Back End Code Explaination.

This document describes all the 3 controller in details. It explain each of the code3 section.

StudentsController.cs

Overview

The StudentsController is a .NET Core Web API controller that manages operations related to student data within the application. It serves several endpoints to retrieve, view, and authenticate student data. Each endpoint is tailored to a specific operation such as listing all students, retrieving student details by ID, handling login, and managing logout.

Dependencies and Imports

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using API.Data;
using API.Models;
using API.Services;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Components;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Data.SqlClient;
using Microsoft.EntityFrameworkCore;
```

Attributes and Routing

```
[Route("api/[controller]")]: Sets up routing so that all
actions within this controller are accessible under api/
Students.
[ApiController]: Defines the controller as an API controller,
automatically handling HTTP responses and validation.
```

Constructor

```
public StudentsController(StudentManagementDbContext context)
{
    _context = context;
}
```

- **Purpose**: Initializes a new instance of the StudentsController class with a StudentManagementDbContextinstance, which is injected by .NET Core’s dependency injection container.
- **Parameters**:
    - StudentManagementDbContext context: Database context used to interact with the database.

Endpoints

1. ShowAllStudents - [HttpGet("ShowAllStudents")]
Description: Retrieves a list of all students.
Return Type: ActionResult<IEnumerable<StudentDTO>>

- **Logic**:
    - Queries the Users table for users where UserType is 2 (indicating a student).
    - Projects each student record into a StudentDTO object, which includes FullName, Email, EnrollmentDate, and UserId.
    - If no students are found, it returns a 404 NotFound response.
    - If students are found, it returns 200 OK with the list of StudentDTO.

Sample Response:

```
[
  {
    "fullName": "John Doe",
    "studentId": 1,
    "email": "john@example.com",
    "enrollmentDate": "2023-01-15"
  },
  {
    "fullName": "Jane Smith",
    "studentId": 2,
    "email": "jane@example.com",
    "enrollmentDate": "2023-02-20"
  }
]
```

2. ShowStudentByID - [HttpGet("ShowStudentByID/{id}")]
Description: Retrieves a student’s details by their unique UserId.
Parameters:

- int id: Unique identifier for the student.

Return Type: ActionResult<StudentDTO>

- **Logic**:
    - Searches the Users table for a student with a matching UserId and UserType of 2.
    - Projects the result into a StudentDTO.
    - If the student does not exist, returns 404 NotFound.
    - If found, returns 200 OK with the StudentDTO.

Sample Response:
```
{
  "fullName": "John Doe",
  "studentId": 1,
  "email": "john@example.com",
  "enrollmentDate": "2023-01-15"
}
```

3. Login - [HttpPost("Login")]
Description: Authenticates a student using their email and password.
Parameters:
- LoginDTO loginDTO: Object containing Email and Password.

Return Type: IActionResult
- **Logic**:
    - Validates that loginDTO is not null and has both Email and Password values.
    - Checks for a user with matching Email and Password in the Users table.
    - If credentials do not match, returns 401 Unauthorized with a message "Invalid email or password."
    - If authenticated, returns 200 OK with:
        - StatusCode: Explicitly set to 200.
        - FullName: Concatenation of FirstName and LastName.
        - UserType: The type of user (e.g., student).
        - UserId: Unique identifier for the user.

Sample Response:

```
{
  "statusCode": 200,
  "fullName": "John Doe",
  "userType": 2,
  "userId": 1
}
```

4. Logout - [HttpPost("Logout")]
Description: Handles the logout process. (Currently a placeholder).

Return Type: IActionResult

- **Logic**:
  - In a real-world scenario, this method might invalidate a token or end a session.
  - Returns 200 OK as a placeholder.

Data Models

- **StudentDTO**:
  - Represents a simplified view of student data that is returned from the API.
  - Properties include:
    - FullName: The full name of the student (combination of FirstName and LastName).
    - StudentId: Unique ID of the student.
    - Email: Student's email address.
    - EnrollmentDate: Date when the student was enrolled.
- **LoginDTO**:
  - Contains the login data required to authenticate a user.
  - Properties include:
    - Email: The email address of the user.
    - Password: The password of the user.

Error Handling

- Each endpoint checks for required parameters and invalid requests:
  - Missing or incorrect data in Login and ShowStudentByID methods result in 400 BadRequest or 404 NotFound responses.
  - Invalid login credentials return 401 Unauthorized.

EnrollmentsController.cs

Documentation

Overview

The EnrollmentsController is a .NET Core Web API controller responsible for handling operations related to student enrollments in courses. It provides endpoints for fetching all enrollments and allowing students to join a course with validation checks. This controller includes two main endpoints: one for retrieving all enrollments and another for processing course enrollment requests.

Dependencies and Imports

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using API.Data;
using API.Models;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
```

Attributes and Routing

`[Route("api/[controller]")]: Sets up routing so that all actions within this controller are accessible under api/Enrollments.`
`[ApiController]: Defines the controller as an API controller, automatically handling HTTP responses and validation.`

Constructor

```
public EnrollmentsController(StudentManagementDbContext
context)
{
    _context = context;
}
```

- **Purpose**: Initializes a new instance of the EnrollmentsController class with a StudentManagementDbContextinstance, injected by .NET Core's dependency injection container.
- **Parameters**:
  - StudentManagementDbContext context: Database context for interacting with the database.

