

Dev-Connector

A Multi Disciplinary Design project report submitted in partial fulfilment of the requirements for the degree of

BACHELOR OF TECHNOLOGY

in

Computer Science Engineering

by

Saurav Borah

(RA1811027010010)

And

Shubham Beena Deodhar

(RA1811027010051)

Under the guidance of

MS. J. Briskilal

Assistant Professor

Department of Computer Science Engineering

SRM Institute of Science & Technology



At **DEPARTMENT OF COMPUTER SCIENCE ENGINEERING** Kattankulathur, Chennai
(Nov 2020)

**SRM INSTITUTE OF SCIENCE & TECHNOLOGY, KTR CAMPUS DEPARTMENT OF
COMPUTER SCIENCE ENGINEERING**

Register No.:- RA1811027010010 & RA1811027010051

BONAFIDE CERTIFICATE

Certified to be the bonafide record of the work done by **Saurav Borah & Shubham Deodhar** of B.Tech-CSE-BDA, Third year, V Semester for the award of B.Tech degree course in the Department of Computer Science Engineering in **18CSP103L-SEMINAR 1** during the Academic year-2020-21.

PROJECT IN-CHARGE

HEAD OF DEPARTMENT

ACKNOWLEDGEMENT

It is our privilege to express our sincerest regards to our project coordinator, MS. J. Briskilal for her valuable inputs, able guidance, encouragement, whole-hearted cooperation and constructive criticism throughout the duration of our project.

The team deeply expressed our sincere thanks to our Head of Department Dr. B.Amutha for encouraging and allowing us to present the project on the topic “Dev-Connector “at our department premises for the partial fulfillment of the requirements leading to the award of B-Tech degree.

The team is thankful to and fortunate enough to get constant encouragement, support and guidance from all teaching staff of the CSE Department that helped us in successfully completing our project work.

The team heartily thank our friends for their help and suggestions during this project work.

Name of the Student 1 :- SAURAV BORAH

Name of the Student 2 :- SHUBHAM DEODHAR

LIST OF CONTENTS:-

Sno. Chapter Page No. :-

1. Abstract
2. Chapter 1: Introduction
3. Chapter 2: Mongo Atlas setup and Express setup
4. Chapter 3: react-redux setup
5. Chapter 4: Implementation
6. Chapter 5: Results/ Conclusion
7. Chapter 6 : References

ABSTRACT

This second decade of the 21st century in the software development aspect can be termed as “Era of Social media platforms”. We use many Social media platforms like instagram , facebook, linkedin etc. on a daily basis. Social media helps everyone to share their knowledge so that everyone in the society can stay up to date. Social media helps people establish better relationships with their family and friends, and now the networking sites also show their significance for apps. So that's why we decided to create our own model of social media platform named as “Dev-Connector” . It's a way to establish a network or a knowledge-based relationship among Developer community.

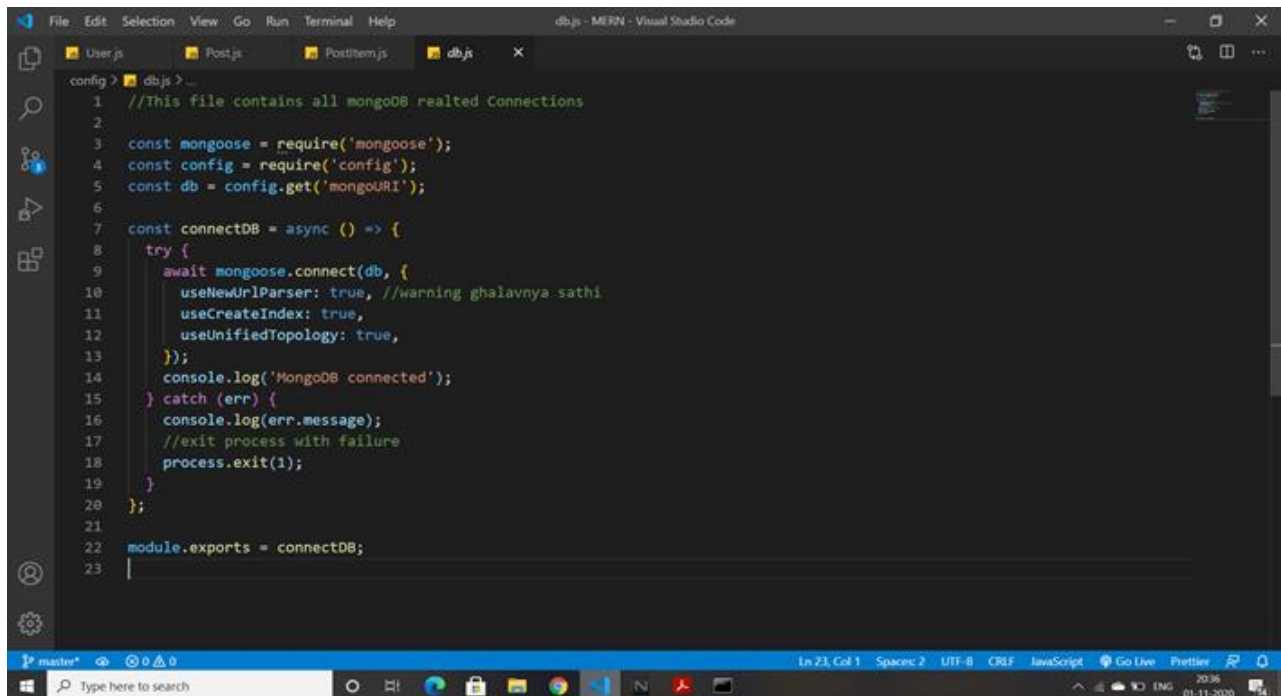
CHAPTER 1: INTRODUCTION

A MERN stack website built with MongoDB, Express.js, React.js and Node.js, which is mainly for connecting developers and allowing them to share their experiences and productions. A platform for developers to connect. They can create their portfolio by adding their experience, education, skills and other important information about their professional career. Users can also create small posts and like/dislike and comment on other posts. Basically, A social resume platform for developers. Create your portfolio by adding experience, education, skills or any important information of your career and post your comments or thoughts for other users. This is a MERN based full stack real world bootstrapping application that can speed up the development process. It uses the popular MongoDB database with Mongoose schemas, the backend web framework Express.js, the frontend library React.js that was created by Facebook and Node.js. Authentication via Json Web Tokens is used with the Passport middleware and customizable linting is built in using eslint.

CHAPTER 2: Mongo-Atlas and Express setup

In this project we chose MERN stack as our development stack. So for the database we used a cloud setup named as mongo-atlas. For API creation we used Node.js , Express JS , Mongoose Schema whereas for encryption of the data we used Json Web token paired with Bcrypt encryption. In a single database we introduced three different collections wiz. Users, Profiles and Posts. We connected those collections to respective Mongoose Schemas . After that those Schemas were used in four different Routing paths wiz. Authentication,Users , Posts and Profiles . During all this setup we strictly followed Model-Controller-View (MVC) setup. Our Backend Code - Code examples: In this project we chose MERN stack as our development stack. So for the database we used a cloud setup named as mongo-atlas. For API creation we used Node.js , Express JS , Mongoose Schema whereas for encryption of the data we used Json Web token paired with Bcrypt encryption. In a single database we introduced three different collections wiz. Users, Profiles and Posts. We connected those collections to respective Mongoose Schemas . After that those Schemas were used in four different Routing paths wiz. Authentication,Users , Posts and Profiles . During all this setup we strictly followed Model-Controller-View (MVC) setup. Our Backend Code - Code examples:

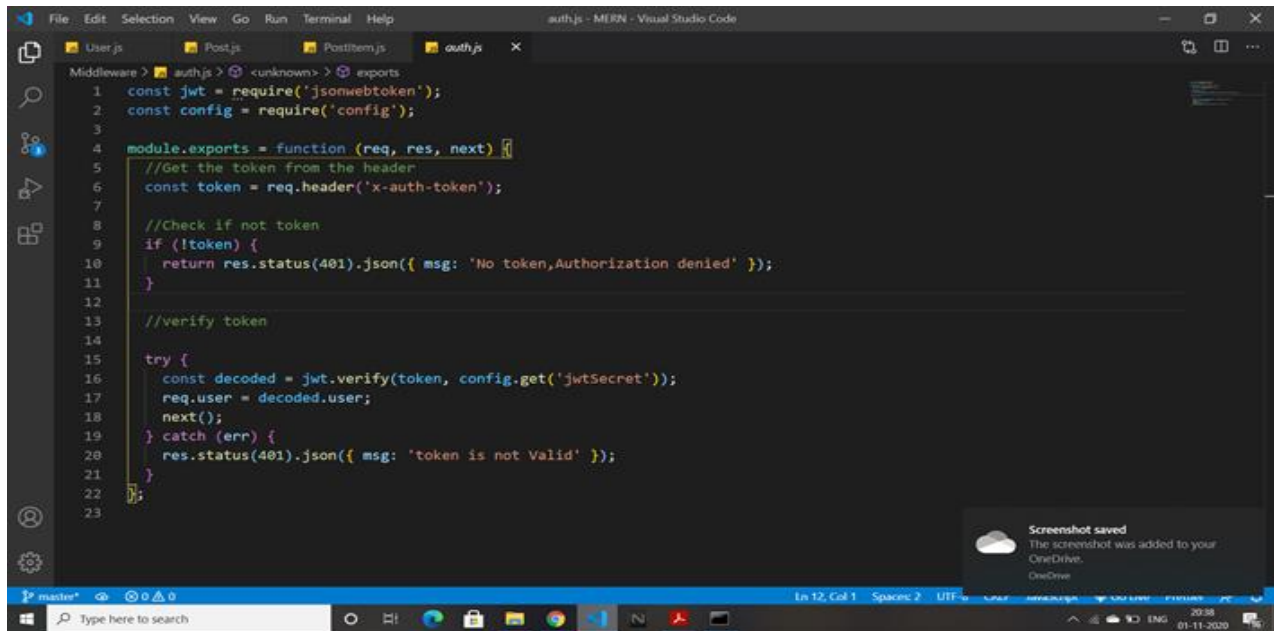
1. Connection between MongoDB-Atlas and express-API

A screenshot of the Visual Studio Code editor interface. The title bar shows 'db.js - MERN - Visual Studio Code'. The editor is open to a file named 'db.js'. The code in the file is as follows:

```
1 //This file contains all mongoDB realted Connections
2
3 const mongoose = require('mongoose');
4 const config = require('config');
5 const db = config.get('mongoURI');
6
7 const connectDB = async () => {
8   try {
9     await mongoose.connect(db, {
10       useNewUrlParser: true, //warning ghalavnya sathi
11       useCreateIndex: true,
12       useUnifiedTopology: true,
13     });
14     console.log('MongoDB connected');
15   } catch (err) {
16     console.log(err.message);
17     //exit process with failure
18     process.exit(1);
19   }
20 };
21
22 module.exports = connectDB;
23
```

The status bar at the bottom indicates 'Ln 23, Col 1', 'Spaces: 2', 'UTF-8', 'CRLF', 'JavaScript', 'Go Live', 'Prettier', and the date '2020-01-11-2020'.

