Job Portal API - Full System Documentation

Table of Contents

---

1. Introduction

The Job Portal API is a fully functional backend system that facilitates job posting, job searching, and candidate application functionalities. It serves three primary user roles: Candidates, Hiring Managers, and Admins. This API is built with scalability, maintainability, and security in mind.

---

2. Technology Stack

- **Language:** TypeScript

- **Runtime:** Node.js
- **Framework:** Express.js
- **ORM:** Prisma
- **Database:** PostgreSQL
- **Validation:** Zod
- **Authentication:** JWT (JSON Web Tokens)
- **Testing:** Jest, Supertest
- **File Upload:** Multer
- **Documentation:** Swagger (OpenAPI)

---

3. System Overview

The system allows multiple types of users to interact with the platform:

- **Candidates** can register, search for jobs, and apply.
- **Hiring Managers** can create and manage job listings.
- **Admins** can manage all users and roles.

The system implements RESTful API design and supports validation, secure authentication, and scalable data handling.

---

4. User Roles and Access Control (Expanded)

User roles in the Job Portal API form the foundation of the platform's access and data control policies. These roles determine how users interact with system resources, enforce security boundaries, and shape the user experience across the portal.

The system uses a **Role-Based Access Control (RBAC)** approach to define and enforce what each user type can or cannot do within the application. There are three primary roles:

4.1 Candidate

Primary Role: Job seeker
Purpose: To discover and apply for relevant job opportunities.

Key Capabilities:

- **Job Browsing**: Access all available job listings.
- **Search & Filter**: Use parameters like job type, location, salary range, and deadline to refine job search.
- **Job Details**: View job descriptions, employer information, and application deadlines.

- **Application Submission**:
    - Apply to jobs if the deadline has not passed.
    - Upload a resume (PDF format only).
    - Cannot reapply if already applied.
- **Profile Management** (optional future enhancement): View and manage personal data.
- **Access Restrictions**:
    - Cannot post or edit jobs.
    - Cannot see other candidates' applications.
    - Cannot access administrative dashboards or user data.

System Safeguards:

- Candidates can only access their own applications.
- Resume uploads are scanned for valid file types and size limits.

4.2 Hiring Manager

Primary Role: Job provider
Purpose: To post, update, and manage job listings and view applications.

Key Capabilities:

- **Job Creation**:
    - Post new jobs with details like title, description, type, salary, location, and deadline.
- **Job Management**:
    - View all jobs they have created.
    - Update or delete their own job listings.
- **Application Management**:
    - View candidates who have applied to their posted jobs.
    - Access uploaded resumes and applicant details.
- **Data Scope**:
    - Cannot view or edit jobs posted by other managers.
    - Cannot access applications for jobs not owned by them.
    - Cannot change user roles or manage users.

System Safeguards:

- Authorization middleware ensures a manager can only manage their own job posts.
- Manager ID is validated against job ownership in the database for sensitive actions.

4.3 Admin

Primary Role: Platform overseer
Purpose: To maintain the system, manage users and roles, and have visibility into all operational data.

Key Capabilities:

- **User Management**:
    - View all registered users (candidates and managers).
    - Promote a user to Hiring Manager role or demote back to Candidate.
- **Platform Oversight**:
    - View all job listings, regardless of who created them.
    - Access all applications and resume files.
- **System Auditing**:
    - Monitor activity logs and manage reports.
    - View usage metrics and system health (via integrations).
- **Access Restrictions**:
    - Cannot apply to jobs or create listings (functionality limited by role but not technically blocked unless enforced).

System Safeguards:

- Admin-only routes require elevated role in JWT.
- Sensitive admin operations are protected with strict middleware and audit logging.

4.4 Middleware Enforcement & RBAC Logic

Access Control Strategy:
Access to all critical operations is enforced through middleware functions that validate user identity and role.

Steps for Authorization:

1. **Token Extraction**: Each protected route includes middleware that reads the JWT from the `Authorization` header.
2. **Token Verification**: JWT is decoded to verify user authenticity and extract user ID and role.
3. **Role Validation**: The role is compared against the allowed roles for the endpoint.
4. **Ownership Check** (if needed): For resources like job listings, the system checks whether the current user is the owner of the resource.

5. **Error Handling**:
    - If role is not authorized → HTTP `403 Forbidden`
    - If token is missing or invalid → HTTP `401 Unauthorized`

Middleware Components:

- `authenticateUser`: Verifies token and attaches user object to request.
- `authorizeRoles(...roles)`: Compares the user's role with the required roles and blocks if not matched.
- `validateOwnership`: (used in job and application routes) Confirms that the resource belongs to the user.

