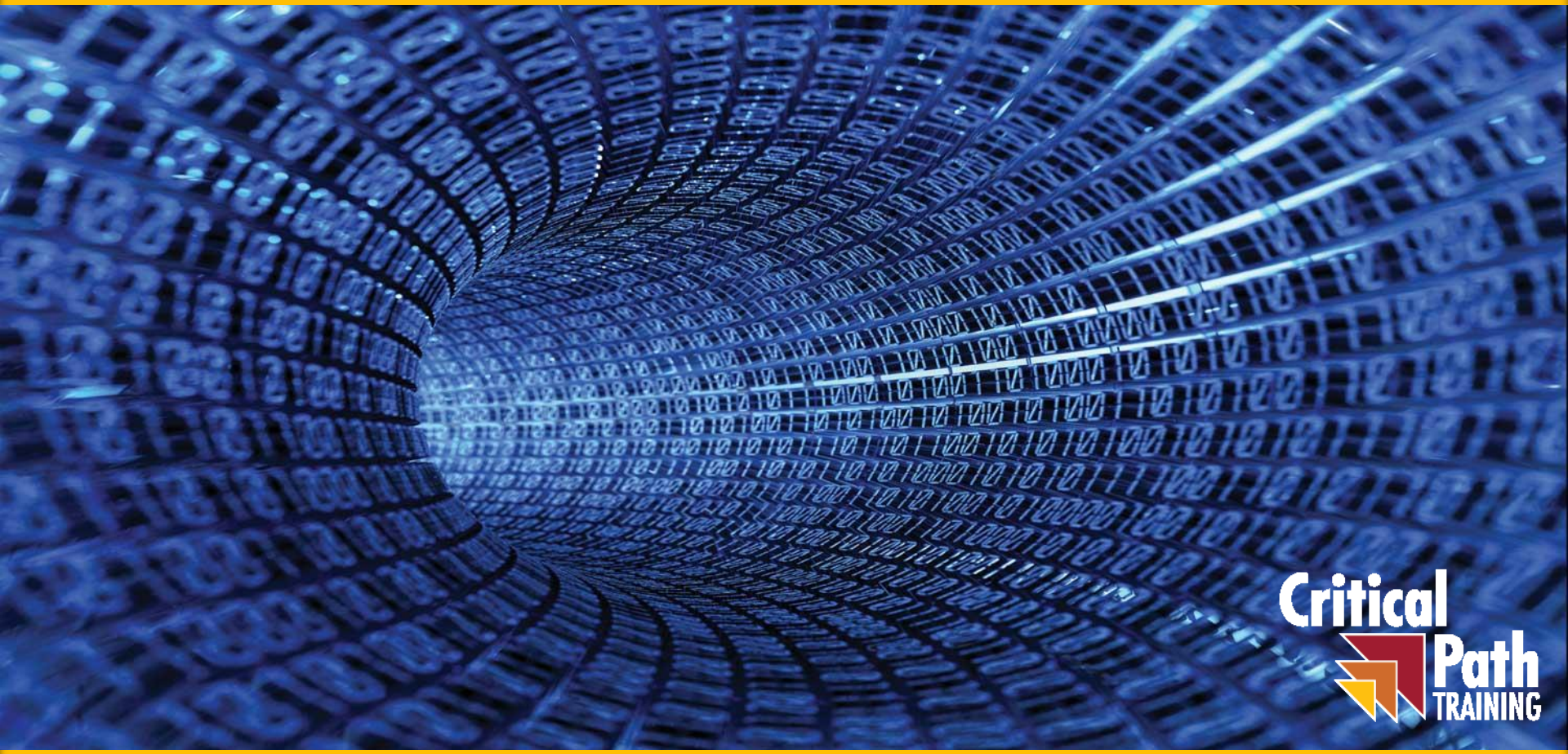
Developing SPAs with React and JSX/TSX



Agenda

- Getting Started with React.js
- Creating SPAs using React.js, TypeScript and Webpack
- Designing a React Component Hierarchy
- Extending a React Project with the React Router
- Understanding the React Component Lifecycle Methods
- Calling Across the Network using the Fetch API

Introducing React

- React is a library for building user interfaces
  - Not as all-encompassing as a framework like Angular
  - Focused on building HTML-based user experiences
  - Based on reusable component-based architecture
  - Components *react* to state changes by updating UI
  - React uses shadow DOM for efficient event handling
- React was originally designed for Facebook
  - Also a good fit for building SPFx web parts

