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

List Of Figures

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)	• 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)	• 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	• 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	• 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: SOFTWEARE 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: Securit3+y 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 DEVELPOERS: 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.

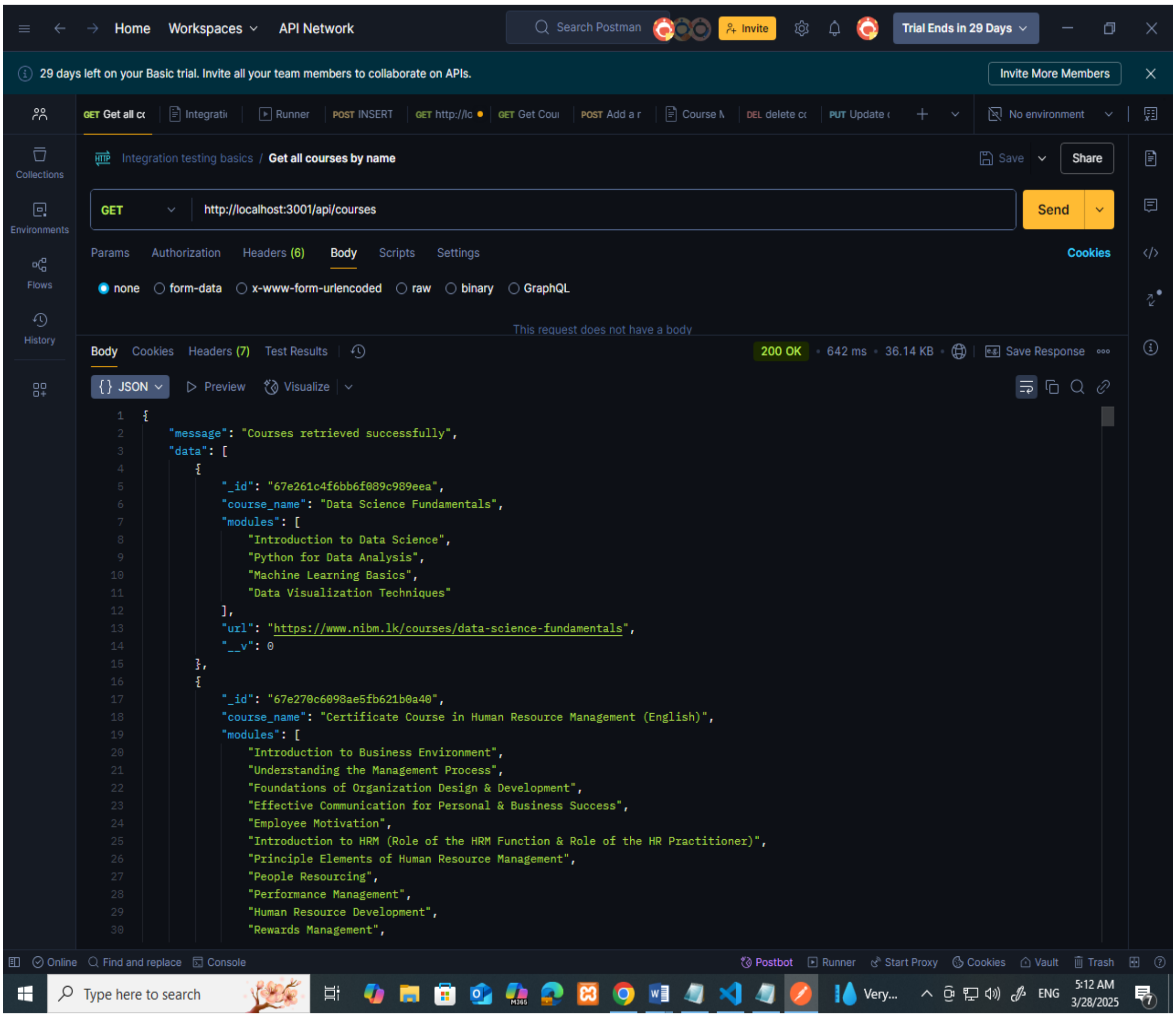
2. API Endpoints

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:



GET http://localhost:3001/api/courses

200 OK • 642 ms • 36.14 KB

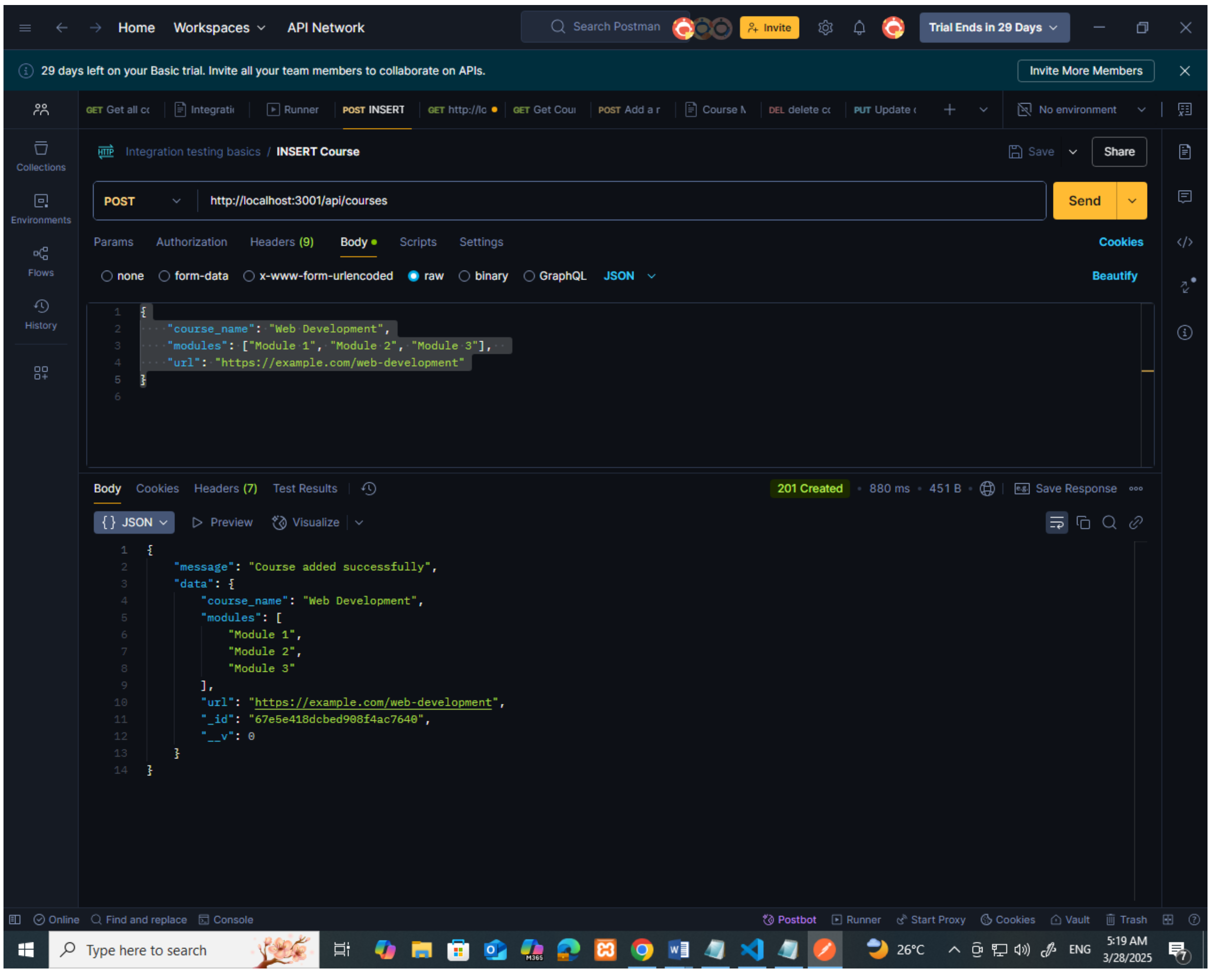
```
{
    "message": "Courses retrieved successfully",
    "data": [
        {
            "_id": "67e261c4f6bb6f089c989eea",
            "course_name": "Data Science Fundamentals",
            "modules": [
                "Introduction to Data Science",
                "Python for Data Analysis",
                "Machine Learning Basics",
                "Data Visualization Techniques"
            ],
            "url": "https://www.nibm.lk/courses/data-science-fundamentals",
            "__v": 0
        },
        {
            "_id": "67e270c6098ae5fb621b0a40",
            "course_name": "Certificate Course in Human Resource Management (English)",
            "modules": [
                "Introduction to Business Environment",
                "Understanding the Management Process",
                "Foundations of Organization Design & Development",
                "Effective Communication for Personal & Business Success",
                "Employee Motivation",
                "Introduction to HRM (Role of the HRM Function & Role of the HR Practitioner)",
                "Principle Elements of Human Resource Management",
                "People Resourcing",
                "Performance Management",
                "Human Resource Development",
                "Rewards Management",
```

