ISYS3413

Assignment 2: Team-Based

Kasup Wellage | s4074242

Kurt Clado | s4003781

Timothy Nancarrow | s3950562

Contents

1. Use Case Diagram

1.1. Use Case Diagram for CodeQA

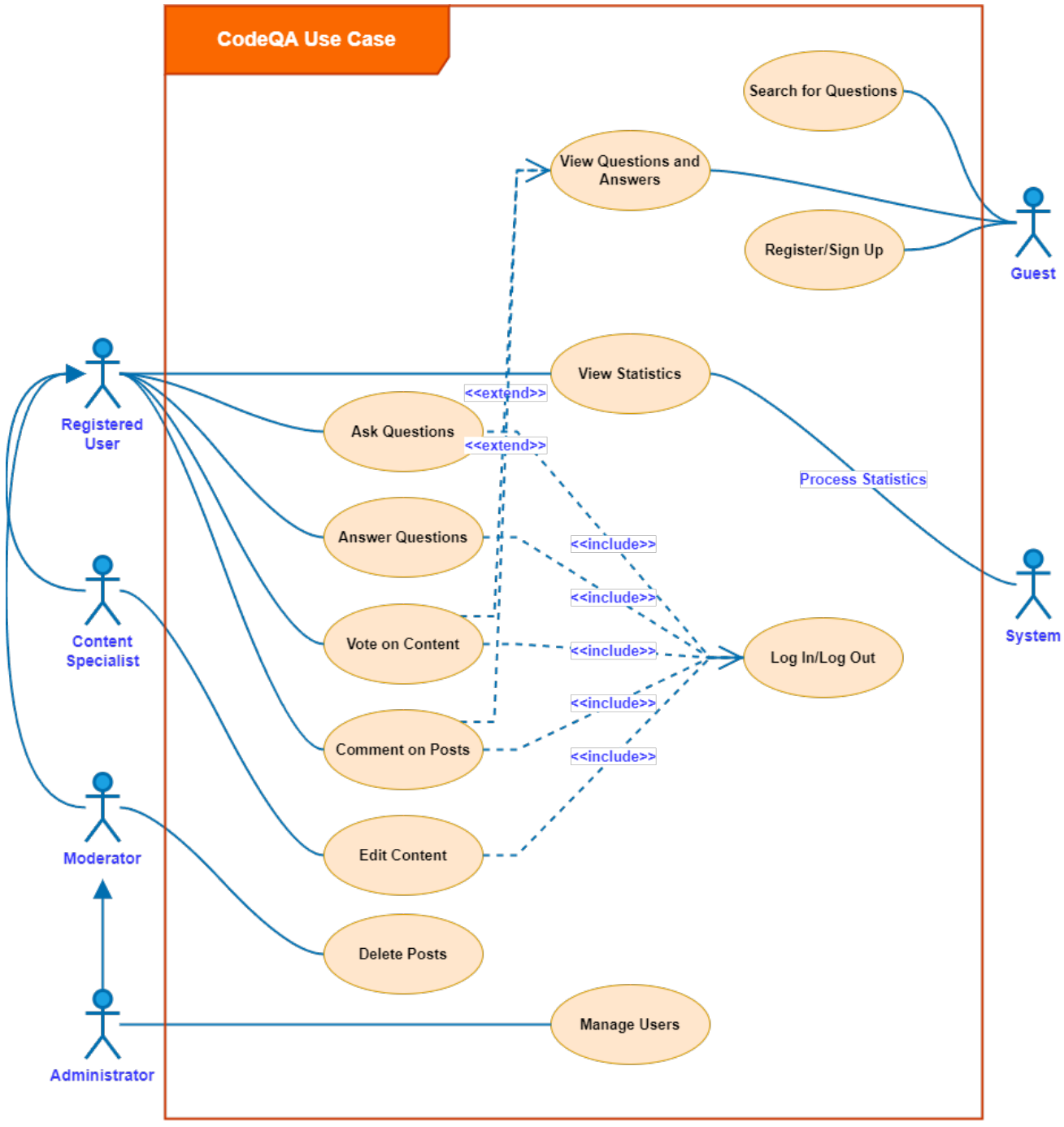


Figure 1: Use Case Diagram of CodeQA, generated from Draw.io

1.2. Details of Use Case Diagram

1.2.1. Actors

- **Guest/Unregistered User -** This actor represents users that are not logged into CodeQA or have yet to create an account.
- **Registered User -** Actor covers users that have successfully logged into CodeQA with their account, allowing core features to be view and accessed.
- **Administrator -**This actor manages the overall system, settings, and oversees user roles allocation.
- **Moderator -** The moderator actor is lower on the hierarchy in contrast to an administrator, they oversee content moderation to ensure content is relevant and within codes of use parameters.
- **Coding Specialist -** This actor enhances the quality and relevance of content posted, a specialist in a coding language; they can edit content or edit tags.
- **System** - CodeQA's system automates many of the backend processes such as statistic gathering and processing, notifications, data handling and storage, etc.

1.2.2. Use Case

User Management	Content Management	User Engagement	Analytics
Create/Sign Up	Search for Questions	Vote on Content	View Statistics
Manage Users	View Questions and Answers	Comment on Posts	
Login/Logout	Post Question		
	Answer Questions		
	Edit Content		
	Delete Posts		

1.2.3. Relationships

- Inclusion (<<include>>) relationships are a mandatory of other cases:
  - **Login/Logout** includes:
    - **Post Question** as users must be logged in to post questions.
    - **Answer Questions** as answering questions requires that users are logged in.
    - **Vote on Content** as users must be logged in to up or down vote.
    - **Comment on Posts** requires that users are logged in.
    - **Editing content** is allowed only after user access is completed.
- Extension relationships (<<extend>>) designate optional cases:
  - **View Questions and Answers** extends**:**
    - **Vote on Content** as users choose to vote on content, but this is optional.
    - **Comment on Posts** as users choose to comment on content, but this is optional.
- Generalisation is the inheritance present:
  - **Registered User** is generalised by:
    - **Content Specialist** and **Moderator** as both inherit the attributes of **Registered Users**.
  - **Moderator is further specialised by:**
    - **Administrator** as it inherits all **Moderator** capabilities and additional functions.

2. Use Case Descriptions

2.1. Use Case of Register/Sign Up

[Use Case] Register/Sign Up	
[Use Case ID]	CQA-001
[Brief Description]	Aims to allows guest users to create an account on **CodeQA**, enabling them to participate in functions such as asking questions, answering, voting, and commenting.
[Primary Actors]	• Guest
