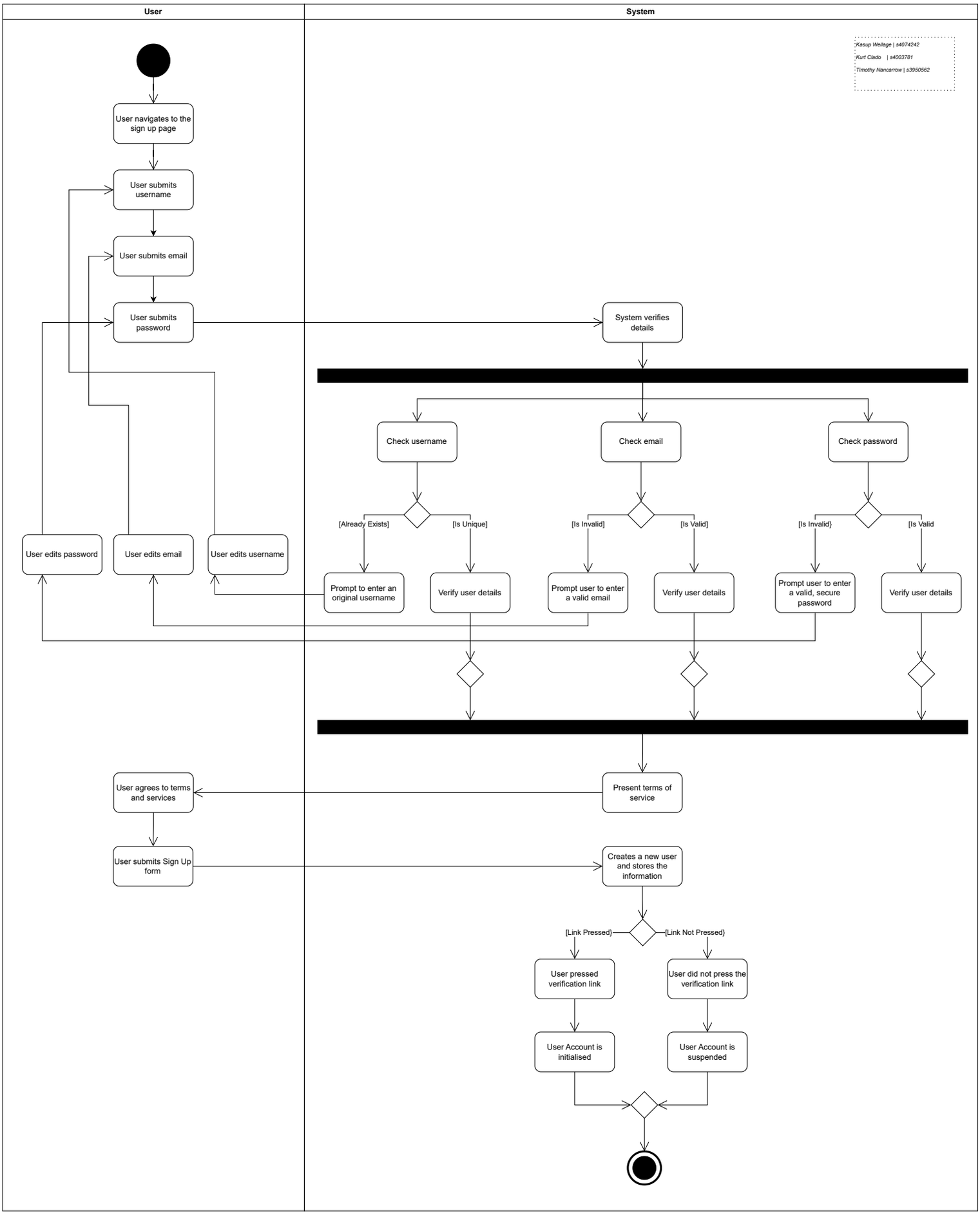
ISYS3413

Assignment 3: Team-Based

Kasup Wellage | s4074242

Kurt Clado | s4003781

Timothy Nancarrow | s3950562


User
System
Kasup Wellage | s4074242
Kurt Clado | s4003781
Timothy Nancarrow | s3950562
User navigates to the sign up page
User submits username
User submits email
User submits password
System verifies details
Check username
Check email
Check password
[Already Exists]
[Is Unique]
[Is Invalid]
[Is Valid]
[Is Invalid}
[Is Valid
User edits password
User edits email
User edits username
Prompt to enter an original username
Verify user details
Prompt user to enter a valid email
Verify user details
Prompt user to enter a valid, secure password
Verify user details
User agrees to terms and services
Present terms of service
User submits Sign Up form
Creates a new user and stores the information
[Link Pressed}
{Link Not Pressed}
User pressed verification link
User did not press the verification link
User Account is initialised
User Account is suspended