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:



POST http://localhost:3001/api/courses

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

201 Created · 880 ms · 451 B

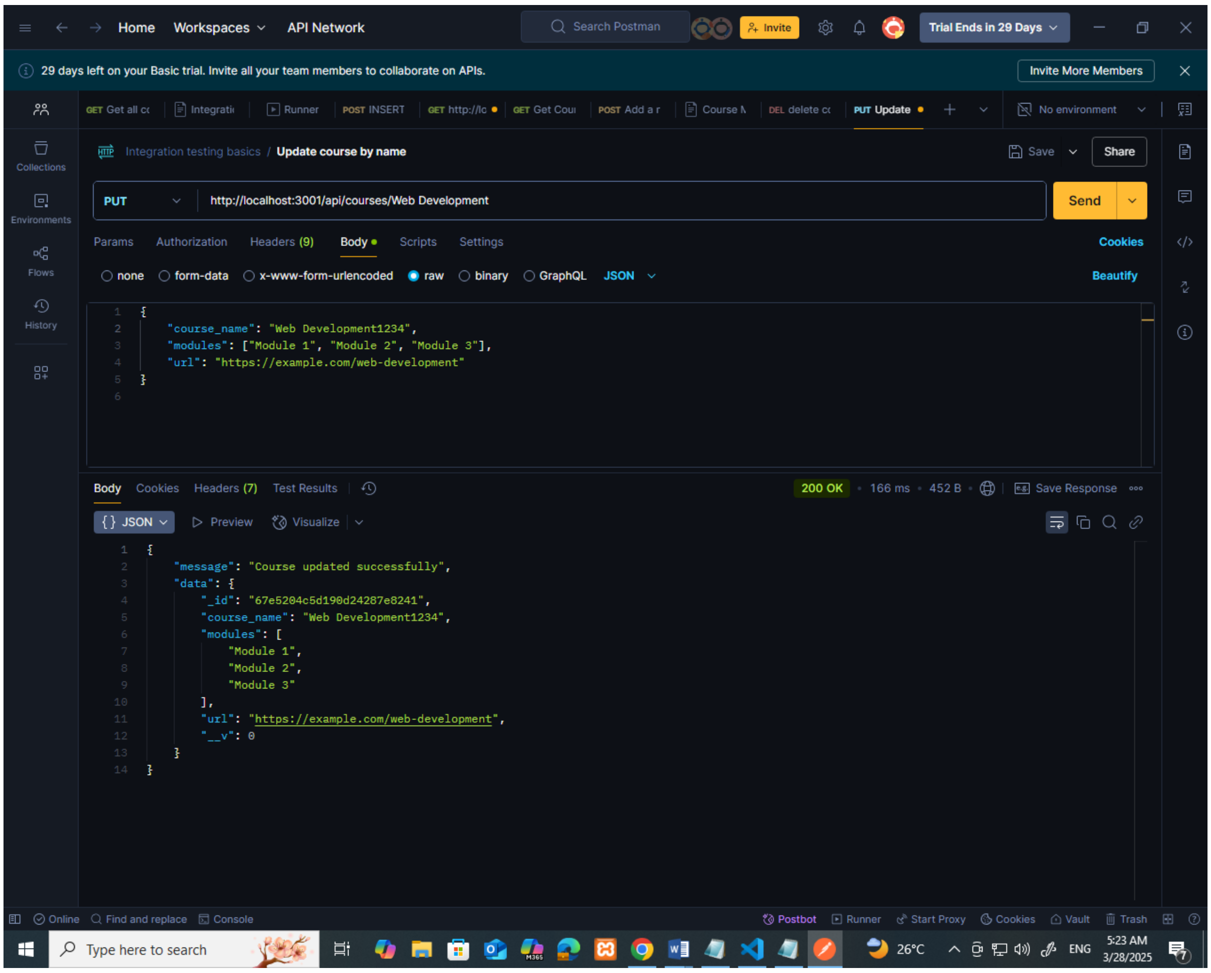
```
{
    "message": "Course added successfully",
    "data": {
        "course_name": "Web Development",
        "modules": [
            "Module 1",
            "Module 2",
            "Module 3"
        ],
        "url": "https://example.com/web-development",
        "_id": "67e5e418dcbed908f4ac7640",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

Figure 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:



PUT http://localhost:3001/api/courses/Web Development

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

Response: 200 OK, 166 ms, 452 B

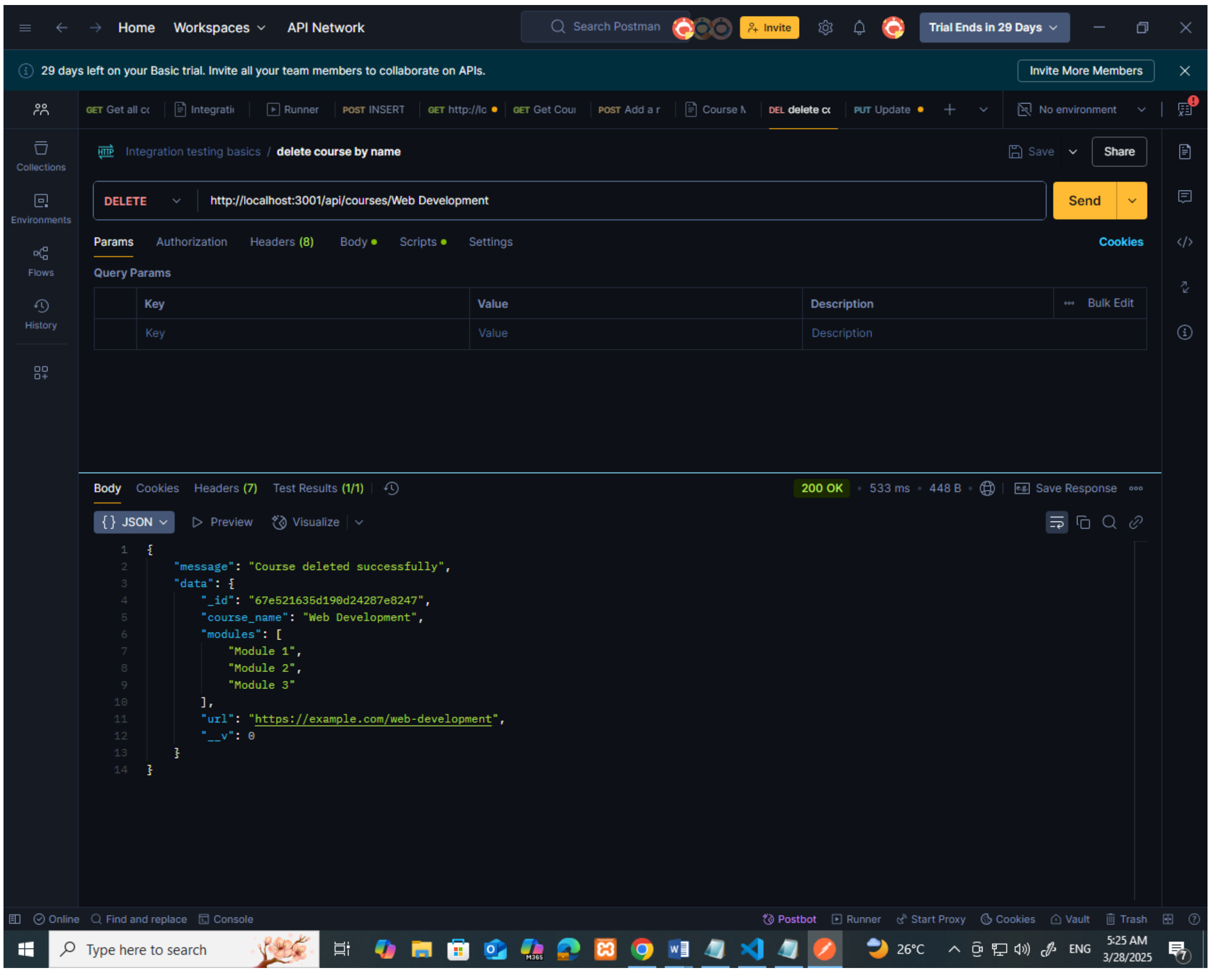
```
{
    "message": "Course updated successfully",
    "data": {
        "_id": "67e5204c5d190d24287e8241",
        "course_name": "Web Development1234",
        "modules": [
            "Module 1",
            "Module 2",
            "Module 3"
        ],
        "url": "https://example.com/web-development",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

