Back-end Document

Database Setup and Connection

Written by Jiaxin Mo

MongoDB

In this project, we followed the previous work of Swatkats to use MongoDB as our main database for storing textual information, including URL of photos and files. It was cheap to use, easy to extend and maintain. MongoDB Atlas provided a free cluster with 500MB storage. When we defined the attributes in our code, the data models could be automatically setup in database. This allowed us to modify changes easily.

Related operations:

1. Create a mongoDB account on Atlas.
2. Create a free cluster.
3. Install 'MongoDB for VS Code' on VSCode (if your IDE is VSCode).
4. Update connection URL in server.js.

Tips: download a mongoDB Compass to monitor the database easier.

There were two data models defined, USERS and PROFILES. USERS database was for storing user`s account information, such as name, login email and login password. PROFILES database stored user`s porfolio information, such as education, work experience, URL of photos and files, etc.

Setup

Server.js

```
const mongoose = require('mongoose')

const mongoURI = "mongodb+srv://Aneesh97:Mufcwazza1997@clusteraneesh.puusf.mongodb.net/test?
retryWrites=true&w=majority"
//replaceable: "mongodb+srv://<username>:<password>@<clustername>.mongodb.net/test?
retryWrites=true&w=majority"
mongoose.connect(
        mongoURI,
    { useUnifiedTopology: true,
      useNewUrlParser: true,
      dbName: "It_project" }
        )
          .then(() => console.log('MongoDB Connected'))
          .catch(err => console.log(err))
require('./models/user.js');
require('./models/profile.js');

mongoose.set('useFindAndModify', false);
```

Models

Users.js

```
var mongoose = require('mongoose');

var userSchema = mongoose.Schema({
        "name": { type: String, required: true },
    "email": { type: String, required: true },
    "password": { type: String, required: true },
    });

var users = mongoose.model('users', userSchema);
```

Profiles.js

```
const mongoose = require('mongoose');
const mongoose_fuzzy_searching = require('mongoose-fuzzy-searching');

const ProfileSchema = new mongoose.Schema({
    user: {
        type: mongoose.Schema.Types.ObjectId,
        ref: 'user'
    },
    name:{type:String},
    profile_picture:{type:String},
    transcript:{type:String},
    website: {type: String},
        email: {type: String},
        phone: {type: String},
    skills: {type: [String]},
    bio: {type: String},
    work: [{
              workplace: {type: String},
              position: {type: String},
              from: {type: String},
              to: {type: String}
    }],
    gallery: [{
                imagesource: {type: String},
              description: {type: String}
    }],
    intro:{ type: String},
    education: [{
                school: {
                        type: String,
                  required: true
        },
        qual: {
                  type: String,
                  required: true
        }
        }],
    subjects: [{
        subjectname:{
                type:String,
                  required:true
        },
        subjectdescripition:{
                  type:String,
                  required:true
        },
        subjectyear:{
                  type:Number,
                  required:true
        }
    }],
    projects:[{
              projectname:{
                        type:String,
                        required:true
                },
              projectdescription:{
                        type:String,
                        required:true
                },
              projectlink:{type:String}
    }]
});

var profile = mongoose.model('profile', ProfileSchema);
```

Amazon Simple Storage Service (AWS S3)

AWS was used as our secondary database. Due to the limitation of small capacity of MongoDB, we selected AWS for its 5GB capacity to store a large number of pictures and PDF files. S3 provided a scalable storage in cloud for free, and was cost effective.

Related operations:

1. Create an AWS account, get the access key and security key.
2. Build two different buckets in your AWS S3, set the policies and permissions.
3. Install 'DOTENV' on VSCode (if your IDE is VSCode).
4. Create a .env file containing your AWS access key and security key at the same folder root of server.js.
5. Update bucket name and region in the files of services folder.

Tips: prepare a valid visa card for register.

