**Budget Tracker**

This Flutter code sets up a basic budget tracking app that integrates Firebase services for multi-platform support, including Android, iOS, macOS, Windows, and Web. The application initializes Firebase in the main function and then launches the `MyApp` widget, which serves as the root of the app. Here’s a summary of each part:

**Main Function**

**1. Firebase Initialization:** `Firebase.initializeApp()` initializes Firebase services. This allows the app to use Firebase features like authentication, database, and storage.

**2. App Launch:** Once Firebase is initialized, `runApp(const MyApp())` starts the application with the `MyApp` widget.

**MyApp Widget**

**1. Material App Setup:** The `MyApp` widget is a `StatelessWidget` that uses `MaterialApp` to define the theme and main settings for the app.

**2. Media Query:** Adjusts text scaling to a fixed scale factor (1.0) for consistent text sizes across devices.

**3. Theme Customization:** The theme uses Material 3 design principles and has a color scheme based on a deep blue color (`Colors.blue.shade900`).

**4. Home Screen:** Sets `AuthGate()` as the home screen, which likely manages user authentication.

**Firebase Options Configuration**

The `DefaultFirebaseOptions` class contains Firebase configuration settings, stored separately for each platform (web, Android, iOS, macOS, and Windows). It uses different API keys, App IDs, and other identifiers to handle Firebase services for each platform. The code ensures that only supported platforms (Android, iOS, macOS, Windows, and Web) are configured.

For unsupported platforms like Linux, an error message is thrown, indicating that Linux support has not been configured.

**Summary**

The app is designed for budget tracking and uses Firebase as its backend. It is built to work across multiple platforms by specifying Firebase options for each one, with the app theme, structure, and entry screen set through the `MyApp` widget.

This code defines a few essential widgets for a budget tracking app in Flutter. Each widget has its own purpose, from handling user authentication to displaying category options for tracking expenses. Here’s a breakdown of what each part does:

**AuthGate Widget**

**Purpose:** Manages user authentication and directs users to the appropriate screen based on their login status.

**Functionality:**

- The `AuthGate` widget listens for authentication state changes with `FirebaseAuth.instance.authStateChanges()`.

- If a user is logged in, it directs them to the `Dashboard` screen.

- If no user is logged in, it shows the `LoginView` screen, prompting the user to log in.

**CategoryDropDown Widget**

**Purpose:** Displays a dropdown menu for users to select a spending category (e.g., Food, Rent).

**Parameters:**

**`cattype`:** The currently selected category.

**`onChanged`:** A callback function that runs when the user selects a different category.

**Functionality:**

- The widget fetches categories from `AppIcons.homeExpensesCategories`.

- Each category is represented with an icon and a name.

- When a user selects a category, it triggers the `onChanged` callback, allowing other parts of the app to respond to this selection.

**CategoryList Widget**

**Purpose:** Shows a horizontal list of categories that users can tap to view specific expense categories.

**Parameters:**

**onChanged`:** A callback function that is triggered when a category is selected.

**Functionality:**

CategoryList` maintains a `currentCategory` variable to track which category is selected.

- `initState` sets up the list of categories by pulling data from `AppIcons.homeExpensesCategories` and adding an "All" category at the beginning.

- A `ListView.builder` creates a horizontal scrollable list of categories, each displaying an icon and name.

- When a category is tapped, `currentCategory` updates, triggering `onChanged` and allowing other parts of the app to respond.

**Summary**

This code is for a budget tracker app that includes authentication and category selection features. `AuthGate` handles navigation based on login status, `CategoryDropDown` provides a dropdown for category selection, and `CategoryList` shows a horizontal scrollable list of categories for quick access. These widgets improve user interaction by making it easy to log in, select expense categories, and navigate through the app.

This Flutter code creates a financial summary card called `HeroCard`, which shows a user’s financial details such as their total balance, total credit, and total debit. It uses Firebase Firestore to retrieve the financial data in real-time for a given user. Here’s a breakdown of each component:

**HeroCard Widget**

**Purpose:** Retrieves user-specific financial data from Firebase Firestore and displays it as a financial overview card.

**Functionality:**

- `HeroCard` uses a `StreamBuilder` to listen to real-time updates from Firestore on a specific document within the `users` collection.

- It displays either an error, a loading message, or the user’s financial details based on the connection state and data availability.

- Once data is retrieved, it passes this data to the `Cards` widget for detailed display.

**Cards Widget**

**Purpose:** Displays the user’s financial summary with total balance, credit, and debit amounts.

**Functionality:**

- Shows the **Total Balance** with a large, bold font for emphasis.

- Displays two `CardOne` widgets: one for \*\*Credit\*\* and one for \*\*Debit\*\*.

