GEEK FINDER

A Project Report submitted in the partial
fulfillment of the requirements for the award of
the degree of

BACHELOR OF TECHNOLOGY

IN

INFORMATION TECHNOLOGY

Submitted by

G. SANJAY VARMA (18PA1A1211)

V.S.L. HARSHA VARDHAN (19PA5A1211)

K. SAI REVANTH (18PA1A1218)

P. V. VIJAYA KRISHNA VARMA (18PA1A1237)

Under the esteemed guidance of

Mr. M. SANDEEP KUMAR

Assistant Professor

DEPARTMENT OF INFORMATION TECHNOLOGY

VISHNU INSTITUTE OF TECHNOLOGY:: BHIMAVARAM

(Affiliated to J.N.T.U. Kakinada, Approved by AICTE & Accredited by NBA)

2021-2022

VISHNU INSTITUTE OF TECHNOLOGY

BHIMAVARAM

(Affiliated to J.N.T.U. Kakinada, Approved by AICTE & Accredited by NBA)

2021-2022

DEPARTMENT OF INFORMATION TECHNOLOGY

VISHNU
UNIVERSAL LEARNING

CERTIFICATE

This is to certify that the project entitled "**GEEK FINDER**", is being submitted by ***G.Sanjay Varma, V.S.L.Harsha Vardhan, K.Sai Revanth,P.V Vijaya Krishna Varma***, bearing the ***REGD.NOS: 18PA1A1211, 19PA5A1211,18PA1A1218*** *and* ***18PA1A1237*** submitted in fulfillment for the award of the degree of "**BACHELOR OF TECHNOLOGY**" in "**INFORMATION TECHNOLOGY**" is a record of bonafide work carried out by them under my guidance and supervision during the academic year 2021-2022 and it has been found worthy of acceptance according to the requirements of university.

Internal Guide

Mr. M. Sandeep Kumar

Assistant Professor

Head of the Department

Mrs.M.SriLakshmi

HOD & Professor

External Examiner

ACKNOWLEDGEMENT

It is natural and inevitable that the thoughts and ideas of other people tend to drift in to the subconscious due to various human parameters, where one feels acknowledge the help and guidance derived from others. We acknowledge each of those who have contributed for the fulfillment of this project.

We take the opportunity to express our sincere gratitude to **Dr.D. Suryanaryana**, director and principal, VIT, Bhimavaram whose guidance from time to time helped us to complete this project successfully.

We are very much thankful to **Mrs. M. Sri Lakshmi**, Head of the Department, Department of Information Technology for her continuous and unrelenting support and guidance. We thank and acknowledge our gratitude for her valuable guidance and support expended to us right from the conception of the idea to the completion of this project.

We are very much thankful to **Mr. M. Sandeep Kumar**, Assistant Professor, our internal guide whose guidance from time to time helped us to complete this project successfully.

Project Associates

G. SANJAY VARMA	**18PA1A1211**
V.S.L. HARSHA VARDHAN	**19PA5A1211**
K. SAI REVANTH	**18PA1A1218**
P. V. VIJAYA KRISHNA VARMA	**18PA1A1237**

ABSTRACT

Now a days having skills are much important to get job. When recruiter is hiring a person, he is basing his judgement on couple of data structure questions whose answers can be found on internet. Due to lack of enough time for recruiter he may not be able to understand whether that student is good enough to do software projects. Students also need teamwork in order to develop amazing software projects. They need to find students around them who are interested in the same area as they do. So, we decided to develop a platform where student can find other students around him who are interested in same domains and projects like him. Recruiters also need to find students to fill their gap in required domains.

So, we are providing a platform where students can collaborate with other students who are developing similar projects like them and also recruiters can get best hires by looking at their projects.

Table of contents

Lists of Images

1)INTRODUCTION

1. INTRODUCTION

In our project we plan to create this platform by using React and Amazon Web Services. We are using react for frontend and using Amazon Web Services for backend. For front end we are using several packages like "react", "react-redux", "react-router", react-router-dom", redux-thunk" in order to develop this particular project. We are also using font awesome icons free solid icon. For backend we are using multiple services like API gateway, AWS Lambda, amazon DynamoDB and web sockets API. By using this powerful frontend and backend services we are going to make a platform which help students to find other students who are doing same project or who are working in the same domain and also, we are helping recruiters who can able to find students who fit their needs and they can surely say they can code well and also can think out of the box.

1.1 Purpose

The purpose for this project is to make a platform that help students to find other students who are working in the same project or same domain. we are also helping recruiters to find students who might be able to fill their needs. We are using powerful frameworks called as react for our frontend and Amazon Web Services for our backend. By using the services, we can be able to create a platform that can be served to millions of students and also the millions of recruiters around the world.

1.2 Hardware Requirements

Processor: Intel Core i5

Ram: 4GB

Hard drive space: 100 GB or above.

x86_64 CPU architecture; 2nd generation Intel Core or newer, or AMD CPU with support for a Windows Hypervisor.

For backend our hardware is maintained by Amazon Web Services only

1.3SoftwareRequirements

Windows 7 or above.

Python 2.7 or above.

Frontend: HTML + CSS, JavaScript, Node + NPM, Redux, Git, JSX.

Backend: AWS Account, API Gateway, AWS Lambda, DynamoDB,

2)FRONTEND

2.FRONTEND

2.1

What technologies we used for frontend?

1)JavaScript

2)Node

3)react

4)react-redux

5)react-router

6)react-router-dom

7)redux-thunk

How to install react and its dependencies?

Initially react depends on Node which is a JavaScript runtime. So, we need to install node and npm before installing react and its packages.

What is JavaScript, HTML and CSS?

JavaScript is a scripting or programming language that allows you to implement complex features on web pages — every time a web page does more than just sit there and display static information for you to look at — displaying timely content updates, interactive maps, animated 2D/3D graphics, scrolling video jukeboxes, etc. — you can bet that JavaScript is probably involved. It is the third layer of the layer cake of standard web technologies, two of which (HTML and CSS) we have covered in much more detail in other parts of the Learning Area.

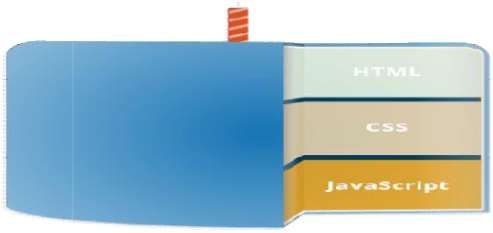

FIG 1: Demonstration of Html,Css and javascript.

HTML is the markup language that we use to structure and give meaning to our web content, for example defining paragraphs, headings, and data tables, or embedding images and videos in the page.

CSS is a language of style rules that we use to apply styling to our HTML content, for example setting background colours and fonts, and laying out our content in multiple columns.

JavaScript is a scripting language that enables you to create dynamically updating content, control multimedia, animate images, and pretty much everything else. (Okay, not everything, but it is amazing what you can achieve with a few lines of JavaScript code.)

What is flex property in CSS? Why we used CSS only?

CSS Flexible Box Layout, commonly known as Flexbox, is a CSS 3 web layout model. It is in the W3C's candidate recommendation (CR) stage. The flex layout allows responsive elements within a container to be automatically arranged depending upon screen size (or device).

Most web pages are written in a combination of HTML (Hypertext Markup Language) and CSS (Cascading Style Sheets). In short, HTML specifies the content and logical structure of the page, while the CSS specifies how it looks: its colours, fonts, formatting, layout, and styling.

CSS flex-box layout is a particular way to specify the layout of HTML pages.

What is node?

Node.js is written in C, C++, and JavaScript, and it is built on the open-source V8 JavaScript engine which also powers JS in browsers such as Google Chrome. As V8 supports new features in JavaScript, they are incorporated into Node.

What is npm?

- npm is the world's largest Software Registry.
- Using npm is Free
- npm is free to use.
- You can download all npm public software packages without any registration or logon.
  Command Line Client npm includes a CLI (Command Line Client) that can be used to download and install software:

Windows Example

C:\>npm install

<package> Mac OS

Example

>npm install <package>

Installing Node and Npm:

Node.js is a run-time environment which includes everything you need to execute a program written in JavaScript. It's used for running scripts on the server to render content before it is delivered to a web browser.

NPM stands for Node Package Manager, which is an application and repository for developing and sharing JavaScript code.

Installing node and npm can be found in below website:

https://phoenixnap.com/kb/install-node-js-npm-on-windows

Now after installing NodeJS and npm let us install react...

Since it is complicated and takes a lot of time, we don't want to configure React manually. create-react-app is a much easier way which does all the configuration and necessary package installations for us automatically and starts a new React app locally, ready for development.

