Ollscoil Teicneolaíochta an Atlantaigh

Atlantic Technological University

House Swiper

Project Engineering

Year 4

Malek Geshash

Bachelor of Engineering (Honours) in Software and Electronic Engineering

Atlantic Technological University

2022/2023

Declaration

This project is presented in partial fulfilment of the requirements for the degree of Bachelor of Engineering (Honours) in Software and Electronic Engineering at Galway-Mayo Institute of Technology.

This project is my own work, except where otherwise accredited. Where the work of others has been used or incorporated during this project, this is acknowledged and referenced.

Malek Geshash

Acknowledgements

In this section, I wish to acknowledge all my lecturers in GMIT/ATU for what they have taught me over 4 years, which allowed me to create this project, I would also like to thank the project organisers Paul Lennon, Niall O'Keeffe, Michelle Lynch, and my supervisor Brian O'Shea

Table of Contents

1 Summary

For my final year project, I wanted to develop an application that could help solve important, relevant real-world problems. As a college student I have experienced the effects of the housing crisis in Ireland, I believe that an application like House Swiper is a necessity, House Swiper is an application inspired by the dating app Tinder [1], where users sign up and swipe on other users to find their match. This application could help alleviate the housing crisis by providing a platform for people to find housing options quickly and easily, and help people match with housing options that meet their preferred requirements like location price etc.

It would also help reduce the time and effort required by people to find suitable housing. Instead of spending hours searching through websites such as daft.ie and rent.ie, contacting landlords, and arranging viewings. users could simply swipe through potential housing options on the app and connect with landlords directly.

This technical report documents the development process of House Swiper. I developed the application using React JS, Next JS, Node JS, Express JS, MongoDB Atlas, Nginx and Amazon Web Services. I used Microsoft one note to document my weekly progress, Jira for project planning. The application includes features such as user authentication, personal accounts, user swiping, user matching, messaging, filtering, and profile editing. The report provides an overview of the project. It explains the technologies used, the features implemented, and the project's future potential.

2 Poster

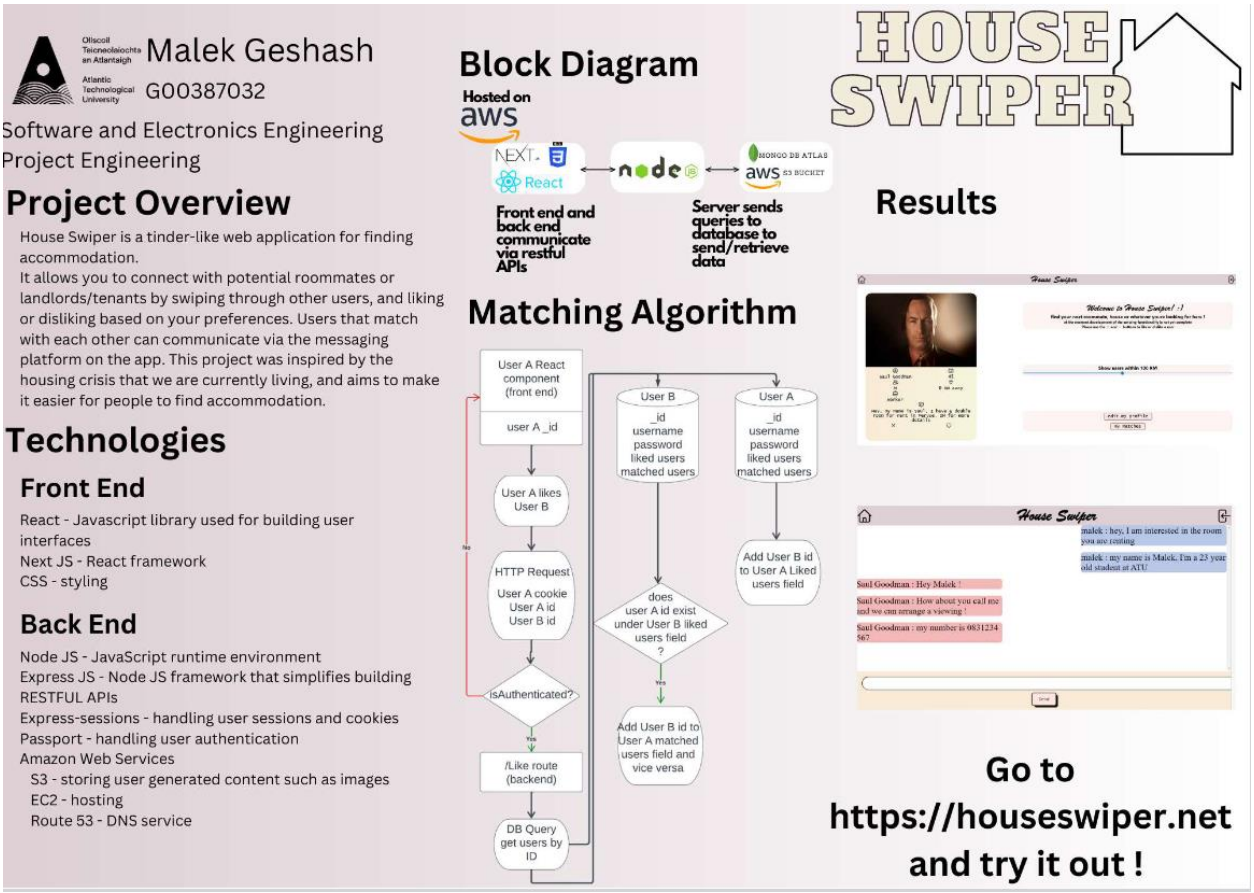

Figure 1

3 Introduction

House Swiper is a web application developed to assist with finding accommodation, inspired by the famous dating app Tinder [1], and by how popular and trendy swiping apps are in this day. House Swiper offers many similar features. Users can register an account with all their personal details relevant to finding accommodation, and swipe on potential accommodation options, the algorithm provides a platform where landlords and people who are looking for a place to rent can match with each other, based on their preferences and swiping patterns. The target user base for this application is college students and young working professionals, who struggle with finding accommodation.

To use House Swiper, a user can register an account and upload relevant details such as name, age, profession, profile picture, and what are they looking for on the app. Then they will be taken to the main page where they can swipe on other users. Users can filter their options based on preferences such as location.

House Swiper offers a messaging platform for users that match with each other, to enable them to connect, discuss details and arrange viewings.

Users can also update their personal details on their profile.

The project uses several technologies, including React JS, Next JS, Node JS, Express JS, Passport JS, MongoDB Atlas, AWS S3, AWS EC2, AWS Route 53 and Nginx.

