# Preparing the project and creating first components

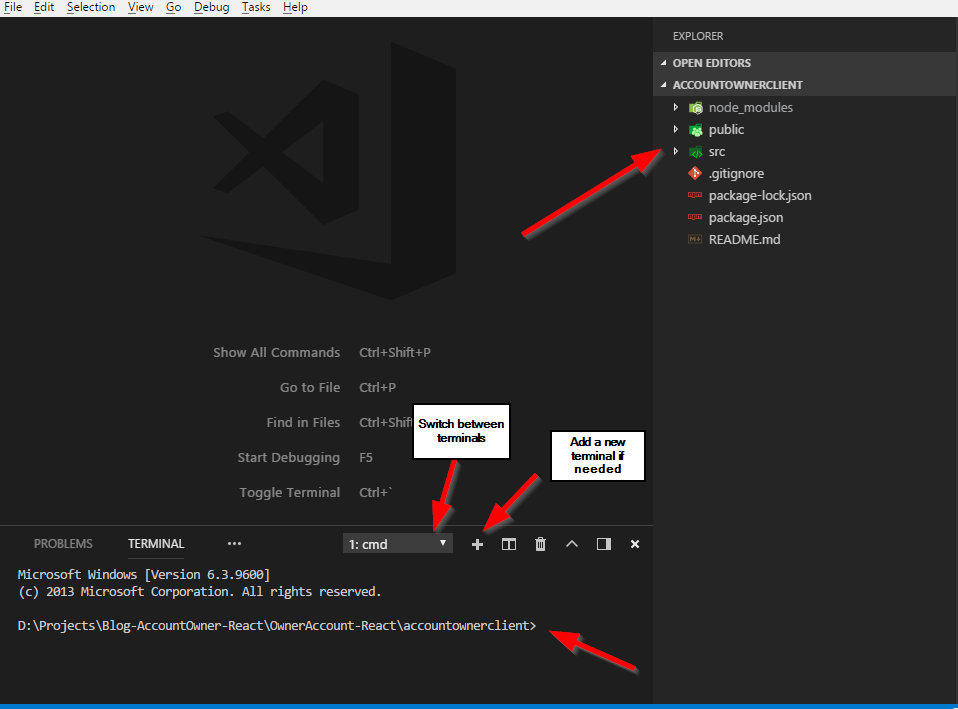
Creating the server part (.NET Core web API part) is just a half of the job we want to accomplish. From this point onwards, we are going to dive into the client side of the application to consume the web API part and show the results to the user by using React components and many other features.

## Creating a New Project

To create a new project, we need to execute Create React App command for creating a new application. Open Visual Studio Code, in a terminal window (CTRL+`), navigate to the folder you want your project in and type the command:

npx create-react-app accountownerclient

After some time a new project is going to be created:



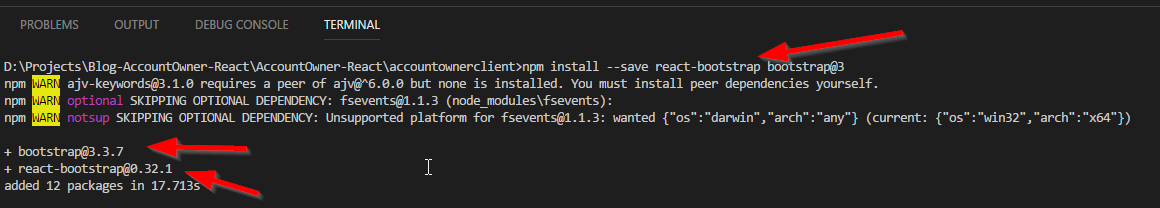
If you want to learn in more detail about the Create React App, visit this site: <https://github.com/facebook/create-react-app>

## Third-Party Libraries

We are going to use the React-Bootstrap library for styling our components so let’s install it and import it into the project.

Type this command to install the React-Bootstrap and the Bootstrap version 3 libraries:

npm install --save react-bootstrap bootstrap@3



After the installation, import the Bootstrap library inside the index.js file:

import 'bootstrap/dist/css/bootstrap.css';

import 'bootstrap/dist/css/bootstrap-theme.css';

Now we can use bootstrap in our project. We are going to install more third-party libraries inside our project, but we are going to deal with the installations once we need those libraries.

