sandman Documentation

Release 0.9.8

Jeff Knupp

June 14, 2016

Contents

Contents:

CHAPTER 1

Installation

Simply run:

```
pip install sandman
```

CHAPTER 2

Using Sandman

2.1 The Simplest Application

Here's what's required to create a RESTful API service from an existing database using `sandman`

```
$ sandmanctl sqlite:////tmp/my_database.db
```

That's it. `sandman` will then do the following:

- Connect to your database and introspect it's contents
- Create and launch a RESTful API service
- Create an HTML admin interface
- *Open your browser to the admin interface*

That's right. Given a legacy database, `sandman` not only gives you a REST API, it gives you a beautiful admin page and *opens your browser to the admin page*. It truly does everything for you.

2.2 Supported Databases

`sandman` , by default, supports connections to the same set of databases as SQLAlchemy (http://www.sqlalchemy.org). As of this writing, that includes:

- MySQL (MariaDB)
- PostgreSQL
- SQLite
- Oracle
- Microsoft SQL Server
- Firebird
- Drizzle
- Sybase
- IBM DB2
- SAP Sybase SQL Anywhere
- MonetDB

2.3 Beyond *sandmanctl*

`sandmanctl` is really just a simple wrapper around the following:

```
from ``sandman`` import app

app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///chinook'

from sandman.model import activate

activate(browser=True)

app.run()
```

Notice you don't even need to tell ''sandman'' what tables your database contains. Just point `sandman` at your database and let it do all the heavy lifting.

If you put the code above into a file named `runserver.py`, You can start this new service and make a request. While we're at it, lets make use of `sandman`'s awesome filtering capability by specifying a filter term:

```
$ python runserver.py &
* Running on http://127.0.0.1:5000/

> curl GET "http://localhost:5000/artists?Name=AC/DC"
```

you should see the following:

```
{
    "resources": [
        {
            "ArtistId": 1,
            "Name": "AC/DC",
            "links": [
                {
                    "rel": "self",
                    "uri": "/artists/1"
                }
            ]
        }
    ]
}
```

If you were to leave off the filtering term, you would get **all** results from the `Artist` table. You can also *paginate* these results by specifying `?page=2` or something similar. The number of results returned per page is controlled by the config value `RESULTS_PER_PAGE`, which defaults to 20.

2.3.1 A Quick Guide to REST APIs

Before we get into more complicated examples, we should discuss some REST API basics. The most important concept is that of a *resource*. Resources are sources of information, and the API is an interface to this information. That is, resources are the actual "objects" manipulated by the API. In `sandman`, each row in a database table is considered a resource.

Groups of resources are called *collections*. In `sandman`, each table in your database is a collection. Collections can be queried and added to using the appropriate *HTTP method*. `sandman` supports the following HTTP methods:

```
* GET
* POST
* PUT
* DELETE
* PATCH
```

(Support for the `HEAD` and `OPTIONS` methods is underway.)

CHAPTER 3

Creating Models

A `Model` represents a table in your database. You control which tables to expose in the API through the creation of classes which inherit from *`sandman.model.models.Model`*. If you create a `Model`, the only attribute you must define in your class is the `__tablename__` attribute. `sandman` uses this to map your class to the corresponding database table. From there, `sandman` is able to divine all other properties of your tables. Specifically, `sandman` creates the following:

- an `__endpoint__` attribute that controls resource URIs for the class
- a `__methods__` attribute that determines the allowed HTTP methods for the class
- `as_dict` and `from_dict` methods that only operate on class attributes that correspond to database columns
- an `update` method that updates only the values specified (as opposed to `from_dict`, which replaces all of the object's values with those passed in the dictionary parameter
- `links`, `primary_key`, and `resource_uri` methods that provide access to various attributes of the object derived from the underlying database model

Creating a `models.py` file allows you to get *even more* out of `sandman`. In the file, create a class that derives from `sandman.models.Model` for each table you want to turn into a RESTful resource. Here's a simple example using the Chinook test database (widely available online):

```
from sandman.model import register, activate, Model

class Artist(Model):
    __tablename__ = 'Artist'

class Album(Model):
    __tablename__ = 'Album'

class Playlist(Model):
    __tablename__ = 'Playlist'

class Genre(Model):
    __tablename__ = 'Genre'

# register can be called with an iterable or a single class
register((Artist, Album, Playlist))
register(Genre)
# activate must be called *after* register
activate(browser=False)
```

3.1 Hooking up Models

The __tablename__ attribute is used to tell sandman which database table this class is modeling. It has *no default* and is *required* for all classes.

