Pythonidae

December 12

2014

The YAAS web application implements an web-based auction site described in the following use cases below:UC1 Create an user account UC2 Edit user account information UC3 Create a new auction UC4 Edit and existing auction UC5 Browse and search auctions UC6 Bid UC7 Ban an auction UC8 Resolve an auction UC9 Support for multiple languages UC10 Support Multiple Concurrent Sessions.

Dawit Nida
www.dawitnida.com

YAAS: Yet another Auction Site

YAAS is a web application and a web service to create and participate in auctions, e-commerce auction-based website for selling and buying goods and services. It is an online marketplace practically anything is sold virtually from anywhere.

1. The following requirements are implemented
    - UC1 create user account
    - UC3 create new auction
    - TR2.1 automated test for UC3
    - TR1 DB fixture and data generation program
    - UC5 browse and search
    - WS1 browse and search API for web service
    - UC7 ban auction
    - UC4 edit auction
    - UC2 edit user account info
    - UC8 resolve auction
    - UC9 support for multiple languages
    - UC6 Bid
    - Optional features: Soft deadlines
    - WS2 Bidding API for web service
    - TR2.2 Automated test for UC6 Bid
    - UC10 Support for multiple concurrent sessions
    - TR2.3 Automated test for testing concurrency when bidding

2. Django verion : 1.7

   Python version: 2.7.8

And all the requirement applications which are implemented within this project are found in *requirement.txt* file.

3. Authentication to YAAS admin

   Username: admin

   Password: admin
4. How does your application implement session management?

The session management is configured using the django middle wares, and installing the session app. Each user will have session managed inside the django session table, with session key, session data and expire date. And also on the browser level, session will expire and even if the user leaves the yaas website for a while (3600 seconds), he/she is obliged to login back again. Setting the session inside *settings.py*

```
INSTALLED_APPS = ('django.contrib.sessions',)
MIDDLEWARE_CLASSES = (
'django.contrib.auth.middleware.SessionAuthenticationMiddleware',
            )
# Clear session when the browser is closed
SESSION_EXPIRE_AT_BROWSER_CLOSE = True

# Session Expire in 3600 seconds and requires re-login by the member user
SESSION_COOKIE_AGE = 3600
```

5. I used session to store the data and pre save the data before confirmation. And if the user confirm, stored data will be retrieved from the session with a key and new product and new auction object will be created and inserted into the database. Below is a snippet.

```
if "_new_product" in request.session:
      product = request.session.get("_new_product")
…………..
del request.session['_new_product'].
```

Then the session is flushed. If the user submit no or did not respond to the confirmation, then nothing will be saved and committed to the database.

6. YAAS will automatically resolve auctions which the deadline meets with the user inserted end date. First I implemented with django-cron and it needed another crontab implementation from outside of the application. So I used django-celery and python celery to satisfy the requirement. Both applications are installed and configured in the *setting.py* file and *manage.py syncdb*, then *tasks.py* file is created, where the periodic task decorator. decorator used: @periodic_task(run_every=crontab(hour="*", minute="*", day_of_week="*"))

Then *run_crontask* is registered in the database of celery. The real work is done when the custom command calls *yaas.crontask.ResolveAuction* class from the *django-cron.* I used this since I implemented django-cron, but the periodic decorator will register any task workers method in *tasks.py* file thus the real job is done in

ResolveAuction class do() method. This function filters out auctions which have *status 1*, and then if the end time is less or equal to *timezone.now()* then the auction will be resolved. If that specific auction has no bidders, then it will be due and email will be sent to the seller, auction state will be changed to *Due*. Otherwise it will be *Adjudicated* and winners and all bidders, owner will be notified by email (all emails are saved under email folder of the yaas application. In this case auction status will be changed to *4*. To run this automated implementation, the server must be running and then celery beat will be fired, then celery worked will be opened from the second terminal which does the real task. In case, django-celery has PID key error, then clear the pid from the celery PID file and rerun again

The following are configurations and outputs from the automated *run_crontask* worker

The task requires django setup to work and also necessary imports from celery.

```
import django
django.setup()
```

