Sondra Documentation

Release 0.1

Jefferson Heard

July 28, 2015

Contents

Sondra is an object-NoSQL persistence layer for RethinkDB and Python based on JSON Schema. Sondra is also a RESTful API generator for persistent objects.

Sondra is meant to be simple to use, quick to get going, and to generate consistent APIs for most use cases.

Sondra is still a very early stage project, but already lets you create data models and APIs from a nearly complete subset of JSON-Schema.

CHAPTER 1

Unique Features

- JSON-Schema based data declaration more closely mirrors the capabilities of document databases, rather than re-using metaphors from ORMs like SQLAlchemy.
- An `@exposable` decorator that allows you to take advantage of PEP3107 function annotations to expose methods at the application, collection, or document level as API methods.
- A consistent scheme for automatically generated RESTful APIs.
- Objects in Sondra simulate Python's mapping types to make for as

CHAPTER 2

Sondra's Data and API Scheme

`Document` s are individual documents stored in a RethinkDB table. `DocumentCollection` s are the tables that contain these collections. `Application` s group `DocumentCollection` s logically. This creates URLs for your API that follow this scheme:

`http://t.io/app.app_method/coll.coll_method/docid.doc_method;fmt/#/frag/ment`

Call Type	Url	Valid Methods
Application[1]_ Application method	http://t.io/app	POST, GET

Collection CRUD

http://t.io/app.method

http://t.io/app/coll

POST, GET
POST, GET, DELETE, PUT

Collection method http://t.io/app/coll.method POST, GET

Document CRUD http://t.io/app/coll/1 POST, GET, DELETE, PUT

Document method http://t.io/app/coll/1.method POST, GET

Each of the above call types can retrieve several different responses depending on the format requested. The format is added as a URL parameter (not a query parameter) to the last element in the path.

CHAPTER 3

Valid Formats

(* **Note** that calling the application by itself requires the format to be either *help* or *schema*)

- help - Gets HTML help for the call. Help is derived from the docstrings of the item that was requested. Docstrings are assumed to be plain reStructuredText.
- schema - Gets a JSON schema for the call. For methods, this returns two schemas in two keys: *returns* and *arguments* which describe the response schema and the request schema respectively.
- json (default) - Gets a JSON string response. This is the default format
- jsonp - Gets a JSON-P response.
- html - Gets a formatted HTML response. This typically formats the JSON crudely, but it can be useful for debug purposes.

CHAPTER 4

Fragments

Fragments can be specified on the URL either with the standard fragment character, # or an alternate fragment character, @. Browsers don't typically pass fragments, so to work around this limitation, we define an alternate character. Either will work assuming the server gets the full URL.

Fragments pre-filter the response to a sub-document. Since all responses are basically JSON, this allows us to traverse down the JSON and pick out what we want.

If a fragment value is passed to the API when the response would be an array (such as a filtered subset of a collection), the fragment is applied to every element in the array.

If a fragment value is passed to the API when the response would be a single object, the fragment will be applied to that object. If this results in a single value, the response will be made into valid JSON by using a dummy object:

```
{ "_": "<value>" }
```

See the assorted tests for examples of usage.

Contents:

4.1 sondra

4.1.1 sondra package

Subpackages

sondra.tests package

Submodules

sondra.tests.settings_test_db module

sondra.tests.settings_test_loader module

sondra.tests.settings_test_models module

sondra.tests.test_db module

sondra.tests.test_loader module

sondra.tests.test_models module

sondra.tests.test_web module

sondra.tests.utils module

Module contents

Submodules

sondra.api module

Sondra's JSON API Services.

API services are provided as a part of Sondra. These will get more complete over time. In the meantime, here is the general combined URL format for calls on a collection:

```
http://host/api/appname.appmethod/collection.classmethod/0000.instancemethod/@/frag/ment;format
                1------ 2-------- 3--------- 4---------- 5--- 6------------ 7---------- 8-----
```

1. **appname**. The application name, in this case "pronto"
2. **appmethod**. A ‘‘@expose‘‘d method that exists on the application object itself.
3. **collection**. The collection name. If the collection is in subgroups, the subgroups are path-separated as well
4. **classmethod**. Any exposed class methods on the
5. **oid**. The object id. This is a UUID-based object, like RethinkDB usually uses.
6. **instancemethod**. A method name that was ‘‘@expose‘‘d on the model.
7. **format**. `(json|jsonp|html|help|schema)` The format we expect it in. `json` is generally the default.
8. **fragment**. The fragment portion of the URL digs deeper into the properties of the object, referring to a JSON fragment within the referred document. Numbers and Python slice syntax work fine to get at pieces of an array within the document.

