DUGBET SOFTWARE ARCHITECTURE DOCUMENT

Group 03

Huynh Huu Phuc

Tu Canh Minh

Nguyen Truc Nhu Binh

Nguyen Duc Hung

Hoang Nhu Vinh

Version 1.1

Revision History

Date	Version	Description	Author
30/11/2023	1.1	Initial version, Reformat	Nguyen Truc Nhu Binh
1/12/2023	1.2	Draft document (section 1, 2,3)	Nguyen Duc Hung
2/12/2023	1.3	Add class diagram and its description	Huynh Huu Phuc
2/12/2023	1.4	Write detail description for each components diagrams	Hoang Nhu Vinh
2/12/2023	1.4	Add package diagram and class diagrams	Tu Canh Minh

Table of Contents

1. Introduction **4**
1.1 Purpose: 4
1.2 Scope: 4
1.3 Definitions, Acronyms, and Abbreviations: 4
1.4 Overview: 4
2. Architectural Goals and Constraints **5**
2.1 Applicable Standards 5
2.2 System Requirements 5
2.3 Performance Requirements 5
2.4 Environmental Requirements 5
2.5 Documentation Requirements 5
3. Use-Case Model **7**
4. Logical View **8**
4.1 Class diagram 8
4.2 Component: Authentication 11
4.3 Component: User 12
4.4 Component: Event 14
4.5 Component: Wallet 15
4.6 Component: BudgetTarget 17
5. Deployment **18**
6. Implementation View **18**

1. Introduction

1.1 Purpose:

- This document provides a comprehensive architectural overview of our money management application (Dugbet). The important architectural choices that have been made for the system are easily shown and understood.

1.2 Scope:

- This Software Architecture Document applies to Dugbet developed using the Flutter framework with Getx for state management, and Firebase as the backend database. The application is designed based on the Model-View-Controller (MVC) architecture.

1.3 Definitions, Acronyms, and Abbreviations:

- Flutter is an open-source UI software development kit created by Google. It is used to develop cross platform applications from a single codebase for any web browser, Fuchsia, Android, iOS, Linux, mac...
- GetX: A lightweight and powerful Flutter solution. It quickly and easily integrates high performance state management, intelligent dependency injection, and route management.
- Firebase: a set of backend cloud computing services and application development platforms provided by Google.
- MVC (Model–view–controller) is a software design pattern commonly used for developing user interfaces that divides the related program logic into three interconnected elements

1.4 Overview:

- The next sections of this document will go into the architecture of Dugbet application in depth, including the system's context and environmental view, architectural and component-level views, and any associated architectural styles or patterns that were employed.
  References:
  Flutter Documentation(Flutter documentation | Flutter), Getx Documentation(get | Flutter Package (pub.dev)), Firebase Documentation(Firebase Documentation (google.com)), MVC architecture Documentation(MVC Framework Introduction - GeeksforGeeks).

2. Architectural Goals and Constraints

2.1 Applicable Standards

- The mobile user interface shall be Android 4.4+ (API level 19+) compliant.

2.2 System Requirements

- The system shall connect directly to the Firebase database.
- The system shall communicate with messaging applications, such as Messenger and SMS, to be able to extract payment messages from users via bot to update accounts.

2.3 Performance Requirements

- The system shall support up to 100 simultaneous users against the central database at any given time.
- The system shall have a high run time speed and respond quickly to users' interaction within 2 seconds.

2.4 Environmental Requirements

None.

2.5 Documentation Requirements

This section describes the documentation requirements of the Dugbet Budget Management system.

