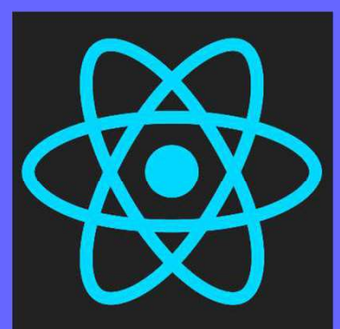
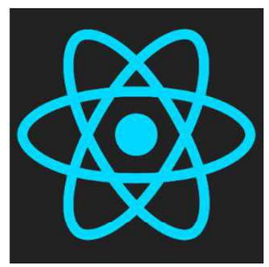
React JS and Components



What is React ?

What Is React ?

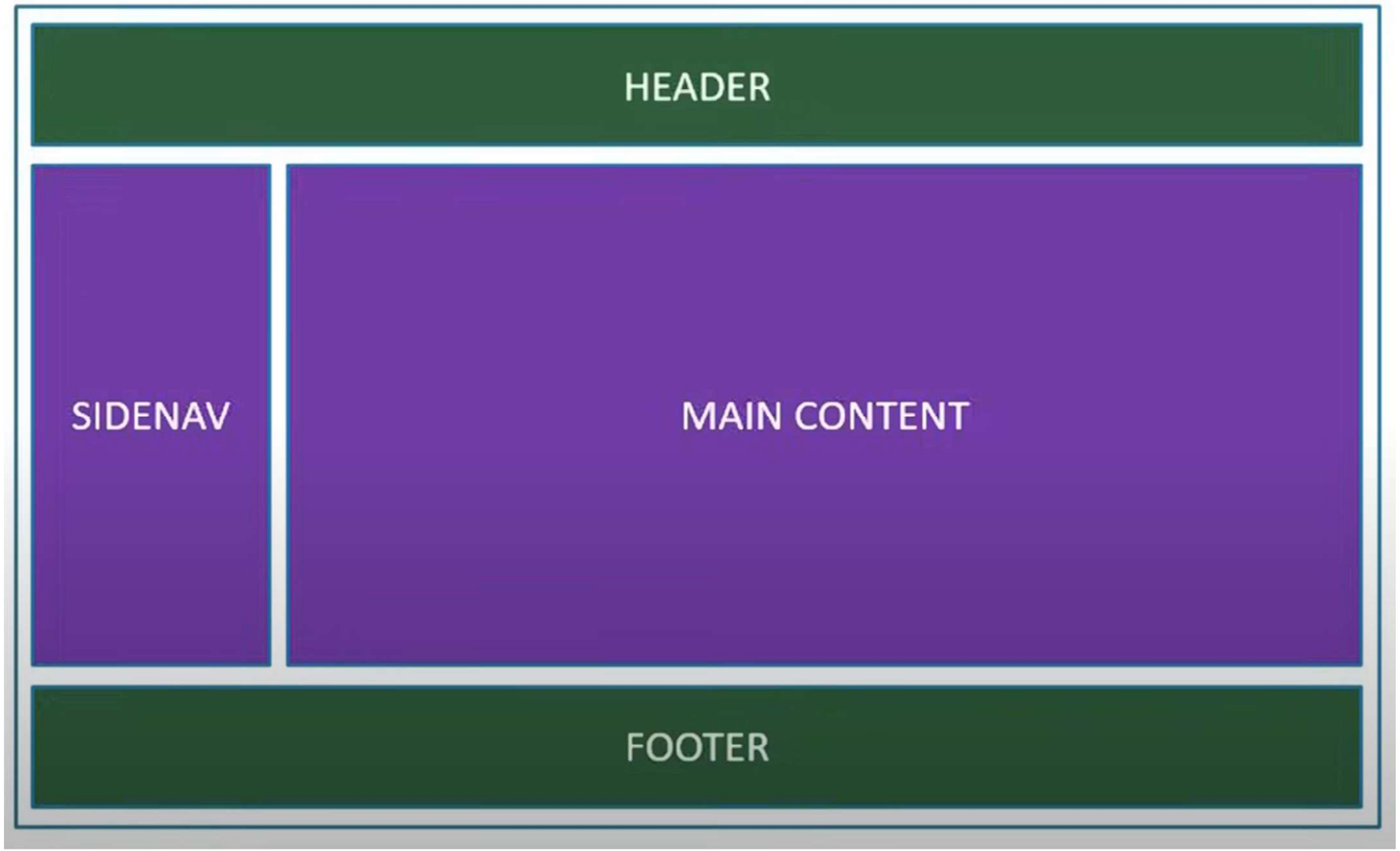
- React JS is a open source JavaScript libraries for building UI & engineered to make it as simple and efficient as possible to build client-side web and hybrid mobile applications based on JavaScript, HTML5, and CSS.
  - It is a Library Not a Framework
  - It is Focused on doing One Thing, And Doing That One Thing Really Well.
  - Focus is UI. [Responsible for Building Rich UI].
  - Rich Eco Systems



Why Learn React ?

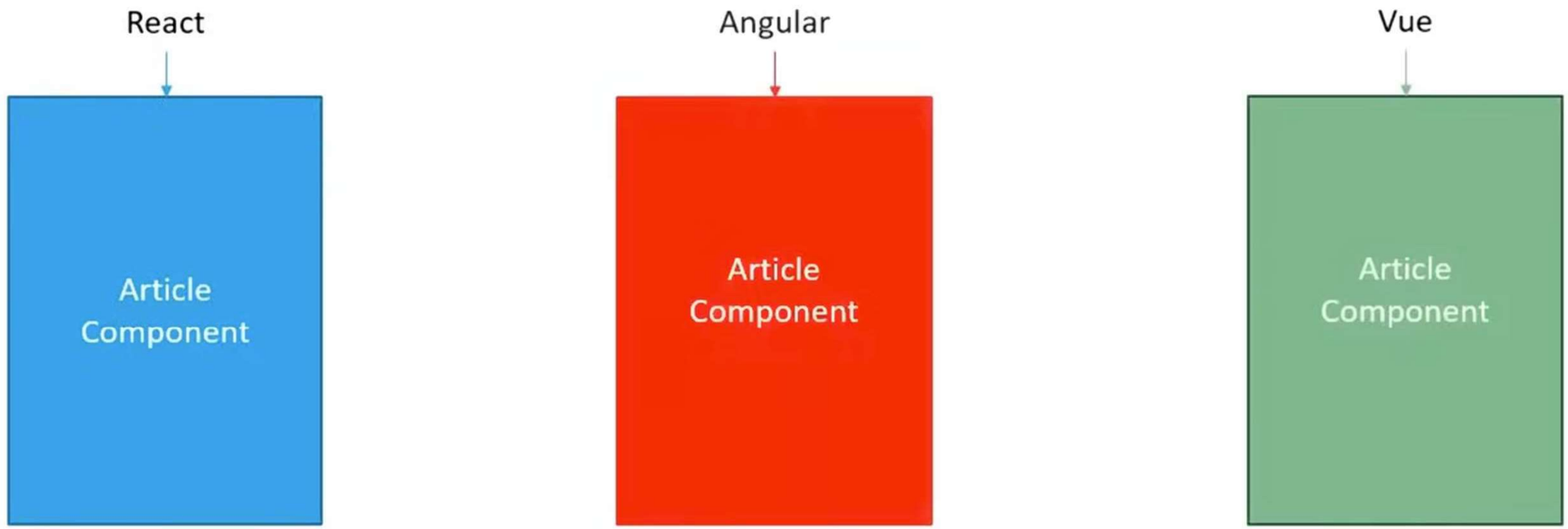
- React is Project Created and maintained by Facebook.
- More than 100k Stars on Github
- Huge Community for Support
- In Demand Skillset
- Unidirectional Data Flow
  - React Applications are Built as a Combination of Parent and Child Components
    - Data Always Flows From Parent to Child and Never in the Other Direction

Component Based Architecture



Resuable Code

- We Could have Component in React, Same Component can be reused in Angular or Vue by simply passing right data into it.



React is Declarative

- Tell React what you want and React Methods and React DOM will build the Actual UI.
  - This is Declarative Paradigm
- Declarative VS Imperative Paradigm
  - Eg : In Declarative We go to an artist and ask them to draw a Landscape, but we don't tell them how to draw it. Its up to them this is Declarative. U tell what has to be done and Artist will get it done for you.
  - Eg : In Imperative We go to a 5 Year old Kid and ask them to draw a Landscape, But this time we tell them first to draw Mountains, then Rivers, then the Trees and so on. We Explicitly specifies Each Steps, And We Control the Flow of the Landscape.

Cont …

- React will handle efficiently Updating and Rendering of the Components.
- DOM Updates are handled gracefully in React.
- Seamlessly Integrate React into any of your Applications.
- Portion of your page or a Complete Page or Even an Entire Application Itself.
- React Native for Mobile Applications.

ReactJS

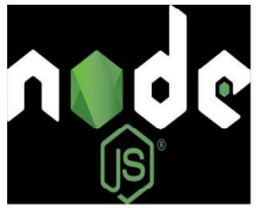
- JavaScript Framework for writing **Web Applications**
  - Like Angular – Snappy Response from Running in Browser
  - Less Opinionated: Only Specifies Rendering View and Handling User Interactions
- User MVC Pattern
  - View Constructed From Components Using Pattern
  - Optional, but Commonly Used HTML Templating
- Minimal Server-Side Support Dicated
- Focus on supporting for Programing in the Large and SPA
  - Modules, Reusable Components, Testing …

Prerequisites

- HTML, CSS and JS
- ES6
- JavaScript – ‘this’ keyword, filter, map and reduce
- ES6 – let const, arrow functions, template Strings

  Default parameters, Object Literals, Rest Parameters,

  Spread Operator and Array Destructuring
- Basics of Type Script



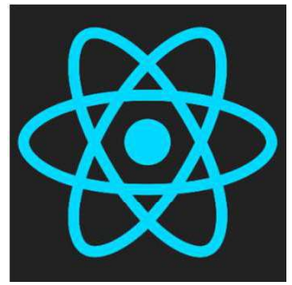
React is designed to meet the following application needs:

- Add interactivity to an existing page.
- Create a new end-to-end client-side web application using JavaScript, HTML5, CSS, and best practices for responsive design.
- Create a hybrid mobile application that looks and feels like a native iOS, Android or Windows application.
- It is a " Structural Framework for Dynamic Web Apps ".
- Helps build interactive, modern Web Applications by increasing abstraction between the Developer and Common Web App Development Tasks.

Why React ?

Why React ?

