



DEVELOPMENT OF SERVER-SIDE WEB SERVICES

Project Report



10/11/2019

STUDENT NUMBER: 42125

Åbo Akademi University

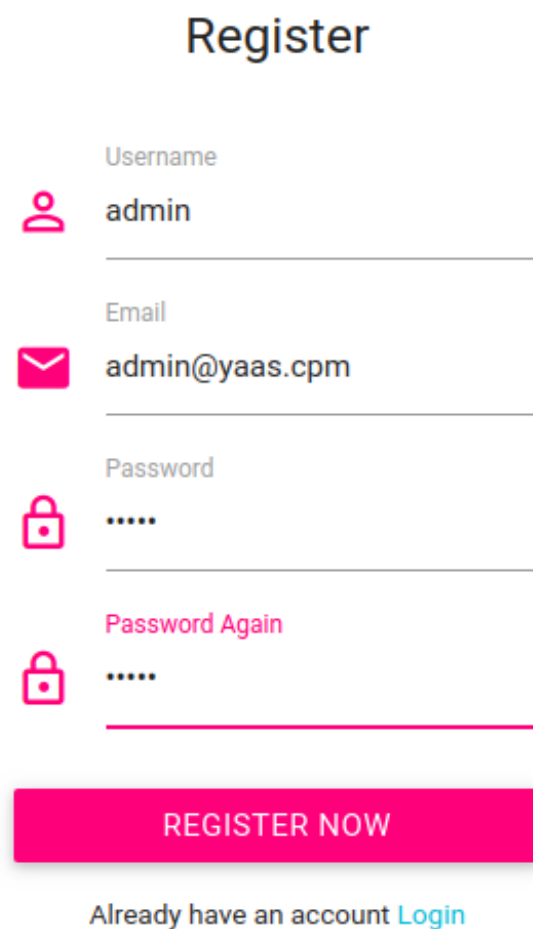
List of Implemented Requirements:

1. UC1 Create user account
2. UC2 Edit account information
3. UC3 Create a new auction
4. UC4 Edit the description of an auction
5. UC5 Browse & Search auctions
6. UC6 Bid
7. UC7 Ban an auction
8. UC8 Resolve auction
9. UC9 Support for multiple languages
10. UC10 Support for multiple concurrent sessions
11. UC11 Support for currency exchange
12. Restful web service

Approaches (with screenshots/ descriptions) that are used to fulfil the requirements:

UC1 Create a user account:

REQ 1.1: Any anonymous user can create a new user account using username, email and password.



The image shows a 'Register' form with the following fields and elements:





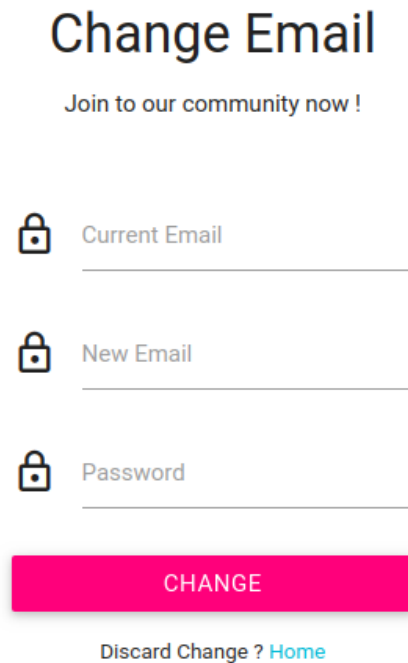
- Register** (Title)
- Username** (Label): 
- Email** (Label): 
- Password** (Label): 
- Password Again** (Label): 
- REGISTER NOW** (Button)
- Already have an account** [Login](#) (Text)

Fig 1 : Screen shot taken during test

UC2 Edit account information:

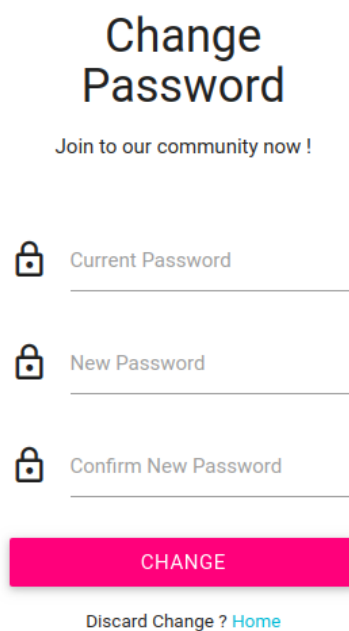
Req 2.1: A logged in user can change his/her email address.



The screenshot shows a web form titled "Change Email". Below the title is a subtitle "Join to our community now !". The form contains three input fields, each preceded by a lock icon: "Current Email", "New Email", and "Password". Below these fields is a prominent pink button labeled "CHANGE". At the bottom of the form, there is a link that says "Discard Change ? [Home](#)".

Fig 2 : Screen shot taken during test

Req 2.2: A logged in user can change his/her password.



The screenshot shows a web form titled "Change Password". Below the title is a subtitle "Join to our community now !". The form contains three input fields, each preceded by a lock icon: "Current Password", "New Password", and "Confirm New Password". Below these fields is a prominent pink button labeled "CHANGE". At the bottom of the form, there is a link that says "Discard Change ? [Home](#)".

