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 Update
{
    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 Updat
{
    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);
}
```

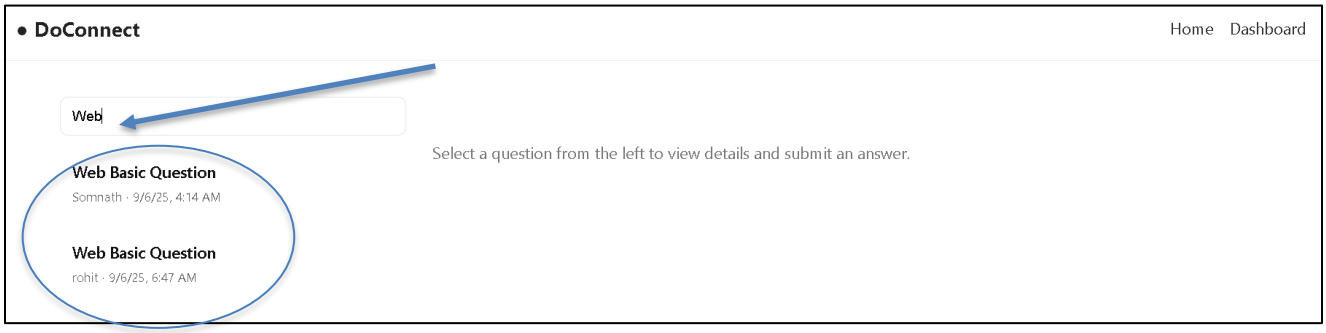


Fig – Search Functionalities in Frontend

- **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

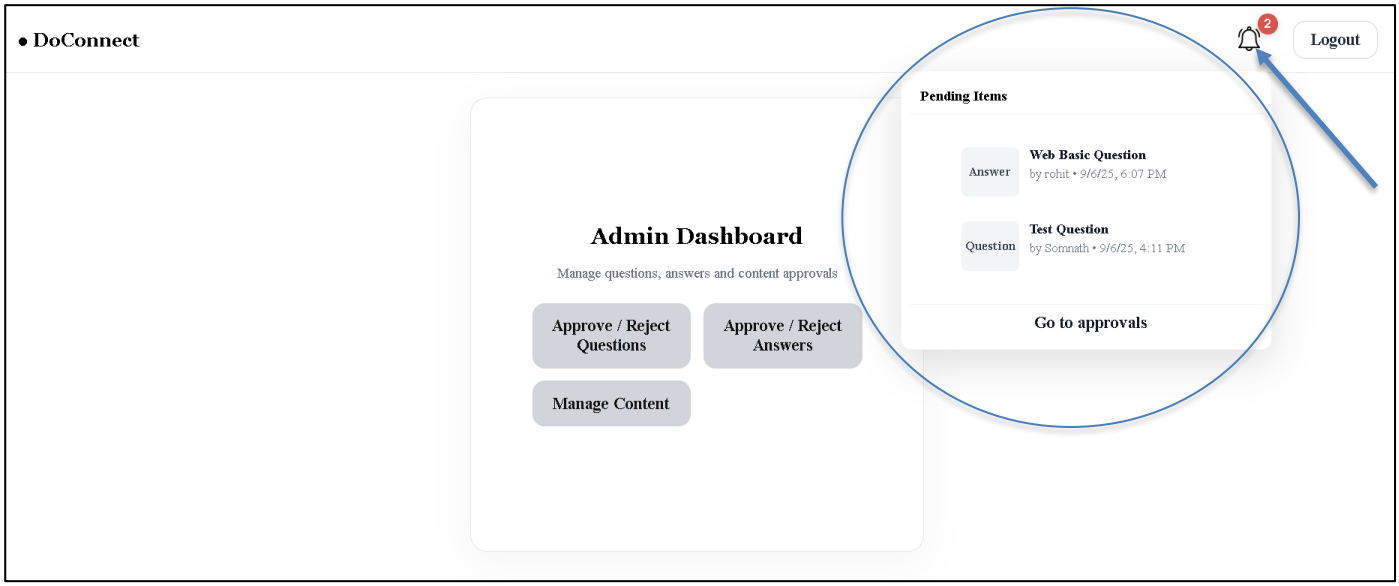


