CAPSTONE PROJECT

FINAL REPORT

DoConnect – A Q&A Platform

(.NET + ANGULAR)

SUBMITTED BY

SARAVANAN M

WIPRO NGA - .Net Full Stack Angular - FY26 – C2

UNDER THE GUIDANCE OF

JYOTI S PATIL

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to my mentor **Mr.Harshavardhan Shandilya**, faculties – Niti Dwivedi and Jyoti S Patil for their continuous guidance and support throughout the completion of this capstone project **"DoConnect – A Q&A Platform"**.

This project would not have been possible without the invaluable learning resources, documentation, and training provided by **"Wipro Pre-Skill Training"** through **"Great Learning Platform"**. I am especially thankful to my instructors for their constructive feedback, which helped me refine the technical design, coding practices, and testing approach.

I also wish to acknowledge the support of my colleagues and friends who encouraged me and provided helpful insights during the development of both the backend and frontend modules.

Finally, I am grateful for the opportunity to apply the skills acquired during the course in a real-world project, which has greatly enhanced my confidence and understanding of **ASP.NET Core, Angular, SQL, and software testing practices**.

LIST OF CONTENT

ABSTRACT

The project **"DoConnect – A Q&A Platform"** is a full-stack web application developed using **ASP.NET Core Web API** for the backend and **Angular** for the frontend. The system is designed to provide a collaborative environment where users can register, authenticate, ask questions, provide answers, and upload images. To ensure content quality, an **Admin role** is implemented with the ability to approve or reject questions and answers, as well as manage user accounts.

The backend uses **Entity Framework Core** with **SQL Server** for persistent data storage, secured with **JWT-based authentication** and role-based authorization. The frontend offers an intuitive interface with search functionality, enabling users to quickly find relevant content. All API endpoints are documented and tested using **Swagger UI**, while the robustness of the application is validated through **Cypress tests**.

This project demonstrates the integration of modern software engineering practices, including **CRUD operations, authentication and authorization, testing frameworks, and modular frontend design**, fulfilling the capstone rubric requirements. The outcome is a reliable, scalable, and secure platform that fosters knowledge sharing in a community-driven manner.

PROBLEM STATEMENT

Project Name: DoConnect (Frontend in Angular, Backend in ASP.NET Core MVC)

PROBLEM:

DoConnect is a Q&A platform where users can ask and answer questions related to various technical topics. The application has two types of users:

- ADMIN
- USER

TECH STACK:

- Frontend : Angular
- Backend : ASP.NET Core MVC with WEB API
- Database : SQL Server (using EF core as ORM)
- File Storage : File upload functionality on the server
- API Documentation : Swagger for Web API

FUNCTIONALITY REQUIREMENTS:

USER STORIES:

1. **User Authentication:**

   As a user, I should be able to log in, log out, and register into the application.

2. **Ask Questions:**

   As a user, I should be able to ask a question under any topic.

3. **Search Questions**:

   As a user, I should be able to search for questions based on a search query.

4. **Answer Questions:**

   As a user, I should be able to answer any question posted

5. **Multiple Answers:**

   As a user, I should be able to provide multiple answers to the same question.

6. **Image Upload:**

   As a user, I should be able to upload images along with my question or answer.

ADMIN STORIES:

1. **Admin Authentication:**

   As an admin, I should be able to log in, log out, and register into the application.

2. **Receive Notifications:**

   As an admin, I should receive notifications when a new question is posted or an answer is given.

3. **Approve Questions & Answers:**

   As an admin, I should be able to approve or reject questions and answers, making them visible on the platform only after approval.

4. **Moderate Content:**

   As an admin, I should be able to delete inappropriate questions or answers.

SOFTWARE REQUIREMENTS:

TECHNOLOGIES AND FRAMEWORKS:

- **Frontend:** Angular, TypeScript, HTML5, SCSS
- **Backend:** ASP.NET Core Web API (C#), Entity Framework Core
- **Database:** Microsoft SQL Server
- **API Documentation & Testing:** Swagger (Swashbuckle)
- **Testing Frameworks:** Npx Cypress Frontend Testing
- **Package Manager:** NuGet (for .NET), npm (for Angular)

IDE / DEVELOPMENT TOOLS:

- **Visual Studio Code** (for frontend development)
- **SQL Server Management Studio (SSMS)** (for database management)
- **Postman / Swagger UI** (for API testing)
- **Git & GitHub** (for version control)

PROGRAMMING LANGUAGES:

- **C#** : Backend
- **TypeScript** : Frontend
- **SQL** : Database
- **HTML5 & SCSS** : Designing

INTRODUCTION

The advancement of technology and the growing availability of online learning resources have significantly increased the demand for collaborative platforms where individuals can exchange knowledge and solve problems together. **DoConnect** is a community-driven **Question & Answer (Q&A) platform** designed to facilitate effective knowledge sharing, much like platforms such as Stack Overflow, but built as a capstone project to demonstrate full-stack development skills.

The system provides two major roles: **User** and **Admin**. Users can register, authenticate, and perform various activities such as posting questions, providing answers, uploading images, and searching existing content. Meanwhile, the Admin oversees the platform by moderating content, approving or rejecting questions and answers, and managing user accounts. This ensures both **quality control** and **secure participation** within the platform.

DoConnect is implemented using a **modern technology stack**:

- **Frontend:** Angular for building a responsive and interactive Single Page Application (SPA).
- **Backend:** ASP.NET Core Web API with C# for handling business logic and data operations.
- **Database:** SQL Server with Entity Framework Core for secure and efficient data storage.
- **Authentication & Authorization:** JSON Web Tokens (JWT) for secure role-based access.
-

In addition, the project emphasizes **software engineering best practices** such as:

- Modular design and separation of concerns.
- Documentation and API testing using **Swagger**.
- Automated validation through **unit and integration testing** (xUnit, Moq, FluentAssertions).
- Version control and collaborative development using Git.

ANGULAR ARCHITECTURE

The **frontend** of the DoConnect application is implemented using **Angular**, a component-based framework for building scalable single-page applications (SPA). Angular's architecture follows the **Model–View–Controller (MVC)** design principle adapted to a client-side environment, ensuring modularity, maintainability, and testability.

KEY ELEMENTS OF ANGULAR ARCHITECTURE

1. MODULES

- Angular applications are divided into **NgModules**, which group related components, directives, pipes, and services.
- In DoConnect :

  AppModule – Root module, bootstraps the app

  AuthModule – Handles login and registration.

  QuestionsModule – Displays, creates, and searches questions.

  AdminModule – Provides admin dashboard and user/content management.

2. COMPONENTS

- Components control sections of the UI and encapsulate **HTML, CSS, and TypeScript logic**.
- Examples in DoConnect:

  login.component.ts – Handles user login.

  list.component.ts – Displays all questions with a search bar.

  detail.component.ts – Shows question details with answers.

  admin-dashboard.component.ts – Admin interface to approve/reject Q&A and manage users.

3. TEMPLATES

- **Angular Templates** are used to define the **view** of a component.
- They combine **HTML** with Angular features to display **dynamic data**.
- Templates support **interpolation**, **property binding**, **event binding**, and **two-way binding**.
- They use **directives** like *ngIf, *ngFor, and ngClass to control the structure and style.
- **Pipes** are used within templates to **format and transform data** easily.

4. DIRECTIVES

Directives in Angular are special instructions in templates that change the behavior or appearance of DOM elements.
They are classified into three main types:

- **Component Directives** → Directives with their own template (e.g., custom components).
- **Structural Directives** → Change the **DOM structure** (e.g., *ngIf, *ngFor, *ngSwitch).
- **Attribute Directives** → Change the **look or behavior** of elements (e.g., ngClass, ngStyle).

5. SERVICES

Services in Angular are used to share data, logic, and functionality across multiple components.
They help keep the code **clean, reusable, and maintainable**.

- Services are usually created with the **@Injectable()** decorator.
- They are used for tasks like **API calls**, **data sharing**, and **business logic**.
- Services are provided to components using **Dependency Injection (DI)**.

6. ROUTING

- Angular's **RouterModule** manages navigation between views without reloading the page.
- Defined in app-routing.module.ts.
- In DoConnect:
    - /login → Login component
    - /register → Register component
    - /questions → Question list component
    - /questions/:id → Question detail component
    - /admin → Admin dashboard

7. DATA BINDING

Data Binding in Angular is the process of connecting the component's data with the template (HTML view). It allows automatic synchronization between the model (component) and the view.

There are **four types** of data binding:

- **Interpolation ({{ }})** → Display data → {{ name }}
- **Property Binding [ ]** → Bind values to HTML properties → <img [src]="imageUrl">
- **Event Binding ( )** → Handle user events → <button (click)="show()">Click</button>
- **Two-Way Binding [(ngModel)]** → Sync data both ways → <input [(ngModel)]="username">

It makes Angular apps **dynamic** and **interactive**.

8. HTTP CLIENT

HttpClient in Angular is used to communicate with servers and perform HTTP requests like GET, POST, PUT, DELETE.
It is part of the HttpClientModule and helps in fetching or sending data to APIs.

STEPS TO USE HTTPCLIENT

1. **Import HttpClientModule** → Add in app.module.ts.
2. **Inject HttpClient** → Use it in your service or component.
3. **Make Requests** → Use methods like get(), post(), put(), delete().

9. DEPENDENCY INJECTION

Dependency Injection (DI) in Angular is a design pattern that allows a class (like a component) to get objects it needs (services) from an external source instead of creating them itself.

- It helps in making code **reusable, testable, and maintainable**.
- Angular has a built-in **injector** that provides dependencies wherever required.
- Services are commonly injected into components using DI.

ASP.NET ARCHITECTURE

The backend of DoConnect is implemented using ASP.NET Core Web API, which follows a layered architecture to separate concerns and improve scalability, maintainability, and testability. The architecture combines the MVC pattern with Entity Framework Core (EF Core) for data access and JWT-based authentication for secure communication.

KEY LAYERS OF THE ARCHITECTURE

1. **PRESENTATION LAYER (CONTROLLERS)**
   - Controllers handle HTTP requests and responses.
   - Each controller corresponds to a domain area:
     - **AuthController** – Handles registration, login, and JWT token generation.
     - **QuestionsController** – CRUD operations for questions, image upload, search.
     - **AnswersController** – CRUD operations for answers.
     - **AdminController** – User management and moderation (approve/reject Q&A).
   - Controllers use **DTOs (Data Transfer Objects)** to shape request/response payloads.
2. **BUSINESS LOGIC LAYER (SERVICES)**
   - Encapsulates the **core logic** of the application.
   - Validates inputs, enforces business rules, and interacts with the data access layer.
   - Example: When an admin approves a question, the service updates the status and notifies relevant users.

