CS 145: Introduction to Databases

Stanford University, Fall 2017

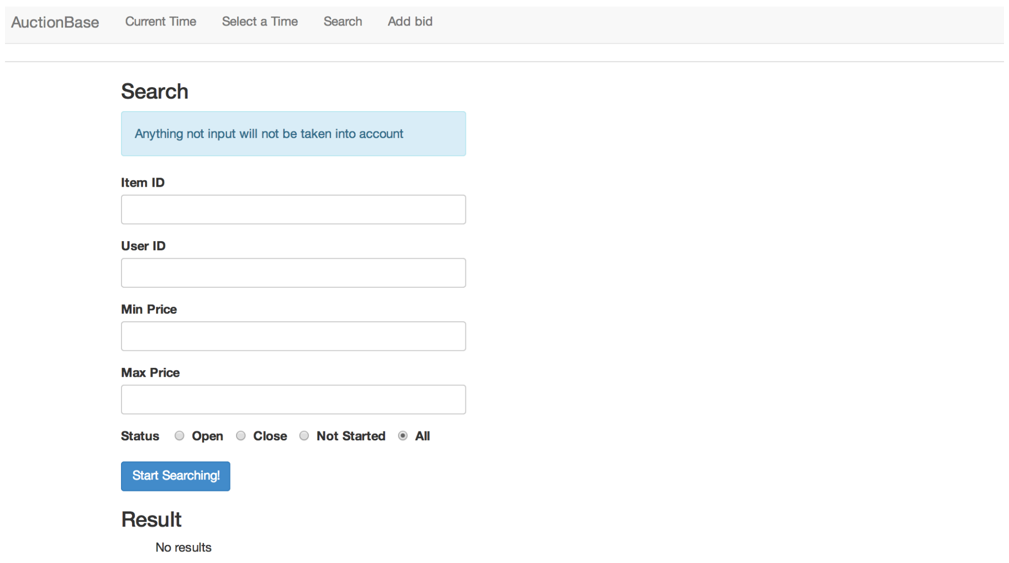
AuctionBase Project: Database and the Web

Part 3: To the Web!

Due Date: Thursday, December 7, 2:59pm

Overview

As a baseline, you will design a set of queries and updates for your AuctionBase system and create a simple web interface for them using the Python web.py framework. Before you get started, we recommend that you **read through the *Getting Started with web.py and Jinja2* supporting document in its entirety!**



Anything not input will not be taken into account

Status: Open Close Not Started [✓ All]

No results

Task A: Getting started

- **Step 1:** *Activate Personal CGI Service (if you haven't already)*
  Follow the instructions from Part 2 of the Project to active your personal CGI service on Stanford AFS. (We hope you've taken care of this already!) Remember: Activation can take up to 24 hours, so please do this as soon as possible. Once activated, you will have a new directory `~/cgi-bin/`.

- **Step 2:** *Copy web.py starter code*
  Copy the web.py starter code that we are providing into your `cgi-bin` directory:

```
cp -r /usr/class/cs145/project/pa3/web.py/* ~/cgi-bin/.
```

You should now see the following files and directories inside of `cgi-bin/`:

- `auctionbase.py` – Your "main" application. Responsible for handling requests from the browser.
- `sqlitedb.py` – Your database manager. Responsible for interacting with your database.
- `templates/` – Directory that contains your template files that correspond to your various URLs. Every web page will require a new template file in this directory.
- `lib/` – Directory that contains the library files for web.py and Jinja2. **Warning: Do not modify any of the files in the `lib/` directory!**

- **Step 3:** *Generate your SQLite Database*
  Modify your `runParser.sh` script to generate your SQLite database using a smaller JSON dataset `items-p3.json`, posted on the class website. Then, using your `createDatabase.sh` script from Part 2 of the project, generate your SQLite database (`.db`) binary file and move it into your `web.py` directory.

- **Step 4:** *Familiarize yourself with web.py and Jinja2*
  If you haven't already, go ahead and read through all of the *Getting Started with web.py and Jinja2* supporting document – this should give you a detailed guide of the starter code, and will demonstrate how to complete the required functionality for the assignment. (We **strongly** recommend that you do this, especially if this is your first time building a web application in Python!)

