DoConnect – Q&A Platform

Final Report

Name: Reddy Pavan Dath

Date: 08-09-2025

Table of Contents

1. Problem Definition and Objectives

Problem Statement:
In the digital era, knowledge sharing platforms are essential. However, unmoderated platforms often suffer from spam, irrelevant content, and misinformation. DoConnect is designed to provide a reliable Q&A platform where content is moderated by admins before becoming publicly visible.

Objectives:
- Allow users to register/login securely.
- Enable users to ask and answer questions with optional images.
- Implement an approval workflow where admins moderate all content.
- Ensure secure authentication and role-based authorization using JWT.
- Provide search functionality for approved questions.

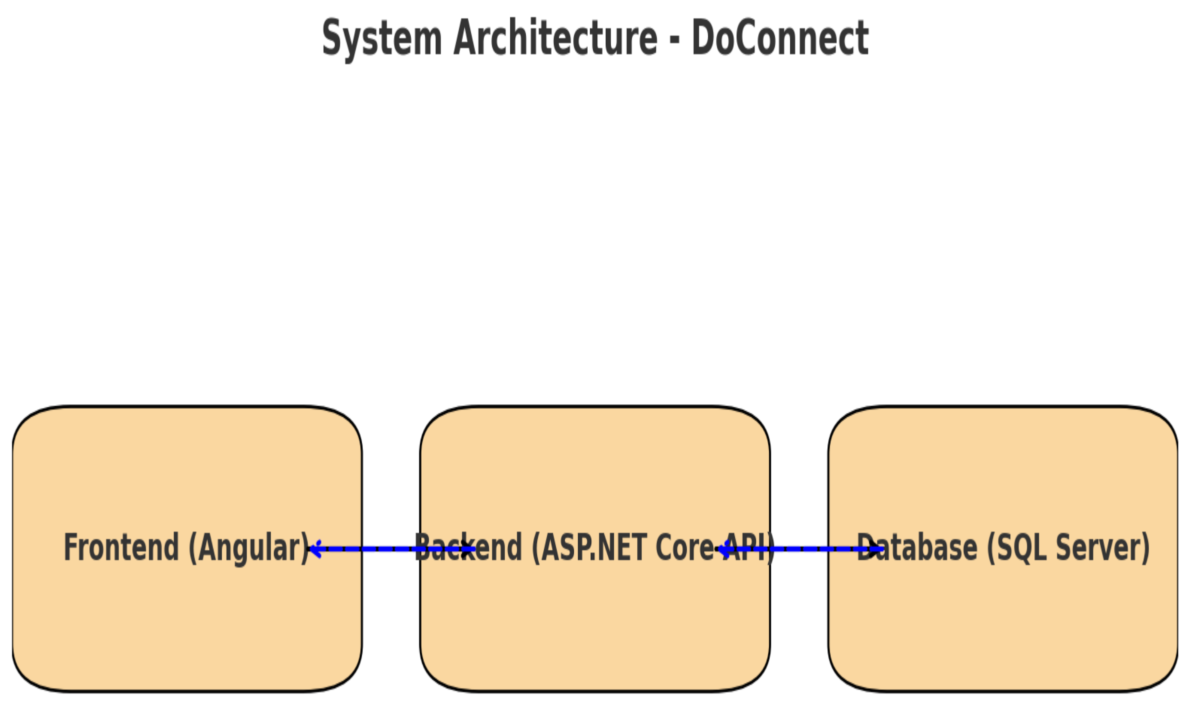
2. Frontend & Backend Architecture

The project follows a three-tier architecture:
- Frontend: Angular framework for building a dynamic and responsive user interface.
- Backend: ASP.NET Core Web API for handling business logic, authentication, and communication with the database.
- Database: Microsoft SQL Server for structured storage of users, questions, answers, and images.

The frontend communicates with the backend via REST APIs, and the backend uses Entity Framework Core to interact with the database.

3. System Design Diagram

The system consists of the following components:
1. Angular Frontend → Handles user interactions.
2. ASP.NET Core Web API → Exposes endpoints for authentication, questions, answers, and admin actions.
3. SQL Server Database → Stores persistent data.



System Architecture

The system follows a **three-tier architecture** (also called n-tier architecture). This makes the project more **scalable, secure, and maintainable**.

