A REST API for Library Data

Jason Thomale & William Hicks | University of North Texas Libraries, User Interfaces Unit

We Wanted Access to our Catalog Data

We've long had plans for it.

In the UNT Libraries' User Interfaces Unit, we've wanted to experiment with our catalog data to try to build new applications that better support resource discovery. And, with RDA and BIBFRAME presenting models ripe for experimentation, there's no better time than now.

But we're an Innovative Interfaces, Inc. (III) shop.

This means historically we've had no programmatic access to the data in our Integrated Library System (ILS).

So we migrated from Millennium to Sierra.

Sierra is III's new ILS, billed as being "open," in theory. In practice, we do at least now have *some* access to our data, primarily through a read-only SQL interface.

But is SQL access enough?

SQL access can be slow – prohibitively slow for most applications, especially when querying bibliographic data.

What about Sierra's REST API?

III released their REST API for Sierra in April of 2014, but it doesn't meet our needs, either. Why not?

ILS APIs model ILS data.

Using ILS APIs as-is restricts us to ILS concepts and conventions, built for a world that uses MARC. We need the freedom to model and access our data differently based on our particular needs.

And we don't want to be at our vendor's mercy.

Ultimately, we have no control over our vendor's APIs. We can't ensure that they will be as functional as we need.

So We Started Building ...

It's a Python Django project with three primary components: a Base, an Exporter and an API.

The Base contains data structures such as models.

The Exporter performs scheduled data extraction, transformation, and loading from Sierra to Solr and Redis.

The API frontend, which uses the Django REST Framework, serializes data to / from Solr and Redis, handles HTTP requests, and serves resources to clients. For now we're mainly interested in building JavaScript clients atop the API, so we focus on JSON.

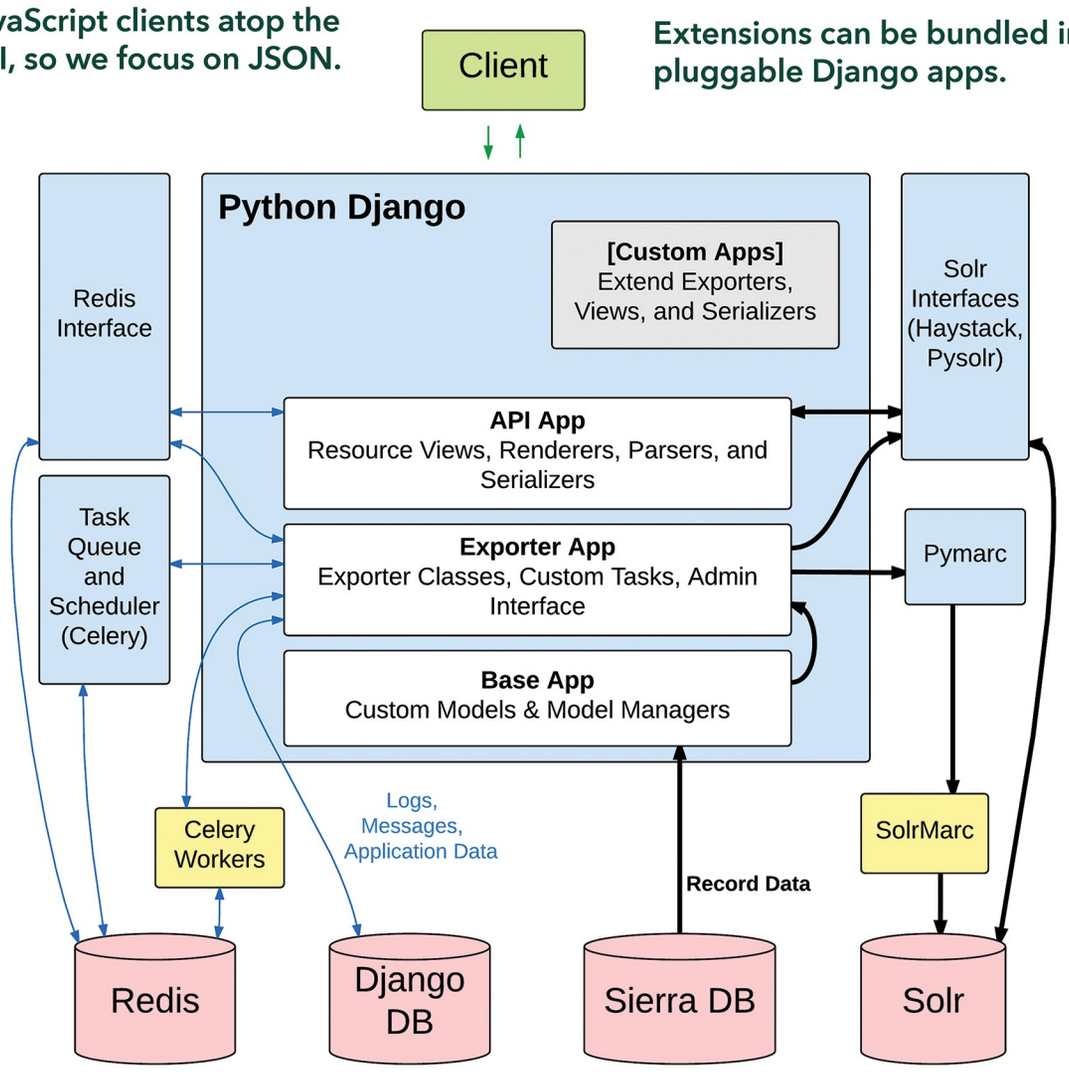
It's also an extensible REST API builder, toolkit, and framework.

