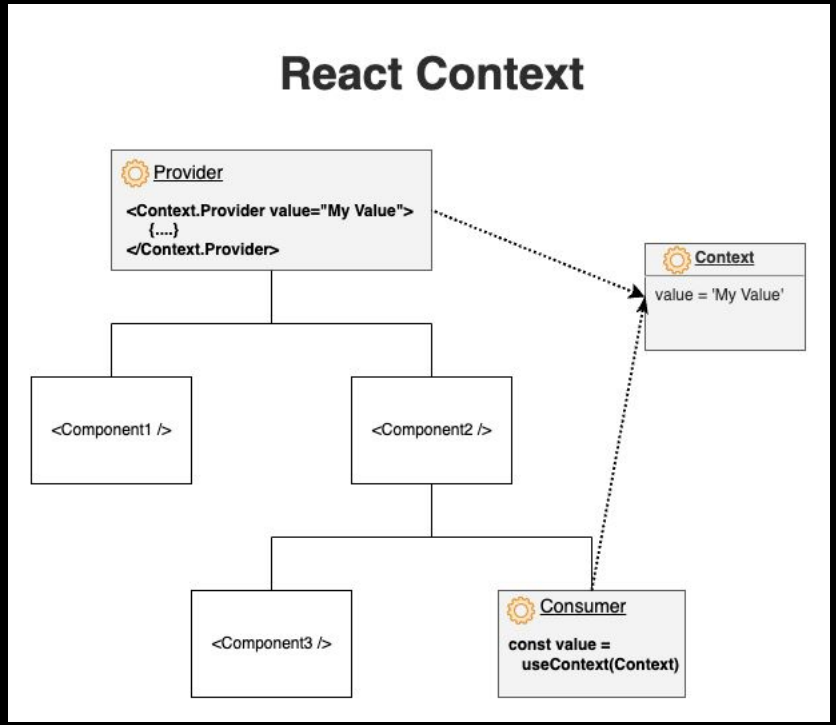
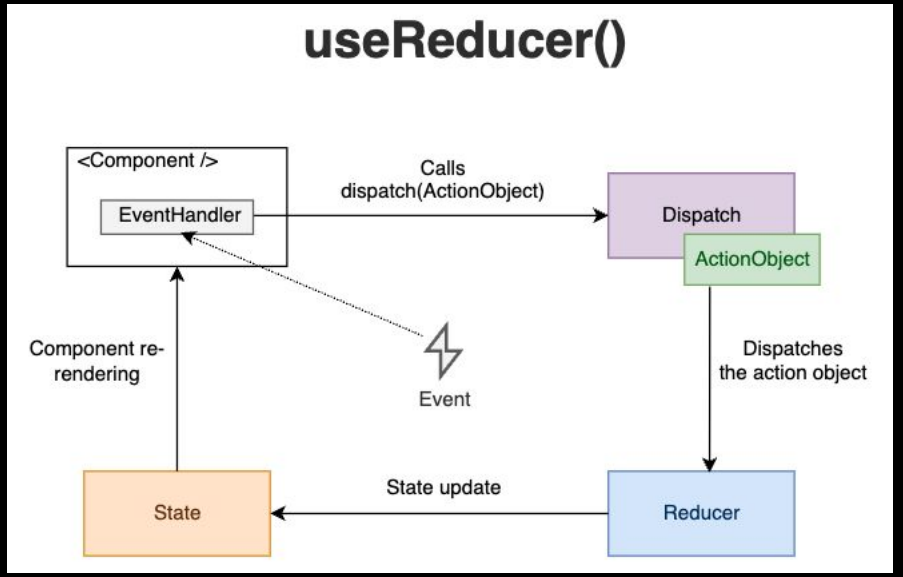
Context API

1. Prop Drilling: Context API addresses prop drilling; component composition is an alternative.
2. Folder Setup: Use a store folder for context files.
3. Initialization: Use React.createContext with initial state and export it.
4. Provider: Implement with contextName.Provider in components.
5. Access Value: Use the useContext hook.
6. Dynamic Data: Combine context value with state.
7. Export Functions: Context can also export functions for actions
8. Logic Separation: This helps keep the UI and business logic separate from each other.

