Front End

Setup

1. Edit the root `package.json`

Edit the "scripts" object to the following:

```
"scripts": {
    "client-install": "npm install --prefix client",
    "start": "node server.js",
    "server": "nodemon server.js",
    "client": "npm start --prefix client",
    "dev": "concurrently \"npm run server\" \"npm run client\""
  },
```

We'll use `concurrently` to run both our backend and frontend (client) at the same time. We'll use npm run dev to run this command later on.

2. Scaffold client with `create-react-app`

Create a client directory and run create-react-app within it.

```
mkdir client
cd client
create-react-app .
```

3. Change `package.json` within `client` directory

When we make requests from React with `axios`, we don't want to have to do the following in our requests:

```
axios.post('http://localhost:5000/api/users/register');
```

We want to be able to do the following instead.

```
axios.post('/api/users/register');
```

To achieve this, add the following under the `"scripts"` object in client's `package.json`.

```
"proxy": "http://localhost:5000",
```

4. Within `client`, install the following dependencies

```
npm i axios classnames jwt-decode react-redux react-router-dom redux redux-thunk
```

A brief description of each package and the function it will serve

- `axios`: promise-based HTTP client for making requests to our back end
- `classnames`: used for conditional classes in our JSX
- `jwt-decode`: used to decode our jwt so we can get user data from it
- `react-redux`: allows us to use Redux with React
- `react-router-dom`: used for routing
- `redux`: used to manage state between components (can be used with React or any other view library)
- `redux-thunk`: middleware for Redux that allows us to directly access the dispatch method to make asynchronous calls from our actions

5. Run `npm run dev` and test if both server and client run concurrently and successfully

6. Clean up React app by removing unnecessary files and code

- Remove `logo.svg` in `client/src`
- Take out the import of `logo.svg` in `App.js`
- Remove all the CSS in `App.css` (keep the import in App.js in case you want to add your own global CSS here)
- Clear out the content in the main `div` in `App.js` and replace it with an `<h1>` for now

You should have no errors at this point.

7. Install `Materialize.css` by editing `index.html` in `client/public`

Go to materializecss.com. Navigate to the CDN portion and grab the CSS and Javascript tags.

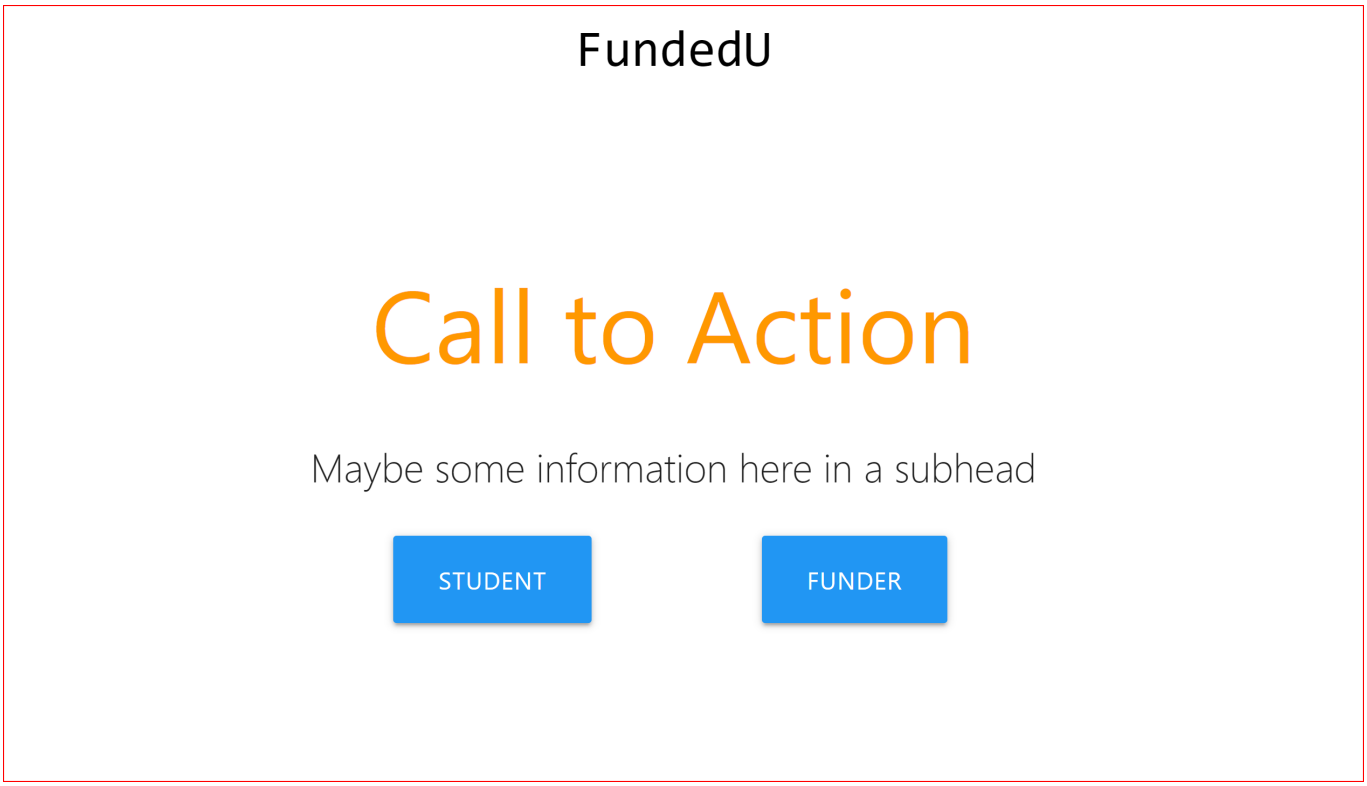
In `client/public/index.html`, add the CSS tag above the `</head>` tag and the JS script right above the `</body>` tag. Change the `<title>` from “React App” to "FundedU" while we’re here.