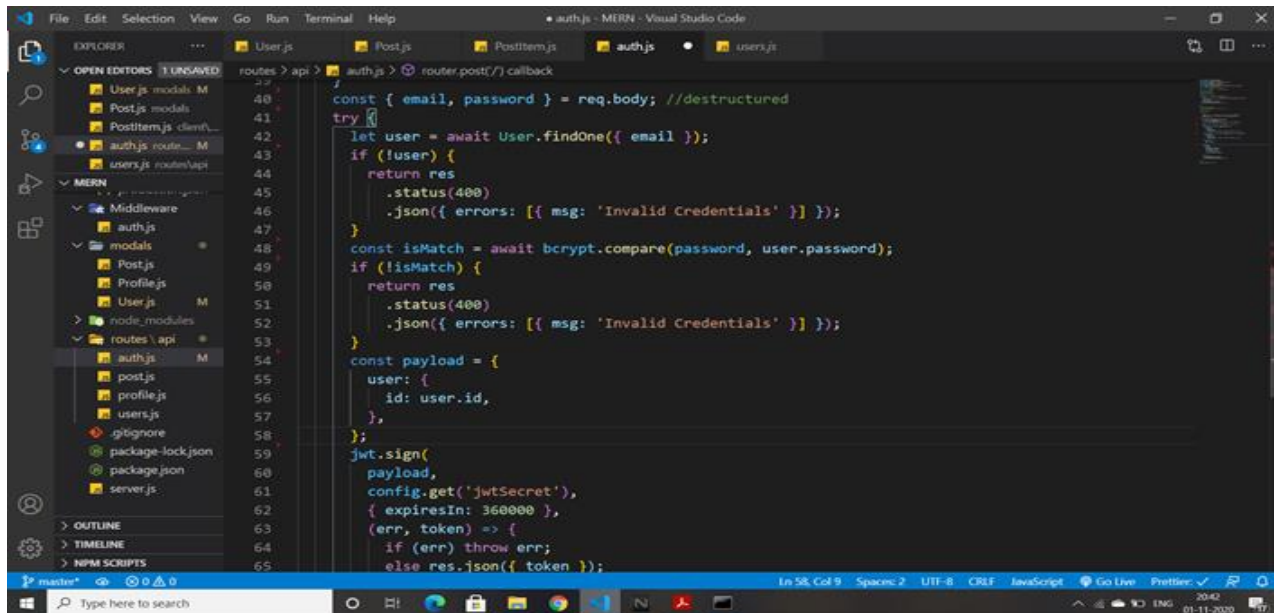
2. Authentication middleware



A screenshot of the Visual Studio Code editor showing a file named `auth.js` in the `authjs` directory. The code implements a JWT authentication middleware. It imports `jsonwebtoken` and `config`. The middleware function checks for a token in the request header, verifies it, and sets the user on the request object before calling `next()`. If the token is missing or invalid, it returns a 401 status with an error message. A status bar at the bottom indicates the cursor is at line 12, column 1.

```
1 const jwt = require('jsonwebtoken');
2 const config = require('config');
3
4 module.exports = function (req, res, next) {
5   //Get the token from the header
6   const token = req.header('x-auth-token');
7
8   //Check if not token
9   if (!token) {
10    return res.status(401).json({ msg: 'No token, Authorization denied' });
11  }
12
13  //verify token
14
15  try {
16    const decoded = jwt.verify(token, config.get('jwtSecret'));
17    req.user = decoded.user;
18    next();
19  } catch (err) {
20    res.status(401).json({ msg: 'token is not Valid' });
21  }
22 };
```

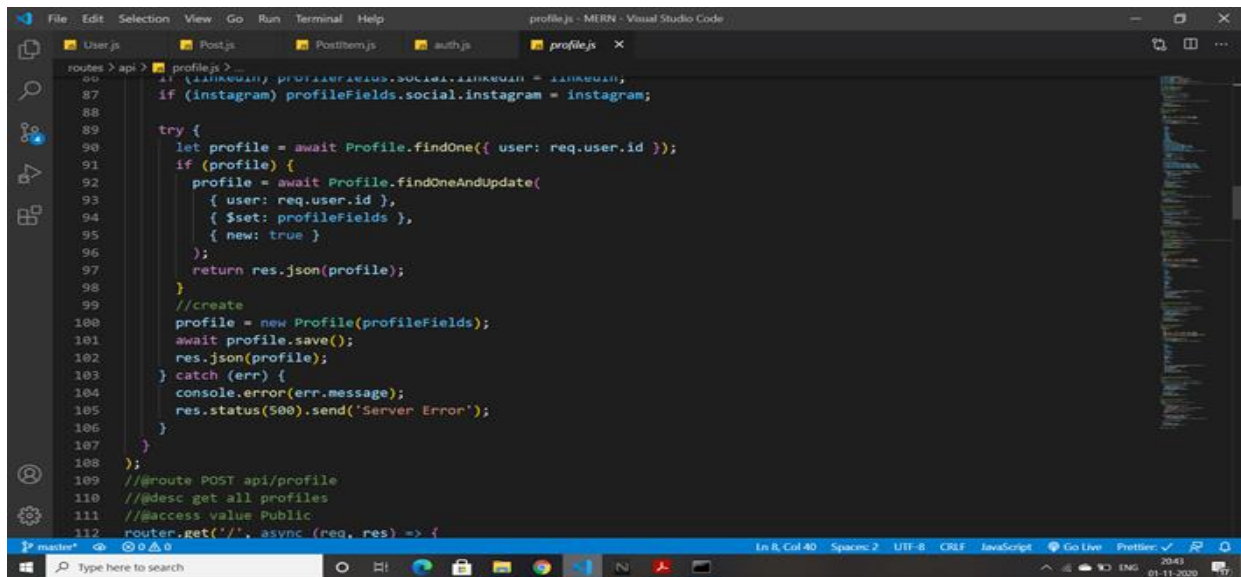
3. Login Setup



A screenshot of the Visual Studio Code editor showing a file named `users.js` in the `users` directory. The code implements a login endpoint. It deconstructs the request body into `email` and `password`. It finds a user by email, compares the password, and if it matches, signs a JWT token and returns it. If the password is invalid, it returns a 400 status with an error message. The status bar at the bottom indicates the cursor is at line 58, column 9.

```
1 const { email, password } = req.body; //destructured
2 try {
3   let user = await User.findOne({ email });
4   if (!user) {
5     return res
6       .status(400)
7       .json({ errors: [{ msg: 'Invalid Credentials' }] });
8   }
9   const isMatch = await bcrypt.compare(password, user.password);
10  if (!isMatch) {
11    return res
12      .status(400)
13      .json({ errors: [{ msg: 'Invalid Credentials' }] });
14  }
15  const payload = {
16    user: {
17      id: user.id,
18    },
19  };
20  jwt.sign(
21    payload,
22    config.get('jwtSecret'),
23    { expiresIn: 360000 },
24    (err, token) => {
25      if (err) throw err;
26      else res.json({ token });
27    }
28  );
29 }
```


4. Profile Creation

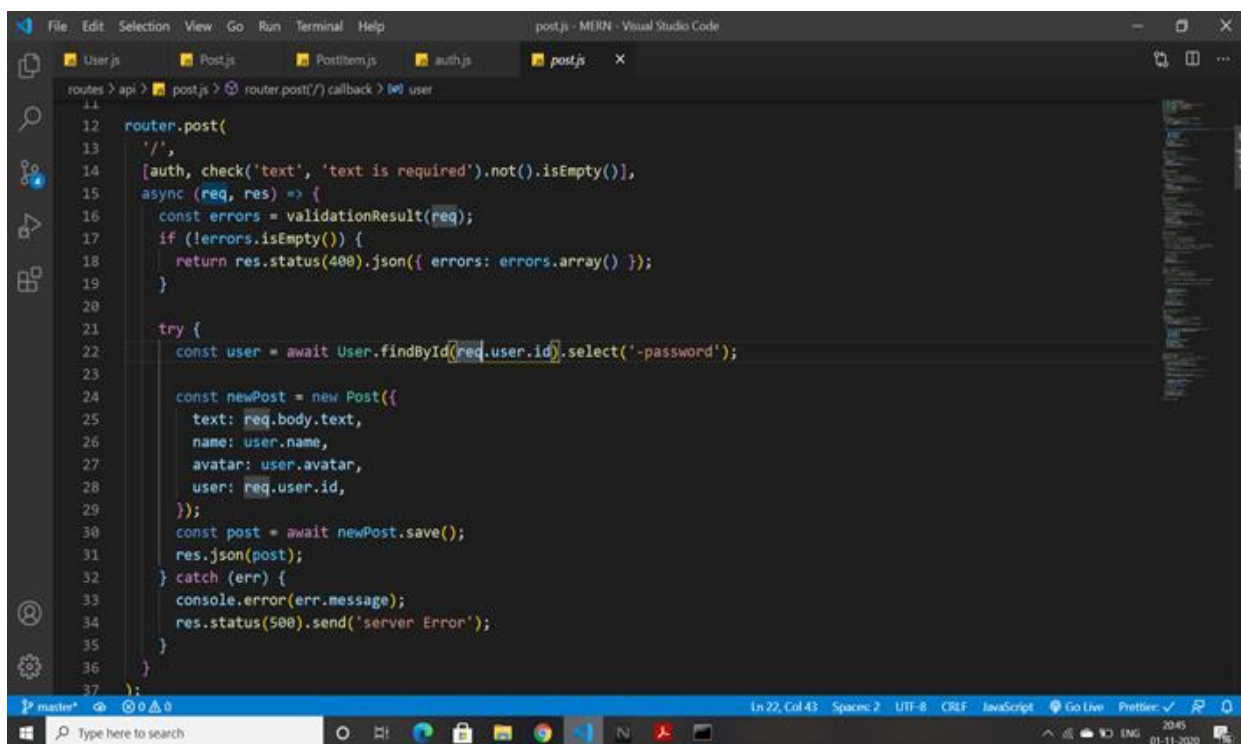


```
File Edit Selection View Go Run Terminal Help
profile.js - MERN - Visual Studio Code

User.js Post.js PostItem.js auth.js profile.js x

routes > api > profile.js > ...
87 if (instagram) profileFields.social.instagram = instagram;
88
89 try {
90   let profile = await Profile.findOne({ user: req.user.id });
91   if (profile) {
92     profile = await Profile.findOneAndUpdate(
93       { user: req.user.id },
94       { $set: profileFields },
95       { new: true }
96     );
97     return res.json(profile);
98   }
99   //create
100   profile = new Profile(profileFields);
101   await profile.save();
102   res.json(profile);
103 } catch (err) {
104   console.error(err.message);
105   res.status(500).send('Server Error');
106 }
107
108 };
109 //@route POST api/profile
110 //@desc get all profiles
111 //@access value Public
112 router.get('/', async (req, res) => {
```

5. Post Creation



```
File Edit Selection View Go Run Terminal Help
post.js - MERN - Visual Studio Code

User.js Post.js PostItem.js auth.js post.js x

routes > api > post.js > router.post() callback > user
11
12 router.post(
13   '/',
14   [auth, check('text', 'text is required').not().isEmpty()],
15   async (req, res) => {
16     const errors = validationResult(req);
17     if (!errors.isEmpty()) {
18       return res.status(400).json({ errors: errors.array() });
19     }
20
21     try {
22       const user = await User.findById(req.user.id).select('-password');
23
24       const newPost = new Post({
25         text: req.body.text,
26         name: user.name,
27         avatar: user.avatar,
28         user: req.user.id,
29       });
30       const post = await newPost.save();
31       res.json(post);
32     } catch (err) {
33       console.error(err.message);
34       res.status(500).send('server Error');
35     }
36   }
37 );
```

6. We can't Share .env file setup bcoz there we stored Sensitive data such as MongoURI, JWT Secret, Github-ClientId and Client Secret etc.

