Task: Firebase Admin SDK CRUD Application

Objective:

Develop a backend application using the Firebase Admin SDK to perform Create, Read, Update, and Delete (CRUD) operations on both Firestore and Realtime Database, while also managing user authentication.

Technologies:

- Firebase Admin SDK
- Firebase Authentication
- Cloud Firestore
- Firebase Realtime Database
- Fast API
- Postman/Swagger UI for testing

Requirements:

1. Project Setup:

- Set up a new Firebase project (or use an existing one).
- Generate a Firebase Admin SDK service account key.
- Initialize the Firebase Admin SDK in your chosen backend environment.

2. Authentication:

- Implement functionality to create new users using the Admin SDK (create User).
- Implement functionality to get user details by UID (get User).
- Implement functionality to update user properties (e.g., email, password, display name) (update User).
- Implement functionality to delete users (delete User).
- Implement functionality to verify ID tokens received from a client application (verifyIdToken) to secure API endpoints.

3. Cloud Firestore CRUD:

- Choose a simple data model (e.g., 'items', 'products', 'tasks').
- Implement functionality to create a new document in a specified collection (add or set).
- Implement functionality to read a single document by ID (get).
- Implement functionality to read multiple documents from a collection (e.g., with queries, get).
- Implement functionality to update an existing document (update or set with merge).
- Implement functionality to delete a document (delete).

4. Firebase Realtime Database CRUD:

- Choose a simple data structure.
- Implement functionality to create new data at a specified path (set or push).
- Implement functionality to update data at a specified path (update).
- Implement functionality to delete data at a specified path (remove).
- Implement functionality to read data from a specified path (once).

5. Integration & Security:

- Ensure that all API endpoints requiring user identity are protected using the verified ID token from Firebase Authentication. Only authenticated and authorized users should be able to perform certain operations.

6. Implementation Details:

- Consider error handling for all Firebase operations (e.g., user not found, permission denied).
- Think about data validation before writing to the databases.
- Structure your code logically (e.g., separate modules for Auth, Firestore, RTDB operations).

7. Deliverables:

- Source code of your backend application.
- Instructions on how to set up and run the application.
- (Optional) Documentation explaining the API endpoints and how to use them.
-

This task provides a solid foundation for understanding how to manage users and data securely from a trusted server environment using the Firebase Admin SDK

Firebase Project Setup Guide (Backend Integration)

Follow this step-by-step guide to set up your Firebase project with authentication, Firestore, and Realtime Database, suitable for backend development using Python.

Step 1: Create a Firebase Project

1. Go to Firebase Studio.
2. Click on **"Go to Console"** (top right corner).
3. Click on **"Get started with a Firebase project"**.
4. Enter a name for your project (e.g., `Music App`).
   - A unique project identifier will automatically be generated just below the name field.
5. Click **"Continue"** through the prompts until you reach the **"Create project"** button.
6. Wait a few seconds while Firebase creates your project.
7. Once created, click **"Continue"** to open your Firebase project dashboard.

Step 2: Set Up Build Features

On the left-hand menu, under the **Build** section, we will configure:

Authentication Setup

1. Click on **Authentication**.
2. Click **"Get Started"**.
3. Under the **Sign-in method** tab, select **Email/Password**.
4. Enable **Email/Password**, then click **Save**.

Authentication is now configured.

Firestore Database Setup

1. Select **Firestore Database** from the Build menu.
2. Click **"Create Database"**.
3. Choose a location (default is fine) → click **Next**.

4. Select **Start in test mode** → click **Enable**.

Firestore Database is now configured.

Realtime Database Setup

1. Select **Realtime Database** from the Build menu.
2. Click **"Create Database"**.
3. Choose the default location → click **Next**.
4. Select **Start in test mode** → click **Enable**.

Realtime Database is now configured.

Step 3: Register Your App

1. Click on the **Settings icon** next to **Project Overview**.
2. Go to **Project Settings**.
3. Scroll down to the **"Your apps"** section.
4. Click on the **Code icon (</>)** to register a new Web app.
5. Enter any **App nickname** of your choice.
6. Click **"Register app"**.
7. Skip the Firebase SDK setup for now (as this is a backend task).
8. Click **"Continue to Console"**.

Your Firebase app is now registered.

Step 4: Generate Firebase Admin SDK Credentials

1. In the **Project Settings**, navigate to the **Service Accounts** tab.
2. Choose your backend programming language — **Python** in this case.
3. Click on **"Generate new private key"**.
4. Confirm by clicking **"Generate key"**.
5. A `.json` file will be downloaded — **store this securely** on your local system.
    - **Do not share this file with anyone** — it contains sensitive credentials.

What is Firebase?

Introduction

Firebase is a powerful platform developed by **Google** that offers a suite of **backend services** and **tools** to help developers build and manage web and mobile applications **easily** and **efficiently**.
It provides **Backend-as-a-Service (BaaS)**, eliminating the need for manual server-side programming, maintenance, and scaling.

