Intro to React

JavaScript Frameworks

- JavaScript frameworks are libraries of pre-written code.
  - Think of SpringBoot, which is a Java framework

- These libraries enhance reusability by encapsulating code that is used often.

- Frameworks are commonly used for: making API calls, managing the state of a web component, assist with routing.
  - The main idea here is that using a readily available framework functionality will save the developer from writing all that code on their own!

React History

- React was released in 2013, since then it has become the most widely used JavaScript framework.

- Other popular frameworks include Angular and VUE.js.

React Study Tips

- Being proficient with regular vanilla JavaScript, HTML, and CSS makes learning a framework like React easier!

- Having a mastery of foundational JavaScript can help you maximize a framework’s capabilities.

Managing React Projects

- NPM (Node Package Manager) is used to manage projects.

package.json

```
{
  "name": "bookmark-manager-web",
  "version": "0.2.0",
  "private": true,
  "type": "module",
  "scripts": {
    "dev": "vite",
    "build": "vite build",
    "preview": "vite preview",
    "lint": "eslint ."
  },
  "dependencies": {
    "react": "^18.3.1",
    "react-dom": "^18.3.1",
    "react-router-dom": "^6.26.2"
  },
  "devDependencies": {
    "@eslint/js": "^9.9.0",
    "@types/react": "^18.3.3",
    "@types/react-dom": "^18.3.0",
    "@vitejs/plugin-react": "^4.3.1",
    "eslint": "^9.9.0",
    "eslint-plugin-react": "^7.35.0",
    "eslint-plugin-react-hooks": "^5.1.0-rc.0",
    "eslint-plugin-react-refresh": "^0.4.9",
    "globals": "^15.9.0",
    "vite": "^5.4.1"
  }
}
```

To install the project's dependencies:
npm install

To run the project:
npm run dev

To make a project from scratch:
npm create vite@latest my-react-app -- --template react

Components

- A React Component is a reusable, self-contained piece of code that defines part of a user interface for a web application.

- A component can be referenced by another component, thus making it a child component.

- A component can have props (inputs from its parent) and state (its own internal data)

Componens: Parent / Child

HelloWorld.jsx
(child component)

```
export default function HelloWorld() {

  return <h1>Hello, world!</h1>;

}
```

In this simple example, HelloWorld is imported into App.jsx.

App.jsx
(parent component)

```
import React from 'react';

import HelloWorld from './HelloWorld';

function App() {

  return (

    <div>

          <h2>Welcome to My App</h2>

          <HelloWorld />

    </div>

  );

}

export default App;
```

Components: JSX

- JSX (JavaScript XML Syntax) is an HTML-like artifact that is returned by a component.

- This JSX code will define what HTML elements will be present on your component.

- Generally speaking, **a component will return a block of JSX**.

Let’s make a blank sandbox project

JSX

HelloWorld.jsx
(child component)

```
export default function HelloWorld() {
  return (
          <div>
            <h1>Hello, world!</h1>
            <p>I have 3 cats:</p>
            <ul>
                    <li>Rambo</li>
                    <li>Zoltan</li>
                    <li>FluffBall</li>
            </ul>
          </div>
  );
}
```

The JSX will render a div containing a h1 heading, a paragraph, and a list with three bullet points.

JSX

```
export default function HelloWorld() {
  const message = "I have 3 cats:";

  return (
    <div>
      <h1>Hello, world!</h1>
      <p>{message}</p>
      <ul>
        <li>Rambo</li>
        <li>Zoltan</li>
        <li>FluffBall</li>
      </ul>
    </div>
  );
}
```

We can parameterize segments of JSX. Note how “I have 3 cats:” is now stored in a variable.

