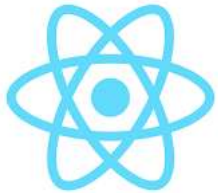
Introduction to ReactJS



What is React.js?

- ReactJS is an open-source JavaScript library for building user interfaces.
- ReactJS is a client-side JavaScript library.
- React makes building complex, interactive and reactive user interfaces simpler.
- It allows to create reusable UI components.
- It uses Virtual DOM, which is faster than real DOM.
- React was created by Facebook.
- Mobile apps and desktop apps feel very “reactive”: Things happen instantly. You don’t wait for new pages to load or actions to start.
- It is currently one of the most popular JavaScript libraries and has a strong foundation and large community behind it.

ACADEMIND

What is React.js?

React.js

A client-side JavaScript library

All about building modern, reactive user interfaces for the web

Declarative, component-focused approach

Why do we need React.js

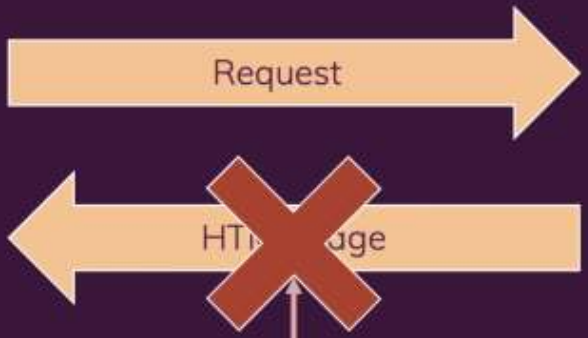
- Traditionally, in web apps, you click a link and wait for a new page to load. You click a button and wait for some action to complete.

JavaScript to the Rescue!

- JavaScript runs in the browser – on the loaded page.
- You can manipulate the HTML structure (DOM) of the page.
- No (visible) request to the server required, no need to wait for a new HTML page as a response.

ACADEMIND

JavaScript To The Rescue!



JS

JavaScript runs in the browser – on the loaded page.

You can manipulate the HTML structure (DOM) of the page.

No (visible) request to the server required, no need to wait for a new HTML page as a response.

Why React.js?

Simplicity

The component-based approach, well-defined lifecycle, and use of just plain JavaScript makes React very simple to learn, build an application and support it. React uses a special syntax called JSX which allows you to mix HTML with JavaScript.

Easy to Learn

Anyone with a basic previous knowledge in programming can easily understand React

Data binding

React uses one-way data binding and an application architecture called Redux which controls the flow of data to components through one control point.

Performance

React improves performance of application due to Virtual DOM.

Testability

ReactJS applications are super easy to test. You can test React components similar to testing other JavaScript code.

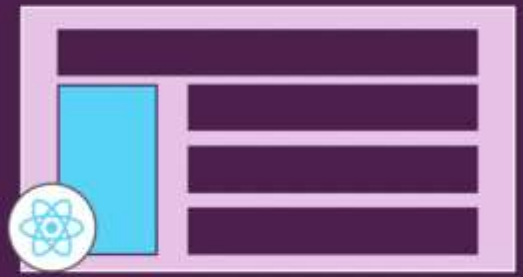
Building Single-Page-Applications (SPAs)

- React can be used to control parts of HTML pages or entire pages.
- “Widget” approach on a multi-page-application.
- (Some) pages are still rendered on and served by a backend server.
- React can also be used to control the entire frontend of a web application.
- “Single-Page-Application” (SPA) approach.
- Server only sends one HTML page, thereafter, React takes over and controls the UI.

ACADEMIND

Building Single-Page-Applications (SPAs)

React can be used to **control parts** of HTML pages or entire pages.

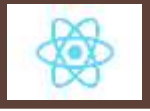


"**Widget**" approach on a multi-page-application.
(Some) pages are still **rendered on and served by a backend server**.

React can also be used to **control the entire frontend** of a web application

"Single-Page-Application" (**SPA**) approach. Server **only sends one HTML page**, thereafter, React takes over and controls the UI.

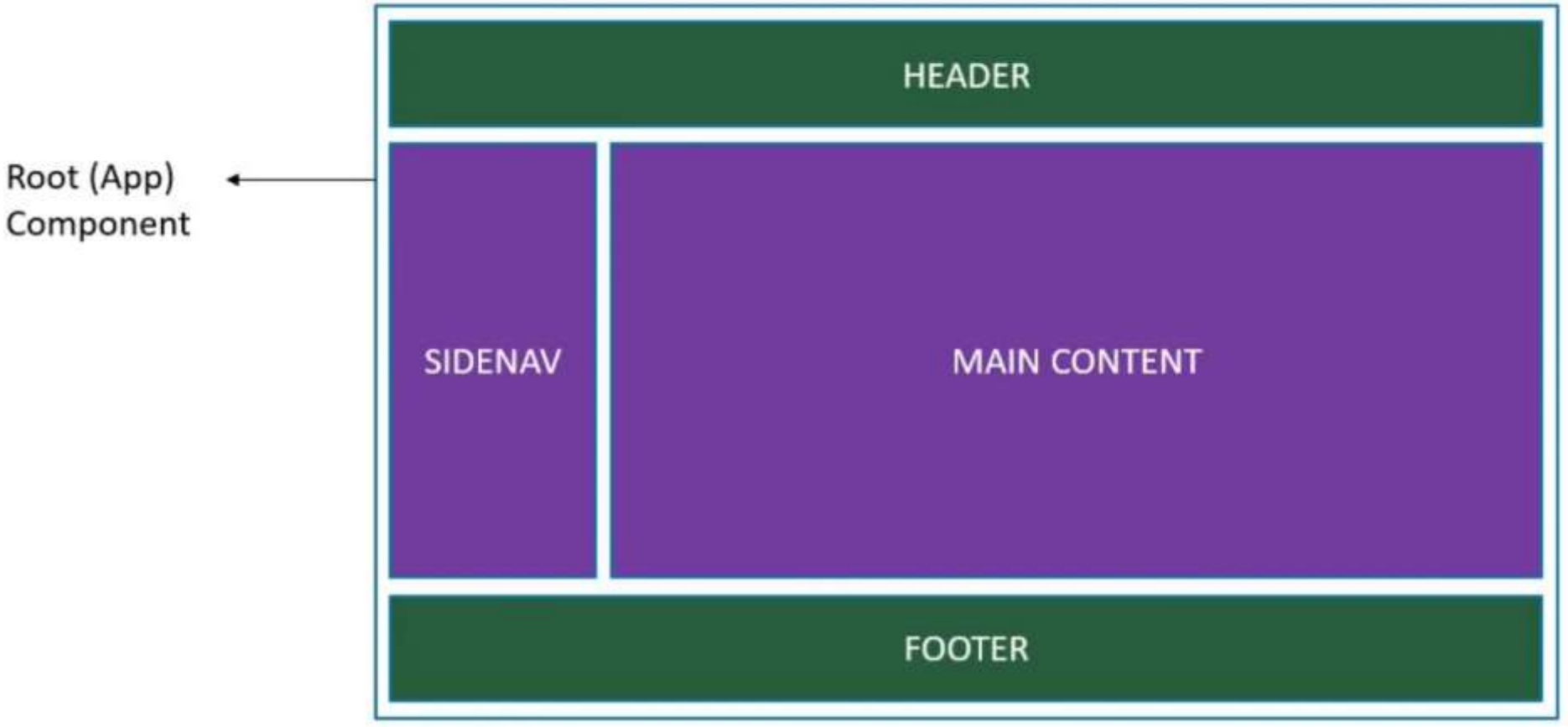
React.js Alternatives

React.js 	Angular 	Vue.js 
Lean and focussed component-based UI library.	Complete component-based UI framework, packed with features.	Complete component-based UI framework, includes most core features.
Certain features (e.g., routing) are added via community packages.	Uses TypeScript. Can be overkill for small projects.	A bit less popular than Angular and React.
Developed by Facebook in the year 2013	Developed by Google in the year 2009	Developed by Google in the year 2014
Library	Framework	Framework
One-way data binding	Two-way data binding	Two-way data binding

Component-Driven User Interfaces

- React uses component based architecture for building interactive and scalable UIs.
- Visualize the web page as a set of components.
- React is all about “Components” because all user interfaces in the end are made up of components.
- Component is a reusable building blocks in our UI.
- Components describe a part of the user interface.
- They are reusable and can be nested inside other components.
- Components are of two types –
  - Stateless functional components
  - Stateful class components

Components



Why Components?

