Sprint 2 - DoConnect Capstone Documentation

1. Introduction

Sprint 2 focused on implementing core functionality to enable end-to-end integration of the frontend and backend of the DoConnect platform. This included building Angular components, backend APIs, authentication mechanisms, and connecting both layers through RESTful endpoints.

2. Sprint 2 Objectives

Frontend Setup (Angular)

- Initialized Angular app with separate modules for user and admin.
- Implemented routing for Login, Register, Ask Question, Answer Question, and Admin Approval pages.

Backend Implementation (ASP.NET Core MVC)

- Developed CRUD operations for Users (Register, Login, Logout).
- Implemented user authentication using JWT Tokens.
- Configured **ApplicationDbContext** with Entity Framework Core and SQL Server.

API for Questions & Answers

- Created Web API endpoints for CRUD operations on Questions and Answers.
- Implemented image upload functionality (for both questions and answers) using **IFormFile** and stored paths in the database.

Admin and User Pages

- Built Angular components for User (Ask Question, Answer Question) and Admin (Approve/Reject Questions, Approve/Reject Answers).
- Connected Angular frontend to backend Web API using Axios for HTTP requests.

3. Deliverables

3.1 Angular Components

User Components:

- Login, Register,UserDashboard, AskQuestion, AnswerQuestion, QuestionList

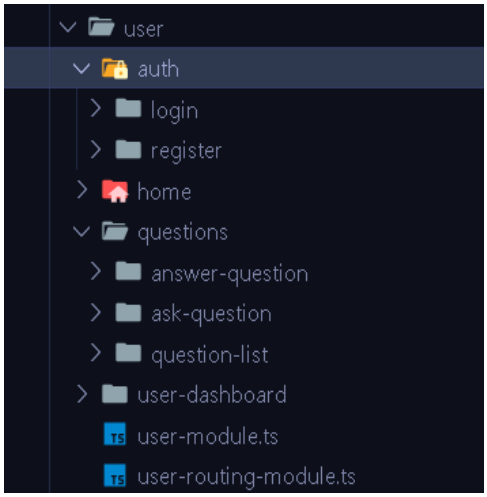

Fig: User Components

Admin Components:

- ApproveQuestions, ApproveAnswers, ManageContent, AdminDashboard

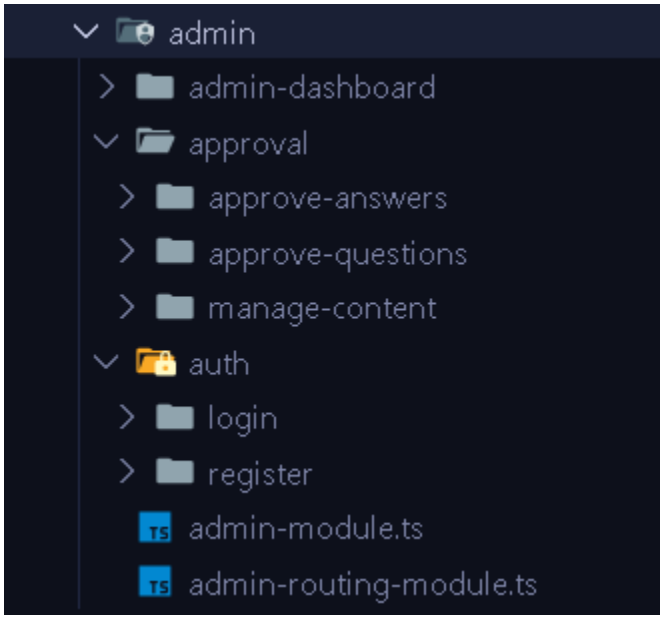

Fig: User Components

3.2 Backend Web API Endpoints

Authentication Endpoints

- POST /api/Auth/register – Register User/Admin
- POST /api/Auth/login – Login User/Admin (returns JWT token)
- POST /api/Auth/logout – Logout (invalidate session/token)

