SOFTWARE REQUIREMENTS SPECIFICATION

for

StudyFlow

Version 1.0

Prepared by : Nourhan ElSheikh
Reem Khaled Ali
Sarah ElShinnawy
Youssef Alaa-eldin

Submitted to : Mohamed Hassan ElGazzar

December 2025

Contents

1 Introduction

1.1 Purpose

This document defines the Software Requirements Specification (SRS) for StudyFlow, a collaborative Q&A web platform designed for academic environments. The system facilitates knowledge sharing between students and instructors through question posting, answering, discussion forums, notifications, and content management. This document serves as a comprehensive guide for developers, testers, and stakeholders throughout the project's development period.

1.2 Intended Audience and Reading Suggestions

This SRS is intended for:

- **Developers:** To understand system architecture and functional details.
- **Project Managers:** To plan and track progress within the development schedule.
- **Testers:** To derive test cases and validation procedures from functional requirements.
- **Stakeholders:** To review system capabilities and constraints.
- **Users:** To understand the platform's purpose and main functions.

The document is organized sequentially, starting from the system overview and continuing through specific requirements, constraints, and design decisions.

1.3 Project Scope

StudyFlow is an educational, collaborative Q&A platform inspired by Quora and Stack Overflow, designed to facilitate structured academic discussions between students and instructors.

Implemented Features (Version 1.0):

- User registration and authentication with role selection (Student/Instructor).
- Token-based authentication using built in Django framework
- Question posting with tagging and categorization.
- Answer submission with notification system.
- User profiles with avatar upload functionality.
- Personalized user dashboard with statistics and activity tracking.
- Content management (CRUD operations for questions).
- Notification system for replies and interactions.
- Popular tags retrieval and display.
- Search and filtering functionality.
- Responsive web interface.
- Real-time notification polling system.

Out of Scope (Current Version):

- AI-based tag suggestions or recommendations.
- Gamification system (badges, reputation points).
- Real-time chat or WebSocket push notifications.
- Mobile applications (responsive web only).

System Actors:

- **Student:** Basic user with posting and interaction privileges.
- **Instructor:** Verified user with additional privileges such as answer verification.

- **Administrator:** Platform moderator with full control.
- **Notification Service:** Automated system responsible for delivering alerts.

2 Overall Description

2.1 Product Perspective

StudyFlow is a web-based platform built using a three-tier architecture: presentation layer (React.js), business logic layer (Django REST Framework), and data layer (PostgreSQL/SQLite). The system stores data in a relational database and supports simultaneous user interactions.

Architecture Overview:

- **Frontend:** React.js single-page application with React Router for navigation.
- **Backend:** Django REST Framework providing RESTful API endpoints.
- **Database:** Relational database with models for Users, Questions, Answers, Tags, Notifications, Votes, Comments, and Reports.
- **Authentication:** Token-based system.

Primary Modules:

- **User Module (users app):** Manages authentication, profiles, and user-generated content.
- **Questions Module (questions app):** Handles questions, tagging, and question management.
- **Answers Module (answers app):** Manages answer submission and retrieval.
- **Core Module (core app):** Handles notifications, voting models, comments models, and reports models.

2.2 User Classes and Characteristics

- **Student:** Can register, ask questions, post answers, view content, receive notifications, and manage their profile.
- **Instructor:** Has all Student privileges, can mark "Best Answers", and provide verified responses.
- **Administrator:** Manages users, posts, categories, and reported content.

2.3 Product Functions

Implemented Core Functions:

- **User Authentication:** Registration with role selection, login with token generation, profile management with avatar upload.
- **Question Management:** Create questions with tags, view all questions, view question details, update questions (author only), delete questions (author only).
- **Answer Management:** Submit answers to questions, view answers for a question, automatic notification generation when answered.
- **Tag System:** Automatic tag creation and association, tag reuse for existing tags, popular tags endpoint showing top 5 most used tags.
- **Notification System:** Automatic notification creation for interactions, notification retrieval endpoint, mark notifications as read functionality, polling-based notification updates (every 30 seconds).

- **User Dashboard:** Personal statistics (questions asked, answers given, reputation score), activity history display, profile picture management.
- **Content Discovery:** Explore feed showing latest questions, question detail view with answers, clickable question cards for navigation.

Database Schema:

- **Users Table:** user_id, username, email, password (hashed), role (Student/Instructor), profile_picture, registration_date, last_login.
- **Questions Table:** question_id, title, body, user_id, creation_date, updated_date, views, tags (many-to-many relationship).
- **Answers Table:** answer_id, question_id, body, user_id, creation_date, votes, is_best_answer.
- **Tags Table:** tag_id, name, description, usage_count.
- **Votes Table:** vote_id, content_id, user_id, vote_type, timestamp.
- **Comments Table:** comment_id, parent_id, content, user_id, creation_date.
- **Reports Table:** report_id, content_id, reporter_id, reason, status.
- **Notifications Table:** notification_id, user_id, type (reply, answer, system), reference_id, is_read, timestamp, message.