React and Next are used for the frontend development of the application, while Node and Express are used for the backend development. Passport is used for user authentication. MongoDB is used as the primary database to store and manage user data. AWS S3 is used to store user-generated content, such as images and videos, enabling users to create and manage their personal accounts AWS EC2 is used for Hosting, AWS Route 53 for DNS and Nginx for reverse proxy. git for version control with GitHub as a cloud repository.

These technologies work together to create a user-friendly interface that simplifies the process of finding accommodation. The project aims to provide a solution for users, eliminating the need to navigate multiple websites and platforms to find suitable accommodation options.

4 Development Process

The development of this web application followed an agile software development methodology [2] which consisted of several phases, including planning, design, implementation, version control, weekly stand-up meetings and feedback sessions.

5 Project Architecture

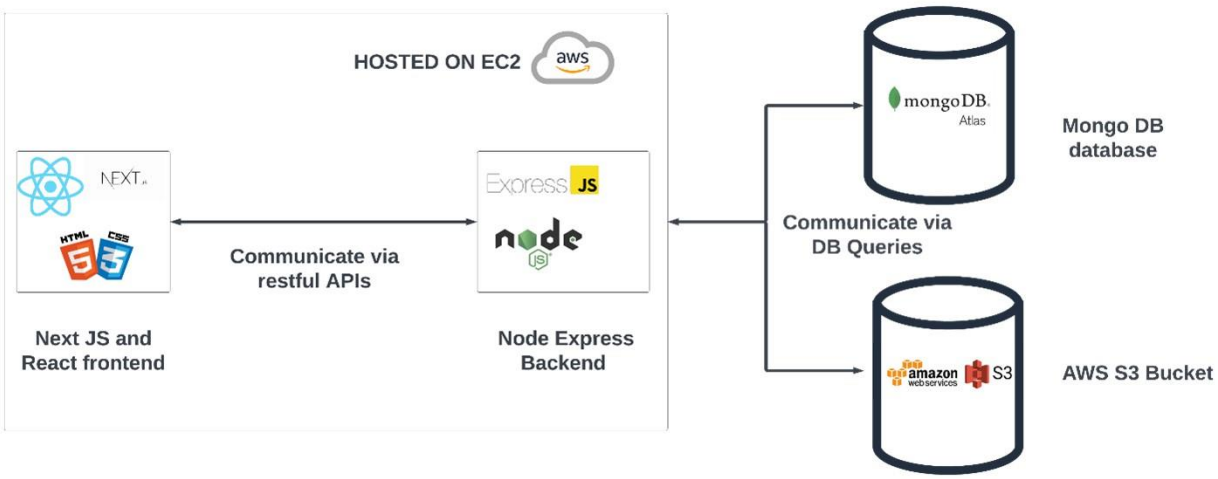

Figure 2

6 Project Planning

I have used Jira [3] and Microsoft OneNote for project planning, this project is planned with an agile approach in mind, which involves breaking down the project into smaller more manageable sprints, each sprint lasting 2 weeks, and each sprint consists of even smaller tasks called stories. I would assign a number of stories in each sprint, depending on the expected amount of work, and then adjust accordingly in the following sprint by looking at the burndown chart to see how much of progress I have made during the sprint, for example, If I couldn't finish all my stories in a sprint, that means I have over assigned stories and should adjust the amount of work expected in the following sprint.

Here is an example of a sprint with stories in progress.

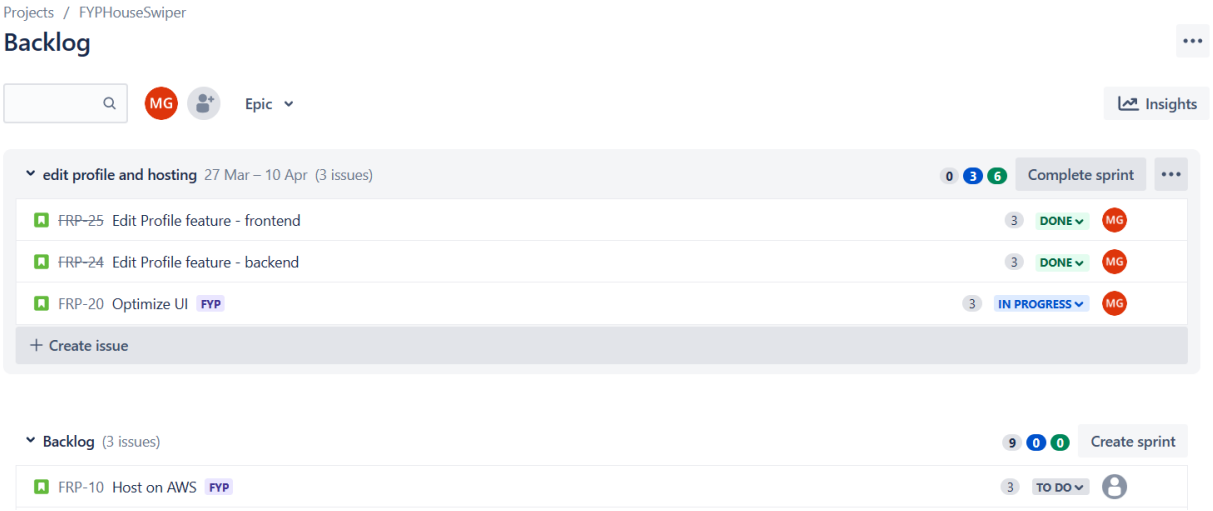

Figure 3

Story points are an estimate of the overall effort required to fully implement a feature.

In the following figure is a screenshot of a sprint burnup chart where I was only able to complete 9 out of 15 story points, which means I over aimed in that sprint and had to adjust in the following one.

Figure 4

And here is the burnup chart of a sprint where I was able to complete all of my tasks one day before the end of the sprint, notice how there's only 9 story points here compared to 15 in the previous chart, which shows that I was being more realistic here.

Figure 5

7 Front End Development

In this section I will talk about the technologies used and brief explanation of them to cover the development of the front end for this web application.

7.1 React JS

React [4] is an open-source JavaScript Library (not a framework!) created by Facebook. It is used for developing user interfaces for web applications.

I chose React for several reasons.

1. Reusability: React allows you to create reusable components which makes the code more efficient and reusable.
2. Efficiency: React uses a virtual DOM (Document Object Model) which provides an efficient way of updating the UI. Instead of updating the entire DOM every time a change is made, it only updates the components that need to be updated. This makes React very fast and efficient.
3. Large Community: React is one of the most popular JavaScript libraries and has a large and active community, and documentation is plentiful.

