ripozo-sqlachemy Documentation

Release 1.0.1.dev0

Tim Martin - Vertical Knowledge

July 07, 2015

Contents

Contents:

CHAPTER 1

ripozo-sqlalchemy API

1.1 Manager

Core pieces of the AlchemyManager

class ripozo_sqlalchemy.alchemymanager.**AlchemyManager**(*session_handler*, **args*, ***kwargs*)

Bases: ripozo.manager_base.BaseManager

This is the Manager that interops between ripozo and sqlalchemy. It provides a series of convience functions primarily for basic CRUD. This class can be extended as necessary and it is recommended that direct database access should be performed in a manager.

Parameters **all_fields** (*bool*) – If this is true, then all fields on the model will be used. The model will be inspected to get the fields.

all_fields = False

create(**args*, ***kwargs*)

Creates a new instance of the self.model and persists it to the database.

Parameters

- **values** (*dict*) – The dictionary of values to set on the model. The key is the column name and the value is what it will be set to. If the cls._create_fields is defined then it will use those fields. Otherwise, it will use the fields defined in cls.fields
- **session** (*Session*) – The sqlalchemy session

Returns The serialized model. It will use the self.fields attribute for this.

Return type dict

delete(**args*, ***kwargs*)

Deletes the model found using the lookup_keys

Parameters

- **session** (*Session*) – The SQLAlchemy session to use
- **lookup_keys** (*dict*) – A dictionary mapping the fields and their expected values

Returns An empty dictionary

Return type dict

Raises NotFoundException

fields = ()

classmethod **get_field_type**(*name*)
Takes a field name and gets an appropriate BaseField instance for that column. It inspects the Model that is set on the manager to determine what the BaseField subclass should be.

Parameters **name** (*unicode*) –

Returns A BaseField subclass that is appropriate for translating a string input into the appropriate format.

Return type ripozo.viewsets.fields.base.BaseField

pagination_pk_query_arg = u'page'

queryset(*session*)
The queryset to use when looking for models.

This is advantageous to override if you only want a subset of the model specified.

retrieve(**args*, ***kwargs*)
Retrieves a model using the lookup keys provided. Only one model should be returned by the lookup_keys or else the manager will fail.

Parameters

- **session** (*Session*) – The SQLAlchemy session to use
- **lookup_keys** (*dict*) – A dictionary mapping the fields and their expected values

Returns The dictionary of keys and values for the retrieved model. The only values returned will be those specified by fields attrbute on the class

Return type dict

Raises NotFoundException

retrieve_list(**args*, ***kwargs*)
Retrieves a list of the model for this manager. It is restricted by the filters provided.

Parameters

- **session** (*Session*) – The SQLAlchemy session to use
- **filters** (*dict*) – The filters to restrict the returned models on

Returns A tuple of the list of dictionary representation of the models and the dictionary of meta data

Return type list, dict

serialize_model(*model*, *field_dict=None*)
Takes a model and serializes the fields provided into a dictionary.

Parameters

- **model** (*Model*) – The Sqlalchemy model instance to serialize
- **field_dict** (*dict*) – The dictionary of fields to return.

Returns The serialized model.

Return type dict

update(**args*, ***kwargs*)
Updates the model with the specified lookup_keys and returns the dictified object.

Parameters

- **session** (*Session*) – The SQLAlchemy session to use

- **lookup_keys** (*dict*) – A dictionary mapping the fields and their expected values
- **updates** (*dict*) – The columns and the values to update them to.

Returns The dictionary of keys and values for the retrieved model. The only values returned will be those specified by fields attrbute on the class

Return type dict

Raises NotFoundException

ripozo_sqlalchemy.alchemymanager.**db_access_point**(*func*)
Wraps a function that actually accesses the database. It injects a session into the method and attempts to handle it after the function has run.

Parameters **func** (*method*) – The method that is interacting with the database.

1.2 Sessions

class ripozo_sqlalchemy.session_handlers.**ScopedSessionHandler**(*engine*)
Bases: object

A ScopedSessionHandler is injected into the AlchemyManager in order to get and handle sessions after a database access.

There are two required methods for any session handler. It must have a

get_session()
Gets an individual session.

Returns The session object.

Return type Session

static handle_session(*session*, *exc=None*)
Handles closing a session.

Parameters

- **session** (*Session*) – The session to close.
- **exc** (*Exception*) – The exception raised, If an exception was raised, else None

class ripozo_sqlalchemy.session_handlers.**SessionHandler**(*session*)
Bases: object

The SessionHandler doesn't do anything. This is helpful in Flask-SQLAlchemy for example where all of the session handling is already under control

get_session()
Gets the session

Returns The session for the manager.

Return type Session

static handle_session(*session*, *exc=None*)
rolls back the session if appropriate.

