# For Job Portal

## ➢ User Authentication & Profiles

1. Signup

   URL: accounts/api/signup

   Method: POST

   Headers: Content-type: application/json

   Description: Register a new user (default role = jobseeker, can be changed to employer).

   Request Body:

```
{
  "username": "string",
  "email": "string",
  "password": "string",
  "role": "jobseeker | employer"
}
```

   Response:

```
{
  "message": "User registered successfully",
  "user_id": "int",
  "role": "string"
}
```

2. Login

   URL: accounts/api/login
   Method: POST
   Headers: Content-type: application/json
   Description: Authenticate user and return JWT/Token.

Request Body:

```
{
 "username_or_email": "string",
 "password": "string"
}
```

Response:

```
{
 "token": "jwt_token_string",
 "user_id": "int",
 "role": "string"
}
```

3. Logout

   URL: accounts/api/logout
   Method: POST
   Headers: Authorization: Bearer <token>
   Description: Logout user and invalidate token.
   Request Body: {}
   Response:

```
{
 "message": "Logged out successfully"
}
```

4. Update User Profile

   URL: accounts/api/profile/update
   Method: PUT
   Headers: Authorization: Bearer <token>, Content-type: application/json
   Description: Update user profile (jobseeker or employer).
   Request Body:

```
{
 "username": "string",
 "email": "string",
 "bio": "string",
 "skills": ["string"],      # for jobseeker
 "company_name": "string"    # for employer
}
```

   Response:

```
{
 "message": "Profile updated successfully"
```

}

5. Change Password
   URL: accounts/api/change-password
   Method: POST
   Headers: Authorization: Bearer <token>, Content-type: application/json
   Description: Allow user to change password.
   Request Body:
   {
    "old_password": "string",
    "new_password": "string"
   }
   Response:
   {
    "message": "Password changed successfully"
   }

6. Delete Account
   URL: accounts/api/delete
   Method: DELETE
   Headers: Authorization: Bearer <token>
   Description: Permanently delete a user account.
   Request Body: {}
   Response:
   {
    "message": "Account deleted successfully"
   }

7. Token Obtain (Login with JWT)
   URL: accounts/api/token
   Method: POST
   Headers: Content-type: application/json
   Description: Generate JWT access & refresh tokens for authentication.
   Request Body:
   {
    "username": "string",
    "password": "string"
   }

Response:
{
"access": "jwt_access_token",
"refresh": "jwt_refresh_token"
}

8. Token Refresh
   URL: accounts/api/token/refresh
   Method: POST
   Headers: Content-type: application/json
   Description: Refresh the JWT access token using the refresh token.
   Request Body:
   {
   "refresh": "jwt_refresh_token"
   }
   Response:
   {
   "access": "new_jwt_access_token"
   }

9. Token Blacklist (Logout for JWT Refresh Tokens)
   URL: accounts/api/token/blacklist
   Method: POST
   Headers: Authorization: Bearer <refresh_token>, Content-type: application/json
   Description: Invalidate (blacklist) a refresh token, usually for logout.
   Request Body:
   {
   "refresh": "jwt_refresh_token"
   }
   Response:
   {
   "message": "Token blacklisted successfully"
   }

## ➢ Job Listings & Applications

1. Create Job Listing (Employer only)
   URL: jobs/api/create
   Method: POST
   Headers: Authorization: Bearer <token>, Content-type: application/json
   Description: Employer posts a new job listing.
   Request Body:
   ```
   {
    "title": "string",
    "description": "string",
    "requirements": ["string"],
    "location": "string",
    "salary_range": "string",
    "employment_type": "full-time | part-time | internship | contract"
   }
   ```
   Response:
   ```
   {
    "message": "Job created successfully",
    "job_id": "int"
   }
   ```

2. Get All Job Listings (Public)
   URL: jobs/api/list
   Method: GET
   Headers: Content-type: application/json
   Description: Fetch all available job listings (public endpoint).
   Request Body: {}
   Response:
   ```
   [
    {
     "job_id": "int",
     "title": "string",
     "company_name": "string",
     "location": "string",
     "employment_type": "string"
    }
   ]
   ```

3. Get Job Details
   URL: jobs/api/detail/<job_id>

Method: GET
Headers: Content-type: application/json
Description: Retrieve detailed information about a specific job.
Request Body: {}
Response:

```
{
 "job_id": "int",
 "title": "string",
 "description": "string",
 "requirements": ["string"],
 "location": "string",
 "salary_range": "string",
 "employment_type": "string",
 "posted_by": "employer_id"
}
```

4. Update Job Listing (Employer only)
   URL: jobs/api/update/<job_id>
   Method: PUT
   Headers: Authorization: Bearer <token>, Content-type: application/json
   Description: Employer updates their job listing.
   Request Body:

```
{
 "title": "string",
 "description": "string",
 "requirements": ["string"],
 "location": "string",
 "salary_range": "string"
}
```

   Response:

```
{
 "message": "Job updated successfully"
}
```

5. Delete Job Listing (Employer only)
   URL: jobs/api/delete/<job_id>
   Method: DELETE
   Headers: Authorization: Bearer <token>
   Description: Employer deletes their job listing.

