Jovio

Backend System Requirements Specification

This document is a personal planning blueprint for the Jovio project. It is created before development to organize my ideas and clearly define what I intend to build. The goal is to have a clear roadmap that guides the development process and avoids randomness.

The project is inspired by the final exam of the Node.js course I completed at **Route Training Center,** with additional features and enhancements beyond the original scope. Jovio is a training project aimed at strengthening my backend skills, with a primary focus on NestJS and building a well-structured, scalable system.

Created by: **Dev Mahmoud Zain**
Backend Developer

Feel free to reach out

Email: dev.mahmoud.zain@gmail.com

Whatsapp: 01551975456

1-Collections

- User Collection
- OTP Collection
- JWT Collection
- Company Collection
- Job opportunity Collection
- Application Collection
- Chat Collection
- Notification Collection

1.1-User Collection

Field Name	Type	Description
firstName	string	User's first name
lastName	string	User's last name
email	string	User's email, used for login
emailConfirmedAt	Date	Timestamp when user confirms email
newEmail	string	Temporary email if user wants to change; replaces email after confirmation
provider	string	Authentication provider (system, google)
role	string	User role (user, admin, super-admin)
status	string	Current user status (Looking For Internship, Looking For Job, Employed, Student, Freelancer, Open To Opportunities, Not Looking Right Now)
password	string	Hashed password
gender	string	User gender
phone	string	User phone number
ChangeCredentialsTime	Date	Last time user updated credentials
profilePicture	object	Main profile picture { url: String, public_id: String }
coverPicture	object	Main cover picture { url: String, public_id: String }
gallery	object[]	User Images
dateOfBirth	Date	User's birth date
bannedAt	Date	Timestamp when user was banned
bannedUntil	Date	Timestamp until which user is banned
freezedAt	Date	Timestamp when user account was frozen
freezedBy	ObjectId	Admin who froze the account
restoredAt	Date	Timestamp when account was restored
restoredBy	ObjectId	Admin who restored the account

Notes :

- Create an index on userId + isUsed + expiresAt to speed up lookup and verification.

- Do not delete the OTP immediately after it expires; it may be needed for security analysis

1.2-OTP Collection

Field Name	Type	Description
userId	ObjectId	Reference to the user this OTP belongs to
otp	string	One-time password code
type	string	OTP type (e.g., "email_verification", "password_reset")
expiresAt	Date	Expiration timestamp of the OTP
createdAt	Date	Timestamp when the OTP was created
isUsed	boolean	Whether the OTP has been used or not
usedAt	Date	Timestamp when the OTP was used (if applicable)
attempts	number	Number of attempts tried with this OTP (optional, for security)
blockedUntil	Date	If attempts exceed 5, user is blocked until this timestamp (e.g., 10 minutes from last attempt)

Notes :

- Each failed attempt increments the attempts count and checks if it has reached 5.
- If attempts reach 5, set blockedUntil = now + 10 minutes.
- Any attempt made before blockedUntil is automatically rejected.

1.3-JWT Collection

Field Name	Type	Description
userId	ObjectId	Reference to the user this token belongs to
token	string	The JWT string itself
type	string	Token type (e.g., "access", "refresh")
createdAt	Date	Timestamp when the token was issued
expiresAt	Date	Expiration timestamp for the token
revoked	boolean	Whether the token has been revoked before expiration
revokedAt	Date	Timestamp when the token was revoked (if applicable)
deviceInfo	object	Normalized device information (device type, OS, browser)
ipAddress	string	IP address from which the token was issued
userAgent	string	Raw user agent string for session tracking

DeviceInfo Structure:

```
{
  "type": "desktop | mobile | tablet",
  "os": "Linux | Windows | macOS | Android | iOS",
  "browser": "Chrome | Firefox | Safari | Chromium-based"
}
```

Notes :

- All issued access and refresh tokens are stored in the database.
- Each token represents a single authenticated session.
- Token validation requires:
    - Valid signature
    - Not expired
    - Not revoked
    - Existing record in the database
- Tokens are revoked instead of being deleted immediately to allow auditing and security analysis.
- Expired tokens may be cleaned up periodically using a background job.