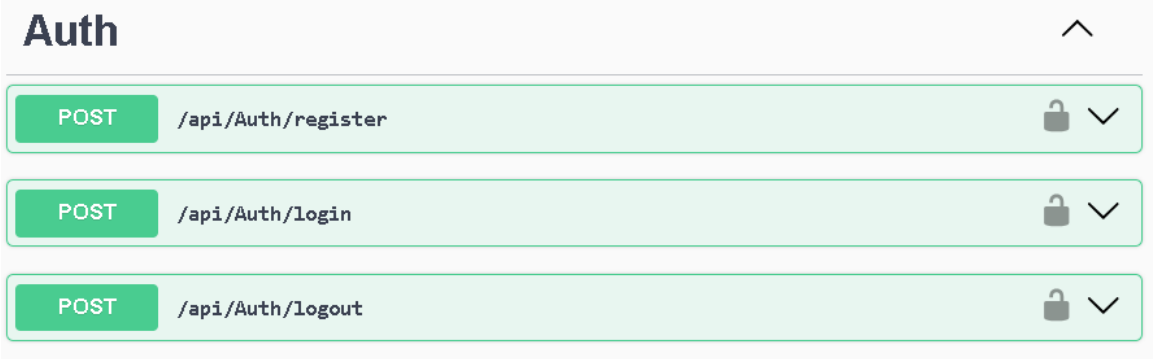

Fig: Authentication Endpoint

Question Endpoints

User

- GET /api/QuestionApi – Get all questions
- GET /api/QuestionApi/{id} – Get question by ID
- POST /api/QuestionApi – Create a new question
- POST /api/QuestionApi/with-image – Create a new question with image

Admin

- PUT /api/QuestionApi/{id} – Update question
- PUT /api/QuestionApi/{id}/approve – Approve a question
- PUT /api/QuestionApi/{id}/reject – Reject a question
- GET /api/QuestionApi/all-including-deleted – Get all questions including deleted
- GET /api/QuestionApi/search – Search questions
- DELETE /api/QuestionApi/{id} – Soft delete question (owner/admin)

Fig: Question Endpoints

Answer Endpoints

User

- POST /api/AnswerApi – Create answer
- GET /api/AnswerApi/question/{questionId} – Get answers for a question
- GET /api/AnswerApi/{id} – Get answer by ID
- PUT /api/AnswerApi/{id}/approve – Approve answer (Admin)
- PUT /api/AnswerApi/{id}/reject – Reject answer (Admin)

Admin

- PUT /api/AnswerApi/{id}/approve – Approve an answer
- PUT /api/AnswerApi/{id}/reject – Reject an answer
- GET /api/AnswerApi/answers/all-including-deleted – Get all answers including deleted
- DELETE /api/AnswerApi/{id} – Delete (soft delete)

