**🧾 Project Synopsis**

**📘 MCA Major Project**

**Project Title: Job Portal for Informal Sector Workers**

**1. Introduction**

The informal sector includes millions of skilled and unskilled workers such as plumbers, electricians, domestic helpers, and daily wage laborers. However, they often lack access to a structured job platform, relying instead on word-of-mouth or middlemen. This project aims to create a web-based job portal to directly connect informal sector workers with employers, thereby improving transparency, accessibility, and job opportunities.

**2. Objective**

* To build a web platform that allows informal workers to register and find jobs.
* To allow employers to post job openings and hire suitable workers.
* To make job searching and hiring efficient, fair, and transparent.
* To enable direct interaction between workers and employers without middlemen.

**3. Scope of the Project**

The system will include:

* Registration/Login for both workers and employers.
* Worker profile creation with skills, experience, and availability.
* Job posting by employers.
* Worker-job matching based on skill and location.
* Search and filter functionality.
* A simple admin panel to manage content and users.

**4. Modules Description**

| **Module** | **Description** |
| --- | --- |
| **Worker Module** | Register/login, create profile, view/apply to jobs, manage availability. |
| **Employer Module** | Post jobs, view worker profiles, hire/contact workers. |
| **Job Module** | Post/view job listings, track application status. |
| **Admin Module** | Approve users, manage job listings and feedback. |
| **Search & Filter** | Filter workers and jobs based on skill, location, and experience. |

**5. Tools & Technologies Used**

| **Component** | **Technology** |
| --- | --- |
| **Frontend** | React.js, HTML5, CSS3 |
| **Backend** | Node.js, Express.js |
| **Database** | MongoDB with Mongoose |
| **Authentication** | JSON Web Tokens (JWT) |
| **Hosting** | Vercel (Frontend), Render (Backend), MongoDB Atlas |
| **Others** | Axios, React Router, CORS, dotenv |

**6. Expected Outcome**

* A fully functional, mobile-responsive job portal.
* Easy-to-use interface for workers and employers.
* A basic admin dashboard for platform moderation.
* Real-time job listing and hiring experience.

**7. Future Enhancements (Optional)**

* Multi-language support for better accessibility.
* Mobile app version of the platform.
* Ratings and reviews for workers and employers.
* In-app messaging and chat support.
* Payment tracking system for verified hires.

**📄 Software Requirements Specification (SRS)**  
**Domain:** Web Application Development  
**Technology Stack:** MERN (MongoDB, Express.js, React.js, Node.js)

**📑 1. Introduction**

**1.1 Purpose**

This SRS document outlines the functional and non-functional requirements of the Job Portal for Informal Sector Workers. The portal aims to bridge the gap between informal workers and employers by providing a simple and accessible job-matching platform.

**1.2 Intended Audience**

* Project Evaluators and MCA Faculty
* Developers (you)
* Future Maintainers

**1.3 Scope**

The portal will provide:

* User authentication (workers and employers)
* Profile management for workers
* Job posting and hiring functions for employers
* Search and filtering capabilities
* Admin panel for verification and moderation
* REST APIs and a responsive frontend

**📊 2. Overall Description**

**2.1 Product Perspective**

This will be a full-stack web application with a backend API and frontend UI. MongoDB stores all data including users, jobs, and applications.

**2.2 Product Functions**

* Register/Login via Email and Password
* Profile management for both user types
* Employers can post and manage job listings
* Workers can search and apply for jobs
* Admin dashboard to moderate the system

**2.3 User Classes & Characteristics**

| **User Role** | **Description** |
| --- | --- |
| Worker | Can register, build a profile, view/apply to jobs |
| Employer | Can register, post jobs, browse worker profiles |
| Admin | Manages users, reviews content, and handles reports |

**2.4 Assumptions & Dependencies**

* Users will need internet and browser access
* All users must have a valid email to register
* Hosted services like MongoDB Atlas and Vercel/Render will be available

