Design II : Use Case Algorithms

Emmet Ledell, Justin Huynh, Raman Khatri Chhetri, Jairen Cheek, Connor Marks, Hadi Khan

Algorithm for Use Case UC1: User Sign Up and Login

Algorithm 1: The createAccount() function

Input: User inputs for account creation: name, date of birth, username, password, and email
Output: User account created and logged in

```
1.1  User visits the create account page;
1.2  System displays a form for user details;
1.3  User inputs name, date of birth, username, and password;
1.4  User checks the terms and conditions box;
1.5  if checkbox is not checked then
1.6  |  Display error and prompt user to agree;
1.7  else
1.8  |  User submits the form;
1.9  |  if input is invalid then
1.10 |  |  Display error with details and prompt user to correct;
1.11 |  else
1.12 |  |  if username is taken then
1.13 |  |  |  Display error and prompt for a new username;
1.14 |  |  else
1.15 |  |  |  if password is too weak then
1.16 |  |  |  |  Display error and prompt user to strengthen password;
1.17 |  |  |  else
1.18 |  |  |  |  Prompt user for email;
1.19 |  |  |  |  if signing up with Google then
1.20 |  |  |  |  |  Open external browser for Google login;
1.21 |  |  |  |  |  Send verification code to Google account;
1.22 |  |  |  |  |  if verification code invalid or expired then
1.23 |  |  |  |  |  |  Display error and prompt user to retry;
1.24 |  |  |  |  |  else
1.25 |  |  |  |  |  |  Create and save user account;
1.26 |  |  |  |  else
1.27 |  |  |  |  |  Send verification code to provided email;
1.28 |  |  |  |  |  if verification code invalid or expired then
1.29 |  |  |  |  |  |  Display error and prompt user to retry;
1.30 |  |  |  |  |  else
1.31 |  |  |  |  |  |  Create and save user account;
```

Algorithm for Use Case UC2: Create Posts

Algorithm 2: The createPost(content) function

Input: Post details: title, text, optional media files
Output: Post created and displayed on the site

```
2.1  User initiates post creation;
2.2  System prompts user to select a community board;
2.3  if no community board selected then
2.4  |  Display error and prompt user to choose a board;
2.5  else
2.6  |  User fills in post details;
2.7  |  if required fields are missing then
2.8  |  |  Display error and prompt user to complete fields;
2.9  |  else
2.10 |  |  if media file type is unsupported then
2.11 |  |  |  Display error and prompt user to remove invalid files;
2.12 |  |  else
2.13 |  |  |  User submits post;
2.14 |  |  |  System saves and displays post;
```

Algorithm for Use Case UC3: Display Feed

Algorithm 3: The displayFeed() function

Input: User preferences, saved posts, interactions
Output: Feed displayed with relevant posts

```
3.1  User accesses home feed;
3.2  System loads posts based on user preferences and frequently visited communities;
3.3  if no relevant recent posts found then
3.4  |  Display trending and frequently visited recent posts;
3.5  else
3.6  |  User interacts with posts (like/comment/view);
3.7  |  if user reports a post as uninteresting then
3.8  |  |  System updates recommendations to exclude similar posts;
3.9  |  else
3.10 |  |  User continues browsing;
3.11 if user refreshes page then
3.12 |  Reload feed with new content if available;
3.13 else
3.14 |  if user scrolls to bottom then
3.15 |  |  Load additional posts;
```

Algorithm for Use Case UC4: Change User Settings

Algorithm 4: The editSettings() function

Input: Settings data: specific user preferences
Output: Settings updated successfully

4.1 User navigates to settings menu;
4.2 User selects category of setting to modify;
4.3 User inputs new values for desired changes;
4.4 **if** *input is invalid* **then**
4.5 Display error and prompt user to correct it;
4.6 **else**
4.7 User submits changes;
4.8 System updates settings and confirms success;

Algorithm for Use Case UC5: User sends a message request to another user

Algorithm 5: The sendChatRequest(targetUser) function

Input: targetUser - the ID of the user that the current user wants to message
Output: Confirmation message or error message

