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

SRM INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be University u/s 3 of UGC Act, 1956)

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. :-

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.

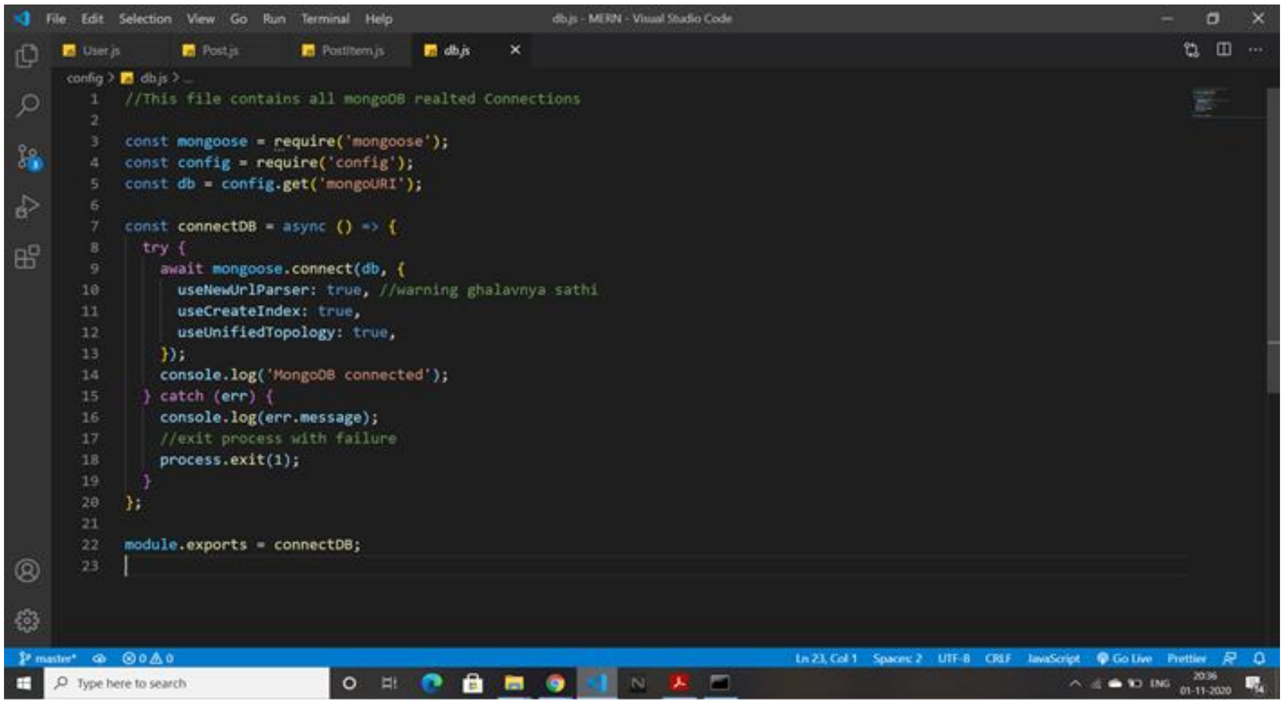
CHAPTER 1: INTRODUCTION

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

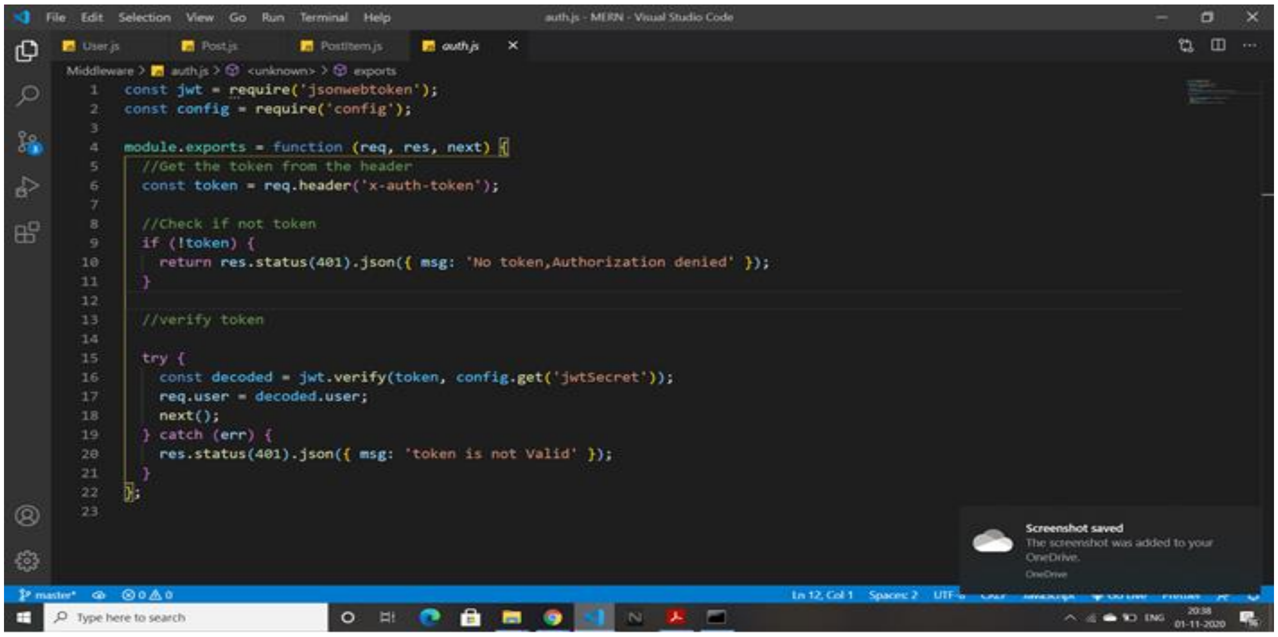
In this project we chose MERN stack as our development stack. So for the database we used a cloud setup named as mongo-atlas. For API creation we used Node.js , Express JS , Mongoose Schema whereas for encryption of the data we used Json Web token paired with Bcrypt encryption. In a single database we 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


File Edit Selection View Go Run Terminal Help
db.js - MERN - Visual Studio Code
User.js
Post.js
PostItem.js
db.js
config > db.js

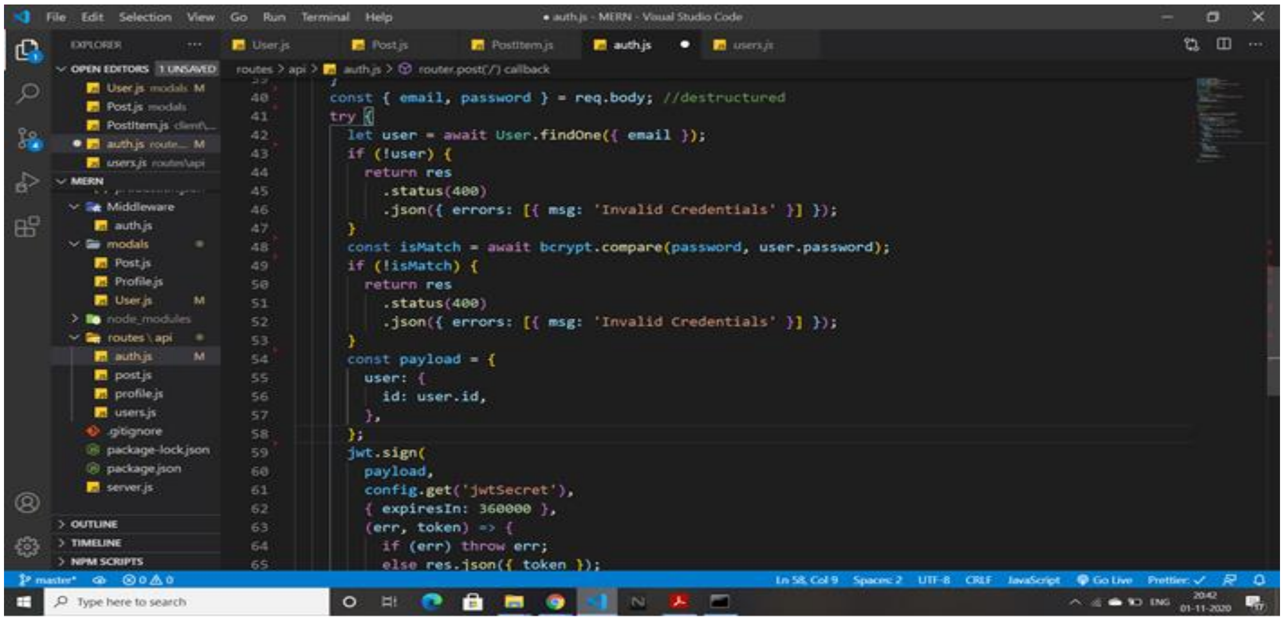
```
//This file contains all mongoDB realted Connections

const mongoose = require('mongoose');
const config = require('config');
const db = config.get('mongoURI');

const connectDB = async () => {
  try {
    await mongoose.connect(db, {
      useNewUrlParser: true, //warning ghalavnya sathi
      useCreateIndex: true,
      useUnifiedTopology: true,
    });
    console.log('MongoDB connected');
  } catch (err) {
    console.log(err.message);
    //exit process with failure
    process.exit(1);
  }
};

module.exports = connectDB;
```

2. Authentication middleware



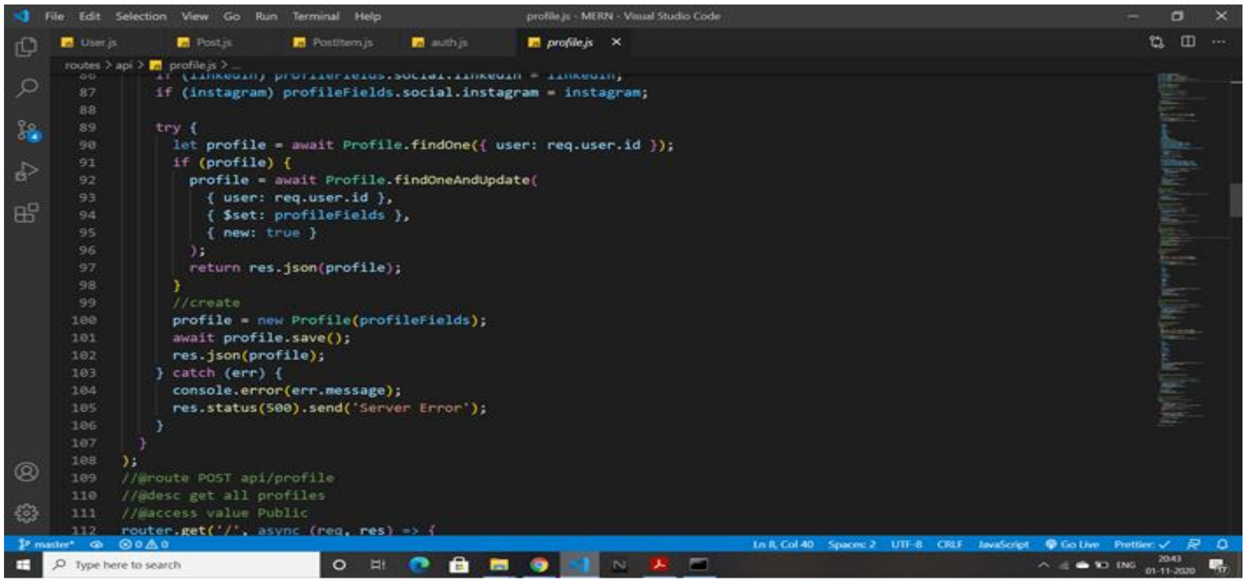
```
const jwt = require('jsonwebtoken');
const config = require('config');

module.exports = function (req, res, next) {
  //Get the token from the header
  const token = req.header('x-auth-token');

  //Check if not token
  if (!token) {
    return res.status(401).json({ msg: 'No token,Authorization denied' });
  }

  //verify token

  try {
    const decoded = jwt.verify(token, config.get('jwtSecret'));
    req.user = decoded.user;
    next();
  } catch (err) {
    res.status(401).json({ msg: 'token is not Valid' });
  }
};
```

