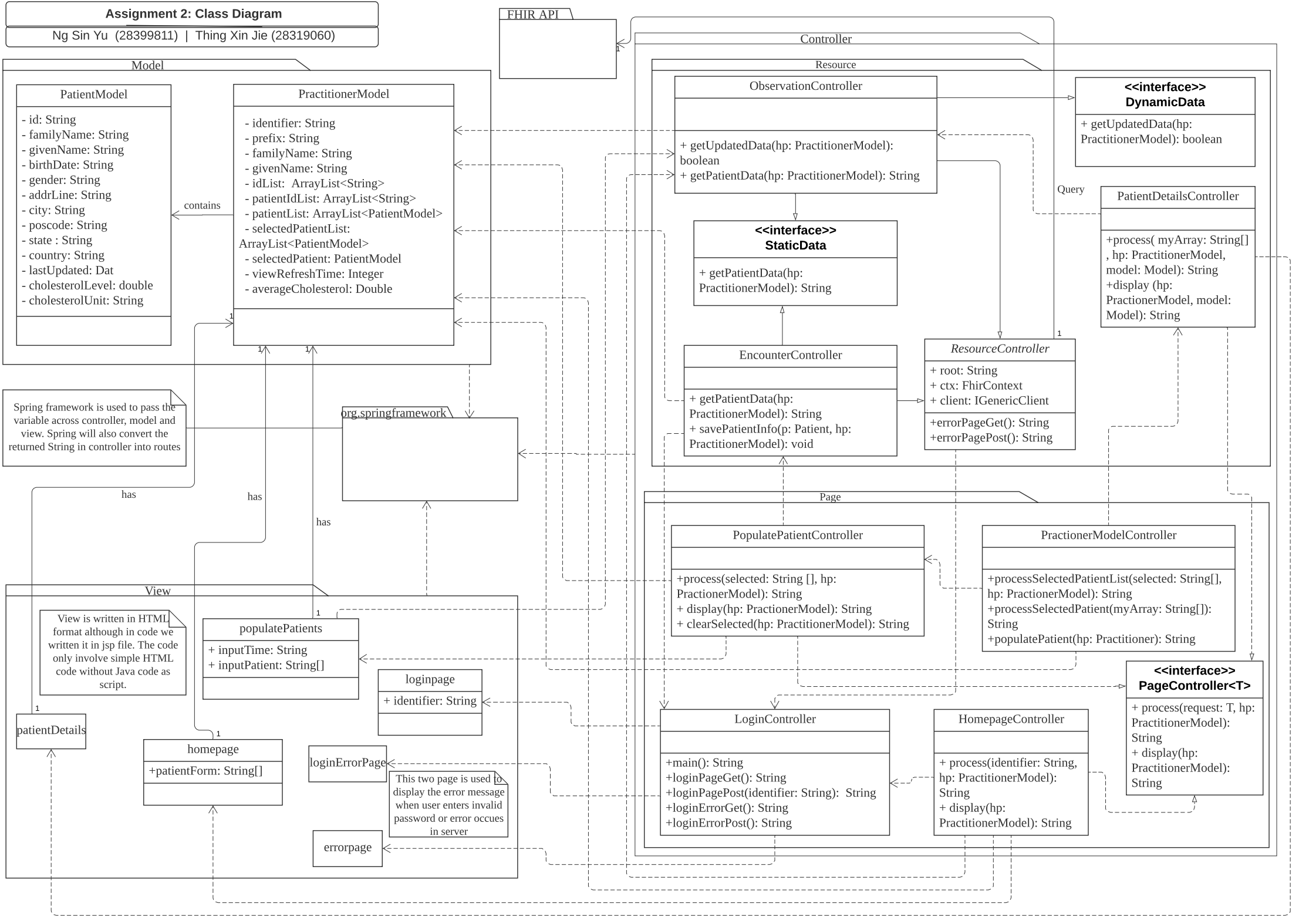
Assignment 2: Class Diagram
Ng Sin Yu (28399811) | Thing Xin Jie (28319060)
FHIR API
Controller
Model
Resource
PatientModel
- id: String
- familyName: String
- givenName: String
- birthDate: String
- gender: String
- addrLine: String
- city: String
- poscode: String
- state : String
- country: String
- lastUpdated: Dat
- cholesterolLevel: double
- cholesterolUnit: String
contains
PractitionerModel
- identifier: String
- prefix: String
- familyName: String
- givenName: String
- idList: ArrayList<String>
- patientIdList: ArrayList<String>
- patientList: ArrayList<PatientModel>
- selectedPatientList: ArrayList<PatientModel>
- selectedPatient: PatientModel
- viewRefreshTime: Integer
- averageCholesterol: Double
ObservationController
+ getUpdatedData(hp: PractitionerModel): boolean
+ getPatientData(hp: PractitionerModel): String
<<interface>> DynamicData
+ getUpdatedData(hp: PractitionerModel): boolean
Query
PatientDetailsController
+process(myArray: String[] , hp: PractitionerModel, model: Model): String
+display (hp: PractionerModel, model: Model): String
<<interface>> StaticData
+ getPatientData(hp: PractitionerModel): String
EncounterController
+ getPatientData(hp: PractitionerModel): String
+ savePatientInfo(p: Patient, hp: PractitionerModel): void
ResourceController
+ root: String
+ ctx: FhirContext
+ client: IGenericClient
+errorPageGet(): String
+errorPagePost(): String
Spring framework is used to pass the variable across controller, model and view. Spring will also convert the returned String in controller into routes
org.springframework
has
has
has
Page
PopulatePatientController
+process(selected: String [], hp: PractionerModel): String
+ display(hp: PractionerModel): String
+ clearSelected(hp: PractitionerModel): String
PractionerModelController
+processSelectedPatientList(selected: String[], hp: PractionerModel): String
+processSelectedPatient(myArray: String[]): String
+populatePatient(hp: Practitioner): String
<<interface>> PageController<T>
+ process(request: T, hp: PractitionerModel): String
+ display(hp: PractitionerModel): String
View
View is written in HTML format although in code we written it in jsp file. The code only involve simple HTML code without Java code as script.
populatePatients
+ inputTime: String
+ inputPatient: String[]
loginpage
+ identifier: String
patientDetails
homepage
+patientForm: String[]
loginErrorPage
This two page is used to display the error message when user enters invalid password or error occues in server
errorpage
LoginController
+main(): String
+loginPageGet(): String
+loginPagePost(identifier: String): String
+loginErrorGet(): String
+loginErrorPost(): String
HomepageController
+ process(identifier: String, hp: PractitionerModel): String
+ display(hp: PractitionerModel): String

