

# React-Firebase Project-By Rohit Sir

## ▼ 1. React+Vite Project Setup

### ▼ Step-1: **Install Node.js**

1. Download and install Node.js from [Node.js official website](#).  
Ensure you install the **LTS (Long-Term Support)** version.
2. After installation, verify Node.js and npm versions:



```
node -v  
npm -v
```

### ▼ Step-2: **Create a React Project with Vite**

#### Option 1: Interactive Process

1. Open your terminal and run:



```
npm create vite@latest
```

2. Follow the prompts:
  - **Project Name:** Enter the name of your project.
  - **Framework:** Select **React**.
  - **Variant:** Select **JavaScript**.

#### Option 2: Single Command

1. To skip the interactive prompts and directly create a project with React + JavaScript template:



```
npm create vite@latest project_name -- --template react
```

---

### ▼ Step-3: **Navigate to Your Project Directory**

- Change into your project directory:



```
cd project_folder_name
```

---

### ▼ Step-4: **Install Dependencies**

- Install the required dependencies by running:



```
npm install
```

---

### ▼ Step-5: **Start the Development Server**

- Run the development server:



```
npm run dev
```

---

### ▼ Step-6: **Open Your Application in the Browser**

- After running the above command, Vite will display a local development URL, typically something like:



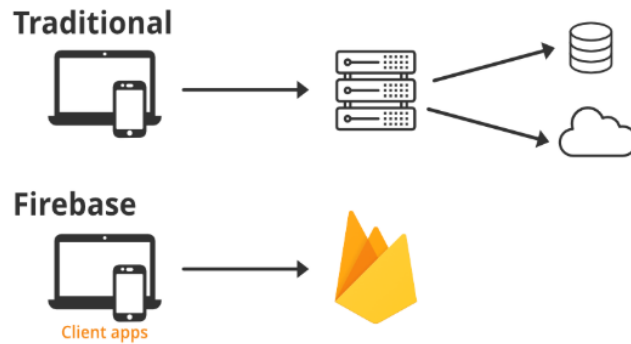
<http://localhost:5173/>

- Copy and open the link in your browser to see your React application running.
- That's it! Your React + Vite environment is now set up and ready for development.

## ▼ 2. What is Firebase?

### ▼ 2.1 **Firebase - Introduction**

- **Firebase** is a platform that helps developers build and run apps for mobile and web.
- It offers backend services like databases, authentication, and hosting, as well as tools for real-time data synchronization, analytics, and app quality.
- Firebase is a product of **Google** which helps developers to build, manage, and grow their apps easily.
- It helps developers to build their apps faster and in a more secure way.
- That means you did not to worry about server side logics or programming (backend infrastructure) , with the help of firebase it makes easy to build apps efficiently.
- Firebase is a **Backend-as-a-Service (BaaS)** platform, which means it provides hosted backend services.
- It provides services to android, ios, web, and unity. It provides cloud storage. It uses NoSQL for the database for the storage of data.



## ▼ 2.2 History of Firebase

- Firebase initially was an **online chat service provider** to various websites through API and ran with the name **Engolve**.
- It became popular because developers found it useful not just for chat, but for sharing real-time data, like the state of a game, across users.
- As a result, the creators of Engolve, **James Tamplin and Andrew Lee**, decided to separate the chat system from the underlying architecture that enabled real-time data exchange.
- They improved and expanded this architecture, which eventually became the Firebase platform we know today.
- This transformation happened in **2012**, and Firebase grew to provide a variety of tools for building apps, not just for real-time data but also for many other features like authentication, storage, and analytics.
- In the year of **2014** **Google** acquired the firebase under them.

## ▼ 2.3 Features or Services of Firebase

- Mainly there are 4 categories in which firebase provides its services.
  1. **Authentication**
  2. **Cloud Firestore**
  3. **Storage**
  4. **Hosting**

## 1. Authentication

- Provides a secure and easy way to authenticate users.
- It helps you manage user logins, supports email/password, phone number, and third-party providers like google, facebook, github, etc.
- Firebase Authentication service provides easy to use UI libraries and SDKs to authenticate users to your app.
- Offers integration with custom authentication systems.

## 2. Cloud Firestore

- A database for storing and syncing your app's data.
- It works in real-time and updates data instantly across users.
- Perfect for things like chat apps, or keeping user data in sync. (**real time data provider**)
- It stores data in the form of objects also known as Documents.

## 3. Storage

- **Firebase Storage** is a **cloud-based storage** solution.
- It is designed to store and serve large files like images, videos, audio, and documents.
- Works well with firebase authentication to ensure only authorized users can access files.

## 4. Hosting

- Provides fast and secure hosting for websites and webpages.
- Ideal for static files like HTML, CSS, and JAVASCRIPT, or single-page applications.
- It supports custom domains.

