**Creating a Social Media App Focused on Privacy**

By Elliot Holbrook, Queen Elizabeth’s Grammar School [017080]

**Contents**

Analysis

* Introduction
* Computational Amenability
* Stakeholders
  + Gamers
  + Students
  + Individuals That Are Privacy Conscious
* Research
  + Existing Implementations
    - Masterdon
    - MeWe
  + Conclusions
    - Positives
    - Negatives
* Requirements For the Proposed Solution
  + Development Requirements
  + End User Requirements
* Limitations Of The Proposed Solution
  + Security
  + Servers
  + Features
  + Bugs
* Success Criteria
  + Simplicity
  + Modern User Interface
  + Messaging
  + Privacy
  + Hubs and Interests

Design

* Methods, Variables, Classes and Parameters
* Algorithms
  + Flowcharts
    - Displaying History of a Chat to a User
    - Allowing a User to Send a Chat
    - Allowing a User to Log In
  + Validation Methods
    - General Inputs
    - Emails
    - Usernames
    - Profile Pictures
    - Passwords
    - Pins
  + Pseudocode
    - Friend Requests
    - Deleting a Chat
* Usability
  + Learnability
  + Efficiency
  + Errors and Validation
  + Satisfaction
* Testing
  + General Testing
  + Specific Testing
* Databases
  + Account Data
  + Messages
  + Hubs
* Interface Designs
  + First Ideas

Development

**Analysis**

**Introduction**

Slowly but surely, data privacy is more and more becoming a luxury rather than a given. The quantity of user data that is secretly collected and sold by large companies, such as Facebook or Google, is astounding and the problem is only getting worse. One of the main sectors that is the culprit of this is the social media industry, who can spy on private conversations and use artificial intelligence to gather more and more data on users. 62% of American adults say that they don’t believe that they can go through daily life without companies collecting data about them (source: Survey June 3rd – 17th 2019 by PEW Research Centre).

This is why I want to make a social media app that focuses on privacy so that people can feel happy about the way their data is handled. I believe that privacy is something that can be restored to the digital landscape, but for this, real working solutions need to be designed and built.

There are already some solutions that mimic the style of existing large social media companies but that focus on data privacy and user safety. There are still some gaps though, where some larger apps do not have privacy centred counterparts. I aim to fill one of these gaps: Discord.

**Computational Amenability**

Social media has already changed the world due to the speed of communication with the technology now being able to send messages and information across the world in seconds.

Messages can only be sent across the world in this time using the existing infrastructure and so it is essential that they are sent digitally through the internet to achieve the same ease of use. It would be completely impractical to communicate through any other method.

My app focuses on privacy, which I believe can be best achieved digitally. The use of modern encryption makes it near impossible to read people’s messages in transit. Any other methods of message transfer, such as post would be much less secure.

It’s much easier to reach a much larger number of people digitally, as 59% of the world’s population now has access to the internet. This is ideal because it’s much easier and more practical to meet people who are interested in the same things as you over the internet. And it also makes it a lot easier to meet friends of your friends who may live geographically quite far away from you.

An open voice chat system is also suits digital communication as well. Unlike a phone call, this would allow any number of users to join a call at any time of the day. Users can leave and join calls when they want with the click of a button.

**Stakeholders**

**Gamers**

This app would be ideal for gamers who want to chat to their friends and organise gaming sessions using large multi-user text chats. It would allow people to set up gaming communities which will allow them to communicate with their teammates and meet new friends that are interested in the same genres as them.

The app is suited to multiplayer gamers because it will be able to run on a browser in the background while they play, allowing them to use voice chats (with very limited delay) to help them win competitions by collaborating closely.

They can then break off into different channels if they decide to split off into subgroups that are no longer playing together.

**Students**

Students, who are interested in growing and excelling in their subjects, would be drawn to the app so that they would not always have to physically meet up with others to discuss topics that interest them.

The existence of chat hubs would allow students create online communities around their disciplines. Hub owners can name their community hubs after these disciplines. This would help them discuss their areas of interest with peers which would aid them in their education but also help them have fun after they have finished their work for the day.

The app will also be free and work with the computer or smartphone that they already own, meaning that using the app will be appealing to users that cannot afford to pay to send messages to others.

