Messaging app documentation

Goals and Requirements:

The goal of this project is to create a messaging service similar to Facebook Messenger. This messaging app is to be browser based. The functionality under pinning such a system being roughly split into three categories:

Accounts:

Users can create an account to access the site.

Friendships:

User discovery allowing users to request a relationship with each other.

Ability to accept or deny friendship requests.

Ability to unfriend or remove friendships.

Chatting:

Ability to send and view messages.

Quality of life and security design choices include:

An intuitive design following popular conventions.

Pagination will be used to nullify performance issues stemming from extremely long chat histories between users.

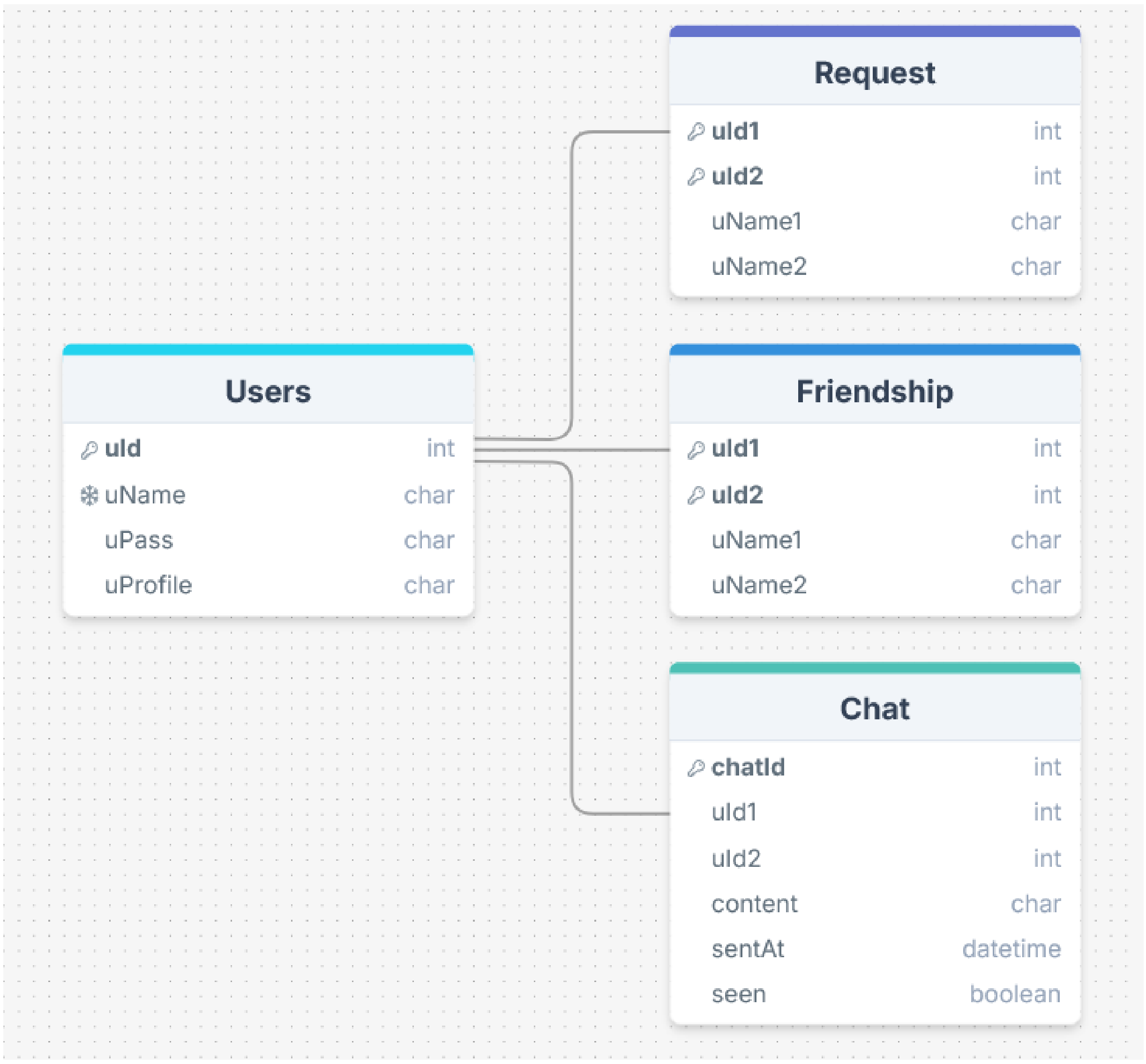
Accommodate PC and mobile (minimum screen height and width of 675 and 380 px) users.

