Figure 1: Wireframe of the post board

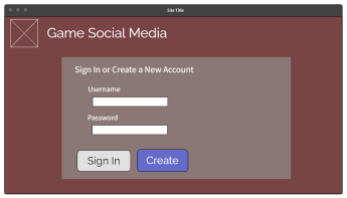

Figure 2: Wireframe of the login page

Figure 3: Wireframe of the create-post page

Figure 4: Wireframes of Game Board, which is used to represent user flows and page relationships

CS5041 P2 REPORT

210000448, 180021533

The Design of Game Board

This project is a social media called Game Board, where users can post their game experiences and share them with others. The main activities on Game Board are logging in and out with a username, posting, viewing own and other users' posts on post board, and reacting to posts with various emojis.

By logging in with a username, the username will be displayed to the public as the author of the post. Username can be changed at any time by logging out, and the user can set username according to the game and the experience, or the mood of the moment to make the post more fun.

Users will input their post content on the create-post page. The post will be divided into three parts, title, game name, and experience, none of which can be empty.

On the post board page, users can view their own and others' posts. In addition to the post content, the author of the post and the posting time are also displayed. The page also has access to Steam API to fetch game profiles, which will be displayed to the right of each post, allowing users to browse directly for information about each game rather than searching for them.

When browsing posts, users can react to each post with various emojis to express their feelings about the post. It is an interactive design that allows users to communicate in a simple way, which gives them a sense of sharing. Users are able to view feedback from others on their posts, as well as react to others' posts.

The Design Implementation Process

Wireframes

After the initial design was complete, we began creating wireframes while simulating the use of the Game Board based on user scenarios.

During the initial design discussions, we clarified the activities for each page and the components needed for the activities. At this stage we presented the above information by wireframes.

Figure 1 shows the wireframe of the home page, the post board, which has access to the login and create post pages, as well as the view of all game experiences posted on the platform and their reactions.

Figure 2 shows the wireframe of the login page, this part was designed to provide a login box for entering a username and password, as well as providing both login and new account functions which would be modified in the later implementation.

Figure 3 shows the wireframe of the create post page, providing input boxes for title, game name and content respectively, and a rich text editor for the content input box. It was also assumed that the user must log in before posting, so the current username is also displayed in the top right corner.

Once the wireframes for the main pages were completed, a wireframe prototype based on user flows and scenarios was created (Figure 4) for designing and adjusting the logic of page buttons and page jumps. In this section, changes have been made to enhance the user experience by adding a cancel button on the login page and the create post page, and changing the login button to log out on the main page after logging in.

Implementation of the login page

In the initial design, the login page would have provided the login and create new account functions, allowing users to

Figure 5: The Post Board

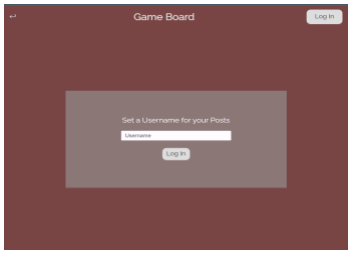

Figure 6: The Login Page

Figure 7: The Create Post Page

access and use the Game Board repeatedly with the same account. However, as firebase auth only provided signInAnonymously, we had to remove the "user" concept and instead design a login system that only creates a username, which this username will be displayed on the post board as the author of the post. To keep the username while viewing all pages, the code for this login system is implemented by Windows.LocalStorage, so that the username is also accessible and displayed during creating post and on the post board.

The logic of this page is that when the user does not enter anything and clicks login, the system will prompt "username cannot be empty"; when the user enters a username and clicks login, the page will jump to the home page and the username will be displayed in the top right corner and will act as the logout button. When the user wishes to discard the creation and return to the home page, they can do so by clicking the navigation on the top left corner.

The user interface on this page (Figure 6) is designed to be uniform with the overall style, retaining the banner with "Game Board" and giving the user the message "Set a Username for your Posts" to avoid confusion. The layout of all components is centred and follows the user's logic (browse information - enter username - confirm) from top to bottom, in line with the visual flow.