React versus ReactDOM

- **React** and **ReactDOM** are separate libraries
  - **React (react.js)** is the primary library used to build out user experiences
  - **ReactDOM** (**react-dom.js**) is used to render **React** user experience in the browser
- **React** library exposes global **React** object
  - **React** object is the main entry point into React API
  - **React.DOM** wraps standard HTML elements
- **ReactDOM** library exposes global **ReactDOM** object
  - **ReactDOM** object used to render React components on web page

```
var reactComponenent = React.DOM.h1(null, "Hello, React!");

var target = document.getElementById("app");

ReactDOM.render(reactComponenent, target);
```

React Component Created Using ES5

- React component can be created using EcmaScript 5
  - React component definition created using **React.createClass**
  - React component must be defined with **render** method
  - React component can be instantiated with **React.createElement**

```
var myComponent = React.createClass({
  render: () => {
    return React.DOM.h1(null, "Hello React!")
  }
});

ReactDOM.render(
  React.createElement(myComponent),
  document.getElementById("app")
);
```

Initializing Element Properties

- Elements created using properties object
  - Object properties used to initialize element properties
  - Use **className** instead of **class** to assign CSS class
  - Use **htmlFor** instead of **for** to define HTML label

```
render: () => {

  var elementProperties = {
    id: "myElementId",
    className: "myCssClass"
  };

  return React.DOM.h1( elementProperties , "Hello React!");

}
```

Initializing Element Styles

- Elements styles initialized using style object
  - style must be defined using an object not a string
  - CSS properties referenced using camel casing

```
render: () => {

  var elementProperties = {
    id: "myElementId",
    style: {
      backgroundColor: "yellow",
      borderStyle: "Solid",
      borderColor: "green",
      padding: 8,
      color: "Blue",
      fontSize: 48
    }
  };

  return React.DOM.h1( elementProperties , "Hello React!");

}
```

React Provides Synthetic Events

- Replaces standard DOM-based event handling
  - React creates virtual DOM for elements in component
  - React interacts with real DOM when required
  - Provides faster event registration and processing
  - No need to write browser-specific code

```
private incrementCounter() {
  var previousCount: number = this.state.count;
  this.setState({ count: previousCount + 1 });
}

public render(): React.ReactElement<IBeanCounterProps> {
  return (
    <div className={styles.beanCounter}>
      <h3>Mr Bean Counter</h3>
      <div className={styles.toolbar}>
        <button onClick={(event) => { this.incrementCounter(); }}  >Add another Bean</button>
      </div>
      <div className={styles.beanCounterDisplay} >
        Bean Count: {this.state.count}
      </div>
    </div>
  );
}
```

Understanding JSX (and TSX)

- JSX provides better syntax for HTML composition
  - JSX allows extends JavaScript with XML-like syntax
  - JSX syntax must be transpiled into JavaScript code

```
var myHtml = <div id="myAppContainer" style={{ backgroundColor:"yellow", padding:8 }}>
               <h2>Hello JSX</h2>
               <p>I'm composing HTML elements using JSX syntax.</p>
             </div>;


ReactDOM.render( myHtml , document.getElementById("app") );
```

- JSX/TSX is separate from React library
  - JSX/TSX commonly used in React development
  - Babel compiler used to transpile JSX to JavaScript
  - TypeScript compiler used to transpile TSX to JavaScript

Agenda

- ✓ Getting Started with React.js
- ➢ Creating SPAs using React.js, TypeScript and Webpack
- Designing a React Component Hierarchy
- Extending a React Project with the React Router
- Understanding the React Component Lifecycle Methods
- Calling Across the Network using the Fetch API

Defining React Components using TypeScript

- Component is class extending `React.Component`
  - Component usually defined in its own **tsx** file
  - Component class must define **render** method

my-component.tsx

```tsx
import * as React from 'react';

export class MyComponent extends React.Component<any, any> {
    render() {
        return <h2>Hello from my component</h2>;
    }
}
```

  - Component can be instantiated with JSX/TSX syntax

app.tsx

```tsx
import * as ReactDOM from 'react-dom';

import { MyComponent } from "./components/my-component"

window.onload = () => {
  // Create and render component
  ReactDOM.render( <MyComponent/>, document.getElementById("app") );
}
```

