Candidate APIs

HTTP	URI	Method (Controller)	Logic	DB Operation (Tables)	Transaction Scope
GET	/me/profile	getProfile()	Fetch candidate profile by user id	SELECT from candidateprofiles JOIN users	Read-only
PUT	/me/profile	updateProfile(dto)	Update candidate profile fields	UPDATE in candidateprofiles by user_id	Single update
POST	/me/resumes	uploadResume(file)	Store file, parse, create new resume	INSERT into resumes (file_url, parsed_text)	Atomic insert/file
GET	/me/resumes	listResumes()	List all user's resumes	SELECT from resumes WHERE user_id = ?	Read-only
PATCH	/me/resumes/{resumeId}	updateResumeStatus(id, dto)	Update status/label of resume	UPDATE resumes.active, version_label	Single update
DELETE	/me/resumes/{resumeId}	deleteResume(id)	Delete resume (if not linked to app)	DELETE from resumes, check FK constraints	Atomic delete
GET	/jobs	searchJobs(filter)	List/search jobs by filters	SELECT from jobs with WHERE on skills/location/keywords	Read-only
GET	/jobs/recommended	getRecommendedJobs()	List best-fit jobs via AI matching	SELECT jobs, JOIN semantics/keywords with candidate/resume data	Read-only
GET	/jobs/{jobId}	getJobDetails(jobId)	Get specific job details	SELECT from jobs WHERE id = ?	Read-only
POST	/jobs/{jobId}/applications	applyToJob(jobId, dto)	Apply to job with a resume	INSERT into applications (job_id, student_id, resume_id, etc.)	Atomic insert
GET	/me/applications	listApplications()	List candidate's own job applications	SELECT from applications JOIN jobs by student_id	Read-only
POST	/me/favorites/jobs/{jobId}	addFavoriteJob(jobId)	Add job to favorites	INSERT into favorite_jobs (user_id, job_id)	Single insert
GET	/me/favorites/jobs	listFavoriteJobs()	List all favorite jobs	SELECT from favorite_jobs JOIN jobs	Read-only

DELETE	/me/favorites/jobs/{jobId}	removeFavoriteJob(jobId)	Remove job from favorites	DELETE from favorite_jobs WHERE user_id, job_id	Single delete

Recruiter/Employer APIs

HTTP	URI	Method (Controller)	Logic	DB Operation (Tables)	Transaction Scope
POST	/orgs	createOrganization(dto)	Create new company/org	INSERT into companies	Single insert
PUT	/orgs/{orgId}	updateOrganization(orgId, dto)	Update organization details	UPDATE companies WHERE id = ?	Single update
GET	/orgs/{orgId}/team	listTeam(orgId)	List org/team members	SELECT from recruiters by company_id	Read-only
POST	/orgs/{orgId}/invite	inviteToTeam(orgId, dto)	Invite recruiter to org	INSERT into recruiters and (optionally) users	Atomic insert
POST	/orgs/{orgId}/jobs	postJob(orgId, dto)	Post new internal job	INSERT into internaljobs	Single insert
PUT	/jobs/{jobId}	updateJob(jobId, dto)	Edit/update job details	UPDATE internaljobs/externaljobs WHERE job_id	Single update
PATCH	/jobs/{jobId}/status	updateJobStatus(jobId, dto)	Change job status	UPDATE internaljobs.status or externaljobs.status	Single update
GET	/orgs/{orgId}/jobs	listJobs(orgId)	List org jobs	SELECT from internaljobs WHERE company_id	Read-only
GET	/orgs/{orgId}/jobs/{jobId}/applications	listApplications(orgId, jobId)	List applications to a job	SELECT from applications WHERE internal_job_id	Read-only
PATCH	/applications/{id}/stage	updateApplicationStage(id, dto)	Update stage of application	UPDATE applications.status or similar	Single update

POST	/applications/{id}/notes	addApplicationNote(id, dto)	Add note to application	INSERT into application_notes	Single insert
GET	/applications/{id}/notes	listApplicationNotes(id)	List notes for application	SELECT from application_notes WHERE application_id	Read-only