**Individuals That Are Privacy Conscious**

A very large proportion of websites track users and their data, so it’s no wonder that so many people are worries about what big companies are doing with their personal data. However, end to end encryption and transparency with the user will be at the forefront of my application. Account security is also very important, and so two factor authentication will be required on all accounts as well as a verified email.

Users will also be able to enter data on who should be able to view their account or send them messages or friend requests so that all users can stay safe and strictly control their own level of privacy.

There will also be the option to add a pin to your account so that you must enter a pin code before you are able to see any messaged that you have previously received or sent or any personal information that you have stored on your account, meaning that even with your login details compromised, you don’t have to worry about your data being stolen.

This group of people if becoming larger and larger as more and more is revealed to the public about how their data is handled. This would give my app a larger and growing userbase.

**Research**

**Existing Implementations**

Mastodon

One existing privacy focused social media app is Mastodon, which is quite similar to Twitter, but it focuses more on user safety and privacy. Mastodon is available on all devices that have access to a web browser and is completely open source and transparent.

The focus on privacy is an important benefit because it has features that allow you to block very specifically who can see your account and who cannot. This enables users to keep safe and I would like to copy this feature so that users will be able to block anybody that causes a threat to them. This would prevent any communication between the two users.

A screenshot of a computer

Description automatically generated with medium confidenceHowever, what I don’t like about Mastodon is that the user doesn’t choose what they are immediately exposed to as they first join the app (as seen in the image below). Instead, for my app, the user will have to seek out hubs that they are interested in, or friends that are already on the app.

I also like how simple the user interface is and will try and keep the user interface of my own app similar to existing social media websites so that transitioning away from them is easier.

I think overall it is quite a successful implementation that may become a rival to Twitter in the future.

MeWe

MeWe is another privacy centered app that is based on trust and the user’s interests, describing itself as “the fun social media experience you can trust, control and love”. It also has a mobile app and is quite accessible for a large range of users and devices.

One good point in favour of MeWe is that is has specific groups for people’s different hobbies and interests which allows people to discuss different topics in small and large communities. This is something that I would like to incorporate into my own app as I think it would benefit the users of the app and increase interaction while making new users more likely to stay.

Graphical user interface, text, application, chat or text message, email

Description automatically generatedHowever, the home screen of the app doesn’t look very modern and it is quite complicated (as seen below) for new users. I believe this is a significant drawback as it is difficult to learn and this could dissuade them from taking to MeWe.

**Conclusion**

Positives

The user should have fine control over the users that can communicate with them and see their account.

There should be a clean and modern look to the user interface that allows for new learners to easily understand how it works.

There should be hubs that the user can join that are based on specific topics.

The app should have a high level of account security including two factor authentication and there should be full transparency with how data is handled.

Negatives

The user interface should not be cluttered and difficult to understand.

The user should not immediately be subjected to content before they have decided that they want to see it.

**Requirements For The Proposed Solution**

**Development Requirements**

I require a computer that can run XXAMP, which will run Apache and MySQL, to function as the web server and database handling server. Ideally I could use a Raspberry Pi to host this as they are low power consumption, can be hooked up directly to a router by ethernet, and require little maintenance, so it could be kept online constantly and I could work on it remotely using a FileZilla file server. This would also allow for testers to connect at any time if I set up port forwarding on the router. This also means that I would need an internet connection and another computer to run a FileZilla client and connect to the Apache server and phpMyAdmin through a browser.

Other than this I will need time to research the programming languages that I will be using as well as time to think about the best way of implementing some of the features.

**End User Requirements**

The user would require simply a computer running a browser and a keyboard to use the app. JavaScript must be allowed to run in the browser.

However, to achieve full functionality, the user would also be required to have a mouse, microphone, and webcam. No other files, or software are required.

In the possibility that a desktop app is developed, then a small amount of space on the user’s drive will be required. The processing power required will also be minimal so there are very low system requirements:

* Dual Core Processor
* 1GB RAM
* An Operating System
* An internet connection

**Limitations of the Proposed Solution**

**Security**

There will most likely be security flaws in the final release of the app. This is because I do not have the knowledge required to keep it completely protected and so I can only do my best and ask for advice from others who have more experience in protecting websites from threats such as SQL injection and XSS attacks.

