

1 YAAS Django project YAA_App application

1.1 Application URL

Application can be found at <http://mikak.pythonanywhere.com/>

2 Implemented requirements

List of tasks for this assignment round. Those tasks, which will be implemented later on so called leftover assignment are also in this list, but they are marked (implemented later).

List of implemented features:

1. UC3: create auction
2. UC4: edit auction description
3. UC5: Browse & Search
4. UC11: currency exchange
5. UC1: create user
6. UC2: edit user
7. UC6: bid
8. UC7: ban auction
9. WS1: Browse & Search API
10. WS2: bid API
11. UC8: Resolve auction
12. OP2: Soft deadlines for biddingUC9: language switching
13. OP3: store language preference (code exist but due to issues with fixtures not taken into a use)
14. OP1: send seller auction link
15. UC10: concurrency
16. TR1: Database fixture and data generation program
17. TR2.1: Functional tests for UC3
18. TR2.2: Functional tests for UC6
19. TR2.3: Functional tests for UC10

2.1 Administrator credentials

Administrator username to database is **mkkvj**. Administrator password is **Mikak1234**.

2.2 List of python packages

```
AWSIoTPythonSDK==1.2.0
certifi==2017.7.27.1
chardet==3.0.4
coverage==4.5.1
Django==1.11.6
django-common-helpers==0.9.1
django-cron==0.5.0
django-filter==1.0.4
django-mathfilters==0.4.0
```

```
django-nose==1.4.5
django-user-language-middleware==0.0.2
djangorestframework==3.7.0
idna==2.6
Markdown==2.6.9
nose==1.3.7
pycron==0.40
pytz==2017.2
requests==2.18.4
simplejson==3.11.1
twill==0.9
urllib3==1.22
UserManager==0.5.3
virtualenv==15.1.0
virtualenvwrapper-win==1.2.1
```

2.3 Database

YAAS_App application is using Sqlite data base. There are Auction, Bid and User model. User model is using Django's build in implementation.

2.4 Implementation descriptions

These implementation descriptions relates to implemented features.

2.4.1 UC1 Create user

From main page link create user leads to URL `url(r'^createuser/$',register_user)`. View function `register_user` will first render user form as it is GET request and not POST request. User form is named `Registrationform` and it will be rendered to template `registration.html`. User form is extended from Django's default user creation form by adding email field to form and template. After user has filled the form and template sends form with POST request to same url. Filled user form is validated and if validation is ok. User data will be saved to data base.

2.4.2 UC2 Edit account information

Link to Edit User Data can be found at home page. User needs to login before link is visible. User can change password and email address from page. a Edit User Data link on home page leads to `edituser` method on views. `Edituser` method checks if user is authenticated and renders form on `edituserdata.html` page with password and email data, if request method is GET. User can change password and email on webpage. Changed password and email are sent back with POST method to same `edituser` URL. Sent form is handled and changes are saved to database by `edituser` method.

2.4.3 UC3 Create a new auction

User can create a new auction from home page by pressing Create User link. Link is only visible if user is logged in. This login check has been done in template level. Link leads to `AddAuction` class, which renders `createauction.html` page as request is a GET method.

Createauction page shows form, which checks that user fills required data to a new auction correctly. Filled form is checked that auction has more than 72 hours run and minimum price is at least 0.01. Also, Title and description are required to proceed. Submitted form is received in AddAuction class, which checks validity of POST method request (checks are already described). Form with all already filled data and additional auction status data and seller data and confirmation form are rendered as confirmation html page. This page shows user given data and asks for user confirmation to save auction. If user selects yes from radio button and submits data, then form is handled by saveauction method. If user selected NO then user is redirected to home page. If user selected YES, then data from html form is saved to database as a new auction. Each auction has fields for seller, title, description, start price, latest bid, auction end time and auction status. After the auction has been saved, confirmation email is sent to auction creator (seller) with html link to homepage. Html parts of message are not rendered as email is sent to console, but link is shown on console.

2.4.4 OP1 send seller auction link

A Link is send to seller as embedded content to email. Html message is not rendered correctly as email is sent to console, which is not rendering html content. A Link to home page is however is shown correctly in sent email. Code for sending link is located in saveauction method in views.py.