Add the following CSS tag under the Materialize tag for access to Google’s Material Icons.

```
<link href="https://fonts.googleapis.com/icon?family=Material+Icons"
rel="stylesheet">
```

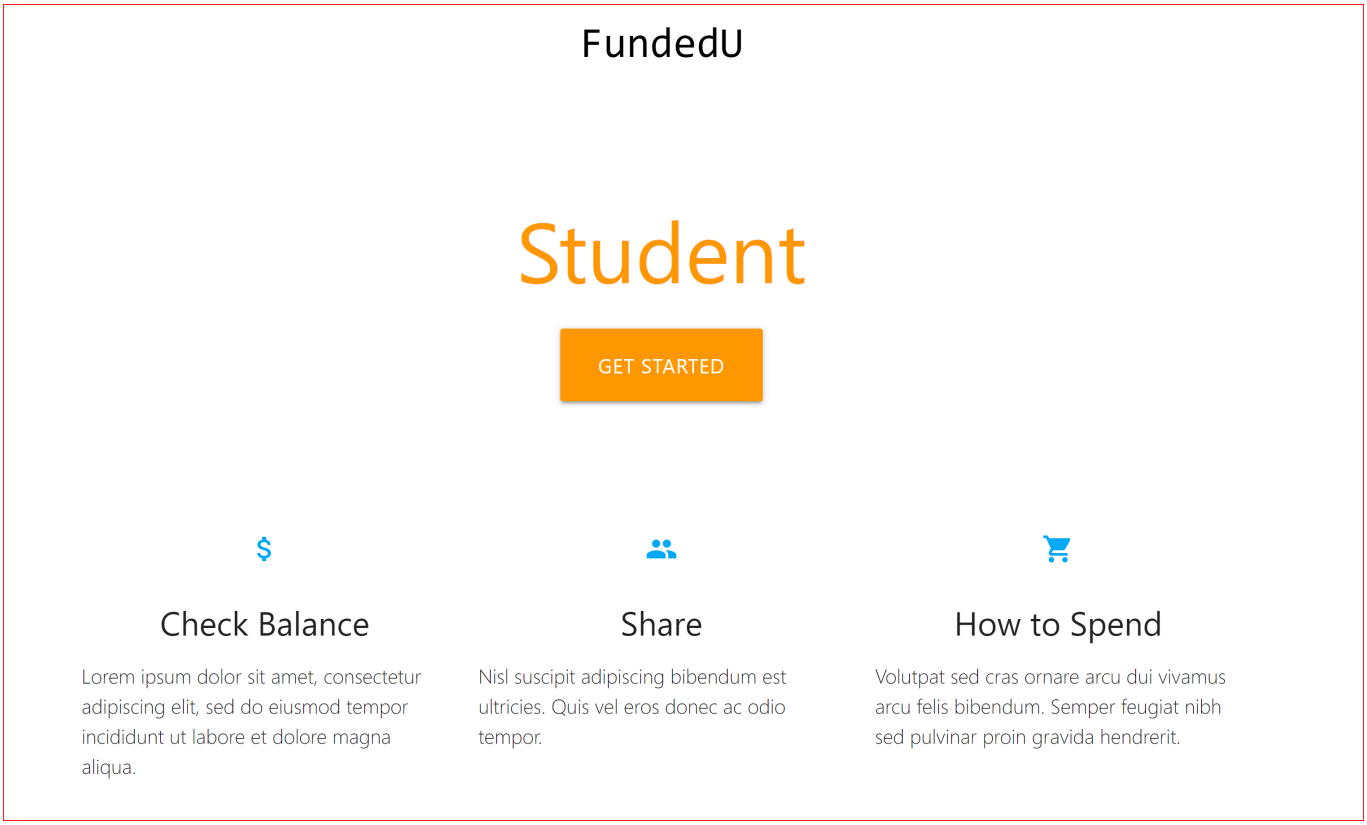
Creating static components

Home (with Navbar)



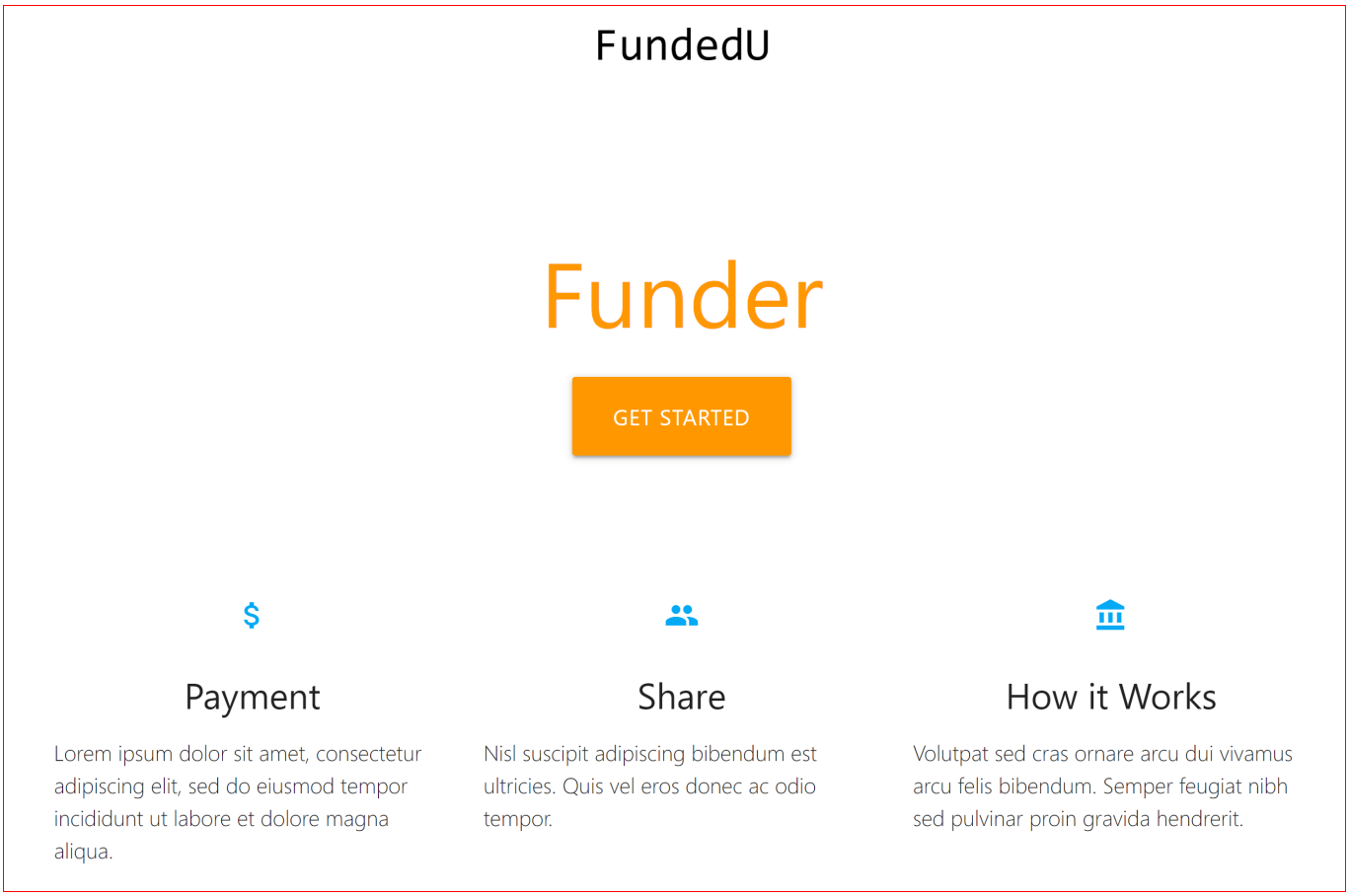
Buttons go to Student Main and Funder Main

