Sprint 1 - DoConnect Capstone Documentation

1. Introduction

The DoConnect platform is a Q&A web application that allows users to post questions, provide answers, and attach images. This document summarizes the Sprint 1 deliverables, including the database design, MVC setup, and placeholder pages.

2. Objectives

- Set up ASP.NET Core MVC Project
- Implement Database Schema using EF Core + SQL Server
- Generate Entity Relationship Diagram (ERD)
- Create initial Controllers & Views (placeholders)
- Integrate Swagger for API documentation
- Prepare foundation for authentication in later sprints

3. Deliverables

Database Schema :

The database schema includes 4 core tables:
- Users (stores user credentials & roles)
- Questions (contains questions posted by users)
- Answers (answers linked to questions and users)
- Images (optional images linked to questions/answers)

See attached SQL file: **DoConnectSchema.sql**

```
IF OBJECT_ID(N'[__EFMigrationsHistory]') IS NULL

BEGIN

  CREATE TABLE [__EFMigrationsHistory] (

    [MigrationId] nvarchar(150) NOT NULL,

    [ProductVersion] nvarchar(32) NOT NULL,

    CONSTRAINT [PK___EFMigrationsHistory] PRIMARY KEY ([MigrationId])

  );
```

```
END;

GO


BEGIN TRANSACTION;

CREATE TABLE [Users] (

   [UserId] int NOT NULL IDENTITY,

   [Username] nvarchar(100) NOT NULL,

   [Password] nvarchar(max) NOT NULL,

   [Role] nvarchar(max) NOT NULL,

   CONSTRAINT [PK_Users] PRIMARY KEY ([UserId])

);


CREATE TABLE [Questions] (

   [QuestionId] int NOT NULL IDENTITY,

   [UserId] int NOT NULL,

   [QuestionTitle] nvarchar(255) NOT NULL,

   [QuestionText] nvarchar(max) NOT NULL,

   [Status] nvarchar(max) NOT NULL,

   [CreatedAt] datetime2 NOT NULL,

   CONSTRAINT [PK_Questions] PRIMARY KEY ([QuestionId]),

   CONSTRAINT [FK_Questions_Users_UserId] FOREIGN KEY ([UserId]) REFERENCES [Users] ([UserId]) ON
DELETE CASCADE

);


CREATE TABLE [Answers] (

   [AnswerId] int NOT NULL IDENTITY,

   [QuestionId] int NOT NULL,
   [UserId] int NOT NULL,

   [AnswerText] nvarchar(max) NOT NULL,

   [Status] nvarchar(max) NOT NULL,

   [CreatedAt] datetime2 NOT NULL,

   CONSTRAINT [PK_Answers] PRIMARY KEY ([AnswerId]),

   CONSTRAINT [FK_Answers_Questions_QuestionId] FOREIGN KEY ([QuestionId]) REFERENCES [Questions]
([QuestionId]) ON DELETE CASCADE,

   CONSTRAINT [FK_Answers_Users_UserId] FOREIGN KEY ([UserId]) REFERENCES [Users] ([UserId]) ON DELETE
NO ACTION

);


CREATE TABLE [Images] (

   [ImageId] int NOT NULL IDENTITY,

   [ImagePath] nvarchar(max) NOT NULL,

   [QuestionId] int NULL,

   [AnswerId] int NULL,

   [UploadedAt] datetime2 NOT NULL,

   CONSTRAINT [PK_Images] PRIMARY KEY ([ImageId]),

   CONSTRAINT [FK_Images_Answers_AnswerId] FOREIGN KEY ([AnswerId]) REFERENCES [Answers]
([AnswerId]),

   CONSTRAINT [FK_Images_Questions_QuestionId] FOREIGN KEY ([QuestionId]) REFERENCES [Questions]
([QuestionId])

);


CREATE INDEX [IX_Answers_QuestionId] ON [Answers] ([QuestionId]);


CREATE INDEX [IX_Answers_UserId] ON [Answers] ([UserId]);
CREATE INDEX [IX_Images_AnswerId] ON [Images] ([AnswerId]);


CREATE INDEX [IX_Images_QuestionId] ON [Images] ([QuestionId]);


CREATE INDEX [IX_Questions_UserId] ON [Questions] ([UserId]);


INSERT INTO [__EFMigrationsHistory] ([MigrationId], [ProductVersion])
VALUES (N'20250902062142_Initial', N'9.0.8');


COMMIT;
GO
```

Entity Relationship Diagram (ERD):

The ERD describes the relationship between Users, Questions, Answers, and Images. One-to-Many between Users and Questions, Questions and Answers

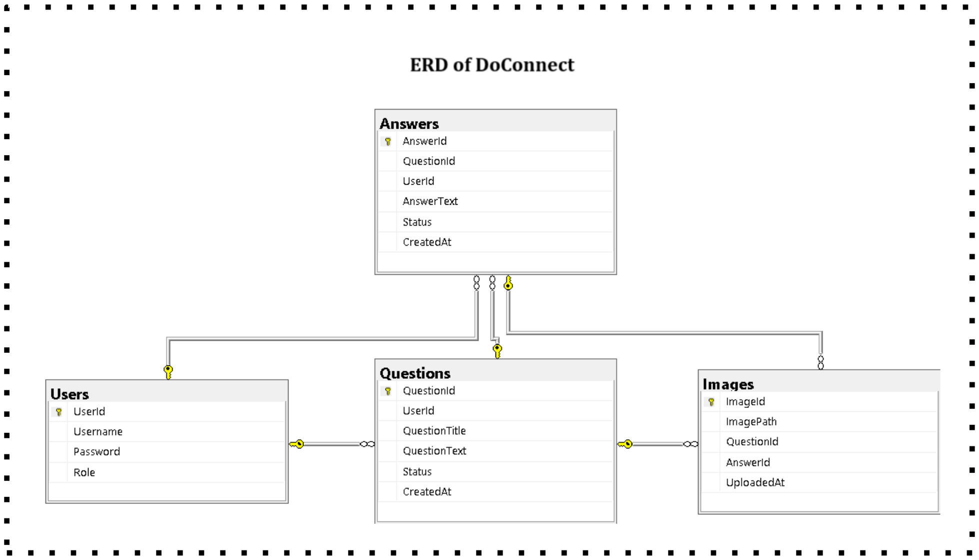


Fig - Entity Relationship Diagram of DoConnect

Static view in MVC for Question, Answer, Authentication :

Controllers :

- UserController

- QuestionController

- AnswerController

- ImageController

Views (Placeholders) :

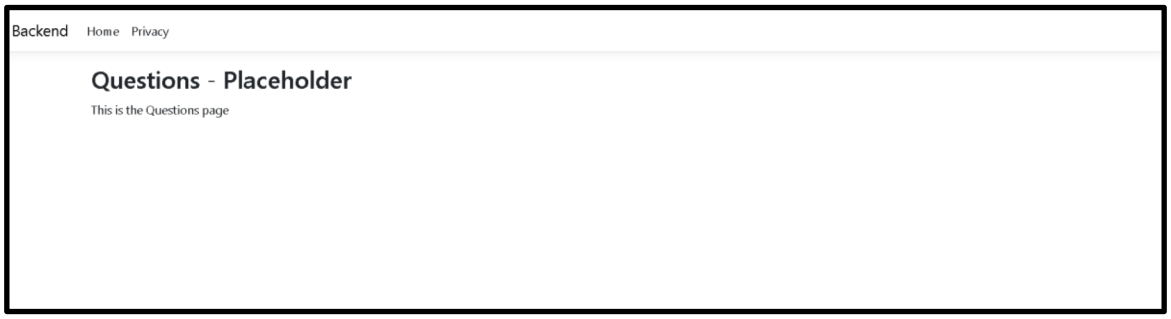
Each controller has an Index.cshtml file with a placeholder message:

For QuestionController :

```
@{

  ViewData["Title"] = "Questions";

}



<h2>Questions - Placeholder</h2>
```

```
<p>This is the Questions page</p>
```

Output (Only Placeholder message) :



Questions - Placeholder

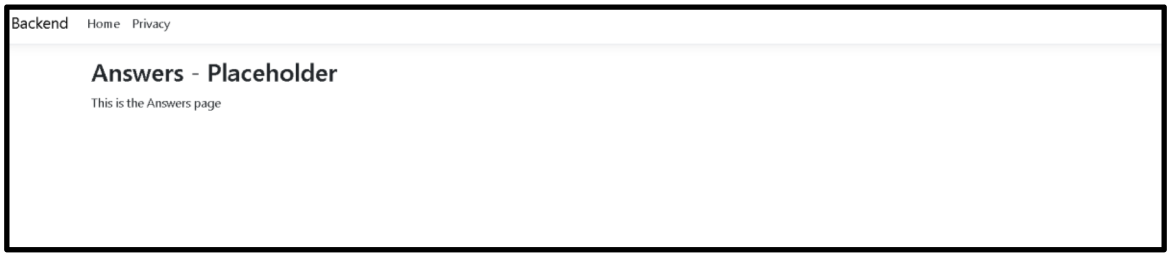
This is the Questions page

Fig - Placeholder image of QuestionController

For AnswerController :

```
@{
   ViewData["Title"] = "Answers";
}


<h2>Answers - Placeholder</h2>
<p>This is the Answers page</p>
```

Output (Only Placeholder message) :



Answers - Placeholder

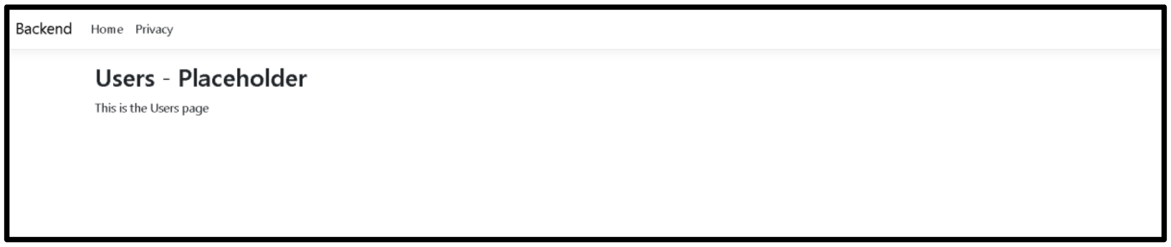
This is the Answers page

Fig - Placeholder image of AnswerController

For UserController :

```
@{
  ViewData["Title"] = "Users";
}


<h2>Users - Placeholder</h2>

<p>This is the Users page</p>
```

Output (Only Placeholder message) :



Backend Home Privacy

Users - Placeholder

This is the Users page

Fig - Placeholder image of UserController

6. Use Cases

- User Management: Database schema for users with role support
- Question Management: Create, Read, Update, Delete (CRUD) – placeholder views only
- Answer Management: Database schema for answers linked to users and questions
- Image Management: Image records linked to questions/answers

8. Conclusion

Sprint 1 successfully established the foundation of the DoConnect application, including database schema, ERD, basic MVC setup, and Swagger integration. Future sprints will build on this foundation to add authentication and frontend integration.

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. 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.

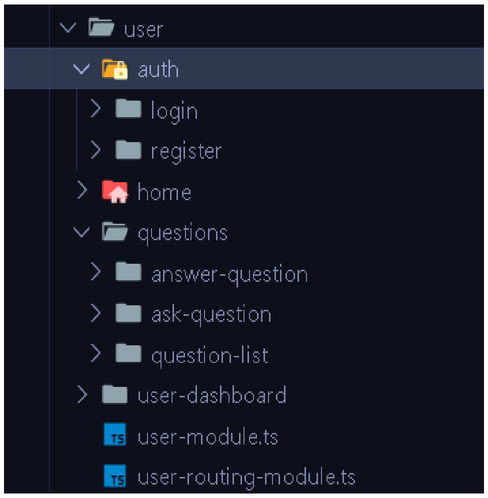
3. Deliverables

Angular Components for User and Admin

Angular components were built to provide a clear separation of functionality between users and administrators. User components included login, register, ask question, answer question, and question list pages, while admin components covered approval of questions, approval of answers, and content management. This ensured smooth navigation and role-specific access within the DoConnect platform.