Setup

img-upload.js

```
const aws = require('aws-sdk');
const dotenv = require('dotenv').config();

const ACCESS_KEY = process.env.S3_SECRET_ACCESS_KEY;
const ACCESS_ID = process.env.S3_ACCESS_KEY;

aws.config.update({
    secretAccessKey: ACCESS_KEY,
    accessKeyId: ACCESS_ID,
    region: 'ap-southeast-2' //your region
})
const s3 = new aws.S3();
```

.env

```
S3_ACCESS_KEY=<your access key>
S3_SECRET_ACCESS_KEY=<your secret key>
```

policies in AWS bucket

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "PublicRead",
            "Effect": "Allow",
            "Principal": "*",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Resource": "arn:aws:s3:::saltyfish/*" //bucket name
        }
    ]
}
```

Services

img-upload.js

```
const multer = require('multer');
const multerS3 = require('multer-s3');

const fileFilter = (req, file, cb) => {
        if (file.mimetype == 'image/jpeg' ||  file.mimetype  == 'image/png') {
            cb(null, true);
          } else {
            cb(new Error('Invalid mime type, only JPEG or PNG'), false);
          }
}

const img_upload = multer({
          fileFilter,
          storage: multerS3({
            s3: s3,
            bucket: 'saltyfish',//your bucket name
            metadata: function (req, file, cb) {
                      cb(null, {fieldName: 'TESTING_IMAGE'});
            },
            key: function (req, file, cb) {
                      cb(null, Date.now().toString() + ".jpg")
            }
          })
})
```

pdf-upload.js

```
const multer = require('multer');
const multerS3 = require('multer-s3');

const fileFilter = (req, file, cb) => {
          if (file.mimetype == 'application/pdf') {
            cb(null, true);
          } else {
            cb(new Error('Invalid mime type, only PDF'), false);
          }
}

const pdf_upload = multer({
    fileFilter,
    storage: multerS3({
                s3: s3,
              bucket: 'it-project-pdf-2021',//your bucket name
              metadata: function (req, file, cb) {
                cb(null, {fieldName: 'TESTING_PDF'});
              },
              key: function (req, file, cb) {
                cb(null, Date.now().toString() + ".pdf")
              }
    })
})
```

Deployment

Written by Jiaxin Mo

Local

This project was bootstrapped with Create React App.

To deploy this project in your local computer, you need to download Node.js firstly: https://nodejs.org/en/download/

In this project directory (same directory as server.js), some available commands and scripts can be run in a cmd prompt:

```
npm install
```

Install Node.js dependencies.

```
npm start
```

It runs the server in the development mode at http://localhost:5000. The page will reload if you save changes of your code.

```
npm run client
```

Open another cmd prompt and enter the command. It loads the client in your browser at http://localhost:3000.

Your app is deployed!

If you are unable to load the app due to some errors, please install the following scripts as well.

```
npm install openssl

npm install bootstrap

npm install react-google-login
//or npm install react-google-login --legacy-peer-deps

npm install react-facebook-login
//or npm install react-facebook-login --legacy-peer-deps
```

OpenSSL is a general-purpose cryptography library that stores some private information for HTTPS.

Packages of react-google-login and react-facebook-login are necessary for Google and Facebook login.

Heroku

How to deploy?

In the package.json, it needs to install the package and some parameters.

```
{
  "name": "it-project",
  "version": "1.0.0",
  "description": "",
  "main": "server.js",
  "scripts": {
    "client": "npm start --prefix client",
    "client-install": "npm install --prefix client",
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "node server.js",
    "heroku-postbuild": "NPM_CONFIG_PRODUCTION=false npm install --prefix client && npm run build --prefix
client",
    "dev": "concurrently \"npm start\" \"npm run client\""
  },
  "engines": {
    "node": "15.14.0"
  },
  "homepage": "https://github.com/UG-SaltyFish/Team-SaltyFish#readme",
  "dependencies": {
    //your dependencies
  }
}
```

After you get a Heroku account, you can run the following commands in a cmd at src/ directory:

```
heroku create saltyfish2 --buildpack mars/create-react-app
```

Login Heroku with your API key. It sets the app to use this buildpack. A new app will be created in your account.