- Modular Approach
- Re Usable Code
- Development Quicker and easier
- Unit Testable
- Many JS Providers
- Huge Ecosystem



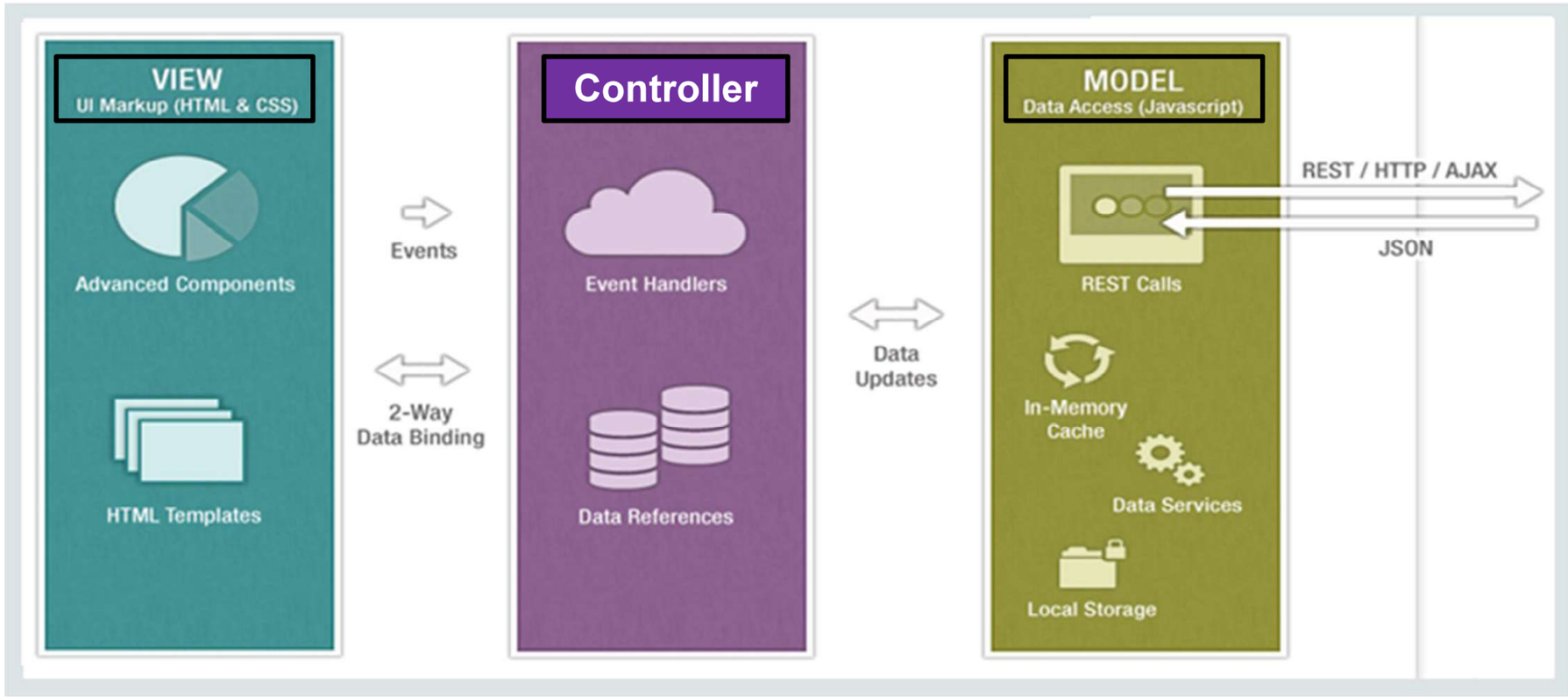
Cont …

- Declarative programming should be used for building UI and associating Software Components.
- “Imperative Programming is Very good for Business Logic”
- React encourages loose coupling between Presentation, Data and Logic Components.

React Architecture

- React supports the Model-View-Controller (MVC) architectural design pattern.

The MVC


VIEW
UI Markup (HTML & CSS)
Advanced Components
HTML Templates
Events
2-Way
Data Binding
Controller
Event Handlers
Data References
Data
Updates
MODEL
Data Access (Javascript)
REST Calls
In-Memory
Cache
Data Services
Local Storage
REST / HTTP / AJAX
JSON

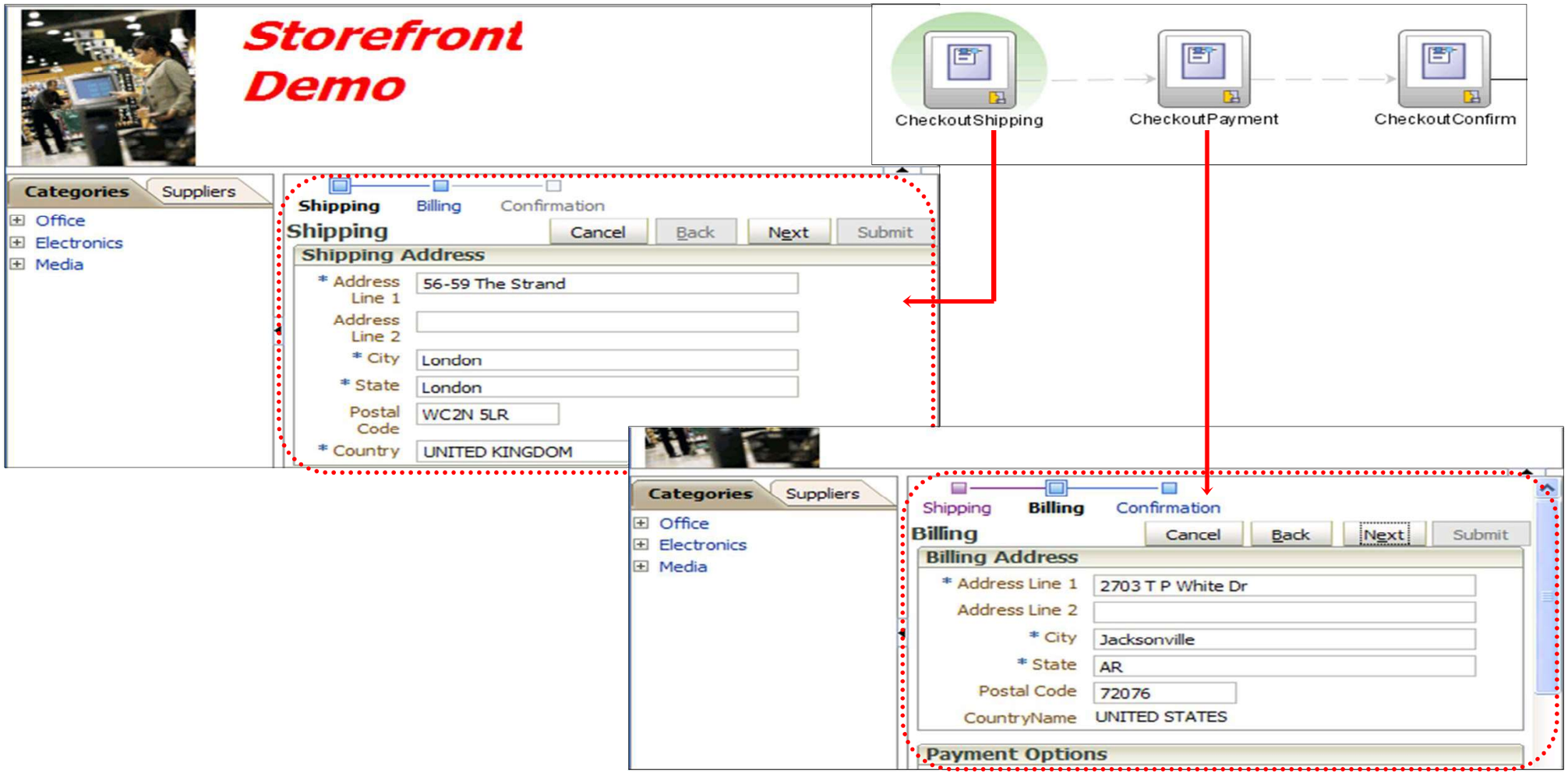
Why React is becoming Famous ?

1. One Way Binding
   - Binding Data to the UI Components.
   - Data Binding is Automatic Synchronization Between VIEW (HTML) and MODEL (Simple JS Variables).
   - MVVM (Model View View Model)
2. Structure front End Code
   - A Pure Separation of Concern
   - HTML will be Focusing on UI (VIEW)
   - JavaScript Will Be Focusing on Model

Cont …

3. Routing Support
   - Navigation Support
   - Focus is On SINGLE PAGE APPLICATION
   - Whole Application Will be Revolved Around A Single Page , But Navigation will Take Place not Between Pages but in turn Between Fragments.
   - Giving a Desktop Application Look and Feel.

Using a Bounded Task Flow



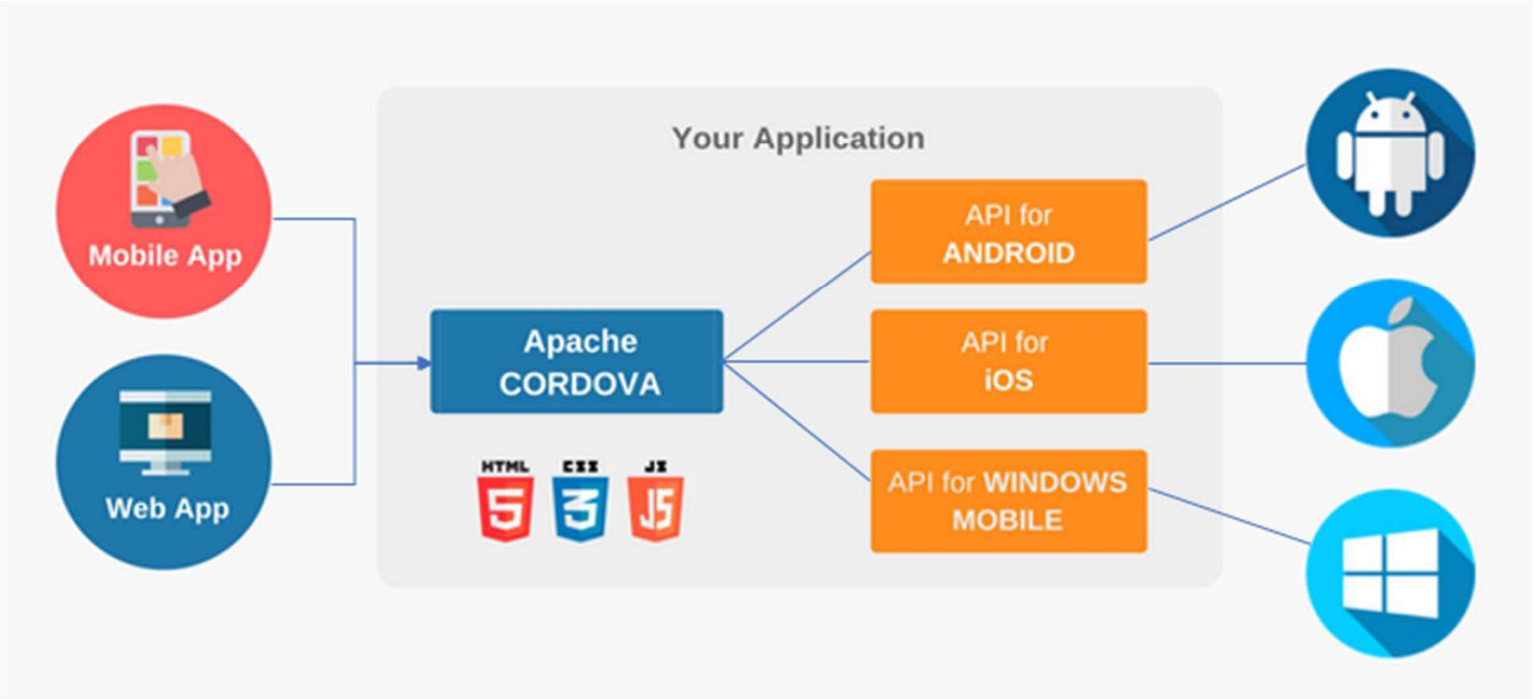
Cont …

4. Validation framework that provides UI element and component validation and data converters
5. Powered By React and There is a JS Community
6. Focus is on Developing Web Application Using Design Pattern
7. Messaging and event services for both Model and View layers
8. Caching services at the Model layer for performance optimization of pagination and virtual scrolling
9. Connection to data sources through Web services, such as Representational State Transfer (REST) or WebSocket
10. Support for Testing (Eg : KARMA)
11. Logging and Security Support

Hybrid Mobile Application Development Toolkit Features

- React includes support for hybrid mobile applications that run on iOS, Android, and Windows mobile devices within the Apache Cordova container.
- Apache Cordova enables you to use web technologies such as HTML5, CSS, and JavaScript to develop applications that you can deploy to mobile devices.
- Using the Cordova JavaScript APIs to access native device services, major mobile platforms such as Android, iOS, and Windows can be supported from a common code base.

