Group03

Memoir
Software Architecture Document

Version 1.7

Revision History

Date	Version	Description	Author
06/07/2025	1.1	Add use-case model, GUI for admin	Nguyễn Thị Trà My
07/07/2025	1.2	Add Architectural Goals and Constraints	Bùi Hải Long
08/07/2025	1.3	Write introduction	Bùi Hải Long
09/07/2025	1.4	GUI for User	Phạm Nam Huyên
09/07/2025	1.5	Add Logical view Add backend component Add Admin controller component	Bùi Hải Long
09/07/2025	1.6	Fill Logical view details Add User controller component Add User model component Add viewer component	Huỳnh Tuấn Phong
12/07	1.7	Add ER diagram Add database component	Bùi Hải Long

Table of Contents

Software Architecture Document

1. Introduction

1.1 Purpose

The purpose of this Software Architecture Document is to describe the high-level structure and design decisions for the Note-Taking Application. It serves as a communication tool between stakeholders—including developers, designers, and future maintainers—to ensure a shared understanding of the system's architecture. This document outlines the major architectural components, their interactions, key design goals, and the rationale behind chosen technologies and patterns.

1.2 Scope

This application is a cross-platform note-taking app initially targeting Android, with future plans for iOS and Web support. It enables users to create, edit, organize, and sync notes across devices with both online and offline functionality. The backend is powered by Supabase, providing authentication, real-time updates, and PostgreSQL storage. The architecture is designed for scalability, ease of maintenance, and a user-friendly experience.

1.3 Definition

- **UI**: User Interface
- **UX**: User Experience
- **RLS**: Row-Level Security
- **CRUD**: Create, Read, Update, Delete
- **SDK**: Software Development Kit
- **Supabase**: An open-source Backend-as-a-Service (BaaS) platform built on PostgreSQL
- **Flutter**: A UI toolkit by Google for building natively compiled applications from a single codebase
- **Dart**: Programming language used with Flutter
- **PostgreSQL**: An open-source relational database system used as the backend database

1.4 Reference

- Project vision: Detail functional and non-functional requirements
- Flutter Documentation: https://flutter.dev/docs
- Supabase Documentation: https://supabase.com/docs
- PostgreSQL Documentation: https://www.postgresql.org/docs/
- Flutter App Architecture: https://docs.flutter.dev/app-architecture

2. Architectural Goals and Constraints

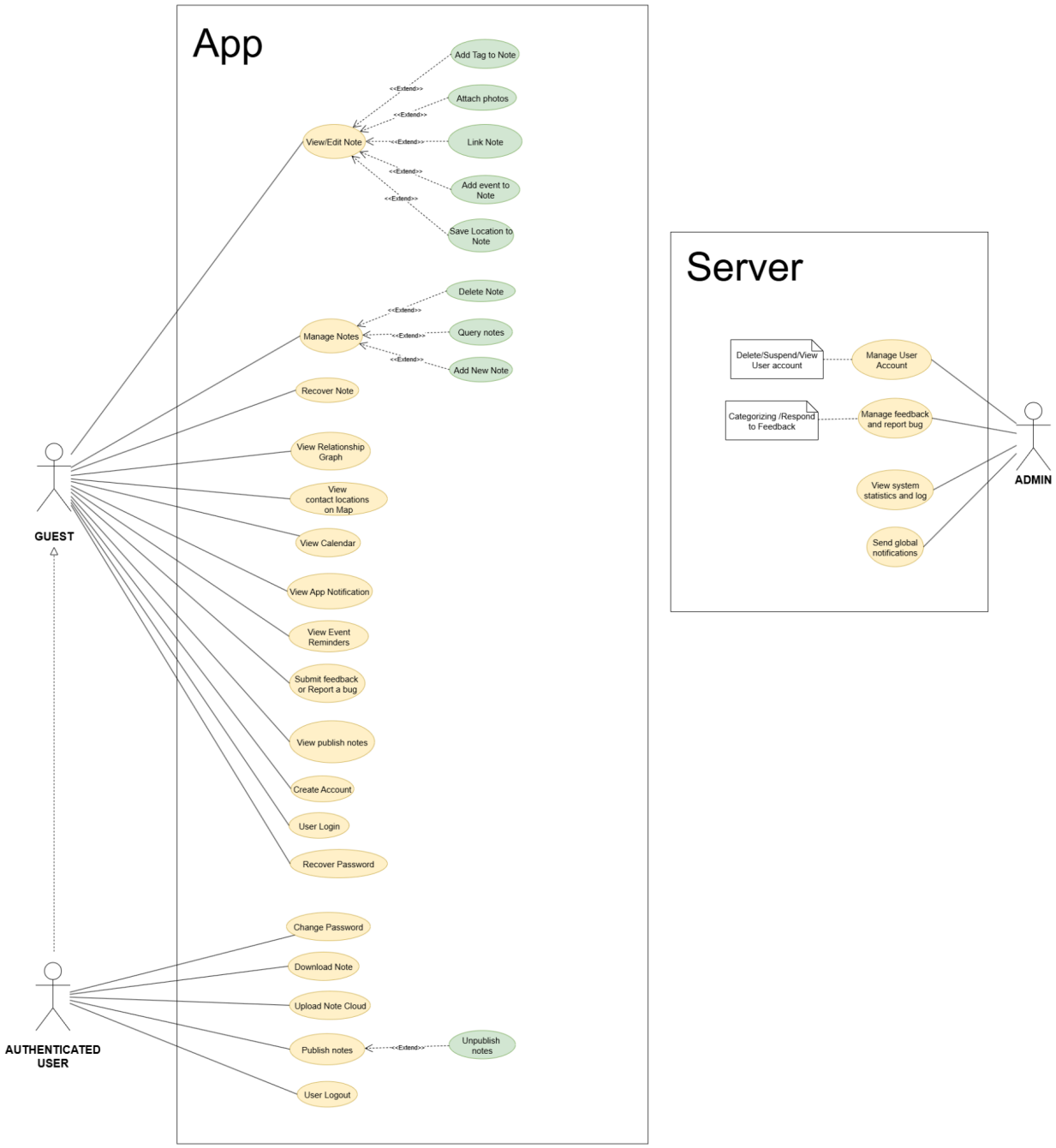
- **Offline-first capability**: The app should allow note viewing and editing while offline, and sync changes when back online.
- **Real-time data sync**: Notes should be synchronized across devices in real-time using Supabase's real-time features.
- **Secure authentication and user isolation**: Each user can only access their own notes. Authentication is handled via Supabase Auth with row-level security enforced.
- **Responsive UI/UX**: UI must be performant and responsive on various screen sizes and device types.
- **Ease of use**: The app must prioritize ease of use for non-technical users, with a clean and intuitive interface.
- **Maintainability and modularity**: Code should follow clean architecture principles to support future features like tagging, collaboration, or note sharing.