3. **DATA ACCESS LAYER (ENTITY FRAMEWORK CORE)**
    - EF Core is used as the ORM to map C# models to SQL Server tables.
    - AppDbContext defines DbSet<User>, DbSet<Question>, DbSet<Answer>, and DbSet<Image>.
    - EF Core LINQ queries ensure secure, optimized, and strongly-typed database operations.
4. **DATABASE LAYER (SQL SERVER)**
    - Relational database stores persistent data.
    - Schema includes tables for **Users**, **Questions**, **Answers**, and **Images**.
    - Foreign key relationships enforce referential integrity (e.g., UserId in Questions links to Users).

SQL DATABASE SCHEMA

SQL DATABASE SCHEMA OVERVIEW

1. HIGH-LEVEL ER DESCRIPTION

The DoConnect database stores users, questions, answers, and optional images. Main entities and relationships:

- **Users** (1) — (N) **Questions**: a user can post many questions.
- **Users** (1) — (N) **Answers**: a user can post many answers.
- **Questions** (1) — (N) **Answers**: a question can have many answers.
- **Questions** (1) — (N) **Images**: optional images attached to a question (could also allow images for answers).

Primary keys are surrogate integer IDs. Referential integrity is enforced with foreign keys. Business rules such as approval flow (Pending → Approved/Rejected) are represented with enum-like columns.

2. CORE TABLES & COLUMNS (RECOMMENDED DATA TYPES)

USERS

- Id INT IDENTITY PRIMARY KEY
- Username NVARCHAR(100) UNIQUE NOT NULL
- Email NVARCHAR(256) UNIQUE NOT NULL
- PasswordHash VARBINARY(MAX) NOT NULL
- PasswordSalt VARBINARY(MAX) NOT NULL
- Role NVARCHAR(20) NOT NULL -- e.g., 'User' / 'Admin' (or smallint if you use enum)
- CreatedAt DATETIME2 NOT NULL DEFAULT SYSUTCDATETIME()
- IsDeleted BIT NOT NULL DEFAULT 0 *(optional for soft-delete)*

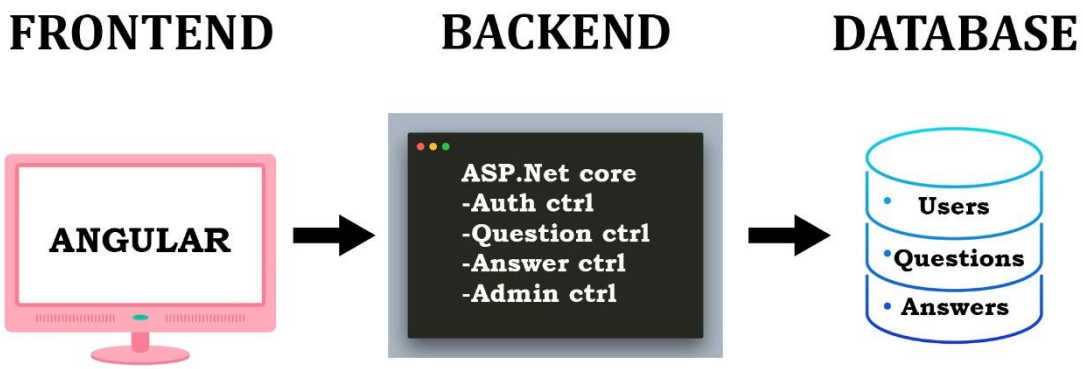
QUESTIONS

- Id INT IDENTITY PRIMARY KEY
- UserId INT NOT NULL (FK → Users.Id)
- Title NVARCHAR(300) NOT NULL
- Body NVARCHAR(MAX) NULL *(or Description depending on your model)*
- Status NVARCHAR(20) NOT NULL DEFAULT 'Pending' -- 'Pending','Approved','Rejected'
- CreatedAt DATETIME2 NOT NULL DEFAULT SYSUTCDATETIME()
- UpdatedAt DATETIME2 NULL
- IsDeleted BIT NOT NULL DEFAULT 0 *(optional)*

ANSWERS

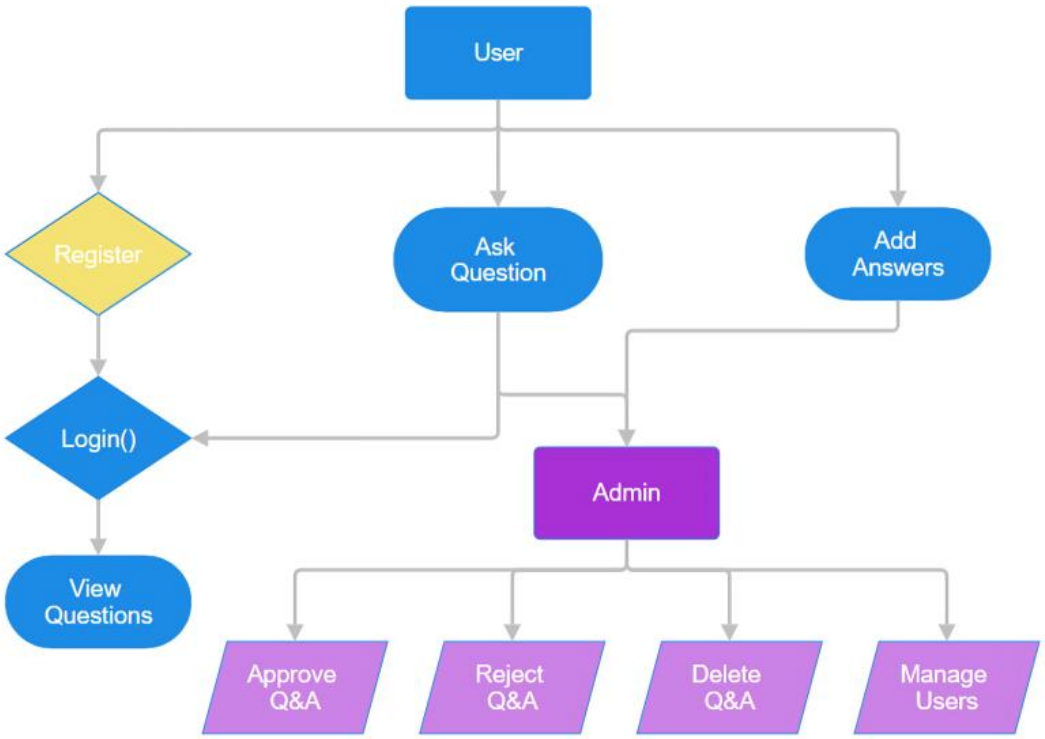
- Id INT IDENTITY PRIMARY KEY
- QuestionId INT NOT NULL (FK → Questions.Id)
- UserId INT NOT NULL (FK → Users.Id)
- Body NVARCHAR(MAX) NOT NULL
- Status NVARCHAR(20) NOT NULL DEFAULT 'Pending'
- CreatedAt DATETIME2 NOT NULL DEFAULT SYSUTCDATETIME()
- UpdatedAt DATETIME2 NULL
- IsDeleted BIT NOT NULL DEFAULT 0 *(optional)*

