Project Report
On

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

Name : UTSAV DUBEY

Batch Name : WIPRO NGA - .Net Full Stack Angular - FY26 - C3

Instructor : Mr. Ramesh Nediyadath

Table of Contents

➢ Problem Definition and Objectives &Goals

The DoConnect application is designed to provide a collaborative Q&A platform where technical enthusiasts, learners, and professionals can interact. The major problem it solves is the lack of a structured, moderated platform where validated technical discussions take place. Unlike generic forums, DoConnect introduces a moderation layer through admins to ensure authenticity and correctness of the content, preventing the spread of misinformation and maintaining high-quality, trustworthy technical knowledge sharing.

Objectives:

Enable Secure Interaction: Provide a secure platform for users to ask and answer technical questions easily, backed by JWT authentication and role-based access control.

Ensure Content Quality: Implement a robust admin approval workflow to vet all questions and answers before they become publicly visible, guaranteeing content reliability.

Enhance User Experience: Offer multimedia support through integrated image uploads and a powerful search functionality to navigate the growing knowledge base.

Promote Real-Time Engagement: Implement a real-time notification system (SignalR) to keep admins instantly informed of new content requiring moderation, speeding up the approval process.

Project Goals:

The primary goal of DoConnect is to create a reliable and interactive Q&A platform specifically targeted for technical knowledge sharing. The system aims to:

Deliver Intuitive User Interfaces: Build a responsive, component-based Angular frontend for both users and admins, facilitating seamless posting and moderation of content.

Construct a Scalable Backend: Develop a robust, secure, and well-documented ASP.NET Core Web API using EF Core for data access and SQL Server for persistent, efficient data storage.

Establish Effective Moderation: Provide admins with the tools necessary to maintain the platform's integrity through an efficient approval workflow and content moderation capabilities.

Implement Real-Time Features: Enhance interactivity and responsiveness with optional SignalR integration for real-time admin notifications.

Ensure Professional Standards: Deliver a fully tested, well-documented, and deployable application, complete with Swagger UI for API exploration

➢ Fronted & Backend Architecture

The system follows a client-server architecture. The Angular-based frontend provides responsive UI interactions, routing, and component-driven design. The ASP.NET Core backend exposes RESTful APIs, implements business logic, and manages authentication using JWT. The SQL Server database ensures persistent data storage. SignalR is optionally integrated for real-time communication for admin notifications.

The architecture is visualized through the following database schema, which underpins the entire application:

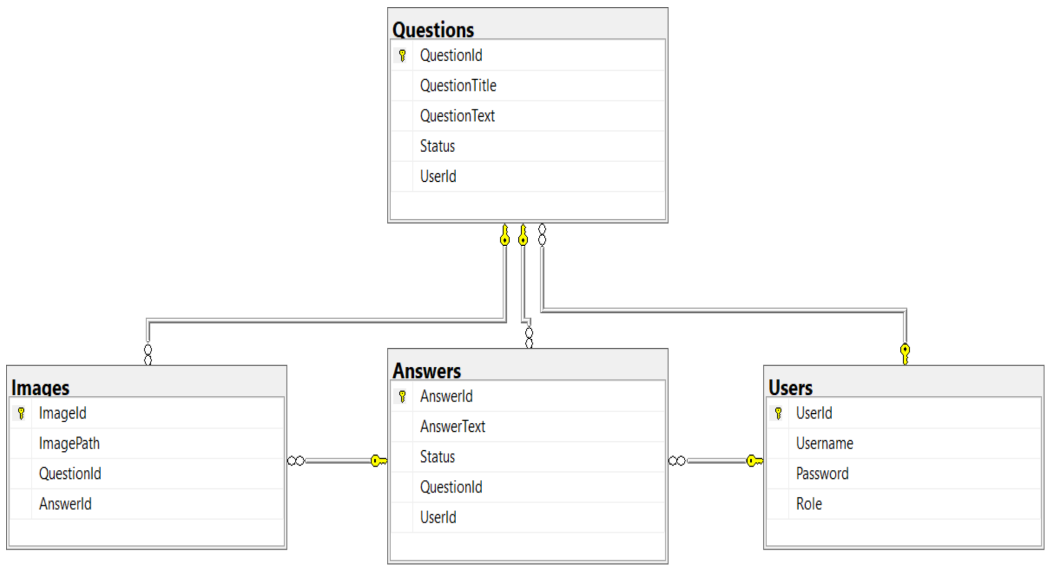


Figure 1: Entity-Relationship Diagram (ERD) for the DoConnect Database

Backend Technology Stack:

ASP.NET Core MVC, Entity Framework Core, JWT Bearer Authentication, Swashbuckle (Swagger), SignalR (optional).

Frontend Technology Stack: Angular, Angular Router, RxJS, Axios/Fetch API for HTTP requests.

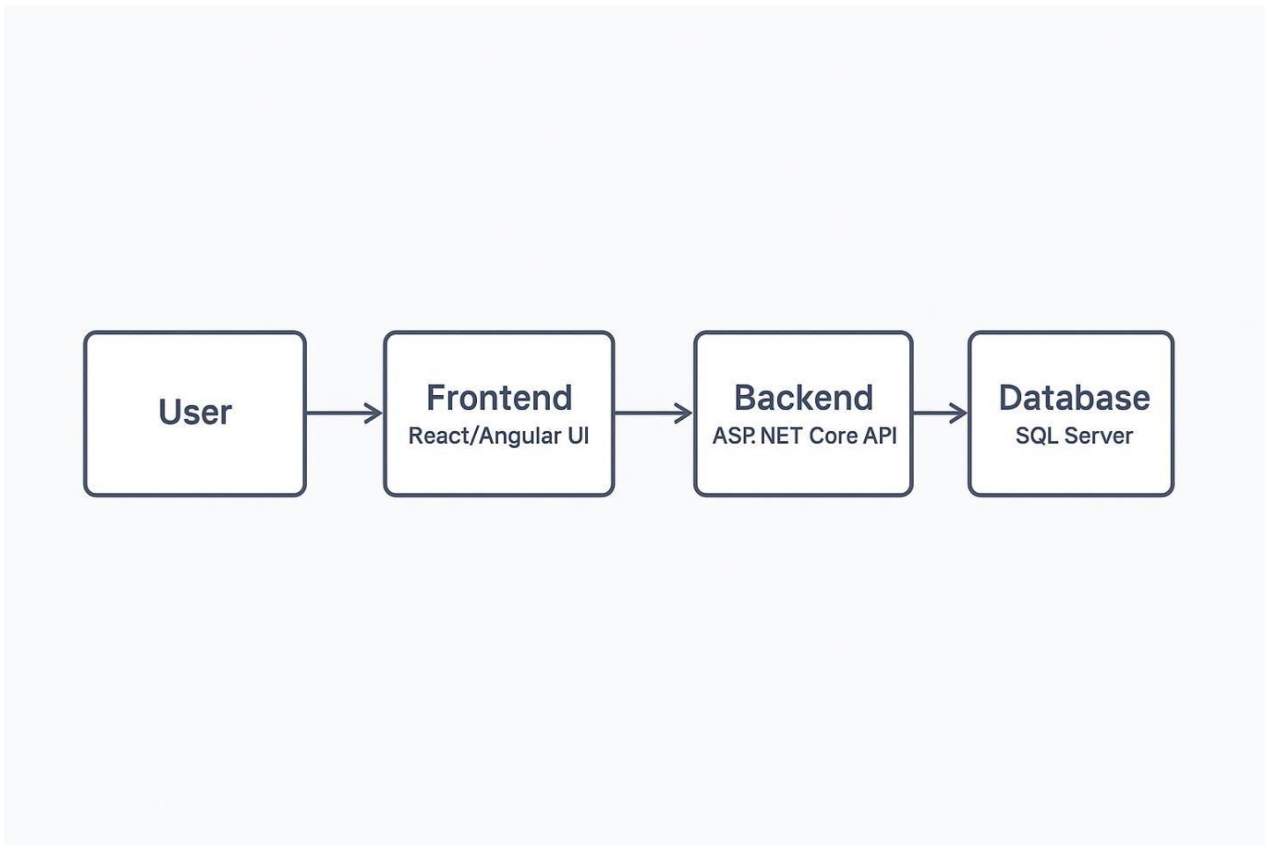


Figure 2:Frontend & Backend Architecture

➢ Component Breakdown & API Design

Frontend Components (Angular):

➢ Routing Module:

App Routing Module configured for navigation between public, user, and admin pages.

