# AeroAspire - SDE Intern

**Gokul Krishna S**

### Week 2 – Day 4 (October 3)

**Questions/Reflections :**

1. Describe how client-side routing works (history API or hash routing).

    - History API: Modern React Routers (like <BrowserRouter>) use HTML5's History API (pushState, replaceState) to update the URL path (e.g., /profile) without reloading the page. Navigation triggers React to swap out components based on the path, maintaining SPA behavior.
    - Hash Routing: <HashRouter> updates the part of the URL after # (e.g., /#/about). Changing the hash doesn't reload the page. It works well when deploying to static file servers, as the hash is never sent to the server.

2. What happens when you navigate: how React Router matches route and renders components.

    - When a navigation occurs (e.g., click a Link), React Router looks at current URL.
    - It matches the URL path against your <Route> definitions using pattern matching.
    - If a match is found, the associated component is rendered. If not, a fallback route like 404 displays.
    - Matching also extracts route parameters (e.g., /users/:id grabs the id from the path).

3. How to pass params or query params; nested routes.

    - Route Params: Defined in the path (e.g., /products/:productId), accessed via useParams() in the component.

- Query Params: Added to URLs (/search?query=cabbage). You can get them via useLocation() and URLSearchParams.
- Passing: Use navigation helpers like history.push('/profile/7?mode=view').
- Nested Routes: Define <Route> inside another <Route>. Use match.path for paths and match.url for links to keep nesting correct.

4. What is the flow: writing to localStorage → reading on app startup?

    - Write: Use localStorage.setItem('key', JSON.stringify(data)) after actions or state changes.
    - Read on Startup: On initial render (usually in a useEffect with an empty dependency array), read and parse stored values: const data = JSON.parse(localStorage.getItem('key')). Then update state with this data if available.
    - This ensures app state persists across sessions if refreshed or reopened.

5. How do you sync state with localStorage safely (e.g. updates, JSON parse/stringify)?

    - Updates: Use a useEffect hook watching your state. Whenever state changes, update localStorage with the new value using JSON.stringify.
    - Startup Read: On mount, read and JSON.parse the value to restore state.
    - Custom hooks: Extract to useLocalStorage for cleaner code.
    - Always stringify: Use JSON.stringify when saving, JSON.parse when loading, to handle objects/arrays safely.

6. What performance / size concerns with storing too much in localStorage?

    - Size limits: Browsers limit localStorage to about 5MB per origin. Storing large datasets or big JSON documents isn’t recommended and can cause slower reads/writes.

- Blocking: localStorage is synchronous; excessive reads/writes can block the main thread and reduce app responsiveness.
- Security: localStorage isn’t private—don’t store sensitive data like passwords or secrets.