# Celery settings

```
INSTALLED APP:( 'djcelery',
                'kombu.transport.django',)

import djcelery
djcelery.setup_loader()

BROKER_URL = 'django://'
CELERY_IMPORTS = ("yaas.tasks",)
CELERYBEAT_SCHEDULER = 'djcelery.schedulers.DatabaseScheduler'
CELERY_TIMEZONE = 'Europe/Helsinki'
CELERY_ACCEPT_CONTENT = ['json']
CELERY_TASK_SERIALIZER = 'json'
CELERY_RESULT_SERIALIZER = 'json'
CELERY_TASK_RESULT_EXPIRES = 1800
```

```
X:\Github\Dewas\Pythonidae>python manage.py celery worker --loglevel=info
 -------------- celery@NDavid-Pro v3.1.16 (Cipater)
---- **** ----- Windows-7-6.1.7601-SP1
- ** ---------- [config]
- ** ---------- .> app:         default:0x3a33470 (djcelery.loaders.DjangoLoader)
- ** ---------- .> transport:   django://localhost//
- ** ---------- .> results:     database
- *** --- * --- .> concurrency: 4 (prefork)
--- ***** ----- [queues]
 -------------- .> celery           exchange=celery(direct) key=celery
[tasks]
  . yaas.tasks.run_crontask

X:\Github\Dewas\Pythonidae>python manage.py celery beat
celery beat v3.1.16 (Cipater) is starting.
Configuration ->
    . broker -> django://localhost//
    . loader -> djcelery.loaders.DjangoLoader
    . scheduler -> djcelery.schedulers.DatabaseScheduler
    . logfile -> [stderr]@%INFO
    . maxinterval -> now (0s)
[2014-11-02 12:23:47,184: INFO/MainProcess] beat: Starting...
[2014-11-02 12:23:47,187: INFO/MainProcess] Writing entries...
[2014-11-02 12:23:47,673: INFO/MainProcess] DatabaseScheduler: Schedule changed.
[2014-11-02 12:23:47,674: INFO/MainProcess] Writing entries...
[2014-11-02 12:23:47,710: INFO/MainProcess] Scheduler: Sending due task
yaas.tasks.run_crontask (yaas.tasks.run_crontask)
[2014-11-02 12:23:47,782: INFO/MainProcess] Writing entries...
[2014-11-02 12:23:47,806: INFO/MainProcess] Scheduler: Sending due task
celery.backend_cleanup (celery.backend_cleanup)
[2014-11-02 12:24:00,003: INFO/MainProcess] Scheduler: Sending due task
yaas.tasks.run_crontask (yaas.tasks.run_crontask)
```

7. Concurenccy

For yaas application, django concurrency package. *Django-concurrency* is an optimistic lock implementation for Django. This uses a database for persistence and optimistic locking to detect and resolve concurrent edits and saves. The edited version of the auction or product is stored in a hidden field of the form. Each model has its own Version Field that returns a “unique” version number for the record. The version number is produced using time.time() * 1000000, to get the benefits of microsecond if the system clock provides them preventing users from doing concurrent editing in Django both from UI and from a django command.

INSTALLED APP:(concurrency,)

8. REST API

The rest full API is implemented by using django-restframework==2.4.3. And header authentication used was done using *'rest_framework.authtoken'* and below are results from the API. All created users need *BasicAuthentication and TokenAuthentication* to login to the API. Users will use Token Authentication and send request to yaas web service in *json* format. For existing users Token was created from python shell but after implementation of Rest framework, every registered user will have generated *Token* by default when registered to YAAS. This is the class implementation which creates *Token* for the existing users.

```
class CreateUserToken:
    def create_token(self):
        for user in User.objects.all():
            Token.objects.get_or_create(user=user)
```

setting.py configuration

```
'rest_framework',
'rest_framework.authtoken',
REST_FRAMEWORK = {
   'DEFAULT_AUTHENTICATION_CLASSES': (
      'rest_framework.authentication.TokenAuthentication',
      'rest_framework.authentication.SessionAuthentication',  # for web browsable API
   ),
   'PAGINATE_BY': 20,
   #'DEFAULT_RENDERER_CLASSES': ('rest_framework.renderers.JSONRenderer'),
   #'DEFAULT_PARSER_CLASSES': ('rest_framework.parsers.JSONResponse'),
}
```

Some tests from the rest API.

```
~$ curl -X POST http://127.0.0.1:8000/api/bid/<auc-pk>/ -H
'Authorization: Token <token>' "Accept: application/json" -H
"Content-type: application/json"
--data '{"bid_amount": <bid amount>}' -v
```

