< Learning Management System >

Project documentation

Student: Hoza Violeta Maria

Group: 30434

MINISTRY OF EDUCATION

TECHNICAL UNIVERSITY
OF CLUJ-NAPOCA, ROMANIA

Contents

I Project specification

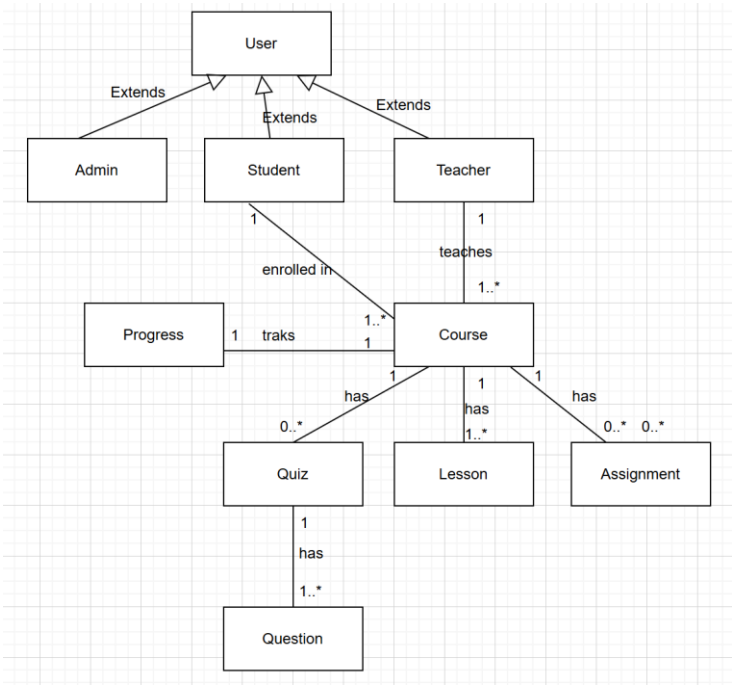
The Learning Management System (LMS) is a web-based platform designed to facilitate online learning. It enables teachers to create and manage courses, while allowing students easier access to educational resources.

There will be 3 types of users: admin, teachers and students.

Key features:

- The admin manages users and monitors the activity.
- Teachers can upload, update, and delete lessons easily.
- Students can enrol in courses and participate in interactive lessons.
- Interactive exercises and quizzes with auto-grading.
- Points, badges, and streaks to boost motivation.
- Visual dashboards for students and teachers to track performance and engagement.
- Course recommendation based on a student's progress and interests.

1.1 Domain Model Diagram



II Use-Case model

In the Learning Management System, the main actors are teachers and students, each with distinct roles and permissions. Teachers manage courses, while students enrol in and complete them. The system also involves potential stakeholders, such as school administrators, content creators and even parents in certain cases.

2.1 Users and stakeholders

- **Users**

  The admin is the one that manages teacher and students accounts, monitors the activity and issues reports.

  The teachers are content creators and course managers. Their main responsibilities include:
  - creating, updating, and deleting courses
  - uploading lessons in various formats
  - designing interactive exercises and quizzes
  - engaging in discussions forums and responding to students' questions.

  The students are the learners who benefit from the system's educational resources. Their key activities include:
  - enrolling in courses and accessing lessons anytime
  - completing interactive exercises, quizzes and assignments
  - using flashcards for concepts retention
  - participating in discussion forums and group chats
  - earning badges and points
  - tracking their learning progress on a dashboard.

- **Stakeholders**

  Besides teachers and students, other stakeholders that may be involved in or benefit from the system are:
  - Educational institutions (schools, universities): can use the LMS to manage and deliver online courses.
  - Parents: parents might track student progress, receiving notifications about assignments and test results.
  - Corporate training teams: companies may use the platform for employee training programs.

2.2 Use-Case identification

Use Case Name: Create account

Level: < User-Goal, Subfunction, Summary >.

Main actor: Teacher/Student

Main success scenario: The user will create an account if there isn't already an existing one. For this, the user will provide the necessary information (name, age, phone, email). After that a button will be clicked and if all the rules were obeyed, then the account will be created.

Extension: If the user has already an account affiliated to its email, then a warning message will be displayed. The same happens if the mandatory fields were not completed or have been filled incorrectly.

Use Case Name: Create course

Level: < User-Goal, Subfunction, Summary >.

Main actor: Teacher

Main success scenario: The teacher will create a new course by providing all required information (title, description, start date, end date, category). The teacher will then add lessons, quizzes with automatically graded questions, and assignments to the course. After submitting the course creation form, the system will validate the information and make the course available for student enrolment.

Extension: If required fields are missing or invalid, the system will display error messages and prompt the teacher to correct the information. If a course with the same title already exists for this teacher, the system will notify the teacher and suggest using a different title.

Use Case Name: View course catalogue

Level: < User-Goal, Subfunction, Summary >.

Main actor: Student

Main success scenario: The student accesses the course catalogue section of the application. The system displays a list of all available courses with basic information (title, instructor, description summary, duration). The student can browse through the catalogue and view detailed information about any course.

Extension: If no courses are available or if all courses are filtered out by search criteria, the system displays a message indicating that no courses match the current criteria.

Use Case Name: Enrol in course

Level: < User-Goal, Subfunction, Summary >.

Main actor: Student

Main success scenario: The student browses available courses and selects one to enroll in. After clicking the "Enrol" button, the system will add the course to the student's enrolled courses list, create initial progress records, and provide access to all course materials.

Extension: If the course requires prerequisites that the student hasn't completed, the system will display a message indicating which courses must be completed first. If the course has reached its maximum enrolment capacity, the system will notify the student and offer to place them on a waiting list.

Use Case Name: View lesson content

Level: < User-Goal, Subfunction, Summary >.

Main actor: Student

Main success scenario: The student selects a lesson from an enrolled course. The system displays the lesson content (text, images, videos) and marks the lesson as viewed in the student's progress record once they reach the end of the content.

Extension: If the lesson has prerequisites that haven't been completed, the system notifies the student and recommends completing those first.

Use Case Name: Take quiz

Level: < User-Goal, Subfunction, Summary >.

