INVESTAGRAM

Design Document

Team 39
2/9/2024

Mohana Barve

Tanay Gondil

Arjav Seshachalam

Elliot Smith

Jason Yang

1) Purpose

Our fintech and social networking platform aims to redefine the way individuals engage with investments by offering an integrated digital ecosystem. In response to the growing interest in investing, we recognize the necessity for a platform that seamlessly combines financial tools with social networking features.

Existing social media platforms lack the specialized functions required for effective portfolio management and meaningful connections within a dedicated financial community. Popular platforms such as Twitter, Instagram, Snapchat, and Facebook lack integrated features for new investors. Even competing platforms aren't really helpful on an individual basis as our platform fills this gap by providing users with tailored insights into their portfolios, particularly focusing on diversification aspects.

We aspire to revolutionize investment engagement by providing a space where users can not only gain valuable insights into their investments but also share their financial journeys with close friends and connect with a broader community of like-minded investors.

1.1 Functional Requirements

1.1a User Registration and Account Management

As a user,

- I would like to be able to create an account.
- I would like to be able to login and manage my account.
- I would like to reset my password or access my username via email if I forget it.

- I would like to change the email, username, or password of my account once I’m logged in to keep my account safe.
- I would like to delete my account along with all my stored information to prevent my information from being accessed if I don’t want to use the social media anymore.

1.1b Social Networking Features

As a user,

- I would like to message other users if I want to talk to them privately
- I would like to make, edit, view, or delete posts to share any news that interests me.
- I would like to see a homepage for when I use the website.
- I would like to like and comment on posts to share my insights.
- I would like to share posts I find interesting on other social media.
- I would like to create group chats to form groups that share a common interest.
- I would like to create or edit my profile page so others can know more about me.
- I would like to set/adjust my profile picture so others can identify who I am easily.
- I would like to view other’s profile pages if they interest me.
- I would like to view someone else’s post history to see what they posted.
- I would like to share posts others make with my friends (message within the website).

1.1c Portfolio Management

As a user,

- I would like to see my portfolio in a visually pleasing manner with charts and graphs so that I can analyze my portfolio better.
- I would like to adjust what other users can see about my portfolio so that I can protect specific information.

- I would like to see the profit/loss of my investments by year/month/day to see my progress over time.
- I would like to be given general diversification insights based on my portfolio.
- I would like to view my daily progress on a calendar so I can see my earnings (History of PnL).

1.1d Friending

As a user,

- I would like to “friend” others to form connections with them.
- I would like to see my friends’ portfolio information so I can gain insight into what to invest in.
- I would like to see recommended users to friends based on prior connections and interests.
- I would like to set my friends to different levels of trust so that they can see varying levels of information about my portfolio.
- I would like to have a tier system for different friend levels where the tiers will be set so that friends can see varying levels of my portfolio information.
- I would like to have an easier way to move my friends within different tiers.
- I would like to have a page where I can view a list of all the friends for each tier.

1.1e Notifications and Insights

As a user,

- I would like to report posts and users that are harmful to the platform.
- I would like to see a notification of friend requests through email.
- I would like to see a notification of the daily summary to be informed of my investments.

- I would like to see a notification of messages sent to me so I can respond.
- I would like to be notified if a friend creates or shares a watchlist.
- I would like to see the results of the poll once it ends via a notification to my email.
- I would like to change what notifications I receive in my email to reduce the clutter.

1.1f Viewing Stocks

As a user,

- I would like to search company stocks by their Tickers (for example, Microsoft is denoted as MSFT).
- I would like to see the financial data on the company's stock.
- I would like to create a watchlist of stocks I am interested in.
- I would like to have the ability to limit who can view my watchlist based on friend trust level.
- I would like to see a performance of my watchlists for comparison every week.

1.1g Polls and Predictions

As a user,

- I would like to start polls to see what others think.
- for any stock of interest, I would like to have the ability to either predict bullish (the stock will go up) or bearish (the stock will go down) for the day.
- I would like to see my history of predictions and the accuracy of my predictions.
- I would like to compare the accuracy of predictions between my friends and me every week.
- I would like to see the history of polls I've voted on.

- If time permits, I would like to be able to sort posts I make, polls I participate in, and my friends by newest to oldest and vice versa.

1.2 Non-Functional Requirements

Architecture and Performance

As a developer,

- I would like to create the front end using React JS, ensuring a responsive and visually appealing user interface.
- I would like to use Node JS as the backend infrastructure which will manage user data efficiently and handle server-side operations seamlessly.
- I would like to use MongoDB as the chosen database which will securely store user profiles and financial data while facilitating scalability.
- I would like the system to respond to user actions within 200 ms, ensuring a smooth and immersive user experience.
- I would like the website to accommodate a minimum of 1000 concurrent users without compromising performance.

Server

As a developer,

- I would like the server to support real-time communication between the server and the client.
- I would like the server to be able to store user information along with other information to a database.

Security

As a developer,

- I would like user data to be stored on MongoDB Atlas, with automatic encryption at the cloud level for enhanced security.
- I would like email verification for account creation to protect user accounts.

Usability

As a developer,

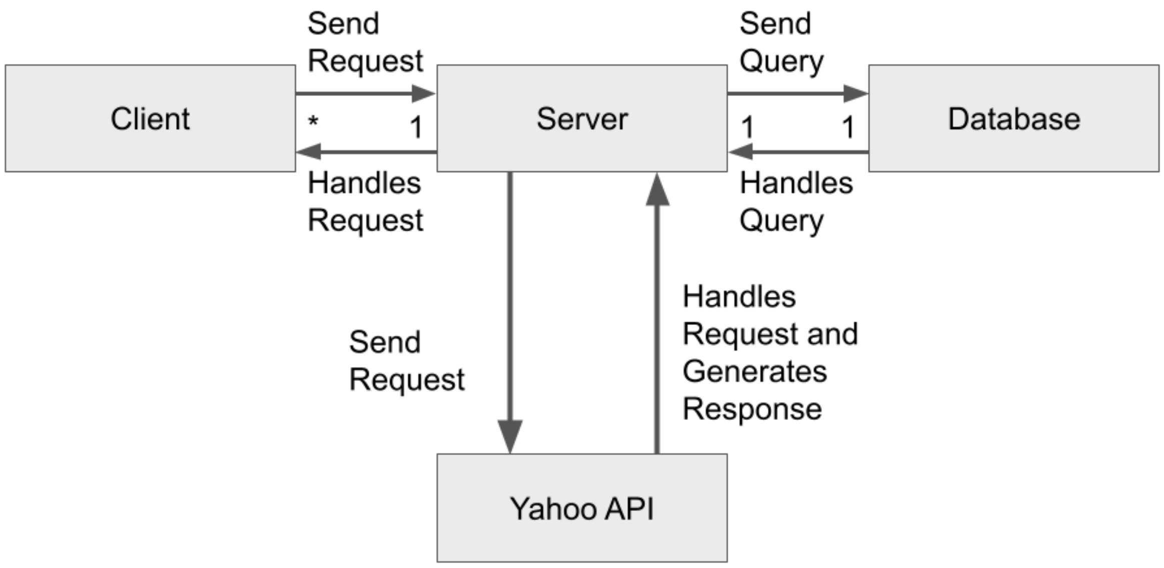
- I would like the platform's user interface and experience to be designed for streamlined navigation, providing users with intuitive access to brokerage accounts, social connections, and post-sharing functionalities.

