1. Project Overview

Our software development project is a financial tracking application that aims to give users a greater understanding and control of their personal finances by enabling them to visualize and understand their spending and income in a more efficient and effective manner.

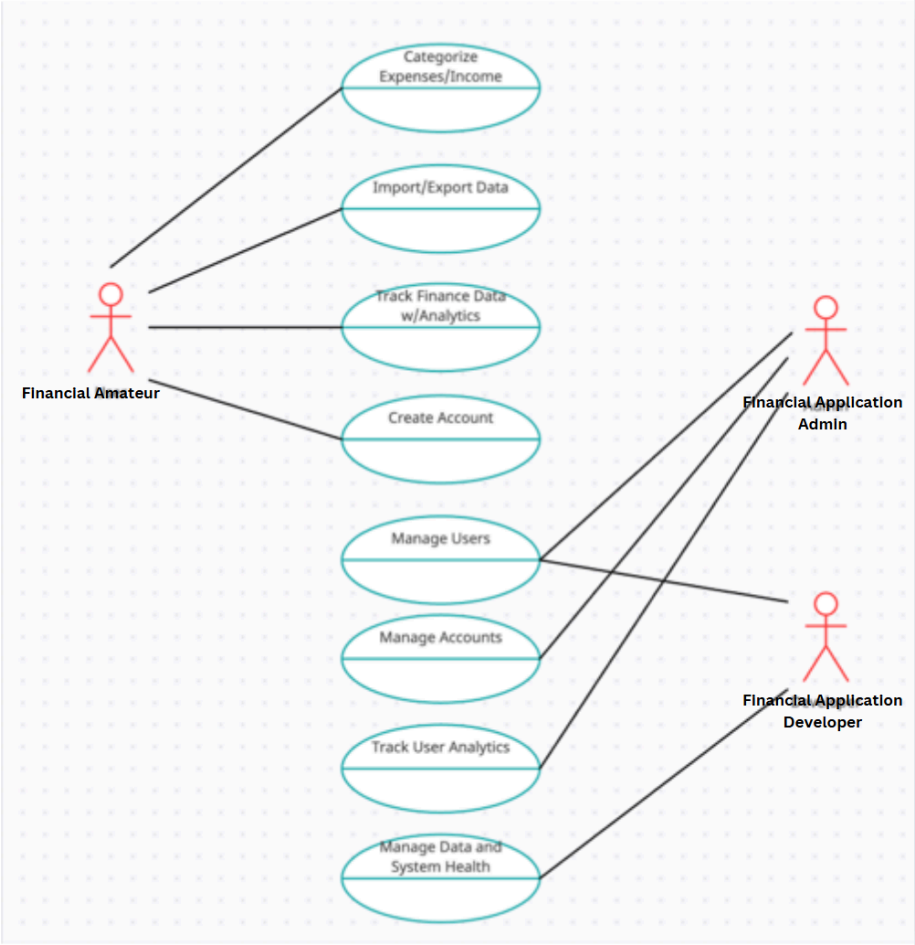
There are several stakeholders for this project:
Financial non-professionals - these will form the general user base of the application; their stake in this application is having a secure, fast, and effective way to track their finances.

Financial application administrators - administrators will have a significant stake in the application, as the application needs to be organized in such a way that they can efficiently manage the application and user data.

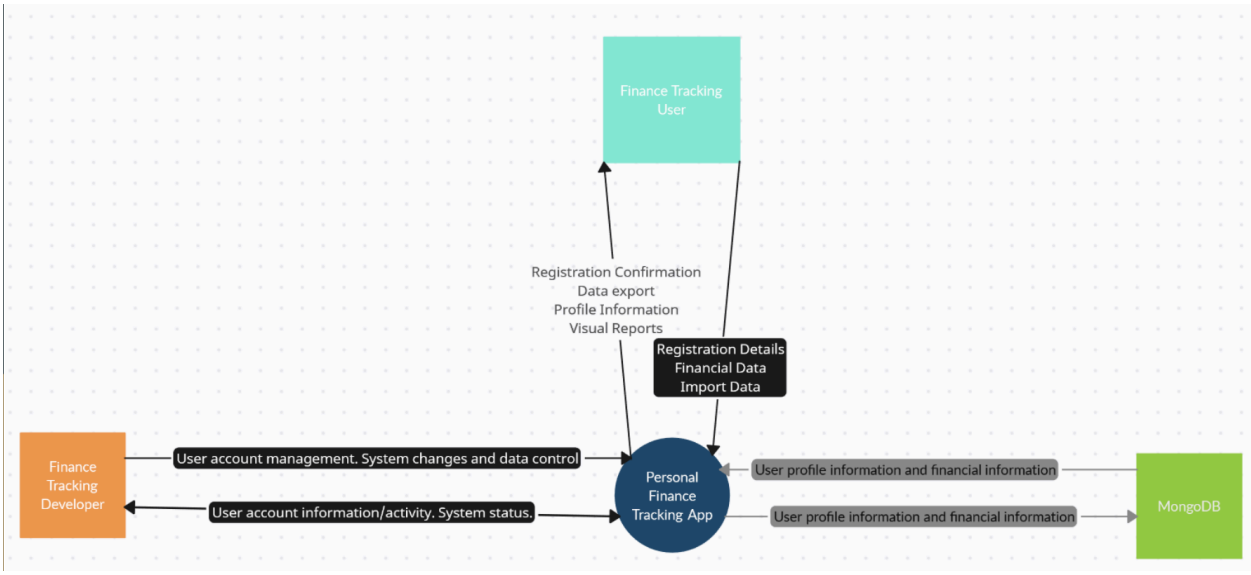
Financial application developers - developers will also have a large stake in the application as the application must be designed in such a way that further development and feature addition is facilitated as further requirements are introduced, as is common in Agile development.

