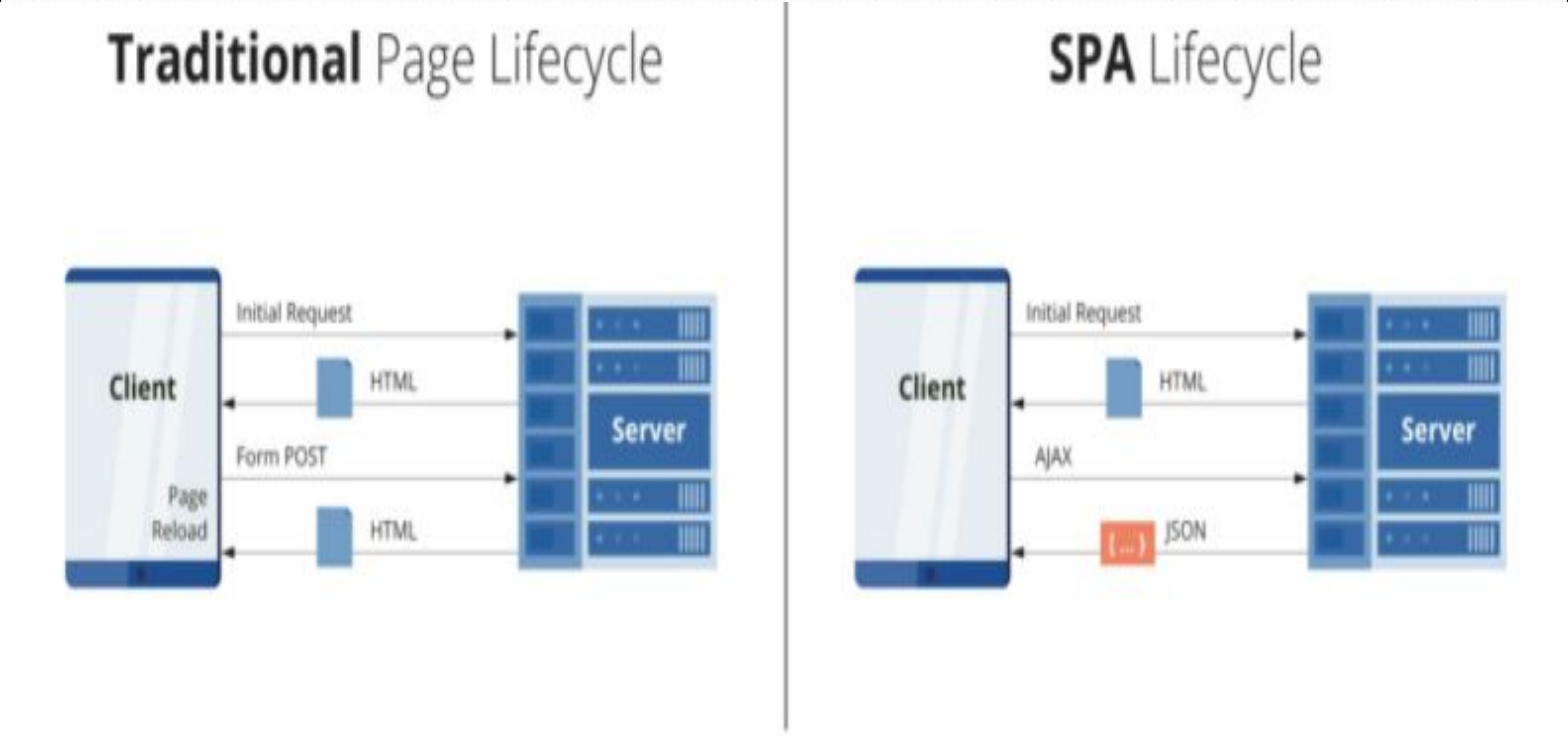
React Fundamentals

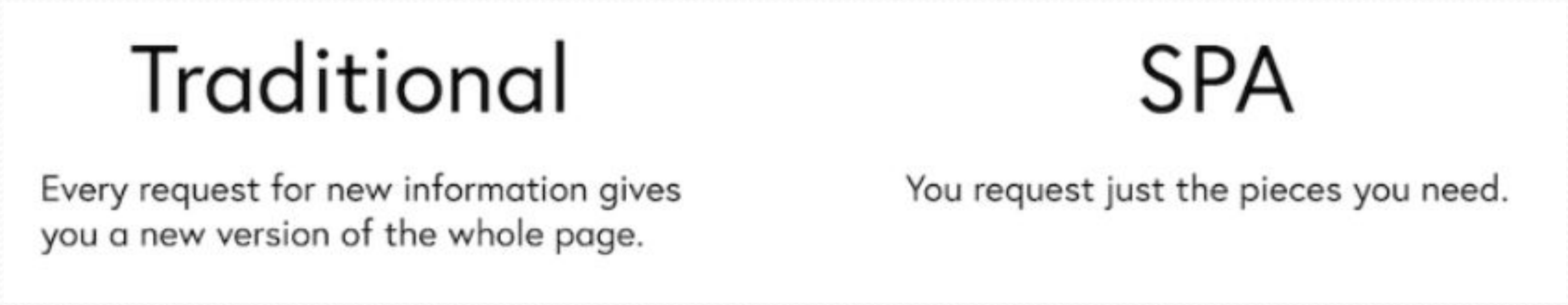
- *React is a JavaScript library for building user interfaces.*
- *React is used to build single-page applications.*
- *React allows us to create reusable UI components.*
- ReactJS is a declarative, efficient, and flexible JavaScript library for building reusable UI components.
- It is an open-source, component-based front end library.
- It was initially developed and maintained by Facebook and later used in its products like WhatsApp & Instagram.

Design Model of Web Application

- Two main design patterns for web apps: multi-page application (MPA) and single-page application (SPA).



- ***There are mainly two design patterns for web applications:***
- ***Single Page Applications***
- ***Multi-Page Applications***
- ***Before moving on to the important differences between Single Page Apps vs. Multi-Page Apps, it is important to understand them independently.***

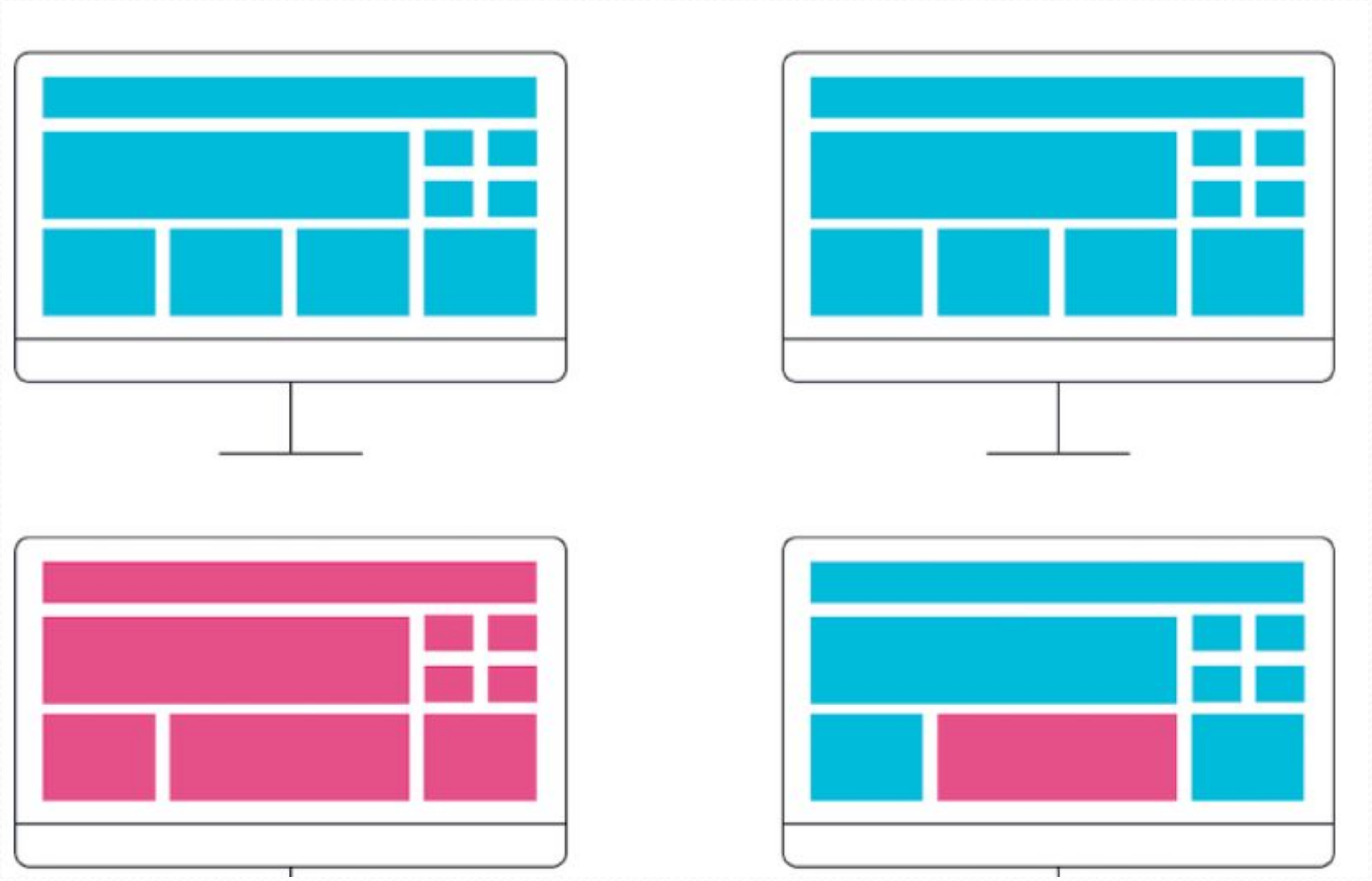


Traditional

Every request for new information gives you a new version of the whole page.

SPA

You request just the pieces you need.



- ***Single-Page Application***
- A single-page application is an app that works inside a browser and does not require page reloading during use.
- You are using this type of applications every day.
- These are, for instance: Gmail, Google Maps, Facebook or GitHub.
- no page reloads, no extra wait time. It is just one web page that you visit which then loads all other content using JavaScript — which they heavily depend on.
- SPA requests the markup and data independently and renders pages straight in the browser.

- Once a single page application is loaded, it can dynamically transfer the content from the server using AJAX requests or WebSocket.
- Data is returned from a server usually in a JSON format, which is processed by SPA.
- This allows the browser to make requests to the server to transfer new or additional content without changing the current web page.
- In addition, HTML5 History API allows us to modify a website’s URL without a full page refresh.

- **Performance & User Experience**
- A Single-page application doesn’t reload the entire page, only required parts of the page.
- It significantly improves application speed and makes seamless user experience.
- As users navigate through web application, most of the resources such as CSS and JavaScript files are downloaded only once, and don’t need to be reloaded during the usage.
- Only data that are transmitted to and from the server changes.
- This decreases the server-client traffic and makes single-page application faster to user requests.

- ***Pros of the Single-Page Application:***
- SPA is fast, as most resources (HTML+CSS+Scripts) are only loaded once throughout the lifespan of application.
- Only data is transmitted back and forth.
- The development is simplified and streamlined.
- There is no need to write code to render pages on the server.
- It is much easier to get started because you can usually kick off development from a file file://URI, without using any server at all.
- SPAs are easy to debug with Chrome, as you can monitor network operations, investigate page elements and data associated with it.
- It's easier to make a mobile application because the developer can reuse the same backend code for web application and native mobile application.
- SPA can cache any local storage effectively.

- ***Cons of the Single-Page Application:***
- It is very tricky and not an easy task to make SEO optimization of a Single-Page Application.
- Its content is loaded by AJAX (Asynchronous JavaScript and XML) — a method of exchanging data and updating in the application without refreshing the page.
- It is slow to download because heavy client frameworks are required to be loaded to the client.
- It requires JavaScript to be present and enabled.
- Compared to the “traditional” application, SPA is less secure.
- Due to Cross-Site Scripting (XSS), it enables attackers to inject client-side scripts into web application by other users.

- ***Multi-Page Application***
- Multiple-page applications work in a “traditional” way.
- Every change eg. display the data or submit data back to server requests rendering a new page from the server in the browser.
- These applications are large, bigger than SPAs because they need to be.
- Due to the amount of content, these applications have many levels of UI.
- On the other hand, it adds more complexity and it is more difficult to develop than a single-page application.

- ***Pros of the Multiple-Page Application:***
- It's the perfect approach for users who need a visual map of where to go in the application.
- Solid, few level menu navigation is an essential part of traditional Multi-Page Application.
- Very good and easy for proper SEO management.
- It gives better chances to rank for different keywords since an application can be optimized for one keyword per page.

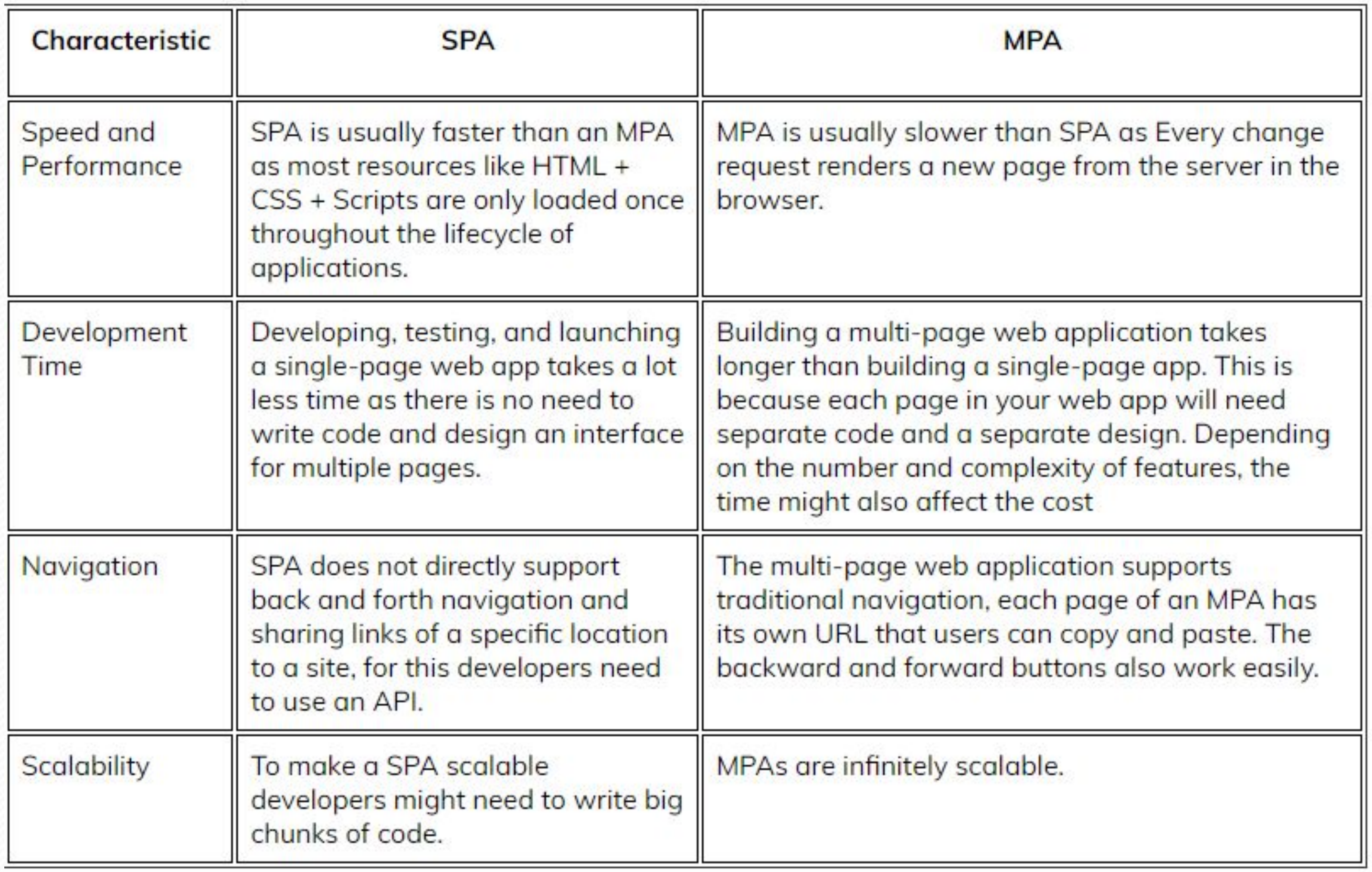
- ***What are Multi-Page Applications?***
- As the name suggests, a multi-page application is an app that has more than one page with static information(text, images, etc.) and links to the other pages with the same content—every change requests rendering a new page from the server in the browser.
- So during a navigation from one page to another, a browser reloads the content of a page completely and downloads the resources again, even if the components are repeated throughout all pages like header and footer.
- MPA generally has a large data and complex architecture. Due to the amount of content, these applications have many levels of UI.
- You can build an MPA simply with HTML and CSS.
- However, many developers also choose to use JavaScript and jQuery to improve the performance of the app.
- **Examples of Multi-page applications are eCommerce websites, blogs, forums, other sites that sell products and various services. eBay and Amazon are the best examples of MPAs.**
- These websites transfer a lot of data from server to client and client to server, hence they are low on speed.

- ***Pros of Multi-Page Application***
- Some of the advantages of Multi-Page application are enlisted below:
- ***Highly Scalable***:
- There is no restriction on the number of pages to add to the existing application. Due to this, when you know that you need to display a lot of information, choosing a Multi-Page application will be a better option
- ***Improved SEO:***
- Multi-page applications are best suited for SEO purpose services or products as it gives better chances to rank for different keywords since an application can be optimized for one keyword per page.
- ***Google Analytics:***
- Google Analytics generates different reports for different pages. With multi-page applications, you can leverage google analytics to draw insights about which pages of your application are performing well and which aren't. Based on these insights, you can make changes to the content of your web application to increase its visibility and average time spent on the pages.

- ***Cons of Multi-Page Application***
- Some of the disadvantages of a multi-page application are enlisted below:
- ***Slow Performance:*** A multi-page application reloads every time a user clicks on a new tab. The resources like HTML, CSS, and JavaScript refresh themselves when there is any action. This affects the speed and performance of web applications.
- ***More Development Time:*** A multi-page application usually has a higher number of features compared to single-page applications so their creation requires more effort and resources.
- ***Hard to Maintain:*** Developers need to maintain each page of a multi-page application separately and regularly. This can get much more tedious than maintaining a single-page application. Moreover, you also need to secure each page separately, which adds to the overall development burden.

MPA and SPA:

- In a multi-page solution, each web page is reloaded every time it receives the corresponding request.
- If you build a SPA, only the necessary content part will be updated.
- It works as follows: when a user enters a one-page app, the server loads the whole page.
- Then, the server transmits the requested data in the form of the JSON (JavaScript Object Notation) files.



Characteristic	SPA	MPA
Speed and Performance	SPA is usually faster than an MPA as most resources like HTML + CSS + Scripts are only loaded once throughout the lifecycle of applications.	MPA is usually slower than SPA as Every change request renders a new page from the server in the browser.
Development Time	Developing, testing, and launching a single-page web app takes a lot less time as there is no need to write code and design an interface for multiple pages.	Building a multi-page web application takes longer than building a single-page app. This is because each page in your web app will need separate code and a separate design. Depending on the number and complexity of features, the time might also affect the cost
Navigation	SPA does not directly support back and forth navigation and sharing links of a specific location to a site, for this developers need to use an API.	The multi-page web application supports traditional navigation, each page of an MPA has its own URL that users can copy and paste. The backward and forward buttons also work easily.
Scalability	To make a SPA scalable developers might need to write big chunks of code.	MPAs are infinitely scalable.

ReactJS:

- ReactJS is a declarative, efficient, and flexible JavaScript library for building reusable UI components.
- It is an open-source, component-based front end library which is responsible only for the view layer of the application.
- It was initially developed and maintained by Facebook and later used in its products like WhatsApp & Instagram.

