Unified Web Services Model for UMBC DoIT

Introduction

REX takes in data across UMBC and stores it, and does so extremely effectively. For end-users, it is convenient to access data through the reporting interfaces. However, the server-server interaction model has not been fully developed, leading to ad-hoc interfacing requests. There is a need for a consistent, comprehensive interface for accessing stored data about students in server-side applications for use cases such as mitigating errors in user-supplied bio/demo data in DocuSign forms, myUMBC portal access, and other external integrations including Civitas (College Scheduler).

Scope

This model aims to describe an interaction method targeted at server-side applications such as myUMBC. It should provide consistent interface for accessing and aggregating several data sources including both REX and PeopleSoft. This project will not

Example Scenario

As-Is

Suppose that Financial Aid has a form that requires a student (the user) to input their name, address, and income from on-campus jobs. In the current interaction model, the static form would be provided by DocuSign, which would display the form to the user who would then manually enter this information. Financial Aid would only be able to catch errors in the data after the form was submitted by manually checking against the various systems that contain the data that the user has now duplicated.

Proposed

The same form as above is used in this example. However, instead of being statically generated, the DocuSign server, as authenticated by an OAuth or Shibboleth service account, and authorized by that account's permissions in LDAP, calls the UWS API Bus on behalf of the user to dynamically fill the form with verified information, using a REST method (e.g. `GET https://uws.umbc.edu/user/{UID}/address`). The UWS then behind the scenes securely executes SQL in REX to provide the most up-to-date version of this data available without the risk of the user mistyping and requiring the form to be resubmitted.

Current Revision

Description 6/28/17

A prototype of this system is currently deployed as a PSGI application on umbc.in for development purposes. The codebase is currently approximately 800 lines of Perl, with an additional 400 lines of views. Seven endpoints have been deployed allowing proof-of-concept access to REX, introspection on schemas, and access to web services. A web service can be deployed end to end in the web browser. Based on testing on a dual core VM with 4GB RAM, the service is capable of serving 3000 queries per minutes, and is limited by CPU available on its current hosting. No authentication is currently used on the API endpoints, but the configuration interface is behind basic authentication.

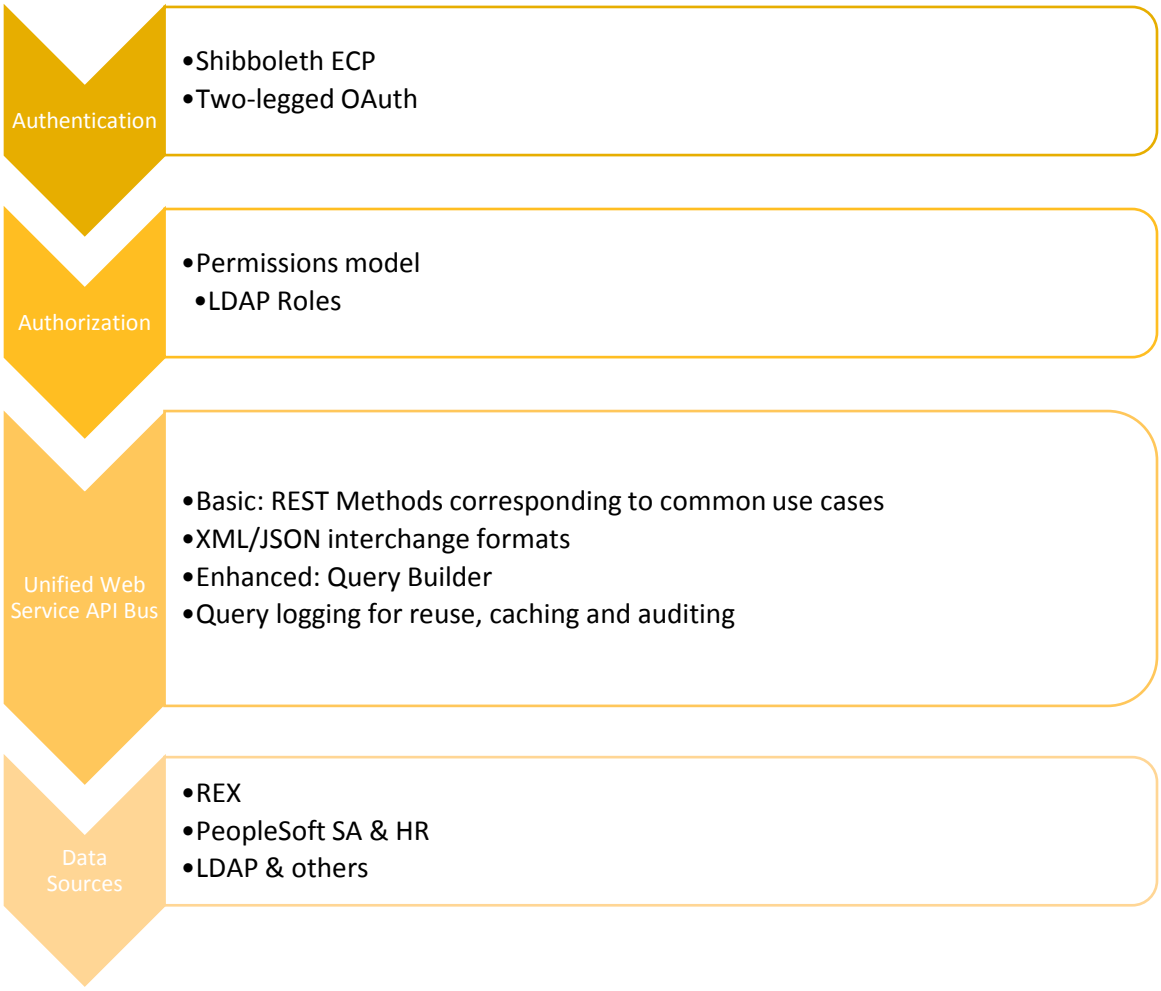

Figure 1: The Unified Web Services model

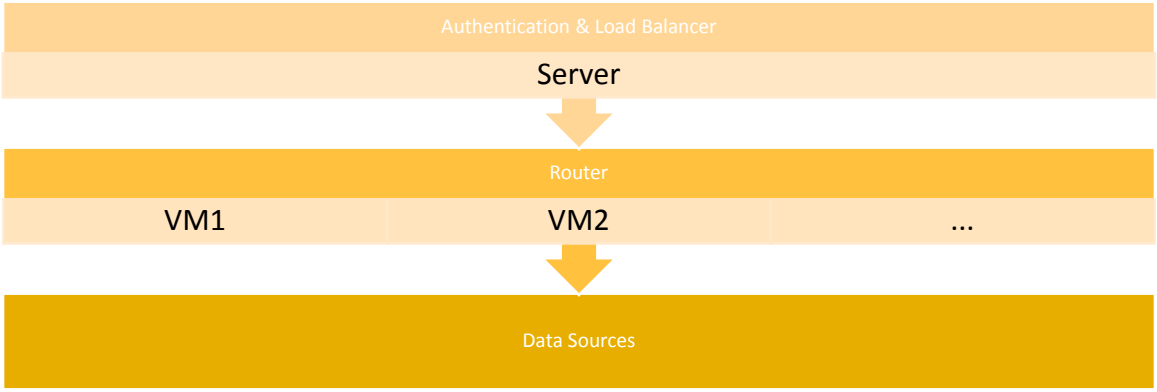

Figure 2: Unified Web Services Server Architecture

Infrastructure	Server	Applications
Load Balancer	**Linux (CentOS), 4GB RAM**	**Nginx, nginx-http-shibboleth, nginx-google-oauth, and Lua**
Router	**Linux (CentOS), 8GB RAM, 5GB disk**	**Perl, Dancer2, and Starman or Java, Java Spark (not Apache Spark), and JDBC**

Database	**Can be on the same server as the router for testing**	**Sqlite3, PostgreSQL, or MySQL**

Figure 3: Infrastructure