**🔧 3. Functional Requirements**

| **ID** | **Requirement** |
| --- | --- |
| FR1 | The system must allow users to register and log in. |
| FR2 | Workers must be able to create/edit a profile. |
| FR3 | Employers must be able to post job listings. |
| FR4 | Workers must be able to browse and apply for jobs. |
| FR5 | Employers must be able to view and hire workers. |
| FR6 | Admin must be able to manage users and content. |

**⚙️ 4. Non-Functional Requirements**

| **Category** | **Requirement** |
| --- | --- |
| Performance | The app must respond within 2 seconds for most actions |
| Usability | The UI must be simple and mobile-responsive |
| Security | Passwords must be stored in hashed form; JWT must be used |
| Availability | System must be online 24/7 after deployment |
| Scalability | Should be able to handle increasing number of users |

**📈 5. System Design (Preview)**

(Full diagrams in the next step — System Design)

* **Frontend:** React.js with Router & Axios
* **Backend:** Node.js + Express REST APIs
* **Database:** MongoDB with collections like Users, Jobs, Applications
* **Authentication:** JWT with Role-based Access (Worker, Employer, Admin)

**📋 6. External Interface Requirements**

**6.1 User Interfaces**

* Simple sign-in/up forms
* Profile management forms
* Job listing and detail views
* Admin dashboard interface