3. Login Setup



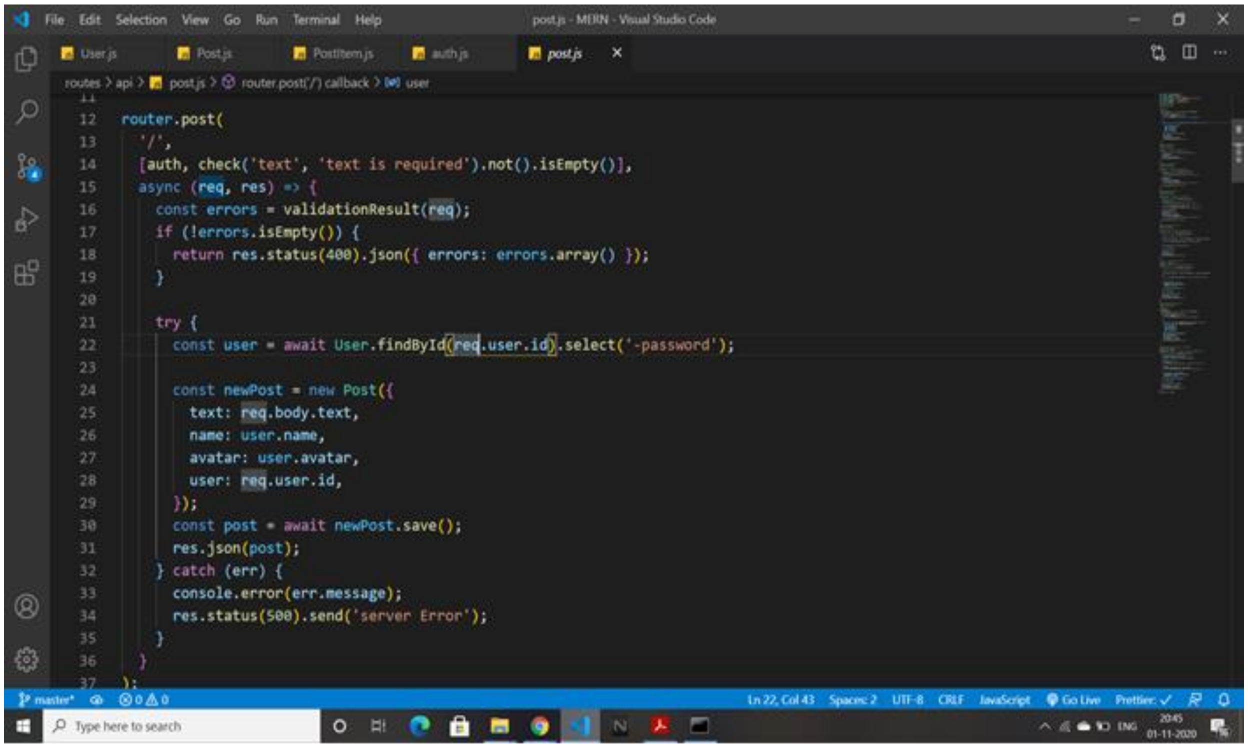
```
    const { email, password } = req.body; //destructured
    try {
      let user = await User.findOne({ email });
      if (!user) {
        return res
          .status(400)
          .json({ errors: [{ msg: 'Invalid Credentials' }] });
      }
      const isMatch = await bcrypt.compare(password, user.password);
      if (!isMatch) {
        return res
          .status(400)
          .json({ errors: [{ msg: 'Invalid Credentials' }] });
      }
      const payload = {
        user: {
          id: user.id,
        },
      };
      jwt.sign(
        payload,
        config.get('jwtSecret'),
        { expiresIn: 360000 },
        (err, token) => {
          if (err) throw err;
          else res.json({ token });
```

4. Profile Creation



```
    if (linkedin) profileFields.social.linkedin = linkedin;
    if (instagram) profileFields.social.instagram = instagram;

    try {
      let profile = await Profile.findOne({ user: req.user.id });
      if (profile) {
        profile = await Profile.findOneAndUpdate(
          { user: req.user.id },
          { $set: profileFields },
          { new: true }
        );
        return res.json(profile);
      }
      //create
      profile = new Profile(profileFields);
      await profile.save();
      res.json(profile);
    } catch (err) {
      console.error(err.message);
      res.status(500).send('Server Error');
    }
  }
);
//@route POST api/profile
//@desc get all profiles
//@access value Public
router.get('/', async (req, res) => {
```

5. Post Creation



```
router.post(
  '/',
  [auth, check('text', 'text is required').not().isEmpty()],
  async (req, res) => {
    const errors = validationResult(req);
    if (!errors.isEmpty()) {
      return res.status(400).json({ errors: errors.array() });
    }

    try {
      const user = await User.findById(req.user.id).select('-password');

      const newPost = new Post({
        text: req.body.text,
        name: user.name,
        avatar: user.avatar,
        user: req.user.id,
      });
      const post = await newPost.save();
      res.json(post);
    } catch (err) {
      console.error(err.message);
      res.status(500).send('server Error');
    }
  }
);
```

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)

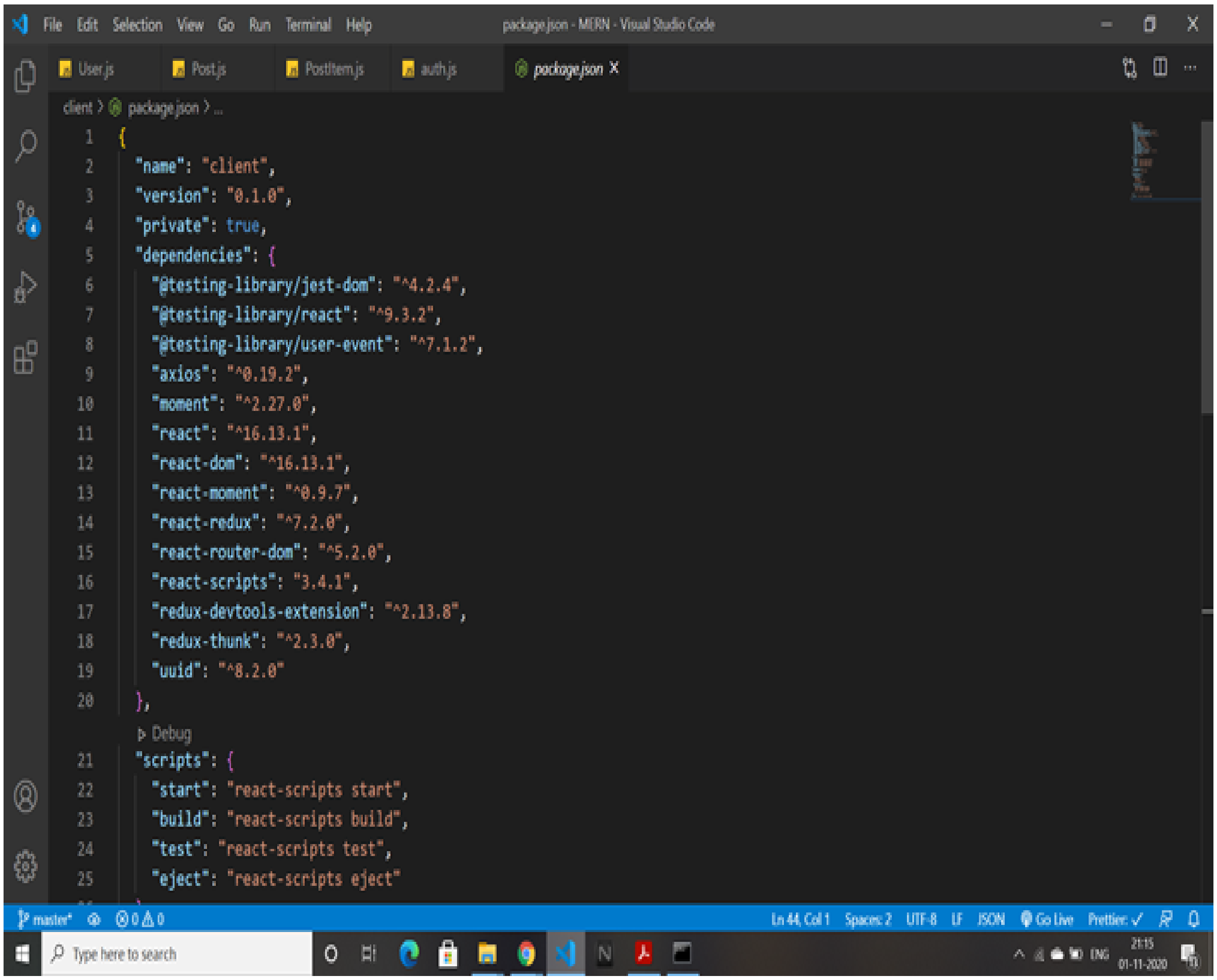
i. nodemon (devDependancy)

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



File Edit Selection View Go Run Terminal Help
package.json - MERN - Visual Studio Code
User.js
Post.js
PostItem.js
auth.js
package.json
client > package.json > ...
master* ⊗ 0 ⚠ 0
Ln 44, Col 1 Spaces: 2 UTF-8 LF JSON Go Live Prettier: ✓
Type here to search
21:15
01-11-2020

```
{
  "name": "client",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "@testing-library/jest-dom": "^4.2.4",
    "@testing-library/react": "^9.3.2",
    "@testing-library/user-event": "^7.1.2",
    "axios": "^0.19.2",
    "moment": "^2.27.0",
    "react": "^16.13.1",
    "react-dom": "^16.13.1",
    "react-moment": "^0.9.7",
    "react-redux": "^7.2.0",
    "react-router-dom": "^5.2.0",
    "react-scripts": "3.4.1",
    "redux-devtools-extension": "^2.13.8",
    "redux-thunk": "^2.3.0",
    "uuid": "^8.2.0"
  },
  ▷ Debug
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
```

Now let's examine our front end code .