```
git push heroku master
//or git push heroku $BRANCH_NAME:master
```

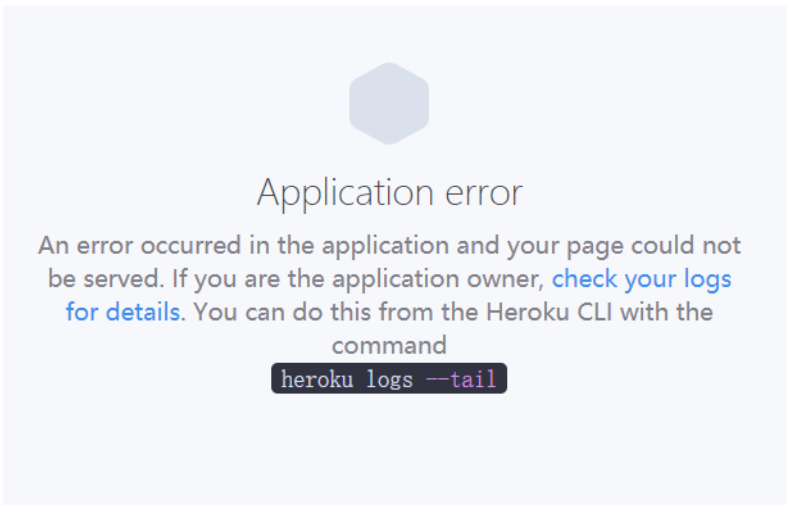
Deploy and push to the Heroku app from local repository.

```
heroku open
```

The app loads in your browser at public URL: https://saltyfish2.herokuapp.com

Result

Unfortunately, we could not successfully deploy the project in Heroku in Sprint 1. We got errors.



Application error

An error occurred in the application and your page could not be served. If you are the application owner, check your logs for details. You can do this from the Heroku CLI with the command

```
heroku logs --tail
```

Google/Facebook Login - Backend

Written by Jiaxin Mo Xiaoyue LIU

1. Google Login

React has a package called as react-google-login. It provides a simple mechanism to add the Google login to backend authentication.

Firstly, we applied for an OAuthClientID from Google developer console. This ClientID was implemented to get the access_token when the user had a successful login. We built this token into our "jwtToken". Then some profile details objects of the user were accessed. We saved Google name as our USER name, email as email, and googleId as password, which were the neccessary information for our ePortfolio USER account. If the Google login is used for the first time, a new profile will be established, or an error message of "Email already registered" is sent. Otherwise, the profile page is loaded. When the token gets expired, it needs to refresh and generate a new token.

Client components

login.jsx

```
import {GoogleLogin } from 'react-google-login';
import { loginUser, setUserLoading, setUserNotLoading, gooLoginUser, refreshTokenSetup } from "../actions
/authActions";

const googleClientId = <your google client id>;

class Login extends Component {
        constructor(props) {
            super(props);
            this.state = {
                        email: '',
                      password: '',
                      errors: '',
                      showmessage:false,
            };
          }

        onSuccess = (res) => {
            //console.log('[Login Success] currentUser:', res.profileObj);
            const googleUser = {
                      name: res.profileObj.name,
                      email: res.profileObj.email,
                      password: res.profileObj.googleId
            };
            this.props.gooLoginUser(googleUser);
            this.props.setUserLoading();
            refreshTokenSetup(res);
          };

          onFailure = (res) => {
            console.log('[Login failed] res:', res);
          };

        render() {
            return (
                        <div>
              <GoogleLogin
                 clientId={googleClientId}
          buttonText="Login"
          onSuccess={this.onSuccess}
          onFailure={this.onFailure}
          cookiePolicy={'single_host_origin'}
          isSignedIn={false}
         />
             </div>
                )
        }
}

Login.propTypes = {
          gooLoginUser: PropTypes.func.isRequired,
          refreshTokenSetup: PropTypes.func.isRequired
}
export default connect(
mapStateToProps,
{ loginUser, setUserLoading, setUserNotLoading, gooLoginUser, refreshTokenSetup }
)(Login);
```

Client actions

authActions.js

```
export const gooLoginUser = user => dispatch => {
    axios
            .post('/goologin', user)
              .then(res => {
                const token=res.data;
                console.log(token._id);
                localStorage.setItem("jwtToken", token._id);
                setAuthToken(token._id);
                const decoded = token._id;
                // Set the current user
                dispatch(setCurrentUser(decoded));
              })
              .catch(err => dispatch({
                type: SHOW_ERROR,
                payload: err.response.data
            }));
};

export const refreshTokenSetup = (res) =>{
    //Timing to renew access token
    let refreshTiming = (res.tokenObj.expires_in || 3600 - 5 * 60) * 1000;

    const refreshToken = async () => {
        const newAuthRes = await res.reloadAuthResponse();
        refreshTiming = (newAuthRes.expires_in || 3600 - 5 * 60) * 1000;
        console.log('newAuthRes:', newAuthRes);

        console.log('new auth Token', newAuthRes.id_token);
        //Setup the other timer after the first one
        setTimeout(refreshToken, refreshTiming);
    };
    //Setup first refreshtimer
    setTimeout(refreshToken, refreshTiming);
};
```

Controller

controller.js