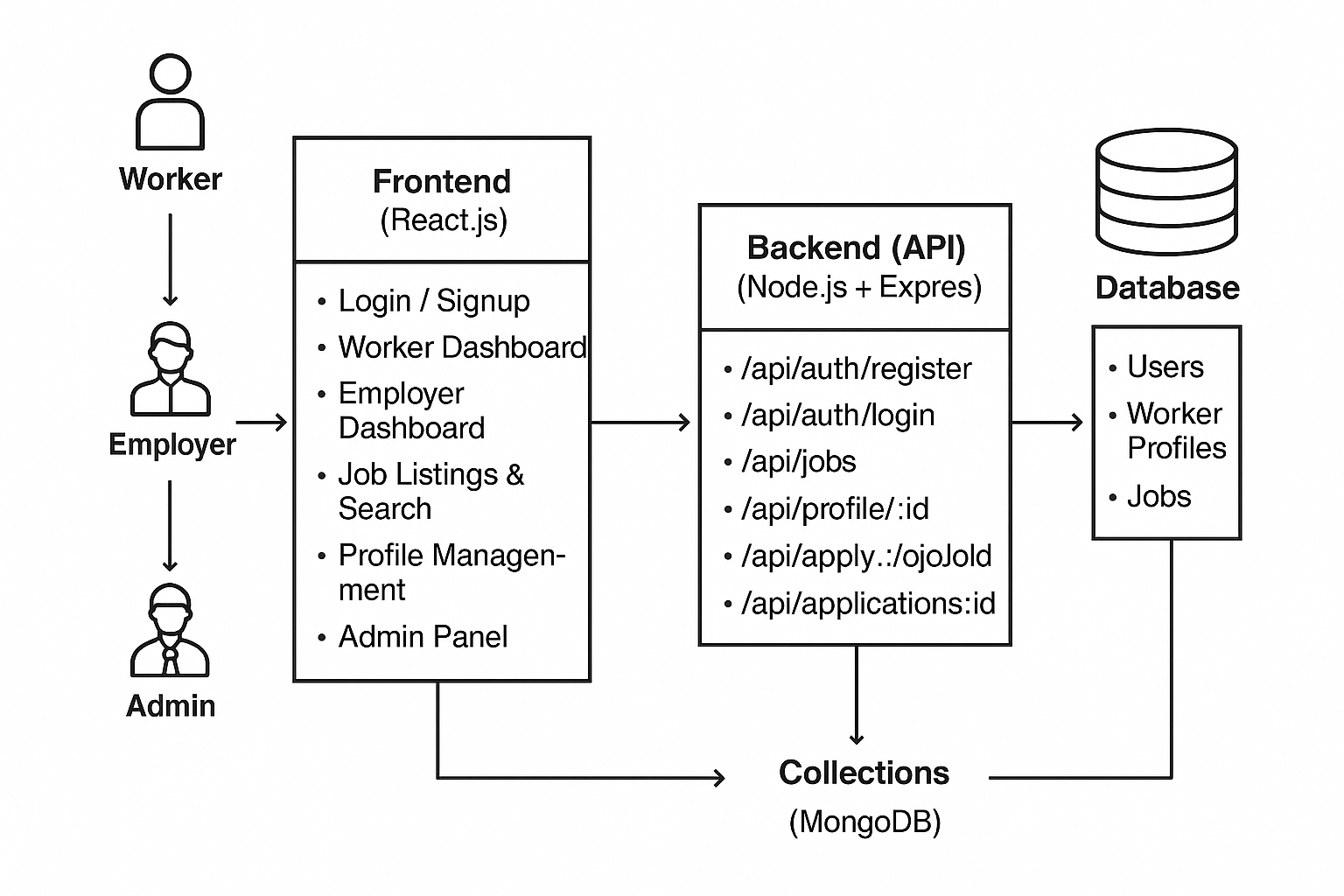
**6.2 Hardware Interfaces**

* Standard laptop or mobile device with internet access

**6.3 Software Interfaces**

* Browser (Chrome, Firefox, Edge, etc.)
* MongoDB Atlas
* Node.js/Express REST APIs

**System Design for Job Portal for Informal Sector Workers**



**1. Architecture Overview**

The system follows the **MERN stack architecture**:

[Frontend (React.js)] ⇄ [Backend (Node.js + Express)] ⇄ [Database (MongoDB)]

* The frontend handles the user interface and client-side logic.
* The backend exposes REST APIs that the frontend consumes.
* The database stores all persistent data.

**2. User Roles**

| **Role** | **Description** | **Main Actions** |
| --- | --- | --- |
| Worker | Informal sector workers | Register, create profile, search/apply for jobs |
| Employer | Job providers | Register, post jobs, view/apply worker profiles |
| Admin | Platform moderator | Manage users, approve content, remove inappropriate data |

**3. Modules and Components**

**Frontend (React.js)**

* **Authentication Module:** Login, Signup, Logout, Password reset
* **Worker Module:** Profile creation/edit, job search, job application, application status view
* **Employer Module:** Job posting, job listing management, worker search, applicant review
* **Admin Module:** User management, job management, reporting tools
* **Shared Components:** Navigation bar, notifications, footer