---

1. Frontend Layer (Angular)

- Built with **Angular**.
- Provides the **User Interface (UI)**: Login page, Question List, Ask Question form, Admin Dashboard, etc.
- Handles **form validation** before sending data.
- Uses **HTTPClient** to call the backend API.

- Stores **JWT token** in local storage/session storage.
- Ensures only logged-in users or admins can access certain routes (via route guards).

Example: When a user clicks **"Ask Question"**, Angular collects input, validates it, and sends a **POST request** to the backend API.

---

2. Backend Layer (ASP.NET Core Web API)

- Acts as the **middle layer** between frontend and database.
- Responsible for:
  - **Authentication & Authorization** (JWT tokens).
  - **Business Logic** (e.g., setting status of new questions as Pending).
  - **Data Validation** (ensuring clean and safe data is stored).
  - **Routing & Endpoints** (e.g., /auth/login, /questions, /admin/approve).
- Uses **Entity Framework Core** to communicate with the database.

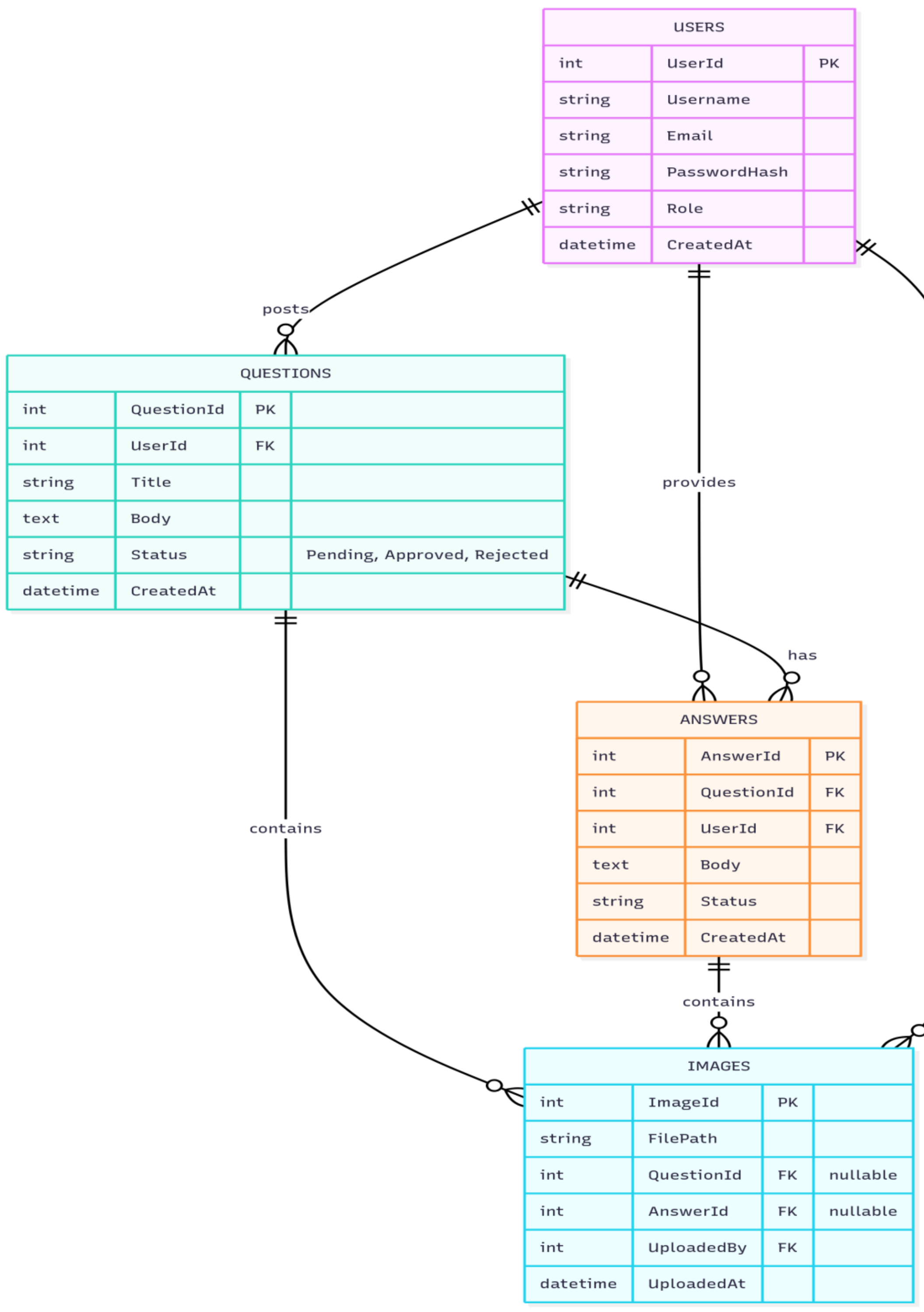
Example: When a new user registers, backend hashes their password with BCrypt, saves it in the database, and returns a success response.