Task B: Required functionality

We will be testing your web app locally on the corn machines. You can test your web app on cgi-bin at `http://www.stanford.edu/~yourusername/cgi-bin/auctionbase.py/urlhere` (see below for example urls) or you can run your server locally by running `python auctionbase.py 8080` where 8080 is some specified port number. If you test locally, you can test it with urls like: `http://localhost:8080/urlhere`. If you choose to develop locally, be sure to test your web app on the corn machines too! Make sure to run our sanity checker on your code:

```
sh /usr/class/cs145/bin/proj3_sanity_check.sh
```

The functionality of your final AuctionBase system is somewhat flexible (you can add additional functionality if you want). However, you must implement at least the following basic capabilities in order to receive full credit on the project:

- Ability to manually change the "current time."
  - a GET request to `http://localhost:8080/selecttime` should render a webpage with a form to input a date and time
  - a POST request to `http://localhost:8080/selecttime` with the following parameters should change the time, viewable at `http://localhost:8080/currtime` in the following format: `yyyy-MM-dd hh:mm:ss`
  - When making a POST request to `http://localhost:8080/selecttime`, your web parameters must be named:

```
MM   # Two digits representing the month (0-12)
dd   # Two digits representing the date (1-31)
yyyy  # Four digits representing the year
HH   # Two digits representing the hour (0-23)
mm   # Two digits representing the minutes (0-59)
ss   # Two digits representing the seconds (0-59)
```

```
enterame # String representing a username
```

- Ability for auction users to enter bids on open auctions.
  - a GET request to `http://localhost:8080/add_bid` should render a webpage with a form to input ItemId, UserId, and bid amount.
  - a POST request to `http://localhost:8080/add_bid` with the following parameters should create a new bid, if valid (meeting your constraints from part 2).
  - When making a POST request to `http://localhost:8080/add_bid`, your web parameters must be named:

```
itemID  # An integer
userID  # A string
price  # A float price
```

- Ability to browse auctions of interest based on the following named input parameters:
  - a GET request to `http://localhost:8080/search` should render a webpage with a form to input the parameters below.
  - a POST request to `http://localhost:8080/search` should render a list of results, displaying at least the full item name and linking the item name to a view item page. (Be sure to use relative urls!)
  - When making a POST request to `http://localhost:8080/search`, your web parameters must be named:

```
itemID  # Integer
userID  # String
category  # String
description  # item description
               # (This should be a substring search,
               # i.e. not an exact match.)
minPrice  # float min price
maxPrice  # float max price
status  # String, one of 'open', 'closed', 'notStarted', 'all'
```

  Note that these parameters are compositional, i.e. you should be able to browse by category **and** price, not category **or** price
- Ability to view all relevant information pertaining to a single auction. This should be displayed on an individual webpage at `http://localhost:8080/view` where the web parameter named `itemID` specifies the specific auction item, and it should display all of the information in your database pertaining to that particular item. This page should be linked to from your search results. In particular, this page should include:
  - all item attributes (title, description, etc.)
  - categories of the item
  - the auction's open/closed status
  - the auction's bids. You should also display all relevant information for each bid, including
    - the name of the bidder
    - the time of the bid
    - the price of the bid
    - the returned HTML must contain the string `No bids` if none exists.
  - if the auction is closed, it should display the winner of the auction (if a winner exists)
    - the returned HTML must contain the string `Winning Bid: UserID` if the winning bid is by UserID and the string `No Winning Bid` if none exists.
- Automatic auction closing: an auction is "open" after its start time and "closed" when its end time is past or its buy price is reached.

- When we view all the details of a particular item, your HTML response must contain the string `Status: OPEN` or `Status: CLOSED`

Furthermore, your AuctionBase system must support "realistic" bidding behavior. For example, it should not accept bids that are less than or equal to the current highest bid, bids on closed auctions, or bids from users that don't exist. Also, as specified above, a bid at the buy price should close the auction. Some of these restrictions may already be checked by your constraints and triggers from Part 2 of the Project; others may require additional triggers or code.

If you do decide to add additional triggers to your database, please create additional `triggerN_add.sql` and `triggerN_drop.sql` files to implement these, and include them as part of your submission. You should also be sure to update your `createDatabase.sh` script to include these extra trigger files. (See the submission instructions at the end of this document for more details.)