Endpoints

1. GetAllEnrollments - [HttpGet("GetAllEnrollments")]

Description: Retrieves a list of all enrollments, including details about the student, course, enrollment date, and completion status.
Return Type: Task<IActionResult>

- **Logic**:

    - Uses Entity Framework Core’s Include method to join the Enrollments table with related Course and Student (User) entities.
    - Selects each enrollment with custom fields:
        - StudentName: Concatenates FirstName and LastName.
        - CourseName: Retrieves the course name.
        - CompletionStatus: Indicates whether the enrollment is complete.
        - EnrollmentDate: Displays only the date portion of the enrollment.
    - Returns 404 NotFound if no enrollments are found; otherwise, returns a structured 200 OK response.

Response Structure:

```
{
  "headers": ["StudentName", "CourseName", "CompletionStatus",
"EnrollmentDate"],
  "data": [
    {
      "StudentName": "John Doe",
      "CourseName": "Math 101",
      "CompletionStatus": "0",
      "EnrollmentDate": "2024-10-23"
    }
  ]
}
```

2. JoinCourse - [HttpPost("JoinCourse")]
Description: Allows a student to enroll in a course if they meet certain conditions.
Parameters:

- EnrollmentRequestDTO enrollmentRequest: DTO containing StudentName and CourseName.

Return Type: Task<IActionResult>

- **Logic**:
    1. **Validates Input**: Checks if StudentName and CourseName are provided; if missing, returns 400 BadRequest.
    2. **Finds Student**:
        - Searches for a student matching the provided StudentName.
        - Returns 404 NotFound if the student does not exist.
    3. **Enrollment Count Check**:
        - Counts active enrollments (where CompletionStatus is 0) for the student.
        - If the student already has 3 active enrollments, returns 400 BadRequest.
    4. **Finds Course**:
        - Searches for a course matching the provided CourseName.

        - Returns 404 NotFound if the course does not exist.
    5. **Checks for Duplicate Enrollment**:
        - Verifies if the student is already enrolled in the selected course.
        - If enrolled, returns 409 Conflict.
    6. **Creates New Enrollment**:
        - Creates a new Enrollment instance with the current date and a default CompletionStatus of 0.
        - Saves the new enrollment to the database.
        - Returns 201 Created with details about the enrollment.

Sample Response:

```
{
  "message": "Student 'John Doe' has been enrolled in 'Math 101'.",
  "enrollmentId": 123,
  "studentName": "John Doe",
  "courseName": "Math 101",
  "completionStatus": "0",
  "enrollmentDate": "2024-10-23"
}
```

Data Models and DTOs

- **EnrollmentRequestDTO**:
    - DTO for capturing enrollment requests.
    - **Properties**:
        - StudentName: Name of the student enrolling in the course.
        - CourseName: Name of the course the student wishes to enroll in.

Error Handling

- Each endpoint handles various scenarios, ensuring proper HTTP responses:
    - Missing or invalid data returns 400 BadRequest.
    - Non-existent students or courses return 404 NotFound.
    - Duplicate enrollments return 409 Conflict.
    - Exceeding active enrollments returns 400 BadRequest.

CourseController.cs

This guide explains each code block in detail, its purpose, functionality, and the communication between components in this CourseController class.

Overview

- **File Purpose**: CourseController.cs is an ASP.NET Core API controller that manages course-related data in the StudentManagementDbContext database. It provides endpoints for retrieving, creating, and managing course data.
- **Primary Technologies**:
  - **Entity Framework Core**: for database access.
  - **DTO (Data Transfer Objects)**: for transferring data between client and server in a controlled format.
  - **Dependency Injection**: used for injecting StudentManagementDbContext to manage database operations.
  - **Microsoft.AspNetCore.Mvc**: for setting up API controller routes and attributes.

Code Blocks Explained

`Namespace and Imports`

```
using Microsoft.AspNetCore.Mvc;
using API.Models;
using API.Services;
using System.Linq;
using System.Collections.Generic;
```

  - **Purpose**: These using statements bring required namespaces into the file.
  - **Functionality**:
    - Microsoft.AspNetCore.Mvc provides attributes and base classes for creating API controllers.
    - API.Models and API.Services are local namespaces containing the CourseDTO model and database context (StudentManagementDbContext).
    - System.Linq enables LINQ queries on collections.
    - System.Collections.Generic allows for defining collection types like IEnumerable<T>.

Controller Declaration and Route Setup

[Route("api/[controller]")]

```
[ApiController]
```

```
public class CourseController : ControllerBase
```

- **Purpose**: Defines CourseController as an API controller with a route pattern.
- **Functionality**:
  - [Route("api/[controller]")]: Sets the base route to api/Course (controller's name is used by convention).
  - [ApiController]: Enables automatic model validation and formatting of responses.