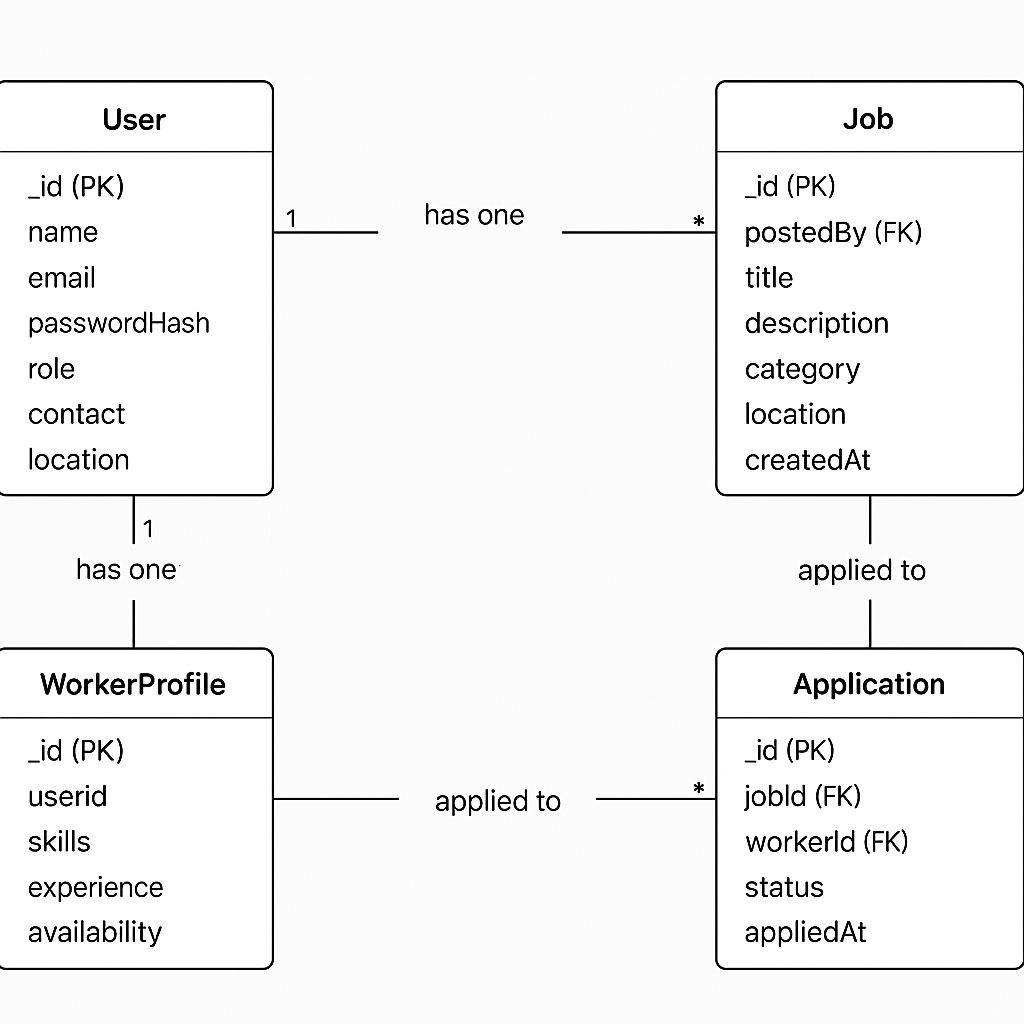
**4. Database Design**

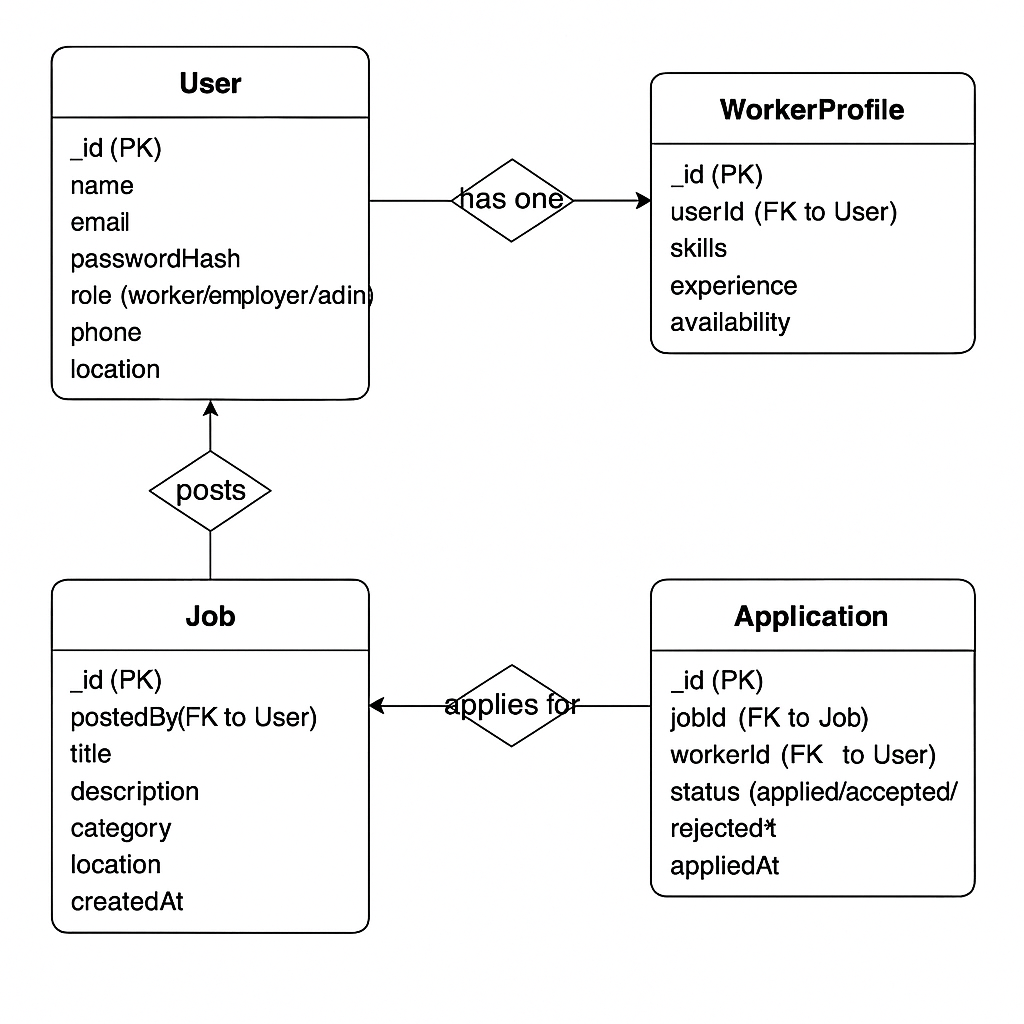
**Collections:**

| **Collection** | **Key Fields** | **Purpose** |
| --- | --- | --- |
| users | \_id, name, email, passwordHash, role, contact, location | Stores all users (workers, employers, admin) |
| workerProfiles | \_id, userId (ref), skills, experience, availability | Detailed worker info |
| jobs | \_id, postedBy (userId), title, description, category, location, createdAt | Job listings posted by employers |
| applications | \_id, jobId, workerId, status, appliedAt | Job applications |

**5. Entity Relationship Diagram (ERD) Description**

* A **User** can be a Worker or Employer (based on role).
* Each Worker has one **WorkerProfile** linked by userId.
* Employers post multiple **Jobs** (postedBy userId).
* Workers apply to jobs, creating **Applications** linking jobId and workerId.
* Admin can manage all collections.





**6. API Endpoints**

| **Method** | **Endpoint** | **Description** | **Access** |
| --- | --- | --- | --- |
| POST | /api/auth/register | Register new user | Public |
| POST | /api/auth/login | Login and get JWT token | Public |
| GET | /api/jobs | List all jobs | Public |
| POST | /api/jobs | Create a new job posting | Employer |
| GET | /api/profile/:id | Get worker profile by user ID | Public |
| POST | /api/apply/:jobId | Apply for a job | Worker |
| GET | /api/applications/:userId | Get applications for a user | Worker/Employer |
| DELETE | /api/job/:jobId | Delete job posting | Employer/Admin |

**7. Data Flow**

1. **Registration/Login:** User registers or logs in → backend validates → JWT token sent → frontend stores token.
2. **Job Posting:** Employer posts a job → backend saves job in DB → job visible in job listings.
3. **Job Search:** Worker searches jobs → frontend requests job list → backend queries DB → results returned.
4. **Job Application:** Worker applies → application saved → employer can view applicants.
5. **Admin Operations:** Admin reviews/modifies users and jobs via backend APIs.