Fig 3 : Screen shot taken during test

UC3 Create a new auction:

Req 3.1: Any registered user can create a new auction:

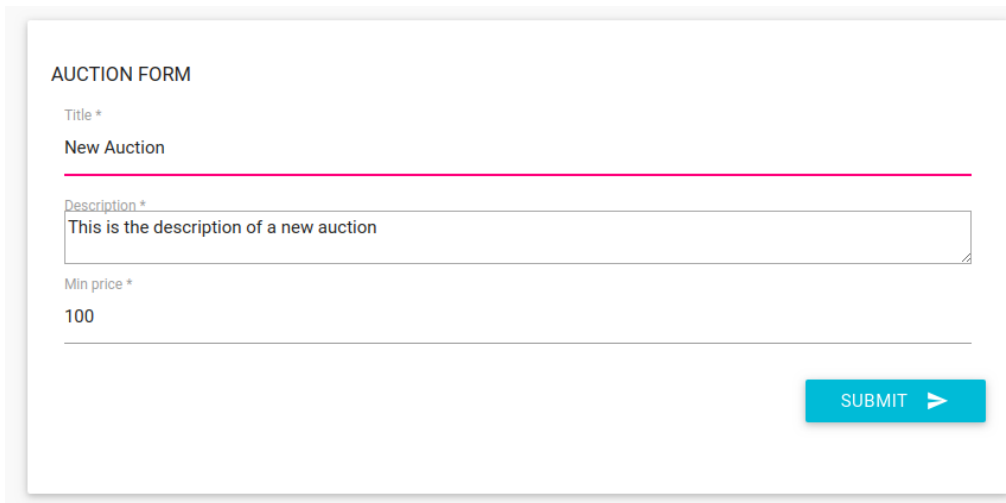
A screenshot of a web form titled "AUCTION FORM". It contains four input fields: "Title *" with the text "New Auction", "Description *" with the text "This is the description of a new auction", and "Min price *" with the text "100". A blue "SUBMIT" button with a right-pointing arrow is located at the bottom right of the form.

Fig 4 : Screen shot taken during test

Note: Unregistered user shall be redirected to login page while trying to access the form.

Req 3.2: When creating an auction, the system registers the following information:

- The user that creates the auction - See in Figure no. 4
- The title of the auction - See in Figure no. 4
- The description of the item(s) to sale - See in Figure no. 4
- The minimum price - See in Figure no. 4
- The deadline -Duration to bid an auction is 72 hours and it is set right at the moment the auction is created by calling a scheduler function automatically. The scheduler function is implemented using django background_task package. See in Figure no. 5

```
@background(schedule=259200)
def set_deadline(auction_id):
    Auction.objects.filter(id=auction_id).update(status_id=Auction_Status.objects.get(status="Due").id)
    message = "Deadline Met"
    print(message)
    return message
```

Fig 5 : Screen shot taken during test

Req 3.3: The user must be asked for a confirmation before creating a new auction:
See in **Figure no. 6**

Dear pranoy,

Please go to the link below and confirm auction you made.

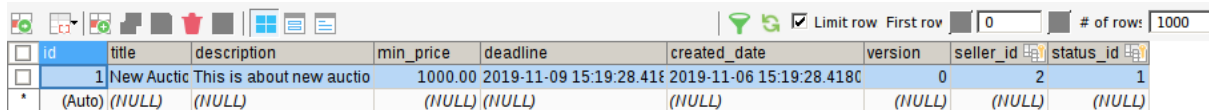
Link: <http://127.0.0.1:8009/auction/create/2>

Thanks.

YAAS Team.

Fig 6 : Screen shot taken during test

Req 3.3.1 : A confirmed auction will be stored in the system: **See in Figure no. 7**



id	title	description	min_price	deadline	created_date	version	seller_id	status_id
1	New Auctic	This is about new auctio	1000.00	2019-11-09 15:19:28.418	2019-11-06 15:19:28.418	0	2	1
*	(Auto)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)

Fig 7 : Screen shot taken during test

Req 3.3.1 : An unconfirmed auction will NOT be stored in the system:

An unconfirmed auction will NOT be stored in the system: Unless the seller save an auction by the link send to his email (see in **fig 6**), the auction would not be saved in the system.

Req 3.4 : The system must confirm by email to the seller that an auction has been created: See in **Figure no. 8**

Dear pranoy,

An auction has been created successfully.

Thanks.
YAAS Team.

Fig 8 : Screen shot taken during test

Req 3.5 The email message includes a special link which would allow the seller to modify the description of the created auction without logging in:

See in **Figure no. 6**

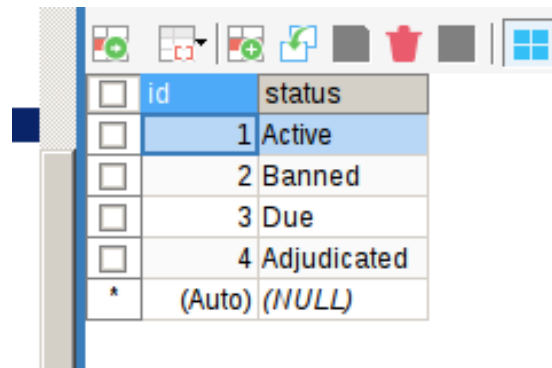
Req 3.6 : A user (other than the seller) can bid on an active auction. See in **Figure no. 9**

You are not allowed to bid on your own created auction.

Fig 9 : Screen shot taken during test

The lifecycle of an auction is as follows:

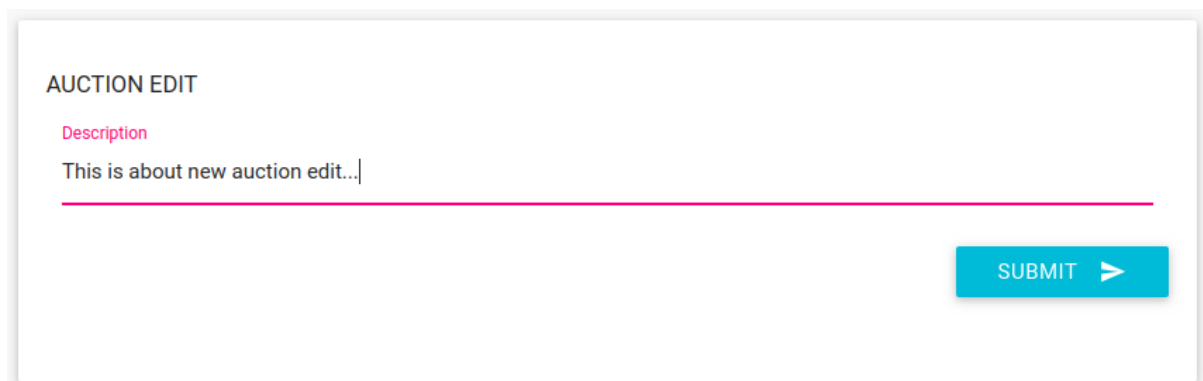
- Active: Once an action has been created, it becomes active until its deadline or until it is banned.
- Banned: An active auction can be banned by an administrator as described by UC7.
- Due: After the auction end date, the auction is due for adjudication.
- Adjudicated (see UC8): A due action has been processed by the system and the winner has been selected from all the bidders. See in **Figure no. 10**.