Implementation of the create post page

in the initial design, the create post page would provide the creating and editing post functions. To fetch the game information, there is a text-input box for game name, in addition to two input boxes for the post title and body. We provided a rich text editor for styling the post content.

The code for creating post is implemented by writing data to firebase, when user confirms to post, the post data will be sent to the "/public/uid" and stored.

The logic of this page is that the post button sends the post data to firebase, and then navigates to the post board, the cancel button takes user back to the post board and the input will not be saved.

The user interface on this page (Figure 7) is designed to uniform with the overall style, retaining the banner with "Game Board" and giving the user the message of page usage "Create a new post" to avoid confusion. Each text input box has a placeholder to tell users what they are for, title, game or content. The layout of all components is centred and follows the user's logic (browse page title – add post content – confirm) from top to bottom, in line with the visual flow. The "Title" and "Game" inputs use the "TextInput" from "react-native-paper", as this provides placeholder text so information can be condensed by avoiding explicitly labeling these inputs. For the "Body" input, the "Editor" from "react-draft-wysiwyg" is used, which use "draft-js" to implement a rich text editor – this is a special object, converted to HTML using "draft-to-html" to be stored in the database.

Posts

The main page of the app is the list of posts, sorted chronologically from the most recent. This uses "Dimensions" from react to dynamically set its height and width to fit the window – though for some reason, the height doesn't work. For layout, the page has a "View" with flex direction row to place a "View" either side of the list of posts, so the posts remain at a comfortable size. The "public" list from firebase is loaded and watched using hooks, with the value being filtered to find posts and reactions using the assigned types containing the "key" for our app – these are exported in a separate file for convenience. The list of posts is then sorted chronologically (though the list should already be chronological due to automatic id generation provided by firebase), and mapped to "Post" elements.

"Post" elements use flex direction row to place content on the left and right of the main content of each post, with emphasis placed on the main content through it being contained in a box with a background and having a larger font size. On the left side, meta information is given – the username, date and time

posted, and game posted about, with the date and time given a smaller font size due to its lower importance. The main content of the post has the post title and body, using "react-html-parser" to retrieve the html elements defined when generated from the rich text input. The main box is set to "stretch", and is given a minimum width and height, so the main content of the posts can resize based on the size of the window while also having a consistent width across the whole list. To the right of the main content is more information about the game, taken from Steam.

Finally, at the bottom right of the page is a floating action button from material UI – the "create post" button. Using a floating action button allows it to remain in a fixed position on the window regardless of the scrolling list of posts. This button checks the state of the "username" store to check that the user has set a username, denying navigation to the create post if not, and instead directing them to the "log in" page for convenience.

Reactions

As stated prior, the reactions are filtered out of the list of all messages when loading from firebase. Since the post id is used as the message for reactions, they can easily be filtered again to provide each "Post" element with the list of reaction messages for only that post. Using the list of reaction messages, we can check whether the username has already been used to add a certain reaction to a post, preventing a user from adding an arbitrary number of the same reaction without going tediously finding a new username every time. Since there's no guarantee that a user has the same uid when the author's name matches the current user's username, reactions can't be removed. The list of reaction messages is tallied to find the count for each different emoji, which is displayed horizontally at the bottom of the post, using "Arial" rather than the usual "Raleway" since it displays the numbers more nicely.

For adding reactions, the material ui "Select" dropdown is used, which allows an arbitrary list of emojis to be used as reactions, as defined in the exported reactionlist. This component was chosen as other dropdowns would display underneath the subsequent post, making them effectively unusable. The button to open the dropdown has a "+" inside it, to make its purpose of adding reactions clearer due to its placement next to the display of existing reactions on the post. Upon selecting a reaction, a message is sent to the database using the reaction-key as the type, the post id as the message, and containing the author and reaction in the content.