3.2 Providing a custom endpoint

In the code above, we created four *sandman.model.models.Model* classes that correspond to tables in our database. If we wanted to change the HTTP endpoint for one of the models (the default endpoint is simply the class's name pluralized in lowercase), we would do so by setting the __endpoint__ attribute in the definition of the class:

```
class Genre(Model):
    __tablename__ = 'Genre'
    __endpoint__ = 'styles'
```

Now we would point our browser (or curl) to localhost:5000/styles to retrieve the resources in the Genre table.

3.3 Restricting allowable methods on a resource

Many times, we'd like to specify that certain actions can only be carried out against certain types of resources. If we wanted to prevent API users from deleting any Genre resources, for example, we could specify this implicitly by defining the __methods__ attribute and leaving out the DELETE method, like so:

```
class Genre(Model):
    __tablename__ = 'Genre'
    __endpoint__ = 'styles'
    __methods__ = ('GET', 'POST', 'PATCH', 'PUT')
```

For each call into the API, the HTTP method used is validated against the acceptable methods for that resource.

3.4 Performing custom validation on a resource

Specifying which HTTP methods are acceptable gives rather coarse control over how a user of the API can interact with our resources. For more granular control, custom a validation function can be specified. To do so, simply define a static method named validate_<METHOD>, where <METHOD> is the HTTP method the validation function should validate. To validate the POST method on Genres, we would define the method validate_POST, like so:

```
class Genre(Model):
    __tablename__ = 'Genre'
    __endpoint__ = 'styles'
    __methods__ = ('GET', 'POST', 'PATCH', 'PUT')

    @staticmethod
    def validate_POST(self, resource=None):
        if isinstance(resource, list):
            return True

        # No classical music!
        return resource and resource.Name != 'classical'
```

The `validate_POST` method is called *after* the would-be resource is created, trading a bit of performance for a simpler interface. Instead of needing to inspect the incoming HTTP request directly, you can make validation decisions based on the resource itself.

Note that the `resource` parameter can be either a single resource or a collection of resources, so it's usually necessary to check which type you're dealing with. This will likely change in a future version of sandman.

3.5 Configuring a model's behavior in the admin interface

`sandman` uses *Flask-Admin* to construct the admin interface. While the default settings for individual models are usually sufficient, you can make changes to the admin interface for a model by setting the *__view__* attribute to a class that derives from *flask.ext.admin.contrib.sqla.ModelView*. The Flask-Admin's documentation should be consulted for the full list of attributes that can be configured.

Below, we create a model and, additionally, tell `sandman` that we want the table's primary key to be displayed in the admin interface (by default, a table's primary keys aren't shown):

```
from flask.ext.admin.contrib.sqla import ModelView

class ModelViewShowPK(ModelView):

  column_display_pk = True

class Artist(Model):
  __tablename__ = 'Artist'
  __view__ = ModelViewShowPK
```

Custom '__view__' classes are a powerful way to customize the admin interface. Properties exist to control which columns are sortable or searchable, as well as as what fields are editable in the built-in editing view. If you find your admin page isn't working exactly as you'd like, the chances are good you can add your desired functionality through a custom *__view__* class.

CHAPTER 4

Model Endpoints

If you were to create a `Model` class named `Resource`, the following endpoints would be created:

- **`resources/`**
  - `GET`: retrieve all resources (i.e. the *collection*)
  - `POST`: create a new resource
- **`resources/<id>`**
  - `GET`: retrieve a specific resource
  - `PATCH`: update an existing resource
  - `PUT`: create or update a resource with the given ID
  - `DELETE`: delete a specific resource
- **`resources/meta`**
  - `GET`: retrieve a description of a resource's structure

4.1 The root endpoint

For each project, a "root" endpoint (`/`) is created that gives clients the information required to interact with your API. The endpoint for each resource is listed, along with the `/meta` endpoint describing a resource's structure.

The root endpoint is available as both JSON and HTML. The same information is returned by each version.

4.2 The `/meta` endpoint

A `/meta` endpoint, which lists the models attributes (i.e. the database columns) and their type. This can be used to create client code that is decoupled from the structure of your database.