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:


29 days left on your Basic trial. Invite all your team members to collaborate on APIs.

Integration testing basics / delete course by name

```
DELETE http://localhost:3001/api/courses/Web Development
```

200 OK • 533 ms • 448 B

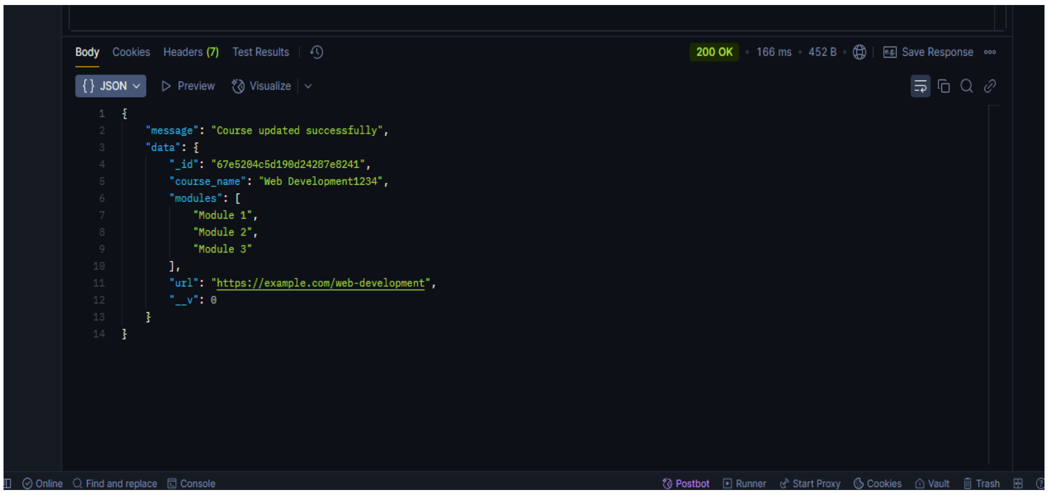
```
{
    "message": "Course deleted successfully",
    "data": {
        "_id": "67e521635d190d24287e8247",
        "course_name": "Web Development",
        "modules": [
            "Module 1",
            "Module 2",
            "Module 3"
        ],
        "url": "https://example.com/web-development",
        "__v": 0
    }
}
```

Figure 4: Delete course by name

NPM Commands

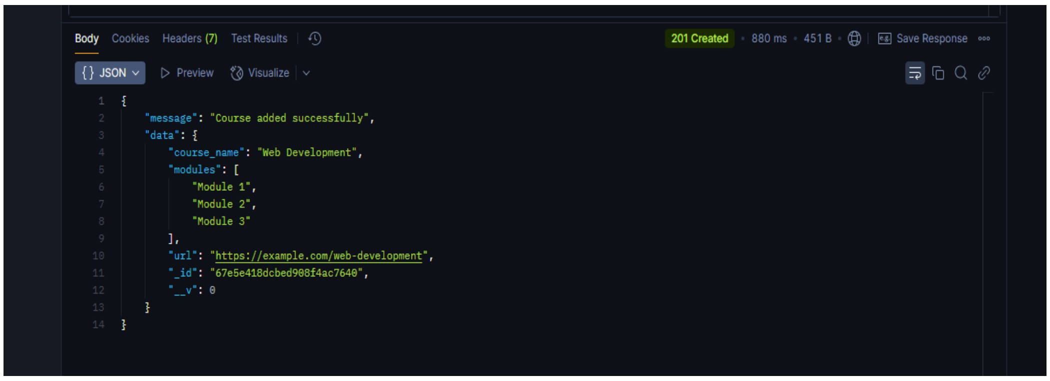
Command	Description
npm install	Installs all dependencies required for the project.
npm start	Starts the server and serves the API.
npm run dev	Starts the server in development mode with hot reloading enabled.
npm test	Runs test cases to verify the functionality of the API and system.