- **User Manual**

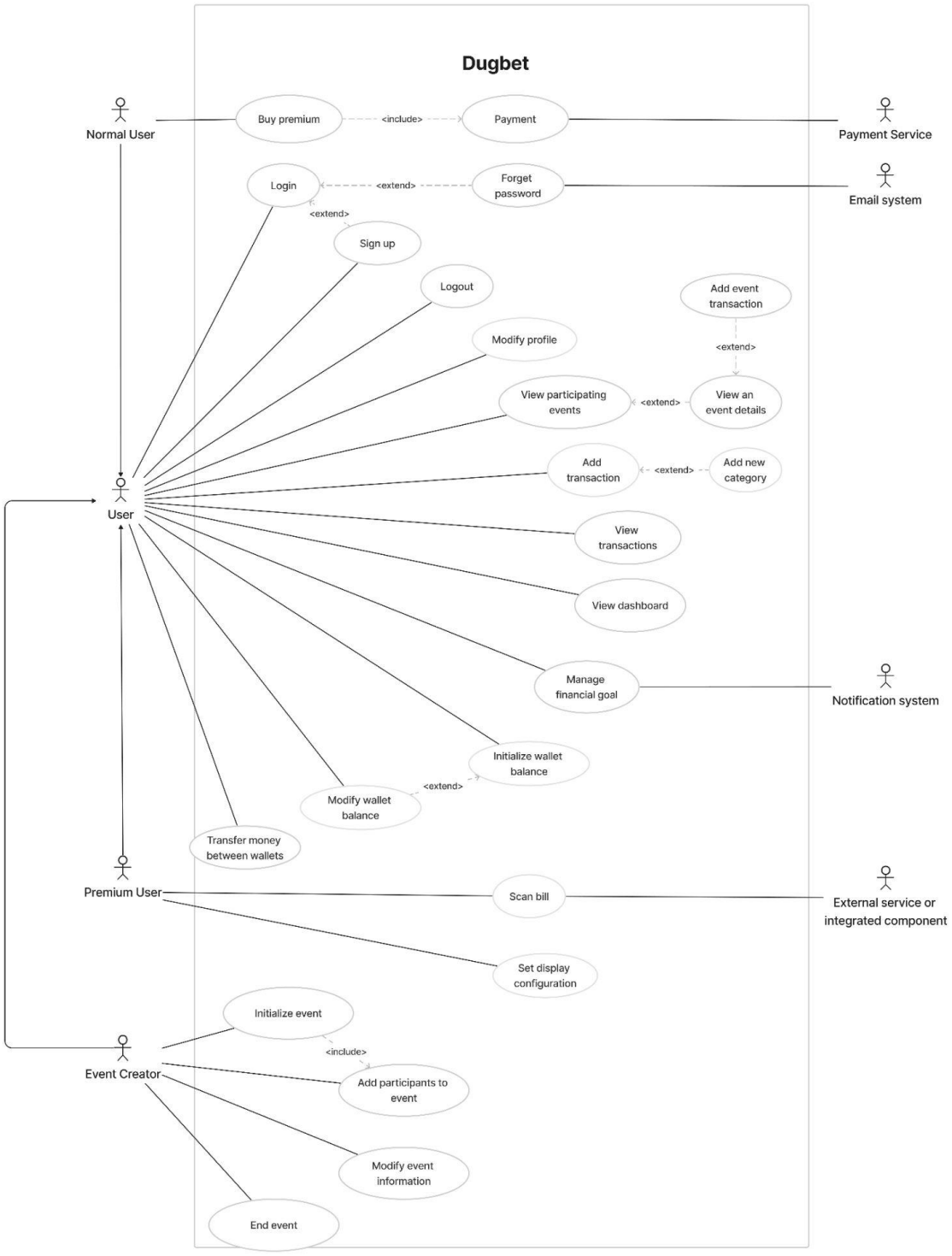
The User Manual shall describe use of the Dugbet system for end users. The User Manual shall include:

- Minimum System Requirements
- Installation of the software
- Logging in/out
- All system features
- Customer support information

- **On-line Help**

- Online Help shall be available to the user for each system function. Each topic covered in the User Manual shall also be available through the online help.

