

Presentation by Uplatz

Contact Us: <https://training.uplatz.com/>

Email: info@uplatz.com

Phone: +44 7836 212635

Table Of Contents:

- What is Reactjs
- Advantages of ReactJS
- Installation or Setup
- How to use NodeJs
- Software package Manager
- Webpack
- Create-React App
- Absolute Basics of Creating Reusable Components

What is ReactJS?

- ReactJS is an open-source, component based front end library responsible only for the view layer of the application. It is maintained by **Facebook**.
- ReactJS uses virtual DOM based mechanism to fill in data (views) in HTML DOM. The virtual DOM works fast owing to the fact that it only changes individual DOM elements instead of reloading complete DOM every time
- A React application is made up of multiple components, each responsible for outputting a small, reusable piece of HTML.
- Components can be nested within other components to allow complex applications to be built out of simple building blocks.

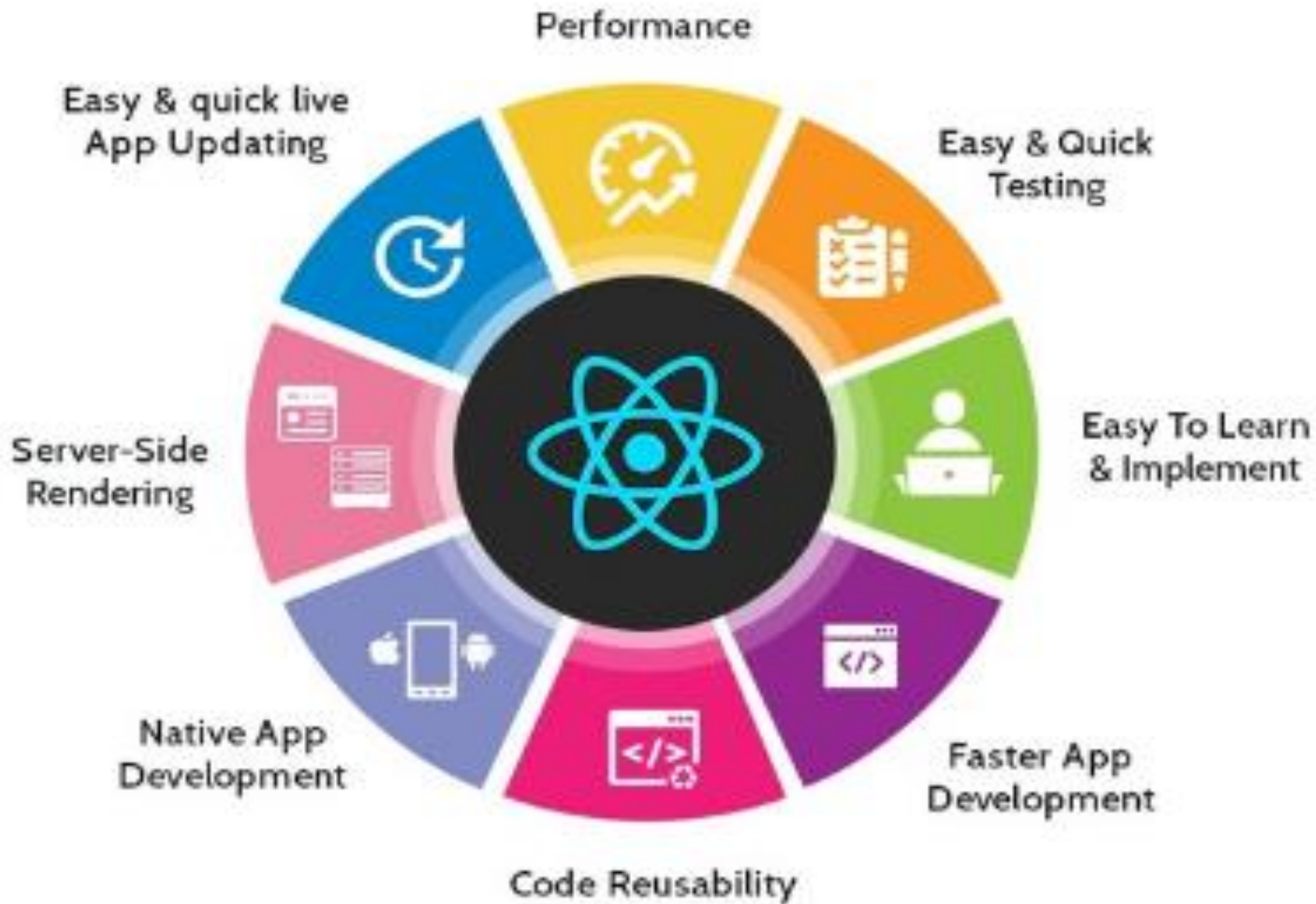
- A component may also maintain internal state - for example, a TabList component may store a variable corresponding to the currently open tab.
- React allows us to write components using a domain-specific language called JSX.
- JSX allows us to write our components using HTML, whilst mixing in JavaScript events.
- React will internally convert this into a virtual DOM, and will ultimately output our HTML for us.
- React "reacts" to state changes in your components quickly and automatically to rerender the components in the HTML DOM by utilizing the virtual DOM.
- The virtual DOM is an in-memory representation of an actual DOM.

- By doing most of the processing inside the virtual DOM rather than directly in the browser's DOM,
- React can act quickly and only add, update, and remove components which have changed since the last render cycle occurred.

when to use?

- React is definitely worthy of your attention, especially if you are working on single page application and wish to make it fast, responsive and user-friendly.
- However, this JS library can be hardly called a **“universal remedy”**