Request Body: {}
Response:

```
{
 "message": "Job deleted successfully"
}
```

6. Apply for a Job (Jobseeker only)
   URL: applications/api/apply/<job_id>
   Method: POST
   Headers: Authorization: Bearer <token>, Content-type: application/json
   Description: Jobseeker applies for a job.
   Request Body:

```
{
 "resume_url": "string",
 "cover_letter": "string"
}
```

   Response:

```
{
 "message": "Application submitted successfully",
 "application_id": "int"
}
```

7. Get Applications for a Job (Employer only)
   URL: applications/api/job/<job_id>
   Method: GET
   Headers: Authorization: Bearer <token>
   Description: Employer views all applications for their posted job.
   Request Body: {}
   Response:

```
[
 {
  "application_id": "int",
  "jobseeker_id": "int",
  "resume_url": "string",
  "cover_letter": "string",
  "status": "applied | reviewed | shortlisted | rejected"
 }
]
```

8. Get Jobseeker Applications (Jobseeker only)
   URL: applications/api/my-applications
   Method: GET
   Headers: Authorization: Bearer <token>
   Description: Jobseeker views all jobs they have applied to.
   Request Body: {}
   Response:

```
[
 {
   "application_id": "int",
   "job_id": "int",
   "job_title": "string",
   "status": "applied | reviewed | shortlisted | rejected"
 }
]
```

9. Update Application Status (Employer only)
   URL: applications/api/update/<application_id>
   Method: PUT
   Headers: Authorization: Bearer <token>, Content-type: application/json
   Description: Employer updates application status.
   Request Body:

```
{
  "status": "reviewed | shortlisted | rejected"
}
```

   Response:

```
{
  "message": "Application status updated"
}
```

## ➢ Advanced Job Search & Filters

1. Filtered Job Search
   URL: jobs/api/search
   Method: GET

Headers: Content-type: application/json
Description: Search jobs using filters like location, industry, experience level, salary range, and job type.
Request Body: {}
Query Parameters:
location=string
industry=string
experience=entry | mid | senior
salary_min=int
salary_max=int
job_type=full-time | part-time | remote | internship | contract

Example:
/jobs/api/search?location=Delhi&industry=IT&experience=mid&job_type=remote

Response:

```
[
 {
   "job_id": "int",
   "title": "string",
   "company_name": "string",
   "location": "string",
   "industry": "string",
   "experience_level": "string",
   "salary_range": "string",
   "job_type": "string"
 }
]
```

2. Keyword-based Job Search
   URL: jobs/api/search/keyword
   Method: GET
   Headers: Content-type: application/json
   Description: Search jobs based on keywords in title or description.
   Request Body: {}
   Query Parameters:
   q=string

   Example: /jobs/api/search/keyword?q=python developer

Response:

```
[
 {
   "job_id": "int",
   "title": "string",
   "company_name": "string",
   "location": "string",
   "salary_range": "string",
   "job_type": "string"
 }
]
```

## ➢ Resume Upload & Parsing

1. Resume Upload
   URL: resume/api/upload
   Method: POST
   Headers: Authorization: Bearer <token>, Content-type: multipart/form-data
   Description: Upload a resume (PDF/DOCX) for job applications.
   Request Body (Form Data):

```
{
 "resume_file": <file>   # Accepts .pdf, .docx
}
```

   Response:

```
{
 "message": "Resume uploaded successfully",
 "resume_url": "string",
 "file_type": "pdf | docx"
}
```

2. Get Uploaded Resume
   URL: resume/api/get/<user_id>
   Method: GET
   Headers: Authorization: Bearer <token>
   Description: Fetch the latest uploaded resume for the user.
   Request Body: {}

Response:
{
"resume_url": "string",
"uploaded_at": "datetime"
}

3. Delete Resume
   URL: resume/api/delete
   Method: DELETE
   Headers: Authorization: Bearer <token>
   Description: Delete the uploaded resume from user profile.
   Request Body: {}
   Response:
   {
   "message": "Resume deleted successfully"
   }