3 System Architecture

3.1 Frontend Architecture

Application Structure:

- **App.tsx:** Main routing component defining all application routes and navigation logic.
- **main.tsx:** Application entry point that renders the React app into the DOM.

Pages (Views):

- **HomePage.tsx:** Landing page displaying popular tags and navigation hubs.
- **user.tsx:** Personal dashboard showing profile, questions, answers, and "Ask Question" modal.
- **Explore.tsx:** Community feed displaying latest 10-20 questions from all users.
- **QuestionDetail.tsx:** Detailed question view with answer submission functionality.

Components (Building Blocks):

- **navbar.tsx:** Persistent navigation bar with logo, links, and notification system (30-second polling).
- **posts.tsx:** Reusable question card component used across Explore, User Profile, and Dashboard.
- **Form.tsx:** Login and registration form with input validation and role selection.

Styling:

- **Global Styles:** index.css and App.css for fonts and reset rules.
- **Page-Specific CSS:** user.css, Explore.css, etc., implementing glassmorphism effects and animations.

3.2 Backend Architecture

API Endpoint Structure:

- **User Management:**
  - POST /api/register/ - User registration with role selection
  - POST /api/login/ - Token-based authentication
  - POST /api/profile/upload-avatar/ - Profile picture upload
  - GET /api/dashboard/ - User statistics and activity
- **Questions:**
  - POST /api/posts/ - Create new question
  - GET /api/posts/ - List all questions
  - GET /api/posts/¡id¿/ - Question details
  - PUT /api/posts/¡id¿/ - Update question (author only)
  - DELETE /api/posts/¡id¿/ - Delete question (author only)
- **Tags:**
  - GET /api/tags/popular/ - Retrieve top 5 most used tags
- **Answers:**
  - POST /api/answers/ - Submit answer to question
  - GET /api/answers/ - List answers (currently global, needs filtering)
- **Notifications:**
  - GET /api/notifications/ - Retrieve user notifications
  - POST /api/notifications/¡id¿/read/ - Mark notification as read

4 System Features

4.1 Functional Requirements

4.1.1 User Management & Authentication (users app)

Features:

- **Registration:** Users can register as Student (default) or Instructor.
  - Endpoint: POST /api/register/
  - Returns: Authentication Token
  - Validation: Username uniqueness, email format, password strength
- **Login:** Token-based authentication using username/password.
  - Endpoint: POST /api/login/
  - Returns: Authentication Token
  - Session management with token storage
- **Profile Management:** Users can upload/update their profile picture.
  - Endpoint: POST /api/profile/upload-avatar/
  - Supports: Image file upload (JPEG, PNG)
  - Updates user model with profile picture URL
- **User Dashboard:** Personalized view showing user stats and history.
  - Endpoint: GET /api/dashboard/
  - Data: Questions asked, Answers given, Reputation score (Upvotes - Downvotes)

– Activity history with recent interactions

Test Cases:

Test Case	Description
Register Student	Verify a user can sign up with Student role and receives a token.
Register Instructor	Verify a user can sign up with Instructor role and receives a token.
Login Valid	Verify valid credentials return an authentication token.
Login Invalid	Verify invalid credentials return 401 Unauthorized.
Dashboard Data	Verify dashboard correctly counts questions, answers, and calculates reputation score based on votes.
Profile Picture Upload	Verify image upload works and updates the user model with the new avatar.

4.1.2 Questions & Tagging (questions app)

Features:

- **Create Question:** Authenticated users can post a question with title, body, and tags.
  - Endpoint: POST /api/posts/
  - Logic: Automatically assigns current user as author
  - Tag handling: Reuses existing tags or creates new ones
  - Returns: Created question with ID
- **List Questions:** View all questions (Public access).
  - Endpoint: GET /api/posts/
  - Returns: List of questions with tags, author, and metadata
  - Used in Explore page
- **Question Detail:** View, Update, or Delete a specific question.
  - Endpoint: GET/PUT/DELETE /api/posts/¡id¿/
  - Permissions: Only author (or admin) can Update/Delete
  - Returns: Full question details with answers
- **Popular Tags:** Retrieve top 5 most used tags.
  - Endpoint: GET /api/tags/popular/
  - Returns: Tags ordered by usage_count
  - Displayed in sidebar navigation

Test Cases:

4.1.3 Answers (answers app)

Features:

- **Answer Question:** Authenticated users can post an answer to a specific question.
  - Endpoint: POST /api/answers/
  - Logic: Links answer to question and current user
  - Triggers: Automatic notification generation for question author

Test Case	Description
Create Question	Verify specific fields (title, body, tags) are saved correctly in database.
Update Question	Verify author can change title, body, and tags. Ensure tags are correctly replaced/updated.
Delete Question	Verify the question record is removed from database.
Popular Tags	Verify API returns tags ordered by usage_count (descending).
Tags Logic	Verify reuse of existing tags vs creation of new ones when posting questions.
Unauthorized Update	Verify non-authors cannot update questions.