Admin APIs

HTTP	URI	Method (Controller)	Logic	DB Operation (Tables)	Transaction Scope
GET	/admin/users	listUsers(filter)	List and search all users	SELECT from users	Read-only
PATCH	/admin/users/{id}/status	updateUserStatus(id, dto)	Enable/suspend user	UPDATE users.status WHERE id = ?	Single update
GET	/admin/companies	listOrganizations()	List/search all companies	SELECT from companies	Read-only
PATCH	/admin/companies/{id}/status	updateOrganizationStatus(id, dto)	Enable/suspend/approve organization	UPDATE companies.status WHERE id = ?	Single update
GET	/admin/jobs?flagged=true	listFlaggedJobs()	List flagged jobs	SELECT from internaljobs/externaljobs with flag	Read-only
PATCH	/admin/jobs/{jobId}/moderate	moderateJob(jobId, dto)	Approve/reject/flag job posting	UPDATE internaljobs.status or externaljobs.status	Single update
GET	/admin/audit	getAuditLogs()	View audit logs	SELECT from adminactions	Read-only

Candidate APIs

GET /me/profile

Response:

json

```
{
  "id": "user-123",
```

```
 "name": "Jane Doe",
 "skills": ["Java", "Spring Boot"],
 "education": "BTech CS",
 "city": "Pune",
 "summary": "Motivated graduate...",
 "links": ["https://github.com/jane", "https://linkedin.com/in/jane"]
}
```


PUT /me/profile

Request:

json


```
{
 "name": "Jane Doe",
 "skills": ["Java", "Spring Boot", "React"],
 "education": "BTech CS",
 "city": "Pune",
 "summary": "Updated summary...",
 "links": ["https://github.com/jane"]
}
```


Response: Same structure as the GET above, with updated fields.

POST /me/resumes

Request: multipart/form-data (file upload)
Response:

json


```
{
 "id": "resume-456",
 "file_url": "https://storage/resume_456.pdf",
 "parsed_text": "Experience: ...",
 "active": false,
 "version_label": "Nov2025"
```

```
}
```

GET /me/resumes

Response:

json

```
[
 {
  "id": "resume-456",
  "file_url": "...",
  "active": true,
  "version_label": "Main Resume",
  "created_at": "2025-11-04T09:13:41Z"
 }
]
```

PATCH /me/resumes/{resumeId}

Request:

json

```
{ "active": true, "version_label": "Primary" }
```

Response: Updated resume object as seen above.

DELETE /me/resumes/{resumeId}

Response: 204 No Content

GET /jobs?skills=java&location=pune

Response:

json

```
[
 {
  "id": "job-789",
  "title": "Java Developer",
  "company": "TechBrains",
  "location": "Pune",
  "skills": ["Java", "Spring"],
```

```
  "status": "open",
  "posted_date": "2025-10-31"
 }
]
```

POST /jobs/{jobId}/applications

Request:

json

```
{
 "resume_id": "resume-456"
}
```

Response:

json

```
{
 "application_id": "app-789",
 "status": "applied",
 "applied_at": "2025-11-04T09:16:00Z"
}
```

GET /me/applications

Response:

json

```
[
 {
  "application_id": "app-789",
  "job_title": "Java Developer",
  "company": "TechBrains",
  "status": "interview",
  "applied_at": "2025-11-04T09:16:00Z"
 }
```

]

POST /me/favorites/jobs/{jobId}

Response:

json

```
{"success": true}
```

GET /me/favorites/jobs

Response:

json

```
[
 {
  "id": "job-789",
  "title": "Java Developer",
  "company": "TechBrains"
 }
]
```

DELETE /me/favorites/jobs/{jobId}

Response: 204 No Content

Recruiter/Employer APIs

POST /orgs

Request:

json

```
{
 "name": "Acme Tech",
 "website": "https://acme.com"
}
```

Response:

json

```
{
 "id": "org-001",
```

```
 "name": "Acme Tech",
 "status": "pending"
}
```

PUT /orgs/{orgId}

Request:

json

```
{
 "name": "Acme Tech Pvt Ltd",
 "website": "https://acme.com",
 "status": "approved"
}
```

Response: Updated org info, same shape as the GET.