The server will not likely be at risk of attacks from hackers because the userbase will be small, so there will be little incentive.

**Servers**

The service will require a server to remain permanently online for the app to function correctly. This is a limitation because it will have running costs for whenever the app is available.

The server will be required to store account information for logging in and to store chats and route messages.

**Features**

The number of features will be always be behind other social media apps because I am the only person working on it. This is a problem because it would mean people would be less likely to switch from their current social media app to mine.

**Bugs**

Because there are so many different ways to interact with an app like this, it is likely that there will be some bugs in the final version. Although, this can be mitigated through rigorous testing.

**Success Criteria**

Here is a list of criteria that I would consider to be an indicator or how successful the project is

**Simplicity**

Simplicity is a requirement for keeping the app friendly to new users as well as reducing the number of bugs and security vulnerabilities. One feature that will make the interface more simple is the use of icons and text for buttons so that the user understands what they are clicking on.

**Modern User Interface**

A modern user interface is significant to maintain the app as a competitor to other social media apps which will encourage users to switch to the app. This is because some will not be so worried about privacy but instead just be looking for a smooth social media experience.

**Messaging**

A core function of the app should be that users can send messages to each other over the internet.

**Privacy**

Users should be able to block other users and their data should be stored securely. Users should know how their data is being handled and feel that the app is being transparent.

**Hubs and Interests**

Users should be able to locate and add other users as friends for easier communication with them. They should also be able to join community hubs for chatting.

**Security**

User’s passwords are stored as hashed values in a database so that if they are stolen then they will still not have been compromised.

**Logging In**

Users can create an account and log in later on to be able to continue the same previous chats, calls and discussions from previous sessions.

**Design**

**Methods, Variables, Classes and Parameters**

|  |  |  |  |
| --- | --- | --- | --- |
| **Module:** | **Script/Method:** | **Variable Names and Types** | **Purpose** |
| Account Handling | databaseConnection.php | $db PDO object | Is used as the connection to the database and is assigned using the databaseConnection.php script. |
| $servername String | Is used when creating the $db PDO object to point to the location of where the database is |
| $username String | Is used to define which user profile should be used to log into the database with |
| $password String | Is the password used to log into the database profile |
| $dbname String | Specifies which database at the location that should be logged into |
| $charset String | Specifies the character set that should be used so that errors aren’t encountered when certain characters are used |
| loginRequestHandler.php | $email String | Will be set using a $\_GET for testing and $\_POST in the final version. Holds the email address that is entered by the user. This is then used to find the required account in the table. |
| $password String | Will be set using a $\_GET for testing and $\_POST in the final version. Holds the password that is entered into |
| $db PDO object | Is set by requiring “databaseConnection.php”. It acts as an object that allows database interaction. |
| $sql String | Is set as the SQL statement in the form of a string that will be used to locate the account in the accounts table. |
| $stmt SQL Operation | Holds the process that enacts the statement in $sql on $db |
| $userdata SQL Select Data | Holds the data that is fetched from the server by the execution of $stmt |
| password\_verify() PHP Function | A built in PHP function that checks a string that has not been hashed against a string that has been hashed to check if they match up. |
| registerRequestHandler.php | $username String | Set by $\_POST from user form and will hold the username that the user would like their account to be called. |
| $email String | Set by $\_POST from user form and will hold the email that the user would like their account to be associated with. |
| $password String | Set by $\_POST from user form and will hold the password that the user would like their account to be accessible with. |
| $repeatPassword String | Set by $\_POST from user form and will hold the second time the user enters the password to reduce the number of instances where a user accidentally makes an account with the wrong password and can’t access it. |
| $token  String | The string will be a 19 character randomised string that should be kept secure for each user. It can be used in combination with the username and tag to verify messages are being sent by the real user. |
| $tag  String | Used as a visual way for the users to send friend requests to specific users even if they share the same username as other people. Set randomly, but must be checked against the database for other usernames so that they do not share both a username and a pin. |
| $id String | Made using the date, time, username, and tag. It acts as a unique identifier that does not have to be kept secure and can be used for looking up information from the account database by other scripts. |

**Algorithms**

**Flowcharts**

Displaying The History of A Chat To A User

Diagram

Description automatically generated

Set number of messages can be a constant that is adjusted for the best performance but allows for users to scroll up in the chat for a short distance before more messages have to be gathered from the database.