  - Returns: Created answer with ID

- **List Answers:** Retrieve answers.
  - Endpoint: GET /api/answers/
  - Note: Currently returns global list; filtering by question ID recommended
  - Returns: List of answers with author and metadata
- **Best Answer:** Feature present in database model (is_best_answer).
  - Logic exists in Answer model (mark_as_best method)
  - API endpoint not yet implemented
  - Planned for future release

Test Cases:

Test Case	Description
Post Answer	Verify answer is correctly linked to question and user in database.
Answer Notification	Verify posting an answer triggers a notification for the question author.
List Answers	Verify answers are retrieved with correct metadata.

4.1.4 Notifications & Core Interactions (core app)

Features:

- **Notifications:** Users receive system alerts for interactions.
  - Endpoint: GET /api/notifications/
  - Types: Answer notifications, question confirmations, system alerts
  - Returns: List of notifications with read/unread status
  - Frontend: Polling every 30 seconds via navbar component
- **Mark Read:** Users can mark notifications as read.
  - Endpoint: POST /api/notifications/¡id¿/read/
  - Logic: Toggles is_read status
  - Returns: Updated notification status
- **Notification Generation:** Automatic creation on specific events.

- Triggers: Question posted, answer submitted, best answer marked
- System: Background notification service
- Storage: Notifications table with reference IDs

Planned Features (Models Exist, API Not Implemented):

- **Voting:** Model exists (Vote) for Up/Down votes on Questions/Answers.
  - Database: vote_id, content_id, user_id, vote_type, timestamp
  - Planned: API endpoints to cast/revoke votes
  - Logic: Prevent double-voting and voting on own content
- **Comments:** Model exists (Comment) for threaded discussions.
  - Database: comment_id, parent_id, content, user_id, creation_date
  - Planned: API endpoints to add/view comments
  - Logic: Nested comment threading
- **Reports:** Model exists (Report) for content moderation.
  - Database: report_id, content_id, reporter_id, reason, status
  - Planned: API endpoints for users to report content
  - Planned: Admin interface to review reports

Test Cases:

Test Case	Description
Receive Notification	Verify system generates Notification object when triggers occur (Question Posted, Answered).
Mark Read	Verify is_read status toggles after calling the endpoint.
Question Notification	Verify posting a question triggers a system notification event.
Answer Notification Integration	Verify when Instructor answers Student's question, Student receives notification.

4.1.5 Administrator Requirements

Planned Features:

- Manage all content and user activities.
- Moderate reports, tags, and categories.
- Receive alerts about new reports or violations.
- User management (suspend, delete accounts).
- Content moderation (edit, delete, feature posts).

Status: Admin functionality planned but not yet implemented in current version.

5 Other Nonfunctional Requirements

5.1 Performance Requirements

- Support at least 100 concurrent users.
- Page load time below 3 seconds.
- API response time under 500ms for standard requests.

- Notification polling interval: 30 seconds.
- Notification latency below 5 seconds for new events.
- 99% uptime during peak academic hours.
- Database query optimization for large datasets.

5.2 Security Requirements

- Passwords hashing
- Token-based authentication with secure token storage.
- Enforce HTTPS connections for all API requests.
- Role-based access control (Student, Instructor, Administrator).
- Sanitize user input to prevent SQL injection and XSS attacks.
- Secure file upload validation for profile pictures.
- CORS configuration for frontend-backend communication.
- Session timeout and token expiration mechanisms.

5.3 Software Quality Attributes

- **Usability:** Clean, intuitive UI with an easy design and responsive layout.
- **Reliability:** Automatic error recovery and stable notification delivery.
- **Maintainability:** Modular structure with separate apps for different functionalities.
- **Scalability:** Architecture supports future feature expansion and increased user load.
- **Responsiveness:** Polling-based notification updates without full page reloads.
- **Testability:** Comprehensive test suite covering all implemented features.
- **Portability:** Platform-independent web application accessible via modern browsers.