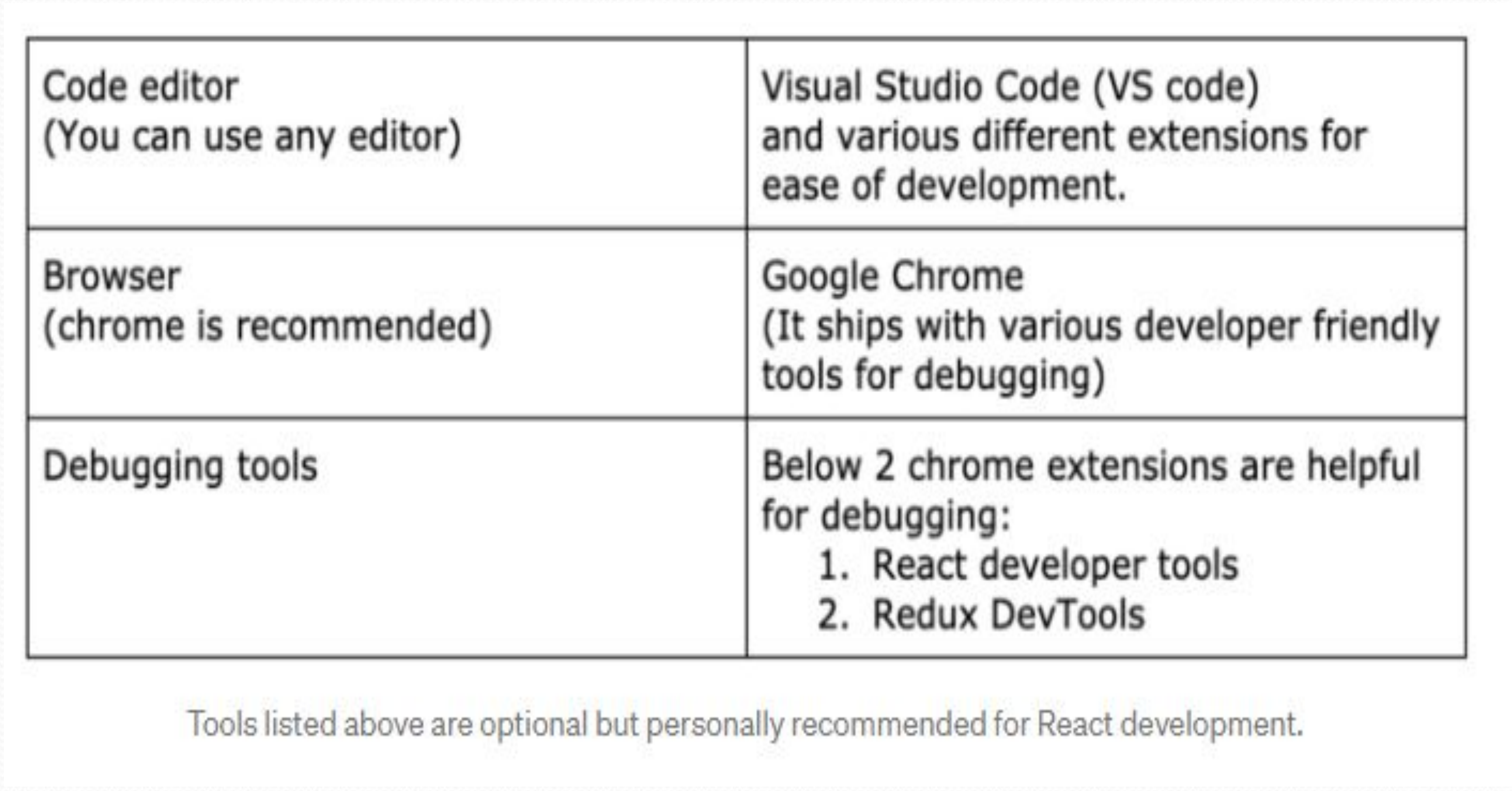
- ***Why we use ReactJS?***
- The main objective of ReactJS is to develop User Interfaces (UI) that improves the speed of the apps.
- It uses virtual DOM (JavaScript object), which improves the performance of the app.
- The JavaScript virtual DOM is faster than the regular DOM.
- We can use ReactJS on the client and server-side as well as with other frameworks.
- It uses component and data patterns that improve readability and helps to maintain larger apps.

- A ReactJS application is made up of multiple components, each component responsible for outputting a small, reusable piece of HTML code.
- The components are the heart of all React applications.
- These Components can be nested with other components to allow complex applications to be built of simple building blocks.
- ReactJS uses virtual DOM based mechanism to fill data in HTML DOM.
- The virtual DOM works fast as it only changes individual DOM elements instead of reloading complete DOM every time.
- To create React app, we write React components that correspond to various elements.
- We organize these components inside higher level components which define the application structure.
- For example, we take a form that consists of many elements like input fields, labels, or buttons.
- We can write each element of the form as React components, and then we combine it into a higher-level component, i.e., the form component itself.
- The form components would specify the structure of the form along with elements inside of it.

- ***Why learn ReactJS?***
- Many JavaScript frameworks are available in the market(like angular, node), but still, React came into the market and gained popularity amongst them.
- The previous frameworks follow the traditional data flow structure, which uses the DOM (Document Object Model).
- DOM is an object which is created by the browser each time a web page is loaded.
- It dynamically adds or removes the data at the back end and when any modifications were done, then each time a new DOM is created for the same page.
- This repeated creation of DOM makes unnecessary memory wastage and reduces the performance of the application.

- Therefore, a new technology ReactJS framework invented which remove this drawback.
- ReactJS allows you to divide your entire application into various components.
- ReactJS still used the same traditional data flow, but it is not directly operating on the browser's Document Object Model (DOM) immediately; instead, it operates on a virtual DOM.
- It means rather than manipulating the document in a browser after changes to our data, it resolves changes on a DOM built and run entirely in memory.
- After the virtual DOM has been updated, React determines what changes made to the actual browser's DOM.
- The React Virtual DOM exists entirely in memory and is a representation of the web browser's DOM.
- Due to this, when we write a React component, we did not write directly to the DOM; instead, we are writing virtual components that react will turn into the DOM.

Section-1 ENVIRONMENT SETUP



Code editor (You can use any editor)	Visual Studio Code (VS code) and various different extensions for ease of development.
Browser (chrome is recommended)	Google Chrome (It ships with various developer friendly tools for debugging)
Debugging tools	Below 2 chrome extensions are helpful for debugging: 1. React developer tools 2. Redux DevTools

Tools listed above are optional but personally recommended for React development.

- ***Requirements***
- ***The Create React App is maintained by Facebook and can works on any platform, for example, macOS, Windows, Linux, etc.***
- ***To create a React Project using create-react-app, you need to have installed the following things in your system.***

- ***Node version >= 8.10***
- ***NPM version >= 5.6***

- Run the following command to check the Node version in the command prompt.

1. $ node -v

-

  $npm -v

- **Install React:**
- We can install React using npm package manager by using the following command.
- **C:\Users\javatpoint> npm install -g create-react-app**
- **Create a new React project**
- Once the React installation is successful, we can create a new React project using create-react-app command.

1. C:\Users\javatpoint> create-react-app reactproject

- **We can combine the above two steps in a single**

1. C:\Users\javatpoint> npx create-react-app reactproject

- Use the following command to add new react project without installing react CLI globally:
- $ npx create-react-app myapp
- $ cd myapp
- $ npm start

- This will create a react js web app in a directory called "myapp" and run the project in the development environment on your default browser.
- It will also watch for the changes and perform hot-reloading for the best development experience.
- React js application that is created via 'create-react-app' ships with some default code.

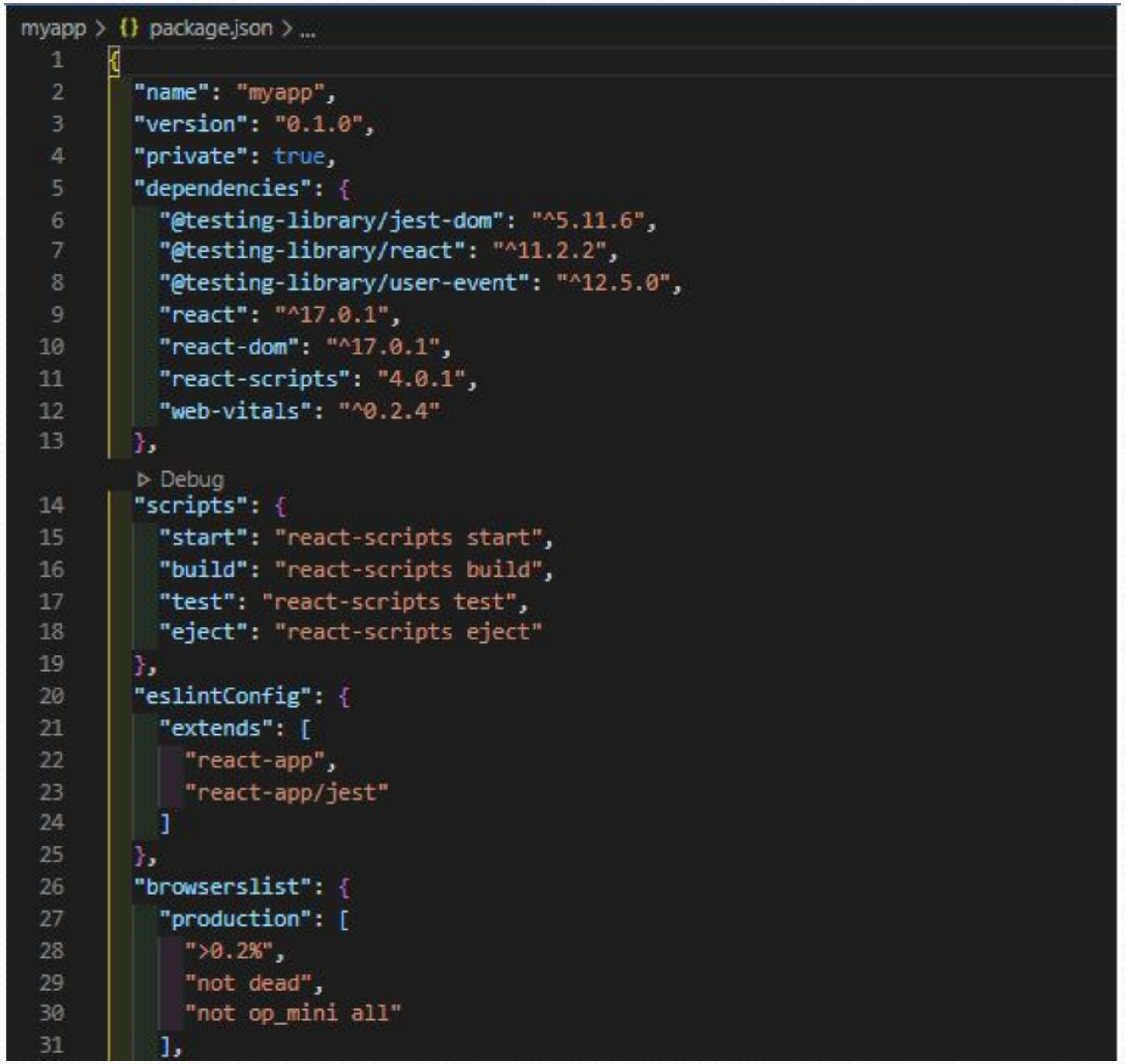
- ***Install node***
- ***Install npm***
- ***Installing globally using CLI***
- npm install -g create-react-app   (install react)
- ***C:\Users\skuma>npm install -g create-react-app***
- ***C:\2021_22\ODD\IP\React>create-react-app reactproject***

- npx create-react-app my-app
- cd my-app
- npm start   ---start server environment
- Building application:
- PS C:\2021_22\ODD\IP\React\my-app> npm run build

- **NPM:** The npm stands for **Node Package Manager** and it is the default package manager for **Node.js.**
- NPX: Npx is a package runner.

FOLDER STRUCTURE:

package.json” is the definition of react js projects.

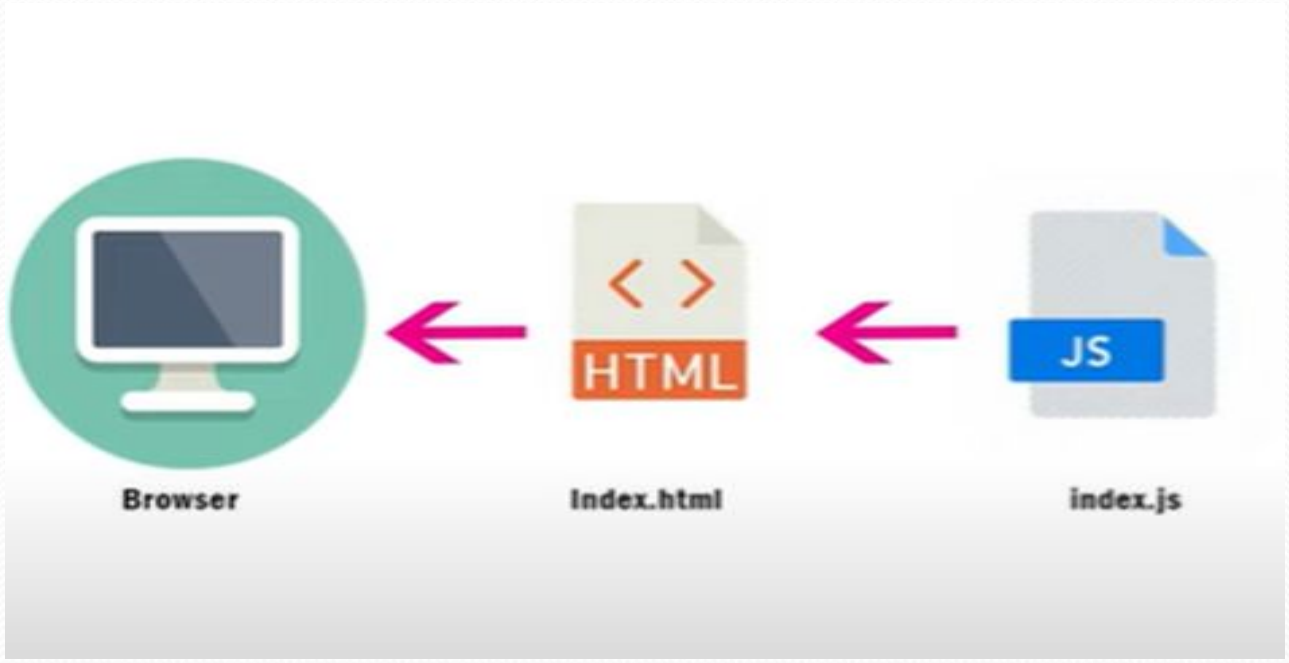


myapp > {} package.json > ...

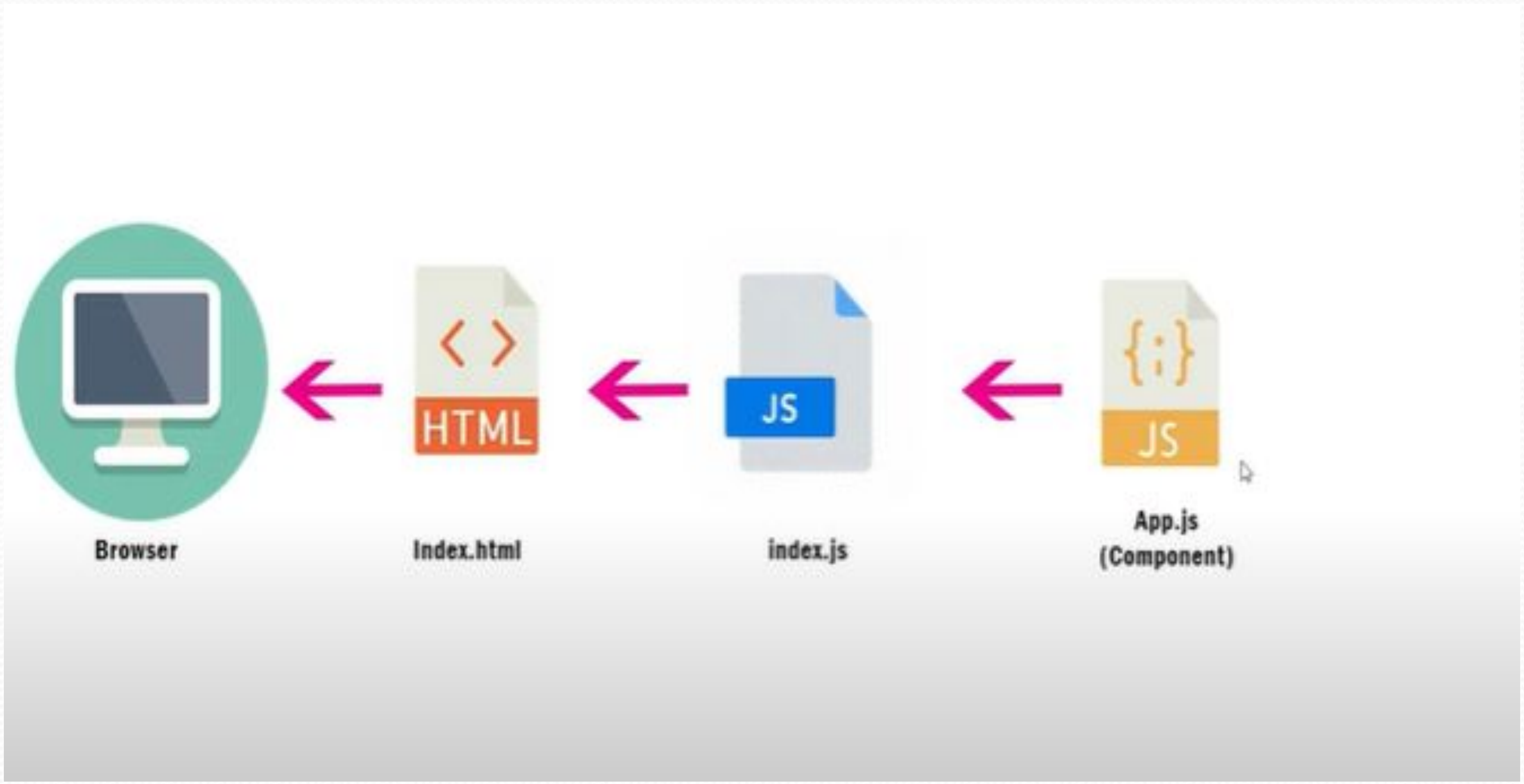
```json
{
  "name": "myapp",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "@testing-library/jest-dom": "^5.11.6",
    "@testing-library/react": "^11.2.2",
    "@testing-library/user-event": "^12.5.0",
    "react": "^17.0.1",
    "react-dom": "^17.0.1",
    "react-scripts": "4.0.1",
    "web-vitals": "^0.2.4"
  },
  ▷ Debug
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  },
  "eslintConfig": {
    "extends": [
      "react-app",
      "react-app/jest"
    ]
  },
  "browserslist": {
    "production": [
      ">0.2%",
      "not dead",
      "not op_mini all"
    ],
```

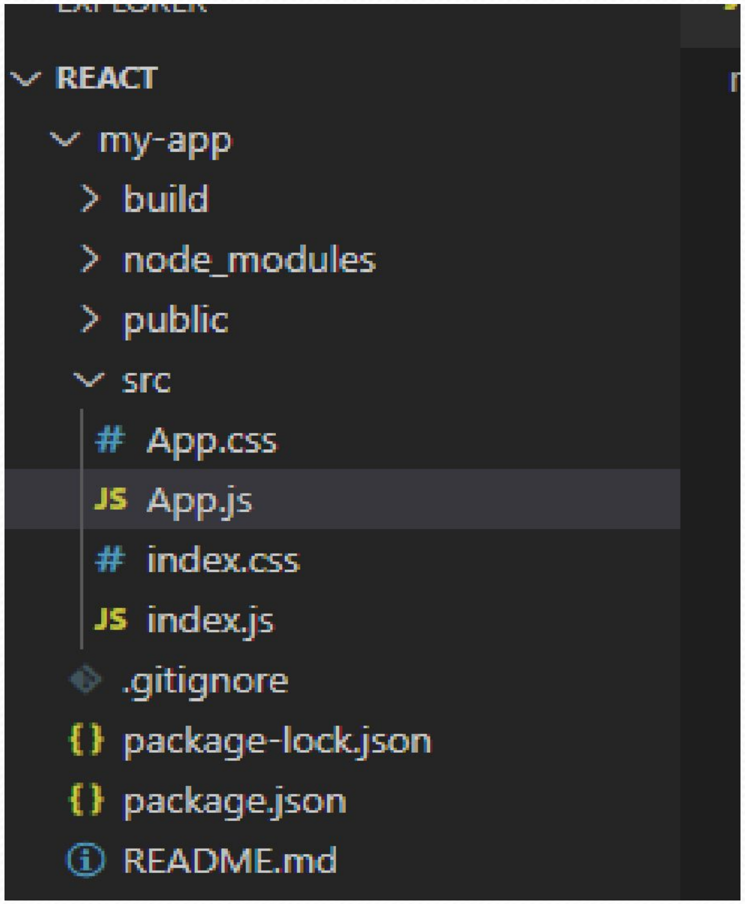
- ***/src/index.js :***
- It is the default entry point of every react application.
- There are no changes in this file at all.
- ***/src/… :***
- All the resources for our application must reside within this directory.
- This will be the core application source code.
- “/src/app.js” file is the container of our app and serves as an entry point.

