Lesson 5-Building RESTful APIs with MERN Stack and Redux Toolkit

Activity 9 –PostIT App – User Profile

The objectives of this activitiy are to:

a) Implement updating of user profile.
b) Write the code to upload profile photo.

IMPLEMENTING UPDATE PROFILE

SERVER-SIDE

1. Index server/index.js, create a PUT API, for updating the user profile. Extract the email from the route parameter.

```
app.put("/updateUserProfile/: email/", async (req, res) => {
    //Retrieve the value from the route
    const email= req.params.email;

    try {
    } catch (err) {
      res.status(500).json({ error: err });
      return;
    }
  });
```

2. Retrieve the values from the request body.

```
app.put("/updateUserProfile/:email/", async (req, res) => {
    //Retrieve the value from the route
    const email = req.params.email;
    //Retrieve the values from the request body.
    const name = req.body.name;
    const password = req.body.password;

    try {
    } catch (err) {
      res.status(500).json({ error: err });
      return;
    }
  });
```

3. Search for the user that will be updated using the findOne method and check if the user is found. If it’s not found, the server responds with a 404 status code, indicating that the requested resource was not found. The json() method sends a JSON response back to the client. The response includes an object with an error property containing a message like "User not found".

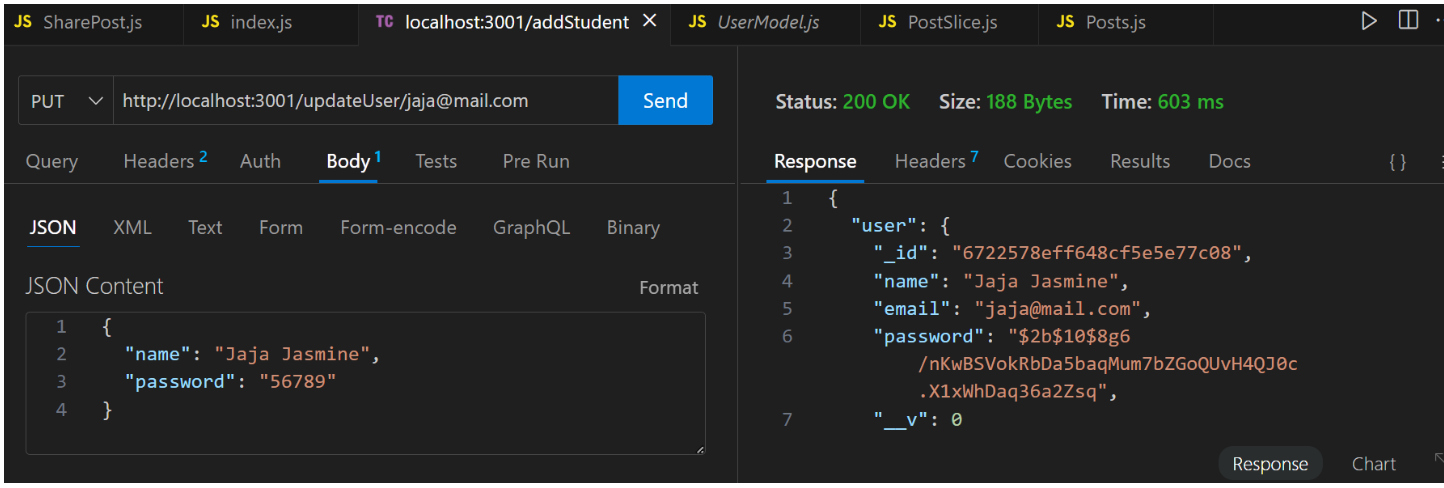
```
app.put("/updateUserProfile/:email/", async (req, res) => {
    //Retrieve the value from the route
    const email = req.params.email;
    //Retrieve the values from the request body.
    const name = req.body.name;
    const password = req.body.password;

    try {
      // Search for the user that will be updated using the findOne method
      const userToUpdate = await UserModel.findOne({ email: email });

      // Check if the user was found
      if (!userToUpdate) {
        return res.status(404).json({ error: "User not found" });
      }
    } catch (err) {
      // Handle errors, including database or validation issues
      res.status(500).json({ error: err.message }); // Send a more descriptive
error message
    }
  });
```

4. If the user is found, update both the name and password from the request body. For the password, check whether it has been changed. If it has, hash the new password before saving it. Finally, execute the save method and return the updated user information.