Our software will address the problems of all of these stakeholders. It will address the issues of financial non-professionals by providing a simple user interface and tools to enable them to better understand their finances. It will address the issues of financial application administrators by providing them a simple application structure from which they can manage the application without significant overhead. Finally, financial application developers will be enabled to develop significantly faster through the use of simple, scalable architecture that will be able to be expanded and changed with significantly less overhead than other applications.

Use-case Diagram:



Context Diagram:



User Stories:
1. As a user, I want to create an account, so that I can access my financial tracking information.
2. As a user, I want to add my income and expenses, so that I can track my monthly and yearly financial activities.
3. As a user, I want to see my financial data represented in graphs, so that I can easily understand my financial patterns
4. As a user, I want to categorize my income and expenses, so that I can understand where I am earning/spending money
5. As a user, I want to import and export my financial data, so that I can back it up and transfer it between services when needed.

Backlog:
Completed:
As a user, I want to create an account, so that I can access my financial tracking information
As a user, I want to add my income and expenses, so that I can track my monthly and yearly financial activities.
As a user, I want to categorize my income and expenses, so that I can understand where I am earning/spending money

To do:
As a user, I want to see my financial data represented in graphs, so that I can easily understand my financial patterns
As a user, I want to import and export my financial data, so that I can back it up and transfer it between services when needed.

Main Use Cases:

USE CASE NAME:	User Registration	**USE CASE TYPE** Business Requirements: ☐ System Analysis: ☐ System Design: ☑
USE CASE ID:	UC-01	
PRIORITY:	High	
SOURCE:	User Story 1	
PRIMARY BUSINESS ACTOR	User	
PRIMARY SYSTEM ACTOR	Personal Finance Tracking Web Application	
OTHER PARTICIPATING ACTORS:	• N/A	
OTHER INTERESTED STAKEHOLDERS:	• N/A	
DESCRIPTION:	A user wants to create an account for the web application to store financial information	
PRE-CONDITION:	User is connected to the internet and on the Sign-Up Form	
TRIGGER:	User submits the sign-up form	
TYPICAL COURSE OF EVENTS:	**Actor Action**	**System Response**
	Step 1: User clicks on the Sign Up button	**Step 2**: System displays the sign-up form
	Step 3: User enters name, email and password	**Step 4:** System validates all information was entered and no account exists for the email
	Step 5: User submits the form	**Step 6:** System creates the user's account and stores it in the database. System redirects the user to the landing page.
ALTERNATE COURSES:	User enters an existing email address. System prompts that an account already exists.	
	User does not fill out all fields. System prompts that all fields must be filled out.	
	User does not enter the correct email format. System prompts that email form must be correct.	
CONCLUSION:	User is able to sign up and have an account created in the system	
POST-CONDITION:	User is redirected to the landing page with successful signup.	
BUSINESS RULES	• User must fill out all required forms to sign up • User cannot have multiple accounts under one email address	
IMPLEMENTATION CONSTRAINTS AND SPECIFICATIONS	• Email must be parsed to make sure it doesn't exist in the database • Passwords must be stored in encrypted format	
ASSUMPTIONS:	• User does not have a pre-existing account/email address in the system.	
OPEN ISSUES:	Duplicate Account Creation	

