IAB207: Rapid Web Application Development
Developing an Auction Website

Week 13 Workshop: Demonstrate partially functional Web application

Code submission on Blackboard: Nov 1st, 2020: 11:59 PM

Overview

The last assignment for this unit is a group assignment in which you will develop a web application that uses the following framework:

1. HTML & Bootstrap CSS
2. Flask Templates
3. Flask WTForms
4. Flask SQLAlchemy
5. Flask Login

You will form **groups of 3 or 4 students**. Groups with a team size of 3 will need to develop the same functionality as teams with a size of 4. For this assignment, your group will need to nominate an online auction Website designed by one of your team members in the earlier assignments of the unit.

Project Deliverables

Please read the below deliverables (requirements) carefully as every team will be expected to address each of the requirements in their final submission.

You are free to re-use the code from the individual assignments of one of the team members to develop all pages again. Ideally, you may choose to re-use your previous work as it will save you time. As part of the project, the following needs to be developed and demonstrated:

Part 1: Project Requirements

The following are the features required in the Auction Web application

1. A landing page containing some useful message of the website showing some recently posted items. Also, your landing page can allow users to *find the items by category* (you are free support this functionality using a drop-down menu or a text search).
   Note: Text-based search functionality is optional, as it is an advanced feature to use text input to match information about the items and displaying them.
2. The items listed on the landing page should show the following:
   a. The # of bids made on the item
   b. The status of a listing – closed or open for bid.
   c. The highest bid.

   Note: A user can check the details of closed bids also.

3. A user should be able to select an item listing and *view the details of it*. The page should have an image of the item, description of the item, starting bid, the highest bid so far, and other item specific information you have designed.
4. If the auction is not yet closed, the user can *bid on the item* by providing a bid amount greater than the highest bid. The user can also *add the item to their watchlist*. If the auction is closed, the user cannot perform these operations.
5. A user can bid on an item by only quoting any price higher than the starting bid.
6. A user can view the items they have added to the watchlist with the following details:
    a. The # of bids made on the item
    b. The status of a listing – closed or open for bid.
    c. The highest bid
7. The application should allow a registered and logged in user to list an item for auction. Since the auction site is specific to a topic, the user will provide specific information about the item. In addition to the item details, the user needs to enter the starting bid.
8. The user who has listed an item can view all the bids and their respective bid amount.
9. The user can close the auction, in which case the highest bid is marked as "won" and the status of the auction is set to "closed"
10. A logged-in user can log-out of the application
11. A user can register to the auction website. The user should provide details of name, email-id, password, contact number, and address.
12. Error Handling:
    a. Handle instances in which a page is not found, and internal server errors within a page that allows users to navigate back to the landing/home page.
    b. A user should be authenticated (via login) to either list an item or bid on an item.
    c. User input validation – empty strings, incorrect input for specific fields, etc. should be correctly managed.

All your pages should include a navigation menu and should utilise data that is dynamically created and stored in a database solution (use SQLite).

Note: **The requirement of posting and viewing comments is not required for this assignment.**

Project Management

Teams are required to maintain a list of tasks and ensure that each task is assigned to individual members within the group. Your final report should contain the progress made by team members each week. You are free to allocate a task to one or more members in the team. In the final submission report, teams will be required to clearly indicate the contribution of individual members to the tasks. You are recommended to have a **private git repository** and share it with one of the tutors allocated to your team. We will provide the details on how to setup a git project shortly.

Please note: Each team will be sent a peer review feedback form. Each member will be reviewed by all other members of the team. The plan and the peer review feedback will be considered when allocating marks to team members. All team members need **not** get the same marking for the assignment. Please consider this when reviewing your team member

An example of a task definition, allocation and progress is shown below:

Task	Week1	Week2	Week3	Week4	Week5
Landing Page	Liam: Worked on the html template		Liam: Worked with the dynamic content		Completed verified all menu items
Registration	Sheru: Complete the form	Sheru: Tested the form validations		Integrated the Flask-login support	
Create item		Liam: Added create item support			
Database model	Abdul: Completed the models.py with all DB elements	Abdul: Tested all the objects and their relationships	Added Flask-login support	Sheru: updated the database model	Abdul: Checked all the database updates, queries
Error Handling					

Development Principles

You must develop code that:

1. Is well written with reasonable comments/documentation.
2. You may use the assessment starter code zip file that is provided to organize your code better.
3. Implements HTML, Bootstrap, and Python Flask (with other Flask modules) only.
4. Deploy the application (or show attempt of deployment) to the Heroku Cloud Platform. Instructions to do so will be provide through a short video tutorial.

