Development of server-side web services

REPORT

POINT THOMAS - 74759
Year 2018-2019 - Period 1

Åbo Akademi

1. Git Hub

You will be able to find my code here : https://github.com/ThomasPoint/YAAS.git

2. Deployment of the application

The app is running here : https://thomaspoint.pythonanywhere.com/

3. Implemented requirements

- UC1: create user
- UC2: edit user
- UC3: create auction
- UC4: edit auction description
- UC6: bid
- UC5: Browse & Search
- UC7: ban auction
- UC8: resolve auction
- UC9: language switching
- UC11: currency exchange
- WS1: Browse & Search API
- WS2 : Bid API
- OP1: send seller auction link
- TR1: data generation program

For the **OP1**, I chose to create a function inside views.py that allows any user with the link to update the description of the auction.

I create the link by finding the domain name of the website, giving the necessary informations to reach the view and the auction id in order to update the right auction. If the auction is not active anymore when the seller wants to update it I redirect him to an html page saying that the resource is not available.

```
def update_auction_without_login(request, auction_id):
    auction = get_object_or_404(Auction, pk=auction_id)
    if auction.state == 'ACTIVE':
        auction_form = AuctionUpdateForm(instance=auction)
    else:
        return redirect('auction_not_active')
    return render(request, 'yaasapp/update_auction.html', {
        'auction_form': auction_form
    })
```

4. Python packages

Python packages	Version
amqp	2.3.2
billiard	3.5.0.4
celery	4.2.1
Django	2.1
django-filter	2.0.0
kombu	4.2.1
Markdown	3.0.1
djangorestframework	3.8.2
pytz	2018.5
vine	1.1.4
virtualenv	16.0.0
requests	2.19.1

5. Admin

The username for the admin has been set to ***admin*** and the password is ***admin2018***.

6. Session management implementation

The session management for this application is implemented using django built-in mechanism (django.contrib.auth.urls). In order to make it work you have to create a login.html file. If the user click on the button login he will reach the login form. If he is not

registered, he can click on a link that will redirect him to the creation profile page.

7. Confirmation form - creation auction

I created an Auction form thanks to django forms mechanism. When the user want to create an auction, we get all the informations provided and we create another form, i.e the auction confirmation form, composed of a field yes/no and all the other fields of the auction form, but this time the fields are hidden (we do not want to alter them at this point).
If the user chooses YES, the server processes the request and create an auction.
if the user chooses NO, the server does not create the auction.

8. Resolution of Bids

To resolve bids I chose to use the Celery python package. In brief it's a tool that allows to automate some tasks, they can be executed synchronously (the program wait for the task to finish) or asynchronously (run as background task). You can find more information here : http://www.celeryproject.org/.

To install it you just have to run the following command :

pip install Celery

In order to execute the task generated by Celery, you will have to install RabbitMQ (more info here : https://www.rabbitmq.com/)

To install it run : **sudo apt-get install rabbitmq-server**

Then run : **sudo systemctl enable rabbitmq-server** to enable the service

Then run : **sudo systemctl <start|stop> rabbitmq-server** to start or stop the service

Once these steps are done, go where the YAAS project is located and run :

celery -A YAAS worker -l info -B.

The -B option will use the broker rabbitmq-server we installed.

That's it for the installation of the program.

Unfortunately it's not possible to configure celery in pythonanywhere.

Then you have to configure the scheduling of the resolution task. My task will be executed each day at 00:01.

It will loop through all the auctions and check if the deadline has been reached, if yes we change the state of the auction to ADJUCATED and we send notification emails to the seller

of the auction to notify him that an auction has been adjucated, another mail to the winner to say that he has won this auction and finally a mail to all the bidders to inform them that they have lost the auction.

9. Concurrency issues

I have not implemented this requirement

10. REST API

a. WS1 : Browse and search API.

I am able to make a search by id for an auction and to make a search by title.

To reach these endpoints you will have to follow the following links :

https://thomaspoint.pythonanywhere.com/yaasapp/api/auctions/310

result :


```
{
   "pk": 310,
   "seller": {
      "username": "tpoint0",
      "first_name": "Thomas0",
      "last_name": "Point0",
      "email": "tpoint0@yopmail.com"
   },
   "title": "Auction 2",
   "description": "I sell 2 items",
   "min_price": 3.02,
   "post_date": "2018-10-10",
   "deadline": "2018-10-13",
   "state": "ACTIVE"
}
```

https://thomaspoint.pythonanywhere.com/yaasapp/api/auctions/?title=Auction 0

result :


```
[
   {
      "pk": 308,
      "seller": {
         "username": "tpoint0",
         "first_name": "Thomas0",
         "last_name": "Point0",
         "email": "tpoint0@yopmail.com"
      },
      "title": "Auction 0",
      "description": "I sell 0 items 1",
      "min_price": 1.0,
      "post_date": "2018-10-10",
      "deadline": "2018-10-13",
      "state": "ACTIVE"
   }
]
```


b. WS2 : Bid API

To use is you will have to go under the namespace /yaasapp/api/bid and give 2 arguments *auction_id* and *bid_price*.

We use Basic Authentication to authenticate a user and only a authenticated user will be able to bid on an auction.

Here are a list of different results that can occur using this API :

- Everything goes fine


```
{
        "id": 85,
        "date": "2018-10-13T10:18:51.717203Z",
        "value": "63.00",
        "bidder": 1211,
        "auction": 308
}
```

- Problem on the given auction_id : no id provided

```
{
        "error": "Please add a auction id"
}
```

- Problem on the bid_price :

```
{
        "error": "Please add a bid price"
}
```

- If the provided bid_price is not greater than the last bid

```
{
        "error": "The value of the bid should be greater than 63.00"
}
```

- If the seller try to bid on his own auction

```
{
        "error": "You cannot bid because you are the owner of the auction !"
}
```

- If the latest bidder try to bid again

```
{
        "error": "You are the current winner of the auction, you cannot bid"
}
```

- Invalid username/password

```
{
        "detail": "Invalid username/password."
}
```

11. Functional tests

I have not implemented this requirement.

12. Language switching mechanism

For the language switching mechanism I used what is proposed by django in the django.util library (ugettext). Ugettext is used in the views.py file and for the templates I used the {% trans “blablabla” %} bloc.

Furthermore you have to run some commands to tell django which languages will be available :

- *manage.py makemessages -l fr* in order to say that I want some french in my app
- *manage.py compilemessages* in order to make them available

I have also set up a function in my views.py. It handles when a user wants to change the language. This function set up the session so that the language will be the same throughout the user session.

```
def change_language(request, lang_code):
  translation.activate(lang_code)
  request.session[translation.LANGUAGE_SESSION_KEY] = lang_code
  return redirect('home')
```

13. Data generation program

The data generation program is located in the views.py file in a function called *generatedata*.

It creates 50 users, 50 auctions and bid on some auctions.

To generate date from the web browser you just have to follow this link :

https://thomaspoint.pythonanywhere.com/yaasapp/generatedata