**React Workshop Follow along document**

The purpose of this document is to serve as a guide to create a React application by offering step by step directions on how to get started, along with some explanations of the React and Javascript concepts that are being leveraged along the way.

**Prerequisites**

* Npm and npx
* Visual Studio code
* React

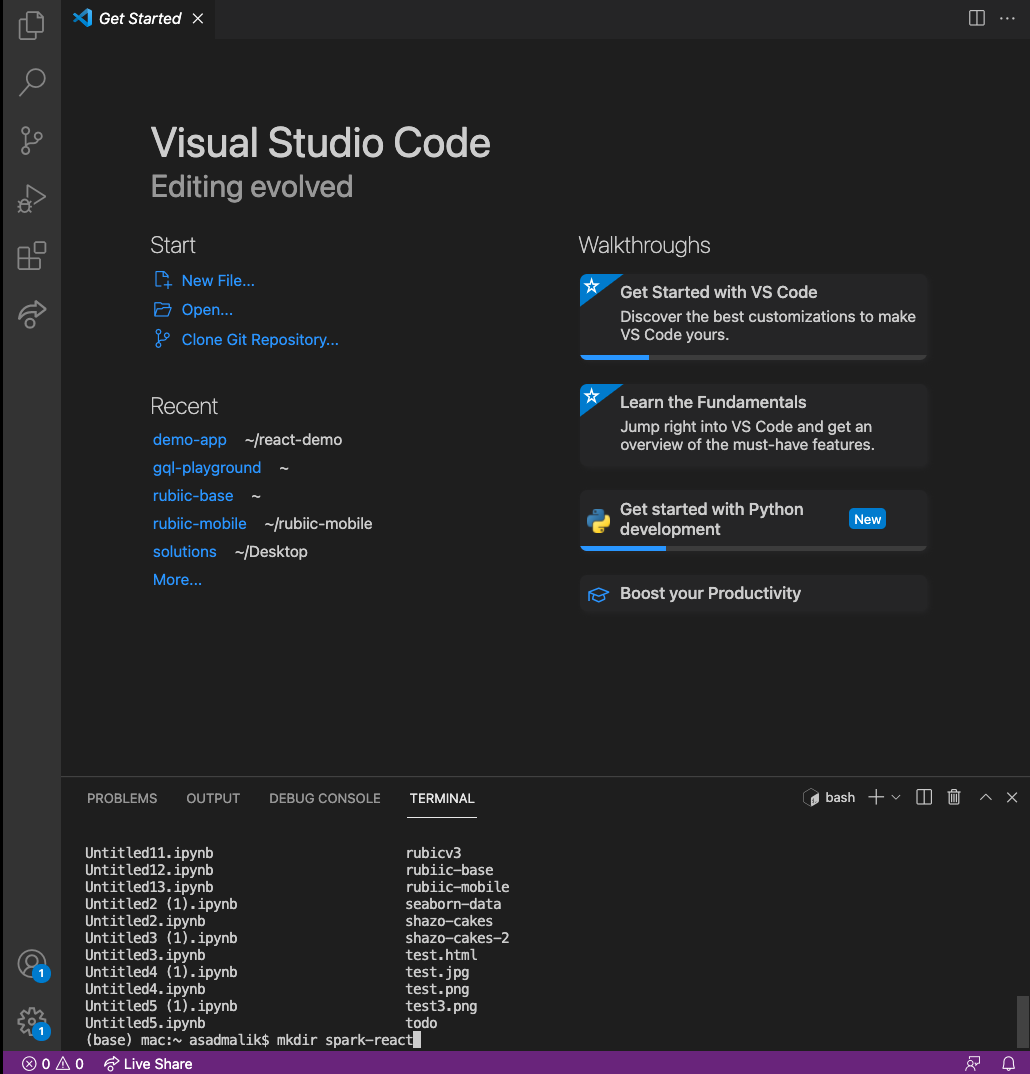
This workshop will assume that you have the above installed and we will being from there.