[Secondary Actors]	• System
[Preconditions]	• User mustn't already be registered with CodeQA. • System is operational and accessible for the user.
[Main Flow]	1. The user navigates to the **Sign-Up** page on CodeQA. 2. The user chooses a unique username. 3. The user enters a *valid* email address. 4. The user creates a password that meets security requirements. 5. The user completes additional requirements such as agreeing to terms of service. 6. The user submits the registration form. 7. The system validates the provided information. 8. The system starts a profile for the new user and writes the registration details. 9. The system sends verification email to the user by the provided email address. 10. The user verifies their email address by clicking on the verification link within the sent the email. 11. Upon successful email verification the system logs the user in automatically.
[Postconditions]	• The user completes process and is registered, verified, and logged into the system with a user profile.
[Alternative Flows]	1. If the email address is already assigned with another user or account, the system prompts user requesting for a different email. 2. If the username has already been created by another, the system prompts the user to enter a different username. 3. If the user fails to verify via email within a timeframe, the system suspense the account.

2.2. Use Case for Ask Questions

[Use Case] Post Question	
[Use Case ID]	CQA-002
[Brief Description]	Allows a verified user to post questions to CodeQA, allowing users to participate with the main function of the platform.
[Primary Actors]	• Registered User
[Secondary Actors]	• System
[Preconditions]	• User must be logged into their account. • System must be operational and accessible.
[Main Flow]	1. The user navigates to the **Post Question** section on CodeQA. 2. The user clicks on the **Create New Question** button. 3. The user is presented with a form to enter the question title and content. 4. The user enters the question title and the content. 5. The user selects tags from a list to categorise the question. 6. The user reviews the question. 7. The user submits the question by clicking **Post Question** button. 8. The system validates the question and ensures it adherence to guidelines. 9. The system writes the question to a database and makes it visible on CodeQA. 10. The system notifies the user that the question has been posted.
[Postconditions]	• The question is posted and viewable publicly. • The user receives confirmation that the posting was successful.
[Alternative Flows]	1. If the form is submitted incomplete, the system displays an error message, prompting the user to complete in the missing fields. 2. If the tags selected do not meet the standards on T&Cs, the system asks the user to select other tags. 3. If the system is not operational when the user tries to submit the question, the user receives a prompt asking them to try later.