Furthermore there's a grammar for combining these, and a list of allowed HTTP methods for each grammar production

Application Metadata Form:

```
http://localhost:5000/appname/api;format
```

By format:

- `help`: Get the docstring for the application object
- `schema`: Get the combined JSON schemas for all collections for the application

Application Method Call Form:

```
http://localhost:5000/appname/api.methodname;format
```

All Applications have the following methods:

- **listing** - get the listing of all APIs and methods on this application

`GET` and `POST` have the same meaning. All other methods are unsupported on the application.

GET Behavior By format:

- `help`: Get the docstring of the method.
- `schema`: Get the request and response schema in JSON Schema format.
- `html`: Get a string response to the method.
- `json`: Get a JSON response to the method.
- ‘‘jsonp‘: Get a JSON-P response to the method.

GET expects you to call the method with arguments passed as query parameters. For more complicated method calls, use POST

POST Behavior POST always calls the method. POST can either be called with urlencoded or multipart form data, or with a JSON string. All of these will be interpreted the same way, as method calls. Only multipart supports FILE posts, and the files posted must be according to the method parameters.

By format:

Formats `form, help, schema` are unsupported.

- `json`: Expect a JSON response.
- `jsonp`: Expect a JSON response.
- `html`: Expect a JSON response.

Listing **To be documented**

Collection Metadata Form:

```
http://localhost:5000/appname/api/collection;format
```

By format:

- `help`: Get the docstring for the collection object
- `schema`: Get the JSON Schema for this collection.

Class Method Call Form:

```
http://localhost:5000/appname/api/collection.classmethod
```

All collections have the following methods:

- **listing** - get the listing of all APIs and methods on this class
- **filter** - query the collection underlying the API

Listing To be documented

Filter To be documented

Instance Method Call Form:

```
http://localhost:5000/appname/api/collection/uuid.instancemethod;format
```

Instance methods follow the same pattern as application and collection method calls. There are no default instance methods.

Object Create, List Form:

```
http://localhsot:5000/appname/api/collection;format
```

`GET` Retrieve a listing of objects Retrieve a listing of objects in the given format, or retrieve an HTML form for entering a new object.

Query parameters accepted:

- **offset**: Retrieve objects, starting with **offset**-th object
- **count**: The number of objects to return.

`POST` Add/Update an object Accepts JSON in the post body. The object being POST-ed is considered to be a wholesale replacement for the object that already exists. Objects are looked up by their 'id' field, if present. The response will the the object that was added, with 'id' field added if it wasn't already present.

Object Detail, Update, Delete Form:

```
http://localhost:5000/appname/api/collection/oid;format#/fragment
```

`GET` Behavior Retrieves the object specified by `oid` and optionally `fragment`. Fragment is a slash- separated path to a particular attribute on the requested object.

Formats:

' `html`: Get an HTML fragment rendered by the object's `__str__(self)` method. - `json`: Get the JSON corresponding to the object - `jsonp`: Get the JSON corresopnding to the object as a JSON-P string

`POST` Behavior Accepts JSON in the request body. The object being POST-ed is treated as a wholesale replacement for the object that already exists. Objects are looked up by their 'id' field, if present. The response will the the object that was added, with 'id' field added if it wasn't already present.

The response is the object (fragment).

Formats:

- `html`: Get an HTML fragment rendered by the object's `__str__(self)` method.
- `json`: Get the JSON corresponding to the object
- `jsonp`: Get the JSON corresopnding to the object as a JSON-P string

`PUT` and `PATCH` Behavior Accepts JSON in the request body. The object of a PUT or PATCH request body is considered only to be updates on the object. All original object attributes that aren't specified are kept. The assignment is also relative to the `fragment` section of the URL, so the replacement / update is of the object located at that path. If the fragment you PUT or PATCH to refers to an array, the object in the body is appended to that array. If you wish to replace the array, use a POST request.

The response is the updated object fragment.