User Components:

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

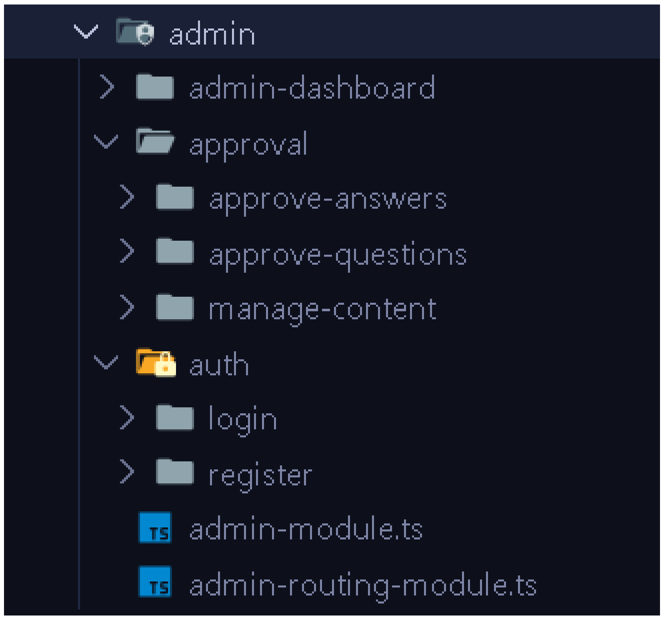


```
user
  auth
    login
    register
  home
  questions
    answer-question
    ask-question
    question-list
  user-dashboard
  user-module.ts
  user-routing-module.ts
```

Fig: User Components

Admin Components:

- ApproveQuestions, ApproveAnswers, ManageContent, AdminDashboard



```
admin
  admin-dashboard
  approval
    approve-answers
    approve-questions
    manage-content
  auth
    login
    register
  admin-module.ts
  admin-routing-module.ts
```

Fig: User Components

Backend Web API Endpoints for Question, Answer, User –

ASP.NET Core Web APIs were implemented to handle CRUD operations for questions and answers, as well as image uploads. Endpoints were created for posting, retrieving, approving, and rejecting content. Image handling was supported through IFormFile to allow users to attach images with their questions and answers, with paths stored in the database for retrieval.

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)

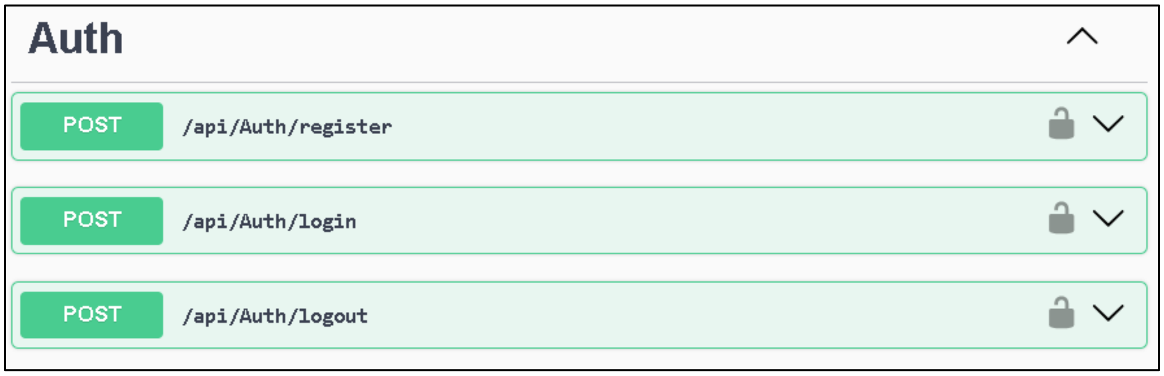


Fig: Authentication Endpoint

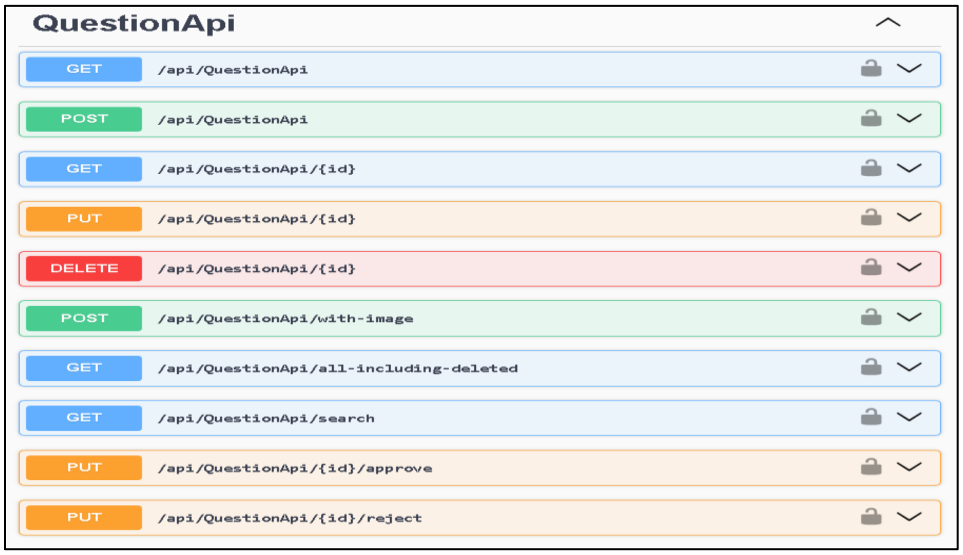
Question Endpoints

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)



QuestionApi

```
GET     /api/QuestionApi
POST    /api/QuestionApi
GET     /api/QuestionApi/{id}
PUT     /api/QuestionApi/{id}
DELETE  /api/QuestionApi/{id}
POST    /api/QuestionApi/with-image
GET     /api/QuestionApi/all-including-deleted
GET     /api/QuestionApi/search
PUT     /api/QuestionApi/{id}/approve
PUT     /api/QuestionApi/{id}/reject
```

Fig: Question Endpoints

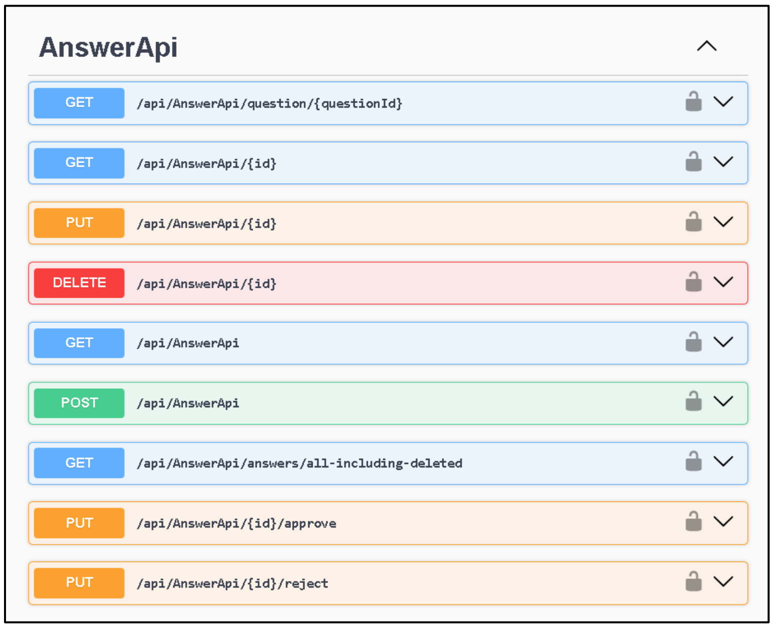
Answer Endpoints-

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)


AnswerApi

```
GET     /api/AnswerApi/question/{questionId}
GET     /api/AnswerApi/{id}
PUT     /api/AnswerApi/{id}
DELETE  /api/AnswerApi/{id}
GET     /api/AnswerApi
POST    /api/AnswerApi
GET     /api/AnswerApi/answers/all-including-deleted
PUT     /api/AnswerApi/{id}/approve
PUT     /api/AnswerApi/{id}/reject
```

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

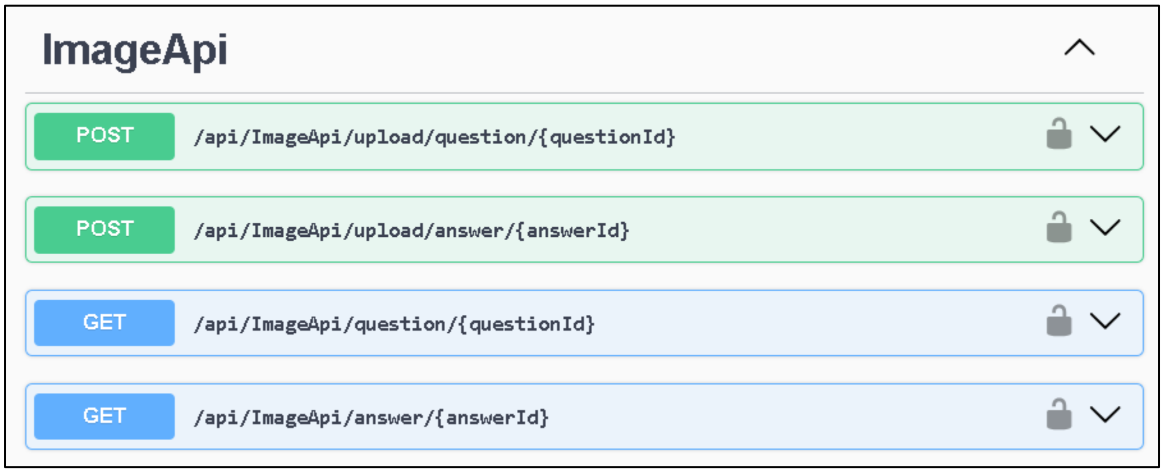
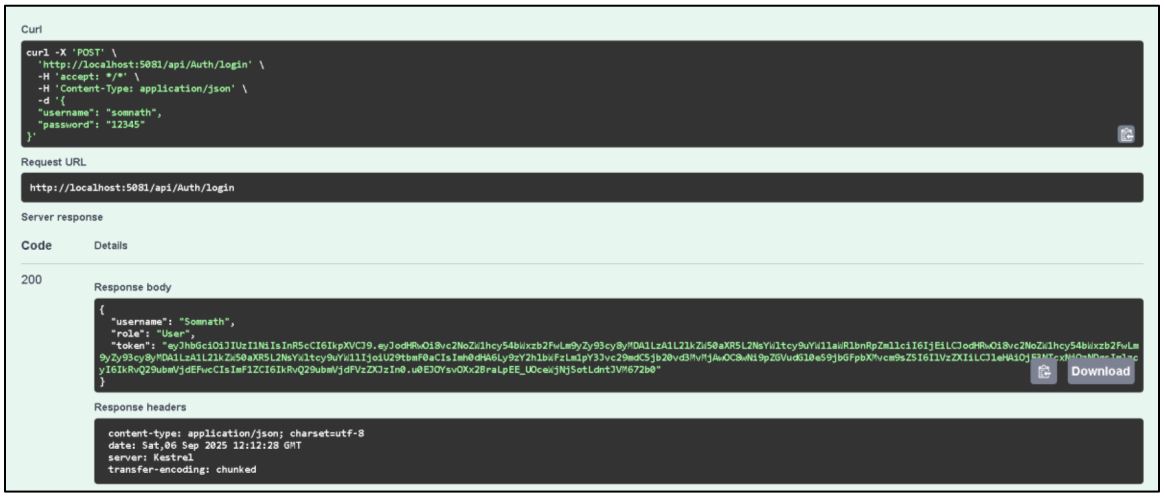

ImageApi
POST
/api/ImageApi/upload/question/{questionId}
POST
/api/ImageApi/upload/answer/{answerId}
GET
/api/ImageApi/question/{questionId}
GET
/api/ImageApi/answer/{answerId}

Fig: Image Endpoints

Working authentication with Admin and User-

JWT-based authentication was introduced to secure access across the application. Users and admins could log in to receive a token, which was then required for accessing protected endpoints. Role-based authorization ensured that only admins could perform actions like approving, rejecting, or deleting content, while users were restricted to posting and viewing approved content.

