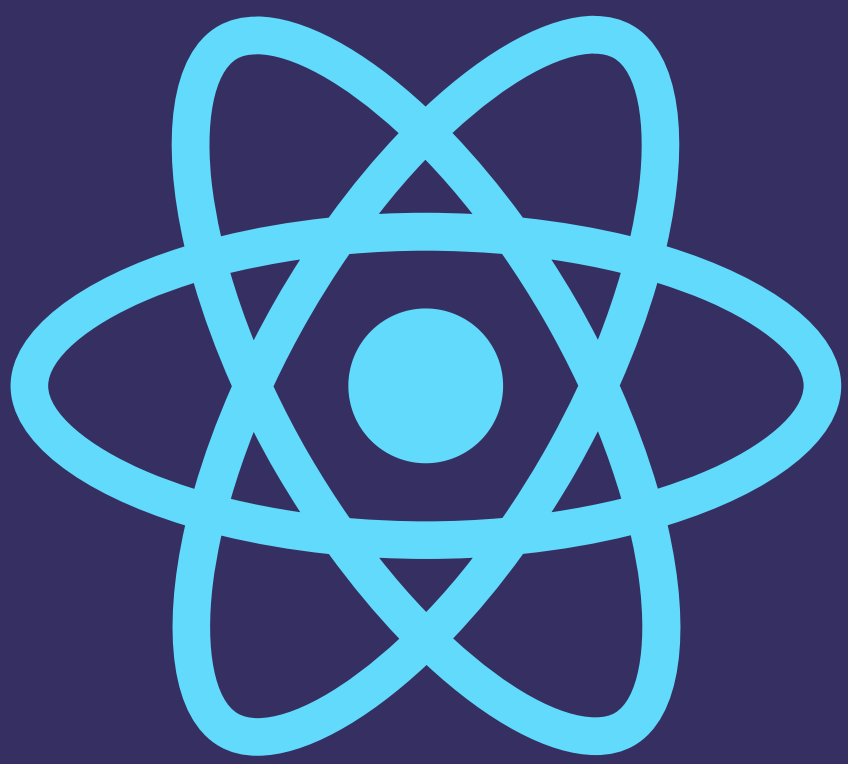

REACT VS SVELTE

WHY SVELTE IS WINNING

STANLY MEDJO
Mobile & Web developer

www.kawlodev.com

Understanding React JS and Svelte

React JS

React is a JavaScript library developed by Facebook for building user interfaces. It is component-based, allowing developers to create reusable UI components. React uses a virtual DOM that optimizes rendering by minimizing the number of updates to the actual DOM, which can be a costly operation.

Svelte

Svelte, unlike React, is a *compiler* that turns your components into efficient imperative code that directly updates the DOM. This approach eliminates the need for a virtual DOM and results in smaller bundle sizes and faster execution times. Svelte components are also easy to write with less boilerplate code compared to React.

React JS vs Svelte

In the next slides, we will compare, we will compare *React JS* and *Svelte* through a practical code example and discuss why Svelte might be a compelling choice for smaller projects.

STANLY MEDJO

Mobile & Web developer

www.kawlodev.com

1 Reactivity

In React, reactivity is managed through a state management system. When state or props change, React re-renders the component to update the view. This is typically achieved using hooks like useState and useEffect to track changes and execute side effects.

Svelte's approach to reactivity is significantly different. It operates on a more automatic and fine-grained level. In Svelte, state updates are made through simple assignments, and the framework takes care of updating the DOM directly. This results in less boilerplate and a more straightforward coding experience.

React:
Changes to state or props trigger component re-renders.

```jsx
import React, { useState } from 'react';

function Counter() {
  const [count, setCount] = useState(0);

  return (
    <button onClick={() => setCount(count + 1)}>
      Count is {count}
    </button>
  );
}
```

Svelte:
Reactivity is built into the language, making it straightforward.

```svelte
<script>
  let count = 0;
</script>

<button on:click={count += 1}>
  Count is {count}
</button>
```

STANLY MEDJO

Mobile & Web developer

2 State Management

React uses the concept of stateful functional components where state is managed via hooks such as useState.

Svelte, in contrast, integrates state management directly into its reactivity model. It uses a simple assignment to update state, and the framework ensures the DOM is updated accordingly.

React:
React uses hooks (like useState) to manage state within functional components.

```jsx
import React, { useState } from 'react';

function App() {
  const [name, setName] = useState('');

  return (
    <input type="text" value={name} onChange={(e) => setName(e.target.value)} />
  );
}
```

Svelte:
Svelte handles state reactively, with variables that update the DOM when their values change.

```svelte
<script>
  let name = '';
</script>

<input type="text" bind:value={name} />
```


STANLY MEDJO

Mobile & Web developer

www.kawlodev.com

3 Handling User Inputs

In React, handling user input typically involves creating a state variable to store the input value and updating this state in response to changes in the input field

Svelte simplifies the process of handling user inputs by using two-way data binding. This means that you can bind a variable directly to an input field, and any changes to that field will automatically update the variable, and vice versa.

React:
React typically handles user input by linking state to form elements.

```jsx
import React, { useState } from 'react';

function Form() {
  const [email, setEmail] = useState('');

  return (
    <form>
      <input
        type="email"
        value={email}
        onChange={(e) => setEmail(e.target.value)}
      />
    </form>
  );
}
```

Svelte:
Svelte simplifies two-way binding with the *bind:value* directive.

```svelte
<script>
  let email = '';
</script>

<form>
  <input type="email" bind:value={email} />
</form>
```


STANLY MEDJO

Mobile & Web developer

www.kawlodev.com

4 Conditional Rendering

In React, conditional rendering is achieved using JavaScript logic directly within the JSX. Developers typically use conditions to include or exclude certain elements based on the state or props.

Svelte handles conditional rendering with a more declarative approach using its own block syntax, such as *{#if ...}*. This syntax makes conditional statements in the template more readable and can be easier to manage, especially when the conditions or the content being toggled are complex.

React:
React uses JavaScript expressions inside JSX to handle conditional rendering.

```jsx
import React, { useState } from 'react';

function VisibilityToggle() {
  const [isVisible, setIsVisible] = useState(false);

  return (
    <div>
      <button onClick={() => setIsVisible(!isVisible)}>
        {isVisible ? 'Hide' : 'Show'}
      </button>
      {isVisible && <p>Text is visible!</p>}
    </div>
  );
}
```

Svelte:
Svelte uses a more declarative approach with its own syntax.

```svelte
<script>
  let isVisible = false;
</script>

<button on:click={() => isVisible = !isVisible}>
  {isVisible ? 'Hide' : 'Show'}
</button>
{#if isVisible}
  <p>Text is visible!</p>
{/if}
```


STANLY MEDJO

Mobile & Web developer

www.kawlodev.com

In conclusion, while *React* has its strengths in large-scale applications due to its robust ecosystem and widespread community support, *Svelte* is a compelling choice for smaller projects due to its simplicity, efficiency, and ease of use. For teams looking to quickly spin up small to medium-sized projects without the overhead of a larger framework, Svelte offers an appealing alternative.

Follow for more cool tips :)

STANLY MEDJO

Mobile & Web developer

www.kawlodev.com