Exploration — Introduction to React

Introduction

We have seen how using JavaScript we can write applications that run in the browser and can update the webpage entirely on the client-side based on user interaction. But using JavaScript to manipulate the DOM requires writing low level code which is effort intensive and error prone. To overcome this limitation, higher level frameworks have been developed that allow the developer to write web applications in a more declarative fashion, i.e., by declaring what needs to be done and letting the framework map the declaration to low level JavaScript code. These frameworks include jQuery, Angular, Vue.js, etc. We will study React which is probably the mostly popular web framework currently used in the industry.

What is React?

React is a framework for writing front-end applications. React was created by Facebook but it is now open source and actively developed by many contributors. React applications are written in JavaScript with additional syntax added by React. Applications in React are based around the concept of **components** which are reusable units of UI. React creates a component tree from the components in our app, and maps it to the real DOM tree that a browser renders. When writing a React application, we modify the component, rather than directly modifying the DOM tree. The React framework then efficiently updates the real DOM to minimize the amount of rendering the browser has to do.

React: A High-Level View

Before we dive into the details of how to write React applications, let us look at some important concepts underlying React applications:

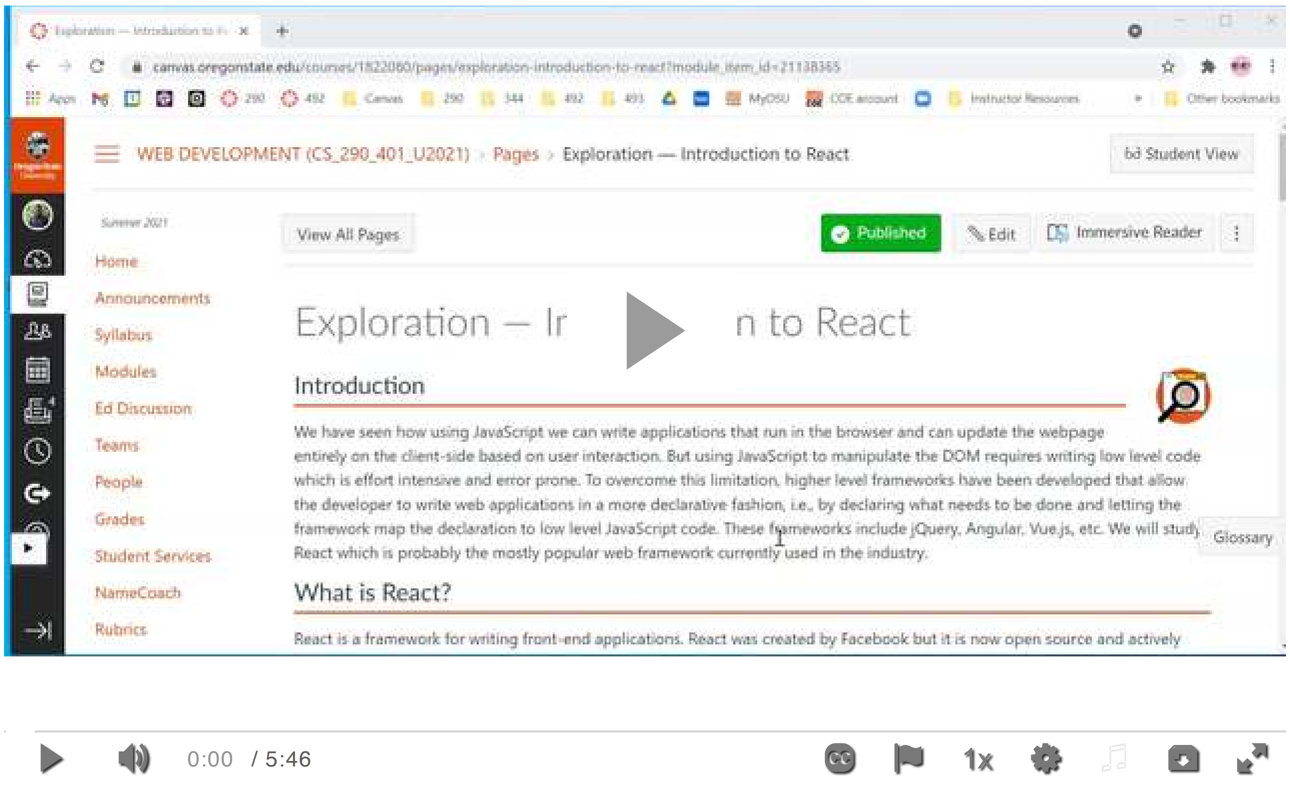
1. **We build pages with components, instead of HTML.**
    - We build React apps by writing components that are reusable units of UI. A page displayed to the user consists of a set of nested components.
    - For example, if we wanted to display a list of books, instead of directly writing an HTML table in our page, we would:
        - Create a component called “Book” to model a book in this list,
        - Create another component “BookList” which contains a list of “Book” components, and
        - Add the “BookList” component in a page.
2. **Components are functions that return React elements.** The React framework converts these React elements to DOM nodes.

