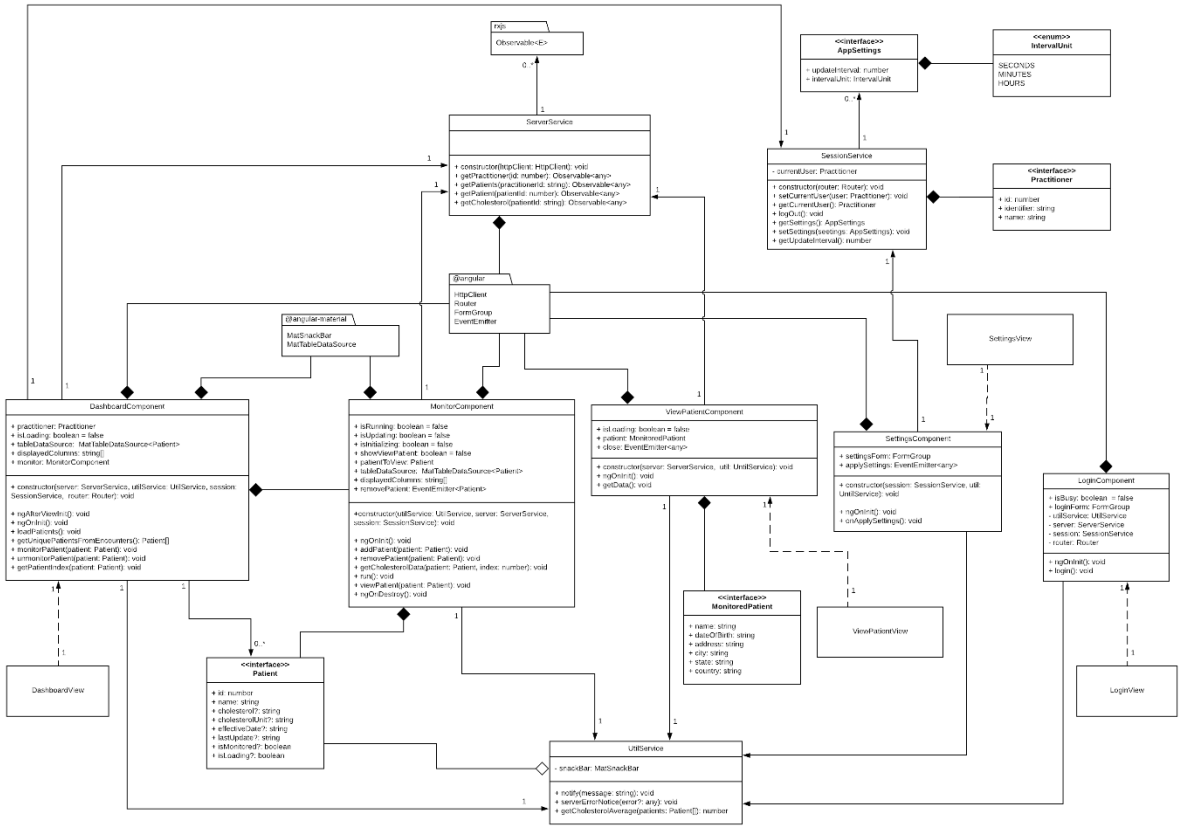
MedicineMan



Medicine-Man Class Diagram

Design Rationale

We decided to build a web application because we want the app to be platform independent and require minimal setup by the user. A browser is all that is needed to use the application. We used Angular CLI web framework because most the tools required for the project are shipped with angular, like the MVC architectural setup.

Angular provides the bases for single-page application, which makes the application simple to deploy and easy to navigate between components with the help of angular routing modules. Therefore, the application can seamlessly be extended my creating more components for implementing additionally functionalities to the medicine-man application in the future. In addition, to writing easily reusable code, testing is also simplified because each component can be tested independently.

Medicine-man is highly event-driven for the purpose of achieving asynchronous application architecture, this helps to achieve loose coupled components.

MVC (Model-View-Controller) pattern: The models are the injectable classes (services) that perform specific tasks. The ServerService class is responsible for retrieving data from the API while other service classes are used to provide specific functionalities to controllers throughout the app.

Observer design pattern: The controllers (components) and subscribes to the ServerService class which contain methods that return observables. The ServerService are responsible for asynchronously fetching data from the API server and the subscribed controllers are provided with the returned data from the server upon availability for display on the views.

The Common Closure Principle: Functionalities that are shared between components like notifications, exceptions, etc, are grouped in a UtilService class that is Injected into components that require it. That way, changes to the implementation of UtilService wil not create a ripple effect in other components. Secondly it ensures that dependency on such functionalities by the components flows in only one direction (from the component to UtilService) thereby eliminating Acyclic Dependencies Principle.

References

Angular SPA: Why Single Page Applications? (2017, April 21). Angular University. https://blog.angular-university.io/why-a-single-page-application-what-are-the-benefits-what-is-a-spa/

Improve Your Designs With The Principles Of Closure And Figure-Ground (Part 2). (, 43:49 + UTC). Smashing Magazine. https://www.smashingmagazine.com/2016/05/improve-your-designs-with-the-principles-of-closure-and-figure-ground-part-2/

The Acyclic Dependencies Principle – EricBackhage.NET. (n.d.). Retrieved May 29, 2020, from https://ericbackhage.net/clean-code/the-acyclic-dependencies-principle/