



National College of Ireland

Digital Business Transformation

Academic Year i.e. 2022/2023

James Croke

x17480714

x17480714@student.ncirl.ie

ShadeStack

Technical Report

Contents

Contents.....	2
Executive Summary.....	3
1.0 Introduction	4
1.1. Background	4
1.2. Aims.....	4
1.3. Technology.....	5
1.4. Structure	7
2.0 System.....	7
2.1. Requirements.....	7
2.1.1. List of Functional Requirements	7
2.1.1.1. Use Case Diagram	8
2.1.1.2. Termination/Post Condition Tags:	9
2.1.1.3. Alternative/Exceptional Flow Use Cases	9
2.1.1.4. Functional Requirements	9
Functional Requirement 1: User Registration.....	9
Functional Requirement 2: User Login	10
Functional Requirement 3: User Account (Update Profile)	11
Functional Requirement 4: Create Room	12
Functional Requirement 5: Add User.....	13
Functional Requirement 6: Remove User	14
Functional Requirement 7: Message Room.....	15
Functional Requirement 8: Message Persons.....	16
Functional Requirement 9: Review Room Messages.....	17
Functional Requirement 10: Review Personal Messages	17
Functional Requirement 11: Search.....	18
Functional Requirement 12: Details	19
Functional Requirement 13: Block.....	20
2.1.2. Data Requirements	21
2.1.3. Environmental Requirements	21
2.1.4. Usability Requirements.....	21
2.2. Design & Architecture / Implementation	22
2.3. Graphical User Interface (GUI).....	34
2.4. Testing.....	36
2.5. Evaluation	53

3.0	Conclusions	53
4.0	Project Plan	53
	54
5.0	Further Development or Research	54
6.0	References	54
7.0	Appendices.....	55
	7.1 Reflective Journals	55
	7.2 Project Proposal.....	70
	Objectives	72
	Background	72
	Technical Approach.....	74
	Technical Details	75
	Special Resources Required	77
	Project Plan	77
	Testing.....	79

Executive Summary

ShadeStack is a web application currently in development for browsers, using multiple additional technologies to allow shrouded or unknown players within the gaming community to search for other players of their skill level on platforms such as Xbox, PlayStation, and pc.

In whatever game they are choosing to play, the users will be able to create a room with their own specific parameters or join another room to team up with players and create their own temporary posse or permanent clan.

Once a player has joined a room, they can communicate with other users of that room to figure out how they are going to connect.

This room can also be used as a 3rd party for communication as some games may have cross-play, but their independent platforms does not support the use of text communication between each other.

Rooms can also be created for the sole purpose of gamers to socialize and talk if they wish about a game.

The application is more aimed towards the professional finesse rather than casual as the target market would be players of high skill or rank within their game to form chosen teams for Esports. This encourages the development of professional gaming and the market itself

then, as players will generally compete within tournaments on platforms such as GameBattles to win cash prizes, or slowly increase their skill for their own benefit.

1.0 Introduction

1.1. Background

Throughout my years of getting older, gaming has been a huge impact towards my balance of a healthy lifestyle with work and hobbies. It has enabled me to procrastinate from where I am [usually] based to instantly being able to jump into some work and repeating the cycle after some burnout.

Within my 17 years of gaming, I have been majorly involved within the social aspect of casual and professional playing, with communicating randomly to persons in online matches to seeking a more objective oriented version to communicating. This can contain looking for specified players with certain skill sets, to looking for specified groups with certain skill sets, generally known as clans.

These clans range from an entire plethora of AAA titles to smaller independent developer games. With the development of platforms such as Xbox, PlayStation, Steam and others like Nintendo, Nvidia, and Google Stadia, it has allowed the progression of communication to find players for groups better but not full as an entirety.

Generally, with these platforms it is not as simple as just going to a brand forum or dedicated communication area to find players, it is a lot more complicated with layers upon layers of verification to then initialize a post for a high possibility of no one to join due to the extra complications.

With ShadeStack the aim of development is to have enough layers of verification there but make the application easily accessible to login and instantly chat within a thread to find a player of choice or even go through extra steps to investigate player skill, games, and connection stability through profiles. Platforms are all included within the user details.

1.2. Aims

Main Objectives of ShadeStack are as follows:

- Easy to navigate UI
- Retrieve and store relevant user information (e.g. games on list, name, age, gender, platform)
- Creation of rooms w/ adding and removing users
- Search for users
- Personal Messaging

- Real Time Chatting/Messaging

1.3. Technology

Developed through IDE's such as VS Code Shadestack will be created using React Routing, Social Auth, Firebase, The React Context API, Chat Engine, Environment Variables,, REST APIS, and standard web development languages such as HTML, CSS and JavaScript. Identification of user requirements have been made through personal recommendation and knowledge throughout the years of gaming but, will also be gathered from persons after consent forms signed and options been chosen regarding the format of ethics.

The technologies stated within the previous heading (technical approach) are to be used and are as follows.

VS Code

Integrated development environments developed by Google and Microsoft used to edit source code that can be used alongside a variety of programming languages, including Java, JavaScript, Go, Node.js, Python, C++ and Dart. VS being built on the Electron framework, used to develop Node.js applications for web running on the Blink layout engine.

React Routing

React (also known as React.js or ReactJS) is a free and open-source front-end JavaScript library for building user interfaces based on UI components. React Router is a powerful routing library built on top of React that helps you add new screens and flows to your application incredibly quickly, all while keeping the URL in sync with what's being displayed on the page. React is only concerned with state management and rendering that state to the DOM, so creating React applications usually requires the use of additional libraries for routing, as well as certain client-side functionality.

Social Auth

Login authentication through 3rd Party applications.

Firebase

A platform developed by Google for creating mobile and web applications. Originally an independent company founded in 2011. Now in the year of 2021 it has been 9 years since Google acquired the platform and is now one of their flagship offerings for app

development providing a whole plethora of services for app and web development i.e., Firebase for storing information which it is widely used for.

The React Context Api

The React Context API is a way for a React app to effectively produce global variables that can be passed around. This is the alternative to "prop drilling" or moving props from grandparent to child to parent, and so on.

Chat Engine

Chat Engine is an API providing a REST API, and NPM components to help with chat UI.

Environment Variables

EVs let you store globally scoped values to the environment your code is running in, making them available throughout the codebase.

Rest APIs

A REST API (also known as RESTful API) is an application programming interface (API) that conforms to the constraints of REST architectural style and allows for interaction with RESTful web services.

HTML

The HyperText Markup Language is the standard markup language for documents designed to be displayed in a web browser. It can be assisted by technologies such as Cascading Style Sheets (CSS) for decorating a page and scripting languages to run commands.

CSS

Cascading Style Sheets (CSS) is a style sheet language used for describing the presentation of a document written in a markup language such as HTML.

JavaScript

Often abbreviated as JS, is a programming language of high-level, often just-in-time compiled and multi-paradigm. It has the ability of dynamic typing, prototype-based object-orientation and first-class functions.

Alongside HTML and CSS, JavaScript is one of the core technologies of the Worldwide WEb. Over 97% of websites using it client-side for web page behaviour, often including third-party libraries for additional functionality.

1.4. Structure

Throughout the remainder of the report structures as follows

Section 5 will outline the key stakeholders of ShadeStack, Use Cases and Functional Requirements.

Section 6 will explain the data requirements.

Section 7 contains the User, Environmental and Usability Requirements.

Section 8 will outline ShadeStack's system design and architecture.

Section 9 will describe the technical implementations its key features.

Section 10 contains images describing the graphical user interface.

Section 11 illustrate the different tested methods

Section 12 will cover evaluation conclusion

Section 13 will discuss further research

Section 14 include referral

2.0 System

2.1. Requirements

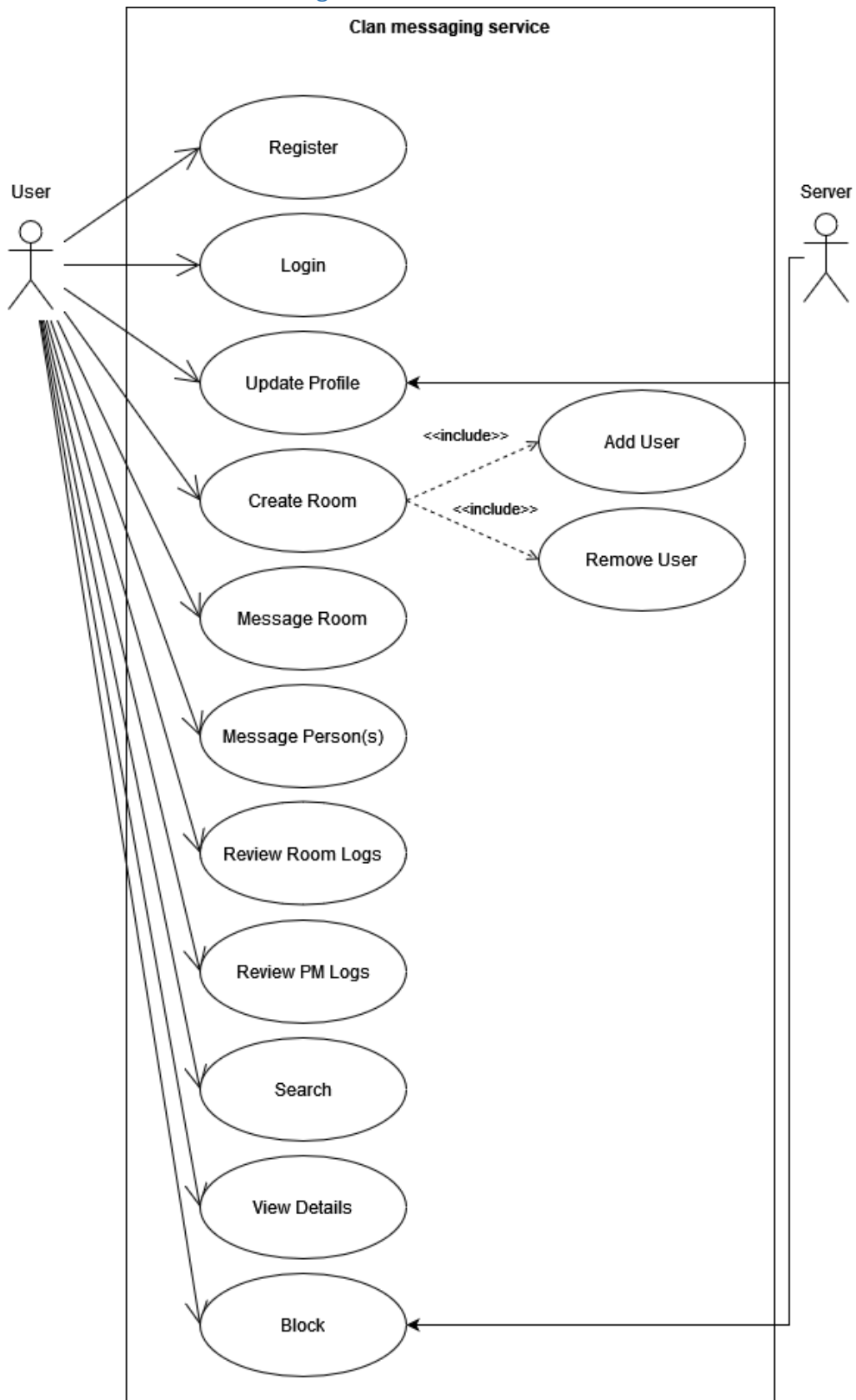
All requirements are verifiable. For example, users shall be able to use all system functions contained within the application.

2.1.1. List of Functional Requirements

1. Users can create an account for the app
2. Users can login to the application with the created account.
3. Users can fill up their profiles with standard general information within gaming such as Name, Age, Gender, Games, and Ranks within those games.
4. Users can create rooms
5. Users can message in rooms
6. Users can message persons
7. Users can review room messages

- 8.Users can review personal messages
- 9.Users can search for players
- 10.Users can look at the details of the accounts
- 11.Users can block users

2.1.1.1. Use Case Diagram



2.1.1.2. Termination/Post Condition Tags:

<TNOTIFY>	User is notified of a successfully completed use case
<THOME>	User is returned to the homepage
<TUSER>	User is admitted to the app
<TWAIT>	System is waiting

2.1.1.3. Alternative/Exceptional Flow Use Cases

<INVALID>	<ol style="list-style-type: none">1. User is deemed to be invalid2. User is notified of invalidity3. User is required to rectify invalid information4. New entered data is verified to be validated
<CANCEL>	<ul style="list-style-type: none">-User selects to cancel-Process proceeds-Changes do not apply

2.1.1.4. Functional Requirements

Functional Requirement 1: User Registration

Purpose: All users are required to create an account to access Shade Stack

Priority

HIGH

Use Case:

Title	User Registration
Tags	<FRACCOUNT> & <FRPROFILE>
Scope	The scope of this use case is to allows users to sign up and create an account to appear on the application. Upon creation, the accounts will be allowed to store data viewable to other's.

