SOCIAL MEDIA FOR STUDENTS

J-COMPONENT REVIEW REPORT

Submitted by

**Deep Sanjay Agrawal (18BCE0518)**

**Shubham Agarwal (18BCE0542)**

Prepared For

**INTERNET AND WEB PROGRAMMING (CSE3002)**

**PROJECT COMPONENT**

Submitted To

**Prof. JAYAKUMAR K**

**School of Computer Science and Engineering**

****

**INDEX**

|  |  |  |
| --- | --- | --- |
| **Sr. No.** | **Chapter** | **Page No.** |
| 1 | Abstract | 3 |
| 2 | Introduction | 3 |
| 3 | Existing System Problems | 4 |
| 4 | Proposed System Designs | 5 |
| 5 | Result and Conclusion | 10 |
| 6 | Output Screenshots | 12 |
| 7 | References | 16 |
| 8 | Appendix | 17 |

**Abstract**

The idea of the website is to provide one place for all the social media requirements of a student. The webapp will require authentication from the user. The users for their profile will have a place where they can display their education, work, experience, skillset, their GitHub repos’, social media links and other such things considered necessary for students. This profile will be visible to everyone with or without login. Once logged in, a user can connect with any other user. The next thing required for students is a forum like post section. Here, a user will be able to post text along with photos and videos and a forum like comment section will be present. Update functionalities for profile and sections of profile will be given. Delete functionalities for comments will be given. Deletion of account will be given as an option. Other functionalities will be added as and when felt required.

**INTRODUCTION**

*Problem Statement:*

To successfully implement the ideas presented using the MERN stack and build upon the foundations of Web Programming obtained through this course. The website created provides one place for all the social media requirements of a student. The website will require authentication from the user. The users for their profile will have a place where they can display their education, work, experience, skillset, their GitHub repos’, social media links and other such things considered necessary for students. This profile will be visible to everyone with or without login. The next thing required for students is a forum like post section. Here, a user will be able to post text along with photos and videos and a forum like comment section will be present. Update functionalities for profile and sections of profile will be given. Delete functionalities for comments will be given. Deletion of account will be given as an option. Other functionalities will be added as and when felt required.

*Applications:*

The website in itself is a complete application of the idea on which the project is based. It can be deployed on a web server and can be used by everyone through WWW.

*Limitations:*

The only limitations to the website are it exhaustive features that can be worked on in the future and the existing ones can be made better.

*Technical Specifications:*

*Client-side* – React JS and Redux

*Server-side* – Node JS and Express

*Database* – MongoDB and Mongoose

*CLI tool* – NPM

*Hardware Requirements:*

*System:* Desktop / Laptop / Mobile

*Processor:* Any with clock speed of 2GHz and above

*RAM:* 2 GB and above

*Internet:* Ethernet or WIFI or Cellular Network

*Software Requirements:*

*Operating System:* Any

*Browsers:* JavaScript Enabled

**EXISTING SYSTEM PROBLEMS**

*Existing System:*

Currently the systems or social websites used by students / programmers / students are divided into two distinct categories according to their purposes. On one hand we have sites like LinkedIn, where people share their profiles and create a network. On other hand we have sites like Stack Exchange or other technical forum websites where people help out their fellow professionals or students with their queries.

*Proposed System:*

The proposed system builds upon these two seemingly disconnected systems to create a one stop go for students. It has the profile and achievements sharing and connections-based functionalities as well as forum like abilities.

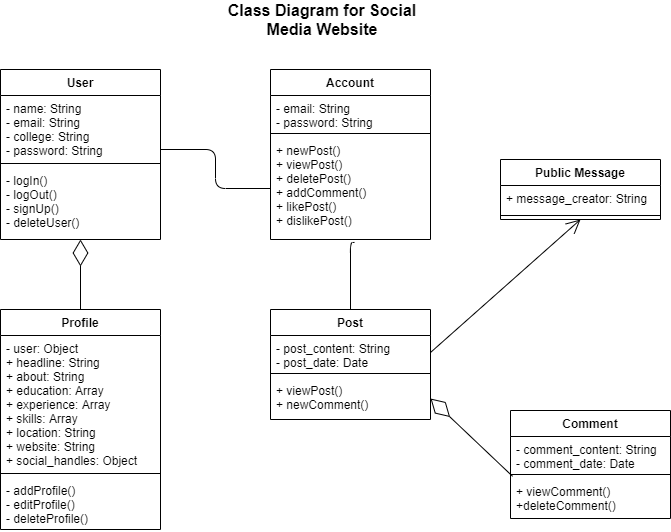
*Benefits of Proposed System:*

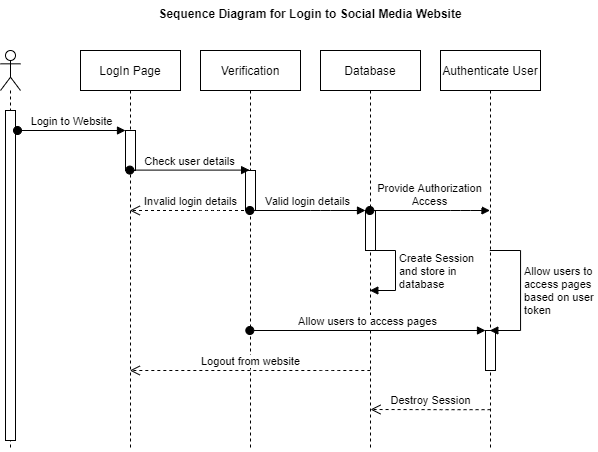
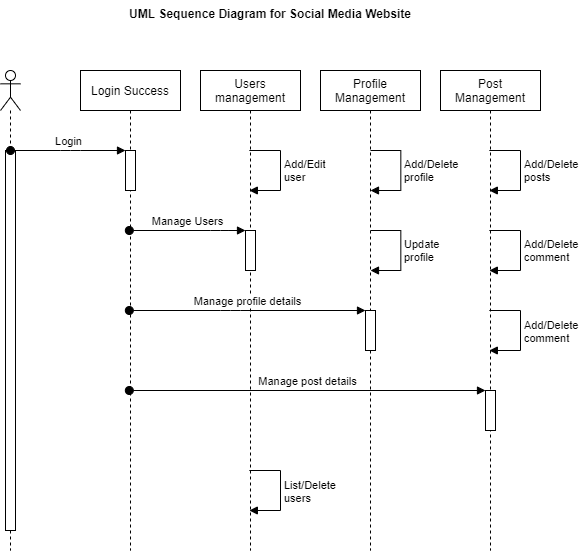
* The hassle of using two different sites for two not so different purposes is gone.
* Increasing your professional network can be also made through helping others through forum rather than just sharing profiles. This is an innovation in socializing.

**PROPOSED SYSTEM DESIGNS**

*System Architecture:*

To explain the system architecture, following UML Diagrams have been used:





*Module Description:*

The system has been broken into following for modules:

1. *Authentication:* This module gets the user information as input like email id and password and compares the information from the database. If the provided inputs match from the entries in the database, the user is given access to the web pages. It includes both login and register pages from the application. Also, multiple users cannot access the site with same credentials. The password stored in the database is encrypted to avoid middle man to get access to our account.
2. *Authorization:* This is the middleware that runs every time a new request is made to a protected route. For example, one user cannot access the page to edit the profile of another user even after accessing the correct route with the URL. Since after logging in, the authentication token of the user is saved as a header, on requesting a protected route, each time the authentication token is verified to check the authorization to the requested page.
3. *User and Profile Module:* This module involves design of the profile view and edit pages of a user and handles the CRUD operation for profile database. The information that profile can contain includes: experience credentials (company, title, years); education credentials (school, degree, years); career; city; skills; github repos; twitter, facebook, linkedin profile URLs; decription; username.
4. *Post:* This modules involves design of posting section which includes comment section common to all users. It also handles CRUD operations on post dataset. Users can create a post and delete theirs later. Users can view others’ posts. Users can comment on posts and delete theirs later. Users can like, dislike any post or comment.
5. *Alert Module:* Alert Module is responsible for all the alerts appearing in the website. Message in the alert can be from both backend and frontend. This module visually shows message passing between client and server using request response model.