1.4- Company Collection

Field Name	Type	Description
name	string	Company official name
slug	string	URL-friendly unique identifier
email	string	Official company email
phone	string	Company contact phone number
website	string	Company official website
description	string	Company overview and description
industry	string	Company industry or field
size	string	Company size (e.g. 1-10, 11-50, 51-200, 200+)
foundedAt	Date	Company founding date
location	object	Company main location { country, city ,street }
isVerified	boolean	Indicates whether the company is verified
verificationStatus	string	pending | approved | rejected
verifiedAt	Date	Timestamp when company was verified
createdAt	Date	Timestamp when company was created
updatedAt	Date	Timestamp when company was last updated
ownerId	ObjectId	User who owns the company
admins	ObjectId[]	Users who can manage the company
createdBy	ObjectId	User who created the company
logo	object	Company logo `{ url:string, public_id:string }`
CoverImage	object	Company cover image `{ url:string, public_id:string }`
Gallery	object[]	company images
socialLinks	object[]	[{name:string , url:string}]
status	string	active | suspended | banned
suspendedAt	Date	Timestamp when company was suspended
suspendedBy	ObjectId	Admin who suspended the company
bannedAt	Date	Timestamp when company was banned
bannedBy	ObjectId	Admin who banned the company
restoredAt	Date	Timestamp when company was restored
restoredBy	ObjectId	Admin who restored the company

Notes :

- Only verified companies can post job opportunities.
- Company status affects visibility and job posting permissions.
- Soft actions (suspend / ban) are preferred over deletion

1.5 – Job Opportunity Collection

Field Name	Type	Description
title	string	Job title
slug	string	URL-friendly unique identifier
description	string	Full job description
requirements	String[]	Required skills and qualifications
responsibilities	string[]	Job responsibilities
employmentType	string	full-time / part-time / internship / freelance / contract
workType	string	on-site / remote / hybrid
experienceLevel	string	junior / mid-level / senior / lead
salary	object	Salary details { min:number, max:number, currency:string, isHidden:boolean }
vacancies	number	Number of open positions
location	object	Job location { country, city ,street }
industry	string	Job industry / field
tags	string[]	Keywords for search and filtering
status	string	draft / published / closed / paused
isUrgent	boolean	Marks urgent hiring
isFeatured	boolean	Highlighted job
publishedAt	Date	Timestamp when job was published
expiresAt	Date	Job expiration date
viewsCount	number	Number of job views
applicationsCount	number	Number of applications
companyId	ObjectId	Reference to the company
createdBy	ObjectId	User (company admin) who created the job
updatedBy	ObjectId	User who last updated the job
createdAt	Date	Creation timestamp
updatedAt	Date	Last update timestamp

Notes :

- Jobs with `status = draft` are visible only to company admins.
- Jobs with `status = published` are public.
- `expiresAt` automatically closes the job when reached
-Closed / expired jobs cannot receive applications.
-`applicationsCount` and `viewsCount` are incremented automatically (no manual edits).
- Salary visibility controlled via `salary.isHidden`.
-`Indexes recommended:`
  - `companyId + status`
  - `tags`
  - `employmentType`
  - `workType`
  - `experienceLevel`

1.6 – Application Collection

Field Name	Type	Description
userId	ObjectId	Reference to the user applying
jobId	ObjectId	Reference to the job opportunity
companyId	ObjectId	Reference to the company (redundant for quick queries)
status	string	pending / reviewed / accepted / rejected / withdrawn
resume	object	User’s uploaded CV { url: string, public_id: string }
coverLetter	string	Optional cover letter text
appliedAt	Date	Timestamp when the application was submitted
reviewedAt	Date	Timestamp when the application was first reviewed
reviewedBy	ObjectId	Company admin who reviewed the application
notes	string	Internal notes by company admin
isShortlisted	boolean	Marks if applicant is shortlisted
interviewDate	Date	Scheduled interview date (optional)
rejectedAt	Date	Timestamp if application rejected
withdrawnAt	Date	Timestamp if user withdraws application
createdAt	Date	Creation timestamp
updatedAt	Date	Last update timestamp

Notes :

- Users can apply only once per job.
- Only active / published jobs accept applications.
- Company admins can change status and add notes.
- status flow: pending → reviewed → accepted/rejected | pending → withdrawn
- Users can withdraw before job closes.
- Shortlisting and interviews are optional but tracked.