GET /orgs/{orgId}/team

Response:

json

```
[
 {
  "id": "recruiter-234",
  "name": "Alice",
  "email": "alice@acme.com",
  "role": "recruiter"
 }
]
```

POST /orgs/{orgId}/invite

Request:

json

```
{
 "email": "newrecruiter@acme.com",
 "role": "recruiter"
}
```

Response:

json

```
{
 "invite_id": "invite-001",
 "status": "sent"
}
```

POST /orgs/{orgId}/jobs

Request:

json

```
{
 "title": "React Developer",
 "description": "Frontend...",
 "location": "Remote",
 "skills": ["React", "JS"],
 "employment_type": "full-time"
}
```

Response:

json

```
{
 "id": "job-123",
 "title": "React Developer",
 "status": "open"
```

```
}
```

GET /orgs/{orgId}/jobs

Response:

json

```
[
 {
  "id": "job-123",
  "title": "React Developer",
  "status": "open"
 }
]
```

GET /orgs/{orgId}/jobs/{jobId}/applications

Response:

json

```
[
 {
  "application_id": "app-789",
  "candidate": {
   "id": "user-123",
   "name": "Jane Doe"
  },
  "resume_id": "resume-456",
  "status": "applied",
  "applied_at": "2025-11-04T09:20:00Z"
 }
]
```

PATCH /applications/{applicationId}/stage

Request:

json

```
{ "stage": "interview" }
```

Response:

json

```
{
 "application_id": "app-789",
 "status": "interview"
}
```

POST /applications/{applicationId}/notes

Request:

json

```
{ "text": "Impressed with project experience." }
```

Response:

json

```
{
 "note_id": "note-002",
 "created_at": "2025-11-04T09:18:22Z"
}
```

GET /applications/{applicationId}/notes

Response:

json

```
[
 {
  "note_id": "note-002",
```

```
  "author": "recruiter-234",
  "text": "Impressed with project experience.",
  "created_at": "2025-11-04T09:18:22Z"
 }
]
```


Admin APIs

GET /admin/users

Response:

json


```
[
 {
  "id": "user-123",
  "email": "jane@example.com",
  "role": "candidate",
  "status": "active"
 }
]
```


PATCH /admin/users/{userId}/status

Request:

json

```
{ "status": "suspended" }
```

Response:

json

```
{ "user_id": "user-123", "status": "suspended" }
```

GET /admin/companies

Response:

json

```
[
 {
  "id": "org-001",
  "name": "Acme Tech",
  "status": "approved",
  "website": "https://acme.com"
 }
]
```

PATCH /admin/companies/{orgId}/status

Request:

json

```
{ "status": "approved" }
```

Response:

json

{ "org_id": "org-001", "status": "approved" }

GET /admin/jobs?flagged=true

Response:

json


```
[
 {
  "id": "job-555",
  "title": "Suspicious Job",
  "status": "flagged"
 }
]
```


PATCH /admin/jobs/{jobId}/moderate

Request:

json

{ "status": "approved" }

Response:

json

{ "job_id": "job-555", "status": "approved" }

GET /admin/audit

Response:

json


```
[
 {
  "action_id": "audit-001",
  "user_id": "admin-001",
  "action": "update_user_status",
  "target_type": "user",
  "target_id": "user-123",
  "created_at": "2025-11-04T09:20:34Z"
 }
```

]

All APIs return standard error responses as:

json

```
{
 "status": "error",
 "message": "Reason for failure"
}
```

There are chances that we will add some more APIs in it , according to our need .

