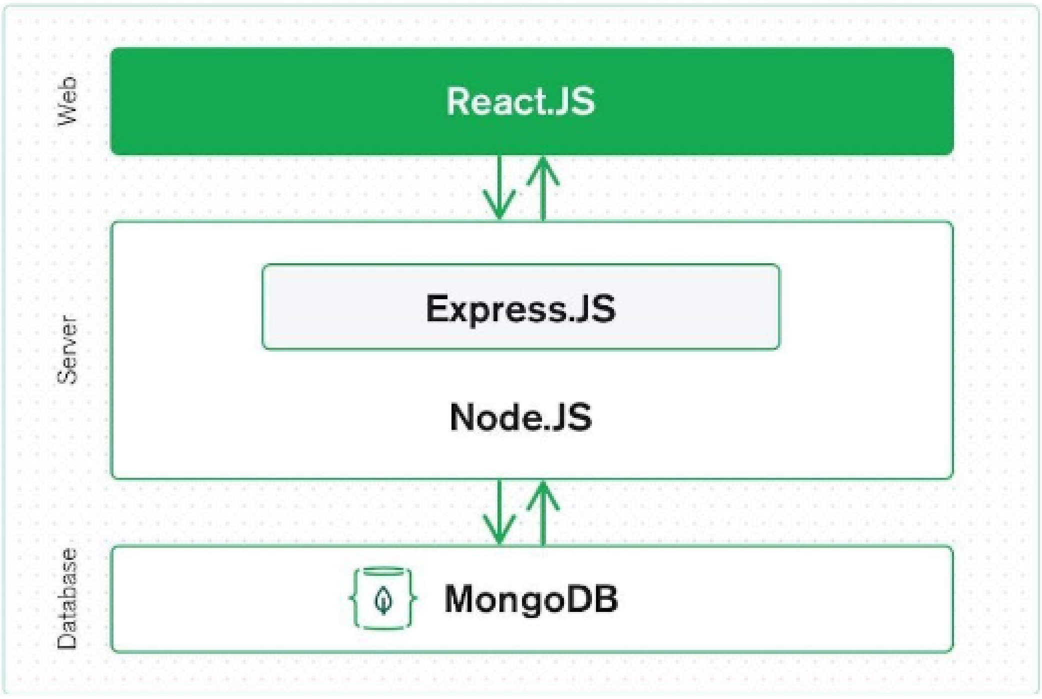
React

Web Dev Stack

- Set of tools used in tandem to develop web apps
- Usually divided into three:
  - Frontend framework
  - Backend framework
  - Database
- Full stack: frameworks for all three
- Eg: MEAN, MERN, Meteor.js. Flutter, LAMP, Ruby on Rails

MERN: MongoDB, Express.js, React.js, Node.js



Why MERN?

- Suited for web apps with a large amount of interactivity on the frontend
- JSON/JavaScript everywhere
- Isomorphic

React.js

- Frontend JS library
- Dynamic client-side apps
- Good at handling stateful, data-driven interfaces

Properties

- **Declarative**
  - Allows devs to simply describe **what the UI should look like** rather than specifying how to update the UI step by step
  - We define desired outcome for a specific state, **React takes care of DOM manipulations** to do the same
- Based on **reusable UI components**
- Express.js server-side framework, running inside a Node.js server
- **Single way data flow**
  - Set of immutable values passed to components -> rendered as properties in HTML
  - **Component cannot modify any properties directly**, but can pass a callback function with which we can do modifications
  - **"Properties flow down, actions flow up"**
- **Virtual DOM**
  - Creates in-memory data structure cache -> computes changes made to app -> updates the browser
  - Allows programmer to code as if the whole page is being rendered on each change even though **React only re-renders the components that are changed**

Creating React App

- Using node package manager (npm)
- Direct import in HTML:

```
<script src="https://unpkg.com/react@18/umd/react.development.js" crossorigin>
</script>

<script src="https://unpkg.com/react-dom@17/umd/react-dom.development.js"
crossorigin> </script>
```

- When deploying, replace `development.js` with `production.min.js`

Using npm

1. Download Node.js
2. `npm install -g create-react-app`
3. `npm create-react-app my-app`
4. `cd my-app`
5. `npm start`
   Check versions: `npm ls react`, `npm ls react-dom`

React Elements

- React element = description of what actual browser DOM element should look like

- Smallest building blocks in React and are returned by React components
- Immutable objects

```
React.createElement("h1", {id:"recipe-0",'data-type': "title"}, "Baked Salmon")
```

