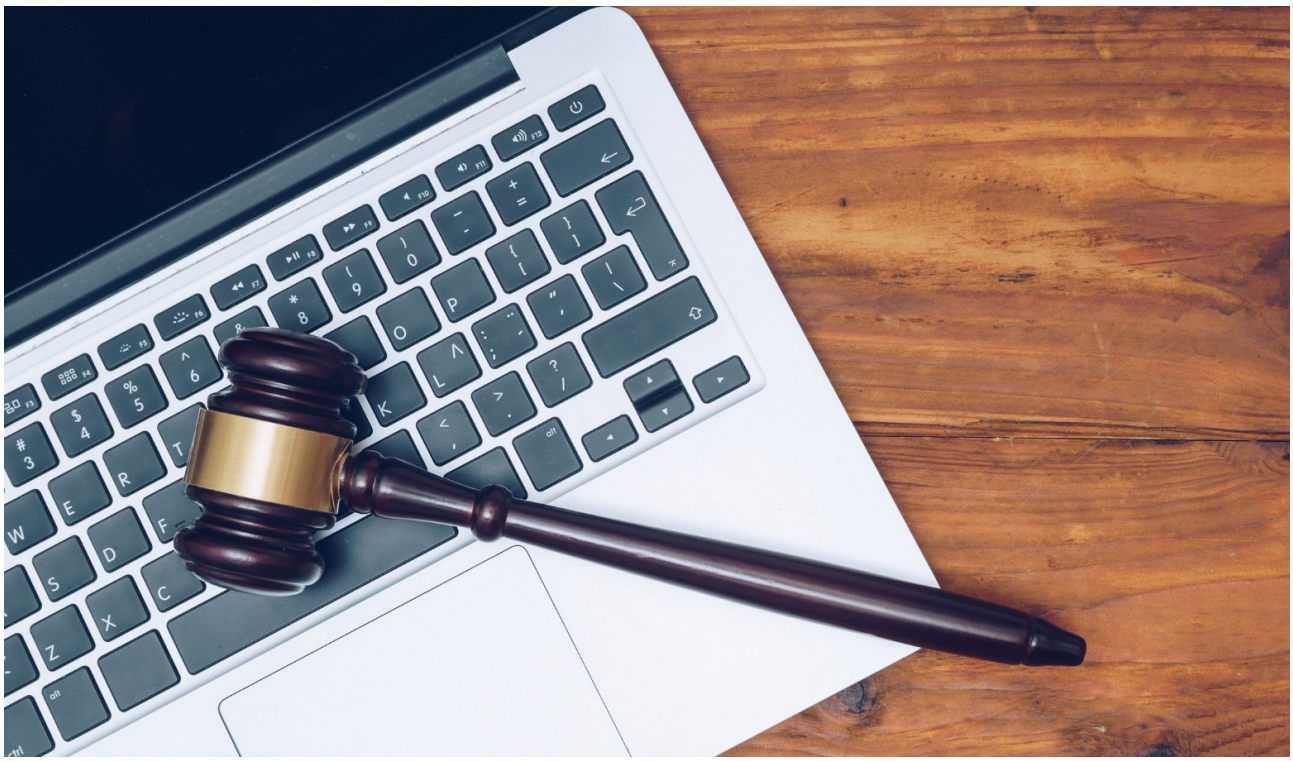
Online Auction

Magradze Nikoloz



Auction Online

The project is based on Django framework, using bootstrap, javascript, crispy and other web technologies. Main programming language used is python.

The project is about Online Auction. The site gives possibilities to sell and buy mobile and immobile assets. There are four types of users:

1) Non authenticated
2) Base user
3) Premium user
4) Admin

Non authorized users can access the site without registration. They can see all the active auctions. See their price, photos, comments, seller photos, etc. Obviously they don’t have any profile so they cannot participate in auctions, neither sell any product. They are guests of the site.

Once a guest registers on the site non authenticated user becomes **base user**. Base user has to insert all needed information to register on the database as the first name, last name, email and password. Authenticated users can visit displayed auctions, write comments and bid. The web app traces all the auctions where the base user tries to participate and on the main page brings in the first position the related products. If the user is only registered there's no profile connected to it's user account. In order to be able to create auctions the user has to compile it's personal information as the address and likely can upload the main photo. If the user doesn't provide the photo the system will assign a default image. After that the user's created profile, can be created at most three auctions each one with only one product photo. If it's needed to upload more photos there is a premium account.

