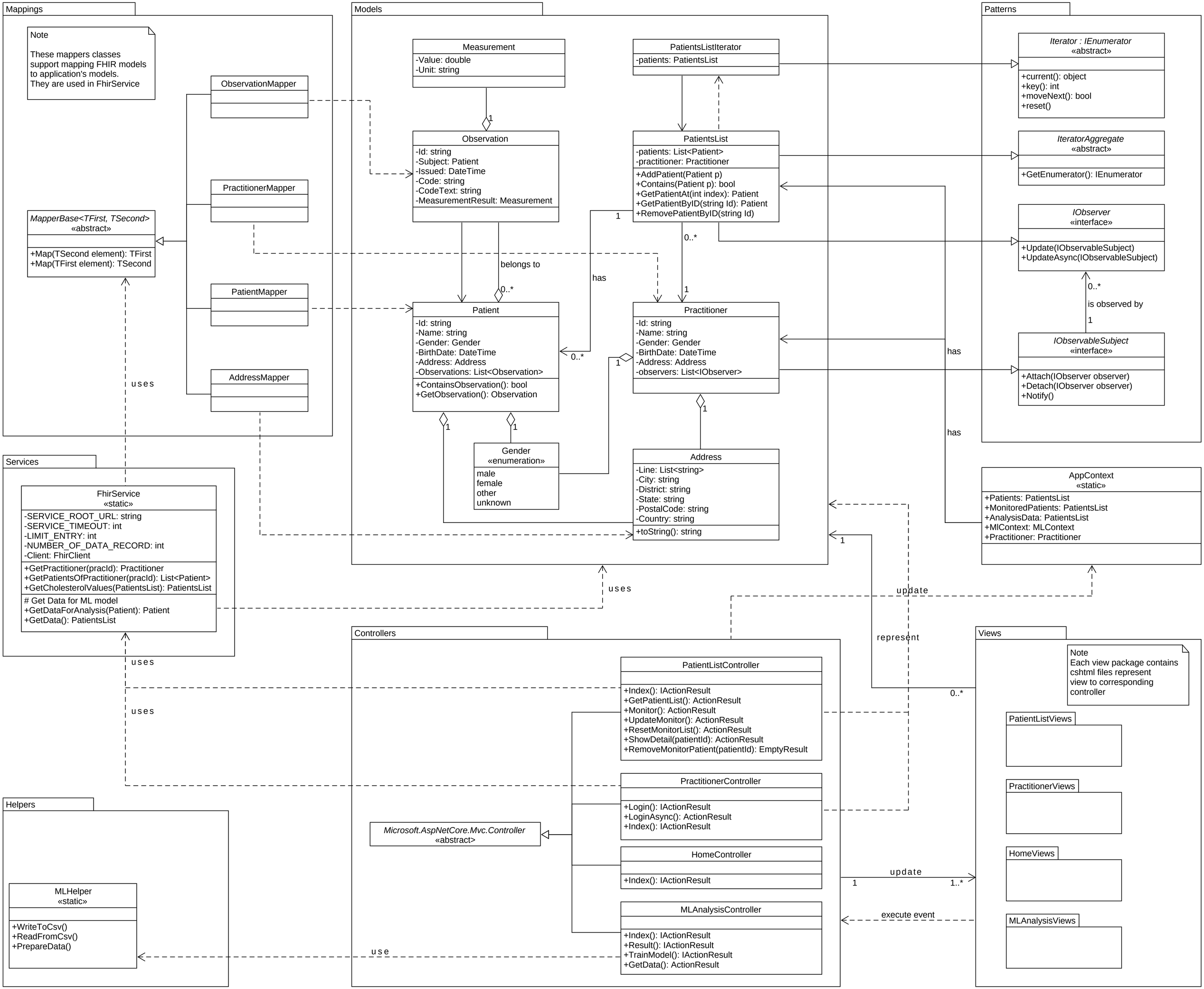
Mappings
Note
These mappers classes support mapping FHIR models to application's models. They are used in FhirService
ObservationMapper
PractitionerMapper
PatientMapper
AddressMapper
MapperBase<TFirst, TSecond>
«abstract»
+Map(TSecond element): TFirst
+Map(TFirst element): TSecond
uses
Models
Measurement
-Value: double
-Unit: string
Observation
-Id: string
-Subject: Patient
-Issued: DateTime
-Code: string
-CodeText: string
-MeasurementResult: Measurement
belongs to
0..*
1
Patient
-Id: string
-Name: string
-Gender: Gender
-BirthDate: DateTime
-Address: Address
-Observations: List<Observation>
+ContainsObservation(): bool
+GetObservation(): Observation
has
Gender
«enumeration»
male
female
other
unknown
PatientsListIterator
-patients: PatientsList
PatientsList
-patients: List<Patient>
-practitioner: Practitioner
+AddPatient(Patient p)
+Contains(Patient p): bool
+GetPatientAt(int index): Patient
+GetPatientByID(string Id): Patient
+RemovePatientByID(string Id)
Practitioner
-Id: string
-Name: string
-Gender: Gender
-BirthDate: DateTime
-Address: Address
-observers: List<IObserver>
Address
-Line: List<string>
-City: string
-District: string
-State: string
-PostalCode: string
-Country: string
+toString(): string
Patterns
Iterator : IEnumerator
«abstract»
+current(): object
+key(): int
+moveNext(): bool
+reset()
IteratorAggregate
«abstract»
+GetEnumerator(): IEnumerator
IObserver
«interface»
+Update(IObservableSubject)
+UpdateAsync(IObservableSubject)
is observed by
IObservableSubject
«interface»
+Attach(IObserver observer)
+Detach(IObserver observer)
+Notify()
AppContext
«static»
+Patients: PatientsList
+MonitoredPatients: PatientsList
+AnalysisData: PatientsList
+MlContext: MLContext
+Practitioner: Practitioner
Services
FhirService
«static»
-SERVICE_ROOT_URL: string
-SERVICE_TIMEOUT: int
-LIMIT_ENTRY: int
-NUMBER_OF_DATA_RECORD: int
-Client: FhirClient
+GetPractitioner(pracId): Practitioner
+GetPatientsOfPractitioner(pracId): List<Patient>
+GetCholesterolValues(PatientsList): PatientsList
Get Data for ML model
+GetDataForAnalysis(Patient): Patient
+GetData(): PatientsList
uses
update
represent
Controllers
PatientListController
+Index(): IActionResult
+GetPatientList(): ActionResult
+Monitor(): ActionResult
+UpdateMonitor(): ActionResult
+ResetMonitorList(): ActionResult
+ShowDetail(patientId): ActionResult
+RemoveMonitorPatient(patientId): EmptyResult
PractitionerController
+Login(): IActionResult
+LoginAsync(): ActionResult
+Index(): IActionResult
Microsoft.AspNetCore.Mvc.Controller
«abstract»
HomeController
+Index(): IActionResult
MLAnalysisController
+Index(): IActionResult
+Result(): IActionResult
+TrainModel(): IActionResult
+GetData(): ActionResult
update
execute event
Views
Note
Each view package contains cshtml files represent view to corresponding controller
PatientListViews
PractitionerViews
HomeViews
MLAnalysisViews
Helpers
MLHelper
«static»
+WriteToCsv()
+ReadFromCsv()
+PrepareData()
use

