JOB–Z

- Your Ultimate Job Portal

A Full-Stack Job Portal Web App

Presented By Hrishi Mehta with all the efforts. 😊

Introduction 😄

What is Job-Z?

- Job-Z is a role-based job portal platform
- Allows users to register as job seekers or recruiters
- Job seekers can apply for jobs
- Recruiters can post jobs via their associated companies
- Built with a full-stack architecture
- User friendly interface

User Roles 😊

-Role-Based Access System

-- 👤 **Job Seekers:**

- Register and log in
- Browse and apply to jobs
- View all applied jobs

-- 🏢 **Recruiters:**

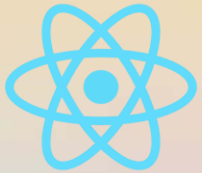
- Register and log in
- Create or manage a company profile
- Post jobs under their company
- View applications for their jobs

Key Features 😄

- **Core Functionalities of Job-Z**
  - Role-based authentication and dashboards
  - Company-based job postings
  - File uploads (CVs, company logos, user profile pictures)
  - JWT-based secure routes
  - Real-time job posting and applying
  - Responsive UI built with Tailwind CSS
  - Clean and user-friendly interface
  - No pricing plans – completely free for all users

Tech Stack – Frontend

- **React**
  - JavaScript library for building UI components
  - Used to create dynamic job seeker and recruiter interfaces



- **Vite**
  - Frontend build tool for faster development
  - Used to serve and bundle the React app efficiently

- **Redux Toolkit**
  - State management library
  - Used for managing auth, job data, applications, and companies

Tech Stack – Frontend

- **Tailwind CSS**
  - Utility-first CSS framework
  - Used for responsive and modern UI design

- **Axios**
  - Promise-based HTTP client
  - Used for sending API requests to the backend (e.g. apply & post jobs)

- **Framer Motion**
  - Animation library for React
  - Used to add smooth transitions and animations in UI

- **Radix UI**
  - Accessible UI components for building modern interfaces
  - Used for popovers, dialogs, radio groups, and other UI elements

Tech Stack – Backend & Tools

- **Node.js**
  - JavaScript runtime environment
  - Used as the core server environment for the backend

- **Express.js**
  - Web application framework for Node.js
  - Used to create APIs, manage routes, and handle middleware

- **MongoDB**
  - NoSQL database
  - Stores user data, job posts, companies, applications

Tech Stack – Backend & Tools

- **Mongoose**
  - MongoDB object modeling tool
  - Used to define schemas and interact with MongoDB collections

- **JWT (JSON Web Token)**
  - Authentication mechanism
  - Used to protect private routes (like apply/post job)

- **Multer**
  - Middleware for handling file uploads
  - Used to upload CVs and company logos

- **Cloudinary**
  - Cloud storage service
  - Stores and retrieves images and files (like resumes/logos) by users

Other Tools

- **Nodemon** – Auto-restarts server on file changes during development
- **dotenv** – Manages environment variables like DB URI and JWT secret
- **ESLint** – Lints code to maintain coding standards
- **PostCSS** – Used with Tailwind for processing CSS

Folder Structure 📂

- **Organized Folder System**
  - `frontend/` - React app with role-based UI
  - `backend/` - Node.js/Express API with routes, models, controllers

- **MVC Architecture**
  - `models/` – MongoDB schemas
  - `controllers/` – Business logic
  - `routes/` – Endpoint definitions
  - `middleware/` – JWT & file upload logic

User Registration – Data Flow 👨‍💻

- **Register and Choose Role**
- User visits Register.jsx
- Fills in details (name, email, password, role: recruiter/job-seeker)
- Data sent to POST `/api/user/register` via Axios
- **user.controller.js** handles input validation and password hashing
- User saved in MongoDB via **user.model.js**
- JWT token issued and stored in cookies
- Role-based dashboard rendered after registration

User Login – Data Flow 👨‍💻

- **Login and Choose Role**
- User logs in via Login.jsx
- Credentials sent to `POST /api/user/login`
- Validated in **user.controller.js**
- If valid, JWT is returned and stored in cookies
- User redirected to their respective dashboard based on role