- Each `CardOne` component is color-coded (green for credit, red for debit) for easy differentiation.

- A bottom section with rounded corners is set to a white background for a clean, separated look from the main blue background.

**CardOne Widget**

**Purpose:** Represents a single data card (either Credit or Debit) that shows the transaction type and amount.

**Parameters:**

- `color`: The color associated with the transaction type (green for Credit, red for Debit).

- `heading`: Either "Credit" or "Debit."

- `amount`: The amount for the transaction type.

**Functionality:**

- Shows the transaction heading and amount with the provided color, making it clear which data is being displayed.

- Includes an icon to show whether it’s a credit (up arrow) or a debit (down arrow) for quick recognition.

- Utilizes `Expanded` so that each card takes up equal space horizontally in the display.

**Summary**

This code creates a real-time financial summary component with a clean, organized layout. `HeroCard` retrieves and listens to updates from Firebase, while `Cards` and `CardOne` visually display financial data with a color-coded, organized structure. This setup helps users quickly see their current financial status, including total balance, credits, and debits, at a glance.

This Flutter code builds the main screens and navigation for a budget-tracking app. It includes user authentication, a dashboard, and options for viewing and adding financial transactions. Here’s a summary of each component:

**1.Dashboard Screen**

**Purpose:** Acts as the main screen, displaying the navigation bar and content areas (Home and Transaction screens).

**Features:**

- The `NavBar` widget allows users to switch between different pages by tapping on icons.

- A `pageViewList` array holds the available screens: `HomeScreen` and `TransactionScreen`.

- The selected screen changes based on the `currentIndex` value, updated when a user selects a different option in the navigation bar.

**2. HomeScreen**

**Purpose:** Displays a summary of user finances and allows users to add new transactions.

**Features:**

**AppBar:** Has a logout button that, when tapped, signs the user out and redirects them to the login screen.

**Floating Action Button (FAB):** This button triggers a form (in a dialog box) where users can add new transactions.

**HeroCard:** Shows an overview of the user's financial status (such as balance and recent transactions).

**TransactionsCard:** Displays a list of recent transactions.

**3. LoginView**

**Purpose:** Allows users to log into their accounts.

**Features:**

**Form:** Users can enter their email and password. The inputs are validated, and any errors are shown.

**Login Button:** Calls the `\_submitForm` method, which sends login credentials to the `AuthService` for validation and authentication.

**Loading Indicaton:** Displays a loading spinner while the login request is processed.

**Create new account" Button:** Redirects to the `SignUpView` screen for users who need to create an account.

**4. SignUpView**

**Purpose:** Allows new users to create an account.

**Features:**

**Form:** Collects information like username, email, phone number, and password. Input validation is handled by the `AppValidator`.

**Create Button:** Calls `\_submitForm`, which sends user details to `AuthService` to create a new account in Firebase.

**Loading Indicator:** Shows a spinner while the account creation request is in progress.

**Login Button:** Redirects to `LoginView` for users who already have an account.

**Input Decorations**

**Purpose:** Standardizes the look of text input fields across the app.

**Features:**

- Each input field has an icon, background color, and text color.

- Border styling makes the input fields visually consistent with the app's color scheme.

**Summary**

This code sets up a user-friendly interface for budget management, allowing users to log in, view their financial data, add transactions, and create new accounts if needed. The app uses Firebase for user authentication, ensuring a secure and responsive experience.

This Flutter code defines a `TransactionScreen` component within a budget tracker app. The `TransactionScreen` provides users with an overview of their transactions, including the ability to filter transactions by month, year, and category. Here’s a breakdown of its main features:

**1. Initialization**

- The screen uses two state variables:

- `category`: Defaults to "All," allowing users to view all transaction categories initially.

- `monthYear`: Displays the current month and year in the format "MMM yyyy" (e.g., "Oct 2024") using the `intl` package for date formatting.

- During initialization (`initState`), the `monthYear` is set to the current date to show transactions for the current month by default.

**2. Screen Layout**

- The `Scaffold` widget creates the basic structure, including an `AppBar` with the title "Expansive."

- The main body layout contains three widgets:

**TimeLineMonth:** A widget that lets users select a month and year to filter transactions. The `onChanged` function updates the `monthYear` state variable when the selected month or year changes.

**CategoryList:** A widget for selecting the transaction category (e.g., "Food," "Shopping"). The `onChanged` function updates the `category` variable when a new category is selected.

**TypeTabBar:** Displays the filtered transactions according to the selected `category` and `monthYear`.

**Summary**

The `TransactionScreen` provides an interactive layout where users can view and filter their transactions based on both category and date. This flexibility allows users to navigate through their transaction history more efficiently. The screen updates in real-time whenever users change the filter criteria.