React Components

- Reusable, self-contained piece of UI that can be composed of multiple elements and other components
- Either functional (stateless) or class (stateful)
- Contains a root component that includes other subcomponents

```
// functional
function Greeting(props) {
  return <h1>Hello, {props.name}!</h1>;
}

// class
class Greeting extends React.Component {
  render() {
    return <h1>Hello, {this.props.name}!</h1>;
  }
}
```

Comparison

Aspect	**React Element**	**React Component**
Definition	A plain object that represents a DOM node.	A reusable piece of UI logic that returns elements.
Creation	Created with `React.createElement()` or JSX.	Created as a function or class.
State	Cannot manage its own state.	Can manage its own state (class components) or hooks (functional components).
Lifecycle Methods	Does not have lifecycle methods.	Class components have lifecycle methods; functional components can use hooks.
Reusability	Typically not reusable.	Designed to be reusable.
Examples	`<h1>Hello</h1>`	`function MyComponent() { return <h1>Hello</h1>; }`

Aspect	**Stateful Components**	**Stateless Components**
Definition	Components that maintain their own internal state.	Components that do not maintain internal state; they rely on props.

Aspect	Stateful Components	Stateless Components
State Management	Can manage and update local state using `this.state` (class components) or `useState` (functional components).	Do not manage state; render UI based solely on received props.
Lifecycle Methods	Can have lifecycle methods (e.g., `componentDidMount`, `componentDidUpdate`).	Do not have lifecycle methods; can use hooks in functional components for side effects.
Usage	Suitable for complex components that require interactivity or state management.	Suitable for simple presentational components that render UI based on props.

Rendering

```
ReactDOM.render(
        React.createElement(HelloClass, null, null),  // React element
        document.getElementById('root')              // placeholder (location)
        );
```

- **React 17**

```
ReactDOM.render(
    <h1>Batman</h1>,
    document.querySelector(“#container”)
  );
```

The render method takes two arguments:

The HTML-like elements (aka JSX) you wish to output

The location in the DOM that React will render the JSX into

- **React 18**

Instead of ReactDOM.render , createRoot is used

const root = createRoot(container);
root.render(element);

Example:
var destination =document.querySelector("#container");
const root=ReactDOM.createRoot(destination);
root.render(<h1>React World</h1>);

Create a React root for the supplied container and return the root.
The root can be used to render a React element into the DOM with render

JSX

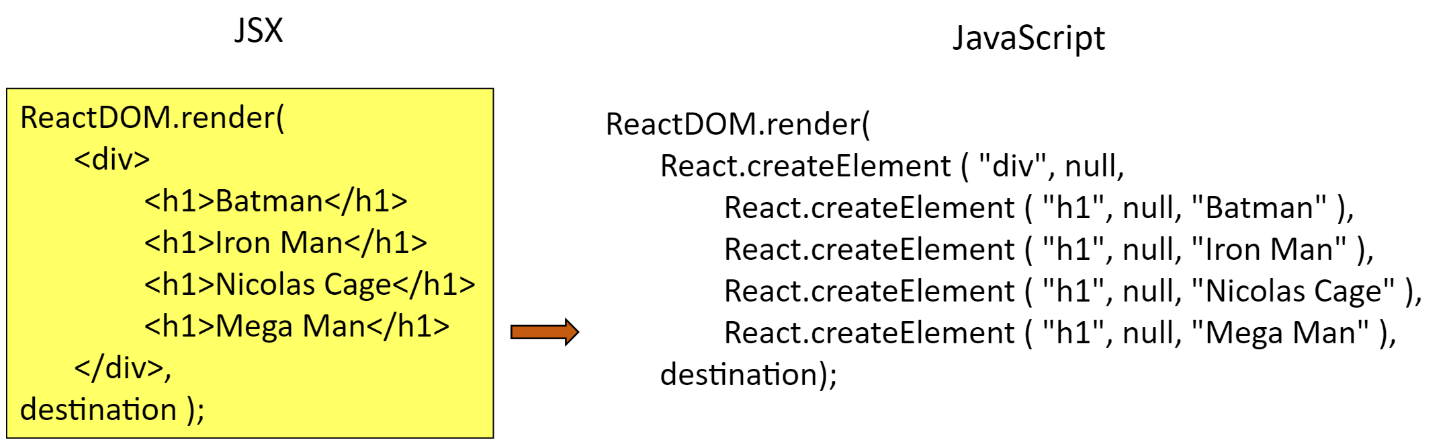
- Special syntax that allows you to mix HTML and Javascript
- Javascript XML used in React apps
- JSX compiles into pure Javascript

```
<script src="https://unpkg.com/@babel/standalone/babel.min.js"></script>
<script type="text/babel">
    const root = ReactDOM.createRoot(document.getElementById('root'));
    root.render(<h1>Welcome to REACTJS</h1>);
</script>
```

