YAAS Report

In this document I present my implementation of the DWAS project: YAAS web application and web service.

---

1. List of implemented requirements:

- UC1 create user account
- UC3 create new auction
- TR2.1 automated test for UC3
- TR1 DB fixture and data generation program
- UC5 browse and search
- WS1 browse and search API for web service
- UC7 ban auction
- UC4 edit auction
- UC2 edit user account info
- UC8 resolve auction
- UC9 support for multiple languages
- UC6 bid + optional feature: Soft deadlines
- WS2 bidding API for web service + optional feature: Soft deadlines
- ~~TR2.2 automated test for UC6 bid~~
- UC10 support for multiple concurrent sessions
- ~~TR2.3 automated test for testing concurrency when bidding~~

---

2. Python and Django version used:

- Python version 2.7.5
- Django version 1.5.5

---

3. Admin username and password:

- Username: alberto
- Password: alberto

4. How does your application implement session management?

With the Django Session API and hidden form fields.

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

When a authenticate user wants to create a new auction, the web application asks for the *createAuction* form. If the user insert all the fields correctly, the system creates a new form of the type *confAuction*, which only have one option field (yes/no).

If the user submits his option, the system call to the *save_auction* function which is responsible of manage it. If the submission is "yes", then the system retrieves the input data and saves the auction; if the submission is "no", the system don't saves the auction and redirects the user to the add_auction page again. If the user don't submits an option the auction won't be saved in the system.

6. UC8 resolving bids must be initiated automatically (and not by the user). How do you implement this?

I wrote a custom *django-admin* command that manages this situation. The function is implemented in the file */YAAS/YAASApp/management/commands/resolve_acution.py*.

The function checks all the auctions, if finds someone which his status is active and his due date has ended the system will change the status to due. Now the system looks for all the auctions with a status = due, changes the status to finished and sends emails to all the participants.

For use this file automatically, we have to create a cron job in our system adding the following command:

python manage.py resolve_auction

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

I have used pessimistic logic (*locks*). When a seller starts the edition of an auction, the lock variable of the auction changes to *True*. At this point if somebody wants to see the auction or bid on it, he will obtain a error page explaining the situation. When the seller has finish with the edition, the lock changes to *False* again.

If a seller start an edition but never press buttons "*Submit*" or "*Cancel*" (POST), the lock won't be changed to *False* and the only person allowed to change the lock again will be the admin of the page.

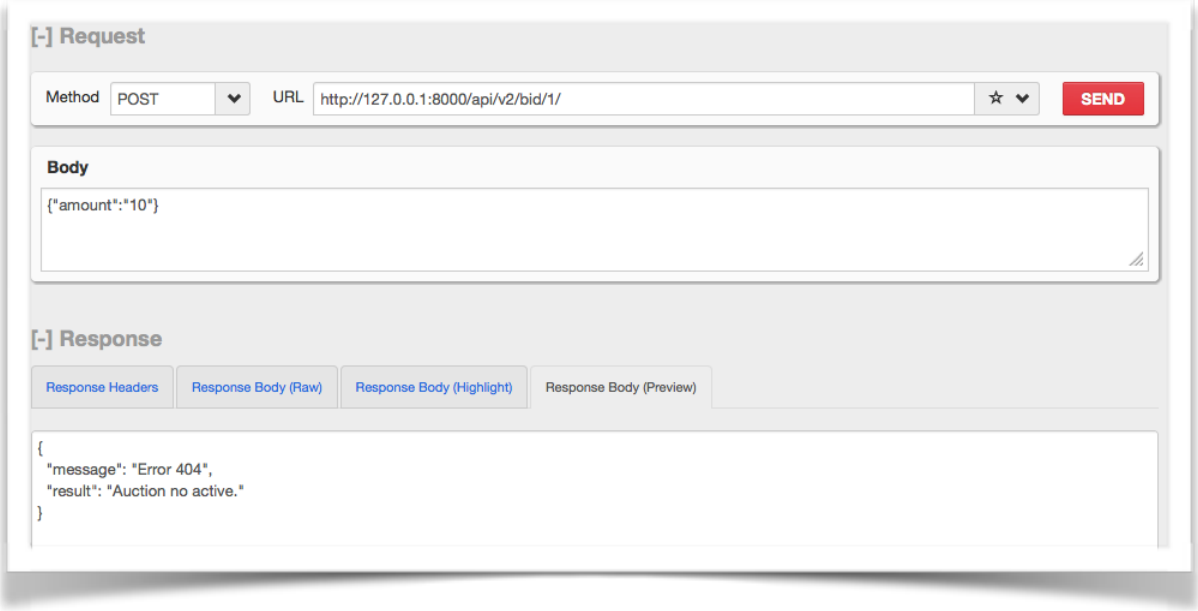
This approach is working well, but i would like have implemented optimistic logic (i don't have enough time).

