CS 295R
Lab 7 – Blog Application
EditUserProfile and EditPost Pages

The objective of this lab is to introduce you to using React Hook Form in a React application. React Hook Form is one of the most popular open-source forms libraries for React. After watching Sections 10 - 12 of Modern React and experimenting with 2 React Hook Form tutorials, you'll add complex forms to the Blog application from the previous lab. Specifically, you will complete the EditUserProfile and EditPost pages for the application.

Setting Up

1. Add react-hook-form to the project.

2. You may choose to use other 3rd party form controls such as react-quil (html editor), react-datepicker and/or react-select. In order to use those controls in your application you will need to install each of them as well.

Creating the EditUserProfile Page

The EditUserProfile page is accessible from a NavBar button when a user is logged in to the application.

When the page loads it should display the current information for the logged in user including: an image of the user and a file picker control that allows the user to change their profile picture, a textbox containing the user's name, a textbox containing the userid, a textbox containing the user's email address, a text area containing the user's bio.

You might also choose to display: a read-only control that contains the user's id, a read-only control that contains the user's password, 2 textbox controls that allow the user to enter and repeat a new password.

All data should be validated in a reasonable way and a reasonable error message should be displayed for each piece of data that is not valid. The user should not be able to submit the form until all of the data in the form is valid.

Pressing the Save button on the page should save the user's data to context and to the db.json file using the functions in UserContext. It should also return the user to the home page.

3. Create the EditUserProfile page component.

    a. Create JSX that renders the controls that you'll need on the page without adding default values, validation or logic related to form submission. Test the look and feel of the form.

    b. Add react hook form to the page and all of the code that handles default or initial values, validation, error messages and form submission for all of the fields BUT the image of the user. Console.log the data entered by the user when the save button is pressed. Remember that the user information is available from the UserContext. Test the functionality of the form.

c. Develop a strategy for allowing the user to see their image as well as to change their image and convert their image to a base 64 string. Implement and test image handling. Below are several code "snippets" from my implementation that may help.

```
const [image, setImage] = useState(user.image);

function convertImageToBase64(imgUrl, callback) {
    const image = new Image();
    image.crossOrigin='anonymous';
    image.onload = () => {
      const canvas = document.createElement('canvas');
      const ctx = canvas.getContext('2d');
      canvas.height = image.naturalHeight;
      canvas.width = image.naturalWidth;
      ctx.drawImage(image, 0, 0);
      const dataUrl = canvas.toDataURL();
      callback && callback(dataUrl)
    }
    image.src = imgUrl;
}

const handleFileChange = (event) => {
     const file = URL.createObjectURL(event.target.files[0]);
     convertImageToBase64(file, removeTypeAndSave)
 }

const removeTypeAndSave = (base64Image) => {
     const updatedImage = base64Image.replace(
"data:image/png;base64,", "" );
     setImage(updatedImage);
 }
```

d. Finish the implementation of the form submission function expression. You'll need to create a new json object that contains all of the properties of the user in the db.json file excluding the id. Calling editUserById with the new user object and the id will save the user information to the db.json file and update the UserContext. The function navigate from the useNavigate from react-router-dom will allow you to return the user to the home page after the form has been submitted.

Creating the EditPost Page

The EditPost page is used for both editing an existing post and creating a new post. It is accessed from a variety of navigation elements in the application.

When editing a post, the page should display the current information for the current post including: an image for the post and a file picker control that allows the user to change the picture, a date picker containing the date the post was written, a textbox containing the title of the post, a dropdown containing the list of category names and an html editor component that contains the content of the post.

You might also choose to display: a read-only control that contains the post id, a read-only control that contains the id of the user who created the post.

The same controls should be displayed when the page is being used for a new post. All of the controls will be empty however.

All data should be validated in a reasonable way and a reasonable error message should be displayed for each piece of data that is not valid. The user should not be able to submit the form until all of the data in the form is valid.

Pressing the Save button on the page should save the post data to context and to the db.json file using the functions in PostContext. It should also return the user to the home page.

4. Create the EditPost page component.
    a. Create JSX that renders the controls that you'll need on the page without adding default values, validation or logic related to form submission. Test the look and feel of the form.
    b. Add react hook form to the page and all of the code that handles default or initial values, validation, error messages and form submission for all of the fields BUT the image for the post and the content for the post. Console.log the data entered by the user when the save button is pressed. Remember that the post information is available from location.state or from PostsContext based on the id in the route parameter. Test the functionality of the form.
    c. Repeat the process you designed and implemented for the user profile picture for the post picture. You may choose to copy and paste code from one component to the other. You might also choose to create a custom component that incorporates all of the image functionality.
    d. Add the html editor component to the page. I've provided a tutorial in moodle that illustrates how to add a relatively simple and fully featured html editor called react-quill.
    e. Finish the implementation of the form submission function expression. You'll need to create a new json object that contains all of the properties of the post in the db.json file excluding the id. Calling editPostById with the new post object and the id will save the user information to the db.json file and update the UserContext. Calling createPost with the new post object and the logged in user will add the new post to db.json file and update the PostsContext. The function navigate from the useNavigate from react-router-dom will allow you to return the user to the home page after the form has been submitted.

5. Create a production version of the app. Deploy the production version to citstudent.
    a. You already have .env file for the project. You already set up an instance of json-server for this application on citweb.
    b. Change the value of the environment variable REACT_APP_SERVER_URL to use the port on citweb.lanecc.net.
    c. Create a production version of the app in the usual way.
    d. Test the application from the build folder on your machine. This version will be using the instance of json-server running on citweb.
    e. Deploy the contents of the build folder to citstudent and test the application again.