2) Design Outline

High-Level Overview

Our project is a web application designed to provide users with insights into their portfolio allocation and facilitate communication between users via text messaging within the platform. To achieve this, we've chosen Next.js for the front end to ensure seamless user interaction and Node.js for the backend to handle requests efficiently.

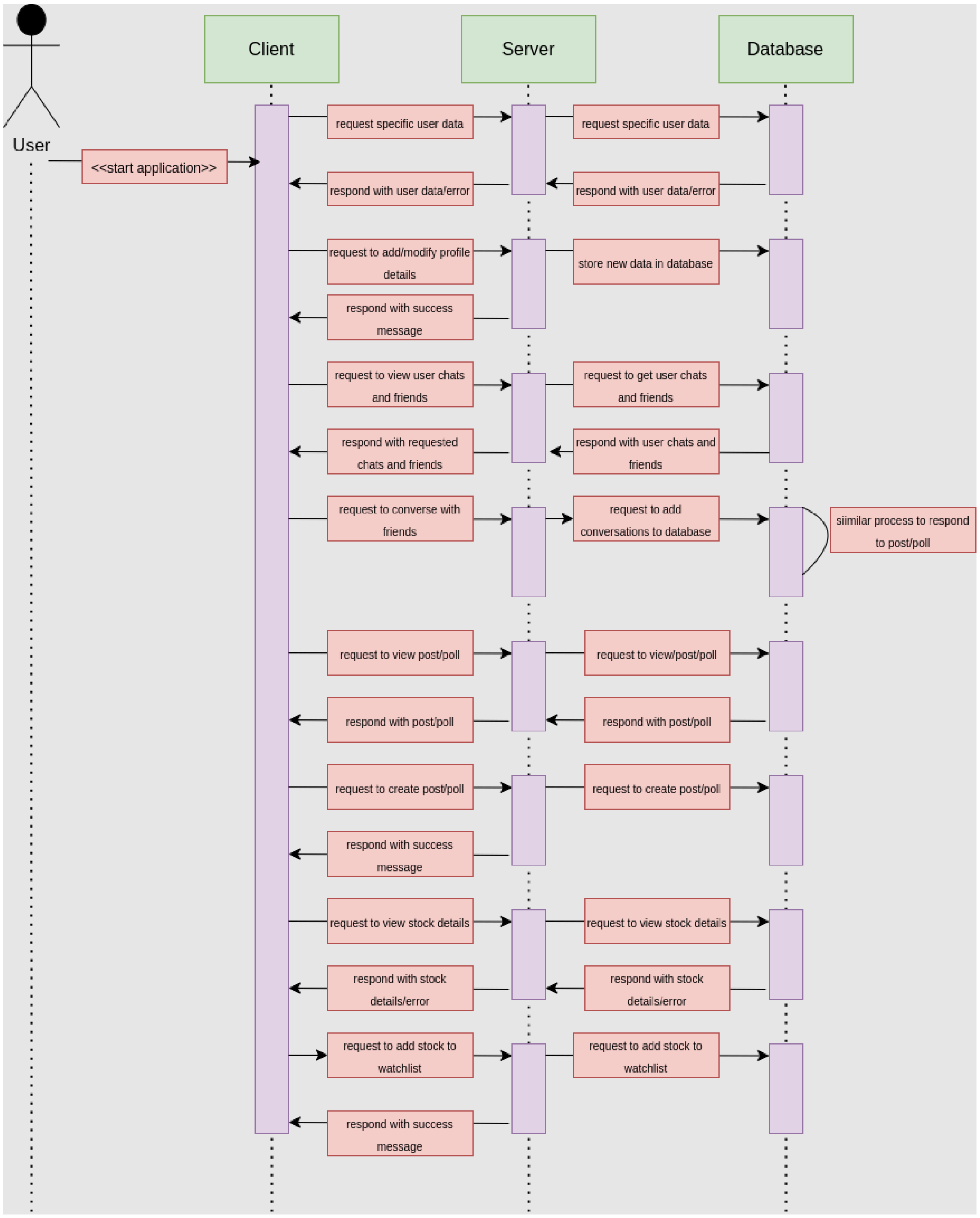
Employing a simple server-client model, we'll utilize MongoDB for database management, given the substantial volume of information we need to store. This combination of technologies allows us to create a robust and scalable application that meets the demands of our users effectively.



- Client
    - The client is responsible for enabling users to interact with our system through an interface. It communicates with the server by sending JSON requests and receives responses in return. After interpreting these responses, the client updates the user interface accordingly, ensuring smooth interaction and displaying changes promptly for a better user experience.
- Server
    - The server is responsible for receiving and managing requests from clients. It validates these requests and then sends queries to the database to add, modify, or retrieve data as needed. After processing the requests, the server generates appropriate responses and sends them back to the respective clients, ensuring seamless communication and interaction with the system.
- Database
    - A database serves as the repository for all application data, including user details, post history, and more. It responds to queries initiated by the server, extracting the requested data, and then sends it back to the server for further processing. This interaction between the database and server ensures efficient retrieval and management of data within the application.
- API
    - The API serves as a bridge for accessing and managing data within the application. Specifically, the Yahoo API for financial information allows developers to request real-time or historical data related to stocks and currencies. It processes incoming requests, retrieves the requested data, and delivers it back to

the requester, enabling seamless integration of financial information into various applications and systems.