Benefits of Middleware-Based RBAC:

- Scalable: Easily extendable as new roles or permissions are added.
- Maintainable: Role logic is decoupled from business logic, allowing cleaner controllers.
- Secure: Prevents unauthorized access through centralized enforcement.

4.5 Role Transition and Hierarchy

The system supports role transitions (primarily managed by Admin):

From	To	Managed By	Use Case Example
Candidate	Hiring Manager	Admin	Promoting a qualified user to post job openings
Hiring Manager	Candidate	Admin	Demoting inactive or unqualified users
Any	Admin	-	Cannot be granted externally

4.7 Future Role-Based Enhancements

- **Sub-Roles**: Introduce levels within roles (e.g., `Senior Hiring Manager`, `Guest Candidate`) for more granular access.
- **Permissions Matrix**: A centralized admin interface to configure route access per role without modifying code.
- **Audit Logs**: Full tracking of role-related changes for regulatory compliance.

5. Feature Overview

5.1 Candidate Features (Expanded)

The **Candidate role** is the core end-user persona of the Job Portal API, representing individuals actively seeking job opportunities. The platform is designed to offer a seamless and accessible

experience for candidates to explore, filter, and apply for jobs efficiently. Below is a detailed explanation of the core features available to users with the Candidate role.

---

1. Register/Login

Purpose:
To authenticate candidates securely and uniquely identify them within the system.

Signup Process (`POST /user/signup`):

- Candidates provide:
    - Full name
    - Email address (must be unique)
    - Password (securely hashed using bcrypt)
- Input validation is handled using **Zod**, ensuring correct email format, strong password rules, and field completeness.
- On successful registration:
    - A default role of `Candidate` is assigned.
    - A JWT token is generated and returned.
    - User record is stored in PostgreSQL via Prisma ORM.

🔐 Login Process (`POST /user/login`):

- Candidates submit email and password.
- Credentials are validated against stored user data.
- On success:
    - A signed JWT token is returned.
    - This token is used for all subsequent protected routes (e.g., applying to jobs).

🔎 User Profile Retrieval (`GET /user/me`):

- Authenticated users can retrieve their profile using the token.
- No passwords or sensitive information are exposed.

---

2. View All Jobs (`GET /jobs`)

Purpose:
Enable candidates to explore job opportunities across industries, companies, and regions.

Features:

- Candidates can retrieve all active job postings.

- Each job entry includes:
    - Job title, company name, type (e.g., full-time, remote), salary range
    - Location, application deadline
    - Short description
    - Posted date and job ID

🔐 **Access:**

- This endpoint is **public**, meaning it doesn't require authentication to encourage open discovery.
- However, certain advanced data (like application status) is only shown to authenticated users.

3. Filter and Sort Jobs

Purpose:
To allow candidates to tailor the job list to their preferences and qualifications.

Filtering Parameters (as Query Params):

Parameter	Description	Example
`location`	Filter jobs by geographic location	`/jobs?location=Berlin`
`jobType`	Filter by job type (Full-time, Remote, etc)	`/jobs?jobType=Remote`
`minSalary`	Minimum salary threshold	`/jobs?minSalary=50000`
`maxSalary`	Maximum salary cap	`/jobs?maxSalary=100000`

🔄 Sorting Parameters:

Parameter	Description	Example
`sortBy`	Column to sort by	`sortBy=salary`
`order`	`asc` or `desc`	`order=desc`

📄 **Pagination (Bonus):**

- Query params:
    - `page`: Which page of results (e.g., `page=1`)
    - `limit`: Number of jobs per page (e.g., `limit=10`)
- Helps with performance and better UX for large datasets.

⚠️ **Input Validation:**

All filters and sorts are type-checked and validated using Zod. Invalid parameters return a `400 Bad Request`.

---

4. Apply to Jobs (`POST /jobs/:id/apply`)

Purpose:
Allow candidates to formally submit interest in job listings with optional resume uploads.

✅ **Application Rules:**

- Only authenticated candidates can apply.
- Duplicate applications to the same job are blocked.
- Applying after the job deadline is prohibited.

💼 **Resume Upload:**

- Accepts only `.pdf` files.
- Uses **Multer** middleware to handle multipart/form-data.
- Resume files are renamed uniquely and stored under `/uploads/resumes/`.
- File size and MIME type are strictly validated.