Formats:

- `html`: Get an HTML fragment rendered by the object's `__str__(self)` method.
- `json`: Get the JSON corresponding to the object
- `jsonp`: Get the JSON corresopnding to the object as a JSON-P string

A Note on Query strings The query string can be used to pass arguments to methods defined on the collection that were decorated with @exposable.

class `sondra.api.`**`APIRequest`**(*sondra*, *headers*, *body*, *method*, *user*, *path*, *query_params*, *files*)

Bases: `builtins.object`

`DEFAULT_FORMAT = 'json'`

`add_collection_items`()

`application_method`()

`application_method_schema`()

`application_schema`()

`collection_method`()

`collection_method_schema`()

`collection_schema`()

`delete_collection_items`()

`delete_document`()

`delete_subdocument`()

`document_method`()

`document_method_schema`()

`document_schema`()

`formats = {'form', 'help', 'html', 'json', 'schema'}`

classmethod `from_django_request`(*sondra*, *request*)

classmethod `from_flask_request`(*sondra*, *request*)

`get_arguments`()

Turn body and query string into a JSON-serializable data structure

`get_collection_items`()

`get_document`()

`get_subdocument`()

`help`()

Return HTML help from a docstring

json_response(*method*)

jsonp_response(*method*)

replace_subdocument()

set_document()

subdocument_method()

subdocument_method_schema()

subdocument_schema()

update_collection_items()

update_document()

update_or_append_subdocument()

exception sondra.api.**ValidationError**
Bases: builtins.Exception

sondra.auth module

Sondra authorization primitives.

exception sondra.auth.**AccessDenied**(*kind*, *user*, *url*)
Bases: builtins.Exception

Raised when access is denied

class sondra.auth.**Authentication**
Bases: builtins.object

class sondra.auth.**Authorization**
Bases: builtins.object

auth_add(*user*, *obj*)

auth_delete(*user*, *obj*)

auth_execute(*user*, *obj*, *method*, *methodname*)

auth_read(*user*, *obj*)

auth_update(*user*, *obj*)

can_add(*user*, *obj*)

can_delete(*user*, *obj*)

can_execute(*user*, *obj*, *method*, *methodname*)

can_read(*user*, *obj*)

can_update(*user*, *obj*)

sondra.db module

Sondra DocumentStores top-level object.

This module defines a helper class that creates connections to document store databases like RethinkDB.

exception sondra.db.**DocumentStoreException**
Bases: builtins.Exception

Connection to the document store failed or the document store is not configured

class sondra.db.**DocumentStores**(*settings*, *attr='DOCUMENT_STORES'*)
Bases: builtins.object

Supports connections to one of many document stores configured in‘‘settings.py‘‘:

```
DOCUMENT_STORES = {
    "kind": "rethinkdb",
    "async": False,  # or 'tornado',
    "stores": {
        "<db_alias>": {
            "host": "localhost",
            "port": 28015,
            "db": "<db name>",
            "auth_key": "<auth key or empty string",
            "timeout": 20
        }
    }
}

# connect to "test" on localhost

import settings

stores = DocumentStores(settings)
stores['test']

# connect to
```

class Kinds
Bases: enum.Enum

DocumentStores.**__getitem__**(*key*)
Get a document store based on a key or a key, db pair.

Parameters **key** (*str or tuple*) – The name of the store to use, or the name of the store to use plus a database name within that store.

Returns A raw document store connection, by default RethinkDB.

Return type Connection

DocumentStores.**kind = None**
DocumentStore.Kinds – The document store type

DocumentStores.**stores = None**
dict – The configuration of stores

DocumentStores.**threadlocal = None**
threadlocal – The connection objects, thread-locally

sondra.env module

Creates the base Sondra application. This prevents us from having to make assumptions about the global environment the way Django does.

class sondra.env.**Sondra**(*settings*)

Bases: collections.abc.Mapping

The Sondra working environment.

This is generally the first thing you initialize in a Sondra application. The Sondra working environment encapsulates some basic environment settings and a mapping of all the applications.

The Sondra object is a mapping object where the keys are application names and values are applications. Applications in Sondra are groupings of related document collections and their methods.

__contains__(*key*)

Determine if an applicaiton has been registered with this instance oc Sondra.

Parameters **key** (*str*) – The name of an Application object.

__getitem__(*key*)

Get the application

Parameters **key** (*str*) – The name of an Application object.

applications = None

dict – A set of collections by name.

base_url = None

str – The base URL for this application.

base_url_netloc = None

str – host

base_url_path = None

str – offset path for all applications. Not supported currently.

base_url_scheme = None

str – http or https

docstring_processor = None

callable – A function for processing docstrings for help API calls

loader = None

JSONLoader – Resolve JSON urls, taking into account settings.ALIASES

ready()

Call after registering all document collections. This sets up the linkages between different collections so that documents are stored in the right collections and joined at retrieval time.

reference(*url=None*, ***kwargs*)

Treat the url as a reference to an application, document collection, document, or sub- document or method call.