HTTP Status Codes

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

```
{
    "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
    }
}
```

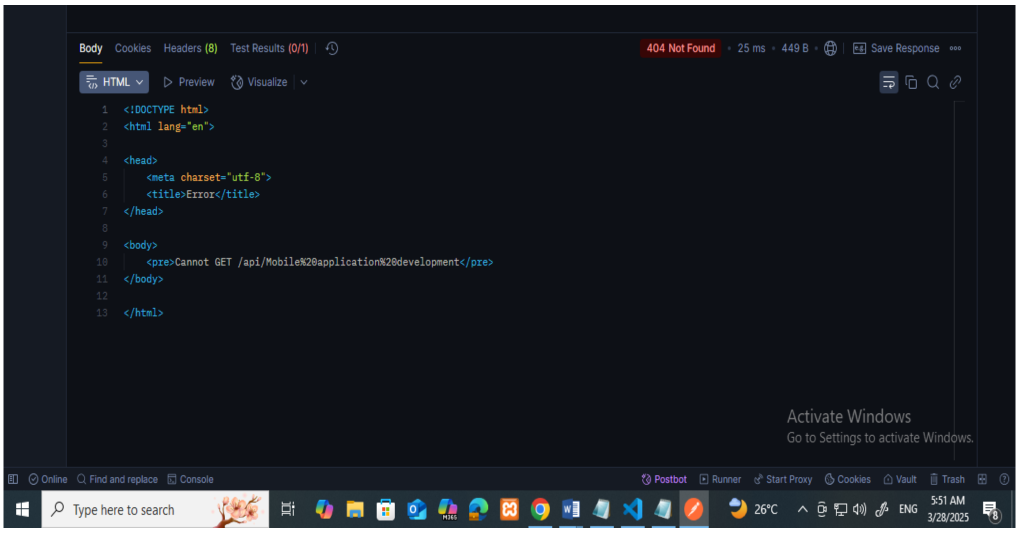
Figure 5:200OK

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

```
{
    "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 6:201 CREATED

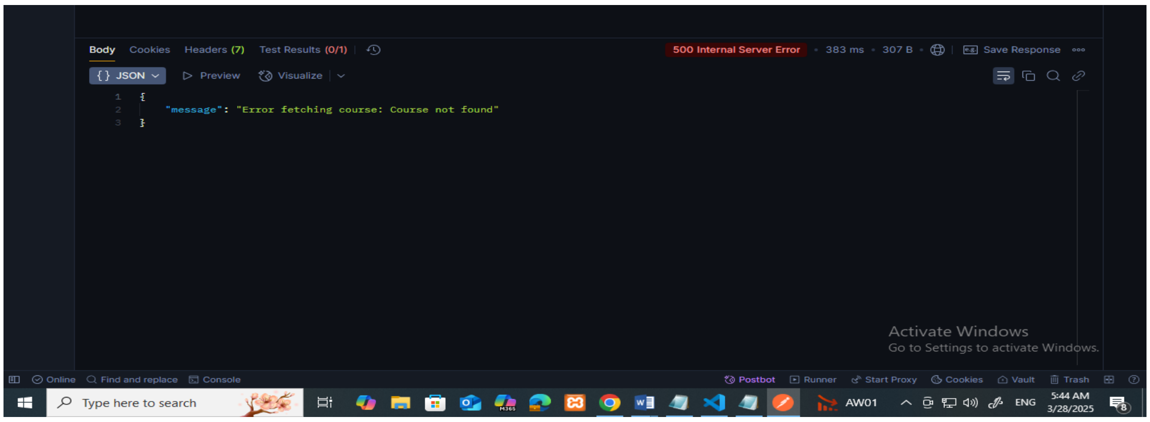
- **404 Not Found: If the resource (e.g., course) does not exist.**



```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="utf-8">
    <title>Error</title>
</head>

<body>
    <pre>Cannot GET /api/Mobile%20application%20development</pre>
</body>

</html>
```

Figure 7:404 NOT FOUND

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



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

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"
}
```

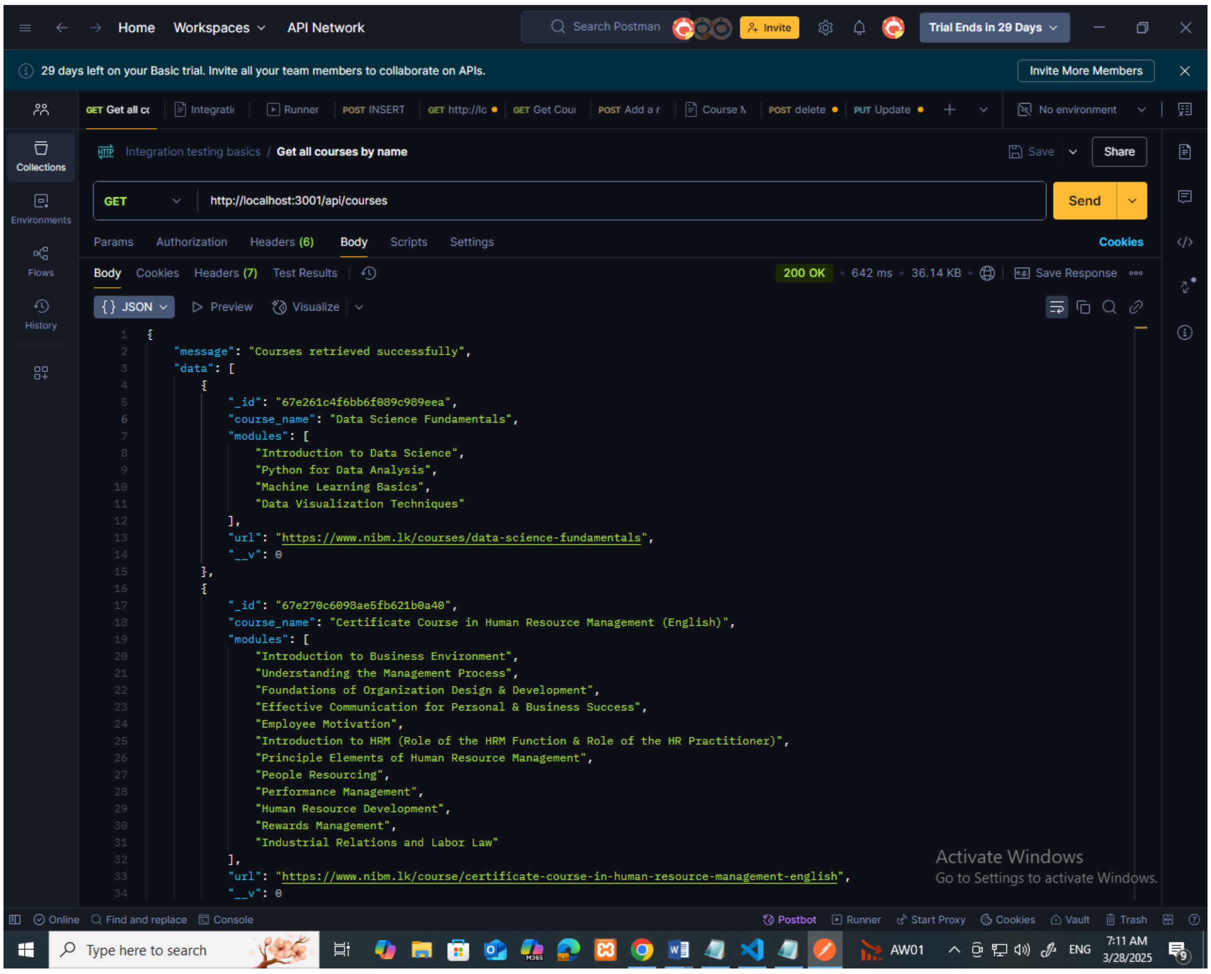
Read All Courses (GET)

Endpoint:/api/courses

MethoE

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
  }
}
```


Integration testing basics / Get all courses by name

GET http://localhost:3001/api/courses

200 OK · 642 ms · 36.14 KB