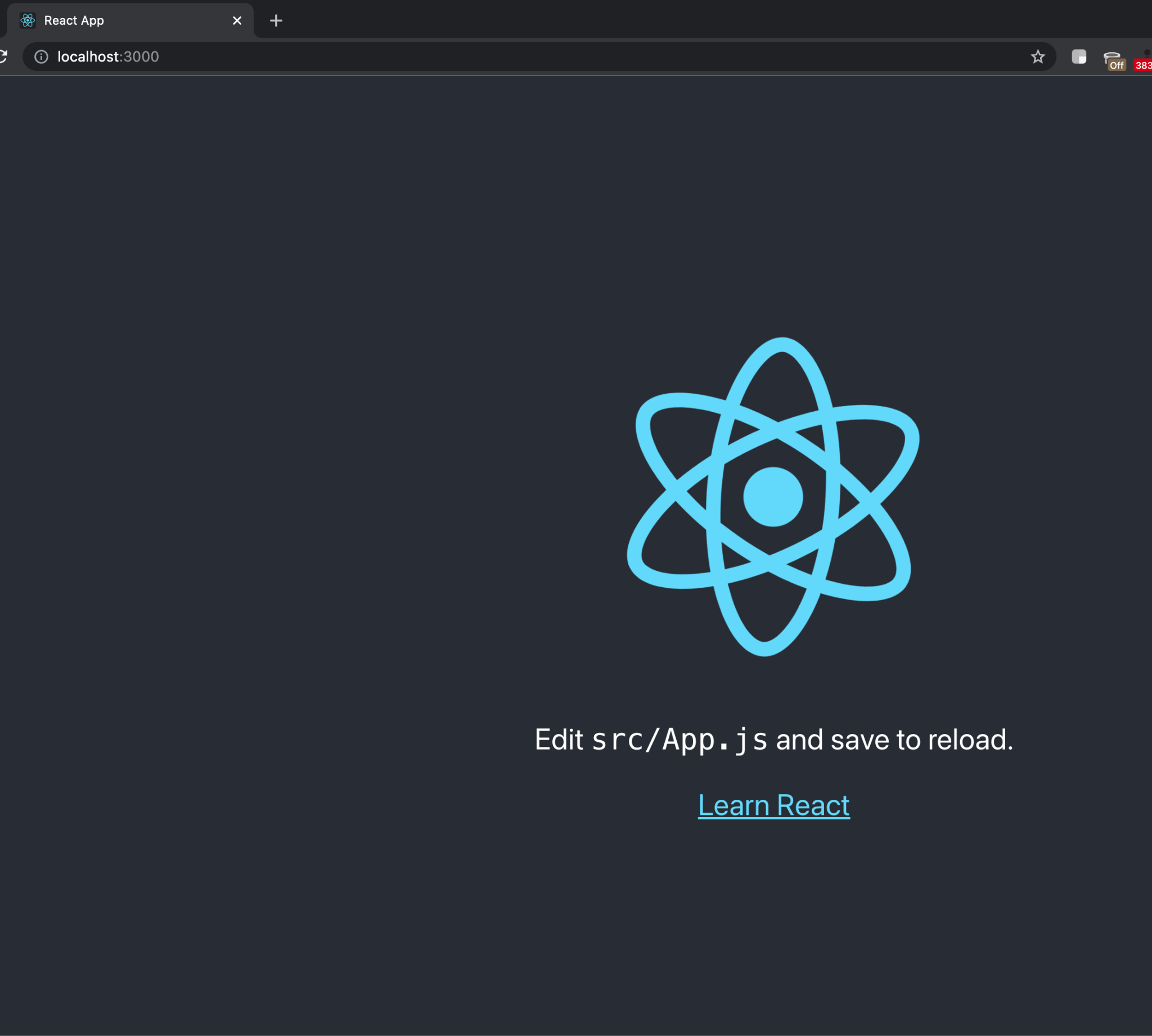
**Create React App**

We will get started by creating a folder for our project and using a tool called create-react-app. This is a npm package that when executed, creates all the boilerplate files and packages to get a React app going.