Users can search auction by title, by sending a key word attached with the header. If any auction exists, the response content is sent in *json* format, otherwise *json* formatted error is sent back to with Header status code:

```
~$ curl -X GET http://127.0.0.1:8000/api/search/pro/ -H "Accept:
application/json" -H "Content-type: application/json"
              {
                "count":1,
                "next":null,
                "previous":null,
                "results":[{
                    "id":112,
                    "title":"admin prod",
                    "current_price":"89",
                    "updated_time":"2014-11-02T01:41:36.738Z",
                    "end_time":"2014-11-12T01:36:36Z",
                    "product":126,
                    "status":1,
                    "version":1414891419716400
                  }]
              }

~$ curl -X POST http://127.0.0.1:8000/api/bid/51/ -H "Accept:
application/json" -H "Content-type: application/json" --data
'{"bid_amount": 99}'
              {
                "detail":"Authentication credentials were not provided."
              }
~$  curl -X POST http://127.0.0.1:8000/api/auth-user/ -H "Accept:
application/json" -H "Content-type: application/json"
              {
                "username":["This field is required."],
                "password":["This field is required."]
              }
```

```
~$  curl -X POST http://127.0.0.1:8000/api/auth-user/ -H "Accept:
application/json" -H "Content-type: application/json" --data
'{"username":"google", "password":"goo"}'
          {
            "token":"0f395a9b4cb018fa975efc9b64f846e7509a59f8"
          }
~$ curl -X POST http://127.0.0.1:8000/api/bid/11112/ -H 'Authorization:
Token 0f395a9b4cb018fa975efc9b64f846e7509a59f8' "Accept:
application/json" -H "Content-type: application/json" --data
'{"bid_amount": 89}'
          {
            "non_auction_error":"Auction is not found."
          }
~$ curl -X GET http://127.0.0.1:8000/api/auc/112/ -H "Accept:
application/json" -H "Content-type: application/json"
          {
            "id":112,
            "title":"admin prod",
            "current_price":"89",
            "updated_time":"2014-11-02T01:41:36.738Z",
            "end_time":"2014-11-12T01:36:36Z",
            "product":126,
            "status":1,
            "version":1414891419716400
          }
~$ curl -X POST http://127.0.0.1:8000/api/bid/112/ -H
'Authorization: Token 0f395a9b4cb018fa975efc9b64f846e7509a59f8'
"Accept: application/json" -H "Content-type: application/json" --
data '{"bid_amount": 89}'
          {
            "duplicate_bidder_error":"No need to bid. You are winning."
          }
~$ curl -X GET http://127.0.0.1:8000/api/auc/51/ -H "Accept:
application/json" -H "Content-type: application/json"
          {
            "id":51,
            "title":"blendeo",
            "current_price":"14.01",
            "updated_time":"2014-11-02T10:38:32.804Z",
            "end_time":"2014-11-12T23:00:54Z",
            "product":66,
            "status":1,
            "version":0
          }
~$ curl -X POST http://127.0.0.1:8000/api/bid/51/ -H 'Authorization:
Token 0f395a9b4cb018fa975efc9b64f846e7509a59f8' "Accept:
application/json" -H "Content-type: application/json" --data
'{"bid_amount": 90}'
          {
            "same_user_error":"You can not bid on your item."
          }
```

```
~$ curl -X POST http://127.0.0.1:8000/api/bid/108/ -H
'Authorization: Token 0f395a9b4cb018fa975efc9b64f846e7509a59f8'
"Accept: application/json" -H "Content-type: application/json" --
data '{"bid_amount": 78}'
          {
            "bid_amount_error":["Invalid bid amount."]
          }
~$ curl -X POST http://127.0.0.1:8000/api/bid/108/ -H
'Authorization: Token 0f395a9b4cb018fa975efc9b64f846e7509a59f8'
"Accept: application/json" -H "Content-type: application/json" --
data '{"bid_amount": 78.02}' -v
          * Connected to 127.0.0.1 (127.0.0.1) port 8000 (#0)
          > POST /api/bid/108/ HTTP/1.1
          > User-Agent: curl/7.38.0
          > Host: 127.0.0.1:8000
          > Accept: */*
          > Authorization: Token 0f395a9b4cb018fa975efc9b64f846e7509a59f8
          > Content-type: application/json
          > Content-Length: 21
          * HTTP 1.0, assume close after body
          < HTTP/1.0 201 CREATED
          < Date: Sun, 02 Nov 2014 12:36:58 GMT
          < Server: WSGIServer/0.1 Python/2.7.8
          < Vary: Accept, Accept-Language, Cookie
          < X-Frame-Options: SAMEORIGIN
          < Content-Type: application/json
          < Content-Language: en
          < Allow: POST, GET, OPTIONS, DELETE
          * Closing connection 0
          * The cache now contains 0 members
          {
            "id":560,
            "unique_bidder":811,
            "auc":108,
            "bid_amount":"78.02",
            "bid_time":"2014-11-02T12:36:58.941Z",
            "version":1414930872263400
          }
```

