| **Purpose / Use-case** | **Tool** | **Strengths** | **Weaknesses / Not the best for this project** |
| --- | --- | --- | --- |
| **Inline code comments & API docs** | **JSDoc** | Standard for JavaScript/TypeScript. Easy to add above functions. Can auto-generate docs from comments. | Only for code-level docs, not design/UX. |
| **Frontend UI components** | **Storybook** | Great for showcasing reusable React components visually. Designers + devs can test props, states. | Overkill if project is small or doesn’t need a UI component library. |
| **Team / product docs** | **Notion / Confluence** | Good for writing project overviews, requirements, user flows, or visual styles. Easy collaboration. | Not tied to code. Doesn’t auto-generate from source. |
| **Full documentation portal** | **Docusaurus** | Can combine API (via JSDoc), guides, and design docs in one website. Great for larger projects. | Needs setup/maintenance. Too heavy for a student project. |

**Decision: So from the above comparison we can take decision that JSDocs will be suitable for our project documentation. As it is suitable to make comments inline code. It generates auto .html file . It is compatible with our project language nextjs.**