Component Properties and State

- Component can contain properties and state
  - Properties are initialized by external components
  - Properties are read-only to hosting component
  - State is set internally by hosting component
  - Changing state triggers UI refresh by calling render
  - UI experience created by ***reacting*** to changes in state

React Component Properties

- Defining component with a property

component1.tsx

```tsx
import * as React from 'react';

export interface MyCustomProps {
  Name: string;
}

export class Component1 extends React.Component<MyCustomProps, {}> {
  render() {
    return <div>Hello, my name is {this.props.Name}</div>;
  }
}
```

- Instantiating component with a property

```tsx
ReactDOM.render(
    <Component1 Name="Fred" />,
    document.getElementById("app")
);
```

Stateful Component

IBeanCounterProps.ts

```ts
export interface IBeanCounterProps {
  StartingValue: number;
}
```

IBeanCounterState.ts

```ts
export interface IBeanCounterState {
    count: number;
}
```

BeanCounter.tsx

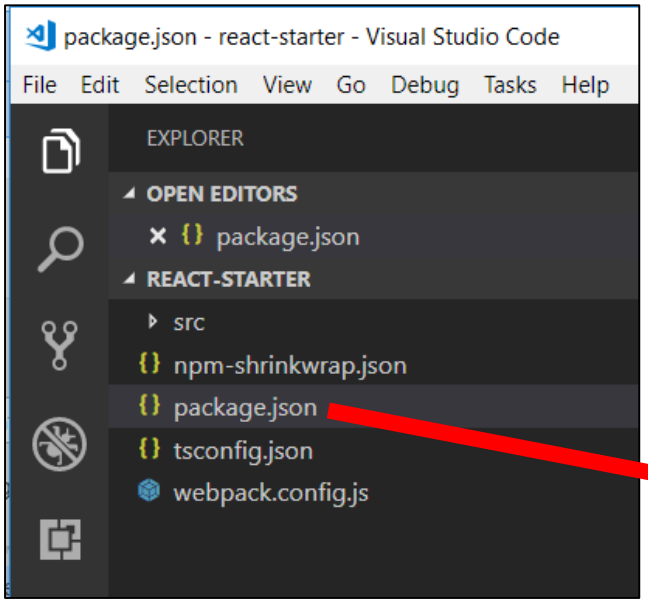
```tsx
import * as React from 'react';
import styles from './BeanCounter.module.scss';
import { IBeanCounterProps } from './IBeanCounterProps';
import { IBeanCounterState } from './IBeanCounterState';

export default class BeanCounter extends React.Component<IBeanCounterProps, IBeanCounterState> {

  constructor(props: any) {
    super(props);
    this.state = { count: this.props.StartingValue };
  }

  private incrementCounter() {
    var previousCount: number = this.state.count;
    this.setState({ count: previousCount + 1 });
  }
```

Stateful Component Rendering

BeanCounter.tsx

```tsx
import * as React from 'react';
import styles from './BeanCounter.module.scss';
import { IBeanCounterProps } from './IBeanCounterProps';
import { IBeanCounterState } from './IBeanCounterState';

export default class BeanCounter extends React.Component<IBeanCounterProps, IBeanCounterState> {

  constructor(props: any) {
    super(props);
    this.state = { count: this.props.StartingValue };
  }

  private incrementCounter() {
    var previousCount: number = this.state.count;
    this.setState({ count: previousCount + 1 });
  }

  public render(): React.ReactElement<IBeanCounterProps> {
    return (
      <div className={styles.beanCounter}>
        <h3>Mr Bean Counter</h3>
        <div className={styles.toolbar}>
          <button onClick={(event) => { this.incrementCounter(); }}  >Add another Bean</button>
        </div>
        <div className={styles.beanCounterDisplay} >
          Bean Count: {this.state.count}
        </div>
      </div>
    );
  }
}
```

