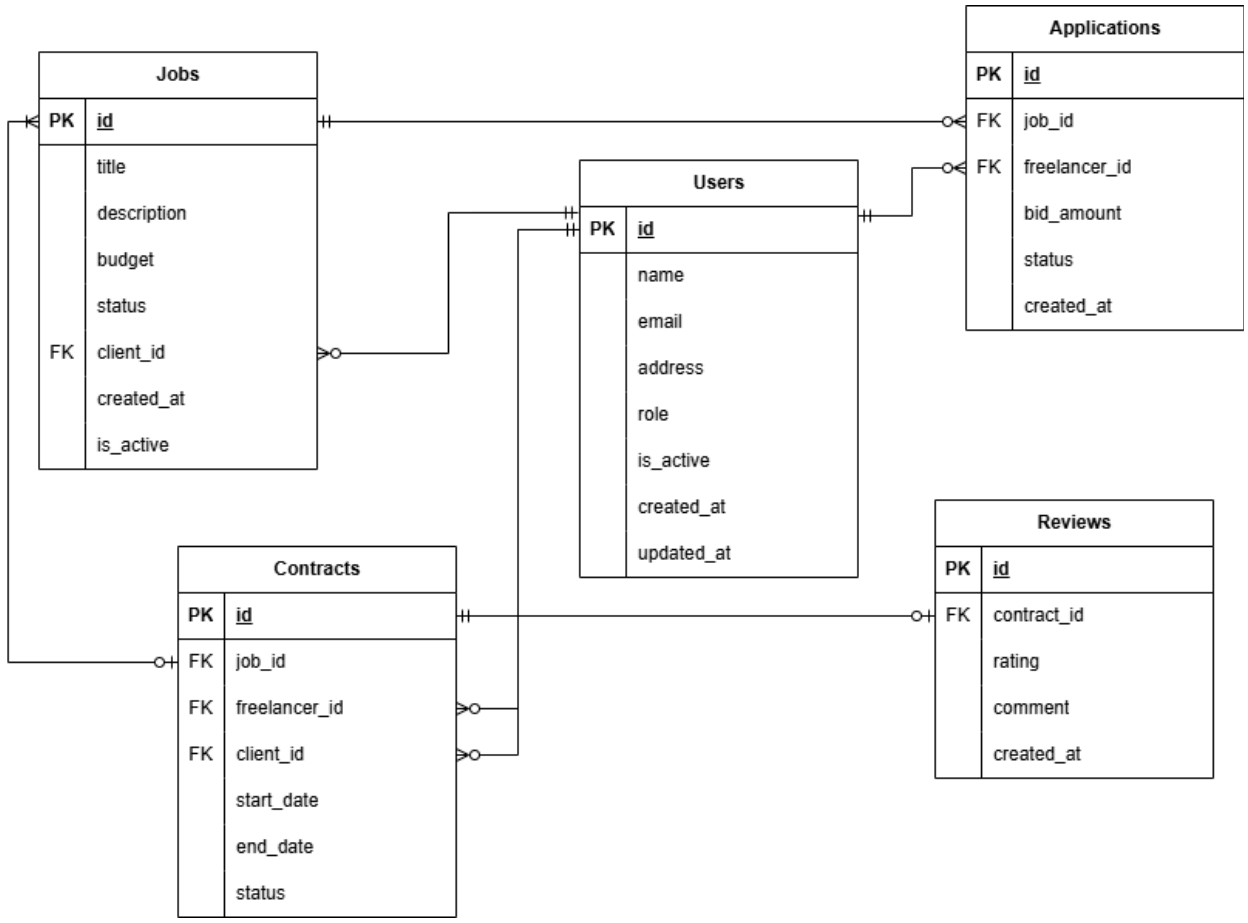
Entity Relationship Diagram and Explanation



The Freelancer Job Board API is structured using a relational database model optimised to Third Normal Form (3NF). The ERD ensures data integrity, avoids redundancy, and maintains clear relationships between entities. Below is a breakdown of each table and its relationships:

Entities and Relationships

1. Users (PK: id)

Represents all freelancers and clients on the platform.

- Attributes: name, email, address, role (freelancer/client/admin), is_active, created_at, updated_at
- Relationships:
    - One-to-Many with Jobs → A client can post multiple jobs. (client_id in Jobs)
    - One-to-Many with Contracts → A freelancer can have multiple contracts. (freelancer_id in Contracts)
    - One-to-Many with Applications → A freelancer can apply for multiple jobs. (freelancer_id in Applications)
    - One-to-Many with Reviews → Users can be reviewers or reviewees via the Contracts table.

---

2. Jobs (PK: id)

Represents freelance job postings created by clients.

- Attributes: title, description, budget, status, created_at, is_active, client_id (FK → Users.id)
- Relationships:
    - One-to-Many with Applications → A job can receive multiple applications. (job_id in Applications)
    - One-to-One with Contracts → A job can have none or one contract. (job_id in Contracts)

---

3. Applications (PK: id)

Tracks applications from freelancers for jobs.

- Attributes: job_id (FK → Jobs.id), freelancer_id (FK → Users.id), bid_amount, status, created_at
- Relationships:
    - Many-to-One with Jobs → A job can have multiple applications.
    - Many-to-One with Users (Freelancers) → A freelancer can apply to multiple jobs.