Actors	User
Precondition	The user must enter into the application
Activation	User creates an account
MainFlow	<ul style="list-style-type: none"> -User indicates they are seeking to join -Data is validated by the system -Application generates a new account - Values for the account are stored within the database - User is brought to their account management page
Exceptional Flow	<INVALID>
TERMINATION	<p>SUCCESS: <TNOTIFY></p> <p>FAILURE:<THOME></p>
POST CONDITION	<p>SUCCESS: <TUSER></p> <p>FAILURE:<TWAIT></p>

Functional Requirement 2: User Login

Purpose: Users must sign into their account to access the application

Priority

HIGH

Use Case:

Title	Sign In
Tags	<FRSIGN>
Scope	The scope of this use case is to allow Users to sign into the system to use the application. Users are validated and can only access the authorised accounts.
Actors	User
Precondition	The user has created an account with the

	application prior to this attempt
Activation	User signs in
MainFlow	-The system prompts the user for their login requirements -The User entered the information -The system ensures the information is correct - User is taken to the homepage
Exceptional Flow	<INVALID>
TERMINATION	SUCCESS: <TNOTIFY> FAILURE:<THOME>
POST CONDITION	SUCCESS: <TUSER> FAILURE:<TWAIT>

Functional Requirement 3: User Account (Update Profile)

Purpose: To update information on an account

Priority

HIGH

Use Case:

Title	Update Profile
Tags	<FRACCOUNT>
Scope	The scope of this use case is to allow users to edit the varying information chosen to be displayed onto the account.
Actors	User, Server
Precondition	The user must have an account with information already provided
Activation	The user edits profile
MainFlow	-System displays existing data

	<ul style="list-style-type: none"> -User alters information -System validates data -Profile is updated
Exceptional Flow	<p><CANCEL></p> <p><INVALID></p>
TERMINATION	<p>SUCCESS: <TNOTIFY></p> <p>FAILURE:<THOME></p>
POST CONDITION	<p>SUCCESS: <TUSER></p> <p>FAILURE:<TWAIT></p>

Functional Requirement 4: Create Room

Purpose: To update information on an account

Priority

HIGH

Use Case:

Title	Create Room
Tags	<FRROOM>
Scope	The scope of this use case is to allow users to create a room for users to message into to connect together.
Actors	User
Precondition	The user has created an account with the application prior to this attempt
Activation	The user creates a room
MainFlow	<ul style="list-style-type: none"> -System displays room creation action -User clicks on create room -User enters room name -User enters other parameters

	-User creates room
Exceptional Flow	<CANCEL> <INVALID>
TERMINATION	SUCCESS: <TNOTIFY> FAILURE:<THOME>
POST CONDITION	SUCCESS: <TUSER> FAILURE:<TWAIT>

Functional Requirement 5: Add User

Purpose: To update information on an account

Priority

HIGH

Use Case:

Title	Add User
Tags	<FRROOM>
Scope	The scope of this use case is to allow users to add other users into the room.
Actors	User
Precondition	The user has created or joined a room prior to this attempt.
Activation	The user adds a user
MainFlow	-System displays the add user action -User clicks on add user -User enters specific name -User adds user
Exceptional Flow	<CANCEL> <INVALID>
TERMINATION	SUCCESS: <TNOTIFY>

	FAILURE:<THOME>
POST CONDITION	SUCCESS: <TUSER> FAILURE:<TWAIT>

Functional Requirement 6: Remove User

Purpose: To update information on an account

Priority

HIGH

Use Case:

Title	Remove User
Tags	<FRROOM>
Scope	The scope of this use case is to allow users to remove users from a room.
Actors	User
Precondition	The user has created or joined another room prior to this attempt and is admin.
Activation	The user removes a user from a room.
MainFlow	-System displays remove user action -User clicks on remove user -User enters name -System checks admin rights. -System validates user. -System removes user from room
Exceptional Flow	<CANCEL> <INVALID>
TERMINATION	SUCCESS: <TNOTIFY> FAILURE:<THOME>
POST CONDITION	SUCCESS: <TUSER>

	FAILURE:<TWAIT>
--	-----------------

Functional Requirement 7: Message Room

Purpose: To send messages to another user

Priority

HIGH

Use Case:

Title	Room Messaging
Tags	<FRMESSAGEROOM>
Scope	The scope of the use case is to allow users to send messages within the room to each other.
Actors	User
Precondition	The user must have created or joined a room.
Activation	The user sends a message
MainFlow	-User enters existing room -System displays text box -The user types a message to send -The user clicks send -The message is sent and saved to the room
Exceptional Flow	<CANCEL> <INVALID>
TERMINATION	SUCCESS: <TNOTIFY> FAILURE:<THOME>
POST CONDITION	SUCCESS: <TUSER> FAILURE:<TWAIT>

Functional Requirement 8: Message Persons

Purpose: To send messages to another user

Priority

HIGH

Use Case:

Title	Personal Messaging
Tags	<FRMESSAGEPM>
Scope	The scope of the use case is to allow a user to send a personal message to another user.
Actors	User
Precondition	The user must have another person to message
Activation	The user sends a message
MainFlow	-System displays search box -The user searches for the person they want to message -User clicks on account -System displays account -User clicks on message account -System displays text box -User types message -User sends message -System sends and saves message
Exceptional Flow	<CANCEL> <INVALID>
TERMINATION	SUCCESS: <TNOTIFY> FAILURE:<THOME>
POST CONDITION	SUCCESS: <TUSER> FAILURE:<TWAIT>

Functional Requirement 9: Review Room Messages

Purpose: To view previous chat logs

Priority

MID

Use Case:

Title	Review Rooms Messages
Tags	<FRROOMLOGS>
Scope	The scope of this use case is to review any previous messages saved and sent to a room.
Actors	User
Precondition	The user must be existing within that room
Activation	The user views previous sent messages
MainFlow	-User enters room -User reviews messages sent
Exceptional Flow	<INVALID>
TERMINATION	SUCCESS: <TNOTIFY> FAILURE:<THOME>
POST CONDITION	SUCCESS: <TUSER> FAILURE:<TWAIT>

Functional Requirement 10: Review Personal Messages

Purpose: To view previous chat logs

Priority

MID

Use Case:

Title	Review Personal Messages
-------	--------------------------

Tags	<FRMESSAGELOGS>
Scope	The scope of this use case is to review any previous messages received or sent.
Actors	User
Precondition	The user must have sent or received messages from a user.
Activation	The user views previous sent messages
MainFlow	-The user reviews existing messages -The user clicks on an account -The user reviews the messages
Exceptional Flow	<INVALID>
TERMINATION	SUCCESS: <TNOTIFY> FAILURE:<THOME>
POST CONDITION	SUCCESS: <TUSER> FAILURE:<TWAIT>

Functional Requirement 11: Search

Purpose: To look for another user

Priority

HIGH

Use Case:

Title	User Search
Tags	<FRSEARCH>
Scope	The scope of this use case is to allows users to search for other players within the application.
Actors	User
Precondition	The user must save a name to search for

Activation	The user searches for an account
MainFlow	-System displays search bar -User inputs name -User is presented
Exceptional Flow	<INVALID> <CANCEL>
TERMINATION	SUCCESS: <TNOTIFY> FAILURE:<THOME>
POST CONDITION	SUCCESS: <TUSER> FAILURE:<TWAIT>

Functional Requirement 12: Details

Purpose: To view details of an account

Priority

HIGH

Use Case:

Title	View Details
Tags	<FRDETAILS>
Scope	The scope of this use case is to allows user to view details of accounts
Actors	User
Precondition	The user must have an account to view
Activation	The user views the account
MainFlow	-The user clicks to view details of an account after searching
Exceptional Flow	<INVALID> <CANCEL>

TERMINATION	SUCCESS: <TNOTIFY> FAILURE:<THOME>
POST CONDITION	SUCCESS: <TUSER> FAILURE:<TWAIT>

Functional Requirement 13: Block

Purpose: To block another user

Priority

HIGH

Use Case:

Title	Block
Tags	<FRBLOCK>
Scope	The scope of this use case is to block another user.
Actors	User, Server
Precondition	The user must have an account to block.
Activation	The user blocks the account
MainFlow	-System displays search bar -User inputs name -System displays account -User clicks on account -User blocks account
Exceptional Flow	<INVALID> <CANCEL>
TERMINATION	SUCCESS: <TNOTIFY> FAILURE:<THOME>
POST CONDITION	SUCCESS: <TUSER>

2.1.2. Data Requirements

User Data

Standard information such as personal details are also necessary. All users will be required to create a Username and Password to log in and out of the application.

Skills and Experience Data

Upon Creating an account through Shade Stack, they are required to supply information relating to gaming and requisites to professional level organization's.

2.1.3. Environmental Requirements

The environmental requirements for Shade Stack are as follows:

Operating Systems:

- Windows version 7+
- MacOS 10+

Browser:

- Chrome
- Firefox
- Safari

Programming Language and Libraries/Frameworks:

- React
- Chat Engine
- HTML
- CSS
- JavaScript

2.1.4. Usability Requirements

Learnability:

ShadeStack must have an interface which provides the users with ease of access to using the application. Each page should be easily understood.

Error Management:

The system must be designed to alert the user of any error that occurs.

Performance:

The application must be designed to perform well using its standard libraries.

2.2. Design & Architecture / Implementation

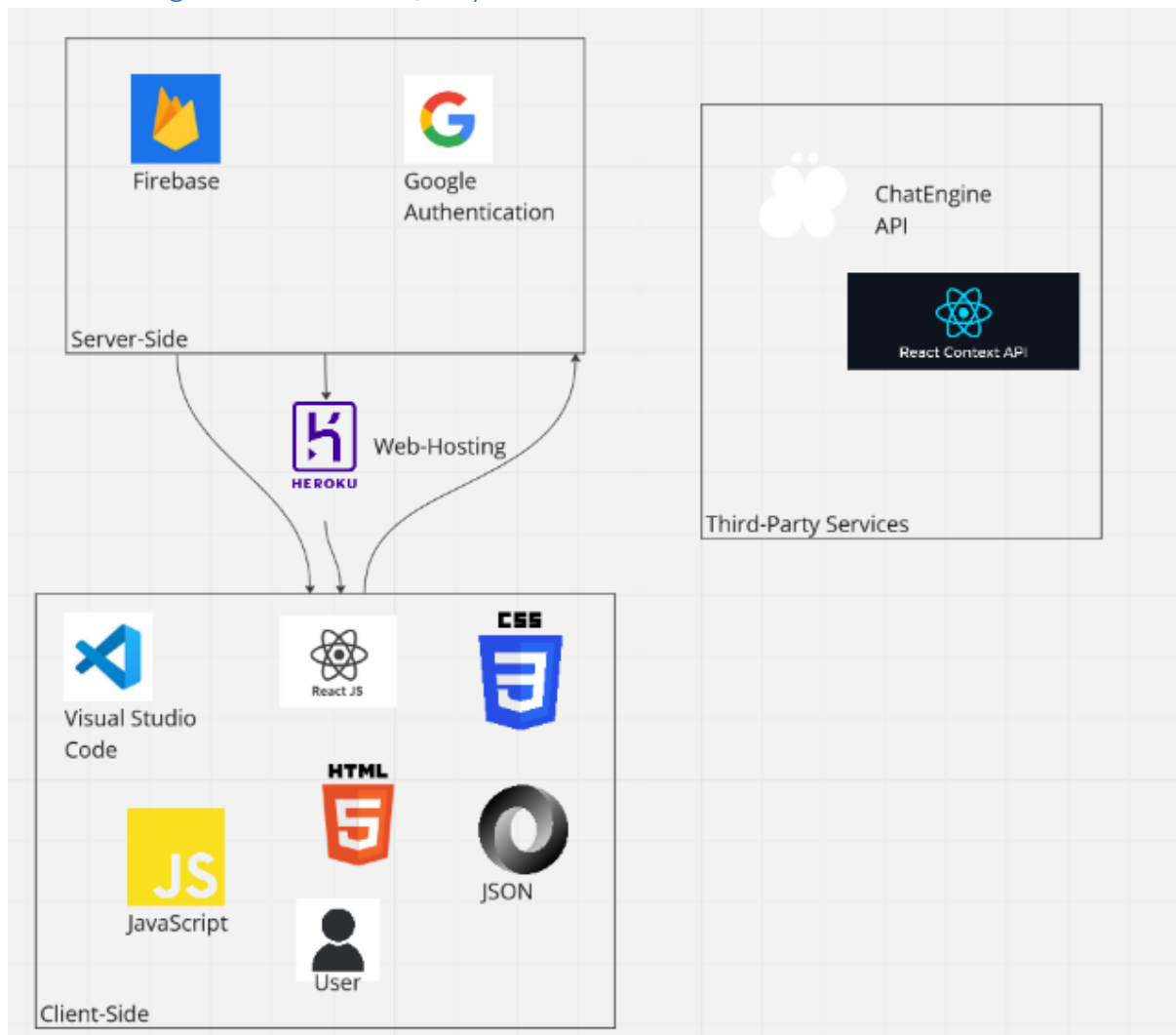


Figure: High Level Architecture of ShadeStack

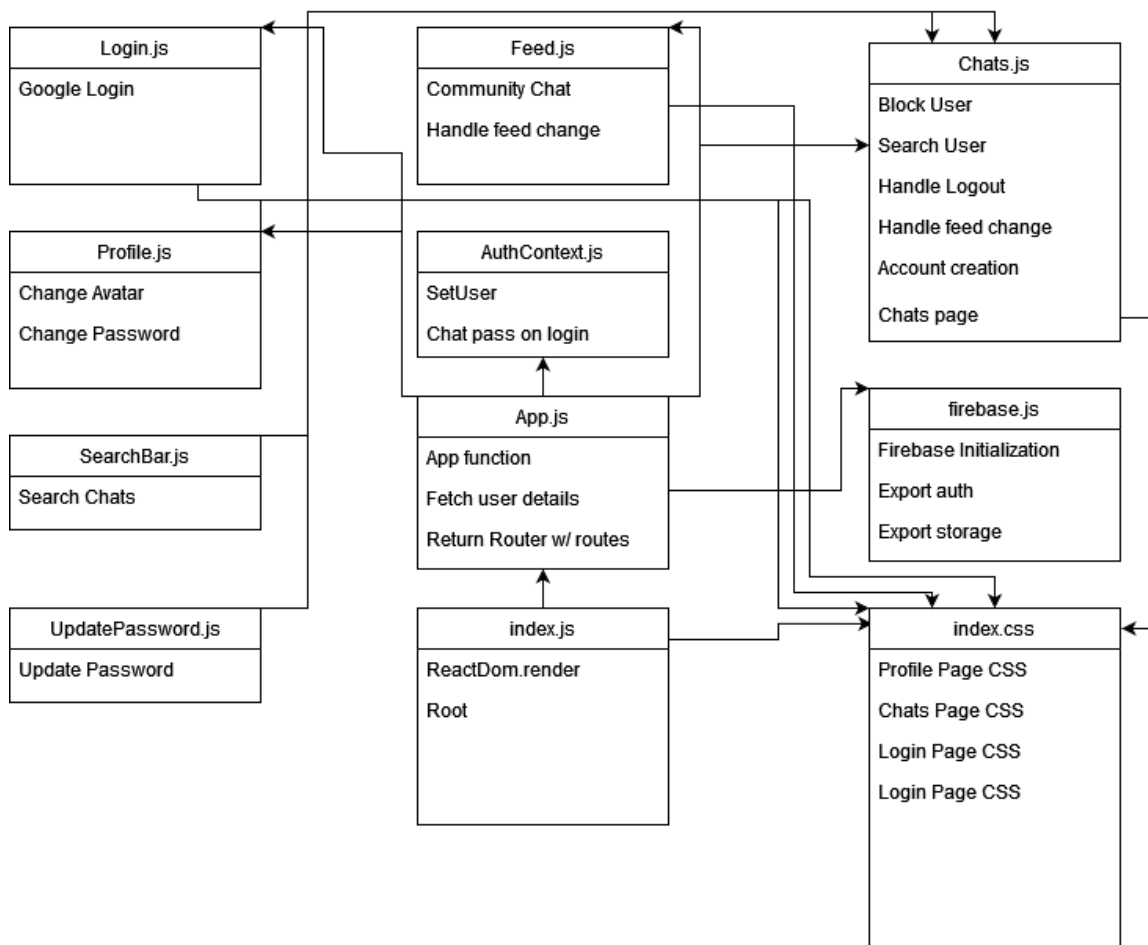



Figure: Class Diagram showing the file structure.