<input type="checkbox"/> id	status
<input type="checkbox"/> 1	Active
<input type="checkbox"/> 2	Banned
<input type="checkbox"/> 3	Due
<input type="checkbox"/> 4	Adjudicated
<input type="checkbox"/> *	(Auto) (NULL)

Fig 10 : Screen shot taken during test

Req 3.7 : The seller can update the description of the auction. See in **Figure no. 11**



AUCTION EDIT

Description

This is about new auction edit...

SUBMIT >

Fig 11 : Screen shot taken during test

How the UC3 implemented is as follows:

1. The user selects the create new auction link.

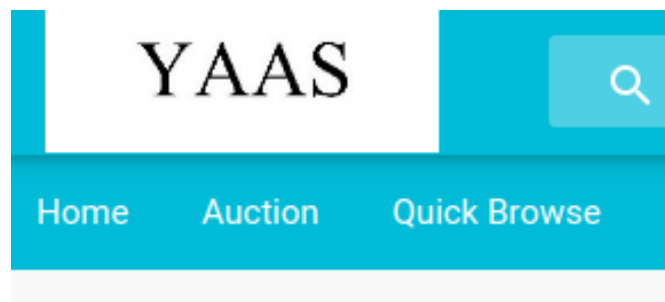


Fig 12 : Screen shot taken during test

2. A form with all the required auction fields is shown. The user fills in the form and submits it. If the user enters incorrect information in the form the system should show a proper error message and show the form again. See in **Figure no 4**.

3. A confirmation message is shown, asking if the user is sure to create a new auction. The user can select Yes or No. See in **Figure No. 13**.

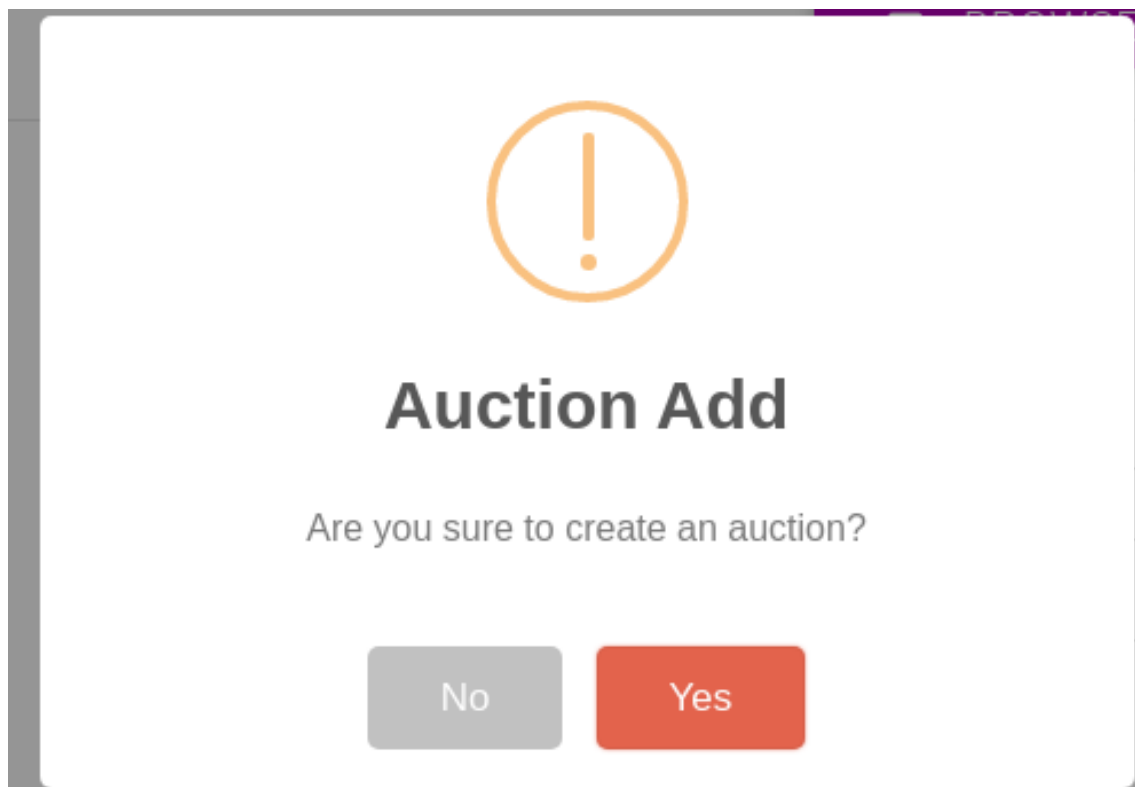


Fig 13 : Screen shot taken during test

4. if the user selects Yes in the confirmation form: The new auction is recorded in the system. See in **Figure No. 13**.

5. if the user selects No in the confirmation form: The new auction is not recorded in the system. See in **Figure No. 13**.

