

|  |  |
| --- | --- |
| COMSATS Logo PNG File | **COMSATS UNIVERSITY ISLAMABAD**  **VEHARI CAMPUS** |

|  |  |
| --- | --- |
| **Submitted by:** | M. Fawad Saqlain |
| **Roll no:** | FA22-BSE-031 |
| **Submitted to:** | Ma’am Yasmeen Jana |
| **Course:** | Advance Web Technologies |
| **Project Name:** | CVUR Portal offerings reviews and course recommendation portal |

# Review App — REST API Design & Implementation Report

## A. Design (8 marks)

### 1. Problem statement & Requirements

This project implements a course review platform for CUI Vehari students. The goal is to let enrolled students securely submit anonymous course evaluations for specific "offerings" (course + teacher + section + term), allow limited editing, and provide admin tools to manage academic terms and the set of offerings. The API is implemented with Express and MongoDB (Mongoose).

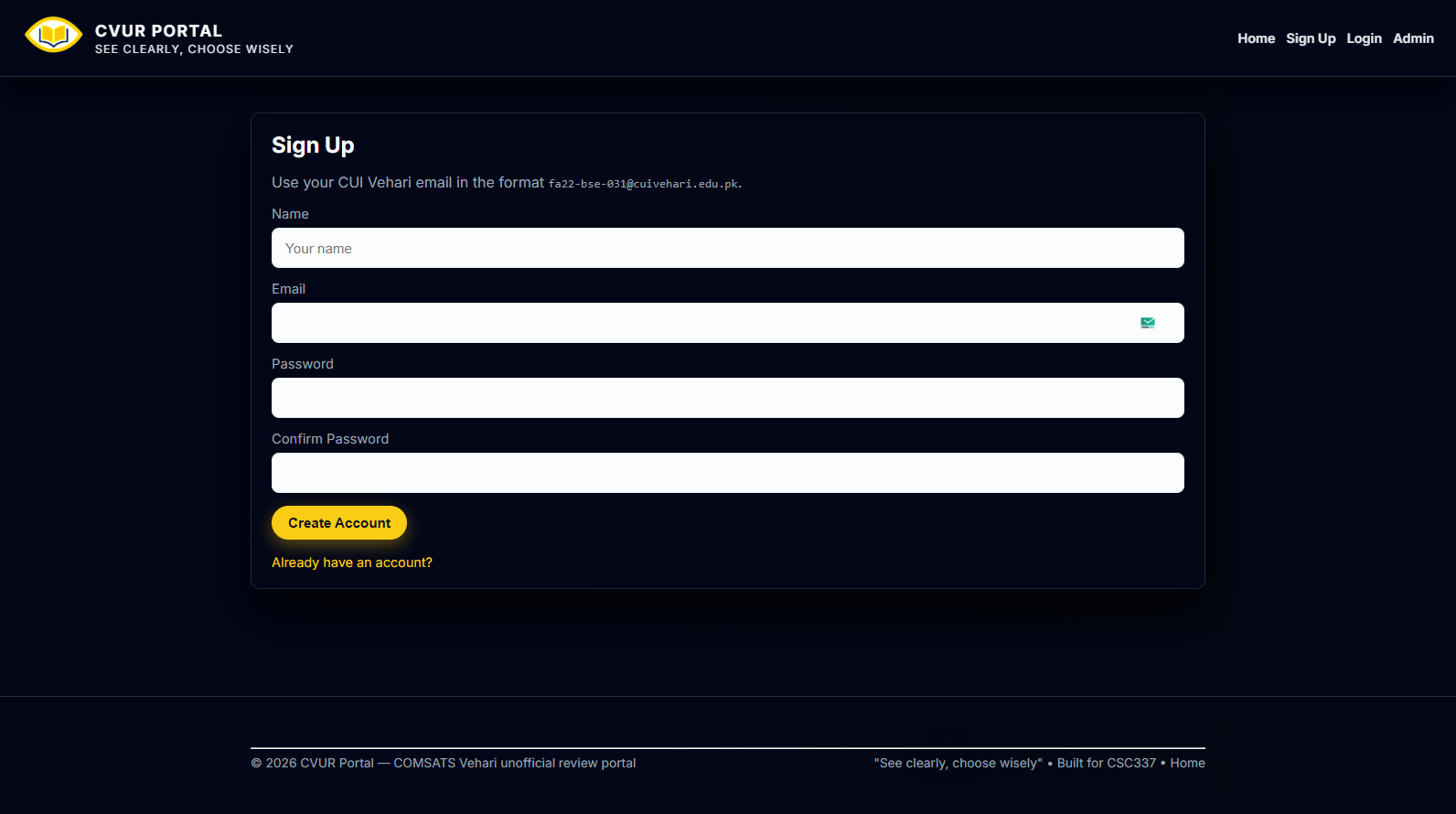
#### Primary goals

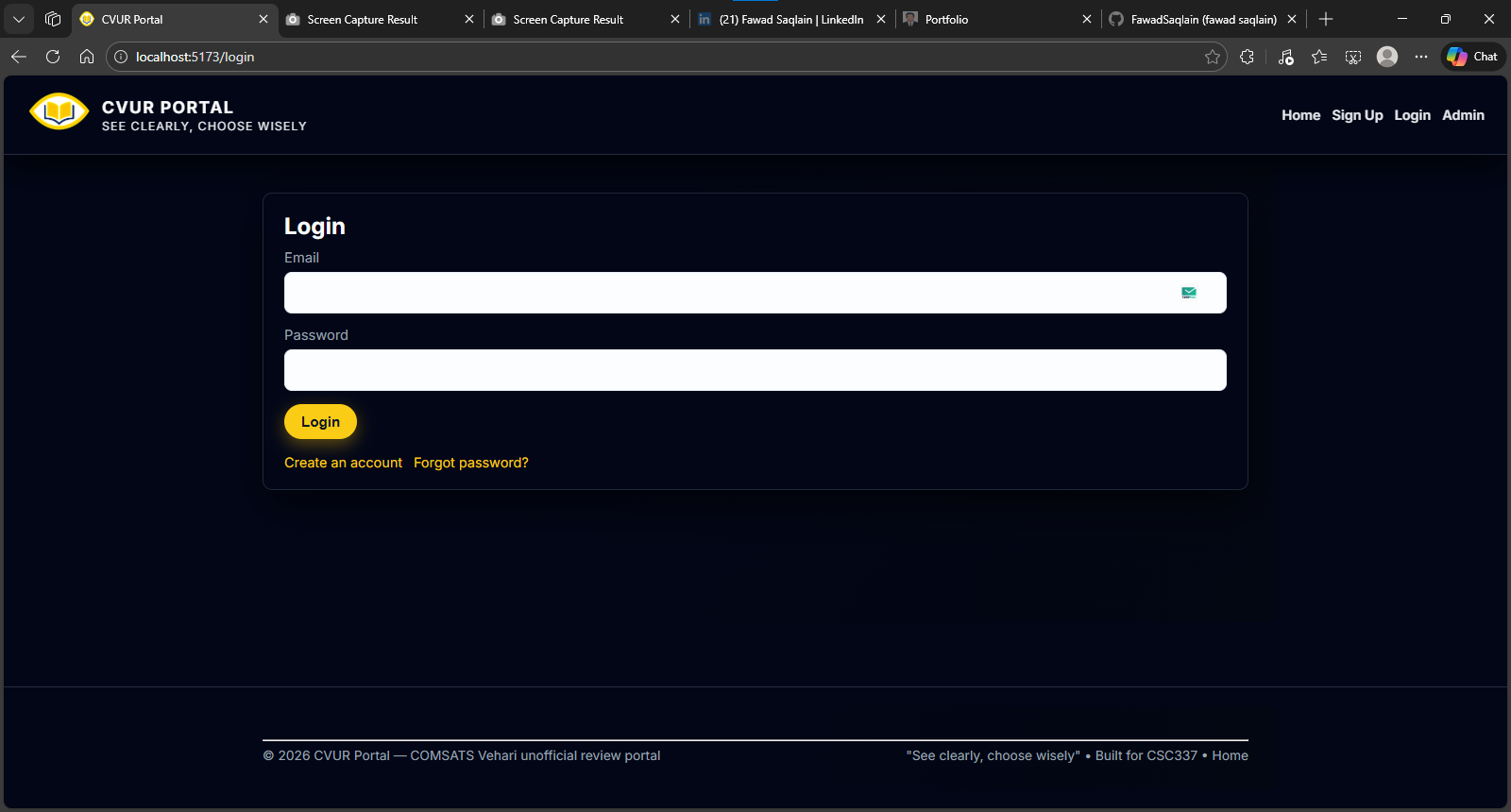
* Secure student onboarding using CUI Vehari email verification (OTP) and allow users to complete a minimal profile required to participate.
* Allow students to submit one anonymous rating per offering. Ratings must include an overall rating (1-5) and obtained marks; optional free-text comments are supported.
* Provide admin capabilities to manage Terms and Offerings: create, update, delete, activate (promote) terms and to bulk import offerings from XLSX file.
* Generate and store aggregated rating summaries for closed terms during promotion so historical summaries are available without heavy runtime computation.

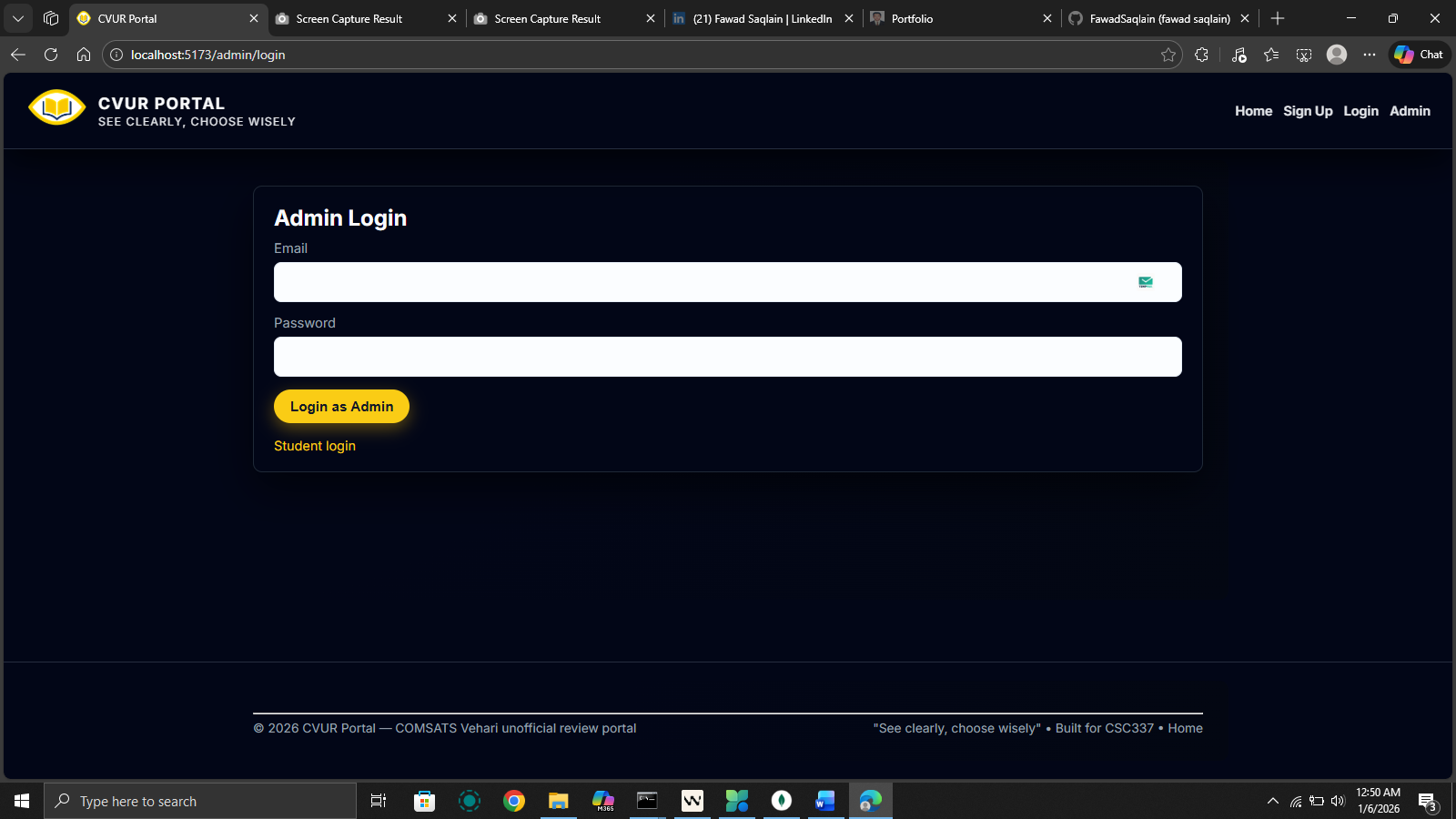
#### Important functional requirements

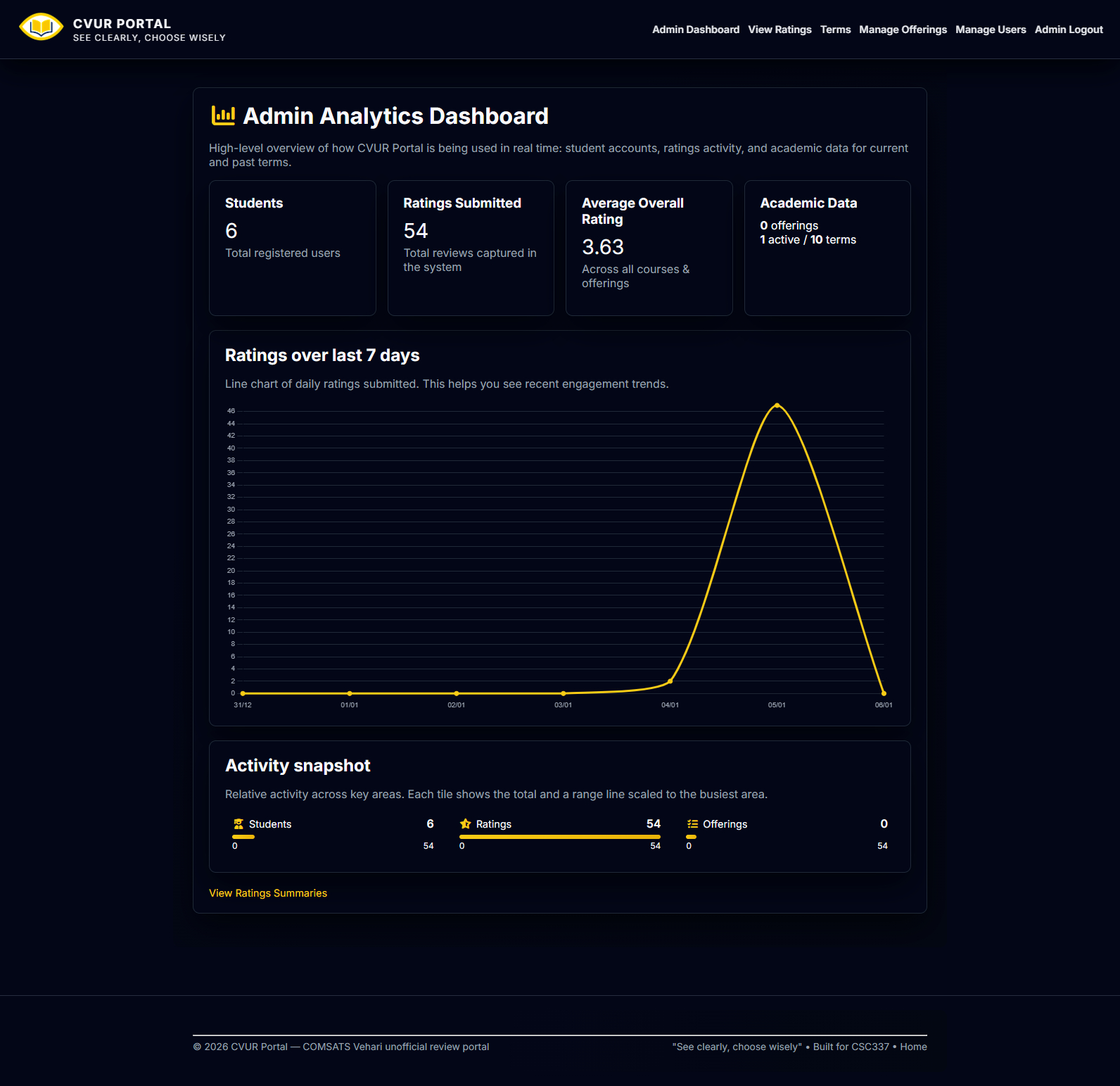
* Signup requires a CUI Vehari student email in the format ^(fa|sp)(\d{2})-([a-z]{2,4})-(\d{3})@cuivehari\.edu\.pk$. The controller enforces this and computes the user's intake and a derived semesterNumber from the email and current date.
* Signup uses a 6-digit OTP (stored hashed in VerificationToken) that expires in 15 minutes; verification activates the user account.
* Login returns a JWT and sets an HttpOnly cookie; /api/auth/me returns the populated user (department/program auto-resolved via an email->program map when missing).
* Profile completion supports uploading an ID card image (express-fileupload). Only images (png/jpg/gif/webp) up to 1MB are accepted; after completion profileComplete is set to true.
* Ratings: POST /ratings requires offering, overallRating (1-5), and obtainedMarks. Ratings are enforced anonymous server-side (anonymized = true), and a unique index prevents more than one rating per student+offering.
* Edit rules: students can only update their own rating and only within the term activation time edit window (controller checks creation date). Editing is closed when the offering's term is no longer active (term is promoted by admin).
* Admin routes are protected by adminAuth/adminRequire. /admin/terms/:id/promote activates the selected term, optionally sets start/end dates, creates the subsequent term, and generates stored rating summaries for the previously active term.
* Offerings upload supports parsing XLSX (ExcelJS) to create Courses, Teachers, Offerings and Departments/Programs. The XLSX uploader accepts common header names (code, title, teacher, department, program, semesterNumber, section) and does in-file dedupe before DB writes.
* Offerings have a unique index on { course, teacher, term, section } to avoid duplicates. Deletion endpoints return JSON for API clients and redirect for HTML form flows.
* Error handling: controllers return a consistent JSON envelope { success: true|false, data:..., error:... } and proper HTTP status codes (400/401/403/404/409/500) according to the failure type.

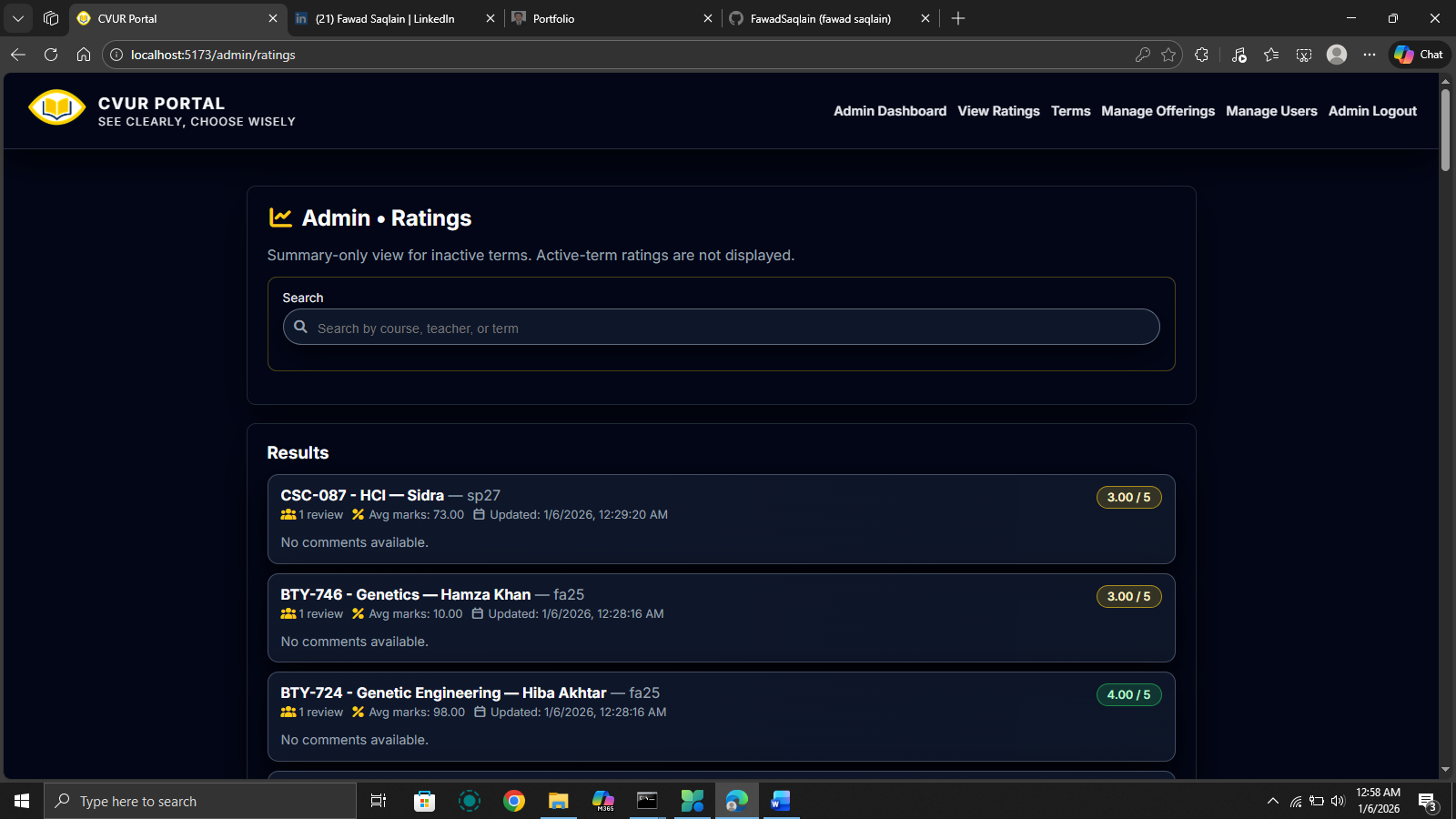
### 2. Screen Shots:

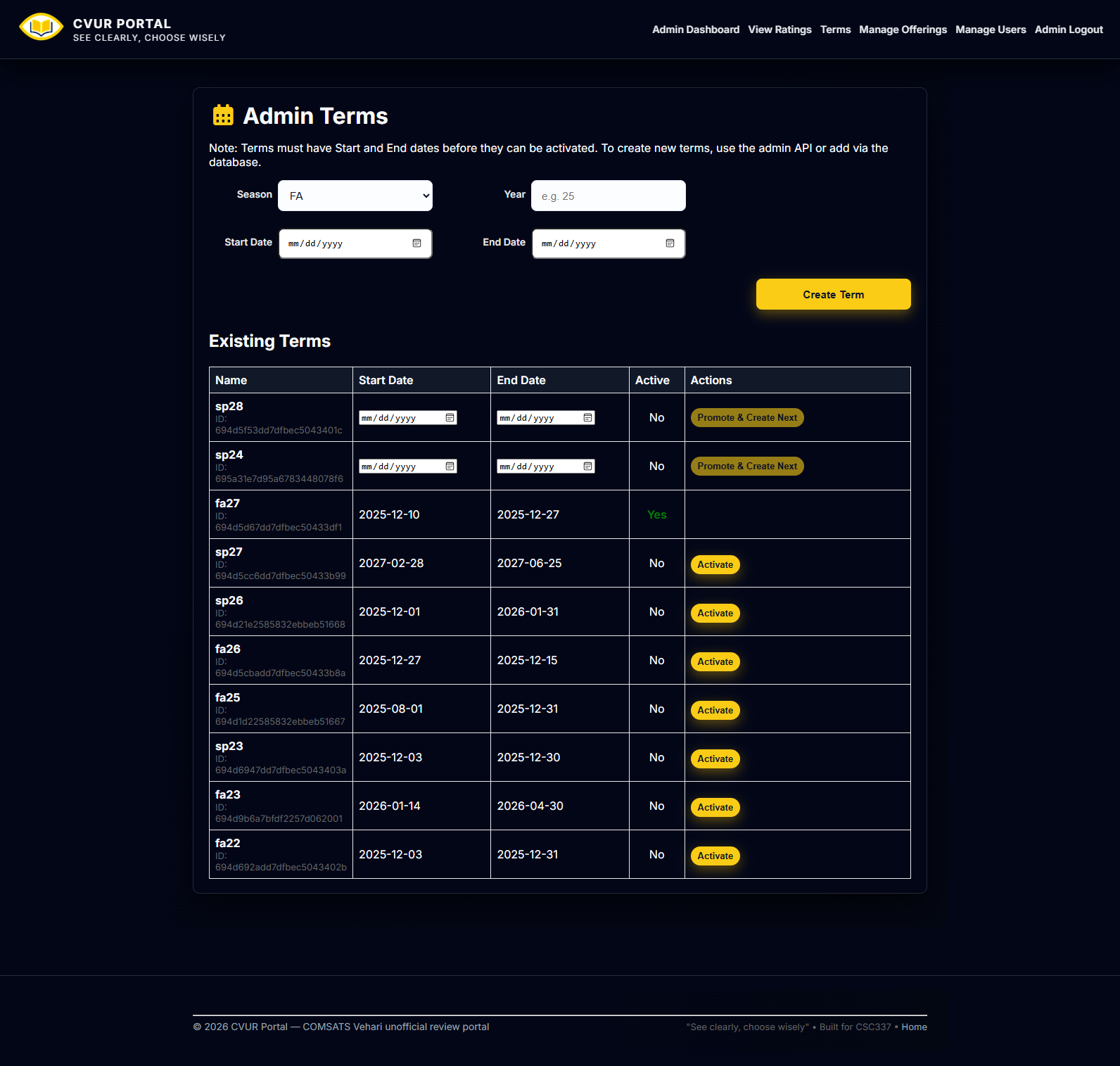


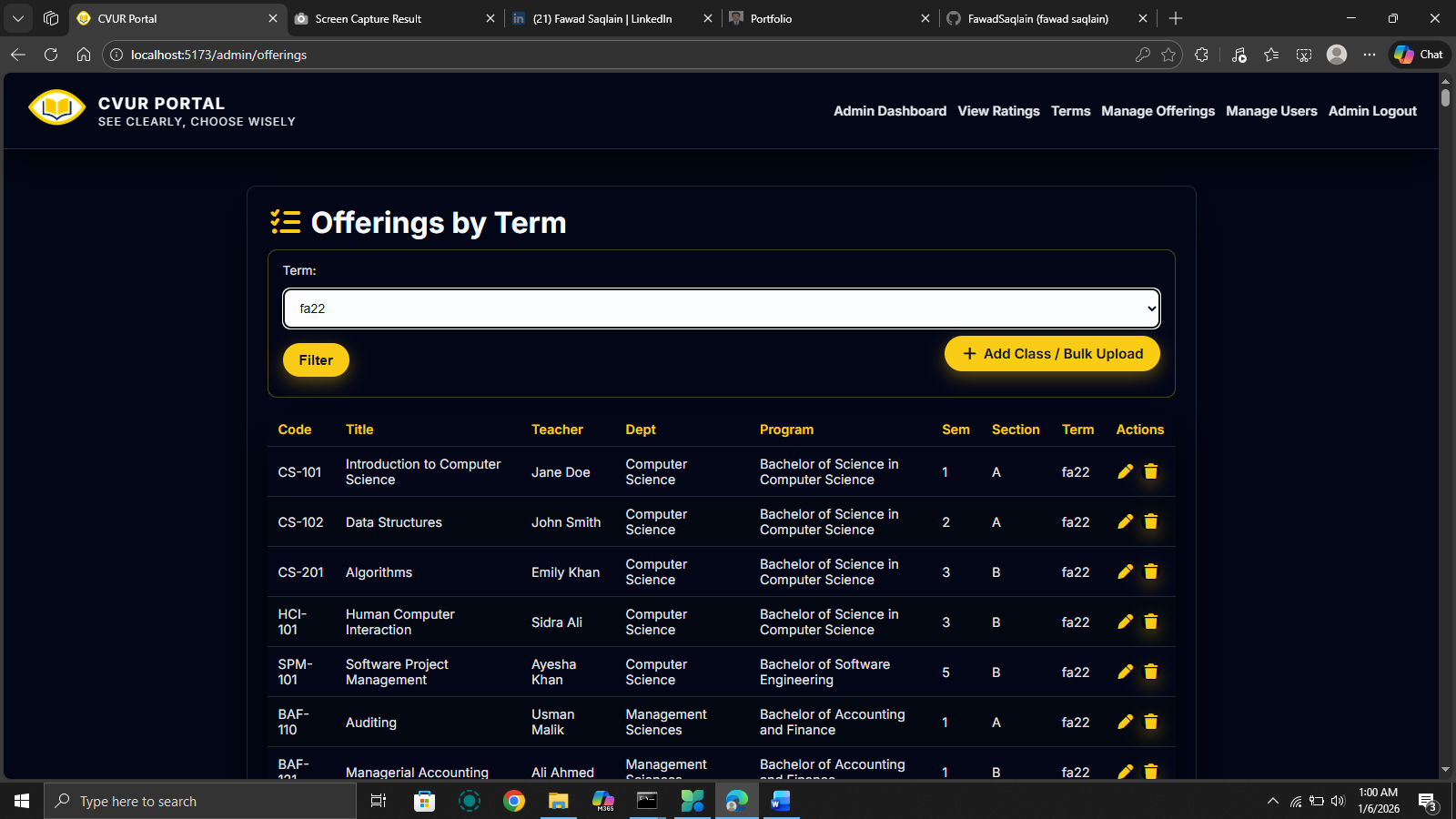


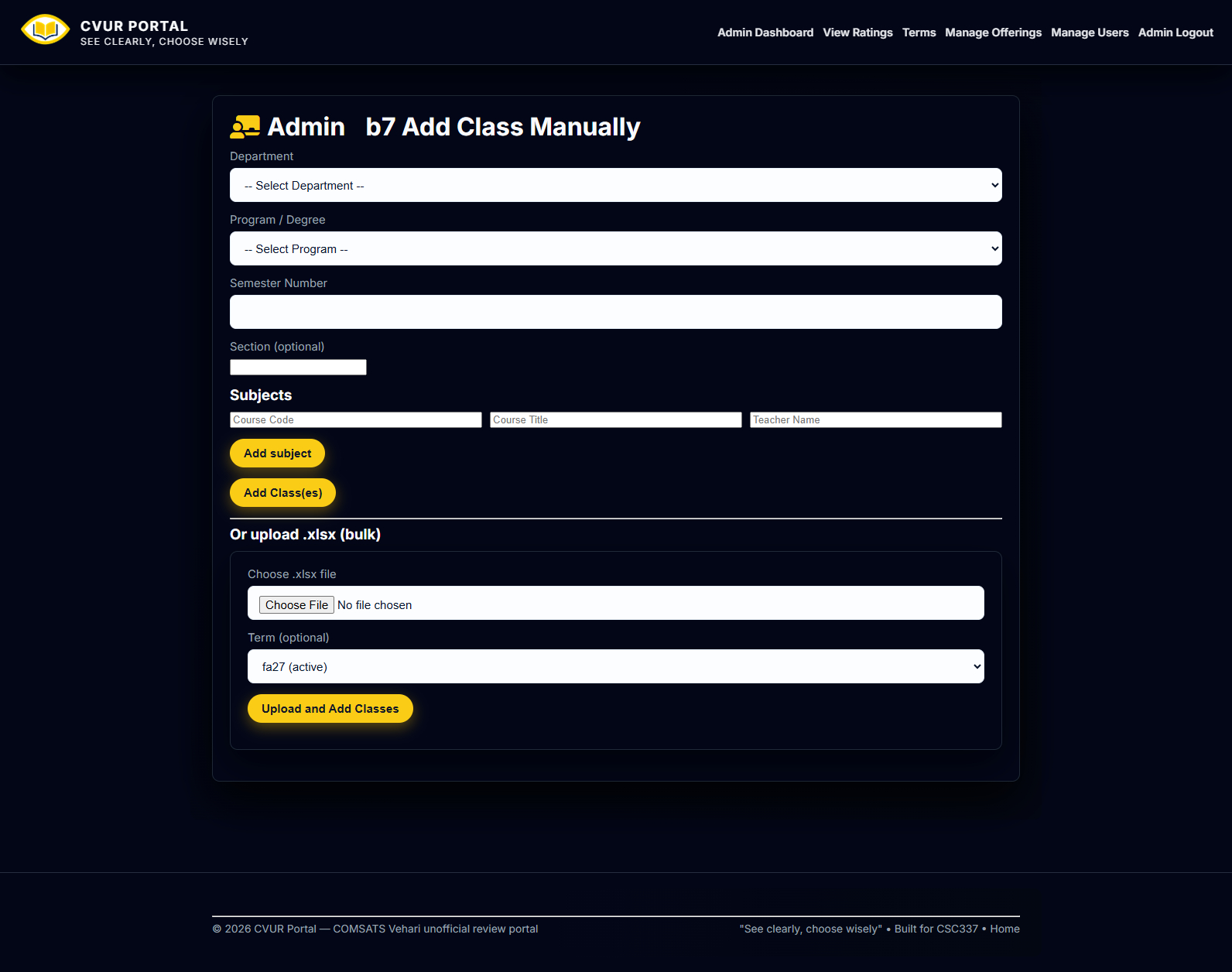


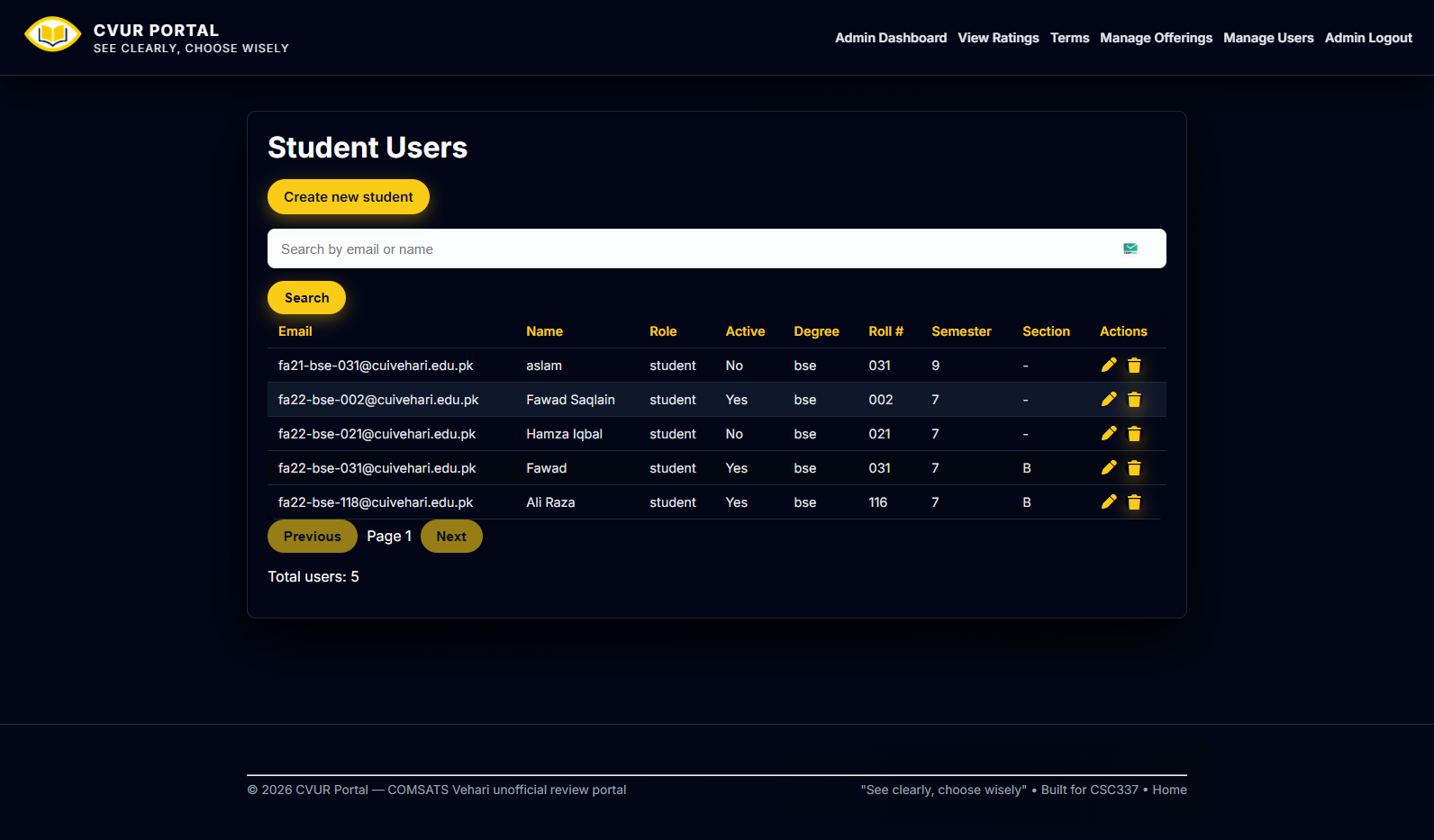




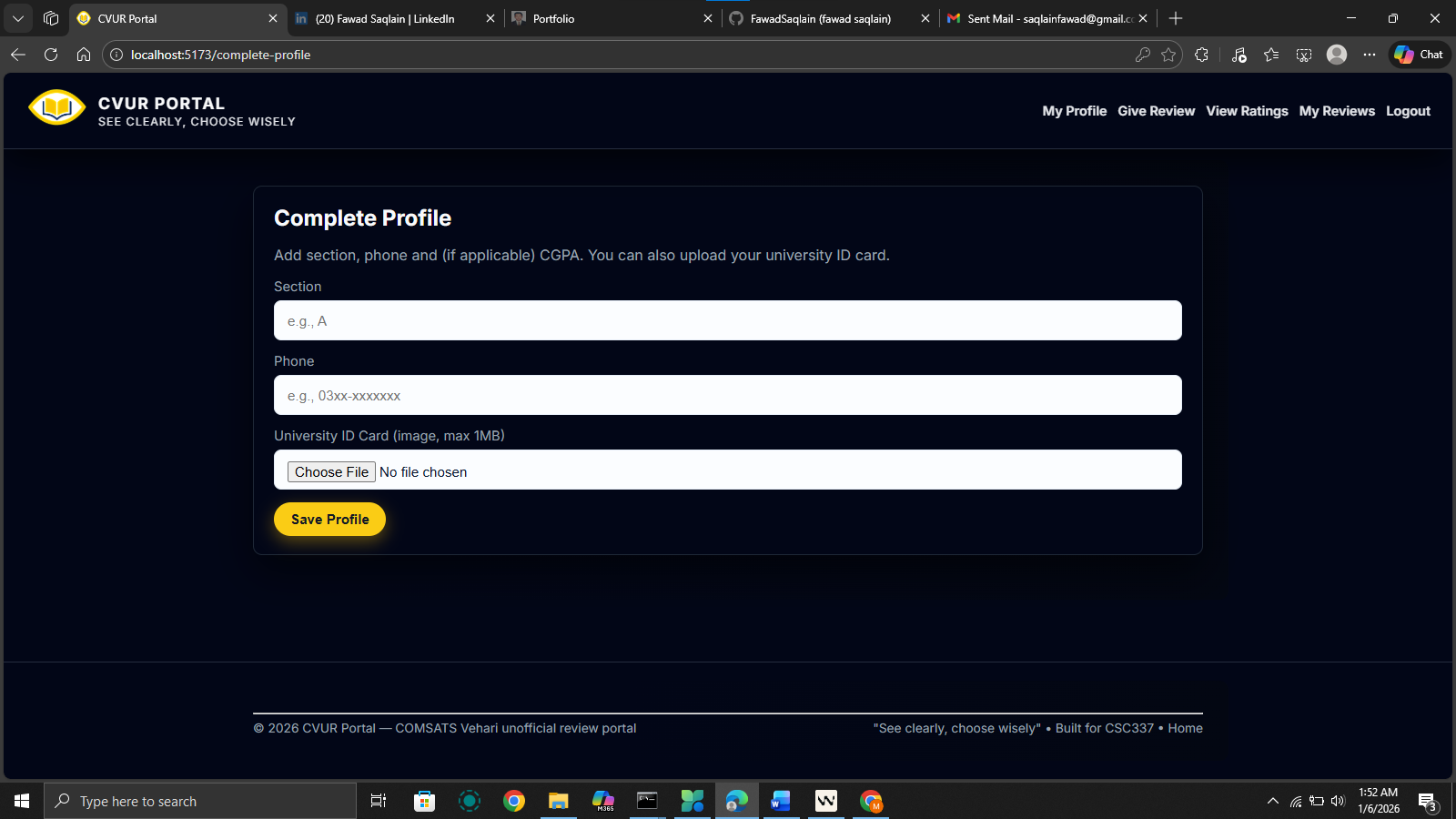


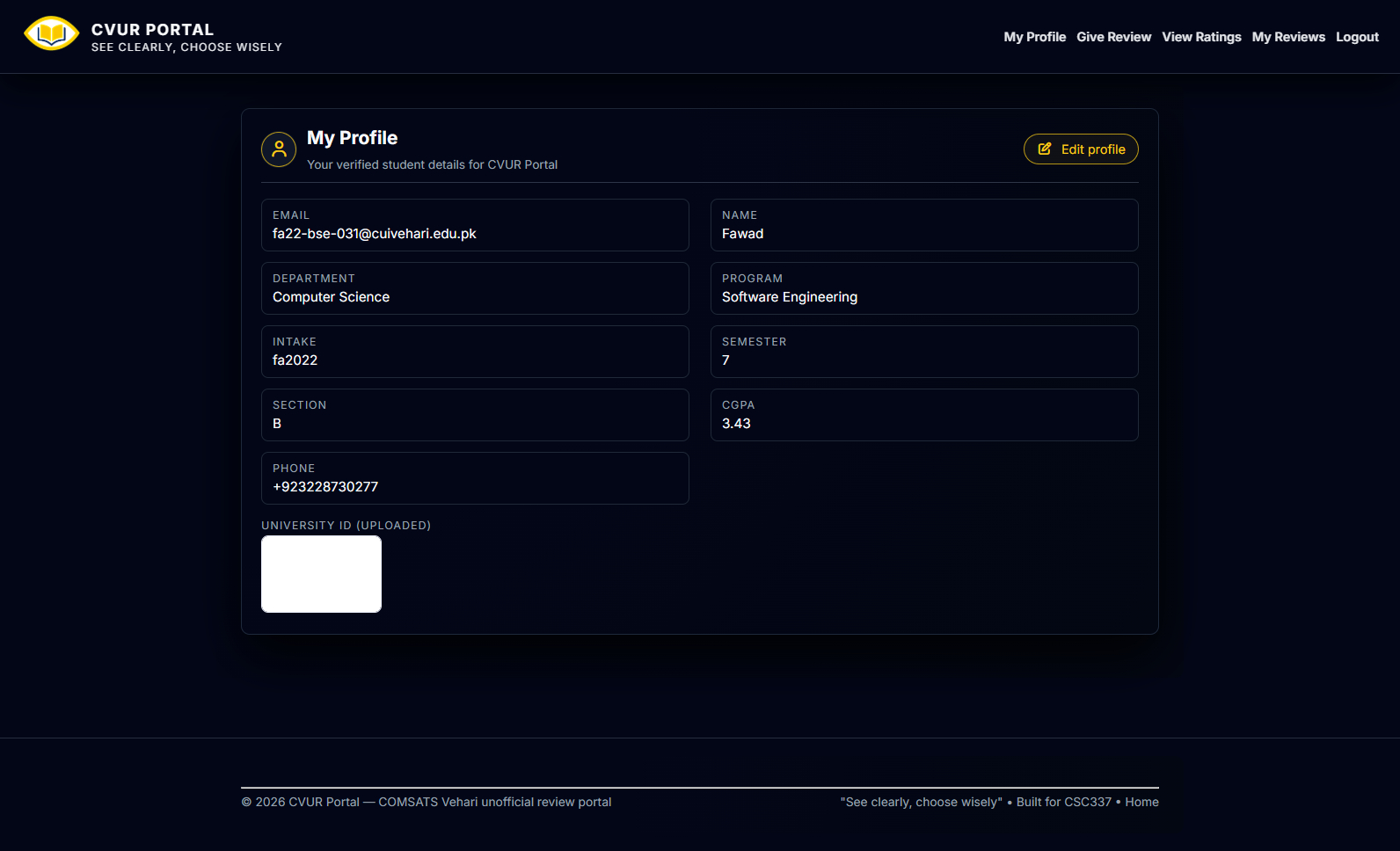


















### 3. Resource model & URIs

Primary resources with example URIs (derived from project):

|  |  |  |
| --- | --- | --- |
| **Resource** | **URI** | **Purpose** |
| User | /api/auth/login, /api/auth/me | Login, profile, complete profile |
| Rating | /ratings (POST), /ratings/:id (PUT/GET) | Create, update, view ratings |
| Offering | /admin/offerings (GET), /admin/offerings/:id (DELETE/PUT) | Admin manage offerings |
| Term | /admin/terms, /admin/terms/:id/promote | Manage academic terms and promotion |
| Public ratings API | /api/ratings, /api/ratings/summary | Public summary data and mobile API |

### 4. HTTP Methods & Status Codes

Use standard RESTful methods and status codes:

* POST — Create (201 Created). Example: POST /ratings -> 201 (created) or 400 (validation error)
* GET — Read (200 OK). Example: GET /ratings?offering=... -> 200 with list
* PUT / PATCH — Update (200 OK). Example: PUT /ratings/:id -> 200 or 403/404
* DELETE — Delete (200 OK or 204 No Content). Example: DELETE /admin/offerings/:id -> 200
* Auth errors: 401 Unauthorized (no/invalid token), 403 Forbidden (insufficient role), 404 Not Found, 409 Conflict (duplicate rating), 500 Server Error

### 5. JSON Structure & Schema

Key resource schemas (Mongoose models in project). Below are simplified JSON schemas (examples) used for validation and API docs.

#### User (simplified)

{

"\_id": "ObjectId",

"email": "string",

"role": "student|admin",

"name": { "first": "string", "last": "string" },

"semesterNumber": 3,

"section": "A",

"phone": "string",

"cgpa": 3.2,

"profileComplete": false

}

#### Rating

{

"\_id": "ObjectId",

"student": "ObjectId (ref User)",

"offering": "ObjectId (ref Offering)",

"overallRating": 1-5,

"obtainedMarks": Number,

"comment": "string",

"anonymized": true,

"createdAt": "ISODate"

}

#### Offering

{

"\_id": "ObjectId",

"course": { "\_id": "ObjectId", "code":"CS-101", "title":"Intro" },

"teacher": { "\_id": "ObjectId", "name":{ "first":"A","last":"B" } },

"department": ObjectId,

"program": ObjectId,

"semesterNumber": Number,

"section": "string",

"term": ObjectId

}

## B. Implementation (14 marks)

### 1. Project structure & code quality

High-level layout (present in the project):

review-app/

├─ app.js # express app setup, middleware, routers

├─ bin/www # server bootstrap

├─ controllers/ # request handlers (auth, rating, timetable, api-ratings)

├─ routes/ # route definitions mapped to controllers

├─ models/ # Mongoose models (User, Rating, Offering, Term, etc.)