```
var gooLoginUser = function(req, res) {

    const user = req.body;

    //Check for existing user to login
    User.findOne({email:user.email,password:user.password}, function(err, user1) {
        if (user1) {
            const payload = {
                _id: user1._id,
                name: user1.name,
                email: user1.email
              }
              let token = payload;
              res.send(token);
        }
        else{
            //Register for google login
            var user = new User({
                "name":req.body.name,
                "email":req.body.email,
                "password":req.body.password
            });

            User.findOne({email:user.email}, function(err, user1) {
                if (user1) {
                    return res.status(401).json("Email already registered");
                } else {
                    user.save(function (err, newUser) {
                        if (!err) {
                            var profile =new Profile({
                                user: newUser,
                                name: user.name,
                                profile_picture: "https://it-project-bucket-2020.s3-ap-southeast-1.amazonaws.
com/blank-profile.png",
                                transcript: "",
                                website:'',
                                gallery:[],
                                education:[],
                                subjects:[],
                                projects:[],
                                work:[],
                                intro:"",
                                email:user.email,
                                phone:'',
                                skills:[],
                                bio:'',
                                date:'',
                                sectionE:'block',
                                sectionW:'block',
                                sectionP:'block',
                                sectionSk:'block',
                                sectionSu:'block',
                                sectionG:'block'
                            });
                            profile.save();
                            return res.send("User created");
                        } else {
                            res.sendStatus(400);
                        }
                    });
                }
            });
        }
    });
};
```

Routes

router.js

```
router.post('/goologin', controller.gooLoginUser);
```

2. Facebook Login

Reset Page - Backend

Written by Jiaxin Mo

First, User account information is mounted from USER database with current login user ID. We set user`s name and email as default information shown in form. User can change any part of name, email or password, but not requiring all of them. When the user submit changes, it will push the update to both USER and PROFILE database with user ID.

Components

reset.jsx

```
componentWillMount(){
    const user = this.props.auth.user;
    console.log(user);

    axios
                .get('/user/' + user)
              .then(res=>{
                this.setState({
                        user:res.data,
                          id:res.data.id,
                          name:res.data.name,
                          email:res.data.email
                });
              });
}

onSubmit(e) {
    e.preventDefault();

    const id = this.props.auth.user;
    const updatedUser = {
              id: this.state.user._id,
              name: this.state.name,
              email: this.state.email,
              password: this.state.password,
              password2: this.state.password2
    };

    axios
              .post('/user/' + id + '/update',updatedUser)
              .then(res => {
                this.setState({
                          email:res.data.email,
                          name:res.data.name
                });
              })
              .then(res => {this.props.history.push("/login");})
              .catch(err => this.setState({ errors: err.response.data }));
  }
```

Routes

user.js

```
const express = require('express');
const router = express.Router();

const User = require('../models/user.js');
var user_controller = require('../controller.js');

// GET request for one user
router.get('/user/:id', user_controller.getUserAccount);

// POST request for update user's detail
router.post('/user/:id/update', user_controller.resetUser);
```

Controller

controller.js

```
// GET one user`s account
var getUserAccount = function (req, res) {
    User.findById(req.params.id, function (err, user) {
        console.log(user.id);
        if (err) {
            return res.status(409).json("User not found");
        } else {
            res.send(user);
        }
    });
};