Fig – Admin gets notified while a User request a question/answer for approval

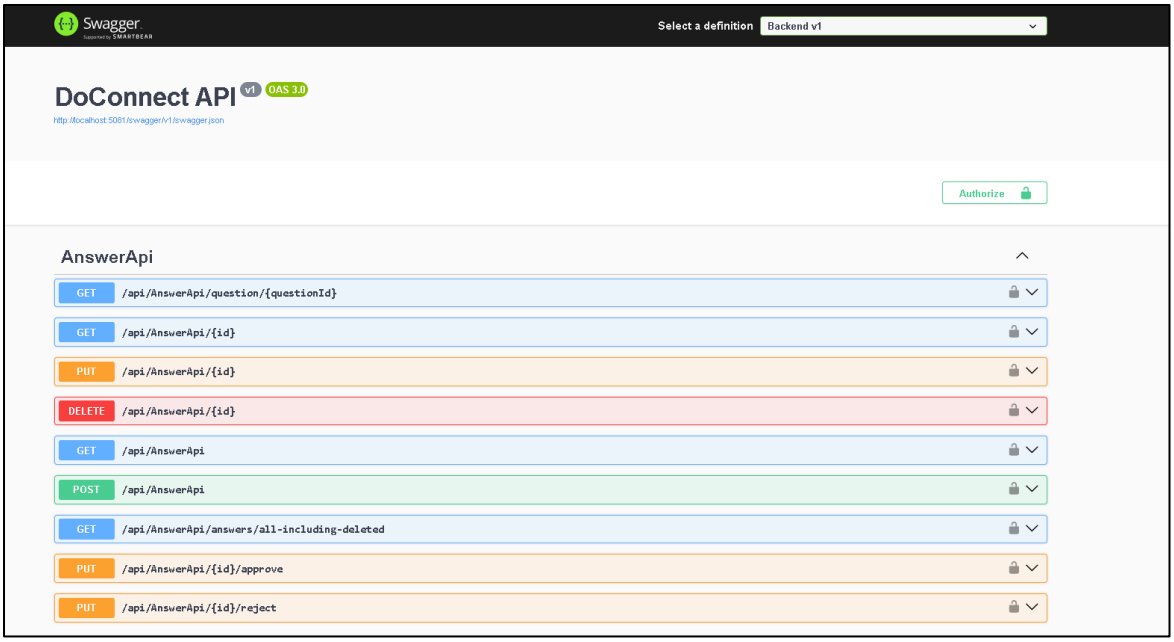
Fig – Approve/Reject operations by Admin

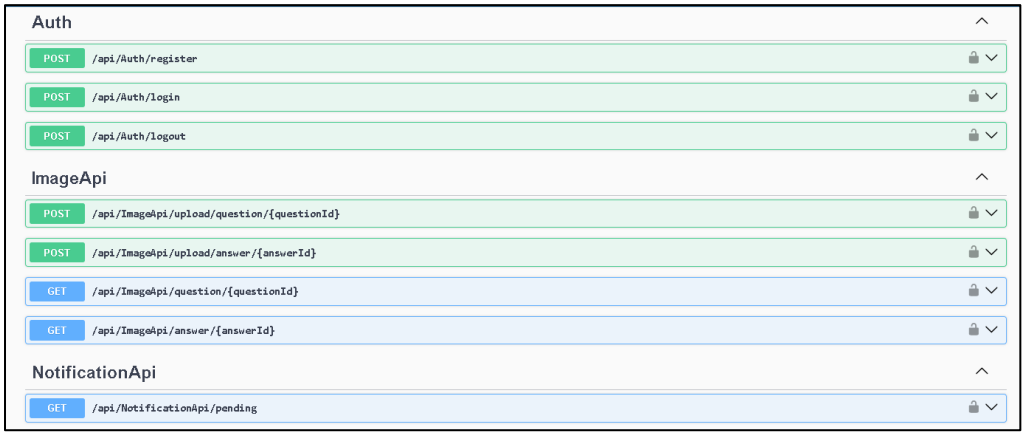
- **Swagger UI documentation for all Web API endpoints-**
  Swagger was configured to document and test all Web API endpoints, including authentication, CRUD operations, and image uploads, with support for JWT authorization.

