NFR

Cscc01 project
Driven Devs

Non-functional requirements for our web app Roomzi:

1. Security
   Prisma is designed with security in mind, primarily through its use of parameterized queries, type safety, and secure data handling practices. These features help prevent common vulnerabilities like SQL injection and promote secure database interactions. Additionally, Prisma Cloud offers advanced threat protection, including runtime defense against malware and other threats.
   Trade-off 1: Prisma ORM is an abstraction. As such, an inherent trade-off of Prisma ORM is a reduced amount of control in exchange for higher productivity. This means, the Prisma Client API might have less capabilities in some scenarios than you get with plain SQL.
   The complexity of our database is not affected by this limitation.
   Trade-off 2:
   While tools like the nexus-plugin-prisma and typegraphql-prisma allow you to quickly generate CRUD operations for your Prisma ORM models in a GraphQL API, these approaches still require you to set up your GraphQL server manually and do some work to expose GraphQL queries and mutations for the models defined in your Prisma schema.
   If you want to get a GraphQL endpoint for your database out-of-the box, other tools might be better suited for your use case.
   This limitation, as it extends to the use of our project, is acceptable.

2. Accessibility
   Radix UI is designed to be accessible to people with disabilities, and it includes a number of features that make it easy to use for everyone. Radix UI ensures accessibility by default. It adheres to WAI-ARIA guidelines,

provides keyboard navigation, and manages focus states. When customizing components, it’s essential to maintain accessibility by following best practices and leveraging the accessibility features provided by Radix UI. Radix UI components can be styled using various approaches, such as CSS, CSS-in-JS libraries, or utility-first frameworks like Tailwind CSS. Best practices for styling include using semantic class names, maintaining a consistent design system, and considering accessibility. Furthermore, Radix UI allows for advanced customization by extending components with custom behavior, adding event listeners, custom props, and refs. It supports both controlled and uncontrolled components, giving developers flexibility in managing state.
Trade off 1: Because the components are unstyled, developers need to invest time in styling them to match their desired design. We are okay with taking the time to develop the required skills to use this technology.
Trade off 2: Like any third-party library, Radix UI introduces a dependency, and updates or changes to the library could potentially impact the project. We have not had this problem so far.

3. Performance

   We have chosen to use Vite mainly for its speed in starting up the server. Vite improves the dev server start time by first dividing the modules in an application into two categories:
   Dependencies are mostly plain JavaScript that do not change often during development.
   Source code often contains non-plain JavaScript that needs transforming (e.g. JSX, CSS or Vue/Svelte components), and will be edited very often.
   In some bundlers, the dev server runs the bundling in memory so that it only needs to invalidate part of its module graph when a file changes, but it still needs to re-construct the entire bundle and reload the web page. Reconstructing the bundle can be expensive, and reloading the page blows away the current state of the application. This is why some bundlers support Hot Module Replacement (HMR): allowing a module to "hot

replace" itself without affecting the rest of the page. When a file is edited, Vite only needs to precisely invalidate the chain between the edited module and its closest HMR boundary (most of the time only the module itself), making HMR updates consistently fast regardless of the size of your application.
Trade off 1: While Vite excels at HMR, the initial page load in development can be slower due to the on-demand processing of modules and HTTP request overhead. We can tolerate this delay in the development phase.
Trade off 2: Vite might have compatibility issues with older libraries or projects relying on CommonJS modules. We are not working with legacy code, as such, we will never encounter this issue.