We divided our react and redux files as follows

a. Actions

b. Components

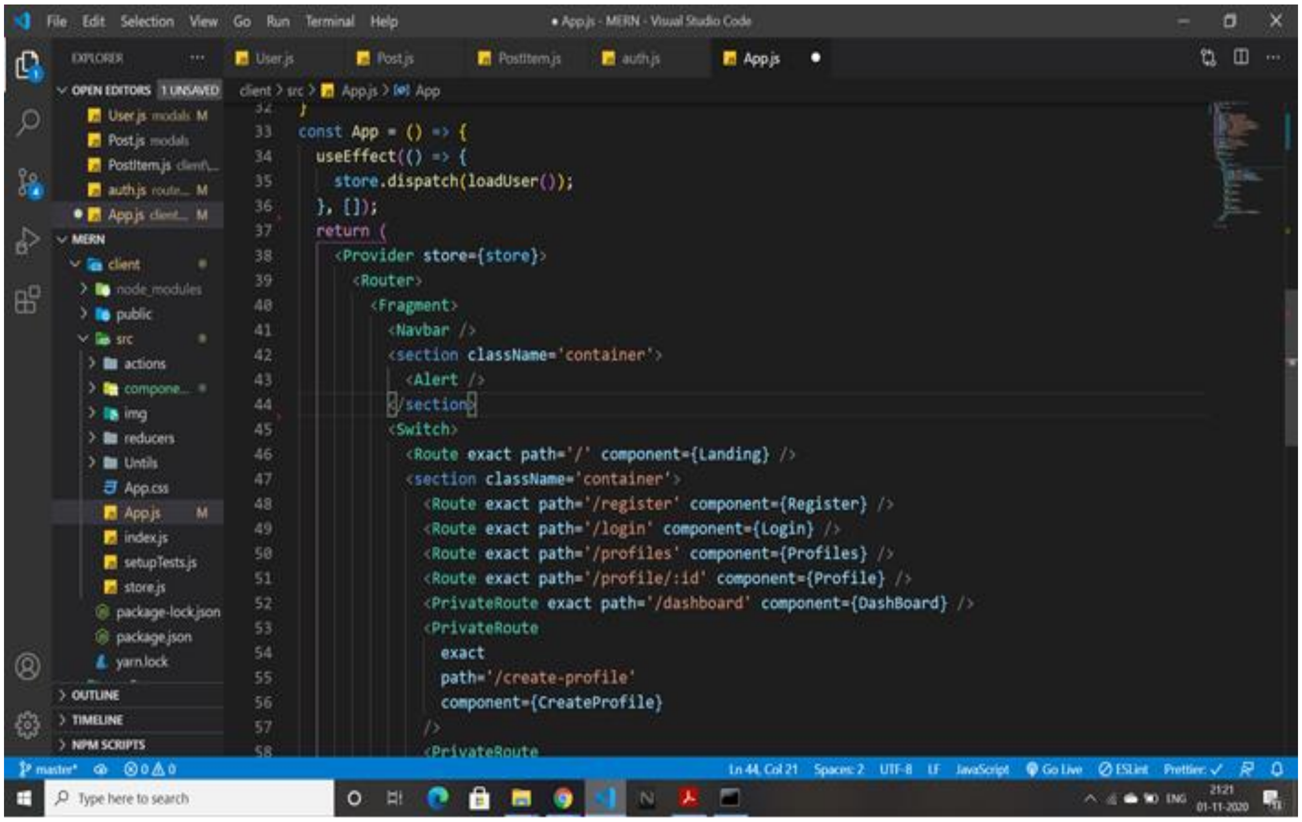
c. Reducers

d. Store

e. Utils

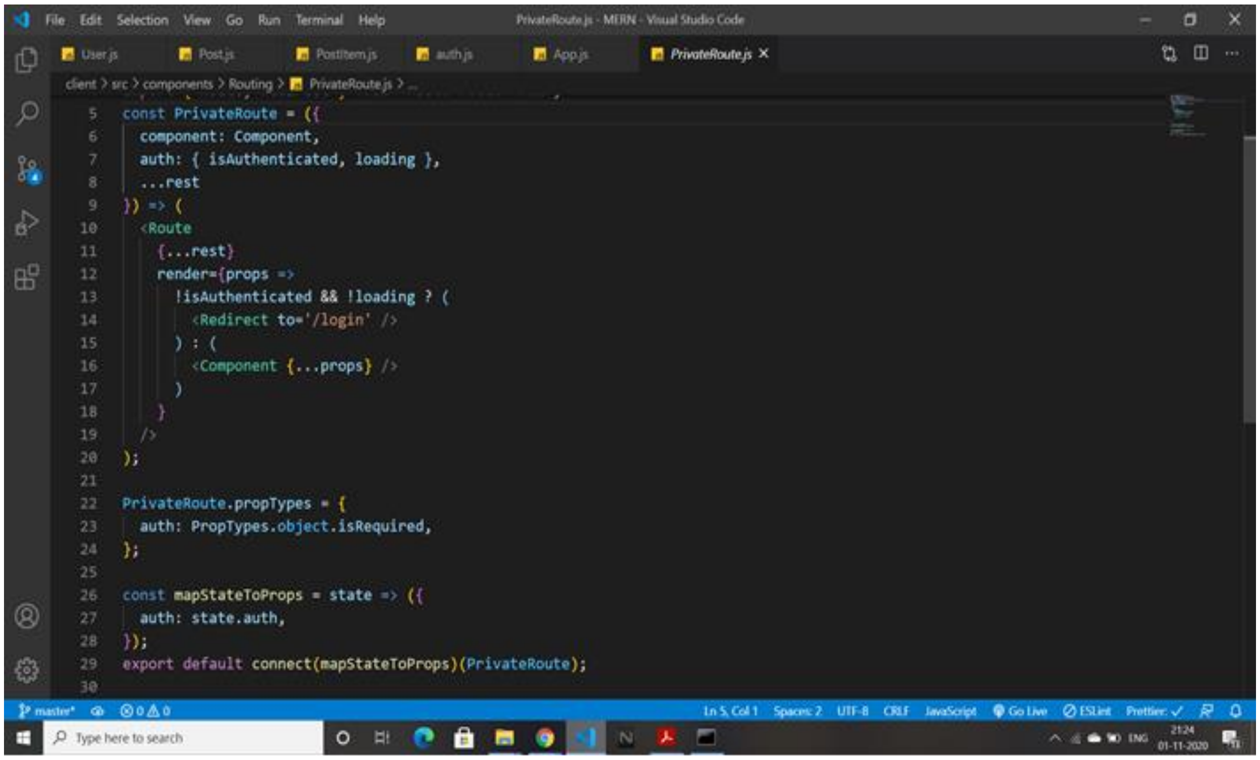
f. App.js

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



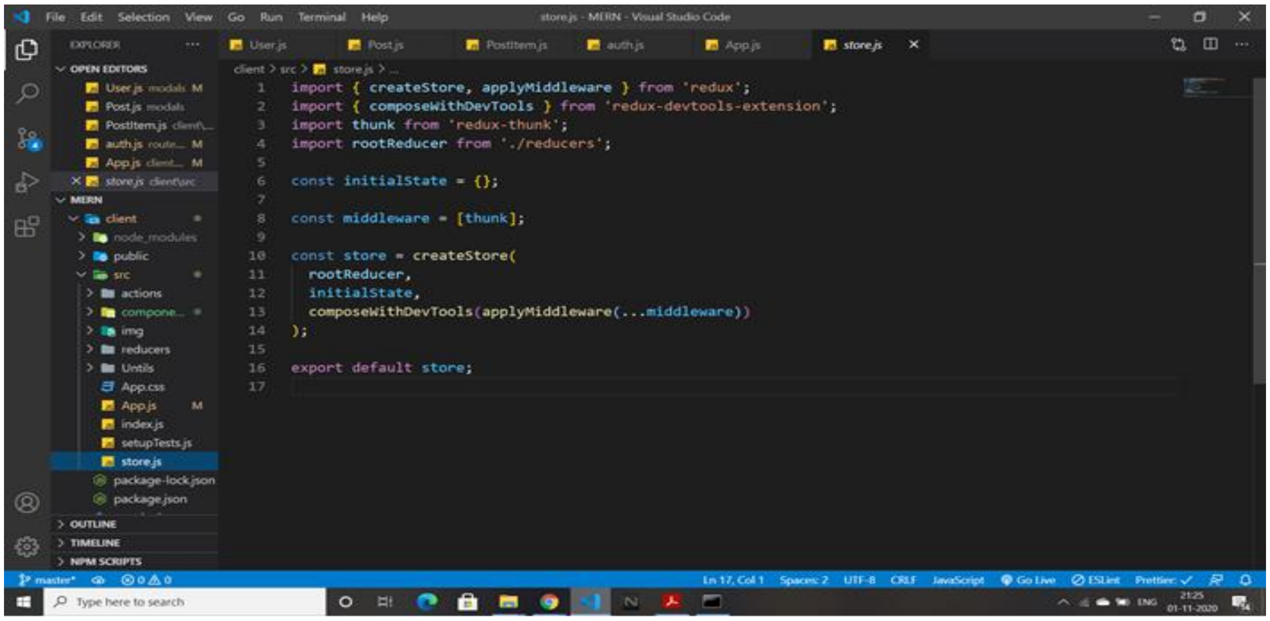
```
}
const App = () => {
  useEffect(() => {
    store.dispatch(loadUser());
  }, []);
  return (
    <Provider store={store}>
      <Router>
        <Fragment>
          <Navbar />
          <section className='container'>
            <Alert />
          </section>
          <Switch>
            <Route 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}
              />
              <PrivateRoute
```

2. **Private Route Setup, to validate JWT for Authentication**



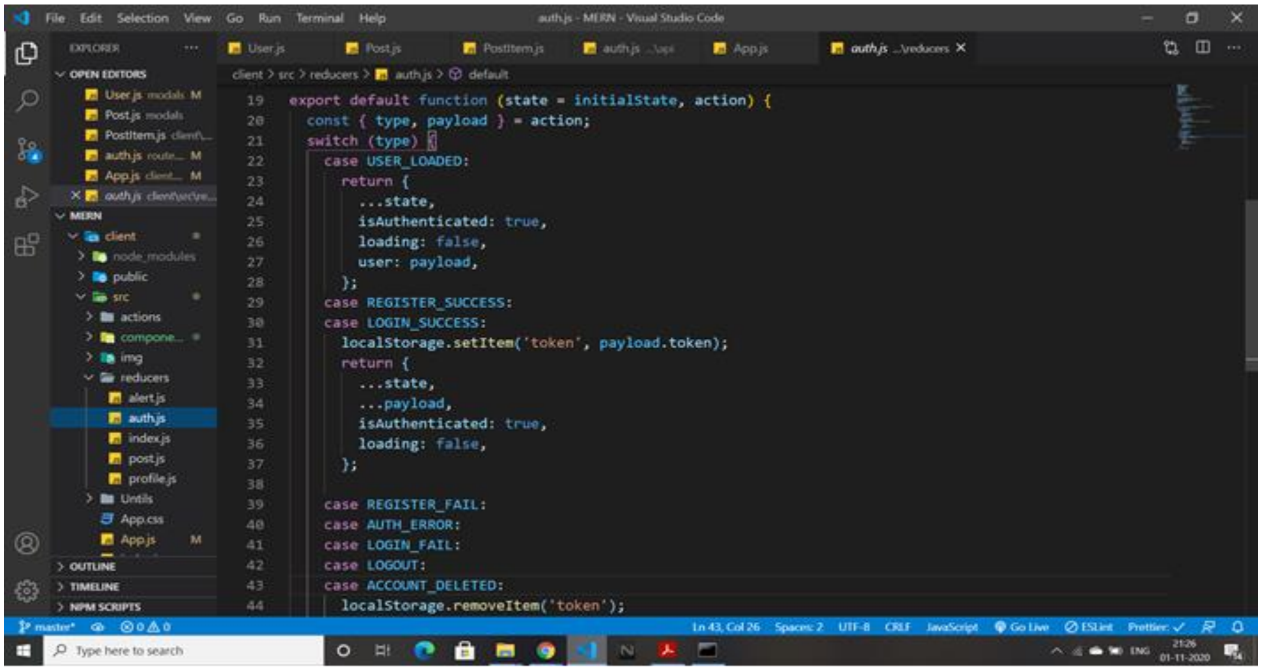
```
const PrivateRoute = ({
  component: Component,
  auth: { isAuthenticated, loading },
  ...rest
}) => (
  <Route
    {...rest}
    render={props =>
      !isAuthenticated && !loading ? (
        <Redirect to='/login' />
      ) : (
        <Component {...props} />
      )
    }
  />
);

PrivateRoute.propTypes = {
  auth: PropTypes.object.isRequired,
};

const mapStateToProps = state => ({
  auth: state.auth,
});
export default connect(mapStateToProps)(PrivateRoute);
```