Implemented token generation upon login-



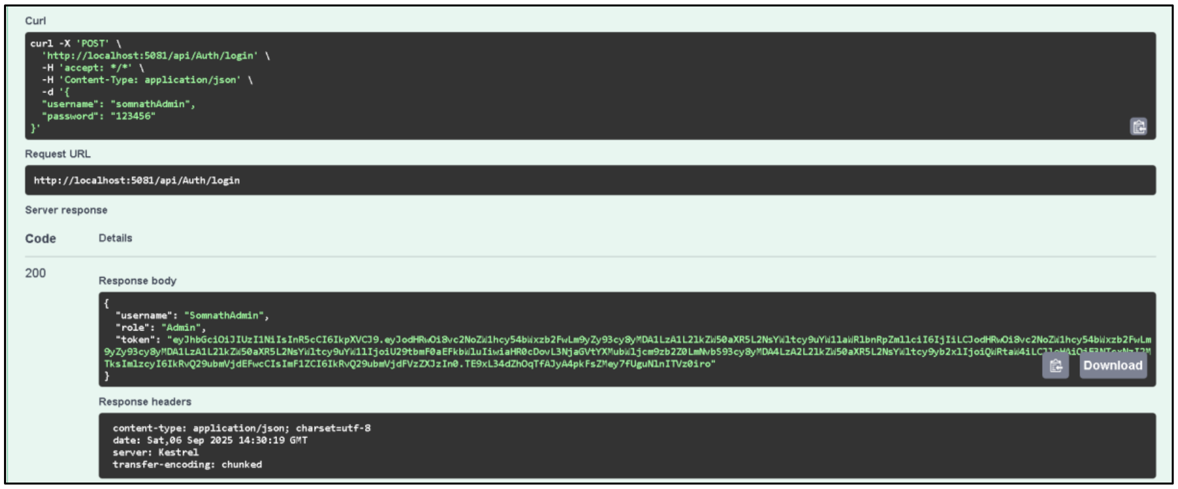
```
curl -X 'POST' \
  'http://localhost:5081/api/Auth/login' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
  "username": "somnath",
  "password": "12345"
}'
```

```
http://localhost:5081/api/Auth/login
```

```
{
  "username": "Somnath",
  "role": "User",
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJodHRwOi8vc2NoZW1hcy54bWxzb2FwLm9yZy93cy8yMDA1LzA1L2lkZW50aXR5L2NsYWltcy9uYW1laWRlbnRpZmllciI6IjEiLCJodHRwOi8vc2NoZW1hcy54bWxzb2FwLm
9yZy93cy8yMDA1LzA1L2lkZW50aXR5L2NsYWltcy9uYW1lIjoiU29tbmF0aCIsImh0dHA6Ly9zY2hlbWFzLm1pY3Jvc29mdC5jb20vd3MvMjAwOC8wNi9pZGVudGl0eS9jbGFpbXMvcm9sZSI6IlVzZXIiLCJleHAiOj[illegible]
yI6IkRvQ29ubmVjdEFwcCIsImF1ZCI6IkRvQ29ubmVjdFVzZXJzIn0.u0EJOYsvOXx2BraLpEE_UOceWjNjSotLdntJVM672b0"
}
```

```
content-type: application/json; charset=utf-8
date: Sat,06 Sep 2025 12:12:28 GMT
server: Kestrel
transfer-encoding: chunked
```

Fig: Token generated while Login as a User



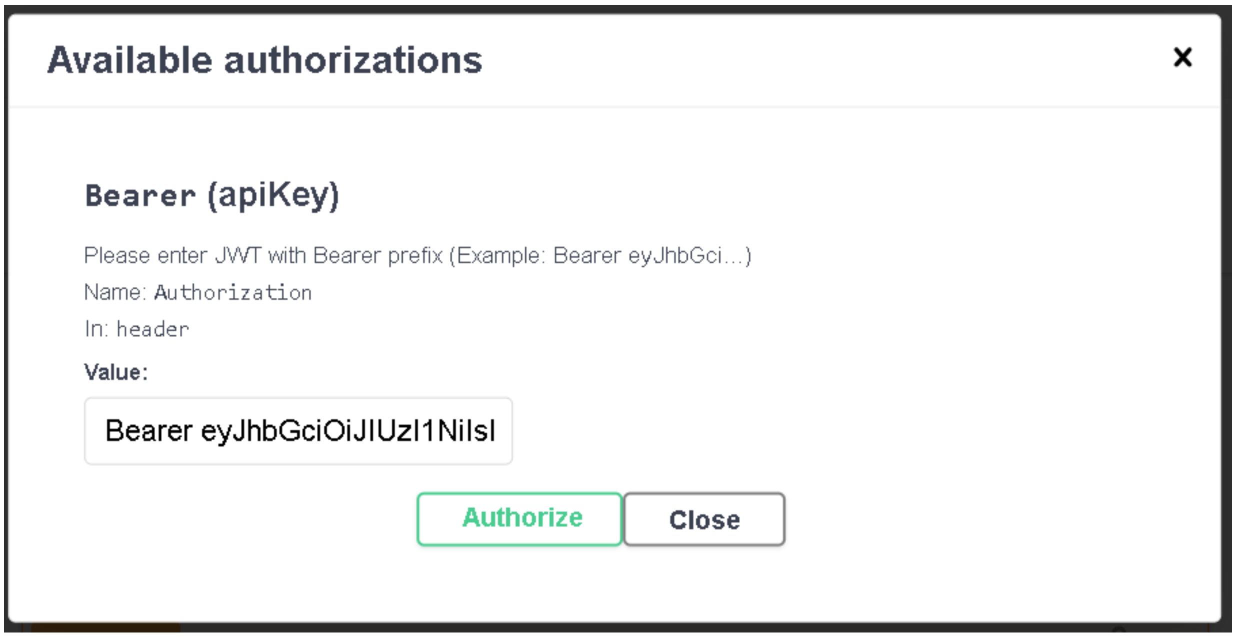
```
curl -X 'POST' \
  'http://localhost:5081/api/Auth/login' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
  "username": "somnathAdmin",
  "password": "123456"
}'
```

```
http://localhost:5081/api/Auth/login
```

```
{
  "username": "SomnathAdmin",
  "role": "Admin",
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJodHRwOi8vc2NoZW1hcy54bWxzb2FwLm9yZy93cy8yMDA1LzA1L2lkZW50aXR5L2NsYWltcy9uYW1laWRlbnRpZmllciI6IjIiLCJodHRwOi8vc2NoZW1hcy54bWxzb2FwLm
9yZy93cy8yMDA1LzA1L2lkZW50aXR5L2NsYWltcy9uYW1lIjoiU29tbmF0aEFkbWluIiwiaHR0cDovL3NjaGVtYXMubWljcm9zb2Z0LmNvbS93cy8yMDA4LzA2L2lkZW50aXR5L2NsYWltcy9yb2xlIjoiQWRtaW4iLC[illegible]
TksImlzcyI6IkRvQ29ubmVjdEFwcCIsImF1ZCI6IkRvQ29ubmVjdFVzZXJzIn0.TE9xL34dZhOqTfAJyA4pkFsZMey7fUguNlnITVz0iro"
}
```

```
content-type: application/json; charset=utf-8
date: Sat,06 Sep 2025 14:30:19 GMT
server: Kestrel
transfer-encoding: chunked
```

Fig: Token generated while Login as an Admin



Available authorizations

Bearer (apiKey)

Please enter JWT with Bearer prefix (Example: Bearer eyJhbGci...)

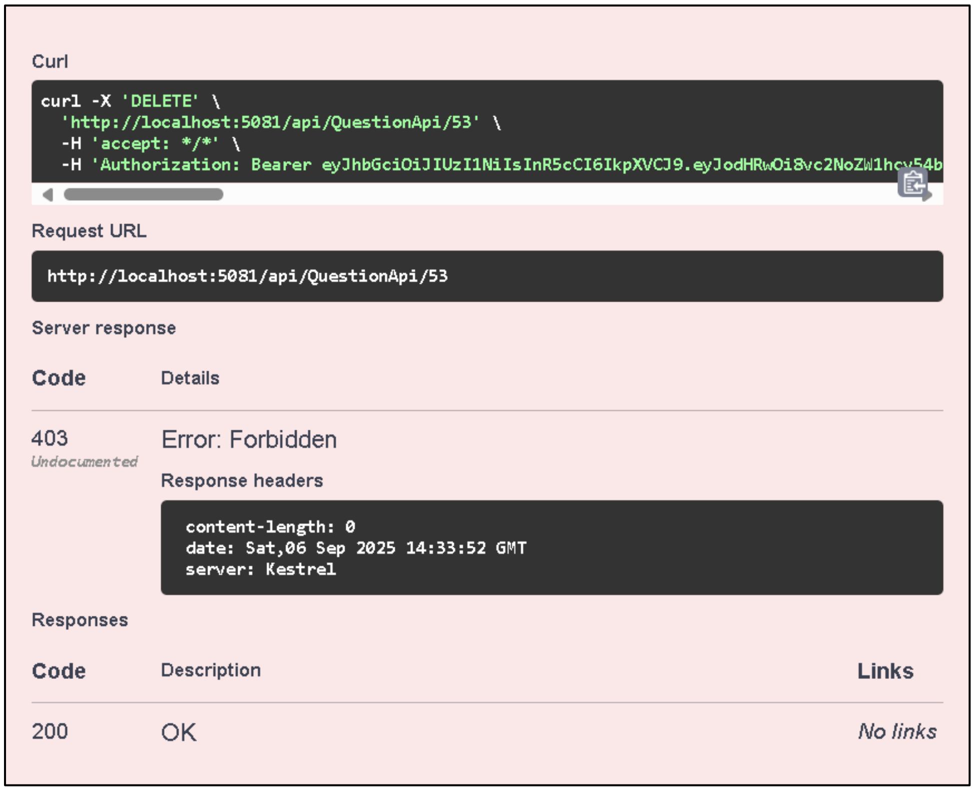
Name: Authorization

In: header

Value:

Bearer eyJhbGciOiJIUzI1NiIsI

Fig: Authorization as User/Admin


Curl

```
curl -X 'DELETE' \
  'http://localhost:5081/api/QuestionApi/53' \
  -H 'accept: */*' \
  -H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJodHRwOi8vc2NoZW1hcy54b
```

Request URL

```
http://localhost:5081/api/QuestionApi/53
```

Server response

Code	Details
403 *Undocumented*	Error: Forbidden

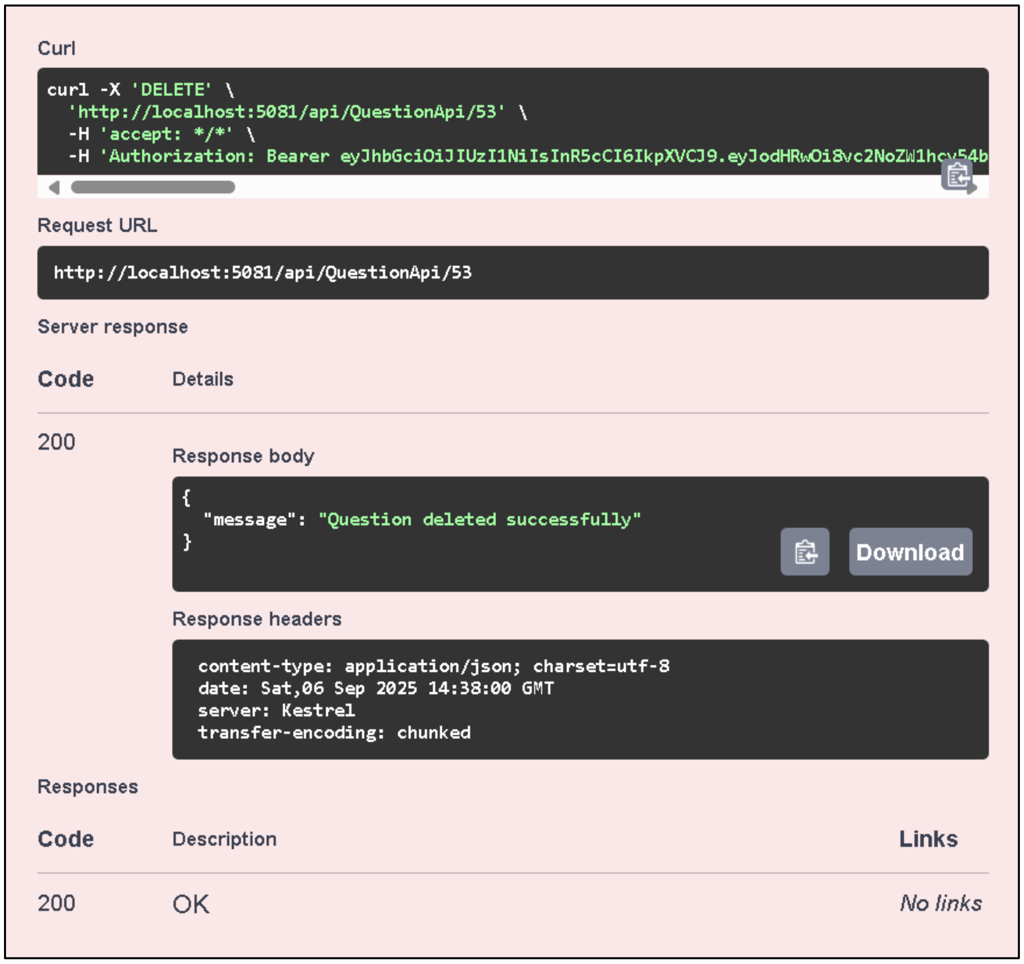
Response headers

```
content-length: 0
date: Sat,06 Sep 2025 14:33:52 GMT
server: Kestrel
```

