Criteria B: Design

System diagram

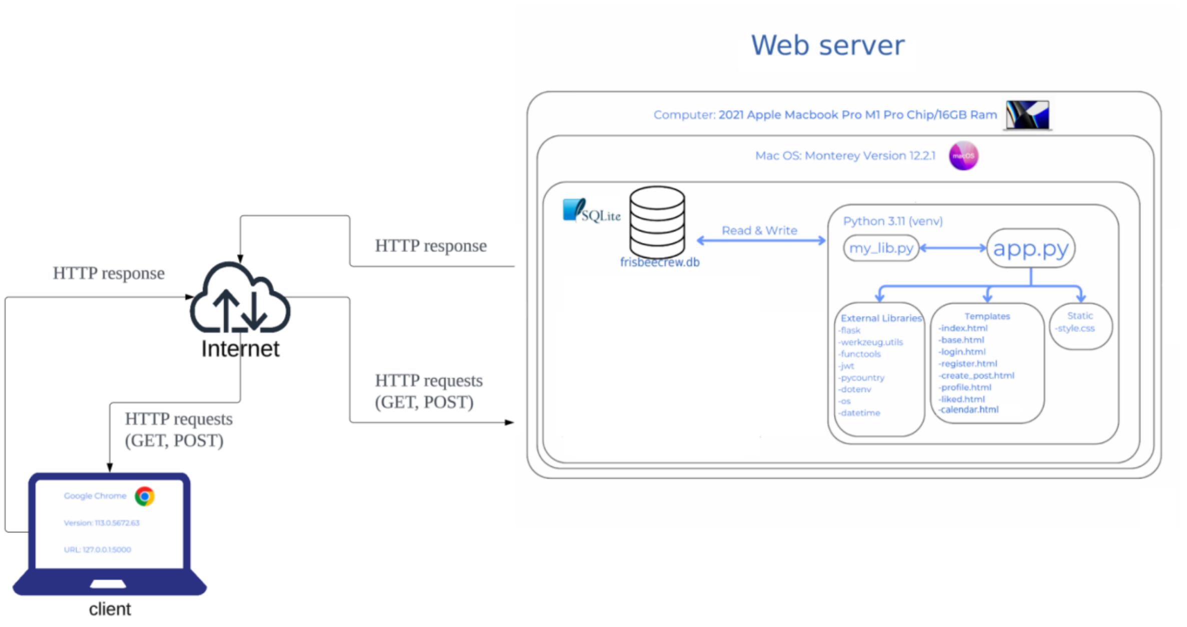


Fig. 1 System diagram of FrisbeeCrew web application

Fig. 1 serves as a visual representation of the system and its components, and their relationships to each other. As shown above, the web application will be programmed in python, and all information will be stored in the SQLite database called “frisbeecrew.db”. The application will have various inputs from the user, which will then output onto their screen through HTTP methods.

