tutorialr.com

workshop

React

Buy me a coffee

tutorialr.com

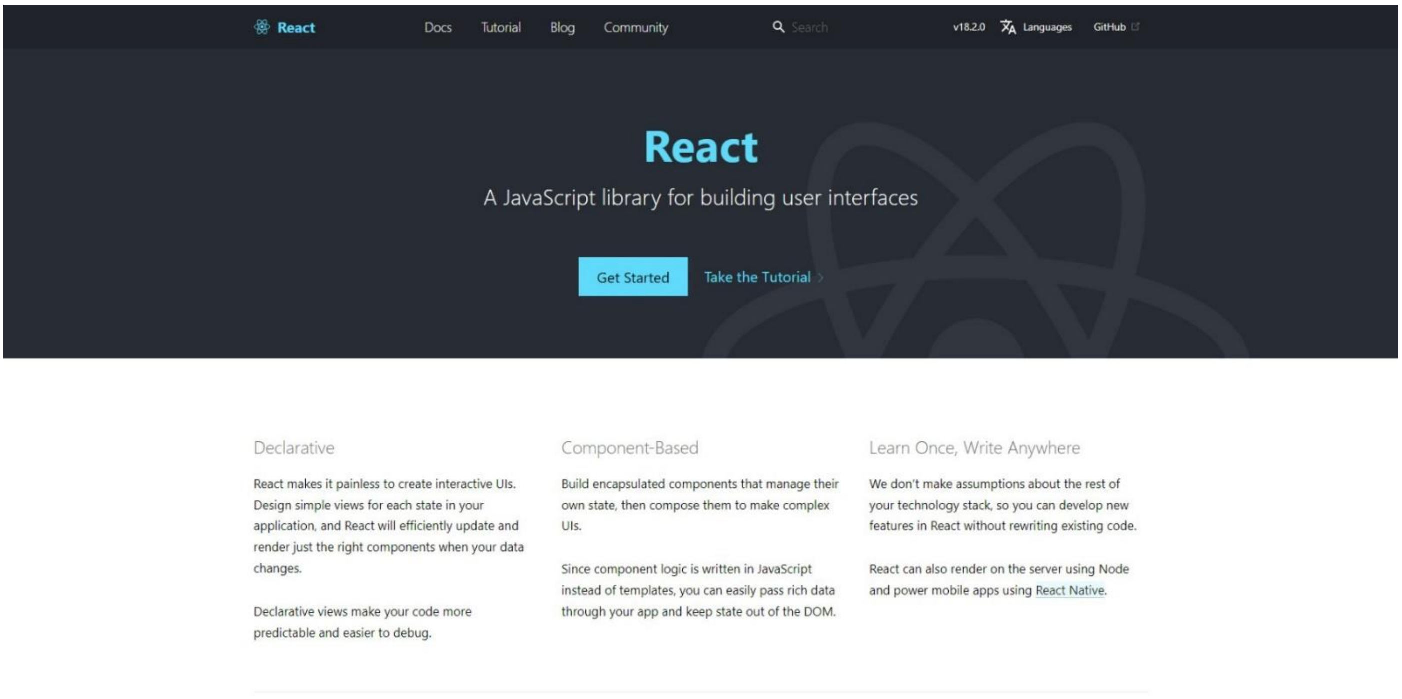
Contents

Contents

Introduction

What is React?

React is a **JavaScript** library for building interactive user interfaces created by **Facebook** that allows you to build Applications for the web and even take this further with **React Native** to build native mobile apps.



React

A JavaScript library for building user interfaces

Declarative

React makes it painless to create interactive UIs. Design simple views for each state in your application, and React will efficiently update and render just the right components when your data changes.

Declarative views make your code more predictable and easier to debug.

Component-Based

Build encapsulated components that manage their own state, then compose them to make complex UIs.

Since component logic is written in JavaScript instead of templates, you can easily pass rich data through your app and keep state out of the DOM.

Learn Once, Write Anywhere

We don't make assumptions about the rest of your technology stack, so you can develop new features in React without rewriting existing code.

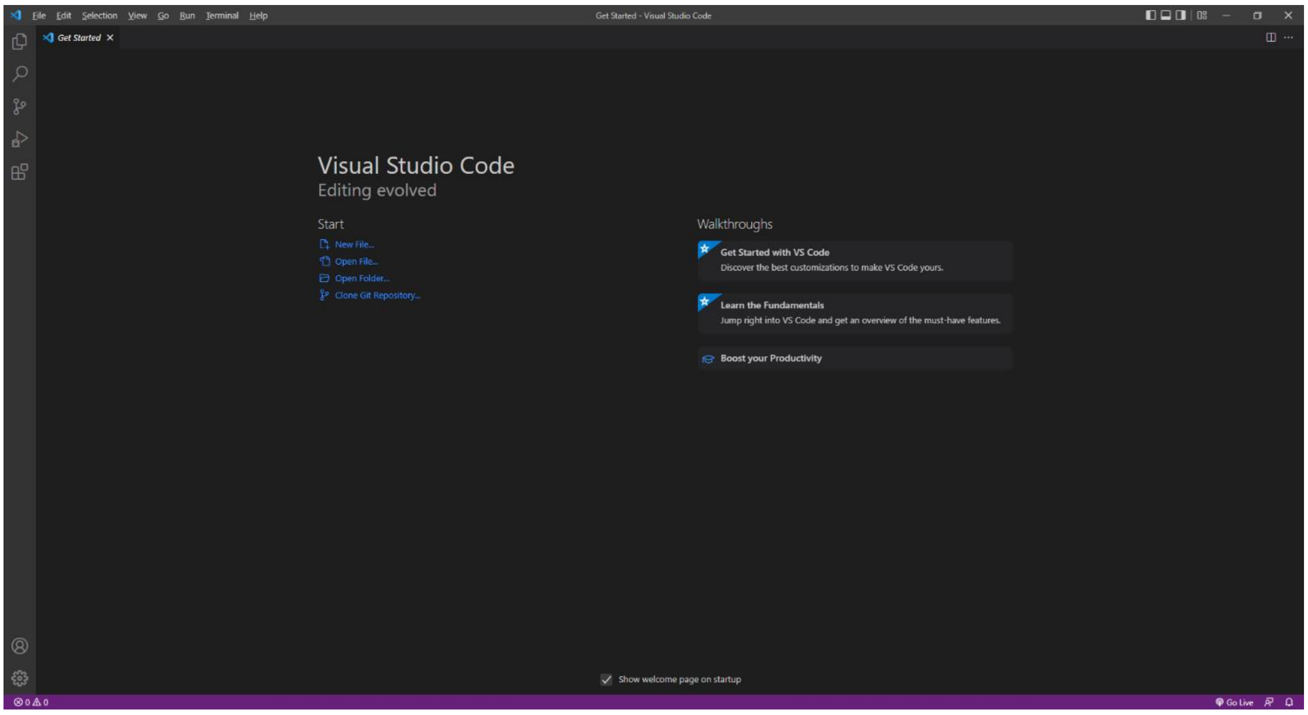
React can also render on the server using Node and power mobile apps using React Native.

React uses **JavaScript** which is a core programming language and development platform used on the web. **JavaScript** allows the web to be more interactive and offer dynamic content, **React** also requires **Node.js** which is a **JavaScript** runtime that allows Applications to be used such as **React** you can find out more about **Node.js** at nodejs.org. **React** along with any Applications created also depend on **npm** packages, **npm** is a software registry for applications, you can find out more about **npm** at docs.npmjs.com.

React makes it straightforward to create interactive user interfaces, as well as have simple views for each state in your Application, and **React** will update efficiently only changing what needs to be changed when using **Components**, which can manage their own state and can compose them to make more complex layouts. **React** can also allow new features on existing **JavaScript** projects to be created without rewriting any existing code for an Application and you can find out more about **React** including documentation, examples and more at reactjs.org.

What is Visual Studio Code?

Visual Studio Code will help create **React** applications even more easily, it is a free **Integrated Development Environment or IDE** created by **Microsoft**.