The following are code snippets of the ShadeStack application. Throughout the code will be comments embedded alongside details underneath each snippet describing functionality.

Index.js



```
1 import React from 'react';
2 import ReactDOM from 'react-dom';
3 import './index.css';
4 import App from './components/App';
5
6 ReactDOM.render(
7   <React.StrictMode>
8     <App />
9   </React.StrictMode>,
10  document.getElementById('root')
11 );
12
```

ReactDOM used to render the react app.

App.js

```
1 //Imports for react, browser router, chats.js, and Login.js
2 import React from "react"
3
4 import { BrowserRouter as Router, Switch, Route } from "react-router-dom"
5 import { AuthProvider } from "../contexts/AuthContext"
6
7 import { useEffect, useState } from 'react';
8 import { ChatEngine } from 'react-chat-engine';
9 import axios from 'axios';
10
11 import Chats from "./Chats"
12 import Login from "./Login"
13 import Feed from "./Feed"
14 import SearchBar from "./SearchBar"
15 import Profile from './Profile';
16
17
18 //React Context one big object contains all user data and wraps all other components
19 //AuthProvider handling the application state
20 function App() {
21   const [userName, setUserName] = useState();
22   const [userSecret, setUserSecret] = useState();
23
24   useEffect(() => {
25     // Fetch user details on app load
26     fetch('/api/user')
27       .then(res => res.json())
28       .then(user => {
29         setUserName(user.email);
30         setUserSecret(user.uid);
31       });
32   }, []);
33
34   return (
35     <div style={{ fontFamily: 'Avenir' }}>
36       <Router>
37         <AuthProvider>
38           <Switch>
39             <Route path="/chats" component={Chats}/>
40             <Route path="/feed" component={Feed} />
41             <Route path="/searchbar" component={SearchBar} />
42             <Route path="/profile" component={Profile}/>
43             <Route path="/" component={Login} />
44           </Switch>
45         </AuthProvider>
46       </Router>
47     </div>
48   )
49 }
50
51 export default App;
52
```

React Router, AuthProvider, Switch and Route' s used to define page paths while logged in.

Firestore.js

```
1 //Imports for firebase and auth
2 import firebase from 'firebase/compat/app';
3 import 'firebase/compat/auth';
4 import 'firebase/compat/storage';
5
6 //create initialize app and use auth as a function
7 export const auth = firebase.initializeApp({
8     apiKey: "AIzaSyDr0zY30YfzG6gHwkDgJKbLpE7zISZxN_U",
9     authDomain: "shadestack-552dc.firebaseio.com",
10    projectId: "shadestack-552dc",
11    storageBucket: "shadestack-552dc.appspot.com",
12    messagingSenderId: "326095191226",
13    appId: "1:326095191226:web:1ee10ee668753d305f9683",
14    measurementId: "G-HJ3L04PHTR"
15 }).auth();
16
17
18 const storage = firebase.storage();
19
20 export { storage };
```

Used to setup auth and storage functions for firebase.

AuthContext.js

```

1 //Imports for context, state, and effect, use history, and auth
2 import React, { useContext, useState, useEffect } from 'react';
3 import { useHistory } from 'react-router-dom';
4 import { auth } from '../firebase';
5
6 //Create AuthContext
7 const AuthContext = React.createContext();
8
9 //Function to export entire context, return useContext Passing AuthContext into React useContext
10 export const useAuth = () => useContext(AuthContext);
11
12
13 //Called when user object changes or history changes
14 //Children passes all jsx into Auth Provider
15 //Set Loading object state
16 export const AuthProvider = ({ children }) => {
17   const [loading, setLoading] = useState(true);
18   const [user, setUser] = useState (null);
19   const history = useHistory();
20
21   //Whenever the state of auth changed call user data
22   useEffect(() => {
23     auth.onAuthStateChanged((user) => {
24       setUser(user);
25       setLoading(false);
26       if(user) history.push('/chats');
27     })
28   }, [user, history]);
29
30 //Value user objectproperty
31   const value = {user};
32
33 //If not Loading show the children
34   return (
35     <AuthContext.Provider value={value}>
36       {!loading && children}
37     </AuthContext.Provider>
38   );
39 }

```

Used to render auth user function.

Chats.js Part 1.(Please zoom in to read code)

```

1 //Imports for React, useRef, useState, useEffect, useHistory, ChatEngine, auth, useAuth, and axios
2 import React, { useRef, useState, useEffect } from 'react';
3 import { useHistory } from 'react-router-dom';
4 import { ChatEngine } from 'react-chat-engine';
5 import { auth } from '../firebase';
6 import { ChatEngineWrapper } from 'react-chat-engine';
7
8 import SearchBar from './SearchBar'
9 import { useAuth } from '../contexts/AuthContext';
10 import axios from 'axios';
11
12 //Chats function
13 const Chats = () => {
14
15   //Use history for history
16   const history = useHistory ();
17
18   //User user for useAuth
19   const { user } = useAuth();
20
21   const [loading, setLoading] = useState(true);
22
23   //Log user
24   console.log(user);
25
26   const [chats, setChats] = useState([]);
27   const [filteredChats, setFilteredChats] = useState([]);
28   const [searchValue, setSearchValue] = useState("");
29
30   const [searchQuery, setSearchQuery] = useState("");
31   const [searchResults, setSearchResults] = useState([]);
32
33   //Block Users
34   const [blockedUsers, setBlockedUsers] = useState([]);
35
36   const blockUser = (username) => {
37     // add the username to the blockedUsers array
38     setBlockedUsers([...blockedUsers, username]);
39
40     // block the user in Chat Engine
41     ChatEngine.global.blockUser(username);
42   };
43
44   const isBlocked = (username) => {
45     return blockedUsers.includes(username);
46   };
47
48   const handleSearch2 = async () => {
49     // Call the Chat Engine API to search for users
50     const response = await ChatEngine.global.searchUsers(searchQuery);
51
52     // Update the search results state with the response
53     setSearchResults(response.results);
54   };
55
56   useEffect(() => {
57     // Fetch chats from ChatEngine API
58     const fetchChats = async () => {
59       const response = await axios.get("https://api.chatengine.io/chats", {
60         headers: {
61           "project-id": process.env.REACT_APP_PROJECT_ID,
62           "user-name": user.email,
63           "user-secret": user.uid,
64         },
65       });
66     };
67
68     setChats(response.data);
69     setFilteredChats(response.data);
70   });
71
72   fetchChats();
73 }, []);
74
75 const handleSearch = (value) => {
76   setSearchValue(value);
77   const filtered = chats.filter((chat) =>
78     chat.title.toLowerCase().includes(value.toLowerCase())
79   );
80   setFilteredChats(filtered);
81 };
82
83 //Handle Logout
84 const handleLogout = async () => {
85   await auth.signOut();
86
87   history.push('/');
88 }
89
90 //Handle feed change
91 const routeChange = () =>{
92   let path = 'newPath';
93   history.push('feed');
94 }
95
96 //Get user profile image
97 const getFile = async (url) => {
98   const response = await fetch(url);
99   const data = await response.blob();
100
101   return new File([data], "userPhoto.jpg", { type: 'image/jpg' })
102 }
103
104 //If there is no user then redirect to Login
105
106 useEffect(() => {
107   if(!user) {
108     history.push('/');
109   }
110
111   return;
112 }
113

```

Part 2.
(Please zoom

in to read
code)

```
1
2 //Axios call to get account if already created
3 //If not, create account
4 axios.get('https://api.chatengine.io/users/me' , {
5   headers: {
6     "project-id": process.env.REACT_APP_PROJECT_ID,
7     "user-name": user.email,
8     "user-secret": user.uid,
9   }
10 })
11 .then(() => {
12   setLoading(false);
13 })
14 .catch(() => {
15   let formdata = new FormData();
16   formdata.append('email', user.email);
17   formdata.append('username', user.email);
18   formdata.append('secret', user.uid);
19
20   getFile(user.photoURL)
21   .then((avatar) => {
22     formdata.append('avatar', avatar, avatar.name)
23
24     axios.post('https://api.chatengine.io/users/',
25       formdata,
26       { headers: { "private-key": process.env.REACT_APP_PROJECT_KEY } }
27     )
28     .then(() => setLoading(false))
29     .catch((error) => console.log(error))
30   })
31 })
32 }, [user, history]);
33
34 //If the user is loading return Loading
35 if(!user || loading) return 'Loading ...';
36
37 //Return navbar and chatengine when logged in
38 return (
39   <div className="chats-page">
40     <div className="nav-bar">
41       <div className="logo-tab">
42         ShadeStack
43       </div>
44       <div onClick={routeChange} className="feed-tab">
45         Feed
46       </div>
47       <div onClick={() => history.push('/profile')} className="profilepage-tab">
48         Profile
49       </div>
50       <div onClick={handleLogout} className="logout-tab">
51         Logout
52       </div>
53     </div>
54     <div>
55       <input
56         type="text"
57         value={searchQuery}
58         onChange={(event) => setSearchQuery(event.target.value)}
59       />
60       <button onClick={handleSearch}>Search For User</button>
61     </div>
62     <div>
63       {searchResults.map((result) => (
64         <div key={result.username}>
65           <p>{result.username}</p>
66         </div>
67       ))}
68     </div>
69     <div>
70       <input
71         type="text"
72         value={searchQuery}
73         onChange={(event) => setSearchQuery(event.target.value)}
74       />
75       <button onClick={handleSearch}>Search User To Block</button>
76     </div>
77     <div>
78       {searchResults.map((result) => (
79         <div key={result.username}>
80           <p>{result.username}</p>
81           <button onClick={() => blockUser(result.username)}>Block User</button>
82         </div>
83       ))}
84     </div>
85     <div className="App">
86       <SearchBar onSearch={handleSearch} />
87       <ChatEngine
88         height="calc(100vh - 66px)"
89         projectId={process.env.REACT_APP_PROJECT_ID}
90         userName={user.email}
91         userSecret={user.uid}
92         chats={filteredChats}
93         isBlocked={isBlocked}
94       />
95     </div>
96   </div>
97 );
98 }
99
100 /*
101 <button onClick={() => blockUser('username_to_block')}>
102   Block User
103 </button>
104 */
105
106 export default Chats;
```

Render for Block, Search and Chats main page. Main Comments embedded within code.

Return() brings back navbar, searchbar, and chatengine.

Auth, user variable, history, and accounts all setup within this page alongside chats.

Feed.js

```

1  import React from 'react'
2  import { useHistory } from 'react-router-dom';
3  import { ChatEngine } from 'react-chat-engine';
4  import { auth } from '../firebase';
5  import { useAuth } from '../contexts/AuthContext';
6  import { ChatEngineWrapper, ChatSocket, ChatFeed } from 'react-chat-engine';
7
8  const Feed = () => {
9    //Use history for history
10   const history = useHistory ();
11
12   //Handle feed change
13   const routeChange = () =>{
14     let path = `newPath`;
15     history.push('chats');
16   }
17
18   return (
19     <div className="chats-page">
20       <div className="nav-bar">
21         <div className="logo-tab">
22           ShadeStack
23         </div>
24         <div onClick={routeChange} className="feed-tab">
25           Feed
26         </div>
27       </div>
28       <ChatEngineWrapper data-testid="chat-engine-wrapper">
29         <ChatSocket
30           data-testid="chat-socket"
31           projectID='fc01842d-4c85-476e-b75c-cc5a0fd3df95'
32           chatID='165492'
33           chatAccessKey='1234'
34           senderUsername='Anonymous'
35         />
36         <ChatFeed data-testid="chat-feed" activeChat='123' />
37       </ChatEngineWrapper>
38     </div>
39   )
40 }
41
42 export default Feed;

```

Render for navbar with divs and ChatEngine feed using ChatEngineWrapper and ChatSocket.