6. The user never answers the confirmation form: The new auction is never recorded in the system. See in **Figure no 7**

Email hosting is used for sending emails. To be more specific: django.core.mail for send_mail. See in **Figure no. 14**

```
EMAIL_USE_TLS = True
EMAIL_HOST = 'smtp.gmail.com'
EMAIL_HOST_USER = 'pranoybhowmick42125@gmail.com'
EMAIL_HOST_PASSWORD = 'Rumon999'
EMAIL_PORT = 587
```

Fig 14 : Screen shot taken during test

UC4 Edit the description of an auction:

Req 4.1: The seller of an active auction can change the description of the item(s) for sale. See in **Figure no.11**

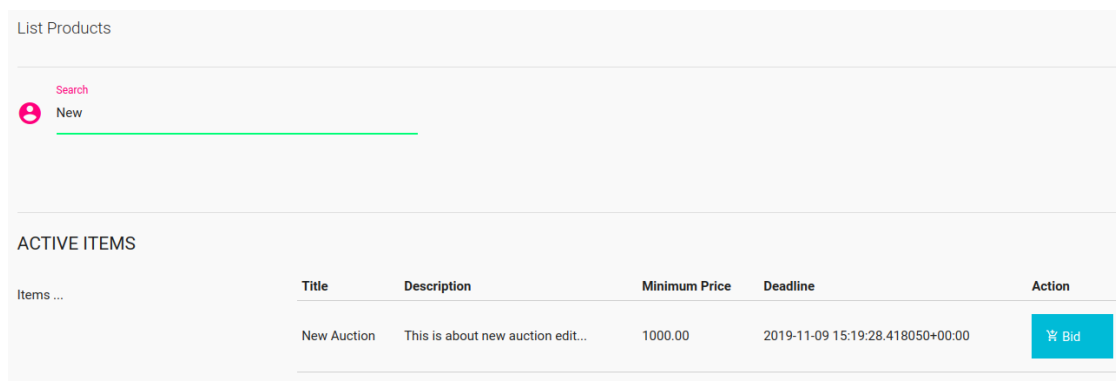
UC5 Browse and search auctions:

Req 5.1 : anonymous user must be able to browse the list of active auctions. See in **figure number 15**.

Req 5.2 : anonymous users must be able to search auctions by title. See in **figure number 15**.

Req 5.3 : registered users must be able to browse the list of active auctions. See in **figure number 15**.

Req 5.4 : registered users must be able to search auctions by title. See in **figure number 15**.



The screenshot shows a web application interface. At the top, there is a search bar with a magnifying glass icon and a 'Search' button. Below the search bar, there is a 'New' button. The main content area is titled 'ACTIVE ITEMS' and contains a table with the following columns: Title, Description, Minimum Price, Deadline, and Action. The table has one row with the following data: Title: New Auction, Description: This is about new auction edit..., Minimum Price: 1000.00, Deadline: 2019-11-09 15:19:28.418050+00:00, and Action: Bid (with a bid icon).


	Title	Description	Minimum Price	Deadline	Action
Items ...	New Auction	This is about new auction edit...	1000.00	2019-11-09 15:19:28.418050+00:00	 Bid

Fig 15 : Screen shot taken during test

UC6 Bid :

Req 6.1 : Any registered user can bid for an active auction, except the seller. See in **figure number 16.**

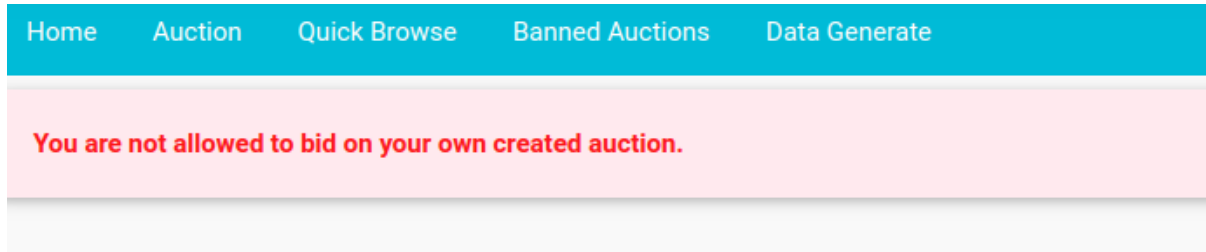


Fig 16: Screen shot taken during test

Req 6.2 : The minimum bid increment is 0.01. Only two decimal places are considered when bidding. See in **Figure no. 17**

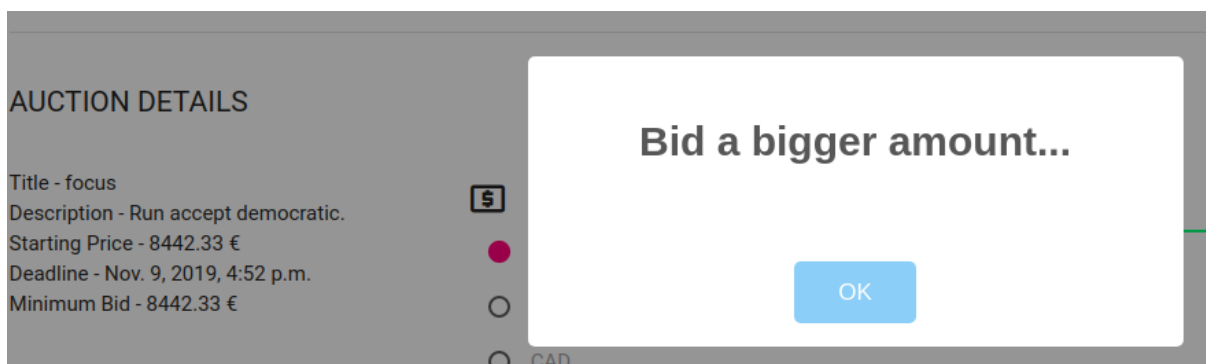


Fig 17 : Screen shot taken during test

Req 6.3 : A seller cannot bid on own auctions. See **figure no. 16**

Req 6.4 : The application must show to the user the most recent description of the auction before accepting bids from the user. See **figure no. 18**

Req 6.5 : A new bid should be greater than any previous bid and the minimum price. See **figure no. 17**

Req 6.6 : A bidder cannot bid on an inactive auction. Inactive auction are not shown in the list. See **figure no. 15**

The screenshot shows a web form titled "Selected Auction". Below the title is a section labeled "AUCTION DETAILS". On the left, there is a text area for "Title - New Auction" and a "Description - This is about new auction edit...". Below these are the "Starting Price - 1000.00 €", "Deadline - Nov. 9, 2019, 3:19 p.m.", and "Minimum Bid - 1000.00 €". To the right of the description is a "Currency List" section with four radio buttons: "EUR/€" (selected), "USD/\$", "CAD", and "BDT". Further right is an "Amount" input field with a red "\$" icon and a green underline, containing the value "1000.01". A blue "SUBMIT" button with a right arrow is located at the bottom right of the form.

