



Cornice and SQLAlchemy

by **Francis Bull** | Mar 5, 2014 | **Developer Blog** | **2 comments**

Cornice provides helpers to build and document REST-ish Web Services with Pyramid; and **SQLAlchemy** is the best Python ORM. I wanted to use Cornice and SQLAlchemy to make a simple RESTful webapp and couldn't find any info on how to put them together.

```
400 {"status": "error", "errors": [{"location": "http://localhost:6543/tasks", "name": "Non-unique task name.", "description": "There is already a task with this name."}]}
```

Cornice Validator

Here's how (full source in this [github repo](#)):

Make a simple Cornice application

Following the [tutorial](#) pretty closely.

```
$ mkvirtualenv blogpostcorniceapp
$ pip install cornice
$ pcreate -t blogpostcorniceapp
$ cd blogpostcorniceapp
$ python setup.py develop
$ pserve blogpostcorniceapp.ini
```

visit localhost:6543 -> {"Hello": "World"}

Excellent!

Now we can define a service, like the tutorial we'll store the models in memory for now.

views.py:

```
1  """ Cornice services.
2  """
3  from cornice import Service
4
5  _TASKS = {}
6  tasks = Service(name='tasks', path='/tasks', description="Tasks")
7
8  @tasks.get()
9  def get_info(request):
10     """Returns a list of all tasks."""
11     return {'tasks': _TASKS.keys()}
12
13  @tasks.post()
14  def create_task(request):
15     """Adds a new task."""
16     task = request.json
17     if task['name'] in _TASKS:
18         raise Exception('That task already exists!')
19     _TASKS[task['name']] = task
```

and if we make a script to exercise it like this:

```
1  import simplejson as json
2
3  import requests
4
5  task = {
6      'name': 'take_out_the_trash',
7      'description': ('empty the trashcan and put the bag in the outside trashcan, "
8                      "don't forget to put a new bag in!"),
9  }
10
11  response = requests.get('http://localhost:6543/tasks')
12  print response.status_code, response.text
13  response = requests.post('http://localhost:6543/tasks', json.dumps(task))
14  print response.status_code, response.text
15  response = requests.get('http://localhost:6543/tasks')
16  print response.status_code, response.text
```

it gives output:

```
200 {"tasks": []}
200 null
200 {"tasks": ["take_out_the_trash"]}
```

All right! we're RESTing!

Hook up to a database with SQLAlchemy

We need to:

- Define a SQLAlchemy model.
- Create a database and create the table structure from the models.
- Set up connections to the database when the webapp loads.
- Add some config to define the url for the database and ask pyramid to handle transaction management for us.

1. models.py

```
1 from sqlalchemy import Column
2 from sqlalchemy import Integer
3 from sqlalchemy import Text
4 from sqlalchemy.ext.declarative import declarative_base
5 from sqlalchemy.orm import scoped_session
6 from sqlalchemy.orm import sessionmaker
7 from zope.sqlalchemy import ZopeTransactionExtension
8
9 Base = declarative_base()
10 DBSession = scoped_session(sessionmaker(extension=ZopeTransactionExtension()))
11
12 class Task(Base):
13     __tablename__ = 'task'
14     task_id = Column(Integer, primary_key=True)
15     name = Column(Text, nullable=False)
16     description = Column(Text)
17
18     @classmethod
19     def from_json(cls, data):
20         return cls(**data)
21
22     def to_json(self):
23         to_serialize = ['task_id', 'name', 'description']
24         d = {}
25         for attr_name in to_serialize:
26             d[attr_name] = getattr(self, attr_name)
27         return d
```

2. Create a database and create the tables from our models.

I'm using Postgres so I run:

```
$ createdb blogpostcorniceapp
```

and then create a script to initialize the db:

```
1 from sqlalchemy import engine_from_config
2 from blogpostcorniceapp.models import DBSession
3 from pyramid.paster import get_appsettings
4
5 from blogpostcorniceapp.models import Base
6
7 settings = get_appsettings('/home/fran/blogpostcorniceapp/blogpostcorniceapp.ini')
8 engine = engine_from_config(settings, 'sqlalchemy.')
9 DBSession.configure(bind=engine)
10
11 Base.metadata.create_all(engine)
```

and now when I want to wipe the db and start again I can run:

```
$ dropdb blogpostcorniceapp && createdb blogpostcorniceapp && python initialize_db.py
```

3. main()

In `blogpostcorniceapp/init.py` in `main()` we want to add:

```
1 engine = engine_from_config(settings, 'sqlalchemy.')
2 DBSession.configure(bind=engine)
```

which will require these imports:

```
1 from sqlalchemy import engine_from_config
2 from blogpostcorniceapp.models import DBSession
```

and that will create connections to the database for the webapp.

4. blogpostcorniceapp.ini

Lastly we need two bits of configuration to define where the database can be found, and to ask pyramid to manage database transactions for us (so the transaction will be committed after the web request is complete or rolled back on errors without us having to do anything).

in `blogpostcorniceapp.ini` in the `[app:main]` section:

```
1 pyramid.includes = pyramid_tm
2 sqlalchemy.url = postgres://fran@localhost/blogpostcorniceapp
```

and now we can change our views.py to look like this:

```
1 @tasks.get()
2 def get_info(request):
3     """Returns a list of all tasks."""
4     return {'tasks': [task.name for task in DBSession.query(Task)]}
5
6 @tasks.post()
7 def create_task(request):
8     """Adds a new task."""
9     task = request.json
10    num_existing = DBSession.query(Task).filter(Task.name==task['name']).count()
11    if num_existing > 0:
12        raise Exception('That task already exists!')
13    DBSession.add(Task.from_json(task))
```

and running the script to exercise the app we get output:

```
200 {"tasks": []}
200 null
200 {"tasks": ["take_out_the_trash"]}
```

nice!