Main actor: Student

Main success scenario: The student navigates to a quiz within an enrolled course and begins the quiz. The system displays quiz questions one by one or all at once depending on the quiz configuration. The student answers all questions and submits the quiz. The system automatically grades the quiz using predefined correct answers, displays the score immediately, and records the result in the student's progress record.

Extension: If the quiz has a time limit and the student exceeds it, the system automatically submits whatever answers have been provided. If the student loses connection during the quiz, the system will save their progress and allow them to resume where they left off.

Use Case Name: Grade assignments

Level: < User-Goal, Subfunction, Summary >.

Main actor: Teacher

Main success scenario: The teacher accesses a list of submitted assignments that require grading. For each submission, the teacher can view the student's work, provide feedback comments, and assign a grade. Once grading is complete, the system updates the student's progress record and sends a notification to the student.

Extension: If the teacher needs more information to properly grade an assignment, they can request additional clarification from the student, which pauses the grading process until the student responds.

Use Case Name: Generate course certificate

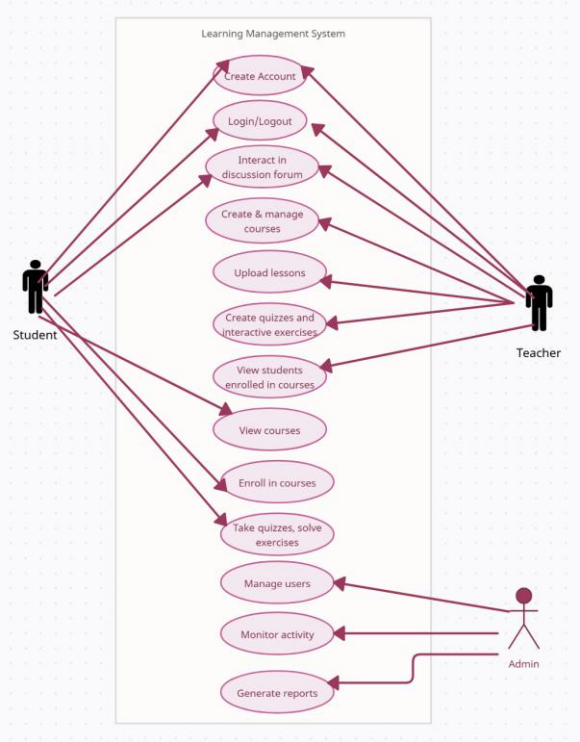
Level: < User-Goal, Subfunction, Summary >.

Main actor: Administrator

Main success scenario: When a student completes all required components of a course with passing grades, the system automatically generates a digital certificate of completion that includes the student's name, course title, and completion date. The system makes the certificate available for download and adds it to the student's achievements record.

Extension: If the student has completed the course but has not achieved the minimum required grade for certification, the system notifies the student about which components need improvement to qualify for the certificate.

2.3 UML Use-Case diagram



III Architectural design

The Learning Management System is a web-based application designed with a multi-layered architecture to support online learning interactions between teachers and students. The system employs a layered architecture, adapted from the Model-View-Controller (MVC) pattern, to create a robust and flexible educational platform.

3.1 Conceptual architecture

The LMS follows a layered architecture, meaning different parts of the system handle different tasks. This makes it easier to maintain and scale.

- Presentation Layer (Frontend): web pages for different user roles (admin, teacher, student), forms for user interactions, dashboards and data visualization – implemented using HTML templates with Jinja2, CSS, JavaScript
- Controller Layer (API Routes): HTTP request handling, routing, and response formatting

- Business Logic Layer (Services): contains all the logic (grading quizzes, checking permissions, validating inputs) and handles errors (like if a student tries to submit a quiz after the deadline).
- Data Access Layer (Models): stores the models we work with (users, courses, lessons, grades) using SQLAlchemy ORM
- Database Layer: communicates with the database to save or retrieve data.

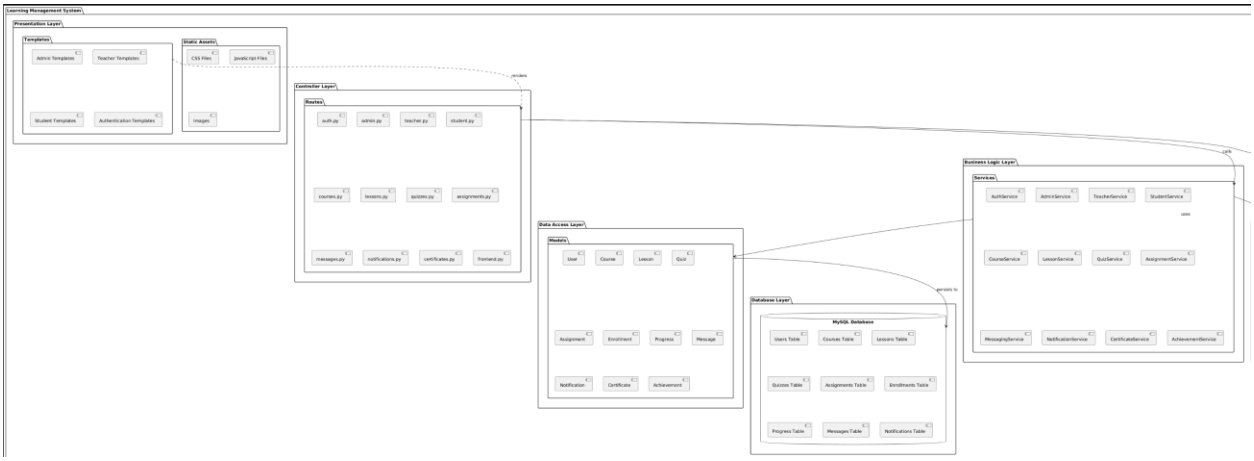
The layered approach allows us to easily modify individual components, add new features without disrupting existing functionality, and scale different parts of the application independently.