3. Redux Store setup



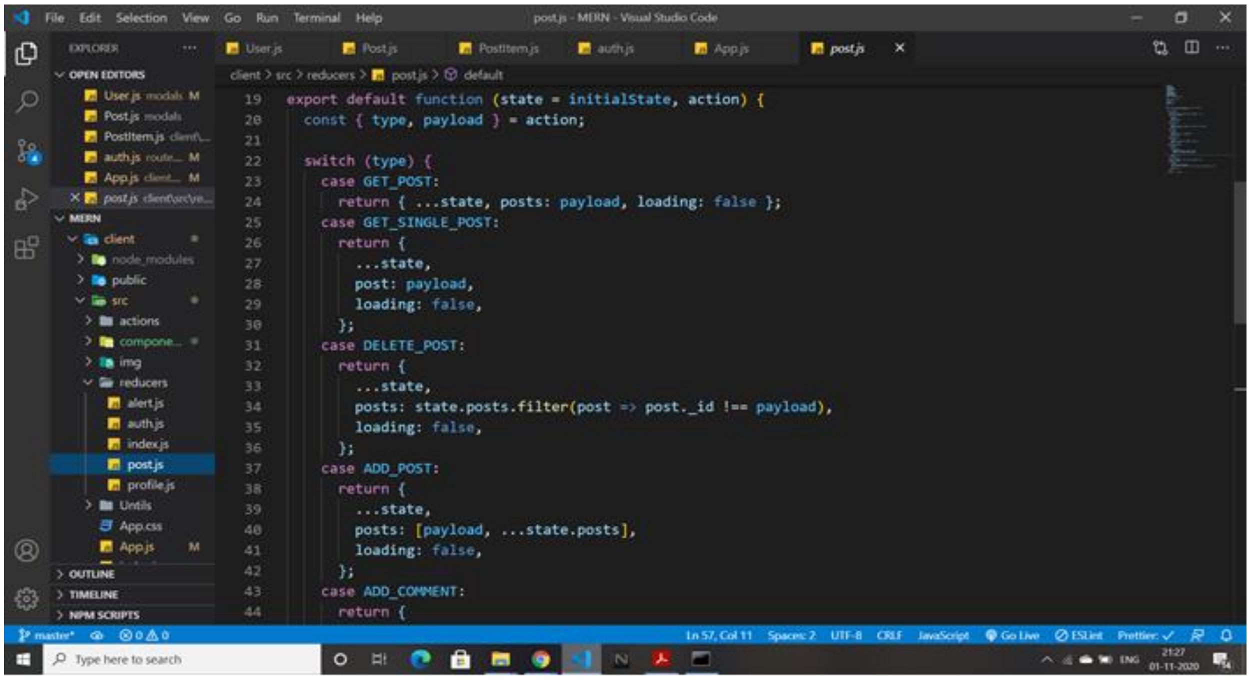
```
import { createStore, applyMiddleware } from 'redux';
import { composeWithDevTools } from 'redux-devtools-extension';
import thunk from 'redux-thunk';
import rootReducer from './reducers';

const initialState = {};

const middleware = [thunk];

const store = createStore(
  rootReducer,
  initialState,
  composeWithDevTools(applyMiddleware(...middleware))
);

export default store;
```

4. Authentication Reducer



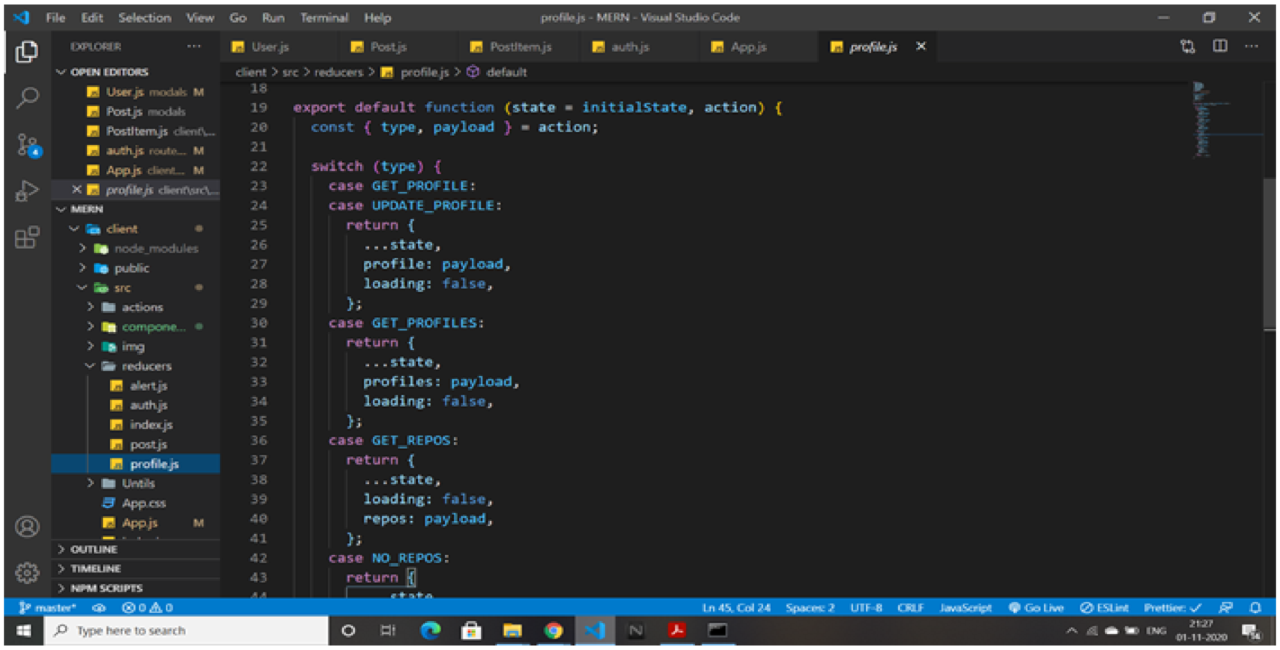
```
export default function (state = initialState, action) {
  const { type, payload } = action;
  switch (type) {
    case USER_LOADED:
      return {
        ...state,
        isAuthenticated: true,
        loading: false,
        user: payload,
      };
    case REGISTER_SUCCESS:
    case LOGIN_SUCCESS:
      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



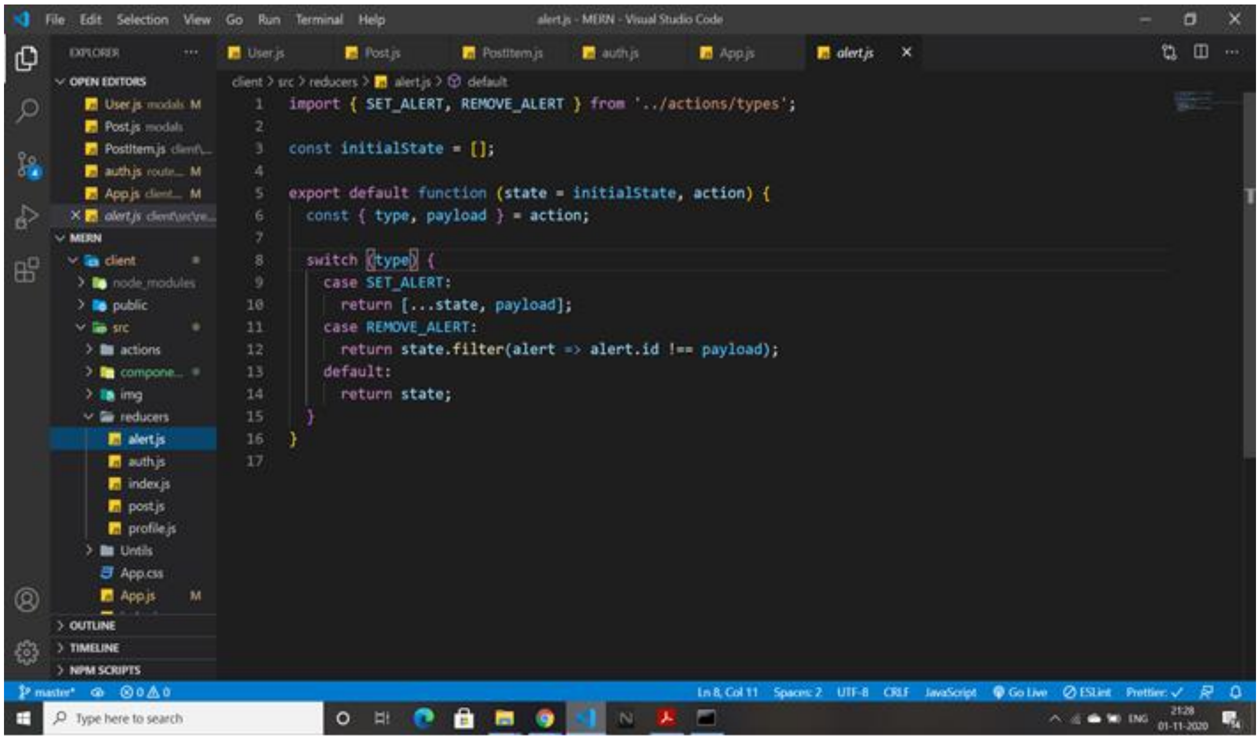
```
export default function (state = initialState, action) {
  const { type, payload } = action;