- We write components using JSX, which is a special React syntax for mixing HTML in JavaScript code.
- We can use React components in our apps just as we use HTML tags.
- Browsers don't understand JSX and React components. But React framework libraries convert components into HTML and JavaScript, and then map/build a DOM tree that browsers can render.

3. **We update pages by changing state, not by directly changing HTML.**
    - To update a page, the code written by the React developer does not directly change the HTML for the page.
    - Instead, our code changes React **state** variables. Based on this change to state variables, React automatically renders the relevant parts of the DOM tree.
4. **We build Single Page Applications (SPAs**) rather than applications with multiple HTML pages.
    - In a traditional web app, when we navigate from one page to another, we send a request to the server and get back a an HTML document, with a new JavaScript environment, and possibly different static assets and CSS.
    - With SPAs the HTML, CSS, and JavaScript for a web app are sent from the web server to the browser exactly once. Once the web app is loaded into the browser, then JavaScript code makes changes to the DOM so that the user feels that they are navigating to a different page.
    - The web app still communicates with the server for various data operations, e.g., to fetch new data, or to update, create and delete data. But updates to the DOM tree are the responsibility of the client-side code and no longer are the concern of the server application.

If this high-level discussion leaves unanswered questions, do not worry. We are going to look at all of these concepts in detail in this module!

Creating a React App


canvas.oregonstate.edu/courses/1822080/pages/exploration-introduction-to-react?module_item_id=21138365
WEB DEVELOPMENT (CS_290_401_U2021) › Pages › Exploration — Introduction to React
Student View
Summer 2021
Home
Announcements
Syllabus
Modules
Ed Discussion
Teams
People
Grades
Student Services
NameCoach
Rubrics
View All Pages
Published
Edit
Immersive Reader
Glossary
0:00 / 5:46
1x

Exploration — Ir n to React

Introduction

We have seen how using JavaScript we can write applications that run in the browser and can update the webpage entirely on the client-side based on user interaction. But using JavaScript to manipulate the DOM requires writing low level code which is effort intensive and error prone. To overcome this limitation, higher level frameworks have been developed that allow the developer to write web applications in a more declarative fashion, i.e., by declaring what needs to be done and letting the framework map the declaration to low level JavaScript code. These frameworks include jQuery, Angular, Vue.js, etc. We will study React which is probably the mostly popular web framework currently used in the industry.

What is React?

React is a framework for writing front-end applications. React was created by Facebook but it is now open source and actively

React apps have a specific structure. While we can set up an app from scratch, React comes with a useful app generator that writes the boiler-plate code for us. We can create a React app by using the following command, substituting `my-app-name` with the name of the app we want to create.

```
npx create-react-app my-app-name --use-npm
```

Notes:

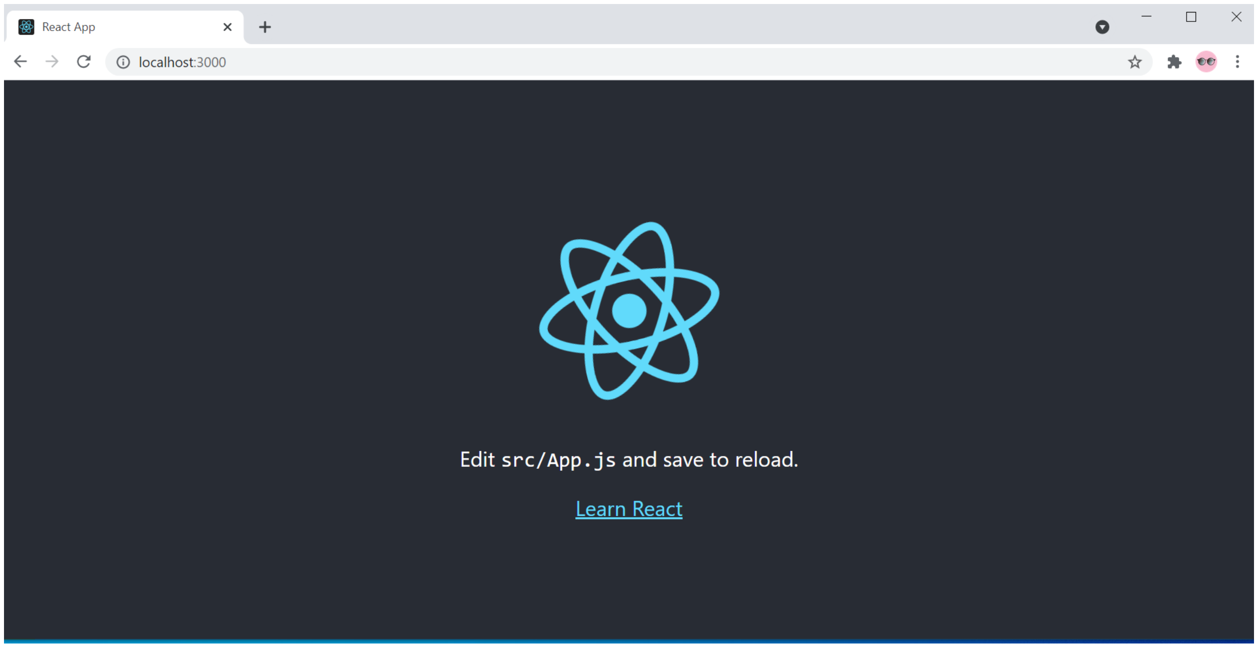
- As also noted in on **the React website (https://reactjs.org/docs/create-a-new-react-app.html)**, `npx` is not a typo in the command shown above. `npx` is a tool that comes bundled with `npm`.
- By default, React uses a popular package manager named `yarn`. `yarn` works a lot like `npm`. However, since we are using `npm` in this course, we added the flag `--use-npm` to the previous command so as to configure our React project to use `npm`.

Example

Let us create an app `hello-world`. To do this, go to the directory where you want to create the React app and run the following command:

```
npx create-react-app hello-world --use-npm
```

This command will create the directory `hello-world` with a boiler-plate React app. `cd` into this directory and run the command `npm start`. The server starts the app and it is available at the URL `http://localhost:3000`. Here is a screenshot of this default app:



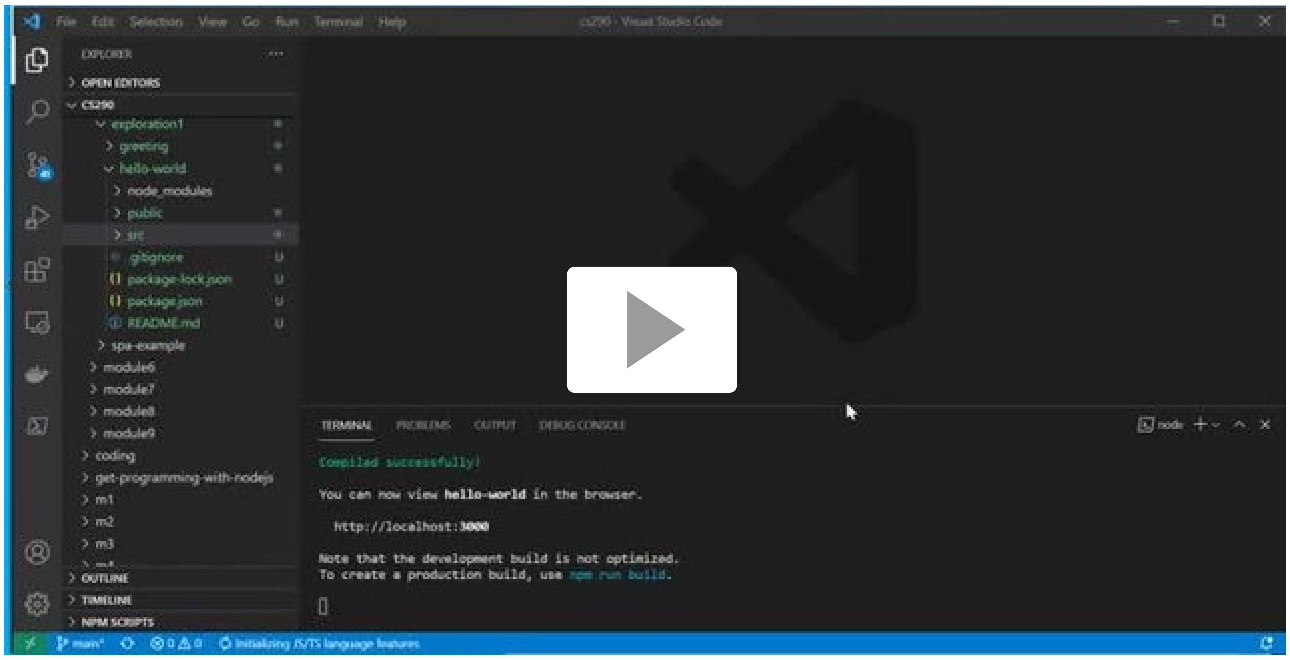
Edit src/App.js and save to reload.