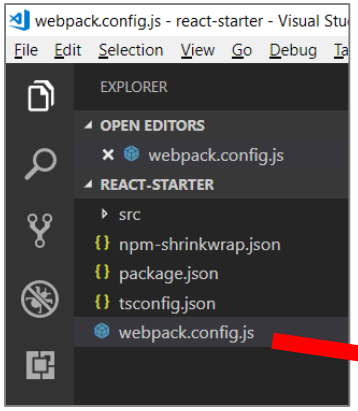
Starter Project - package.json



package.json

```
{
  "name": "react-starter",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "build": "webpack",
    "start": "webpack-dev-server --open --history-api-fallback"
  },
  "devDependencies": {
    "@types/react": "^16.4.13",
    "@types/react-dom": "^16.0.7",
    "awesome-typescript-loader": "^5.2.0",
    "bootstrap": "^4.1.3",
    "clean-webpack-plugin": "^0.1.19",
    "copy-webpack-plugin": "^4.5.2",
    "css-loader": "^0.28.11",
    "expose-loader": "^0.7.5",
    "file-loader": "^1.1.11",
    "html-webpack-plugin": "^3.2.0",
    "jquery": "^3.3.1",
    "popper.js": "^1.14.4",
    "react": "^16.4.2",
    "react-dom": "^16.4.2",
    "style-loader": "^0.21.0",
    "typescript": "^3.0.1",
    "url-loader": "^1.0.1",
    "webpack": "^4.16.4",
    "webpack-cli": "^3.1.0",
    "webpack-dev-server": "^3.1.5"
  }
}
```