  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



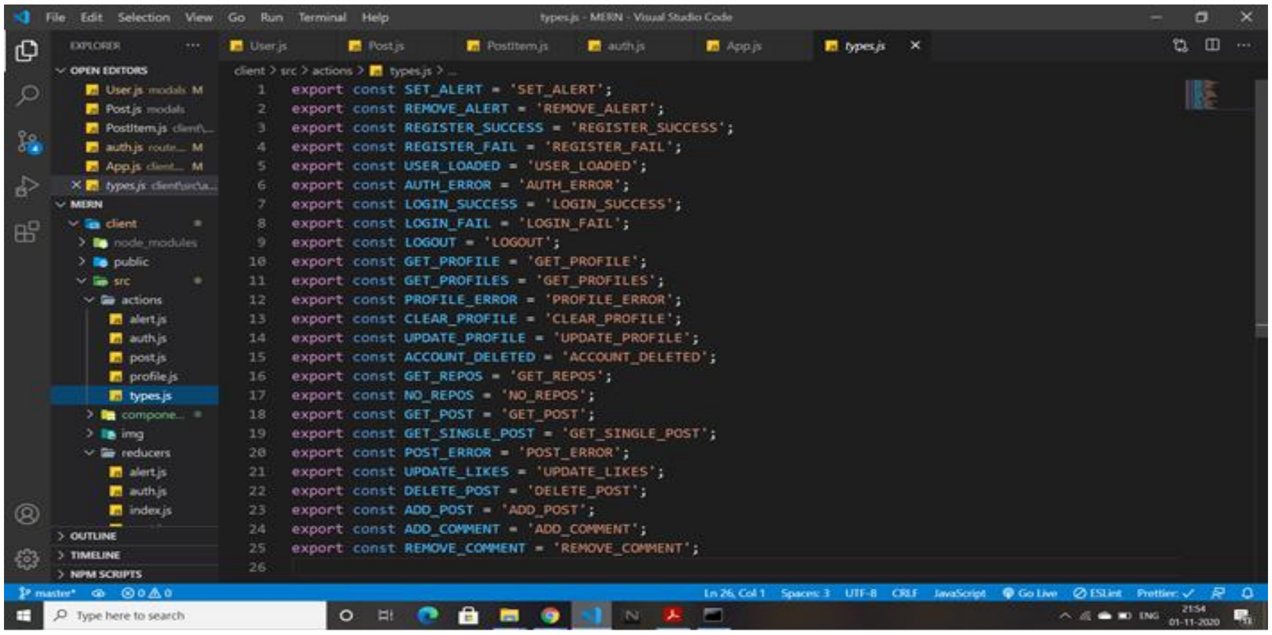
```
export default function (state = initialState, action) {
  const { type, payload } = action;

  switch (type) {
    case GET_PROFILE:
    case UPDATE_PROFILE:
      return {
        ...state,
        profile: payload,
        loading: false,
      };
    case GET_PROFILES:
      return {
        ...state,
        profiles: payload,
        loading: false,
      };
    case GET_REPOS:
      return {
        ...state,
        loading: false,
        repos: payload,
      };
    case NO_REPOS:
      return {
```

7. Alert Reducer



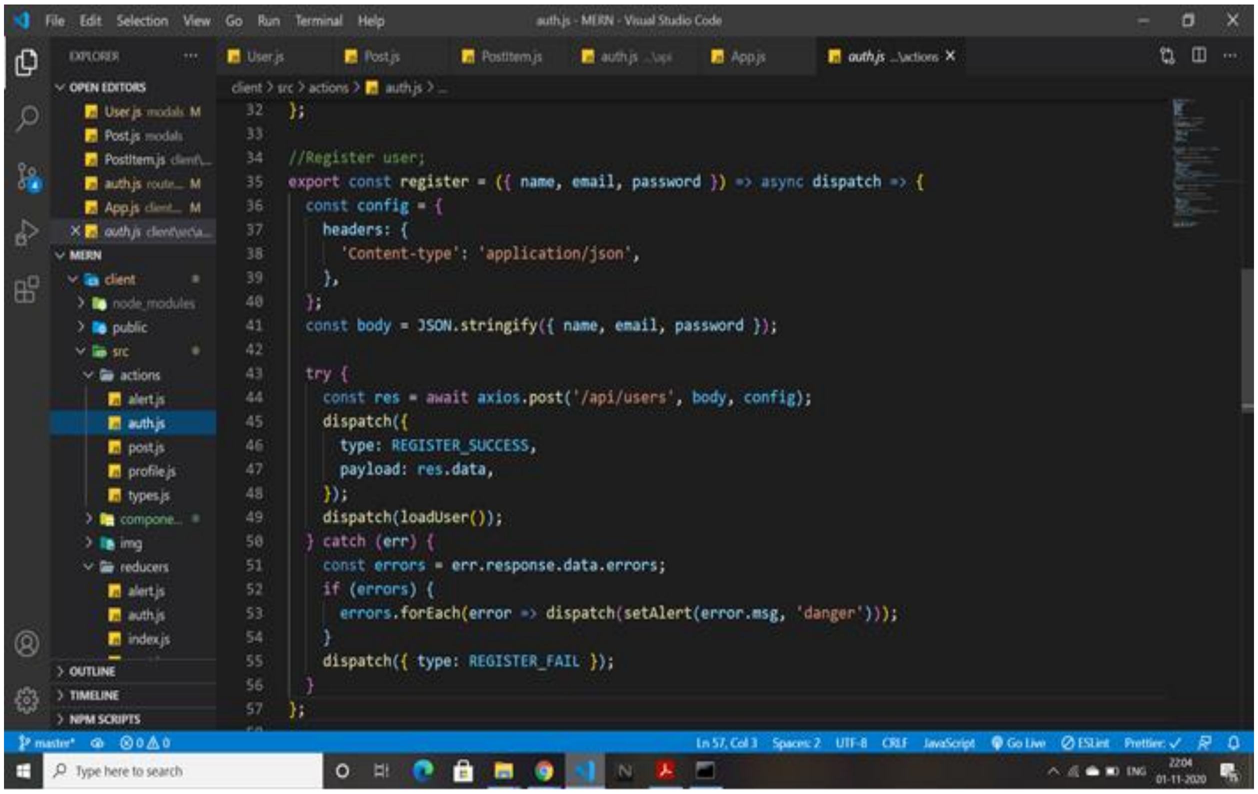
```
import { SET_ALERT, REMOVE_ALERT } from '../actions/types';

const initialState = [];

export default function (state = initialState, action) {
  const { type, payload } = action;

  switch (type) {
    case SET_ALERT:
      return [...state, payload];
    case REMOVE_ALERT:
      return state.filter(alert => alert.id !== payload);
    default:
      return state;
  }
}
```

8. Action types



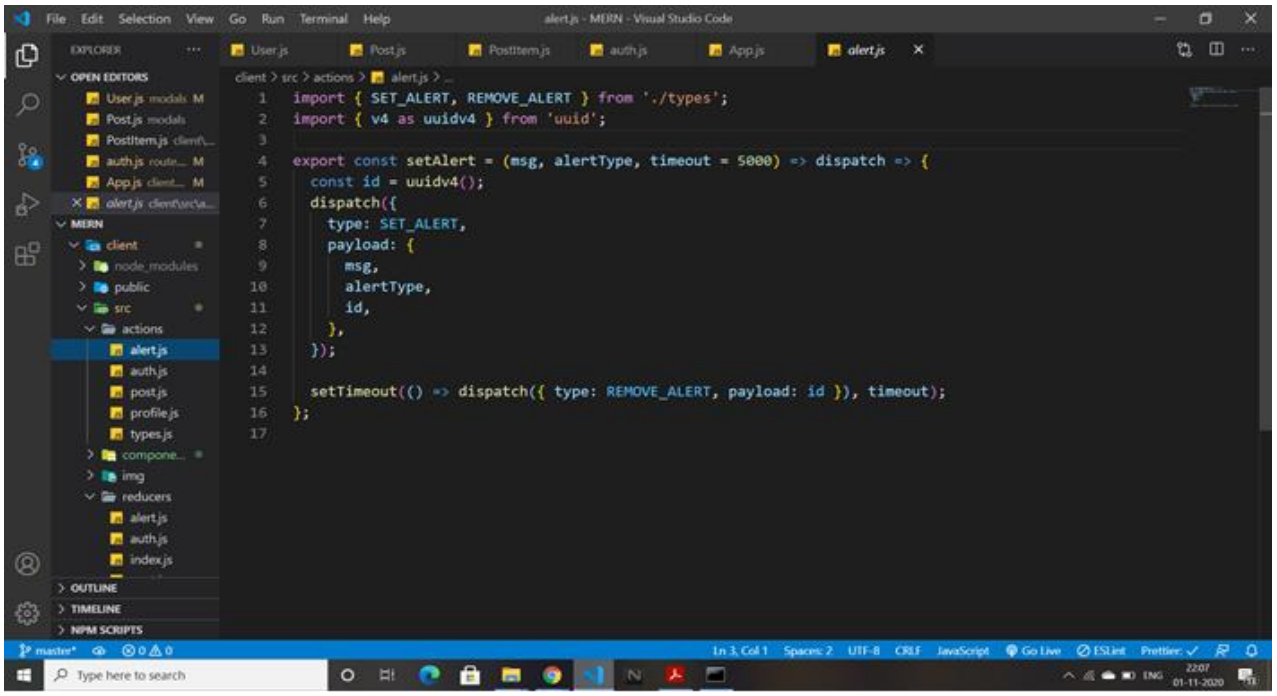
```
export const SET_ALERT = 'SET_ALERT';
export const REMOVE_ALERT = 'REMOVE_ALERT';
export const REGISTER_SUCCESS = 'REGISTER_SUCCESS';
export const REGISTER_FAIL = 'REGISTER_FAIL';
export const USER_LOADED = 'USER_LOADED';
export const AUTH_ERROR = 'AUTH_ERROR';
export const LOGIN_SUCCESS = 'LOGIN_SUCCESS';
export const LOGIN_FAIL = 'LOGIN_FAIL';
export const LOGOUT = 'LOGOUT';
export const GET_PROFILE = 'GET_PROFILE';
export const GET_PROFILES = 'GET_PROFILES';
export const PROFILE_ERROR = 'PROFILE_ERROR';
export const CLEAR_PROFILE = 'CLEAR_PROFILE';
export const UPDATE_PROFILE = 'UPDATE_PROFILE';
export const ACCOUNT_DELETED = 'ACCOUNT_DELETED';
export const GET_REPOS = 'GET_REPOS';
export const NO_REPOS = 'NO_REPOS';
export const GET_POST = 'GET_POST';
export const GET_SINGLE_POST = 'GET_SINGLE_POST';
export const POST_ERROR = 'POST_ERROR';
export const UPDATE_LIKES = 'UPDATE_LIKES';
export const DELETE_POST = 'DELETE_POST';
export const ADD_POST = 'ADD_POST';
export const ADD_COMMENT = 'ADD_COMMENT';
export const REMOVE_COMMENT = 'REMOVE_COMMENT';
```