🎁 **Application Data Stored:**

- Candidate ID (linked to user)
- Job ID
- Resume path (if uploaded)
- Timestamp of application

Application Feedback:

- Returns a success message: `"Application submitted successfully."`
- If resume is included, the file path is also returned in response metadata.

🔄 Application Status (Optional Enhancement)

Future Feature:

- Allow candidates to view their application history:
    - Status: Pending, Reviewed, Rejected
    - Resume link
    - Applied date
- Route: `GET /user/applications`
- Helps candidates track which jobs they've applied to and avoid duplicate submissions.

🛡 Security and Integrity

- **Authorization**: Only logged-in candidates can apply to jobs.
- **Ownership Checks**: Applications are tied to a user ID to prevent impersonation.
- **Validation**: Every field (resume, job ID, deadlines) is validated before submission.
- **Error Handling**:
    - `403 Forbidden`: If applying to same job again.
    - `410 Gone`: If applying after the job deadline.
    - `422 Unprocessable Entity`: If resume file is invalid.

---

summary

Feature	Public Access	Auth Required	Input Validation	File Upload	Ownership Logic
Register/Login	✓	🔒 (Login)	✓ (Zod)	✗	✗
View Jobs	✓	✗	✓	✗	✗
Filter & Sort Jobs	✓	✗	✓	✗	✗
Apply to Jobs	✗	✓	✓	✓ (Multer)	✓

5.2 Hiring Manager Features (Expanded)

The **Hiring Manager role** is a key administrative persona in the Job Portal API ecosystem. Hiring Managers are responsible for creating and managing job listings, as well as reviewing candidates who apply for those jobs. The platform enforces strict role-based access so that Hiring Managers can only interact with resources they own.

---

1. Register/Login

Purpose:
Enable secure access to the job management system for employers or company representatives who wish to post jobs.

Signup Process (`POST /user/signup`):

- Hiring Managers can register using the same signup route as Candidates.
- By default, users are registered with the `Candidate` role. Only Admins can promote users to the `HiringManager` role.
- Required fields:

- Full name
- Email (must be unique)
- Password

🔐 Login Process (`POST /user/login`):

- Email and password are verified.
- A signed **JWT token** is returned upon successful authentication.
- The token includes the user's role (used to gate access to job management features).

🔄 User Role Promotion:

- Only an **Admin** can update a user's role to `HiringManager` via:
  - `PATCH /admin/users/:id/role`
- This prevents unauthorized users from self-assigning job posting privileges.

---

2. Post Jobs (`POST /jobs`)

Purpose:
Allow Hiring Managers to create job postings that are visible to Candidates.

Input Fields:

Field

Field	Description	Validation
title	Job title	Required, string
description	Full job description	Required, string
jobType	e.g., Full-time, Remote, Contract	Enum, required
location	City or Remote	Required, string
salary	Numeric salary value or range	Required, number
deadline	ISO date string	Required, future date
companyName	Name of the hiring company	Required

- All fields are validated using **Zod**.
- Only users with the `HiringManager` role can create jobs.

✅ Post-Creation Behavior:

- Job is associated with the hiring manager's `userId`.
- `createdAt` and `updatedAt` timestamps are automatically added.
- The job becomes immediately visible on the public job listings endpoint (`GET /jobs`).

---

3. View, Edit, and Delete Own Job Listings

Purpose:
Enable Hiring Managers to manage the jobs they've posted.

🕮 View All Jobs by Manager (`GET /manager/jobs`):

- Returns a list of jobs **created by the authenticated manager**.
- Requires JWT token with a `HiringManager` role.
- Response includes:
    - Job title, type, location, deadline, and number of applicants
    - Metadata like created date and status (active/expired)

🔍 View Job by ID (`GET /manager/jobs/:id`):

- Retrieves a single job **only if** it belongs to the requesting manager.
- Includes:
    - Full job details
    - List of applicants (if any)
- If the job belongs to another user:
    - Returns `403 Forbidden`

✏ Edit Job (`PATCH /jobs/:id`):

- Allows updating fields like title, salary, deadline, or description.
- Only allowed for the job's original creator (verified via `userId`).
- Zod schema enforces partial update validation.
- Updated timestamp is refreshed.

❌ Delete Job (Optional Feature):

- Deletion can be implemented via `DELETE /jobs/:id`
- Permanently removes job and associated applications (or flags as inactive).

---

4. View Applicants to Jobs

Purpose:
Allow Hiring Managers to access applications submitted to their job posts.