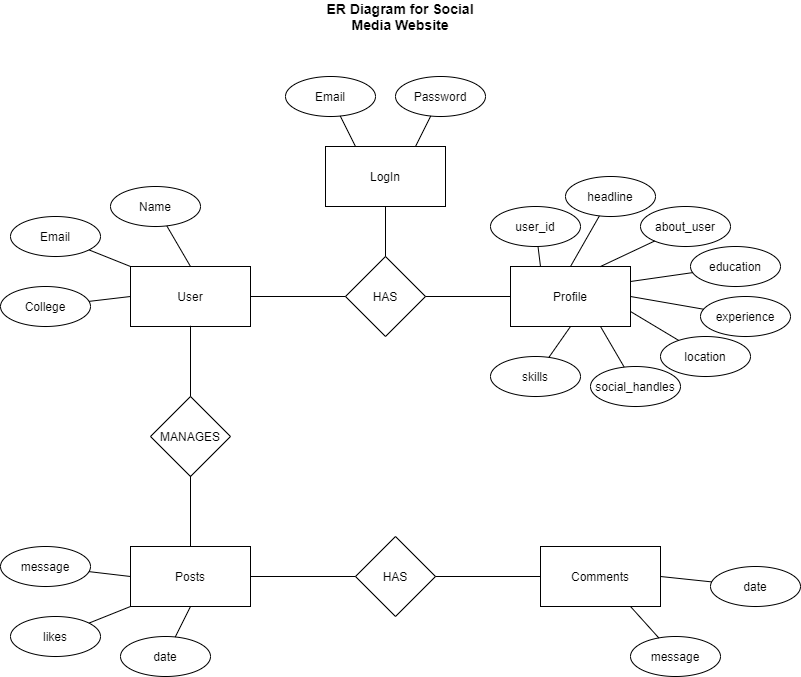
**RESULTS AND CONCLUSION**

The objective set and the functionalities desired initially were both achieved through the website that was created. A lot of functionalities can be added to make it more inclusive and feature rich. Following are some of the ideas:

* File attachment to posts.
* Following another user’s post automatically by connecting with them.
* Messaging another user privately.
* Explore and recommend section.

*Database Design:*

To explain the database design, following ER Diagram has been used:



**OUTPUT SCREENSHOTS**

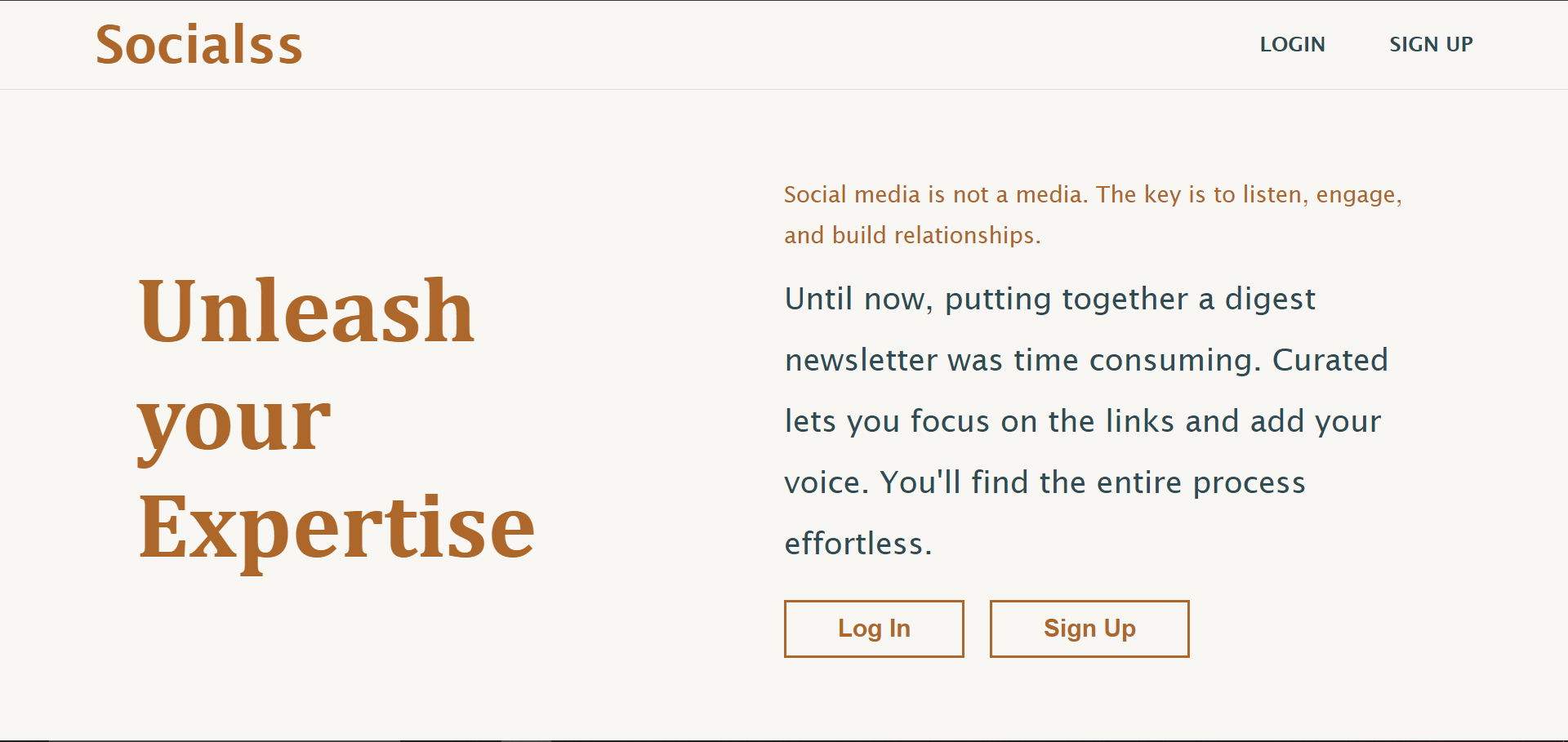


Figure : Home page

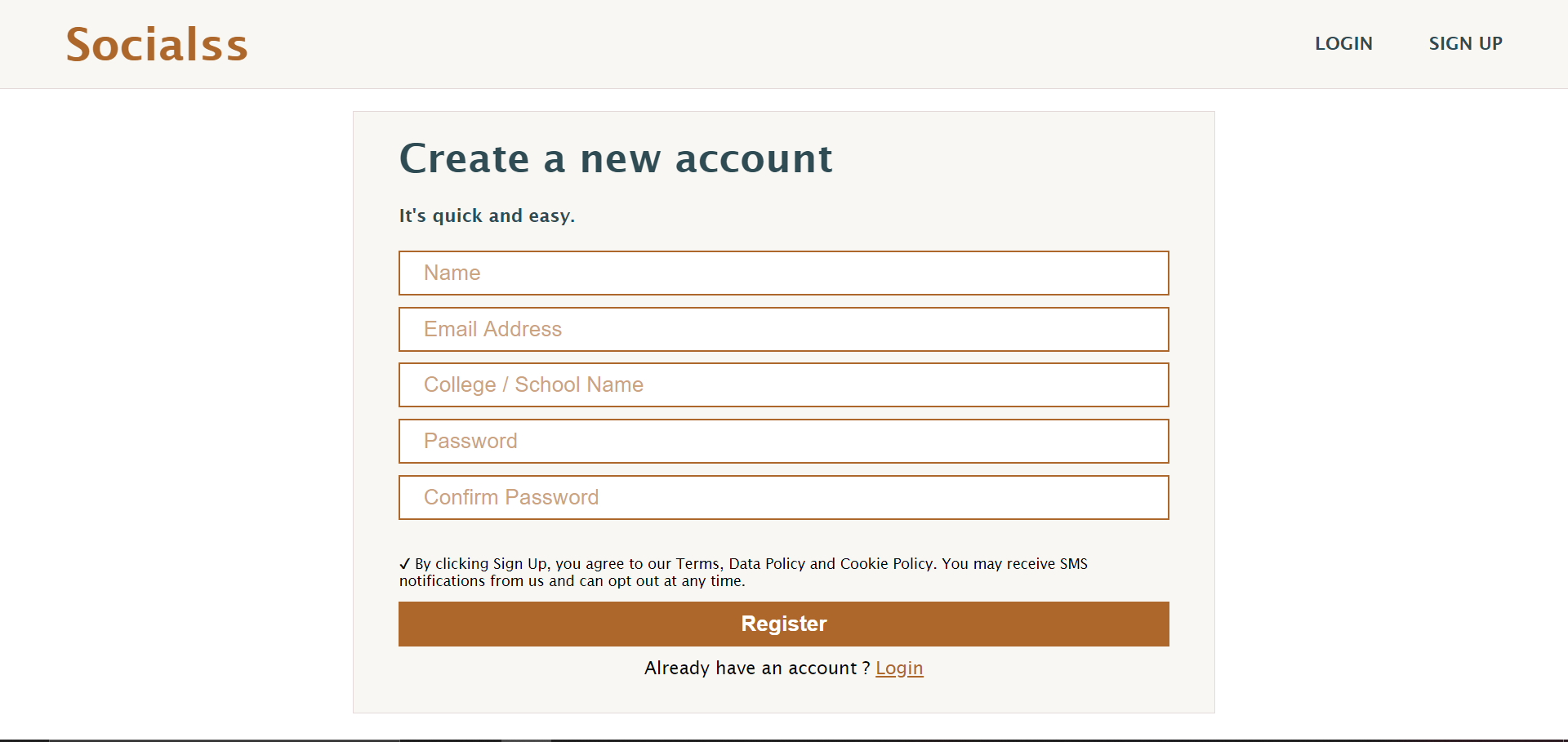


Figure : Register page

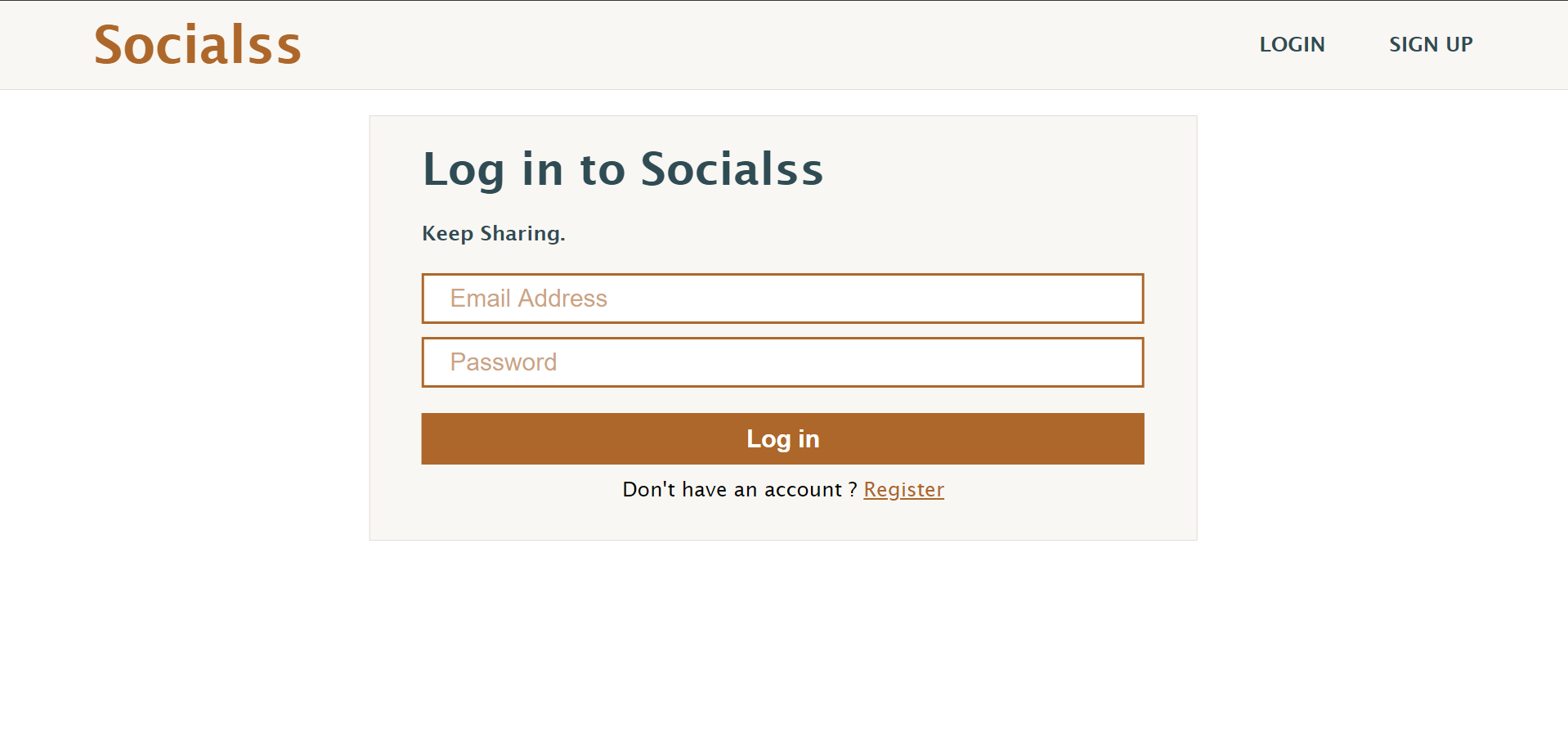


Figure : Login page

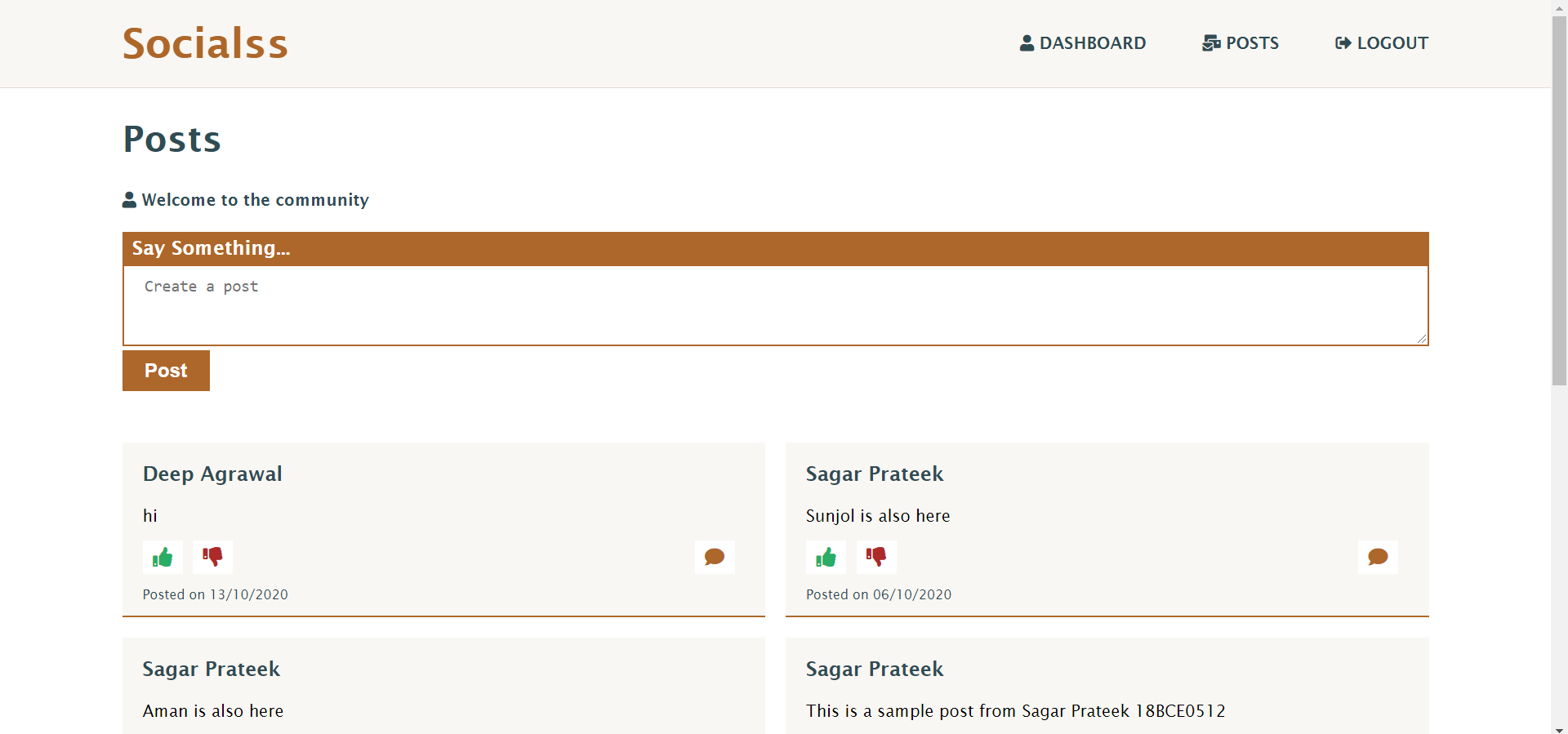


Figure : Post page

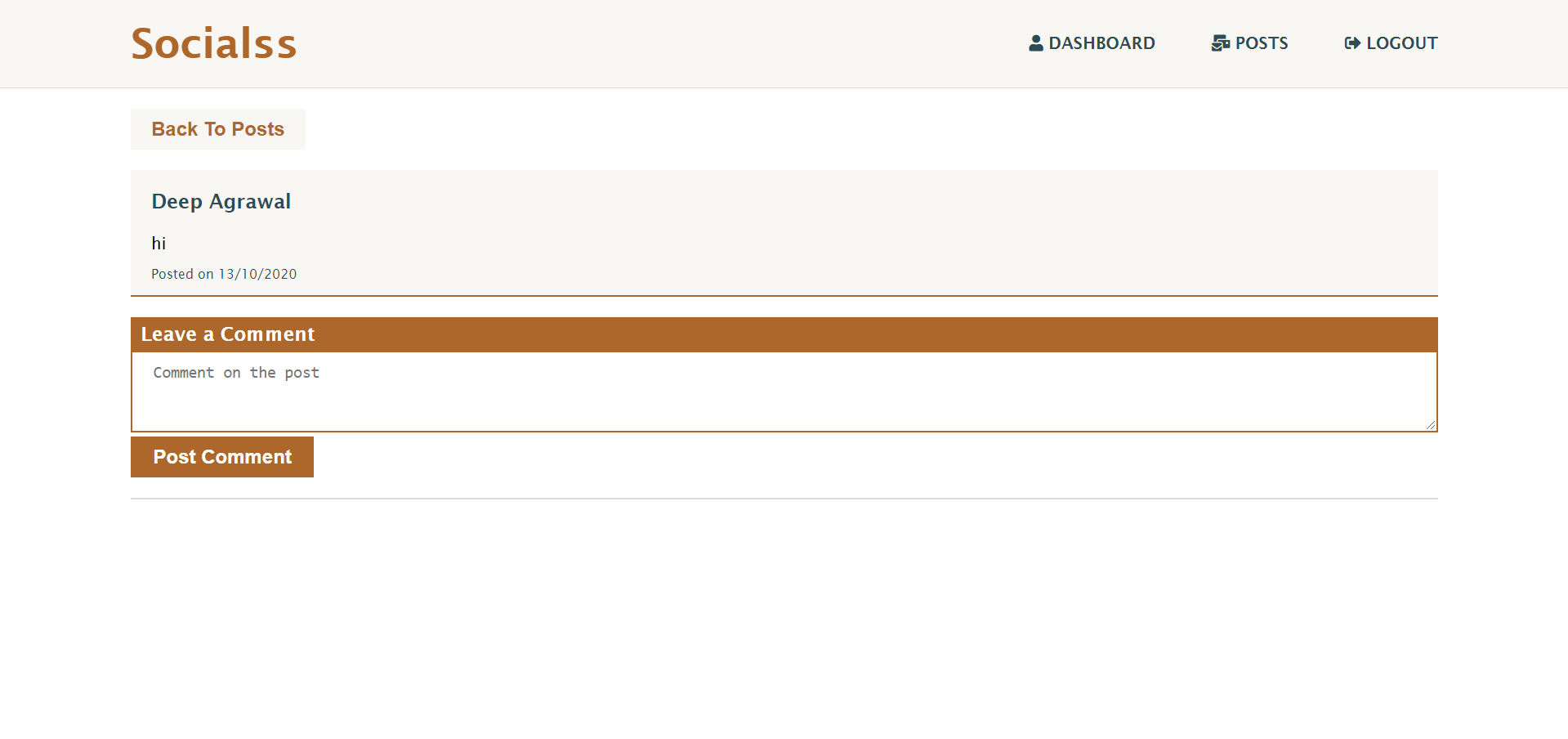


Figure : Comments page

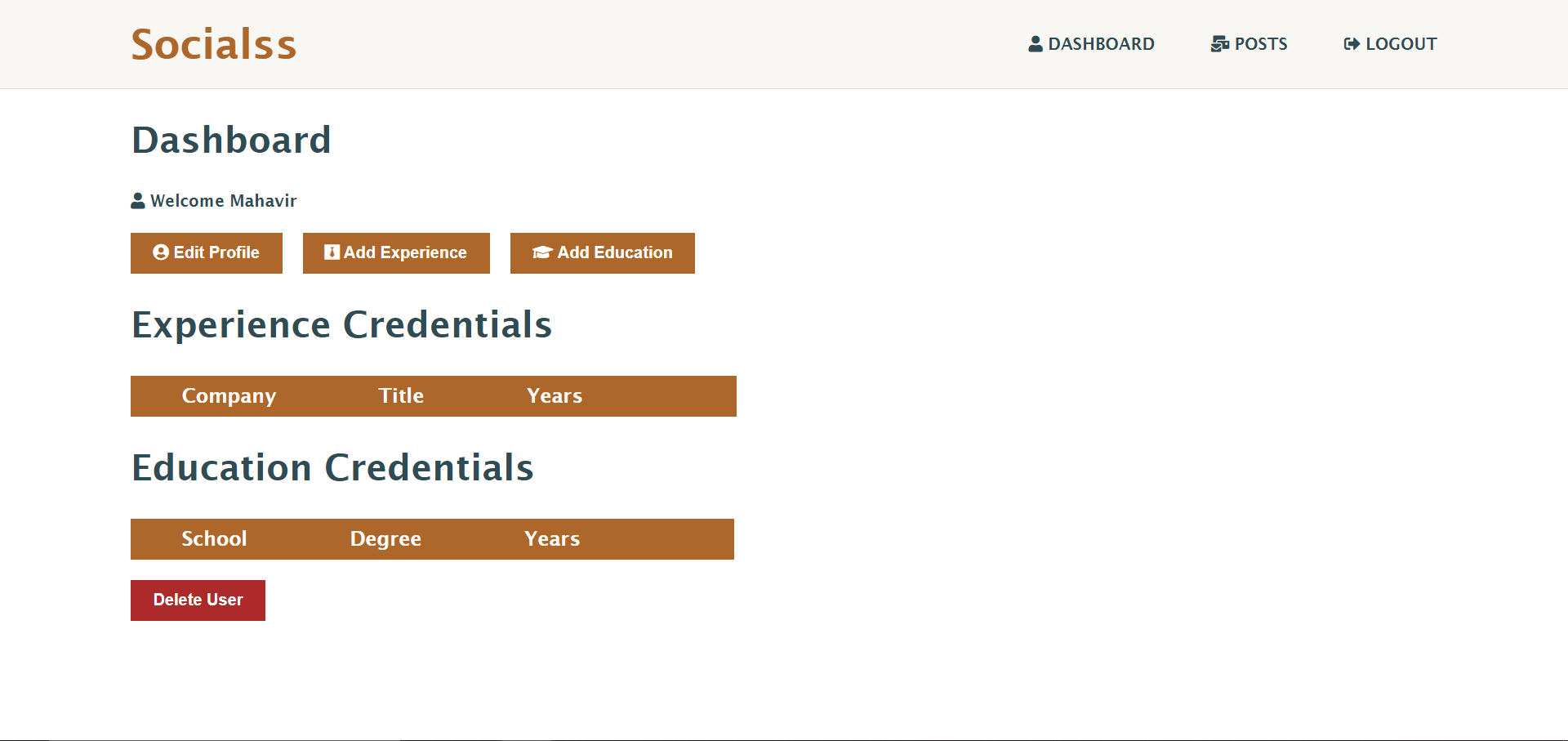


Figure : Dashboard



Figure : Edit profile

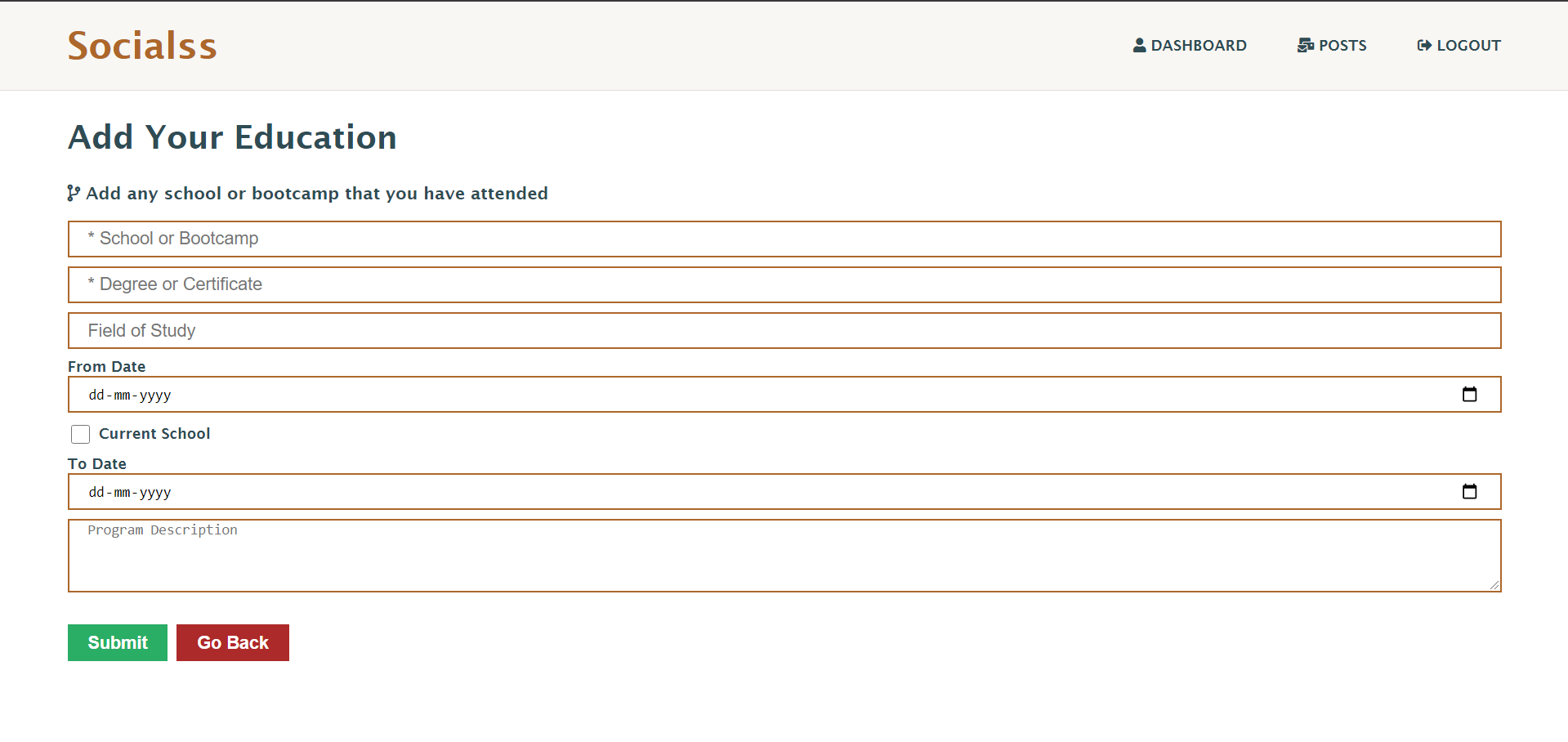


Figure : Add Education

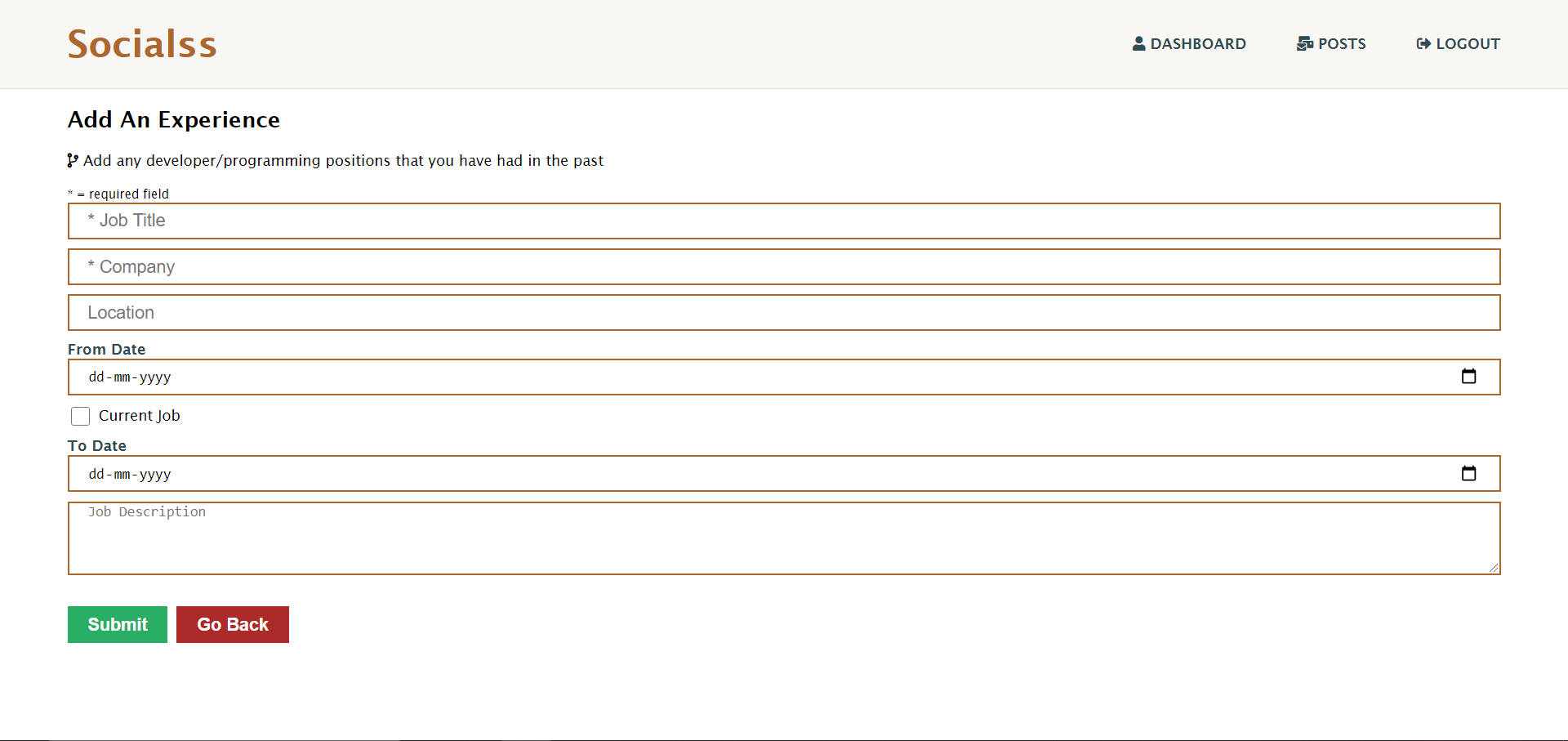


Figure : Add experience

**REFERENCES**

[1] React – A JavaScript library for building user interfaces. (n.d.). Retrieved October 30, 2020, from https://reactjs.org/

[2] Redux - A predictable state container for JavaScript apps.: Redux. (n.d.). Retrieved October 30, 2020, from https://redux.js.org/

[3] Mongoose. (n.d.). Retrieved October 30, 2020, from https://mongoosejs.com/

[4] Goyal, H. (2020, July 15). Building a Social Network using MERN stack. Retrieved October 30, 2020, from https://medium.com/code-dementia/building-a-social-network-using-mern-stack-85037914b944

[5] MERN Social 2.0. (n.d.). Retrieved October 30, 2020, from https://awesomeopensource.com/project/shamahoque/mern-social?categoryPage=37

**APPENDIX:**

*Sample Source Code:*

*Frontend:*

* Register Page

const Register = ({ setAlert, register, isAuthenticated }) => {