9. Authentication (user Registration)



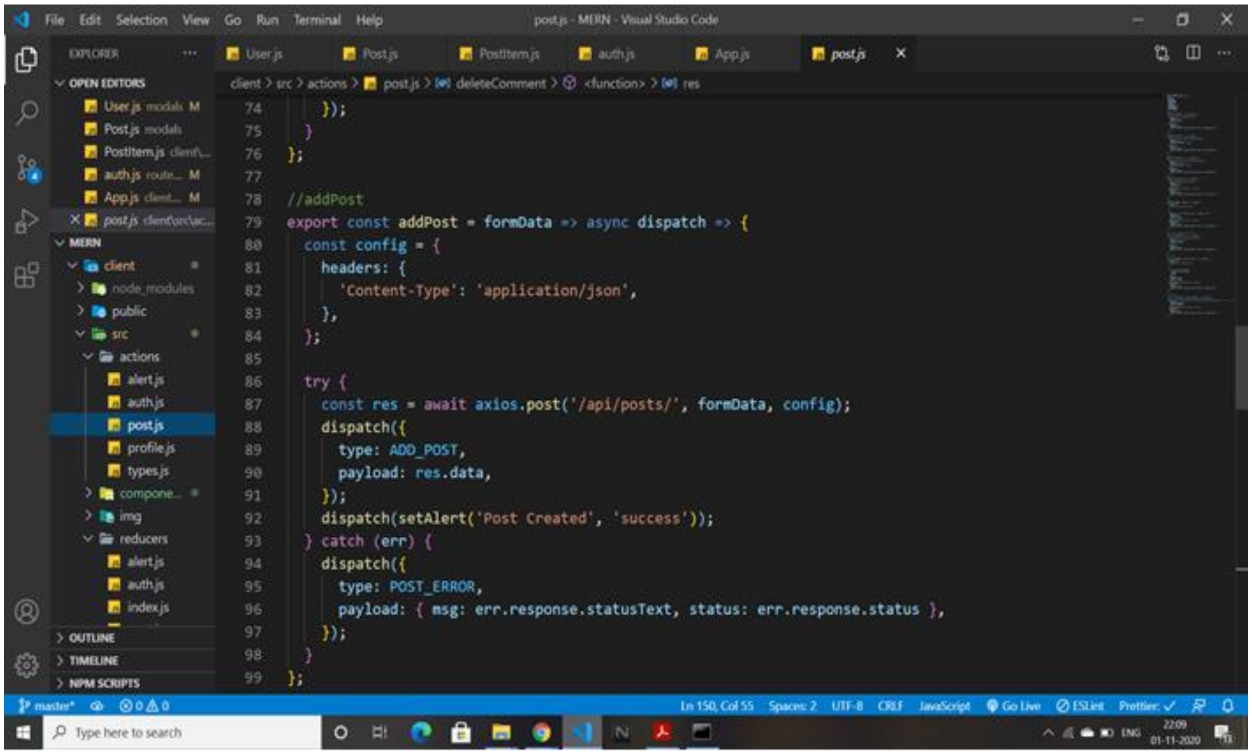
```
};

//Register user;
export const register = ({ name, email, password }) => async dispatch => {
  const config = {
    headers: {
      'Content-type': 'application/json',
    },
  };
  const body = JSON.stringify({ name, email, password });

  try {
    const res = await axios.post('/api/users', body, config);
    dispatch({
      type: REGISTER_SUCCESS,
      payload: res.data,
    });
    dispatch(loadUser());
  } catch (err) {
    const errors = err.response.data.errors;
    if (errors) {
      errors.forEach(error => dispatch(setAlert(error.msg, 'danger')));
    }
    dispatch({ type: REGISTER_FAIL });
  }
};
```

10. Alert Action



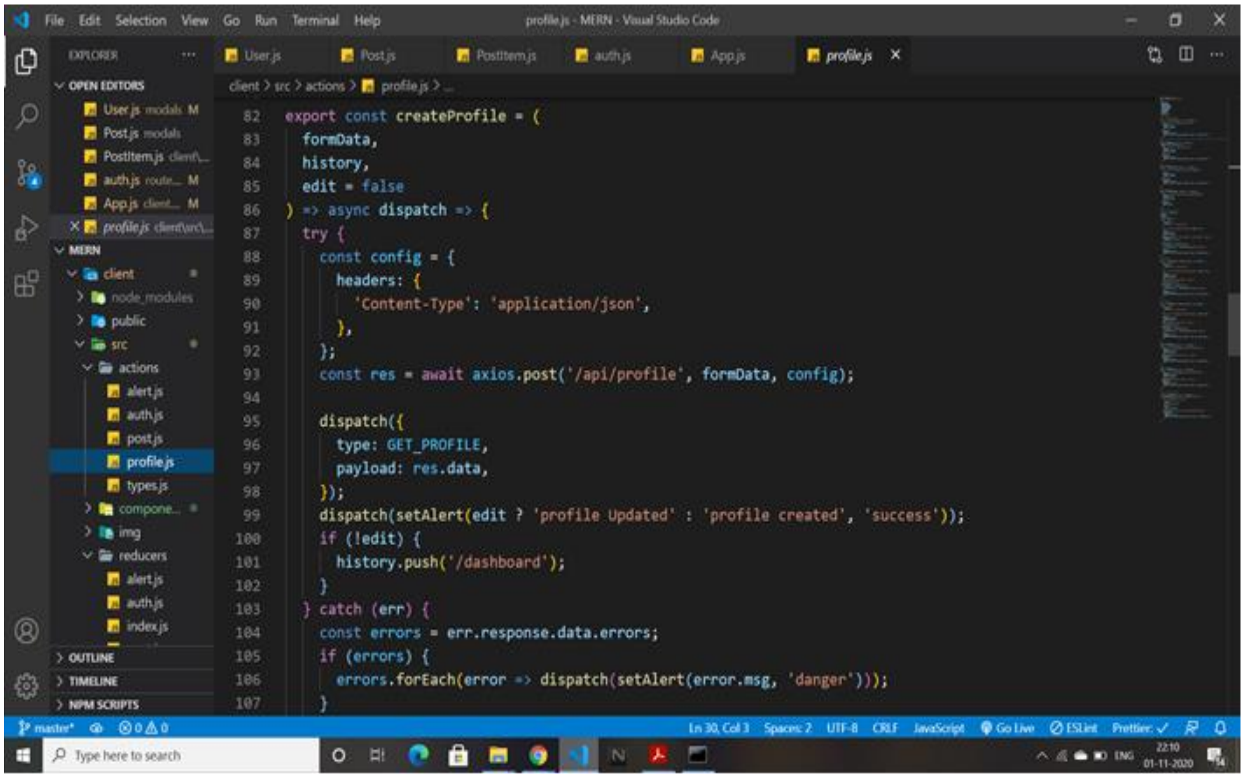
```
import { SET_ALERT, REMOVE_ALERT } from './types';
import { v4 as uuidv4 } from 'uuid';

export const setAlert = (msg, alertType, timeout = 5000) => dispatch => {
  const id = uuidv4();
  dispatch({
    type: SET_ALERT,
    payload: {
      msg,
      alertType,
      id,
    },
  });

  setTimeout(() => dispatch({ type: REMOVE_ALERT, payload: id }), timeout);
};
```

11. Add Post



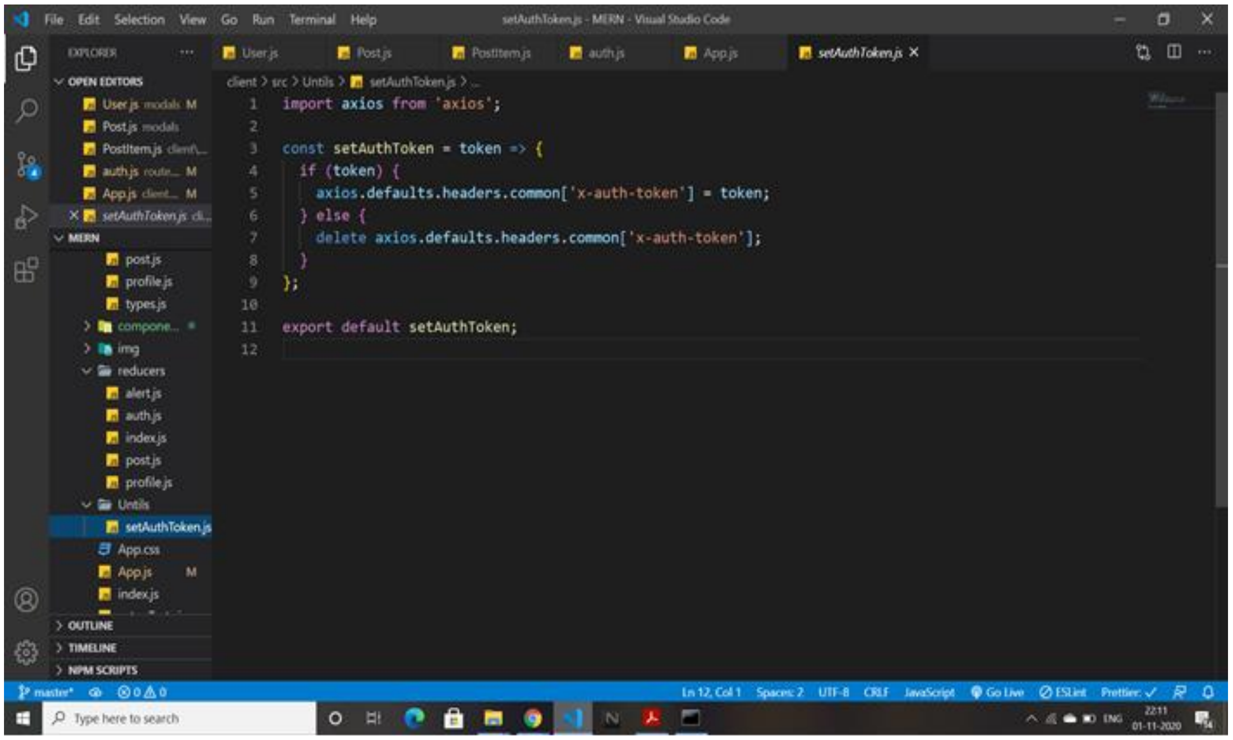
```
      });
    }
  };

//addPost
export const addPost = formData => async dispatch => {
  const config = {
    headers: {
      'Content-Type': 'application/json',
    },
  };

  try {
    const res = await axios.post('/api/posts/', formData, config);
    dispatch({
      type: ADD_POST,
      payload: res.data,
    });
    dispatch(setAlert('Post Created', 'success'));
  } catch (err) {
    dispatch({
      type: POST_ERROR,
      payload: { msg: err.response.statusText, status: err.response.status },
    });
  }
};
```

12. Create or Update profile


profile.js - MERN - Visual Studio Code
client > src > actions > profile.js

```
export const createProfile = (
  formData,
  history,
  edit = false
) => async dispatch => {
  try {
    const config = {
      headers: {
        'Content-Type': 'application/json',
      },
    };
    const res = await axios.post('/api/profile', formData, config);

    dispatch({
      type: GET_PROFILE,
      payload: res.data,
    });
    dispatch(setAlert(edit ? 'profile Updated' : 'profile created', 'success'));
    if (!edit) {
      history.push('/dashboard');
    }
  } catch (err) {
    const errors = err.response.data.errors;
    if (errors) {
      errors.forEach(error => dispatch(setAlert(error.msg, 'danger')));
    }
```