Student Main



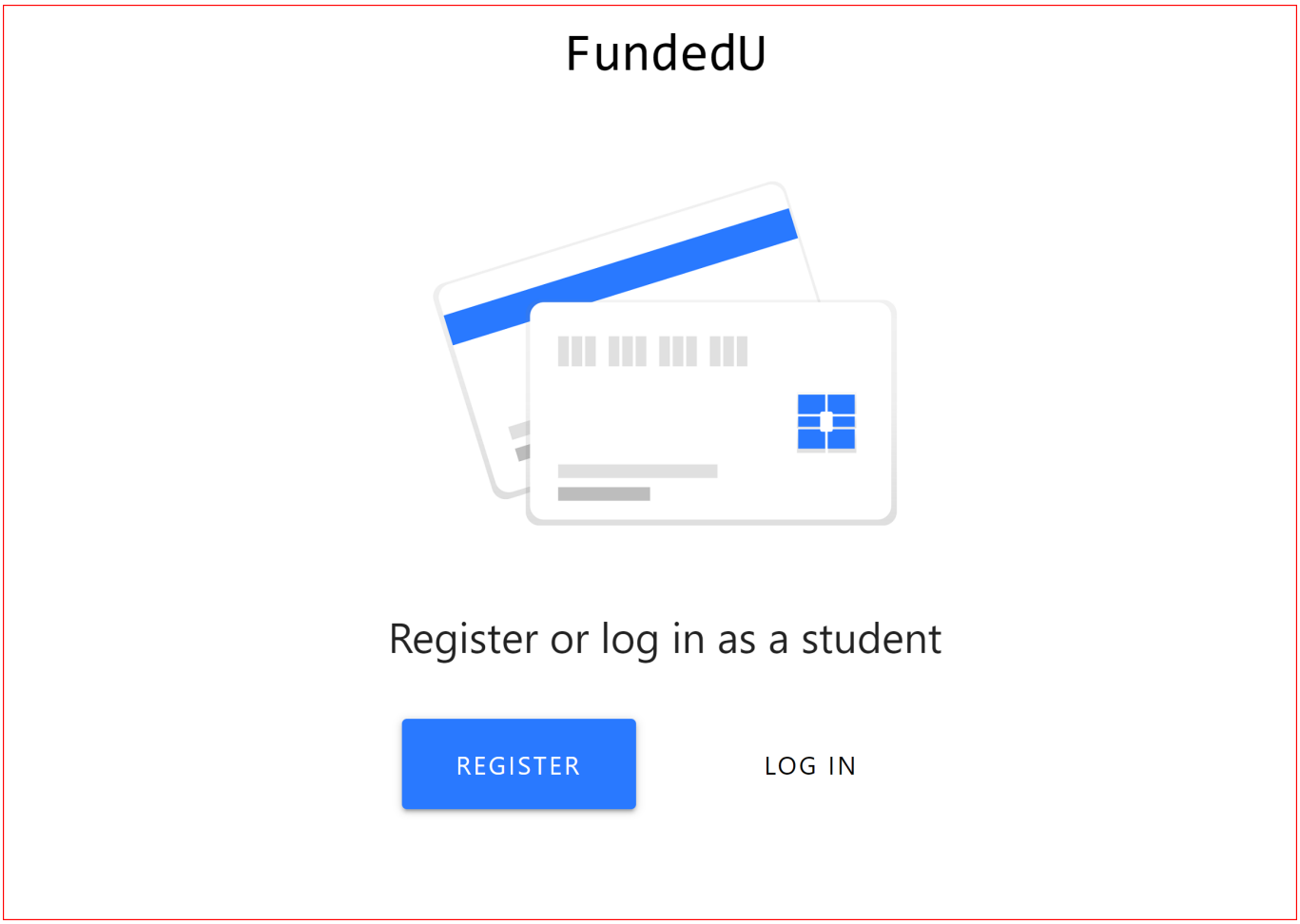
Button goes to Student Landing

Funder Main



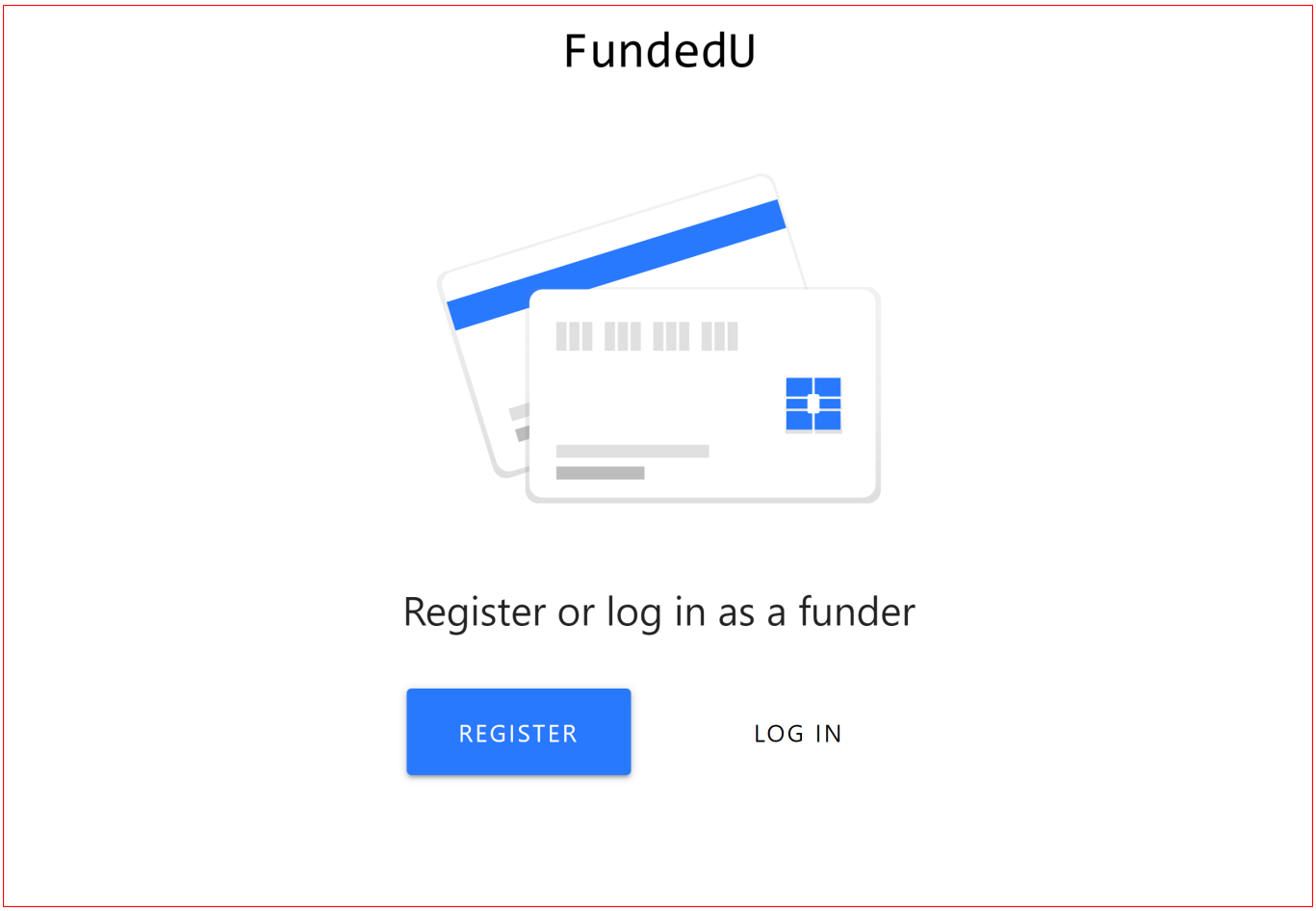
Button goes to Funder Landing

Student Landing



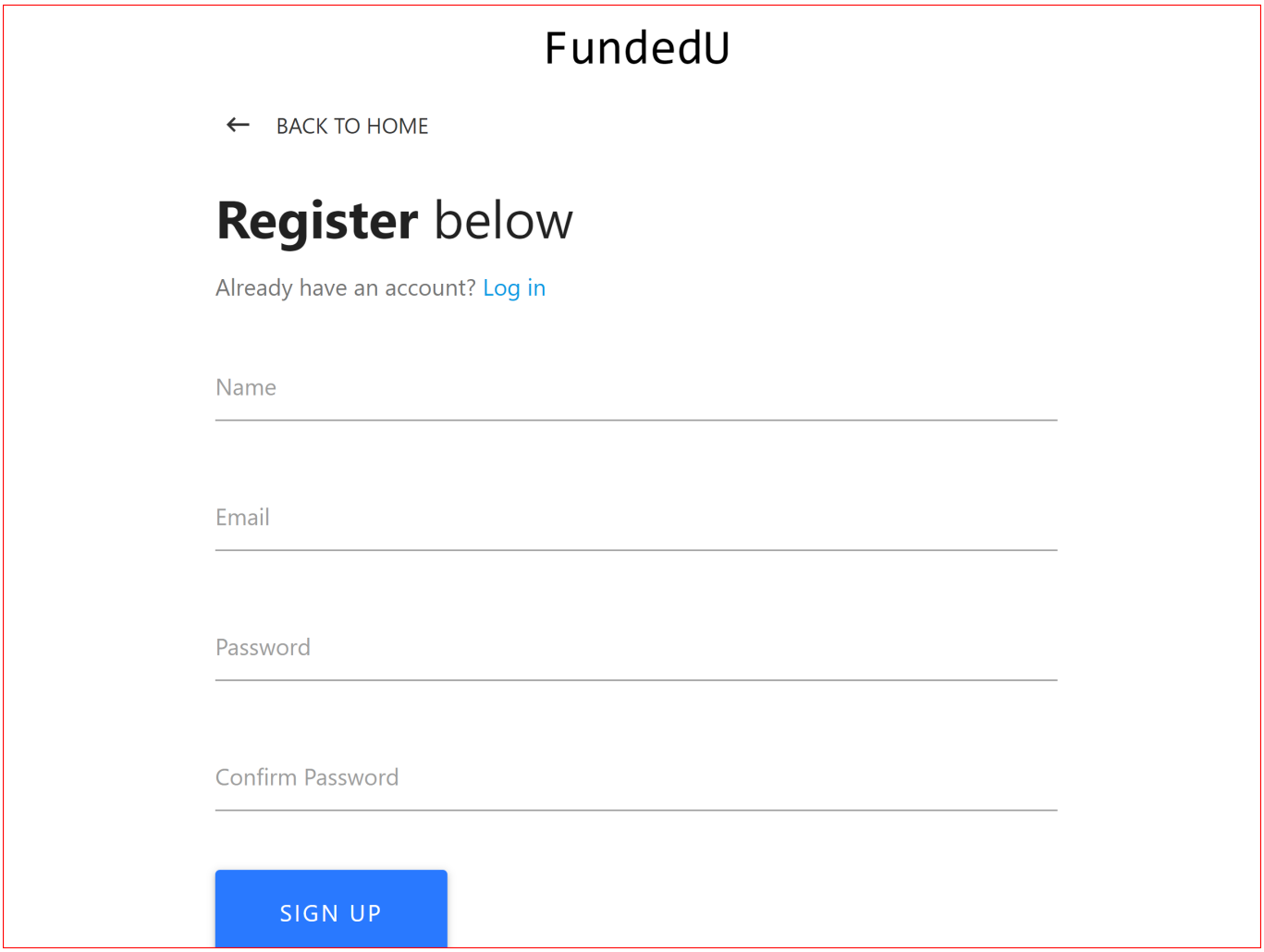
Button goes to Student Landing

Funder Landing

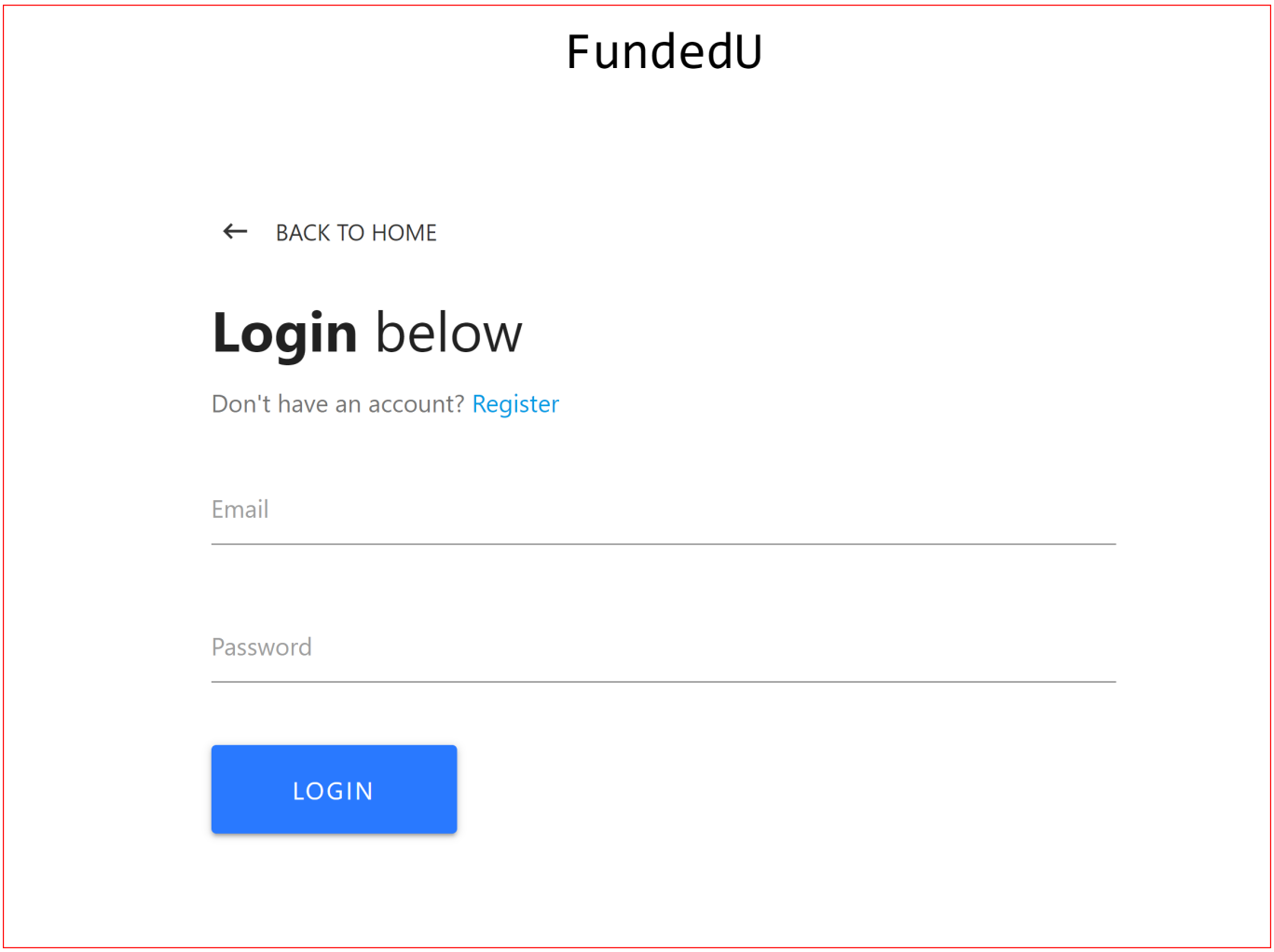


Button goes to Funder Landing

Register


FundedU
BACK TO HOME
Register below
Already have an account? Log in
Name
Email
Password
Confirm Password
SIGN UP

Login



Creating reducers

authReducer.js

```
import { SET_CURRENT_USER } from "../actions/types";

const isEmpty = require("is-empty");

const initialState = {
  isAuthenticated: false,
  user: {}
};