Full credit also requires general error- and constraint-checking as specified in Task C below. For starters, all of the constraints you implemented in Part 2 should be checked in your "live" AuctionBase system. Every error message must begin with `ERROR:` (with the colon).

Note that you can receive full credit on the project by implementing just the basic capabilities specified earlier, along with constraint-checking, error-checking, and a simple web interface. That is the standard against which projects will be graded. CS145 is not a user interface class and, again, you can receive full credit for a solid system with simple input boxes, menus, and simple HTML output tables. **However, under no circumstances should you be expecting the end-user to write SQL!**

Task C: Transactions, errors, and constraint-checking

Commands that modify the database need to be handled carefully, and you should group them into transactions whenever it makes sense for them to be executed as a unit. Using transactional behavior, each unit should either complete in its entirety or, due to failed constraints or other errors, should not modify the database at all. Constraint violations, and other errors due to bad input values or data entry, should be managed gracefully: It must be possible for users to continue interacting with the system after a constraint violation or error is detected, and the database should not be corrupt. You should inform users when errors occur, but your error message need not indicate the exact violation that caused the error.

If it helps, you may assume that AuctionBase has only one user operating on it at a time. Although transactions may be useful for database modifications and constraint-checking, you do not need to worry about transactions as a concurrency-control mechanism. That said, even without special effort your system may turn out to be fairly robust for multiple users.

Task D: Other Miscellaneous Requirements

- When you generate dynamic HTML pages from your program, please use relative paths, rather than absolute paths, for the links to the various URLs in your website. For example, make your links and forms point to `webpage` instead of `http://www.stanford.edu/~yourusername/cgi-bin/webpage`. Relative paths enable us to grade your project in our own webspace.
- You don't have to implement user authentication. For example, it's okay to ask the user to enter his/her username when bidding, without asking for a password.
- Please make your website accessible by user interface. For example, a user should not have to "know" a URL and type it directly into the browser address bar. They should be able to click on links to navigate.
- Lastly, a suggestion: it's a good idea to debug your queries directly in SQLite before hooking them into your web interface. Use the SQLite command-line interface first, to ensure that your queries are working properly and are finishing in a reasonable amount of time. In the command-line interface, you can kill runaway queries using `Ctrl-C`. Once you are certain your queries are working properly, incorporate them into your web interface.

Submission instructions

Congratulations! You've completed the AuctionBase project! ☺

To submit your work, first create a submission directory with the following:

```
Your web.py/ directory (WITH your .db binary file!)
parser.py
runParser.sh
create.sql
load.txt
constraints_verify.sql
trigger{1..N}_add.sql
trigger{1..N}_drop.sql
createDatabase.sh
```

Finally, prior to submitting your project, you should perform the following to verify correctness:

1. Run your parser: `./runParser.sh`
2. Run your database creation script: `./createDatabase.sh`
3. Move your auctions.db database file into your web.py directory: `mv auctions.db web.py/`
4. Visit your auction website and confirm all of the features work correctly.
5. Delete any extraneous files (such as .dat) created from running your parser: `rm *{,/*}.{dat}`
6. Run our sanity checker on your code:

   ```
   sh /usr/class/cs145/bin/proj3_sanity_check.sh
   ```

In particular, note that we **do** request that you include a fresh copy of your `.db` databse binary file as part of your submission. We will only generate your database file using your `createDatabase.sh` script if you break the autograder.

This means that you must include all necessary `*.sql` files that are required for `createDatabase.sh` to run properly! In particular, be sure to include any extra `triggerN_add.sql` and `triggerN_drop.sql` files that you may have added for Part 3!

Also, if you made any other modifications to your submission from Part 1 (such as your schema design, or your parser), **please also include those modified files as part of your submission**.

Once your submission directory is properly assembled, **with no extraneous files**, execute the following script from from your submission directory:

```
/usr/class/cs145/bin/submit-project
```

Be sure to select "Part3" when the script prompts you for which assignment you're submitting!

You may resubmit as many times as you like; however, only the latest submission and timestamp will be saved, and we will use your latest submission for grading your work and determining any late penalties that may apply. Submissions via email will not be accepted!