Project Weekly Progress Report Agile – Scrum

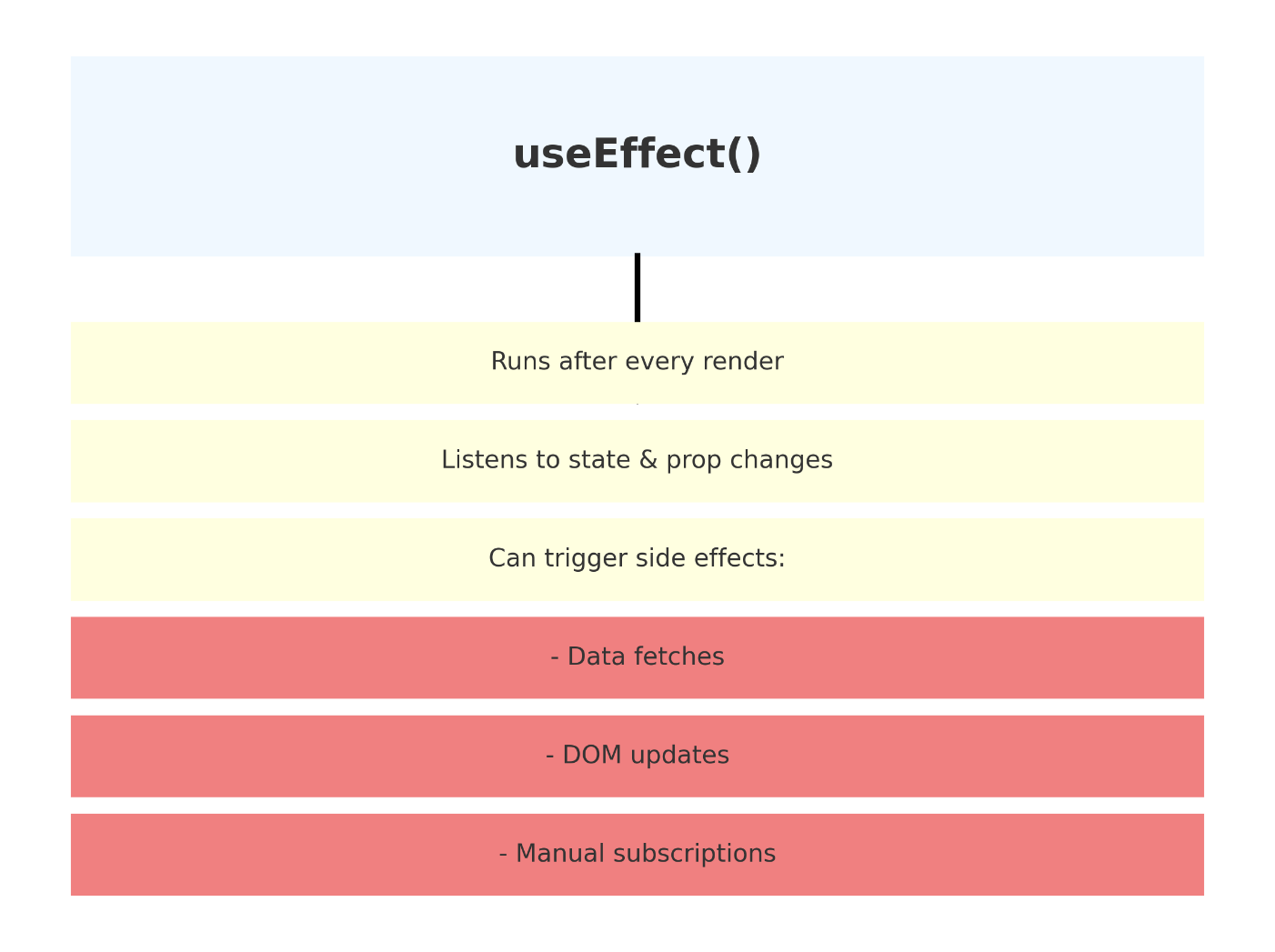
|  |  |
| --- | --- |
| **Semester** | **W2023, SEM-2** |
| **Course Code** | **AML-2404** |
| **Section** | **Section 2** |
| **Group Name** | **D** |
| **Student names/Student IDs** | **Jash Vaghasiya - C0884733**  **Nivedini Kathagonda - C0872720**  **Keval Parmar - C0882386**  **Monil Rupawala - C0882370**  **Sai Divya Madhuri Guntupalli - C0882360** |
| **Reporting Week** | **12** |
| **Team Lead for the reporting week** | **Jash Vaghasiya** |

1. **Progress Made in Reporting** **Week:**

Significant progress was made this week in the frontend development of our application. The change of frontend material to be dynamic is a crucial highlight of this innovation. We successfully developed a dynamic user side panel and content for the main container by harnessing the capabilities of React. This allows the interface to modify and show content based on user activities and preferences, resulting in a more responsive and personalised experience for our users.

We used React hooks, namely the useEffect hook, to create this dynamic functionality. The useEffect hook is used for executing side effects in function components. It runs after the render is committed to the screen, ensuring that our content refreshes smoothly without requiring excessive re-rendering or disrupting the user experience. By incorporating this hook into our frontend structure, we can manage and react to state changes more quickly, resulting in a smoother and more intuitive user experience.

The structure of the useEffect hook is depicted in the following conceptual image:

****

**2. Difficulties Encountered in Reporting Week:**

We encountered a huge difficulty in our application's performance and stability, which was mostly caused by the React framework on the frontend. We discovered that the useEffect hook was being called more frequently than expected. The useEffect hook was triggered repeatedly and uncontrollably, resulting in an excessive number of requests being sent to our backend services. Each time the hook was used, it launched an API call, which quickly depleted our server's resources, causing it to fail. Such uncontrolled behaviour not only disturbed the normal operation of the backend, but also affected the overall user experience due to higher load times and rare outages.

To address this technical issue, we investigated the root causes of the frequent useEffect hook invocations. We discovered that the hook's dependents array, which governs when it should be re-invoked, was either wrongly defined or was catching regularly changing information. We immediately began refactoring our React components, making certain that the dependencies supplied to useEffect were both necessary and stable. We also used debouncing and throttling techniques to minimise the number of calls made to the backend. By doing so, we were able to successfully address the problem, restoring stability to our backend services and improving frontend responsiveness.