Core Features of Firebase

- **Real-time Database:** Stores and syncs data across all clients in real-time.
- **Authentication:** Provides easy-to-integrate authentication services (Google, Facebook, Twitter, Email/Password, etc.).
- **Cloud Firestore:** A flexible, scalable NoSQL database to store and sync data for client- and server-side development.
- **Cloud Functions:** Serverless functions that automatically respond to events in Firebase features.
- **Hosting:** Fast, secure, and easy-to-use hosting for web applications.
- **Cloud Messaging (FCM):** Free service for sending push notifications and messages across platforms.
- **Crashlytics:** Real-time crash reporting to track, prioritize, and fix app stability issues.
- **Remote Config:** Dynamically change the behaviour and appearance of your app without releasing a new version.
- **Analytics:** Free and unlimited app usage analytics for informed decision-making.

Why Should You Use Firebase?

Feature	**Benefit**
Backend Management	No need to manage servers manually — Firebase handles it for you.
Real-time Synchronization	Ideal for apps like chats, live dashboards, and multiplayer games.
Simple SDK Integration	Available for Android, iOS, Web — easy to set up and use.

Scalable Infrastructure	Grows automatically with your app's needs, from 10 users to 10 million.
Generous Free Tier	Great for start-ups, hobby projects, and initial MVPs.
Built-in Security	Advanced user authentication and database security rules.

Real-world Example:

Suppose you are building a **chat application**.

Without Firebase:

- You would need to set up your own server, database, security layers, and build complex real-time communication protocols.

With Firebase:

- You can have your real-time chat functionality up and running in just a few hours with a few lines of code, using Firebase's **Realtime Database** and **Authentication** services.

Conclusion:

Firebase is especially suitable for **start-ups**, **solo developers**, and **small teams** who want to quickly launch apps without worrying about backend complexities. It is a **reliable**, **scalable**, and **developer-friendly** platform, making it one of the most popular choices for modern app development.

What is Firebase Admin SDK?

Introduction

The **Firebase Admin SDK** is a set of server-side libraries provided by **Firebase** that allows trusted environments (such as servers, cloud functions, or backend applications) to interact with Firebase services **securely and directly**.

It is mainly used when you need **privileged access** to Firebase services — actions that should **not** be exposed to users' devices (for example: managing users, accessing the database with admin rights, or sending notifications to users).

Core Capabilities of Firebase Admin SDK

- **Authentication Management**
    - Create, update, and delete user accounts.
    - Verify user tokens.
    - Manage user sessions securely from the server.
- **Database Access with Admin Privileges**
    - Read and write data to the Firebase Realtime Database or Cloud Firestore without needing user-based security rules.
- **Cloud Messaging (FCM)**
    - Send customized push notifications to devices directly from the server.
- **Custom Claims and Roles**
    - Assign custom claims (roles) to users for role-based access control.
- **Cloud Storage Management**
    - Upload, download, and manage files stored in Firebase Storage.

Example Use Case

Imagine you are running a large e-commerce app.

Your backend server needs to:

- Create admin users,
- Assign special access rights,
- Push promotional notifications automatically to specific users.

The **Firebase Admin SDK** allows you to perform all these actions **securely from your backend** without exposing sensitive operations to end-users.

Why Use Firebase Admin SDK?

Feature	Benefit
Server-Side Security	Handles sensitive operations securely on the server, not on the client.
Advanced User Management	Create, update, verify users directly through code.
Full Access	Bypass normal user-permission restrictions for administrative operations.
Cross-Platform	Available for Node.js, Java, Python, Go, C#, etc.
Efficient Communication	Easily send mass notifications to users.

Are There Other Firebase Services?

Yes, Firebase offers **many additional services** beyond just the Admin SDK. Here's a quick look:

Other Important Firebase Services

Service Name	Purpose	Quick Description
Firebase Authentication	Identity Management	Easy sign-in and authentication with email/password, phone, and popular providers like Google, Facebook.
Firebase Realtime Database	Live Data Storage	NoSQL database that syncs data across all clients in real-time.
Cloud Firestore	Flexible Data Storage	Modern NoSQL database for scalable app development.
Firebase Hosting	Web Hosting	Fast, secure, production-grade hosting for web apps.

Cloud Functions	Backend Logic	Serverless backend code triggered by Firebase events or HTTPS requests.
Firebase Cloud Messaging (FCM)	Notifications	Send push notifications and messages across Android, iOS, and Web.
Firebase Analytics	App Insights	Collects data on user behaviour for better app optimization.
Firebase Crashlytics	Crash Reporting	Real-time crash tracking, alerting, and analysis.
Firebase Remote Config	Dynamic App Updates	Change app behaviour and appearance without deploying a new version.
Firebase Performance Monitoring	App Performance Tracking	Get insights into app performance, network latency, and stability.

Real-world Example for Other Services

Suppose you build a **food delivery app**:

- Use **Authentication** for secure user signups.
- **Realtime Database** for live order tracking.
- **Cloud Functions** to auto-notify users when orders are prepared.
- **Cloud Messaging** to send promotional push notifications.
- **Crashlytics** to instantly fix app crashes in production.

