WEALTHWISE FINANCIAL ADVISING PLATFORM

Design Document

Draix Wyatt, John Akujobi, Norman Nguyen, Sawyer Theis

SE 305 – Foundations of Software Engineering

Dr. Zainab Albujasim

Nov 18th, 2024

Table of Contents

I. Design Document Introduction

A. Purpose of the Design Document

This document defines all design specifications for the software system Arqer is creating for WealthWise. This document shall be completed before implementation begins. It shall serve as an internal reference guide for the software developers of Arqer while developing this software. The design specifications were created by the Arqer team in light of the requirements listed in the Requirements Document and various meetings with the client.

B. Major Problems

1. Data Security and Privacy Risks

WealthWise deals with sensitive financial information and plans. This makes it a tempting target for cyberattacks. A data breach will lead to lawsuits, significant financial penalties, identity endangerment, compromised user safety, and loss of trust in our system.

To mitigate this risk, Arqer shall enforce strong encryption protocols and comply with data protection regulations like GDPR and PCI DSS (Payment Card Industry Data Security Standard) and should use multi-factor authentication where feasible.

2. Implementation of Video Conferencing

To implement features such as advisor meetings, webinars, and other events, the website requires a video conferencing gateway. This will be external to our system and will be a challenge to implement. Additionally, disruptions to any of these components will cause downtime.

To implement this, Arqer shall implement flexible APIs to third-party video conferencing services (ex. Google Meets, Zoom). Arqer should, where feasible, build redundancy into the system by integrating dependable services via alternative third-party services. Any implemented services and APIs implemented shall be ones of good repute within the given industry.

3. Payment

In order to implement payment-based subscriptions, the website will require payment gateways. This will be a challenge as said gateways will be outside our system and will carry sensitive data.

To implement this, Arqer shall integrate payment gateways, financial data providers, and other APIs into the website and resolve compatibility and dependency issues with said services. services implemented should include PayPal and credit cards (visa, Master Card, etc.).

4. Managing Real-Time Data Updates

Features like market news and stock data require real-time updates and can be resource intensive. Other systems, such as advisor scheduling and event scheduled will also require real-time data syncing.

To create the best user experience and have any data as up to date as possible. Arqer shall cache data as necessary and feasible, redirect the users to the actual data source where applicable, and implement database management systems such as the ACID (atomic, consistency, isolation, and durability) model.

C. Project Goals

1. Centralize WealthWise's Services

The primary goal of this website is to bring all of WeathWise's services to one central website. Arqer shall bring all listed services (see Software Requirements section) to one central website.

2. Providing Accessibility to Certified Advisors

One major service, WeathWise, provides access to conferences with certified financial advisors. Arqer shall allow users to hold video calls with certified financial advisors. Users shall be able to schedule a video meeting with a financial advisor through an event scheduler that allows for meeting scheduling, editing, and cancellation.

3. Provide Advanced Financial Tools (Calculators)

To allow users to make sound financial decisions, Arqer shall implement a variety of calculators including a loan interest calculator, a mortgage calculator, a budgeting tool (by subscription), a retirement planning calculator, and an insurance coverage calculator.

4. Education and News Resources

To allow users to stay up to date on financial education and news, Arqer shall implement a learning hub with resources such as news articles, financial tutorials, webinars, and other educational resources as necessary. External resources shall be accessed via relevant APIs and gateways.

5. Secure Payment

To allow system monetization and subscriptions, Arqer shall implement systems to allow the user to make payments to WealthWise. Said payments shall be made secure via reputable third-party services such as PayPal and its 128-bit SSL protocol.

6. System Security Monitoring

Site administrators shall be able to monitor the system and log events. Significant system events (logins, data transfers, API requests) shall be logged. An intruder detection system shall be used to identify unauthorized access to or tampering with the system.

7. Intuitive UI/UX

The website should be designed such that the target user shall be able to navigate to every feature and tool without significant delay or loss of user experience. This shall be achieved through intuitive web design that clearly indicates where tools are located throughout the site.

D. Proposed System Overview

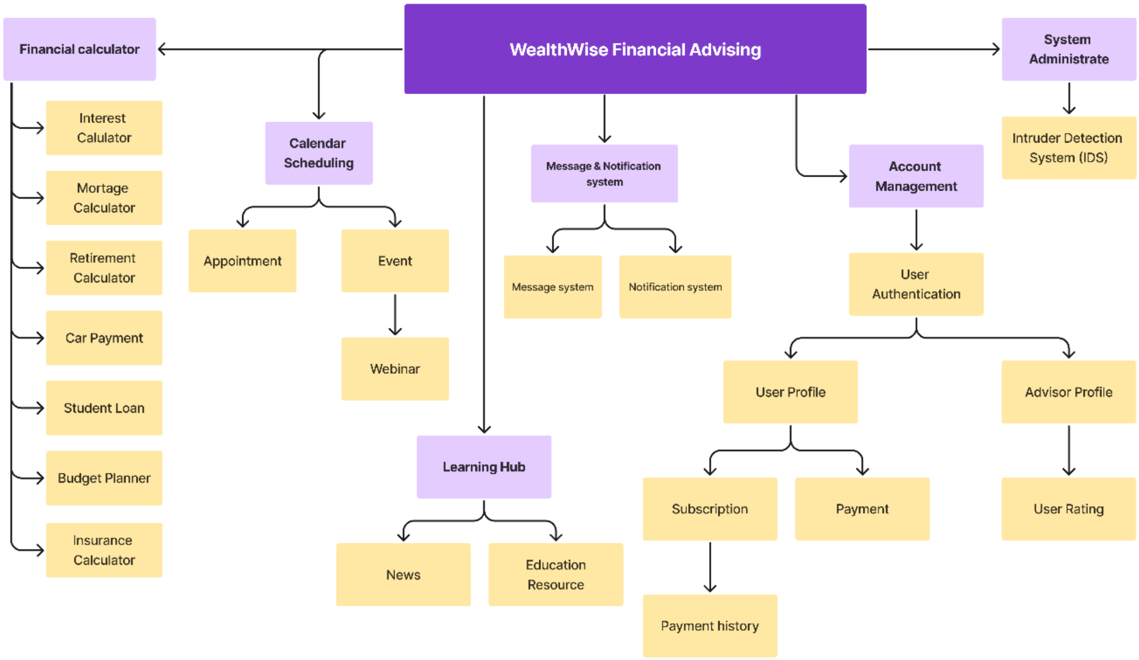


Figure 1. WealthWise Proposed System

II. Hardware and Software

The next section discusses the hardware requirements and internal and external software stacks needed for the WealthWise Financial Advising platform. It enumerates the minimum specifications required to efficiently host and run the application, elaborating on the chosen software components to achieve the required functionality.

A. Hardware

The hardware requirements enumerate specifications of the server, which hosts the environment, and also outline client-side requirements for hassle-free access and operation.

1. Server

In order to support the WealthWise platform, the server should have at least:

- 4 GB RAM: It should be sufficient to handle the number of concurrent users, caching, and backend processing for an MVP. The RAM should be increased if the user load scales up significantly.
- 2 CPUs: Two CPU cores allow for parallel processing and can handle moderate levels of simultaneous requests. This is important for running background tasks, managing asynchronous calls, and improving response times.
- Storage: At least 50 GB of SSD storage for storing application data, database backups, logs, and assets. Note that SSD storage is recommended for better data access speed and application performance.
- Operating System: Linux-compatible (preferably Ubuntu) for easy deployment, compatibility with contemporary cloud infrastructure, and a large community.

2. Client-side Requirement

Make sure the application will work for any user by supporting all modern browsers:

- Supported Browsers: Google Chrome, Firefox, Safari, and Microsoft Edge.
- Minimum Browser Version: Support the last three major versions of each browser, for security and functionality.
- Device Compatibility: The platform should be accessible on desktop, tablet, and mobile devices, and have responsive design for different screen sizes.

B. Internal Software Stack

The internal software stack includes all core technologies required to build, run, and maintain the application: from the front end and back end to the database and deployment tools.

1. Front-end

The front-end of the application itself is designed to be very interactive and user-friendly.

- Frameworks: Next.js with App Router
    - Next.js with App Router: Selected for its support of modern React features, such as Server Components and Streaming, providing a performance edge and better SEO for server-rendered components. Next.js offers rich component-based architecture that enables easy UI reuse, robust routing, and a growing ecosystem with libraries for handling forms, UI components, and more.
- Alternative: Vue.js

  - Known for its lower learning curve and reactivity, Vue can be a good choice for smaller projects but lacks some of the broader ecosystem support and integrations that Next.js offers for complex applications.
- Styling: CSS frameworks or libraries like Tailwind CSS or Bootstrap to make responsive design easier and to maintain consistency in styling throughout the application.
- State Management: Redux for global state management, particularly useful in handling complex data interactions, such as booking and financial calculations. For Next.js, useContext can also be utilized for simpler state management within the component tree.

2. Back-end

Its backend is designed to deal with business logic, process requests, and interact with the database.

- Django or FastApi Framework:
  - FastAPI: It's the choice since it's asynchronous and very lightweight, which fits well for a high-CPU application with APIs. It also provides support for JSON responses, automatic data validation, and interactive API documentation.
  - Django: Also, a strong contender, with a full web framework, ORM, admin panel, and built-in user auth, though it's synchronous and heavier than FastAPI.
- Data Serialization: Use Pydantic models with FastAPI for the validation of data input and output schemas to ensure secure and consistent data handling.

- Asynchronous Processing: Look into using Celery or BackgroundTasks with FastAPI to run tasks asynchronously, like sending emails or processing payments.

3. Database

Database selection is used to support safe data storage and effective data retrieval.

- Main Database: PostgreSQL
    - PostgreSQL has the advantage of being reliable with strong ACID compliance, with strong support for complex querying. It can be applied to user profiles, transactions, and huge financial datasets.
- Alternative: SQLite
    - SQLite can be used as a lightweight replacement for development and testing, but it lacks a few advanced features to deal with high concurrency, so it's not recommended for production.

4. Deployment

Deployment involves managing the application in a live environment accessible by users.

- Main Deployment: Namecheap with C-Panel and Cloudflare
    - Namecheap: Cheap hosting with C-Panel, simplifies website management and supports Python applications with custom configurations.
    - Cloudflare: Acts as a content delivery network (CDN) while simultaneously providing DDoS protection, SSL/TLS encryption, and caching in order to provide better security and load times.
- Alternatives: Heroku or Vercel

  - Heroku: Easy deployment of small apps, with little configuration. It's great for rapid prototyping and MVPs.
  - Vercel: Optimized for frontend deployments and serverless functions but might require extra setup for Python-based backends.

C. External Software Stack

The external software stack includes third-party APIs and services that have more specialized features, like authentication, payment processing, or analytics.

1. Authentication and Security

- **Auth0 API**:
  - Role: Managing user authentication and authorization, including password management, multi-factor authentication (MFA), and role-based access control.
  - Technical Integration: Auth0's SDK will be integrated into the backend to manage user sessions in a secure manner; it provides OAuth2, and OpenID Connect for social logins, ensuring secure token-based authentication
- **Cloudflare CDN**:
  - Role: Speeding up loads through distributed caching; provides security features like DDoS protection and SSL/TLS encryption.
  - Technical Integration: Traffic is routed through Cloudflare to reduce the load on servers and increase content delivery efficiency, mainly for static assets.

2. Payment Processing

- **Stripe API/PayPal API**:

  - Role: Payment, subscription, and invoicing management.
  - Technical Integration: APIs are implemented for the processing of payments at both frontend and backend level. Stripe and PayPal offer secure gateways of payment that integrate on webhooks and update in real time the status of payments and subscriptions.

3. Calendar and Scheduling

- **Google Calendar API/Outlook Calendar API**:
  - Role: Allow users to view, schedule, and manage appointments with advisers.
  - Technical Integration: The APIs integrate appointments with the users' personal calendars and offer availability in real time. Integration with OAuth2 ensures that access to users' calendar data remains secure.

4. Video Conferencing

- **Zoom API / Google Meet API**:
  - Role: It enables virtual consultations between clients and advisors.
  - Technical Integration: includes the integration of OAuth, which provides secure access in generating meeting links on demand. Session details and reminders are stored in the user profile.

5. Email and Notifications

- **SendGrid API**:
  - Role: Handles transactional emails, such as password reset, subscription reminders, and notifications.

    - Technical Integration: SendGrid's REST API will be used to send automated emails from the backend with templating for personalized messages.

- **Twilio API**:
    - Role: Sends SMS notifications for important events or reminders.
    - Technical Integration: The API of Twilio is called from the backend for sending SMS notifications on a programmable schedule for reminders and updates.

6. Financial Data Providers

- **Alpha Vantage API / Yahoo Finance API**:
    - Role: Provides real-time and historical financial data, which are critical to obtain stock insights, news updates, and financial calculators.
    - Technical Integration: Financial Data APIs that bring live market data to show stock prices, news articles, and trends directly on the platform.

7. Analytics and Performance Monitoring

- **Google Analytics API**:
    - Role: To monitor user engagement, site traffic, and usage trends.
    - Technical Integration: The Google Analytics JavaScript tag was implemented in the frontend to track user behavior, page views, and events.
- **Lighthouse API**:
    - Role: Analyzes performance, accessibility, SEO, and other metrics of the site.
    - Technical Integration: The Lighthouse API can be run periodically or integrated with CI/CD pipelines to ensure that site optimizations are in place and monitor changes in performance.

III. Design Priority

A. Alternatives

1. Criteria for Choices

- Free or low cost
- Easy for the team to learn
- Compatibility with project requirements
- Has libraries and community support

2. Tech Stack

The tech stack was selected based on the team's experience, ease of learning, availability of libraries and community support, and compatibility with project requirements. Here's a comparison of primary choices and viable alternatives:

Component	**Primary Choice**	**Alternative Choices**	**Language**	**Reasoning**
Front-end	Next.js	Vue.js	JavaScript	Next.js gives us extensive library support, allows us to resuse components, and improves SEO through server rendering. This is good for our complex application with dynamic content.
Back-end	FastAPI	Django	Python	FastAPI is lightweight, fast, and async-capable, making it suitable for a scalable API-focused app.
Database	PostgreSQL	SQLite	SQL	PostgreSQL provides reliability and scalability, whereas SQLite could be simpler but less robust.

Deployment	Namecheap + C-Panel + Cloudflare	Heroku, Vercel	Not Applicable	Namecheap with C-Panel and Cloudflare is cost-effective for small deployments. Heroku/Vercel offers auto-deployment but with potential cost and configuration limitations.

3. Front-end

- **Next.js with App Router:** Chosen for its comprehensive support for React's latest features, including Server Components, which allow for faster load times and improved SEO. The App Router further supports modular routing and better performance for dynamic content, essential for our application's interactivity.
- **React.js**: Known for its robust component-based structure, React is widely used and has a wealth of supporting tools, but it lacks some of Next.js's advanced routing and server-side capabilities.
- **Vue.js**: Offers a similar component-based structure with a lower learning curve, though it has a smaller ecosystem and may lack some features compared to Next.js, especially for larger applications.

Final Choice: Next.js with App Router, due to its advanced routing, performance optimization, and server-side rendering capabilities.

4. Back-end

- **FastAPI**: FastAPI is asynchronous, meaning it can handle more requests simultaneously, making it efficient for a high-demand API-driven application. It's designed for building APIs quickly and integrates well with Python's data-handling libraries.

- **Django**: Django is a robust, fully-featured web framework that includes an ORM, authentication, and admin panel out-of-the-box. However, it's synchronous and may be less performant for an API-focused application.

Final choice: FastAPI, due to its speed, asynchronous capabilities, and suitability for API development.

5. Database

- **PostgreSQL**: Offers advanced features, robust data integrity, scalability, and support for complex queries. Well-suited for a production environment that handles a substantial amount of financial and user data.
- **SQLite**: Simple and lightweight but lacks advanced functionality for complex querying and scalability.

Final Choice: **PostgreSQL**, due to reliability and support for complex transactions.

6. Deployment

- Namecheap + C-Panel + Cloudflare: Provides affordable hosting with a manageable interface. Cloudflare adds security features like DDoS protection and CDN. Great for a small project with budget constraints.
- **Heroku**: Cloud-based platform-as-a-service (PaaS) that simplifies deployment but has limitations on free-tier services.
- **Vercel**: Excellent for frontend deployments and allows serverless functions, but not as robust for Python backend needs.

Final Choice: **Namecheap + C-Panel + Cloudflare**, due to cost-effectiveness and ease of setup.

B. Design Priority Table

The following table highlights the priority levels for major system components based on their importance and impact on the core functionality.

Component	Priority	Reason for Priority
User Registration	High	Essential for access control and personalized user services.
Advisory Booking	High	Core features to enable user engagement with advisors.
Financial Calculators	Medium	Adds value and improves user retention through utility.
Payment System	High	Necessary for managing subscriptions and monetization.
Educational Resources	Medium	Provides long-term value but is non-essential for MVP.
Event Management	Medium	Supports user engagement but secondary to core services.

C. Development and Execution Environment

- **Development Environment**: The development will take place on local machines equipped with Python, Node.js, PostgreSQL, and VS Code. Using these local setups allows flexibility, ease of debugging, and efficient testing before deployment.
- **Execution Environment**: The final app will be deployed on Namecheap hosting with C-Panel and Cloudflare for added performance and security. This setup supports our budget, offers enough resources for initial traffic, and integrates well with other tools.

D. Programming Tools

This section lists the tools used for different aspects of development, testing, and code management.

- **Front-end Tools:**
    - **VS Code**: IDE for frontend and backend development, supporting multiple languages.
    - **Prettier**: Code formatter to ensure consistent style across team contributions.
    - **ESLint**: Linter to catch syntax and style errors in JavaScript code.
    - **React DevTools**: Browser extension for inspecting React component trees.
- **Back-end Tools:**
    - **PyTest**: Framework for writing and running automated tests on the backend.
    - **FastAPI Framework**: Lightweight and asynchronous, ideal for API development.
    - **Swagger**: Integrated with FastAPI for API documentation, making it easier to test endpoints.
- **Database Tools**:
    - **pgAdmin**: GUI for managing PostgreSQL databases, useful for testing queries and visualizing tables.
    - **SQLAlchemy**: ORM for interfacing with the PostgreSQL database in Python, ensuring database interactions are streamlined and secure.
- **Version Control:**
    - **Git**: For version control, with collaborative workflows managed on **GitHub**.
- **Design Tools:**

  - **Figma**: Collaborative design tool used by UI/UX team members to create wireframes and mockups.

E. Naming and Coding Standards

To maintain consistency, a set of naming conventions and coding standards is essential:

- **Naming Conventions**: Use **CamelCase** for variables and function names and **PascalCase** for class names.
- **Code Style**: Adhere to **PEP 8** standards for Python code and **ESLint** rules for JavaScript to ensure consistency and readability.
- **Documentation**: Use comments to clarify the purpose of functions, classes, and complex logic. All team members are encouraged to document their code for better maintainability.

F. Error Handling

Implementing error handling across both frontend and backend ensures a smoother user experience:

- **Back-end:**
  - Use **try-except blocks** to catch and log errors at critical points in the backend.
  - Implement logging to capture error details for debugging and monitoring in production.
  - Include validation checks for user inputs to prevent database errors and security issues.
- **Front-end:**

  - Use **error boundaries** in React to catch UI errors and display fallback interfaces.
  - Provide user-friendly error messages for input validation, API call failures, and connectivity issues.