Technical constraint

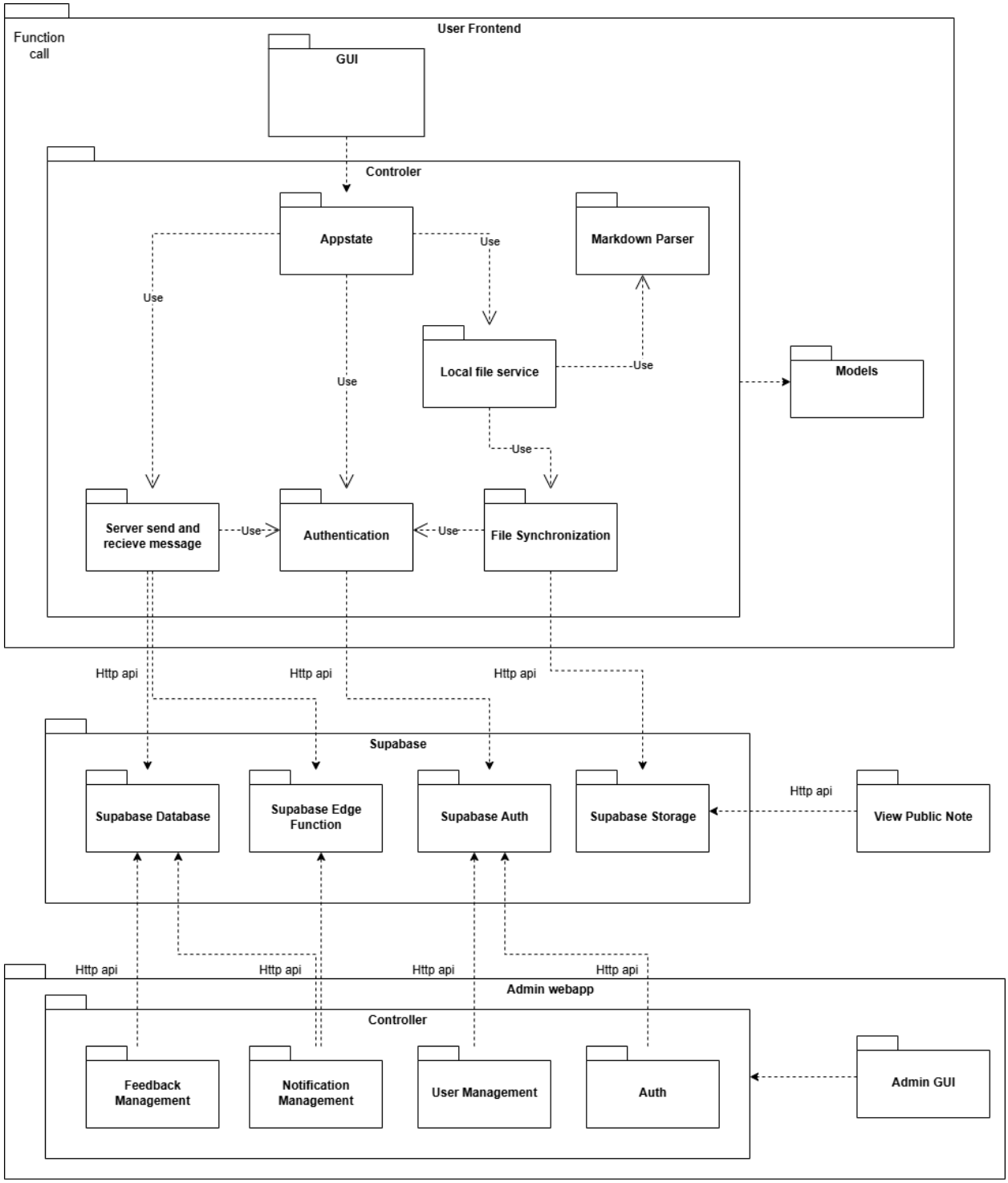
- **Application environment**: Flutter is used to ensure a single codebase that can be ported easily to iOS or Web in the future. Initial focus is Android to simplify testing and release process.
- **Programming language**: Dart is used as required by Flutter.

- **Backend**: Supabase is chosen as an off-the-shelf backend solution to reduce backend development effort and allow the team to focus on frontend and user experience.
- **Database**: PostgreSQL is used as it is the default underlying database provided by Supabase and supports advanced features like row-level security and JSON storage.