Parameters **url** (*str*) – A URL

Returns A sondra.web.Reference

register_application(*application*)

Register an application with the Sondra instance.

Subsequent calls to sondra_instance[name] will return the collection. Name can be specified in the class definition, but if not, it defaults to the classname, converted from camel case to python's standard underscore case.

Parameters **application** (*class[Application]*) – An application subclass.

registered_schemas = None

JSONLoader – Resolve schemas from registered collections

resolve(*url*)
Load a JSON (fragment) from a url.

settings = None
module – The settings module

stores = None
DocumentStores – The document stores singleton.

exception sondra.env.**SondraException**
Bases: builtins.Exception

Exception master class for exceptions in Sondra

sondra.env.**convert_camelcase**(*name*)

sondra.env.**resolve_settings**(*settings*)
import the settings module if we need to

sondra.loader module

class sondra.loader.**AliasResolver**(*default_bindings={}*, *memoize=False*, *search_paths=()*, *application_url='http://localhost:5000'*, *aliases=None*, *my_url=''*)
Bases: collections.abc.Mapping

Resolves aliases from settings into file-like objects.

This is a highly sophisticated rewrite-rule based URL / file resolver. Calling AliasResolver.resolve(url):

1. **Rewrites the url if the url is a relative path, or if it's relative to the application base** path. Paths beginning with a "@" are relative to the application base path. Paths beginning with a "." are relative to the my_url argument passed to __init__.
2. **Heirarchically matches along each alias tree. If a match is made, the URL is rewritten** according to the rewrite rule.
3. **If the new, rewritten URL scheme is http://, ftp://, or https://, then the document is** requested and parsed as JSON.
4. **If the new, rewritten URL scheme is file, then the document is opened from the absolute** path.
5. **Otherwise, the URL is assumed to be a file path relative to the search path. All paths in** settings.RESOLVER_SEARCH_PATHS are tested to see if they contain the file. The first matched file is opened.

Aliases are represented as a tree made up of tuples or lists of objects. Items enclosed in ?(...) are optional. Each level of a tree "consumes" its matching input and matches at the beginning of the string. This means, in practice, that the top-level rule could match on the base url, and paths could be specified as sub-rules.

Component matchers first parse their input with urllib.parse(). Then each component of the URL is matched separately. All components that are specified must match. Component matchers can only contain other component matchers. There is no mixing. Nested component matchers also consume their matches. A component matcher is simply a Python dict.

Any named groups are captured and scoped lexically along the rule heirarchy, possibly being masked by inner scopes. These named groups, as well as: scheme, netloc, path, fragment, query, and params can all be used as substitute variables in the final template.

Templates are python format strings, objects, or componentized format strings. If a component template does not specify a component for rewrite, then the original URL's component will carry through.

If the template is an arbitrary object, it is treated as the actual fragment of JSON itself, rather than as something to fetch. Any strings are treated as python format strings, whether they are keys or values, and all bindings from the pattern matching are substituted in.

Bindings are simply string-to-string mappings of names to values that can be used to substitute in for match groups.

```
aliases := rule*
rule := pattern, action, ?(bindings)
pattern := <regex> | component_matcher
component_matcher := {
    ?("scheme": <regex>),
    ?("netloc": <regex>),
    ?("path": <regex>),
    ?("fragment": <regex>),
    ?("query": <regex>),
    ?("params": <regex>)
}
action := <pyformat_str> | component_template | object_template rule*
component_template := {
    ?("scheme": <py3format_str>),
    ?("netloc": <py3format_str>),
    ?("path": <py3format_str>),
    ?("fragment": <py3format_str>),
    ?("query": <py3format_str>),
    ?("params": <py3format_str>)
}
object_template := { "type": "object", "object": ot_body }
ot_body = ot_list | ot_object
ot_list = ot_body*
ot_object = { *(<py3format_str>: ot_object_val) }
ot_object_val = <py3format_str> | <int> | <number> | ot_list
bindings := <dict>
```