You'll need to have Node >= 14 on your local development machine (but it's not required on the server). You can use nvm (macOS/Linux) or nvm-windows to switch Node versions between different projects.

```
npx create-react-app my-app
(npx comes with npm 5.2+ and higher, see instructions for older
 npm versions) npm
npm init react-app my-app
 npm init <initializer> is available in npm 6+
Yarn
 yarn create react-app my-app
 yarn create is available in Yarn 0.25+
```

we will be using npm as a standard mechanism for installing packages and starting project

cd my-app

Scripts

Inside the newly created project, you can run some built-in commands:

npm start or yarn start

Runs the app in development mode. Open **http://localhost:3000** to view it in

the browser.

The page will automatically reload if you make changes to the code. You will see the build errors and lint warnings in the console.

Since we installed all of the node, npm and react let us now install some of the libraries of react like "react-redux", "react-router", react-router-dom", redux- thunk".

Before that let us understand what JavaScript, react html, CSS etc. Then we will download these libraries.

What is react?

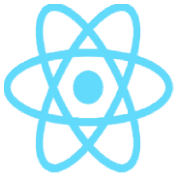

FIG 2:React Framework official logo.

React (also known as React.js or ReactJS) is a free and open-source front-end JavaScript library [3] for building user interfaces based on UI components. It is maintained by Meta (formerly Facebook) and a community of individual developers and companies.[4][5][6] React can be used as a base in the development of single-page, mobile, or server-rendered applications with frameworks like Next.js. However, 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.[7][8]

What is react-redux?

Redux is an open-source JavaScript library used to manage application state. React uses Redux for building the user interface. It was first introduced by Dan Abramov and Andrew Clark in 2015.

React Redux is the official React binding for Redux. It allows React components to read data from a Redux Store, and dispatch Actions to the Store to update data. Redux helps apps to scale by providing a sensible way to manage state through a unidirectional data flow model. React Redux is conceptually simple. It subscribes to the Redux store, checks to see if the data which your component wants have changed, and re-renders your component.

Installation

```
npm install react-redux
```

What is react-router?

React Router is the standard routing library for React.

```
npm i react-router
```

What is redux-thunk?

Redux Thunk is middleware that allows you to return functions, rather than just actions, within Redux. This allows for delayed actions, including working with promises.

```
npm install redux-thunk
```

What is react-router-dom?

React Router DOM enables you to implement dynamic routing in a web app. Unlike the traditional routing architecture in which the routing is handled in a configuration outside of a running app, React Router DOM facilitates component-based routing according to the needs of the app and platform.

Installation

Because we are creating a web app, let's install react-router-dom:

npm install — save react-router-dom

2.2 Login page explanation

Our login page is divided into two parts. They are

- Login Header
- Login Body

LoginHeader contains code for our buttons and we use CSS flexbox for it.

Inside LoginHeader we are having two buttons. When we click on those buttons

we are guiding user into AWS Cognito sign up page.

Once user signs in we are redirecting him into Main Menu pageLoginBody has parallex component which creates the illusion of background not moving and foreground moving.

LoginHeader.js

```
import './LoginHeader.css';
import { useNavigate }
from "react-router-dom";
function LoginHeader()
{
let navigate = useNavigate();
```

```
        return(
         <div className="mainHeader">
          <div className='placeholderDiv'></div>
          <div className="iconDiv">
          <img className="icon"
src="https://t3.ftcdn.net/jpg/03/54/95/66/360_F_354956609_tuIKYmk13U
P2yNaJ4NGcsj8uTsZ4cojL.jpg" />
            <p>Geek finder</p>
          </div>
         <div className="loginButtonPart">
          <button className='recriuterloginButton' ><a href="https://geekfinder.auth.us-
east-
1.amazoncognito.com/login?client_id=2kcq3go6f405oks13u5eafumer&response_type=
token&scope=aws.cognito.signin.u
ser.admin+email+openid+phone+profile&redirect_uri=http://localhost:3000/home">Rec
ruiters login</a></button>
           <button className='studentloginButton' ><a
href="https://geekfinder.auth.us-east-
1.amazoncognito.com/login?client_id=2kcq3go6f405oks13u5eafumer&response_ty
pe=token&scope=aws.cognito.signin.u
ser.admin+email+openid+phone+profile&redirect_uri=http://localhost:3000/home">st
udent login</a></button>
          </div>
         </div>
          );
    }
export default LoginHeader;
```

The loginHeader.js file and LoginHeader.css can be found in below links for complete details:

https://github.com/GadirajuSanjayvarma/GeekFinder/blob/main/src/login/Login.js

https://github.com/GadirajuSanjayvarma/GeekFinder/blob/main/src/login/Login.css

Redux Store:

Workflow of redux is as follows whenever we dispatch an action, we will add a constant to that particular action in redux.so whenever we dispatch an action that dispatch in our application that is handled by only a single redux which contains switch statements handed over to a particular a case statement that our case code will take the particular values and will do logic as follows and update the real state.

redux Store code is as follows:

```
export const ConfigureStore=()=>{

    const
      store=createStore(combineReduc
      ers({ homeresults : HomeResults,
      searchresults: SearchResults,
      userdetails : UserDetails,
      randomuserdetails:RandomUser
      Details, notifications
      :NotificationResults,
      socket:SocketResults,
      messages:MessageResults
```

Connecting store to our application is as follows:

```
<Provider store={store}>
    <Router>
      <Routes>
       <Route path="/" element={<Login />} />
       <Route path="/home" element={<Home socket={socket}/>} />
       <Route path="/profile" element={<Profile/>} />
       <Route path="/search/:searchterm" element={<Search/>} />
```

You can more details on redux store and its implementation in below files:

https://github.com/GadirajuSanjayvarma/GeekFinder/blob/main/src/App.j s

https://github.com/GadirajuSanjayvarma/GeekFinder/blob/main/src/redux /configureStore.js

The modification of data is as follows. We will take an example of getting user details results by using redux

Main Menu dispatching the load results:

```
const mapStateToProps = (state) => { return { homeresults:state.homeresults,
userdetails:state.userdetails, notifications:state.notifications,

};

};

const mapDispatchToProps = (dispatch) => ({

fetchHomeResults: () => dispatch(fetchHomeResults()), fetchUserDetails :
(idToken,accessToken) =>


updateNotifications :(payload) => dispatch(addNotificationDetails(payload))

});

function Home(props)

{

const navigate = useNavigate(); useEffect(() => {
console.log(props.userdetails);

if(props?.userdetails?.results?.accessToken)

{

}

else{

if(!(window.location.href.substring(27) &&
window.location.href.substring(27).split("&")))

{

navigate("/"); return;

}
```

```
        const urlParams =window.location.href.substring(27).split("&");
if(urlParams.length <=0)
        {
        navigate("/");
        }
        const idToken=urlParams[0].substring(9);
        const accessToken =urlParams[1].substring(13);
            if(idToken === null)
        {
        navigate("/");
        }
        //getting access token from url and calling fetch user details
props.fetchUserDetails(idToken,accessToken);
        }
        }, [props.userdetails.results?.data?.username]);

        return(
        <div className='root'>
        <HomeHeader />
        <HomeExplorer data={props.homeresults} />
        <HomeBody posts={props.homeresults} userdata={props.userdetails}/>
        </div>
        );

        }
        export default connect(mapStateToProps, mapDispatchToProps)(Home);
```

Inside fetch User action is as follows:

```
    const fetchUserDetailsResults = (idToken,accessToken) => (dispatch) => {
      dispatch(UserResultsLoading());
       // Send data to the backend via POST
```

```
fetch('https://00mqnzs6ma.execute-api.us-east-
1.amazonaws.com/test/getuserdetails', { // Enter your IP address here

method: 'POST',
mode: 'cors',
body: JSON.stringify({accessToken: accessToken,idToken : idToken})

}).then((result) =>result.json())

.then((result) => { console.log(result);


dispatch(addNotificationDetails(result.playerData.userDetails.notifications));

dispatch(addMessageDetails(result.playerData.userDetails.messages));

dispatch(addUserDetails({data : result.playerData.userDetails,accessToken :
accessToken,idToken:idToken}))

 .catch((err) => { console.log(err);

dispatch(FailedUserResults(err));
});
};
export const UserResultsLoading = () => ({
type: ActionTypes.ADD_USER_RESULTS_LOADING, payload: null,
});
export const addUserDetails = (payload) => (
{
type: ActionTypes.ADD_USER_RESULTS, payload: payload,
});
export default fetchUserDetailsResults;
```

The redux which takes these details and modify the state is as follows:

```
import * as ActionTypes from "./ActionTypes"; const UserDetails = (
state = { isLoading: true, errMessage: null, results: [] }, action
) => {
switch (action.type) {
case ActionTypes.ADD_USER_RESULTS_LOADING:
return { ...state, isLoading: true, errMessage: null, results: [] }; case
ActionTypes.ADD_USER_RESULTS:
return {
...state, isLoading: false, errMessage: null,
results: action.payload,
};
```

```
case ActionTypes.ADD_USER_RESULTS_FAILED: return {
...state, isLoading: false,
errMessage: action.payload, results: [],
};
default:
return { ...state};
}
```

You can find complete implementation of the actions and javascript code in below files:

https://github.com/GadirajuSanjayvarma/GeekFinder/blob/main/src/redux/UserGetDetailsActionCreators%20.js

https://github.com/GadirajuSanjayvarma/GeekFinder/blob/main/src/redux/UserDetailsReducer.js

https://github.com/GadirajuSanjayvarma/GeekFinder/blob/main/src/homePage/ Home.js

2.3 Home page explanation

In home page there will be home header

Inside home header you will find search bar, messages, notifications and user profile button.