If a user has the necessity to create more than tree trial auctions and upload up to four images for every auction, there is a **premium account** for these kind of sellers. A premium account is activated after the online payment to the site. Premium user account gives the possibility to start unlimited number of auctions.

Admin is the django system admin. Admin has all permissions. This user can delete, modify and add every instance of every class. There's no specific Admin view but is used django provided one.

As described above registered users initialize auctions. Every auction is connected to a one and only one user-profile, that's why if the user has no profile can't be initiated an auction even if the user is registered. The auction has the start and end date, start date obviously is equal to actual date. Auction has the description, price, best bidder which is the last user who gave the highest offer, minimum price increment - set to control lots of small amount bids. Auctions instances have comment section and a section where there is possible to see who's selling and seller's photo too. On the page which shows the product is possible to see product images. Once the product's end date is expired the product becomes active no more so it will disappear from the main product page and will appear in auction winner's "won auctions" section. So after the payment the product is sent to the buyers house[1]. If the user took part in the auction but "lost" it, the product will add in "lost auctions" section. There is also to possibility to see which ones are own auctions and if desired remove them.

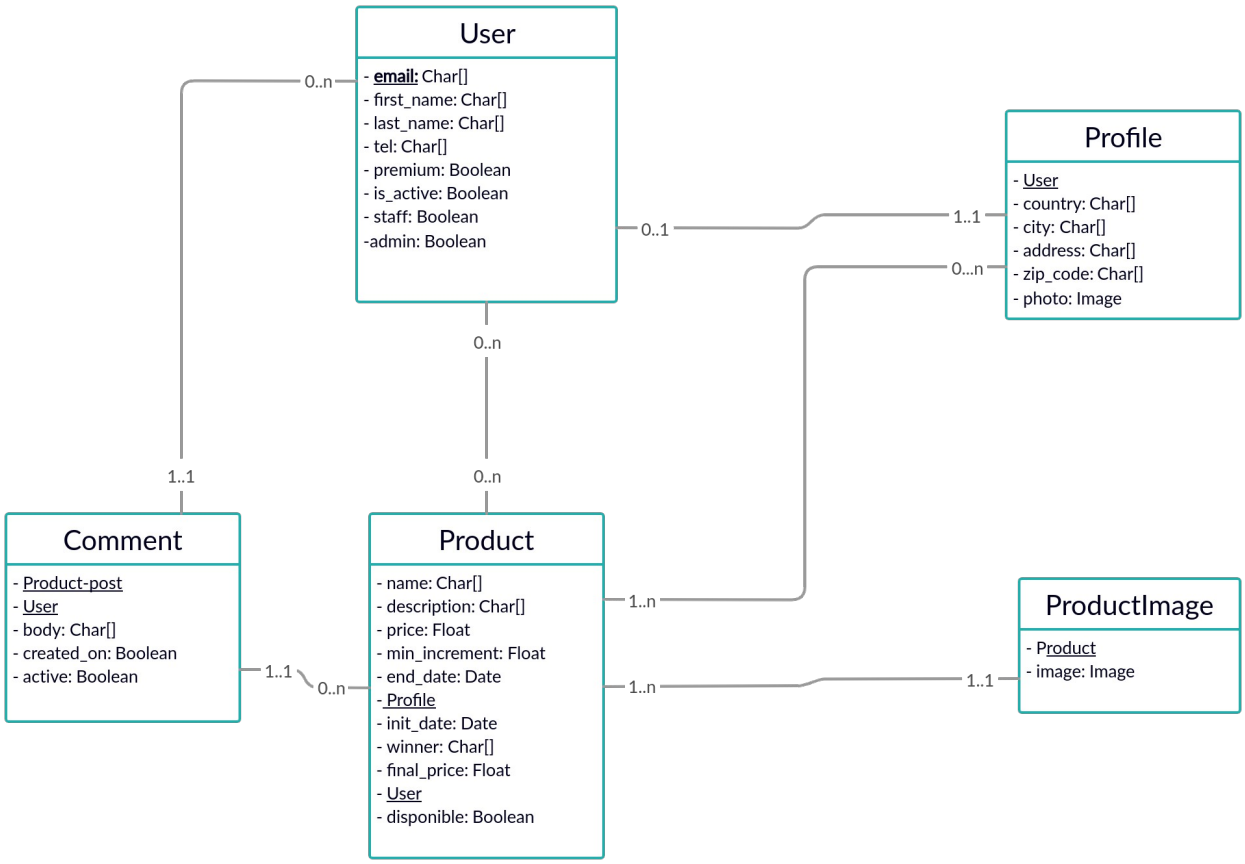
As the products are added on the platform it's expected that the auctions number will grow on the main page so there is a search field and a paginator which helps us navigate on the web and not to load all the products on the home page.