7. This is the list of all NPM packages that we used during the development .

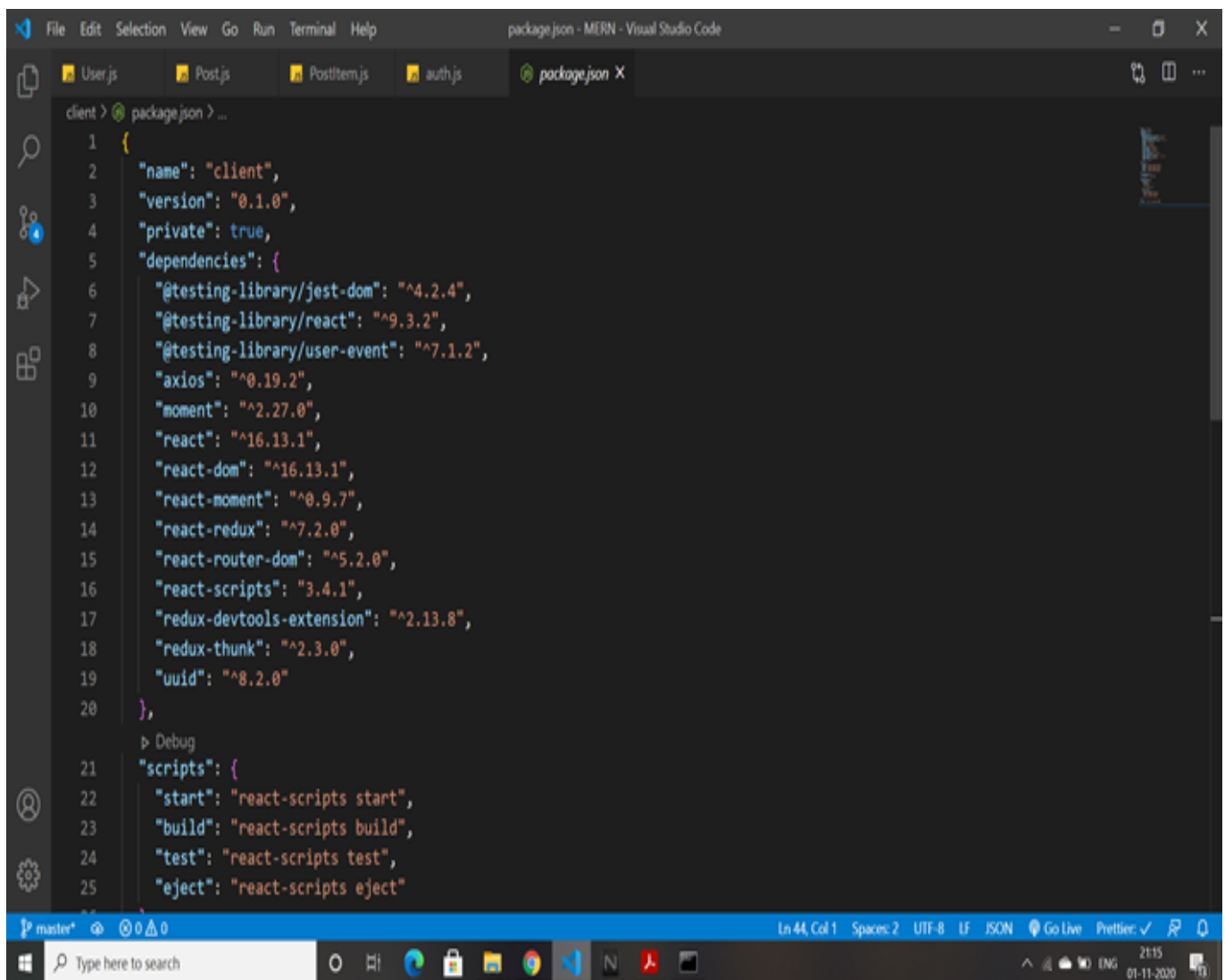
- a. bcrypt
- b. config
- c. express
- d. express-validator
- e. Gravatar
- f. Jsonwebtoken
- g. Mongoose
- h. Concurrently (devDependency)
- i. nodemon (devDependency)

Chapter 3: React-Redux Setup

Once we were done with our Back end Setup , We started to create our Front end . We used Rudimentary knowledge of HTML CSS and Embedded it in react js as JSX .

Then we used redux to connect the React front end and the node backend using axios package and the react-redux package .

These are the packages that we used during front end Development



The screenshot shows the Visual Studio Code editor with a file named `package.json` open. The file contains the following JSON configuration:

```
1 {
2   "name": "client",
3   "version": "0.1.0",
4   "private": true,
5   "dependencies": {
6     "@testing-library/jest-dom": "^4.2.4",
7     "@testing-library/react": "^9.3.2",
8     "@testing-library/user-event": "^7.1.2",
9     "axios": "^0.19.2",
10    "moment": "^2.27.0",
11    "react": "^16.13.1",
12    "react-dom": "^16.13.1",
13    "react-moment": "^0.9.7",
14    "react-redux": "^7.2.0",
15    "react-router-dom": "^5.2.0",
16    "react-scripts": "3.4.1",
17    "redux-devtools-extension": "^2.13.8",
18    "redux-thunk": "^2.3.0",
19    "uuid": "^8.2.0"
20  },
21  "scripts": {
22    "start": "react-scripts start",
23    "build": "react-scripts build",
24    "test": "react-scripts test",
25    "eject": "react-scripts eject"
26  }
27 }
```

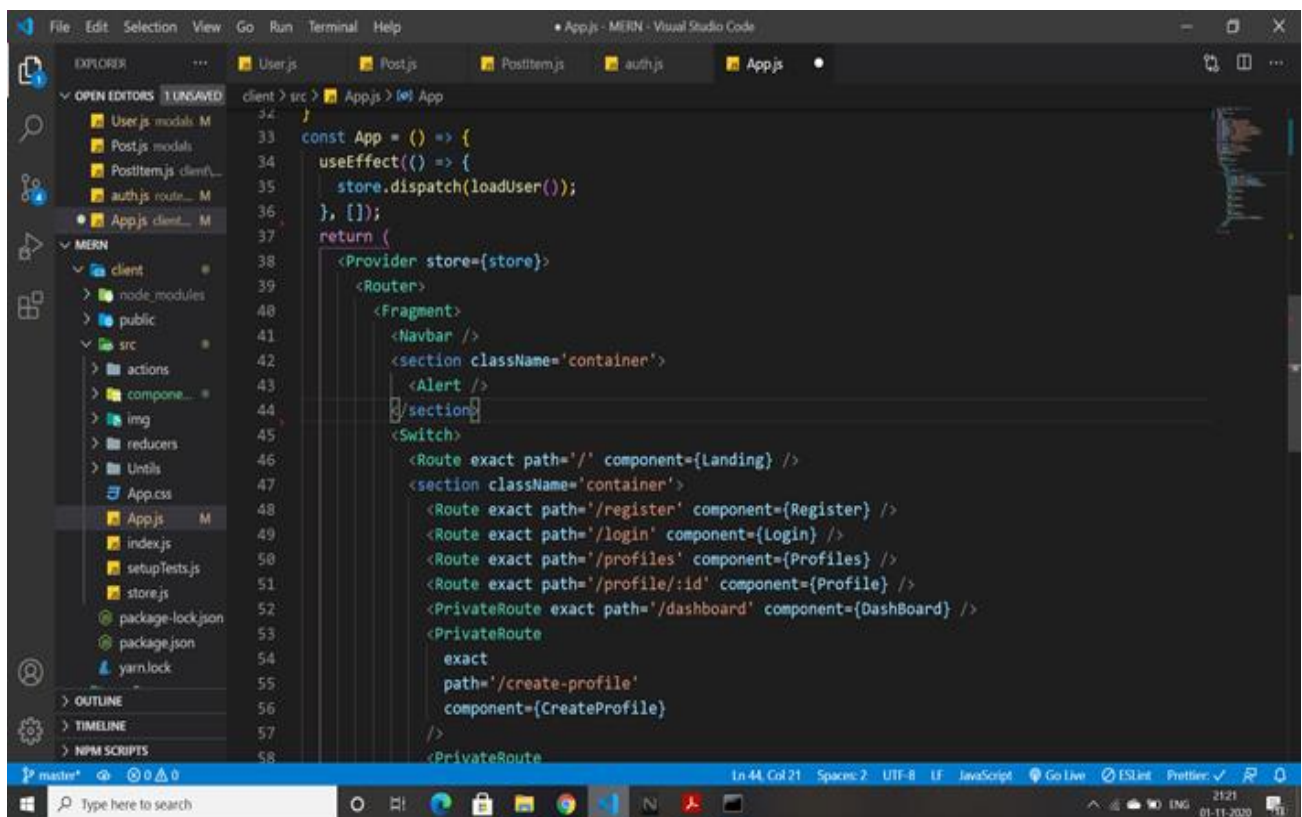
The editor interface includes a sidebar on the left with icons for Explorer, Search, Source Control, Run and Debug, and Extensions. The top status bar shows the current file is `package.json` in the `client` directory, with a line number of 44 and column 1. The bottom status bar displays the Windows taskbar with the search bar and various application icons.

Now let's examine our front end code .

We divided our react and redux files as follows

- a. Actions
- b. Components
- c. Reducers
- d. Store
- e. Utils
- f. App.js

1. Basic react routing setup in root file app.js



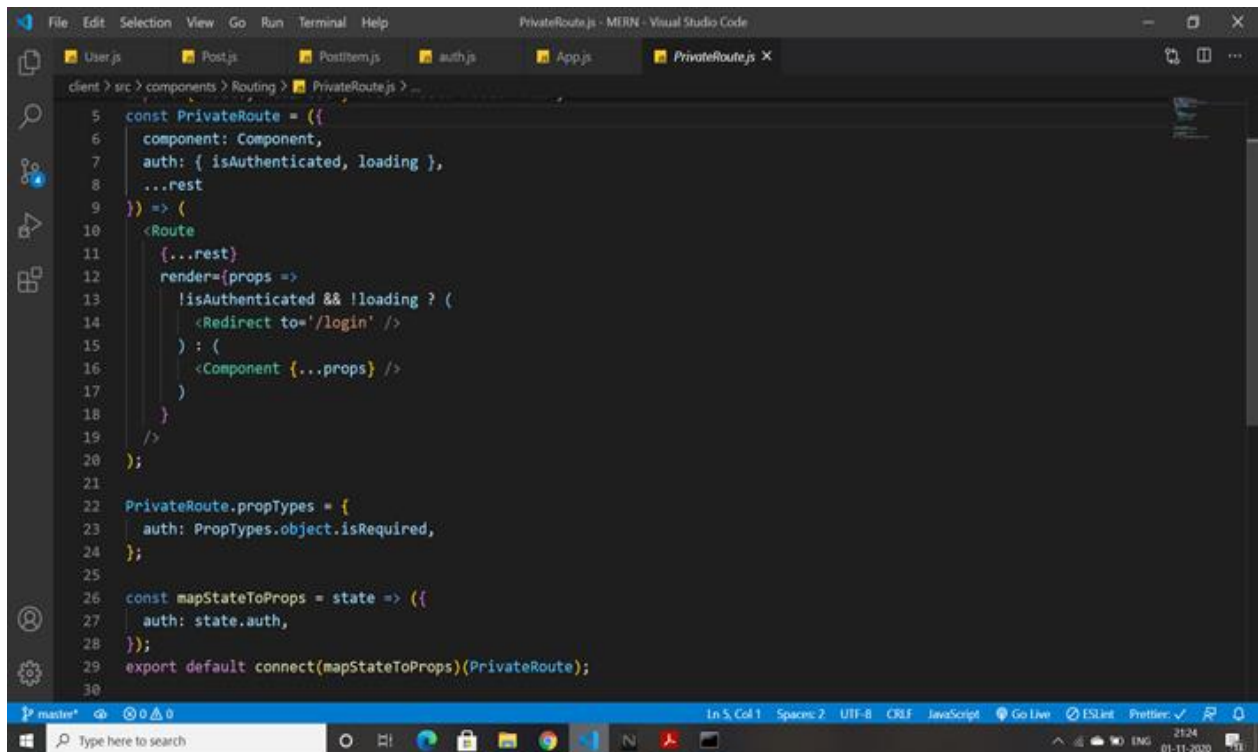
```
File Edit Selection View Go Run Terminal Help
App.js - MERN - Visual Studio Code

EXPLORER
client > src > App.js
  User.js
  Post.js
  PostItem.js
  auth.js
  App.js
  MERN
    client
      node_modules
      public
      src
        actions
        components
        img
        reducers
        utils
        App.css
        App.js
        index.js
        setupTests.js
        store.js
        package-lock.json
        package.json
        yarn.lock
      OUTLINE
      TIMELINE
      NPM SCRIPTS

client > src > App.js
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58

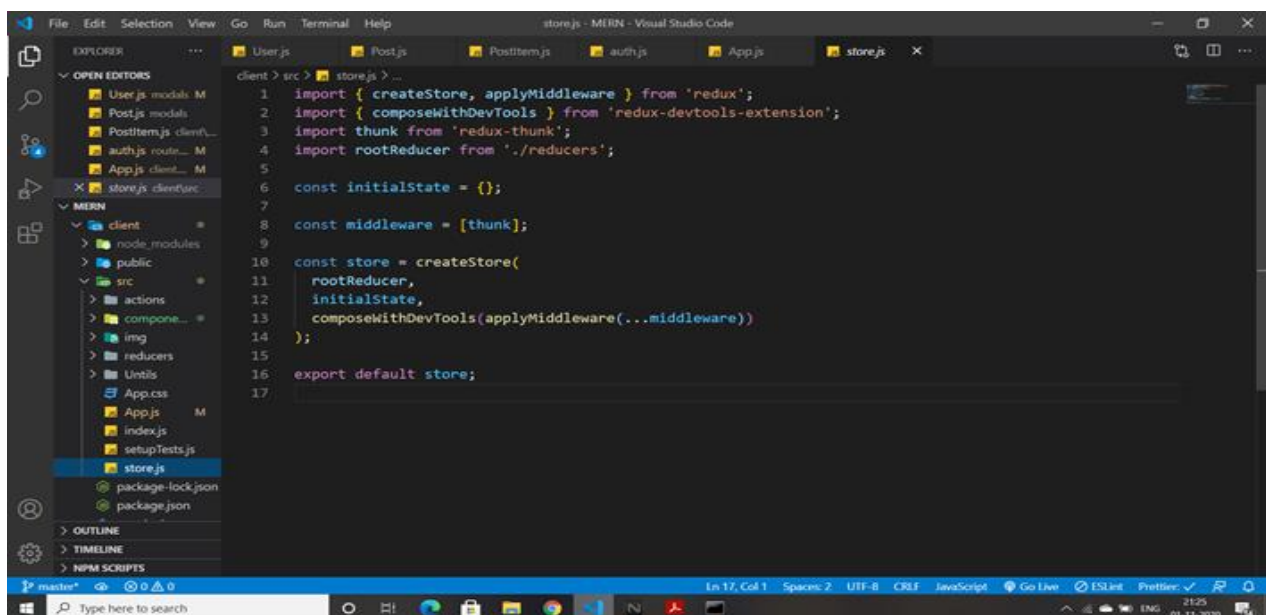
const App = () => {
  useEffect(() => {
    store.dispatch(loadUser());
  }, []);
  return (
    <Provider store={store}>
      <Router>
        <Fragment>
          <Navbar />
          <section className='container'>
            <Alert />
            </section>
          <Switch>
            <Route exact path='/' component={Landing} />
            <section className='container'>
              <Route exact path='/register' component={Register} />
              <Route exact path='/login' component={Login} />
              <Route exact path='/profiles' component={Profiles} />
              <Route exact path='/profile/:id' component={Profile} />
              <PrivateRoute exact path='/dashboard' component={DashBoard} />
              <PrivateRoute
                exact
                path='/create-profile'
                component={CreateProfile}
              />
            </section>
          </Switch>
        </Fragment>
      </Router>
    </Provider>
  );
};
```

