Yet another Auction Site

Emil Aura

33417

Project in Development of Web Applications and Web Services 2012-2013

Examiner: Dragos Truscan

Software Engineering

Department of Information Technologies

Åbo Akademi University

March 30, 2013

Contents

1 Intoduction

The assignment was to implement a simple auction site using the Django framework. The following report will give a concise description of the functionalities of the system. The django version used is 1.6.0-alpha and the python version is 2.7.3.

2 Implemented

1. UC1 create user account
2. UC3 create new auction
3. TR2.1 automated test for UC3
4. TR1 DB fixture and data generation program
5. UC5 browse and search
6. WS1 browse and search API for web service
7. UC7 ban auction
8. UC4 edit auction
9. UC2 edit user account info
10. UC8 resolve auction
11. UC9 support for multiple languages
12. UC6 Bid
13. WS2 Bidding API for web service
14. TR2.2 Automated test for UC6 Bid
15. UC10 Support for multiple concurrent sessions
16. TR2.3 Automated test for testing concurrency when bidding

3 Session management

The application uses Django's own session management functions. These are use in the creation of a new auction. When a user tries to create an auction, and the auction passes the validation, he is redirected to a confirmation page. The auction to be created is passed from the "Create an auction page" to the confirmation page using Django's sessions.

```
1 #Saves the auction in sessions
2 request.session['auction'] = auction
3
4 #Fetches the auction from sesions and removes the auction from sessions
5 auction = request.session.get('auction')
6 del request.session['auction']
```

If the user chooses yes, on the confirmation page, the auction is saved to the database and he is redirected to the new auction. Otherwise he is redirected to the index page.

The user authentication is, moreover, implemented using djangos built-in authentication module.

4 Resolving bids using cron jobs (django-celery)

The resolving of auctions is accomplished using cron-jobs that are run every minute. The django-celery extension handles the execution of the cron-jobs at specified time intervals. During development celery has been run as a worker using the *python manage.py celery worker* command, however, it would probably be run as a daemon in a production environment.

The task executed by celery is the following

```
1 @periodic_task(run_every=crontab(hour="*", minute="*", day_of_week="*"))
2 def updateAuctions():
3     auctions = Auction.objects.filter(Q(status=AuctionStatus.active))
4     i=0
5     for auction in auctions:
6         if auction.deadline<timezone.now():
7             auction.status=AuctionStatus.adjudicated
```

```
8                 i += 1
9                 winner = auction.getLatestBid()
10                if winner is not None:
11                    sendMails(auction)
12                auction.save()
13        return i
```

The task fetches all active auctions from the database and checks that their deadline has passed, sets their status to adjudicated, sets the winner to the latest bid and sends mails.

5 Concurrency

Concurrency is handled with versions of an auction and optimistic concurrency control. The auctions version number is stored in a hidden field, when a user views an auction, and is then compared to the version number in the database when the user tries to place a bid. This prevents users from placing bids on auctions whose description has changed since the user read it.

6 REST API

The application has a rest API accessed through **/API/**. The API provides search functionality for non-authenticated users and bid functionality for authenticated users.

6.1 Search

A search is initialized by sending a **GET** request to **/API/search/**search string**. The system will then return all auctions, whose title matches the search string, in JSON.

6.2 Bid

Authenticated users can place bids on auctions by sending a **POST** request to **/API/*username*/auction/*Auction id*/bid/*bid*** with their login credenials in the authentication header in base64 format (*username:password*). The system will then check that the bid validates and return status code 200 if so, else 409.

7 Testing

The system has tests for Auction creation and bid placement. They are divided into two s*CreateAuctionTest* and *PlaceBidTest*. Both uses a fixture with generated data, generated through a view method not available in production (outcommented).

7.1 CreateAuctionTest

CreateAuctionTest tests auction creation with and without authentication and checks if the user is redirected to the login page. Furthermore, it tests the confirmation logic by going through all alternatives (Yes/No/Blank) and checking the amount of auctions before and after the request, to ensure that an auction is created only if the user chooses Yes.

Finally it tests that all the correct information is stored in the correct fields in the DB by fetching the latest added auction from the DB and comparing the values with the on provided in the save POST request.

7.2 PlaceBidTest

The PlaceBidTest test mostly the same things as the previous test, authentication, information correctness. Furthermore it tests the validation logic by passing the same bid twice, passing a bid lower than the minimum price of the auction and finally placing a bid on a auction that's due.