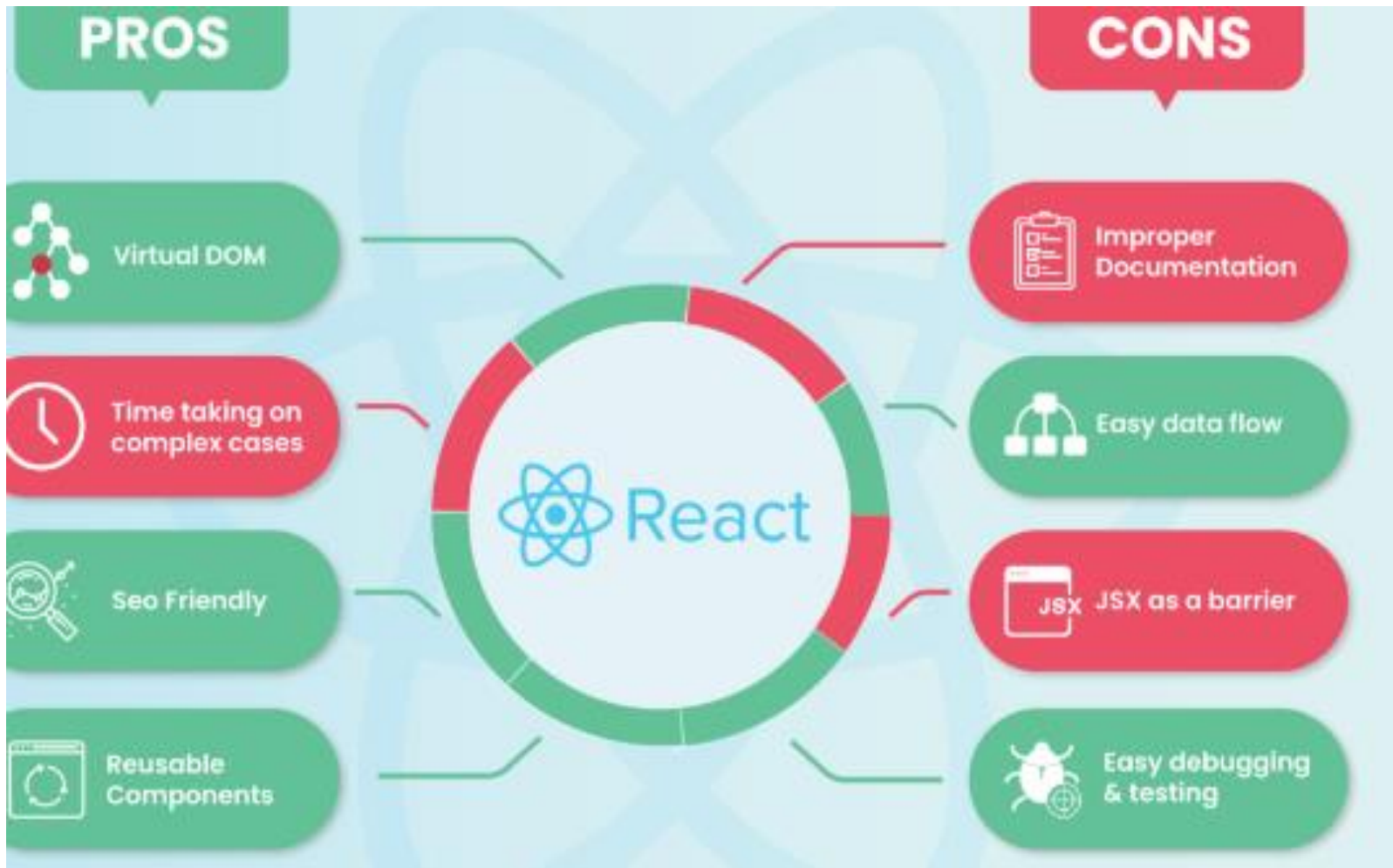
Why Reactjs?




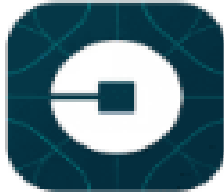







Why Reactjs so popular?



Pros and Cons of ReactJS



Where ReactJs Using?

1 000 000 000+	100 000 000+		50 000 000+
 Instagram	 Uber	 Musical.ly	 Amazon Prime Video
 Messenger	 Flipkart Online Shopping	 Microsoft OneDrive	 PhonePe
 Skype			


Installation Steps:


Step 1)


- Go to the site **<https://nodejs.org/en/download/>** and download the necessary binary files.
- In below example, I am going to download the 32-bit setup files for Node.js

Download the Node.js source code or a pre-built installer for your platform, and start developing today.

LTS
Mature and Dependable
Stable
Latest Features


Windows Installer
node-v4.2.3-x86.msi


Macintosh Installer
node-v4.2.3.pkg

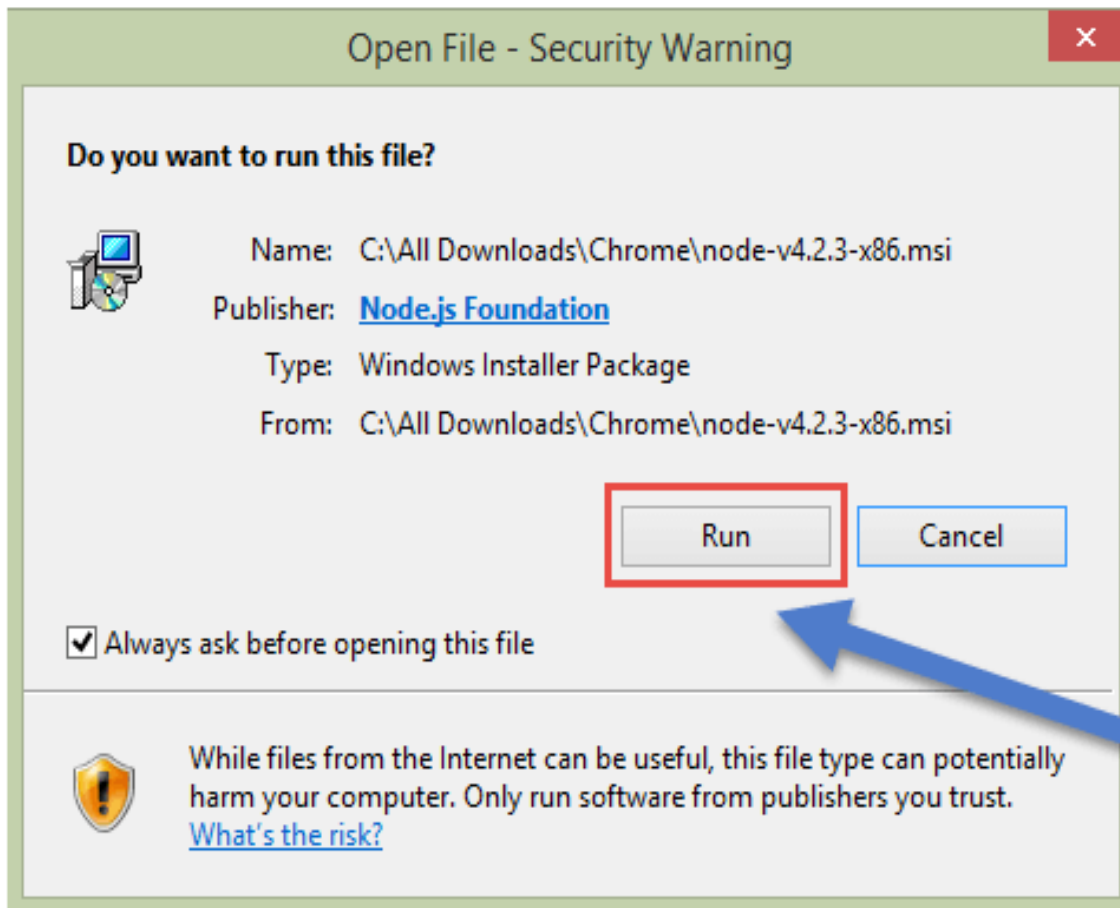

Source Code
node-v4.2.3.tar.gz

Windows Installer (.msi)	32-bit	64-bit
Windows Binary (.exe)	32-bit	64-bit
Mac OS X Installer (.pkg)	64-bit	
Mac OS X Binaries (.tar.gz)	64-bit	