7.2 Next JS

Next [5] is a framework built on top of React, which allows you to build React applications and provides many features such as, Dynamic routes, API routes and built in CSS modules.

7.3 CSS

CSS stands for Cascading Style Sheets, is a language used for describing how a HTML document are displayed on a web page, I'm using a Flexible Box Layout, also known as Flexbox, to style my pages, Flexbox allows me to create flexible and responsive layouts.

8 Back End Development

In this section I will talk about the technologies used give a brief explanation of them to cover the development of the back end for this web application.

8.1 Node JS

Node JS is a JavaScript runtime environment that allows you to build server-side applications using JavaScript, it made sense to use Node JS since that would allow me to write JavaScript on both the Client and Server side.

8.2 Express JS

Express [6] is an open-source framework for Node JS, it allows you to build REST APIs [7]for your web application using Node JS to handle HTTP requests.

Express provides a routing app which allows you to define routes for different requests based on HTTP method [8]and incoming URL, this makes it possible to create different API endpoints for your web application.

8.3 Passport JS and Express Sessions

Passport JS [9] is an authentication middleware for Node-Express. It allows you to handle user authentication, Passport JS provides several authentication strategies, in this project I'm using a local strategy, which authenticates users by username and password.

I'm using Express sessions to create browser cookies and start a user session; passport can store authentication data in sessions which makes it possible to persist user authentication across requests.

8.4 Mongo DB and Mongoose

Mongo DB is a NoSQL database; it stores data in documents in a JSON like format called BSON (Binary JSON). MongoDB allows for flexible and dynamic schemas for defining data structure.

Mongoose is a library for MongoDB and Node JS, it provides a high-level abstraction over the MongoDB driver in Node JS, making it easier to develop in.

9 Cloud Development

9.1 Amazon Web Services – S3

AWS S3 [10] which stands for Simple Storage Device. Is a cloud-based storage service provided by Amazon Web Services, It can be used as storage for objects such as documents, Images or videos, I'm using it to store user content such as video or images, by uploading the image to S3 and giving it a randomly generated ID, and generating a signed URL for each image, that URL is then saved in the Mongo DB database, and is used on the front end whenever a user is being displayed.

9.2 Amazon Web Services – EC2

AWS EC2 [11] stands for Elastic Cloud Compute. Provides a server in the cloud, which allows me to host my application in the cloud and replaces the need for an actual physical server. Also gives me complete control over the server, which allows me to configure security groups and network settings as I need. I am running the server on Amazon Linux OS, which is better optimized for EC2 than a Linux distribution like Ubuntu.

9.3 Amazon Web Services – Route 53

Route 53 [12] is a Domain Name Service (DNS) web service used for domain name registration and management, I have bought a domain name for my application, www.houseswiper.net using Route 53 and registered the domain name to my application using this service, with Route 53, I was able to create a DNS record to point my domain name to my EC2 instances IP address, which is running my application.

9.4 Mongo DB Atlas

Mongo DB Atlas is the cloud version of Mongo DB which is being used in this project.

9.5 Nginx reverse proxy and SSL

Nginx [13] is a web server that can act as a reverse proxy server [14] , which means it can forward requests from a client to a backend server, I'm using it to listen for HTTPS requests to houseswiper.net on port 443 and forward those requests over a secure connection to port 3000 which exposes my front end.

Nginx supports SSL encryption, by obtaining a SSL cert [15] for my domain name I was able to achieve this.

10 House Swiper Algorithms

In this section I will explain the most important features of House Swiper and how they work

10.1 Matching

Liking and Matching users is probably the main feature of this app, and how the users connect with each other, in this section I will explain in detail how this feature works.

When the user signs in and goes to the index page, a React useEffect hook is executed, which fetches all the relevant profiles to be shown to the signed in user, the profiles are mapped to a custom Card component, these components allow you to like or dislike a profile. To avoid confusion, let us assume that the user who signed in is called user A, and that user A wants to match with User B, when User A likes User B, that calls the handleClickRight arrow function which makes a HTTP PATCH request to the backend API route /matching/likeProfile with the username of User B in the body of the request.

```
useEffect(() => {
  async function fetchData() {
    try {
      const response = await axios.get(
        `http://${process.env.SERVER_URI}:5000/profiles/`,
        { withCredentials: true },
        { headers: { "Content-Type": "application/json" } }
      );
      const data = await response.data;
      console.log(data);
      setCards(
        data.map((person, index) => {
          const distance = getDistanceFromLatLonInKm(latitude, longitude, person.location.lat, person.location.long)
          return (
            <Card
              key={index}
              name={person.username ? person.username : "Name missing"}
              description={
                person.description
                  ? person.description
                  : "description missing"
              }
              age={person.age ? person.age : "age missing"}
              email={person.email ? person.email : "email missing"}
              gender={person.gender ? person.gender : "gender missing"}
              occupation={
                person.occupation ? person.occupation : "occupation missing"
              }
              distance={Math.ceil(distance)}
              image={person.image ? person.image : "/room7.jpg"}
              handleClickRight={async () => {
```

Figure 6

```
handleClickRight={async () => {
  const res = await axios.patch(
    `http://${process.env.SERVER_URI}:5000/matching/likeProfile`,
    { username: person.username },
    { withCredentials: true },
    { headers: { "Content-Type": "application/json" } }
  );
  if (res.data != "no match") {
    console.log(res);
    swal("its a match ! you matched with " + res.data);
  }
  setCardIterator((prevState) => prevState + 1);
}}
handleClickLeft={() => {
  setCardIterator((prevState) => prevState + 1);
}}
></Card>
```

Figure 7

Now onto the Backend and the likeProfile route, first I have a check if the request is authenticated to protect the route and ensure the user is signed in, then I search for User A and User B in the database.