4. AI-based Resume Parsing (Profile Auto-fill)
   URL: resume/api/parse
   Method: POST
   Headers: Authorization: Bearer <token>, Content-type: multipart/form-data
   Description: Upload a resume (PDF/DOCX) and extract structured data using AI for automated profile creation.
   Request Body (Form Data):
   {
   "resume_file": <file>
   }
   Response:
   {
   "message": "Resume parsed successfully",
   "extracted_data": {
   "name": "string",
   "email": "string",
   "phone": "string",
   "skills": ["string"],
   "education": [
   {
   "degree": "string",
   "institution": "string",

```
      "year": "string"
     }
    ],
    "experience": [
     {
      "job_title": "string",
      "company": "string",
      "duration": "string"
     }
    ]
   }
}
```


5. Apply Parsed Data to Profile
   URL: resume/api/parse/apply
   Method: POST
   Headers: Authorization: Bearer <token>, Content-type: application/json
   Description: Save extracted resume data directly into the user's profile.
   Request Body:

```
{
 "name": "string",
 "email": "string",
 "phone": "string",
 "skills": ["string"],
 "education": [...],
 "experience": [...]
}
```

   Response:

```
{
 "message": "Profile updated with parsed resume data"
}
```


## ➢ Application Tracking System (ATS)

1. Review Applications (Employer)
   URL: ats/api/review/<job_id>
   Method: GET
   Headers: Authorization: Bearer <token>

Description: Employer fetches all applications for a specific job posting.
Request Body: {}
Response:

```
[
 {
  "application_id": "int",
  "jobseeker_id": "int",
  "jobseeker_name": "string",
  "resume_url": "string",
  "cover_letter": "string",
  "status": "applied | reviewed | shortlisted | rejected"
 }
]
```

2. Update Application Status (Employer)
   URL: ats/api/update-status/<application_id>
   Method: PUT
   Headers: Authorization: Bearer <token>, Content-type: application/json
   Description: Employer updates application status (reviewed, shortlisted, rejected).
   Request Body:

```
{
 "status": "reviewed | shortlisted | rejected",
 "notes": "string"
}
```

   Response:

```
{
 "message": "Application status updated",
 "application_id": "int",
 "new_status": "string"
}
```

3. **Schedule Interview**
   URL: ats/api/schedule-interview/<application_id>
   Method: POST
   Headers: Authorization: Bearer <token>, Content-type: application/json
   Description: Employer schedules an interview for a shortlisted candidate.
   Request Body:

```
{
 "date": "YYYY-MM-DD",
```

```
 "time": "HH:MM",
 "mode": "online | offline",
 "location": "string (if offline)",
 "meeting_link": "string (if online)"
}
Response:
{
 "message": "Interview scheduled successfully",
 "application_id": "int",
 "interview_details": {
  "date": "string",
  "time": "string",
  "mode": "string",
  "location_or_link": "string"
 }
}
```

4. Get Scheduled Interviews (Employer & Jobseeker)
   URL: ats/api/interviews/my
   Method: GET
   Headers: Authorization: Bearer <token>
   Description: Fetch upcoming interviews for the logged-in user (jobseeker sees their interviews, employer sees all they scheduled).
   Request Body: {}
   Response:

```
[
 {
  "interview_id": "int",
  "application_id": "int",
  "candidate_name": "string",
  "job_title": "string",
  "date": "YYYY-MM-DD",
  "time": "HH:MM",
  "mode": "online | offline",
  "location_or_link": "string"
 }
]
```

5. Cancel / Reschedule Interview

URL: ats/api/interview/update/<interview_id>
Method: PUT
Headers: Authorization: Bearer <token>, Content-type: application/json
Description: Employer updates or cancels a scheduled interview.
Request Body:

```
{
 "action": "reschedule | cancel",
 "new_date": "YYYY-MM-DD (optional if reschedule)",
 "new_time": "HH:MM (optional if reschedule)",
 "new_mode": "string (optional)",
 "new_location_or_link": "string (optional)"
}
```

Response:

```
{
 "message": "Interview rescheduled/canceled successfully",
 "interview_id": "int"
}
```

## ➢ Company Pages & Reviews

1. Create Company Profile (Employer only)
   URL: company/api/create
   Method: POST
   Headers: Authorization: Bearer <token>, Content-type: application/json
   Description: Employer creates a company profile page.
   Request Body:

```
{
 "company_name": "string",
 "industry": "string",
 "location": "string",
 "website": "string",
 "description": "string",
 "logo_url": "string"
}
```

   Response:

```
{
 "message": "Company profile created successfully",
 "company_id": "int"
}
```

