CSCD01 Deliverable 1: Software Architecture

Team 11 - Phantom Developers

February 26, 2020

Contents

1 Summary of Packages **3**

1.1 Main Package 3

1.2 Sub Packages 3

2 Interesting aspects of the design **8**

2.1 Factory Design Pattern 8

2.2 Modular Architecture 8

3 UML Diagram **9**

4 Possible Improvements **10**

4.1 Enum Class 10

4.2 Appropriate Packages 10

4.3 Moving Classes 10

5 Software Development Process (Kanban) **11**

6 Why did we choose Kanban? **12**

6.1 Kanban vs Waterfall 12

6.2 Kanban vs Extreme Programming 12

6.3 Kanban vs Incremental 12

1 Summary of Packages

1.1 Main Package

org.openmrs

This is the main package for the OpenMRS API. It contains many sub packages for each different section of the application. There are also several class objects in this package.

1.2 Sub Packages

annotation

This package contains classes associated with the custom java 1.5+ annotations.

aop

This package contains the classes associated with Aspect Oriented Programming (AOP).

api

This package provides interfaces of services to be implemented. These services deal with actions on that object. For example, the PatientService provides numerous methods about the patient object, such as setting and getting patients.

api.cache

This package deals with the configuration of caching and its properties.

api.context

This package deals with user authentication as well as providing access to services.

api.db

This package provides interfaces of object DAOs, which are used to access the database of an object.

api.db.hibernate

This package contains the resources for Hibernate Object-Relational-Mapping. It maps Java objects models to relational models that can be used in the database.

api.db.hibernate.search

This subpackage of hibernate contains the different types of searches that can be used. Criteria queries, Lucene queries, etc.

api.db.hibernate.search.bridge

This subpackage of search contains classes that convert Locale objects to strings and OpenMrsObjects to indexes(IDs). It contains the helper functions when making search queries.

api.handler

This package contains the "handler" interfaces for Encounter, visit, required, retire, save, etc actions and different implementations of them. Handlers decide whether or not something is a specific action. For example, BaseEncounterHandler and ExistingOrNewVisitAssignmentHandler both implement EncounterVisitHandler, and they determine the different types of visits.

api.impl

This package contains the classes that implement the interfaces in the api package. Eg AdminstrationServiceImpl class implements the AdminstrationServiceInterface interface in api.

attribute

Contains attributes that add on to a class. For example a Visit can have Visit Attributes that are associated with Visits.

collection

Contains wrappers for the Java Standard Collection and List

comparator

Contains wrappers for the Java Standard Comparator to order patients by certain attributes

messagesource

This package is responsible for l10n (localization) and i18n (internalization), it will access the message.properties file and extract message according to the locale of the user.

obs

Contains the interfaces for handling obs objects. An obs object collects information on a patient, including encounters, comments, etc.

order

Contains methods for handling orders and suggesting drugs.

customdatatype

Contains custom datatypes for global properties, attribute types, etc

hl7

OpenMRS HL7 module.

hl7.db.hibernate

HL7-specific DAO classes

hl7.handler

Handles different types of HL7 messages

layout

Contains templates for Addresses and Names

logic

The OpenMRS Logic Service provides access to granular and derived data. This package is currently not being used

logic.datasource

Logic data sources are responsible for providing data to the logic service engine.

logic.op

Contains operators

logic.result

Contains the results classes from the logic service

logic.rule

Contains logic rules

messagesource

This package is majorly for localization and internalization of the program. It will access to a specific property file according to the user's locale

module

This package contains all the classes that connect and interface with the different modules (addons) to the OpenMRS core including the User Interface.

notification

This package contains the logic of send user email and alert notification

patient

This package contains the interface for validating the different patient fields.

patient.impl

This package contains the implementations of validating different patient fields of different forms of numbers and characters.

person

This package contains a few class objects for storing person logs.

propertyeditor

This package contains classes that extend the ‘OpenmrsPropertyEditor‘ class to edit the set and get the text of different properties of objects.

scheduler

This package contains the classes objects for the schedules that start tasks and the tasks that can be scheduled.

scheduler.db

This package contains the class for creating, getting, and deleting tasks and schedules from the database.

scheduler.tasks

This package contains the implementation for the different types of tasks.

scheduler.timer

This package contains the classes for scheduling the different tasks.

serialization

This package contains the classes for serializing different objects to string and deserializing string to objects.

util

This package contains the shared utilities for many OpenMRS classes in the application.

util.databasechange

This package contains different classes for accessing the database.

validator

This package contains the shared validators for input for many OpenMRS classes in the application.