Hybrid Mobile Application Development



1. JavaScript and HTML in the same file (JSX)

2. Embrace functional programming

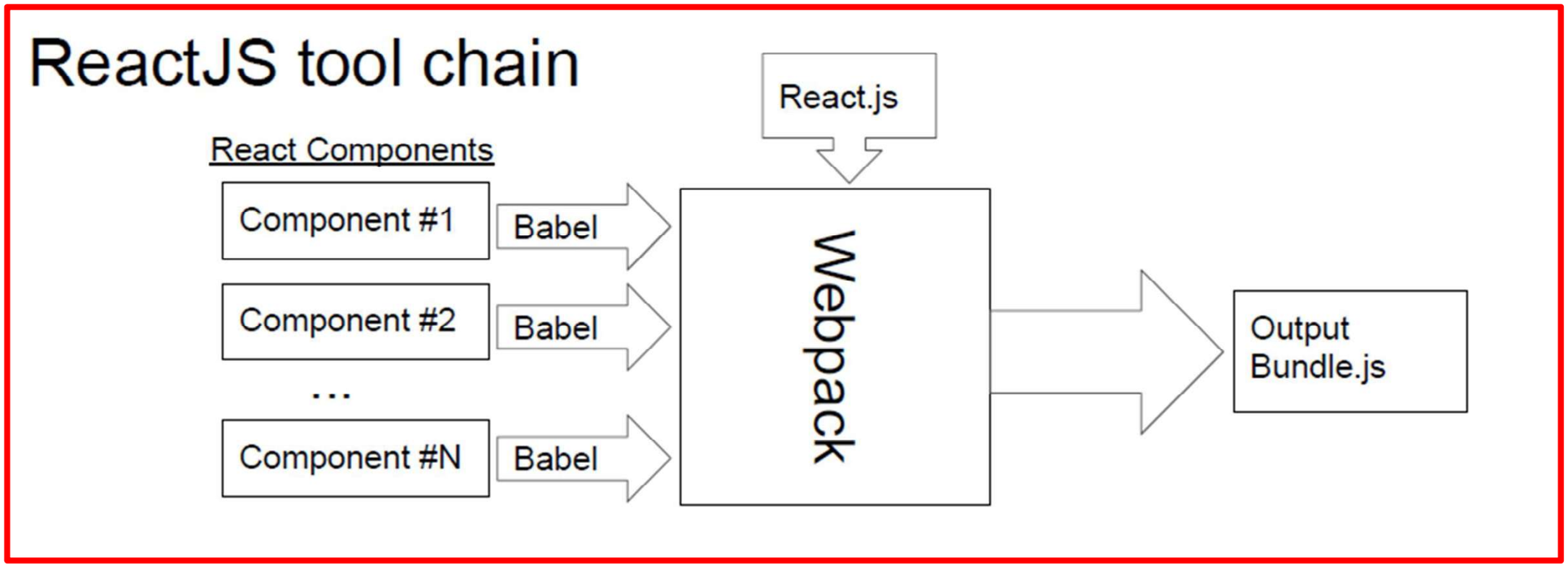
3. Components everywhere

ReactJS Web Application Page

```
<!doctype html>
<html>
   <head>
      <title>CS142 Example</title>
   </head>
   <body>
       <div id="reactapp"></div>
       <script src="./webpackOutput/reactApp.bundle.js"></script>
   </body>
</html>
```

ReactJS applications come as a **JavaScript blob** that will use the DOM interface to write the view into the div.

ReactJS tool chain



- Babel – Transpile Language Features (Eg: ECMAScript, JSX) to Basic JS
- Webpack – Bundle modules and Resources (CSS, Images)

JavaScript and HTML *in the same file*

Traditional approach

React approach

JSX: the React programming language

```
const first = "Aaron";
const last  = "Smith";

const name = <span>{first} {last}</span>;


const list = (
  <ul>
    <li>Dr. David Stotts</li>
    <li>{name}</li>
  </ul>
);
```

```
const listWithTitle = (
  <>
    <h1>COMP 523</h1>
    <ul>
      <li>Dr. David Stotts</li>
      <li>{name}</li>
    </ul>
  </>
);
```

"React is just JavaScript"

Functional Programming

1. Functions are “first class citizens”
2. Variables are immutable
3. Functions have no side effects

Functional programming

Functions are "first class citizens"

This means functions can be...

1. Saved as **variables**
2. Passed as **arguments**
3. **Returned** from functions

```
let add = function() {
  console.log('Now adding numbers');
  const five = 3 + 2;
};
```

```
function performTask(task) {
  task();
  console.log('Task performed!');
}

performTask(add);
```

```
function foo() {
  return function() {
    console.log('What gets printed?');
  };
}

foo
foo();
foo()();
```

Functional programming

Variables are immutable

Tip: Use **`const`** instead of **`let`** to declare variables!

```
let a = 4;

a = 2; // Mutates `a`
```

```
let b = [1, 2, 3];

b.push(4); // Mutates `b`

let c = [...b, 4]; // Does not mutate `b`
```

Functional programming

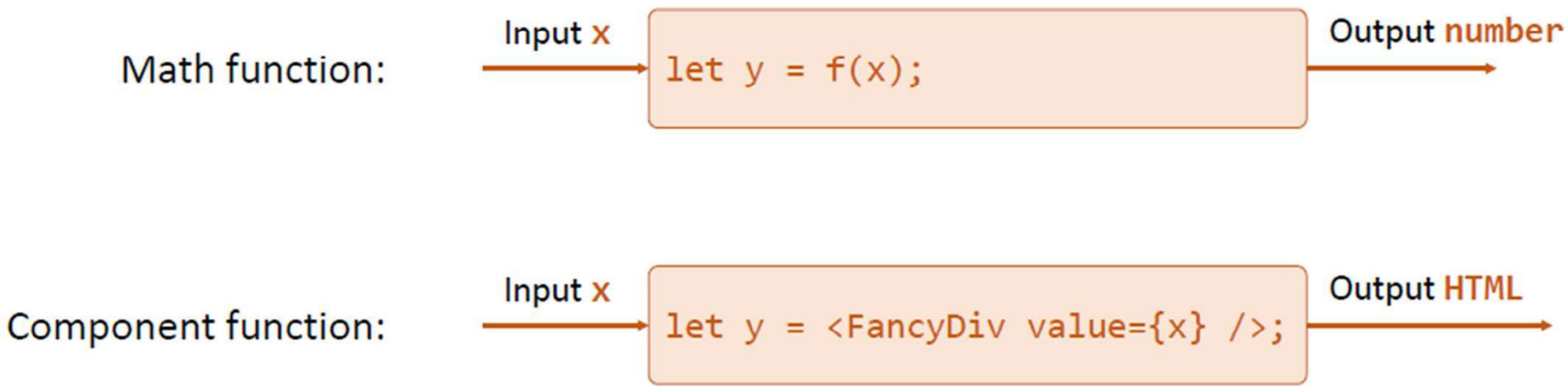
Functions have no side effects

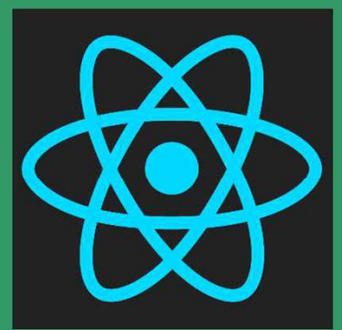
```
const b = [];

function hasSideEffects() {
  b = [0];
}
```

Components

Functions help break your code into small, reusable pieces

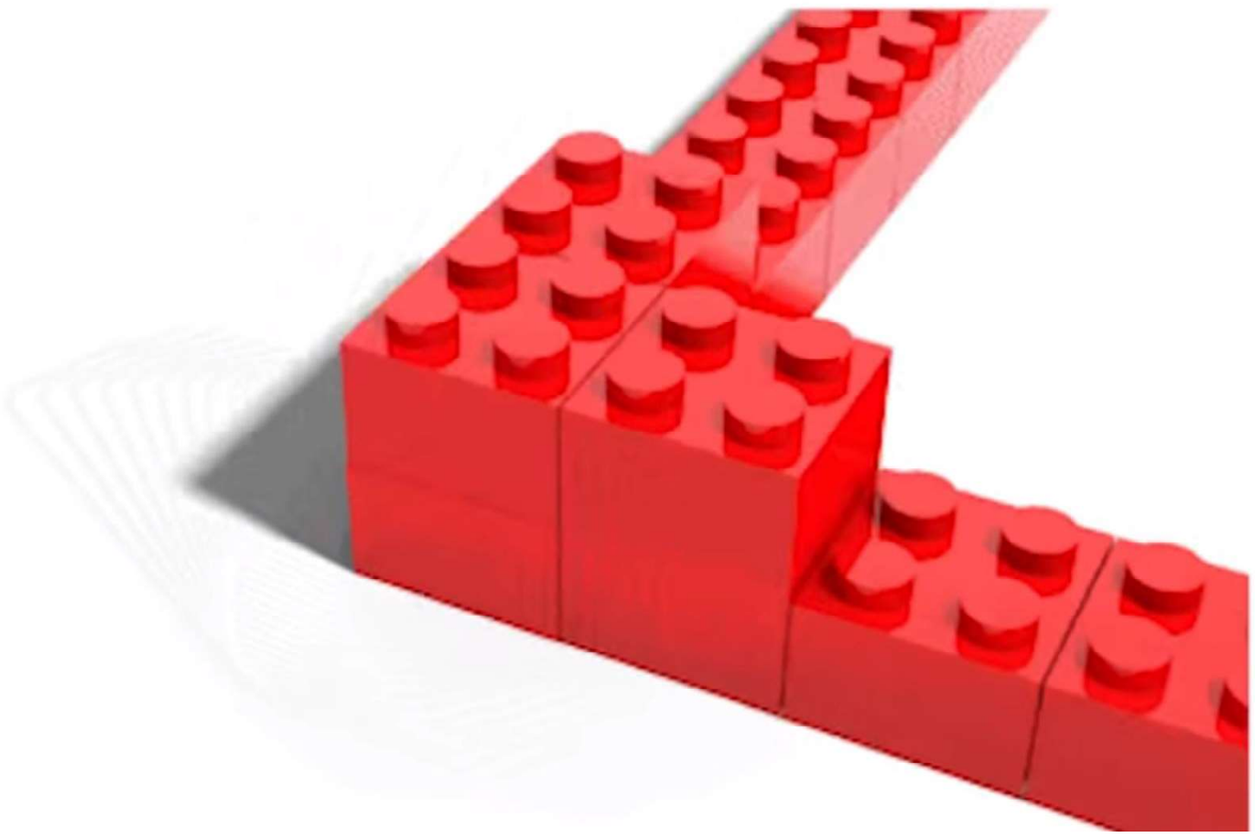
Components are functions for user interfaces





The Components

Angular Components are the Building Blocks Like



Cont ...