## React Components Overview

The React is a framework for creating the SPA’s (Single Page Application) applications. Therefore, we are going to create all of our pages on one page. That page is the index.html. If you look at that page, you are going to notice this line of code:

<div id="root"></div>

This is the place where all of our pages are going to be generated.

But how does React know to render all the pages inside that tag?

Well, if we look in the index.js file, we are going to notice this line of code:

ReactDOM.render(<App />, document.getElementById('root'));

What this means is that React will render all the components from the App component to the index.html page inside the div with id=root. This also means that App component is going to be the main component of our entire application.

So let’s talk a bit about components in React. There are two types of components:

* Stateful (class) components and
* Stateless (functional) components

The App component is a stateful component because it has access to the state. Stateful components also have the lifecycle events and access to the props with the this.props expression. If you look in the App.js file, you are going to notice the render() function which is one of the lifecycle functions. Stateful components must have at least render() function from all the lifecycle functions inside the React. They must extend from the Component class.

Stateless components are much simpler than the stateful components. They don’t have a lifecycle and should be used whenever you don’t need state inside your component. They also have access to the props, which React provides to them as a parameter.

Whether you using the stateful or the stateless components you must export that component to use it in any other components.

## Creating Our Components

Before we start with the creation, let’s modify the App.js and App.css files. Remove all the code from the App.css file and modify the App.js file:

import React, { Component } from 'react';

import logo from './logo.svg';

import './App.css';

class App extends Component {

render() {

return (

);

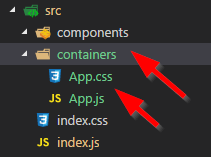
}

}

export default App;

You are going to have an error now because return() functions demand one root tag, but we are going to fix this soon.

Before we continue, let’s create the base folder structure for the containers (stateful components) and components (functional components). This action is not a must but it is a good practice to separate your class and functional components. We are going to put the App.js file inside containers folder and to modify index.js file because it imports the App.js.

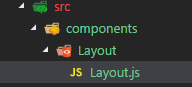


Now just modify import statement inside the index.js file and remove import for the logo.svg:

import App from './containers/App';

Excellent.

Let’s create our first component. In the components folder, create a new folder and name it Layout. Then inside this folder, create a Layout.js file and modify it:



import React from 'react';

import { Grid, Row } from 'react-bootstrap';

const layout = (props) => {

return (

<Grid>

<Row>

This is the place for the navigation component.

</Row>

<main>

{props.children}

</main>

</Grid>

)

}

export default layout;

Let’s take some time to review this code. First of all, this is a stateless component (functional) and you may notice that because we don’t have class in here but just function (arrow function to be more precise). Because this is not a class component we don’t need to extend Component, therefore we are not importing it at all. The Grid and a Row are React-Bootstrap components. Those are equivalents to the <div class=”row”> and <div class=”container”> elements if you are familiar with the Bootstrap library.

The functional component is fetching the props object through the props argument and all the properties from the props object are going to be available inside the functional component. One of those properties is the “children” property, which is going to show all the data between opening and closing Layout tag (<Layout> everything in here is children property of props object </Layout>). We are going to see this in action in next example. Finally we are exporting this component. This type of export is called default export.

Let’s continue by modifying the App.js file:

import React, { Component } from 'react';

import './App.css';

import Layout from '../components/Layout/Layout';

class App extends Component {

render() {

return (

<Layout>

<strong>This content is going to be rendered as the props.children inside Layout component.</strong>

</Layout>

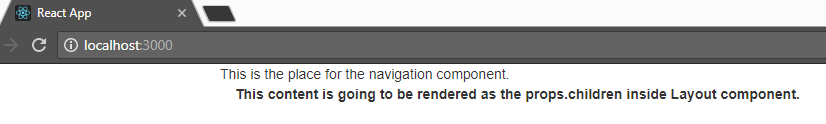
);

}

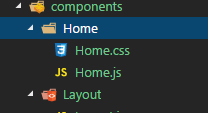
}

export default App;

Execute the npm start command in your terminal and you should be able to see your page on the localhost:3000 (not so pretty page but it is a start :D )



We are going to continue with the Home component. So, firstly create a folder structure for this component:



Modify the Home.js component:

import React from 'react';

import { Col, Row } from 'react-bootstrap';

import './Home.css';

const home = (props) => {

return (

<Row>

<Col md={12}>

<div className={'homeText'}>

"WELCOME TO ACCOUNT-OWNER APPLICATION"

</div>

</Col>

</Row>

)