├─ middleware/ # auth, adminRequire, loginRequire

├─ public/ # client assets & JS

├─ views/ # ejs templates

├─ scripts/ # seed-admin.js, helper scripts

└─ docs/ # generated reports

Code quality notes:

* Controllers are modular and small, each function handles one route.
* Validation is present in controllers (explicit checks of required fields, ranges).
* Mongoose used for schema enforcement; try/catch blocks around DB operations and logging on errors.
* Server returns consistent JSON format: { success: true|false, data:..., error:... }.

### 2. Endpoints (CRUD)

Representative endpoints with method + purpose + sample request/response.

#### Authentication

Post http:*//127.0.0.1:3000/api/auth/login*

Headers: Content-Type: application/json

Body: {  "email": "fa22-bse-116@cuivehari.edu.pk",  "password": "]=[-p0o9"}

Success 200: { {

    "success": true,

    "data": {

        "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiI2OGZkOTg2NWI5YzdjYTU0ZjdiY2IxMmIiLCJyb2xlIjoic3R1ZGVudCIsImlhdCI6MTc2MTQ1MDUzNywiZXhwIjoxNzYxNDU0MTM3fQ.Puv56ZlTTMt\_C5P8ItFVovyx4\_ikqGA8qR0dg4G9B08",

        "expiresIn": "1h",

        "user": {

            "id": "68fd9865b9c7ca54f7bcb12b",

            "email": "fa22-bse-116@cuivehari.edu.pk",

            "role": "student",

            "profileComplete": false

        }

    }

}

}

#### Create Rating

POST http:*//127.0.0.1:3000/ratings*

Headers: Authorization: Bearer

Body JSON: {

  "offering": "68fddade8b6dde8bc35a74f9",

  "overallRating": 4,

  "obtainedMarks": 72,

  "comment": "Good course, helpful instructor."

}  
Response 201: { success:true, data:{ message: 'Rating submitted' } }

#### Update Rating

PUT http:*//127.0.0.1:3000/ratings/68fdf825b4a85fdf17ae24c7*

Headers: Authorization: Bearer

Body: {"offering": "68fdf825b4a85fdf17ae24c7",

  "overallRating": 2,

  "obtainedMarks": 20,

  "comment": "not good."

}

Response 200: { success:true, data:{ message:'Rating updated' } }

#### List Offerings (admin)

GET http:*//127.0.0.1:3000/admin/offerings*

Authentication: admin session or bearer token

Response 200: renders admin-offerings page (HTML) or JSON if used API style

#### Delete Offering (admin)

DELETE http:*//127.0.0.1:3000/admin/offerings/68fddad98b6dde8bc35a6495*

Headers: Authorization: Bearer

Response 200: {

    "success": true,

    "data": {

        "id": "68fddad98b6dde8bc35a6495"

    }

}

### 3. Validation & Error Handling

* Controller-level checks: required fields, numeric ranges (overallRating 1-5), required offering or term existence.
* Respond with 400 for validation errors, 401 for authentication, 403 for authorization, 404 for missing resources, 409 for duplicates, 500 for server errors.
* Unique constraints handled (duplicate rating -> 409).
* Try/catch blocks and console.error logging; audit events recorded after DB changes.

### 4. Authentication / Security

* JWT used for stateless auth. Admin login sets an HttpOnly cookie `adminToken` as well as returning the token in JSON for API clients.
* Middleware `adminRequire` verifies JWT, checks blacklist (token revocation) and ensures `user.role === 'admin'`.
* Routes protected with `loginRequire` or `adminAuth` as appropriate.
* Passwords hashed with bcrypt; seed script uses same hashing for admin.
* Server uses consistent authorization checks for sensitive operations.

### 5. Database Integration

Mongoose + MongoDB used for persistence. Typical patterns:

* Models defined under `models/` and used with .find(), .findById(), .save(), .deleteOne()
* Indexes and unique constraints for preventing duplicates (e.g., unique rating per student/offering).
* Transactions are not used in the simple flows; complex bulk operations (timetable upload) have dedupe checks and summary reporting.

### 6. Documentation & Testing

### Requests through Postman

Login (admin):

POST http:*//127.0.0.1:3000/api/auth/admin/login*

Content-Type: application/json

Body:

{ "email": "admin@cuivehari.edu.pk", "password": "Secret123" }

Create rating (student):

POST http:*//127.0.0.1:3000/ratings*

Authorization: Bearer <STUDENT\_JWT>

Content-Type: application/json

Body: see rating example above

Delete offering (admin) via curl (PowerShell):

curl -X DELETE "http://127.0.0.1:3000/admin/offerings/68fddad98b6dde8bc35a6487" `

-H "Authorization: Bearer <ADMIN\_JWT>"

### Observed Postman tests (actual requests & responses ran)

Below are the exact requests and JSON responses observed during your Postman testing. I included them verbatim to make the report traceable to your manual tests.

#### Signup (POST /api/auth/signup) — sample response

{

"success": true,

"data": {

"id": "68fd9865b9c7ca54f7bcb12b",

"email": "fa22-bse-116@cuivehari.edu.pk",

"message": "OTP sent to your university email"

}

}

#### Verify signup (POST /api/auth/verify-signup) — sample response

{

"success": true,

"data": {

"message": "Account verified. You may now login."

}

}

#### Login (POST /api/auth/login) — sample response (student)

{

"success": true,

"data": {

"token": "",

"expiresIn": "1h",

"user": {

"id": "68fd9865b9c7ca54f7bcb12b",

"email": "fa22-bse-116@cuivehari.edu.pk",

"role": "student",

"profileComplete": false

}

}

}

#### Get profile (GET /api/auth/me) — sample response

{

"success": true,

"data": {

"user": {

"intake": { "season": "fa", "year": 2022 },

"\_id": "68fd9865b9c7ca54f7bcb12b",

"email": "fa22-bse-116@cuivehari.edu.pk",

"role": "student",

"name": { "first": "Ali Raza" },

"degreeShort": "bse",

"rollNumber": "116",

"semesterNumber": 7,

"profileComplete": false,

"isActive": true,

"department": { "\_id": "68fca7cdb0280827abe7a1bc", "name": "Computer Science" },

"program": { "\_id": "68fcb68983845ac7cd532d61", "name": "Software Engineering" }

}

}

}

#### Logout (GET /api/auth/logout) — request and response

Request body used:

{

"token":" eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiI2OGZkOTg2NWI5YzdjYTU0ZjdiY2IxMmIiLCJyb2xlIjoic3R1ZGVudCIsImlhdCI6MTc2MTQ1NTc3NywiZXhwIjoxNzYxNDU5Mzc3fQ.lQfbW1SZjdCZuLHcg8lWw01YyZy-U9g\_Hix3MPZjiy0"

}

Response:

{"success":true,"data":{"message":"Logged out"}}

#### Admin login (POST /api/auth/admin/login) — sample response

{"success":true,"data":{"token":" eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiI2OGZhMzgzZTQ0ZmIxY2M4Y2ZmNWRmYjgiLCJyb2xlIjoiYWRtaW4iLCJpYXQiOjE3NjE0NTYzOTAsImV4cCI6MTc2MTQ1OTk5MH0.KfzXFzSCBmO2MBpjAFvtbKig0zd\_ycvGmZIb8f5FGpo ","expiresIn":"1h","user":{"id":"68fa383e44fb1cc8cff5dfb8","email":"admin@cuivehari.edu.pk","role":"admin"}}}

#### List terms (GET /admin/terms) — sample response (rendered JSON snapshot)

{"title":"Terms","terms":[{"\_id":"68fd08329054756c2b3b10f5","name":"sp27","isActive":true,"createdAt":"2025-10-25T17:26:10.222Z","updatedAt":"2025-10-25T17:48:00.752Z","\_\_v":0,"endDate":"2027-07-25T00:00:00.000Z","startDate":"2027-01-31T00:00:00.000Z"},{"\_id":"68fca76dad4157e36025c038","name":"sp26","isActive":false,"endDate":"2026-11-25T00:00:00.000Z","startDate":"2026-03-25T00:00:00.000Z","updatedAt":"2025-10-25T17:48:00.701Z"},{"\_id":"68fca87ab0280827abe7cb8d","name":"fa26","isActive":false,"createdAt":"2025-10-25T10:37:46.044Z","updatedAt":"2025-10-25T17:48:00.701Z","\_\_v":0,"endDate":"2026-12-25T00:00:00.000Z","startDate":"2026-01-25T00:00:00.000Z"},{"\_id":"68fca769ad4157e36025c037","name":"fa25","startDate":"2025-08-01T00:00:00.000Z","endDate":"2025-12-31T00:00:00.000Z","isActive":false,"updatedAt":"2025-10-25T17:48:00.701Z"},{"\_id":"68fd0d5039bfa44c9696f704","name":"fa27","isActive":false,"createdAt":"2025-10-25T17:48:00.777Z","updatedAt":"2025-10-25T17:48:00.777Z","\_\_v":0}],"message":null,"error":null}

#### Promote a term (POST /admin/terms/68fd0d5039bfa44c9696f704/promote) — sample request & response

Request body:

{

"startDate": "2028-01-15T00:00:00.000Z",

"endDate": "2028-06-30T00:00:00.000Z"

}

Response (success):

{"success":true,"data":{"activated":{"\_id":"68fd0d5039bfa44c9696f704","name":"fa27","isActive":true,"createdAt":"2025-10-25T17:48:00.777Z","updatedAt":"2025-10-26T06:07:03.196Z","\_\_v":0,"startDate":"2028-01-15T00:00:00.000Z","endDate":"2028-06-30T00:00:00.000Z"},"next":{"name":"sp28","isActive":false,"\_id":"68fdba8721f8144cf483629e","createdAt":"2025-10-26T06:07:03.293Z","updatedAt":"2025-10-26T06:07:03.293Z","\_\_v":0}}}

#### List offerings (GET /admin/offerings) — sample response

{

"title": "Offerings",

"offerings": [

{

"\_id": "68fddad98b6dde8bc35a6487",

"course": {

"\_id": "68fca7cdb0280827abe7a1c8",

"code": "CS-101",

"title": "Introduction to Computer Science",

"createdAt": "2025-10-25T10:34:53.741Z",

"updatedAt": "2025-10-25T10:34:53.741Z",

"\_\_v": 0

},

"teacher": {

"\_id": "68fca7cdb0280827abe7a1ce",

"name": {

"first": "Jane",

"last": "Doe"

},

"createdAt": "2025-10-25T10:34:53.752Z",

"updatedAt": "2025-10-25T10:34:53.752Z",

"\_\_v": 0

},

"section": "A",

"department": {

"\_id": "68fca7cdb0280827abe7a1bc",

"name": "Computer Science",

"createdAt": "2025-10-25T10:34:53.721Z",

"updatedAt": "2025-10-25T10:34:53.721Z",

"\_\_v": 0

},

{

"\_id": "68fca87ab0280827abe7cb8d",

"name": "fa26",

"isActive": false,

"createdAt": "2025-10-25T10:37:46.044Z",

"updatedAt": "2025-10-26T06:07:03.146Z",

"\_\_v": 0,

"endDate": "2026-12-25T00:00:00.000Z",

"startDate": "2026-01-25T00:00:00.000Z"

},

{

"\_id": "68fca769ad4157e36025c037",

"name": "fa25",

"startDate": "2025-08-01T00:00:00.000Z",

"endDate": "2025-12-31T00:00:00.000Z",

"isActive": false,

"updatedAt": "2025-10-26T06:07:03.146Z"

},

{

"\_id": "68fdba8721f8144cf483629e",

"name": "sp28",

"isActive": false,

"createdAt": "2025-10-26T06:07:03.293Z",

"updatedAt": "2025-10-26T06:07:03.293Z",

"\_\_v": 0

}

],

"currentTerm": {

"\_id": "68fd0d5039bfa44c9696f704",

"name": "fa27",

"isActive": true,

"createdAt": "2025-10-25T17:48:00.777Z",

"updatedAt": "2025-10-26T06:07:03.196Z",

"\_\_v": 0,

"endDate": "2028-06-30T00:00:00.000Z",

"startDate": "2028-01-15T00:00:00.000Z"

},

"user": {

"\_id": "68fa383e44fb1cc8cff5dfb8",

"email": "admin@cuivehari.edu.pk",

"passwordHash": "$2b$10$NDhJjeu.oumUL6NmSsokAO3jK6cJEvvjgzDIkCrLtc1mC1PQU.MoG",

"role": "admin",

"name": {

"first": "Site",

"last": "Admin"

},

"profileComplete": false,

"isActive": true,

"createdAt": "2025-10-23T14:14:22.842Z",

"updatedAt": "2025-10-23T14:14:22.842Z",

"\_\_v": 0

}

}

#### Give review page (GET /ratings/give) — sample response

{

"title": "Give Review",

"offerings": [

{

"\_id": "68fddada8b6dde8bc35a668b",

"course": {

"\_id": "68fca7ceb0280827abe7a526",

"code": "BAF-724",

"title": "Auditing",

"createdAt": "2025-10-25T10:34:54.805Z",

"updatedAt": "2025-10-25T10:34:54.805Z",

"\_\_v": 0

},

"teacher": {

"\_id": "68fca7ceb0280827abe7a52c",

"name": {

"first": "Amna",

"last": "Anwar"

},

"createdAt": "2025-10-25T10:34:54.810Z",

"updatedAt": "2025-10-25T10:34:54.810Z",

"\_\_v": 0

},

"section": "B",

"department": "68fca7cdb0280827abe7a236",

"program": "68fca7cdb0280827abe7a23c",

"semesterNumber": 7,

"term": {

"\_id": "68fd0d5039bfa44c9696f704",

"name": "fa27",

"isActive": true,

"createdAt": "2025-10-25T17:48:00.777Z",

"updatedAt": "2025-10-26T06:07:03.196Z",

"\_\_v": 0,

"endDate": "2028-06-30T00:00:00.000Z",

"startDate": "2028-01-15T00:00:00.000Z"

},

"createdAt": "2025-10-26T08:24:58.161Z",

"updatedAt": "2025-10-26T08:24:58.161Z",

"\_\_v": 0

},

"activeTerm": {

"\_id": "68fd0d5039bfa44c9696f704",

"name": "fa27",

"isActive": true,

"createdAt": "2025-10-25T17:48:00.777Z",

"updatedAt": "2025-10-26T06:07:03.196Z",

"\_\_v": 0,

"endDate": "2028-06-30T00:00:00.000Z",

"startDate": "2028-01-15T00:00:00.000Z"

},

"nextTerm": {

"\_id": "68fdba8721f8144cf483629e",

"name": "sp28",

"isActive": false,

"createdAt": "2025-10-26T06:07:03.293Z",

"updatedAt": "2025-10-26T06:07:03.293Z",

"\_\_v": 0

},

"selectedTerm": "68fd0d5039bfa44c9696f704",

"user": {

"\_id": "68f4691b806d5524494c7067",

"email": "fa22-bse-031@cuivehari.edu.pk",

"passwordHash": "$2b$10$CjUtdYd9U6JV3fje9VDcE.aBSnjDa9gvGndW/2RQY4rKlkRvJF.hu",

"role": "student",

"name": {

"first": "fawd",

"last": "saqlain"

},

"profileComplete": true,

"isActive": true,

"createdAt": "2025-10-19T04:29:15.886Z",

"updatedAt": "2025-10-25T11:37:45.451Z",

"\_\_v": 0,

"cgpa": 3.08,

"degreeShort": "bse",

"phone": "894578457839",

"rollNumber": "031",

"section": "B",

"semesterNumber": 7,

"idCardImage": "/uploads/68f4691b806d5524494c7067-1760862855152.jpg",

"department": "68fca7cdb0280827abe7a1bc",

"program": "68fcb68983845ac7cd532d61"

}  
}

#### Create rating (POST /ratings) — sample request & response

Request body:

{

"offering": "68fddade8b6dde8bc35a74f9",

"overallRating": 4,

"obtainedMarks": 72,

"comment": "Good course, helpful instructor."

}

Response:

{"success":true,"data":{"message":"Rating submitted"}}

#### Get ratings for student & term (GET http://127.0.0.1:3000/api/ratings?student

#### =68f4691b806d5524494c7067&termActive=true) — sample response

{

"success": true,

"data": {

"items": [

{

"\_id": "68fdf825b4a85fdf17ae24c7",

"student": {

"\_id": "68f4691b806d5524494c7067",

"email": "fa22-bse-031@cuivehari.edu.pk",

"name": {

"first": "fawd",

"last": "saqlain"

}

},

"offering": {

"\_id": "68fddade8b6dde8bc35a74f9",

"course": {

"\_id": "68fca7d4b0280827abe7ba74",

"code": "BEN-746",

"title": "Microprocessors and Interfacing",

"createdAt": "2025-10-25T10:35:00.207Z",

"updatedAt": "2025-10-25T10:35:00.207Z",

"\_\_v": 0

},

"teacher": {

"\_id": "68fca7d4b0280827abe7ba7a",

"name": {

"first": "Amna",

"last": "Yousaf"

},

"createdAt": "2025-10-25T10:35:00.211Z",

"updatedAt": "2025-10-25T10:35:00.211Z",

"\_\_v": 0

},

"section": "B",

"department": "68fca7cdb0280827abe7a1bc",

"program": "68fca7d3b0280827abe7b7ae",

"semesterNumber": 7,

"term": "68fd0d5039bfa44c9696f704",

"createdAt": "2025-10-26T08:25:02.235Z",

"updatedAt": "2025-10-26T08:25:02.235Z",

"\_\_v": 0

},

"overallRating": 4,

"obtainedMarks": 72,

"comment": "Good course, helpful instructor.",

"anonymized": true,

"answers": [],

"createdAt": "2025-10-26T10:29:57.100Z",

"updatedAt": "2025-10-26T10:29:57.100Z",

"\_\_v": 0

},

{

"\_id": "68fdf2a1b4a85fdf17ae248b",

"student": {

"\_id": "68f4691b806d5524494c7067",

"email": "fa22-bse-031@cuivehari.edu.pk",

"name": {

"first": "fawd",

"last": "saqlain"

}

},

"offering": {

"\_id": "68fddadc8b6dde8bc35a6db1",

"course": {

"\_id": "68fca7d1b0280827abe7afec",

"code": "BCS-724",

"title": "Introduction to Computer Science",

"createdAt": "2025-10-25T10:34:57.317Z",

"updatedAt": "2025-10-25T10:34:57.317Z",

"\_\_v": 0

},

"teacher": {

"\_id": "68fca7d1b0280827abe7aff2",

"name": {

"first": "Maryam",

"last": "Siddiqui"

},

"createdAt": "2025-10-25T10:34:57.323Z",

"updatedAt": "2025-10-25T10:34:57.323Z",

"\_\_v": 0

},

"section": "B",

"department": "68fca7cdb0280827abe7a1bc",

"program": "68fca7d0b0280827abe7ad46",

"semesterNumber": 7,

"term": "68fd0d5039bfa44c9696f704",

"createdAt": "2025-10-26T08:25:00.309Z",

"updatedAt": "2025-10-26T08:25:00.309Z",

"\_\_v": 0

},

"overallRating": 4,

"obtainedMarks": 86,

"comment": "good",

"anonymized": true,

"answers": [],

"createdAt": "2025-10-26T10:06:25.438Z",

"updatedAt": "2025-10-26T10:06:25.438Z",

"\_\_v": 0

}

],

"total": 2,

"page": 1,

"limit": 25,

"aggregates": {

"avgOverall": 0,

"avgMarks": 0,

"count": 0

}

}

#### Update rating (PUT http://127.0.0.1:3000/ratings/68fdf825b4a85fdf17ae24c7) — sample response

Request body example:

{"offering":"68fdf825b4a85fdf17ae24c7","overallRating":2,"obtainedMarks":20,"comment":"not good."}

Response:

{"success":true,"data":{"message":"Rating updated"}}

#### Delete offering (DELETE http://127.0.0.1:3000/admin/offerings/68fddad98b6dde8bc35a6495) — sample response

{"success":true,"data":{"id":"68fddad98b6dde8bc35a6495"}}

#### List ratings with paging & aggregates (GET http://127.0.0.1:3000/api/ratings?page=1&offering=68fca7cdb0280827abe7a1d4&sort=createdAt) — sample response

{

"success": true,

"data": {

"items": [

{

"\_id": "68fdf825b4a85fdf17ae24c7",

"student": {

"\_id": "68f4691b806d5524494c7067",

"email": "fa22-bse-031@cuivehari.edu.pk",

"name": {

"first": "fawd",

"last": "saqlain"

}

},

"offering": {

"\_id": "68fddade8b6dde8bc35a74f9",

"course": {

"\_id": "68fca7d4b0280827abe7ba74",

"code": "BEN-746",

"title": "Microprocessors and Interfacing",

"createdAt": "2025-10-25T10:35:00.207Z",

"updatedAt": "2025-10-25T10:35:00.207Z",

"\_\_v": 0

},

"teacher": {

"\_id": "68fca7d4b0280827abe7ba7a",

"name": {

"first": "Amna",

"last": "Yousaf"

},

"createdAt": "2025-10-25T10:35:00.211Z",

"updatedAt": "2025-10-25T10:35:00.211Z",

"\_\_v": 0

},

"section": "B",

"department": "68fca7cdb0280827abe7a1bc",

"program": "68fca7d3b0280827abe7b7ae",

"semesterNumber": 7,

"term": "68fd0d5039bfa44c9696f704",

"createdAt": "2025-10-26T08:25:02.235Z",

"updatedAt": "2025-10-26T08:25:02.235Z",

"\_\_v": 0

},

"overallRating": 2,

"obtainedMarks": 20,

"comment": "not good.",

"anonymized": true,

"answers": [],

"createdAt": "2025-10-26T10:29:57.100Z",

"updatedAt": "2025-10-26T13:30:10.013Z",

"\_\_v": 0

},

{

"\_id": "68fdf2a1b4a85fdf17ae248b",

"student": {

"\_id": "68f4691b806d5524494c7067",

"email": "fa22-bse-031@cuivehari.edu.pk",

"name": {

"first": "fawd",

"last": "saqlain"

}

},

"offering": {

"\_id": "68fddadc8b6dde8bc35a6db1",

"course": {

"\_id": "68fca7d1b0280827abe7afec",

"code": "BCS-724",

"title": "Introduction to Computer Science",

"createdAt": "2025-10-25T10:34:57.317Z",

"updatedAt": "2025-10-25T10:34:57.317Z",

"\_\_v": 0

},

"teacher": {

"\_id": "68fca7d1b0280827abe7aff2",

"name": {

"first": "Maryam",

"last": "Siddiqui"

},

"createdAt": "2025-10-25T10:34:57.323Z",

"updatedAt": "2025-10-25T10:34:57.323Z",

"\_\_v": 0

},

"section": "B",

"department": "68fca7cdb0280827abe7a1bc",

"program": "68fca7d0b0280827abe7ad46",

"semesterNumber": 7,

"term": "68fd0d5039bfa44c9696f704",

"createdAt": "2025-10-26T08:25:00.309Z",

"updatedAt": "2025-10-26T08:25:00.309Z",

"\_\_v": 0

},

"overallRating": 4,

"obtainedMarks": 86,

"comment": "good",

"anonymized": true,

"answers": [],

"createdAt": "2025-10-26T10:06:25.438Z",

"updatedAt": "2025-10-26T10:06:25.438Z",

"\_\_v": 0

},

{

"\_id": "68fd0d1739bfa44c9696f6e5",

"student": {

"\_id": "68f4691b806d5524494c7067",

"email": "fa22-bse-031@cuivehari.edu.pk",

"name": {

"first": "fawd",

"last": "saqlain"