Fig 18 : Screen shot taken during test

Req 6.7 : the seller must be notified by email that a new bid has been registered: See in **figure no. 19**

Dear pranoy,

A bid just took place. Below here is the details of the auction you created.

Auction Details :: Title - New Auction Price - 1000.00

Latest Bid Price: 1000.01

Thanks.
YAAS Team.

Fig 19 : Screen shot taken during test

Req 6.8 : the latest bidder must be notified by email when a new bid has been placed:
See in **figure no. 20**

Dear admin,

Thanks for bidding. Below here is the auction you bid for.

Auction Details :: Title - New Auction Price - 1000.00

Bid Price: 1000.01

Thanks.

YAAS Team.

Fig 20 : Screen shot taken during test

UC7 Ban an auction :

Req 7.1 : An administrator user can ban an active auction that does not comply with the usage terms of the site. See in **figure no. 21**

Req 7.2 : A banned auction is not deleted from the sy2sem. Database just change the state of banned auction object from active to banned.

Req 7.3 : The seller and all bidders are notified by email that the auction has been banned. Like before, an email will be sent to seller and bidder.

Req 7.4 : It is not possible to bid on a banned auction. It will not be shown in the list of active auctions.

Req7.5: Banned auctions are not shown in the list of auctions neither in search results.

Req 7.6 : Banned auctions are not resolved. They are no more active.

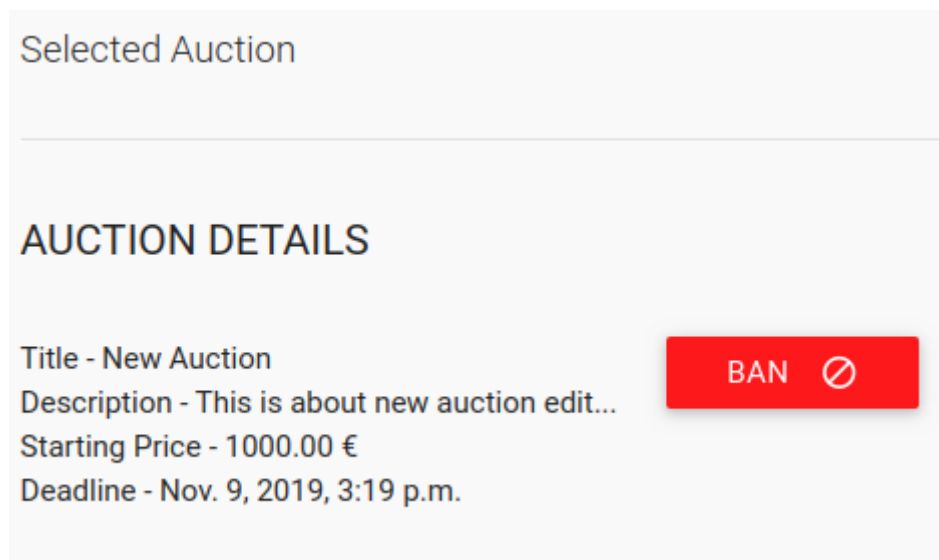


Fig 21 : Screen shot taken during test

Req 7.7 : Admin should be able to see the list of banned auctions: See in **figure no. 22**

List of Banned Auctions

BANNED BY ADMIN

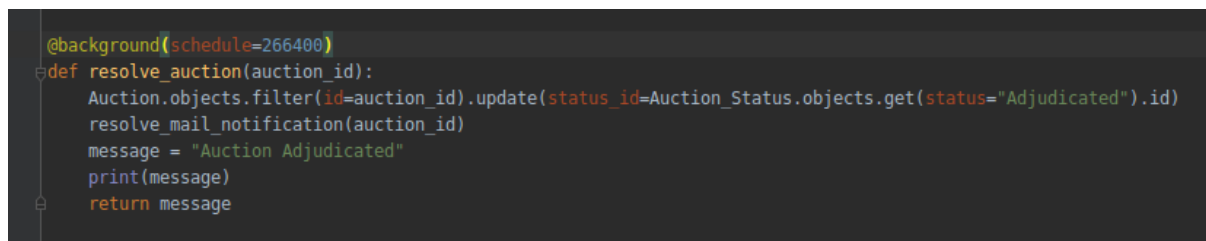
Items ...

Title	Seller	Minimum Price	Deadline
New Auction	pranoy	1000.00	Nov. 9, 2019, 3:19 p.m.

Fig 22 : Screen shot taken during test

UC8 Resolve auction :

Req 8.1 : This resolving of auctions is triggered via a GET request to /auction/resolve, which can be invoked manually by a user or by an external scheduler : : When an auction is created the day after the deadline expires, an automatic resolve process will invoke to change the database. It is done through the scheduler of background_task package we discussed before. See in **figure no. 23**



```
@background(schedule=266400)
def resolve_auction(auction_id):
    Auction.objects.filter(id=auction_id).update(status_id=Auction_Status.objects.get(status="Adjudicated").id)
    resolve_mail_notification(auction_id)
    message = "Auction Adjudicated"
    print(message)
    return message
```

Fig 23 : Screen shot taken during test

Req 8.2 : All the bidders are notified by email that the auction has been resolved: All the bidders from Bid table will be notified by email using django.core.mail package as we sent mail while creating auction.

Req 8.3 : The seller notified by email that the auction has been resolved as well.

