



NATIONAL INSTITUTE OF BUSINESS MANAGEMENT

School of Computing

Higher National Diploma in Information Systems

Batch – KAHNDISM24.1F

Software Quality Engineering - Coursework

Module Lecturer: Mr. Niranga Dharmarathna

Group Members	
Index	Name
KAHNDISM241F-002(Backend Developer)	A.DESHMIGA
KAHNDISM241F-011 (DevOps Engineer)	W.M.S.P JAYARATHNE
KAHNDISM241F-012 (Project Manager)	M.G.K.E.WADUGODA
KAHNDISM241F-015 (API Developer)	D.H.A. GOONASEKERE
KAHNDISM241F-017(Quality Assurance)	H.P.A.S.M. KUMARI
Submission Date:	28/03/2025

Ethical Declaration of Original Work

We declare that the work presented in this coursework is entirely our own. We confirm that:

1. The work presented in this coursework is conducted by us, and any contributions from others are appropriately acknowledged.
2. Any external sources of information and ideas used in this work are cited and referenced accurately. We have provided proper credit to the original authors through citations in the text and a comprehensive list of references.
3. The data and findings presented in this work are genuine and have not been manipulated or fabricated. Any assistance received in the collection and analysis of data is acknowledged appropriately.
4. We have not submitted this work, or any part of it, for any other academic qualification.
5. We understand the ethical principles governing academic work, including honesty, integrity, and accountability. We have adhered to these principles throughout the process.

We are aware of the consequences of academic misconduct and understand that any violation of ethical standards may result in disciplinary action.

28/03/2025

Index #	Name	Signature
KAHNDISM241F-002	A. DESHMIGA	
KAHNDISM241F-011	W.M.S.P JAYARATHNE	
KAHNDISM241F-012	M.G.K.E. WADUGODA	
KAHNDISM241F-015	D.H.A. GOONASEKERE	
KAHNDISM241F-017	H.P.A.S.M. KUMARI	

Contents

Ethical Declaration of Original Work.....	2
CHAPTER 1: INTRODUCTION	4
1.1 Overview	5
1.2 Objectives.....	5
1.3 Scope of the Project.....	5
1.4 Team Structure & Responsibilities	6
CHAPTER 2: BUSINESS ANALYSIS.....	8
2.1 Business Requirements	8
2.2 Stakeholders.....	8
CHAPTER 3: SOFTWARE REQUIREMENT SPECIFICATIONS (SRS)	9
3.1 Functional Requirements	9
3.2 Non-Functional Requirements	10
3.3 External Interface Requirements	10
CHAPTER 4: PROJECT MANAGEMENT (KAHNDISM241F-012).....	11
4.1 Project Planning & Timeline.....	11
3.3 Task Assignments & Team Coordination.....	11
3.4 Challenges & Risk Management.....	12
CHAPTER 5: BACKEND API DEVELOPMENT (API DEVELOPERS: KAHNDISM241F-002/KAHNDISM241F-015)	13
4.1 Overview of API Design.....	13
4.2 Database Development (DB Dev).....	19
4.3 Issues Encountered and Resolutions	28
Chapter 6: Quality Assurance Testing (QA Engineer – KAHNDISM241F – 017).....	30
6.1 Load Test Plan	30
6.2 Test Plan for Spike Testing.....	31
6.3 API Test / Integration Test	34
6.4 Edge Case	36
6.5 Unit Test	38
Chapter 7: DEVOPS & DEPLOYMENT (DEVOPS ENGINEER: KAHNDISM241F-011).....	42
References and Appendices	Error! Bookmark not defined.

List Of Figures

Figure 1: Retrieve all courses	14
Figure 2: Add a new course	15
Figure 3: Update course by name request and response	16
Figure 4: Delete course by name	17
Figure 5:200OK	18
Figure 6:201 CREATED.....	18
Figure 7:404 NOT FOUND	19
Figure 8: 500 internal server error	19
Figure 9:Get all course by name	23
Figure 10:Defect reporting.....	
Figure 11:integration test	35
Figure 12:negative test.....	36
Figure 13:Edge case.....	

CHAPTER 1: INTRODUCTION

1.1 Overview

In the modern digital era, educational institutions increasingly rely on web-based solutions to streamline course management and enhance accessibility for students and faculty. Managing course data efficiently is crucial for academic institutions to provide up-to-date information regarding courses, modules, and related resources. Traditional methods of course management often involve manual data handling, which can lead to inefficiencies, errors, and delays in information dissemination.

To address these challenges, institutions are transitioning toward API-driven architectures that facilitate seamless integration between different systems and ensure efficient data management. A well-structured API enables educational institutions to automate course data retrieval, updates, and distribution while ensuring security and scalability.

This project focuses on developing a headless API for the National Institute of Business Management (NIBM), leveraging FastAPI, Node.js, and MongoDB Atlas to store and manage course-related data efficiently. The API is designed to integrate seamlessly with the NIBM website and other external systems to provide a dynamic and scalable solution for managing course information.

1.2 Objectives

The objectives of this project are as follows:

- Develop a robust, high-performance RESTful API to handle course data efficiently.
- Enable CRUD operations to allow users to create, read, update, and delete course information.
- Ensure secure data storage by integrating with MongoDB Atlas and implementing authentication mechanisms.
- Maintain optimal API performance by leveraging cloud hosting solutions and scalable architecture.
- Implement structured testing and quality assurance to ensure the reliability and functionality of the API.

1.3 Scope of the Project

The scope of this project includes:

- Development of a RESTful API to manage course-related data.
- Implementation of CRUD operations for course retrieval, addition, modification, and deletion.
- Integration with MongoDB Atlas as the cloud-based database solution.
- Implementation of API security measures to prevent unauthorized access.
- Deployment using DevOps practices, including CI/CD pipelines and cloud hosting.
- Testing strategies for API validation, performance, and security.

This project excludes the development of a front-end user interface, as the focus is on providing a headless API for integration with existing systems.

1.4 Team Structure & Responsibilities

ROLE	KEY RESPONSIBILITIES
Project Manager (PM)	<ul style="list-style-type: none">• Ensured alignment with project requirements, technical guidelines, and simplicity principles (no frontend).• Managed team coordination, task allocation, and deadlines.• Oversaw GitHub activities (issues, commits, pull requests) to track contributions.• Addressed gaps in project execution by resolving bottlenecks and ensuring adherence to CI/CD workflows.
Backend & API Developers (2 Members)	<ul style="list-style-type: none">• Developed a scalable RESTful API using FastAPI and Node.js.• Implemented CRUD operations for course data, including database integration with MongoDB Atlas.• Designed modular, well-documented, and error-logged code.• Conducted unit and integration testing to validate API functionalities

DevOps Engineer	<ul style="list-style-type: none"> • Configured cloud hosting and deployment (AWS or equivalent). • Implemented CI/CD pipelines for continuous testing and automated deployment. • Ensured API availability, load balancing, and performance optimization. • Managed database backups, security policies, and deployment logs.
Quality Assurance (QA) Engineer	<ul style="list-style-type: none"> • Designed and executed 5 distinct test types (unit, integration, security, performance, load testing). • Used tools like Postman for API testing and security validation. • Documented test cases, identified defects, and ensured adherence to repeatable testing methodologies. • Verified API robustness and compliance with security protocols.

1.5 Link to the deployed system and GitHub repository

- GitHub repository: (<https://github.com/Deshmiga21/Course-Finder-API->)

CHAPTER 2: BUSINESS ANALYSIS

2.1 Business Requirements

The Course Management API is designed to fulfill several business requirements for NIBM, including:

- **Efficient Management of Course Data:** Streamline the process of storing and managing course data, enabling real-time updates and easier data retrieval.
- **Increased Accessibility:** Make course data accessible to students, faculty, and external systems for better integration and user experience.
- **Reduced Operational Costs:** Automate tasks like course updates, reducing manual errors and administrative overhead.
- **Scalability and Performance:** Ensure the system can handle large amounts of course data without performance degradation.
- **Security:** Safeguard sensitive academic data and prevent unauthorized access.

By meeting these requirements, the API will support NIBM in its ongoing digital transformation, enabling better management, visibility, and accessibility of its course offerings.

2.2 Stakeholders

The stakeholders in this project include:

- **NIBM Management:** Responsible for overseeing the course data system and ensuring it meets institutional needs.
- **Students and Faculty:** Primary users of the system, accessing and interacting with course data.
- **System Administrators:** Responsible for maintaining the API and ensuring its uptime.
- **Developers:** Will be responsible for integrating the API with NIBM's website and other systems.
- **External Partners:** May require integration with the course data system for external systems like student portals or online course providers.

CHAPTER 3: SOFTWARE REQUIREMENT SPECIFICATIONS (SRS)

3.1 Functional Requirements

FR1: Course Retrieval by Name

- Description: Users must be able to retrieve course details using the course name.
- Rationale: Allow students and faculty to search for course data efficiently by using the course name.

FR2: Response Format

- Description: The API must return course details in JSON format.
- Rationale: JSON format is widely used and easy to process by both human developers and machines.

FR3: Database Storage and Indexing

- Description: The system must store course data in a MongoDB Atlas database with indexing for optimized queries.
- Rationale: Storing data in MongoDB allows flexible schema design, and indexing improves query performance.

FR4: Data Refresh Endpoint

- Description: The API must provide an endpoint for refreshing the scraped data.
- Rationale: Ensure that the course data is always up-to-date by providing an endpoint to refresh the data from the source.

FR5: Course Fee and URL

- Description: The API must return course fees along with course details and a URL link.
- Rationale: It provides complete information regarding the course and enables users to access more details via the URL.

FR6: Authentication and Authorization

- Description: Authentication and authorization mechanisms must be implemented for access control.
- Rationale: Protect course data by restricting access to authorized users only, ensuring the system's security.

3.2 Non-Functional Requirements

NFR1: Response Time

- Description: The API must provide responses within 500ms under normal load.
- Rationale: Ensuring fast response times guarantees a smooth user experience when retrieving course data.

NFR2: Uptime

- Description: The API must be hosted on a cloud platform with 99.9% uptime.
- Rationale: High availability is essential to ensure the API is accessible at all times, minimizing downtime and disruptions.

NFR3: Security Measures

- Description: Security measures such as API rate limiting and encryption must be implemented.
- Rationale: Protect the system from abuse, unauthorized access, and data breaches by implementing encryption and limiting the number of requests.

3.3 External Interface Requirements

User Interfaces

- API Access: The API will be accessible via REST endpoints.
- Response Format: All responses will be provided in JSON format, which is easy to use and understand for developers and systems.

Software Interfaces

- Database: The system will use MongoDB Atlas for storing course data.
- API Development: The API will be developed using the FastAPI framework for high performance and ease of use.

Communication Interfaces

- API Access: All API endpoints will be accessible over HTTPS to ensure secure communication between clients and the server.

CHAPTER 4: PROJECT MANAGEMENT (KAHNDISM241F-012)

4.1 Project Planning & Timeline

Day	Objective	Tasks
Day 1-2	Project Kickoff & Requirement Gathering	Finalize project scope and objectives.
Day 3-4	API Design & Database Structure	Create detailed API endpoints (GET, POST, PUT, DELETE).
Day 5-6	Initial Backend Development (CRUD Operations)	Implement API endpoints for retrieving and adding courses.
Day 7-8	Security Implementation & Authentication	Implement user authentication (JWT, OAuth).
Day 9-10	Error Handling & Performance Optimization	Implement error handling and logging.
Day 11-12	DevOps Setup & Deployment Pipeline	Set up GitHub Actions for CI/CD pipeline.
Day 13	API Testing & Quality Assurance	Conduct functional, unit, and integration tests (Postman, Jest).
Day 14	Final Review & Project Documentation	Conduct final review meeting with the team.

3.3 Task Assignments & Team Coordination

Task Assignment Overview

Tasks were assigned based on team members' strengths and expertise, ensuring efficient progress. The tasks were tracked via project management tools like Jira or Trello for visibility and accountability.

Role-Based Task Assignment

- Project Manager (PM): Oversees project progress, sprint planning, communication, and risk management.
- Backend/API Developers (Members 2 & 3): Develop the API, implement CRUD operations, security, and performance optimization.
- Quality Assurance (QA) Engineer (Member 4): Conduct tests, identify bugs, and ensure quality.
- DevOps Engineer (Member 5): Manage cloud infrastructure, CI/CD pipelines, and deployment.

3.4 Challenges & Risk Management

Key Challenges

- Integration Issues: Difficulty in integrating MongoDB Atlas and syncing with the API.
- Security: Challenges with implementing secure authentication and encryption.
- Performance: Ensuring fast response times under load.
- QA Testing: Ensuring comprehensive test coverage for all scenarios.
- Deployment: Configuring cloud hosting and optimizing CI/CD pipelines.