Responses

Code	Description	Links
200	OK	*No links*

Fig: Deletion unsuccessful because Authorize happens as User


Curl

```
curl -X 'DELETE' \
  'http://localhost:5081/api/QuestionApi/53' \
  -H 'accept: */*' \
  -H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJodHRwOi8vc2NoZW1hcy54b
```

Request URL

```
http://localhost:5081/api/QuestionApi/53
```

Server response

Code	Details
200	Response body

```
{
  "message": "Question deleted successfully"
}
```

Response headers

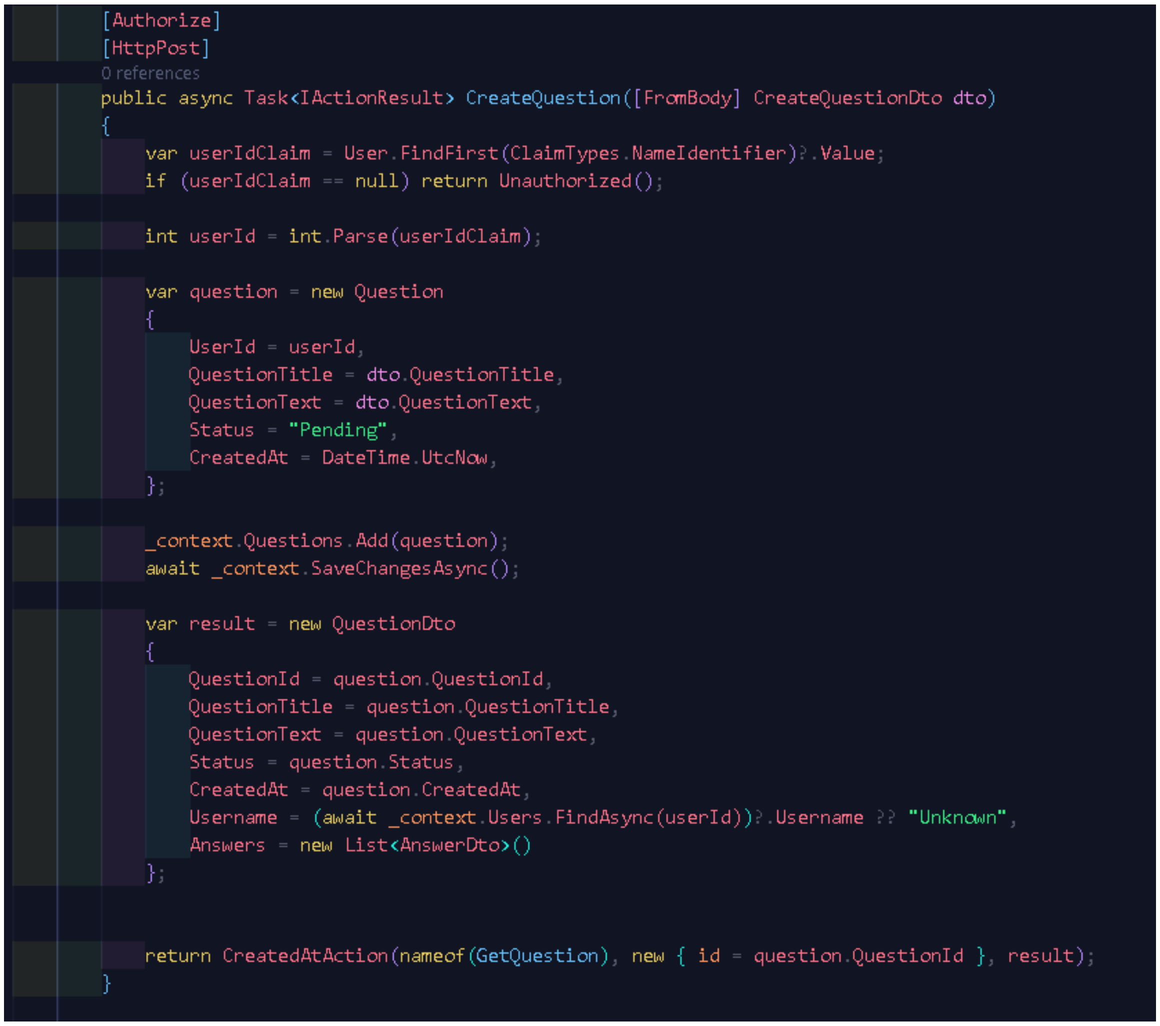
```
content-type: application/json; charset=utf-8
date: Sat,06 Sep 2025 14:38:00 GMT
server: Kestrel
transfer-encoding: chunked
```

Responses

Code	Description	Links
200	OK	*No links*

Fig: Deletion successful because Authorize happens as Admin

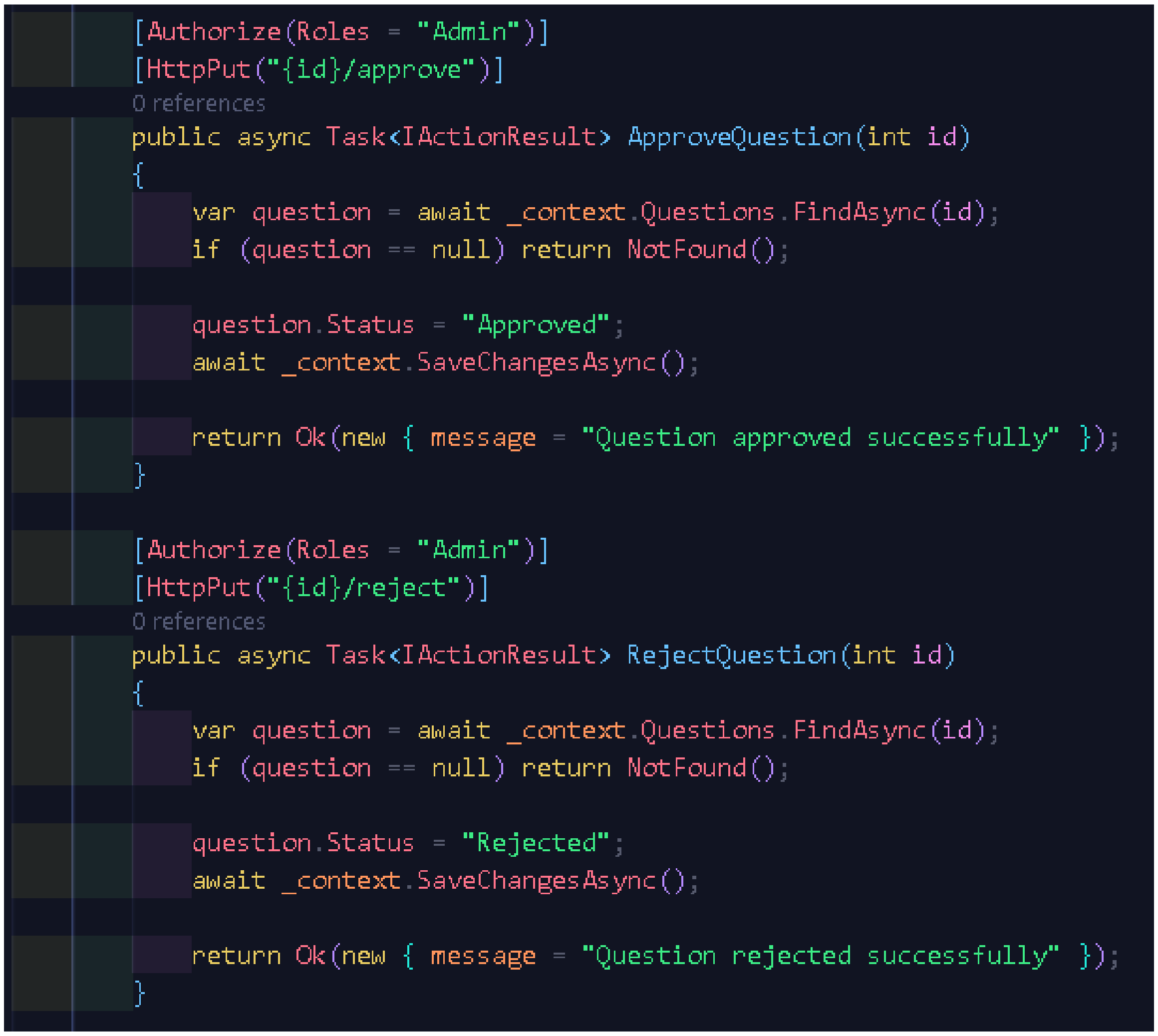
Secured protected endpoints with [Authorize] attribute



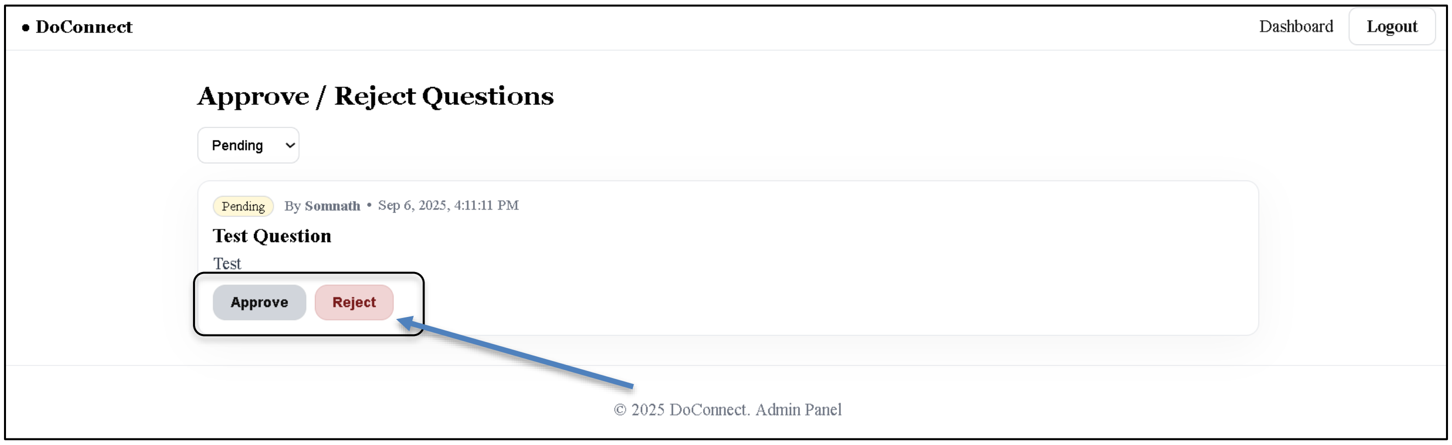
```csharp
[Authorize]
[HttpPost]
0 references
public async Task<IActionResult> CreateQuestion([FromBody] CreateQuestionDto dto)
{
    var userIdClaim = User.FindFirst(ClaimTypes.NameIdentifier)?.Value;
    if (userIdClaim == null) return Unauthorized();

    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" });
}
```



• DoConnect

Approve / Reject Questions

Pending

Pending By Somnath • Sep 6, 2025, 4:11:11 PM

Test Question

Test

Fig: Role-based actions for admin specific actions (Backend + UI)

4. Use Cases

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.