Let’s create a folder for our project.

* Open VS code
* Open a new terminal window inside VS code
* Using terminal commands, navigate to the base folder where you will be creating your folder. (I will be staying in the base folder)
* Create a new folder by using the command mkdir <Folder name> (below is what it should look like)



* In the example above, it will create a folder called spark-react
* After the folder is created, step into that folder. Cd <Folder Name>
* In my example, I will execute cd spark-react
* Using visual studio, open the folder that you just created
* That might close the terminal window you had open so open a new one if that happened
* The terminal window should open in the folder that you have just opened in vs code (the folder you just created for your project)
* Execute npx create-react-app <Project Name> to execute the library. This will do create all the boilerplate react files.
* I executed npx create-react-app spark-react.
* Navigate into the newly created project: cd <Project Name>
* Start your local dev environment by executing yarn start
* Your browser should automatically launch at localhost:3000 where your local server is running and you should see something like the following image, which is the default page that create-react-app creates
* 

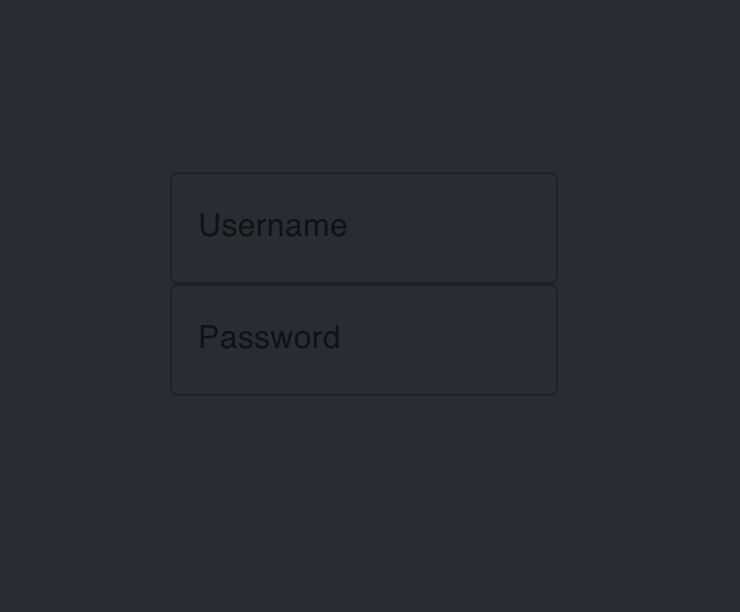
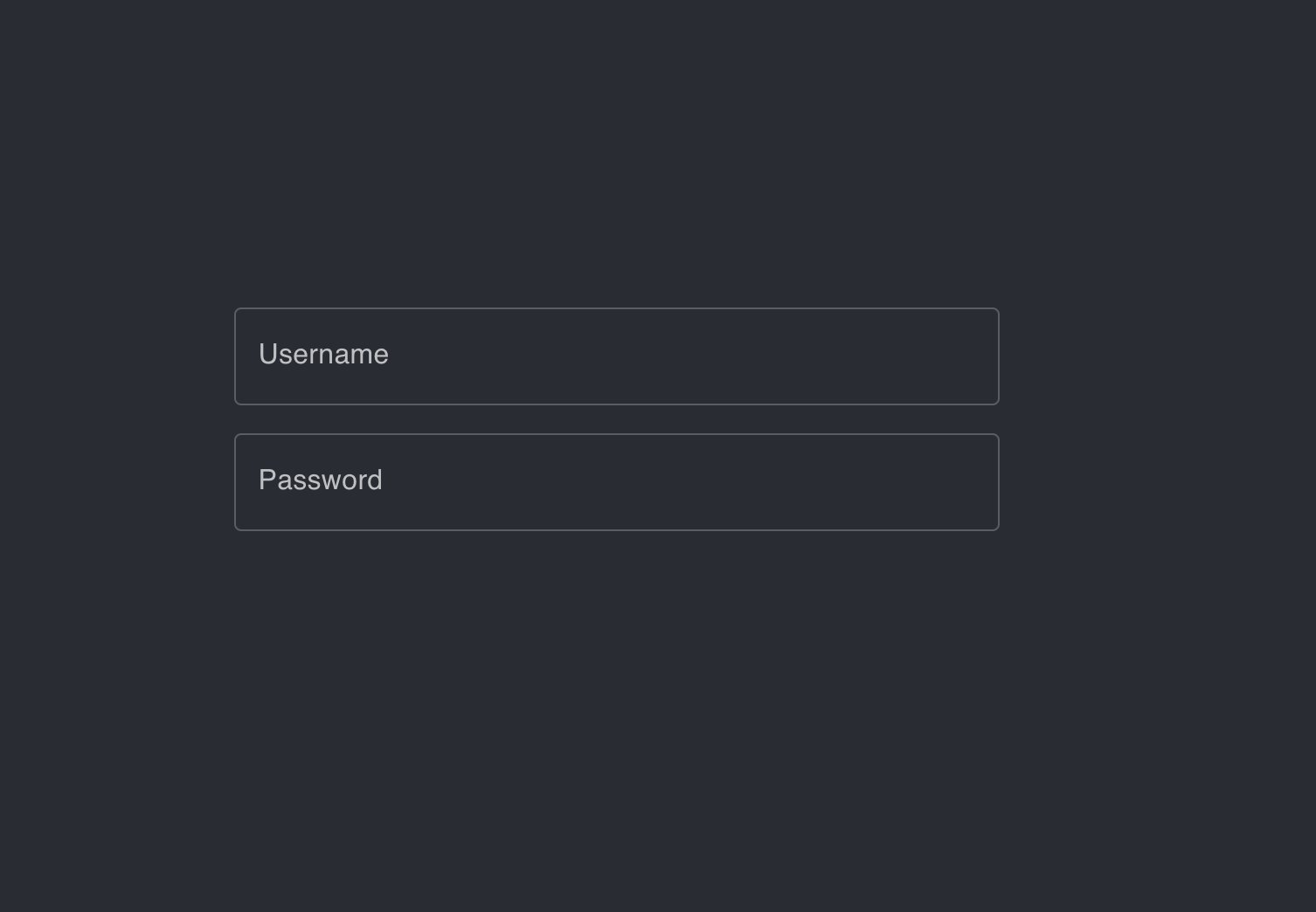
**Let’s start creating.**

To give you a sense of what our end goal is here, we are going to create a simple login form. Only the front end part, do a fake validation (This will make sense when we get to it) and once validated, navigate to the home page. We will cover state, routing, and material ui.

Let’s get started!

**Using Material UI and Saving data into state.**

An overview of this part is that we will be importing textboxes from the material ui library and placing that into our project. After that, the text that is entered into those textboxes, we will save in variables. In React, the saving data into variables is done using state.