Risk Management Strategies

- Task Breakdown: Dividing complex tasks into smaller parts for easier management.
- Frequent Testing & Code Reviews: Regular unit, integration, and performance testing.
- Security Best Practices: Using OAuth/JWT for authentication and encryption.
- Monitoring & Optimization: Continuous performance monitoring and tuning.
- Cloud Deployment: Setting up backup systems and auto-scaling for resilience.

CHAPTER 5: BACKEND API DEVELOPMENT (API DEVELOPERS: KAHNDISM241F-002/KAHNDISM241F-015)

4.1 Overview of API Design

1. API Architecture and Framework

The Course Management API is designed as a RESTful API built using Node.js and the Express.js framework. The API follows standard RESTful principles and supports CRUD (Create, Read, Update, Delete) operations for managing course data.

RESTful Principles: The API uses HTTP methods (GET, POST, PUT, DELETE) to interact with resources (course details) represented by URLs.

- GET: Retrieve course data (e.g., course name, modules, URL).
- POST: Add new course data to the system.
- PUT: Update an existing course record.
- DELETE: Remove course record.
- Data Format: All data exchanged with the API (requests and responses) is in JSON format.

2. API Endpoints

Endpoint	HTTP Method	Description
/Courses	GET	Retrieve a list of all courses.
/courses/name}	GET	Retrieve course details by name.
/Courses	POST	Add a new course to the system.
/courses/name}	PUT	Update an existing course by name.
/courses/name}	DELETE	Delete a course by name.
/courses/refresh	GET	Refresh course data

Retrieve All Courses

- Endpoint: GET /api/courses
- Description: This endpoint retrieves a list of all available courses in the system.

Response:

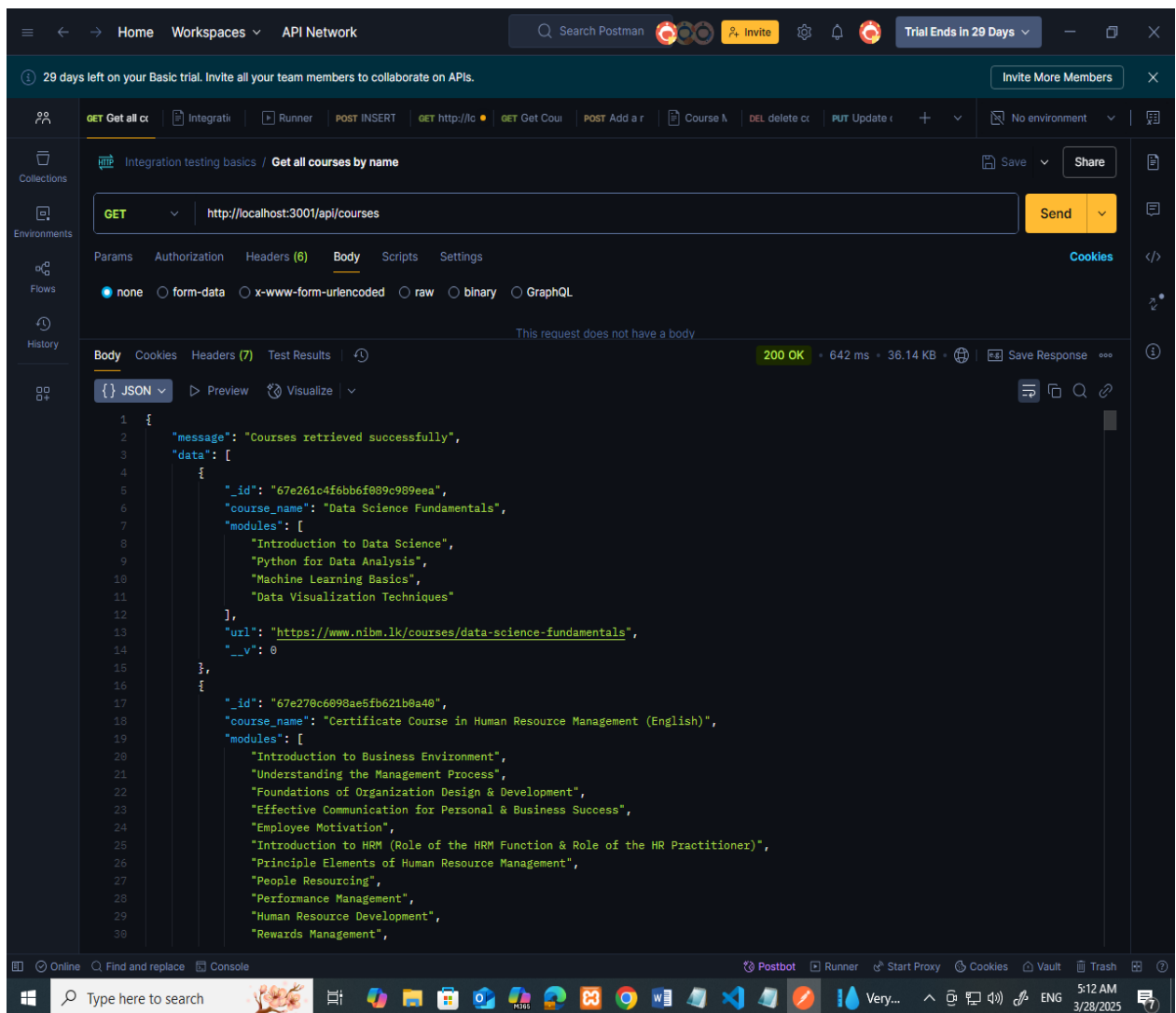


Figure 1: Retrieve all courses

Add a New Course

- Endpoint: POST `/api/courses`
- Description: This endpoint allows the user to add a new course to the system by providing course details in the request body.

Request Body and Response:

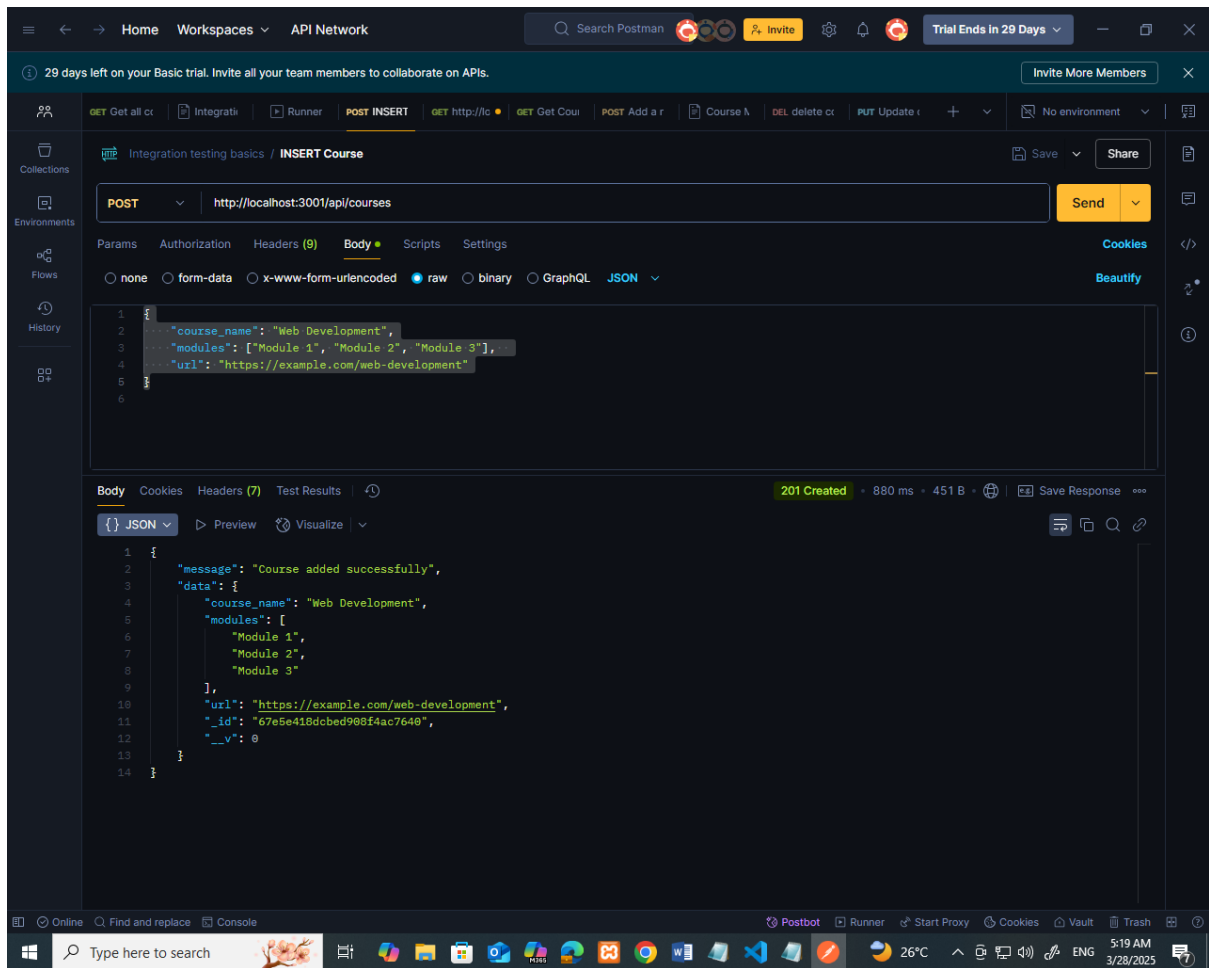


Figure 2: Add a new course

Update a Course

- Endpoint: PUT /api/courses/:course_name
- Description: This endpoint allows updating an existing course by its ID. The course name can be updated in the request body.

Request Body and Response:

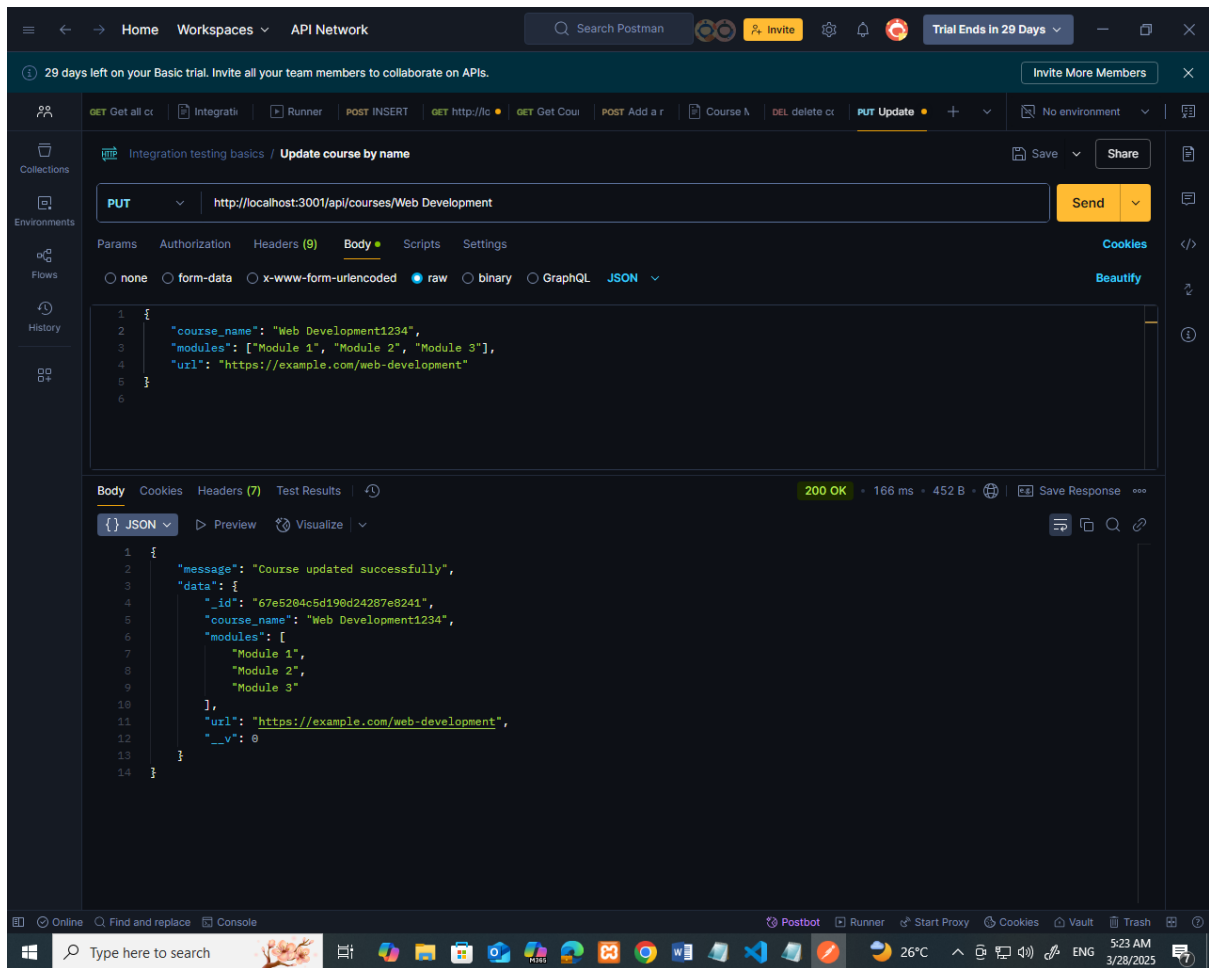


Figure 3: Update course by name request and response

Delete a Course

- Endpoint: DELETE `/api/courses/:course_name`
- Description: This endpoint allows deleting a course by its ID from the system.