## Firebase Cloud Messaging(FCM):

- FCM is a service that lets you send messages or notifications from your server to user's devices.
- It helps you send updates like alert, promotions, SMS OTP verification, email verification, etc.

---

## ▼ 2.4 Pros and Cons of Using Firebase

### 1. Pros (Advantages):

- **Free for Beginners** → Firebase offers a free plan, which is great for small projects or trying it out. (approx. 1GB of storage)
- **Real-time Database** → Data updates instantly for all users, making it perfect for apps like chat or live updates.
- **Growing Community** → Many developers use firebase, so its easy to find help of solutions online.
- **Lots of Services** → Firebase provides a wide range of tools, like hosting, storage, authentication, and analytics, all in one place.

### 2. Cons (Disadvantages):

- **NoSQL Database** → Firebase uses on NoSQL, which is different from traditional SQL databases. If you're used to SQL, it might take time to learn and adjust.
- **Still Growing** → Firebase is relatively new and hasn't been tested as much as older more established platforms. Some advanced use cases might face challenges.

---

## ▼ 2.5 Companies using Firebase

Below are some reputable organizations that rely on a firebase backend for its functioning:

- The New York Times
- Alibaba.com
- Gameloft
- Duolingo

- Trivago
  - Venmo
  - Lyft
- 

## ▼ 2.6 **Firestore Pricing**

Firestore has **2 pricing plans**:

### 1. **Spark Plan (Free)**:

- Perfect for beginners and small projects.
- It covers most basic features for free.

### 2. **Blaze Plan (Pay-as-you-go)**:

- You pay based on how much you use Firestore services.
- Ideal for larger apps or when your user base grows.

For most new developers, the **Spark Plan** is enough while learning or starting out. You can upgrade to the **Blaze Plan** when your app scales.

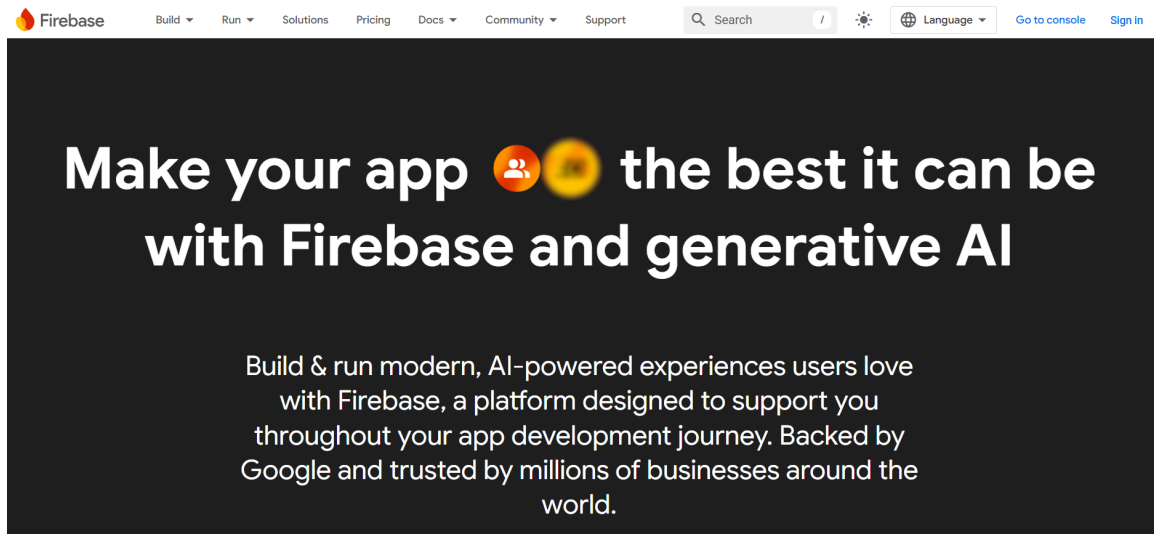
---

## ▼ 3. **Firestore Integration with React.js**

Step-by-Step Guide to Integrating Firestore with React JS:

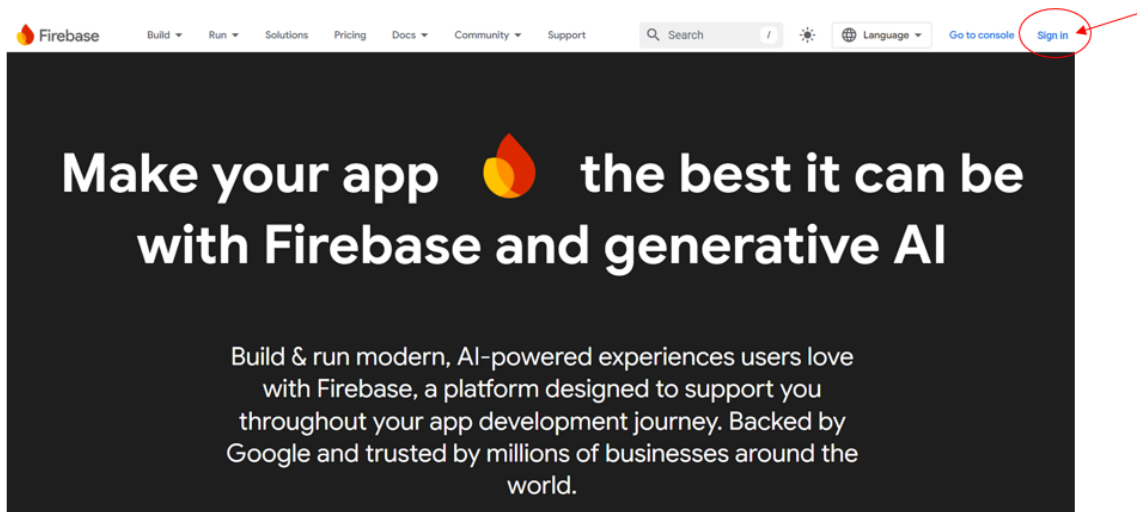
### ▼ **Step-1:** **Go to Firestore Website**

- Open your browser and visit [firebase.google.com](https://firebase.google.com).



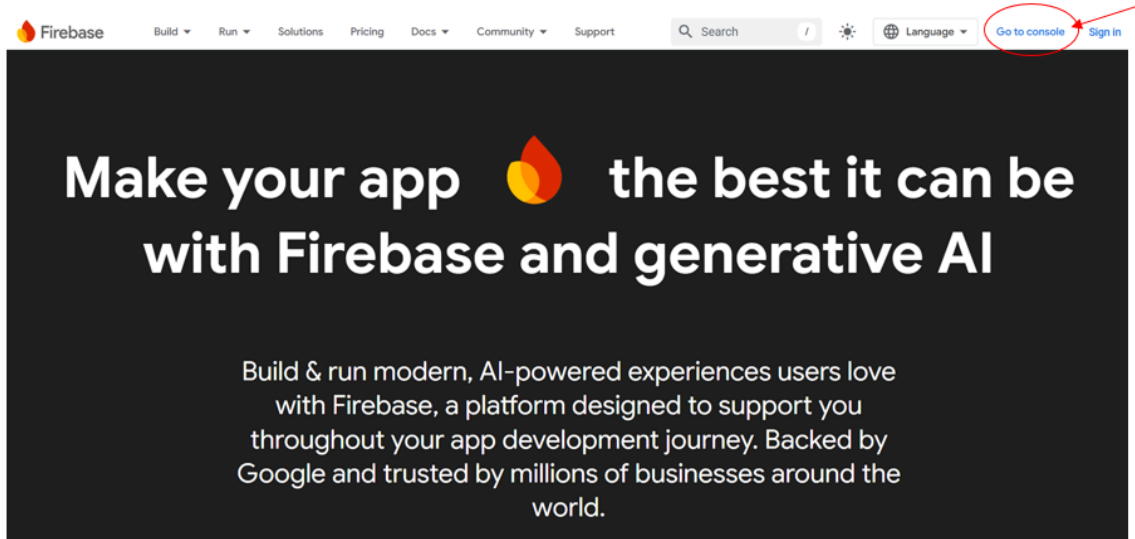
## ▼ Step-2: Login to Firebase

- If you're not logged in, it will prompt you to sign in with your **Google account**. Log in with your Google email.