5.1 Verify if the current user is logged in;
5.2 **if** *user is NULL* **then**
5.3 **return** "You must be logged in to send a chat request.";
5.4 Retrieve the messaging preferences of targetUser from the database;
5.5 **if** *messaging is disabled for targetUser* **then**
5.6 **return** "Messaging is not available for this user.";
5.7 Prompt the current user to write an optional introductory message;
5.8 **if** *current user cancels the chat request* **then**
5.9 **return** "Chat request discarded.";
5.10 Record the chat request in the database with the currentUserID, targetUser, and timestamp;
5.11 Send a notification to targetUser about the chat request;
5.12 Notify the current user: "Success: Your chat request has been sent.";
5.13 **if** *targetUser declines the chat request* **then**
5.14 Update the database to mark the request as declined;
5.15 Notify the current user: "Chat request was declined, no further messages can be sent.";
5.16 **return** "Chat request process completed.";

Algorithm for Use Case UC:6 Users message each other (all messages after the first message)

Algorithm 6: The sendMessage(message, targetUser) function

Input: message - text being sent, targetUser - the id of the user being sent
Output: Confirmation of successful message delivery or an error message

```
6.1  Verify if the current user is logged in;
6.2  if user is NULL then
6.3      return "You must be logged in to send a message.";
6.4  Check if the targetUser is connected for direct messaging;
6.5  if targetUser is not connected then
6.6      return "You cannot message this user, please request to message them.";
6.7  Store the message in the database with senderID, targetUser, message content, and
       timestamp;
6.8  Attempt to deliver the message to targetUser in real-time;
6.9  if delivery fails due to network issues then
6.10     Display "Message failed to send. Will retry later";
6.11     Retry sending the message when the connection is restored;
6.12 if real-time notifications are enabled for targetUser then
6.13     Notify targetUser of the new message with a notification or sound;
6.14 else
6.15     Store the message in the database for future delivery;
6.16     return "Message will be delivered when the user is active.";
6.17 if read receipts are enabled for targetUser then
6.18     Update message status to "Delivered" once the message is received;
6.19     Update message status to "Read" once the recipient reads the message;
6.20 else
6.21     Only update the status to "Delivered.";
6.22 return "Your message has been sent.";
```

Algorithm for Use Case UC7: User shares a post (via URL)

Algorithm 7: The sharePost(post) function

```
    Input: post - the post object or ID to be shared
    Output: Unique URL for the post or an error message
7.1 Verify if the user is logged in;
7.2 if User is NULL then
7.3     return "You must be logged in to share a post.";
7.4 Retrieve the post's privacy settings from the database;
7.5 if post is private and the user does not have sharing permissions then
7.6     return "This post cannot be shared due to privacy settings.";
7.7 Generate a unique, shareable URL for the post;
7.8 Store the generated URL in the database with the postID and timestamp;
7.9 Display sharing options to the user;
7.10 if user selects Copy URL then
7.11     Copy the generated URL to the user's clipboard;
7.12     return "The URL has been copied to your clipboard.";
7.13 if user selects a social media sharing option then
7.14     Format the URL for the chosen platform;
7.15     Open the sharing interface for the platform;
7.16     return "The post has been shared.";
7.17 if recipient opens the URL then
7.18     Retrieve the post details from the database;
7.19     if post is public then
7.20         Display the post to the recipient;
7.21     else if post is private and the recipient is not logged in then
7.22         Prompt the recipient to log in or create an account to view the post;
7.23 return "Post shared.";
```

Algorithm for Use Case UC8: User deletes their account

Algorithm 8: The deleteAccount() function

Input: user - user wanting to delete account
Output: Confirmation of account deletion or an error message

8.1 Verify if the user is logged in;
8.2 **if** *user is NULL* **then**
8.3 ⌊ **return** "You must be logged in to delete your account.";
8.4 Display the Delete Account option in account settings;
8.5 Prompt the user with a confirmation dialog explaining that the action is irreversible and all associated data will be permanently deleted;
8.6 **if** *user cancels the deletion process* **then**
8.7 ⌊ **return** "Account not deleted.";
8.8 Optionally prompt the user to enter their password to confirm their identity;
8.9 **if** *password is incorrect* **then**
8.10 ⌊ **return** "Password is incorrect.Account deletion aborted.";
8.11 User goes through with deleting account;
8.12 Log the user out of the platform;
8.13 Display a confirmation message to the user stating that their account has been successfully deleted;
8.14 **return** "Your account has been deleted.";

