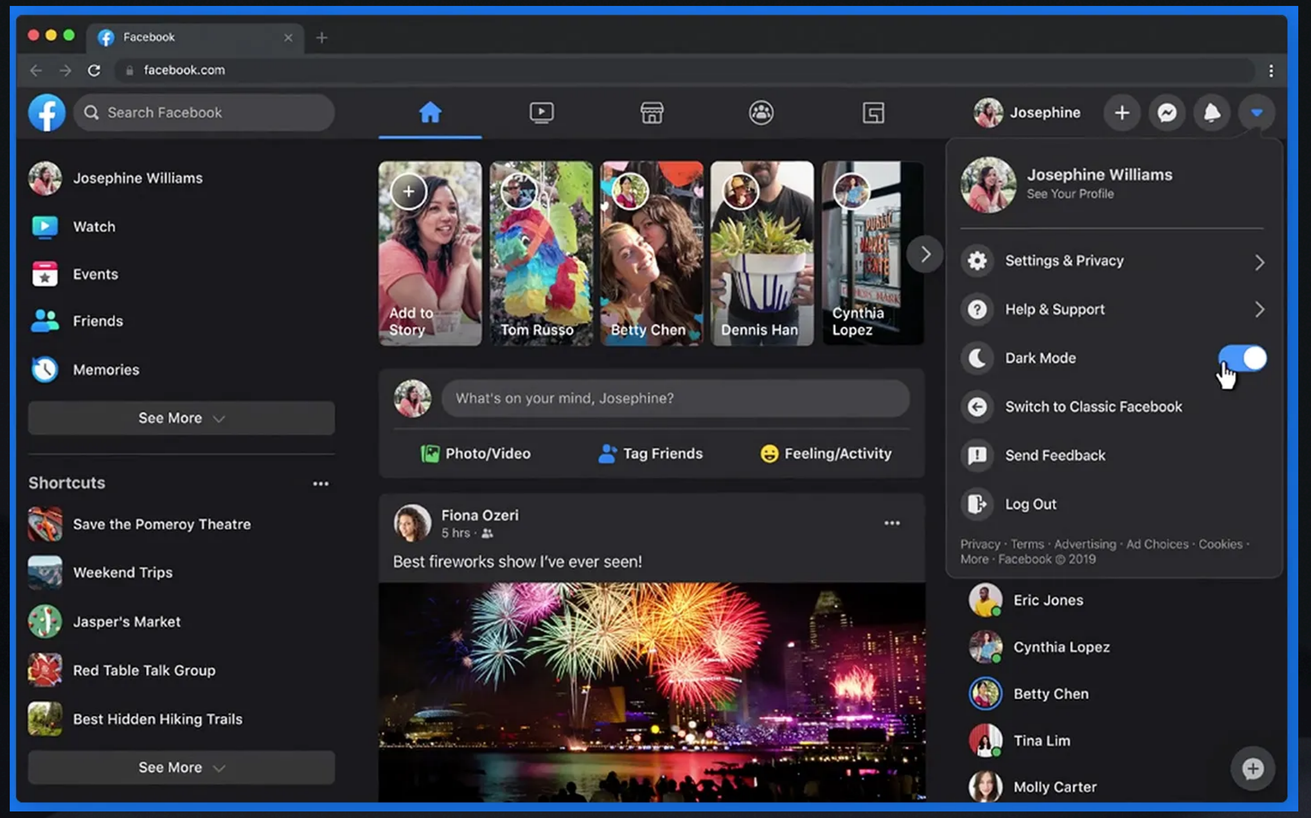
React

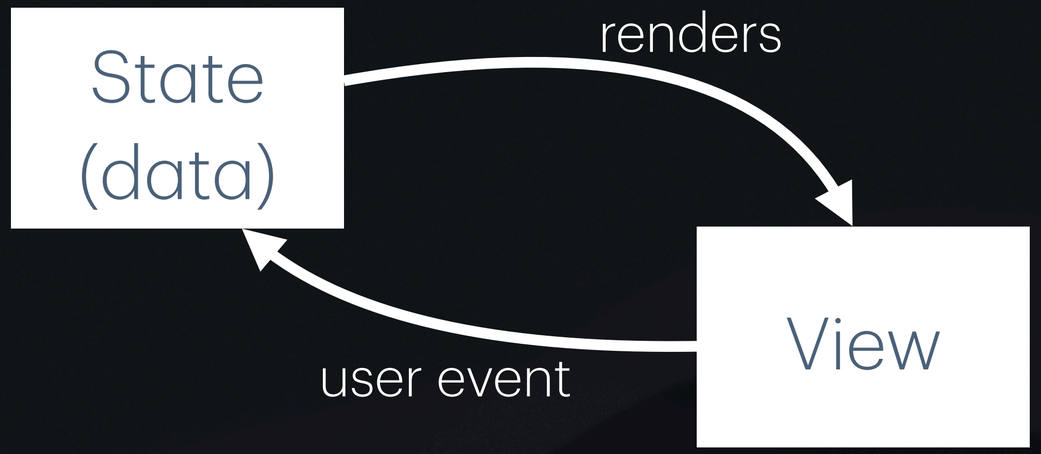
Building User Interfaces in JavaScript since 2013

