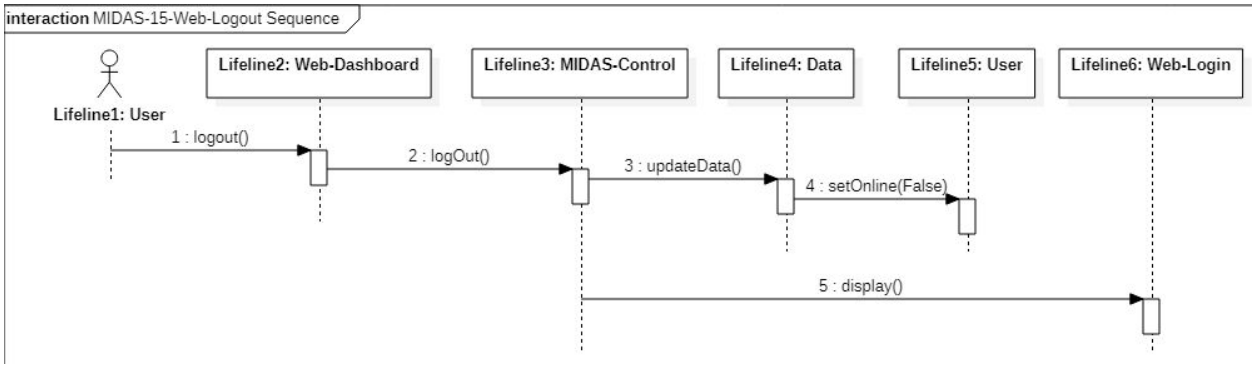


Figure 5.3.1.s MIDAS-15-Web-Logout Sequence

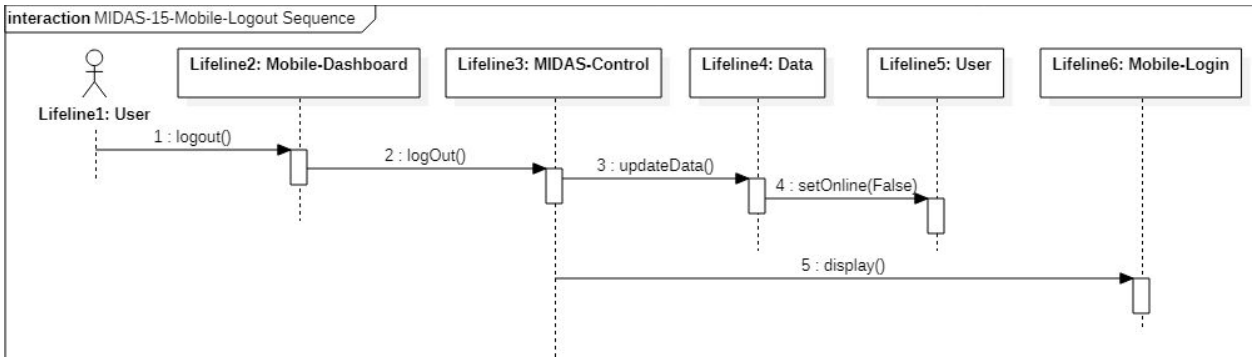
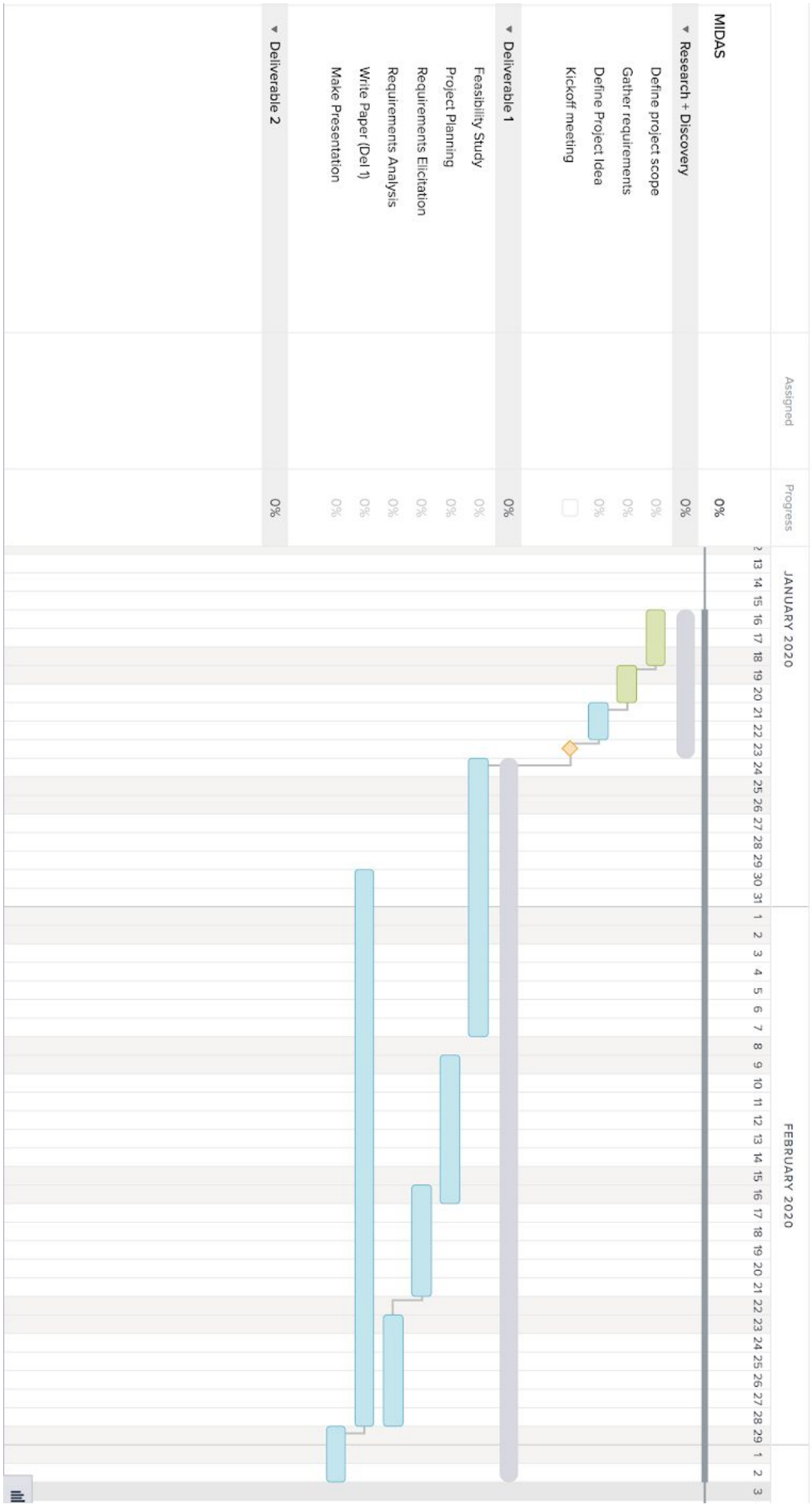


Figure 5.3.1.t MIDAS-15-Mobile-Logout Sequence

6. Appendix

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


MIDAS
Assigned
Progress
JANUARY 2020
FEBRUARY 2020
Research + Discovery
0%
Define project scope
0%
Gather requirements
0%
Define Project Idea
0%
Kickoff meeting
Deliverable 1
0%
Feasibility Study
0%
Project Planning
0%
Requirements Elicitation
0%
Requirements Analysis
0%
Write Paper (Del 1)
0%
Make Presentation
0%
Deliverable 2
0%

6.2. Appendix B - Feasibility Matrix

Feasibility Criteria	Weight	Candidate 1	Candidate 2	Candidate 3
Operational Feasibility **Functionality.** A description of to what degree the candidate system solves the problem. **Political.** A description of how well received this solution would be from a user's perspective.	**30%**	Fully supports user required functionality. Well received by a lot of users, although iPhone users may not utilize the application. **Score: 75**	Same as candidate 1. Covers most of the potential users that would be content with these platforms supported. **Score: 95**	Same as candidate 1. Well received by almost all users, although desktop application may be superfluous. **Score: 90**