Traditional Developments

- HTML which is the static portion of your application and then you write your JavaScript which is the dynamic portion of the application.
- The JavaScript is kind of embedded into your HTML so when the HTML loads the JavaScript also gets loaded

Show Current Date and Time

This is how We Do in Traditional Application

Add a div and paragraph

Code to get date/time

Get the paragraph DOM element

Update value

How Do We Make it Dynamic

Add a div and paragraph

Add button

Code to get date/time

Get the paragraph DOM element

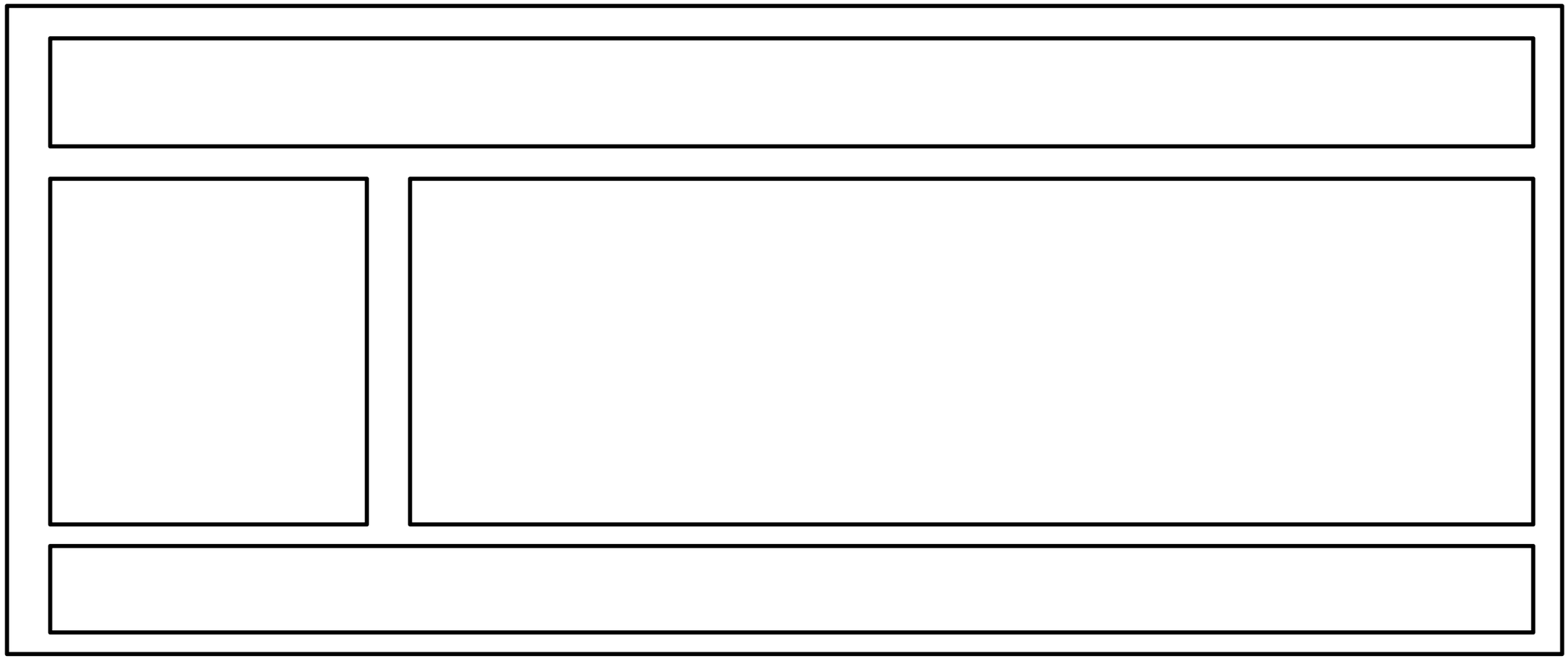
Update value

Code function to handle button click

Component Based Approach

In React

- In React applications we do not have this implicit divide at least in the mentioned module we don't have to think about it when we creating something
  - What is the HTML side
  - What is the JavaScript side
- In React We think about is components
- React has a component based approach to developing web applications



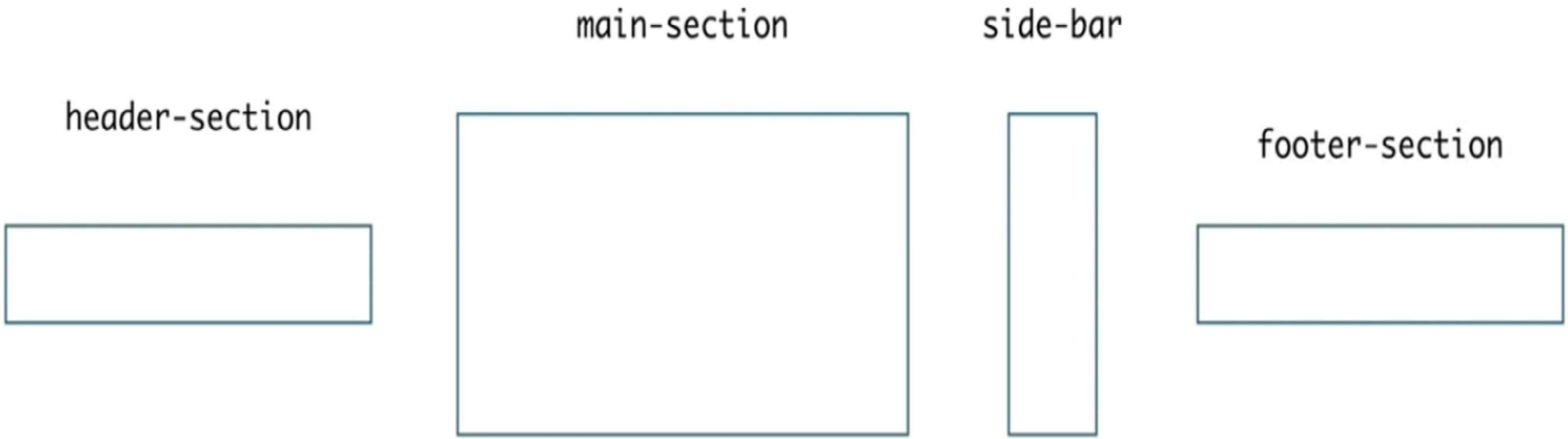
- So every significant portion of real estate on our page which can be self-sufficient which knows what to do with that area can be split and created as a component
- This is the approach we use in developing an React, Angular, OJET Application.
- We are going to create this entity which contains HTML and JavaScript together in it right so this is a self-sufficient piece of web application that can be plugged in somewhere else and that knows what to do.

- Creating a component assign a name to that component write assign a selector that selector is what a consumer can use to call and render that component