3. Use-Case Model

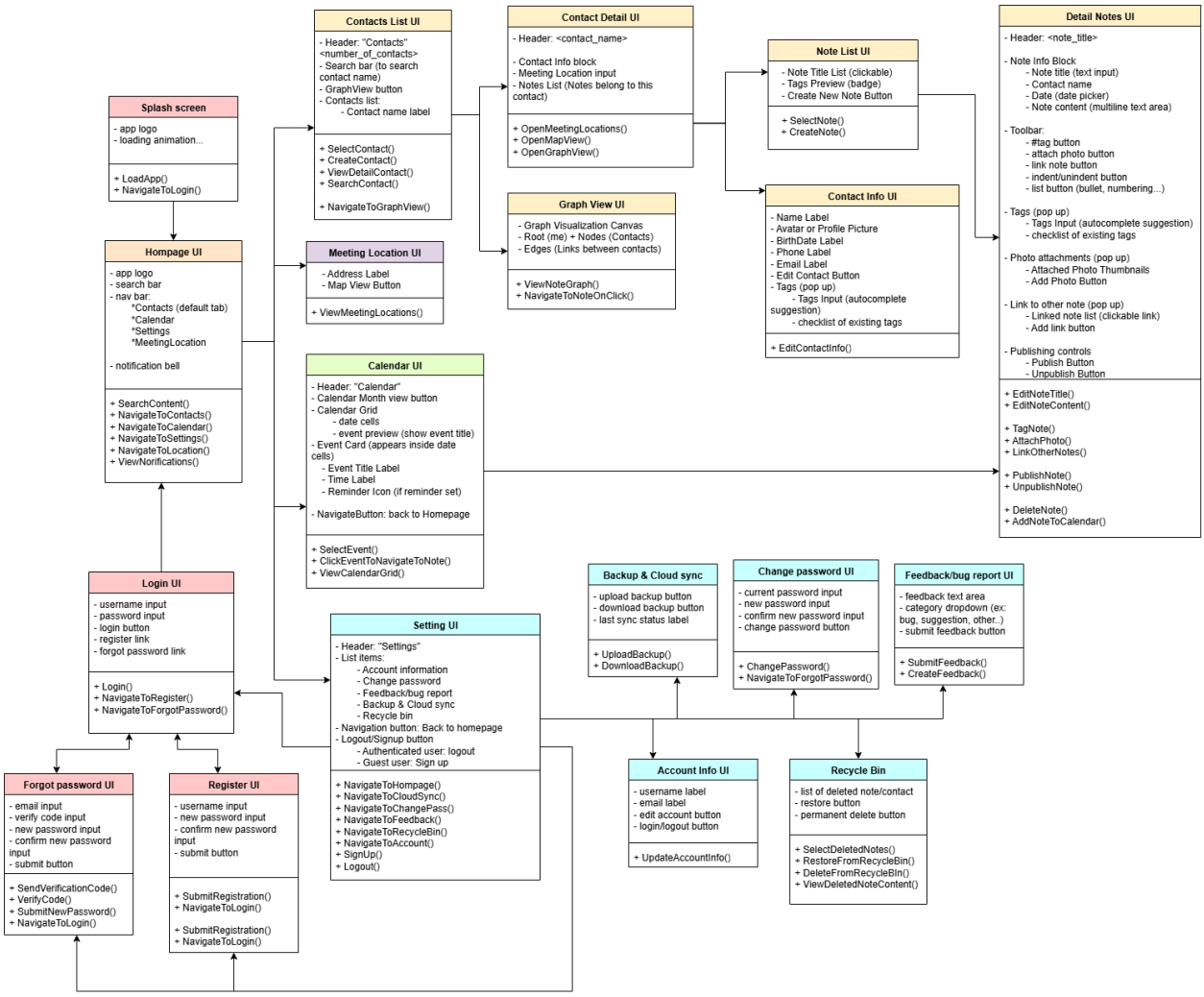


4. Logical View



4.1 User Frontend

4.1.1 Component: User GUI

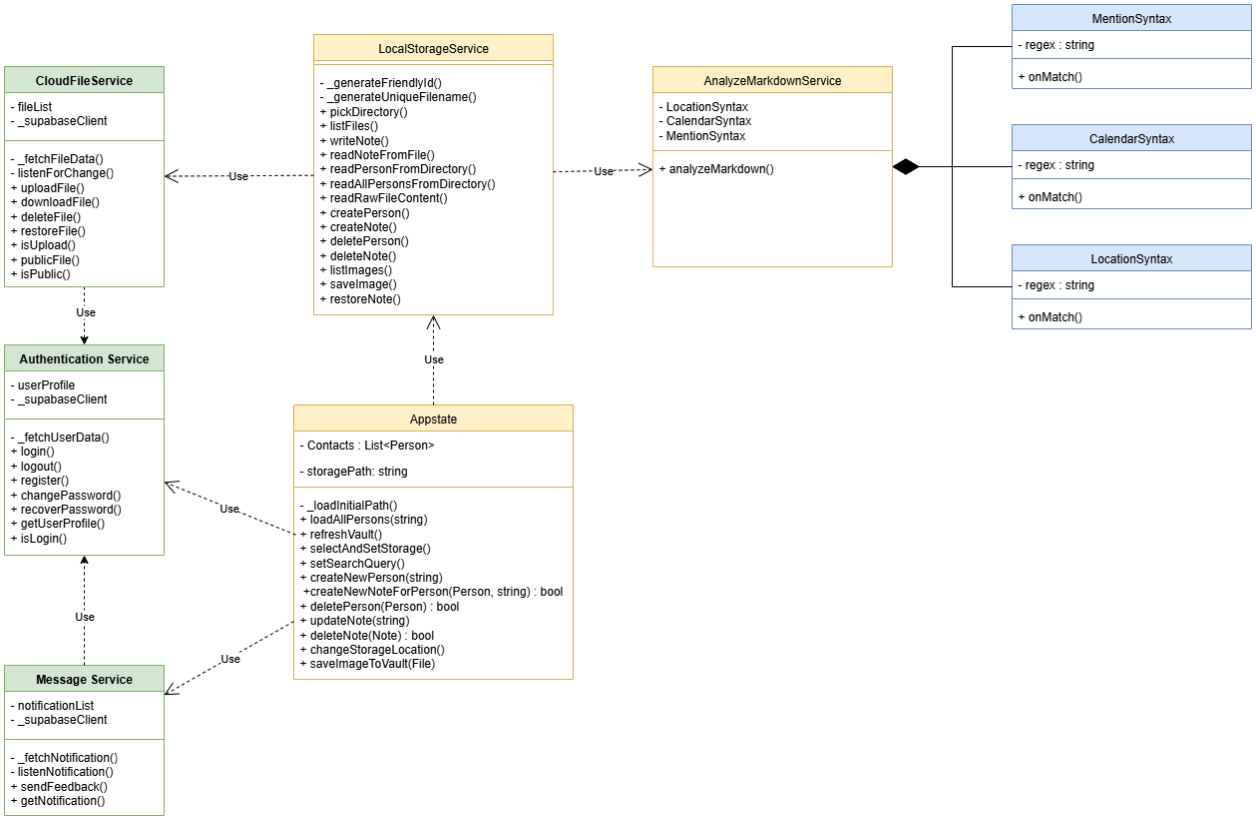


The GUI component's primary responsibility is to manage the user interface and interactions. The accompanying diagram, while not a strict UML Class Diagram, illustrates the application's screens and their navigation flow. For each screen, it details the key widgets and UI elements it will contain, as well as the essential user-triggered functions and actions available on that particular view.

- **Splash screen:** Displays the application logo and loading animation during initialization, includes an app logo and a loading animation.
- **Homepage UI:** Serves as the main screen after login. Displays the search bar, navigation bar (contacts, calendar, meetings, notes). Receives notifications (notification bell). Processing function: SearchContent(), NavigateTo...() depending on the tab.
- **Login UI:** Provides the interface for user authentication, includes input fields for username and password, along with links to register and recover password.
- **Forgot password UI:** Supports password recovery for users who have forgotten their credentials, includes email input, verification code field, and new password fields.
- **Register UI:** Allows new users to create an account, include username, password, confirm password fields, and a registration button.