6 Implementation Details

6.1 Technical Stack

Frontend:

- **Framework:** React.js with TypeScript
- **Routing:** React Router for SPA navigation
- **Styling:** Custom CSS with glassmorphism effects
- **State Management:** React Hooks (useState, useEffect)
- **HTTP Client:** Fetch API or Axios for backend communication

Backend:

- **Framework:** Django with Django REST Framework
- **API:** RESTful API architecture
- **Authentication:** Token-based authentication system
- **Database ORM:** Django ORM for database operations

Database:

- **Database:** PostgreSQL (production) / SQLite (development)
- **Schema:** Relational database with foreign key relationships
- **Migrations:** Django migrations for schema management

Development Tools:

- **Version Control:** Git with GitHub
- **Testing:** Django TestCase for backend unit tests
- **API Testing:** Django REST Framework test utilities
- **Development Server:** Django development server (127.0.0.1:8000)

Deployment:

- **Backend:** Django production server
- **Frontend:** Static file serving or CDN
- **Hosting:** Cloud platform

6.2 Implementation Timeline

Phase	Completed Tasks
Phase 1 (Days 1–2)	Project structure setup, database schema design, user authentication system with registration and login.
Phase 2 (Days 3–4)	Core Q&A features: questions module with CRUD operations, tagging system, answer submission.
Phase 3 (Days 5–6)	Notification system implementation, profile management, avatar upload functionality.
Phase 4 (Days 7–8)	Frontend development: React components, routing, styling with glassmorphism effects.
Phase 5 (Days 9–10)	User dashboard, popular tags endpoint, explore feed, question detail page.
Phase 6 (Days 11–12)	Testing suite development, bug fixes, notification polling integration, final refinements.

6.3 Test Coverage Summary

Implemented Tests (backend/core/tests.py):

- **Questions Tests:**
  - test_create_question_with_tags: Verifies question creation with tags
  - test_update_question: Verifies author can update question
  - test_delete_question: Verifies author can delete question
- **Answers & Notifications Tests:**
  - test_answer_creates_notification: Integration test for notification generation
  - test_question_creates_notification: Verifies question posting triggers notification
- **Dashboard Tests:**
  - test_dashboard_stats: Complex logic test for user statistics calculation
  - test_mark_notification_read: Verifies notification read status update
- **Tags Tests:**
  - test_popular_tags: Verifies tags sorted by popularity

Missing Test Coverage (To Be Added):

- Registration/Login failure cases (wrong password, duplicate username)

- Profile picture upload validation
- Voting system tests (pending API implementation)
- Comments system tests (pending API implementation)
- Authorization tests for unauthorized access attempts

6.4 System Gaps & Planned Enhancements

Features with Database Models but Missing API Implementation:

- **Voting System:**
  - Models: Vote table with vote_id, content_id, user_id, vote_type, timestamp
  - Missing: API endpoints to cast/revoke votes
  - Missing: Logic to prevent double-voting and self-voting
  - Missing: Reputation score calculation integration
- **Comments System:**
  - Models: Comment table with threaded discussion support
  - Missing: API endpoints to add/view/delete comments
  - Missing: Frontend components for comment display and submission
- **Best Answer Feature:**
  - Models: is_best_answer field exists in Answer model
  - Logic: mark_as_best method implemented
  - Missing: API endpoint to mark best answer
  - Missing: UI to display and select best answers
- **Content Reporting:**
  - Models: Report table for moderation
  - Missing: API endpoints for users to report content
  - Missing: Admin dashboard to review and process reports
- **Answer Filtering:**
  - Current: GET /api/answers/ returns global list
  - Needed: Filter answers by question_id parameter
  - Needed: Sorting options (newest, votes, best answer first)

6.5 Future Enhancements

- Advanced notification customization (frequency, delivery type).
- WebSocket-based real-time notifications replacing polling.
- Push notifications using Firebase Cloud Messaging.
- AI-driven tag and question suggestions.
- Gamification with badges and reputation points.
- Mobile app versions for iOS and Android.
- Integration with Learning Management Systems (LMS).
- Advanced search with filters (date range, tags, users).
- Rich text editor for questions and answers (Markdown support).
- File attachments for questions and answers.

- Email digest notifications for activity summaries.
- Social features (follow users, share questions).
- Analytics dashboard for administrators.
- API rate limiting and throttling.
- Multi-language support (i18n).

7 Appendices

7.1 Data Flow Overview

Typical User Interaction Flow:

1. User navigates to "Explore Topics" on HomePage.tsx
2. HomePage.tsx triggers navigate('/explore')
3. App.tsx routes to Explore.tsx based on URL path
4. Explore.tsx executes useEffect hook
5. Frontend sends GET request to http://127.0.0.1:8000/api/posts/
6. Django backend processes request through questions app views
7. Database query retrieves questions with related tags and authors
8. Backend returns JSON response with question list
9. React maps through data and renders Card component for each question
10. CSS applies glassmorphism effects and animations
11. User sees rendered question cards on screen

7.2 Glossary

- **API:** Application Programming Interface - RESTful endpoints for frontend-backend communication
- **CRUD:** Create, Read, Update, Delete - basic database operations
- **JWT:** JSON Web Token - authentication token format
- **ORM:** Object-Relational Mapping - Django database abstraction layer
- **SPA:** Single Page Application - React-based frontend architecture
- **Glassmorphism:** Modern UI design trend with translucent glass-like effects
- **Polling:** Periodic checking for updates (notification system)
- **Role-Based Access Control:** Permission system based on user roles
- **Tag:** Keyword or label associated with questions for categorization
- **Best Answer:** Instructor-verified answer marked as the most helpful response