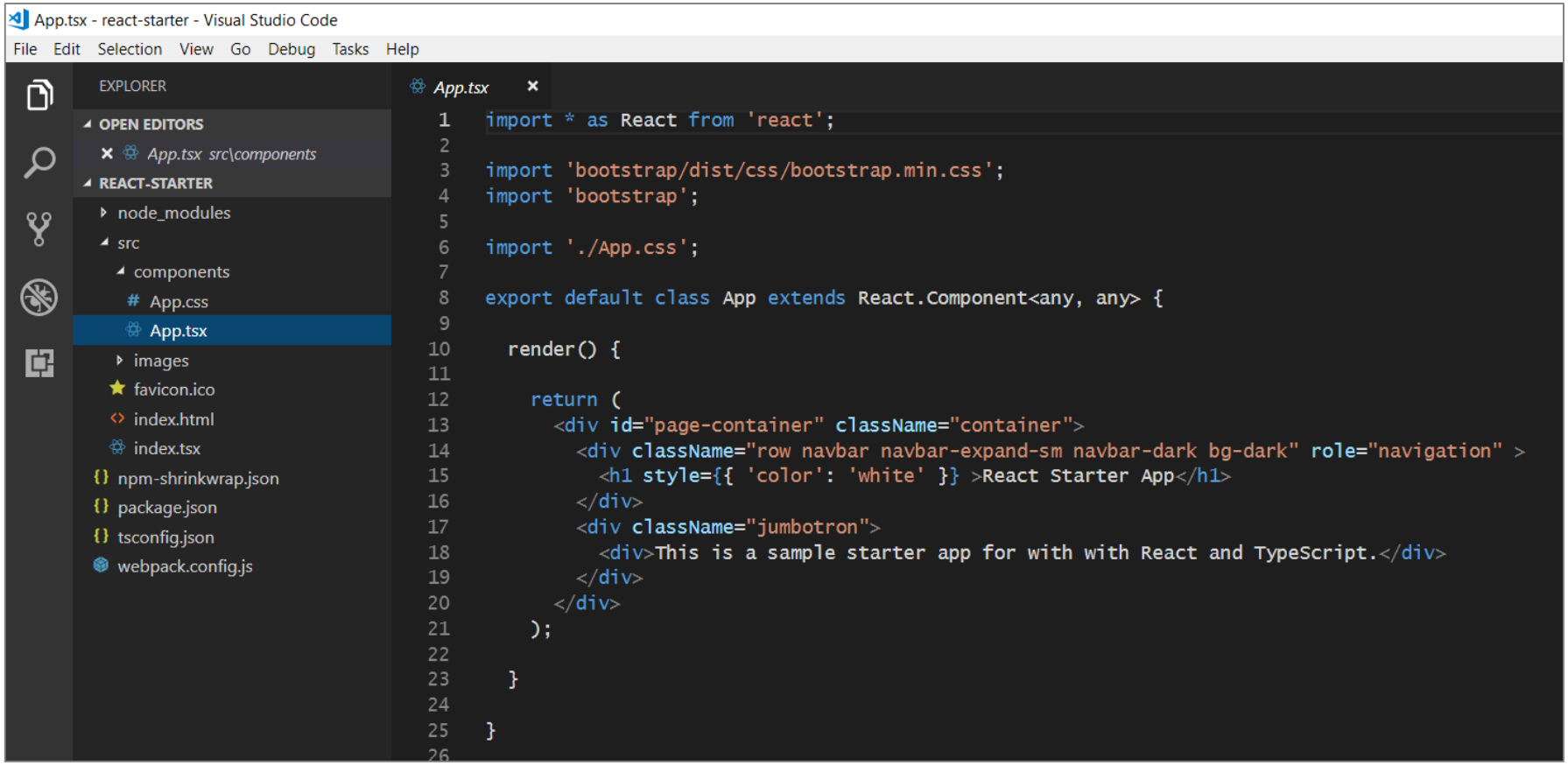
Starter Project - webpack.config.js



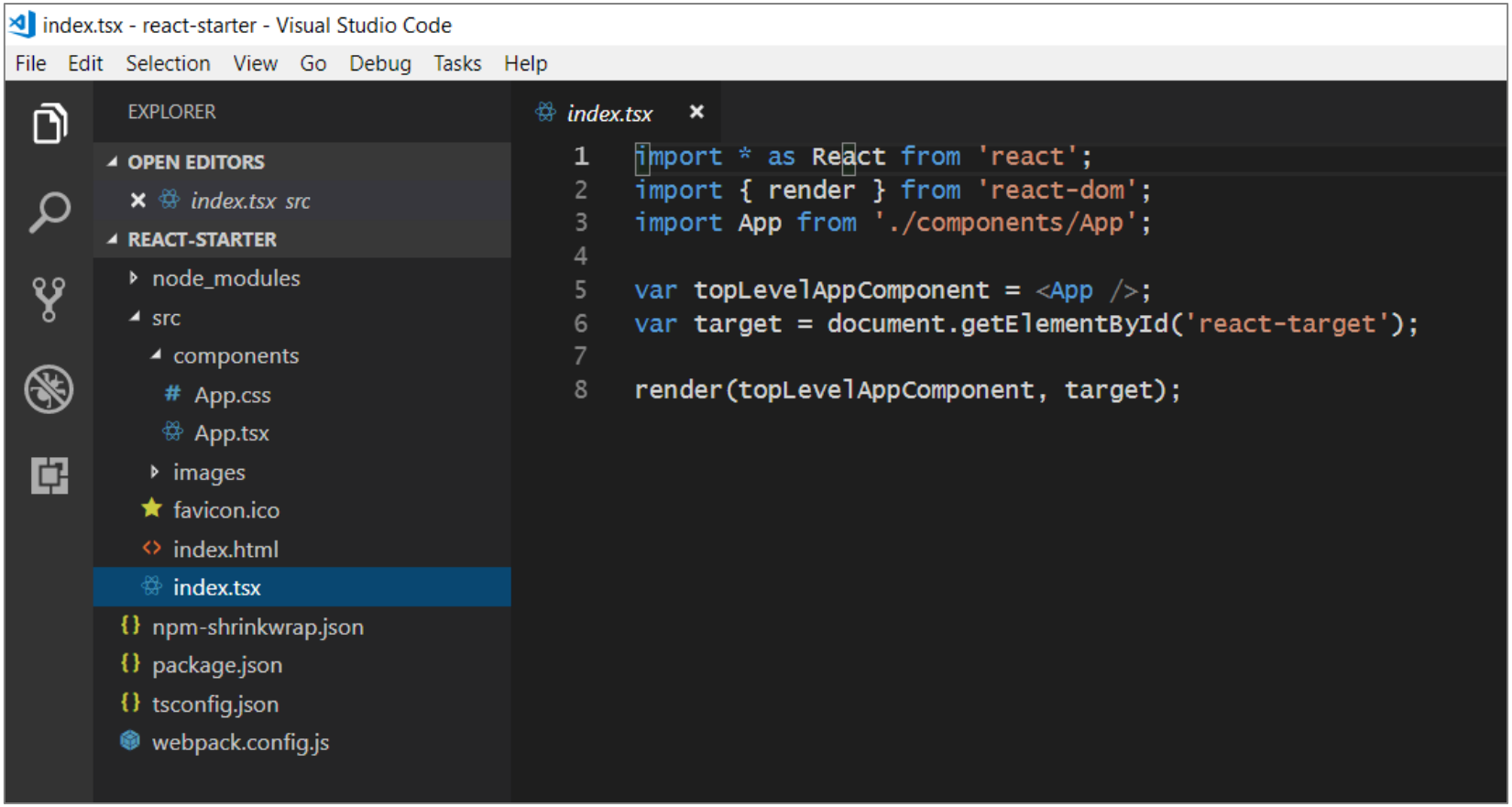
webpack.config.js

```
const path = require('path');

const HtmlWebpackPlugin = require('html-webpack-plugin');
const CopyWebpackPlugin = require('copy-webpack-plugin');
const CleanWebpackPlugin = require('clean-webpack-plugin')

module.exports = {
    entry: './src/index.tsx',
    output: {
        filename: 'scripts/bundle.js',
        path: path.resolve(__dirname, 'dist'),
    },
    resolve: {
        extensions: ['.js', '.json', '.ts', '.tsx'],
    },
    plugins: [
        new CleanWebpackPlugin(['dist']),
        new HtmlWebpackPlugin({ template: path.join(__dirname, 'src', 'index.html') }),
        new CopyWebpackPlugin([{ from: './src/favicon.ico', to: 'favicon.ico' }])
    ],
    module: {
        rules: [
            { test: /\.(ts|tsx)$/, loader: 'awesome-typescript-loader' },
            { test: /\.css$/, use: ['style-loader', 'css-loader'] },
            { test: /\.(png|jpg|gif)$/, use: [{ loader: 'url-loader', options: { limit: 8192 } }] }
        ],
    },
    mode: "development",
    devtool: 'source-map',
    devtool: 'cheap-eval-source-map'
};
```