```
{
    "message": "Courses 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
        },
        {
            "_id": "67e270c6098ae5fb621b0a40",
            "course_name": "Certificate Course in Human Resource Management (English)",
            "modules": [
                "Introduction to Business Environment",
                "Understanding the Management Process",
                "Foundations of Organization Design & Development",
                "Effective Communication for Personal & Business Success",
                "Employee Motivation",
                "Introduction to HRM (Role of the HRM Function & Role of the HR Practitioner)",
                "Principle Elements of Human Resource Management",
                "People Resourcing",
                "Performance Management",
                "Human Resource Development",
                "Rewards Management",
                "Industrial Relations and Labor Law"
            ],
            "url": "https://www.nibm.lk/course/certificate-course-in-human-resource-management-english",
            "__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

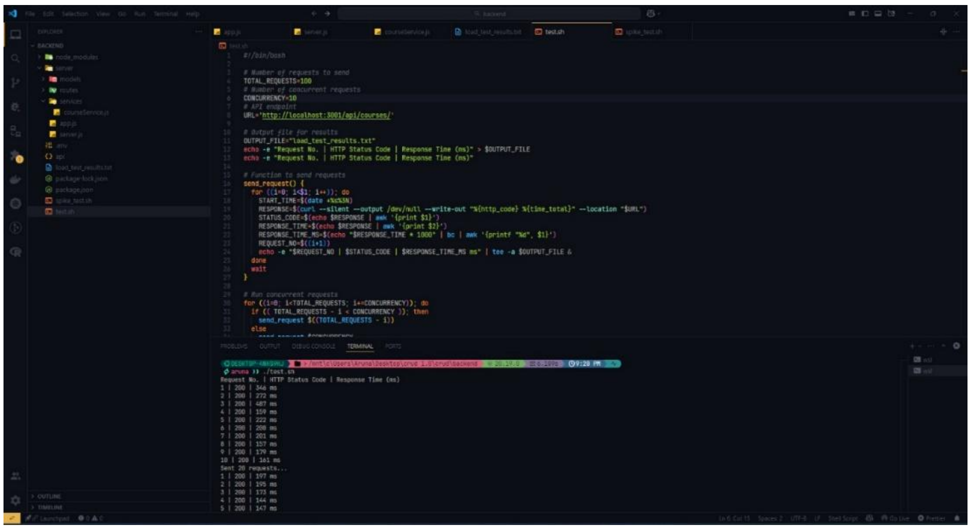
TC-02	API stability under load	100 total requests	No server crashes, consistent response times	No crashes, but response time varies	Pass/Fail
TC-03	Response time tracking	Capture response times per request	Average response time < 300ms	Some responses > 300ms	Pass/Fail based on threshold

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.



```bash
#!/bin/bash

# Number of requests to send
TOTAL_REQUESTS=100
# Number of concurrent requests
CONCURRENCY=10
# API endpoint
URL='http://localhost:3001/api/courses/'

# Output file for results
OUTPUT_FILE="load_test_results.txt"
echo -e "Request No. | HTTP Status Code | Response Time (ms)" > $OUTPUT_FILE
echo -e "Request No. | HTTP Status Code | Response Time (ms)"

# Function to send requests
send_request() {
  for ((i=0; i<$1; i++)); do
    START_TIME=$(date +%s%3N)
    RESPONSE=$(curl --silent --output /dev/null --write-out "%{http_code} %{time_total}" --location "$URL")
    STATUS_CODE=$(echo $RESPONSE | awk '{print $1}')
    RESPONSE_TIME=$(echo $RESPONSE | awk '{print $2}')
    RESPONSE_TIME_MS=$(echo "$RESPONSE_TIME * 1000" | bc | awk '{printf "%d", $1}')
    REQUEST_NO=$((i+1))
    echo -e "$REQUEST_NO | $STATUS_CODE | $RESPONSE_TIME_MS ms" | tee -a $OUTPUT_FILE &
  done
  wait
}

# Run concurrent requests
for ((i=0; i<TOTAL_REQUESTS; i+=CONCURRENCY)); do
  if (( TOTAL_REQUESTS - i < CONCURRENCY )); then
    send_request $((TOTAL_REQUESTS - i))
  else
```

```
aruna )) ./test.sh
Request No. | HTTP Status Code | Response Time (ms)
1 | 200 | 346 ms
2 | 200 | 272 ms
3 | 200 | 487 ms
4 | 200 | 159 ms
5 | 200 | 222 ms
6 | 200 | 200 ms
7 | 200 | 201 ms
8 | 200 | 157 ms
9 | 200 | 179 ms
10 | 200 | 161 ms
Sent 20 requests...
1 | 200 | 197 ms
2 | 200 | 195 ms
3 | 200 | 173 ms
4 | 200 | 144 ms
5 | 200 | 147 ms
```

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.

