Assumption and design:

Entities:

1. **User** is an entity that includes UserName, UserID, Password, is_Admin and etc.
2. **Job** entities include every job listed on the platform.
3. **Company** is an entity, it includes the company name, the location, and related information in the company overview.
4. **Review** is an entity. In order to protect users' privacy and allow users to express their true thoughts about the jobs, all the reviews are anonymous. Reviews are not an attribute of a job, because the same job can have multiple reviews.
5. **UploadedHistory** is an entity. It stores jobs uploaded by users. This is distinct from the Job entity because it tracks who uploaded the job and includes a comment field for Admin to approve or reject.

Relationships:

- **Favorite: User - Job**: many-to-many relationship (a user can have many favorite jobs)
- **Provide: Company - Job:** one-to-many (a job can only have one company, one company can have many jobs)
- **Upload: User - UploadedHistory:** one-to-many (a user can upload many jobs, but a job can only be uploaded by one user)
- **ReviewOfJob: Job - Review:** one-to-many (a job can have multiple reviews, a review can only be associated with one job)

Relational schema:

We create tables for each entity. For many-to-many relationships, we create tables. For one-to-many relationships, we add foreign keys instead of creating tables.

1. **User**:(UserName: VARCHAR(255), UserID: VARCHAR(50) [PK], Password: VARCHAR(50), is_Admin: BOOLEAN, FirstName: VARCHAR(255), LastName: VARCHAR(255), Age: INT, Location: VARCHAR(50), PhoneNumber: VARCHAR(50), EmailAddress: VARCHAR(255))

   Stores user account and profile information.

2. **Job**: **(**JobID: VARCHAR(255) [PK], JobTitle: VARCHAR(255), JobSnippet: VARCHAR(4096), JobLink: VARCHAR(255), Sponsored: BOOLEAN, Salary: INT, Rating: INT, CompanyName: VARCHAR(255)[FK to Company.CompanyName], ApprovalStatus: BOOLEAN**)**

   Stores information about different job listings (JobID, title, description, salary, etc.).

3. **Company**:(CompanyName: VARCHAR(255)[PK], Location: VARCHAR(255)), CompanyOverview: VARCHAR(255))

   Store information about the companies

4. **Review:** (ReviewID: VARCHAR(50)[PK], JobID: VARCHAR(255)[FK to Job.JobID], Content: VARCHAR(4096), Rating: Real**)**

   Store reviews and ratings submitted by users for jobs

5. **UploadedHistory**: **(**UploadID: VARCHAR(50)[PK], UserID: VARCHAR(50) [FK to User.UserID], JobID: VARCHAR(50)[FK to Job.JobID], AdminComment:VARCHAR(512)**)**

   Store jobs the user uploaded

6. **Favorite: (**JobID: VARCHAR(255) [FK to Job.JobID], UserID: VARCHAR(50) [FK to User.UserID], PK[JobID, UserID]**)**

Normalization Proof:

We find that the schema we converted to satisfies BCNF, so we just prove that.

- **User:**
  - UserID → All other attributes
  - Candidate Key: UserID
  - Normalization Level: BCNF
  - All attributes are fully functionally dependent on the primary key, and there are no transitive dependencies. The table is in BCNF.
- **Job:**
  - JobID → All other attributes
  - Candidate Key: JobID
  - Normalization Level: BCNF
  - All attributes are fully functionally dependent on the primary key, and there are no transitive dependencies. The table is in BCNF.
- **Company:**
  - CompanyName → Location, CompanyOverview
  - Candidate Key: CompanyName
  - Normalization Level: All attributes are fully functionally dependent on the primary key, and there are no transitive dependencies. The table is in BCNF.
- **Review:**
  - ReviewID→ JobID, Content, Rating
  - Candidate Keys: ReviewID
  - Normalization Level: All attributes are fully functionally dependent on the primary key, and there are no transitive dependencies. The table is in BCNF.
- **UploadedHistory:**
  - ReviewID → UserID, JobID, AdminComment

    - (UserID, JobID) → AdminComment
    - Candidate Keys: UploadID, (UserID, JobID)
    - Normalization Level: All attributes are fully functionally dependent on the primary key, and there are no transitive dependencies. The table is in BCNF.
- **Favorite (relationship):**
    - Attributes: UserID, JobID
    - Does not have functional dependencies, therefore the table is in BCNF.

Fix suggestions for the search feature in stage 1:

In stage 1, we have already designed a 'search' feature. In our stage 1 document, the first picture shows that users can input some job features in the corresponding blanks and click on the 'search' button to search for jobs. The search results will be shown in the 'list' part, and the visualization will be on the right of the page. However, we missed a description of this feature. Here is the supplement: Users can fill in one or multiple job-related attributes to perform a search. For example, users can enter the keyword "Software Development Engineer" in the job title field, input "San Jose" in the location field, and set the lower and upper limits of the salary field to 8000 and 12000, respectively. By clicking search, they can retrieve all job information that matches the criteria.

The functionality will be implemented using text-based queries on the job dataset. We will join Job, Company and other necessary tables, and use full-text search and keyword matching to retrieve relevant job postings. The filtering options will be integrated with the search to allow users to refine their results.