- Notifications can be triggered on:

  - Application submission
  - Status change
  - Interview scheduling

1.7 – Notification Collection

Field Name	Type	Description
userId	ObjectId	Reference to the user receiving the notification
type	string	Notification type (e.g. application_submitted, application_status_changed, job_posted, interview_scheduled, system)
title	string	Short notification title
message	string	Notification message content
data	object	Optional payload (jobId, applicationId, companyId, etc.)
isRead	boolean	Indicates whether the notification was read
readAt	Date	Timestamp when notification was read
isArchived	boolean	Indicates if notification was archived
priority	string	low / normal / high
channel	string	in-app / email / push (future-proof)
createdAt	Date	Creation timestamp
expiresAt	Date	Optional expiration date for notification

Notes :

- Users can mark notifications as read or archived.
- Archived notifications are hidden from main inbox but not deleted.
- High priority notifications can trigger additional channels (email / push).
- Expired notifications are ignored by default.

Common Notification Types :

- `job_posted`
- `application_submitted`
- `application_reviewed`
- `application_accepted`
- `application_rejected`
- `interview_scheduled`
- `company_verified`
- `system`

1.8 – Chat Collection

Chat functionality is scoped to job applications and is only available between a candidate and a company after an application reaches a review or shortlist state.

Field Name	Type	Description
applicationId	ObjectId	Reference to the related job application
jobId	ObjectId	Reference to the job opportunity
companyId	ObjectId	Reference to the company
candidateId	ObjectId	Reference to the applicant user
participants	ObjectId[]	Chat participants (candidate + company admins)
isActive	boolean	Indicates whether chat is active
closedAt	Date	Timestamp when chat was closed
closedBy	ObjectId	User who closed the chat
lastMessageAt	Date	Timestamp of the last message
createdAt	Date	Creation timestamp
updatedAt	Date	Last update timestamp

Message Collection

Field Name	Type	Description
chatId	ObjectId	Reference to the chat
senderId	ObjectId	User who sent the message
senderRole	string	candidate / company
content	string	Message content
attachments	object[]	Optional files { url, public_id, type }
isRead	boolean	Indicates whether message was read
readAt	Date	Timestamp when message was read
createdAt	Date	Message creation timestamp

Notes :

- Chat is created only after application status becomes reviewed or shortlisted.
- Only participants listed in participants can send messages.
- Chat is automatically closed when:Application is rejected Or Job is closed or expired .
- Closed chats are read-only.
- One chat per application .
- Notifications should be triggered on:
    - New message
    - Chat closed
    -
- Chat Lifecycle : Application reviewed → Chat opened → Messaging → Chat closed

2-Authorization & Access Control

**This phase describes the authorization strategy used in the system.
It does not represent database collections, but rather defines how roles and permissions are evaluated during runtime to control access to system resources.**

2.1 System Roles

System roles define the global access level of a user across the platform.

Role	Description
user	Regular user (job seeker)
admin	Platform administrator
super_admin	Full system access and control

2.2 – Company Roles (Scoped Roles)

Role	Description
company_owner	Creator of the company with full control
company_admin	Manages jobs, applications, and company data

2.3 – Core Permissions

Permissions define specific actions that can be performed on system resources.

Permission	Description
create_company	Create a company profile
update_company	Update company information
manage_company_admins	Add or remove company admins
create_job	Create job opportunity
update_job	Update job details
publish_job	Publish job
pause_job	Pause job
close_job	Close job
review_application	Review job applications
update_application_status	Accept or reject applications
access_chat	Access application-related chat
send_message	Send chat messages
verify_company	Verify company profile
suspend_company	Suspend company
ban_company	Ban company

2.4 – Permission Evaluation Flow

Access to any protected resource is granted only if all the following conditions are met:

- **The user is authenticated.**
- **The user has the required system role or company-scoped role.**
- **The user is associated with the target resource (ownership or membership).**
- **The target entity is in a valid state.**

Example – Create Job Opportunity

- POST /jobs

Conditions:

User role: `company_owner` or `company_admin`

- **Company status: `active`**
- **Company verification: `approved`**

2.5 – Entity State Constraints

Authorization decisions may depend on the current state of the entity.