Algorithm for Use Case UC9: User searches for a post or user profile

Algorithm 9: The **search(query) function** for UC9

Input: The input string "query" entered by the user in the search field.
Output: The list of all results (if they exist) matching that query.

```
9.1  User types a query into the search field;
9.2  if query is empty then
9.3      Display: "Please enter a valid input";
9.4  Search the database for questions, answers, posts, or users matching the query;
9.5  if no results found then
9.6      Display: "No results found.";
9.7      Prompt user if they would like to modify their search using suggested keywords;
9.8      if user selects yes then
9.9          Search the database again with the new term;
9.10     else
9.11         Prompt user to enter a new query;
9.12 else if broad range of search results is displayed then
9.13     Display filtering options to narrow the result;
9.14     Prompt user to select a filter option and search the database again;
9.15 else
9.16     Display the list of search results;
9.17     if user selects a result then
9.18         Fetch the information associated with that page and redirect;
9.19         Display the page;
```

Algorithm for Use Case UC10: User receives notifications

Algorithm 10: sendNotification(user, notification) function for UC10

Input: A user with a "true" flag set for receiving notifications
Output: The notifications sent to the user

```
10.1 Fetch the specific user preferences and profile. if User profile is not set up then
10.2     Do not send a notification;
10.3 if Notification flag is false then
10.4     Do not send a notification;
10.5 if Notification flag is true then
10.6     foreach preference in user preferences do
10.7         if preference is notification through email then
10.8             Send notification through email.
10.9         if preference is notification through text then
10.10            Send notification through text.
10.11        if preference is notification through app then
10.12            Send notification through the app.
```

Algorithm for Use Case UC11: User edits their post

Algorithm 11: editPost(user, post, editedPostContent) function for the post to be edited

Input: A logged-in user, the post to be edited, and the updated post information entered by the user.

Output: The updated post replacing the old one and visible on the website.

```
11.1  User navigates to the post in question and selects the post, calling the editPost
      function;
11.2  if invalid post then
11.3      Display: "Post does not exist";
11.4      return;
11.5  if user is not author of the post then
11.6      Display: "Insufficient permissions";
11.7      return;
11.8  if user cancels edits then
11.9      Display: "Canceled edits";
11.10     return back to user's post page;
11.11 while user is editing post do
11.12     Fetch action of the user;
11.13     switch action do
11.14         case submit do
11.15             Update the post with the editedPostContent parameter;
11.16         case cancel do
11.17             Display: "Edited post canceled, no changes made";
11.18             return;
11.19         case review do
11.20             Display the draft of the edited post;
```

Algorithm for Use Case UC12: User applies for verification tag.

Algorithm 12: applyForVerificationTag(user, usersDocuments) function for UC12, applying for the verification tag

Input: A logged-in user with their verification documents with the accetable file format and extension

Output: A verification tag for the user if successful, or a notification of an unsuccessful verification.

```
12.1  User navigates to the page with the application for verification;
12.2  if required fields are missing then
12.3      Display "please fill all required fields.";
12.4  foreach field in inputFields do
12.5      if field is invalid then
12.6          Display "Please enter a valid field for " + field + "".";
12.7  User submits information.;
12.8  User selects "upload documents";
12.9  foreach document in Documents do
12.10     if document is invalid or invalid file type then
12.11         Display "document is invalid or file type is invalid.";
12.12     Process uploaded documents;
12.13     if verification is successful then
12.14         updateUserProfile(user);
12.15     else
12.16         sendFailureNotification(notification,user);
12.17         Display "Error in verification, please try again later.";
```

Algorithm for Use Case UC13: Poster marks question as resolved

Algorithm 13: The modifyPost(content, post) function

Input: content -the content of the post, post - the post object that the user wants to resolve
Output: Confirmation message or error message

```
13.1  Verify if the user is logged in;
13.2  if user is NULL then
13.3  |  return "User must be logged in to perform this action.";
13.4  end
13.5  Retrieve the question details from the database using the post object;
13.6  if post does not exist then
13.7  |  return "Post not found.";
13.8  end
13.9  if post.isRevoled == True then
13.10 |  return "The question is already marked as resolved. Please select another
      |   question.";
13.11 end
13.12 Verify that the current user is the poster of the question;
13.13 if post.creator != user then
13.14 |  return "Only the original poster can mark a question as resolved.";
13.15 end
13.16 Update the question status to "resolved";
13.17 Disable further responses for the question;
13.18 return "The question has been marked as resolved!.";
```