ER diagram

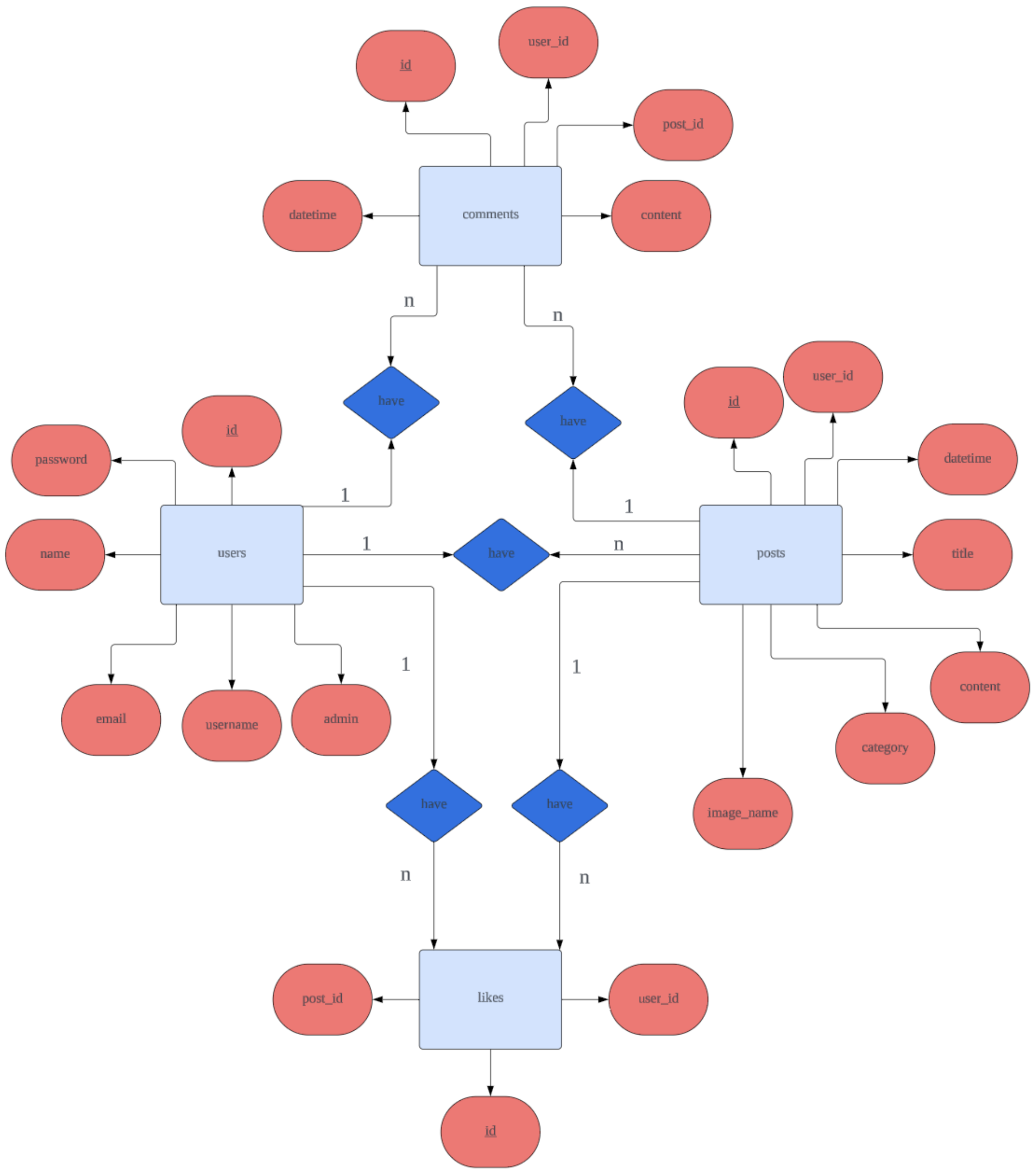
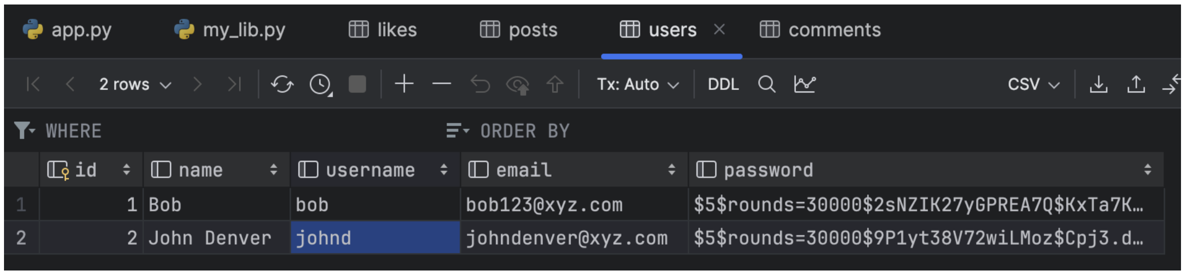
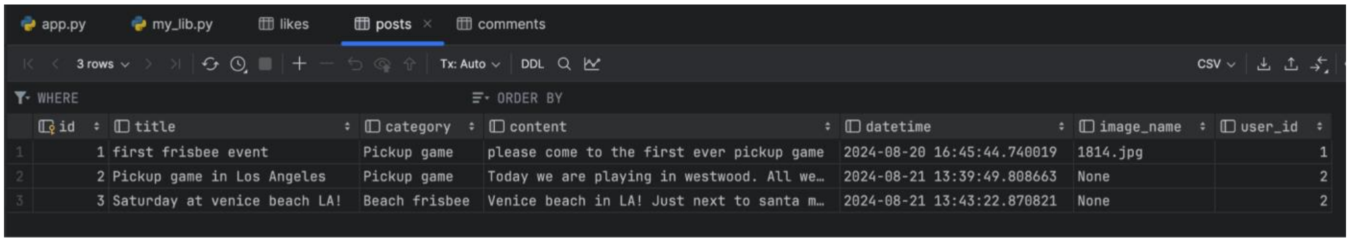


Fig. 2 Entity-Relationship (ER) diagram for FrisbeeCrew's database. Relationships between users, posts, likes, and comments in the database are illustrated, with one-to-many relationships being represented by "1" and "n." A user can like and post multiple posts.