* In VS code, from the sidebar, open src/App.js
* Let’s keep the background color and keeping everything centered, so in the file, lets keep the outer div and header tags, but lets get rid of the stuff in the middle.
* It should look like this:
* function App() {
* return (
* <div className="App">
* <header className="App-header">
* </header>
* </div>
* );
* }
* In the terminal window, install the material ui library plus some supporting libraries.
* npm install @mui/material @emotion/react @emotion/styled
* Let’s import the textbox components we want to use and plug them in.
* Open <https://mui.com/components/text-fields/>
* In the basic textfield section, for the examples that are give, open the source code.
* Import the textfield component to your file
* import TextField from '@mui/material/TextField';
* Import the html for the TextField component
* <TextField id="outlined-basic" label="Username" variant="outlined" />
* <TextField id="outlined-basic" label="Password" type="password" variant="outlined" />
* ****
* The above is what we should have at this point.
* Styling and scaling is a little bit out of the scope of this workshop, but we’ll add some basic stuff to make it look a little bit better. We will add two things, scale to the textfield component, and inject dark theme to the project.
* Let’s start by injecting a dark theme.
* Theming documentation is here <https://mui.com/customization/dark-mode/> but we will just use the createTheme object to create a basic dark mode theme and inject it into the application using the themeProvider object.
* Lets start with the import
* import { createTheme, ThemeProvider } from "@mui/material/styles";
* Let’s use createTheme to create a theme in the App function.
* const darkTheme = createTheme({
* palette: {
* mode: "dark",
* },
* });
* And then you can inject this theme using the ThemeProvider component
* <div className="App">
* <ThemeProvider theme={darkTheme}>
* <header className="App-header">
* <TextField id="outlined-basic" label="Username" variant="outlined" />
* <TextField id="outlined-basic" label="Password" type="password" variant="outlined" />
* </header>
* </ThemeProvider>
* </div>
* It should look like the above.
* Now let’s just add some scale into the TextField component, you can get a sense of styling and scaling functionalities by reading documentation on material ui’s website. For now, well just add sx={{ m: 1, width: "25ch" }}
* So the TextField line should look like
* <TextField sx={{ m: 1, width: "25ch" }} id="outlined-basic" label="Username" variant="outlined" />
* Here is what the code for the component should look like:
* ​​function App() {
* const darkTheme = createTheme({
* palette: {
* mode: "dark",
* },
* });
* return (
* <div className="App">
* <ThemeProvider theme={darkTheme}>
* <header className="App-header">
* <TextField sx={{ m: 1, width: "25ch" }} id="outlined-basic" label="Username" variant="outlined" />
* <TextField sx={{ m: 1, width: "25ch" }} id="outlined-basic" label="Password" type="password" variant="outlined" />
* </header>
* </ThemeProvider>
* </div>
* );
* }
* And here is what the page should look like in the browser:
* ****
* Thats a little bit better.
* Hopefully at this point, things are making sense and you’re getting a sense of how to be able to import components from a ui library and just putting style to make it look like what you want it to.
* Im going to add a header by using a h3 tag and put a button from MUI (material ui) so that we can execute a function once data has been entered.
* Now what we want to do is save the data entered into those textboxes into variables, so we will use state.
* Start by importing the useState hook into App.js
* import React, { useState} from "react";
* Now lets declare some state variables
* const [username, setUsername] = useState("");
* const [password, setPassword] = useState("");
* We have two elements here, the variable itself, plus a setter method that sets the value.
* Let’s link the username TextField to the variable we just created.
* <TextField sx={{ m: 1, width: "25ch" }} id="outlined-basic" label="Username" variant="outlined" onChange={ (e) => setUsername(e.target.value) }/>
* By adding the onChange, a listener is added to the text field and everytime the value is changed, it executes the function thats added in the block. I added the function straight into the block, but you could just as easily create a function in the component, and reference it in the onchange block.
* You can do the exact same thing for the password textfield.
* Next, we want to do some type of validation. We’ll do something for demonstrable purposes but this is something that you will most likely want to handle on the server side.
* We will keep track of if validation has been passed or not by declaring two boolean state variables, and a function that updates those variables based on the data that has been entered into the form.
* Here is what the block looks like
* const [validationPassed, setValidationPassed] = useState(false);
* const [validationFailed, setValidationFailed] = useState(false);
* function validation() {
* if (username === 'asadmalik' && password === 'buspark!'){
* setValidationPassed(true);
* setValidationFailed(false);
* return;
* }
* setValidationPassed(false);
* setValidationFailed(true);
* }
* And then we want to execute the validation function created above when the button that we imported from mui is clicked.
* <Button variant="outlined" onClick={validation}>Log In</Button>

Ok, hopefully, that was straightforward to follow along with. To recap, what we did there was import some ready made components from MUI library, hooked up state variables with those components so that once data is entered, it gets stored in those variables, and created a dummy validation mechanism, that updates the validation state so we know if validation has passed or not.