A `/meta` endpoint is automatically generated for every `Model` you register. This is available both as JSON and HTML.

CHAPTER 5

Automatic Introspection

Of course, you don't actually need to tell `sandman` about your tables; it's perfectly capable of introspecting all of them. To use introspection to make *all* of your database tables available via the admin and REST API, simply remove all model code and call *activate()* without ever registering a model. To stop a browser window from automatically popping up on sandman initialization, call *activate()* with *browser=False*.

CHAPTER 6

Running sandman alongside another app

If you have an existing WSGI application you'd like to run in the same interpreter as sandman, follow the instructions described here. Essentially, you need to import both applications in your main file and use Flask's DispatcherMiddleware to give a unique route to each app. In the following example, sandman-related endpoints can be accessed by adding the /sandman prefix to sandman's normally generated URIs:

```
from my_application import app as my_app
from sandman import app as sandman_app
from werkzeug.wsgi import DispatcherMiddleware

application = DispatcherMiddleware(my_app, {
    '/sandman': sandman_app,
    })
```

This allows both apps to coexist; my_app will be rooted at / and sandman at /sandman.

6.1 Using existing declarative models

If you have a Flask/SQLAlchemy application that already has a number of existing declarative models, you can register these with sandman as if they were auto-generated classes. Simply add your existing classes in the call to *sandman.model.register()*