Resilience to SQL injection attacks.

Information about the Stack:

The front end of the application will be created with React and styled using CSS with the backend created through PHP and an SQL database.

Here is a diagram representation of the SQL tables used



Users have a unique Name(case sensitive) and Id. They are also required to have a password and are given the option to choose a profile picture.

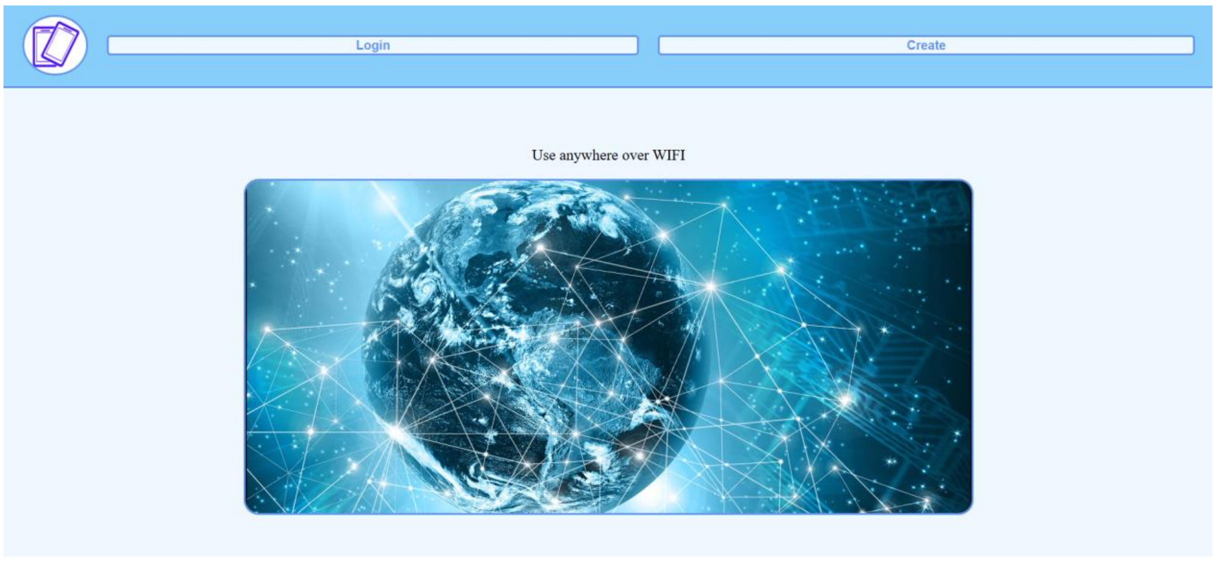
A Request stores the Name and Id of the requesting user in uId1 and uName1 with the recipient's details stored in uId2 and uName2 (The repeating of user data has the potential for parity issues but used to cut down on query complexity).

The Friendship table follows the same structure as the request table.