You will have explore section where user profiles are begin displayed there according to the recommendation algorithm.

You will find posts sections and filter section where posts section display all of the posts in column wise order.

Filter section you can filter out and request new posts and results based on new parameters you provided.

Skeleton Loading:

We are skeleton loading before data is null. When data gets filled, we will be using Actual rendering without Skeleton Rendering.

The skeleton Loading is as follows:

```
import React from "react"; import "./ProfileScroll.css";
import {useNavigate} from "react-router-dom"

import ExplorerSkeletonLoading from "../Loading/ExplorerSkeletonLoading";
function ProfileScroll(props) {
const navigate = useNavigate(); let render_movies;
```

```
        // console.log(props.data);

        if (props.data.isLoading === false) {

        render_movies = props.data.results[1].map((show, index) => { return (
                    <div key={index} className="movie_render">

        <img src={show.profileUrl} alt="loading" onClick ={()=>{
navigate("/Profile/"+show.username)
                }
        else {
        const array = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0];
        const load = array.map((ele, index) => { return (
        <div key={index} className="movie_render">
        <ExplorerSkeletonLoading />
        </div>
        );
 });
        render_movies = load;
}
        return (
        <div className="container_movies">{render_movies}</div>
        );
 }
        export default ProfileScroll;
```

Explorer Skeleton code is as follows:

ExplorerSkeleton.js

```
import React from "react";
import './ExplorerSkeletonLoading.css'; function ExplorerSkeletonLoading() { return (
<div className="skeleton_container">
<div className="skeleton_loading">
</div>
<div className="skeleton_element">
</div>
<div className="skeleton_element_text"></div>
);
```

The complete implementation can be found below:

https://github.com/GadirajuSanjayvarma/GeekFinder/blob/main/src/Loading/ExplorerSkeletonLoading.js

https://github.com/GadirajuSanjayvarma/GeekFinder/blob/main/src/Loading/ExplorerSkeletonLoading.css

SO, by using similar tactic we will be getting data from databases. We will be getting user details and post, other players details.

Fetch in React:

Whenever we are connecting to server, we need to send data to server. For that we use Fetch API which is built in react

The code sample for sending a http request is as follows:

```
// Send data to the backend via POST
fetch('https://00mqnzs6ma.execute-api.us-east-1.amazonaws.com/test/createpost', { // Enter your IP address here
method: 'POST', mode: 'cors',
body: JSON.stringify(post) // body data type must match "Content-Type" header
}).then((result) =>result.json())
.then((result) => { dispatch(fetchUserDetailsResults(idToken,accessToken));
})
.catch((err) => { console.log(err);
dispatch(FailedUserResults(err));

});
```

So, in this way we use redux to update the post details and profile details. Then once we got details from server, we will render the required data

We will create a post component and we will create posts inside a for loop. The post component code is as follows:

Posts.js

```
function Posts(props)
{
const navigate = useNavigate();
let renderVersions=post?.versions.map((version, index) => { return (
<Versions data={version} topicName={settopicName} details
={setDetails}/>
);
});
return(
<div>
<div className='post_mainContainer'>
<button onClick={() => {setDetails(post.description);
</div>
<div className='versionsContainer'>
{renderVersions}
</div>
<div className='bottomContainer'>
<div className='bottomContainerFooter'>
<button className='socialButton' onClick={() => { console.log({
"postid": post.postid, "type":"incrementLike",

"likes":likes+"", "receiver":post.userName,
"sender":props.userdata.results.data.username
});
setLikes(likes+1);

  {modalOpen &&<EditPostModal setOpenModal={setModalOpen} post={post}
  accessToken ={props?.userdata?.accessToken} idToken
```

```
={props?.userdata?.idToken} userData={props?.userdata?.data} />}
    </div>
    );
    }
    export default Posts;
```

The complete explanation can be found below:

https://github.com/GadirajuSanjayvarma/GeekFinder/blob/main/src/homePage/Posts.js

https://github.com/GadirajuSanjayvarma/GeekFinder/blob/main/src/homePage/Posts.css

The loop for displaying posts inside column is as follows:

Here we are checking if data is still Loading. If they are still loading then we will display skeleton otherwise we will display components with data in.

HomeBody.js

```
let renderPosts;
if (props.isLoading === false) {
  renderPosts = props?.data.map((show,
   index) => { return (
    <div key={index} className="post_render">
     <Posts data={show} own={props?.own} userdata={props.userdata}/>
    </div>);
  });
} else {
 const array = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0];
 const load = array.map((ele, index) => {
  return (
   <div key={index} className="post_render">
    <PostLoading />
   </div>
  );
 })
```

The complete explanation can be found below:

https://github.com/GadirajuSanjayvarma/GeekFinder/blob/main/src/homePage/HomeBody.js

https://github.com/GadirajuSanjayvarma/GeekFinder/blob/main/src/homePage/HomeBody.css

2.4 Search page explanation

Inside search page we will again have three parts:

Home Header

People container

Posts container

Home Header is same as header in home page
We have people container.
We will take search term from browser URL.

The code is as follows:

Search.js

```
const urlParams
=window.location.href.substring(29);
console.log(urlParams);
props.fetchSearchResults(urlParams);
```

Once we got the search term then we will send the term to action. It will take the term and send the term

to server and server gives the search results back. The code for calling actions is as follows:

```
const mapDispatchToProps = (dispatch) => ({
  fetchSearchResults: (searchterm) => dispatch(fetchSearchResults(searchterm)),
  });
  function Search(props)
  {
```

```
useEffect(() => {

const urlParams =window.location.href.substring(29);
props.fetchSearchResults(urlParams);
```

SearchAction.js is as follows:

```
const fetchSearchResults = (searchterm) => (dispatch) => {

dispatch(searchResultsLoading());

// Send data to the backend via POST

fetch('https://00mqnzs6ma.execute-api.us-east-1.amazonaws.com/test/getsearchresults',
{ // Enter your IP address here

method: 'POST', mode: 'cors',

body: JSON.stringify({ "searchterm":searchterm

}) // body data type must match "Content-Type" header

}).then((result) =>result.json())

.then((result) => { dispatch(addSearchDetails(result.searchData));

})

.catch((err) => { dispatch(FailedSearchResults(err));

});

};
```

The people container code is as follows. It will check if data is loading.

PeopleContainer.js

```
if (props.data.isLoading === false) {

render_movies = props.data.results[1].map((show,

index) => { return (

<div key={index} className="people_render">

<img src={show.profileUrl} alt="loading" onClick ={()=>{

navigate("/Profile/"+show.username)
}}/>

<div className="people_render_content">

<p>Name:{show.username}</p>
```

```
                <p>description :{show.description}</p>
<p>Interests : {show.topics}</p>
          </div>
        </div>
      );
    });
} else {
 const array = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0];

 const load = array.map((ele,
  index) => { return (
   <div key={index} className="people_render">
    <ProfileLoading />
   </div>
  );
});
render_movies = load;
}
```

The posts container code is as follows. It will check if data is still loading. If it is loading then we will load a skeleton otherwise we will load actual posts data. We will be using same code that is being used in home page.

PostsContainer.js

```
if (props.isLoading === false) {
    renderPosts =
     props?.data.map((show, index)
     => { return (
      <div key={index} className="post_render">
        <Posts data={show} own={props?.own} userdata={props.userdata}/>
```

```
<div key={index} className="post_render">
 <PostLoading />
</div>
 );
});
renderPosts = load;
}
```

The below links will take you to more complete code versions:

https://github.com/GadirajuSanjayvarma/GeekFinder/blob/main/src/search/Search.js

https://github.com/GadirajuSanjayvarma/GeekFinder/blob/main/src/search/PostsSearch.js

https://github.com/GadirajuSanjayvarma/GeekFinder/blob/main/src/search/PeopleSearch.js

2.5 User profile explanation

Inside user profile they are dividing into two parts. One is header and other is body.

One left side is we will display details and right side is we are displaying posts container.

Since our user details is already in redux so we don't need to call the api again. We load the details in our ui.

We also have two models in our code. We have total two buttons. One is edit details which displays modals

when you submit it will send those details to server.