---

1 *For simplicity and because the project doesn't deal with payment and sending technology the payment and delivery system is not part of the project, but if there is necessity to deploy the site it's enough to add these two functions as the system would integrate them perfectly.*

Project uses Sql Lite database. This is well integrated and native Django DB system. Also unittests with SQLite is simple and efficient.

Auction class/DB relations is designed as follows :



The authentication system is based on Django auth system with small changes. As the site is not Social Network there is left only email with double purposes, as the primary key value(even if there is autoincrement id in the DB) and as the actual email field. User table is the table which communicates with quite all tables, that's because the user is the main actor in the system.

User – Profile : Are connected with one to one relation. A user can have at most one profile and a profile if exists should be related to one and only one user.

User – Product : User can be interested in (0,n) products and a product can have (0,n) people interested in it. Attention! Don't read the connection between user and product as the "~~user creates products~~" this is wrong. Read as user "is interested in".

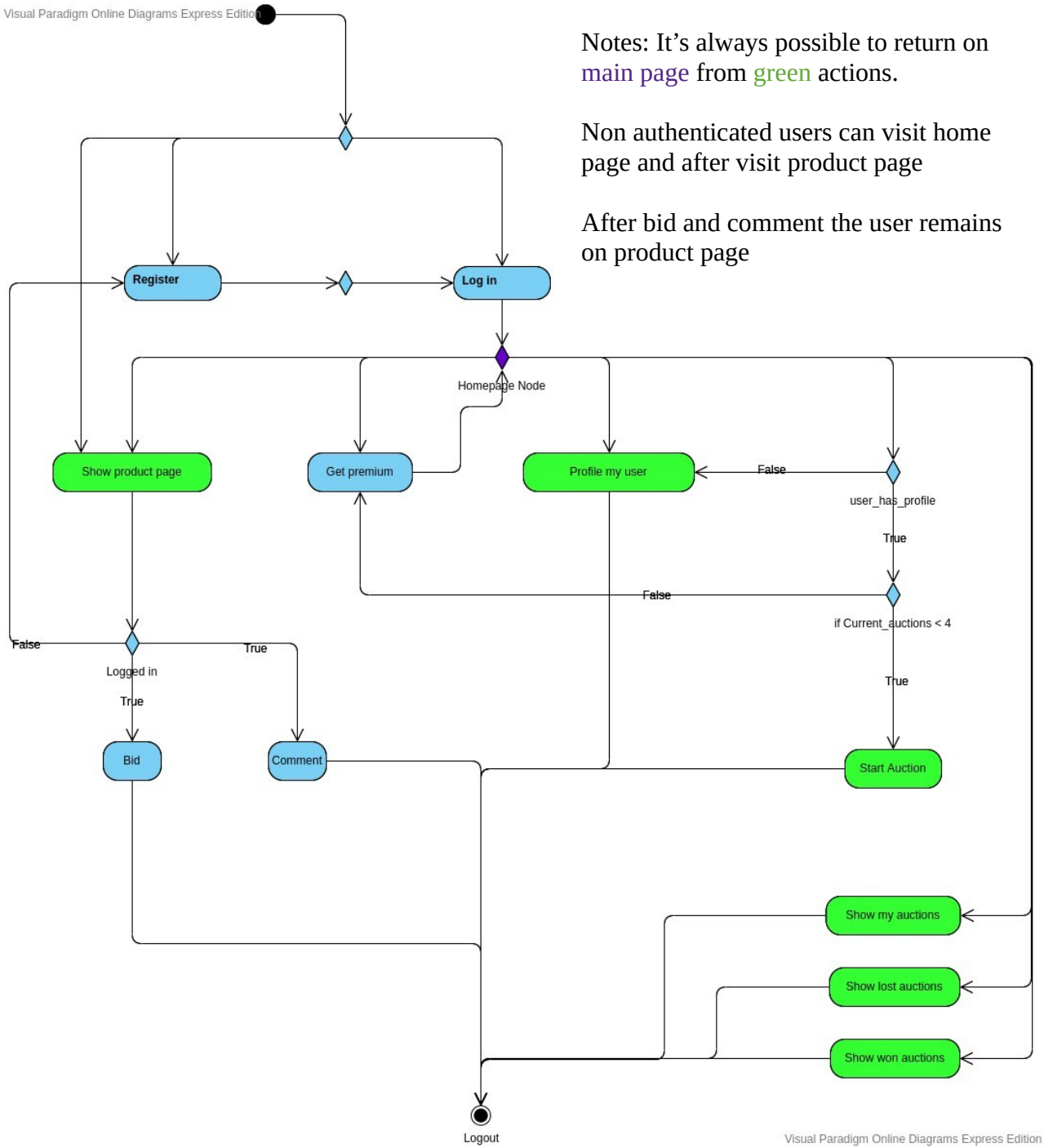
User – Comment : A user can write 0 or more on each Auction and a comment can be written by one and only one user.