9. There are three major tests in yaas project.

For UC3: The *add_product* method is tested using class *AuctionCreateTest(TestCase):*by reversing the url. The user will be tested if he is authenticated and get 200 response code from the template. Then after filling the form, which is also asserted by the test function, the user will be asked for confirmation. The test suit also checks if the data is saved in the session key. Then after confirming the auction to be created, user is redirected to index, or if not to other url. Here both cases are asserted. After that both Product and Auction object creation are asserted. This test case can easily be run using the following custom command.

```
X:\Github\Dewas\Pythonidae>manage.py testUC3
            Ran 1 test in 2.431s        OK
            Destroying test database for alias 'default'...
```

Again for TR2 Automated Functional Tests is implemented on for UC, creating bid using class *BidTest(TestCase):* This test suit asserts if, the placed bid is correct and user is redirected to the appropriate url.

```
X:\Github\Dewas\Pythonidae>manage.py testUC6
            Ran 3 tests in 4.862s       OK
            Destroying test database for alias 'default'...
```

On the last test, *ConcurrencyTestMixin* and *RecordModifiedError* where imported from *django-concurrency* package and the test basically checks if concurrency raise the expected exceptions or not. Class *AuctionConcurrencyTest(ConcurrencyTestMixin, TestCase):* and class *BidConcurrencyTest(ConcurrencyTestMixin, TestCase):* This assert that simultaneous task on the same auction will raise exception So YAAS application should handle this exception <*django-concurrency*> solves this problem by creating *IntegerField* for each model *Version Field* that returns a 'unique' version number for the record.

```
X:\Github\Dewas\Pythonidae>manage.py testUC10
            Ran 1 test in 0.086s        OK
            Ran 1 test in 0.038s        OK
            Destroying test database for alias 'default'...
```

10. The language switching is basically done using the django translation method, only

in this case, the form action calls *switch_language(request):* method from the view. This will get the request and then will save it for future return. If to the user submits the change language button, then the language code will be checked from the settings and then it will be written on the session. The language also implements using cookies to manage the languages. Inside the base template, the form will display the language selections.

The following are the configuration inside *settings.py*

```
LOCALE_PATHS = ('locale',)
ugettext = lambda s: s
LANGUAGES = (
   ('en', ugettext('English')),
   ('fi', ugettext('Suomi')),
   ('sv', ugettext('Svenska')),
)
```

NOTES:

Fixitures can be easily loaded to the yaas database using the following command.

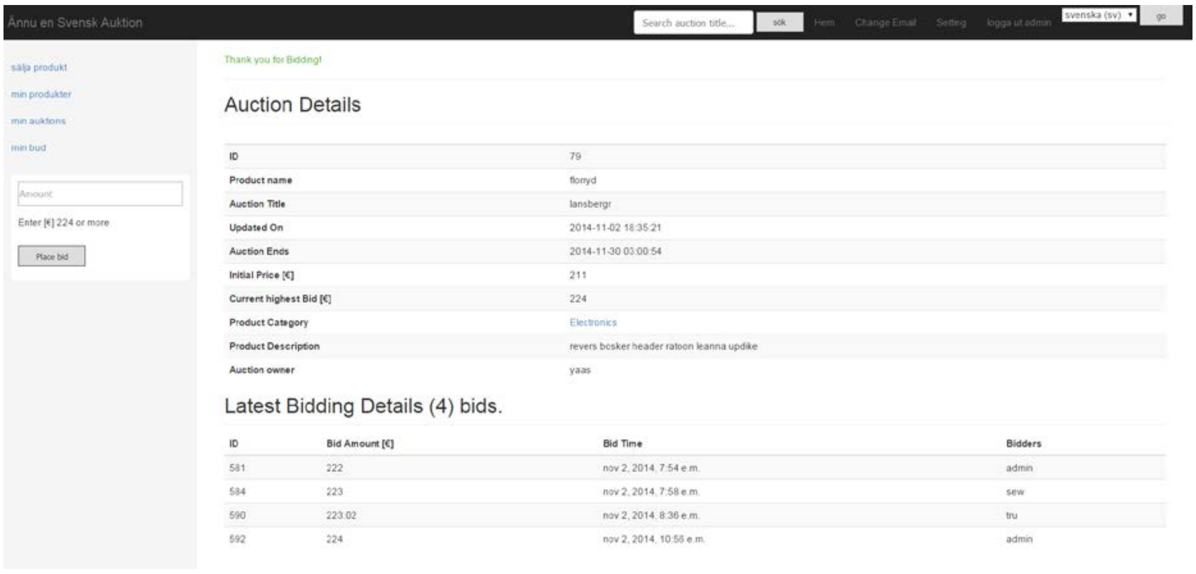
```
X:\Github\Dewas\Pythonidae>manage.py autofixDB
```