UML Diagram


User
Client
Server
Database
<<start application>>
request specific user data
request specific user data
respond with user data/error
respond with user data/error
request to add/modify profile details
store new data in database
respond with success message
request to view user chats and friends
request to get user chats and friends
respond with requested chats and friends
respond with user chats and friends
request to converse with friends
request to add conversations to database
siimilar process to respond to post/poll
request to view post/poll
request to view/post/poll
respond with post/poll
respond with post/poll
request to create post/poll
request to create post/poll
respond with success message
request to view stock details
request to view stock details
respond with stock details/error
respond with stock details/error
request to add stock to watchlist
request to add stock to watchlist
respond with success message

3) Design Issues

Functional Issues

How are we going to have user profile customization?

- Options:
- Option 1: Allow users to choose from a set of predefined avatars and backgrounds
- Option 2: Enable users to upload their own profile picture and select custom backgrounds
- Option 3: Provide a combination of predefined avatars and backgrounds along with the option to upload custom content

Choice: Option 2

Justification: We want to give the users the freedom to upload their own profile pictures and choose custom backgrounds because it will allow themselves to uniquely identify themselves, and it allows the user to have a place to showcase their individuality. It will contribute to an engaging user experience, and we believe that this is the option that the user will appreciate the most. We believe that this fits with the goal of our project, which includes fostering creativity and it allows the user an opportunity to personalize their account.

How will we implement our portfolio privacy settings?

- Options:
- Option 1: Allow users to set their entire portfolio as public, visible to all platform users
- Option 2: Implement privacy settings that enable users to customize visibility based on friend trust levels

- Option 3: Introduce a tiered privacy system, allowing users to selectively share different portions of their portfolio with specific friend groups

Choice: Option 3

Justification: A tiered privacy system allows the user to have control over their portfolio visibility, and it will allow the users to learn from each other and see whatever their privacy settings allow, all while protecting their privacy. This choice allows users to have their own preferences in their privacy settings so they can share specific aspects of their portfolio with different groups of friends or colleagues that they deem to be appropriate. We believe that this will enhance the user's confidence in protecting their financial information as well as make smart and educated financial decisions.

What will our algorithm be for the user's feed?

- Options:

- Option 1: Display the feed in chronological order, showing the latest posts at the top
- Option 2: Implement an algorithm that prioritizes posts based on user engagement, relevance, or friend interactions
- Option 3: Provide users with customizable news feed preferences, allowing them to choose between chronological and algorithmic displays

Choice: Option 2

Justification: We want to create an algorithm that prioritizes posts based on user engagement because it will make the user's experience on the platform more interesting and informative. In choosing this choice, the user will be able to see more relevant information for their own needs, and they will be more engaged because of this as well as by seeing what their friends and the largest and most meaningful posts are doing. This will continue to make the experience on the platform feel personalized, which should help with user engagement, retention, and recommendation.

How will we implement predictive polls?

- Options:
- ● Option 1: Enable users to create polls with fixed answer options
- ● Option 2: Allow users to input their own poll questions and answer choices
- ● Option 3: Implement a predictive element, where users can predict the outcome of a poll before it concludes

Choice: Option 2

Justification: We want to allow users to be able to ask the community whatever they feel is going to be most interesting and specific to their needs, and so it is essential that users need to be able to ask their own questions and have the answer choices be what they will find helpful. This will also add a layer of diversity to the community and the poll feature more specifically, so that polls will be less likely to be repeated. This will help to foster a culture that is active, and it will allow for different perspectives to be heard, which will give insight, and allow the users to see the varied opinions of those on the site.

Will we have some form of watchlist collaboration?

- Options:
- ● Option 1: Keep watchlists private, visible only to the user who created them
- ● Option 2: Allow users to share specific watchlists with friends or the entire community

- Option 3: Introduce collaborative watchlists where multiple users can contribute and modify the list

Choice: Option 2

Justification: We want users to be able to share specific watchlists with friends because we want to focus on creating great community engagement and information sharing, because we believe that it will create the most engaging and useful experience for the user. This would allow the platform to be more collaborative so that users can share their personal insights, advice and investment strategies. This also keeps the social aspect of the platform as a focus which will allow users to make connections over investment interests and strategies.

What will we take as sign-up information?

- Options:
- Option 1: Username and password only
- Option 2: Username, password, email address
- Option 3: Username, password, email address, phone number

Choice: Option 2

Justification: We want to collect all of the information that we need, but we also want to protect the users privacy, so we do not need to require the phone number because it will not be necessary for the purposes of the platform. We want to balance security and identification for the user, and we believe that means that we should just collect the username, password and email address. We will use the username and password for the sign up and for basic user identification, and we will also need the email address so that we can have another layer of verification and account recovery. We want the process to be as easy and stress free as possible, and we believe that this sign up and verification/recovery process will be the best for the user.

Non-Functional Issues

What frontend language/framework should we use?

- Options:
- Option 1: raw HTML + JavaScript
- Option 2: React.js
- Option 3: Angular
- Option 4: Next.js

Choice: Next.js

Justification: Next.js is a framework that works on top of React - it expands and streamlines its capabilities. React itself is well-documented and has a plethora of online support. React uses virtual DOM, which provides much better UX and performance as compared to raw HTML + JS. Furthermore, when compared to Angular, React is faster, and more programmer friendly. Next.js also has automated server rendering, as opposed to client-side rendering, which vanilla React uses.

What backend language/framework should we use?

- Options:
- Option 1: Spring Boot
- Option 2: PHP
- Option 3: Node.js

Choice: Node.js

Justification: Node.js is an extremely viable choice in building fast, scalable network applications. Using node also allows us to access npm - a package management system that allows for importing modules, such as Express.js, a very popular web development framework for Node.js. Furthermore, Node.js is also a logical choice for us because it allows us to stick to Javascript across the stack, for both frontend and backend.

What database should we use?

- Options:

- Option 1: MySQL
- Option 2: MongoDB
- Option 3: Oracle XE

Choice: MongoDB

Justification: Compared to MySQL and Oracle XE, MongoDB is a non-relational database, allowing us to break free from traditional rigid relational database structures. MongoDB can be scaled easily with no downtime, without changing our application. MongoDB is also incredibly fast for simple queries, and we can add new columns or fields without affecting existing rows or application performance. Furthemore, MongoDB doesn't limit storable data types - allowing for greater flexibility.

What styling language/framework should we use?