2.4.5 UC4 Edit the description of an auction

Seller can edit auction description by pressing Edit Auction link from main page. a Link leads to editauction method, which checks if user is authenticated and if user is seller of item. GET-method is sent with id number of selected auction. User is redirected to login page or home page, if user is not authenticated or seller. Correct auction is found by help of id number in request parameter. If user is seller then auction lockedby status is checked and if auction is not locked by someone else, then auction is locked for edit and editauction.html page is rendered with auction data. A Seller can modify description of auction on html page and submit the page for saving to savechanges URL which leads to savechanges method. Correct auction is again filtered with help of offset parameter. Modified auction description from POST message is saved over older version in database. Also, auction is released from locked status before auction is saved. User is redirected to home page with message enabled for translation.

2.4.6 UC5 Browse and search auctions

Auction browsing is implemented in browseauctions method, which lists all auction objects in order by title. Listed auctions are rendered with many other things to auctionlist.html page. Auctionlists.html page is shown as block on base.html template and together they form home page. Auctionlists.html page renders and shows each auction as a list which can be browsed by scrolling home page up and down.

Auctions on home page are shown if auction status is not B, which means that auction is banned.

Auction can be searched by its title by pressing Search Auction link on home page. The Link leads to search method. Search method checks if request has a query parameter or not. If parameter

doesn't exist, then filtering return empty list. Search method renders searchauction.html page with empty auctions list. Rendered page has input box for searching by title. User writes searched title and accepts by search button. Form is sent and received again by search method. Now query parameter exists and filters shows 10 closest matching auctions. User can end searching by clicking home page link.

2.4.7 UC6 Bid

User activated add bid from main page by selecting auction and clicking add bid. This will send GET request with auction number as request parameter. User authentication status is first checked and redirected to login page if not authenticated. Next user is tested if he is administrator or seller and redirected to home with appropriate message if he/she is. As only normal authenticated users are allowed to bid. If user is ok form is rendered. In this case there is data from bid and auction tables so data from both database tables is rendered to form and to template. Data from auction table is presented to bidder. Bidder can change only bid value. Bid validity is checked by default validator. Time, when bid is placed will be checked. If bidding time exceeds the bid's end time, bid is not accepted. If bid is placed 5 minutes before bid's closing time the bid time will be extended by 5 minutes. All additional bids will extend this time by another 5 minutes until nobody adds bids within 5 minutes of closing time. If bid is placed after the end time the bid is cancelled and auction status changed to D due to wait for resolve process. Form is send for further checking and saving POST request. Savebid view function will checked if form is valid. Then it will check if placed bid was incremented more than minimum 0.01 euros. If bid is not valid, user is redirected back to bidding page.

If bid is accepted, email is send to seller (console) and all the bidders.

2.4.8 UC7 Ban

Banned auction is selected at home page in same manner as in bid. Auction to be banned is parameter of GET request. User who bans the auction needs to be authenticated administrator. This is first checked and if authentication is ok, confirmban.html page is rendered with auction table. At confirmation page there is auction data and simple yes and no selection to confirmation. form is send as POST request to another banauction URL. POST request is processed by ban function. Selected auction's auction status value will be changed to 'B', means banned. When auction status is banned it is not shown in main page. It is not possible to bid banned auction. Also, automatic resolve process will not process it.

After auction status is changed the seller and all bidders are notified about ban by email.

2.4.9 UC8 Resolve auction

Auctions are resolved once a day by Django's management command. Management command needs to be fired by external scheduler in order to do tasks in automatically. This process is handled by task scheduler provided by Pythonanywhere site. It runs once a day resolveauction.py file in YAAS_App management directory. More detailed management command from task

scheduler is following **/home/MikaK/.virtualenvs/YAAS-virtualenv/bin/python /home/MikaK/web-services-2017-project-part2-xzerkses/manage.py resolveauctions.**