G. Fault Tolerance

Fault tolerance is critical for reliable user experience. Key strategies include:

- **Redundant Servers**: Design the deployment to support failover if a server becomes unavailable, minimizing downtime.
- **Database Backups**: Set up automatic backups for the PostgreSQL database at regular intervals, allowing data restoration if an error or failure occurs.
- **Load Balancing**: Although not immediately necessary, load balancing can be implemented later if usage demands scale up significantly.

H. Design Constraints

The following constraints shape the project's design and development process:

- **Timeline**: The project must be completed within a three-month period, necessitating rapid development cycles and efficient prioritization.
- **Team Skillset**: The team has limited experience with JavaScript and web development frameworks, making simplicity a high priority.
- **Budget**: Restricted to open-source or free tools to keep costs low, with paid options considered only if essential for MVP success.

I. Hardware and Software Constraints

- **Hardware**: Hosted on Namecheap servers, which have limited resources compared to major cloud platforms. This limits scalability initially but is sufficient for an MVP.
- **Software**: Preference for free or open-source tools to align with budget restrictions. The application stack must run on Namecheap's infrastructure and be compatible with C-Panel configurations.

IV. Data Flow Diagram (DFD), Module and Data Dictionary

A. Data Flow Diagram (DFD)

Level 0:

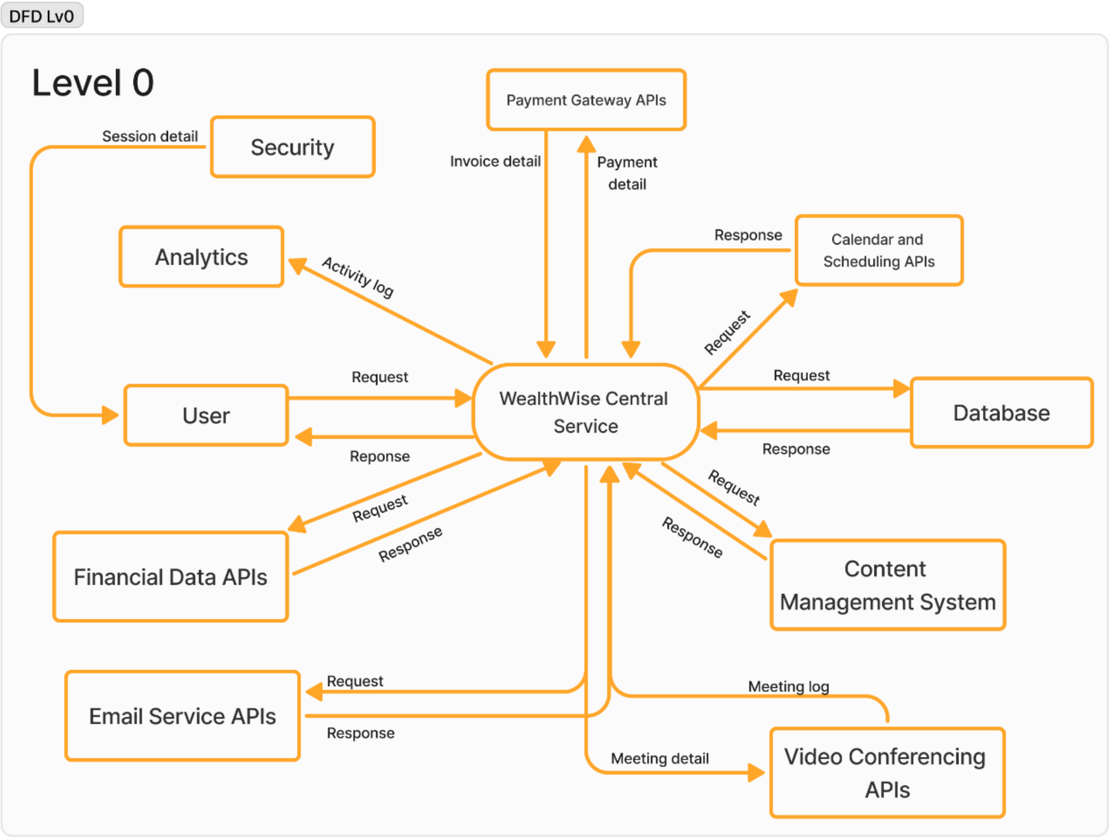


Figure 2. Level 0 Data Flow Diagram

Level 1:

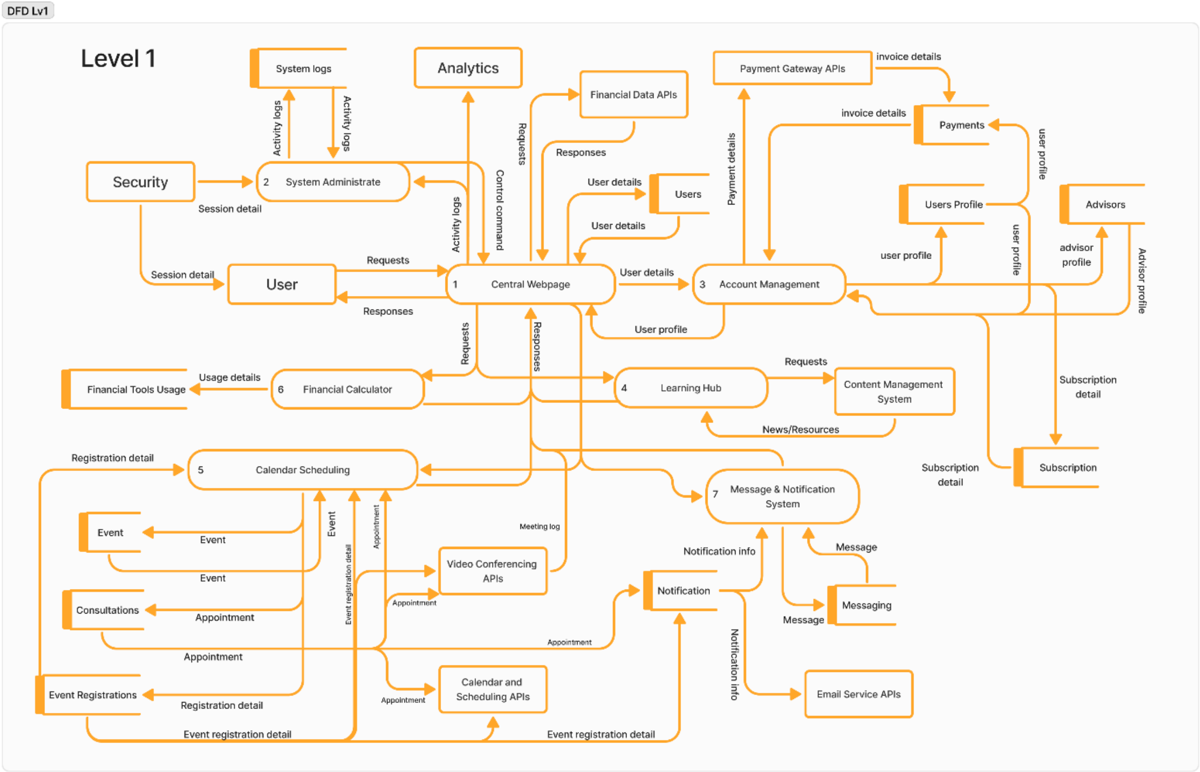


Figure 3. Level 1 Data Flow Diagram

Level 2:

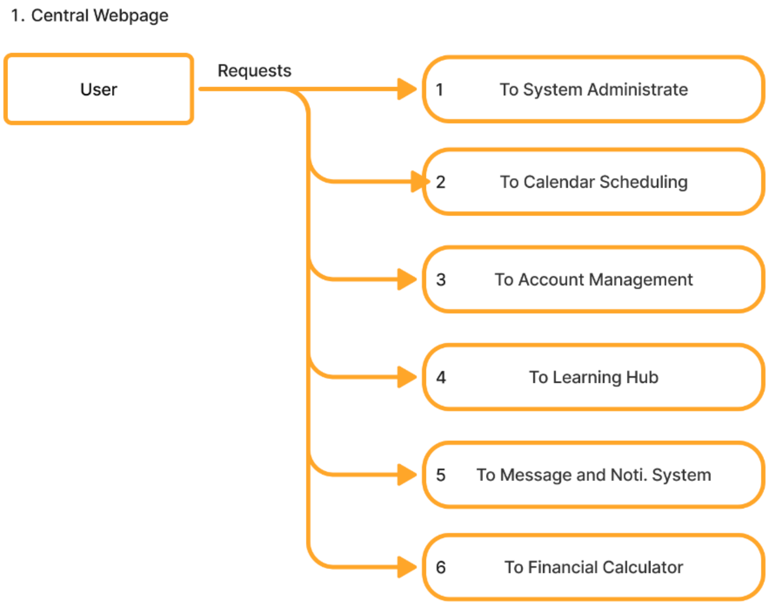


Figure 4. DFD Level 2 - Central webpage

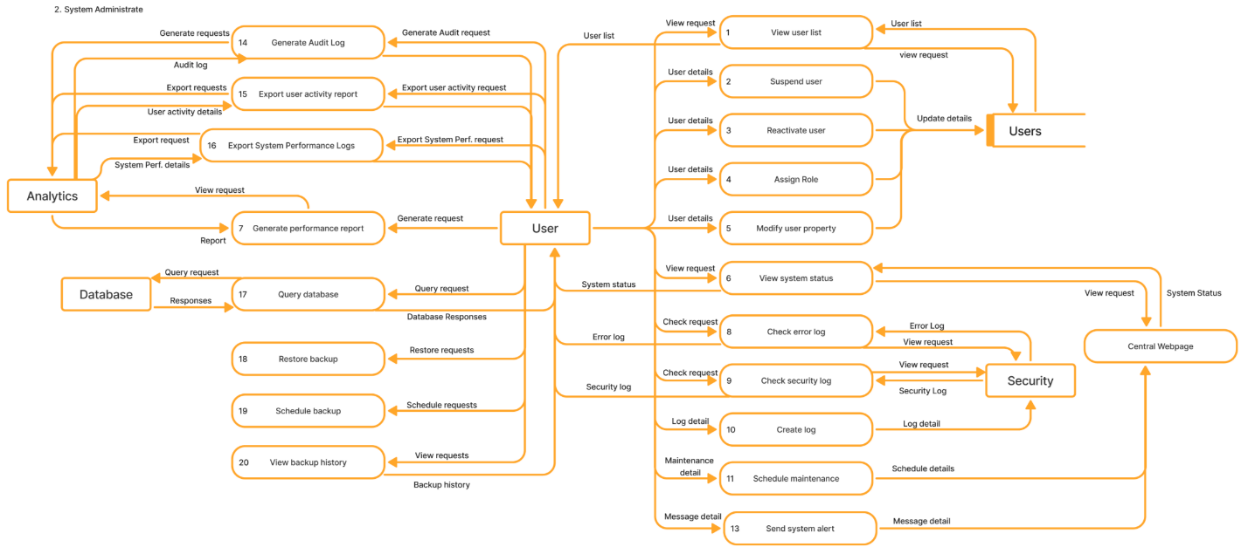


Figure 5. DFD Level 2 - System Administrate

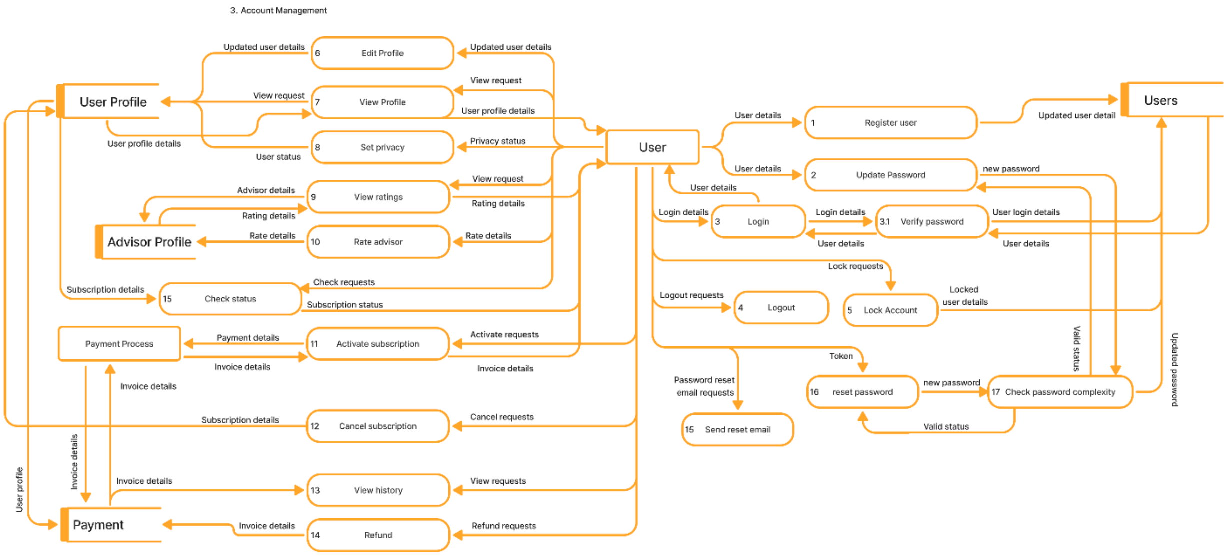


Figure 6. DFD Level 2 - Account management

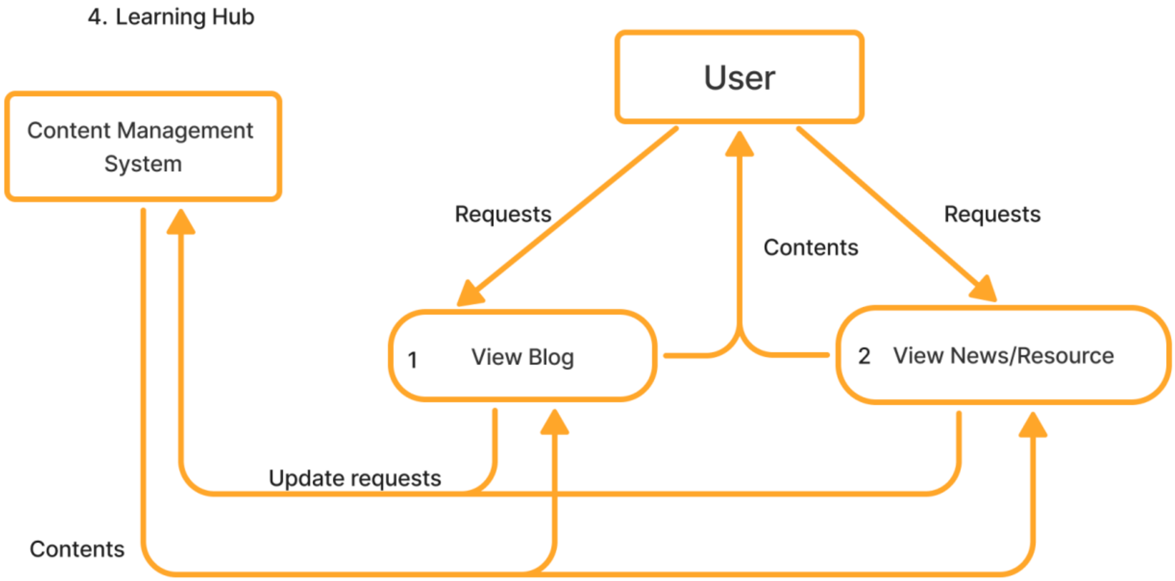


Figure 7. DFD Level 2 - Learning hub

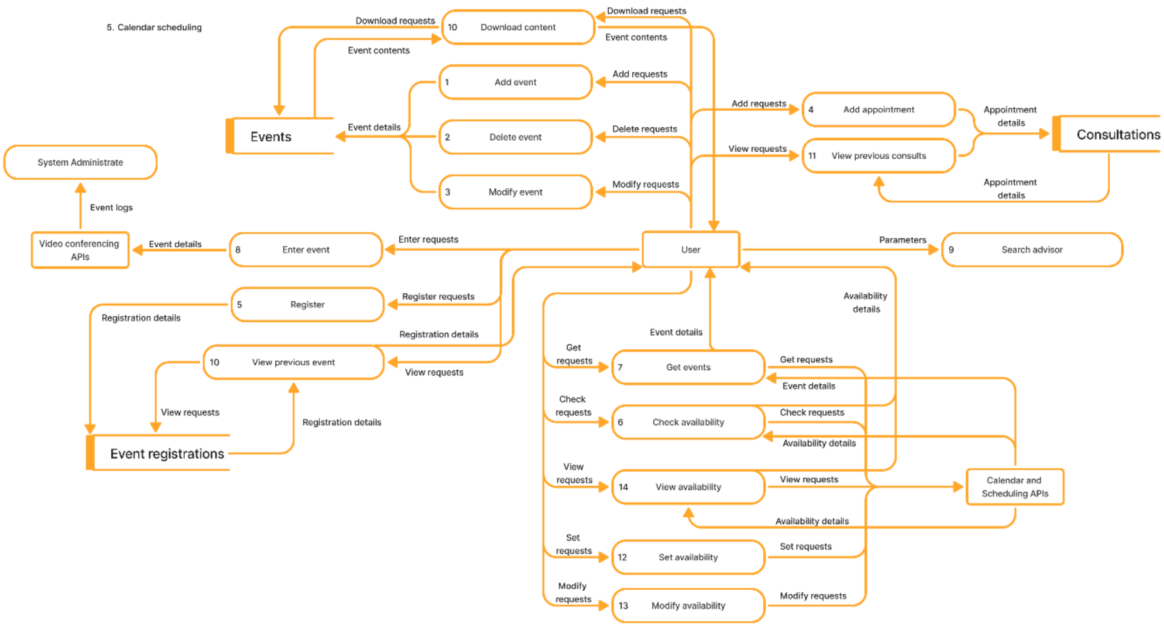


Figure 8. DFD Level 2 - Calendar scheduling

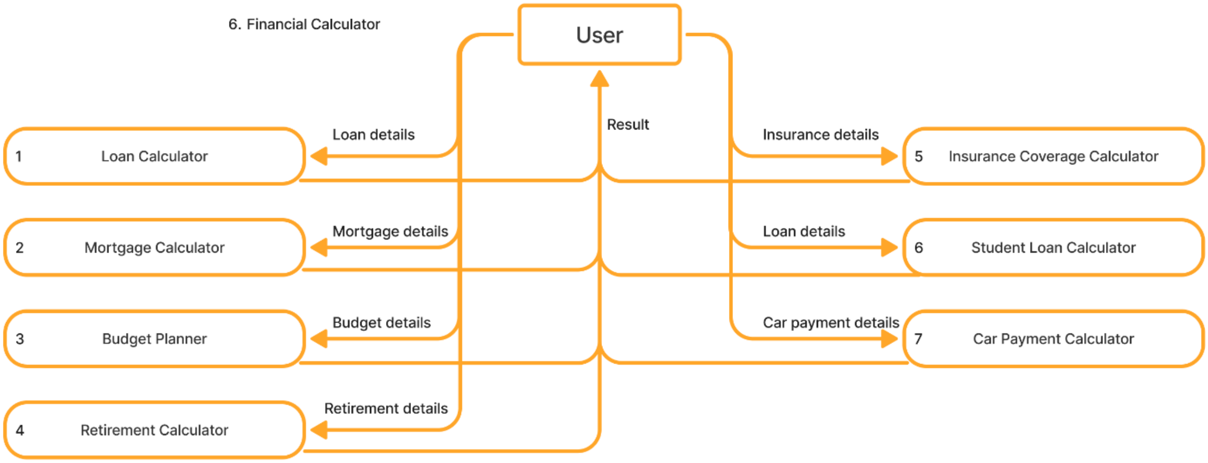


Figure 9. DFD Level 2 - Financial Calculator

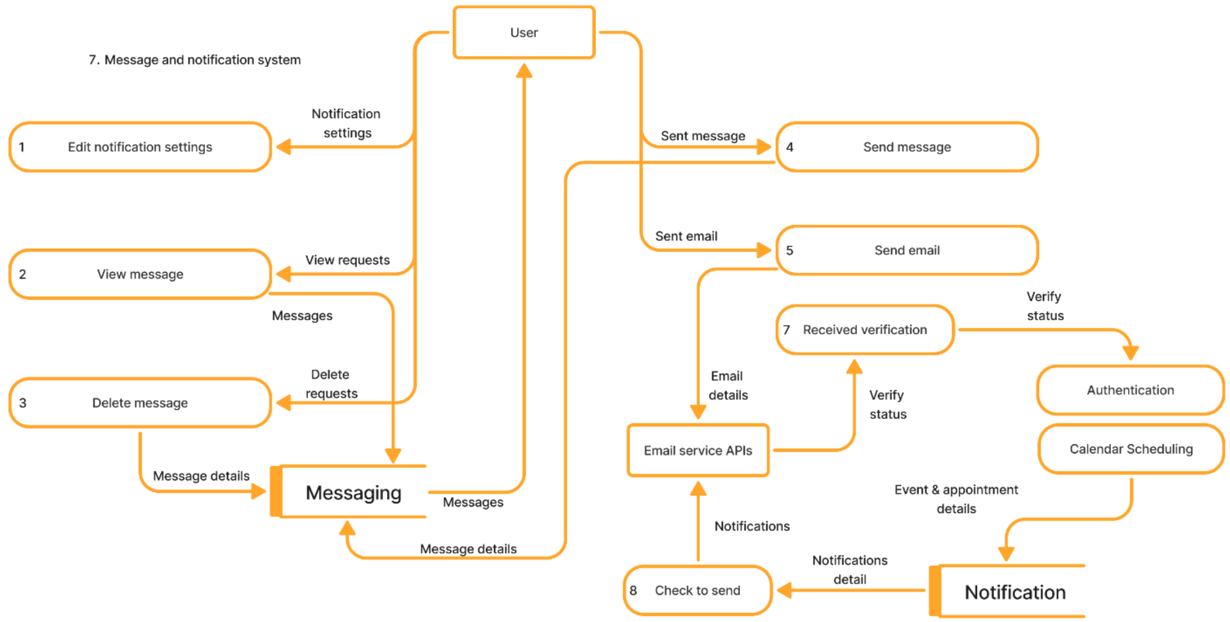


Figure 10. DFD Level 2 - Message and notification system

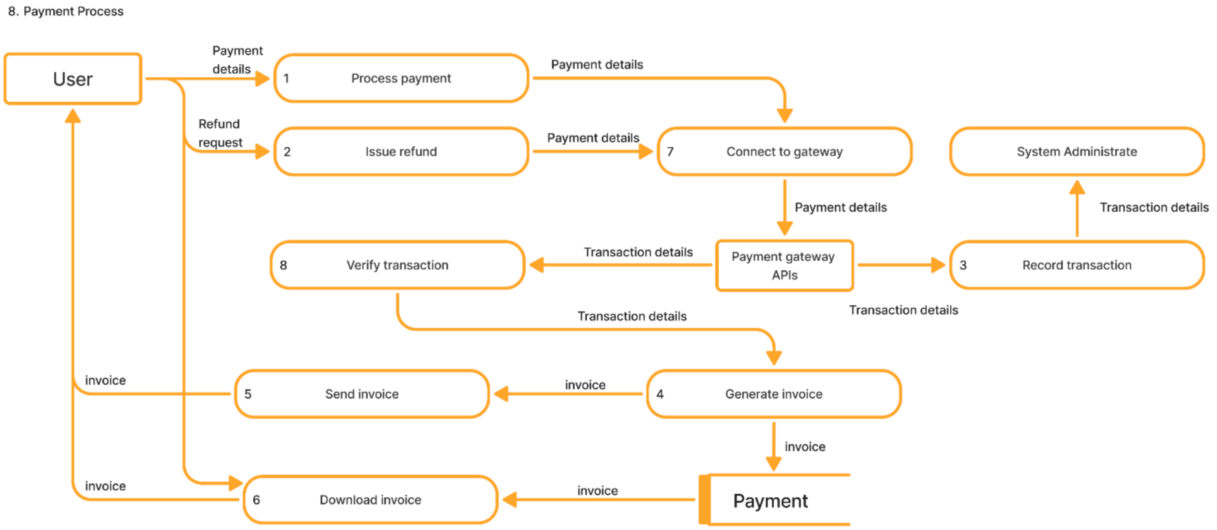


Figure 11. DFD Level 2 - Payment process

B. Module dictionary

- **Central Webpage**: landing page of the website. It handles user’s request and response to them. The central webpage allows users to navigate to other modules in the system without interrupting their experience. It connects to other modules to provide services to system users.

- **System Administrate**: it monitors the whole system activity to ensure all modules are available. It also acts as a barrier to prevent attacks from outside to protect the system and alerts the administrator if it detects attacks or intruders.
- **Account Manager**: managing all aspects of user's account such as subscription, information, etc. It allows users to make payments for subscriptions and changes to their profile.
- **Learning hub:** allowing users to access the available resources and read the news articles fetch from news API.
- **Calendar scheduling**: allowing users with advisor level permission or higher to create, modify, cancel events. The lower-level user can view the list of events, schedule an appointment with the advisor. All users will be available to attend the meeting through external video hosting services.
- **Financial Calculator**: allow users to access financial tools.
- **Message & Notification System**: get the event details to send the user a notification reminding them about up-coming events. Using user details to select the person they want to send a message to.

C. Data Dictionary

- **Users**: contain email, hashed password, first name, last name, phone number, role, verification status, creation date, update date.

- **User profiles**: contains profile ID, user ID, profile picture, financial goals, address, city, state, country, postal code, privacy settings.
- **Advisors**: contain advisor ID, user ID, bio, certifications, specialties, rating.
- **Consultations**: contains consultation ID, client ID, advisor ID, scheduled date, status, client rating, session notes.
- **Event registrations**: contains registration ID, event ID, user ID, registration date.
- **Events**: contains event ID, organizer ID, title, description, event date, location, event type.
- **Notifications**: contains notification ID, user ID, content, notification type, status, created at.
- **Messaging**: message ID, sender ID, receiver ID, message content, sent at, read status.
- **Payments**: contains payment ID, user ID, amount, payment method, payment date, transaction ID, payment status.
- **Subscriptions**: contains subscription ID, user ID, plan type, start date, end date, auto renewal.
- **Financial Tools Usage**: contains usage ID, user ID, tool type, input data, result data, usage date.
- **System logs**: contains log ID, user ID, event type, event description, event timestamp, IP address.

V. Design Legend

A. I/O

The UI for the Wealthwise website will aim to be minimalist. It will be responsive to resize elements to fit the user's screen size. The website should be able to function on both mobile and desktop, but since most of our users will most likely be using the website, we will focus on making the UI work for desktop. There will be a top bar for navigating through pages and user settings. The page below the top bar will display the current page contents. A working color palette is listed below. The primary colors are green and blue. Green was chosen because it signifies wealth, while blue is a nice color acceptable to many.

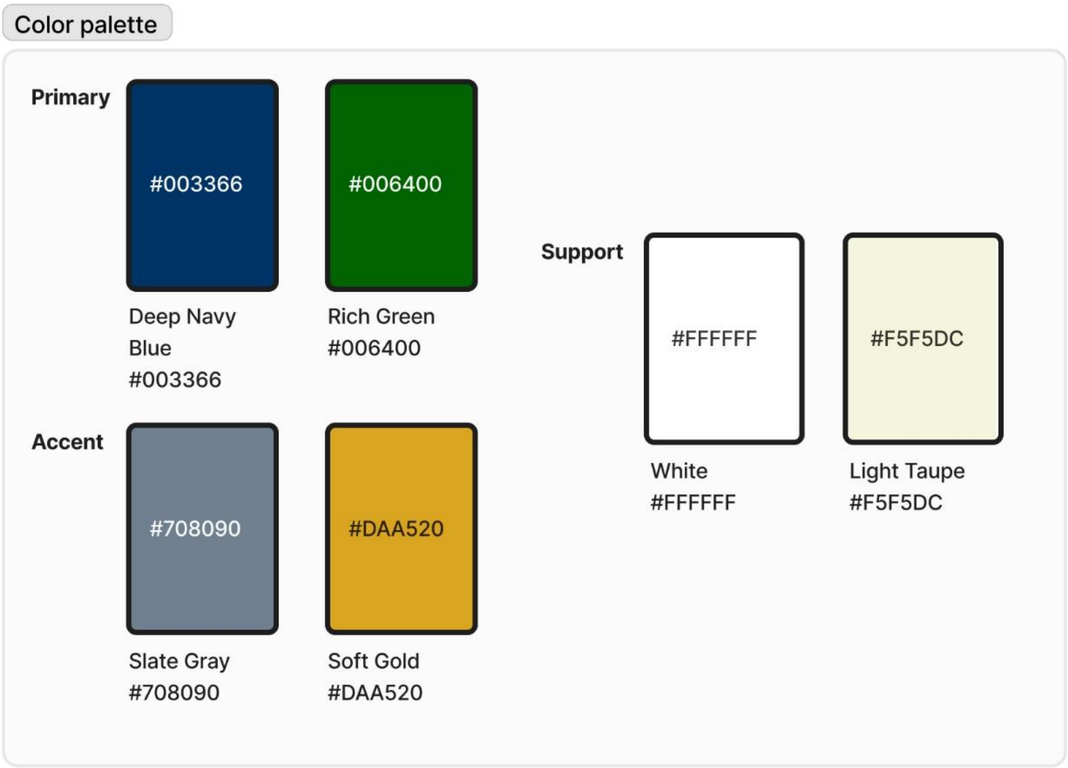


Figure 12. WealthWise color palette

B. Human Interfaces

The user experience should be a simple, intuitive one. Our website has to perform simplistic functions and draw in users. Therefore, it should be minimalistic and straightforward so new users can easily find what they are looking for without it being overwhelming. The website must be similar to users' expectations so they can easily navigate it. Everyday actions should be readily accessible, and navigation should follow a logical path. The website also needs fast load times, as responsiveness is crucial to retaining users. It should also come with some accessibility features, as listed below, to increase the number of possible customers.

- **Keyboard Navigation**: Enable users to navigate using a keyboard for those with difficulty with a mouse.
- **High Contrast Mode**: Offer a high-contrast color scheme for users with visual impairments or color blindness.
- **Text Resizing Options**: Allowing users to adjust font sizes for better readability.

C. Report Format

We will use Google Analytics to report user analytics since it is free. Additionally, it is a premade tool, so we won’t have to spend extra time developing it. Google Analytics offers built-in capabilities for tracking user sessions, behavior flows, conversion rates, and demographics. It provides all the necessary capabilities and is easily scalable.

For security analysis, we currently have no plans to develop a report-generating tool.

Database reports can be made through SQL queries to the database. This method may require a learning curve to use, but it can tell you anything you want to know about the database.

Queries can be highly customizable to get the exact information you need. The tool won’t have to be redeveloped if the database changes.

D. Input

The user will be able to use a variety of inputs to navigate the website. The user will input into the website primarily through mouse clicks on the website window. However, there will also be keyboard shortcuts for actions that require mouse clicks. Additionally, there will be some text inputs where the user can use a keyboard to type in information. We will also use an API where users can input their credit card or PayPal information.

E. Output

The program will have a couple of output methods. The primary method is simply displaying the information in a readable format in text boxes. There will also be a calendar page to see when appointments or other events are scheduled. Additionally, the information will be displayed in an intuitive table for some calculators that require it, like the budgeting calculator.

There are a couple of additional outputs that rely on outside systems. The website will generate video conference service hyperlinks for appointments and webinars that the user can click to be transferred to that service. The last output method is through email for sending reminders for appointments or receipts.

VI. Top Level Design

A. Actual Diagram

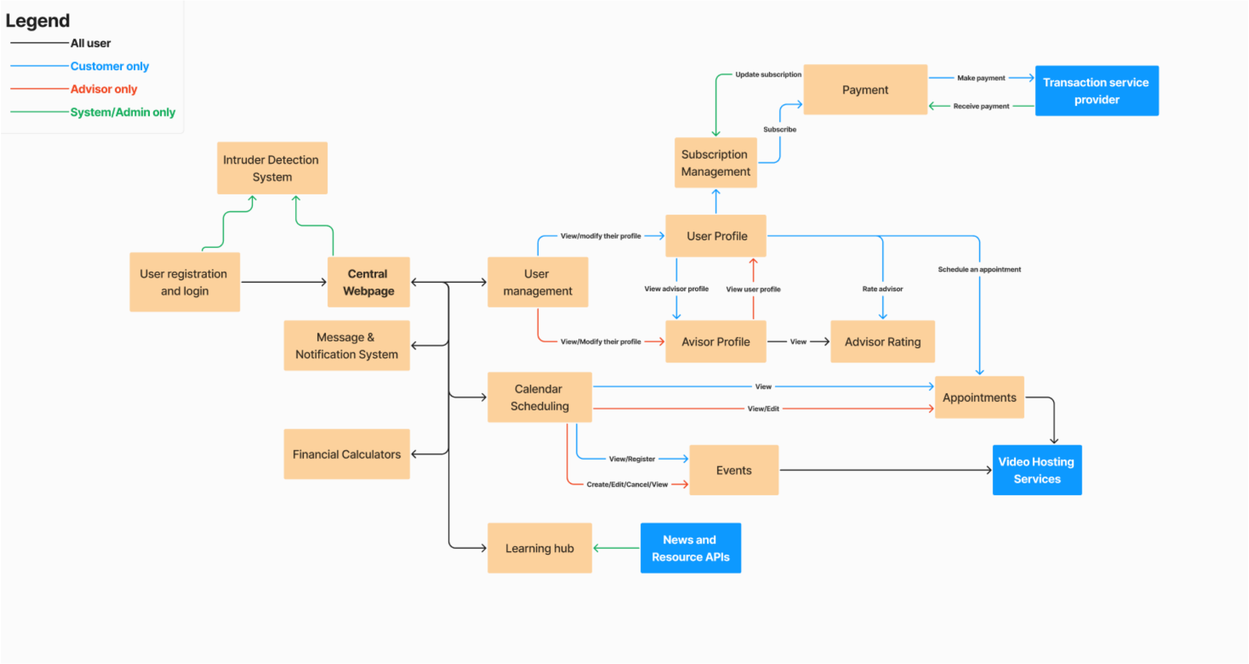


Figure 13. WealthWise Top Level Design - Actual Diagram

B. Performance Constraints

- The system should be designed to handle up to 25,000 concurrent users. This means every other performance requirement and constraint should hold true when the web app is at its expected peak of 25,000 concurrent users.
- The web app shall have three nines of availability, or 99.9% availability (uptime), or less than .1% downtime. This does not include regularly scheduled maintenance, which shall take place outside of peak hours.

- The average response time of the web app shall be 3 seconds or less for calculations and page transitions. The maximum response time of the web app shall be 10 seconds for calculations and page transitions.
- The system shall utilize no more than 5% of a 2.3 GHz CPU per user session and utilize less than 1GB of RAM per session.

C. Archival Procedures

- The system shall provide manual backup options to allow system administrators to back up the system before a suspected incident, failed scheduled backup, or before a major system overhaul. The manual system backup shall be implemented with standard operating system commands, such as Window's robust copy (robocopy) command.
- Daily incremental and weekly full backups shall be able to be scheduled at the site administrator's discretion, which shall create a full snapshot of both the database and file storage. This will ensure device failure or cyberattacks do not result in total data loss.

D. Fault Handling Approach

- In case of payment failure, the web app shall roll back all aspects of the transaction just before the transaction began.
- In case of a database action failure, the web app shall retry the command two additional times before informing the user and/or system logs of the failure.
- In case of a server access failure (while attempting user login, website interaction, event scheduling, etc.), the web app shall retry the connection two additional times before informing the user of the failure.

- In the case of a report generation or administrative action failure, the systems logs and relevant users shall be notified of the failure.
- In the case of bad parameters being entered into a calculator, the web apps shall display a message informing the user of the error.
- In the case of a scheduling conflict, the web app shall inform the user of the conflict and not proceed with scheduling the conflicting event/advising.
- Upon an API failure, the failure shall be noted in the system logs. If this is in response to a user action (e.g. payment) the user shall also receive an error message.

VII. Medium Level Design

A. Class definitions

1. Main module

- **Main**
    - Description: The command center of operations, where all other classes will originate from.
    - Purpose: This function will spin classes up and down as necessary and will facilitate how often any of their functions run. This will support the central webpage by default.
    - Attributes
        - User ID
        - One of each class is listed below.
    - Methods:

        - None.
    - Requirements:
        - None, but this is the underlying layer that makes everything work..

2. Account management module

- **User**
    - Description: Represents a user of the platform, containing essential details for identifying and managing users such as unique ID, role, and email.
    - Purpose: Manages core user data and performs essential account actions like registration and password management. This class will allow password updating and internal password working such as hashing and requirement validation.
    - Attributes:
        - user_id: Unique identifier for each user.
        - email: User's email address.
        - role: Defines user type (e.g., client, advisor, admin).
        - is_active: Status of the user account (active/inactive).
        - -created_at: Timestamp of account creation.
        - -updated_at: Timestamp of last account update.
    - Methods:
        - register_user(): Handles new user registration and stores user data. Will call notification to send email to user.
        - update_Password(): Handles updating the password.