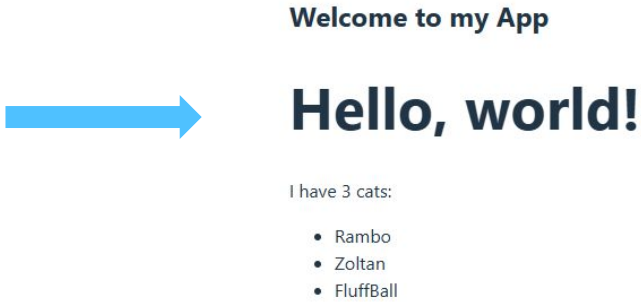
We escape the variable by escaping JSX using the curly braces then placing the variable inside.

JSX

```
export default function HelloWorld() {
  const message = "I have 3 cats:";

  return (
    <div>
      <h1>Hello, world!</h1>
      <p>{message}</p>
      <ul>
        <li>Rambo</li>
        <li>Zoltan</li>
        <li>FluffBall</li>
      </ul>
    </div>
  );
}
```

We can parameterize segments of JSX. Note how “I have 3 cats:” is now stored in a variable.

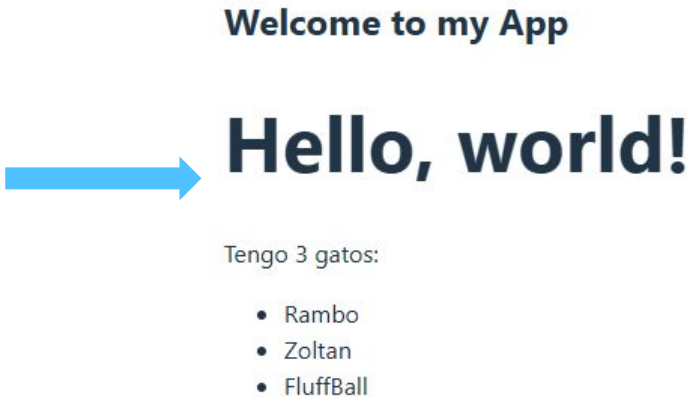
We escape JSX using the curly braces then placing the variable inside.



JSX

```
export default function HelloWorld() {
    const message = "I have 3 cats:";
    const messageSp = "Tengo 3 gatos:"
    const english = false;

    return (
      <div>
        <h1>Hello, world!</h1>
        <p>{ english ? message : messageSp }</p>
        <ul>
          <li>Rambo</li>
          <li>Zoltan</li>
          <li>FluffBall</li>
        </ul>
      </div>
    );
  }
```

We can conditionally render using ternary statements, here we display the English message if the boolean english is true, otherwise the Spanish message is printed.



JSX

```
export default function HelloWorld() {
    const message = "I have 3 cats:";
    const messageSp = "Tengo 3 gatos:"
    const english = true;

    return (
      <div>
        <h1>Hello, world!</h1>
        <p>{ english ? <strong>{message}</strong>: messageSp}</p>
        <ul>
          <li>Rambo</li>
          <li>Zoltan</li>
          <li>FluffBall</li>
        </ul>
      </div>
    );
  }
```

You can go back to writing JSX within the escaped segment. Note how we just added the tag.

But if you want to evaluate JS code again (like injecting a variable), you need to escape again, note the curly braces around message

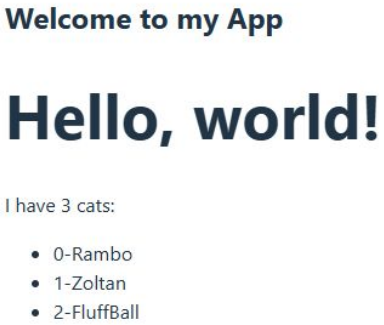
JSX

```
export default function HelloWorld() {
   const message = "I have 3 cats:";
   const catNames = ["Rambo", "Zoltan", "FluffBall"];

   return (
     <div>
       <h1>Hello, world!</h1>
       <p>{message}</p>
       <ul>
          {
             catNames.map (
                (name, index) => (
                   <li key={index}>{index}-{name}</li>
                )
             )
          }
       </ul>
     </div>
   );
}
```

We are now iterating through an array. An <li> is created per array element.

