React JS

CPSC 349 - 02

Mahitha Pasupuleti

Cal State Fullerton

Framework vs Library

- The main difference between a JavaScript library and a JavaScript framework lies in their scope and approach to development:
- **Scope**:
  1. **Library**: A JavaScript library is a collection of pre-written JavaScript code that provides specific functionality. It focuses on providing functions and utilities that can be called upon to perform tasks without dictating the overall structure of the application.
  2. **Framework**: A JavaScript framework, on the other hand, is a more comprehensive tool that provides a structured way to build web applications. It often includes libraries as part of its package but also enforces a specific architecture or design pattern.
- **Approach**:
  1. **Library**: Libraries are generally more flexible and allow developers to use them as needed within their codebase. They are often used to enhance the functionality of an existing application.
  2. **Framework**: Frameworks are more opinionated and provide a set structure or workflow for building applications. They often require developers to follow certain conventions and patterns.

Client Side Rendering(CSR)

- In CSR, the browser loads the initial HTML file, along with CSS and JavaScript files.
- The JavaScript code runs in the browser and generates the DOM (Document Object Model) structure, which defines the webpage's layout and content.
- Any subsequent interactions, such as clicking a link or submitting a form, trigger AJAX requests to the server for data, which is then used to update the DOM without reloading the entire page.
- CSR is commonly used in single-page applications (SPAs) where content is dynamically loaded and updated without full page refreshes.

Source: https://www.searchenginejournal.com/client-side-vs-server-side/482574/

Server Side Rendering(SSR)

- In SSR, the server generates the complete HTML for a webpage and sends it to the browser.
- This means that when a user requests a page, they receive a fully rendered HTML document from the server, including the initial content, styles, and scripts.
- SSR can improve the perceived performance of a website because the user sees content more quickly, especially on slower connections or devices.
- SSR is often used in traditional multi-page applications (MPAs) where content is static or changes infrequently.

Source: https://www.searchenginejournal.com/client-side-vs-server-side/482574/

CSR vs SSR

- **Performance**: CSR can lead to faster subsequent page loads because only data needs to be fetched, not the entire HTML structure. However, the initial load time might be longer as the browser needs to download and execute JavaScript. SSR provides faster initial load times because the server sends pre-rendered HTML to the browser, but subsequent interactions might be slower as the server needs to generate and send new HTML for each request.
- **SEO**: SSR is generally better for SEO because search engine crawlers can easily parse the fully rendered HTML sent by the server. CSR requires additional techniques (e.g., server-side rendering for specific routes) to ensure content is indexed properly.
- **Complexity**: CSR can be more complex to implement and manage, especially in large applications, due to the need for client-side routing, state management, and handling of asynchronous data fetching. SSR is often simpler to implement, especially for content-heavy websites, as it relies on server-side logic to render pages.

Source: https://www.searchenginejournal.com/client-side-vs-server-side/482574/

Single Page Application(SPA)

- A single-page application (SPA) is a web application or website that interacts with the user by dynamically rewriting the current web page with new data from the web server, instead of the default method of the browser loading entire new pages. This approach provides a more fluid user experience, similar to that of a desktop application.
- Key characteristics of single-page applications include:
    - **Dynamic Content Loading**
    - **Client-Side Rendering**
    - **Browser History Management**
    - **State Management**
    - **Back-End API**
    - **Smooth Transitions**

Node.js, Node package manager, and Node Virtual Machine

- As an asynchronous event-driven JavaScript runtime, Node.js is designed to build scalable network applications. Node.js is designed to be lightweight and efficient, making it ideal for building scalable network applications.
- The Node Package Manager (npm) is the default package manager for Node.js. It is used to install, share, and manage dependencies in Node.js projects. npm allows developers to easily reuse code and manage project dependencies, making it easier to build complex applications.
- Node Version Manager (nvm) is a tool that allows you to manage multiple Node.js versions on your system. It provides an easy way to install, switch between, and manage different versions of Node.js.
- npm vs npx