Technical Feasibility **Technology.** An assessment of the computer technology needed to support this candidate. **Expertise.** An assessment of the technical expertise required to implement the candidate.	**30%**	All hardware and software required for project can be acquired and is mature. Required expertise is possessed. **Score: 95**	Same as candidate 1. Training required to gain expertise in iOS development. **Score: 90**	Same as candidate 1. Training required to gain expertise in iOS and Windows development. **Score: 85**
Economic Feasibility Cost to develop:	**30%**	Approximately $34250. **Score: 95**	Approximately $42225. **Score: 85**	Approximately $48620. **Score: 80**
Schedule Feasibility An assessment of how long the solution will take to design	**10%**	3 months **Score: 100**	3 months **Score: 100**	3 months **Score: 100**
Ranking	**100%**	**89.5**	**91**	**86.5**

6.3. Appendix C - Cost Matrix

Cost Matrix for Candidate 1

Personnel:

Quantity	Resource	Cost
1	Project Manager (160 hours/ea $45.00/hr)	$7200
1	Business Analyst (100 hours/ea $30.00/hr)	$3000
1	Systems Analyst (50 hours/ea $25.00/hr	$1250
1	UML Engineer (70 hours/ea $30.00/hr)	$2100
1	Quality Analyst (50 hours/ea $25.00/hr)	$1250
1	Technical Writer (25 hours/ea $25.00/hr)	$625
1	Database Specialist (50 hours/ea $35.00/hr)	$1750
1	System Architect (50 hours/ea $35.00/hr)	$1750
2	UI Designer (30 hours/ea $30.00/hr)	$1800
3	Programmer (70 hours/ea $25.00/hr)	$5250

Expenses:

1	Google Developer Registration	$25

New Hardware and Software:

1	Development server (HPE ProLiant DL380)	$3250
1	MySQL Enterprise Database	$5000

Total development costs: **$34250**

Cost Matrix for Candidate 2

Personnel:

Quantity	Resource	Cost
1	Project Manager (160 hours/ea $45.00/hr)	$7200
1	Business Analyst (100 hours/ea $30.00/hr)	$3000
1	Systems Analyst (50 hours/ea $25.00/hr	$1250
1	UML Engineer (70 hours/ea $30.00/hr)	$2100
1	Quality Analyst (50 hours/ea $25.00/hr)	$1250
1	Technical Writer (25 hours/ea $25.00/hr)	$625
1	Database Specialist (50 hours/ea $35.00/hr)	$1750
1	System Architect (50 hours/ea $35.00/hr)	$1750
2	UI Designer (30 hours/ea $30.00/hr)	$1800
3	Programmer (95 hours/ea $25.00/hr)	$7125

Expenses:

1	Google Developer Registration	$25
3	Swift and iOS development training ($2000/person)	$6000
1	Apple Developer Registration	$100

New Hardware and Software:

1	Development server (HPE ProLiant DL380)	$3250
1	MySQL Enterprise Database	$5000

Total development costs: **$42225**

Cost Matrix for Candidate 3

Personnel:

Quantity	Resource	Cost
1	Project Manager (160 hours/ea $45.00/hr)	$7200
1	Business Analyst (100 hours/ea $30.00/hr)	$3000
1	Systems Analyst (50 hours/ea $25.00/hr	$1250
1	UML Engineer (70 hours/ea $30.00/hr)	$2100
1	Quality Analyst (50 hours/ea $25.00/hr)	$1250
1	Technical Writer (25 hours/ea $25.00/hr)	$625
1	Database Specialist (50 hours/ea $35.00/hr)	$1750
1	System Architect (50 hours/ea $35.00/hr)	$1750
2	UI Designer (30 hours/ea $30.00/hr)	$1800
3	Programmer (120 hours/ea $25.00/hr)	$9000

