

2020

Development of a Cloud Software as a Service: Online Auctioning System

Microservices principles: Automation and Integration

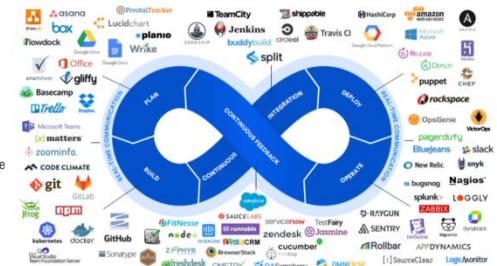
Follow a:

- Plan
- Build
- Integrate
- Deploy
- Operate
- Feedback

Continuous deployment

- We will talk about this in the future...

Integrate easily with a selection of state-of-the-art tools to use!



Contents

Contents	2
1 Introduction.....	4
1.1 General.....	4
2 Solution Development Approach	5
2.1 Step 1: Initial design as single integrated application.....	5
2.2 Step 2: Split into separate apps for “authorization and authentication” and “Auctioning RESTful APIs”	5
3 Software applications/Technology used: Testing and Development	6
4 How do the two decoupled apps communicate?.....	7
5 Phase B: Development of Authorization and Authentication services	8
6 Phase C: Development of Auctioning RESTful APIs	9
6.1 Database Design.....	9
6.2 Constraints Identification and implementation.....	10
6.3 Query design	11
6.4 API’s developed.....	12
6.5 Exposing the API’s	13
7 Phase D: Development of a testing application and Test results	14
7.1 TC1: Olga, Nick and Mary register in the application and are ready to access the API	15
7.2 TC2: Olga, Nick and Mary will use the oAuth v2 authorisation service to get their tokens .	16
7.3 TC3: Olga makes a call to the API (any endpoint) without using a token. This call should be unsuccessful as the user is unauthorised	16
7.4 TC4: Olga adds an item for auction with an expiration time using her token.	16
7.5 TC5: Nick adds an item for auction with an expiration time using his token.	17
7.6 TC6: Mary adds an item for auction with an expiration time using her token.	18
7.7 TC7: Nick and Olga browse all the available items, there should be three items available.	19
7.8 TC8: Nick and Olga get the details of Mary’s item.....	19
7.9 TC9: Mary bids for her own item. This call should be unsuccessful, an owner cannot bid for own items.....	20
7.10 TC10: Nick and Olga bid for Mary’s item in a round robin fashion (one after the other). ...	21
7.11 TC11: Nick or Olga wins the item after the end of the auction.	21
7.12 TC12: Olga browses all the items sold.	22
7.13 TC13: Mary queries for a list of bids as historical records of bidding actions of her sold item	23
8 Acknowledgments and Future Work	24
9 References	25

1 Introduction

1.1 General

This report summarises the work done to develop an online auctioning app with access provided to registered users only.

The overall solution architecture is given in Figure 1. It includes two separate microservices – one for handling “Authorization and Authentication” and the other for the “Online Auctioning RESTful APIs”. Communication is handled via well-defined RESTful API endpoints created for each service.

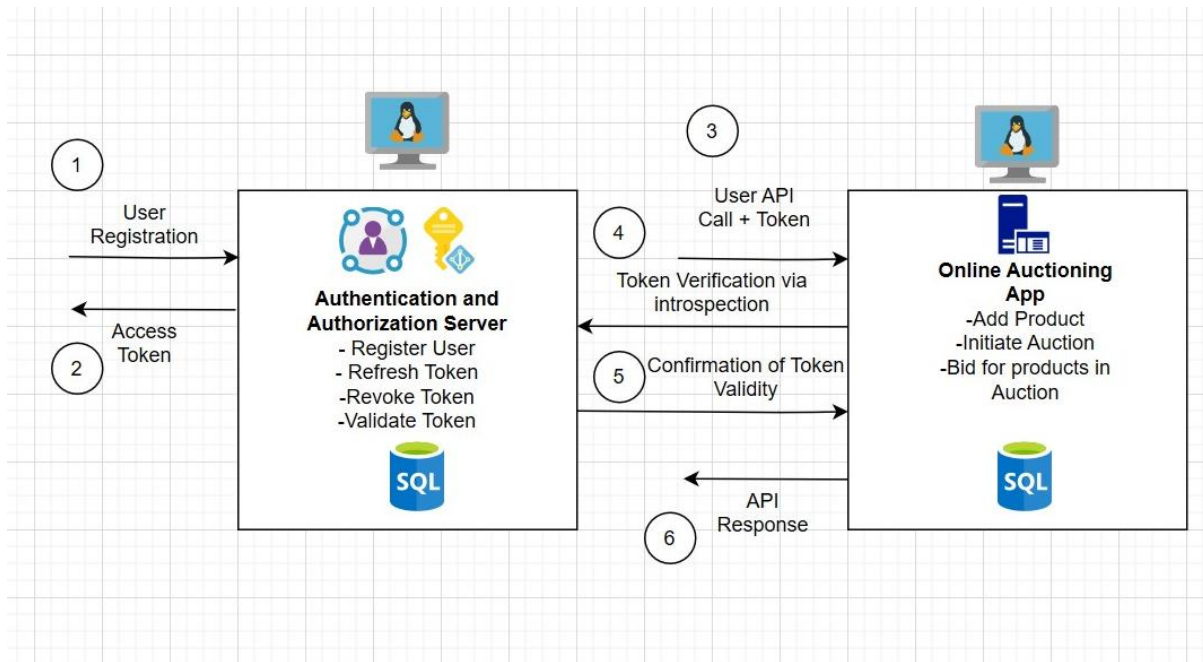


Figure 1: Overall Solution Architecture

A typical flow through the application is as follows:

1. User registers with the Authorization and Authentication via a dedicated API
2. An access token is provided
3. The user makes a call to any of the API's defined for Online Auctioning app passing in the token with call.
4. The token is first verified. This involves the Online Auctioning app making a call back to the “Authorization and Authentication” server.
5. The “Authorization and Authentication” server confirms the validity of the token.
6. An appropriate response is provided.

```
student@cc: ~/cw-auth-server/src
(cw-auth-server) student@cc:~/cw-auth-server/src$ pip install -r requirements.txt
Collecting asgiref==3.2.3 (from -r requirements.txt (line 1))
```

Figure 2: Installing requirements on the production environment

2 Solution Development Approach

2.1 Step 1: Initial design as single integrated application

The initial development approach was to develop a single integrated app i.e. a self-contained application that included both “authorization and authentication” and “Auctioning RESTful APIs”.

Once complete, the apps were split into separate microservices as outlined below.

2.2 Step 2: Split into separate apps for “authorization and authentication” and “Auctioning RESTful APIs”

Once integrated development and testing of both applications were complete, work then proceeded to split them into separate microservices as follows:

- a. “Authorization and Authentication” Server
- b. Online Auctioning App

(a) handles authorization and authentication for the Auction app; (b) hosts the online auctioning app.

The apps are in separate folders below on the supplied VM. (a) is in the “cw-auth-server” folder; (b) in the “cw-res-server” folder. The setup of each app was done in an isolated Python virtual environment.