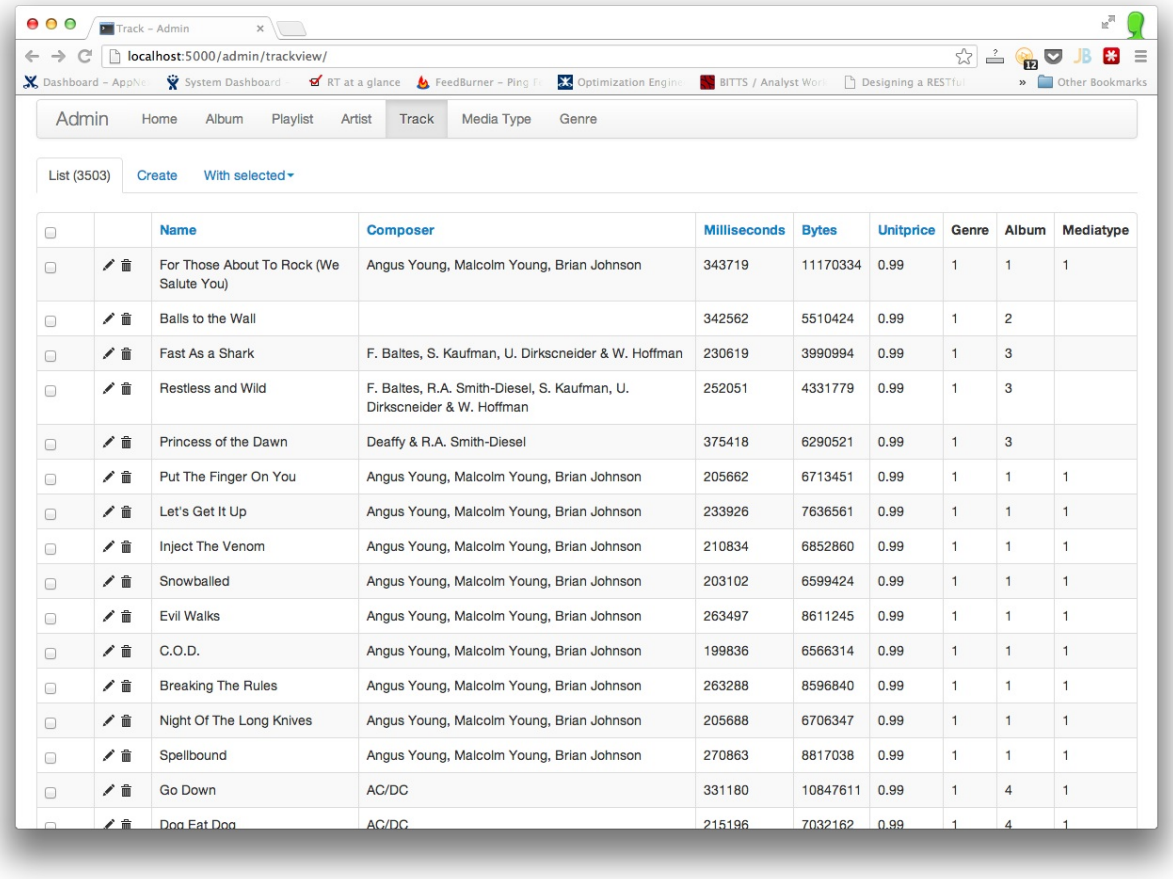
CHAPTER 7

The sandman Admin Interface

7.1 Activating the sandman Admin Interface

sandman supports an admin interface, much like the Django admin interface. `sandman` currently uses [Flask-Admin](https://flask-admin.readthedocs.org/en/latest/) and some SQLAlchemy, erm, alchemy to allow your resources to be administered via the admin interface. Note, though, that the admin interface may drastically change in the future.

Here's a look at the interface generated for the `chinook` database's `Track` table, listing the information about various music tracks:



		Name	Composer	Milliseconds	Bytes	Unitprice	Genre	Album	Mediatype
☐	✎ 🗑	For Those About To Rock (We Salute You)	Angus Young, Malcolm Young, Brian Johnson	343719	11170334	0.99	1	1	1
☐	✎ 🗑	Balls to the Wall		342562	5510424	0.99	1	2	
☐	✎ 🗑	Fast As a Shark	F. Baltes, S. Kaufman, U. Dirkscneider & W. Hoffman	230619	3990994	0.99	1	3	
☐	✎ 🗑	Restless and Wild	F. Baltes, R.A. Smith-Diesel, S. Kaufman, U. Dirkscneider & W. Hoffman	252051	4331779	0.99	1	3	
☐	✎ 🗑	Princess of the Dawn	Deaffy & R.A. Smith-Diesel	375418	6290521	0.99	1	3	
☐	✎ 🗑	Put The Finger On You	Angus Young, Malcolm Young, Brian Johnson	205662	6713451	0.99	1	1	1
☐	✎ 🗑	Let's Get It Up	Angus Young, Malcolm Young, Brian Johnson	233926	7636561	0.99	1	1	1
☐	✎ 🗑	Inject The Venom	Angus Young, Malcolm Young, Brian Johnson	210834	6852860	0.99	1	1	1
☐	✎ 🗑	Snowballed	Angus Young, Malcolm Young, Brian Johnson	203102	6599424	0.99	1	1	1
☐	✎ 🗑	Evil Walks	Angus Young, Malcolm Young, Brian Johnson	263497	8611245	0.99	1	1	1
☐	✎ 🗑	C.O.D.	Angus Young, Malcolm Young, Brian Johnson	199836	6566314	0.99	1	1	1
☐	✎ 🗑	Breaking The Rules	Angus Young, Malcolm Young, Brian Johnson	263288	8596840	0.99	1	1	1
☐	✎ 🗑	Night Of The Long Knives	Angus Young, Malcolm Young, Brian Johnson	205688	6706347	0.99	1	1	1
☐	✎ 🗑	Spellbound	Angus Young, Malcolm Young, Brian Johnson	270863	8817038	0.99	1	1	1
☐	✎ 🗑	Go Down	AC/DC	331180	10847611	0.99	1	4	1
☐	✎ 🗑	Dog Eat Dog	AC/DC	215196	7032162	0.99	1	4	1

Pretty nice! From here you can directly create, edit, and delete resources. In the "create" and "edit" forms, objects

related via foreign key (e.g. a `Track`'s associated `Album`) are auto-populated in a dropdown based on available values. This ensures that all database constraints are honored when making changes via the admin.

