OpenMRS Access Monitor

Group A

Guanming (guanming@mail.sfsu.edu)

Travis Deve

Prakash Gurung

Jiawen Zhu

5/9/2018

Final Project for CSC 668-868 Spring 2018

https://github.com/DreamTeam668-868-team-project/Openmrs-group-a

Table of Contents

Code Base 3

Group Member Contributions 3

Indicate the platform and software 4

Platform: 4

Software: 5

Project description 8

Challenges we faced 9

User Guide 11

Installation 11

Login as a Admin 11

view Monitor 12

1. Check the user access from the chart 12
2. Check smaller range by clicking moving the left and right buttons
3. Move the left and right buttons to see the hourly access 13
4. Switch button for switching between two types of detail tables by Users and by Records 13
5. Search who is access from the chart by searching the date time 14
6. Search a specific user from the chart. 14
7. Quick search by using a prefix 15

Use Cases 15

Design overview 16

1. MVC 18
2. Why do we use MVC 18
3. Where do we use MVC in our project 18
4. Front-end design overview 19

Package diagram/descriptions 21

Class Diagrams 26

Javadoc comments for public classes

Code Base

https://github.com/DreamTeam668-868-team-project/Openmrs-group-a

Group Member Contributions

Guanming

- Leaded and coordinated team members to guarantee the success of the project.
- Designed access_monitor table for saving access records.
- Implemented 2 Spring beans and 1 controller to transfer data by Java
- Implemented 3 methods of controller for supporting Ajax accessing data dynamically by Java
- Redesigned final UI
- Implemented the final UI page by JQuery, HTML, Javascript,CSS
- Implemented a interactive date slider by javascript
- Implemented a dynamic chart by Ajax and Chartjs
- Implemented 2 type of detail table by Ajax and OpenMRS UI Framework
- Tested all module functions from backend to frontend and fixed bugs.
- Provided all demo data for demonstration.
- Codes:

Backend:(Created)

org.openmrs.module.accessmonitor.fragment.controller: AccessMonitorFragFragmentController.java

org.openmrs.module.accessmonitor: ByUserData.java

org.openmrs.module.accessmonitor: ChartData.java

Frontend: (Created)

WebPages.fragments: accessMonitorFrag.gsp

WebPages.resources.scripts: date_slider.js

WebPages.resources.scripts: moment.min.js

WebPages.pages: accessMonitor.gsp

WebPages.resources.scripts: Chart.js

WebPages.resources.scripts: analytics.js

WebPages.resources.scripts: utils.js

Config files: (Modified)

liquibase.xml

settings.xml

pom.xml

modeulApplicationContext.xml

accessmonitor_app.json

Travis

- Created the skeleton code for the basis of the project
- Implemented the POJO for hibernate mapping
- Implemented the Interfaces the OpenMRS Context framework
- Implemented the DAO classes for Hibernate table management
- Created relevant liquibase and hibernate files for the database creation and mapping
- Implemented the advice classes for the backend of the project
- Classes:

  org.openmrs.module.accessmonitor.advice: PatientAdvice.java

  org.openmrs.module.accessmonitor.advice: PersonAdvice.java

  org.openmrs.module.accessmonitor.advice: OrderAdvice.java

  org.openmrs.module.accessmonitor.advice: VisitrAdvice.java

  org.openmrs.module.accessmonitor: AccessMonitor.java

  org.openmrs.module.accessmonitor.api: AccessMonitorService.java

  org.openmrs.module.accessmonitor.api.impl: AccessMonitorServiceImpl.java

  org.openmrs.module.accessmonitor.api.db: AccessMonitorDAO.java

  org.openmrs.module.accessmonitor.api.db.hibernate: HibernateAccessMonitorDAO.java

  Files:

  Access Monitor Module:

pom.xml

settings.xml

AccessMonitor Module API:

Liquibase.xml

AccessLog.hbm.xml

modeulApplicationContext.xml

AccessMonitor Module OMOD:

Pom.xml

Config.xml

Prakash Gurung

- Created mock up UIs using a couple of frameworks such as React, and Highchart to facilitate our needs and specs.
- Crated mockup highcharts both with dynamic (Json data) and static values.
- Created a webpage to display charts, filters, and search.

Jiawen

- Create four filter for users, type. Location and patients.
- Create four search boxes for users , type, location and patients.
- Create a logic for search boxes which will get static data from Json and display on index.html
- Create a logic for calculating the # of calls for each user based on the timestamp.

Indicate the platform and software

Platform:

1. OpenMRS Reference Application 2.4
2. Spring framework

Included in OpenMRS, this is used to implement Aspect Oriented Programming

3. Liquibase
4. Hibernate

Included in OpenMRS, this is used to connect Java objects to database tables

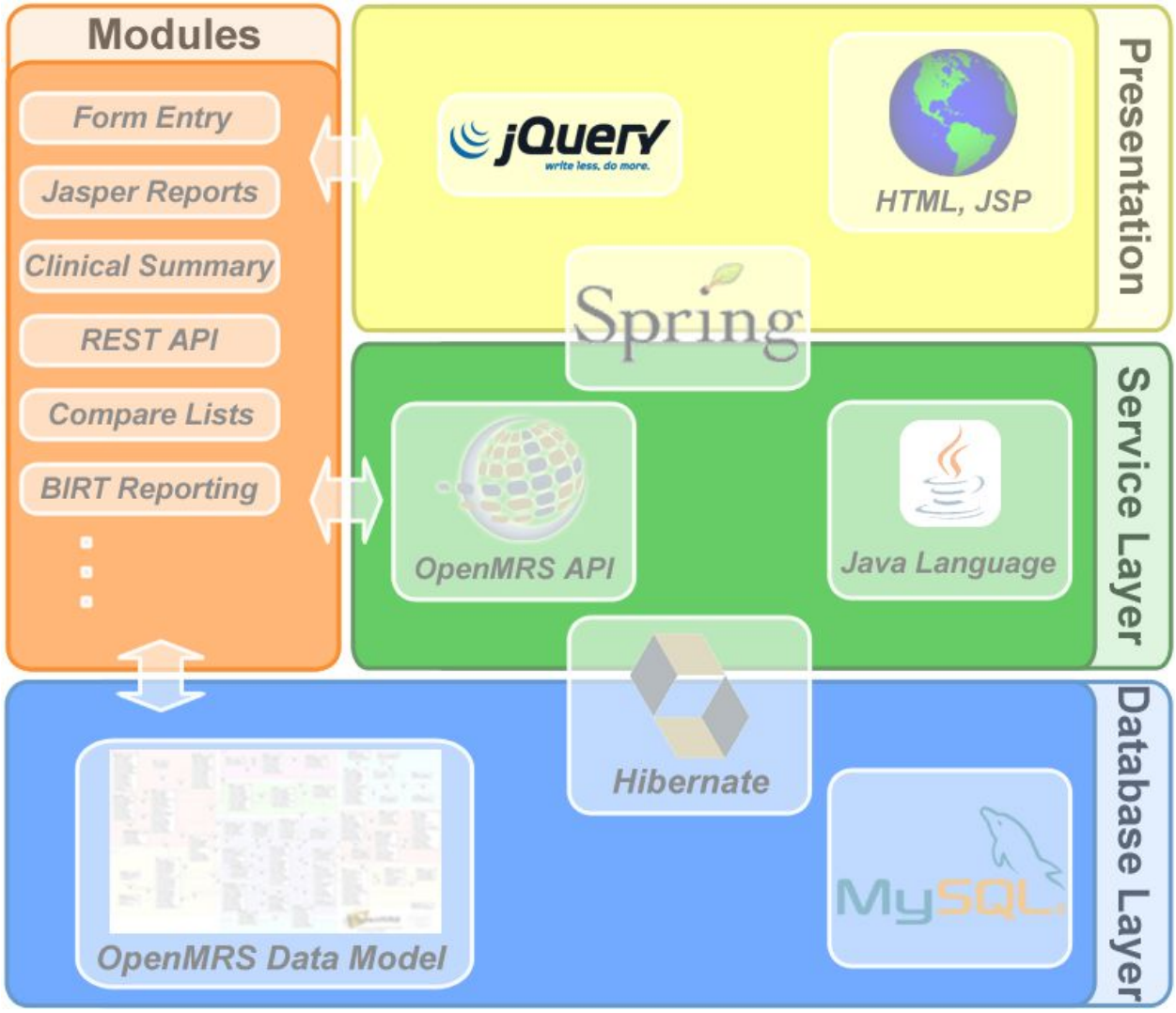
5. Groovy
6. Java 7 JDK

Software:

1. Netbeans 8.2

Platform Information

Figure 1, OpenMrs High level Architecture diagram



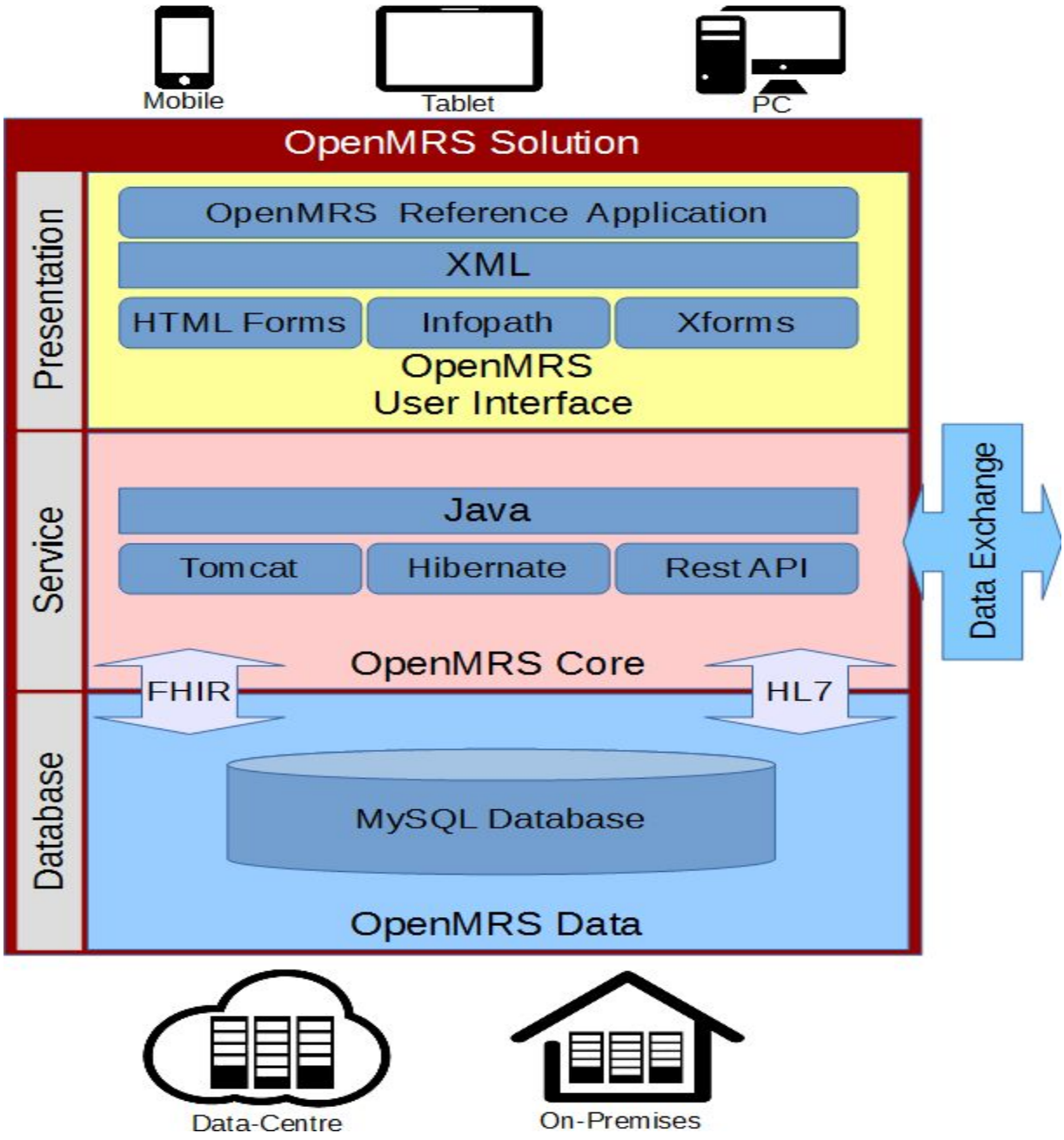
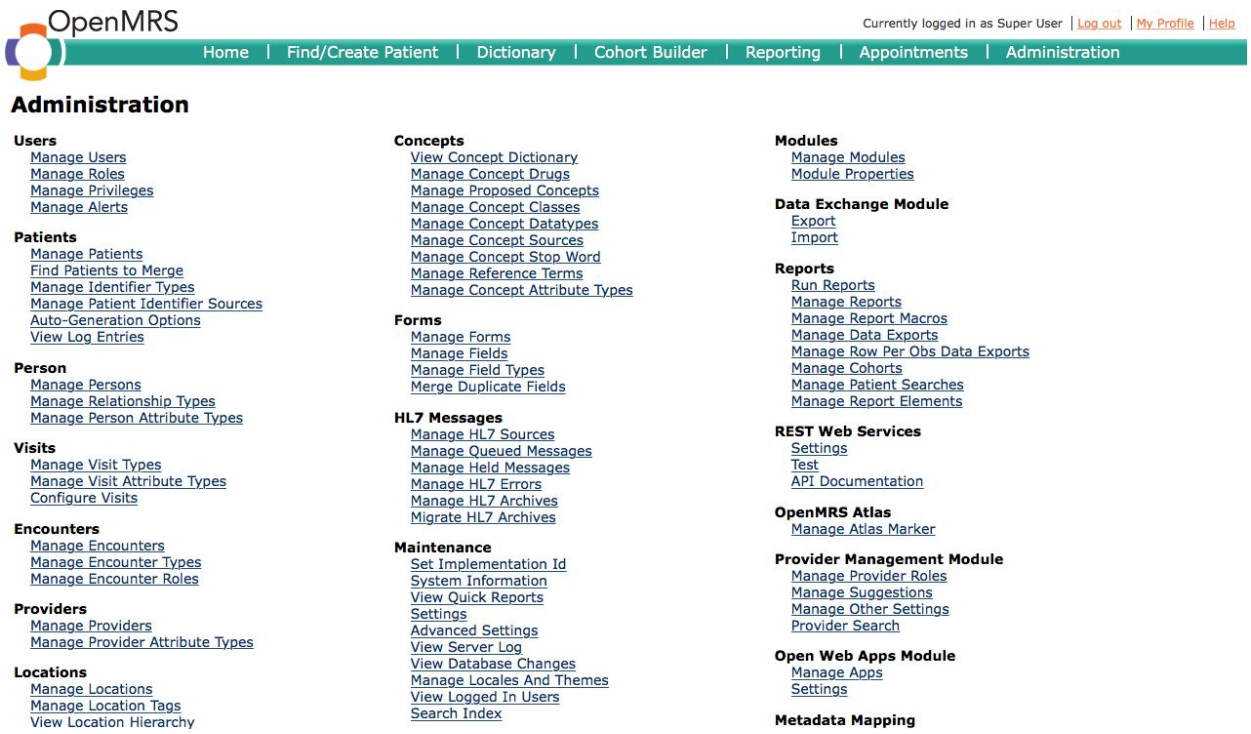


Figure 1.: OpenMRS Administration



Project description

The aim of our project was to create a new module for the OpenMRS system that would allow a system admin to monitor and view attempts to access the system and help identify potential malicious uses of the software, with one of the main concerns being the protection of patient privacy.

To accomplish this, we are logging various method calls through Aspect Oriented Programming through the Java Spring framework. Spring allows a programmer to wrap specific classes, known as advice classes, around method calls made via an interface during program execution, and even prevent execution of the intercepted code and return a dummy value, if desired.

To support this, we also created a database table through liquibase to store the records of access attempts, which is defined and managed by the Hibernate ORM. These records are then recalled, parsed, and displayed in a graph format using the Spring MVC Framework, Ajax, and JavaScript, the display of which can be changed dynamically in order to make it easier for patterns to be detected in the display of record accesses. While we had initially planned for

multiple graphs and filters, this ended up being too ambitious for the timeframe of the project, and we dropped those features.

The data we are storing in the user which triggered the initial action, what kind of record they were accessing, the date and time of the access, and what kind of action the access represented (i.e. creation of a record, a retrieval, or a deletion, etc.). We had also planned to store the location from which the access took place, but hit a roadblock in the development process and were unable to include this.

Challenges we faced

1. Lack of understanding of OpenMRS
2. The documentation of OpenMRS is inconsistent, which makes it very hard to gain understanding of what some methods are actually doing
3. The timeframe of the project was incredibly small, and the group did not take this seriously enough

- Development Process

- We had a diverse set of tools to achieve our goal within OpenMRS
- Most work was done in Netbeans(8.2)
- Most coding was in Java (at various levels of sophistication)
- Javascript, Jquery
- XML, Liquibase, Hibernate, Spring Framework
- We also had to deal with and work around (mainly for documentation) each of us having different operating systems

- Overcoming Issues

- Schedules & Communications
- We all had different schedules & Locations
- This led to issues with dependencies and communication
- We would meet after class
- Several times in the Computer Lab (2-3 hours each time)
- Communicate via Slack