PROJECT OVERVIEW DIAGRAM



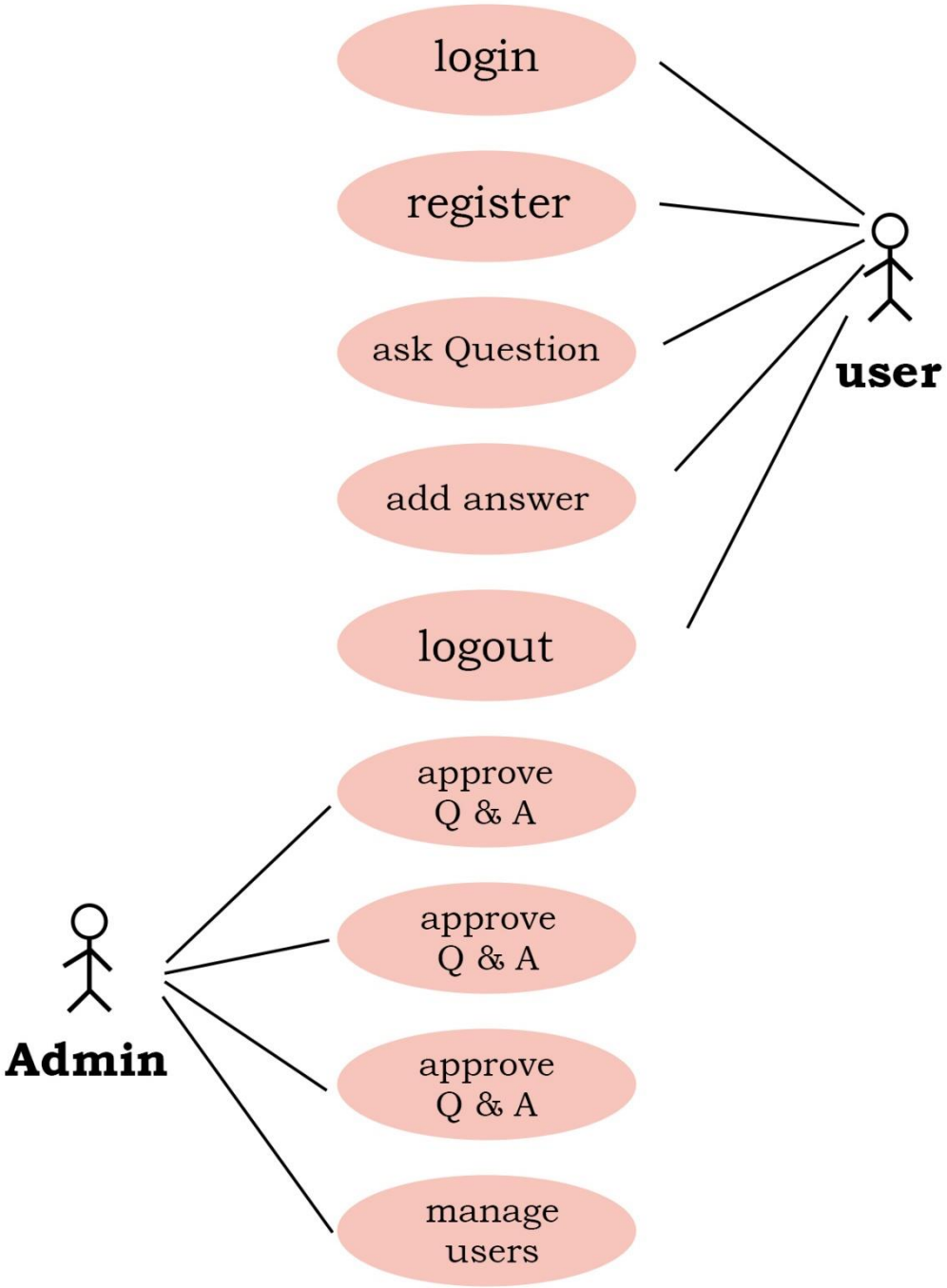
Project Overview Diagram

SYSTEM WORKFLOW DIAGRAM



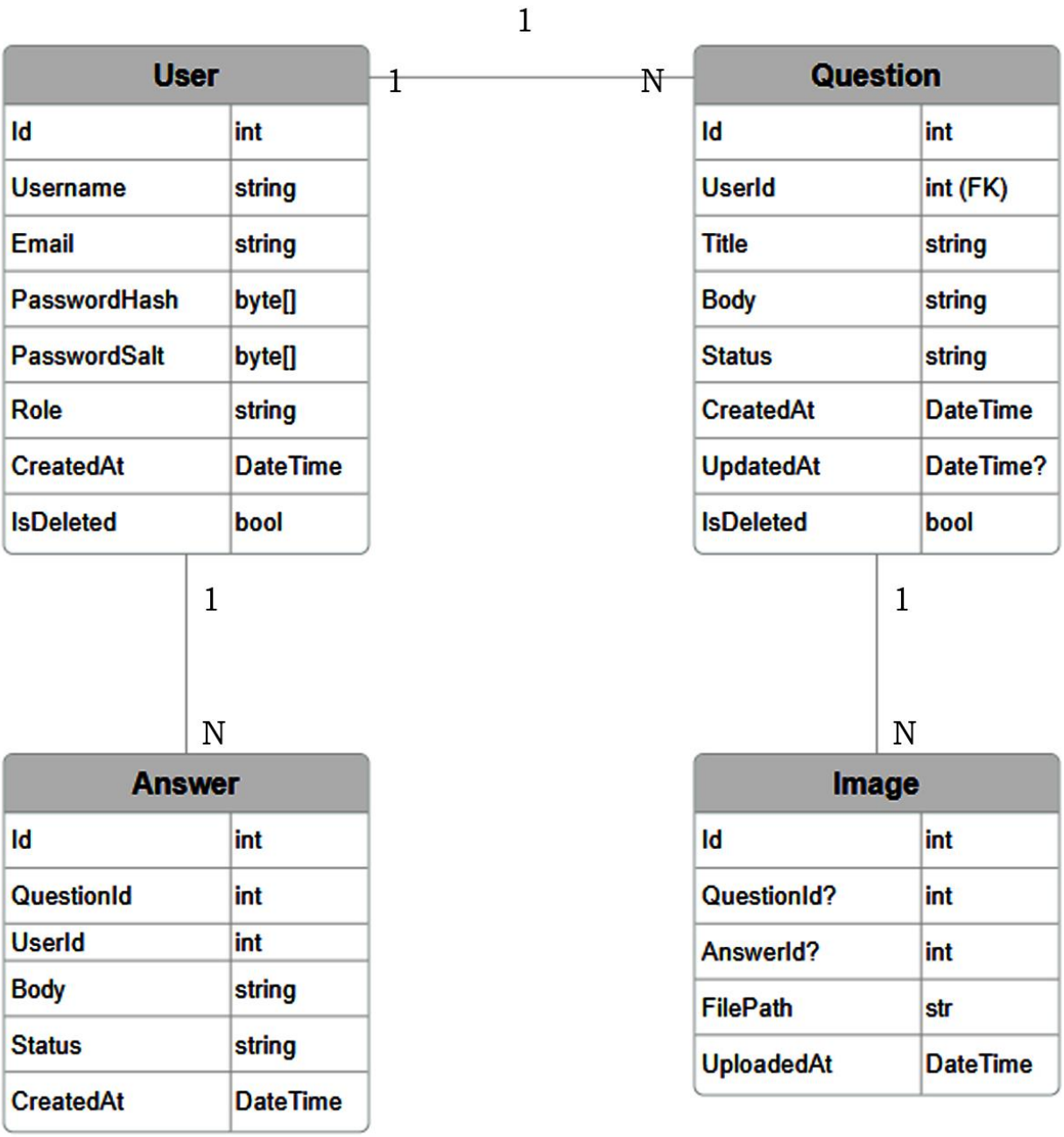
System Workflow Diagram

USE CASE DIAGRAM



Use Case Diagram

CLASS DIAGRAM



Entity Relationship Diagram

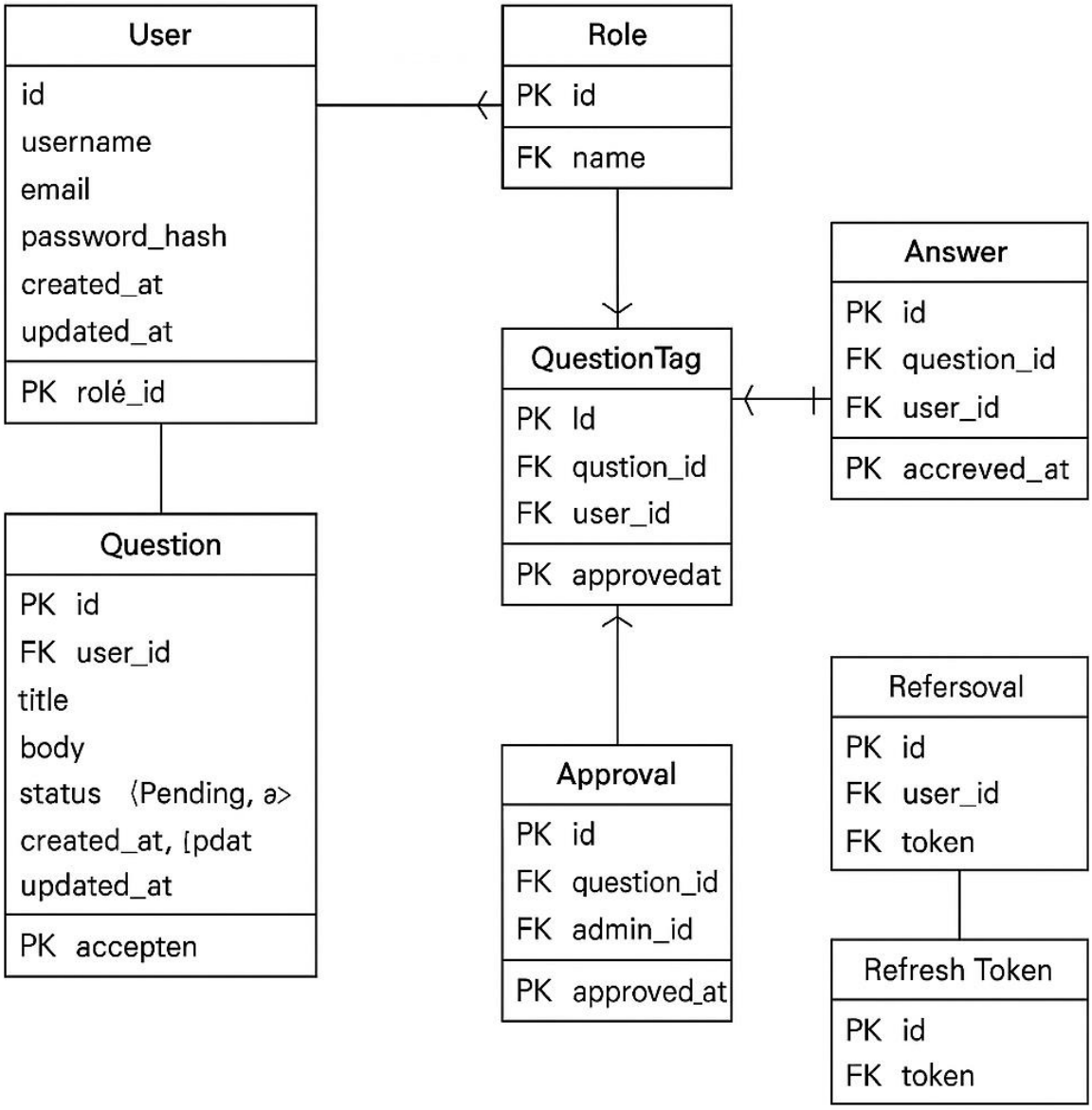
LOGIN FLOW DIAGRAM

```
          Angular SPA          AuthController/API        Database
               |                      |                      |
- Enter creds ---->|                  |                      |
               |---- POST /login ----->|                     |
               |                      |<-- SELECT user ----->|
               |                      |<-- user row (hash) --|
               |                      |--- verify password --|
               |                      |--- generate JWT -----|
               |<--- 200 OK + token ---|                     |
- logged in -------|                  |                      |
               |---- store token ------|                     |
               |    (localStorage)    |                      |
```

ASK QUESTION → ADMIN FLOW DIAGRAM

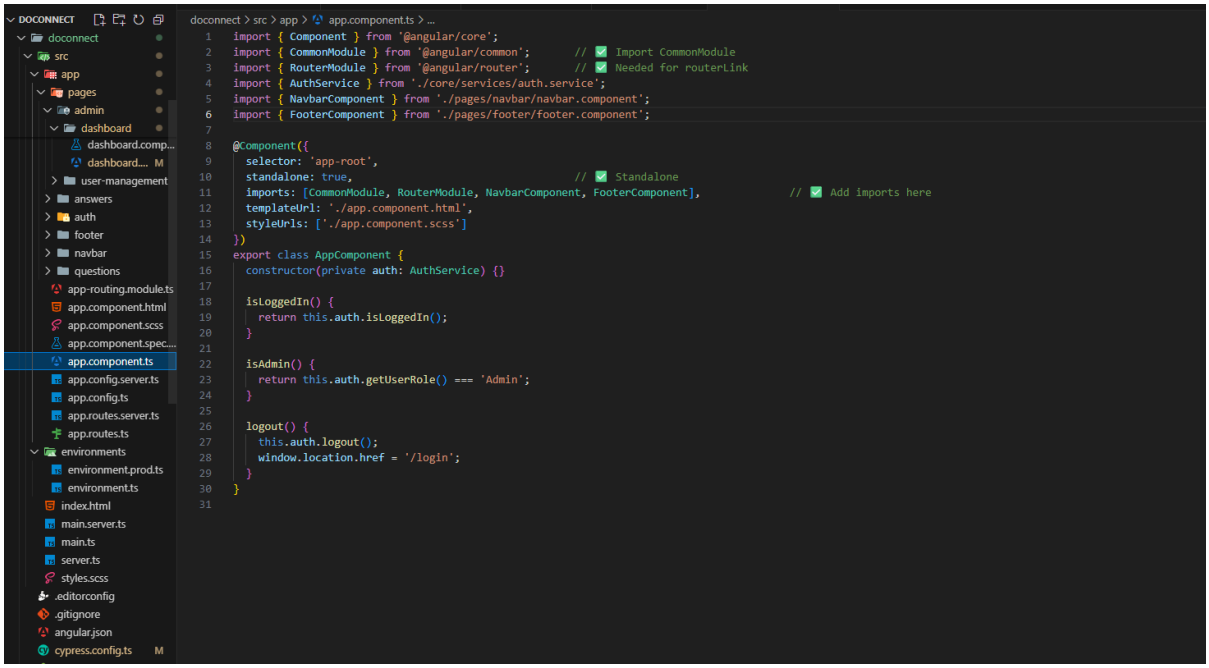
```
User              Angular SPA            QuestionsController/API        Database
 |                    |                        |                          |
 |--- fills form ----->|                        |                          |
 |                    |--- POST /questions ->||                          |
 |                    |                        |--- INSERT Question ---|
 |                    |                        |   (Status=Pending)    |
 |                    |<-- 201 Created -------|                          |
 |<-- sees 'Pending' --|                        |                          |

                      (Later...)
Admin             Admin Angular UI        AdminController/API          Database
 |                    |                        |                          |
 |--- GET pending -->|                        |--- SELECT Questions ---|
 |                    |                        |   WHERE Status=Pending |
 |                    |<-- pending list ------|<------------------------|
 |--- Approve Q ------|                        |    UPDATE Question -->|
 |                    |                        |   Status=Approved     |
 |                    |<-- 204 No Content ----|                          |
 |<-- success --------|                        |                          |

User/All
 |                    |                        |                          |
 |--- view questions ------------------------->|                          |
 |                    |                        |                          |
 |<— sees question --------------------------|                          |
```