- **Contacts List UI:** Displays a list of saved contacts, includes a searchable and filterable contact list, with options to add or delete contacts.
- **Contact Detail UI:** Shows detailed information about a selected contact with 2 parts: contact information block and a list of notes linked to the contact.
- **Contact Info UI:** The contact information block of `Contact Detail UI`, displays contact data within the note editing context, includes labels for name, phone number, email, and a tag input with suggestions.
- **Notes List UI:** The notes list of `Contact Detail UI`, it provides access to all created notes, includes a clickable list of note titles and a button to create a new note.
- **Note Detail UI:** Enables creation and editing of detailed note content. The user can compose or update a note, add images, tags, contacts, or share/delete the note.
- **Meeting Location UI:** Displays physical meeting locations related to contacts by an address label and an interactive map. The user can view and interact with meeting locations on the map.
- **Calendar UI:** Offers a calendar view for organizing reminders and note-related events. The user can view scheduled events and navigate to linked notes.
- **Graph view UI:** Visualizes relationships among contacts in a graph format with a node-edge graph representing contacts and their links. The user can click on nodes to navigate to related notes.
- **Setting UI:** Central hub for application and account settings (password change, backup, feedback, account info, and logout)
- **Account Info UI:** Displays user account details, with labels for username and email, with options to edit or log out
- **Backup & Cloud sync UI:** Facilitates cloud synchronization and data backup with buttons for uploading and downloading backup files. The user can upload current data to the cloud or restore from a backup.
- **Change Password UI:** enables users to update their login password with a fields for current password, new password, and password confirmation.
- **Feedbacks/bugs report UI:** Collects user feedback or bug reports, includes a text input for feedback and an option to attach files.
- **Recycle Bin UI:** Manages deleted notes with options for restoration or permanent removal. The user can view, restore, or permanently delete notes from the bin

4.1.2 Component: Controller



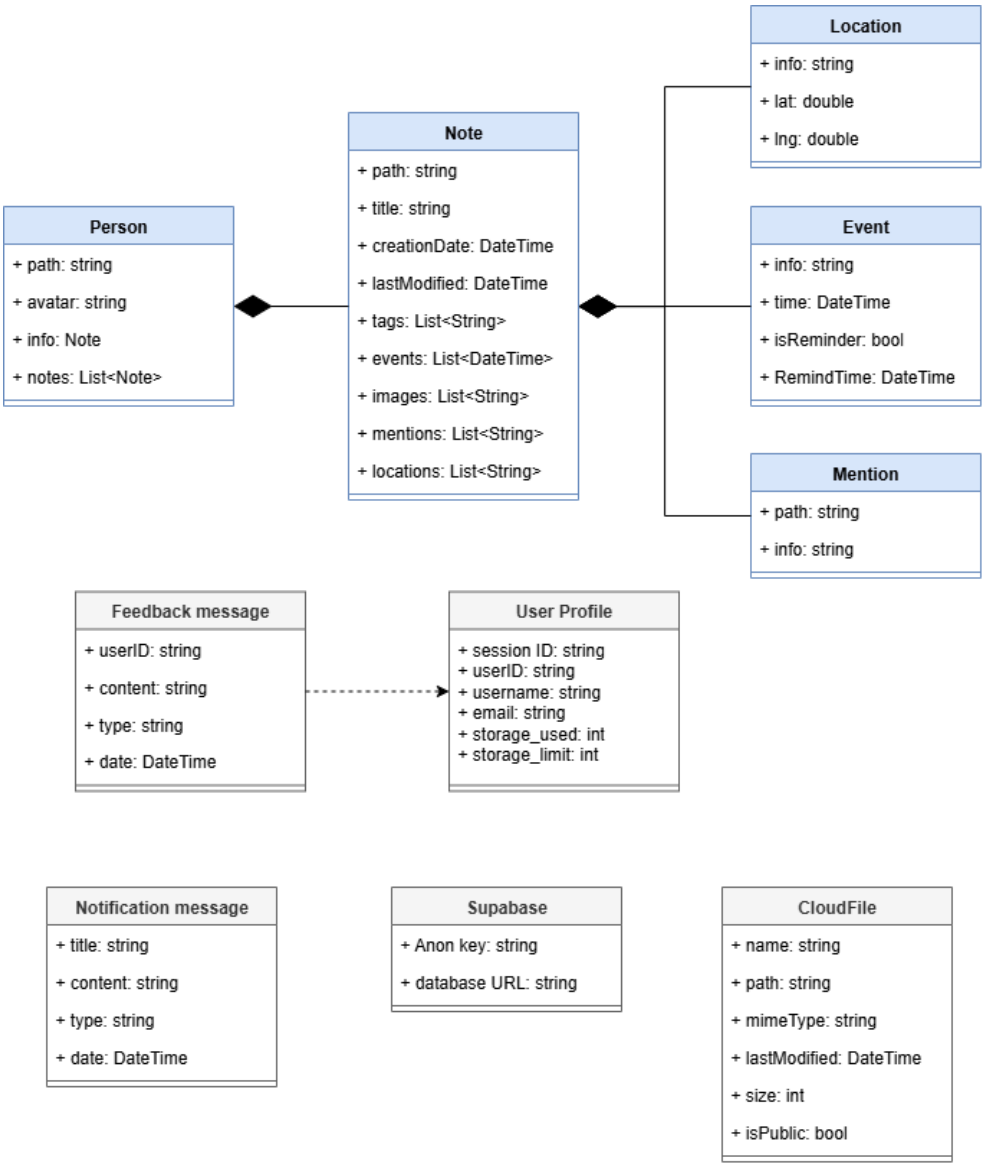
The controller is the core of the memoir app, used for handling the business logic and I\O. The yellow colored classes are the main classes needed for the entire app both offline and online,

- **Appstate** is used to provide and contain models that are the source of truth for the user interface to interact with.
- **LocalStorageService** is used to talk with the local file (mostly markdowns and images) of the user device.
- **AnalyzeMarkdownService** is needed to parse the markdown for important data that we need like mentions, events, geolocation and can be extended easily by adding the syntax we want as class to extend our parsing needs in the future.