Conclusion

- The **Firebase Admin SDK** is essential for server-side operations that require **higher-level permissions**.
- Firebase as a whole provides a **complete ecosystem** for app development — from **authentication** and **database management** to **analytics**, **notifications**, and **crash reporting**.
- Whether you're a solo developer or part of a big team, Firebase offers the tools to **build, grow, and scale** apps faster and smarter.

Firebase Services Visual Map

```
Firebase
├── Firebase Admin SDK
│   ├── User Management (Create / Delete / Update Users)
│   ├── Server-side Database Access (Firestore, Realtime DB)
│   ├── Cloud Messaging (Push Notifications)
│   ├── Custom Claims (Roles & Permissions)
│   └── Cloud Storage Management
│
├── Core Services
│   ├── Authentication
│   │   ├── Email/Password Login
│   │   ├── Google, Facebook, Twitter Login
│   │   └── Phone Authentication
│   │
│   ├── Realtime Database
│   │   ├── Live Data Sync
│   │   └── Offline Support
│   │
│   ├── Cloud Firestore
│   │   ├── Scalable NoSQL Database
│   │   └── Offline Persistence
│   │
│   ├── Cloud Functions
│   │   ├── Event-driven Backend
│   │   └── HTTPS Callable APIs
│   │
│   ├── Hosting
│   │   ├── Static Website Hosting
│   │   └── Free SSL and Global CDN
│   │
│   ├── Cloud Messaging (FCM)
│   │   ├── Push Notifications
│   │   └── Targeted Messaging
│   │
│   ├── Analytics
│    │    ├── User Behaviour Tracking
│    │    └── Audience Segmentation
│    │
│    ├── Crashlytics
│    │    ├── Crash Reporting
│    │    └── Real-time Alerts
│    │
│    ├── Remote Config
│    │    ├── Dynamic App Behaviour
│    │    └── A/B Testing
│    │
│    └── Performance Monitoring
│         ├── App Speed Insights
│         └── Network Latency Reports
```

What is Firestore Database and Realtime Database?

Introduction

Firebase offers two powerful cloud-hosted database solutions for developers:

- **Cloud Firestore**
- **Firebase Realtime Database**

Both are used to store and sync data for web and mobile apps, but they are designed for slightly different use-cases and have different architectures.

What is a Firestore Database?

Cloud Firestore is a **NoSQL** document database built for **automatic scaling**, **high performance**, and **ease of application development**.

It stores data in **documents** organized into **collections** — making it very flexible and ideal for modern app development.

Core Features of Firestore:

- **Structured Data:** Data is stored in documents (like JSON objects) and organized in collections.
- **Strong Query Capabilities:** You can perform complex queries, compound queries, and ordering/filtering easily.
- **Real-time Synchronization:** Supports real-time data updates across clients.
- **Offline Support:** Automatically handles offline data persistence for mobile and web apps.
- **Scalability:** Designed to scale automatically from small apps to global enterprises.
- **Multi-Region Support:** Replicates data across multiple regions for high availability and durability.

What is a Realtime Database?

The **Firebase Realtime Database** is a **legacy NoSQL database** that stores data as one big **JSON tree**. It is optimized for **real-time syncing** of data between users and devices.

Core Features of Realtime Database:

- **Real-time Updates:** Data changes are pushed instantly to connected clients.
- **Low Latency:** Super-fast — ideal for chat apps, gaming, and live feeds.
- **Offline Support:** Local caching on devices when offline; syncs changes when reconnected.
- **Simple Data Structure:** Stores all data as a large JSON tree.
- **Cost-Effective:** Great for simple apps with real-time requirements and a lower budget.

Firestore Database vs Realtime Database

Here's a side-by-side comparison:

Feature	**Firestore Database**	**Realtime Database**

Data Model	Documents & Collections	JSON Tree
Scalability	Horizontally scalable, handles very large apps	Limited scalability for complex apps
Querying	Powerful, complex queries supported	Basic querying (filters by one child at a time)
Offline Support	Yes, for both web and mobile	Yes, mainly for mobile
Real-time Updates	Yes	Yes (primary feature)
Multi-Region Support	Yes, automatic replication	Single-region (can configure manually)
Ideal Use Case	Large apps with complex data	Small apps needing instant updates
Pricing Model	Based on document reads/writes	Based on bandwidth and data transferred

Real-world Example:

Suppose you are building a **social media app**:

- Use **Firestore Database** to manage user profiles, posts, comments, and likes — because of its structured data and complex querying capabilities.

- If you are building a **simple live chat app** or a **real-time multiplayer game**, **Realtime Database** would be a faster and cheaper solution.

Conclusion

- **Firestore Database** is best for **scalable**, **structured**, and **complex apps** where advanced querying and large-scale support are needed.

- **Realtime Database** is best for **simple**, **speed-focused**, and **cost-sensitive applications** needing real-time data sync.

Both databases are powerful **choices based on the app’s complexity, scaling needs, and your real-time update requirements.**