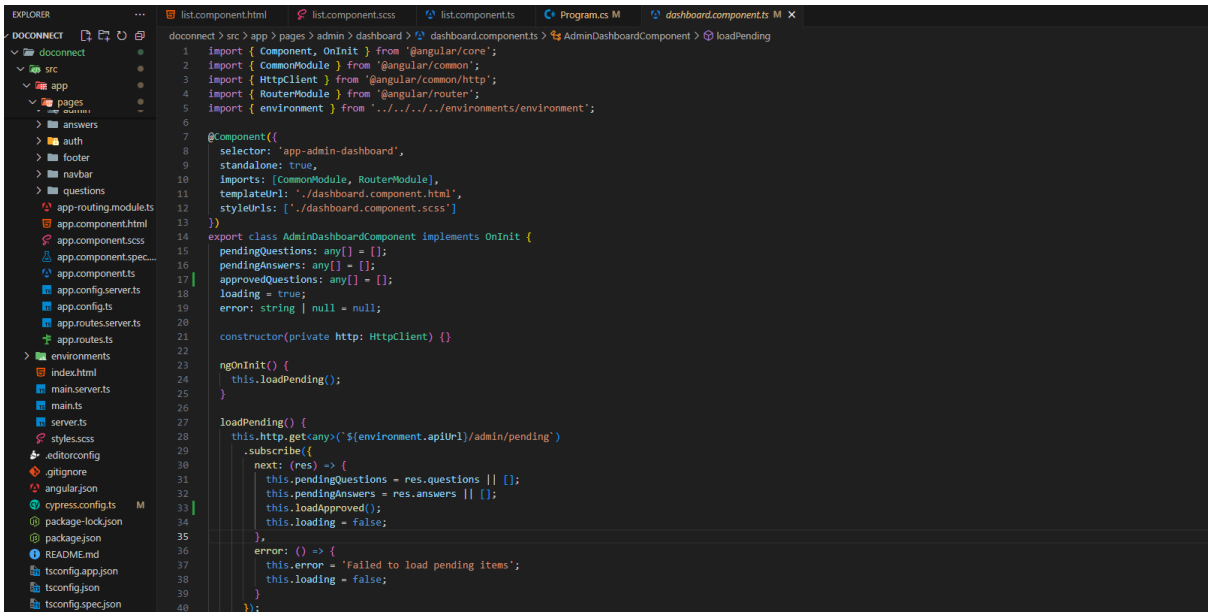
ER DIAGRAM


User
id
username
email
password_hash
created_at
updated_at
PK rolé_id
Role
PK id
FK name
Answer
PK id
FK question_id
FK user_id
PK accreved_at
QuestionTag
PK Id
FK qustion_id
FK user_id
PK approvedat
Question
PK id
FK user_id
title
body
status ⟨Pending, a>
created_at, [pdat
updated_at
PK accepten
Approval
PK id
FK question_id
FK admin_id
PK approved_at
Refersoval
PK id
FK user_id
FK token
Refresh Token
PK id
FK token

FRONTEND CODE STRUCTURE & SCREENSHOT

```
1  src/
2  ├── app/
3  │   ├── core/
4  │   │   ├── guards/
5  │   │   │   └── auth.guard.ts
6  │   │   ├── interceptors/
7  │   │   │   └── auth.interceptor.ts
8  │   │   ├── models/
9  │   │   │   ├── user.model.ts
10 │   │   │   ├── question.model.ts
11 │   │   │   └── answer.model.ts
12 │   │   └── services/
13 │   │       ├── auth.service.ts
14 │   │       ├── question.service.ts
15 │   │       ├── answer.service.ts
16 │   │       └── image.service.ts
17 │   │
18 │   ├── pages/
19 │   │   ├── auth/
20 │   │   │   ├── login/
21 │   │   │   │   ├── login.component.html
22 │   │   │   │   ├── login.component.scss
23 │   │   │   │   └── login.component.ts
24 │   │   │   └── register/
25 │   │   │       ├── register.component.html
26 │   │   │       ├── register.component.scss
27 │   │   │       └── register.component.ts
28 │   │   │
29 │   │   ├── questions/
30 │   │   │   ├── list/
31 │   │   │   │   ├── list.component.html
32 │   │   │   │   ├── list.component.scss
33 │   │   │   │   └── list.component.ts
34 │   │   │   ├── detail/
35 │   │   │   │   ├── detail.component.html
36 │   │   │   │   ├── detail.component.scss
37 │   │   │   │   └── detail.component.ts
38 │   │   │   └── ask/
39 │   │   │       ├── ask.component.html
40 │   │   │       ├── ask.component.scss
41 │   │   │       └── ask.component.ts
42 │   │   │
43 │   │   ├── answers/
44 │   │   │   └── answer-form/
45 │   │   │       ├── answer-form.component.html
46 │   │   │       ├── answer-form.component.scss
47 │   │   │       └── answer-form.component.ts
48 │   │   │
49 │   │   └── admin/
50 │   │       ├── admin.component.html
51 │   │       ├── admin.component.scss
52 │   │       └── admin.component.ts
53 │   │
54 │   ├── shared/
55 │   │   ├── navbar/
56 │   │   │   ├── navbar.component.html
57 │   │   │   ├── navbar.component.scss
58 │   │   │   └── navbar.component.ts
59 │   │   └── footer/
60 │   │       ├── footer.component.html
61 │   │       ├── footer.component.scss
62 │   │       └── footer.component.ts
63 │   │
64 │   ├── app.routes.ts
65 │   └── app.component.ts
66 │
67 ├── assets/
68 │   └── images/
69 │
70 ├── environments/
71 │   ├── environment.ts
72 │   └── environment.development.ts
73 │
74 ├── index.html
75 ├── main.ts
76 └── styles.scss
```


DOCONNECT
doconnect
src
app
pages
admin
dashboard
dashboard.comp...
dashboard.... M
user-management
answers
auth
footer
navbar
questions
app-routing.module.ts
app.component.html
app.component.scss
app.component.spec....
app.component.ts
app.config.server.ts
app.config.ts
app.routes.server.ts
app.routes.ts
environments
environment.prod.ts
environment.ts
index.html
main.server.ts
main.ts
server.ts
styles.scss
.editorconfig
.gitignore
angular.json
cypress.config.ts M
doconnect > src > app > app.component.ts > ...
import { Component } from '@angular/core';
import { CommonModule } from '@angular/common'; // ✅ Import CommonModule
import { RouterModule } from '@angular/router'; // ✅ Needed for routerLink
import { AuthService } from './core/services/auth.service';
import { NavbarComponent } from './pages/navbar/navbar.component';
import { FooterComponent } from './pages/footer/footer.component';
@Component({
selector: 'app-root',
standalone: true, // ✅ Standalone
imports: [CommonModule, RouterModule, NavbarComponent, FooterComponent], // ✅ Add imports here
templateUrl: './app.component.html',
styleUrls: ['./app.component.scss']
})
export class AppComponent {
constructor(private auth: AuthService) {}
isLoggedIn() {
return this.auth.isLoggedIn();
}
isAdmin() {
return this.auth.getUserRole() === 'Admin';
}
logout() {
this.auth.logout();
window.location.href = '/login';
}
}


EXPLORER
list.component.html
list.component.scss
list.component.ts
Program.cs M
dashboard.component.ts M
DOCONNECT
doconnect
src
app
pages
answers
auth
footer
navbar
questions
app-routing.module.ts
app.component.html
app.component.scss
app.component.spec....
app.component.ts
app.config.server.ts
app.config.ts
app.routes.server.ts
app.routes.ts
environments
index.html
main.server.ts
main.ts
server.ts
styles.scss
.editorconfig
.gitignore
angular.json
cypress.config.ts M
package-lock.json
package.json
README.md
tsconfig.app.json
tsconfig.json
tsconfig.spec.json
doconnect > src > app > pages > admin > dashboard > dashboard.component.ts > AdminDashboardComponent > loadPending
import { Component, OnInit } from '@angular/core';
import { CommonModule } from '@angular/common';
import { HttpClient } from '@angular/common/http';
import { RouterModule } from '@angular/router';
import { environment } from '../../../environments/environment';
@Component({
selector: 'app-admin-dashboard',
standalone: true,
imports: [CommonModule, RouterModule],
templateUrl: './dashboard.component.html',
styleUrls: ['./dashboard.component.scss']
})
export class AdminDashboardComponent implements OnInit {
pendingQuestions: any[] = [];
pendingAnswers: any[] = [];
approvedQuestions: any[] = [];
loading = true;
error: string | null = null;
constructor(private http: HttpClient) {}
ngOnInit() {
this.loadPending();
}
loadPending() {
this.http.get<any>(`${environment.apiUrl}/admin/pending`)
.subscribe({
next: (res) => {
this.pendingQuestions = res.questions || [];
this.pendingAnswers = res.answers || [];
this.loadApproved();
this.loading = false;
},
error: () => {
this.error = 'Failed to load pending items';
this.loading = false;
}
});