7.3 References

- Django Documentation: https://docs.djangoproject.com/
- Django REST Framework: https://www.django-rest-framework.org/
- React Documentation: https://react.dev/
- React Router: https://reactrouter.com/
- PostgreSQL Documentation: https://www.postgresql.org/docs/
- RESTful API Design Best Practices

8 System Diagrams

8.1 Use Case Diagram

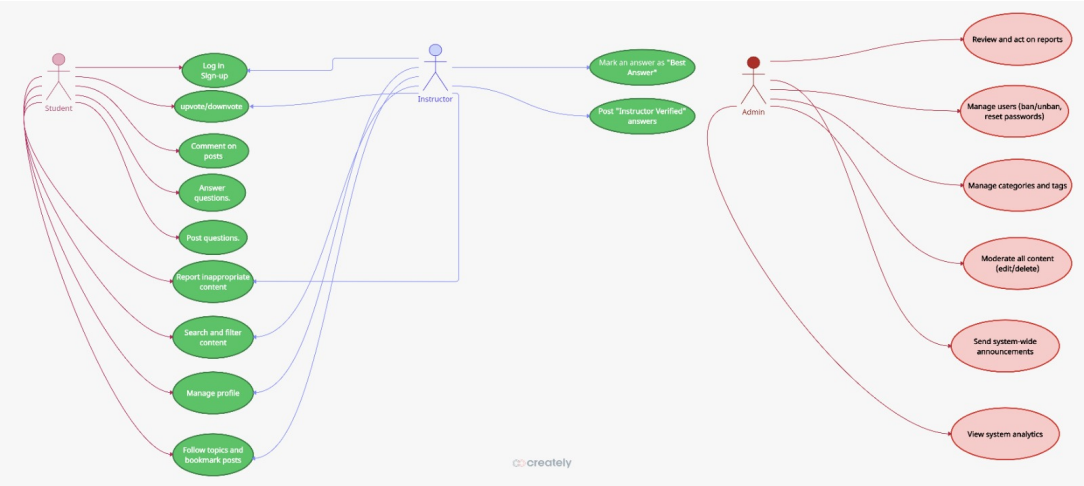

Figure 1: System Use Case Diagram showing interactions between Student, Instructor, Administrator, and the System.

8.2 Class Diagram

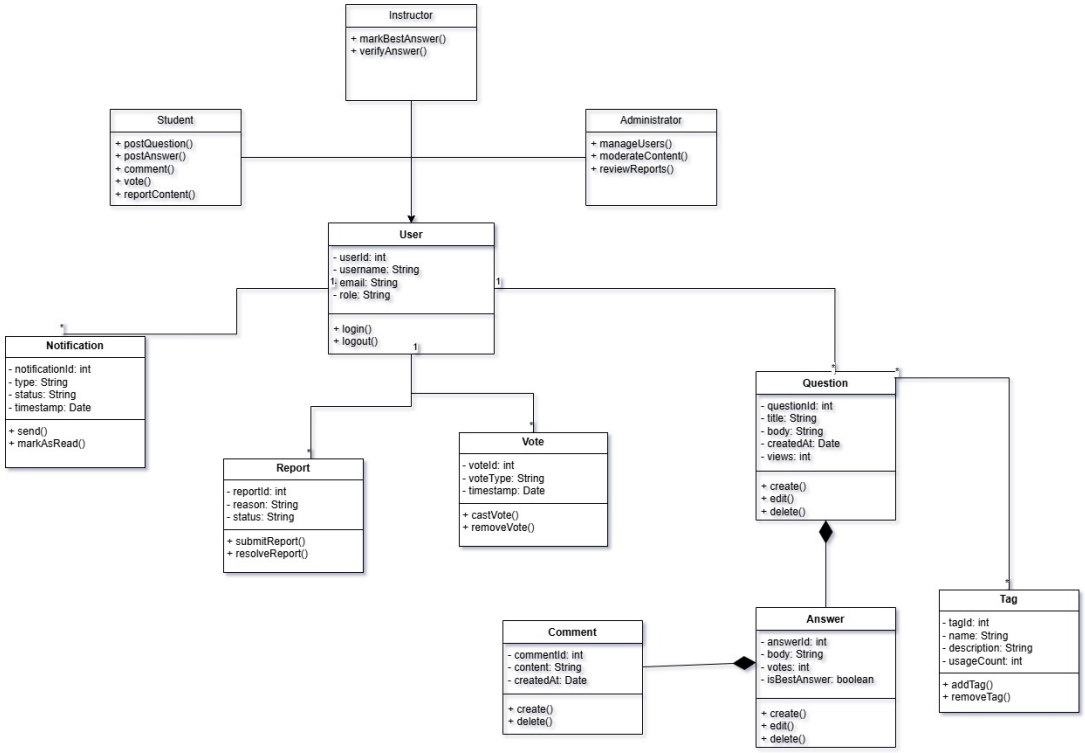

Figure 2: System Class Diagram illustrating User, Question, Answer, Tag, Notification, Vote, Comment, and Report classes with relationships.