}

},

"offering": {

"\_id": "68fd087e9054756c2b3b2d0f",

"course": {

"\_id": "68fca7d5b0280827abe7bdc1",

"code": "BES-746",

"title": "Introduction to Environmental Science",

"createdAt": "2025-10-25T10:35:01.199Z",

"updatedAt": "2025-10-25T10:35:01.199Z",

"\_\_v": 0

},

"teacher": {

"\_id": "68fca7d3b0280827abe7b7ba",

"name": {

"first": "Sara",

"last": "Tariq"

},

"createdAt": "2025-10-25T10:34:59.521Z",

"updatedAt": "2025-10-25T10:34:59.521Z",

"\_\_v": 0

},

"section": "B",

"department": "68fca7cfb0280827abe7a5fc",

"program": "68fca7d4b0280827abe7bb10",

"semesterNumber": 7,

"term": "68fca87ab0280827abe7cb8d",

"createdAt": "2025-10-25T17:27:26.258Z",

"updatedAt": "2025-10-25T17:27:26.258Z",

"\_\_v": 0

},

"overallRating": 5,

"obtainedMarks": 100,

"comment": "greate teacher",

"anonymized": true,

"answers": [],

"createdAt": "2025-10-25T17:47:03.223Z",

"updatedAt": "2025-10-25T17:47:03.223Z",

"\_\_v": 0

},

{

"\_id": "68fd0d0339bfa44c9696f6d3",

"student": {

"\_id": "68f4691b806d5524494c7067",

"email": "fa22-bse-031@cuivehari.edu.pk",

"name": {

"first": "fawd",

"last": "saqlain"

}

},

"offering": {

"\_id": "68fd08799054756c2b3b1c91",

"course": {

"\_id": "68fca7d1b0280827abe7afec",

"code": "BCS-724",

"title": "Introduction to Computer Science",

"createdAt": "2025-10-25T10:34:57.317Z",

"updatedAt": "2025-10-25T10:34:57.317Z",

"\_\_v": 0

},

"teacher": {

"\_id": "68fca7d1b0280827abe7aff2",

"name": {

"first": "Maryam",

"last": "Siddiqui"

},

"createdAt": "2025-10-25T10:34:57.323Z",

"updatedAt": "2025-10-25T10:34:57.323Z",

"\_\_v": 0

},

"section": "B",

"department": "68fca7cdb0280827abe7a1bc",

"program": "68fca7d0b0280827abe7ad46",

"semesterNumber": 7,

"term": "68fca87ab0280827abe7cb8d",

"createdAt": "2025-10-25T17:27:21.598Z",

"updatedAt": "2025-10-25T17:27:21.598Z",

"\_\_v": 0

},

"overallRating": 4,

"obtainedMarks": 28,

"comment": "failed me",

"anonymized": true,

"answers": [],

"createdAt": "2025-10-25T17:46:43.420Z",

"updatedAt": "2025-10-25T17:46:43.420Z",

"\_\_v": 0

},

{

"\_id": "68fd07a79054756c2b3b10c3",

"student": {

"\_id": "68f4691b806d5524494c7067",

"email": "fa22-bse-031@cuivehari.edu.pk",

"name": {

"first": "fawd",

"last": "saqlain"

}

},

"offering": {

"\_id": "68fd072e9054756c2b3b0421",

"course": {

"\_id": "68fca7d4b0280827abe7ba74",

"code": "BEN-746",

"title": "Microprocessors and Interfacing",

"createdAt": "2025-10-25T10:35:00.207Z",

"updatedAt": "2025-10-25T10:35:00.207Z",

"\_\_v": 0

},

"teacher": {

"\_id": "68fca7d4b0280827abe7ba7a",

"name": {

"first": "Amna",

"last": "Yousaf"

},

"createdAt": "2025-10-25T10:35:00.211Z",

"updatedAt": "2025-10-25T10:35:00.211Z",

"\_\_v": 0

},

"section": "B",

"department": "68fca7cdb0280827abe7a1bc",

"program": "68fca7d3b0280827abe7b7ae",

"semesterNumber": 7,

"term": "68fca76dad4157e36025c038",

"createdAt": "2025-10-25T17:21:50.681Z",

"updatedAt": "2025-10-25T17:21:50.681Z",

"\_\_v": 0

},

"overallRating": 3,

"obtainedMarks": 30,

"comment": "failed me",

"anonymized": true,

"answers": [],

"createdAt": "2025-10-25T17:23:51.322Z",

"updatedAt": "2025-10-25T17:23:51.322Z",

"\_\_v": 0

},

{

"\_id": "68fd07909054756c2b3b10b2",

"student": {

"\_id": "68f4691b806d5524494c7067",

"email": "fa22-bse-031@cuivehari.edu.pk",

"name": {

"first": "fawd",

"last": "saqlain"

}

},

"offering": {

"\_id": "68fd072f9054756c2b3b0688",

"course": {

"\_id": "68fca7d5b0280827abe7bdc1",

"code": "BES-746",

"title": "Introduction to Environmental Science",

"createdAt": "2025-10-25T10:35:01.199Z",

"updatedAt": "2025-10-25T10:35:01.199Z",

"\_\_v": 0

},

"teacher": {

"\_id": "68fca7d3b0280827abe7b7ba",

"name": {

"first": "Sara",

"last": "Tariq"

},

"createdAt": "2025-10-25T10:34:59.521Z",

"updatedAt": "2025-10-25T10:34:59.521Z",

"\_\_v": 0

},

"section": "B",

"department": "68fca7cfb0280827abe7a5fc",

"program": "68fca7d4b0280827abe7bb10",

"semesterNumber": 7,

"term": "68fca76dad4157e36025c038",

"createdAt": "2025-10-25T17:21:51.312Z",

"updatedAt": "2025-10-25T17:21:51.312Z",

"\_\_v": 0

},

"overallRating": 5,

"obtainedMarks": 10,

"comment": "i got full marks",

"anonymized": true,

"answers": [],

"createdAt": "2025-10-25T17:23:28.815Z",

"updatedAt": "2025-10-25T17:26:33.836Z",

"\_\_v": 0

},

{

"\_id": "68fd07749054756c2b3b10a1",

"student": {

"\_id": "68f4691b806d5524494c7067",

"email": "fa22-bse-031@cuivehari.edu.pk",

"name": {

"first": "fawd",

"last": "saqlain"

}

},

"offering": {

"\_id": "68fd072c9054756c2b3afcdd",

"course": {

"\_id": "68fca7d1b0280827abe7afec",

"code": "BCS-724",

"title": "Introduction to Computer Science",

"createdAt": "2025-10-25T10:34:57.317Z",

"updatedAt": "2025-10-25T10:34:57.317Z",

"\_\_v": 0

},

"teacher": {

"\_id": "68fca7d1b0280827abe7aff2",

"name": {

"first": "Maryam",

"last": "Siddiqui"

},

"createdAt": "2025-10-25T10:34:57.323Z",

"updatedAt": "2025-10-25T10:34:57.323Z",

"\_\_v": 0

},

"section": "B",

"department": "68fca7cdb0280827abe7a1bc",

"program": "68fca7d0b0280827abe7ad46",

"semesterNumber": 7,

"term": "68fca76dad4157e36025c038",

"createdAt": "2025-10-25T17:21:48.421Z",

"updatedAt": "2025-10-25T17:21:48.421Z",

"\_\_v": 0

},

"overallRating": 3,

"obtainedMarks": 78,

"comment": "good",

"anonymized": true,

"answers": [],

"createdAt": "2025-10-25T17:23:00.548Z",

"updatedAt": "2025-10-25T17:23:00.548Z",

"\_\_v": 0

},

{

"\_id": "68fca849b0280827abe7cb69",

"student": {

"\_id": "68f4691b806d5524494c7067",

"email": "fa22-bse-031@cuivehari.edu.pk",

"name": {

"first": "fawd",

"last": "saqlain"

}

},

"offering": {

"\_id": "68fca7d2b0280827abe7b378",

"course": {

"\_id": "68fca7d2b0280827abe7b370",

"code": "BEC-724",

"title": "Mathematics for Economists",

"createdAt": "2025-10-25T10:34:58.438Z",

"updatedAt": "2025-10-25T10:34:58.438Z",

"\_\_v": 0

},

"teacher": {

"\_id": "68fca7d0b0280827abe7ab7e",

"name": {

"first": "Sana",

"last": "Rehman"

},

"createdAt": "2025-10-25T10:34:56.243Z",

"updatedAt": "2025-10-25T10:34:56.243Z",

"\_\_v": 0

},

"section": "B",

"department": "68fca7d1b0280827abe7b0b8",

"program": "68fca7d1b0280827abe7b0be",

"semesterNumber": 7,

"term": "68fca769ad4157e36025c037",

"createdAt": "2025-10-25T10:34:58.444Z",

"updatedAt": "2025-10-25T10:34:58.444Z",

"\_\_v": 0

},

"overallRating": 2,

"obtainedMarks": 99,

"comment": "not good she failed me",

"anonymized": true,

"answers": [],

"createdAt": "2025-10-25T10:36:57.910Z",

"updatedAt": "2025-10-25T17:24:36.128Z",

"\_\_v": 0

},

{

"\_id": "68fca835b0280827abe7cb58",

"student": {

"\_id": "68f4691b806d5524494c7067",

"email": "fa22-bse-031@cuivehari.edu.pk",

"name": {

"first": "fawd",

"last": "saqlain"

}

},

"offering": {

"\_id": "68fca7d0b0280827abe7ac80",

"course": {

"\_id": "68fca7d0b0280827abe7ac74",

"code": "BBA-724",

"title": "Business Research Methods",

"createdAt": "2025-10-25T10:34:56.481Z",

"updatedAt": "2025-10-25T10:34:56.481Z",

"\_\_v": 0

},

"teacher": {

"\_id": "68fca7d0b0280827abe7ac7a",

"name": {

"first": "Usman",

"last": "Saleem"

},

"createdAt": "2025-10-25T10:34:56.488Z",

"updatedAt": "2025-10-25T10:34:56.488Z",

"\_\_v": 0

},

"section": "B",

"department": "68fca7cdb0280827abe7a236",

"program": "68fca7cfb0280827abe7a9a4",

"semesterNumber": 7,

"term": "68fca769ad4157e36025c037",

"createdAt": "2025-10-25T10:34:56.492Z",

"updatedAt": "2025-10-25T10:34:56.492Z",

"\_\_v": 0

},

"overallRating": 4,

"obtainedMarks": 95,

"comment": "grate",

"anonymized": true,

"answers": [],

"createdAt": "2025-10-25T10:36:37.502Z",

"updatedAt": "2025-10-25T10:36:37.502Z",

"\_\_v": 0

}

],

"total": 9,

"page": 1,

"limit": 25,

"aggregates": {

"avgOverall": 3.5555555555555554,

"avgMarks": 60.666666666666664,

"count": 9

}

}

}

Rating summary (GET http:*//127.0.0.1:3000/api/ratings/summary?offering=68fca7d2b0280827abe7b378*

#### ) — sample response

{"success":true,"data":{"status":"stored","summary":"Summary (based on 1 comment(s)): average overall 2.00, average marks 40.00. ...","avgOverall":2,"avgMarks":40,"count":1}}

#### Delete user (DELETE /admin/users/:id) — sample response

{"success":true,"data":{"id":"68fa1c2d16f9f3cd789671a7"}}

**Code files:**

**controllers\adminUsersController.js:**

const { User, Audit } = require('../models');

const bcrypt = require('bcrypt');

*// List users (paginated basic)*

exports.list = async (req, res) => {

  try {

    const q = req.query.q || '';

    const page = Math.max(1, parseInt(req.query.page || '1', 10));

    const limit = 50;

    const filter = q ? { $or: [{ email: new RegExp(q, 'i') }, { 'name.first': new RegExp(q, 'i') }, { 'name.last': new RegExp(q, 'i') }] } : {};

    const total = await User.countDocuments(filter);

    const users = await User.find(filter).sort({ email: 1 }).skip((page - 1) \* limit).limit(limit).lean();

    return res.render('admin-users-list', { title: 'Manage Users', users, page, total, q });

  } catch (err) {

    console.error('adminUsers.list error', err);

    return res.status(500).send('Server error');

  }

};

*// Render new user form*

exports.renderCreate = async (req, res) => {

  return res.render('admin-user-form', { title: 'Create User', user: null });

};

*// helper: parse CUI email like fa22-bse-031@cuivehari.edu.pk*

function parseCuiEmail(email) {

  const m = (email || '').toLowerCase().match(/^(fa|sp)(\d{2})-([a-z]{2,4})-(\d{3})@cuivehari\.edu\.pk$/i);

  if (!m) return null;

  const season = m[1].toLowerCase();

  const yearTwo = m[2];

  const degreeShort = m[3].toLowerCase();

  const rollStr = m[4];

  const intakeYear = 2000 + parseInt(yearTwo, 10);

  return { season, yearTwo, intakeYear, degreeShort, rollStr };

}

function computeSemesterNumberFromIntake(season, intakeYear) {

  const now = new Date();

  const month = now.getMonth() + 1;

  const getPeriodIndexForDate = (d) => {

    const y = d.getFullYear();

    const m = d.getMonth() + 1;

    if (m === 1) return (y - 1) \* 2 + 1;

    if (m >= 2 && m <= 9) return y \* 2 + 0;

    return y \* 2 + 1;

  };

  const currentPeriodIndex = getPeriodIndexForDate(now);

  const intakePeriodIndex = (season === 'fa') ? (intakeYear \* 2 + 1) : (intakeYear \* 2 + 0);

  let semesterNumber = currentPeriodIndex - intakePeriodIndex + 1;

  if (!Number.isFinite(semesterNumber) || semesterNumber < 1) semesterNumber = 1;

  return semesterNumber;

}

*// Create user*

exports.create = async (req, res) => {

  try {

    const { email, password, role, firstName, lastName, degreeShort, rollNumber, intake, semesterNumber, section, cgpa, phone } = req.body;

    if (!email || !password) return res.status(400).send('Email and password required');

    const existing = await User.findOne({ email: email.toLowerCase() });

    if (existing) return res.status(409).send('Email already exists');

    const salt = await bcrypt.genSalt(10);

    const passwordHash = await bcrypt.hash(password, salt);

*// Try to infer intake/degree/roll if not provided*

    let intakeObj = null;

    let degree = degreeShort || null;

    let roll = rollNumber || null;

    let semNum = semesterNumber ? Number(semesterNumber) : null;

    if ((!degree || !roll || !semNum) && email) {

      const parsed = parseCuiEmail(email);

      if (parsed) {

        degree = degree || parsed.degreeShort;

        roll = roll || parsed.rollStr;

        semNum = semNum || computeSemesterNumberFromIntake(parsed.season, parsed.intakeYear);

        intakeObj = { season: parsed.season, year: parsed.intakeYear };

      }

    }

    const user = new User({

      email: email.toLowerCase(),

      passwordHash,

      role: role || 'student',

      name: { first: firstName || '', last: lastName || '' },

      isActive: true,

      degreeShort: degree || undefined,

      rollNumber: roll || undefined,

      intake: intakeObj || undefined,

      semesterNumber: semNum || undefined,

      section: section || undefined,

      cgpa: cgpa ? Number(cgpa) : undefined,

      phone: phone || undefined

    });

    await user.save();

    await Audit.create({ action: 'admin.user.create', actor: req.user.\_id, targetType: 'User', targetId: user.\_id });

    return res.redirect('/admin/users');

  } catch (err) {

    console.error('adminUsers.create error', err);

    return res.status(500).send('Server error');

  }

};

*// Render edit form*

exports.renderEdit = async (req, res) => {

  try {

    const user = await User.findById(req.params.id).lean();

    if (!user) return res.status(404).send('User not found');

    return res.render('admin-user-form', { title: 'Edit User', user });

  } catch (err) {

    console.error('adminUsers.renderEdit error', err);

    return res.status(500).send('Server error');

  }

};

*// Update user*

exports.update = async (req, res) => {

  try {

    const user = await User.findById(req.params.id);

    if (!user) return res.status(404).send('User not found');

    const { email, role, firstName, lastName, isActive, degreeShort, rollNumber, intake, semesterNumber, section, cgpa, phone } = req.body;

    user.email = email ? email.toLowerCase() : user.email;

    user.role = role || user.role;

    user.name = { first: firstName || (user.name && user.name.first) || '', last: lastName || (user.name && user.name.last) || '' };

    user.isActive = typeof isActive !== 'undefined' ? !!isActive : user.isActive;

*// intake/degree/roll/semester parsing*

    let semNum = semesterNumber ? Number(semesterNumber) : user.semesterNumber;

    let intakeObj = user.intake;

    let degree = degreeShort || user.degreeShort;

    let roll = rollNumber || user.rollNumber;

    if ((!degree || !roll || !semNum) && user.email) {

      const parsed = parseCuiEmail(user.email);

      if (parsed) {

        degree = degree || parsed.degreeShort;

        roll = roll || parsed.rollStr;

        semNum = semNum || computeSemesterNumberFromIntake(parsed.season, parsed.intakeYear);

        intakeObj = intakeObj || { season: parsed.season, year: parsed.intakeYear };

      }

    }

    user.degreeShort = degree || user.degreeShort;

    user.rollNumber = roll || user.rollNumber;

    user.intake = intakeObj || user.intake;

    user.semesterNumber = semNum || user.semesterNumber;

    user.section = section || user.section;

    user.cgpa = cgpa ? Number(cgpa) : user.cgpa;

    user.phone = phone || user.phone;

*// change password if provided*

    if (req.body.password && req.body.password.length > 0) {

      const salt = await bcrypt.genSalt(10);

      user.passwordHash = await bcrypt.hash(req.body.password, salt);

    }

    await user.save();

    await Audit.create({ action: 'admin.user.update', actor: req.user.\_id, targetType: 'User', targetId: user.\_id });

    return res.redirect('/admin/users');

  } catch (err) {

    console.error('adminUsers.update error', err);

    return res.status(500).send('Server error');

  }

};

*// Delete user*

exports.delete = async (req, res) => {

  try {

    const user = await User.findById(req.params.id);

    if (!user) return res.status(404).send('User not found');

    await User.deleteOne({ \_id: req.params.id });

    await Audit.create({ action: 'admin.user.delete', actor: req.user.\_id, targetType: 'User', targetId: req.params.id });

*// If the request expects HTML (browser form submit), redirect back to listings for UX.*

    const accept = req.headers && req.headers.accept ? req.headers.accept : '';

    if (accept.indexOf('text/html') !== -1) return res.redirect('/admin/users');

*// otherwise return JSON for API clients (e.g., fetch or Postman)*

    return res.json({ success: true, data: { id: req.params.id } });

  } catch (err) {

    console.error('adminUsers.delete error', err);

    return res.status(500).send('Server error');

  }

};

**controllers\authController.js:**

const bcrypt = require('bcrypt');

const jwt = require('jsonwebtoken');

const crypto = require('crypto');

const { User, VerificationToken } = require('../models');

const Audit = require('../models').Audit;

const Verification = require('../models').VerificationToken;

const emailUtil = require('../utils/email');

const path = require('path');

const fs = require('fs');

const JWT\_SECRET = process.env.JWT\_SECRET || 'change-this-secret';

const JWT\_EXPIRES\_IN = process.env.JWT\_EXPIRES\_IN || '1h';

const blacklist = require('../utils/tokenBlacklist');

*// Signup (student)*

exports.signup = async (req, res) => {

  try {

    const { email, password, name, studentId } = req.body;

    if (!email || !password) return res.status(400).json({ success: false, error: { code: 'ERR\_VALIDATION', message: 'Email and password required' } });

    const normalizedEmail = email.toLowerCase();

    const existing = await User.findOne({ email: normalizedEmail });

    if (existing) return res.status(409).json({ success: false, error: { code: 'ERR\_CONFLICT', message: 'Email already registered' } });

*// Validate CUI Vehari official email format:*

*// Examples: fa22-bse-031@cuivehari.edu.pk or sp22-ben-031@cuivehari.edu.pk*

*// Pattern: ^(fa|sp)(\d{2})-([a-z]{2,4})-(\d{3})@cuivehari\.edu\.pk$*

    const cuiPattern = /^(fa|sp)(\d{2})-([a-z]{2,4})-(\d{3})@cuivehari\.edu\.pk$/i;

    const m = normalizedEmail.match(cuiPattern);

    if (!m) {

      return res.status(400).json({ success: false, error: { code: 'ERR\_INVALID\_EMAIL', message: 'Please use your CUI Vehari email in the format fa22-bse-031@cuivehari.edu.pk or sp22-ben-031@cuivehari.edu.pk' } });

    }

*// parse parts*

    const season = m[1].toLowerCase(); *// 'fa' or 'sp'*

    const yearTwo = m[2]; *// e.g., '22'*

    const degreeShort = m[3].toLowerCase(); *// e.g., 'bse' or 'ben'*

    const rollStr = m[4]; *// '031'*

*// Compute intake year full (assume 20xx unless near future) - use 2000 + two-digit*

    const intakeYear = 2000 + parseInt(yearTwo, 10);

*// Compute semesterNumber as the count of half-year semesters since intake.*

*// We model two periods per year:*

*//  - Period 0: months Feb(2) - Sep(9)*

*//  - Period 1: months Oct(10) - Jan(1) (January belongs to previous year's period 1)*

*// Map intake season to a period index: 'fa' -> period 1 of intake year, 'sp' -> period 0 of intake year.*

    const now = new Date();

    const month = now.getMonth() + 1; *// 1-12*

    const getPeriodIndexForDate = (d) => {

      const y = d.getFullYear();

      const m = d.getMonth() + 1;

      if (m === 1) {

*// Jan belongs to previous year's period 1*

        return (y - 1) \* 2 + 1;

      }

      if (m >= 2 && m <= 9) {

        return y \* 2 + 0;

      }

*// Oct-Dec*

      return y \* 2 + 1;

    };

    const currentPeriodIndex = getPeriodIndexForDate(now);

*// intake period index (fall -> period 1, spring -> period 0)*

    const intakePeriodIndex = (season === 'fa') ? (intakeYear \* 2 + 1) : (intakeYear \* 2 + 0);

    let semesterNumber = currentPeriodIndex - intakePeriodIndex + 1;

    if (!Number.isFinite(semesterNumber) || semesterNumber < 1) semesterNumber = 1;

    const salt = await bcrypt.genSalt(10);

    const passwordHash = await bcrypt.hash(password, salt);

*// create inactive user — will be activated after OTP verification*

*// normalize name into embedded shape expected by User schema*

    let nameObj = null;

    if (typeof name === 'string' && name.trim().length > 0) {

      nameObj = { first: name.trim() };

    } else if (name && typeof name === 'object') {

*// accept { first, last }*

      nameObj = { first: name.first || '', last: name.last || '' };

    }

*// persist intake/degree/roll/semester info on the user record for eligibility checks later*

    const user = new User({

      email: normalizedEmail,

      passwordHash,

      role: 'student',

      name: nameObj,

      studentId,

      isActive: false,

      intake: { season, year: intakeYear },

      degreeShort,

      rollNumber: rollStr,

      semesterNumber

    });

*// attempt to set department/program from email mapping (optional)*

    try {

      const mapUtil = require('../utils/emailProgramMap');

      const resolved = await mapUtil.resolveByEmail(normalizedEmail);

      if (resolved) {

        user.department = resolved.departmentId;

        user.program = resolved.programId;

      }

    } catch (e) {

      console.warn('email->program mapping failed', e && e.message ? e.message : e);

    }

    await user.save();