Dependency Injection for Database Context

```
private readonly StudentManagementDbContext _context;

public CourseController(StudentManagementDbContext context)
{
    _context = context;
}
```

- **Purpose**: Injects StudentManagementDbContext to allow interaction with the database.
- **Functionality**:
  - The StudentManagementDbContext instance is injected through the constructor, making it available throughout the controller.
  - Enables unit testing and dependency injection, enhancing modularity and testability.

ShowAllCourses Endpoint

```
[HttpGet("ShowAllCourses")]
public ActionResult<IEnumerable<CourseDTO>> ShowAllCourses()
{
    var courses = _context.Courses
        .Select(c => new CourseDTO(
            c.CourseId,
            c.CourseName,
            c.Description,
            c.MaxSeats,
            c.CurrentSeats,
            c.StartDate,
            c.EndDate))
        .ToList();

    if (courses == null || !courses.Any())
    {
        return NotFound("No courses found.");
    }

    return Ok(courses);
}
```

- **Purpose**: Retrieves a list of all courses from the database.
- **Functionality**:
  - Queries the Courses table in _context and projects each Course entity to a CourseDTO.
  - Uses LINQ's Select method to map database entities to CourseDTO, which reduces the data size and ensures only essential information is returned to the client.
  - If no courses are found, it returns a 404 NotFound status with an error message.
  - Otherwise, it returns the list of courses with a 200 OK status.

ShowCourseByID Endpoint

```
[HttpGet("ShowCourseByID/{id}")]
public ActionResult<CourseDTO> ShowCourseByID(int id)
{
    var course = _context.Courses
        .Where(c => c.CourseId == id)
        .Select(c => new CourseDTO(
            c.CourseId,
            c.CourseName,
            c.Description,
            c.MaxSeats,
            c.CurrentSeats,
            c.StartDate,
            c.EndDate))
        .FirstOrDefault();

    if (course == null)
    {
        return NotFound($"Course with ID {id} not found.");
    }

    return Ok(course);
}
```

- **Purpose**: Retrieves a specific course by its ID.
- **Functionality**:
    - Uses LINQ to filter Courses based on the given id parameter, selecting only the relevant CourseDTOproperties.
    - FirstOrDefault() returns the first course matching the criteria or null if none is found.
    - If course is null, it returns 404 NotFound with a descriptive message.
    - If found, it returns the course details as a CourseDTO object with a 200 OK status.

CreateCourse Endpoint

```
[HttpPost("CreateCourse")]
public ActionResult<CourseDTO> CreateCourse([FromBody]
CourseDTO courseDTO)
{
    if (courseDTO == null)
    {
        return BadRequest("Invalid course data.");
    }

    var course = new Course
    {
        CourseId = courseDTO.CourseId,
        CourseName = courseDTO.CourseName,
        Description = courseDTO.Description,
        MaxSeats = courseDTO.MaxSeats,
        CurrentSeats = courseDTO.CurrentSeats,
        StartDate = courseDTO.StartDate,
        EndDate = courseDTO.EndDate
    };

    _context.Courses.Add(course);
    _context.SaveChanges();

    return Ok(courseDTO);
}
```

- **Purpose**: Creates a new course entry in the database.
- **Functionality**:
    - [FromBody] CourseDTO courseDTO binds JSON data from the request body to a CourseDTO object.

        - Checks if courseDTO is null and, if so, returns 400 BadRequest with an error message.
        - Maps CourseDTO properties to a new Course entity to match the database schema.
        - Adds the new Course entity to the _context.Courses table and calls SaveChanges() to persist the changes.
        - Returns the created CourseDTO back to the client with a 200 OK status.

How Code Blocks Communicate

1. **Controller and Database Context**:
    - The CourseController accesses the database via StudentManagementDbContext, which serves as the gateway to the Courses table.
    - The controller doesn’t directly manipulate the database schema; instead, it relies on the context and Entity Framework’s ORM capabilities.
2. **Data Flow**:
    - For GET endpoints (ShowAllCourses, ShowCourseByID):
        - The controller queries the _context.Courses DbSet and maps each Course entity to a CourseDTO.
        - The resulting CourseDTO objects are then returned as HTTP responses to avoid exposing the entire Course model.
    - For POST endpoint (CreateCourse):
        - The client sends a JSON request body mapped to a CourseDTO.
        - The controller maps this DTO to a Course entity, then adds and saves it in the database.
3. **DTO Mapping**:
    - The use of CourseDTO abstracts the internal representation of courses, making responses lightweight and omitting unnecessary database properties.
    - In the CreateCourse endpoint, the controller converts CourseDTO back to the Course entity format, ensuring compatibility with the database schema.
4. **Error Handling**:
    - Each endpoint includes checks to handle potential issues, such as missing data (e.g., NotFound, BadRequest).
    - Errors return descriptive HTTP responses, making it easier for the client to understand and handle issues.

Summary

The CourseController in this document provides essential CRUD operations for course data in the database. The use of DTOs, dependency injection, and clear endpoint design makes the controller modular, testable, and secure. Each method serves a distinct purpose, working cohesively to provide a RESTful API interface for managing course data. This guide should help developers understand the functionality and flow within the CourseController.