   The **await** keyword is used to wait for a Promise to resolve. In this case, it waits for the save() method to complete.

```
app.put("/updateUserProfile/:email/", async (req, res) => {
    …

      try {
    …
        if (!userToUpdate) {
          return res.status(404).json({ error: "User not found" });
        }

        // Update the user's name
        userToUpdate.name = name;

        //if the user changed the password, change the password in the Db to the
new hashed password
        if (password !== userToUpdate.password) {
          const hashedpassword = await bcrypt.hash(password, 10);
          userToUpdate.password = hashedpassword;
        } else {
          //if the user did not change the password
          userToUpdate.password = password;
        }

        // Save the updated user
        await userToUpdate.save(); // Make sure to save the changes

        // Return the updated user as a response
        res.send({ user: userToUpdate, msg: "Updated." });
      } catch (error) {
        // Handle errors, including database or validation issues
        res.status(500).json({ error: error.message }); // Send a more
descriptive error message optional
      }
    });
```

5. Test your API in Thunder Client.



```
PUT http://localhost:3001/updateUser/jaja@mail.com
```

```
{
  "name": "Jaja Jasmine",
  "password": "56789"
}
```

Status: 200 OK Size: 188 Bytes Time: 603 ms

```
{
  "user": {
    "_id": "6722578eff648cf5e5e77c08",
    "name": "Jaja Jasmine",
    "email": "jaja@mail.com",
    "password": "$2b$10$8g6
        /nKwBSVokRbDa5baqMum7bZGoQUvH4QJ0c
        .X1xWhDaq36a2Zsq",
    "__v": 0
```

CLIENT SIDE

6. In the Profile component, import the ReactStrap components.

```
import {
    Form,
    FormGroup,
    Input,
    Label,
    Button,
    Container,
    Row,
    Col,
  } from "reactstrap";
```

7. Use the Reactstrap Layout component to implement the following:

User Component	Update Form

```
…
const Profile = () => {
….
  return (
    <Container fluid>
      <h1>Profile</h1>
      <Row>
        <Col md={2}>
          <User />
        </Col>
        <Col md={4}>Update Profile</Col>
      </Row>
    </Container>
  );
};

export default Profile;
```

8. Create the form for updating the user profile.

```
…
  return (
    <Container fluid>
…
        <Col md={4}>
          Update Profile
          <Form>
            <input type="file" name="profilePic" />
            <div className="appTitle"></div>
            Update Profile
            <FormGroup>
              <Label for="name">Name</Label>
              <Input id="name" name="name" placeholder="Name..." type="text" />
            </FormGroup>
            <FormGroup>
              <Label for="email">Email</Label>
              <Input
                id="email"
                name="email"
                placeholder="Email..."
                type="email"
              />
            </FormGroup>
            <FormGroup>
              <Label for="password">Password</Label>
              <Input
                id="password"
                name="password"
                placeholder="Password..."
                type="password"
              />
            </FormGroup>
            <FormGroup>
              <Label for="confirmPassword">Confirm Password</Label>
              <Input
                id="confirmPassword "
                name="confirmPassword"
                placeholder="Confirm Password..."
                type="password"
              />
            </FormGroup>
            <FormGroup>
              <Button color="primary" className="button">
                Update Profile
              </Button>
            </FormGroup>
          </Form>
        </Col>
      </Row>
    </Container>
  );
```

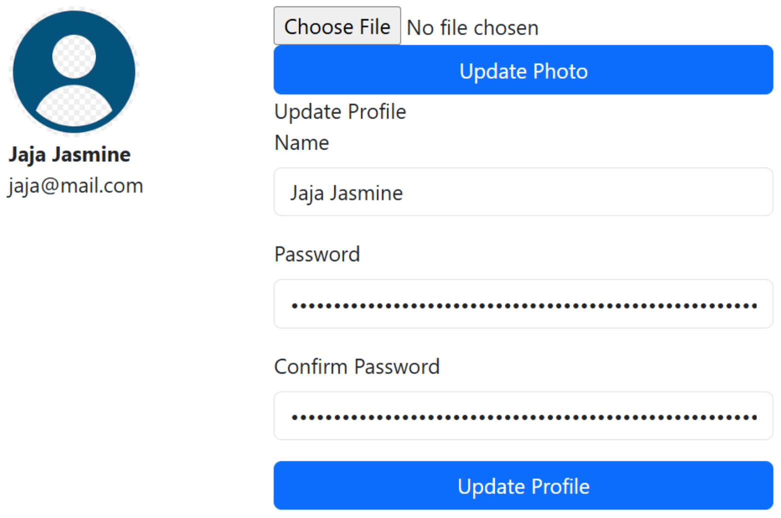
9. Use the **useSelector** hook from React Redux to retrieve the email, name, and password from the Redux state of the current logged user. After retrieving the user details from the Redux store, create state variables and initialize these state variables with the values retrieved from the Redux store.

```
const Profile = () => {
    //Retrieve the user details from Redux store.
  const user = useSelector((state) => state.users.user);

    // Create state variables for user input and assign initial values from the
Redux store

    const [userName, setUserName] = useState(user.name);
    const [pwd, setPwd] = useState(user.password);
    const [confirmPassword, setConfirmPassword] = useState(user.password);
    const [profilePic, setProfilePic] = useState(user.profilePic);
…..
}
```