BACKEND CODE STRUCTURE & SCREENSHOT

```
DoConnect/
├── DoConnect.Api/
│   ├── Controllers/
│   │   ├── AdminController.cs
│   │   ├── AuthController.cs
│   │   ├── QuestionsController.cs
│   │   ├── AnswersController.cs
│   │   └── ImagesController.cs
│   │
│   ├── Data/
│   │   └── AppDbContext.cs
│   │
│   ├── Dtos/
│   │   ├── Auth/
│   │   │   ├── LoginDto.cs
│   │   │   ├── RegisterDto.cs
│   │   │   └── AuthResponseDto.cs
│   │   ├── Questions/
│   │   │   ├── QuestionDto.cs
│   │   │   ├── CreateQuestionDto.cs
│   │   │   └── AnswerDto.cs
│   │   └── Shared/
│   │       └── PagedResultDto.cs
│   │
│   ├── Entities/
│   │   ├── User.cs
│   │   ├── Question.cs
│   │   ├── Answer.cs
│   │   └── Image.cs
│   │
│   ├── Migrations/
│   │   └── {timestamp}_InitialMig.cs
│   │
│   ├── Services/
│   │   ├── JwtService.cs
│   │   ├── IJwtService.cs
│   │   ├── QuestionService.cs
│   │   └── AnswerService.cs
│   │
│   ├── Properties/
│   │   └── launchSettings.json
│   │
│   ├── Program.cs
│   ├── appsettings.json
│   ├── appsettings.Development.json
│   └── appsettings.Testing.json
│
├── DoConnect.Api.Tests/
│   ├── Integration/
│   │   ├── AuthControllerTests.cs
│   │   ├── AdminControllerTests.cs
│   │   └── QuestionControllerTests.cs
│   │
│   ├── Unit/
│   │   └── JwtServiceTests.cs
│   │
│   ├── CustomWebApplicationFactory.cs
│   ├── TestHelpers.cs
│   ├── AssemblyInfo.cs
│   └── DoConnect.Api.Tests.csproj
│
├── DoConnect.sln
└── README.md
```

EXPLORER

DOCONNECT

- DoConnect.Api
  - bin
  - Controllers
    - AdminController.... M
    - AnswerController.cs
    - AuthController.cs
    - QuestionController.cs
  - Data
  - Dtos
  - Entities
  - Migrations
  - obj
  - Properties
  - Services
  - wwwroot
  - appsettings.Developme...
  - appsettings.json
  - DoConnect.Api.csproj
  - DoConnect.Api.http
  - Program.cs M
  - schema.sql
- DoConnect.Api.Tests
  - bin
  - Integration
    - AdminControllerT... U
    - AuthControllerTe... U
  - obj
  - Unit
    - JwtServiceTests.cs U
    - AssemblyInfo.cs U
    - CustomWebApplic... U
    - DoConnect.Api.Tes... U
    - TestHelpers.cs U
  - Database&Config.zip U
  - DoConnect.Api.Tests... U
  - DoConnect.sln M
  - doconnect.zip U
  - README.md

Program.cs M

DoConnect.Api > Program.cs > ...

```
using DoConnect.Api.Data;
using DoConnect.Api.Entities;
using DoConnect.Api.Services;
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.EntityFrameworkCore;
using Microsoft.IdentityModel.Tokens;
using System.Security.Cryptography;
using System.Security.Claims;
using Microsoft.OpenApi.Models;


using System.Text;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddCors(options =>
{
    options.AddPolicy("AllowLocal", b => b
        .AllowAnyHeader()
        .AllowAnyMethod()
        .WithOrigins("http://localhost:4200", "https://localhost:4200"));
});

builder.Services.Configure<JwtOptions>(builder.Configuration.GetSection("Jwt"));
builder.Services.AddScoped<IJwtService, JwtService>();

// if (builder.Environment.IsEnvironment("Testing"))
// {
//     builder.Services.AddDbContext<AppDbContext>(opt =>
//     opt.UseSqlServer(builder.Configuration.GetConnectionString("Default")));
// }
// else
// {

// }
builder.Services.AddDbContext<AppDbContext>(opt =>
    opt.UseSqlServer(builder.Configuration.GetConnectionString("Default")));

builder.Services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
    .AddJwtBearer(opt =>
    {
        var jwt = builder.Configuration.GetSection("Jwt");
        opt.TokenValidationParameters = new TokenValidationParameters
        {
            ValidateIssuer = true,
```

EXPLORER

DOCONNECT

- doconnect
  - src
    - server.ts
    - styles.scss
  - .editorconfig
  - .gitignore
  - angular.json
  - cypress.config.ts M
  - package-lock.json
  - package.json
  - README.md
  - tsconfig.app.json
  - tsconfig.json
  - tsconfig.spec.json
- DoConnect.Api
  - bin
  - Controllers
    - AdminController.... M
    - AnswerController.cs
    - AuthController.cs
    - QuestionController.cs
  - Data
  - Dtos
  - Entities
  - Migrations
  - obj
  - Properties
  - Services
  - wwwroot
  - appsettings.Developme...
  - appsettings.json
  - DoConnect.Api.csproj
  - DoConnect.Api.http
  - Program.cs M
  - schema.sql
- DoConnect.Api.Tests
  - bin
  - Integration
    - AdminControllerT... U

OUTLINE

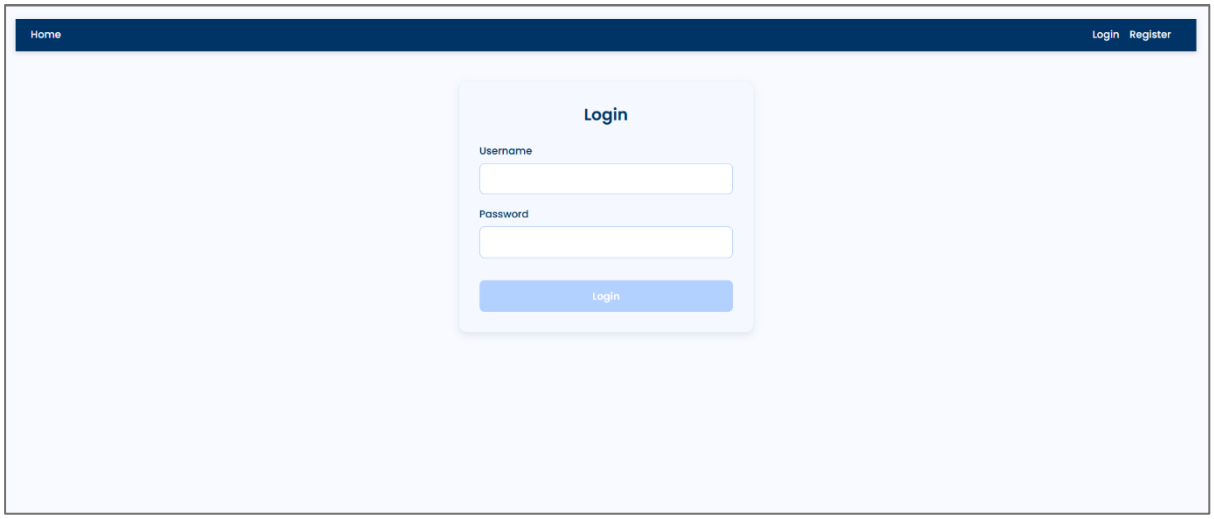
AdminController.cs M

DoConnect.Api > Controllers > AdminController.cs > AdminController > AdminController

```
using DoConnect.Api.Data;
using DoConnect.Api.Entities;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using System.Security.Cryptography;
using System.Text;

namespace DoConnect.Api.Controllers;

[ApiController]
[Route("api/[controller]")]
[Authorize(Roles = "Admin")]
0 references
public class AdminController(AppDbContext db) : ControllerBase
{
    // -------------------------
    // 📌 USER MANAGEMENT (CRUD)
    // -------------------------

    // ✅ Get all users
    [HttpGet("users")]
    0 references
    public async Task<ActionResult<IEnumerable<User>>> GetUsers()
    {
        return await db.Users.ToListAsync();
    }

    // ✅ Get single user
    [HttpGet("users/{id:int}")]
    1 reference
    public async Task<ActionResult<User>> GetUser(int id)
    {
        var user = await db.Users.FindAsync(id);
        return user is null ? NotFound() : user;
    }

    // ✅ Add new user (Admin can create a user account)
    [HttpPost("users")]
    0 references
    public async Task<ActionResult<User>> AddUser(string username, string password, string email, bool isAdmin = false)
    {
        if (await db.Users.AnyAsync(u => u.Username == username))
            return BadRequest("Username already exists");

        CreatePasswordHash(password, out var hash, out var salt);
```