Design Rationale: ASP.NET MVC

Presented with the requirements of developing an application for health practitioners to monitor their patients, we decided to implement the Model-View-Controller pattern to develop an ASP.NET application using ASP.NET MVC framework. Using the MVC pattern enables a clean separation of concern (input, processing, and output). The application will need to have multiple views (login, practitioner details, patient lists…) so using MVC pattern enable providing multiple views while separating data and business logic from display. A lot of calls to the FHIR server needed to be made, so the ability to do asynchronous programming from the framework helps provide a faster and more responsive application.

Practitioner, Patient, PatientsList, Observation… are concrete classes designed to represent the application state and encapsulate implementation logic. We used the Abstract Factory pattern in our MapperBase interface to map the Model from FHIR server to our own Model objects, this helps with consistency and avoidance of coupling between the FHIR model and client code in the application. This follows the Single Responsibility Principle as it extracts the product creation code into one place for easier support and the Open/Closed Principle it can introduce new variants of products without breaking existing client code (adding more Mappers). We decided to apply the Iterator pattern to iterate through the patients list. This helps clean up the client code and put bulky traversal algorithms into separate classes (Single Responsibility) and be open to extension of new types of collections and iterators (Open/Closed Principle). Here in the app it is not very effective yet as the PatientsList is of type List<> but it is open to more update in the future.

For updating the Model data on user input (changes in View), we chose to apply the Observer pattern, this pattern helps comply the rules of the OCP(not having to update observed object when adding new observers, open to add more observers easily) and the LSP.

The MVC pattern chosen helped grouping reusable classes together in packages (models, views, controllers, mappings, patterns…) following the Release Reuse Equivalent Principle. There is also a moderation in the size of the packages to balance the Common Reuse Principle and Common Closure Principle. The Models package is designed to be the most depended on and the most stable, Models package is responsible to Controller and View packages, does not depend on anything (SDP). All the dependencies do not form cycles (ADP).

Disadvantages of design: the requirements data formed a Models package without any abstraction possible so the Stable Abstractions Principle could not be followed. The use of layers helps to control and encapsulate the complexity of large application but now in developing a relatively simple application so it may add complexity.

Network Traffic:

In the application, all the interactions with FHIR Server are in FHIRService.cs file. This helps reduce the dependency on FHIR Models in the system. Whenever user updates a monitor list, the application checks if the added patient already has the observations values. If the patient's observations are queried before, it will not re-query that patient.

ML Task:

ML Task is implemented in C# using ML.NET package, so it can be integrated as a part of the web application. Web application user can build model from the web. The model uses Fast Tree binary classification algorithm because the data is labelled into True/False value (True: High Cholesterol). Getting data from FHIR Server is dynamically implemented. In the web application run-time, user can

choose getting more data, and it will fetch a fixed number of records (default by 200) and add them to a csv file. Getting data from server is implemented as an asynchronous task, so when the process is running, it will not block the entire application. Therefore, user can continue using the app.

Model result with threshold = 190mg/dL, dataset 120 records:

- Accuracy: 0.74 (74%)
- F1Score: 0.8 (80%)

Citations:

Overview of ASP.NET Core MVC. (n.d.). Retrieved March, 2020, from https://docs.microsoft.com/en-us/aspnet/core/mvc/overview?view=aspnetcore-3.1

Principles of Object-Oriented Design 1. Retrieved March, 2020, from https://lms.monash.edu/pluginfile.php/10536249/mod_resource/content/2/OOP1.pdf

Principles of Object-Oriented Design 2. Retrieved March, 2020, from https://lms.monash.edu/pluginfile.php/10668187/mod_resource/content/1/OOP2.pdf

Design Patterns 1. Retrieved March, 2020, from https://lms.monash.edu/pluginfile.php/10580885/mod_resource/content/1/Design_Patterns_1.pdf

Design Patterns 2. Retrieved March 5, 2020, from https://lms.monash.edu/pluginfile.php/10625707/mod_resource/content/1/Design_Patterns_2.pdf