3.2 Design Patterns

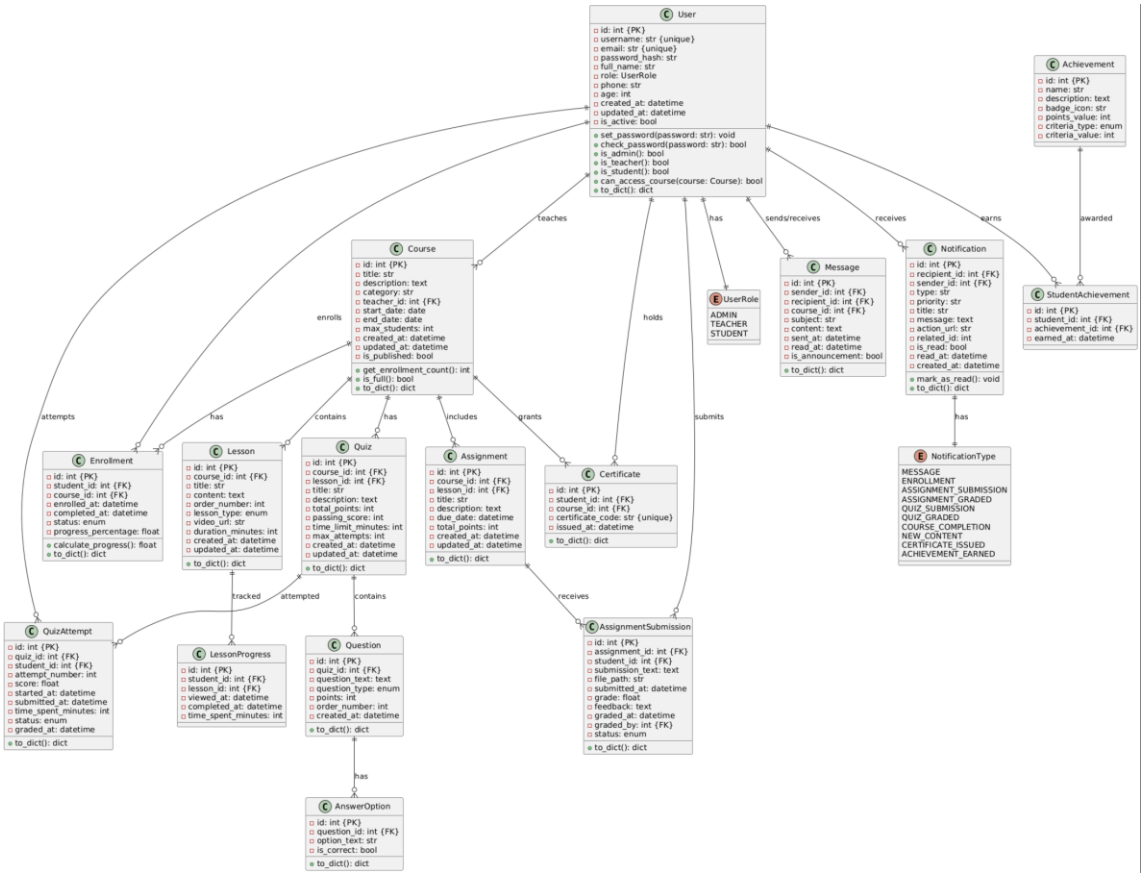
The Learning Management System implements several key design patterns that contribute to its architectural integrity and maintainability:

- The **Repository Pattern** is implemented through SQLAlchemy ORM models, which provide an abstraction layer over database operations and enable object-relational mapping without requiring direct SQL manipulation.
- The **Factory Pattern** is utilized in the application initialization process through the create_app() function, which dynamically instantiates Flask applications with environment-specific configurations for development, testing, and production scenarios.
- The **Decorator Pattern** is extensively employed for cross-cutting concerns, particularly authentication and authorization, with decorators like @teacher_required(), @admin_required(), and @student_required() that provide clean separation of security logic from business functionality; instead of checking "is this user a teacher?" in every function, we simply add @teacher_required() above the function definition, making the code cleaner.
- The **Strategy Pattern** is evident in the quiz grading system, where different grading strategies are applied based on question types (multiple choice, true/false, short answer) through polymorphic behavior in the QuizService.submit_quiz_attempt() method.
- The **Observer Pattern** is implemented through the notification system in NotificationService, where various events (course completion, assignment submission, quiz grading) automatically trigger notifications to relevant users without tight coupling between the event source and notification recipients.
- The **Singleton Pattern** is applied to the database connection and configuration management, ensuring single instances of critical resources throughout the application lifecycle