This is a required function because it allows the user to see messages that they and others have previously sent in a channel so they can continue a conversation after closing and reopening a chat or read new messages that have been sent to them.

Allowing A User To Send A Chat

**Diagram

Description automatically generated**

This is required for the core chat functionality of the app which allows the user to update the chat data table with their message so that users on the receiving end will be able to display it on their screen. It also includes the requirement for how to notify users that are not looking at the channel that there is a new message waiting for them.

It is important that we use this update system rather than a polling system because it is much more efficient. Otherwise we would essentially be either DDoS attacking our own servers which would exacerbate with more users or leaving user’s messages unchecked, increasing the time between when the sender sends the message and the receiver receives it.

Update chats on client

function checkAttributes(msg, updatedMsg)

newMsg = msg

for atr in attributes

if msg[atr] != updatedMsg[atr]

newMsg[atr] = updatedMsg[atr]

return newMsg

newChats = []

for msg in chats

newMsg = checkAttributes(msg, updatedChats[msg])

newChats.append(newMsg)

newChats.sort(DATE, NEWEST)

display(newChats)

Allowing A User To Log In

**Diagram

Description automatically generated**

PHP has built in functions to test if emails are valid and to protect against SQL injection and HTML has attributes that ensure that fields are not left empty.

To achieve privacy and security, it is essential that we can differentiate between users. Therefore, the log in system is required, because it allows us to do this securely.

This also opens up possibilities for advanced personalisation and preference options that allow the user to customise the GUI, such as through “themes” that would change the colour scheme of the app.

Furthermore, users will already most likely be used to using social media apps that use similar log in systems, and so it would be extremely learnable.

**Validation Methods**

General Inputs

All test inputs must be protected against SQL injection, otherwise this would be a massive security flaw. PHP has built in functions used to sanitise strings and validate inputs. To remove HTML tags, we can use the filter\_var($input,FILTER\_SANITIZE\_STRING) function which is important for displaying usernames and messages correctly. To protect the database operations, we can disable emulated prepared statements and use only real prepared statements. This should give an attacker no chance to inject malicious SQL. We can also introduce the data during the execution rather than while we prepare the SQL or prepare the operation. This ensures that the user’s inputted data is always treated as data, not instructions.

Email Input

Email inputs can be verified using filter\_var($email, FILTER\_VALIDATE\_EMAIL). I will not sanitize the inputted email, because then the email that is processed may be different than the intended email. Instead a warning can be thrown to the user about their input and the action can be cancelled. After this the user can amend the field.

Usernames

Usernames will be blocked from entering specific Unicode characters that may have connotations of racism, sexism, or other hate crimes. Also no spaces allowed.

Profile Pictures

Uploaded files must have an image file ending such as JPEG or PNG to prevent people uploading scripts. They should also be under 8MB in size. If they are larger than this then they can be compressed to see if they fit the size requirements after compression. If not, then an error can be thrown.

Passwords

Passwords should be longer than 6 characters and include numbers, at least one upper case and lower case letter, and a special character (!”£$%^&\*()@?<>~#’`¬|-\_=+{}[]:;,.).

Pins

Pins should be 6 whole numbers between 0 and 9

**Pseudocode**

Friend Requests

userId = GET(userId)  
friendName = GET(friendName)

validate(userId, ID)

sanitise(friendName, INVALID\_UNICODE)  
  
check = db -> SELECT \* FROM account\_data WHERE username = friendName  
  
if check != null

db -> UPDATE account\_data SET incomingFR = userId WHERE username = friendName  
 return “Success”

updateUsers()

else

return “Error: No user with that username”

Deleting A Chat

msgId = GET(msgId)

senderId = GET(userID)  
  
check = db -> SELECT \* FROM messages WHERE sender = senderId, id = msgId

if check != null

db -> DELETE FROM messages WHERE id = msgId

updateUsers()

**Usability**

**Learnability**

It should be easy for users to transition to my app from other social media apps. Therefore a system based on simple familiar buttons and icons is required. After looking at the way large social media apps display their buttons, I have noticed every button is accompanied with a visual clue that it is a button when you hover over it. Usually this hover effect includes some pop up text explaining what the button does, a change of colour of the button, or a subtle change of shape.

