Development of Web Service

Course Project Report

Yuan Gao(38185)

1. Introduction

This report describes the project work for the course Development of Web Applications and Web Services. The goal of the project is to develop the YAAS (Yet Another Auction Site)web application.

2. Implemented Requirements:

2.1. UC1 create an user account

- Any anonymous user can create a new regular user account;
- Django admin interface are enabled;
- Administrator user account are created via Django admin interface;

2.2 UC2 Edit account information

- authenticated user can change his/her email, and password;

2.3 UC3 Create a new auction

- Any registered user can create a new account;
- The created auction has the following information:
  - seller: the user who created the auction;
  - name: the name of the auction;
  - description: description of the auction;
  - minprice: starting price of the auction;
  - deadline: the end date of the auction, minimum duration is 72 hours from the time the auction is created.
- Confirm by email: after create the auction, the seller(who create the auction) receive a confirmation email. The seller must click the confirmation link in the email to activate the auction, otherwise the auction is inactivated and will not show on the auction list.
- OP1(optional features):

- the confirmation email also include a link, which allow the user to see the created auction without login;
- auction has a life cycle: active, banned, due, adjudicated:
- when the auction is created, all these four features is False;
- active: after the user confirm the email, the active become True; the auction will keep active until its deadline, or it is banned;
- banned: An administrator can ban the auction, the banned auction can’t be bid, and will not show on the auction list.
- due: auction due is True after the deadline ends;
- adjudicated: after due turns True, the auction is adjudicated automatically by the system, and a winner is selected;

2.4 UC4 Edit the description of an auction

- The seller of an auction can change the description of the auction;

2.5 UC5 Browse and Search auctions

- Both anonymous and registered users can browse the list of the auctions and search auction by auction name;

2.6 UC6 Bid

- Any registered user can bid for an active auction, except the seller;
- minimum bid increment is 0.01;
- the seller can’t bid on his/her own auctions;
- show the most recent description of the auction before accepting any bids;
- a new bid should be greater than min price and any previous bids;
- the seller of the auction, new bidder, the previous bidders will notified by email that a new bid has beed registered;
- OP2:
    - soft deadline: if a bid is made within five minutes before the deadline, the acution’s deadline will be extends automatically additional 5 minutes;

2.7 UC7 Ban an auction

- An administrator can ban an active auction;

- The banned auction is not deleted from the system, it will not show on the auction list or search result;
- The banned auction will not be resolved;
- The seller and all the bidders are notified by email that the auction is banned;

2.8 UC8 Resolve auction

- After the deadline of the auction, the system automatically resolve the auction and elect the winner.
- All bidders and the seller are notified by email that the auction has been resolved;

2.9 UC9 Support for multiple languages

2.10 UC10 Support multiple concurrent sessions

2.11 UC11 Support for currency exchange

2.12 WS1 Browse and search via API

2.13 WS2 Bid via API

2.14 TR1 Database fixture

2.15 TR2 Automated functional tests

3.list of Python packages used beside django

- amqp==2.2.2
- billiard==3.5.0.3
- celery==4.1.0
- dj-database-url==0.5.0
- Django==2.0.3
- django-celery-beat==1.1.1
- django-crispy-forms==1.7.2
- django-dynamic-fixture==2.0.0
- django-filter==1.1.0
- djangorestframework==3.7.7
- djangorestframework-jwt==1.11.0
- gunicorn==19.7.1

- kombu==4.1.0
- Markdown==2.6.11
- olefile==0.45.1
- Pillow==5.0.0
- psycopg2==2.7.4
- psycopg2-binary==2.7.4
- Pygments==2.2.0
- PyJWT==1.6.1
- pytz==2018.3
- six==1.11.0
- vine==1.1.4

3.Admin username and password for your DB

username:**gy**

password:**password123**

4. How does your application implement session management.

- Edit MIDDLEWARE setting and make sure MIDDLEWARE contains 'django.contrib.sessions.middleware.SessionMiddleware’.
- Make sure 'django.contrib.sessions' is in your INSTALLED_APPS setting

5. How do you implement the confirmation form in UC3.

- In auctions.models , define send_activation_email method, after the seller create the auction, the system will send a confirmation email to the seller. The method will generate a unique activation_key for each created auction, this activation_key will be used in the link in confirmation email; in terms of sending email, the system user the send_mail and send_mass_mail method;

- In the confirmation email, there is a link for seller to active the auction. The link include the generated activation_key, which identify each auction; If the user click the link, it will raise the activate_user_view method to active the auction.

6. UC8 Resolving bids must be initiated automatically (and not by a user). How do you initiate this?

- Implement celery , django-celery-beat, call the auction_save method periodically, like every 10 seconds;
- auction_save automatically save all the activated auctions every 10 seconds, if some activated auctions pass the deadline, set the auction.active to False and auction.due to True;
- auction_save also automatically save all the due auctions every 10 seconds, if there are due auctions, set the auction.adjudicated to True, and elect the winner;

7. How do you avoid possible concurrency problems, specially in UC6?

- the system will always show the most recent auction description before accepting the bid from the user:
  - give the auction model a feature called version; every time the seller updated the description of the auction, the version of the auction will add one; (auction.version+=1);
  - when a bidder is in the process of bidding an auction, if the seller change the description of the auction, when the bidder place the bid, the bidding page refreshed, there is a warning message shows that 'The auction description has been changed, please replace your bid again.', and the description of auction is updated to the newest version;
- new bid should always greater than any previous bid in the same auction:
  - if the bidder's bid is not greater than the maximum bid, when the bidder place the bid, the bidding page refreshed, there is a warning message shows that ''A bid has been

make before you. Please replace new bid.'', and the current_bid of the auction is updated to the recently created bid.