Entity	State Constraint
Company	Must be verified and active
Job	Must be published to accept applications
Application	Must not be rejected or withdrawn
Chat	Must be active to allow messaging

2.6 – Guard Strategy

Authorization logic is implemented through layered checks:

- **Authentication validation – ensures the user is logged in and token is valid.**
- **Role-based checks – verifies the user’s system or company role.**
- **Ownership and association checks – ensures the user is associated with the target resource.**
- **Entity state validation – confirms the entity is in a valid state to perform the action.**

Each request must pass all required checks before reaching business logic.

2.7 – Design Principles

Authorization logic must be centralized and reusable.

- **Permissions should not be hardcoded inside controllers.**
- **Roles define who the user is.**
- **Permissions define what the user can do.**
- **Access decisions are evaluated at runtime, based on role, ownership, and entity state.**
- **Authorization failures must return meaningful errors (e.g., 403 Forbidden).**

3-API Endpoints Map

This phase provides a complete mapping of all API endpoints required for the Jovio system.
It serves as a reference for backend development, describing each route with its method, authentication requirement, permissions, request data, and purpose.

The goal is to have a clear and organized blueprint, using a module-based structure, so development is consistent, predictable, and aligned with the data model and business rules defined in Phases 1 and

Modules

- Auth Module
  Handles all authentication and authorization flows.

- User Module
  Handles user profile management, viewing and updating personal information, and retrieving user data (no authentication logic here).

- Company Module
  Handles company profiles, verification, admins, ownership, suspension, banning, and CRUD operations on companies.

- Job Opportunity Module
  Manages job postings by companies, including job details, status, requirements, and category.

- Application Module
  Handles user applications to jobs, tracking status, timestamps, and links to the relevant user and job.

- Chat Module
  Manages chat sessions between users and companies related to job applications, including participants and messages.

- Notification Module
  Sends and stores notifications for users and companies regarding system events, job applications, or messages.

3.1 – Auth Module

1. Register – System Provider

- **Method / URL:** POST `/auth/register`
- **Auth:** ❌
- **Permission:** ❌
- **Description:**
  Creates a new user account via the system provider.
  Requires `{ firstName, lastName, email, password, gender, phone, dateOfBirth }`.
  After registration, the user must **activate the account** by confirming their email address using an OTP sent to their email.

2. Confirm Email – System Provider

- **Method / URL:** POST `/auth/confirm-email`
- **Auth:** ❌
- **Permission:** ❌
- **Description:**
  Confirms the user's email address after registration using the OTP sent to their email.
  Activates the account so the user can log in.

3. Register / Login – Google Provider

- **Method / URL:** POST `/auth/google`
- **Auth:** ❌
- **Permission:** ❌
- **Description:**
  Registers or logs in a user via Google OAuth.
  Automatically confirms the email from the Google account and creates the user if not existing.
  Returns `{ accessToken, refreshToken, user }`.

4. Login – System Provider

- **Method / URL:** POST `/auth/login`
- **Auth:** ❌
- **Permission:** ❌
- **Description:**
  Authenticates user credentials (email + password) and returns **access** and **refresh JWT tokens** with user info.
  Sets authentication cookies and records device info, IP, and user agent.

5. Logout

- **Method / URL: POST /auth/logout**
- **Auth: ✅**
- **Permission: user**
- **Description:**
  Logs out the authenticated user by revoking the current JWT token or clearing session cookies.

6. Refresh Token

- **Method / URL:** POST `/auth/refresh`
- **Auth:** ✅
- **Permission:** user
- **Description:**
  Generates a new **access token** using a valid refresh token to keep the session alive without re-login.

7. Request Password Reset

- **Method / URL:** POST `/auth/request-password-reset`
- **Auth:** ❌
- **Permission:** ❌
- **Description:**
  Initiates the password reset process by sending an **OTP** to the user's registered email.

8. Reset Password

- **Method / URL:** POST `/auth/reset-password`
- **Auth:** ❌
- **Permission:** ❌
- **Description:**
  Resets the user's password after validating the OTP sent to their email.
  Requires `{ token, newPassword }`.

9. Request Email Change

- **Method / URL:** POST `/auth/request-email-change`
- **Auth:** ✅
- **Permission:** user
- **Description:**
  Starts the email change process by sending an OTP to the **new email address** provided.