The green colored classes are the classes needed for the online features:

- **CloudFileService** is used to synchronize files between local and cloud.
- **AuthenticationService** is used to handle login sessions of a user and is a must for user safety.
- **MessageService** is used to handle feedback and notifications from and to the server.

4.1.3 Component: Model

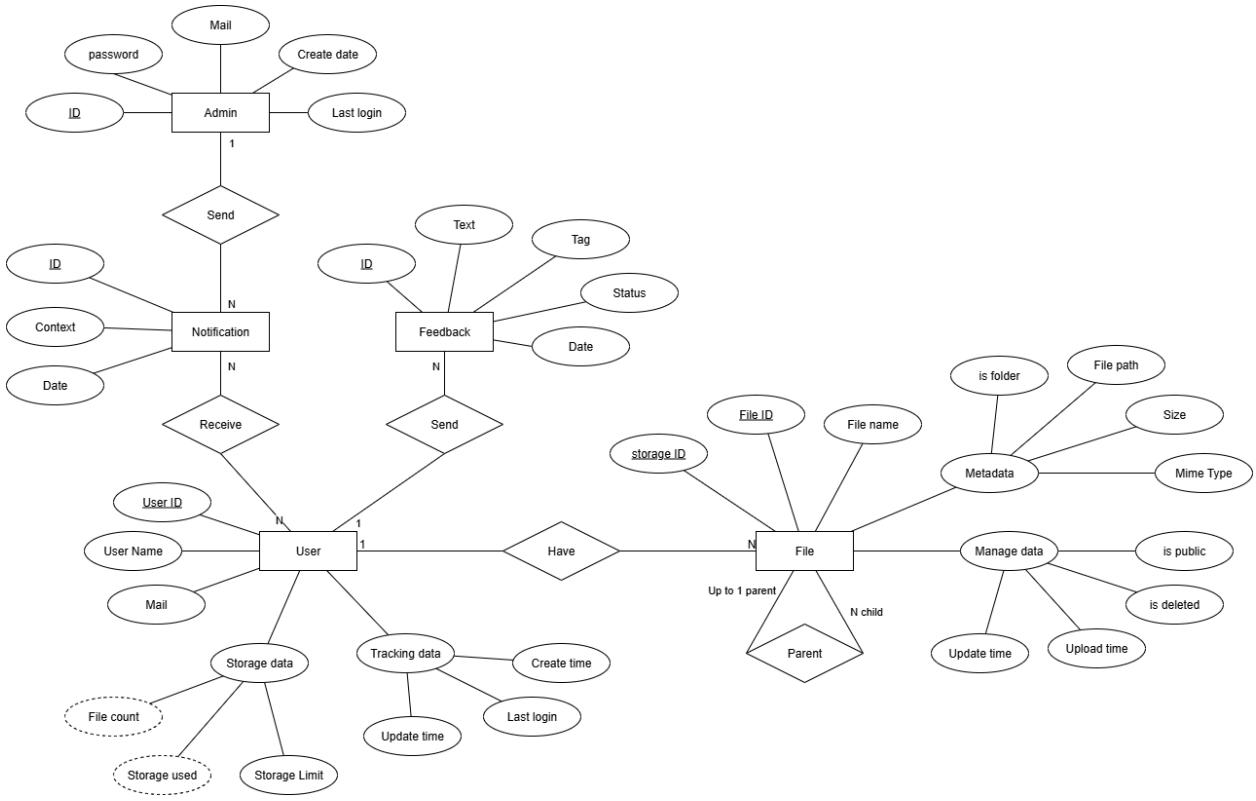


These are classes that represent the model the entire app operates on. In blue are models that core and does not require online connection while grey are for online functionality.

- **Person**: contain many notes and a special information note, representing a contact.
- **Note**: contains a lot of data for quality of life use cases, this class is very closely aligned with the markdown file we stored in the user device.
- **Location, Event, Mention**: these are what I called data fields and are extracted from the markdown content.
- **Supabase**: contain key and URL link to connect to supabase server
- **FeedbackMessage:** contain information about a user feedback
- **UserProfile:** contain information for a user as well as session information to communicate with supabase
- **CloudFile:** contains necessary information about file that user have upload to the cloud
- **NotificationMessage:** contains information for notification that receive from admin