- Options:

- Option 1: Vanilla CSS

- Option 2: Bootstrap CSS
- Option 3: Tailwind CSS

Choice: Tailwind CSS

Justification: Tailwind CSS is one of the most used CSS frameworks. When compared to just using Vanilla CSS or Bootstrap, Tailwind is not only much easier to learn and understand, but it also speeds up the development process significantly. Tailwind has a very comprehensive documentation page, and also has much more customizability, when compared to other frameworks like Bootstrap.

How should we implement our verification process?

- Options:
- Option 1: Verify users through email
- Option 2: Implement a multi-step verification process
- Option 3: Have users verify through google

Choice: Option 1

Justification: We believe that verification through the email that the user provides is the most straightforward and effective method for user authentication. This choice aligns with common industry practices for similar products and it makes sure that the process is simple and easy of use for users. The email verification also adds another layer of security by confirming the legitimacy of user accounts and the information that they provide, which makes for the most reliable verification process.

What should be our target response time for user actions?

- Options:

- Option 1: Respond to any user action under 200 ms
- Option 2: Respond to any user action under 500 ms
- Option 3: Respond to any user action under 1 second

Choice: Option 1

Justification: A target response time under 200 milliseconds allows for the product to have a smooth and responsive user experience. This puts an emphasis on the efficiency of the product, ensuring that users experience minimal delay and enjoy a seamless and enjoyable time on the platform. A quick response time enhances user satisfaction and contributes to the overall perception of the platform's performance, and we believe that the 200 ms threshold is important to stay under to provide the best experience for the user.

How many users should our platform be able to handle concurrently?

- Options:

- Option 1: Handle at least 500 users concurrently
- Option 2: Handle at least 1000 users concurrently
- Option 3: Handle at least 2000 users concurrently

Choice: Option 2

Justification: We want our product to be able to handle a large number of people because we want it to be a place where groups of individuals can come together and discuss their plans and connect with each other, and we think 1000 concurrent users would be able to achieve that at this time. Handling at least 1000 users concurrently allows for a scalable and approach to the platform, that we could increase in the future. This choice ensures that the platform can a large number of users without significant performance degradation. It aligns with the project's goal of providing a platform that can support growing community of users while maintaining the performance that is up to our standards.

How should we handle data security at the cloud level?

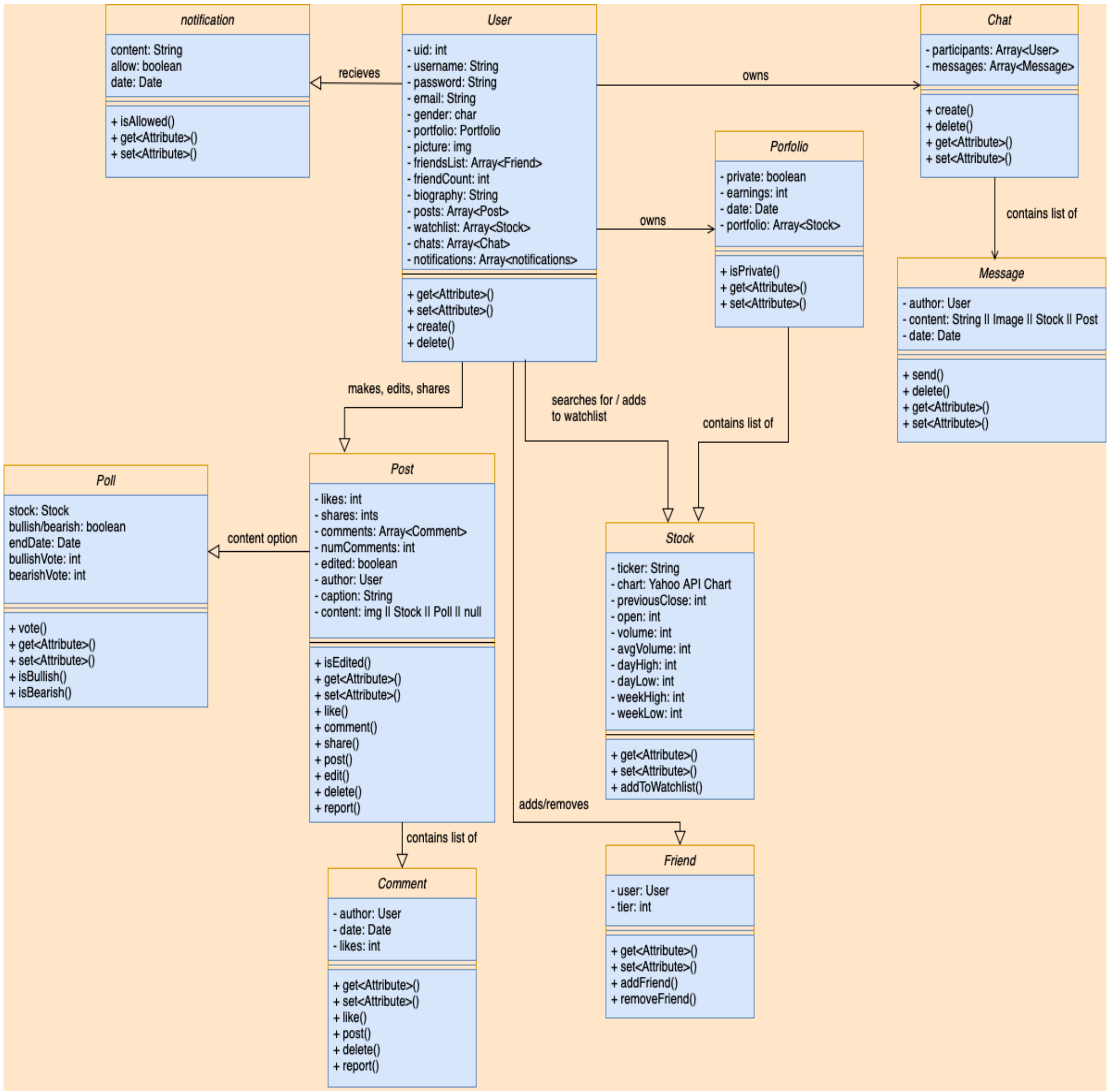
- Options:
- Option 1: Implement automatic encryption on MongoDB Atlas
- Option 2: Utilize a third-party encryption service
- Option 3: Manually handle encryption at the application level

Choice: Option 1

Justification: Implementing the automatic encryption on MongoDB provides a built-in security measure that is important for ensuring data security for the platform. The security features offered by MongoDB will make sure that the data is protected at the cloud level. Automatic encryption simplifies the implementation process, as it reduces the different potential vulnerabilities. It aligns with best practices for securing data in cloud-based environments, which will contribute to the integrity, security and confidentiality of user information.