“Vanilla” JS is hard!

- Keep track of state
- Find the correct DOM node to update
- Keep track of user events
- Make it performant!

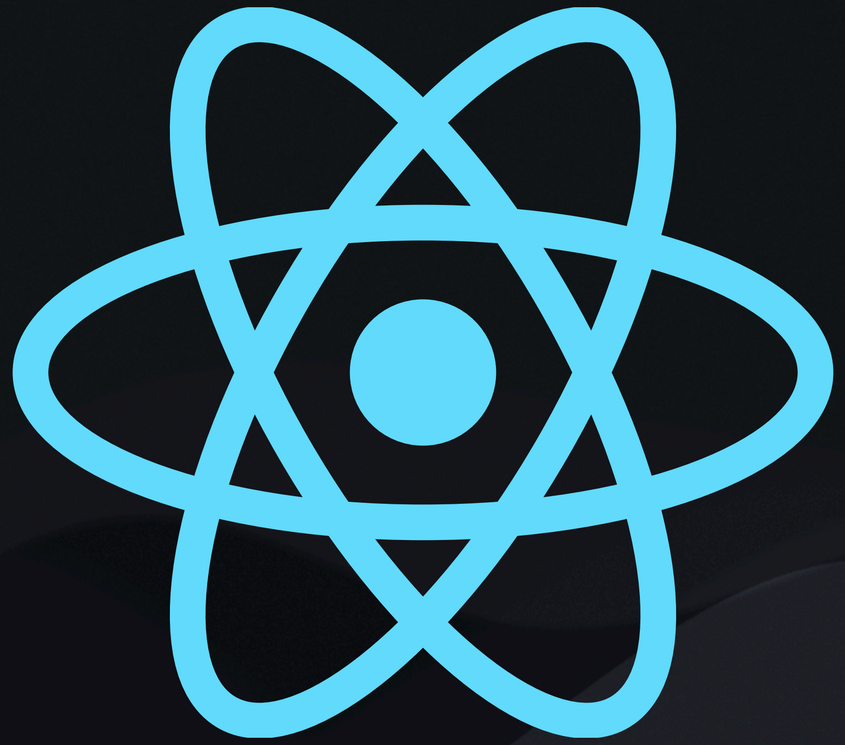


Wouldn’t it be nice…



- Declarative instead of imperative
- When data changes, pretend that we’re re-rendering “everything”
  - React will take care of making sure we only actually make necessary changes
  - “Virtual DOM” 🤔

React



react.dev

React

State

Effects

Components

Props

Hooks

JSX

JavaScript XML (JSX)

```jsx
import React from "react";

const Clicker = () => {
  return (
    <div>
      <h1>Clicker</h1>
    </div>
  );
};

export default Clicker;
```

✨JSX

- NOT regular JavaScript
- Needs To be compiled (translated) to regular JavaScript so that browsers can understand it:

```
React.createElement(“div", null,
  React.createElement(“h1", null, “Clicker")
)
```

State

```
import React from "react";

const Clicker = () => {
  const [count, setCount] = useState(0);
  const handleClick = () => {
    setCount(count + 1);
  };
  return (
    <div>
      <h1>Clicker</h1>
      <p>You have clicked the button {count} times</p>
      <button onClick={handleClick}>Click me</button>
    </div>
  );
};

export default Clicker;
```

- Set the initial state: `useState(0)`
- Get the current state: `count`
- Update the state: `setCount`
- Note the square brackets!
  `[count, setCount]`
- Note the curly brackets, passing data into JSX!

Components

```
import Clicker from "./clicker";

const App = () => {
  return (
    <div>
      <Clicker /> count state: 0
      <Clicker /> count state: 0
      <Clicker /> count state: 0
      <Clicker /> count state: 0
    </div>
  );
};
```

- Components are capitalized – unlike HTML tags
- Components are functions that return JSX
- Each component has it’s own state
- Must return a single element
  - We often use a wrapper, e.g.