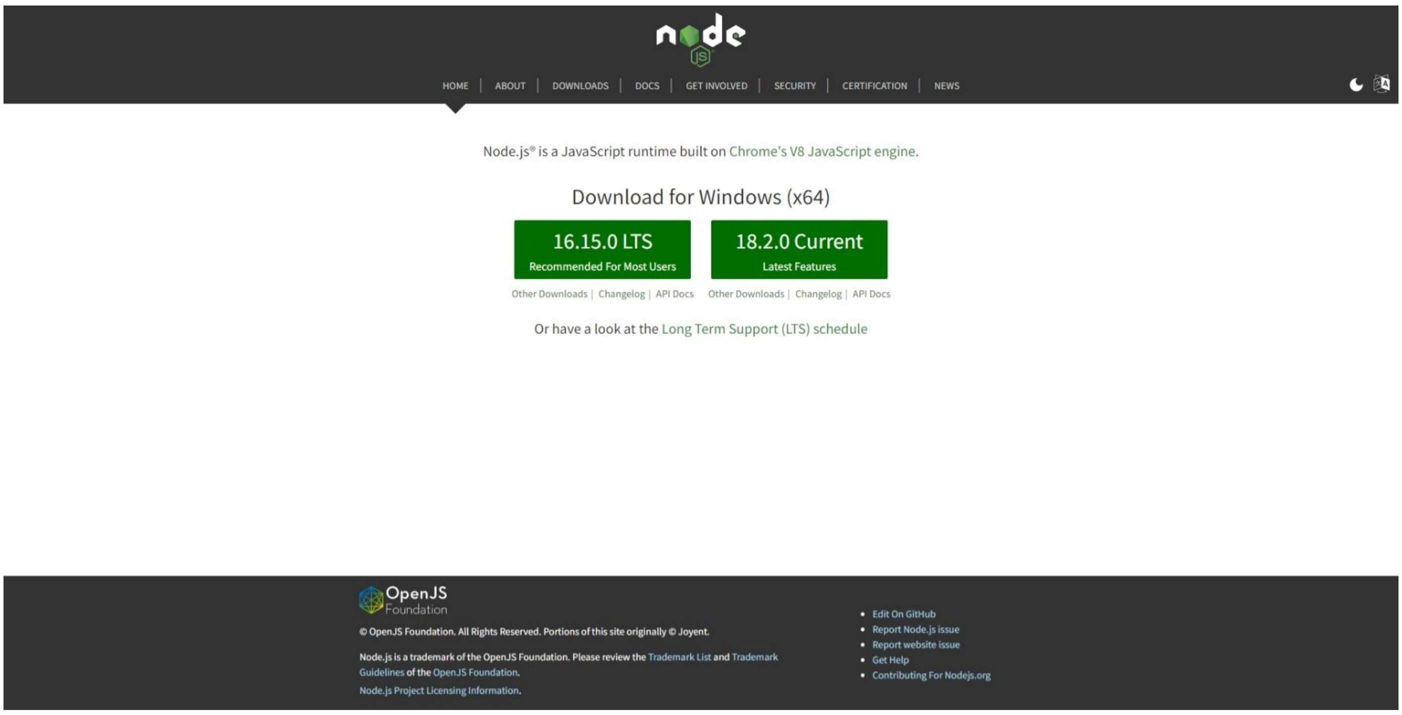

File Edit Selection View Go Run Terminal Help
Get Started - Visual Studio Code
Get Started
Visual Studio Code
Editing evolved
Start
New File...
Open File...
Open Folder...
Clone Git Repository...
Walkthroughs
Get Started with VS Code
Discover the best customizations to make VS Code yours.
Learn the Fundamentals
Jump right into VS Code and get an overview of the must-have features.
Boost your Productivity
Show welcome page on startup
Go Live

Visual Studio Code supports syntax highlighting which will add colours to certain parts of the text and make it easy to make sure everything is being entered correctly when writing **React** Applications. You can also use **Visual Studio Code** to edit any other **JavaScript**, **CSS**, **HTML** and more, making more than just creating **React** applications more straightforward. If you want to find out more about **Visual Studio Code** along with documentation, extensions and more you can visit code.visualstudio.com.

Setup and Start

React

React requires an **Active** or **Long Term Support / LTS** version of **Node.js** which if you don't have it already, you can **Download** the **LTS** version for your Platform such as **Windows** from nodejs.org.



Node.js® is a JavaScript runtime built on Chrome's V8 JavaScript engine.

Download for Windows (x64)

16.15.0 LTS
Recommended For Most Users

18.2.0 Current
Latest Features

Other Downloads | Changelog | API Docs   Other Downloads | Changelog | API Docs

Or have a look at the Long Term Support (LTS) schedule

OpenJS Foundation

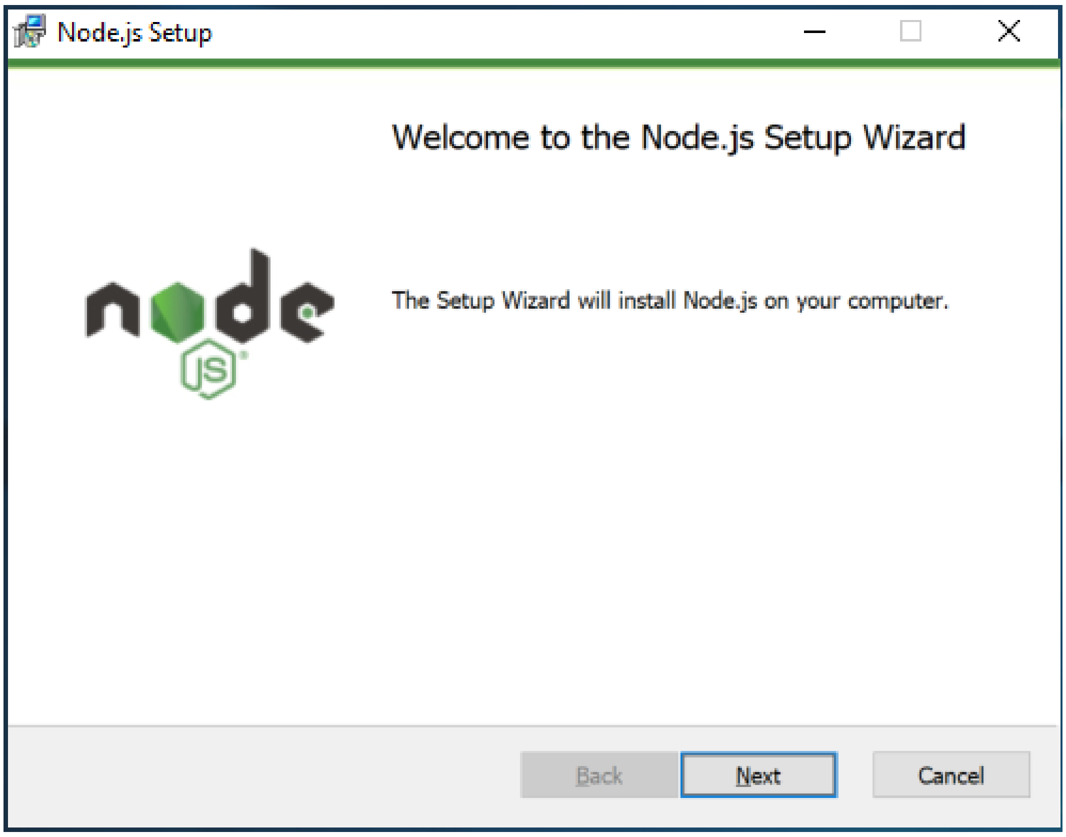
© OpenJS Foundation. All Rights Reserved. Portions of this site originally © Joyent.

Node.js is a trademark of the OpenJS Foundation. Please review the Trademark List and Trademark Guidelines of the OpenJS Foundation.

Node.js Project Licensing Information.

- Edit On GitHub
- Report Node.js issue
- Report website issue
- Get Help
- Contributing For Nodejs.org

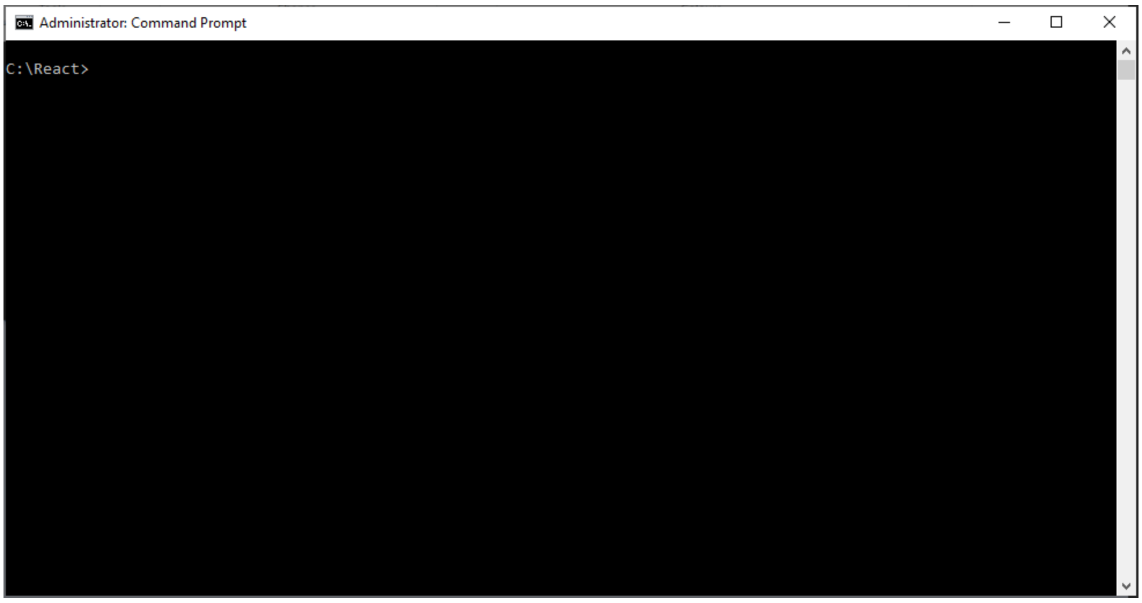
Once **Downloaded,** you can then **Install** it by following the steps in the **Installation Wizard**



Welcome to the Node.js Setup Wizard

The Setup Wizard will install Node.js on your computer.