```
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)





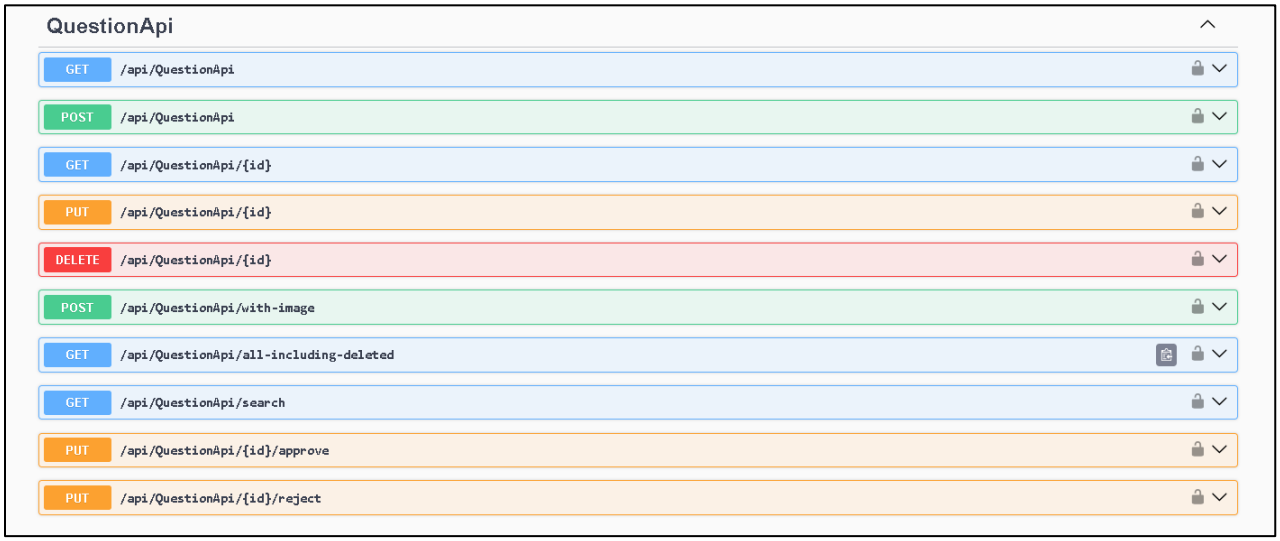


Fig – Swagger UI

- **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

