Introduction

The Price Tracker app is designed to assist users in tracking product prices and receiving notifications when the prices of their desired products match their expected amounts. This documentation provides an in-depth overview of the app's features, functionality, and architecture.

**Key Features:**

* User authentication using Firebase Authentication.
* Product search and selection from various online platforms.
* Input of expected prices for products.
* Notification delivery through Firebase Cloud Messaging (FCM).
* Saved list of tracked items for easy reference.

**Benefits:**

* Empowers users to make informed purchasing decisions.
* Saves time by automatically tracking and notifying users about price drops.
* Offers a user-friendly and intuitive interface for smooth navigation.

Features

**User Authentication**

The app enables users to create accounts and log in securely using their Google accounts. This ensures a personalized experience and allows users to save their tracked items across devices.

**Product Tracking**

Users can search for products across multiple platforms and select items they want to track. The app will monitor the prices of these products and notify users when the prices drop.

**Notification Delivery**

The app sends real-time notifications to users' devices when the prices of tracked products match their expected amounts. This ensures users never miss out on a great deal.

**Saved Lists**

Users can view a list of their saved items and their expected prices. This feature helps users keep track of the products they are interested in.

Frontend Components

**LoginPage**

The LoginPage provides a secure authentication process using Google Sign-In. Users can log in or sign up using their Google accounts.

**ListBuilder**

The ListBuilder component displays search results to users and allows them to select products for tracking.

**PriceInput**

Users can input their expected prices for tracked items and submit them for tracking. The data is stored in Firebase Firestore.

**SavedList**

SavedList displays users' saved items and their expected prices retrieved from Firebase Firestore.

**FinalPage**

The FinalPage provides feedback to users based on tracking results. It displays messages indicating if the price is already lower or if they will be notified later.

Backend Integration

**Firebase Authentication**

Firebase Authentication is used for secure user authentication. Users can log in and register using their Google accounts.

**Firebase Firestore**

Firestore is used to store user data, including tracked items and expected prices.

Data Flow

The Price Tracker app follows a structured data flow process, ensuring smooth interaction between users, the frontend, and backend services. Let's explore the data flow using a user scenario:

**User Scenario:** Tracking Product Prices and Receiving Notifications

1. User logs in using their Google account via the LoginPage component.
2. User searches for products using the ListBuilder component and selects items to track.
3. User inputs their expected prices for the selected items using the PriceInput component.
4. The app stores the user's expected prices in Firebase Firestore.
5. App continuously monitors product prices against the expected prices.
6. When a product's price matches the expected amount, the app sends a notification trigger to Firebase Cloud Messaging (FCM).
7. FCM delivers the notification to the user's device, alerting them about the price drop.

User Flows

**User Login and Registration**

1. User launches the app and is directed to the LoginPage.
2. User taps the Google Sign-In button.
3. App redirects to Google Sign-In page, where the user selects their account.
4. Firebase Authentication validates the user's credentials.
5. If a new user, their account is created. If an existing user, they are logged in.
6. User gains access to the main app interface.

**Product Search and Selection**

1. User enters keywords in the search bar on the ListBuilder page.
2. App queries external platforms and displays search results.
3. User taps on desired products to select them for tracking.
4. Selected products are highlighted and added to the tracking list.

**Price Input and Submission**

1. User navigates to the PriceInput page from the tracking list.
2. User inputs their expected price for a tracked product.
3. User taps the "Submit" button.
4. App stores the expected price in Firestore, associating it with the respective product.

**Saved Items and Notifications**

1. User visits the SavedList page to view their tracked products.
2. App fetches data from Firestore and displays saved products along with expected prices.
3. App continuously monitors the prices of saved products.
4. When a price matches the expected amount, a trigger is sent to FCM.
5. FCM delivers a notification to the user's device, informing them of the price drop.

External Services

The Price Tracker app integrates the following external services:

* **Firebase Authentication:** Provides secure user authentication and registration.
* **Firebase Firestore:** Stores user data, including tracked products and expected prices.
* **Firebase Cloud Messaging (FCM):** Sends real-time notifications to users' devices.

A Flask server is used for sending notifications based on user preferences. The Flask server communicates with FCM to trigger notifications when product prices match expected amounts.

Third-Party Libraries

The Price Tracker app leverages a variety of third-party libraries to enrich its functionality and streamline development. These libraries enhance user experience, facilitate data management, and enable seamless integration with external services. Here are the key third-party libraries used in both Dart and Python codebases:

**Dart (Frontend) Libraries:**

* **Firebase Packages:** The following Firebase packages are integrated into the app to provide essential backend services:
  + **firebase\_auth**: Enables secure user authentication and registration.
  + **cloud\_firestore**: Facilitates real-time data storage, retrieval, and synchronization.
  + **firebase\_messaging**: Allows for the sending and receiving of push notifications.
* **Google Sign-In Package (google\_sign\_in):** This library simplifies the integration of Google Sign-In functionality, enabling users to log in securely using their Google accounts.
* **Provider Package (provider):** The **provider** library is used for efficient state management and dependency injection. It helps manage the app's global state, ensuring smooth communication between components.

**Python (Backend) Libraries:**

* **Flask:** The Flask framework is used to create a lightweight and flexible web server for delivering notifications to users. Flask enables the app to interact with Firebase Cloud Messaging (FCM) for sending push notifications.
* **firebase-admin:** This Python library allows the Flask server to interact with Firebase services. It provides tools for authenticating with Firebase and sending messages through FCM.
* **requests:** The **requests** library is used to make HTTP requests to external services, including FCM. It facilitates communication between the Flask server and Firebase services.

These third-party libraries collectively contribute to the robustness and efficiency of the Price Tracker app, enhancing user engagement and overall functionality. By utilizing established and well-maintained libraries, the development process is streamlined, allowing for a more focused approach to implementing key features.

Conclusion

The Price Tracker app empowers users to track product prices and receive timely notifications about price drops. By seamlessly integrating frontend components with Firebase services and external communication, the app offers an intuitive and efficient experience. With features such as user authentication, product tracking, and real-time notifications, the app simplifies the process of making informed purchasing decisions. As you continue to enhance the app, consider additional features and improvements to further elevate the user experience.

**(\*Detailed documentation of backend is mentioned backend.docx)**