A little bit more Cornice

Cornice has a better way to model a RESTful API on a collection of models. We should be using a **resource** instead of a service. So we'll rewrite the views.py like this:

```
1 from cornice.resource import resource
2 from cornice.resource import view
3
4 from blogpostcorniceapp.models import Task
5 from blogpostcorniceapp.models import DBSession
6
7 @resource(collection_path='/tasks', path='/tasks/{id}')
8 class TaskResource(object):
9
10    def __init__(self, request):
11        self.request = request
12
13    def collection_get(self):
14        return {'tasks': [task.name for task in DBSession.query(Task)]}
15
16    def collection_post(self):
17        """Adds a new task."""
18        task = self.request.json
19        num_existing = DBSession.query(Task).filter(Task.name==task['name']).count()
20        if num_existing > 0:
21            raise Exception('That task already exists!')
22        DBSession.add(Task.from_json(task))
```

and the exercise script still gets the output we expect:

```
200 {"tasks": []}
200 null
200 {"tasks": ["take_out_the_trash"]}
```

A method on a class decorated with resource named for a HTTP verb (get, put, post, delete) will be exposed (in our case) at /tasks/{id}. A method called collection_put (or any HTTP verb) will be exposed at /tasks. To add a get for a individual task add to TaskResource:

```
1 def get(self):
2     return DBSession.query(Task).get(int(self.request.matchdict['id'])).to_json()
```

A Cornice Validator

Currently if we send two tasks with the same name:

```
1 response = requests.get('http://localhost:6543/tasks')
2 print response.status_code, response.text
3 response = requests.post('http://localhost:6543/tasks', json.dumps(task))
4 print response.status_code, response.text
5 response = requests.post('http://localhost:6543/tasks', json.dumps(task))
6 print response.status_code, response.text
```

we get:

```
200 {"tasks": []}
200 null
500 Internal Server Error
```

The server encountered an unexpected internal server error

(generated by waitress)

The unhandled Exception raised in TaskResource collection_post causes the app to generate a 500 Internal Server Error. Let's make it so we get a useful error message back instead. In views.py, move the code that checks for other tasks with this name to a separate method and decorate collection_post to use that method as a validator:

```
1 @view(validators=('validate_post',))
2 def collection_post(self):
3     """Adds a new task."""
4     task = self.request.json
5     DBSession.add(Task.from_json(task))
6
7     def validate_post(self, request):
8         name = request.json['name']
9         num_existing = DBSession.query(Task).filter(Task.name==name).count()
10        if num_existing > 0:
11            request.errors.add(request.url, 'Non-unique task name.', 'There is already a task with this name.'
```

And now if we try to add the same task twice as above we get:


```
200 {"tasks": []}
200 null
400 {"status": "error", "errors": [{"location": "http://localhost:6543/tasks", "name": "Non-unique task name.", "description":
```

So...

Cornice makes creating a RESTy webapp quite straightforward. And we showed how to integrate it with SQLAlchemy. Hope this helps.







Bio

Latest Posts




Francis Bull

Share this:




Tags: [Cornice](#) [database](#) [pyramid](#) [python](#) [REST](#) [sqlalchemy](#)




This work is licensed under a [Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License](#).

2 Comments



Pyramid Climber on October 28, 2014 at 5:09 pm
very good tutorial, thank you, but this site has endless reloads of app.leadliaison.com, maybe you would like to fix that, it's disturbing.



Brett g Porter on October 29, 2014 at 12:41 pm
Thanks — should be fixed now.

Search

Discussing your project with one of our developers is a great way to begin the process.

Recent Posts

Downloading Client-side Generated Content

Searching App Content with Core Spotlight

Indexing App Content with Core Spotlight

A C++ Class Factory for JUCE

Are You Attending CES 2017?

FOLLOW OUR DEVELOPER BLOG VIA RSS

Downloading Client-side Generated Content

Categories

› A&L Insights

› Articles

› Custom Mobile Applications

› Custom Software Development and Design

› Developer Blog

› Field Solutions

› Giving

› Uncategorized

Tags

2014 review android api apple audio book review C++ cocoa coffeescript Core Data cryptography CSS
custom software development data visualization design development Django encryption Frameworks github HTML html5
infosec internet of things iOS IoT javascript JQuery JUCE Learning Mac OS X mobile music node.js Objective-C
programming python rails ruby security Software development testing twitter ui user interface

Tweets



Art & Logic @artandlogic



ift.tt/2jUFDCo "Downloading Client-side Generated Content" by Christopher Keefer



[Embed](#)

[View on Twitter](#)

[Home](#)

[Our Work](#)

[Software Development](#)

[Software Solutions](#)

[Our Development Process](#)

[A&L Insights](#)

[Developer Blog](#)

[Careers](#)

[About Us](#)

[Contact Us](#)

Celebrating 25 years of custom software development. We have built software for over 900 clients from a diverse set of industries including education, aerospace, music technology, consumer electronics, entertainment, financial services, and many more. Coding the "impossible."™

Tweets by @artandlogic



Art & Logic @artandlogic



ift.tt/2jUFDCo "Downloading Client-side Generated Content" by Christopher Keefer



[Embed](#)

[View on Twitter](#)



2 North Lake Avenue, Suite 1050

Pasadena, CA 91101

626-427-7184

[CONTACT US](#)