2 Interesting aspects of the design

2.1 Factory Design Pattern

The openmrs-core API uses a factory design pattern that allows developers to interact with the complex OpenMRS services more easily. The object - org.openmrs.api.context.Context is the "Factory Class" that have access to every services (i.e. Authentication, Retrieving Users etc.). In addition, all services are accessed in a static way from the Context object. Hence, we do not need to instantiate a new "Context" object nor new service object every time when we want to have access to an OpenMrs service.

Example 1: Use UserService to get Users with name "bob"

```
UserService userService = Context.getUserService();
List<User> bob = userService.getUsers("bob");
```

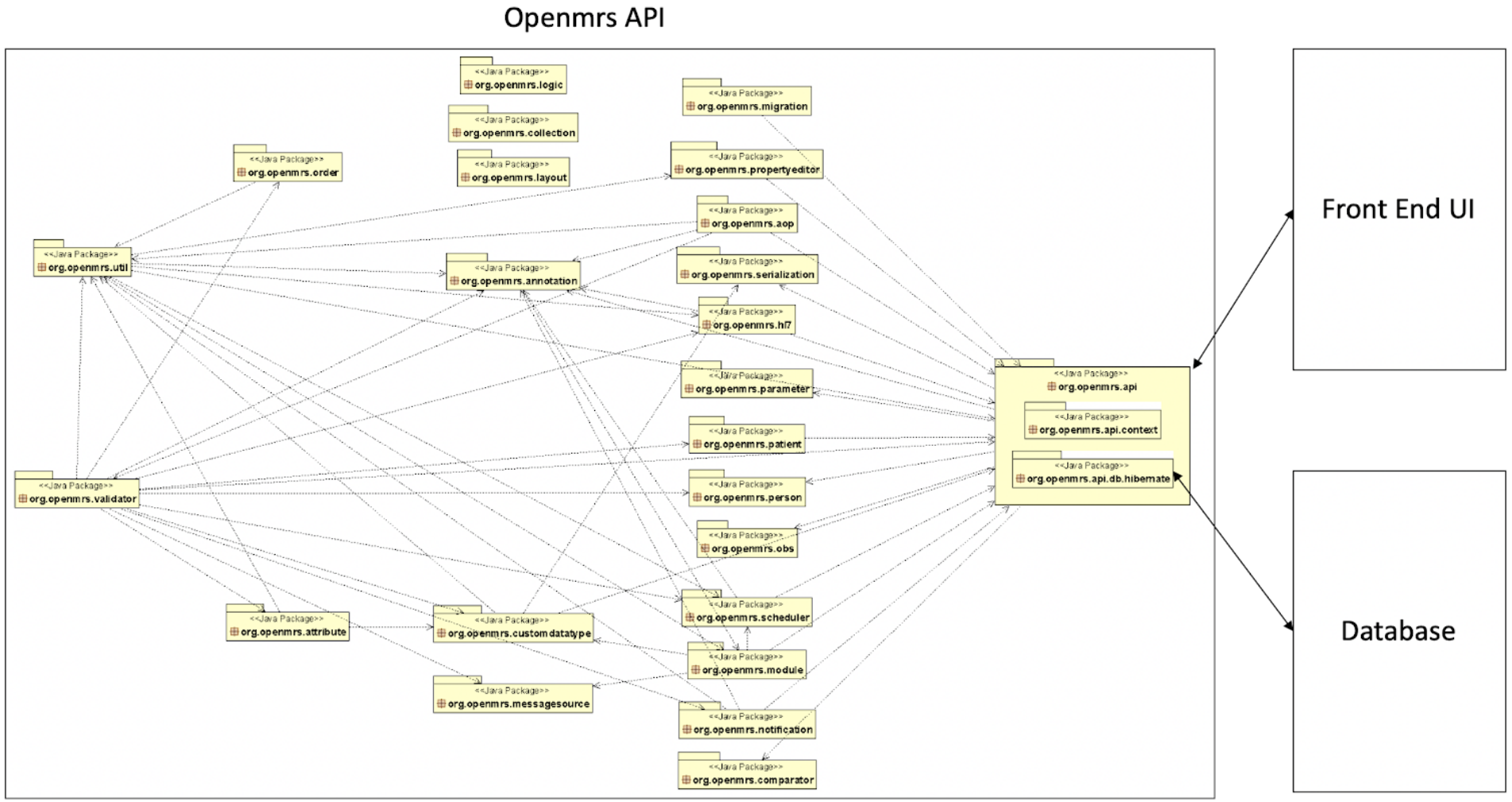
Example 2: Use PersonService to change the gender of the person with id 1

```
PersonService personService = Context.getPersonService();
Person p = personService.getPerson(1);
p.setGender("M");
personService.savePerson(p);
```