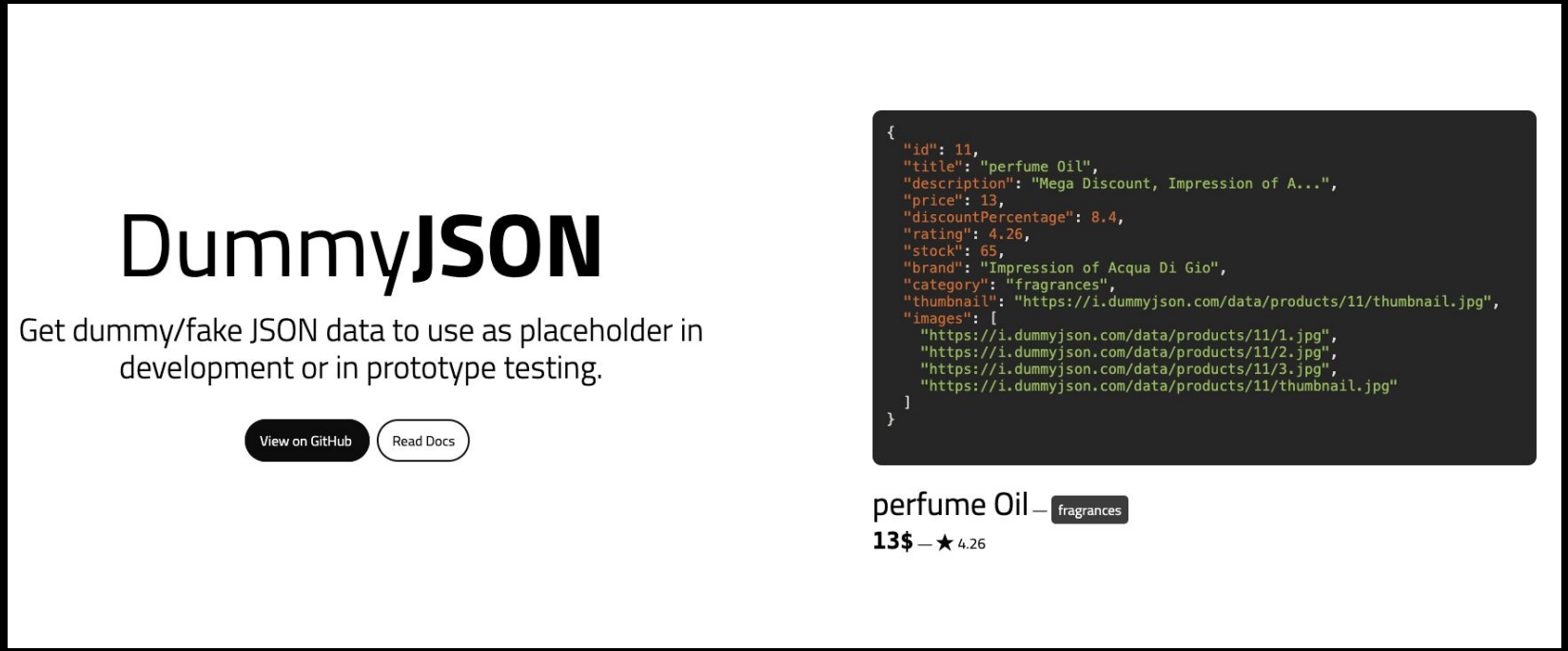


Use Reducer

1. useReducer is a hook in React that offers more control over state operations compared to useState, especially for complex state logic.
2. Components: It involves two main components:
   - Reducer: A pure function that takes the current state and an action and returns a new state.
   - Action: An object describing what happened, typically having a type property.
3. Initialization: It's invoked as `const [state, dispatch] = useReducer(reducer, initialState)`.
4. Dispatch: Actions are dispatched using the dispatch function, which invokes the reducer with the current state and the given action.
5. Use Cases: Particularly useful for managing state in large components or when the next state depends on the previous one.
6. Predictable State Management: Due to its strict structure, it leads to more predictable and maintainable state management.