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

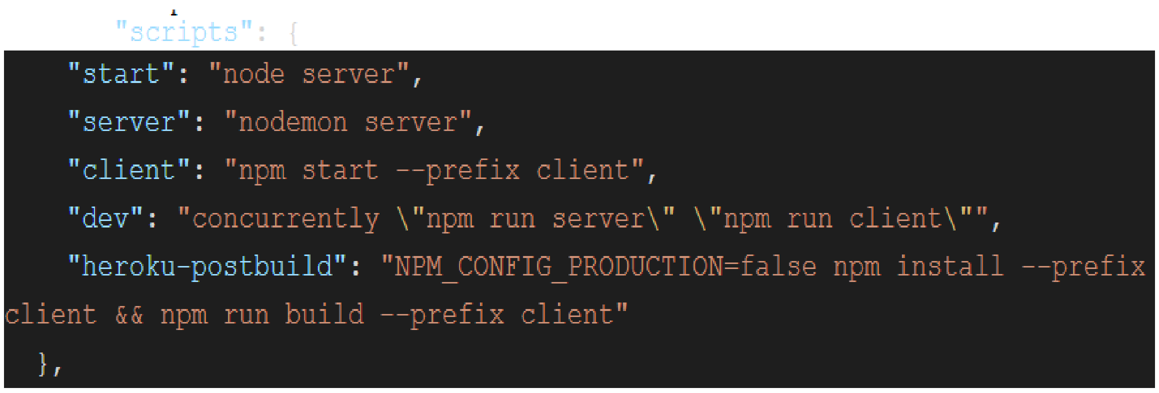

setAuthToken.js - MERN - Visual Studio Code
client > src > Untils > setAuthToken.js

```
import axios from 'axios';

const setAuthToken = token => {
  if (token) {
    axios.defaults.headers.common['x-auth-token'] = token;
  } else {
    delete axios.defaults.headers.common['x-auth-token'];
  }
};

export default setAuthToken;
```

Chapter 4: Implementation

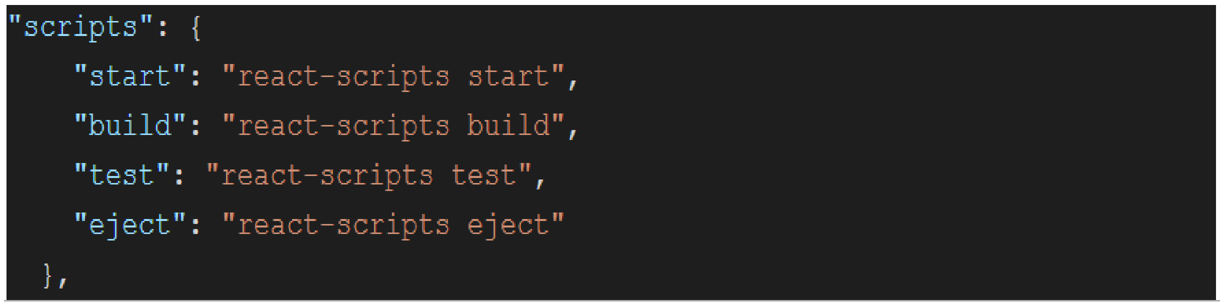
1. To run the project on local environment we used Concurrently , so that we can implement react and node js scripts together