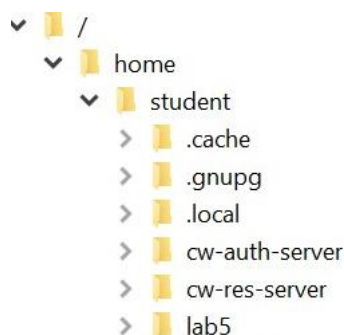
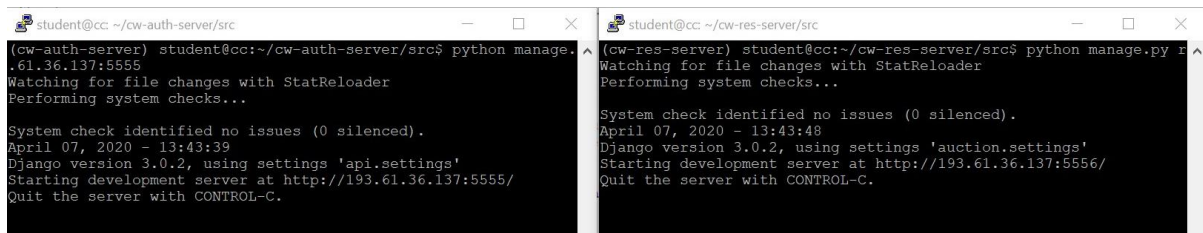


Figure 3: Location of apps on supplied VM

Both apps were hosted on the same VM using different ports:

- a. “Authorization and Authentication” Server: 193.61.36.137:5555
- b. Online Auctioning App: 193.61.36.137:5556

Accessing the auction databased requires user registration and a token from the OAuth service.



The image shows two side-by-side terminal windows. The left window is titled 'student@cc: ~/cw-auth-server/src' and shows the command 'python manage.py runserver' being executed. The output indicates that the system check passed, the Django version is 3.0.2, and the development server is running at http://193.61.36.137:5555/. The right window is titled 'student@cc: ~/cw-res-server/src' and shows the command 'python manage.py runserver'. The output indicates that the system check passed, the Django version is 3.0.2, and the development server is running at http://193.61.36.137:5556/.

```
student@cc: ~/cw-auth-server/src
(cw-auth-server) student@cc:~/cw-auth-server/src$ python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
April 07, 2020 - 13:43:39
Django version 3.0.2, using settings 'api.settings'
Starting development server at http://193.61.36.137:5555/
Quit the server with CONTROL-C.

student@cc: ~/cw-res-server/src
(cw-res-server) student@cc:~/cw-res-server/src$ python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
April 07, 2020 - 13:43:48
Django version 3.0.2, using settings 'auction.settings'
Starting development server at http://193.61.36.137:5556/
Quit the server with CONTROL-C.
```

Figure 4: Screenshot showing developed services running on separate ports

3 Software applications/Technology used: Testing and Development

Initial development and testing were done on a VM installed on Oracle VM Virtual Box in a Windows Environment: The following software/technologies were:

- Python programming language (Python 3) [1]
- Django==3.0.2 [2]
- django-oauth-toolkit==1.3.0 [3]
- django-rest-framework==0.1.0 [4]
- PuTTY: SSH client [5]
- Notepad++Portable: Text Editor [6]
- Oracle VM VirtualBox – Type 2 Hypervisor [7]
- Ubuntu 18.04.3 LTS (Bionic Beaver) Image: Linux based OS [8]

Django==3.0.2 [2], django-oauth-toolkit==1.3.0 [3], django-rest-framework==0.1.0 [4] are packages that together significantly simplify the amount of work involved in building a web service using system oriented architecture that conforms to REST principles and uses OAuth for authorization and authentication. These tools were key to ensuring the delivery of this project within the specified timescale.

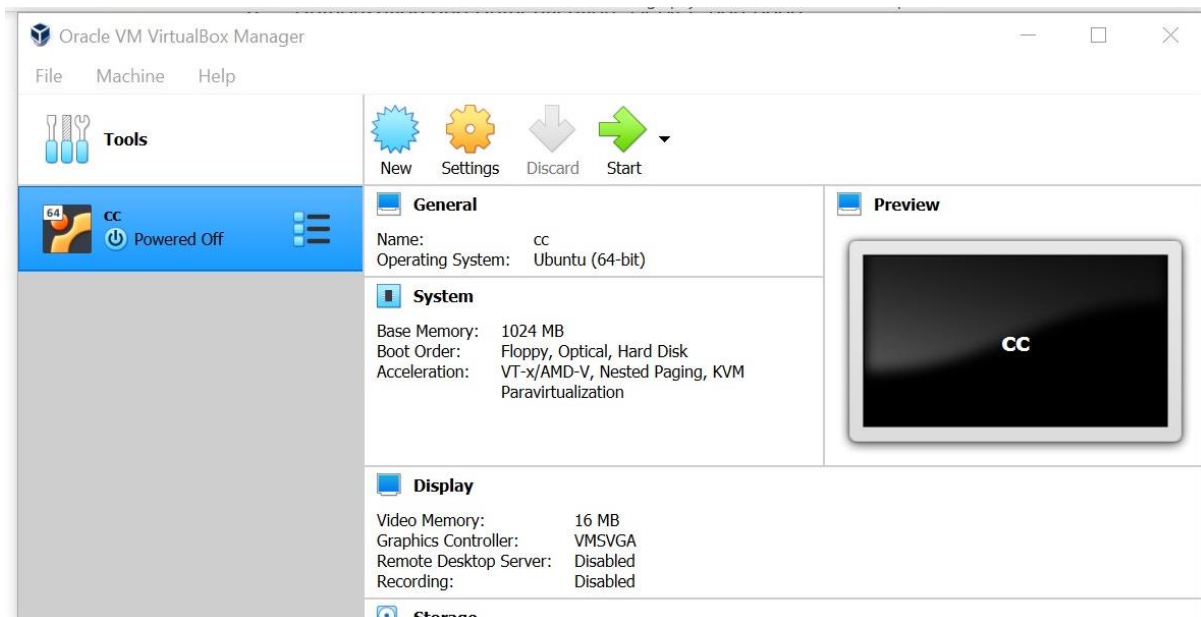


Figure 5: Image of Oracle VM VirtualBox running on local windows machine

Once development and testing were complete, both apps were deployed onto the VM environment provided. “Pip list/pip freeze” were used to export/install packages across environments.

4 How do the two decoupled apps communicate?

The separate decoupled Auth and resource servers introduce the need to maintain a connection between the Auth server and the resource server. Why? The resource server needs some way of verifying that the tokens presented to it are from the Auth server. The verification process is described in section 1.1.

We use the “introspection” protocol to achieve this [9] [10].

- 1) The Resource Server (i.e. “Online Auctioning App”) will try to verify its requests on the “Authorization and Authentication” server.
- 2) The scope of the OAUTH2_PROVIDER on the “Authorization and Authentication” server is modified to allow introspection. Specifically, a new scope attribute (introspection) is added to the settings.py file.

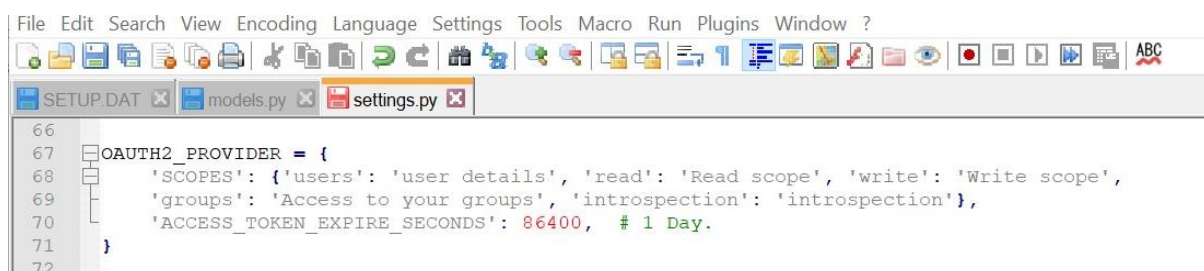
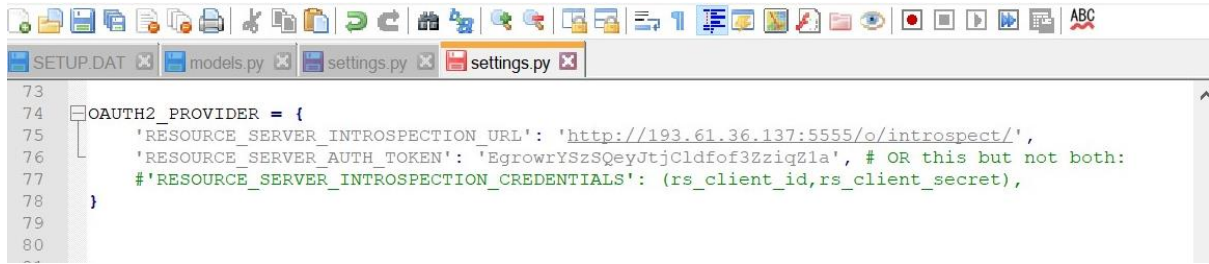


