CherryPy Homework

Due March 18th, 2016
CSE332 Programming Paradigms, Spring 2016

Overview Use CherryPy to write a RESTful web service for our movie database. The service will allow access to the movies, users, and ratings. It will also allow us to add, modify, and delete entries. This service will replace the one we have provided for you to use with the other assignments so far. We provide unittests for every resource, and you may (should!) use your CherryPy Primer homework as a foundation:

```
$ mkdir my_homework_dir; cd my_homework_dir
$ cp /afs/nd.edu/user37/cmc/Public/cse332_sp16/cherrypy/tests/*.py .
$ ls
test_movies_index.py  test_recommendations_index.py  test_users.py
test_movies.py        test_recommendations.py
test_ratings.py       test_users_index.py
$
```

Interface to Build

Resource	GET	PUT	POST	DELETE
/movies/	List all available movies.		Add a new movie.	Clear movies DB.
/movies/:movie_id	Retrieve movie title and description.	Replace movie, or create it if it does not exist.		Delete the movie.
/users/	List all users.		Add a new user.	Clear users DB.
/users/:user_id	Retrieve user's profile information.	Replace user, or create if does not exist		Delete the user.
/recommendations/				Clear votes DB.
/recommendations/:user_id	Retrieve movie recommendation for a user.	Add a new vote for a given movie.		
/ratings/				
/ratings/:movie_id	Retrieve the average rating for a movie.			
/reset/		Recreates database from .dat files		
/reset/:movie_id		Resets one movie from .dat files		

Discussion Observe the similarity between the interface above and the interface you wrote for the first Python assignment! ☺ Use this similarity to your advantage. You may even consider this assignment to be a web service layer "on top" of the database backend you wrote before. From the CherryPy Primer assignment, you should have all the examples you need.

Poster Images From the PyQt primer assignment, you recall that the JSON response for a movie requests includes a key-value pair that specifies a filename for a movie poster image for most of the movies. The key is `img` and the value is a filename. You can assume, for now, that the client has access to the images; your web service need not send the image file contents (just the name). We provide a map of the poster image files to the movie IDs in `images.dat`. The format should seem familar:

```
$ cp /afs/nd.edu/user37/cmc/Public/cse332_sp16/cherrypy/data/images.dat .
$ head -n 5 images.dat
1::862::/agy8DheVu5zpQFbXfAdvYivF2FU.jpg
2::8844::/vzmL6fP7aPKNKPRTFnZmiUfciyV.jpg
3::15602::/fHHH30JKWzb6A6gnnicY9FzsAWn.jpg
4::31357::/21aTe5Cp6tayodcDrsEoF4gTh7.jpg
5::11862::/e64sOI48hQXyru7naBFyssKFxVd.jpg
$
```

The first item is the ID number of the movie matching the `ml-1m` database (`movies.dat`). The second item is the movie ID from `http://www.themoviedb.org/`, which can safely ignore for now. The third item is the image filename. To get the full path, append the filename to the images directory:

```
/afs/nd.edu/user37/cmc/Public/cse332_sp16/cherrypy/data/images/
```

So for example, Toy Story, which is ID #1, has the filename `agy8DheVu5zpQFbXfAdvYivF2FU.jpg` in the directory above.

You should read in `images.dat` and use it to return the image filename for `GET /movies/` and `GET /movies/:movie_id`. See the `JSON` guide at the end of this document.

```
  output['img'] = self.get_poster_by_mid(movie_id)
...
  def get_poster_by_mid(self, mid):
    if mid in self.posters.keys():
      return self.posters[mid]
    return '/default.jpg'
...
  def load_posters(self, posters_file):
    self.posters = {}
    f = open(posters_file)
    for line in f:
      line = line.rstrip()
      components = line.split("::")
      mid = int(components[0])
      mimg = components[2]
      self.posters[mid] = mimg
  f.close()
```

Reset for Testing You may have noticed a resource called `/reset/`. This is for testing purposes. The controller for it should work just like any other controller. But note that a `PUT` should completely restore the movie database by reloading the `dat` files:

```
def PUT(self):
    output = {'result':'success'}
    # note that you need authentication code here, too
    self.mdb.__init__()
    self.mdb.load_movies('ml-1m/movies.dat')
    self.mdb.load_users('ml-1m/users.dat')
    self.mdb.load_ratings('ml-1m/ratings.dat')

    return json.dumps(output, encoding='latin-1')
```

What's next? You are now well on your way! You may edit your `_movie_database.py` file as much as you like to support this assignment. Probably the only addition you will need to make is a function `get_highest_rated_unvoted_movie(user_id)`. You will need that functionality for the `GET` call to `/recommendations/:user_id`. See below...