// Reset UserAcc
var resetUser = function(req, res) {
   User.findOne({ email: req.body.email }).then((user) => {
        if (user && (req.body.id != user.id)) {
            console.log(req.body.id);
            return res.status(409).json("Email already exists");
        } else {
            //If password is not changed
            if (req.body.password == '') {
                //Update User name and email
                User.findOneAndUpdate({ _id: req.body.id },
                    {
                        name: req.body.name,
                        email: req.body.email
                    },function (err, updatedUser) {
                        console.log(updatedUser);
                        if (err) {
                            return res.status(409).json("Wrong");
                        } else {
                            //Update Profile name and email
                            Profile.findOne({user: req.body.id}).then((file1) => {
                                if (!file1) {
                                    return res.status(409).json("Profile not found");
                                } else {
                                    Profile.findOneAndUpdate({user: req.body.id},
                                        {
                                            name: req.body.name,
                                            email: req.body.email
                                        },{},function (err, updatedProfile) {
                                            if (err) {
                                                return res.status(400).json("Error: Can not update profile");
                                            } else {
                                                return null;
                                            }
                                        }
                                    );
                                }
                            });
                            res.send(updatedUser);
                        }
                    }
                );
            } else if (req.body.password != req.body.password2) {
                return res.status(409).json("Password doesn`t match");
            } else {
                //If password is changed, update
                User.findOneAndUpdate({_id: req.body.id },
                    {
                        name: req.body.name,
                        email: req.body.email,
                        password: req.body.password,
                    },function (err, updatedUser2) {
                        if (err) {
                            return res.status(400).json("Wrong!");
                        } else {
                            Profile.findOne({user: req.body.id}).then((file2) => {
                                if (!file2) {
                                    return res.status(409).json("Profile not found");
                                } else {
                                    Profile.findOneAndUpdate({user: req.body.id},
                                        {
                                            name: req.body.name,
                                            email: req.body.email
                                        },{},function (err, updatedProfile2) {
                                            if (err) {
                                                return res.status(400).json("Error: Can not update profile");
                                            } else {
                                                return null;
                                            }
                                        }
                                    );
                                }
                            });
                            res.send(updatedUser2);
                        }
                    }
                );
            }
        }
    });
};
```

Select section - Backend

Written by Xiaoyue LIU

When the user clicks on the checkbox corresponding to each section, the system will record the display status of the section and store it in the database. Each section has corresponding attributes. When the user logs in again or refreshes, the data of the corresponding section in the database will be read again and set to the section and checkbox. The checkbox will also determine whether defaultchecked is required according to the corresponding section status in the database.

Component

profile.jsx

```
componentDidMount() {
    axios
        .get('/profile1/'+(this.props.auth.user))
        .then(res=>{
          this.setState({email:res.data[0].email,
                         name:res.data[0].name,
                         bio:res.data[0].bio,
                         intro:res.data[0].intro,
                         skills:res.data[0].skills,
                         work:res.data[0].work,
                         projects:res.data[0].projects,
                         subjects:res.data[0].subjects,
                         gallery:res.data[0].gallery,
                         education:res.data[0].education,
                         website:res.data[0].website,
                         phone:res.data[0].phone,
                         sectionE:res.data[0].sectionE,
                         sectionW:res.data[0].sectionW,
                         sectionP:res.data[0].sectionP,
                         sectionSk:res.data[0].sectionSk,
                         sectionSu:res.data[0].sectionSu,
                         sectionG:res.data[0].sectionG,
                         profilePicture: res.data[0].profile_picture,
                         transcript: res.data[0].transcript,
                         imgHash: Date.now()
                        });




constructor(props) {
    super(props);
    this.state = {
      visible:'',
      intro:'',
      email: '',
      name: '',
      bio: '',
      skills: [],
      subjects: [],
      education: [],
      gallery: [],
      projects:[],
      website: '',
      phone: '',
      sectionE:'',
      sectionW:'',
      sectionP:'',
      sectionSk:'',
      sectionSu:'',
      sectionG:'',
      selectedFile: null,
      profilePicture: '',
      transcript: '',
      showAdd:false,
      showlang:false,
      addsubjectname:'',
      addsubjectyear:'',
      addsubjectdescripition:'',
      addgallerydescription:'',
      showintro:false,
      addintro:'',
      addinfo:'',
      showskills:'',
      showedu:false,
      addschoolname:'',
      addqual:'',
      showwork:false,
      addworkplace:'',
      addposition:'',
      addfrom:'',
      addto:'',
      showproject:false,
      addprojectname:'',
      addprojectdescripition:'',
      addprojectlink:'',
      showphone:false,
      lang:'en',
      hastranscript:'notranscript',
      filename:''
    };








hideEdu = ()=>{
    var checkbox1 = document.getElementById("edusec");
    var edusection = document.getElementById("educationsec");
    if(checkbox1.checked == false){
      edusection.style.display = "none";
    }else{
      edusection.style.display = "block";
    }
    const userData={
      sectionE:edusection.style.display