**Conditional Rendering**

In this mini step, we will use conditional rendering on the basis of the validation state variables to display a message confirming if validation has passed or failed.

In React, there are many things that you do conditionally. This can be applied broadly, but for now, we will simply render some text based on if validation in our project has passed. You will likely get used to this with practice as you start to use it in different ways, but here is what the code looks like to display conditional text based on the validation states that we defined earlier.

<div className="App">

<ThemeProvider theme={darkTheme}>

<header className="App-header">

<h3>Log In Form</h3>

<TextField sx={{ m: 1, width: "25ch" }} id="outlined-basic" label="Username" variant="outlined" onChange={ (e) => setUsername(e.target.value) }/>

<TextField sx={{ m: 1, width: "25ch", marginBottom: "40px" }} id="outlined-basic" label="Password" type="password" variant="outlined" onChange={ (e) => setPassword(e.target.value) }/>

<Button variant="outlined" onClick={validation}>Log In</Button>

{

validationPassed ? (

<div>

<h5>Validation Passed</h5>

</div>

) : validationFailed ? (

<div>

<h5>Validation Failed</h5>

</div>

) : (

<div>

<h5>Enter Data</h5>

</div>

)

}

</header>

</ThemeProvider>

</div>

Basically what is going on is when the validationPassed state is set to true, it will display the code in the brackets following it. If validationFailed is true, then the code block following it will display and if neither of those is true, then the last block will render. Basically think of it like an if else block.

**Components**

Component is a very fundamental React concept. In React, everything is a component. On a single page, there will be a collection of components that will come together to form a larger component that will form a page. So we will demonstrate that now with what we have done so far. Up until now, everything we have coded so far has been on the App.js file. In your code bases, you will have a number of files and folders, organized in a way specific to your projects. We will refactor our code here to replicate that.