Reusability

Don't repeat yourself

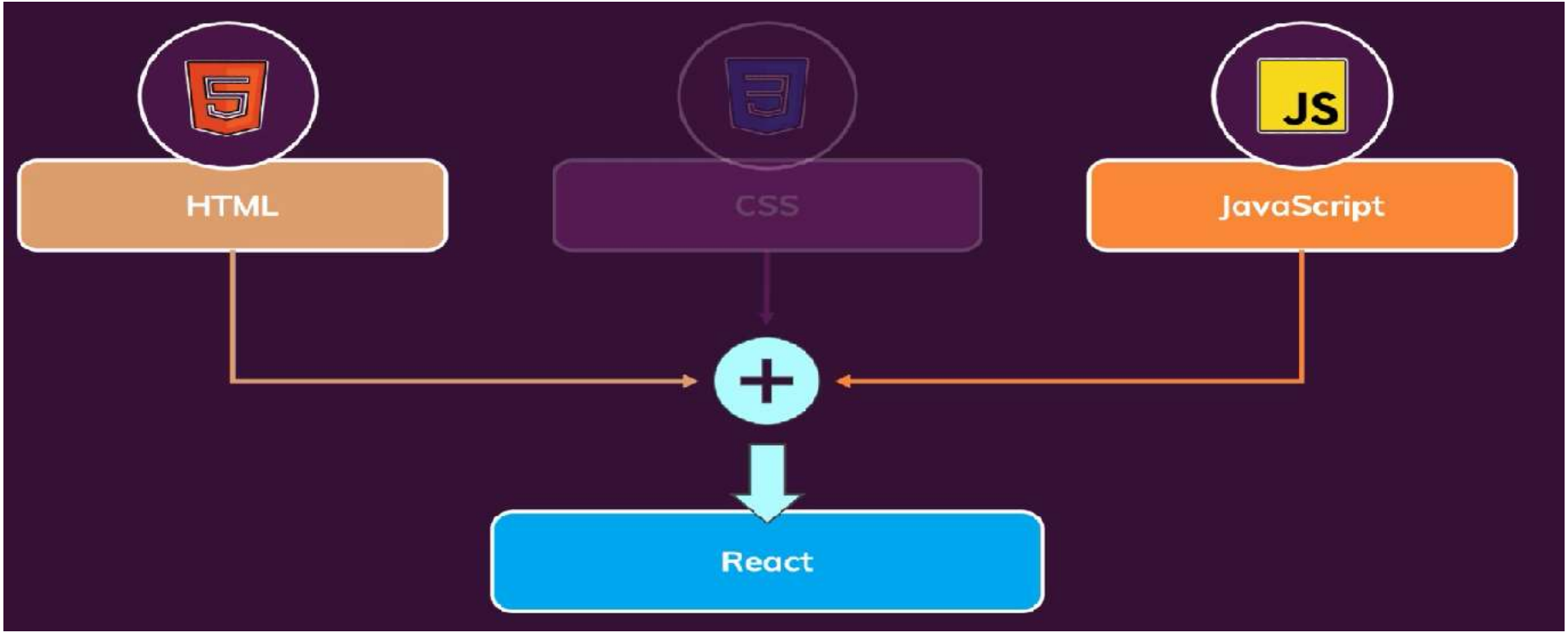
Separation of Concerns

Don't do too many things in one and the same place (function)

Split big chunks of code into multiple smaller functions

How is a component Built?

React allows you to create re-usable and reactive components consisting of HTML and JavaScript (and CSS).



React & Components

React allows you to create **re-usable and reactive components** consisting of **HTML and JavaScript** (and CSS)

Declarative Approach

Define the desired target state(s) and let React figure out the actual JavaScript DOM instructions

Who uses React?

COMPANIES

7829 companies reportedly use **React** in their tech stacks, including **Airbnb**, **Uber**, and **Facebook**.


Airbnb

Uber


Facebook

Netflix


Instagram


medium.com


Pinterest


Twitter

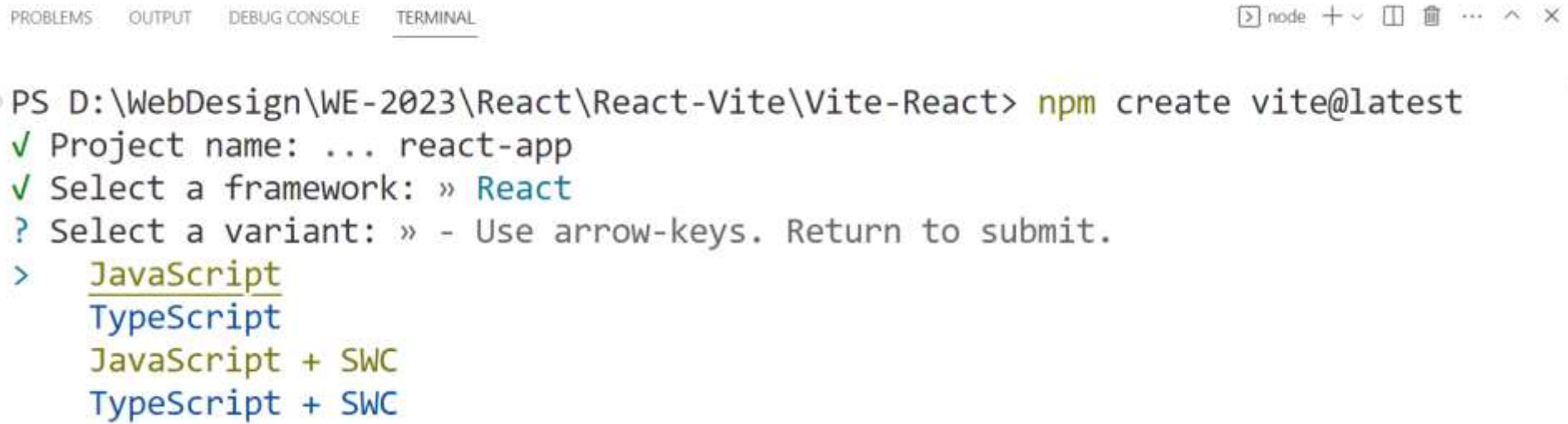

reddit

Creating a React app

```
npx create-react-app my-app
cd my-app
npm start
```

```
npm create vite@latest
cd react-app
npm run dev
```


PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
node
PS D:\WebDesign\WE-2023\React\React-Vite\React-Vite> npm create vite@latest
? Project name: » vite-project
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
node
PS D:\WebDesign\WE-2023\React\React-Vite\React-Vite> npm create vite@latest
? Project name: » react-app
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
node
√ Project name: ... react-app
? Select a framework: » - Use arrow-keys. Return to submit.
Vanilla
Vue
> React
Preact
Lit
Svelte
Others


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
node
PS D:\WebDesign\WE-2023\React\React-Vite\Vite-React> npm create vite@latest
√ Project name: ... react-app
√ Select a framework: » React
? Select a variant: » - Use arrow-keys. Return to submit.
> JavaScript
TypeScript
JavaScript + SWC
TypeScript + SWC

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL powershell

```
PS D:\WebDesign\WE-2023\React\React-Vite\React-Vite> npm create vite@latest
√ Project name: ... react-app
√ Select a framework: » React
Scaffolding project in D:\WebDesign\WE-2023\React\React-Vite\React-Vite\react-app...

Done. Now run:

  cd react-app
  npm install
  npm run dev
```