3.2 Package diagram



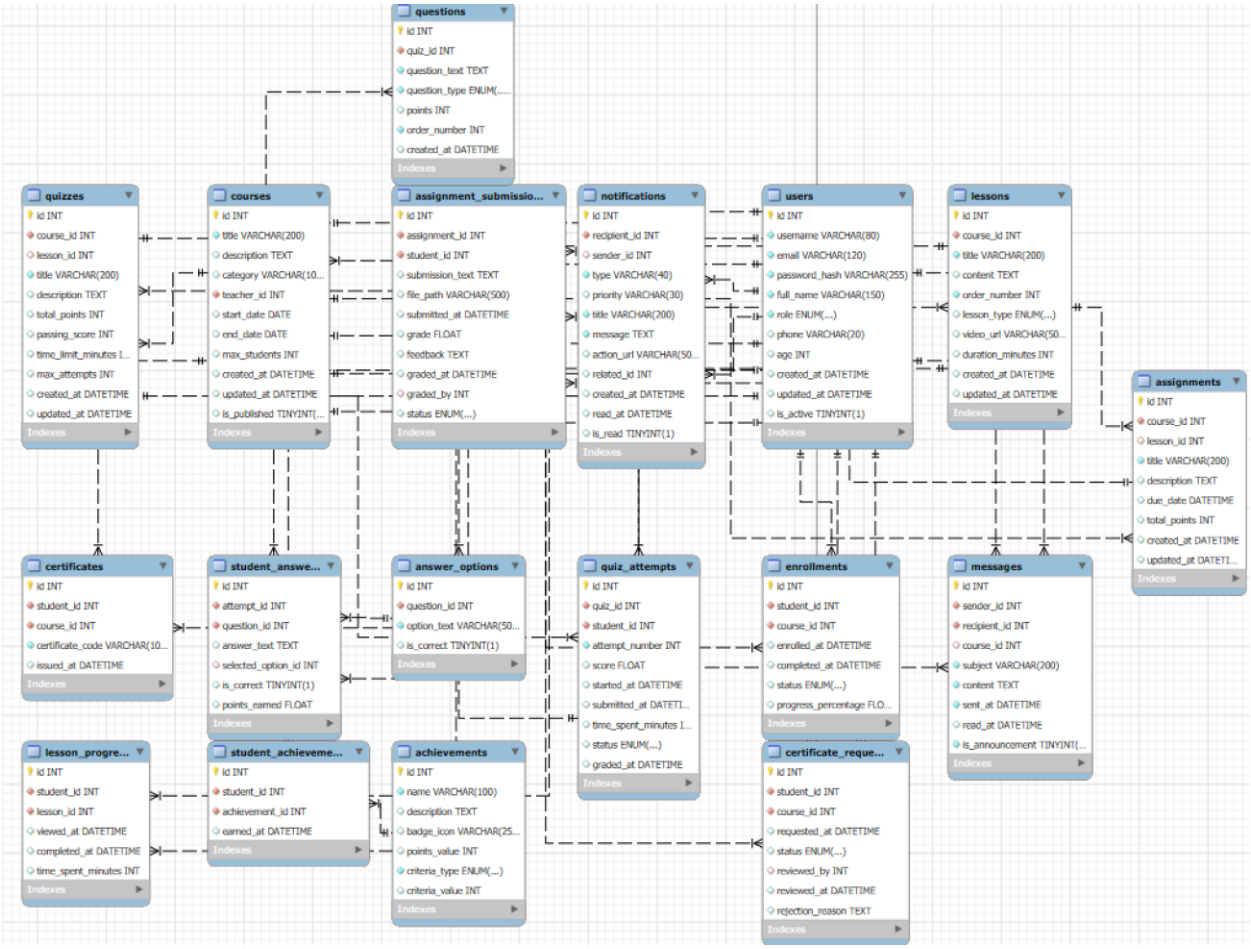
3.3 Class diagram



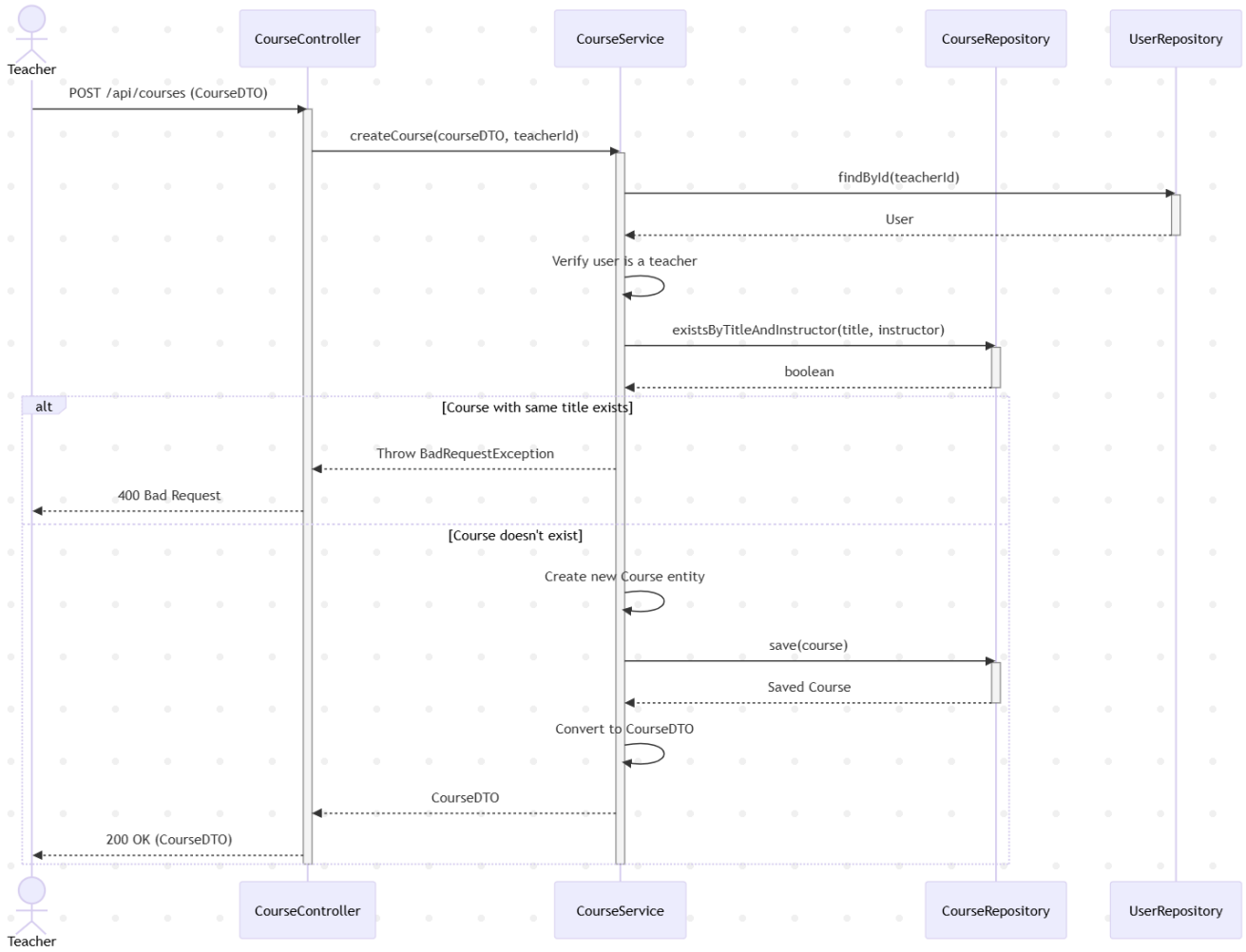
3.4 Database (E-R/Data model) diagram

The LMS follows a relational database model with the following key tables:

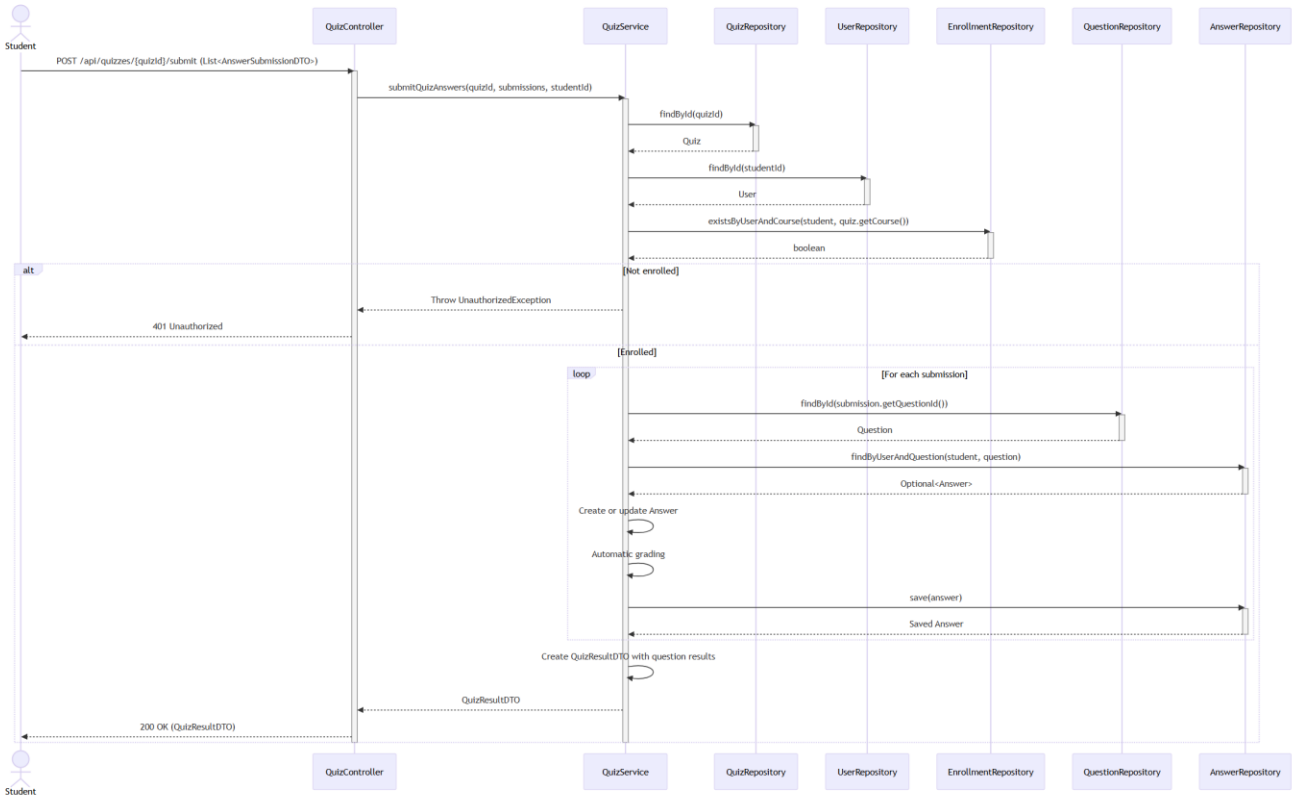
- Users (stores information on admins, teachers, and students)
- Courses (details about each course and its instructor)
- Lessons (course modules, including videos and text content)
- Enrolments (tracking student participation in courses)
- Quizzes and Quiz Attempts, Questions and Answer Options (for assessments and auto-grading)
- Assignment and Assignment Submissions
- Notifications, Certificates