}

export default home;

Then modify the Home.css file:

.homeText{

font-size: 35px;

color: red;

text-align: center;

position: relative;

top:30px;

text-shadow: 2px 2px 2px gray;

}

Finally, modify the App.js file:

import React, { Component } from 'react';

import './App.css';

import Layout from '../components/Layout/Layout';

import Home from '../components/Home/Home';

class App extends Component {

render() {

return (

<Layout>

<Home />

</Layout>

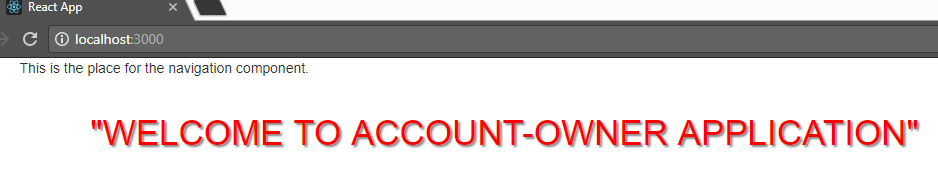
);

}

}

export default App;

When you save all of your files, the page on localhost:3000 should look like this:



# Navigation and Routing

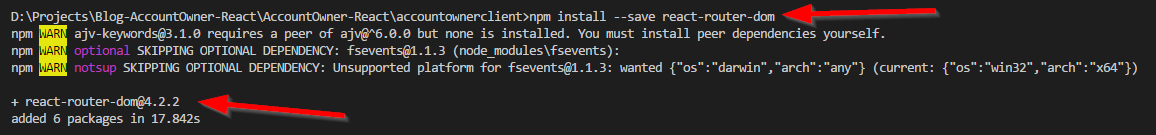
Even though we are generating all of our pages inside one page, we still need a navigation to load these different pages inside our index.html. React Router enables this navigation functionality for our project.

We are going to create three menu options: Home, Owner Actions, and Account Actions. This would be enough for us to see how navigation works in the React and how to set up lazy load functionality for asynchrony component loading inside the project.

## React Router Installation, BrowserRouter, Switch, and Route

To install React Router executed this command in the terminal window:

**npm install --save react-router-dom**



After installation let’s modify the App.js file:

import React, { Component } from 'react';

import './App.css';

import Layout from '../components/Layout/Layout';

import Home from '../components/Home/Home';

import { BrowserRouter, Switch, Route } from 'react-router-dom';

class App extends Component {

render() {

return (

<BrowserRouter>

<Switch>

<Layout>

<Route path="/" exact component={Home} />

</Layout>

</Switch>

</BrowserRouter>

);

}

}

export default App;

For the routing to work, the central component of our application must be inside BrowserRouter component. Our central component is the Layout component and we are wrapping it with the BroswerRouter. You could do this in another way inside the index.js file by wrapping the <App/> component inside the ReactDOM.render() function. Either way you choose it gives the same result.

Switch component renders the first child that matches the location. Once it finds the matching rout it will stop searching for another route.

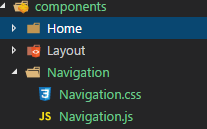
Finally, the Route component is the one that is going to redirect us from one component to another. For now, we only have the Home component and it is going to be served on the “/” path.

If you navigate to localhost:3000 you are going to see the same result as before but now we are using routing to provide our components.

## Navigation menu

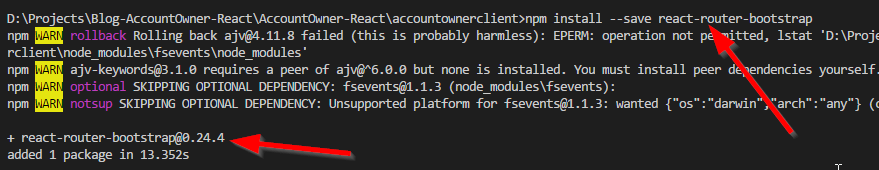
Now when we have the routing in place let’s create a navigation menu.