Download the 32-bit installer

Step2:

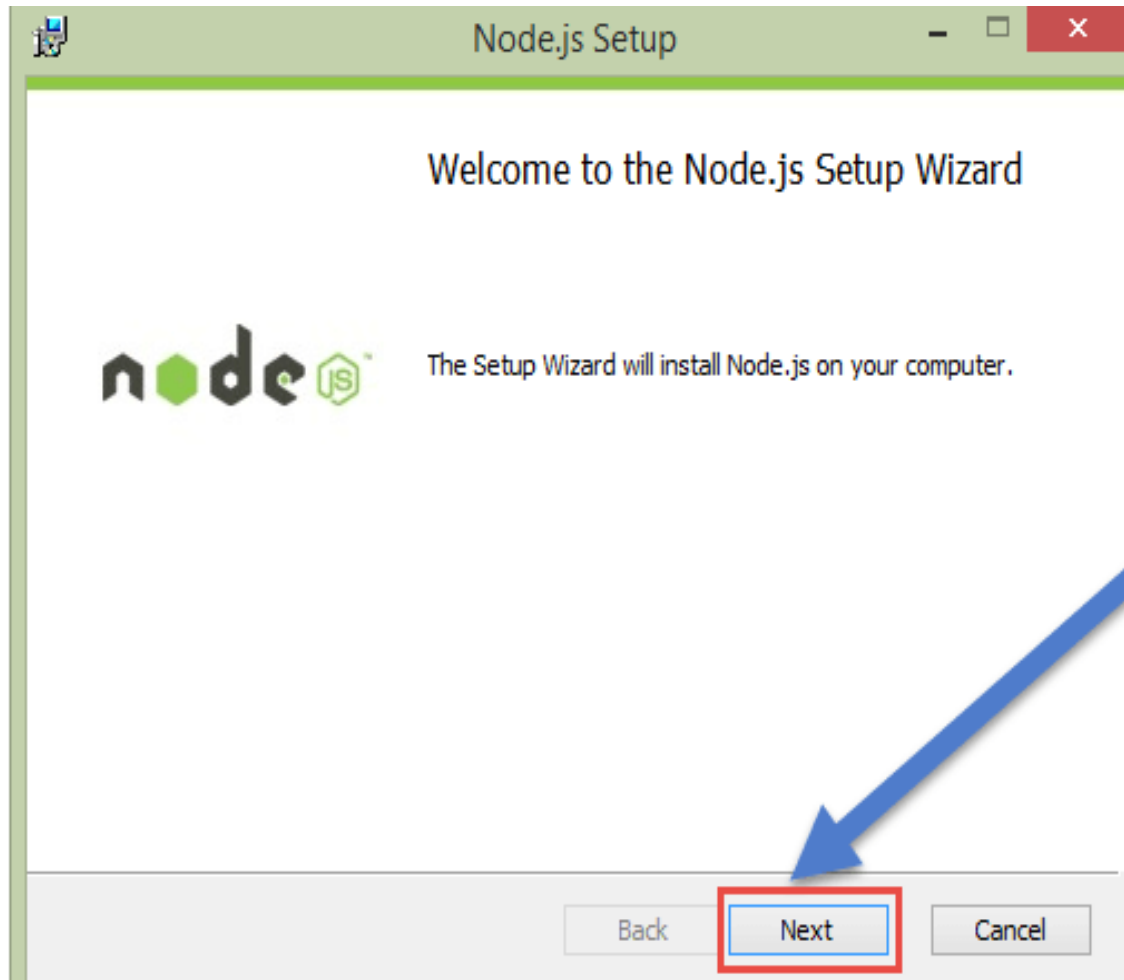
- Double click on the downloaded .msi file to start the installation.
- Click the Run button on the first screen to begin the installation



Click the
Run button

Step3:

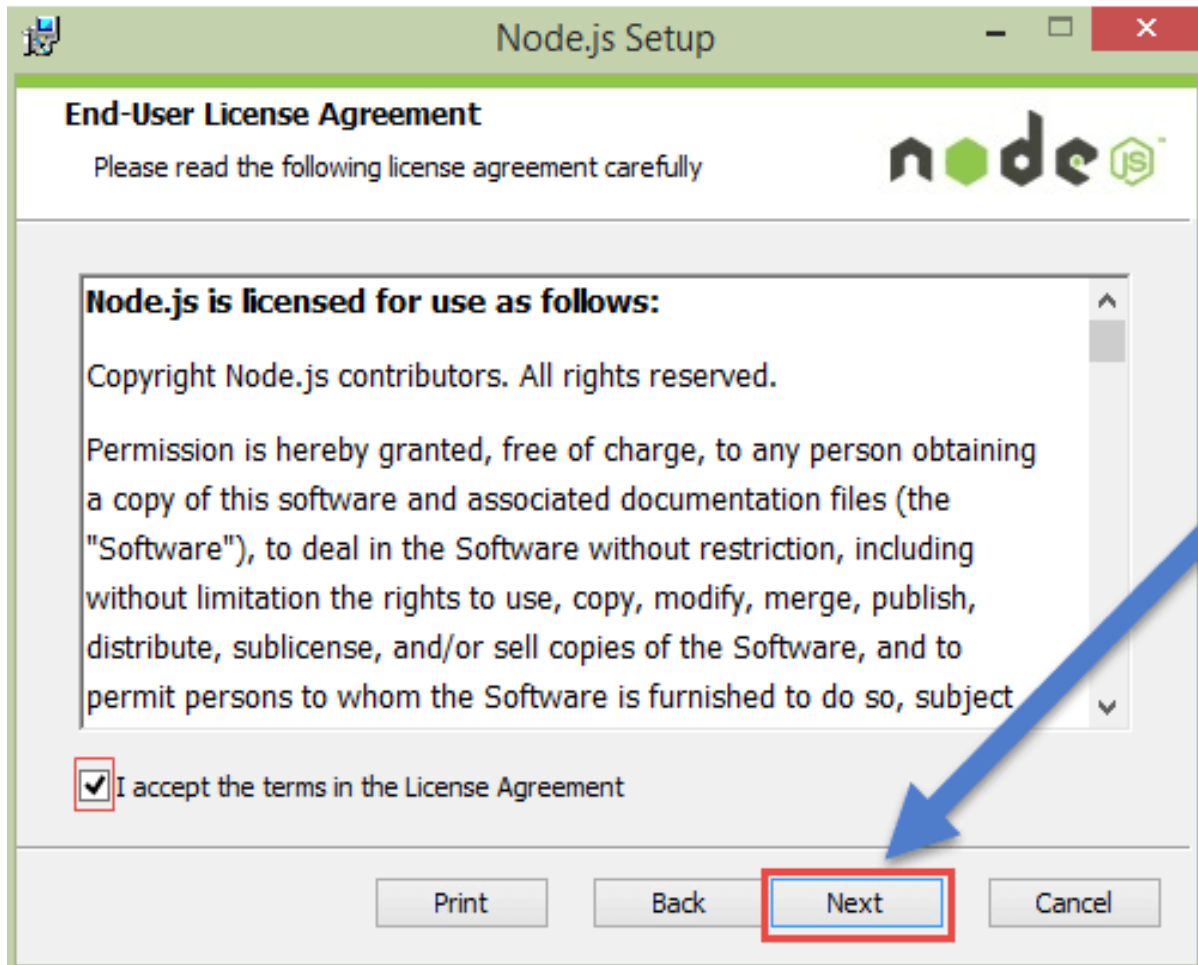
- In the next screen, click the "Next" button to continue with the installation