The user table contains all necessary fields to get the main information about it. Contains also premium and admin fields and there usage is quite intuitive. Staff and is_active are not actually used in the project but it's best practice to have them in the Django Auth system, is_active can be useful in case of user delete.

Profile – Product : A profile creates n products and a product is created by only one profile.

A **product** has from one to n images and from 0 to n comments.

To explain the main logic there is shown a kind of activity diagram. Note that as the site has always on navbar there is always direct way to switch from one page to another so drawing every single line would become impossible to read, so was kept only the activities that can clarify doubts about some particular processes. Don't take it as a complete Activity diagram.



Notes: It's always possible to return on main page from green actions.

Non authenticated users can visit home page and after visit product page

After bid and comment the user remains on product page

Front end technologies

The front end is done with HTML, CSS and JavaScript. Forms are done with Bootstrap and crispy forms, they have nice look and are easy to use. In addition Bootstrap does very good job to make the site usable from the mobile device. Front end objects are handled with Flex.

Django apps

There are two main apps in the project: users and products.

Users contain profile management and products handles comment section too. At first there was a tempting desire to create a new app for comments but this class is so connected to the product and so small that it was added to product app.

Users app:

Redefines Django auth system using UserManager. Overloads create_user with:

- create_user (with different parameters)
- create_staffuser
- create_premiumuser
- create_superuser

Email is used as username to authenticate on the site.

The project implements password recovery system confirming the link sent to the email. (Actually the program is in debug mode so the to reset the password is enough to click on the link sent to the console)
Profile class is extended with PrpfileManger too and has a method create_update_Profile which creates the instance if there is no profile for user or updates info if the user has already had.

Products app:

Products app handles products, comments and Product Images.

Products model has two methods

- get_product_age
- time_left_to_end

These two function will be the object of unittesting so there will be better explanation what these two methods do.

Every product has comment section and registered users can add comments. Comments order is based on creating date. Comments have methods decorated with @register.filter :

- get_name
- get_surname
- get_time

so these methods can be called from the html file django template.

Main logic of Auction application

As already described every user can initiate the auction and every user can take part of it. When an auction instance is created there is no winner and if the user offers higher bid as the initial price is this user becomes the actual winner. So this “winner substitution” system goes on until the expiration time arrives. For the implementation choices every auction expires at 00:00 of the expiration day. But there is a problem. There should be a scheduler which controls if there is any expiration on given time. In this case we use Celery.

Celery and Redis

“Celery is a task queue/job queue based on distributed message passing. It is focused on real-time operation, but supports scheduling as well.
The execution units, called tasks, are executed concurrently on a single or more worker servers. Tasks can execute asynchronously (in the background) or synchronously (wait until ready).” - link

Auction uses celery to create task queues to schedule periodical expiration checks and uses Redis which is a performant, in memory, key-value data store. Redis is used to store messages produced by the application code describing the work to be done in the Celery task queue.

In this case the code is simple:

tasks.py

```
@periodic_task(run_every=timedelta(seconds=20))
def every_20_seconds():

    all_products = Product.objects.all()
    for p in all_products:
        if p.end_date < datetime.date(timezone.localtime(timezone.now())):
            p.disponible = False
        else:
            p.disponible = True
        p.save()
    print(all_products)
```

This is very simple job done every 20 seconds by Celery and Redis. Redis server can be used on host PC but it requires server installation and configuration. Heroku.com offers Redis service freely. So the configuration is as simple as register on the site and request a link and insert in the *settings.py*:

CELERY_BROKER_URL = **'redis://h:pc0e01da54eb02a2b79626baa69830...”**

And with the command celery -A Auction worker -B every_20_seconds() task begins to be executed every 20 seconds.