Sprint 3 - DoConnect Capstone Documentation

1. Introduction

Sprint 3 focused on enhancing the DoConnect platform with advanced functionalities, including search integration, notifications, admin approval workflows, and final system testing. The goal was to complete all core features and ensure smooth integration between the Angular frontend and the ASP.NET Core backend.

2. Objectives

A. **Search Functionality (API + Frontend)**
   - Developed a search API for questions based on query strings.
     - Integrated search functionality into the Angular frontend.

B. **Admin Notifications**
   - Implemented a notification system to alert admins when a question or answer is added.
     - Displayed notifications on the admin dashboard.

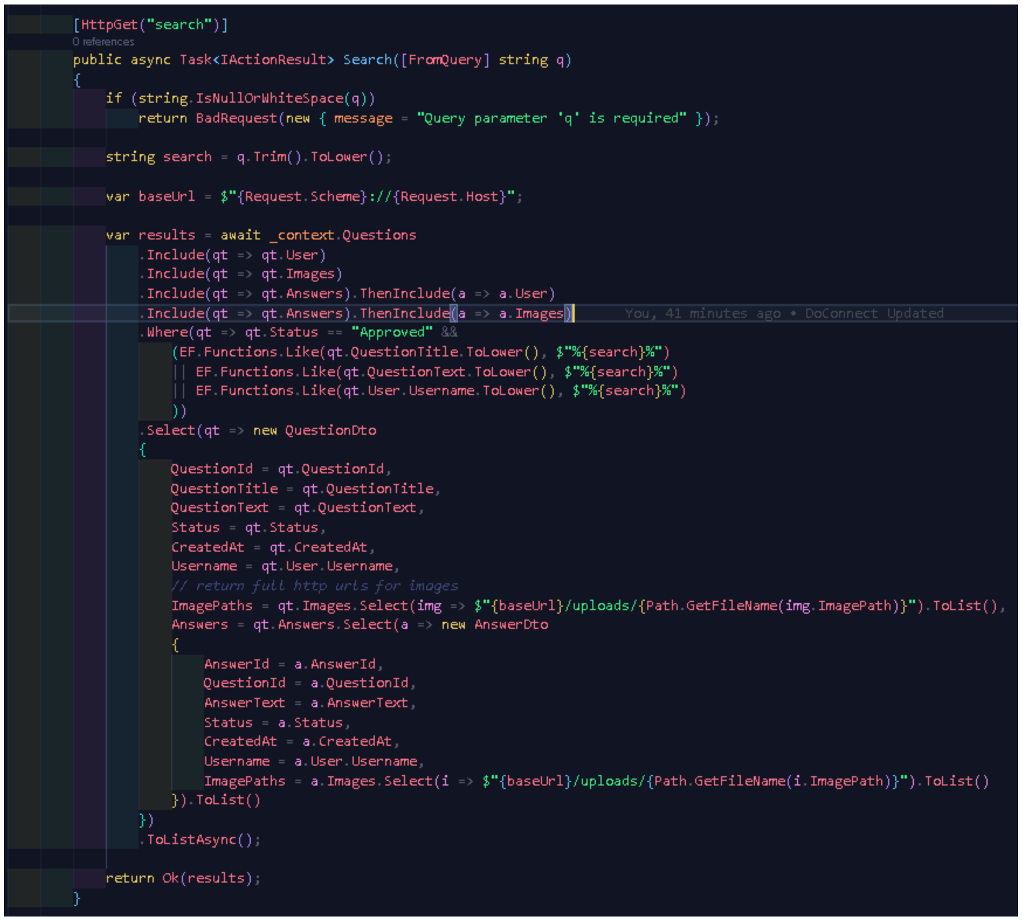
C. **Admin Approval Workflow**
   - Implemented workflows for admins to approve or reject questions and answers.
     - Ensured that only approved content is visible to users.

D. **Swagger API Testing**
   - Integrated Swagger UI for API documentation and testing.
     - Verified endpoints for authentication, questions, answers, and images.

E. **Final Integration**
   - Ensured complete integration of Angular frontend with backend API.
     - Performed full system testing to identify and fix bugs.

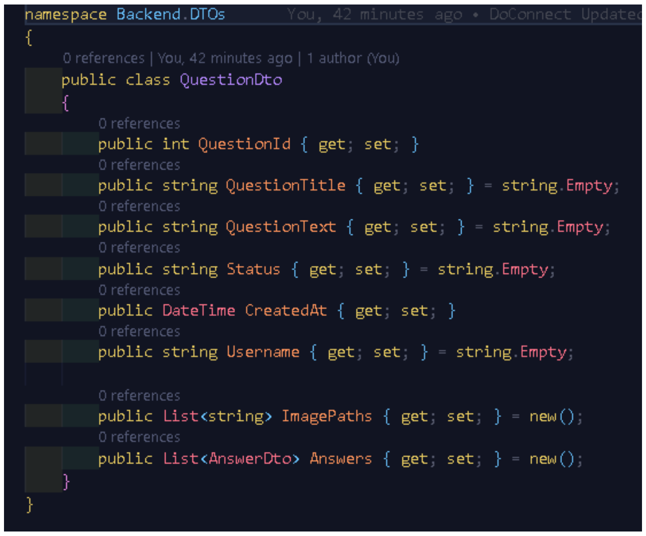
3. Deliverables

➢ **Search functionality implemented in both API and frontend –**
A search API was added in the backend to filter questions by query strings. On the frontend, search bars were integrated in the Question List and Answer Question pages to let users quickly find relevant approved questions.



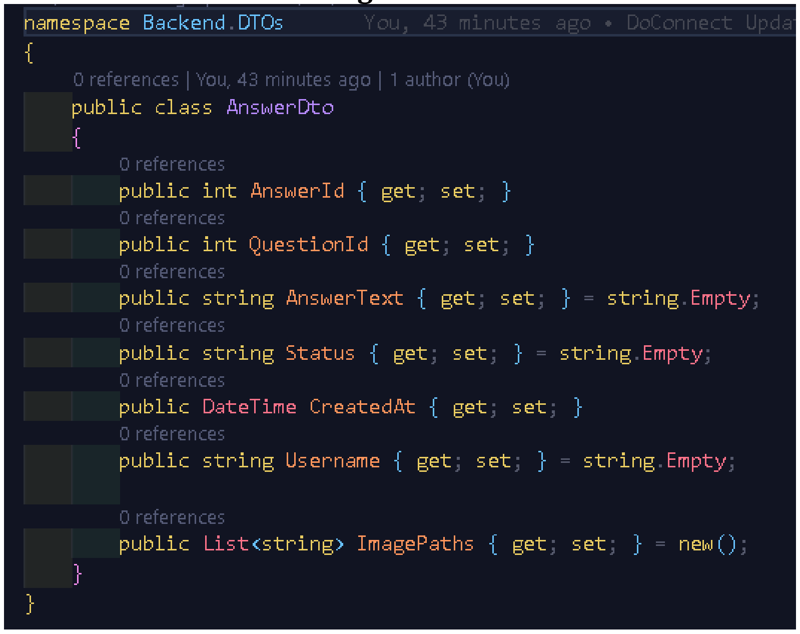
```
[HttpGet("search")]
0 references
public async Task<IActionResult> Search([FromQuery] string q)
{
    if (string.IsNullOrWhiteSpace(q))
        return BadRequest(new { message = "Query parameter 'q' is required" });

    string search = q.Trim().ToLower();

    var baseUrl = $"{Request.Scheme}://{Request.Host}";

    var results = await _context.Questions
        .Include(qt => qt.User)
        .Include(qt => qt.Images)
        .Include(qt => qt.Answers).ThenInclude(a => a.User)
        .Include(qt => qt.Answers).ThenInclude(a => a.Images)        You, 41 minutes ago • DoConnect Updated
        .Where(qt => qt.Status == "Approved" &&
            (EF.Functions.Like(qt.QuestionTitle.ToLower(), $"%{search}%")
            || EF.Functions.Like(qt.QuestionText.ToLower(), $"%{search}%")
            || EF.Functions.Like(qt.User.Username.ToLower(), $"%{search}%")
            ))
        .Select(qt => new QuestionDto
        {
            QuestionId = qt.QuestionId,
            QuestionTitle = qt.QuestionTitle,
            QuestionText = qt.QuestionText,
            Status = qt.Status,
            CreatedAt = qt.CreatedAt,
            Username = qt.User.Username,
            // return full http urls for images
            ImagePaths = qt.Images.Select(img => $"{baseUrl}/uploads/{Path.GetFileName(img.ImagePath)}").ToList(),
            Answers = qt.Answers.Select(a => new AnswerDto
            {
                AnswerId = a.AnswerId,
                QuestionId = a.QuestionId,
                AnswerText = a.AnswerText,
                Status = a.Status,
                CreatedAt = a.CreatedAt,
                Username = a.User.Username,
                ImagePaths = a.Images.Select(i => $"{baseUrl}/uploads/{Path.GetFileName(i.ImagePath)}").ToList()
            }).ToList()
        })
        .ToListAsync();

    return Ok(results);
}
```

Fig – Controller Endpoint implemented



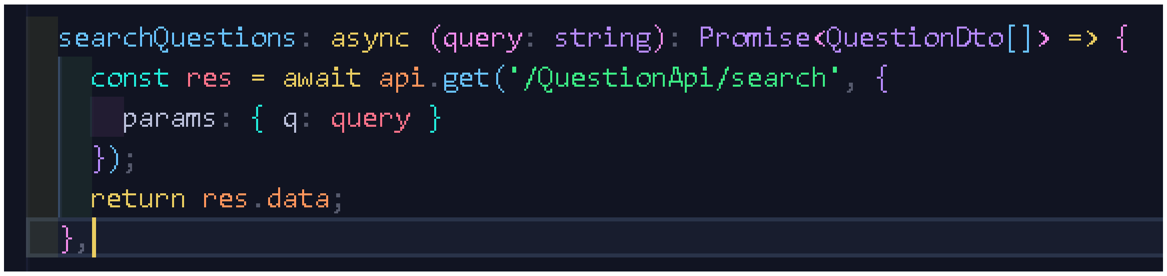
```
namespace Backend.DTOs        You, 42 minutes ago • DoConnect Updated
{
    0 references | You, 42 minutes ago | 1 author (You)
    public class QuestionDto
    {
        0 references
        public int QuestionId { get; set; }
        0 references
        public string QuestionTitle { get; set; } = string.Empty;
        0 references
        public string QuestionText { get; set; } = string.Empty;
        0 references
        public string Status { get; set; } = string.Empty;
        0 references
        public DateTime CreatedAt { get; set; }
        0 references
        public string Username { get; set; } = string.Empty;

        0 references
        public List<string> ImagePaths { get; set; } = new();
        0 references
        public List<AnswerDto> Answers { get; set; } = new();
    }
}
```