Click the
Next button

Step4:

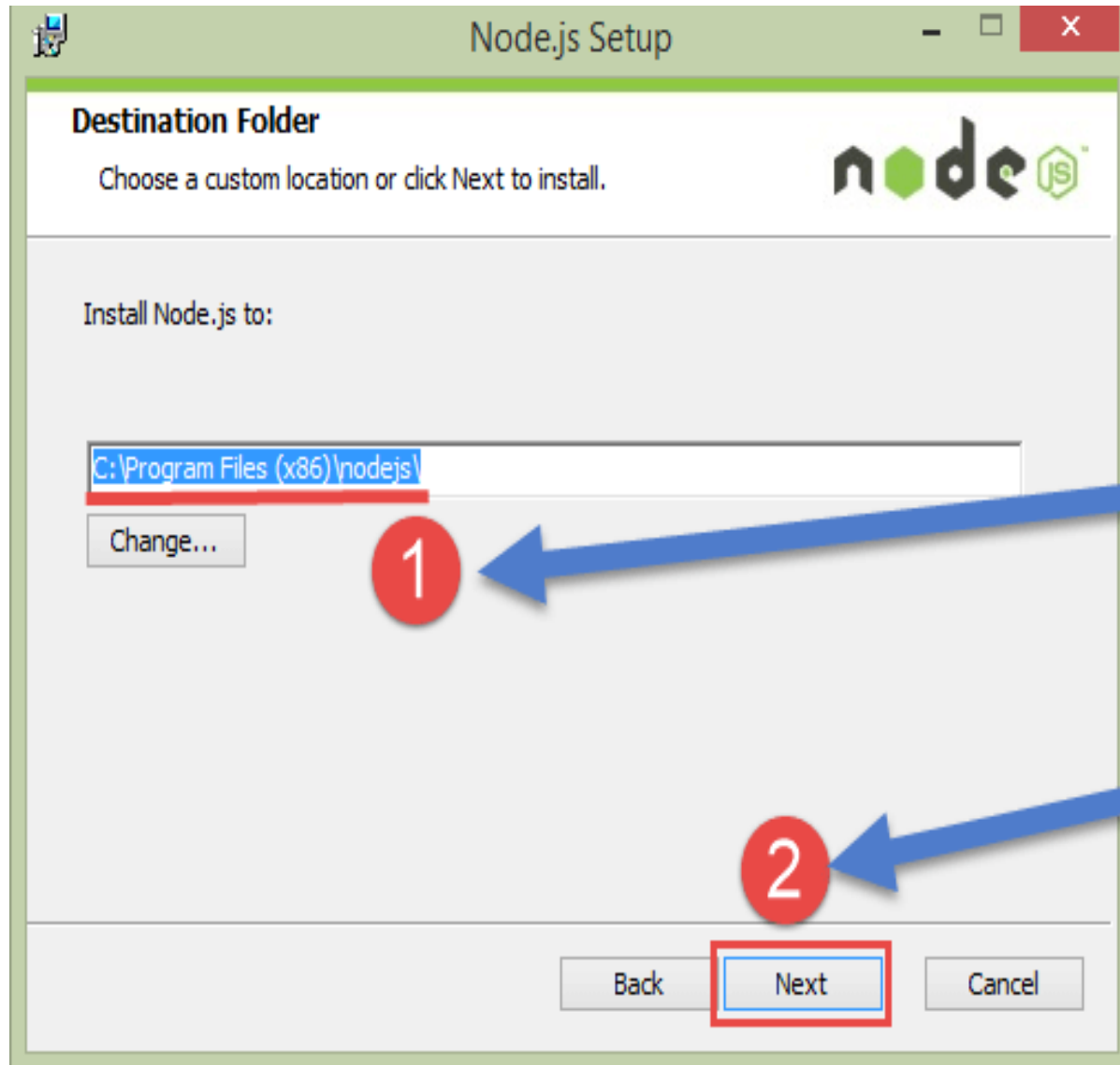
- In the next screen, Accept the license agreement and click on the Next button



Accept the
license
agreement
and click
the Next

Step 5:

- In the next screen, choose the location where Node.js needs to be installed and then click on the Next button.
- 1. First, enter the file location for the installation of Node.js.
- This is where the files for Node.js will be stored after the installation.
- 2. Click on the Next button to proceed ahead with the installation.