4) Design Details

Class Diagrams



Descriptions of Classes and Models

- User
  - A user object is created every time a new user signs up for our platform.
  - In our database, each user is assigned a unique UID.
  - A user signs up with their email and password, after which they are prompted to select a username.
  - When customizing their profile, the user has options to enter their gender, attach a profile picture, and add a biography. They will also enter their portfolio (more on this in the portfolio section).
  - The user also has a friends list, a counter showcasing the number of their friends, a list of their posts, a stock watchlist, notifications, and a chat inbox.
  - Users can also be deleted if a user no longer wants to be associated with the platform.

- Friend
  - A friend is similar to the User object, except each friend is assigned to a tier (trust level).
  - This tier system is used to decide the extent to a which a user's portfolio information is shared with a friend.

- Post
  - A post object is created every time a user creates a post.
  - There are counters assigned to count and display the number of likes, shares, and comments.
  - The author of the post is a User, whose username is displayed above the post.
  - Users are given the choice to add a caption to the post, as well as choose any other additional content, such as images, Stocks and Polls to enrich their post.
  - If a post is edited after it was originally posted, a small text field saying *'Edited'* will appear below the caption.
  - Post authors can post, edit, and delete their posts. Other users can like, comment, share, and report a post.

- Comment
  - A comment object is created every time a user comments.
  - The author of a comment is a User.
  - The timestamp when the comment was created and the number of likes on the comment are also stored within this class.
  - An author can post a comment and delete a comment. Other users can like and report a comment.

- Poll
    - A poll is one of the key features of our product, as it allows for greater user engagement and more thought-provoking discussions.
    - Polls are time-bound, and the results of the poll are finalized once the timer runs out.
    - Users can vote whether they think a stock is bullish or bearish, and these votes are used to decide the winner of the poll.

- Notification
    - Notifications are simple ways to keep the user engaged and up to date.
    - A user can choose to enable or disable notifications.
    - A notification contains a simple string, alongside a timestamp establishing when the notification was sent to the user.

- Chat
    - A chat object is created when a user starts a new chat.
    - Chats can be either one-to-one or group chats.
    - Chats contain a list of messages.
    - They can be both created and deleted.

- Message
    - A message object is created when a user sends a message. It is the composition of a chat object.
    - The message contains a timestamp indicating when the message was sent, as well as the content, which can range between a String, an image, a Stock and a Post.
    - Messages can be deleted by their authors.

- Portfolio
    - A portfolio object is created by a user once they enter their stock portfolio.
    - A user can choose whether to make the details of their portfolio public or private. If public, the extent to which the portfolio is showcased to their friends is dependent on the friend's aforementioned trust level.
    - The portfolio showcases the present date, as well as cumulative earnings.
    - Finally, it contains the list of Stocks that are part of the portfolio.

- Stock
    - A stock object contains the details about a relevant stock.
    - It contains the ticker, a string, as well as financial information (closing price,

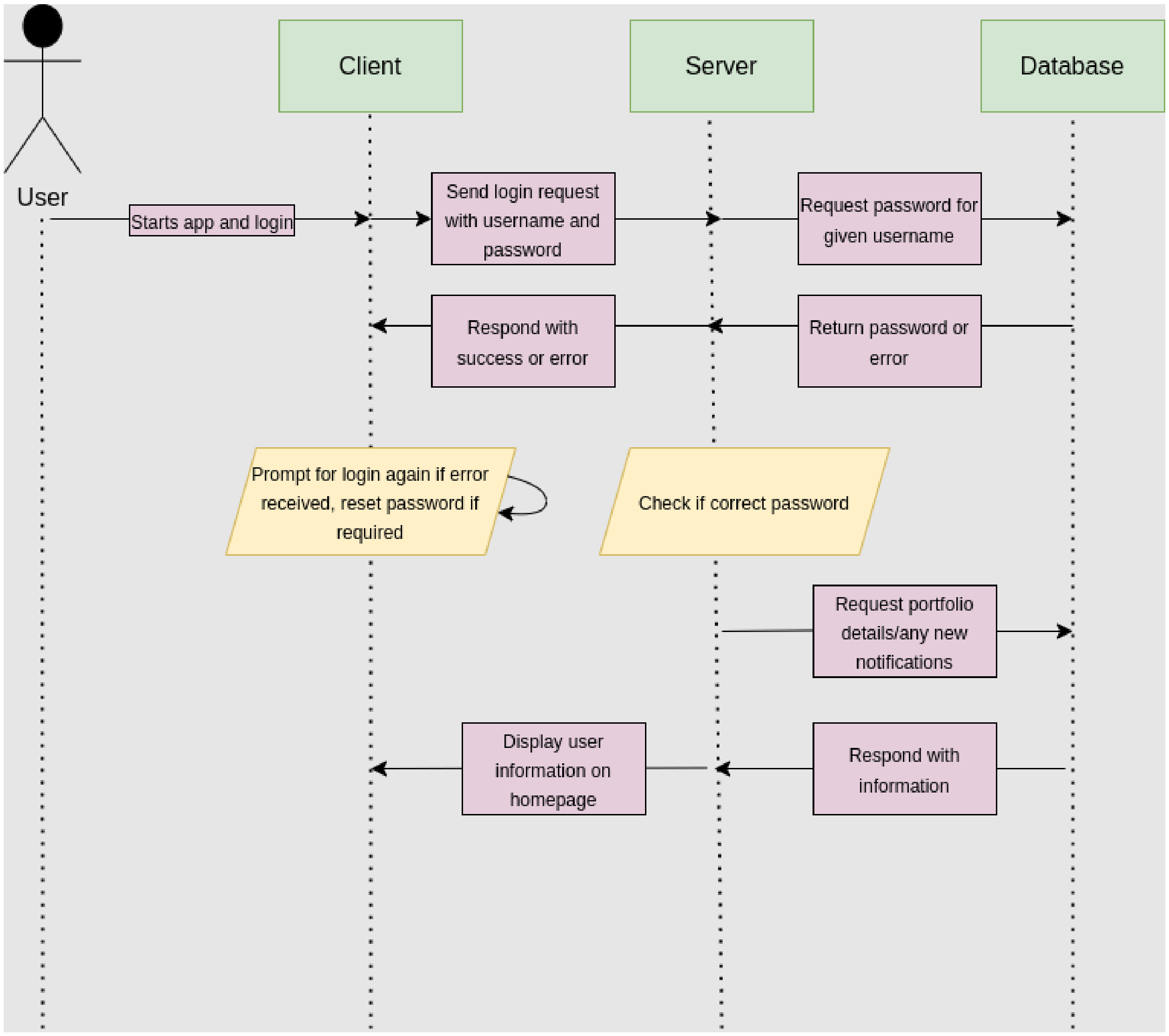
opening price, volume, average volume, day high and low, and week high and low) all stored as integer values.

