✅ Do’s and ❌ Don’ts for React Frontend Mock Test & Milestone Assessment

✅ Do’s

1️⃣ Read the Problem Statement Carefully

✔ Ensure you understand the UI behavior, state management, API calls, and event handling expected in the problem.
✔ Follow the **input/output format** and **component structure** as mentioned.
✔ Check for specific **props, state values, event handlers, or lifecycle methods** required.

🔹 **Example:**
If the problem states that clicking a button should fetch data and display it, don’t manually render static data; implement an API call using `useEffect()`.

```
useEffect(() => {

  fetch("https://api.example.com/data")

    .then(response => response.json())

    .then(data => setData(data));

}, []);
```

2️⃣ Use the Provided Boilerplate Code

✔ Work within the given component structure and **do not modify function names or file names**.
✔ Implement logic **inside the designated functions/hooks** as required.

🔹 **Example:**
If a function `handleClick()` is pre-defined for handling button clicks, write logic inside it instead of creating a new function.

```
function handleClick() {

  setCount(count + 1); // Correct

}
```

This file is meant for personal use by test_av2u1y35w4@greatlearning.in only.
Sharing or publishing the contents in part or full is liable for legal action.

❌ **Avoid** creating a new function like this:

```
function increaseCount() { // Wrong

  setCount(count + 1);

}
```

3️⃣ Use Correct State Management

✔ Use **useState/useEffect correctly** to manage state.
✔ If the problem specifies **Redux** or **Context API**, use them accordingly.

🔹 **Example:**
If the UI requires dynamically updating a list when a button is clicked:

```
const [items, setItems] = useState([]);


function addItem() {

  setItems([...items, "New Item"]);

}
```

4️⃣ Test with Provided Test Cases & Edge Cases

✔ Ensure your implementation works for both **visible and hidden test cases**.
✔ Handle **empty inputs, large datasets, and incorrect values**.

🔹 **Example:**
If a component displays user data but the API fails, handle errors:

```
fetch("https://api.example.com/users")
```

This file is meant for personal use by test_av2u1y35w4@greatlearning.in only.
Sharing or publishing the contents in part or full is liable for legal action.

```
  .then(res => res.json())

  .then(setUsers)

  .catch(error => console.error("Error fetching data:", error));
```

5️⃣ Follow JSX & Component Best Practices

✔ Use **camelCase for props** and functions.
✔ Ensure **semantic HTML elements** and proper component structuring.
✔ Pass **props correctly** and use **prop-types** if required.

🔹 **Example:**

```
function Card({ title, description }) {

  return (

    <div className="card">

      <h2>{title}</h2>

      <p>{description}</p>

    </div>

  );

}
```

6️⃣ Submit Before Time Ends

✔ Keep track of the **timer** to avoid last-minute rush.
✔ **Run all test cases** and check for failures before submission.

❌ Don’ts

1️⃣ Don’t Modify the Provided Component Structure

This file is meant for personal use by test_av2u1y35w4@greatlearning.in only.
Sharing or publishing the contents in part or full is liable for legal action.

🚫 **Wrong Approach:**

```
function NewComponent() {  // Avoid creating new files/components unless
asked.

  return <h1>Hello</h1>;

}
```

✅ **Correct:** Use the provided component and **add logic inside it**.

2️⃣ Don't Hardcode Data or Outputs

🚫 **Wrong Approach:**

```
return <div>Name: John Doe</div>;  // Hardcoded output
```

✅ **Correct Approach:** Fetch and display dynamic data:

```
return <div>Name: {user.name}</div>;
```

3️⃣ Don't Use Inline Styles Instead of CSS Modules or Tailwind (If Mentioned)

🚫 **Wrong:**

```
<div style={{ color: "red", fontSize: "20px" }}>Hello</div>;
```

✅ **Correct:**

This file is meant for personal use by test_av2u1y35w4@greatlearning.in only.
Sharing or publishing the contents in part or full is liable for legal action.

```
<div className="text-red-500 text-xl">Hello</div>; // Using Tailwind (if provided)
```

4️⃣ Don’t Forget to Handle API Errors

🚫 **Wrong:**

```
fetch("https://api.example.com")
  .then(response => response.json())
  .then(setData);
```

✅ **Correct:**

```
fetch("https://api.example.com")
  .then(response => response.json())
  .then(setData)
  .catch(error => console.error("API call failed", error));
```

5️⃣ Don’t Ignore Event Handling or Button Clicks

🚫 **Wrong:** Clicking a button does nothing.

```
<button>Add</button>
```

✅ **Correct:** Attach an event handler.

```
<button onClick={addItem}>Add</button>
```

This file is meant for personal use by test_av2u1y35w4@greatlearning.in only.
Sharing or publishing the contents in part or full is liable for legal action.

6️⃣ Don’t Refresh the Page or Navigate Away

🚫 Avoid refreshing the page as it may **reset progress or auto-submit incomplete answers**.

7️⃣ Don’t Assume Default Input Handling

🚫 **Wrong:** If user input is required, don’t assume default values.

```
const [input, setInput] = useState("");  // Ensure controlled inputs
```

8️⃣ Don’t Ignore Console Errors or Warnings

🚫 **Wrong:** Ignoring `React Hook useEffect has a missing dependency` warning.
✅ **Correct:** Always pass dependencies properly.

```
useEffect(() => {

  fetchData();

}, [dependency]); // Add dependencies as required
```

🚀 Final Tips

✔ **Test Code with Sample Inputs** before submission.
✔ **Follow React Best Practices** and maintain clean code.
✔ **Handle Edge Cases** like empty inputs, API failures, and UI re-renders.
✔ **Manage Time Efficiently** and avoid last-minute debugging.

This file is meant for personal use by test_av2u1y35w4@greatlearning.in only.
Sharing or publishing the contents in part or full is liable for legal action.