➢ Core Components:

1.)Login Component

➢ 2.) Register Component

3.)Question List Component (Homepage with search)

4.) Question Form Component

5.)Answer List Componen

➢ 6.)Answer Form Component

7.)User Management Component

8.) Admin Dashboard Component (With approval queue and notifications)

9.)Search Question Component

10.)Profile Component

➢ Services:

1.)Auth Service

2.)Question Service

3.)Answer Service

4.)User Service

➢ State Management: Angular Services (with RxJS BehaviorSubject) are used to manage application state (e.g., user authentication status, list of questions).

Backend Components & APIs (ASP.NET Core Web API):

Controllers:

➢ Web Api Controllers:

1.)User Api Controller

2.)Question Api Controller

3.)Answer Api Controller

4.)Image Api Controller

➢ Mvc Controllers:

1.)Account Controller

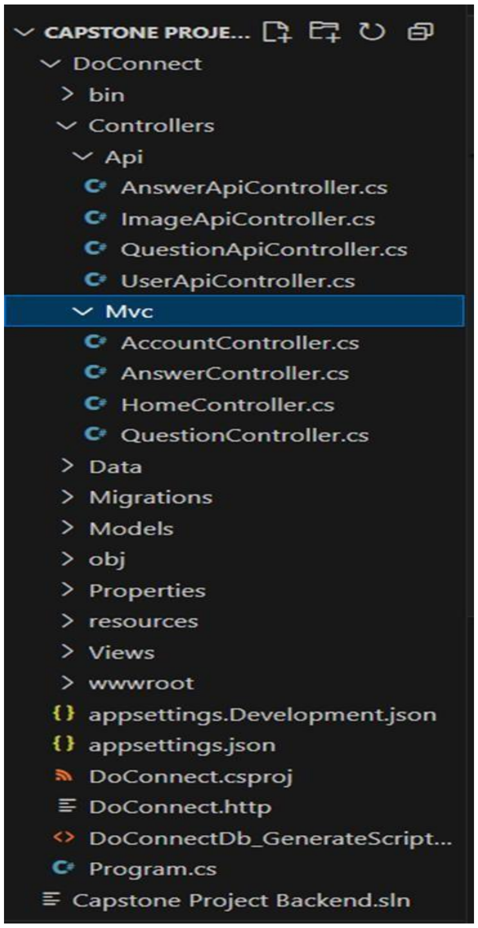
2.)Home Controller

3.)Question Controller

4.) Answer Controller

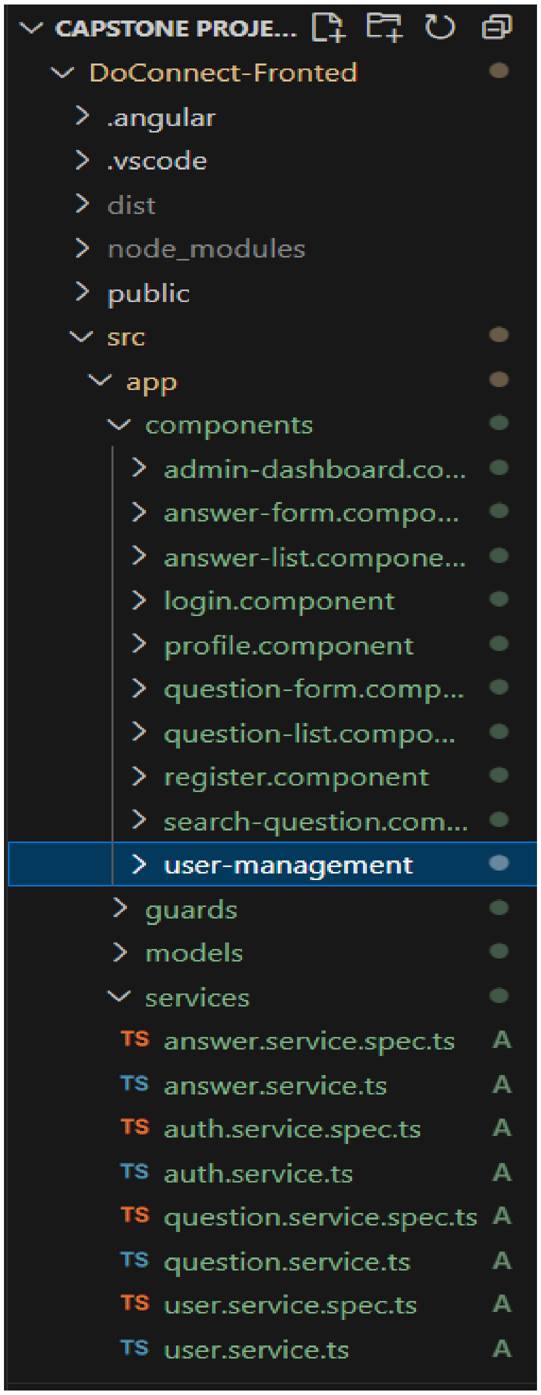
Authentication: JWT Bearer Authentication is implemented globally, validating tokens on secured endpoints.

Documentation: Swagger UI is integrated, providing interactive documentation for all API endpoints, authentication methods, and request/response models.



```
CAPSTONE PROJE...
  DoConnect
    bin
    Controllers
      Api
        AnswerApiController.cs
        ImageApiController.cs
        QuestionApiController.cs
        UserApiController.cs
      Mvc
        AccountController.cs
        AnswerController.cs
        HomeController.cs
        QuestionController.cs
    Data
    Migrations
    Models
    obj
    Properties
    resources
    Views
    wwwroot
    appsettings.Development.json
    appsettings.json
    DoConnect.csproj
    DoConnect.http
    DoConnectDb_GenerateScript...
    Program.cs
  Capstone Project Backend.sln
```

Figure 3:Backend Controller's(Web Api's &Mvc Controllers)



```
CAPSTONE PROJE...
  DoConnect-Fronted
    .angular
    .vscode
    dist
    node_modules
    public