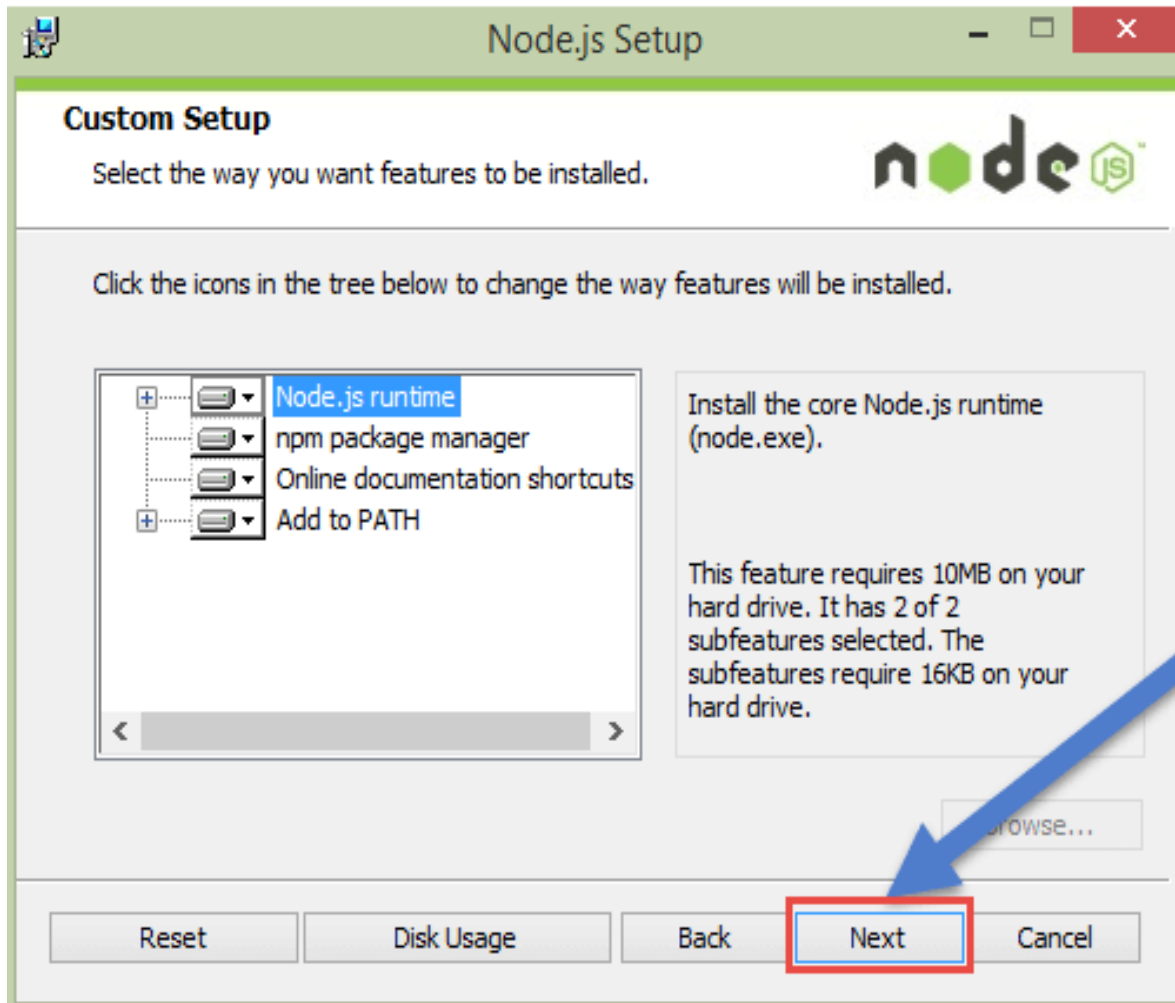


Enter the
file location
for the
installation

Click on the
Next button

Step 6:

- Accept the default components and click on the Next button



Accept the
default
components
and click on
Next

Step7:

- In the next screen, click the Install button to start the installation.



Click the
Next button
to begin
the
installation

Running your first Hello World application in Node.js

```
var http = require('http');  
http.createServer(function (req, res) {  
  res.writeHead(200, {'Content-Type': 'text/html'});  
  res.end('Hello World!');  
}).listen(8080);
```

Code Explanation:

- The basic functionality of the "**require**" function is that it reads a JavaScript file, executes the file, and then proceeds to return an object.
- Using this object, one can then use the various functionalities available in the module called by the require function.
- So in our case, since we want to use the functionality of HTTP and we are using the require(http) command.

- In this 2nd line of code, we are creating a server application which is based on a simple function.
- This function is called, whenever a request is made to our server application.
- When a request is received, we are asking our function to return a "Hello World" response to the client.
- The **writeHead** function is used to send header data to the client, and while the end function will close the connection to the client.
- We are then using the **server.listen** function to make our server application listen to client requests on port no 8080.
- You can specify any available port over which you are using for your application

Executing the code:

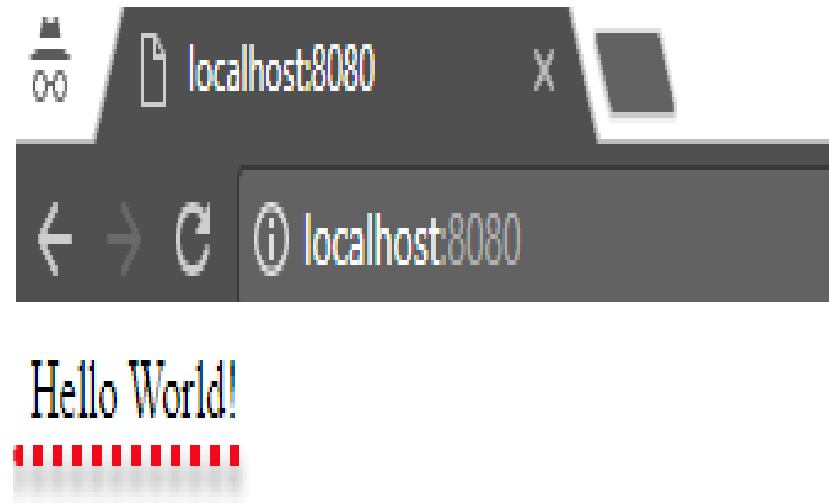
- Save the file on your computer: C:\Users\Your Name\ firstprogram.js
- In the command prompt, navigate to the folder where the file is stored.
- Enter the command Node **first.js**

A screenshot of a Windows Command Prompt window with a black background and white text. The window has a light blue title bar at the top. The text inside shows the following commands and their execution paths:

```
C:\>  
C:\>cd Users  
  
C:\Users>cd welcome  
  
C:\Users\welcome>node first.js
```

- Now, your computer works as a server! If anyone tries to access your computer on port 8080, they will get a "Hello World!" message in return!
- Start your internet browser, and type in the address:
<http://localhost:8080>

Output:



Software Package Manager:

- The name npm (**Node Package Manager**) stems from when npm first was created as a package manager for Node.js.
- All npm packages are defined in files called package.json.
- The content of package.json must be written in JSON.
- At least two fields must be present in the definition file: name and version.