An example:

```
# settings.py

BASE_URL = "http://localhost:5000"  # the base URL of the current webapp

ALIASES = [
    ('https://www.terrahub.io/schemas/common.json', {"$url": {
        "netloc": 'localhost:5000',
        "scheme": 'http',
    }}),
    ('http://www.terrahub.io', [
        ('/schemas/', [
            ('ticket.json', "ticket.schema.json"),
            ('money.json', "ticket.schema.json#/definitions/money"),
            ('contractor.json', "file:///Users/jeff/Projects/pronto/schemas/contractor.json"),
            ('(?P<name>.*).json', "@/static/schemas/{name}.json"),
        ])
    ])
]

RESOLVER_SEARCH_PATHS = [
    "/Users/jeff/Projects/pronto/schemas"
]

CACHE_RESOURCES = True  # If resources don't change, then do this.
```

add (*alias*)

clear_cache()

compile_aliases(*input*)

classmethod from_settings(*settings*)

remove_from_cache(*name*)

resolve(*name*, *memoize=None*)

resolve_name(*name*, *fail_if_missing=False*)

class sondra.loader.**JSONLoader**(*default_bindings={}*, *memoize=False*, *search_paths=()*, *application_url='http://localhost:5000'*, *aliases=None*, *my_url=''*)

Bases: sondra.loader.AliasResolver

Resolves aliases from settings into JSON objects.

This is a highly sophisticated rewrite-rule based URL / file resolver. Calling `JSONResolver.resolve(url)`:

1. **Rewrites the url if the url is a relative path, or if it's relative to the application base** path. Paths beginning with a "@" are relative to the application base path. Paths beginning with a "." are relative to the `my_url` argument passed to `__init__`.
2. **Heirarchically matches along each alias tree. If a match is made, the URL is rewritten** according to the rewrite rule.
3. **If the new, rewritten URL scheme is http://, ftp://, or https://, then the document is** requested and parsed as JSON.
4. **If the new, rewritten URL scheme is file, then the document is read from the absolute path** and parsed as JSON.
5. **Otherwise, the URL is assumed to be a file path relative to the search path. All paths in** `settings.RESOLVER_SEARCH_PATHS` are tested to see if they contain the file. The first matched file is opened and parsed as JSON.
6. **Any "fragment" part of the URL (the bit after the #) is treated as a path to a nested** object within the enclosing document. Paths are not dot-separated like in JavaScript, but slash-separated like the UNIX file path.

Aliases are represented as a tree made up of tuples or lists of objects. Items enclosed in `?(...)` are optional. Each level of a tree "consumes" its matching input and matches at the beginning of the string. This means, in practice, that the top-level rule could match on the base url, and paths could be specified as sub-rules.

Component matchers first parse their input with `urllib.parse()`. Then each component of the URL is matched separately. All components that are specified must match. Component matchers can only contain other component matchers. There is no mixing. Nested component matchers also consume their matches. A component matcher is simply a Python dict.

Any named groups are captured and scoped lexically along the rule heirarchy, possibly being masked by inner scopes. These named groups, as well as: scheme, netloc, path, fragment, query, and params can all be used as substitute variables in the final template.

Templates are python format strings, objects, or componentized format strings. If a component template does not specify a component for rewrite, then the original URL's component will carry through.

If the template is an arbitrary object, it is treated as the actual fragment of JSON itself, rather than as something to fetch. Any strings are treated as python format strings, whether they are keys or values, and all bindings from the pattern matching are substituted in.

Bindings are simply string-to-string mappings of names to values that can be used to substitute in for match groups.

```
aliases := rule*
rule := pattern, action, ?(bindings)
```

```
pattern := <regex> | component_matcher
component_matcher := {
    ?("scheme": <regex>),
    ?("netloc": <regex>),
    ?("path": <regex>),
    ?("fragment": <regex>),
    ?("query": <regex>),
    ?("params": <regex>)
}
action := <pyformat_str> | component_template | object_template rule*
component_template := {
    ?("scheme": <py3format_str>),
    ?("netloc": <py3format_str>),
    ?("path": <py3format_str>),
    ?("fragment": <py3format_str>),
    ?("query": <py3format_str>),
    ?("params": <py3format_str>)
}
object_template := { "type": "object", "object": ot_body }
ot_body = ot_list | ot_object
ot_list = ot_body*
ot_object = { *(<py3format_str>: ot_object_val) }
ot_object_val = <py3format_str> | <int> | <number> | ot_list
bindings := <dict>
```