Login.js

```

1 //Imports for react, google and facebook buttons, firebase app, auth value and firebase
2 import React from 'react';
3 import { GoogleOutlined, FacebookOutlined } from '@ant-design/icons';
4 import 'firebase/compat/app';
5 import { auth } from '../firebase';
6 import firebase from 'firebase/compat/app';
7
8 //Login function returning divs created with css and buttons
9 const Login = () => {
10   return(
11     <div id="login-page">
12       <div id="login-main">
13         <div className="nav-bar">
14           <div className="logo-tab">
15             Welcome to: ShadeStack
16           </div>
17         </div>
18         <div id="login-card2">
19           <h2>
20             The all purpose application for teaming up!
21           </h2>
22           <h2>
23             Chat in our main community feed
24           </h2>
25           <h2>
26             or
27           </h2>
28           <h2>
29             Create your own room with your team members!
30           </h2>
31         </div>
32         <div id="login-card">
33           <h2>
34             Your way to gaming groups now!
35           </h2>
36           <div className="login-button google"
37             onClick={() => auth.signInWithRedirect(new firebase.auth.GoogleAuthProvider())}
38           >
39             <GoogleOutlined /> Sign In / Up with Google
40           </div>
41           <div>
42             </div>
43         </div>
44       </div>
45     </div>
46   );
47 }
48
49 export default Login;

```

Div setup for logging in. Google button within login card for signInWithRedirect to Google.

Profile.js (Please zoom in to read code)

```
1 import React, { useState } from 'react';
2 import { useHistory } from 'react-router-dom';
3 import { ChatEngine } from 'react-chat-engine';
4 import { useAuth } from '../contexts/AuthContext';
5 import { auth } from '../firebase';
6 import { updateUser } from '../helpers/updateUser';
7
8 const Profile = () => {
9   //Set Loading
10  const [loading, setLoading] = useState(false);
11  //Use user for useAuth
12  const { user } = useAuth();
13  //Use history for useHistory
14  const history = useHistory();
15
16  //HandleLogout await signout button then push to Login
17  const handleLogout = async () => {
18    await auth.signOut();
19    history.push('/');
20  };
21
22  //Push to UpdatePassword upon press
23  const handleChangePassword = () => {
24    history.push('/Update-Password');
25  };
26
27  //HandleAvatarUpdate, set Loading to true, await user chatengine secret, target said image, set Loading to false.
28  const handleUpdateAvatar = async (e) => {
29    setLoading(true);
30    await updateUser(user.uid, e.target.files[0]);
31    setLoading(false);
32  };
33
34  //Return profile page
35  return (
36    <div className="profile-page">
37      <div className="profile-info">
38        <h2>{user.displayName}</h2>
39        <p>{user.email}</p>
40        <div className="profile-actions">
41          <label htmlFor="avatar" className="update-avatar">
42            {loading ? 'Updating...' : 'Update Avatar'}
43          <input
44            id="avatar"
45            type="file"
46            onChange={handleUpdateAvatar}
47            style={{ display: 'none' }}
48          />
49        </label>
50        <button onClick={handleChangePassword}>Change Password</button>
51        <button onClick={handleLogout}>Logout</button>
52      </div>
53    </div>
54    <ChatEngine
55      height="calc(100vh - 66px)"
56      projectID={process.env.REACT_APP_PROJECT_ID}
57      userName={user.email}
58      userSecret={user.uid}
59    />
60  </div>
61  );
62 };
63
64 export default Profile;
```

Main Comments embedded.

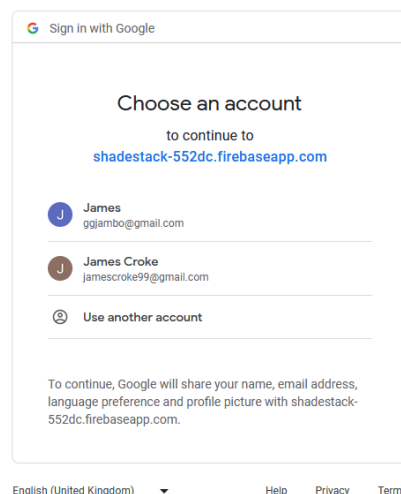
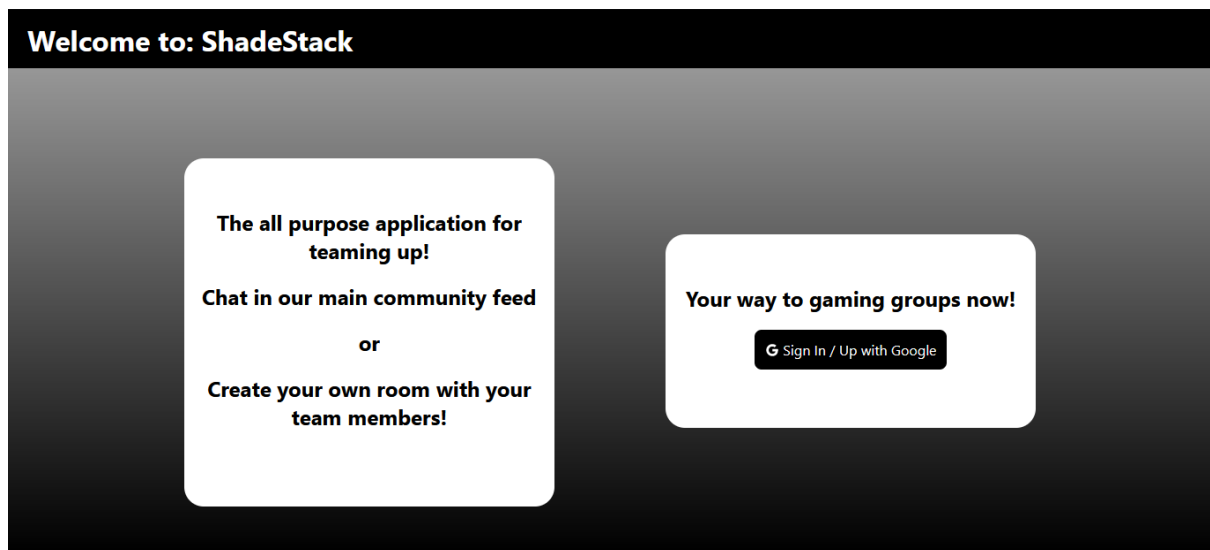
The following Pages such as ErrorBoundary.js, SearchBar.js, UpdatePassword.js, updateUser.js and index.css are all important to the entire build-up of the application but

are not the main classes and functions such as the previously stated and will not be included within the document. These pages can still be viewed within the code repository (<https://github.com/ggjambo/ComputingProject>).

2.3. Graphical User Interface (GUI)

Sign-In Page (Authentication):

- Google

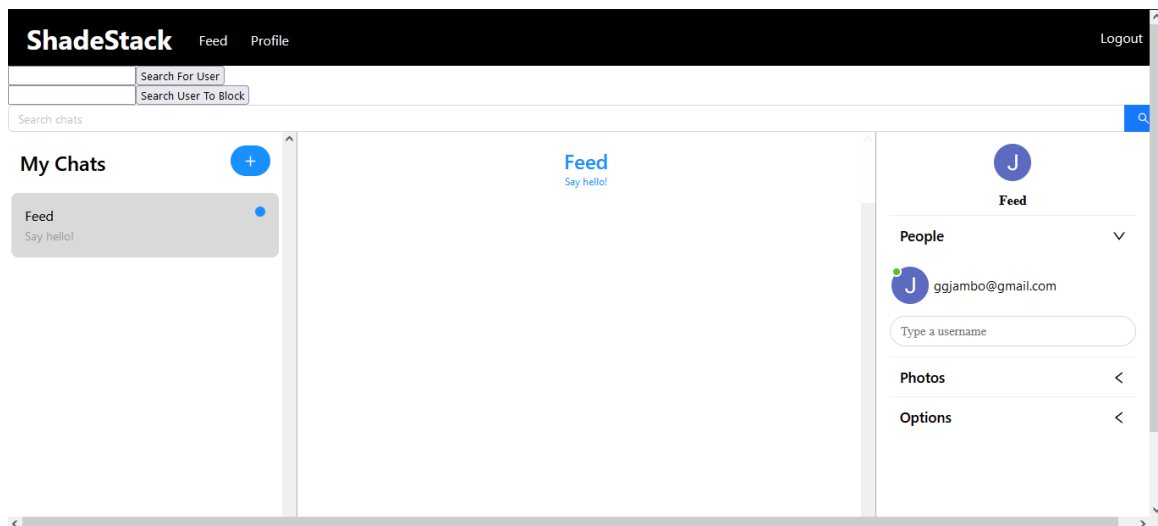


Chats:

- Redirect to feed
- Redirect to profile
- Search for User
- Search for User to Block
- Search For Chats
- Allows for the creation of chat rooms

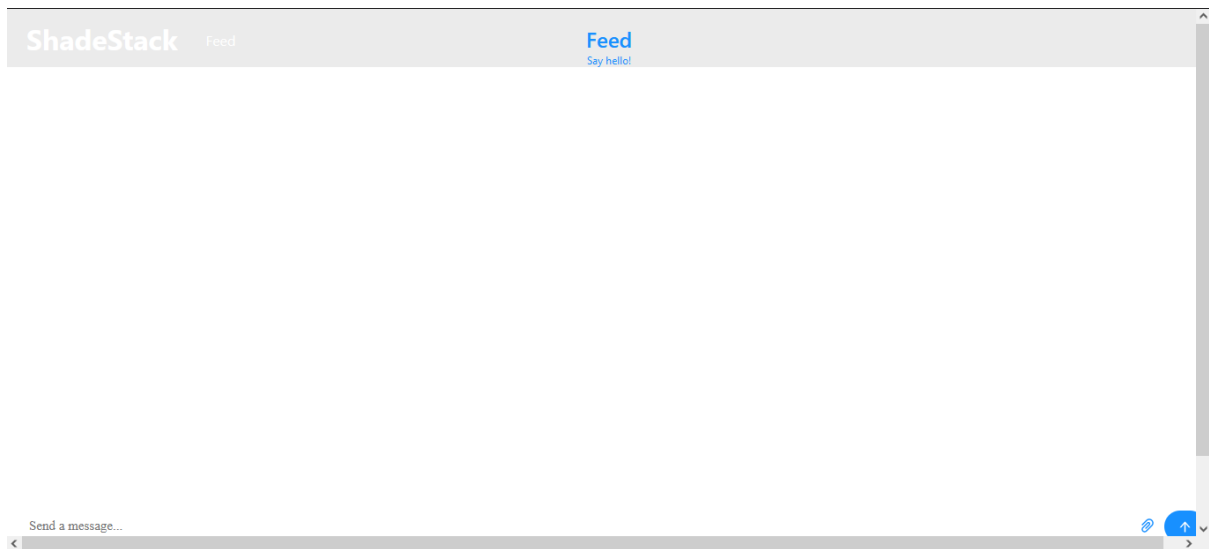
Chat Room:

- Allows for messaging into the room.
- Users can be added / removed
- Attachments added &
- Room deleted



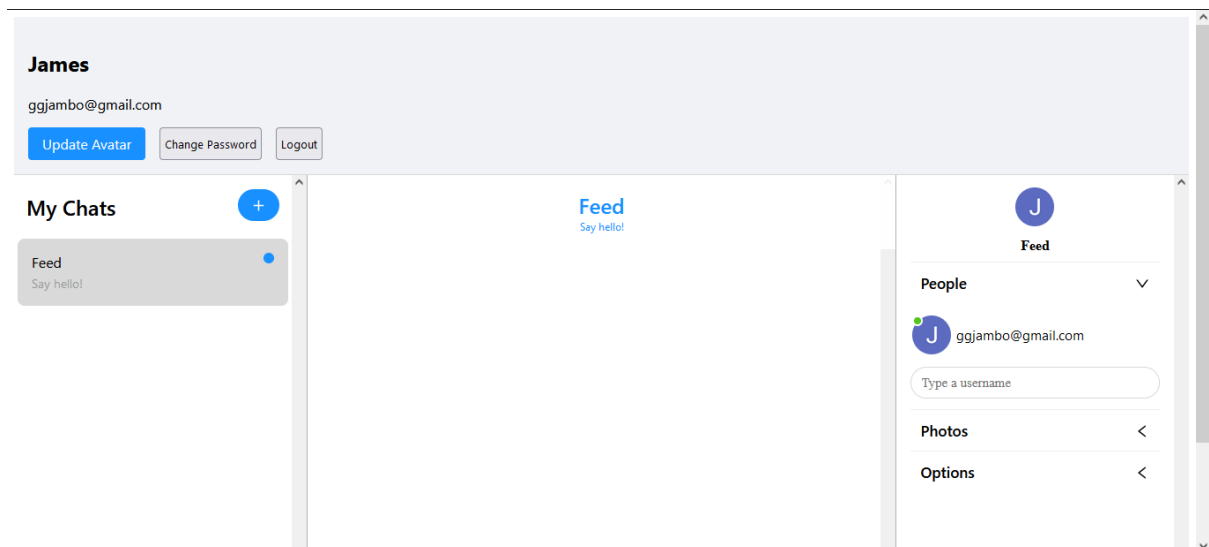
Feed:

- Community feed to anonymously link up with new players



Profile:

- Update Avatar
- Change Password
- Account Bio to be implemented



2.4. Testing

Black-Box Testing:

Throughout the project's development, black-box testing was continuously carried out to confirm the addition of new code and features. This was done by simply navigating the UI of the application to see if it performed correctly.

Unit Testing:

Individual units of source code—sets of one or more computer program modules together with associated control data, usage procedures, and operating procedures—are tested to determine their suitability for use using unit testing.

Throughout the following are tests for each individual page created with its associated tests to render the outcomes of the page.

Throughout the test suites there was an underlying error with Jest itself not being able to recognize the output of the AuthContext class to be function. There was also an error with Jest not being able to recognize the auth.AuthStateChanged to be a function and the React Router API. Others I could not figure out how to fix for testing but still tried.