User Guide

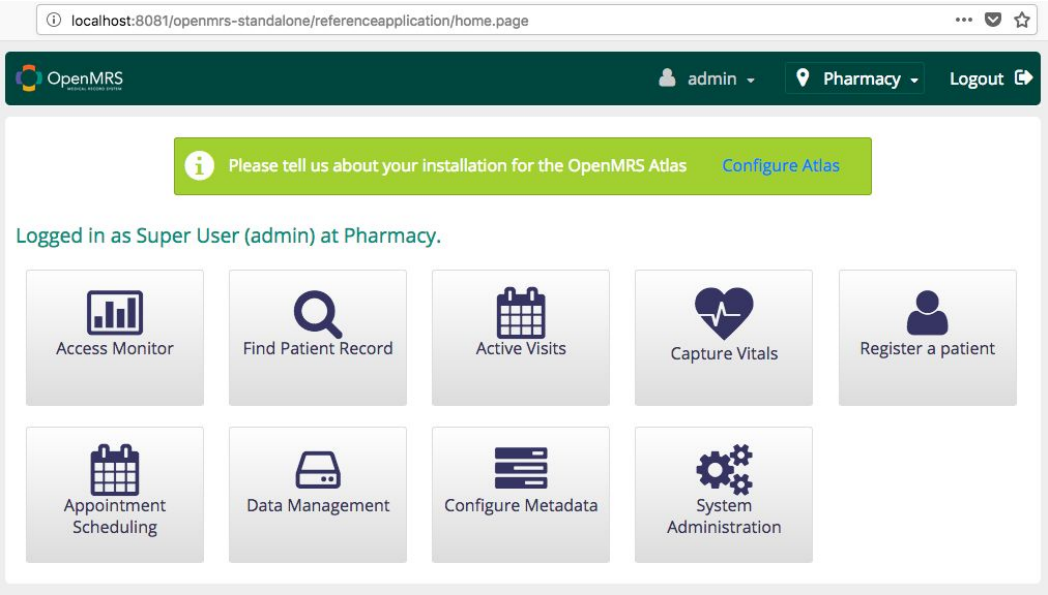
This guide describes how to use OpenMrs Access Monitor software step by step.

Installation

In order to utilize the OpenMRS Access Monitor , and administrator must install the module into OpenMRS. This can be done through the OpenMRS web interface by navigating to the "Administration" menu and following the link "Manage Modules." From there, click "Add or Upgrade Module"; under "Add Module", click browse. Then, locate and select the Module. Once this has been selected, click "Upload." OpenMRS should then install and start the module, and notify you of any issues with the installation process.

Login as a Admin

Log onto OpenMRS with your corresponding Admin name and password. Passwords are case-sensitive. Once logged in, you will see the following screen.

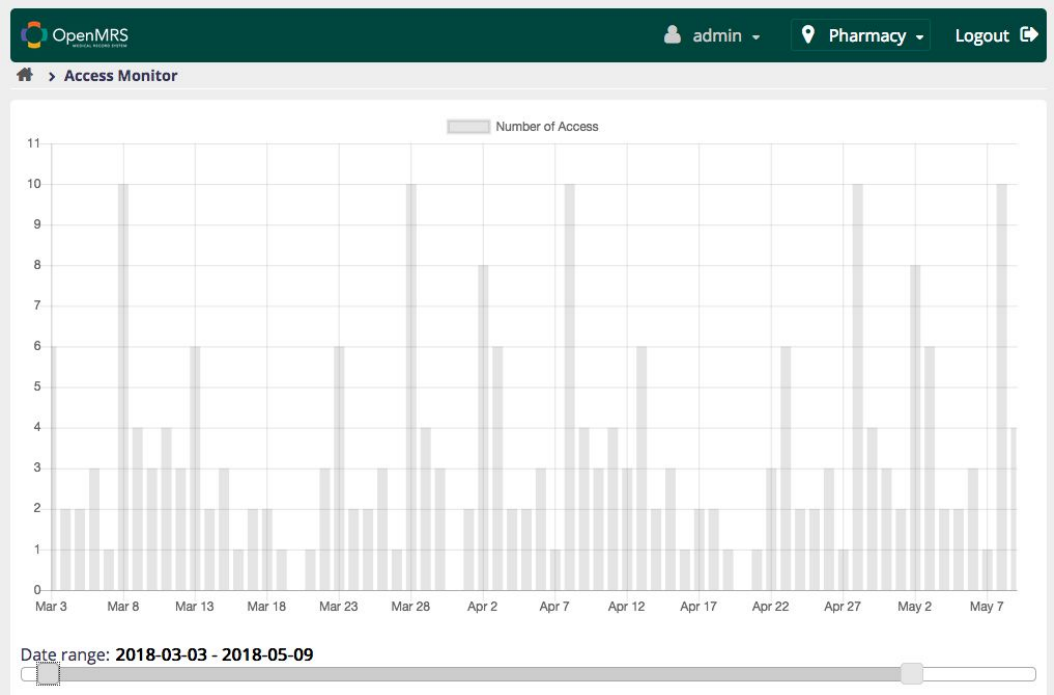


Go ahead and click the Access Monitor button
The first module that you are seeing is Access Monitor that we have developed.
Once you click the Access Monitor , you will be greeting by following screen

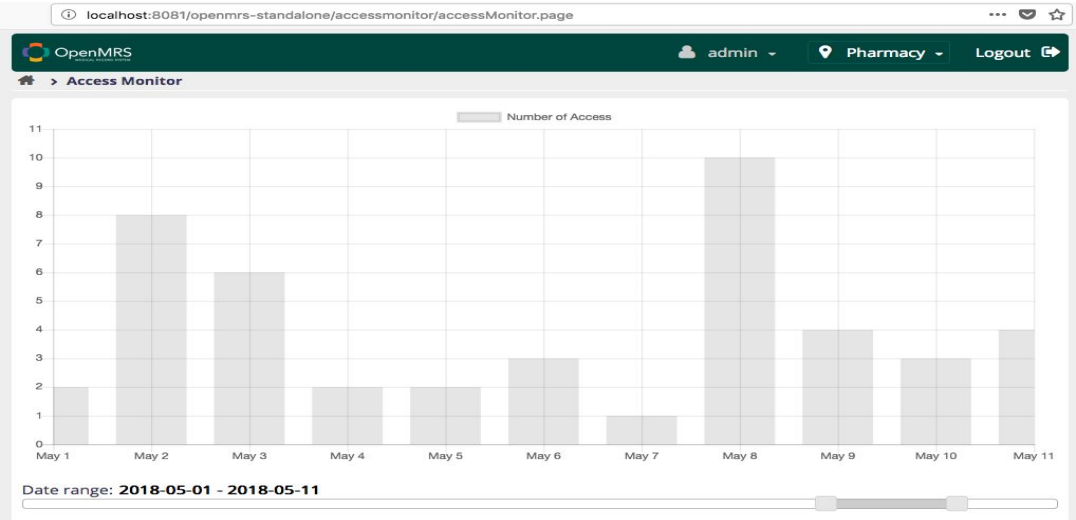
view Monitor

This is the page where you will see all the details such as number of data accessed by an employee, timestamps, slider to zoom in and zoom out, and the detailed table view of every logs.
The following diagram shows the general overview of our access monitor.

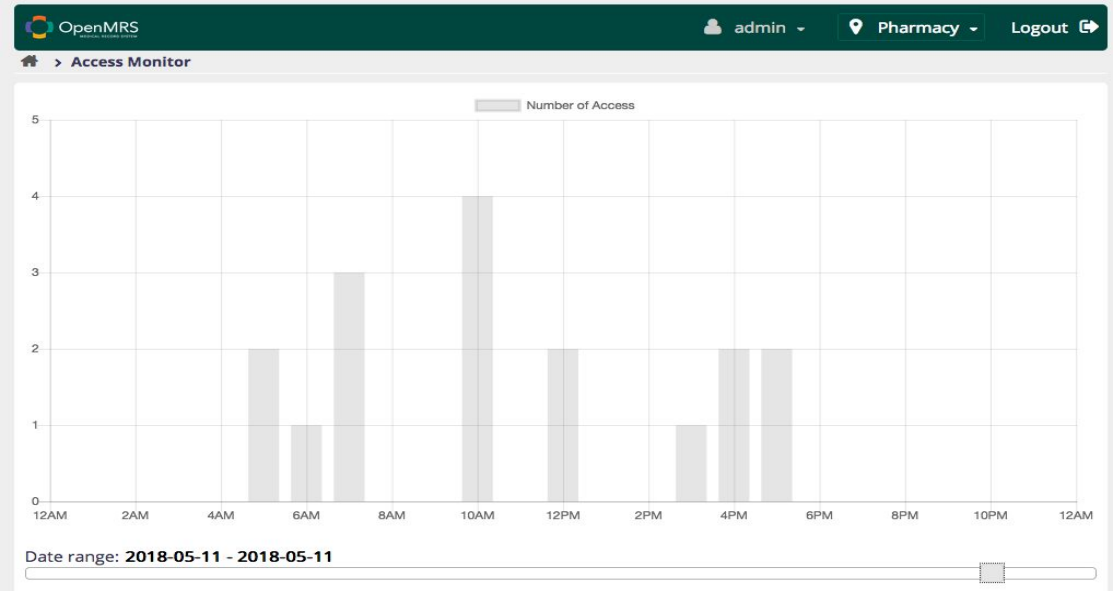
1. Check the user access from the chart



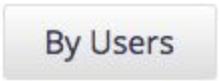
2. Check smaller range by clicking moving the left and right buttons



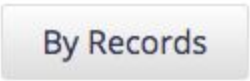
3. Move the left and right buttons to see the hourly access.