Algorithm for Use Case UC14: User blocks another user

Algorithm 14: The blockUser(userId, targetUserID) function

Input: userId - the id the of the current user, targetUserID - the id of the user the current user wishes to block
Output: Confirmation message or error message

```
14.1  Verify if the current user is logged in;
14.2  if user is NULL then
14.3  |  return "User must be logged in to perform this action.";
14.4  Retrieve blocking relationship from the database;
14.5  if userId has already blocked targetUserID then
14.6  |  return "This user is already blocked.";
14.7  Prompt the current user with a confirmation dialog to block the target user;
14.8  if user confirms blocking then
14.9  |  Update the database to add a blocking relationship between userId and
      |   targetUserID;
14.10 |  Apply real-time updates to restrict interactions between the two users;
14.11 |  return "The user has been blocked.";
14.12 else
14.13 |  return "Action canceled. The user was not blocked.";
```

Algorithm for Use Case UC15: User Upvotes/Downvotes Posts or Responses

Algorithm 15: The upvote(post) or downvote(post) function

```
    Input: post- the post object which the user intends to vote on
    Output: Confirmation message or error message
15.1  Verify if the user is logged in;
15.2  if user is NULL then
15.3      return "User must be logged in to perform this action.";
15.4  Retrieve the user's voting history for the post from the database;
15.5  if user has already cast the same vote on post then
15.6      return "You cannot give the same vote twice to the same post.";
15.7  Retrieve the current vote count for the post from the database;
15.8  if unable to fetch vote count from database then
15.9      return "Unable to retrieve vote data.";
15.10 if upvote() then
15.11     Increment the post's vote count by 1;
15.12 else if downvote() then
15.13     Decrement the post's vote count by 1;
15.14 Update the database with the new vote count for the post;
15.15 if unable to update database then
15.16     return "Unable to store the vote.";
15.17 Update the user's voting history in the database to record this action;
15.18 if unable to update user's voting history then
15.19     return "Unable to record your vote.";
15.20 Reflect the new vote total on the post in real-time;
15.21 if system has trouble communicating with the server then
15.22     Retry connection up to 3 times;
15.23     if retries fail then
15.24         Alert the system administrator of the issue;
15.25         return "Unable to sync the vote with the server.";
15.26 return "Your vote has been recorded!";
```

Algorithm for Use Case UC16: User Reports Inappropriate Content or Behavior

Algorithm 16: The reportContent(post) function

Input: post - The post object in which the user is trying to report
Output: Confirmation message or error message

16.1 Verify if the user is logged in;
16.2 **if** *user is NULL* **then**
16.3 ⌊ **return** "User must be logged in to report content.";
16.4 Retrieve the content from the database using post;
16.5 **if** *post does not exist* **then**
16.6 ⌊ **return** "Can not preform action.The content no longer exists.";
16.7 **if** *user has blocked the post's creator OR the post's creator has blocked user* **then**
16.8 ⌊ **return** "You cannot interact with content from a blocked user.";
16.9 **if** *content has already been reviewed and deemed acceptable* **then**
16.10 ⌊ **return** "This content has already been reviewed and deemed acceptable.";
16.11 Prompt the user for a description of why they are reporting the post;
16.12 Record the report in the database;
16.13 Send the report to the system administrators for manual review;
16.14 Prompt user with confirmation message: "Your report has been submitted and will be reviewed.";
16.15 Notify the offending user with a message: "Your post has been flagged as offensive by another user and is under review.";
16.16 **return** "Report process completed.";

Algorithm for Use Case UC17: User Follows a Topic or Category

Algorithm 17: The followTopic(topicID) function

Input: User ID, Topic or Category Id to follow
Output: User is subscribed to the topic or category