header-section

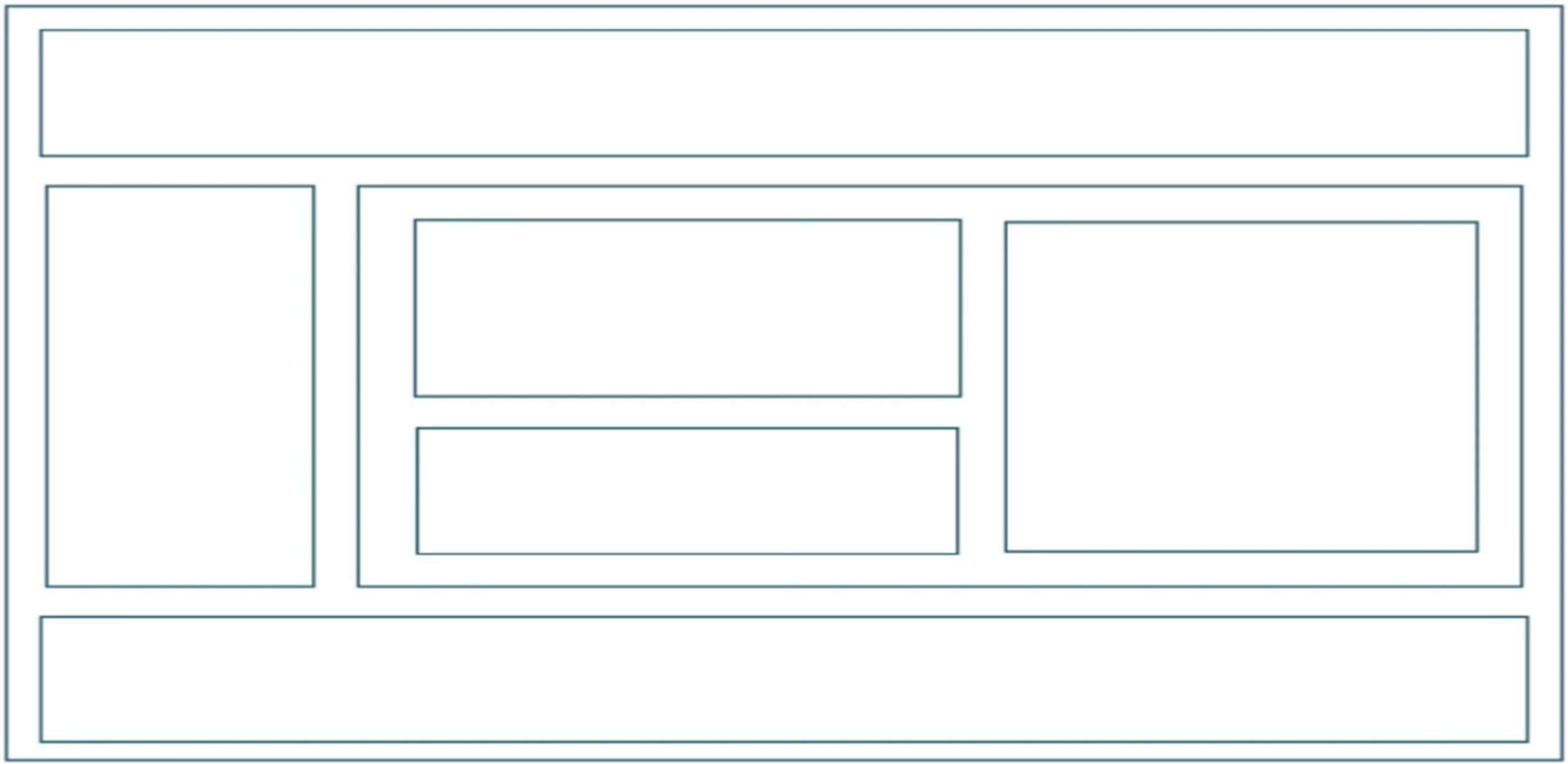
<header-section></header-section>

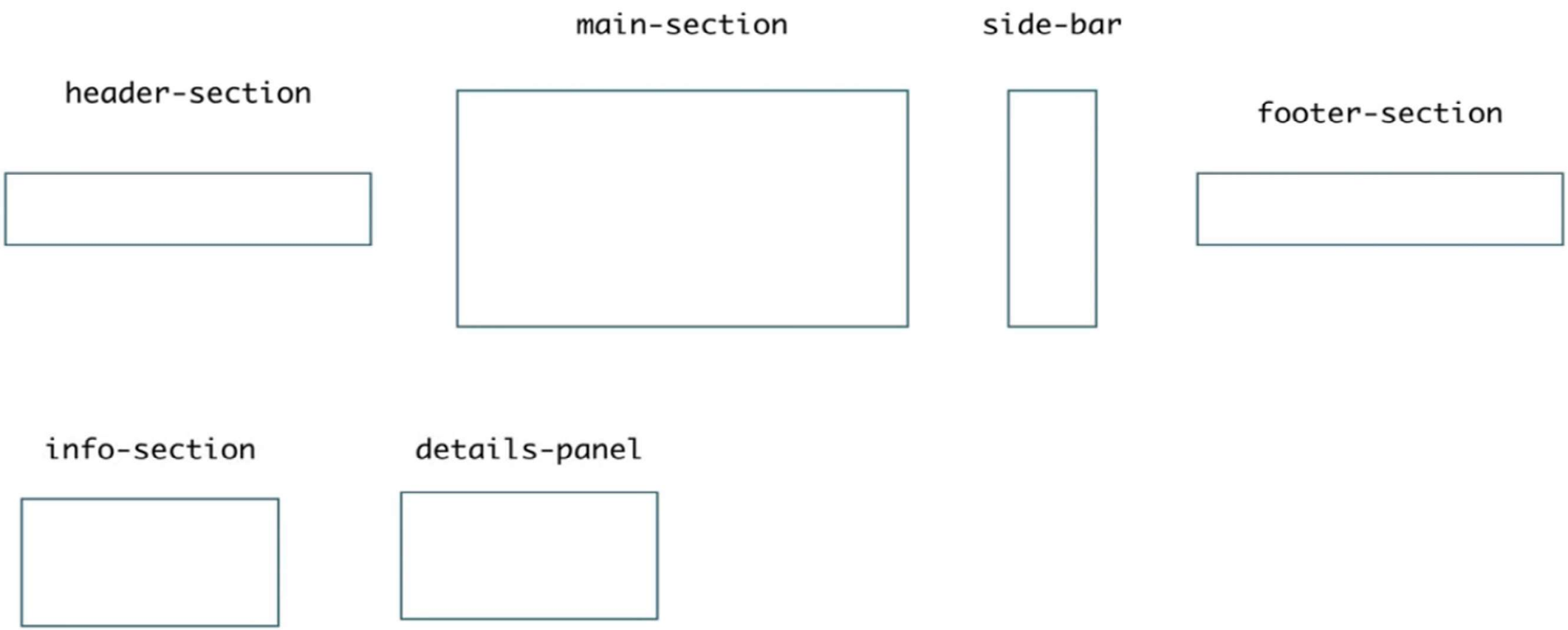
Component Based Development



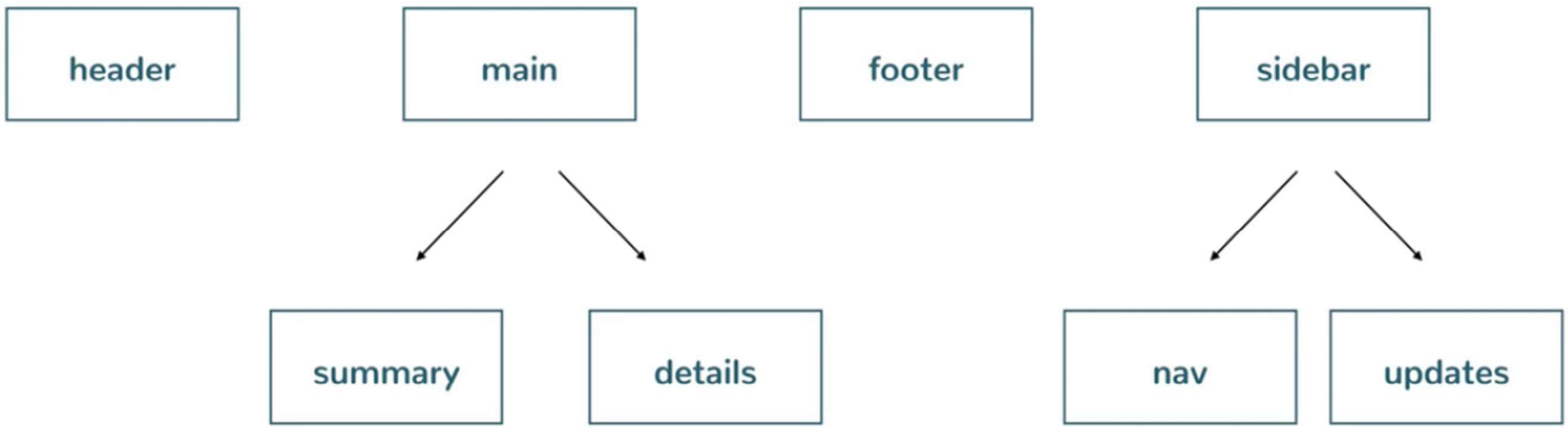
Components on a Page

```
<header-section></header-section>

<main-section></main-section>

<side-bar></side-bar>

<footer-section></footer-section>
```