An example:

```
# settings.py

BASE_URL = "http://localhost:5000"  # the base URL of the current webapp

ALIASES = [
    ('https://www.terrahub.io/schemas/common.json', {"$url": {
        "netloc": 'localhost:5000',
        "scheme": 'http',
    }}),
    ('http://www.terrahub.io', [
        ('/schemas/', [
            ('ticket.json', "ticket.schema.json"),
            ('money.json', "ticket.schema.json#/definitions/money"),
            ('contractor.json', "file:///Users/jeff/Projects/pronto/schemas/contractor.json"),
            ('(?P<name>.*).json', "@/static/schemas/{name}.json"),
        ])
    ])
]

RESOLVER_SEARCH_PATHS = [
    "/Users/jeff/Projects/pronto/schemas"
]

CACHE_RESOURCES = True  # If resources don't change, then do this.
```

See `tests/test_loader.py` for more info on how to use this class to resolve data.

resolve(*name*, *memoize=None*)

resolve_fragment(*son*, *fragment*)

class sondra.loader.**ListPattern**(*patterns*)
Bases: builtins.object

class sondra.loader.**RewriteObject**(*object_prototype*)
Bases: builtins.object

Rewrite rule specified by a JSON object

treesub(*bindings*, *input*)

class sondra.loader.**RewriteParts**(*scheme=None*, *netloc=None*, *path=None*, *query=None*, *fragment=None*, *params=None*)
Bases: builtins.object

Rewrite a URL specified by its components

class sondra.loader.**RewriteString**(*template*)
Bases: builtins.object

Rewrite a URL specified by a string

class sondra.loader.**StringPattern**(*pattern*, *action*, *default_bindings=None*)
Bases: builtins.object

Pattern matching rule where the pattern is a string regex

class sondra.loader.**UrlPattern**(*pattern*, *action*, *default_bindings*)
Bases: builtins.object

Pattern matching rule where the pattern is a dict of parsed URL components after urllib.ParseResult.

update_bindings(*bindings*, *match_object*)

sondra.manipulate module

Tools to rename, extract, embed json-schema documents into others

class sondra.manipulate.**Workbench**(*settings='settings'*, *writeback=True*, *loader=None*, **urls*)
Bases: collections.abc.MutableMapping

refactor(*src_schema*, *src_path=None*, *dest_schema=None*, *dest_path=None*)
Move a sub-schema from src_schema to dest_schema, renaming any '$ref's to account for the move.Not thread-safe. Edits the master list in place.

Parameters

- **master_list** (*dict*) – dictionary of schemas, by full URL
- **src_schema** (*unicode*) – url of source in the master list
- **src_path** (*unicode or None*) – path of the source sub-schema within the source schema document
- **dest_schema** (*unicode or None*) – url of destination in the master list
- **dest_path** (*unicode or None*) – path portion of the target within the destination schema document.

Returns The master list

Return type dict

save(*auth=None*, *as_file=False*, *param=None*, **keys*)

save_key(*key*, *auth=None*, *as_file=False*, *param=None*, *callback=None*)

sondra.manipulate.**clean_path**(*path*)
Remove #/ or / from the beginning of a path.

sondra.manipulate.**read_schema**(*master_list*, *file_or_url*, *schema_id=None*)
Read a schema in from a file object or a URL. Edits master_list in place and is not thread- safe.

Parameters

- **master_list** (*dict*) – A dictionary of schemas compiled by URL.
- **file_or_url** (*unicode or file-like object*) – A file-like object or URL to a remote file containing the schema.
- **schema_id** (*unicode*) – Optionally replace the "id" attribute

Returns the master list, updated.

Return type dict

sondra.models module

Sondra document object model.

This module is similar to models.py in Django and defines a way of programmatically accessing the contents of a document database based on a JSON Schema. Programmers create classes that correspond to document collections. These can relate to each other and link to each other to create deep, heirarchical structures quickly.

A base DocumentCollection class is defined, and links are described in the class body, similar to Django. These links can be accessed on Document objects, which are wrapped dictionaries that follow links and provide lazy lookup and JSON serialization.

In addition a decorator is introduced that makes methods exposable as API calls, and a pre-processor is declared that takes function annotations in methods and constructs API calls baseed on these annotations.

