Åbo Akademi

Web Services (YAAS Project)

Salman Aslam Gill

Student ID: 42009

1 Implemented

1.1 Compulsory Requirements

1. **UC1:** Create an user account
2. **UC2:** Edit account information
3. **UC3:** Create a new auction
4. **UC4:** Edit the description of an auction
5. **UC5:** Browse and search auctions
6. **UC6:** Bid
7. **UC7:** Ban an auction
8. **UC8:** Resolve auction
9. **UC9:** Support for multiple languages
10. **UC10:** Support Multiple Concurrent Sessions
11. **UC11:** Support for currency exchange
12. **WS1:** Browse and search via API
13. **WS2:** Bid via API
14. **TR1:** Database Fixture for functional testing
15. **TR2:** Automated Functional Tests
    - i. **TR2.1:** Functional tests for UC3
    - ii. **TR2.2:** Functional tests for UC6
    - iii. **TR2.3:** Functional tests for UC10

1.2 Optional Requirements

1. **OP1:** The email message includes a link which would allow the user to see the created auction details without logging in
2. **OP2:** Soft deadlines. If a user bids at an auction within five minutes of the auction deadline, the auction deadline is extended automatically for an additional five minutes.
3. **OP3:** Store language preference permanently for registered users

1.3 Extra Features

1. Choosing hours for auction rather than selecting date. Gives ease of use to users so they dont have to think about timezones etc
2. In auction details page, give users auction ending time in (X Hours Y Minutes) format. As well as through api
3. Enhance **OP1** to include also a link in email using which seller can edit description of auction without logging in. Following is an example of sent link:
    - *(Hostname)/auctions/auction/non/edit/20171013125508fIzsSs7UeRjr/*
4. Api also provides ‘count‘ parameter in search and browse to restrict the number of returned auctions
5. Twitter bootstrap is used for front-end to make application user friendly
6. Latest auctions section where users can see latest ten(10) auctions
7. Banned auctions section for admin users, where they can see the auctions they have banned
8. Ability to unban the banned auctions for admin users
9. Helping information on forms for users.

2 List of Python packages same as in “requirement.txt”

- Django==1.11.5
- djangorestframework==3.6.4
- requests==2.18.4

3 Admin username and password for DB

- Username: admin
- Password: Easports

4 Implementation of session management

The application uses Django’s own session management functions.

Session is used for following purposes

1. Users identification (authenticated or not)
2. Currency preference mangement
3. Language preference management

4. New auction details temporary storage before confirmation from user

User authentication is, moreover, implemented using djangos built-in authentication module

Example:

Saves the infomation in session

```
request.session [’key’] = value
```

Fetches the information from session

```
value = request.session [’key’]
```

Deletes the information from session

```
del request.session [’key’]
```

5 How do you implement the confirmation form in UC3?

Django’s session management functions are used to temporarily store the auction information.

Following is flow of implementation:

1. User fills the auction form and submits
2. If form validation is passed, auction information is stored in user‘s session storage
3. User is redirected to confirmation page
4. If user selects ‘Yes‘ as confirmation option, auction information is retrieved from user’s session storage and saved in database
5. Auction information is deleted from user’s session
6. If user selects ‘No‘ as confirmation option, auction is not saved in database

6 Resolving auctions

The resolving of auctions is accomplished using deamoned(backgrounded) threaded functions (*threading.Timer*)

1. **put_into_due():**

   runs every 60 seconds to put active auctions into ‘Due’ state after auction duration is elapsed

2. **resolve_auction():**

   - runs every 5 minutes to resolve auctions from ‘Due state to ‘Adjudicated‘
   - sets the winner to the latest bid and sends mails

7 Concurrency (specially in UC6)

- Concurrency is handled with revisions of an auction and optimistic concurrency control. The auctions revision number is stored in a hidden field, when a user views an auction. And is then compared to the revision number in the database when the user tries to place a bid. This prevents users from placing bids on auctions whose description has changed since the user read it