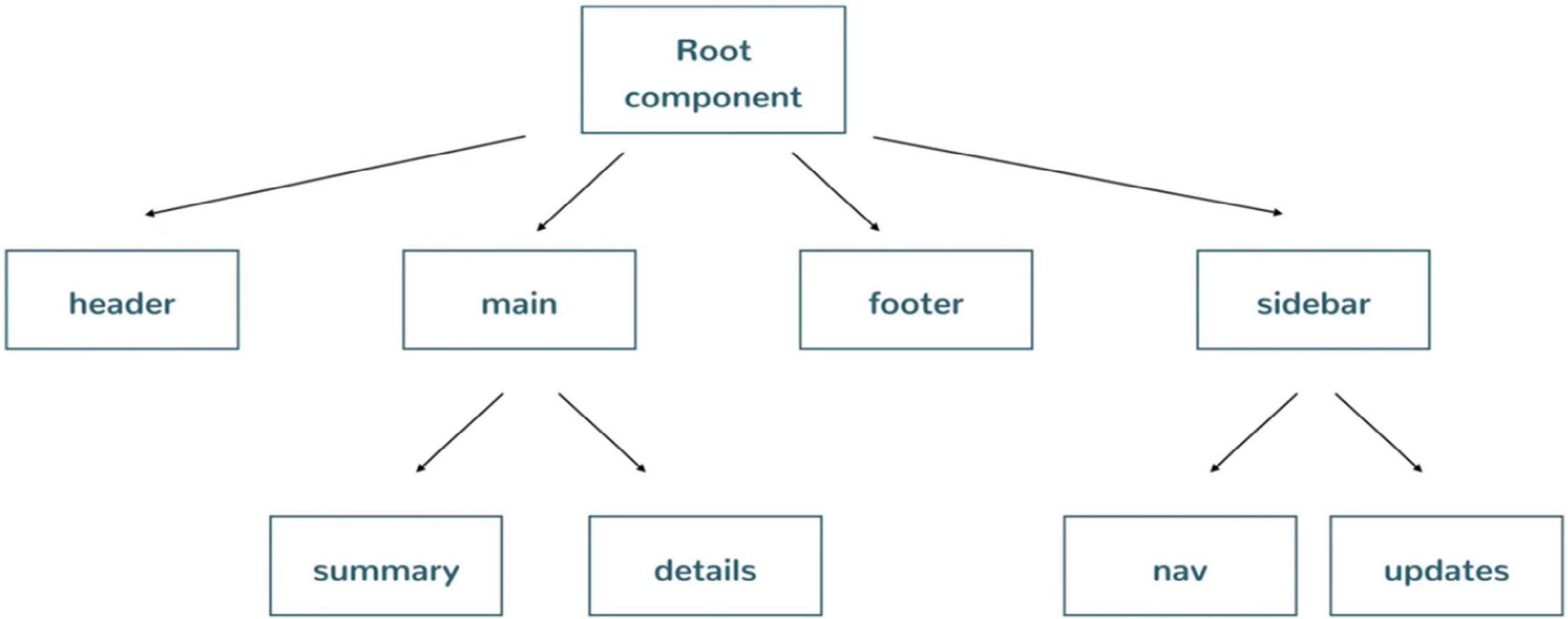



main-section
side-bar
header-section
footer-section
info-section
details-panel

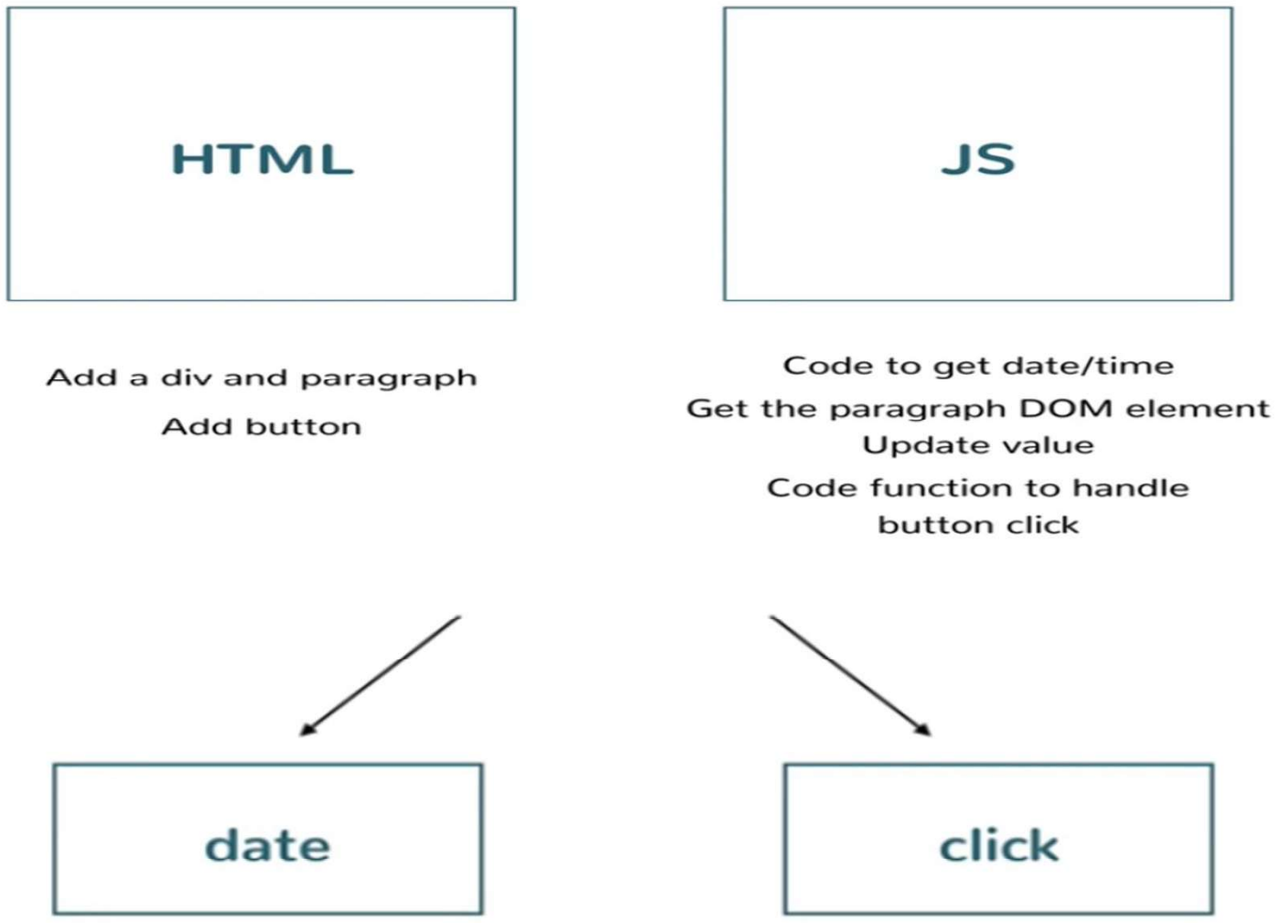
Component Tree Structure



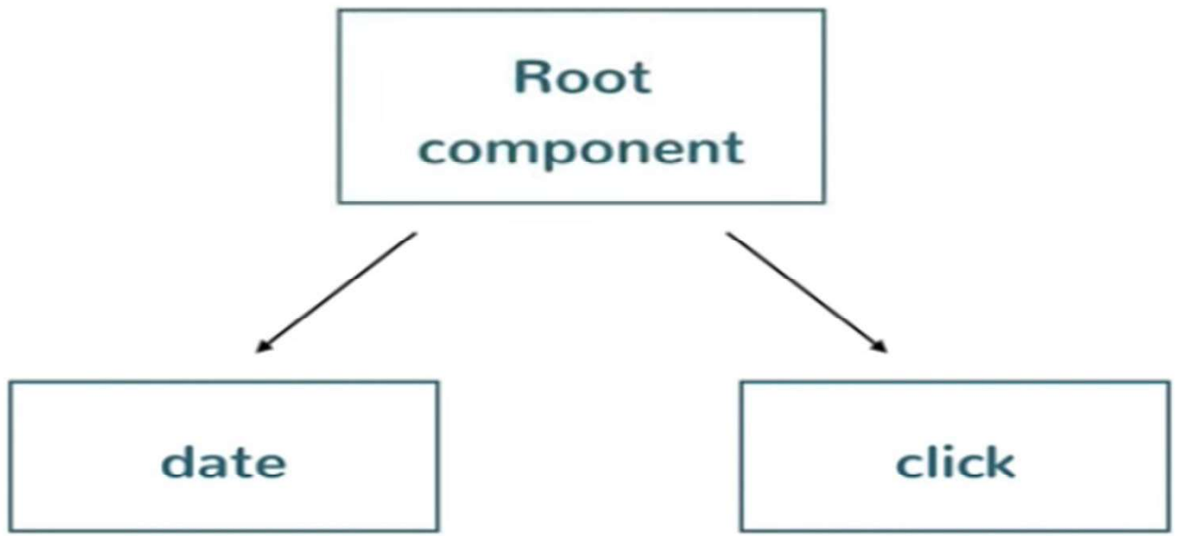
Every React Application as a Root Component Which Holds Other Comps



How it Alters Our Application

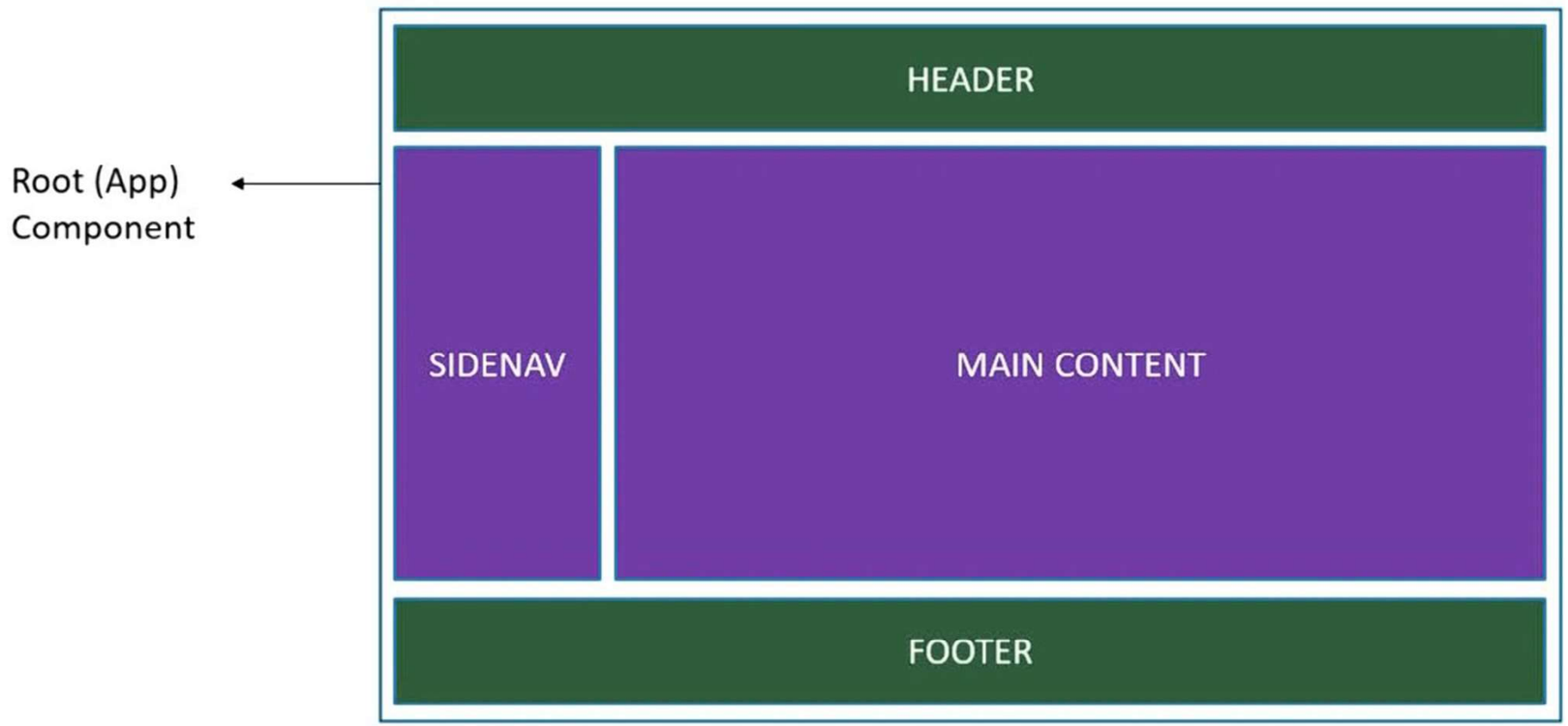


Finally Our React Application Looks Like this



"In React, everything is a component"

- In React Component Represents the Part of the UI
- All The Components Come together to make up the Entire Application



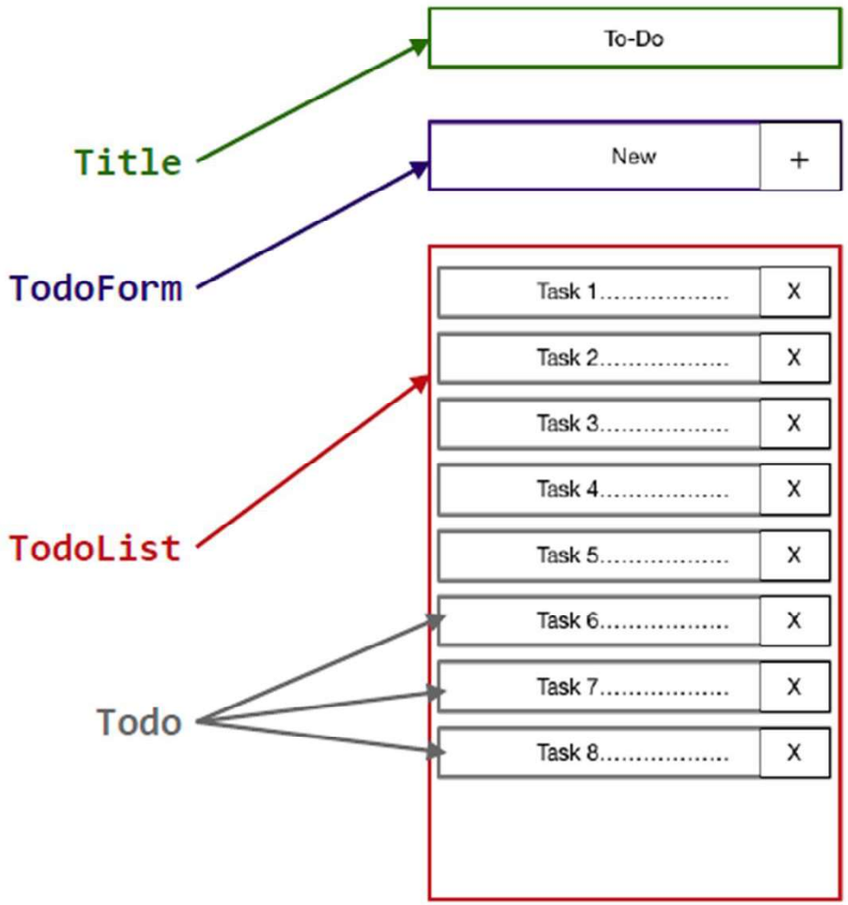
Todo application

Big idea:

- A digital to-do list

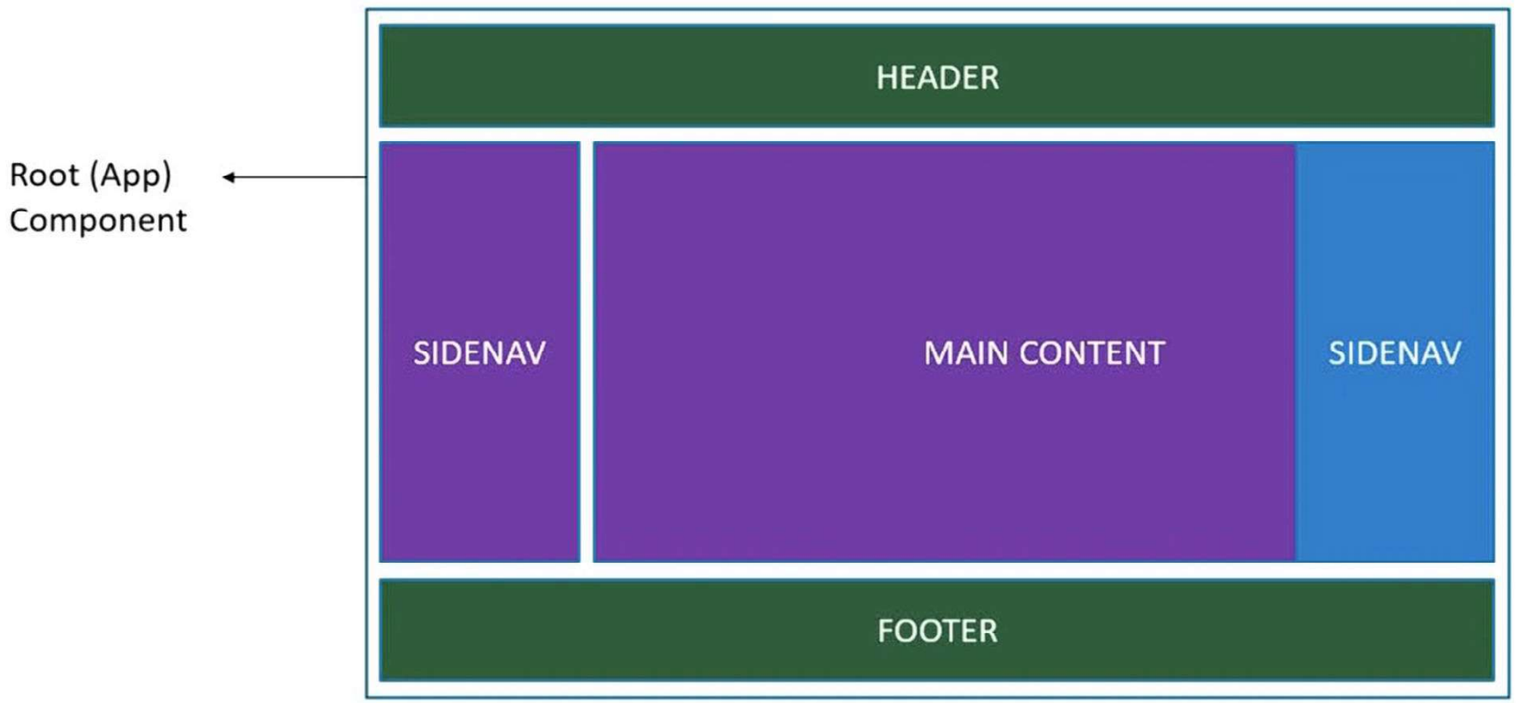
First step:

- mockup / wireframe

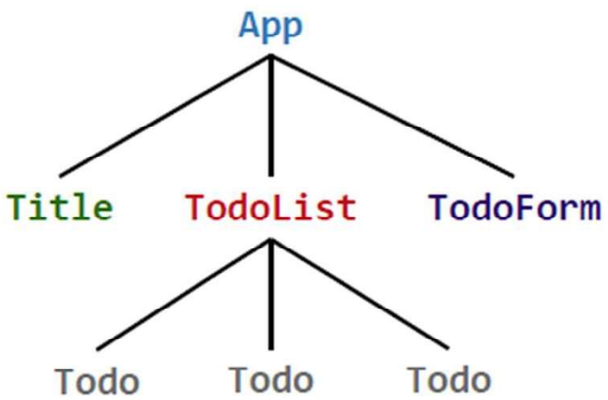


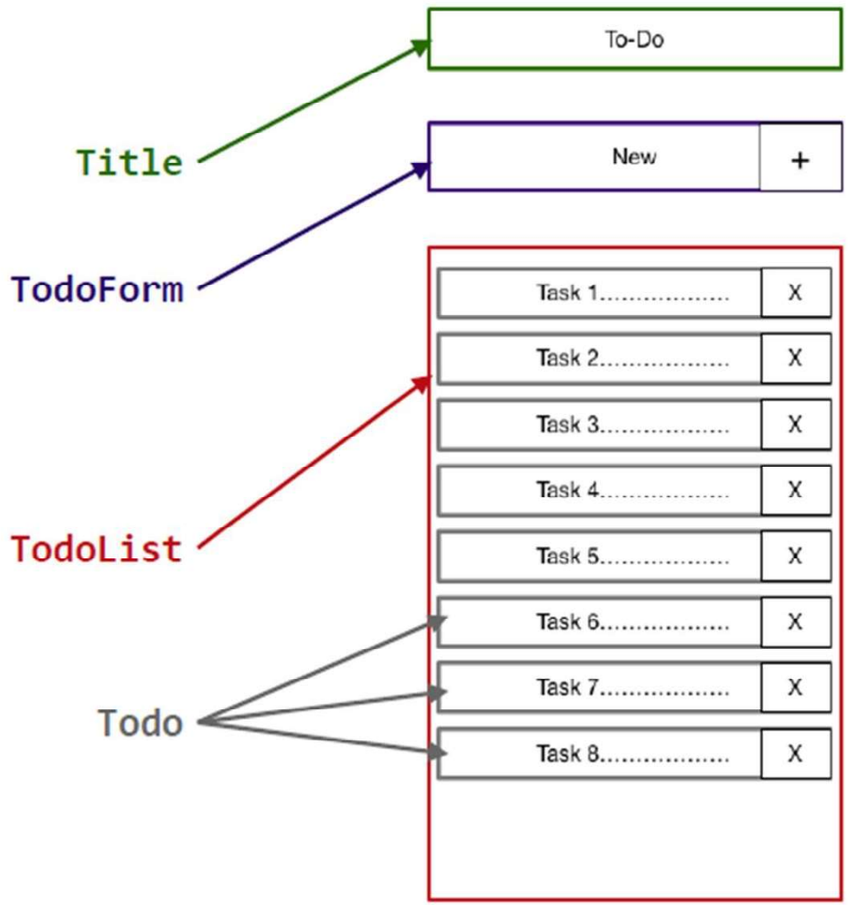
1. Components are Reusable
2. Components can be Nested
3. The Same Component can be used with different Properties to display different Information

   For Eg:

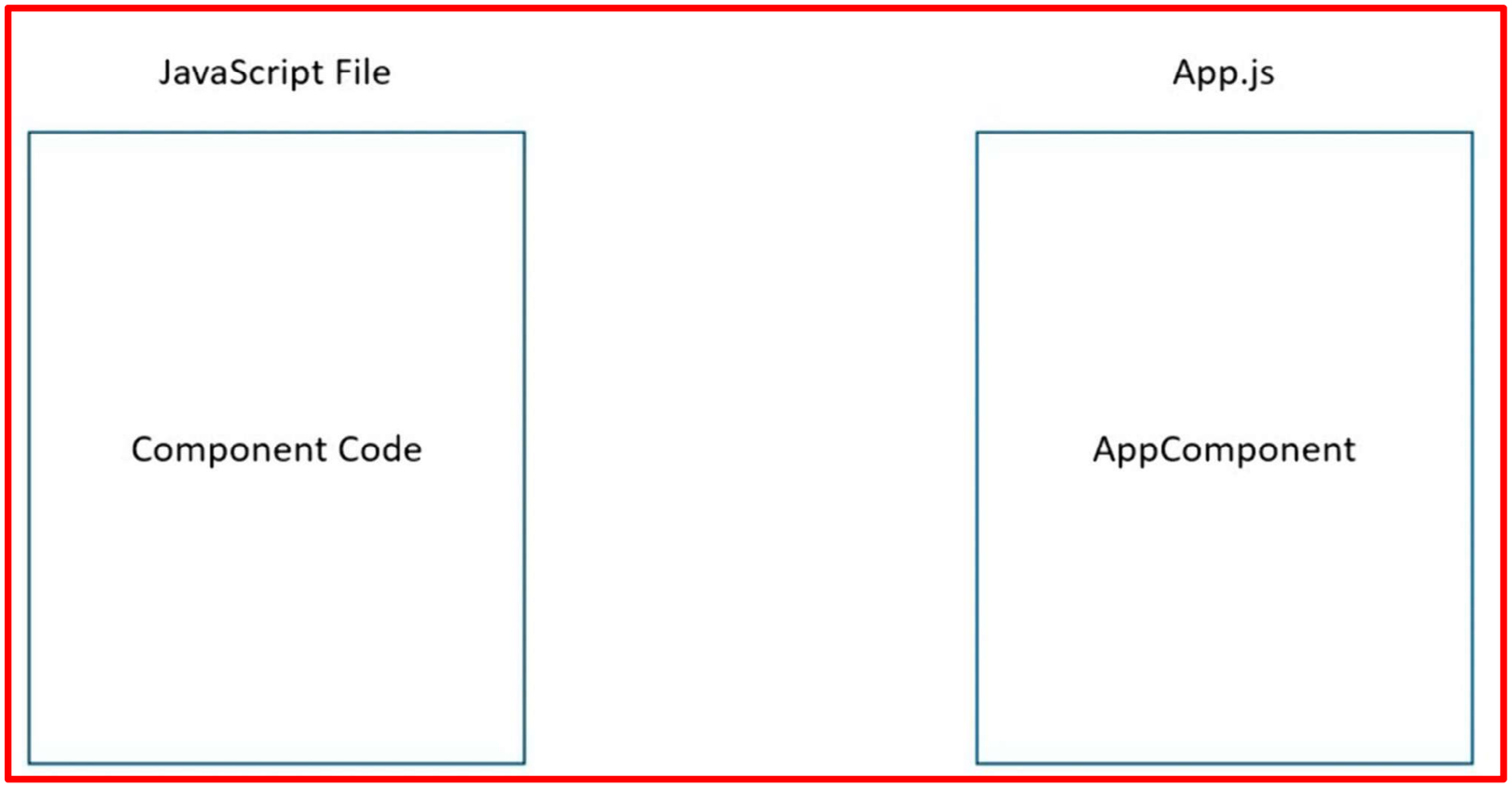


Component Hierarchy

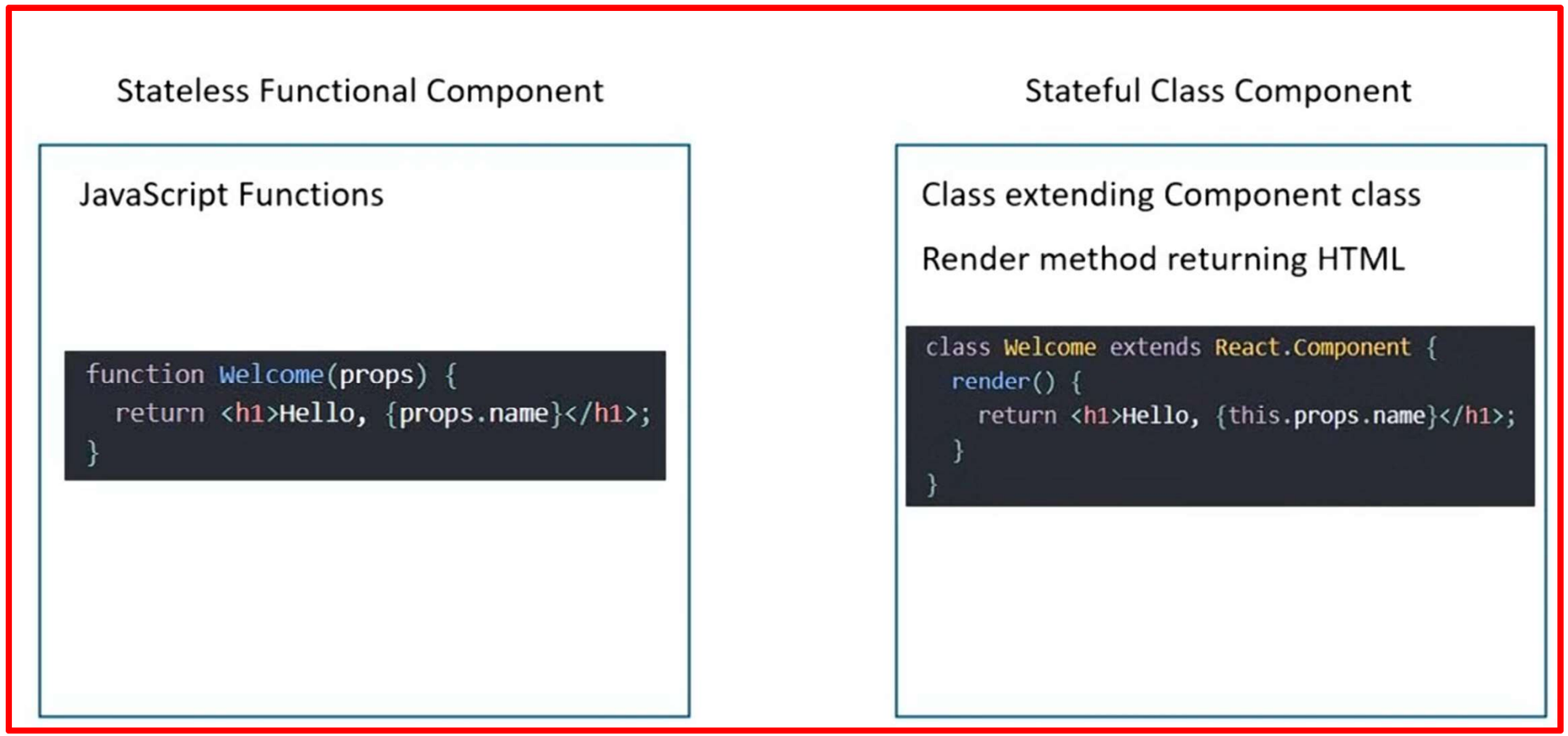




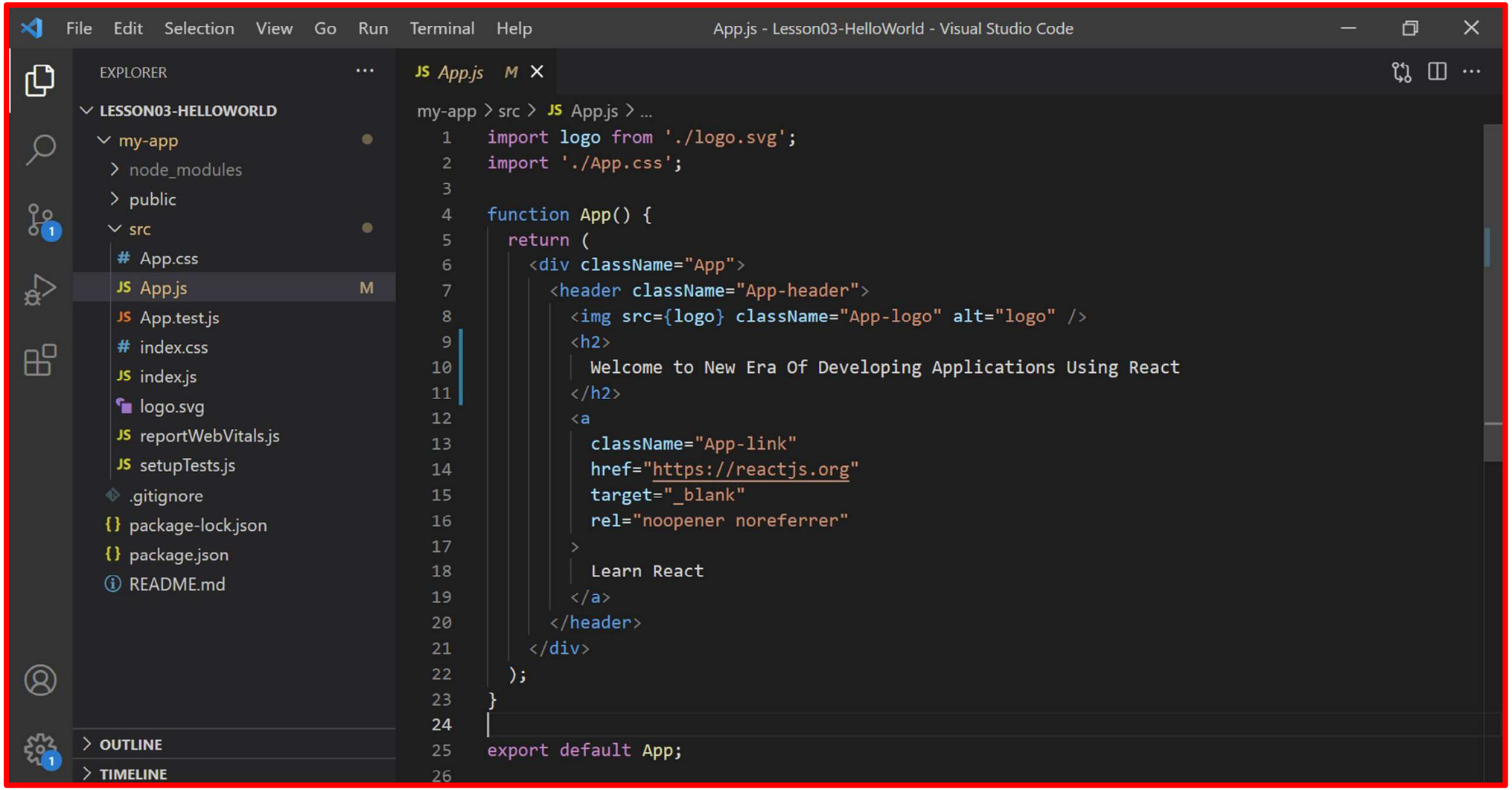
Component in Code



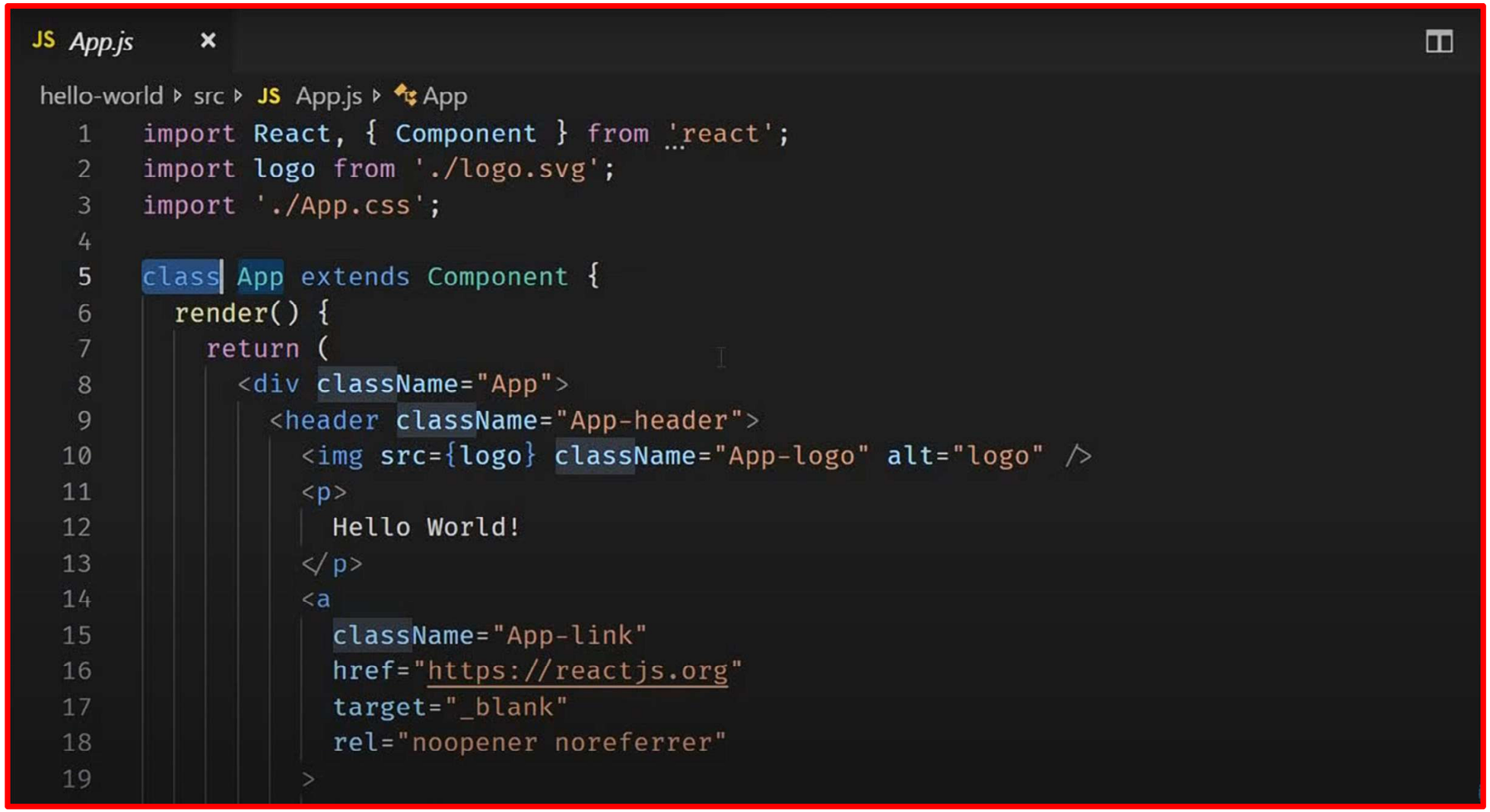
Component Types



App Component [Function based Component]



App Component [Class Based]



Eg :

- Facebook as over 30K+ Components.
- More Complex the Applications, more no of Components

Components Summary

- Components Describe a part of the UI
- They are Reusable and Can be Nested inside Other Components
- Two Types –
  - Stateless Functional Components
  - Stateful Class Components
- Components are the Building Blocks of any React Application

Anatomy of a React component

The component is just a function

Inputs are passed through a single argument called “props”

```
export default function MyComponent(props) {
  return <div>Hello, world! My name is {props.name}</div>;
}

const html = <MyComponent name="aaron" />;
```

The function outputs HTML

The function is executed as if it was an HTML tag

Parameters are passed in as HTML attributes

Component rendering

- When a component function **executes**, we say it "**renders**"
- Assume components may re-render at any time

Our job is to ensure that
every time the component re-renders,
the correct output is produced