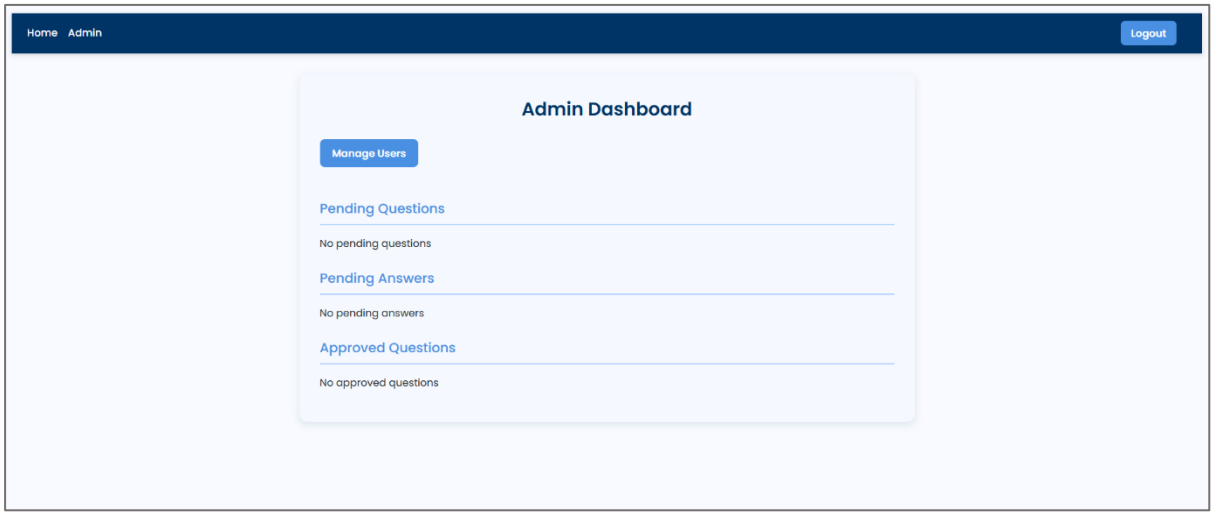
SCREENSHOTS

LOGIN PAGE

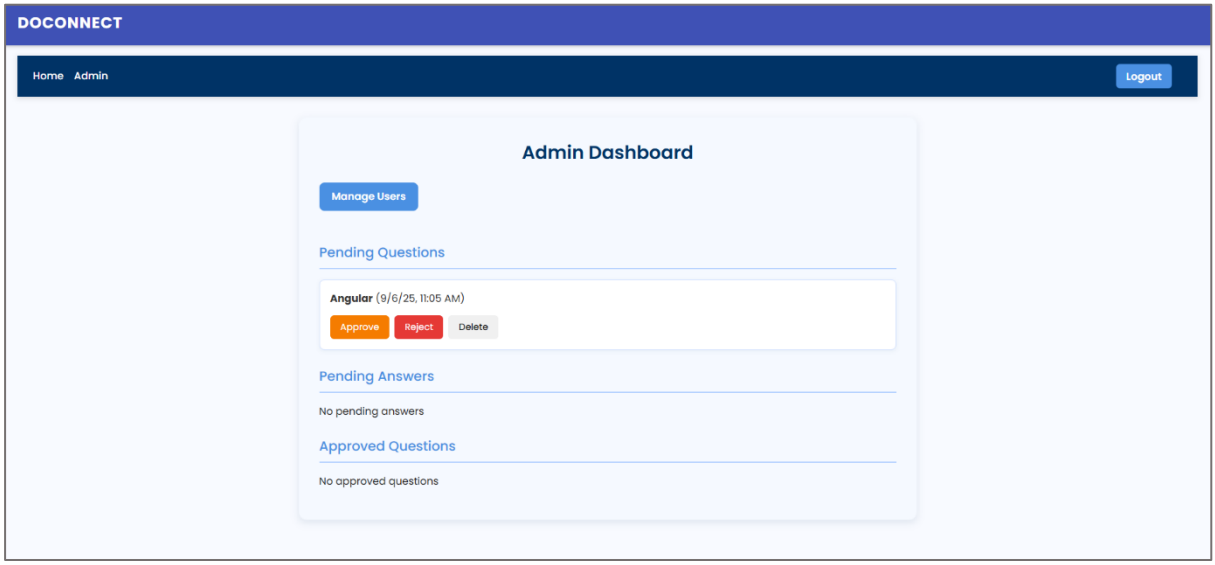


ADMIN

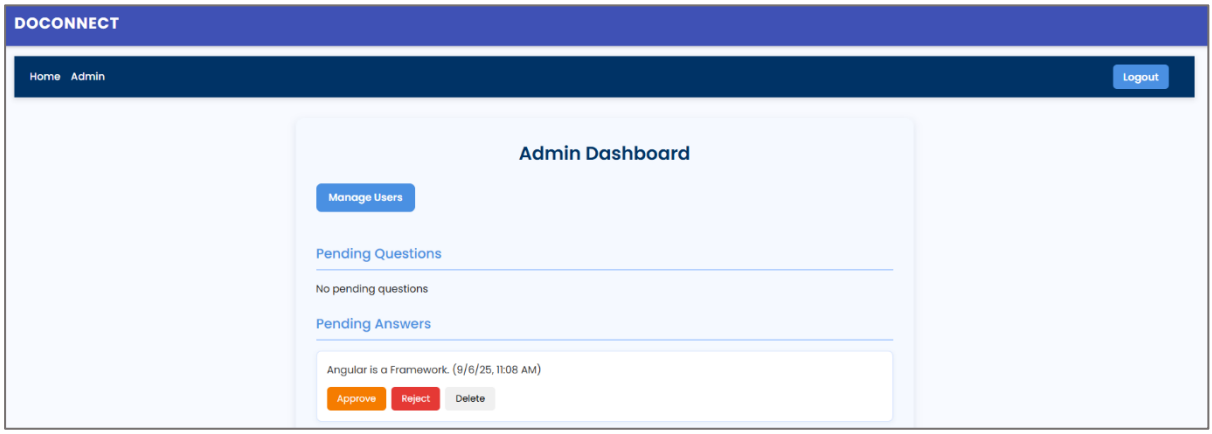
DASHBOARD



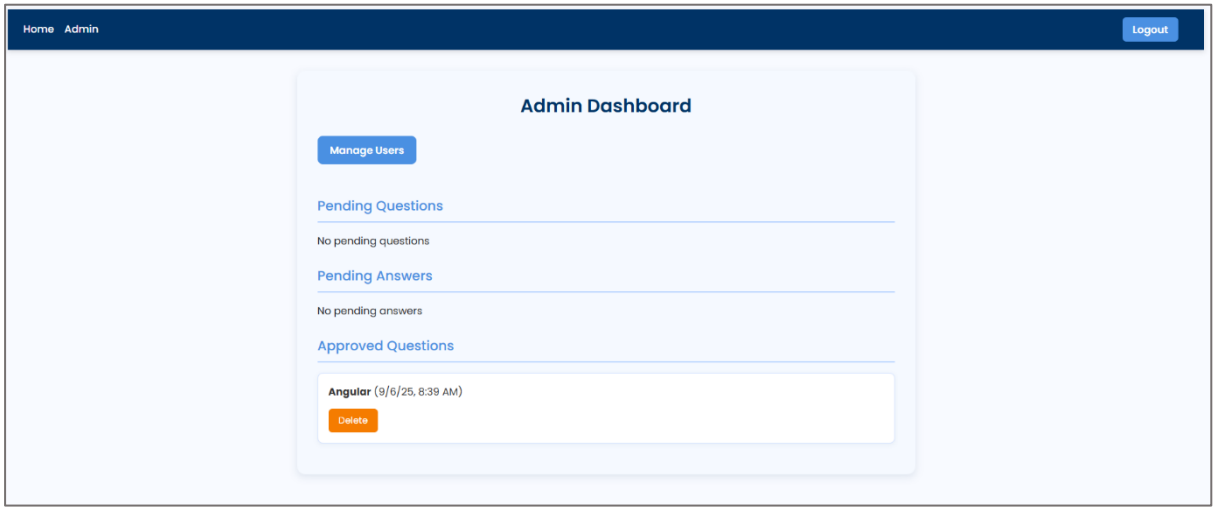
APPROVE / REJECT QUESTION



APPROVE / REJECT ANSWER

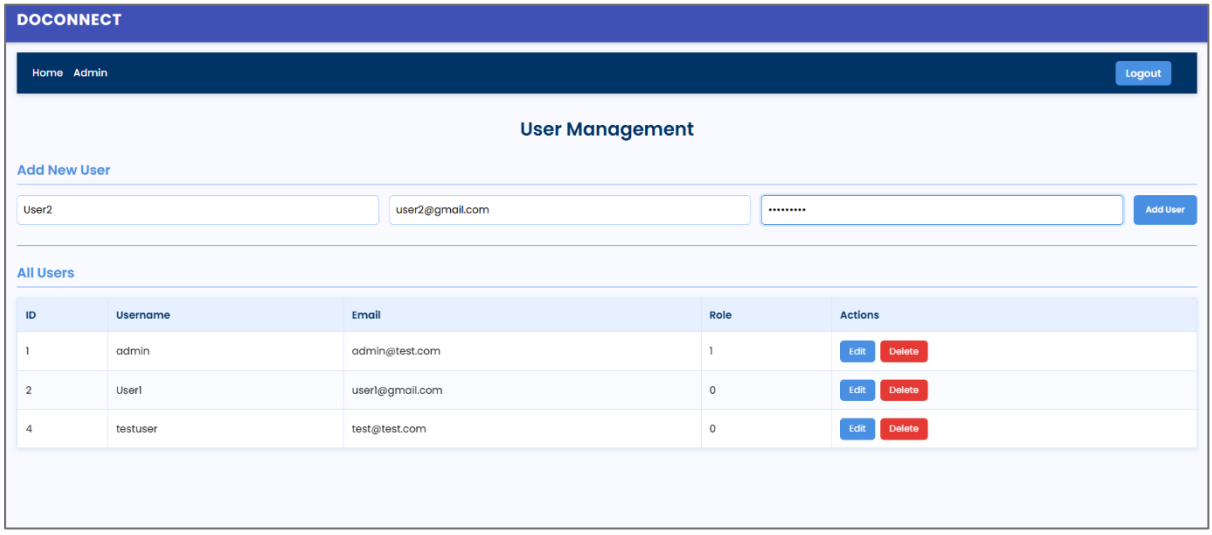


DELETE QUESTION

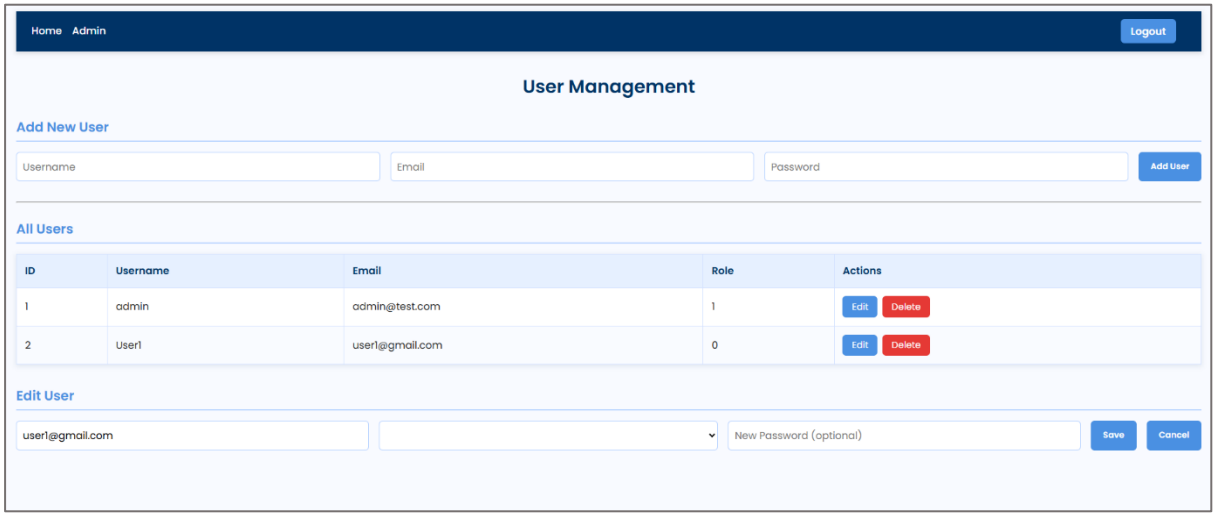


ADMIN – USER MANAGEMENT

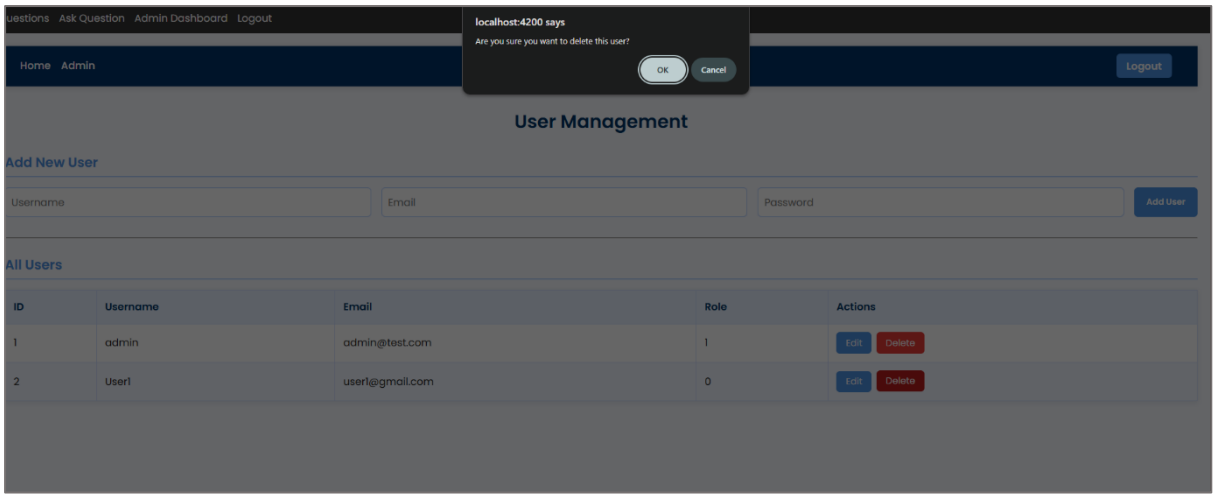
ADD USER



UPDATE USER

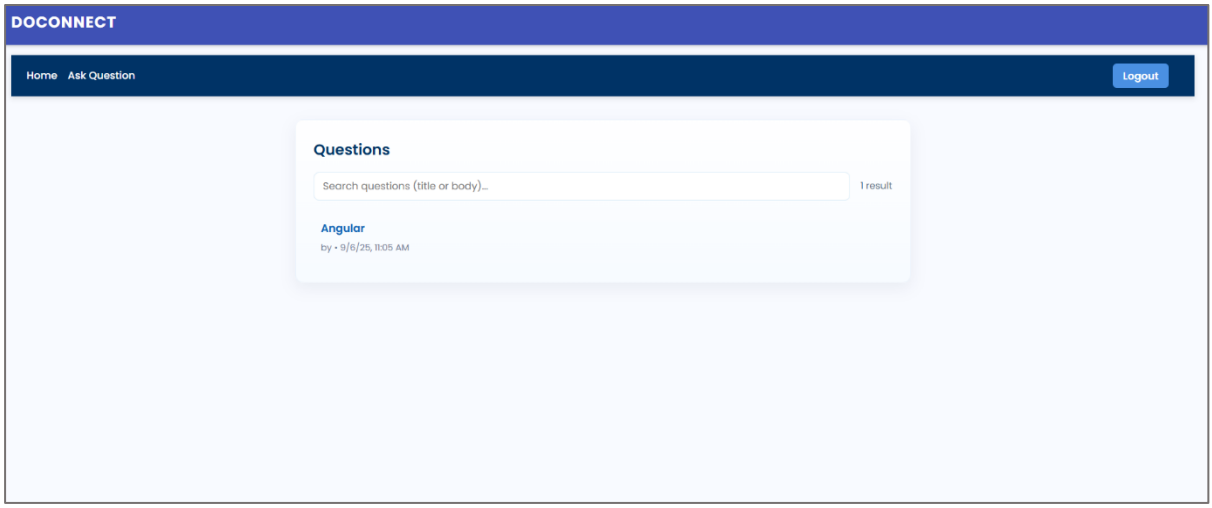


DELETE USER

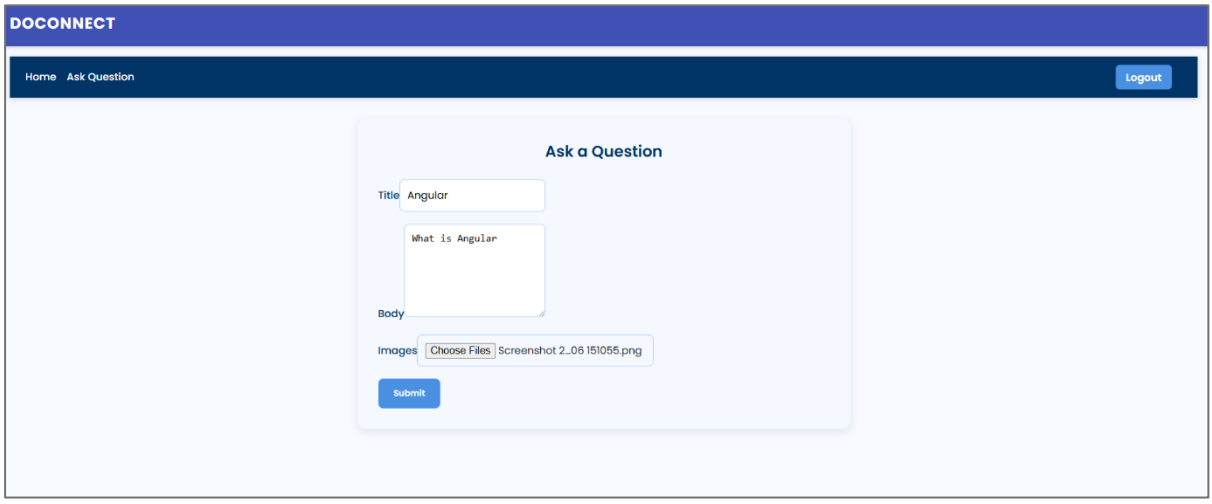


USER

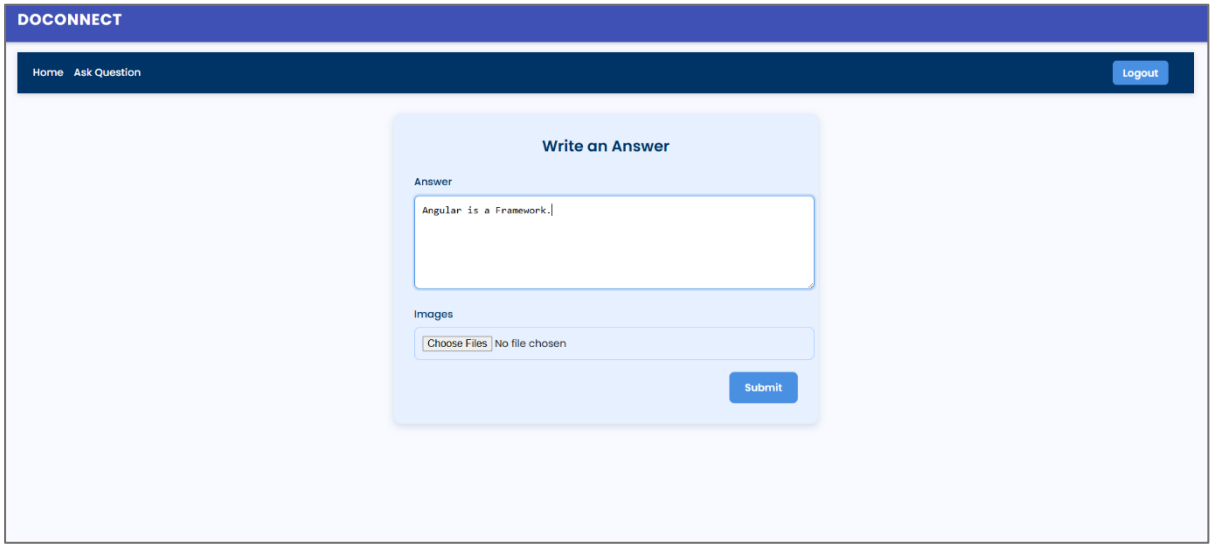
USER DASHBOARD



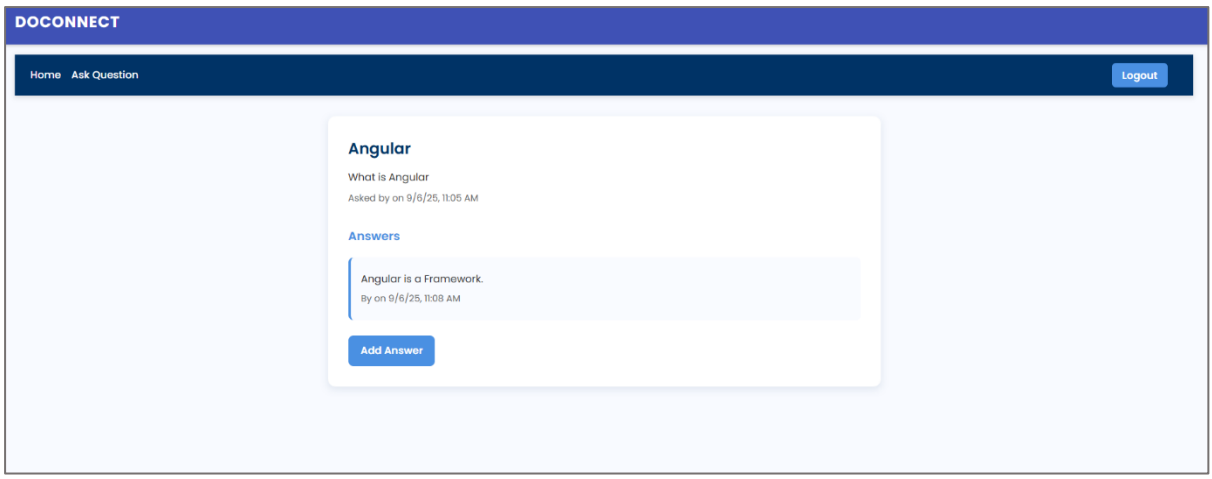
ASK QUESTION



ADD ANSWER



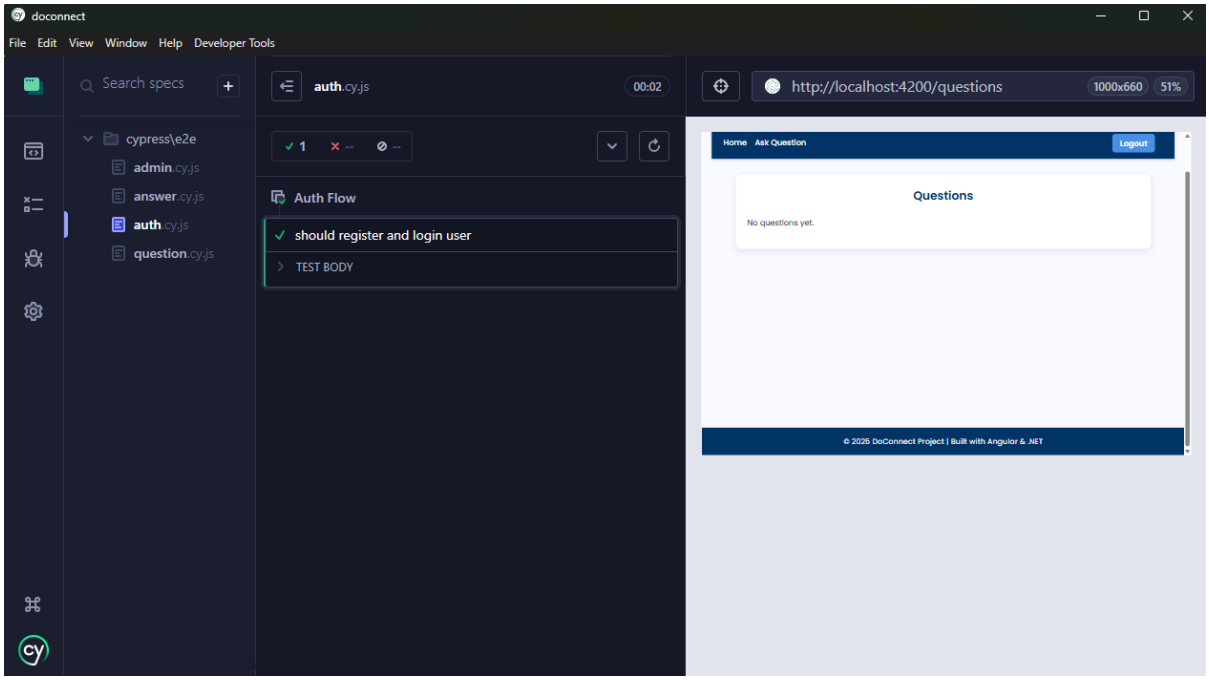
QUESTION DETAILS



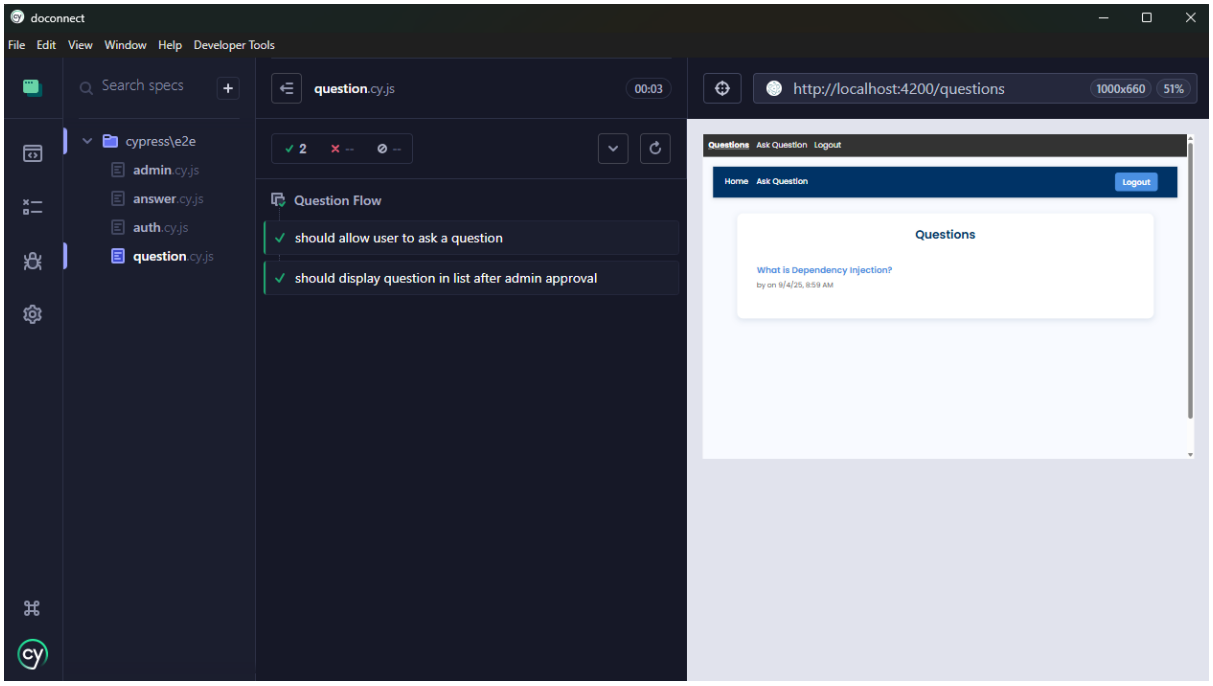
TESTING

FRONTEND – CYPRESS

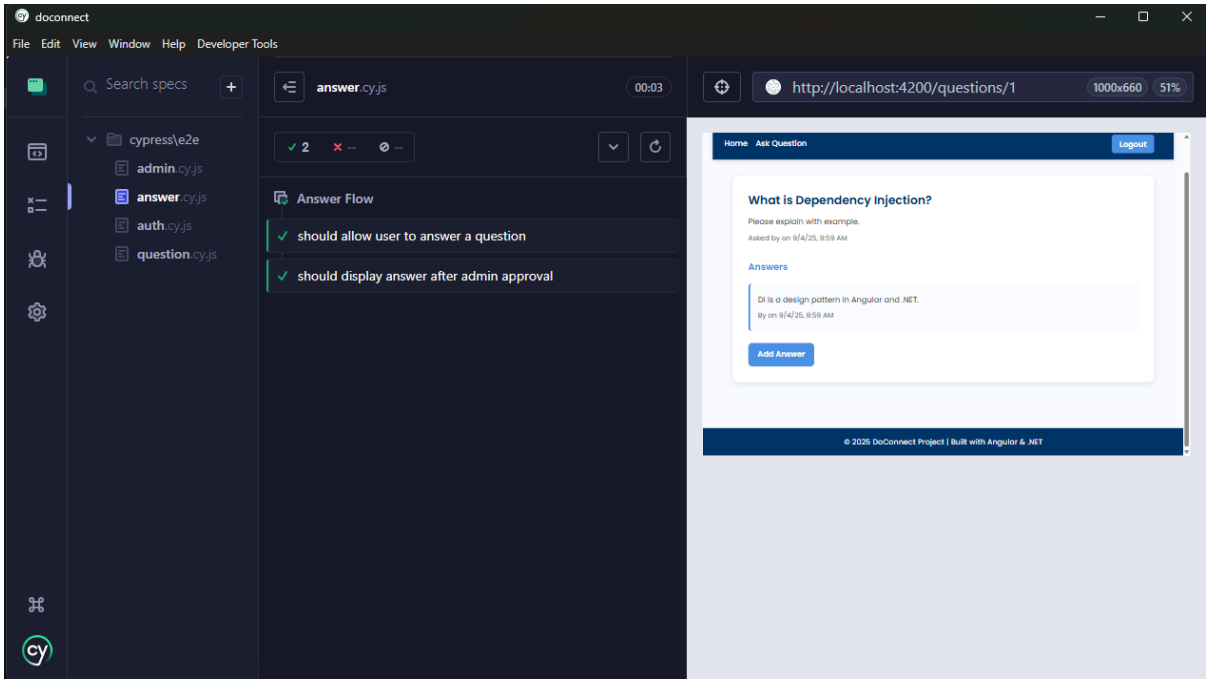
AUTHENTICATION



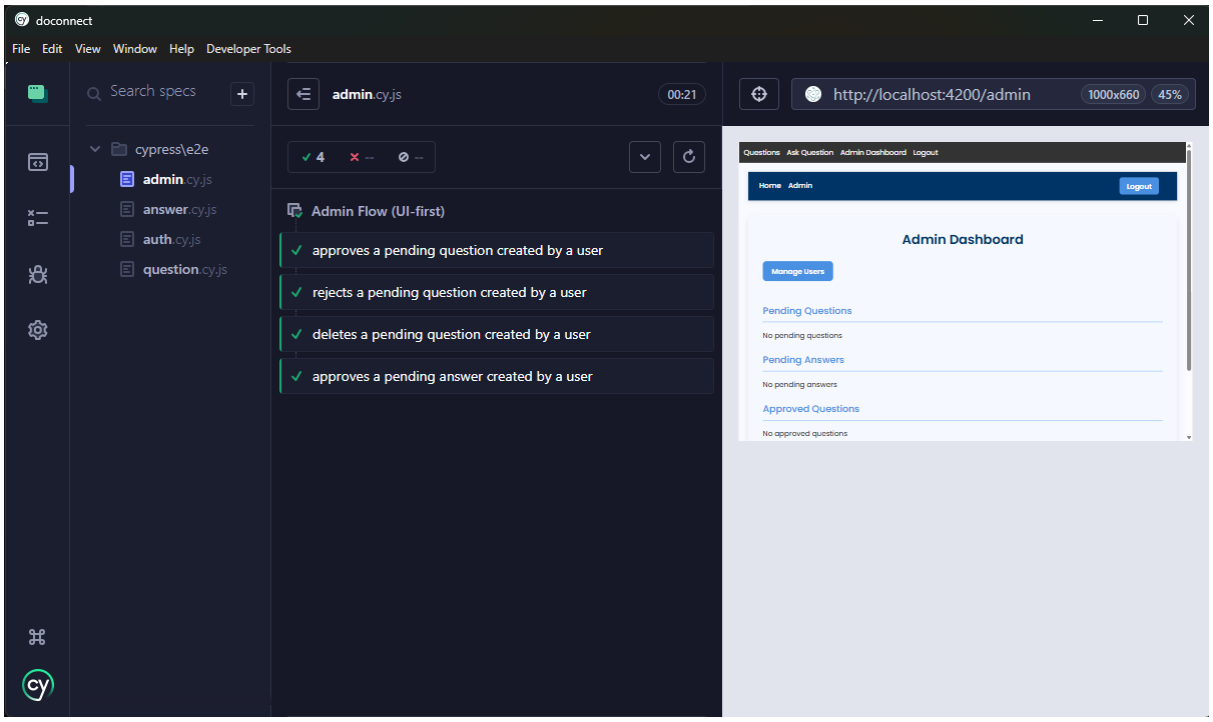