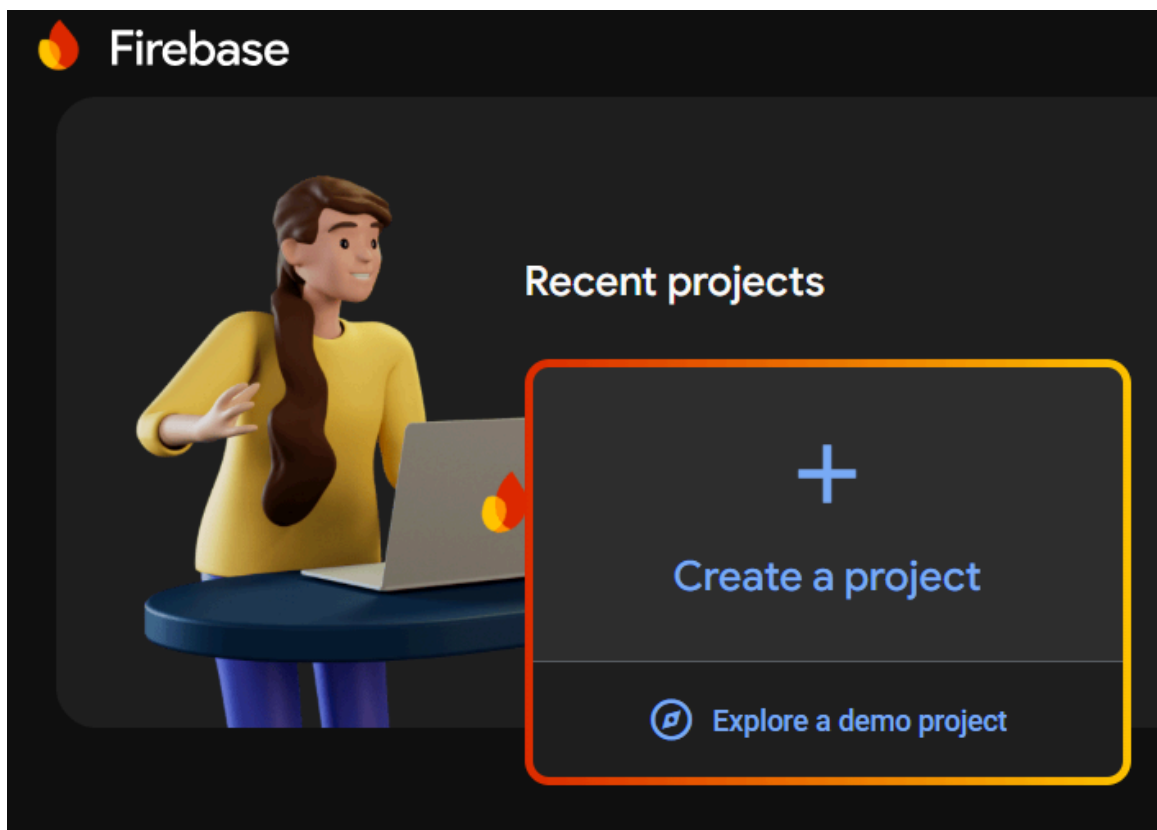
- Click on the **Go to Console** button located in the top-right corner of the page.





### ▼ Step-3: Create a New Project

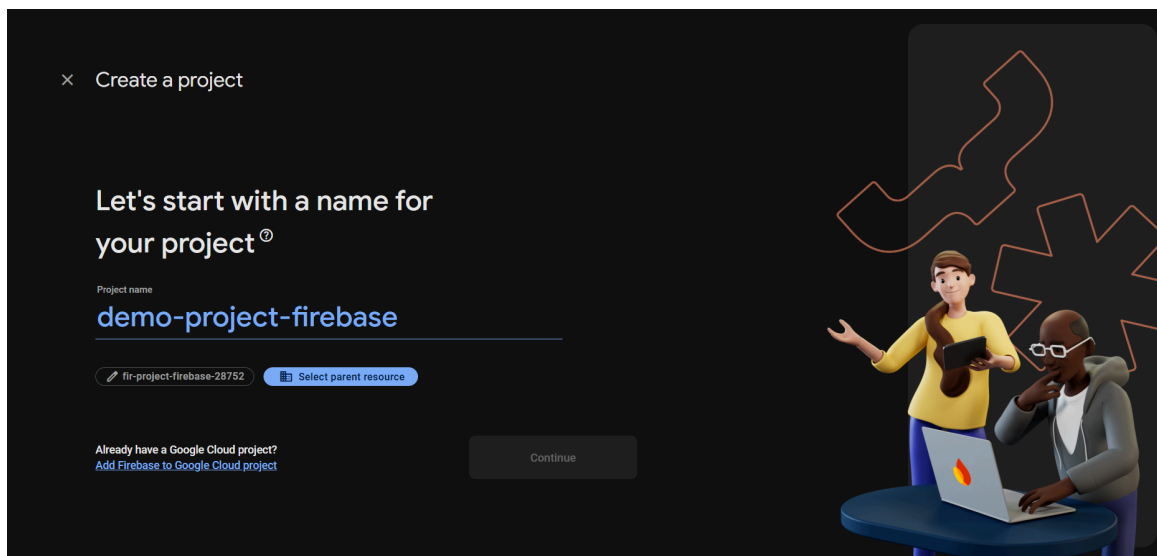
- Once logged in, you'll be taken to the Firebase Console.
- Click on **Add Project** or **Create a New Project**.



## ▼ Step-4: Set Up Your Firebase Project

You'll be asked to fill out some details for your project:

- **Project Name:** Choose a unique name for your Firebase project.
- **Project ID:** This will be auto-generated, but you can change it if needed.
- **Region Settings:** Choose the location of your Firebase project (leave the default if unsure).
- Accept the terms and proceed.





## ▼ Step-5: Enable Google Analytics (Optional)

- You'll be asked whether you want to enable Google Analytics for your project. You can choose to skip this or enable it.
- If you enable it, you'll need to set up your Analytics settings.
- But if you want analytics then only enable but if you don't want analytics just disable the option.

## ▼ Step-6: Firebase SDK Setup

- Once your project is created, you'll be directed to the Firebase project dashboard.