Response:

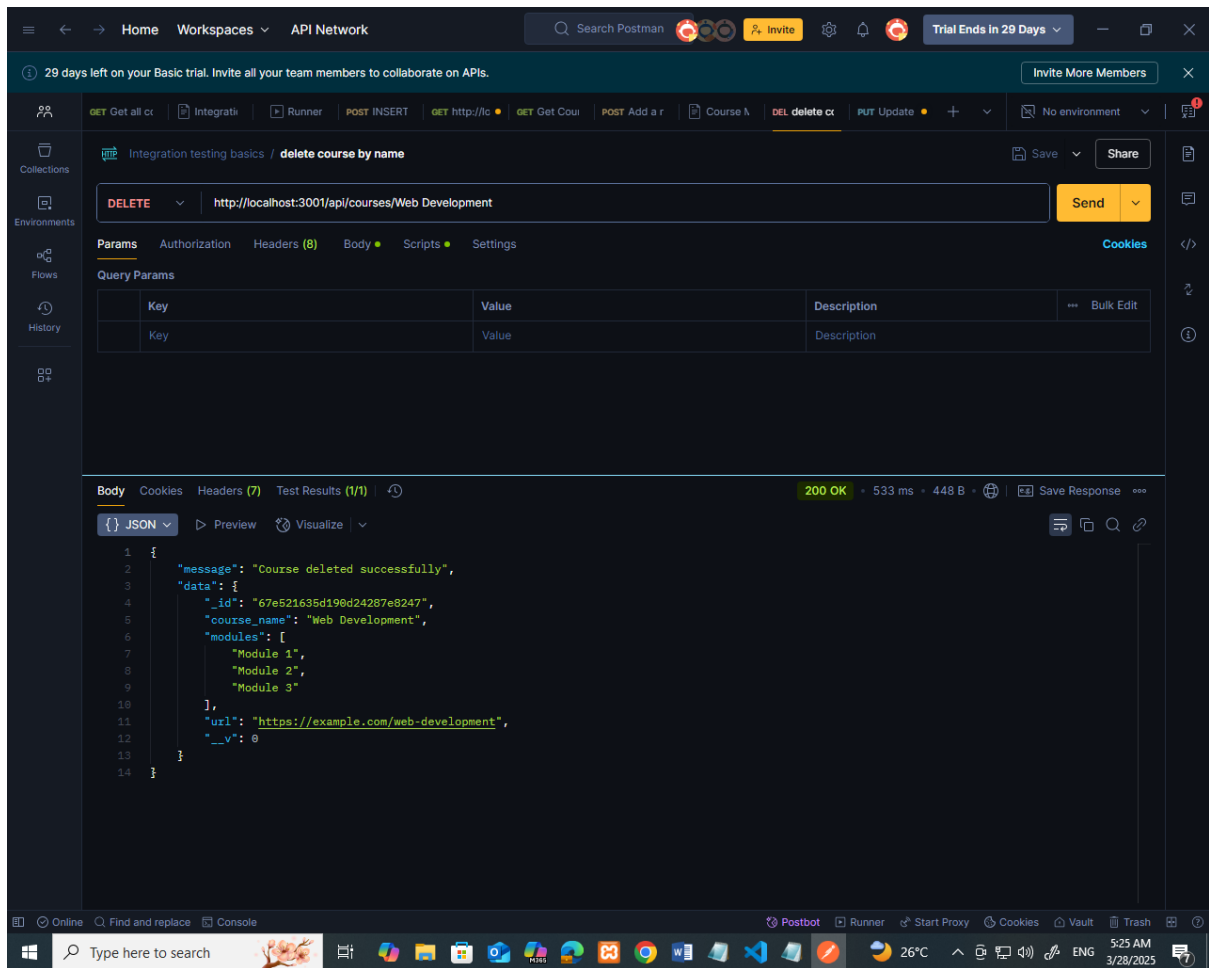


Figure 4: Delete course by name

NPM Commands

Command	Description
npm install	Installs all dependencies required for the project.
npm start	Starts the server and serves the API.
npm run dev	Starts the server in development mode with hot reloading enabled.
npm test	Runs test cases to verify the functionality of the API and system.

HTTP Status Codes

- **200 OK:** When the request is successful (e.g., retrieving course data).

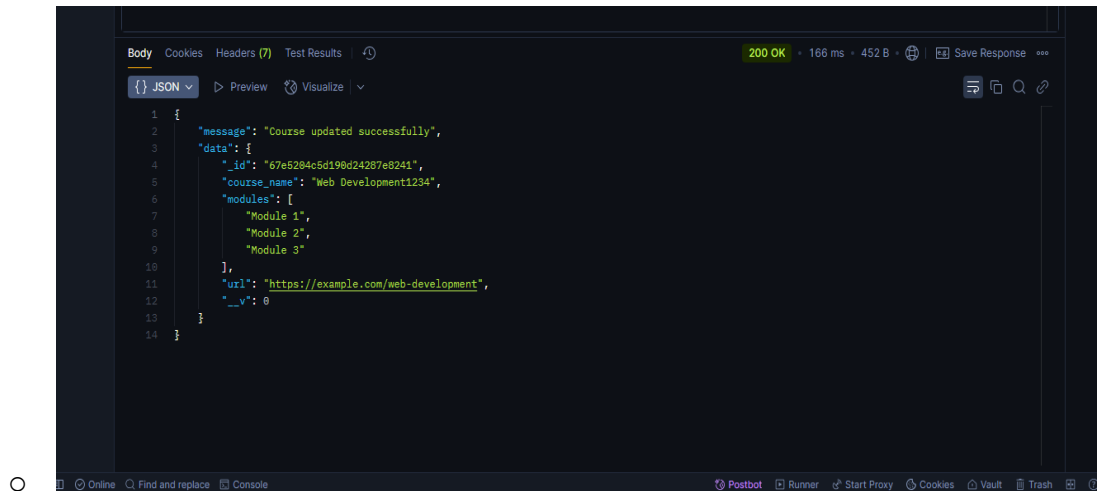


Figure 5:200OK

- **201 Created:** When a new resource is created (e.g., adding a new course).

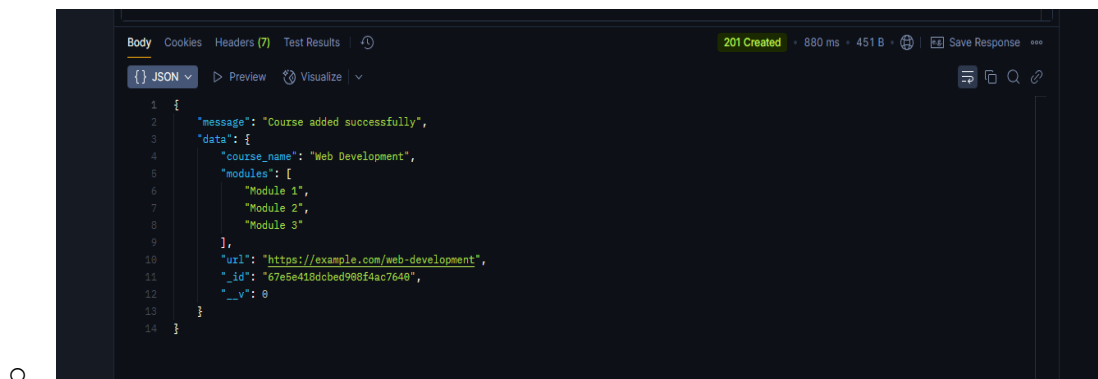


Figure 6:201 CREATED

- **404 Not Found:** If the resource (e.g., course) does not exist.

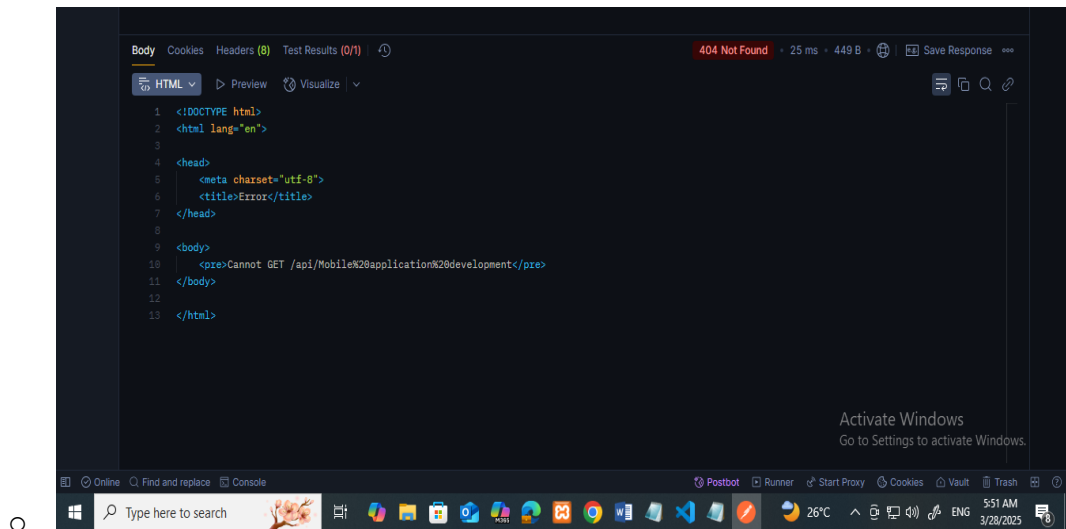


Figure 7:404 NOT FOUND

- **500 Internal Server Error: When something goes wrong on the server side.**

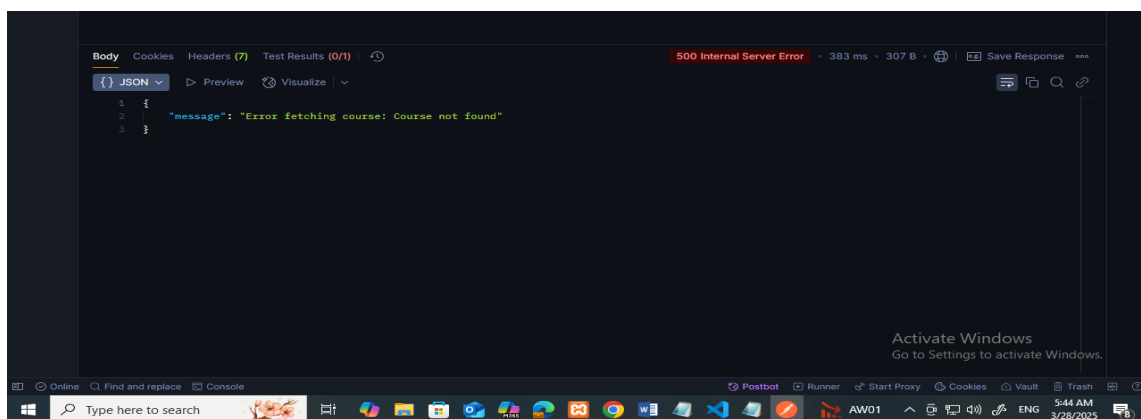


Figure 8: 500 internal server error

4.2 Database Development (DB Dev)

- Getting Started with MongoDB Atlas

MongoDB Atlas is a complete database service hosted in the cloud for the purposes of deploying, operating, and scaling in-the-cloud MongoDB databases. The presence of automated backup, monitoring performance in real time, and extending horizontally all combine to paint MongoDB Atlas as being highly suited to be the cloud storage for applications that require flexible and scalable storage. For the purposes of this project, course data would be modulated into the management of the Course API with MongoDB Atlas.

- Preparing the Database

The preparation of the database starts with the creation of a MongoDB Atlas account and goes on to cluster setup for the cloud-hosted site from which course data would be fetched. The following actions were undertaken:

- Account Creation: The MongoDB Atlas account was created at MongoDB Atlas
- Cluster Configuration: It creates new clusters in the AWS cloud region for optimal performance and security.
- Database Creating: Create a database called courses DB to hold all the information about the course introduction, course names, modules, and their URLs.
- User Access: Created a new user on the database and granted read/write permission for secure interaction with the database.
- IP Whitelisting: The local machine's IP address has been successfully whitelisted to allow secure access to the database.