USE CASE NAME:	Adding income and expenses	**USE CASE TYPE** Business Requirements: ☐ System Analysis: ☐ System Design: ☑
USE CASE ID:	UC-02	
PRIORITY:	High	
SOURCE:	User Story 2	
PRIMARY BUSINESS ACTOR	User	
PRIMARY SYSTEM ACTOR	Personal Finance Tracking Web Application	
OTHER PARTICIPATING ACTORS:	• N/A	
OTHER INTERESTED STAKEHOLDERS:	• N/A	
DESCRIPTION:	A user wants to enter their income and expense information for tracking and storage	
PRE-CONDITION:	User is logged into their account and on the Income/Expenses page	
TRIGGER:	User inputs income or expense into the appropriate field and submits the form	
TYPICAL COURSE OF EVENTS:	**Actor Action**	**System Response**
	Step 1: User clicks on Income/Expenses page	**Step 2**: System navigates the user to the page if user is signed in or redirects to login page
	Step 3: User enters income and expenses details in the form	**Step 4:** System validates the fields
	Step 5: User submits the form	**Step 6:** System stores the information in the database under the users account
ALTERNATE COURSES:	User attempts to submit the form without filling out appropriate fields. System prompts users to fill out all fields required.	
	User attempts to input information in the wrong format. System will prompt user to fix the formatting in the field.	
CONCLUSION:	Users income/expenses are uploaded to the database and added to their profile	
POST-CONDITION:	User's financial information is updated with new income/expenses	
BUSINESS RULES	• User must be logged in and data must get stored securely on MongoDB	
IMPLEMENTATION CONSTRAINTS AND SPECIFICATIONS	• Data must be securely stored and linked to the user using their unique identifier(email)	
ASSUMPTIONS:	• Information entered by users is accurate	
OPEN ISSUES:	Duplicate income/expense entries	

USE CASE NAME:	Financial Patterns	**USE CASE TYPE** Business Requirements: ☐ System Analysis: ☐ System Design: ☑
USE CASE ID:	UC-03	
PRIORITY:	Medium	
SOURCE:	User Story 3	
PRIMARY BUSINESS ACTOR	User	
PRIMARY SYSTEM ACTOR	Personal Finance Tracking Web Application	
OTHER PARTICIPATING ACTORS:	• N/A	
OTHER INTERESTED STAKEHOLDERS:	• N/A	
DESCRIPTION:	Users want to view graphical summaries based on their information to view financial patterns	
PRE-CONDITION:	User is logged into their account and has income/expenses uploaded to their profile	
TRIGGER:	User opens the Financial Patterns page within the application	
TYPICAL COURSE OF EVENTS:	**Actor Action**	**System Response**
	Step 1: User clicks on Financial Patterns page	**Step 2**: System navigates the user to the page if user is signed in or redirects to login page
		Step 3: System generates graphs and presents them to the user based on their information
ALTERNATE COURSES:	Users do not have existing data for their profile. System informs user no data exists and shows "Add Data" button to redirect user to Income/Expenses page	
CONCLUSION:	System displays graphical data and summary to user	
POST-CONDITION:	User is able to view their financial graphs	
BUSINESS RULES	• Graphs must be parsed based on users existing income/expenses information	
IMPLEMENTATION CONSTRAINTS AND SPECIFICATIONS	• Data must be displayed through proper data visualization tools and be legible	
ASSUMPTIONS:	• User has pre-existing financial data and is logged in	
OPEN ISSUES:	API performance for large amount of data	

USE CASE NAME:	Income/Expense Categorization	**USE CASE TYPE** Business Requirements: ☐ System Analysis: ☐ System Design: ☑
USE CASE ID:	UC-04	
PRIORITY:	High	
SOURCE:	User Story 4	
PRIMARY BUSINESS ACTOR	User	
PRIMARY SYSTEM ACTOR	Personal Finance Tracking Web Application	
OTHER PARTICIPATING ACTORS:	• N/A	
OTHER INTERESTED STAKEHOLDERS:	• N/A	
DESCRIPTION:	Users want to be able to categorize their expenses/income within the application	
PRE-CONDITION:	User is logged into the application and has sufficient permissions to edit their expenses/income	
TRIGGER:	User clicks on dropdown next to expense/income	
TYPICAL COURSE OF EVENTS:	**Actor Action**	**System Response**
	Step 1: User clicks on dropdown next to expense/income	**Step 2**: System displays list of expense/income types (dependent on whether expense or income)
	Step 3: User clicks on a type in the dropdown menu	**Step 4:** Income/expense is categorized under the selected type and the type is displayed in the dropdown. In the DB, the category is associated with this expense/income so it persists between sessions..
ALTERNATE COURSES:	If the user clicks outside of the dropdown menu when the menu is open instead of selecting a category, the dropdown menu will close.	
	If an item is already categorized, the selection of a 2nd category will unselect the first.	
CONCLUSION:	The user is able to categorize their expenses and income.	
POST-CONDITION:	The dropdown menu shows the category of the expense/income and the type is saved in the database to persist between sessions.	
BUSINESS RULES	• User should be limited to selecting one category per expense/income item	
IMPLEMENTATION CONSTRAINTS AND SPECIFICATIONS	• User permissions should be validated prior to allowing the user to access this item.	
ASSUMPTIONS:	• User has sufficient permissions to access and edit the income/expense item.	
OPEN ISSUES:	Duplicate categories	