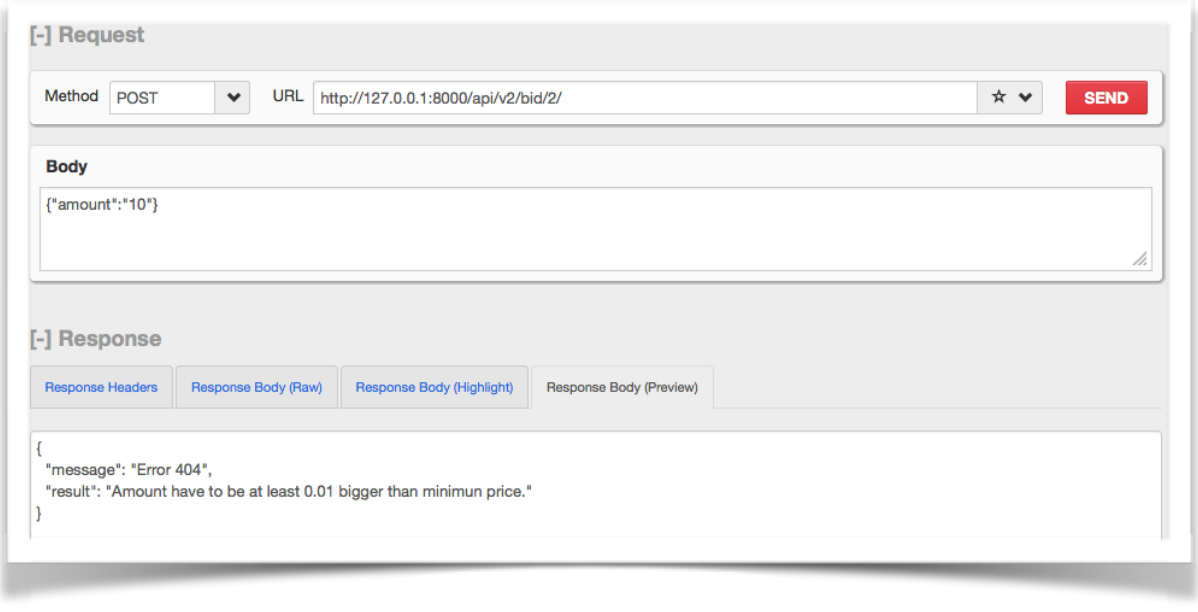
8. The REST API for your service. Is your service RESTful? What resources do you expose over the API and what is the URI to access them. Include concrete examples of HTTP request and responses for bidding in an auction.

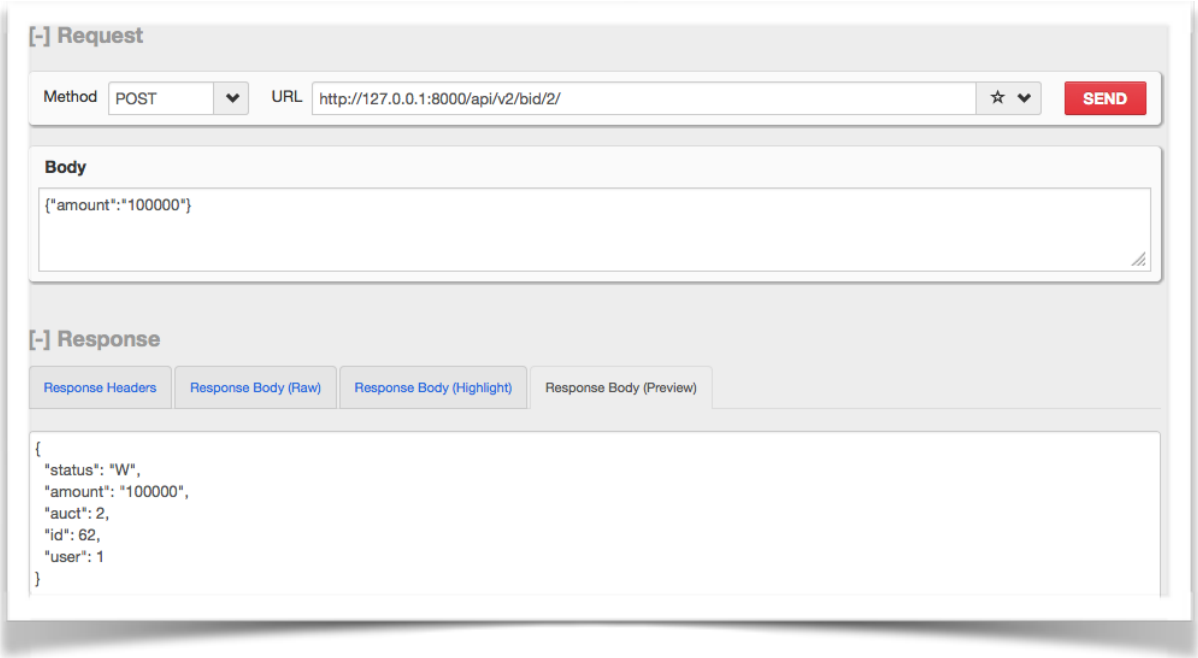
Yes, it's RESTful. The resources and the URIs are the following:

/api/v1/search/	-	Browse all active, due or adjudicated auctions.
/api/v1/search/<title>/	-	Browse auctions with title = <title>
/api/v2/bid/<auction>/	-	Bid on the auction = <auction> the amount that you send in the POST body

The following are three examples of request and responses for bidding in an auction:







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

I have tested the *createAuction* form, and the *add_auction*/*save_auction* views. I have tested what happen if somebody try to add an auction when he is a login user, a non login user and what happen if he finally submit a *Yes/No* in the confirmation form.

10. How have you implemented the language switching mechanism?

To the *settings.py* file I have had to add the following lines:

MIDDLEWARE_CLASSES: '***django.middleware.locale.LocaleMiddleware',***

LOCALE_PATHS = ('locale',)

I also need to import: ***from django.utils.translation import ugettext as*** _

Now in all my views and templates when I have a string that I want to translate to other language, I have to define it like this: ***_("string")***.

When i have finished with the last task, i have to run the following commands:

django-admin.py makemessages -a

Then I have to edit the file */YAAS/YAASApp/locale/lang/LC_MESSAGES/django.po* and insert the strings translations there. Finally run the following command for compile the file:

django-admin.py compilemessages