Q1.

Component-Based Design breaks the entire UI into self-contained components. These components are reusable, and each of these components handling their own logic and rendering independently. An example of such will be the Gmail Composing Email or a sidebar function. When composing an Email, Gmail doesn't reload the entire page, instead it pops up an individual UI and runs individually. This would ensuring consistency and reducing code duplication.

Client-Side Rendering web pages directly in the browser using JavaScript, allowing dynamic updates without reloading the page, reducing server requests and improving interactivity. For instance, in a dashboard app, CSR enables real-time updates of charts and data dynamically without requiring a full page refresh, hence improving user experience.

Declarative programming describes what the UI should look like for a given state based on data changes, rather than manually manipulating the DOM like using framework such as React. In the example of giving a 'Like' button, we can declare how the button could render based on the current state of tasks, I could 'Like' the content by clicking the 'Like' button, or I could 'Dislike' it so to clear the Liked State.

Q3.

As described in Q1, CSR rendering web pages directly in the browser using JavaScript, allowing dynamic updates without reloading the page, reducing server requests and improving interactivity. While for SSR (Server-Side Rendering), content is pre-rendered on the server and sent as a fully formed HTML page. The browser only needs to display the pre-rendered HTML.

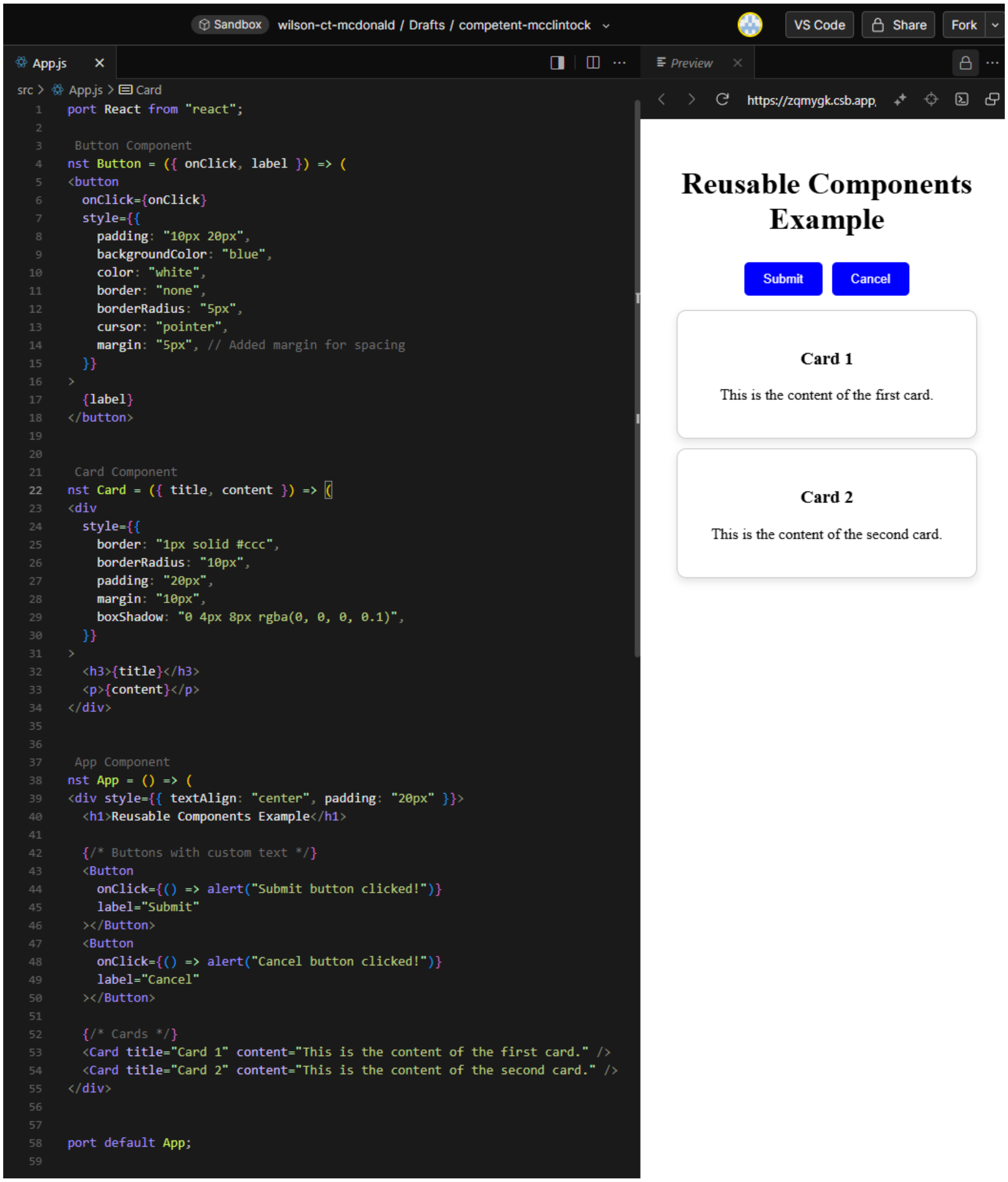
While SSR rendered on the server, CSR rendered in the browser. Because of this feature, SSR will have a faster initial loading time as pre-rendered HTML is sent while CSR might have a relatively slower initial load time as it requires JavaScript to render. However, CSR is more interactive after initial loading due to its scalability and reusability. Moreover, SSR will have a better SEO (Search Engine Optimization) performance as Search Engine can index SSR more easily whereas CSR's content loads dynamically, resulting in a less efficient performance.

Hence, in a short summary, SSR is more suitable for content-heavy sites like E-Commerce like Shopee or Blogs while CSR is more suitable when the site focuses more on User-Interaction SPA like Gmail or Facebook.

Q4.

SPA is where a web application loads a single HTML page and dynamically updates the content as the user interacts with the app, without requiring full page reloads. It would have faster user experience, more interactive and responsive with better scalability and maintainability. For instance, users experience smoother transitions between views like Google Maps. Also, once the initial page is loaded, the server only needs to provide data like JSON instead of rendering full HTML pages.

Q2.



```
port React from "react";

 Button Component
nst Button = ({ onClick, label }) => (
<button
  onClick={onClick}
  style={{
    padding: "10px 20px",
    backgroundColor: "blue",
    color: "white",
    border: "none",
    borderRadius: "5px",
    cursor: "pointer",
    margin: "5px", // Added margin for spacing
  }}
>
  {label}
</button>


 Card Component
nst Card = ({ title, content }) => (
<div
  style={{
    border: "1px solid #ccc",
    borderRadius: "10px",
    padding: "20px",
    margin: "10px",
    boxShadow: "0 4px 8px rgba(0, 0, 0, 0.1)",
  }}
>
  <h3>{title}</h3>
  <p>{content}</p>
</div>


 App Component
nst App = () => (
<div style={{ textAlign: "center", padding: "20px" }}>
  <h1>Reusable Components Example</h1>

  {/* Buttons with custom text */}
  <Button
    onClick={() => alert("Submit button clicked!")}
    label="Submit"
  ></Button>
  <Button
    onClick={() => alert("Cancel button clicked!")}
    label="Cancel"
  ></Button>

  {/* Cards */}
  <Card title="Card 1" content="This is the content of the first card." />
  <Card title="Card 2" content="This is the content of the second card." />
</div>


port default App;
```

Reusable Components Example

Card 1

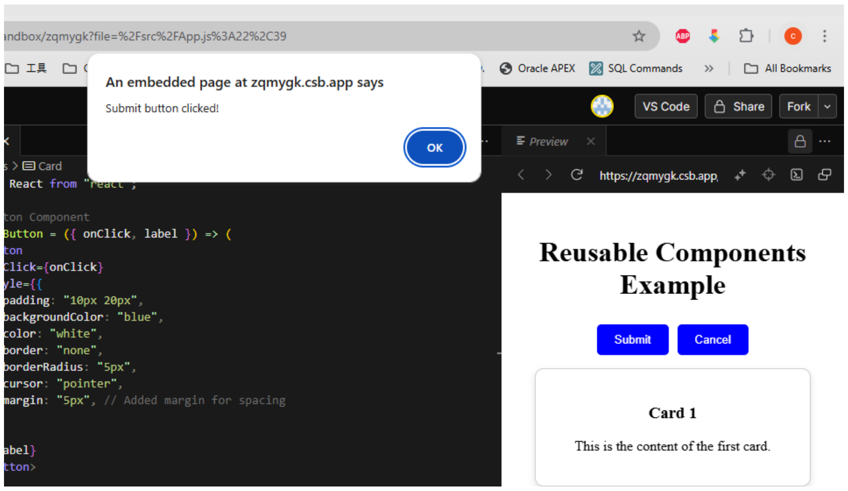
This is the content of the first card.

Card 2

This is the content of the second card.

In my example, the Button component is a functional component in React. It takes 2 props: onClick and Label. It can be reused anywhere in the application by passing different onClick handlers and Label content.

I've set the first button label to 'Submit' and a pop up will show upon clicking. And the second button will pop 'Cancel button Clicked!' upon clicking as demonstrated in my screenshots below.



An embedded page at zqmygk.csb.app says

Submit button clicked!

OK

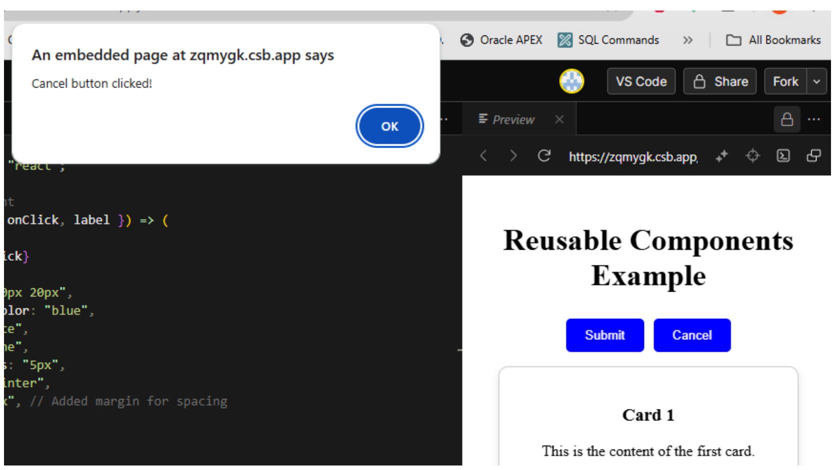
```
 React from "react";

ton Component
Button = ({ onClick, label }) => (
ton
Click={onClick}
yle={{
adding: "10px 20px",
ackgroundColor: "blue",
olor: "white",
order: "none",
orderRadius: "5px",
ursor: "pointer",
argin: "5px", // Added margin for spacing


abel}
ton>
```

Reusable Components Example

Submit Cancel

Card 1

This is the content of the first card.



An embedded page at zqmygk.csb.app says

Cancel button clicked!

OK

```
"react";

t
 onClick, label }) => (

ick}

px 20px",
lor: "blue",
e",
ne",
: "5px",
inter",
", // Added margin for spacing
```

Reusable Components Example

Submit Cancel

Card 1

This is the content of the first card.

For card components part, it takes in 2 props: title and the content. It can be reused to display different cards with varying titles and contents. In my example, simply input title and contents to the 2 props, it will then display the inputs like in my screenshots.