Task 1 - Project Description: Online Learning Platform

For this group project, we are building an Online Learning Platform that aims to support a complete digital learning experience for students and instructors. The platform is designed as a web-based system where instructors can create and manage courses, publish lessons and exams, and monitor students' performance. Students can browse available courses and enroll. Administrators are responsible for overseeing the system, managing users, and ensuring that all platform data remains consistent and secure.

To manage different types of users effectively, we implemented role-based access control (RBAC). The platform supports three main user roles:

- Students, who can enroll in courses, view lessons, complete assessments, and submit assignments.
- Instructors, who can create and manage their own courses, upload content, and evaluate student performance.
- Administrators, who have full access to manage users, monitor platform activities, and maintain data integrity.

Authentication and session control are handled using JWT (JSON Web Tokens), allowing for secure login and personalized access to system features based on user roles. Each user's permissions are enforced through backend middleware, and access restrictions are reflected in the API design.

On the database side, we use MongoDB Atlas as a cloud-hosted NoSQL solution. The data model includes five main collections:

- users: stores user information, including roles and profile data
- courses: contains course metadata and instructor relationships
- lessons: includes lesson content linked to specific courses
- enrollments: tracks student-course relationships and progress
- exams: stores assessment information and student results

These collections are designed using a mix of embedded documents and references to allow flexibility, maintain normalization where needed, and support future scalability. We also included nested structures (e.g., user preferences, course settings) and proper indexing to optimize performance and query speed.

The frontend of the platform provides an interactive web interface that allows users to seamlessly engage with the system based on their roles. Students can browse and enroll in courses, view lessons and assignments, and take exams. Instructors can manage courses, upload content, and evaluate student submissions. Administrators have access to system-wide controls and analytics. The frontend is designed to communicate with the backend APIs to deliver a responsive and user-friendly experience, with features such as real-time updates, visual dashboards, and dynamic user interactions.

Although the frontend is still under development, we have prepared the backend to support real-time updates, analytics, and future visual dashboards. Planned enhancements include discussion forums, student leaderboards, performance reports, and recommendation systems based on course activity and outcomes.

Task 1 - MongoDS Schema Design for Online Learning Platform

1. users Collection

Stores information for all users including students, instructors, and admins.

Key Fields:

- firstName, lastName, email, password
- role: "student" | "instructor" | "admin"
- profile: nested document (avatar, bio, address, etc.)
- preferences: notification and localization settings
- stats: course activity stats (coursesEnrolled, totalPoints)
- emailVerified, lastLogin, isActive
- createdAt, updatedAt (via timestamps: true)

Example Document:


```
{
  "firstName": "Alice",
  "lastName": "Chen",
  "email": "alice@example.com",
  "password": "hashedpassword",
  "role": "student",
  "profile": {
    "avatar": "https://img.com/avatar.jpg",
    "bio": "CS major at NU",
    "address": { "city": "Boston", "country": "USA" }
  },
  "preferences": {
    "language": "en",
    "timezone": "UTC"
  },
  "stats": {
    "coursesEnrolled": 3,
    "totalPoints": 90
  },
  "emailVerified": true,
  "lastLogin": "2024-07-09T10:00:00.000Z",
  "isActive": true,
  "createdAt": "2024-07-01T10:00:00.000Z",
  "updatedAt": "2024-07-10T12:00:00.000Z"
}
```


2. courses Collection

Holds metadata for each course created by instructors.

Key Fields:

- title, description, category, level, price
- instructor: reference to users._id
- syllabus: course outline URL
- resources: array of supporting materials
- settings: visibility and config flags
- stats: enrollment count, average rating
- createdAt, updatedAt

Example Document:

```
{
  "title": "Intro to Python",
  "description": "Beginner Python course",
  "instructor": "ObjectId('64a123...')",
  "category": "Programming",
  "level": "Beginner",
  "price": 49,
  "syllabus": "https://example.com/syllabus.pdf",
  "resources": [
    { "type": "pdf", "url": "https://cdn.com/resource1.pdf" }
  ],
  "settings": { "isPublished": true },
  "stats": {
    "enrollmentCount": 120,
    "averageRating": 4.5
  },
  "createdAt": "2024-07-01T12:00:00.000Z",
  "updatedAt": "2024-07-10T12:00:00.000Z"
}
```

3. lessons Collection

Lessons belong to courses and include various types: video, quiz, assignment, etc.