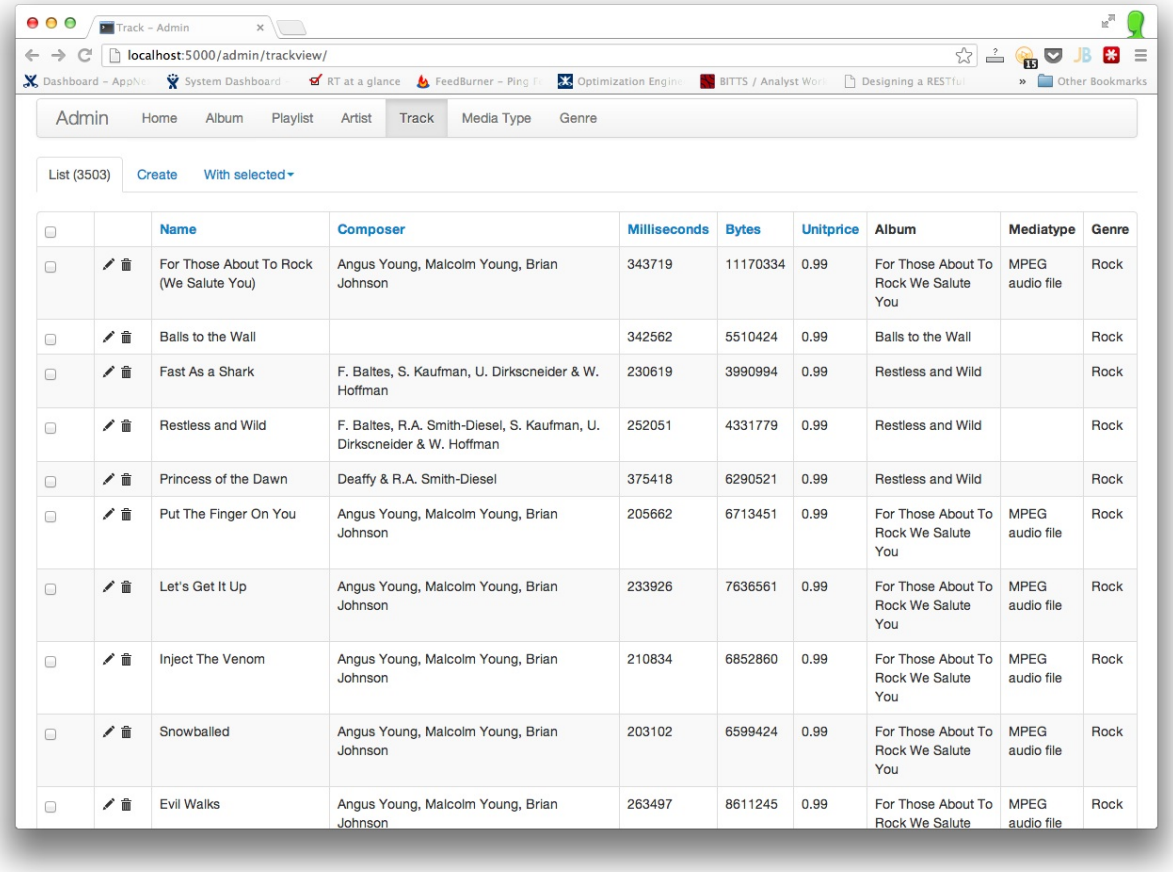
The admin interface (which adds an `/admin` endpoint to your service, accessible via a browser), is enabled by default. To disable it, pass `admin=False` as an argument in your call to `activate`. By default, calling this function will make _all_ Models accessible in the admin. If you'd like to prevent this, simply call `register()` with `use_admin=False` for whichever Model/Models you don't want to appear. Alternatively, you can control if a model is viewable, editable, creatable, etc in the admin by setting your class's `__view__` attribute to your own `Admin` class.

7.2 Getting Richer Information for Related Objects

The sharp-eyed among you may have noticed that the information presented for `Album`, `Genre`, and `MediaType` are not very helpful. By default, the value that will be shown is the value returned by `__str__` on the associated table. Currently, `__str__` simply returns the value of a Model's `primary_key()` attribute. By overriding `__str__`, however, we can display more useful information. After making the changes below:

```
from sandman.model import register, Model

class Track(Model):
    __tablename__ = 'Track'

    def __str__(self):
        return self.Name

class Artist(Model):
    __tablename__ = 'Artist'

    def __str__(self):
        return self.Name

class Album(Model):
    __tablename__ = 'Album'

    def __str__(self):
        return self.Title

class Playlist(Model):
    __tablename__ = 'Playlist'

    def __str__(self):
        return self.Id

class Genre(Model):
    __tablename__ = 'Genre'

    def __str__(self):
        return self.Name

class MediaType(Model):
    __tablename__ = 'MediaType'

    def __str__(self):
        return self.Name

register((Artist, Album, Playlist, Genre, Track, MediaType))
```

we get much more useful information in the columns mentioned, as you can see here:



		Name	Composer	Milliseconds	Bytes	Unitprice	Album	Mediatype	Genre
☐	✎ 🗑	For Those About To Rock (We Salute You)	Angus Young, Malcolm Young, Brian Johnson	343719	11170334	0.99	For Those About To Rock We Salute You	MPEG audio file	Rock
☐	✎ 🗑	Balls to the Wall		342562	5510424	0.99	Balls to the Wall		Rock
☐	✎ 🗑	Fast As a Shark	F. Baltes, S. Kaufman, U. Dirkscneider & W. Hoffman	230619	3990994	0.99	Restless and Wild		Rock
☐	✎ 🗑	Restless and Wild	F. Baltes, R.A. Smith-Diesel, S. Kaufman, U. Dirkscneider & W. Hoffman	252051	4331779	0.99	Restless and Wild		Rock
☐	✎ 🗑	Princess of the Dawn	Deaffy & R.A. Smith-Diesel	375418	6290521	0.99	Restless and Wild		Rock
☐	✎ 🗑	Put The Finger On You	Angus Young, Malcolm Young, Brian Johnson	205662	6713451	0.99	For Those About To Rock We Salute You	MPEG audio file	Rock
☐	✎ 🗑	Let's Get It Up	Angus Young, Malcolm Young, Brian Johnson	233926	7636561	0.99	For Those About To Rock We Salute You	MPEG audio file	Rock
☐	✎ 🗑	Inject The Venom	Angus Young, Malcolm Young, Brian Johnson	210834	6852860	0.99	For Those About To Rock We Salute You	MPEG audio file	Rock
☐	✎ 🗑	Snowballed	Angus Young, Malcolm Young, Brian Johnson	203102	6599424	0.99	For Those About To Rock We Salute You	MPEG audio file	Rock
☐	✎ 🗑	Evil Walks	Angus Young, Malcolm Young, Brian Johnson	263497	8611245	0.99	For Those About To Rock We Salute	MPEG audio file	Rock

CHAPTER 8

Authentication

sandman supports HTTP basic authentication, meaning a username and password must be passed on each request via the Authorization header.

8.1 Enabling Authentication

Enabling authentication in your sandman installation is a straight-forward task. You'll need to define two functions:

- get_password()
- before_request()

The former is required by Flask-HTTPAuth, which powers sandman's authentication. The latter is used to ensure that _all_ requests are authorized.

8.1.1 get_password

The get_password function takes a username as an argument and should return the associated password for that user. To notify Flask-HTTPAuth that this is the function responsible for returning passwords, it must be wrapped with the @auth.get_password decorator (auth is importable from sandman, e.g. from sandman import app, db, auth). How you implement your user management system is up to you; you simply need to implement get_password in whatever way is most appropriate for your security setup.

As a trivial example, here's an implementation of get_password that always returns secret, meaning secret must be the password, regardless of the username:

```
@auth.get_password
def get_password(username):
    """Return the password for *username*."""
    return 'secret'
```

8.1.2 before_request

Once you've hooked up your password function, it's time to tell Flask which requests should require authentication. Rather than picking and choosing on a request by request basis, we use the @app.before_request decorator included in Flask to make sure _all_ requests are authenticated. Here's a sample implementation:

```
@app.before_request
@auth.login_required
def before_request():
    pass
```

Notice the function just calls `pass`; it needn't have any logic, since the logic is added by Flask-HTTPAuth's `@auth.login_required` decorator.

8.2 Token-based Authentication

There are plans for `sandman` to support token-based authentication, but this currently isn't supported and no time frame for implementation has been set.

CHAPTER 9

sandman API

9.1 exception Module

Exception specifications for Sandman

exception sandman.exception.**InvalidAPIUsage**(*code=400*, *message=None*, *payload=None*)
Bases: exceptions.Exception

Excecption which generates a flask.Response object whose *data* is JSON rather than HTML

abort()
Return an HTML Response representation of the exception.

to_dict()
Return a dictionary representation of the exception.

9.2 model Module

The model module is repsonsible exposes the sandman.model.Model class, from which user models should derive. It also makes the *register()* function available, which maps endpoints to their associated classes.

sandman.model.**register**(*cls*, *use_admin=True*)
Register with the API a sandman.model.Model class and associated endpoint.