2.2 Modular Architecture

The openmrs-core only supports the business logic. It's the "model" in MVC architecture. The "view" and "controllers" are modules and can be customized to suit the user. For instance, the user can choose from different user interface modules or create one themselves. When creating a custom module, users have to follow a specific Module File Architecture in order for openmrs-core to import it. For example, the recommended user interface module is the Legacy UI module. It is in its own repository, so you need to clone the repository, build the maven project, and copy the omod file (a renamed jar file) into the openmrs-core server modules directory. The user interface will then be automatically loaded when mvn jetty:run is run.
(Click here to learn how to create a custom module)

3 UML Diagram



4 Possible Improvements

4.1 Enum Class

The logic.op package currently has a class for every operator that simply overrides the toString method. That could be replaced by an enum class that holds all the constants in one class.

4.2 Appropriate Packages

Many of the classes in the openmrs.org package can be moved into their appropriate sub package or a new sub package. This makes the package more navigable and easier to find classes you're looking for. For example, the VisitAttribute class is only used inside the api package but is outside the package.

4.3 Moving Classes

Classes in org,openmrs.api.order could be moved to other places. The OrderUtil.java could be moved to org.openmrs.api.util and the class DrugSuggestion.java is not being referenced anywhere.

5 Software Development Process (Kanban)

We will be following the Kanban software development process with some slight modifications for our deliverables. The modifications we will make to the process is to assign deadlines for each card because we have to follow the deadlines for our deliverables. We have four columns in our Kanban board's workflow. Requested, Ready to start, In Progress, and Done. We will be setting the WIP limit to be five for the "In Progress" column (one for each member), if a task is blocked, we will increase the WIP limit so that everyone is able to continue working. Afterwards, we will adjust it back. We also order the tasks so that higher priority tasks are higher on the board. We will also have daily team meetings on Discord, similar to scrum stand up meetings. We will not be using swimlanes because our team size is small and have very few tasks in progress at a time.

Pros of Kanban

- Have a visual board to keep track of tasks that need to be accomplished. This allows us to see bottlenecks clearly and then take action to get past those blockers
- This process allows us to make changes depending on the situation
- Having a limited WIP (i.e. Having each person working on 1 task instead of letting them work on multiple tasks) will make tasks get done faster and in a predictable manner because work is pulled, not pushed.
- Because work is not assigned all at once like in Scrum, we can avoid situations where someone ends up with too much work / too little work since everyone pulls a task once they are finished with their current task

Cons of Kanban

- There are no roles in the team, (ex. scrum master) so each member needs to be responsible and diligent with the work they are given.
- Can have daily meetings to ensure that everyone knows what is going on and plan accordingly

- Can lead to issues if Kanban board team members do not update the board when tasks are completed or problems occur, leading to the board to be outdated

6 Why did we choose Kanban?

6.1 Kanban vs Waterfall

We decided against using Waterfall for our project because Waterfall is not very flexible, if a stage takes longer than expected then the whole project will be delayed and because of the time-sensitive nature of our project that can be too costly. Kanban allows each member to work in parallel which saves time. Waterfall is more appropriate for large teams that have little full team communication, but for our project we want to keep everyone informed.

6.2 Kanban vs Extreme Programming

XP requires that the code is working at the end of every iteration, which is very unnecessary because we do not have to deliver after each iteration. XP also requires that you have continuous feedback from the customer, but in this case it may be difficult to communicate with the openmrs customers frequently.

6.3 Kanban vs Incremental

Incremental development is meant for developing a project incrementally, which allows for the deployment of the application in small chunks. In this project, we will not have to release small fixes continuously, thus Incremental was not a good process to choose.