        - -verification_recieved(): Enter registration information into the database user when the user verifies their email.
        - -validate_input(): Validates user registration inputs for correct formats.
        - -hash_password(): Encrypts user passwords before storing them.
        - -verify_password(): Validates user password during login.
    - Requirements: 1.1.1, 1.1.2, 1.1.3, 1.7.2

- **User Profile**
    - Description: Stores additional personal details associated with each user, including contact information and financial preferences. The data here can be used within the user profile page.
    - Purpose: Provides a personalized experience by storing individual user data, preferences, and financial goals. This will be used to display data on the user profile page so the user can view and modify.
    - Attributes:
        - profile_id: Unique identifier for user profile.
        - user_id: Link to the associated user account.
        - first_name: User's first name.
        - last_name: User's last name.
        - phone: Phone number.
        - privacy_settings: Configurations for profile visibility.
        - financial_goals: User's set financial goals.
        - subscription: An instance of the subscription class
    - Methods:

        - edit_profile(): Allows users to update their profile information.
        - view_profile(): Retrieves and displays profile information.
        - set_privacy(): Sets privacy settings for the profile.
        - edit_notification_settings(): set the notification settings for the user
        - view_messages(): Retrieves messages for user.
        - view_prev_events(): Shows the user's previous event participation.
        - view_prev_consults(): Shows the user's previous consultations.
        - delete_message(): Deletes a message.
    - Requirements:1.4.1, 1.4.3, 1.6, 1.8.4
- **AdvisorProfile (inherits from UserProfile)**
    - Description: A specialized profile for advisors, containing additional fields like credentials and ratings. This class is an extension of the UserProfile class. It differentiates users and advisors. It will contain additional information, such as bio and credentials.
    - Purpose: Enables advisors to maintain a professional profile, allowing users to assess qualifications and ratings for consultations. This will allow the advisors to have the tools they need to do their job.
    - Attributes:
        - advisor_id: Unique identifier for each advisor profile.
        - bio: Advisor's biography.
        - credentials: Certifications and qualifications of the advisor.
        - ratings: Average rating based on user reviews.
    - Methods:

    - view_ratings(): Displays ratings and reviews for advisors.
    - rate_advisor() : Allows a user to rate an advisor
    - set_availability(): Allows advisors to set available time slots.
    - modify_availability(): Updates existing time slots.
    - view_availability(): Displays current availability.
  - Requirements: 1.4.2, 1.7.1, 8.2.1
- **Subscription (user will have a instance of it)**
  - Description: Represents a user's subscription status, plan type, and history of payments. This class will store the subscription data for each package.
  - Purpose: Manages subscription-related actions, including renewals, cancellations, and tracking the subscription lifecycle. This will be what the purchase activates.
  - Attributes:
    - subscription_id: Unique identifier for each subscription.
    - status: Current status (active/inactive) of the subscription.
    - start_date: Subscription start date.
    - end_date: Subscription end date.
    - plan_type: Type of subscription plan.
  - Methods:
    - activate_subscription(): Subscribes the user or renews an active subscription. Can be used for any subscription package.
    - cancel_subscription(): Cancels the subscription.
    - check_status(): Checks the current status of the subscription.

        - view_history(): Shows the user's transaction history for subscriptions.
        - refund(): Starts the refund process.
    - Requirements: 1.5.1, 1.5.2, 1.5.3, 6.2.2
- **Authentication**
    - Description: Manages user authentication and session handling. The class will also verify the user's email verification link.
    - Purpose: Ensures secure login, session management, and token validation to maintain authorized access. This class allows users to enter and exit the site. An administrator or too many failed login attempts can lock the account.
    - Attributes:
        - session_id: Unique identifier for each session.
        - user_id: Associated user ID for the session.
        - session_token: Token for authenticating the session.
        - -expiry_time: Expiry time for the session token.
    - Methods:
        - login(): Authenticates user and initiates a session.
        - logout(): Ends the user session.
        - lock_account(): Locks account after repeated failed login attempts.
        - -verify_token(): Verifies the verification token has not expired
    - Requirements: 1.2, 1.3.2, 8.2.2
- **PasswordManager**
    - Description: Facilitates password reset and complexity validation for user security.

  - Purpose: Provides a secure way for users to reset forgotten passwords and ensures password strength. This class will do the actual calculation on password strength and manage rese t tokens.
  - Attributes:
    - user_id: User identifier for password management.
    - reset_token: Token for password reset.
    - expiry_time: Expiry for the reset token.
  - Methods:
    - send_reset_email(): Sends password reset link to the user.
    - reset_password(): Resets password using reset token.
    - check_password_complexity(): Ensures password meets security requirements.
  - Requirements: 1.3.1

3. Calendar Module

- **Calendar**
  - Description: Core calendar structure to manage both events and appointments.
  - Purpose: Allows users and advisors to schedule events, manage appointments, and register for available slots. This class will be inherited by other classes such as event and Appointments. It will allow them the functionality necessary to schedule events.
  - Attributes:
    - events[]: Array of structs storing current events and their associated content.

- Methods:
  - add_event(): Allows user to add an event.
  - delete_event(): Allows the user who created the event (or an admin) to delete the event.
  - modify_event(): Allows the user who created the event (or an admin) to modify the event.
  - add_Appointment(): Allows the user to schedule an appointment with an advisor
  - register(): Allows a user to register to an event
  - check_availability(): Gets a list of appointments to check scheduling availability (for advisor appointments).
  - get_Events(): Gets event list for further availability checking (for advisor appointments).
  - enter_Event(): allows users to enter an event or consultation via an external service. Will log the user's participation in the database for future use.
  - download_content(): Allows a user to download content associated with the event.
- Requirements: 3.1.2, 3.2.2, 5.1, 5.2

- **Appointments (inherits from calendar)**
  - Description: Manages appointments with advisors, including availability and scheduling.

  - Purpose: Provides functionality for users to book advisor consultations based on available time slots. This class will primarily use the functionality from the calendar but will have a few members and methods of its own to manage advisory-specific data and searching.
  - Attributes:
    - available_slots[]: Data type allowing the listing of time slots when the advisor is available.
    - advisors: a struct with advisors and their availability
    - appointments[]: an array of structs with appointments ID, date and time, user ID, advisor ID, consultation link, and status.
  - Methods:
    - search_advisors(): allows a user to search advisors with given parameters.
  - Requirements: 3.1.1
- **Event (inherits from calendar)**
  - Description: Manages events like seminars and webinars, with scheduling and participation tracking.
  - Purpose: Enables users to participate in events by managing event details, registration, and access. This class will primarily use the functionality from the calendar but will have a few members of its own to manage more detailed event information.
  - Attributes

        - event_info[]: list of event dates and times, ID, host user ID, event link, and status
    - Methods
        - -
    - Requirements: 5.1, 5.2

4. Financial Calculators Module

- **LoanCalculator**
    - Description: A tool for calculating monthly payments and total interest for loans.
    - Purpose: Assists users in financial planning by estimating loan repayment costs based on given parameters. The loan calculator will take in principle, interest, and term and find the monthly payment and the total interest paid.
    - Attributes:
        - loan_amount: Amount of the loan.
        - interest_rate: Interest rate applied to the loan.
        - loan_term: Duration of the loan in months/years.
    - Methods:
        - calculate_monthly_payment(): Calculates the monthly payment.
        - calculate_total_interest(): Calculates total interest over the loan term.
    - Requirements: 2.1
- **MortgageCalculator**
    - Description: A specialized calculator for estimating mortgage payments and total interest.

- Purpose: Helps users understand mortgage expenses, enabling better home-financing decisions. The calculator will read in home price, down payment, interest rate, and loan term, and calculate the monthly payment, and total cost of the mortgage.
- Attributes:
    - home_price: Price of the home.
    - down_payment: Amount of the down payment.
    - interest_rate: Interest rate for the mortgage.
    - loan_term: Duration of the mortgage.
- Methods:
    - calculate_monthly_mortgage(): Calculates monthly mortgage payment.
    - calculate_total_cost(): Calculates the total cost of the mortgage.
- Requirements: 2.2

- **BudgetPlanner**
    - Description: A tool for tracking income, expenses, and savings to assist with budgeting.
    - Purpose: Supports users in managing finances by analyzing spending habits and offering savings recommendations. This class will take in income sources, fixed expenses, variable expenses, savings goals, and various user report options and will display analytics in spending, recommendations, and a report that the user may store in the database.
    - Attributes:
        - income_sources: User's income sources.

        - fixed_expenses: List of fixed expenses.
        - variable_expenses: List of variable expenses.
        - savings_goals: User's savings goals.
        - report_options: List of input user options for report generation.
    - Methods:
        - log_results(): stores user’s results in the database.
        - track_expenses(): Logs expenses.
        - analyze_spending(): Analyzes spending categories.
        - provide_recommendations(): Suggests savings adjustments.
        - update_report_options(): allows the user to update their user settings.
        - generate_report(): Allows the user to finalize the current analytics to a report.
    - Requirements: 2.3, 7.1

- **RetirementCalculator**
    - Description: Calculates estimated retirement savings based on user inputs.
    - Purpose: Assists users in planning for retirement by projecting potential savings and suggesting adjustments to contributions. This calculator reads in the current age, retirement savings goal current savings, and monthly contribution, and calculates recommended contribution adjustment and projected savings by retirement age.
    - Attributes:
        - current_age: Current age of the user.
        - retirement_goal: Desired savings by retirement.

        - current_savings: Current retirement savings.
        - monthly_contribution: Amount contributed monthly.
    - Methods:
        - calculate_estimated_savings(): Projects retirement savings.
        - provide_saving_recommendations(): Suggests contribution adjustments.
    - Requirements: 2.4

- **InsuranceCoverageCalculator**
    - Description: Determines recommended insurance coverage based on personal factors like dependents and current coverage.
    - Purpose: Guides users in assessing their insurance needs and potential gaps in coverage. This class will read in age, dependents, current coverage, and additional coverage required and calculate needed coverage and insurance plan options recommended.
    - Attributes:
        - age: Age of the user.
        - dependents: Number of dependents.
        - current_coverage: Existing insurance coverage.
        - additional_needs: Additional coverage needs.
    - Methods:
        - calculate_needed_coverage(): Calculates suggested coverage.
        - suggest_coverage_options(): Recommends coverage plans.
    - Requirements: 2.5

- **StudentLoanCalculator**
    - Description: Calculates costs associated with student loans, including monthly payments and interest.
    - Purpose: Helps users understand the implications of student loans and provides options for optimizing repayment. This class reads in loan amount, interest rate, repayment term, and payment amount and will calculate loan data based on 3 payment options, and what the effect of additional payments would be.
    - Attributes
        - Loan_amount: The principle of the loan
        - interest_rate: The interest rate of the loan
        - repayment_term: Amount of time the loan must be re-paid in
        - payment_amount: monthly payment for the loan
    - Methods
        - calculate_loan_data(): calculates total interest and overall cost based on the monthly payment amount and values 10% lower and 10% higher.
        - calculate_extra_payment(): calculates the same output based on paying 1, 2, and 3, additional payments with the first payment.
    - Requirements: 2.6
- **CarPaymentCalculator**
    - Description: Estimates of car loan payments, including impact of down payments and interest.
    - Purpose: Aids users in evaluating car financing options by calculating monthly payments and total costs. The function takes in loan amount, interest rate, loan

term, and down payment and calculates the impact of additional payments, loan term, and down payment.

- Attributes
    - loan_amount: the principal loan amount
    - interest_rate: the interest rate of the load
    - loadn_term: the duration of the payment period of the loan
    - down_payment: the initial down payment of the loan
- Methods
    - calculate_loan_data(): calculates monthly payment, total interest paid, total load cost, amortization schedule, payoff date, and impact of paying 1, 2, and 3, additional payments with the first payment.
    - calculate_alternate_payment(): calculates effects of adjusting loan term and down payment.
- Requirements: 2.7

5. Payment Processing Module

- **Payment**
    - Description: Manages individual transactions, including payment amounts and status.
    - Purpose: Processes payments, records transactions, and handles refunds to support subscriptions and event fees. This class handles all refunds, payments, and recording of transition to the database.
    - Attributes:
        - payment_id: Unique ID for each transaction.

    - user_id: ID of the user making the payment.
    - amount: Amount of payment.
    - payment_date: Date of transaction.
    - payment_method: Method used (e.g., credit card).
    - transaction_status: Status of the payment.
  - Methods:
    - process_payment(): Initiates the payment process.
    - issue_refund(): Issues refunds if applicable.
    - record_transaction(): Logs transaction details.
  - Requirements: 1.5.4, 1.5.5, 6.2.2
- **Invoice**
  - Description: Manages individual transactions, including payment amounts and status.
  - Purpose: Processes payments, records transactions, and handles refunds to support subscriptions and event fees. This class handles all refunds, payments, and recording of transition to the database.
  - Attributes:
    - payment_id: Unique ID for each transaction.
    - user_id: ID of the user making the payment.
    - amount: Amount of payment.
    - payment_date: Date of transaction.
    - payment_method: Method used (e.g., credit card).
    - transaction_status: Status of the payment.

  - Methods:
    - process_payment(): Initiates the payment process.
    - issue_refund(): Issues refunds if applicable.
    - record_transaction(): Logs transaction details.
  - Requirements: 1.5.4, 1.5.5, 6.2.2

- **PaymentGateway**
  - Description: Interfaces with external payment providers (e.g., Stripe, PayPal) to process payments securely.
  - Purpose: Manages connections to payment services, ensuring secure transactions and handling disputes if needed. This class primarily manages connection with payment API calls to external parties.
  - Attributes:
    - gateway_id: ID for each payment gateway.
    - provider_name: Name of the payment provider.
    - api_key: API key for accessing the gateway.
    - secret_key: Secret key for secure transactions.
  - Methods:
    - connect_to_gateway(): Connects to the payment service.
    - verify_transaction(): Verifies transaction status.
  - Requirements: 6.1

6. Messaging and Notifications Module

- **Message**

- Description: Handles communication between users and advisors via in-app messaging and email notifications.
- Purpose: Facilitates communication and notification of critical events, ensuring timely updates for users. This class manages distributing messages to the user and allowing email verification.
- Attributes:
  - message_id: Unique identifier for each message.
  - sender_id: ID of message sender.
  - receiver_id: ID of the message receiver.
  - content: Message text.
  - sent_at: Timestamp of sending.
  - status: Read/unread status.
  - type: Email vs in-app message
- Methods:
  - send_message(): Sends a new message.
  - send_email(): Sends a new email.
  - check_to_send(): Checks the database for approaching events the user will be notified of.
  - received_verification(): Executed when the user verifies the account and passes the call back to Authentication.
- Requirements: 1.1.3, 1.3.2, 1.8.1, 1.8.2, 1.8.3, 3.1.3

7. Learning hub

- Description: Manages access to educational resources like blogs and articles, stored within the system.
- Purpose: Keeps users informed on financial topics by updating content through external APIs and internal storage. This class has an update function that will run the API to update the content management system (WordPress)
- Attributes:
  - stored_content: data storage for content as necessary
- Methods:
  - update(): this runs against the API to general content updates
- Requirements: 4

8. System Administration

- Description: Manages administrative tasks, including user control, system monitoring, and security logging.
- Purpose: Provides system administrators with tools to oversee user activity, manage backups, and ensure system performance and security. This class allows users to view every aspect of the site's users, traffic, analytics, security, and more. This class can also be used to make custom queries on the database, send system alerts, and schedule backup and downtime.
- Attributes:
  - admin_id: Unique identifier for each system administrator.
  - name: Administrator's full name.

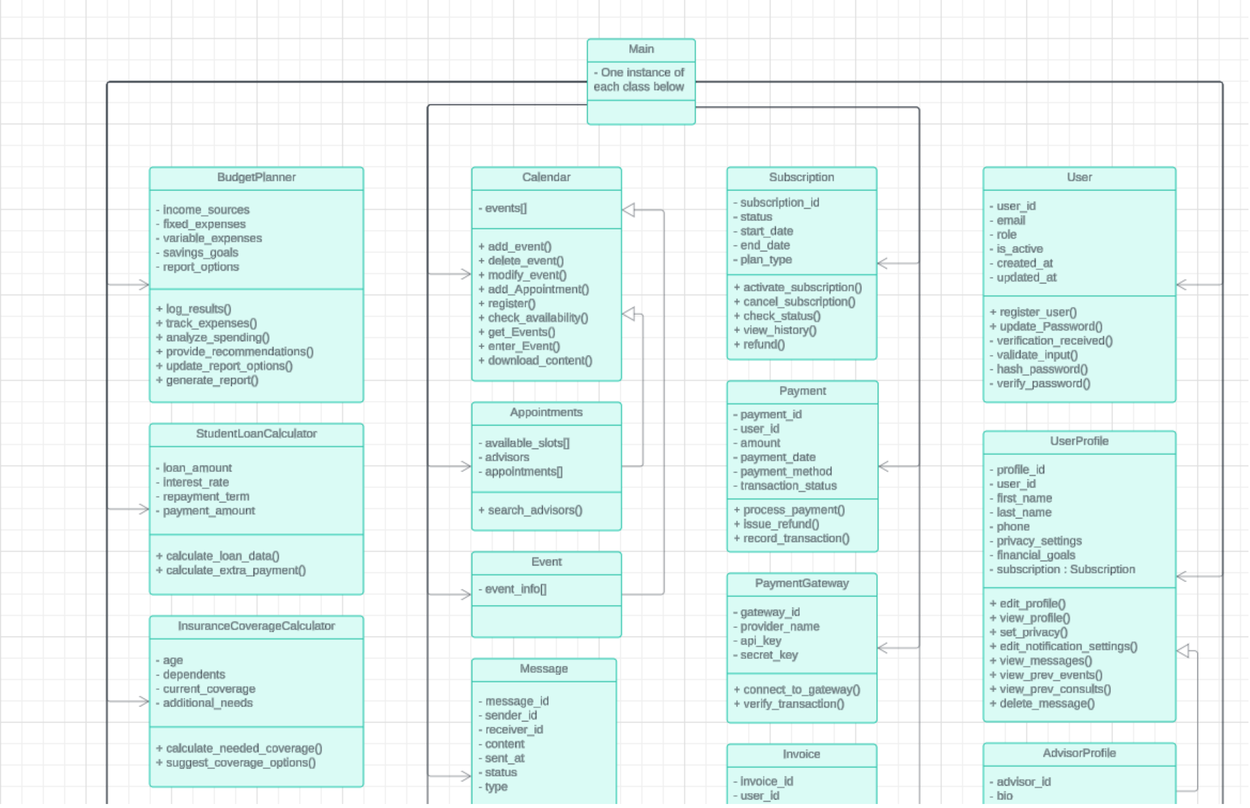
  - email: Administrator’s email address for login and notifications.
  - role: Defines the level of access (e.g., Super Admin, Support Admin).
  - last_login: Timestamp of the last time the admin logged in.
  - permissions: List of permissions granted to the admin based on their role.
  - activity_logs: Collection of recent actions taken by this admin for monitoring purposes.
  - system_status: Real-time system health metrics (CPU usage, memory, active users, etc.).
- Methods:
  - User Management
    - view_user_list(): Retrieves a list of all users registered on the platform.
    - suspend_user(user_id): Suspends a user’s account based on their user_id.
    - reactivate_user(user_id): Reactivates a previously suspended user account.
    - reset_user_password(user_id): Resets the password for a specified user account.
    - assign_role(): Assigns a new role to a user, changing their access permissions (mostly for creating new advisors)
    - modify_user_property(): allows the admin to modify a user’s property
  - System Monitoring
    - view_system_status(): Displays current system health metrics, such as CPU usage, memory usage, and server uptime.

  - generate_performance_report(): Compiles and exports a report on the system's performance over a selected time frame.
  - check_error_logs(): Retrieves recent error logs for troubleshooting system issues.
  - check_security_logs(): Retrieves the security log history with data such as event type, specified user, IP addresses, and timestamps.
  - create_log() : Create a custom log entry to error or security.
  - schedule_maintenance(downtime_schedule): Sets up a scheduled maintenance period, notifying users and restricting access.
- Notification and Communication
  - send_system_alert(message, time, specified_user): Sends an alert or announcement at a specified time (default: now) to a specified user (default: all users)
- Audit and Reporting
  - generate_audit_log(start_date, end_date): Compiles a log of all admin activities within a specified date range.
  - export_user_activity_report(user_id): Generates a report of all actions taken by a specified user.
  - export_system_performance_logs(): Exports logs related to system performance metrics for further analysis.
  - query_database() : Allows the system admin to send a query to the database.
- Backup and Recovery

  - restore_backup(backup_id): Restores the system to a previous state from a specified backup.
  - schedule_backup(interval): Schedules automated backups at a set interval (e.g., daily, weekly).
  - view_backup_history(): Lists recent backup actions and their statuses.