Functional vs Class components

Functional

Simple functions

Use Func components as much as possible

Absence of 'this' keyword

Solution without using state

Mainly responsible for the UI

Stateless/ Dumb/ Presentational

Class

More feature rich

Maintain their own private data - state

Complex UI logic

Provide lifecycle hooks

Stateful/ Smart/ Container

JSX

JSX

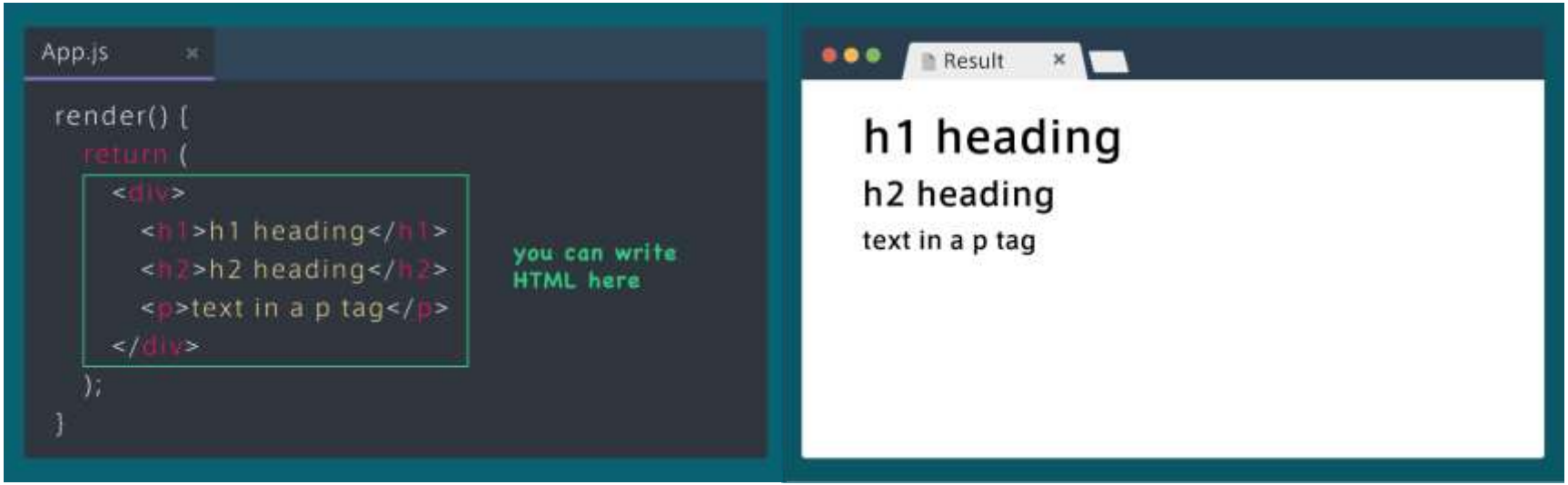
- JSX stands for JavaScript XML.
- JSX allows us to write HTML in React.
- JSX makes it easier to write and add HTML in React.
- JSX is not valid JavaScript that browsers can read.
- It’s a JavaScript file that contains JSX code, very similar to HTML.
- JSX files must be complied before they reach the web browser, with a JSX compiler that translates the JSX into JavaScript.

JSX

- Each JSX expression must have exactly one outermost element. Usually wrap the JSX expression in a <*div*></*div*> or a fragment <></>.
- In HTML, it's common to use the class as an attribute name, but in JSX, we cannot use the word class. We have to use className instead.
- In HTML, it's optional to include a forward slash for self-closing tags like br, img, input, etc., but not JSX.
- Event listeners same as HTML
- It is not mandatory to write JSX, but JSX makes it easier to write React applications (recommended).

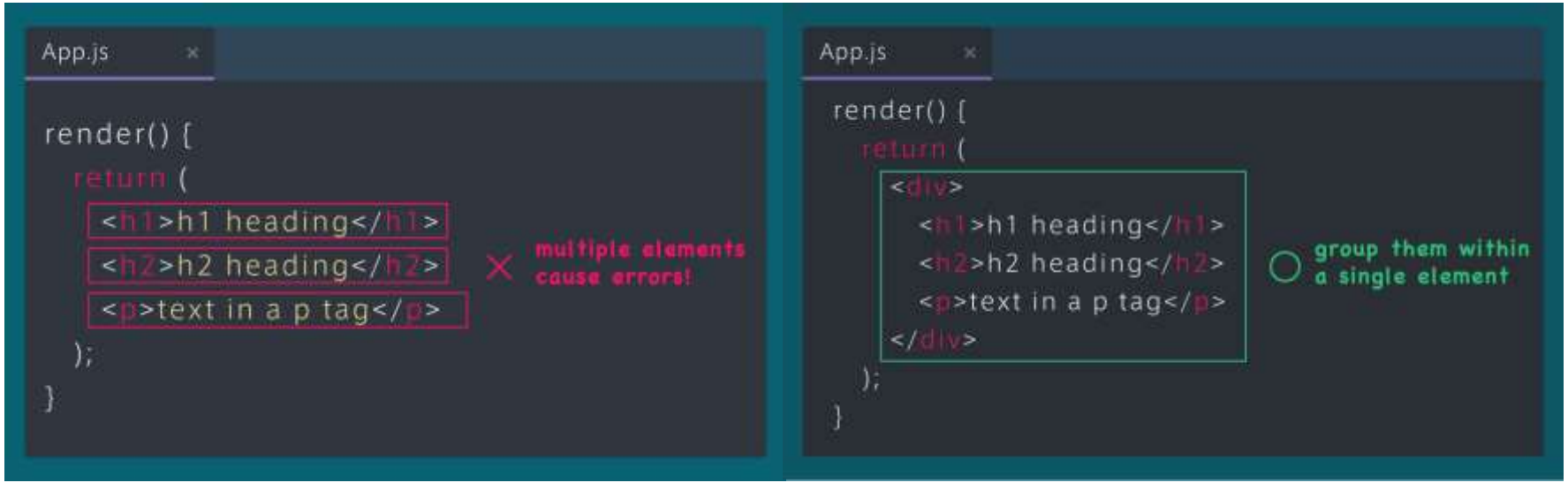
Writing JSX

Writing JSX is almost the same way as writing HTML. You can use many of the exact same tags as HTML, like <h1> and <h2> for headings or <p> for paragraphs, and for sections and containers.



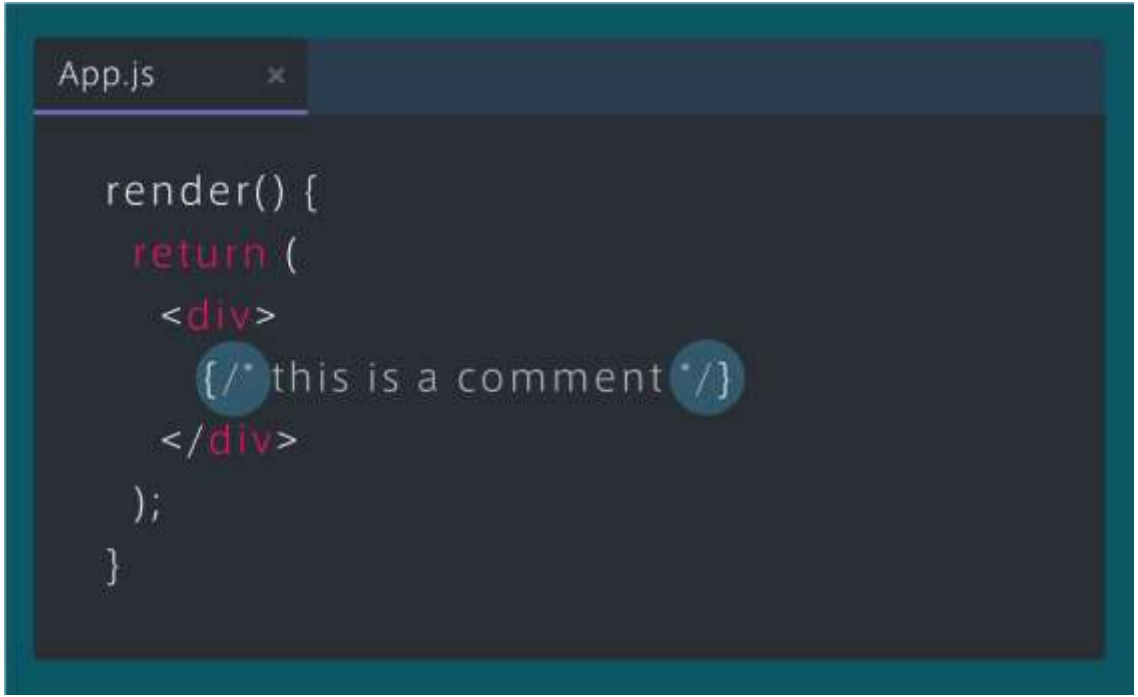
Common Mistakes with JSX

Although JSX is very similar to HTML, there are a few differences. As shown on the left, multiple elements cannot be put inside return. When you have multiple elements, put them in a single tag like in the example on the right.



When JSX is put between the symbols {/* and */}, the text inside becomes a comment. Comments will not be displayed in the browser and they do not change the result of your code. Most text editors will show comments in gray text.