This will run the following commands from the custom management:

```
management.call_command('loaddata', 'fresh_users.json')
management.call_command('loaddata', 'master_yaas_db.json')
```

Emails are saved inside *email* folder: the following is the configuration inside *settings.py*.

```
EMAIL_BACKEND = 'django.core.mail.backends.filebased.EmailBackend'
EMAIL_FILE_PATH = BASE_DIR + '/yaas/emails/messages/'
```



Thank you for Bidding!

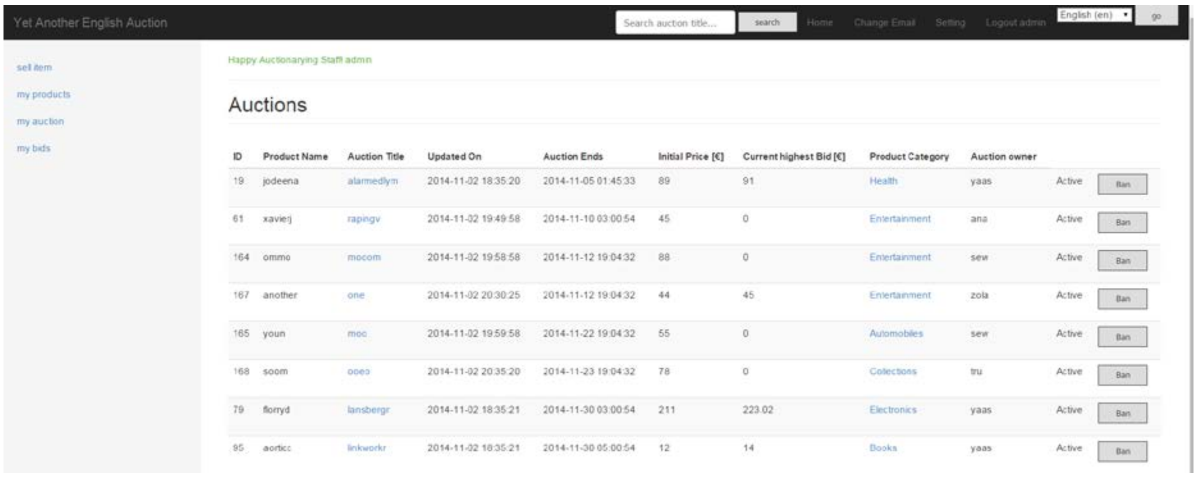
Auction Details

ID	79
Product name	flonyd
Auction Title	lansbergr
Updated On	2014-11-02 18:35:21
Auction Ends	2014-11-30 03:00:54
Initial Price [€]	211
Current highest Bid [€]	224
Product Category	Electronics
Product Description	revers bosker header ratoon leanna updike
Auction owner	yaas

Latest Bidding Details (4) bids.

ID	Bid Amount [€]	Bid Time	Bidders
581	222	nov 2, 2014, 7:54 e.m.	admin
584	223	nov 2, 2014, 7:58 e.m.	sew
590	223.02	nov 2, 2014, 8:36 e.m.	tru
592	224	nov 2, 2014, 10:56 e.m.	admin