    src
      app
        components
          admin-dashboard.co...
          answer-form.compo...
          answer-list.compone...
          login.component
          profile.component
          question-form.comp...
          question-list.compo...
          register.component
          search-question.com...
          user-management
        guards
        models
        services
          TS answer.service.spec.ts      A
          TS answer.service.ts           A
          TS auth.service.spec.ts        A
          TS auth.service.ts             A
          TS question.service.spec.ts    A
          TS question.service.ts         A
          TS user.service.spec.ts        A
          TS user.service.ts             A
```

Figure 4:Frontend Components & Services

➢ Database Design & Storage Optimization

The database schema is normalized across four main tables to reduce redundancy and ensure data integrity. Entity Framework Core Code-First approach was used to define the models and create the database schema.

Users Table: Stores user credentials and role. Passwords are stored as hashed and salted values.

Questions Table: Contains all user-submitted questions. The Status field (e.g., 'Pending', 'Approved', 'Rejected') is crucial for the approval workflow.

Answers Table: Contains all answers linked to a specific question. Also includes a Status field for approval.

Images Table: Stores the file paths of uploaded images, linked to either a specific question or an answer. This avoids storing binary data in the database, optimizing performance.

Optimization Techniques:

- Indexing: Indexes are applied on foreign keys (UserId, QuestionId) and frequently searched columns like QuestionTitle and Status to dramatically speed up query performance.

- EF Core Performance: Queries are optimized using .AsNoTracking() where appropriate, and eager loading (.Include()) is used to prevent the N+1 query problem when fetching questions with their answers and images.