The default React app includes a logo, the text “Edit src/App.js and save to reload” — **Learn React** (https://reactjs.org/)

Directory Structure and Important Files

`create-react-app` creates 3 directories within the top level directory which is `hello-world` in our example.

1. `node-modules`
    - This directory is similar to what we saw in our apps using Node.js and contains the Node modules the app depends on. Recall that a `package.json` file lists these dependencies and we can find that file at the top level directory.
2. `public`
    - This directory is similar to the `public` directory we saw in Express applications and is used to serve static content.
    - It contains the file `index.html` which is displayed when a user visits the app.
    - If we look at the `body` element in `index.html`, we will see that it contains just one `div` element `<div id="root"></div>`. As we will shortly see, the React framework renders the app by replacing this `div` element with the app content.
3. `src`
    - This directory contains the code for our React app. The two most important files in this directory are `index.js` and `App.js`



```
Compiled successfully!

You can now view hello-world in the browser.

  http://localhost:3000

Note that the development build is not optimized.
To create a production build, use npm run build.
```

0:00 / 7:46

index.js

Let us look at the file `index.js` which is the entry point to our React code. Here are the essential contents of the file.

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App';

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

There are two important packages that all React apps use, `react` and `react-dom`. These are imported in the first two lines of this file. From `react-dom` package, we use the function `ReactDOM.render()` which creates and renders the app. This function takes two arguments:

1. The first argument is the root component of our React component tree, in this case the `App` component.
2. The second argument specifies where in the DOM tree this component should be placed. We see that our `App` component will replace the `div` element with ID `root` in our `index.html` file.

When writing our own app, we typically do not make any changes to `index.js`. Instead, our changes start with `App.js` which is also imported by `index.js`. Let us look at `App.js` next.

App.js

This file defines a component. As we stated earlier, React components are JavaScript functions that return elements defined using JSX. Here is the default `App.js` file created for us.

```
import logo from './logo.svg';
import './App.css';

function App() {
return (
<div className="App">
  <header className="App-header">
    <img src={logo} className="App-logo" alt="logo" />
    <p>
      Edit <code>src/App.js</code> and save to reload.
    </p>
    <a
      className="App-link"
      href="https://reactjs.org"
      target="_blank"
      rel="noopener noreferrer"
    >
      Learn React
    </a>
  </header>
</div>
);
}

export default App;
```

The function `ReactDOM.render()` in the file `index.js` calls this function `App`, converts the content returned by it to HTML and replaces the `div` element in `index.html` with this HTML. We are not going to dissect this file. But note how the JavaScript function named `App` mixes HTML elements within JavaScript code.

Global semantic HTML page layout elements, such as <header>, <nav>, <main>, <section>, and <footer> get added to this file.

Summary

In this exploration, we introduced the React web framework and its fundamental concepts at a high-level. In the coming explorations, we will study these concepts in detail, starting with using JSX to define components.

Additional Resources

Here are some references to learn more about the topics we discussed in this exploration.

- The **official homepage** (https://reactjs.org/) of the React project.
- The **official React tutorial** (https://reactjs.org/tutorial/tutorial.html) .