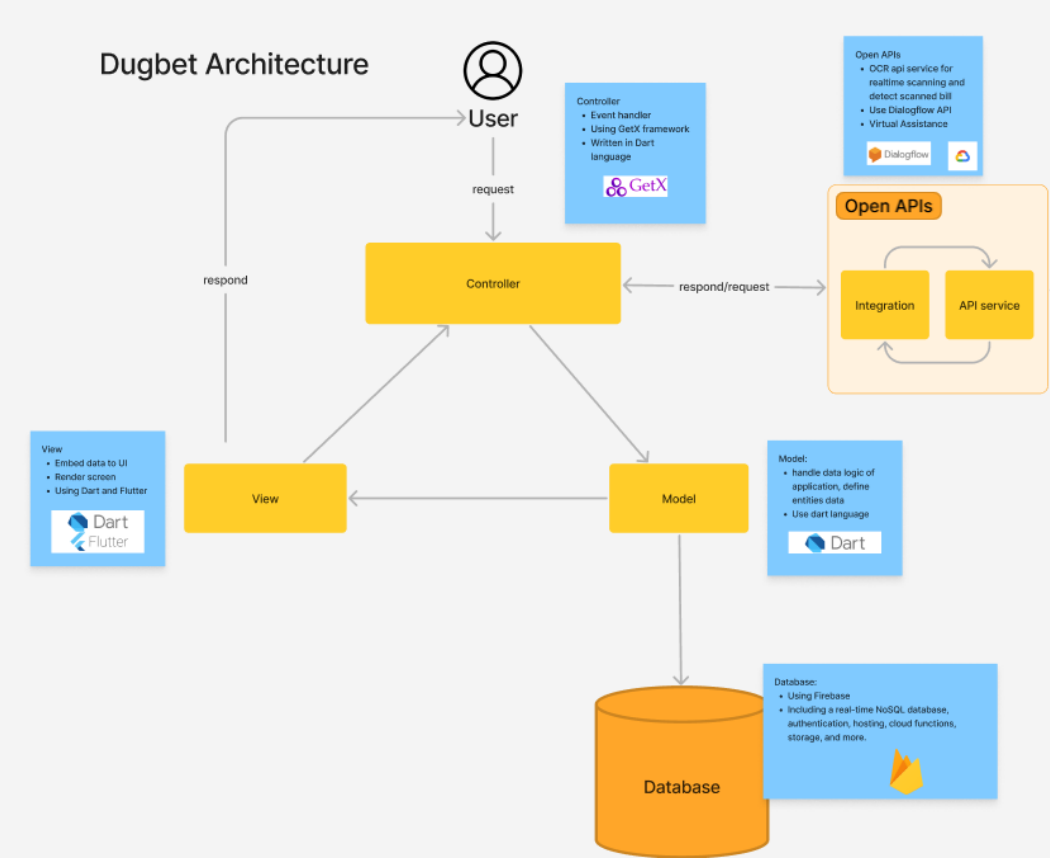
3. Use-Case Model



4. Logical View

4.1 Overview Architecture

The Dugbet application utilizes the Model-View-Controller (MVC) architecture.



4.2 Package Diagram

The package diagram comprises five main packages that form the foundation of the system: **Authentication**, **User**, **Wallet**, **BudgetTarget**, and **Event**.

The **Authentication** component contains controller classes to manage user logging information and boundary classes for the interface with the Firebase system and external mail service system.