1. Table Creation

sql

```
CREATE TABLE Users (
 user_id CHAR(36) PRIMARY KEY,
 email VARCHAR(255) NOT NULL UNIQUE,
 password VARCHAR(255) NOT NULL,
 role ENUM('student','recruiter','admin') NOT NULL,
 is_deleted BOOLEAN DEFAULT FALSE,
 created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
 updated_at DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
 updated_by CHAR(36),
 col1 VARCHAR(255),
 col2 VARCHAR(255),
 col3 VARCHAR(255),
 col4 VARCHAR(255)
);
```

```
CREATE TABLE Organizations (
 company_id CHAR(36) PRIMARY KEY,
 name VARCHAR(255) NOT NULL,
 website VARCHAR(255),
 verification_status ENUM('pending','approved','inactive','suspended') DEFAULT 'pending',
```

```
  is_deleted BOOLEAN DEFAULT FALSE,
  created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
  updated_at DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  updated_by CHAR(36),
  col1 VARCHAR(255),
  col2 VARCHAR(255),
  col3 VARCHAR(255),
  col4 VARCHAR(255)
);
```

```
CREATE TABLE OrgUsers (
  recruiter_id CHAR(36) PRIMARY KEY,
  user_id CHAR(36),
  name VARCHAR(255),
  position VARCHAR(128),
  org_role ENUM('owner','recruiter','viewer') NOT NULL,
  is_deleted BOOLEAN DEFAULT FALSE,
  created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
  updated_at DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  updated_by CHAR(36),
  col1 VARCHAR(255),
  col2 VARCHAR(255),
  col3 VARCHAR(255),
  col4 VARCHAR(255),
  FOREIGN KEY (user_id) REFERENCES Users(user_id)
);

CREATE TABLE CandidateProfiles (
  student_id CHAR(36) PRIMARY KEY,
  user_id CHAR(36),
  name VARCHAR(255),
  skills_json JSON,
  education_json JSON,
  experience_json JSON,
```

```
  links_json JSON,
  is_deleted BOOLEAN DEFAULT FALSE,
  created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
  updated_at DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  updated_by CHAR(36),
  col1 VARCHAR(255),
  col2 VARCHAR(255),
  col3 VARCHAR(255),
  col4 VARCHAR(255),
  FOREIGN KEY (user_id) REFERENCES Users(user_id)
);
```

```
CREATE TABLE Resumes (
  resume_id CHAR(36) PRIMARY KEY,
  user_id CHAR(36),
  storage_path VARCHAR(512),
  parsed_json JSON,
  template_id CHAR(36),
  source ENUM('uploaded','generated'),
  version_label VARCHAR(128),
  active BOOLEAN DEFAULT FALSE,
  is_deleted BOOLEAN DEFAULT FALSE,
  created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
  updated_at DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  updated_by CHAR(36),
  col1 VARCHAR(255),
  col2 VARCHAR(255),
  col3 VARCHAR(255),
  col4 VARCHAR(255),
  FOREIGN KEY (user_id) REFERENCES Users(user_id)
);

CREATE TABLE Jobs (
  job_id CHAR(36) PRIMARY KEY,
  org_id CHAR(36),
```

```
  title VARCHAR(255),
  location_type ENUM('remote','hybrid','onsite') NOT NULL,
  employment_type ENUM('full-time','part-time','intern','contract') NOT NULL,
  experience_min INT,
  experience_max INT,
  skills_required_json JSON,
  skills_preferred_json JSON,
  jd_text TEXT,
  status ENUM('draft','open','closed') DEFAULT 'open',
  is_deleted BOOLEAN DEFAULT FALSE,
  created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
  updated_at DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  updated_by CHAR(36),
  col1 VARCHAR(255),
  col2 VARCHAR(255),
  col3 VARCHAR(255),
  col4 VARCHAR(255),
  FOREIGN KEY (org_id) REFERENCES Organizations(company_id)
);



CREATE TABLE Applications (
  application_id CHAR(36) PRIMARY KEY,
  job_id CHAR(36),
  user_id CHAR(36),
  resume_id CHAR(36),
  stage ENUM('applied','shortlisted','interview','offer','hired','rejected') DEFAULT 'applied',
  decision ENUM('offer','reject') DEFAULT NULL,
  is_deleted BOOLEAN DEFAULT FALSE,
  created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
  updated_at DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  updated_by CHAR(36),
  col1 VARCHAR(255),
  col2 VARCHAR(255),
  col3 VARCHAR(255),
  col4 VARCHAR(255),
  FOREIGN KEY (job_id) REFERENCES Jobs(job_id),
  FOREIGN KEY (user_id) REFERENCES Users(user_id),
  FOREIGN KEY (resume_id) REFERENCES Resumes(resume_id)
);

CREATE TABLE Scores (
  application_id CHAR(36) PRIMARY KEY,
  keyword_score INT,
```

```
  semantic_score INT,
  fit_flag BOOLEAN,
  explanation TEXT,
  is_deleted BOOLEAN DEFAULT FALSE,
  created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
  updated_at DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  updated_by CHAR(36),
  col1 VARCHAR(255),
  col2 VARCHAR(255),
  col3 VARCHAR(255),
  col4 VARCHAR(255),
  FOREIGN KEY (application_id) REFERENCES Applications(application_id)
);
```

```
CREATE TABLE AdminAudit (
  id CHAR(36) PRIMARY KEY,
  actor_user_id CHAR(36),
  action TEXT,
  target_type ENUM('user','company','job','application'),
  target_id CHAR(36),
  payload_json JSON,
  is_deleted BOOLEAN DEFAULT FALSE,
  created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
  updated_at DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  updated_by CHAR(36),
  col1 VARCHAR(255),
  col2 VARCHAR(255),
  col3 VARCHAR(255),
  col4 VARCHAR(255),
  FOREIGN KEY (actor_user_id) REFERENCES Users(user_id)
);
```