App.test.js

```

1 import {render, screen} from '@testing-library/react'
2 import userEvent from '@testing-library/user-event'
3 import React from 'react'
4 import '@testing-library/jest-dom'
5 import {App} from './components/app'
6 import {BrowserRouter as Router, MemoryRouter} from "react-router-dom"
7 import {AuthProvider} from "../contexts/AuthContext"
8
9 test('full app rendering/navigating', async () => {
10   const history = createMemoryHistory()
11   render(
12     <Router history={history}>
13       <AuthProvider>
14         <App />
15       </AuthProvider>
16     </Router>,
17   )
18
19   const user = userEvent.setup()
20   // Verify page content for expected route
21   expect(screen.getByText(/you are at login/i)).toBeInTheDocument()
22
23   await user.click(screen.getByText(/chats/i))
24
25   // Check that the content changed to the new page
26   expect(screen.getByText(/you are on the chats page/i)).toBeInTheDocument()
27 })
28
29 test('landing on a bad page', () => {
30   const history = createMemoryHistory()
31   history.push('/some/bad/route')
32   render(
33     <Router history={history}>
34       <AuthProvider>
35         <App />
36       </AuthProvider>
37     </Router>,
38   )
39
40   expect(screen.getByText(/no match/i)).toBeInTheDocument()
41 })

```

firebase.test.js



```
1 // Import Firebase auth and storage
2 import { auth, storage } from './firebase';
3
4 // Test Firebase authentication
5 describe('Firebase auth', () => {
6   it('should initialize Firebase auth', () => {
7     expect(auth).toBeDefined();
8   });
9 });
10
11 // Test Firebase storage
12 describe('Firebase storage', () => {
13   it('should initialize Firebase storage', () => {
14     expect(storage).toBeDefined();
15   });
16 });
17
```

Chats.test.js

```

1 import { render, screen, waitFor } from '@testing-library/react';
2 import { MemoryRouter } from 'react-router-dom';
3 import { AuthContext } from '../contexts/AuthContext';
4 import Chats from './Chats';
5
6 const mockUser = {
7   email: 'ggjambo@gmail.com',
8   uid: 'ce057a44-d96a-4a92-a1a9-d382487e3ac8',
9 };
10
11 jest.mock('../firebase', () => ({
12   auth: {
13     signOut: jest.fn(),
14   },
15 }));
16
17 describe('Chats', () => {
18   it('renders chats page', async () => {
19     render(
20       <MemoryRouter>
21         <AuthContext.Provider value={{ user: mockUser }}>
22           <Chats />
23         </AuthContext.Provider>
24       </MemoryRouter>
25     );
26
27     // Wait for the loading state to disappear
28     await waitFor(() => {
29       expect(screen.queryByText(/Loading/i)).not.toBeInTheDocument();
30     });
31
32     // Expect the navbar to be rendered
33     expect(screen.getByText(/ShadeStack/i)).toBeInTheDocument();
34     expect(screen.getByText(/Feed/i)).toBeInTheDocument();
35     expect(screen.getByText(/Profile/i)).toBeInTheDocument();
36     expect(screen.getByText(/Logout/i)).toBeInTheDocument();
37
38     // Expect the search input and button to be rendered
39     expect(screen.getByLabelText(/search for user/i)).toBeInTheDocument();
40     expect(screen.getByRole('button', { name: /search for user/i })).toBeInTheDocument();
41
42     // Expect the block user search input and button to be rendered
43     expect(screen.getByLabelText(/search user to block/i)).toBeInTheDocument();
44     expect(screen.getByRole('button', { name: /search user to block/i })).toBeInTheDocument();
45   });
46 });
47

```

Feed.test.js



```
1 import React from 'react';
2 import { MemoryRouter } from 'react-router-dom';
3 import { render, screen } from '@testing-library/react';
4 import Feed from './Feed';
5
6 describe('Feed component', () => {
7   it('should render Feed component', () => {
8     render(
9       <MemoryRouter>
10        <Feed />
11      </MemoryRouter>
12    );
13
14    //Expect feed to be rendered
15    expect(screen.getByText('ShadeStack')).toBeInTheDocument();
16    expect(screen.getByText('Feed')).toBeInTheDocument();
17    expect(screen.getByTestId('chat-engine-wrapper')).toBeInTheDocument();
18    expect(screen.getByTestId('chat-socket')).toBeInTheDocument();
19    expect(screen.getByTestId('chat-feed')).toBeInTheDocument();
20  });
21 });
22
```

Login.test.js

```

1 import React from 'react';
2 import { render, screen } from '@testing-library/react';
3 import userEvent from '@testing-library/user-event';
4 import Login from './Login';
5
6 describe('Login component', () => {
7   it('renders without errors', () => {
8     render(<Login />);
9     const loginPageElement = screen.getByTestId('login-page');
10    expect(loginPageElement).toBeInTheDocument();
11  });
12
13  it('has a Google sign-in button', () => {
14    render(<Login />);
15    const googleButton = screen.getByRole('button', { name: /sign in \\/ up with google/i });
16    expect(googleButton).toBeInTheDocument();
17  });
18
19  it('can sign in with Google when the button is clicked', () => {
20    const mockSignInWithRedirect = jest.fn();
21    jest.spyOn(require('firebase/compat/app'), 'auth').mockReturnValue({
22      signInWithRedirect: mockSignInWithRedirect,
23    });
24
25    render(<Login />);
26    const googleButton = screen.getByRole('button', { name: /sign in \\/ up with google/i });
27    userEvent.click(googleButton);
28
29    expect(mockSignInWithRedirect).toHaveBeenCalled();
30  });
31 });
32

```

Profile.test.js

```

1 import React from 'react';
2 import { render, screen, waitFor, fireEvent } from '@testing-library/react';
3 import { MemoryRouter, Router } from 'react-router-dom';
4 import { createMemoryHistory } from 'history';
5 import Profile from './Profile';
6 import { AuthProvider } from './contexts/AuthContext';
7 import { auth } from './firebase';
8
9 jest.mock('./firebase', () => ({
10   auth: {
11     signOut: jest.fn(),
12   },
13 }));
14
15 jest.mock('./helpers/updateUser', () => ({
16   updateUser: jest.fn(),
17 }));
18
19 const mockUser = {
20   uid: 'ce057a44-d96a-4a92-a1a9-d382487e3ac8',
21   displayName: 'Test User',
22   email: 'ggjambo@gmail.com',
23 };
24
25 const setup = () => {
26   return render(
27     <MemoryRouter>
28       <AuthProvider currentUser={mockUser}>
29         <Profile />
30       </AuthProvider>
31     </MemoryRouter>
32   );
33 };
34
35 describe('Profile', () => {
36   it('displays user info and logout button', async () => {
37     setup();
38
39     await waitFor(() => {
40       expect(screen.getByText('Test User')).toBeInTheDocument();
41       expect(screen.getByText('testuser@example.com')).toBeInTheDocument();
42       expect(screen.getByText('Logout')).toBeInTheDocument();
43     });
44   });
45
46   it('logs out the user when Logout button is clicked', async () => {
47     setup();
48
49     fireEvent.click(screen.getByText('Logout'));
50
51     await waitFor(() => {
52       expect(auth.signOut).toHaveBeenCalled();
53     });
54   });
55
56   it('navigates to Update Password page when Change Password button is clicked', async () => {
57     const history = createMemoryHistory();
58     setup();
59
60     fireEvent.click(screen.getByText('Change Password'));
61
62     await waitFor(() => {
63       expect(history.location.pathname).toEqual('/update-password');
64     });
65   });
66
67   it('updates user avatar when Update Avatar button is clicked and file is selected', async () => {
68     const file = new File(['test'], 'test.png', { type: 'image/png' });
69     setup();
70
71     fireEvent.change(screen.getByLabelText('Update Avatar'), { target: { files: [file] } });
72
73     await waitFor(() => {
74       expect(screen.getByText('Updating...')).toBeInTheDocument();
75       expect(updateUser).toHaveBeenCalledWith('1234', file);
76       expect(screen.getByText('Update Avatar')).toBeInTheDocument();
77     });
78   });
79 });
80

```

SearchBar.test.js

```
1 import React from 'react';
2 import { render, screen, fireEvent } from '@testing-library/react';
3 import SearchBar from './SearchBar';
4
5 test('calls onSearch with correct value when enter is pressed', () => {
6   const onSearch = jest.fn();
7   render(<SearchBar onSearch={onSearch} />);
8
9   const input = screen.getByPlaceholderText('Search chats');
10  fireEvent.change(input, { target: { value: 'test' } });
11  fireEvent.keyDown(input, { key: 'Enter', code: 'Enter' });
12
13  expect(onSearch).toHaveBeenCalledWith('test');
14 });
15
16 test('calls onSearch with correct value when search button is clicked', () => {
17   const onSearch = jest.fn();
18   render(<SearchBar onSearch={onSearch} />);
19
20   const input = screen.getByPlaceholderText('Search chats');
21   fireEvent.change(input, { target: { value: 'test' } });
22
23   const button = screen.getByLabelText('search');
24   fireEvent.click(button);
25
26   expect(onSearch).toHaveBeenCalledWith('test');
27 });
28
```

UpdatePassword.test.js (Please zoom in to read code)

```
1 import React from 'react';
2 import { render, fireEvent, act } from '@testing-library/react';
3 import { BrowserRouter as Router } from 'react-router-dom';
4 import { AuthProvider } from '../contexts/AuthContext';
5 import UpdatePassword from './UpdatePassword';
6
7 describe('UpdatePassword', () => {
8   const setup = () => {
9     return render(
10       <Router>
11         <AuthProvider>
12           <UpdatePassword />
13         </AuthProvider>
14       </Router>
15     );
16   };
17
18   it('should render without errors', () => {
19     const { getByText, getByLabelText } = setup();
20
21     expect(getByText('Update Password')).toBeInTheDocument();
22     expect(getByLabelText('Password')).toBeInTheDocument();
23     expect(getByLabelText('Confirm Password')).toBeInTheDocument();
24     expect(getByText('Update Password')).toBeInTheDocument();
25     expect(getByText('Cancel')).toBeInTheDocument();
26   });
27
28   it('should show error message if passwords do not match', async () => {
29     const { getByLabelText, getByText } = setup();
30     const passwordInput = getByLabelText('Password');
31     const confirmPasswordInput = getByLabelText('Confirm Password');
32     const submitButton = getByText('Update Password');
33
34     fireEvent.change(passwordInput, { target: { value: 'password' } });
35     fireEvent.change(confirmPasswordInput, { target: { value: 'notmatching' } });
36
37     await act(async () => {
38       fireEvent.click(submitButton);
39     });
40
41     expect(getByText('Passwords do not match')).toBeInTheDocument();
42   });
43
44   it('should show error message if update fails', async () => {
45     const mockUpdatePassword = jest.fn(() => {
46       return Promise.reject();
47     });
48
49     const { getByLabelText, getByText } = render(
50       <Router>
51         <AuthProvider value={{ currentUser: {}, updatePassword: mockUpdatePassword }}>
52           <UpdatePassword />
53         </AuthProvider>
54       </Router>
55     );
56
57     const passwordInput = getByLabelText('Password');
58     const confirmPasswordInput = getByLabelText('Confirm Password');
59     const submitButton = getByText('Update Password');
60
61     fireEvent.change(passwordInput, { target: { value: 'password' } });
62     fireEvent.change(confirmPasswordInput, { target: { value: 'password' } });
63
64     await act(async () => {
65       fireEvent.click(submitButton);
66     });
67
68     expect(getByText('Failed to update account')).toBeInTheDocument();
69   });
70
71   it('should update password successfully', async () => {
72     const mockUpdatePassword = jest.fn(() => {
73       return Promise.resolve();
74     });
75
76     const { getByLabelText, getByText, history } = render(
77       <Router>
78         <AuthProvider value={{ currentUser: {}, updatePassword: mockUpdatePassword }}>
79           <UpdatePassword />
80         </AuthProvider>
81       </Router>
82     );
83
84     const passwordInput = getByLabelText('Password');
85     const confirmPasswordInput = getByLabelText('Confirm Password');
86     const submitButton = getByText('Update Password');
87
88     fireEvent.change(passwordInput, { target: { value: 'password' } });
89     fireEvent.change(confirmPasswordInput, { target: { value: 'password' } });
90
91     await act(async () => {
92       fireEvent.click(submitButton);
93     });
94
95     expect(mockUpdatePassword).toHaveBeenCalled();
96     expect(history.location.pathname).toBe('/');
97   });
98 });
99
```