Once **Node.js** has been **Installed**, or if it was already **Installed**, then if using **Windows** you need to go to **Start** then search for **Command Prompt** and then select it.



```
C:\React>
```

Once in the **Command Prompt** you can use **mkdir** to **Create** a new **Folder**, then **cd** to **Change Directory** to this new **Folder** as follows:

```
mkdir React
cd React
```

Then you can type in the following to create a single-page Application using the **Create React App** command with **npx** Package Runner which comes with **Node.js** and then press **Enter**:

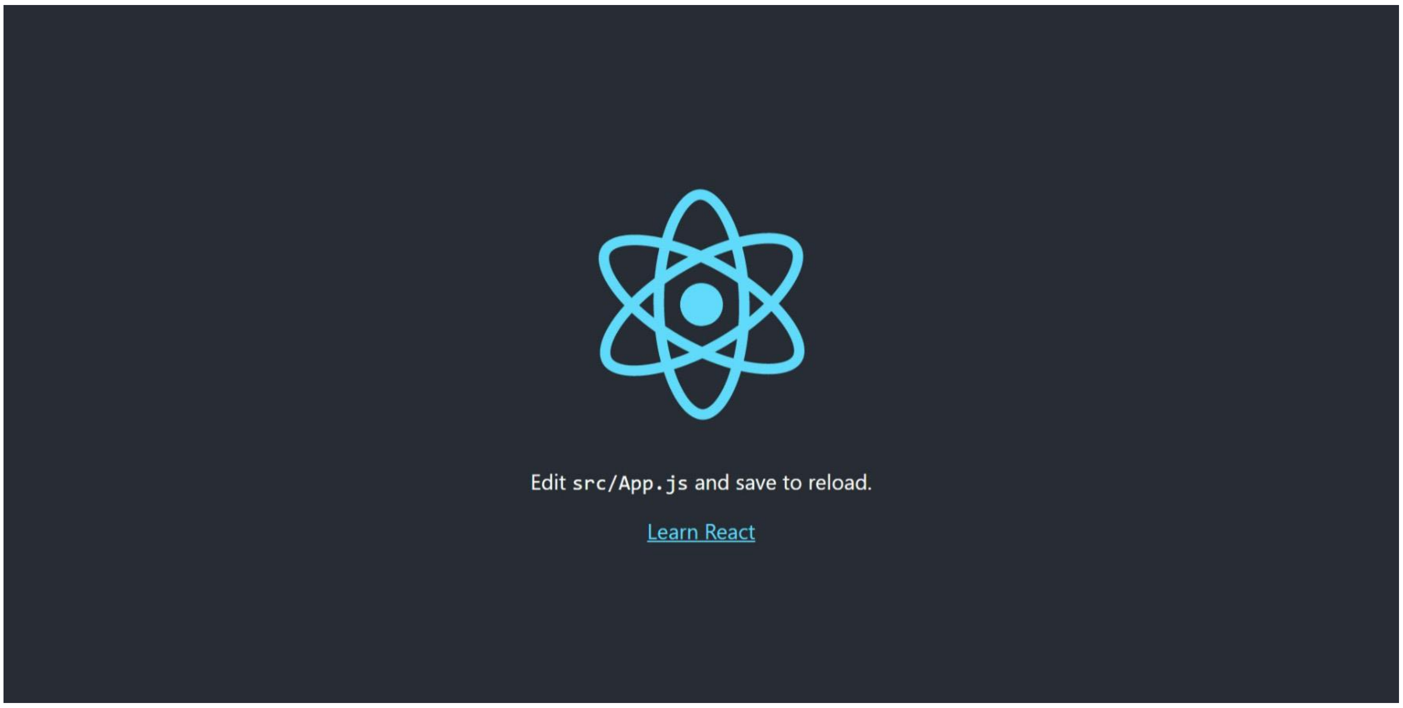
```
npx create-react-app workshop
```

You will be asked **Ok to proceed?** You can just accept the default by pressing **Enter**. After this in the **Command Prompt** you will need to change to the **Folder** for the **Workshop** by typing in the following command and then press **Enter**:

```
cd workshop
```

Once done while still in the **Command Prompt** you can type in the following command followed by **Enter** which will **Build** and **Serve** the Application which will also display it in your **Browser** you need to keep the **Command Prompt** open but you won't need to do anything else using this **Command Prompt**.

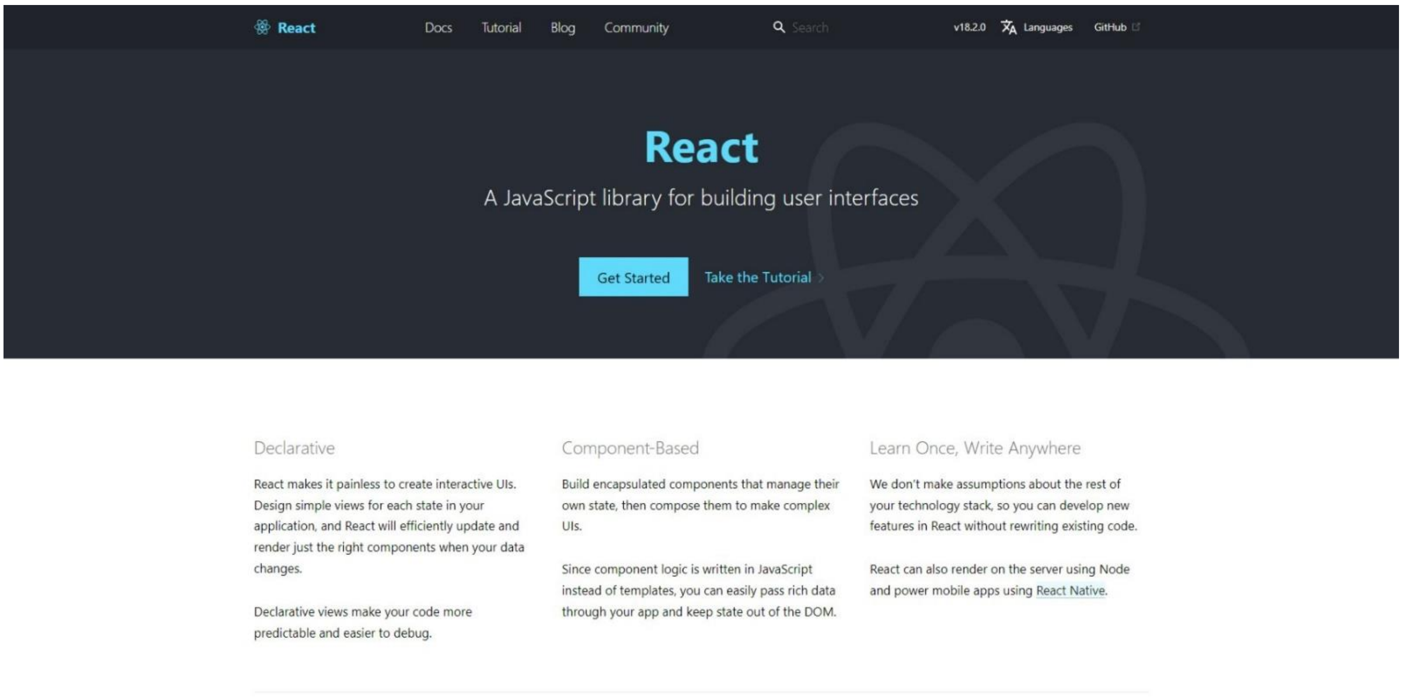
```
npm start
```



Edit `src/App.js` and save to reload.

Learn React

Should you need to you can get information, documentation and more about **React** at reactjs.org.



React

A JavaScript library for building user interfaces

Declarative

React makes it painless to create interactive UIs. Design simple views for each state in your application, and React will efficiently update and render just the right components when your data changes.

Declarative views make your code more predictable and easier to debug.

Component-Based

Build encapsulated components that manage their own state, then compose them to make complex UIs.