- To handle concurrency in bidding and expiration of auction, it is always validated in program before storing the bid in database that there are no bids by other users in meantime that are higher than user's current bidding amount

8 REST API

The application has a rest API accessed through */api/.* The api provides browse and search functionality for non-authenticated users and bid functionality for authenticated users

8.1 Browse

Auctions can be browsed by sending a GET request to */api/auctions/*

The system will then return all auctions in JSON format and sorted by created date in descending order (i.e. latest first)

```
[
  {
    "id": 409,
    "title": "title 50",
    "end_date": "2017-11-11T12:47:49.660536Z"
  },
  {
    "id": 364,
    "title": "title 5",
    "end_date": "2017-11-11T12:47:41.349262Z"
  }
]
```

8.2 Search

A search is initialized by sending a GET request to */api/auctions/?title=<search words>*

The system will then return all auctions, whose title matches the "search words" in descending order by created date

There is also a "***count***" parameter supported in this request through which you can limit the number of returned auctions. By default, its value is set to **10.**

Example request can be like this:

GET request to */api/auctions/?title=<search words>&count=10*

Or:

GET request to */api/auctions/?count=10*

Note: Resulting JSON for search is same as browse request above.

8.3 Details

Detail of an auction can be obtained by sending GET request to */api/auctions/<id>/*

The system will then return details of auction in JSON format, whose "id" is provided in request.

```
{
  "auction_details": {
    "id": 409,
    "title": "title 50",
    "description": "description of item 50",
    "revision": 1,
    "starting_price": "0.05",
    "bid_price": "0.06",
    "currency": "EUR",
    "end_date": "2017-11-11T12:47:49.660536Z",
    "seller": "user50",
    "bid_set": [
      "user2"
    ]
  },
  "time_remaining": {
    "hours": 601,
    "minutes": 42
  }
}
```

Where "*bid_price*" is current bid amount, "*bid_set*" is list of usernames of current bidders, "*revision*" is current version of the auction and "*time_remaining*" tells how much time left for auction.

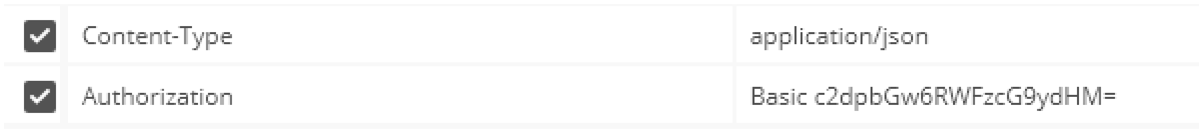
8.4 Bid

Authenticated users can place bids on auctions by sending a POST request to

/api/auctions/<id>/bid/ with their login credentials in the authentication header in base64 format ***username:password***

You can use website like https://www.base64encode.org/ to encode in base64 format.

Header example in Postman looks like as below:



✓	Content-Type	application/json
✓	Authorization	Basic c2dpbGw6RWFzcG9ydHM=

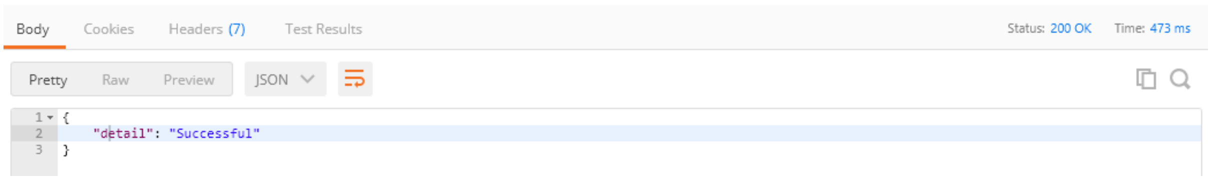
The request body should contain bid details in JSON using following format:

```
{
  "revision": 1,
  "my_bid": 0.19
}
```

Where ***revision*** is the current revision of auction which can be obtained from auction details and ***my_bid*** is the amount user want to bid on an auction.