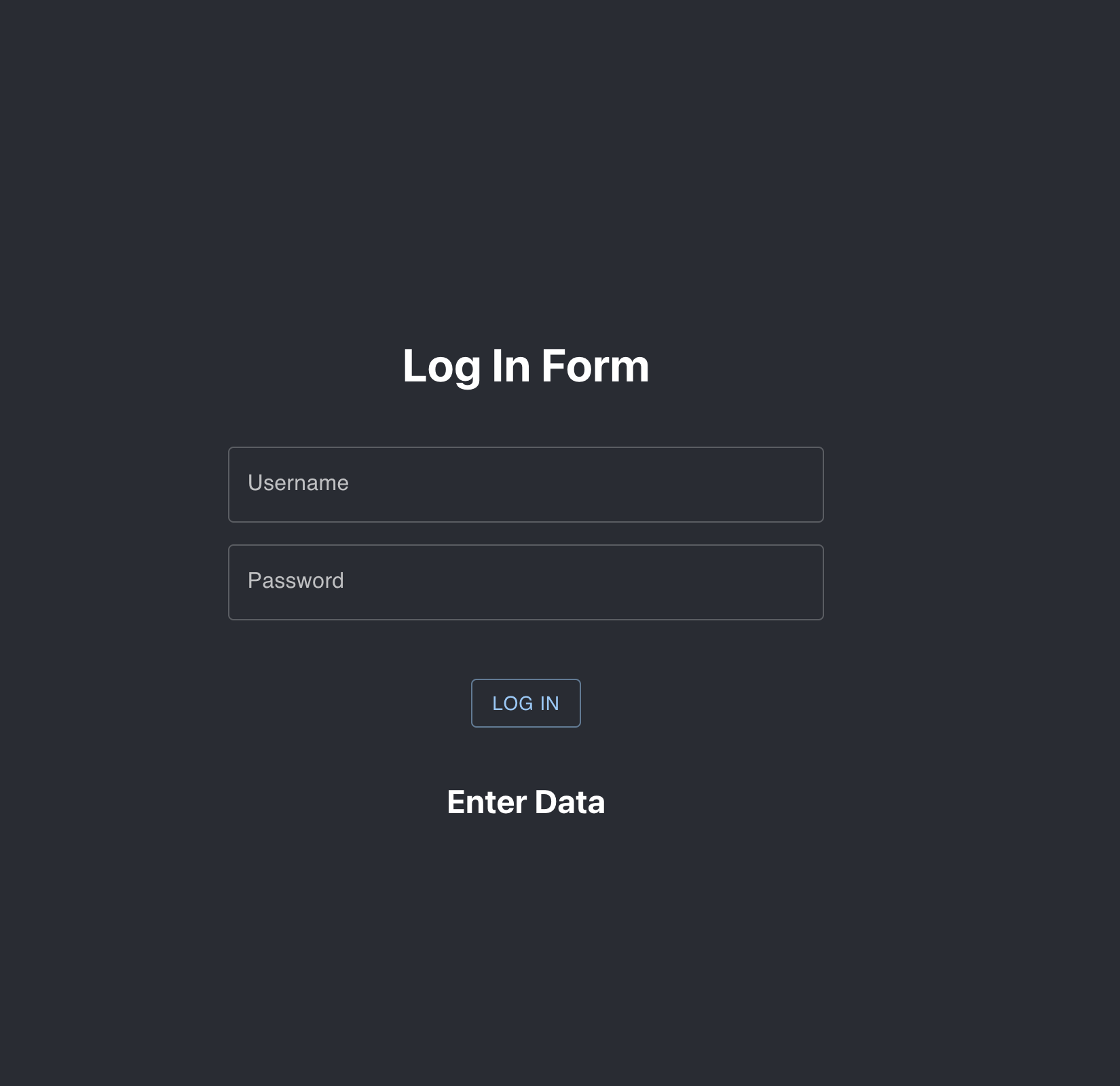
* First, install a VS code extension called React snippets. It is a tool that lets create React Components with simple shortcuts.
* In the src folder, create a new file called LoginComponent.js
* In the file, type in rfc and if your snippet has been installed right, it should give you dropdown options and select React functional component from that list.
* import React from 'react'
* export default function LoginComponent() {
* return (
* <div>
* </div>
* )
* }
* The above is what you should have
* Now, lets copy over the login code we generated into this file.

There are some things to consider before we copy over code from App.js into this new file that we have created. We can copy over the html pretty much exactly as we have it in App.js. The variables and functions that we have here i.e username, password and the validation functions that we have, we have to consider if we want to place them directly on to the component, or if we want to pass it from a parent component. The rule of thumb is that if some data needs to be used across components, pass it as a prop to child components, if it’s only to be used on the component, create it there.

So let’s start copying over code into the new file.