USE CASE NAME:	Import/Export Financial Data	**USE CASE TYPE** Business Requirements: ☐ System Analysis: ☐ System Design: ☑
USE CASE ID:	UC-05	
PRIORITY:	High	
SOURCE:	User Story 5	
PRIMARY BUSINESS ACTOR	User	
PRIMARY SYSTEM ACTOR	Personal Finance Tracking Web Application	
OTHER PARTICIPATING ACTORS:	• User operating system	
OTHER INTERESTED STAKEHOLDERS:	• N/A	
DESCRIPTION:	Users should have the ability to import and export their financial data, so that they can back it up and transfer it between services when needed.	
PRE-CONDITION:	User must be signed in and validated	
TRIGGER:	User clicks the import/export button	
TYPICAL COURSE OF EVENTS:	**Actor Action**	**System Response**
	Step 1: User selects export option	**Step 2**: Application opens a prompt through their operating system allowing the user to choose where to save the output
	Step 3: User selects a file location in their OS	**Step 4:** Application generates an Excel output file from the user's data and sends it to the user where it is saved at the specified location.
ALTERNATE COURSES:	Users can cancel the process at any point by canceling the prompt within their OS or clicking outside of the menu in the application.	
	If the user selects import instead of export, a similar prompt will open allowing the user to select an existing Excel file in their OS to import. The application will then attempt to import it and validate the input. If the input is in the correct format, the import will proceed, otherwise it will fail. Any imported data will be displayed on the user's page once completed.	
CONCLUSION:	Data should be able to be imported/exported by the user at will.	
POST-CONDITION:	Either the data is imported (or an error is thrown if the import file is in the incorrect format) or the data is exported and saved locally for the user depending on their selection	
BUSINESS RULES	• User data should be updated within the UI immediately after data import is complete • User access must be validated immediately prior to import/export	
IMPLEMENTATION CONSTRAINTS AND SPECIFICATIONS	• As this is a sensitive operation, user validation should occur immediately prior to the import/export procedure to ensure the validity of the operation.	
ASSUMPTIONS:	• User is logged in	
OPEN ISSUES:	Duplicate import data/overwriting existing data	