- Requirements: 1.7.1, 1.7.2, 1.7.3, 7.2, 10.1.2, 8.2.1, 8.2.3

B. Class Diagram



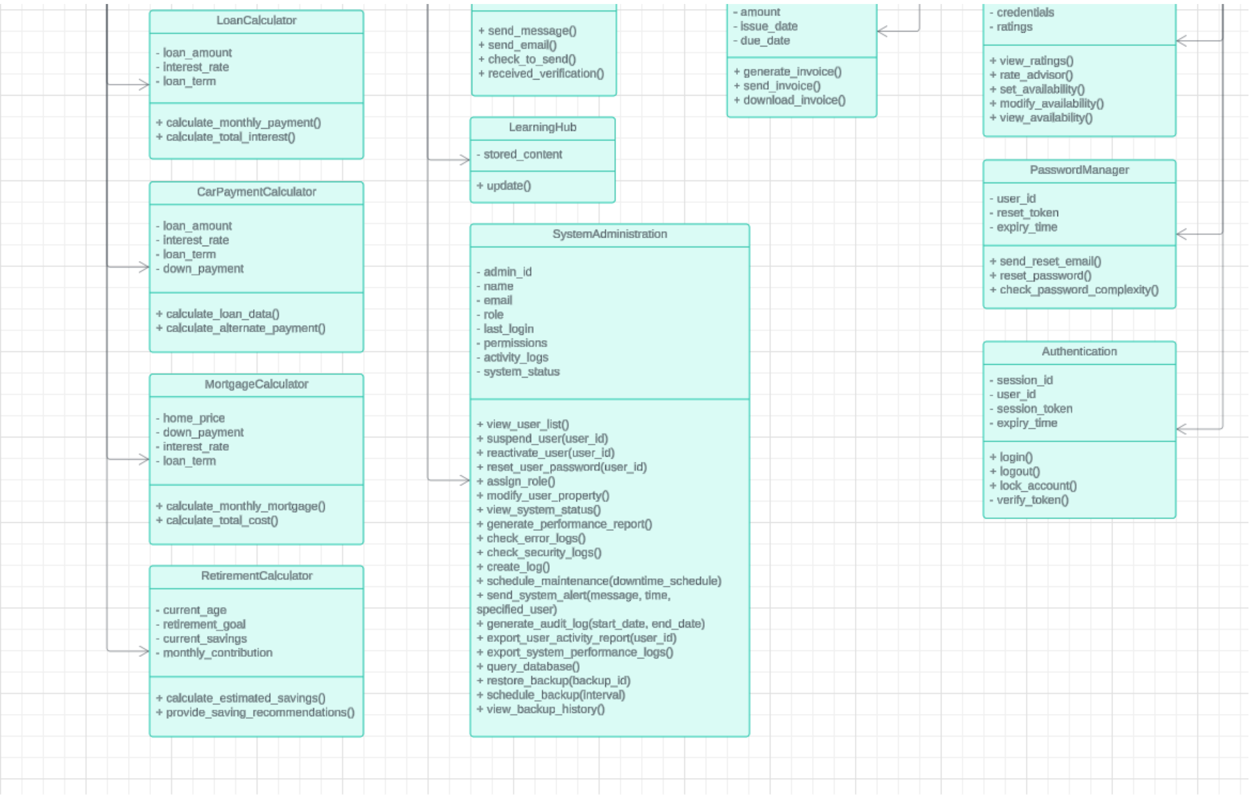
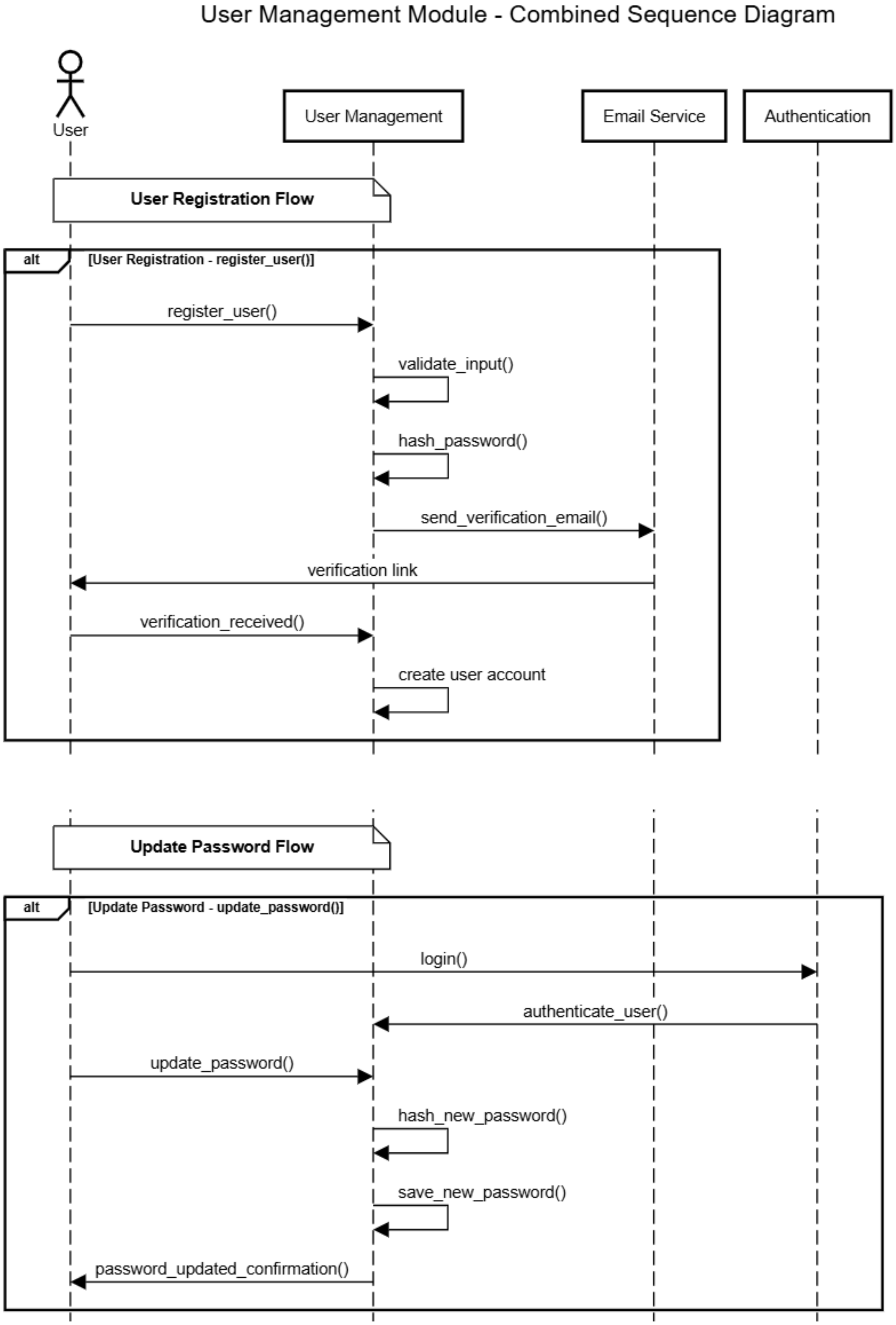
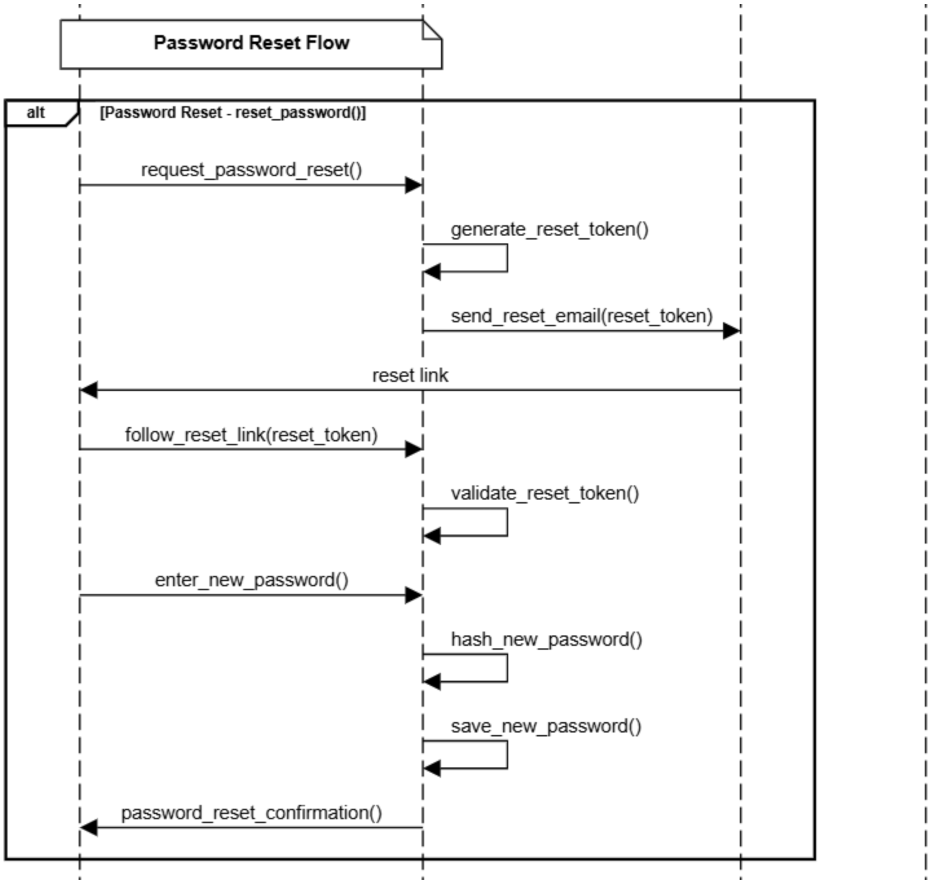
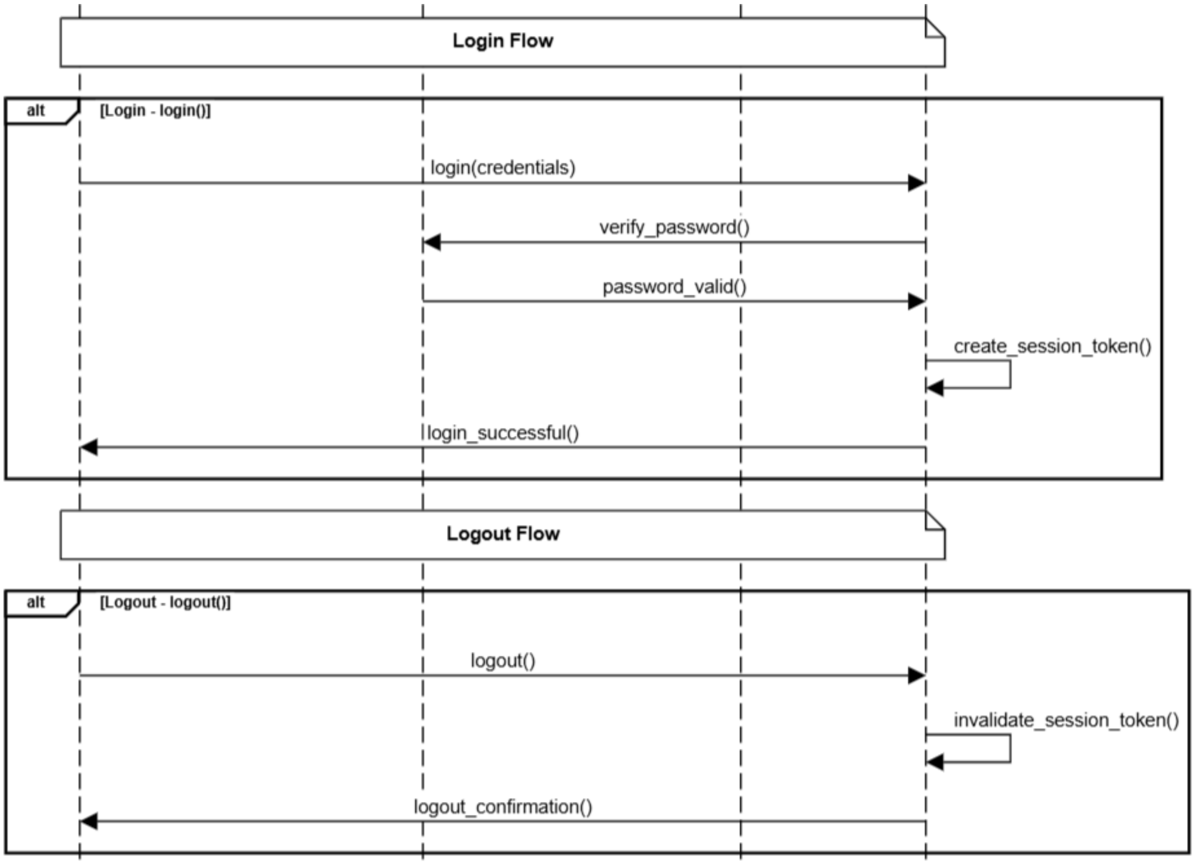


Figure 14. Class Diagram

C. Sequence Diagrams




Password Reset Flow
alt
[Password Reset - reset_password()]
request_password_reset()
generate_reset_token()
send_reset_email(reset_token)
reset link
follow_reset_link(reset_token)
validate_reset_token()
enter_new_password()
hash_new_password()
save_new_password()
password_reset_confirmation()


Login Flow
alt
[Login - login()]
login(credentials)
verify_password()
password_valid()
create_session_token()
login_successful()
Logout Flow
alt
[Logout - logout()]
logout()
invalidate_session_token()
logout_confirmation()

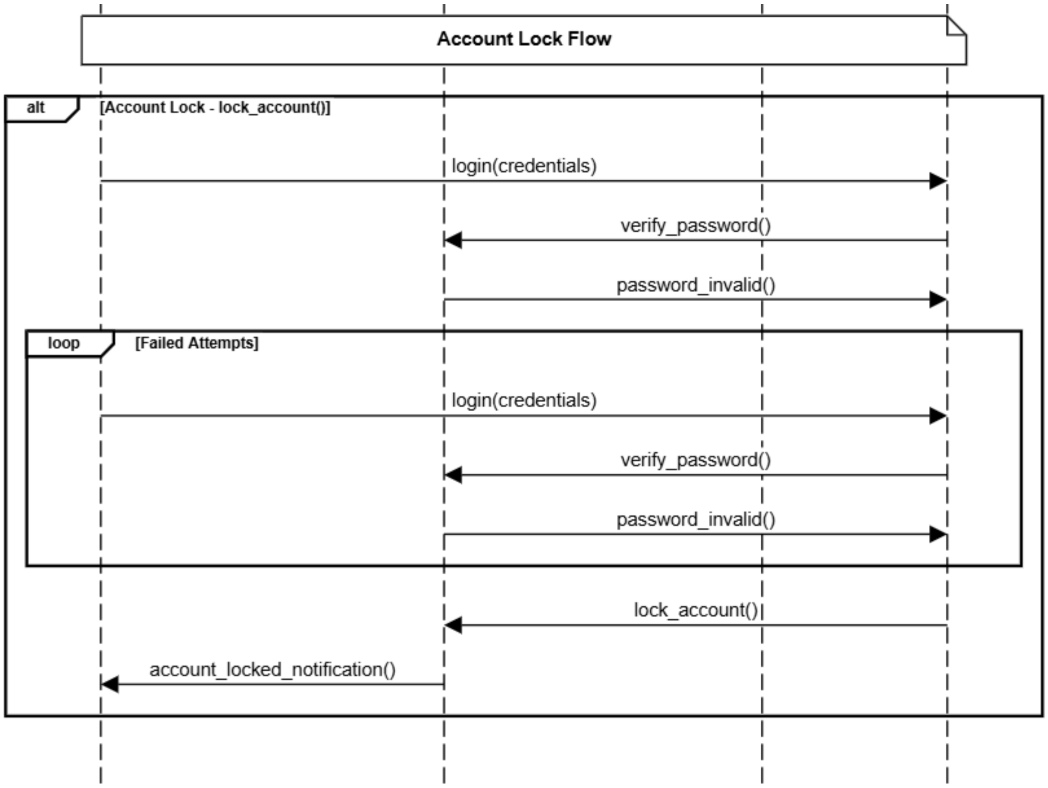
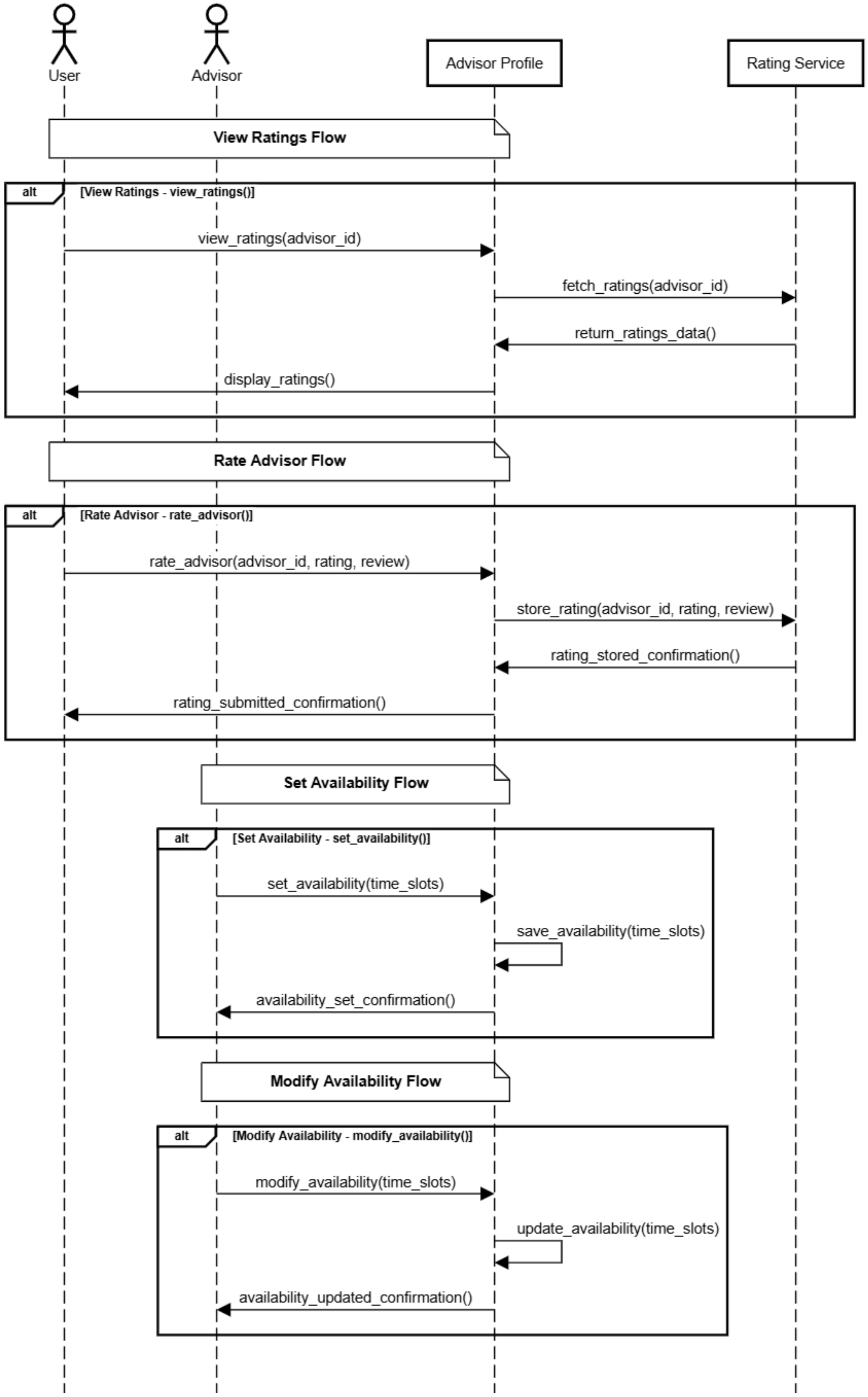