React JS

- ReactJS, also known as React, is a popular JavaScript library for building user interfaces. It's free and open-source, and it's maintained by Meta (formerly Facebook) along with a community of developers and companies.
- Here are some key things to know about React:
  - **Component-Based:** React applications are built using reusable components. These components can manage their own state and can be nested to create complex UIs.
  - **Declarative:** With React, you describe what the UI should look like for a given state, and React takes care of updating and rendering the UI efficiently when the state changes.
  - **Cross-Platform:** While React is primarily used for web development, you can also use libraries like React Native to build native mobile apps for iOS and Android using the same React concepts.
- Being a part of the JavaScript language, using React spawns many advantages. Products built with React are simple to scale, a single language used on the server/client/mobile side(React Native) of things grants outstanding productivity, there are workflow patterns for convenient teamwork, UI code is readable and maintainable, and more.

Source: https://www.techmagic.co/blog/why-we-use-react-js-in-the-development/

Why React?

- **Efficient Updates:** React's virtual DOM minimizes the number of times the browser needs to update the actual web page, making your application snappier.
- **Reusable Components:** Components promote code reusability, reducing development time and making code easier to maintain.
- **Large Community & Ecosystem:** React has a vast and active community of developers, along with many pre-built components and libraries available (like Redux for state management). This extensive ecosystem provides great support and resources for developers.
- **Flexibility:** React focuses on building UIs, allowing you to integrate it with other libraries or frameworks for a tailored solution.

Key concepts in React

- **Components:** React applications are built using reusable components, like Lego blocks for your UI. You can create simple components like buttons or complex ones like navigation bars.
- **JSX:** JSX is a special syntax that lets you write HTML-like structures within your JavaScript code. This makes it easier to visualize and build Uis
- **Virtual DOM:** React uses a virtual DOM, which is an in-memory representation of the actual DOM (Document Object Model, the structure of your web page). When changes occur, React efficiently updates the real DOM only with the necessary modifications. This improves performance.

Start a new react project

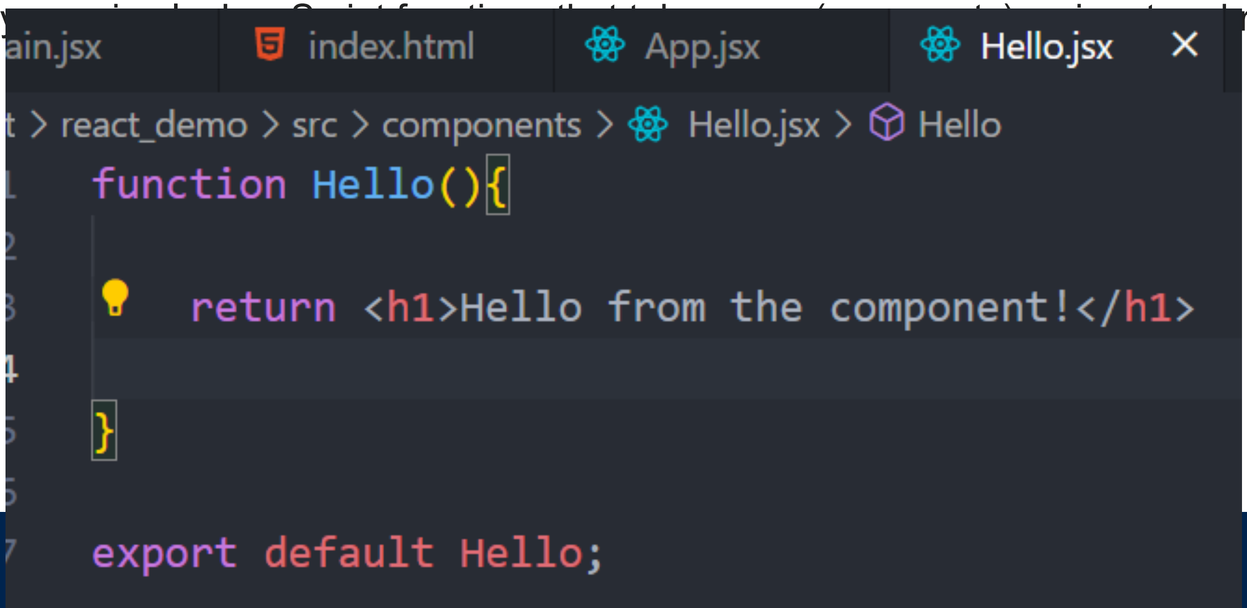
- Create a folder and navigate to this folder using the terminal. Now enter the following command,
    - npm create vite@latest
    - Now it will ask for the project name, so give your project an appropriate name,
    - Next, it will ask you to select a framework and has an option for React(we know it is a library), select React.
    - Then, it will ask for the js variant, and select Javascript.
    - Now it will create our React project
- Now once we have the project, use terminal's "cd" command to go inside the project and run the following command,
    - npm install
    - This command will install all the dependencies listed in package.json. Now lets run the project, enter command
    - npm run dev
    - After running the above command, on the terminal, you will find a url where your React App is hosted. Copy this url and paste it in a browser.
- You have now successfully run your first react app. If you have doubts, ask the instructor or visit the links given below,
- https://react.dev/learn/start-a-new-react-project
- https://vitejs.dev/guide/