Parameters

- **session** (*Session*) – The session in use.
- **exc** (*Exception*) – The exception raised, If an exception was raised, else None

1.3 Easy Resources

ripozo_sqlalchemy.easy_resource.**create_resource**(*model*, *session_handler*, *resource_bases=(<class 'ripozo.resources.restmixins.CRUDL'>,)*, *relationships=None*, *links=None*, *preprocessors=None*, *postprocessors=None*, *fields=None*, *paginate_by=100*, *auto_relationships=True*, *pks=None*, *create_fields=None*, *update_fields=None*, *list_fields=None*)

Creates a ResourceBase subclass by inspecting a SQLAlchemy Model. This is somewhat more restrictive than explicitly creating managers and resources. However, if you only need any of the basic CRUD+L operations,

Parameters

- **model** (*sqlalchemy.Model*) – This is the model that will be inspected to create a Resource and Manager from. By default, all of it's fields will be exposed, although this can be overridden using the fields attribute.
- **resource_bases** (*tuple*) – A tuple of ResourceBase subclasses. Defaults to the restmixins.CRUDL class only. However if you only wanted Update and Delete you could pass in `(restmixins.Update, restmixins.Delete)` which would cause the resource to inherit from those two. Additionally, you could create your own mixins and pass them in as the resource_bases
- **relationships** (*tuple*) – extra relationships to pass into the ResourceBase constructor. If auto_relationships is set to True, then they will be appended to these relationships.
- **links** (*tuple*) – Extra links to pass into the ResourceBase as the class _links attribute. Defaults to an empty tuple.
- **preprocessors** (*tuple*) – Preprocessors for the resource class attribute.
- **postprocessors** (*tuple*) – Postprocessors for the resource class attribute.
- **session_handler** (*ripozo_sqlalchemy.SessionHandler*) – A session handler to use when instantiating an instance of the Manager class created from the model. This is responsible for getting and handling sessions in both normal cases and exceptions.
- **fields** (*tuple*) – The fields to expose on the api. Defaults to all of the fields on the model.
- **auto_relationships** (*bool*) – If True, then the SQLAlchemy Model will be inspected for relationships and they will be automatically appended to the relationships on the resource class attribute.
- **create_fields** (*list*) – A list of the fields that are valid when creating a resource. By default this will be the fields without any primary keys included
- **update_fields** (*list*) – A list of the fields that are valid when updating a resource. By default this will be the fields without any primary keys included
- **list_fields** (*list*) – A list of the fields that will be returned when the list endpoint is requested. Defaults to the fields attribute.

Returns A ResourceBase subclass and AlchemyManager subclass

Return type ResourceMetaClass

CHAPTER 2

ripozo-sqlalchemy example

```
from sqlalchemy import Column, Integer, String, create_engine
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import Session

# Setup the database with sqlalchemy
engine = create_engine('sqlite:///:memory:', echo=True)
Base = declarative_base(engine)
session = Session(engine)

# Declare your ORM model
class Person(Base):
    __tablename__ = 'person'
    id = Column(Integer, primary_key=True)
    first_name = Column(String)
    last_name = Column(String)
    secret = Column(String)

Base.metadata.create_all()
```

2.1 Creatings your manager

```
from ripozo_sqlalchemy import AlchemyManager, SessionHandler

class PersonManager(AlchemyManager):
    fields = ['id', 'first_name', 'last_name']
    model = Person
    paginate_by = 10

session_handler = SessionHandler(session)
```

2.2 And the resource...

```
from ripozo import restmixins

class PersonResource(restmixins.CRUDL):
    resource_name = 'people'
    manager = PersonManager(session_handler)
```

```
    namespace = '/api'
    pks = ['id']
```

2.3 Creating a person

```
>>> from ripozo import RequestContainer
>>> req = RequestContainer(body_args=dict(first_name='Hey', last_name='there'))
>>> person = PersonResource.create(req)
>>> print(person.properties['first_name'])
Hey
>>> print(person.properties['last_name'])
there
>>> print(person.url)
/api/people/1
```

2.4 Retrieving a person

```
>>> person_id = person.properties['id']
>>> req = RequestContainer(url_params=dict(id=person_id))
>>> retrieved = PersonResource.retrieve(req)
>>> print(person.properties['first_name'])
Hey
>>> print(person.properties['last_name'])
there
```

2.5 Updating a person

```
>>> req = RequestContainer(url_params=dict(id=person_id), body_args=dict(first_name='Bob'))
>>> person = PersonResource.update(req)
>>> print(person.properties['first_name'])
Bob
>>> print(person.properties['last_name'])
there
>>> req = RequestContainer(url_params=dict(id=person_id))
>>> retrieved = PersonResource.retrieve(req)
>>> print(person.properties['first_name'])
Bob
>>> print(person.properties['last_name'])
there
```

2.6 Retrieving many

```
>>> for i in range(10):
...     req = RequestContainer(body_args=dict(first_name='John', last_name=i))
...     res = PersonResource.create(req)
>>> req = RequestContainer()
>>> resource_list = PersonResource.retrieve_list(req)
>>> assert len(resource_list.related_resources[0].resource) == 10 # only ten because paginate_by=10
```

```
>>> print(resource_list.url)
/api/people
```

CHAPTER 3

ripozo-sqlalchemy

This package is a ripozo extension that provides a Manager that integrate SQLAlchemy with ripozo. It provides convience functions for generating resources. In particular, it focuses on creating shortcuts for CRUD type operations. It fully implements the BaseManager class that is provided in the ripozo package.

