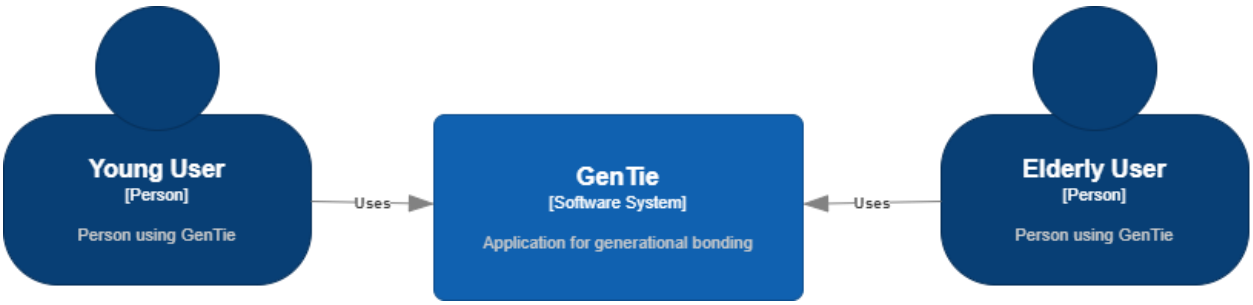
Software architecture document

Introduction

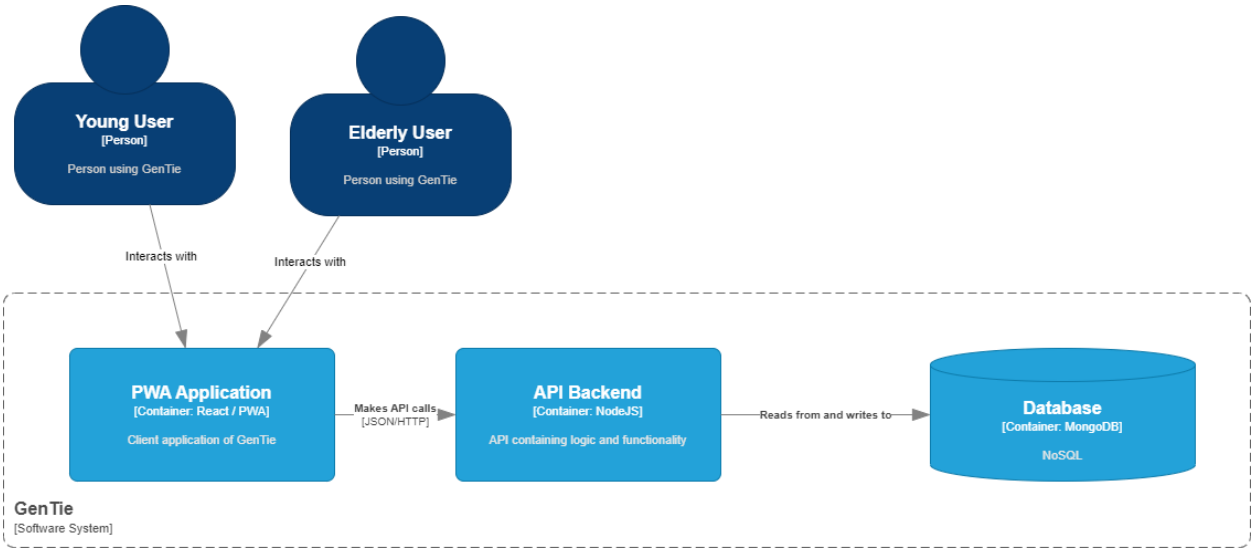
This document provides a comprehensive architectural overview of the system, using a number of different views to depict different aspects of the system. It is intended to capture and convey the significant architectural decisions which have been made to implement the software solution. It also to help developers more easily grasp the software's modules and components, without digging into the code, while also being incredibly useful as it gives an oversight of the entire system and exposes any potential pitfalls the developers might experience.

System context - C1



From the system context diagram, it can be seen that the software has two different users as the target audience – elderly and young people. This is what the project is defined by, so the software must be custom tailored to both of the end user groups.

Containers and technology - C2



From the diagram it can be seen that the software system is composed from three main containers:

- Single-page ReactJS PWA application
- REST API using NodeJS (proof of concept)
- NoSQL MongoDB database (proof of concept)

Separation of concerns

With the three main containers divided in this way it is ensured that the different divisions of the software are properly separated. The single-page application takes care of all of the UI and does not deal with any business logic. The REST API is where all of the business logic resides and is essentially the brain of operations. It provides all of the functionality of the software solution and passes data from the database to the Single-page application. The final block of the system is the database where all of the information persists.

Proof of concept?

The state of the software so far is a proof of concept. This means that so far only the PWA application is implemented. The back end and database are mocked – everything is persisted locally. This is not ideal for a production environment, but that makes it a proof of concept. The layer that saves everything locally can very easily be replaced with the intended back-end to finalize the project.

Why ReactJS

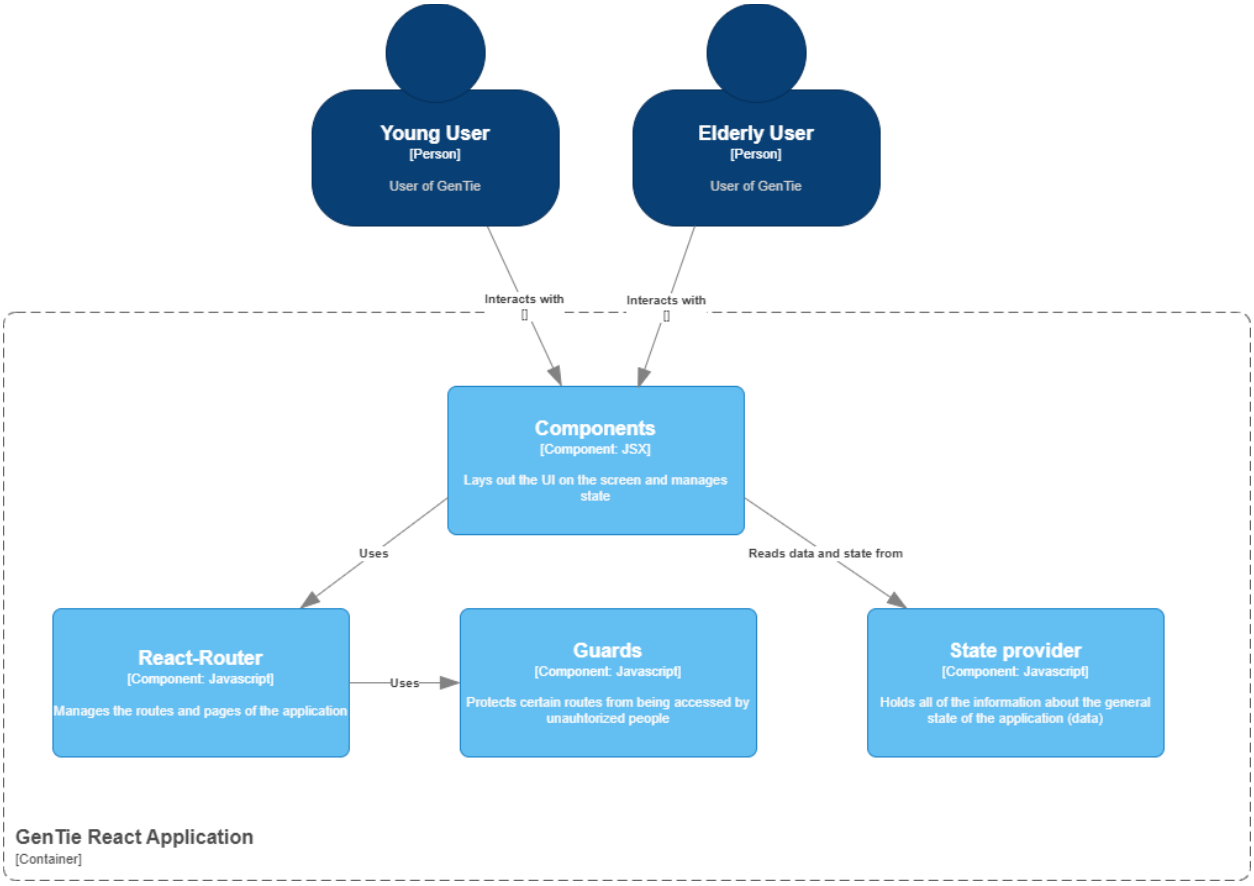
ReactJS is a very popular Javascript framework. It is incredibly powerful and well-documented, and if used properly, you can masterfully build the front-end of your application with both style and functionality. It has exceeding performance as it is

lightweight and works on the single-page application principle, where all of the front-end's scripts are loaded at once and the page never actually refreshes. Regular maintenance and updates make sure that it's always up to date and its popularity means that there are plenty of tutorials and support in case you run into any kind of issues. It is also supported by a lot of third-party libraries, so in the event of needing something incredibly niche and specific, chances are that someone already made it specifically for React. It plays very nicely with the PWA aspect of the application, since in the end it is boiled down to a simple "static" site, which can take use of service workers and caching.

Why Mui

The React front-end is heavily reliant on the mui framework. The framework follows the principles of Google's Material UI, which makes the application look very neat and good in a mobile environment. This also makes it easy to follow most of Google's well established design principles.

Components - C3



From the component diagram it can be seen in more detail how the React PWA app is structured. React components are used all throughout the application and are mostly UI elements – whether that would be whole pages or small UI elements. It is all linked by React's router that manages navigating through pages. Some pages are protected by guards, which ensure that you can only access them if you are authenticated (so far accounts are mocked, only 2 exist in memory). The whole application is wrapper in the state provider, which is the layer that mocks the back-end and database. This layer stores everything locally and can very easily be replaced by API calls to an actual back-end while still retaining the functionality of the application.