  - The chart containing the stock performance is displayed using the Yahoo finance API.
  - Stocks can be added to a user’s watchlist.

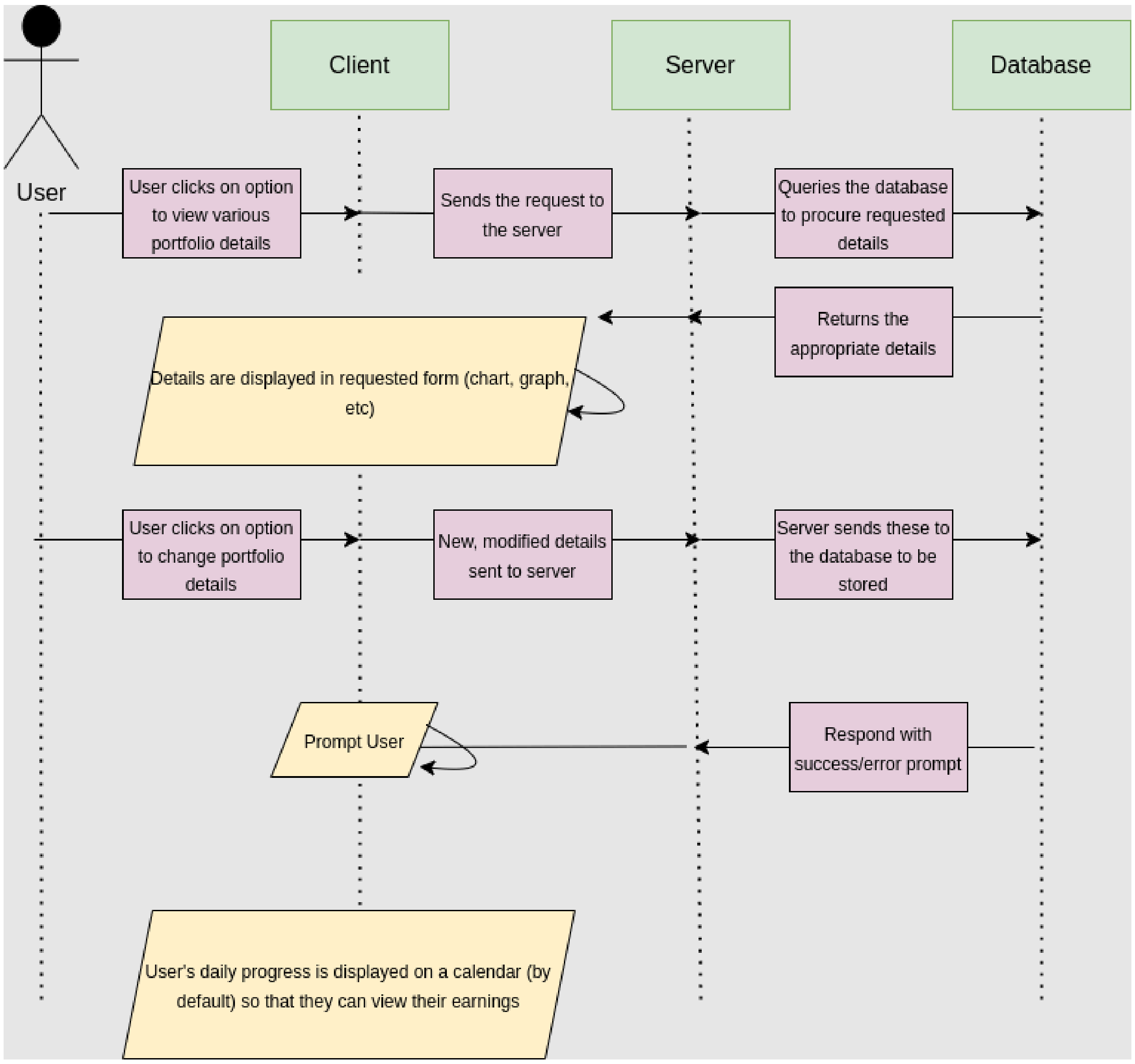
Sequence of Events Overview

The sequence diagram below shows the typical interaction among clients, server and database. The sequence is initiated by a user starting the web app. As the user logs in, the client sends a request to the server. The server handles the request, then sends a query to the database and receives data from the database. After logging in, the client further sends a request to the server for other actions, like managing the profile data, creating posts, creating polls, and interacting with other user’s posts. To respond to these requests, the server sends queries to the database to acquire data from the database. The database responds with updated data after receiving update requests. With the requested data from the database, the server returns the results to the client.