The Top-level App Component



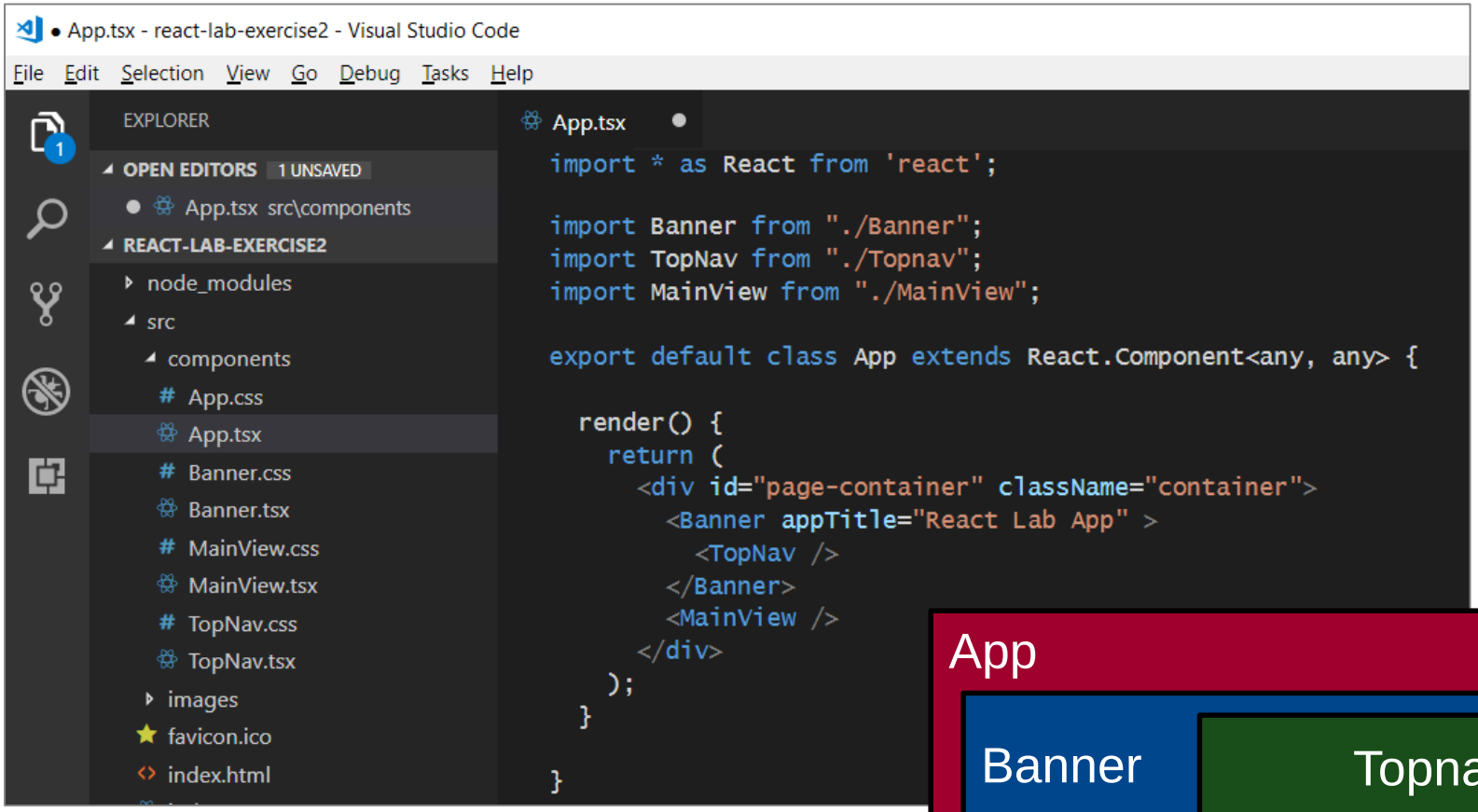
Bootstrapping the App Component

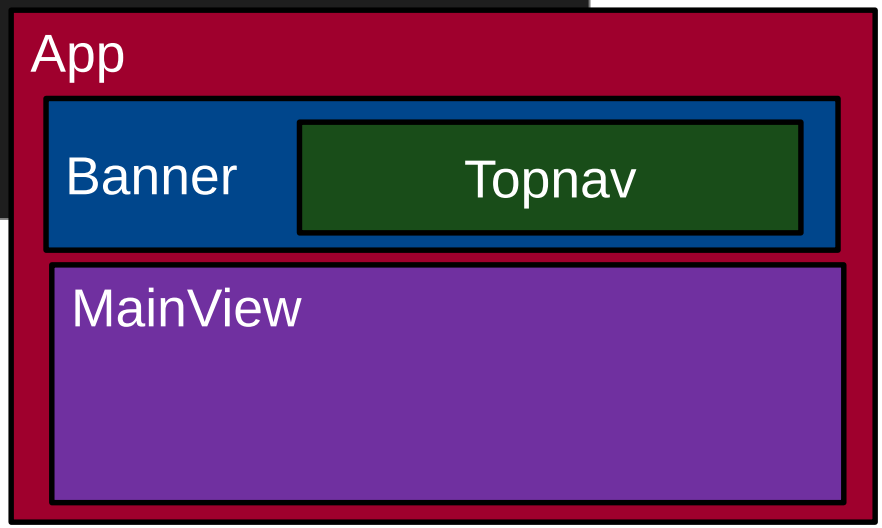


Agenda

- ✓ Getting Started with React.js
- ✓ Creating SPAs using React.js, TypeScript and Webpack
- ➢ Designing a React Component Hierarchy
- Extending a React Project with the React Router
- Understanding the React Component Lifecycle Methods
- Calling Across the Network using the Fetch API

React Component Hierarchies





Agenda

- ✓ Getting Started with React.js
- ✓ Creating SPAs using React.js, TypeScript and Webpack
- ✓ Designing a React Component Hierarchy
- ➢ Extending a React Project with the React Router
- • Understanding the React Component Lifecycle Methods
- • Calling Across the Network using the Fetch API

React Router

- Used to create route map in single page application (SPA)
  - Installed as a pair of npm packages

```
npm install react-router @types/react-router --save-dev
npm install react-router-dom @types/react-router-dom --save-dev
```

- Router must be added in as top-level component above App

index.tsx

```tsx
import * as React from 'react';
import { render } from 'react-dom';
import App from './components/App';
import { HashRouter } from 'react-router-dom';

var topLevelAppComponent =
  <HashRouter>
    <App />
  </HashRouter>;

var target = document.getElementById('react-target');

render(topLevelAppComponent, target);
```