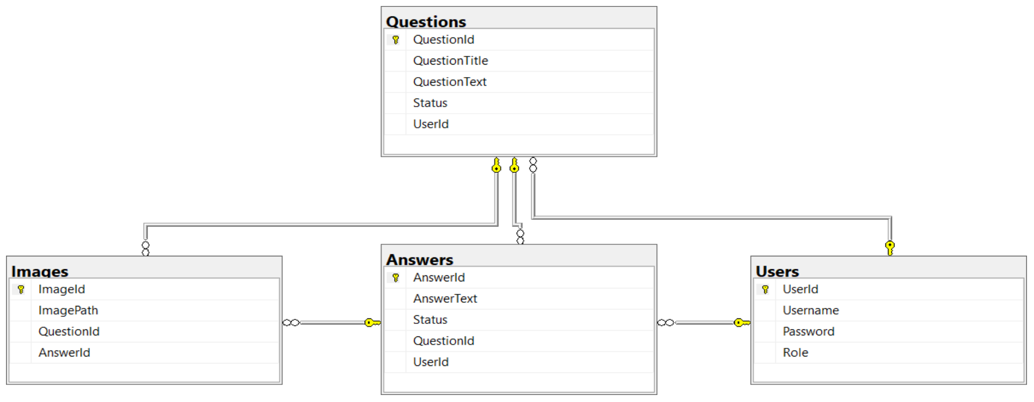


Figure 5: Database Diagram

Implementation Details (Sprints)

➢ Sprint 1: Foundation & Design:

Activities: Prepared comprehensive use case documents detailing all user and admin interactions. Designed the database schema and ERD. Set up the initial ASP.NET Core MVC project structure with placeholder views and defined the DbContext and EF Core models.

Deliverables: Use case document, SQL database creation script, and a structured backend project with defined data models.

Sprint II: Core Functionality & Integration:

Activities: Initialized the Angular application with routing and UI components. Implemented JWT authentication on both frontend and backend. Developed complete CRUD APIs for Questions, Answers, and Image Upload. Connected Angular services to consume these APIs.

Deliverables: Functional Angular frontend, secured Web API endpoints, and working user registration/login system.

Sprint III: Advanced Features & Polish

Activities: Developed and integrated the search API and frontend component. Implemented the admin approval workflow (update status). Added SignalR for real-time notifications to the admin dashboard. Integrated Swagger UI for API testing. Conducted end-to-end integration testing and bug fixing.

Deliverables: Fully functional search, admin approval module with real-time notifications, Swagger documentation, and a tested, integrated application.

Testing & Validation

The application was rigorously tested at multiple levels to ensure functionality, security, and performance:

Unit Testing: XUnit tests were written for backend API controllers and core business logic. Jasmine/Karma tests were created for Angular services and components.

Integration Testing: Verified complete workflows like user registration -> login -> ask question -> admin notification -> approval -> question visible to other users. API endpoints were tested using Swagger and Postman.

UI/UX Testing: Angular components were tested for responsiveness, form validation, and correct data binding across different browsers.

Security Testing: Verified that JWT tokens were validated correctly, role-based authorization enforced (e.g., users cannot access admin endpoints), and user data was properly isolated.

Performance Testing: Assessed API response times under load and optimized database queries to ensure scalability.