I'm prototyping everyhting in this module first, as that'll help me understand what I'm doing better.

class sondra.models.**Application**(*sondra*)
Bases: collections.abc.Mapping

A group of document collections and a set of methods that can be called on the application instance as a whole.

get_endpoint()

get_name()

get_url()

help()
Return the help associated with this application (docstring)

name = None
Human readable name for this application. If this isn't defined, then we define it automatically

post_registration()

reference

register_collection(*coll*, *endpoint=None*)
Register a document collection with this application

schema

class sondra.models.**ArrayFragment**(*parent*, *seq*)
Bases: collections.abc.MutableSequence

A fragment that holds a JSON array. When a Document contains an array, this proxy is returned by `__getitem__`.

insert(*i*, *x*)

save(*validate=True*, *conflict='error'*, *durability='hard'*, *return_changes=False*)

class sondra.models.**Document**(*collection*, *document=None*)
Bases: collections.abc.MutableMapping

A RethinkDB document conforming to a schema.

__mod__(*value*)
Set the whole document, but keep the same id

application

delete()

get_url()

id

json(*dereference_method='dict'*, *indent=None*, *file_obj=None*)
Return the serialization as a JSON string.

py
Return the full serialization as a dict object

reference

save(*validate=True*, *conflict='error'*, *durability='hard'*, *return_changes=False*)
Save this document into the collection.

Parameters

- **validate** (*bool*) – Default True.
- **conflict** (*'error' or 'replace'*) – Default error.
- **durability** (*'hard' or 'soft'*) – Default hard.
- **return_changes** (*bool*) – Default False.

Returns self, with the newly saved `id` attribute set.

url

validate()

class sondra.models.**DocumentCollection**(*sondra_env*)
Bases: collections.abc.MutableMapping

A document collection.

Everything here is developed with RethinkDB in mind. Other document databases could be supported, logically, but for my first pass I'm not going to generalize too early.

Eventually, we're going to have a metaclass that constructs the subclass of this and sets some special attributes up at construction time. For instance, the schema attribute needs to substitute any reference types to be of type "string" with a URL pattern, if those reference types appear in the "links" mapping.

The following declares two collections, Authors and Books. Books refers to Authors and is lazy about its access of those authors. Our Document class handles retrieving the attribute in the case of the attribute being a link to another document:
class Authors(DocumentCollection): document_store = 'library' collection = 'authors' schema = 'http://localhost:8888/schemas/authors'

class Books(DocumentCollection): document_store = 'library' collection = 'books' schema = 'http://localhost:8888/schemas/books' id = 'ISBN'

links = { '#/properties/author': Authors

}
s = Sondra()

get all books whose genre tags include 'science fiction' >>> sf_books = s.books.q(
s.books.table.filter(lambda book: book['title'].contains('science fiction')))

```
>>> sf_books
<generator object <genexpr> at 0x100000>
```

# get Wintersmith by its ISBN, which we have used here as its id >>> wintersmith = s.books['9781407042527'].py >>> wintersmith
{ "title": "Wintersmith", "author": {

"id": "1234919028234", "first_name": "Terry", "last_name": "Pratchett"

}
}

__getitem__(*key*)
Returns the document whose id is `key`

add(*document*, *validate=True*, *conflict='error'*, *durability='hard'*)
Add a document to the collection

Parameters

- **document** (*dict or str or Document*) – The document to add
- **validate** (*bool*) – Default True. Whether or not to validate the document against the collection schema.
- **conflict** (*'error' or 'replace'*) – Default 'error'. See RethinkDB meaning
- **durability** (*'hard' or 'soft'*) – Default 'hard'. See RethinkDB meaning

application = None
Application object – This is set when register_application is called. Set if you want to restrict to a specific application

collection = None
string – The name of the collection in the store. Defaults to the name of the schema.

create()
Create the collection in the database

document_class
Document subclass: The class of Document to use (some documents might implement instance methods)

alias of `Document`

document_store = 'default'
string – The name of the document store in settings.

documents(*cursor*)
Wrap the results of a query in Document instances

drop()
Drop the collection from the database

fresh(***document*)
Returns a fresh, unsaved document for this collection

get_endpoint()
Return the unique name for this collection.

get_name()
Return the name of this document collection.

get_url(*document=None*)

json(*cursor*, *dereference_method='dict'*, *file_obj=None*, *indent=None*)
Wrap the resulting documents into JSON

linked_schema()

name = None
str – The name of this collection

post_registration()

q(*query*)
Run a RethinkDB query in the raw

query(*query*)
Wrap query results in Document instances

reference

save(*validate=True*, *conflict='error'*, *durability='hard'*, **documents*)
Save the listed documents in the database, optionally validating them.