Actual auction resolve is handled by Command class in resolveauction.py file, which is under YAAS_App/management/commands folder. Handle function reads periodically all auctions saved to system and check auction status and auction ending time. If auction status is still Active and auctions ending time has passed, the auction will be resolved. Also, if auction's status is 'C', it will be resolved. In all these mentioned cases status of auction will be changed to D, which is for adjudicated. All involved bidders and seller is notified by email that auction is resolved.

```
Automatic auction resolving process starts.
*****
Successfully resolved auctions
Successfully resolved auctions
Successfully resolved auctions
Successfully resolved auctions
Successfully resolved auctions
Successfully resolved auctions
Successfully resolved auctions
Successfully resolved auctions
Successfully resolved auctions
Successfully resolved auctions
Successfully resolved auctions
Successfully resolved auctions
Successfully resolved auctions
Successfully resolved auctions
Successfully resolved auctions
*****
Automatic auction resolving process ends.

2017-10-20 19:03:10 -- Completed task, took 65.00 seconds, return code was 0.
```

Picture 1 Pythonanywhere task scheduler log. All auctions have already been resolved.

2.4.10 UC9 Support for multiple languages

User can select language from English, German or French. Selection is done in home page from dropdown menu. Selected language is sent as POST message to set_lang method in views.py. Selected language is parsed from message and language translation is activated and saved to a session. Selected language is read from session in browseauctions method. Selected language is then rendered along with all available languages for home page to use. So, set_lang activates translation and browseauction renders the selection and all available languages.

Views.py methods contain hooks for translatable texts and uses ugettext function for collecting those texts. Templates also contains hooks ({% trans %}) for translatable texts. Translation strings are included in message files for each extra language, which are German and French. Message files with .po extension were compiled to .mo files. All message files and compiled translation files can be found at locale directory.

2.4.11 OP3 store language preference

Language preference was implemented, but is not in use at the moment as there was issues with duplicates when using fixture loading. Code that was used for store language preference is commented out in views.py and in models.py

Users language preference is stored to database to lang_preference field in Profile class. Profile class has a direct one to one relationship with User class. Related to Profile class are signal

methods which are connecting User class save and creation operations to Profile class. In other words, when a new User object is created, also a new Profile object is created. Also, if User object is saved, then also corresponding Profile is saved.

Language preference is saved in `set_lang` method in `views.py` when user has made his language selection in home page. Language preference saving works only for registered and authenticated user and this is checked before saving selection. User's language is read in `browseauction` method. User authentication status is also checked before language preference is read from database. If user doesn't have any selection then default `en` language is saved to database.

Permanent language preference works along with session-based language selection. Only difference is that permanent preference is in use for registered and authenticated user.

2.4.12 UC10 Support for multiple concurrent sessions

Support for concurrent sessions has been implemented by using pessimistic locking. When user is placing a bid to an auction, the `lockedby` status of auction will be checked in `addbid` method. If auction is not locked by anybody, then user can place a bid and auction is locked for any edit during that time. Locking is done by writing user session key to `lockedby` field in auction model. Auction model lock will be released when a bid value is saved to database. This is done by writing `lockedby` field back to `""` and saving auction model. Auction is locked in similar fashion, when seller edits auction description. `Lockedby` status will be checked in `Editauction` method before seller can modify auction description. If `lockedby` status is empty, a seller's session key is saved to `lockedby` field in auction model and that auction is locked for editing. When seller has made changes to auction description and submitted form, the `lockedby` status will be again written to `""` and saved to auction model and thus releasing a lock.

If user chooses to cancel placing a bid or cancel auction editing he/she must use application home link. Home link will handle that auction lock is released properly. This is done in `clearhome` method again by writing `lockedby` status to `""` and saving model, just before redirecting back to home page.

2.4.13 UC11 Support for currency exchange

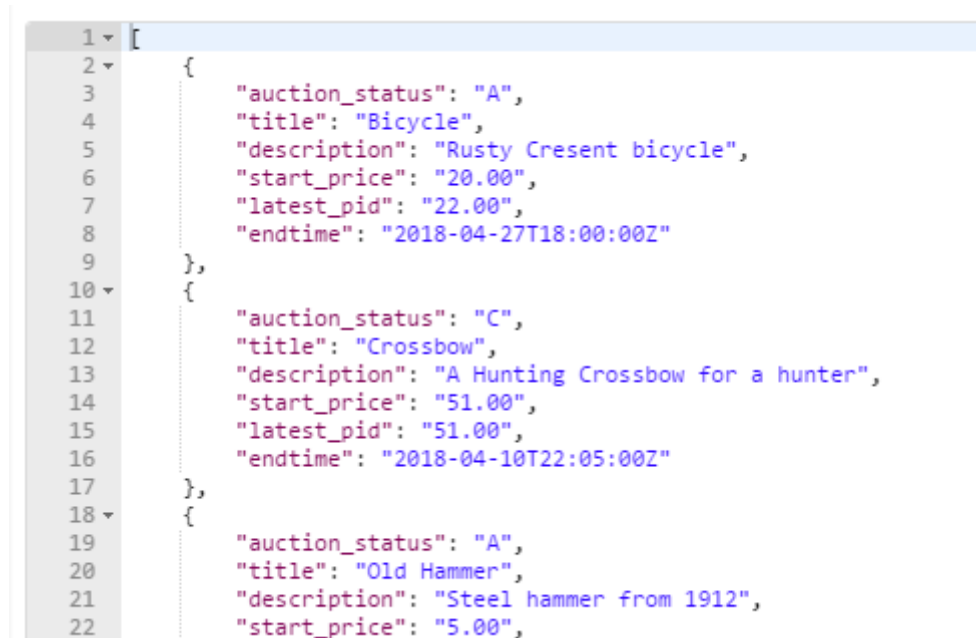
User can select used currency from home page using dropdown menu and pressing apply. Selected currency is sent as POST message to `readjson` method in `views.py`. Selected currency id and corresponding currency rate is parsed from POST message. Both values are then saved to selected currency and rate sessions and redirected to home page. So, this method only saves user selected data to sessions. Rendering of currency data for home page is done in `browseauctions` method in `views`. `Browseauction` method first reads latest currency data from <http://api.fixer.io/latest> and then checks if selected currency and rate sessions exist. Default euro currency and currency rate of 1 is used, if session data is not available. Currency and rate is then read from sessions and this data is rendered for `auctionlist.html` which is one section to create home page. Rendered data is used in `base.html` in form, which shows all available currency options to user in dropdown menu.

2.4.14 WS1 Browse & Search API

Django's Rest framework was installed for this task. URL for browsing is `api/auctions/$`, which uses `AuctionList` class. `AuctionList` class handles GET method by simply selecting all auctions and then serializing them. Serialized auctions are, then send to client in json format or if auctions are not found then 404 response is sent. Search is implemented by `AuctionSearchAndBid` class. Selected auction parameter is transferred in GET request's URL, which is then used to select correct auction. Selected auction is serialized and response to user sent back in json format.

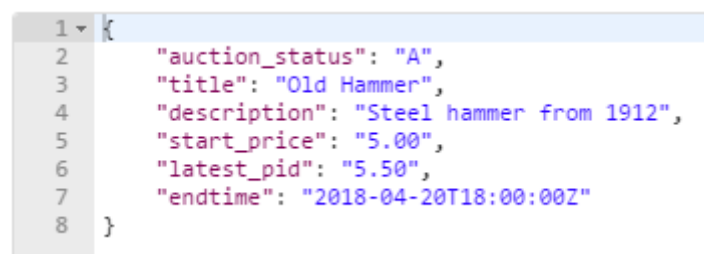
Related to this Rest API implementation is serializer found in `serializer.py`, which serializes the auction model. Rest functions can be found in `RestfulAPI.py` and serializer is found in `serializers.py` file.

`Settings.py` contains global setting for default authentication classes and default permission classes, which have been set to `TokenAuthentication` and `IsAuthenticatedOrReadOnly`. This allows to use browse and search functions without authenticating the user.



```
1 {
2   {
3     "auction_status": "A",
4     "title": "Bicycle",
5     "description": "Rusty Crescent bicycle",
6     "start_price": "20.00",
7     "latest_pid": "22.00",
8     "endtime": "2018-04-27T18:00:00Z"
9   },
10  {
11    "auction_status": "C",
12    "title": "Crossbow",
13    "description": "A Hunting Crossbow for a hunter",
14    "start_price": "51.00",
15    "latest_pid": "51.00",
16    "endtime": "2018-04-10T22:05:00Z"
17  },
18  {
19    "auction_status": "A",
20    "title": "Old Hammer",
21    "description": "Steel hammer from 1912",
22    "start_price": "5.00",
```