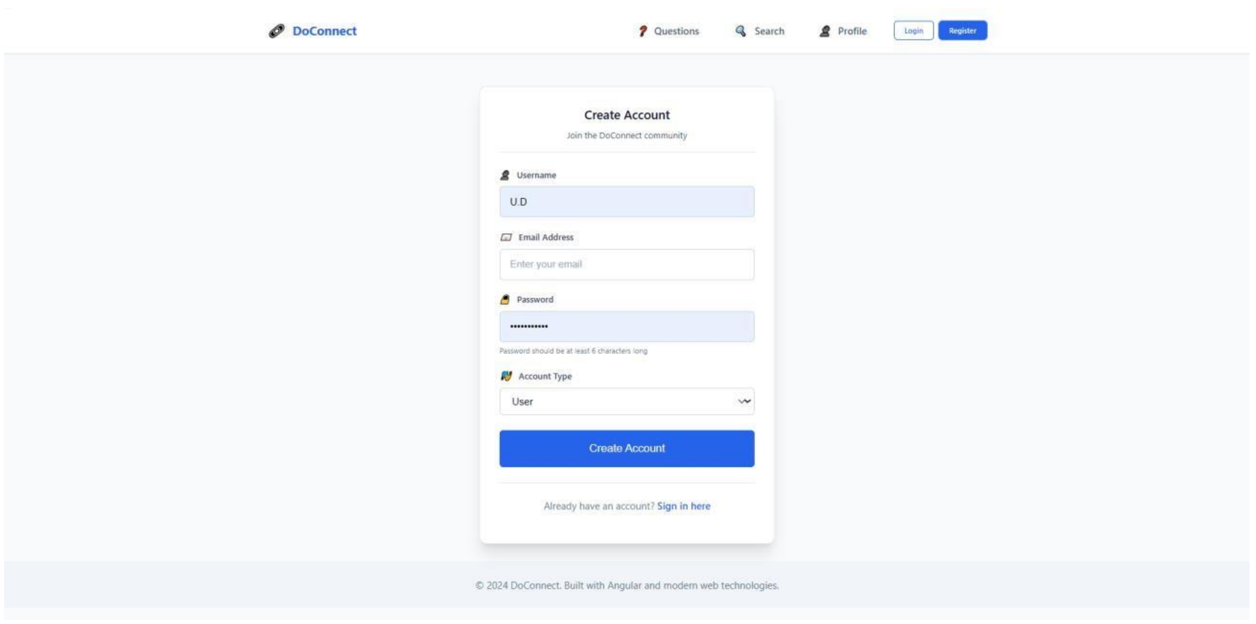
Conclusion & Future Enhancements

The DoConnect project successfully delivers a fully functional, secure, and moderated Q&A platform for technical knowledge sharing. All objectives outlined in the problem statement have been met, including user and admin authentication, question/answer management with approval workflow, image upload, search functionality, and API documentation.

Future Enhancements:

- Mobile Application: Develop a cross-platform mobile app using Ionic or React Native for on-the-go access.
- **Advanced Search:** Integrate AI-based semantic search or tags for more accurate and intuitive results.
- **Gamification:** Introduce a reputation system, badges, and points for users to increase engagement.
- **Cloud Deployment & Scalability:** Migrate the backend and database to a cloud provider like Azure or AWS, utilizing services like Azure SQL Database and Blob Storage for images.
- **Enhanced Analytics:** Develop a detailed analytics dashboard for admins to track platform growth, popular topics, and user activity.