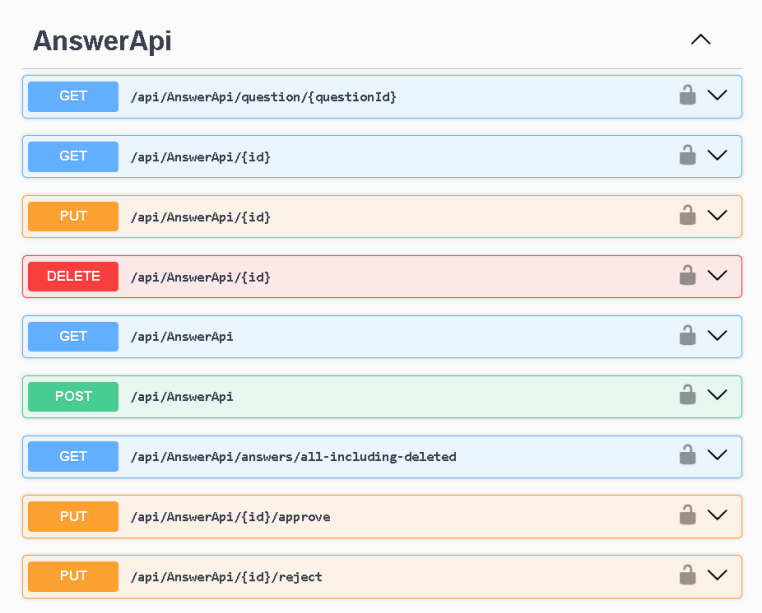

Fig: Answer Endpoints

Image Endpoints

- POST /api/ImageApi/upload/question/{questionId} – Upload image for a question
- POST /api/ImageApi/upload/answer/{answerId} – Upload image for an answer
- GET /api/ImageApi/question/{questionId} – Get images for a question
- GET /api/ImageApi/answer/{answerId} – Get images for an answer