2. Encapsulated Views

sql

```sql
-- View: All non-deleted users
CREATE VIEW ActiveUsers AS
SELECT user_id, email, role, created_at
FROM Users
WHERE is_deleted = FALSE AND role <> 'admin';

-- View: All open jobs by company and employment type
CREATE VIEW OpenJobsSummary AS
SELECT org_id, title, employment_type, status
FROM Jobs
WHERE is_deleted = FALSE AND status = 'open';

-- View: Application status matrix
CREATE VIEW ApplicationStatusMatrix AS
SELECT job_id, stage, COUNT(*) AS application_count
FROM Applications
WHERE is_deleted = FALSE
GROUP BY job_id, stage;
```

3. Stored Procedures (Business Logic Encapsulation)

```sql
-- Soft delete user and track admin
DELIMITER //
CREATE PROCEDURE SoftDeleteUser(IN p_user_id CHAR(36), IN p_admin_id CHAR(36))
BEGIN
  UPDATE Users
  SET is_deleted = TRUE, updated_at = NOW(), updated_by = p_admin_id
  WHERE user_id = p_user_id;
END //
DELIMITER ;

-- Update job status (open/closed) and track admin
DELIMITER //
CREATE PROCEDURE UpdateJobStatus(IN p_job_id CHAR(36), IN p_status ENUM('draft','open','closed'), IN
p_admin_id CHAR(36))
BEGIN
  UPDATE Jobs
```

```
  SET status = p_status, updated_at = NOW(), updated_by = p_admin_id
  WHERE job_id = p_job_id;
END //
DELIMITER ;

-- Promote application to stage and track admin
DELIMITER //
CREATE PROCEDURE PromoteApplicationStage(IN p_application_id CHAR(36), IN p_stage
ENUM('applied','shortlisted','interview','offer','hired','rejected'), IN p_admin_id CHAR(36))
BEGIN
  UPDATE Applications
  SET stage = p_stage, updated_at = NOW(), updated_by = p_admin_id
  WHERE application_id = p_application_id;
END //
DELIMITER ;
```

Stack/Layer	How to Generate and Use UUID
Spring Boot/Java	UUID.randomUUID().toString()
Node.js	require('uuid').v4()
MySQL SQL	UUID()
Python	str(uuid.uuid4())
Hibernate/JPA	@GenericGenerator(name = "uuid2", strategy = "uuid2")

Entities (POJOs)

User.java

java

```
@Entity
@Table(name = "Users")
public class User {
  @Id
  @Column(name = "user_id")
  private String userId;

  private String email;
  private String password;
  private String role;
  private Boolean isDeleted;
  private LocalDateTime createdAt;
  private LocalDateTime updatedAt;
  private String updatedBy;
  private String col1, col2, col3, col4;
  // getters/setters
```

```
}
```

Organization.java

java

```
@Entity
@Table(name = "Organizations")
public class Organization {
  @Id
  @Column(name = "company_id")
  private String companyId;

  private String name;
  private String website;
  private String verificationStatus;
  private Boolean isDeleted;
  private LocalDateTime createdAt;
  private LocalDateTime updatedAt;
  private String updatedBy;
  private String col1, col2, col3, col4;
  // getters/setters
}
```

OrgUser.java

java

```
@Entity
@Table(name = "OrgUsers")
public class OrgUser {
  @Id
  @Column(name = "recruiter_id")
  private String recruiterId;

  private String userId;
  private String name;
  private String position;
  private String orgRole;
  private Boolean isDeleted;
  private LocalDateTime createdAt;
  private LocalDateTime updatedAt;
  private String updatedBy;
  private String col1, col2, col3, col4;
```

```
    // getters/setters
}
```