Comments in JSX



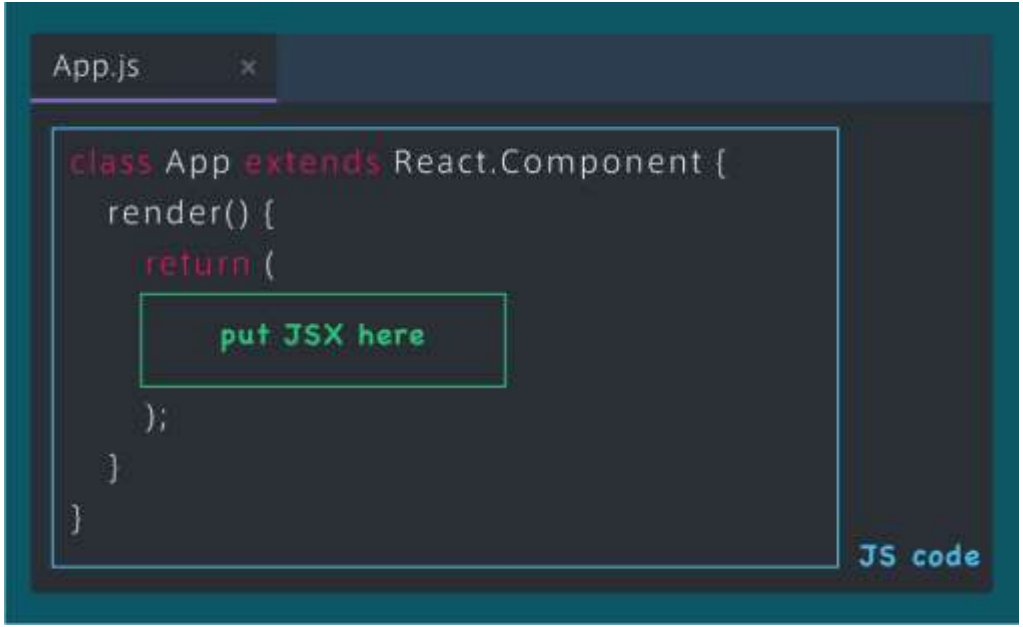
Img Tag

In HTML, the img tag did not have a closing tag like <img src='imageURL'>. However, a closing tag is necessary in JSX. Like , a slash / is written at the end of the tag.

JSX and JS

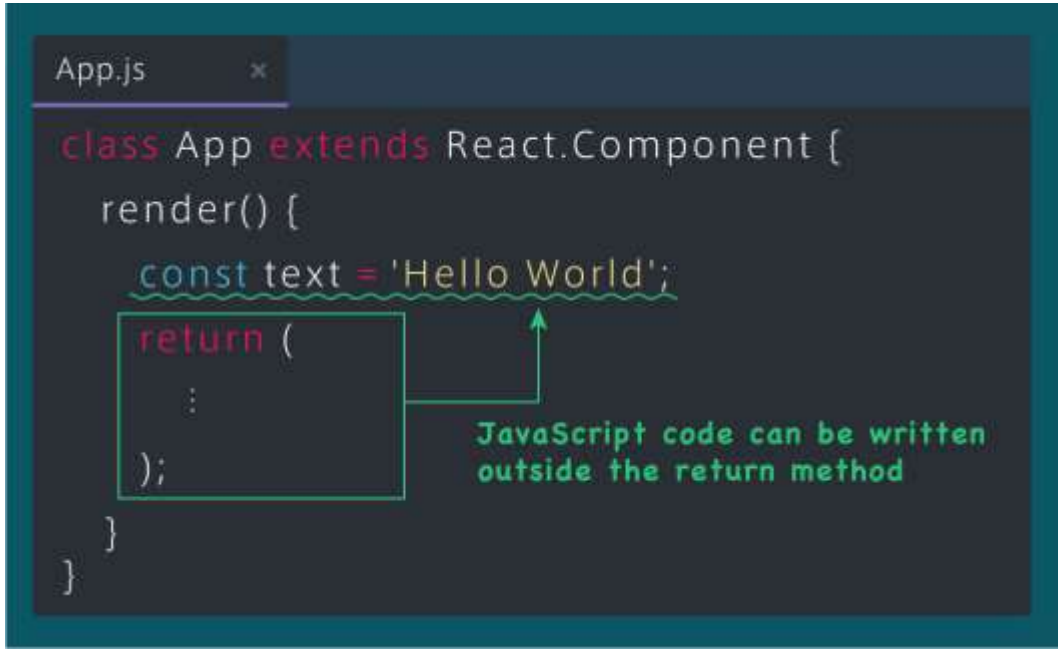
The sections to write JSX and JavaScript are clearly divided. As you can see below, the areas of JSX and JS (JavaScript) are clearly divided. JSX is only written inside the return section. The JSX elements will be displayed in the browser.

Sections for JSX and JS



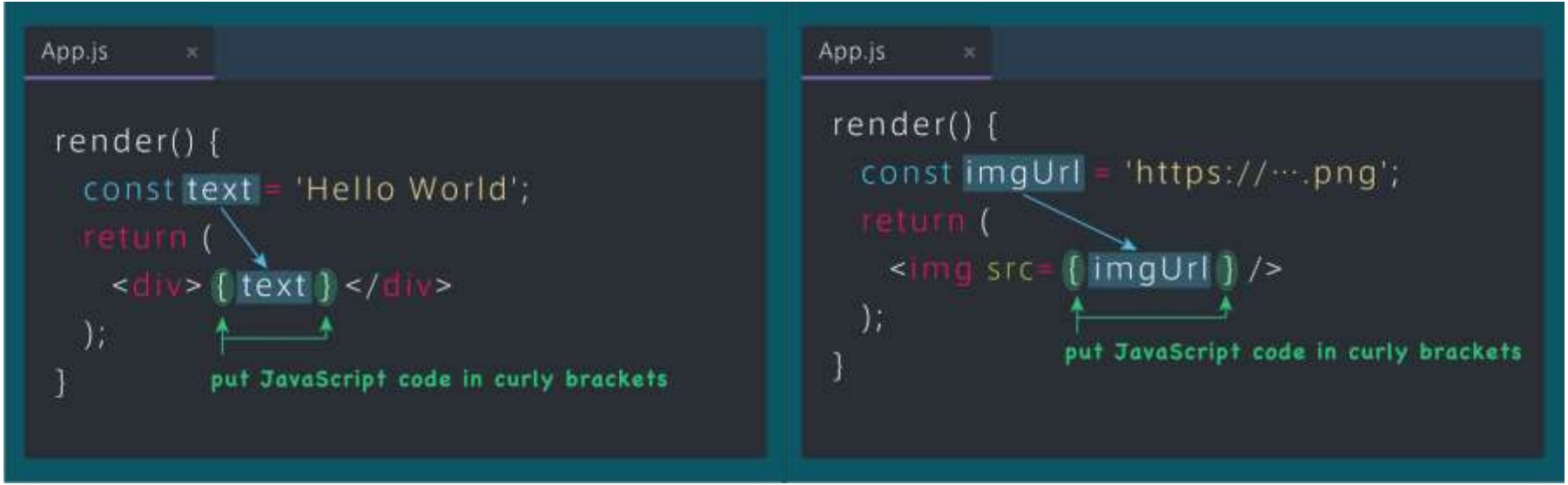
JavaScript can be written outside the return section. In the example below, the text constant is defined with JavaScript code. This is possible because it's outside the return section.

Writing JavaScript



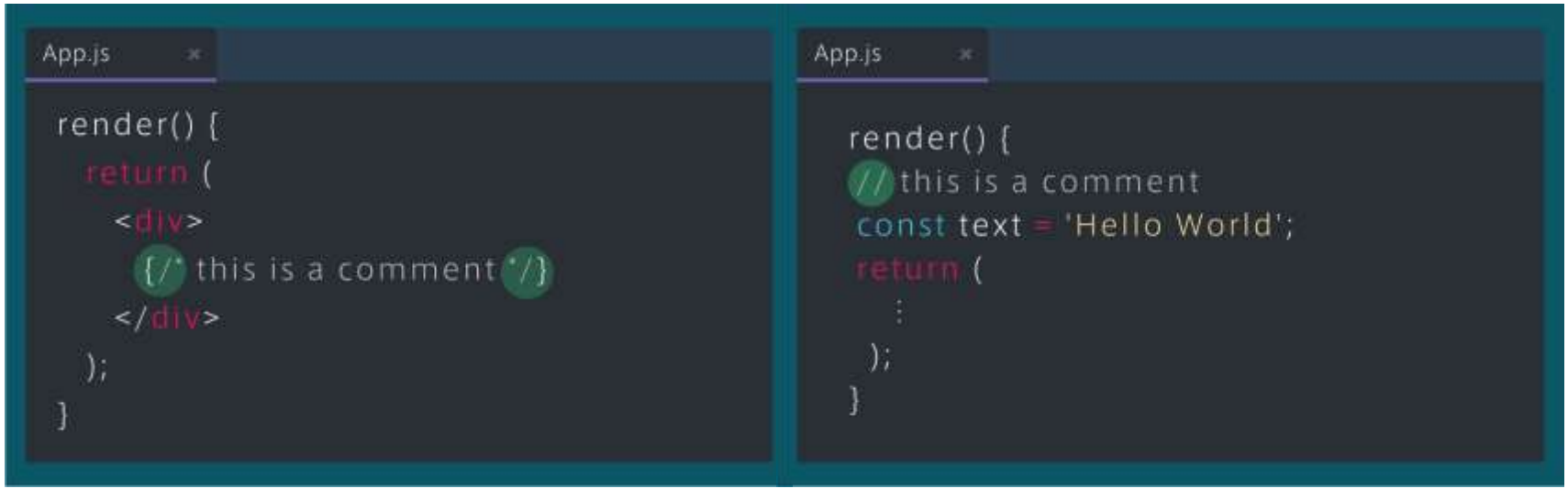
Putting JS Code Inside JSX

Even in the return section, JavaScript can be written inside JSX. To do this, the JavaScript code must be put in curly brackets { }. Also, tag attribute values can also be inserted with JavaScript curly brackets { } (like the example on the right).