Using React Router

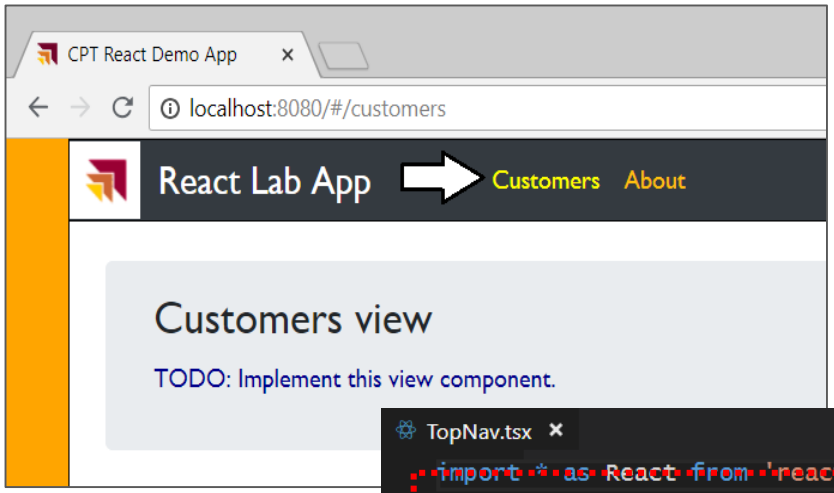
- Import Route and Switch components

```
import * as React from 'react';
import { Route, Switch } from 'react-router-dom';
```

- Create route map in HTML output

```
export default class App extends React.Component<any, any> {

  render() {

    return (
      <div id="page-container" className="container">
        <Banner appTitle="React Lab App" >
          <TopNav />
        </Banner>
        <Switch>
          <Route path="/" exact component={ViewHome} />
          <Route path="/customers" component={ViewCustomers} />
          <Route path="/about" component={ViewAbout} />
        </Switch>
      </div>
    );
```

Creating Route Links



TopNav.tsx

```tsx
import * as React from 'react';
import { Link, NavLink } from 'react-router-dom';

import "./TopNav.css";

export default class TopNav extends React.Component<any, any> {

  render() {
    return (
      <div id="top-nav" className="navbar-collapse collapse" >
        <nav>
          <ul className="nav navbar-nav" >
            <li className="nav-item" >
              <NavLink exact to="/" className="navbar-link" activeClassName="active-nav-link" >
                Home
              </NavLink>
            </li>
            <li className="nav-item" >
              <NavLink to="/customers" className="navbar-link" activeClassName="active-nav-link" >
                Customers
              </NavLink>
            </li>
```

Agenda

- ✓ Getting Started with React.js
- ✓ Creating SPAs using React.js, TypeScript and Webpack
- ✓ Designing a React Component Hierarchy
- ✓ Extending a React Project with the React Router
- ➢ Understanding the React Component Lifecycle Methods
- • Calling Across the Network using the Fetch API

Component Lifecycle

- componentWillUpdate
  - executed before component is rendered
- componentDidUpdate
  - executed after component is rendered
- componentWillMount
  - executed before node is added to the DOM
- componentDidMount
  - executed after node is added to the DOM
- componentWillUnmount
  - executed before node is removed from the DOM
- shouldComponentUpdate(newProps, newState)
  - executed before component is updated

Agenda

- ✓ Getting Started with React.js
- ✓ Creating SPAs using React.js, TypeScript and Webpack
- ✓ Designing a React Component Hierarchy
- ✓ Extending a React Project with the React Router
- ✓ Understanding the React Component Lifecycle Methods
- ➢ Calling Across the Network using the Fetch API

Calling a Web Service using the Fetch API

```
getCustomers(): Promise<ICustomer[]> {
  const restUrl =
    "http://subliminalsystems.com/api/Customers/?" +
    "$select=CustomerId,LastName,FirstName,EmailAddress,WorkPhone,HomePhone,Company" +
    "&$filter=(CustomerId+le+12)&$top=200";
  return fetch(restUrl)
    .then(response => response.json())
    .then(response => {
      console.log(response.value);
      return response.value;
    });
}
```

```
getCustomer(customerId: string): Promise<ICustomerDetail> {
  const restUrl = "http://subliminalsystems.com/api/Customers(" + customerId + ")";
  return fetch(restUrl)
    .then(response => response.json())
    .then(response => {
      console.log(response);
      return response;
    });
}
```

Summary

- ✓ Getting Started with React.js
- ✓ Creating SPAs using React.js, TypeScript and Webpack
- ✓ Designing a React Component Hierarchy
- ✓ Extending a React Project with the React Router
- ✓ Understanding the React Component Lifecycle Methods
- ✓ Calling Across the Network using the Fetch API