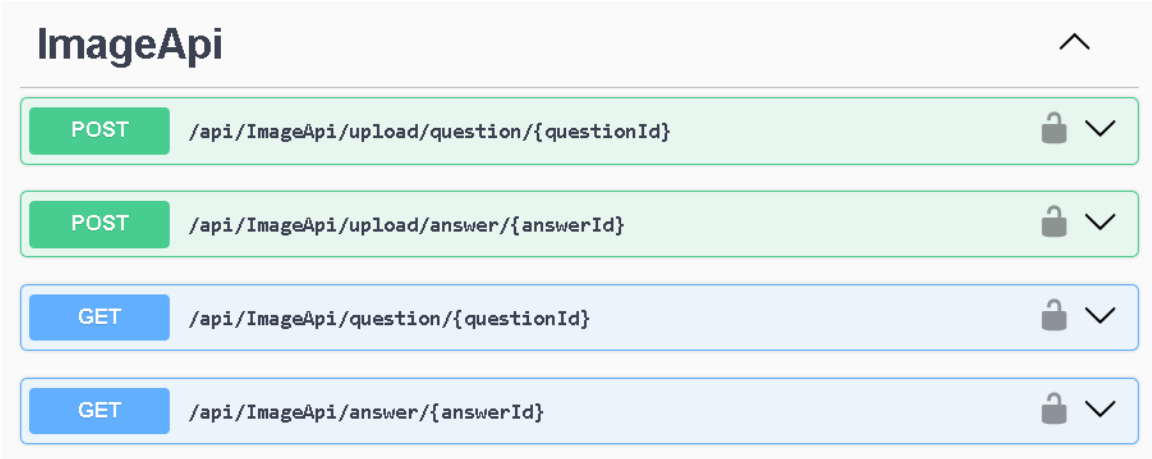

Fig: Image Endpoints

3.3 Authentication with JWT

- Implemented token generation upon login

Fig: Token generated while Login as a User

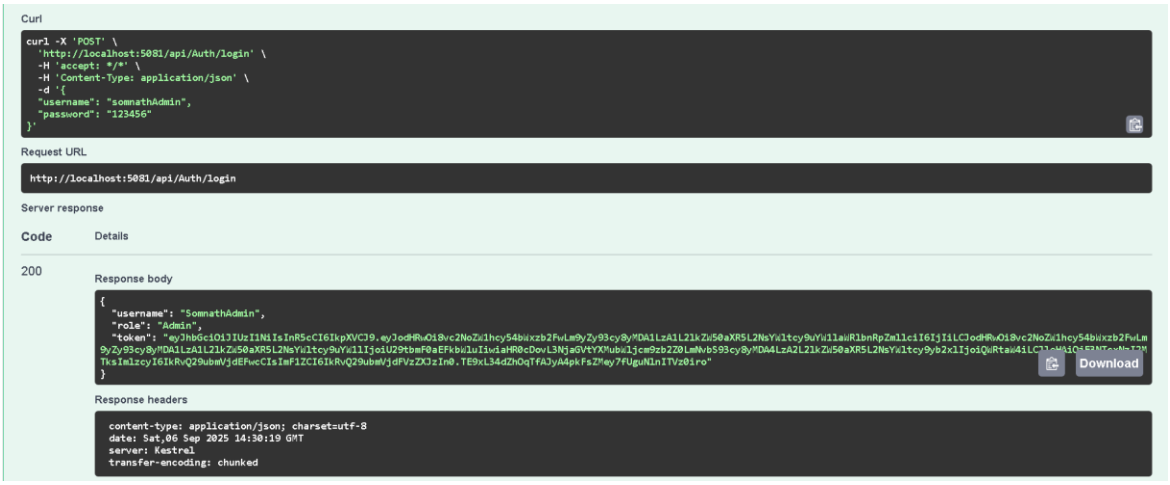

Fig: Token generated while Login as an Admin

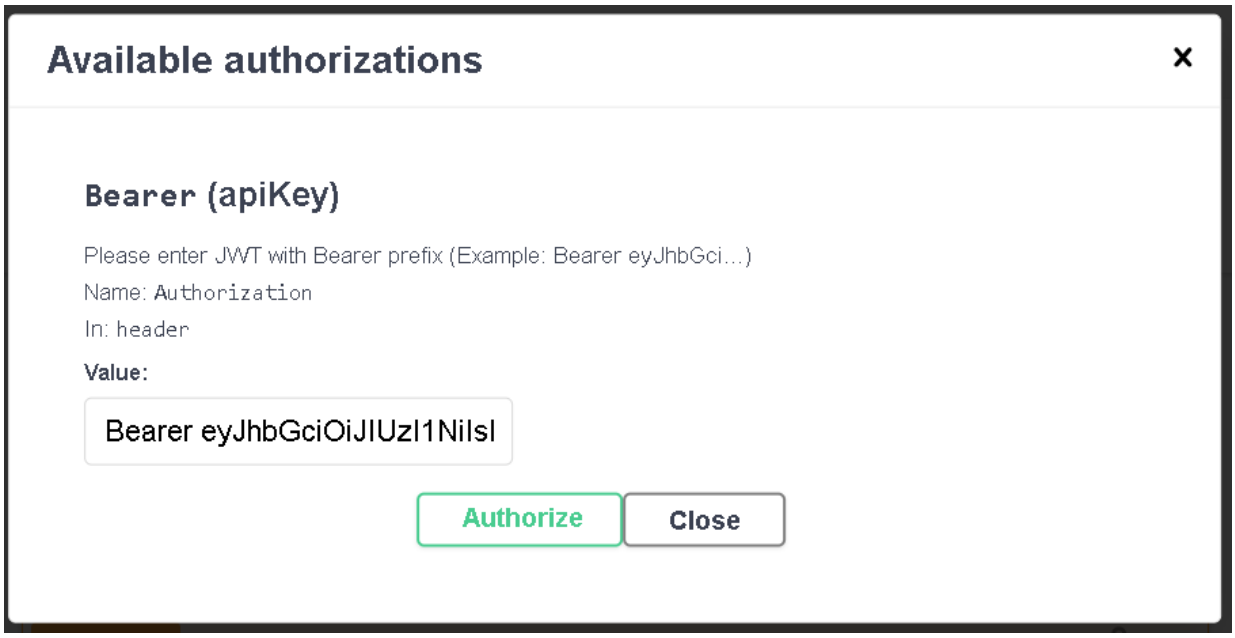

Fig: Authorization as User/Admin

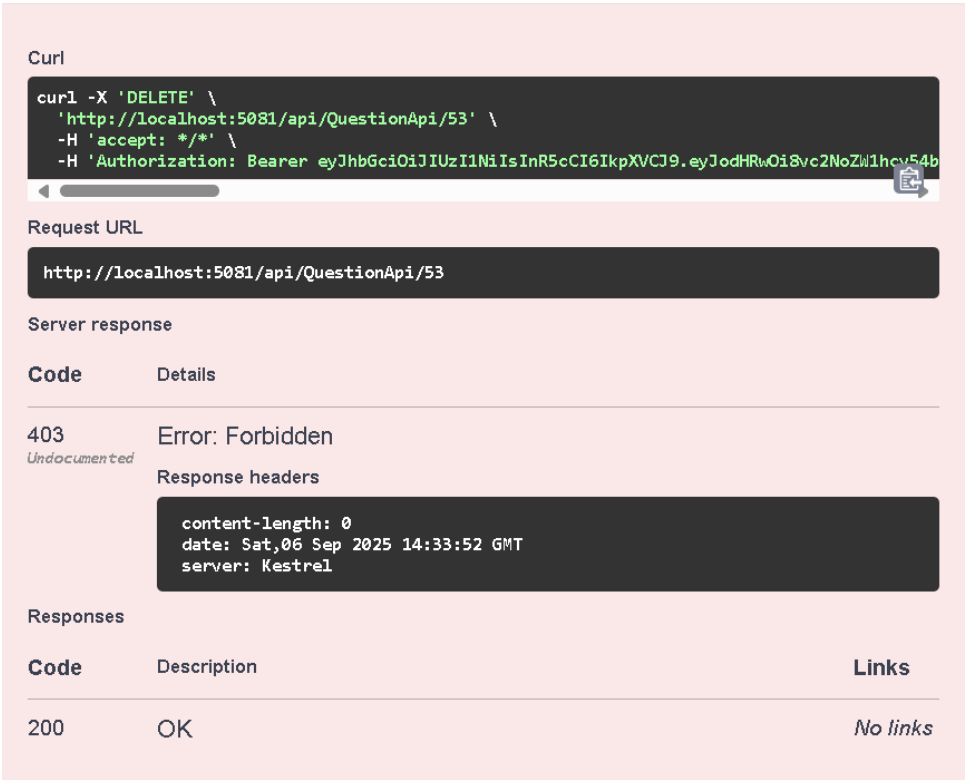

Fig: Deletion unsuccessful because Authorize happens as User

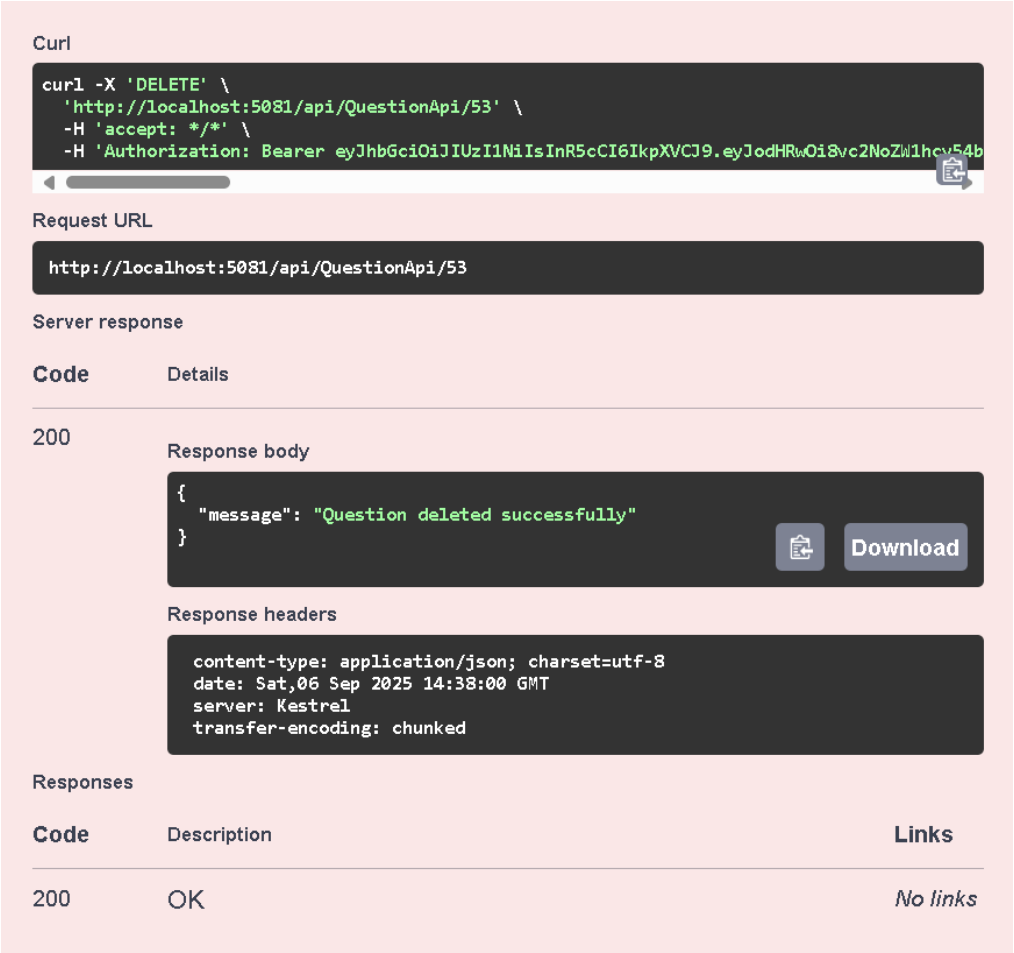

Fig: Deletion successful because Authorize happens as Admin

- **Secured protected endpoints with [Authorize] attribute**

```
[Authorize]
[HttpPost]
0 references
public async Task<IActionResult> CreateQuestion([FromBody] CreateQuestionDto dto)
{
    var userIdClaim = User.FindFirst(ClaimTypes.NameIdentifier)?.Value;
    if (userIdClaim == null) return Unauthorized();

    int userId = int.Parse(userIdClaim);

    var question = new Question
    {
        UserId = userId,
        QuestionTitle = dto.QuestionTitle,
        QuestionText = dto.QuestionText,
        Status = "Pending",