📖 **View Applications (`GET /manager/jobs/:id`):**

- Embedded in job details is a list of applicants, each containing:
    - Candidate name and email
    - Submitted resume file link (PDF)
    - Application submission date
- If no applications exist, returns an empty list.

🔐 **Access Control:**

- Middleware ensures that **only the job owner** can access the application list.
- Unauthorized access attempts return:
    - `403 Forbidden` if the job belongs to another user
    - `404 Not Found` if job doesn't exist

5.3 Admin Features (Expanded)

The **Admin** role is the highest-privilege user type within the Job Portal API system. Admins are responsible for overseeing the platform's health, managing user access and roles, and monitoring all data interactions. This role is critical for maintaining security, fairness, and integrity across the platform.

---

1. View All Users

Purpose:
To allow Admins to monitor, audit, and manage all users registered on the platform—regardless of role (Candidate, Hiring Manager, or Admin).

Endpoint: `GET /admin/users`

Functionality:

- Returns a complete list of users.
- Supports pagination and filtering for large datasets (optional feature).
- Each user object includes:
    - `id`, `name`, `email`
    - `role` (e.g., Candidate, HiringManager, Admin)
    - `createdAt`, `updatedAt` timestamps

Use Cases:

- View new registrations
- Detect unauthorized or suspicious accounts
- Prepare for role promotions or demotions

Security:

- Only accessible with a valid **Admin JWT token**
- Returns `403 Forbidden` for non-admin roles

---

2. Update User Roles

Purpose:
To enable Admins to control user access and functionality by promoting/demoting roles. This feature is especially important for transforming a regular Candidate into a Hiring Manager.

🔁 Endpoint: `PATCH /admin/users/:id/role`

Functionality:

- Allows an Admin to change the role of a specific user.
- Accepts a payload like:
- {
- "role": "HiringManager"
- }

Valid Roles:

- `"Candidate"`
- `"HiringManager"`
- `"Admin"` (assigning Admin role is usually restricted to super-admins)

Validation:

- Uses **Zod** to ensure the role is a valid enum.
- Admins cannot demote themselves (optional restriction).

Use Cases:

- Promote a Candidate to a Hiring Manager after verification
- Demote a Hiring Manager for policy violations
- Assign Admin privileges in exceptional cases

Security:

- Requires Admin-level JWT
- Returns:

- `403 Forbidden` if attempted by a non-admin
- `400 Bad Request` for invalid role updates
- `404 Not Found` if user ID doesn't exist

---

3. View All Jobs and Applications

Purpose:
To grant Admins full transparency and control over the platform's job ecosystem, including job posts and applications submitted by candidates.

📄 View All Jobs: `GET /admin/jobs`

Functionality:

- Retrieves all job postings on the platform.
- Returns:
    - Job title, salary, deadline
    - Associated Hiring Manager
    - Number of applicants

Use Cases:

- Monitor job activity across industries
- Detect inappropriate or expired postings
- Audit performance of hiring managers

📝 View All Applications: `GET /admin/applications` (optional route)

Functionality:

- Lists all applications submitted by candidates.
- Includes:
    - Job applied for
    - Candidate details (name, email)
    - Uploaded resume (PDF link)
    - Application status (optional future field)
    - Submission date

Use Cases:

- Detect spam or repeated applications

- Audit resume uploads for policy compliance
- Evaluate hiring trends across roles or job types

Security:

- Both endpoints require an **Admin token**
- Middleware ensures strict role-based access to sensitive data

---

🔐 Security & Access Rules for Admin Features

Feature	Route	Method	Access Level	Notes
View all users	`/admin/users`	GET	Admin Only	Cannot be accessed by other roles
Update user roles	`/admin/users/:id/role`	PATCH	Admin Only	Role must be a valid enum
View all jobs	`/admin/jobs`	GET	Admin Only	Lists every job across all hiring managers
View all applications	`/admin/applications`	GET	Admin Only	Lists all applications with candidate data

---

6. System Architecture

The system follows a modular MVC (Model-View-Controller) architecture:

- **Models** are defined with Prisma.
- **Controllers** manage the request-response cycle.
- **Routes** are grouped by role (candidate, manager, admin).
- **Services** encapsulate business logic.
- **Middlewares** handle authentication, validation, and role checking.

---

7. Database Design and Models

- User: Contains name, email, password, and role
- Job: Contains title, description, type, location, salary, deadline, and relation to hiring manager

- Application: Stores resume file path, applicant info, job info

Referential integrity and constraints are implemented using Prisma schema.