Expenses:

1	Google Developer Registration	$25
3	Swift and iOS development training ($2000/person)	$6000
1	Apple Developer Registration	$100

3	UWP development training ($1500/person)	$4500
1	Windows Developer Registration	$20

New Hardware and Software:

1	Development server (HPE ProLiant DL380)	$3250
1	MySQL Enterprise Database	$5000

Total development costs: **$48620**

6.4. Appendix D - 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

Relevant Requirements: Google's reCAPTCHA

Post-conditions:

1. System displays the dashboard.

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-Web-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: 2/28/2020

Use Case ID: MIDAS-2-Mobile-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: 2/28/2020

Use Case ID: MIDAS-3-Web-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: 3/3/2020

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

Use Case Level: {High-level, system-level end-to-end, functional sub-use }

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-Mobile-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: 3/3/2020

Use Case ID: MIDAS-4-Web-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: 3/3/2020

Use Case ID: MIDAS-4-Mobile-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-Mobile-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: 3/3/2020

Use Case ID: MIDAS-5-Web-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: 3/3//2020

Use Case ID: MIDAS-5-Mobile-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: 3/3//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-Web-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: 03/02/2020

Use Case ID: MIDAS-7-Mobile-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-Mobile-Login, MIDAS-15-Mobile-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: 03/02/2020

Use Case ID: MIDAS-8-Web-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: 01/23/2020

Use Case ID: MIDAS-8-Mobile-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: 01/23/2020

Use Case ID: MIDAS-9-Web-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: 03/02/2020

Use Case ID: MIDAS-9-Mobile-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: 03/02/2020

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-Web-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: 3/3/2020

Use Case ID: MIDAS-11-Mobile-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: 3/3/2020

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-Web-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: 3/3/2020

Use Case ID: MIDAS-14-Mobile-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-Mobile-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: 3/3/2020

Use Case ID: MIDAS-15-Web-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: 3/3/2020

Use Case ID: MIDAS-15-Mobile-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-Mobile-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: 3/3/2020

6.5. Appendix E - Diary of Meetings

Date: January 7, 2020

Time: 8:50 P.M. – 9:05 P.M.

Location: PG6 114 – Classroom

Persons in Attendance:

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

Persons Late: N/A

Summary of Discussion:

- Exchanged contact information between team members
- Created an online group chat where team members can communicate at any time.
- Assigned team roles.
- Began brainstorming project ideas, but no decision was made.

Assigned Tasks:

- Continue to think of a project idea – assigned to all team members.

Date: January 9, 2020

Time: 8:45 P.M. – 9:07 P.M.

Location: PG6 114 – Classroom

Persons in Attendance:

· Manuel Toledo – Team Leader

· Edward Gil – Minute Taker

· Gabriel Alfonso Alonso – Time Keeper

Persons Late: N/A

Summary of Discussion:

· Chose a project idea to be our potential final choice – a personal budgeting app that allows viewing stocks and includes an A.I. to predict future monthly budget.

· Started thinking of use cases for the project.

Assigned Tasks:

· Create a google doc that will be used to list our use cases – assigned to Manuel.

· Come up with at least 3 use cases (including 1 security use case) by next class – assigned to all team members.

Date: January 13, 2020

Time: 8:34 P.M. – 9:09 P.M.

Location: PG6 114 – Classroom

Persons in Attendance:

· Manuel Toledo – Team Leader

· Edward Gil – Minute Taker

· Gabriel Alfonso Alonso – Time Keeper

· Maxime Camille

Persons Late: N/A

Summary of Discussion:

· Further discussed project idea and use cases; professor approved the project.

· Possible security use cases that are more specific to the application: 1. Encrypt sensitive data like user spending history or stock purchases. 2. Link bank accounts by signing into the bank's website rather than simply inputting bank account info.

· Discussed who the application will cater to (middle class or the rich), so that it is not just a copy of an existing application.