Graphical user interface, table

Description automatically generated with medium confidenceGraphical user interface, application

Description automatically generatedGraphical user interface, text, application

Description automatically generatedGraphical user interface, text, application

Description automatically generatedText, chat or text message

Description automatically generatedGraphical user interface, application, icon

Description automatically generated

Another commonality is that there are vertical menu options at the sides of the screen. This makes sense because it uses up horizontal space where it is to spare on landscape screen viewing ratios, and thus it has become our go to place to search for menus.

As well as this, similar icons for certain functions are used across the web. By using these icons it mean that a user will immediately understand what a button does, allowing them to complete tasks quickly and easily.

Background pattern

Description automatically generated

**Efficiency**



Every action should be accessible within a few clicks of the home screen.

This will be done using a series of scrollable panes that span across the screen and use the aforementioned familiar icons. These menus will include the hubs menu, the channel module, the chat section, and the online user list. These will switch to reflect the hub that you are currently interacting with, or the direct message list that you are currently viewing. This will be easy to navigate for new or returning users because they can always return to the home page with the large, easy to locate home button in the top left corner of the screen.

**Errors and Validation**

Very few actions cannot be undone and so it’s difficult to make permanent errors. With each option will be the option to undo it by deleting the message or flicking a switch back or renaming your account etc.

If users enter invalid data then no action will be taken and the user will be asked to amend the invalid field. After they have amended the field then they can try and submit the data again, after which it will then be revalidated.

When creating an account, the user will have to enter the password twice to ensure that on the first entry, they did not make a mistake. This is relevant because the password will be hidden from the client screen as they enter it for security reasons. It’s also important that they enter the right password so that they can still log in when they want to.

If users accidentally insert problematic and invalid characters into a field, we can alert the user of the problem.

**Satisfaction**

The GUI will be designed around usability, aesthetics, and satisfaction. This will be achieved by using undistracting block colours for buttons and backgrounds which will give it the modern and futuristic look. When hovering over icons, they may subtly change shape or colour. Each action may also include a fun animation. Icons will be basic and will keep the same meaning throughout the app as to not cause any confusion. The page must also respond quickly, because a slow page can be frustrating.

**Testing**

**General Testing**

|  |  |  |  |
| --- | --- | --- | --- |
| Number of Testers | Method | Instructions | Explanation |
| 20 | Blackbox | Testers will be instructed to use the app how they normally would and explore the different available menus | This is to uncover any bugs that users may encounter when using the app normally. |
| 20 | Blackbox | Testers will be instructed to try and break and interact with the app as much as possible by spamming different inputs. | This is to find problems that may arise when a regular user is specifically trying to break the app using a brute force method. |
| 2 | Whitebox | Testers will be able to look through the code and try different actions on the site in order to try and break it. This includes entering invalid characters, formats and SQL code. | This is to reveal any bugs that may be very specific, or that could be targeted by attackers. |
| 1 | Iterative | I will test each feature after having developed it to make sure that it is working as intended. | This is important because when a user is using the app normally and as intended, then it should definitely not have bugs. |