Full Documentation

CHAPTER 4

Example

This is a minimal example of creating ripozo managers with ripozo-sqlalchemy and integrating them with a resource.

First we need to setup our SQLAlchemy model.

```
from ripozo import apimethod, ResourceBase

from sqlalchemy import Column, Integer, String, create_engine
from sqlalchemy.ext.declarative import declarative_base

# Setup the database with sqlalchemy
engine = create_engine('sqlite:///:memory:', echo=True)
Base = declarative_base()

# Declare your ORM model
class Person(Base):
    __tablename__ = 'person'
    id = Column(Integer, primary_key=True)
    first_name = Column(String)
    last_name = Column(String)

# Sync the models wiht the database
Base.metadata.create_all()
```

Now we can get to the ripozo-sqlalchemy part.

```
from ripozo_sqlalchemy import AlchemyManager, ScopedSessionHandler

# A session handler if responsible for getting
# And handling a session after either a successful or unsuccesful request
session_handler = ScopedSessionHandler(engine)

# This is the code that is specific to ripozo-sqlalchemy
# You give it the session, a SQLAlchemy Model, and the fields
# You wish to serialize at a minimum.
class PersonManager(AlchemyManager):
    model = Person
    fields = ('id', 'first_name', 'last_name')


# This is the ripozo specific part.
# This creates a resource class that can be given
# to a dispatcher (e.g. the flask-ripozo package's FlaskDispatcher)
class PersonResource(ResourceBase):
    manager = PersonManager(session_handler)
```

```
    pks = ['id']
    namespace = '/api'

    # A retrieval method that will operate on the '/api/person' route
    # It retrieves the id, first_name, and last_name properties
    @apimethod(methods=['GET'])
    def get_person(cls, primary_keys, filters, values, *args, **kwargs):
        properties = self.manager.retrieve(primary_keys)
        return cls(properties=properties)
```

4.1 Easy Resources

Alternatively, we could use the create_resource method which will automatically create a manager and resource that corresponds to the manager.

```
from ripozo import restmixins
from ripozo_sqlalchemy import ScopedSessionHandler, create_resource

session_handler = ScopedSessionHandler(engine)
person_resource_class = create_resource(Person, session_handler)
```

By default create_resource will give you full CRUD+L (Create, Retrieve, Update, Delete, List). Although there are many options that you can pass to create_resource to modify exactly how the resource class is constructed.

After you create your resource class, you will need to load it into a dispatcher corresponding to your framework. For example, in flask-ripozo

```
from flask import Flask
from flask_ripozo import FlaskDispatcher
from ripozo.adapters import SirenAdapter, HalAdapter # These are the potential formats to return

app = Flask(__name__)
dispatcher = FlaskDispatcher(app)
dispatcher.register_adapters(SirenAdapter, HalAdapter)
dispatcher.register_resources(person_resource_class)
# or in the first style of generating resources
# dispatcher.register_resources(PersonResource)

app.run()
```

CHAPTER 5

1.0.1 (unreleased)

- Easy resources updated to use create_fields, update_fields, and list_fields options.

CHAPTER 6

1.0.0 (2015-06-30)

- Added _COLUMN_FIELD_MAP for determining python type. Transparent change.

CHAPTER 7

1.0.0b1 (2015-06-29)

- Fixed bug in retrieve_list with improperly named link to previous (“prev” –> “previous”)
- Removed all fields flag.
- Renamed alcehmymanager to alchemymanager
- easy resources added. By simply calling create_resource with a session_handler and sqlalchemy model, you can automatically create a full resource. and immediately add it to a dispatcher.

CHAPTER 8

0.2.0 (2015-06-08)

- Tests fixed.

CHAPTER 9

0.2.0b1 (2015-06-05)

- Breaking Change: You are now required to inject a session handler on instantiation of the manager.

0.1.6b1 (2015-06-04)

- Sessions are only grabbed once in any given method. This allows you to safely return a new session every time
- Added a method for after a CRUD statement has been called.

CHAPTER 11

0.1.5 (2015-04-28)

- Optimization for retrieving lists using `AlchemyManager.list_fields` property for retrieving lists
- Retrieve list now properly applies filters.
- meta links updated in retrieve_list. They now are contained in the links dictionary
- previous linked rename to prev in retrieve_list meta information

0.1.4 (2015-03-26)

- Nothing changed yet.

0.1.3 (2015-03-26)

- Nothing changed yet.

0.1.2 (2015-03-24)

- NotFoundException raised when retrieve is called and no model is found.

0.1.1 (2015-03-15)

- Added convience attribute for using all of the columns on the model.

CHAPTER 16

Indices and tables

- genindex
- modindex
- search

r