AuthContext.test.js

```
1 import React from 'react';
2 import { render } from '@testing-library/react';
3 import { AuthProvider, useAuth } from './AuthContext';
4
5 describe('AuthContext', () => {
6   test('AuthProvider should render children', () => {
7     const { getByTestId } = render(
8       <AuthProvider>
9         <div data-testid="test-child" />
10      </AuthProvider>
11    );
12    expect(getByTestId('test-child')).toBeInTheDocument();
13  });
14
15   test('useAuth should return user object', () => {
16     const TestComponent = () => {
17       const { user } = useAuth();
18       return <div data-testid="test-user">{user}</div>;
19     };
20     const { getByTestId } = render(
21       <AuthProvider>
22         <TestComponent />
23       </AuthProvider>
24     );
25     expect(getByTestId('test-user')).toBeInTheDocument();
26   });
27 });
28
```

updateUser.test.js (Please zoom in to read code)

```
1 import { storage } from '../firebase';
2 import axios from 'axios';
3
4 jest.mock('../firebase', () => ({
5   storage: {
6     ref: jest.fn(),
7   },
8 }));
9
10 jest.mock('axios');
11
12 describe('updateUser', () => {
13   const uid = 'user123';
14   const file = new File(['(-□□)'], 'avatar.png', { type: 'image/png' });
15   const putMock = jest.fn(() => Promise.resolve());
16   const getDownloadURLMock = jest.fn(() => Promise.resolve('https://example.com/avatar.png'));
17
18   beforeEach(() => {
19     jest.clearAllMocks();
20     storage.ref.mockReturnValue({
21       child: jest.fn(() => ({
22         put: putMock,
23         getDownloadURL: getDownloadURLMock,
24       })),
25     });
26   });
27
28   it('uploads the file to Firebase Storage and updates the user avatar in ChatEngine', async () => {
29     axios.patch.mockResolvedValueOnce({ data: {} });
30
31     await updateUser(uid, file);
32
33     expect(storage.ref).toHaveBeenCalledTimes(1);
34     expect(storage.ref).toHaveBeenCalledWith('avatars/user123');
35     expect(putMock).toHaveBeenCalledTimes(1);
36     expect(putMock).toHaveBeenCalledWith(file);
37     expect(getDownloadURLMock).toHaveBeenCalledTimes(1);
38     expect(axios.patch).toHaveBeenCalledTimes(1);
39     expect(axios.patch).toHaveBeenCalledWith(
40       `https://api.chatengine.io/users/${uid}/`,
41       { avatar: 'https://example.com/avatar.png' },
42       {
43         headers: {
44           'Private-Key': process.env.REACT_APP_CHAT_ENGINE_PRIVATE_KEY,
45         },
46       }
47     );
48   });
49
50   it('throws an error if there was a problem uploading the file to Firebase Storage', async () => {
51     putMock.mockRejectedValueOnce(new Error('Upload error'));
52
53     await expect(updateUser(uid, file)).rejects.toThrow('Upload error');
54
55     expect(storage.ref).toHaveBeenCalledTimes(1);
56     expect(storage.ref).toHaveBeenCalledWith('avatars/user123');
57     expect(putMock).toHaveBeenCalledTimes(1);
58     expect(putMock).toHaveBeenCalledWith(file);
59     expect(getDownloadURLMock).not.toHaveBeenCalled();
60     expect(axios.patch).not.toHaveBeenCalled();
61   });
62
63   it('throws an error if there was a problem updating the user avatar in ChatEngine', async () => {
64     putMock.mockResolvedValueOnce();
65     axios.patch.mockRejectedValueOnce(new Error('ChatEngine error'));
66
67     await expect(updateUser(uid, file)).rejects.toThrow('ChatEngine error');
68
69     expect(storage.ref).toHaveBeenCalledTimes(1);
70     expect(storage.ref).toHaveBeenCalledWith('avatars/user123');
71     expect(putMock).toHaveBeenCalledTimes(1);
72     expect(putMock).toHaveBeenCalledWith(file);
73     expect(getDownloadURLMock).toHaveBeenCalledTimes(1);
74     expect(axios.patch).toHaveBeenCalledTimes(1);
75     expect(axios.patch).toHaveBeenCalledWith(
76       `https://api.chatengine.io/users/${uid}/`,
77       { avatar: 'https://example.com/avatar.png' },
78       {
79         headers: {
80           'Private-Key': process.env.REACT_APP_CHAT_ENGINE_PRIVATE_KEY,
81         },
82       }
83     );
84   });
85 });
86
```

The outcomes of the tests are as follows:

console.error

Warning: React.jsx: type is invalid -- expected a string (for built-in components) or a class/function (for composite components) but got: undefined. You likely forgot to export your component from the file it's defined in, or you might have mixed up default and named imports.

Check your code at Profile.test.js:30.

```
28 |   <MemoryRouter>
29 |     <AuthProvider currentUser={mockUser}>
> 30 |       <ErrorBoundary>
    |         ^
31 |         <Profile />
32 |       </ErrorBoundary>
33 |     </AuthProvider>
```

console.error

The above error occurred in the <AuthProvider> component:

- in AuthProvider (at Profile.test.js:29)
- in Router (created by MemoryRouter)
- in MemoryRouter (at Profile.test.js:28)

- Profile › displays user info and logout button

TypeError: _firebase.auth.onAuthStateChanged is not a function

```
21 | //Whenever the state of auth changed call user data
22 | useEffect(() => {
> 23 |     auth.onAuthStateChanged((user) => {
    |     ^
24 |         setUser(user);
25 |         setLoading(false);
26 |         if(user) history.push('/chats');
```

- Profile › logs out the user when Logout button is clicked

TypeError: _firebase.auth.onAuthStateChanged is not a function

```
21 | //Whenever the state of auth changed call user data
22 | useEffect(() => {
> 23 |     auth.onAuthStateChanged((user) => {
    |     ^
24 |         setUser(user);
25 |         setLoading(false);
26 |         if(user) history.push('/chats');
```

- Profile › navigates to Update Password page when Change Password button is clicked

TypeError: _firebase.auth.onAuthStateChanged is not a function

```
21 | //Whenever the state of auth changed call user data
```

```
22 |     useEffect(() => {
> 23 |         auth.onAuthStateChanged((user) => {
    |         ^
24 |             setUser(user);
25 |             setLoading(false);
26 |             if(user) history.push('/chats');
```

- Profile › updates user avatar when Update Avatar button is clicked and file is selected

TypeError: _firebase.auth.onAuthStateChanged is not a function

```
21 |     //Whenever the state of auth changed call user data
22 |     useEffect(() => {
> 23 |         auth.onAuthStateChanged((user) => {
    |         ^
24 |             setUser(user);
25 |             setLoading(false);
26 |             if(user) history.push('/chats');
```

FAIL src/firebase.test.js

- Test suite failed to run

INTERNAL ASSERTION FAILED: Expected a class definition

```
13 |     appId: "1:326095191226:web:1ee10ee668753d305f9683",
14 |     measurementId: "G-HJ3L04PHTR"
> 15 | }).auth();
```

```
| ^
16 |
17 |
18 | const storage = firebase.storage();
```

FAIL src/components/Feed.test.js

- Test suite failed to run

INTERNAL ASSERTION FAILED: Expected a class definition

```
13 |   appId: "1:326095191226:web:1ee10ee668753d305f9683",
14 |   measurementId: "G-HJ3L04PHTR"
> 15 | }).auth();
    | ^
16 |
17 |
18 | const storage = firebase.storage();
```

FAIL src/components/Chats.test.js (78.82 s)

- Chats › renders chats page

TypeError: Cannot read properties of undefined (reading 'Provider')

```
19 |   render(
20 |     <MemoryRouter>
> 21 |       <AuthContext.Provider value={{ user: mockUser }}>
    |           ^
```

```
22 |     <Chats />
23 |   </AuthContext.Provider>
24 | </MemoryRouter>
```

at Object.<anonymous> (src/components/Chats.test.js:21:22)

FAIL src/App.test.js

- Test suite failed to run

INTERNAL ASSERTION FAILED: Expected a class definition

```
13 |   appId: "1:326095191226:web:1ee10ee668753d305f9683",
14 |   measurementId: "G-HJ3L04PHTR"
> 15 | }).auth();
    |   ^
16 |
17 |
18 | const storage = firebase.storage();
```

Test Suites: 9 failed, 1 passed, 10 total

Tests: 8 failed, 2 passed, 10 total

Snapshots: 0 total

Time: 108.393 s

Ran all test suites.

2.5. Evaluation

The system was evaluated by determining whether or not all of the requirements established at the outset of the project were met, as well as whether or not the project's end goals were met. It was also evaluated for speed and performance.

Given circumstances on how this is met, the application was not 100% complete. Some issues still remain within the project such as the searchbar's, and Profile updating not working. Some functionality is still missing as well and yet to be implemented such as account bios.

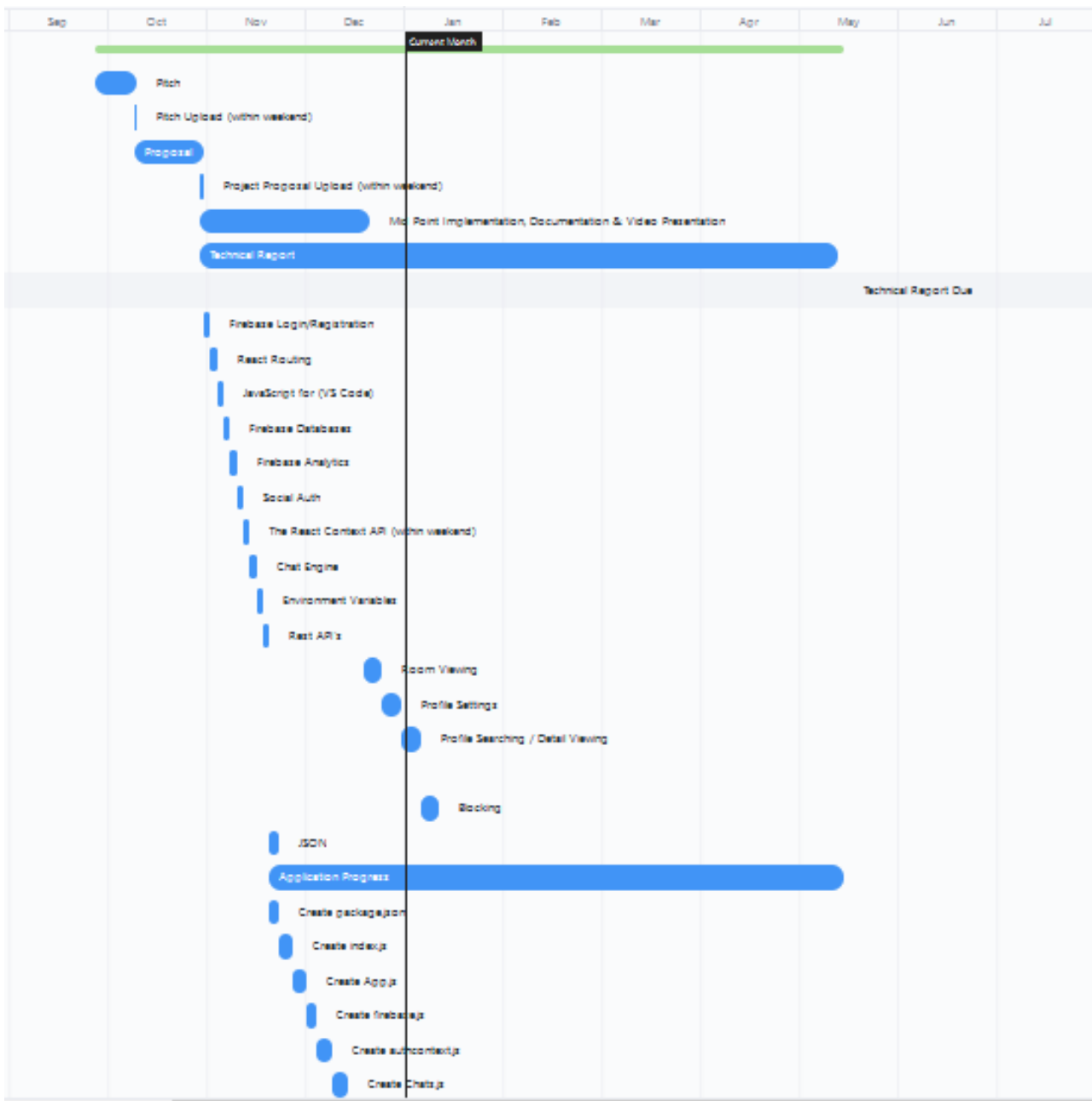
As these are the only issues to remain within the project, it is roughly around 80-85% complete with some bits of code needing to be fleshed out. As for speed and performance logging in, setting up rooms, adding users, sending messages, sending attachments, deleting rooms, and sending messages within the community feed are all up to scratch.

The application works as intended and is mostly finished.

3.0 Conclusions

ShadeStack is a very adverse communication platform in terms of other platforms out there from highly known brands to smaller companies. It is very simple to use, all you need to do is click the google button, sign in with your account and you're in. From communicating within the community feed to then creating a room with found members it is a very quick way to find teammates for games and sorting out details. On other platforms you may be only able to contact a person one by one and them not even see the message, as well as that some chat rooms available online only feature the communication of games, not to link players up. ShadeStack is an all in one platform for enabling players to quickly and comfortably communicate with other users to exchange details for meetups. A disadvantage of ShadeStack is that it is not complete. It would have been nice to search for players to then find their details and or the search functions work to find different already created rooms to meetup quicker. Another disadvantage of ShadeStack is that the community feed is at risk of spam and users not directly communicating about their gaming details. Other than these disadvantages ShadeStack would be have been a very complete application for randomly searching chats/users, to communicating with them.

4.0 Project Plan



5.0 Further Development or Research

With additional time and resources, the project would definitely be 100% complete for the issues that are still remaining with the application. Accounts would then be searchable / blockable, rooms searchable, and bios available for accounts.

6.0 References

7.0 Appendices

7.1 Reflective Journals

Supervision & Reflection

Student Name	James Croke
Student Number	17480714
Course	Digital Business Transformation
Supervisor	Adriana Chis

Month: October

What?

Reflect on what has happened in your project this month:

Throughout the initial weeks of this first semester I began to brainstorm and research into interests of mine and what I could do to translate one I could pick into a project.

I ended up finalizing that idea to creating a web application for the short- and long-term networking of gamers. This being the most suitable option was decided due to my knowledge and understanding of web development.

With what I have found, the technologies such as React Routing, Social auth, Firebase, the React Context API, Chat Engine, Environment Variables and Rest APIs gives a solid foundation for development of the application.

Gaming has also been a big passion of mind especially the pro side of it and really motivated me towards creating some network for gamers given the lack of resources out there for quick finding of players.

So What?

Consider what that meant for your project progress. What were your successes? What challenges still remain?

Successes:

From the technologies that I have found so far has led to a significant head start into development of the project.