8.3 Sequence Diagrams

8.3.1 Edit Post Flow

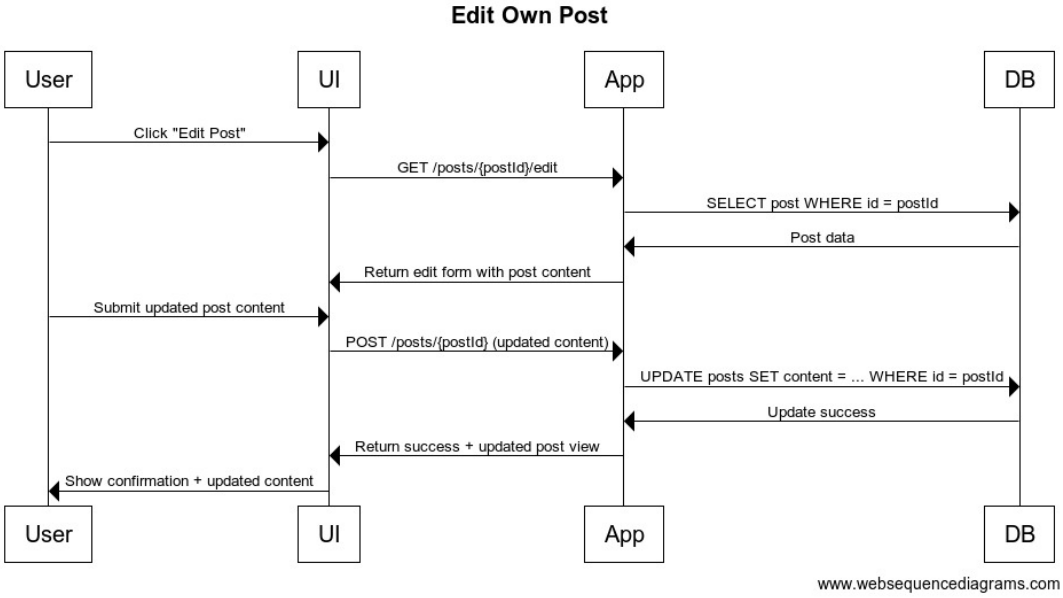

Figure 3: Sequence Diagram illustrating the User Editing a Question.

8.3.2 Notification Flow

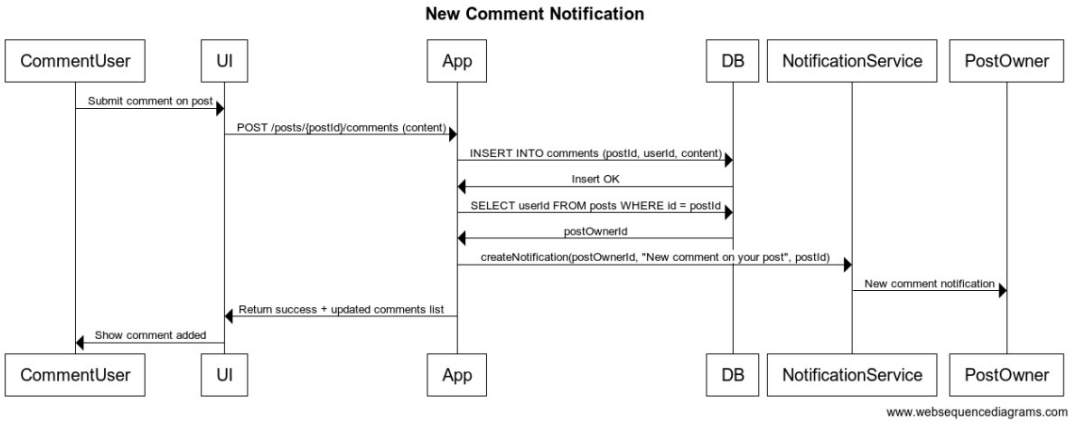

Figure 4: Sequence Diagram illustrating Notification Flow.