Example

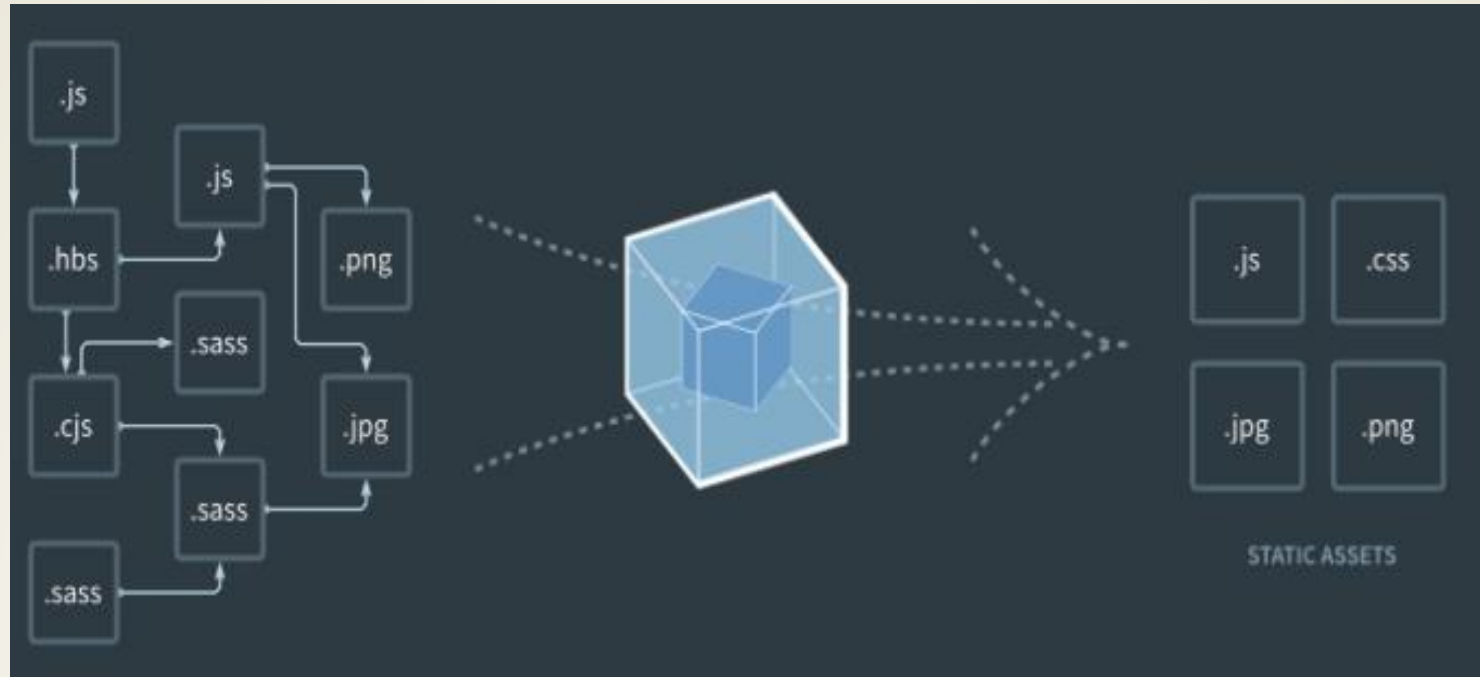
```
{  
  "name" : "foo",  
  "version" : "1.2.3",  
  "description" : "A package for fooing things",  
  "main" : "foo.js",  
  "keywords" : ["foo", "fool", "foolish"],  
  "author" : "John Doe",  
  "licence" : "ISC"  
}
```

Managing Dependencies

- npm can manage dependencies.
- npm can (in one command line) install all the dependencies of a project.
- Dependencies are also defined in **package.json**.

What is Webpack

- Webpack is a static module bundler for JavaScript applications —
- it takes all the code from your application and makes it usable in a web browser.
- Modules are reusable chunks of code built from your app's JavaScript, node_modules, images, and the CSS styles which are packaged to be easily used in your website.
- Webpack separates the code based on how it is used in your app, and with this modular breakdown of responsibilities, it becomes much easier to manage, debug, verify, and test your code.



- When Webpack processes your application, it builds a dependency graph which maps out the modules that your project needs and generates one or more bundles.
- A bundle is a distinct grouping of connected code that has been compiled and transformed for the browser.

- If one file depends on another (it uses the code from a separate file), Webpack treats this as a dependency.
- Webpack also takes your non-code assets (images, fonts, styles, etc.) and converts them to dependencies for your application.

Webpack can be broken down into these 5 principals:

- **Entry**
- **Output**
- **Loaders**
- **Plugins**
- **Mode**

Entry is the entry point for the application.

It is the first module (JavaScript file) that Webpack will process to build out the full dependency graph. 

- It will look at the files that are imported into the entry, and these are added to the dependency graph.
- Webpack figures out which other modules depend on the entry point, both directly and indirectly.
- The default entry point in modern JavaScript applications is **./src/index.js**, but it can be set to any file of your choosing.
- It is also possible to have multiple entry points

```
1 // Single Entry
2 module.exports = {
3   entry: './src/index.js'
4 }
```

```
// Multiple Entry points for Multiple Dependency Graphs
module.exports = {
  entry: {
    pageOne: './src/pageOne/index.js',
    pageTwo: './src/pageTwo/index.js',
    pageThree: './src/pageThree/index.js'
  }
}
```

- Multiple entry points are specifically for multi-page applications.
- With the example above, you are telling Webpack to create three separate dependency graphs. It fetches a new HTML document with new assets.
- This allows for maximizing reusable code throughout the different pages of an application, making it more optimized.
- The Output point is where the files are to be written on disk with the name of the files.
- The main output file is written as `./dist/main.js` and any other files are added to the `dist` directory.
- The output is set at the same location as the entry point. The output can also use hash or chunk names for the content, allow it to be dynamically updated when the code changes.

This ensures you are able to serve the correct code

```
1  module.exports = {  
2    ...  
3    entry: {  
4      pageOne: '.src/index.js',  
5    },  
6    output: {  
7      filename: '[name].js',  
8      pathname: __dirname + '/dist'  
9    }  
10  }  
11  
12  // file on disk: ./dist/index.js
```

- By default, Webpack only knows how to process .js or .json files, which can be limiting.
- With Loaders, Webpack can extend functionality to process other file types by converting them into modules for your application.

- The loaders are specified in the module key under rules.
- There are 2 configuration options required to add a loader — test which identifies file or file types should be transformed and use which tells Webpack which loader to use in order to transform these files.

```
module.exports = {  
  ...  
  output: {  
    filename: 'bundle.js'  
  },  
  module: {  
    rules: [  
      { test: /\.txt$/, use: 'raw-loader'}  
    ]  
  }  
}
```

- Plugins handle the additional tasks that can't be completed by a loader.
- This includes things such as bundle optimization, defining environment variables, etc.
- Another example would be extracting a style sheet or generating an index.html file for a single page web application.

```
module.exports = {  
  ...  
  output: {  
    filename: 'bundle.js'  
  },  
  module: {  
    rules: [  
      { test: /\.txt$/, use: 'raw-loader'  
    ]  
  },  
  plugins: [  
    new HtmlWebpackPlugin({template: './src/index.html'})  
  ]  
}
```

- Mode tells Webpack which configuration and optimizations to use for your application.
- This triggers some mode-specific plugins for Webpack that are built into Webpack, building it for the correct environment.
- The modes are development, production, or none. If mode is not specified, then Webpack automatically defaults to production.

```
module.exports = {  
  mode: 'production'  
};
```

```
// Can be used in CLI webpack --mode = production
```


What is Babel?