- Database Connectivity

Connection to MongoDB Atlas is handled using Mongoose, a popular Object-Document Mapping library for Node.js. Below is a sample connection string used by the application to connect the Node.js backend with MongoDB Atlas:

- Database connection Server.js

```
const mongoose = require('mongoose');
const app = require('./app');
const dotenv = require('dotenv');

// Load environment variables
dotenv.config();

// Connect to MongoDB
mongoose.connect(process.env.MONGODB_URI)
```

```

.then(() => console.log('MongoDB connected'))
.catch((err) => console.error('MongoDB connection error:', err));

// Start the server
const PORT = process.env.PORT || 3001;
app.listen(PORT, () => {
  console.log(`Server running on port ${PORT}`);
});

```

Database Schema Design

```

const mongoose = require('mongoose');

// Define the Course Schema
const courseSchema = new mongoose.Schema({
  course_name: { type: String, required: true },
  modules: [{ type: String, required: true }], // Array of module names
  url: { type: String, required: true }, // URL for the course
});

// Create and export the Course model
module.exports = mongoose.model('Course', courseSchema);

```

CRUD Operations

MongoDB Atlas is being used by the Course API to execute actions through CRUD operations (Create, Read, Update, and Delete) on the courses. Some of the actions have the following implementations:

Create a Course (POST)

Endpoint: /api/courses

Method: POST

Request Body (JSON):

Response:

```
{
  "course_name": "Web Development",
  "modules": ["Module 1", "Module 2", "Module 3"],
  "url": "https://example.com/web-development"
}
```

Read All Courses (GET)

Endpoint:/api/courses

Method:GET

Response:

```
{
  "message": "Course added successfully",
  "data": {
    "course_name": "Web Development",
    "modules": [
      "Module 1",
      "Module 2",
      "Module 3"
    ],
    "url": "https://example.com/web-development",
    "_id": "67e5e418dcbed908f4ac7640",
    "__v": 0
  }
}
```

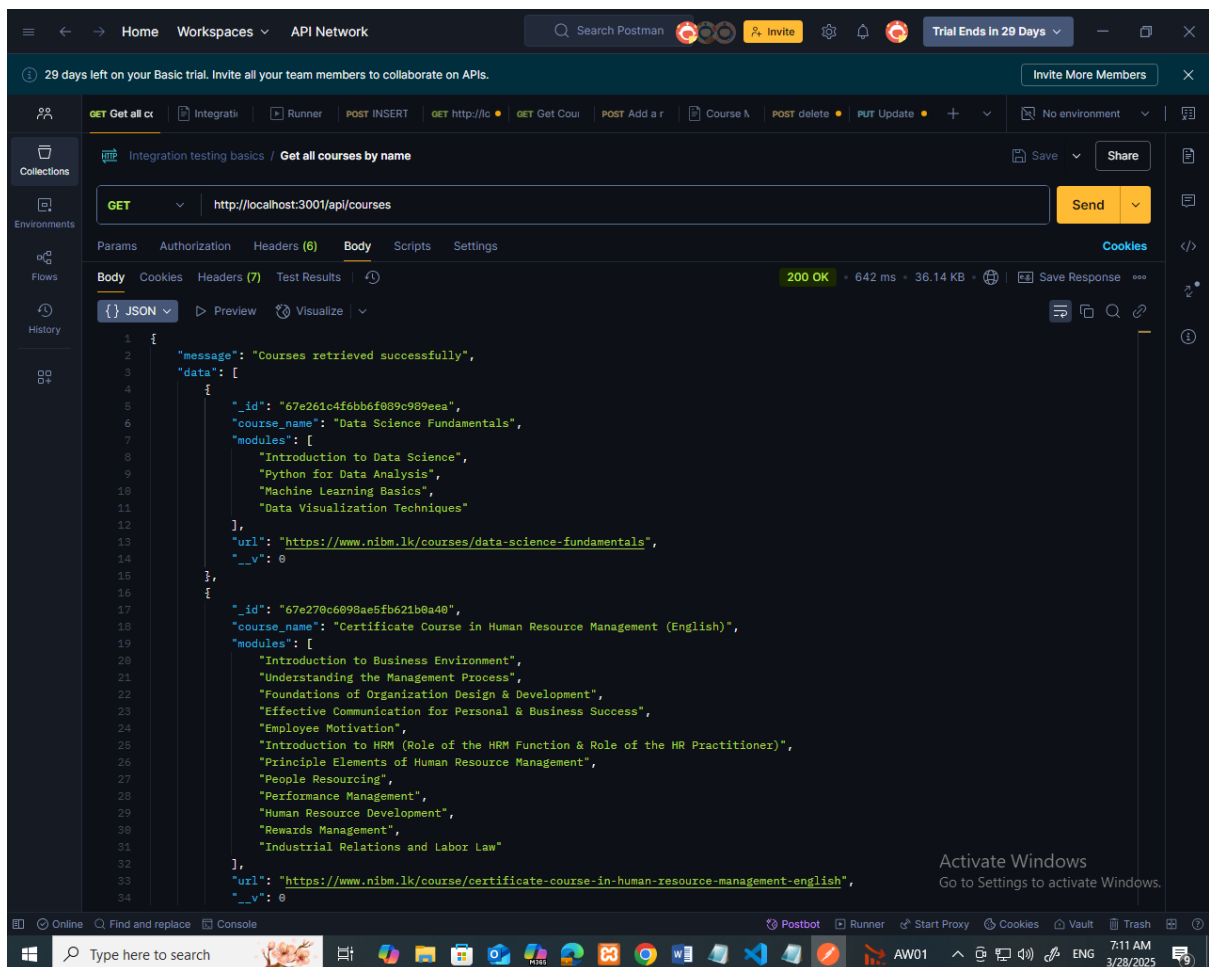


Figure 9: Get all course by name

Read a Course by Name (GET)

Endpoint: `/api/courses/:course_name`

Example

Request:

GET `/api/courses/ Data Science Fundamentals`

```
{
  "message": "Course retrieved successfully",
  "data": {
    "_id": "67e261c4f6bb6f089c989eea",
    "course_name": "Data Science Fundamentals",
    "modules": [
      "Introduction to Data Science",
      "Python for Data Analysis",
      "Machine Learning Basics",
      "Data Visualization Techniques"
    ],
    "url": "https://www.nibm.lk/courses/data-science-fundamentals",
    "__v": 0
  }
}
```

If the course does not exist:

```
{
  "message": "Error fetching course: Course not found"
}
```

Update a Course (PUT)

Endpoint: /api/courses/:course_name

Example

Request:

PUT /api/courses/JavaScript Basics

Request Body (JSON):

```
{  
  "course_name": "Web Development1234",  
  "modules": ["Module 1", "Module 2", "Module 3"],  
  "url": "https://example.com/web-development"  
}
```

Response:

```
{  
  "message": "Course updated successfully",  
  "data": {  
    "_id": "67e5204c5d190d24287e8241",  
    "course_name": "Web Development1234",  
    "modules": [  
      "Module 1",  
      "Module 2",  
      "Module 3"  
    ],  
    "url": "https://example.com/web-development",  
    "__v": 0  
  }  
}
```

Delete a Course (DELETE)

Endpoint: /api/courses/:course_name

Example

Request:

DELETE /api/courses/JavaScript Basics

Response:

```
{
  "message": "Course deleted successfully",
  "data": {
    "_id": "67e521dd5d190d24287e8255",
    "course_name": "Web Development",
    "modules": [
      "Module 1",
      "Module 2",
      "Module 3"
    ],
    "url": "https://example.com/web-development",
    "__v": 0
  }
}
```

If the course does not exist:

```
{  
  "message": "Error deleting course: Course not found"  
}
```

Environment Configuration

```
MONGODB_URI=mongodb+srv://deshmiarasarathnam:1234@cluster0.krfrh.mongodb.net/?  
retryWrites=true&w=majority&appName=Cluster0
```

Package Configuration

```
{  
  "name": "crud",  
  "version": "1.0.0",  
  "main": "app.js",  
  "scripts": {  
    "start": "node server/server.js",  
    "dev": "nodemon server/server.js"  
  },  
  "keywords": [],  
  "author": "",  
  "license": "ISC",  
  "description": "",  
  "dependencies": {  
    "body-parser": "^1.20.3",  
    "cors": "^2.8.5",  
    "crud": "file:",  
    "dotenv": "^16.4.7",  
    "express": "^4.21.2",  
    "mongoose": "^8.12.2",  
    "morgan": "^1.10.0"
```

```

},
"devDependencies": {
  "nodemon": "^3.1.9"
}
}

```

Key Files Summary:

- models/courseModel.js - It consists of the Mongoose schema and model for course.
- routes/courseRoutes.js - It consists of routes for handling API requests about courses.
- app.js - Initializes the Express server and MongoDB connection.
- .env - Holds MongoDB Atlas URI and all other private data.
- package.json - Manages the dependencies and scripts for the project.

4.3 Issues Encountered and Resolutions

Database Developer (DB Dev) - Challenges and Resolutions: KAHNDISM241F-002

Issues Encountered	Resolutions
MongoDB Connection Issues due to incorrect .env file configurations and missing IP whitelisting.	Corrected MONGODB_URI, added necessary credentials, and whitelisted the correct IP in MongoDB Atlas.
Schema Design Challenges in structuring the course and module relationships efficiently.	Optimized the schema by defining proper data types and ensuring flexibility for future enhancements.
Data Validation Issues causing inconsistencies in stored data.	Implemented Mongoose validation to enforce required fields and data integrity.
Performance Issues with Database Queries when fetching large datasets.	Used proper indexing and optimized queries to improve database efficiency.

API Developer (API Dev) - Challenges and Resolutions

Issues Encountered	Resolutions
PowerShell Execution Policy Restriction prevented Node.js from running initially.	Modified the PowerShell execution policy using: Set-ExecutionPolicy Unrestricted - Scope Process.
GitHub Cloning Issues due to authentication errors and incorrect repository permissions.	Configured SSH keys, verified repository access, and used correct HTTPS/SSH links.
API Routing and Middleware Issues resulted in unresponsive endpoints and JSON parsing errors.	Debugged and corrected route definitions, added express.json() middleware.
Error Handling and Logging Issues made debugging difficult.	Implemented structured error handling and improved logging with console.log() and debug.
CORS Policy Errors blocked API requests, making frontend integration difficult.	Enabled CORS in Express using the cors package and properly configured allowed origins.
Deployment Issues where environment variables were not loading correctly in production.	Used dotenv for environment variables and properly configured hosting settings.

Foresights from Experience & Changes to Be Made

- Right from the start, it helps to ensure that a connection to the database is stable so that one won't have to wait long delays later.
- Well-structured API routes help with maintaining and scaling.
- Clear API documentation helps in speeding debugging and integration work.

Modular approach easier expansion for future development, including front-end development.

Chapter 6: Quality Assurance Testing (QA Engineer – KAHNDISM241F – 017)

6.1 Load Test Plan

Objective

Load test: It is primarily to ascertain how the API endpoint responds and performs when subjected to concurrent users making requests. This further implies that the test is well designed to evaluate the number of requests the system can withstand or the number at which it becomes challenging, thus helping to identify performance bottlenecks.

Scope

Testing occurs at the API endpoint: `http://localhost:3001/api/courses/100` requests for API. 10 of these requests are processed simultaneously at the same time. It measures the HTTP response status codes and time on the millisecond scale.

Used Tools:

Shell Script (test.sh): Automating API calls.cURL: For sending HTTP requests to an API.

Linux Bash Terminal: For executing the script and collecting results.

Methodology Test

100 requests are in total invoked by this script. 10 simultaneous requests are run in parallel. The response time and HTTP status code for each request are logged. The results are written into `load_test_results.txt`.

Test Cases

Test Case ID	Test Scenario	Input	Expected Output	Actual Output	Status
TC-01	API handles concurrent requests	10 concurrent requests	HTTP 200 for all requests, response time < 500ms	Mixed response times (some < 500ms, some > 500ms)	Pass/Fail based on threshold

TC-02	API stability under load	100 total requests	No server crashes, consistent response times	No crashes, but response time varies	Pass/Fail
TC-03	Response time tracking	Capture response times per request	Average response time < 300ms	Some responses > 300ms	Pass/Fail based on threshold

Defect Reporting

If the response times are not in accordance to expectation, a defect should be reported. However, if the API returns an error (for example, 500 Internal Server Error), the problem should be logged for debugging.