- **Add Firebase to your web app:** Find and click on the **Web**  icon (  ) to set up Firebase for your React project.
- This will show you a Firebase config snippet that contains SDK (Software Development Kit) details (API key, Auth domain, etc.).
- **Copy the Firebase SDK:** Copy this configuration and paste it into a text editor (like Notepad) for later use.
- It will look like this, but don't copy paste from others it is unique for each and every user project structure.

```
import { initializeApp } from "firebase/app";
// Your web app's Firebase configuration
const firebaseConfig = {
  apiKey: "AlzaSyAcl9flxv_H7t_tor3Nj7FtJgSBRiayyl8",
  authDomain: "beat-life-music.firebaseio.com",
  projectId: "beat-life-music",
  storageBucket: "beat-life-music.firebaseio.com",
  messagingSenderId: "388850963619",
  appId: "1:388850963619:web:fd4f99b03200d6beaa1b16"
};

// Initialize Firebase
const firebaseApp = initializeApp(firebaseConfig);
```

### ▼ Step-7: **Install Firebase SDK in Your React App**

- Open your React project in your code editor (VS Code or similar).
- Install Firebase using npm:



npm install firebase

### ▼ Step-8: **Initialize Firebase in Your React App**

- In your React project, create a new folder ( **Backend** ) inside that create a new file (e.g., **firebase-config.js** ) and paste the Firebase configuration you copied earlier.
- Import Firebase and initialize it:
- Paste your firebase-configuration code not others:

```
import { initializeApp } from "firebase/app";
// Your web app's Firebase configuration
const firebaseConfig = {
  apiKey: "AlzaSyAcl9flxv_H7t_tor3Nj7FtJgSBRiayyl8",
  authDomain: "beat-life-music.firebaseio.com",
  projectId: "beat-life-music",
  storageBucket: "beat-life-music.firebaseio.com",
  messagingSenderId: "388850963619",
  appId: "1:388850963619:web:fd4f99b03200d6beaa1b16"
};

// Initialize Firebase
const firebaseApp = initializeApp(firebaseConfig);
```

## ▼ Step-9: **Use Firebase Services in Your App**

### ▼ 9.1 **Importing Firebase Functions**

The code imports necessary functions from Firebase SDK (Software Development Kit):

- **initializeApp** : Initializes Firebase in your app.
- **getAuth** : Allows you to use Firebase's authentication services (login, register, etc.).
- **getFirestore** : Allows you to use Firestore, which is Firebase's cloud database for storing and retrieving data.
- **getStorage** : Lets you use Firebase's cloud storage to upload and manage files like images, videos, etc.

## ▼ 9.2 **Firebase Configuration**

- The `firebaseConfig` object contains your app's Firebase settings like the API key, project ID, etc. These are specific to your Firebase project. It's how Firebase knows which project to connect to.

### 1. **apiKey:**

- The `apiKey` is a unique identifier for your Firebase project. It allows Firebase to authenticate your app and connect it to your specific Firebase project.
- This key is required by Firebase services to ensure that the requests you make (such as reading data from the database or uploading files) come from your authorized app.

### 2. **authDomain:**

- This is the domain used for Firebase Authentication to manage sign-ins and logins. It is the URL where Firebase Authentication handles authentication requests.
- **Example:** `beat-life-music.firebaseio.com` means Firebase Authentication for your project is set up to work with this domain.

### 3. **projectId:**

- The `projectId` is a unique identifier for your Firebase project. It helps Firebase know which project the app belongs to, as you can have multiple Firebase projects (for different apps, environments, etc.).
- `beat-life-music` is the ID of your project.

### 4. **storageBucket:**

- This is the URL for Firebase Storage, where you can store files (like images, videos, etc.). It's tied to your Firebase project.
- **Example:** `beat-life-music.firebaseio.com` is the bucket (or container) where your app's files are stored.

### 5. **messagingSenderId:**

- This is a unique identifier used for Firebase Cloud Messaging (FCM). It's used to send push notifications to your app users.
- If your app uses push notifications (e.g., to notify users of new messages, updates, etc.), this ID is needed to send notifications to the right app.

#### 6. **appId:**

- The `appId` is a unique identifier for the app instance. It helps Firebase distinguish between different app instances and ensures your app is properly connected to Firebase services.
- **Example:** `1:388850963619:web:fd4f99b03200d6beaa1b16` uniquely identifies the app within your Firebase project.

### Summary:

These values together help Firebase services like Authentication, Database, Storage, and Messaging identify and securely interact with your app. They are part of the configuration that ensures Firebase knows which project and services to use when your app makes requests.

## ▼ 9.3 Initialize Firebase

- The `initializeApp(firebaseConfig)` function initializes Firebase with the settings provided in `firebaseConfig`.
- This is necessary to connect your app to Firebase services.

## ▼ 9.4 Exporting Firebase Services

The code exports the initialized Firebase services for later use:

- `_AUTH`: Firebase's authentication service for handling user authentication.
- `_DB`: Firestore, used for interacting with the database.
- `_STORAGE`: Firebase Storage, used for file uploads.