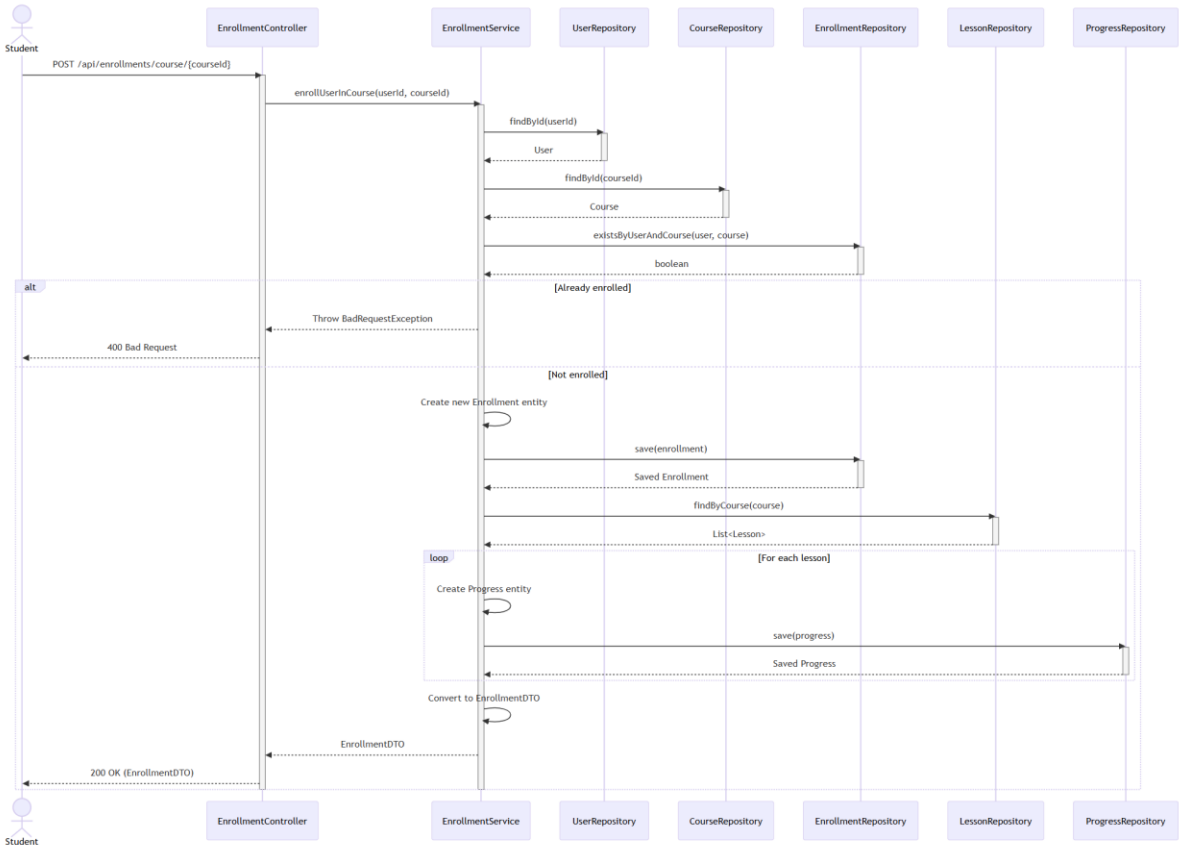
3.5 Sequence diagram



Sequence diagram - Course creation

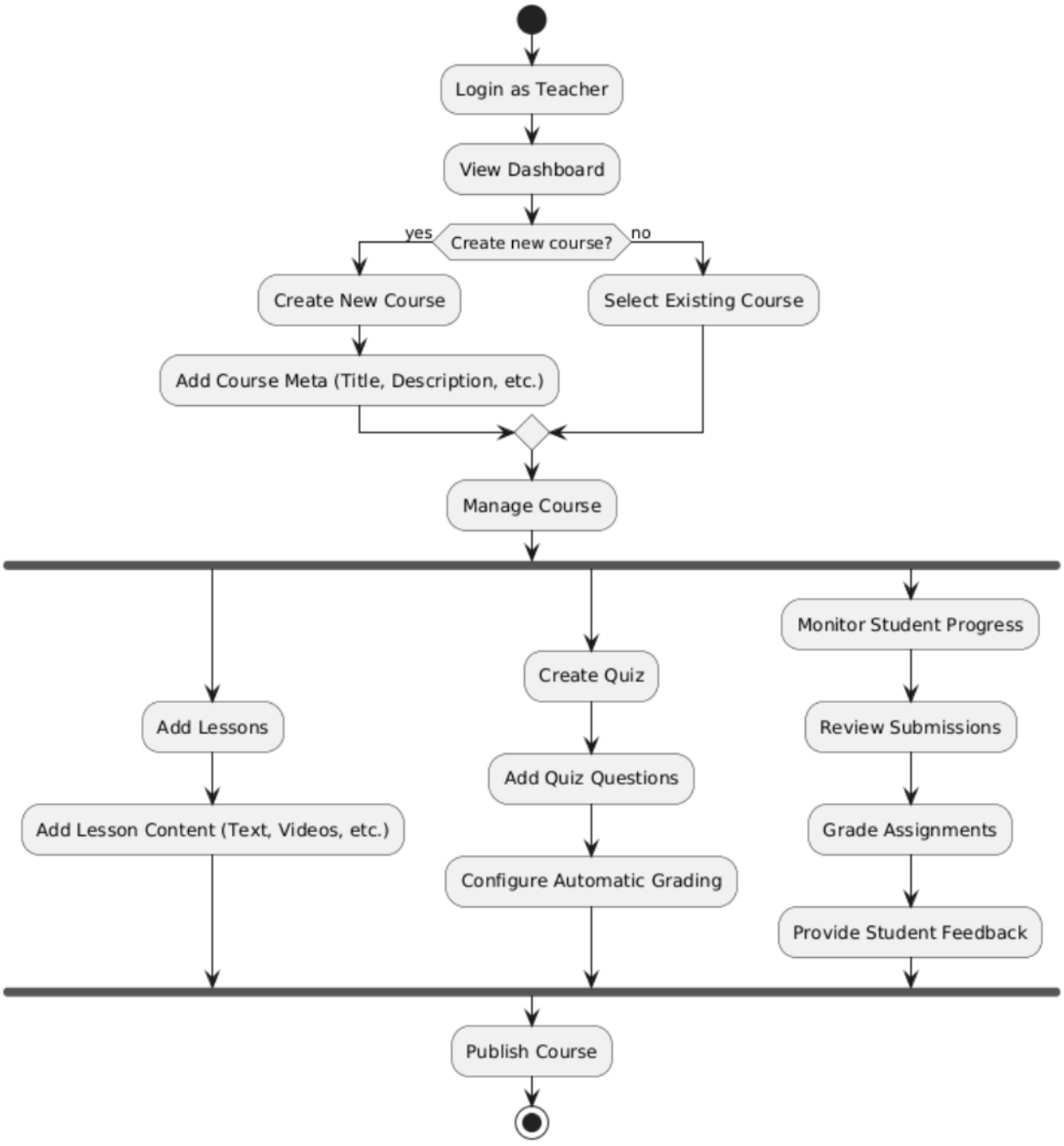


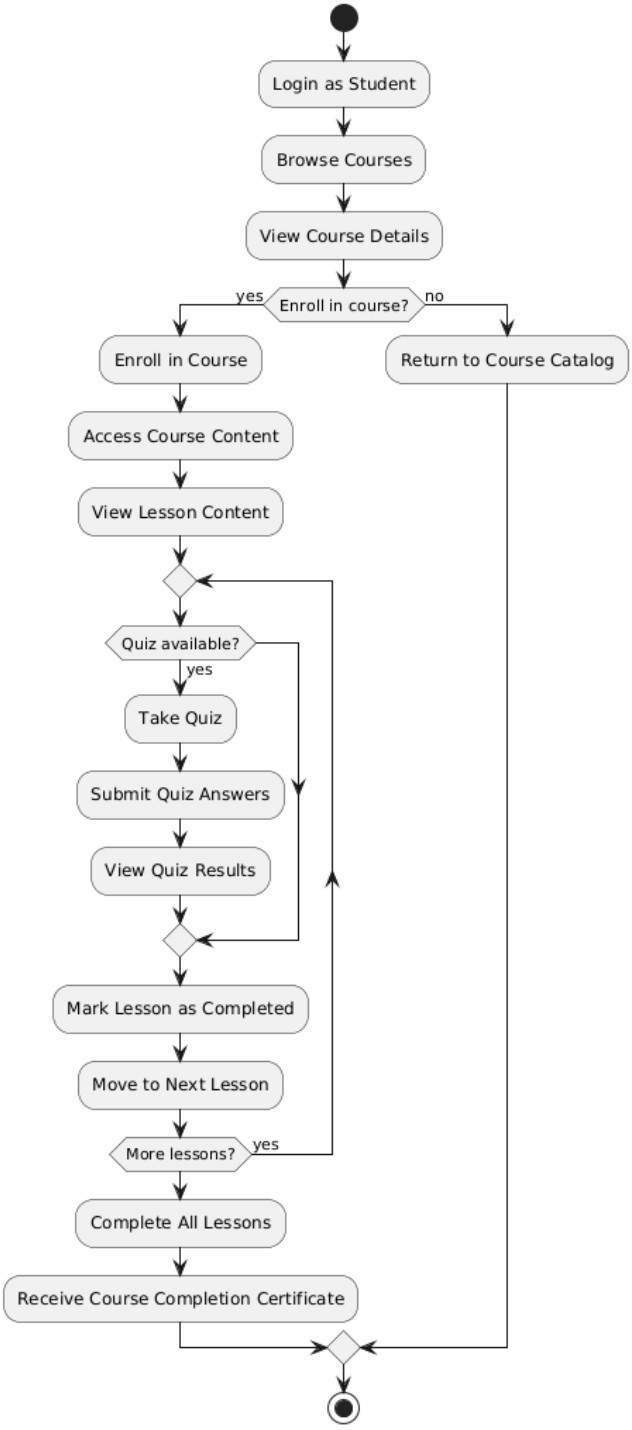
Sequence diagram - Quiz submission



Sequence diagram - Course enrolment

3.6 Activity diagram




Login as Student
Browse Courses
View Course Details
Enroll in course?
yes
no
Enroll in Course
Return to Course Catalog
Access Course Content
View Lesson Content
Quiz available?
yes
Take Quiz
Submit Quiz Answers
View Quiz Results
Mark Lesson as Completed
Move to Next Lesson
More lessons?
yes
Complete All Lessons
Receive Course Completion Certificate

IV Supplementary specifications

4.1 Non-functional requirements

- Performance: quick content retrieval and quiz loading, even when many users are online – the system must minimize latency when fetching course materials, videos, or quizzes.
- Security: the application should implement role-based access to restrict unauthorized actions – only teachers should be able to upload course content, and only students enrolled in a course should access its materials.
- Accessibility across devices: the application should be fully responsive for desktop, tablet, and mobile usage, users can access learning resources at any time, the only pre-requisites being an internet connection and a browser.
- Usability: intuitive interface design with easy navigation between courses and lessons, user-friendly dashboards and simple course enrolment and lesson access.
- Scalability: the application must support multiple concurrent users and handle efficiently the course materials and user progress.
- Reliability: the system must recover gracefully from failures.
- Maintainability: separating UI, logic, and database layers.

4.2 Design constraints

For the backend development, I've decided to use Python as the main programming language because it's versatile, has excellent web development frameworks, and offers rapid development capabilities. I will use Flask for handling API development, routing, and web application structure, along with Flask-JWT-Extended for authentication and security management. SQLAlchemy ORM is used for managing database operations, providing an elegant Python interface for database interactions without writing raw SQL. For the frontend, I will use HTML templates with Jinja2 templating engine combined with JavaScript and CSS because this approach provides server-side rendering, reduces complexity, and creates a cohesive full-stack application within the Flask ecosystem. For the database, I will use MySQL with PyMySQL connector because it's a stable, widely used relational database that ensures data integrity and handles complex relationships between users, courses, and educational content effectively. The frontend and backend communicate seamlessly through Flask's integrated request handling (for example, when you click "Submit answers" in the web form, it sends a POST request to the Flask backend which processes the quiz submission and saves your answers in the MySQL database). The testing framework used is Python's built-in testing capabilities along with Flask's testing utilities, providing comprehensive coverage for both unit and integration testing scenarios.

V Testing

5.1 Testing methods/frameworks