4. Switch button for switching between two types of detail tables by Users and by Records



or



5. Search who is access from the chart by searching the date time.

Search: Show 10 entries

User Id	Person Name	Number of Access	Time Frame
1	Super User	1	2018-05-11 05:00:00 - 2018-05-11 06:00:00
2	Jane Smith	1	2018-05-11 05:00:00 - 2018-05-11 06:00:00
1	Super User	1	2018-05-11 06:00:00 - 2018-05-11 07:00:00
1	Super User	3	2018-05-11 07:00:00 - 2018-05-11 08:00:00
1	Super User	4	2018-05-11 10:00:00 - 2018-05-11 11:00:00
1	Super User	2	2018-05-11 12:00:00 - 2018-05-11 13:00:00
1	Super User	1	2018-05-11 15:00:00 - 2018-05-11 16:00:00
1	Super User	2	2018-05-11 16:00:00 - 2018-05-11 17:00:00
1	Super User	2	2018-05-11 17:00:00 - 2018-05-11 18:00:00

Showing 1 to 9 of 9 entries First Previous 1 Next Last

Description: The chart down below shows a detailed table by **Users**. There are 4 categories. First **UserId**; which is the id of the particular user, second, **Person Name**; which is self-explanatory, third, **Number of Access;** which shows how many number of access that the particular person made, and the last one, **TimeFrame;** which displays the timestamp.

6. Search a specific user from the chart.

Search: jane Show 10 entries

Record Id	Record Type	Access Type	User Id	Person Name	Timestamp
31	PATIENT	RETRIEVAL	2	Jane Smith	2018-05-02 22:36:34
26	PATIENT	UPDATE	2	Jane Smith	2018-05-02 22:36:35
31	PATIENT	RETRIEVAL	2	Jane Smith	2018-05-02 22:36:35
26	VISIT	RETRIEVAL	2	Jane Smith	2018-05-04 23:19:09
31	PATIENT	RETRIEVAL	2	Jane Smith	2018-05-05 23:19:09
26	PATIENT	RETRIEVAL	2	Jane Smith	2018-05-05 23:19:09
31	PATIENT	RETRIEVAL	2	Jane Smith	2018-05-06 22:36:34
26	PATIENT	UPDATE	2	Jane Smith	2018-05-08 22:36:35
31	PATIENT	RETRIEVAL	2	Jane Smith	2018-05-08 22:36:35
26	VISIT	RETRIEVAL	2	Jane Smith	2018-05-09 23:19:09

Showing 1 to 10 of 13 entries (filtered from 45 total entries) First Previous 1 2 Next Last

description : The chart down below shows a detailed table by **Records**. There are 6 categories. First **RecordId**, which is the id of the data, second one is **RecordType;** which is the type of

record data, third one is **AccessType;** which will display what type of access it is, fourth is the **UserId**; which is the id of the particular user, fifth is the **Person Name**; which is self-explanatory, and the last one, **TimeFrame;** which displays the timestamp.

7. Quick search by using a prefix

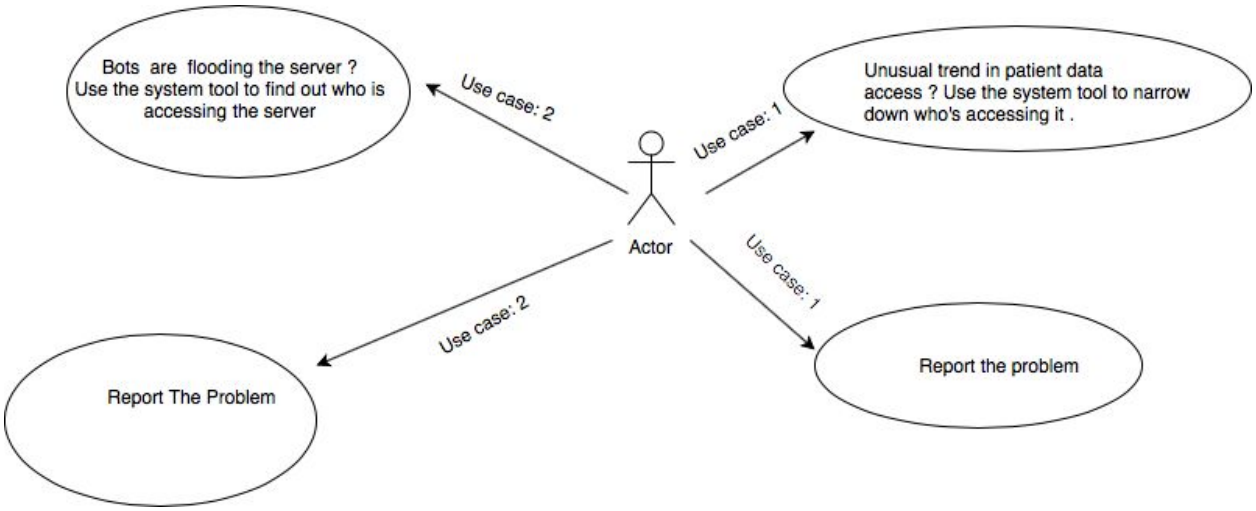
Search: Ju Show 10 entries

User Id	Person Name	Number of Access	Time Frame
6	Julie Smith	1	2018-05-02 00:00:00 - 2018-05-03 00:00:00
6	Julie Smith	1	2018-05-06 00:00:00 - 2018-05-07 00:00:00
6	Julie Smith	1	2018-05-10 00:00:00 - 2018-05-11 00:00:00

Showing 1 to 3 of 3 entries (filtered from 32 total entries) First Previous 1 Next Last

Chart demonstrating the spike of number of access which is part of our use case(Data breach/ potential hacking)

Use Cases



A **Security Administrator**, who checks the system for suspicious behavior regularly, notices a trend in patient data access that seems suspicious. Using the system tools, they are able to narrow it down to a nightly occurrence, where patient data is being accessed at the end of one of the employee shifts.

Example 1: (Tracking methods call report)
Jenny is a security admin. One day she sees a lot of unusual number of drug orders being placed by a member of staff, Doctor Lee. She also notices that Doctor Lee made these orders on a Saturday, a day he is not generally scheduled to work. She will take some action by calling him on Monday about the suspicious activity.

Assumption for example 1:
Jenny needs to know each person's schedule at her clinic. She is responsible for checking the graph to see if there is any activity that is suspicious to her knowledge.

Use case #	1
Use case name	Detect Suspicious activity
Summary	The **Security Admin** notice a trend in patient data access that seems suspicious.
Actor	Security Admin
Description	- System Admin will regularly check the system if there is any suspicious activity. - System Admin uses system tools to narrow down the drug orders
Precondition	NA
Postcondition	Using the system tools, they are able to narrow it down to a nightly occurrence, where patient data is being accessed at the end of one of the employee shifts.

A **Security Administrator**, notices a sudden spike in data access., where one user is accessing the data repeatedly. Using the system tools, the system admin shall be able to pinpoint the certain rise in data access and narrow down the cause.

Example 2: (potential hacking track)
On another day, Jenny sees an unusual amount of activity on the system. She notices that an employee is viewing a large number of records, more than would generally be made for normal work purposes. She will alert the user of the potential breach and have temporarrily deactivate the employees account. She will then have the employees password changed, and alert the patients of the potential breach of privacy.

Assumption for example 2:
Jenny is knowledgeable about the system and the general workflow of employees. Jenny must be able to determine what would be reasonable for employees to do in a given timeframe.

Use case #	2
Use case name	Detect Hacking/Data Breach
Summary	**Security Admin** notices a possible data breach/ hacking where an anonymous user is accessing the data repeatedly
Actor	**Security Admin**
Description	- **S**ecurity Admin uses the system tool and finds a spike in the data access. - Security Admin assessing the issue by using the 'Slider' function to zoom in and zoom out the total number of order placed.
Precondition	NA

Postcondition	Using the system tools, the system admin shall be able to pinpoint the certain rise in data access and narrow down the cause.

Design overview

Design pattern used:

1 MVC

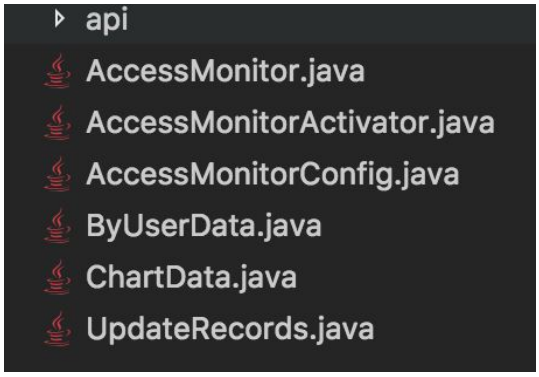
Model–view–controller is commonly used for developing software that divides an application into three interconnected parts. This is done to separate internal representations of information from the ways information is presented to and accepted from the user. Model is the data which determined by controller; View is what user see in front of the screen; Controller determine the type of data.