Key Fields:

- title, description, order
- course: reference to courses._id
- instructor: reference to users._id
- type: "video" | "text" | "quiz" | ...
- content: embedded content per type
- resources: additional materials
- stats: views, likes
- createdAt, updatedAt

Example Document:

```
{
  "title": "Variables and Data Types",
  "description": "Covers basic Python variables",
  "course": "ObjectId('64a456...')",
```

```
  "instructor": "ObjectId('64a123...')",
  "order": 1,
  "type": "video",
  "content": {
    "videoUrl": "https://cdn.com/video.mp4",
    "videoDuration": 540
  },
  "resources": [
    { "type": "image", "url": "https://cdn.com/img1.png" }
  ],
  "stats": { "views": 300 },
  "createdAt": "2024-07-01T12:00:00.000Z",
  "updatedAt": "2024-07-10T12:00:00.000Z"
}
```


4. exams Collection

Assessments linked to specific courses and optionally lessons.

Key Fields:

- title, description, type: "quiz" | "assignment" | "final"
- course, instructor: reference fields
- questions: embedded array of question objects
- settings: time limit, attempt rules, review settings
- availability: availability status
- stats: attemptCount, averageScore
- createdAt, updatedAt

Example Document:


```
{
  "title": "Python Basics Quiz",
  "description": "Quiz on variables and basic syntax",
  "course": "ObjectId('64a456...')",
  "instructor": "ObjectId('64a123...')",
  "type": "quiz",
  "questions": [
    {
      "question": "What is the output of print(2 + 2)?",
      "type": "multiple-choice",
      "options": [
        { "text": "3", "isCorrect": false },
        { "text": "4", "isCorrect": true }
      ]
    }
  ],
  "settings": {
    "timeLimit": 15,
    "maxAttempts": 3
  },
```

```
  "availability": { "isAvailable": true },
  "stats": { "attemptCount": 200 },
  "createdAt": "2024-07-01T12:00:00.000Z",
  "updatedAt": "2024-07-10T12:00:00.000Z"
}
```


5. enrollments Collection

Tracks student enrollment, progress, payments, and course review.

Key Fields:

- student: reference to users._id (with role: "student")
- course: reference to courses._id
- status: "enrolled" | "completed" | "dropped"
- progress: nested object tracking progress
- paymentDetails: method, amount
- certificate: issued status and date
- rating, review: feedback
- createdAt, updatedAt

Example Document:


```
{
  "student": "ObjectId('64a111...')",
  "course": "ObjectId('64a456...')",
  "status": "completed",
  "progress": {
    "completionPercentage": 100,
    "totalTimeSpent": 200,
    "lessonsCompleted": ["ObjectId('lesson1')", "ObjectId('lesson2')"],
    "examsCompleted": ["ObjectId('exam1')"]
  },
  "paymentDetails": {
    "method": "credit_card",
    "amountPaid": 49
  },
  "certificate": {
    "issued": true,
    "issueDate": "2024-07-10"
  },
  "rating": 5,
  "review": {
    "comment": "Great course!"
  },
  "createdAt": "2024-07-01T12:00:00.000Z",
  "updatedAt": "2024-07-10T12:00:00.000Z"
}
```

Relationships Overview

In our project, we designed the database schema using reference fields to clearly define the relationships between collections. Each course in the courses collection includes an instructor field that references a document in the users collection, indicating which user created the course. Similarly, both lessons and exams have their own instructor fields pointing to the same users collection.