* First let’s start with import.
* import React, {useState} from 'react'
* import TextField from '@mui/material/TextField';
* import Button from '@mui/material/Button';
* And then we are going to copy over the username and password state as well as the validation function. We are not going to copy over the validationPassed and validationFailed states because they are needed at the higher level, it will make sense a little later.
* const [username, setUsername] = useState("");
* const [password, setPassword] = useState("");
* function validation() {
* if (username === 'asadmalik' && password === 'buspark!'){
* props.data.setValidationPassed(true);
* props.data.setValidationFailed(false);
* console.log('validated');
* return;
* }
* console.log('validation failed');
* props.data.setValidationPassed(false);
* props.data.setValidationFailed(true);
* }
* And finally, we are going to copy over the entire html segment
* <div>
* <h3>Log In Form</h3>
* <TextField sx={{ m: 1, width: "25ch" }} id="outlined-basic" label="Username" variant="outlined" onChange={ (e) => setUsername(e.target.value) }/>
* <TextField sx={{ m: 1, width: "25ch", marginBottom: "40px" }} id="outlined-basic" label="Password" type="password" variant="outlined" onChange={ (e) => setPassword(e.target.value) }/>
* <Button variant="outlined" onClick={validation}>Log In</Button>
* {
* props.data.validationPassed ? (
* <div>
* <h5>Validation Passed</h5>
* </div>
* ) : props.data.validationFailed ? (
* <div>
* <h5>Validation Failed</h5>
* </div>
* ) : (
* <div>
* <h5>Enter Data</h5>
* </div>
* )
* }
* </div>
* We can save this and import this into our App.js main file.
* In App.js, import like this:
* import LoginComponent from "./LoginComponent";
* And then you can reference this in the App.js file in html as:
* <LoginComponent />
* But we still need to pass the validation data. So here is what that will look like
* <LoginComponent data={{setValidationFailed, setValidationPassed, validationPassed, validationFailed}} />
* When we reference the component, we simply pass data into the component and the value of the data is an object that contains all the data that we want to pass.
* This data that is passed can be referenced within the component in the props.data object.
* If you look above, in the code that we copied into the new component file, references to validation are now done through the props.data object as instead of being directly referenced, they are being passed down to the component as props.
* Now in App.js file, we can delete all the code that we copied over, but make sure to not delete the values that we are passing down as props.
* Now the html part of App.js simply looks like this
* <div className="App">
* <ThemeProvider theme={darkTheme}>
* <header className="App-header">
* <LoginComponent data={{setValidationFailed, setValidationPassed, validationPassed, validationFailed}} />
* </header>
* </ThemeProvider>
* </div>

Just a quick recap, what we have done here is created a component, we put our login textfields into a component, then imported that component into our main App.js file and passed some data into that component through props. Here is what our page should look like.



**Routing**

The last thing we are going to cover as part of this workshop is routing.

Lets get started by installing react-router

* npm install --save react-router in terminal
* Npm install react-router-dom
* Let’s create a component that we can route to from the component that we have created.
* Create a new file in the src folder called HomeComponent.js
* Enter rfc and hit the React Functional Component to create the boiler plate code.
* We wont be modifying this component too much, lets add a header so we know that we have successfully routed to this component when its added in our routing later.
* Lets go back to App.js and import the newly created HomeComponent.js in there.
* import HomeComponent from "./HomeComponent";
* Lets import some functionality from react-router into App.js as well.
* import { Route, Switch } from "react-router";
* import { BrowserRouter as Router} from "react-router-dom";
* And then lets use it in App.js, here is what that looks like.
* And then we need to wrap components within routes. Here is what that looks like.
* <Router>
* <div className="App">
* <ThemeProvider theme={darkTheme}>
* <header className="App-header">
* <Switch>
* <Route path="/home">
* <HomeComponent />
* </Route>
* <Route path="/">
* <LoginComponent data={{setValidationFailed, setValidationPassed, validationPassed, validationFailed}} />
* </Route>
* </Switch>
* </header>
* </ThemeProvider>
* </div>
* </Router>
* You can test this by going into the browser where your local server is running and add /home to the localhouse route and it should show the home component.
* Lastly, lets go back to the LoginComponent, and route to the home component programmatically, if the validation passes.
* In LoginComponent, first import useHistory hook:
* import { useHistory } from "react-router-dom";
* Declare a variable that uses that:
* const history = useHistory();
* And then in the validation function, in the block where validation passes, add history.push('/home');
* The full function should look like this
* function validation() {
* if (username === 'asadmalik' && password === 'buspark!'){
* props.data.setValidationPassed(true);
* props.data.setValidationFailed(false);
* console.log('validated');
* history.push('/home');
* return;
* }
* console.log('validation failed');
* props.data.setValidationPassed(false);
* props.data.setValidationFailed(true);
* }