Also, we have created post modal where once you click on the create post button it will display create post modal when we submit it send details to server

The user details profile is as follows:

userprofile.js

```
<div className='profileInforoot'>

{modalOpen &&<Modal setOpenModal={setModalOpen} userData ={userData}
accessToken
```

```
={props?.userdetails?.results?.accessToken} idToken
={props?.userdetails?.results?.idToken} />}

      {PostmodalOpen &&<PostModal setPostModalOpen={setPostModalOpen} userData
={userData} accessToken ={props?.userdetails?.results?.accessToken} idToken

<p>Likes:{userData?.likes}</p>

    <p>Comments:{userData?.comments}</p>
    <p>Posts : {userData?.posts.length}</p>

    <button onClick={() => {

        setPostModalOpen(true

        );

       }}>Create Post</button>

 </div>

 <div className='profileInfoContainer'>

  <PostsContainer isLoading={props.userdetails.isLoading} data={userData?.posts}
own={true} userdata={props?.userdetails?.results}/>

  </div>

</div>
```

The complete code for user profile is as follows:

https://github.com/GadirajuSanjayvarma/GeekFinder/blob/main/src/Userprofile/UserProfile.js

https://github.com/GadirajuSanjayvarma/GeekFinder/blob/main/src/Userprofile/ProfileInfo.js

https://github.com/GadirajuSanjayvarma/GeekFinder/blob/main/src/Userprofile/PostModal.js

https://github.com/GadirajuSanjayvarma/GeekFinder/blob/main/src/Userprofile/EditPostModal.js

We will also have posts container where we take posts data from user and display them in column wise order.

The code used to create post on server is:

```
export const addPostUserDetailsResults =
(postid,postname,username,userprofilepic,description,imageLink,youtubelink,githublink,co
llege,locati on,topics,likes,comments,versions,userData,accessToken,idToken) =>
(dispatch) => {
```

```
        const post={
    "college":college,
    "locations":location,
    "topics":topics,
    "likes":likes,
    "comments":comments
    , "versions":versions
};
// Send data to the backend via POST
    fetch('https://00mqnzs6ma.execute-api.us-east-1.amazonaws.com/test/createpost', { //
  Enter your IP address here
    method: 'POST', mode: 'cors',
    body: JSON.stringify(post) // body data type must match "Content-Type" header
    }).then((result) =>result.json())
    .then((result) => {
    dispatch(fetchUserDetailsResults(idToken,accessToken));
    })
    .catch((err) => { console.log(err);
    dispatch(FailedUserResults(err));
    });
    };
```

The below link will take you to code that explains a lot better:

https://github.com/GadirajuSanjayvarma/GeekFinder/blob/main/src/redux/UserGetDetailsActionCreators%20.js

https://github.com/GadirajuSanjayvarma/GeekFinder/blob/main/src/redux/UserDetailsReducer.js

The code used to edit details of user will be discussed in backend:

We will be using same function for updating post details because since we will be having same primary key our server will overwrite it.

2.6 Other people profile explanation

Inside other people profile we will again have three parts:

- HomeHeader
- user info
- user uploaded posts

HomeHeader is same as header in

home page. We will take

username from browser URL. **The**

code is as follows:

Profile.js

```
const urlParams
=window.location.href.substring(30);
props.fetchRandomUserDetailsResults(urlPa
rams);
```

Once we got the username then we will send the term to action.It will take the term and send the term

to server and server gives the search results back.

The code for calling actions is as follows:

Profile.js

```
<div className='profileroot'>
<HomeHeader />
<ProfileInfo data={props.randomuserdetails}/>
</div>
```

RandomUserGetDetails.js is as follows:

```
import * as ActionTypes from "./ActionTypes";

const fetchRandomUserDetailsResults = (username) =>

   (dispatch) => { dispatch(RandomUserResultsLoading());

    // Send data to the backend via POST

    fetch('https://00mqnzs6ma.execute-api.us-east-
    1.amazonaws.com/test/getrandomuserdetails',
{ // Enter your IP address

    here method: 'POST',

    mode: 'cors',

    body: JSON.stringify({username:username}) // body data type must match
"Content-Type" header

  }).then((result) =>result.json())

  .then((resul

    t) => {

    console.lo

    g(result);

    console.log(result.playerData.userDetails);

    dispatch(RandomaddUserDetails(result.playerData.userDetails));

  })

  .catch((err) =>

   {

   console.log(err

   );

   dispatch(RandomFailedUserResults(err));
```

The below link will explain a lot better about code used above:

https://github.com/GadirajuSanjayvarma/GeekFinder/blob/main/src/redux/RandomUserGetDetailsActionCreators.js

https://github.com/GadirajuSanjayvarma/GeekFinder/blob/main/src/redux/RandomUserDetailsReducer.js

WebSocket Explanation:

The web has travelled a long way to support full-duplex (or two-way) communication between a client and server. This is the prime intention of the WebSocket protocol: to provide persistent real-time communication between the client and the server over a single TCP socket connection.

2.7 Notification explanation

Notification button is available in home header.
Here when you click on notification it will be displaying all Latest notifications inside a drop down. It shows the username and the action he does. If a user likes a post, then it is like username likes the post.

For notifications I am using redux store. I created a state called notifications and I will be updating it using actions and reducer.

The actions for notifications are as follows

```
import * as ActionTypes from "./ActionTypes";
export const addNotificationDetails = (payload) =>
({
type:
ActionTypes.ADD_NOTIFICATION_RESULTS,
payload: payload,
});

switch (action.type) {
case
ActionTypes.ADD_NOTIFICATION_RESULTS:
return {
...state,
isLoading: false,
errMessage:
null,
results: action.payload,
}
default:
return { ...state};
}
}
export default NotificationResults;
```

How to transfer the notification:

```
<button className='socialButton' onClick={() => {
 console.log({
  "postid": post.postid,
  "type":"incrementLike",
  "likes":likes+"",
  "receiver":post.userName,
  "sender":props.userdata.results.data.username

 });

  // Send data to the backend via POST
fetch('https://00mqnzs6ma.execute-api.us-
east-
1.amazonaws.com/test/updatepostdetails', { // Enter your IP address here

method: 'POST',
mode: 'cors',
body: JSON.stringify({
 "postid": post.postid,
 "type":"incrementLike",
 "likes":likes+"",
 "receiver":post.userName,
 "sender":props.userdata.results.data.username

}) // body data type must match "Content-Type" header

});
```

So, when I press the like button then I am sending a rest API request to update the likes.

So, at the same time inside the backend, we will find user who posted that post and we will send notification to owner of that post.

The backend code is explained in backend section:

Once we got the message to socket then on frontend, we need to get that message and need to store it on notification state inside redux state.

The frontend code is as follows:

```
socket.onmessage =

({data})=>{
```

```
            if(socketdata.type==="message")
            {
        }
    else
            {
             if(socketdata.type==="notification")
             {
               // our code gets here
               const
               senderName=socketdata.message.senderNam
               e; const
               message=socketdata.message.message;
               console.log(message);
               const temp=props.notifications.results;
               temp.push({"senderName":senderName,"message":message});
               console.log(temp);
                props.updateNotifications(temp);
             }
```

The below links will take you to code that implemented notifications better:

https://github.com/GadirajuSanjayvarma/GeekFinder/blob/main/src/homePage/HomeHeader.js

https://github.com/GadirajuSanjayvarma/GeekFinder/blob/main/src/homePage/HomeHeader.css

2.8 Inbox messages explanation

Inbox button is available in home header. Here when you click on it it will take you to inbox page.

The page is divided into three parts.

- One is header component
- Other is profiles component
- Another is messages component

Header component is same as home header component. There are no changes in that

For messages I am using redux store. I created a redux state.

So initially we will look at message's reducer and actions.

Messages Actions are as follows:

```
import * as ActionTypes from "./ActionTypes"; const MessageResults = (
    state = { isLoading: true, errMessage: null, results: [] }, action
    ) => {

    switch (action.type) {
    case ActionTypes.ADD_MESSAGES_RESULTS: return {
    ...state, isLoading: false, errMessage: null,
    results: action.payload,
    };
    case ActionTypes.ADD_SINGLE_MESSAGES: if(action.payload.index===-1)
    {
    const singleMessage=action.payload.message; const sender=action.payload.sender;
    const tempstate=state.results; const message=tempstate.push( { "username": sender,
    "messages": [ singleMessage
    ],
    "profilepic": "https://cdn.pixabay.com/photo/2016/08/08/09/17/avatar-
1577909_1280.png"
    });
    return {
    ...state, isLoading: false, errMessage: null, results: tempstate,
    };
    }
    else
    {
    const singleMessage=action.payload.message; const tempstate=state.results;
```

```
const message=tempstate[action.payload.index];
message.messages.push(singleMessage); tempstate[action.payload.index]=message; return {

...state, isLoading: false, errMessage: null, results: tempstate,

};

}

default:

return { ...state};

}

};

export default MessageResults;
```

So, after completing them we will be looking at how ui is implemented.

The below links will take you to code that explains inbox better:

https://github.com/GadirajuSanjayvarma/GeekFinder/blob/main/src/inbox/Inbox.css

https://github.com/GadirajuSanjayvarma/GeekFinder/blob/main/src/inbox/Inbox.js

3)BACKEND

3.BACKEND

We are using Amazon Web Service for backend communication and for server-side code execution and data storage. In this part we will be looking at AWS and services we used in AWS at greater depth.