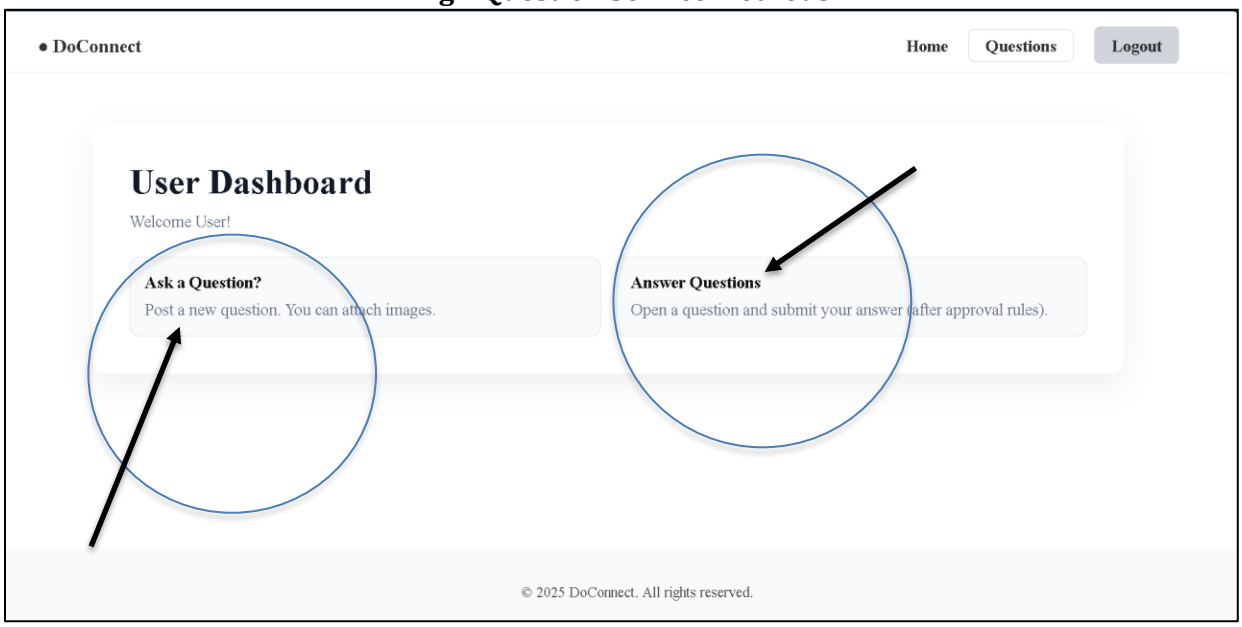


Fig- User Workflow Components

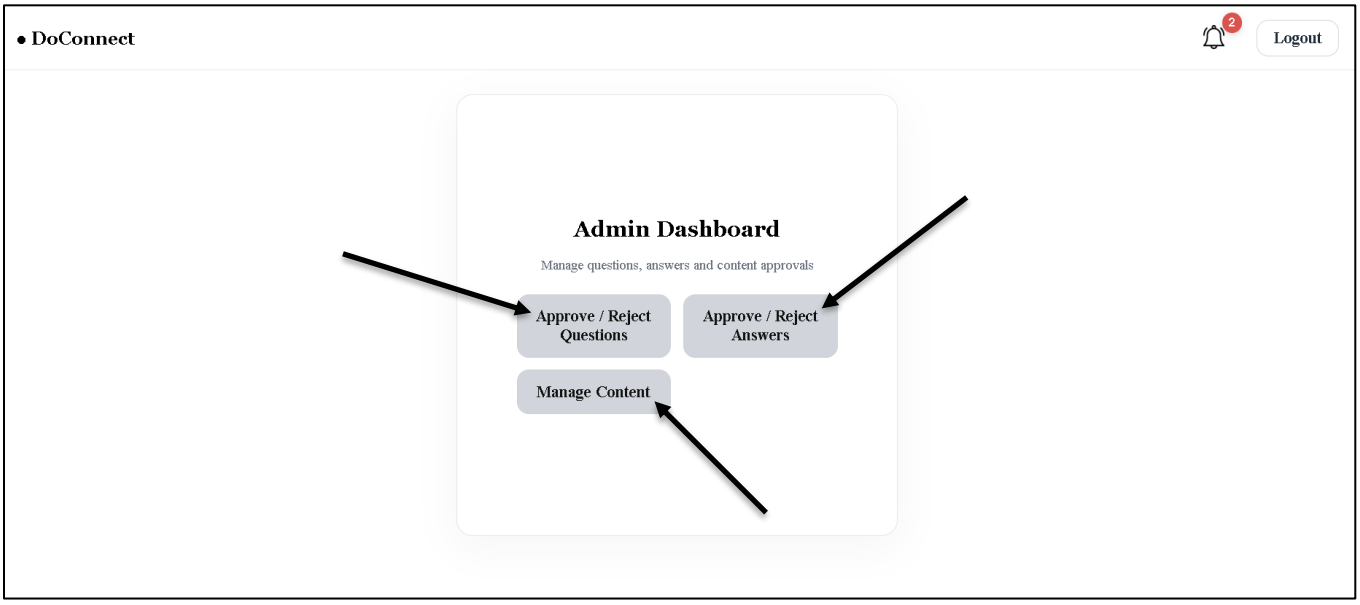


Fig- Admin Workflow Components

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.