Kasup Wellage | s4074242

Kurt Clado | s4003781

Timothy Nancarrow | s3950562

User

Program

User navigates to Post Question tab

User presses Create New Question

Generate question form

Display question form

User writes title

User writes question content

User adds tags

User presses the Post Question button

[System is down]

Notify the user system is down and they should post their question later

[System operational]

Validate user content

User fixes post error

Validate question title

Validate question content

Validate question tags

[Has system error]

[Has validation error]

[Is valid]

[Has system error]

[Has validation error]

[Is valid]

[Has system error]

[Has validation error]

[Is valid]

Do not post and send user a validation error message

Do not post and send user a validation error message

Do not post and send user a validation error message

Do not post and send user a validation error message

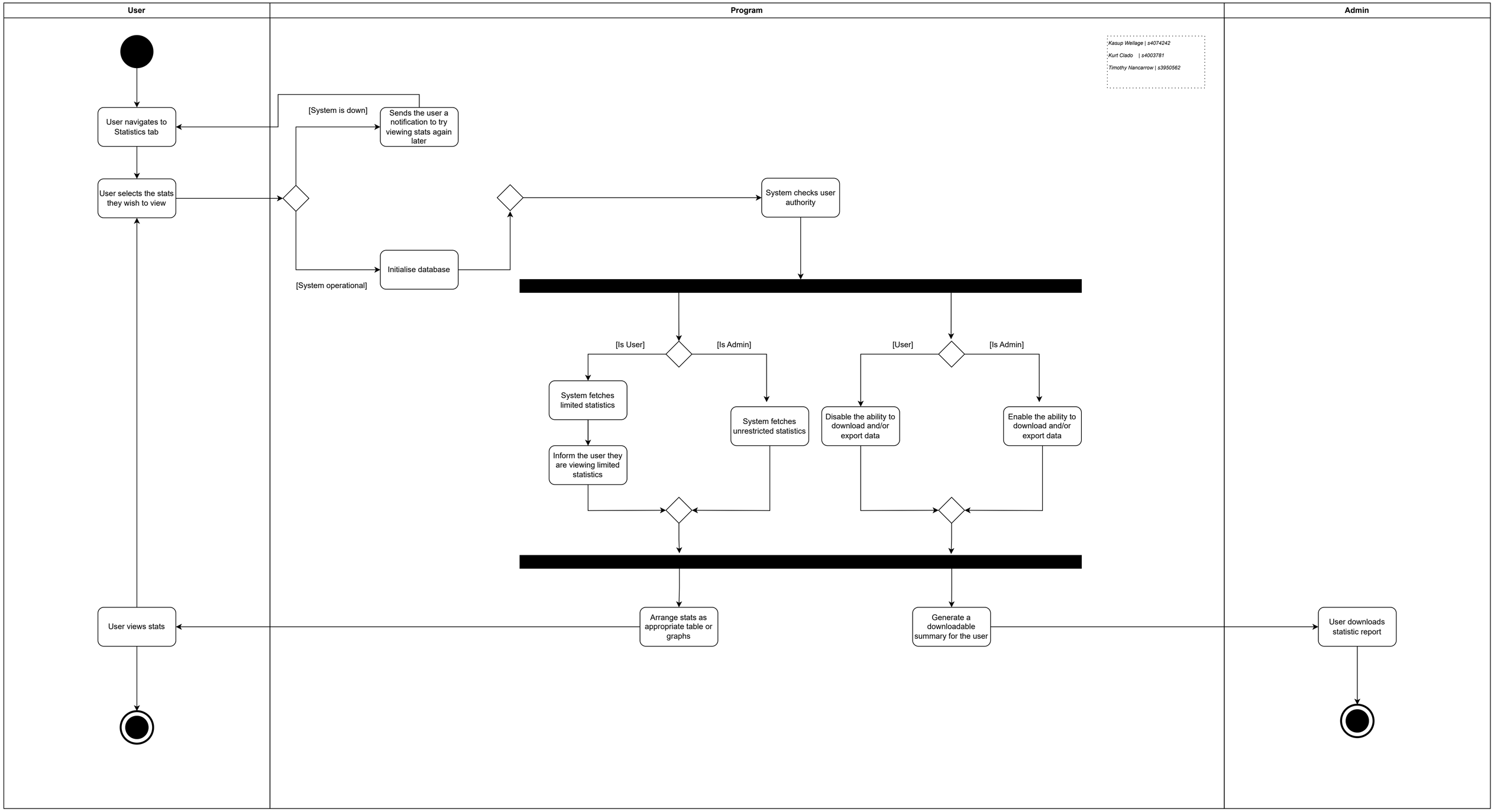
Do not post and send user a validation error message