JSX and JS Comments

We have already seen that when JSX is put between {/* and */} it becomes a comment (like in the left example). For JavaScript, though, lines that start with // are comments, like the right example.



External Stylesheet

- We can add styles to our JSX code using an External stylesheet by creating a styles.css file under the src folder.
- In JSX code, while we are trying to refer to the styles that have been defined in the styles.css file, we are supposed to use className as an attribute instead of the class attribute that we use in HTML.

Example:

```
<h1 className="heading">Heading</h1>
<p className="para">Paragraph</p>
```

Inline Stylesheet

- We can also add styles to our JSX code using the Inline stylesheet by adding the CSS under the *style* attribute.
- The styles should be added as *key-value* pair as in *JSON.*

Example:

```
<div style={{ padding: "20px 0", backgroundColor: "violet" }}>
```

Inline Stylesheet

- We can also create a JSON object and apply the CSS by using the style attribute.

Example:

```
const customStyle = {
   color: "red",
   fontSize: "20 px",
   border: "1px solid black",
 };

<h1 style={customStyle}>Heading</h1>
```

- Later, we can change the style by using
  customStyle.color = "blue";
- So, our style can change on the go.

Using Click Events

- You can use HTML and even put JavaScript code inside JSX.
- To change what is displayed when someone clicks a button, you need 2 tools, events and state.
- Your React Toolbox
    1. Events
    2. State

Functional vs Class components

Functional

Simple functions

Use Func components as much as possible

Absence of 'this' keyword

Solution without using state

Mainly responsible for the UI

Stateless/ Dumb/ Presentational

Class

More feature rich

Maintain their own private data - state

Complex UI logic

Provide lifecycle hooks

Stateful/ Smart/ Container

Fundamentals

Props

- Information that gets passed from one component to another is known as '*props*'.
- A component's props is an object. It holds information about that component.

```
class Greeting extends React.Component {
render() {
return <h1>My name is {this.props.firstName}!</h1>;
}}
ReactDOM.render(<Greeting firstName='Thanos' />, document.getElementById('app'));
```

State

- Unlike props, a component's state is not passed in from the outside. A component decides its own state.
- To make a component have state, give the component a state property.

```
constructor(props) {
super(props);
this.state = { breakfast: idly };
}
pancake() {
this.setState({breakfast: 'pancake'});
}}
```

React Hooks

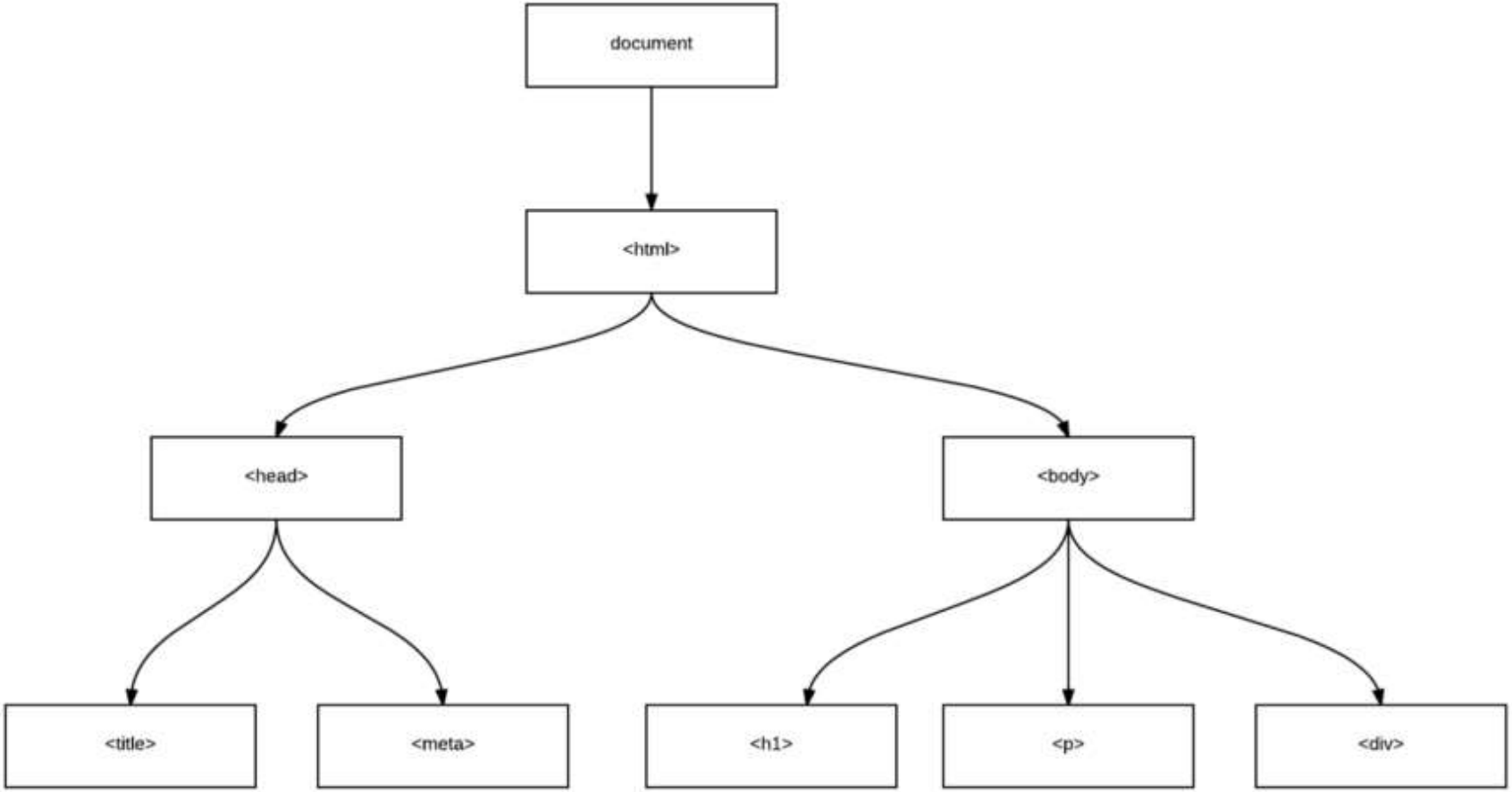
- On February 16, 2019, React 16.8 was released to the public.
- The release introduced React Hooks.
- Hooks are functions that let developers "hook into" React state and lifecycle features from function components.
- Hooks do not work inside classes — they let developers use React without classes.

How React Virtual DOM works?

What is DOM?