Screenshots of Application


DoConnect
Questions
Search
Profile
Login
Register

Create Account
Join the DoConnect community

Username
U.D

Email Address
Enter your email

Password
•••••••••••
Password should be at least 6 characters long

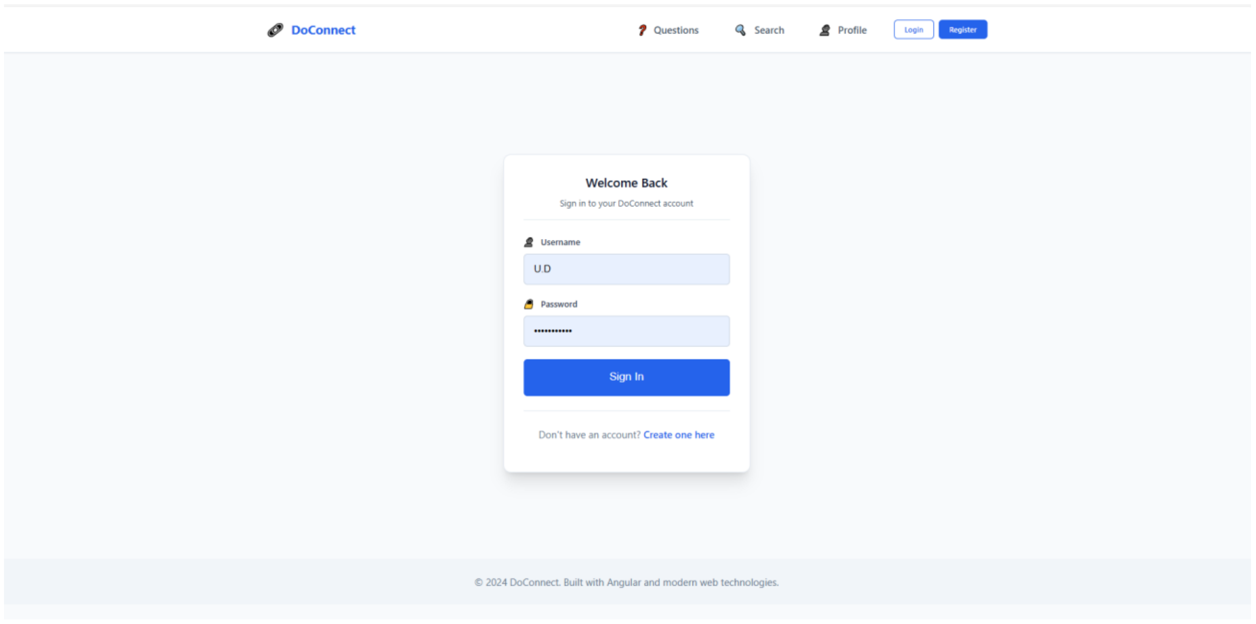
Account Type
User

Create Account

Already have an account? Sign in here

© 2024 DoConnect. Built with Angular and modern web technologies.

Figure 6: Registration Page


DoConnect
Questions
Search
Profile
Login
Register

Welcome Back
Sign in to your DoConnect account

Username
U.D

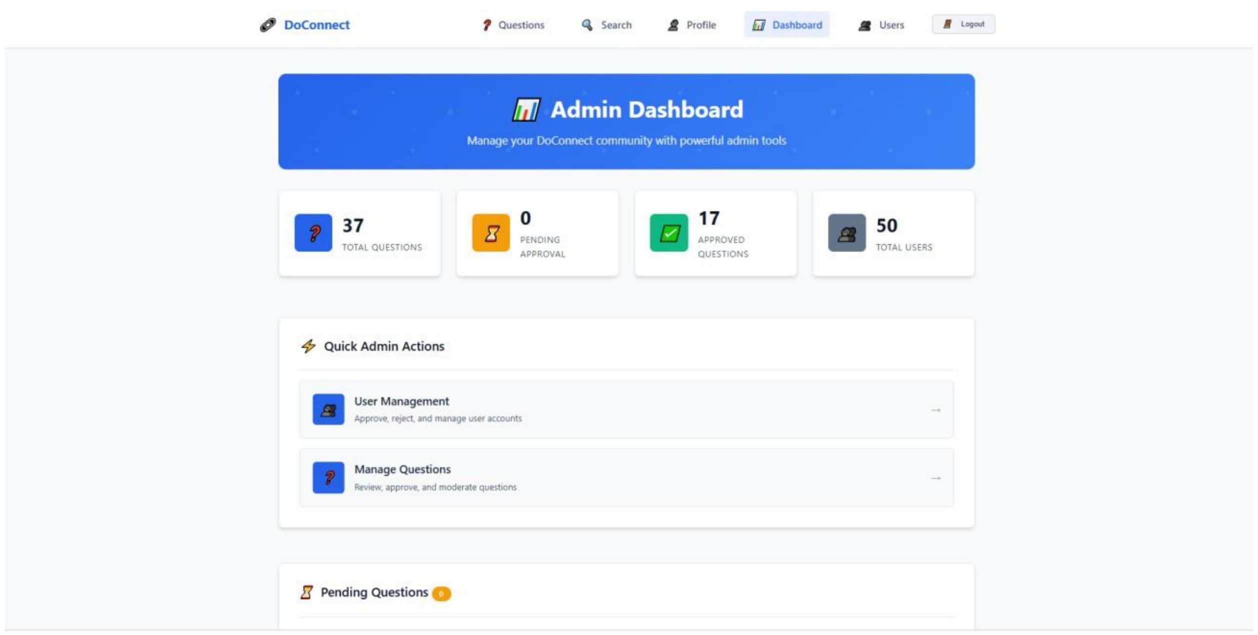
Password
•••••••••••

Sign In

Don't have an account? Create one here

© 2024 DoConnect. Built with Angular and modern web technologies.