I add User Bs ID to the “likedUsers” field of User A, then I check for a match, I can do that by checking if User B has already liked User B. To check for that I search for User A’s ID in User B’s “likedUsers” field, if it exists, that means both users liked each other and it’s a match. So, I add User A’s ID to User B’s “matchedUsers” field, and vice versa. If that condition is false, it just ends the request, but next time User B likes User A, it will be a match because User A already liked User B.

```
router.patch("/likeProfile",async (req,res)=>{
    if (req.isAuthenticated()){
            if(req.body.username){
                likedProfile = await Profile.findOne({username: new RegExp('^'+req.body.username+'$', "i")});
                console.log(likedProfile)
                // find current user by id then add liked user to current  users liked users list
                const profile = await Profile.findByIdAndUpdate(req.session.passport.user,      MALEK GESHASH - STUDENT, 3 months
                    { "$addToSet": { "likedUsers": likedProfile._id } },
                    { "new": true, "upsert": true },
                );
                // if liked user has already previously liked current user its a match and add both users to each others matched u
                if(likedProfile.likedUsers.includes(profile._id)){
                    await Profile.findByIdAndUpdate(req.session.passport.user,
                        { "$addToSet": { "matchedUsers": likedProfile._id } },
                        { "new": true, "upsert": true },
                    );
                    await Profile.findByIdAndUpdate(likedProfile._id,
                        { "$addToSet": { "matchedUsers": profile._id } },
                        { "new": true, "upsert": true },
                    );
                    res.send(req.body.username)
                }
                else{
                    res.send("no match")
                }
            }
    }
    else{
        res.send("not authorized")
    }
})
```

Figure 8

(In case User A Clicks left on User B it discards that profile and shows the next one.)

I have created a UML diagram to help visualise the matching algorithm.

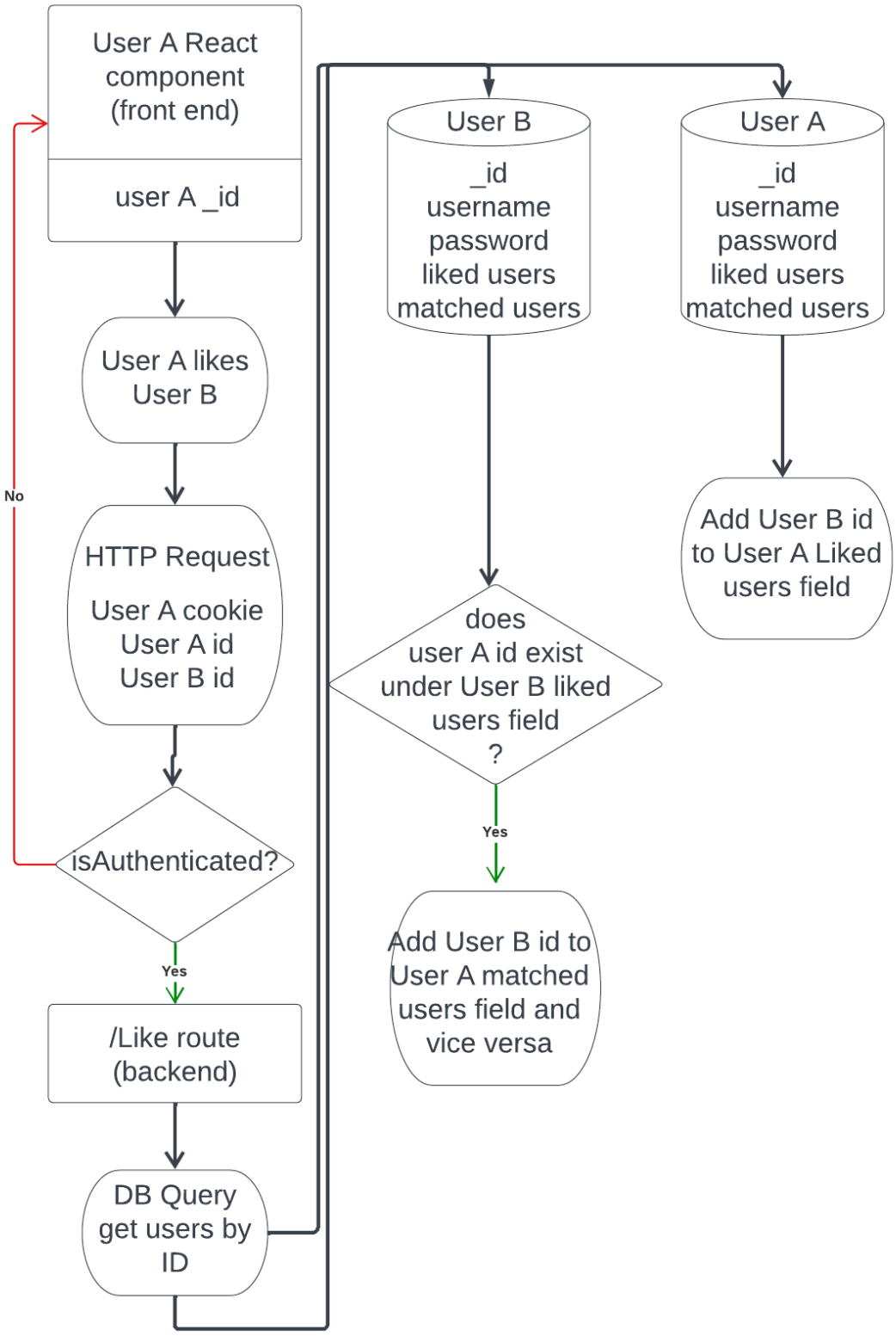
User A React component (front end)
user A _id
User A likes User B
HTTP Request
User A cookie
User A id
User B id
isAuthenticated?
No
Yes
/Like route (backend)
DB Query get users by ID
User B
_id
username
password
liked users
matched users
User A
_id
username
password
liked users
matched users
Add User B id to User A Liked users field
does user A id exist under User B liked users field ?
Yes
Add User B id to User A matched users field and vice versa

Figure 9

Final Result in the matches page:

Figure 10

10.2 Filtering

One of the most important features of this apps is filtering, what would be the point of showing houses hundreds of kilometres away? In this section I will explain the filtering by location functionality.

On the front-end side, I am using Material UIs custom slider component in my main page for the users to set the value of the location filter.