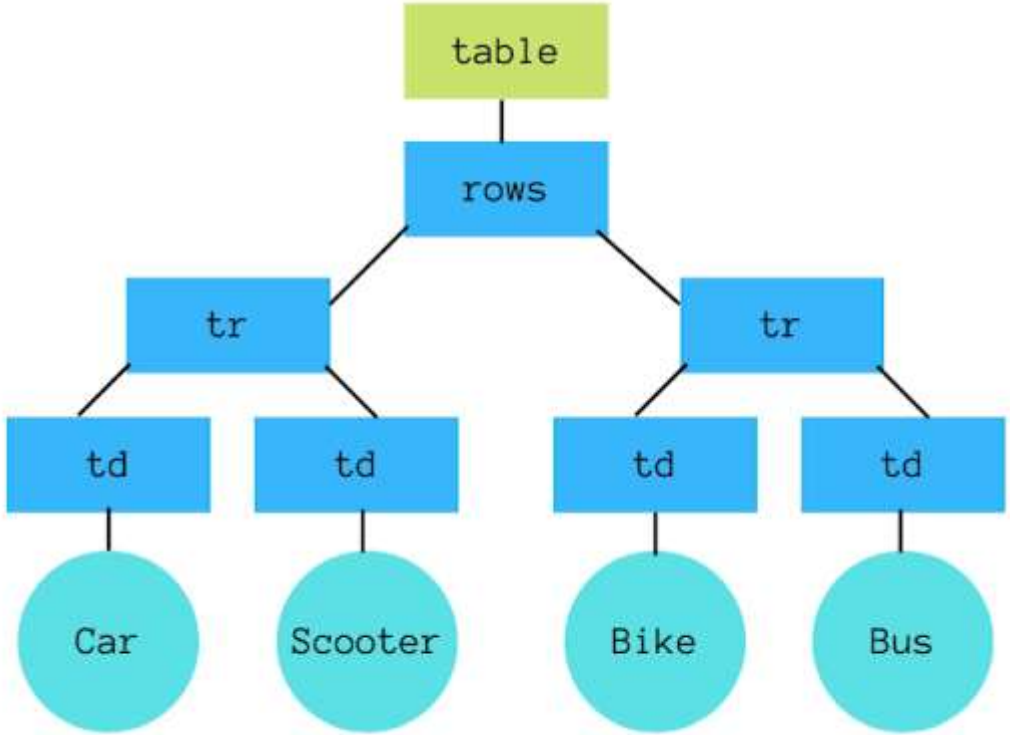
- DOM is a tree-structured abstraction of a page's HTML code.
- The primary benefit is the API that it provides, which allows us to scan through the HTML elements of a page quickly and manipulate them as needed.
  - to add new nodes
  - to edit a given node's content
- It also provides methods such as,
  - createElement
  - removeChild
  - getElementByID

Document Object Model (DOM) Example



Browser DOM

```
<table>
  <rows>
    <tr>
      <td>Car</td>
      <td>Scooter</td>
    </tr>
    <tr>
      <td>Bike</td>
      <td>Bus</td>
    </tr>
  </rows>
</table>
```



Updating Browser DOM

Every time something changes in our application, the DOM gets updated to reflect those changes in the browser. Whenever a DOM element is updated its child nodes also need to be re-rendered. The re-rendering or re-painting of the UI elements is what makes the whole process slow.

Efficiency issues with real DOM

- The “real” DOM updates a “target” node along with the entire web page (with its corresponding layout and CSS).
- You have a list of items, and it's just one of those items that you need to update. Traditionally, the “real” DOM would re-render the entire list and not exclusively the items that receive updates.
- The real DOM can't cope with pages carrying thousands and thousands of components to be re-rendered when updates are being passed through.

Virtual DOM

Virtual DOM is a virtual representation of the actual DOM. Just like the actual DOM, the Virtual DOM is a node tree that lists elements and their attributes and content as objects and properties.

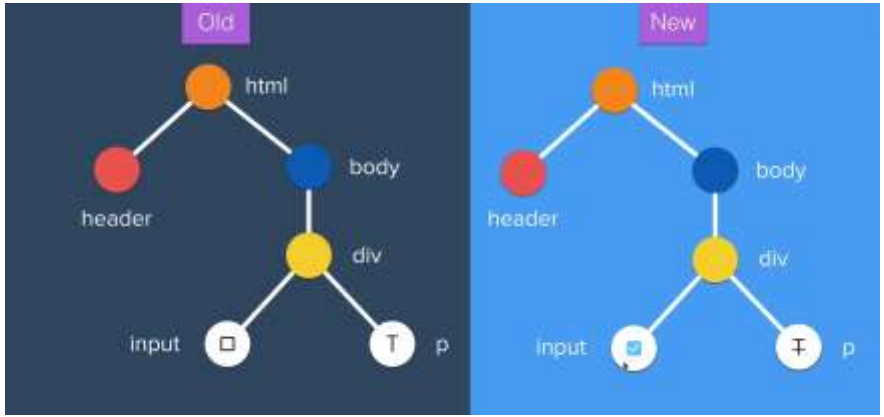
What is Virtual DOM?

- A lightweight abstraction/copy of the HTML DOM, having the same properties as the “real” one.
- The only difference is that it can't write to the screen as the actual DOM “can.”
- Also, it's local to React.
- A copy of the actual DOM that you get to update “intensively” without impacting the real DOM.

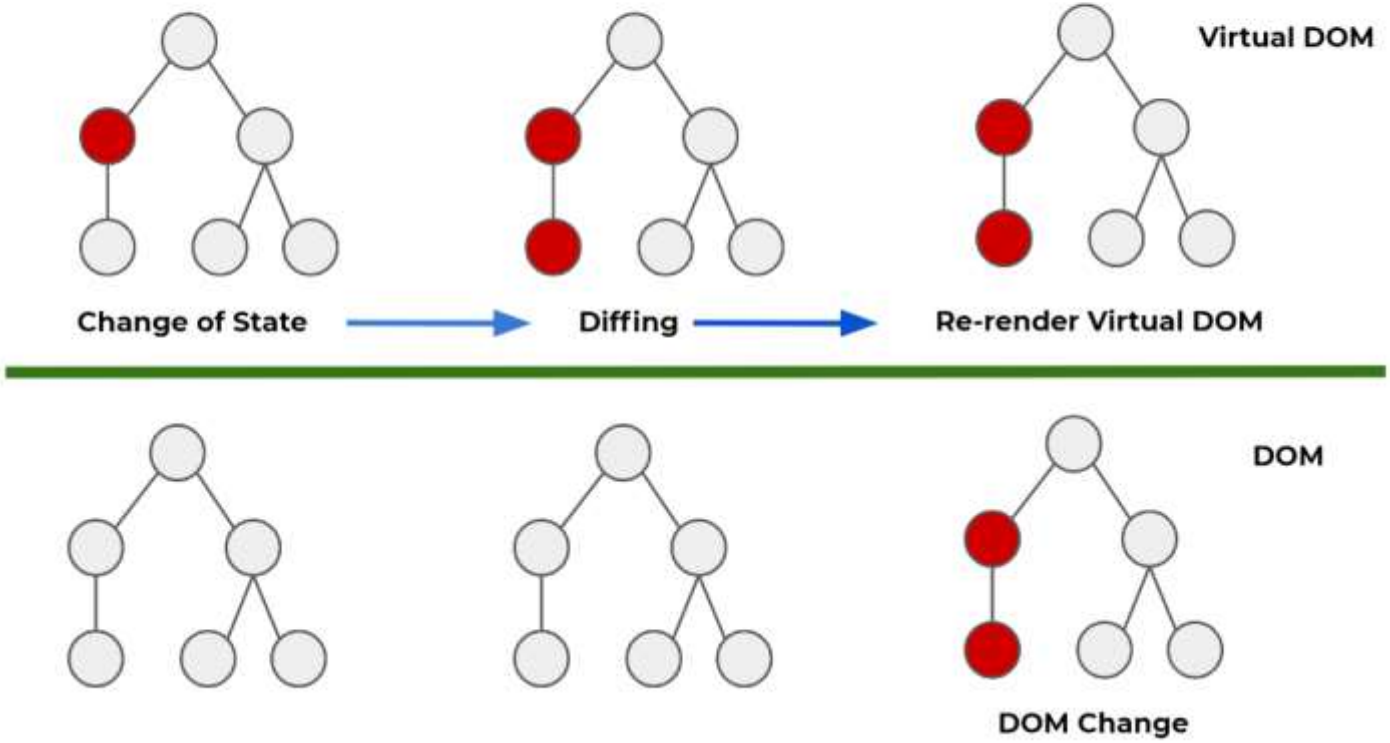
How does Virtual DOM work?

Snapshots, Diffing and Reconciliation

1. First, **it applies the given updates to the whole Virtual DOM**
2. Then, **it compares the updated virtual DOM with the snapshot of the virtual DOM**, using an algorithm called “diffing” during this entire process of comparing and spotting any changes/contrasts
3. Then, it's **specifically** (and exclusively) **those changed elements that it updates** in the actual DOM