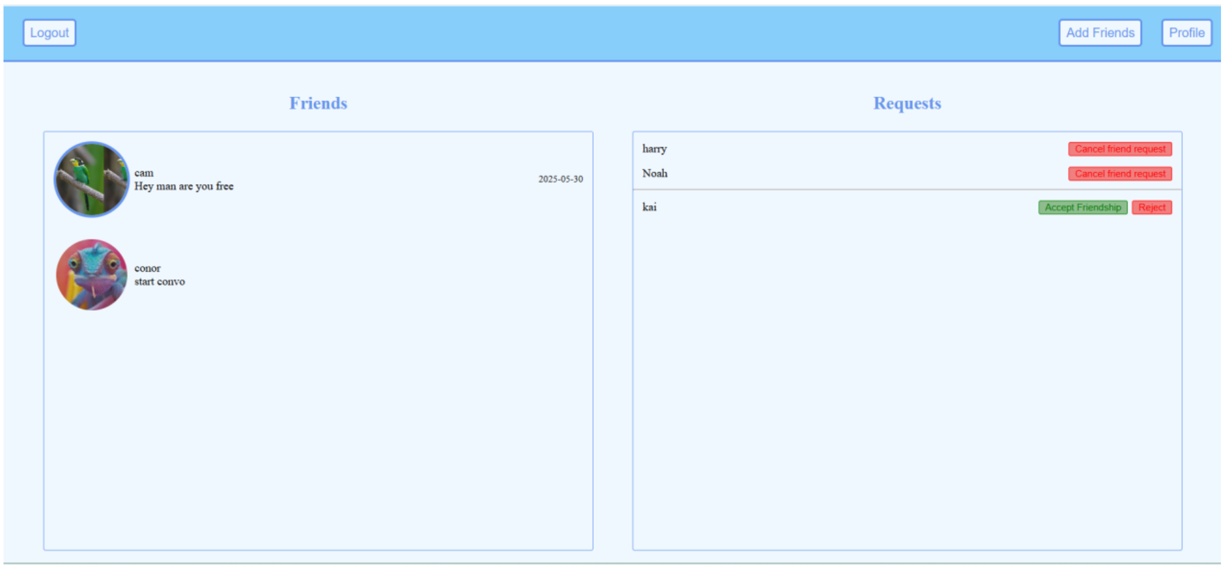
the Chat table is uniquely identified by chatId. They have the sender's Id stored in uId1 and receiver's id stored in uId2. Each entry also includes the content, when it was sent and if the receiving user has seen this message yet.

Demonstration:

This is presentation to the main features of the application with mobile and PC demonstration videos are available in the Repo.



1] The Landing displays a carousel of the features of the app with a header containing button that move the user to the Login or Create-Users Pages.