Finally it test the concurrency logic by creating two different clients with different users logged in on the different clients. Client 1 is the seller, that creates a new auction, which Client 2 opens in his browser. Client 1 will then update the auction's description, after which client 2 tries to place a bid on the old version of the auction. The system should then redirect client 2 to he auctions page and display an error message without adding the bid to the DB. The sequence can be seen in figure 1

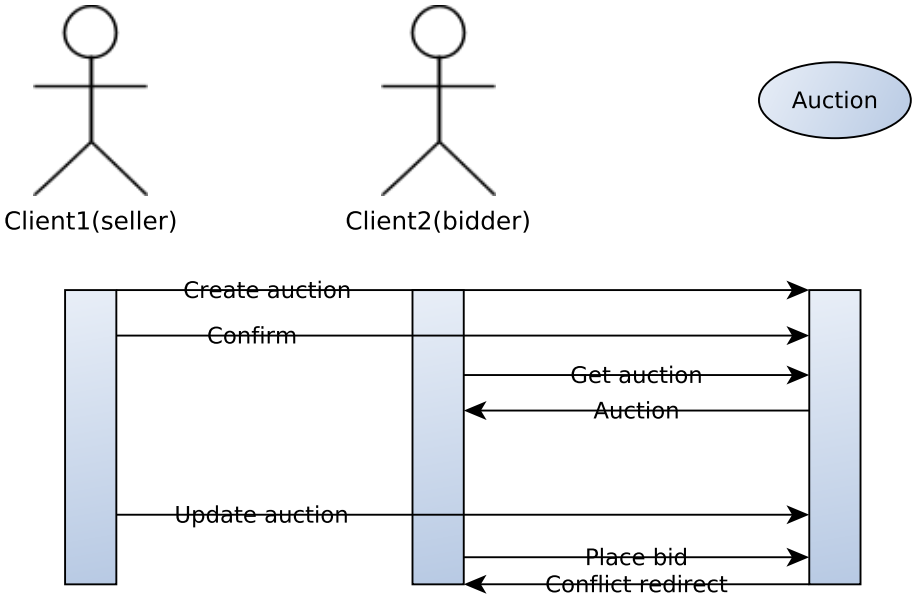

Figure 1: Test sequence

8 UML-chart

The domain model for the project can be seen in figure 2.

9 Language selection

Language selection is implemented with Django's built-in support. It currently handles Swedish, Finnish and English with Swedish being the

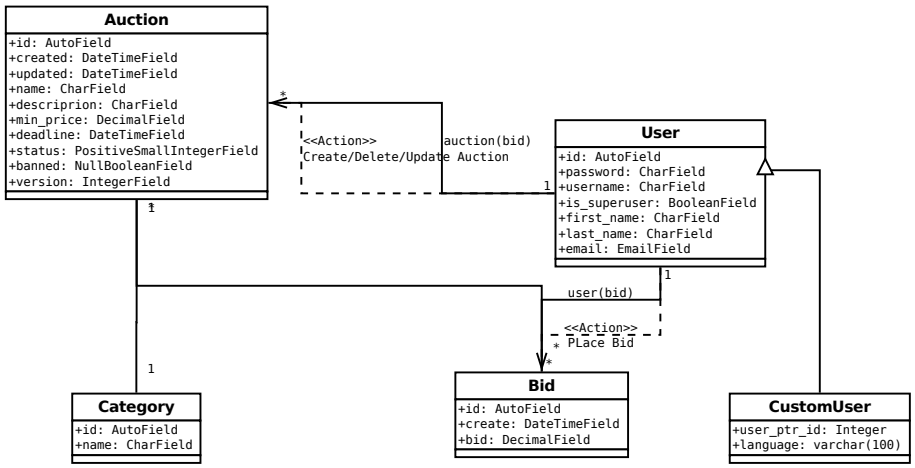

Figure 2: UML-chart

10 Other features

10.1 Optional

default language.

10.1.1 Soft deadlines

If a user bids at an auction within five minutes of the auction deadline, the auction deadline is extended au- tomatically for an additional five minutes. This allows other users to place new bids.

10.1.2 Users preferred language

The user can choose his preferred language upon registration. The page is then automatically translated into this language when the user logs on. Implemented by making a subclass module of User with a varchar field language.

10.2 Additional

10.2.1 Categories

All auctions belong to a category and the user can browse these categories through the navigation panel.

10.2.2 Bootstrap layout

The layout is built with the Bootstrap (`http://twitter.github.com/bootstrap/index.html`) front-end framework.

10.2.3 Sortable table & DatePicker

The auction table is made sortable by a jQuery plugin called tablesorter (`http://tablesorter.com/`) and the DateFields are decorated with the jQuery Datepicker plugin `http://jqueryui.com/datepicker/`.