Currently we expose only a few resources, but nearly all 344 database views accessible to Sierra customers are modeled using the Django ORM.

Creating a new type of export is as easy as modifying a Solr schema and extending the base Exporter class. To start loading data, we just plug the new class into the scheduler.

We create new resources by extending the Resource View and Serializer classes.

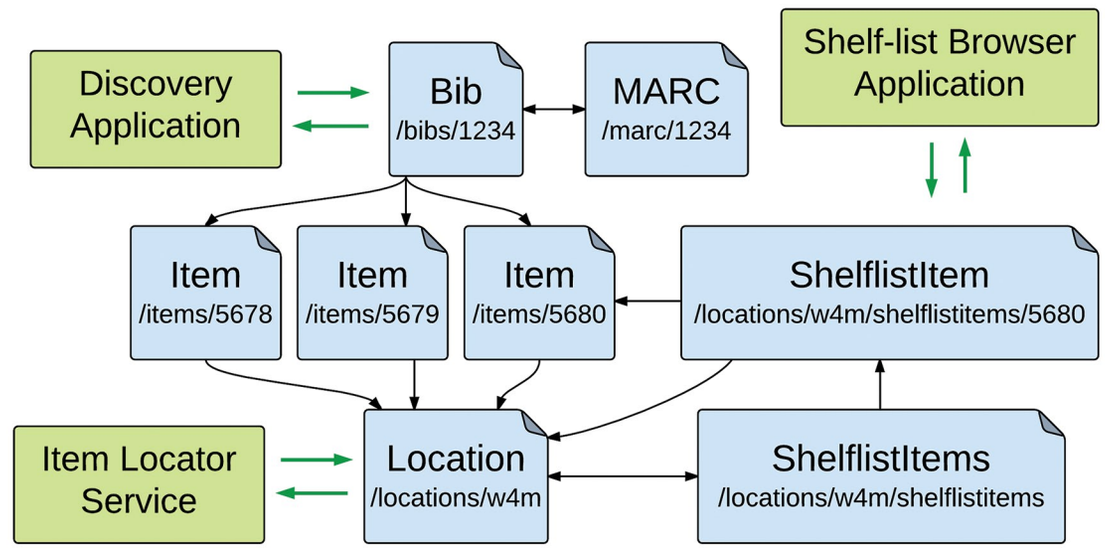
Extensions can be bundled into pluggable Django apps.



... A Web-Based, RESTful API

We like REST because it's the architectural style underlying the Web.

We want to build Web applications that can take advantage of our bibliographic data in a natively Web-based way: as crawlable, interlinked resources.