Do not post and send user a validation error message

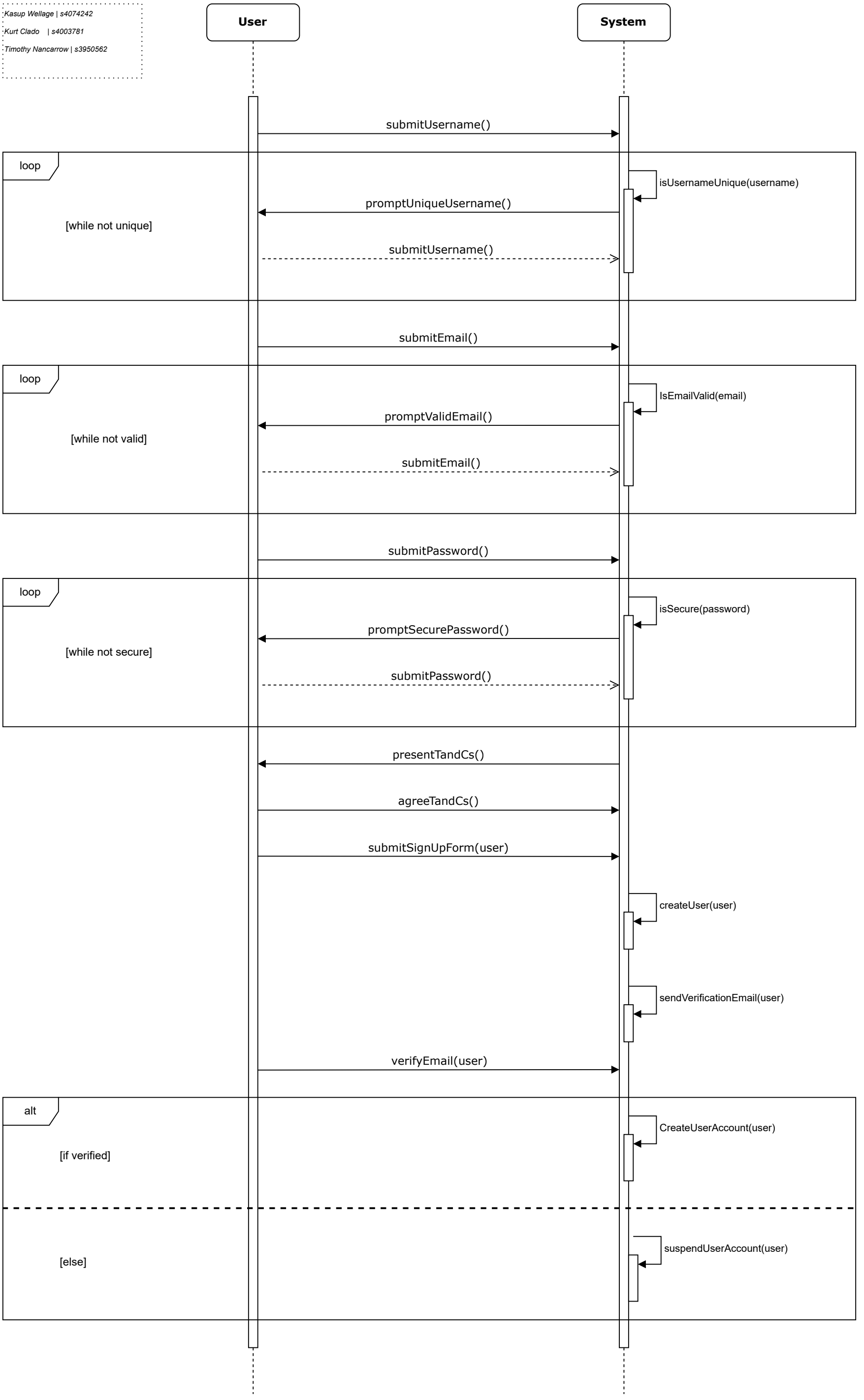
Write post to database

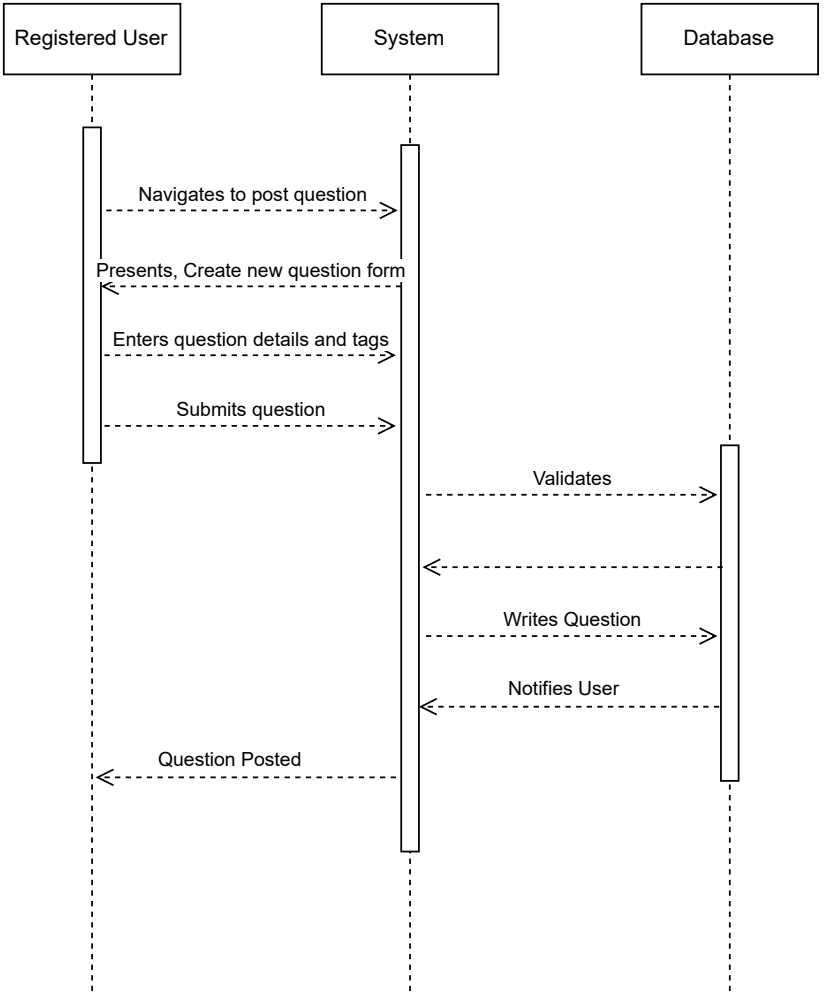
Set post to visible on the CodeQA site

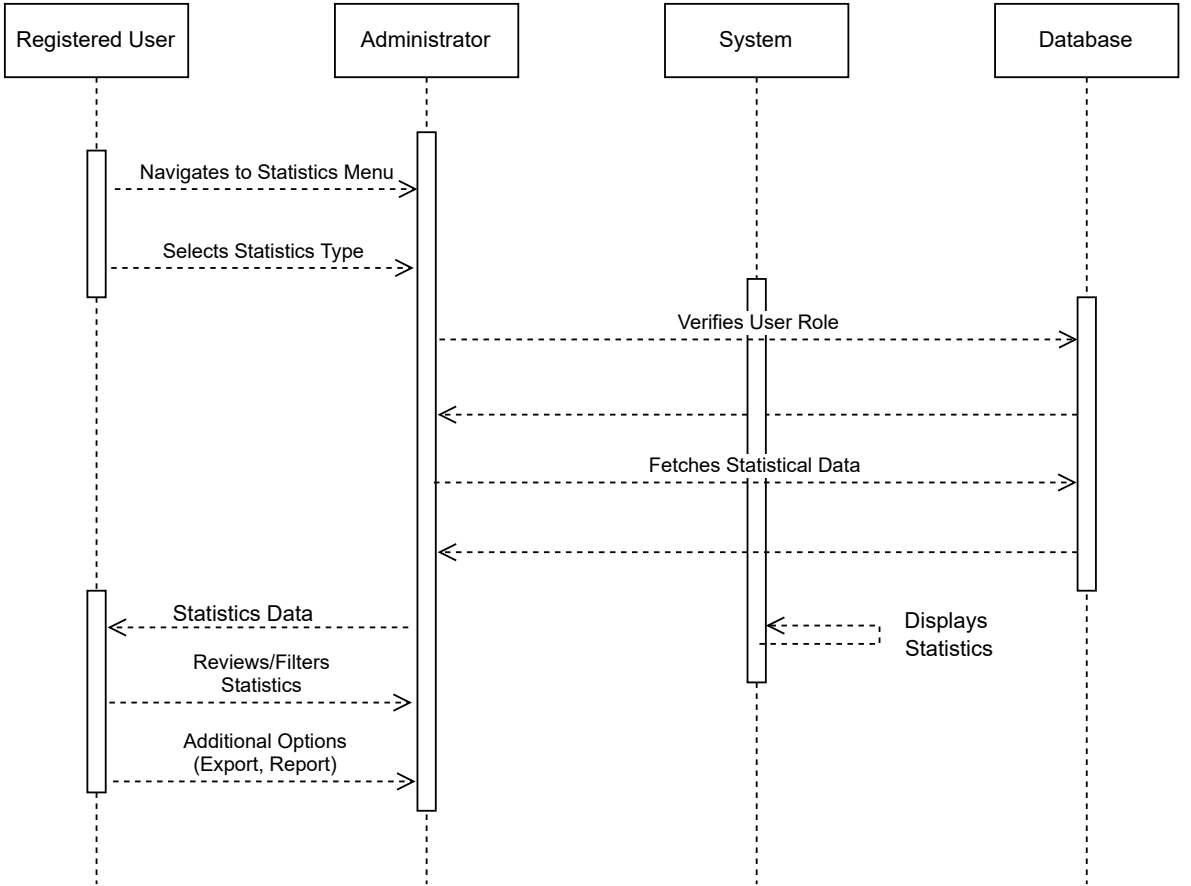
Notify user their post was successful


User
Program
Admin
Kasup Wellage | s4074242
Kurt Clado | s4003781
Timothy Nancarrow | s3950562
User navigates to Statistics tab
User selects the stats they wish to view
[System is down]
Sends the user a notification to try viewing stats again later
Initialise database
[System operational]
System checks user authority
[Is User]
[Is Admin]
[User]
[Is Admin]
System fetches limited statistics
Inform the user they are viewing limited statistics
System fetches unrestricted statistics
Disable the ability to download and/or export data
Enable the ability to download and/or export data
Arrange stats as appropriate table or graphs
Generate a downloadable summary for the user
User views stats
User downloads statistic report

Kasup Wellage | s4074242
Kurt Clado | s4003781
Timothy Nancarrow | s3950562




Registered User
System
Database
Navigates to post question
Presents, Create new question form
Enters question details and tags
Submits question
Validates
Writes Question
Notifies User
Question Posted


Registered User
Administrator
System
Database
Navigates to Statistics Menu
Selects Statistics Type
Verifies User Role
Fetches Statistical Data
Statistics Data
Displays Statistics
Reviews/Filters Statistics
Additional Options (Export, Report)