Picture 2 Response to GET message (Browse) sent with postman. Sent message URI GET `/api/auctions/`



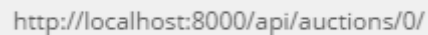
```
1 {
2   "auction_status": "A",
3   "title": "Old Hammer",
4   "description": "Steel hammer from 1912",
5   "start_price": "5.00",
6   "latest_pid": "5.50",
7   "endtime": "2018-04-20T18:00:00Z"
8 }
```

Picture 3 Response to GET message (Search) sent with postman. Sent message URI GET `/api/auctions/2/`

2.4.15 WS2 Bid via API



BID is added via API by using PUT method. Implementation can be found in `AuctionSearchAndBid` class in `RestfulApi.py` file. Parsing method uses additional permission and authentication class function decorators in order to restrict access for authenticated users. Authentication is done with

token authentication. Method parse PUT message's payload and makes same checks that was done with main bid function. After checks have been successfully verified bid is saved to database.

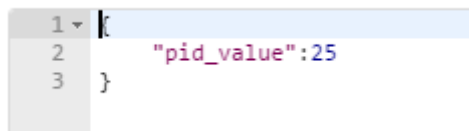


```
http://localhost:8000/api/auctions/0/
```

Picture 4 URI of PUT message (BID fuction)

	Key	Value
	Content-Type	application/json
	Authorization	Token 8bd067592d59600e33a012d65e2d4cb4d17559...
	New key	Value

Picture 5 PUT method Header content.



```
1 {
2   "pid_value":25
3 }
```

Picture 6 body of PUT method (bid value)

2.4.16 OP2: Soft deadlines for bidding

Soft deadlines are implemented by savebid function in views.py. As mentioned earlier before bid is saved the auction's end time is checked and if bid is placed closer than 5 min of closing time the auction end time will be extended by 5 minutes. This is done simply by adding 5 minutes to auction table's auction endtime value. Resolve process and other checks for ending time will rely on this value and therefore it will extent the bidding time.

2.4.17 TR1 Database fixture and data generation program

Data generation program is populatedatabase.py under management/commands directory. program extends BaseCommand class and it is basically custom-made management command. It picks random username from 9 available and adds randomized numbers to picked names and uses 5 different email suffixes to create 50 users with emails and passwords. Passwords are 8 characters long randomized passwords. Sellers are picked from 50 users randomly, starting prices and items are selected randomly. There are 7 items where program can choose.

Program creates from 6 to 15 bids on some of the auctions and using available biddersp.

Program can be started by running YAAS directory command **python manage.py populatedatabase**.

2.4.18 TR2.1: Functional tests for UC3

All tests for all test cases are located in test.py. Functional tests related to auction creation are placed in AuctionCreationTestCases class. Auction creation is first tested in lowest level, that basic create function works. Form used in auction creation and form validation methods and messages are checked that they work when data is valid and, in few cases, where data is incorrect. These test cases are:

- auction time is too short
- time format is entered in wrong format
- too low auction price
- auction doesn't have a title
- auction doesn't have a description
- email is sent, after the auction is saved

2.4.19 TR2.2: Functional tests for UC6

Tests for bidding are placed in BidTestCases class. Few additional auctions with different bidding end time are created for testing purposes. Bid creation is tested with following test cases:

- bid is created with normal valid data
 - sent email is checked
- bid created with too low bid value
- bid created with less than 5 minutes to bidding time left (soft bid ending)
- bid created with bidding time already ended
- seller adds bid for his/her own bid
- user adds bid to auction
 - test GET method

2.4.20 TR2.3: Functional tests for UC10

There are also test cases for testing concurrency. Test setup creates normal auction. Auction is then placed in locked status before testing starts. Test cases are:

- Concurrency is tested by trying to add bid while auction is locked
- Seller tries to edit auction while auction is locked
- Seller tries to ban auction while auction is locked