3.1 What is AWS??

Amazon Web Services (AWS) is the world's most comprehensive and broadly adopted cloud platform, offering over 200 fully featured services from data centres globally. Millions of customers—including the fastest-growing start-ups, largest enterprises, and leading government agencies—are using AWS to lower costs, become more agile, and innovate faster.

How to create an AWS account??

In order to create a AWS account, you need to follow some steps in order to achieve that.

The steps are mentioned clearly in below link.website(https://progressivecoder.com/creating-an-aws-account-a-step-by- step-process-guide/)

Finally, after logging in, you should be able to see the AWS Management Console as below:

Now once you created the AWS account now we will take a look at what is API gateway and how to create one in step-by-step process and then we will look at our rest API gateway.

3.2 What is API gateway??

An API gateway is an API management tool that sits between a client and a collection of backend services.

An API gateway acts as a reverse proxy to accept all application programming interface (API) calls, aggregate the various services required to fulfil them, and return the appropriate result.

Why use an API gateway?

Most enterprise APIs are deployed via API gateways. It's common for API gateways to handle common tasks that are used across a system of API services, such as user authentication, rate limiting, and statistics.

At its most basic, an API service accepts a remote request and returns a response. But real life is never that simple. Consider your various concerns when you host large-scale APIs.

How to create Rest API using AWS API gateway??

In this getting started exercise, you create a serverless API. Serverless APIs let you focus on your applications, instead of spending time provisioning and managing servers. This exercise takes less than 20 minutes to complete, and is possible within the AWS Free Tier.

We need to follow certain steps in order to create API using API Gateway. The steps are mentioned in below link:

https://docs.aws.amazon.com/apigateway/latest/developerguide/getting- started.html

GeekFinder API explanation:

We can find our API in the lists. Our API name is geekfinderapi.

When we click on that we will take to resources within our API

The resources and explanation are as follows:

create post:

it is a post method which takes details from user and

handles it to function that handles that request.

edit details:

it is a post method which takes details from user and edit posts.

Getmethod details:

it is a post method which takes details from user and return home details.

Getrandomuser details:

it is a post method which takes details from user and handles user details.

Getsearch results:

it is a post method which takes details from user and handles data that is relevant to that search term.

Getuserdetails:

it is a post method which takes details from user and the user details.

send message:

it is a post method which takes details from user and sends messages to the user.

Update post details:

it is a post method which takes details from user and updates post details.

3.3 What is lambda function??

AWS Lambda is a serverless compute service that runs your code in response to events and automatically manages the underlying computer resources for you. These events may include changes in state or an update, such as a user placing an item in a shopping cart on an ecommerce website.

How to create lambda function?

The below link contains article that explains how to create a lambda function better

https://docs.aws.amazon.com/step-functions/latest/dg/tutorial-creating- lambda-state-machine.html

Lambda functions in our project:

The lambda functions that we have are:

GF-createPost

This lambda function takes input from user and cerates post in data base according to the details submitted by user.
source Code:

Complete source code:

https://github.com/GadirajuSanjayvarma/GeekFinder/blob/main/src/lambdaFunctions/GF-createPost.txt

```
const AWS = require('aws-sdk');
const DynamoDb = new AWS.DynamoDB.DocumentClient({region: 'us-east-1'});

async function addPostToUser(username,postid) {

const docClient = new AWS.DynamoDB.DocumentClient(); let params = {
TableName: 'GF-userData', Key: {
username: username,
},

ReturnValues: 'ALL_NEW',
UpdateExpression: 'set #posts = list_append(#posts,:item)', ExpressionAttributeNames: {
'#posts': 'posts'
},
ExpressionAttributeValues: { ':item': [postid]
}
};
return await docClient.update(params).promise();

}
```

GF-AddDetails

This lambda function takes input from user and add details in users' data base according to the details submitted by user.

source Code:

Complete source code:

https://github.com/GadirajuSanjayvarma/GeekFinder/blob/main/src/lambdaFunctions/GF-AddDetails.txt

```
updateDetails(collegeName,cgpa,profileUrl,Interests,userName) {
    const params = {
    TableName: 'GF-userData', Key: {
    username: userName,
    },
    UpdateExpression: 'set college = :x , cgpa = :y, profileUrl = :z, topics = :a', ExpressionAttributeValues: {
    ':x': collegeName, ':y': cgpa,
    ':z': profileUrl, ':a': Interests
    },
    ReturnValues:"UPDATED_NEW"
    };
    let result = await docClient.update(params).promise();
    //console.log(result); return result;
    }
```

GF-updatePostDetails

This lambda function takes input from user and update posts in posts data base according to the details submitted by user.

Source Code:

Complete source code:

https://github.com/GadirajuSanjayvarma/GeekFinder/blob/main/src/lambdaFunctions/GF-updatePostDetails.txt

```
async function updateLikes(postid,likes) {
const docClient = new AWS.DynamoDB.DocumentClient(); let params = {
TableName: 'GF-posts', Key: {
postid: postid,
},
UpdateExpression: 'set likes = :item', ExpressionAttributeValues: { ':item':
}
};

return await docClient.update(params).promise();
}
async function updateComment(postid,commentJson) {
const docClient = new AWS.DynamoDB.DocumentClient();

let params = { TableName: 'GF-posts', Key: {
postid: postid,
},
UpdateExpression: 'set comments = list_append(comments,:item)',

ExpressionAttributeValues: { ':item': [commentJson]
}
};
return await docClient.update(params).promise();
}
```

GF-retreiveUserData

This lambda function takes input from user and retrieve users' data from users'

Source Code:

Complete source code:

https://github.com/GadirajuSanjayvarma/GeekFinder/blob/main/src/lambdaFunctions/GF-retreiveUserData.txt

```
async function getData(username) {

    let documentClient = new AWS.DynamoDB.DocumentClient(); let
    params =    {

    TableName: 'GF-userData', Key: {'username': username}

    };

    let userDetails = await documentClient.get(params).promise(); return
{"userDetails":userDetails.Item};
```

```
    async function getPosts(posts) {
    let documentClient = new AWS.DynamoDB.DocumentClient(); let result=[];


    for(const postid of posts)
    {
    let params = { TableName: 'GF-posts', Key: {'postid': postid}
    };
    let userDetails = await documentClient.get(params).promise();
    result.push(userDetails.Item);
    }
    return result;
}
```

GF-validatePlayerDetails

This lambda function takes input from user and validates if username or email or phone number exists or not from users' data base according to the details submitted by user. It will then decide whether to create account for user or not.

Source Code:

The complete source code is:

https://github.com/GadirajuSanjayvarma/GeekFinder/blob/main/src/lambdaFunctions/GF-validatePlayerDetails.txt

```
exports.handler = async (event) => {
    if (event.userName && event.userPoolId && event.request &&
event.request.userAttributes && event.request.userAttributes.email) {
    //searching for mail
    let cognitoRequestParams = { UserPoolId: event.userPoolId,
    Filter: 'email = "' + event.request.userAttributes.email + '"'
    };
    await Cognito.listUsers(cognitoRequestParams)
```

```
        .promise().then(data => {
        if (data.Users.length > 0 && data.Users[0].Username.toLowerCase() !=
event.userName.toLowerCase()) {
        throw new Error('An account already exists for that email');
        }
        })
        .catch(err => { throw(err);
        });
        }
        return event;
};
```

GF-WS-handleUser

This lambda function takes input from user and stores the users web socket connection id in databases

Source Code:

The complete source code is as follows:

https://github.com/GadirajuSanjayvarma/GeekFinder/blob/main/src/lambdaFunctions/GF-WS-handleUser.txt

```
 exports.handler = async (event) => {
        const body=JSON.parse(event.body);
        const userName=body.userName;
        const connectionId=event.requestContext.connectionId;
        const dynamoDbRequestParams = { TableName: 'GF-
playerConnections', Item: {
        userName: {S: userName}, connectionId:{S:connectionId}}};
        await DynamoDb.putItem(dynamoDbRequestParams)
        .promise().then(data => { console.log(data);
        }) .catch(err => {
                throw(err);
        });
```

};

GF-searchResults

This lambda function takes input from user and searches for users or posts related to that

Source Code:

The complete source code is as:

https://github.com/GadirajuSanjayvarma/GeekFinder/blob/main/src/lambdaFunctions/GF-searchResults.txt

```
async function getSearchData(searchterm) { let temp;
    let result=[]
    //now searching for posts var params = { TableName: 'GF-posts',
    FilterExpression: "begins_with(college,:searchterm) or
begins_with(locations,:searchterm) or begins_with(#posterName,:searchterm) or
begins_with(userName,:searchterm) or begins_with(topics,:searchterm)",
    ExpressionAttributeValues: { ":searchterm": searchterm }, ExpressionAttributeNames:
{
    "#posterName": "Name",
    },
};
   temp=await DynamoDb.scan(params).promise(); result.push(temp.Items);
    //now searching for profiles var params = {
    TableName: 'GF-userData',
        FilterExpression: "begins_with(username,:searchterm) or
    begins_with(college,:searchterm) or begins_with(familyName,:searchterm) or
    begins_with(#locations,:searchterm)"

    ExpressionAttributeValues: { ":searchterm": searchterm }, ExpressionAttributeNames:
{
    "#locations": "location",
    },
};
    temp=await DynamoDb.scan(params).promise(); result.push(temp.Items);
```

```
return result;
```

GF-getuserdetails

This lambda function takes input from user and searches for users or posts related to that.