2. Why do we use MVC

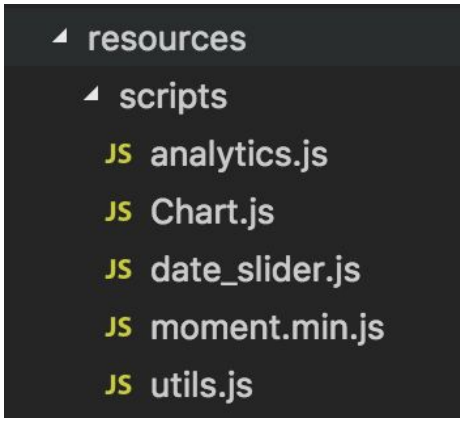
The MVC design pattern decouples these major components allowing for efficient code reuse and parallel development.

3. Where do we use MVC in our project

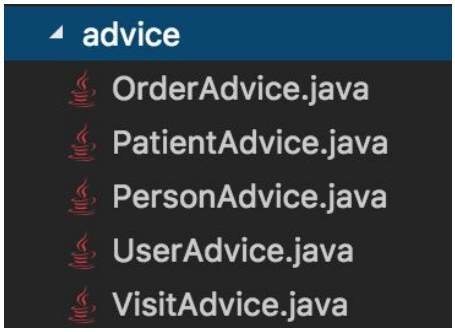
Model



view



Controler



4. Front-end design overview:

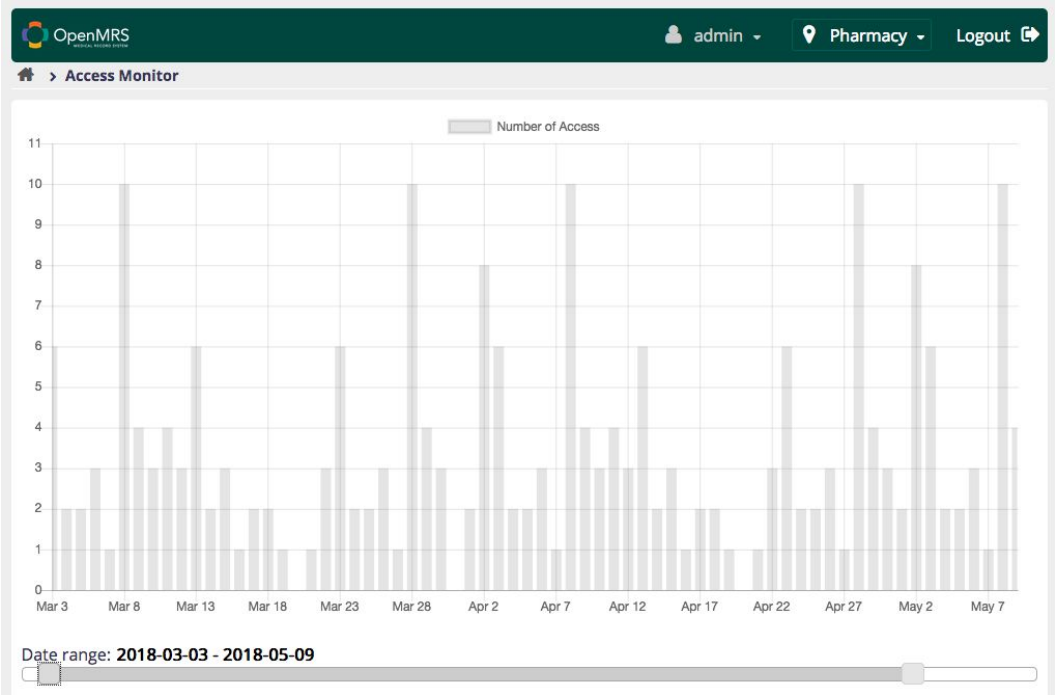
The whole project is based on the security alert system. The project can divided into front-end design and backend design.

The front-end mainly deal with javaScripts classes that with postfix .js in the end. These class define the view of our chart and the function that go along with it.

We are going to talk about all the function in our charts. We can divide these into 4 parts which are list below.

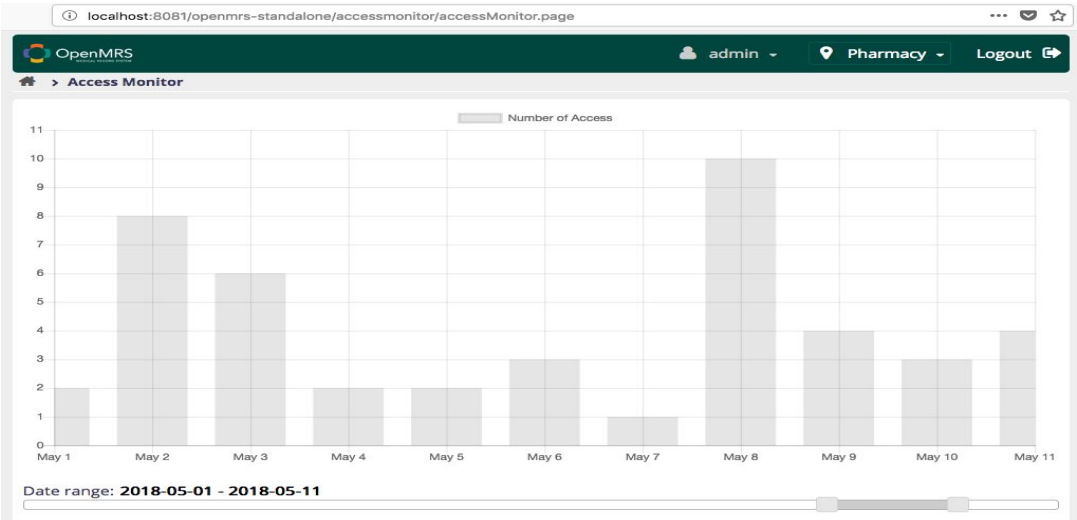
The front-end design will focus on (**1) the view of chart, (2) the data of the chart, (3) the function of the chart and (4) the top access from the chart.**

(1) The view of the chart :



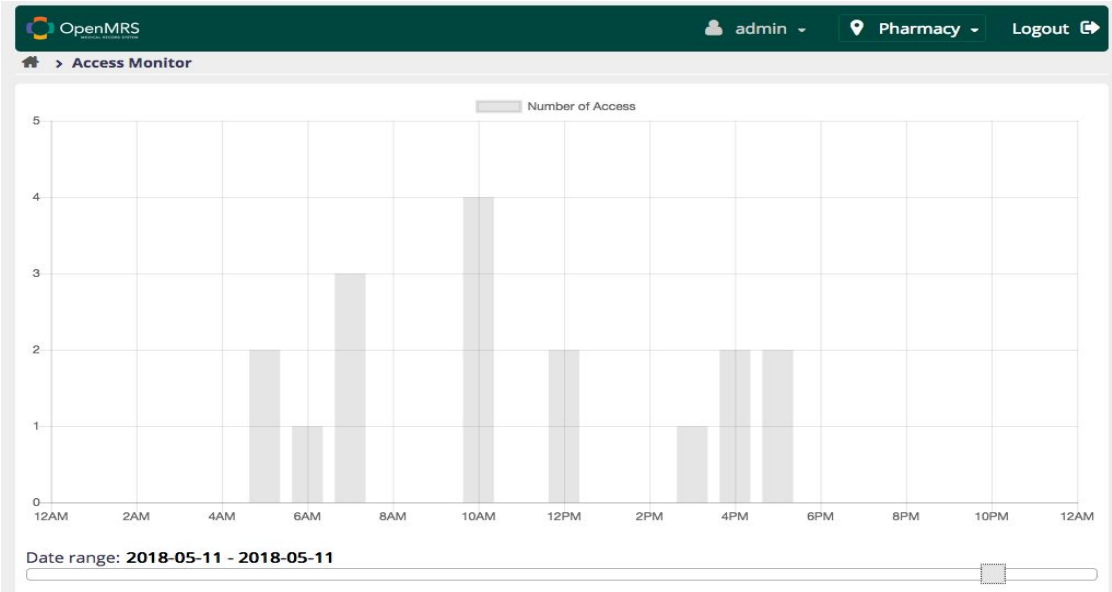
Description: The x axis are dates and hours (daily divided and hourly divided which will be explain in detail later). The y axis are how many access that are being generated by the controller.

Dates (Daily divided)



Description: X axis are daily divided. Y axis are number of total user access.

hours(hourly divided)



Description: X axis are hourly divided. Y axis are number of total user access.

(2) the data of the chart:

the data is stored in the json file. The jason is just a javaScript file with data inside of it's array. In each array, there are timestamp stored in the array corresponding to each key.

For instance, the {"key:" "value:"[1167609600000

, 1167782400000]}

(3) The function of the chart:

Function description in depths:

1. Select a time to view the chart.
2. Select and move left button to go back in time and graph will be changed according .
3. Select and move right button to go future in time.
4. Move the left and right button at same spot to see the hourly graph.

(4) top access from the chart:

The table shows the how many users will accessing the data from the selected date.