2. Private Route Setup, to validate JWT for Authentication

A screenshot of the Visual Studio Code editor showing the file 'PrivateRoute.js' in the 'client > src > components > Routing' directory. The code defines a 'PrivateRoute' component that checks for authentication before rendering. It uses 'isAuthenticated' and 'loading' props to decide whether to redirect to '/login' or render the 'Component'. The component is wrapped with 'connect' and 'mapStateToProps'.

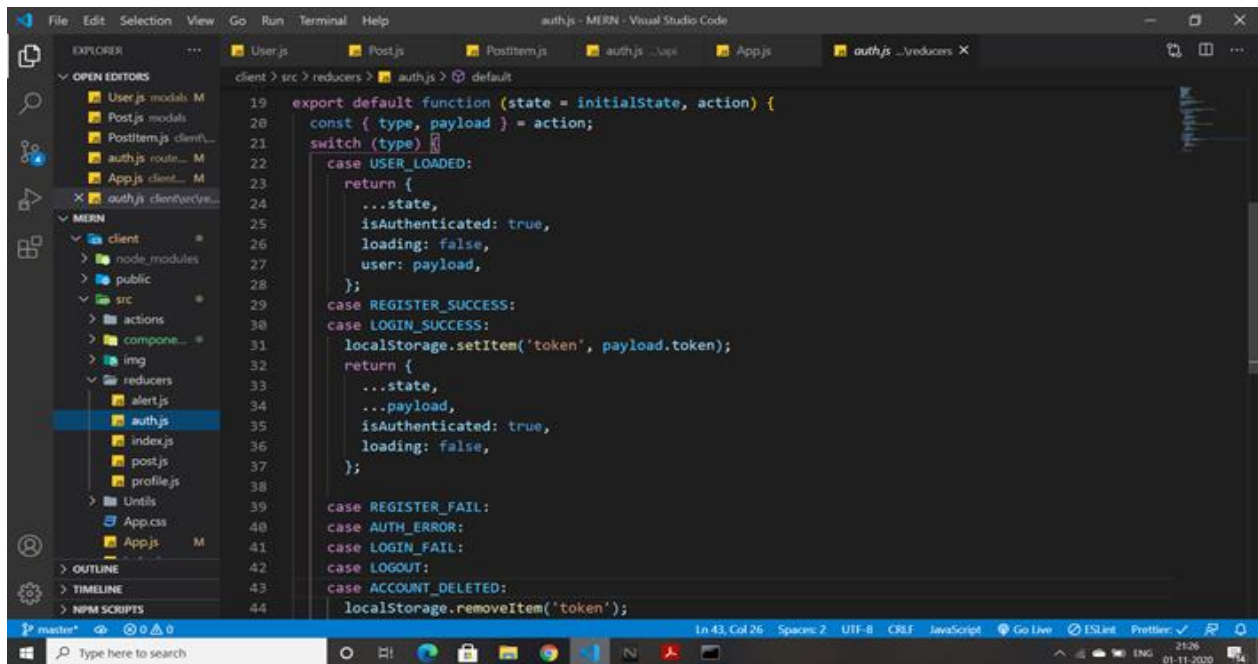
```
5 const PrivateRoute = ({
6   component: Component,
7   auth: { isAuthenticated, loading },
8   ...rest
9 }) => {
10   <Route
11     {...rest}
12     render={props =>
13       !isAuthenticated && !loading ? (
14         <Redirect to="/login" />
15       ) : (
16         <Component {...props} />
17       )
18     }
19   />
20 };
21
22 PrivateRoute.propTypes = {
23   auth: PropTypes.object.isRequired,
24 };
25
26 const mapStateToProps = state => ({
27   auth: state.auth,
28 });
29 export default connect(mapStateToProps)(PrivateRoute);
30
```

3. Redux Store setup

A screenshot of the Visual Studio Code editor showing the file 'store.js' in the 'client > src > store' directory. The code sets up a Redux store using 'createStore', 'applyMiddleware', and 'composeWithDevTools'. It imports 'rootReducer' from './reducers' and 'thunk' from 'redux-thunk'. The initial state is an empty object, and the middleware array includes 'thunk'. The store is then exported as the default export.

```
1 import { createStore, applyMiddleware } from 'redux';
2 import { composeWithDevTools } from 'redux-devtools-extension';
3 import thunk from 'redux-thunk';
4 import rootReducer from './reducers';
5
6 const initialState = {};
7
8 const middleware = [thunk];
9
10 const store = createStore(
11   rootReducer,
12   initialState,
13   composeWithDevTools(applyMiddleware(...middleware))
14 );
15
16 export default store;
17
```

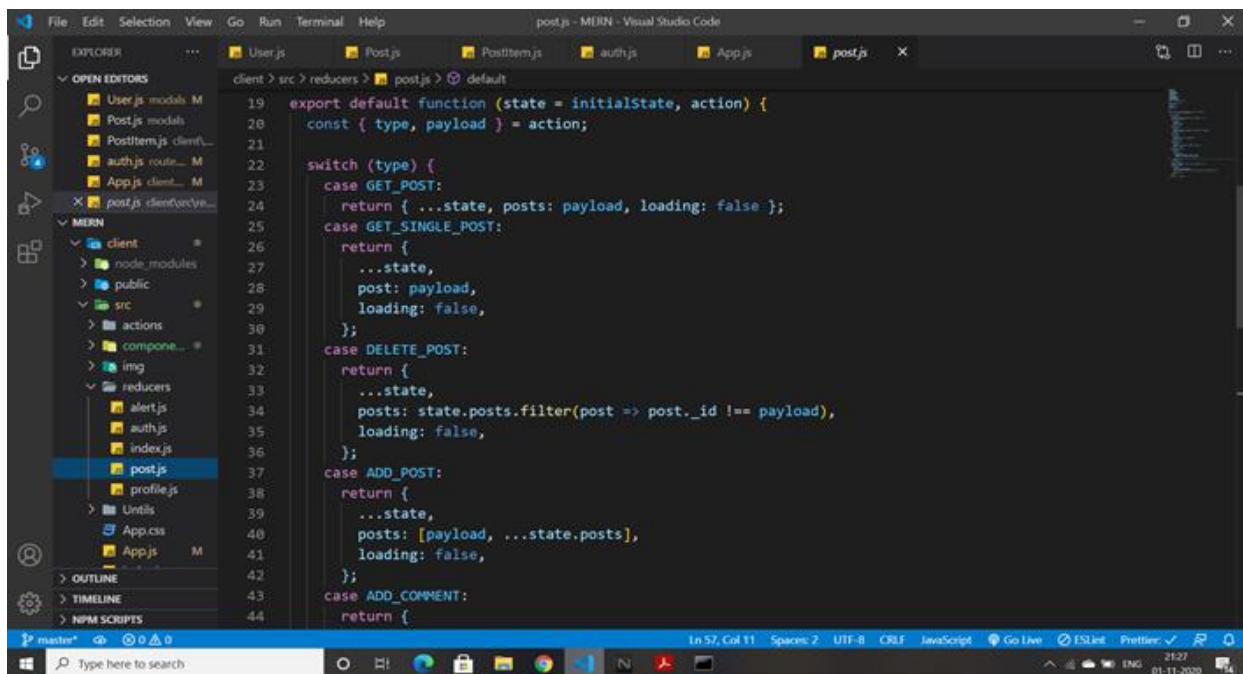
4. Authentication Reducer



The screenshot shows the Visual Studio Code editor with the file explorer on the left. The file explorer shows a project structure with folders like 'client', 'node_modules', 'public', 'src', 'actions', 'components', 'img', 'reducers', 'utils', 'App.css', and 'App.js'. The 'reducers' folder is expanded, and 'auth.js' is selected. The main editor displays the code for the authentication reducer in 'auth.js'.

```
export default function (state = initialState, action) {
  const { type, payload } = action;
  switch (type) {
    case USER_LOADED:
      return {
        ...state,
        isAuthenticated: true,
        loading: false,
        user: payload,
      };
    case REGISTER_SUCCESS:
    case LOGIN_SUCCESS:
      localStorage.setItem('token', payload.token);
      return {
        ...state,
        ...payload,
        isAuthenticated: true,
        loading: false,
      };
    case REGISTER_FAIL:
    case AUTH_ERROR:
    case LOGIN_FAIL:
    case LOGOUT:
    case ACCOUNT_DELETED:
      localStorage.removeItem('token');
  }
}
```