Fig - DTOs

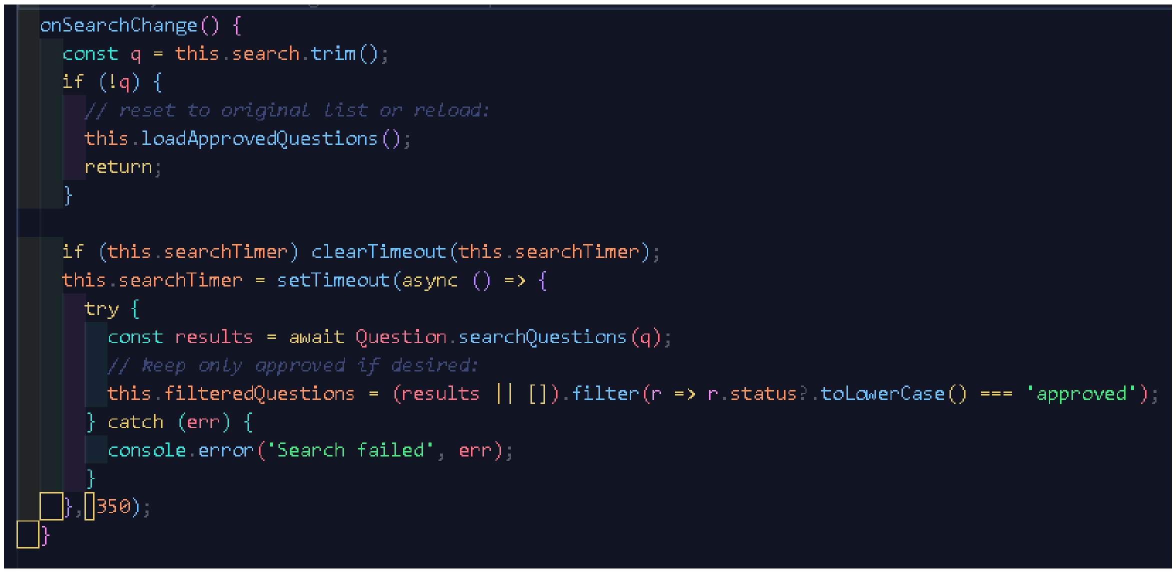


```
namespace Backend.DTOs        You, 43 minutes ago • DoConnect Upda
{
    0 references | You, 43 minutes ago | 1 author (You)
    public class AnswerDto
    {
        0 references
        public int AnswerId { get; set; }
        0 references
        public int QuestionId { get; set; }
        0 references
        public string AnswerText { get; set; } = string.Empty;
        0 references
        public string Status { get; set; } = string.Empty;
        0 references
        public DateTime CreatedAt { get; set; }
        0 references
        public string Username { get; set; } = string.Empty;

        0 references
        public List<string> ImagePaths { get; set; } = new();
    }
}
```

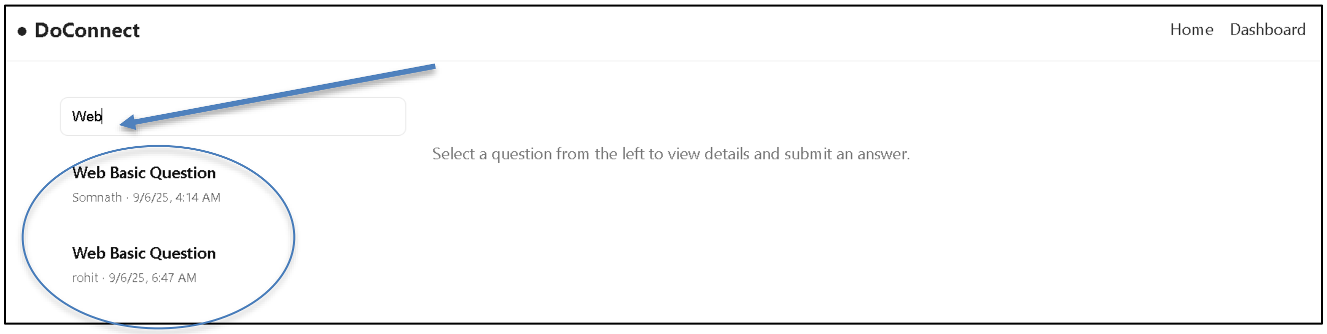
Fig - DTOs



```
searchQuestions: async (query: string): Promise<QuestionDto[]> => {
  const res = await api.get('/QuestionApi/search', {
    params: { q: query }
  });
  return res.data;
},
```



```
onSearchChange() {
  const q = this.search.trim();
  if (!q) {
    // reset to original list or reload:
    this.loadApprovedQuestions();
    return;
  }

  if (this.searchTimer) clearTimeout(this.searchTimer);
  this.searchTimer = setTimeout(async () => {
    try {
      const results = await Question.searchQuestions(q);
      // keep only approved if desired:
      this.filteredQuestions = (results || []).filter(r => r.status?.toLowerCase() === 'approved');
    } catch (err) {
      console.error('Search failed', err);
    }
  }, 350);
}
```



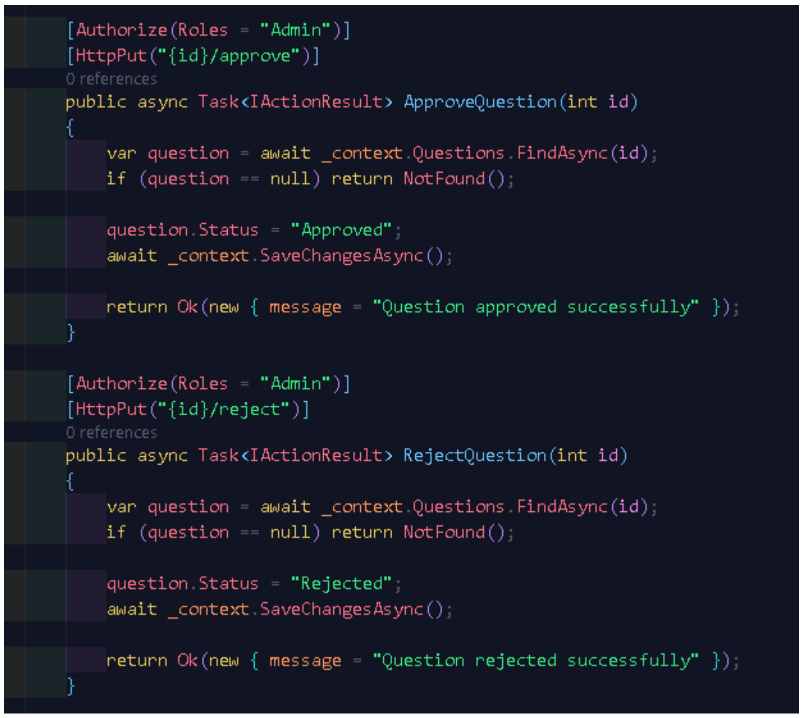
Web Basic Question
Somnath · 9/6/25, 4:14 AM

Web Basic Question
rohit · 9/6/25, 6:47 AM

Select a question from the left to view details and submit an answer.

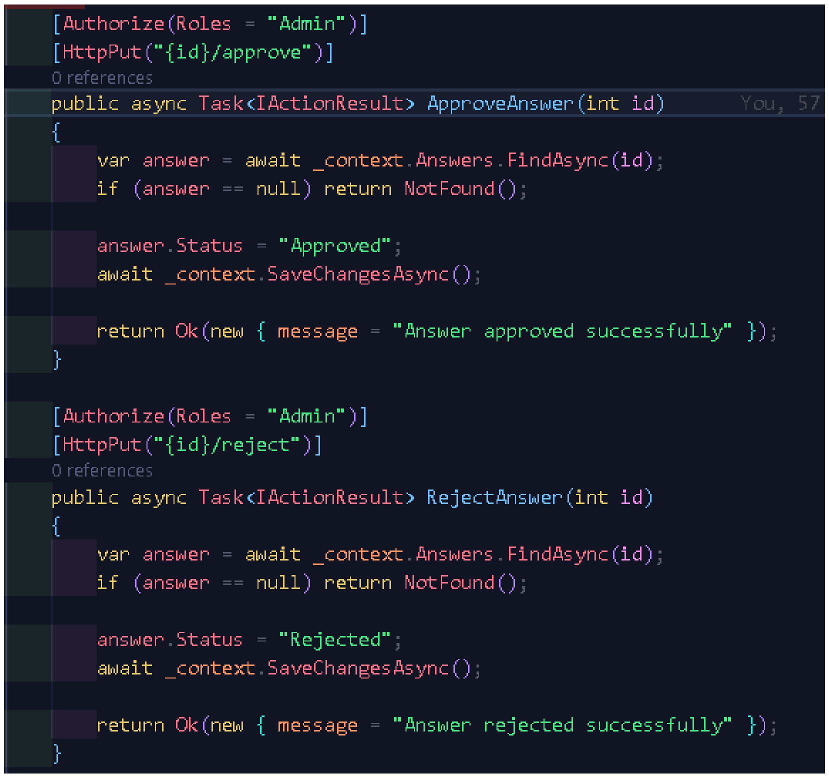
Fig – Search Functionalities in Frontend

➢ **Admin approval module with notifications –**
Approval workflows for questions and answers were completed. Admins can approve/reject pending items, and buttons are only shown for Pending status. A notification system was added to alert admins whenever new questions or answers are submitted.



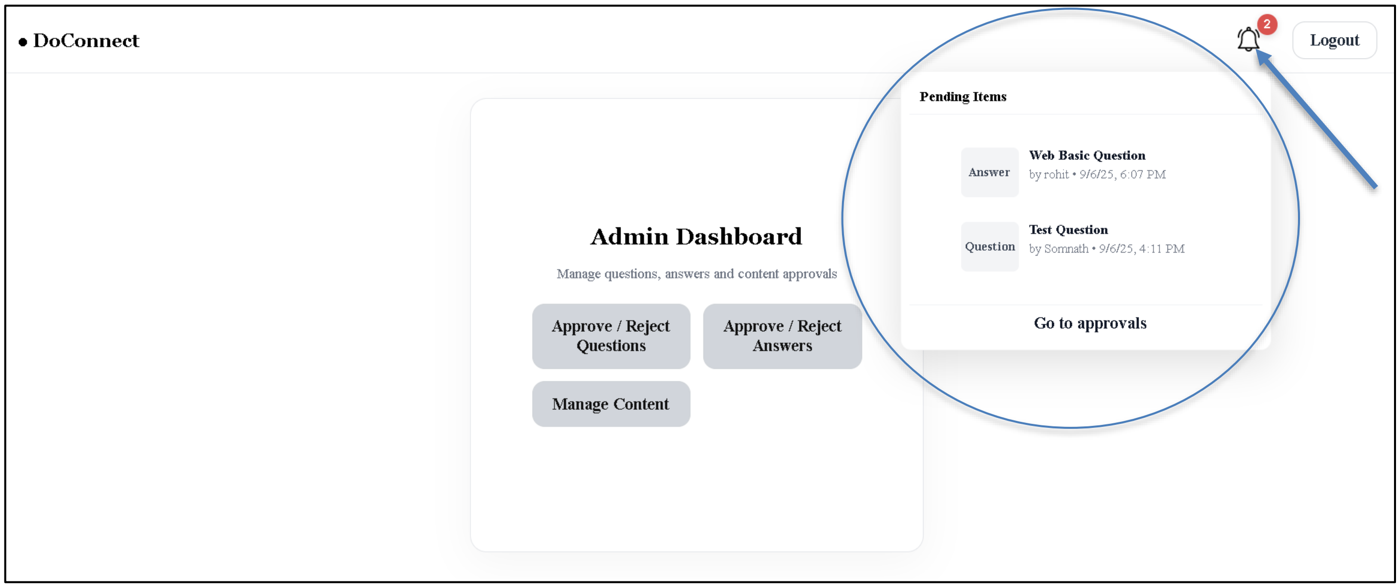
```
[Authorize(Roles = "Admin")]
[HttpPut("{id}/approve")]
0 references
public async Task<IActionResult> ApproveQuestion(int id)
{
    var question = await _context.Questions.FindAsync(id);
    if (question == null) return NotFound();

    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- Approve/Reject Endpoints



```
[Authorize(Roles = "Admin")]
[HttpPut("{id}/approve")]
0 references
public async Task<IActionResult> ApproveAnswer(int id)        You, 57
{
    var answer = await _context.Answers.FindAsync(id);
    if (answer == null) return NotFound();