CandidateProfile.java

```
java
@Entity
@Table(name = "CandidateProfiles")
public class CandidateProfile {
    @Id
    @Column(name = "student_id")
    private String studentId;

    private String userId;
    private String name;
    private String skillsJson;
    private String educationJson;
    private String experienceJson;
    private String linksJson;
    private Boolean isDeleted;
    private LocalDateTime createdAt;
    private LocalDateTime updatedAt;
    private String updatedBy;
    private String col1, col2, col3, col4;
    // getters/setters
}
```

Resume.java

```
java
@Entity
@Table(name = "Resumes")
public class Resume {
    @Id
    @Column(name = "resume_id")
    private String resumeId;

    private String userId;
    private String storagePath;
    private String parsedJson;
    private String templateId;
    private String source;
    private String versionLabel;
    private Boolean active;
```

```
    private Boolean isDeleted;
    private LocalDateTime createdAt;
    private LocalDateTime updatedAt;
    private String updatedBy;
    private String col1, col2, col3, col4;
    // getters/setters
}
```

Job.java

```
java
@Entity
@Table(name = "Jobs")
public class Job {
    @Id
    @Column(name = "job_id")
    private String jobId;

    private String orgId;
    private String title;
    private String locationType;
    private String employmentType;
    private Integer experienceMin;
    private Integer experienceMax;
    private String skillsRequiredJson;
    private String skillsPreferredJson;
    private String jdText;
```

```
    private String status;
    private Boolean isDeleted;
    private LocalDateTime createdAt;
    private LocalDateTime updatedAt;
    private String updatedBy;
    private String col1, col2, col3, col4;
    // getters/setters
}
```

Application.java

```
java
@Entity
@Table(name = "Applications")
public class Application {
    @Id
    @Column(name = "application_id")
    private String applicationId;

    private String jobId;
    private String userId;
    private String resumeId;
    private String stage;
    private String decision;
    private Boolean isDeleted;
    private LocalDateTime createdAt;
    private LocalDateTime updatedAt;
    private String updatedBy;
    private String col1, col2, col3, col4;
    // getters/setters
```

```
}
```

Score.java

```
java
@Entity
@Table(name = "Scores")
public class Score {
  @Id
  @Column(name = "application_id")
  private String applicationId;

  private Integer keywordScore;
  private Integer semanticScore;
  private Boolean fitFlag;
  private String explanation;
  private Boolean isDeleted;
  private LocalDateTime createdAt;
  private LocalDateTime updatedAt;
  private String updatedBy;
  private String col1, col2, col3, col4;
  // getters/setters
}
```

AdminAudit.java

```
java
@Entity
@Table(name = "AdminAudit")
public class AdminAudit {
  @Id
  private String id;

  private String actorUserId;
  private String action;
  private String targetType;
  private String targetId;
  private String payloadJson;
  private Boolean isDeleted;
  private LocalDateTime createdAt;
  private LocalDateTime updatedAt;
  private String updatedBy;
  private String col1, col2, col3, col4;
```

```
  // getters/setters
}
```

DTOs : Add More DTOs As Needed

Only include fields relevant for API inputs/outputs.

UserDTO.java

```
java
public class UserDTO {
  private String userId;
  private String email;
  private String role;
  // no password
}
```

OrganizationDTO.java, JobDTO.java, ResumeDTO.java, etc.

```
java
public class OrganizationDTO {
  private String companyId;
  private String name;
```

```
    private String website;
    private String verificationStatus;
}
```

java

```
public class JobDTO {
    private String jobId;
    private String orgId;
    private String title;
    private String locationType;
    // ...other public facing fields
}
```

DAO (Repository) interfaces

```
@Repository
public interface UserRepository extends JpaRepository<User, String> {}

@Repository
public interface OrganizationRepository extends JpaRepository<Organization, String> {}

@Repository
public interface OrgUserRepository extends JpaRepository<OrgUser, String> {}

@Repository
public interface CandidateProfileRepository extends JpaRepository<CandidateProfile, String> {}

@Repository
public interface ResumeRepository extends JpaRepository<Resume, String> {}

@Repository
public interface JobRepository extends JpaRepository<Job, String> {}
```

```
@Repository
public interface ApplicationRepository extends JpaRepository<Application, String> {}

@Repository
public interface ScoreRepository extends JpaRepository<Score, String> {}

@Repository
public interface AdminAuditRepository extends JpaRepository<AdminAudit, String> {}
```