Since component logic is written in JavaScript instead of templates, you can easily pass rich data through your app and keep state out of the DOM.

Learn Once, Write Anywhere

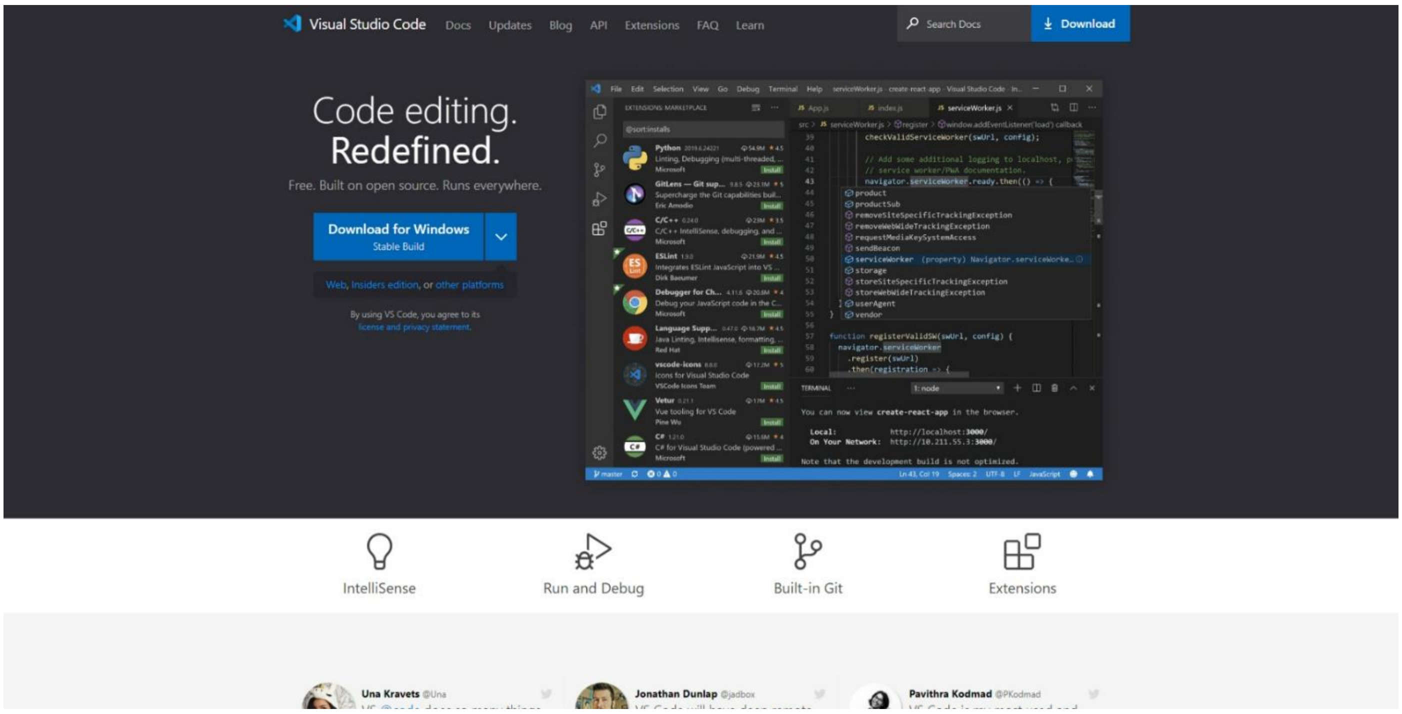
We don't make assumptions about the rest of your technology stack, so you can develop new features in React without rewriting existing code.

React can also render on the server using Node and power mobile apps using React Native.

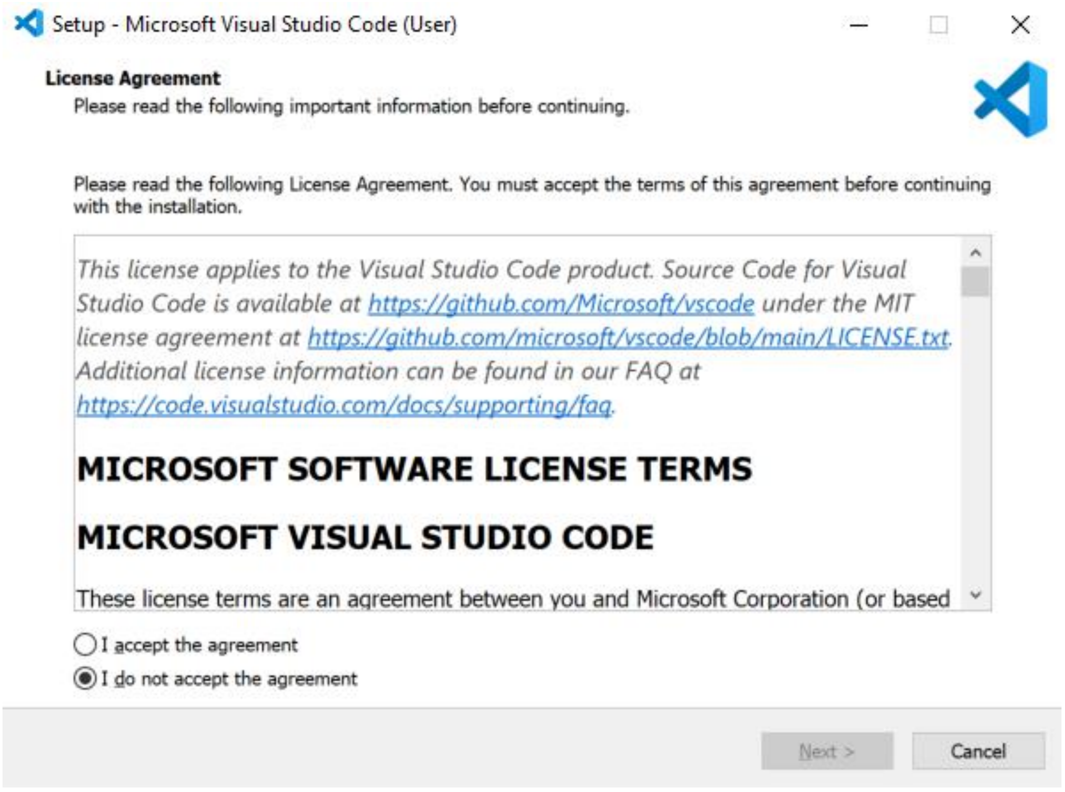
This **Workshop** supports at least **Version** *15* of **React** with **Version** *18* being used throughout.

Visual Studio Code

To be able to **Edit** your Application you will need to **Download**, if you don't have it already, **Visual Studio Code** for your Platform such as **Windows** from code.visualstudio.com.



Once **Downloaded,** you can then **Install** it by following the steps in the **Installation Wizard**



Setup - Microsoft Visual Studio Code (User)

License Agreement

Please read the following important information before continuing.

Please read the following License Agreement. You must accept the terms of this agreement before continuing with the installation.

This license applies to the Visual Studio Code product. Source Code for Visual Studio Code is available at https://github.com/Microsoft/vscode under the MIT license agreement at https://github.com/microsoft/vscode/blob/main/LICENSE.txt. Additional license information can be found in our FAQ at https://code.visualstudio.com/docs/supporting/faq.