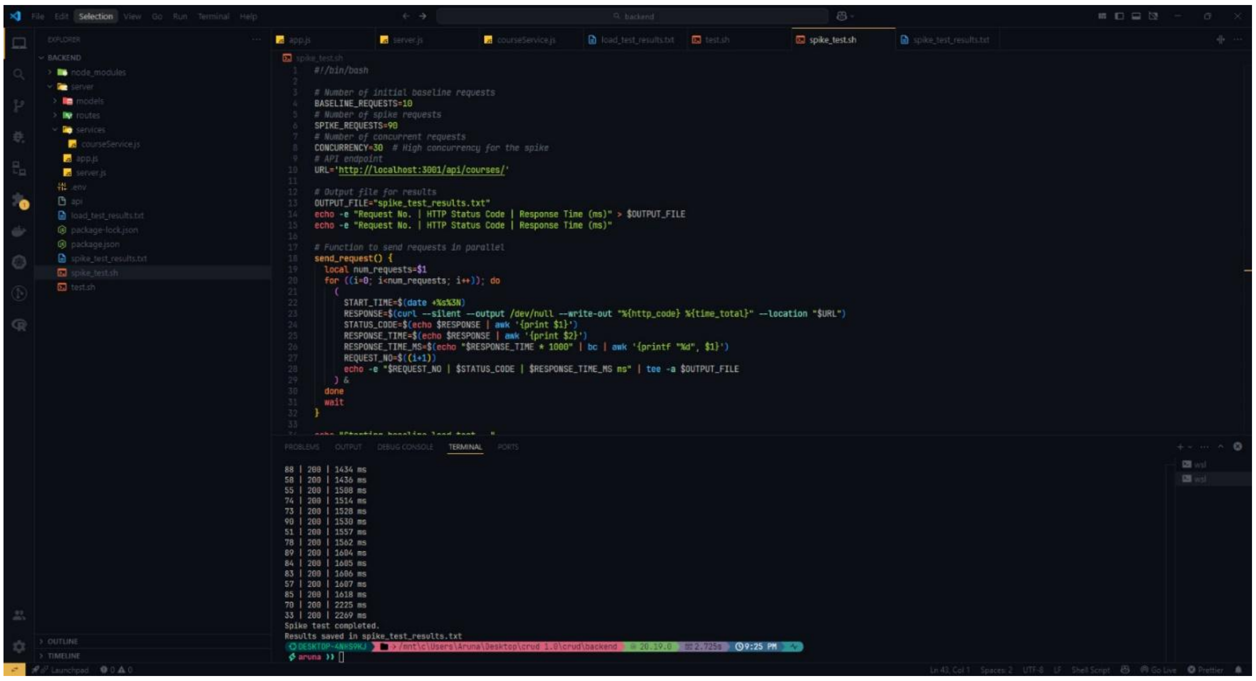
Test Case

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.



```bash
#!/bin/bash

# Number of initial baseline requests
BASELINE_REQUESTS=10
# Number of spike requests
SPIKE_REQUESTS=90
# Number of concurrent requests
CONCURRENCY=30  # High concurrency for the spike
# API endpoint
URL='http://localhost:3001/api/courses/'

# Output file for results
OUTPUT_FILE="spike_test_results.txt"
echo -e "Request No. | HTTP Status Code | Response Time (ms)" > $OUTPUT_FILE
echo -e "Request No. | HTTP Status Code | Response Time (ms)"

# Function to send requests in parallel
send_request() {
  local num_requests=$1
  for ((i=0; i<num_requests; i++)); do
    (
      START_TIME=$(date +%s%3N)
      RESPONSE=$(curl --silent --output /dev/null --write-out "%{http_code} %{time_total}" --location "$URL")
      STATUS_CODE=$(echo $RESPONSE | awk '{print $1}')
      RESPONSE_TIME=$(echo $RESPONSE | awk '{print $2}')
      RESPONSE_TIME_MS=$(echo "$RESPONSE_TIME * 1000" | bc | awk '{printf "%d", $1}')
      REQUEST_NO=$((i+1))
      echo -e "$REQUEST_NO | $STATUS_CODE | $RESPONSE_TIME_MS ms" | tee -a $OUTPUT_FILE
    ) &
  done
  wait
}
```

```
88 | 200 | 1434 ms
58 | 200 | 1436 ms
55 | 200 | 1508 ms
74 | 200 | 1514 ms
73 | 200 | 1528 ms
90 | 200 | 1530 ms
51 | 200 | 1557 ms
78 | 200 | 1562 ms
89 | 200 | 1604 ms
84 | 200 | 1605 ms
83 | 200 | 1606 ms
57 | 200 | 1607 ms
85 | 200 | 1618 ms
70 | 200 | 2225 ms
33 | 200 | 2269 ms
Spike test completed.
Results saved in spike_test_results.txt
DESKTOP-4NHS9KJ  /mnt\c\Users\Aruna\Desktop\crud 1.0\crud\backend  20.19.0  2.725s  9:25 PM
aruna ››
```

Conclusion

Thus, this spike test gauge measures how well the API manages sudden bursts of traffic and then lets-in-insight analytics where it can be vertically optimized for scalability and ideal performance.

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

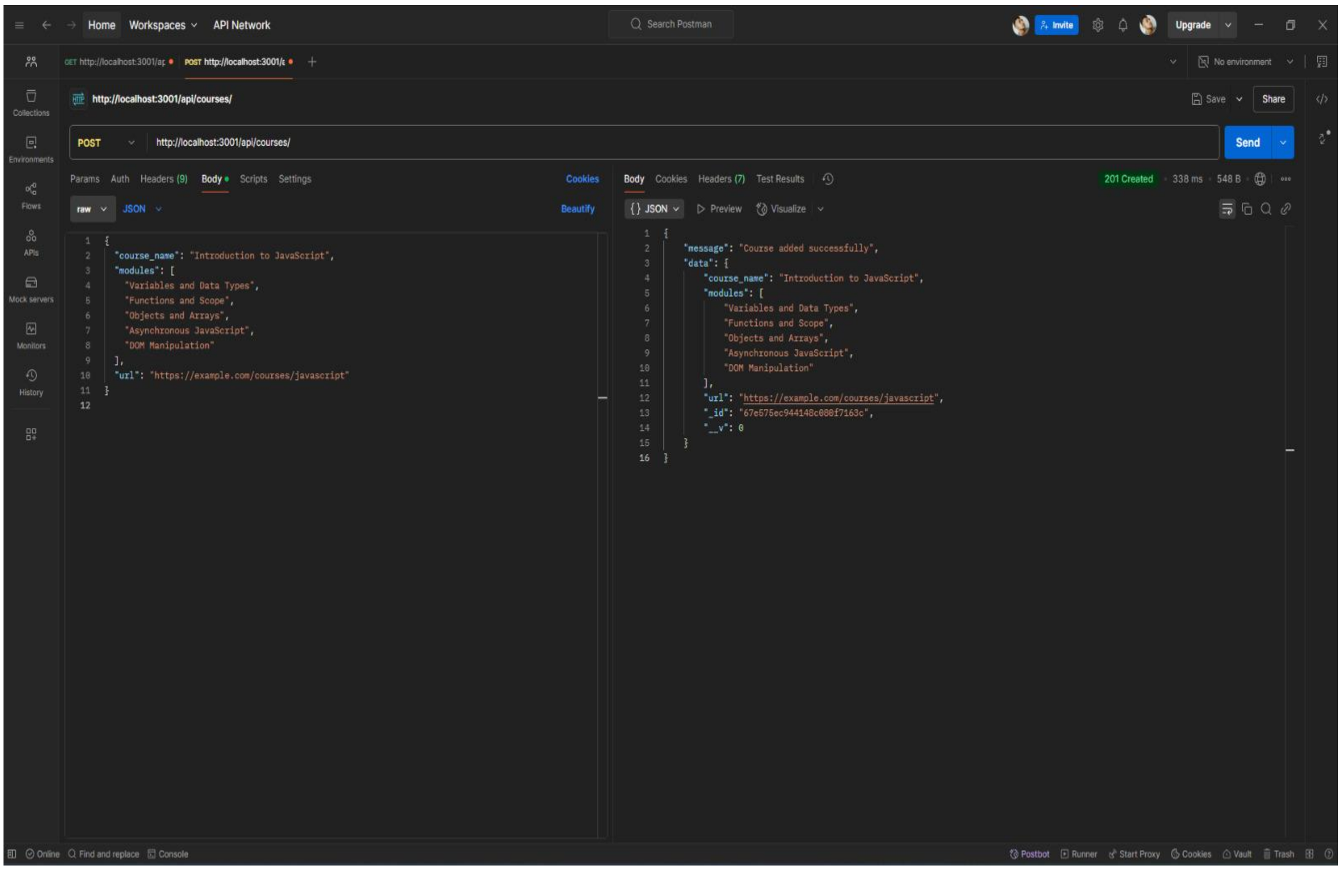
Title: Create Course with Putative Data

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"
}
```


POST http://localhost:3001/api/courses/

```
{
  "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"
}
```

201 Created 338 ms 548 B