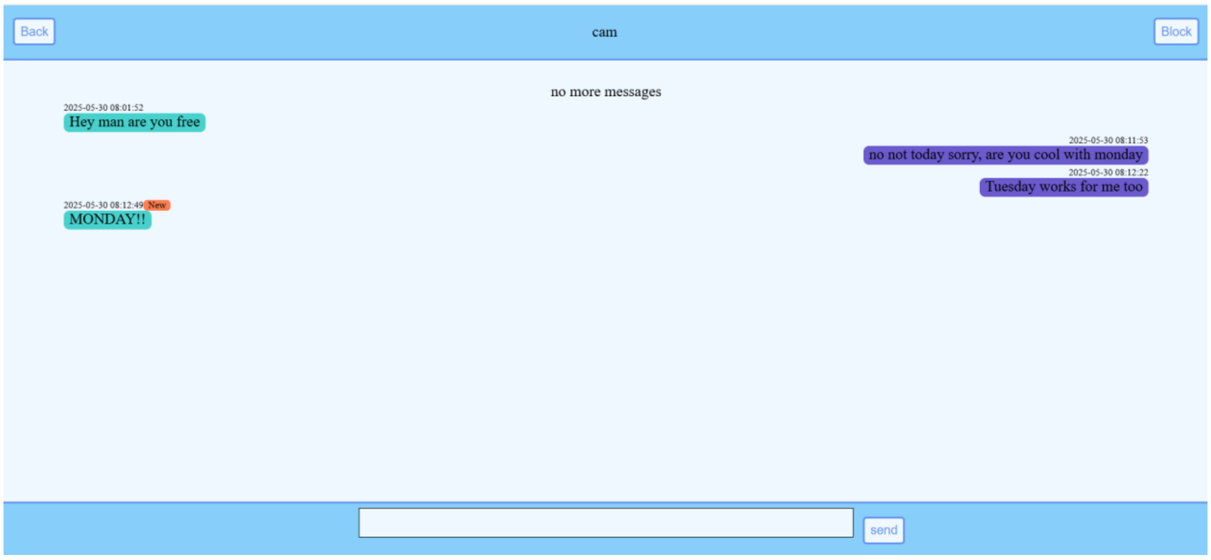


cam
Hey man are you free
2025-05-30

conor
start convo

harry
Noah
kai

2] Upon signing in, the user is directed to the User-Page. On the left side of the user the User's Friends are listed in order of when they were last chatted with. On the right side the User's requests to others user and other Users' requests to this user are listed allowing them to retract the invitation in the case that they initiated it or accept or deny in the case that They didn't.



no more messages

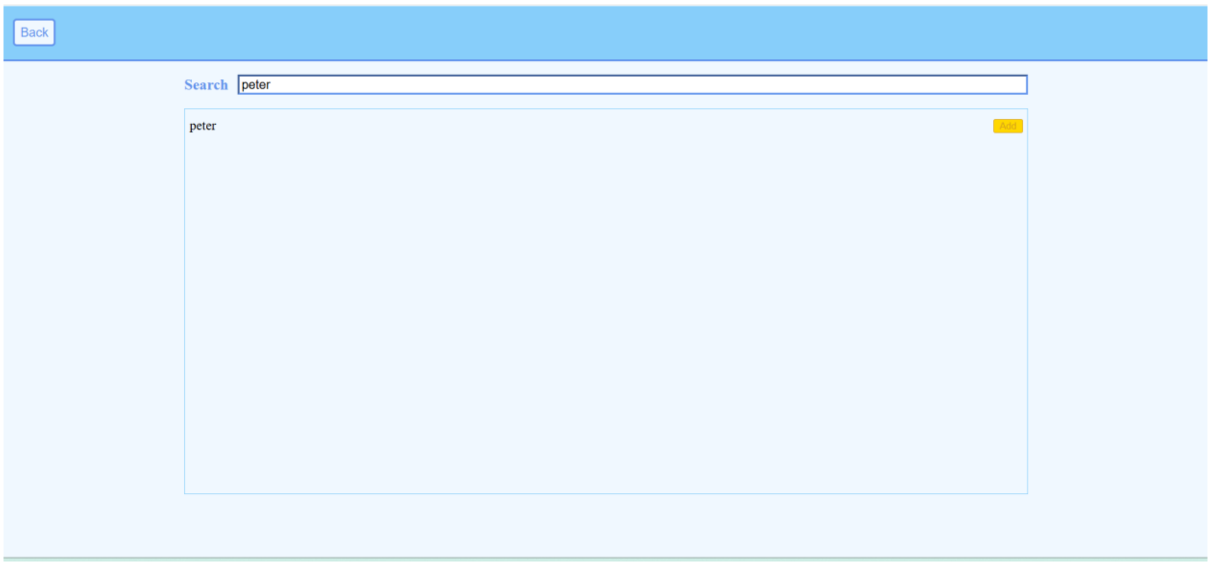
2025-05-30 08:01:52
Hey man are you free