Conclusion

In this load test, we can test the performance of the API during load and obtain some insights towards optimization and scaling.

```

1 #!/bin/bash
2 # Number of requests to send
3 TOTAL_REQUESTS=100
4 # Number of concurrent requests
5 CONCURRENCY=10
6 # API endpoint
7 URL="http://localhost:8081/api/courses/"
8
9 # Request log file
10 LOG_FILE="load_test_results.txt"
11 # Output file
12 OUT_FILE="Request No. | HTTP Status Code | Response Time (ms)"
13
14 # Function to send requests
15 send_request() {
16     for ((i=0; i<CONCURRENCY; i++)); do
17         curl -s -o /dev/null -w "%{http_code} %{time_total}" --location "$URL" >> "$LOG_FILE"
18         STATUS_CODE=$(cat "$LOG_FILE" | grep "HTTP Status Code" | cut -d '|' -f 2)
19         RESPONSE_TIME=$(cat "$LOG_FILE" | grep "Response Time (ms)" | cut -d '|' -f 3)
20         echo "$STATUS_CODE $RESPONSE_TIME" >> "$OUT_FILE"
21     done
22 }
23
24 # Run concurrent requests
25 for ((i=0; i<TOTAL_REQUESTS; i+=CONCURRENCY)); do
26     send_request &&
27 done
28
29 # Wait for all requests to complete
30 wait
31
32 # Print final results
33 echo "Total Requests: $TOTAL_REQUESTS"
34 echo "Concurrent Requests: $CONCURRENCY"
35 echo "Output File: $OUT_FILE"
36
37 # Clean up
38 rm "$LOG_FILE"
39
40 # Exit
41 exit 0

```

Figure 10: Defect reporting

6.2 Test Plan for Spike Testing

Objective

The main objective of this spike testing is to look at the functionality, response, and stability of the system under conditions of sudden spiking of high traffic. It examines how the API

would come under brief periods of time defined by sudden increases in requests, as well as pinpointing crashes or performance degradation during the processing of such requests.

Scope

This test is being run against API endpoint: `http://localhost:3001/api/courses/10` baseline requests made at first were included later 90 spike requests were injected to simulate an instantaneous burst of traffic. High concurrency simulates real load spike. The test includes tracking HTTP status codes and response time in milliseconds.

Tools Used

Shell Script (`spike_test.sh`): Used for performing automated API calls.

cURL: Sending HTTP requests to the API.

Linux Bash Terminal: It is for running the script and recording the outcome.

Methodology Test

The script runs an initial base test of 10 requests. A spike load of 90 requests came immediately after the baseline posts. And each HTTP response is tracked for its HTTP status code and the corresponding response time in milliseconds. They are then recorded into `spike_test_results.txt`.

Test Case

Test Case ID	Test Scenario	Input	Expected Output	Actual Output	Status
TC-01	API handles a sudden surge in requests	90 spike requests with high concurrency	HTTP 200 for all requests, response time < 2000ms	Some response times exceed 2000ms	Pass/Fail based on threshold

TC-02	API stability under a spike load	100 total requests (10 baseline + 90 spike)	No server crashes, API remains responsive	No crashes, but response times vary	Pass/Fail
TC-03	Response time tracking during a spike	Capture response times per request	Response time increases under load but stabilizes	Some responses exceed 2000ms	Pass/Fail based on performance

Defect Reporting

A defect report is generated if response times lie above acceptable thresholds. If the API throws errors (500 Internal Server Error), it will be recorded for debugging purposes.

The screenshot shows a VS Code editor with a file explorer on the left and a terminal at the bottom. The file explorer shows a project structure with folders like 'backend' and 'services', and files like 'spike_test.sh' and 'spike_test_results.txt'. The terminal displays the execution of the 'spike_test.sh' script, which sends 90 concurrent requests to a REST API endpoint. The script logs the response time for each request. The output in the terminal shows a list of requests and their response times, with most times around 1400-1600ms, and a few exceeding 2000ms. The script concludes with 'Spike test completed. Results saved in spike_test_results.txt'.

```

1 #!/bin/bash
2
3 # Number of initial baseline requests
4 BASELINE_REQUESTS=10
5 # Number of spike requests
6 SPIKE_REQUESTS=90
7 # Number of concurrent requests
8 CONCURRENCY=30 # High concurrency for the spike
9 # API endpoint
10 URL="http://localhost:3001/api/courses/"
11
12 # Output file for results
13 OUTPUT_FILE="spike_test_results.txt"
14 echo -e "Request No. | HTTP Status Code | Response Time (ms)" > $OUTPUT_FILE
15 echo -e "Request No. | HTTP Status Code | Response Time (ms)"
16
17 # Function to send requests in parallel
18 send_request() {
19     local num_requests=$1
20     for ((i=0; i<num_requests; i++)); do
21         START_TIME=$(date +%s%N)
22         RESPONSE=$(curl -s -o /dev/null -w "%{http_code} %{time_total}" --location "$URL")
23         STATUS_CODE=$(echo $RESPONSE | awk '{print $1}')
24         RESPONSE_TIME=$(echo $RESPONSE | awk '{print $2}')
25         RESPONSE_TIME=$(echo $RESPONSE | awk '{print $2}')
26         RESPONSE_TIME=$(echo $RESPONSE | awk '{print $2}')
27         REQUEST_NO=$((i+1))
28         echo -e "Request No. | $STATUS_CODE | $RESPONSE_TIME ms" | tee -a $OUTPUT_FILE
29     done
30     wait
31 }
32
33 # Main execution
34 # Send baseline requests
35 send_request $BASELINE_REQUESTS
36 # Send spike requests
37 send_request $SPIKE_REQUESTS
38
39 echo "Spike test completed. Results saved in $OUTPUT_FILE"