5. Post Reducer

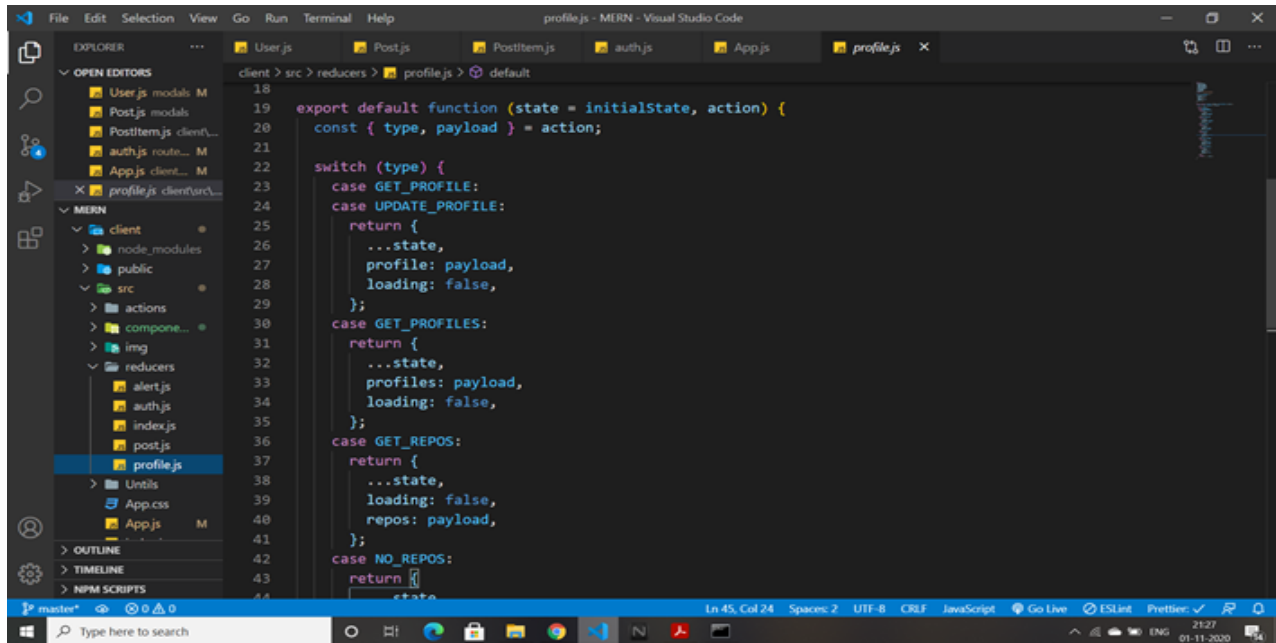


The screenshot shows the Visual Studio Code editor with the file explorer on the left. The file explorer shows a project structure with folders like 'client', 'node_modules', 'public', 'src', 'actions', 'components', 'img', 'reducers', 'utils', 'App.css', and 'App.js'. The 'reducers' folder is expanded, and 'post.js' is selected. The main editor displays the code for the post reducer in 'post.js'.

```
export default function (state = initialState, action) {
  const { type, payload } = action;
  switch (type) {
    case GET_POST:
      return { ...state, posts: payload, loading: false };
    case GET_SINGLE_POST:
      return {
        ...state,
        post: payload,
        loading: false,
      };
    case DELETE_POST:
      return {
        ...state,
        posts: state.posts.filter(post => post._id !== payload),
        loading: false,
      };
    case ADD_POST:
      return {
        ...state,
        posts: [payload, ...state.posts],
        loading: false,
      };
    case ADD_COMMENT:
      return {

```

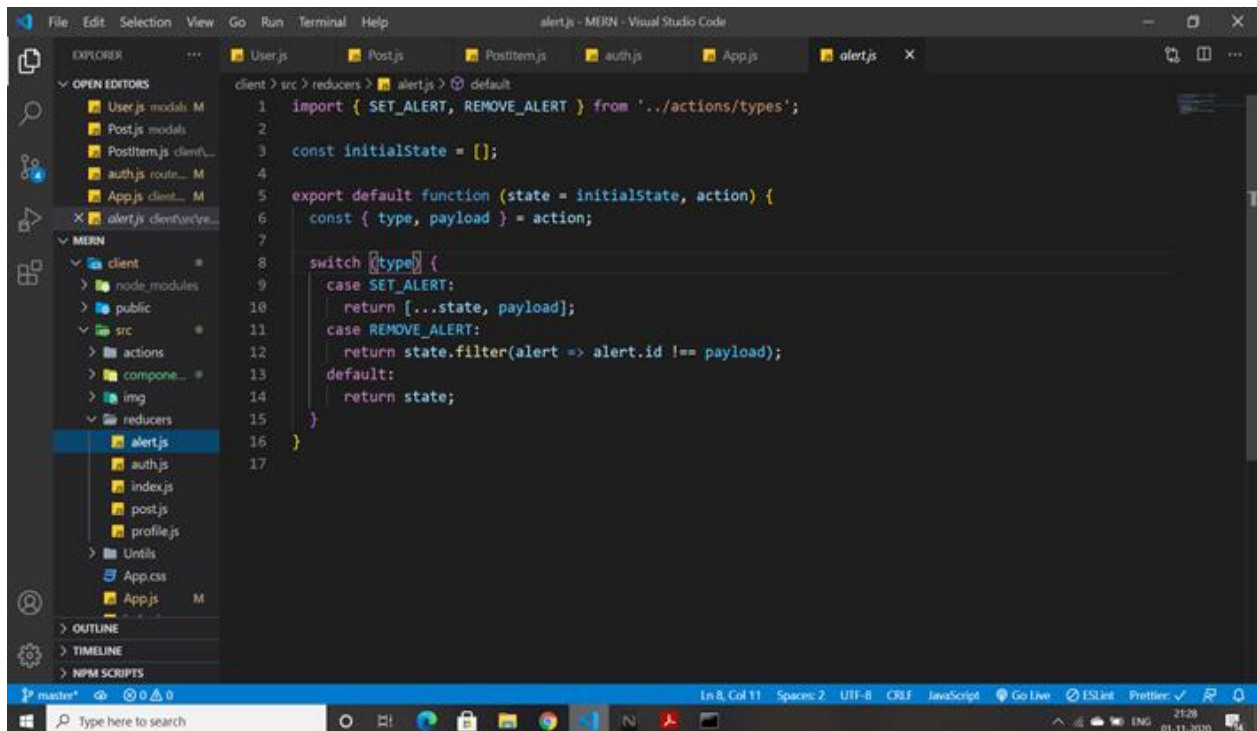

6. Profile Reducer



The screenshot shows the Visual Studio Code editor with the file explorer on the left. The file explorer shows the project structure with folders like 'client', 'node_modules', 'public', 'src', 'actions', 'components', 'img', 'reducers', 'utils', and 'App.js'. The 'reducers' folder is expanded, and 'profile.js' is selected. The main editor shows the code for the 'profile.js' reducer. The code defines a default function that takes state and action as arguments. It uses a switch statement to handle different action types: GET_PROFILE, UPDATE_PROFILE, GET_PROFILES, GET_REPOS, and NO_REPOS. Each case returns a new state object with the appropriate updates.

```
18 export default function (state = initialState, action) {
19   const { type, payload } = action;
20
21   switch (type) {
22     case GET_PROFILE:
23     case UPDATE_PROFILE:
24       return {
25         ...state,
26         profile: payload,
27         loading: false,
28       };
29     case GET_PROFILES:
30       return {
31         ...state,
32         profiles: payload,
33         loading: false,
34       };
35     case GET_REPOS:
36       return {
37         ...state,
38         loading: false,
39         repos: payload,
40       };
41     case NO_REPOS:
42       return {
43         ...state,
44         loading: false,
45       };
46   }
47 }
```

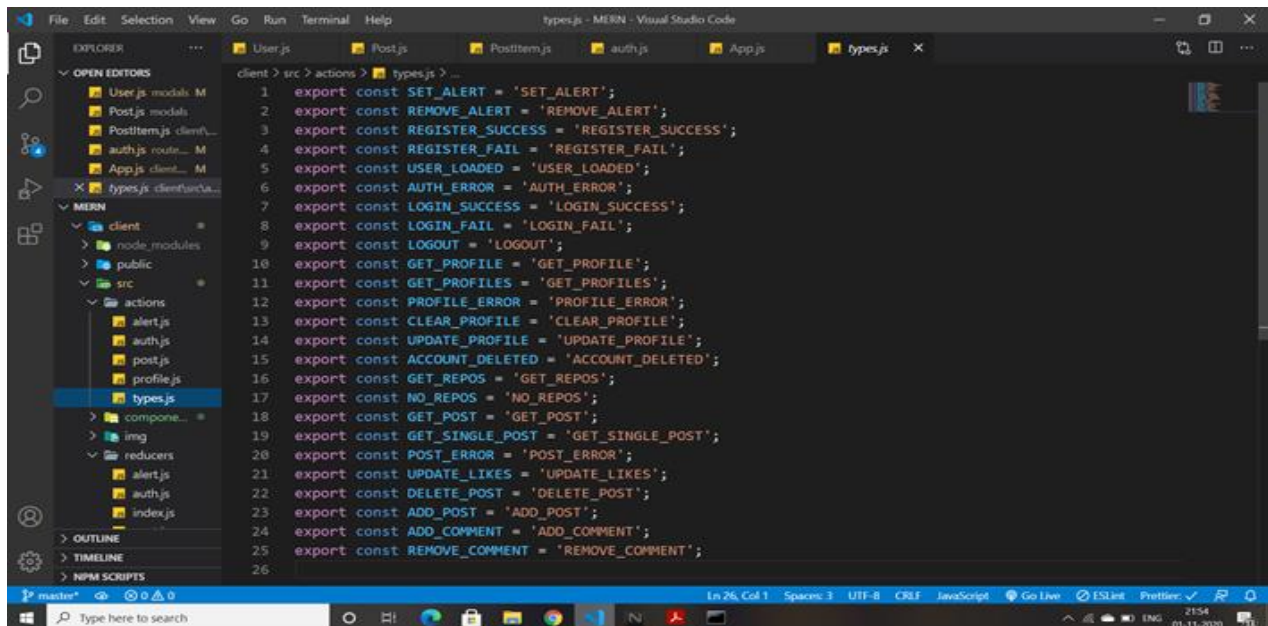
7. Alert Reducer



The screenshot shows the Visual Studio Code editor with the file explorer on the left. The file explorer shows the project structure with folders like 'client', 'node_modules', 'public', 'src', 'actions', 'components', 'img', 'reducers', 'utils', and 'App.js'. The 'reducers' folder is expanded, and 'alert.js' is selected. The main editor shows the code for the 'alert.js' reducer. The code defines a default function that takes state and action as arguments. It uses a switch statement to handle different action types: SET_ALERT and REMOVE_ALERT. Each case returns a new state object with the appropriate updates.

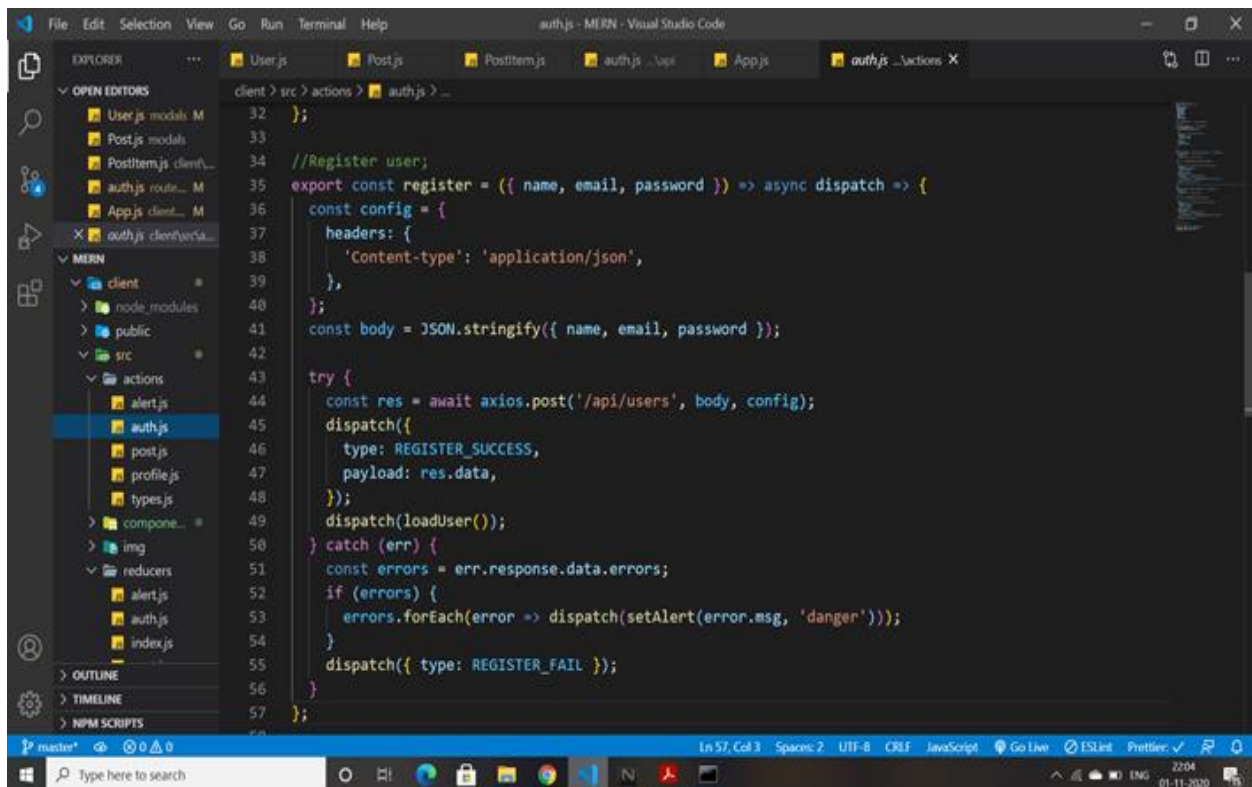
```
1 import { SET_ALERT, REMOVE_ALERT } from '../actions/types';
2
3 const initialState = [];
4
5 export default function (state = initialState, action) {
6   const { type, payload } = action;
7
8   switch (type) {
9     case SET_ALERT:
10       return [...state, payload];
11     case REMOVE_ALERT:
12       return state.filter(alert => alert.id !== payload);
13     default:
14       return state;
15   }
16 }
17
```

