DATABASE PLAN

This section outlines the data structure of our Freelancer Management System, implemented using Firebase Firestore — a NoSQL, document-oriented cloud database. The database is structured into multiple collections(folder), each representing a core entity of the system. Documents within these collections store relevant data fields, and references are used to model relationships**.**

1. Step-by-Step Flow Explanation

Payment Transaction After Project Completion

1. **Client initiates payment** — After the job is completed , the client triggers a payment through the platform.
2. **Payment processing (handled by external gateway or mock-up)** — Once payment is processed, the backend confirms transaction success.
3. **New document in /payments**:
   - Created with job_ref, freelancer_ref, amount, status: "Paid", and payment_date.

2. Deployment and Storage Strategy

Technology Justification & Implementation Details

Why Firebase Firestore

We selected **Firebase Firestore** as our database solution because of its flexibility, scalability, and seamless integration with modern web technologies. Key reasons include:

- **Real-time updates**: Firestore allows real-time synchronization of data across clients, which is ideal for features like live project updates and notifications.
- **NoSQL schema**: Its document-based structure gives us the freedom to model data with flexible fields, ideal for a system with varying user roles and actions.
- **Scalability**: Firestore automatically scales to handle more users and data without requiring manual configuration or provisioning.
- **Easy integration**: Tight integration with Firebase Authentication, Hosting, and Cloud Functions simplifies full-stack development.

Hosting: Firebase Hosting for Static Web App

Our application is a static web app built with frontend technologies like HTML, CSS, and JavaScript. We are using **Firebase Hosting** for deployment because:

- It provides **fast and secure delivery** via global CDN.
- Offers **one-command deployment** using the Firebase CLI.
- Supports **custom domain configuration** and **automatic SSL encryption**.

Firestore Security Rules

To maintain data privacy and system integrity, we will implement **Firestore Security Rules** with **role-based access control**. Example access logic includes:

- Only authenticated users can read/write to the database.
- Clients can only access and modify their own project data.
- Freelancers can only apply to projects and view their own applications.
- Admin-level access can be defined for monitoring and moderation tasks (if needed).

Security rules will use custom claims or user role fields stored in the /users collection to enforce these restrictions.

Authentication: Firebase Authentication

We are using **Firebase Authentication** to manage user accounts securely. Features include:

- Support for **email and password** login (with future support for Google login if needed).
- Seamless integration with Firestore — user ID (uid) is used to match user data in the /users collection.
- Secure handling of account creation, login, and session management.
- Built-in support for **email verification**, password resets, and multi-device logins.

---

3. RELEVANT DATA FIELDS

1. **users** Collection

This collection stores shared information for all users, including both clients and freelancers. Authentication is managed using Firebase Authentication, and each user document matches the Firebase uid(user id). Fields:

- full_name: string
- email: string
- user_type: string ("client" or "freelancer")
- created_at: timestamp
- is_verified: boolean

---

2. clients Collection

Stores client-specific information, such as location and their posted projects. Fields:

- user_ref: reference to /users
- location: string

---

3. freelancers Collection

Stores freelancer-specific profile data including skills, experiencing. Fields:

- user_ref: reference to /users
- skills: array of strings
- experience_level: string
  - full name : string
  - email address : string
  - role : string
  - time stamp : date

---

4. payments Collection

Manages payment transactions between clients and freelancers. Fields:

- job_ref: reference to /projects
- freelancer_ref: reference to /freelancers
- amount: number
- status: string ("Paid", "Pending", "Failed")
- payment_date: timestamp