- Babel is a JavaScript compiler
- Babel is a toolchain that is mainly used to convert ECMAScript 2015+ code into a backwards compatible version of JavaScript in current and older browsers or environments.
- Here are the main things Babel can do for you:
- **Transform syntax**
- Polyfill features that are missing in your target environment (through @babel/polyfill)
- Source code transformations (codemods)

// **Babel Input: ES2015 arrow function**

```
[1, 2, 3].map((n) => n + 1);
```

// Babel Output: ES5 equivalent

```
[1, 2, 3].map(function(n) {  
  return n + 1;  
});
```

What is JSX

- It is called JSX, and it is a syntax extension to JavaScript.
- We recommend using it with React to describe what the UI should look like.
- JSX may remind you of a template language, but it comes with the full power of JavaScript.
- JSX produces React “elements”

Why JSX?

- React embraces the fact that rendering logic is inherently coupled with other UI logic

- how events are handled, how the state changes over time, and how the data is prepared for display.
- Instead of artificially separating technologies by putting markup and logic in separate files, React separates concerns with loosely coupled units called “components” that contain both.

```
const name = 'Josh Perez';  
const element = <h1>Hello, {name}</h1>;  
  
ReactDOM.render(  
  element,  
  document.getElementById('root')  
);
```

- You can put any valid JavaScript expression inside the curly braces in JSX.
- For example, `2 + 2`, `user.firstName`, or `formatName(user)` are all valid JavaScript expressions.
- In the example below, we embed the result of calling a JavaScript function, `formatName(user)`, into an `<h1>` element.

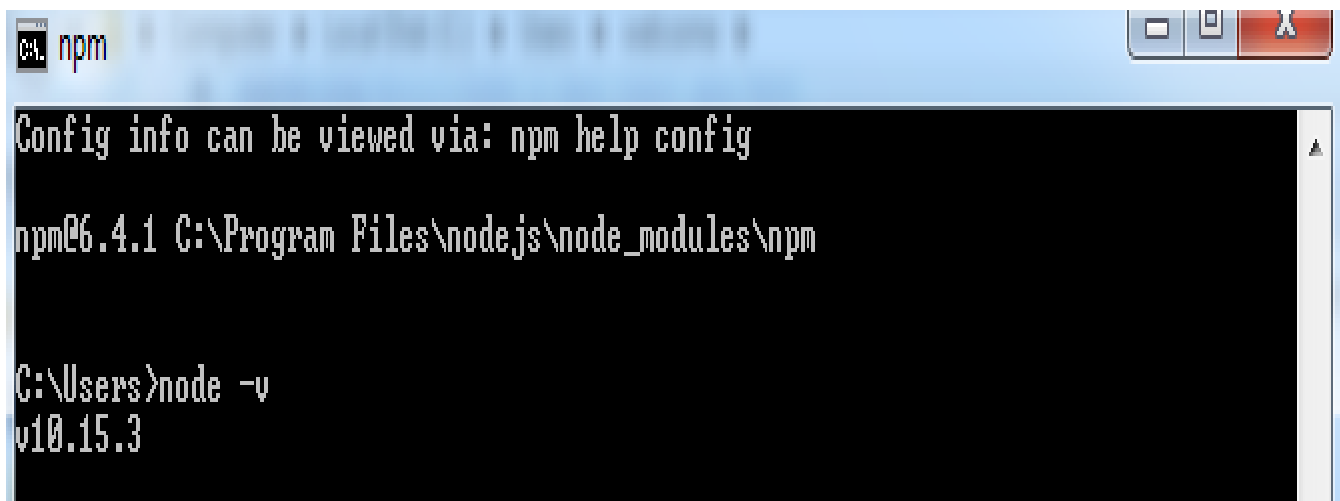
```
function formatName(user) {  
  return user.firstName + ' ' + user.lastName;  
}  
  
const user = {  
  firstName: 'Harper',  
  lastName: 'Perez'  
};  
  
const element = (  
  <h1>  
    Hello, {formatName(user)}!  
  </h1>  
)  
);  
  
ReactDOM.render(  
  element,  
  document.getElementById('root')  
);
```

- The package.json is used for more than dependencies - like defining project properties, description, author & license information, scripts, etc.
- The package-lock.json is solely used to lock dependencies to a specific version number.

Install React-App:

Step1:

- Need to check whether node is installed or not?



```
npm
Config info can be viewed via: npm help config
npm@6.4.1 C:\Program Files\nodejs\node_modules\npm

C:\Users>node -v
v10.15.3
```

Step2:

- Create React App is a comfortable environment for learning React, and is the best way to start building a new single-page application in React.
- It sets up your development environment so that you can use the latest JavaScript features, provides a nice developer experience, and optimizes your app for production. You'll need to have Node ≥ 8.10 and npm ≥ 5.6 on your machine.

```
C:\Users\welcome\workspace>npx create-react-app my-app2
npx: installed 99 in 43.253s

Creating a new React app in C:\Users\welcome\workspace\my-app2.

Installing packages. This might take a couple of minutes.
Installing react, react-dom, and react-scripts with cra-template...

[■.....] / fetchMetadata: sill resolveWithNewModule js-tokens@4.0.0
```

Step3:

- After creating react-app you will be able to see the below image on your terminal

```
Inside that directory, you can run several commands:

  npm start
    Starts the development server.

  npm run build
    Bundles the app into static files for production.

  npm test
    Starts the test runner.

  npm run eject
    Removes this tool and copies build dependencies, configuration files
    and scripts into the app directory. If you do this, you can't go back!

We suggest that you begin by typing:

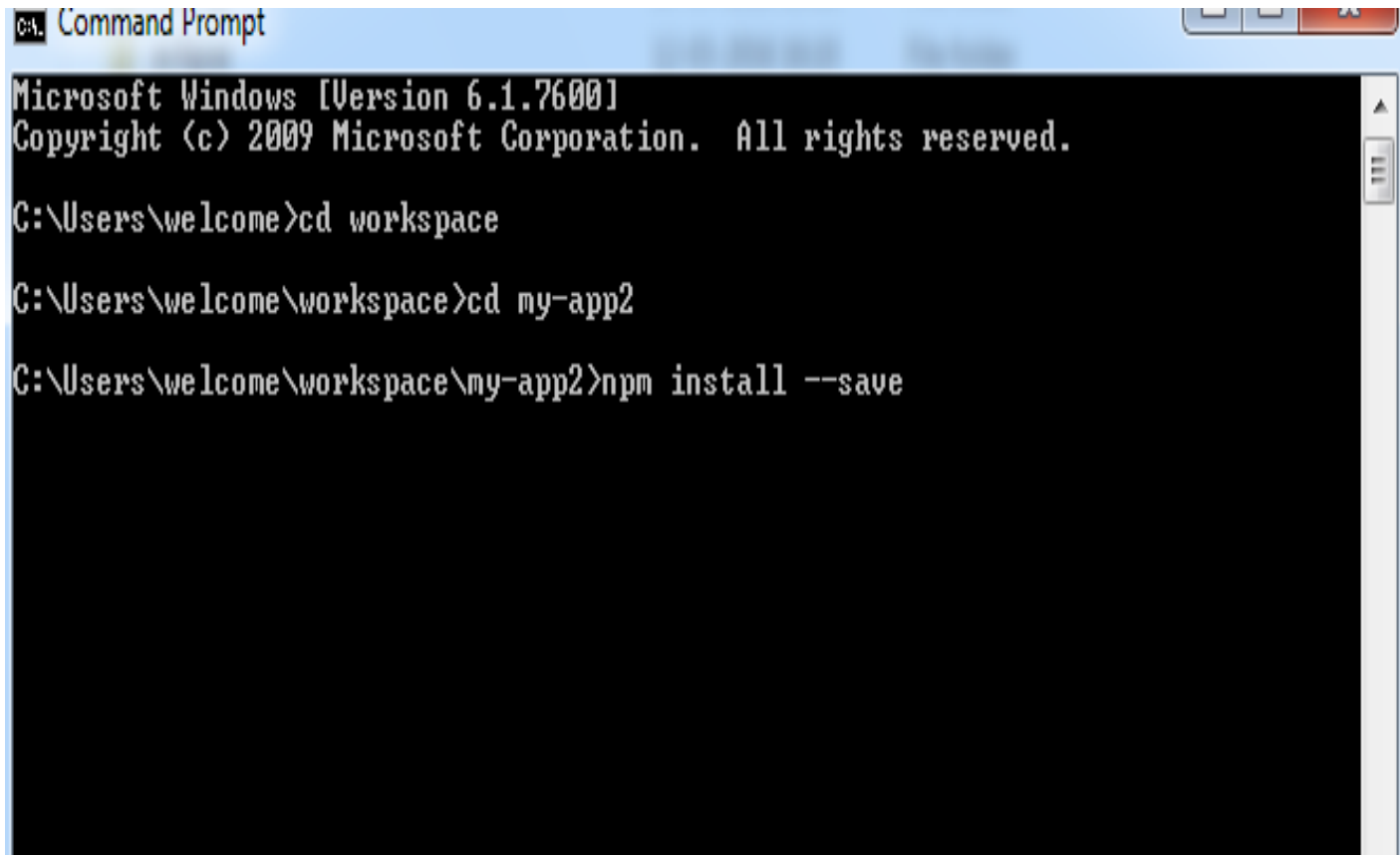
  cd my-app2
  npm start

Happy hacking!

C:\Users\welcome\workspace>
```

Step4:

- After that we need install all the dependencies for that project
- You need to go inside that project folder and need to give **npm install --save**



```
Command Prompt
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\welcome>cd workspace

C:\Users\welcome\workspace>cd my-app2

C:\Users\welcome\workspace\my-app2>npm install --save
```


Step5:

- After installing package.json will get created with all dependencies

```
> core-js-pure@3.6.4 postinstall C:\Users\welcome\workspace\my-app2\node_modules
\core-js-pure
> node -e "try{require('./postinstall')}catch(e){}"

+ cra-template@1.0.3
+ react-scripts@3.4.1
+ react@16.13.1
+ react-dom@16.13.1
added 1604 packages from 750 contributors and audited 931146 packages in 2072.38
1s
found 0 vulnerabilities
```

Hello World with Stateless Functions:

- Stateless components are getting their philosophy from functional programming.

Which implies that: A function returns all time the same thing exactly on what is given to it.

For example:

```
const statelessSum = (a, b) => a + b;
```

```
let a = 0;
```

```
const statefulSum = () => a++;
```

- As you can see from the above example that, statelessSum is always will return the same values given a and b.
- However, statefulSum function will not return the same values given even no parameters

- This type of function's behaviour is also called as a side-effect.
- Since, the component affects somethings beyond.
- So, it is advised to use stateless components more often, since they are side-effect free and will create the samebehaviour always.
- That is what you want to be after in your apps because fluctuating state is the worst casescenario for a maintainable program
- The most basic type of react component is one without state.
- React components that are pure functions of their
- props and do not require any internal state management can be written as simple JavaScript functions.

- These are said to be Stateless Functional Components because they are a function only of props, without having any state to keep track of.
- Here is a simple example to illustrate the concept of a Stateless Functional Component:

// In HTML

<div id="element"></div>

// In React

```
const MyComponent = props => {  
  return <h1>Hello, {props.name}!</h1>;  
};
```

```
ReactDOM.render(<MyComponent name="Arun" />,  
element);
```

// Will render <h1>Hello, Arun!</h1>

- Note that all that this component does is render an h1 element containing the name prop.
- This component doesn't keep track of any state. Here's an ES6 example as well:

```
import React from 'react'  
const HelloWorld = props => (  
  <h1>Hello, {props.name}!</h1>  
)  
HelloWorld.propTypes = {  
  name: React.PropTypes.string.isRequired  
}  
export default HelloWorld
```

- Since these components do not require a backing instance to manage the state, React has more room for optimizations.

Absolute Basics of Creating Reusable Components

Components and Props

- As React concerns itself only with an application's view, the bulk of development in React will be the creation of components.
- A component represents a portion of the view of your application.
- "Props" are simply the attributes used on a JSX node (e.g. `<SomeComponent someProp="some prop's value" />`), and are the primary way our application interacts with our components.
- In the snippet above, inside of `SomeComponent`, we would have access to `this.props`, whose value would be the object `{someProp: "some prop's value"}`.

- It can be useful to think of React components as simple functions - they take input in the form of "props", and produce output as markup.
- Many simple components take this a step further, making themselves "Pure Functions", meaning they do not issue side effects, and are idempotent (given a set of inputs, the component will always produce the same output).
- This goal can be formally enforced by actually creating components as functions, rather
- than "classes".

There are three ways of creating a React component:

Functional ("Stateless") Components

```
const FirstComponent = props => (  
  <div>{props.content}</div>  
);
```

```
React.createClass()  
const SecondComponent = React.createClass({  
  render: function () {  
    return (  
      <div>{this.props.content}</div>  
    );  
  }  
});
```

ES2015 Classes

```
class ThirdComponent extends React.Component {  
  render() {  
    return (  
      <div>{this.props.content}</div>  
    );  
  }  
}
```



```
return (  
  <FirstComponent content={someText} />  
  <SecondComponent content={someText} />  
  <ThirdComponent content={someText} />  
);  
}
```

The above examples will all produce identical markup.

- Functional components cannot have "state" within them.
- So if your component needs to have a state, then go for class based components.
- As a final note, react props are immutable once they have been passed in, meaning they cannot be modified from within a component.

- If the parent of a component changes the value of a prop, React handles replacing the old
- props with the new, the component will rerender itself using the new values

Summary:



What is Reactjs



Advantages of ReactJS



Installation or Setup



How to use NodeJs



Software package Manager



Webpack



Create-React App



Absolute Basics of Creating Reusable Component

Thank You.....

If you have any queries please write to info@uplatz.com".