The default value is set to 100KM, which means all profiles shown will be within 100 KM radius of the user's current location, when that slider is changed, the getLocationValue function is called.

```
<div style={{ backgroundColor: "#fdf3f1", borderRadius: 30 }}>
  <h2 className={styles.moretext}>Show users within {filter} KM</h2>
  <div>
    <Slider
      defaultValue={100}
      aria-label="Small"
      max={200}
      min={10}
      valueLabelDisplay="auto"
      onChange={getLocationValue}
    />
  </div>
</div>
```

Figure 11

This function will make an API call to the backend to set the value of the location filter for the user.

```
router.post("/setLocationFilter",async (req,res,next)=>{
    if (req.isAuthenticated()){
        console.log(req.body.radius)
        await Profile.findOneAndUpdate({ _id: [req.session.passport.user] },{locationFilter:req.body.radius})
        res.sendStatus(200)
```

Figure 12

if that API call is successful, it will change the value of the variable filter on the frontend and toggle a state variable called reload.

```
const getLocationValue = async (e, value) => {
  const response = await axios.post(
    `http://${process.env.SERVER_URI}:5000/profiles/setLocationFilter`,
    { radius: value },
    { withCredentials: true },
    { headers: { "Content-Type": "application/json" } }
  );
  const data = response.data;
  console.log(data);
  setFilter(value)
  setReload(prev => !prev)
};
```

Figure 13

Reload is in the dependencies array of the useEffect in my main page, which means that a state change of that variable will cause a re render and the use Effect will be called again, and a new set of users will be fetched.

```
  }
  fetchData();
}, [reload]);
```

Figure 14

Now in the backend we have a look at the /profiles/ route which fetches the profiles to be shown with the new location filter

in the route we define a variable maxRadius which is equal to location filter value we have set

then we call the function getDistanceFromLatLonInKM and assign the value to a variable called distance. If distance is greater than maxRadius that means the profile is outside of the radius and gets filtered out, we do this for all profiles to filter out all profiles that are outside our desired radius.

```
const maxRadius = user.locationFilter
const distance = getDistanceFromLatLonInKm(user.location.lat,user.location.long,profile.location.lat,profile.location.long)
if(maxRadius && distance){
    if(distance>maxRadius){
        getAllProfiles = getAllProfiles.filter(filteredProfiles => filteredProfiles._id != profile._id );
    }
}
```

Figure 15

Now we look at the getDistanceFromLatLonInKM function, this function uses the latitude and longitude of 2 coordinates to calculate the distance between them in Kilometres.

```
function getDistanceFromLatLonInKm(lat1,lon1,lat2,lon2) {
    var R = 6371; // Radius of the earth in km
    var dLat = deg2rad(lat2-lat1);  // deg2rad below
    var dLon = deg2rad(lon2-lon1);
    var a =
      Math.sin(dLat/2) * Math.sin(dLat/2) +
      Math.cos(deg2rad(lat1)) * Math.cos(deg2rad(lat2)) *
      Math.sin(dLon/2) * Math.sin(dLon/2)
      ;
    var c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1-a));
    var d = R * c; // Distance in km
    return d;
  }

  function deg2rad(deg) {
    return deg * (Math.PI/180)
  }
```

Figure 16

By now you must be wondering how am I getting the Latitude and Longitude coordinates for every user?

I am using the browser geolocation [16] built in feature which gets your location by using data from your Wi-Fi network or IP address.

And calling it in my main app component which means it fetches your location as soon as you open the app.

```
import { useContext ,useState, useEffect } from 'react'
import { LocationContext } from '../store/location-context'

function MyApp({ Component, pageProps }) {
  const [latitude,setLat] = useState("")
  const [longitude,setLong] = useState("")

  useEffect(()=>{
        if(navigator.geolocation){
            navigator.geolocation.getCurrentPosition(function(position) {
            setLat(position.coords.latitude)
            setLong(position.coords.longitude)
        });
      }
  },[])

  return  <LocationContext.Provider value = {{latitude, longitude}}>
            <Component {...pageProps} />
          </LocationContext.Provider>
}

export default MyApp
```

Figure 17

Then I'm using the context API to use these values in other components. Every time the user logs in their location gets sent to the database and gets updated with the new location, which is how I get the coordinates for calculating the distance between users.

```
export default function LogInForm() {
    const router = useRouter()
    const {latitude,longitude} = useContext(LocationContext)
    const [inputValues, setInputValues] = useState({
        username: "",
        password: ""
    });

    const logInData = {
        username: inputValues.username,       MALEK GESHASH - STUDENT, 3 months ago • work in progress …
        password: inputValues.password,
        lat : latitude,
        long : longitude
    }

    async function logInHandler(event) {
        event.preventDefault()
        try {
            await axios.post(`http://${process.env.SERVER_URI}:5000/profiles/login`, logInData,{ withCredentials: true },
            router.push("/")
        } catch (error) {
```

Figure 18

I'm also using the location coordinates to calculate the distance between your current location and each profile's location, on each Card component there is a distance value that is calculated using the coordinates and is displayed so you can know how far each user is from you.

```
setCards(
  data.map((person, index) => {
    const distance = getDistanceFromLatLonInKm(latitude, longitude, person.location.lat, person.location.long)
    return (
      <Card
        key={index}
```

Figure 19

Result in main page with filtered profiles shown:

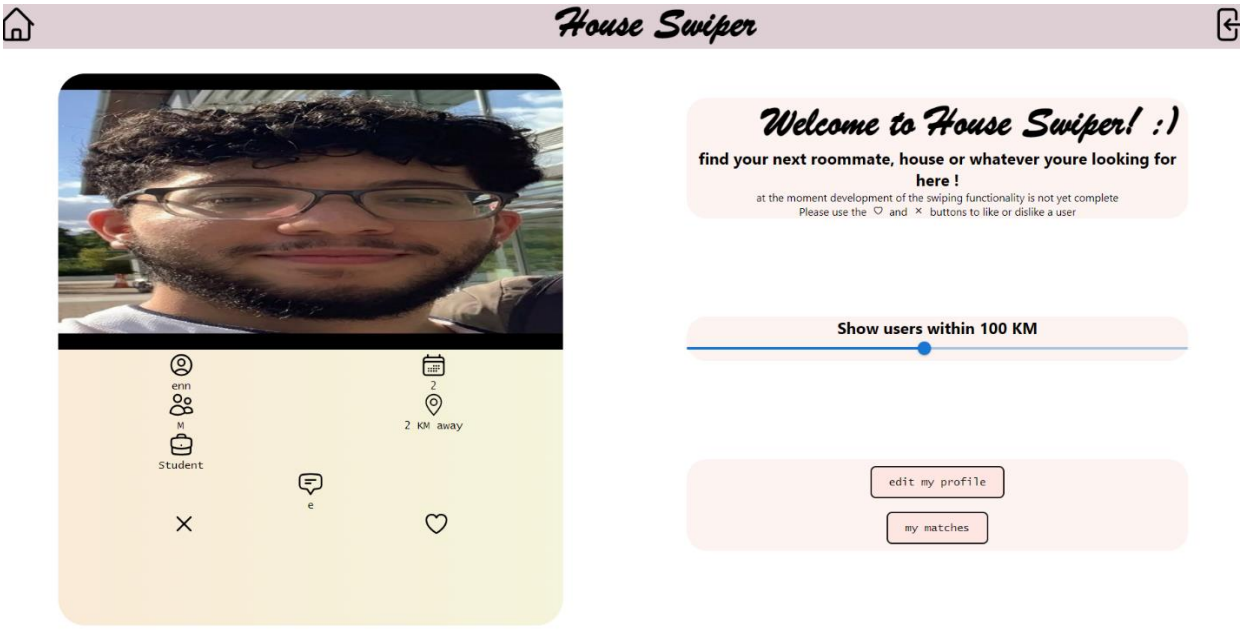

Figure 20

10.3 Messaging

I needed a messaging platform on this web app to enable matches to contact each other, or else it would render the app useless, in this section I will explain the process that allows matches to message each other.