Flow of React Application Development:



Flow of React Application Development:





```
REACT
  my-app
    build
    node_modules
    public
    src
      App.css
      App.js
      index.css
      index.js
    .gitignore
    package-lock.json
    package.json
    README.md
```

index.js

```
import ReactDom from "react-dom";
import "./index.css";
import App from "./App"
//ReactDom.render(<h1> Hello </h1>, document.getElementById('root’))
 ReactDom.render(<App />,document.getElementById('root’))
```

index.css

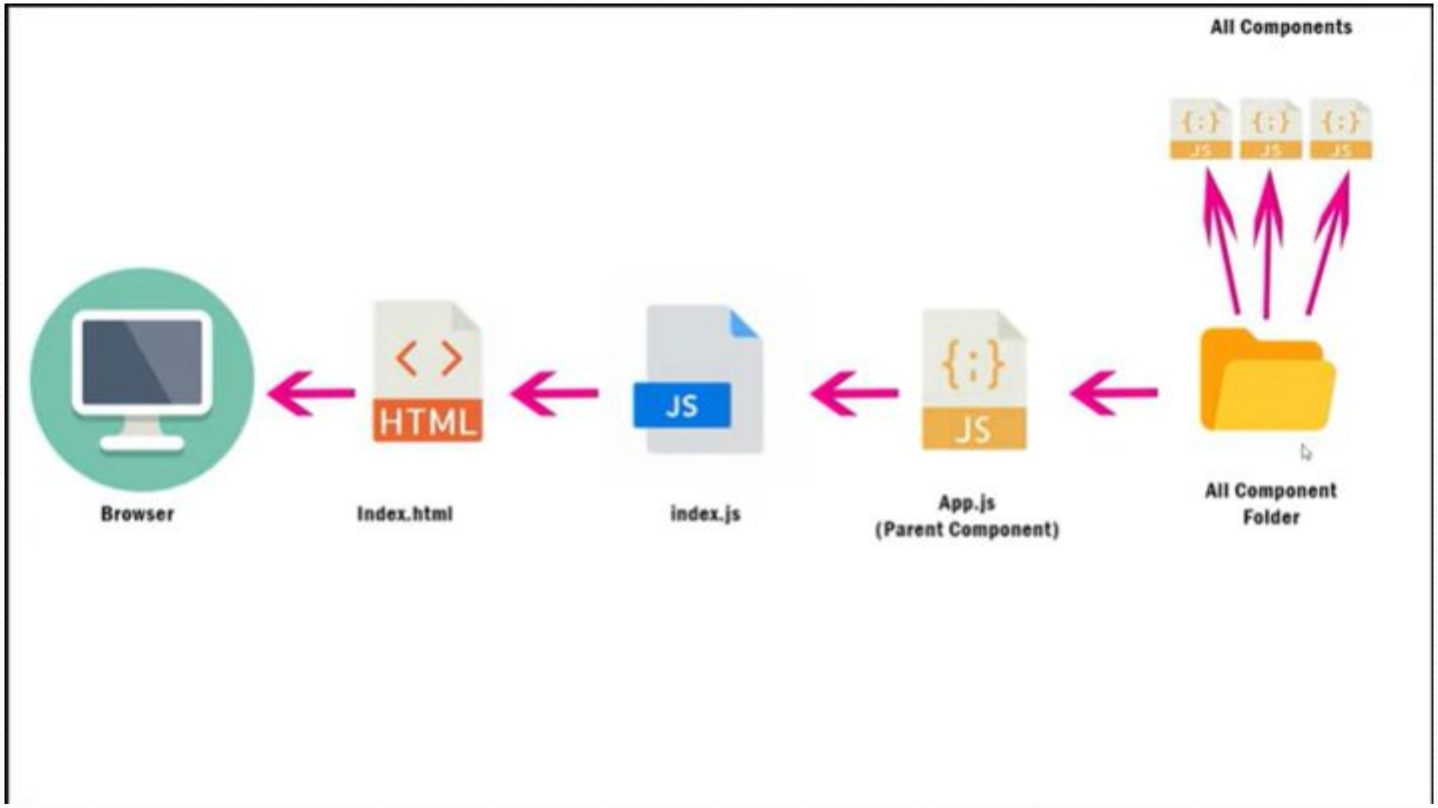
```
body
{
 background-color:aqua;
 color: brown;
  }
```

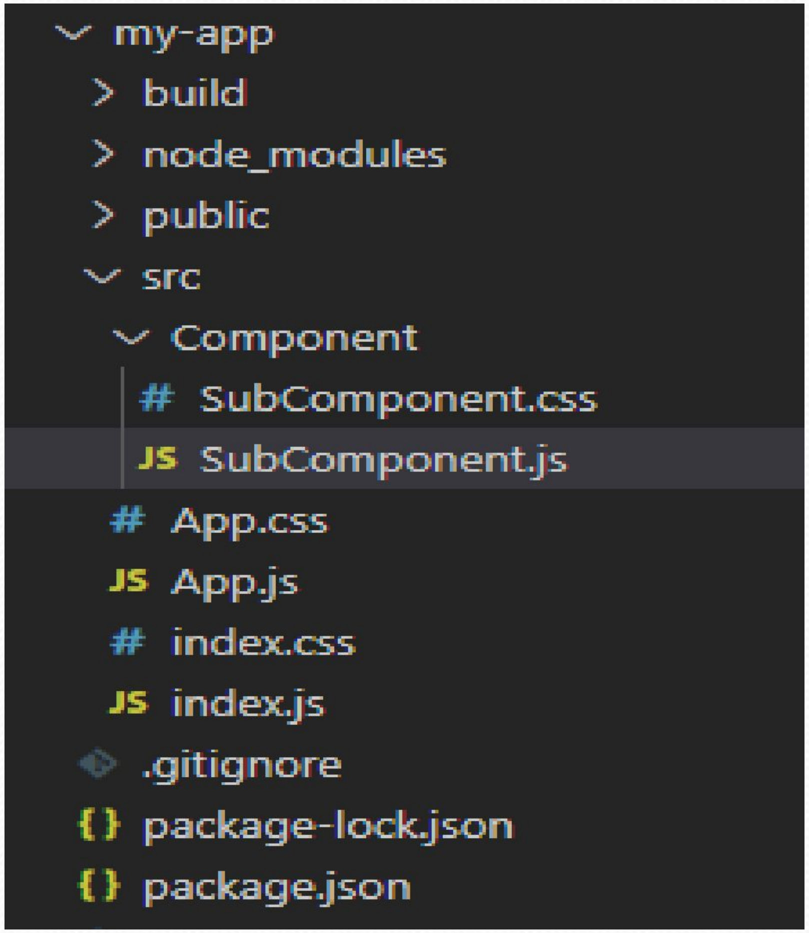
App.js

```
import './App.css'
function App()
{
    return (<div className="app-container">
            <h1>My component</h1>,
            <p>sample component</p>
            </div>
            );
}
export default App;
```

App.css

```
.app-container
{
    background-color:darkgoldenrod;
}
```

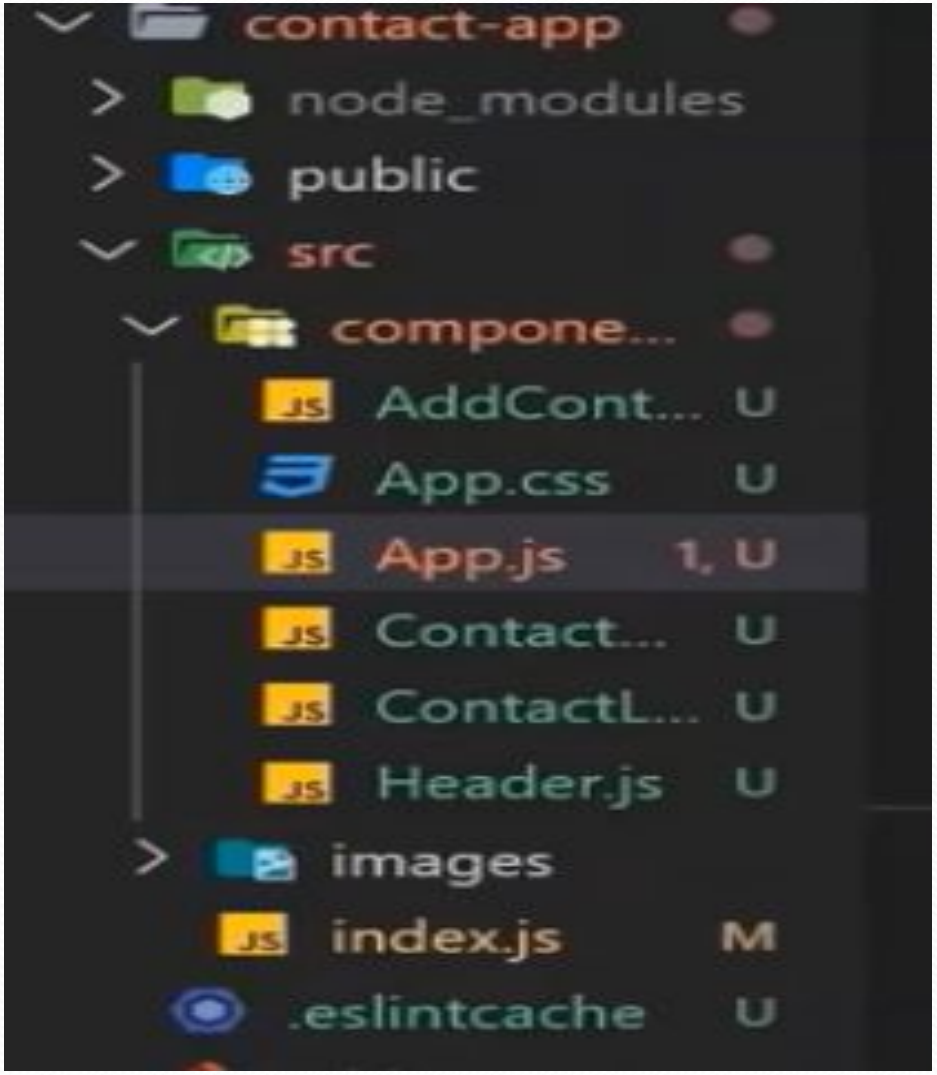

All Components
Browser
Index.html
index.js
App.js
(Parent Component)
All Component
Folder



```
my-app
  build
  node_modules
  public
  src
    Component
      SubComponent.css
      SubComponent.js
    App.css
    App.js
    index.css
    index.js
  .gitignore
  package-lock.json
  package.json
```

- ***Index.js***

```
import './App.css'
import SubComponent from './Component/SubComponent';
function App()
{
   return (<SubComponent/>
           );
}

export default App;
```



```
contact-app
  node_modules
  public
  src
    compone...
      AddCont...      U
      App.css         U
      App.js       1, U
      Contact...      U
      ContactL...     U
      Header.js       U
    images
    index.js          M
  .eslintcache        U
```

- *Installing semantic UI*
- *$npm install semantic-ui-react semantic-ui-css*
- *import 'semantic-ui-css/semantic.min.css'*

- *npm install react-bootstrap@next bootstrap@5.1.1*
- *import 'bootstrap/dist/css/bootstrap.min.css';*

- ***React Environment Setup***
- In this section, we will learn how to set up an environment for the successful development of ReactJS application.

- ***Pre-requisite for ReactJS***
- NodeJS and NPM
- React and React DOM

- ***There are two ways to set up an environment for successful ReactJS application. They are given below.***

- Using the npm command
- Using the create-react-app command
- 1. Using the npm command
- Install NodeJS and NPM

- NodeJS and NPM are the platforms need to develop any ReactJS application.

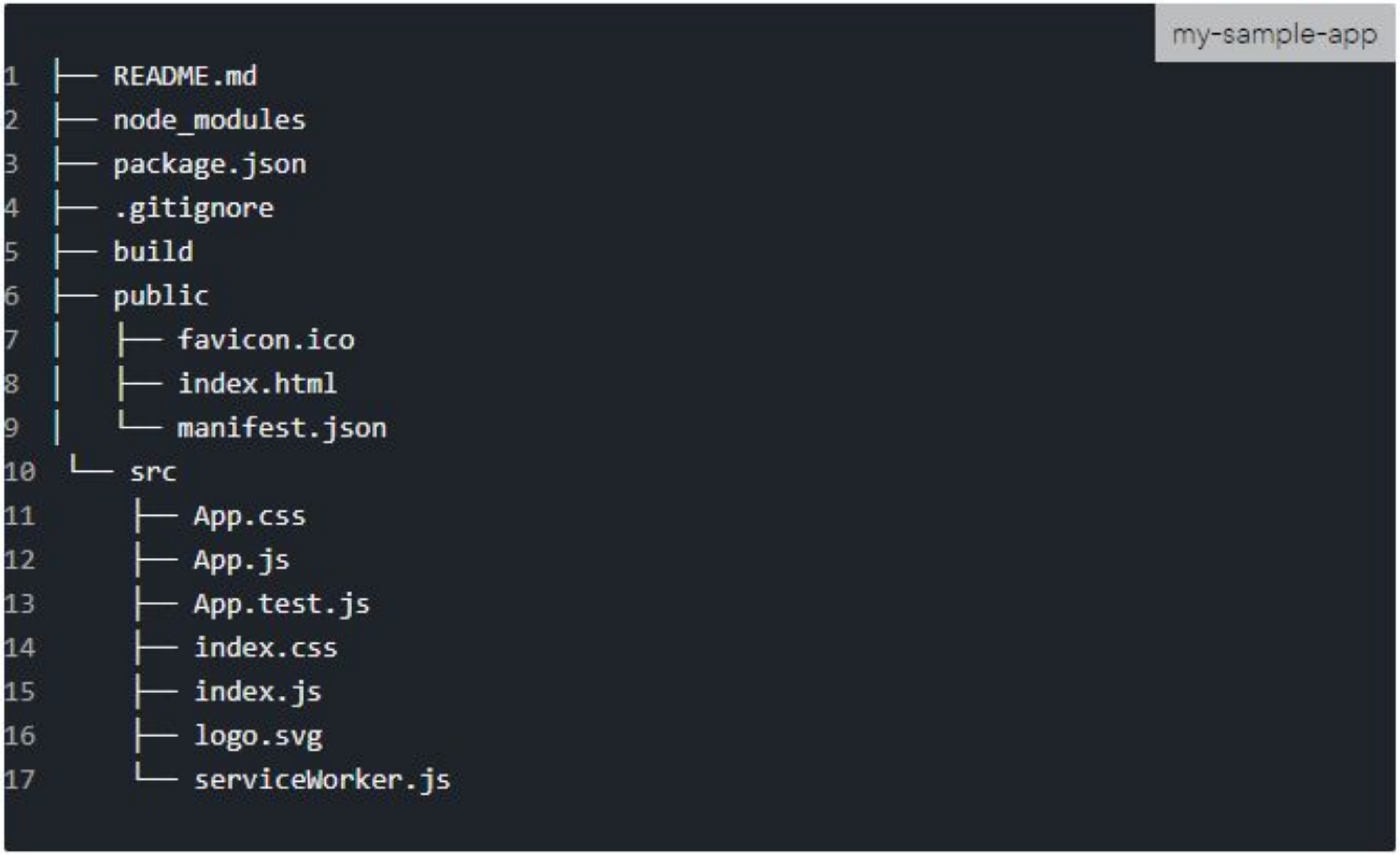
- ***Install node***
- ***Install npm***
- ***Installing globally using CLI***
- npm install -g create-react-app   (install react)
- ***C:\Users\skuma>npm install -g create-react-app***
- ***C:\2021_22\ODD\IP\React>create-react-app reactproject***

- npx create-react-app my-app
- cd my-app
- npm start   ---start server environment
- Building application:
- PS C:\2021_22\ODD\IP\React\my-app> npm run build

- ***Create a New React Project***
- To create a new app/project using this tool, all we need to do is run the command "create-react-app" followed by the app name.

- ***create-react-app my-sample-app***
- After running the above command, a new folder called "my-sample-app" will get created and that would have all of our application code.

- ***Project Layout***
- ***Below is how the project will be structured:***



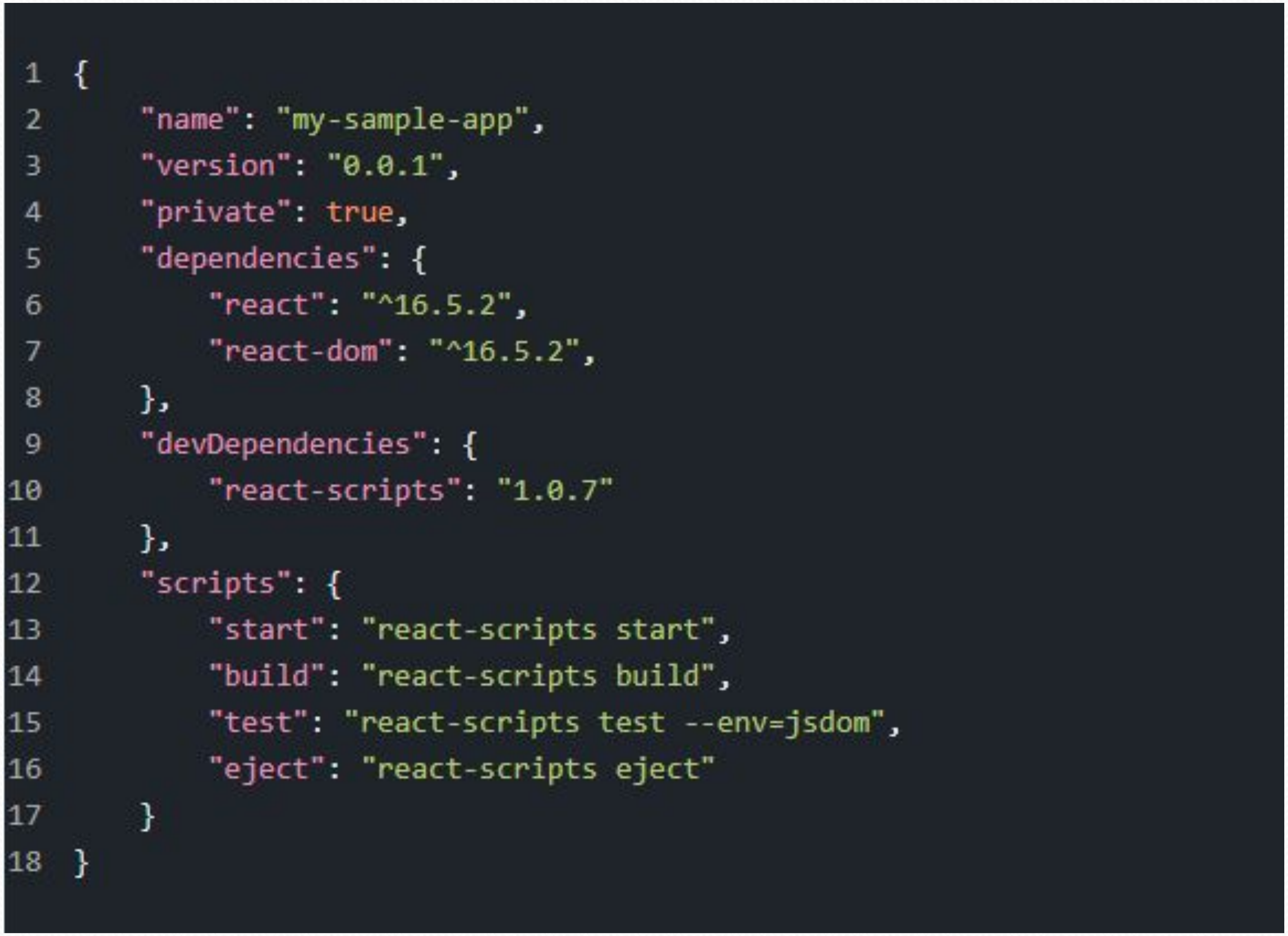
my-sample-app

```
├── README.md
├── node_modules
├── package.json
├── .gitignore
├── build
├── public
│   ├── favicon.ico
│   ├── index.html
│   └── manifest.json
 └── src
     ├── App.css
     ├── App.js
     ├── App.test.js
     ├── index.css
     ├── index.js
     ├── logo.svg
     └── serviceWorker.js
```

- We can see all the "dependencies" and "devDependencies" required by our React app in node_modules.
- These are as specified or seen in our package.json file.
- If we just run the ls -l command, we'll see almost 800 sub-directories.