Source Code:

The complete source code is:

https://github.com/GadirajuSanjayvarma/GeekFinder/blob/main/src/lambdaFunctions/GF-getuserdetails.txt

```
async function getData(username) {
   let documentClient = new AWS.DynamoDB.DocumentClient(); let params = {
   TableName: 'GF-userData',

   Key: {'username': username}
   };
   let userDetails = await documentClient.get(params).promise(); return
{"userDetails":userDetails.Item};

}
```

GF-InitializeUserdata

This lambda function executes only once when new user cerates account and it will cerates records of that user in database.

Source Code:

```
const dynamoDbRequestParams =

  { TableName: 'GF-userData',

  Item: {

    username: {S:

    userName}, userid : {S :

    playerId}, email:  {S:

    email}, location: {S:

    location}, birthdate: {S:

    birthday}, gender: {S:

    gender}, profileUrl: {S:
```

```
        profileUrl}, phoneno: {S:
        phoneNo},
        familyName: {S:
        familyName}, college : {S : ''},
        cgpa : {N : '0'},
        topics : {L : []},
        posts : {L : []},


        likes :{N : '0'},
        comments :{N : '0'},
        friends : {L : []},
        messages: {L:[]},
        notifications:{L:[]}

      }
    };
      await DynamoDb.putItem(dynamoDbRequestParams)
    .promise().then(data => {
    })
    .catch(err => {
      throw(err);
    });
```

GF-WS-sendMessage

This lambda function takes sends messages to the user using web sockets for Realtime messaging and notifications

Source Code:

The complete source code is:

https://github.com/GadirajuSanjayvarma/GeekFinder/blob/main/src/lambdaFunctions/GF-WS-sendMessage.txt

async function

```
addMessage(senderName,receiverName,senderItem,receiverItem) { const
docClient = new AWS.DynamoDB.DocumentClient();

        },

    ReturnValues: 'ALL_NEW', UpdateExpression: 'set #messages =
:item', ExpressionAttributeNames: {

    '#messages': 'messages'

},

    ExpressionAttributeValues: { ':item': senderItem

    }

    };

    result.push(await docClient.update(params).promise()); params = {

    TableName: 'GF-userData', Key: {

    username: receiverName,

    },

    ReturnValues: 'ALL_NEW', UpdateExpression: 'set #messages =
:item',

    ExpressionAttributeNames: {

    '#messages': 'messages'

    },

    ExpressionAttributeValues: { ':item': receiverItem

    }

    };

    result.push(await docClient.update(params).promise()); return result;

    }

    try{

    const result= await replyToMessage({ "sender":sender,
"message":message

    },"message", receiverConnectionDetails.Item.connectionId);


    }

    catch (err){ console.log(err);
```

```
}

async function replyToMessage(response,type, connectionId) { const
params = {

ConnectionId: connectionId,

Data: Buffer.from(JSON.stringify( {message: response,"type":type }))

};

return api.postToConnection(params).promise()

}
```

GF-GetHomeData

This lambda function takes input from user and gives relevant home information.

Source Code:

The complete source code is:

https://github.com/GadirajuSanjayvarma/GeekFinder/blob/main/src/lambdaFunctions/GF-GetHomeData.txt

```
async function getHomeData(locations,topics,college,username) {
// Create DynamoDB document client console.log(locations,topics,college);
let temp; let result=[]
var params = { TableName: 'GF-posts',
FilterExpression: "begins_with(college,:college) or begins_with(locations,:location) or
begins_with(topics,:topics)",


ExpressionAttributeValues: { ":college" : college , ":location" : locations,
":topics" : topics

}
};
temp=await DynamoDb.scan(params).promise(); result.push(temp.Items);
//now searching for profiles var params = {
TableName: 'GF-userData',
FilterExpression: "begins_with(college,:college) or begins_with(locations,:location) or
begins_with(topics,:topics)",
ExpressionAttributeValues: { ":college" : college , ":location" : locations,
":topics" : topics
}
```

```
temp=await DynamoDb.scan(params).promise(); result.push(temp.Items);
        //now searching for profiles var params = {
        TableName: 'GF-userData',
        FilterExpression: "begins_with(college,:college) or begins_with(locations,:location) or
begins_with(topics,:topics)",
        ExpressionAttributeValues: { ":college" : college , ":location" : locations,
        ":topics" : topics
        }
        };
        temp=await DynamoDb.scan(params).promise(); result.push(temp.Items);
        return result;
        }
```

3.4 What is AWS Cognito??

Amazon Cognito lets you add user sign-up, sign-in, and access control to your web and mobile apps quickly and easily. Amazon Cognito scales to millions of users and supports sign-in with social identity providers, such as Apple, Facebook, Google, and Amazon, and enterprise identity providers via SAML
and OpenID Connect.

How to create AWS Cognito??

Using an Amazon Cognito user pool, you can create and maintain a user directory, and add sign-up and sign-in to your mobile app or web application.

- Original console
- New console
- To create a user pool
- Go to the Amazon Cognito console. If prompted, enter your AWS credentials.
- Choose Manage User Pools.
- In the top-right corner of the page, choose Create a user pool.
- Enter a name for your user pool, and choose Review defaults to save the name.
- In the top-left corner of the page, choose Attributes, choose Email address or phone number and Allow email addresses, and then choose Next step to save.
- In the left navigation menu, choose Review.
- Review the user pool information and make any

necessary changes. When the information is correct, choose Create pool.

Pricing of AWS Cognito:

Pricing Tier (MAUs)	Price per MAU
First 50,000	Free
Next 50,000	$0.00550
Next 900,000	$0.00460
Next 9,000,000	$0.00325
Greater than 10,000,000	$0.00250

How we use AWS Cognito in our project??

In our project we use AWS Cognito toi save users data. We create user

pools to save user

data. Ideally each application should have a unique user pool to save

the data. When user is signed up for first time there will be pre sign

up trigger.

There are two kinds of triggers.

- Pre sign up trigger
- Post sign up trigger

3 In pre-sign up trigger will be executed when user filled the sign-up form and submitted the details.

4 Once user submitted the details and if everything is fine then our pre sign up lambda function will return 200 status code which makes us to execute post sign up trigger.

5 Post sign-up trigger will be executed when user successfully creates a account for first time.

6 We will link they trigger to a lambda function which will execute it.

7 In that lambda function we will add details to the DynamoDB table . We will create a record in the table and most important stuff will be obtained from Cognito and remaining stuff will be empty and later be filled by user edit details option obtained in our

8 The required attributes for our AWS Cognito application are: Which standard attributes are required?

9 These attributes were selected when the pool was created and cannot be changed.

- address
- birthdate
- email
- family name
- gender
- given name
- middle name
- phone number
- picture

3.5 What is DynamoDB??

Welcome to the Amazon DynamoDB Developer Guide.

Amazon DynamoDB is a fully managed NoSQL database service that provides fast and predictable performance with seamless scalability. DynamoDB lets you offload the administrative burdens of operating and scaling a distributed database so that you don't have to worry about hardware provisioning, setup and configuration, replication, software patching, or cluster scaling.

How to create one and how to update, put, delete values in DynamoDB??

The below link clearly explains how to create a DynamoDB table: https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/getting- started-step-1.html

How DynamoDB is being used in our project:

We have three DynamoDB tables. They are

GF-playerConnections

GF-playerConnections is used to store connectionid for users who are connecting via web sockets. When users connect to WebSocket's they will get a unique connection id. In order to connect to user, the serve needs to know this information. So, we store this information in the server so server and other users can get this information.

The item storage schema is as follows:

{"userName": "sanjayVarma",
"connectionId":
"SxVQTf_1oAMCJxQ="
}

username is primary key and it should be unique.

Connection id is WebSocket connection id and it will change the connectionid every time it connects to the

WebSocket;

GF-posts

GF-posts is used to store post information. When users post a new post, they will get a unique postid. We will get the information from user from REST API and lambda function and needs to post into the database.