QUESTION



ANSWER



ADMIN



BACKEND – X-UNIT

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS    QUERY RESULTS
      SELECT "q"."Id", "q"."Body", "q"."CreatedAt", "q"."Status", "q"."Title", "q"."UserId"
      FROM "Questions" AS "q"
      WHERE "q"."Id" = @__p_0
      LIMIT 1
[xUnit.net 00:00:15.61]   Finished:    DoConnect.Api.Tests
  DoConnect.Api.Tests test succeeded (20.9s)

Test summary: total: 4, failed: 0, succeeded: 4, skipped: 0, duration: 20.9s
Build succeeded in 35.1s

Workload updates are available. Run `dotnet workload list` for more information.

murug@SARAVANAN MINGW64 /d/DoConnect (main)
$
```

CONCLUSION

The DoConnect Project successfully implements a Q&A discussion platform where users can post questions, provide answers, and interact in a structured environment.

Frontend (Angular) provides a clean, responsive, and user-friendly interface with features like:

- User login/registration
- Asking questions and submitting answers
- Admin panel for approving/rejecting questions and answers
- Search and filter options for better usability

Backend (ASP.NET Core Web API + EF Core + SQL Server) ensures:

- Secure authentication/authorization with JWT
- Proper entity relationships between Users, Questions, Answers, and Images
- Role-based access control for Admin and Users
- RESTful APIs supporting CRUD operations and approval workflows

Testing was covered at two levels:

Frontend (Cypress E2E Tests): Validated real user flows like login, asking/answering questions, and admin approvals.

Backend (xUnit Integration & Unit Tests): Ensured controllers, services, and authentication logic function correctly in isolation and integration.

Database design was handled through Entity Framework migrations, with an ER diagram that defines relationships clearly and prevents inconsistencies.