8. Action types



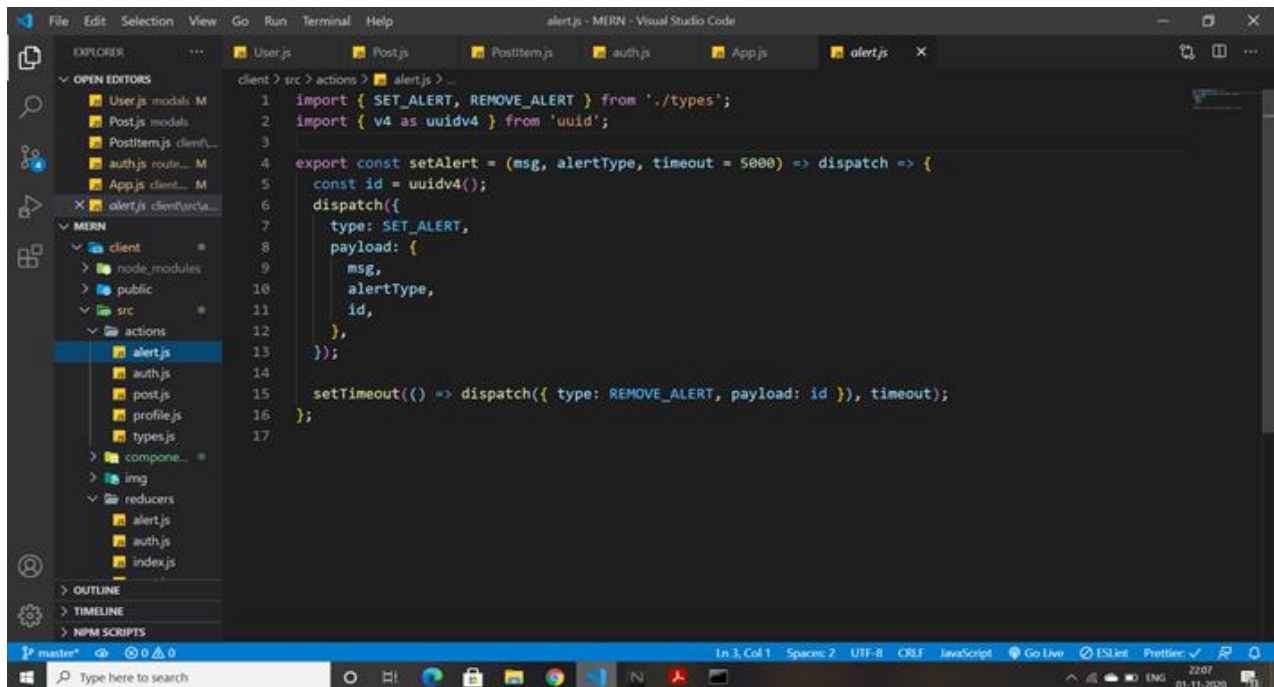
```
1 export const SET_ALERT = 'SET_ALERT';
2 export const REMOVE_ALERT = 'REMOVE_ALERT';
3 export const REGISTER_SUCCESS = 'REGISTER_SUCCESS';
4 export const REGISTER_FAIL = 'REGISTER_FAIL';
5 export const USER_LOADED = 'USER_LOADED';
6 export const AUTH_ERROR = 'AUTH_ERROR';
7 export const LOGIN_SUCCESS = 'LOGIN_SUCCESS';
8 export const LOGIN_FAIL = 'LOGIN_FAIL';
9 export const LOGOUT = 'LOGOUT';
10 export const GET_PROFILE = 'GET_PROFILE';
11 export const GET_PROFILES = 'GET_PROFILES';
12 export const PROFILE_ERROR = 'PROFILE_ERROR';
13 export const CLEAR_PROFILE = 'CLEAR_PROFILE';
14 export const UPDATE_PROFILE = 'UPDATE_PROFILE';
15 export const ACCOUNT_DELETED = 'ACCOUNT_DELETED';
16 export const GET_REPOS = 'GET_REPOS';
17 export const NO_REPOS = 'NO_REPOS';
18 export const GET_POST = 'GET_POST';
19 export const GET_SINGLE_POST = 'GET_SINGLE_POST';
20 export const POST_ERROR = 'POST_ERROR';
21 export const UPDATE_LIKES = 'UPDATE_LIKES';
22 export const DELETE_POST = 'DELETE_POST';
23 export const ADD_POST = 'ADD_POST';
24 export const ADD_COMMENT = 'ADD_COMMENT';
25 export const REMOVE_COMMENT = 'REMOVE_COMMENT';
26
```

9. Authentication (user Registration)

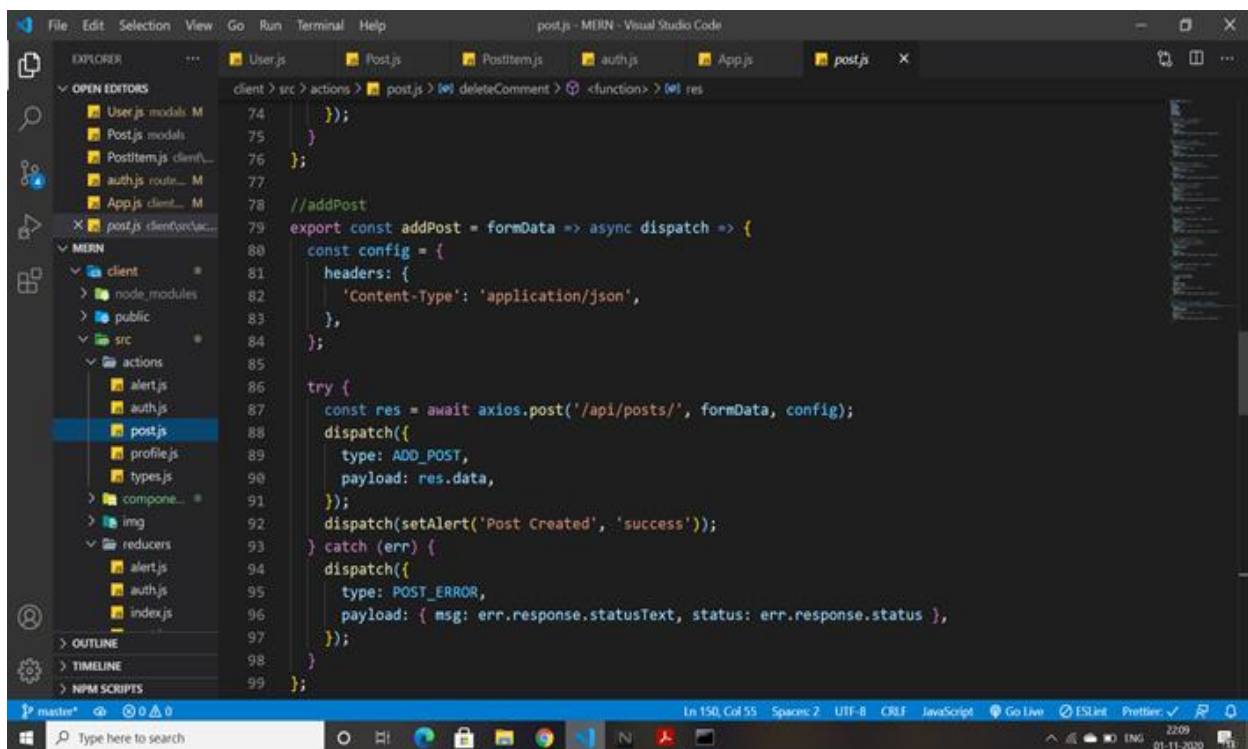


```
32 };
33
34 //Register user;
35 export const register = ({ name, email, password }) => async dispatch => {
36   const config = {
37     headers: {
38       'Content-type': 'application/json',
39     },
40   };
41   const body = JSON.stringify({ name, email, password });
42
43   try {
44     const res = await axios.post('/api/users', body, config);
45     dispatch({
46       type: REGISTER_SUCCESS,
47       payload: res.data,
48     });
49     dispatch(loadUser());
50   } catch (err) {
51     const errors = err.response.data.errors;
52     if (errors) {
53       errors.forEach(error => dispatch(setAlert(error.msg, 'danger')));
54     }
55     dispatch({ type: REGISTER_FAIL });
56   }
57 };
58
```