JSX

- It's a syntax extension for JavaScript that allows you to write HTML-like structures within your code. JSX is not actual HTML, but rather a way to describe what the UI should look like.
- JSX makes writing React components easier to read and understand.
- It improves the separation of concerns by keeping UI structure and logic together.
- It allows for embedding JavaScript expressions within your markup for dynamic content.
- JSX tags can contain JavaScript expressions within curly braces {}. This allows for dynamic content based on data or state.

Components

- Components are reusable units of code that encapsulate both the UI structure (JSX) and the logic (JavaScript) for a specific part of your application.
- Think of them like Lego blocks - you can create various components (buttons, navigation bars, etc.) and assemble them to build the entire UI.
- Two types;
  - Functional Components(preferred)
  - Class Components
- Functional Components: They return JSX describing the UI.

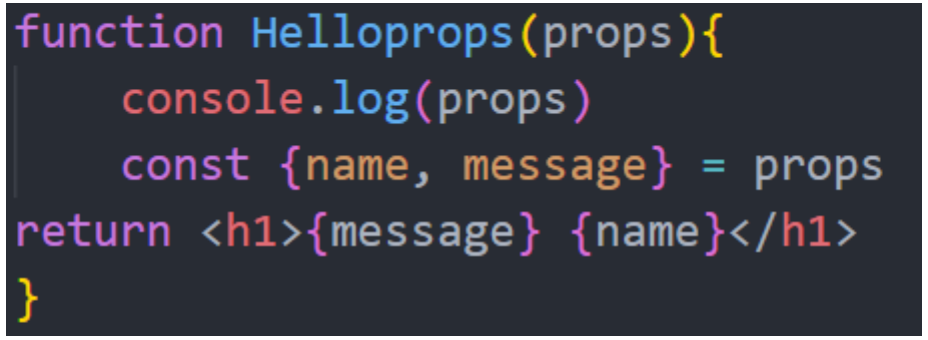


```
function Hello(){

    return <h1>Hello from the component!</h1>

}

export default Hello;
```