    }

    axios.put('/hideE/'+this.props.auth.user,userData)
    .then(res=> this.setState({sectionE:res.data.sectionE}))

  }

  hideWork = ()=>{
    var checkbox1 = document.getElementById("worksec");
    var worksection = document.getElementById("worksection");
    if(checkbox1.checked == false){
      worksection.style.display = "none";
    }else{
      worksection.style.display = "block";
    }
    const userData={
      sectionW:worksection.style.display
    }
    axios.put('/hideW/'+this.props.auth.user,userData)
    .then(res=> this.setState({sectionW:res.data.sectionW}))
  }

  hideProj = ()=>{
    var checkbox1 = document.getElementById("projsec");
    var projsection = document.getElementById("projectsection");
    if(checkbox1.checked == false){
      projsection.style.display = "none";
    }else{
      projsection.style.display = "block";
    }
    const userData={
      sectionP:projsection.style.display
    }
    axios.put('/hideP/'+this.props.auth.user,userData)
    .then(res=> this.setState({sectionP:res.data.sectionP}))
  }

  hideSkill = ()=>{
    var checkbox1 = document.getElementById("skillsec");
    var skisection = document.getElementById("skillsection");
    if(checkbox1.checked == false){
      skisection.style.display = "none";
    }else{
      skisection.style.display = "block";
    }
    const userData={
      sectionSk:skisection.style.display
    }
    axios.put('/hideSk/'+this.props.auth.user,userData)
    .then(res=> this.setState({sectionSk:res.data.sectionSk}))
  }

  hideSub = ()=>{
    var checkbox1 = document.getElementById("subsec");
    var subsection = document.getElementById("subjectsection");
    if(checkbox1.checked == false){
      subsection.style.display = "none";
    }else{
      subsection.style.display = "block";
    }
    const userData={
      sectionSu:subsection.style.display
    }
    axios.put('/hideSu/'+ this.props.auth.user,userData)
    .then(res=> this.setState({sectionSu:res.data.sectionSu}))
  }

  hideGal = ()=>{
    var checkbox1 = document.getElementById("galsec");
    var galsection = document.getElementById("gallerysection");
    if(checkbox1.checked == false){
      galsection.style.display = "none";
    }else{
      galsection.style.display = "block";
    }
    const userData={
      sectionG:galsection.style.display
    }
    axios.put('/hideG/'+ this.props.auth.user,userData)
    .then(res=> this.setState({sectionG:res.data.sectionG}))
  }
```

```
<input type="checkbox" id="edusec" defaultChecked={this.state.sectionE!=='none'} onClick={this.hideEdu} ><
/input>
<input type="checkbox" id ="worksec" defaultChecked={this.state.sectionW!=='none'} onClick={this.hideWork}><
/input>
<input type="checkbox" id ="projsec" defaultChecked={this.state.sectionP!=='none'} onClick = {this.hideProj}
></input>
<input type="checkbox" id ="skillsec" defaultChecked={this.state.sectionSk!=='none'} onClick = {this.
hideSkill}></input>
<input type="checkbox" id = "subsec" defaultChecked={this.state.sectionSu!=='none'} onClick = {this.hideSub}
></input>
<input type="checkbox" id = "galsec" defaultChecked={this.state.sectionG!=='none'} onClick = {this.hideGal}><
/input>
```

```
<section id="education" style={{display:this.state.sectionE}}>
<section id="work" style={{display:this.state.sectionW}} >
<section id='projects' style={{display:this.state.sectionP}}>
<section id="skills" style={{display:this.state.sectionSk}}>
<section id="subjects" style={{display:this.state.sectionSu}}>
<section id = "gallery" style={{display:this.state.sectionG}}>
```

Public View Component

publicprofile.jsx

```
constructor(props) {
    super(props);
    this.state = {
      intro:'',
      email: '',
      name: '',
      bio: '',
      skills: [],
      subjects: [],
      education: [],
      gallery: [],
      projects:[],
      website: '',
      phone: '',
      sectionE:'',
      sectionW:'',
      sectionP:'',
      sectionSk:'',
      sectionSu:'',
      sectionG:'',
      selectedFile: null,
      profilePicture: '',
      transcript: '',
      showAdd:false,
      showlang:false,
      addsubjectname:'',
      addsubjectyear:'',
      addsubjectdescripition:'',
      addgallerydescription:'',
      showintro:false,
      addintro:'',
      addinfo:'',
      showskills:'',
      showedu:false,
      addschoolname:'',
      addqual:'',
      showwork:false,
      addworkplace:'',
      addposition:'',
      addfrom:'',
      addto:'',
      showproject:false,
      addprojectname:'',
      addprojectdescripition:'',
      addprojectlink:'',
      showphone:false,
      lang:'en',
      filename:''
    };
```