Fig 1. Logged in user

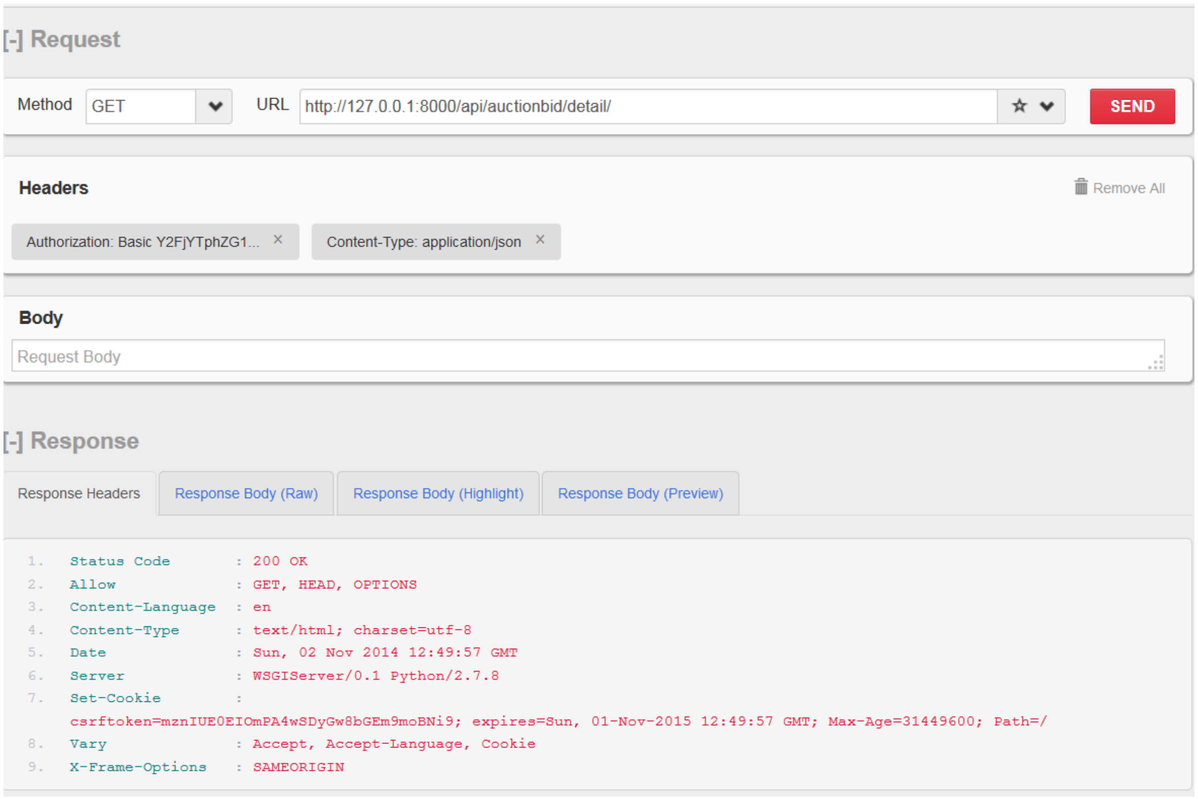


Happy Auctionarying Staff! admin

Auctions

ID	Product Name	Auction Title	Updated On	Auction Ends	Initial Price [€]	Current highest Bid [€]	Product Category	Auction owner		
19	jodeena	alarmedlym	2014-11-02 18:35:20	2014-11-05 01:45:33	89	91	Health	yaas	Active	Ban
61	xavierj	rapingv	2014-11-02 19:49:58	2014-11-10 03:00:54	45	0	Entertainment	ana	Active	Ban
164	ommo	mocom	2014-11-02 19:58:58	2014-11-12 19:04:32	88	0	Entertainment	sew	Active	Ban
167	another	one	2014-11-02 20:30:25	2014-11-12 19:04:32	44	45	Entertainment	zola	Active	Ban
165	youn	moc	2014-11-02 19:59:58	2014-11-22 19:04:32	55	0	Automobiles	sew	Active	Ban
168	soom	ooeo	2014-11-02 20:35:20	2014-11-23 19:04:32	78	0	Collections	tru	Active	Ban
79	florryd	lansbergr	2014-11-02 18:35:21	2014-11-30 03:00:54	211	223.02	Electronics	yaas	Active	Ban
95	aorticc	linkworkr	2014-11-02 18:35:21	2014-11-30 05:00:54	12	14	Books	yaas	Active	Ban