Posting a Job – Data Flow 💼

->How a Recruiter Posts a Job

1. Recruiter logs in
2. Creates or selects a company
3. Fills job form in PostJob component
4. Data sent via Axios to `POST /api/job`
5. Validated by middleware (`isAuthenticated.js`)
6. `job.controller.js` stores job in MongoDB
7. Linked to the recruiter's company

Applying to a Job – Data Flow 💼

->How a Job Seeker Applies

1. Job seeker logs in
2. Browses jobs from `Jobs.jsx`
3. Clicks Apply → Sends request to `POST /api/application`
4. JWT middleware validates token
5. `application.controller.js` stores application
6. `application.model.js` saves jobId, userId, and optional resume
7. Recruiter can view applicants via company dashboard

Middleware Usage

- **Purpose of Middleware**
- `isAuthenticated.js`:
  - Verifies JWT token
  - Secures protected routes (apply, post jobs)

- `multer.js`:
  - Handles file uploads (e.g., CVs, logos)
  - Works with Cloudinary for storage

MongoDB Models

- **Data Schemas Used**
  - **User Model**: stores job seeker or recruiter role, name, email, password
  - **Company Model**: stores company details linked to recruiter
  - **Job Model**: job details linked to companyId
  - **Application Model**: stores jobId, userId, and optional CV link

API Endpoints

- **API Routes**
- `POST /api/user/register` – User registration
- `POST /api/user/login` – User login
- `POST /api/company` – Company setup
- `POST /api/job` – Post a job
- `GET /api/job` – Get all jobs
- `POST /api/application` – Apply to a job
- `GET /api/application/:jobId` – Get applicants for a job

ER Diagram

This system allows two types of users—**Recruiters** and **Job Seekers**—to interact within a job portal where recruiters post jobs through companies, and job seekers apply to those jobs.

1. **User**

- Central entity for authentication and identification.
- Contains shared attributes such as `name`, `email`, `phone`, and `role`.
- Differentiated by role:
    - `job_seeker`: A user who applies for jobs.
    - `recruiter`: A user who posts jobs through companies.

  Relationships:
- A recruiter **can own multiple companies**.
- A job seeker **has a one-to-one relationship with a profile**.

ER Diagram 🔷

2. **Profile**

- Represents additional data for **Job Seekers** only.
- Stores professional information like `bio`, `skills`, `resume`, and `profile photo`.
- **Relationships**:
- One-to-one with `User` (only applicable when the user's role is `job_seeker`).