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

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

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

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

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

Fig- Approve/Reject Endpoints



Admin Dashboard

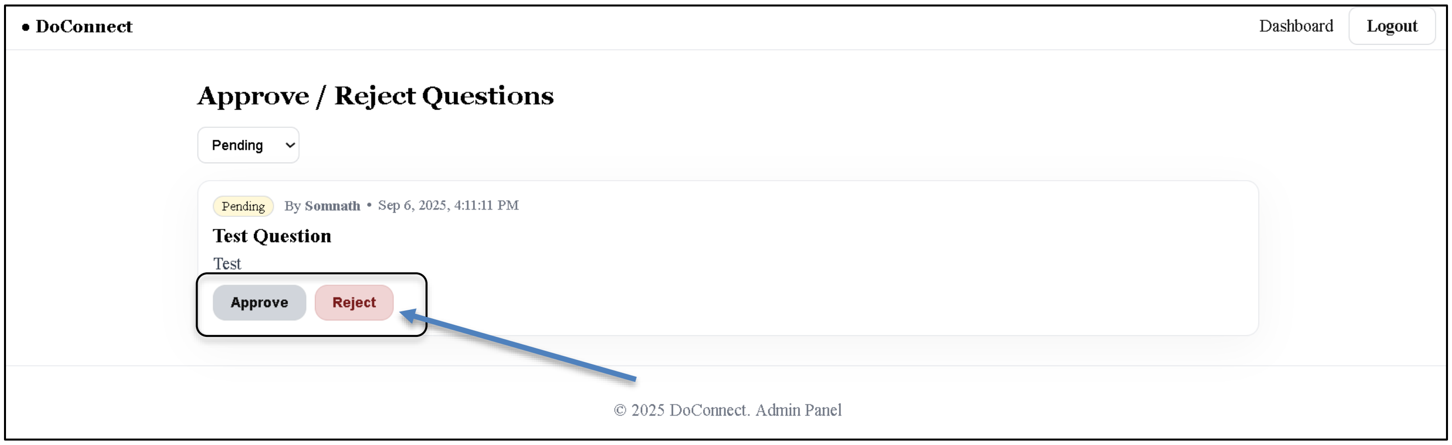
Manage questions, answers and content approvals

Pending Items

Answer — Web Basic Question — by rohit • 9/6/25, 6:07 PM

Question — Test Question — by Somnath • 9/6/25, 4:11 PM

Fig – Admin gets notified while a User request a question/answer for approval



• DoConnect

Dashboard Logout

Approve / Reject Questions

Pending

Pending By Somnath • Sep 6, 2025, 4:11:11 PM

Test Question

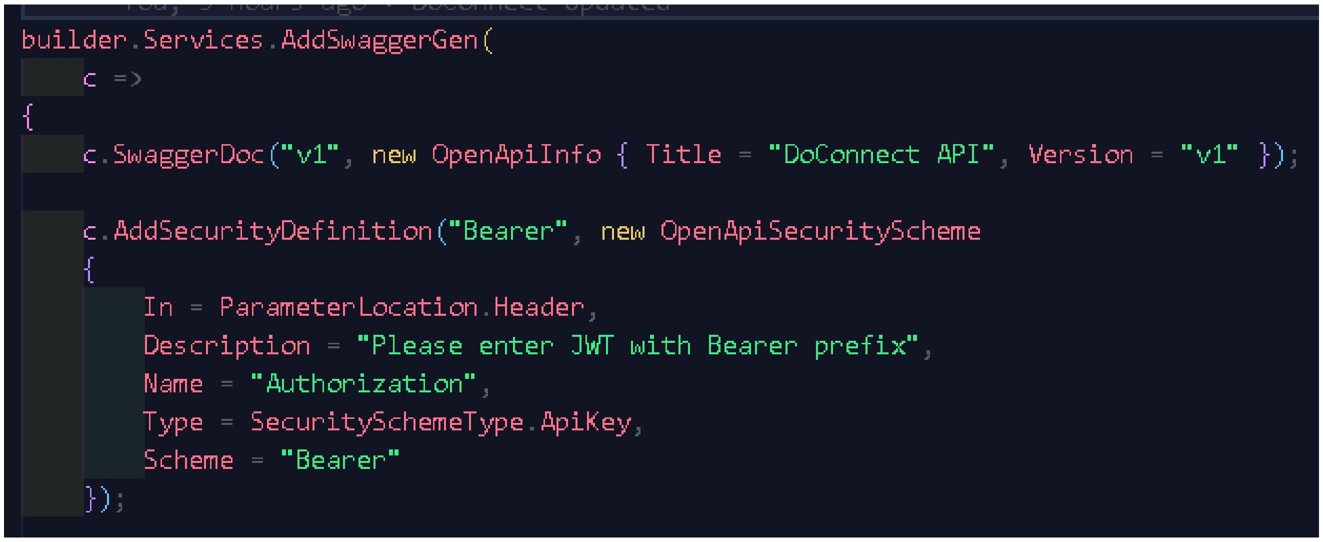
Test

Approve Reject

© 2025 DoConnect. Admin Panel

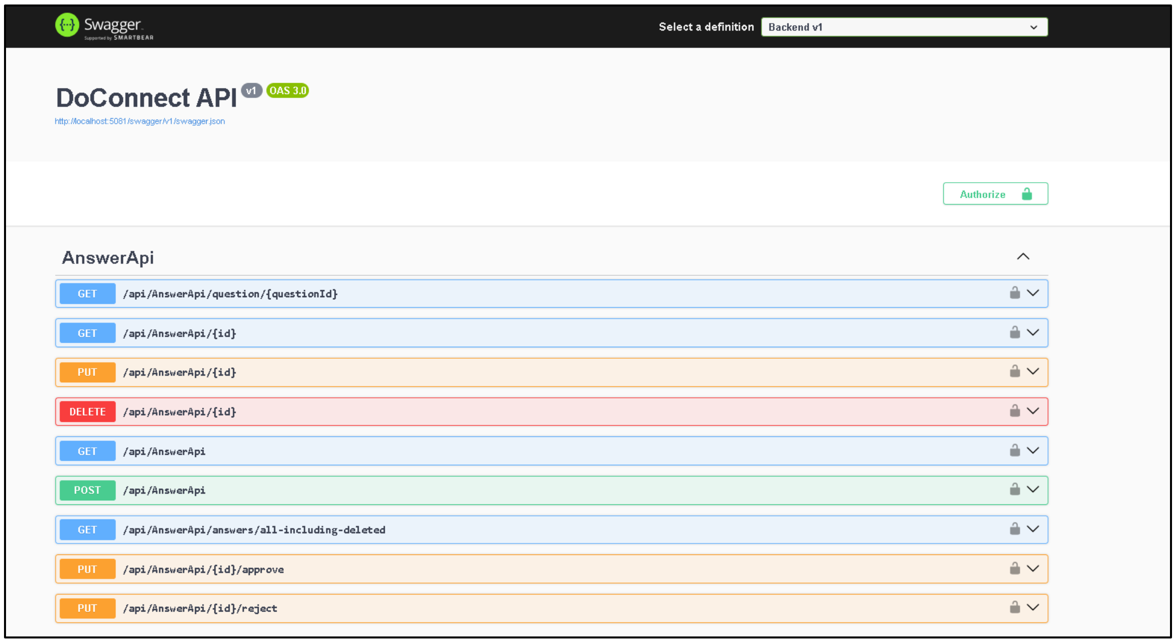
Fig – Approve/Reject operations by Admin

➢ **Swagger UI documentation for all Web API endpoints-**
Swagger was configured to document and test all Web API endpoints, including authentication, CRUD operations, and image uploads, with support for JWT authorization.



```
builder.Services.AddSwaggerGen(
    c =>
{
    c.SwaggerDoc("v1", new OpenApiInfo { Title = "DoConnect API", Version = "v1" });

    c.AddSecurityDefinition("Bearer", new OpenApiSecurityScheme
    {
        In = ParameterLocation.Header,
        Description = "Please enter JWT with Bearer prefix",
        Name = "Authorization",
        Type = SecuritySchemeType.ApiKey,
        Scheme = "Bearer"
    });
```

Fig – Swagger Setup (Program.cs)


Swagger

Select a definition Backend v1

DoConnect API v1 OAS 3.0

http://localhost:5081/swagger/v1/swagger.json

Authorize

AnswerApi

GET /api/AnswerApi/question/{questionId}

GET /api/AnswerApi/{id}

PUT /api/AnswerApi/{id}

DELETE /api/AnswerApi/{id}

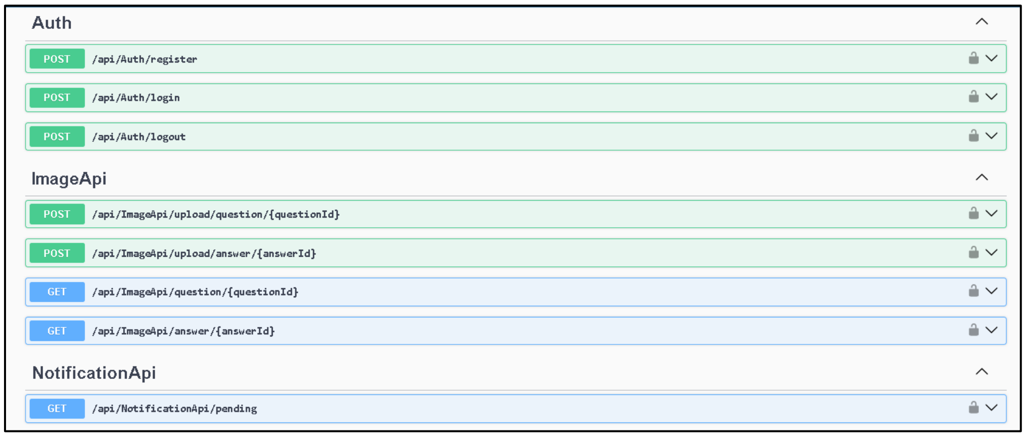
GET /api/AnswerApi

POST /api/AnswerApi

GET /api/AnswerApi/answers/all-including-deleted

PUT /api/AnswerApi/{id}/approve

PUT /api/AnswerApi/{id}/reject


Auth

POST /api/Auth/register

POST /api/Auth/login

POST /api/Auth/logout

ImageApi

POST /api/ImageApi/upload/question/{questionId}

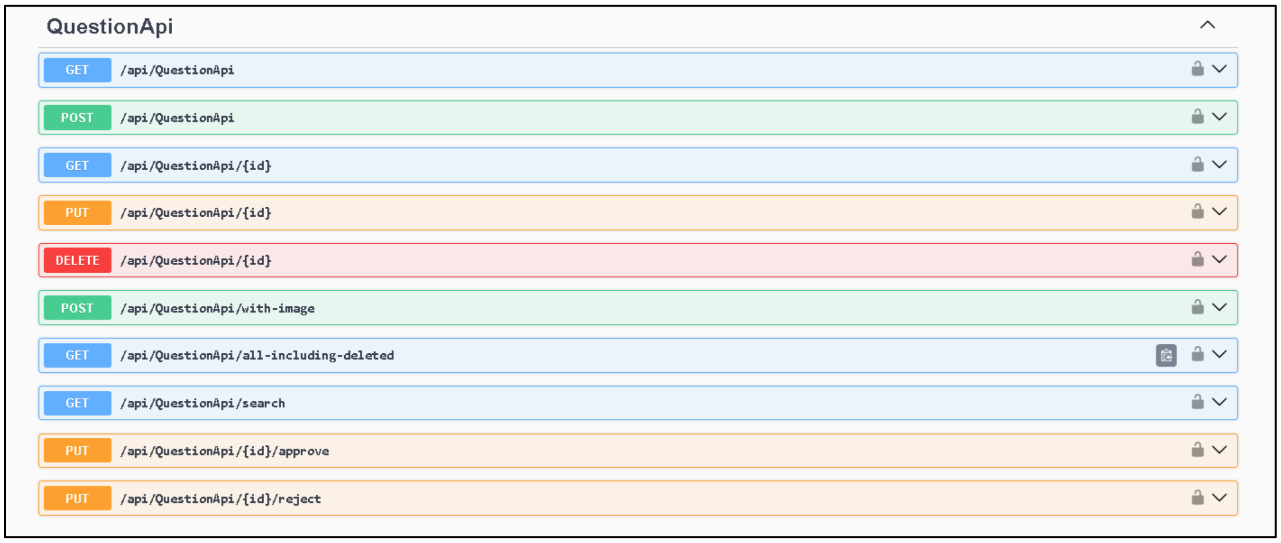
POST /api/ImageApi/upload/answer/{answerId}