Fig 2. Staff/admin logged in



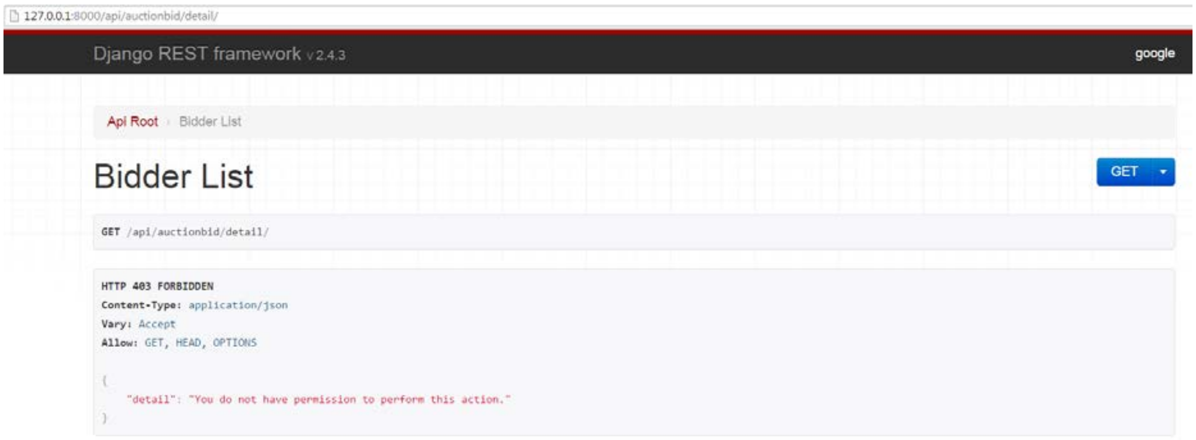
http://127.0.0.1:8000/api/auctionbid/detail/

Authorization: Basic Y2FjYTphZG1...

Content-Type: application/json

```
Status Code       : 200 OK
Allow             : GET, HEAD, OPTIONS
Content-Language  : en
Content-Type      : text/html; charset=utf-8
Date              : Sun, 02 Nov 2014 12:49:57 GMT
Server            : WSGIServer/0.1 Python/2.7.8
Set-Cookie        :
csrftoken=mznIUE0EIOmPA4wSDyGw8bGEm9moBNi9; expires=Sun, 01-Nov-2015 12:49:57 GMT; Max-Age=31449600; Path=/
Vary              : Accept, Accept-Language, Cookie
X-Frame-Options   : SAMEORIGIN
```