UC9 Support for multiple languages :

Req 9.1 : all users, including anonymous users, must be able to choose their favourite language. We have two language available in this application. One is in English and another in Finnish. A registered or non-registered may choose his favourite language at any instance. See in **Figure no. 24**

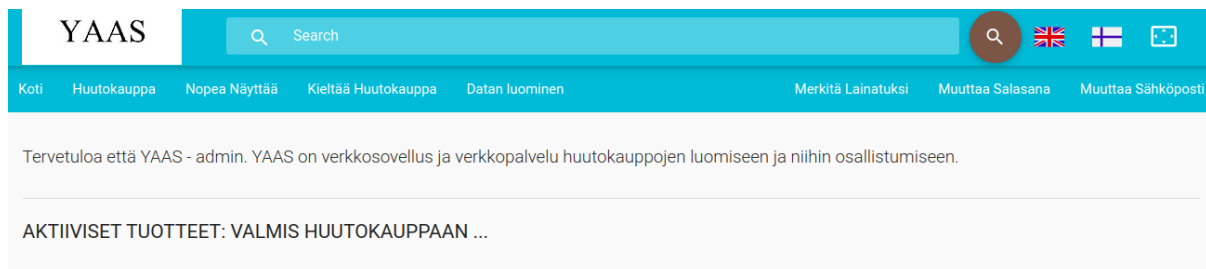


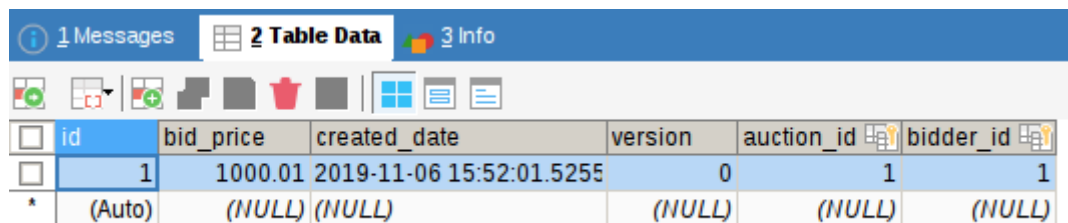
Fig 24 : Screen shot taken during test

Req 9.2 : The application remember the language preference during the user session and load language preference from session it created at the beginning with preferred language in the rest of the page's user browses.

Req 9.3 : The application store the language preference permanently for registered users by keeping track in User table against each individual user.

UC10 Support Multiple Concurrent Sessions :

Req 10.1 : The web application is able to handle multiple concurrent user sessions. As you can see in the Bid table below, it keeps track of version and each time any data is updated against the record, the version number is incremented by 1. so, if the version number does not match with the object version number stored in the table it won't allow user to update its status or in other words it will lock the object. Version check is enabled during bid and while auction description edit. See in **Figure no. 25**

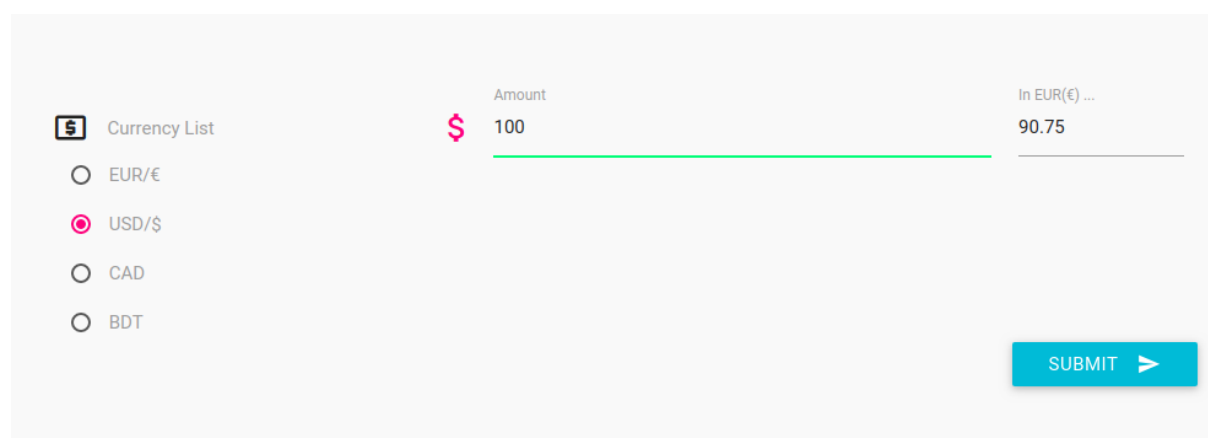


id	bid_price	created_date	version	auction_id	bidder_id
1	1000.01	2019-11-06 15:52:01.5255	0	1	1
*	(Auto)	(NULL)	(NULL)	(NULL)	(NULL)

Fig 25 : Screen shot taken during test

UC11 Support for currency exchange :

Req 11.1 : All the auctions are created by default in EUROS: While creating an auction the min price is considered to input in EUROS. See in **Figure no. 26**



Currency List

☐ EUR/€

☒ USD/\$

☐ CAD

☐ BDT

Amount

\$ 100

In EUR(€) ...

90.75

SUBMIT >

Fig 26 : Screen shot taken during test

Req 11.2 : A user should be able to visualize the prices of the auctions and bids in a different currency based on the latest exchange rate. See in **Figure no. 26**

Req 11.3 : The exchange rate for the currency should be fetched when needed from sites like <https://currencylayer.com> (free accounts available) and <http://fixer.io/> .(had to create a free account.) Through the guidance explained on how to implement and fetch json data, the currency converter created here returns currency converted in EUROS from USD, CAD and BDT. See in **Figure no. 27**

```
class CurrencyConverter:
    rates = {}

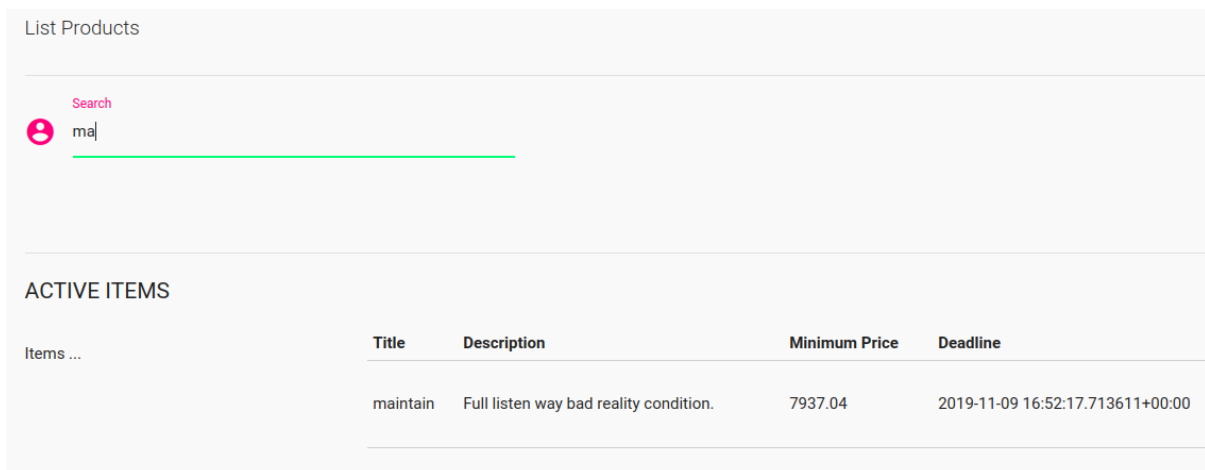
    def __init__(self, url):
        req = urllib.request.Request(url, headers={'User-Agent': 'YAAS Currency Bot'})
        data = urllib.request.urlopen(req).read()
        data = json.loads(data.decode('utf-8'))
        self.rates = data["rates"]

    def convert(self, amount, from_currency, to_currency):
        initial_amount = amount
        if from_currency != "EUR":
            amount = amount / self.rates[from_currency]
        if to_currency == "EUR":
            return initial_amount, from_currency, '=', amount, to_currency
        else:
            return initial_amount, from_currency, '=', amount * self.rates[to_currency], to_currency
```