```
{
    "message": "Course added successfully",
    "data": {
        "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",
        "_id": "67e575ec944148c080f7163c",
        "__v": 0
    }
}
```

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": ["

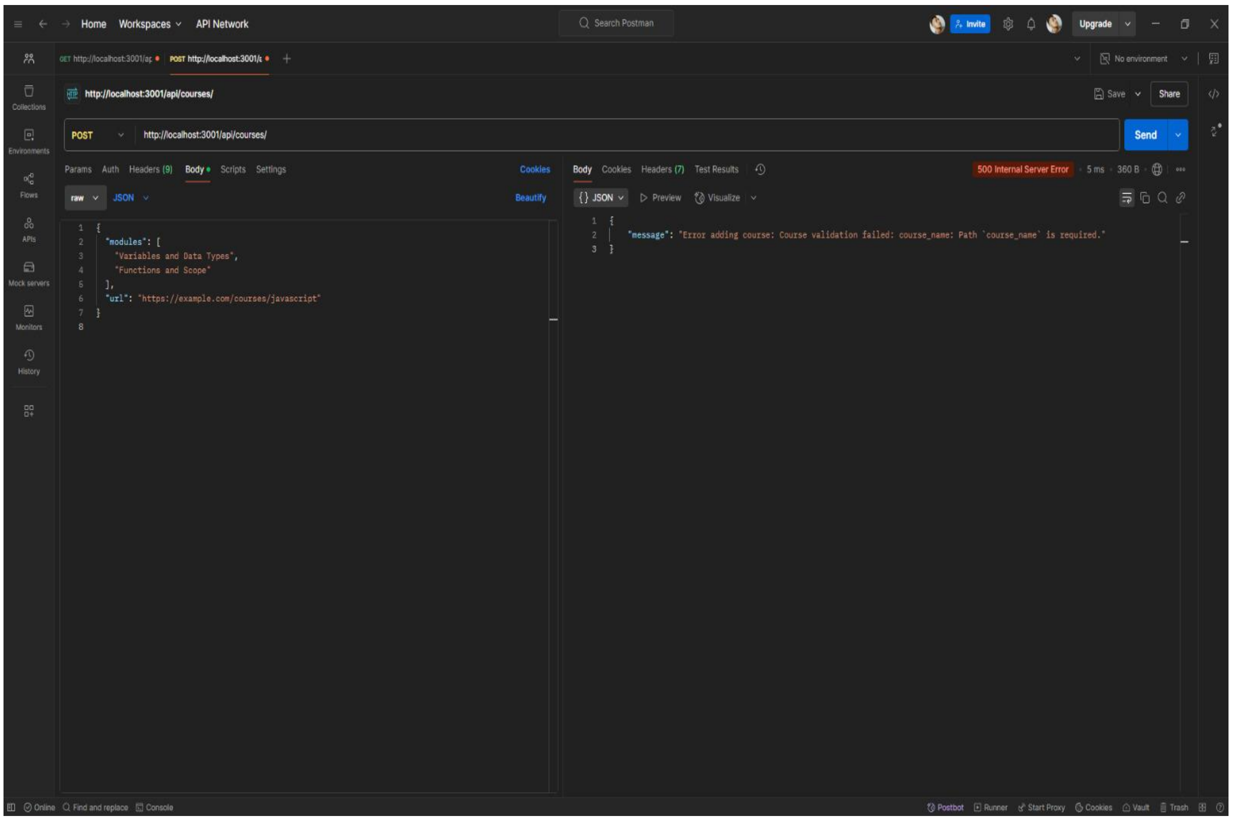
"Variables and Data Types"

"Functions and Scope"

]

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

}



POST http://localhost:3001/api/courses/

500 Internal Server Error · 5 ms · 360 B

```
{
  "modules": [
    "Variables and Data Types",
    "Functions and Scope"
  ],
  "url": "https://example.com/courses/javascript"
}
```

```
{
  "message": "Error adding course: Course validation failed: course_name: Path `course_name` is required."
}
```

Figure 12:negative test

Expected Output:

Status Code: 400 Bad Request

Error Message: "course name is required"

6.4 Edge Case

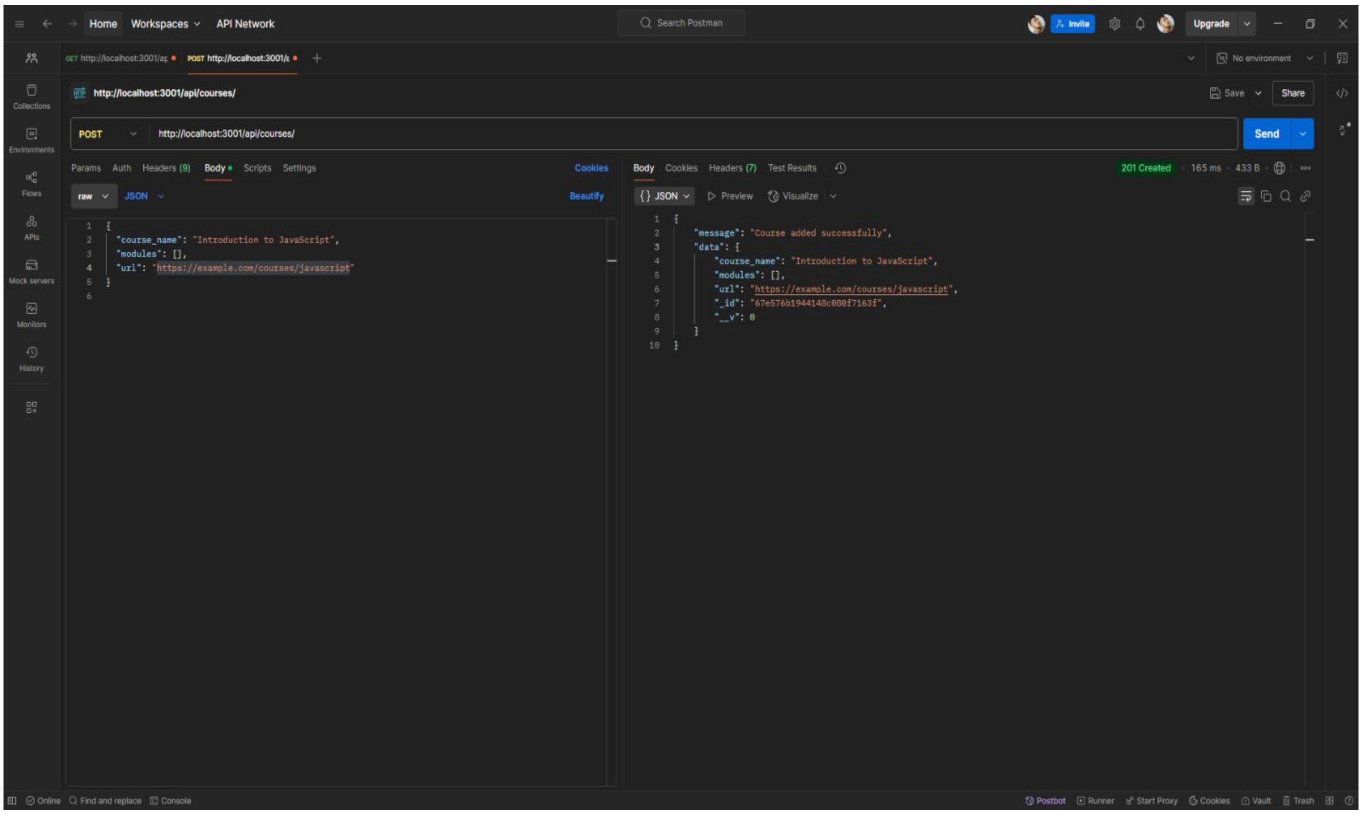
Test Name: Empty Array For Modules

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"

}
```



POST http://localhost:3001/api/courses/

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

201 Created · 165 ms · 433 B

```
{
    "message": "Course added successfully",
    "data": {
        "course_name": "Introduction to JavaScript",
        "modules": [],
        "url": "https://example.com/courses/javascript",
        "_id": "67e576b1944148c080f7163f",
        "__v": 0
    }
}
```