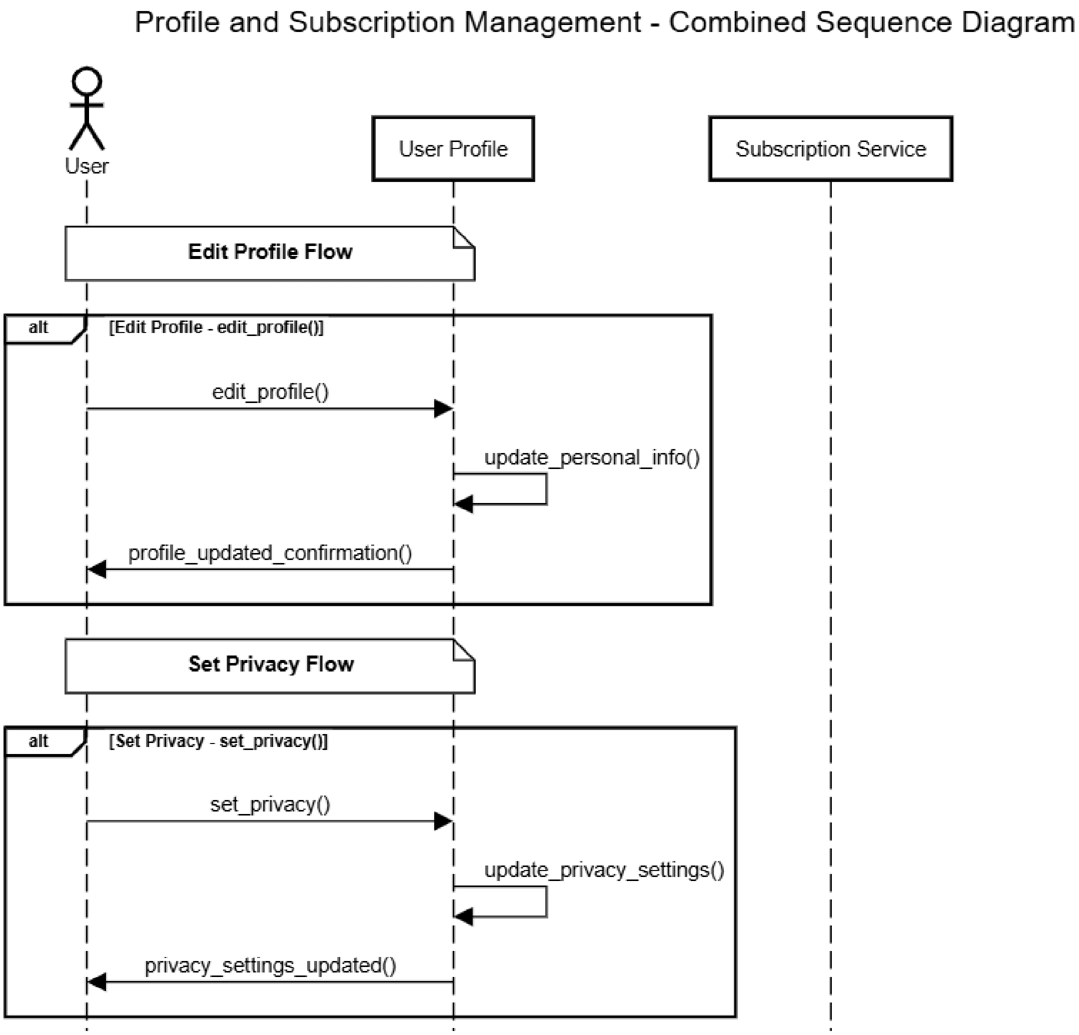
Figure 15. Sequence Diagram – User Management Module

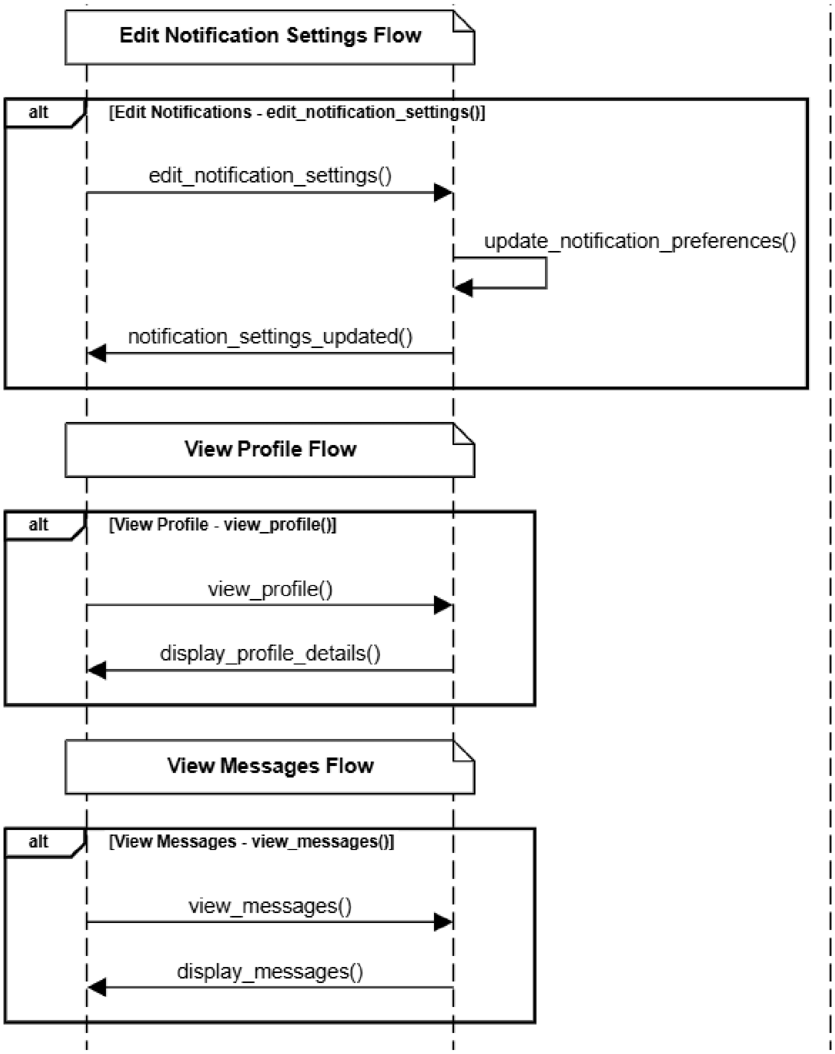
Advisor and Rating Management - Combined Sequence Diagram

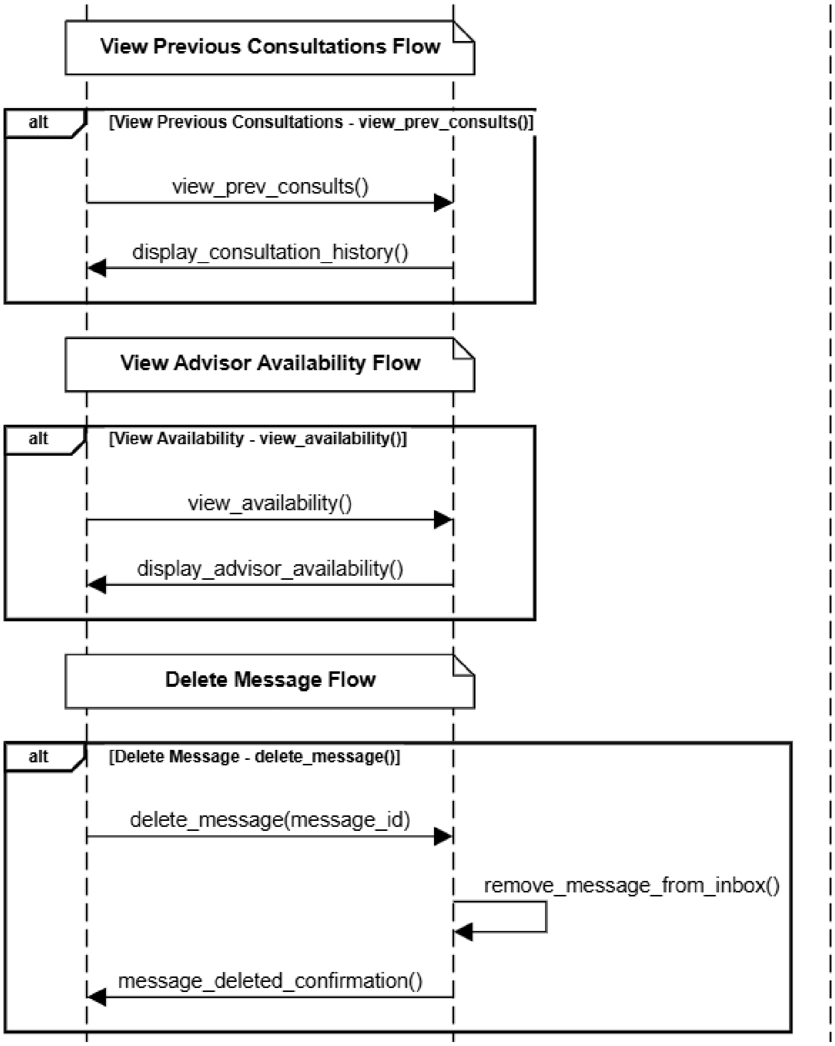


User
Advisor
Advisor Profile
Rating Service
View Ratings Flow
alt
[View Ratings - view_ratings()]
view_ratings(advisor_id)
fetch_ratings(advisor_id)
return_ratings_data()
display_ratings()
Rate Advisor Flow
alt
[Rate Advisor - rate_advisor()]
rate_advisor(advisor_id, rating, review)
store_rating(advisor_id, rating, review)
rating_stored_confirmation()
rating_submitted_confirmation()
Set Availability Flow
alt
[Set Availability - set_availability()]
set_availability(time_slots)
save_availability(time_slots)
availability_set_confirmation()
Modify Availability Flow
alt
[Modify Availability - modify_availability()]
modify_availability(time_slots)
update_availability(time_slots)
availability_updated_confirmation()

Figure 16. Sequence Diagram - Advisor and Rating Management


Profile and Subscription Management - Combined Sequence Diagram
User
User Profile
Subscription Service
Edit Profile Flow
alt
[Edit Profile - edit_profile()]
edit_profile()
update_personal_info()
profile_updated_confirmation()
Set Privacy Flow
alt
[Set Privacy - set_privacy()]
set_privacy()
update_privacy_settings()
privacy_settings_updated()


Edit Notification Settings Flow
alt
[Edit Notifications - edit_notification_settings()]
edit_notification_settings()
update_notification_preferences()
notification_settings_updated()
View Profile Flow
alt
[View Profile - view_profile()]
view_profile()
display_profile_details()
View Messages Flow
alt
[View Messages - view_messages()]
view_messages()
display_messages()


View Previous Consultations Flow
alt
[View Previous Consultations - view_prev_consults()]
view_prev_consults()
display_consultation_history()
View Advisor Availability Flow
alt
[View Availability - view_availability()]
view_availability()
display_advisor_availability()
Delete Message Flow
alt
[Delete Message - delete_message()]
delete_message(message_id)
remove_message_from_inbox()
message_deleted_confirmation()

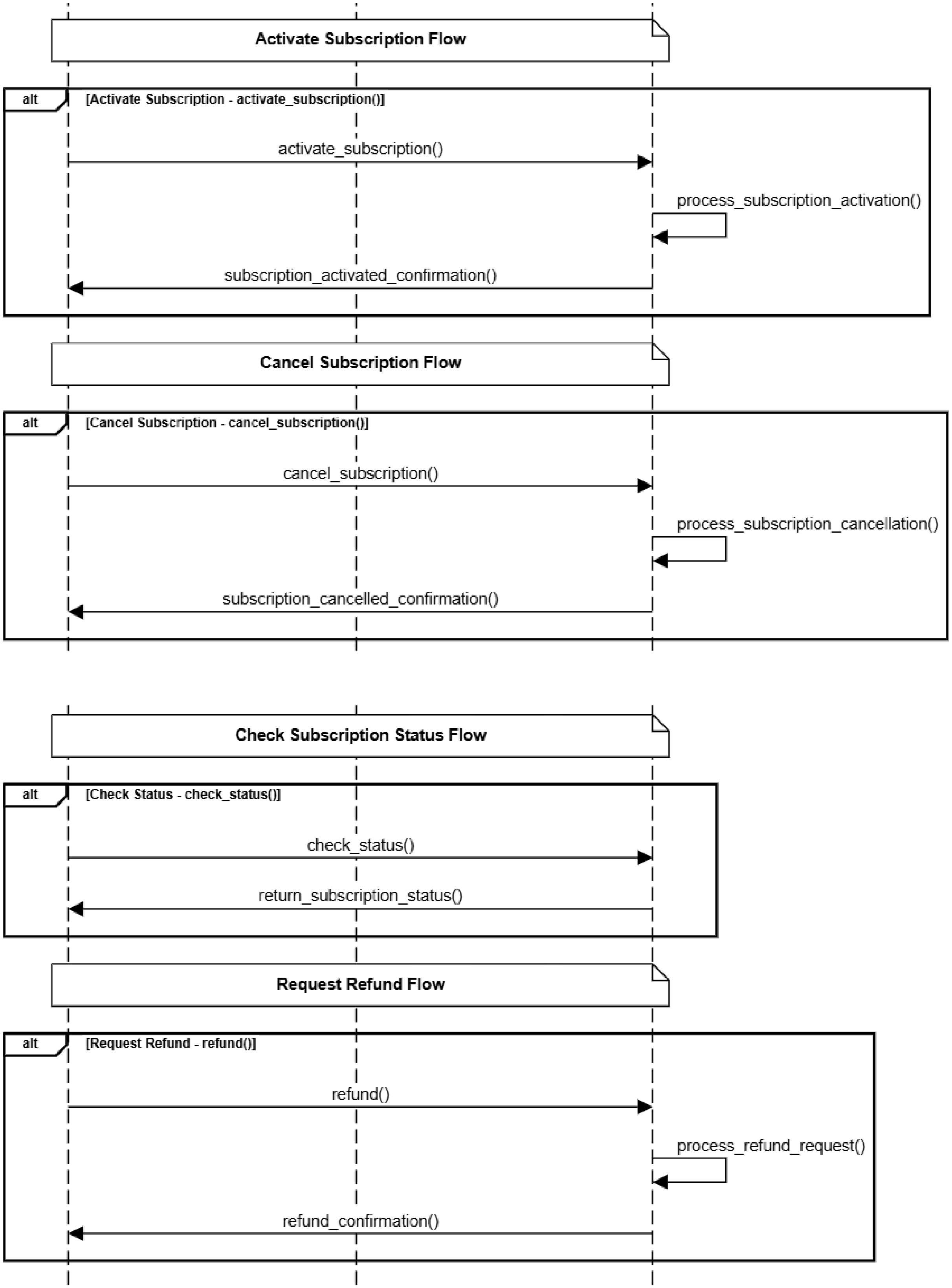

Activate Subscription Flow
alt
[Activate Subscription - activate_subscription()]
activate_subscription()
process_subscription_activation()
subscription_activated_confirmation()
Cancel Subscription Flow
alt
[Cancel Subscription - cancel_subscription()]
cancel_subscription()
process_subscription_cancellation()
subscription_cancelled_confirmation()
Check Subscription Status Flow
alt
[Check Status - check_status()]
check_status()
return_subscription_status()
Request Refund Flow
alt
[Request Refund - refund()]
refund()
process_refund_request()
refund_confirmation()

Figure 17. Sequence Diagram - Profile and Subscription Management

VIII. Files and Tables

A. Files

- SQL activity log
    - This file will record all data entry, modification, deletion, and queries.
- Error log
    - This file will record all errors the system encounters in any module.
- Security event log
    - This file will record any security events, such as login failures.
- Source Files - all .py files are back-end python files that perform calculations and data manipulation necessary for the website to function. All .js files are web page files used for front-end display.
    - main.py
    - user.py
    - userProfile.py
    - userProfile.js
    - AdvisorProfile.py
    - AdvisorProfile.js
    - subscription.py
    - subscription.js
    - authentication.py
    - authentication.js
    - passwordManager.py
    - calender.py
    - calender.js
    - appointment.py
    - appointment.js
    - event.py
    - event.js
    - loanCalculator.py
    - loanCalculator.js
    - mortgageCalculator.py
    - mortgageCalculator.js
    - budgetPlanner.py

  - BudgetPlanner.js
  - RetirementCalculator.py
  - RetirementCalculator.js
  - InsuranceCoverageCalculator.py
  - InsuranceCoverageCalculator.js
  - StudentLoanCalculator.py
  - StudnetLoanCalculator.js
  - CarPaymentCalculator.py
  - CarPaymentCalcualtor.js
  - Payment.py
  - Payment.js
  - Invoice.py
  - PaymentGateway.py
  - Message.py
  - Message.js
  - LearningHub.py
  - LearningHub.js
  - SystemAdmin.py
  - SystemAdmin.js
  - CentralWebpage.js
- Terms_of_Service.txt
  - Terms of service agreement between Wealthwise and end users.
- Useage_Manual.txt
  - User manual for both users and Wealthwise

B. Tables

Also, see 4.3 Data Dictionary.

- Consultations
  - The table used for storing consultation history for each use
- Users
  - The table used for storing user information.
- Event Registrations

    - The table used for strong users registered to a given event

- Events
    - The table listing events and their relevant information
- Advisors
    - The table listing advisor-specific information, such as scheduling, bio, and credentials
- User Profile
    - The table used for storing additional user-specific information such as settings, and additional identifying information.
- Payments
    - The table used for storing payment history.
- Messaging
    - The table used for storing email messages and and blog posts.
- Notifications
    - The table used for storing in-app notifications.
- Financial Tool Usage
    - The table used for storing financial calculator usage (ie. budget planner reports)
- Subscriptions
    - The table is used for storing subscription packaging information.
- Systems logs
    - The table is used for storing system logs.

IX. Appendix

A. Log of Meetings, Reviews, and Meetings[sic]

- **25 September 2024 - Virtual Team Meeting**

Attendees: Draix, John, Norman, Sawyer

The team discussed the Introduction and Work Structure Breakdown sections of the Requirement Document following the reception of the Request of Proposal from the client. The team talked through initial thoughts and confusion about the requirements.