  const [name, setName] = useState('');

  const [email, setEmail] = useState('');

  const [password, setPassword] = useState('');

  const [cpassword, setCpassword] = useState('');

  const [college, setCollege] = useState('');

  const onSubmit = async (e) => {

    e.preventDefault();

    if (password !== cpassword) {

      setAlert('Passwords do not match', 'danger');

    } else {

      register({ name, email, college, password });

    }

  };

  // Redirect if authenticated

  if (isAuthenticated) {

    return <Redirect to='/dashboard' />;

  }

  return (

    <React.Fragment>

      <div className='register container'>

        <div className='form-container'>

          <h2>Create a new account</h2>

          <h4>It's quick and easy.</h4>

          <form className='form' onSubmit={(e) => onSubmit(e)}>

            <div className='form-group'>

              <input

                type='text'

                value={name}

                onChange={(e) => setName(e.target.value)}

                placeholder='Name'

                name='name'

                required

              />

            </div>

            <div className='form-group'>

              <input

                type='email'

                value={email}

                onChange={(e) => setEmail(e.target.value)}

                placeholder='Email Address'

                name='email'

                required

              />

            </div>

            <div className='form-group'>

              <input

                type='text'

                value={college}

                onChange={(e) => setCollege(e.target.value)}

                placeholder='College / School Name'

                name='college'

                required

              />

            </div>

            <div className='form-group'>

              <input

                type='password'

                value={password}

                onChange={(e) => setPassword(e.target.value)}

                placeholder='Password'

                name='password'

                minLength='6'

              />

            </div>

            <div className='form-group'>

              <input

                type='password'

                value={cpassword}

                onChange={(e) => setCpassword(e.target.value)}

                placeholder='Confirm Password'

                name='cpassword'

                minLength='6'

              />

            </div>

            <div className='terms'>

              ✔ By clicking Sign Up, you agree to our Terms, Data Policy and

              Cookie Policy. You may receive SMS notifications from us and can

              opt out at any time.

            </div>

            <button type='submit'>Register</button>

            <div className='login'>

              Already have an account ? <Link to='/login'>Login</Link>

            </div>

          </form>

        </div>

      </div>

    </React.Fragment>

  );

};

* Login Page

const Login = ({ login, isAuthenticated }) => {

  const [email, setEmail] = useState('');

  const [password, setPassword] = useState('');

  const onSubmit = async (e) => {

    e.preventDefault();

    login(email, password);

  };

  // Redirect if authenticated

  if (isAuthenticated) {

    return <Redirect to='/posts' />;

  }

  return (

    <React.Fragment>

      <div className='register container'>

        <div className='form-container'>

          <h2>Log in to Socialss</h2>

          <h4>Keep Sharing.</h4>

          <form className='form' onSubmit={(e) => onSubmit(e)}>

            <div className='form-group'>

              <input

                type='email'

                value={email}

                onChange={(e) => setEmail(e.target.value)}

                placeholder='Email Address'

                name='email'

                required

              />

            </div>

            <div className='form-group'>

              <input

                type='password'

                value={password}

                onChange={(e) => setPassword(e.target.value)}

                placeholder='Password'

                name='password'

                minLength='6'