*// create OTP token (6-digit) and store hashed*

    const otp = Math.floor(100000 + Math.random() \* 900000).toString();

    const tokenHash = crypto.createHash('sha256').update(otp).digest('hex');

    const expiresAt = new Date(Date.now() + 1000 \* 60 \* 15); *// 15 minutes*

  await Verification.create({ email: normalizedEmail, user: user.\_id, tokenHash, purpose: 'signup', campus: 'CUI-Vehari', expiresAt });

*// send OTP to university email*

    try {

      const subject = 'COMSATS Vehari - Account Verification OTP';

      const text = `Your verification code is: ${otp}. It expires in 15 minutes.`;

      console.log('sending signup email to', normalizedEmail,'with OTP', otp);

*// await emailUtil.send({ to: normalizedEmail, subject, text });*

    } catch (sendErr) {

      console.warn('failed to send email, returning OTP in response for dev', sendErr);

*// in dev mode return token; in prod we wouldn't do this*

      await Audit.create({ action: 'user.signup.emailFail', actor: user.\_id, targetType: 'User', targetId: user.\_id, details: { error: String(sendErr) } });

      return res.status(201).json({ success: true, data: { id: user.\_id, email: user.email, otpForDev: otp } });

    }

    await Audit.create({ action: 'user.signup', actor: user.\_id, targetType: 'User', targetId: user.\_id });

    return res.status(201).json({ success: true, data: { id: user.\_id, email: user.email, message: 'OTP sent to your university email' } });

  } catch (err) {

*// handle duplicate key (race) - return 409 conflict*

    if (err && (err.code === 11000 || (err.name === 'MongoServerError' && err.code === 11000))) {

*// Duplicate key (email) - warn and return 409 without full stack to reduce noise*

      const dup = err.keyValue && err.keyValue.email ? err.keyValue.email : (err.message || 'duplicate key');

      console.warn(`signup conflict: duplicate email ${dup}`);

      return res.status(409).json({ success: false, error: { code: 'ERR\_CONFLICT', message: 'Email already registered' } });

    }

    console.error('signup error', err);

    return res.status(500).json({ success: false, error: { code: 'ERR\_INTERNAL', message: 'Server error' } });

  }

};

*// verify signup: POST /verify-signup { email, otp }*

exports.verifySignup = async (req, res) => {

  try {

    const { email, otp } = req.body;

    if (!email || !otp) return res.status(400).json({ success: false, error: { code: 'ERR\_VALIDATION', message: 'Email and otp required' } });

    const normalizedEmail = email.toLowerCase();

    const record = await Verification.findOne({ email: normalizedEmail, purpose: 'signup' }).sort({ createdAt: -1 });

    if (!record) return res.status(400).json({ success: false, error: { code: 'ERR\_INVALID\_TOKEN', message: 'No verification token found' } });

    if (record.expiresAt < new Date()) return res.status(400).json({ success: false, error: { code: 'ERR\_EXPIRED\_TOKEN', message: 'Token expired' } });

    const hash = crypto.createHash('sha256').update(otp).digest('hex');

    if (hash !== record.tokenHash) return res.status(400).json({ success: false, error: { code: 'ERR\_INVALID\_TOKEN', message: 'Invalid OTP' } });

*// activate user*

    await User.findByIdAndUpdate(record.user, { isActive: true });

    await Audit.create({ action: 'user.verifySignup', actor: record.user, targetType: 'User', targetId: record.user });

    return res.status(200).json({ success: true, data: { message: 'Account verified. You may now login.' } });

  } catch (err) {

    console.error('verifySignup error', err);

    return res.status(500).json({ success: false, error: { code: 'ERR\_INTERNAL', message: 'Server error' } });

  }

};

*// Resend signup OTP — POST { email }*

exports.resendSignupOtp = async (req, res) => {

  try {

    const { email } = req.body;

    if (!email) return res.status(400).json({ success: false, error: { code: 'ERR\_VALIDATION', message: 'Email required' } });

    const normalizedEmail = email.toLowerCase();

    const user = await User.findOne({ email: normalizedEmail });

    if (!user) return res.status(404).json({ success: false, error: { code: 'ERR\_NOT\_FOUND', message: 'User not found' } });

*// create new OTP*

    const otp = Math.floor(100000 + Math.random() \* 900000).toString();

    const tokenHash = crypto.createHash('sha256').update(otp).digest('hex');

    const expiresAt = new Date(Date.now() + 1000 \* 60 \* 15); *// 15 minutes*

    await Verification.create({ email: normalizedEmail, user: user.\_id, tokenHash, purpose: 'signup', campus: 'CUI-Vehari', expiresAt });

*// send*

    try {

      const subject = 'COMSATS Vehari - Account Verification OTP (resend)';

      const text = `Your verification code is: ${otp}. It expires in 15 minutes.`;

      await emailUtil.send({ to: normalizedEmail, subject, text });

    } catch (sendErr) {

      console.warn('resend signup email failed', sendErr);

      await Audit.create({ action: 'user.signup.resend.emailFail', actor: user.\_id, targetType: 'User', targetId: user.\_id, details: { error: String(sendErr) } });

      return res.status(200).json({ success: true, data: { message: 'OTP resent (dev)', otpForDev: otp } });

    }

    await Audit.create({ action: 'user.signup.resend', actor: user.\_id, targetType: 'User', targetId: user.\_id });

    return res.status(200).json({ success: true, data: { message: 'OTP resent to your email' } });

  } catch (err) {

    console.error('resendSignupOtp error', err);

    return res.status(500).json({ success: false, error: { code: 'ERR\_INTERNAL', message: 'Server error' } });

  }

};

*// Login (student or admin)*

exports.login = async (req, res) => {

  try {

    const { email, password } = req.body;

    if (!email || !password) return res.status(400).json({ success: false, error: { code: 'ERR\_VALIDATION', message: 'Email and password required' } });

    const user = await User.findOne({ email: email.toLowerCase(), isActive: true });

    if (!user) return res.status(401).json({ success: false, error: { code: 'ERR\_AUTH', message: 'Invalid credentials' } });

    const match = await bcrypt.compare(password, user.passwordHash);

    if (!match) return res.status(401).json({ success: false, error: { code: 'ERR\_AUTH', message: 'Invalid credentials' } });

    const token = jwt.sign({ sub: user.\_id, role: user.role }, JWT\_SECRET, { expiresIn: JWT\_EXPIRES\_IN });

    await Audit.create({ action: 'user.login', actor: user.\_id, targetType: 'User', targetId: user.\_id });

*// indicate whether the user has completed their profile*

    const profileComplete = !!user.profileComplete;

*// also set cookie for server-side rendered pages*

  try { res.cookie('token', token, { httpOnly: true, secure: false, maxAge: 1000 \* 60 \* 60 }); } catch (e) {}

  return res.status(200).json({ success: true, data: { token, expiresIn: JWT\_EXPIRES\_IN, user: { id: user.\_id, email: user.email, role: user.role, profileComplete } } });

  } catch (err) {

    console.error('login error', err);

    return res.status(500).json({ success: false, error: { code: 'ERR\_INTERNAL', message: 'Server error' } });

  }

};

*// GET /api/auth/me - return current user profile (for client-side checks)*

exports.me = async (req, res) => {

  try {

    if (!req.user) return res.status(401).json({ success: false, error: { message: 'Not authenticated' } });

*// re-fetch user to populate department/program names*

    const User = require('../models').User;

    let user = await User.findById(req.user.\_id).populate('department').populate('program');

    if (!user) return res.status(401).json({ success: false, error: { message: 'Not authenticated' } });

*// if department/program missing, try to resolve from email and persist (lazy backfill)*

    try {

      if ((!user.department || !user.program) && user.email) {

        const mapUtil = require('../utils/emailProgramMap');

        const resolved = await mapUtil.resolveByEmail(user.email);

        if (resolved) {

          user.department = user.department || resolved.departmentId;

          user.program = user.program || resolved.programId;

          await user.save();

*// re-populate after save*

          user = await User.findById(user.\_id).populate('department').populate('program');

        }

      }

    } catch (e) {

      console.warn('me: email->program mapping failed', e && e.message ? e.message : e);

    }

    const u = user.toObject ? user.toObject() : user;

    delete u.passwordHash;

    delete u.\_\_v;

    return res.json({ success: true, data: { user: u } });

  } catch (err) {

    console.error('auth.me error', err);

    return res.status(500).json({ success: false, error: { message: 'Server error' } });

  }

};

*// render complete profile page*

exports.renderCompleteProfile = async (req, res) => {

  try {

    let user = null;

*// if authenticated, use req.user, otherwise allow ?email= for dev flow*

    if (req && req.user) user = req.user;

    else {

      const email = req.query.email || '';

      if (email) user = await User.findOne({ email: email.toLowerCase() });

    }

    return res.render('complete-profile', { title: 'Complete Profile', user });

  } catch (err) {

    console.error('renderCompleteProfile error', err);

    return res.status(500).send('Server error');

  }

};

*// POST /api/auth/complete-profile*

exports.completeProfile = async (req, res) => {

  try {

*// prefer authenticated user if available*

    const { section, phone, cgpa } = req.body;

    let user = null;

    if (req && req.user) user = req.user;

    else {

      const email = req.body.email;

      if (!email) return res.status(400).json({ success: false, error: { code: 'ERR\_VALIDATION', message: 'Email required' } });

      user = await User.findOne({ email: email.toLowerCase() });

    }

    if (!user) return res.status(404).json({ success: false, error: { code: 'ERR\_NOT\_FOUND', message: 'User not found' } });

*// only allow profile completion if user is active (and optionally verified)*

  user.section = section || user.section;

  user.phone = phone || user.phone;

*// store CGPA only if semesterNumber > 1*

  if (cgpa && user.semesterNumber && user.semesterNumber > 1) user.cgpa = Number(cgpa);

*// handle optional uploaded university ID image (express-fileupload)*

    try {

      if (req.files && req.files.idCard) {

        const file = req.files.idCard;

*// basic server-side validation*

        const allowed = ['image/png', 'image/jpeg', 'image/jpg', 'image/gif', 'image/webp'];

        if (!allowed.includes(file.mimetype)) {

          return res.status(400).json({ success: false, error: { code: 'ERR\_INVALID\_FILE', message: 'Only image files are allowed (png, jpg, gif, webp)' } });

        }

*// size is already limited by middleware, but double-check*

        const MAX = 1 \* 1024 \* 1024; *// 1MB*

        if (file.size > MAX) {

          return res.status(400).json({ success: false, error: { code: 'ERR\_FILE\_TOO\_LARGE', message: 'File too large (max 1MB)' } });

        }

        const uploadsDir = path.join(\_\_dirname, '..', 'public', 'uploads');

        if (!fs.existsSync(uploadsDir)) fs.mkdirSync(uploadsDir, { recursive: true });

*// remove previous file if present*

        if (user.idCardImage) {

          try {

            const prev = path.join(\_\_dirname, '..', 'public', user.idCardImage.replace(/^\//, ''));

            if (fs.existsSync(prev)) fs.unlinkSync(prev);

          } catch (e) {

            console.warn('failed to remove previous idCard image', e && e.message ? e.message : e);

          }

        }

*// preserve extension if available*

        const originalExt = path.extname(file.name) || '';

        const ext = originalExt || (file.mimetype === 'image/png' ? '.png' : '.jpg');

        const filename = `${user.\_id.toString()}-${Date.now()}${ext}`;

        const dest = path.join(uploadsDir, filename);

*// express-fileupload exposes mv*

        await new Promise((resolve, reject) => {

          file.mv(dest, (err) => err ? reject(err) : resolve());

        });

        user.idCardImage = '/uploads/' + filename;

      }

    } catch (fileErr) {

      console.error('file upload error', fileErr);

      return res.status(500).json({ success: false, error: { code: 'ERR\_FILE', message: 'Failed to process uploaded file' } });

    }

    user.profileComplete = true;

*// try to set department/program from email if not already set*

    try {

      if (!user.department || !user.program) {

        const mapUtil = require('../utils/emailProgramMap');

        const resolved = await mapUtil.resolveByEmail(user.email);

        if (resolved) {

          user.department = user.department || resolved.departmentId;

          user.program = user.program || resolved.programId;

        }

      }

    } catch (e) {

      console.warn('completeProfile: email->program mapping failed', e && e.message ? e.message : e);

    }

    await user.save();

    await Audit.create({ action: 'user.completeProfile', actor: user.\_id, targetType: 'User', targetId: user.\_id });

    return res.status(200).json({ success: true, data: { message: 'Profile updated' } });

  } catch (err) {

    console.error('completeProfile error', err);

    return res.status(500).json({ success: false, error: { code: 'ERR\_INTERNAL', message: 'Server error' } });

  }

};

*// render profile view*

exports.viewProfile = async (req, res) => {

  try {

    let user = null;

    if (req && req.user) user = req.user;

    else {

      const email = req.query.email || '';

      if (!email) return res.status(400).send('Email required');

      const User = require('../models').User;

      user = await User.findOne({ email: email.toLowerCase() }).populate('department').populate('program');

      if (!user) return res.status(404).send('User not found');

    }

*// if department/program missing, try to resolve and save*

    try {

      const User = require('../models').User;

      let fresh = await User.findById(user.\_id).populate('department').populate('program');

      if ((!fresh.department || !fresh.program) && fresh.email) {

        const mapUtil = require('../utils/emailProgramMap');

        const resolved = await mapUtil.resolveByEmail(fresh.email);

        if (resolved) {

          fresh.department = fresh.department || resolved.departmentId;

          fresh.program = fresh.program || resolved.programId;

          await fresh.save();

          fresh = await User.findById(fresh.\_id).populate('department').populate('program');

        }

      }

      user = fresh;

    } catch (e) {

      console.warn('viewProfile: email->program mapping failed', e && e.message ? e.message : e);

    }

    return res.render('profile', { title: 'My Profile', user });

  } catch (err) {

    console.error('viewProfile error', err);

    return res.status(500).send('Server error');

  }

};

*// POST /api/auth/logout*

exports.logout = async (req, res) => {

  try {

*// determine token from Authorization header, request body, or cookies*

    const auth = (req.headers.authorization || req.headers.Authorization || '').split(' ');

    let token = auth.length === 2 && auth[0].toLowerCase() === 'bearer' ? auth[1] : (req.body && req.body.token ? req.body.token : null);

    if (!token && req.cookies) {

      token = req.cookies.adminToken || req.cookies.token || null;

    }

*// if token available, compute expiry and add to blacklist*

    if (token) {

      let payload = null;

      try { payload = jwt.decode(token); } catch (e) { payload = null; }

      let expMs = Date.now() + 3600 \* 1000; *// default 1h*

      if (payload && payload.exp) expMs = payload.exp \* 1000;

      blacklist.add(token, expMs);

      await Audit.create({ action: 'user.logout', actor: (req.user && req.user.\_id) || null, targetType: 'User', targetId: (req.user && req.user.\_id) || null });

    } else {

*// no token provided; still record a logout audit and clear cookies*

      await Audit.create({ action: 'user.logout', actor: (req.user && req.user.\_id) || null, targetType: 'User', targetId: (req.user && req.user.\_id) || null, details: { note: 'logout-no-token' } });

    }

*// clear cookies if present (best-effort)*

    try { res.clearCookie('adminToken'); res.clearCookie('token'); } catch (e) {}

    return res.status(200).json({ success: true, data: { message: 'Logged out' } });

  } catch (err) {

    console.error('logout error', err);

    return res.status(500).json({ success: false, error: { code: 'ERR\_INTERNAL', message: 'Server error' } });

  }

};

*// Admin login: dedicated endpoint that only allows users with role 'admin'*

exports.adminLogin = async (req, res) => {

  try {

    const { email, password } = req.body;

    if (!email || !password) return res.status(400).json({ success: false, error: { code: 'ERR\_VALIDATION', message: 'Email and password required' } });

    const user = await User.findOne({ email: email.toLowerCase(), isActive: true });

    if (!user) return res.status(401).json({ success: false, error: { code: 'ERR\_AUTH', message: 'Invalid credentials' } });

    if (user.role !== 'admin') return res.status(403).json({ success: false, error: { code: 'ERR\_FORBIDDEN', message: 'Admin access required' } });

    const match = await bcrypt.compare(password, user.passwordHash);

    if (!match) return res.status(401).json({ success: false, error: { code: 'ERR\_AUTH', message: 'Invalid credentials' } });

    const token = jwt.sign({ sub: user.\_id, role: user.role }, JWT\_SECRET, { expiresIn: JWT\_EXPIRES\_IN });

    await Audit.create({ action: 'admin.login', actor: user.\_id, targetType: 'User', targetId: user.\_id });

*// set secure cookie for admin session (HttpOnly)*

    try {

      res.cookie('adminToken', token, { httpOnly: true, secure: false, maxAge: 1000 \* 60 \* 60 });

    } catch (e) { */\* ignore cookie set failures \*/* }

    return res.status(200).json({ success: true, data: { token, expiresIn: JWT\_EXPIRES\_IN, user: { id: user.\_id, email: user.email, role: user.role } } });

  } catch (err) {

    console.error('adminLogin error', err);

    return res.status(500).json({ success: false, error: { code: 'ERR\_INTERNAL', message: 'Server error' } });

  }

};

*// Forgot password — generate a one-time token and (placeholder) send via email*

exports.forgotPassword = async (req, res) => {

  try {

    const { email } = req.body;

    if (!email) return res.status(400).json({ success: false, error: { code: 'ERR\_VALIDATION', message: 'Email required' } });

    const user = await User.findOne({ email: email.toLowerCase() });

    if (!user) return res.status(200).json({ success: true, data: { message: 'If the email exists, a reset link will be sent' } });

*// create a token (short-lived) — store hashed in audit/details for now; in production use a dedicated passwordReset collection*

    const token = crypto.randomBytes(20).toString('hex');

    const tokenHash = crypto.createHash('sha256').update(token).digest('hex');

    await Audit.create({ action: 'user.forgotPassword', actor: user.\_id, targetType: 'User', targetId: user.\_id, details: { tokenHash, createdAt: new Date() } });

*// TODO: integrate email provider (Nodemailer/SendGrid). For now, return token in response for manual testing.*

    return res.status(200).json({ success: true, data: { message: 'Password reset token generated (development)', token } });

  } catch (err) {

    console.error('forgotPassword error', err);

    return res.status(500).json({ success: false, error: { code: 'ERR\_INTERNAL', message: 'Server error' } });

  }

};

**controllers\ratingApiController.js:**

const { Rating, User } = require('../models');

*// GET /api/ratings*

*// query: minMarks, minStars, search, sort (createdAt|overallRating|obtainedMarks), order (asc|desc), page, limit*

exports.list = async (req, res) => {

  try {

    const q = {};

  const { minMarks, minStars, search, sort='createdAt', order='desc', page=1, limit=25, student } = req.query;

*// optionally filter ratings to only offerings belonging to the active term when requested*

  const termActiveFilter = req.query.termActive || req.query.activeTerm || null; *// accept either param name*

  if (minMarks) q.obtainedMarks = { $gte: Number(minMarks) };

    if (minStars) q.overallRating = { $gte: Number(minStars) };

  if (student) q.student = student;

*// search in comment or student email*

    let userIds = null;

    if (search) {

      const s = String(search).trim();

*// find users matching email or name*

      const users = await User.find({ $or: [{ 'email': { $regex: s, $options: 'i' } }, { 'name.first': { $regex: s, $options: 'i' } }, { 'name.last': { $regex: s, $options: 'i' } }] }).select('\_id').lean();

      userIds = users.map(u => u.\_id);

      q.$or = [{ comment: { $regex: s, $options: 'i' } }, { student: { $in: userIds } }];

    }

    const sortObj = {};

    sortObj[sort] = order === 'asc' ? 1 : -1;

    const pg = Math.max(1, Number(page) || 1);

    const lim = Math.min(200, Number(limit) || 25);

    const skip = (pg - 1) \* lim;

*// if caller requested only active-term ratings, resolve offering ids for active term*

    if (termActiveFilter && String(termActiveFilter) === 'true') {

      try {

        const Term = require('../models').Term;

        const Offering = require('../models').Offering;

        const activeTerm = await Term.findOne({ isActive: true }).select('\_id').lean();

        if (activeTerm && activeTerm.\_id) {

          const offeringDocs = await Offering.find({ term: activeTerm.\_id }).select('\_id').lean();

          const offeringIds = offeringDocs.map(o => o.\_id);

*// restrict ratings to those offerings*

          q.offering = offeringIds.length ? { $in: offeringIds } : { $in: [] };

        } else {

*// no active term -> no results*

          q.offering = { $in: [] };

        }

      } catch (e) { */\* ignore filter failures \*/* }

    }

    const [items, total, agg] = await Promise.all([

      Rating.find(q).populate('student', 'email name').populate({ path: 'offering', populate: [{ path: 'course' }, { path: 'teacher' }] }).sort(sortObj).skip(skip).limit(lim).lean(),

      Rating.countDocuments(q),

      Rating.aggregate([

        { $match: q },

        { $group: { \_id: null, avgOverall: { $avg: '$overallRating' }, avgMarks: { $avg: '$obtainedMarks' }, count: { $sum: 1 } } }

      ])

    ]);

    const aggResult = (agg && agg[0]) ? { avgOverall: agg[0].avgOverall || 0, avgMarks: agg[0].avgMarks || 0, count: agg[0].count || 0 } : { avgOverall: 0, avgMarks: 0, count: 0 };

    return res.json({ success: true, data: { items, total, page: pg, limit: lim, aggregates: aggResult } });

  } catch (err) {

    console.error('api.ratings.list', err);

    return res.status(500).json({ success: false, error: { message: 'Server error' } });

  }

};

*// admin update - update any rating (admin only middleware should protect)*

exports.adminUpdate = async (req, res) => {

  try {

    const id = req.params.id;

    const rating = await Rating.findById(id);

    if (!rating) return res.status(404).json({ success: false, error: { message: 'Not found' } });

    const { overallRating, obtainedMarks, anonymized, comment } = req.body;

    if (typeof overallRating !== 'undefined') rating.overallRating = Number(overallRating);

    if (typeof obtainedMarks !== 'undefined') rating.obtainedMarks = Number(obtainedMarks);

    if (typeof anonymized !== 'undefined') rating.anonymized = !!anonymized;

    if (typeof comment !== 'undefined') rating.comment = comment;

    await rating.save();

    return res.json({ success: true, data: { message: 'Updated' } });

  } catch (err) {

    console.error('api.ratings.adminUpdate', err);

    return res.status(500).json({ success: false, error: { message: 'Server error' } });

  }

};

*// GET /api/ratings/summary?offering=...*

*// NOTE: This endpoint no longer performs on-demand summarization. It will return a stored*

*// RatingSummary (generated during admin Promote) when available. If the offering belongs to*

*// the active term, the endpoint returns status 'active' (no summary). If the offering is from*

*// a past term and no stored summary exists, the endpoint returns status 'pending' (no runtime*

*// summarization is performed here).*

exports.summary = async (req, res) => {

  console.log('api.ratings.summary called');

  try {

    const offeringId = req.query.offering;

    if (!offeringId)

      return res.status(400).json({ success: false, error: { message: 'offering required' } });

*// Load offering and its term to determine active state*

    const Offering = require('../models').Offering;

    const RatingSummary = require('../models').RatingSummary;

    const offering = await Offering.findById(offeringId).populate('term').lean();

*// If offering not found, return 404*

    if (!offering) return res.status(404).json({ success: false, error: { message: 'Offering not found' } });

    const term = offering.term;

*// If term is present and active, do NOT provide a summary (ratings are live and changing)*

    if (term && term.isActive) {

      return res.json({ success: true, data: { status: 'active', message: 'Offering belongs to active term; no stored summary.' } });

