Connecting the Rails Web Framework to a VIVO Jena Store
Yin Aphinyanaphongs MD PhD, Lawrence Fu PhD
NYU Langone Medical Center, Center for Health Informatics and Bioinformatics, New York, NY

Abstract
We present a proof of concept application that connects a Jena RDF VIVO store to a Rails based frontend. Rails allows rapid prototyping that is typically not available in the VIVO Java development environment. We discuss the design of the system and demonstrate the ease of customization by showing how easily indexing and search is tailored within the application. The code used in this application is available from github.

Introduction
Rails [1] is an open-source web framework that enhances programmer happiness and productivity. The framework supports rapid, agile development of web applications. The framework is tested and used by high profile sites such as GroupOn, Hulu, Twitter, and LivingSocial [2]. In addition, the agile web practices provide advantages in reducing code complexity, incorporating high level abstractions, and allowing quick configuration and building of indices. We proposed to leverage these advantages by connecting Rails to an RDF VIVO store.

In this paper, we report a proof of concept application that connects an RDF VIVO store to Rails. We aim for several goals:
- Demonstrate one method for reading a Jena store with Ruby [3] and deploying a Rails application on top.
- Design with a plug and play architecture which allows any VIVO installation to install and use our application.
- Demonstrate easy search customization using the Ruby domain specific languages.
- Outreach to the community interested in working with VIVO and Rails.

Why Rails?
Our institution developed an internal research networking tool based on Rails. This framework enabled rapid prototyping of algorithms and user interfaces by our team. Because of our success with the framework, we wanted to use our existing codebase while leveraging the work on VIVO.

Design
VIVO on Rails is currently a read-only application. More specifically, Rails converts the Jena RDF VIVO store to a compatible relational database. This design fits with our enterprise data model. This model does not allow users to make changes to their profiles and all updates must be vetted for accuracy.

Connecting VIVO to Rails
Figure 1 shows the two designs to connect VIVO to Rails. 1. Dump the RDF VIVO store to a comparable SQL store and execute Rails on top on this database. 2. Connect Rails directly to the RDF VIVO store. We chose the first option for several reasons. First, Rails favors convention over configuration. For example, table names in rails are plural and the object to interact with the table from Rails is singular. By using SQL, we may use these types of programmer shortcuts. Second, we considered building an Active Model plug-in that would connect directly from the RDF VIVO store to the application. Unfortunately, building an Active Model plug-in would require significant changes to the ontology. Third, by separating the RDF store with a SQL database, we enable an application to fill in the sql database in whatever means possible (i.e. sparql query, sql etc), while maintaining the ability to use the Rails infrastructure.

Converting RDF VIVO to SQL
We converted the RDF VIVO store to SQL by implementing the QUALEG algorithm [4]. The algorithm applies mapping rules to transform constructs of the ontologic model to those of the relational model. For example, a class maps to a table, and a data type property maps to a column. We applied selected rules to specific class and data property mappings in the ontology. Figure 2 illustrates this operation on the Academic Articles class.

Customizing Search
We highlight several examples of customizing search tasks within the Rails framework. Rails uses the Model-View-Controller design pattern [5] adopted by recent VIVO releases. The design pattern encapsulates functionality, and benefits include improved readability, code organization, and re-use of code. The model contains the database, the controller interacts with the database, and the view interacts with the controller. In the Rails application, we use the sunspot ruby gem [6] that communicates with a SOLR index/search server [7]. In Figure 3, we illustrate several code snippets utilizing the model-view-controller design pattern.

Conclusions
We demonstrated a proof of concept application that connects a JENA RDF VIVO store to a Rails application frontend. We discussed our rationale for our design choices. We also showed a search/indexing example that illustrated the ease of customization within the framework. The application is available on github [9].

We also intend to continue development of this application. We encourage any interested parties to contact us and contribute.

References
[8] https://github.com/yin/vivo_on_rails

Acknowledgements
Supported in part by grant ULI TR000038 from the National Center for the Advancement of Translational Science (NCATS), National Institutes of Health. Acknowledgements to the Elman Medical Library and Drs. Constanin Alferis and Bruce Cronestein for their support of this work.

Contact
Yin Aphinyanaphongs, yin@nyumc.org
Larry Fu, lawrence fu@nyumc.org