Figure 7: Login Page



Admin Dashboard

Manage your DoConnect community with powerful admin tools

37 TOTAL QUESTIONS

0 PENDING APPROVAL

17 APPROVED QUESTIONS

50 TOTAL USERS

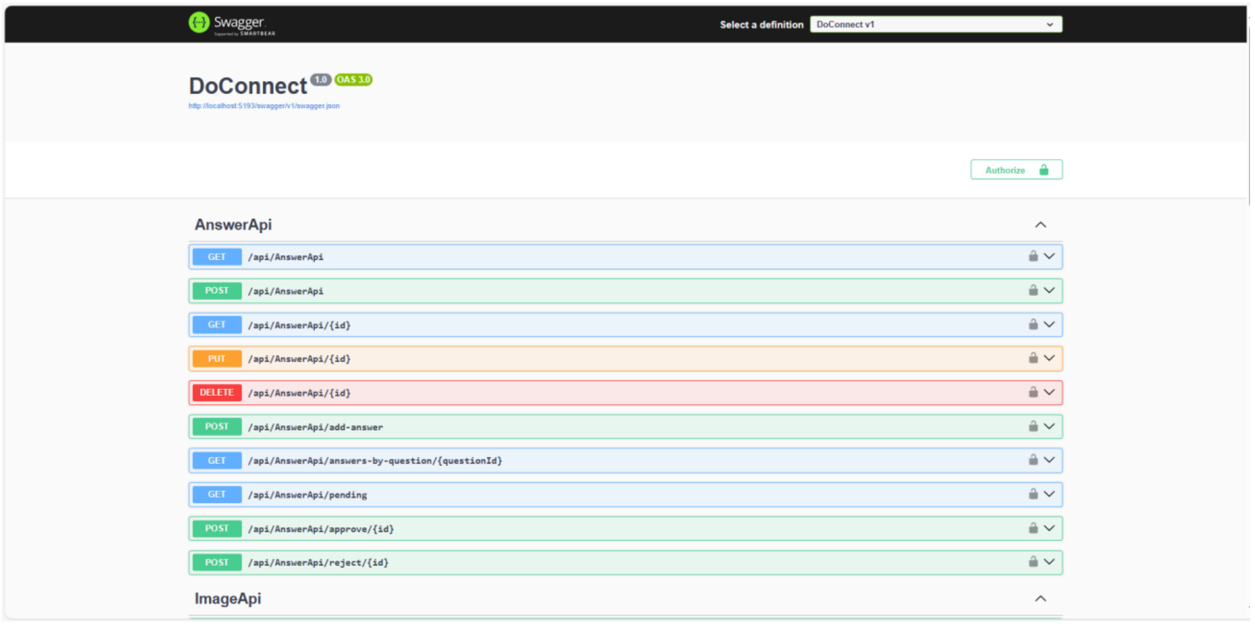
Quick Admin Actions

User Management
Approve, reject, and manage user accounts

Manage Questions
Review, approve, and moderate questions

Pending Questions 0

***Figure 8: Admin DashBoard ***



DoConnect 1.0 OAS 3.0

http://localhost:5193/swagger/v1/swagger.json

AnswerApi

```
GET     /api/AnswerApi
POST    /api/AnswerApi
GET     /api/AnswerApi/{id}
PUT     /api/AnswerApi/{id}
DELETE  /api/AnswerApi/{id}
POST    /api/AnswerApi/add-answer
GET     /api/AnswerApi/answers-by-question/{questionId}
GET     /api/AnswerApi/pending
POST    /api/AnswerApi/approve/{id}
POST    /api/AnswerApi/reject/{id}
```

ImageApi

***Figure 9:Backend Swagger Api's ***