3. **Company**

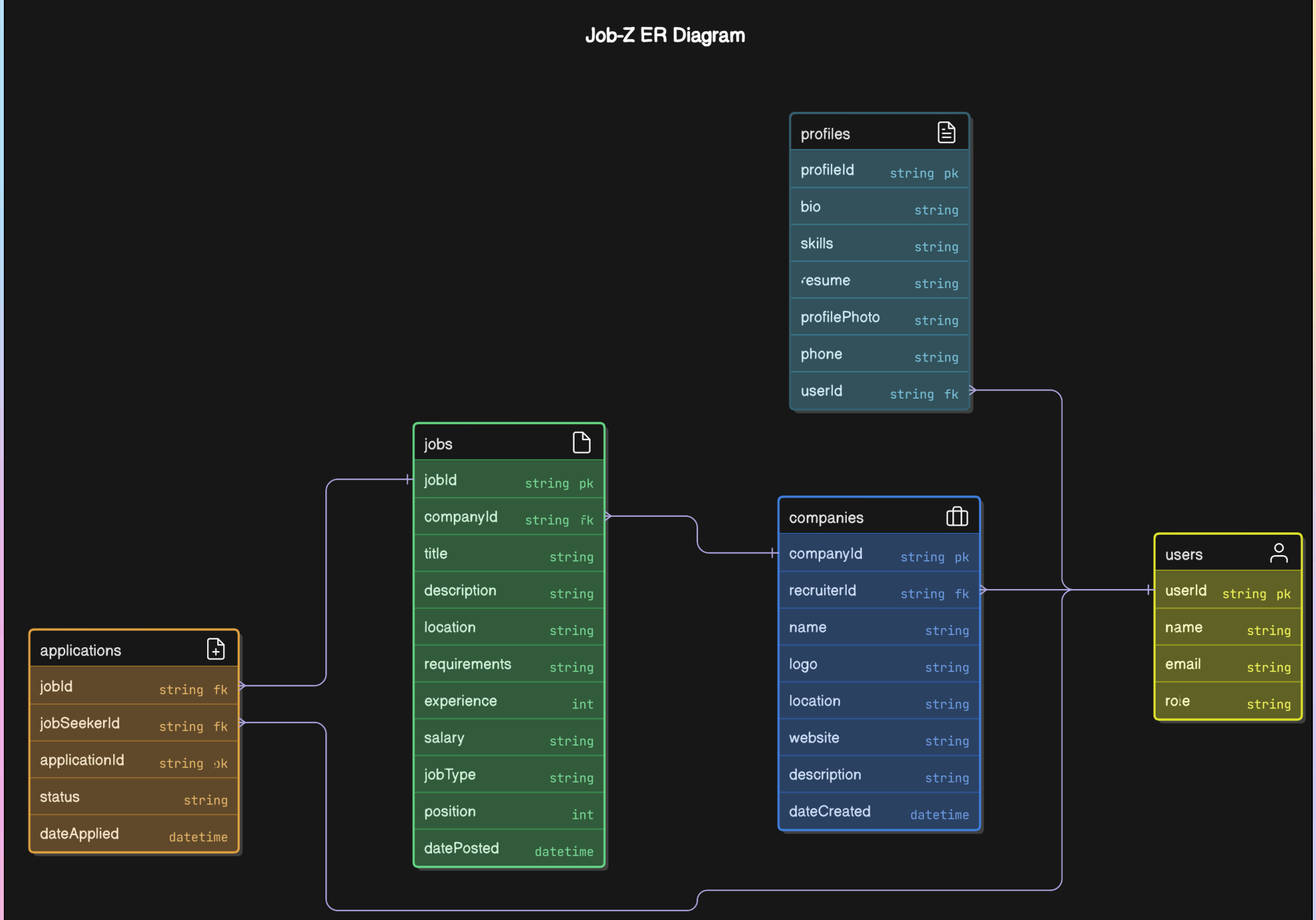
- Represents organizations created and managed by **Recruiters**.
- Contains details like `name`, `description`, `location`, `website`, `logo`, and `dateCreated`.

  Relationships:
- A **recruiter** can create **multiple companies** (one-to-many).
- A **company** can post **multiple jobs**.

ER Diagram

- 4. **Job**
- Represents a job listing posted under a specific company.
- Includes job-specific data such as:
  - `title`, `description`, `location`, `salary`, `jobType`, and `datePosted`.
  - Derived UI fields like `companyName` and `companyLogo` (can be fetched via relation).
  - **Relationships**:
- Belongs to one **company**.
- Can receive **many applications** from different job seekers.

- 5. **Application**
- Junction table between `User` (as Job Seeker) and `Job`.
- Represents a job seeker's application to a job.
- Stores:
  - `dateApplied`
  - `status` (e.g., `applied`, `accepted`, `rejected`)
  - Actions like "Accept" or "Reject" can be performed by recruiters.
  - **Relationships**:
- Many-to-one with `Job`.
- Many-to-one with `User` (where user role is `job_seeker`).
- Captures **many-to-many** relationship between **jobs** and **job seekers**.