---

8. API Design

RESTful principles:

- **GET**: Retrieve resources
- **POST**: Create resources
- **PATCH**: Update resources
- **DELETE**: Delete resources (if applicable)

Endpoints are grouped under:

- `/user` for authentication
- `/jobs` for job listings
- `/manager/jobs` for manager-specific jobs
- `/admin` for administrative actions

---

9. Validation and Error Handling

All inputs are validated using Zod. This ensures that all API inputs meet required format, length, and type expectations. Errors are centralized and descriptive.

---

10. Authentication and Authorization

JWT is used for stateless user authentication. Each request is validated via middleware to ensure token authenticity. Role-based middleware restricts access to certain endpoints.

---

11. File Uploads (Resume)

Candidates can upload resumes in PDF format using Multer. Files are stored in the `/uploads` directory with unique filenames.

---

12. Filtering, Sorting, and Pagination

Candidates can use query parameters to:

- Filter jobs by location, type, and salary
- Sort by salary or deadline

- Paginate results for large datasets

---

13. Admin Features and Control

Admins have elevated privileges:

- View all candidates and hiring managers
- Promote or demote user roles
- View job applications across the platform

---

14. Optional Advanced Features

- **Top Paid Jobs**: Returns top 10 jobs with highest salary
- **Most Applied Jobs**: Returns top 5 jobs with most applications
- **Job Recommendations**: (Future) Suggest jobs based on candidate activity

---

15. Testing Strategy

Testing is implemented using Jest and Supertest:

- Unit tests for utility functions and controllers
- Integration tests for full API flow
- Mocking of Prisma client for isolation

---

16. Security Considerations

- JWT expiration and refresh strategies
- Password hashing using bcrypt
- Input sanitization to prevent XSS/SQL Injection
- File upload limits and MIME type checking

---

17. Logging and Monitoring

- Use `morgan` for HTTP logging
- `winston` for structured error logging
- Future integration with centralized logging services (e.g., LogRocket, Datadog)

---

18. Deployment Strategy

- Use Docker for containerization
- Deploy to platforms like Render, Heroku, or AWS EC2
- Use environment variables for secrets
- Enable HTTPS with SSL certs

---

19. CI/CD Integration

- GitHub Actions for automated tests
- Linting and formatting with ESLint and Prettier
- Build and deploy on push to main branch

---

20. Future Enhancements

- Real-time job alerts with WebSockets
- Role-based dashboard UI
- Admin analytics dashboard
- Resume parsing and keyword matching

---

21. Conclusion

This Job Portal API is a scalable and extensible backend system designed to support a full-featured job portal platform. With role-based access, solid architecture, strong validation, and advanced features, it offers a complete solution for job searching and hiring workflows.

This documentation serves as a standalone reference for any team working with or extending the API, and can be converted to PDF for sharing.

🔐 Authentication Routes (`/user`)

Method	Endpoint	Description
POST	`/user/signup`	Register a new user (Candidate or Manager)
POST	`/user/login`	Login and receive a JWT token
GET	`/user/me`	Get logged-in user's profile from token

Candidate Routes (`/jobs`, `/jobs/:id`)

Method	Endpoint	Description
GET	`/jobs`	View all available jobs
GET	`/jobs/:id`	View a single job with full details and manager info
POST	`/jobs/:id/apply`	Apply to a job (upload resume, must be before deadline)

Method	Endpoint	Description

Method	Endpoint	Description
POST	`/jobs`	Create a new job posting
GET	`/manager/jobs`	View all jobs posted by the logged-in hiring manager
GET	`/manager/jobs/:id`	View job details (includes applicants and resumes)
PATCH	`/jobs/:id`	Update a job listing (only if it belongs to the manager)
DELETE	`/jobs/:id` (optional)	Delete a job listing

Method	Endpoint	Description
GET	`/admin/users`	View all registered users (candidates and managers)
PATCH	`/admin/users/:id/role`	Update a user's role (e.g., promote to Hiring Manager)
GET	`/admin/jobs`	View all job postings across the platform
GET	`/admin/applications` (optional)	View all job applications submitted by all candidates
GET	`/admin/candidates` (optional)	View all candidate profiles
GET	`/admin/candidates/:id` (optional)	View a specific candidate's applied jobs
GET	`/admin/managers` (optional)	View all hiring managers and their job stats

Method	Endpoint	Description
GET	`/jobs/top-paid`	Get top 10 highest paying jobs
GET	`/jobs/most-applied`	Get top 5 jobs with most candidate applications
GET	`/jobs/recommendations`	(Future) Job recommendations based on user activity