2. Update Company Profile (Employer only)
   URL: company/api/update/<company_id>
   Method: PUT
   Headers: Authorization: Bearer <token>, Content-type: application/json
   Description: Employer updates their company profile.
   Request Body:
   ```
   {
    "industry": "string",
    "location": "string",
    "website": "string",
    "description": "string",
    "logo_url": "string"
   }
   ```
   Response:
   ```
   {
    "message": "Company profile updated successfully"
   }
   ```

3. Get Company Profile (Public)
   URL: company/api/detail/<company_id>
   Method: GET
   Headers: Content-type: application/json
   Description: Fetch details of a company profile.
   Request Body: {}
   Response:
   ```
   {
    "company_id": "int",
    "company_name": "string",
    "industry": "string",
    "location": "string",
    "website": "string",
    "description": "string",
    "logo_url": "string",
    "average_rating": "float"
   }
   ```

4. List All Companies (Public)

URL: company/api/list
Method: GET
Headers: Content-type: application/json
Description: Fetch a list of all companies with basic details.
Request Body: {}
Response:

```
[
 {
  "company_id": "int",
  "company_name": "string",
  "industry": "string",
  "location": "string",
  "average_rating": "float"
 }
]
```

5. Add Company Review
   URL: reviews/api/add/<company_id>
   Method: POST
   Headers: Authorization: Bearer <token>, Content-type: application/json
   Description: Employees/jobseekers leave a review & rating for a company.
   Request Body:

```
{
 "rating": 1-5,
 "review_text": "string",
 "pros": "string",
 "cons": "string"
}
```

   Response:

```
{
 "message": "Review added successfully",
 "review_id": "int"
}
```

6. Get Company Reviews (Public)
   URL: reviews/api/company/<company_id>
   Method: GET
   Headers: Content-type: application/json
   Description: Fetch all reviews for a company.

Request Body: {}
Response:

```
[
 {
   "review_id": "int",
   "user_id": "int",
   "rating": 1-5,
   "review_text": "string",
   "pros": "string",
   "cons": "string",
   "created_at": "datetime"
 }
]
```

7. Update Review (Author only)
   URL: reviews/api/update/<review_id>
   Method: PUT
   Headers: Authorization: Bearer <token>, Content-type: application/json
   Description: User updates their own review.
   Request Body:

```
{
 "rating": 1-5,
 "review_text": "string",
 "pros": "string",
 "cons": "string"
}
```

   Response:

```
{
 "message": "Review updated successfully"
}
```

8. Delete Review (Author only)
   URL: reviews/api/delete/<review_id>
   Method: DELETE
   Headers: Authorization: Bearer <token>
   Description: User deletes their own review.
   Request Body: {}

Response:
{
"message": "Review deleted successfully"
}

## ➢ Notifications & Alerts

1. Set Notification Preferences
   URL: notifications/api/preferences/update
   Method: PUT
   Headers: Authorization: Bearer <token>, Content-type: application/json
   Description: User sets preferences for email & push notifications.
   Request Body:
   {
   "email_notifications": true,
   "push_notifications": true,
   "job_alerts": true,
   "application_updates": true,
   "interview_reminders": true
   }
   Response:
   {
   "message": "Notification preferences updated successfully"
   }

2. Get Notification Preferences
   URL: notifications/api/preferences/get
   Method: GET
   Headers: Authorization: Bearer <token>
   Description: Retrieve user's notification settings.
   Request Body: {}
   Response:
   {
   "email_notifications": true,
   "push_notifications": false,
   "job_alerts": true,
   "application_updates": true,
   "interview_reminders": false

}

3. Get All Notifications (In-App)
   URL: notifications/api/list
   Method: GET
   Headers: Authorization: Bearer <token>
   Description: Fetch all in-app notifications for a user.
   Request Body: {}
   Response:
   [
    {
     "notification_id": "int",
     "type": "job_alert | application_update | interview",
     "title": "string",
     "message": "string",
     "is_read": false,
     "created_at": "datetime"
    }
   ]

4. **Mark Notification as Read**
   URL: notifications/api/mark-read/<notification_id>
   Method: PUT
   Headers: Authorization: Bearer <token>
   Description: Mark a single notification as read.
   Request Body: {}
   Response:
   {
    "message": "Notification marked as read"
   }

5. Delete Notification
   URL: notifications/api/delete/<notification_id>
   Method: DELETE
   Headers: Authorization: Bearer <token>
   Description: Delete a notification from user's inbox.
   Request Body: {}
   Response:

{
"message": "Notification deleted successfully"
}

## ➢ Payment Gateway for Premium Listings

1. Create Payment for Featured Job
   URL: payments/api/job/checkout
   Method: POST
   Headers: Authorization: Bearer <token>, Content-type: application/json
   Description: Initiate a payment request for featuring a job posting.
   Request Body:
   {
   "job_id": "int",
   "plan": "basic | standard | premium", # Different highlight levels
   "amount": "float",
   "currency": "string"
   }
   Response:
   {
   "payment_id": "string",
   "job_id": "int",
   "amount": "float",
   "currency": "string",
   "payment_url": "string" # Redirect to payment gateway
   }

2. Verify Payment for Job Feature
   URL: payments/api/job/verify
   Method: POST
   Headers: Authorization: Bearer <token>, Content-type: application/json
   Description: Verify the payment transaction after job listing purchase.
   Request Body:
   {
   "payment_id": "string",
   "transaction_id": "string",
   "status": "success | failed"
   }

Response:

```
{
 "message": "Payment verified successfully",
 "job_id": "int",
 "featured_until": "datetime"
}
```

3. Get Featured Job Listings (Public)
   URL: jobs/api/featured
   Method: GET
   Headers: Content-type: application/json
   Description: Fetch all featured job listings (highlighted on homepage).
   Request Body: {}
   Response:

```
[
 {
  "job_id": "int",
  "title": "string",
  "company_name": "string",
  "featured_until": "datetime"
 }
]
```

4. Subscribe to Plan
   URL: payments/api/subscription/checkout
   Method: POST
   Headers: Authorization: Bearer <token>, Content-type: application/json
   Description: Initiate subscription payment for recruiters/companies.
   Request Body:

```
{
 "plan": "monthly | quarterly | yearly",
 "amount": "float",
 "currency": "string"
}
```

   Response:

```
{
 "subscription_id": "string",
 "plan": "string",
 "amount": "float",
```

```
 "payment_url": "string"
}
```

5. Verify Subscription Payment
   URL: payments/api/subscription/verify
   Method: POST
   Headers: Authorization: Bearer <token>, Content-type: application/json
   Description: Verify subscription payment after purchase.
   Request Body:
   ```
   {
    "subscription_id": "string",
    "transaction_id": "string",
    "status": "success | failed"
   }
   ```
   Response:
   ```
   {
    "message": "Subscription activated successfully",
    "subscription_id": "string",
    "valid_until": "datetime"
   }
   ```

6. Get Active Subscription
   URL: payments/api/subscription/active
   Method: GET
   Headers: Authorization: Bearer <token>
   Description: Retrieve details of the logged-in company's active subscription.
   Request Body: {}
   Response:
   ```
   {
    "subscription_id": "string",
    "plan": "string",
    "started_at": "datetime",
    "valid_until": "datetime",
    "status": "active | expired"
   }
   ```

7. Cancel Subscription
   URL: payments/api/subscription/cancel/<subscription_id>

Method: PUT
Headers: Authorization: Bearer <token>
Description: Cancel an active subscription (no refund after cut-off).
Request Body: {}
Response:

```
{
 "message": "Subscription cancelled successfully",
 "subscription_id": "string"
}
```

# For Professional Community

## ➢ Networking & Connections

1. Send Connection Request
   URL: community/api/connections/request
   Method: POST
   Headers: Authorization: Bearer <token>, Content-type: application/json
   Description: Send a connection request to another user.
   Request Body:

```
{
 "receiver_id": "int"
}
```

   Response:

```
{
 "message": "Connection request sent",
 "request_id": "string",
 "status": "pending"
}
```

2. Accept/Reject Connection Request
   URL: community/api/connections/respond/<request_id>
   Method: PUT
   Headers: Authorization: Bearer <token>
   Description: Accept or reject a pending connection request.

Request Body:

```
{
 "action": "accept | reject"
}
```

Response:

```
{
 "message": "Connection accepted",
 "connection_id": "string"
}
```

3. Get My Connections

   URL: community/api/connections/my
   Method: GET
   Headers: Authorization: Bearer <token>
   Description: Fetch a list of all confirmed connections for the logged-in user.
   Request Body: {}
   Response:

```
[
 {
  "user_id": "int",
  "name": "string",
  "headline": "string",
  "profile_picture": "string"
 }
]
```

4. Follow a Professional

   URL: community/api/follow/<user_id>
   Method: POST
   Headers: Authorization: Bearer <token>
   Description: Follow a professional without sending connection request.
   Request Body: {}
   Response:

```
{
 "message": "Now following this user",
 "user_id": "int"
}
```

5. Direct Messaging (1-on-1 Chat)
   URL: community/api/messages/send
   Method: POST
   Headers: Authorization: Bearer <token>, Content-type: application/json
   Description: Send a private message to a connection.
   Request Body:
   ```
   {
    "receiver_id": "int",
    "message": "string"
   }
   ```
   Response:
   ```
   {
    "message_id": "string",
    "status": "sent",
    "timestamp": "datetime"
   }
   ```

6. Fetch Chat History
   URL: community/api/messages/history/<user_id>
   Method: GET
   Headers: Authorization: Bearer <token>
   Description: Retrieve chat history with a specific user.
   Request Body: {}
   Response:
   ```
   [
    {
     "message_id": "string",
     "sender_id": "int",
     "receiver_id": "int",
     "message": "string",
     "timestamp": "datetime"
    }
   ]
   ```

7. Create Forum Post
   URL: community/api/forums/post
   Method: POST
   Headers: Authorization: Bearer <token>, Content-type: application/json
   Description: Create a discussion post in a professional community forum.