Job-Z ER Diagram
profiles
profileId string pk
bio string
skills string
resume string
profilePhoto string
phone string
userId string fk
jobs
jobId string pk
companyId string fk
title string
description string
location string
requirements string
experience int
salary string
jobType string
position int
datePosted datetime
companies
companyId string pk
recruiterId string fk
name string
logo string
location string
website string
description string
dateCreated datetime
users
userId string pk
name string
email string
role string
applications
jobId string fk
jobSeekerId string fk
applicationId string pk
status string
dateApplied datetime

Class Diagram

Entities and Attributes:

- **User** (Dark Green Box)
  - `userId: string` (PK)
  - `name: string`
  - `email: string`
  - `role: string` ('job_seeker' or 'recruiter')

- **Profile** (Peach Box)
  - `profileId: string` (PK)
  - `userId: string` (FK, links to User)
  - `bio: string`
  - `skills: string` (comma-separated)
  - `resume: string`
  - `profilePhoto: string`
  - `phone: string`

Class Diagram 🟥 🟪

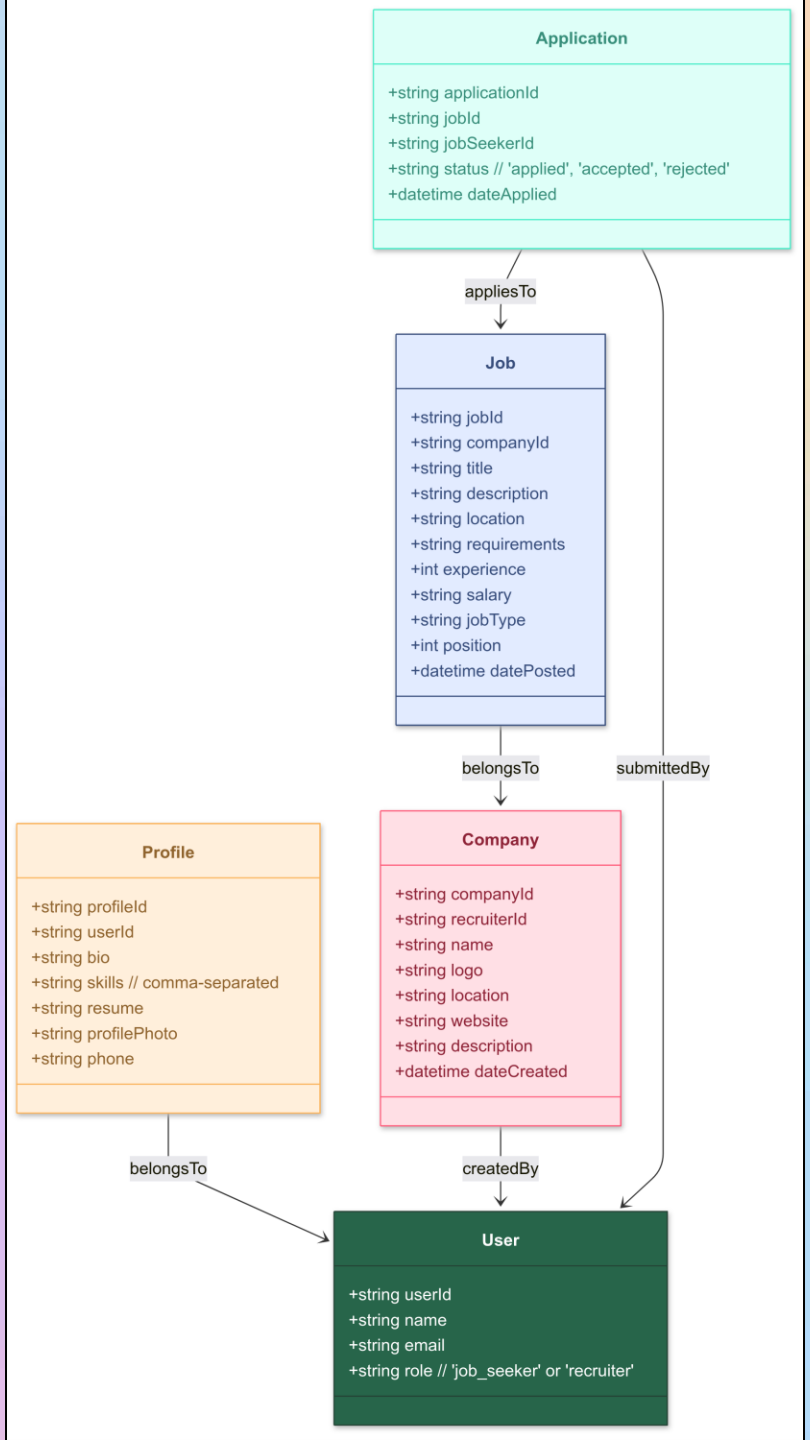
- **Company** (Red Box)
- `companyId: string` (PK)
- `recruiterId: string` (FK, links to User)
- `name: string`
- `logo: string`
- `location: string`
- `website: string`
- `description: string`
- `dateCreated: datetime`