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"
  },
```

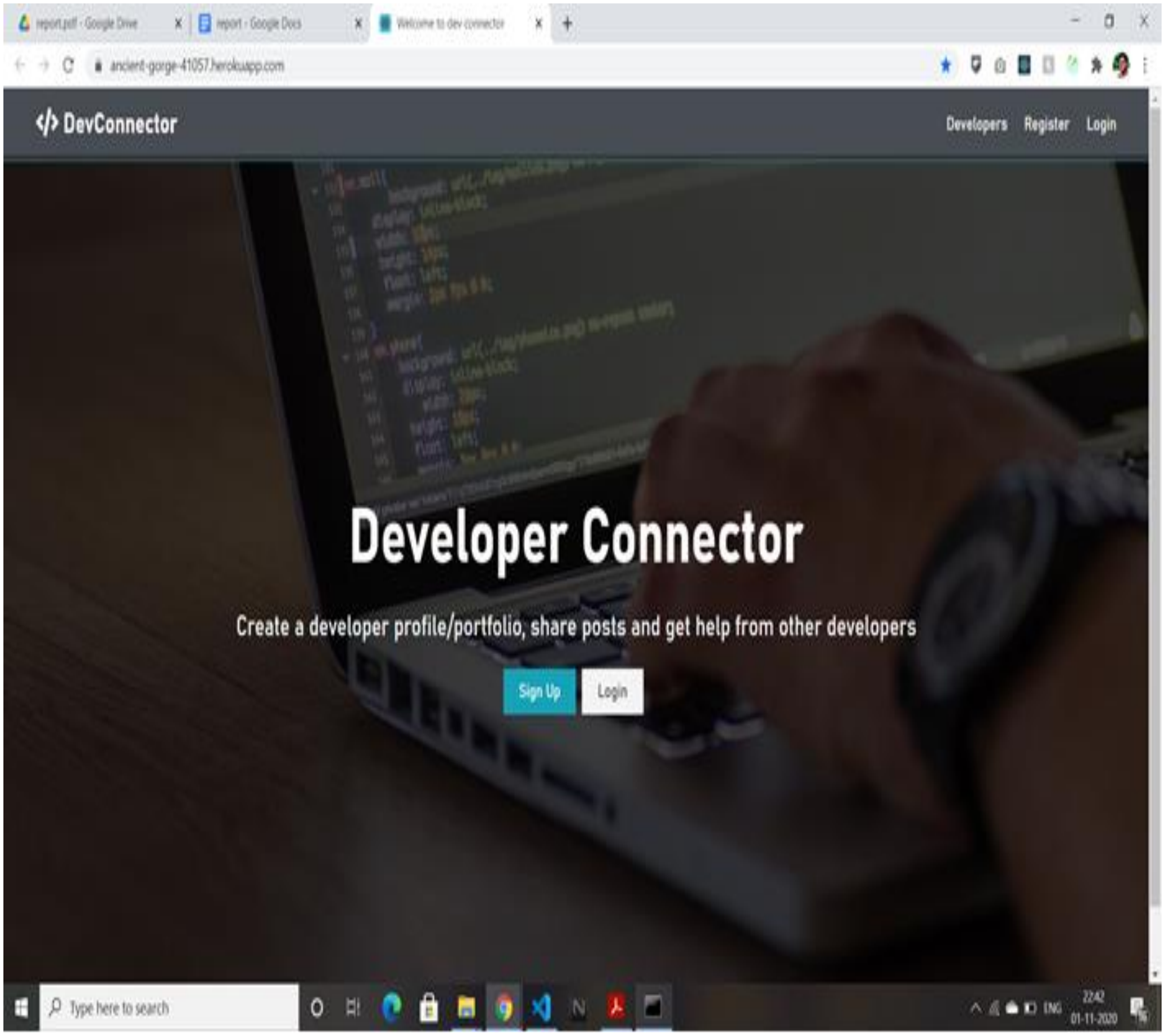
So as we are using concurrently to run two scripts from different files together we can write following command on the terminal >>npm run dev It will launch react app on localhost:3000 And express server on localhost:5000 simultaneously

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.

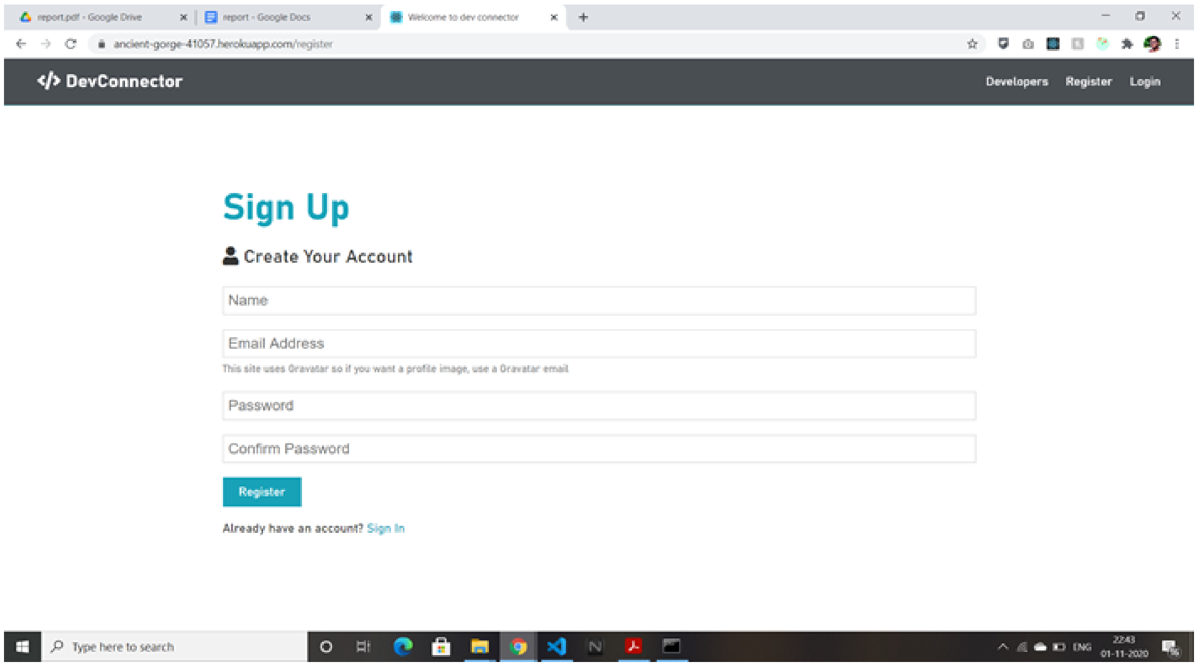
Chapter 5 : Final Results.

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)



Sign Up

Create Your Account

Name

Email Address

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

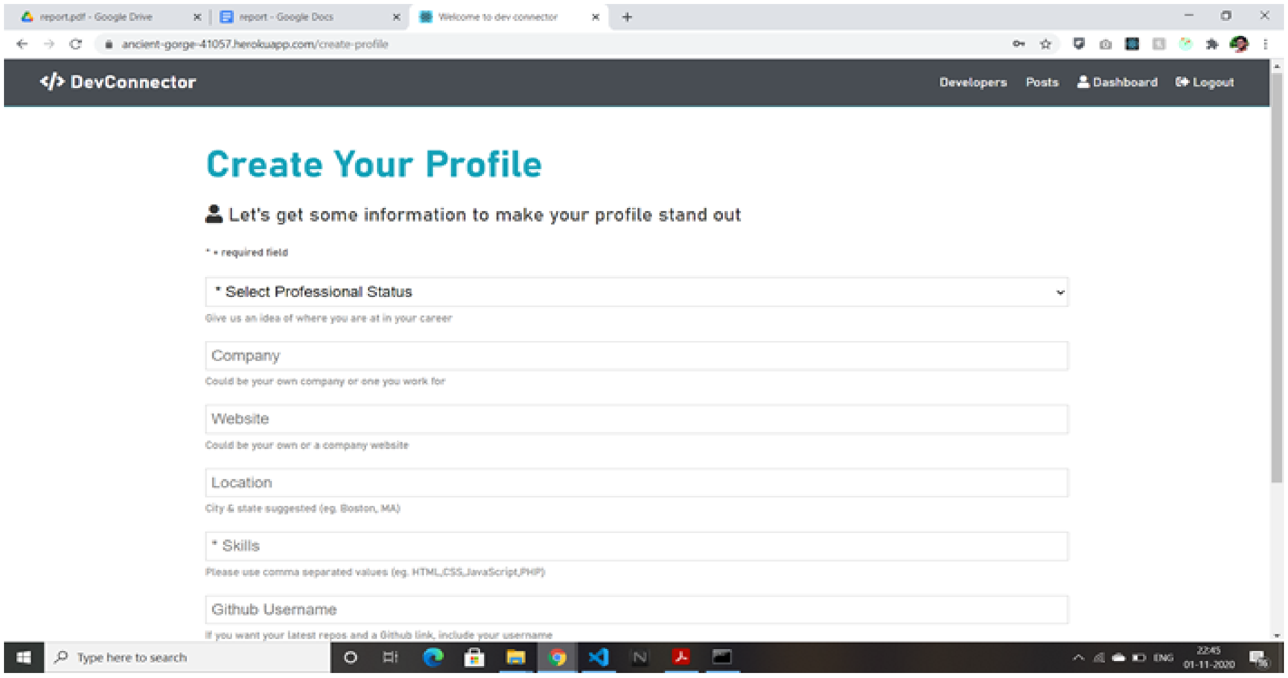
Password

Confirm Password

Register

Already have an account? Sign In

3. Profile creation



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)

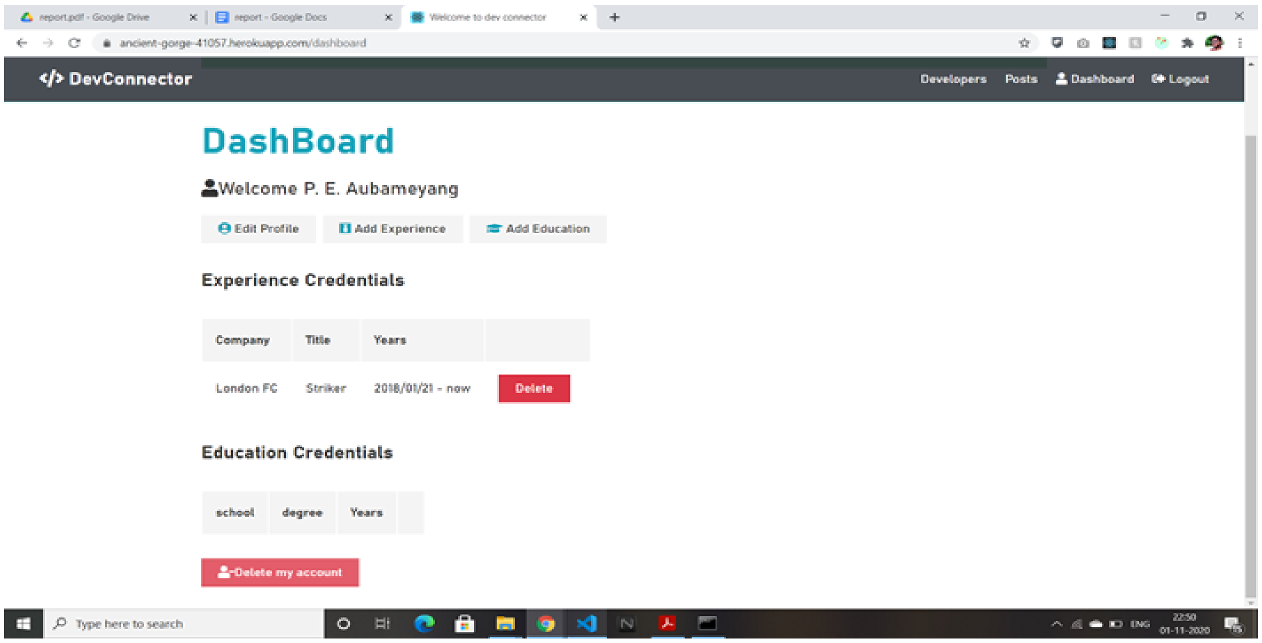
* Skills

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

4. Dashboard after profile creation



DashBoard

Welcome P. E. Aubameyang

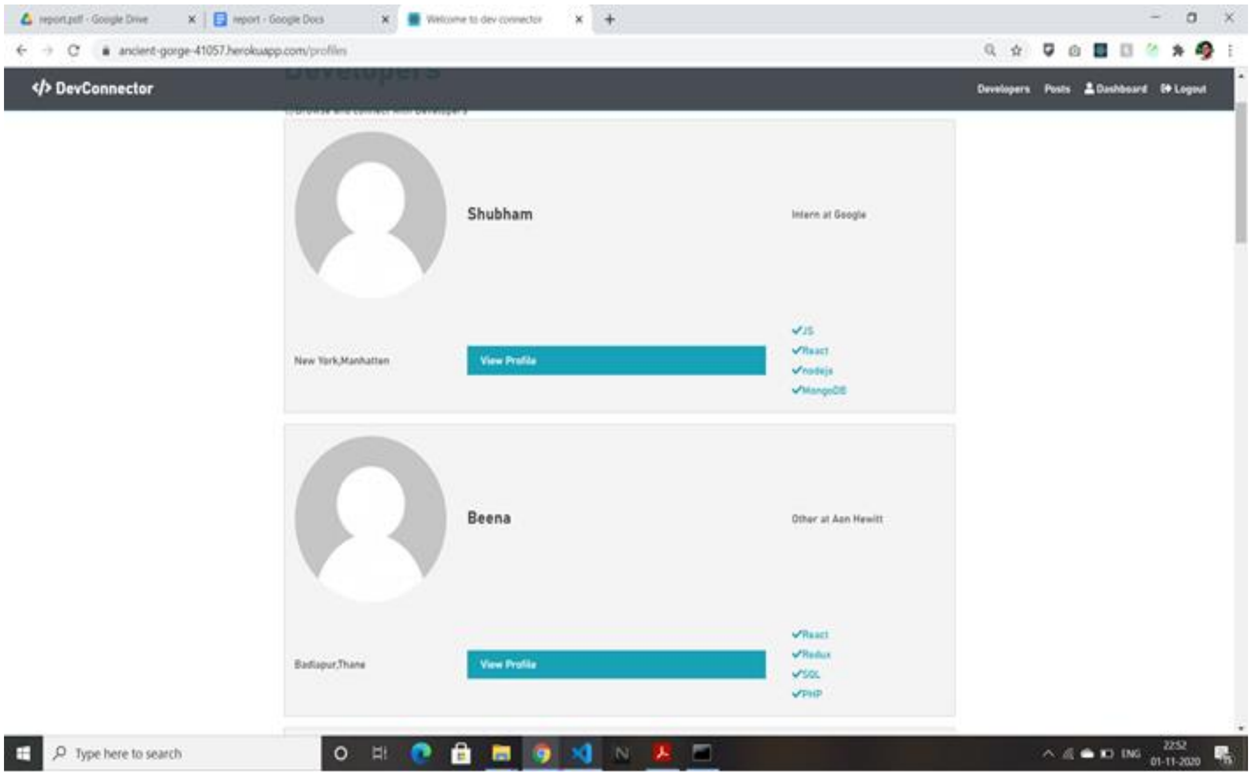
Experience Credentials

Company	Title	Years	
London FC	Striker	2018/01/21 - now	Delete

Education Credentials

school	degree	Years	

5. Browsing profile of other Developers



Shubham

Intern at Google

New York,Manhatten

✓JS
✓React
✓nodejs
✓MongoDB

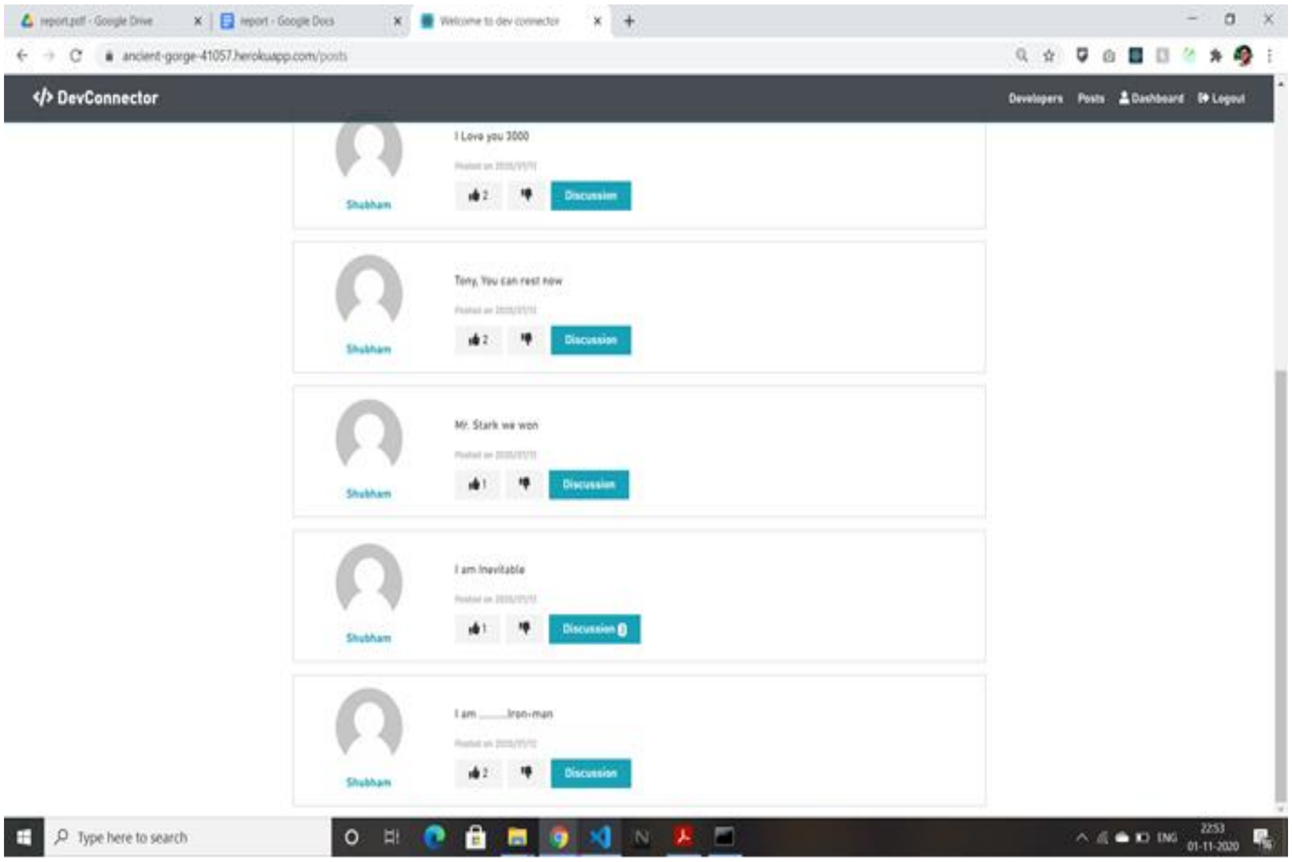
Beena

Other at Aon Hewitt

Badlapur,Thane

✓React
✓Redux
✓SQL
✓PHP

6. Recent Posts



Shubham

I Love you 3000

Posted on [illegible]

2 Discussion

Shubham

Tony, You can rest now

Posted on [illegible]

2 Discussion

Shubham

Mr. Stark we won

Posted on [illegible]

1 Discussion

Shubham

I am Inevitable

Posted on [illegible]

1 Discussion

Shubham

I am ....Iron-man

Posted on [illegible]

2 Discussion

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