- **26 September 2024 – Client Meeting**

Attendees: Draix, John, Norman, Sawyer, Dr. Albujasim

The team met with the client and discussed a fair portion of the requirements and initial insight into what types of diagrams were necessary and how the WSB should be developed.

- **7 October 2024 – In-person Team Meeting**

Attendees: Draix, John, Norman, Sawyer

The team met and started jointly filling out sections of the Introduction and started the design of the WSB. The team then assigned responsibility for the remaining sub-sections to its members.

- **8 October 2024 – Client Meeting**

Attendees: Draix, John, Norman, Sawyer, Dr. Albujasim

The team met with the client and discussed the full feature list the website should implement. The team also gained additional insight into how the client wanted the Proposed System Overview and the Work Structure Breakdown to be formatted and developed. The team agreed to work asynchronously on the remaining sections and meet virtually before submitting the increment of the RD known as "Assignment 2."

- **9 October 2024 – Virtual Team Meeting**

Attendees: Draix, John, Norman, Sawyer

The team met to review the final changes to the remaining sections of the Introduction and WBS before the final submission the next day.

- **18 October 2024 – Virtual Team Meeting**

Attendees: Draix, John, Norman, Sawyer

The team met to review the full requirements for the RD. Initial thoughts for how the requirements needed to be specified were considered and the next meeting time was set up.

- **21 October 2024 – In-person Team Meeting**

Attendees: Draix, John, Norman, Sawyer

The team met and began jointly filling out all sections of the RD. Most notably, initial versions of all requirements were filled out. All remaining sections were assigned to members to be worked on Asynchronously.

- **22 October 2024 – Client Meeting**

Attendees: Draix, John, Norman, Sawyer, Dr. Albujasim

The team met with the client to discuss the depth of specification for the final requirements. Questions concerning Change Control, Meeting Review, and Terms and Conditions were also discussed. The team decided to finish the RD asynchronously and meet virtually before submitting the “final” RD.

- **23 October 2024 – Team Meeting**

Attendees: Draix, John, Norman, Sawyer

The team met and reviewed the final changes to the RD before the RD was submitted to the client.

- **6 November 2024 – Team Meeting**

Attendees: Draix, John, Norman, Sawyer

The team met and divided responsibilities of the document and planned future meetings with the client and team.

- **7 November 2024 – Client meeting**

Attendees: Draix, John, Norman, Sawyer, Dr. Albujasim

The team met with the client and discussed requirements for both the design document and the specifics of the project.

- **10 November 2024 – Virtual Team Meeting**

Attendees: Draix, John, Norman, Sawyer

The team reviewed independent sections and focused on completing the medium-level design.

- **14 November 2024 – In-person Team Meeting**

Attendees: Drax, John, Norman, Sawyer

The team collaboratively designed all the classes and documented their attributes and interactions within the system. Additionally, we discussed and reviewed all the tasks that needed to be completed.

B. Work Structure Breakdown

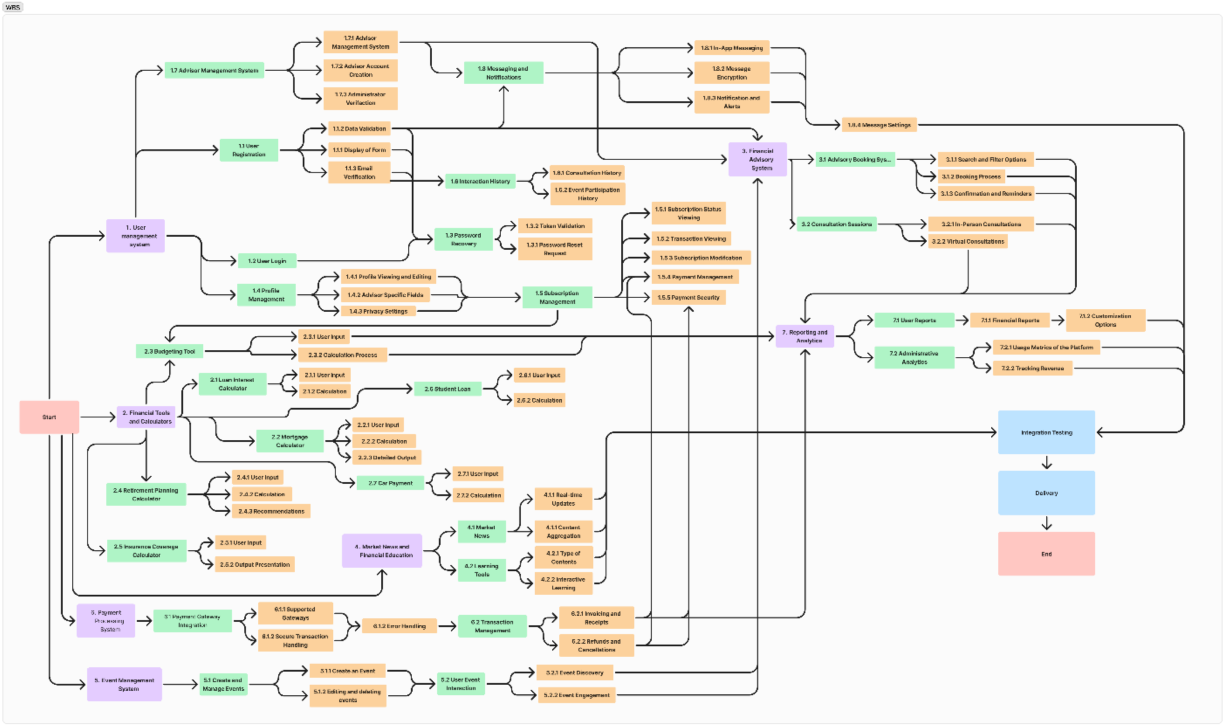
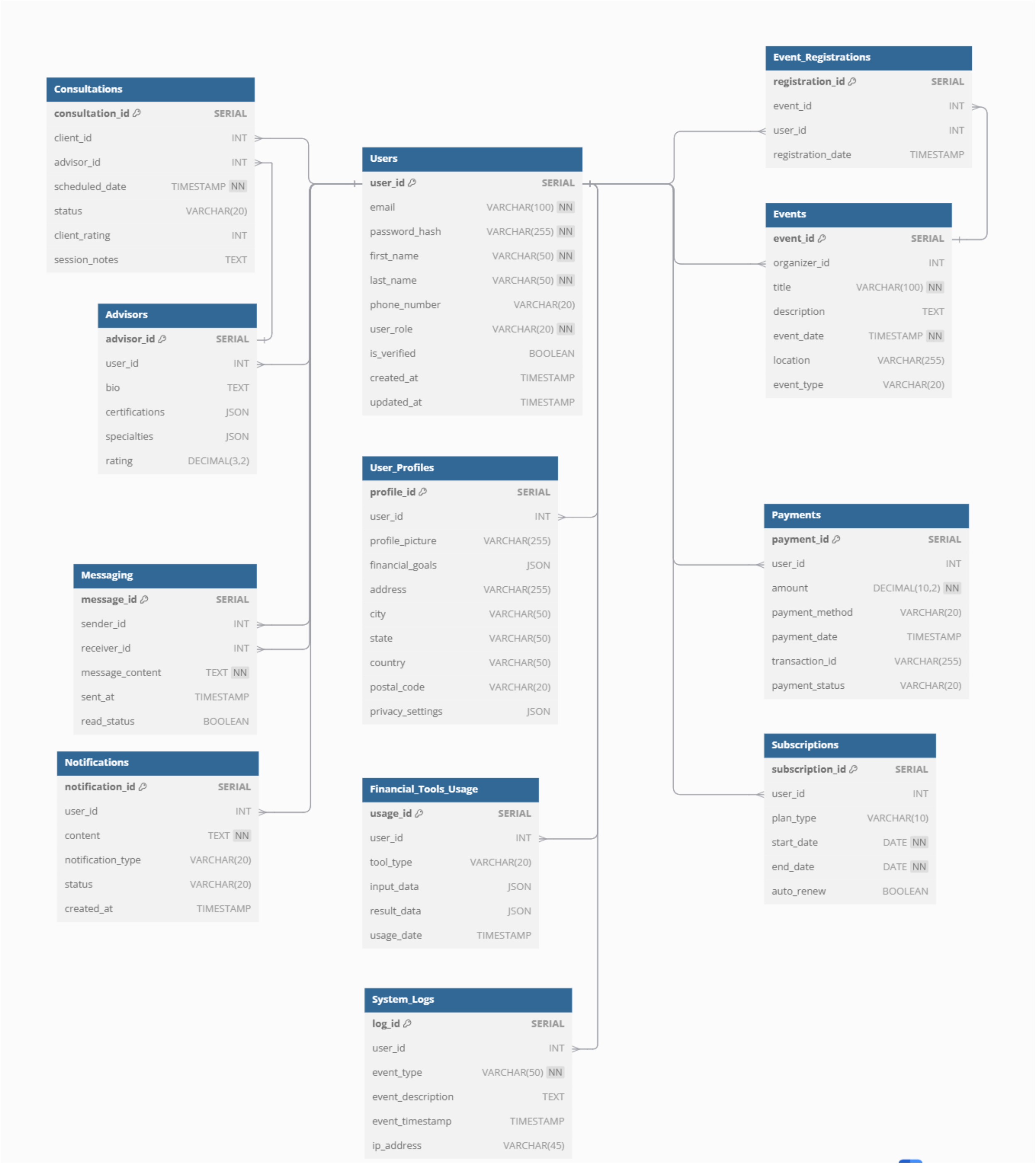

WBS
Start
1. User management system
1.1 User Registration
1.1.2 Data Validation
1.1.1 Display of Form
1.1.3 Email Verification
1.2 User Login
1.3 Password Recovery
1.3.2 Token Validation
1.3.1 Password Reset Request
1.4 Profile Management
1.4.1 Profile Viewing and Editing
1.4.2 Advisor Specific Fields
1.4.3 Privacy Settings
1.5 Subscription Management
1.5.1 Subscription Status Viewing
1.5.2 Transaction Viewing
1.5.3 Subscription Modification
1.5.4 Payment Management
1.5.5 Payment Security
1.6 Interaction History
1.6.1 Consultation History
1.6.2 Event Participation History
1.7 Advisor Management System
1.7.1 Advisor Management System
1.7.2 Advisor Account Creation
1.7.3 Administrator Verification
1.8 Messaging and Notifications
1.8.1 In-App Messaging
1.8.2 Message Encryption
1.8.3 Notification and Alerts
1.8.4 Message Settings
2. Financial Tools and Calculators
2.1 Loan Interest Calculator
2.1.1 User Input
2.1.2 Calculation
2.2 Mortgage Calculator
2.2.1 User Input
2.2.2 Calculation
2.2.3 Detailed Output
2.3 Budgeting Tool
2.3.1 User Input
2.3.2 Calculation Process
2.4 Retirement Planning Calculator
2.4.1 User Input
2.4.2 Calculation
2.4.3 Recommendations
2.5 Insurance Coverage Calculator
2.5.1 User Input
2.5.2 Output Presentation
2.6 Student Loan
2.6.1 User Input
2.6.2 Calculation
2.7 Car Payment
2.7.1 User Input
2.7.2 Calculation
3. Financial Advisory System
3.1 Advisory Booking Sys...
3.1.1 Search and Filter Options
3.1.2 Booking Process
3.1.3 Confirmation and Reminders
3.2 Consultation Sessions
3.2.1 In-Person Consultations
3.2.2 Virtual Consultations
4. Market News and Financial Education
4.1 Market News
4.1.1 Real-time Updates
4.1.1 Content Aggregation
4.2 Learning Tools
4.2.1 Type of Contents
4.2.2 Interactive Learning
5. Event Management System
5.1 Create and Manage Events
5.1.1 Create an Event
5.1.2 Editing and deleting events
5.2 User Event Interaction
5.2.1 Event Discovery
5.2.2 Event Engagement
6. Payment Processing System
6.1 Payment Gateway Integration
6.1.1 Supported Gateways
6.1.2 Secure Transaction Handling
6.1.2 Error Handling
6.2 Transaction Management
6.2.1 Invoicing and Receipts
6.2.2 Refunds and Cancellations
7. Reporting and Analytics
7.1 User Reports
7.1.1 Financial Reports
7.1.2 Customization Options
7.2 Administrative Analytics
7.2.1 Usage Metrics of the Platform
7.2.2 Tracking Revenue
Integration Testing
Delivery
End

Figure 18. Work Structure Breakdown

C. ER Diagram



Consultations		
consultation_id	SERIAL	
client_id	INT	
advisor_id	INT	
scheduled_date	TIMESTAMP	NN
status	VARCHAR(20)	
client_rating	INT	
session_notes	TEXT	

Advisors		
advisor_id	SERIAL	
user_id	INT	
bio	TEXT	
certifications	JSON	
specialties	JSON	
rating	DECIMAL(3,2)	

Messaging		
message_id	SERIAL	
sender_id	INT	
receiver_id	INT	
message_content	TEXT	NN
sent_at	TIMESTAMP	
read_status	BOOLEAN	

Notifications		
notification_id	SERIAL	
user_id	INT	
content	TEXT	NN
notification_type	VARCHAR(20)	
status	VARCHAR(20)	
created_at	TIMESTAMP	

Users		
user_id	SERIAL	
email	VARCHAR(100)	NN
password_hash	VARCHAR(255)	NN
first_name	VARCHAR(50)	NN
last_name	VARCHAR(50)	NN
phone_number	VARCHAR(20)	
user_role	VARCHAR(20)	NN
is_verified	BOOLEAN	
created_at	TIMESTAMP	
updated_at	TIMESTAMP	

User_Profiles		
profile_id	SERIAL	
user_id	INT	
profile_picture	VARCHAR(255)	
financial_goals	JSON	
address	VARCHAR(255)	
city	VARCHAR(50)	
state	VARCHAR(50)	
country	VARCHAR(50)	
postal_code	VARCHAR(20)	
privacy_settings	JSON	

Financial_Tools_Usage		
usage_id	SERIAL	
user_id	INT	
tool_type	VARCHAR(20)	
input_data	JSON	
result_data	JSON	
usage_date	TIMESTAMP	

System_Logs		
log_id	SERIAL	
user_id	INT	
event_type	VARCHAR(50)	NN
event_description	TEXT	
event_timestamp	TIMESTAMP	
ip_address	VARCHAR(45)	

Event_Registrations		
registration_id	SERIAL	
event_id	INT	
user_id	INT	
registration_date	TIMESTAMP	

Events		
event_id	SERIAL	
organizer_id	INT	
title	VARCHAR(100)	NN
description	TEXT	
event_date	TIMESTAMP	NN
location	VARCHAR(255)	
event_type	VARCHAR(20)	

Payments		
payment_id	SERIAL	
user_id	INT	
amount	DECIMAL(10,2)	NN
payment_method	VARCHAR(20)	
payment_date	TIMESTAMP	
transaction_id	VARCHAR(255)	
payment_status	VARCHAR(20)	

Subscriptions		
subscription_id	SERIAL	
user_id	INT	
plan_type	VARCHAR(10)	
start_date	DATE	NN
end_date	DATE	NN
auto_renew	BOOLEAN	

Figure 19. ER Diagram

D. Data Flow Diagram

Level 0

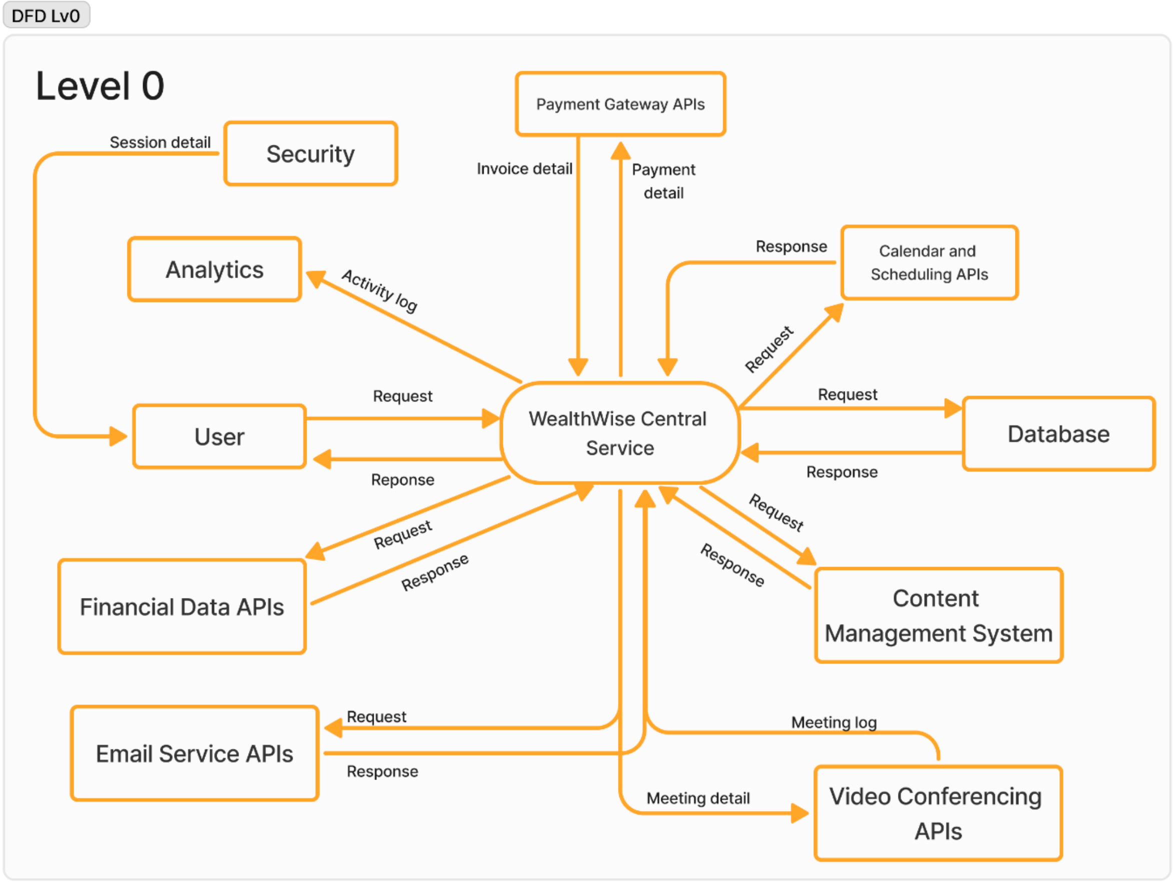


Figure 20. Level 0 Data Flow Diagram

Level 1

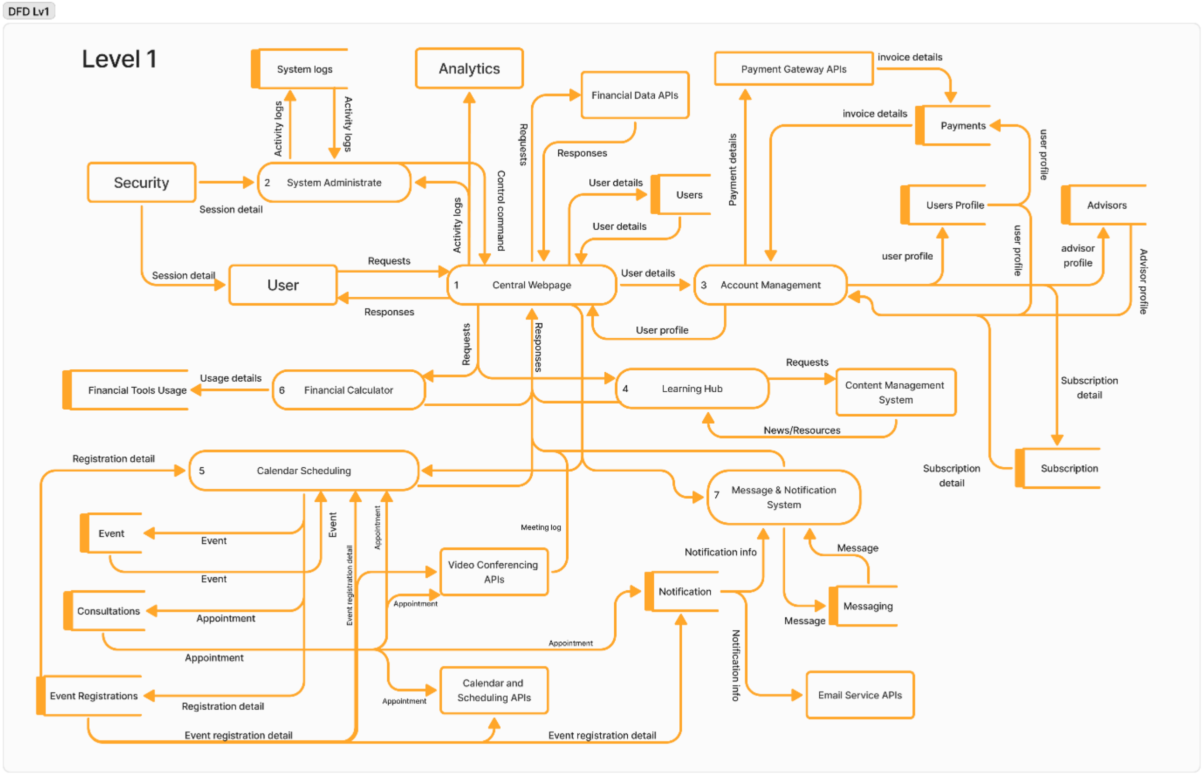


Figure 21. Level 1 Data Flow Diagram

Level 2:

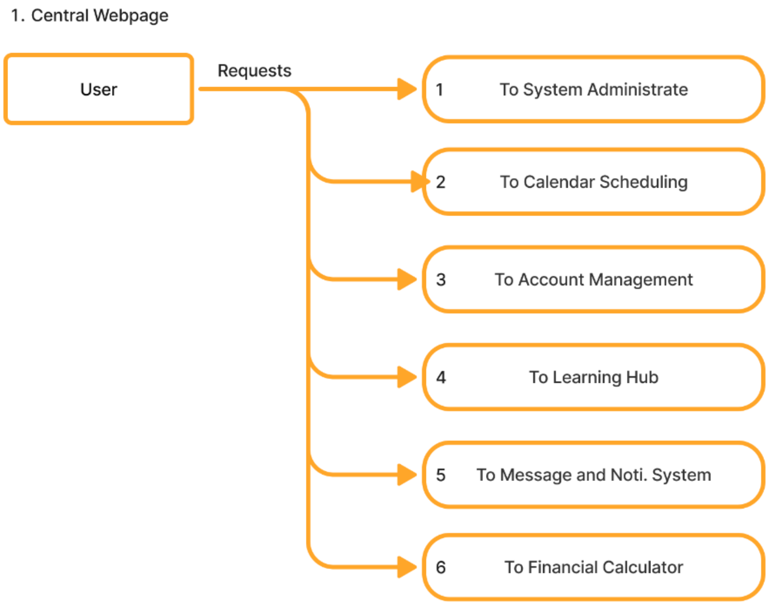


Figure 22. DFD Level 2 - Central Webpage

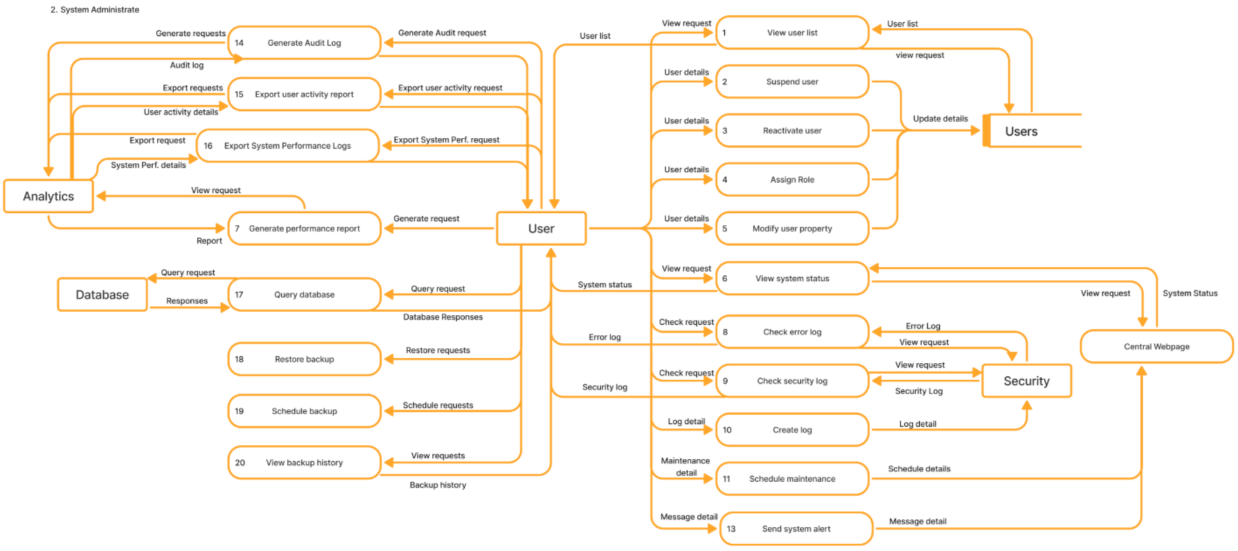


Figure 23. DFD Level 2 - System Administrate

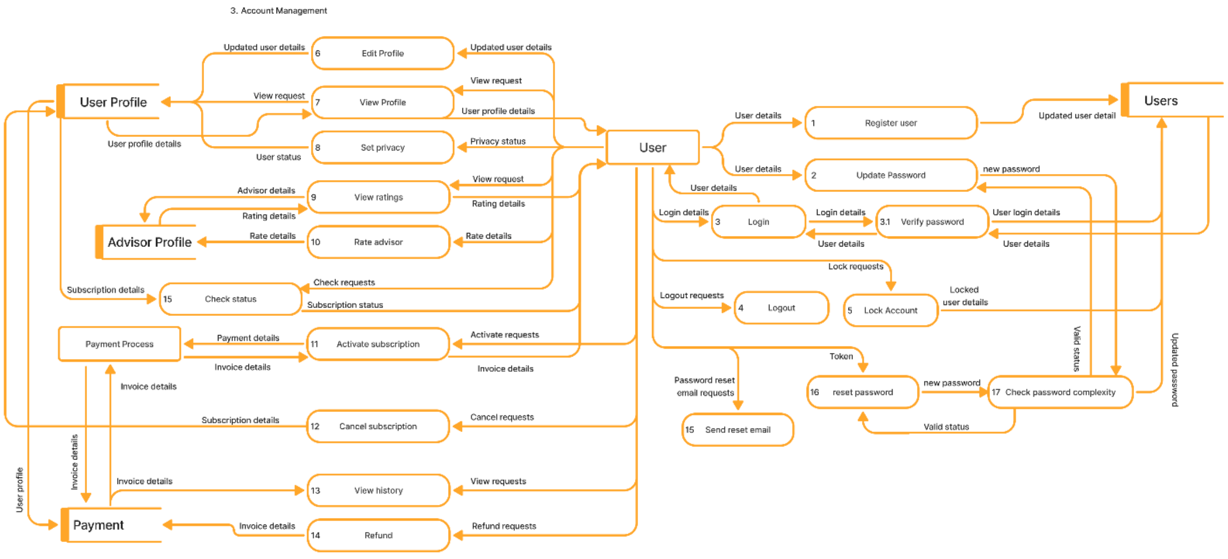


Figure 24. DFD Level 2 - Account management

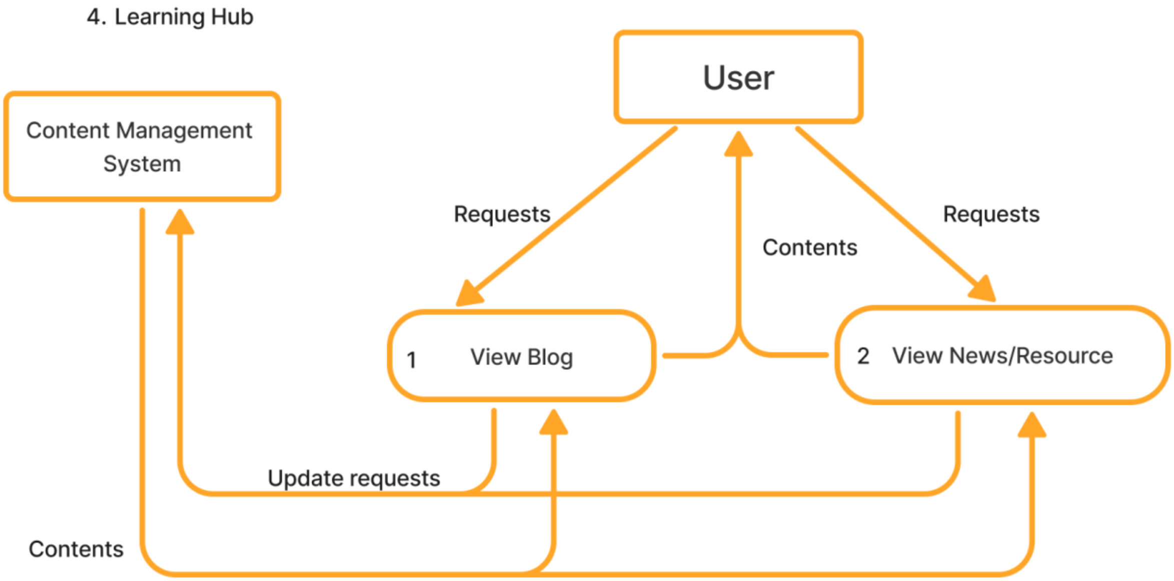


Figure 25. DFD Level 2 - Learning hub

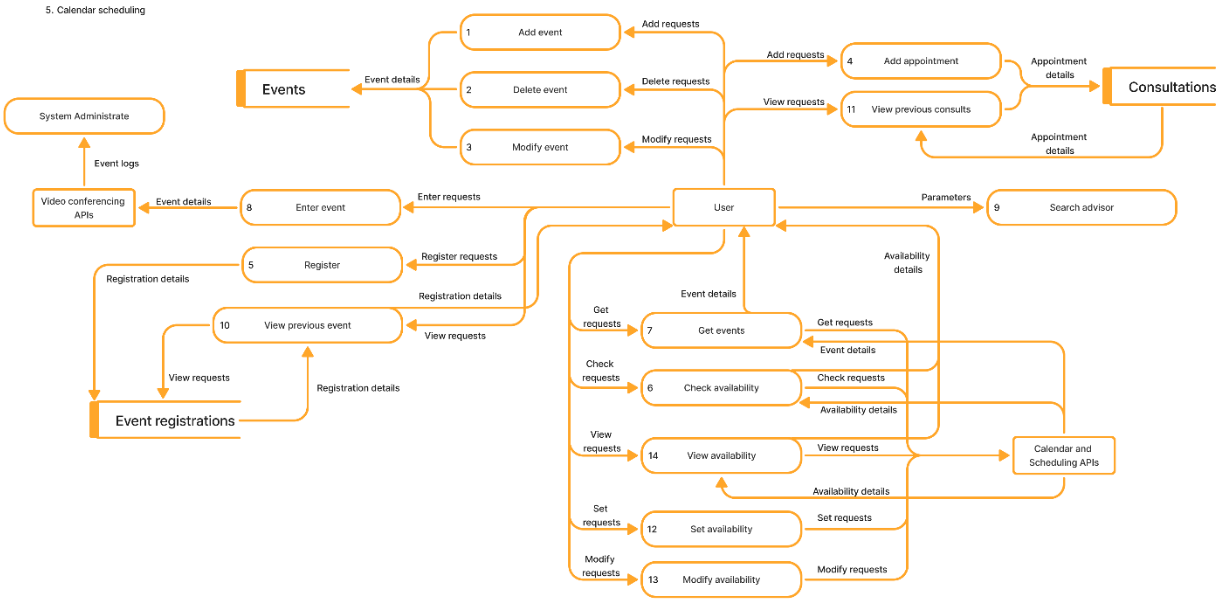


Figure 26. DFD Level 2 - Calendar scheduling

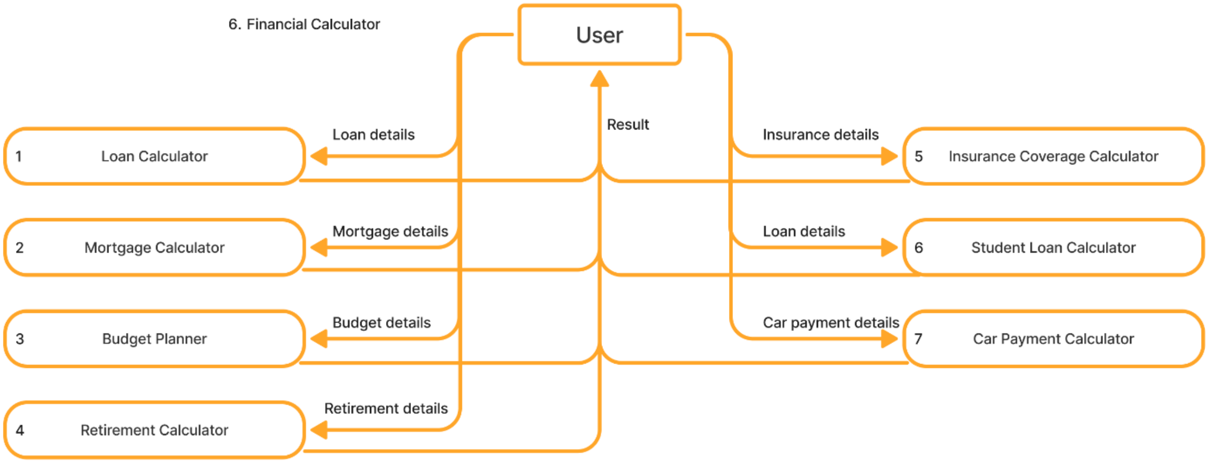


Figure 27. DFD Level 2 - Financial calculator

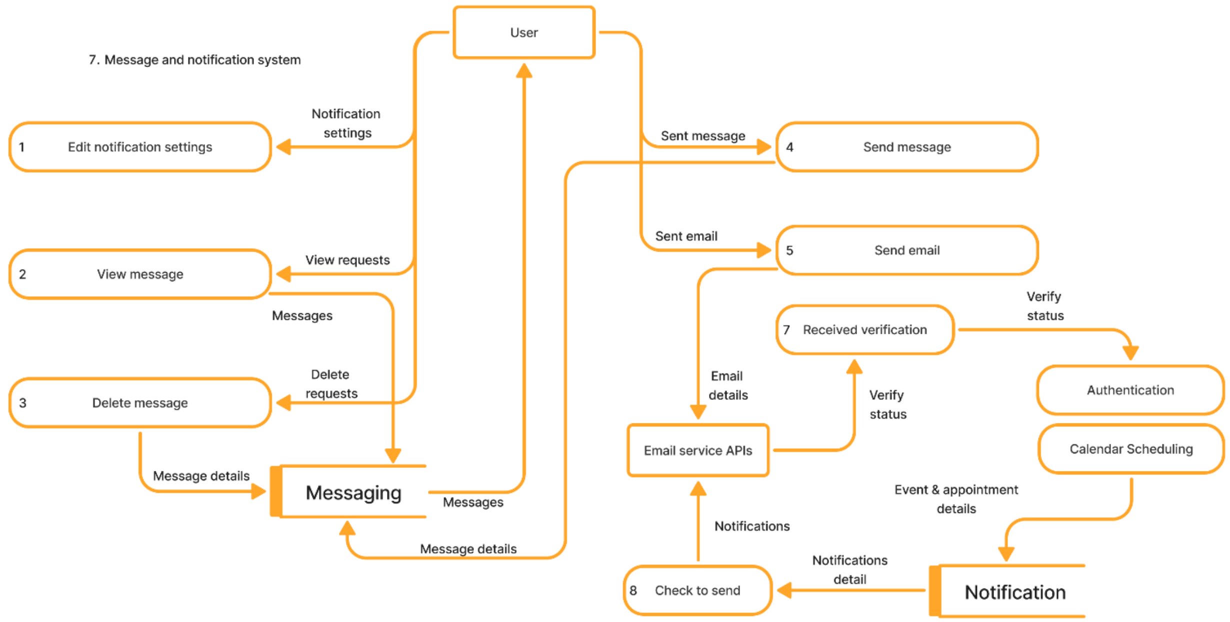


Figure 28. DFD Level 2 - Message and notification system

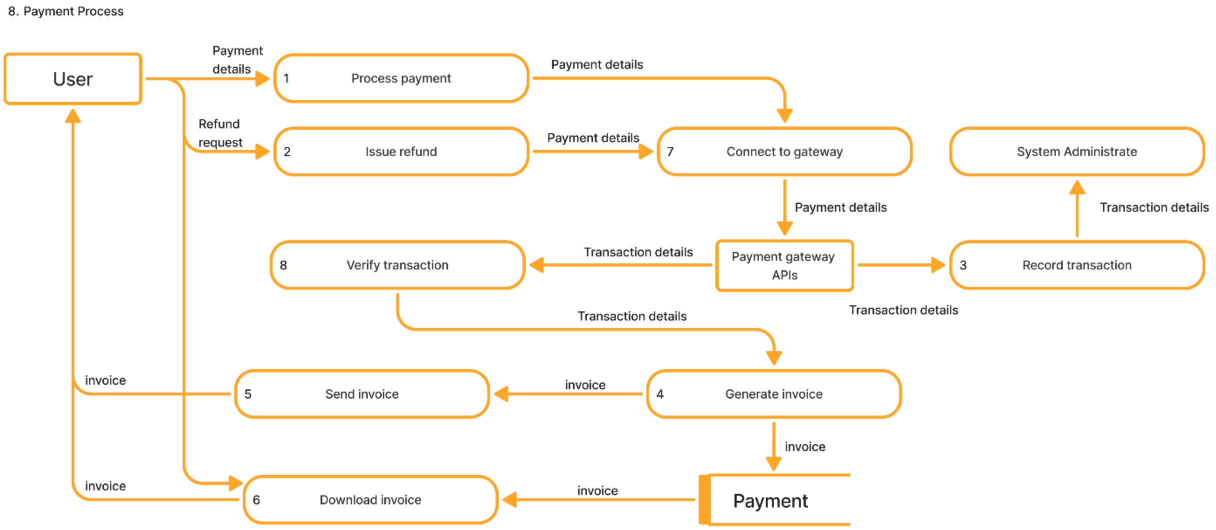


Figure 29. DFD Level 2 - Payment process