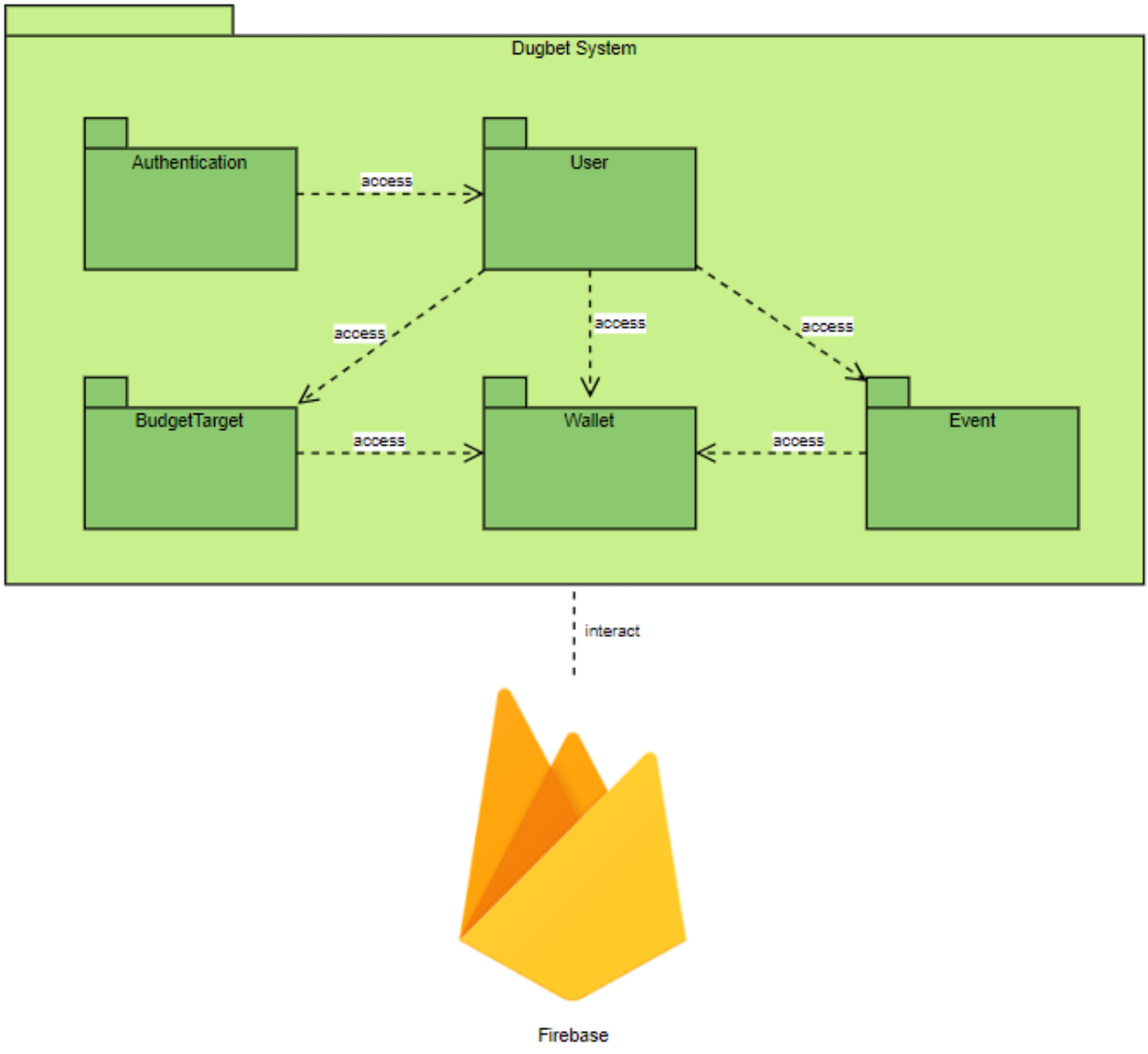
The **User** component consists of entity classes and controller classes for managing users' basic information and their custom settings of the application.

The **Wallet** component comprises entity and controller classes for managing the data of users' wallets.