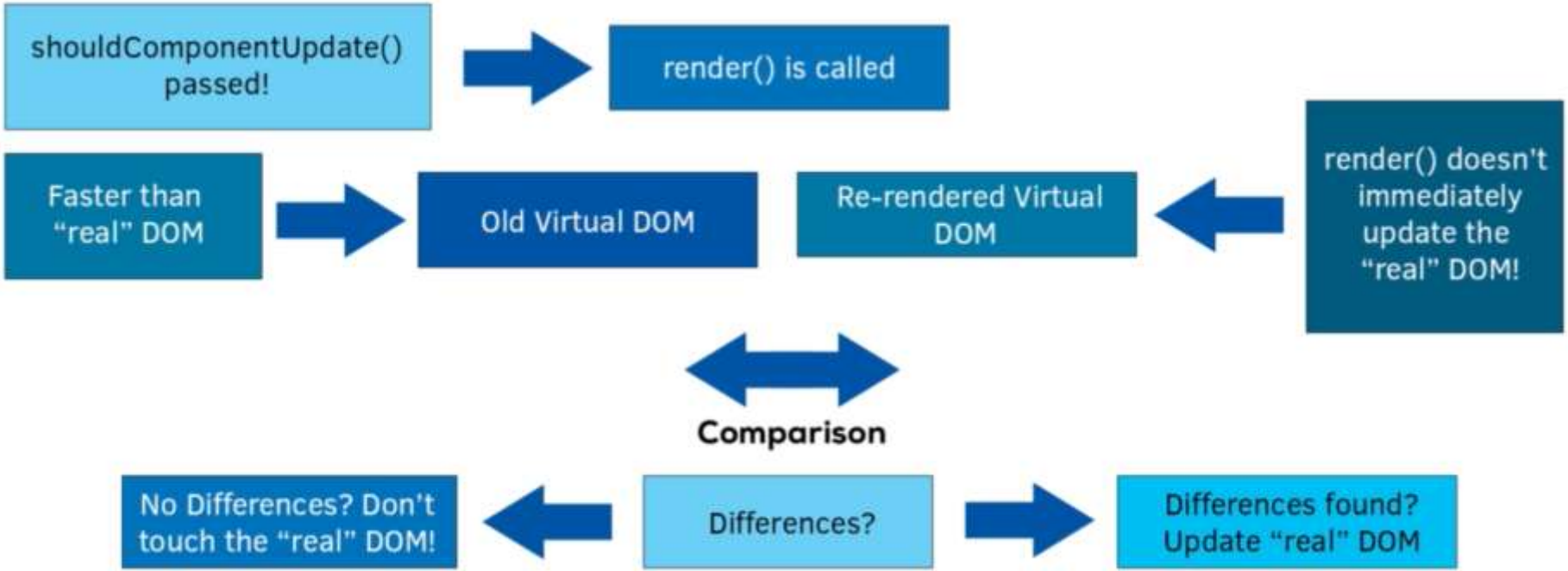
Virtual DOM


Virtual DOM
Change of State
Diffing
Re-render Virtual DOM
DOM
DOM Change

How does Virtual DOM work?

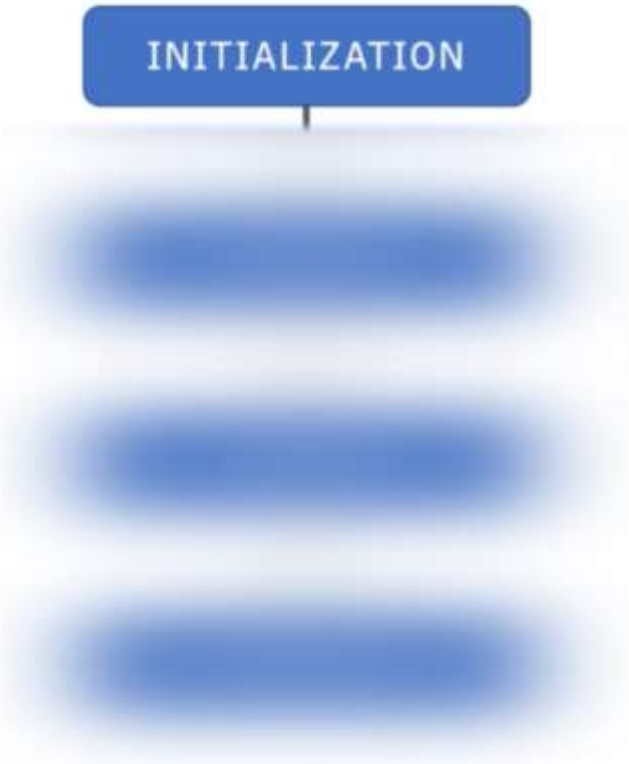
- Also, the entire process is called “reconciliation”:
- *Whenever updates need to be made to the actual DOM, React updates the Virtual DOM first, and then, once it has done its comparing, it syncs the Real DOM.*

How React Updates The DOM



Component Lifecycle: Creation

Component Lifecycle: Creation



It is the stage when the component starts its journey by setting up the state and the props.
The initialization process is done inside the constructor.

Component Lifecycle: Creation

It is the stage when the component is created and inserted in the browser DOM.

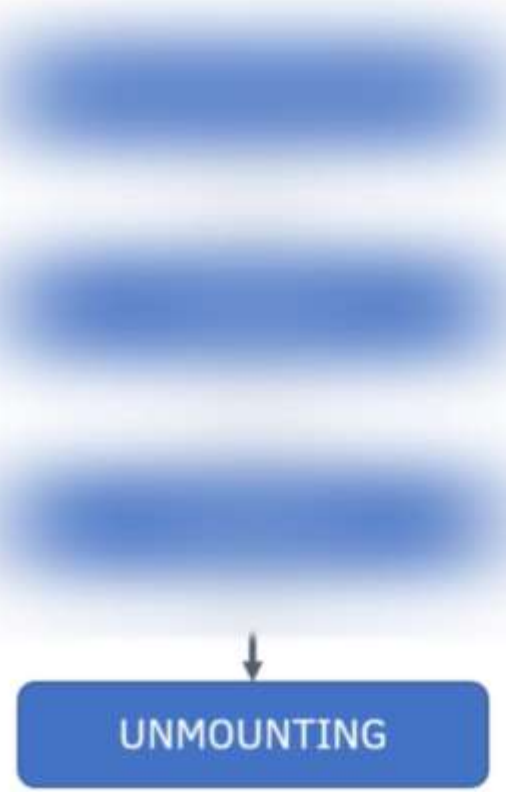
This stage comes after the initialization stage is completed.

This is the stage when render() function is called for the first time.

Component Lifecycle: Creation

It is the stage when the component is re-rendered due to state and/or props changes.

Component Lifecycle: Creation



It is the stage when the component is unmounted or removed from the browser DOM.

This happens automatically when the browser tab is closed or the route is changed which results in the mounting of different components..

Life cycle Methods

Component Lifecycle - Creation

constructor(props) {super(props)}

Use props to set default state

render()

Structure and Design JSX Code

Render Child Components

componentDidMount()

Make HTTP Requests

Component Lifecycle - Updation

shouldComponentUpdate(nextProps, nextState)

Decide whether to update or not?

render()

Structure and Design JSX Code

Render Child Components

componentDidUpdate()

Make HTTP Requests

Lifecycle methods

- Lifecycle methods are methods that get called at certain moments in a react component's life.
- There are three mounting lifecycle methods, when a component mounts, it automatically calls these three methods, in order:

componentWillMount, r**ender, componentDidMount**

- There are five updating lifecycle methods, which is called in this order whenever a component instance is updated:

componentWillReceiveProps, shouldComponentUpdate, componentWillUpdate, render, componentDidUpdate

- Finally, there is only one unmounting lifecycle method:

componentWillUnmount

componentWillMount

When a component renders for the first time only, componentWillMount gets called right before render.

render

- This method is responsible for returning HTML markup
- Using this.setState() ie., updating state is not recommended

componentDidMount

When a component renders for the first time, componentDidMount gets called right after the HTML from render has finished loading.

componentWillReceiveProps

When a component instance updates, componentWillReceiveProps gets called before the rendering only if the component will receive props.

shouldComponentUpdate

- shouldComponentUpdate automatically receives two arguments: nextProps and nextState.
- It’s typical to compare nextProps and nextState to the current this.props and this.state.
- If shouldComponentUpdate returns true, then nothing noticeable happens. But if shouldComponentUpdate returns false, then the component will not update.

componentWillUpdate

- componentWillUpdate gets called in between shouldComponentUpdate and render.
- It receives two arguments: nextProps and nextState.
- We cannot call this.setState from componentWillUpdate. Its main purpose is to interact with things outside like checking the window size or interacting with an API.

componentDidUpdate

- componentDidUpdate automatically gets passed two arguments: prevProps and prevState.
- prevProps and prevState are references to the component's props and state before the current updating period began.
- We can call this.setState from componentWillUpdate based on condition.