        CreatedAt = DateTime.UtcNow,
    };

    _context.Questions.Add(question);
    await _context.SaveChangesAsync();

    var result = new QuestionDto
    {
        QuestionId = question.QuestionId,
        QuestionTitle = question.QuestionTitle,
        QuestionText = question.QuestionText,
        Status = question.Status,
        CreatedAt = question.CreatedAt,
        Username = (await _context.Users.FindAsync(userId))?.Username ?? "Unknown",
        Answers = new List<AnswerDto>()
    };


    return CreatedAtAction(nameof(GetQuestion), new { id = question.QuestionId }, result);
}
```

Fig: Authorize attribute implemented for secure endpoint

- Added role-based access for Admin-specific actions (approve/reject, delete)

```
[Authorize(Roles = "Admin")]
[HttpPut("{id}/approve")]
0 references
public async Task<IActionResult> ApproveQuestion(int id)
{
    var question = await _context.Questions.FindAsync(id);
    if (question == null) return NotFound();

    question.Status = "Approved";
    await _context.SaveChangesAsync();

    return Ok(new { message = "Question approved successfully" });
}

[Authorize(Roles = "Admin")]
[HttpPut("{id}/reject")]
0 references
public async Task<IActionResult> RejectQuestion(int id)
{
    var question = await _context.Questions.FindAsync(id);
    if (question == null) return NotFound();

    question.Status = "Rejected";
    await _context.SaveChangesAsync();

    return Ok(new { message = "Question rejected successfully" });
}
```

Fig: Role-based actions for admin specific actions

4. Use Cases in Sprint 2

User Workflow

- Register/Login, Post questions with/without images, Answer approved questions, Upload images with answers.

Admin Workflow

- Approve/Reject questions and answers.
- View/manage content via dashboard.

5. Conclusion

Sprint 2 successfully integrated the front-end Angular app with the backend Web API. Authentication was implemented with JWT, and core CRUD functionality for questions, answers, and images was completed. Admin approval workflows were introduced, setting the stage for advanced features in Sprint 3 such as notifications and content management.