Components

```
import Clicker from "./clicker";

const App = () => {
  return (
    <div>
      <Clicker />
      <Clicker />
      <Clicker />
      <Clicker />
    </div>
  );
};
```

- Components are capitalized – unlike HTML tags
- Components are functions that return JSX
- Each component has it’s own state
- Must return a single element
  - We often use a wrapper, e.g.

Props

```
const Clicker = (props) => {
  const { max, min, name } = props;
  const [count, setCount] = useState(0);
  const addOne = () => {
    if (count < max) {
      setCount(count + 1);
    }
  };
  const subtractOne = () => {
    if (count > min) {
      setCount(count - 1);
    }
  };
  return (
    <div>
      <h1>Clicker for {name}</h1>
      <p>You have clicked the button {count} times</p>
      <button onClick={addOne}>Add One</button>
      <button onClick={subtractOne}>Subtract One</button>
    </div>
  );
};
```

- Props is the first argument to each component
- Props is an object with named properties
- Useful for distinguishing the same component in different circumstances

Props

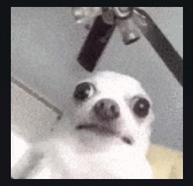
```
const Clicker = (props) => {
  const { max, min, name } = props;
  const [count, setCount] = useState(0);
  const addOne = () => {
    if (count < max) {
      setCount(count + 1);
    }
  };
  const subtractOne = () => {
    if (count > min) {
      setCount(count - 1);
    }
  };
  return (
    <div>
      <h1>Clicker for {name}</h1>
      <p>You have clicked the button {count} times</p>
      <button onClick={addOne}>Add One</button>
      <button onClick={subtractOne}>Subtract One</button>
    </div>
  );
};
```

```
import Clicker from "./clicker";

const App = () => {
  return (
    <div>
      <Clicker max={100} min={1} name="Mangos" />
      <Clicker max={5} min={0} name="Apples" />
      <Clicker max={10} min={5} name="Bananas" />
      <Clicker max={20} min={10} name="Pears" />
    </div>
  );
};
```

```
import React from "react";
```

```
export default Clicker;
```



```
return (<div></div>);
```

If not JavaScript,

why JavaScript shaped?

Why browser not 🤮?

webpack.js.org

babeljs.io

Webpack

webpack.config.js

```
module.exports = {
  mode: "development",
  entry: "./src/app.jsx",
  output: {
    path: path.resolve(__dirname, "public"),
    filename: "main.js",
  },
  // other options...
};
```

- Webpack is a *bundler*
- Give Webpack an entry file
- Webpack will follow import and export statements
- Tell Webpack where to put the output file
- Instead of many different <script> tags in your HTML, only one!

Webpack

app.jsx

```jsx
import React from "react";
import { createRoot } from "react-dom/client";

const App = () => {
  return (
    <div>
      <h1>Hello React! ⚛️</h1>
    </div>
  );
};

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

index.html

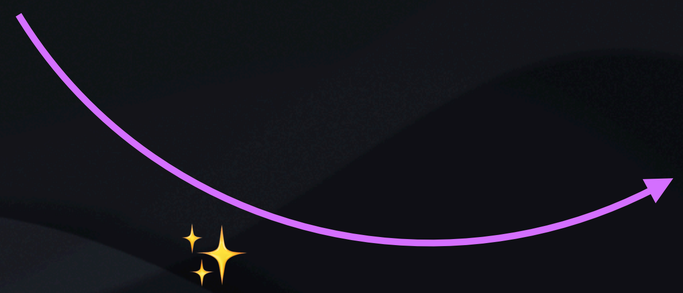
```html
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>React Mini</title>
    <script defer src="main.js"></script>
  </head>
  <body>
    <div id="root"></div>
  </body>
</html>
```

Babel

Babel is a compiler

IN

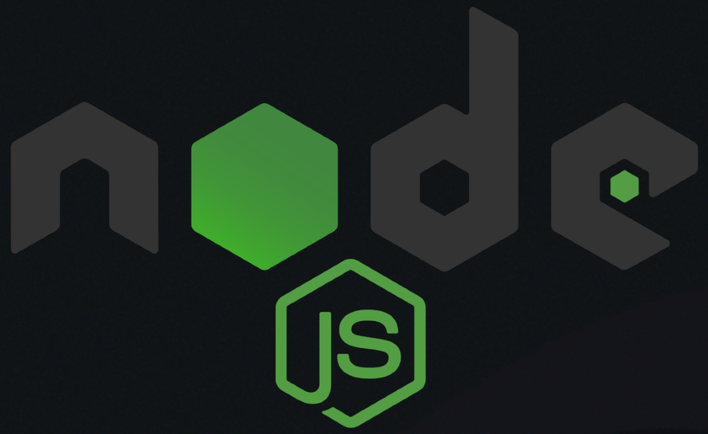
```
const profile = (
  <div>
    <img src="avatar.png" className="profile" />
    <h3>{[user.firstName, user.lastName].join(" ")}</h3>
  </div>
);
```



OUT

