**Development of Web Application and Web Services 2012/2013**

**Olabode Obe (35650)**

1. Implemented requirements

    - UC1 Create a user account
    - Edit user account information
    - Create a new auction and send mail that a new auction has been created
    - Edit and existing auction
    - Browse and search auction
    - Bid
    - Ban an auction – only the administrator of the site can ban an auction
    - Resolve an auction excluding sending of mails if auction has ended
    - Support for multiple languages –
    - Support for the multiple concurrent sessions – I only implemented the multiple concurrent requirements with edit auction.

2. The YAAS Web Service
    - UC5 – browsing and searching WS1
    - (r'^api/v1/search/(\d{1,3})$', apisearch),
    - (r'^auctions/$', 'YaaasApp.auction.views.auction_list')

3. Functional Testing

    - Database fixture (TR.1.1) and test data generation program (TR1.2) with django-autofixture
    - Automated functional tests created for UC3 (TR 2.1), UC6 (TR 2.2) and UC10 (TR 2.3)

4. Python & Django version

    - Python 2.7.4 and django 2.7.2

5. SQL Schema

```
CREATE TABLE "auction_seller" (
    "id" integer NOT NULL PRIMARY KEY,
    "user_id" integer NOT NULL UNIQUE REFERENCES "auth_user" ("id")
)
;
CREATE TABLE "auction_auctionevent" (
    "id" integer NOT NULL PRIMARY KEY,
    "Title" varchar(200) NOT NULL,
    "description" text NOT NULL,
    "category" varchar(200) NOT NULL,
    "MinimumPrice" decimal NOT NULL,
    "StartDate" datetime NOT NULL,
    "EndDate" datetime NOT NULL,
    "Seller_id" integer NOT NULL REFERENCES "auth_user" ("id"),
    "winning_bidder_id" integer REFERENCES "auth_user" ("id"),
    "is_active" bool NOT NULL,
    "status" integer NOT NULL,
    "is_banned" bool NOT NULL,
    "lockedby" text NOT NULL
)
;
CREATE TABLE "auction_bid" (
    "id" integer NOT NULL PRIMARY KEY,
    "auction_event_id" integer NOT NULL REFERENCES "auction_auctionevent" ("id"),
    "bidder_id" integer NOT NULL REFERENCES "auth_user" ("id"),
    "amount" decimal NOT NULL
)
;
```

6. Session Management

I implemented session management in the index view of the application to keep track of the status of the user. That is, an anonymous user can only see the list of list auctions while a registered and logged in user will be redirected to the user home where they view the latest auction, place, view history as well see the winners of the auction. Session management was also implemented in the set_lang view for the purpose of switching between different languages as well as remembering the favorite language of the user.

NB. I did not manage to complete this requirement but i created a def set_lang view method for switching between languages. Django also handles session management.

7. Confirmation form in UC3

   I implemented this feature by defining two methods for the view which are the createauctionConf method and the saveauctionConf. There is a createauction html form that prompts the user to enter details of the auction. The auction information are saved in the hidden of an html file while the user is prompted for YES/NO confirmation. If the user click YES, the auction information will be sent to the database but if the user clicks NO, the transaction will be cancelled.

8. Concurrency in UC6

   I ran into problems trying to use “select_for_update” to lock down the bid when one user is bidding because it is not supported by sqlite3. I eventually made use of django aggregation. This bid is incremented by 0.01and counted after one user bid and so another user cannot bid less than the previous bid.

   https://docs.djangoproject.com/en/dev/topics/db/aggregation/

9. RESTFUL API

   The search auctions and list all auctions webservice were implemented and they are both restful. Below is the URI used to access these features;

   - (r'^api/v1/search/(\d{1,3})$', apisearch),
   - (r'^auctions/$', 'YaaasApp.auction.views.auction_list')

10. Functional tests

I created database fixtures for the requirements (TR1.1) and test data generation program (TR2.1). The database fixtures was created the normal way as was taught in the class while I generated the test data of the 50 users, 50 auctions and some bids using the django-autofixture. Screen shots of the generated data are attached to this report.

Automated tests

I created functional tests for UC3, UC6 and UC10. In UC3, the following tests were conducted;

- I tested the create auction Conf view to check that it is working
- Anonymous users cannot create auction  and they redirected to the login prompt where they can either login or register
- Registered user was tested to see if the user can login to create auction and finally
- Created the auction

UC6

- I tested the bid view which is the view auction event
- Anonymous user cannot bid on auction event but are redirected to login
- Tested with a real user to see if the user can login to bid
- Tested the bid form

UC10

- I only tested the view the confirm that it is working as expected