Fig 3. Auction/detail api response



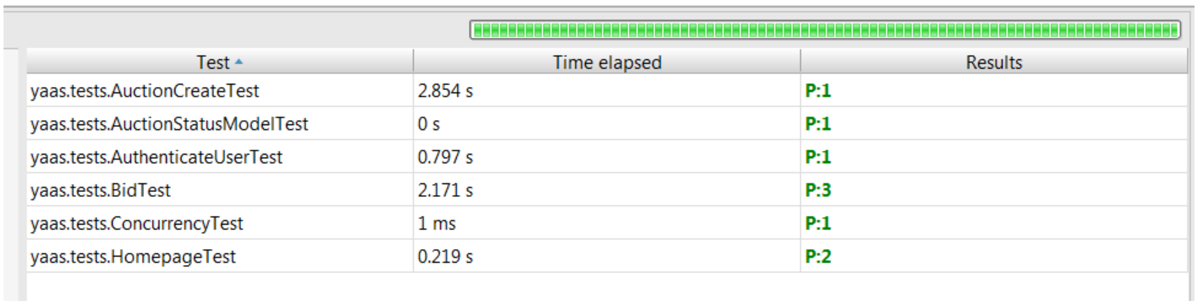
127.0.0.1:8000/api/auctionbid/detail/

Bidder List

```
GET /api/auctionbid/detail/
```

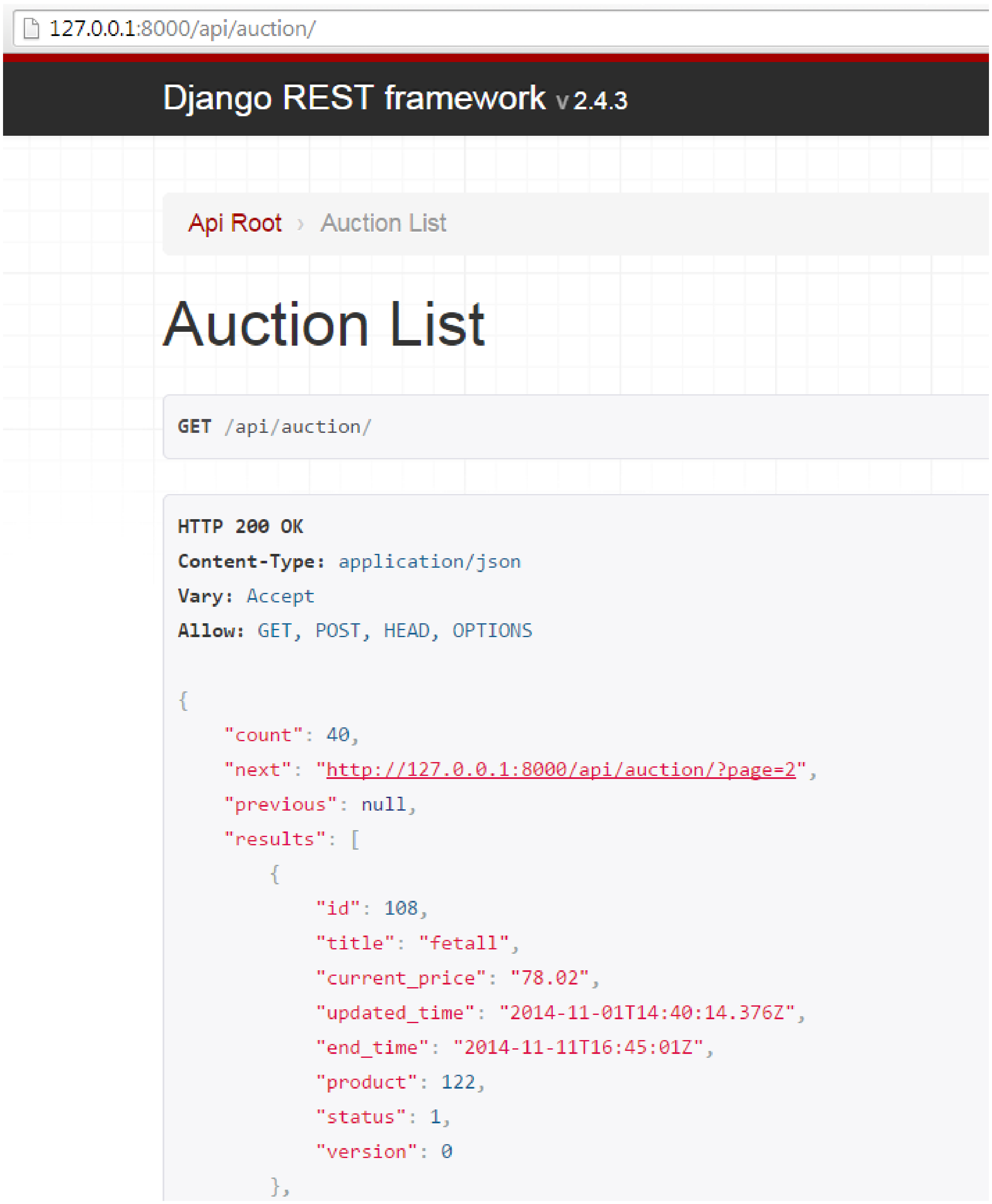
```
HTTP 403 FORBIDDEN
Content-Type: application/json
Vary: Accept
Allow: GET, HEAD, OPTIONS

{
    "detail": "You do not have permission to perform this action."
}
```

Fig 4. Bid/api authentication fail



Test	Time elapsed	Results
yaas.tests.AuctionCreateTest	2.854 s	P:1
yaas.tests.AuctionStatusModelTest	0 s	P:1
yaas.tests.AuthenticateUserTest	0.797 s	P:1
yaas.tests.BidTest	2.171 s	P:3
yaas.tests.ConcurrencyTest	1 ms	P:1
yaas.tests.HomepageTest	0.219 s	P:2

Fig 5. Test results



Auction List

```
GET /api/auction/
```

```
HTTP 200 OK
Content-Type: application/json
Vary: Accept
Allow: GET, POST, HEAD, OPTIONS

{
    "count": 40,
    "next": "http://127.0.0.1:8000/api/auction/?page=2",
    "previous": null,
    "results": [
        {
            "id": 108,
            "title": "fetall",
            "current_price": "78.02",
            "updated_time": "2014-11-01T14:40:14.376Z",
            "end_time": "2014-11-11T16:45:01Z",
            "product": 122,
            "status": 1,
            "version": 0
        },
```

Fig 6. Browsable api for auctions