MICROSOFT SOFTWARE LICENSE TERMS

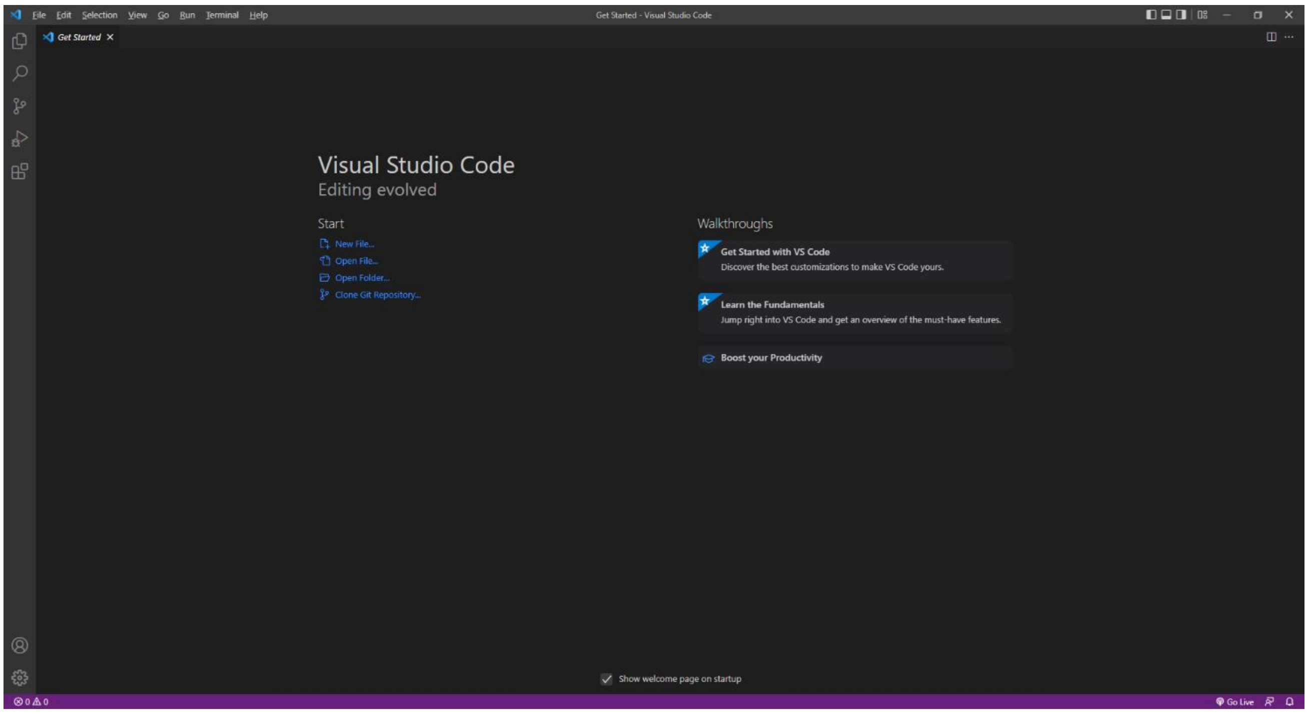
MICROSOFT VISUAL STUDIO CODE

These license terms are an agreement between you and Microsoft Corporation (or based

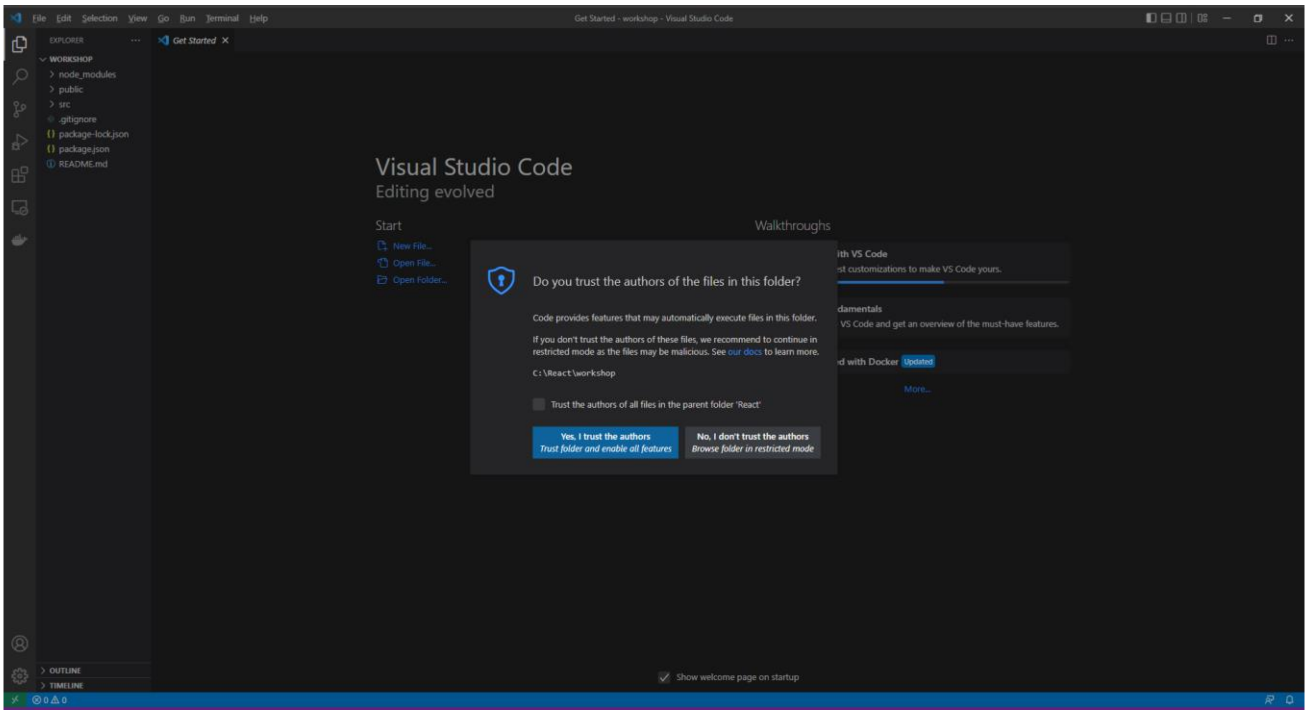
I accept the agreement [✓ I do not accept the agreement]

Next > Cancel

Once **Visual Studio Code** has been **Installed**, or if it was already **Installed**, then if using **Windows** you need to go to **Start** then search for **Visual Studio Code** and then select it.


File Edit Selection View Go Run Terminal Help
Get Started - Visual Studio Code
Get Started
Visual Studio Code
Editing evolved
Start
New File...
Open File...
Open Folder...
Clone Git Repository...
Walkthroughs
Get Started with VS Code
Discover the best customizations to make VS Code yours.
Learn the Fundamentals
Jump right into VS Code and get an overview of the must-have features.
Boost your Productivity
Show welcome page on startup
Go Live

Once **Visual Studio Code** has opened from the **Menu** choose **File** then **Open Folder...** then select the **Folder** for your Application e.g. *C:\React\workshop*. Then once the **Folder** has been opened Select the **Yes, I trust the authors** option in the **Do you trust the authors of the files in this folder?** if this is displayed.



Do you trust the authors of the files in this folder?

Code provides features that may automatically execute files in this folder.

If you don't trust the authors of these files, we recommend to continue in restricted mode as the files may be malicious. See our docs to learn more.

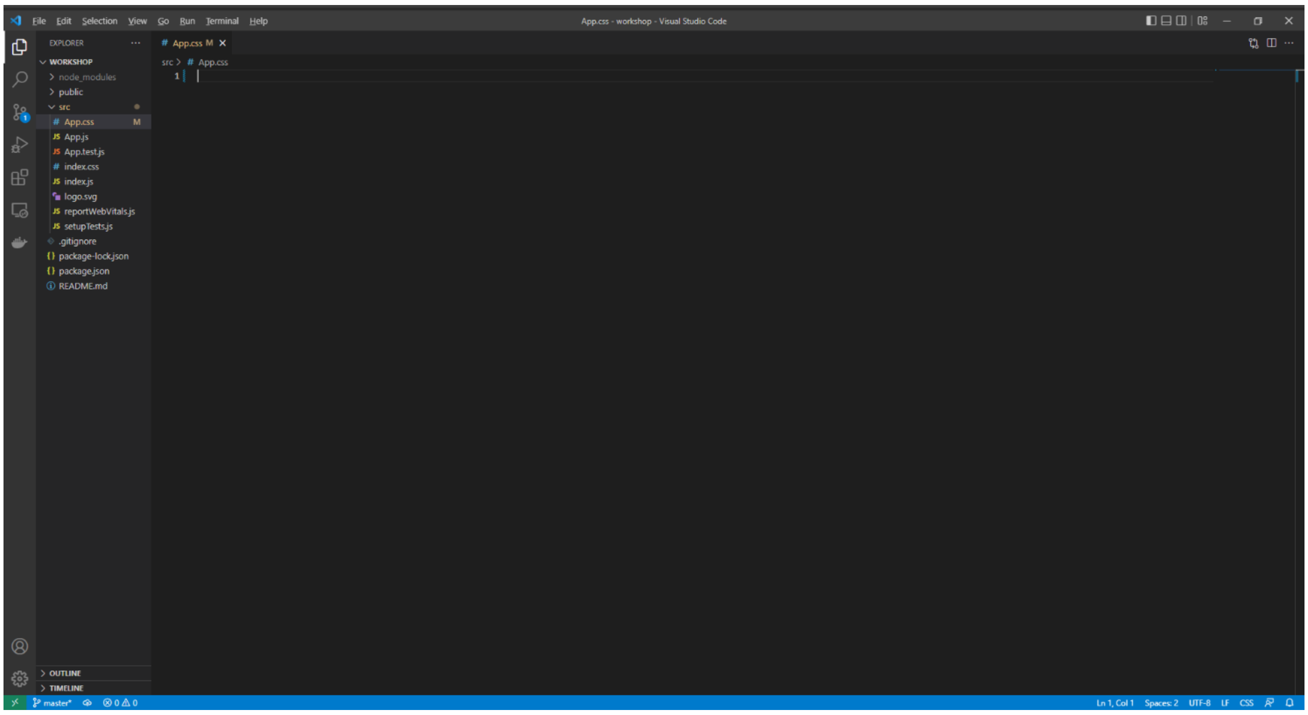
C:\React\workshop

Trust the authors of all files in the parent folder 'React'