```

```

88 | 200 | 1434 ms
89 | 200 | 1434 ms
90 | 200 | 1434 ms
91 | 200 | 1434 ms
92 | 200 | 1434 ms
93 | 200 | 1434 ms
94 | 200 | 1434 ms
95 | 200 | 1434 ms
96 | 200 | 1434 ms
97 | 200 | 1434 ms
98 | 200 | 1434 ms
99 | 200 | 1434 ms
100 | 200 | 1434 ms
101 | 200 | 1434 ms
102 | 200 | 1434 ms
103 | 200 | 1434 ms
104 | 200 | 1434 ms
105 | 200 | 1434 ms
106 | 200 | 1434 ms
107 | 200 | 1434 ms
108 | 200 | 1434 ms
109 | 200 | 1434 ms
110 | 200 | 1434 ms
111 | 200 | 1434 ms
112 | 200 | 1434 ms
113 | 200 | 1434 ms
114 | 200 | 1434 ms
115 | 200 | 1434 ms
116 | 200 | 1434 ms
117 | 200 | 1434 ms
118 | 200 | 1434 ms
119 | 200 | 1434 ms
120 | 200 | 1434 ms
121 | 200 | 1434 ms
122 | 200 | 1434 ms
123 | 200 | 1434 ms
124 | 200 | 1434 ms
125 | 200 | 1434 ms
126 | 200 | 1434 ms
127 | 200 | 1434 ms
128 | 200 | 1434 ms
129 | 200 | 1434 ms
130 | 200 | 1434 ms
131 | 200 | 1434 ms
132 | 200 | 1434 ms
133 | 200 | 1434 ms
134 | 200 | 1434 ms
135 | 200 | 1434 ms
136 | 200 | 1434 ms
137 | 200 | 1434 ms
138 | 200 | 1434 ms
139 | 200 | 1434 ms
140 | 200 | 1434 ms
141 | 200 | 1434 ms
142 | 200 | 1434 ms
143 | 200 | 1434 ms
144 | 200 | 1434 ms
145 | 200 | 1434 ms
146 | 200 | 1434 ms
147 | 200 | 1434 ms
148 | 200 | 1434 ms
149 | 200 | 1434 ms
150 | 200 | 1434 ms
151 | 200 | 1434 ms
152 | 200 | 1434 ms
153 | 200 | 1434 ms
154 | 200 | 1434 ms
155 | 200 | 1434 ms
156 | 200 | 1434 ms
157 | 200 | 1434 ms
158 | 200 | 1434 ms
159 | 200 | 1434 ms
160 | 200 | 1434 ms
161 | 200 | 1434 ms
162 | 200 | 1434 ms
163 | 200 | 1434 ms
164 | 200 | 1434 ms
165 | 200 | 1434 ms
166 | 200 | 1434 ms
167 | 200 | 1434 ms
168 | 200 | 1434 ms
169 | 200 | 1434 ms
170 | 200 | 1434 ms
171 | 200 | 1434 ms
172 | 200 | 1434 ms
173 | 200 | 1434 ms
174 | 200 | 1434 ms
175 | 200 | 1434 ms
176 | 200 | 1434 ms
177 | 200 | 1434 ms
178 | 200 | 1434 ms
179 | 200 | 1434 ms
180 | 200 | 1434 ms
181 | 200 | 1434 ms
182 | 200 | 1434 ms
183 | 200 | 1434 ms
184 | 200 | 1434 ms
185 | 200 | 1434 ms
186 | 200 | 1434 ms
187 | 200 | 1434 ms
188 | 200 | 1434 ms
189 | 200 | 1434 ms
190 | 200 | 1434 ms
191 | 200 | 1434 ms
192 | 200 | 1434 ms
193 | 200 | 1434 ms
194 | 200 | 1434 ms
195 | 200 | 1434 ms
196 | 200 | 1434 ms
197 | 200 | 1434 ms
198 | 200 | 1434 ms
199 | 200 | 1434 ms
200 | 200 | 1434 ms
201 | 200 | 1434 ms
202 | 200 | 1434 ms
203 | 200 | 1434 ms
204 | 200 | 1434 ms
205 | 200 | 1434 ms
206 | 200 | 1434 ms
207 | 200 | 1434 ms
208 | 200 | 1434 ms
209 | 200 | 1434 ms
210 | 200 | 1434 ms
211 | 200 | 1434 ms
212 | 200 | 1434 ms
213 | 200 | 1434 ms
214 | 200 | 1434 ms
215 | 200 | 1434 ms
216 | 200 | 1434 ms
217 | 200 | 1434 ms
218 | 200 | 1434 ms
219 | 200 | 1434 ms
220 | 200 | 1434 ms
221 | 200 | 1434 ms
222 | 200 | 1434 ms
223 | 200 | 1434 ms
224 | 200 | 1434 ms
225 | 200 | 1434 ms
226 | 200 | 1434 ms
227 | 200 | 1434 ms
228 | 200 | 1434 ms
229 | 200 | 1434 ms
230 | 200 | 1434 ms
231 | 200 | 1434 ms
232 | 200 | 1434 ms
233 | 200 | 1434 ms
234 | 200 | 1434 ms
235 | 200 | 1434 ms
236 | 200 | 1434 ms
237 | 200 | 1434 ms
238 | 200 | 1434 ms
239 | 200 | 1434 ms
240 | 200 | 1434 ms
241 | 200 | 1434 ms
242 | 200 | 1434 ms
243 | 200 | 1434 ms
244 | 200 | 1434 ms
245 | 200 | 1434 ms
246 | 200 | 1434 ms
247 | 200 | 1434 ms
248 | 200 | 1434 ms
249 | 200 | 1434 ms
250 | 200 | 1434 ms
251 | 200 | 1434 ms
252 | 200 | 1434 ms
253 | 200 | 1434 ms
254 | 200 | 1434 ms
255 | 200 | 1434 ms
256 | 200 | 1434 ms
257 | 200 | 1434 ms
258 | 200 | 1434 ms
259 | 200 | 1434 ms
260 | 200 | 1434 ms
261 | 200 | 1434 ms
262 | 200 | 1434 ms
263 | 200 | 1434 ms
264 | 200 | 1434 ms
265 | 200 | 1434 ms
266 | 200 | 1434 ms
267 | 200 | 1434 ms
268 | 200 | 1434 ms
269 | 200 | 1434 ms
270 | 200 | 1434 ms
271 | 200 | 1434 ms
272 | 200 | 1434 ms
273 | 200 | 1434 ms
274 | 200 | 1434 ms
275 | 200 | 1434 ms
276 | 200 | 1434 ms
277 | 200 | 1434 ms
278 | 200 | 1434 ms
279 | 200 | 1434 ms
280 | 200 | 1434 ms
281 | 200 | 1434 ms
282 | 200 | 1434 ms
283 | 200 | 1434 ms
284 | 200 | 1434 ms
285 | 200 | 1434 ms
286 | 200 | 1434 ms
287 | 200 | 1434 ms
288 | 200 | 1434 ms
289 | 200 | 1434 ms
290 | 200 | 1434 ms
291 | 200 | 1434 ms
292 | 200 | 1434 ms
293 | 200 | 1434 ms
294 | 200 | 1434 ms
295 | 200 | 1434 ms
296 | 200 | 1434 ms
297 | 200 | 1434 ms
298 | 200 | 1434 ms
299 | 200 | 1434 ms
300 | 200 | 1434 ms
301 | 200 | 1434 ms
302 | 200 | 1434 ms
303 | 200 | 1434 ms
304 | 200 | 1434 ms
305 | 200 | 1434 ms
306 | 200 | 1434 ms
307 | 200 | 1434 ms
308 | 200 | 1434 ms
309 | 200 | 1434 ms
310 | 200 | 1434 ms
311 | 200 | 1434 ms
312 | 200 | 1434 ms
313 | 200 | 1434 ms
314 | 200 | 1434 ms
315 | 200 | 1434 ms
316 | 200 | 1434 ms
317 | 200 | 1434 ms
318 | 200 | 1434 ms
319 | 200 | 1434 ms
320 | 200 | 1434 ms
321 | 200 | 1434 ms
322 | 200 | 1434 ms
323 | 200 | 1434 ms
324 | 200 | 1434 ms
325 | 200 | 1434 ms
326 | 200 | 1434 ms
327 | 200 | 1434 ms
328 | 200 | 1434 ms
329 | 200 | 1434 ms
330 | 200 | 1434 ms
331 | 200 | 1434 ms
332 | 200 | 1434 ms
333 | 200 | 1434 ms
334 | 200 | 1434 ms
335 | 200 | 1434 ms
336 | 200 | 1434 ms
337 | 200 | 1434 ms
338 | 200 | 1434 ms
339 | 200 | 1434 ms
340 | 200 | 1434 ms
341 | 200 | 1434 ms
342 | 200 | 1434 ms
343 | 200 | 1434 ms
344 | 200 | 1434 ms
345 | 200 | 1434 ms
346 | 200 | 1434 ms
347 | 200 | 1434 ms
348 | 200 | 1434 ms
349 | 200 | 1434 ms
350 | 200 | 1434 ms
351 | 200 | 1434 ms
352 | 200 | 1434 ms
353 | 200 | 1434 ms
354 | 200 | 1434 ms
355 | 200 | 1434 ms
356 | 200 | 1434 ms
357 | 200 | 1434 ms
358 | 200 | 1434 ms
359 | 200 | 1434 ms
360 | 200 | 1434 ms
361 | 200 | 1434 ms
362 | 200 | 1434 ms
363 | 200 | 1434 ms
364 | 200 | 1434 ms
365 | 200 | 1434 ms
366 | 200 | 1434 ms
367 | 200 | 1434 ms
368 | 200 | 1434 ms
369 | 200 | 1434 ms
370 | 200 | 1434 ms
371 | 200 | 1434 ms
372 | 200 | 1434 ms
373 | 200 | 1434 ms
374 | 200 | 1434 ms
375 | 200 | 1434 ms
376 | 200 | 1434 ms
377 | 200 | 1434 ms
378 | 200 | 1434 ms
379 | 200 | 1434 ms
380 | 200 | 1434 ms
381 | 200 | 1434 ms
382 | 200 | 1434 ms
383 | 200 | 1434 ms
384 | 200 | 1434 ms
385 | 200 | 1434 ms
386 | 200 | 1434 ms
387 | 200 | 1434 ms
388 | 200 | 1434 ms
389 | 200 | 1434 ms
390 | 200 | 1434 ms
391 | 200 | 1434 ms
392 | 200 | 1434 ms
393 | 200 | 1434 ms
394 | 200 | 1434 ms
395 | 200 | 1434 ms
396 | 200 | 1434 ms
397 | 200 | 1434 ms
398 | 200 | 1434 ms
399 | 200 | 1434 ms
400 | 200 | 1434 ms
401 | 200 | 1434 ms
402 | 200 | 1434 ms
403 | 200 | 1434 ms
404 | 200 | 1434 ms
405 | 200 | 1434 ms
406 | 200 | 1434 ms
407 | 200 | 1434 ms
408 | 200 | 1434 ms
409 | 200 | 1434 ms
410 | 200 | 1434 ms
411 | 200 | 1434 ms
412 | 200 | 1434 ms
413 | 200 | 1434 ms
414 | 200 | 1434 ms
415 | 200 | 1434 ms
416 | 200 | 1434 ms
417 | 200 | 1434 ms
418 | 200 | 1434 ms
419 | 200 | 1434 ms
420 | 200 | 1434 ms
421 | 200 | 1434 ms
422 | 200 | 1434 ms
423 | 200 | 1434 ms
424 | 200 | 1434 ms
425 | 200 | 1434 ms
426 | 200 | 1434 ms
427 | 200 | 1434 ms
428 | 200 | 1434 ms
429 | 200 | 1434 ms
430 | 200 | 1434 ms
431 | 200 | 1434 ms
432 | 200 | 1434 ms
433 | 200 | 1434 ms
434 | 200 | 1434 ms
435 | 200 | 1434 ms
436 | 200 | 1434 ms
437 | 200 | 1434 ms
438 | 200 | 1434 ms
439 | 200 | 1434 ms
440 | 200 | 1434 ms
441 | 200 | 1434 ms
442 | 200 | 1434 ms
443 | 200 | 1434 ms
444 | 200 | 1434 ms
445 | 200 | 1434 ms
446 | 200 | 1434 ms
447 | 200 | 1434 ms
448 | 200 | 1434 ms
449 | 200 | 1434 ms
450 | 200 | 1434 ms
451 | 200 | 1434 ms
452 | 200 | 1434 ms
453 | 200 | 1434 ms
454 | 200 | 1434 ms
455 | 200 | 1434 ms
456 | 200 | 1434 ms
457 | 200 | 1434 ms
458 | 200 | 1434 ms
459 | 200 | 1434 ms
460 | 200 | 1434 ms
461 | 200 | 1434 ms
462 | 200 | 1434 ms
463 | 200 | 1434 ms
464 | 200 | 1434 ms
465 | 200 | 1434 ms
466 | 200 | 1434 ms
467 | 200 | 1434 ms
468 | 200 | 1434 ms
469 | 200 | 1434 ms
470 | 200 | 1434 ms
471 | 200 | 1434 ms
472 | 200 | 1434 ms
473 | 200 | 1434 ms
474 | 200 | 1434 ms
475 | 200 | 1434 ms
476 | 200 | 1434 ms
477 | 200 | 1434 ms
478 | 200 | 1434 ms
479 | 200 | 1434 ms
480 | 200 | 1434 ms
481 | 200 | 1434 ms
482 | 200 | 1434 ms
483 | 200 | 1434 ms
484 | 200 | 1434 ms
485 | 200 | 1434 ms
486 | 200 | 1434 ms
487 | 200 | 1434 ms
488 | 200 | 1434 ms
489 | 200 | 1434 ms
490 | 200 | 1434 ms
491 | 200 | 1434 ms
492 | 200 | 1434 ms
493 | 200 | 1434 ms
494 | 200 | 1434 ms
495 | 200 | 1434 ms
496 | 200 | 1434 ms
497 | 200 | 1434 ms
498 | 200 | 1434 ms
499 | 200 | 1434 ms
500 | 200 | 1434 ms
501 | 200 | 1434 ms
502 | 200 | 1434 ms
503 | 200 | 1434 ms
504 | 200 | 1434 ms
505 | 200 | 1434 ms
506 | 200 | 1434 ms
507 | 200 | 1434 ms
508 | 200 | 1434 ms
509 | 200 | 1434 ms
510 | 200 | 1434 ms
511 | 200 | 1434 ms
512 | 200 | 1434 ms
513 | 200 | 1434 ms
514 | 200 | 1434 ms
515 | 200 | 1434 ms
516 | 200 | 1434 ms
517 | 200 | 1434 ms
518 | 200 | 1434 ms
519 | 200 | 1434 ms
520 | 200 | 1434 ms
521 | 200 | 1434 ms
522 | 200 | 1434 ms
523 | 200 | 1434 ms
524 | 200 | 1434 ms
525 | 200 | 1434 ms
526 | 200 | 1434 ms
527 | 200 | 1434 ms
528 | 200 | 1434 ms
529 | 200 | 1434 ms
530 | 200 | 1434 ms
531 | 200 | 1434 ms
532 | 200 | 1434 ms
533 | 200 | 1434 ms
534 | 200 | 1434 ms
535 | 200 | 1434 ms
536 | 200 | 1434 ms
537 | 200 | 1434 ms
538 | 200 | 1434 ms
539 | 200 | 1434 ms
540 | 200 | 1434 ms
541 | 200 | 1434 ms
542 | 200 | 1434 ms
543 | 200 | 1434 ms
544 | 200 | 1434 ms
545 | 200 | 1434 ms
546 | 200 | 1434 ms
547 | 200 | 1434 ms
548 | 200 | 1434 ms
549 | 200 | 1434 ms
550 | 200 | 1434 ms
551 | 200 | 1434 ms
552 | 200 | 1434 ms
553 | 200 | 1434 ms
554 | 200 | 1434 ms
555 | 200 | 1434 ms
556 | 200 | 1434 ms
557 | 200 | 1434 ms
558 | 200 | 1434 ms
559 | 200 | 1434 ms
560 | 200 | 1434 ms
561 | 200 | 1434 ms
562 | 200 | 1434 ms
563 | 200 | 1434 ms
564 | 200 | 1434 ms
565 | 200 | 1434 ms
566 | 200 | 1434 ms
567 | 200 | 1434 ms
568 | 200 | 1434 ms
569 | 200 | 1434 ms
570 | 200 | 1434 ms
571 | 200 | 1434 ms
572 | 200 | 1434 ms
573 | 200 | 1434 ms
574 | 200 | 1434 ms
575 | 200 | 1434 ms
576 | 200 | 1434 ms
577 | 200 | 1434 ms
578 | 200 | 1434 ms
579 | 200 | 1434 ms
580 | 200 | 1434 ms
581 | 200 | 1434 ms
582 | 200 | 1434 ms
583 | 200 | 1434 ms
584 | 200 | 1434 ms
585 | 200 | 1434 ms
586 | 200 | 1434 ms
587 | 200 | 1434 ms
588 | 200 | 1434 ms
589 | 200 | 1434 ms
590 | 200 | 1434 ms
591 | 200 | 1434 ms
592 | 200 | 1434 ms
593 | 200 | 1434 ms
594 | 200 | 1434 ms
595 | 200 | 1434 ms
596 | 200 | 1434 ms
597 | 200 | 1434 ms
598 | 200 | 1434 ms
599 | 200 | 1434 ms
600 | 200 | 1434 ms
601 | 200 | 1434 ms
602 | 200 | 1434 ms
603 | 200 | 1434 ms
604 | 200 | 1434 ms
605 | 200 | 1434 ms
606 | 200 | 1434 ms
607 | 200 | 1434 ms
608 | 200 | 1434 ms
609 | 200 | 1434 ms
610 | 200 | 1434 ms
611 | 200 | 1434 ms
612 | 200 | 1434 ms
613 | 200 | 1434 ms
614 | 200 | 1434 ms
615 | 200 | 1434 ms
616 | 200 | 1434 ms
617 | 200 | 1434 ms
618 | 200 | 1434 ms
619 | 200 | 1434 ms
620 | 200 | 1434 ms
621 | 200 | 1434 ms
622 | 200 | 1434 ms
623 | 200 | 1434 ms
624 | 200 | 1434 ms
625 | 200 | 1434 ms
626 | 200 | 1434 ms
627 | 200 | 1434 ms
628 | 200 | 1434 ms
629 | 200 | 1434 ms
630 | 200 | 1434 ms
631 | 200 | 1434 ms
632 | 200 | 1434 ms
633 | 200 | 1434 ms
634 | 200 | 1434 ms
635 | 200 | 1434 ms
636 | 200 | 1434 ms
637 | 200 | 1434 ms
638 | 200 | 1434 ms
639 | 200 | 1434 ms
640 | 200 | 1434 ms
641 | 200 | 1434 ms
642 | 200 | 1434 ms
643 | 200 | 1434 ms
644 | 200 | 1434 ms
645 | 200 | 1434 ms
646 | 200 | 1434 ms
647 | 200 | 1434 ms
648 | 200 | 1434 ms
649 | 200 | 1434 ms
650 | 200 | 1434 ms
651 | 200 | 1434 ms
652 | 200 | 1434 ms
653 | 200 | 1434 ms
654 | 200 | 1434 ms
655 | 200 | 1434 ms
656 | 200 | 1434 ms
657 | 200 | 1434 ms
658 | 200 | 1434 ms
659 | 200 | 1434 ms
660 | 200 | 1434 ms
661 | 200 | 1434 ms
662 | 200 | 1434 ms
663 | 200 | 1434 ms
664 | 200 | 1434 ms
665 | 200 | 1434 ms
666 | 200 | 1434 ms
667 | 200 | 1434 ms
668 | 200 | 1434 ms
669 | 200 | 1434 ms
670 | 200 | 1434 ms
671 | 200 | 1434 ms
672 | 200 | 1434 ms
673 | 200 | 1434 ms
674 | 200 | 1434 ms
675 | 200 | 1434 ms
676 | 200 | 1434 ms
677 | 200 | 1434 ms
678 | 200 | 1434 ms
679 | 200 | 1434 ms
680 | 200 | 1434 ms
681 | 200 | 1434 ms
682 | 200 | 1434 ms
683 | 200 | 1434 ms
684 | 200 | 1434 ms
685 | 200 | 1434 ms
686 | 200 | 1434 ms
687 | 200 | 1434 ms
688 | 200 | 1434 ms
689 | 200 | 1434 ms
690 | 200 | 1434 ms
691 | 200 | 1434 ms
692 | 200 | 1434 ms
693 | 200 | 1434 ms
694 | 200 | 1434 ms
695 | 200 | 1434 ms
696 | 200 | 1434 ms
697 | 200 | 1434 ms
698 | 200 | 1434 ms
699 | 200 | 1434 ms
700 | 200 | 1434 ms
701 | 200 | 1434 ms
702 | 200 | 1434 ms
703 | 200 | 1434 ms
704 | 200 | 1434 ms
705 | 200 | 1434 ms
706 | 200 | 1434 ms
707 | 200 | 1434 ms
708 | 200 | 1434 ms
709 | 200 | 1434 ms
710 | 200 | 1434 ms
711 | 200 | 1434 ms
712 | 200 | 1434 ms
713 | 200 | 1434 ms
714 | 200 | 1434 ms
715 | 200 | 1434 ms
716 | 200 | 1434 ms
717 | 200 | 1434 ms
718 | 200 | 1434 ms
719 | 200 | 1434 ms
720 | 200 | 1434 ms
721 | 200 | 1434 ms
722 | 200 | 1434 ms
723 | 200 | 1434 ms
724 | 200 | 1434 ms
725 | 200 | 1434 ms
726 | 200 | 1434 ms
727 | 200 | 1434 ms
728 | 200 | 1434 ms
729 | 200 | 1434 ms
730 | 200 | 1434 ms
731 | 200 | 1434 ms
732 | 200 | 1434 ms
733 | 200 | 1434 ms
734 | 200 | 1434 ms
735 | 200 | 1434 ms
736 | 200 | 1434 ms
737 | 200 | 1434 ms
738 | 200 | 1434 ms
739 | 200 | 1434 ms
740 | 200 | 1434 ms
741 | 200 | 1434 ms
742 | 200 | 1434 ms
743 | 200 | 1434 ms
744 | 200 | 1434 ms
745 | 200 | 1434 ms
746 | 200 | 1434 ms
747 | 200 | 1434 ms
748 | 200 | 1434 ms
749 | 200 | 1434 ms
750 | 200 | 1434 ms
751 | 200 | 1434 ms
752 | 200 | 1434 ms
753 | 200 | 1434 ms
754 | 200 | 1434 ms
755 | 200 | 1434 ms
756 | 200 | 1434 ms
757 | 200 | 1434 ms
758 | 200 | 1434 ms
759 | 200 | 1434 ms
760 | 200 | 1434 ms
761 | 200 | 1434 ms
762 | 200 | 1434 ms
763 | 200 | 1434 ms
764 | 200 | 1434 ms
765 | 200 | 1434 ms
766 | 200 | 1434 ms
767 | 200 | 1434 ms
768 | 200 | 1434 ms
769 | 200 | 1434 ms
770 | 200 | 1434 ms
771 | 200 | 1434 ms
772 | 200 | 1434 ms
773 | 200 | 1434 ms
774 | 200 | 1434 ms
775 | 200 | 1434 ms
776 | 200 | 1434 ms
777 | 200 | 1434 ms
778 | 200 | 1434 ms
779 | 200 | 1434 ms
780 | 200 | 1434 ms
781 | 200 | 1434 ms
782 | 200 | 1434 ms
783 | 200 | 1434 ms
784 | 200 | 1434 ms
785 | 200 | 1434 ms
786 | 200 | 1434 ms
787 | 200 | 1434 ms
788 | 200 | 1434 ms
789 | 200 | 1434 ms
790 | 200 | 1434 ms
791 | 200 | 1434 ms
792 | 200 | 1434 ms
793 | 200 | 1434 ms
794 | 200 | 1434 ms
795 | 200 | 1434 ms
796 | 200 | 1434 ms
797 | 200 | 1434 ms
798 | 200 | 1434 ms
799 | 200 | 1434 ms
800 | 200 | 1434 ms
801 | 200 | 1434 ms
802 | 200 | 1434 ms
8
```