Once Celery marks the product as not disponible(not available) this product is not shown any more on site’s home page but it’s moved in winner’s “won auctions” section. People who bided and lost will find this auction in their “lost auctions” section.

Images and media files

Every profile can have one user image and every product can have up to four images. In the case of profiles the solution is as simple as add a ImageField to the model, but for products there is a different model connected to the profile which saves every single photo.

Static images are saved in static files but images uploaded by user are saved in media files as Django documentation suggests.

Search

Search system uses very simple but efficient Django SQLite DB command.

```
if query != "":
    products = Product.objects.filter(name__contains=query)
    products = products.union(Product.objects.filter(description__contains=query))
```

This command filters every product name or description which contains the searched word.

Recommendation system

Recommendation system works only for registered users. When the user goes to the main page this triggers a function that looks up products in which the user is or was interested and displays them in the first 3 places in homepage. Searching is based on Jaro–Winkler distance. Jaro–Winkler distance is a measure of edit distance which gives more similar measures to words in which the beginning characters match.

The algorithm is following:

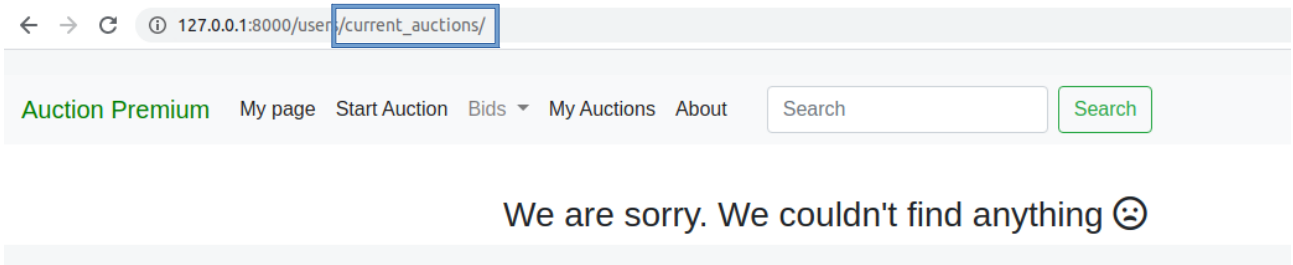
```
for i in all_products_names:
    for j in user_products_names:
        d = round(textdistance.jaro_winkler(i, j), 4)
        m = min([t[2] for t in list])
        if d > m:
            l = [j,i, d]
            for k in range(3):
                if m == list[k][2]:
                    list[k] = l
                    break
```

```
from django.db.models import Q
list = Product.objects.filter(Q(name = list[0][1]) | Q(name = list[1][1]) | Q(name = list[2][1]), disponible=True )
return list
```

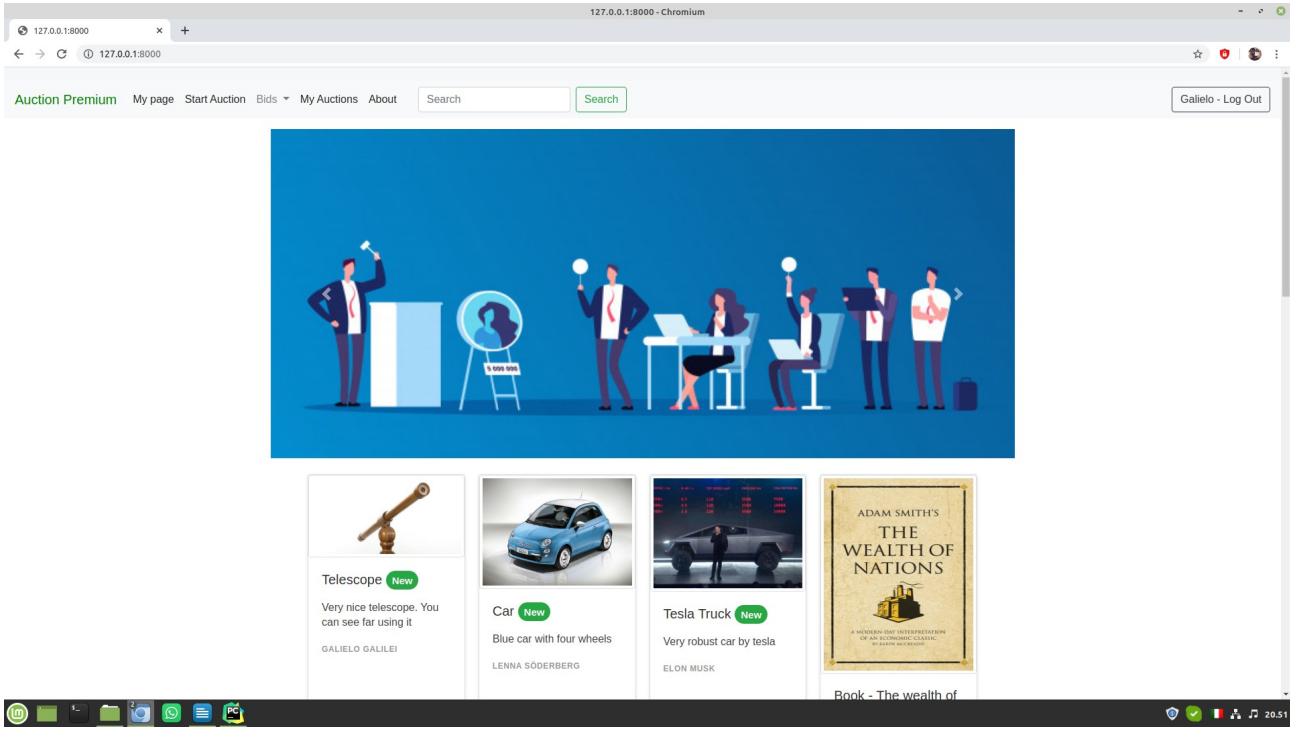
Test of recommendation system:

In the database there are some articles and among them there are “Tesla car”, “Tesla Truck “ and “Car”. My current auctions is empty because I didn’t bid on any auction.

Initial situation:



I bidded on “Tesla car” so in the first three positions appeared “Tesla Truck” and “Car”



So the second and third positions are occupied by related products.

Unittesting

Product module has 2 methods

- get_product_age()
- time_left_to_end()

The first function returns 0 if the auction was posted in the last 7 days. Returns 1 if the auction was posted from 7 to 30 days, returns 2 for more than 30 days. In any other case returns -1.

The second function returns the time that is left to the end of the auction.

Here are reported some tests made on the first function:

We can test the function inserting many possible dates of init and end of auction and see if it crashes.

For example:

```
class AgeMethodTest(TestCase):
    def test_get_product_age_today(self):
        init_date = datetime.date(datetime.now())

        product = Product(name="Test",description="Test_description",price = 300, min_increment=1,profile=None,init_date = init_date )
        self.assertEqual(product.get_product_age(),0,"Should return 0")

    def test_get_product_age_within_week_two_days_ago(self):
        delta = timedelta(days=2)
        days_ago = datetime.now() - delta
        init_date = datetime.date(days_ago)

        product = Product(name="Test",description="Test_description",price = 300, min_increment=1,profile=None,init_date = init_date )
        self.assertEqual(product.get_product_age(),0,"Should return 0")

    def test_get_product_age_within_week_seven_days_ago(self):
        delta = timedelta(days=7)
        days_ago = datetime.now() - delta
        init_date = datetime.date(days_ago)

        product = Product(name="Test",description="Test_description",price = 300, min_increment=1,profile=None,init_date = init_date )
        self.assertEqual(product.get_product_age(),0,"Should return 0")
```

And tested also the second function, altering init date and end date:

```
class TimeLeftMethodTest(TestCase):
    def test_time_left_to_end(self):
        delta = timedelta(days=5)
        days_ago = datetime.now() - delta
        days_after = datetime.now() + delta
        init_date = datetime.date(days_ago)
        end_date = datetime.date(days_after)

        product = Product(name="Test",description="Test_description",price = 300,end_date=end_date, min_increment=1,profile=None,init_date = init_date )
        self.assertGreaterEqual(product.time_left_to_end(),0,"Days left shouldn't be less than 0")

    def test_time_left_to_end_inverted(self):
        delta = timedelta(days=5)
        days_after = datetime.now() - delta
        days_ago = datetime.now() + delta
        init_date = datetime.date(days_ago)
        end_date = datetime.date(days_after)

        product = Product(name="Test",description="Test_description",price = 300,end_date=end_date, min_increment=1,profile=None,init_date = init_date )
        self.assertGreaterEqual(product.time_left_to_end(),0,"Days left shouldn't be less than 0")
```

None of these tests produced errors.

View test:

Tested current auctions. This view returns the auctions in which the user is interested. So tried to access with no user, authenticated, than tried to display current auctions when one product is bided or when more than one is bided. Tried to see what happens when the user is registered and doesn't have profile.

No errors found.

Some screenshots from the site: