Development of server-side web services

John Nordqvist, 24461-25-2017

Link to GitHub repo: https://github.com/it-teaching-abo-akademi/dt00bt67-server-side-web-services-project-2018-DerJontte

Python Anywhere deployment: http://derjontte.pythonanywhere.com/

Implemented requirements:

REQ1.1 Any anonymous user can create a new user account.

REQ2.2 A logged in user can change his/her email address

REQ2.3 A logged in user can change his/her password.

REQ3.1: Any registered user can create a new auction.

REQ3.2: When creating an auction, the system registers information.

REQ3.3: the user must be asked for a confirmation before creating a new auction.

REQ3.3.1: a confirmed auction will be stored in the system

REQ3.3.1: an unconfirmed auction will NOT be stored in the system

REQ3.4: The system must confirm by email to the seller that an auction has been created.

REQ3.5 (optional): the email message includes a special link which would allow the seller to modify the description of the created auction without logging.

REQ3.6 A user (not the seller) can bid on an active auction.

REQ3.7 An administrator can ban an active auction.

REQ3.8 The seller can update the description of the auction.

REQ4.1 The seller of an active auction can change the description of the item(s) for sale.

REQ5.1 anonymous users must be able to browse the list of active auctions

REQ5.2 anonymous users must be able to search auctions by title.

REQ5.3 registered users must be able to browse the list of active auctions

REQ5.4 registered users must be able to search auctions by title.

REQ6.1 Any registered user can bid for an active auction, except the seller.

REQ6.2 The minimum bid increment is 0.01. Only two decimal places are considered when bidding.

REQ6.3 A seller cannot bid on own auctions.

REQ6.4 The application must show to the user the most recent description of the auction before accepting bids from the user.

REQ6.5 A new bid should be greater than any previous bid and the minimum price.

REQ6.6 A bidder cannot bid on an auction that he/she is already winning.

REQ6.7 the seller must be notified by email that a new bid has been registered

REQ6.8 the latest bidder must be notified by email when a new bid has been placed

REQ6.9 (optional) Soft deadlines. If a user bids at an auction within five minutes of the auction deadline, the auction deadline is extended automatically for an additional five minutes. This allows other users to place new bids.

REQ 7.1 An administrator user can ban an active auction that does not comply with the usage terms of the site.

REQ 7.2 A banned auction is not deleted from the system.

REQ 7.3 The seller and all the bidders are notified by email that the auction has been banned.

REQ 7.4 It is not possible to bid on a banned auction.

REQ 7.5 Banned auctions are not shown in the list of auctions neither in search results.

REQ 7.6 Banned auctions are not resolved.

REQ 7.7 Admin should be able to see the list of banned auctions.

REQ8.1 After the end date of an auction the system should automatically resolve the auction by electing the winner. The winning bid is the bid with the largest offer.

REQ8.2 Resolving auctions must be initiated automatically by the system and not by a user.

REQ8.3 All the bidders are notified by email that the auction has been resolved.

REQ8.4 The seller notified by email that the auction has been resolved.

REQ10.1 The web application should be able to handle multiple concurrent user sessions.

REQ11.1 All the auctions are created by default in EUROs.

REQ11.2 A user should be able to visualize the prices of the auctions and bids in a different currency based on the latest exchange rate.

REQ11.3 The exchange rate for the currency should be fetched regularly (on each request or at interval) from sites like https://currencylayer.com (free accounts available) and http://fixer.io/ .

REQ12.1 - A user should be able to send a GET request for browsing the auctions and fetching a specific auction by ID.

REQ12.2 - A user should be able to send a GET request along with a search string parameter to search auctions by their titles.

TREQ4.1.1 You must create automated functional tests for the use cases UC3

TREQ4.1.2 You must create automated functional tests for the use cases UC6

DREQ4.3 Deploy your final project at pythonanywhere.com

Python packages used beside Django:

certifi==2018.10.15

chardet==3.0.4

colorama==0.4.0

django-cron==0.5.1

django-session-timeout==0.0.3

django-sslserver==0.20

djangorestframework==3.8.2

httpie==0.9.9

idna==2.7

Pygments==2.2.0

pytz==2018.5

requests==2.20.0

six==1.11.0

urllib3==1.24

Implementation of session management

Sessions are maintained through django's built in sessions framework using the request.session['key'] – commands.

Confirmation when adding a new auction

When the user has filled out the form for the new auction, the information is POSTed to a confirmation method withinin the same view. The method presents the data to the user, and if the user chooses to click 'Yes', the data is committed to the database. If the user chooses 'No' or never answers the question, the auction is not saved.

Resolving of auctions

The resolving is initiated through django-cron, which in turn is being initiated by crontab every 5 minutes.

Concurrency

Concurrency is solved by always checking critical pieces of information within an atomic block of code. When a user makes a bid in UC6, the bidding is initiated by entering the desired bid and preliminarily submitting it. The user is then presented with a confirmation page, and when he/she confirms the bid, an atomic block of code locks the database, fetches the current description and highest bid from the database, and if the description is exactly the same as the one the user was presented with AND the highest bid is smaller than the one the user is about to make, the bid is committed to the database and the lock on the database released. This is a very inefficient way of handling concurrency when the number of users grow, but in very small systems it works pretty well.

REST API

The API supports listing, searching and retrieving auctions. Bidding is not implemented.

To get a list of all active auctions in JSON-format, send a GET to http://derjontte.pythonanywhere.com/api/auctions/ . Be sure to include the trailing / in the URI!

To fetch a specific auction in JSON-format, send a GET to

http://derjontte.pythonanywhere.com/api/auctions/<auction_id>, for instance

http://derjontte.pythonanywhere.com/api/auctions/12 . Note that there is no trailing / when fetching an auction.

To search the list of auctions, send a GET to http://derjontte.pythonanywhere.com/api/auctions/search=<query>, for instance http://derjontte.pythonanywhere.com/api/auctions/search=serv . The results are presented in JSON-format.

The API is not strictly RESTful, since the URI of the search-function includes a verb.

Functional tests

To be able to do any further testing, I first create three different users in the test database. I then go on to create a new auction, which requires both the initial POST and the confirmation to work correctly. Assertions are made regarding both HTTP responses and the reply strings that the app itself presents as human readable strings.

For UC6 I start testing with registering a bid that is correctly made and confirmed and asserting it is inserted into the database. After that I test that the same user can not make another higher bid on the same auction. Finally, I try to get another user (not the seller) to make a bid that is not higher than the previous bid, and assert that it fails.

I forgot to make a formal test case where the seller tries to bid on his/her own auction. (This is not possible, though, as was experienced countless times when I was beta testing the app together with my girlfriend.)