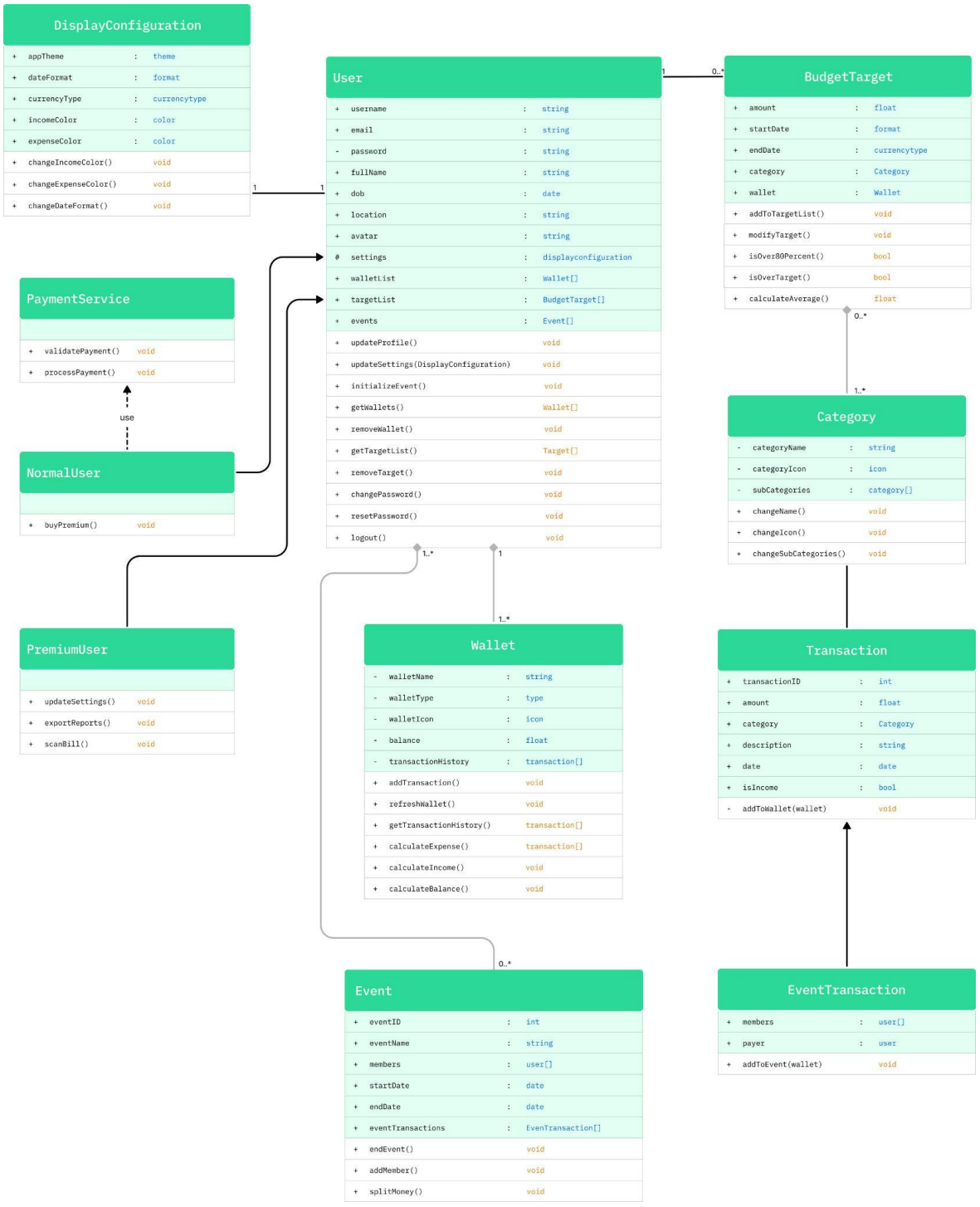
The **BudgetTarget** component consists of classes and controllers relevant to setting personal budget goals.

The **Event** component includes entity and controller classes that represent transactional activities within an event group.

Below is our Dugbet Overview MVC Architecture and Package Diagram.



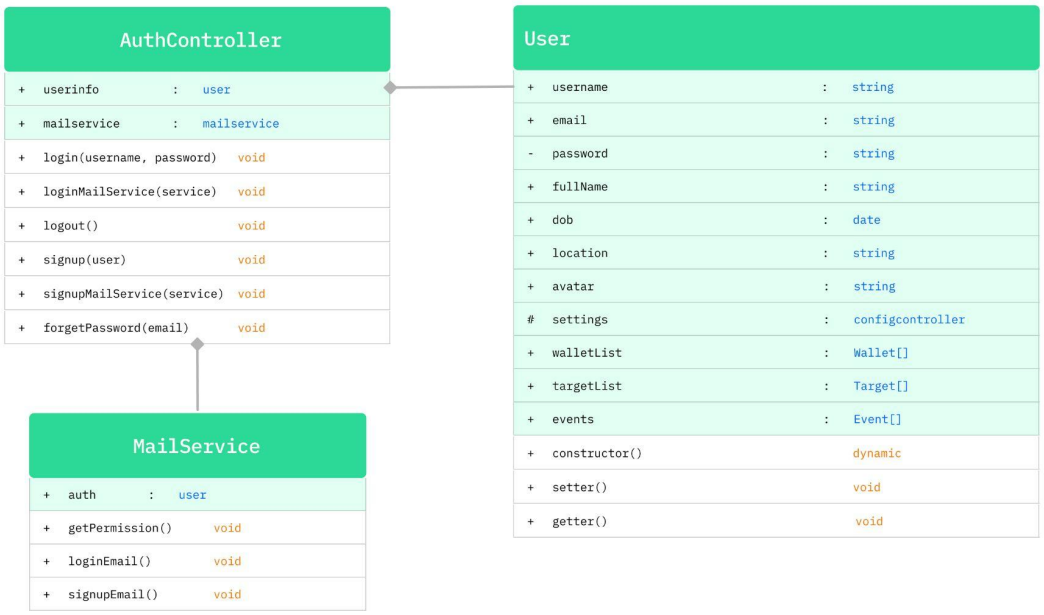
4.3 Class diagram



Description:

- User: Represents all users of the system.
- Wallet: Stores a list of transactions belonging to that wallet.
- Transaction: Represents a financial operation, either expense or income
- EventTransaction: This class is inherited from the Transaction class, representing a transaction specifically within an event
- NormalUser: This class is inherited from the User class, having limited access and functionality
- PremiumUser: This class is inherited from the User class, having full access and functionality.
- Event: Represents a time-bound event, containing several users and their transactions which belong to that event.
- Target: Represents a financial goal with specific timeframe, associated wallet, and description.
- PaymentService: Provides payment methods for normal users to upgrade to premium accounts.
- DisplayConfiguaration: Stores user-defined settings for appearance (color, theme), currency, etc.
- Category: Represents both pre-defined and user-created categories for transactions.

4.4 Component: Authentication

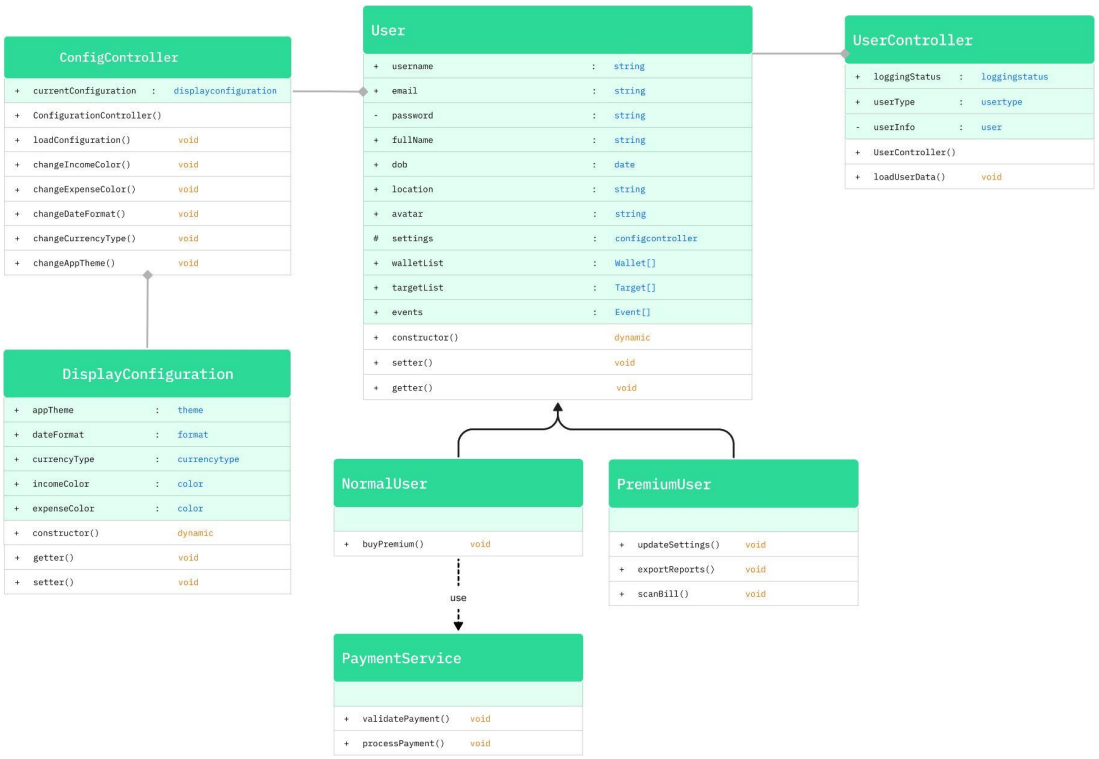


- The diagram above shows the interfaces and the dependencies between the components and classes using arrows relationship.
- The authentication component is responsible for managing user accounts and handling login, registration, and password recovery processes. It interacts with the user database to store and retrieve user information, and it provides methods for users to log in, sign up, and forget their passwords. The authentication component also handles mail services, sending emails to users for registration confirmations and password resets.
- It contains AuthController, MailService and User (used as model) for Authentication Component.
    + The User Model represents a user account in the system. It includes attributes such as username, email, password, full name,

date of birth, location, and avatar.