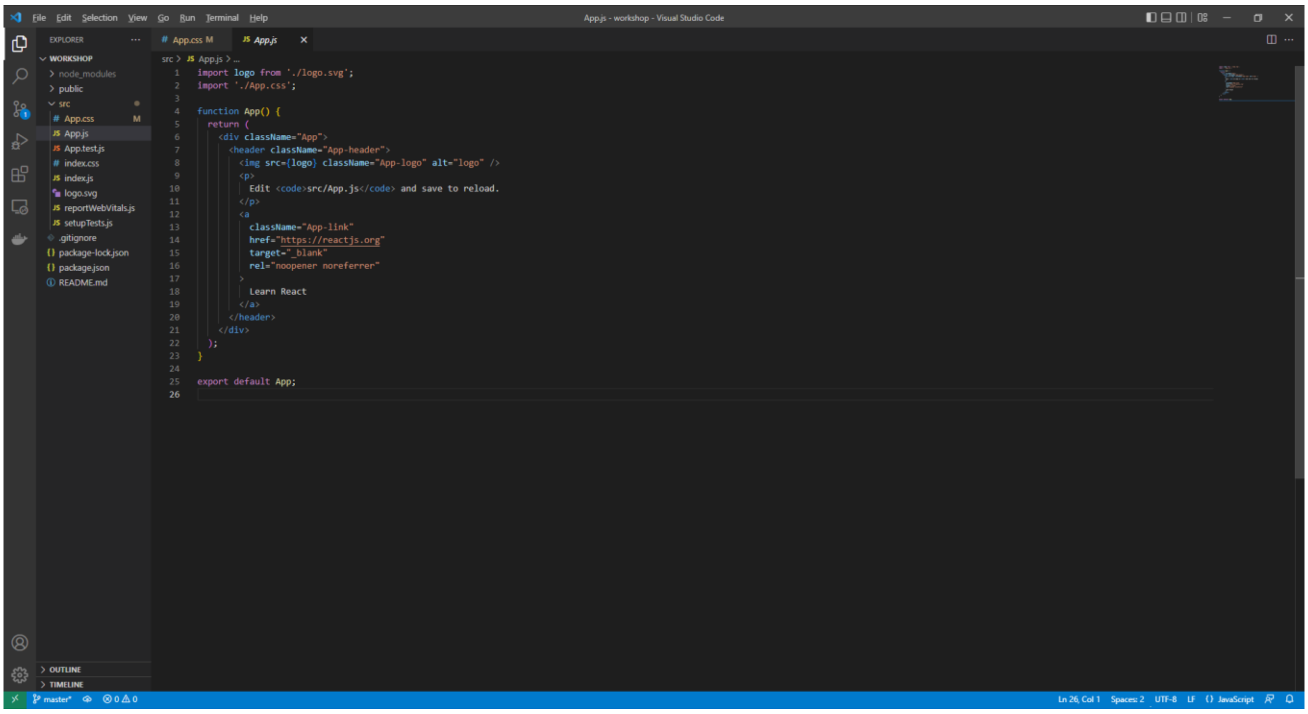
Yes, I trust the authors *Trust folder and enable all features*

No, I don't trust the authors *Browse folder in restricted mode*

Within **Visual Studio Code** will be the **Explorer** you can then Expand the **Folder** for **src** to find *App.css* file which defines any **CSS** Styles for the Application you should Clear the contents of this file so that it is blank as follows:



Then in **Visual Studio Code** from the **Menu** select **File** then **Save** to save these **Changes** to *App.css*. You should always do this when you make any Changes to *App.css* and other files.
You will also find the main **Component** for the Application which is *App.js* which is where you will be spending most of your time in the **Workshop**.



```
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;
```

You will also find *App.test.js* and see other files like these but they won't be used in the **Workshop** but they are used when **Testing** a **React** Application.

Within the **Component** of *App.js* you should also clear the contents and then in *App.js* type in the following:

```
import React from 'react';

import './App.css';
// Classes

// Variables

// Methods

function App() {
  return (
    <div className="App">

    </div>
  );
}

export default App;
```

Then in **Visual Studio Code** from the **Menu** select **File** then **Save** to save these **Changes** to *App.js*. You should always do this when you make any Changes to *App.js* and other files.

When you need to add or declare a **Class** for the **Component** of *App.js* in the **Workshop**, then these should be placed below the **Comment** of **`// Classes`** for each part of the **Workshop** you can also optionally create files for them such as *Class.js* instead if you want to.

Then when you need to add or declare a **Variable** for *App.js* then these should be placed on their own line below the **Comment** of **`// Variables`**

Finally when you need to add a **Method** for *App.js* then these should be placed below the **Comment** of **`// Methods`** for each part of the **Workshop**.

You can use the same **React** Application for each part of the **Workshop** and you do not need to remove anything else unless explicitly told to do so.

Components and Props

Components

Components in **React** can be **Function** or **Class** based and can use Inputs called **Props**. **Function** based **Components** are the simplest kind, like the **Component** for the Application of *App.js*.

After following **Setup and Start** in **Visual Studio Code** within the **Explorer** in the **Folder** of **src** in the **Component** for the Application of *App.js* below the **Comment** for **`// Variables`** type in the following **Variable**:

```
const message = 'Hello World';
```

To use this **Variable** in the **HTML** you can do so by enclosing it with curly braces of **`{`** and **`}`** by typing in below **`<div className="App">`** the following:

```
<h1>{message}</h1>
```

You can then select the **Browser** that was opened with **`npm start`** from the **Command Prompt** and you should see the text *Hello World* displayed in a **`h1`** Tag.

Props

You can also add a **Class** based **Component** to use **Props**. To do this, return to **Visual Studio Code** then in the **Component** for the Application of *App.js* below the **Comment** for **`// Classes`** type the following **Class**:

```
class Message extends React.Component {
    render() {
        return <h2>{this.props.value}</h2>;
    }
}
```

This **Class** for **`Message`** will **`extend`** the **`React.Component`** which has a **Method** for **`render`** and within this it uses a **`value`** of **`props`** for the **Props** and displays this within an **`h2`** element using **JSX** which is an extension to **JavaScript** to make working with **Elements** in **HTML** much simpler.

To use this **Component** below **`<h1>{message}</h1>`** type in the following:

```
<Message value="Hello Again!"/>
```

This will display the **`Hello Again!`** message in a **`h2`** Tag. You can select the **Browser** that was opened with **`npm start`** and you should see the text *Hello Again!*

Function based **Components** can also take advantage of **Props**, back in **Visual Studio** Code for the **Component** of *App.js*, below the **Comment** for `// Variables` and after any previously declared **Variables,** type in the following **Variable**:

```
const dateOfBirth = new Date('23-June-1912');
```

To add a **Function** based **Component**, type below the **Comment** for `// Methods` the following **Method**:

```
function AsDate(props) {
  return <div>{props.value.toDateString()}</div>
}
```

To use this **Component** below `<Message value="Hello Again!"/>`, type in the following:

```
<AsDate value={dateOfBirth}/>
```

If you switch to the **Browser** that was opened, you will see the **Date** being displayed as *Sun Jun 23 1912* which is Alan Turing's birthday, a pioneer in the field of computing.

CSS and Styles

CSS

In **React** you can apply **CSS** Styles either for defined **CSS** with **`className`**. After following **Setup and Start** and **Component and Props** in **Visual Studio Code** from **Explorer** in the **Folder** of **src** define some **CSS** for the Application in *App.css* by typing in the following:

```
.inverted {
  color: white;
  background-color: black;
}

.large {
  font-size: 2.0em;
}
```

In the **Component** of *App.js* below the **Comment** for **`// Variables`** and after any previously declared **Variables,** type in the following **Variable**:

```
const contrast = ['inverted', 'large'].join(' ');
```

This will create a list that then will be connected with **`' '`** using **`join`** as **CSS** needs to be defined when used in the **HTML** as a **String** with **`className`**. In **Visual Studio Code** while still in the **Component** of *App.js* below **`const dateOfBirth = new Date('23-June-1912');`** type in the following:

```
<div><span className={contrast}>Contrast</span></div>
```

If you switch over to the **Browser** that was opened with **`npm start`** from the **Command Prompt** it will have the Text of *Contrast* in *white* with a *black* Background.