**Specific Testing**

|  |  |  |  |
| --- | --- | --- | --- |
| Feature Being Tested | Method | Expected Output | Real Output |
| Account Registration | Whitebox.  The tester should first enter the registration screen and follow the on-screen steps to making an account. After that, they should repeat the test again. When making the account, they should use the inputs given below. The data from Login 2 should not be accepted so they should then proceed to Login 3 and end up with 2 successfully make accounts.  Data: Login 1  Username: “test\_user1”  Password: “Pa$$w0rd”  Pin: “123456”  Pfp Image: Any small image file  Login 2  Username: “<b>\bad\_test\_username</b>”  Password: “smol”  Pin: “dwed”  Pfp: Any small executable file  Login 3  Replace login 2 information -  Password: “Pa$$w0rd”  Pin: “654321”  Pfp: any small png | Checking the account\_data data table afterwards reveals that 2 new rows have been made, and the username and tag, token, and id are all unique. The first username should be as entered but the second username should have had the HTML tags and backslashes removed. The second password should be too small and therefore disallowed and the second pin should not be accepted.  The hashed passwords on the two different rows should be different if they have been salted. |  |
| Account Login | Blackbox  This test can be completed after the account registration test – it is essential that the data table contains only the result of this test when successful, and that all other entries have been removed. The user should enter the three sets of data (logging out between each) that they see below and see if it allows them to view an account and the message history.  Data  Set 1  Username: “test\_user1”  Password: “Pa$$w0rd”  Pin: “123456”  Set 2  Username: “not a real username”  Password: “Pa$$w0rd”  Pin: “123456”  Set 3  Username: “test\_user1”  Password: “Not a real password”  Pin: “324443” | They can view an account on the final set of data, but the tester is warned about their other inputs. |  |
| Friend Requests | Blackbox  Requires two testers. Both testers should be logged into separate accounts and should know the username and tag of each other. The first tester should navigate to the “add friends tab” and enter the information of the other tester into the required fields. They should then send the friend request. The second tester should then refresh their page.  The first tester should then enter the username “sBeve” and pin “3423”. | The second tester should see an option to accept or block the friend request and the first tester’s username and tag.  The second request should not come through and should just appear as a warning on the first tester’s screen that their inputs are invalid. |  |

**Databases**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Token | Email | Username | Pin | Hashed Password | PFP Token | pubKey | Friends | Friend Requests to | Friend Requests from | Channel Notifications | Channel Pings | Personal Description | Blocked Users |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |

**Account Data**

The account data table is used for storing information on users, most of which will need to be accessed during (blue), or right after (green) login. Other will need to be accessed as the user is interacting with the app.

After logging in, the information will be processed and automatically sent to the client to process the visuals. For example, the PFP (profile picture) token is a unique identifier used to locate the profile picture in a directory on the server. This image will then be sent to the client to be displayed at the top right of the screen.

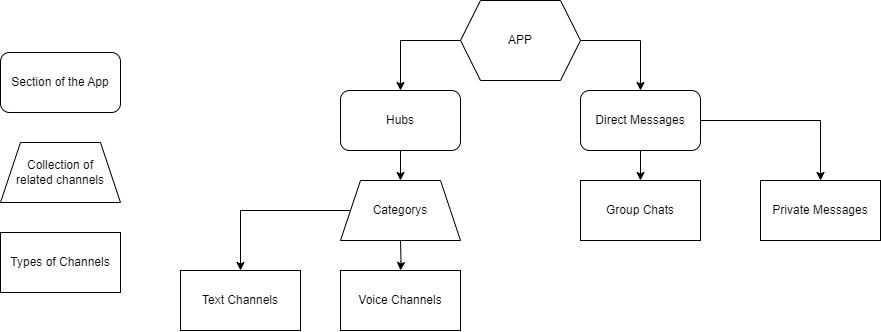
It is also essential that a client can access some information about other users, such as the PFP token, username, and personal description to help the user to identify exactly who they are speaking to.

**Messages**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ID | Time & Date Sent | Edited | Channel ID | Sender ID | Text Contents | Image Contents | Pinged Users | Verified | File Contents |
|  |  |  |  |  |  |  |  |  |  |

A database of messages is required, because some messages will need to be accessed by any user at a moments notice. This is because most channels in hubs will be public, meaning users will only need an invite, or for public hubs, a search, to be able to access the channel.

Message storage will be split into “channels”. A channel will be a link between users. It could be direct, only including two users that wish to talk privately, or it could be accessible by hundred of users that all have the same interests.



Key

**Hubs**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | Name | Number of Users | Date Created | Directory Location | Owner |
|  |  |  |  |  |  |

The information on how a hub should be organised and displayed needs to be kept as long as the hub is around. As hubs are supposed to be permanent social centres, the information must be stored keeping long term access requirements in mind. I have opted to store the information in both a specially made directory and data table row. The main server properties and attributes can be stored in the database (as seen above), and larger, more complex data can be stored in XML, image format, and JSON files in the directory.

Examples seen below:

Text

Description automatically generated

**Graphical user interface, text

Description automatically generated**

Text

Description automatically generated

**Graphical User Interface Designs**

**First Ideas**

This is my first idea about how the app may look. I’ve abstracted and labelled buttons.