2.3. Use Case of Register/Sign Up

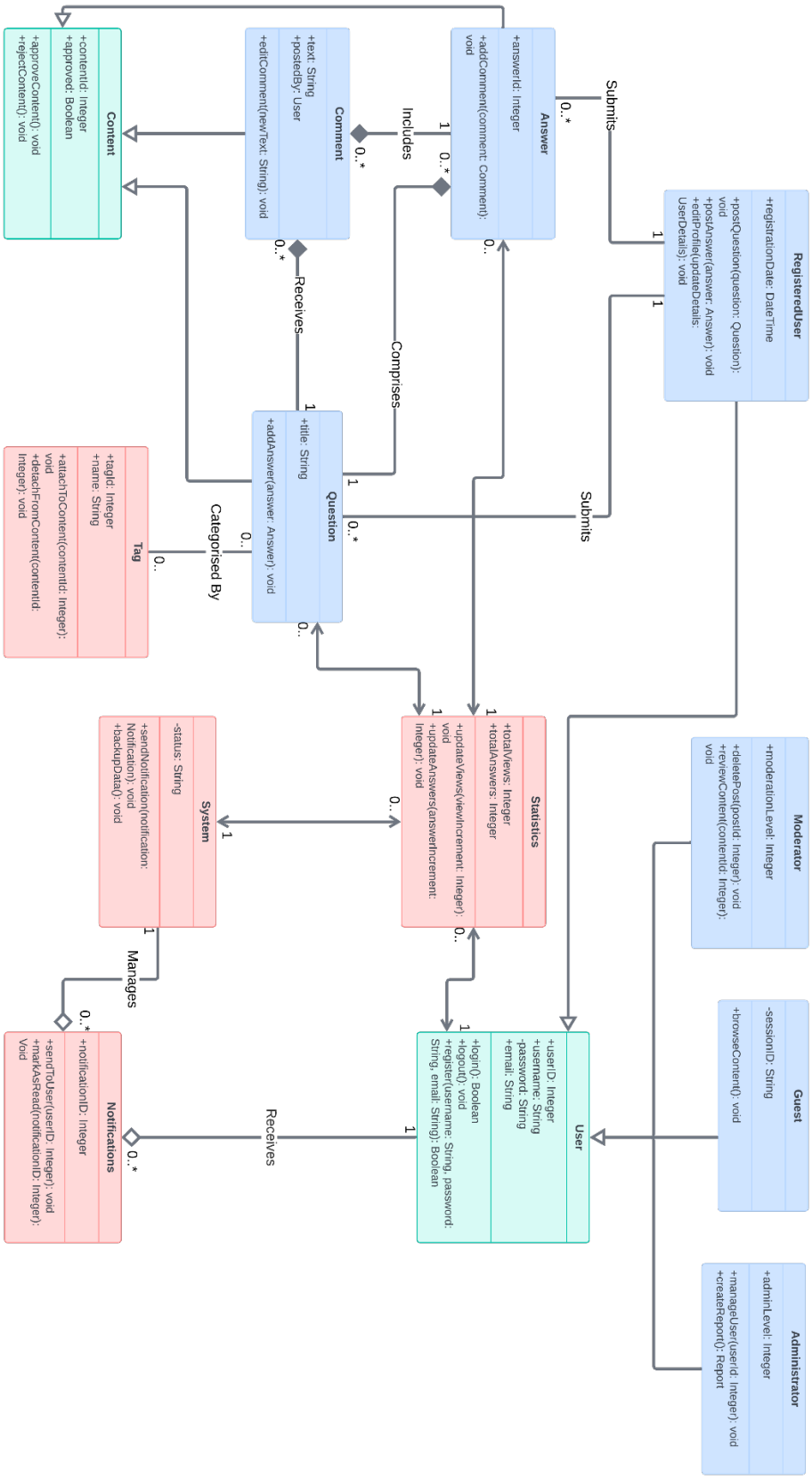
[Use Case] Delete Posts	
[Use Case ID]	CQA-003
[Brief Description]	Allows moderators to remove posts that are inappropriate, irrelevant, or violate the T&Cs, ensuring the quality and safety.
[Primary Actors]	• Moderator
[Secondary Actors]	• System
[Preconditions]	• Moderator must be logged into account. • Moderator has identified a post that violates the guidelines.
[Main Flow]	1. The moderator navigates to the post in question within the platform interface. 2. The moderator reviews the post and its compliance with the T&Cs. 3. The moderator clicks the **Delete** button linked with the post. 4. The system prompts the moderator to confirm the deletion to avoid unintended deletions. 5. The moderator confirms the deletion. 6. The system removes the post from public view. 7. The system logs the deletion action; post details and the moderator for record keeping. 8. The system sends a notification to the user to whom created the deleted post, advising of the deletion and any reason. 9. The moderator is redirected to the previous page and receives confirmation that the post has been successfully deleted.
[Postconditions]	• The post is no longer visible or accessible on the platform. • The original poster is informed of the deletion and the reason (if available).
[Alternative Flows]	1. If the moderator accidentally clicks the **Delete** button, they can cancel the action during the confirmation step. 2. If due to a system fault the deletion fails, the moderator is prompted with an error message to retry the deletion or report the issue for further support. 3. If a moderator without sufficient privileges attempts to delete a post, the system denies the action and logs the attempt for security monitoring.

2.4. Use Case for View Statistics

[Use Case] View Statistics	
[Use Case ID]	CQA-004
[Brief Description]	Allows users to access or view statistical information within **CodeQA**, including their content reach and overall site metrics. Different levels of detail are available based on user roles, i.e., administrators accessing the most thorough data.
[Primary Actors]	• Registered User
[Secondary Actors]	• Administrator • System
[Preconditions]	• User must be logged into their account. • System is operational and data processed is up to date.
[Main Flow]	1. The user navigates to the **Statistics** section by using CodeQAs menu. 2. The user selects the type of statistics to view, content performance or engagement metrics. 3. The system verifies the user's role to decide the level of detail and range available to requesting user. 4. System fetches statistical data from the database. 5. The system displays the statistics in a user-friendly format, such as graphs or summary tables. 6. The user reviews the displayed statistics, which may include total views, likes, comments, most active times, and other relevant metrics. 7. The user may choose to filter or adjust the data range to refine the displayed statistics. 8. If the user is an administrator, additional options for downloading or exporting the data to be available. 9. The user exits the statistics view when finished.
[Postconditions]	• The user gains insights from the viewed statistics, potentially influencing future interactions or content creation on the platform.
[Alternative Flows]	• If a user attempts to access statistics not permitted for their role, the system displays an error message and does not show the restricted data. • If the requested statistics are temporarily unavailable due to system maintenance or updates, the system informs the user of the issue and may suggest when to try again. • If a user requests statistics outside the usual reporting range, the system may need additional time to compile these, or it might instruct the user on how to make a special request.

3. Class Diagram

Figure 2: Class Diagram of CodeQA, generated from LucidChart.



4. Code

```
// -- User Class Hierarchy --
public abstract class User {
    private int userID;
    protected String username;
    private String password;
    protected String email;

    public User(int userID, String username, String password, String email) {
        this.userID = userID;
        this.username = username;
        this.password = password;
        this.email = email;
    }

    // Getter for userID
    public int getUserID() {
        return userID;
    }

    // Getter for username
    public String getUsername() {
        return username;
    }
    // Getter for email
    public String getEmail() {
        return email;
    }

    public abstract boolean login(String username, String password);
    public abstract void logout();
    public static boolean register(String username, String password, String email) {
        return true;
    }}

public class Guest extends User {
    private String sessionID;
    public Guest(int userID, String username, String password, String email) {
        super(userID, username, password, email);
    }

    public void browseContent() {
        // Code: Allows a user without an account to browse the site's content
```

```
47.     }}
48.
49. public class RegisteredUser extends User {
50.     protected DateTime registrationDate;
51.     public RegisteredUser(int userID, String username, String password,
    String email, DateTime registrationDate) {
52.         super(userID, username, password, email);
53.         this.registrationDate = registrationDate;
54.     }
55.
56.     public void postQuestion(Question question) {
57.         // Code: Posts a user's question
58.     }
59.
60.     public void postAnswer(Answer answer) {
61.         // Code: Posts a user's answer
62.     }}
63.
64. public class Moderator extends RegisteredUser {
65.     protected int moderationLevel;
66.     public Moderator(int userID, String username, String password, String
    email, DateTime registrationDate, int moderationLevel) {
67.         super(userID, username, password, email, registrationDate);
68.         this.moderationLevel = moderationLevel;
69.     }
70.
71.     public void deletePost(int postId) {
72.         // Code here: to delete a post
73.     }}
74.
75. public class Administrator extends Moderator {
76.     protected int adminLevel;
77.
78.     public Administrator(int userID, String username, String password,
    String email, DateTime registrationDate, int moderationLevel, int
    adminLevel) {
79.         super(userID, username, password, email, registrationDate,
    moderationLevel, adminLevel);
80.     }
81.
82.     public void manageUser(int userId) {
83.         // Code here for managing users
84.     }
85.
86.     public Report createReport() {
87.         // Code here to create a report
88.         return new Report();
89.     }}
```

```
90. // -- CONTENT CLASSES --
91. public abstract class Content {
92.     protected int contentId;
93.     protected boolean approved;
94.
95.     public Content(int contentId) {
96.         this.contentId = contentId;
97.         this.approved = false;
98.     }
99.
100.    public void approveContent() {
101.        this.approved = true;
102.    }
103.
104.    public void rejectContent() {
105.        this.approved = false;
106.    }}
107.
108.public class Question extends Content {
109.    protected String title;
110.    private List<Answer> answers = new ArrayList<>();
111.
112.    public Question(int contentId, String title) {
113.        super(contentId);
114.        this.title = title;
115.    }
116.
117.    public void addAnswer(Answer answer) {
118.        answers.add(answer);
119.    }}
120.
121.public class Answer extends Content {
122.    private List<Comment> comments = new ArrayList<>();
123.    public Answer(int contentId) {
124.        super(contentId);
125.    }
126.
127.    public void addComment(Comment comment) {
128.        comments.add(comment);
129.    }}
130.
131.public class Comment extends Content {
132.    protected String text;
133.    protected User postedBy;
134.
135.    public Comment(int contentId, String text, User postedBy) {
136.        super(contentId);
137.        this.text = text;
```

```
138.         this.postedBy = postedBy;
139.     }
140.
141.     public void editText(String newText) {
142.         this.text = newText;
143.     }
144. }
145.
146. // -- SUPPORTING CLASSES --
147. public class Notification {
148.     protected int notificationId;
149.     protected String message;
150.
151.     public void sendToUser(int userId) {
152.         // code to send notification to user
153.     }
154.
155.     public void markAsRead() {
156.         // code to change notification to read
157.     }
158. }
159.
160. public class System {
161.     private List<Notification> notifications = new ArrayList<>();
162.
163.     public void sendNotification(Notification notification) {
164.         notifications.add(notification);
165.     }
166.
167.     public void backupData() {
168.         // code for backup data
169.     }
170. }
171.
172. public class Statistics {
173.     protected int totalViews;
174.     protected int totalAnswers;
175.
176.     public void updateViews(int viewIncrement) {
177.         totalViews += viewIncrement;
178.     }
179.
180.     public void updateAnswers(int answerIncrement) {
181.         totalAnswers += answerIncrement;
182.     }
183. }
```

References

Anda, B., & Sjøberg, D. I. K. (2005). Investigating the Role of Use Cases in the Construction of Class Diagrams. *Empirical Software Engineering*, *10*(3), 285 309. https://doi.org/10.1007/s10664-005-1289-3

Anthropic. (2023). *Claude*. Claude.ai. https://claude.ai/chats

ChatGPT - Latest News and Chat About AI. (n.d.). Chatgpt.com. https://chatgpt.com/

Draw.io. (2024). *Flowchart Maker & Online Diagram Software*. App.diagrams.net. https://app.diagrams.net/

Liang, Y. (2003). From use cases to classes: a way of building object model with UML. *Information and Software Technology*, *45*(2), 83–93. https://doi.org/10.1016/s0950-5849(02)00164-7

Lucidchart. (2023). *Online diagram software & visual solution | Lucidchart*. Lucidchart. https://www.lucidchart.com/pages/

Stack Overflow. (2024). *Stack Overflow - Where Developers Learn, Share, & Build Careers*. Stack Overflow. https://stackoverflow.com/