Figure 6: Adding introspection to OAUTH2 provider scopes

- 3) Additional settings are made to the settings.py on the “Online Auctioning App” as follows:
 - a. 'RESOURCE_SERVER_INTROSPECTION_URL': 'http://193.61.36.137:5555/o/introspect/', and
 - b. 'RESOURCE_SERVER_AUTH_TOKEN': 'EgrowrYSzSQeyJtjCldfof3ZziqZ1a', a token generated from this server.



```
73
74 OAUTH2_PROVIDER = {
75     'RESOURCE_SERVER_INTROSPECTION_URL': 'http://193.61.36.137:5555/o/introspect/',
76     'RESOURCE_SERVER_AUTH_TOKEN': 'EgrowrYSzSqeyJtjCldfof3ZziqZ1a', # OR this but not both:
77     #'RESOURCE_SERVER_INTROSPECTION_CREDENTIALS': (rs_client_id,rs_client_secret),
78 }
79
80
```

Figure 7: “Online Auctioning App” updates to support introspection

Other sources used to understand and implement token introspection in this project include: [11] [12] [13] [14] [15].

[15] is particularly for useful for defining more low-level scopes e.g. on a per model basis.

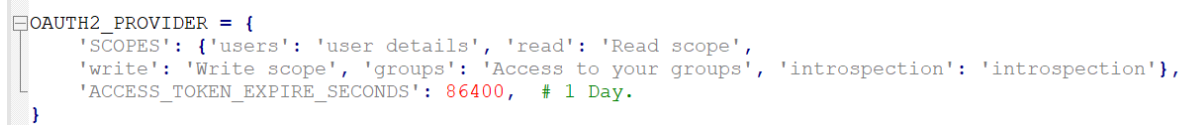
5 Phase B: Development of Authorization and Authentication services

The “Authorization and Authentication” Server offers the following functionality via the endpoints shown:

- Register: <http://193.61.36.137:5555/authentication/token/>
- Request token: <http://193.61.36.137:5555/authentication/token/>
- Refresh a token: <http://193.61.36.137:5555/authentication/token/refresh/>
- Revoke a token: <http://193.61.36.137:5555/authentication/token/revoke/>

Potential users of the “Online Auctioning App” will first need to:

- Register (via the endpoint shown above)
- Once registered, a token to access the “Online Auctioning App” will be issued in the response
 - o Each token lasts for 24hrs.
- Separate methods are provided to request tokens (i.e. not as part of registration process)
- Expired tokens can be refreshed
- Issues tokens can be revoked (e.g. to address misuse)



```
OAUTH2_PROVIDER = {
    'SCOPES': {'users': 'user details', 'read': 'Read scope',
              'write': 'Write scope', 'groups': 'Access to your groups', 'introspection': 'introspection'},
    'ACCESS_TOKEN_EXPIRE_SECONDS': 86400, # 1 Day.
}
```

Figure 8: Code snippet showing specification of token duration

6 Phase C: Development of Auctioning RESTful APIs

6.1 Database Design

The auction app database model consists of the following Entities/Attributes which are translated into tables/fields within Django:

Entities	Attributes	Comments
User	Username	This uses the default (i.e. inbuilt) Django user model
Product	Owner (<i>Foreign key onto the User table</i>) Title Description Quantity Category: (one of 5 categories for ease of reference – Laptop, Console, Gadget, Game or TV) Condition: New, Nearly New or Used Date posted Sold status: Boolean – Yes or No	The FK to the user model ensures the project specification that only registered users can add items to the app is met.
Auction	Product_id (<i>Foreign key onto the Product table</i>) Time_starting Time_ending Active: Boolean – Yes or No	The FK to the product entities allows us to track the owner of the product.
Bid	Auction_id: (<i>Foreign key onto the Auction table</i>) Bid_time: Auto Bid_amount Bidder: (<i>Foreign key onto the User</i>)	<p>The FK onto the Auction model allows us to track the owner of the product.</p> <p>The FK onto the User entity ensures the project specification that only registered users can bid on items is met.</p>

The relationship between entities is shown in Figure 9. This implies the following relationships:

User <-> Product:

- Only users (registered) can add products
- A user can have zero or multiple products
- Each product can only have 1 owner

Product <-> Auction:

- A product can be added to 0,1 or many actions
- An auction can only be a single product

User <-> Bid:

- A user can make 0,1 or many bids
- Each bid is made by a single unique bidder

Bid <-> Auction:

- An auction can have 0,1 or many bids

- 0,1 or many bids can be made on an auction

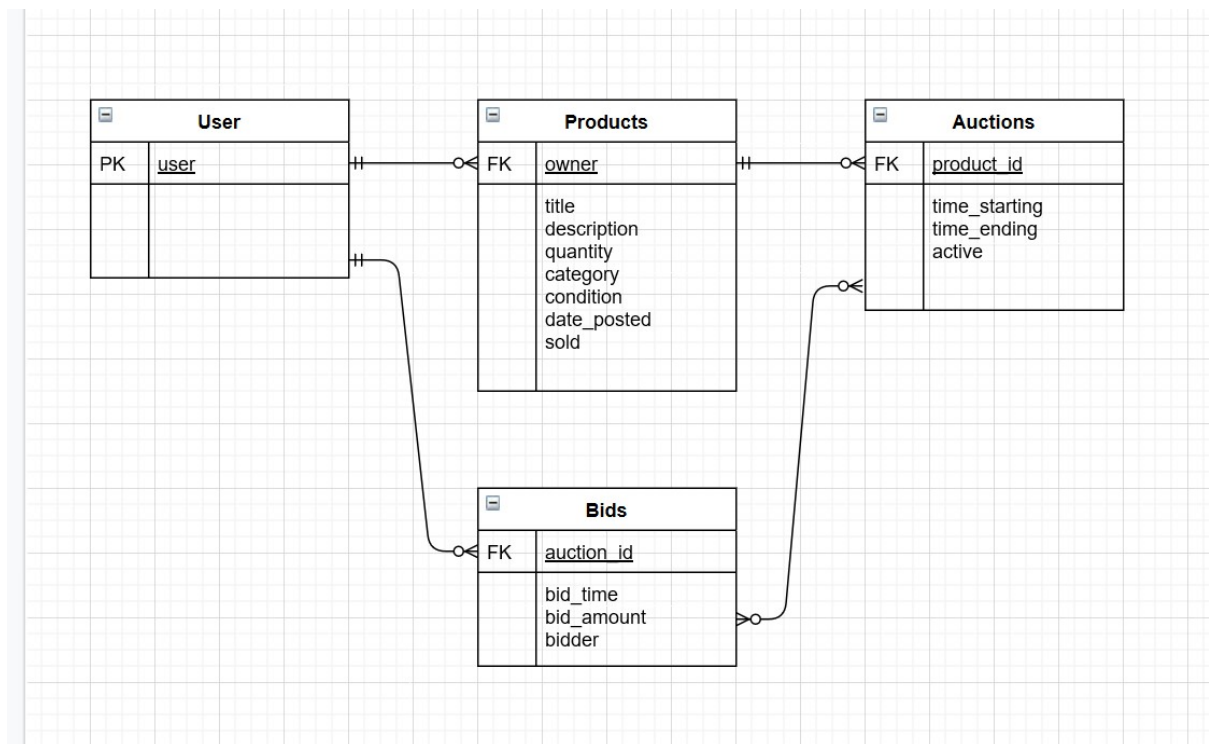


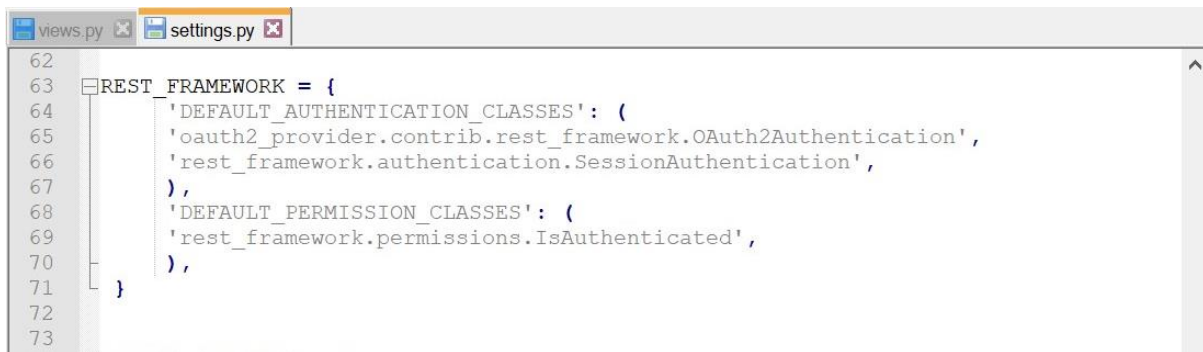
Figure 9: Online Auctioning App ER Diagram

6.2 Constraints Identification and implementation

Several constraints on functionality were included within project specification. Some of these are:

- 1) Only authorised users should access the API
- 2) The API should allow any registered user to post items
- 3) Users that post items cannot bid for their own items.

The first constraint was achieved by adding an "IsAuthenticated" permission class to the API views (implemented in the settings.py file).



```
62
63 REST_FRAMEWORK = {
64     'DEFAULT_AUTHENTICATION_CLASSES': (
65         'oauth2_provider.contrib.rest_framework.OAuth2Authentication',
66         'rest_framework.authentication.SessionAuthentication',
67     ),
68     'DEFAULT_PERMISSION_CLASSES': (
69         'rest_framework.permissions.IsAuthenticated',
70     ),
71 }
72
73
```

Figure 10: Configuration of authenticated user to access API Views

For the second, users are registered in the app once a successful call is made. An API call to post items requires the user as one of the key inputs to the json record. If the user is not registered with the app, the call will fail.

Finally, several approaches were investigated to achieve the “constraint that users that post items cannot bid on their items”. It was decided to use the most low-level approach and prevent such an action being saved in the database. This was achieved by overriding the save method of the bid model and adding a check to see if the bidder is the same as the product owner. See Figure 11.

```
def save(self, *args, **kwargs):
    if self.bidder == self.auction_id.product_id.owner:
        raise ValidationError("Owners cannot bid on their own products")
```

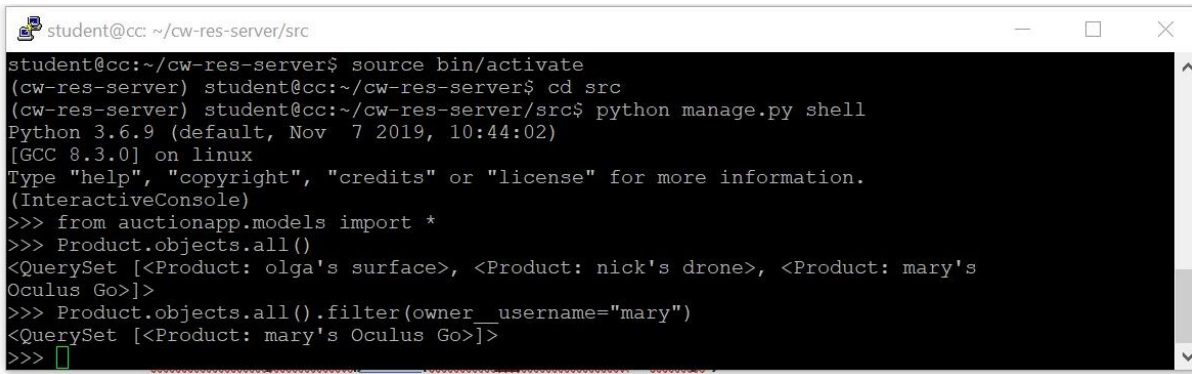
Figure 11: Bidder not Owner Validation Check

This solution achieves the specification but is not user friendly i.e. no useful information is fed back to the user who tries to make such a call as the validation error is raised on the server side and not sent to the user (see for e.g. the results of TC9 in section 7.9. This is an area to improve the user experience.

Other constraints were resolved largely via the use of foreign keys between tables/model entities.

6.3 Query design

Django includes a built-in shell application (as part of the admin application) that allows users to query the database directly [16]. This tool was used to design the queries. Once ready, the queries were then implemented into the API function call. [17] provided useful info in how to modify the API view to accommodate “url” based queries.



```
student@cc: ~/cw-res-server/src
student@cc:~/cw-res-server$ source bin/activate
(cw-res-server) student@cc:~/cw-res-server$ cd src
(cw-res-server) student@cc:~/cw-res-server/src$ python manage.py shell
Python 3.6.9 (default, Nov 7 2019, 10:44:02)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>> from auctionapp.models import *
>>> Product.objects.all()
<QuerySet [<Product: olga's surface>, <Product: nick's drone>, <Product: mary's
Oculus Go>]>
>>> Product.objects.all().filter(owner__username="mary")
<QuerySet [<Product: mary's Oculus Go>]>
>>>
```

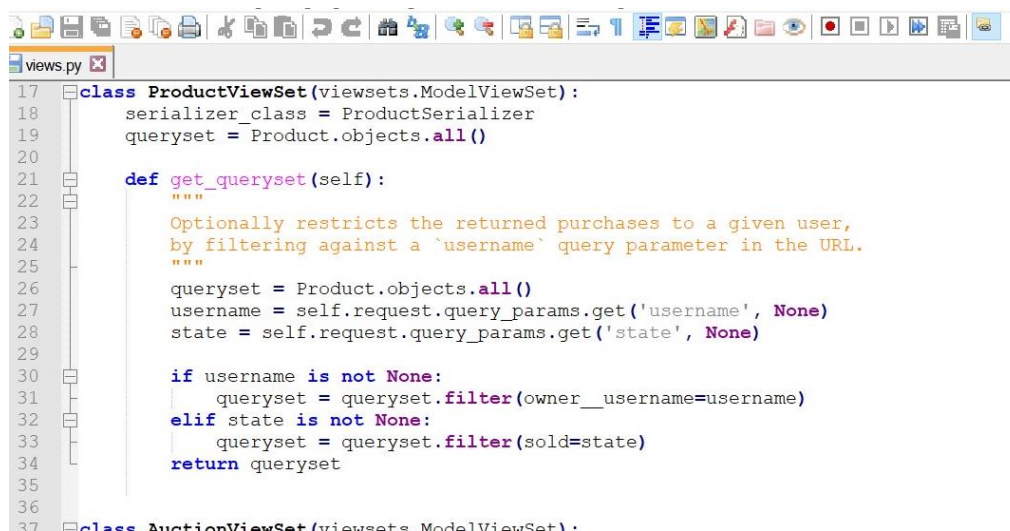
Figure 12: Image of Django Database Shell Application