4.2 Backend

4.2.1 Component: Database design

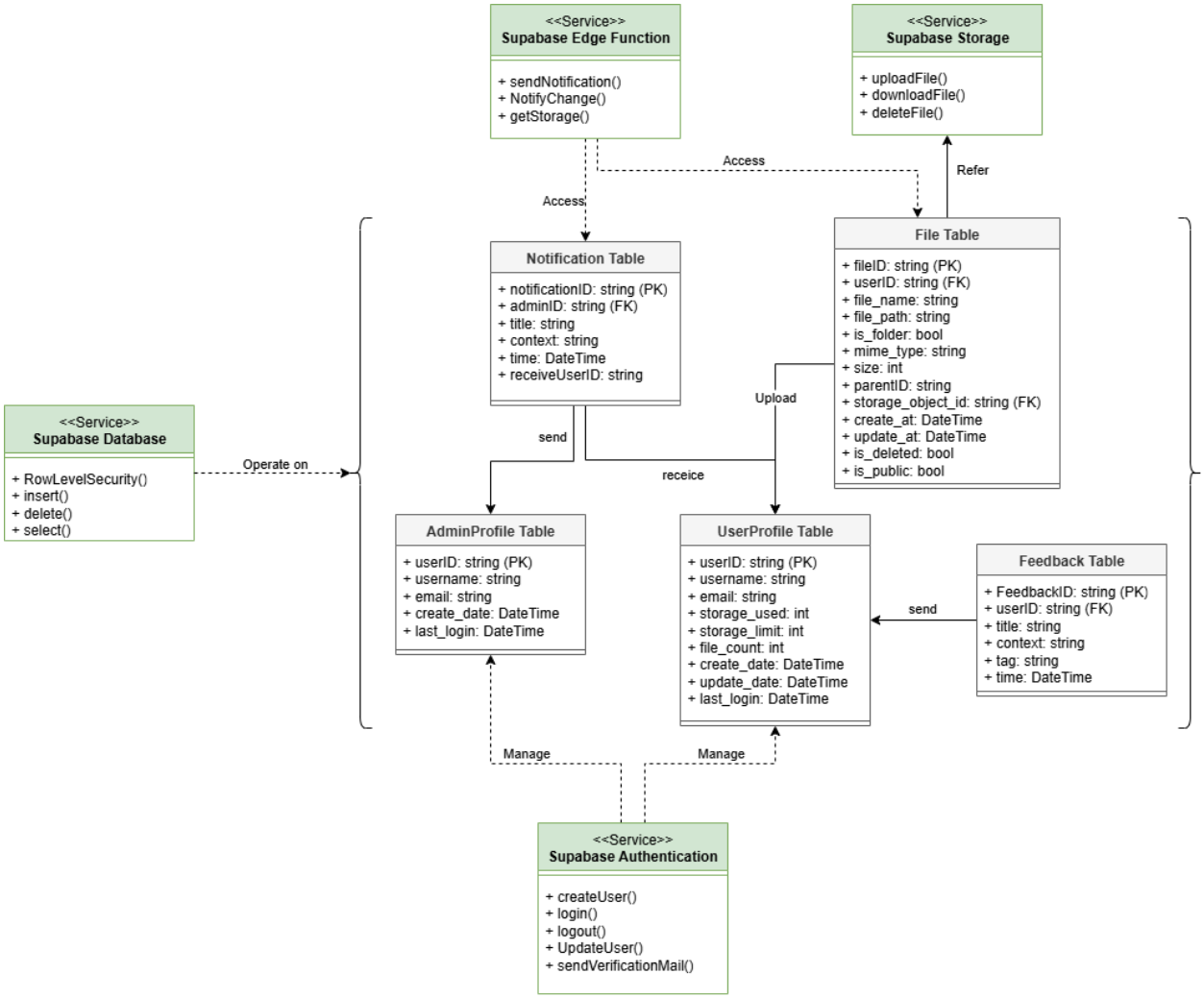


The Entity-Relationship (ER) model visually represents the conceptual structure of our application's data. It identifies the key entities (data objects), their attributes (properties), and the relationships between them. This model serves as the foundational, platform-agnostic design for our database schema, upon which our Supabase implementation is built.

- **Entities and Key Attributes:**
  - **Admin**: Represents application administrators. Key attributes include ID (Primary Key), Mail, password, Create date, and Last login.
  - **User**: Represents regular users of the application. Key attributes include User ID (Primary Key), User Name, Mail, File count, Storage used, Storage Limit, Update time, and Date.
  - **Notification**: Represents system-generated messages. Key attributes include ID (Primary Key), Context, and Date.
  - **Feedback**: Represents user-submitted feedback or bug reports. Key attributes include ID (Primary Key), Context, Tag, Date, and Status.
  - **File**: Represents metadata for files uploaded by users. Key attributes include File ID (Primary Key), File name, storage ID, File path, is folder, Mime Type, Size, is public, Upload time, Update time, and is deleted.
- **Relationships:**
  - **Admin sends Notification**: A one-to-many relationship, where one Admin can Send many Notification messages.
  - **User receives Notification**: A one-to-many relationship, where a User can Receive many Notification messages.

- **User sends Feedback**: A one-to-many relationship, where a User can Send many Feedback messages.
- **User has File**: A one-to-many relationship, where a User can Have many Files.
- **File Parent-Child**: A recursive relationship where a File can have Up to 1 parent and N child files, indicating folder structures or nested files.

4.2.2 Component: Supabase Service



The supabase backend component responsibility is to provide backend service for the app such as authentication and cloud file storage. The accompanying diagram, while structured as a class diagram, represents how we've designed and integrated the services provided by Supabase with our database tables.

The green class represent service that supabase provide:

- **Supabase Database:** This service provides the core relational database capabilities for the application. It handles all persistent data storage, including user profiles, contact information, notes, files, notifications, and feedback.
- **Supabase Edge Function:** This service hosts serverless functions (written in TypeScript/JavaScript) that extend Supabase's capabilities. We use it for sending notifications to users.
- **Supabase Authentication:** This service manages all user authentication and authorization aspects of the application. It handles user registration, login, session management, password recovery, and email verification.

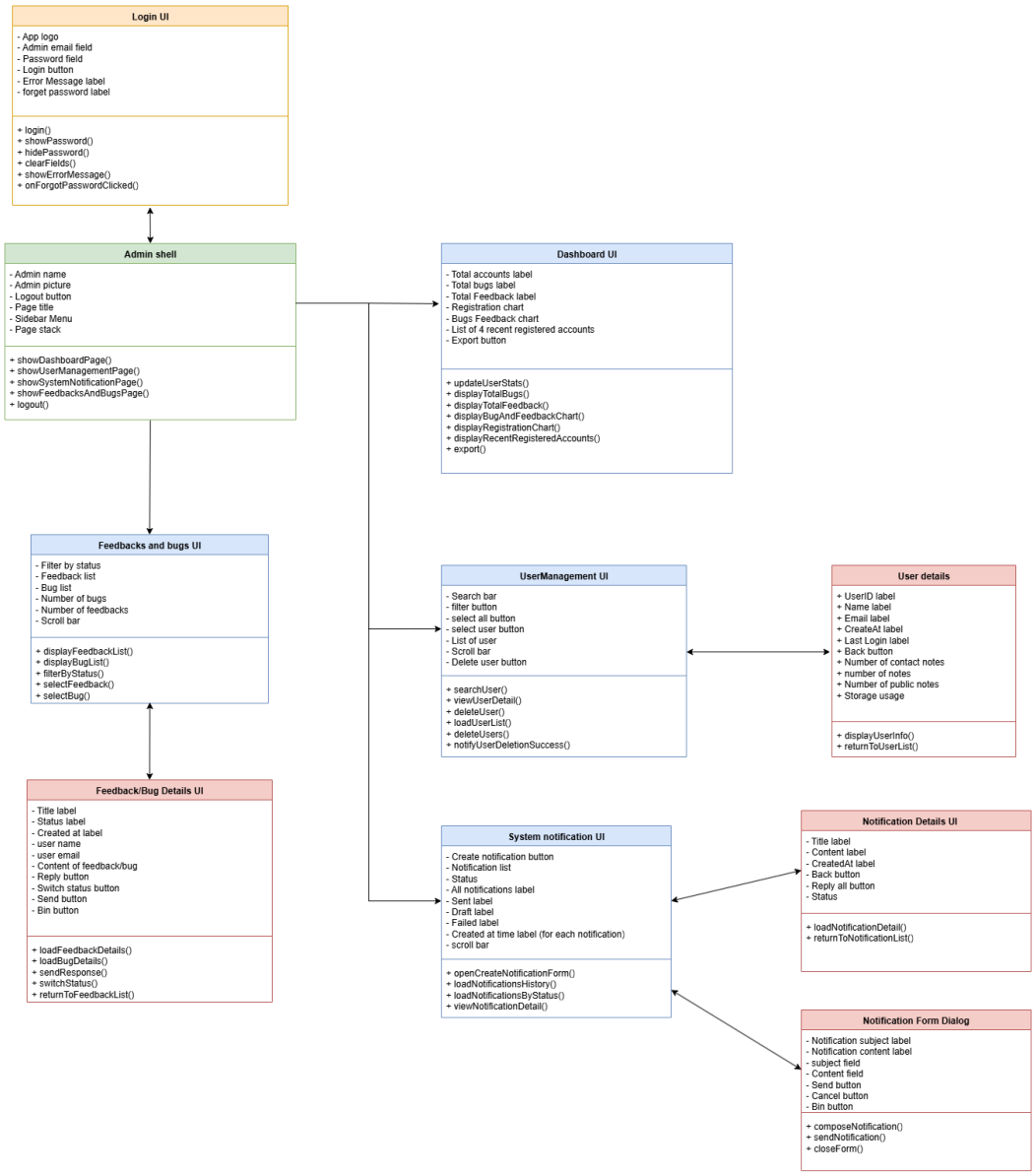
- **Supabase Storage:** This service provides object storage for files associated with the application. It's where images attached to notes and potentially other user-uploaded files are stored securely.