Parameters

- **validate** (*bool*) – Do or do not perform validation on each document before saving.
- **conflict** (*'error' or 'replace'*) – How to handle it if documents already exist in the collection by id.
- **durability** (*'hard' or 'soft'*) – RethinkDB durability argument to `r.table().insert()`
- ***documents** (*Document*) –

schema = None
dict – Optional. Either define schema or schema_url. The schema of the collection.

schema_url = None
string – Optional. Define schema if not this. URL of schema the collection conforms to. Uses JSON-Loader

table

url

exception sondra.models.**DocumentCollectionException**
Bases: builtins.Exception

An exception in the specification, access, or construction of a DocumentCollection

exception sondra.models.**InvalidSpecification**
Bases: sondra.models.DocumentCollectionException

Invalid specification of the DocumentCollection subclass

class sondra.models.**ObjectFragment**(*parent*, *value*)
Bases: collections.abc.MutableMapping

A fragment that holds a JSON object (dict). When a Document contains an Object, this proxy is returned by __getitem__

save(*validate=True*, *conflict='error'*, *durability='hard'*, *return_changes=False*)

sondra.models.**camelcase_slugify**(*name*)

sondra.models.**convert_camelcase**(*name*)

sondra.models.**document_encoder**(*document_serializer='dict'*, *parent_default=None*)
Constructor for JSON "default" function that encodes documents

Parameters

- **document_serializer** (*str*) – 'dict', 'url', or 'link'. Determine how to serialize documents.
- **parent_default** (*function*) – a json.dump default function to pass objects to if this encoder
- **fails.** –

sondra.models.**exposable**(*permissions=()*, *modifies_args=()*)
Defines a method that is exposable as an API call on the defining class.

This method parses function annotations to determine the schema of arguments and returns. All exposed methods MUST have all arguments (except self) specified as given here. Valid parameter annotations:

```
@exposable(permissions=('my_object.write'))
def attach_object(
    self, x: float,
    y: float,
    obj: "/my_app/some_collection",
    freeze: bool=False
) -> None:
    ...
    ...
    ...
```

Keyword arguments and variable length arguments are not supported.

Parameters

- **{str}** (*permissions*) – A set of permission names, one of which the user must have to execute this method.
- **modifies_args={str}** – The names of any Document arguments that the method might modify. This helps the API do permissions checking.

sondra.views module

class sondra.views.**ExposeAPI**
Bases: flask.views.MethodView

get(*object_id=None*)

methods = ['GET']

sondra.ref module

exception sondra.ref.**EndpointError**
Bases: builtins.Exception

Raised when an API request is parsed correctly, but the endpoint isn't found

exception sondra.ref.**ParseError**
Bases: builtins.Exception

Called when an API request is not parsed as valid

class sondra.ref.**Reference**(*sondra*, *url=None*, ***kw*)
Bases: builtins.object

Contains the application, collection, document, methods, and fragment the URL refers to

FORMATS = {'help', 'json', 'jsonp', 'schema', 'form', 'html'}

construct()
Construct a URL from its base parts

classmethod dereference(*sondra*, *value*)

classmethod dereference_all(*sondra*, *value*)

get_application()
Return an Application instance for the given URL.

Parameters **url** (*str*) – The URL to a collection or a document.

Returns An Application object

get_application_method()
Return everything you need to call an application method.

Returns

- application object
- method name
- method object

Return type A three-tuple of

Raises EndpointError if the method or application is not found or the method is not exposable.

get_collection()
Return a DocumentCollection instance for the given URL.

Parameters **url** (*str*) – The URL to a collection or a document.

Returns A DocumentCollection object

get_collection_method()
Return everything you need to call an collection method.

Returns

- collection object
- method name
- method object

Return type A three-tuple of

Raises EndpointError if the method or collection is not found or the method is not exposable.

get_document()
Return the Document for a given URL.

get_document_method()
Return everything you need to call an document method.

Returns

- document object
- method name
- method object

Return type A three-tuple of

Raises EndpointError if the method or document is not found or the method is not exposable.

get_subdocument()
Return the fragment within the Document referred to by this URL.

get_subdocument_method()
Return everything you need to call an subdocument method.

Returns

- subdocument object
- method name
- method object
- parent document object

Return type A four-tuple of

Raises EndpointError if the method or subdocument is not found or the method is not exposable.

is_application()

is_application_method_call()

is_collection()

is_collection_method_call()

is_document()

is_document_method_call()

is_subdocument()

is_subdocument_method_call()

kind

classmethod maybe_reference(*sondra*, *value*)

value

Module contents

CHAPTER 5

Indices and tables

- *genindex*
- *modindex*
- *search*

S

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W