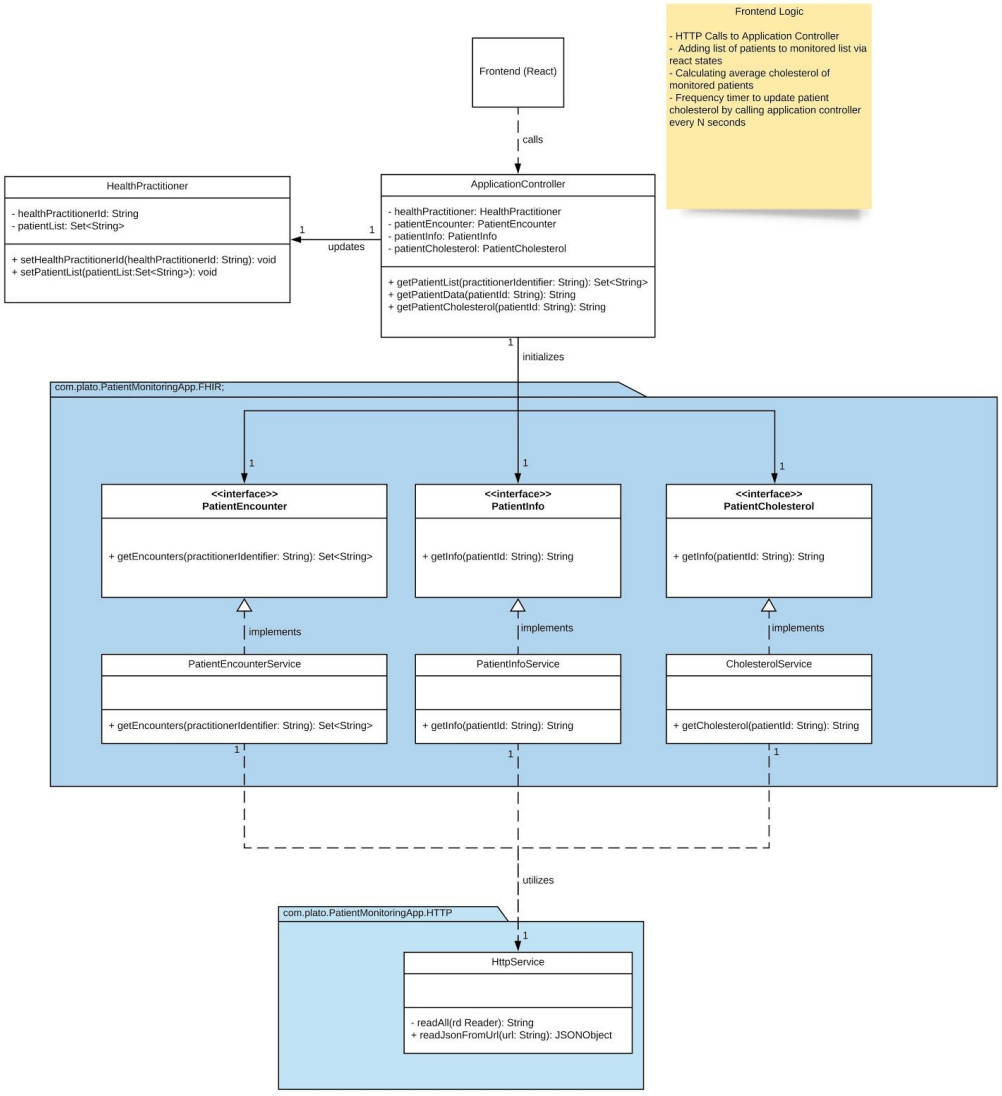

Introduction

The application that we were tasked to design and implement was a patient monitoring application where health practitioners can easily monitor the cholesterol levels of their patients. Through analyzing the requirements needed for the application and some discussions, we have decided to adopt a microservice approach for our system. The application would be designed as a web app with React as our front-end and Java Spring Boot as our backend. In this design rationale, we will be discussing about the current design of our system as well as the decision we have made along the way to achieve it.