There will be no need into researching specific technologies for certain requirements of the project as they are already there.

Requirements gathering is pretty much completed also and will just require research from the listed technologies in order to implement.

Challenges remaining:

- Identify methods of coding using the technologies found
- Research more into potential functional requirements that can be added to the project from the existing technologies
- Research if additional technologies need to be added in order to complete the project
- Complete Project Proposal
- Start Coding

Now What?

What can you do to address outstanding challenges?

- Use trustworthy resources in order to learn
- Become more familiar with the technologies
- Figure out if there technologies that work cohesively with what is currently found with the technologies
- Slowly complete project proposal with research that has been found
- Become familiar with the first part of the application for example the login and registration and start there

Student Signature

James Croke

Supervision & Reflection

Student Name	James Croke
Student Number	17480714
Course	Digital Business Transformation
Supervisor	Adriana Chis

Month: November

What?

Reflect on what has happened in your project this month:

During the course of this month has been very busy as a lot of other projects have had deadlines.

It has been very difficult to put time into the computing project as these other projects have taken priority for the moment due to the level of difficulty for learning and development.

In the Computing Project the project proposal and technical document have been worked on.

The project proposal itself completed and the technical document currently in progress.

As for the coding of the project, that is yet to be initialized.

So What?

Consider what that meant for your project progress. What were your successes? What challenges still remain?

Successes:

The project proposal is completed.

The technical document is nearly finished.

Other projects are nearly finished to begin coding of the computing project.

Challenges remaining:

- Begin the computing project
- Finish the Technical document

Now What?

What can you do to address outstanding challenges?

-Finish projects as soon as possible

Student Signature

James Croke

Supervision & Reflection

Student Name	James Croke
Student Number	17480714
Course	Digital Business Transformation
Supervisor	Adriana Chis

Month: December

What?

Reflect on what has happened in your project this month:

During the course of this month has also been very busy as a lot of other projects deadlines have been coming up.

It has been very difficult to put time into the computing project as these other projects have taken priority but progress has been made with finishing the prototype and completing the presentation.

In the Computing Project the project technical document has also been worked on and nearly finished.

So What?

Consider what that meant for your project progress. What were your successes? What challenges still remain?

Successes:

Prototype finished.

The technical document is nearly finished.

Challenges remaining:

-Finish researching room viewing for application.

-Implement room viewing and other functions.

- Finish the Technical document

Now What?

What can you do to address outstanding challenges?

-Finish projects as soon as possible to continue on with research.

Student Signature

James Croke

Supervision & Reflection

Student Name	James Croke
Student Number	17480714
Course	Digital Business Transformation
Supervisor	Adriana Chis

Month: January

What?

Reflect on what has happened in your project this month:

As the month has led on to the second semester it was time to plan again which projects needed to be prioritized in order to be as productive as possible.

Due to their not being too much strenuous work left with the computing project it has left a bit of a comfortable workflow for the next while with researching and implementation.

It would take awhile to figure out how to implement the room viewing but it will be done.

The same for account searching/editing.

So What?

Consider what that meant for your project progress. What were your successes? What challenges still remain?

Successes:

Prototype finished.

Calm workflow.

The technical document is nearly finished.

Challenges remaining:

-Finish researching room viewing for application.

-Finish researching account searching/editing.

-Implement room viewing and other functions.

- Finish the Technical document

Now What?

What can you do to address outstanding challenges?

-Research as planning has been completed.

Student Signature

James Croke

Supervision & Reflection

Student Name	James Croke
Student Number	17480714
Course	Digital Business Transformation
Supervisor	Adriana Chis

Month: February

What?

Reflect on what has happened in your project this month:

As this month has led on finding how to implement room viewing has been quite difficult as there is not a lot of coverage online.

That being said research into testing has also been difficult as there is a lot of different code from what has already been done in order to be perform this task for unit, bottom-up, and component testing. It will take awhile to learn this new style.

Hopefully research into the functionality will not be too difficult before implementing code for testing.

So What?

Consider what that meant for your project progress. What were your successes? What challenges still remain?

Successes:

Some research has been completed.

Challenges remaining:

- Implementation for room viewing needs to be completed
- Implementation for testing of the application needs to be completed
- Finish the Technical document

Now What?

What can you do to address outstanding challenges?

- Continue researching with the time I have before other projects need to be started.

-Start implementing testing.

Student Signature

James Croke

Supervision & Reflection

Student Name	James Croke
Student Number	17480714
Course	Digital Business Transformation
Supervisor	Adriana Chis

Month: March

What?

Reflect on what has happened in your project this month:

Throughout this month I have found more information on all of the final functionality for the application.

The code for testing has additionally been looked into.

So What?

Consider what that meant for your project progress. What were your successes? What challenges still remain?

Successes:

Prototype finished.

Calm workflow

The technical document is nearly finished.

Challenges remaining:

-Implement room viewing for application.

-Implement account searching/editing.

-Implement room viewing and other functions.

- Finish the Technical document

Now What?

What can you do to address outstanding challenges?

-Continue Coding.

Student Signature

James Croke

Supervision & Reflection

Student Name	James Croke
Student Number	17480714
Course	Digital Business Transformation
Supervisor	Adriana Chis

Month: April

What?

Reflect on what has happened in your project this month:

Throughout this month I have implemented the final functionality for the application as best as I could.

The code for testing has also been written to the best of my ability.

So What?

Consider what that meant for your project progress. What were your successes? What challenges still remain?

Successes:

Prototype finished.

Application Finished.

Calm workflow.

The technical document is nearly finished.

Challenges remaining:

- Finish the Technical document

Now What?

What can you do to address outstanding challenges?

-Continue finalizing the technical document.

Student Signature	James Croke

7.2 Project Proposal

National College of Ireland

Project Proposal

ShadeStack

31st Oct 2022

Digital Business Transformation

Academic Year i.e., 2022/2023

James Croke

X17480714

X17480714@student.ncirl.ie

Objectives

ShadeStack is a web application currently in development for browsers, using multiple additional technologies to allow shrouded or unknown players within the gaming community to search for other players of their skill level on platforms such as Xbox, PlayStation, and pc.

In whatever game they are choosing to play, the users will be able to create a room with their own specific parameters or join another room to team up with players and create their own temporary posse or permanent clan.

Once a player has joined a room, they can communicate with other users of that room to figure out how they are going to connect.

This room can also be used as a 3rd party for communication as some games may have cross-play, but their independent platforms does not support the use of text communication between each other.

Rooms can also be created for the sole purpose of gamers to socialize and talk if they wish about a game.

The application is more aimed towards the professional finesse rather than casual as the target market would be players of high skill or rank within their game to form chosen teams for Esports. This encourages the development of professional gaming and the market itself then, as players will generally compete within tournaments on platforms such as GameBattles to win cash prizes, or slowly increase their skill for their own benefit.

Main Objectives of ShadeStack are as follows:

- Easy to navigate UI
- Retrieve and store relevant user information (e.g. games on list, name, age, gender, platform)
- Creation of rooms w/ adding and removing users
- Search for users
- Personal Messaging
- Real Time Chatting/Messaging

Background

Throughout my years of getting older, gaming has been a huge impact towards my balance of a healthy lifestyle with work and hobbies. It has enabled me to procrastinate from where I am [usually] based to instantly being able to jump into some work and repeating the cycle after some burnout.

Within my 17 years of gaming, I have been majorly involved within the social aspect of casual and professional playing, with communicating randomly to persons in online matches to seeking a more objective oriented version to communicating. This can contain looking for specified players with certain skill sets, to looking for specified groups with certain skill sets, generally known as clans.

These clans range from an entire plethora of AAA titles to smaller independent developer games. With the development of platforms such as Xbox, PlayStation, Steam and others like Nintendo, Nvidia, and Google Stadia, it has allowed the progression of communication to find players for groups better but not full as an entirety.

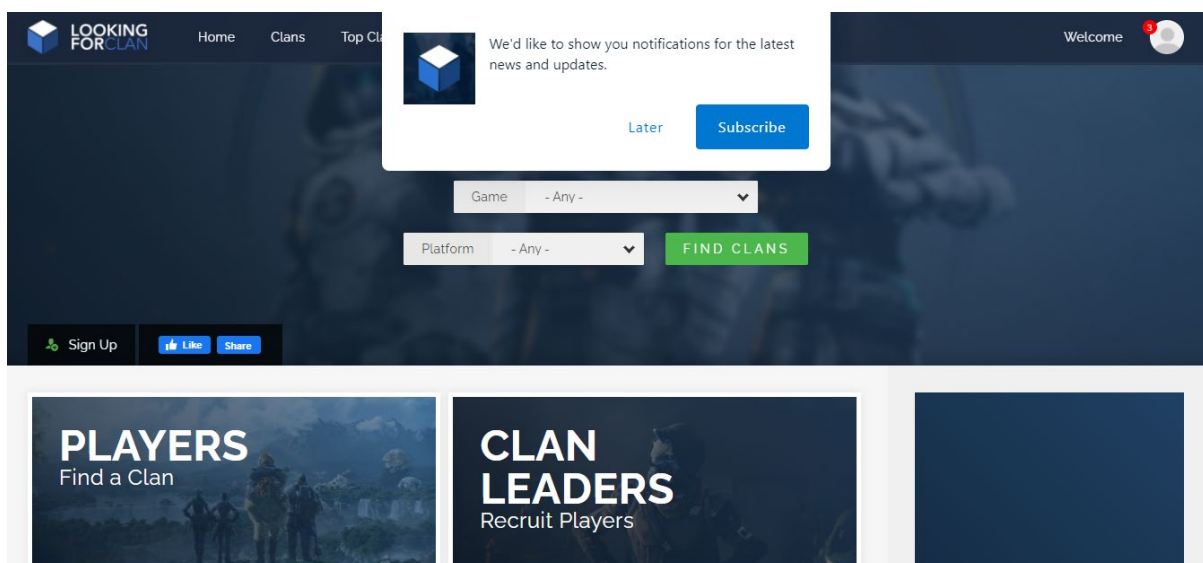
Generally, with these platforms it is not as simple as just going to a brand forum or dedicated communication area to find players, it is a lot more complicated with layers upon layers of verification to then initialize a post for a high possibility of no one to join due to the extra complications.

With ShadeStack the aim of development is to have enough layers of verification there but make the application easily accessible to login and instantly chat within a thread to find a player of choice or even go through extra steps to investigate player skill, games, and connection stability through profiles. Platforms are all included within the user details.

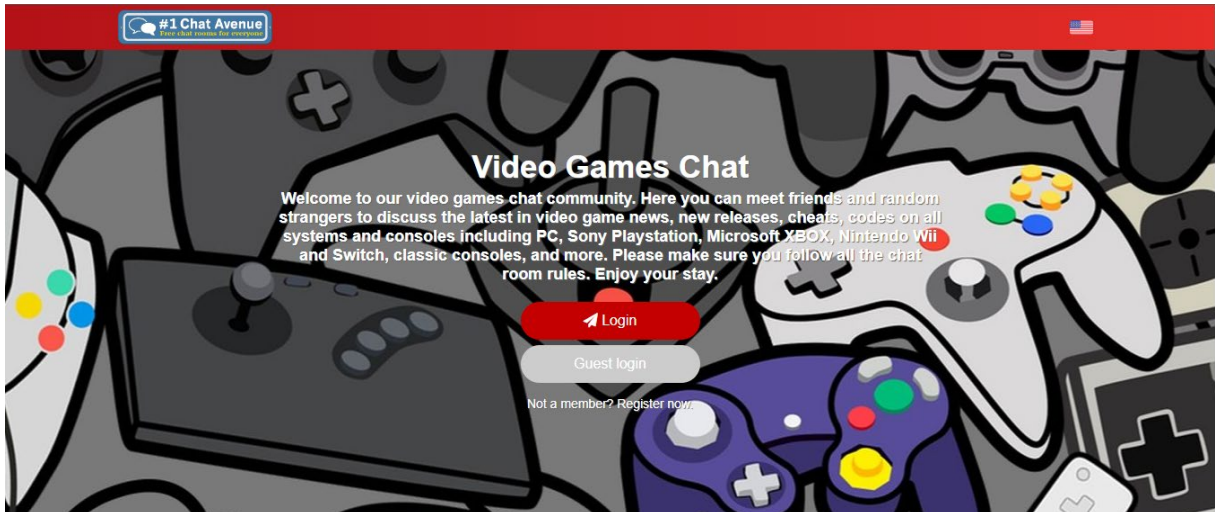
State of the Art

The only other applications out there that performs similarly are called lookingforclan.com, chat-avenue.com/videogames/ and cmxchat.com/gaming-chat/.