+ The AuthController provides the login method verifies a user's credentials and grants access to the application from User Model. The signup method creates a new user account and sends a confirmation email. The forgetPassword method initiates a password reset process and sends an email with a reset link.
+ The MailService class handles sending emails related to user authentication, such as registration confirmation emails and password reset emails. It interacts with a mail service provider to send emails.

4.5 Component: User



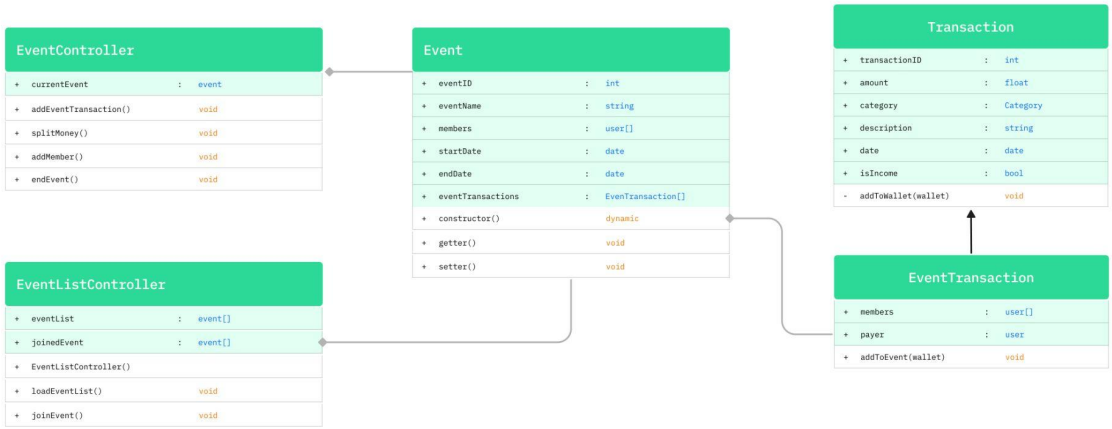
- The diagram above shows the interfaces and the dependencies

between the components and classes using arrows relationship.

- The user component manage user accounts, handling login, registration, password recovery, and user settings. It contains various classes and functionalities for user interaction and data management.
- It contains UserController, User Model (includes two types of user and its payment services), Configuration subcomponents (includes DisplayConfiguration and ConfigController).
    + The Configuration subcomponents use DisplayConfiguration class for managing user settings, such as app theme, date format, currency type,... for the user interface and stores the user's settings in a database storage. ConfigController class manages the display configuration for the user interface. It interacts with the DisplayConfiguration class to load, update, and save the user's display preferences.
    + User Model stores the user's essential information, such as username, email, password, date of birth, location, and avatar. It also has methods for loading user data, updating user settings, and changing the user's display preferences.
    + NormalUser class represents a regular user of the app. It has all the functionalities of the User class.
    + PremiumUser class represents a premium user of the app. It inherits from the NormalUser class and adds additional features, such as the ability to buy premium features, export reports, and use advanced money management tools.
    + UserController class handles user-related tasks, it interacts with the User Model class to retrieve and update user information.
    + PaymentService class handles payment processing for premium features and other in-app purchases. It validates payment information and processes payments through a secure payment

gateway.

4.6 Component: Event



- The diagram above shows the interfaces and the dependencies between the components and classes using arrows relationship.
- The event (group money) component used for collaborative money management for group activities, expenses, and shared finances. It enables users to create, join, and manage group events, track shared expenses, and split payments among participants.
- It contains EventController, EventListController, Event Model, EventTransaction classes.
    + Event Model class represents a group money event. It stores the event ID, event name, members, description, start date, end date, and whether it is an income event.
    + Transaction class represents a transaction related to an event. It stores the transaction ID, amount, category, description, and date.
    + EventController class manages event-related tasks, such as creating, editing, joining, and ending events. It interacts with the Event and Transaction classes to handle event data and

transactions.