Search: Show 10 entries

User Id	Person Name	Number of Access	Time Frame
1	Super User	1	2018-05-11 05:00:00 - 2018-05-11 06:00:00
2	Jane Smith	1	2018-05-11 05:00:00 - 2018-05-11 06:00:00
1	Super User	1	2018-05-11 06:00:00 - 2018-05-11 07:00:00
1	Super User	3	2018-05-11 07:00:00 - 2018-05-11 08:00:00
1	Super User	4	2018-05-11 10:00:00 - 2018-05-11 11:00:00
1	Super User	2	2018-05-11 12:00:00 - 2018-05-11 13:00:00
1	Super User	1	2018-05-11 15:00:00 - 2018-05-11 16:00:00
1	Super User	2	2018-05-11 16:00:00 - 2018-05-11 17:00:00
1	Super User	2	2018-05-11 17:00:00 - 2018-05-11 18:00:00

Showing 1 to 9 of 9 entries First Previous 1 Next Last

The image shows that **top 10 patients** whose information has been access during the from May 08 - May 12.

Addition functions: the search will get match any data type listed in the table. For example, a user can search for user id, person's name, # of access and access time.

prefix search: type a few prefix words to see the result. Here is the example of person name search.

Search: Ju Show 10 entries

User Id	Person Name	Number of Access	Time Frame
6	Julie Smith	1	2018-05-02 00:00:00 - 2018-05-03 00:00:00
6	Julie Smith	1	2018-05-06 00:00:00 - 2018-05-07 00:00:00
6	Julie Smith	1	2018-05-10 00:00:00 - 2018-05-11 00:00:00

Showing 1 to 3 of 3 entries (filtered from 32 total entries) First Previous 1 Next Last

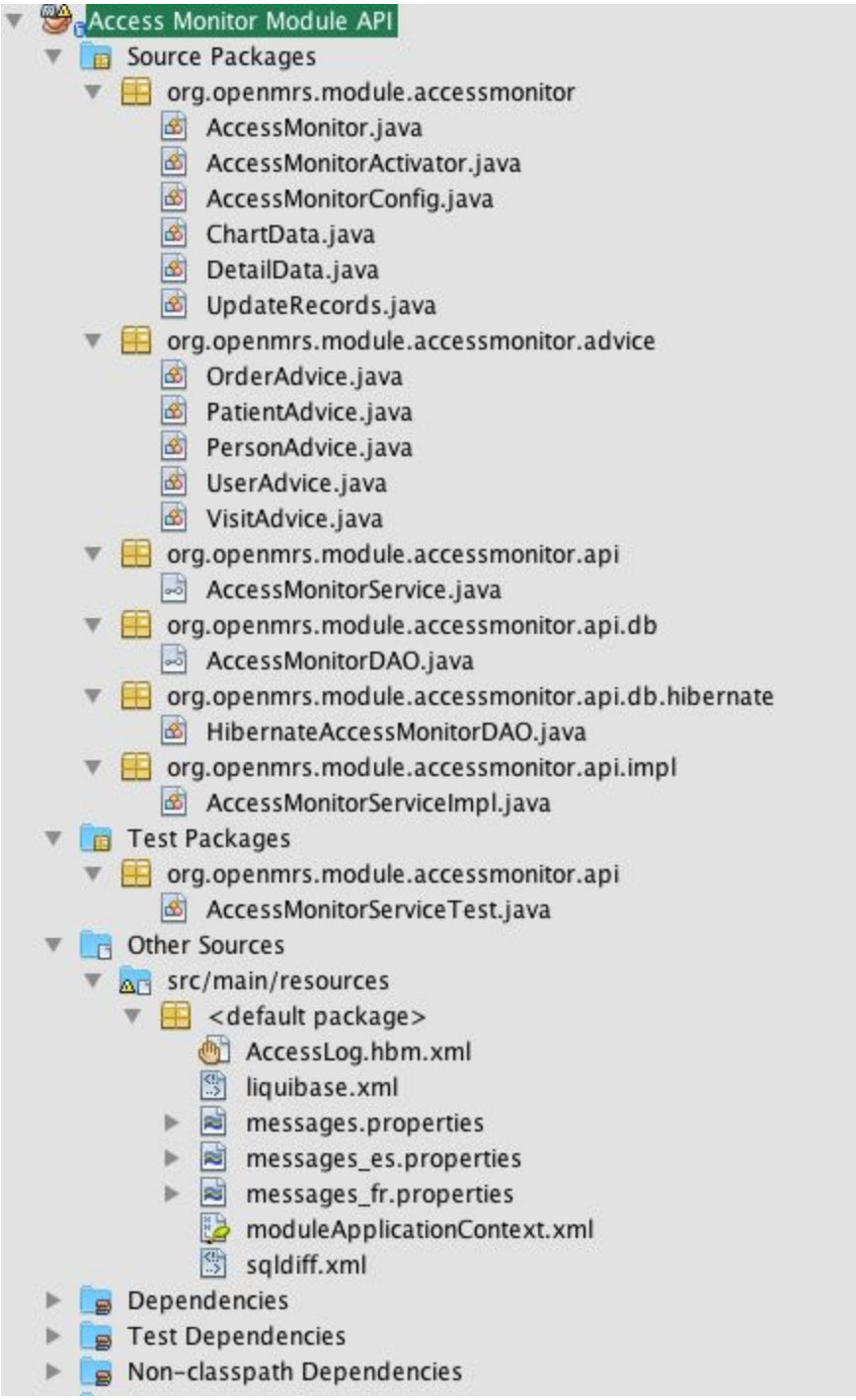
External documentation on algorithms, etc., as appropriate

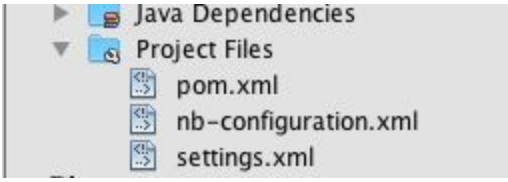
Since the scope of our project is to identify threads in the medical record system. The code we write has not involved with much algorithms.

That's not saying we are not going to improve our code efficiency. After implement all the feature, the next step is to take look at our code and try to improve our code efficiency. We will do a code analysis to see if there is any redundant code or code that doesn't follow the MVC pattern.