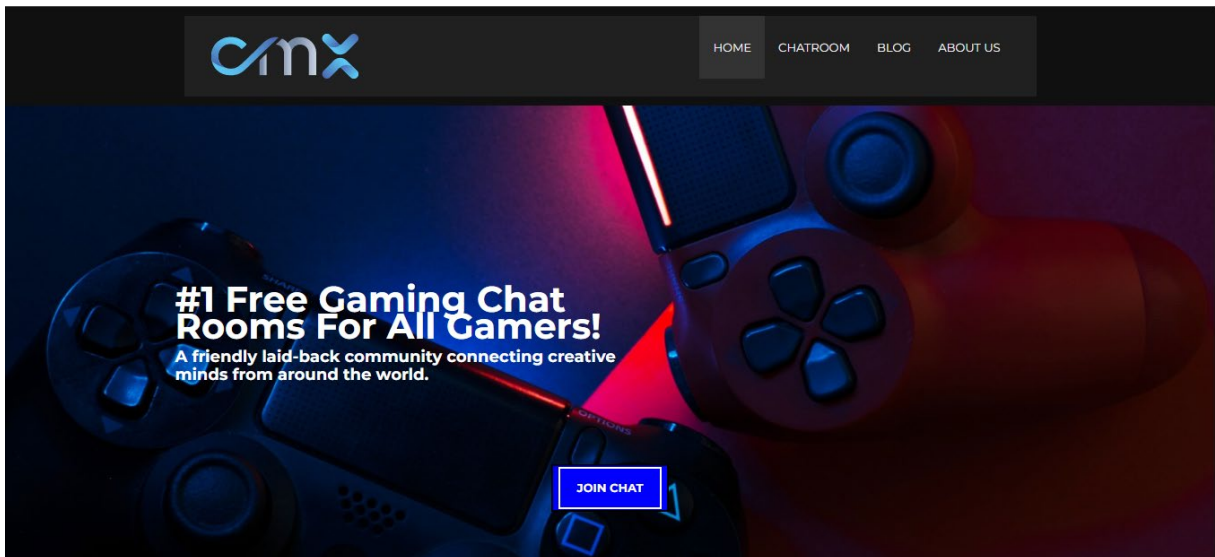
Lookingforclan.com seeks to create clans between players but does not have chat rooms to quickly and readily find players looking to join as a team to compete together.



chat-avenue.com/videogames/ seeks to create chat rooms for gamers but only to discuss the latest in video game news, new releases and cheat codes on various systems.



cmxchat.com/gaming-chat/ offers the same as the previous chat-avenue.com/videogames/ with nothing for connecting gamers to play.



Technical Approach

Developed through an IDE' such as (VS Code) Shadestack will be created using React Routing, Social Auth, Firebase, the React Context Api, Chat Engine, Environment Variables and Rest APIs to create.

Identification of user requirements have been made through personal recommendation and knowledge throughout the years of gaming but, will also be gathered from persons after consent forms signed and options been chosen regarding the format of ethics.

Functional Requirements:

- A1 Users can create an account for the application (ShadeStack)

Users can login to the application with the created account.

Users can fill up their profiles with standard general information within gaming such as Name, Age, Gender, Games, and Ranks within those games.

Users can create rooms:

8.0 Users can add other users a created room

Users can remove users from a created room

Users can message in rooms or personal message

Users can go back to previous messages of a room or personal messages to review information

Users can search for players

A2 Users can look at the details an account

A3

A4 Users can block certain information being displayed

Project milestones, activities, and tasks will be calculated and documented using an iterative approach throughout the coming future, a rigid approach will be taken through the projects development and examined to stay readily close to the plan as said within tab number 7.0.

Technical Details

The technologies stated within the previous heading (technical approach) are to be used and are as follows.

VS Code

Integrated development environments developed by Google and Microsoft used to edit source code that can be used alongside a variety of programming languages, including Java, JavaScript, Go, Node.js, Python, C++ and Dart. VS being built on the Electron framework, used to develop Node.js applications for web running on the Blink layout engine.

React Routing

React (also known as React.js or ReactJS) is a free and open-source front-end JavaScript library for building user interfaces based on UI components. React Router is a powerful routing library built on top of React that helps you add new screens and flows to your application incredibly quickly, all while keeping the URL in sync with what's being displayed on the page. React is only concerned with state management and rendering that state to the DOM, so creating React applications usually requires the use of additional libraries for routing, as well as certain client-side functionality.

Social Auth

Login authentication through 3rd Party applications.

Firestore

A platform developed by Google for creating mobile and web applications. Originally an independent company founded in 2011. Now in the year of 2021 it has been 9 years since Google acquired the platform and is now one of their flagship offerings for app development providing a whole plethora of services for app and web development i.e., Firestore for storing information which it is widely used for.

The React Context Api

The React Context API is a way for a React app to effectively produce global variables that can be passed around. This is the alternative to "prop drilling" or moving props from grandparent to child to parent, and so on.

Chat Engine

Chat Engine is an API providing a REST API, and NPM components to help with chat UI.

Environment Variables

EVs let you store globally scoped values to the environment your code is running in, making them available throughout the codebase.

Rest APIs

A REST API (also known as RESTful API) is an application programming interface (API) that conforms to the constraints of REST architectural style and allows for interaction with RESTful web services.

HTML

The HyperText Markup Language is the standard markup language for documents designed to be displayed in a web browser. It can be assisted by technologies such as Cascading Style Sheets (CSS) for decorating a page and scripting languages to run commands.

CSS

Cascading Style Sheets (CSS) is a style sheet language used for describing the presentation of a document written in a markup language such as HTML.

JavaScript

Often abbreviated as JS, is a programming language of high-level, often just-in-time compiled and multi-paradigm. It has the ability of dynamic typing, prototype-based object-orientation and first-class functions.

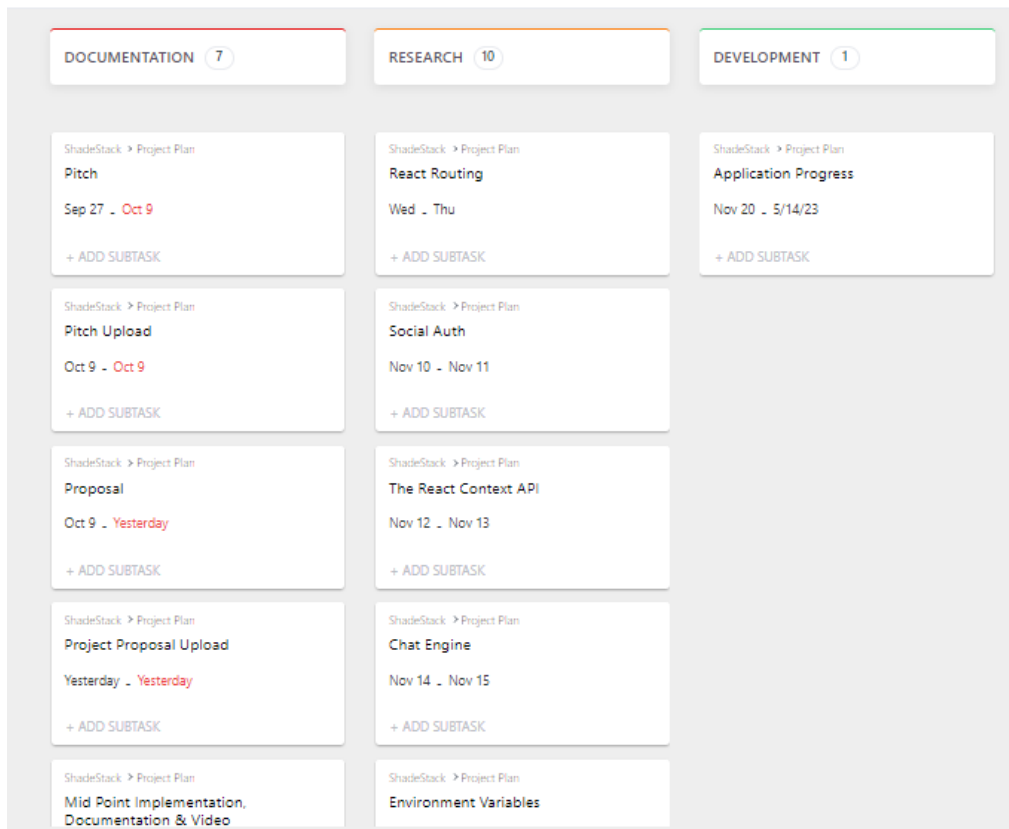
Alongside HTML and CSS, JavaScript is one of the core technologies of the Worldwide WEb. Over 97% of websites using it client-side for web page behaviour, often including third-party libraries for additional functionality.

Special Resources Required

The use of Google and Facebook API's will be attempted within the login / verification of registering for a new account.

Project Plan

Below details the task board and Gantt Chart:



DOCUMENTATION 7

+ ADD SUBTASK

ShadeStack > Project Plan
Mid Point Implementation, Documentation & Video Presentation
 Yesterday - Dec 20
 + ADD SUBTASK

ShadeStack > Project Plan
Technical Report
 Yesterday - 5/12/23
 + ADD SUBTASK

ShadeStack > Project Plan
Technical Report Due
 5/14/23
 + ADD SUBTASK

RESEARCH 10

ShadeStack > Project Plan
Rest API's
 Nov 18 - Nov 19
 + ADD SUBTASK

ShadeStack > Project Plan
Firebase Databases
 Sun - Nov 7
 + ADD SUBTASK

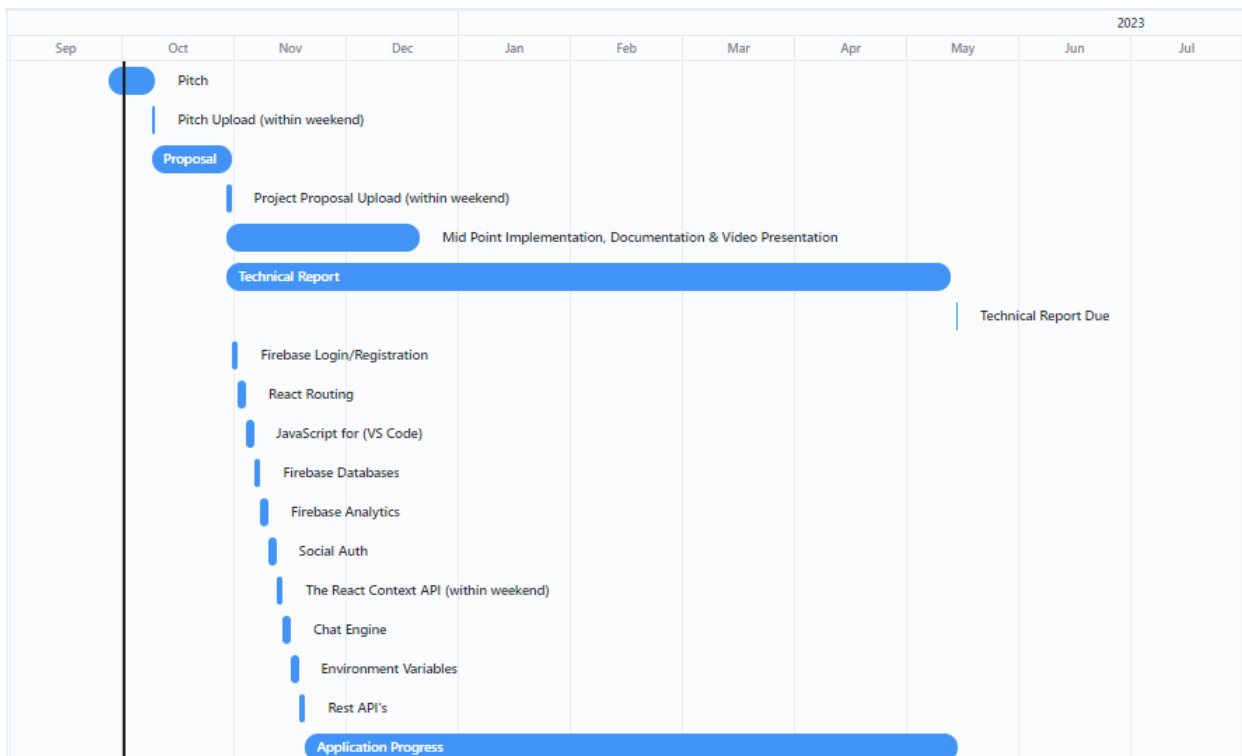
ShadeStack > Project Plan
Firebase Login/Registration
 Today - Tomorrow
 + ADD SUBTASK

ShadeStack > Project Plan
Firebase Analytics
 Nov 8 - Nov 9
 + ADD SUBTASK

ShadeStack > Project Plan
JavaScript for (VS Code)

DEVELOPMENT 1

ShadeStack > Project Plan
Application Progress
 Nov 20 - 5/14/23
 + ADD SUBTASK



Testing

Components / Unit Testing, Bottom-Up Integration Testing and System Testing Tests:

User Tests:

Test Application Boots up

Test Application has Network Connectivity

A1 Login/Register:

Test Displays 'User/Password' Text Boxes

Test can Accept Usernames/Passwords including after multiple invalid attempts

Test Validity of Account can be checked - Test 'Username' and 'Password' values can be verified on Firebase

Test Logging In of Account can be done – Check if account is passed to next screen

Test Invalidity of Account can be checked

Test Error Display Box can be presented after invalidity occurs

Test reverting to previous step can be performed after invalidity occurs

Test to Display 'Try Again' Text Box after Invalid Validation of Details

A2 (Occurrence after previous tests) Profiles:

Test Displays 'User/[Age]/Gender/Games/Ranks' Text Boxes

Test can Read Displays 'User/[Age]/Gender/Games/Ranks' values from the Database

Test the ability to edit 'User/[Age]/Gender/Games/Ranks' Text Boxes

Test 'User/[Age]/Gender/Games/Ranks' Text Box values update to Firebase

Test Invalidity of updating 'User/[Age]/Gender/Games/Ranks' Text Boxes

Test Error Display Box can be presented after invalidity occurs

Test reverting to previous step can be performed after invalidity occurs

Test to Display 'Error' Text Boxes if information cannot be read from Firebase

A3 Rooms:

Test creation of rooms

Test Joining Room

Test showing error upon failed room join

Test shows new room

Test shows existing rooms

Test adding users to room

Test showing error upon non-existent user

Test removing users to room

Test showing error upon non-existent user

Test input into Chat

Test 'Display is Typing'

Test shows the text on the right side

Test to Display 'Try Again' Text Box after Invalid text has been inputted due to profanity or hacking attempts (e.g. SQL Injection)..

Test text is saved to Firebase

Test showing error upon no name created for room

A4 Personal Messaging:

Test search for user's

Test input into Chat

Test 'Display is typing'

Test shows the text on the right side

Test to Display 'Try Again' Text Box after Invalid text has been inputted due to profanity or hacking attempts (e.g. SQL Injection)..

Test text is saved to Firebase

Test showing error upon non existent