```
const profile = _jsxs("div", {
  children: [
    _jsx("img", {
      src: "avatar.png",
      className: "profile",
    }),
    _jsx("h3", {
      children: [user.firstName, user.lastName].join(" "),
    }),
  ],
});
```

Node & NPM



nodejs.org

npmjs.com

Node

```
const paragraphs = document.getElementsByTagName("p");

paragraphs.forEach((p) => {
  p.textContent = "Hello World";
});
```

```
> node script.js
ReferenceError: document is not defined
```

```
// Import the fs module, built-in to Node.js
const fs = require("fs");

// Read in a CSV file
const rawData = fs.readFileSync("data.csv", "utf8");

console.log(rawData);
```

```
> node script.js
Name,Age,City
John,25,New York
Jane,30,Los Angeles
Jim,35,Chicago
Jill,40,Houston
```

- Allows you to run JavaScript outside of the browser
- Necessary to run Webpack and Babel before handing over our compiled script to the browser
- 🕵️ Sneak Peak: We'll discuss Node in more detail when we start building servers
- Does NOT have access to the DOM
- DOES have access to your filesystem

NPM (Node Package Manager)

```
> npm init -y
Wrote to ~/code/my-app/package.json:

{
  "name": "my-app",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "description": ""
}
```

```
> npm install react

added 1 package, and audited 2 packages in 651ms

found 0 vulnerabilities
```

```
> cat package.json
{
  .......
  "license": "ISC",
  "description": "",
  "dependencies": {
    "react": "^19.1.0"
  }
}
```

- NPM allows you to download packages from the internet
- `npm init` initializes a new package.json
- `npm install` installs a new package
- package.json keeps track of the packages you've installed
- node_modules stores the packages
  - Add node_modules to .gitignore .
- If you are given a package.json, just run `npm install`. It will install all of the necessary packages – no need to install them individually

Please Install Node Now

nodejs.org

```
> node --version 😭
command not found: node
```

```
> node --version 🥳
v22.14.0
```

React Practice Time

Rendering Lists in React

```jsx
const HomePage = () => {
  const cats = [
    { id: 1, name: "Finn" },
    { id: 2, name: "AJ" },
    { id: 3, name: "Shahid" },
  ];
  return <CatList cats={cats} />;
};

const CatList = (props) => {
  const { cats } = props;
  return (
    <div>
      <h1>Cat List</h1>
      <ul>
        {cats.map((cat) => (
          <li key={cat.id}>{cat.name}</li>
        ))}
      </ul>
    </div>
  );
};
```

✅

- Whenever you want to execute a JavaScript expression within JSX, wrap it in **{…}**
- **`array.map()`** is a *very* commonly used array method in React
- Each element must have a unique key
- Whatever you put inside the curly brackets must be an *expression* – it must *return* something.

```jsx
<ul>
  {for (let i = 0; i < cats.length; i++) {
    <li key={cats[i].id}>{cats[i].name}</li>;
  }}
</ul>
```

💥

Functions As Props

```
const HomePage = () => {
  const [cats, setCats] = useState([]);
  const addCat = () => {
    setCats([
      ...cats,
      {
        id: cats.length + 1,
        name: randomName(),
      },
    ]);
  };
  return <CatList cats={cats} addCat={addCat} />;
};

const CatList = (props) => {
  const { cats, addCat } = props;
  return (
    <div>
      <h1>Cat List</h1>
      <button onClick={addCat}>Add Cat</button>
      <ul>
        {cats.map((cat) => (
          <li key={cat.id}>{cat.name}</li>
        ))}
      </ul>
    </div>
  );
};
```

```
cats state: [
    { id: 1, name: ... },
    { id: 2, name: ... },
    { id: 3, name: ... }
]
```

- It is often necessary for a component to update another component's state
- This can be done by simply passing the necessary function down as a prop
  - Remember: functions are just data!