Styles

In **React** you can use **`style`** for **CSS Styles**, you defined these as **Objects**, to define a **Style** in **Visual Studio Code** from **Explorer** within the **Folder** of **src** in the **Component** of *App.js* below the **Comment** for **`// Variables`** and after any previously declared **Variables,** type in the following **Variable**:

```
const style = { backgroundColor: 'yellow' };
```

You can use this with **`style`** enclosed in curly braces as **{** and **}** below **`<div><span className={contrast}>Contrast</span></div>`** by typing the following:

```
<div><span style={style}>Highlighted</span></div>
```

If you switch over to the **Browser** that was opened with **`npm start`** you will see the Text of *Highlighted* with a Background of *yellow*.

Context

Context allows data to be used at any **Component** level without having to pass **Props** down at each level. After following **Setup and Start**, **Component and Props** and **CSS and Styles** in **Visual Studio Code** from **Explorer** in the **Folder** of **src** in the **Component** of *App.js* below the **Comment** for **`// Variables`** and after any previously declared **Variables,** type in the following **Variable**:

```
const ImageContext = React.createContext('');
```

This will create an **ImageContext** using **`createContext`** with a default of an empty **String** of **`''`**.

You can use this **Context** within the **Component** for the Application of **App.js** by typing below the **Comment** for **`// Methods`** and after any previous **Method** the following **Method**:

```
function Image() {
  let image = React.useContext(ImageContext);
  return <img src={image} alt="React" height="150" width="150"/>
}
```

The **Component** will use **`useContext`** to get the Image then will return an **`img`** with the **`src`** set to the **Value** in the **Context**, to do this within the **Component** for the Application of **App.js** type below **`<div><span style={style}>Highlighted</span></div>`** the following:

```
<ImageContext.Provider value="https://openmoji.org/data/color/svg/1F600.svg">
  <Image/>
</ImageContext.Provider>
```

This will set the **Context** and then use the **Component** to display the Image, this **Component** could be nested within another **Component** and this would still work no matter how many levels there were, this **Context** would also be available to those **Components** so could use this in multiple places if needed.

Back in the **Browser** that was opened with **`npm start`** you should see a *Grinning Face* displayed, image courtesy of openmoji.org.

State

State allows for the storage of values that can be modified or used to control what should be or what is to be displayed. After following **Setup and Start**, **Component and Props**, **CSS and Styles** and **Context** from **Explorer** in **Visual Studio Code** from the **Folder** of **src** in the **Component** of *App.js* below the **Comment** for **`// Classes`** and after any previously declared **Classes,** type in the following **Class**:

```
class Change extends React.Component {
  change = event => {
      this.setState(
          { value: event.target.value }
      );
  }

  constructor(props) {
      super(props);
      this.state = {
          value: props.value
      }
  }

  render() {
      return (
          <div>
              <input type="text" onChange={this.change}/>
              <h2>{this.state.value}</h2>
          </div>
      );
  }
}
```

This **`class`** has an event in the **Method** for change this uses **`setState`** to set some **State** which will be the `value` from the **Event**. This **Component** has a **`constructor`** which can be provided with **Props** if needed to set the initial **State** and then there is the **Method** for **`render`** which will output an **`input`** which when changed or **onChange** will invoke the **Method** of **`change`** and the **`value`** in the **State** will be displayed in a **`h2`** Tag.

To use this **Component** in the **Component** of *App.js* below **`</ImageContext.Provider>`** type in the following:

```
<Change/>
```

If you switch over to the **Browser** that was opened previously and then type anything into the **`input`** it will be displayed below it in in a **`h2`** Tag.

Events

Events in **React** which is similar to handling events on elements however **Events** in **React** are named using **camelCase** and with **JSX** a **Method** is passed surrounded by curly braces of **{** and **}**.

After following **Setup and Start**, **Component and Props**, **CSS and Styles** and **Context** from **Explorer** in **Visual Studio Code** from the **Folder** of **src** to add a **Method** to be called from an **Event** within the **Component** of *App.js*, below the **Comment** for **`// Methods`** and after any previous **Methods** type in the following **Method**:

```
function showMessage() {
  let message = 'Hello World';
  alert(message);
}
```

This **`function`** will display an **`alert`** with the value of **`message`** when **`showMessage`** is called. This will be called from an **Event** on a **`button`** when it is clicked or **`onClick`** by typing in below **`<Change/>`** the following:

```
<button type="button" onClick={showMessage}>Show Message</button>
```

You can select the **Browser** opened with **`npm start`** from the **Command Prompt**, in the **Browser** you will see a **`button`** labelled *Show Message* which when **Clicked** will display an **`alert`** displaying the Message of *Hello World*.

Hooks

Hooks allow **React** features to be used without having to use a **Class** for **State** and **Effects** as well as being able to create your own.

To use the **Hook** for **State** after following **Setup and Start**, **Component and Props**, **CSS and Styles**, **Context** and **Events** in **Visual Studio Code** from **Explorer** in the **Folder** of **src** near the top of the **Component** of *App.js* after **`import React from 'react';`** type in the following:

```
import { useState } from 'react';
```

This will allow **Hook** for **`useState`** to be used. Then while still within the **Component** of *App.js* below the **Comment** for **`// Methods`** and after any previous **Methods** type in the following **Method**:

```
function ToggleStyle() {
  const [isSelected, selected] = useState(false);
  return (
  <div>
      <button style={{fontWeight: isSelected ? 'bold' : 'normal' }}
      onClick={() => selected((value) => value = !value)}>Toggle Style</button>
  </div>
  );
}
```

This will set the **Variable** of **`isSelected`** to be used for the **State** along with a **Callback** which will be used to change the value in the **State**.
Then for the **`return`** there is a **`button`** that when clicked or **`onClick`** will invoke the **Callback** and will change the value using the **`!`** operator which means **not** so that when **`isSelected`** is **`false`** will become **`true`** and the **`fontWeight`** of the a will be **`bold`** and when **`isSelected`** is **`true`** it will become **`false`** and the **`fontWeight`** of the **`button`** will be **`normal`**.
The **`?`** operator is used to define the behaviour when the **Value** is **`true`** before the **`:`** and **`false`** after it.

Again while still in the **Component** of *App.js* type in the following below **`<button type="button" onClick={showMessage}>Show Message</button>`**:

```
<ToggleStyle/>
```

Go to the **Browser** that was opened and **Click** the **`button`** with *Toggle Style* on it, this Text will toggle between being **`bold`** or **`normal`** when the **`button`** is **Clicked**.

To use the **Hook** for **Effects** which will perform **Side Effects** in **Components**. Near the top of the **Component** of *App.js* after **`import { useState } from 'react';`** type in the following:

```
import {useEffect } from 'react';
```

This will add the **Hook** for **`useEffect`**, then to use this in the **Component** of *App.js* below the **Comment** for **`// Methods`** and after any previous **Methods** type in the following **Method**:

```
function Sizer(props) {
  const [size, change] = useState(props.value);
  const resize = (delta) => change(() => Math.min(40, Math.max(8, + size + delta)));
  const decrease = () => {
    resize(-1);
  }
  const increase = () => {
    resize(+1);
  }

  useEffect(() => {
      document.getElementsByTagName('h1')[0].style.fontSize = size + 'px';
  })

  return (
    <div>
      <button type="button" onClick={decrease} title="Decrease">-</button>
      <button type="button" onClick={increase} title="Increase">+</button>
      <span style={{fontSize: size + 'px'}}>Font Size: {size}px</span>
    </div>
  );
}
```

This **Component** also uses **`useState`** and then defines a **Method** that will use the **Callback** for change to update the **Value**, the initial **`size`** will be passed in from the **Props**.
Then there is **`resize`** which will perform the updates using **`change`** to adjust the value of **`size`** which is used in the following **Methods** for **`decrease`** to reduce the **Value** and **`increase`** to make the **Value** larger.
Then **`useEffect`** is used to update the **`h1`** from the first part of the **Workshop** to match the **`size`** then the **Method** for **`render`** is used to output the Elements including **`button`** to call **`increase`** and **`decrease`** when they are clicked or **`onClick`** and then a **`span`** to display the current **`size`** and to **`style`** the Element the same way.

Then you can use this **Component** with the **`value`** of **`30`** by typing in below **`<ToggleStyle/>`** the following:

```
<Sizer value="30"/>
```

In the **Browser** you can use the *Sizer* to change the *font-size* of itself and at the top the **`h1`** of *Hello World!*

Ref

Ref allows access to **DOM** or **React** Elements that have been created. After following **Setup and Start**, **Component and Props**, **CSS and Styles**, **Context**, **Events** and **Hooks** from **Visual Studio Code** from the **Explorer** in the **Folder** of **src** and in the **Component** of *App.js* below the **Comment** for **`// Variables`** and after any previously declared **Variables,** type in the following **Variable**:

```
const inputMessage = React.createRef();
```

This will use **`createRef`** to create the **Ref** then to use this in the **Component** of *App.js* below the **Comment** for **`// Methods`** and after any previous **Methods** type in the following **Method**:

```
function MessageInput()
{
  const show = () => {
    alert(inputMessage.current.value);
  }

  return (
    <div>
      <input type="text" ref={inputMessage}/>
      <button type="button" onClick={show}>Show</button>
    </div>
  );
}
```

This **Component** defines a **Method** for `show` that will display the `current` contents of the **Ref** `value` with an **`alert`** and will display an **`input`** using the **`ref`** of **`inputMessage`** to get the **`value`** of the **`input`** then when the **`button`** is **Clicked** or **`onClick`** this will call the **Method** for **`show`** to display the message.