8. Present the REST API for your service. Is your service RESTful? What resources do you expose over the API and what is the URI to access them. Include concrete examples of HTTP requests and responses for bidding in an auction.

This YAAS system is RESTful.

the list of activated auctions http://127.0.0.1:8000/api/auctions/

the list of bids and create new bid http://127.0.0.1:8000/api/bids/

HTTP requests:

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

{
    "name": "Bid Create Api",
    "description": "",
    "renders": [
        "application/json",
        "text/html"
    ],
    "parses": [
        "application/json",
        "application/x-www-form-urlencoded",
        "multipart/form-data"
    ],
    "actions": {
        "POST": {
            "bidder": {
                "type": "field",
                "required": false,
                "read_only": true,
                "label": "Bidder"
            },
            "auction": {
                "type": "field",
                "required": true,
                "read_only": false,
                "label": "Auction"
            },
            "bid": {
                "type": "decimal",
                "required": true,
                "read_only": false,
                "label": "Bid"
            }
        }
    }
}
```

HTTP responses:

```
HTTP 201 Created
Allow: GET, POST, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

{
    "bidder": 1,
    "auction": 5,
    "bid": "11.00"
}
```

9. A description of your functional tests, what are you testing and why.

there are two tests files each in auctions and bids modules;

- In the tests file in auctions module, it test the AuctionModel, AuctionCreateForm
  - AuctionModelTest: test the Auction model , like the text users for labels, the size of the field allocated for the text, because these are part of the design that could be broken/changed in future; also test_email_send to test if email be will send if an auction is created;

- AuctionCreateFormTest: test the AuctionCreateForm, like the labels, and custom validations to validate the data in the field is valid;

- In the test file in bids module, it tests the BidModel , BidCreateForm

  - BidModelTest: test the Bid model. like the labels, and also the test_clean method to test the clean method in model; also test_email_send method to test if email will be send if bid is created;

  - BidCreateFormTest:test the BidCreateForm, like the labels;

  - BidCreateViewTest: test the BidCreateView, test_description_change tests that if the system will always show the most recent auction description before accepting the bid from the user; test_bid_greater_than_any tests that if new bid always greater than any previous bid in the same auction;

10. How have you implemented the language switching mechanism?

- in the setting.py, set:

```
LANGUAGE_CODE = 'en'
LANGUAGES = (
    ('en', _('English')),
    ('zh-hans', _('Chinese')),
    ('fi', _('Finnish')),
)
LOCALE_PATHS=(
    os.path.join(BASE_DIR, 'locale'),
)
```

- in the urls.py, set the pages which need to be translated in this following pattern:

```
urlpatterns += i18n_patterns(
    path('', HomeView.as_view(),name='home'),
    path('my_bids/',MyBidListView.as_view(),name='my-bids-list'),
        ……
        )
```

- in the template, implement the following code to create a select form, in which users can select the want language;

```
        <form action="{% url 'set_language' %}" method="post">{% csrf_token %}
                <input name="next" type="hidden" value="{{ redirect_to }}" />
                <select name="language" onChange="this.form.submit()">
                {% get_current_language as LANGUAGE_CODE %}
                {% get_available_languages as LANGUAGES %}
                {% get_language_info_list for LANGUAGES as languages %}
                {% for language in languages %}
                        <option value="{{ language.code }}"{% if language.code ==
LANGUAGE_CODE %} selected{% endif %}>
                        {{ language.name_local }} ({{ language.code }})
                        </option>
                {% endfor %}
                </select>
        </form>
```

- in the template where there are scripts needed to be translated, add `{% load 118n %}` to the top of the template, and `{% trans %}` to a constant string or variable;
- create message files: `django-admin makemessages -l zh_hans`
  - this will create a .po file which contains messages mapping between translation strings and the actual translated text;
- compile message file: `django-admin compilemessages`

11. Specify the location data generation program and how it can be used (invoked)

In this design, the data generation code is in tests file in auctions models.

```
for i in range(0,50):
        instance_user = G(User,username='test_user_'+str(i))
        instance_auction=G(
                Auction,
                seller=instance_user,
                name=instance_user.username+'auction',
                description=instance_user.username+'description',
                minprice=i+0.1,
                deadline=timezone.now()+timezone.timedelta(hours=72+i),
                version=0,
                activated=True,
                banned=False,
                due=False,
                adjudicated=False,
        )
for i in range(0,10):
        instance_bid=G(
                Bid,
                bidder=User.objects.filter(pk=310+i)[0],
                auction=Auction.objects.filter(pk=220+i)[0],
                bid=Auction.objects.filter(pk=220+i)[0].minprice+Decimal(0.1),
        )
```

Before test, comment out all the other test method and leave only the code in the file(in the code, using G from django_dynamic_fixture to generate data) run the test file, and 50 users, 50 auctions and 10 bids are created in the system. Then use dumpdata to create fixtures: users.json, auctions.json, bids.json;

After generating all the data, comment out the code, and do the other test;