8.3.3 Ban User Flow

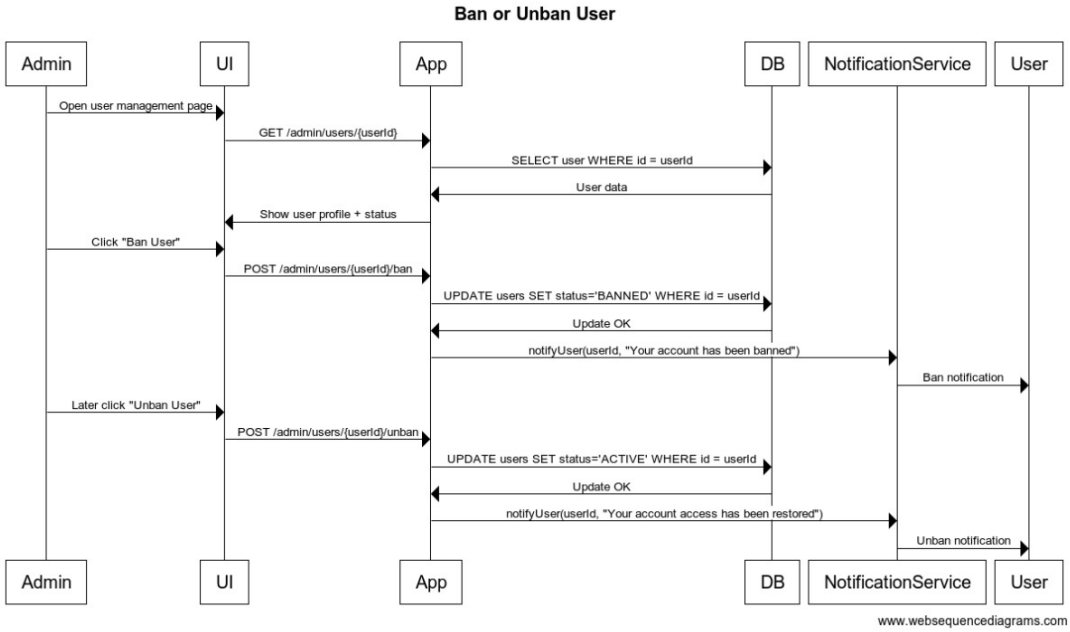

Figure 5: Sequence Diagram illustrating User Ban Flow.

8.3.4 Report Flow

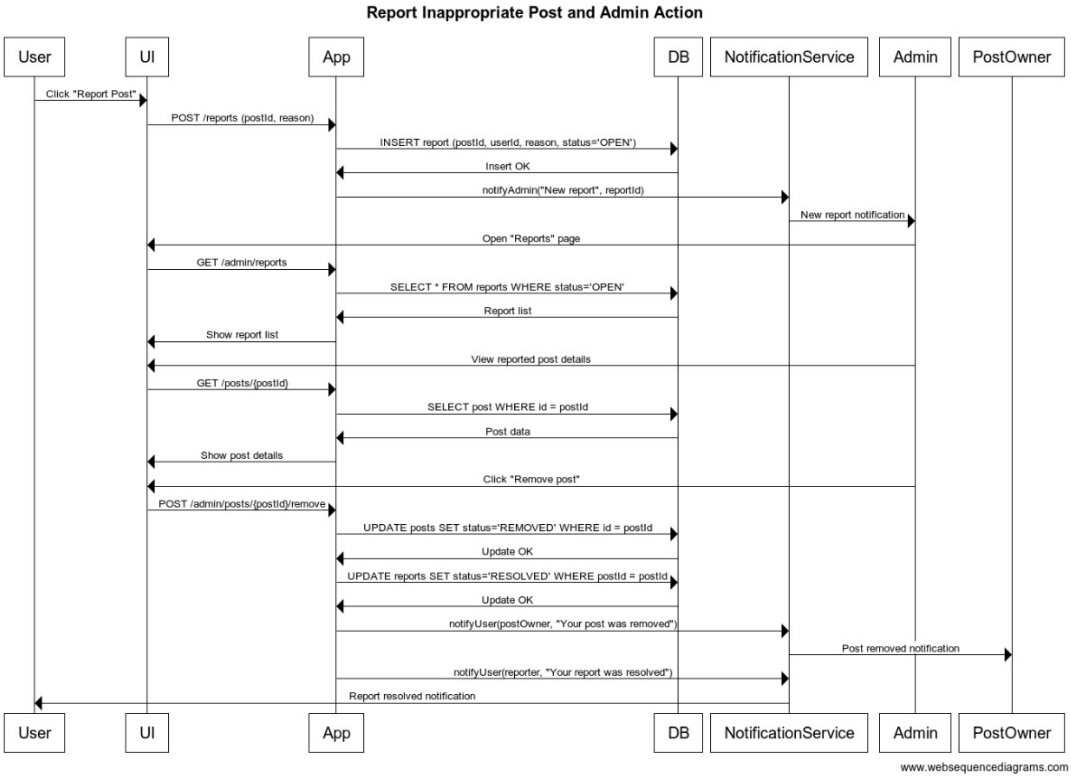

Figure 6: Sequence Diagram illustrating Report Flow.