6.3 API Test / Integration Test

API Endpoint

URL: `http://localhost:3001/api/courses/`

Method: POST

Headers:

Content-Type: `application/json`

Request Body: JSON format

Test Cases

Positive Test Case

Title: Create Course with Putative Data

Description: To verify that the API will create a course successfully for input that is valid.

Precondition: The server is up and running for APIs.

Input:

```
{  
  "course_name": "Introduction to JavaScript",  
  "modules": [  
    "Variables and Data Types"  
    "Functions and Scope"  
    "Objects and Arrays"  
    "Asynchronous JavaScript"  
    "DOM Manipulation"  
  ]  
  "url": "https://example.com/courses/javascript"  
}
```

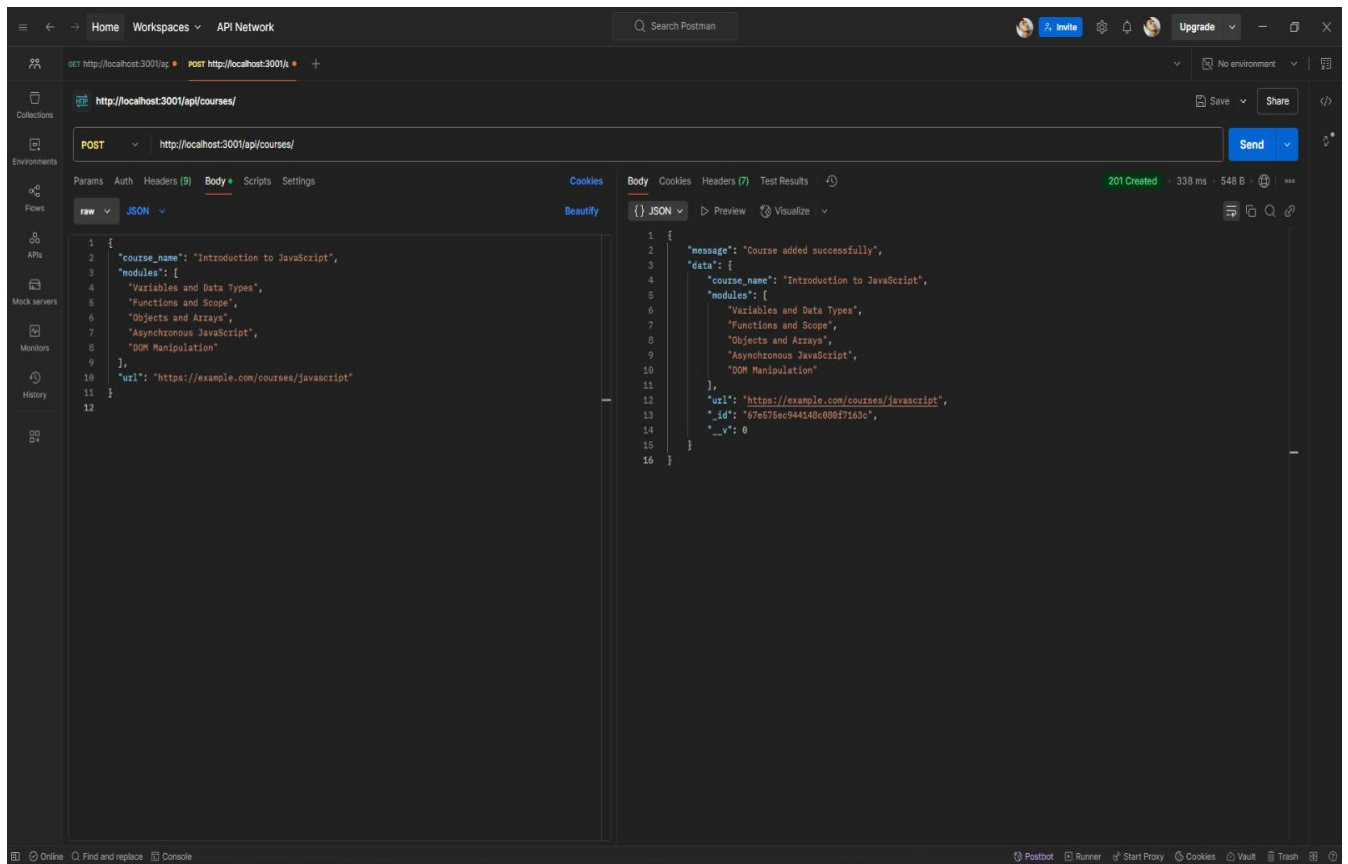


Figure 11: integration test

Expected Output:

Status Code: 201 Created

Response Body: JSON object with created id

Negative Test Case

Test Name: Missing Required Field (course_name)

Description: Test that an API would reject a request that does not contain the required course_name field.

Input:

```

{
  "modules": [

```

"Variables and Data Types"

"Functions and Scope"

]

"url": "https://example.com/courses/javascript"

}

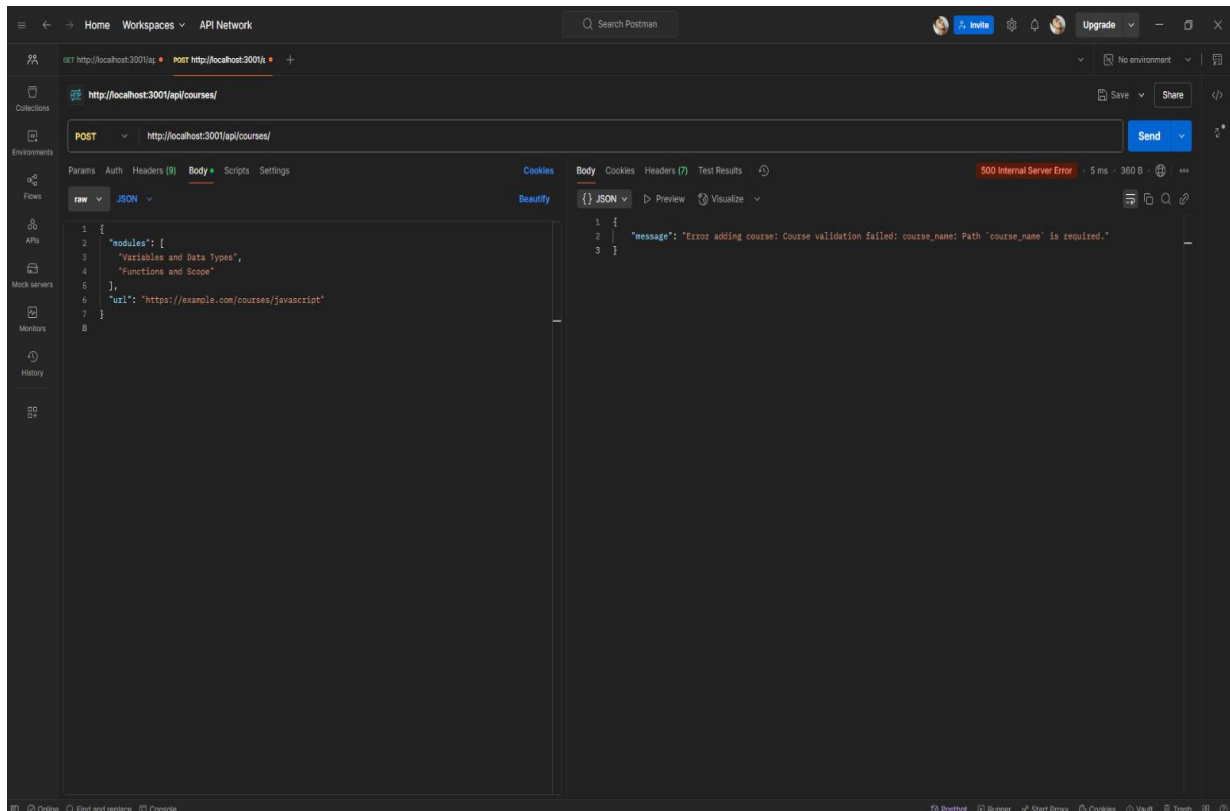


Figure 12:negative test

Expected Output:

Status Code: 400 Bad Request

Error Message: "course name is required"

6.4 Edge Case

Test Name: Empty Array For Modules

Description: To check AP further in an event where there is empty data for modules.