- Our static files are located in the **public directory.**
- Files in this directory will retain the same name when deployed to production.
- Thus, they can be cached at the client-side and improve the overall download times.
- All of the dynamic components will be located in the src.
- To ensure that, at the client side, only the most recent version is downloaded and not the cached copy, Webpack will generally have the updated files a unique file name in the final build.
- Thus, we can use simple file names e.g. header.png, instead of using header-2019-01-01.png.

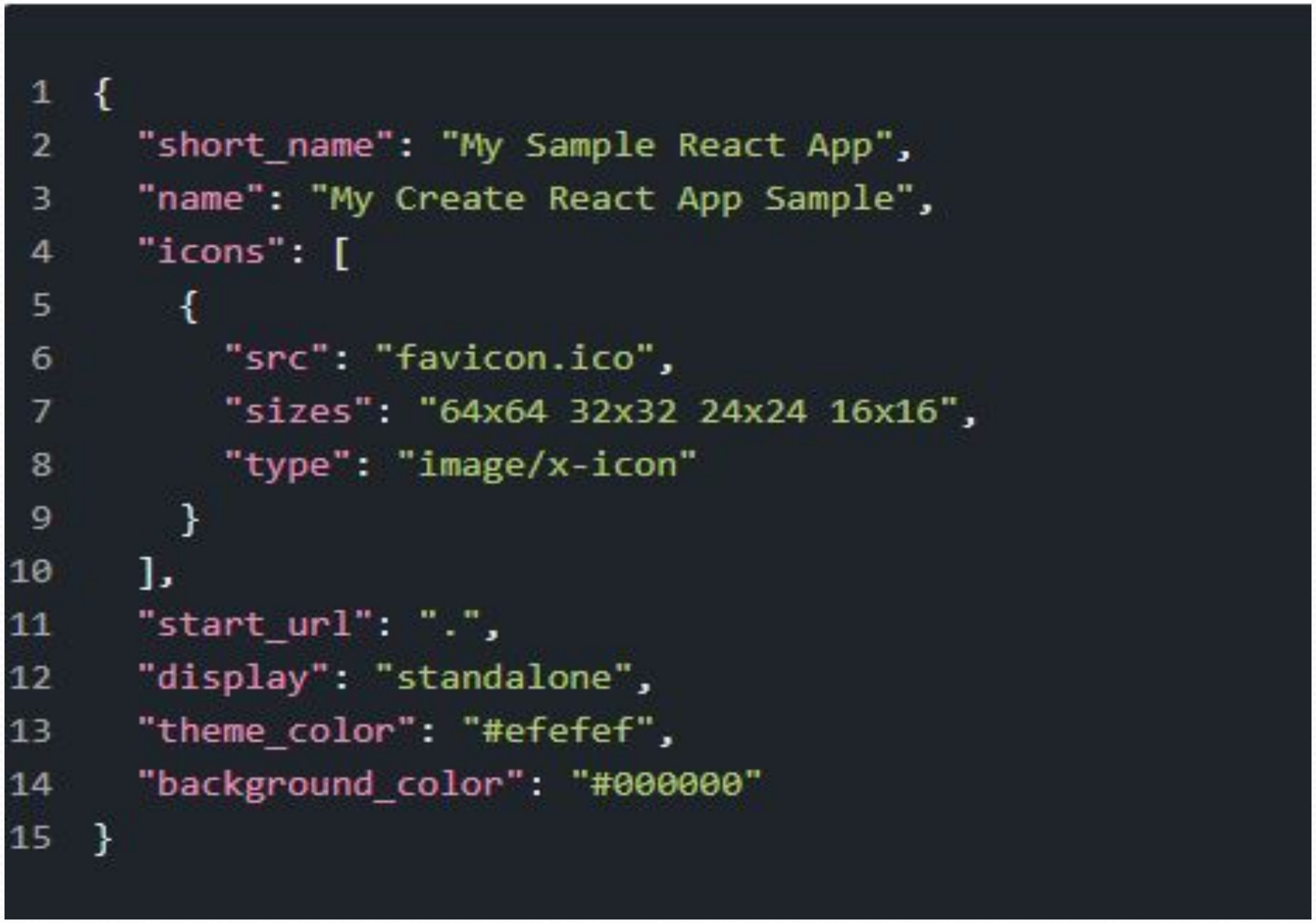
- ***The overall configuration for the React project is outlined in the package.json. Below is what that looks like:***



```json
{
    "name": "my-sample-app",
    "version": "0.0.1",
    "private": true,
    "dependencies": {
        "react": "^16.5.2",
        "react-dom": "^16.5.2",
    },
    "devDependencies": {
        "react-scripts": "1.0.7"
    },
    "scripts": {
        "start": "react-scripts start",
        "build": "react-scripts build",
        "test": "react-scripts test --env=jsdom",
        "eject": "react-scripts eject"
    }
}
```

- name - Represents the app name which was passed to create-react-app.
- version - Shows the current version.
- dependencies - List of all the required modules/versions for our app. By default, npm would install the most recent major version.
- devDependencies - Lists all the modules/versions for running the app in a development environment.
- scripts - List of all the aliases that can be used to access react-scripts commands in an efficient manner. For example, if we run npm build in the command line, it would run "react-scripts build" internally.

- We can also see files like App.js which is kind of our main JS component and the corresponding styles go in App.css.
- In case, we want to add any unit tests, we can use the App.test.js for that.
- Also, index.js is the entry point for our App and it triggers the registerServiceWorker.js.
- As a side-note, we mostly add a 'components' directory here to add new components and their associated files, as that improves the organization of our structure.
- manifest.json This file is used to describe our app e.g. On mobile phones, if a shortcut is added to the home screen. Below is how that would look like;



```json
{
  "short_name": "My Sample React App",
  "name": "My Create React App Sample",
  "icons": [
    {
      "src": "favicon.ico",
      "sizes": "64x64 32x32 24x24 16x16",
      "type": "image/x-icon"
    }
  ],
  "start_url": ".",
  "display": "standalone",
  "theme_color": "#efefef",
  "background_color": "#000000"
}
```

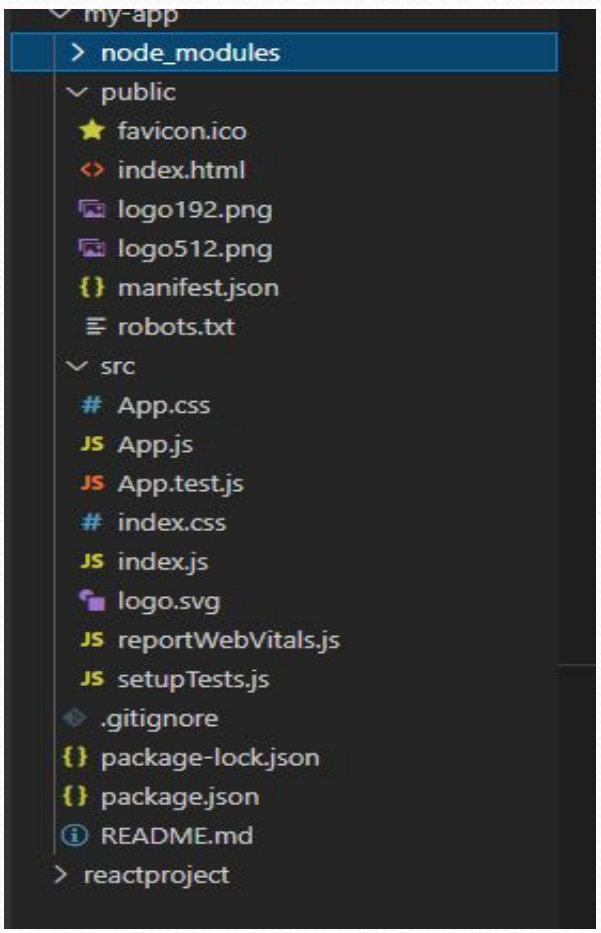
- favicon.ico This is the icon image file used by our project. It is also linked inside index.html and manifest.json.
- ***Component Directory***
- The component directory structure is the most important thing in any React app.
- While components can reside in src/components/my-component-name, it is recommended to have an index.js inside that directory.
- Thus, whenever someone imports the component using src/components/my-component-name, instead of importing the directory, this would actually import the index.js file.

- ***Index Page***
- Let's also have a look inside the index.js as well as the index.html page which gets generated.
- ***Below is how our index.js file looks;***

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import registerServiceWorker from './registerServiceWorker';
ReactDOM.render(<App />, document.getElementById('root'));
registerServiceWorker();
```

- ***Below is the html page; index.html***

```
<!DOCTYPE html>
<html lang="en">
 <head>
   <meta charset="utf-8" />
   <link rel="shortcut icon" href="%PUBLIC_URL%/favicon.ico" />
   <meta name="viewport“  content="width=device-width, initial-scale=1,
    shrink-to-fit=no"/>
   <meta name="theme-color" content="#000000" />
<link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
<title>React App</title>
 </head>
 <body>
   <noscript>You need to enable JavaScript to run this app.</noscript>
   <div id="root"></div>
</body>
</html>
```



```
my-app
  > node_modules
  v public
      favicon.ico
      index.html
      logo192.png
      logo512.png
      manifest.json
      robots.txt
  v src
      App.css
      App.js
      App.test.js
      index.css
      index.js
      logo.svg
      reportWebVitals.js
      setupTests.js
    .gitignore
    package-lock.json
    package.json
    README.md
> reactproject
```

index.html
index.js   starting point of Application

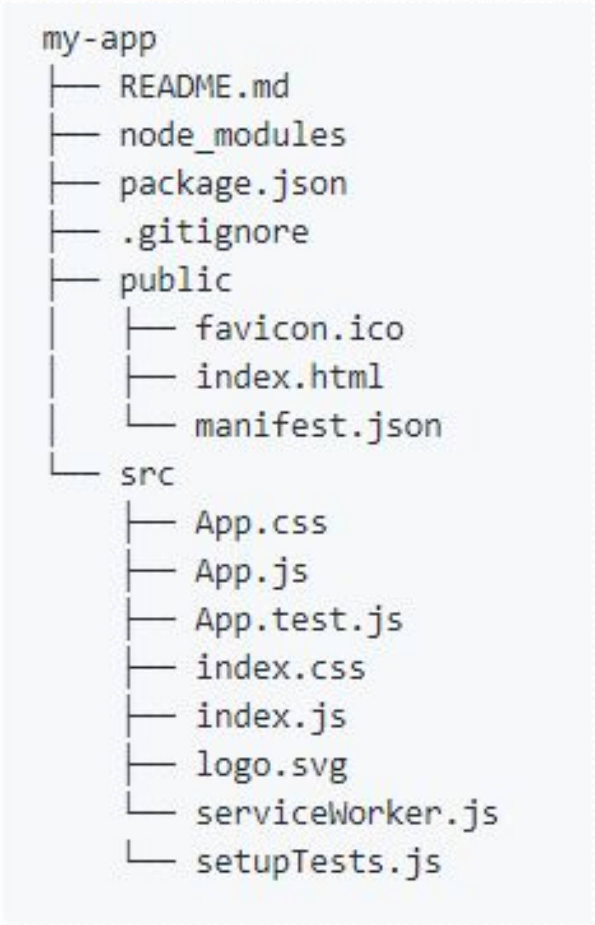
- ***Building React application:***
- Create project folder
- Open in VS code
- start terminal execute following commands
- npx create-react-app my-app
- cd my-app
- npm start

- npx ***is a package runner tool that comes with npm***

- ***npx***
- npx create-react-app my-app
- (npx is a package runner tool that comes with npm 5.2+ and higher, see instructions for older npm versions)

- ***npm***
- ***npm init react-app my-app***

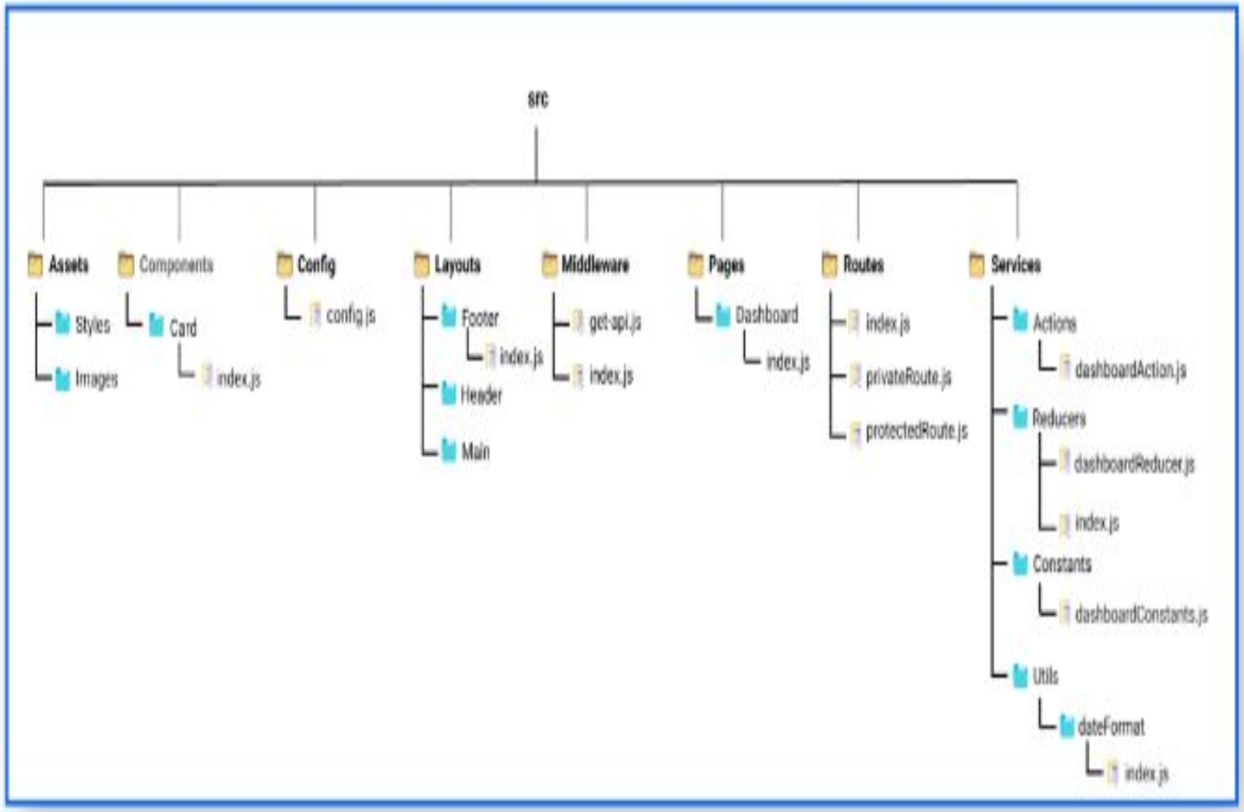
- It will create a directory called my-app inside the current folder.
- Inside that directory, it will generate the initial project structure and install the transitive dependencies:



```
my-app
├── README.md
├── node_modules
├── package.json
├── .gitignore
├── public
│   ├── favicon.ico
│   ├── index.html
│   └── manifest.json
└── src
    ├── App.css
    ├── App.js
    ├── App.test.js
    ├── index.css
    ├── index.js
    ├── logo.svg
    └── serviceWorker.js
    └── setupTests.js
```

- cd my-app
- npm start or yarn start
- npm test or yarn test
- npm test or yarn test
- npm build
- http://localhost:3000/

- **What is React?**
- It is a JavaScript library created by Facebook, a User Interface(UI) library, and a tool for building UI components.
- **Folder Structure for React Project:**
- The folder structure looks like this.
- Assets Folder
- Layouts Folder
- Components Folder
- Pages Folder
- Middleware Folder
- Routes Folder
- Config Folder
- Services Folder
- Utils Folder



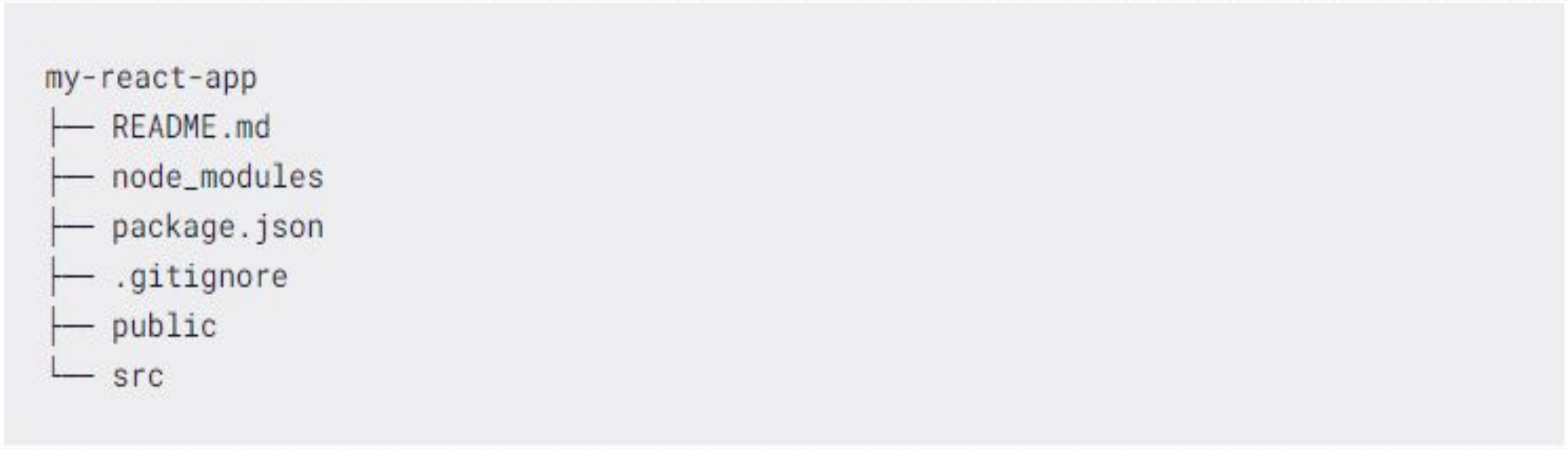
- **Assets Folder**
- As the name says, it contains assets of our project.
- It consists of images and styling files.
- Here we can store our global styles.
- We are centralizing the project so we can store the page-based or component-based styles over here.
- But we can even keep style according to the pages folder or component folder also. But that depends on developer comfortability.
- **Layouts Folder**
- As the name says, it contains layouts available to the whole project like header, footer, etc. We can store the header, footer, or sidebar code here and call it.

- ***Components Folder***
- *Components are the building blocks of any react project.*
- *This folder consists of a collection of UI components like buttons, modals, inputs, loader, etc., that can be used across various files in the project. Each component should consist of a test file to do a unit test as it will be widely used in the project.*
- ***Pages Folder***
- *The files in the pages folder indicate the route of the react application. Each file in this folder contains its route. A page can contain its subfolder. Each page has its state and is usually used to call an async operation. It usually consists of various components grouped.*
- **Middleware Folder**
- This folder consists of middleware that allows for side effects in the application. It is used when we are using redux with it. Here we keep all our custom middleware.

- ***Routes Folder***
- *This folder consists of all routes of the application. It consists of private, protected, and all types of routes. Here we can even call our sub-route.*
- **Config Folder**
- This folder consists of a configuration file where we store environment variables in config.js. We will use this file to set up multi-environment configurations in your application.
- ***Services Folder***
- *This folder will be added if we use redux in your project. Inside it, there are 3 folders named actions, reducers, and constant subfolders to manage states. The actions and reducers will be called in almost all the pages, so create actions, reducers & constants according to pages name.*

- ***Utils Folder***
- *Utils folder consists of some repeatedly used functions that are commonly used in the project. It should contain only common js functions & objects like dropdown options, regex condition, data formatting, etc.*

- ***Step 2. Reviewing the Project Structure***
- ***Once our project files have been created and our dependencies have been installed, our project structure should look like this:***

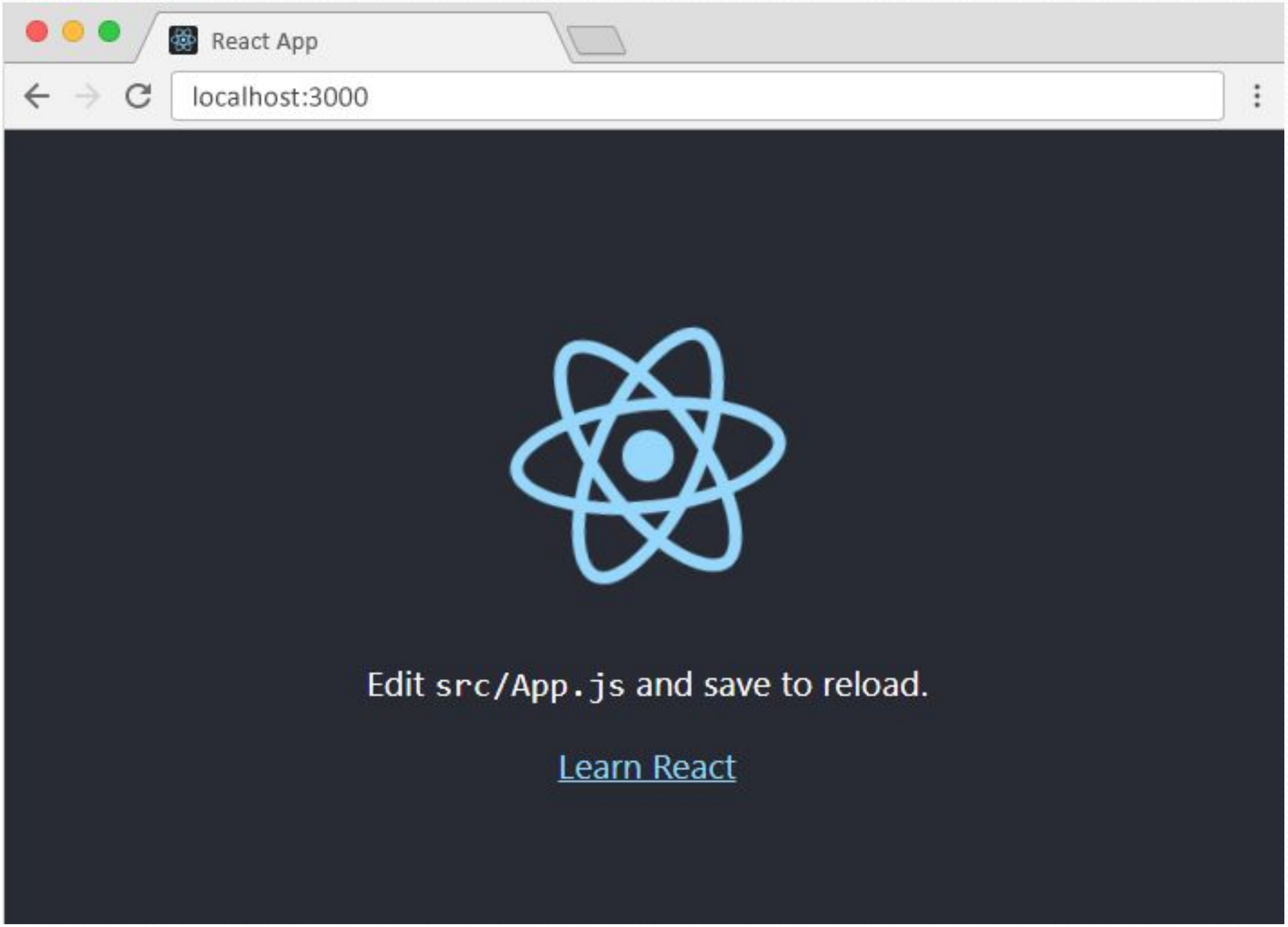


```
my-react-app
├── README.md
├── node_modules
├── package.json
├── .gitignore
├── public
└── src
```

- README.md is a markdown file that includes a lot of helpful tips and links that can help you while learning to use Create React App.
- node_modules is a folder that includes all of the dependency-related code that Create React App has installed. You will never need to go into this folder.
- package.json that manages our app dependencies and what is included in our node_modules folder for our project, plus the scripts we need to run our app.
- .gitignore is a file that is used to exclude files and folders from being tracked by Git. We don't want to include large folders such as the node_modules folder

- public is a folder that we can use to store our static assets, such as images, svgs, and fonts for our React app.
- src is a folder that contains our source code.
- It is where all of our React-related code will live and is what we will primarily work in to build our app.

- ***Create React App***
- *npx create-react-app my-react-app*
- create-react-app will set up everything you need to run a React application.
- Run the React Application
- Run this command to move to the my-react-app directory:
- cd my-react-app
- Run this command to execute the React application my-react-app:
- npm start
- A new browser window will pop up with your newly created React App! If not, open your browser and type localhost:3000 in the address bar.



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

Learn React

- ***/src/App.js:***

```
function App() {
  return (
    <div className="App">
      <h1>Hello World!</h1>
    </div>
  );
}

export default App;
```

index.js:

```
import React from 'react';
import ReactDOM from 'react-dom';
const myfirstelement = <h1>Hello React!</h1>
ReactDOM.render(myfirstelement, document.getElementById('root'));
```

- ***React JSX***
- ***What is JSX?***
- *JSX stands for JavaScript XML.*
- *JSX allows us to write HTML in React.*
- *JSX makes it easier to write and add HTML in React.*

- ***JSX:***
- *const myelement = <h1>JSX code!</h1>;*
- *ReactDOM.render(myelement, document.getElementById('root'));*
- *JSX allows us to write HTML directly within the JavaScript code.*
- *JSX is an extension of the JavaScript language based on ES6, and is translated into regular JavaScript at runtime.*

```
//JSX code
import React from 'react';
function App() {
  return <h1>Hello World</h1>;
}
//JSX code converted to Java Script
import React from 'react';
function App() {
  return React.createElement('h1', null, 'Hello world');
}
```

- ***Expressions in JSX***
- *With JSX you can write expressions inside curly braces { }.*
- *The expression can be a React variable, or property, or any other valid JavaScript expression. JSX will execute the expression and return the result:*
- ***Example***
- *Execute the expression 5 + 5:*
- *const myelement = <h1>React is {5 + 5} times better with JSX</h1>;*

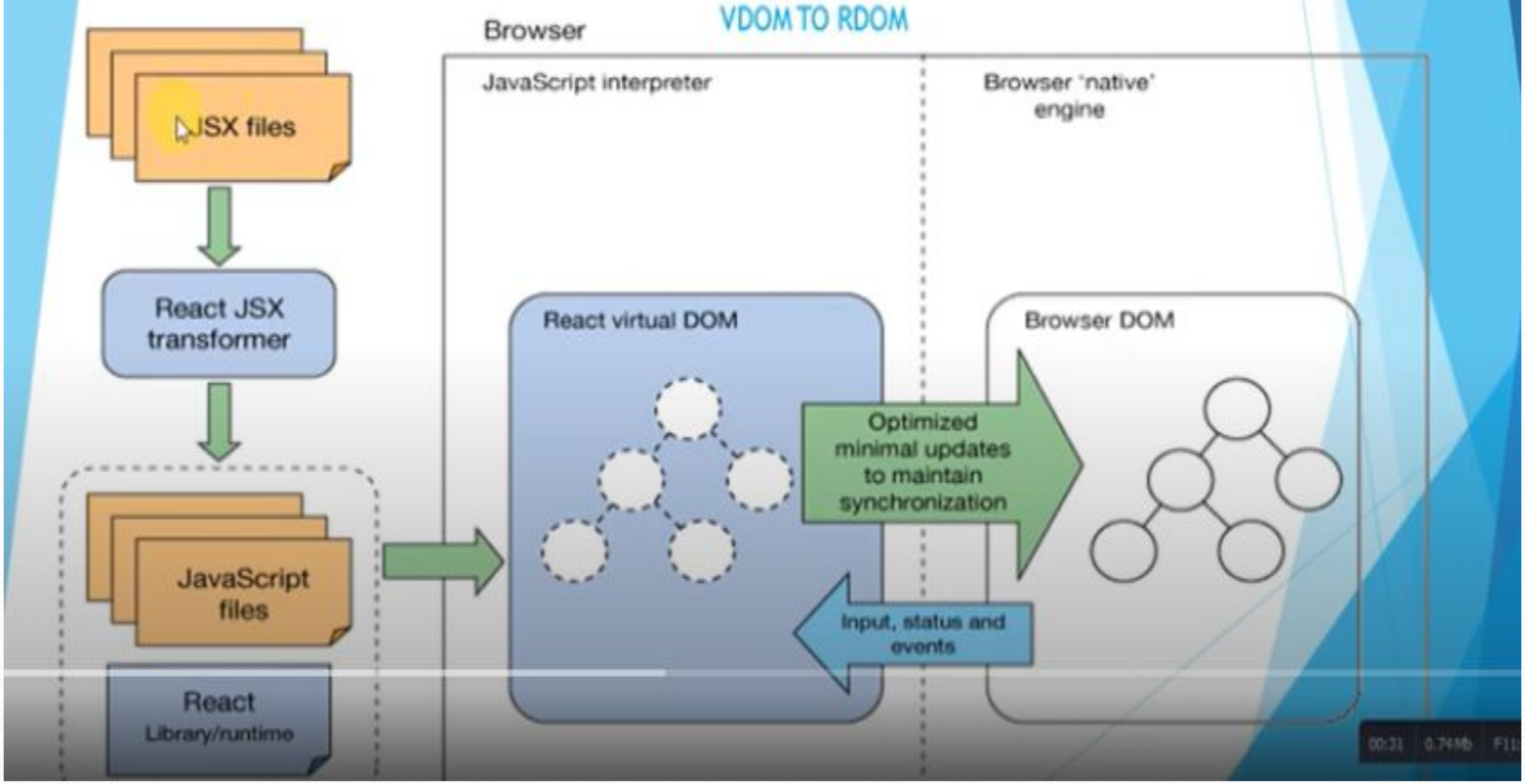
```
const myelement = (
  <ul>
    <li>Apples</li>
    <li>Bananas</li>
    <li>Cherries</li>
  </ul>
);
```

- ***React Render HTML***
- React's goal is in many ways to render HTML in a web page.
- React renders HTML to the web page by using a function called ReactDOM.render().

- ***The Render Function***
- The ReactDOM.render() function takes two arguments, HTML code and an HTML element.
- The purpose of the function is to display the specified HTML code inside the specified HTML element.

- ***But render where?***
- There is another folder in the root directory of your React project, named "public". In this folder, there is an index.html file.
- The HTML code in this tutorial uses JSX which allows you to write HTML tags inside the JavaScript code.

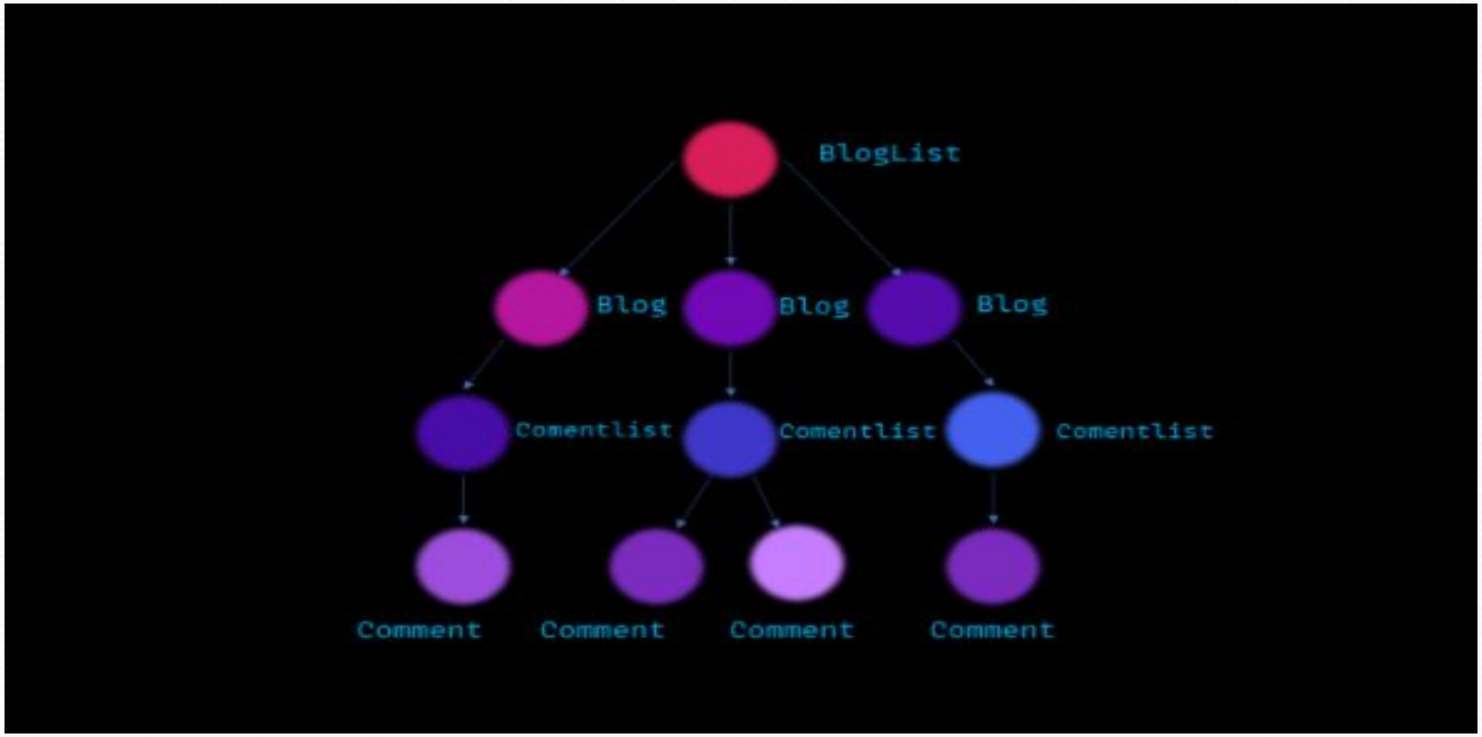
React JS Architecture



- When you use JSX, the compiler transforms it into React function calls that the browser can understand.

```
//JSX code
import React from 'react';
function App() {
  return <h1>Hello World</h1>;
}
//JSX code converted to Java Script
import React from 'react';
function App() {
  return React.createElement('h1', null, 'Hello world');
}
```

- ***What is DOM?***
- DOM stands for Document Object Model. It is the hierarchical representation of your web page(UI).
- For Example you have a blog website, so the hierarchical representation of the website would be as follows.



This is how the browser renders your web page. This is called DOM.

- Because DOM manipulation is very slower.
- Consider the blogging website example and in any of the blog post if a user modifies a comment then the whole DOM (UI) needs to be repainted because of that one little change.
- This change is very expensive in terms of time complexity.

- ***DOM Working:***
- *document.getElementById('some-id').innerValue = 'updated value';*

- *The browser parses the HTML to find the node with this id.*
- *It removes the child element of this specific element.*
- *Updates the element(DOM) with the ‘updated value’.*
- *Recalculates the CSS for the parent and child nodes.*
- *Update the layout.*
- *Finally, traverse the tree and paint it on the screen(browser) display.*

- ***What is Virtual DOM ?***
- In simple words, virtual DOM is just a copy of the original DOM kept in the memory and synced with the real DOM by libraries such as ReactDOM.
- This process is called **Reconciliation(**the process of making consistent or compatible)**.**
- **Diffing Algorithm is used for** generating the minimum number of operations to transform one tree into another.
- Virtual DOM has the same properties that of the Real DOM, but it lacks the power to directly change the content of the screen.

- ***How Virtual DOM works ?***
- So when there is a update in the virtual DOM, react compares the virtual DOM with a snapshot of the virtual DOM taken right before the update of the virtual DOM.
- With the help of this comparison React figures out which components in the UI needs to be updated.
- This process is called diffing.
- The algorithm that is used for the diffing process is called as the diffing algorithm.
- Once React knows which components has been updated, then it replaces the original DOM nodes with the updated DOM node.

- **How Virtual DOM helps React:**
- In react, everything is treated as a component be it a functional component or class component.
- A component can contain a state.
- Each time we change something in our JSX file - whenever the state of any component is changed react updates its Virtual DOM tree.
- React maintains two Virtual DOM at each time, one contains the updated Virtual DOM and one which is just the pre-update version of this updated Virtual DOM.
- Now it compares the pre-update version with the updated Virtual DOM and figures out what exactly has changed in the DOM like which components have been changed.
- This process of comparing the current Virtual DOM tree with the previous one is known as **‘diffing’**.
- Once React finds out what exactly has changed then it updated those objects only, on real DOM.
- React uses something called batch updates to update the real DOM.

- *It just means that the changes to the real DOM are sent in batches instead of sending any update for a single change in the state of a component.*
- *We have seen that the re-rendering of the UI is the most expensive part and React manages to do this most efficiently by ensuring that the Real DOM receives batch updates to re-render the UI.*
- *This entire process of transforming changes to the real DOM is called Reconciliation*

- *Instead of allowing the browser to redraw all the page elements after every re-render or DOM update, React uses the concept of virtual DOM to figure out what exactly has changed without involving the actual DOM and then ensures that the actual DOM only repaints the necessary data.*

- Frequent DOM manipulations are expensive.
- Virtual DOM is a virtual representation of DOM in memory.
- Virtual DOM is synced with real DOM with ReactDOM library.
- This process is called Reconciliation.
- React compares the Virtual DOM and pre-updated Virtual DOM and only marks the sub-tree of components that are updated.
- This process is called diffing.
- The algorithm behind diffing is called Diffing algorithm.
- React uses keys to avoid unnecessary re-renders.

- ***How does React Work?***
- React creates a VIRTUAL DOM in memory.

- Instead of manipulating the browser's DOM directly, React creates a virtual DOM in memory, where it does all the necessary manipulating, before making the changes in the browser DOM.

- React only changes what needs to be changed!

- React finds out what changes have been made, and changes only what needs to be changed.

- ***Modules***
- JavaScript modules allow you to break up your code into separate files.

- This makes it easier to maintain the code-base.

- ES6 Modules rely on the import and export statements.

- ***Export***
- You can export a function or variable from any file.
- Let us create a file named person.js, and fill it with the things we want to export.
- There are two types of exports: Named and Default.
- Named Exports
- You can create named exports two ways. In-line individually, or all at once at the bottom.

- ***In-line individually:***
- person.js
- export const name = "Jesse"
- export const age = "40"

- ***All at once at the bottom:***
- person.js
- const name = "Jesse"
- const age = "40"
- export { name, age }

- ***Default Exports***
- Let us create another file, named message.js, and use it for demonstrating default export.
- You can only have one default export in a file.
- ***Example***

message.js

```
const message = () => {
  const name = "Jesse";
  const age = "40";
  return name + ' is ' + age + 'years old.';
};
export default message;


Import message from “./message”
```

- ***Import:***
- *You can import modules into a file in two ways, based on if they are named exports or default exports.*

- *Named exports must be destructured using curly braces. Default exports do not.*

- *Import from named exports*
- *Import named exports from the file person.js:*

- *import { name, age } from "./person.js";*

- ***Import from default exports***
- *Import a default export from the file message.js:*

- *import message from "./message.js";*

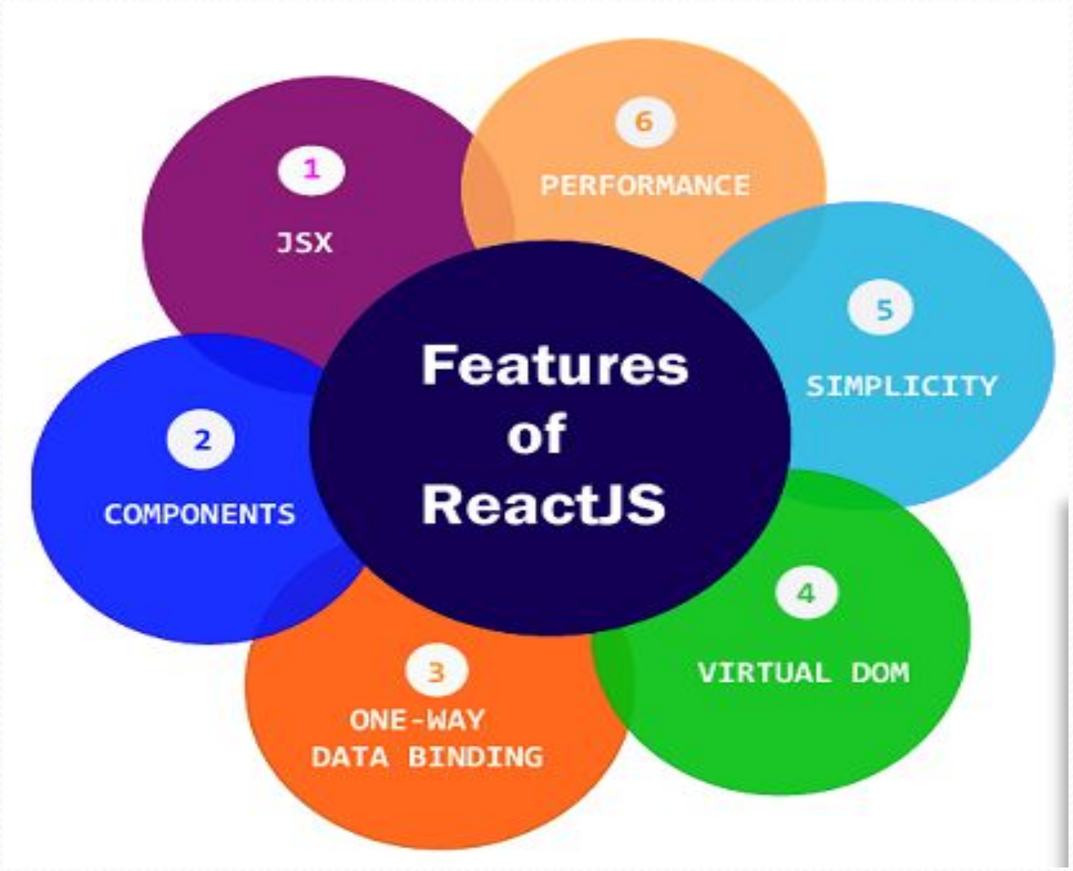
- */file math.js*

```
function square(x) {
  return x * x;
}
function cube(x) {
  return x * x * x;
}
export { square, cube };


//while importing square function in test.js
import { square, cube } from './math;
console.log(square(8)) //64
console.log(cube(8)) //512
```

- ***Default Exports:***
- *Default exports are useful to export only a single object, function, variable. During the import, we can use any name to import.*
- *// file module.js*

```
var x=4;
export default x;


// test.js
// while importing x in test.js
import y from './module';
// note that y is used import x instead of
// import x, because x was default export
console.log(y);
// output will be 4
```

- **React Features**
- Currently, ReactJS gaining quick popularity as the best JavaScript framework among web developers.
- It is playing an essential role in the front-end ecosystem.
- The important features of ReactJS are as following.
- **JSX**
- **Components**
- **One-way Data Binding**
- **Virtual DOM**
- **Simplicity**
- **Performance**



- ***JSX***
- JSX stands for JavaScript XML. It is a JavaScript syntax extension.
- Its an XML or HTML like syntax used by ReactJS.
- This syntax is processed into JavaScript calls of React Framework.
- It extends the ES6 so that HTML like text can co-exist with JavaScript react code.
- It is not necessary to use JSX, but it is recommended to use in ReactJS.

- ***Components***
- ReactJS is all about components. ReactJS application is made up of multiple components, and each component has its own logic and controls.
- These components can be reusable which help you to maintain the code when working on larger scale projects.

- ***Data Binding:***
- *Data Binding is the process of connecting the view element or user interface, with the data which populates it.*

- *In ReactJS, components are rendered to the user interface and the component's logic contains the data to be displayed in the view(UI). The connection between the data to be displayed in the view and the component's logic is called data binding in ReactJS.*

- *One-way Data Binding: ReactJS uses one-way data binding. In one-way data binding one of the following conditions can be followed:*

- *Component to View: Any change in component data would get reflected in the view.*
- *View to Component: Any change in View would get reflected in the component's data.*

- **One-way Data Binding**
- ReactJS is designed in such a manner that follows unidirectional data flow or one-way data binding.
- The benefits of one-way data binding give you better control throughout the application.
- If the data flow is in another direction, then it requires additional features.
- It is because components are supposed to be immutable and the data within them cannot be changed.
- Flux is a pattern that helps to keep your data unidirectional.
- This makes the application more flexible that leads to increase efficiency.

- **Virtual DOM**
- A virtual DOM object is a representation of the original DOM object.
- It works like a one-way data binding. Whenever any modifications happen in the web application, the entire UI is re-rendered in virtual DOM representation.
- Then it checks the difference between the previous DOM representation and new DOM.
- Once it has done, the real DOM will update only the things that have actually changed.
- This makes the application faster, and there is no wastage of memory.

- **Simplicity**
- ReactJS uses JSX file which makes the application simple and to code as well as understand.
- We know that ReactJS is a component-based approach which makes the code reusable as your need. This makes it simple to use and learn.

- **Performance**
- ReactJS is known to be a great performer. This feature makes it much better than other frameworks out there today.
- The reason behind this is that it manages a virtual DOM.
- The DOM is a cross-platform and programming API which deals with HTML, XML or XHTML.
- The DOM exists entirely in memory.
- Due to this, when we create a component, we did not write directly to the DOM.
- Instead, we are writing virtual components that will turn into the DOM leading to smoother and faster performance.

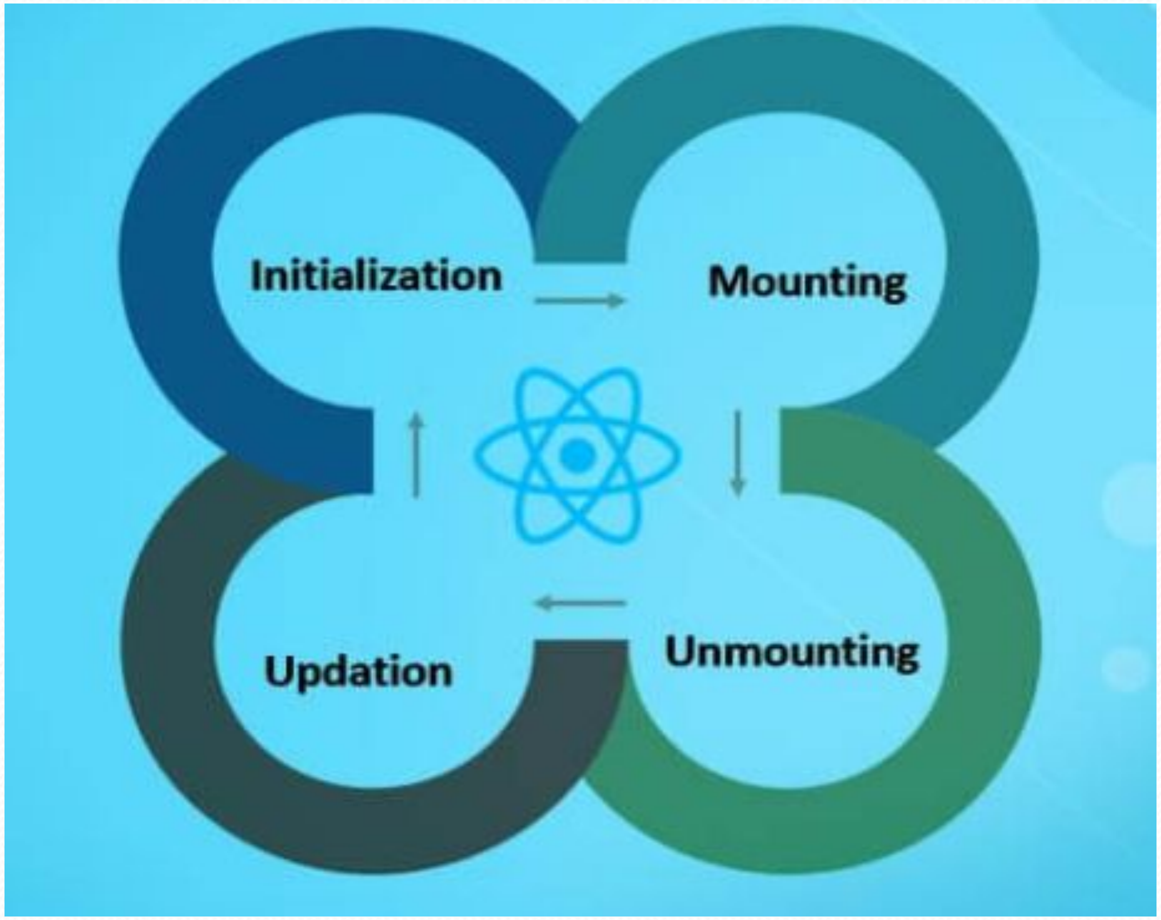
- **React Components**
- A Component is considered as the core building blocks of a React application. It makes the task of building UIs much easier.
- Each component exists in the same space, but they work independently from one another and merge all in a parent component, which will be the final UI of your application.
- Every React component have their own structure, methods as well as APIs.
- They can be reusable as per your need.
- For better understanding, consider the entire UI as a tree.
- Here, the root is the starting component, and each of the other pieces becomes branches, which are further divided into sub-branches.

- ***React Components***
- Components are like functions that return HTML elements.

- ***React Components***
- Components are independent and reusable bits of code.
- They serve the same purpose as JavaScript functions, but work in isolation and return HTML.
- Components come in two types, Class components and Function components.

- ***The Component Lifecycle***
- In ReactJS, every component creation process involves various lifecycle methods.
- These lifecycle methods are termed as component's lifecycle.
- These lifecycle methods are called at various points during a component's life.
- The lifecycle of the component is divided into **four phases**.
- **They are:**
- **Initial Phase**
- **Mounting Phase**
- **Updating Phase**
- **Unmounting Phase**

The Component Lifecycle



- Each phase contains some lifecycle methods that are specific to the particular phase. Let us discuss each of these phases one by one.
- **1. Initial Phase**
- It is the birth phase of the lifecycle of a ReactJS component.
- Here, the component starts its journey on a way to the DOM.
- In this phase, a component contains the default Props and initial State.
- These default properties are done in the constructor of a component.
- The initial phase only occurs once and consists of the following methods.
- **getDefaultProps()**
- It is used to specify the default value of this.props.
- It is invoked before the creation of the component or any props from the parent is passed into it.
- **getInitialState()**
- It is used to specify the default value of this.state.
- It is invoked before the creation of the component.

- **Initialization**
- This stage requires the developer to define properties and the initial state of the component. This is done in the constructor of the component.
- The following code shows the initialization phase of a react component:

```
class Test extends React.Component {
constructor(props)
{
//Calling parent class constructor
super(props);
// Set initial state
this.state = { hello : "Test component!" };
}
}
```

- **2. Mounting**
- Mounting is the phase of the react lifecycle that comes after the initialization is completed.
- Mounting occurs when the component is placed on the DOM container and the component is rendered on a webpage.
- The mounting phase has two methods which are:
- **compnentWillMount():**
- This method is called just before the component is placed on DOM that is this function gets called just before the render function is executed for the very first time.
- **componentDidMount():**
- This method is called just after the component is placed on DOM that is this function gets called just after the render function is executed. For the very first time.

- **Mounting Phase**
- In this phase, the instance of a component is created and inserted into the DOM.
- It consists of the following methods.
- **componentWillMount()**
- This is invoked immediately before a component gets rendered into the DOM.
- When you call setState() inside this method, the component will not re-render.
- **componentDidMount()**
- This is invoked immediately after a component gets rendered and placed on the DOM.
- **render()**
- This method is defined in each and every component.
- It is responsible for returning a single root HTML node element.
- If you don't want to render anything, you can return a null or false value.

- ***3. Updation***
- Updation is a phase where the state and properties populated at the time of initialization are changed if required after some user events.
- The following are different functions invoked during updation phase :
- **componentWillReceiveProps():**
- This function is independent of component state.
- This method is called before a component that is mounted on DOM gets its props reassigned.
- The function accepts new props which can be identical or different from original props.
- Mainly some pre-rendering checks can be applied in this step.

- **shouldComponentUpdate():**
- Sometimes it is desirable not to show the new props on the output page.
- To achieve this, this method returns false, which means the newly rendered props should not be displayed on the output page.
- **componentWillUpdate():**
- This function is called before a component is re-rendered that is this method is called once before the render function is executed post updation.
- **componentDidUpdate():**
- This function is called after a component is re-rendered that is this method is called once after the render function is executed post updation.

- ***Updating Phase***
- It is the next phase of the lifecycle of a react component.
- Here, we get new Props and change State.
- This phase also allows to handle user interaction and provide communication with the components hierarchy.
- The main aim of this phase is to ensure that the component is displaying the latest version of itself.
- This phase repeats again and again.
- This phase consists of the following methods.
- **componentWillRecieveProps()**
- It is invoked when a component receives new props.
- If you want to update the state in response to prop changes, you should compare this.props and nextProps to perform state transition by using this.setState() method.

- **shouldComponentUpdate()**
- It is invoked when a component decides any changes/updation to the DOM.
- It allows you to control the component's behavior of updating itself.
- If this method returns true, the component will update.
- Otherwise, the component will skip the updating.
- **componentWillUpdate()**
- It is invoked just before the component updating occurs.
- Here, you can't change the component state by invoking this.setState() method.
- It will not be called, if shouldComponentUpdate() returns false.
- **render()**
- It is invoked to examine this.props and this.state and return one of the following types: React elements, Arrays and fragments, Booleans or null, String and Number.
- If shouldComponentUpdate() returns false, the code inside render() will be invoked again to ensure that the component displays itself properly.

- ***render***
- The render() method is required, and is the method that actually outputs the HTML to the DOM.
- ***Example:***
- A simple component with a simple render() method:

```
class Header extends React.Component {
  render() {
    return (
      <h1>This is the content of the Header component</h1>
    );
  }
}
ReactDOM.render(<Header />, document.getElementById('root'));
```

- ***componentDidUpdate()***
- It is invoked immediately after the component updating occurs.
- In this method, you can put any code inside this which you want to execute once the updating occurs.
- This method is not invoked for the initial render.

- ***4. Unmounting***
- This is the last phase in the component lifecycle and in this phase, a component is detached from the DOM container.
- The following method falls under this phase.

- ***componentWillUnmount():***
- This function is invoked before a component is finally detached from the DOM container that is this method is called when a component is fully removed from the page and this shows the end of its lifecycle.

- ***Unmounting Phase***
- It is the final phase of the react component lifecycle.
- It is called when a component instance is destroyed and unmounted from the DOM.
- This phase contains only one method and is given below.
- ***componentWillUnmount()***
- This method is invoked immediately before a component is destroyed and unmounted permanently.
- It performs any necessary cleanup related task such as invalidating timers, event listener, canceling network requests, or cleaning up DOM elements.
- If a component instance is unmounted, you cannot mount it again.

- ***Example of React Lifecycle***
- ***example showing the lifecycle of a react component.***

```
class TestLifecycle extends React.Component {
constructor(props)
{
super(props);
this.state = { hello : "React World!" };
}
componentWillMount()
{
console.log("componentWillMount() called");
}
componentDidMount()
{
console.log("componentDidMount() called");
}
```

```
changeState()
{
this.setState({ hello : "Changed React World!" });
}
render()
{
return (
<div>
<h1>Edubca , Hello{ this.state.hello }</h1>
<h2>
<a onClick={this.changeState.bind(this)}>Click Here!</a>
</h2>
</div>);
}
```

```
shouldComponentUpdate(nextProps, nextState)
{
console.log("shouldComponentUpdate() called");
return true;
}
componentWillUpdate()
{
console.log("componentWillUpdate() called");
}
componentDidUpdate()
{
console.log("componentDidUpdate() called");
}
}
ReactDOM.render(
<TestLifecycle />,
document.getElementById('root'));
```

- **Introduction to React JS Components:**
- A component can be defined as a reusable block of code that divides the user interface into smaller parts.
- Basically, there are two types of components in react JS.
- Class Components and functional components.
- **Class components are stateful components whereas functional components are stateless components**.
- Class components are called stateful components because they allow us to do more complex things like react to state management whereas a stateless component does not provide such capability.
- Every application in react can be considered as a collection of components.
- Components play a major role in user interface designing as a given user interface is broken down into several components.
- These components can be developed independently and then finally merged to create a final user interface.

- ***Components of React JS***
- Here are the main types of components in react:
- ***1. Functional Components***
- Functional components can be considered as simple functions build using javascript.
- These are stateless components that cannot be used to achieve complex tasks. Functional components can be created using the below syntax:

- Syntax:

```
function Componentname()
{
return <h1>Welcome Text!</h1>;
}
```

- ***2. Class Components***
- These are more complex components as compared to functional components. Due to their complex nature, they are stateful in nature.
- Class components can be used whenever state management is required to react.
- The functional components have no idea about other components present in the application whereas class components can communicate with other components that their data can be transferred to other components using class components.
- Creating a class component requires us to create javascript classes.
- ***Syntax:***

```
class Componentname extends React.Component
{
render(){
return <h1>Welcome Text!</h1>;
}
}
```

- **React Class Components**
- Class components were the only way to track state and lifecycle on a React component.
- Function components were considered "state-less".
- With the addition of Hooks, Function components are now almost equivalent to Class components.
- The differences are so minor that you will probably never need to use a Class component in React.
- Even though Function components are preferred, there are no current plans on removing Class components from React.

- ***The below example shows a simple functional component and how is it called.***

```
import React from 'react';
import ReactDOM from 'react-dom';
// functional component
function StartPage()
{
return <h1>Welcome to React!</h1>
}
ReactDOM.render(
<StartPage />,
document.getElementById("root")
);
```

- **Class Components:**

```
import React from 'react';
import ReactDOM from 'react-dom';
//class component
class Bike extends React.Component {
constructor() {
super();
this.state = {color: "green"};
}
render() {
return <h2>This is a Bike!</h2>;
}
}
ReactDOM.render(
< Bike />,
document.getElementById("root")
);
```

- ***Lifecycle Methods***
- ***render()***
- The render() method is the only required method in a class component.
- When called, it should examine this.props and this.state and return one of the following types:
- React elements. Typically created via JSX. For example, <div /> and <MyComponent /> are React elements that instruct React to render a DOM node, or another user-defined component, respectively.

- ***constructor()***
- ***constructor(props)***
- If you don’t initialize state and you don’t bind methods, you don’t need to implement a constructor for your React component.
- The constructor for a React component is called before it is mounted.
- When implementing the constructor for a React.Component subclass, you should call super(props) before any other statement.
- Otherwise, this.props will be undefined in the constructor, which can lead to bugs.
- In React constructors are only used for two purposes:
- Initializing local state by assigning an object to this.state.
- Binding event handler methods to an instance.

- ***Create Your First Component***
- When creating a React component, the component's name MUST start with an upper case letter.

- ***Class Component***
- A class component must include the extends React.Component statement. This statement creates an inheritance to React.Component, and gives your component access to React.Component's functions.
- The component also requires a render() method, this method returns HTML.

- ***Example***
- ***Create a Class component called Car***

```
class Car extends React.Component {
  render() {
    return <h2>Hi, I am a Car!</h2>;
  }
}
```

- ***Function Component***
- Here is the same example as above, but created using a Function component instead.
- A Function component also returns HTML, and behaves much the same way as a Class component, but Function components can be written using much less code, are easier to understand, and will be preferred in this tutorial.

- ***Example***
- ***Create a Function component called Car***

```
function Car() {
  return <h2>Hi, I am a Car!</h2>;
}
```

- **Rendering a Component**
- Now your React application has a component called Car, which returns an <h2> element.
- To use this component in your application, use similar syntax as normal HTML: <Car />
- ***Example***
- Display the Car component in the "root" element:
- ReactDOM.render(<Car />, document.getElementById('root'));

- **Props**
- Components can be passed as props, which stands for properties.
- Props are like function arguments, and you send them into the component as attributes.
- **Example**
- Use an attribute to pass a color to the Car component, and use it in the render() function:

```
function Car(props) {
  return <h2>I am a {props.color} Car!</h2>;
}
ReactDOM.render(<Car color="red"/>, document.getElementById('root'));
```

- **Components in Components**
- We can refer to components inside other components:
- **Example**
- Use the Car component inside the Garage component:

```
function Car() {
  return <h2>I am a Car!</h2>;
}

function Garage() {
  return (
    <div>
      <h1>Who lives in my Garage?</h1>
      <Car />
    </div>
  );
}
```

- **Components in Files**
- React is all about re-using code, and it is recommended to split your components into separate files.
- To do that, create a new file with a .js file extension and put the code inside it:
- Note that the filename must start with an uppercase character.
- **Example**

This is the new file, we named it "Car.js":

```
function Car() {
  return <h2>Hi, I am a Car!</h2>;
}
export default Car;
```

- To be able to use the Car component, you have to import the file in your application.
- **Example**
- Now we import the "Car.js" file in the application, and we can use the Car component as if it was created here.

```
import React from 'react';
import ReactDOM from 'react-dom';
import Car from './Car.js';
ReactDOM.render(<Car />, document.getElementById('root'));
```

- **React Class Components**
- Class components were the only way to track state and lifecycle on a React component.
- Function components were considered "state-less".
- With the addition of Hooks, Function components are now almost equivalent to Class components.
- The differences are so minor that you will probably never need to use a Class component in React.
- Even though Function components are preferred, there are no current plans on removing Class components from React.

- **Create a Class Component**
- When creating a React component, the component's name must start with an upper case letter.
- The component has to include the extends React.Component statement, this statement creates an inheritance to React.Component, and gives your component access to React.Component's functions.
- The component also requires a render() method, this method returns HTML.

- ***Example***
- **Create a Class component called Car**

```
class Car extends React.Component {
  render() {
    return <h2>Hi, I am a Car!</h2>;
  }
}
```

- Now your React application has a component called Car, which returns a <h2> element.
- To use this component in your application, use similar syntax as normal HTML: <Car />
- **Example**
- Display the Car component in the "root" element:
- ReactDOM.render(<Car />, document.getElementById('root'));

- **Component Constructor**
- If there is a constructor() function in your component, this function will be called when the component gets initiated.
- The constructor function is where you initiate the component's properties.
- In React, component properties should be kept in an object called state.
- You will learn more about state later in this tutorial.
- The constructor function is also where you honor the inheritance of the parent component by including the super() statement, which executes the parent component's constructor function, and your component has access to all the functions of the parent component (React.Component).

- ***Example***
- **Create a constructor function in the Car component, and add a color property:**

```
class Car extends React.Component {
  constructor() {
    super();
    this.state = {color: "red"};
  }
  render() {
    return <h2>I am a Car!</h2>;
  }
}
```

- **Use the color property in the render() function:**

Example

```
class Car extends React.Component {
  constructor() {
    super();
    this.state = {color: "red"};
  }
  render() {
    return <h2>I am a {this.state.color} Car!</h2>;
  }
}
```

- **Props**
- Another way of handling component properties is by using props.
- Props are like function arguments, and you send them into the component as attributes.
- You will learn more about props in the next chapter.
- **Example**
- Use an attribute to pass a color to the Car component, and use it in the render() function:

```
class Car extends React.Component {
  render() {
    return <h2>I am a {this.props.color} Car!</h2>;
  }
}
ReactDOM.render(<Car color="red"/>, document.getElementById('root'));
```

- **Props in the Constructor**
- If your component has a constructor function, the props should always be passed to the constructor and also to the React.Component via the super() method.
- ***Example***

```
class Car extends React.Component {
  constructor(props) {
    super(props);
  }
  render() {
    return <h2>I am a {this.props.model}!</h2>;
  }
}
ReactDOM.render(<Car model="Mustang"/>, document.getElementById('root'));
```

- **Components in Components**
- **We can refer to components inside other components:**
- ***Example***
- **Use the Car component inside the Garage component:**

```
class Car extends React.Component {
 render() {
   return <h2>I am a Car!</h2>;
 }
}
```

```
class Garage extends React.Component {
  render() {
    return (
      <div>
      <h1>Who lives in my Garage?</h1>
      <Car />
      </div>
    );
  }
}

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

- **Components in Files**
- React is all about re-using code, and it can be smart to insert some of your components in separate files.
- To do that, create a new file with a .js file extension and put the code inside it:
- Note that the file must start by importing React (as before), and it has to end with the statement export default Car;.
- ***Example***
- This is the new file, we named it Car.js:

```
import React from 'react';
class Car extends React.Component {
  render() {
    return <h2>Hi, I am a Car!</h2>;
  }
}
export default Car;
```

- **To be able to use the Car component, you have to import the file in your application.**

- ***Example***
- Now we import the Car.js file in the application, and we can use the Car component as if it was created here.
- import React from 'react';
- import ReactDOM from 'react-dom';
- import Car from './Car.js';
- ReactDOM.render(<Car />, document.getElementById('root'));

- **React Class Component State**
- React Class components have a built-in state object.
- You might have noticed that we used state earlier in the component constructor section.
- The state object is where you store property values that belongs to the component.
- When the state object changes, the component re-renders.

- ***Creating the state Object***
- The state object is initialized in the constructor:
- ***Example***
- Specify the state object in the constructor method:

```
class Car extends React.Component {
 constructor(props) {
  super(props);
 this.state = {brand: "Ford"};
 }
 render() {
  return (
   <div>
    <h1>My Car</h1>
   </div>
  );
 }
}
```

- ***The state object can contain as many properties as you like:***
- ***Example***
- ***Specify all the properties your component need:***

```
class Car extends React.Component {
 constructor(props) {
  super(props);
  this.state = {
   brand: "Ford",
   model: "Mustang",
   color: "red",
   year: 1964
  };
 }
render() {
  return (
    <div>
      <h1>My Car</h1>
    </div>
  );
 }
}
```

- ***React Class Component State***
- React Class components have a built-in state object.
- The state object is where you store property values that belongs to the component.
- When the state object changes, the component re-renders.

- **Creating the state Object**

The state object is initialized in the constructor:

Example:

Specify the state object in the constructor method:

```
class Car extends React.Component {
  constructor(props) {
    super(props);
  this.state = {brand: "Ford"};
  }
  render() {
    return (
      <div>
        <h1>My Car</h1>
      </div>
    );
  }
}
```

- **The state object can contain as many properties as you like:**
- ***Example***
- **Specify all the properties your component need:**

```
class Car extends React.Component {
 constructor(props) {
  super(props);
  this.state = {
   brand: "Ford",
   model: "Mustang",
   color: "red",
   year: 1964
  };
 }
render() {
   return (
     <div>
       <h1>My Car</h1>
     </div>
   );
 }
}
```

- **Using the state Object**
- Refer to the state object anywhere in the component by using the this.state.propertyname syntax:
- ***Example:***
- Refer to the state object in the render() method:

```
class Car extends React.Component {
 constructor(props) {
  super(props);
  this.state = {
   brand: "Ford",
   model: "Mustang",
   color: "red",
   year: 1964
  };
 }
render() {
  return (
    <div>
      <h1>My {this.state.brand}</h1>
      <p>
        It is a {this.state.color}
        {this.state.model}
        from {this.state.year}.
      </p>
    </div>
  );
 }
}
```

- ***Changing the state Object***
- To change a value in the state object, use the this.setState() method.
- When a value in the state object changes, the component will re-render, meaning that the output will change according to the new value(s).

- ***Example:***
- **Add a button with an onClick event that will change the color property:**

```
class Car extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      brand: "Ford",
      model: "Mustang",
      color: "red",
      year: 1964
    };
  }
```

```
 changeColor = () => {
   this.setState({color: "blue"});
  }
  render() {
   return (
     <div>
       <h1>My {this.state.brand}</h1>
       <p>
         It is a {this.state.color}
         {this.state.model}
         from {this.state.year}.
       </p>
       <button
         type="button"
         onClick={this.changeColor}
       >Change color</button>
     </div>
   );
  }
}
```

React State

- The state is an updatable structure that is used to contain data or information about the component.
- The state in a component can change over time.
- The change in state over time can happen as a response to user action or system event.
- A component with the state is known as stateful components.
- It is the heart of the react component which determines the behavior of the component and how it will render.
- They are also responsible for making a component dynamic and interactive.

- A state must be kept as simple as possible.
- It can be set by using the setState() method and calling setState() method triggers UI updates.
- A state represents the component's local state or information.
- It can only be accessed or modified inside the component or by the component directly.
- To set an initial state before any interaction occurs, we need to use the getInitialState() method.

- **Defining State**
- To define a state, you have to first declare a default set of values for defining the component's initial state.
- To do this, add a class constructor which assigns an initial state using this.state. The 'this.state' property can be rendered inside render() method.

```
import React, { Component } from 'react';
class App extends React.Component {
 constructor() {
      super();
      this.state = { displayBio: true };
      }
      render() {
          const bio = this.state.displayBio ? (
              <div>
              <p><h3>Javatpoint is one of the best Java training institute in Noida,
Delhi, Gurugram, Ghaziabad and Faridabad. We have a team of experienced Java
developers and trainers from multinational companies to teach our campus
students.</h3></p>
            </div>
              ) : null;
 return (
 <div>
                     <h1> Welcome to Java point!! </h1>
                     { bio }
                 </div>
             );
     }
}
export default App;
```

To set the state, it is required to call the super() method in the constructor.

It is because this.state is uninitialized before the super() method has been called.

- **React Props**
- Props stand for "Properties." They are read-only components.
- It is an object which stores the value of attributes of a tag and work similar to the HTML attributes.
- It gives a way to pass data from one component to other components.
- It is similar to function arguments.
- Props are passed to the component in the same way as arguments passed in a function.
- Props are immutable so we cannot modify the props from inside the component.
- Inside the components, we can add attributes called props.
- These attributes are available in the component as this.props and can be used to render dynamic data in our render method.
- When you need immutable data in the component, you have to add props to reactDom.render() method in the index.js file of your ReactJS project and used it inside the component in which you need.

```
import React, { Component } from 'react';
class App extends React.Component {
   render() {
      return (
         <div>
           <h1> Welcome to { this.props.name } </h1>
           <p> <h4> Javatpoint is one of the best Java training institute in Noida,
Delhi, Gurugram, Ghaziabad and Faridabad. </h4> </p>
         </div>
      );
   }
}
export default App;
```

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App.js';
ReactDOM.render(<App name = "JavaTpoint!!" />,
document.getElementById('app'));
```

- ***Default Props***
- It is not necessary to always add props in the reactDom.render() element.
- You can also set default props directly on the component constructor. It can be explained in the below example.

App.js

```
import React, { Component } from 'react';
class App extends React.Component {
   render() {
      return (
          <div>
              <h1>Default Props Example</h1>
            <h3>Welcome to {this.props.name}</h3>
            <p>Javatpoint is one of the best Java training institute >
          </div>
        );
    }
}
App.defaultProps = {
   name: "JavaTpoint"
}
export default App;
```

Index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App.js';
ReactDOM.render(<App/>, document.getElementById('app'));
```

- ***React Forms***
- Forms are an integral part of any modern web application.
- It allows the users to interact with the application as well as gather information from the users.
- Forms can perform many tasks that depend on the nature of your business requirements and logic such as authentication of the user, adding user, searching, filtering, booking, ordering, etc.
- A form can contain text fields, buttons, checkbox, radio button, etc.

- **Creating Form**
- React offers a stateful, reactive approach to build a form.
- The component rather than the DOM usually handles the React form.
- In React, the form is usually implemented by using controlled components.

- **There are mainly two types of form input in React.**
- Uncontrolled component
- Controlled component
- **Uncontrolled component**
- The uncontrolled input is similar to the traditional HTML form inputs.
- The DOM itself handles the form data.
- Here, the HTML elements maintain their own state that will be updated when the input value changes.
- To write an uncontrolled component, you need to use a ref to get form values from the DOM.
- In other words, there is no need to write an event handler for every state update.
- You can use a ref to access the input field value of the form from the DOM.

```
import React, { Component } from 'react';
class App extends React.Component {
  constructor(props) {
      super(props);
      this.updateSubmit = this.updateSubmit.bind(this);
      this.input = React.createRef();
  }
  updateSubmit(event) {
      alert('You have entered the UserName and CompanyName successfully.');
      event.preventDefault();
  }
  render() {
    return (
      <form onSubmit={this.updateSubmit}>
        <h1>Uncontrolled Form Example</h1>
        <label>Name:
          <input type="text" ref={this.input} />
        </label>
        <label>
          CompanyName:
          <input type="text" ref={this.input} />
        </label>
        <input type="submit" value="Submit" />
      </form>
    );
  }
}
export default App;
```

- ***Controlled Component***
- In HTML, form elements typically maintain their own state and update it according to the user input.
- In the controlled component, the input form element is handled by the component rather than the DOM.
- Here, the mutable state is kept in the state property and will be updated only with setState() method.
- Controlled components have functions that govern the data passing into them on every onChange event, rather than grabbing the data only once, e.g., when you click a submit button.
- This data is then saved to state and updated with setState() method.
- This makes component have better control over the form elements and data.
- A controlled component takes its current value through props and notifies the changes through callbacks like an onChange event.
- A parent component "controls" this changes by handling the callback and managing its own state and then passing the new values as props to the controlled component.
- It is also called as a "dumb component."

```
import React, { Component } from 'react';
class App extends React.Component {
 constructor(props) {
    super(props);
    this.state = {value: ''};
    this.handleChange = this.handleChange.bind(this);
    this.handleSubmit = this.handleSubmit.bind(this);
 }
 handleChange(event) {
    this.setState({value: event.target.value});
 }
 handleSubmit(event) {
    alert('You have submitted the input successfully: ' + this.state.value);
    event.preventDefault();
 }
 render() {
      return (
          <form onSubmit={this.handleSubmit}>
            <h1>Controlled Form Example</h1>
            <label>
               Name:
              <input type="text" value={this.state.value}
onChange={this.handleChange} />
            </label>
            <input type="submit" value="Submit" />
          </form>
      );
  }
}
export default App;
```

- **Forms in React**

```
import React from 'react';
import ReactDOM from 'react-dom';
class TestForm extends React.Component {
render() {
return (
<form>
<h1>Hello</h1>
<p>Please Enter your UserName:</p>
<input
type="text"
/>
</form>
);
}
}
ReactDOM.render(<TestForm />, document.getElementById('root'));
```

```
import React from 'react';
import ReactDOM from 'react-dom';
class TestForm extends React.Component {
constructor(props) {
super(props);
this.state = { username: '' };
}
submitmyeventHandler = (myevent) => {
alert("You are submitting " + this.state.username);
}
changeEventHandler = (myevent) => {
this.setState({username: myevent.target.value});
}
render() {
return (
<form onSubmit={this.submitmyeventHandler}>
<h1>Hello {this.state.username}</h1>
<p>Please Enter your user name, and click submit:</p>
<input
type='text'
onChange={this.changeEventHandler}
/>
<input
type='submit'
/>
</form>
);
}
}
ReactDOM.render(<TestForm />, document.getElementById('root'));
```

- ***React Events***
- An event is an action that could be triggered as a result of the user action or system generated event.
- For example, a mouse click, loading of a web page, pressing a key, window resizes, and other interactions are called events.
- React has its own event handling system which is very similar to handling events on DOM elements.
- The react event handling system is known as Synthetic Events.
- The synthetic event is a cross-browser wrapper of the browser's native event.
- Synthetic events are that **ReactJS reuses these events objects, by pooling them, which increase the performance**.

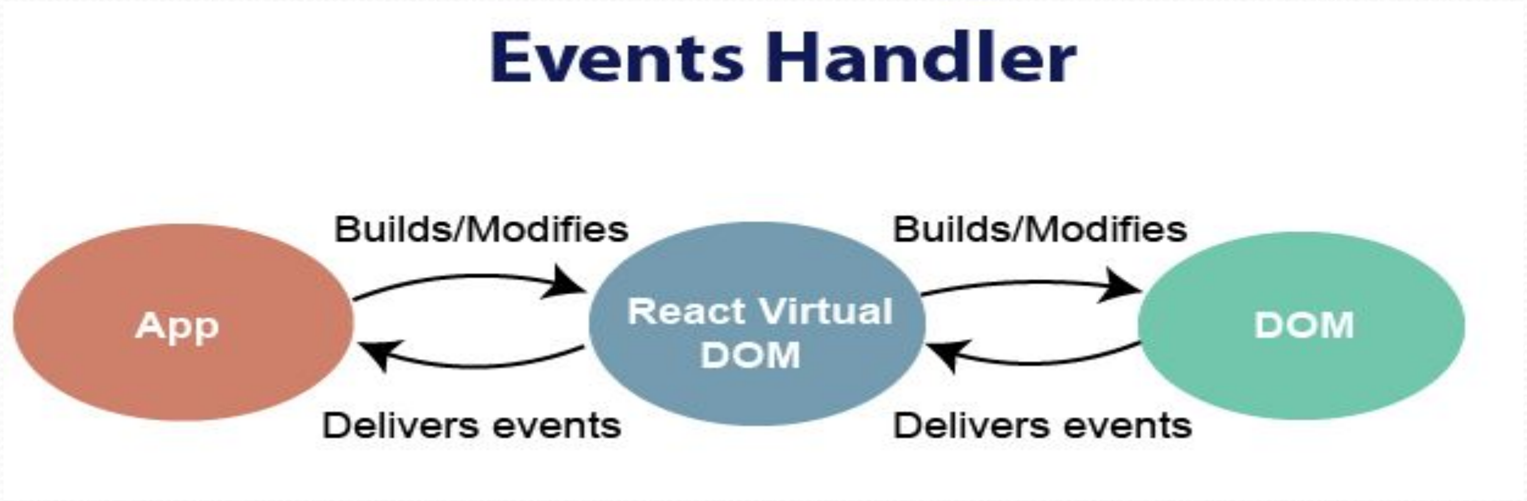
```
import React from "react";
function App() {
  function handleClick() {
   alert("You clicked me!");
 }
 return (
   <div style={{
     display: "flex",
     alignItems: "center",
     justifyContent: "center",
     height: "100vh"
     }}>
    <button onClick={() => {handleClick()}}>show alert </button>
   </div>
 );
}
export default App;
```

- ***Types of Events:***
- ***Clipboard Events***
- ***Composition Events***
- ***Keyboard Events***
- ***Focus Events***
- ***Form Events***
- ***Generic Events***
- ***Mouse Events***
- ***Pointer Events***
- ***Selection Events***
- ***Touch Events***
- ***UI Events***
- ***Wheel Events***
- ***Media Events***
- ***Image Events***
- ***Animation Events***
- ***Transition Events***
- ***Other Events***

- *onFocus*
- *The onFocus event is called when the element (or some element inside of it) receives focus. For example, it’s called when the user clicks on a text input.*

```
function Example() {
 return (
  <input
   onFocus={(e) => {
    console.log('Focused on input');
   }}
   placeholder="onFocus is triggered when you click this input."
  />
 )
}
```

- ***https://reactjs.org/docs/events.html***



Handling events with react have some syntactic differences from handling events on DOM. These are:

1. React events are named as **camelCase** instead of **lowercase**.
2. With JSX, a function is passed as the **event handler** instead of a **string**.

```
function ActionLink() {
  const handleClick = (e) => {
    e.preventDefault();
    console.log('The link was clicked.');
  }
  return (
    <button onClick={handleClick}>
      Click me
    </button>
  );
}
```

- **For example:**
- ***Event declaration in React:***

```
<button onClick={showMessage}>
     Hello JavaTpoint
</button>
 <a href="#" onClick={handleClick}>
            Click_Me
</a>
```

```
import React, { Component } from 'react';
class App extends React.Component {
   constructor(props) {
      super(props);
      this.state = {
         companyName: ''
      };
   }
   changeText(event) {
      this.setState({
         companyName: event.target.value
      });
   }
```

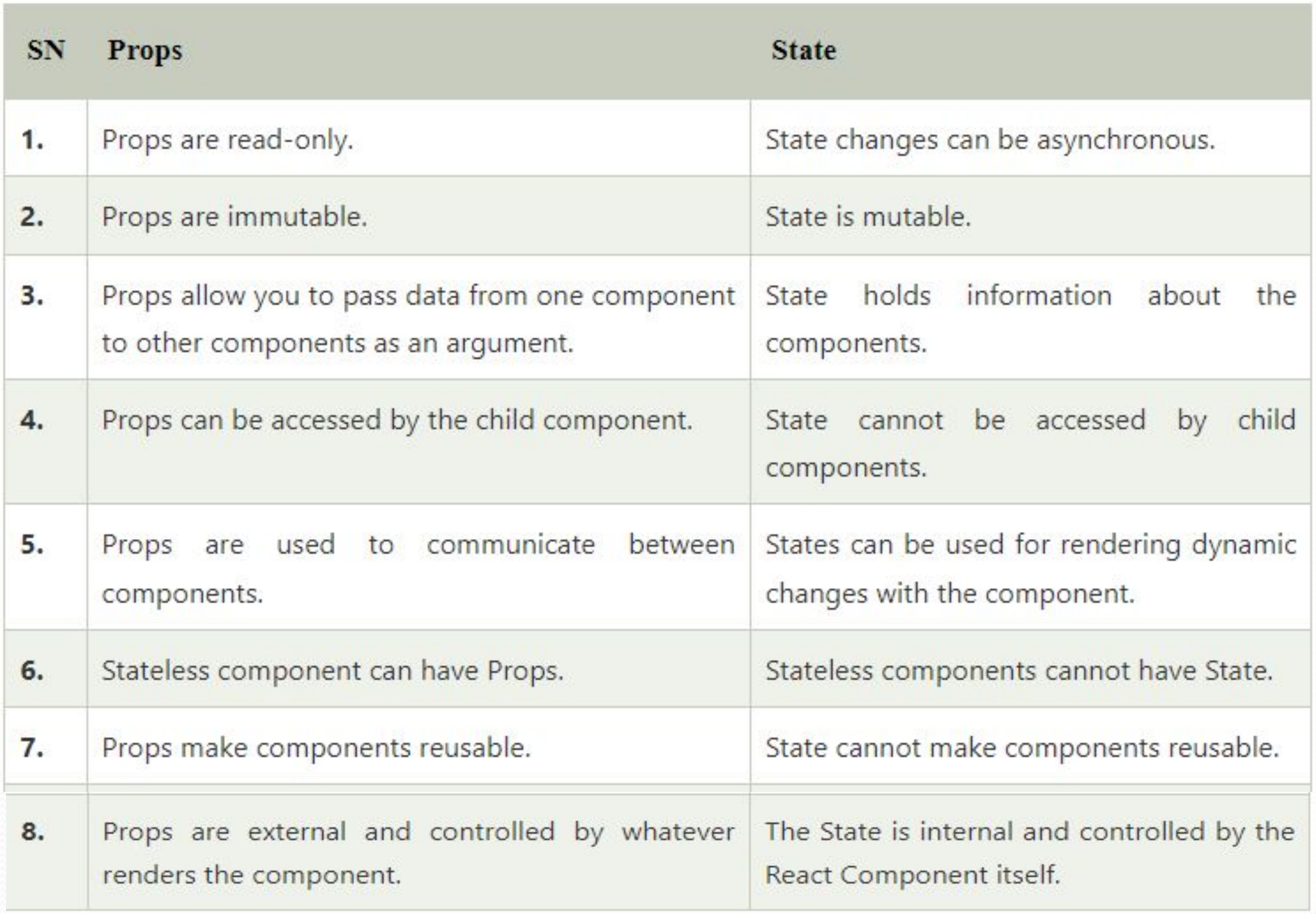
```
 render() {
        return (
            <div>
                <h2>Simple Event Example</h2>
                <label htmlFor="name">Enter company name: </label>
                <input type="text" id="companyName"
onChange={this.changeText.bind(this)}/>
                <h4>You entered: { this.state.companyName }</h4>
            </div>
        );
    }
}
export default App;
```

State and Props

- **State**
- The state is an updatable structure that is used to contain data or information about the component and can change over time.
- The change in state can happen as a response to user action or system event.
- It is the heart of the react component which determines the behavior of the component and how it will render.
- A state must be kept as simple as possible. It represents the component's local state or information.
- It can only be accessed or modified inside the component or by the component directly.

- **Props**
- Props are read-only components.
- It is an object which stores the value of attributes of a tag and work similar to the HTML attributes.
- It allows passing data from one component to other components.
- It is similar to function arguments and can be passed to the component the same way as arguments passed in a function.
- Props are immutable so we cannot modify the props from inside the component.

Difference between State and Props



SN	Props	State
1.	Props are read-only.	State changes can be asynchronous.
2.	Props are immutable.	State is mutable.
3.	Props allow you to pass data from one component to other components as an argument.	State holds information about the components.
4.	Props can be accessed by the child component.	State cannot be accessed by child components.
5.	Props are used to communicate between components.	States can be used for rendering dynamic changes with the component.
6.	Stateless component can have Props.	Stateless components cannot have State.
7.	Props make components reusable.	State cannot make components reusable.
8.	Props are external and controlled by whatever renders the component.	The State is internal and controlled by the React Component itself.

- ***React Router***
- Routing is a process in which a user is directed to different pages based on their action or request.
- ReactJS Router is mainly used for developing Single Page Web Applications.
- React Router is used to define multiple routes in the application.
- When a user types a specific URL into the browser, and if this URL path matches any 'route' inside the router file, the user will be redirected to that particular route.
- React Router is a standard library system built on top of the React and used to create routing in the React application using React Router Package.
- It provides the synchronous URL on the browser with data that will be displayed on the web page.
- It maintains the standard structure and behavior of the application and mainly used for developing single page web applications.

- ***Need of React Router***
- React Router plays an important role to display multiple views in a single page application.
- Without React Router, it is not possible to display multiple views in React applications.
- Most of the social media websites like Facebook, Instagram uses React Router for rendering multiple views.
- ***React Router Installation***
- React contains three different packages for routing. These are:
- ***react-router: It provides the core routing components and functions for the React Router applications.***
- ***react-router-native: It is used for mobile applications.***
- ***react-router-dom: It is used for web applications design.***

- ***What is Route?***
- ***It is used to define and render component based on the specified path.***
- ***It will accept components and render to define what should be rendered.***

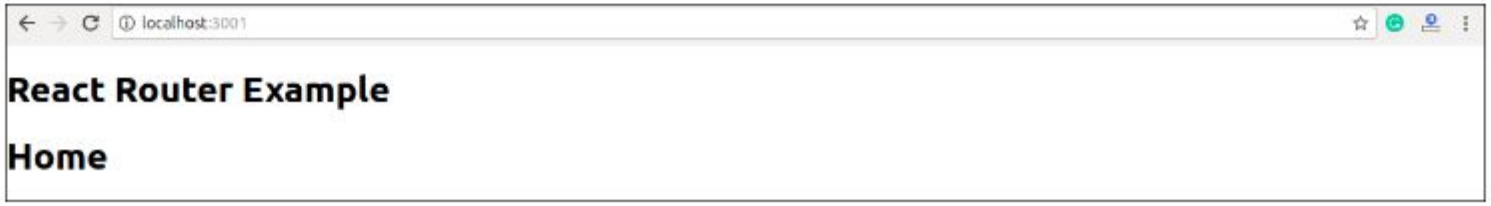
- It is not possible to install react-router directly in your application.
- To use react routing, first, you need to install react-router-dom modules in your application. The below command is used to install react router dom.

- $ npm install react-router-dom --save
- Components in React Router
- There are two types of router components:

- <BrowserRouter>: It is used for handling the dynamic URL.
- <HashRouter>: It is used for handling the static request.

- ***Index.js***

```
import React from 'react';
import ReactDOM from 'react-dom';
import { Route, Link, BrowserRouter as Router } from 'react-router-dom'
import './index.css';
import App from './App';
import About from './about'
import Contact from './contact'
const routing = (
  <Router>
    <div>
      <h1>React Router Example</h1>
      <Route path="/" component={App} />
      <Route path="/about" component={About} />
      <Route path="/contact" component={Contact} />
    </div>
  </Router>
)  ReactDOM.render(routing, document.getElementById('root'));
```

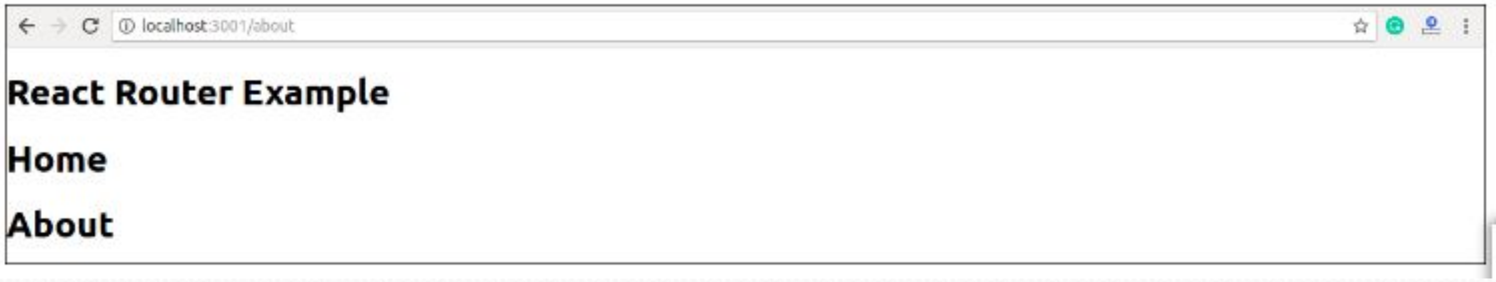


localhost:3001

React Router Example

Home

Now, if you enter **manually** in the browser: **localhost:3000/about**, you will see **About** component is rendered on the screen.



localhost:3001/about

React Router Example

Home

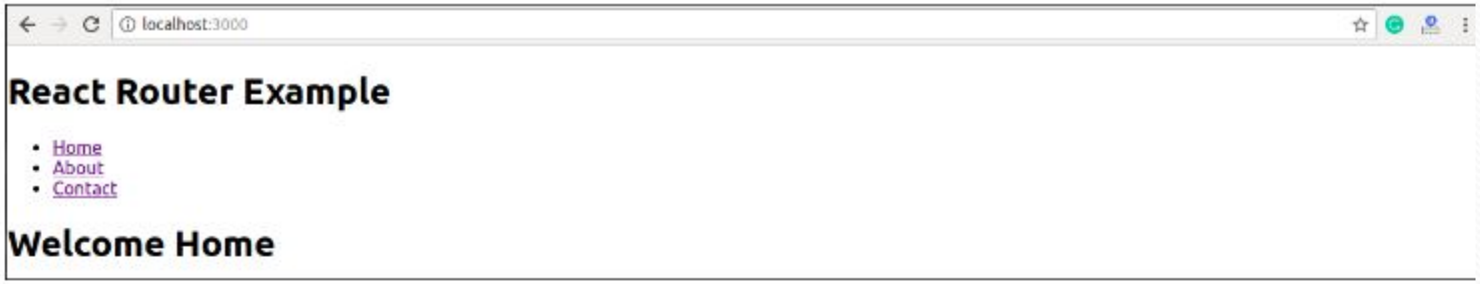
About

- ***What is < Link> component?***
- *This component is used to create links which allow to navigate on different URLs and render its content without reloading the webpage.*

```
import React from 'react';
import ReactDOM from 'react-dom';
import { Route, Link, BrowserRouter as Router } from 'react-router-dom'
import './index.css';
import App from './App';
import About from './about'
import Contact from './contact'

const routing = (
  <Router>
    <div>
      <h1>React Router Example</h1>
      <ul>
        <li>
          <Link to="/">Home</Link>
        </li>
        <li>
```

```
 <Link to="/about">About</Link>
       </li>
       <li>
         <Link to="/contact">Contact</Link>
       </li>
     </ul>
     <Route exact path="/" component={App} />
     <Route path="/about" component={About} />
     <Route path="/contact" component={Contact} />
   </div>
 </Router>
)
ReactDOM.render(routing, document.getElementById('root'));
```

Output

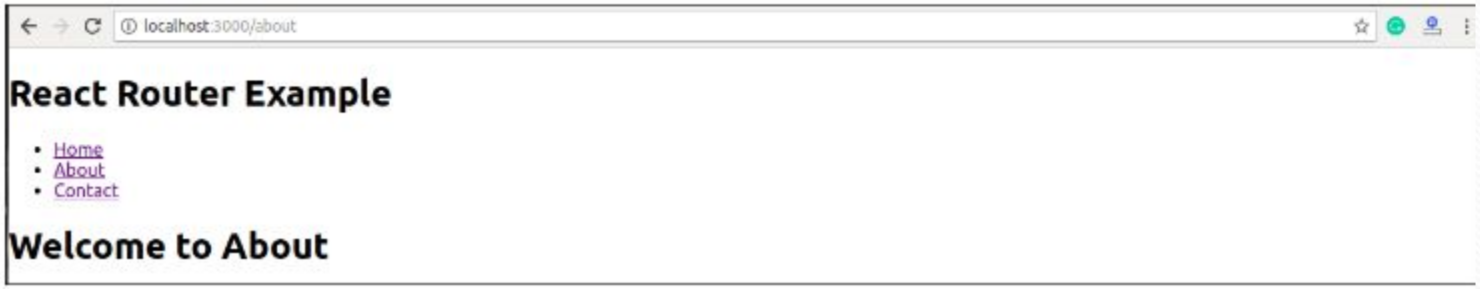

localhost:3000

React Router Example

- Home
- About
- Contact

Welcome Home

After adding Link, you can see that the routes are rendered on the screen. Now, if you click on the **About**, you will see URL is changing and About component is rendered.


localhost:3000/about

React Router Example

- Home
- About
- Contact

Welcome to About

- ***Example***
- ***Step-1: In our project, we will create two more components along with App.js, which is already present.***

- ***About.js***

```
import React from 'react'
class About extends React.Component {
 render() {
  return <h1>About</h1>
 }
}
export default About
```

- ***Contact.js***

```
import React from 'react'
class Contact extends React.Component {
  render() {
    return <h1>Contact</h1>
  }
}
export default Contact
```

App.js

```
import React from 'react'
class App extends React.Component {
  render() {
    return (
      <div>
        <h1>Home</h1>
      </div>
    )
  }
}
export default App
```

- ***Step-2:***
- For Routing, open the index.js file and import all the three component files in it.
- Here, you need to import line: import { Route, Link, BrowserRouter as Router } from 'react-router-dom' which helps us to implement the Routing.

Index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import { Route, Link, BrowserRouter as Router } from 'react-router-dom'
import './index.css';
import App from './App';
import About from './about'
import Contact from './contact'
const routing = (
  <Router>
    <div>
      <h1>React Router Example</h1>
      <Route path="/" component={App} />
      <Route path="/about" component={About} />
      <Route path="/contact" component={Contact} />
    </div>
  </Router>  )
ReactDOM.render(routing, document.getElementById('root'));
```

- ***Step-4:***
- In the above screen, you can see that Home component is still rendered.
- It is because the home path is '/' and about path is '/about', so you can observe that slash is common in both paths which render both components.
- To stop this behavior, you need to use the exact prop. It can be seen in the below example.

```
import React from 'react';
import ReactDOM from 'react-dom';
import { Route, Link, BrowserRouter as Router } from 'react-router-dom'
import './index.css';
import App from './App';
import About from './about'
import Contact from './contact'
  const routing = (
  <Router>
    <div>
      <h1>React Router Example</h1>
      <Route exact path="/" component={App} />
      <Route path="/about" component={About} />
      <Route path="/contact" component={Contact} />
    </div>
  </Router>
)  ReactDOM.render(routing, document.getElementById('root'));
```

- *Adding Navigation using Link component*
- *Sometimes, we want to need multiple links on a single page.*
- *When we click on any of that particular Link, it should load that page which is associated with that path without reloading the web page.*
- *To do this, we need to import <Link> component in the index.js file.*

- *What is < Link> component?*
- *This component is used to create links which allow to navigate on different URLs and render its content without reloading the webpage.*

```
import React from 'react';
import ReactDOM from 'react-dom';
import { Route, Link, BrowserRouter as Router } from 'react-router-dom'
import './index.css';
import App from './App';
import About from './about'
import Contact from './contact'
 const routing = (
  <Router>
    <div>
      <h1>React Router Example</h1>
      <ul>
        <li>
          <Link to="/">Home</Link>
        </li>
<li>
      <Link to="/about">About</Link>
     </li>
     <li>
      <Link to="/contact">Contact</Link>
     </li>
    </ul>
    <Route exact path="/" component={App} />
    <Route path="/about" component={About} />
    <Route path="/contact" component={Contact} />
   </div>
  </Router>
)
ReactDOM.render(routing, document.getElementById('root'));
```

- ***Benefits Of React Router***
- The benefits of React Router is given below:
- In this, it is not necessary to set the browser history manually.
- Link uses to navigate the internal links in the application. It is similar to the anchor tag.
- It uses Switch feature for rendering.
- The Router needs only a Single Child element.
- In this, every component is specified in .
- https://www.javatpoint.com/react-router

- ***https://reactjs.org/docs/components-and-props.html***

- ***Handling Form Inputs in React:***

- ***C:\Users\Your Name> npx create-react-app react-form-handling***

- ***The command will create a project folder called react-form-handling in your choosing directory.***

- ***C:\Users\Your Name\react-form-handling > npm start***

- ***Project running on port 3000***

```
import React from "react"
import "./App.css"
function App() {
  return (
    <div>
      <h1>React Form Handling</h1>
      <form>
        <label>
          First Name: <input type="text" />
        </label>
      </form>
    </div>
  )
}

export default App
```

- ***C:\Users\Your Name> npx create-react-app react-form-handling***

- ***Uncontrolled and Controlled Input***
- *At the moment, just like pure HTML form, this input element maintains its internal state. That is why we can write something in it by default.*
- *In this case, we call this type of input an uncontrolled input.*
- *In React, it is the responsibility of the component rendering the form to control the input state. This way, the input would no longer listen to its internal state but the state declared in its component. By so doing, we are making the component state a single source of truth.*
- *When you have this type of input, then you have a controlled input.*

- *How does it work?*

- *Depending on your component type, you will store your input data in the component state.*
- *Here, we will be using the React Hook to manage our form data. However, the approach is the same if you are using a class-based component.*
- *All you have to do is to declare a state object where your data would live.*

- *From there, you will set up logic to listen to changes in the input and control it (i.e update the state) using the onChange event.*
- *This way, you will always get up-to-date value as you will see in a moment.*

```
import React, { useState } from "react"
import "./App.css"
function App() {
  const [fname, setFname] = useState("")
  return (
    <div>
      <h1>React Form Handling</h1>
      <form>
        <label>
          First Name: <input type="text" value={fname} />
        </label>
      </form>
      <h5>First name: {fname}</h5>
    </div>
  )
}
export default App
```

- ***In the code, we added a state using the useState Hook and assigned a default empty string to the state variable, fname.***
- ***This is similar to declaring a state object in a class-based component.***
- ***The second item return by the useState Hook (I called it setFname, but you can name it anything you like) is a function that will allow us to update the state value.***

- ***Example to Update the code to include an onChange event handler.***

```
import React, { useState } from "react"
import "./App.css"
function App() {
  const [fname, setFname] = useState("")
  const handleChange = e => {
    setFname(e.target.value)
  }
  return (
    <div>
      <h1>React Form Handling</h1>
      <form>
        <label>
          First Name:{" "}
          <input type="text" value={fname} onChange={handleChange} />
        </label>
      </form>
      <h5>First name: {fname}</h5>
    </div>
  )
}
export default App
```

- ***https://ibaslogic.com/simple-guide-to-react-form/***