8.4 Activity Diagram

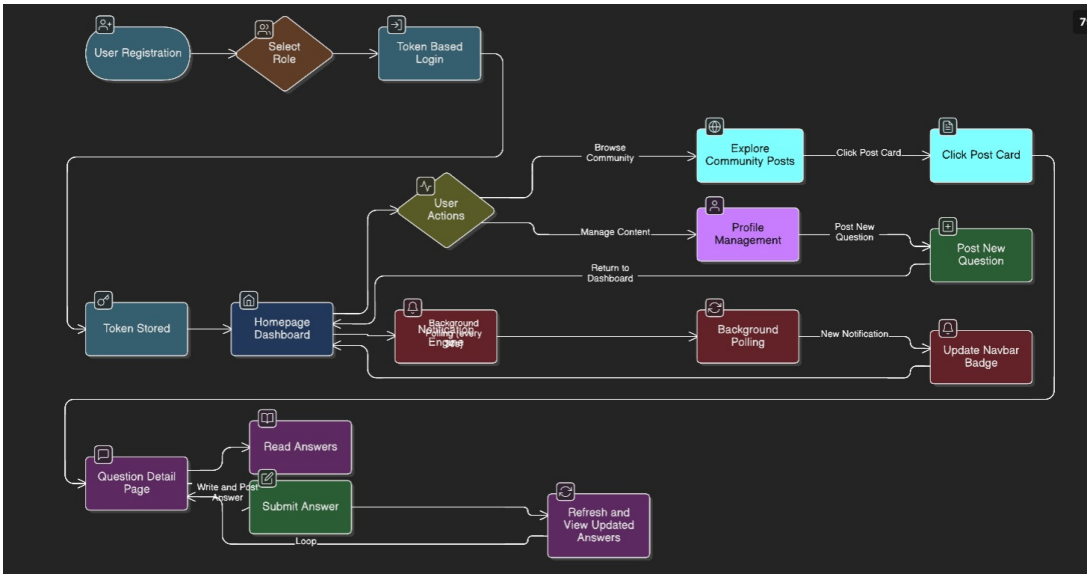

Figure 7: Activity Diagram Showing Overall Website Flow.

8.5 General API Diagram

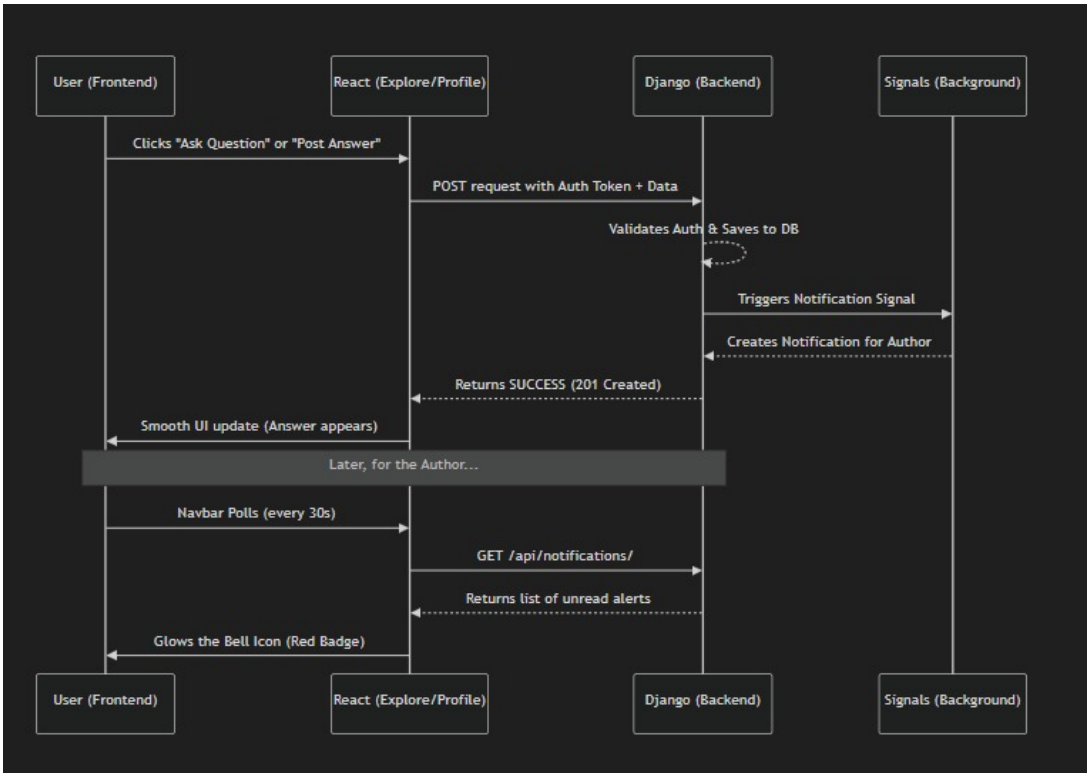

Figure 8: Graph Illustrating the API Flow within the Website.