These exports allow you to use the Firebase services throughout your app.

### ▼ Step-10: **Test Your Setup**

- Run your React app ( `npm run dev` ).
- Check your Firebase Console to verify that the integration works as expected.

## ▼ 4. Authentication Methods

- When you integrate Firebase with React.js, Firebase provides a seamless way to add backend functionality to your React application without needing to set up your own server. Here's an explanation of the Firebase methods:

### ▼ 1. **createUserWithEmailAndPassword(auth, email, password)**

- This method creates a new user account with the specified email and password. If successful, the method will return a `UserCredential` object.
- **Syntax** →



```
createUserWithEmailAndPassword(auth, email, password);
```

- `auth` : The Firebase Authentication instance (usually created by `getAuth()` ).
- `email` : The user's email address.
- `password` : The user's password.

- **Returns:**

**UserCredential:** This object contains the user's information after successful account creation. It includes:

- `user` : The newly created `User` object containing the user's profile information.

- `additionalUserInfo` : Provides extra details about the user (such as whether it's their first login).
- `credential` : The authentication credential, which is usually `null` for email/password-based auth.

## ▼ 2. `signInWithEmailAndPassword(auth, email, password)`

- This method signs the user in with their email and password. If the credentials are correct, it returns a `UserCredential` object, which contains the authenticated user's information.
- **Syntax** →



```
signInWithEmailAndPassword(auth, email, password);
```

- `auth` : The Firebase Authentication instance.
- `email` : The user's email address.
- `password` : The user's password.
- **Returns:**
  - **UserCredential**: This object contains the following:
    - `user` : The authenticated `User` object, which contains information like the user's ID ( `uid` ), email, and more.
    - `credential` : The authentication credential used for the sign-in.
    - `additionalUserInfo` : Contains information about the authentication provider (e.g., whether the account is new or not).

## ▼ 3. `sendEmailVerification(user)`

- This method sends a verification email to the user's email address. It's commonly used after creating a new account to confirm the user's email.
- **Syntax** →





`sendEmailVerification(user);`

- `user` : The `User` object (from the `UserCredential` returned by `createUserWithEmailAndPassword ()` or `signInWithEmailAndPassword ()` ).
- **Returns: Promise**
  - **Promise:** This method returns a promise that resolves when the email has been successfully sent. There is no specific value returned, but you can use `.then()` or `async/await` to handle the result.

#### ▼ 4. `onAuthStateChanged(auth, callback(user))`

- This method listens for changes to user's authentication state.
- It trigger's whenever a user signs-in, signs-out, or when the current user's session is restored after a age reload.
- **Syntax →**



`onAuthStateChanged(auth, callback(user));`

- `auth` : It is a authentication instance provided by the firebase `getAuth()` .
- `callback(user)` : It is a currently signed in user object.
- **Return Value:**
  - The `onAuthStateChanged()` method in Firebase Authentication returns an **unsubscribe function**.
  - This function can be called to stop the authentication state listener when it is no longer needed.
  - The return value is a function (commonly referred to as the "unsubscribe" function).
- It is used to track user authentication status in real-time and update UI accordingly.

- The authentication listener is set up when the component is mounted and cleaned up when the component is unmounted.

## ▼ 5. `updateProfile(user, {profile})`

- This method updates the profile information (eg. displayName, photoURL) of the currently signed-in user.
- **Syntax** →

 `updateProfile(user, {profile});`

- `user`: The currently logged-in user object.
- `profile`: A object containing profile fields to update → displayName, photoURL.
- **Return Value:**
  - It will return a promise. So we have to use `try()` and `catch()` block to handle the promise or `async/await` for cleaner syntax.
  - If promise resolves → update is successful.
  - If promise rejects → Error → `token-expired`, `user-account disabled`.

## ▼ 6. `signOut(auth)`

- This method logs out the currently signed-in user.
- **Syntax** →

 `signOut(auth);`

- `auth`: It is a authentication instance provided by the firebase `getAuth()`.
- **Return Value:**

- It will also return promise for that we have to use `try()` and `catch()` block to handle the promise or `async/await` for cleaner syntax.
- If promise resolves → sign-out process is completed.
- If promise rejects → There is an error → common error → `network-error`.