---

4. Contracts (PK: id)

Represents formal agreements between clients and freelancers after job acceptance.

- Attributes: job_id (FK → Jobs.id), freelancer_id (FK → Users.id), client_id (FK → Users.id), start_date, end_date, status
- Relationships:
    - One-to-One or One-to-Many with Jobs → A job may result in one or more contracts, depending on if multiple freelancers are hired.
    - Many-to-One with Users (Clients & Freelancers) → Links both the client and freelancer to a contract.
    - One-to-One with Reviews → A contract can have one review once completed.

---

5. Reviews (PK: id)

Captures ratings and feedback from freelancers and clients after contract completion.

- Attributes: contract_id (FK → Contracts.id), rating, comment, created_at
- Relationships:
    - One-to-One with Contracts → A contract receives one review upon completion.

Normalization & Optimisation (3NF Compliance)

- 1NF (First Normal Form) → Each table has a primary key and stores atomic (indivisible) values.
- 2NF (Second Normal Form) → All non-key attributes are fully dependent on the primary key (e.g., no job details stored in Applications).
- 3NF (Third Normal Form) → No transitive dependencies (e.g., freelancer and client details are not duplicated across tables, instead referenced via FKs).
- Higher Normalization Considerations: The model aligns with the Boyce-Codd Normal Form (BCNF) since there are no partial dependencies.

Chosen Database System: PostgreSQL

For this project, I have chosen PostgreSQL, an advanced open-source relational database management system (RDBMS) known for its strong reliability, extensibility, and standards compliance. PostgreSQL is an excellent choice for a Freelancer Job Board API because it provides robust support for complex queries, data integrity, and scalability.

Why PostgreSQL?

1. **Relational Structure & ACID Compliance:**
   PostgreSQL is an ACID-compliant (Atomicity, Consistency, Isolation, Durability) database, ensuring that transactions are executed reliably. This is crucial for a job board where data integrity is essential—for example, ensuring that jobs, applications, and payments are correctly processed without data corruption.

2. **SQL Compliance & Extensibility:**
   PostgreSQL follows the SQL standard while offering powerful extensions like PostGIS (for geolocation) and JSONB (for semi-structured data). It allows a mix of structured and unstructured data, making it flexible.

3. **Scalability & Performance:**

   - PostgreSQL handles large-scale applications efficiently with indexing, partitioning, and query optimisation.
   - It supports horizontal and vertical scaling, making it ideal for a growing application with increasing freelancer-client interactions.

4. **Strong Security Features:**
   PostgreSQL provides role-based access control (RBAC), SSL encryption, and built-in authentication methods (e.g., password-based, certificate-based, or external authentication).

5. **ORM Compatibility (SQLAlchemy):**
   Since I am using SQLAlchemy, PostgreSQL integrates seamlessly with it, allowing me to interact with the database in an efficient and structured way.

Comparison with Other Database Systems

To justify the choice of PostgreSQL, let's compare it with other popular database types:

1. PostgreSQL vs MySQL (Relational DB)

Feature	PostgreSQL	MySQL
ACID Compliance	Fully ACID-compliant	ACID-compliant but with some exceptions
Complex Queries	Handles complex queries well	optimised for read-heavy workloads
JSON Support	JSON & JSONB for semi-structured data	Basic JSON support (less efficient)
Extensibility	Supports procedural languages, full-text search, and indexing	Limited extensibility
Concurrency Control	Uses MVCC (better concurrency & performance)	Uses table-level locking more often

Verdict: PostgreSQL is superior for complex queries, extensibility, and JSON data handling. MySQL is simpler but less feature-rich.

---

2. PostgreSQL vs MongoDB (NoSQL)

Feature	PostgreSQL	MongoDB
Data Structure	Relational (tables & rows)	Document-based (JSON-like BSON)
Schema Enforcement	Strongly enforced schema (structured data)	Schema-less (flexible data)
Query Language	SQL	MongoDB Query Language (MQL)
Transactions	ACID transactions	Multi-document transactions (added later)

Scalability	Vertical & horizontal scaling	High horizontal scaling

Verdict: If my project required highly dynamic or semi-structured data (e.g., social media feeds), MongoDB would be useful. However, since I need structured relationships (Users, Jobs, Applications, Reviews, Contracts) with strong data integrity, PostgreSQL is the better choice.

---

3. PostgreSQL vs Firebase (Realtime NoSQL)

Feature	PostgreSQL	Firebase
Type	Relational SQL Database	NoSQL Realtime Database
Schema	Structured & strict	Unstructured & flexible
Query Power	Advanced SQL & indexing	Basic querying
Performance	optimised for structured data	optimised for real-time syncing
Offline Mode	Not natively designed for offline mode	Designed for mobile apps with offline syncing

Verdict: Firebase is ideal for real-time applications like messaging but not suitable for a structured job board that requires complex queries, relationships, and transactions.

Final Justification for PostgreSQL

Considering the project requirements—structured job listings, user accounts, applications, reviews, and contracts—PostgreSQL is the best fit because:

- It ensures data integrity and supports relationships between entities.
- It allows complex queries for filtering jobs, applications, and transactions.
- It scales well with indexing and optimization features.
- It integrates well with SQLAlchemy, making ORM-based database management easier.