MVC architecture
We use MVC as the main architecture for our web-based application in which the UI component is refactored into Controller and View. We also add the ViewModel layer into the application in order to lose coupling between presentation layer and lower layers. ViewModel helps us interact with the Model and display the data according to the need of what is to be displayed by the application without interfering/modifying the Models of the application and adding logic to the view

Repository pattern
In the design of our software system, we use the Repository design system to connect to the FHIR server as we dont want the logic to access the database logic to be directly in the business logic. Direct access of the database can make the business logic harder to unit test and later when extending the application we might run into a problem of duplicating data access code throughout the business logic layer; and so having a repository design pattern can eliminate these problems.

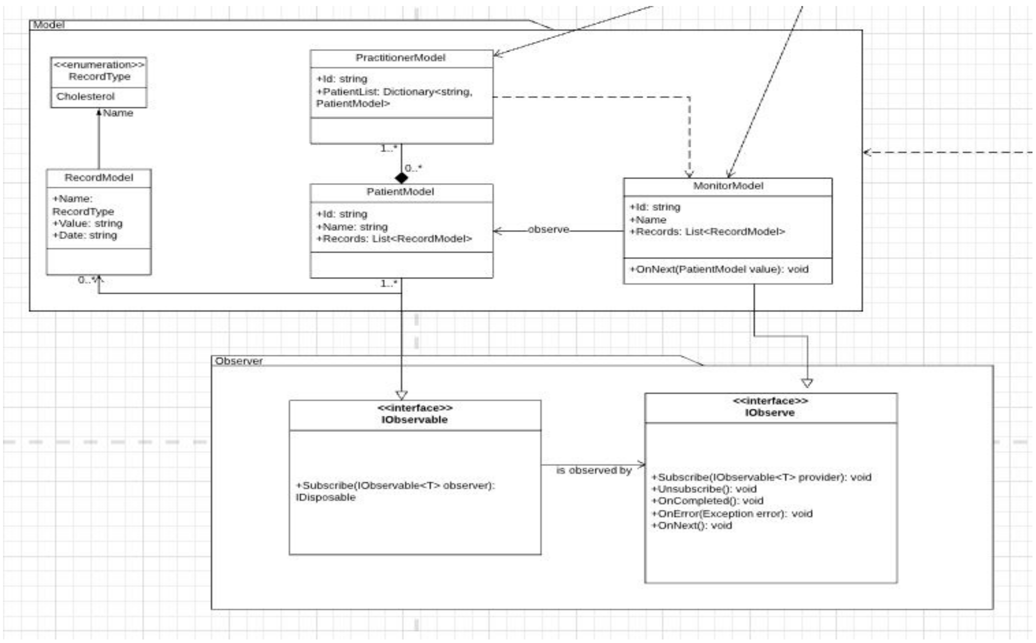
Enum for different measurements
To make our application extensible, we decided to use enum as a signature for our measurement. Enum violates the Open-Closed principle as it cannot be extended without modification. However, upon the requirements for our software piece, enum is a right choice we can create new measurements effortlessly by adding a new value to the enum set

Interface Segregation Principle
We define a separate interface for each of the repositories in the system instead of a large generic interface because we want to minimize dependencies of concrete class and to avoid clients being exposed to code that they don't need.

Limitation cause by web app
Because a web application is stateless so every time a request is made, the application does not save the client's data generated in one session for use in the next session with that client. Each session is carried out as if it was the first time and responses are not dependent upon data from the previous session. This causes us several problems in working with the application in terms of preserving data, retrieving data and more importantly updating the data. This would not even exist if we choose to work with a stateful application that saves data about each client session and uses that data the next time the client makes a request. And so we get around this by creating a temporary cache to save the necessary data retrieved when the client performs actions on the app; however this approach does not solve all of our problems such as when we want to introduce the observer pattern in our application, we cannot update the data as we want to when there is a change. We recognize that in order to fully optimize a web application we also need to implement a database for our application.

Potential problems we might have with current design and how to solve

We decided to remove the observer pattern we designed like below because of the limitation of technology we met. We came to a solution by making an update on the list of patients and then on the monitor list in order. On a large scale system, this could be slow and not efficient.
Moreover, we are currently making an interval update by using ajax to request the controller. However, this is inefficient as we update data that has no change in the server. Also, this is the infinite loop and makes our website waste more resources than it needs. We found a solution of using SignalR to stream the data intervally and pull them out on change but we are not able to implement it successfully


Model
<<enumeration>>
RecordType
Cholesterol
Name
RecordModel
+Name: RecordType
+Value: string
+Date: string
0..*
PractitionerModel
+Id: string
+PatientList: Dictionary<string, PatientModel>
1..*
0..*
PatientModel
+Id: string
+Name: string
+Records: List<RecordModel>
1..*
observe
MonitorModel
+Id: string
+Name
+Records: List<RecordModel>
+OnNext(PatientModel value): void
Observer
<<interface>>
IObservable
+Subscribe(IObservable<T> observer): IDisposable
is observed by
<<interface>>
IObserve
+Subscribe(IObservable<T> provider): void
+Unsubscribe(): void
+OnCompleted(): void
+OnError(Exception error): void
+OnNext(): void