The LMS uses **pytest** to automatically test all the important features like login, course creation, quizzes, and assignments. The tests run in a safe environment that doesn't affect real user data, so we can make sure everything works properly before students and teachers use it. We can run tests for specific parts (like just testing the login system) or test everything at once. This helps us catch bugs early and make sure new features don't break existing ones. The testing shows us how much of our code is actually being tested, so we know we're building a reliable learning platform.

5.2 Future improvements

Looking ahead, there are several enhancements that could significantly improve the Learning Management System's functionality and user experience. Real-time features could be implemented using WebSocket technology to enable live chat between students and teachers, instant notifications, and collaborative learning sessions. The system could benefit from advanced analytics and reporting capabilities using Python data science libraries like Pandas and Matplotlib to provide detailed insights into student performance patterns, course effectiveness, and learning outcomes. Mobile application development using React Native or Flutter would extend accessibility, allowing students to access courses and complete assignments on their smartphones and tablets. Video streaming and storage integration with services like AWS S3 and video processing APIs would enable teachers to upload and manage video lectures more effectively. Machine learning integration using scikit-learn or TensorFlow could power intelligent course recommendations, automated plagiarism detection in assignments, and personalized learning paths based on individual student progress. Enhanced security measures including two-factor authentication, OAuth integration with Google and Microsoft accounts, and advanced encryption for sensitive data would strengthen the platform's security posture. Finally, microservices architecture migration using Docker containers and REST API improvements would enhance scalability, maintainability, and allow for independent deployment of different system components as the user base grows.

VI API Documentation

1. **POST API Calls**: Used to create new resources on the server.
   - POST /api/auth/register
     Request: body (JSON): {
     "username": "violeta_hoza",
     "email": "vio@yahoo.com",
     "password": "Password123",

```
    "full_name": "Violeta Hoza",
    "role": "student",
    "phone": "1234567890",
    "age": 20
}
```

Responses:

- 201 CREATED - User registered successfully

- 400 BAD REQUEST - Invalid data or user already exists

- http://127.0.0.1:5000/teacher/courses/create
  Request:
  body (JSON): {

```
    "title": "Introduction to Python",
    "description": "Learn Python programming from basics",
    "category": "Programming",
    "start_date": "2024-01-15",
    "end_date": "2024-03-15",
    "max_students": 30,
    "is_published": true
}
```

  Responses:

  - 201 CREATED - Course was successfully created
  - 400 BAD REQUEST - Course cannot be created (validation errors)
  - 403 FORBIDDEN - Teacher role required

- POST /api/courses/{course_id}/enroll
  Request: body (JSON): {}
  Responses:
  - 201 CREATED - Enrolled successfully
  - 400 BAD REQUEST - Already enrolled or course full
  - 403 FORBIDDEN - Student role required