Parameters **cls** (sandman.model.Model or tuple) – User-defined class derived from sandman.model.Model to be registered with the endpoint returned by endpoint()

sandman.model.**activate**(*admin=True*, *browser=True*, *name='admin'*, *reflect_all=False*)
Activate each pre-registered model or generate the model classes and (possibly) register them for the admin.

Parameters

- **admin** (*bool*) – should we generate the admin interface?
- **browser** (*bool*) – should we open the browser for the user?
- **name** – name to use for blueprint created by the admin interface. Set this to avoid naming conflicts with other blueprints (if trying to use sandman to connect to multiple databases simultaneously)

The Model class is meant to be the base class for user Models. It represents a table in the database that should be modeled as a resource.

class sandman.model.models.**AdminModelViewWithPK**(*model*, *session*, *name=None*, *category=None*, *endpoint=None*, *url=None*)

Bases: flask_admin.contrib.sqla.view.ModelView

Mixin admin view class that displays primary keys on the admin form

_default_view = 'index_view'

_urls = [('/action/', 'action_view', ('POST',)), ('/ajax/lookup/', 'ajax_lookup', ('GET',)), ('/new/', 'create_view', ('GET',

action_view(**args*, ***kwargs*)
Mass-model action view.

ajax_lookup(**args*, ***kwargs*)

column_display_pk = True

create_view(**args*, ***kwargs*)
Create model view

delete_view(**args*, ***kwargs*)
Delete model view. Only POST method is allowed.

edit_view(**args*, ***kwargs*)
Edit model view

index_view(**args*, ***kwargs*)
List view

class sandman.model.models.**Model**

Bases: object

A mixin class containing the majority of the RESTful API functionality.

sandman.model.Model is the base class of *:class:'sandman.Model*, from which user models are derived.

__endpoint__ = None
override *__endpoint__* if you wish to configure the sandman.model.Model's endpoint.

Default: __tablename__ in lowercase and pluralized

__methods__ = ('GET', 'POST', 'PATCH', 'DELETE', 'PUT')
override *__methods__* if you wish to change the HTTP methods this sandman.model.Model supports.

Default: ('GET', 'POST', 'PATCH', 'DELETE', 'PUT')

__table__ = None
Will be populated by SQLAlchemy with the table's meta-information.

__tablename__ = None
The name of the database table this class should be mapped to

Default: None

as_dict(*depth=0*)
Return a dictionary containing only the attributes which map to an instance's database columns.

Parameters **depth** (*int*) – Maximum depth to recurse subobjects

Return type dict

classmethod endpoint()
Return the sandman.model.Model's endpoint.

Return type string

from_dict (*dictionary*)
Set a set of attributes which correspond to the `sandman.model.Model`'s columns.

Parameters **dictionary** (*dict*) – A dictionary of attributes to set on the instance whose keys are the column names of the `sandman.model.Model`'s underlying database table.

links ()
Return a list of links for endpoints related to the resource.

Return type list

classmethod meta ()
Return a dictionary containing meta-information about the given resource.

classmethod primary_key ()
Return the name of the table's primary key

Return type string

replace (*dictionary*)
Set all attributes which correspond to the `sandman.model.Model`'s columns to the values in *dictionary*, inserting None if an attribute's value is not specified.

Parameters **dictionary** (*dict*) – A dictionary of attributes to set on the instance whose keys are the column names of the `sandman.model.Model`'s underlying database table.

resource_uri ()
Return the URI at which the resource can be found.

Return type string

9.3 sandman Module

Sandman REST API creator for Flask and SQLAlchemy

sandman.sandman.**attribute_response** (*resource*, *name*, *value*)
Return a response for the *resource* of the appropriate content type.

Parameters **resource** (`sandman.model.Model`) – resource to be returned in request

Return type `flask.Response`

sandman.sandman.**collection_response** (*cls*, *resources*, *start=None*, *stop=None*)
Return a response for the *resources* of the appropriate content type.

Parameters **resources** – resources to be returned in request

Return type `flask.Response`

sandman.sandman.**delete_resource** (*collection*, *key*)
Return the appropriate *Response* for deleting an existing resource in *collection*.

Parameters

- **collection** (*string*) – a `sandman.model.Model` endpoint
- **key** (*string*) – the primary key for the `sandman.model.Model`

Return type `flask.Response`

sandman.sandman.**endpoint_class** (*collection*)
Return the `sandman.model.Model` associated with the endpoint *collection*.