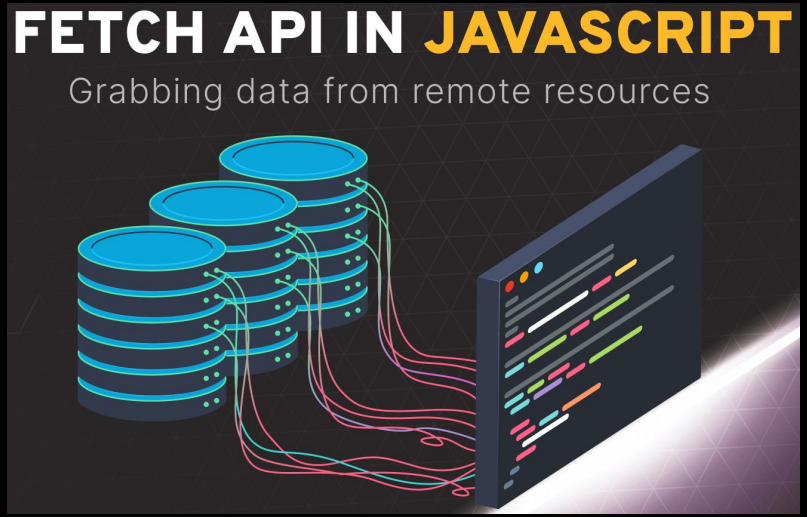


Introducing Dummy API



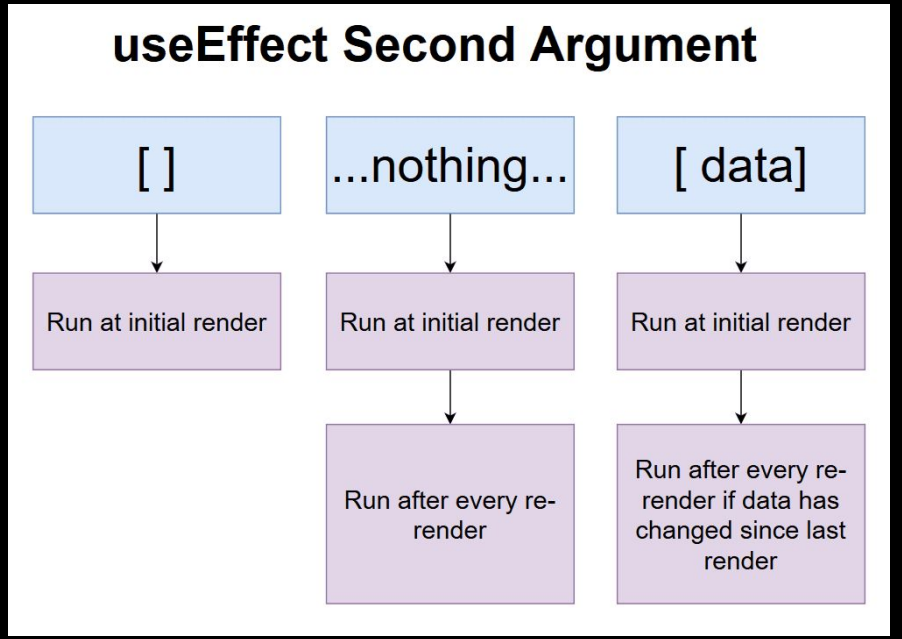
Data fetching using Fetch

1. fetch: Modern JavaScript API for network requests.
2. Promise-Based: Returns a Promise with a Response object.
3. Usage: Default is GET. For POST use method: 'POST'
4. Response: Use .then() and response.json() for JSON data.
5. Errors: Doesn't reject on HTTP errors. Check response.ok.
6. Headers: Managed using the Headers API.