10. For the form elements, add the value prop and assign the {userName} state. By doing this, you are making a controlled input component, ensuring that the input reflects the state and that the state reflects user input. Do this for all input elements.

```
            <Input
                id="name"
                name="name"
                placeholder="Name..."
                type="text"
                value={userName}
                onChange={(e) => setUserName(e.target.value)}
              />
```

By doing these steps, the form will be populated with the user data.



Jaja Jasmine
jaja@mail.com

Update Profile
Name
Jaja Jasmine
Password
••
Confirm Password
••

11. In **Features/UserSlice.js,** create a new thunk for updating user profile.

```
// Define an async thunk to update the user profile in the Redux store
export const updateUserProfile = createAsyncThunk(
    "user/updateUserProfile", // Action type string for Redux
    async (userData) => {
      try {
        // Log the user data being sent for debugging purposes
        // console.log(userData);

        // Send a PUT request to the server to update the user profile
        const response = await axios.put(
          `http://localhost:3001/updateUserProfile/${userData.email}`, // API
endpoint for updating user profile
          {
            // Request payload with user data to be updated
            email: userData.email,
            name: userData.name,
            password: userData.password,
            profilePic: userData.profilePic,
          },
          {
            headers: {  //headers is necessary when uploading files with form-
data in a request.
              "Content-Type": "multipart/form-data",
            },
          }
        );
        // Extract the updated user data from the server response
        const user = response.data.user;

        // Return the updated user data, which will be used by Redux to update
the state
        return user;
      } catch (error) {
        // Log any errors that occur during the request
        console.log(error);
      }
    }
  );
```

12. In the userSlice, add a new extraReducer to handle the updateUserProfile thunk.

```
export const userSlice = createSlice({
    extraReducers: (builder) => {
      …
        .addCase(updateUserProfile.pending, (state) => {
          state.isLoading = true;
        })
        .addCase(updateUserProfile.fulfilled, (state, action) => {
          state.user = action.payload;
          state.isLoading = false;
        })
        .addCase(updateUserProfile.rejected, (state) => {
          state.isLoading = false;
          state.isError = true;
        });
    },
  })
```

13. In the Profile component, import:

```
import { updateUserProfile } from "../Features/UserSlice";
```

14. Create variable for useNavigate and useDispatch hooks.

```
const Profile = () => {
    …
      const navigate = useNavigate();
      const dispatch=useDispatch();
    ….
```

15. Create a new function **handleUpdate** to manage the update process.

```
const Profile = () => {
    …
      // Function to handle updating the user profile
    const handleUpdate = (event) => {
        // Prevent the default form submission behavior
        event.preventDefault();

        // Prepare the user data object with the current user's email and updated
details
        const userData = {
          email: user.email, // Retrieve email from the Redux store
          name: userName, // Get the updated name from the state variable
          password: pwd, // Get the updated password from the state variable
          profilePic: profilePic,
        };
        console.log(userData);
        // Dispatch the updateUserProfile action to update the user profile in
the Redux store
        dispatch(updateUserProfile(userData));
        alert("Profile Updated.");
        // Navigate back to the profile page after the update is completed
        navigate("/profile");
      };
      return (
    ….
    )
```

16. In the update form, invoke the handleUpdate function within the onSubmit event handler.

```
<Form onSubmit={handleUpdate}></Form>
```

UPDATING PROFILE PICTURE

CLIENT SIDE

17. In **client/src/Profile.js**, add a new function to handle the selection of the file.

```
  const handleUpdate = (event) => {
 …..
```

```
  // Function to handle file input change
  const handleFileChange = (event) => {
    // Use e.target.files[0] to get the file itself
    const uploadFile = event.target.files[0];
    if (!uploadFile) alert("No file uploaded");
    else setProfilePic(event.target.files[0]);
  };
…..
return (
….)
```

SERVER SIDE

18. Index **server/index.js**, install **multer**. Multer is a middleware for handling file uploads in Node.js, specifically for use with the express framework. It makes it easy to handle multipart/form-data requests, which are essential for uploading files, images, or other media content to the server.

    npm install multer

19. Import the following in index.js. The **fs** module provides API for interacting with the file system in a way that allows you to read, write, and manipulate files and directories. The **path** module provides utilities for working with file and directory paths.

```
import multer from "multer";
import fs from "fs";
import path from "path";
```