3. Class Diagram

3.1. Skeleton Code for Sign Up

```
4.  // Class that shows the process for a user to sign up on CodeQA
5.  public class UserSignUp {
6.      public static void main(String[] args) {
7.          UserSignUp signupProcess = new UserSignUp();
8.          signupProcess.startSignUp();
9.      }
10.
11.     // sign-up process by creating a user object.
12.     public void startSignUp() {
13.         User user = new User();
14.
15.         // [1] user submits username
16.         user.setUsername(submitUsername());
17.
18.         // [2] check if username is unique
19.         while (!isUsernameUnique(user.getUsername())) {
20.             user.setUsername(promptUniqueUsername());
21.         }
22.
23.         // [3] user submits email
24.         user.setEmail(submitEmail());
25.
26.         // [4] check if email is valid
27.         while (!isEmailValid(user.getEmail())) {
28.             user.setEmail(promptValidEmail());
29.         }
30.
31.         // [5] user submits password
32.         user.setPassword(submitPassword());
33.
34.         // [6] check if password is secure
35.         while (!isSecure(user.getPassword())) {
36.             user.setPassword(promptForSecurePassword());
37.         }
38.
39.         // [7] present T&Cs
40.         presentTandCs();
41.
42.         // [8] user agrees to terms and services
43.         if (agreeToTerms()) {
44.             // [9] user submits Sign-Up form
45.             submitSignUpForm(user);
46.
47.             // [10] create new user and store information
```

```
48.             createUser(user);
49.
50.             // [11] send verification email
51.             sendVerificationEmail(user);
52.
53.             // [12] user verifies their email
54.             if (verifyEmail(user)) {
55.                 // [13] create user account
56.                 CreateUserAccount(user);
57.             } else {
58.                 // User did not press the verification link
59.                 suspendUserAccount(user);
60.             }}}
61.
62.     // skeleton for the user submitting a username.
63.     // this method returns the username.
64.     private String submitUsername() {
65.         return ""; // user input
66.     }
67.
68.     // checks if the username is unique.
69.     // this method returns true if the username is unique, false if not.
70.     private boolean isUsernameUnique(String username) {
71.         // Placeholder for actual check
72.         return !username.equals("existingUsername");
73.     }
74.
75.     // prompts the user to enter a unique username.
76.     // returns a new unique username.
77.     private String promptUniqueUsername() {
78.         return "newUniqueUsername"; // Placeholder for user input
79.     }
80.
81.     // skeleton for the user submitting an email.
82.     // returns a email.
83.     private String submitEmail() {
84.         return "user@example.com"; // Placeholder for user input
85.     }
86.
87.     // checks if the email is valid.
88.     // returns true if the email is valid, false if not.
89.     private boolean isEmailValid(String email) {
90.         // Placeholder for actual check
91.         return email.contains("@");
92.     }
93.
94.     // prompts the user to enter a valid email.
95.     // returns a valid email.
```

```
96.     private String promptValidEmail() {
97.         return ""; // user input
98.     }
99.
100.    // skeleton for the user submitting a password.
101.    // returns a password.
102.    private String submitPassword() {
103.        return ""; // Placeholder for user input
104.    }
105.
106.    // checks if the password is secure.
107.    // returns true if the password is secure, false otherwise.
108.    private boolean isSecure(String password) {
109.        // actual check placeholder
110.        return password.length() > 6;
111.    }
112.
113.    // prompts the user to enter a secure password.
114.    // returns a secure password.
115.    private String promptForSecurePassword() {
116.        return "securePassword"; // Placeholder for user input
117.    }
118.
119.    // skeleton for presenting the T&Cs to the user.
120.    private void presentTandCs() {
121.        // Placeholder
122.    }
123.
124.    // skeleton for the user agreeing to the terms of service.
125.    // returns true for agree/confirmed.
126.    private boolean agreeToTerms() {
127.        return true;
128.    }
129.
130.    // skeleton for submitting the sign-up form.
131.    private void submitSignUpForm(User user) {
132.        // Placeholder for form submission
133.    }
134.
135.    // skeleton for creating a new user and storing their information.
136.    private void createUser(User user) {
137.        // Placeholder for creating user
138.    }
139.
140.    // skeleton for sending a verification email to the user.
141.    private void sendVerificationEmail(User user) {
142.        // Placeholder for sending email
143.    }
```

```
144.
145.     // skeleton for user verifying their email.
146.     // returns true to indicate successful verification.
147.     private boolean verifyEmail(User user) {
148.         return true; // Placeholder for actual verification
149.     }
150.
151.     // skeleton for creating the user account.
152.     private void CreateUserAccount(User user) {
153.         // Placeholder
154.     }
155.
156.     // skeleton for suspending the user's account.
157.     private void suspendUserAccount(User user) {
158.         // suspending account
159.     }
160.
161.     // class of the user's information.
162.     // holds the username, email, and password of the user.
163.     class User {
164.         private String username;
165.         private String email;
166.         private String password;
167.
168.         public String getUsername() {
169.             return username;
170.         }
171.
172.         public void setUsername(String username) {
173.             this.username = username;
174.         }
175.
176.         public String getEmail() {
177.             return email;
178.         }
179.
180.         public void setEmail(String email) {
181.             this.email = email;
182.         }
183.
184.         public String getPassword() {
185.             return password;
186.         }
187.
188.         public void setPassword(String password) {
189.             this.password = password;
190.         }
191.     }}
```