17.1 User navigates to the topic/category section;
17.2 System displays available topics/categories;
17.3 User selects a topic or category to follow;
17.4 **if** *user is not already following the selected topic* **then**
17.5 ⌊ System subscribes user to the topic;
17.6 ⌊ System confirms subscription to the user;
17.7 ⌊ User starts receiving updates;
17.8 **else**
17.9 ⌊ Display error message: "Already subscribed to this this topic";
17.10 **if** *technical issue occurs* **then**
17.11 ⌊ Prompt user to try again later;

Algorithm for Use Case UC18: User Saves a Post for Later

Algorithm 18: The savePost(postID) function

Input: User ID, Post Id to save
Output: Post is saved to the user's profile

18.1 User navigates to the desired post;
18.2 User clicks the "Save" button;
18.3 **if** *post is not already saved* **then**
18.4 | System saves the post to the user's profile;
18.5 | System confirms: "Post saved successfully.";
18.6 **else**
18.7 | Display message: "Post is already saved.";
18.8 **if** *technical issue occurs* **then**
18.9 | Prompt user to try again later;

Algorithm for Use Case UC19: Admin/Moderator Reviews Reported Content

Algorithm 19: The reviewReportedContent(contentId) function

Input: Admin ID, Reported Content ID
Output: Content status updated based on the decision

19.1 Admin logs into the platform;
19.2 System displays the reported content queue;
19.3 Admin selects a reported content item;
19.4 System displays details of the report;
19.5 **if** *Admin decides to dismiss the report* **then**
19.6 | Notify the reporting user of dismissal;
19.7 **else if** *Admin decides to issue a warning* **then**
19.8 | Send warning to the content creator;
19.9 | Notify content creator about the infraction;
19.10 **else if** *Admin decides to remove the content* **then**
19.11 | System removes the content from the platform;
19.12 **if** *technical issue occurs* **then**
19.13 | Prompt admin to try again later;

Algorithm for Use Case UC20: User Creates a Group

Algorithm 20: createGroup(groupName,groupDescription) Function

Input:

- *groupName*: The name of the group to be created (string).
- *groupDescription*: A description of the group (optional string).

Output: Group created successfully or error message.

20.1 **Step 1: Input**

20.2 Prompt the user to input the following:

- Group Name.
- Group Members List (can be empty initially).
- Group Description (optional).

Step 2: Sanity Checks
if *groupName is empty* **then**
    Display error: *"Group name cannot be empty"* and return;
end
if *groupName already exists* **then**
    Display error: *"Group name already exists. Please choose another name"* and return;
end
Step 3: Group Creation
if *all checks pass* **then**
    Initialize a new group object with the given *groupName*, *groupMembers*, and *groupDescription*;
    Assign a unique group ID to the group;
    Save the group to the database;
    Display success message: *"Group created successfully"*;
end
else
    Display error: *"Failed to create group due to invalid input"*;
end

Algorithm for Use Case UC21: User Joins a Group

Algorithm 21: joinGroup() Function

Input:

- *userID*: The ID of the user attempting to join a group.
- *groupID*: The ID of the group the user wants to join.
- *loginCredentials*: The user's login credentials (username, password).

Output: Confirmation of group membership or error message.

Step 1: User Authentication
Verify *loginCredentials*: **if** *loginCredentials are invalid* **then**
    Display error: *"Invalid login credentials. Please try again or reset your password"*;
    Terminate the process;
end
Step 2: Group Validation
Retrieve group information from the database using *groupID*;
if *groupID does not exist* **then**
    Display error: *"Group does not exist"*;
    Terminate the process;
end
if *group is inactive or full* **then**
    Display error: *"Group is inactive or has reached its member limit. Explore similar groups"*;
    Terminate the process;
end
Step 3: Membership Type Check
Retrieve group's membership requirements: **if** *group is Closed or Invite-Only* **then**
    Prompt the user to submit a join request or invitation code;
    if *join request is denied by admin* **then**
        Display error: *"Membership request denied. Consider exploring other groups"*;
        Terminate the process;
    end
    else if *join request is pending* **then**
        Display message: *"Your membership request is pending administrator approval"*;
        Terminate the process;
    end
end
Step 4: Join Group
if *all conditions are satisfied* **then**
    Add *userID* to the group's member list in the database;
    Notify the user: *"You have successfully joined the group"*;
    Grant the user access to the group's content (posts, discussions, events, etc.);
end
Step 5: Notification and Error Handling
if *membership confirmation is delayed due to technical issues* **then**
    Notify the user: *"Membership confirmation is delayed. Please wait for further updates"*;
end
else if *unexpected error occurs* **then**
    Display error: *"An unexpected error occurred. Please try again later"*;
end

Algorithm for UC22: User Participates in Polls or Surveys

Algorithm 22: participateInPoll()

Input:

- *userID*: The ID of the user participating in the poll.
- *pollID*: The ID of the poll the user wants to respond to.
- *response*: The user's response to the poll.