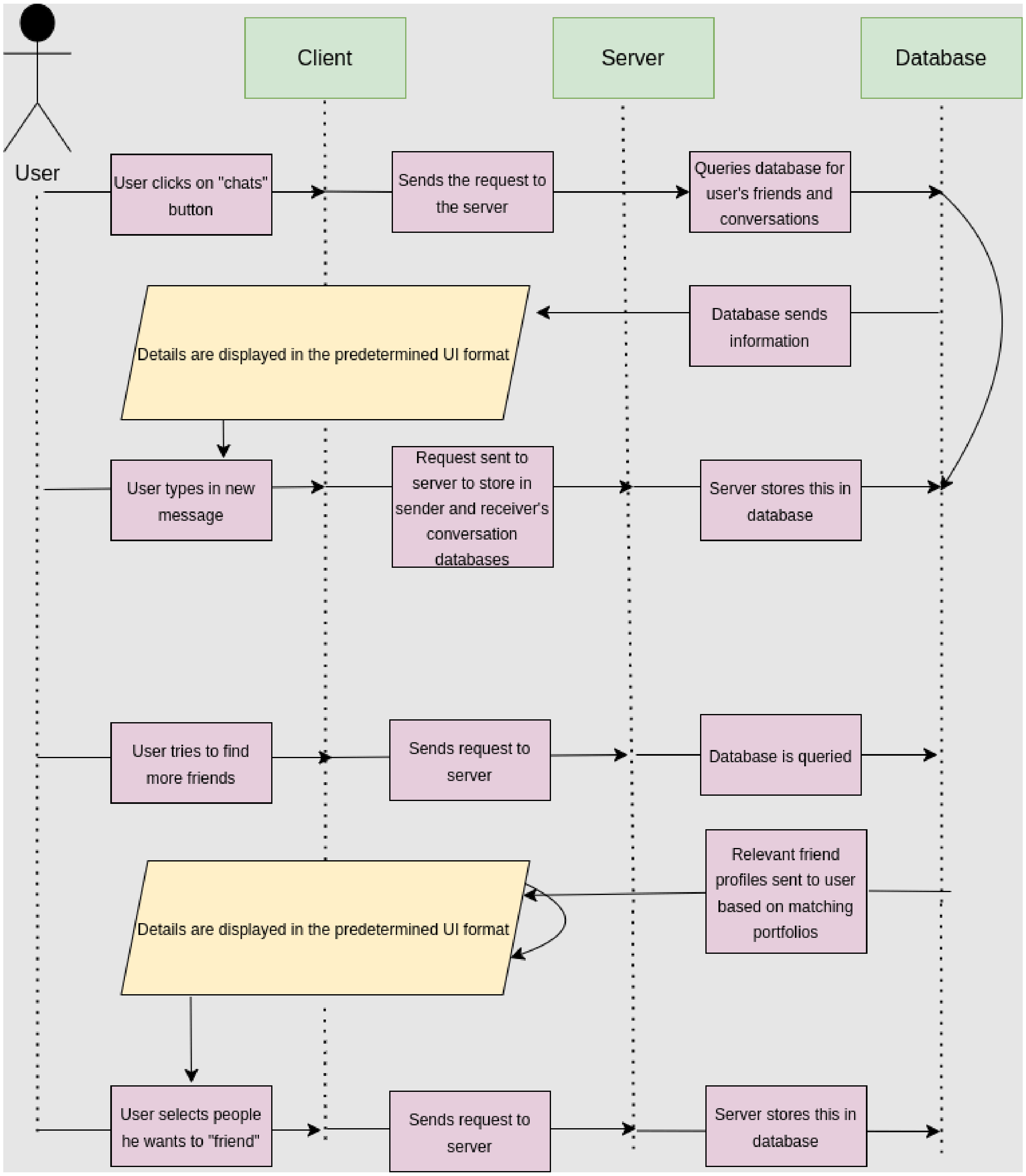
1. Sequence of events when users login


User
Client
Server
Database
Starts app and login
Send login request with username and password
Request password for given username
Respond with success or error
Return password or error
Prompt for login again if error received, reset password if required
Check if correct password
Request portfolio details/any new notifications
Display user information on homepage
Respond with information

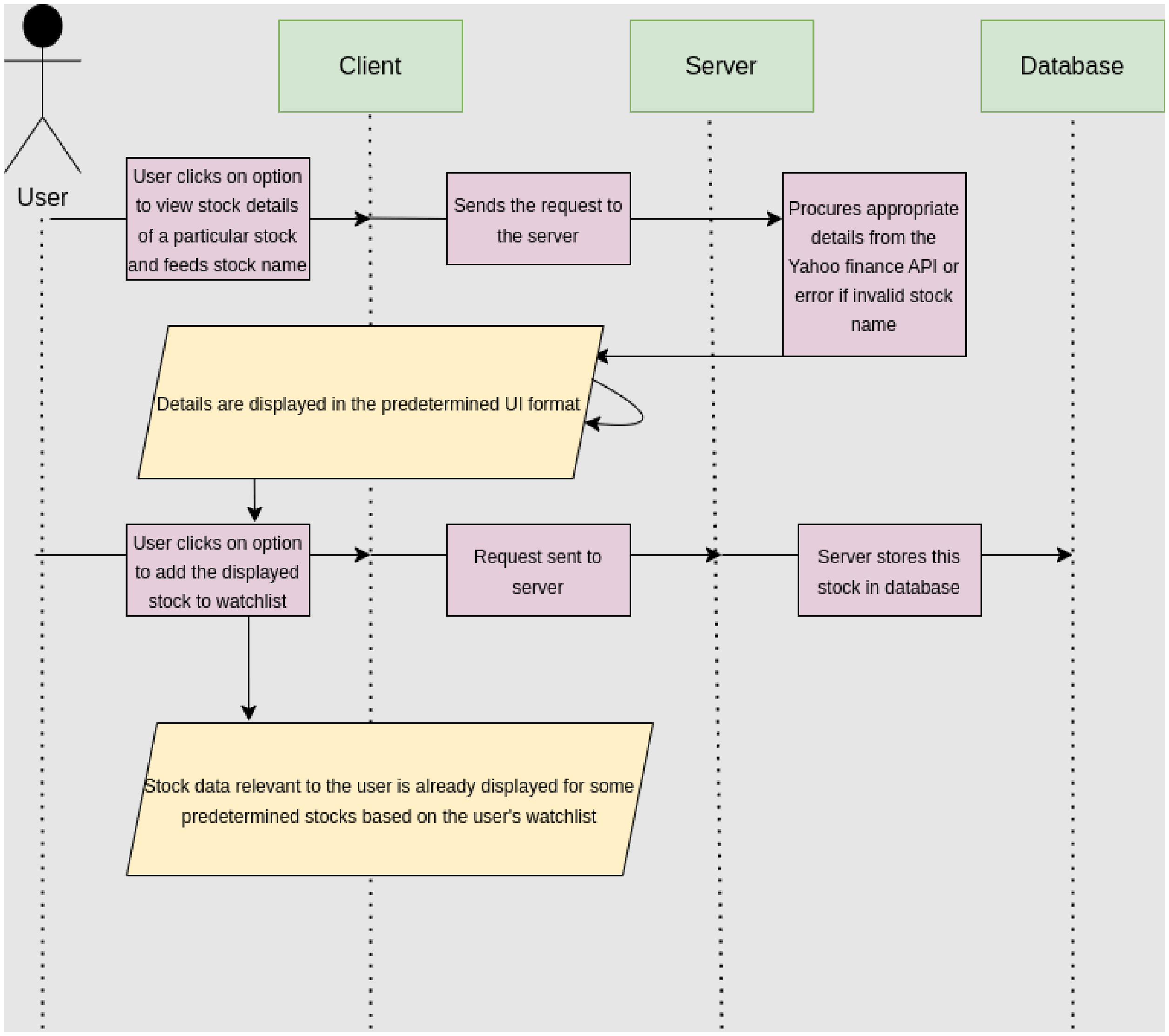
2. Sequence of events when user interacts with their portfolio