Database Design

A database solution will be required to achieve desired functionality in your solution. Your database design will consist of four tables:

1. An items table containing the item details – you can add more tables to store more information or can choose to keep it simple if you see fit.
2. A bid table storing information about the bid price, bid status, bid date with relationship to a user and item
3. A user table storing information about user, including their login credentials.
4. A watchlist with items added by a user and the date it was added

Things to note

- All code should be written and contributed by team members.
- Use of JavaScript is optional. None of the functionality described in this assessment relies on JavaScript, however if you want to support some additional interactive functionality, you are free to do implement it using JavaScript.

Submission Instructions

Source Code

Each team is required to submit a ZIP file containing the following:

1. A report containing the following:
    a. Names of all team members
    b. screenshots of all the pages described in the test script should be submitted.
    c. The Heroku URL for your web application. If you were unable to deploy the application on Heroku, you must provide a detailed description of the steps you followed and what failed. Evidence proving that you tried this step should be presented (for example: screenshots of error messages).

2. A ZIP file with your python code (please include your sqllite db file) and the report.

3. Please do not include any **venv** directory (Python virtual environment) of any Python installation. Ensure also that you check your file size before uploading.

Test Script

The following sequence of tests will be carried out by the marking team.

1. The landing page of the application will be accessed. The 'search-by-category' feature that allows a user to view items by category will be tested.
2. The details of one of the items on the landing page will be assessed. The user should be able to the status of the item. The user can either 'bid on item' or 'add to watchlist'. The user who listed the item will be able to 'close the auction' and 'view all bids'
3. On trying to create item, the user should be prompted to a login page.
4. Three new users will be registered, and details of the registration page will be tested.
5. One user will login to the application.
6. The user will create two items. The item creation will be tested
7. The user will logout of the application. Logout functionality will be tested.
8. The 'bid on an item listing' functionality will be tested. The user should be taken to the login page if a user is not logged-in
9. The user will login and enter bid amount that is lower than the starting bid. The user will be prompted with an appropriate error message. The user will enter a valid bid amount.
10. Another use will bid with a higher bid amount and add to Watchlist
11. The user who has listed the item will close the auction
12. Relevant updates to the status of the auction will be checked.

Marking

Web Application Demonstration (35 marks)

Please note that the final mark of each person in the team gets is based on their contribution and uses the rating provided by the team members.

Requirements	Fully Functional Requirements are fully met, and the page is fully functional without errors. *A submission of this standard succeeds in the fundamental requirements. The solution is thorough and well tested.*	Incomplete or Errors Present Most of the requirements have been met and the page is mostly functional with a few errors. *In a submission of this standard, there would be evidence of the minimal requirements advised in the project deliverables of the assessment. There may be indications of some errors in the solution, however these would be minimal, and would not impair the usage of the solution.*	Missing Content Requirements misunderstood or ignored, does not meet requirements. *A submission of this standard would not function (or partially function) as a minimal viable solution. There would be presence of serious errors that would impair the solution's basic processes.*
Landing page *(#1, #2)*	5	3	2
Search by category (drop-down/text) *(#1)*	3	2	1
Item details page *(#3)*	5	3	1
Bid on an item with validation (#4, #5)	5	2	1
Add and View Watchlist *(#6)*	4	2	1
Create item listing & View bids & Close auction (*#7, #8 & #9*)	7	4	2
Register a user (#10)	3	2	1
Login/Log-out of user/Error handling (#11 & #12)	3	2	1
TOTAL	**35**	**20**	**10**

Development and Deployment (5 marks)

TOTAL: 5 Marks					
Criteria	**5**	**4**	**3**	**2**	**1**
Deployment or attempted deployment on Heroku Cloud Platform (file #5 in the source code zip)	Successfully deployed the application with Flask and PostgreSQL	Successfully deployed the application with Flask and SQLLite	Unsuccessful deployment, application failed to start on Heroku.	Unable to deploy the software on Heroku. There were some missing steps	Unable to use the Heroku interface.

Team Collaboration and Participation (10 marks)

TOTAL: 10 Marks					
Criteria	**5**	**4**	**3**	**2**	**1**
Workshop participation through exercise completion.	The travel Web application as developed in all workshop exercises is complete.	The travel Web application as developed in the workshop exercise has most components including the database	The travel Web application as developed in the workshop exercise has some components	The travel Web application as developed in the workshop exercise has used Flask templates	Only the HTML exercises are complete
Team presents an evidence of progress either through Github/weekly update shared with the Tutor (Week 12-Submission)	Progress has been maintained weekly as available in GitHub code updates	Progress shared for few weeks through Github or updates presented to the Tutor	Progress has been shared partially through Github or updates presented to the Tutor	Team has minimally shared evidence of weekly progress	Team has not shared any evidence of progress made.
Please note that peer review is an individual component, that is, your individual marks will depend on how your team has rated you.					