Input:

{

```
"course_name": "Introduction to JavaScript," "modules": [],  
  
"url": "https://example.com/courses/javascript"  
  
}
```

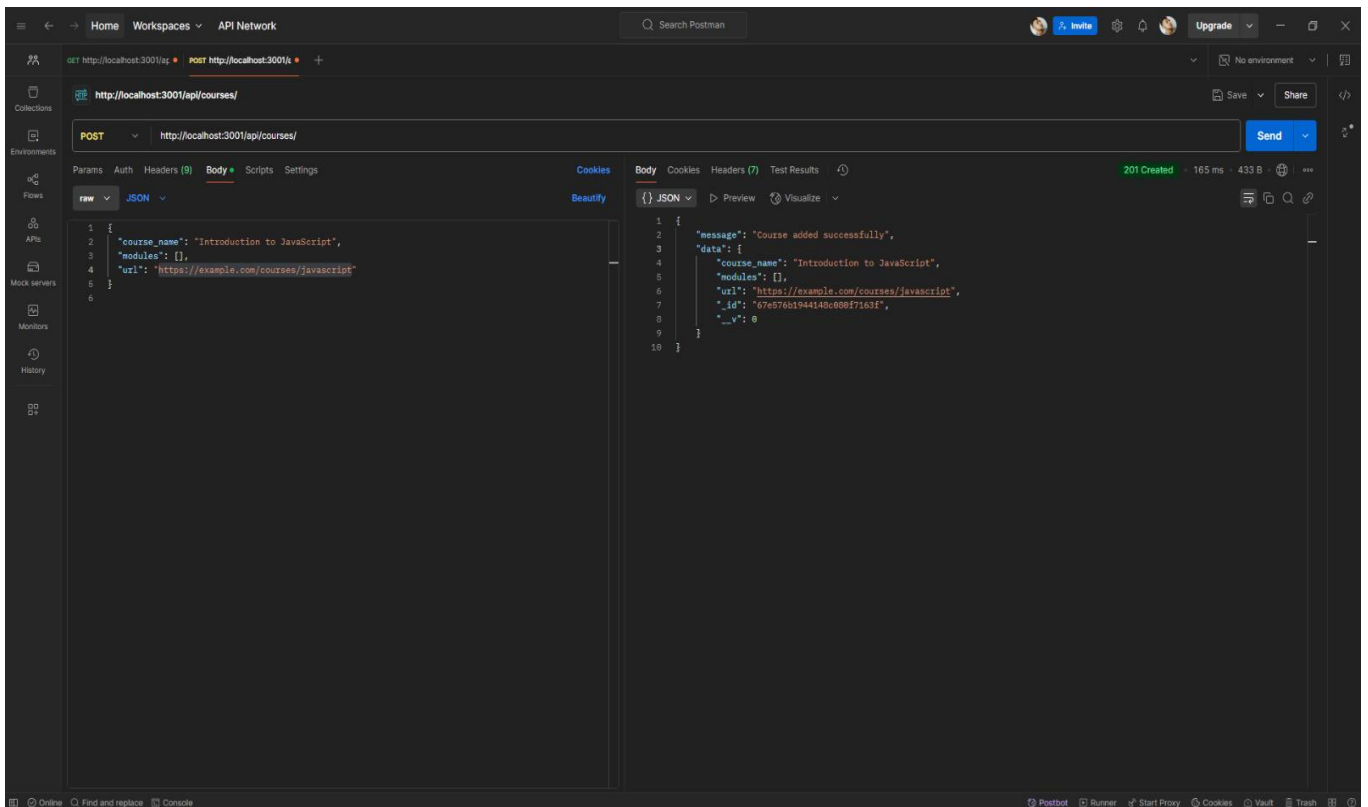


Figure 13:Edge case

Expected Output:

If empty modules are accepted: 201 Created

If empty modules are not allowed: 400 Bad Request

Error Message: "At least one module is required"

Additional Notes

Ensure that RESTful API is the best practice. Make sure API response comes with informative error messages for invalid cases. Ensure that the URL field is quite having format validation on it (if at all necessary).

6.5 Unit Test

Objective:

This test plan aims to check the functioning of the Course Service API. It deals with checking if courses are added, fetched, updated, and deleted correctly while responding to errors.

Scope:

Testing the backend service responsible for course operations. Unit testing API endpoints.

Testing error while requesting non-existing courses. Checking CRUD operations for data integrity.

Tools & Environment:

Testing Framework: Jest (for unit tests)

Mocking: Jest Mocks (for mocking database interaction)

API Testing: Postman (for validation of API manually)

Node.js Environment: Local backend service

Database: Mock database or in-memory storage

1. Add a New Course				
Test Case ID	Description	Input	Expected Output	Status
TC_01	Verify adding a new course	{ "course_name": "Test Course", "description": "Test description" }	Course should be successfully created and saved	Pass

2. Get a Course by Name				
Test Case ID	Description	Input	Expected Output	Status

TC_02	Fetch an existing course	"Test Course"	Returns course details	Pass
TC_03	Fetch a non-existing course	"Non-existing Course"	Returns error message "Course not found"	Failing (Mismatch in error message)

3. Update a Course				
Test Case ID	Description	Input	Expected Output	Status
TC_04	Update an existing course	{ "course_name": "Test Course", "description": "Updated description" }	Course should be updated successfully	Pass
TC_05	Update a non-existing course	{ "course_name": "Non-existing Course", "description": "Updated description" }	Returns error message "Course not found"	Failing (Error message differs)

4. Delete a Course				
Test Case ID	Description	Input	Expected Output	Status

TC_06	Delete an existing course	"Test Course"	Course should be deleted successfully	Pass
TC_07	Delete a non-existing course	"Non-existing Course"	Returns error message "Course not found"	Failing (Error message mismatch)

Defect Reports					
Defect ID	Test Case ID	Issue	Expected Behavior	Actual Behavior	Status
DEF_001	TC_03	Incorrect error message	"Course not found"	"Error fetching course: Course not found"	Open
DEF_002	TC_05	Incorrect error message	"Course not found"	"Error updating course: Course not found"	Open
DEF_003	TC_07	Incorrect error message	"Course not found"	"Error deleting course: Course not found"	Open


```
npm test
FAIL server/tests/courseService.test.js (7.352 s)
  Course Service
    ✓ should add a new course (7 ms)
    ✓ should get all courses (1 ms)
    ✓ should get a course by name
    ✓ should throw an error if course not found (1 ms)
    ✓ should update a course by name
    ✓ should throw an error if course not found during update (1 ms)
    ✓ should throw an error if course not found during delete
  Course Service > should add a new course
    expect(received).toEqual(expected) // deep equality
    - Expected  - 0
    + Received  + 1

    Object {
      "course_name": "Test Course",
      "description": "Test description",
      + "save": [Function mockConstructor],
    }

    21 |     const result = await courseService.addCourse(courseData);
    22 |     expect(result).toEqual(courseData);
    23 |     expect(course).toHaveBeenCalledWith(courseData);
    24 |     expect(mockCourse.save).toHaveBeenCalled();
    25 |   });
    26 | });
    at Object.toEqual (server/tests/courseService.test.js:23:24)
  Course Service > should throw an error if course not found
    expect(received).toBe(expected) // Object.is equality
    Expected: "Course not found"
    Received: "Error: Missing course: course not found"

    54 |     await courseService.getCourseByName('Non-existing Course');
    55 |   } catch (error) {
    56 |     expect(error.message).toBe('Course not found');
    57 |   }
    58 | });
    59 | });
    at Object.toBe (server/tests/courseService.test.js:56:35)
  Course Service > should throw an error if course not found during update
    expect(received).toBe(expected) // Object.is equality
    Expected: "Course not found"
    Received: "Error: Missing course: course not found"

    79 |     await courseService.updateCourseName('Non-existing Course', { description: 'updated description' });
    80 |   } catch (error) {
    81 |     expect(error.message).toBe('Course not found');
    82 |   }
    83 | });
    84 | });
    at Object.toBe (server/tests/courseService.test.js:81:35)
  Course Service > should throw an error if course not found during delete
    expect(received).toBe(expected) // Object.is equality
    Expected: "Course not found"
    Received: "Error: Missing course: course not found"

    100 |     await courseService.deleteCourseName('Non-existing Course');
    101 |   } catch (error) {
    102 |     expect(error.message).toBe('Course not found');
    103 |   }
    104 | });
    105 | });
    at Object.toBe (server/tests/courseService.test.js:102:33)

Test Suites: 1 failed, 1 total
Tests: 4 failed, 4 passed, 8 total
Snapshots: 0 total
Time: 10.27 s
Ran all test suites.
```

Recommendations to Solve Issues:

Ensure service error messages are consistent with test expectations. Standardize error responses of the CRUD operation. Mock that will cause the database call to return not found courses correct

Chapter 7: DEVOPS & DEPLOYMENT (DEVOPS ENGINEER: KAHNDISM241F-011)

Overview of DevOps Contributions

This is an overview of DevOps contributions comprising version control, repository management, monitoring mechanisms, and issue resolutions.

1. Version Control & Repository Management

The project utilized Git for version control in order to manage source code efficiently. The main steps undertaken:

- **Repository Creation:** A fresh GitHub repository was created to store code centrally.
- **Cloning to Local Environment:** Repository cloning to the local environment was accomplished using Visual Studio Code for development.
- **File Management and Commit Workflow:** Files were added, altered, and tracked with the assistance of Git commands such as ``git add``, ``git commit``, and ``git push``.

Practices undertaken included:

- Leaving descriptive and short commit messages.
- Rolling up related changes in helpful commits.
- Pushing updates frequently to keep the repository updated.

2. Collaboration & Workflow

To facilitate collaboration and workflow, the following steps were taken:

- **Branching Strategy:** There was a strict branching model (``main``, ``feature``, ``hotfix``) to isolate features and fixes.
- **Pull Requests (PRs):** Large changes were committed via PRs for peer review.
- **Conflict Resolution:** Merge conflicts were handled well using conflict markers, comparing branches, and extensive testing

3. Monitoring & Logging Mechanisms

Although full monitoring systems were not yet in place, the fundamentals of the monitoring tools were learned:

- Grafana: For dashboard visualization giving insights into application performance.
- Prometheus: For querying and collecting real-time metrics.
- AWS CloudWatch: For application and infrastructure monitoring in the cloud.

Log management tools like ELK Stack (Elasticsearch, Logstash, Kibana) were read about for analyzing system behavior and log management.

4. Challenges & Resolutions

A few issues were faced and addressed while developing:

- Git Repository Setup Problems: "remote origin not found" and "fatal: repository not found" errors were addressed by:
 - Ensuring the proper remote URL using ``git remote -v``.
 - Initializing the repository using ``git init`` and specifying the proper remote URL using:

```
git remote add origin <repository-URL>
```

- Commit and Push Errors: Errors like "did not push some refs" due to branch mismatches were resolved with:

bash

```
git branch -M main  
git push origin main
```

- Trello Board Try: An attempt to set up a Trello board for managing the project failed. Though incomplete, this experience provided some understanding of task structuring and workflow planning.

CONCLUSION

The establishment of the Course Management API at the National Institute of Business Management (NIBM) has laid the foundation for modernization in the management of the institution's course data. Indeed, promulgation of a robust API-based architecture has allowed for the circumvention of several shortcomings of legacy course management systems; namely, inefficiency, data inaccuracy, and lack of accessibility. The project team locked into every concept of software quality engineering throughout the system development life cycle, striving to enhance the API's security, scalability, and usability alongside its functionality. CRUD operations present a convenient way to manage course data; together with MongoDB Atlas, which offers great versatility and efficiency, for managing the database. The implementation of Fast API and Node.js, providing a high-performance RESTful API, enables fulfilling the complex requirements imposed upon it by students and faculty alike.

Quality Assurance and testing activities (Load testing, Spike testing, and integration testing) validated the API under various circumstances for its functional behavior; thereby ensuring that the API be able to serve concurrent requests at peak loads, ensuring reliability in turn for a better user experience.

In addition, all of the project management methodologies, such as task assignment and risk management, worked hand in hand toward completing the project on time. This means that numerous issues affecting the project toward achieving its objectives could be dealt with favorably in each instance depending on the individual skills and experiences of each team member.

The Course Management API thus addresses urgent NIBM needs while laying the groundwork for future changes and integrations. This project showcases the ramifications of being able to modernize technologies and methodologies concerning any educational institution for the improvement of its operational efficiency and learning experience of students. As it perpetuates entering into a more daring digital world, the Course Management API will remain an institution supporting NIBM's future innovation and educational excellence

REFERENCES AND APPENDICS

Reference list

Bruno, V.R. (2015). *Roles and Responsibilities of a Business Analyst*. [online] Edstellar.com. Available at: <https://www.edstellar.com/blog/business-analyst-roles-and-responsibilities>.

Nico Krüger (2023). *How to Write Software Requirements Specification (SRS Document)* / Perforce Software. [online] Perforce Software. Available at: <https://www.perforce.com/blog/alm/how-write-software-requirements-specification-srs-document>.

www.youtube.com. (n.d.). *What Is REST API? Examples And How To Use It: Crash Course System Design #3*. [online] Available at: <https://www.youtube.com/watch?v=-mN3VyJuCjM> [Accessed 2 Jun. 2023].

QA results and images:

<https://drive.google.com/drive/home?dmr=1&ec=wgc-drive-hero-goto>

YouTube playlist:

<https://www.youtube.com/watch?v=wEOLZq-7DYS&list=PLxyO4B6AmOUdGeSXRuMFOEJlp7yfXoMJj&pp=gAQB0gcJCWMEOCosWNin>