For building a real time messaging app I had several possible approaches, I mostly focused on Web-sockets [17] and Polling [18], after researching and consulting with my Supervisor Brian, I have decided to go with Polling due to the complexity of Web-sockets and the scale of the project.

On the Front-End side, I am using Next JS's Dynamic routing [19] to create the messaging page, Dynamic routing allows you to create pages with flexible and dynamic content depending on the URL which means each individual chat between 2 users will have its own dynamic layout.

```
const router = useRouter();
const userID = router.query.userID
```

Figure 21

To achieve that I created a file with square brackets in the file name, then I could access the URL via the useRouter hook.

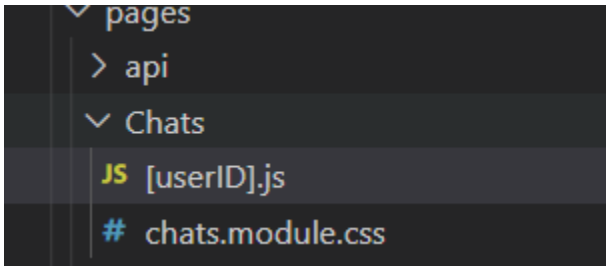

Figure 22

In the [userID].js page I have a useEffect hook that gets called on first render, this hook calls the backend API to fetch the messages from the database and sets them using a SetMessages function which we will look at shortly. The set Interval function then toggles a state variable to cause a re render and call the useEffect again to poll the server for new messages.

```
useEffect(() => {
    async function fetchData() {
        try {
            const getTexts = await axios.get(`http://${process.env.SERVER_URI}:5000/messaging/getChats`, { withCredentials: true, para
            const data = await getTexts.data
            console.log(data)
            setMesagges(settingMessages(data.messages))
        } catch (error) {
            console.log(error)
        }
    }
    fetchData()
    const interval = setInterval(() => {
        setFlag(prev => !prev)
       fetchData();
    }, 1000);
   return() => {
       clearInterval(interval)
    }
}, [])
```

Figure 23

How does it know which users to fetch the messages for? I am using the useRouter hook to get the ID of the user I'm talking to and sending it in the body of the request.

```
{ withCredentials: true, params: { talkingTo: userID }, head
```

Figure 24

On the backend side I have a route defined for fetching all the chats between two users. First it checks if the request is authenticated, then it searches for a room that contains the participants with the ID of the user signed in and the user you're talking to, and it returns that in a JSON format.

```
router.get("/getChats",async (req,res)=>{
    if (req.isAuthenticated()){
        const user = await Profile.findOne({ _id: [req.session.passport.user] })

        const receiver = req.query.talkingTo
        let room = await chatRoom.findOne({particapants : { $all: [user.username,receiver]}})
        res.json(room)
    }
    else{
        res.status(401).json({ msg: 'You are not authorized to view this resource' });
    }
})
```

Figure 25

By looking at the chat room schema you can see that each room has an array of participants and an array of message objects with values from, to, message content.

```
const mongoose = require("mongoose")

const chatRoomSchema = new mongoose.Schema({
    particapants: [{
        type : String
    }],
    messages:[{
        from : { type: String },
        to : { type : String } ,
        message: { type: String }
    }]      You, 3 months ago • created schema, save chats to databas

})
module.exports = mongoose.model('chatRoomSchema',chatRoomSchema)
```

Figure 26

Now that we know how we get the messages we can go back to the mapping function setMessages, we pass the array of object we got from the back end, and we check who sent the

message and apply relevant CSS styles inside a div component

```
function settingMessages(data) {
    return (
        <div className={styles.outerMessagesContainer}>
            <ul className={styles.list}>
                {data.map((messageContent, index) => (
                    <li key={index}>
                        <div className={messageContent.from == userID ? styles.receiver : styles.messages}>{messageContent.from} : {m
                        <div></div>
                    </li>))}
            </ul>
        </div>
    )
}
```

Figure 27

Now lets look at how sending messages is handled, I have an input box that calls the sendMessageHandler function on change, which checks that the message isnt empty and then sends a post request with the message content and who the receiever ID

```
async function sendMessageHandler(event) {
    event.preventDefault()
    if (chatboxContent.length != 0) {
        const response = await axios.post(`http://${process.env.SERVER_URI}:5000/messaging/sendChat`, { talkingTo: userID, message: c
        console.log(response)
        setChatBoxContent("")
    }
}
return (

    <div className={styles.page}>
        <Nav />
        {messages}

        <div className={styles.chatBox}>
            <div id="message-container"></div>
            <form id="send-container" onSubmit={sendMessageHandler}>
                <input type="text" className={styles.inputBox} id="message-input" value={chatboxContent} onChange={(newText) => {
                    setChatBoxContent(newText.target.value)
                }}></input>
                <button className={styles.buttons} type="submit" id="send-button">Send</button>
```

Figure 28

Then in the backend route called /sendChat it finds the sender ID using the session ID, then it looks for a room with the sender and reciever as particapants, if it doesn't exist that means this is the first time they message eachother and it creates a room and updates the chat, but if it finds a room then it updates the room with the new message, the new message will have info about the sender, receiver and content of the message.

```
router.post("/sendChat", async (req,res)=>{
    if (req.isAuthenticated()){
        const user = await Profile.findOne({ _id: [req.session.passport.user] })
        console.log(user.username + " sent this message to " + req.body.talkingTo + " : " + req.body.message)
        let messageDetails = {
            from : user.username,
            to: req.body.talkingTo,
            message : req.body.message
        }
        let room = await chatRoom.findOne({particapants : { $all: [user.username,req.body.talkingTo ]}})
        console.log(room)       You, 3 months ago • created schema, save chats to database …
        if(room == null || room.length  == 0){
            room = new chatRoom({
                particapants : [user.username,req.body.talkingTo ],
                messages : messageDetails
            })
            await room.save()
            res.send("success")
            return
        }
        await chatRoom.updateOne(room,
        { "$addToSet": { "messages": messageDetails} },
        { "new": true, "upsert": true },)
        res.send("success")
    }
    else{
        res.status(401).json({ msg: 'You are not authorized to view this resource' });
    }
})
```