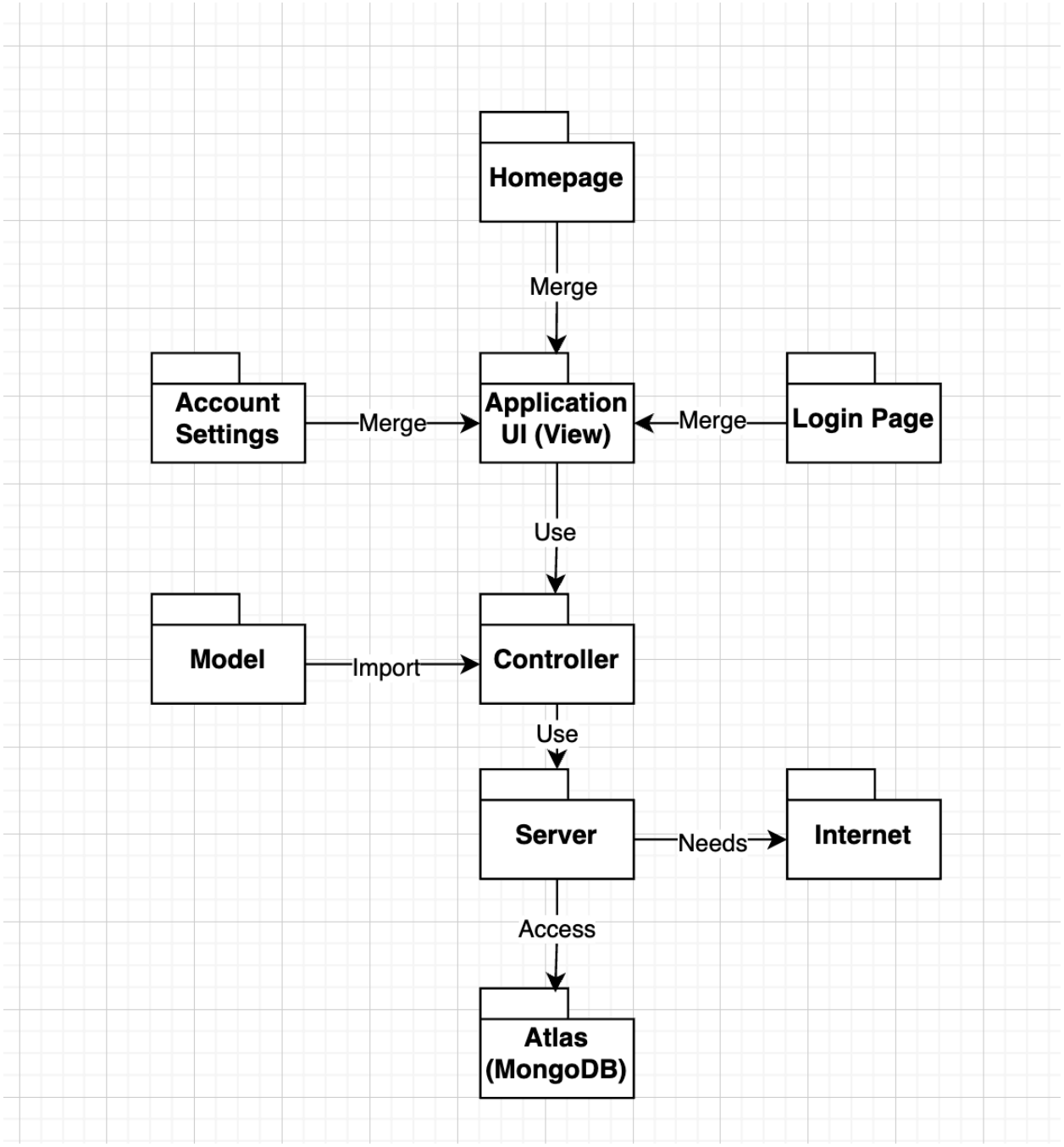
2. Architectural Overview

For our application, we have decided to go with the MVC (Model View Controller) design pattern. We arrived at this decision after considering several other design patterns, but none of them fit our needs as well as MVC. We examined the N-tier architecture design pattern for our application, but we decided that this would be inefficient for our application, which is relatively simple and does not require a large number of distinct layers. In particular, we wanted to avoid the complexity and inefficiency that comes with the communication between each layer as a separate entity. We also considered the monolithic architecture pattern, but we determined that this would not be feasible for our application, as we wanted to separate the development of our UI, business logic, and database components, as well as using an externally hosted DB to persist data. We then considered microservices which are potentially helpful in larger scale applications, but due to the small scale of this application, we felt that this architecture would not offer any significant benefits to our application. If we were to later add more application components, we might consider scaling to a microservice architecture using our current components as microservices.

By comparison, MVC offers a wide variety of advantages for our application: by logically separating the model, view, and controller, we are able to divide the development effort between these items. It also offers us an effective way to organize our work while avoiding unnecessary dependencies between components, making it easier to decouple components and fix issues. Finally, MVC was a familiar architecture that is relatively simple to implement, allowing for developers to quickly and easily make changes to the application without as much learning required.

2.1 Subsystem Architecture

Subsystem Dependencies Diagram:



Our application utilizes the MVC (Model View Controller) architecture. The 3 pages, homepage, account settings, and login page, make up our UI, or view. Our view then sends information to the controller, which controls the business logic of the application. The controller also imports the model, which is used to control our data structures of our database. The controller sends this information to the server component, which runs the server application and builds the connection to the database. It requires

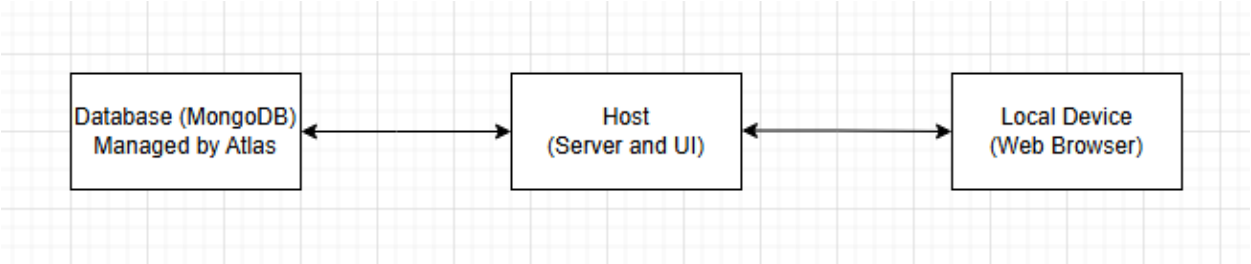
access to the internet in order to create the DB connection. Finally, our application connects to Atlas which is a cloud host for MongoDB that we are using as our database.

This architecture offers us a wide variety of benefits including all of the typical benefits of the MVC architecture, such as ease of development, decoupling of components, and simple organization of the project structure.

2.2 Deployment Architecture

For references on UML deployment diagrams, see the Sparx tutorial or the optional UML text listed on the syllabus, or visit www.uml.org.

UML Deployment Diagram:



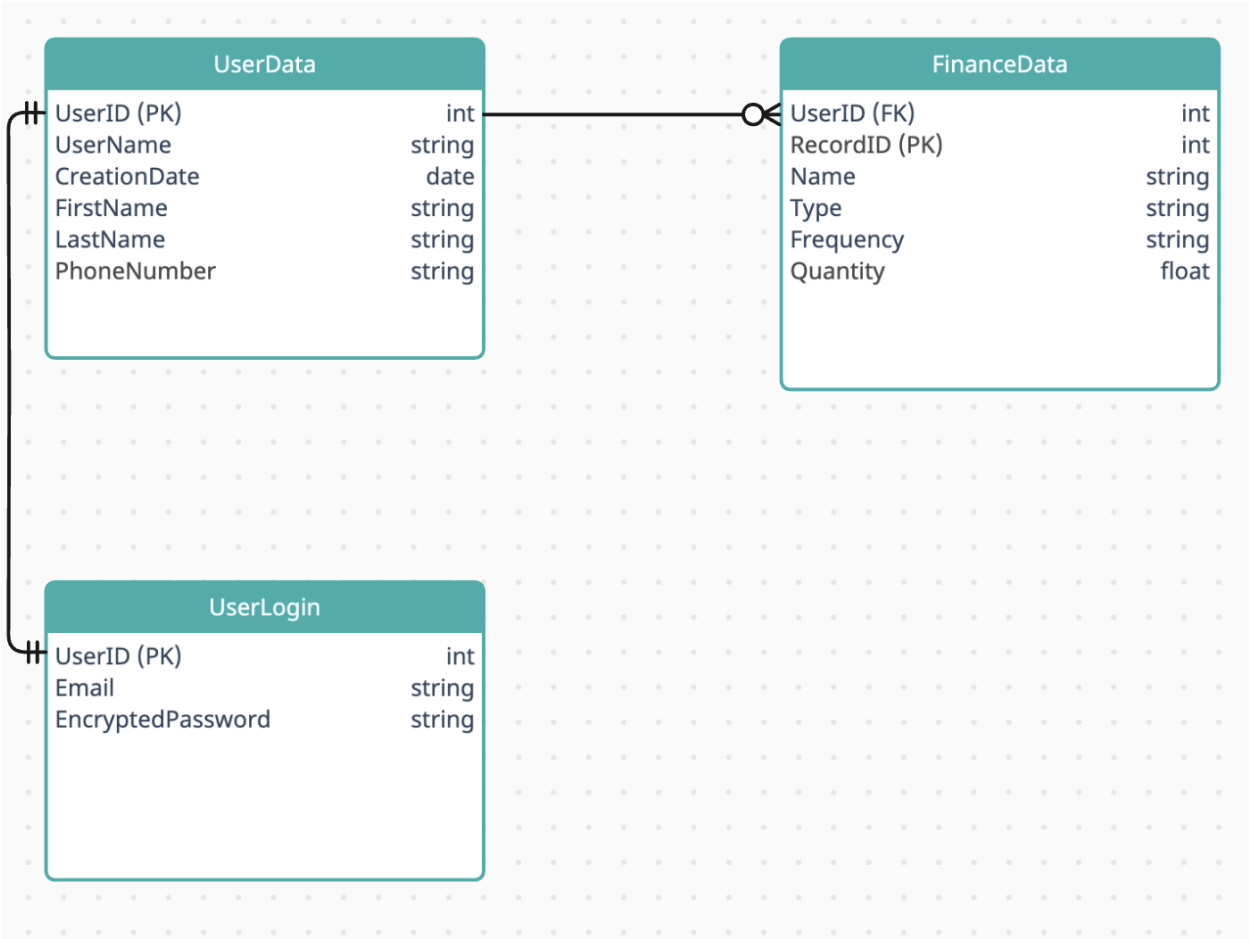
Deployment diagram - Database and Host are linked via HTTP, and the host and local device communicate via HTTP. Both connections are created wirelessly using wi-fi connection.

Our application will be deployed primarily on the host system, which will be running both the UI and server components. The application will be accessed by sending HTTP requests through a web browser on the local device used to access the application. These requests will then be handled by the host, which will send the necessary information (including the UI and user data) to the local device. User data is persisted in an Atlas MongoDB cloud instance. Any updates or changes to user data will be sent to the Host, which will then process these requests and send updates to the database via HTTP requests. Effectively, all communication between entities will be managed through HTTP requests, which will be handled through the use of an API on the host side as well as predetermined protocols for the cloud database.

Due to the abstracted nature of the Atlas hardware and the small scale of the project, we have not considered hardware limitations in the scope of this project - however, the local device should be any consumer device capable of running a web browser with a standard amount of RAM, and the host and database can be scaled appropriately to meet demand through either horizontal or vertical scaling.

2.3 Persistent Data Storage

Persistent data storage in the financial application will be accomplished through the use of MongoDB, which is acting as our database. We will be following this ERD for our data model:



Data being stored will include user login information and user data, which are used in the user management and log-in processes, and finance data for each user, which is used to save and display user financial data for application.

2.4 Global Control Flow

Overall, our application is primarily event-driven. Users can trigger a wide variety of operations with minimal order dependence. The system waits for the user to send requests with relevant information and then processes the request and sends a response to the user. This is how we handle our log-in and user information access structure, as well as the how the user financial data is served and displayed to the user.