The grey class represent the SQL table schema for the data that the supabase backend operate on:

- **AdminProfile:** Stores profile information specifically for administrative users of the application. This table is linked to the Supabase Authentication service for admin login.
- **UserProfile:** Stores profile information for regular users of the note-taking application. This table is linked to the Supabase Authentication service for user login and manages user-specific data like storage limits.
- **Notification:** Stores all system-generated notifications intended for users or administrators, such as updates or maintenance alerts.
- **Files:** Stores metadata about files uploaded to Supabase Storage, linking them to specific users and notes. This table allows tracking file ownership, paths, and public status.
- **Feedbacks:** Stores user feedback submitted through the application, allowing administrators to review and manage it.

4.3 Admin Frontend

4.3.1 Component: Admin GUI



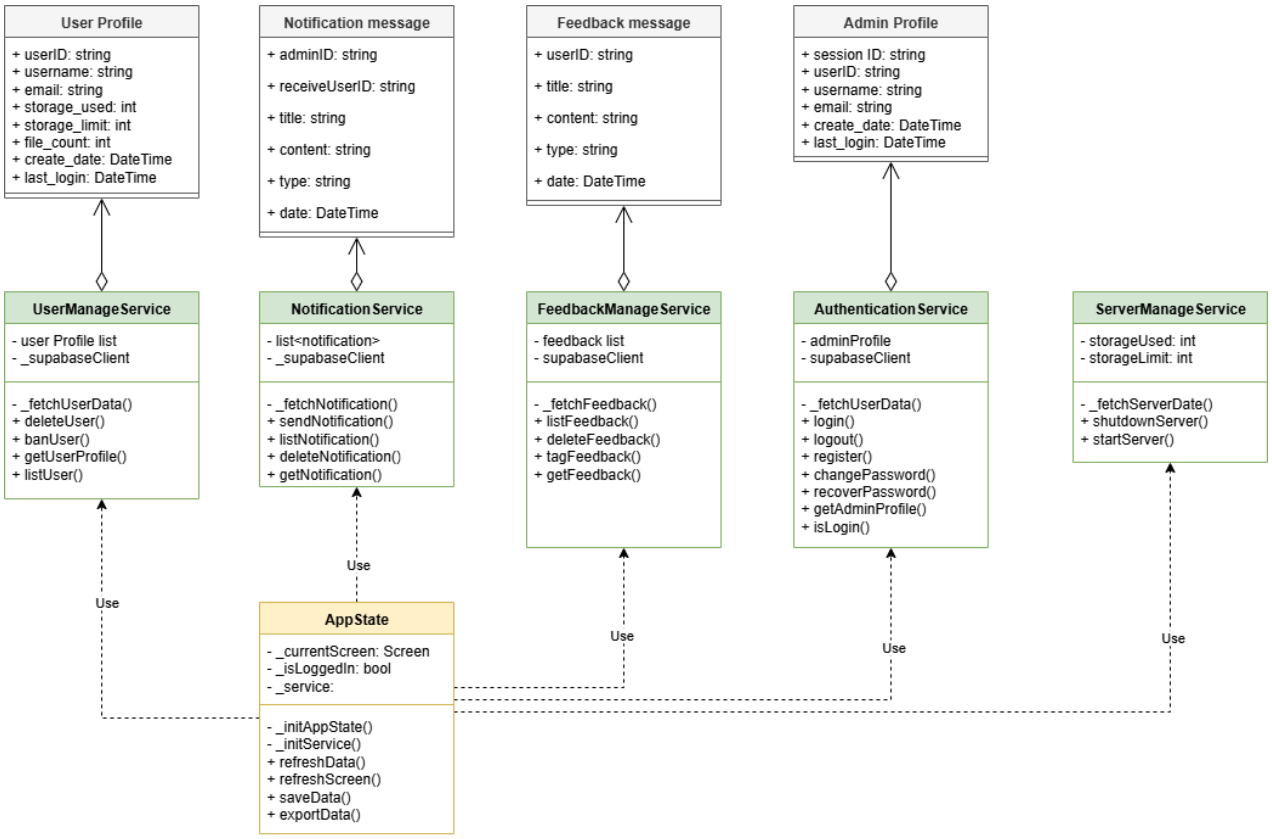
The Admin GUI component is responsible for presenting the administrative user interface. The diagram illustrates the various view screens within the Admin UI, detailing the elements to be rendered on each, along with the functionalities available to the administrator on those screens.

- **Login screen**: contains email and password fields, a login button and a forgot password link. Successful login navigates to Admin shell and Dashboard screen.
- **Admin shell**: is the main interface shown after login, featuring the app logo, admin name, logout button, page title, and a navigation menu (Dashboard, Users, Notifications, Feedbacks). This interface displays admin screens, manages navigation, opens the Dashboard by default, and allows the admin to switch pages or log out.
- **Dashboard screen**: displays a system overview, including total accounts, bugs, and feedback, a registration

chart, a bugs/feedbacks chart, a list of the 4 most recently registered accounts, and an export button.