---

3. Database Layer (SQL Server)

- Stores all **persistent data**: Users, Questions, Answers, Images.
- Uses **foreign keys** to maintain relationships.
- Ensures **referential integrity** (cascade deletes, no orphan records).
- Optimized with **indexes** for faster searches.

Example: If an admin deletes a user, all their questions and answers can also be deleted automatically via cascade delete.



USERS		
int	UserId	PK
string	Username	
string	Email	
string	PasswordHash	
string	Role	
datetime	CreatedAt	

QUESTIONS			
int	QuestionId	PK	
int	UserId	FK	
string	Title		
text	Body		
string	Status		Pending, Approved, Rejected
datetime	CreatedAt		

ANSWERS		
int	AnswerId	PK
int	QuestionId	FK
int	UserId	FK
text	Body	
string	Status	
datetime	CreatedAt	

IMAGES			
int	ImageId	PK	
string	FilePath		
int	QuestionId	FK	nullable
int	AnswerId	FK	nullable
int	UploadedBy	FK	
datetime	UploadedAt		

ERD Diagram

Entities and Their Purpose

1. **Users**: Represents registered users.
    - Attributes: UserId (PK), Username, Email, PasswordHash, Role, CreatedAt.
    - Purpose: Stores user account details, including authentication and role-based access.
2. **Questions**: Represents questions posted by users.
    - Attributes: QuestionId (PK), UserId (FK to Users), Title, Body, Status (Pending, Approved, Rejected), CreatedAt.
    - Purpose: Captures question details with moderation status.
3. **Answers**: Represents answers to questions.
    - Attributes: AnswerId (PK), QuestionId (FK to Questions), UserId (FK to Users), Body, Status, CreatedAt.
    - Purpose: Stores user responses to questions, linked to specific questions and users.
4. **Images**: Represents images uploaded to questions or answers.
    - Attributes: ImageId (PK), FilePath, QuestionId (FK to Questions, nullable), AnswerId (FK to Answers, nullable), UploadedBy (FK to Users), UploadedAt.
    - Purpose: Stores image metadata, allowing images to be associated with either a question or an answer, with uploader tracking.

4. Component Breakdown & API Design

Frontend Components:

- Auth Module (Register/Login)
- Question Module (Ask Question, View Questions)
- Answer Module (Post Answers)
- Admin Dashboard (Approve/Reject content)
- Shared Components (Navigation bar, forms, routing)

Backend API Endpoints:

- POST /auth/register → Register user
- POST /auth/login → Authenticate user, return JWT
- POST /questions → Create new question (User only)
- GET /questions → Search/list approved questions
- POST /answers → Submit answer (User only)
- POST /images/upload → Upload image
- GET /admin/pending → List pending content
- POST /admin/approve → Approve/Reject content

Authentication: JWT-based authentication with role-based authorization.

5. Database Design & Storage Optimization

The database schema includes four main tables: Users, Questions, Answers, and Images.
Relationships:
- A User can create multiple Questions and Answers.
- A Question can have multiple Answers and Images.
- An Answer can also have multiple Images.

Optimization Techniques:
- Indexed foreign keys for faster joins.
- NVARCHAR used only where necessary to optimize storage.
- Cascade delete rules to maintain referential integrity.

6. Conclusion & Future Scope

The DoConnect platform successfully demonstrates a secure and moderated Q&A system built using Angular, ASP.NET Core, and SQL Server. It meets the core objectives of secure authentication, question/answer posting, and admin moderation.

Future enhancements include:
- Real-time notifications using SignalR.
- Advanced search with full-text indexing.
- Mobile app integration.
- Enhanced admin analytics dashboard.