              />

            </div>

            <button type='submit'>Log in</button>

            <div className='login'>

              Don't have an account ? <Link to='/register'>Register</Link>

            </div>

          </form>

        </div>

      </div>

    </React.Fragment>

  );

};

* Dashboard page

const Dashboard = ({

  getCurrentProfile,

  deleteAccount,

  auth: { user },

  profile: { profile, loading },

}) => {

  useEffect(() => {

    getCurrentProfile();

  }, [getCurrentProfile]);

  return loading && profile === null ? (

    <Spinner />

  ) : (

    <React.Fragment>

      <div className='container dashboard'>

        <h2>Dashboard</h2>

        <h4>

          <i className='fas fa-user'></i> Welcome {user && user.name}

        </h4>

        {profile !== null ? (

          <React.Fragment>

            <DashboardActions />

            <Experience experience={profile.experience} />

            <Education education={profile.education} />

            <div className='delete-acc'>

              <button onClick={() => deleteAccount()}>Delete User</button>

            </div>

          </React.Fragment>

        ) : (

          <React.Fragment>

            <div className='no-profile'>

              <p>You have not setup your profile yet.</p>

              <Link to='create-profile'>

                <button>Create Profile</button>

              </Link>

            </div>

          </React.Fragment>

        )}

      </div>

    </React.Fragment>

  );

};

* Post Page

const Post = ({ getPost, post: { post, loading }, match }) => {

  useEffect(() => {

    getPost(match.params.id);

  }, [getPost, match.params.id]);

  return loading || post === null ? (

    <Spinner />

  ) : (

    <div className='post'>

      <button className='back-btn'>

        <Link to='/posts'>Back To Posts</Link>

      </button>

      <PostItem post={post} showActions={false} />

      <CommentForm postId={post.\_id} />

      <hr />

      <div className='comments'>

        {post.comments.map((comment) => (

          <CommentItem key={comment.\_id} comment={comment} postId={post.\_id} />

        ))}

      </div>

    </div>

  );

};

There are more pages to all the four modules mentioned, but only the main ones are attached. Stylesheets also have not been attached.

*Backend:*

* Authentication

router.get('/', auth, async (req, res) => {

  try {

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

    res.json(user);

  } catch (error) {

    console.error(error.message);

    res.status(500).send('💥Server error');

  }

});

router.post(

  '/',

  [

    check('email', 'Please enter a valid email').isEmail(),

    check('password', 'Password is required').exists(),

  ],

  async (req, res) => {

    // Validation Result

    const errors = validationResult(req);

    if (!errors.isEmpty()) {

      return res.status(400).json({ errors: errors.array() });

    }

    const { email, password } = req.body;

    try {

      let user = await User.findOne({ email });

      // See if user exists

      if (!user) {

        return res

          .status(400)

          .json({ errors: [{ msg: 'User does not exist' }] });

      }

      // Validating passwod

      const isMatch = await bcrypt.compare(password, user.password);

      if (!isMatch) {

        return res

          .status(400)

          .json({ errors: [{ msg: 'Invalid credentials' }] });

      }

      // Return jsonwebtoken

      const payload = {

        user: {

          id: user.id,

        },

      };

      jwt.sign(

        payload,

        config.get('jwtSecret'),

        { expiresIn: 360000 }, // @todo change this to 3600

        (err, token) => {

          if (err) throw err;

          res.json({ token });

        }

      );

    } catch (error) {

      console.error(error.message);

      res.status(500).send('💥Server Error');

    }

  }

);

* Authorization

router.post(

  '/',

  [

    check('name', 'Name is Required').not().isEmpty(),

    check('email', 'Please enter a valid email').isEmail(),

    check('college', 'Your current College is Required').not().isEmpty(),

    check(

      'password',

      'Please enter password with 6 or more characters'

    ).isLength({ min: 6 }),

  ],

  async (req, res) => {

    // Validation Result

    const errors = validationResult(req);

    if (!errors.isEmpty()) {

      return res.status(400).json({ errors: errors.array() });

    }

    const { name, email, college, password } = req.body;

    try {

      let user = await User.findOne({ email });

      // See if user exists

      if (user) {

        return res

          .status(400)

          .json({ errors: [{ msg: 'User already exists' }] });

      }

      user = new User({

        name,

        email,

        college,

        password,

      });

      // Encrypt password

      const salt = await bcrypt.genSalt(10);

      user.password = await bcrypt.hash(password, salt);

      await user.save();

      // Return jsonwebtoken

      const payload = {

        user: {

          id: user.id,

        },

      };

      jwt.sign(

        payload,

        config.get('jwtSecret'),

        { expiresIn: 360000 }, // @todo change this to 3600

        (err, token) => {

          if (err) throw err;

          res.json({ token });

        }

      );

    } catch (error) {

      console.error(error.message);

      res.status(500).send('💥Server Error');

    }

  }

);

* Post

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,

        user: req.user.id,

      });

      const post = await newPost.save();

      res.json(post);

    } catch (error) {

      console.error(error.message);

      res.status(500).send('Server Error 💥');

    }

  }

);

// get a post

router.get('/', auth, async (req, res) => {

  try {

    const posts = await Post.find().sort({ date: -1 });

    res.json(posts);

  } catch (error) {

    console.error(error.message);

    res.status(500).send('Server Error 💥');

  }

});

// get specific post by id

router.get('/:id', auth, async (req, res) => {

  try {

    const post = await Post.findById(req.params.id);

    if (!post) {

      return res.status(404).json({ msg: 'Post was not found' });

    }

    res.json(post);

  } catch (error) {

    console.error(error.message);

    if (error.kind === 'ObjectId') {

      return res.status(404).json({ msg: 'Post was not found' });

    }

    res.status(500).send('Server Error 💥');

  }

});

// delete a post

router.delete('/:id', auth, async (req, res) => {

  try {

    const post = await Post.findById(req.params.id);

    if (!post) {

      return res.status(404).json({ msg: 'Post was not found' });

    }

    // Check post user

    if (post.user.toString() !== req.user.id) {

      return res.status(401).json({ msg: 'User not authorized' });

    }

    await post.remove();

    res.json({ msg: 'Post removed' });

  } catch (error) {

    console.error(error.message);

    if (error.kind === 'ObjectId') {

      return res.status(404).json({ msg: 'Post was not found' });

    }

    res.status(500).send('Server Error 💥');

  }

});

// like post by id

router.put('/like/:id', auth, async (req, res) => {

  try {

    const post = await Post.findById(req.params.id);

    // check if post is already liked

    if (

      post.likes.filter((like) => like.user.toString() === req.user.id).length >

      0

    ) {

      return res.status(400).json({ msg: 'Post already liked' });

    }

    post.likes.unshift({ user: req.user.id });

    await post.save();

    res.json(post.likes);

  } catch (error) {

    console.error(error.message);

    res.status(500).send('Server Error 💥');

  }

});

// unlike

router.put('/unlike/:id', auth, async (req, res) => {

  try {

    const post = await Post.findById(req.params.id);

    // check if post is already liked

    if (

      post.likes.filter((like) => like.user.toString() === req.user.id)

        .length === 0

    ) {

      return res.status(400).json({ msg: 'Post has not yet been liked' });

    }

    // get remove index

    const removeIndex = post.likes.map((like) =>

      like.user.toString().indexOf(req.user.id)

    );

    post.likes.splice(removeIndex, 1);

    await post.save();

    res.json(post.likes);

  } catch (error) {

    console.error(error.message);

    res.status(500).send('Server Error 💥');