```
componentDidMount() {
    axios
        .get('/profile2/'+(this.props.match.params.user))
        .then(res=>{
          this.setState({email:res.data[0].email,
                         name:res.data[0].name,
                         bio:res.data[0].bio,
                         intro:res.data[0].intro,
                         skills:res.data[0].skills,
                         work:res.data[0].work,
                         projects:res.data[0].projects,
                         subjects:res.data[0].subjects,
                         gallery:res.data[0].gallery,
                         projects:res.data[0].projects,
                         education:res.data[0].education,
                         website:res.data[0].website,
                         phone:res.data[0].phone,
                         sectionE:res.data[0].sectionE,
                         sectionW:res.data[0].sectionW,
                         sectionP:res.data[0].sectionP,
                         sectionSk:res.data[0].sectionSk,
                         sectionSu:res.data[0].sectionSu,
                         sectionG:res.data[0].sectionG,
                         profilePicture: res.data[0].profile_picture,
                         transcript: res.data[0].transcript,
                         imgHash: Date.now()
                        });






<section id="education" style={{display:this.state.sectionE}}>
<section id="work" style={{display:this.state.sectionW}}>
<section id='projects' style={{display:this.state.sectionP}}>
<section id="skills" style={{display:this.state.sectionSk}}>
<section id="subjects" style={{display:this.state.sectionSu}}>
<section id = "gallery" style={{display:this.state.sectionG}}>
```

Routes

router.js

```
router.put('/hideE/:user',controller.deleE);
router.put('/hideW/:user',controller.deleW);
router.put('/hideP/:user',controller.deleP);
router.put('/hideSk/:user',controller.deleSk);
router.put('/hideSu/:user',controller.deleSu);
router.put('/hideG/:user',controller.deleG);
```

Controller

controller.js

```
var deleE= function(req, res){
    var user1=req.params.user;

    const sec= req.body;
    Profile.findOneAndUpdate({user:user1},{$set: {sectionE:sec.sectionE}},{new: true},function(err,user2){
        if(err){

            res.send("wrong");

        }else{

           res.send(user2);
        }
    });
}

var deleW= function(req, res){
    var user1=req.params.user;

    const sec= req.body;


    Profile.findOneAndUpdate({user:user1},{$set: {sectionW:sec.sectionW}},{new: true},function(err,user2){
        if(err){

            res.send("wrong");

        }else{

           res.send(user2);
        }
    });
}

var deleP= function(req, res){
    var user1=req.params.user;

    const sec= req.body;


    Profile.findOneAndUpdate({user:user1},{$set: {sectionP:sec.sectionP}},{new: true},function(err,user2){
        if(err){

            res.send("wrong");

        }else{

           res.send(user2);
        }
    })
}
var deleSk= function(req, res){
    var user1=req.params.user;

    const sec= req.body;


    Profile.findOneAndUpdate({user:user1},{$set: {sectionSk:sec.sectionSk}},{new: true},function(err,user2){
        if(err){

            res.send("wrong");

        }else{

           res.send(user2);
        }
    })
}

var deleSu= function(req, res){
    var user1=req.params.user;

    const sec= req.body;
    Profile.findOneAndUpdate({user:user1},{$set: {sectionSu:sec.sectionSu}},{new: true},function(err,user2){
        if(err){

            res.send("wrong");

        }else{

           res.send(user2);
        }
    })
}

var deleG= function(req, res){
    var user1=req.params.user;

    const sec= req.body;


    Profile.findOneAndUpdate({user:user1},{$set: {sectionG:sec.sectionG}},{new: true},function(err,user2){
        if(err){

            res.send("wrong");

        }else{

           res.send(user2);
        }
    })
}



module.exports.deleE = deleE;
module.exports.deleW = deleW;
module.exports.deleP = deleP;
module.exports.deleSk = deleSk;
module.exports.deleSu = deleSu;
module.exports.deleG = deleG;
```