Extensibility of the system

Since the application is new, we are only given the basic requirements needed for the application and we need to take into account all the functionalities that the client might want in the future. With that in mind, we needed to design the system so that it could be easily extendable to add in new features. One of the design principles that immediately came to mind was the **Open/Closed Principle** [1]**.** Firstly, by utilizing this principle, we can ensure that the classes that have been implemented would be closed off from modification thus making it more stable and reliable for other classes. Secondly, we can also ensure that all future extensions to our system can be implemented with ease.
To achieve the **Open/Closed Principle,** we can utilize another famous principle which is the **Dependency Inversion Principle** [2]. With this principle in mind, we designed our system to utilize

interfaces. Our high level module (*ApplicationController*) will now depend on interfaces rather than concrete implementations of lower level modules. The abstraction in our design provides “hinge points” so the system can be easily modified or extended which conforms to the **Open/Closed Principle**.

In addition, this design also follows the **Liskov Substitutability Principle** [3] where our concrete implementations of our interfaces are true subtypes of the interfaces and they honour the contracts/constraints laid down

---

Pattern Decisions

We have considered a few design patterns that might be useful to our design and did research and spikes on them, however in the end we decided not to use them. In this section, we will go through the reasons why we made the decision.

Firstly, the pattern that we considered to implement was the **Observer** [4] pattern. Our application has a clear data source as well as a representation of the data in a table format. This pattern would be useful if in the future the client decided to have different representation of the data in a format other than a table, for example a bar chart.

The main reason why we did not adopt this pattern into our design is because it conflicts with our initial decision to use a web app with React as our frontend. The observer pattern is more of a push approach meaning when there is new data to be updated, the observable informs all of its observers and “pushes” the information to them. React however is more of a pull approach and backend calls are only made whenever necessary, which we believe is more computationally efficient.

Ther other main pattern that we considered was a structural one, **bridge** [5]. Since our design had lots of interfaces/abstraction and implementation, this pattern looked promising. The reason why we did not adopt it is because unlike shapes and colors given in the lecture examples, the interfaces that we have don’t really have a solid connection with each other. However this pattern might be useful in the future when additional functionality is added so we will keep that in mind.

Other Principles

There were a few other notable principles that we used when designing the system.

- Firstly, for the **Acyclic Dependencies Principle** [6]**,** we made sure that none of the packages/components in our design form any cycles. The advantage of this is to reduce dependencies among code that might cause unexpected errors/bugs.
- Secondly, for the **Stable Dependencies Principle** [7] we made sure our classes depend on the direction of stability. Our HTTP package represents all of the http services which are being depended on by a lot of the other classes. The package does not depend on anything else and would rarely be changed thus making it stable and reliable.
- Finally, for the **Single Responsibility Principle** [8] the reason why we separated all of our FHIR services into different interfaces was to make sure that each of them would only fulfill one specific purpose.

React Logic

We decided to put some of the logic in the frontend instead of backend. One of the features required was to update the cholesterol info every N seconds. Since this is a web app, it would make more sense to have this logic in the front end. When the application is actually open, only then will we make calls to the backend which will fetch data from the server. This feels like a more resource friendly approach for the FHIR server.

References

[1]
FIT3077 Week 4 Lecture Principles of Object-Oriented Design 1
https://lms.monash.edu/pluginfile.php/10536249/mod_resource/content/2/OOP1.pdf
[2]
FIT3077 Week 4 Lecture Principles of Object-Oriented Design 1
https://lms.monash.edu/pluginfile.php/10536249/mod_resource/content/2/OOP1.pdf
[3]
FIT3077 Week 4 Lecture Principles of Object-Oriented Design 1
https://lms.monash.edu/pluginfile.php/10536249/mod_resource/content/2/OOP1.pdf
[4]
FIT3077 Week 5 Lecture Design Patterns 1
https://lms.monash.edu/pluginfile.php/10580885/mod_resource/content/1/Design_Patterns_1.pdf
[5]
FIT3077 Week 6 Lecture Design Patterns 2
https://lms.monash.edu/pluginfile.php/10625707/mod_resource/content/1/Design_Patterns_2.pdf
[6]
FIT3077 Week 7 Lecture Principles of Object-Oriented Design 2
https://lms.monash.edu/pluginfile.php/10668187/mod_resource/content/1/OOP2.pdf
[7]
FIT3077 Week 7 Lecture Principles of Object-Oriented Design 2
https://lms.monash.edu/pluginfile.php/10668187/mod_resource/content/1/OOP2.pdf
[8]
Dhurim Kelmendi - SOLID Principles made easy
https://medium.com/@dhkelmendi/solid-principles-made-easy-67b1246bcdf