3.2. Skeleton Code for Post Question

```
// Class that shows the process for a user to post a question on CodeQA

public class UserPostQuestion {

    public static void main(String[] args) {
        UserPostQuestion postProcess = new UserPostQuestion();
        postProcess.start();
    }

    // Starts by creating a question object.
    // initiating the flow of creating and posting a question.
    public void start() {
        User user = new User("exampleUsername");
        Question question = new Question();

        // [1] user creates a question
        // the user sets the title and content for their question.
        question.setTitle(getTitle());
        question.setContent(getContent());

        // [2] system checks if the question is valid
        // to ensure it has a title and content.
        if (checkValidQuestion(question)) {
            // [3] user submits the question
            // If valid, the user submits the question.
            submitQuestion(user, question);

            // [4] system saves the question to the database.
            saveQuestion(question);

            // [5] System confirms the question is posted
            // The system confirms that the question has been posted successfully.
            confirmPosted(question);
        } else {
            // If the question is invalid, prompt the user to edit it
            // If the question is invalid, the user is prompted to make corrections.
            promptEditQuestion();
        }
    }

    // user creates a question title.
    // returns a title for the question.
    private String getTitle() {
        System.out.println("What is the question title?");
        return "";
```

```
49.     }
50.
51.     // user writing question content.
52.     // returns content for the question.
53.     private String getContent() {
54.         System.out.println("Write your question...");
55.         return "";
56.     }
57.
58.     // checks if the question is valid.
59.     // is valid if it is not empty with title and content.
60.     private boolean checkValidQuestion(Question question) {
61.         System.out.println("Checking if the question is valid...");
62.         return question.getTitle() != null &&
    !question.getTitle().isEmpty() &&
63.                 question.getContent() != null &&
    !question.getContent().isEmpty();
64.     }
65.
66.     // user submitting the question.
67.     // prints the username and the question title being submitted.
68.     private void submitQuestion(User user, Question question) {
69.         System.out.println(user.getUsername() + " posting your question
    to the community - " + question.getTitle());
70.     }
71.
72.     // saving the question in the database.
73.     // prints the title of the question being saved.
74.     private void saveQuestion(Question question) {
75.         System.out.println("Saving " + question.getTitle());
76.     }
77.
78.     // confirms the question has been posted.
79.     // prints a confirmation message with the question title.
80.     private void confirmPosted(Question question) {
81.         System.out.println("Your question '" + question.getTitle() + "'
    has been posted!");
82.     }
83.
84.     // prompts the user to edit their question if it is invalid.
85.     // prints a message prompting the user to edit their question.
86.     private void promptEditQuestion() {
87.         System.out.println("The question is not able to be posted.
    Please edit your question and try again.");
88.     }
89.
90.     // class of the user's information.
91.     // holds the username of the user.
```

```
92.     class User {
93.         private String username;
94.
95.         public User(String username) {
96.             this.username = username;
97.         }
98.
99.         public String getUsername() {
100.                return username;
101.            }
102.
103.            public void setUsername(String username) {
104.                this.username = username;
105.            }
106.        }
107.
108.        // class of the question information.
109.        // holds the title and content of the question.
110.        class Question {
111.            private String title;
112.            private String content;
113.
114.            public String getTitle() {
115.                return title;
116.            }
117.
118.            public void setTitle(String title) {
119.                this.title = title;
120.            }
121.
122.            public String getContent() {
123.                return content;
124.            }
125.
126.            public void setContent(String content) {
127.                this.content = content;
128.            }
129.        }
130.    }
```