Client
Server
Database
User
User clicks on option to view various portfolio details
Sends the request to the server
Queries the database to procure requested details
Returns the appropriate details
Details are displayed in requested form (chart, graph, etc)
User clicks on option to change portfolio details
New, modified details sent to server
Server sends these to the database to be stored
Prompt User
Respond with success/error prompt
User's daily progress is displayed on a calendar (by default) so that they can view their earnings

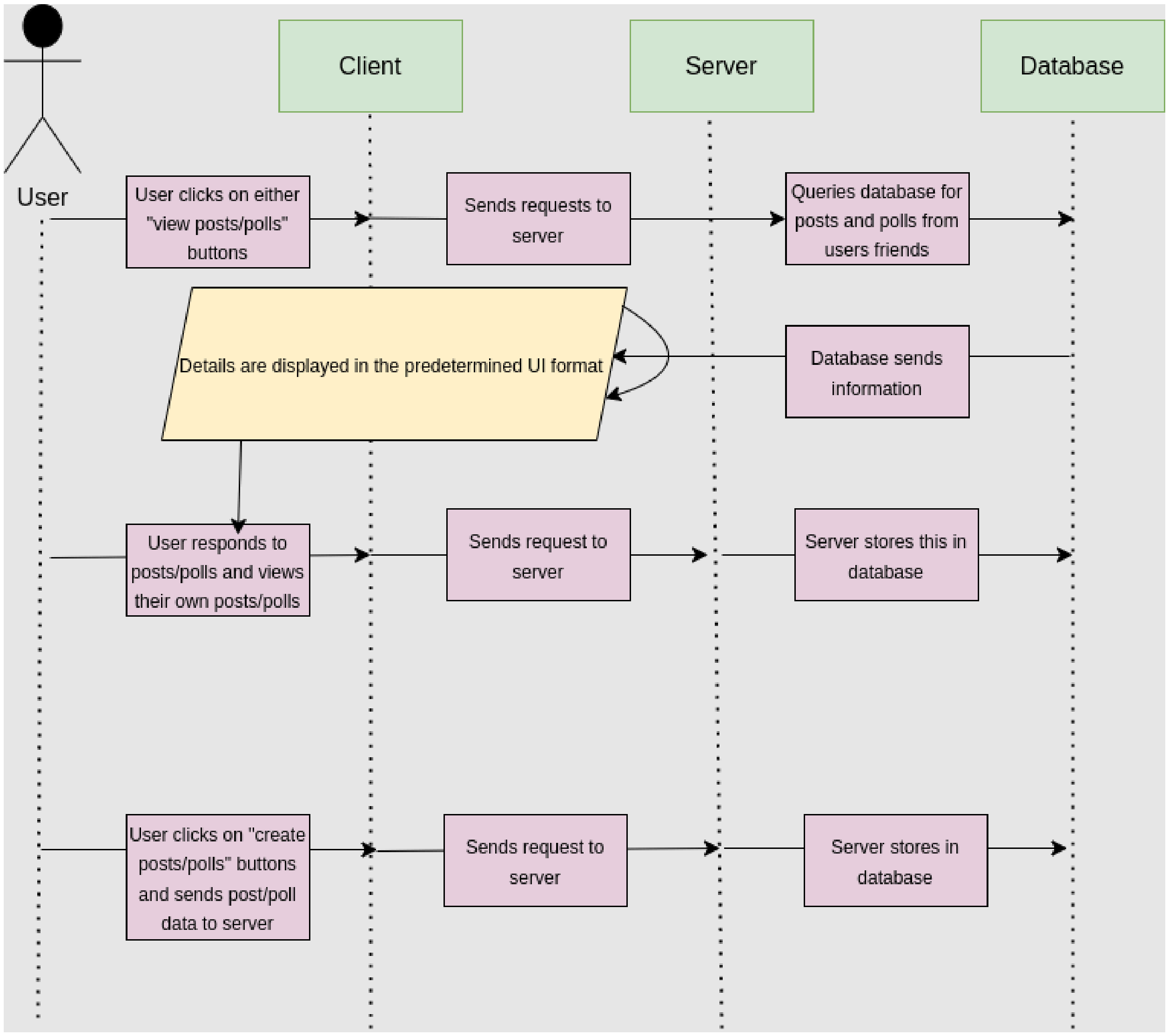
3. Sequence of events when user engages in social networking (example here involves the process of “friending” as well as communication with a friend)


User
Client
Server
Database
User clicks on "chats" button
Sends the request to the server
Queries database for user's friends and conversations
Database sends information
Details are displayed in the predetermined UI format
User types in new message
Request sent to server to store in sender and receiver's conversation databases
Server stores this in database
User tries to find more friends
Sends request to server
Database is queried
Relevant friend profiles sent to user based on matching portfolios
Details are displayed in the predetermined UI format
User selects people he wants to "friend"
Sends request to server
Server stores this in database

4. Sequence of events when user views various stocks


User
Client
Server
Database
User clicks on option to view stock details of a particular stock and feeds stock name
Sends the request to the server
Procures appropriate details from the Yahoo finance API or error if invalid stock name
Details are displayed in the predetermined UI format
User clicks on option to add the displayed stock to watchlist
Request sent to server
Server stores this stock in database
Stock data relevant to the user is already displayed for some predetermined stocks based on the user's watchlist

5. Sequence of events when user creates or responds to any posts or polls


Client
Server
Database
User
User clicks on either "view posts/polls" buttons
Sends requests to server
Queries database for posts and polls from users friends
Details are displayed in the predetermined UI format
Database sends information
User responds to posts/polls and views their own posts/polls
Sends request to server
Server stores this in database
User clicks on "create posts/polls" buttons and sends post/poll data to server
Sends request to server
Server stores in database