The post schema is as follows:

{
"postid": "sanjayVarma0",
"college": "vishnu",
"comments": [
{
"comment": "Amazing work",
"userprofilepic":
"https://upload.wikimedia.org/wikipedia/commons/thumb/3/34/Elon_Musk_Royal_Society_%28crop2%29.jpg/330px-Elon_Musk_Royal_Society_%28crop2%29.jpg"

```
        {
    "comment": "Nice work.Love to collaborate with you",
    "userprofilepic":
"https://upload.wikimedia.org/wikipedia/commons/thumb/3/34/Elon_Musk_Royal
_Society_%28crop2%29.jpg/330px-
Elon_Musk_Royal_Society_%28crop2%29.jpg"
    }
    ],
    "description": "Data is the most important part in the deep neural
networks ",
    "githubLink": "https://github.com/GadirajuSanjayvarma/DenseDepth-and-
Mask-Prediction",
    "image":
"https://github.com/GadirajuSanjayvarma/S15/raw/master/results_images/depth1
3.jpg",
    "likes": 13,
    "locations": "Bhimavaram",
    "Name": "background and foreground seperation",
    "topics": "DeepLearning CS",
    "userProfilePic":
"https://upload.wikimedia.org/wikipedia/commons/thumb/3/34/Elon_Musk_Royal
_Society_%28crop2%29.jpg/330px-
Elon_Musk_Royal_Society_%28crop2%29.jpg",
    "versions": [
    {
    "versionDetails": "completed audio",
    "versionName": "1.0"
    },
    {
    "versionDetails": "completed audio",
    "versionName": "2.0"
    },
    {
    "versionDetails": "completed audio",
    "versionName": "3.0"
    }
    ],
    "youtubeLink": "https://www.youtube.com/watch?v=H5_odM9-0xU"
    }
```


Here,

postid is the primary key which has postid college represents our college name.

comments is an array of JSON objects which has comment and userprofilepic.

Description: It contains posts description. GitHub Link: it contains GitHub link for post. Image: it contains post images.

Likes: it contains post likes.

Location: it contains location at which this post is being created.

Name: Name of the post

Topics: topics that this post belonged to. They are separate dby spaces

userprofilepic: It contains user profilepic that posts this post

Versions: It is an array of json objects which constins versions details and version number.

YouTube link: It contains YouTube link that is relevant to the post.

GF-userData

GF-userData is used to store users' information. When users' data is stored here. When they create a new account, they will get a unique username basedon ouyr pre-trigger function code. We will get the information from user from REST API and lambda function and needs to post into the database.

The schema is as follows:


```
{
  "username": "sanjayVarma",
  "birthdate": "11/09/2000",
  "cgpa": "10",
  "college": "vishnu",
  "comments": 0,
  "email": "sanjayhillstudios@gmail.com",
  "familyName": "Gadiraju Sanjay Varma",
  "friends": [],
  "gender": "male",
  "likes": 0,
  "location": "Bhimavaram",
  "messages": [
  {
    "username": "brahmi",
    "messages": [
    "sorry i am not able to respond#me",
    "sorry i am not ",
    "No problem#other",
```

```
            "No problem#other"
            ],
            "profilepic": "https://lh3.googleusercontent.com/ogw/ADea4I6lybwezLtuCk-gaeaGQLG2T5ThMCNCy8mBtYk9=s32-c-mo"
        }
        ],
        "notifications": [],
        "phoneno": "+918985516677",
        "posts": [
        "sanjayVarma0",
        "sanjayVarma1"
        ],
        "profileUrl": "https://images.news18.com/ibnlive/uploads/2015/02/brahmafeb2.jpg",
        "topics": [
        "CS",
        "ML"
        ],
        "userid": "51c9da52-6d47-4833-a7bb-d1b7706ffad6"
    }
```


The attributes are as follows:

username: primary key of table and user name which is used in api to get details of the user.

birthdate: birthdate of user.

cgpa: cgpa of user.

college: college of the user.

comments: no of comments received by user posts.

email: email of the user.

familyName: formal name given by family to user.

friends: array of json object that contains other users' usernames.

gender: gender of the user.

likes: total likes received by user.

location: present location of the user.

messages: array of json objects where each json object has other contact user name and array of message send by both parties.

the local sending messages will be appended #me while the other party message is appended by #by #another message.

Notifications: Notifications is the array which has group of notifications.

phone no: phone no is the number of user.

posts: posts is the array that contains the primary key of posts.

profileurl: profileurl is the profile picture of the user.

topics: it is the array of elements that the user is interested in.

userid: it is the user id that is obtained by user from amazon Cognito when he signs in

4)RESULTS

4. RESULTS

Students are capable of creating accounts.Once they created a account they can visit main menu and they can take a look at projects happening near them.They will also get recommend a lot of projects and also a lot of students.Students can add posts and can edit them and add new version whenever they want.They can visit other student profiles and message them. We also developed Real-time messaging system and notification system to help students as well.

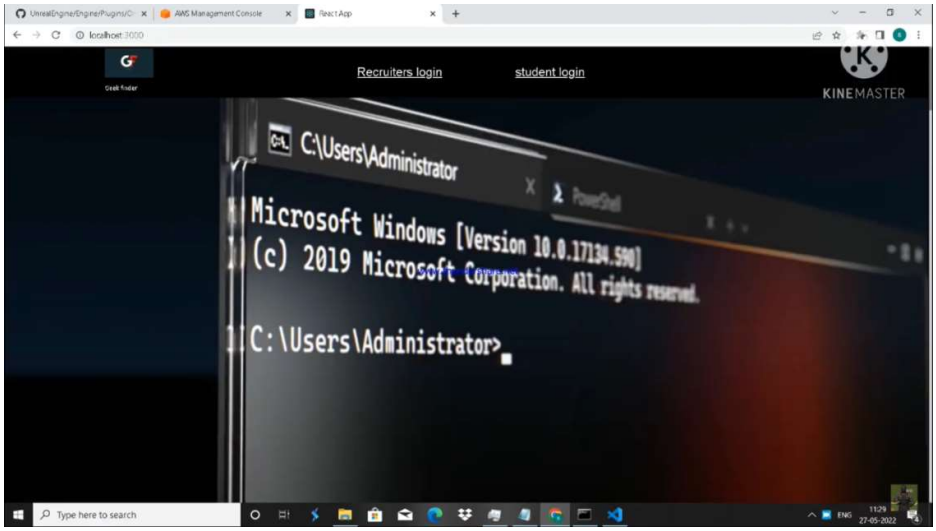

FIG 3:Login Page.

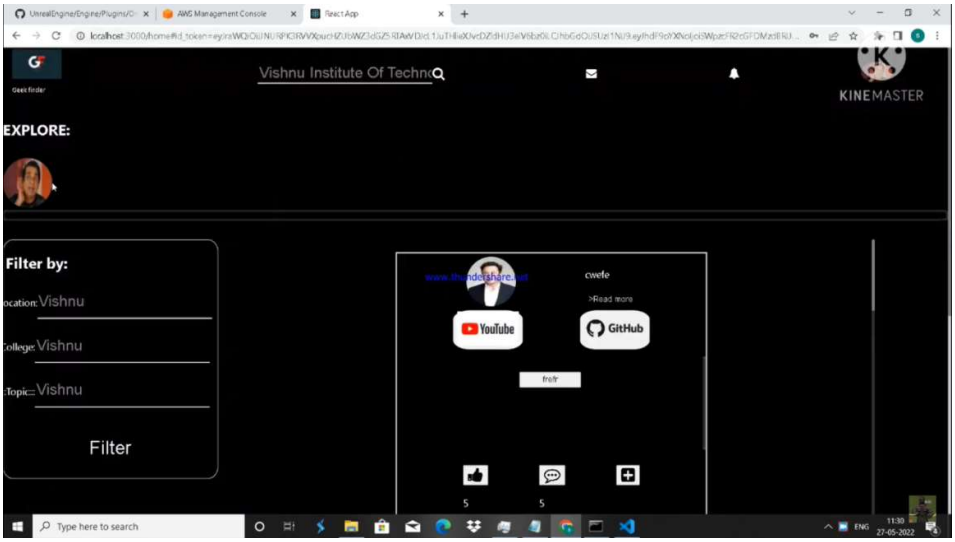

FIG 4:Home Page.

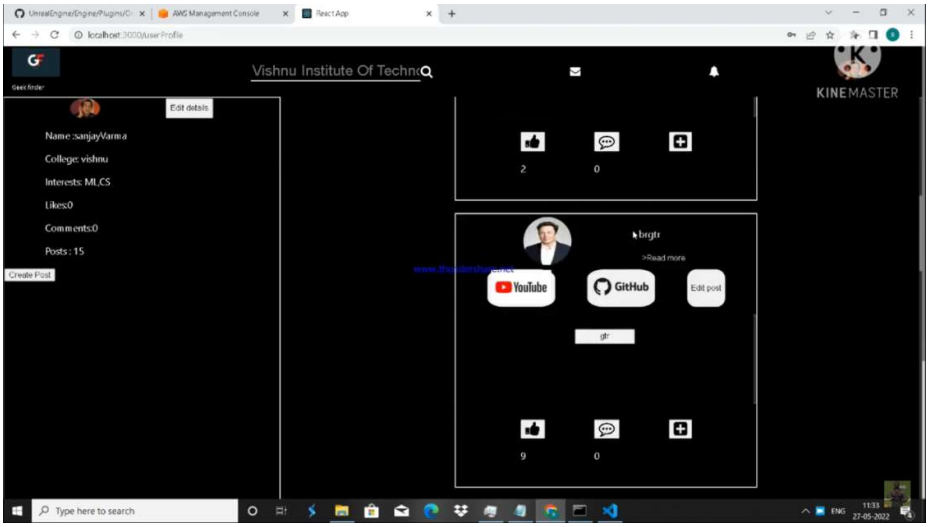

FIG 5:User Profile Page.