Request Body:

```
{
 "title": "string",
 "content": "string",
 "tags": ["string"]
}
```

Response:

```
{
 "post_id": "string",
 "message": "Post created successfully"
}
```

8. Get Forum Posts
   URL: community/api/forums/posts
   Method: GET
   Headers: Content-type: application/json
   Description: Fetch latest forum posts (paginated).
   Request Body: {}
   Response:

```
[
 {
   "post_id": "string",
   "title": "string",
   "content": "string",
   "author": "string",
   "tags": ["string"],
   "created_at": "datetime",
   "likes": "int",
   "comments_count": "int"
 }
]
```

9. Comment on Forum Post
   URL: community/api/forums/comment
   Method: POST
   Headers: Authorization: Bearer <token>, Content-type: application/json
   Description: Add a comment to a forum post.
   Request Body:

```
{
```

"post_id": "string",
"comment": "string"
}
Response:
{
"comment_id": "string",
"message": "Comment added successfully"
}

10. Like/Unlike Forum Post
URL: community/api/forums/like/<post_id>
Method: POST
Headers: Authorization: Bearer <token>
Description: Like or unlike a forum post.
Request Body: {}
Response:
{
"message": "Post liked",
"likes_count": "int"
}

## ➢ Groups & Events

1. Create a Group
URL: community/api/groups/create
Method: POST
Headers: Authorization: Bearer <token>, Content-type: application/json
Description: Create an industry-specific group.
Request Body:
{
"name": "string",
"description": "string",
"industry": "string",
"privacy": "public | private"
}
Response:
{
"group_id": "string",

"message": "Group created successfully"
}

2. Join a Group
   URL: community/api/groups/join/<group_id>
   Method: POST
   Headers: Authorization: Bearer <token>
   Description: Request to join a group (auto-approve for public groups).
   Request Body: {}
   Response:
   {
   "message": "Join request sent/approved",
   "status": "pending | approved"
   }

3. Post in Group
   URL: community/api/groups/post
   Method: POST
   Headers: Authorization: Bearer <token>, Content-type: application/json
   Description: Post content inside a specific group.
   Request Body:
   {
   "group_id": "string",
   "title": "string",
   "content": "string",
   "attachments": ["string"]
   }
   Response:
   {
   "post_id": "string",
   "message": "Post added to group"
   }

4. Get Group Posts
   URL: community/api/groups/posts/<group_id>
   Method: GET
   Headers: Authorization: Bearer <token>
   Description: Fetch latest posts in a group.

Request Body: {}
Response:

```
[
 {
   "post_id": "string",
   "author": "string",
   "title": "string",
   "content": "string",
   "created_at": "datetime"
 }
]
```

5. Leave Group
   URL: community/api/groups/leave/<group_id>
   Method: POST
   Headers: Authorization: Bearer <token>
   Description: Leave a group.
   Response:

```
{
  "message": "You have left the group"
}
```

6. Create Event
   URL: community/api/events/create
   Method: POST
   Headers: Authorization: Bearer <token>, Content-type: application/json
   Description: Create an event (webinar, meetup, job fair).
   Request Body:

```
{
  "title": "string",
  "description": "string",
  "event_type": "webinar | meetup | job_fair",
  "location": "string",
  "start_time": "datetime",
  "end_time": "datetime",
  "registration_required": true
}
```

   Response:

```
{
```

```
 "event_id": "string",
 "message": "Event created successfully"
}
```

7. Register for Event
   URL: community/api/events/register/<event_id>
   Method: POST
   Headers: Authorization: Bearer <token>
   Description: Register for a specific event.
   Request Body: {}
   Response:

```
{
 "message": "Registered successfully",
 "event_id": "string",
 "status": "confirmed"
}
```

8. Get Upcoming Events
   URL: community/api/events/upcoming
   Method: GET
   Headers: Content-type: application/json
   Description: Fetch a list of upcoming events.
   Response:

```
[
 {
  "event_id": "string",
  "title": "string",
  "event_type": "webinar",
  "location": "string",
  "start_time": "datetime",
  "end_time": "datetime",
  "registered_users": "int"
 }
]
```