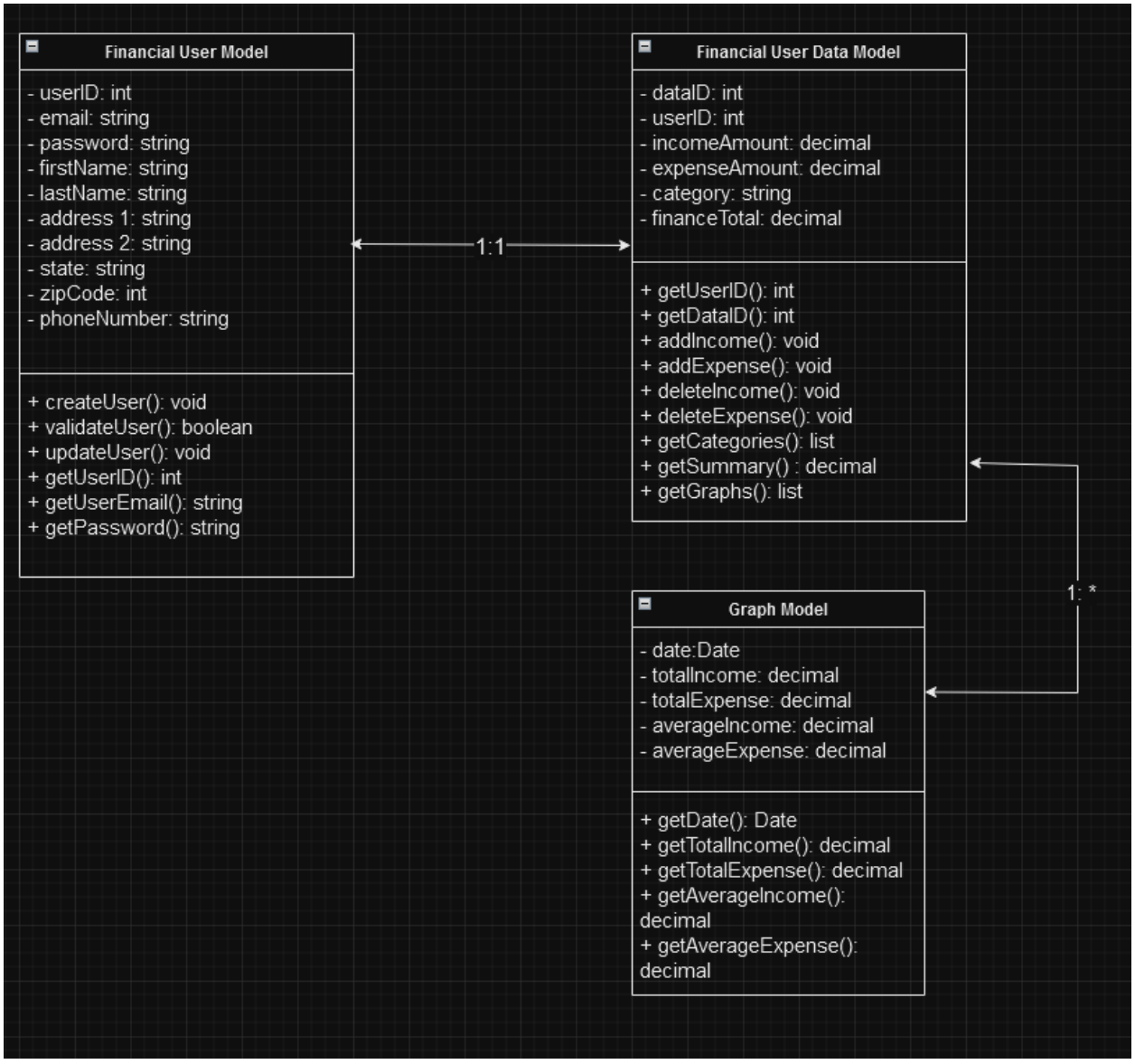
We also have minimal time dependency on our system - the only significant time dependency is that the application reads the local system clock for the user in order to display the correct month information. There are no timer-controlled actions, nor are any of our system responses time-sensitive, so there are no time constraints. Requests and updates are simply performed as the user triggers events through using the application.

Finally, there is no significant use of concurrency in our application. The only exception is the database which is externally managed; typically, databases are multithreaded in order to allow for multiple users to access the database at the same time without significant performance conflicts. The server and UI components are not currently multithreaded, though if this application were scaled to a larger user base, it would become necessary to multithread the request handling in order to avoid performance issues associated with large numbers of users making simultaneous requests to the system.

3 Detailed System Design

3.1 Static view

For reference material, see Robert Martin's article on UML class diagrams, the Sparx tutorial on UML package and class diagrams, or the optional UML textbook recommended on the syllabus. For reference material on design patterns, check out the POSA and GoF or this very handy online catalog of design patterns or this website on common patterns.