Inside the components folder, create a new folder and name it Navigation. Inside create two files Navigation.js and Navigation.css:



We are going to use Bootstrap navigation element for creating our own navigation. Prior to creation let’s install another library which React needs to merge Bootstrap navigation with React-Router navigation:

npm install –save react-router-bootstrap



Now, modify the Navigation.js file:

import React from 'react';

import './Navigation.css';

import { Col, Navbar, Nav, NavItem } from 'react-bootstrap';

import { NavLink } from 'react-router-dom';

import { LinkContainer } from 'react-router-bootstrap';

const navigation = (props) => {

return (

<Col md={12} >

<Navbar inverse collapseOnSelect>

<Navbar.Header>

<Navbar.Brand>

<NavLink to={'/'} exact >Account-Owner</NavLink>

</Navbar.Brand>

<Navbar.Toggle />

</Navbar.Header>

<Navbar.Collapse>

<Nav>

<LinkContainer to={'/owner-list'} exact>

<NavItem eventKey={1}>

Owner Actions

</NavItem>

</LinkContainer>

<LinkContainer to={'/account-list'}>

<NavItem eventKey={2}>

Account Actions

</NavItem>

</LinkContainer>

</Nav>

</Navbar.Collapse>

</Navbar>

</Col>

)

}

export default navigation;

Let’s explain this code. As you might have noticed from the import statement Navbar, Nav and NavItem are the Bootstrap components and they serve us to create visual part of our navigation component. But for the routing to work we need to use NavLink, which is the react-router-doms component. With NavLink component we can navigate to exact path and also it implements styling the active link inside the navigation.

Now pay attention to the NavItem component. This is Bootstraps component to create a single navigation item. And as you already know we need to use the NavLink and not the NavItem for the routing to work. Replacing the NavItem with the NavLink won't do the trick because visual part of navigation would be scrambled. So the solution is using LinkContainer component from the react-router-bootstrap library, which is already installed. This component simulates the NavLink component completely, therefore enabling us to use this navigation without any problems.

All we have to do is to modify Navigation.css file and to include this Navigation component into the Layout component:

div a.active, ul li.active{

font-weight: bold!important;

font-style: italic!important;

color: #fff!important;

}

import React from 'react';

import { Grid, Row } from 'react-bootstrap';

import Navigation from '../Navigation/Navigation';

const layout = (props) => {

return (

<Grid>

<Row>

<Navigation/>

</Row>

<main>

{props.children}

</main>

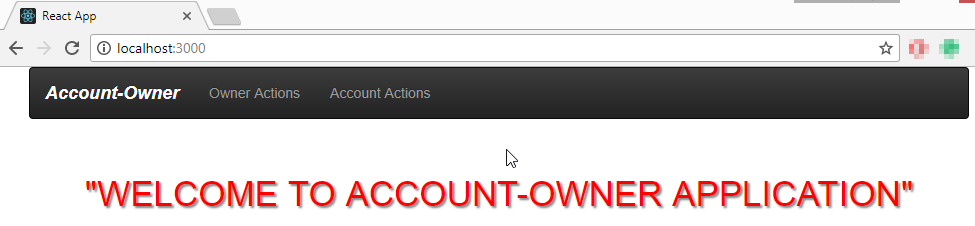
</Grid>

)

}

export default layout;

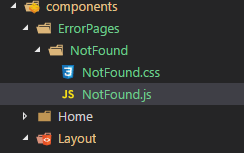
And you can check the result:



## Creating Not-Foud Component

We are going to add one more feature to complete this part of the post. Whenever a user types a nonexisting url address in the URL bar we are going to redirect him to the not-found (404) component.

Firstly let’s create a new folder inside the components folder and name it ErrorPages. Inside create a new folder with a name NotFound. There, create two new files NotFound.js and NotFound.css:



Modify the NotFound.js file:

import React from 'react';

import './NotFound.css'

const notFound = (props) => {

return (

<p className={'notFound'}>

"404 SORRY COULDN'T FIND IT!!!"

</p>

)

}

export default notFound;

We are just creating a functional component which returns some JSX code inside return block. JSX is syntax extension to JavaScript. It is a recommendation to use it with the React to describe what the UI should look like. Even though it could remind you of a template language, it comes with the full power of JavaScript.

Then modify the NotFound.css file:

.notFound{

font-weight: bold;

font-size: 50px;

text-align: center;

color: #f10b0b;

}

Finally, modify the App.js file:

import NotFound from '../components/ErrorPages/NotFound/NotFound';

<BrowserRouter>

<Layout>

<Switch>

<Route path="/" exact component={Home} />

<Route pat="\*" component={NotFound} />

</Switch>

</Layout>

</BrowserRouter>

With this modification in place whenever the user types an unexisting url address, the NotFound component is going to be rendered (even if you type localhost:3000/404 :D )