Example queries include:

1. Query on the products table to find the products owned by a user
Product.objects.all().filter(owner__username="mary")
2. Query to find the person who won the auction. This gets a list of bids for item and sorts by the highest bidder (who wins)

Bid.objects.all().filter(auction_id__product_id__owner__username="mary").order_by('bid_amount')[0]

Key learning – querysets work with fields in the database – such that querying on model attributes is not straightforward – there are way around this, but it was felt as too complex for this project.



```
17 class ProductViewSet(viewsets.ModelViewSet):
18     serializer_class = ProductSerializer
19     queryset = Product.objects.all()
20
21     def get_queryset(self):
22         """
23         Optionally restricts the returned purchases to a given user,
24         by filtering against a `username` query parameter in the URL.
25         """
26         queryset = Product.objects.all()
27         username = self.request.query_params.get('username', None)
28         state = self.request.query_params.get('state', None)
29
30         if username is not None:
31             queryset = queryset.filter(owner__username=username)
32         elif state is not None:
33             queryset = queryset.filter(sold=state)
34         return queryset
35
36
37 class AuctionViewSet(viewsets.ModelViewSet):
```

Figure 13: Modification of API View to accommodate bespoke queries

6.4 API's developed

The online auctioning app includes the following protected API's:

1. Users
 - a. Users can be added directly to the auction app, but this is useless as they cannot be authenticated i.e. no password provided. This means they cannot access the other API's
2. Products

- a. Authorized users can add products
 - b. “Sold” field which is currently manually set. IMPROVE: use a task-based/countdown service to set this automatically based on the outcome of the auction.
3. Auctions
- a. Authorized users can start auctions
 - b. “Active” field which is currently manually set. IMPROVE: used a task-based/countdown service to set this automatically based on the outcome of the auction.
 - i. Several options were explored here – defining attributes via methods on the Auctions class
 - ii. Using the computed property


```

@property
def status(self):
    if self.time_ending > timezone.now():
        return "ACTIVE"

from django.core.exceptions import ValidationError
from django.utils import timezone
#import computed_property
          
```
 - iii. Fundamentally these were still “static” approaches the update wouldn’t be done until the function/method was triggered.
 - iv. As above, a dynamic service using celery [add reference or similar] would be the best option to achieve something dynamic.
 - c. Foreign key to Products

6.5 Exposing the API’s

The following endpoints are available for the Online Auctioning app:

1. "users": "http://193.61.36.137:5556/v1/users/",
2. "products": "http://193.61.36.137:5556/v1/products/",
3. "bids": "http://193.61.36.137:5556/v1/bids/",
4. "auctions": "http://193.61.36.137:5556/v1/auctions/"

In developing these, a range of default Django utilities were leveraged, specifically:

- Inbuilt model classes
- Class based views
- Serializers to handle conversions to/from json
- Querysets to query the database

The standard API methods are available for these end point. What is available will vary depending on the type – “collection” or “instance”

For a collection, GET or POST are allowed.

For an instance, further options are included: GET, PUT, PATCH, DELETE. Basically, the extra options are to allow further manipulation of a specific record e.g. update or delete.

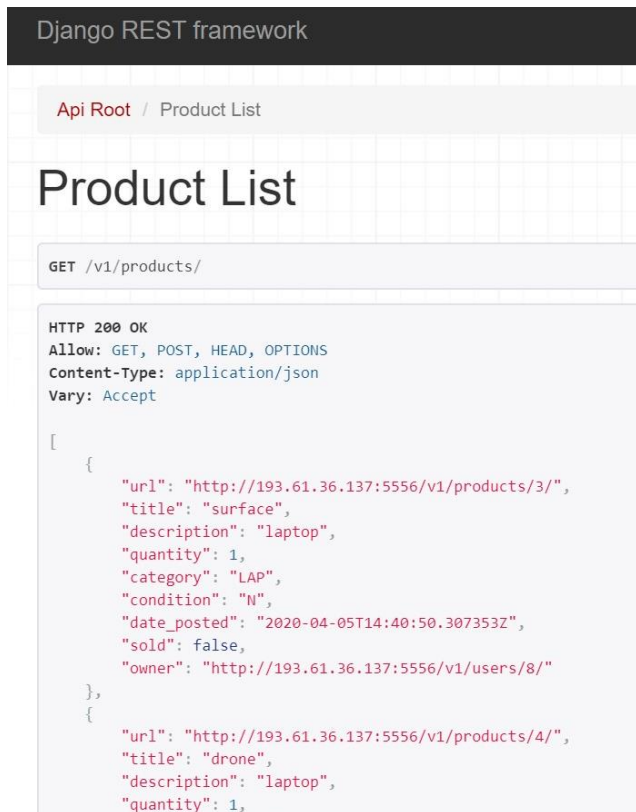


Figure 15: Example of a "collection"

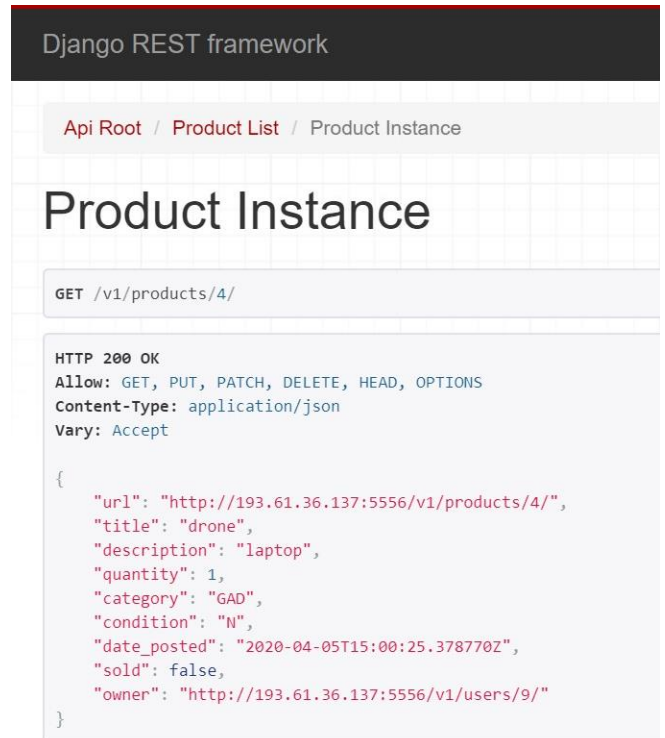


Figure 14: Example of an instance

7 Phase D: Development of a testing application and Test results

A testing application consisting of a python script (deployed via a Jupyter notebook) was developed to test the application against the specification outlined in the project scope.

The specification included 13 tests listed from TC1 to TC13 [add reference]. Appropriate code was developed to ensure the objectives of the test specification were met. The python script includes code snippets to make calls to the various servers (API endpoints), access resources and test functionality.

Table 1 provides a high-level summary of each test in terms of pass/fail.

Table 1: Summary results for test specification

Test ID	Pass/Fail
TC1	Pass
TC2	Pass
TC3	Pass
TC4	Pass
TC5	Pass
TC6	Pass
TC7	Pass
TC8	Pass
TC9	Pass
TC10	Pass
TC11	Pass
TC12	Pass
TC13	Pass

7.1 TC1: Olga, Nick and Mary register in the application and are ready to access the API

TC1: Olga, Nick and Mary register in the application and are ready to access the API.

