Development of server-side web service

Name: **Uchenna Jigo**

Student number: 75005

October 28, 2018

Link to github:

https://github.com/UchennaJ/git-remote-add-origin-https-github.com-it-teaching-abo-akademi-web-services-2018-assignment1

Link to pythonanywhere:

http://uchenna.pythonanywhere.com/YAAS/

1. List of implemented requirements
    - UC1 create user account
    - UC2 edit user account information
    - UC3 create new auction
    - UC4 edit the decription of an auction
    - UC5 browse and search auction
    - UC6 bid (with soft deadlines)
    - UC7 ban an auction
    - UC8 resolve auction
    - UC9 support for multiple language
    - UC10 support multiple concurrent sessions
    - RESTFul web service
    - Testing
    - Test data generation program
    - Deployment

2. Python packages used:
    - Python 3.7
    - Django 2.1.1
    - Django-cron 1.4.1
    - Django rest frame work 3.9
    - Django money rates
    - Gettext- 0.19
    - Database: SQLite3

3. The admin login was created, and the login information are:
    Username: Admin
    Password: Admin 2018

    The data generation program generates 50 users with username and password.

    The data generation program can be accessed via the link

    YAAS/generatedata/

4. Implementing session management
    - The session management was implemented in the language changing which saves the users choice. It uses Django's built in session.

```
request.session[translation.LANGUAGE_SESSION_KEY] = lang_code
```

5. Confirmation form implementation

    The confirmation form was implemented in such a way that when a user creates an auction, if the auction is valid, a confirmation is sent in form of a html template. The auction is created when the user chooses yes. However, if the user selects 'no', the page will be redirected showing the auction was not created.

```
def confirmation(request):
    option = request.POST.get('option', '')
    if option == 'Yes':
        # Confirmation was 'Yes', save auction, send email and redirect to front
page.
        form = CreateAuctionForm(request.POST)
        if form.is_valid():
            auction = Auction()
            auction.title = form.cleaned_data["title"]
            auction.seller = request.user
            auction.description = form.cleaned_data["description"]
            auction.end_date = form.cleaned_data["end_date"]
            auction.minimum_price = form.cleaned_data["minimum_price"]
            auction.save()
            message = "This is a confirmation message that the following auction
has been created: \n\n"
            message += auction.information()
            send_mail("Auction created", message, "noreply@YAAS.com",
[auction.seller.email], fail_silently=False)
            return HttpResponseRedirect('/YAAS/')

    template = "message.html"
    context = {"message": _("Auction was not created")}
    # Auction was not created
    return render(request, template, context)
```

6. Resolving bids

    Django cron job was used in resolving bids. The cron job can be initiated on the server using cron and the command can be run using 'python manage.py runcrons.'

```
class ResolveAuctions(CronJobBase):
    RUN_EVERY_MINS = 1
    RETRY_AFTER_FAILURE_MINS = 1

    schedule =
Schedule(run_every_mins=RUN_EVERY_MINS,retry_after_failure_mins=RETRY_AFTER_FAILURE
_MINS)
    code = 'YAAS.ResolveAuctions'    # a unique code

    def do(self):
        print("Resolving auctions")
        auctions = Auction.objects.filter(end_date__lte=timezone.now())
        for a in auctions:
            if a.banned:
```

```
                # Banned auctions are not resolved
                pass
            else:
                winning_bid = Bid.getLatestBidForAuction(a)
                if winning_bid:
                    a.winner = winning_bid.bidder
                a.active = False
                a.save()

                # Auction resolved, send email to seller and bidders (including
winner)
                receivers = [a.seller.email, ]
                bidders = a.getBidders()
                for b in bidders:
                    receivers.append(b.email)
                message = "The following auction has ended: \n\n"
                message += a.information()
                if winning_bid:
                    message += "\n\nThe winner was " +
winning_bid.bidder.get_full_name()
                else:
                    message += "\n\nThere was no bidders."
                send_mail("Auction ended", message, "noreply@YAAS.com", receivers,
fail_silently=False)

                print("Auction " + str(a.id) + " was resolved!")
```

7. Concurrency

   The concurrency in bidding is done by making a comparison of the form data with database data. The bid will not be accepted when the ‘updated’ date is older than the auctions ‘updated’ data in the database. In this case the bid is not accepted. In addition, the user can change the description of the auction before accepting the bid.

8. REST API

   The application is developed with a rest API which can be in the python file restframework_api. The API provides bid functionality for authenticated users, lists all active auctions, view all auctions by id and search auctions with or without login.

   - List of all active auctions
     YAAS/api/auctions

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

[
    {
        "id": 1,
```

```
            "title": "Iphone 6",
            "description": "Nice and Affordable Iphone 6 still in good condition . you have to buy it",
            "seller": 1,
            "start_date": "2018-10-24T22:31:42.855754Z",
            "updated_date": "2018-10-25T10:48:52.683964Z",
            "end_date": "2018-10-26T18:00:00Z",
            "minimum_price": "200.00",
            "active": true,
            "banned": false
        },
        {
            "id": 2,
            "title": "Tv 50 inch",
            "description": "Good colorful smart Tv at cheaper price",
            "seller": 4,
            "start_date": "2018-10-24T23:34:30.898084Z",
            "updated_date": "2018-10-24T23:34:30.898084Z",
            "end_date": "2018-10-27T12:00:00Z",
            "minimum_price": "300.00",
            "active": true,
            "banned": false
        },
        {
            "id": 4,
            "title": "bmw",
            "description": "good and nice",
            "seller": 7,
            "start_date": "2018-10-25T23:10:36.797691Z",
            "updated_date": "2018-10-25T23:11:06.522905Z",
            "end_date": "2018-10-28T18:00:00Z",
            "minimum_price": "1000.00",
            "active": true,
            "banned": false
        },
        {
            "id": 5,
```

```
        "title": "house for sale",
        "description": "nice and affordable",
        "seller": 4,
        "start_date": "2018-10-25T23:12:24.897783Z",
        "updated_date": "2018-10-27T21:09:58.522909Z",
        "end_date": "2018-10-28T18:00:00Z",
        "minimum_price": "2000.00",
        "active": true,
        "banned": false
    }
]
```


9. Testing

    - The testing was done that a user must be logged in to create an auction. The auction is also saved in the database.
    - A bidder must be valid user in order to make a bid. A seller cannot bid on his own auction.

10. Language switching

    The language switching has three options: English, Finnish and Swedish. The language switching is activated using the ‘request.session.’

11. Optional (soft deadline)

    When a user bids on an auction within five minutes of the deadline, the auction deadline is extended automatically for five minutes. This will allow other users to place a new bid.

```
# Soft deadlines, A bid made within 5 minutes of deadline
# automatically extends the deadline with 5 minutes.
if (auction.end_date - timezone.now()).seconds < 5*60:
    auction.end_date += timedelta(minutes=5)
    auction.save()
```