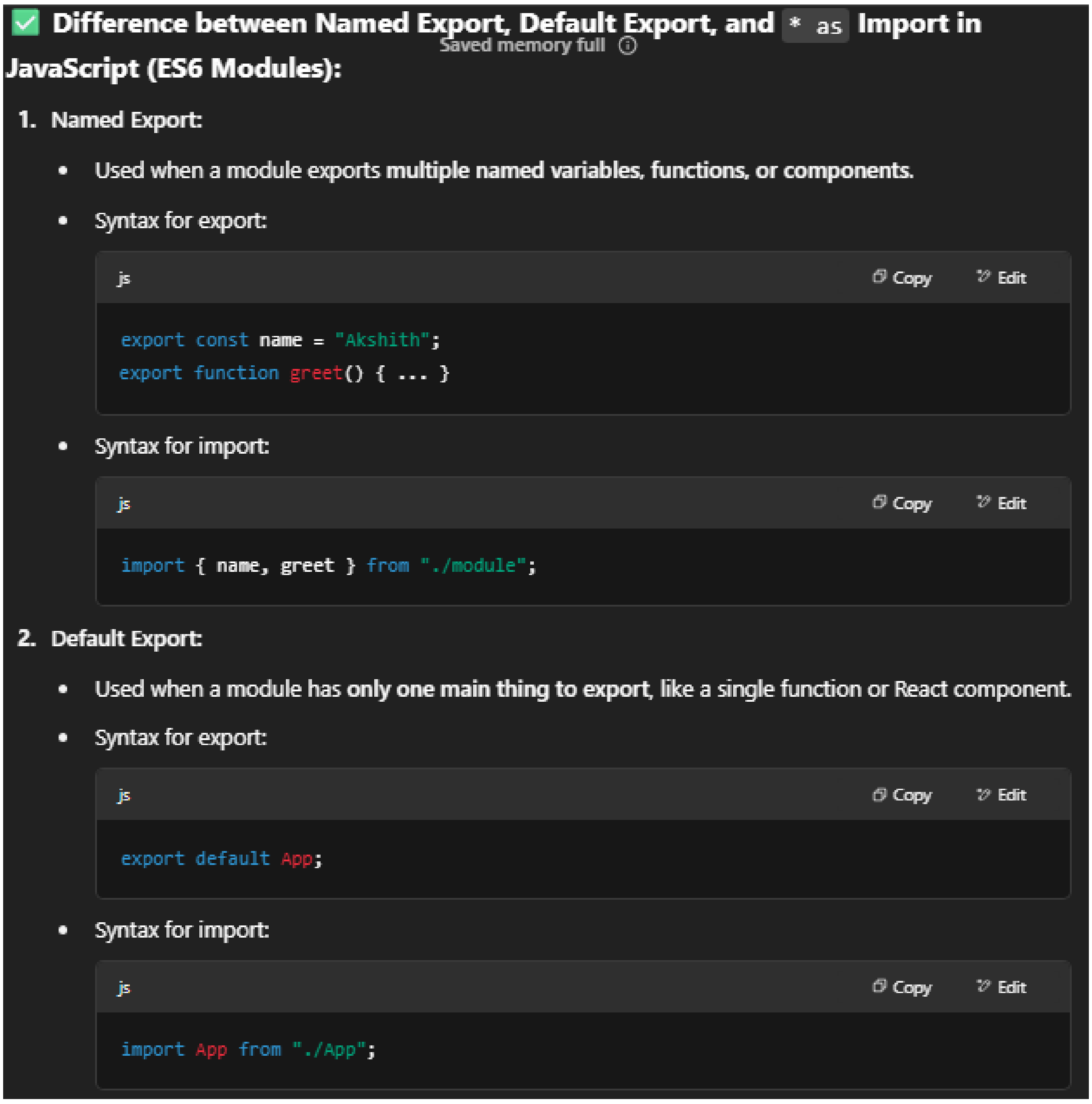


✅ Difference between Named Export, Default Export, and `* as` Import in JavaScript (ES6 Modules):

1. **Named Export:**
   - Used when a module exports **multiple named variables, functions, or components.**
   - Syntax for export:

```js
export const name = "Akshith";
export function greet() { ... }
```

   - Syntax for import:

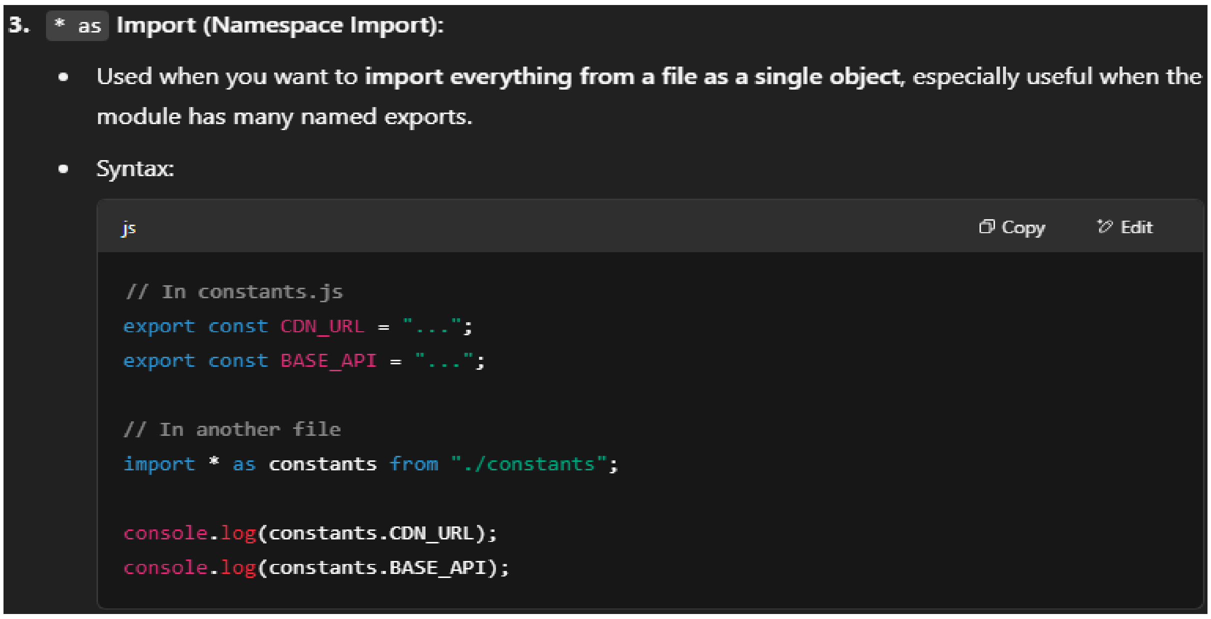
```js
import { name, greet } from "./module";
```

2. **Default Export:**
   - Used when a module has **only one main thing to export**, like a single function or React component.
   - Syntax for export:

```js
export default App;
```

   - Syntax for import:

```js
import App from "./App";
```

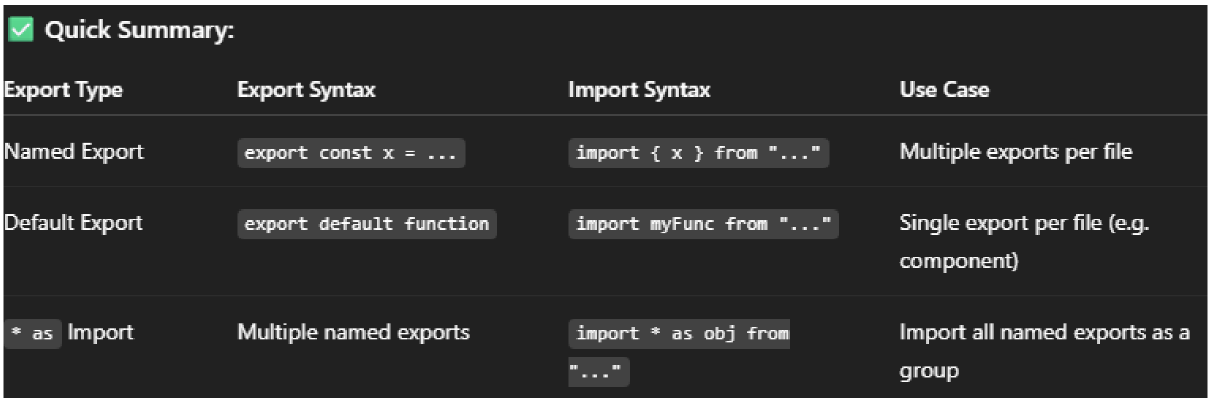


3. `* as` **Import (Namespace Import):**
   - Used when you want to **import everything from a file as a single object**, especially useful when the module has many named exports.
   - Syntax:

```js
// In constants.js
export const CDN_URL = "...";
export const BASE_API = "...";

// In another file
import * as constants from "./constants";

console.log(constants.CDN_URL);
console.log(constants.BASE_API);
```



✅ **Quick Summary:**

Export Type	Export Syntax	Import Syntax	Use Case
Named Export	`export const x = ...`	`import { x } from "..."`	Multiple exports per file
Default Export	`export default function`	`import myFunc from "..."`	Single export per file (e.g. component)
`* as` Import	Multiple named exports	`import * as obj from "..."`	Import all named exports as a group

2. What is the importance of config.js file

A config.js file is used to store **configuration data** that the application might need across different modules. This includes values like **API endpoints, image URLs, keys, environment-specific variables, constants, etc.** Keeping such values in a central config.js file makes the code **easier to maintain, scalable, and more organized**. It allows developers to update configurations in one place without modifying multiple files throughout the codebase.

3. What are React Hooks?

React Hooks are **special functions provided by React** that let you "hook into" React features such as **state management, lifecycle methods, and side effects** from functional components. Before hooks, these features were only available in class components. Hooks simplify code, improve reusability, and allow developers to build components using functions rather than classes.

They are called “hooks” because they let you **hook into React’s internal features** like state (useState), lifecycle (useEffect), context (useContext), and more.

4. Why do we need a useState Hook?

The useState hook is used to **create and manage state variables** in a functional component. In React, when a state variable changes, React **automatically re-renders** the component to reflect the updated data in the UI.

The useState hook returns a **state variable** and a **setter function**. The setter function is the **only correct way to update state** so that React knows something has changed and can trigger a re-render. If state is updated without using the setter, React will not detect the change and the UI will not update.

React tracks changes only through **state and props**, so using useState is essential when we want a component to respond dynamically to user interactions, input changes, API calls, or any internal logic that modifies data.