    }

*// Try to fetch a stored summary for this offering+term*

    const termId = term ? term.\_id : null;

    const stored = await RatingSummary.findOne({ offering: offeringId, term: termId }).lean();

    if (stored) {

      return res.json({ success: true, data: { status: 'stored', summary: stored.summary, avgOverall: stored.avgOverall, avgMarks: stored.avgMarks, count: stored.count, updatedAt: stored.updatedAt, createdAt: stored.createdAt } });

    }

*// No stored summary. Return pending status — do not run heavy summarization here.*

*// Provide lightweight aggregates (counts/averages) for display if useful.*

    const ratings = await Rating.find({ offering: offeringId }).select('overallRating obtainedMarks').lean();

    const count = ratings.length;

    const avgOverall = count ? (ratings.reduce((s, r) => s + (r.overallRating || 0), 0) / count) : 0;

    const avgMarks = count ? (ratings.reduce((s, r) => s + (r.obtainedMarks || 0), 0) / count) : 0;

    return res.json({ success: true, data: { status: 'pending', message: 'No stored summary', count, avgOverall, avgMarks } });

  } catch (err) {

    console.error('api.ratings.summary', err);

    return res.status(500).json({ success: false, error: { message: 'Server error' } });

  }

};

**controllers\ratingController.js:**

const { Rating, Audit, Offering } = require('../models');

*// helper to check edit window (7 days)*

const canEdit = (doc) => {

  if (!doc || !doc.createdAt) return false;

  const created = new Date(doc.createdAt).getTime();

  const now = Date.now();

  const sevenDays = 1000 \* 60 \* 60 \* 24 \* 7;

  return (now - created) <= sevenDays;

};

*// render form to rate an offering*

exports.renderForm = async (req, res) => {

  try {

    const offeringId = req.query.offering;

    if (!offeringId) return res.status(400).send('offering required');

*// Check if user already rated this offering*

    const existing = await Rating.findOne({ student: req.user.\_id, offering: offeringId });

    if (existing) {

*// If already rated, redirect to the ratings dashboard where they can edit their existing ratings*

      return res.redirect('/ratings');

    }

*// If not rated yet, show the rating form*

    return res.render('rating-form', { title: 'Rate Course', offeringId, existing: null });

  } catch (err) {

    console.error('renderForm', err);

    return res.status(500).send('Server error');

  }

};

*// submit a new rating (create)*

exports.create = async (req, res) => {

  try {

    const { offering, comment, anonymized, overallRating, obtainedMarks } = req.body;

    if (!offering) return res.status(400).json({ success: false, error: { message: 'offering required' } });

*// Authorization: ensure only students who were taught in this offering can submit a review.*

*// Admins are still allowed to create ratings (for testing/management) — adjust if needed.*

*// Load offering to check section/semester/teacher*

    const offeringDoc = await Offering.findById(offering).lean();

    if (!offeringDoc) return res.status(404).json({ success: false, error: { message: 'Offering not found' } });

*// ensure offering's term is active — ratings are only allowed while the term is active*

    try {

      const Term = require('../models').Term;

      let termDoc = null;

      if (offeringDoc.term && typeof offeringDoc.term === 'object' && offeringDoc.term.isActive !== undefined) termDoc = offeringDoc.term;

      else if (offeringDoc.term) termDoc = await Term.findById(offeringDoc.term).lean();

      if (termDoc && !termDoc.isActive) {

        return res.status(403).json({ success: false, error: { message: 'Ratings for this offering are closed (term has been promoted)' } });

      }

    } catch (e) { */\* ignore term lookup failures \*/* }

*// If the requester is not an admin, enforce that their section and semesterNumber match the offering.*

    if (!req.user || req.user.role !== 'admin') {

*// require user to be authenticated student*

      if (!req.user) return res.status(401).json({ success: false, error: { message: 'Authentication required' } });

      const userSection = (req.user.section || '').toString().trim();

      const offeringSection = (offeringDoc.section || '').toString().trim();

*// If offering has a semesterNumber, require user's semesterNumber to match*

      const offeringSem = offeringDoc.semesterNumber;

      const userSem = req.user.semesterNumber;

      const sectionMatches = offeringSection === '' || userSection === offeringSection;

      const semesterMatches = (typeof offeringSem === 'undefined' || offeringSem === null) || (typeof userSem !== 'undefined' && userSem === offeringSem);

      if (!sectionMatches || !semesterMatches) {

        return res.status(403).json({ success: false, error: { message: 'Not allowed to review this offering — you were not enrolled in this class/section' } });

      }

    }

*// overallRating required and must be 1-5*

  const or = parseInt(overallRating, 10);

  if (!or || or < 1 || or > 5) return res.status(400).json({ success: false, error: { message: 'overallRating (1-5) required' } });

*// obtainedMarks required*

  if (typeof obtainedMarks === 'undefined' || obtainedMarks === null || obtainedMarks === '') return res.status(400).json({ success: false, error: { message: 'obtainedMarks required' } });

*// Ratings are always anonymous in this system — enforce server-side*

  const ratingData = { student: req.user.\_id, offering, overallRating: or, anonymized: true };

    if (typeof comment !== 'undefined') ratingData.comment = comment;

    if (typeof obtainedMarks !== 'undefined' && obtainedMarks !== null && obtainedMarks !== '') ratingData.obtainedMarks = Number(obtainedMarks);

    const rating = new Rating(ratingData);

    await rating.save();

    await Audit.create({ action: 'rating.create', actor: req.user.\_id, targetType: 'Rating', targetId: rating.\_id });

    return res.status(201).json({ success: true, data: { message: 'Rating submitted' } });

  } catch (err) {

    console.error('rating.create error', err);

    if (err && err.code === 11000) return res.status(409).json({ success: false, error: { message: 'You have already rated this offering' } });

    return res.status(500).json({ success: false, error: { message: 'Server error' } });

  }

};

*// update an existing rating (only within 7 days)*

exports.update = async (req, res) => {

  try {

    const id = req.params.id;

    const rating = await Rating.findById(id);

    if (!rating) return res.status(404).json({ success: false, error: { message: 'Rating not found' } });

    if (rating.student.toString() !== req.user.\_id.toString()) return res.status(403).json({ success: false, error: { message: 'Not allowed' } });

*// ensure the offering's term is still active — once admin promotes the term, editing is closed*

    try {

      const offeringDoc = await Offering.findById(rating.offering).populate('term').lean();

      if (offeringDoc && offeringDoc.term && offeringDoc.term.isActive === false) {

        return res.status(403).json({ success: false, error: { message: 'Cannot edit rating: term has been promoted and ratings are closed' } });

      }

    } catch (e) { */\* ignore \*/* }

  const { comment, anonymized, overallRating, obtainedMarks } = req.body;

    if (typeof comment !== 'undefined') rating.comment = comment;

*// anonymized is always true for this app — do not allow changing*

  rating.anonymized = true;

    if (typeof overallRating !== 'undefined') {

      const or = parseInt(overallRating, 10);

      if (!or || or < 1 || or > 5) return res.status(400).json({ success: false, error: { message: 'overallRating must be 1-5' } });

      rating.overallRating = or;

    }

    if (typeof obtainedMarks !== 'undefined' && obtainedMarks !== null && obtainedMarks !== '') {

      const om = Number(obtainedMarks);

      if (isNaN(om) || om < 0) return res.status(400).json({ success: false, error: { message: 'obtainedMarks must be a number' } });

      rating.obtainedMarks = om;

    }

    await rating.save();

    await Audit.create({ action: 'rating.update', actor: req.user.\_id, targetType: 'Rating', targetId: rating.\_id });

    return res.status(200).json({ success: true, data: { message: 'Rating updated' } });

  } catch (err) {

    console.error('rating.update error', err);

    return res.status(500).json({ success: false, error: { message: 'Server error' } });

  }

};

*// view ratings for an offering (aggregate basic stats)*

exports.viewForOffering = async (req, res) => {

  try {

    const offering = req.query.offering || req.params.offering;

    if (!offering) {

*// No offering provided: render a table-driven ratings index page (with offerings list for a filter)*

      const offerings = await Offering.find().populate('course').populate('teacher').limit(200).lean();

      return res.render('ratings-index', { title: 'Ratings', offerings });

    }

*// load offering to inspect term*

    const offeringDoc = await Offering.findById(offering).populate('term').lean();

    if (!offeringDoc) return res.status(404).send('Offering not found');

*// if offering belongs to a non-active (previous) term, prefer to show stored summary if available*

    const TermModel = require('../models').Term;

    const RatingSummary = require('../models').RatingSummary;

    let termIsActive = false;

    if (offeringDoc.term && offeringDoc.term.\_id) {

      const termDoc = offeringDoc.term;

      termIsActive = !!termDoc.isActive;

    } else if (offeringDoc.term) {

      const tdoc = await TermModel.findById(offeringDoc.term).lean();

      termIsActive = !!(tdoc && tdoc.isActive);

    }

    if (!termIsActive) {

*// try to fetch a pre-generated summary*

      const summary = await RatingSummary.findOne({ offering: offering }).lean();

      if (summary) {

        return res.render('rating-list', { title: 'Ratings', summary });

      }

      const userSection = (req.user.section || '').toString().trim();

      const userSem = req.user.semesterNumber;

*// if we don't have a section for the user, they cannot give reviews for specific sections*

      if (!userSection) {

*// Ensure template variables are always present (avoid ReferenceError in EJS)*

        return res.render('rating-give-list', { title: 'Give Review', offerings: [], activeTerm: null, nextTerm: null, selectedTerm: null, user: req.user || null });

      }

*// determine term: allow override via ?term=<id>, otherwise use active term*

      const Term = require('../models').Term;

      activeTerm = await Term.findOne({ isActive: true }).lean();

*// compute next term name and ensure it exists (do not create here; only lookup)*

      if (activeTerm && activeTerm.name) {

        const m = String(activeTerm.name).toLowerCase().match(/^(fa|sp)(\d{2})$/);

        if (m) {

          const season = m[1];

          const y = parseInt(m[2], 10);

          let nextName = null;

          if (season === 'fa') nextName = 'sp' + String((y + 1)).padStart(2, '0');

          else nextName = 'fa' + String(y).padStart(2, '0');

          nextTerm = await Term.findOne({ name: nextName }).lean();

        }

      }

      if (req.query && req.query.term) {

        termFilter = req.query.term;

      } else {

        termFilter = activeTerm ? activeTerm.\_id : null;

      }

*// match offerings where section equals the student's section and term matches*

      filter = {

        section: userSection,

        term: termFilter,

        $or: [ { semesterNumber: { $exists: false } }, { semesterNumber: null }, { semesterNumber: userSem } ]

      };

    } else {

*// admin: allow optional term filter via query*

      if (req.query && req.query.term) termFilter = req.query.term;

    }

    let offerings = await Offering.find(filter).populate('course').populate('teacher').populate('term').limit(200).lean();

*// Remove offerings the student has already rated (they can view/edit those on the dashboard)*

    const rated = await Rating.find({

      student: req.user.\_id,

      offering: { $in: offerings.map(o => o.\_id) }

    }).distinct('offering');

    offerings = offerings.filter(o => !rated.includes(o.\_id.toString()));

*// Get next term info for display if needed*

    if (!nextTerm && activeTerm) {

      const nextTermName = activeTerm.name.toLowerCase().startsWith('fa')

        ? 'sp' + String(parseInt(activeTerm.name.slice(2)) + 1).padStart(2, '0')

        : 'fa' + activeTerm.name.slice(2);

      nextTerm = await Term.findOne({ name: nextTermName }).lean();

    }

    console.log(`Found ${offerings.length} offerings to rate for user ${req.user.\_id}`);

    return res.render('rating-give-list', {

      title: 'Give Review',

      offerings,

      activeTerm: activeTerm || null,

      nextTerm: nextTerm || null,

      selectedTerm: termFilter,

      user: req.user

    });

  } catch (err) {

    console.error('renderGiveList error', err);

    return res.status(500).send('Server error');

  }

};

*// Export the renderGiveList function*

exports.renderGiveList = async (req, res) => {

  try {

*// First check if user has required profile data*

    if (!req.user) {

      return res.status(401).send('Please log in to give reviews');

    }

*// Get active term*

    const Term = require('../models').Term;

    const activeTerm = await Term.findOne({ isActive: true }).lean();

    if (!activeTerm) {

      return res.render('rating-give-list', {

        title: 'Give Review',

        offerings: [],

        activeTerm: null,

        nextTerm: null,

        selectedTerm: null,

        user: req.user,

        error: 'No active term found'

      });

    }

*// Build base query for offerings in active term*

    const filter = {

      term: activeTerm.\_id

    };

*// For regular students (non-admin), filter by their section and semester*

    if (!req.user.role || req.user.role !== 'admin') {

      const userSection = (req.user.section || '').toString().trim();

      const userSemester = req.user.semesterNumber;

*// Match either:*

*// 1. Offering matches user's section exactly OR offering has no section requirement*

*// 2. Semester matches OR offering has no semester requirement*

      filter.$or = [{

        $and: [

          {

            $or: [

              { section: userSection },

              { section: { $exists: false } },

              { section: null },

              { section: '' }

            ]

          },

          {

            $or: [

              { semesterNumber: userSemester },

              { semesterNumber: { $exists: false } },

              { semesterNumber: null }

            ]

          }

        ]

      }];

    }

*// Get the list of offerings already rated by this user*

    const ratedOfferings = await Rating.distinct('offering', {

      student: req.user.\_id

    });

*// Add the filter to exclude already rated offerings*

    filter.\_id = { $nin: ratedOfferings };

*// Get all offerings matching the filter*

    let offerings = await Offering.find(filter)

      .populate({

        path: 'course',

        select: 'title code'

      })

      .populate({

        path: 'teacher',

        select: 'name'

      })

      .populate('term')

      .lean();

*// Offerings are already filtered at the database level*

*// No need for additional filtering here*

*// Get next term info for display*

    const nextTerm = await Term.findOne({

      isNext: true

    }).lean();

    console.log(`Found ${offerings.length} offerings to rate for user ${req.user.\_id}`);

    return res.render('rating-give-list', {

      title: 'Give Review',

      offerings,

      activeTerm: activeTerm,

      nextTerm: nextTerm,

      selectedTerm: activeTerm.\_id,

      user: req.user

    });

  } catch (err) {

    console.error('renderGiveList error', err);

    return res.status(500).send('Server error');

  }

};

*// render prefilled edit form for a rating*

exports.renderEditForm = async (req, res) => {

  try {

    const id = req.params.id;

    const rating = await Rating.findById(id).lean();

    if (!rating) return res.status(404).send('Rating not found');

*// ensure owner*

    if (!req.user || rating.student.toString() !== req.user.\_id.toString()) return res.status(403).send('Not allowed');

*// ensure the offering's term is active — edits are only allowed while the term is active*

    try {

      const offeringDoc = await Offering.findById(rating.offering).populate('term').lean();

      if (offeringDoc && offeringDoc.term && offeringDoc.term.isActive === false) {

        return res.status(403).send('Edit not allowed: term has been promoted and ratings are closed');

      }

    } catch (e) { */\* ignore \*/* }

  return res.render('rating-edit-form', { title: 'Edit Rating', rating });

  } catch (err) {

    console.error('renderEditForm', err);

    return res.status(500).send('Server error');

  }

};

**controllers\timetableController.js:**

const mongoose = require('mongoose');

const { Course, Teacher, Offering } = require('../models');

const Audit = require('../models').Audit;

const Term = require('../models').Term;

const ExcelJS = require('exceljs');

const fs = require('fs');

const path = require('path');

*// Helper: normalize name "First Last"*

function normalizeName(name) {

  if (!name) return null;

  return name.trim().split(/\s+/).map(s => s.charAt(0).toUpperCase() + s.slice(1).toLowerCase()).join(' ');

}

*// Escape string to safely use inside RegExp*

function escapeRegExp(str) {

  if (str === undefined || str === null) return '';

  return String(str).replace(/[.\*+?^${}()|[\]\\]/g, '\\$&');

}

*// PDF timetable parsing removed. Bulk timetable uploads are expected to be*

*// Excel (.xlsx/.xls) files and handled by `addClassesFromXlsx` below.*

*// Controller: upload timetable — XLSX-only bulk upload handler*

*// This endpoint now delegates to addClassesFromXlsx which implements the*

*// Excel parsing and DB upsert logic. It will return 400 for non-Excel files.*

exports.uploadTimetable = async (req, res) => {

  try {

*// accept multer-style req.file or express-fileupload req.files.file*

    const origName = (req.file && req.file.originalname) || (req.files && req.files.file && req.files.file.name) || '';

    if (!origName) return res.status(400).json({ success: false, error: { code: 'ERR\_NO\_FILE', message: 'Excel file (.xlsx) required' } });

    if (!/\.xls(x)?$/i.test(origName)) return res.status(400).json({ success: false, error: { code: 'ERR\_INVALID\_TYPE', message: 'Only .xlsx/.xls files are accepted' } });

*// Delegate to existing XLSX handler which supports the same upload formats*

    return await exports.addClassesFromXlsx(req, res);

  } catch (err) {

    console.error('uploadTimetable error', err && err.stack ? err.stack : err);

    const isProd = process.env.NODE\_ENV === 'production';

    const payload = { code: 'ERR\_INTERNAL', message: isProd ? 'Server error' : (err && err.message) || 'Server error' };

    if (!isProd && err && err.stack) payload.stack = err.stack;

    return res.status(500).json({ success: false, error: payload });

  }

};

*// Render manual add-class form for admin*

exports.renderAddClassForm = async (req, res) => {

  try {

*// provide active term and next term for admin selection*

    const active = await Term.findOne({ isActive: true }).lean();

    let next = null;

    if (active && active.name) {

*// compute next term name: if active starts with 'fa' -> next = 'sp' with year+1; if 'sp' -> next = 'fa' same year*

      const m = String(active.name).toLowerCase().match(/^(fa|sp)(\d{2})$/);

      if (m) {

        const season = m[1];

        const y = parseInt(m[2], 10);

        let nextName = null;

        if (season === 'fa') nextName = 'sp' + String((y + 1)).padStart(2, '0');

        else nextName = 'fa' + String(y).padStart(2, '0');

*// try find next term; create it if missing (inactive)*

        let nextDoc = await Term.findOne({ name: nextName });

        if (!nextDoc) {

          nextDoc = new Term({ name: nextName, isActive: false });

          await nextDoc.save();

        }

        next = nextDoc.toObject();

      }

    }

    let terms = [];

    if (active) terms.push(active);

    if (next) terms.push(next);

*// If there is no active term, fall back to listing all known terms (latest first)*

    if (!terms.length) {

      const all = await Term.find().sort({ startDate: -1 }).lean();

      terms = all || [];

    }

    return res.render('admin-add-class', { terms, user: req.user });

  } catch (err) {

    console.error('renderAddClassForm error', err && err.stack ? err.stack : err);

    return res.status(500).send('Server error');

  }

};

*// List terms (admin)*

exports.listTerms = async (req, res) => {

  try {

    const terms = await Term.find().sort({ startDate: -1 }).lean();

*// pass optional messages from query string (used after redirects)*

    const message = req.query && req.query.message ? req.query.message : null;

    const error = req.query && req.query.error ? req.query.error : null;

    return res.render('admin-terms', { title: 'Terms', terms, message, error });

*// return res.json({ title: 'Terms', terms, message, error });*

  } catch (err) {

    console.error('listTerms error', err);

    return res.status(500).send('Server error');

  }

};

*// Create a term (admin) - expects { name, startDate, endDate }*

exports.createTerm = async (req, res) => {

  try {

    const { name, startDate, endDate, isActive } = req.body;

    if (!name) return res.status(400).json({ success: false, error: { message: 'name required' } });

    const t = new Term({ name: String(name).trim(), startDate: startDate ? new Date(startDate) : undefined, endDate: endDate ? new Date(endDate) : undefined, isActive: !!isActive });

*// if isActive true, deactivate others*

    if (t.isActive) await Term.updateMany({ \_id: { $ne: t.\_id } }, { isActive: false });

    await t.save();

    await Audit.create({ action: 'term.create', actor: req.user ? req.user.\_id : null, targetType: 'Term', targetId: t.\_id });

    return res.status(201).json({ success: true, data: { term: t } });

  } catch (err) {

    console.error('createTerm error', err);

    return res.status(500).json({ success: false, error: { message: 'Server error' } });

  }

};

*// Activate an existing term*

exports.activateTerm = async (req, res) => {

  try {

    const id = req.params.id;

  const term = await Term.findById(id);

*// capture currently active term (if any) so we can generate summaries for it after activation*

  const prevActiveTerm = await Term.findOne({ isActive: true }).lean();

    if (!term) return res.status(404).json({ success: false, error: { message: 'Term not found' } });

*// enforce start and end dates before activation*

    if (!term.startDate || !term.endDate) return res.status(400).json({ success: false, error: { message: 'Term must have startDate and endDate before activation' } });

    await Term.updateMany({}, { isActive: false });

    term.isActive = true;

    await term.save();

    await Audit.create({ action: 'term.activate', actor: req.user ? req.user.\_id : null, targetType: 'Term', targetId: term.\_id });

    return res.json({ success: true, data: { term } });

  } catch (err) {

    console.error('activateTerm error', err);

    return res.status(500).json({ success: false, error: { message: 'Server error' } });

  }

};

*// Promote term: activate the term and create the following next term (with optional start/end dates)*

exports.promoteTerm = async (req, res) => {

  try {

    const id = req.params.id;

    const { nextStartDate, nextEndDate } = req.body || {};

    const term = await Term.findById(id);

*// capture currently active term BEFORE we change anything so summaries can be generated for it*

    const prevActiveTerm = await Term.findOne({ isActive: true }).lean();

    if (!term) return res.status(404).json({ success: false, error: { message: 'Term not found' } });

*// If the request provided start/end dates (from the UI), persist them first*

    if (req.body && (req.body.startDate || req.body.endDate)) {

      if (req.body.startDate) term.startDate = new Date(req.body.startDate);

      if (req.body.endDate) term.endDate = new Date(req.body.endDate);

      await term.save();

    }

*// Enforce that the selected term has startDate and endDate before activating*

    if (!term.startDate || !term.endDate) {

      const accept = req.headers && req.headers.accept ? req.headers.accept : '';

      const errMsg = 'Selected term must have Start and End dates before it can be activated.';

      if (accept.indexOf('text/html') !== -1) {

*// redirect back with an error query so the UI can display the message*

        return res.redirect('/admin/terms?error=' + encodeURIComponent(errMsg));

      }

      return res.status(400).json({ success: false, error: { message: errMsg } });

    }

*// activate selected term*

    await Term.updateMany({}, { isActive: false });

    term.isActive = true;

    await term.save();

    await Audit.create({ action: 'term.activate', actor: req.user ? req.user.\_id : null, targetType: 'Term', targetId: term.\_id });

*// compute next term name from activated term name*

    let nextName = null;

    const m = String(term.name).toLowerCase().match(/^(fa|sp)(\d{2})$/);

    if (m) {

      const season = m[1];

      const y = parseInt(m[2], 10);

      if (season === 'fa') nextName = 'sp' + String(y + 1).padStart(2, '0');

      else nextName = 'fa' + String(y).padStart(2, '0');

    }

    let createdNext = null;

    if (nextName) {

      let next = await Term.findOne({ name: nextName });

      if (!next) {

        next = new Term({ name: nextName, isActive: false });

*// next term dates may be provided optionally, but not required*

        if (nextStartDate) next.startDate = new Date(nextStartDate);

        if (nextEndDate) next.endDate = new Date(nextEndDate);

        await next.save();

        createdNext = next;

        await Audit.create({ action: 'term.create', actor: req.user ? req.user.\_id : null, targetType: 'Term', targetId: next.\_id });