- POST /api/lessons
  Request:

body (JSON): {
"course_id": 1,
"title": "Variables and Data Types",
"content": "In this lesson, we'll learn about...",
"order_number": 1,
"lesson_type": "mixed",
"video_url": "https://example.com/video.mp4",
"duration_minutes": 45
}

Responses:

- 201 CREATED - Lesson was successfully created
- 400 BAD REQUEST - Lesson cannot be created (validation errors)
- 403 FORBIDDEN - Teacher role required

- POST /api/lessons/{lesson_id}/complete
  Request: body (JSON): { "time_spent_minutes": 30 }
  Responses:
  - 200 OK - Lesson marked as complete
  - 400 BAD REQUEST - Prerequisites not met
  - 403 FORBIDDEN - Student role required

- POST /api/quizzes
  Request: body (JSON): {
  "course_id": 1,
  "title": "Python Basics Quiz",
  "description": "Test your knowledge of Python basics",
  "total_points": 100,
  "passing_score": 70,
  "time_limit_minutes": 60,
  "max_attempts": 3
  }
  Responses:
  - 201 CREATED - Quiz created successfully
  - 400 BAD REQUEST - Invalid data
  - 403 FORBIDDEN - Teacher role required

- POST /api/quizzes/{quiz_id}/start
  Request: body (JSON): {}
  Responses:
  - 200 OK - Quiz started, returns questions and attempt ID
  - 400 BAD REQUEST - Max attempts reached or quiz in progress
  - 403 FORBIDDEN - Student role required

- POST /api/quizzes/attempt/{attempt_id}/submit
  Request: body (JSON): {}
  Responses:
  - 200 OK - Quiz submitted and graded
  - 400 BAD REQUEST - Invalid answers or time expired
  - 404 NOT FOUND - Attempt not found

- POST /api/assignments
  Request: body (JSON): {
  "course_id": 1,
  "title": "Python Programming Project",
  "description": "Create a simple calculator application",
  "due_date": "2024-02-15T23:59:59",
  "total_points": 100
  }
  Responses:
  - 201 CREATED - Assignment created successfully
  - 400 BAD REQUEST - Invalid data
  - 403 FORBIDDEN - Teacher role required

- POST /api/assignments/submissions/{submission_id}/grade
  Request: body (JSON): {
  "grade": 85.5,
  "feedback": "Good work! Consider improving error handling."
  }
  Responses:
  - 201 OK - Assignment graded successfully
  - 400 BAD REQUEST - Invalid grade
  - 403 FORBIDDEN - Teacher role required

- POST /api/student/certificates/request/{course_id}
  Request: body (JSON): {}
  Responses:
  - 201 CREATED - Certificate request submitted
  - 400 BAD REQUEST - Course not completed
  - 403 FORBIDDEN - Student role required

2. **GET API Calls**: Used to retrieve/read data from the server without modifying anything.
   - GET /api/auth/profile
     Request: body (JSON): {}
     Responses:
     - 200 OK - Profile data returned
     - 401 UNAUTHORIZED - Invalid or missing token

   - GET /api/courses/{course_id}
     Request: body (JSON): {}
     Responses:
     - 200 OK - Course details returned
     - 404 NOT FOUND - Course not found
     - 403 FORBIDDEN - Access denied

   - GET /api/courses/enrolled
     Request: body (JSON): {}
     Responses:
     - 200 OK – List of enrolled courses
     - 401 UNAUTHORIZED - Invalid or missing token

   - GET /api/lessons/course/{course_id}
     Request: body (JSON): {}
     Responses:
     - 200 OK - List of lessons with progress
     - 404 NOT FOUND - Course not found
     - 403 FORBIDDEN - Access denied

   - GET /api/quizzes/course/{course_id}
     Request: body (JSON): {}
     Responses:

- 200 OK - List of quizzes with attempt info
- 404 NOT FOUND - Course not found
- 403 FORBIDDEN - Access denied

- GET /api/quizzes/{quiz_id}
  Request: body (JSON): {}
  Responses:
  - 200 OK - Quiz details (questions for teachers, basic info for students)
  - 404 NOT FOUND - Quiz not found
  - 403 FORBIDDEN - Access denied

- GET /api/quizzes/attempt/{attempt_id}/results
  Request: body (JSON): {}
  Responses:
  - 200 OK - Detailed quiz results
  - 404 NOT FOUND - Quiz not found
  - 403 FORBIDDEN - Access denied

- GET /api/assignments/course/{course_id}
  Request: body (JSON): {}
  Responses:
  - 200 OK - List of assignments with submission status
  - 404 NOT FOUND - Course not found
  - 403 FORBIDDEN - Access denied

- GET /api/assignments/{assignment_id}
  Request: body (JSON): {}
  Responses:
  - 200 OK - Assignment details and submission info
  - 404 NOT FOUND - Assignment not found
  - 403 FORBIDDEN - Access denied

- GET /api/teacher/course/{course_id}/students
  Request: body (JSON): {}
  Responses:
  - 200 OK - Detailed student progress data
  - 404 NOT FOUND - Course not found

- 403 FORBIDDEN - Access denied

3. **PUT API Calls**: Used to update/modify existing resources on the server.
   - PUT /api/auth/profile

     Request: body (JSON): {

     "full_name": "Violeta Maria Hoza",

     "phone": "1234567891",

     "age": 26

     }

     Responses:
     - 200 OK - Profile updated successfully
     - 400 BAD REQUEST - Invalid data
     - 401 UNAUTHORIZED - Invalid token

   - PUT /api/courses/{course_id}

     Request:

     body (JSON): {

     "title": "Introduction to Python – Part 1",

     "description": "Learn Python programming from basics",

     "category": "Programming",

     "start_date": "2024-01-15",

     "end_date": "2024-03-15",

     "max_students": 50,

     "is_published": false

     }

     Responses:

     - 200 OK - Course updated successfully
     - 404 NOT FOUND - Course not found
     - 403 FORBIDDEN - Access denied

   - PUT /api/admin/users/{user_id}

     Request:

     body (JSON): {

     "full_name": "Updated Name",

     "email": "updated@example.com",

```
  "is_active": true
}
```

Responses:

- 200 OK - User updated successfully
- 404 NOT FOUND - User not found
- 403 FORBIDDEN – Admin role required