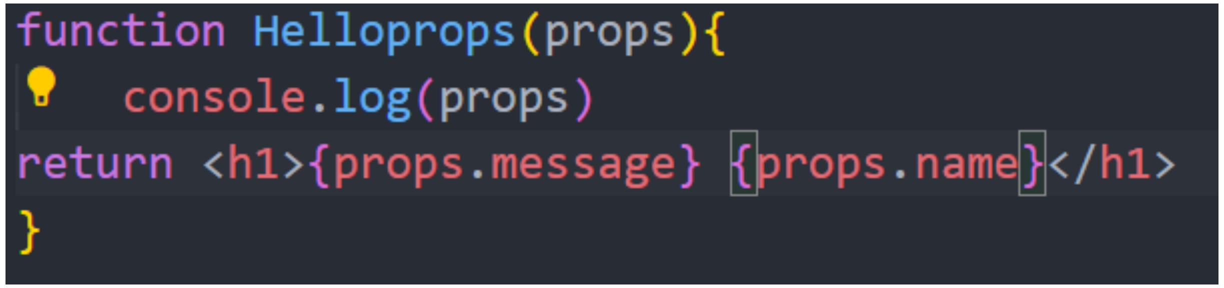
Components (cont…)

- **Props in Components:**
  - Props are like arguments passed to a function component or properties passed to a class component.
  - They provide a way to customize the behavior and appearance of components when used in different parts of your application.
  - Props are read-only within the component, promoting a unidirectional data flow.
- **Component Structure:**
  - **JSX:** The component defines the UI structure using JSX. It can include elements, attributes, and children to create the desired visual representation.
  - **JavaScript Logic (Optional):** Components can optionally contain JavaScript logic to handle user interactions (events), manage data (state), or perform side effects (fetching data).
- **Component Hierarchy:**
  - React applications are built by composing components. Components can be nested within other components to create complex UIs.
  - Parent components can pass data (props) down to their child components, but data flow is typically unidirectional (parent to child).

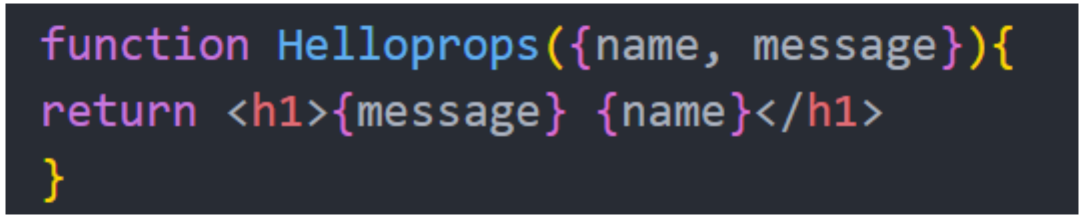
Destructuring of Props



```
function Helloprops(props){
    console.log(props)
    const {name, message} = props
return <h1>{message} {name}</h1>
}
```