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**

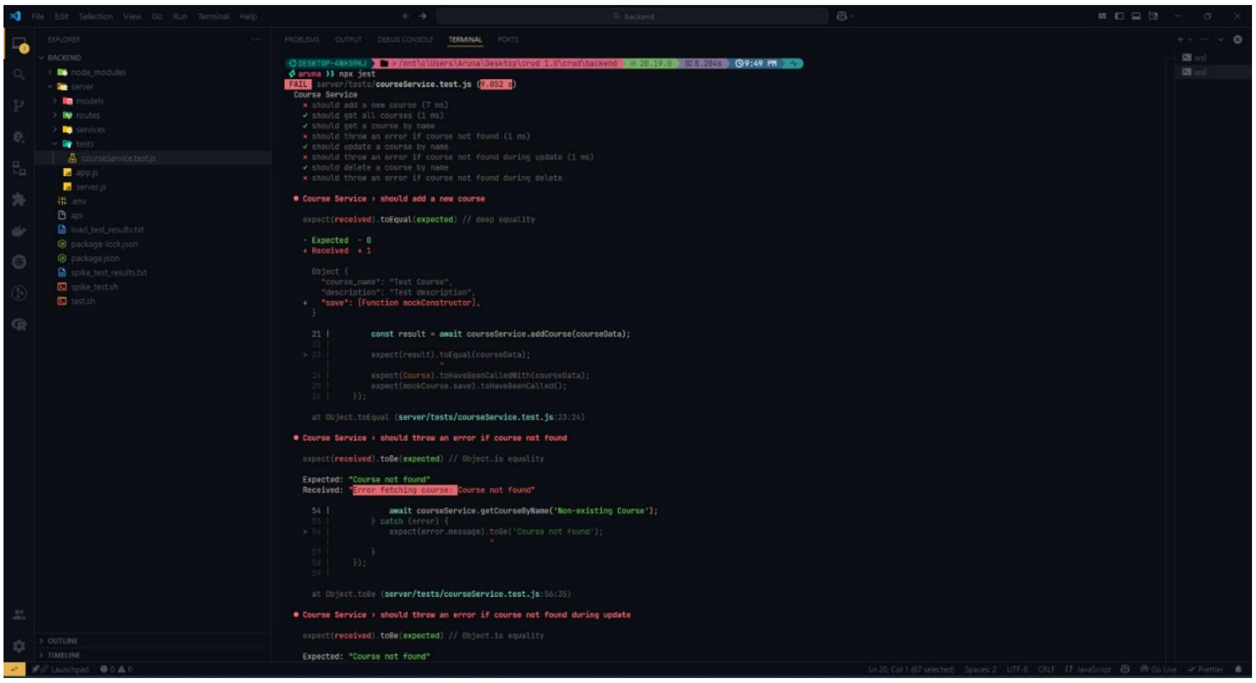
TC_02	Fetch an existing course	"Test Course"	Returns course details	Pass
TC_03	Fetch a non-existing course	"Non-existing Course"	Returns error message "Course not found"	Failing (Mismatch in error message)

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)

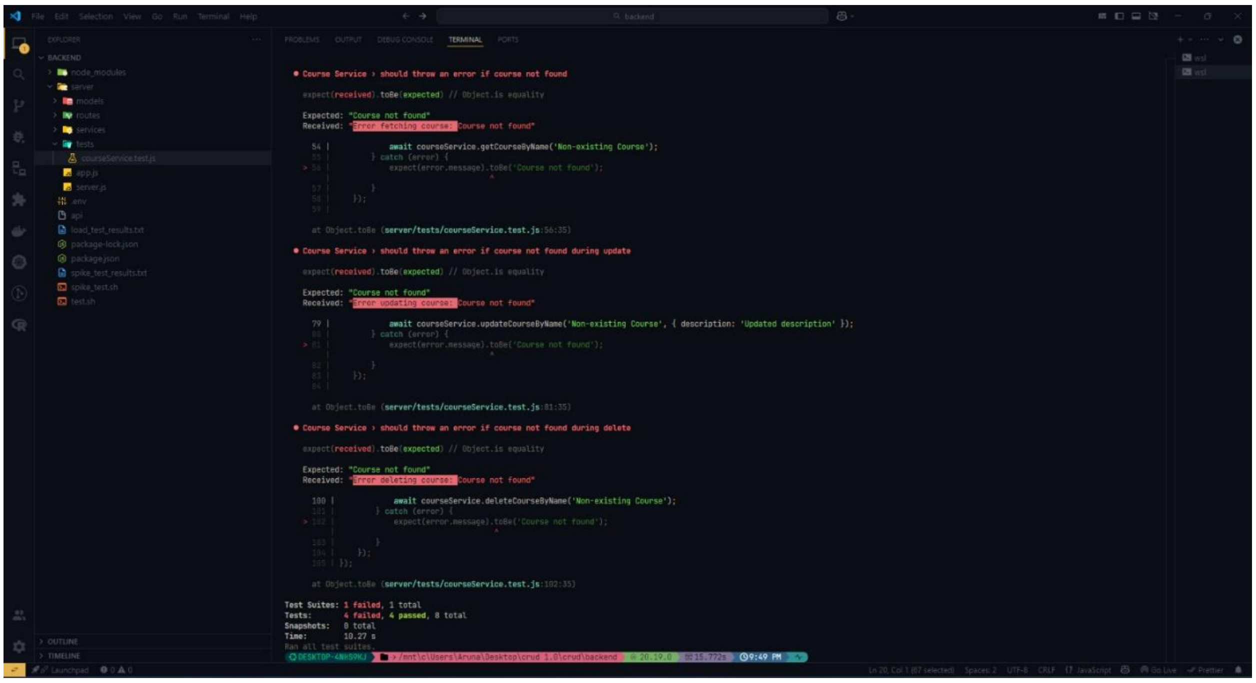
4. Delete a Course				
Test Case ID	**Description**	**Input**	**Expected Output**	**Status**

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



```
DESKTOP-4NHS9KJ  ~/mnt\c\Users\Aruna\Desktop\crud 1.0\crud\backend  20.19.0  8.204s  9:49 PM
aruna )) npx jest
FAIL  server/tests/courseService.test.js (9.852 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 delete a course by name
    × 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 |
    > 23 |         expect(result).toEqual(courseData);
         |                        ^
      24 |         expect(Course).toHaveBeenCalledWith(courseData);
      25 |         expect(mockCourse.save).toHaveBeenCalled();
      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 fetching 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"
```



```
  ● Course Service › should throw an error if course not found

    expect(received).toBe(expected) // Object.is equality

    Expected: "Course not found"
    Received: "Error fetching 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 updating course: Course not found"

      79 |             await courseService.updateCourseByName('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 deleting course: Course not found"

      100 |             await courseService.deleteCourseByName('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:102:35)

Test Suites: 1 failed, 1 total
Tests:       4 failed, 4 passed, 8 total
Snapshots:   0 total
Time:        10.27 s
Ran all test suites.
DESKTOP-4NHS9KJ  ~/mnt\c\Users\Aruna\Desktop\crud 1.0\crud\backend  20.19.0  15.772s  9:49 PM
```

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