GET /api/ImageApi/question/{questionId}

GET /api/ImageApi/answer/{answerId}

NotificationApi

GET /api/NotificationApi/pending


QuestionApi

GET /api/QuestionApi

POST /api/QuestionApi

GET /api/QuestionApi/{id}

PUT /api/QuestionApi/{id}

DELETE /api/QuestionApi/{id}

POST /api/QuestionApi/with-image

GET /api/QuestionApi/all-including-deleted

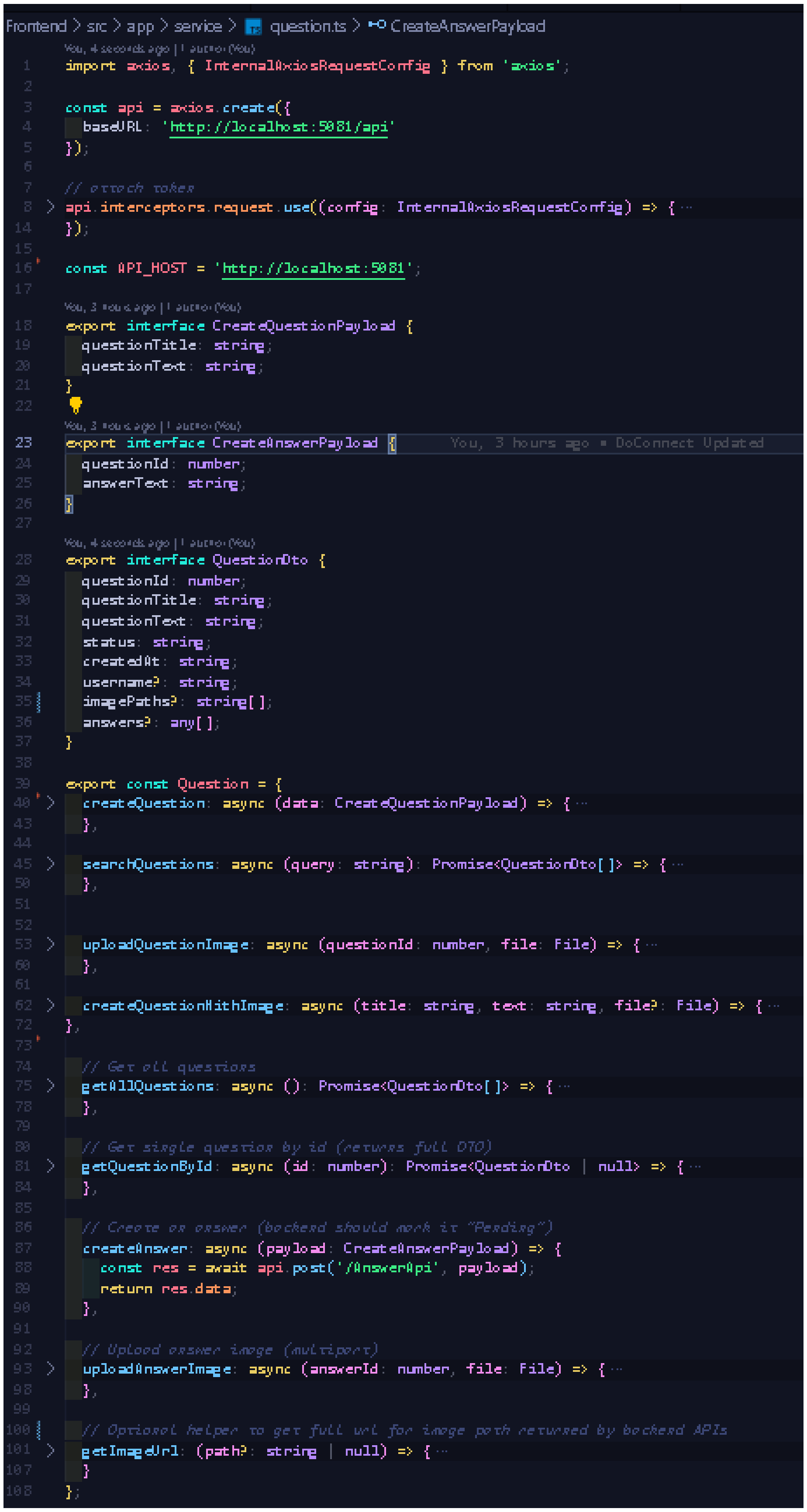
GET /api/QuestionApi/search

PUT /api/QuestionApi/{id}/approve

PUT /api/QuestionApi/{id}/reject

Fig – Swagger UI

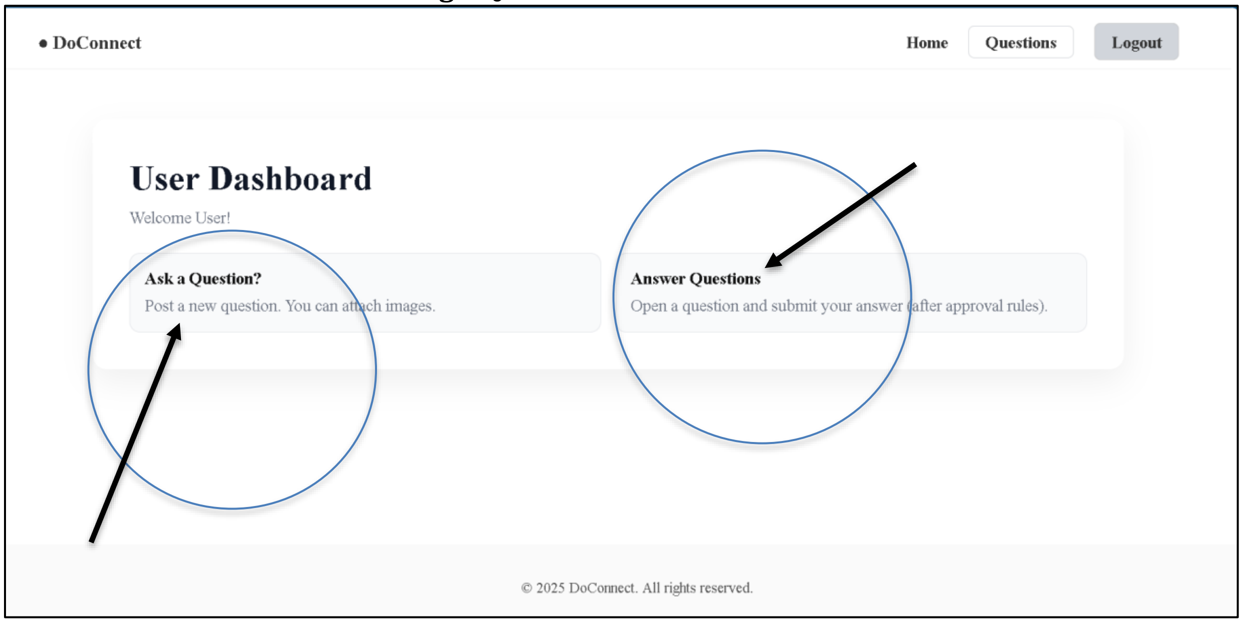
➢ **Fully functional Angular frontend consuming ASP.NET Core MVC backend API –** Swagger was configured to document and test all Web API endpoints, including authentication, CRUD operations, and image uploads, with support for JWT authorization.



Frontend > src > app > service > question.ts > CreateAnswerPayload

```
You, 4 seconds ago | 1 author (You)
import axios, { InternalAxiosRequestConfig } from 'axios';

const api = axios.create({
  baseURL: 'http://localhost:5081/api'
});

// attach token
api.interceptors.request.use((config: InternalAxiosRequestConfig) => { ···
});

const API_HOST = 'http://localhost:5081';

You, 3 hours ago | 1 author (You)
export interface CreateQuestionPayload {
  questionTitle: string;
  questionText: string;
}

You, 3 hours ago | 1 author (You)
export interface CreateAnswerPayload {        You, 3 hours ago • DoConnect Updated
  questionId: number;
  answerText: string;
}

You, 4 seconds ago | 1 author (You)
export interface QuestionDto {
  questionId: number;
  questionTitle: string;
  questionText: string;
  status: string;
  createdAt: string;
  username?: string;
  imagePaths?: string[];
  answers?: any[];
}

export const Question = {
  createQuestion: async (data: CreateQuestionPayload) => { ···
  },

  searchQuestions: async (query: string): Promise<QuestionDto[]> => { ···
  },


  uploadQuestionImage: async (questionId: number, file: File) => { ···
  },

  createQuestionWithImage: async (title: string, text: string, file?: File) => { ···
},

  // Get all questions
  getAllQuestions: async (): Promise<QuestionDto[]> => { ···
  },

  // Get single question by id (returns full DTO)
  getQuestionById: async (id: number): Promise<QuestionDto | null> => { ···
  },

  // Create an answer (backend should mark it "Pending")
  createAnswer: async (payload: CreateAnswerPayload) => {
    const res = await api.post('/AnswerApi', payload);
    return res.data;
  },

  // Upload answer image (multipart)
  uploadAnswerImage: async (answerId: number, file: File) => { ···
  },

  // Optional helper to get full url for image path returned by backend APIs
  getImageUrl: (path?: string | null) => { ···
  }
};
```

Fig – Question Service Methods



User Dashboard

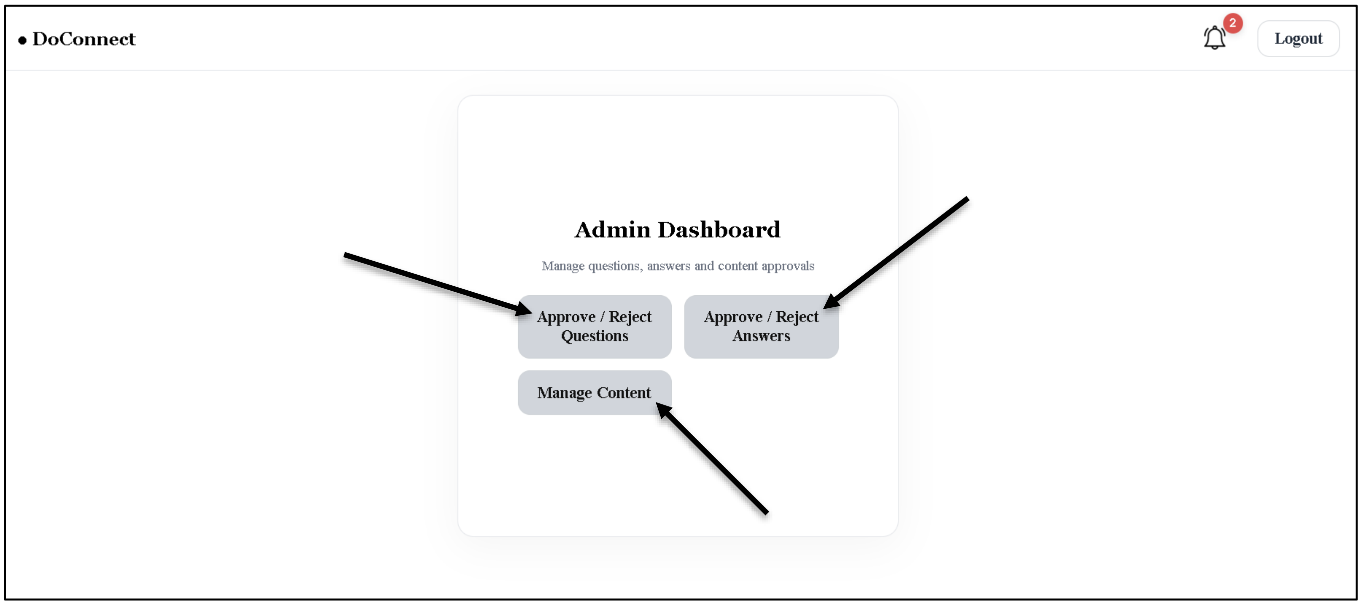
Welcome User!

Ask a Question?
Post a new question. You can attach images.

Answer Questions
Open a question and submit your answer (after approval rules).

© 2025 DoConnect. All rights reserved.

Fig- User Workflow Components



Admin Dashboard

Manage questions, answers and content approvals

Fig- Admin Workflow Components

4. Use Cases

User Workflow:
- Search and view approved questions and answers.
- Continue posting questions/answers with images (pending approval).

Admin Workflow:
- Receive notifications when new questions/answers are submitted.
- Approve/Reject submitted content.
- Manage content visibility across the platform.

5. Conclusion

Sprint 3 successfully completed the DoConnect project by integrating all modules, enhancing admin workflows with notifications, and ensuring robust testing. With search, approval, and notification features in place, the system is fully functional and ready for deployment.