20. Setup multer to handle the file uploaded and save the file in the uploads folder in the server.

```
….
import multer from "multer";

const app = express();
app.use(express.json());
app.use(cors());

//Database connection
….
// Set up multer for file storage
const storage = multer.diskStorage({
  destination: (req, file, cb) => {
    cb(null, "uploads/"); // Specify the directory to save uploaded files
  },
  filename: (req, file, cb) => {
    cb(null, Date.now() + "-" + file.originalname); // Unique filename
  },
});
// Create multer instance
const upload = multer({ storage: storage });
```

21. Update the **updateUserProfile** API to handle the uploading of profile picture using multer. Before updating the profile picture, you first check if there is a file uploaded. If there, is then check if there is an existing profile picture of the user. If there is, delete first the original profile picture from the uploads folder.

```
app.put(
    "/updateUserProfile/:email/",
    upload.single("profilePic"), // Middleware to handle single file upload
    async (req, res) => {
      const email = req.params.email;
      const name = req.body.name;
      const password = req.body.password;

      try {
        // Find the user by email in the database
        const userToUpdate = await UserModel.findOne({ email: email });

        // If the user is not found, return a 404 error
        if (!userToUpdate) {
          return res.status(404).json({ error: "User not found" });
        }
        // Check if a file was uploaded and get the filename
        let profilePic = null;
        if (req.file) {
          profilePic = req.file.filename; // Filename of uploaded file
          // Update profile picture if a new one was uploaded but delete first
the old image
```

```
          if (userToUpdate.profilePic) {
            const oldFilePath = path.join(
              __dirname,
              "uploads",
              userToUpdate.profilePic
            );
            fs.unlink(oldFilePath, (err) => {
              if (err) {
                console.error("Error deleting file:", err);
              } else {
                console.log("Old file deleted successfully");
              }
            });
            userToUpdate.profilePic = profilePic; // Set new profile picture path
          }
        } else {
          console.log("No file uploaded");
        }

        // Update user's name
        userToUpdate.name = name;

        // Hash the new password and update if it has changed
        if (password !== userToUpdate.password) {
          const hashedPassword = await bcrypt.hash(password, 10);
          userToUpdate.password = hashedPassword;
        } else {
          userToUpdate.password = password; // Keep the same password if
unchanged
        }

          // Save the updated user information to the database
        await userToUpdate.save();

        // Send the updated user data and a success message as a response
        res.send({ user: userToUpdate, msg: "Updated." });
      } catch (err) {
        // Handle any errors during the update process
        res.status(500).json({ error: err.message });
      }
    }
  );
```

SHOWING STATIC FILES

22. Static files are assets that a web server delivers to clients (like browsers) without any modification or processing. To serve static files from the uploads directory, you can use the express.static middleware. Import the following functions **fileURLToPath** and **dirname**. The url and path are built-in modules in Node and thus, there is no need to install these.

```
import { fileURLToPath } from "url";
import { dirname } from "path";
```

23. Set up static files in Express. In this project, the images that are uploaded will be saved in the uploads folder in the server.

```
…
import { fileURLToPath } from "url";
import { dirname } from "path";

const app = express();
app.use(express.json());
app.use(cors());

//Database connection
…

// Serve static files from the 'uploads' directory

// Convert the URL of the current module to a file path
const __filename = fileURLToPath(import.meta.url);

// Get the directory name from the current file path
const __dirname = dirname(__filename);

// Set up middleware to serve static files from the 'uploads' directory
// Requests to '/uploads' will serve files from the local 'uploads' folder
app.use("/uploads", express.static(__dirname + "/uploads"));

// Set up multer for file storage
…
```

The code **const __filename = fileURLToPath(import.meta.url),** retrieves the path of the current module, for example, the index.js file located at:

D:\react\FullStack\postitapp\server\index.js.

On the other hand, the **code const __dirname = dirname(__filename**), provides the directory path of that module, which is:
D:\react\FullStack \postitapp\server.

Thus, the code **app.use("/uploads", express.static(__dirname + "/uploads")),** sets up Express to serve static files from the uploads directory located within the server's directory path.

SHOWING THE IMAGE IN THE BROWSER

24. In the User Component, update the code to retrieve profilePic of the currently logged user.

```
import user from "../Images/user.png";
import { useSelector } from "react-redux";

const User = () => {
  const user = useSelector((state) => state.users.user);
  const picURL = "http://localhost:3001/uploads/" + user.profilePic;

  return (
    <div>
      <img src={picURL} className="userImage" />
      <p>
        <b>{user.name}</b>
        <br />
        {user.email}
      </p>
    </div>
  );
};

export default User;
```