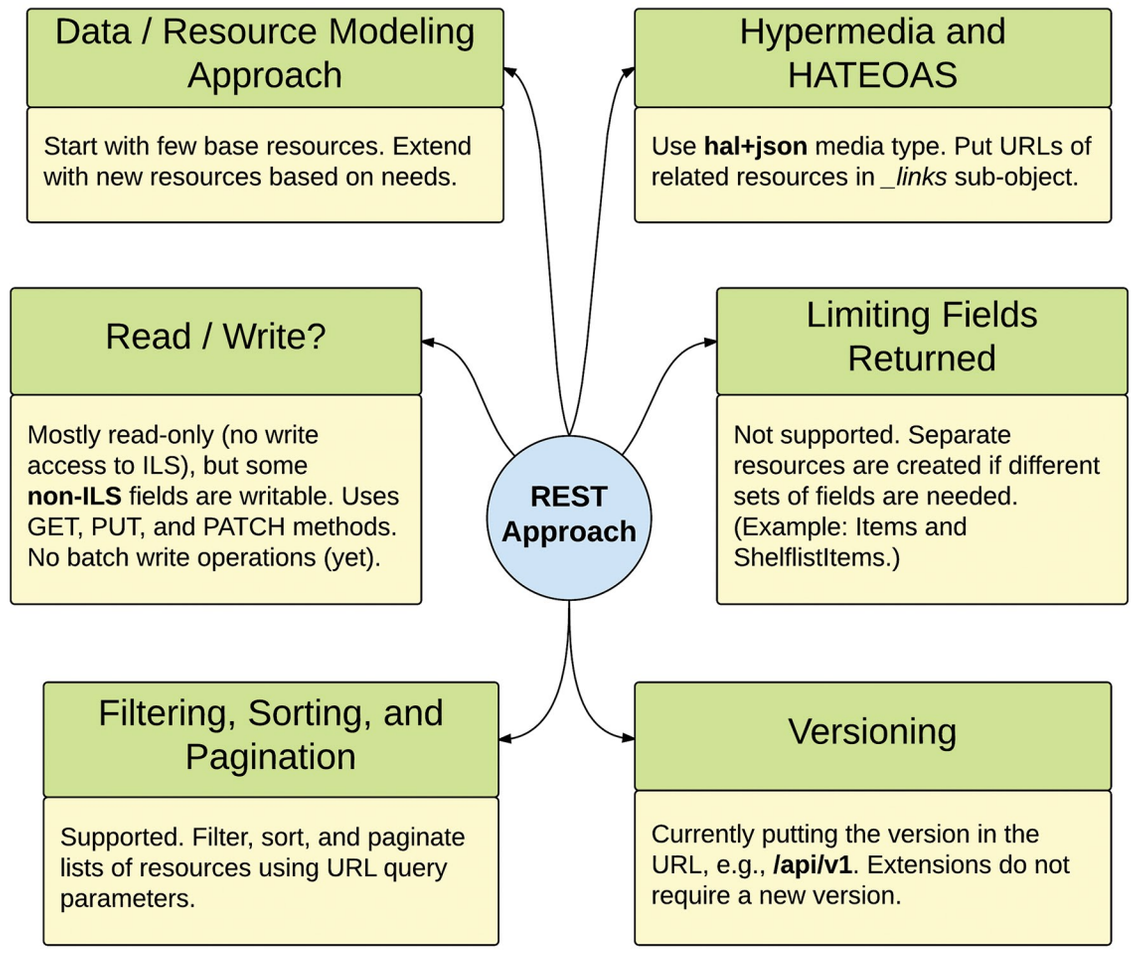
And REST concepts are basically compatible with Linked Data concepts.

Implementing new bibliographic data models using a REST API could serve as a stepping stone toward serving bibliographic Linked Data, as BIBFRAME in JSON-LD or something else, without requiring additional infrastructure or affecting existing API clients.

But – The Devil is in the Details

Beyond the basics such as using nouns for resources and handling errors using HTTP status codes, REST practices vary widely and are sometimes hotly debated.

Here are some details of our approach, some of which may affect how "RESTful" it is, depending on whom you ask.



Are these "best" practices? They're "best" for us insofar as they're working for our needs, for now.

Have You Built a REST API? Let's Talk!

Jason Thomale

Resource Discovery Systems Librarian
jason.thomale@unt.edu