10. Confirm Email Change

- **Method / URL:** POST `/auth/confirm-email-change`
- **Auth:** ✅
- **Permission:** user
- **Description:**
  Confirms the new email address using the OTP sent during the request. Updates the user email.

11. Get User Sessions

- **Method / URL:** GET `/auth/sessions`
- **Auth:** ✅
- **Permission:** user
- **Description:**
  Retrieves all active JWT sessions for the authenticated user, including **device info, IP, and user agent**.

12. Revoke Session

- **Method / URL:** POST `/auth/sessions/revoke`
- **Auth:** ✅
- **Permission:** user
- **Description:**
  Revokes a specific session token or all active sessions.
  Requires `{ token? }` (optional, if omitted revokes all).

3.2 – User Module

1. Get User Profile

- **Method / URL: GET `/users/profile/:id`**
- **Auth: ✅**
- **Permission: user / admin (if viewing others)**
- **Description: Retrieves the profile information of a user by ID.**
- **Response: User object with public and non-sensitive fields.**

2. Update Own Profile

- **Method / URL:** PATCH `/users/profile`
- **Auth:** ✅
- **Permission:** user
- **Description:** Updates the authenticated user's profile partially.
  Fields include: `{ firstName?, lastName?, phone?, profilePicture?, coverPicture?, gender?, dateOfBirth? }`

3. List Users

- **Method / URL:** GET `/users`
- **Auth:** ✅
- **Permission:** admin / super_admin
- **Description:** Returns a paginated list of users with optional filters (email, name, status, role).

4. Get Own Profile

- **Method / URL:** GET `/users/me`
- **Auth:** ✅
- **Permission:** user
- **Description:** Retrieves the profile of the currently authenticated user.

5. Upload Profile Picture

- **Method / URL:** POST `/users/profile/picture`
- **Auth:** ✅
- **Permission:** user
- **Description:** Uploads or updates the user’s profile picture. Accepts an image file.

6. Upload Cover Picture

- **Method / URL:** POST `/users/profile/cover`
- **Auth:** ✅
- **Permission:** user
- **Description:** Uploads or updates the user’s cover picture. Accepts an image file.

7. Freeze Account

- **Method / URL:** DELETE `/users/me`
- **Auth:** ✅
- **Permission:** user
- **Description:** Soft-deletes the authenticated user account (marks as deleted, does not remove from DB).

8. Admin – Update User Role

- **Method / URL:** PATCH `/users/:id`
- **Auth:** ✅
- **Permission:** admin / super_admin
- **Description:** Allows admin to update any user’s role.

9. Admin – Freeze Any User

- **Method / URL:** DELETE `/users/:id`
- **Auth:** ✅
- **Permission:** admin / super_admin
- **Description:** Allows admin to soft-delete any user account.

3.3 – Company Module

1. Create Company

- **Method / URL:** POST `/companies`
- **Auth:** ✅
- **Permission:** user
- **Description:**
  Creates a new company profile. Requires `{ name, email, phone, website, description, industry, size, foundedAt, location, logo?, coverImage?, socialLinks? }`.
  Sets `ownerId` to the authenticated user and `status` to pending verification.

2. Get Company Profile

- **Method / URL:** GET `/companies/:id`
- **Auth:** ❌
- **Permission:** public / user / admin
- **Description:**
  Retrieves company profile information by ID.
  Only verified companies may have full details exposed publicly.

3. Update Company

- **Method / URL:** PATCH `/companies/:id`
- **Auth:** ✅
- **Permission:** owner / admin
- **Description:**
  Updates company details partially.
  Fields include `{ name?, email?, phone?, website?, description?, industry?, size?, foundedAt?, location?, logo?, coverImage?, socialLinks? }`.

4. Delete Company

- **Method / URL:** DELETE `/companies/:id`
- **Auth:** ✅
- **Permission:** owner / admin
- **Description:**
  Soft-deletes the company (marks as inactive / deleted, does not remove from DB).

5. List Companies

- **Method / URL:** GET `/companies`
- **Auth:** ❌
- **Permission:** public / user / admin
- **Description:**
  Returns a paginated list of companies with optional filters (`industry`, `size`, `location`, `status`).

6. Search Companies

- **Method / URL:** GET `/companies/search`
- **Auth:** ❌
- **Permission:** public / user
- **Description:**
  Searches companies by name, industry, location, or other criteria. Supports query parameters like `?name=`, `?industry=`, `?location=`.

