# OSCAR Architectural Notes

## Document History

**2016-01-29:** Created - Simon Diemert
**2016-02-12:** Added code level architecture - Simon Diemert

## Introduction

### Purpose

This document outlines the current architecture of the OSCAR restful web service and "new" (OSCAR 14) user interface. This is intended to be a guide to assist developers who are working with features related to the OSCAR 14 interface or web service. The "tickler" and "medication summary" features are used as a working examples.

### Context

OSCAR is a real-world legacy software system. To date, little documentation exists that discusses the architecture of OSCAR. Thus the task of exploring and documenting the architecture of the system in a manner that is easily consumable to developers was undertaken.

### Background

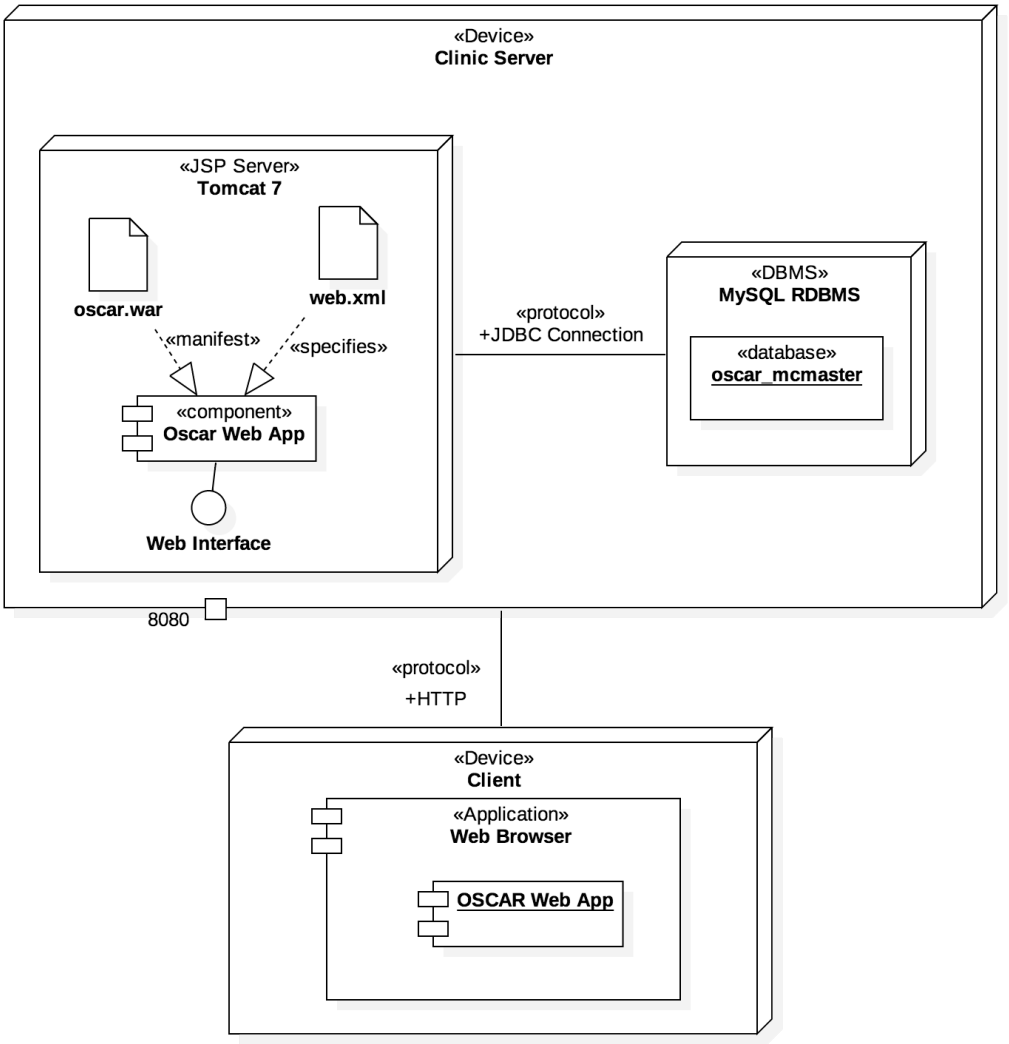
OSCAR (Open Source Clinical Application Resource) is an Electronic Medical Record (EMR) system used in primary and secondary care offices throughout Canada. The primary stakeholders are the OSCAR community, comprised mostly of physician users; other stakeholders include OSPs (OSCAR Service Providers), software engineers/developers, and researchers.

Read more about OSCAR at the following locations:

- http://oscarcanada.org/
- https://oscar-emr.com/
- https://en.wikipedia.org/wiki/OSCAR_McMaster

# High Level Architecture

At a high level, all of OSCAR follows a standard Client-Server architecture. Clients consist of a web browsers on user devices, often located within clinics. Most OSCAR clinics have a single server (physical or virtual) that services the clients within the clinic. The server consists of a Java EE application running within Apache Tomcat and uses a MySQL database to persist data. The following UML deployment diagram describes the high level architecture of a typical OSCAR deployment:




Other more complex deployments may exist in form of cloud deployments offered by OSPs.

## Frameworks

A number of frameworks are used to support the OSCAR web application. This section gives a brief overview of the frameworks and provides links to useful resources.

### JAX-RS and Apache CXF

The Java (X) API for Restful web Services (JAX-RS) provides an API for creating RESTful web services. A number of implementations exist for this API; OSCAR uses Apache's CXF implementation. JAX-RS allows developers to use annotations in the source code to indicate the relationship between HTTP request paths and request handler functions.

See the `oscar/src/main/resources/ApplicationContextREST.xml` (~ line 66 when this was written) for configuration of the REST interface and handlers.

### Spring Dependancy Injection

The Spring Framework provides numerous utilities to Java EE applications, one of which is Dependancy Injection (DI). DI allows for Inversion of Control which abstracts away runtime details and encourages high levels of abstraction. In OSCAR Spring DI provides runtime creation of objects without requiring developers to instantiate the objects themselves.

Spring DI is used extensively in the OSCAR code base; developers should have a working knowledge of the framework. Some useful resources for learning or refreshing your memory:

- Spring Framework
- Spring Inversion of Control
- Spring DI Tutorial
- Spring DI Tutorial Videos

### Java Persistence

OSCAR uses the Java Persistence API (JPA) library for interacting with the MySQL database.

Useful links:

- Java 6 Persistence JavaDoc
- Official JPA Docs
- JPA Tutorial

### AngularJS

Angular JS is a front-end JavaScript framework that uses the Model-View-Controller design pattern. OSCAR 14 (15) introduced this framework to support the new "look and feel" for the user interface.

Find source code related to AngularJS in: `oscar/src/main/webapp/web/` .

A number of tutorials exist for Angular JS, though the official tutorial is quite good.

### Bootstrap

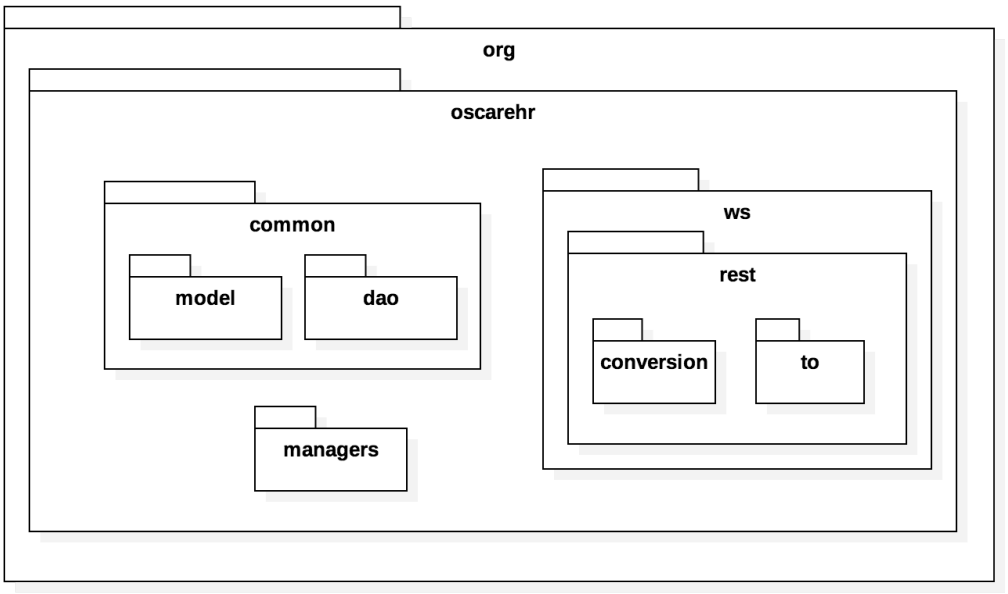
Bootstrap is a CSS library that provides responsive layouts and a clean look and feel. This library was introduced in OSCAR 14 (15). The documentation for Bootstrap is excellent.

# Restful Web Service

OSCAR 14 (15) introduced a RESTful web service. This section describes *some* of the code level architecture for the RESTful web service implementation. The A general architecture is described and the Tickler and Medication Summary features are used as examples.

## Package Structure

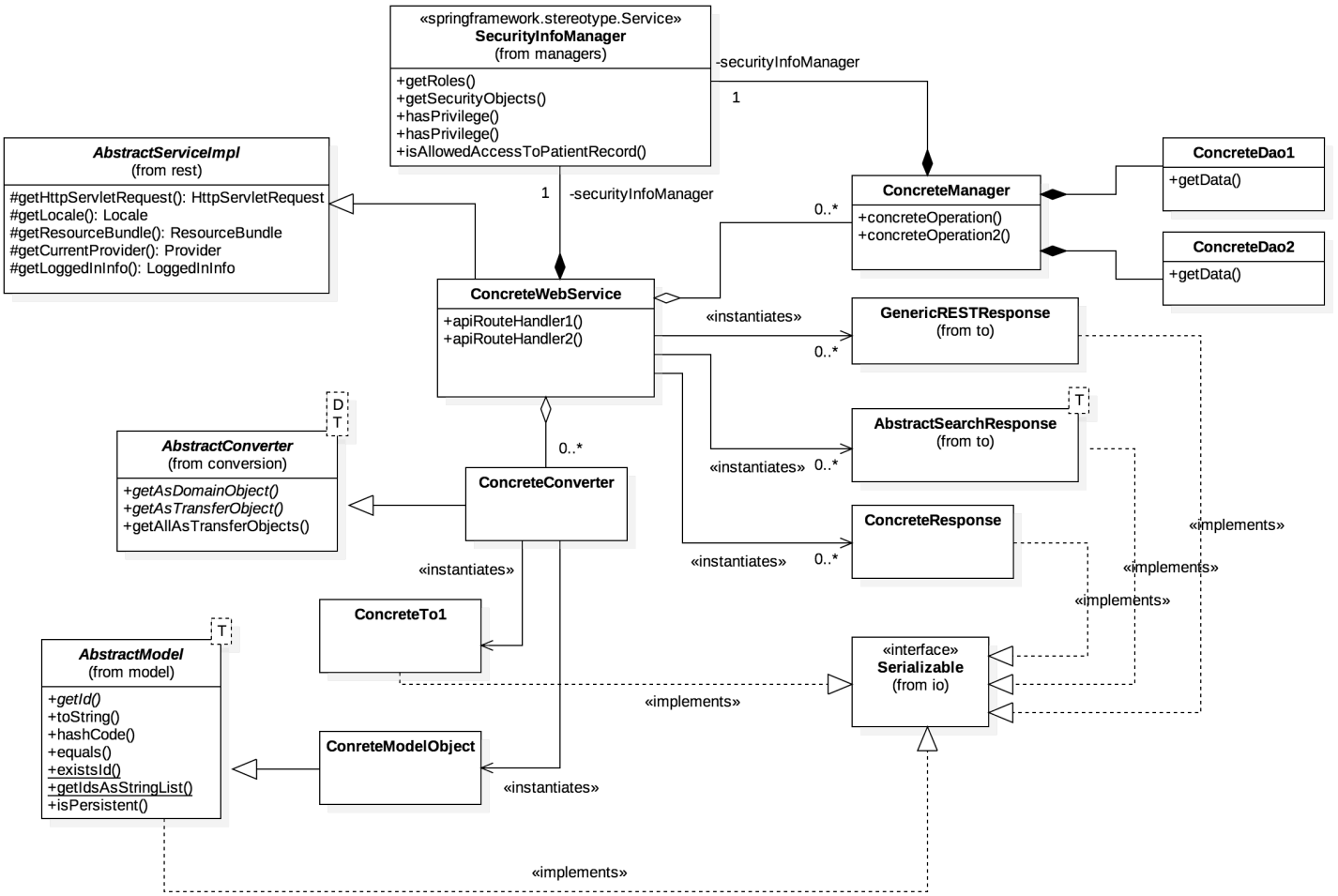
The majority of classes that implement or support the RESTful web service can be found within the `org.oscarehr` package. The following package diagram shows a number of packages which may be referenced in subsequent diagrams.




## Service Implementation

RESTful web services bind HTTP(S) paths (e.g. `http://hello.org/a/rest/path` to a particular route handler function within a specific handler/service class. OSCAR uses Apache's CXF implementation of the JAX-RS API to map HTTP requests to paths to route handler functions.

In OSCAR, the handler function for a particular path is contained within a service class that handles related requests.The following UML class diagram describes a general service class and associated classes, important classes and relationships are described below.




*Note: To save space, the proceeding class diagram has intentionally omitted some class attributes and methods.*

## Web Services

The *ConcreteWebService* class is a prototypical web service class that contains related request handlers. All web service classes extend the *AbsractServiceImpl* class. Request handlers take parameters that match the HTTP request parameters and return a response object that implements the *Serializable* interface. Response objects include:

- **GenericRESTResponse:** Used for returning non-specific information. For example a 200-OK status code.
- **AbstractSearchResponse:** An abstract response type that can be specialized to describe the results of a search on the database (e.g. "search for all Tickers for Dr. OscarDoc").

- **ConcreteResponse:** Used for describing domain/application specific information. For example, a *DrugResponse* might describe a single instance of a Drug object that is prescribed to a patient.

### Security Manager

The *SecurityInfoManager* provides functionality for checking user permissions and privileges. This class is usually instantiated by Spring DI at run time. Route handler functions should use this class to determine whether incoming requests have permission to view/modify data.

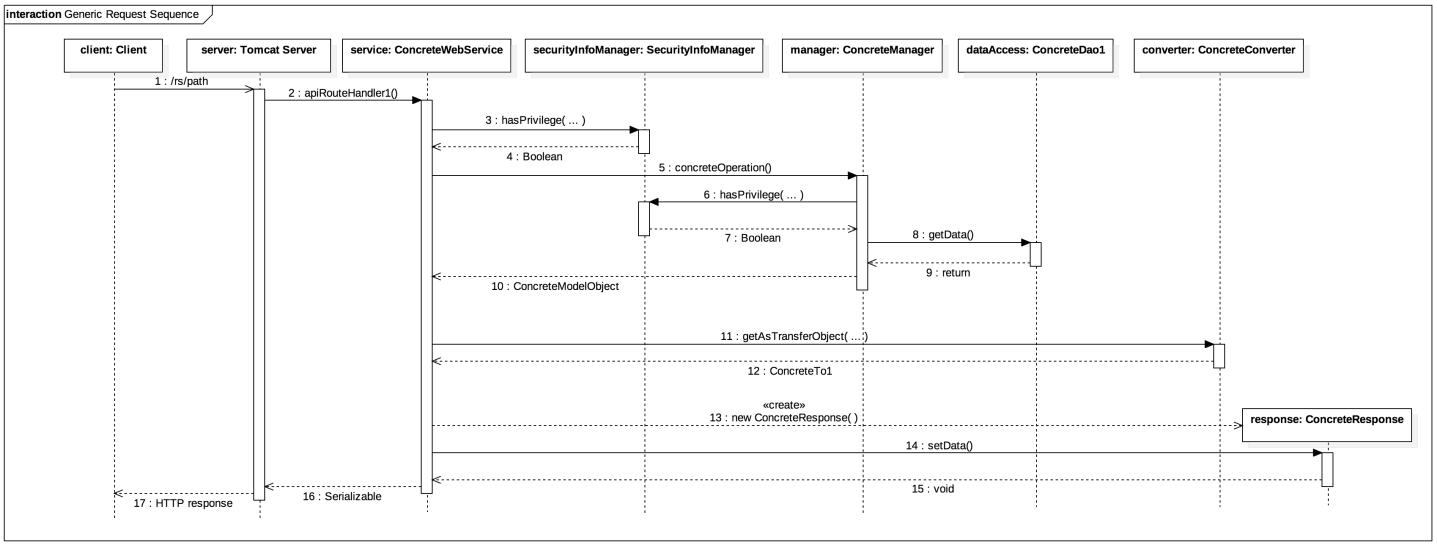
### Managers

The *ConcreteManager* provides access to relevant data and business logic classes that are required by the *ConcreteWebService*'s route handlers. A given web service class may use several manager classes to access the required data/computations. Managers use DAOs to interact with the database and may also use the *SecurityInfoManager* to control access to data.

### Converters

The *ConcreteConverter*, which extends *AbstractConverter*, provides a means of converting between model objects, like a *ConcreteModelObject*, to a transfer object, like *ConcreteTo1*. Transfer objects implement the *Serializable* interface and can be wrapped by a Response object to be sent back to the client. A web service may have several different converters to transform the various model objects it deals with.

The general interactions between these components are given in the following sequence diagram:




# Misc Notes and Resources

Notes and resources that were useful while investigating the architecture.

- Uses JAX-RS (Java Restful Web Services)
  - https://en.wikipedia.org/wiki/Java_API_for_RESTful_Web_Services.
  - JAX-RS is a specification for an API as well as a library of code.
  - Apache CXF is a library that implements the API and pairs well with the Struts framework.
  - Allows annotation of methods to indicate the role they play in a REST API.

- http://cxf.apache.org/docs/jaxrs-services-configuration.html
- Really good example here:
  - http://www.dreamsyssoft.com/blog/blog.php?/archives/7-Simple-REST-Web-Service-in-Java-with-Spring-and-CXF.html

- http://www.uml-diagrams.org/deployment-diagrams-overview.html