Each lesson and exam also references a course through the course field, allowing us to group them under the correct course. This makes it easy to retrieve all the related lessons and exams for any course when building course content pages.

For student activity tracking, the enrollments collection connects each student (via student, referencing users) to the course they are taking (via course, referencing courses). Inside the progress object, we store arrays like lessonsCompleted and examsCompleted, which reference the IDs of completed documents in the lessons and exams collections.

Some exams also have a prerequisites field, which references lessons that must be completed before taking the exam.

Overall, this reference-based approach allows us to build a flexible and efficient structure that supports features like progress tracking, user-course relationships, and lesson-based assessments. It also keeps the schema organized and easy to work with when writing queries or expanding functionality in future iterations of the platform.

Task 2 - API Documentation for Online Learning Platform

This document outlines the API endpoints implemented for the Online Learning Platform. Each section describes the purpose, HTTP method, URL path, expected input (via URL params, query, or request body), and the expected response format. The APIs are grouped by resource type: Authentication, Courses, Lessons, Enrollments, Exams, Users.

1. Authentication API

Base Route: /api/auth

POST /register - Register a new user (Create)

Request:

```
{
  "firstName": "Alice",
  "lastName": "Wang",
  "email": "alice@example.com",
  "password": "password123",
  "role": "student"
}
```

Response:

```
{
  "success": true,
  "data": {
          "user": {
          "id": "123",
          "email": "alice@example.com",
          "role": "student"
          },
          "token": "JWT_TOKEN"
          }
}
```

POST /login - Authenticate a user (Read/Authenticate)

Request:

```
{
  "email": "alice@example.com",
  "password": "password123"
}
```

Response:

```
{
  "success": true,
  "data": {
          "user": {...},
          "token": "JWT_TOKEN"
  }
}
```

2. Courses API

Base Route: /api/courses

GET / - Retrieve all courses (Read)
POST / - Create a new course (Create)
GET /:id - Retrieve course by ID (Read)
PUT /:id - Update a course (Update)
DELETE /:id - Delete a course (Delete)

3. Lessons API

Base Route: /api/lessons
GET /?course= - Retrieve lessons by course (Read)
POST / - Create a lesson (Create)
GET /:id - Retrieve lesson by ID (Read)
PUT /:id - Update a lesson (Update)
DELETE /:id - Delete a lesson (Delete)

4. Enrollments API

Base Route: /api/enrollments
GET / - Retrieve student enrollments (Read)
PUT /:id/progress - Update progress (Update)
DELETE /:id - Unenroll from course (Delete)

5. Exams API

Base Route: /api/exams
GET /?course= - Retrieve exams for course (Read)
POST / - Create an exam (Create)
PUT /:id - Update an exam (Update)
DELETE /:id - Delete an exam (Delete)
POST /:id/submit - Submit exam answers (Create)

6. Users API

Base Route: /api/users
GET / - Retrieve all users (Read, Admin only)
GET /:id - Retrieve user by ID (Read, self or admin)
POST / - Create new user (Create, Admin only)
PUT /:id - Update user information (Update)
DELETE /:id - Delete user (Delete, Admin only)
GET /:id/stats - Retrieve user statistics (Read)

Example: GET /api/users/64a8...
Response:


```
{
  "success": true,
  "data": {
          "user": {
          "id": "64a8...",
          "firstName": "Alice",
```

```
        "lastName": "Wang",
        "role": "student",
        "stats": { ... }
        }
  }
}
```


CRUD Operations Summary

The API implements full or partial CRUD functionality across all key resources in the Online Learning Platform:

- Authentication: Create (Register), Read (Login)
- Users: Create, Read, Update, Delete
- Courses: Create, Read, Update, Delete
- Lessons: Create, Read, Update, Delete
- Enrollments: Create (via /courses/:id/enroll), Read, Update, Delete
- Exams: Create, Read, Update, Delete

This structure ensures a modular, secure, and scalable API design, with a consistent pattern for handling resources and user interactions in the Online Learning Platform.