7. Search Companies by Job Availability

- **Method / URL:** GET `/companies/search/jobs`
- **Auth:** ❌
- **Permission:** public / user
- **Description:**
  Searches companies that currently have active job opportunities. Supports filters like `?jobTitle=`, `?category=`, `?location=`.

8. Verify Company

- **Method / URL:** POST `/companies/:id/verify`
- **Auth:** ✅
- **Permission:** admin / super_admin
- **Description:**
  Approves or rejects a company verification request. Sets `verificationStatus` and `verifiedAt`.

9. Suspend Company

- **Method / URL:** POST `/companies/:id/suspend`
- **Auth:** ✅
- **Permission:** admin / super_admin
- **Description:**
  Suspends a company account, sets `suspendedAt` and `suspendedBy`. Suspended companies cannot post jobs.

10. Restore Company

- **Method / URL:** POST `/companies/:id/restore`
- **Auth:** ✅
- **Permission:** admin / super_admin
- **Description:**
  Restores a previously suspended or soft-deleted company. Sets `restoredAt` and `restoredBy`.

11. Ban Company

- **Method / URL:** POST `/companies/:id/ban`
- **Auth:** ✅
- **Permission:** super_admin
- **Description:**
  Permanently bans a company account. Sets `bannedAt` and `bannedBy`.

3.4 – Job Opportunity Module

1. Create Job Opportunity

- **Method / URL:** POST `/jobs`
- **Auth:** ✅
- **Permission:** company owner / company admin
- **Description:**
  Creates a new job posting for a company.
  Requires fields like `{ title, description, requirements, location, salary, category, type, status, deadline }`.
  Associates the job with the company ID of the authenticated user.

2. Get Job Opportunity

- **Method / URL:** GET `/jobs/:id`
- **Auth:** ❌
- **Permission:** public / user / admin
- **Description:**
  Retrieves details of a specific job opportunity by ID.

3. Update Job Opportunity

- **Method / URL:** PATCH `/jobs/:id`
- **Auth:** ✅
- **Permission:** company owner / company admin
- **Description:**
  Updates details of a specific job posting partially. Fields include `{ title?, description?, requirements?, location?, salary?, category?, type?, status?, deadline? }`.

4. Delete Job Opportunity

- **Method / URL:** DELETE `/jobs/:id`
- **Auth:** ✅
- **Permission:** company owner / company admin
- **Description:**
  Soft-deletes a job posting. Marks it as inactive instead of removing from DB.

5. List Job Opportunities

- **Method / URL:** GET `/jobs`
- **Auth:** ❌
- **Permission:** public / user / admin
- **Description:**
  Returns a paginated list of job opportunities with optional filters like `{ companyId, category, type, location, status }`.

6. Search Job Opportunities

- **Method / URL:** GET `/jobs/search`
- **Auth:** ❌
- **Permission:** public / user
- **Description:**
  Searches job opportunities by title, category, location, company, or other criteria.
  Supports query parameters like `?title=`, `?category=`, `?location=`, `?companyId=`.

7. Apply to Job

- **Method / URL:** POST `/jobs/:id/apply`
- **Auth:** ✅
- **Permission:** user
- **Description:**
  Allows a user to apply for a specific job opportunity.
  Includes optional fields like `{ coverLetter?, resume? }`.

8. Get Applicants for Job

- **Method / URL:** GET `/jobs/:id/applicants`
- **Auth:** ✅
- **Permission:** company owner / company admin
- **Description:**
  Lists all users who have applied to a specific job opportunity.

9. Update Job Status

- **Method / URL:** PATCH `/jobs/:id/status`
- **Auth:** ✅
- **Permission:** company owner / company admin
- **Description:**
  Updates the status of a job posting (e.g., active, closed, paused).

3.5 – Application Module

1. Apply to Job

- **Method / URL:** POST `/applications`
- **Auth:** ✅
- **Permission:** user
- **Description:**
  Allows a user to apply for a specific job opportunity.
  Requires `{ jobId, coverLetter?, resume? }`.
  Creates a new application associated with the user and the job.