---

## ▼ 7. `sendPasswordResetEmail(auth, email)`

- This method is used to send an email to user containing a link to reset their password.
- It's a part of firebase authentication and is useful for enabling password recovery in your application.
- **Syntax →**



`sendPasswordResetEmail(auth, email, [actionCodeSettings])`

- `auth` : The firebase authentication instance ( `getAuth()` )
- `email` : The user's email address.
- `actionCodeSettings` : Optional → It will define the configuration for object for how the reset link behave.
- **Return Value:** A `Promise<void>`
  - Resolves → When the password reset email is sent successfully.
  - Rejects → with an error if there's an issue. (e.g. `invalid-email` , `user-not-found` )

---

## ▼ 8. `updatePassword(user, newPassword)`

- This method updates the current user's password.
- It is used for authenticated users who want change their password.
- **Syntax →**



`updatePassword(user, newPassword)`

- `user` : The currently signed-in user. (`auth.currentUser`)
  - `newPasword` : The new password to be set for the user.
  - **Return Value:** A `Promise<void>`
    - Resolves → when the password is updated successfully.
    - Rejects → with an error if there's an issue. (e.g. `requires-recent-login` )
- 

## ▼ 9. `deleteUser(user)`

- This method deletes the currently signed-in user from the firebase authentication system.
- After deletion, the user is signed out automatically.
- **Syntax** →



`deleteUser(user)`

- **Return Value:** A `Promise<void>`
    - Resolves → when the user is deleted successfully.
    - Rejects → with an error if there's an issue. (e.g. `user-isn't-authenticated` )
- 

## ▼ 10. `onSnapshot(doc())`

- **onSnapshot()** is a firestore method is used to listen real-time updates for a document or a collection in firestore.
- Whenever the data changes in firestore (e.g. document is updated, deleted, or updated), the listener is triggered and you get the updated data.
- **Syntax** →

- For **document**:

💡 `onSnapshot(doc(dbInstance, "collection_name", documentID),  
(docSnapshot) => {  
 console.log(docSnapshot.data());  
})`

- `docSnapshot` : It contains the updated document data.
- For **Collection**:

💡 `onSnapshot(collection(dbInstance, "collection_name"),  
(querySnapshot) => {  
 console.log(doc.id, "=>", doc.data());  
})`

- `querySnapshot` : It contains all the documents in the collection.
- **Return Value:**
  - The `onSnapshot()` function returns an "`unsubscribe function`" that you can call inside `useEffect()`, to detach the listener when it's no longer needed. (e.g. when the component unmounts)

---

## ▼ 5. Public and Protected Routes

### ▼ 1. `Public Routes`

- The public routes component is used to restrict access to pages that are **only accessible by un-authenticated users**, such as login and registration pages.
- The `authUser` object holds the information about the currently authenticated user.

- It checks → Two Conditions.
    1. Condition-1: Is `authUser` exists and does not have `access token` (✗).
    2. Condition-2: Access Token stored in local storage.
  - Either condition is `true` → already authenticated → Navigate to Home Page. ("`/`")
  - If condition `not true` → It will renders the child component.
- 

## ▼ 2. Protected Routes (Private Routes)

- The protected routes component is used to restrict access to pages that are **only accessible by authenticated users**, such as dashboard or user profile.
  - It retrieves the `authUser` object from `AuthContext`.
  - It checks → Two Conditions.
    1. Condition-1: Is `authUser` exists and does have `access token` (✓).
    2. Condition-2: Access Token stored in local storage.
  - Either condition is `true` → user is authenticated → It renders the child component.
  - if condition `not true` → user is not authenticated → Navigate to login page.
- 

## ▼ 6. FileReader Object

- `FileReader` is a built-in JavaScript object that allows web applications to read files (like images, text files, etc.) from the user's local device.
- It can read the file contents as different types of data, such as text or binary data.
- Here, `FileReader` is used to read the image file the user selects (`photoFile`) and then convert it into a base64-encoded string (which is a type of binary data) so it can be previewed on the page.

```
let reader = new FileReader();
reader.readAsDataURL(file); // This converts the file into a base64 string
(Way to encode Binary Data)
reader.onloadend = function (e) {
  setPhotoPreview(e.target.result); // e.target.result contains the base64 string
};
```

- The `reader.onloadend` is an event handler that is triggered when the `FileReader` has finished reading the file. Specifically, it occurs once the `FileReader` has either successfully completed reading the file or encountered an error.
- `reader.readAsDataURL(file)` is a method of the `FileReader` object in JavaScript. It reads the content of the file you provide as input and converts it into a **base64-encoded string** representing the file's data. This base64 string is prefixed with a `Data URL scheme`, which makes it suitable for embedding directly in web pages, such as in the `src` attribute of an `<img>` tag.
- In simple terms, `FileReader` helps in transforming the user's uploaded file into a format that can be displayed in the browser (like an image preview) before uploading it to the server.

## ▼ 7. FormData API (Object)

- `FormData` is a built-in JavaScript object used to easily construct a set of key-value pairs representing form fields and their values, which can then be sent to a server via an `.fetch()` or `HTTP` request (usually a `POST` request).
- `FormData()` → Creates a new `FormData` object.
- Here, `FormData` is used to package the file ( `photoFile` ) and send it to `Cloudinary` for upload.

```
const data = new FormData();
data.append("file", photoFile); // Add the file to FormData
data.append("upload_preset", "your_preset_name"); // Cloudinary upload
preset name (authentication)
```

```
data.append("cloud_name", "your_cloud_name"); // Cloudinary cloud name (unique to your account)
```

- The `FormData.append()` method is used to add a new key-value pair to a `FormData` object. This is particularly useful for constructing form data to send in an HTTP request (e.g., when uploading files or sending other form data to a server). If the key already exists, the new value will be added to the existing key rather than overwriting it.