Graphical user interface

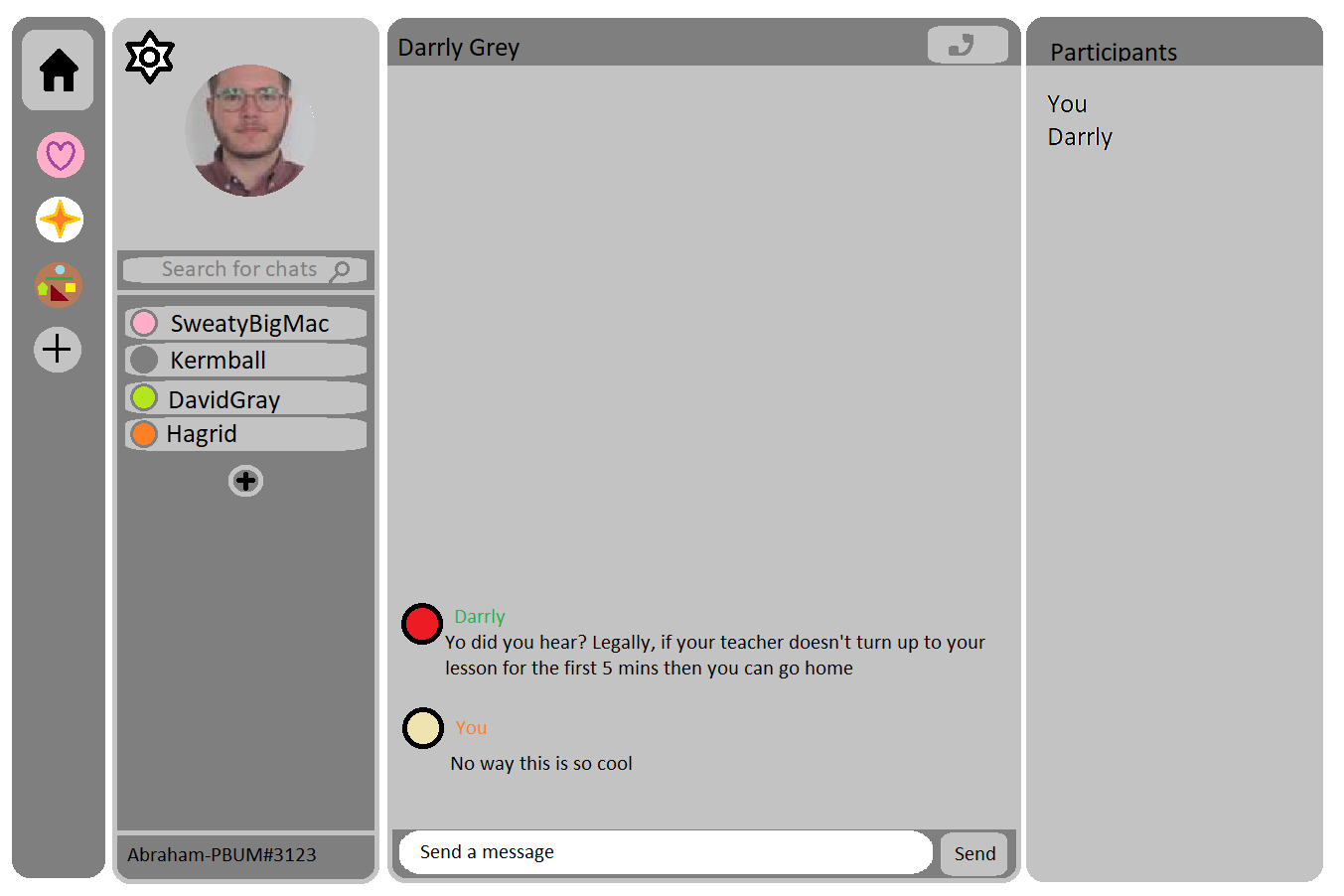
Description automatically generated

I then made this to get an idea of what this would actually look like. I don’t really like the colour scheme right now but that can be finely adjusted when I am using CSS to tweak the look.

Graphical user interface, text, application

Description automatically generated

Finally I then removed the thick black borders to make it look more modern.



**Development**

**Prototype 1.0**