Fig 27 : Screen shot taken during test

RESTful web service :

Req 12.1 : A user should be able to send a GET request for browsing the auctions and fetching a specific auction by ID. See in **figure no. 28**



The screenshot shows a web interface with a search bar at the top. The search bar has a magnifying glass icon and the text "ma" entered. Below the search bar, there is a table titled "ACTIVE ITEMS". The table has four columns: "Title", "Description", "Minimum Price", and "Deadline". The first row of the table shows the title "maintain", the description "Full listen way bad reality condition.", the minimum price "7937.04", and the deadline "2019-11-09 16:52:17.713611+00:00".

Title	Description	Minimum Price	Deadline
maintain	Full listen way bad reality condition.	7937.04	2019-11-09 16:52:17.713611+00:00

Fig 28 : Screen shot taken during test

Req 12.2 : A user is able to send a GET request along with a search string parameter to search auctions by their titles. Through ajax call a restful URL is hit at every key pressed on search tab. See in **Figure no. 28**

Constraint : An auction or a list of auctions should be sent back in the JSON format.

Req 12.3 : A registered user can bid on a given auction by sending a POST request with the necessary information attached in the json format. See in **figure no. 29**

Req 12.3.1 : The web service is able to verify the bidder's credentials. See in **figure no. 29**

Req 12.3.2 : The web service is able to verify bid validity. Method will be check if user and bid credentials is valid or not. See in **figure no. 29**

Req 12.3.3 : The web service can respond depending on the result of the bidding with a message containing the description of the new bid or a description of the error also in json format. URL is stated. See in **figure no. 29**

```
from django.urls import path
from .views import *

app_name = 'auction'

urlpatterns = [
    path('create/', auction_add, name='auction_add'),
    path('create/<int:id>/', auction_confirm, name='auction_confirm'),
    path('browse/', auction_browse, name='auction_browse'),
    path('list/', auction_list, name='auction_list'),
    path('banned/list/', banned_auctions_list, name='banned_auctions_list'),
    path('yaas.com/generatedata/', data_generation, name='data_generation'),
    path('generate_user/', generate_user, name='generate_user'),
    path('generate_auction/', generate_auction, name='generate_auction'),
    path('generate_bid/', generate_bid, name='generate_bid'),
    path('edit/<int:id>/', auction_edit, name='auction_edit'),
    path('bid/<int:id>/', auction_bid, name='auction_bid'),
    path('ban/<int:id>/', auction_ban, name='auction_ban'),
    path('search_auction/<search>/', search_auction, name='search_auction'),
    path('convert/<bid_amount>/<currency>/', auction_convert, name='auction_convert'),
]
```

Fig 29 : Screen shot taken during test

Additional requirements :

Testing:

Req 4.1.1 : An automated functional test for REQ9.3 - “store language permanently” has been written. An additional application testApp has been introduced and inside it django.test module is implemented to check functionality and it returns status code for 200 as success as a response of request. See in **Figure no. 30**

```

class TestViews(TestCase):

    def setUp(self):
        self.client = Client()
        self.list_url = reverse('list')
        self.auc_url = reverse('auction_add')

    def test_language_store(self):
        response = self.client.get(self.list_url)
        self.assertEqual(response.status_code, 200)

    def test_add_auction(self):
        response = self.client.get(self.auc_url)
        self.assertEqual(response.status_code, 302)

```

Fig 30 : Screen shot taken during test

Req 4.1.2 : An automated functional test for REQ 3.5 - “auction link” has been written. See in **Figure no. 30**

Req 4.1.3(NEW): An automated functional test for REQ 10.1 “concurrency” has been written in auction module inside test folder and test_views.py file. It is implemented with the module itself and using django.test module. See in **Fig no. 31**

```

from django.test import TestCase, Client
from django.urls import reverse

class TestViews(TestCase):

    def setUp(self):
        self.client = Client()
        self.bid_url = reverse('auction:auction_bid', args=[1])

    def test_bid_auction(self):
        response = self.client.get(self.bid_url)
        self.assertEqual(response.status_code, 302)

```

Fig 31: Screen shot taken during test

Test data generation program:

Req 4.2 : A data generation program is written which populates the database with at least 50 users, 50 auctions and bids for some of these auctions. The data generation programme is accessible via URL in the format myapp.com/generatedata. It can be generated by accessing the web panel that can be accessed by only admin in that web application. A menu is added on the top menu bar. User is asked to login as admin and if the user is admin a web page will be visible asking to create a user/auction and bid on those auctions with minimum 50 entries. Here the bid generation took a little extra time as it produces email for bidder and seller.