      }

    }

*// generate and store comment summaries for the previous term (if any)*

    try {

      const prevTerm = prevActiveTerm; *// captured earlier*

      if (prevTerm && prevTerm.\_id) {

        const Rating = require('../models').Rating;

        const RatingSummary = require('../models').RatingSummary;

        const Offering = require('../models').Offering;

        const { summarizeComments } = require('../lib/commentSummarizer');

        const offerings = await Offering.find({ term: prevTerm.\_id }).select('\_id').lean();

        for (const o of offerings) {

          try {

            const ratings = await Rating.find({ offering: o.\_id }).lean();

            if (!ratings || ratings.length === 0) continue;

            const comments = ratings.map(r => r.comment).filter(Boolean);

            const avgOverall = ratings.reduce((s, r) => s + (r.overallRating || 0), 0) / ratings.length;

            const avgMarks = ratings.reduce((s, r) => s + (typeof r.obtainedMarks !== 'undefined' && r.obtainedMarks !== null ? r.obtainedMarks : 0), 0) / ratings.length;

            const summaryObj = summarizeComments(comments, avgOverall || 0, avgMarks || 0);

            await RatingSummary.findOneAndUpdate({ offering: o.\_id, term: prevTerm.\_id }, { summary: summaryObj.summary, avgOverall: summaryObj.avgOverall, avgMarks: summaryObj.avgMarks, count: summaryObj.count }, { upsert: true, new: true });

          } catch (inner) {

            console.warn('Could not generate summary for offering', o.\_id, inner && inner.message);

          }

        }

      }

    } catch (e) {

      console.error('Error generating summaries for previous term', e && e.stack ? e.stack : e);

    }

*// If request comes from browser form, redirect back to terms page with success message*

    const accept = req.headers && req.headers.accept ? req.headers.accept : '';

    if (accept.indexOf('text/html') !== -1) return res.redirect('/admin/terms?message=' + encodeURIComponent('Term activated and next term created'));

    return res.json({ success: true, data: { activated: term, next: createdNext } });

  } catch (err) {

    console.error('promoteTerm error', err);

    return res.status(500).json({ success: false, error: { message: 'Server error' } });

  }

};

*// Update term (set start/end dates and optionally activate)*

exports.updateTerm = async (req, res) => {

  try {

    const id = req.params.id;

    const { startDate, endDate, activate } = req.body || {};

    const term = await Term.findById(id);

    if (!term) return res.status(404).json({ success: false, error: { message: 'Term not found' } });

    if (startDate) term.startDate = new Date(startDate);

    if (endDate) term.endDate = new Date(endDate);

    if (activate) {

*// require dates*

      if (!term.startDate || !term.endDate) return res.status(400).json({ success: false, error: { message: 'Start and End dates required to activate' } });

      await Term.updateMany({}, { isActive: false });

      term.isActive = true;

    }

    await term.save();

    await Audit.create({ action: 'term.update', actor: req.user ? req.user.\_id : null, targetType: 'Term', targetId: term.\_id });

    const accept = req.headers && req.headers.accept ? req.headers.accept : '';

    if (accept.indexOf('text/html') !== -1) return res.redirect('/admin/terms');

    return res.json({ success: true, data: { term } });

  } catch (err) {

    console.error('updateTerm error', err);

    return res.status(500).json({ success: false, error: { message: 'Server error' } });

  }

};

*// helper: resolve the term to use (req.body.term or req.query.term or active term)*

async function resolveTermFromRequest(req) {

  const TermModel = Term;

  const termId = (req.body && req.body.term) || (req.query && req.query.term);

  if (termId) {

    const t = await TermModel.findById(termId);

    if (t) return t;

  }

*// fallback to active term*

  const active = await TermModel.findOne({ isActive: true });

  return active || null;

}

*// POST handler to add a single class manually (course + teacher + offering)*

exports.addClassManually = async (req, res) => {

  try {

*// Accept either a batch of subjects (subjects: [{code,title,teacherName}, ...])*

*// or single-subject fields (code, title, teacherName)*

    const { department: deptName, program: programName, semesterNumber, section } = req.body;

    let subjects = [];

    if (Array.isArray(req.body.subjects) && req.body.subjects.length > 0) {

      subjects = req.body.subjects;

    } else if (req.body.code && req.body.teacherName) {

      subjects = [{ code: req.body.code, title: req.body.title, teacherName: req.body.teacherName }];

    } else {

      return res.status(400).json({ success: false, error: { code: 'ERR\_NO\_SUBJECTS', message: 'No subjects provided' } });

    }

    if (!deptName) return res.status(400).json({ success: false, error: { code: 'ERR\_NO\_DEPARTMENT', message: 'department required' } });

    if (!programName) return res.status(400).json({ success: false, error: { code: 'ERR\_NO\_PROGRAM', message: 'program required' } });

    if (!semesterNumber) return res.status(400).json({ success: false, error: { code: 'ERR\_NO\_SEMESTER', message: 'semesterNumber required' } });

*// resolve or create department and program*

    const dn = String(deptName).trim();

    let department = await require('../models').Department.findOne({ name: dn });

    if (!department) { department = new (require('../models').Department)({ name: dn }); await department.save(); }

    const pn = String(programName).trim();

    let program = await require('../models').Program.findOne({ name: pn, department: department.\_id });

    if (!program) { program = new (require('../models').Program)({ name: pn, department: department.\_id }); await program.save(); }

    const summary = { createdCourses: 0, createdTeachers: 0, createdOfferings: 0, skipped: 0, details: [] };

*// resolve term (body may include term id)*

  const resolvedTerm = await resolveTermFromRequest(req);

  if (!resolvedTerm) return res.status(400).json({ success: false, error: { code: 'ERR\_NO\_TERM', message: 'No active term and no term provided' } });

  for (const s of subjects) {

      const code = s.code ? String(s.code).replace(/\s+/g, '').toUpperCase() : null;

      const title = s.title ? String(s.title).trim() : code;

      const teacherName = s.teacherName ? normalizeName(String(s.teacherName)) : null;

      if (!code) { summary.skipped++; summary.details.push({ subject: s, reason: 'no code' }); continue; }

      if (!teacherName) { summary.skipped++; summary.details.push({ subject: s, reason: 'no teacher' }); continue; }

*// upsert course*

      let course = await Course.findOne({ code });

      if (!course) { course = new Course({ code, title }); await course.save(); summary.createdCourses++; }

*// upsert teacher*

      const parts = teacherName.split(/\s+/).filter(Boolean);

      const first = parts[0] || '';

      const last = parts.slice(1).join(' ') || '';

      let teacher = await Teacher.findOne({ 'name.first': new RegExp('^' + escapeRegExp(first) + '$', 'i'), 'name.last': new RegExp('^' + escapeRegExp(last) + '$', 'i') });

      if (!teacher) { teacher = new Teacher({ name: { first, last } }); await teacher.save(); summary.createdTeachers++; }

*// create offering if not exists*

      const offeringQuery = { course: course.\_id, teacher: teacher.\_id, term: resolvedTerm.\_id };

      if (section) offeringQuery.section = String(section).trim();

      let existing = await Offering.findOne(offeringQuery);

      if (!existing) {

        const offering = new Offering({

          course: course.\_id,

          teacher: teacher.\_id,

          section: section ? String(section).trim() : undefined,

          department: department.\_id,

          program: program.\_id,

          semesterNumber: Number(semesterNumber)

        });

        offering.term = resolvedTerm.\_id;

        try {

          await offering.save();

          summary.createdOfferings++;

        } catch (saveErr) {

          if (saveErr && (saveErr.code === 11000 || saveErr.code === 11001)) {

            summary.skipped++;

            summary.details.push({ row: r, reason: 'duplicate offering', error: (saveErr.keyValue || saveErr.message) });

          } else {

            throw saveErr;

          }

        }

      }

      summary.details.push({ subject: s, courseId: course.\_id, teacherId: teacher.\_id });

    }

    await Audit.create({ action: 'timetable.manualAddBatch', actor: req.user ? req.user.\_id : null, targetType: 'Program', targetId: program.\_id, details: summary });

    return res.status(200).json({ success: true, data: summary });

  } catch (err) {

    console.error('addClassManually error', err && err.stack ? err.stack : err);

    const isProd = process.env.NODE\_ENV === 'production';

    const payload = { code: 'ERR\_INTERNAL', message: isProd ? 'Server error' : (err && err.message) || 'Server error' };

    if (!isProd && err && err.stack) payload.stack = err.stack;

    return res.status(500).json({ success: false, error: payload });

  }

};

*// List offerings filtered by term (admin view)*

exports.listOfferingsByTerm = async (req, res) => {

  try {

    const termId = req.query.term;

    let term = null;

    if (termId) term = await Term.findById(termId).lean();

    if (!term) term = await Term.findOne({ isActive: true }).lean();

    const filter = {};

    if (term && term.\_id) filter.term = term.\_id;

    const offerings = await Offering.find(filter).populate('course teacher department program term').sort({ 'course.code': 1 }).lean();

    const terms = await Term.find().sort({ startDate: -1 }).lean();

  return res.render('admin/admin-offerings', { title: 'Offerings', offerings, terms, currentTerm: term, user: req.user });

*// return res.json({ title: 'Offerings', offerings, terms, currentTerm: term, user: req.user });*

  } catch (err) {

    console.error('listOfferingsByTerm error', err);

    return res.status(500).send('Server error');

  }

};

*// Render offering edit form*

exports.renderOfferingEditForm = async (req, res) => {

  try {

    const id = req.params.id;

    const offering = await Offering.findById(id).populate('course teacher department program term').lean();

    if (!offering) return res.status(404).send('Offering not found');

    const terms = await Term.find().sort({ startDate: -1 }).lean();

    const departments = await require('../models').Department.find().lean();

    const programs = await require('../models').Program.find().lean();

  return res.render('admin/admin-offering-edit', { title: 'Edit Offering', offering, terms, departments, programs, user: req.user });

  } catch (err) {

    console.error('renderOfferingEditForm error', err);

    return res.status(500).send('Server error');

  }

};

*// Update offering (admin)*

exports.updateOffering = async (req, res) => {

  try {

    const id = req.params.id;

    const offering = await Offering.findById(id);

    if (!offering) return res.status(404).json({ success: false, error: { message: 'Offering not found' } });

    const { department: deptId, program: programId, semesterNumber, section, term: termId } = req.body;

  const courseId = req.body.courseId;

  const courseTitle = req.body.courseTitle;

    if (deptId) offering.department = deptId;

    if (programId) offering.program = programId;

    if (typeof semesterNumber !== 'undefined') offering.semesterNumber = Number(semesterNumber) || offering.semesterNumber;

    if (typeof section !== 'undefined') offering.section = section || offering.section;

    if (termId) offering.term = termId;

*// update course title if requested*

    if (courseId && courseTitle) {

      try {

        const Course = require('../models').Course;

        await Course.findByIdAndUpdate(courseId, { title: String(courseTitle).trim() });

      } catch (e) { console.warn('Could not update course title', e && e.message); }

    }

    await offering.save();

    await Audit.create({ action: 'offering.update', actor: req.user ? req.user.\_id : null, targetType: 'Offering', targetId: offering.\_id });

*// If request expects HTML (form submit), redirect back to listings for a smoother UX*

    const accept = req.headers && req.headers.accept ? req.headers.accept : '';

    if (accept.indexOf('text/html') !== -1) {

      return res.redirect('/admin/offerings');

    }

    return res.json({ success: true, data: { offering } });

  } catch (err) {

    console.error('updateOffering error', err);

    return res.status(500).json({ success: false, error: { message: 'Server error' } });

  }

};

*// Delete offering (admin)*

exports.deleteOffering = async (req, res) => {

  try {

    const id = req.params.id;

    const offering = await Offering.findByIdAndDelete(id);

    if (!offering) return res.status(404).json({ success: false, error: { message: 'Offering not found' } });

    await Audit.create({ action: 'offering.delete', actor: req.user ? req.user.\_id : null, targetType: 'Offering', targetId: id });

    const accept = req.headers && req.headers.accept ? req.headers.accept : '';

    if (accept.indexOf('text/html') !== -1) {

      return res.redirect('/admin/offerings');

    }

    return res.json({ success: true, data: { id } });

  } catch (err) {

    console.error('deleteOffering error', err);

    return res.status(500).json({ success: false, error: { message: 'Server error' } });

  }

};

*// POST handler: parse uploaded XLSX and add classes in bulk*

exports.addClassesFromXlsx = async (req, res) => {

  try {

*// Support multer-style (req.file.buffer), express-fileupload (req.files.file.data),*

*// and express-fileupload with useTempFiles (req.files.file.tempFilePath).*

    let buffer = null;

    let tempPath = null;

    if (req.file && req.file.buffer) {

      buffer = req.file.buffer;

    } else if (req.files && req.files.file) {

      const f = req.files.file;

      if (f.data && f.data.length && f.data.length > 0) {

        buffer = f.data;

      } else if (f.tempFilePath) {

        tempPath = f.tempFilePath;

      }

    }

    if (!buffer && !tempPath) return res.status(400).json({ success: false, error: { code: 'ERR\_NO\_FILE', message: 'Excel file required' } });

*// extra diagnostics: check buffer length or temp file stats*

*// optional diagnostics removed - keep code minimal*

    const workbook = new ExcelJS.Workbook();

    try {

      if (buffer) {

        if (!buffer || buffer.length === 0) throw new Error('uploaded buffer empty');

        await workbook.xlsx.load(buffer);

      } else {

*// read from temp file path*

        const st = fs.statSync(tempPath);

        if (!st || st.size === 0) throw new Error('uploaded temp file empty');

        await workbook.xlsx.readFile(tempPath);

      }

    } finally {

*// cleanup temp file if present*

      try { if (tempPath && fs.existsSync(tempPath)) fs.unlinkSync(tempPath); } catch (e) { */\* ignore cleanup errors \*/* }

    }

    const worksheet = workbook.worksheets && workbook.worksheets[0];

    if (!worksheet) return res.status(400).json({ success: false, error: { code: 'ERR\_EMPTY', message: 'No sheets found in workbook' } });

*// Convert rows to objects using header row (first non-empty row assumed headers)*

    const rows = [];

    let headerMap = null;

    const maxRows = 2000; *// safety cap*

    worksheet.eachRow({ includeEmpty: false }, (row, rowNumber) => {

      if (rows.length > maxRows) return; *// stop collecting beyond cap*

      const values = row.values ? row.values.slice(1) : [];

      if (!headerMap) {

*// build header map from this row*

        headerMap = values.map(h => (h || '').toString().trim());

        return;

      }

      const obj = {};

      for (let i = 0; i < headerMap.length; i++) {

        const key = headerMap[i] || `col${i}`;

        obj[key] = values[i] !== undefined ? values[i] : '';

      }

      rows.push(obj);

    });

    if (!rows || rows.length === 0) return res.status(400).json({ success: false, error: { code: 'ERR\_EMPTY', message: 'No data rows found in sheet' } });

    const summary = { createdCourses: 0, createdTeachers: 0, createdOfferings: 0, skipped: 0, details: [] };

*// resolve term (may be provided as a column 'term' or passed in body)*

  const resolvedTerm = await resolveTermFromRequest(req);

  if (!resolvedTerm) return res.status(400).json({ success: false, error: { code: 'ERR\_NO\_TERM', message: 'No active term and no term provided' } });

  const seenKeys = new Set(); *// in-memory dedupe for rows within the same uploaded file*

  for (const r of rows) {

*// attempt to find common header keys*

      const code = r.code || r.Code || r['Course Code'] || r['course code'] || r['Code'] || r['code'] || r['course\_code'];

      const title = r.title || r.Title || r['Course Title'] || r['course title'] || r['Title'] || r['title'];

      const teacherName = r.teacher || r.Teacher || r['Teacher Name'] || r['teacher name'] || r['Instructor'] || r['instructor'];

      const deptName = r.department || r.Department || r['Department'] || 'Unknown';

      const programName = r.program || r.Program || r['Program'] || 'Unknown';

      const semesterNumber = r.semesterNumber || r.Semester || r['Semester Number'] || 1;

      const section = r.section || r.Section || r['Section'] || undefined;

      const codeNorm = code ? String(code).replace(/\s+/g, '').toUpperCase() : null;

      const teacherNorm = teacherName ? normalizeName(String(teacherName)) : null;

*// in-file dedupe key: courseCode|teacherNormalized|section*

      const fileKey = `${codeNorm || ''}|${teacherNorm || ''}|${(section || '')}`;

      if (seenKeys.has(fileKey)) {

        summary.skipped++;

        summary.details.push({ row: r, reason: 'duplicate in uploaded file' });

        continue;

      }

      seenKeys.add(fileKey);

      if (!codeNorm || !teacherNorm) { summary.skipped++; summary.details.push({ row: r, reason: 'missing code or teacher' }); continue; }

      let department = await require('../models').Department.findOne({ name: String(deptName).trim() });

      if (!department) { department = new (require('../models').Department)({ name: String(deptName).trim() }); await department.save(); }

      let program = await require('../models').Program.findOne({ name: String(programName).trim(), department: department.\_id });

      if (!program) { program = new (require('../models').Program)({ name: String(programName).trim(), department: department.\_id }); await program.save(); }

      let course = await Course.findOne({ code: codeNorm });

      if (!course) { course = new Course({ code: codeNorm, title: String(title || codeNorm) }); await course.save(); summary.createdCourses++; }

      const parts = teacherNorm.split(/\s+/).filter(Boolean);

      const first = parts[0] || '';

      const last = parts.slice(1).join(' ') || '';

      let teacher = await Teacher.findOne({ 'name.first': new RegExp('^' + escapeRegExp(first) + '$', 'i'), 'name.last': new RegExp('^' + escapeRegExp(last) + '$', 'i') });

      if (!teacher) { teacher = new Teacher({ name: { first, last } }); await teacher.save(); summary.createdTeachers++; }

      const offeringQuery = { course: course.\_id, teacher: teacher.\_id, term: resolvedTerm.\_id };

      if (section) offeringQuery.section = String(section).trim();

      let existing = await Offering.findOne(offeringQuery);

      if (!existing) {

        const offering = new Offering({ course: course.\_id, teacher: teacher.\_id, section: section ? String(section).trim() : undefined, department: department.\_id, program: program.\_id, semesterNumber: Number(semesterNumber) });

        offering.term = resolvedTerm.\_id;

        try {

          await offering.save();

          summary.createdOfferings++;

        } catch (saveErr) {

*// handle duplicate key errors gracefully during bulk operations*

          if (saveErr && (saveErr.code === 11000 || saveErr.code === 11001)) {

            summary.skipped++;

            summary.details.push({ row: r, reason: 'duplicate offering', error: (saveErr.keyValue || saveErr.message) });

          } else {

*// record and continue to avoid aborting the entire bulk upload*

            summary.skipped++;

            summary.details.push({ row: r, reason: 'save error', error: (saveErr && saveErr.message) || String(saveErr) });

          }

        }

      }

      summary.details.push({ row: r, courseId: course.\_id, teacherId: teacher.\_id });

    }

    await Audit.create({ action: 'timetable.bulkUpload', actor: req.user ? req.user.\_id : null, targetType: 'BulkUpload', details: summary });

    return res.status(200).json({ success: true, data: summary });

  } catch (err) {

    console.error('addClassesFromXlsx error', err && err.stack ? err.stack : err);

    const isProd = process.env.NODE\_ENV === 'production';

    const payload = { code: 'ERR\_INTERNAL', message: isProd ? 'Server error' : (err && err.message) || 'Server error' };

    if (!isProd && err && err.stack) payload.stack = err.stack;

    return res.status(500).json({ success: false, error: payload });

  }

};

*// debug endpoint removed*

**middleware\adminAuth.js:**

const auth = require('./auth');

module.exports = async function adminAuth(req, res, next) {

*// first run general auth to populate req.user*

  auth(req, res, async function(err) {

    if (err) return next(err);

    try {

      if (!req.user || req.user.role !== 'admin') return res.status(403).json({ success: false, error: { code: 'ERR\_FORBIDDEN', message: 'Admin access required' } });

      return next();

    } catch (e) {

      console.error('adminAuth wrapper error', e);

      return res.status(500).json({ success: false, error: { code: 'ERR\_INTERNAL', message: 'Server error' } });

    }

  });

};

**middleware\adminRequire.js:**

const jwt = require('jsonwebtoken');

const User = require('../models').User;

const blacklist = require('../utils/tokenBlacklist');

const JWT\_SECRET = process.env.JWT\_SECRET || 'change-this-secret';

module.exports = async function adminRequire(req, res, next) {

  try {

*// look for token in cookies or Authorization header or query*

    let token = null;

    if (req.cookies && req.cookies.adminToken) token = req.cookies.adminToken;

    else if (req.cookies && req.cookies.token) token = req.cookies.token;

    else if (req.headers.authorization) {

      const parts = req.headers.authorization.split(' ');

      if (parts.length === 2 && /^Bearer$/i.test(parts[0])) token = parts[1];

      else token = parts[0];

    } else if (req.query && req.query.token) token = req.query.token;

    if (!token) {

      return res.redirect('/admin/login?next=' + encodeURIComponent(req.originalUrl));

    }

    if (blacklist.isBlacklisted(token)) return res.redirect('/admin/login?next=' + encodeURIComponent(req.originalUrl));

    let payload = null;

    try { payload = jwt.verify(token, JWT\_SECRET); } catch (e) { return res.redirect('/admin/login?next=' + encodeURIComponent(req.originalUrl)); }

    const user = await User.findById(payload.sub);

    if (!user || user.role !== 'admin') return res.redirect('/admin/login?next=' + encodeURIComponent(req.originalUrl));

    req.user = user;

    next();

  } catch (err) {

    console.error('adminRequire error', err);

    return res.redirect('/admin/login');

  }

};

**middleware\auth.js:**

const jwt = require('jsonwebtoken');

const User = require('../models').User;

const blacklist = require('../utils/tokenBlacklist');

const JWT\_SECRET = process.env.JWT\_SECRET || 'change-this-secret';

module.exports = async function auth(req, res, next) {

  try {

  const auth = req.headers.authorization || req.headers.Authorization || req.query.token || (req.cookies && (req.cookies.token || req.cookies.adminToken)) || null;

    if (!auth) return res.status(401).json({ success: false, error: { code: 'ERR\_AUTH', message: 'Authorization required' } });

    const parts = String(auth).split(' ');

    let token = null;

    if (parts.length === 2 && /^Bearer$/i.test(parts[0])) token = parts[1];

    else token = parts[0];

    if (!token) return res.status(401).json({ success: false, error: { code: 'ERR\_AUTH', message: 'Authorization token missing' } });

    if (blacklist.isBlacklisted(token)) return res.status(401).json({ success: false, error: { code: 'ERR\_AUTH', message: 'Token revoked' } });

  let payload = null;

  try { payload = jwt.verify(token, JWT\_SECRET); } catch (err) { return res.status(401).json({ success: false, error: { code: 'ERR\_AUTH', message: 'Invalid or expired token' } }); }

    const user = await User.findById(payload.sub);

    if (!user) return res.status(401).json({ success: false, error: { code: 'ERR\_AUTH', message: 'Invalid token user' } });

    req.user = user;

    req.authToken = token;

    next();

  } catch (err) {

    console.error('auth middleware error', err);

    return res.status(500).json({ success: false, error: { code: 'ERR\_INTERNAL', message: 'Server error' } });

  }

};

**middleware\loginRequire.js:**

const jwt = require('jsonwebtoken');

const User = require('../models').User;

const blacklist = require('../utils/tokenBlacklist');

const JWT\_SECRET = process.env.JWT\_SECRET || 'change-this-secret';

module.exports = async function loginRequire(req, res, next) {

  try {

*// check cookies, Authorization header, or query token*

    let token = null;

    if (req.cookies && req.cookies.token) token = req.cookies.token;

    else if (req.cookies && req.cookies.adminToken) token = req.cookies.adminToken;

    else if (req.headers && req.headers.authorization) {

      const parts = req.headers.authorization.split(' ');

      if (parts.length === 2 && /^Bearer$/i.test(parts[0])) token = parts[1];

      else token = parts[0];