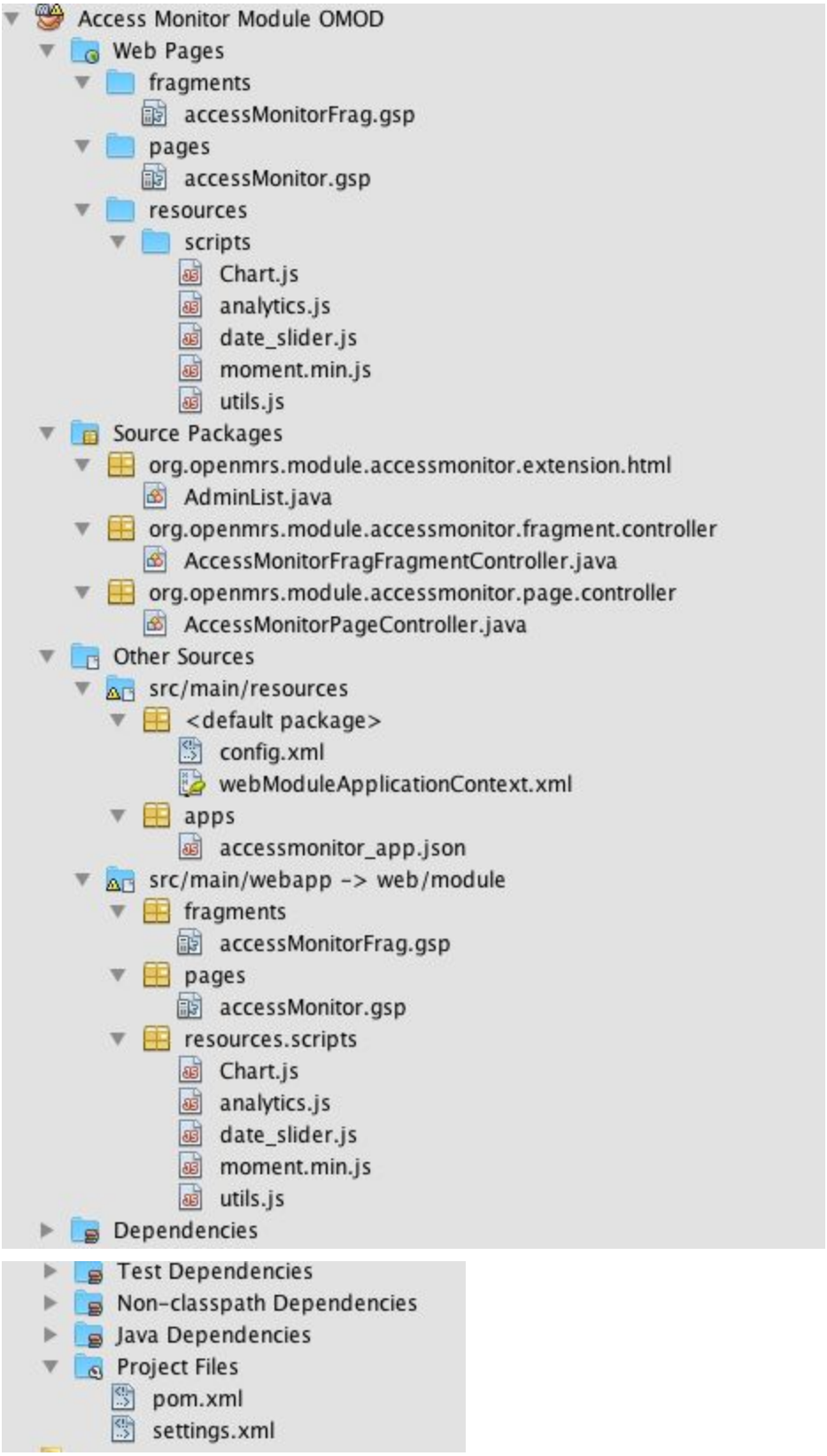
Package diagram/descriptions

Package diagram**; Access Monitor Module Api**



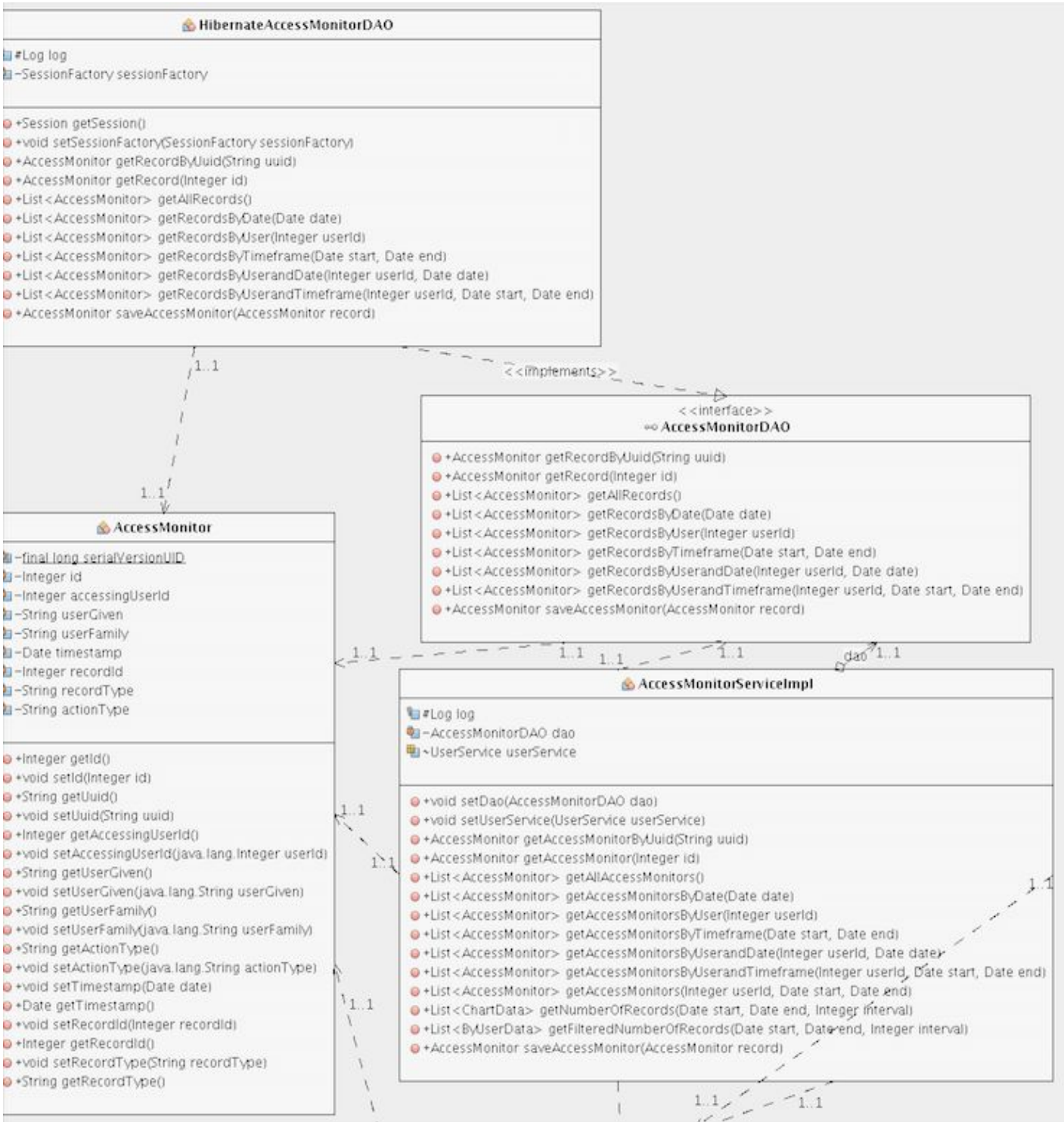


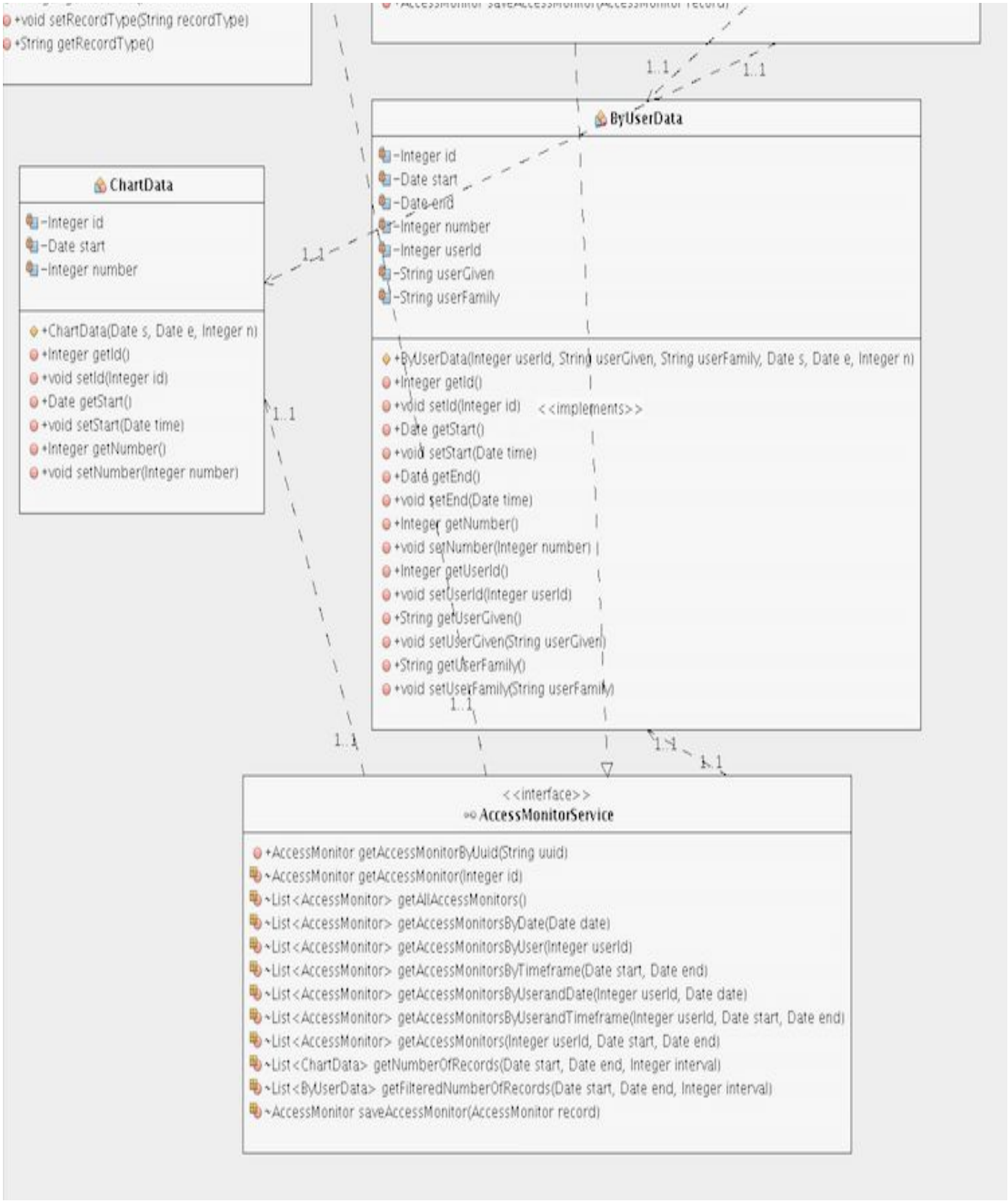
Access Monitor Module OMOD:


Access Monitor Module OMOD
Web Pages
fragments
accessMonitorFrag.gsp
pages
accessMonitor.gsp
resources
scripts
Chart.js
analytics.js
date_slider.js
moment.min.js
utils.js
Source Packages
org.openmrs.module.accessmonitor.extension.html
AdminList.java
org.openmrs.module.accessmonitor.fragment.controller
AccessMonitorFragFragmentController.java
org.openmrs.module.accessmonitor.page.controller
AccessMonitorPageController.java
Other Sources
src/main/resources
<default package>
config.xml
webModuleApplicationContext.xml
apps
accessmonitor_app.json
src/main/webapp -> web/module
fragments
accessMonitorFrag.gsp
pages
accessMonitor.gsp
resources.scripts
Chart.js
analytics.js
date_slider.js
moment.min.js
utils.js
Dependencies
Test Dependencies
Non-classpath Dependencies
Java Dependencies
Project Files
pom.xml
settings.xml