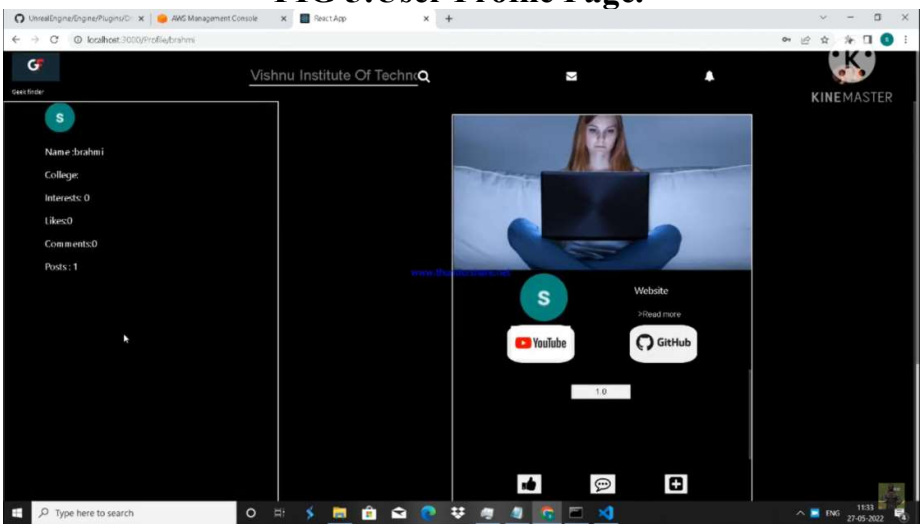

FIG 6:OtherUserProfile Page.

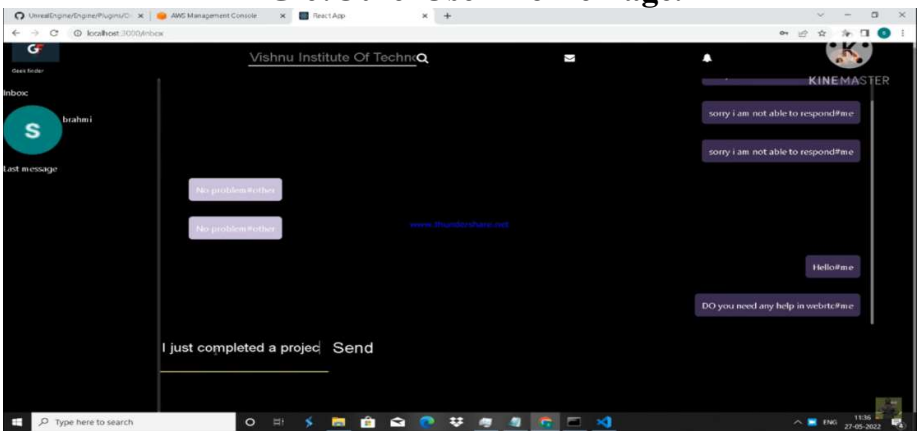

FIG 7:Messages Page.

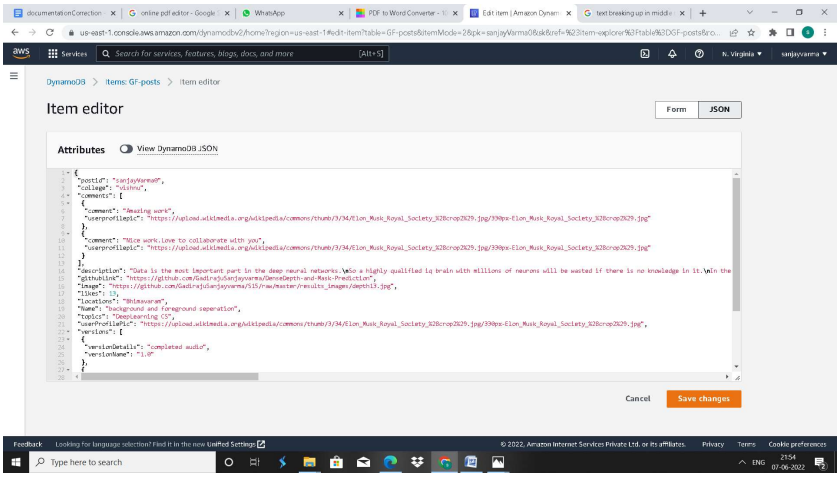

FIG 8:Values Stored in Database.

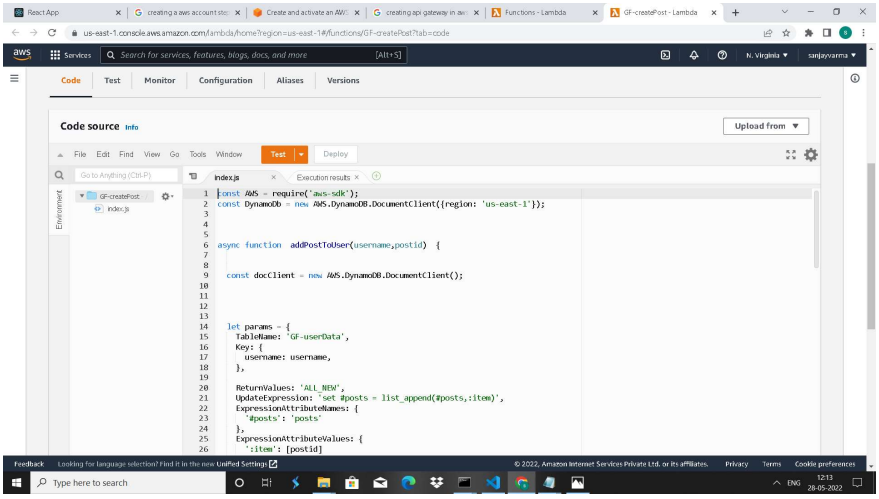

FIG 9:Lambda Function.

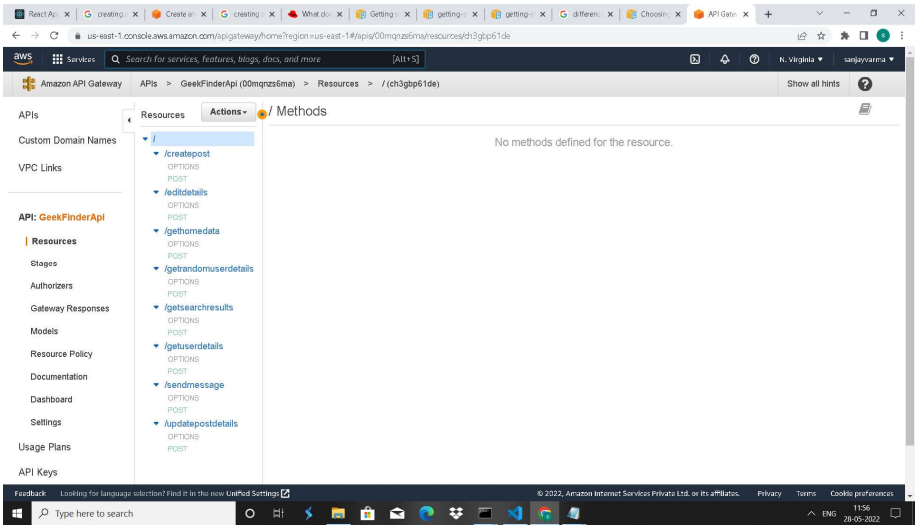

FIG 10:API Gateway.

5)CONCLUSION

5.CONCLUSION

Geekfinder will be a wonderful place for students in order to let other students know what they are doing. They can use our real time messaging system in order to communicate with each other and do wonderful projects together and also improve their communication skills. Recruiters will also be able to find students who are doing projects in domain the company is interested in.

It saves times for recruiters as well. In the end GeekFinder will help students achieve their full potential and getting their dream job and also companies benefitting from that.

6)BIBLIOGRAPHY

6.BIBLIOGRAPHY

[1] https://aws.amazon.com/lambda/
[2] https://aws.amazon.com/dynamodb/
[3] https://reactjs.org/
[4] https://www.npmjs.com/package/@react-spring/parallax
[5] https://react-redux.js.org/
[6] https://nodejs.org/en/
[7] https://blog.logrocket.com/websockets-tutorial-how-to-go-real-time-with-node-and-react-8e46
[8] https://www.npmjs.com/package/react-parallax
[9] https://aws.amazon.com/cognito/
[10] https://www.w3schools.com/react/react_router.asp