· Existing similar applications for feasibility assessment – robinhood, coinbase

Assigned Tasks:

· Begin working on a use case documentation to be presented next week – assigned to Manuel.

· Being working on a security use case documentation to be presented next week – assigned to Gabriel.

Date: January 16, 2020

Time: 8:37 P.M. – 9:08 P.M.

Location: PG6 114 – Classroom

Persons in Attendance:

· Edward Gil – Minute Taker

· Gabriel Alfonso Alonso – Time Keeper

Persons Late: N/A

Summary of Discussion:

· Began working on a use case for linking accounts; not too sure what the use case level means: high-level, system-level end-to-end, etc.

· Use case document template: our documents should follow a similar structure to the example from Canvas.

Assigned Tasks:

· Continue working on use case documentation – assigned to all team members.

Date: January 21, 2020

Time: 8:48 P.M. – 9:13 P.M.

Location: PG6 114 – Classroom

Persons in Attendance:

· Manuel Toledo – Team Leader

· Edward Gil – Minute Taker

· Gabriel Alfonso Alonso – Time Keeper

Persons Late: N/A

Summary of Discussion:

· Clarified the correct use case document format and continued working on them.

- Professor clarified some information:
    - Use case id: use projectAcronym-number-shortDescription.
    - Use case level: most will be system level end-to-end. High-level is more general.
    - Trigger system response: assume normal execution. Any possible variances in the steps should go under Alternative Courses of Action.
    - Exceptions: are unforeseen circumstances.

Assigned Tasks:

- Continue working on use case documentation to present next class – assigned to all team members.

Date: January 23, 2020

Time: 8:55 P.M. – 9:05 P.M.

Location: PG6 114 – Classroom

Persons in Attendance:

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

Persons Late: N/A

Summary of Discussion:

- Professor gave feedback on Captcha use case:
    - In general, the captcha is a separate use case from the 4th login attempt
    - Preconditions: Should include the 3 unsuccessful logins, and the steps where credentials are sent to the system and checked for validity.

- Trigger: The captcha appears on the screen
  - § System responds by: captcha shows a message saying how to proceed (input phrase, or click on images, etc.)
- Post conditions: captcha accepts the user input
- Alternative courses of actions:
  - § At step …, the user may choose to cancel the captcha process
  - § At step …, the user may enter incorrect captcha phrase, or click incorrect captcha images (depending on which captcha we want)
- Related use cases: 4th login
- Frequency: how many accounts in total?

Assigned Tasks:

· Continue working on use case documentation – assigned to all team members.

Date: February 11, 2020

Time: 8:56 P.M. – 9:02 P.M.

Location: PG6 114 – Classroom

Persons in Attendance:

· Edward Gil – Minute Taker

· Gabriel Alfonso Alonso – Time Keeper

Persons Late: N/A

Summary of Discussion:

- Discussed possible alternative solutions:
  - § Web interface + ios app + android app
  - § Web interface + ios app + android app + Windows desktop app

Assigned Tasks:

· Continue working on use case documentation and UML diagrams – assigned to all team members.

Date: February 20, 2020

Time: 8:51 P.M. – 9:05 P.M.

Location: PG6 114 – Classroom

Persons in Attendance:

· Edward Gil – Minute Taker

· Gabriel Alfonso Alonso – Time Keeper

Persons Late: N/A

Summary of Discussion:

· Refined the items on the project schedule.

· Discussed technical roles
    - Business analyst – Edward
    - UML engineer - Manuel

Assigned Tasks:

· Continue working on use case documentation and UML diagrams – assigned to all team members.

7. References

1. Casserly, M. (2019, February 8). Which is the more popular platform: iPhone or Android? Retrieved from https://www.macworld.co.uk/feature/iphone/iphone-vs-android-market-share-3691861/

2. Mobile Vs. Desktop Usage (Latest Data For 2020). (n.d.). Retrieved from https://www.broadbandsearch.net/blog/mobile-desktop-internet-usage-statistics