2. Get Application

- **Method / URL:** GET `/applications/:id`
- **Auth:** ✅
- **Permission:** user (owner) / company owner / company admin / admin
- **Description:**
  Retrieves details of a specific application by ID.
  Includes status, timestamps, and related job and user info.

3. List User Applications

- **Method / URL:** GET `/applications/user/:userId`
- **Auth:** ✅
- **Permission:** user (self) / admin
- **Description:**
  Returns all applications submitted by a specific user. Supports pagination and filtering by status.

4. List Job Applications

- **Method / URL:** GET `/applications/job/:jobId`
- **Auth:** ✅
- **Permission:** company owner / company admin / admin
- **Description:**
  Returns all applications for a specific job opportunity. Supports pagination and filtering by status.

5. Update Application Status

- **Method / URL:** PATCH `/applications/:id/status`
- **Auth:** ✅
- **Permission:** company owner / company admin
- **Description:**
  Updates the status of an application (e.g., pending, accepted, rejected, shortlisted).

6. Delete Application

- **Method / URL:** DELETE `/applications/:id`
- **Auth:** ✅
- **Permission:** user (owner) / admin
- **Description:**
  Allows the applicant or admin to withdraw or remove an application. Soft-delete recommended.

7. Search Applications

- **Method / URL:** GET `/applications/search`
- **Auth:** ✅
- **Permission:** company owner / company admin / admin
- **Description:**
  Searches applications by user name, job title, status, or date. Supports query parameters for filtering and pagination.

3.6 – Chat Module

1. Create Chat

- **Method / URL:** POST `/chats`
- **Auth:** ✅
- **Permission:** user / company owner / company admin
- **Description:**
  Creates a new chat session between participants.
  Requires `{ participants: [userId, companyId] }`.
  Can be linked to a specific application or job if needed.

2. Get Chat

- **Method / URL:** GET `/chats/:id`
- **Auth:** ✅
- **Permission:** participants / admin
- **Description:**
  Retrieves a specific chat session and its metadata.

3. List User Chats

- **Method / URL:** GET `/chats/user/:userId`
- **Auth:** ✅
- **Permission:** user (self) / admin
- **Description:**
  Lists all chat sessions a user is participating in. Supports pagination and filtering.

4. List Company Chats

- **Method / URL:** GET `/chats/company/:companyId`
- **Auth:** ✅
- **Permission:** company owner / company admin / admin
- **Description:**
  Lists all chat sessions a company is participating in. Supports pagination and filtering.

5. Send Message

- **Method / URL:** POST `/chats/:id/messages`
- **Auth:** ✅
- **Permission:** participants
- **Description:**
  Sends a message in a chat session.
  Requires `{ content, type? }` (type can be text, image, file, etc.).

6. Get Messages

- **Method / URL:** GET `/chats/:id/messages`
- **Auth:** ✅
- **Permission:** participants / admin
- **Description:**
  Retrieves messages from a specific chat session. Supports pagination.

7. Delete Message

- **Method / URL:** DELETE `/chats/:id/messages/:messageId`
- **Auth:** ✅
- **Permission:** sender / admin
- **Description:**
  Deletes a message from a chat session (soft-delete preferred).

8. Search Chats

- **Method / URL:** GET `/chats/search`
- **Auth:** ✅
- **Permission:** participants / admin
- **Description:**
  Searches chats or messages by content, participants, or related application/job. Supports filtering and pagination.

3.7 – Notification Module

1. List User Notifications

- **Method / URL:** GET `/notifications/user/:userId`
- **Auth:** ✅
- **Permission:** user (self) / admin
- **Description:**
  Lists all notifications for a specific user. Supports pagination, filtering by read/unread status, and sorting.

2. List Company Notifications

- **Method / URL:** GET `/notifications/company/:companyId`
- **Auth:** ✅
- **Permission:** company owner / company admin / admin
- **Description:**
  Lists all notifications for a specific company. Supports pagination and filtering.

3. Mark Notification as Read

- **Method / URL:** PATCH `/notifications/:id/read`
- **Auth:** ✅
- **Permission:** recipient
- **Description:**
  Marks a specific notification as read.

4. Delete Notification

- **Method / URL:** DELETE `/notifications/:id`
- **Auth:** ✅
- **Permission:** recipient / admin
- **Description:**
  Deletes a notification (soft-delete preferred).