To use this **Component**, while still within the **Component** of *App.js* below **`<Sizer value="30"/>`** type in the following:

```
<MessageInput/>
```

In the **Browser** opened with **`npm start`** from the **Command Prompt** there should be a **`Button`** called *Show* that when **Clicked** will Display an **`alert`** with anything that was typed in the **`Input`** before it.

Conditions and Lists

Conditions

Conditions can be used to control **HTML** output in **React** this is done using standard **JavaScript** such as **`if`** and **`switch`**. After following **Setup and Start**, **Component and Props**, **CSS and Styles**, **Context**, **Events**, **Hooks** and **Ref** return to **Visual Studio Code** and within the **Component** of *App.js* found in the **Folder** of **src** below the **Comment** for **`// Methods`** and after any previous **Methods** type in the following **Method**:

```
function Toggle() {
  const [isShown, toggle] = useState(false);
  let message = '';
  if(isShown)
  {
    message = <h2>Hello World!</h2>
  }
  return (
  <div>
      <button onClick={() => toggle((value) => value = !value)}>Click Here</button>
      {message}
  </div>
  );
}
```

This will set the **Variable** of **`isShown`** to be used for the **State** along with a **Callback** which will be used to change the value in the **State**.
There is a **Variable** for **`message`** which used with **`if`** when **`isShown`** is **`true`** will be set to a **`h2`**. Then for the `return` there is a **`button`** that when clicked or **`onClick`** will invoke the **Callback** and will change the value using the **`!`** operator which means **not** so something that is **`false`** will become **`true`** and something that is **`true`** will become **`false`**.
Then **`message`** will be used to either be **`''`** when **`isShown`** is **`false`** and **`message`** was never set to anything else, or when **`isShown`** is **`true`** it will be set to the **`h2`**. To use this in the **Component** of *App.js* type below **`<MessageInput/>`** the following:

```
<Toggle/>
```

If you switch over to the **Browser** there will be a **`button`** of *Click Here* when **Clicked** will show then hide a **h2** below with the Text of *Hello World!*

Lists

Lists can be displayed using **`map`** which will take an **Array** of **Numbers** and allow their values to be displayed for each item, back in **Visual Studio Code** an **Array** can be defined below the **Comment** for **`// Variables`** of the **Component** of *App.js* and after any previously declared **Variables,** by typing in the following **Variables**:

```
const items = ['Hello', 'World'];
const itemElements = items.map((item) =>
  <li key={item}>{item}</li>
);
```

The **Variable** for **`items`** is an **Array** denoted with **`[`** and **`]`** and then there is **`itemElements`** which will represent each **`item`** using a **`li`** or **List Item** this is also used with a **`key`** as this is needed for items in a **List** when using **React**. To display the **`items`** type below **`<Toggle/>`** the following:

```
<ul style={{textAlign:'left'}}>{itemElements}</ul>
```

This will place the **`li`** Elements in their appropriate **Parent** which in this case is a **`ul`** for an **Unordered List** or **Bulleted List**, and will set the **`style`** to align the items to the **`left`** of the screen.

To see this switch over to the **Browser** that was opened with **`npm start`** from the **Command Prompt** there will be a **Bulleted List** showing the **List Items** of *Hello* and *World*.

You can also combine a **List** with a **Component** using **`switch`** to display values from a **Variable** back in **Visual Studio Code** below the **Comment** for **`// Variables`** for the **Component** of *App.js* and after any previously declared **Variables,** by typing in the following **Variable**:

```
const values = [
  {
    name: 'None',
    status: ''
  },
  {
    name: 'Danger',
    status: 'red'
  },
  {
    name: 'Warning',
    status: 'yellow'
  },
  {
    name: 'Proceed',
    status: 'green'
  }
];
```

This will define a list of items that will be used in the **Component** of *App.js*

While still in the **Component** of *App.js* below the **Comment** for **`// Classes`** and after any previously declared **Classes,** type in the following **Class**:

```
class Elements extends React.Component {
  render() {
    const display = (value) => {
      switch(value.status)
      {
        case 'red':
          return <span style={{backgroundColor: 'red'}}>Danger</span>
        case 'yellow':
          return <span style={{backgroundColor: 'yellow'}}>Warning</span>
        case 'green':
          return <span style={{backgroundColor: 'green'}}>Proceed</span>
        default:
          return <span>None</span>
      }
    }
    const elements = (values) =>
    {
      return values.map((item) =>
        <li key={item.name}>{display(item)}</li>
      );
    };
    return (
      <ul style={{textAlign:'left'}}>
        {elements(this.props.value)}
      </ul>);
  }
}
```

This **Component** is comprised of a single **Method** for **`render`** within this is a **Method** defined for **`display`** this contains the **`switch`** statement to control the **`span`** that will be displayed and then there is **`elements`** that will display this within an **`li`** which will then be returned inside an **`ul`** which is an **Unordered List** or **Bulleted List** with the **`style`** to align it to the **`left`**.

In the **Component** of *App.js* below **`<ul style={{textAlign:'left'}}>{itemElements}</ul>`** type the following:

```
<Elements value={values}/>
```

If you switch over to the **Browser** that was opened, there will be another **Bulleted List** showing the **List Items** of *None*, then *Danger* with a *red* Background, *Warning* with a yellow Background and *Proceed* with a *green* Background.

Forms

Forms in **React** can either be **Controlled** where each Element in the **Form** maintains their own state and this is updated based on **Input** by the user and is designed for smaller **Forms** to update a few **Values** or **Uncontrolled** where data for the **Form** is handled by the **DOM** itself.

Controlled

To create a **Controlled Component** after following **Setup and Start**, **Component and Props**, **CSS and Styles**, **Context**, **Events**, **Hooks**, **Ref** and **Conditions and Lists** return to **Visual Studio Code** and within the **Component** of *App.js* found in the **Folder** of **src** below the **Comment** for **`// Methods`** and after any previous **Methods** type in the following **Method**:

```
function Controlled() {
  const [name, setName] = useState('');
  const handleSubmit = (event) =>
  {
    event.preventDefault();
    alert(name);
  }

  return (
    <form onSubmit={handleSubmit}>
      <input id="name" type="text"
      onChange={(event) => setName(event.target.value)}/>
      <input type="submit" value="Controlled"/>
    </form>
  )
}
```

This **Component** uses **`useState`** to store **State** it defines **`handleSubmit`** to show an **`alert`** with **name** and in the **`form`** it has an **`input`** which when changed or **`onChange`** will call **`setName`** to set the **Value** in **State** and on submitting the **Form** the **`handleSubmit`** will be called. While still in the **Component** of *App.js* below **`<Elements value={values}/>`** type the following:

```
<Controlled/>
```

If you switch over to the **Browser** that was opened with **`npm start`** from the **Command Prompt** there should be a **`Button`** called *Controlled* that when **Clicked** will Display an **`alert`** with anything that was typed in the **`Input`** before it.

Uncontrolled

To create an **Uncontrolled Component** back in **Visual Studio Code** and within the **Component** of *App.js* found in the **Folder** of **src** below the **Comment** for `// Methods` and after any previous **Methods** type in the following **Method**:

```
function Uncontrolled() {
  let value = React.createRef();
  const handleSubmit = (event) =>
  {
    event.preventDefault();
    alert(value.current.value);
  }

  return (
    <form onSubmit={handleSubmit}>
      <input id="name" type="text" ref={value}/>
      <input type="submit" value="Uncontrolled"/>
    </form>
  )
}
```

This **Component** uses **`createRef`** to create a **Ref** and defines **`handleSubmit`** to show an **`alert`** with the **`value`** and in the `form` it has an **`input`** which can be typed into that uses the **`ref`** to set the **`value`** and on submitting the **Form** the **`handleSubmit`** will be called. While still in the **Component** of *App.js* below **`<Controlled/>`** type in the following:

```
<Uncontrolled/>
```

If you switch over to the **Browser** that was opened with **`npm start`** from the **Command Prompt** there should be a **`Button`** called *Uncontrolled* that when **Clicked** will Display an **`alert`** with anything that was typed in the **`Input`** before it and that concludes this **Workshop** about **React** from tutorialr.com!