9. Event Discussions (Chat/Forum for Event)
   URL: community/api/events/discussion/<event_id>
   Method: POST

Headers: Authorization: Bearer <token>, Content-type: application/json
Description: Post a discussion message in an event's forum.
Request Body:

```
{
 "message": "string"
}
```

Response:

```
{
 "discussion_id": "string",
 "message": "Discussion added"
}
```

10. Event Attendance
    URL: community/api/events/attendees/<event_id>
    Method: GET
    Headers: Authorization: Bearer <token>
    Description: Get list of registered attendees for an event.
    Response:

```
[
 {
  "user_id": "int",
  "name": "string",
  "headline": "string"
 }
]
```

## ➢ Skill Endorsements & Recommendations

1. Endorse a Skill
   URL: community/api/skills/endorse
   Method: POST
   Headers: Authorization: Bearer <token>, Content-type: application/json
   Description: Endorse a skill on another user's profile.
   Request Body:

```
{
 "user_id": "string",      // person receiving the endorsement
 "skill_id": "string"      // e.g., Python, ReactJS
}
```

Response:

```
{
 "message": "Skill endorsed successfully",
 "endorsement_id": "string",
 "skill_id": "string",
 "endorsed_by": "string"
}
```

2. Get Endorsements for a User

   URL: community/api/skills/endorsements/<user_id>

   Method: GET

   Headers: Content-type: application/json

   Description: Get all skill endorsements for a user.

   Response:

```
[
 {
  "skill_id": "string",
  "skill_name": "string",
  "endorsement_count": "int",
  "endorsed_by": [
   {"user_id": "string", "name": "string"}
  ]
 }
]
```

3. Remove an Endorsement

   URL: community/api/skills/remove_endorsement

   Method: POST

   Headers: Authorization: Bearer <token>

   Description: Remove an endorsement you gave.

   Request Body:

```
{
 "endorsement_id": "string"
}
```

   Response:

```
{
 "message": "Endorsement removed successfully"
}
```

4. Leave Recommendation

   URL: community/api/recommendations/create

   Method: POST

   Headers: Authorization: Bearer <token>, Content-type: application/json

   Description: Employers leave professional recommendations for users.

   Request Body:

```
{
 "user_id": "string",     // candidate being recommended
 "employer_id": "string",  // company/HR account
 "relationship": "string", // e.g., 'Manager', 'HR Recruiter'
 "recommendation_text": "string"
}
```

   Response:

```
{
 "recommendation_id": "string",
 "message": "Recommendation submitted successfully"
}
```

5. Get User Recommendations

   URL: community/api/recommendations/<user_id>

   Method: GET

   Headers: Content-type: application/json

   Description: Fetch all recommendations for a given user.

   Response:

```
[
 {
  "recommendation_id": "string",
  "employer": {
   "employer_id": "string",
   "company_name": "string",
   "contact_name": "string"
  },
  "relationship": "string",
  "recommendation_text": "string",
  "created_at": "datetime"
 }
]
```

6. Employer Verify/Withdraw Recommendation
   URL: community/api/recommendations/manage
   Method: POST
   Headers: Authorization: Bearer <token>, Content-type: application/json
   Description: Employers can update or withdraw recommendations.
   Request Body:
   {
    "recommendation_id": "string",
    "action": "update | withdraw",
    "updated_text": "string (optional)"
   }
   Response:
   {
    "message": "Recommendation updated/withdrawn successfully"
   }

## ➢ Content Sharing & Blogs

1. Create a Blog/Article
   URL: community/api/blogs/create
   Method: POST
   Headers: Authorization: Bearer <token>, Content-type: application/json
   Description: User creates a blog/article.
   Request Body:
   {
    "title": "string",
    "content": "string",
    "tags": ["string"],
    "visibility": "public | connections_only"
   }
   Response:
   {
    "blog_id": "string",
    "message": "Blog created successfully"
   }

2. Get All Blogs (Feed)
   URL: community/api/blogs/feed
   Method: GET
   Headers: Authorization: Bearer <token>
   Description: Fetch latest blogs/posts for a user feed (based on connections + public posts).
   Response:

```
[
 {
  "blog_id": "string",
  "title": "string",
  "author": {
    "user_id": "string",
    "name": "string",
    "profile_pic": "string"
  },
  "tags": ["string"],
  "content_preview": "string",
  "created_at": "datetime"
 }
]
```