Comment: Test is successful. Users are registered and tokens are issued.

```
2]: M reg_url = "http://193.61.36.137:5555/authentication/register/"

olga = {"username": "olga", "password": "1234abcd"}
nick = {"username": "nick", "password": "1234abcd"}
mary = {"username": "mary", "password": "1234abcd"}

reg_list = [olga, nick, mary]

user_details = dict()
for _ in reg_list:
    user = _["username"]+"_response"
    user = requests.post(reg_url, data = _)
    user = user.json()
    user_details[_["username"]] = user

print(user_details)

{'olga': {'access_token': 'ARQGqSWSnP9oqxznDjNHeQzImSAe1', 'expires_in': 86400, 'token_type': 'Bearer', 'scope': 'users read write groups introspection', 'refresh_token': 'hDE774bUKnxisxiCkFJlh66OpIegCJy'}, 'nick': {'access_token': 'SX79WUMfi3PXP L98DpHZJEJI00DWY1', 'expires_in': 86400, 'token_type': 'Bearer', 'scope': 'users read write groups introspection', 'refresh_token': 'hn3g2FFumWlbcNw42pNMDK39b0Hm3A'}, 'mary': {'access_token': 'D0wYo9wmitk1F219IvClK70ZyCRoRc', 'expires_in': 86400, 'token_type': 'Bearer', 'scope': 'users read write groups introspection', 'refresh_token': '6vfjh7SedykI9iXcfiRtQ1PU2bQvR Q'}}
```


7.2 TC2: Olga, Nick and Mary will use the oAuth v2 authorisation service to get their tokens

TC2: Olga, Nick and Mary will use the oAuth v2 authorisation service to get their tokens.

NB Olga, Nick and Mary already have tokens as shown above.

7.3 TC3: Olga makes a call to the API (any endpoint) without using a token. This call should be unsuccessful as the user is unauthorised

TC3: Olga makes a call to the API (any endpoint) without using a token. This call should be unsuccessful as the user is unauthorised.

We will use the products endpoint for this test

Comment: Test is successful. User credentials - token etc were not provided as part of the call; hence no data was returned.

```
59]: products_url = "http://193.61.36.137:5556/v1/products/"
      olga_response = requests.get(products_url)
      print(olga_response.json())

{'detail': 'Authentication credentials were not provided.'}
```

7.4 TC4: Olga adds an item for auction with an expiration time using her token.

TC4: Olga adds an item for auction with an expiration time using her token.

Comment:

- 1) Only users can post items and our users are registered on the "Authorization and Authentication" Server.
- 2) To post items, we need to pass in some info about the users as part of the call - in this case the url of the user as seen from the Online Auctioning app
- 3) Hence, before we get our users to post items, we need to get them visible on the "Online Auctioning App". This occurs once each user has made a call (any call) to the server. As we have used each user to call the products API - each user is now set up in the Online auctioning app. The list of users is available at the following endpoint: <http://193.61.36.137:5556/v1/users/>
- 4) Henceforth, any API call that needs user details, we will need to pass in the url for each user as part of the call to the API.

The urls for each user are as follows:

Olga: <http://193.61.36.137:5556/v1/users/8/>

Nick: <http://193.61.36.137:5556/v1/users/9/>

Mary: <http://193.61.36.137:5556/v1/users/10/>

NB - Other users were added and deleted from the system hence the reason the records do not start from 1

- 5) Finally, we have separate entities for Products and Auctions. Adding an item for an auction involves two steps: 1) adding the product 2) Creating an auction

Step 1: Call the products api to add a product

```
7]: M olga_token = user_details['olga']['access_token']

headers = {'Authorization': 'Bearer '+str(olga_token)}

record = {
    "title": "surface",
    "description": "laptop",
    "quantity": 1,
    "category": "LAP",
    "condition": "N",
    "sold": "false",
    "owner": "http://193.61.36.137:5556/v1/users/8/"
}

# Let's post the data and print the response
olga_response = requests.post(products_url, headers=headers, data = record)
print(olga_response.json())

{'url': 'http://193.61.36.137:5556/v1/products/3/', 'title': 'surface', 'description': 'laptop', 'quantity': 1, 'category': 'LAP', 'condition': 'N', 'date_posted': '2020-04-05T14:40:50.307353Z', 'sold': False, 'owner': 'http://193.61.36.137:5556/v1/users/8/'}
```

Step 2: Call the Auction API to add the product to an auction

```
9]: M auction_url = "http://193.61.36.137:5556/v1/auctions/"

olga_auction = {
    "time_starting": "2020-04-05T12:00:00Z",
    "time_ending": "2020-04-30T12:00:00Z",
    "active": "true",
    "product_id": "http://193.61.36.137:5556/v1/products/3/"
}

#NB we know the url for the product from the response we got when we posted it

# Let's post the data and print the response
olga_response_auct = requests.post(auction_url, headers=headers, data = olga_auction)
print(olga_response_auct.json())

{'url': 'http://193.61.36.137:5556/v1/auctions/2/', 'time_starting': '2020-04-05T12:00:00Z', 'time_ending': '2020-04-30T12:00:00Z', 'active': True, 'product_id': 'http://193.61.36.137:5556/v1/products/3/'}
```

7.5 TC5: Nick adds an item for auction with an expiration time using his token.

TC5: Nick adds an item for auction with an expiration time using his token.

We will be adding a drone

Step 1: Call the products api to add a product

```
n [70]: M headers = {'Authorization': 'Bearer '+str(nick_token)}

record = {
    "title": "drone",
    "description": "laptop",
    "quantity": 1,
    "category": "GAD",
    "condition": "N",
    "sold": "false",
    "owner": "http://193.61.36.137:5556/v1/users/9/"
}

# Let's post the data and print the response
nick_response = requests.post(products_url, headers=headers, data = record)
print(nick_response.json())

{'url': 'http://193.61.36.137:5556/v1/products/4/', 'title': 'drone', 'description': 'laptop', 'quantity': 1, 'category': 'GAD', 'condition': 'N', 'date_posted': '2020-04-05T15:00:25.378770Z', 'sold': False, 'owner': 'http://193.61.36.137:5556/v1/users/9/'}
```

Step 2: Call the Auction API to add the product to an auction

```
In [71]: █ nick_auction = {
    "time_starting": "2020-04-05T12:00:00Z",
    "time_ending": "2020-04-30T12:00:00Z",
    "active": "true",
    "product_id": "http://193.61.36.137:5556/v1/products/4/"
}

#NB we know the url for the product from the response we got when we posted it

# Let's post the data and print the response
nick_response_auct = requests.post(auction_url, headers=headers, data = nick_auction)
print(nick_response_auct.json())

{'url': 'http://193.61.36.137:5556/v1/auctions/3/', 'time_starting': '2020-04-05T12:00:00Z', 'time_ending': '2020-04-30T12:00:00Z', 'active': True, 'product_id': 'http://193.61.36.137:5556/v1/products/4/'}
```

7.6 TC6: Mary adds an item for auction with an expiration time using her token.

TC6: Mary adds an item for auction with an expiration time using her token.