So how do we recommend movies, anyway? The movie recommendation algorithm we will use is simple: recommend the highest rated movie that the user has not voted on before. Say we have three movies, `Jurassic Park`, `Star Wars`, and `Jaws`. And say these movies are rated `4.6`, `4.4`, and `4.3`, respectively. If a user `Alice` has not voted on any movies, we would recommend `Jurassic Park` to her, because it is the highest rated. But if a user `Bob` has voted on `Jurassic Park`, we would recommend `Star Wars`, because it is the highest rated movie that `Bob` has not voted on yet.

If you are interested in improving this algorithm, look up "collaborative filtering" for some of the most advanced techniques. There is plenty of related literature:

> Yehuda Koren. 2010. Collaborative filtering with temporal dynamics. Commun. ACM 53, 4 (April 2010), 89-97.
>
> Golbeck, J.; Hendler, J., FilmTrust: movie recommendations using trust in web-based social networks, Consumer Communications and Networking Conference, 2006. CCNC 2006. 3rd IEEE , vol.1, no., pp.282,286, 8-10 Jan. 2006

Kamal Ali and Wijnand van Stam. 2004. TiVo: making show recommendations using a distributed collaborative filtering architecture. In Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining (KDD '04). ACM, New York, NY, USA, 394-401.

And remember that NetFlix paid $1 million for just one algorithm!
`http://www.netflixprize.com/community/viewtopic.php?id=1537`

Grading We will grade your assignment based on the unit test above. If your code passes all tests (with the usual caveat that we will inspect it to ensure no hard-coding ☺), you will receive full credit. If a test fails, we will assign partial credit based on the severity of the programming error. Code that is illegible, does not follow the assignment parameters, or does not run will receive a zero. The assignment is out of 60 points.

Learning Goals The objective of this assignment is to practice building RESTful web services in CherryPy. These services are used as a basis for a huge number of web and mobile applications. The experience you gain with this assignment will give you valuable and marketable skills for the future of development, considering the growth of web and mobile apps. At the same time, web services are commonly used in an area called "middleware", in between the backend (such as your movie database) and the frontend (such as your PyQt program).

Turn-in Package your working program in a `tar.gz` or `zip` file and turn in this file to `prog.paradigms.secN.sp16@gmail.com`, where `N` is your section number. Be sure to include your name.

Command	Resource	Input Example	Output Example
GET	/movies/		{”movies”: [{”genres”: ”Drama—Sci-Fi”, ”title”: ”Twelve Monkeys (1995)”, ”result”: ”success”, ”id”: 32, “img”:“poster.jpg”},...], ”result”: ”success”}
POST	/movies/	{”genres”: ”Animation”, ”title”: “The Brain”, ”apikey”:“AaD72Feb3”}	{”result”: ”success”, ”id”: 3953}
DELETE	/movies/	{“apikey”:“AaD72Feb3”}	{”result”:”success”}
GET	/movies/:movie_id		{”genres”: ”Drama—Sci-Fi”, ”title”: ”Twelve Monkeys (1995)”, ”result”: ”success”, ”id”: 32, “img”:“poster.jpg”}
PUT	/movies/:movie_id	{”genres”: ”Animation”, ”title”: “The Brain”, “apikey”:“AaD72Feb3”}	{”result”: ”success”}
DELETE	/movies/:movie_id	{“apikey”:“AaD72Feb3”}	{”result”: ”success”}
GET	/users/		{”result”: ”success”, ”users”: [{”zipcode”: ”48067”, ”age”: 1, ”gender”: ”F”, ”id”: 1, ”occupation”: 10},...]}
POST	/users/	{”zipcode”: ”48067”, ”age”: 1, ”gender”: ”F”, ”occupation”: 10, “apikey”:“AaD72Feb3”}	{”result”:”success”, ”id”: 1}
DELETE	/users/	{“apikey”:“AaD72Feb3”}	{”result”:”success”}
GET	/users/:user_id		{”gender”: ”M”, ”age”: 25, ”zipcode”: ”55455”, ”result”: ”success”, ”id”: ”5”, ”occupation”: 20}
PUT	/users/:user_id	{”gender”: ”M”, ”age”: 28, ”zipcode”: ”46545”, ”occupation”: 15, “apikey”:“AaD72Feb3”}	{”result”:”success”}
DELETE	/users/:user_id	{“apikey”:“AaD72Feb3”}	{”result”:”success”}
DELETE	/recommendations/	{“apikey”:“AaD72Feb3”}	{”result”:”success”}
GET	/recommendations/:user_id		{”movie_id”: 989, ”result”: ”success”}
PUT	/recommendations/:user_id	{”movie_id”:”32”, ”rating”:”5”, “apikey”:“AaD72Feb3”}	{”result”: ”success”}
GET	/ratings/:movie_id		{”rating”: 3.9463931171409663, ”movie_id”: 32, ”result”: ”success”}
PUT	/reset/	{“apikey”:“AaD72Feb3”}	{”result”: ”success”}
PUT	/reset/:movie_id	{“apikey”:“AaD72Feb3”}	{”result”: ”success”}