- **User Management screen**: From the AdminShell sidebar, the admin selects “User Management” to display a central screen for managing user accounts. This screen includes a search bar, filter and select buttons, a scrollable user list, and a delete button.
- **User details screen**: displays specific information of the selected user, including User ID, name, email, account creation date, last login, number of contact notes, total notes, public notes, and storage usage. The screen also provides a back button to return to the User Management page. Accessed when the admin clicks on a user in the user list.
- **System notification screen**: allows the admin to manage system-wide announcements. The screen provides a create notification button, a notification list, labels for four categories (All Notifications, Sent, Draft, Failed), a created-at timestamp for each item, and a scroll bar. Admin accesses this screen via the sidebar in Admin Shell.
- **Notification details screen**: displays the title, content, creation time, status, a back button, and a reply-all button. The screen opens when the admin selects a notification from the list in the System notification screen.
- **Notification form dialog**: provides fields for notification subject and content, along with a send button, cancel button, and bin button. When the admin clicks the create notification button in the System notification screen, this form appears. Choosing "Cancel" saves the notification as a draft; selecting the bin button discards it.
- **Feedbacks and bugs screen**: displays feedback and bug reports submitted by users. The screen includes filters by status, separate lists for bugs and feedback, the number of each and a scroll bar. Admin accesses this screen via the sidebar in Admin Shell.
- **Feedback/Bug details screen**: shows the report title, status, creation time, user name, user email, and content. The screen also includes a reply button, switch status button, send button, and bin button. This screen opens when the admin selects a report from the Feedbacks and Bugs screen.

4.3.2 Component: Admin Controller



The admin controller component, like the user controller component, is used to handle business logic for the admin GUI:

The grey class serve as a model for the data that the web app operate on:

- **UserProfile**: This class represents the data structure for a user's profile, including details relevant to administration. It's used within the UserManageService to fetch and display user information.
- **NotificationMessage:** This class defines the structure for notifications sent to users. It's used by the NotificationService to manage and send messages.
- **FeedbackMessage**: This class defines the structure for feedback submitted by users. It's managed by the FeedbackManageService.
- **AdminProfile**: This class represents the data structure for an administrator's profile, used for authentication and tracking admin sessions. It is managed by the AuthenticationService.

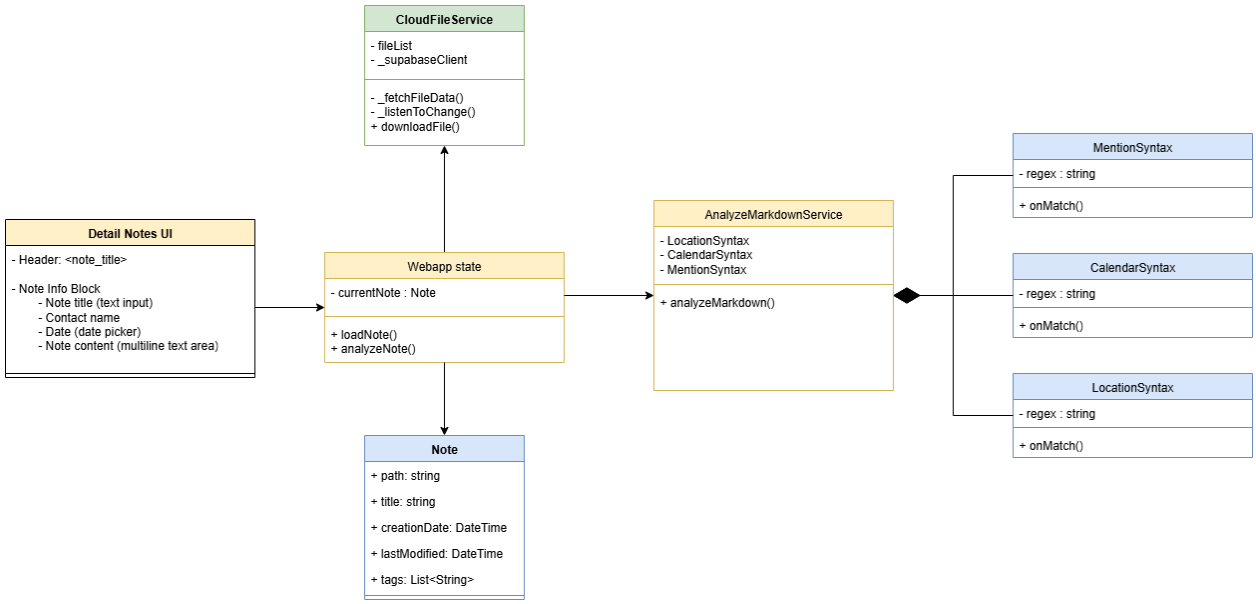
The green class act as a service for each main feature of admin web application:

- **UserManageService:** This service handles all operations related to managing user accounts from the administrator's perspective. It retrieves user data from the Supabase backend.
- **NotificationService**: This service manages the creation, sending, listing, and deletion of notifications for the application. It interacts with the Supabase backend to store and retrieve notification messages.
- **FeedbackManageService**: This service is responsible for managing user feedback, including listing, deleting, tagging, and retrieving individual feedback entries.
- **AuthenticationService**: This service handles administrator authentication processes, including login, logout, registration, password changes, and recovery. It manages the admin's session.
- **ServerManageService:** This service is responsible for providing administrators with information about server status and potentially controlling basic server operations.

The yellow class is the app state:

- **AppState:** serves as the central, single source of truth for the entire web application's state. It is responsible for managing the current UI state, navigation flows, and potentially holding global data that needs to be accessible across different parts of the Admin GUI.

4.4 Component: View Public Note



This is for the viewers to see published notes on their browsers; most classes are the reduced version of the User frontend.

- **CloudFileService:** service to communicate with supabase backend to fetch and download published notes to serve users.
- **AnalyzeMarkdownService:** to analyze special syntax in the markdown note file to render the note correctly
- **MentionSyntax, CalendarSyntax, LocationSyntax:** class represent special syntax a markdown note can have.
- **DetailNotesUI:** use to render the note to user view screen
- **WebAppState:** manage user view
- **Note:** class to hold note quality of life use case

5. Deployment

6. Implementation View