Figure 29

Final Result, chatting with match:

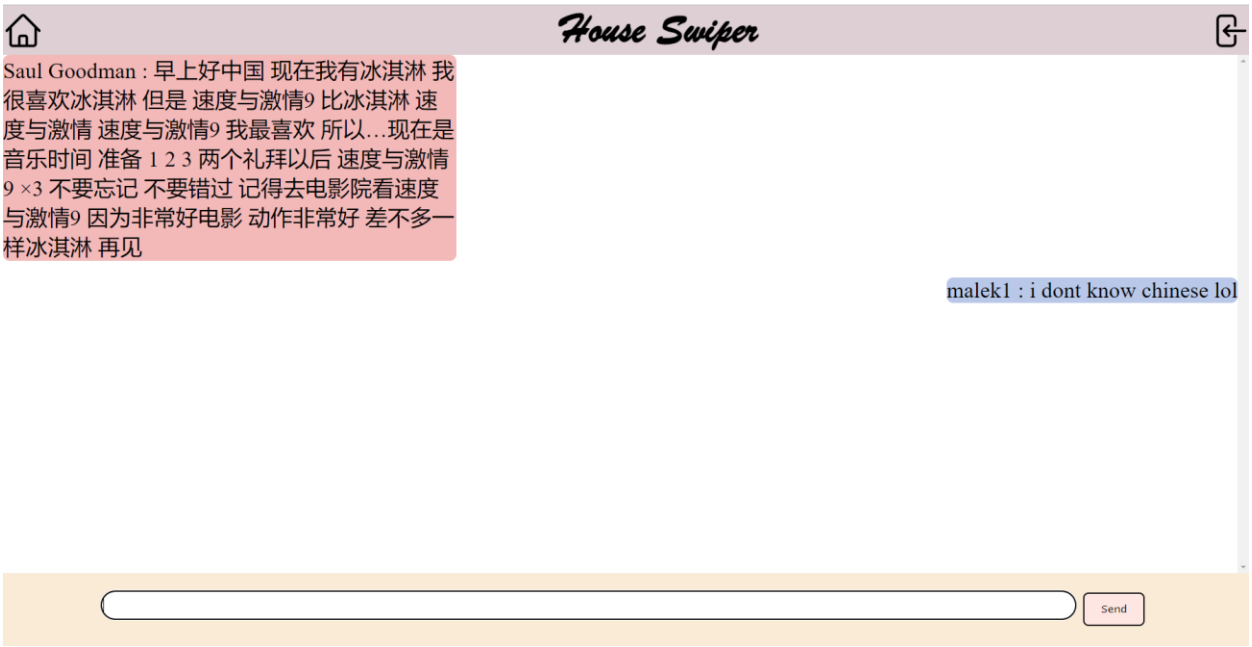

Figure 30

10.4 Authentication

For the authentication I am using a combination of express sessions and passport js, express sessions handle the session cookie, and passport js handles the sign in authentication.

Here I'm using the express session middleware to set a session cookie in the browser when it makes a request, I'm saving the sessions info on a mongo dB collection, so the sessions persist even if the users close the window/browser.

```
const MongoDBStore = require('connect-mongodb-session')(session);

const store = new MongoDBStore({
  uri: process.env.DB_URI,
  collection: 'sessions'
});
      MALEK GESHASH, 2 weeks ago • added session store …
app.use(session({
    secret: "secretcode",
    // store: db,
    cookie: {
        maxAge: 1000 * 60 * 60 * 24 // Equals 1 day (1 day * 24 hr/1 day * 60 min/1 hr * 60 sec/1 min * 1000 ms / 1 sec)
    },
    store: store
}));
```

Figure 31

Here I initialize passport middleware and enable it to serialize and de serialize user ID's from the session.

```
app.use(passport.initialize());        You, 3 mont
app.use(passport.session());
```

Figure 32

Passport JS allows you to define local strategies to set up your own custom authentication, here I want to authenticate using username and password, so I define a function called verify to validate the username and passport in the database and instruct passport to use that function

for my local strategy.

```
async function verify (username,password,done) {
    try {
        const profile = await Profile.findOne({username : username})
        if(!profile){
            return done(null,false)
        }
        const isValid = validatePassword(password,profile.hash,profile.salt);
        if(isValid){
            return done(null,profile)
        }
        else{
            return done(null,false)
        }
    }
    catch (error) {
        // res.send(500).json({
        //     message : err.message
        // })
        console.log(error)
    }
}       MALEK GESHASH - STUDENT, 3 months ago • add passport strategy
const strategy  = new LocalStrategy(verify);

passport.use(strategy);
```

Figure 33

I'm using these 2 functions to generate and validate passwords.

The gen password takes the password in plain text, generates a random salt using the crypto function, and then hashes using the plain text password and the salt, these values are stored in the database (hash and salt), then to verify if a user entered a correct password, it takes the entered password in plain text and hashes it in with the salt stored in the database, if the value is equal to the hash stored in the database the password is correct

```
function genPassword(password) {
    var salt = crypto.randomBytes(32).toString('hex');
    var genHash = crypto.pbkdf2Sync(password, salt, 10000, 64, 'sha512').toString('hex');

    return {
      salt: salt,
      hash: genHash
    };
}       MALEK GESHASH - STUDENT, 3 months ago • add passport strategy

function validatePassword(password, hash, salt) {
    var hashVerify = crypto.pbkdf2Sync(password, salt, 10000, 64, 'sha512').toString('hex');
    return hash === hashVerify;
}
```

Figure 34

Here I define my login and logout routes, I pass the local strategy and request object to the passport.authenticate function, which will use the verify function to verify the user credentials, and if they are correct, it logs in the request. To log out I'm using the req.session.destroy method to destroy the logged in session.

```
router.post("/login", (req, res, next) => {
    passport.authenticate("local", (err, user) => {
        if (!user) {
            return res.status(401).json({ message: "Invalid credentials" });
        }
        req.logIn(user, async (err) => {
            const location = {
                lat : req.body.lat,
                long : req.body.long
            }
            await Profile.findByIdAndUpdate(req.session.passport.user,
                { "$set": { "location": location } },
                { "new": true, "upsert": true },
            );
            res.end()
        });

    })(req, res);
  });


router.delete('/logout', (req, res, next) => {
    req.logout(function(err){
        if(err){ return next(err)}
        req.session.destroy(function (err) {
            if (err) { return next(err); }
            // The response should indicate that the user is no longer authenticated.
            return res.send({ authenticated: req.isAuthenticated() });
          });
    });

});
```

Figure 35

11 Version Control

I am using git for version control and GitHub as a remote repository for my source code, version control was an essential component in my project allowed me to create new branches and experiment with new features without the risk of it breaking the main codebase, also gave me the whole history of the project and allowed me to rollback to different commits if needed.

