Mappings

Note
These Mapper classes support mapping FHIR models to application's models. Implement Abstract Factory pattern

ObservationMapper

Component ObservationMapper

PractitionerMapper

MapperBase<TFirst, TSecond>
«abstract»
+Map(TSecond element): TFirst
+Map(TFirst element): TSecond

PatientMapper

AddressMapper

Models

Measurement
-Value: double
-Unit: string

Observation
-Id: string
-Issued: DateTime
-Code: string
-CodeText: string
-MeasurementResult: Measurement

Tracker
-Observations: List<Observation>
-Patient: Patient
-Type: string
-GraphData: List<double>

Patient
-Id: string
-Name: string
-Gender: Gender
-BirthDate: DateTime
-Address: Address
-Observations: List<Observation>
+ContainsObservation(): bool
+GetObservation(): Observation
+GetAllObservations(): List<Observation>

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

Note
Iterator Pattern is used to traverse a collection of Patients without knowing how the collection is structured

Note
Observer Pattern is used to allow Practitioner to notify PatientList objects when it's changed by user

Patterns

Iterator : IEnumerator
«abstract»
+current(): object
+key(): int
+moveNext(): bool
+reset()

IteratorAggregate
«abstract»
+GetEnumerator(): IEnumerator

IObserver
«interface»
+Update(IObservableSubject)
+UpdateAsync(IObservableSubject)

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>
+GetObservationValues(PatientsList): PatientsList

Note
New classes are in red
Extended methods are in bold text

Controllers

PatientListController
+Index(): IActionResult
+GetPatientList(): ActionResult
+Monitor(): ActionResult
+HistoricalMonitor(): ActionResult
+UpdateMonitor(): ActionResult
+ResetView(): ActionResult
+ShowDetail(patientId): ActionResult
+RemoveMonitorPatient(patientId): ActionResult
+SetUpdateInterval(newInterval): ActionResult
+GetUpdateInterval(): JsonResult
+SetThreshold(systolic, diastolic): ActionResult
+GetThreshold(): JsonResult
+UpdateGraphData(ViewName): JsonResult

Microsoft.AspNetCore.Mvc.Controller
«abstract»

PractitionerController
+Login(): IActionResult
+LoginAsync(): ActionResult
+Index(): IActionResult

HomeController
+Index(): IActionResult

Views

Note
Each view package contains cshtml files represent view to corresponding controller

PatientListViews

PractitionerViews

HomeViews

Relationships / labels
contains, has, belongs to, uses, is observed by, update, represent, execute event, has

Design Rationale: ASP.NET MVC

Presented with the new requirements of extending our initial design, the MVC pattern is still used and extended with a new View to display historical monitor. And the current Monitor view is extended to add more component to meet the new requirements.

We used the Abstract Factory pattern in our MapperBase interface to map the Models from FHIR server to our own Model classes, this helps with consistency and avoidance of coupling between the FHIR model and client code in the application.

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.

The Observer pattern is applied to manage updates in display classes and in the model data classes. When model history is changed, the Observer pattern is chosen to make the model notifies all the views related. 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.

From our initial design, we only used the Abstract Factory pattern for the new blood pressure mapping from FHIR server to our own Observation object. The Open/Closed pattern is followed in our initial design, so we did not need to modify classes from our models but only extended our Abstract Factory pattern for 1 more mapper and the Service class to read data from FHIR server.

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.

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