Output: Response is successfully recorded, or error message is displayed.

Step 1: User Authentication
Check if the user is logged in;
if *userID is not valid* **then**
    Display error: *"You must be logged in to participate in polls"*;
    Redirect user to login page;
    Terminate the process;
end
Step 2: Poll Validation
Check if the poll exists in the system;
if *the poll is not found* **then**
    Display error: *"Poll does not exist"*;
    Terminate the process;
end
Check if the poll is active and accepting responses;
if *the poll is closed* **then**
    Display error: *"This poll is no longer accepting responses"*;
    Display poll results;
    Terminate the process;
end
Check if the user has already responded to the poll;
if *the user has responded* **then**
    Display error: *"You have already participated in this poll"*;
    Terminate the process;
end
Step 3: Validate Response
if *response is invalid or incomplete* **then**
    Display error: *"Your response is incomplete or invalid. Please fix your input"*;
    Terminate the process;
end
Step 4: Record Response
Save the response in the poll's data;
Log the user's participation for future reference;
Step 5: Notification and Feedback
Display confirmation message: *"Your response has been successfully submitted"*;
if *the poll displays real-time results* **then**
    Show updated results to the user;
end

Algorithm for UC23: User Hosts a Poll or Survey

Algorithm 23: hostPoll()

Input:

- *userID*: The ID of the user creating the poll.
- *groupID*: The ID of the group where the poll will be posted.
- *pollDetails*: The details of the poll (e.g., title, question, response type, privacy settings).

Output: Poll is successfully created and published, or error message is displayed.

Step 1: User Authentication and Group Validation
Check if the user is logged in;
if *userID is not valid* **then**
    Display error: *"You must be logged in to host a poll"*;
    Redirect user to login page;
    Terminate the process;
end
Check if the user is a member of the specified group;
if *the user is not a member* **then**
    Display error: *"You must be a member of the group to post a poll"*;
    Terminate the process;
end
Step 2: Permission Check
Check if the user has the required permissions to host a poll in the group;
if *the user lacks permissions* **then**
    Display error: *"You do not have permission to host polls in this group"*;
    Terminate the process;
end
Step 3: Validate Poll Input
if *any field in pollDetails (title, question, response type) is empty or invalid* **then**
    Display error: *"Poll details are incomplete or invalid"*;
    Prompt the user to fix the errors;
    Terminate the process;
end
Check if the poll title is unique in the group;
if *the title is already in use* **then**
    Display error: *"A poll with this title already exists in the group"*;
    Terminate the process;
end
Step 4: Create Poll
Save the poll details in the system;
Associate the poll with the specified group;
Log the poll creation under the user's account;
Step 5: Confirmation and Feedback
Display confirmation message: *"Poll has been successfully created and published"*;
Redirect the user to the poll management page for monitoring responses and analytics;

Algorithm for Use Case UC24: User Logs Out

Algorithm 24: The logout() function

Input: User is logged into their account, active session exists
Output: User session terminated, confirmation of successful logout

User navigates to account settings;
User selects the "Log Out" option;
if *system requires logout confirmation (mobile device)* **then**
    Prompt user to confirm logout;
    if *user cancels the action* **then**
        Return to settings menu;
    else
        Proceed to terminate session;
else
    Terminate user session directly;
if *session terminated successfully* **then**
    Redirect user to login page or homepage;
    Display message confirming successful logout;
else
    Display error message and prompt user to retry;
Extensions:
if *user closes the browser/app without logging out* **then**
    Automatically terminate session after timeout or inactivity;
if *system error occurs during logout* **then**
    Inform user of unsuccessful logout and suggest retrying;