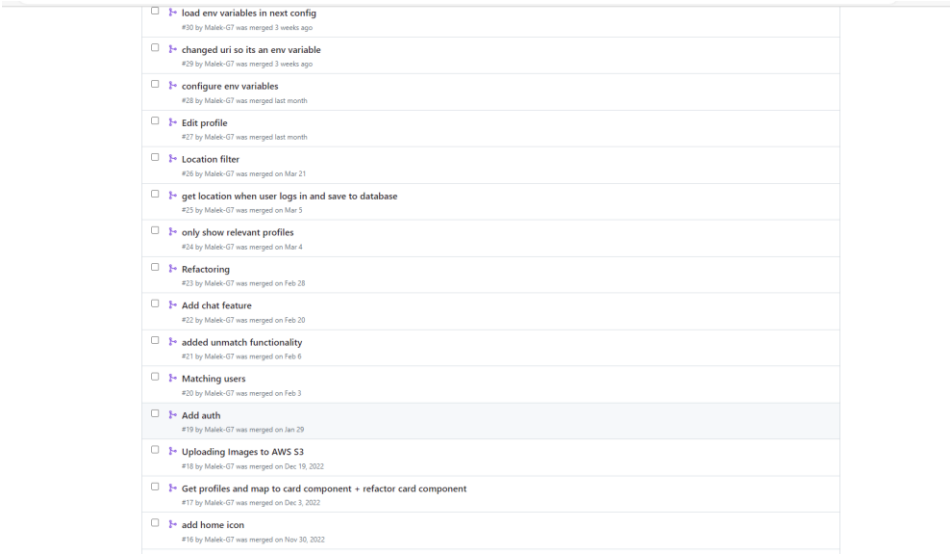

Figure 36

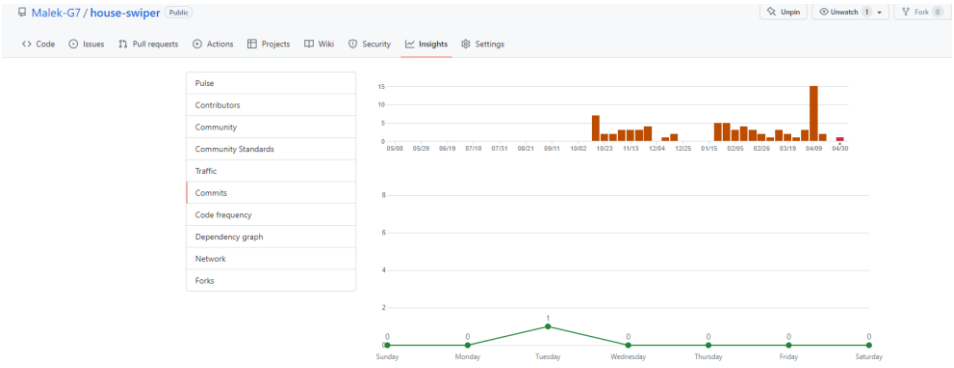

Figure 37

Link to remote repository: [Malek-G7/house-swiper (github.com)](https://github.com/Malek-G7/house-swiper)

12 Ethics

It was important for me to consider the ethical side of this project. My main concern would be introducing additional discrimination against accommodation finders.

We have real data that suggests that digital interactions enable discrimination. For example, it would be a lot easier for a racist landlord to refuse to rent out his accommodation to a person of colour while hiding behind a screen and an anonymous profile, than doing that in person.

However, I do believe that the current popular methods for finding accommodation, Facebook, rent.ie, etc already enable that, and that my application introduces no significant discrimination, another argument is in the case of a racist landlord, they would refuse to rent out their accommodation to a person of colour regardless of the method, it would just speed up the process, instead of rejecting them by phone after a viewing, they would just straight up swipe left on them on the app, which you could argue saves time for all parties, which is a positive thing.

13 Conclusion

In conclusion, I believe I met all the targets I have set for this project, I have learnt many new technologies and languages such as React and Node JS, and AWS cloud technologies, I learnt how the whole process of developing a web application works, from front end to backend to DB to cloud side and hosting, I do believe that this project has huge potential and a lot of room for improvement, and I do plan to continue working on House Swiper after graduation.

Figure 38

14 References

[1] "Tinder," [Online]. Available: https://tinder.com/.

[2] "atlassian," agile, [Online]. Available: https://www.atlassian.com/agile.

[3] "atlassian," Jira, [Online]. Available: https://community.atlassian.com/t5/Jira-articles/What-is-Jira-Software-and-why-use-it/ba-p/2323812.

[4] "Reactjs," [Online]. Available: https://legacy.reactjs.org/docs/getting-started.html.

[5] "nextjs," [Online]. Available: https://nextjs.org/learn/foundations/about-nextjs/what-is-nextjs.

[6] "expressjs," [Online]. Available: https://expressjs.com/.

[7] "postman," [Online]. Available: https://blog.postman.com/rest-api-examples/.

[8] "mdn web docs," HTTP, [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview.

[9] "stack overflow," [Online]. Available: https://stackoverflow.com/questions/45428107/what-does-passport-js-do-and-why-we-need-it.

[10] "aws," S3, [Online]. Available: https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html.

[11] "aws," EC2, [Online]. Available: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html.

[12] "aws," Route 53, [Online]. Available: https://docs.aws.amazon.com/Route53/latest/DeveloperGuide/Welcome.html.

[13] "nginx," [Online]. Available: https://www.nginx.com/resources/glossary/nginx/.

[14] "set up nginx reverse proxy," [Online]. Available: https://youtu.be/_EBARqreeao.

[15] "zerossl," [Online]. Available: https://zerossl.com/.

[16] "mdn web docs," GeoLocation, [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/Geolocation_API.

[17] "mdn web docs," Websockets, [Online]. Available: https://developer.mozilla.org/en-US/docs/web/api/websockets_api.

[18] "wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/Polling_(computer_science).

[19] "youtube," Next js tutorial, [Online]. Available: https://youtu.be/MFuwkrseXVE.