    } else if (req.query && req.query.token) token = req.query.token;

    if (!token) return res.redirect('/login?next=' + encodeURIComponent(req.originalUrl));

    if (blacklist.isBlacklisted(token)) return res.redirect('/login?next=' + encodeURIComponent(req.originalUrl));

    let payload = null;

    try { payload = jwt.verify(token, JWT\_SECRET); } catch (e) { return res.redirect('/login?next=' + encodeURIComponent(req.originalUrl)); }

    const user = await User.findById(payload.sub);

    if (!user) return res.redirect('/login?next=' + encodeURIComponent(req.originalUrl));

    req.user = user;

    next();

  } catch (err) {

    console.error('loginRequire error', err);

    return res.redirect('/login');

  }

};

**models\audit.js:**

const mongoose = require('mongoose');

const { Schema } = mongoose;

const auditSchema = new Schema({

  action: { type: String, required: true },

  actor: { type: Schema.Types.ObjectId, ref: 'User' },

  targetType: { type: String },

  targetId: { type: Schema.Types.ObjectId },

  details: { type: Schema.Types.Mixed }

}, { timestamps: true });

module.exports = mongoose.model('Audit', auditSchema);

**models\course.js**

const mongoose = require('mongoose');

const { Schema } = mongoose;

const courseSchema = new Schema({

  code: { type: String, required: true, trim: true, index: true },

  title: { type: String, required: true, trim: true },

  description: { type: String }

}, { timestamps: true });

courseSchema.index({ code: 1 }, { unique: true });

module.exports = mongoose.model('Course', courseSchema);

**models\department.js:**

const mongoose = require('mongoose');

const { Schema } = mongoose;

const departmentSchema = new Schema({

  name: { type: String, required: true, trim: true, index: true }

}, { timestamps: true });

departmentSchema.index({ name: 1 }, { unique: true });

module.exports = mongoose.model('Department', departmentSchema);

**models\index.js:**

module.exports = {

  User: require('./user'),

  Course: require('./course'),

  Offering: require('./offering'),

  Teacher: require('./teacher'),

  Department: require('./department'),

  Program: require('./program'),

  Rating: require('./rating'),

  RatingSummary: require('./ratingSummary'),

  Term: require('./term'),

  Audit: require('./audit')

  ,VerificationToken: require('./verificationToken')

};

**models\offering.js:**

const mongoose = require('mongoose');

const { Schema } = mongoose;

const offeringSchema = new Schema({

  course: { type: Schema.Types.ObjectId, ref: 'Course', required: true, index: true },

  teacher: { type: Schema.Types.ObjectId, ref: 'Teacher', required: true, index: true },

  section: { type: String },

  term: { type: Schema.Types.ObjectId, ref: 'Term' },

  department: { type: Schema.Types.ObjectId, ref: 'Department' },

  program: { type: Schema.Types.ObjectId, ref: 'Program' },

  semesterNumber: { type: Number }

}, { timestamps: true });

*// Unique per course+teacher+term+section to avoid duplicate identical offerings across terms*

*// Note: if your database already has a different unique index (older schema), you'll need to drop it*

*// in the database (mongo shell or Mongo GUI) before this new index can be created without error.*

offeringSchema.index({ course: 1, teacher: 1, term: 1, section: 1 }, { unique: true });

module.exports = mongoose.model('Offering', offeringSchema);

**models\program.js:**

const mongoose = require('mongoose');

const { Schema } = mongoose;

const programSchema = new Schema({

  name: { type: String, required: true, trim: true, index: true },

  department: { type: Schema.Types.ObjectId, ref: 'Department' }

}, { timestamps: true });

programSchema.index({ name: 1 });

module.exports = mongoose.model('Program', programSchema);

**models\rating.js:**

const mongoose = require('mongoose');

const { Schema } = mongoose;

const ratingSchema = new Schema({

  student: { type: Schema.Types.ObjectId, ref: 'User', required: true, index: true },

  offering: { type: Schema.Types.ObjectId, ref: 'Offering', required: true, index: true },

*// overall rating 1-5 (required)*

  overallRating: { type: Number, min: 1, max: 5, required: true, index: true },

*// obtained marks (required, 0-100)*

  obtainedMarks: { type: Number, required: true, min: 0 },

  comment: { type: String },

  ipHash: { type: String },

  anonymized: { type: Boolean, default: false }

}, { timestamps: true });

*// prevent duplicate rating per student per offering*

ratingSchema.index({ student: 1, offering: 1 }, { unique: true });

ratingSchema.index({ offering: 1 });

module.exports = mongoose.model('Rating', ratingSchema);

**models\ratingSummary.js:**

const mongoose = require('mongoose');

const { Schema } = mongoose;

const ratingSummarySchema = new Schema({

  offering: { type: Schema.Types.ObjectId, ref: 'Offering', required: true, index: true },

  term: { type: Schema.Types.ObjectId, ref: 'Term', required: true, index: true },

  summary: { type: String },

  avgOverall: { type: Number },

  avgMarks: { type: Number },

  count: { type: Number },

}, { timestamps: true });

ratingSummarySchema.index({ offering: 1, term: 1 }, { unique: true });

module.exports = mongoose.model('RatingSummary', ratingSummarySchema);

**models\teacher.js:**

const mongoose = require('mongoose');

const { Schema } = mongoose;

const teacherSchema = new Schema({

  name: { first: String, last: String },

  staffId: { type: String, index: true },

  email: { type: String, lowercase: true, trim: true },

  department: { type: String }

}, { timestamps: true });

module.exports = mongoose.model('Teacher', teacherSchema);

**models\term.js:**

const mongoose = require('mongoose');

const { Schema } = mongoose;

const termSchema = new Schema({

  name: { type: String, required: true, unique: true, trim: true },

  startDate: { type: Date },

  endDate: { type: Date },

  isActive: { type: Boolean, default: false }

}, { timestamps: true });

module.exports = mongoose.model('Term', termSchema);

**models\user.js:**

const mongoose = require('mongoose');

const { Schema } = mongoose;

const nameSchema = new Schema({

  first: { type: String },

  last: { type: String }

}, { \_id: false });

const userSchema = new Schema({

  email: { type: String, required: true, unique: true, lowercase: true, trim: true },

  passwordHash: { type: String, required: true },

*// Note: teachers are stored in a separate Teacher collection and do not have user logins.*

  role: { type: String, required: true, enum: ['student', 'admin'], index: true },

  name: { type: nameSchema },

  intake: { season: String, year: Number },

  degreeShort: { type: String },

  rollNumber: { type: String },

  semesterNumber: { type: Number },

  department: { type: Schema.Types.ObjectId, ref: 'Department' },

  program: { type: Schema.Types.ObjectId, ref: 'Program' },

*// profile fields*

  section: { type: String },

  phone: { type: String },

  cgpa: { type: Number },

  profileComplete: { type: Boolean, default: false },

  idCardImage: { type: String },

  studentId: { type: String, index: true, sparse: true },

  staffId: { type: String, index: true, sparse: true },

  isActive: { type: Boolean, default: true }

}, { timestamps: true });

module.exports = mongoose.model('User', userSchema);

**models\verificationToken.js:**

const mongoose = require('mongoose');

const { Schema } = mongoose;

const verificationTokenSchema = new Schema({

  email: { type: String, required: true, lowercase: true, index: true },

  user: { type: Schema.Types.ObjectId, ref: 'User' },

  tokenHash: { type: String, required: true },

  purpose: { type: String, enum: ['signup', 'password'], default: 'signup' },

  campus: { type: String },

  expiresAt: { type: Date, required: true }

}, { timestamps: true });

verificationTokenSchema.index({ tokenHash: 1 });

module.exports = mongoose.model('VerificationToken', verificationTokenSchema);

**routes\adminRatings.js:**

const express = require('express');

const router = express.Router();

const adminRequire = require('../middleware/adminRequire');

*// admin dashboard for ratings*

router.get('/', adminRequire, (req, res) => {

  res.render('admin/ratings-dashboard', { title: 'Ratings Dashboard (Admin)' });

});

module.exports = router;

**routes\adminTimetable.js:**

const express = require('express');

const router = express.Router();

const timetableController = require('../controllers/timetableController');

const adminAuth = require('../middleware/adminAuth');

*// Manual add form: GET requires admin authentication (redirects to login if not authenticated)*

*// Manual add form: GET requires admin authentication (redirects to login if not authenticated)*

const adminRequire = require('../middleware/adminRequire');

router.get('/class/add', adminRequire, timetableController.renderAddClassForm);

router.post('/class/add', adminAuth, timetableController.addClassManually);

*// Batch upload (XLSX) to add classes in bulk — uses express-fileupload middleware (app.js)*

router.post('/class/upload', adminAuth, timetableController.addClassesFromXlsx);

*// Term management (admin)*

router.get('/terms', adminRequire, timetableController.listTerms);

router.post('/terms', adminAuth, timetableController.createTerm);

router.post('/terms/:id/activate', adminAuth, timetableController.activateTerm);

*// Promote a term to active and create the following next term (optional start/end dates)*

router.post('/terms/:id/promote', adminAuth, timetableController.promoteTerm);

router.post('/terms/:id/edit', adminAuth, timetableController.updateTerm);

*// Offerings management per term (list, edit, update, delete)*

router.get('/offerings', adminRequire, timetableController.listOfferingsByTerm);

router.get('/offerings/:id/edit', adminRequire, timetableController.renderOfferingEditForm);

router.post('/offerings/:id/edit', adminAuth, timetableController.updateOffering);

*// Replace POST delete with proper DELETE method for RESTful semantics.*

*// Keep route protected by adminAuth. Client pages will call DELETE /admin/offerings/:id.*

router.delete('/offerings/:id', adminAuth, timetableController.deleteOffering);

module.exports = router;

**routes\adminUsers.js:**

const express = require('express');

const router = express.Router();

const adminAuth = require('../middleware/adminAuth');

const adminRequire = require('../middleware/adminRequire');

const controller = require('../controllers/adminUsersController');

*// list*

router.get('/', adminRequire, controller.list);

*// new*

router.get('/new', adminRequire, controller.renderCreate);

router.post('/new', adminAuth, controller.create);

*// edit*

router.get('/:id/edit', adminRequire, controller.renderEdit);

router.post('/:id/edit', adminAuth, controller.update);

*// delete (use DELETE method for RESTful semantics)*

router.delete('/:id', adminAuth, controller.delete);

module.exports = router;

**routes\api-ratings.js:**

const express = require('express');

const router = express.Router();

const api = require('../controllers/ratingApiController');

const adminAuth = require('../middleware/adminAuth');

*// list ratings with filters*

router.get('/', api.list);

router.get('/summary', api.summary);

*// admin update*

router.put('/:id', adminAuth, api.adminUpdate);

module.exports = router;

**routes\auth.js:**

const express = require('express');

const router = express.Router();

const authController = require('../controllers/authController');

const auth = require('../middleware/auth');

*// POST /api/auth/signup*

router.post('/signup', authController.signup);

*// POST /api/auth/verify-signup*

router.post('/verify-signup', authController.verifySignup);

*// POST /api/auth/resend-signup-otp*

router.post('/resend-signup-otp', authController.resendSignupOtp);

*// POST /api/auth/login*

router.post('/login', authController.login);

*// GET /api/auth/me - return current authenticated user*

router.get('/me', auth, authController.me);

*// complete profile UI and API*

router.get('/complete-profile', authController.renderCompleteProfile);

*// allow authenticated POST to complete-profile; also supports an unauthenticated body-based flow*

router.post('/complete-profile', auth, authController.completeProfile);

*// profile view: allow GET with auth or ?email= query (rendering)*

router.get('/profile', auth, authController.viewProfile);

*// logout*

router.post('/logout', auth, authController.logout);

*// POST /api/auth/admin/login*

router.post('/admin/login', authController.adminLogin);

*// POST /api/auth/forgot-password*

router.post('/forgot-password', authController.forgotPassword);

module.exports = router;

**routes\dashboard.js:**

const express = require('express');

const router = express.Router();

const loginRequire = require('../middleware/loginRequire');

router.get('/ratings', loginRequire, (req, res) => {

  res.render('student/ratings-dashboard', { title: 'My Ratings', currentUserId: req.user ? req.user.\_id : null });

});

module.exports = router;

**routes\index.js:**

var express = require('express');

var router = express.Router();

*/\* GET home page. \*/*

router.get('/', function(req, res, next) {

  res.render('index', { title: 'Express' });

});

module.exports = router;

**routes\ratings.js:**

const express = require('express');

const router = express.Router();

const ratingController = require('../controllers/ratingController');

const loginRequire = require('../middleware/loginRequire');

*// render rating form (student must be logged in)*

router.get('/form', loginRequire, ratingController.renderForm);

*// list offerings for student to pick and give a rating*

router.get('/give', loginRequire, ratingController.renderGiveList);

*// create rating (student)*

router.post('/', loginRequire, ratingController.create);

*// update rating (student, owner, within 7 days)*

router.put('/:id', loginRequire, ratingController.update);

*// view ratings for offering (login required now)*

router.get('/offering/:offering', loginRequire, ratingController.viewForOffering);

*// ratings index (login required)*

router.get('/', loginRequire, ratingController.viewForOffering);

*// render edit form for an existing rating (student owner, within edit window)*

router.get('/:id/edit', loginRequire, ratingController.renderEditForm);

module.exports = router;

**routes\users.js:**

var express = require('express');

var router = express.Router();

*/\* GET users listing. \*/*

router.get('/', function(req, res, next) {

  res.send('respond with a resource');

});

module.exports = router;

**routes\views.js:**

const express = require('express');

const router = express.Router();

const authController = require('../controllers/authController');

router.get('/signup', (req, res) => res.render('signup', { title: 'Sign Up' }));

router.get('/login', (req, res) => res.render('login', { title: 'Login' }));

router.get('/forgot-password', (req, res) => res.render('forgot-password', { title: 'Forgot Password' }));

const adminRequire = require('../middleware/adminRequire');

const loginRequire = require('../middleware/loginRequire');

router.get('/admin/login', (req, res) => res.render('admin-login', { title: 'Admin Login' }));

*// Use manual class add page instead of timetable upload*

router.get('/admin/class/add', adminRequire, (req, res) => res.render('admin-add-class', { title: 'Add Class', user: req.user }));

router.get('/verify-signup', (req, res) => {

    const email = req.query.email || '';

    res.render('verify-signup', { title: 'Verify Account', email });

});

router.get('/complete-profile', loginRequire, authController.renderCompleteProfile);

router.get('/profile', loginRequire, authController.viewProfile);

module.exports = router;

**utils\email.js:**

const nodemailer = require('nodemailer');

const getTransporter = () => {

  if (process.env.SMTP\_HOST && process.env.SMTP\_USER) {

    return nodemailer.createTransport({

      host: process.env.SMTP\_HOST,

      port: Number(process.env.SMTP\_PORT) || 587,

      secure: false,

      auth: { user: process.env.SMTP\_USER, pass: process.env.SMTP\_PASS }

    });

  }

*// fallback: jsonTransport (does not deliver emails externally)*

  return nodemailer.createTransport({ jsonTransport: true });

};

exports.send = async ({ to, subject, text, html }) => {

*// If SMTP isn't configured, throw so callers (signup) can fall back to dev behaviour*

  if (!process.env.SMTP\_HOST || !process.env.SMTP\_USER || !process.env.SMTP\_PASS) {

    throw new Error('SMTP not configured. Set SMTP\_HOST, SMTP\_USER and SMTP\_PASS in your .env to send real emails.');

  }

  const transporter = getTransporter();

  const info = await transporter.sendMail({ from: process.env.SMTP\_FROM || 'noreply@example.com', to, subject, text, html });

  return info;

};

**utils\emailProgramMap.js:**

const { Department, Program } = require('../models');

*// mapping of degree short codes (as appear in CUI email local-part) to department/program names*

*// extend this object as needed*

const MAP = {

  baf: { department: 'Management Sciences', program: 'Accounting and Finance' },

  bag: { department: 'Environmental Sciences', program: 'Agriculture' },

  bba: { department: 'Management Sciences', program: 'Business Administration' },

  bcs: { department: 'Computer Science', program: 'Computer Science' },

  bec: { department: 'Economics', program: 'Economics' },

  bed: { department: 'Education', program: 'Education' }, *// not listed, assumed placeholder*

  ben: { department: 'Computer Science', program: 'Computer Engineering' }, *// assumed*

  bes: { department: 'Environmental Sciences', program: 'Environmental Sciences' },

  bmd: { department: 'Mathematics', program: 'Mathematics with Data Science' },

  bse: { department: 'Computer Science', program: 'Software Engineering' },

  bsm: { department: 'Mathematics', program: 'Mathematics' },

  bty: { department: 'Biotechnology', program: 'Biotechnology' }

};

*/\*\**

*\* Given a normalized email (lowercase), attempt to extract the degree short code*

*\* from the CUI Vehari pattern and resolve or create Department and Program.*

*\* Returns null if no mapping is found.*

*\**

*\* Example: fa22-bse-031@cuivehari.edu.pk -> degreeShort 'bse' -> map to Computer Science / Software Engineering*

*\*/*

async function resolveByEmail(email) {

  if (!email || typeof email !== 'string') return null;

  const m = email.toLowerCase().match(/^(?:fa|sp)\d{2}-([a-z]{2,4})-\d{3}@cuivehari\.edu\.pk$/i);

  if (!m) return null;

  const degree = m[1].toLowerCase();

  const entry = MAP[degree];

  if (!entry) return null;

*// find or create department*

  let dept = await Department.findOne({ name: entry.department });

  if (!dept) {

    dept = new Department({ name: entry.department });

    await dept.save();

  }

*// find or create program under department*

  let prog = await Program.findOne({ name: entry.program, department: dept.\_id });

  if (!prog) {

    prog = new Program({ name: entry.program, department: dept.\_id });

    await prog.save();

  }

  return { departmentId: dept.\_id, programId: prog.\_id, departmentName: dept.name, programName: prog.name };

}

module.exports = { resolveByEmail, MAP };

**utils\jwt.js:**

const jwt = require('jsonwebtoken');

const JWT\_SECRET = process.env.JWT\_SECRET || 'change-this-secret';

exports.verify = (token) => {

  return jwt.verify(token, JWT\_SECRET);

};

**utils\tokenBlacklist.js:**

*// Simple in-memory token blacklist with expiry. Not persistent across restarts.*

const map = new Map(); *// token -> expiry (ms)*

function add(token, expMs) {

  if (!token) return;

  const now = Date.now();

  const ttl = Math.max(0, expMs - now);

  map.set(token, expMs);

*// schedule removal*

  setTimeout(() => map.delete(token), ttl + 1000);

}

function isBlacklisted(token) {

  if (!token) return false;

  const exp = map.get(token);

  if (!exp) return false;

  if (Date.now() >= exp) { map.delete(token); return false; }

  return true;

}

module.exports = { add, isBlacklisted };

**app.js:**

var createError = require('http-errors');

var express = require('express');

var path = require('path');

var cookieParser = require('cookie-parser');

var logger = require('morgan');

const fileUpload = require('express-fileupload');

const fs = require('fs');

*// load environment*

require('dotenv').config();

*// Log whether GenAI/Gemini key is present (do not print the key itself)*

const hasGenAIKey = !!(process.env.GENAI\_API\_KEY || process.env.GEMINI\_API\_KEY);

console.log('GenAI key present:', hasGenAIKey);

*// connect to MongoDB*

const mongoose = require('mongoose');

mongoose.set('strictQuery', false);

const MONGO\_URI = process.env.MONGO\_URI || 'mongodb://localhost:27017/review-app';

mongoose.connect(MONGO\_URI, { useNewUrlParser: true, useUnifiedTopology: true })

  .then(() => console.log('Connected to MongoDB'))

  .catch(err => {

    console.error('MongoDB connection error:', err);

*// exit if DB is required*

    process.exit(1);

  });

var indexRouter = require('./routes/index');

var usersRouter = require('./routes/users');

var authApiRouter = require('./routes/auth');

var adminTimetableRouter = require('./routes/adminTimetable');

var adminUsersRouter = require('./routes/adminUsers');

var viewsRouter = require('./routes/views');

var ratingsRouter = require('./routes/ratings');

var apiRatingsRouter = require('./routes/api-ratings');

var adminRatingsRouter = require('./routes/adminRatings');

var dashboardRouter = require('./routes/dashboard');

var app = express();

*// express-ejs-layouts*

const expressLayouts = require('express-ejs-layouts');

app.use(expressLayouts);

*// view engine setup*

app.set('views', path.join(\_\_dirname, 'views'));

app.set('view engine', 'ejs');

*// ensure uploads dir exists*

const uploadsDir = path.join(\_\_dirname, 'public', 'uploads');

if (!fs.existsSync(uploadsDir)) fs.mkdirSync(uploadsDir, { recursive: true });

*// ensure tmp dir for express-fileupload temp files exists*

const tmpDir = path.join(\_\_dirname, 'tmp');

if (!fs.existsSync(tmpDir)) fs.mkdirSync(tmpDir, { recursive: true });

*// file upload middleware: increase limit to 10MB and use temp files for stability*

app.use(fileUpload({ limits: { fileSize: 10 \* 1024 \* 1024 }, useTempFiles: true, tempFileDir: path.join(\_\_dirname, 'tmp') }));

app.use(logger('dev'));

app.use(express.json());

app.use(express.urlencoded({ extended: false }));

app.use(cookieParser());

app.use(express.static(path.join(\_\_dirname, 'public')));

app.use('/', indexRouter);

app.use('/users', usersRouter);

app.use('/api/auth', authApiRouter);

*// Mount admin routes (manual class add). Upload functionality has been removed.*

app.use('/admin', adminTimetableRouter);

app.use('/admin/users', adminUsersRouter);

app.use('/ratings', ratingsRouter);

app.use('/api/ratings', apiRatingsRouter);

app.use('/admin/ratings', adminRatingsRouter);

app.use('/dashboard', dashboardRouter);

app.use('/', viewsRouter);

*// catch 404 and forward to error handler*

app.use(function(req, res, next) {

  next(createError(404));

});

*// error handler*

app.use(function(err, req, res, next) {

*// set locals, only providing error in development*

  res.locals.message = err.message;

  res.locals.error = req.app.get('env') === 'development' ? err : {};

*// log full stack for easier debugging*

  console.error(err && err.stack ? err.stack : err);

*// render the error page*

  res.status(err.status || 500);

  res.render('error');

});

module.exports = app;

**package.json:**

{

  "name": "review-app",

  "version": "0.0.0",

  "private": true,

  "scripts": {

    "start": "node ./bin/www",

    "seed-admin": "node scripts/seed-admin.js"

  },

  "dependencies": {

    "@google/genai": "^1.26.0",

    "bcrypt": "^5.1.0",

    "cookie-parser": "~1.4.4",

    "debug": "~2.6.9",

    "dotenv": "^16.0.0",

    "ejs": "^3.1.10",

    "express": "^4.21.2",

    "express-ejs-layouts": "^2.5.1",

    "express-fileupload": "^1.4.0",

    "http-errors": "~1.6.3",

    "jsonwebtoken": "^9.0.0",

    "mongoose": "^7.0.0",

    "morgan": "^1.10.1",

    "multer": "^1.4.5-lts.1",

  "nodemailer": "^7.0.9",

  "exceljs": "^4.3.0"

  },

  "devDependencies": {

    "nodemon": "^3.1.10"

  }

}