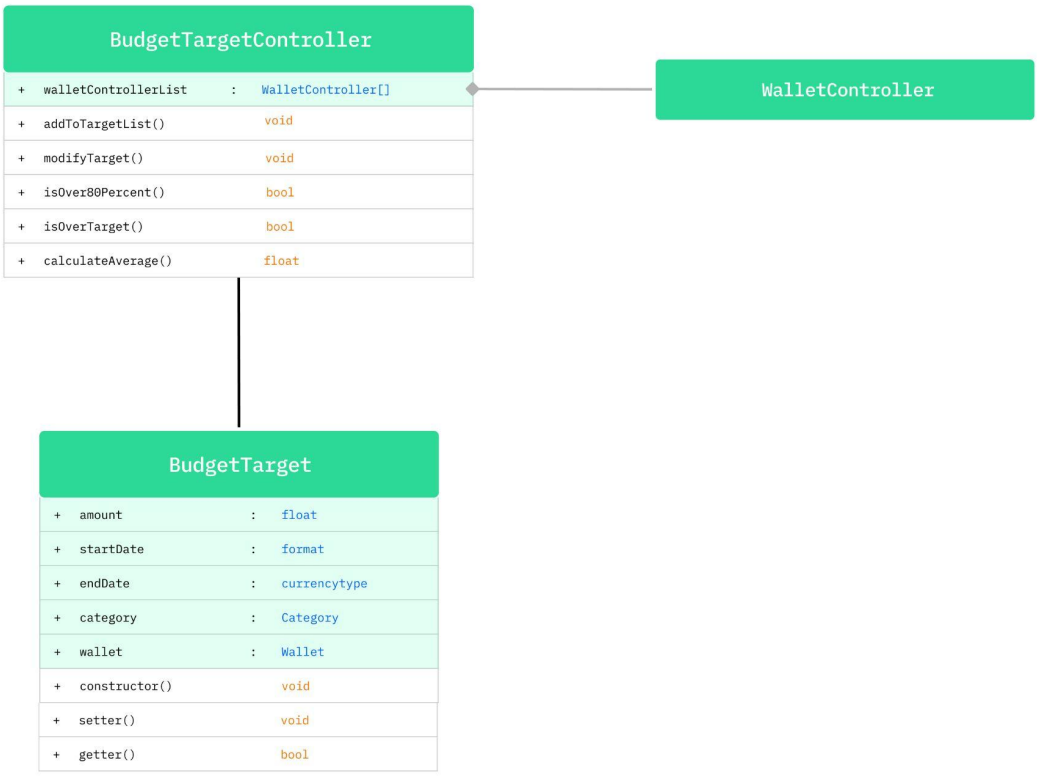
+ EvenTransaction class represents a specific transaction within an event, use for storages and logs the members and payers of event. It also has methods for adding the transaction to the wallet of the corresponding user.
+ EventListController class manages the user's list of events. It loads the list of events, allows users to join events, and provides access to event details.

4.7 Component: Wallet

- The diagram above shows the interfaces and the dependencies between the components and classes using arrows relationship.
- The wallet component enables users to manage their personal finances effectively. It provides a platform for storing, tracking, and analyzing financial data, empowering individuals to make informed decisions about their spending and saving habits.

- It contains Wallet Model, Wallet Controller, WalletListController.
    + Wallet Model class stores the wallet's name, type, icon, balance, transaction history, and associated transactions.
    + The WalletController class manages the user's wallets, including loading wallet data, adding new wallets, and retrieving specific wallets. It interacts with the Wallet class to handle wallet information and transactions.
    + WalletListController class manages the user's list of wallets. It provides methods for loading the wallet list, adding new wallets, and removing existing wallets.

4.8 Component: BudgetTarget



- The diagram above shows the interfaces and the dependencies between the components and classes using arrows relationship.
- The BudgetTarget component enables users to establish personalized spending goals for individual wallets and categories. The component also tracks user spending and provides feedback on whether users are on track to meet their budget targets.
- It contains BudgetTargetController, BudgetTarget Model and WalletController.
    + BudgetTarget relies on the Wallet controller to track and monitor the income and expenses associated with each wallet.

+ The BudgetTarget Model class represents a single budget target. It stores information about the budget target, such as the amount of money that is budgeted, the start and end dates of the budget target, and the category of the budget target.
+ The BudgetTargetController class is used for managing the list of budget targets and handling user interactions related to budget targets. It provides methods for adding new budget targets, modifying existing budget targets, and checking whether a budget target is overspent.

5. Deployment

This work belongs to PA4

6. Implementation View

This work belongs to PA4