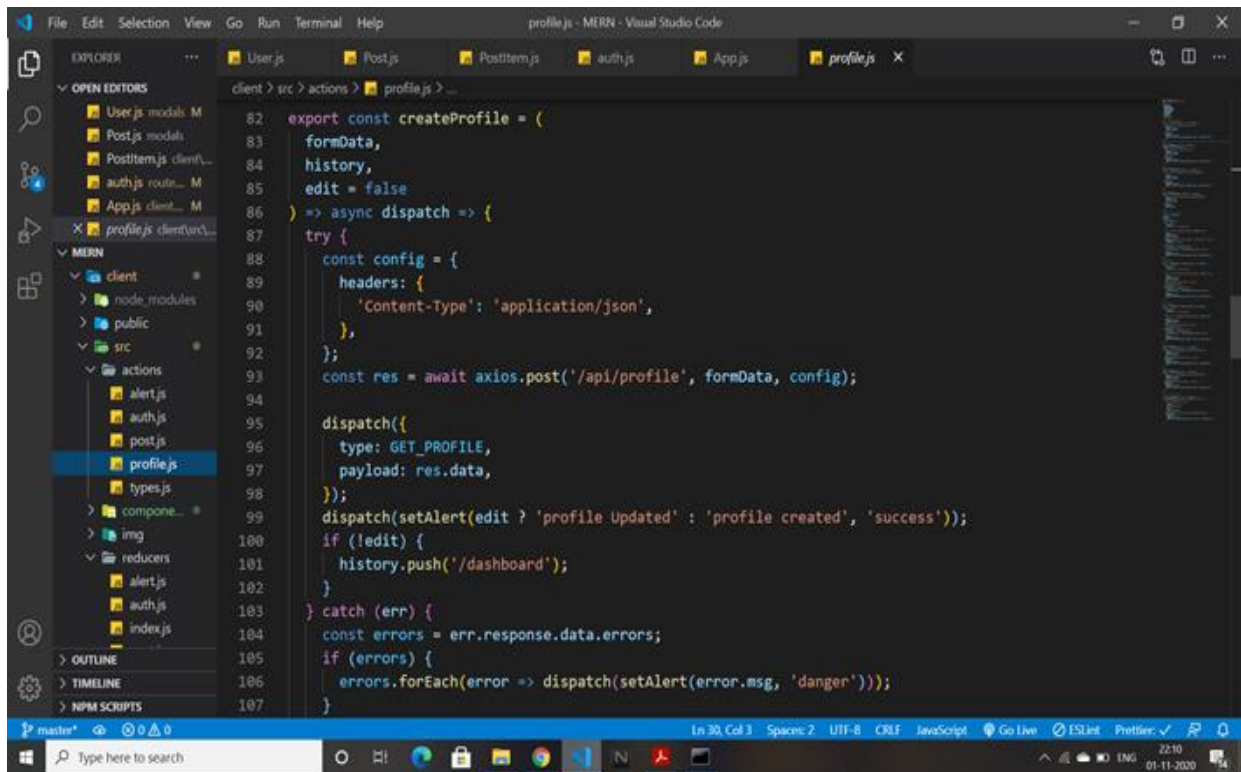
10. Alert Action



11. Add Post

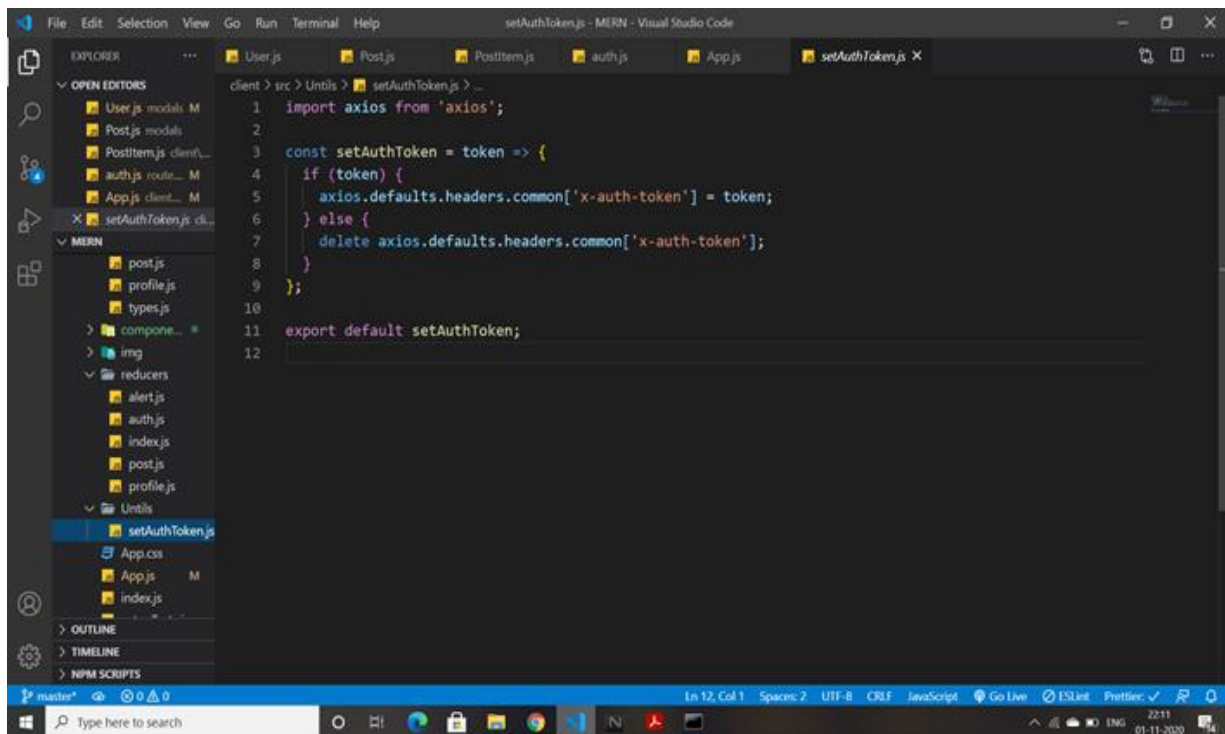


12. Create or Update profile



```
client > src > actions > profile.js > ...
82 export const createProfile = (
83   formData,
84   history,
85   edit = false
86 ) => async dispatch => {
87   try {
88     const config = {
89       headers: {
90         'Content-Type': 'application/json',
91       },
92     };
93     const res = await axios.post('/api/profile', formData, config);
94
95     dispatch({
96       type: GET_PROFILE,
97       payload: res.data,
98     });
99     dispatch(setAlert(edit ? 'profile Updated' : 'profile created', 'success'));
100     if (!edit) {
101       history.push('/dashboard');
102     }
103   } catch (err) {
104     const errors = err.response.data.errors;
105     if (errors) {
106       errors.forEach(error => dispatch(setAlert(error.msg, 'danger')));
107     }
108   }
109 }
```

13. Code to set JWT in local memory of the browser



```
client > src > Utils > setAuthToken.js > ...
1 import axios from 'axios';
2
3 const setAuthToken = token => {
4   if (token) {
5     axios.defaults.headers.common['x-auth-token'] = token;
6   } else {
7     delete axios.defaults.headers.common['x-auth-token'];
8   }
9 };
10
11 export default setAuthToken;
12
```

Chapter 4: Implementation

1. To run the project on local environment we used Concurrently , so that we can implement react and node js scripts together

The Scripts from Backend as follows:-

```
"scripts": {  
  "start": "node server",  
  "server": "nodemon server",  
  "client": "npm start --prefix client",  
  "dev": "concurrently \"npm run server\" \"npm run client\"",  
  "heroku-postbuild": "NPM_CONFIG_PRODUCTION=false npm install --prefix  
client && npm run build --prefix client"  
},
```

The Scripts from Front end are as follows:-

```
"scripts": {  
  "start": "react-scripts start",  
  "build": "react-scripts build",  
  "test": "react-scripts test",  
  "eject": "react-scripts eject"  
},
```

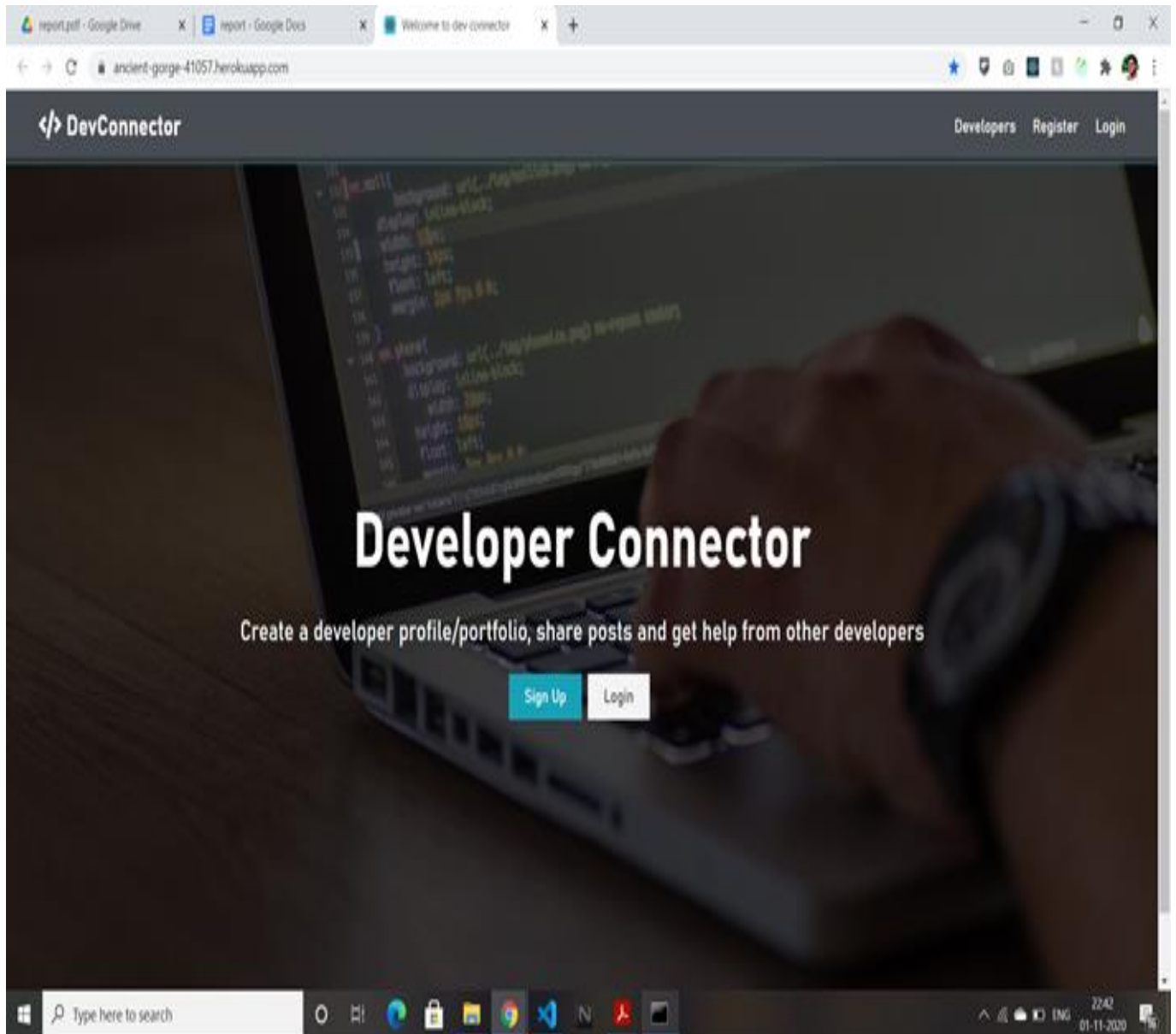
So as we are using concurrently to run two scripts from different files together we can write following command on the terminal >>npm run dev It will launch react app on localhost:3000 And express server on localhost:5000 simultaneously

After that we need to deploy it on Heroku using heroku CLI . For that we created a file named production.json , where we mentioned all the key variables so that it can be used as environmental variable setup by the heroku.

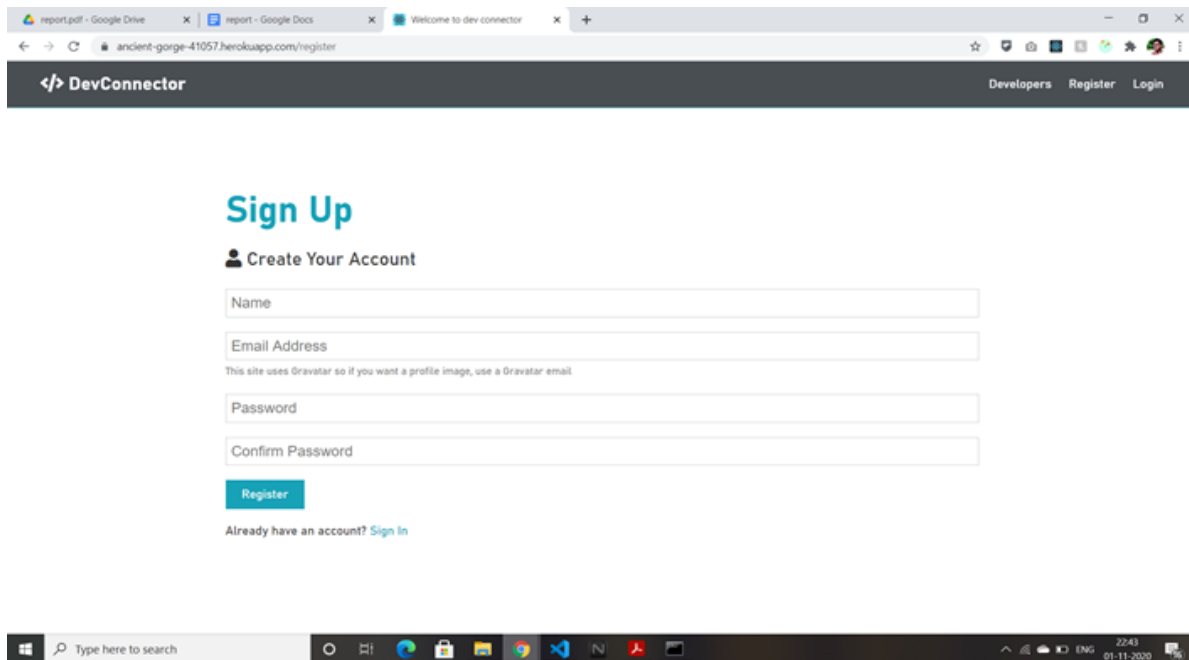
Chapter 5 : Final Results.