Model-View-Controller (MVC) Architectural Pattern

- MVC architecture segregates the model, view and controller well, making it easy to maintain and reuse each resource and business logic. MVC is also suitable for developing a web application, which meets our initial intention as well since we are planning to develop a web application as our deliverable.
- Controllers will perform services and pass the processed data (fetched data from server in our case) into model, and view will only need to get information to be displayed from model while model does not need to know anything about controller and view, which makes acyclic dependencies possible to achieve.
- The development of view (the UI) is independent of the backend processes, hence frequent changes on UI especially on UI design will not affect or affected by business logic processes. Moreover, as a team project, when one of our teammates is developing a frontend interface, another teammate can develop backend services without interfering with each other.
- Model is used to store the information only. Model does not need to know anything about the controller and view. Because of this separation of the responsibility of model, view and controller, it helps to reduce the effort for the future modification. So if there is any new requirement being added to the project, the project can accommodate with it

Interface Segregation Principle (ISP)

- Controller classes (ObservationController and EncounterController) are fetching data from FHIR server differently, where ObservationController needs to perform data fetching to obtain patients' cholesterol level at interval while EncounterController will only need to fetch the data once to obtain details such as name, gender, address etc
- By applying ISP, EncounterController implements a function in StaticData interface and perform data fetching once
- Meanwhile, ObservationController needs to fetch a patient's initial cholesterol level once to initialise patients' cholesterol level in the homepage (where practitioners can select patients to monitor) and fetching data once is exactly what a StaticData interface is trying to do. ObservationController also needs to implement DynamicData which will perform data fetching at intervals.
- ISP allows changes in interface to only affecting the classes that are implementing it and avoids implicit dependency between classes. For example, if we need extra methods in DynamicData, only ObservationController that implements it will need to add extra methods. This also makes sense where Encounter information on server might not be changed but Observation data on server might be updated, hence having different interface makes our system more flexible especially for the classes to perform data fetching

Common Closure Principle (CCP) and Single Responsibility Principle

- Page package store controllers that handle the view rendering while resource package store controllers that are used for database query.
- One of the reasons that we separate the controller to two different packages is because if there are any changes must be made within one of the controller packages, the other will not be directly affected.
- These then will increase the maintainability of our code especially if there is any other type of controller that is needed for new functionality, it can be easily added to our design.
- Common Closure Principle is actually closely related with Single Responsibility Principle, so the separation of the package based on its functionality gathers the same component together. So, for the resource package it will only change if there are any changes in the ways for the data fetching.