index is an automatically generated one up number

name is the placeholder for the current array element



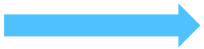
JSX

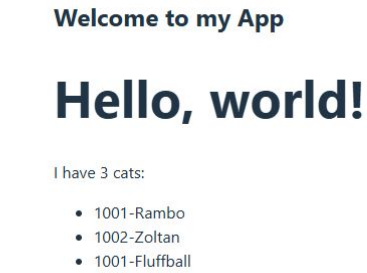
```
export default function HelloWorld() {
   const message = "I have 3 cats:";
   const catNames = [
      { id: 1001, name: 'Rambo' },
      { id: 1002, name: 'Zoltan' },
      { id: 1001, name: 'Fluffball' }
   ];

   return (
     <div>
       <h1>Hello, world!</h1>
       <p>{message}</p>
       <ul>
          {
             catNames.map (
                (cat) => (
                   <li key={cat.id}>{cat.id}-{cat.name}</li>
                )
             )
          }
       </ul>
     </div>
   );
}
```

Now in this case, we are looping through an array of objects.

cat is now a placeholder for each object.

If we know that id is going to always be unique, then we don't need to provide the key.





Let’s practice implementing JSX

Event Handling

```
function meow(message) {
    alert(message)
}

return (
  <div>
    <h1>Hello, world!</h1>
    <p>{message}</p>
    <ul>
      {
        catNames.map (
          (cat) => (
            <>
              <li key={cat.id}>{cat.id}-{cat.name}
              <button onClick={ () => meow(cat.sound) }>Make Sound</button>
              </li>
            </>
          )
        )
      }
    </ul>
  </div>
);
```

Now assume we have an extra property called sound:

```
const catNames = [
    { id: 1001, name: 'Rambo', sound: 'Rawr' },
    { id: 1002, name: 'Zoltan', sound: 'Meow' },
    { id: 1001, name: 'Fluffball', sound: 'Mlem' }
];
```

We use onClick and assign to it an anonymous function that calls the meow function.

Event Handling

```
export default function HelloWorld() {

   function doSomething(evt) {
      alert(evt.target.textContent);
   }

   return (
     <div>
       <h1>Hello, world!</h1>
       <button onClick={ (evt) => doSomething(evt)}>Click Here</button>
     </div>
   );
 }
```

The anonymous function attached to the onClick can optionally take on a single parameter

You can call this parameter whatever you want, but most people call it event, or in this case evt.

This variable is an object that contains data on the event that just happened.

From this object we can extract a lot of helpful info, in this case, the text of the JSX element that kicked off the event!

Let’s implement some event handling

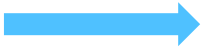
CSS

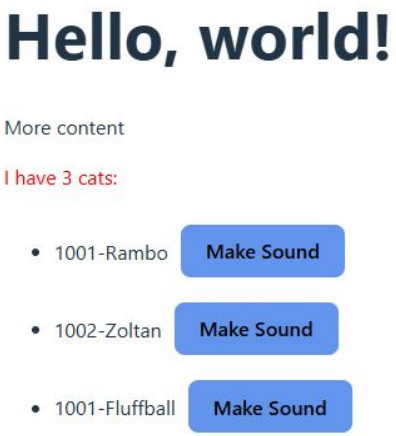
HelloWorld.jsx (child component)

```
import styles from './HelloWorld.module.css'
export default function HelloWorld() {
… // code
   return (
     <div>
       <h1>Hello, world!</h1>
       <p>More content</p>
       <p className={styles.important}>{message}</p>
       <ul>
          {
             catNames.map (
                (cat) => (
                   <>
                      <li key={cat.id}>{cat.id}-{cat.name}
                      <button onClick={ () => meow(cat.sound) }>Make Sound</button>
                      </li>
                   </>
                )
             )
          }
       </ul>
     </div>
   );
 }
```

HelloWorld.module.css

```
button {
   background-color: cornflowerblue;
   margin: 10px;
}

.important {
   color: red;
}
```





Let’s do some CSS formatting