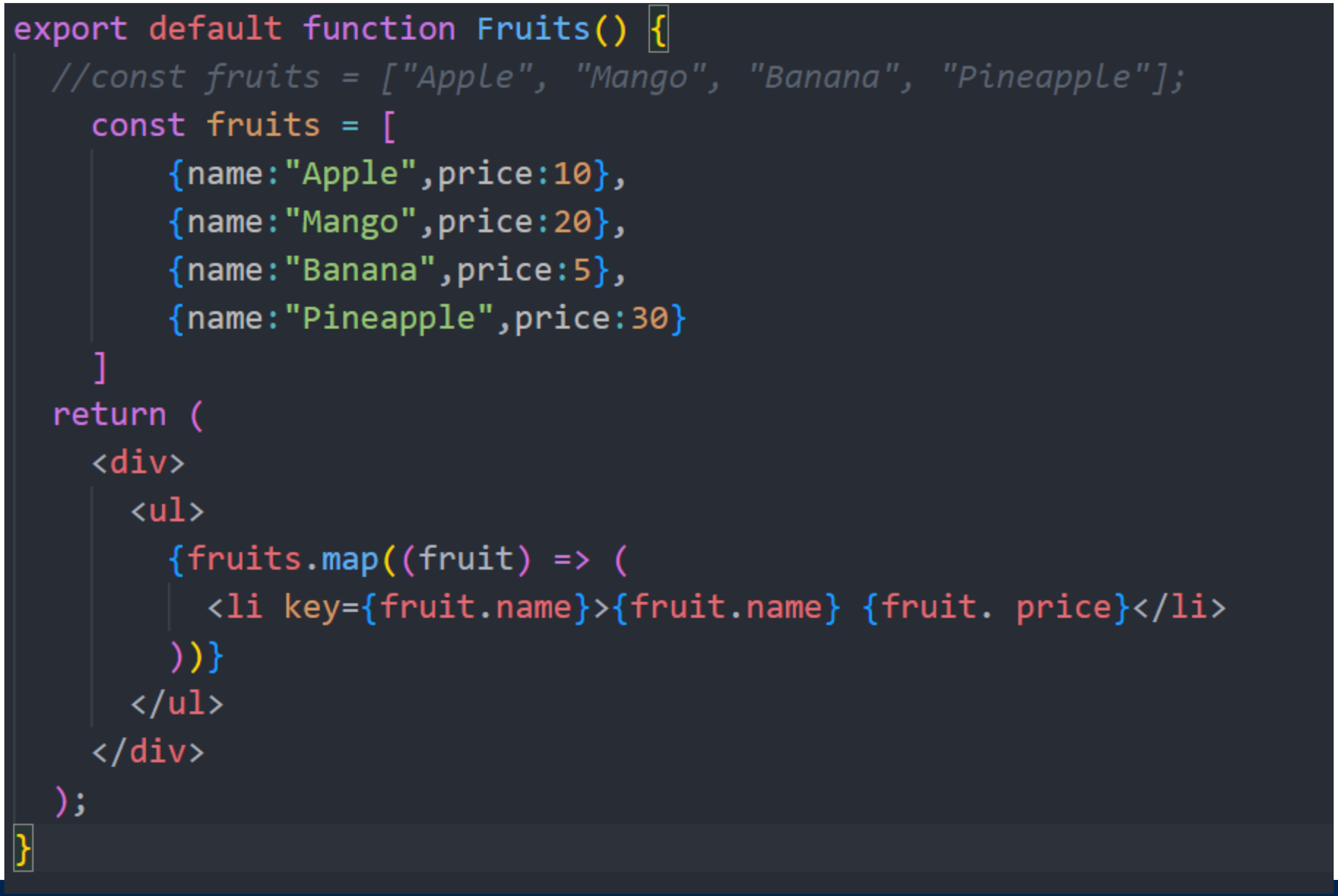


```
function Helloprops(props){
    console.log(props)
return <h1>{props.message} {props.name}</h1>
}
```



```
function Helloprops({name, message}){
return <h1>{message} {name}</h1>
}
```

Rendering of objects



```
export default function Fruits() {
  //const fruits = ["Apple", "Mango", "Banana", "Pineapple"];
    const fruits = [
        {name:"Apple",price:10},
        {name:"Mango",price:20},
        {name:"Banana",price:5},
        {name:"Pineapple",price:30}
    ]
  return (
    <div>
      <ul>
        {fruits.map((fruit) => (
          <li key={fruit.name}>{fruit.name} {fruit. price}</li>
        ))}
      </ul>
    </div>
  );
}
```

Cal State Fullerton

References

- https://www.microverse.org/blog/javascript-library-vs-javascript-frameworks-the-differences
- https://www.geeksforgeeks.org/javascript-libraries-and-frameworks/
- https://nodejs.org/en/about
- https://en.wikipedia.org/wiki/Node.js
- https://www.npmjs.com/
- https://react.dev/learn
- https://www.techmagic.co/blog/why-we-use-react-js-in-the-development/
- https://www.geeksforgeeks.org/server-side-rendering-vs-client-side-rendering-vs-server-side-generation/
- https://www.geeksforgeeks.org/difference-between-typescript-and-javascript/#