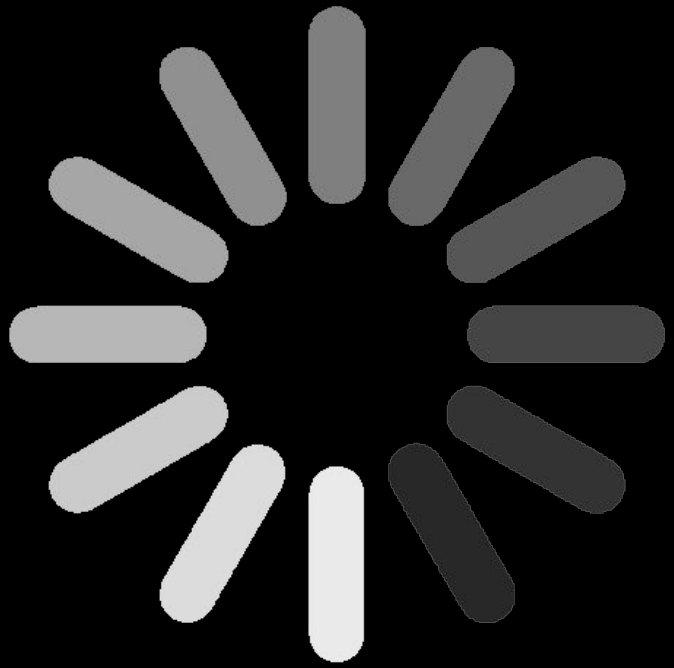
The useEffect Hook

1. In function-based components, useEffect handles side effects like data fetching or event listeners.
2. useEffect runs automatically after every render by default.
3. By providing a dependency array, useEffect will only run when specified variables change. An empty array means the effect runs once.
4. Multiple useEffect hooks can be used in a single component for organizing different side effects separately.

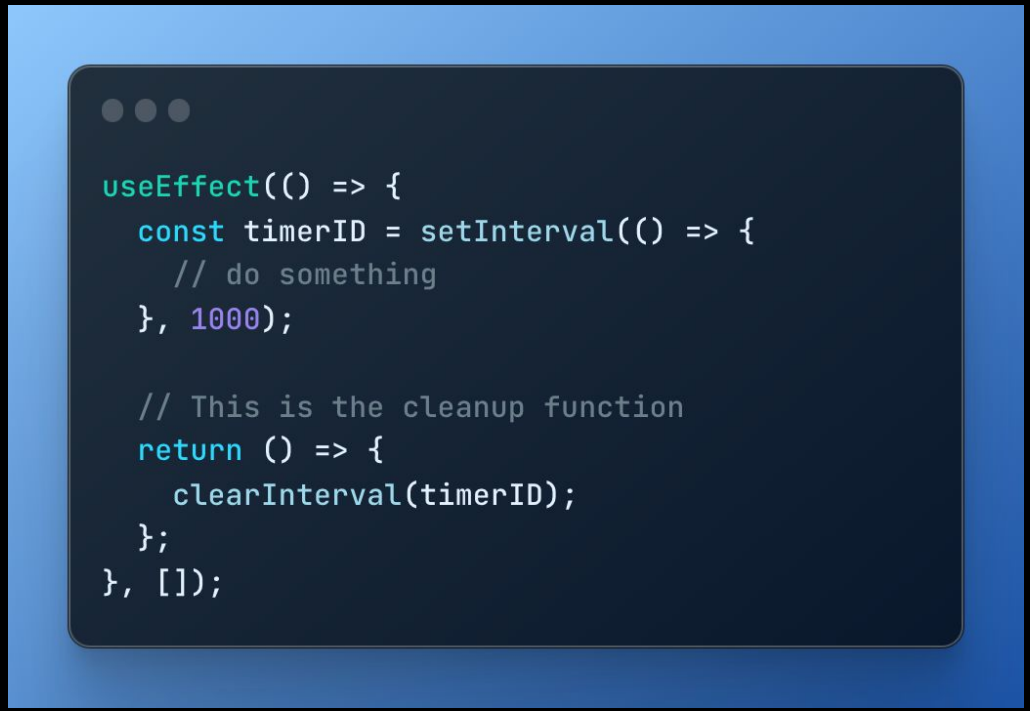


React

Handling Loading State



The useEffect Hook Cleanup



```
useEffect(() => {
  const timerID = setInterval(() => {
    // do something
  }, 1000);

  // This is the cleanup function
  return () => {
    clearInterval(timerID);
  };
}, []);
```

Returning a function from `useEffect` allows for cleanup, ideal for removing event listeners.

Advanced useEffect

React

Junior

```
useEffect(() => {
  fetch(`/api/users/${id}`)
    .then((res) => res.json(
    .then((data) => {
      setUser(data);
    });
}, [id]);
```

Pro

```
useEffect(() => {
  const controller = new AbortController();
  const signal = controller.signal;

  fetch(`/api/users/${id}`, { signal })
    .then((res) => res.json())
    .then((data) => {
      setUser(data);
    });

  return () => {
    controller.abort();
  };
}, [id]);
```