In order to generate automatic data, a python library called faker was used that can produce fake data each time an instance of that object is invoked. Web address (<https://pypi.org/project/Faker/>)

See in **figure no. 32**

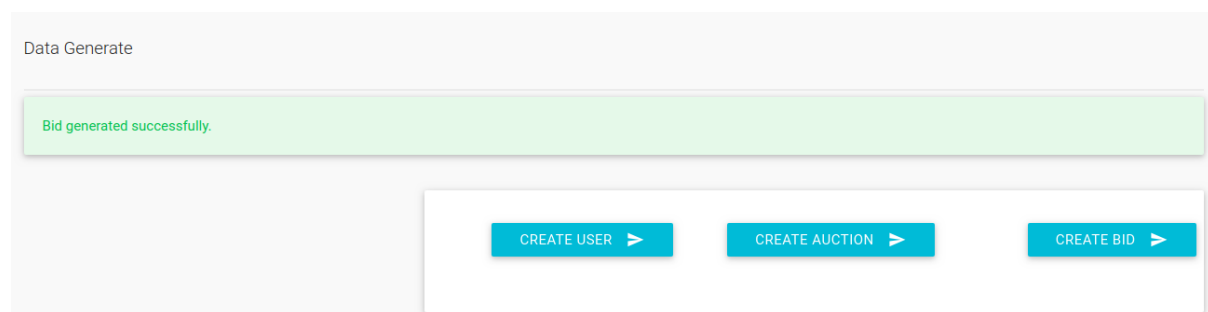


Fig 32 : Screen shot taken during test

After successful data generation, a user table will be looked like this. See in **figure no. 33**

id	username	password	email	language	role_id
1	admin	admin	admin@gmail.com	Eng	1
2	pranoy	1	p@gmail.com	Eng	2
3	matias_nykänen	matias_nykänen	toivonenjulia@puustinen.co	Eng	2
4	suvi_koskinen	suvi_koskinen	yanntila@gmail.com	Eng	2
5	seppo_kovanen	seppo_kovanen	turunenjoni@hyttinen.com	Eng	2
6	arto_virtanen-lahti	arto_virtanen-lahti	siskolassila@salonen.org	Eng	2
7	airi_savolainen-laine	airi_savolainen-laine	wvirtanen@googlemail.com	Eng	2
8	katriina_rantanen	katriina_rantanen	tiihonenaurora@suomi24.fi	Eng	2
9	tommi_salmela	tommi_salmela	kari86@hotmail.com	Eng	2
10	tapani_kyllönen	tapani_kyllönen	heidi00@liukkonen.org	Eng	2
11	ellen_pehkonen	ellen_pehkonen	matilaessi@juvonen.com	Eng	2
12	sakari_kuisma-nikula	sakari_kuisma-nikula	maria31@kangas.com	Eng	2
13	irja_häkkinen	irja_häkkinen	zniemi@luukku.com	Eng	2
14	susanna_lehtinen	susanna_lehtinen	qvurinen@luukku.com	Eng	2
15	marko_järvinen	marko_järvinen	xkoivula@leppanen.org	Eng	2
16	anneli_hyvärinen-simola	anneli_hyvärinen-simola	juhanioksanen@tanskanen	Eng	2
17	juhani_mustonen	juhani_mustonen	peltonenpauliina@lahti.com	Eng	2
18	antero_pekkari	antero_pekkari	olepisto@rossi.fi	Eng	2
19	petteri_juntunen	petteri_juntunen	maria98@suomi24.fi	Eng	2
20	anne_koskinen	anne_koskinen	virtanenotto@kuusisto.com	Eng	2
21	jaakko_karvonen	jaakko_karvonen	keskinenjoannes@venala	Eng	2
22	minna_pennanen	minna_pennanen	ulahti@kanerva.com	Eng	2
23	ari_aalto	ari_aalto	koivistochristian@heino.org	Eng	2
24	tapio_eriksson	tapio_eriksson	laaksonenanita@surffi.net	Eng	2
25	jussi_karlsson	jussi_karlsson	vuokkovesterinen@pekkari	Eng	2
26	eero_hirvonen	eero_hirvonen	vainioviljami@vuorinen.fi	Eng	2
27	päivi_nevala	päivi_nevala	heidinuutinen@luukku.com	Eng	2
28	samuli_laine	samuli_laine	marianne48@hwarinen.cor	Eng	2

Fig 33: Screen shot taken during test

Technical Requirements :

- **Python 3, www.python.org**
- **Django 2.2, www.djangoproject.org**
- **Git - www.github.com**

Development Tools:

1. PyCharm, Professional Edition editor, www.jetbrains.com/pycharm/
free license at <https://www.jetbrains.com/student/>

2. SQLyog to access mysql database

3. Operating System – Linux Ubuntu 16.04 LTS

Note : Both the admin username and password for the web application is set as -
“admin”.

Browser Used to Test Requirements: Chromium Web Browser for Ubuntu.

List of Python Packages:

1. Django - 2.2.5
2. django-background-tasks 1.2.0 (<https://pypi.org/project/django-background-tasks/>)
3. django.core.mail for send_mail
4. Faker 2.0.3 (<https://pypi.org/project/Faker/>)
5. django-rest-framework 3.10.2 (<https://pypi.org/project/django-rest-framework/3.10.2/>)
6. pip – 19.2.3 (<https://pypi.org/project/pip/19.2.3/>)
7. mysqlclient – 1.4.4 (<https://pypi.org/project/mysqlclient/1.4.4/>)

Additional libraries

```
from django.shortcuts > render, redirect
from django.contrib > messages
from datetime > datetime, timedelta
socket
urllib.request
json
from django.http > JsonResponse
from django.views.decorators.csrf > csrf_exempt
random
```


Deliverables :

Git repository - <https://github.com/>

The github includes:

1. Full project source codes
2. Instructional README.md file for manual understanding
3. requirements.txt file for installation of additional python packages for project dependencies.
4. That report is also included in the root folder as readme.pdf file. The report is a PDF document.
5. sql script named yaas_database.sql to create database with additional default data that needed to be stored.

Report contains:

1. My full name and student number at Åbo Akademi.
2. List of implemented requirements (that successfully passed the tests and do not crash in the browser)
3. Browser used to test the application
4. List of Python packages used beside django and their version.