For our application of a Personal Finance Tracker we decided to have three data models to represent how our application functions and how these models are connected to one another. The financial user model contains information about a user's account when they sign up for the application. The model contains userID, email, name, password, address, and phone number. This model is responsible for

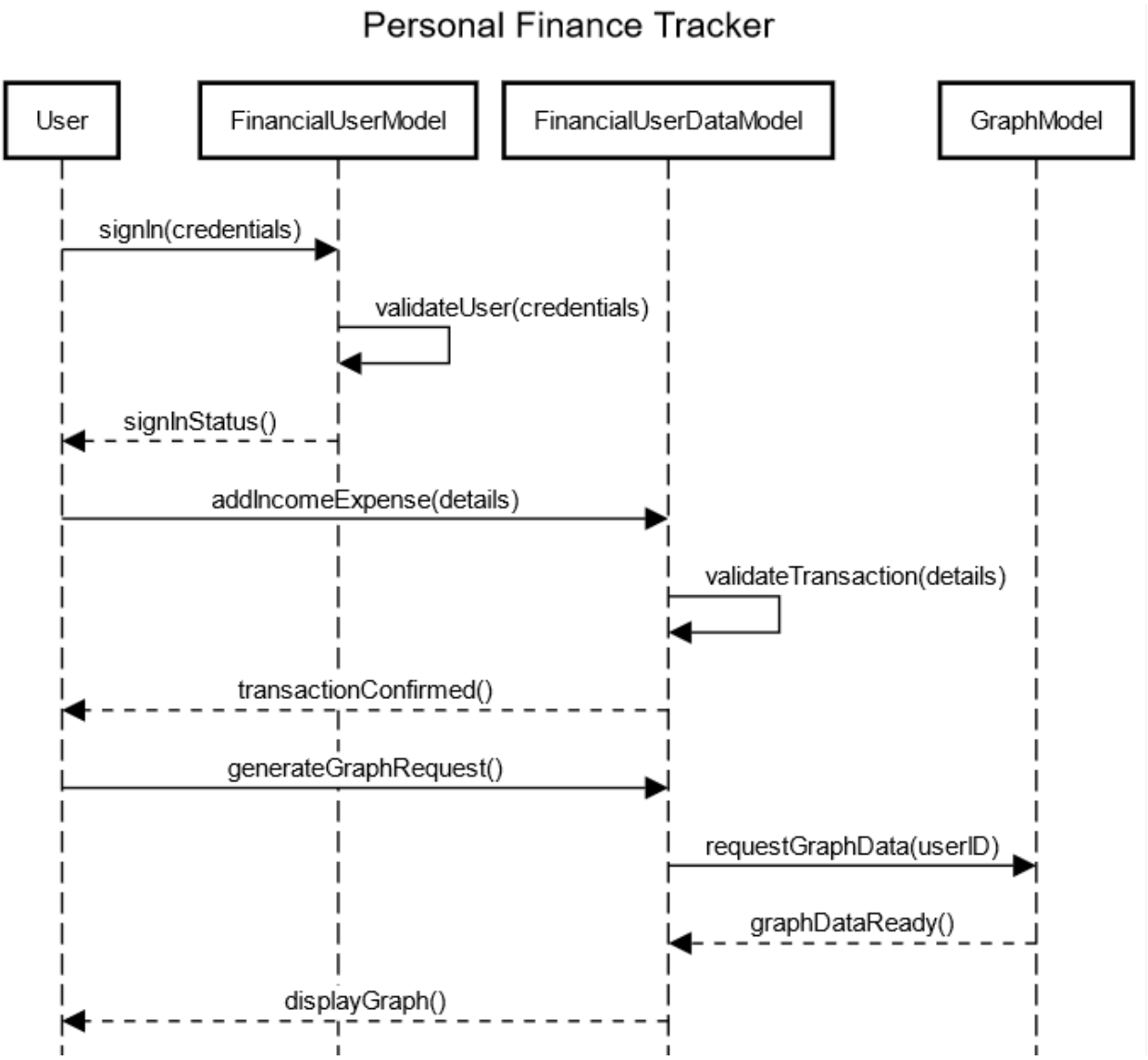
creating unique user models so that each user can have their data stored securely based on their primary key, userID. The user model will allow us to link users to the data model to access user specific financial information they upload to MongoDB.

The financial user data model represents the financial data transactions a user has on our application. The data model contains information such as dataID, userID, incomeAmount, expenseAmount, categories and financial total. The data model is responsible for uniquely storing financial information for each user by using the foreign key userID and having a primary key called dataID. The data model has functions such as adding income and expenses, deleting income and expenses, creating categories, and getting data and graphical information. The data model has a 1 to 1 relationship with the user model because each user will have one data model connected to their account.

The graph model will be responsible for taking advantage of the graphing API to create visualizations for users' financial information. The model contains attributes such as date, total income and expense, and average income and expense. The model performs functions such as getting the date, getting total income and expense, and getting average income and expense.

3.2 Dynamic view

Make sure to add your Testplans, previous and current sprints and brief sprint reviews as well as any additional items you feel are helpful at any point. Remember, it's not necessary to add documents that are not providing useful information. Only necessary items. You can add/subtract items or update them using version control to your design.



4.0 GitHub Repository

Repo: https://github.com/Sylnus/Personal-Finance-Tracking-App

Please use the Google Drive link with you UNCC email address to access the video because the file size was too large for GitHub

Link: https://drive.google.com/file/d/1lOEb6BHzQ1cW136lh_IUtOkFJ1yi1GPW/view?usp=sharing