Following Screenshots of our web app will give the brief view of our final phase-1 finished product

1. Landing page



2. Sign up page (sign in page is also similar to this one)



The screenshot shows a web browser window with the URL `ancient-gorge-41057.herokuapp.com/register`. The page has a dark header with the DevConnector logo and navigation links for Developers, Register, and Login. The main content area is titled "Sign Up" and "Create Your Account". It contains a form with fields for Name, Email Address, Password, and Confirm Password. A note mentions Gravatar for profile images. A Register button is at the bottom, along with a link to Sign In for existing users. The Windows taskbar at the bottom shows the time as 22:43 on 01-11-2020.

report.pdf - Google Drive x report - Google Docs x Welcome to dev connector x +

← → ↻ ancient-gorge-41057.herokuapp.com/register ☆

</> DevConnector Developers Register Login

Sign Up

👤 Create Your Account

Name

Email Address

This site uses Gravatar so if you want a profile image, use a Gravatar email

Password

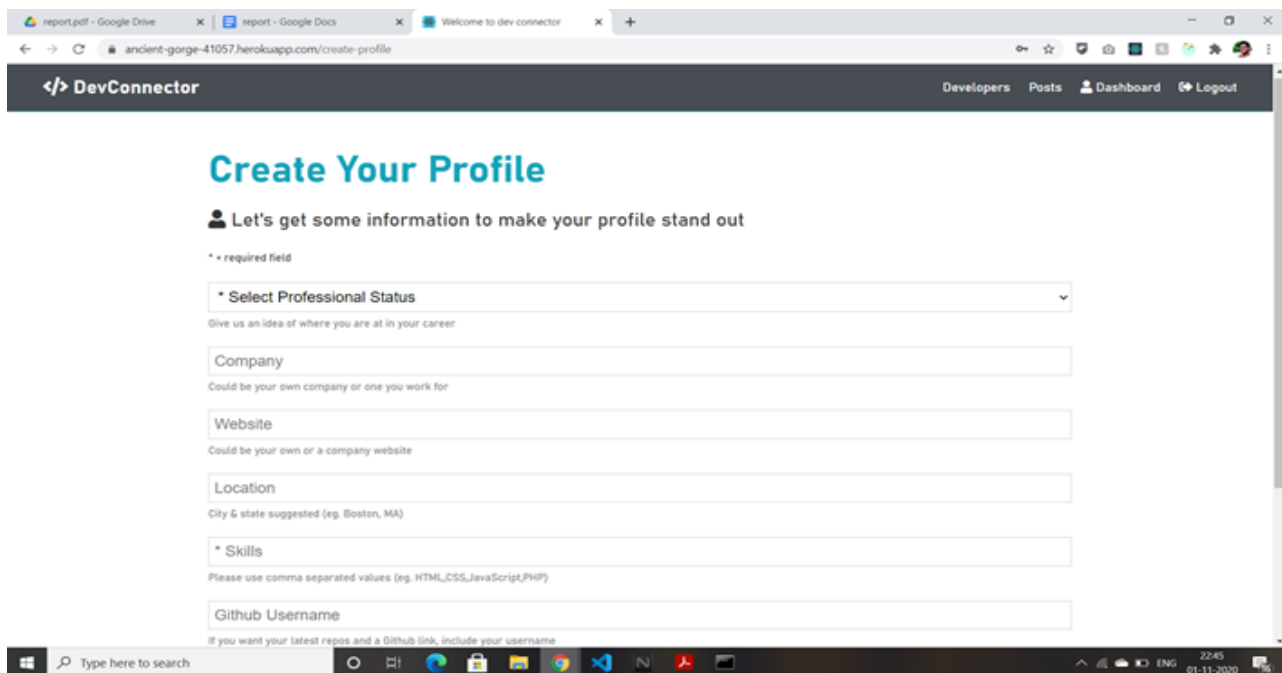
Confirm Password

Register

Already have an account? [Sign In](#)

Windows taskbar: Type here to search, 22:43, 01-11-2020

3. Profile creation



The screenshot shows a web browser window with the URL `ancient-gorge-41057.herokuapp.com/create-profile`. The page has a dark header with the DevConnector logo and navigation links for Developers, Posts, Dashboard, and Logout. The main content area is titled "Create Your Profile" and "Let's get some information to make your profile stand out". It contains a form with fields for Professional Status, Company, Website, Location, Skills, and Github Username. A note mentions Gravatar for profile images. A Register button is at the bottom, along with a link to Sign In for existing users. The Windows taskbar at the bottom shows the time as 22:43 on 01-11-2020.

report.pdf - Google Drive x report - Google Docs x Welcome to dev connector x +

← → ↻ ancient-gorge-41057.herokuapp.com/create-profile ☆

</> DevConnector Developers Posts Dashboard Logout

Create Your Profile

👤 Let's get some information to make your profile stand out

* required field

* Select Professional Status

Give us an idea of where you are at in your career

Company

Could be your own company or one you work for

Website

Could be your own or a company website

Location

City & state suggested (eg. Boston, MA)

* Skills

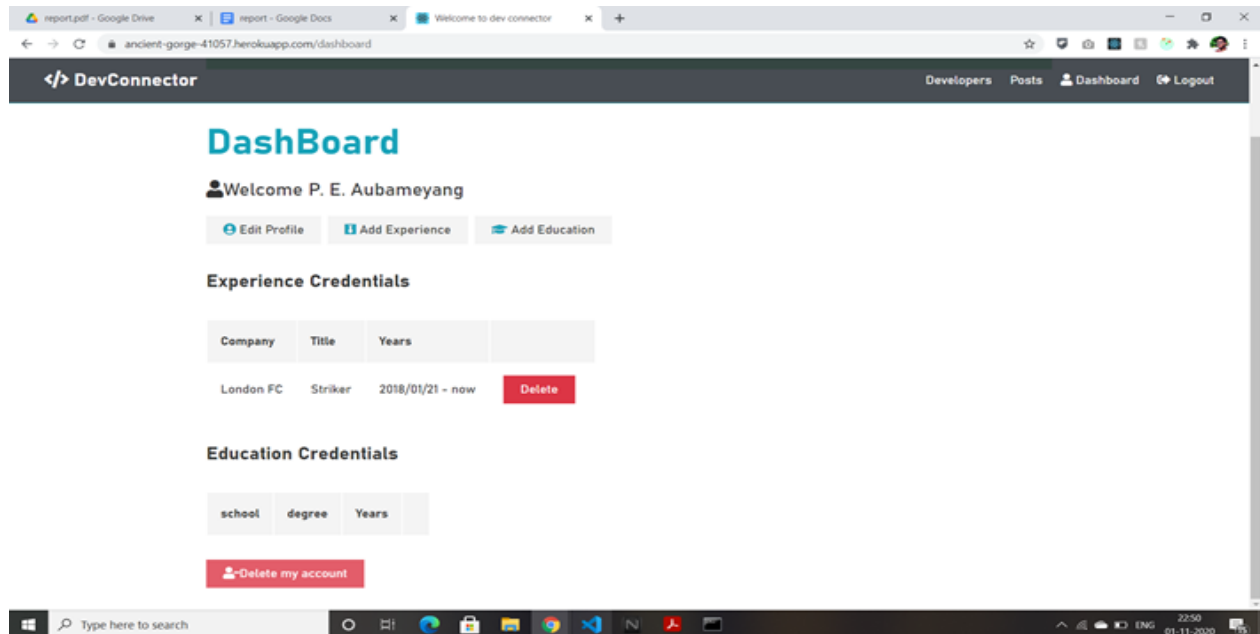
Please use comma separated values (eg. HTML,CSS,JavaScript,PHP)

Github Username

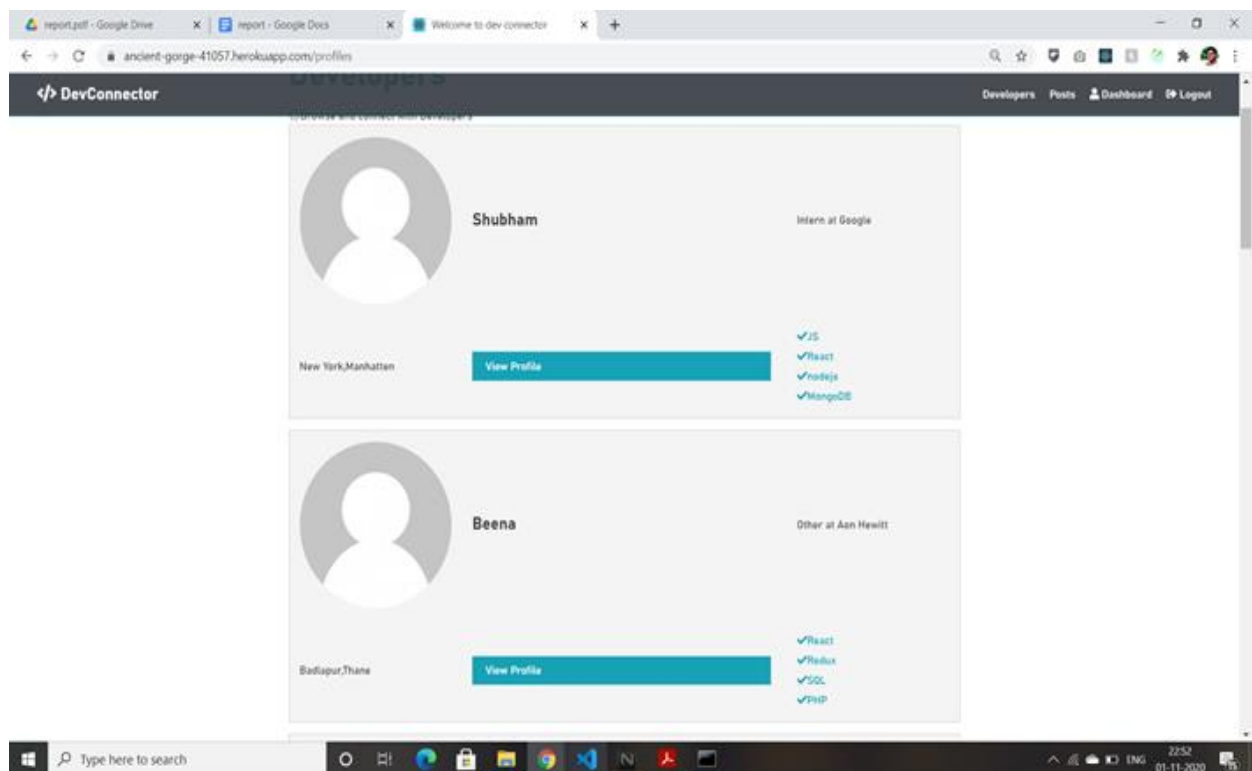
If you want your latest repos and a Github link, include your username

Windows taskbar: Type here to search, 22:43, 01-11-2020

4. Dashboard after profile creation



5. Browsing profile of other Developers



6. Recent Posts

The screenshot displays a web browser window with the following elements:

- Browser Tabs:** 'report.pdf - Google Drive', 'report - Google Docs', and 'Welcome to dev connector'.
- Address Bar:** 'ancient-gorge-41057.herokuapp.com/posts'.
- Application Header:** 'DevConnector' logo and navigation links: 'Developers', 'Posts', 'Dashboard', 'Logout'.
- Recent Posts List:**
 - Post 1:** Title 'I Love you 3000', User 'Shubham', Posted on 2019/11/11, 2 likes, Discussion button.
 - Post 2:** Title 'Tony, You can rest now', User 'Shubham', Posted on 2019/11/11, 2 likes, Discussion button.
 - Post 3:** Title 'Mr. Stark we won', User 'Shubham', Posted on 2019/11/11, 1 like, Discussion button.
 - Post 4:** Title 'I am Inevitable', User 'Shubham', Posted on 2019/11/11, 1 like, Discussion button.
 - Post 5:** Title 'I amIron-man', User 'Shubham', Posted on 2019/11/11, 2 likes, Discussion button.
- Windows Taskbar:** Search bar 'Type here to search' and various application icons.

Chapter 6: Conclusion

After Doing all the procedure , steps and Writing previously mentioned code , we were able to create a Full Stack web Development project using MERN stack.

The finished product is a fully functional social media site which helps fellow developers to connect to each other .

It serves all the purposes except direct messaging and Image uploading mechanism which can be easily coded with node package named Multer .

Chapter 7 : Bibliology

All the references that helped us to achieve our final product -

a. MongoDB atlas Server Documentation

<https://docs.mongodb.com/manual/>

b. React Documentation

<https://reactjs.org/docs/getting-started.html>

c. React-redux documentation

<https://react-redux.js.org/introduction/quick-start>

d. Moment js Documentation

<https://momentjs.com/docs/>

e. Node js documentation

<https://nodejs.org/en/docs/>

f. Express js Documentation

<https://expressjs.com/en/guide/routing.html>

h. Mongoose Documentation

<https://mongoosejs.com/docs/api.html>

i. Heroku CLI

<https://devcenter.heroku.com/categories/reference>