**📘 What is a DFD?**

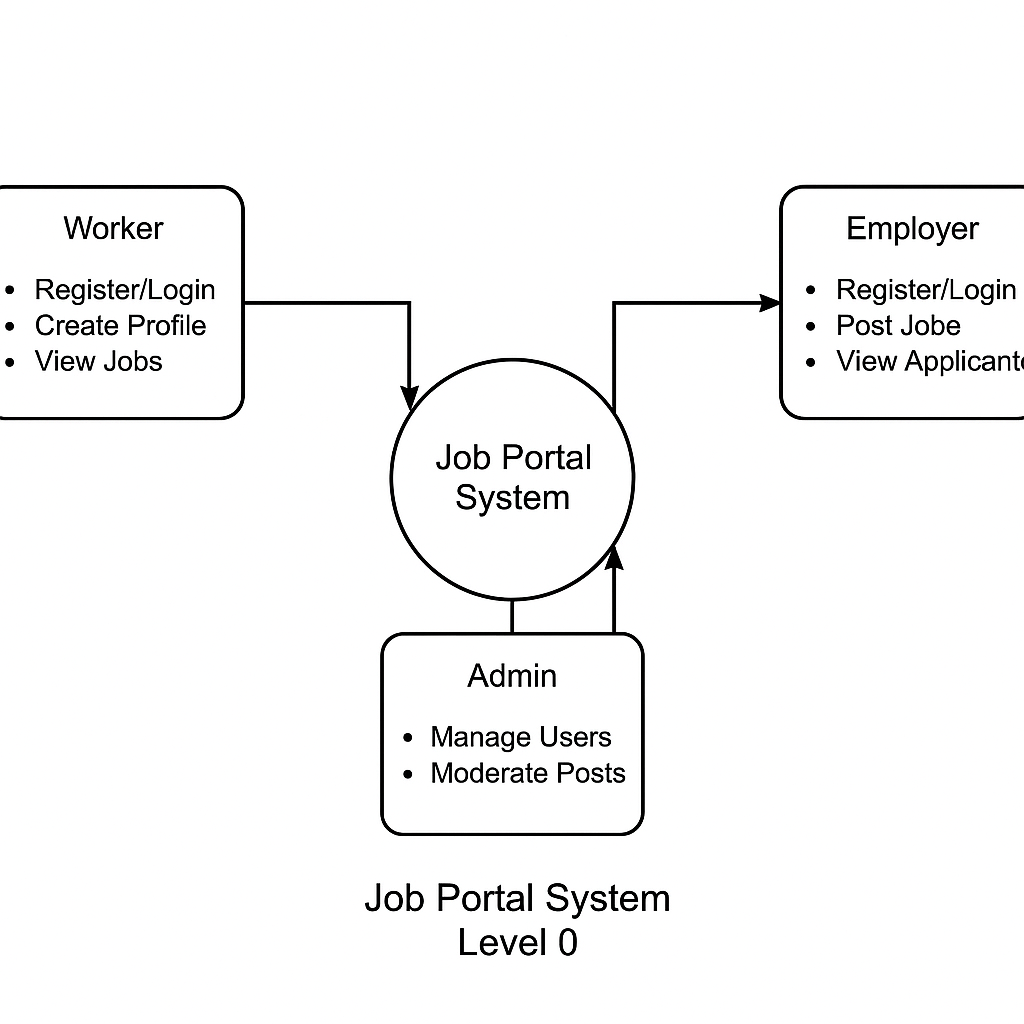
A **Data Flow Diagram (DFD)** shows **how data moves** through your system:

* What data comes in and goes out
* Where it's stored
* How users interact with the system

**✅ We’ll Create Two DFDs:**

**1. DFD Level 0 (Context Level)**

* Shows the system as a **single block**
* Highlights interaction with external entities