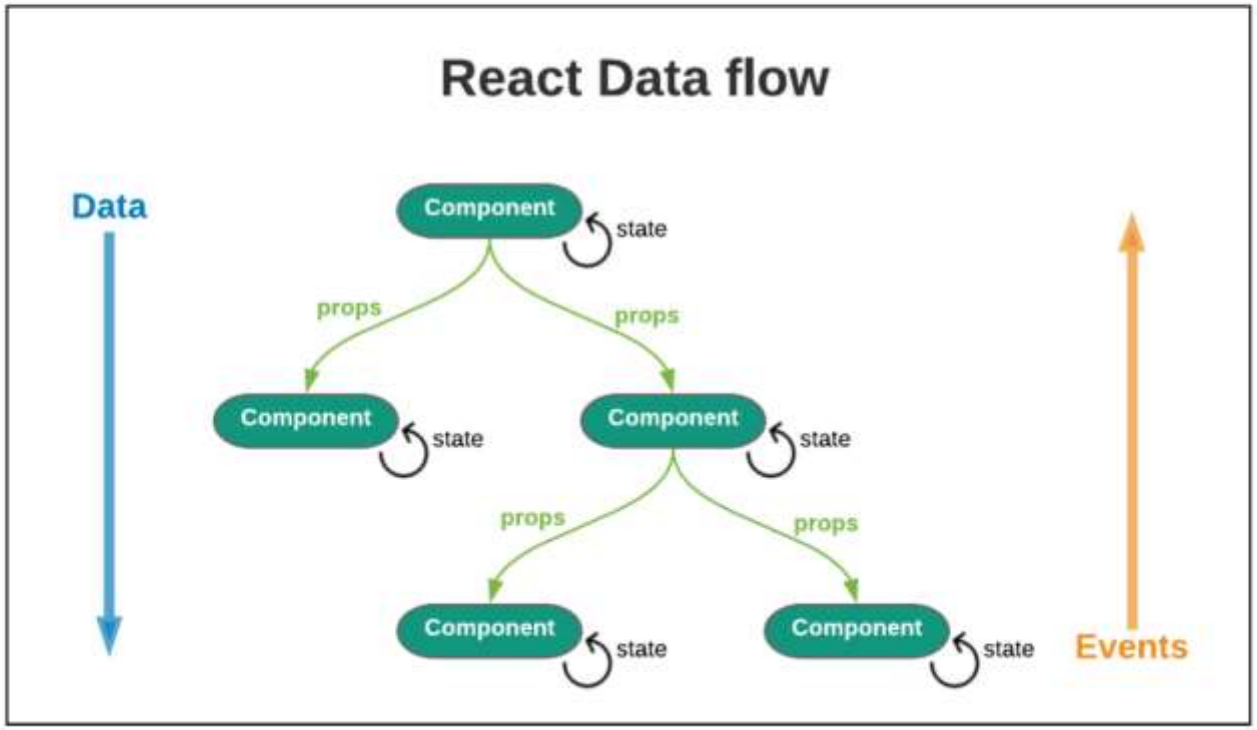
componentWillUnmount

It gets called right before a component is removed from the DOM.

One-way Data Binding

- One-way data-binding means that through the whole application data flows only in one direction. This gives better control over it.
- The data is passed from the parent component to the child component through read-only props.
- These props cannot be sent back to the parent component. This is how one-way data binding works.
- Although, the child component can communicate with the parent component to update the state via callback functions.

One-way Data Binding



How to Pass Data from Child to Parent in React

- React follows one-way data binding i.e. data flows only one way, from the parent to the child component.
- The reverse of that, i.e. passing data from the child to the parent is not possible, at least not in theory.
- While there is no direct way to pass data from the child to the parent component, there are workarounds.
- The most common one is to pass a handler function from the parent to the child component that accepts an argument which is the data from the child component.

Fetching API Data using Axios

- Axios is lightweight package and use to make HTTP requests in any Javascript library like React, Angular or Vue.

- Axios makes it easy to send asynchronous HTTP requests to REST endpoints and perform CRUD operations.

- If you use fetch method in Javascript, Axios is the “Easy to use” version of fetch.

Advantages of using Axios

- Axios by default Work in JSON format.So no more JSON parsing.

- Make all types of HTTP requests (GET, POST, PUT, DELETE)

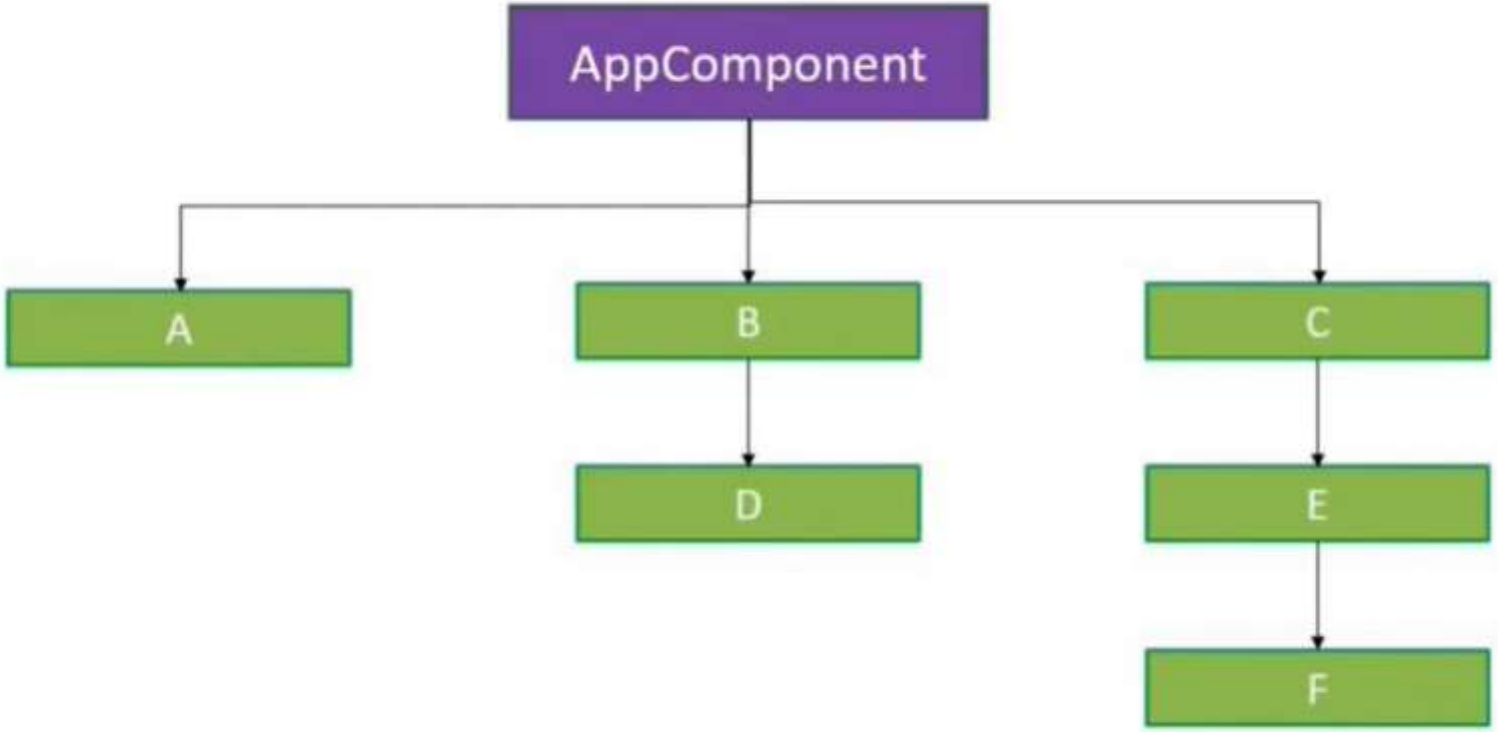
    - Installing axios using

      => npm install axios

useContext React Hook

- React Context is a way to manage state globally.
- It can be used together with the useState Hook to share state between deeply nested components more easily than with useState alone.
- It can be used to send the props from one component to another deeply nested component directly.
- For example, if we have 4 components: component A, B, C & D. If we want to send the prop value from A to D, without useContext, we will need to pass the props through each nested component. This is called ‘prop drilling’.
- Context provides a way to pass data through the component tree without having to pass props down manually at every level.

Context



Example1 **Example2** **Example3**

React useRef Hook

- The useRef Hook allows you to persist values between renders.

- It can be used to store a mutable value that does not cause a re-render when updated.

- It can be used to access a DOM element directly.

useState vs useRef

- useState will re-render when the content change and update in UI.

- useRef doesn’t notify you when its content changes. Mutating the (.current) property doesn’t cause a re-render.

React useMemo Hook

- The React useMemo Hook returns a memoized value.
- Think of memoization as caching a value so that it does not need to be recalculated.
- useMemo will only recompute the memoized value when one of the dependencies has changed. This optimization helps to avoid expensive calculations on every render.
- This can improve performance of application, when we are performing most expensive calculations

useEffect vs useMemo

- useEffect() will run after rendering of the component.
- useMemo() will run during rendering of the component.
- Syntax
  - useEffect(()=>{},[a,...])
  - useMemo(()=>{},[a,...])