Class Diagrams

Class Diagram; Access Monitor Module Api

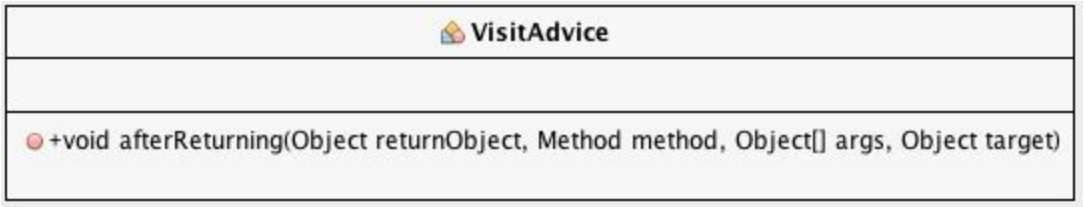



+void setRecordType(String recordType)
+String getRecordType()
ChartData
-Integer id
-Date start
-Integer number
+ChartData(Date s, Date e, Integer n)
+Integer getId()
+void setId(Integer id)
+Date getStart()
+void setStart(Date time)
+Integer getNumber()
+void setNumber(Integer number)
ByUserData
-Integer id
-Date start
-Date end
-Integer number
-Integer userId
-String userGiven
-String userFamily
+ByUserData(Integer userId, String userGiven, String userFamily, Date s, Date e, Integer n)
+Integer getId()
+void setId(Integer id)
<<implements>>
+Date getStart()
+void setStart(Date time)
+Date getEnd()
+void setEnd(Date time)
+Integer getNumber()
+void setNumber(Integer number)
+Integer getUserId()
+void setUserId(Integer userId)
+String getUserGiven()
+void setUserGiven(String userGiven)
+String getUserFamily()
+void setUserFamily(String userFamily)
1..1
<<interface>>
AccessMonitorService
+AccessMonitor getAccessMonitorByUuid(String uuid)
~AccessMonitor getAccessMonitor(Integer id)
~List<AccessMonitor> getAllAccessMonitors()
~List<AccessMonitor> getAccessMonitorsByDate(Date date)
~List<AccessMonitor> getAccessMonitorsByUser(Integer userId)
~List<AccessMonitor> getAccessMonitorsByTimeframe(Date start, Date end)
~List<AccessMonitor> getAccessMonitorsByUserandDate(Integer userId, Date date)
~List<AccessMonitor> getAccessMonitorsByUserandTimeframe(Integer userId, Date start, Date end)
~List<AccessMonitor> getAccessMonitors(Integer userId, Date start, Date end)
~List<ChartData> getNumberOfRecords(Date start, Date end, Integer interval)
~List<ByUserData> getFilteredNumberOfRecords(Date start, Date end, Integer interval)
~AccessMonitor saveAccessMonitor(AccessMonitor record)

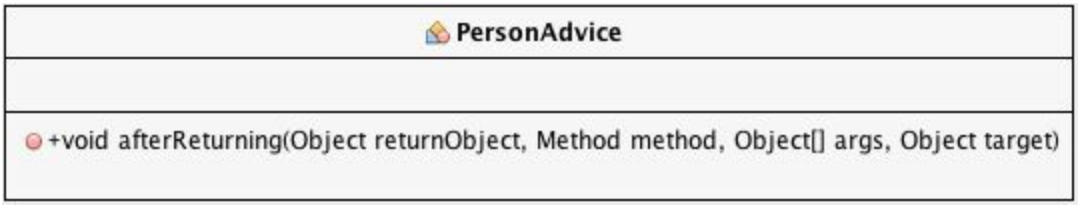
Full High resolution class diagram can be accessed here: https://imgur.com/DZlD5Fl

Individual Class Diagram:

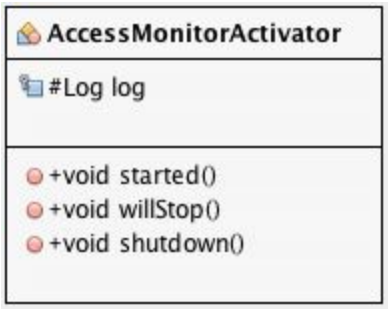
1) VisitAdvice



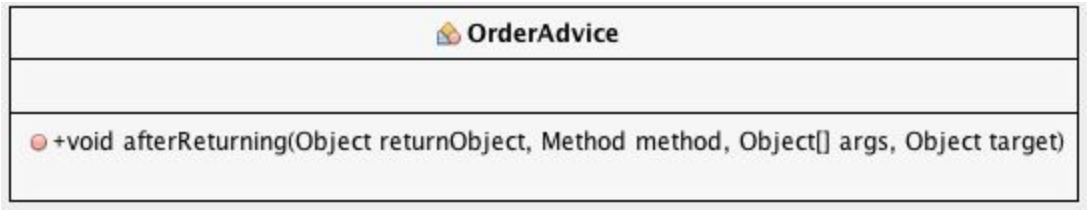
3) PersonAdvice



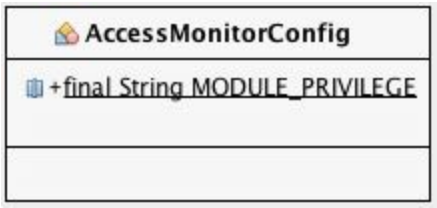
4) AccessMonitorActivator



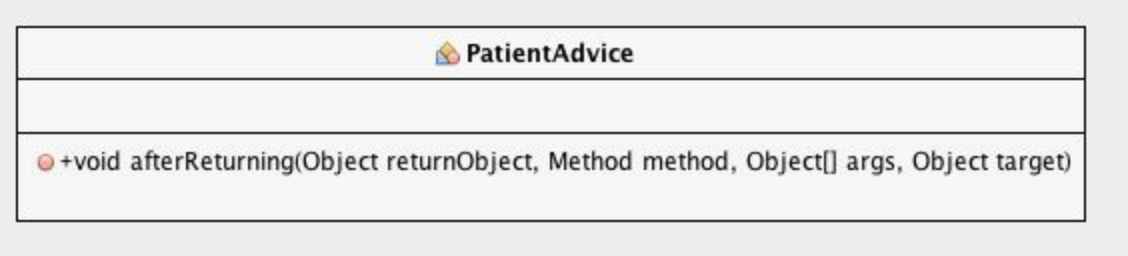
5) OrderAdvice



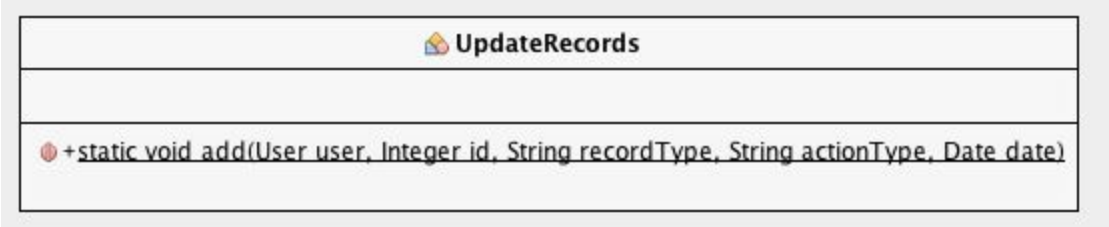
6) AccessMonitorConfig



7) PatientAdvice

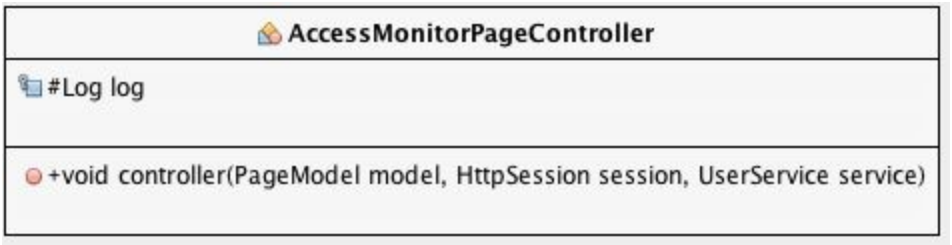


8) UpdateRecords

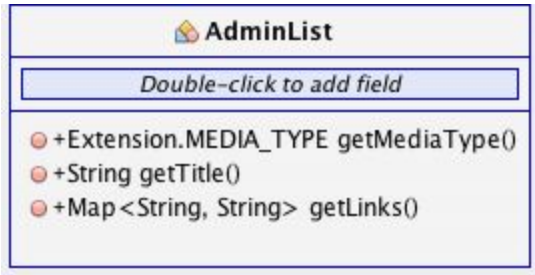


Individual Class Diagram: Access Monitor Module OMOD

1) AccessMonitorPageController



2) AdminList



3) AccessMonitorFragFragmentController