192. Code

```
193. // -- User Class Hierarchy --
194. public abstract class User {
195.     private int userID;
196.     protected String username;
197.     private String password;
198.     protected String email;
199. 
200.     public User(int userID, String username, String password, String
    email) {
201.         this.userID = userID;
202.         this.username = username;
203.         this.password = password;
204.         this.email = email;
205.     }
206. 
207.     // Getter for userID
208.     public int getUserID() {
209.         return userID;
210.     }
211. 
212.     // Getter for username
213.     public String getUsername() {
214.         return username;
215.     }
216.     // Getter for email
217.     public String getEmail() {
218.         return email;
219.     }
220. 
221.     public abstract boolean login(String username, String password);
222.     public abstract void logout();
223.     public static boolean register(String username, String password,
    String email) {
224.         return true;
225.     }}
226. 
227. public class Guest extends User {
228.     private String sessionID;
229.     public Guest(int userID, String username, String password, String
    email) {
230.         super(userID, username, password, email);
231.     }
232. 
233.     public void browseContent() {
234.         // Code: Allows a user without an account to browse the site's
    content
```

```
235.     }}
236.
237. public class RegisteredUser extends User {
238.     protected DateTime registrationDate;
239.     public RegisteredUser(int userID, String username, String password,
     String email, DateTime registrationDate) {
240.         super(userID, username, password, email);
241.         this.registrationDate = registrationDate;
242.     }
243.
244.     public void postQuestion(Question question) {
245.         // Code: Posts a user's question
246.     }
247.
248.     public void postAnswer(Answer answer) {
249.         // Code: Posts a user's answer
250.     }}
251.
252. public class Moderator extends RegisteredUser {
253.     protected int moderationLevel;
254.     public Moderator(int userID, String username, String password, String
     email, DateTime registrationDate, int moderationLevel) {
255.         super(userID, username, password, email, registrationDate);
256.         this.moderationLevel = moderationLevel;
257.     }
258.
259.     public void deletePost(int postId) {
260.         // Code here: to delete a post
261.     }}
262.
263. public class Administrator extends Moderator {
264.     protected int adminLevel;
265.
266.     public Administrator(int userID, String username, String password,
     String email, DateTime registrationDate, int moderationLevel, int
     adminLevel) {
267.         super(userID, username, password, email, registrationDate,
     moderationLevel, adminLevel);
268.     }
269.
270.     public void manageUser(int userId) {
271.         // Code here for managing users
272.     }
273.
274.     public Report createReport() {
275.         // Code here to create a report
276.         return new Report();
277.     }}
```

```
// -- CONTENT CLASSES --
public abstract class Content {
    protected int contentId;
    protected boolean approved;

    public Content(int contentId) {
        this.contentId = contentId;
        this.approved = false;
    }

    public void approveContent() {
        this.approved = true;
    }

    public void rejectContent() {
        this.approved = false;
    }}

public class Question extends Content {
    protected String title;
    private List<Answer> answers = new ArrayList<>();

    public Question(int contentId, String title) {
        super(contentId);
        this.title = title;
    }

    public void addAnswer(Answer answer) {
        answers.add(answer);
    }}

public class Answer extends Content {
    private List<Comment> comments = new ArrayList<>();
    public Answer(int contentId) {
        super(contentId);
    }

    public void addComment(Comment comment) {
        comments.add(comment);
    }}

public class Comment extends Content {
    protected String text;
    protected User postedBy;

    public Comment(int contentId, String text, User postedBy) {
        super(contentId);
        this.text = text;
```

```
326.         this.postedBy = postedBy;
327.     }
328.
329.     public void editText(String newText) {
330.         this.text = newText;
331.     }
332. }
333.
334. // -- SUPPORTING CLASSES --
335. public class Notification {
336.     protected int notificationId;
337.     protected String message;
338.
339.     public void sendToUser(int userId) {
340.         // code to send notification to user
341.     }
342.
343.     public void markAsRead() {
344.         // code to change notification to read
345.     }
346. }
347.
348. public class System {
349.     private List<Notification> notifications = new ArrayList<>();
350.
351.     public void sendNotification(Notification notification) {
352.         notifications.add(notification);
353.     }
354.
355.     public void backupData() {
356.         // code for backup data
357.     }
358. }
359.
360. public class Statistics {
361.     protected int totalViews;
362.     protected int totalAnswers;
363.
364.     public void updateViews(int viewIncrement) {
365.         totalViews += viewIncrement;
366.     }
367.
368.     public void updateAnswers(int answerIncrement) {
369.         totalAnswers += answerIncrement;
370.     }
371. }
```