Game details

Details about the games are fetched from Steam, which allows details to be loaded about a game using a request with its app ID. To find the id of a game by name, the list of all steam apps must be acquired, which requires accessing the Steam Web API using an api key. Both requests have CORS disabled, so implementing this feature required routing these through a backend server. Thankfully, a project to do this easily - "cors-anywhere" - already existed, so this was pulled into the project and run to provide a tunnel for requests to be able to fetch data from steam.

The list of games is a cached request made on loading the main page, with the response being inverted from a list of entries by app ID to a javascript object mapping game names to app ID, so these could be found quickly. To make finding the app id more reliable, both the names received from steam and the names found on posts are stripped of non-alphanumeric characters, set to all lower case, and have leading and trailing whitespace removed.

Fetching the game information from Steam is a "useEffect" hook with the game list as a dependency, so it updates once the game list loads, finding the app id from the game list to fetch the game information. A lot of information is returned, but I decided to use the header image for the game as the header for the game information, with no explicit use of the game's name – the game is already named to the left of the post, and the header typically contains the name in the image and is recognisable. Below the header are the lists of developers, publishers and genres, and below that the game

details, in a slightly larger font to distinguish it from the other, less important information. The game details contains some html, which is stripped along with unnecessary new lines, resulting in pure text which can be used in a "ReactReadMoreReadLess" component, which limits the display to a certain character limit and provides a toggle to extend to the full information. This allows the game details to fit comfortably beside the post, even if the full details are very long, while still allowing the user to see the full details if desired.

***Username* state**

React redux toolkit is used to create a shared state store for the app so any element can examine the username state. This also handles storage into and loading from "localstorage", allowing the selected username to persist after closing and re-opening the app.

Navigation

Navigation was originally provided through Stack navigation, but this caused an unusual error with react hooks, which are used frequently to provide a reactive experience. This error was originally solved by moving all hooks outside of the direct child of the stack screen, but this didn't work for the posts, so navigation was changed to using react router, which eliminated this problem with only minor changes to most of the app. The only major change was how the "Banner" worked. This was originally provided to the stack navigator as a custom header, but react router has no header. The "Banner" needs navigation functionality, but it cannot have this outside of a "routes" components, and "routes" can only have "route" components as children. To fix this, I made "banner" a wrapper, giving it the page component as a prop so it can display the banner at the top of the screen along with the rest of the page. The "Log in" button uses redux to update itself once the user has "logged in" (set a username), swapping the "Log in" button for a "Log out" button which displays the username while not hovered, using the "mouseEnter" and "mouseLeave" callbacks. The "Log out" button is set to display the username when pressed, so the button correctly displays this if the user logs in after logging out.

Reflection - 180021533

After we each made wireframes mocking up our visions for the project, we decided to go with my idea, with me taking the lead role. I took the initiative when suggesting how tasks were delegated, and rough deadlines for completing those tasks. I was responsible for creating the overall code layout of the project, as well as handling the banner posts page, including the logic for loading the posts and reactions, loading game information from steam, and adding reactions, along with the overall implementation of styling for the project, including the inclusion of the rich text editor for creating posts. I think the groupwork went well, with solid and prompt communication throughout the project, and good delegation of tasks so we would interfere with each other's work as little as possible. I already have experience working with React, so this project served mostly as a refresher on how to use the framework. I also learned how to quickly make a good wireframe using mockflow's wireframepro tool and the fundamentals of working with firebase and building an app which only signs in anonymously to the database – since anonymous sign-in provides a unique id which all messages are stored under, data in the database at the end of a session effectively becomes exclusively read-only. I learned how to use the stack navigator, then refreshed my knowledge on using react router (which I previously used and has since had a significant update). I also learned how to manage state across an app with redux and reinforced my knowledge on the tricks to create certain stylings in React (such as elements which fill the window, and creating space around elements, such as the space either side of the list of posts to keep posts at a comfortable size.