export default function(state = initialState, action) {
  switch (action.type) {
    case SET_CURRENT_USER:
      return {
        ...state,
        isAuthenticated: !isEmpty(action.payload),
        user: action.payload
      };
    default:
      return state;
```

```
  }
}
```

Setting `auth` token

Before creating actions, let's create a `utils` directory within `src`, and within it, a `setAuthToken.js` file.

We'll use this to set and delete the Authorization header for our axios requests depending on whether a user is logged in or not

Place the following in `setAuthToken.js`:

```
import axios from "axios";

const setAuthToken = token => {
  if (token) {
    // Apply authorization token to every request if logged in
    axios.defaults.headers.common["Authorization"] = token;
  } else {
    // Delete auth header
    delete axios.defaults.headers.common["Authorization"];
  }
};

export default setAuthToken;
```

Creating actions

Our general flow for our actions will be as follows:

- Import dependencies and action definitions from `types.js`
- Use axios to make HTTP requests
- Use `dispatch` to send actions to reducers

Place the following in `authActions.js`:

```
import axios from "axios";
import setAuthToken from "../utils/setAuthToken";
import jwt_decode from "jwt-decode";

import { GET_ERRORS, SET_CURRENT_USER } from "./types";

// Register User
export const registerUser = (userData, history) => dispatch => {
  axios
    .post("/api/users/register", userData)
    .then(res => history.push("/login"))
    .catch(err =>
      dispatch({
```

```
        type: GET_ERRORS,
        payload: err.response.data
      })
    );
};

// Login - get user token
export const loginUser = userData => dispatch => {
  axios
    .post("/api/users/login", userData)
    .then(res => {
      // Save to localStorage

      // Set token to localStorage
      const { token } = res.data;
      localStorage.setItem("jwtToken", token);
      // Set token to Auth header
      setAuthToken(token);
      // Decode token to get user data
      const decoded = jwt_decode(token);
      // Set current user
      dispatch(setCurrentUser(decoded));
    })
    .catch(err =>
      dispatch({
        type: GET_ERRORS,
        payload: err.response.data
      })
    );
};

// Set logged in user
export const setCurrentUser = decoded => {
  return {
    type: SET_CURRENT_USER,
    payload: decoded
  };
};

// Log user out
export const logoutUser = () => dispatch => {
  // Remove token from local storage
  localStorage.removeItem("jwtToken");
  // Remove auth header for future requests
  setAuthToken(false);
  // Set current user to empty object {} which will set isAuthenticated to false
  dispatch(setCurrentUser({}));
};
```