- **Job** (Purple Box)
- `jobId: string` (PK)
- `companyId: string` (FK, links to Company)
- `title: string`
- `description: string`
- `location: string`
- `requirements: string`
- `experience: int`
- `salary: string`
- `jobType: string`
- `position: int`
- `datePosted: datetime`

Class Diagram

- **Application** (Orange Box)
  - `applicationId: string` (PK)
  - `jobId: string` (FK, links to Job)
  - `jobSeekerId: string` (FK, links to User)
  - `status: string` ('applied', 'accepted', 'rejected')
  - `dateApplied: datetime`

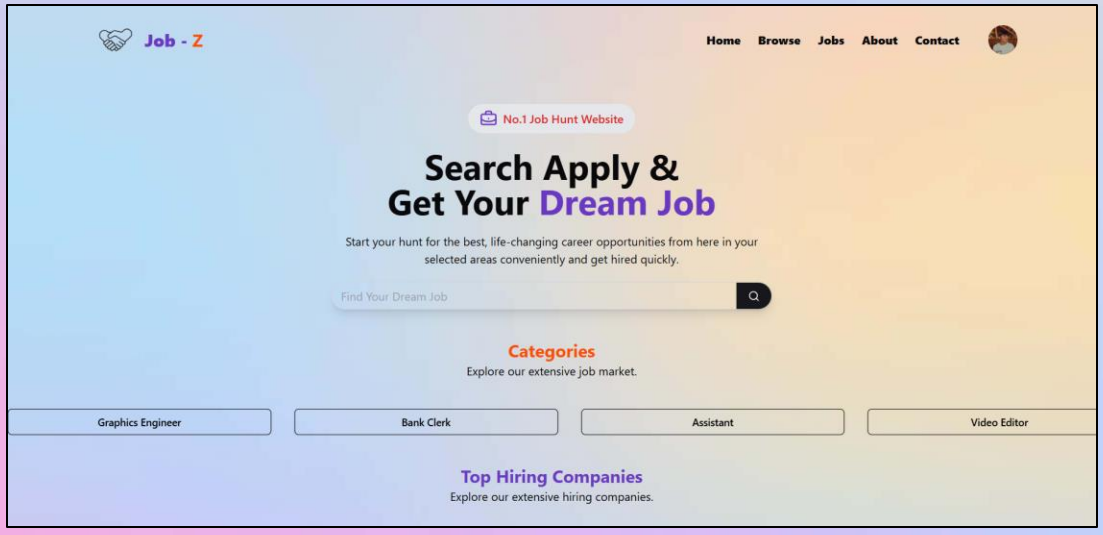
Class Diagram a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects

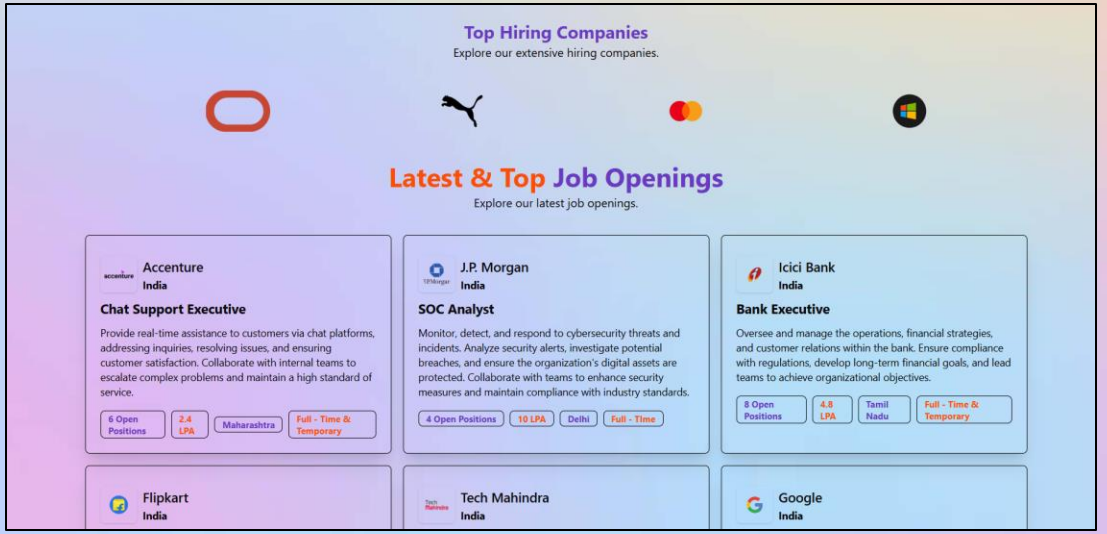

Application
+string applicationId
+string jobId
+string jobSeekerId
+string status // 'applied', 'accepted', 'rejected'
+datetime dateApplied
appliesTo
Job
+string jobId
+string companyId
+string title
+string description
+string location
+string requirements
+int experience
+string salary
+string jobType
+int position
+datetime datePosted
belongsTo
submittedBy
Profile
+string profileId
+string userId
+string bio
+string skills // comma-separated
+string resume
+string profilePhoto
+string phone
Company
+string companyId
+string recruiterId
+string name
+string logo
+string location
+string website
+string description
+datetime dateCreated
belongsTo
createdBy
User
+string userId
+string name
+string email
+string role // 'job_seeker' or 'recruiter'

Postman API Testing & Manual Testing

- Used Postman for testing all API endpoints
- Sample endpoints tested:
  - `POST /api/user/register`
  - `POST /api/job`
  - `GET /api/job`
  - `POST /api/application`
- Validated headers, body, status codes
- Manually tested integration of different units
- Manually tested User Interface & Functionality

JOB-Z JobSeeker Home Page 🖼️





JOB-Z Recruiter Home Page 🖼️

Advantages & Disadvantages

Advantages 😇

- **Role-based Access:** Clear role separation for Job Seekers and Recruiters.
- **Real-time Job Posting:** Job postings and applications are reflected immediately, ensuring up-to-date information.
- **Secure Authentication:** JWT-based authentication ensures secure and efficient login and session management.
- **Scalable Architecture:** The modular architecture allows easy scaling and future expansion of features.
- **Cloud-based Media Management:** Cloudinary and Multer ensure efficient handling and storage of media like profile pictures and resumes.

Advantages & Disadvantages

Disadvantages 😈

- **No Admin Role:** The system does not have an admin role, which can limit moderation or oversight in certain scenarios.
- **Dependence on External Services:** The app relies on external services like Cloudinary for media storage, meaning any downtime or service issues can affect functionality.
- **Role Validation Complexity:** The job seeker/recruiter role validation could lead to restrictions in certain user flow scenarios.
- **Limited Third-party Integrations:** Currently, integrations are limited to media storage, with no third-party job board integration.

Summary 🤓

- **Key Features of Job-Z**
- Full-stack job portal with role-based access (Job Seekers & Recruiters)
- Secure JWT Authentication with cookie-based sessions
- Real-time job posting and application process
- Company-based job system for more accurate role matching
- Efficient media handling with Cloudinary and Multer
- Scalable and modular design for future feature additions

Thank You 🫡

- **Thank you for your attention!**
- Feel free to connect or ask any questions.
- **Contact Info:**
- 🌐 https://github.com/mehtahrishi/Job_Z
- 📧 mehtahrishi45@gmail.com
- 💬 https://www.linkedin.com/in/hrishi-mehta-889732256/
- 💼 https://portfolio-theta-lemon-33.vercel.app/