3. Get Single Blog
   URL: community/api/blogs/<blog_id>
   Method: GET
   Headers: Content-type: application/json
   Description: Fetch a full blog/article by ID.
   Response:

```
{
 "blog_id": "string",
 "title": "string",
 "content": "string",
 "tags": ["string"],
 "author": {
  "user_id": "string",
  "name": "string",
  "profile_pic": "string"
 },
 "likes": 15,
 "comments_count": 5,
```

"created_at": "datetime"
}

4. Update Blog
   URL: community/api/blogs/update/<blog_id>
   Method: PUT
   Headers: Authorization: Bearer <token>, Content-type: application/json
   Description: Update an existing blog/article.
   Request Body:
   {
   "title": "string (optional)",
   "content": "string (optional)",
   "tags": ["string"] (optional)
   }
   Response:
   {
   "message": "Blog updated successfully"
   }

5. Delete Blog
   URL: community/api/blogs/delete/<blog_id>
   Method: DELETE
   Headers: Authorization: Bearer <token>
   Description: Delete a blog/article.
   Response:
   {
   "message": "Blog deleted successfully"
   }

6. Like/Unlike a Blog
   URL: community/api/blogs/like
   Method: POST
   Headers: Authorization: Bearer <token>, Content-type: application/json
   Description: Like or unlike a blog.
   Request Body:
   {
   "blog_id": "string",
   "action": "like | unlike"

}
Response:
{
"message": "Blog liked/unliked successfully"
}

7. Comment on Blog
URL: community/api/blogs/comment
Method: POST
Headers: Authorization: Bearer <token>, Content-type: application/json
Description: Add a comment on a blog/article.
Request Body:
{
"blog_id": "string",
"comment_text": "string"
}
Response:
{
"comment_id": "string",
"message": "Comment added successfully"
}

## ➢ Mentorship Program

1. Register as Mentor
URL: community/api/mentorship/register-mentor
Method: POST
Headers: Authorization: Bearer <token>, Content-type: application/json
Description: A professional registers as a mentor.
Request Body:
{
"expertise": ["string"], // e.g., ["Software Engineering", "Data Science"]
"experience_years": "number",
"available_slots": "number", // max mentees
"bio": "string"
}

Response:

```
{
 "mentor_id": "string",
 "message": "Mentor profile created successfully"
}
```

2. Browse Available Mentors

   URL: community/api/mentorship/mentors

   Method: GET

   Headers: Authorization: Bearer <token>

   Description: Job seekers browse available mentors.

   Query Params (optional):

   ?expertise=Data Science&location=India

   Response:

```
[
 {
  "mentor_id": "string",
  "name": "string",
  "expertise": ["string"],
  "experience_years": "number",
  "bio": "string",
  "available_slots": "number"
 }
]
```

3. Request Mentorship

   URL: community/api/mentorship/request

   Method: POST

   Headers: Authorization: Bearer <token>, Content-type: application/json

   Description: A job seeker requests mentorship from a mentor.

   Request Body:

```
{
 "mentor_id": "string",
 "mentee_message": "string"   // short intro or request note
}
```

   Response:

```
{
 "request_id": "string",
 "message": "Mentorship request sent"
```

}

4. Mentor Accept/Reject Request
   URL: community/api/mentorship/respond
   Method: POST
   Headers: Authorization: Bearer <token>, Content-type: application/json
   Description: Mentor accepts or rejects a mentorship request.
   Request Body:
   {
    "request_id": "string",
    "action": "accept | reject"
   }
   Response:
   {
    "message": "Request accepted/rejected successfully"
   }

5. Active Mentorship Sessions
   URL: community/api/mentorship/sessions
   Method: GET
   Headers: Authorization: Bearer <token>
   Description: Fetch active mentorship sessions (mentor ↔ mentee connections).
   Response:
   [
    {
     "session_id": "string",
     "mentor": {
      "mentor_id": "string",
      "name": "string"
     },
     "mentee": {
      "mentee_id": "string",
      "name": "string"
     },
     "status": "active | completed",
     "started_at": "datetime"
    }
   ]

6. End Mentorship Session
   URL: community/api/mentorship/end/<session_id>
   Method: POST
   Headers: Authorization: Bearer <token>
   Description: Mentor or mentee can end the mentorship session.
   Response:
   {
    "message": "Mentorship session ended successfully"
   }

7. Rate & Review Mentor
   URL: community/api/mentorship/review
   Method: POST
   Headers: Authorization: Bearer <token>, Content-type: application/json
   Description: Mentee rates and reviews mentor after session.
   Request Body:
   {
    "mentor_id": "string",
    "rating": "number (1-5)",
    "review": "string"
   }
   Response:
   {
    "message": "Review submitted successfully"
   }

Submitted by **Abin Santhosh**