Worker: - Register/Login

- Create Profile

- View Jobs

- Apply to Jobs

Employer: - Register/Login

- Post Jobs

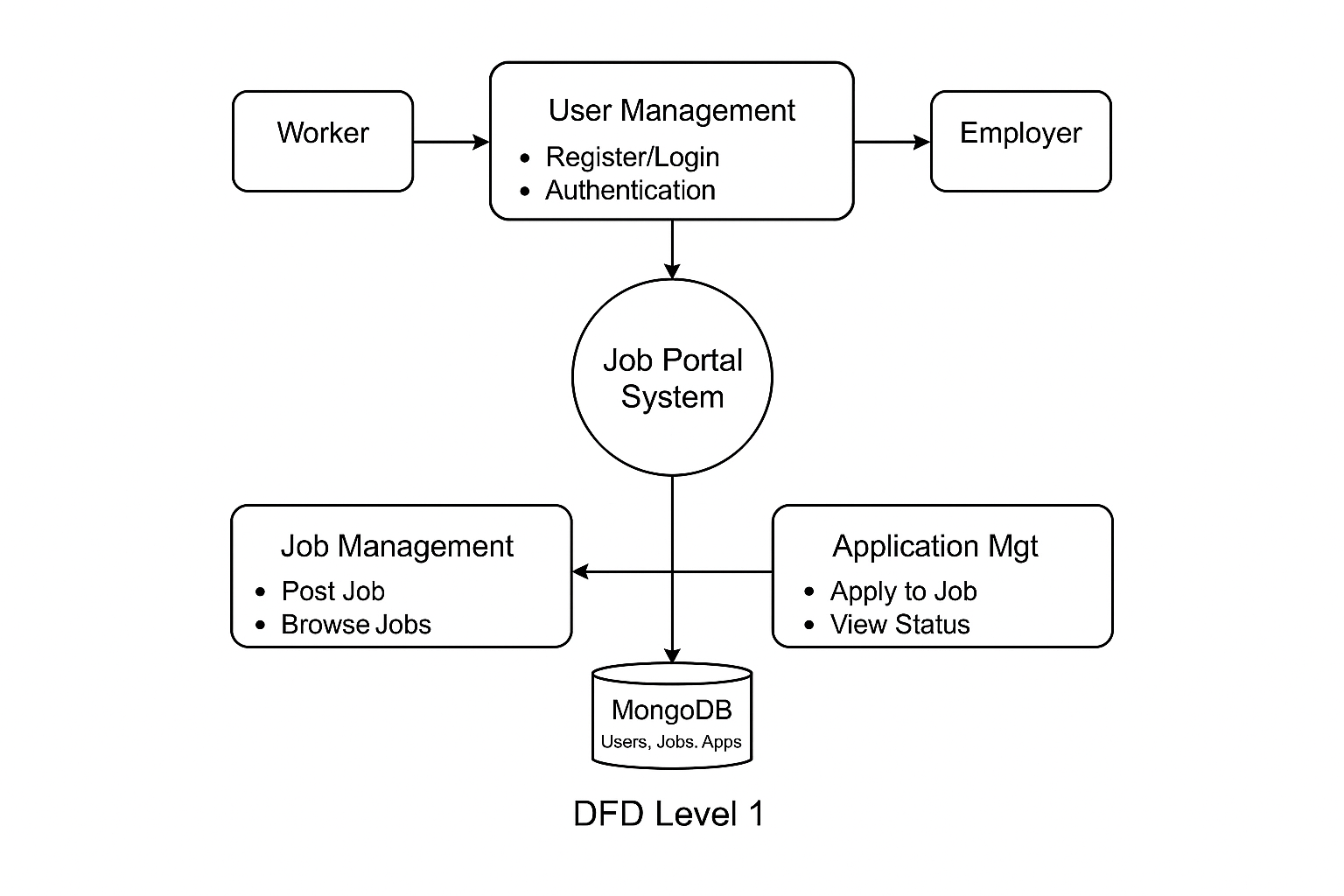
- View Applicants

Admin: - Manage Users

- Moderate Posts

**2. DFD Level 1**

* Breaks down the system into major **functional modules**
* Shows **data flow** between users, modules, and the database



**✅ DFD Level 1 – Job Portal System**

**🔹 External Entities:**

* **Worker**: Register, create profile, view/apply for jobs
* **Employer**: Register, post jobs, view applicants

**🔹 Main Modules:**

1. **User Management**
   * Handles login, registration, and authentication
   * Used by both workers and employers
2. **Job Management**
   * Allows employers to post and browse jobs
3. **Application Management**
   * Lets workers apply for jobs and track status

**🔹 Database (MongoDB):**

* Stores users, profiles, jobs, and applications
* All modules interact with MongoDB for data operations

**8. User Interface Wireframe Outline**

| **Page Name** | **Description** |
| --- | --- |
| Login / Signup | Authentication form |
| Worker Dashboard | Job listings, applied jobs, profile update |
| Employer Dashboard | Post new jobs, view applicants |
| Job Listing | List of jobs with filters by skill/location |
| Profile Page | Worker profile details |
| Admin Panel | Manage users and jobs |

**9. Security Measures**

* Passwords stored hashed (bcrypt).
* JWT tokens for stateless authentication.
* Role-based access control on API routes.
* Input validation on both frontend and backend.

**10. Future Enhancements**

* Multi-language support.
* Real-time chat between workers and employers.
* Rating & feedback system.
* Mobile app development.