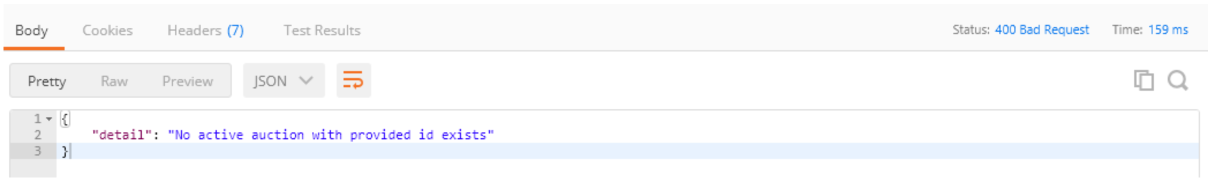
The system will then check that the bid validates and return status code 200 if successful otherwise the error status code and description of error.

An example of successfull bid response (postman):



```
{
    "detail": "Successful"
}
```

An example of failed bid response (postman):



```
{
    "detail": "No active auction with provided id exists"
}
```

9 Functional tests

The system has tests for auction creation and bids covering concurrency scenario as well.These tests uses a fixture with generated data that is generated through a view method.

Followings are details of tests:

1. ***testCreateAuctionWithoutAuthentication(self):***

   In this test it, is checked that auction creation is not possible without authienticated user

2. ***testCreateAuctionWithAuthentication(self):***

   In this test it, is checked that auction can be created with authenticated user. As well as folowing scenario’s

   - 72 hours minimum auction duration check
   - Confirmation of auction with ‘False‘ option scenario
   - Confirmation of auction with ‘True‘ option scenario

3. ***testPlaceBidWithoutAuthentication(self):***

   In this test it, is checked that bid placement is not possible without authienticated user

4. ***testPlaceBidWithAuthenticationSuccesfully(self):***

   In this test, it is checked that bid can be placed with authenticated user

5. ***testFailBidDueToEqualThanStartPrice(self):***

   In this test, it is checked the bid is not placed when bid amount is equal to start price of auction

6. ***testFailBidDueToLessThanStartPrice(self):***

   In this test, it is checked the bid is not placed when bid amount is less than the start price of auction

7. ***testFailBidDueToAlreadyWinning(self):***

   In this test, it is checked the bid is not placed when the user is already winning

8. ***testFailBidDueToBidderEqualToSeller(self):***

   In this test, it is checked the bid is not placed when the user is the seller

9. ***testFailBidDueToNonActiveAuction(self):***

   In this test, it is checked the bid is cannot be placed on non-active auction

10. ***testBidConcurrencyForDescriptionUpdate(self):***

    In this test, concurrency is checked when description of an auction is updated in real time

11. ***testBidConcurrencyForAnotherUserBid(self):***

In this test, concurrency is checked when there are bids from multiple user on an auction

10 Language switching mechanism

There is support for three languages in language switching mechanism

- English (default)
- Swedish
- Finnish

User can select from the available languages and its preference is stored in the user's session storage. If the user is authenticated, language preference is also stored in database for the user.

Language switching is basically achieved by using different template for each language. There is a base directory for English language templates and sub-directories for Finnish and Swedish language.

On every request, template is selected based on user's preferred language. An example of the returned template selection is as follows:

return render (request,
*'auctions/'+**language_folder_selection(request)**+'auction_confirmation_form.html', context)*

where "*language_folder_selection(request)"* function returns string that indicates the language folder inside auction folder for the current user.

11 Location data generation program

Data generation program generates the following data in database:

- 50 users
- 50 auctions
- Some Bids

Data generation program can be invoked using following *URI:*

*(hostname)**/populate/***

12 Currency Exchange

For currency exchange following api endpoint is used:

http://api.fixer.io/latest

Latest values of currency are gathered from this api endpoint and saved into database. Values of currencies are obtained from api every 10 minutes using threaded task. Then for users, on each request, currency value is updated from database.