We will adding a Oculus VR Headset

Step 1: Call the products api to add a product

```
In [72]: █ headers = {'Authorization': 'Bearer '+str(mary_token)}

record = {
    "title": "Oculus Go",
    "description": "VR Headset",
    "quantity": 1,
    "category": "GAD",
    "condition": "N",
    "sold": "false",
    "owner": "http://193.61.36.137:5556/v1/users/10/"
}

# Let's post the data and print the response
mary_response = requests.post(products_url, headers=headers, data = record)
print(mary_response.json())

{'url': 'http://193.61.36.137:5556/v1/products/5/', 'title': 'Oculus Go', 'description': 'VR Headset', 'quantity': 1, 'category': 'GAD', 'condition': 'N', 'date_posted': '2020-04-05T15:11:16.310969Z', 'sold': False, 'owner': 'http://193.61.36.137:5556/v1/users/10/'}
```

Step 2: Call the Auction API to add the product to an auction

```
In [73]: █ mary_auction = {
    "time_starting": "2020-04-05T12:00:00Z",
    "time_ending": "2020-04-30T12:00:00Z",
    "active": "true",
    "product_id": "http://193.61.36.137:5556/v1/products/5/"
}

#NB we know the url for the product from the response we got when we posted it

# Let's post the data and print the response
mary_response_auct = requests.post(auction_url, headers=headers, data = mary_auction)
print(mary_response_auct.json())

{'url': 'http://193.61.36.137:5556/v1/auctions/4/', 'time_starting': '2020-04-05T12:00:00Z', 'time_ending': '2020-04-30T12:00:00Z', 'active': True, 'product_id': 'http://193.61.36.137:5556/v1/products/5/'}
```

7.7 TC7: Nick and Olga browse all the available items, there should be three items available.

TC7: Nick and Olga browse all the available items, there should be three items available.

Nick browsing the product api...

```
In [78]: M headers = {'Authorization': 'Bearer '+str(nick_token)}
        nick_response = requests.get(products_url, headers=headers)
        print(nick_response.json())
        print(len(nick_response.json()))
3
```

There are three products as expected...

Olga browsing the product api...

```
In [79]: M headers = {'Authorization': 'Bearer '+str(olga_token)}
        olga_response = requests.get(products_url, headers=headers)
        print(olga_response.json())
        print(len(olga_response.json()))
3
```

There are three products as expected...

7.8 TC8: Nick and Olga get the details of Mary's item.

TC8: Nick and Olga get the details of Mary's item.

Comment: This call will use the product endpoint which has been modified to take query parameters in the url - in this case the username

Nic's call...

```
In [90]: M headers = {'Authorization': 'Bearer '+str(nick_token)}
        query_products_url = "http://193.61.36.137:5556/v1/products/?username="
        user = "mary"
        nick_response = requests.get(query_products_url+user, headers=headers)
        print(nick_response.json())

[{'url': 'http://193.61.36.137:5556/v1/products/5/', 'title': 'Oculus Go', 'description': 'VR Headset', 'quantity': 1, 'category': 'GAD', 'condition': 'N', 'date_posted': '2020-04-05T15:11:16.310969Z', 'sold': False, 'owner': 'http://193.61.36.137:5556/v1/users/10/'}]
```

Olga's call...

```
In [91]: M headers = {'Authorization': 'Bearer '+str(olga_token)}
        query_products_url = "http://193.61.36.137:5556/v1/products/?username="
        user = "mary"
        olga_response = requests.get(query_products_url+user, headers=headers)
        print(olga_response.json())

[{'url': 'http://193.61.36.137:5556/v1/products/5/', 'title': 'Oculus Go', 'description': 'VR Headset', 'quantity': 1, 'category': 'GAD', 'condition': 'N', 'date_posted': '2020-04-05T15:11:16.310969Z', 'sold': False, 'owner': 'http://193.61.36.137:5556/v1/users/10/'}]
```

7.9 TC9: Mary bids for her own item. This call should be unsuccessful, an owner cannot bid for own items.

TC9: Mary bids for her own item. This call should be unsuccessful, an owner cannot bid for own items

```
In [94]: > bid_url = "http://193.61.36.137:5556/v1/bids/"

headers = {'Authorization': 'Bearer '+str(mary_token)}

mary_own_bid = {
    "bid_amount": 5,
    "auction_id": "http://193.61.36.137:5556/v1/auctions/4/",
    "bidder": "http://193.61.36.137:5556/v1/users/10/"
}

#NB we know the url for the product from the response we got when we posted it

# Let's post the data and print the response
mary_bid = requests.post(bid_url, headers=headers, data = mary_own_bid)
print(mary_bid.json())

-----
JSONDecodeError                                Traceback (most recent call last)
<ipython-input-94-3eb3ab42717c> in <module>
    13 # Let's post the data and print the response
    14 mary_bid = requests.post(bid_url, headers=headers, data = mary_own_bid)
--> 15 print(mary_bid.json())
    16
    17

C:\ProgramData\Anaconda3\lib\site-packages\requests\models.py in json(self, **kwargs)
    895         # used.
    896         pass
    ---
    897         raise ValueError('No JSON object could be decoded')
```

JSONDecodeError: Expecting value: line 1 column 1 (char 0)

Call is unsuccessful. Validation error raised in the background

```
obj.save(force_insert=True, using=self.db)
File "/home/student/cw-res-server/src/auctionapp/models.py", line 116, in save
    raise ValidationError("Owners cannot bid on their own products")
django.core.exceptions.ValidationError: ['Owners cannot bid on their own products']
[05/Apr/2020 18:01:49] "POST /v1/bids/ HTTP/1.1" 500 16615
```

7.10 TC10: Nick and Olga bid for Mary's item in a round robin fashion (one after the other).

TC10: Nick and Olga bid for Mary's item in a round robin fashion (one after the other).

Nicks bid

```
In [95]: bid_url = "http://193.61.36.137:5556/v1/bids/"

headers = {'Authorization': 'Bearer '+str(nick_token)}

nick_own_bid = {
    "bid_amount": 5,
    "auction_id": "http://193.61.36.137:5556/v1/auctions/4/",
    "bidder": "http://193.61.36.137:5556/v1/users/9/"
}

nick_bid = requests.post(bid_url, headers=headers, data = nick_own_bid)
print(nick_bid.json())

{'url': 'http://193.61.36.137:5556/v1/bids/1/', 'bid_time': '2020-04-05T18:04:27.870220Z', 'bid_amount': 5, 'auction_id': 'http://193.61.36.137:5556/v1/auctions/4/', 'bidder': 'http://193.61.36.137:5556/v1/users/9/'}
```

Nick's Bid is successful...

Olga's bid

```
In [96]: bid_url = "http://193.61.36.137:5556/v1/bids/"

headers = {'Authorization': 'Bearer '+str(olga_token)}

olga_own_bid = {
    "bid_amount": 10,
    "auction_id": "http://193.61.36.137:5556/v1/auctions/4/",
    "bidder": "http://193.61.36.137:5556/v1/users/8/"
}

olga_bid = requests.post(bid_url, headers=headers, data = olga_own_bid)
print(olga_bid.json())

{'url': 'http://193.61.36.137:5556/v1/bids/2/', 'bid_time': '2020-04-05T18:07:00.710579Z', 'bid_amount': 10, 'auction_id': 'http://193.61.36.137:5556/v1/auctions/4/', 'bidder': 'http://193.61.36.137:5556/v1/users/8/'}
```

Olga's Bid is successful...

7.11 TC11: Nick or Olga wins the item after the end of the auction.

Olga's Bid is successful...

TC11: Nick or Olga wins the item after the end of the auction.

The person with the highest bid wins the auction. In this case - that will be Olga as she put in a higher bid (10 vs 5).

Step 1: End Auction Ideally, the auction will end automatically once the expiry date is reached and the status will be set to sold. In this case we will manually end the auction by setting the "active" field to false. We will use a HTTP PATCH request for this.

mary's auction currently holds the following info:

```
{ "url": "http://193.61.36.137:5556/v1/auctions/4/", "time_starting": "2020-04-05T12:00:00Z", "time_ending": "2020-04-30T12:00:00Z", "active": true,
  "product_id": "http://193.61.36.137:5556/v1/products/5/" }
```

Step 2: Update status of mary's item to "sold" i.e. "true" Once mary's auction has ended, we will assume that her item has been sold - hence we will need to change the sold attribute to "true".

