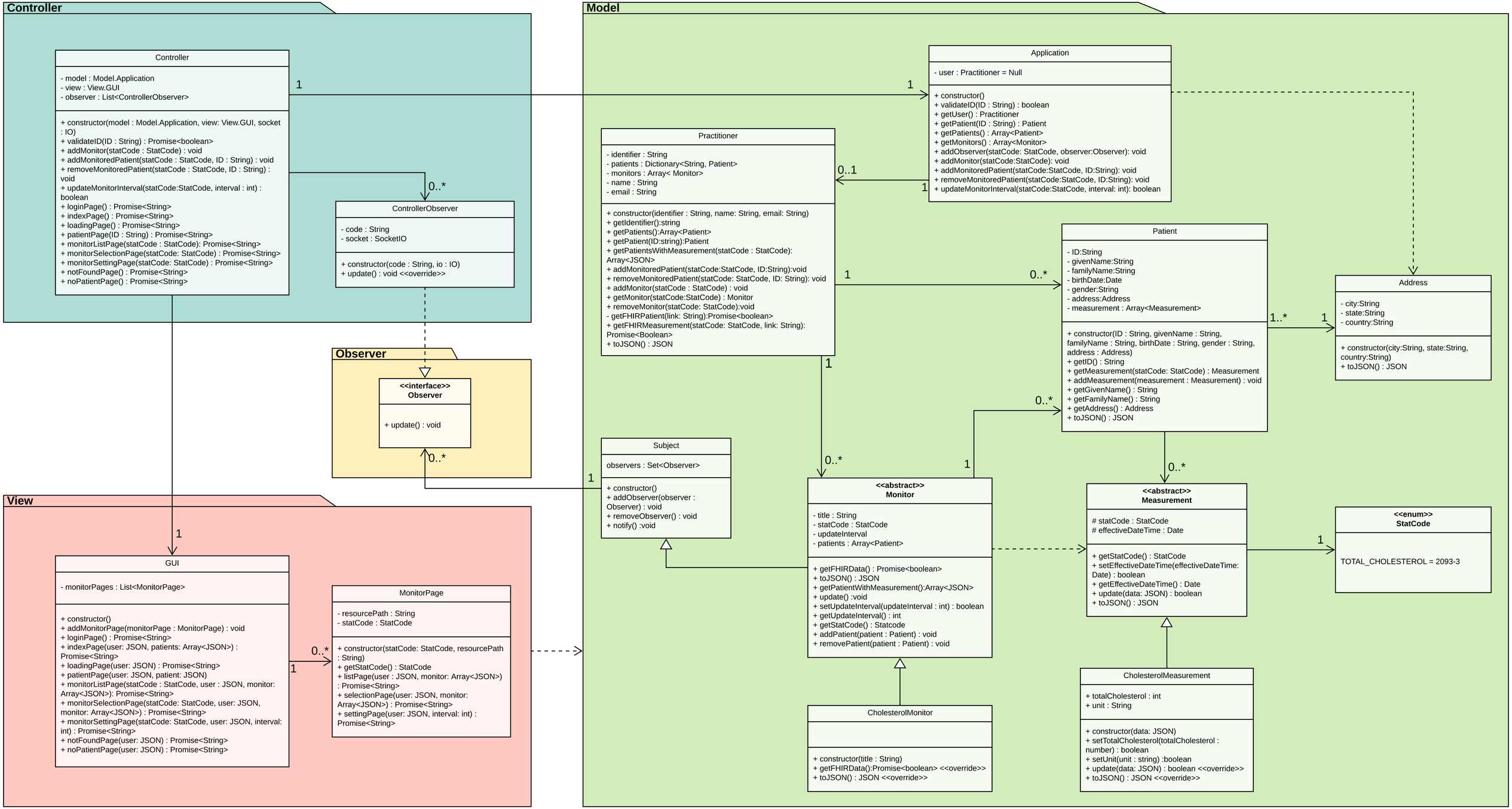
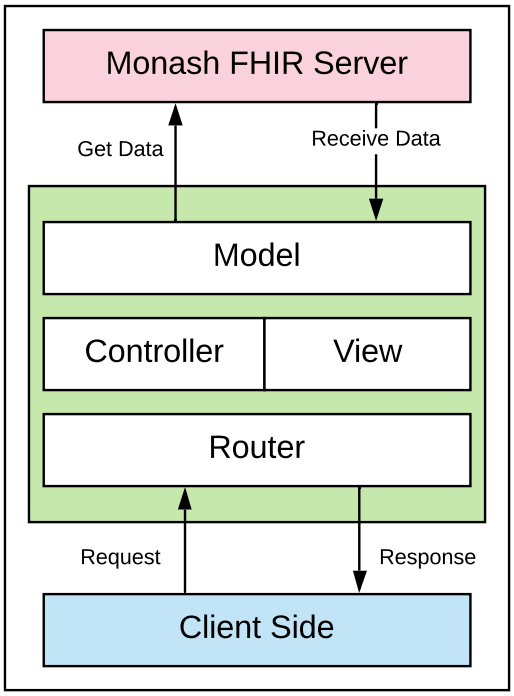

Controller
Controller
- model : Model.Application
- view : View.GUI
- observer : List<ControllerObserver>
+ constructor(model : Model.Application, view: View.GUI, socket : IO)
+ validateID(ID : String) : Promise<boolean>
+ addMonitor(statCode : StatCode) : void
+ addMonitoredPatient(statCode : StatCode, ID : String) : void
+ removeMonitoredPatient(statCode : StatCode, ID : String) : void
+ updateMonitorInterval(statCode:StatCode, interval : int) : boolean
+ loginPage() : Promise<String>
+ indexPage() : Promise<String>
+ loadingPage() : Promise<String>
+ patientPage(ID : String) : Promise<String>
+ monitorListPage(statCode : StatCode): Promise<String>
+ monitorSelectionPage(statCode: StatCode) : Promise<String>
+ monitorSettingPage(statCode: StatCode) : Promise<String>
+ notFoundPage() : Promise<String>
+ noPatientPage() : Promise<String>
1
0..*
ControllerObserver
- code : String
- socket : SocketIO
+ constructor(code : String, io : IO)
+ update() : void <<override>>
Observer
<<interface>>
Observer
+ update() : void
0..*
View
GUI
- monitorPages : List<MonitorPage>
+ constructor()
+ addMonitorPage(monitorPage : MonitorPage) : void
+ loginPage() : Promise<String>
+ indexPage(user: JSON, patients: Array<JSON>) : Promise<String>
+ loadingPage(user: JSON) : Promise<String>
+ patientPage(user: JSON, patient: JSON)
+ monitorListPage(statCode : StatCode, user : JSON, monitor: Array<JSON>): Promise<String>
+ monitorSelectionPage(statCode: StatCode, user: JSON, monitor: Array<JSON>) : Promise<String>
+ monitorSettingPage(statCode: StatCode, user: JSON, interval: int) : Promise<String>
+ notFoundPage(user: JSON) : Promise<String>
+ noPatientPage(user: JSON) : Promise<String>
1
0..*
MonitorPage
- resourcePath : String
- statCode : StatCode
+ constructor(statCode: StatCode, resourcePath : String)
+ getStatCode() : StatCode
+ listPage(user : JSON, monitor: Array<JSON>) : Promise<String>
+ selectionPage(user: JSON, monitor: Array<JSON>) : Promise<String>
+ settingPage(user: JSON, interval: int) : Promise<String>
Model
Application
- user : Practitioner = Null
+ constructor()
+ validateID(ID : String) : boolean
+ getUser() : Practitioner
+ getPatient(ID : String) : Patient
+ getPatients() : Array<Patient>
+ getMonitors() : Array<Monitor>
+ addObserver(statCode: StatCode, observer:Observer): void
+ addMonitor(statCode:StatCode): void
+ addMonitoredPatient(statCode:StatCode, ID:String): void
+ removeMonitoredPatient(statCode:StatCode, ID:String): void
+ updateMonitorInterval(statCode:StatCode, interval: int): boolean
1
1
Practitioner
- identifier : String
- patients : Dictionary<String, Patient>
- monitors : Array< Monitor>
- name : String
- email : String
+ constructor(identifier : String, name: String, email: String)
+ getIdentifier():string
+ getPatients():Array<Patient>
+ getPatient(ID:string):Patient
+ getPatientsWithMeasurement(statCode : StatCode): Array<JSON>
+ addMonitoredPatient(statCode:StatCode, ID:String):void
+ removeMonitoredPatient(statCode: StatCode, ID: String): void
+ addMonitor(statCode : StatCode) : void
+ getMonitor(statCode:StatCode) : Monitor
+ removeMonitor(statCode: StatCode):void
- getFHIRPatient(link: String):Promise<boolean>
+ getFHIRMeasurement(statCode: StatCode, link: String): Promise<Boolean>
+ toJSON() : JSON
0..1
1
1
0..*
Patient
- ID:String
- givenName:String
- familyName:String
- birthDate:Date
- gender:String
- address:Address
- measurement : Array<Measurement>
+ constructor(ID : String, givenName : String, familyName : String, birthDate : String, gender : String, address : Address)
+ getID() : String
+ getMeasurement(statCode: StatCode) : Measurement
+ addMeasurement(measurement : Measurement) : void
+ getGivenName() : String
+ getFamilyName() : String
+ getAddress() : Address
+ toJSON() : JSON
1..*
1
Address
- city:String
- state:String
- country:String
+ constructor(city:String, state:String, country:String)
+ toJSON() : JSON
1
0..*
Subject
observers : Set<Observer>
+ constructor()
+ addObserver(observer : Observer) : void
+ removeObserver() : void
+ notify() :void
1
0..*
1
0..*
<<abstract>>
Monitor
- title : String
- statCode : StatCode
- updateInterval
- patients : Array<Patient>
+ getFHIRData() : Promise<boolean>
+ toJSON() : JSON
+ getPatientWithMeasurement():Array<JSON>
+ update() :void
+ setUpdateInterval(updateInterval : int) : boolean
+ getUpdateInterval() : int
+ getStatCode() : Statcode
+ addPatient(patient : Patient) : void
+ removePatient(patient : Patient) : void
0..*
<<abstract>>
Measurement
statCode : StatCode
effectiveDateTime : Date
+ getStatCode() : StatCode
+ setEffectiveDateTime(effectiveDateTime: Date) : boolean
+ getEffectiveDateTime() : Date
+ update(data: JSON) : boolean
+ toJSON() : JSON
1
<<enum>>
StatCode
TOTAL_CHOLESTEROL = 2093-3
CholesterolMonitor
+ constructor(title : String)
+ getFHIRData():Promise<boolean> <<override>>
+ toJSON() : JSON <<override>>
CholesterolMeasurement
+ totalCholesterol : int
+ unit : String
+ constructor(data: JSON)
+ setTotalCholesterol(totalCholesterol : number) : boolean
+ setUnit(unit : string) :boolean
+ update(data: JSON) : boolean <<override>>
+ toJSON() : JSON <<override>>

Design Rationale

We are developing our application as a Web-based Application. In web-based Application, there are exist a clear seperation between client-side and server-side.



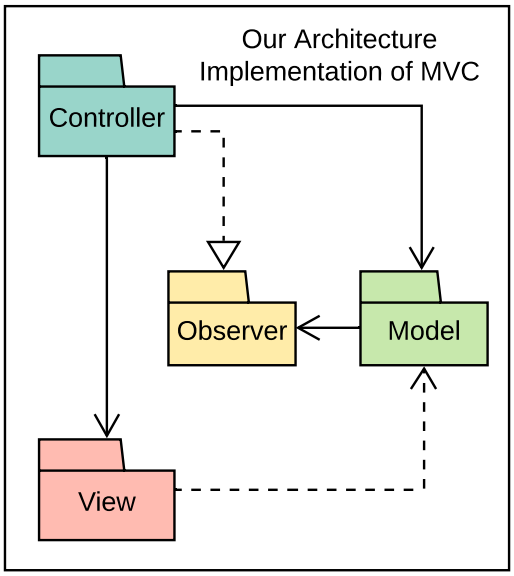
Server Side:

- Handle Incoming Request and User Input
- Enforce the Business Logics and rule
- Communicate with FHIR Server
- Store the needed Data

Client Side:

- Show the data in Graphical User Interface
- Get the User Input
- Send the User Input to the Server

In our architecture, we are following the MVC (Model-View-Controller) architectural pattern. One of the main reason why we are following MVC, is because MVC pattern provide a clear seperation between between GUI and the backend, here is our division of responsibility by following the MVC pattern:



Model:

- Enforce the Business Logics and rule
- Communicate with FHIR Server
- Store the needed Data

Controller:

- Handle Incoming Request and User Input

View:

- Show the data in Graphical User Interface
- Get the User Input
- Send the User Input to the Server

One of the requirement is that the user (practitioner) can monitor selected patients for their total cholesterol value, and the application need to keep check if the data is up to date. This feature is impossible to be achieve by MVC passive model, because Model can't inform the Controller at all. Therefore we used MVC active model, by using **Observer pattern** so that Model can inform Controller without violating the **Acyclic Dependencies Principle (ADP)**.

On the model itself, we enforce the use of Abstraction by following **Open-Closed Principle (OCP)**, to reduce the rigidity for any extension. this can be seen on the Monitor and Measurement class where each of them have different behaviour depend on the statcode. If there are any new statcode, we can easily make a new measurement subclasses that conforms the Measurement Base Class **(Liskov Substitutability Principle (LSP))**. The biggest benefit of following LSP, is that we can easily extend the abstract class without changing other classes.