- Babel: JS compiler used to convert JSX (and others) to JS

```
import React from 'react';
import ReactDOM from 'react-dom/client';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<h1>Welcome to REACTJS</h1>);
```



JSX

```
ReactDOM.render(
    <div>
        <h1>Batman</h1>
        <h1>Iron Man</h1>
        <h1>Nicolas Cage</h1>
        <h1>Mega Man</h1>
    </div>,
destination );
```

JavaScript

```
ReactDOM.render(
    React.createElement ( "div", null,
        React.createElement ( "h1", null, "Batman" ),
        React.createElement ( "h1", null, "Iron Man" ),
        React.createElement ( "h1", null, "Nicolas Cage" ),
        React.createElement ( "h1", null, "Mega Man" ),
    destination);
```

Example for Function

With JSX

```
<script type="text/babel">
  function App() {
    return <h1>Welcome to REACTJS</h1>;
  }

  const root = ReactDOM.createRoot(document.getElementById('root'));
  root.render(<App />);
</script>
```

Without JSX

```
<script type="text/babel">
  function App() {
    return React.createElement('h1', null, 'Welcome to REACTJS');
  }

  const root = ReactDOM.createRoot(document.getElementById('root'));
  root.render(React.createElement(App));
</script>
```

Properties (props)

- Mechanism for passing data from one component to another, typically from a parent component to a child component
- Props allow data to flow **downward** in the component hierarchy (from parent to child).
- Props are **immutable** from the perspective of the child component. This means that a child component cannot modify the props it receives.

- If a child component needs to change data, it should notify the parent to change the state, and then the parent can pass updated props back down.
- Allow you to customise components; props are to React components as attributes are to HTML elements

```
import React from 'react';
import ReactDOM from 'react-dom/client';

// Define the HelloWorld component
class HelloWorld extends React.Component {
  render() {
    return <p>Hello, {this.props.greetTarget}!</p>;
  }
}

// Create a root for rendering
const root = ReactDOM.createRoot(document.querySelector("#container"));

// Render the HelloWorld components
root.render(
  <div>
    <HelloWorld greetTarget="Batman"/>
    <HelloWorld greetTarget="Iron Man"/>
  </div>
);
```

props.children

- Represents content/elements inside a component's JSX tag
- Component will have opening and closing tag

```
import React from 'react';
import ReactDOM from 'react-dom/client';

// Define the Buttonify component
class Buttonify extends React.Component {
  render() {
    return (
      <div>
        <button type={this.props.behavior}>
          {this.props.children}  {/* Accessing the children passed to Buttonify */}
        </button>
      </div>
    );
  }
}

// Create a root for rendering
const root = ReactDOM.createRoot(document.querySelector("#container"));
```

```
// Render the Buttonify component
root.render(
  <div>
    <Buttonify behavior="submit">SEND DATA</Buttonify>  {/* "SEND DATA" becomes
props.children */}
  </div>
);
```

Validating property values

```
import React from 'react';
import PropTypes from 'prop-types';

class MyComponent extends React.Component {
  render() {
    return <div>{this.props.message}</div>;
  }
}

// Prop type validation
MyComponent.propTypes = {
  message: PropTypes.string.isRequired,  // message must be a string and is required
  count: PropTypes.number,                // count must be a number, but is not
required
};
```

Setting default property values

```
// greeting.js

import React from 'react';
// Define the component
class Greeting extends React.Component {
  render() {
    return <h1>Hello, {this.props.name}!</h1>;
  }                 // curly braces used for JS expressions in JSX
}

// Set default props
Greeting.defaultProps = {
  name: 'Guest'
};

export default Greeting;
```

```
// main.js
```

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import Greeting from './Greeting'; // Assume the component is saved in Greeting.js

const root = ReactDOM.createRoot(document.querySelector("#container"));

root.render(
  <div>
    <Greeting name="Batman" />      {/* This will render: Hello, Batman! */}
    <Greeting />                   {/* This will render: Hello, Guest! */}
  </div>
);
```

Styling React Components

Inline styling:

```
class myFormat extends React.Component {
        render()
        {
                return(
                <div>
                <h1 style = {{color: "red"}}> This is styled h1 </h1>
                <p style={{ fontSize: "18px", color: "blue" }}>This is a para</p>
                </div>
                );
        }
}

class myFormat2 extends React.Component {
        render()
        {
                var letterStyle = {padding: 10, margin: 10, color: "blue",
fontFamily="monospace"};
                return(
                        <div style={letterStyle}>
                                {this.props.children}
                        </div>
                )
        }
}
```

- Double `{{}}`:
  - The first set of curly braces `{}` allows you to write JavaScript inside JSX.
  - The second set of curly braces `{}` creates a JavaScript object representing CSS styles.