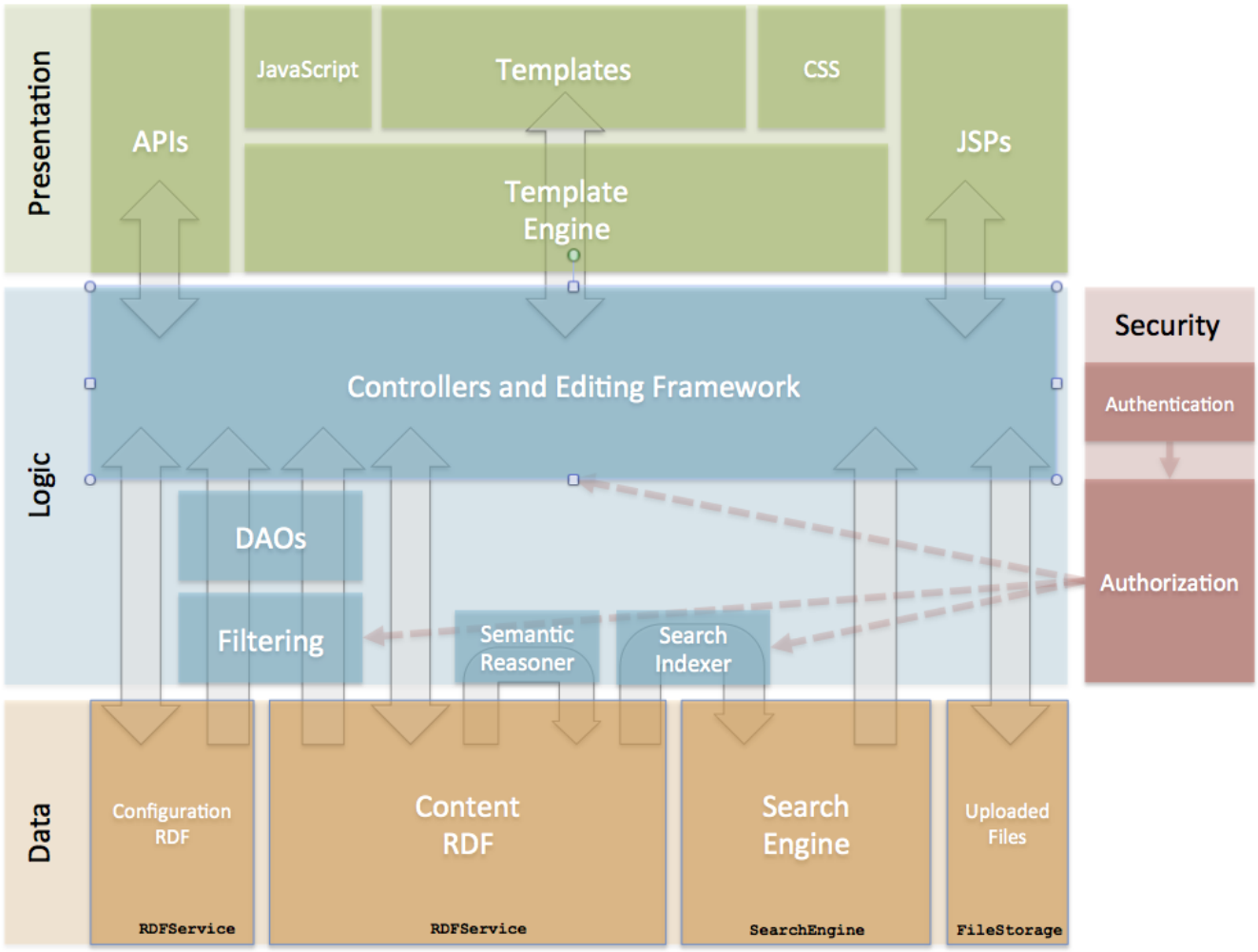
Software Architecture Overview

A diagram of the VIVO software architecture, with notes.



Data

VIVO has four data stores. When copying, backing up, or restoring a VIVO installation, all four data stores should be considered.

Content RDF

This is where most of VIVO's information is stored. Names of individuals, relationships between individuals, types of individuals (for example, Person or Organization), are all stored in the Content RDF

Content RDF uses a triple-store or other SPARQL endpoint. Usually, the triple-store is a Jena SDB implementation, with a MySQL database.

The interface is specified by `RDFService.java`.

Configuration RDF

This is where VIVO's parameters are stored, like which templates are used to display what types of data. Other parameters describe how the custom editing screens are applied to complex data structures. The Configuration RDF is also the storage for VIVO's user accounts.

Configuration RDF uses a triple-store or other SPARQL endpoint. Beginning with release 1.7 of VIVO, the triple-store is a Jena TDB implementation, with files kept in the home directory of the VIVO application. In previous releases, the Configuration RDF used a Jena RDB implementation, sharing the MySQL database with the Content RDF.

The interface is specified by `RDFService.java`.

Search Engine

In theory, all of the search operations in VIVO could be performed using SPARQL queries against the RDF. In practice, however, a dedicated search engine gives a much faster response. The search engine is available to VIVO's users, to assist in finding pages of interest. The search engine is also used internally, to provide prompt response to requests for auto-completion, indexes, counts, and other data.

The search engine permits queries that yield faceted results, for a more successful search. Usually, it is implemented with a Solr web application. By default, Solr is installed in the same web server as VIVO. However, it is easy to move Solr to a different web server, to improve performance.

The interface is specified by `SearchEngine.java`

Uploaded Files

VIVO allows individuals to upload images for their profile pages. VIVO also generates a thumbnail image for more compact display. These images are kept in the Uploaded Files storage. Each file is assigned a URI, so it can be distinguished from other files of the same name. Currently this is only used for images, but VIVO could be customized to store other types of files here as well.

The default implementation uses a storage system similar to PairTree.

The interface is specified by `FileStorage.java`

Logic

VIVO adds a layer of "business logic" to the data storage. It uses inference to add to the data. It applies policies to determine which users are authorized to see which pieces of data.

Controllers and Editing Framework

The controllers contain the top-level logic, determining how to respond to web requests. This includes fetching data, making decisions based on that data, and displaying the results.

The Editing Framework provides the user with the tools needed to edit the RDF data. In some cases, a simple default screen will suffice. For more elaborate data structures, the Editing Framework creates related groups of data objects, and enforces the relationships among them.

DAOs

The DAOs, or Data Access Objects, form a layer of secondary logic. They provide a large number of utility subroutines, to take the repetitive processing tasks away from the Controllers and Editing Framework.

The DAOs also provide a framework for the filtering layer.

There are a large number of interfaces that define the DAO layer.

Filtering

Data within VIVO can be public or private, or shades of gray. The Filtering layer works with the Authentication system to determine which pieces of data may be displayed to a particular user.

The Filtering layer means that the Controllers don't need to include logic for this sort of decision. The Controller asks the Filtered DAOs for data, and receives as much data as the current user is authorized to see.

The interface is specified by `VitroFilters.java`

Semantic Reasoner

One of the principal strengths of RDF is that we can infer additional data from the data at hand. However, the logic involved can be complicated and time-consuming.

Currently VIVO applies two different reasoners to the Content RDF. The TBOX - or Ontology models - are small enough that extensive reasoning can be applied. Currently, the Pellet reasoner is used. Applying that same level of inference to the ABOX - or Assertions models - would take a prohibitive amount of time. VIVO uses its own reasoner for this, applying only those logical inferences that VIVO requires to function.

Search Indexer

The Search Indexer reacts to changes in the Content RDF, updating the search index to reflect those changes. Several types of logic are employed to determine which individuals are affected by the RDF changes, and how to build the search records for those individuals. Sometimes a single change requires that several search records be rebuilt.

Presentation

The presentation layer is where the web pages of VIVO are created. Most of the web pages are created using the Freemarker template engine. However, a number of pages are still created by JSPs.

Template Engine and Templates

VIVO uses the Freemarker Template Engine to construct the HTML for its web pages. The templates describe the format and structure of the pages, and the template engine inserts relevant data each time the template is used.

JavaScript

VIVO relies heavily on JQuery to create a rich and responsive user interface. Other scripts are used also.

CSS

Like any web application, VIVO uses CSS files to produce a consistent style across the user interface.

JSPs

Early releases of VIVO were built almost entirely using JSPs. The change to Freemarker was an attempt to insure better separation between the Logic layer and the Presentation layer.

Some JSPs are still used in VIVO. In general these are restricted to administrative pages, including advanced data manipulation and "back-end editing".

APIs

VIVO supports a collection APIs for importing and exporting data. The APIs have no presentation layer, per se. The format of their responses is determined by the nature of the request, and usually does not involve HTML. Responses to API requests are constructed entirely by the Controllers.

Security

The security system determines what data a user may see, what data they may modify, and what functions they may perform.

Authentication

VIVO includes its own authentication system, including user accounts with email addresses as identifiers and passwords as credentials. VIVO can be configured to use an external authentication system also. In this case, VIVO still maintains a user account for each user, but no passwords are stored. If the external authentication system asserts that a user has properly logged in, VIVO accepts that assertion.

Authorization

The Authorization system relies on a list of Policy objects to determine what a user may do. Before the Controllers or the Editing Framework or the Filtering layer take any action, they consult the Policy list to determine whether that action is authorized for the current user.

This very flexible set of Policies permits VIVO to classify some data as public or private, while other data is private except to the user who owns it.