Step 1: End Mary's Auction

```
In [98]: M mary_auction_url = "http://193.61.36.137:5556/v1/auctions/4/"

headers = {'Authorization': 'Bearer '+str(mary_token)}

mary_end_auction = {
    "active": "false"
}

mary_update = requests.patch(mary_auction_url, headers=headers, data = mary_end_auction)
print(mary_update.json())

{'url': 'http://193.61.36.137:5556/v1/auctions/4/', 'time_starting': '2020-04-05T12:00:00Z', 'time_ending': '2020-04-30T12:00:00Z', 'active': False, 'product_id': 'http://193.61.36.137:5556/v1/products/5/'}
```

Step 2: Update status of mary's item to "sold" i.e. "true"

Details of mary's product are as follows:

```
{ "url": "http://193.61.36.137:5556/v1/products/5/", "title": "Oculus Go", "description": "VR Headset", "quantity": 1, "category": "GAD", "condition": "N",
"date_posted": "2020-04-05T15:11:16.310969Z", "sold": false, "owner": "http://193.61.36.137:5556/v1/users/10/" }
```

The current status is not sold i.e. sold = false.

```
In [99]: M mary_product_url = "http://193.61.36.137:5556/v1/products/5/"

headers = {'Authorization': 'Bearer '+str(mary_token)}

mary_product_status = {
    "sold": "true"
}

mary_product_update = requests.patch(mary_product_url, headers=headers, data = mary_product_status)
print(mary_product_update.json())

{'url': 'http://193.61.36.137:5556/v1/products/5/', 'title': 'Oculus Go', 'description': 'VR Headset', 'quantity': 1, 'category': 'GAD', 'condition': 'N', 'date_posted': '2020-04-05T15:11:16.310969Z', 'sold': True, 'owner': 'http://193.61.36.137:5556/v1/users/10/'}
```

Update successful - sold status is "True".

7.12 TC12: Olga browses all the items sold.

TC12: Olga browses all the items sold.

Comment: This call is essentially a filter on all products with the sold value == "True".

This call will use the product endpoint which has been modified to take query parameters in the url - in this case the username

```
In [103]: M headers = {'Authorization': 'Bearer '+str(olga_token)}

sold_products_url = "http://193.61.36.137:5556/v1/products/?state="

state = "1"

olga_sold_response = requests.get(sold_products_url+state, headers=headers)
print(olga_sold_response.json())

[{'url': 'http://193.61.36.137:5556/v1/products/5/', 'title': 'Oculus Go', 'description': 'VR Headset', 'quantity': 1, 'category': 'GAD', 'condition': 'N', 'date_posted': '2020-04-05T15:11:16.310969Z', 'sold': True, 'owner': 'http://193.61.36.137:5556/v1/users/10/'}]
```

call is successful. Only 1 product is returned. Mary's product that has sold.

7.13 TC13: Mary queries for a list of bids as historical records of bidding actions of her sold item

TC13: Mary queries for a list of bids as historical records of bidding actions of her sold item

Comment:

Comment: This call is essentially a filter on all bids for marys products

This call will use the bid endpoint which has been modified to take query parameters in the url - in this case the username

```
In [105]: > headers = {'Authorization': 'Bearer '+str(mary_token)}

bid_byuser_url = "http://193.61.36.137:5556/v1/bids/?username="

user = "mary"

mary_bid_list = requests.get(bid_byuser_url+user, headers=headers)

print(mary_bid_list.json())
print(len(mary_bid_list.json()))

[{'url': 'http://193.61.36.137:5556/v1/bids/1/', 'bid_time': '2020-04-05T18:04:27.870220Z', 'bid_amount': 5, 'auction_id':
'http://193.61.36.137:5556/v1/auctions/4/', 'bidder': 'http://193.61.36.137:5556/v1/users/9/'}, {'url': 'http://193.61.36.1
37:5556/v1/bids/2/', 'bid_time': '2020-04-05T18:07:00.710579Z', 'bid_amount': 10, 'auction_id': 'http://193.61.36.137:5556/
v1/auctions/4/', 'bidder': 'http://193.61.36.137:5556/v1/users/8/'}]
2
```

Call is successful. Mary has had two bids on her item. One from Nick for 5 and the other from Olga for 10.

This particular call is not generic enough and should be improved to take both the username and the sold status of the product.

8 Acknowledgments and Future Work

This project is the culmination of several man hours work. The work could not have been achieved without the resources made available in the cloud computing class administered by Stelios Sotiriadis. Particular mention is made of the high-quality class presentations and very detailed lab manuals. These were key resources utilized in delivering this project. The teaching assistants were also very helpful. My thanks to Stelios and the TA's. The course and this course work project have provided an in-depth knowledge of key aspects of distributed computing systems of which services development is a key element.

Due to time-constraints, not all desired functionality/potential approaches could be explored or implemented. The list below identifies areas the project could be improved:

1. Consider containers as a deployment vehicle
2. Implement scheduler to handle time-based activities e.g. end of auction, change of auction status etc.
3. Implement validation in the API view to provide feedback to the calling client. This will allow for meaningful response to be given to the caller. More generally, improve the way constraints are defined and handled.
4. Improve database queries and API views e.g. to readily identify winners of products of ended auctions

9 References

- [1] "Welcome to Python.org," [Online]. Available: <https://www.python.org/>.
- [2] "The Web framework for perfectionists with deadlines | Django," [Online]. Available: <https://www.djangoproject.com/>.
- [3] "Welcome to Django OAuth Toolkit Documentation — Django OAuth Toolkit 1.3.2 documentation," [Online]. Available: <https://django-oauth-toolkit.readthedocs.io/en/latest/>.
- [4] "Home - Django REST framework," [Online]. Available: <https://www.django-rest-framework.org/>.
- [5] "Download PuTTY - a free SSH and telnet client for Windows," [Online]. Available: <https://www.putty.org/>.
- [6] "Notepad++," [Online]. Available: <https://notepad-plus-plus.org/>.
- [7] "Oracle VM VirtualBox," [Online]. Available: <https://www.virtualbox.org/>.
- [8] "Ubuntu 18.04.4 LTS (Bionic Beaver)," [Online]. Available: <http://releases.ubuntu.com/releases/18.04/>.
- [9] "OAuth 2.0 Token Introspection," [Online]. Available: <https://oauth.net/2/token-introspection/>.
- [10] "RFC 7662 - OAuth 2.0 Token Introspection," [Online]. Available: <https://tools.ietf.org/html/rfc7662>.
- [11] "Separate Resource Server — Django OAuth Toolkit 1.3.2 documentation," [Online]. Available: https://django-oauth-toolkit.readthedocs.io/en/latest/resource_server.html.
- [12] "security - Django-OAuth-ToolKit : Generating access token's for multiple resources/services using client credentials grant type of OAuth2.0 - Stack Overflow," [Online]. Available: <https://stackoverflow.com/questions/55351275/django-oauth-toolkit-generating-access-tokens-for-multiple-resources-services>.
- [13] "python - Django OAuth- Separate Resource and Authorization Server - Stack Overflow," [Online]. Available: <https://stackoverflow.com/questions/47587486/django-oauth-separate-resource-and-authorization-server>.
- [14] "c# - How Resource Server can identify user from token? - Stack Overflow," [Online]. Available: <https://stackoverflow.com/questions/48770574/how-resource-server-can-identify-user-from-token?answertab=oldest#tab-top>.
- [15] "django scopes: Online Courses, Training and Tutorials on LinkedIn Learning," [Online]. Available: <https://www.linkedin.com/learning/search?keywords=django%20scopes>.

- [16] “django-admin and manage.py | Django documentation | Django,” [Online]. Available: <https://docs.djangoproject.com/en/3.0/ref/django-admin/>.
- [17] “Filtering against urls,” [Online]. Available: <https://www.django-rest-framework.org/api-guide/filtering/#filtering-against-the-url>.