2025-05-30 08:11:53
no not today sorry, are you cool with monday

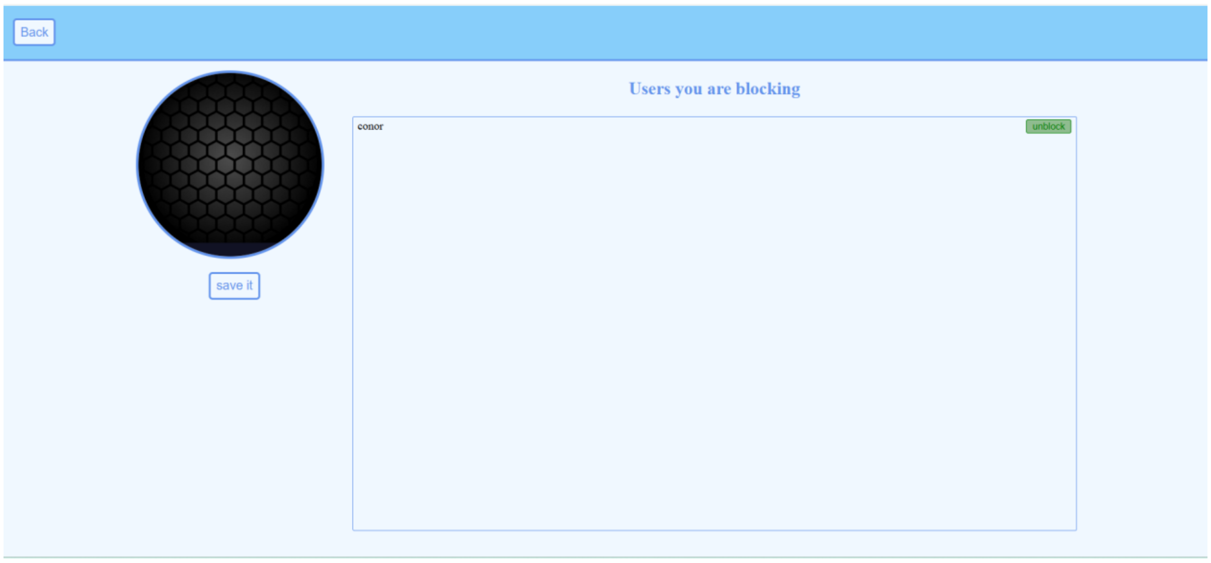
2025-05-30 08:12:22
Tuesday works for me too

2025-05-30 08:12:49 New
MONDAY!!

3] When an account from the friends list is clicked on the user is directed to the Chat-Page. Users can text their friend here. Text chats are paginated with the 40 most recent texts loading automatically and 40 subsequent texts loading when the users presses the load more texts button (this button has been replaced by the "no more messages" text because the users' full chat history is loaded). In the top right corner users can unfriend/ block this user.



4] The Search-Page accessible through a button on the top right of the User-Page allows user to send invites to other users. Users can be accessed by searching their name.



5] The Profile-Page is similarly accessible through a button on the top right of the User-Page. This Page allows the User to unblock their friends and change their profile picture.

Testing:

The application is designed to be robust security wise. In the implementation and testing the following are some of the areas that were considered potentially problematic and what methods are being used to secure them.

> Functions called by the backend have the potential to fail for a variety of reasons. Specifically for this application, image processing functions are called to handle the user's profile picture. When functions are called that can fail the function's output is checked and returned to the calling JavaScript where it is anticipated and handled.
>
> Incorrect user data in the form of incorrectly formatted names or pictures. This has been handled in most cases using both client and server-side validation.
>
> For all data access the application requires authentication to stop users requesting confidential data directly from the back end of the application.
>
> SQL injections are prevented using prepared statements.
>
> Users have the potential to retract an invite at the same time as the target accepts it. These scenarios are possible so the backend is constructed in a way that no assumptions are made and the presence of a request is always checked before operation.
>
> Friend names can vary from 3 to 30 characters in length, special care has been taken to make sure all names display correctly on the screen.

Improvements:

The project meets the requirements that were initially listed although a few improvements can be made.

Allow users to send images to each other.

A better interface for selecting a profile picture allowing a user to focus the image on a specific portion.

When user have passed the login stage their credentials are save in local storage so that they can be reverified every time they try to access their user specific data. This is not a secure approach and would require the implementation of some sort of key system to enhance security.

Some improvements can be made In the communication of sever errors to the user.

As mentioned above there is repetition in the database this could be improved upon at the expense of backend complexity and performance.

The site could be made disability friendly.

Thanks you for viewing my Project