Parameters **collection** (*string*) – a `sandman.model.Model` endpoint

Return type sandman.model.Model

sandman.sandman.**get_collection**(**args*, ***kwargs*)
Return the appropriate *Response* for retrieving a collection of resources.

Parameters

- **collection** (*string*) – a sandman.model.Model endpoint
- **key** (*string*) – the primary key for the sandman.model.Model

Return type flask.Response

sandman.sandman.**get_meta**(**args*, ***kwargs*)
Return the meta-description of a given resource.

Parameters **collection** – The collection to get meta-info for

sandman.sandman.**get_resource**(**args*, ***kwargs*)
Return the appropriate *Response* for retrieving a single resource.

Parameters

- **collection** (*string*) – a sandman.model.Model endpoint
- **key** (*string*) – the primary key for the sandman.model.Model

Return type flask.Response

sandman.sandman.**get_resource_attribute**(**args*, ***kwargs*)
Return the appropriate *Response* for retrieving an attribute of a single resource.

Parameters

- **collection** (*string*) – a sandman.model.Model endpoint
- **key** (*string*) – the primary key for the sandman.model.Model

Return type flask.Response

sandman.sandman.**get_resource_data**(*incoming_request*)
Return the data from the incoming *request* based on the Content-type.

sandman.sandman.**handle_exception**(*error*)
Return a response with the appropriate status code, message, and content type when an InvalidAPIUsage exception is raised.

sandman.sandman.**index**(**args*, ***kwargs*)
Return information about each type of resource and how it can be accessed.

sandman.sandman.**no_content_response**(**args*, ***kwargs*)
Return the appropriate *Response* with status code *204*, signaling a completed action which does not require data in the response body

Return type flask.Response

sandman.sandman.**patch_resource**(*collection*, *key*)
"Upsert" a resource identified by the given key and return the appropriate *Response*.

If no resource currently exists at */<collection>/<key>*, create it with *key* as its primary key and return a *resource_created_response()*.

If a resource *does* exist at */<collection>/<key>*, update it with the data sent in the request and return a *no_content_response()*.

Note: HTTP *PATCH* (and, thus, *patch_resource()*) is idempotent

Parameters

- **collection** (*string*) – a sandman.model.Model endpoint
- **key** (*string*) – the primary key for the sandman.model.Model

Return type flask.Response

sandman.sandman.**post_resource**(*collection*)
Return the appropriate *Response* based on adding a new resource to *collection*.

Parameters **collection** (*string*) – a sandman.model.Model endpoint

Return type flask.Response

sandman.sandman.**put_resource**(*collection*, *key*)
Replace the resource identified by the given key and return the appropriate response.

Parameters **collection** (*string*) – a sandman.model.Model endpoint

Return type flask.Response

sandman.sandman.**resource_created_response**(*resource*)
Return HTTP response with status code *201*, signaling a created *resource*

Parameters **resource** (sandman.model.Model) – resource created as a result of current request

Return type flask.Response

sandman.sandman.**resource_response**(*resource*, *depth=0*)
Return a response for the *resource* of the appropriate content type.

Parameters **resource** (sandman.model.Model) – resource to be returned in request

Return type flask.Response

sandman.sandman.**retrieve_collection**(*collection*, *query_arguments=None*)
Return the resources in *collection*, possibly filtered by a series of values to use in a 'where' clause search.

Parameters

- **collection** (*string*) – a sandman.model.Model endpoint
- **query_arguments** (*dict*) – a list of filter query arguments

Return type class:*sandman.model.Model*

sandman.sandman.**retrieve_resource**(*collection*, *key*)
Return the resource in *collection* identified by key *key*.

Parameters

- **collection** (*string*) – a sandman.model.Model endpoint
- **key** (*string*) – primary key of resource

Return type class:*sandman.model.Model*

sandman.sandman.**update_resource**(*resource*, *incoming_request*)
Replace the contents of a resource with *data* and return an appropriate *Response*.

Parameters

- **resource** – sandman.model.Model to be updated
- **data** – New values for the fields in *resource*

CHAPTER 10

Indices and tables

- genindex
- modindex
- search

S

Symbols

A

C

D

E

F

G

H

I

L

M

N

P

R

S

T

U