  }

});

// Add comment to a post

router.post(

  '/comment/:id',

  [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 post = await Post.findById(req.params.id);

      const newComment = {

        text: req.body.text,

        name: user.name,

        user: req.user.id,

      };

      post.comments.unshift(newComment);

      await post.save();

      res.json(post.comments);

    } catch (error) {

      console.error(error.message);

      res.status(500).send('Server Error 💥');

    }

  }

);

// Delete comment

router.delete('/comment/:id/:comment\_id', auth, async (req, res) => {

  try {

    const post = await Post.findById(req.params.id);

    const comment = post.comments.find(

      (comment) => comment.id === req.params.comment\_id

    );

    if (!comment) {

      return res.status(404).json({ msg: 'Comment does not exist' });

    }

    // check comment owner

    if (comment.user.toString() !== req.user.id) {

      return res.status(401).json({ msg: 'User not authorized' });

    }

    // get remove index

    const removeIndex = post.comments.map((comment) =>

      comment.user.toString().indexOf(req.user.id)

    );

    post.comments.splice(removeIndex, 1);

    await post.save();

    res.json(post.comments);

  } catch (error) {

    console.error(error.message);

    res.status(500).send('Server Error 💥');

  }

});

* Dashboard/Profile

router.get('/me', auth, async (req, res) => {

  try {

    const profile = await Profile.findOne({

      user: req.user.id,

    }).populate('user', ['name', 'college']);

    if (!profile) {

      return res.status(400).json({ msg: 'Profile not found' });

    }

    res.json(profile);

  } catch (error) {

    console.error(error.message);

    res.status(500).send('💥Server error');

  }

});

// Create or Update user profile

router.post(

  '/',

  [

    auth,

    [

      check('headline', 'Headline is required').not().isEmpty(),

      check('skills', 'Skills are required').not().isEmpty(),

    ],

  ],

  async (req, res) => {

    const errors = validationResult(req);

    if (!errors.isEmpty()) {

      return res.status(400).json({ error: errors.array() });

    }

    const {

      headline,

      about,

      location,

      website,

      skills,

      githubusername,

      twitter,

      facebook,

      linkedin,

    } = req.body;

    // Build profile object

    const profileFields = {};

    profileFields.user = req.user.id;

    if (headline) profileFields.headline = headline;

    if (about) profileFields.about = about;

    if (location) profileFields.location = location;

    if (website) profileFields.website = website;

    if (skills) {

      profileFields.skills = skills.split(',').map((skill) => skill.trim());

    }

    if (githubusername) profileFields.githubusername = githubusername;

    profileFields.social = {};

    if (twitter) profileFields.social.twitter = twitter;

    if (facebook) profileFields.social.facebook = facebook;

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

    try {

      let profile = await Profile.findOne({ user: req.user.id });

      if (profile) {

        // update

        profile = await Profile.findOneAndUpdate(

          { user: req.user.id },

          { $set: profileFields },

          { new: true }

        );

        return res.json(profile);

      }

      // if not found then create

      profile = new Profile(profileFields);

      await profile.save();

      res.json(profile);

    } catch (error) {

      console.error(error.message);

      res.status(500).send('💥Server Error');

    }

  }

);

// get all profiles

router.get('/', async (req, res) => {

  try {

    const profiles = await Profile.find().populate('user', ['name', 'college']);

    res.json(profiles);

  } catch (error) {

    console.error(error.message);

    res.status(500).send('💥Server Error');

  }

});

// get specific user profile by user\_id

router.get('/user/:user\_id', async (req, res) => {

  try {

    const profile = await Profile.findOne({

      user: req.params.user\_id,

    }).populate('user', ['name', 'college', 'email']);

    //  If profile does not exist

    if (!profile) {

      return res.status(400).send({ msg: 'Profile not found' });

    }

    res.json(profile);

  } catch (error) {

    console.error(error.message);

    if (error.kind == 'ObjectId') {

      return res.status(400).send({ msg: 'Profile not found' });

    }

    res.status(500).send('💥Server Error');

  }

});

// delete a profile by user id

router.delete('/', auth, async (req, res) => {

  try {

    //  remove user's post

    await Post.deleteMany({ user: req.user.id });

    // remove user profile

    await Profile.findOneAndRemove({ user: req.user.id });

    //  remove user

    await User.findOneAndRemove({ \_id: req.user.id });

    res.json({ msg: 'User removed' });

  } catch (error) {

    console.error(error.message);

    res.status(500).send('💥Server Error');

  }

});

// Add profile experience

router.put(

  '/experience',

  [

    auth,

    [

      check('title', 'Title is required').not().isEmpty(),

      check('company', 'Company is required').not().isEmpty(),

      check('from', 'From date is required and needs to be from the past')

        .not()

        .isEmpty()

        .custom((value, { req }) => (req.body.to ? value < req.body.to : true)),

    ],

  ],

  async (req, res) => {

    const errors = validationResult(req);

    if (!errors.isEmpty()) {

      return res.status(400).json({ errors: errors.array() });

    }

    const {

      title,

      company,

      location,

      from,

      to,

      current,

      description,

    } = req.body;

    const newExp = {

      title,

      company,

      location,

      from,

      to,

      current,

      description,

    };

    try {

      const profile = await Profile.findOne({ user: req.user.id });

      profile.experience.unshift(newExp);

      await profile.save();

      res.json(profile);

    } catch (err) {

      console.error(err.message);

      res.status(500).send('Server Error');

    }

  }

);

// Delete experience from profile

router.delete('/experience/:exp\_id', auth, async (req, res) => {

  try {

    const foundProfile = await Profile.findOne({ user: req.user.id });

    foundProfile.experience = foundProfile.experience.filter(

      (exp) => exp.\_id.toString() !== req.params.exp\_id

    );

    await foundProfile.save();

    return res.status(200).json(foundProfile);

  } catch (error) {

    console.error(error);

    return res.status(500).json({ msg: 'Server error' });

  }

});

// Add profile education

router.put(

  '/education',

  [

    auth,

    [

      check('school', 'School is required').not().isEmpty(),

      check('degree', 'Degree is required').not().isEmpty(),

      check('fieldofstudy', 'Field of study is required').not().isEmpty(),

      check('from', 'From date is required and needs to be from the past')

        .not()

        .isEmpty()

        .custom((value, { req }) => (req.body.to ? value < req.body.to : true)),

    ],

  ],

  async (req, res) => {

    const errors = validationResult(req);

    if (!errors.isEmpty()) {

      return res.status(400).json({ errors: errors.array() });

    }

    const {

      school,

      degree,

      fieldofstudy,

      from,

      to,

      current,

      description,

    } = req.body;

    const newEdu = {

      school,

      degree,

      fieldofstudy,

      from,

      to,

      current,

      description,

    };

    try {

      const profile = await Profile.findOne({ user: req.user.id });

      profile.education.unshift(newEdu);

      await profile.save();

      res.json(profile);

    } catch (err) {

      console.error(err.message);

      res.status(500).send('Server Error');

    }

  }

);

// Delete education from profile

router.delete('/education/:edu\_id', auth, async (req, res) => {

  try {

    const foundProfile = await Profile.findOne({ user: req.user.id });

    foundProfile.education = foundProfile.education.filter(

      (edu) => edu.\_id.toString() !== req.params.edu\_id

    );

    await foundProfile.save();

    return res.status(200).json(foundProfile);

  } catch (error) {

    console.error(error);

    return res.status(500).json({ msg: 'Server error' });

  }

});