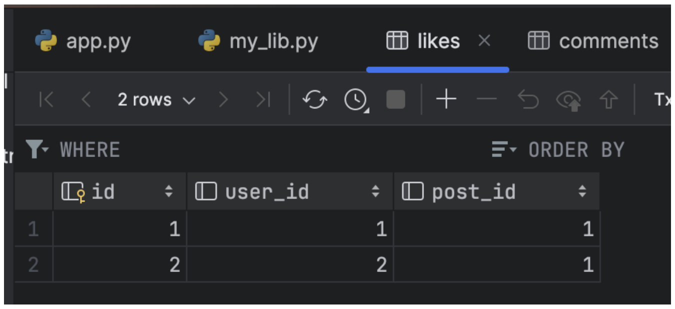
	id	name	username	email	password
1	1	Bob	bob	bob123@xyz.com	5rounds=30000$2sNZIK27yGPREA7Q$KxTa7K…
2	2	John Denver	johnd	johndenver@xyz.com	5rounds=30000$9P1yt38V72wiLMoz$Cpj3.d…

Fig. 3 Screenshot of the table users from **Fig. 2**. Note that passwords are hashed using sha-256



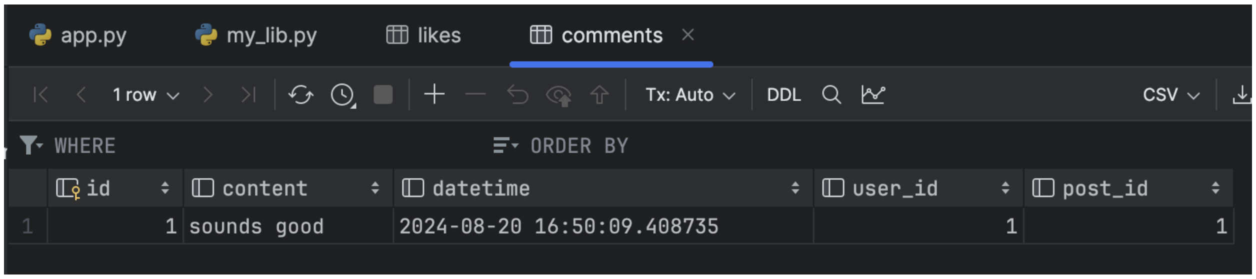
	id	title	category	content	datetime	image_name	user_id
1	1	first frisbee event	Pickup game	please come to the first ever pickup game	2024-08-20 16:45:44.740019	1814.jpg	1
2	2	Pickup game in Los Angeles	Pickup game	Today we are playing in westwood. All we…	2024-08-21 13:39:49.808663	None	2
3	3	Saturday at venice beach LA!	Beach frisbee	Venice beach in LA! Just next to santa m…	2024-08-21 13:43:22.870821	None	2

Fig. 4 Screenshot of the table posts from **Fig. 2.** Note that image_name contains the name of the file rather than the image information.



	id	user_id	post_id
1	1	1	1
2	2	2	1

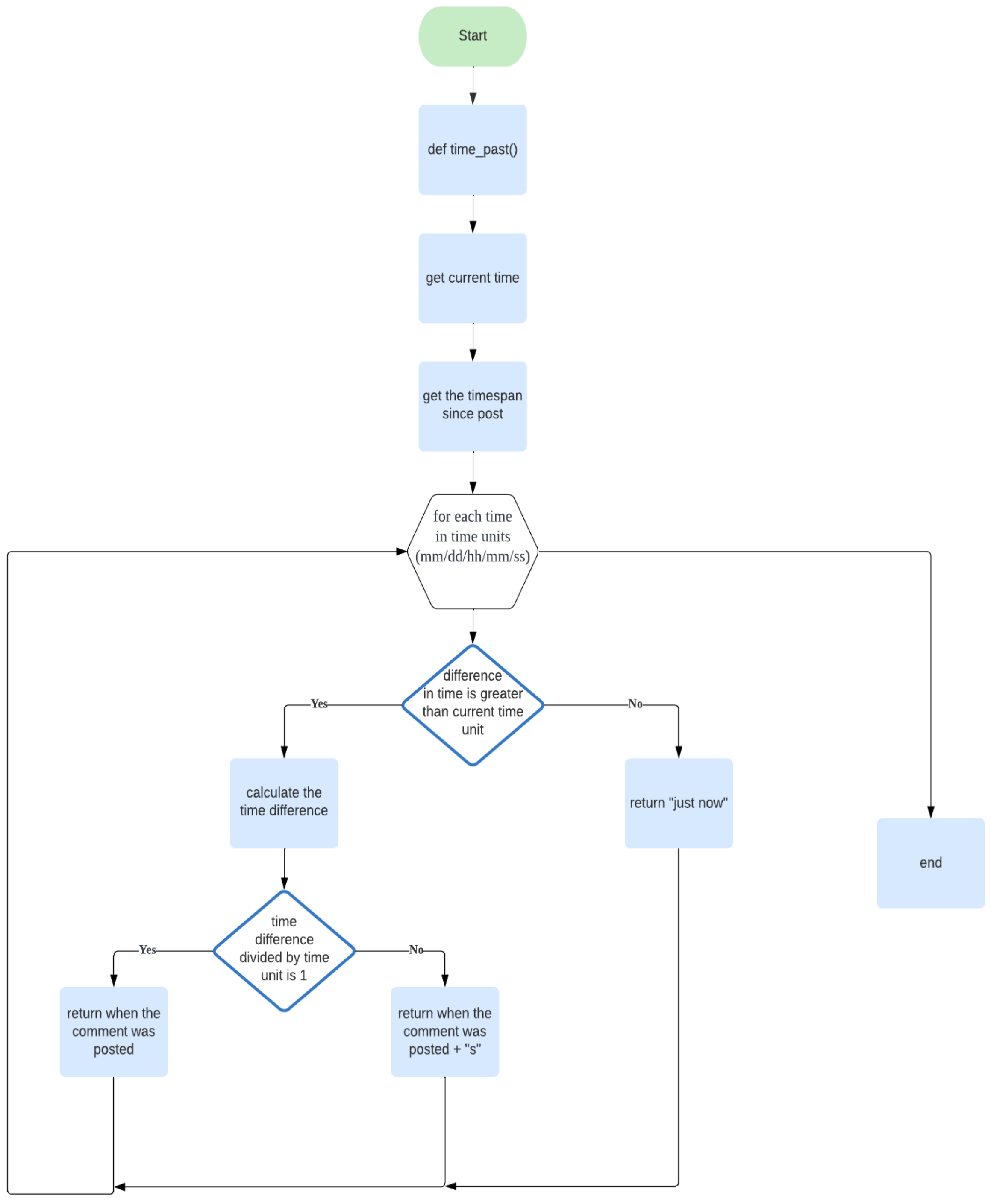
Fig. 5 Screenshot of the table likes from **Fig. 2**. As shown in the screenshot, users with id 1 and id 2 both like the post with id 1.



	id	content	datetime	user_id	post_id
1	1	sounds good	2024-08-20 16:50:09.408735	1	1

Fig. 6 Screenshot of the table comments from **Fig. 2.** Datetime is recorded real time.

Flow diagrams



Start
def time_past()
get current time
get the timespan since post
for each time in time units (mm/dd/hh/mm/ss)
difference in time is greater than current time unit
Yes
No
calculate the time difference
return "just now"
end
time difference divided by time unit is 1
Yes
No
return when the comment was posted
return when the comment was posted + "s"

Fig. 7 This flow diagram explains the process of how the web application determines how long ago has the user posted

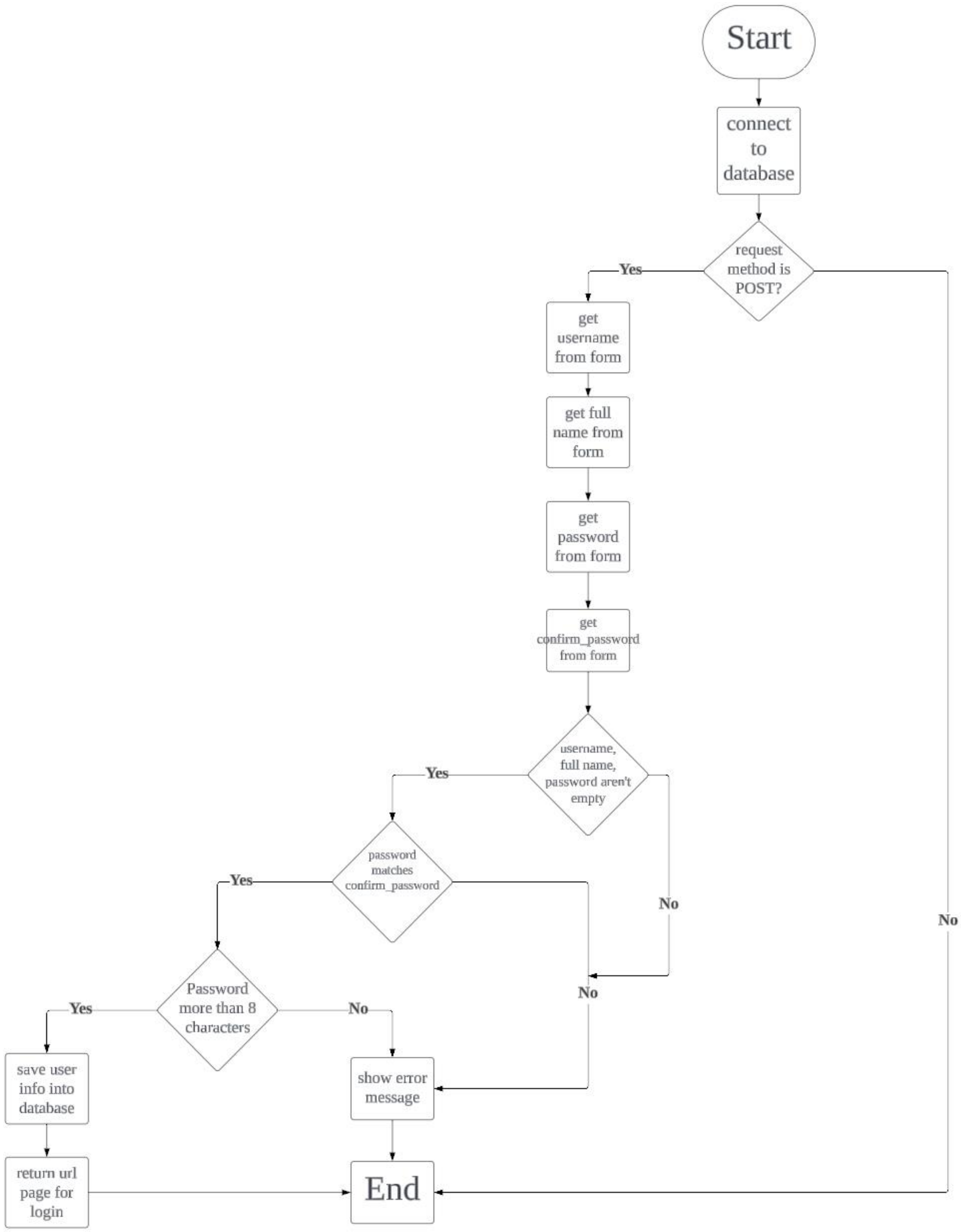

Start
connect to database
request method is POST?
Yes
No
get username from form
get full name from form
get password from form
get confirm_password from form
username, full name, password aren't empty
Yes
No
password matches confirm_password
Yes
No
Password more than 8 characters
Yes
No
save user info into database
return url page for login
show error message
End

Fig. 8 This flow diagram illustrates the functionality of user registration

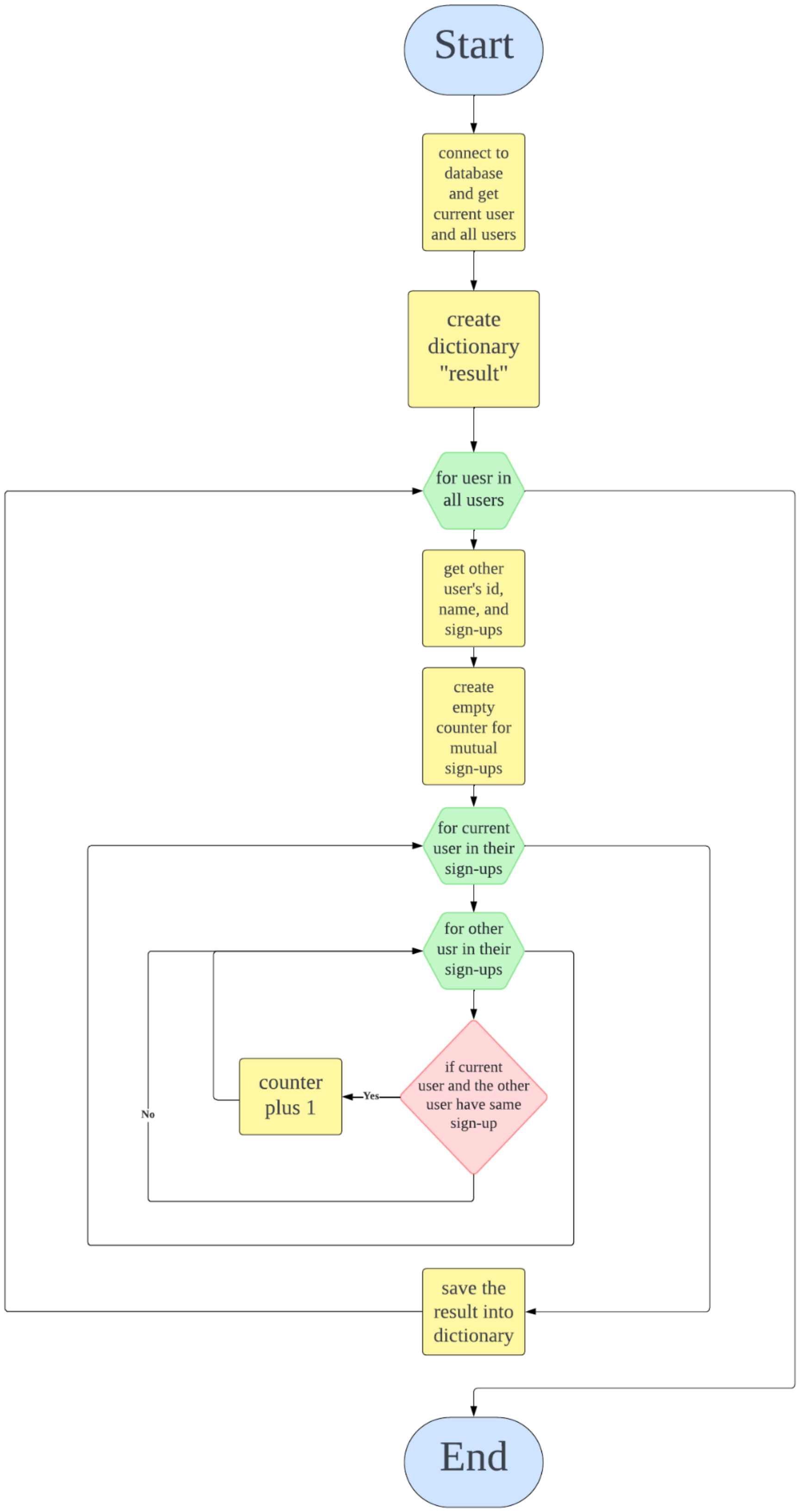
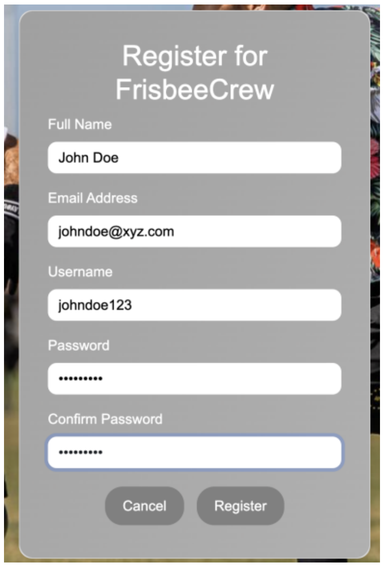

Start
connect to database and get current user and all users
create dictionary "result"
for uesr in all users
get other user's id, name, and sign-ups
create empty counter for mutual sign-ups
for current user in their sign-ups
for other usr in their sign-ups
if current user and the other user have same sign-up
Yes
counter plus 1
No
save the result into dictionary
End

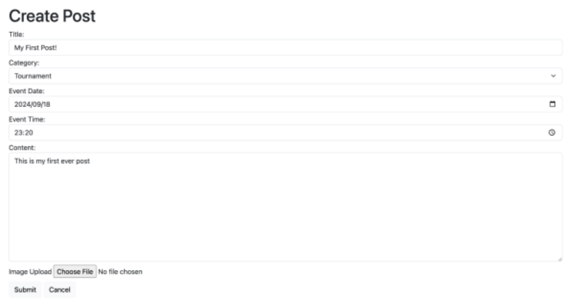
Fig. 9 This flow diagram explains how the application finds mutual sign-ups described by SC#7.

Test plan

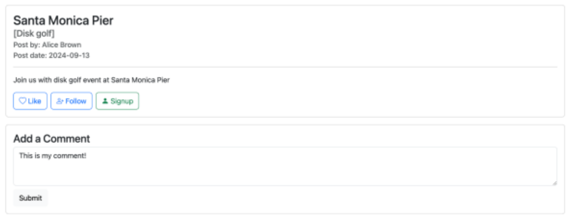
Description	Input	Expected Output
SC#1: Registration System	1.Open Website by navigating to url: http://127.0.0.1:5000 2.Click on the register button 3. Enter `"John Doe"` for the full name. Enter `"johndoe@xyz.com`" for the email address. Enter `"johndoe123"` for the username. Enter `"johnd1234"` for the password and confirm the password. 4. Click the register button on the page.  Register for FrisbeeCrew Full Name: John Doe Email Address: johndoe@xyz.com Username: johndoe123 Password: ••••••••• Confirm Password: ••••••••• 	After clicking the register button, it should redirect to the login page with a message on the screen saying, "Registration completed. Please log in."

SC#1: Test for Error Scenario for Registration System	1.Open Website by navigating to url: http://127.0.0.1:5000 2.Click on the register button 3. Enter `"John Doe"` for the full name. Enter `"johndoe@gmail.com"` for the email address. Enter nothing for the username. Enter `"john123"` for the password and confirm the password. 4. Click the register button on the page.	After clicking the register button, the page should refresh, with a red error message appearing at the top of the screen detailing the errors saying: "Incorrect characters for the users Full Name" and "Password too short, min. 8 characters". This outcome demonstrates that the application effectively validates user input during registration, ensuring data integrity.
SC#1: Test for Users Database	1. Open Website by navigating to url: http://127.0.0.1:5000 2. Click on the register button 3. Enter `"Jane Smith"` for the full name. Enter `"janesmith@xyz.com"` for the email address. Enter `"janesmith123"` for the username. Enter `"janesmithlovesskating"` for the password and confirm password. 4. Click the register button on the page.	After successfully registering, the new user's information should be stored into the "frisbeecrew.db" database, specifically in the "users" table. Each time a user registers, their information should be added to the "users" table, ensuring the system effectively manages user data and integrates the registration process with the database operations.

SC#1: Test for Login System	1. Open Website by navigating to url: http://127.0.0.1:5000 2. Click on the login button 3. Enter a valid email and password for an existing user (e.g., `"johndoe@xyz.com"` and `"johndoelovestoski"`) 4. Click the login button on the page.	After clicking the submit button, the page should redirect to the homepage of the website
SC#1: Test for error scenario for Login System	1. Open Website by navigating to url: http://127.0.0.1:5000 2. Click on the login button 3. Enter an invalid email (e.g., `"johndoe@gmail.com"`) and/or an incorrect password (e.g., `"wrongpassword"`) 4. Click the login button on the page	After clicking the login button, the page should refresh, displaying a red error message at the top of the screen, indicating that the login attempt has failed due to incorrect email and/or password. This test ensures that the application effectively validates user input during login and prevents unauthorized access with incorrect credentials.

SC#2: Test for Creating Posts	1.Open Website by navigating to url: http://127.0.0.1:5000 2.Login 3.Click Create Post on the navigation bar 4.Enter in `"My First Post!"` as the title, select `"Tournament"` as the category, enter `"This is my first ever post."` as the content. 5.Select current time and date 6.Click the submit button  Create Post Title: My First Post! Category: Tournament Event Date: 2024/09/18 Event Time: 23:20 Content: This is my first ever post Image Upload Choose File No file chosen Submit Cancel	After clicking the submit button, the page should redirect to the homepage of the website, displaying the new post with its Title, Category, Content, Author, and Post Date. Fulfilling the clients need of being able to create posts with categories
SC#2: Test for Error Scenario for Creating Posts	1.Open Website by navigating to url: http://127.0.0.1:5000 2.Login 3.Click Create Post on the navigation bar 4.Enter in only the title `"My Second Post!"` and leave the category and content fields empty 5.Select current time and date 6.Click the submit button	After clicking the submit button, the user should see a specific error message indicating that the required fields (category and content) are not filled in. The post should not be created or displayed on the homepage. This test ensures that the application properly handles invalid input, enforces mandatory field requirements, and prevents incomplete posts from being created.

SC#2: Test for Posts Database	1. Open Website by navigating to url: http://127.0.0.1:5000 2. Login 3. Click Create Post on the navigation bar 4. Enter `"My Post!"` as the title, select "Analysis" as the category, enter `"This is my post."` as the content. 5. Select current time and date 6. Click the submit button	After clicking the submit button, the new post's information should be stored into the "frisbeecrew.db" database, specifically in the "posts" table. Each time a user creates a post, all post information should be added to the "posts" table, ensuring the system effectively manages post data and integrates the post creation process with the database operations.
SC#2: Test for Editing Posts	1.Open Website by navigating to url: http://127.0.0.1:5000 2.Login 3.Navigate to a post created by the logged-in user 4. Click the 'Edit' button below the post 5. Modify the post's title to `'My Edited Post!'`, change the category to `“Beach frisbee”`, and update the content to 'This post has been edited.' 6. Click the 'Save Changes' button	After clicking the 'Save Changes' button, the page should refresh and display the updated post with its new Title, Category, and Content. This confirms that the application allows users to edit their own posts successfully, providing a way to correct or update information as needed.

SC#2: Test for Edited Posts Database	1. Open Website by navigating to url: http://127.0.0.1:5000 2. Login 3. Navigate to a post created by the logged-in user 4. Click the 'Edit' button below the post 5. Modify the post's title to `“My Edited Post Again!”`, change the category to “Disk golf”, and update the content to `“This post has been edited again.”` 6. Click the “Save Changes” button	After clicking the 'Save Changes' button and refreshing the page, the updated post should be reflected in the "posts" table of the "frisbeecrew.db" database, confirming that the application correctly stores edited post data in the database. This ensures that the changes made by users are successfully saved and persist across sessions.
SC#2: Test for Creating Comments	1. Open Website by navigating to url: http://127.0.0.1:5000 2. Login 3. Navigate to a post 4. Enter `"This is my comment!"` in the comment text box below the post 5. Click the 'Submit' button  Santa Monica Pier [Disk golf] Post by: Alice Brown Post date: 2024-09-13 Join us with disk golf event at Santa Monica Pier Add a Comment This is my comment! 	After clicking the 'Submit' button, the page should refresh, and the new comment should be displayed below the post, including the commenter's name and the comment timestamp. This fulfills the client's need for users to be able to add comments to posts.

SC#2: Test for Error Scenario for Creating Comments	1. Open Website by navigating to url: http://127.0.0.1:5000 2. Login 3. Navigate to a post 4. Leave the comment text box empty 5. Click the “Submit” button	After clicking the 'Submit' button, an error message should appear, stating that the comment cannot be empty. The comment should not be created or displayed below the post.
SC#2: Test for Commentin g Database	1. Open Website by navigating to url: http://127.0.0.1:5000 2. Login 3. Navigate to a post 4. Enter `"This is another comment!"` in the comment text box below the post 5. Click the “Submit” button	After clicking the 'Submit' button and refreshing the page, the new comment should be stored in the "comments" table of the "frisbeecrew.db" database, confirming that the application correctly stores comment data in the database. This ensures that user comments persist across sessions and are retrievable for future reference.

SC#2: Test for Liking Posts	1. Open Website by navigating to url: http://127.0.0.1:5000 2. Login 3. Navigate to a post 4. Click the 'Like' button below the post	After clicking the 'Like' button, the page should refresh, and the like button should now read "unlike", and the number of likes should increase by 1. This fulfills the client's need for users to be able to like posts, helps to identify popular content, and allows users to easily find and revisit posts they have liked in the past.
SC#2: Test for Error Scenario for Liking Posts	1. Open Website by navigating to url: http://127.0.0.1:5000 2. Login 3. Navigate to a post 4. Click the 'Like' button twice	After clicking the 'Like' button twice, the button should still show "like", as the post was liked then unliked, and the number of likes should still only increase by 1. This test ensures that the application properly handles multiple clicks on the 'Like' button and prevents users from liking a post multiple times.

SC#2: Test for Liking Posts Database	1. Open Website by navigating to url: http://127.0.0.1:5000 2. Login 3. Navigate to a post 4. Click the 'Like' button for the post	After clicking the 'Like' button, a record should be stored in the "likes" table of the "frisbeecrew.db" database, reflecting the user's like of the post. This ensures that user likes are correctly stored in the database.
SC#6: Test for calendar page	1.Open Website by navigating to url: http://127.0.0.1:5000 2.Login 3.Click Calendar on the navigation bar	A calendar should be shown with events on specific dates, showing also Category, Content, and Author.
SC#65 Test for editing sign-ups	1.Open Website by navigating to url: http://127.0.0.1:5000 2.Login with admin account 3.Click Edit sign-ups on the navigation bar 4.Select the event `"LA pickup"` 5.Delete participant `“Bob”` 6.Click the save button	After clicking the submit button, the page should redirect to the edit sign-ups page, displaying the event with the updated list of participants. This fulfills the success criteria of allowing admin users to edit sign-ups.

SC#4: Test for messaging other members	1.Open Website by navigating to url: http://127.0.0.1:5000 2.Login with admin account 3.Click Message on the navigation bar 4.Select “test account” to message from the contact list 5.Enter `“Hi”` in the chat box 6.Click send button	Logging into the other account and clicking Message on the navigation bar, the new message “Hi” should be shown.
SC#7: Test for signing up for events	1.Open Website by navigating to url: http://127.0.0.1:5000 2.Login 3.Click “Beach frisbee” on the navigation bar 4.Click “Sign up” button at the end of the post	After clicking the “Sign up” button, there should be a pop up message reminding that the user has successfully signed up for the event.
SC#7: Test for signing up for events in Database	1.Open Website by navigating to url: http://127.0.0.1:5000 2.Login 3.Click “Beach frisbee” on the navigation bar 4.Click “Sign up” button at the end of the post 5. Check “posts” table in the “frisbeecrew.db” database	After clicking the “Sign up” button, a record should be stored in the "posts" table of the "frisbeecrew.db" database, showing the signup record of the user. This ensures that the signup is correctly stored in the database.

SC#7: Test for checking mutual sign-ups	1.Open Website by navigating to url: http://127.0.0.1:5000 2.Login 3.Click “Beach frisbee” on the navigation bar 4.Click “Mutual sign-ups” button at the end of the post	After clicking the “Mutual sign-ups” button, a list of users that have mutual sign-ups with the number of mutual sign-ups should be displayed.
Code Review	Review the entire codebase, identifying and removing any unused code or debugging and testing artifacts, and adding comments to detail parts of code.	Revised version of the code that follows good coding practices, with improved readability and organization