## ▼ 8. Cloudinary API ( **SaaS** )

- **Cloudinary** is a cloud-based **Software-as-a-Service (SaaS)** solution designed for managing media assets such as images and videos.
- Cloudinary is an online tool that helps you handle images and videos for websites or apps.
- You can upload, store, edit, and share media files easily.
- It's especially helpful because it makes your media load faster and look better, whether people are using a computer, tablet, or phone.
- Developers can use its secure tools to manage media automatically, saving time and effort.
- Cloudinary is particularly useful for web and mobile applications, offering features that cater to both developers and creative designers
- **Cloudinary Base URL →**
  - In Cloudinary, the base URL for uploading images via the API depends on your cloud name, which is a unique identifier assigned to your Cloudinary account.



[https://api.cloudinary.com/v1\\_1/{cloud\\_name}/image/upload](https://api.cloudinary.com/v1_1/{cloud_name}/image/upload)

- Replace `{cloud_name}` with your actual Cloudinary cloud name.
- **Cloudinary Configuration →**



- The Cloudinary configuration is where you're setting up the API request to upload the image to Cloudinary.

```
const response = await fetch(  
  `https://api.cloudinary.com/v1_1/{}/image/upload`,  
  {  
    method: "POST",  
    body: data, // The FormData object with the file and additional fields  
  }  
);
```

- After the file is successfully uploaded, Cloudinary returns a response with information about the uploaded image, such as its URL. This URL is then used to update the user's profile photo in Firebase.

## ▼ 9. What is `doc()` & `setDoc()` & `getDoc()`

### ▼ 1. What is `doc()` ?

- **doc** is a function from the firebase firestore library used to create a reference to a specific document in a firebase collection.
- **Syntax** →



`doc(databaseInstance, collectionName, documentID)`

- `databaseInstance` : The database instance from `getFirestore()` (Here, `_DB` )
- `collectionName` : It is the name of collection. (Here `user_profile` )
- `documentID` : The unique identifier for every user. (Here `uid` )
- **Return Value:**
  - A `DocumentReference` object that points to the specific document in the firestore collection.

- This reference is used for further operations like setting, updating, or retrieving data.

---

## ▼ 2. What is `setDoc()` ?

- **setDoc** is a function used to write data to document in a firestore collection.
- It creates the document if it is doesn't already exist or overwrites the document if it does.
- **Syntax** →



`setDoc(documentReference, data, [options?])`

- **documentReference** : The reference to the firestore document. ( `doc()` )
- **data** : The object containing the data you want to store in the document.
- **options ?** : (optional) → An object with additional settings. `{merge: true}`
- **Return Value:** A `Promise<void>`
  - Resolves → when the data has been written successfully.
  - Rejects → with an error if the operation fails.

---

## ▼ 3. What is `getDoc()` ?

- In firebase, `getDoc()` is a function from the firestore library that is used to **retrieve a single document** from the firestore collection.
- It's a simple way to fetch data for a specific document by its ID.

### How it Works:

1. We have to use `doc()` to create a reference to the document you want to get.
  2. Then, we have to pass that reference to `getDoc()` to fetch the document's data.
-

- **Syntax →**



// Create a reference to the document

```
const docRef = doc(firestore, "collectionName", "documentID");
```

// Fetch the document

```
const docSnap = await getDoc(docRef);
```

// accessing the document data

```
docSnap.data()
```

- `doc()` : Used to point to the specific document .
- `getDoc()` : Actually fetches the document data from firestore.
- **Return Value:** The return value of `getDoc()` in Firebase Firestore is a `DocumentSnapshot` object.

---

## ▼ 4. What is a `DocumentSnapshot` ?

- A `DocumentSnapshot` represents a single document in Firestore.
- It provides methods to check if the document exists and to access its data and metadata.

---

## ▼ 5. What is `.data()` ?

- Returns the actual data from the document as a JavaScript object.
- If the document doesn't exist, this will return `undefined` .

---

## ▼ 10. What is Payload?

- In programming, a **payload** generally refers to the actual data or information being transmitted, stored, or processed.

- Th term is often used in the context of API's, databases, or communication between systems.
  - Here. payload object holds user-related data.
  - It represents the **core data**.
  - We can just consider payload as " `package` " of information.
  - It's being used as the main container to store all the user's profile information that will later sent to the firebase database.
- 

## ▼ 11. Steps to Store Data in Firestore Database

- `Step-1` : Payload creation (object)
  - `Step-2` : Creating document reference ( `doc()` )
  - `Step-3` : Writing data to firestore. ( `setDoc()` )
-