Auction Website About

Overview

This auction website is a dynamic online marketplace designed to connect sellers and buyers. The purpose is to allow a platform to host auctions that are accessible to users of all levels. Users can be both sellers and buyers, listing several items or participate in bids. The target audience is individual sellers looking to find new homes for their belongings and bargain hunters to find their next treasures.

In the rest of the documentation, you will find the technical information and our thoughts over the development journey. Below are the official requirements for this project:

Project ID	10
Company/Organization / Research Group Name	MSCI
Contact Person Full Name	An Truong/Vu Nguyen
Contact Number	4054107398
Contact Email	an.truong@msci.com
Website (Optional)	0
Project Problem Identification	Create an online auctioning application (think eBay).
Project Objectives	The application should be designed with both the buyer and auctioneer in mind. Platform stability will be a priority. (Imagine a platform that crashes just as an auction ends due to load...)
Project Requirements	Auctioneer: - Add items - Set initial price - Upload pictures - Set deadline - Ability to edit items User: - Add bid - Add bid limit Application: - Load testing - Implement queuing system
Any other Comments to the students	All features/configurations should be documented in a way that another group can continue where work was left off. All testing should be documented well.

Core Features

1. **Item Listing:**
   a. Users can create listings for auction items, providing necessary details such as item title, auction end date and time, starting price, item description.
2. **User Registration:**
   a. Prospective users can register by providing a unique username.
3. **Real-time Bidding:**
   a. Users can place bids on items.
4. **Feedback Mechanism:**
   a. After performing actions such as adding items or users, the system provides feedback to the user, confirming or rejecting the action taken.

Technology Stack

We use Django as our main web framework since it can provide back-end and front-end functionality. In this section, we will detail it as back-end since its main implementation is to handle server-side logic.

1. **Frontend:**
   a. HTML and CSS

    i. These are standard technologies for creating web pages and are supported by all web browsers. HTML structures the content on web pages, while CSS is responsible for style and layout. Within the Django framework, the templating engine allows developers to generate HTML dynamically, populating placeholders with data from the database. Additionally, Django's management of static files, such as CSS stylesheets and images, enables a separation of content and presentation, ensuring that the frontend is both functional and visually appealing.

2. **Backend:**
    a. Django
        i. Django is a comprehensive Python web framework that allows for rapid development and clean and practical design. This was chosen because it contains many built-in features such as ORM (Object-Relational Mapper) for database interactions and a templating engine (for the front-end). Django's architecture is designed to efficiently handle high volumes of data and traffic, making it an ideal choice for the backend of an auction website that requires reliability and scalability.
3. **Database:**
    a. MySQL
        i. MySQL is an open-source relational database management system. This was chosen for its open source and free use characteristics. It is utilized to store data such as user information, item listings, and bids. The database is used to store procedures and execute them based on calls from the backend.
4. **API or Server Communication:**
    a. RESTful API
        i. RESTful API is used for server communication, which uses standard HTTP methods like GET, POST, PUT, and DELETE for creating, reading, updating, and deleting resources. This approach was chosen for its simplicity, scalability, and statelessness, making it well-suited for web applications where the frontend and backend operate independently. In the context of the auction website, the RESTful API allows the frontend to communicate with the backend to perform actions such as submitting bids, querying item data, and managing user accounts.

We chose this architecture for its potential in scalability and robustness for future improvements.

Notes from the Developers

This project started in January 2024 for CS 4273: Capstone Design Project (SPRING 2024) taught under Dr. Mansoor Abdulhak at the School of Computer Science in the Gallogly College of Engineering at the University of Oklahoma. This is the result of the collaborative effort of developers consisting of Michael Bartlett, Ryan Fitzgerald, Claire Nguyen, Uzo Ogbanufe, Jazmin Ramirez, and Gabe Wiley. And a huge thank you to our mentors An Nguyen and Vu Truong for guiding us through this semester!

README for GitHub

Auction Website

A full stack auction website created using Django and MySQL, designed to connect sellers and buyers in an accessible online marketplace. Users can list items for auction, bid on items, and manage their listings, all within a user-friendly interface.

Table of Contents

Quick Start

Prerequisites

Ensure you have Python 3.8 or newer, MySQL, and MySQL Workbench set up on your local machine. You should have a database created for this project, with a user granted all privileges on the database.

Clone Repository

Clone this repository to your local machine using the following command:

```
git clone https://github.com/uzo-ogbanufe/cs_capstone.git
```

Install Dependencies

Install the required Python packages using pip:

```
pip install mysqlclient django
```

Configuration

In the project file `auction_site/settings.py`, replace `USER` and `PASSWORD` with your MySQL root username and password.

In your MySQL database, open the SQL scripts from the `database` folder and run `create_database.sql`, `database_procedures.sql`, and `test_data`.

Run Server

Navigate to the project's directory and run the Django development server with:

```
python manage.py runserver
```

The server will start, and you can access the application through `localhost:8000` in your browser.

Technologies

Django, HTML and CSS, and MySQL

User Stories

Feature: User Registration and Login

User Story: As a user, I want to register and log in to the auction platform to participate in auctions.

Acceptance Criteria:

- Users can create a new account by choosing a unique username.
- Upon account creation, users receive a confirmation message.
- If the username is taken or registration fails, users are informed with an appropriate error message.
- Registered users can log in using their username, which grants them access to auction functionalities.

Feature: Add Items

User Story: As a seller, I want to list items for auction easily so that I can sell them efficiently.

Acceptance Criteria:

- Sellers can list an item by providing its title, auction end date and time, starting price, item description, and their username.
- The system validates that the auction end date is in the future and confirms the listing with a success message.
- An error message is displayed if the listing fails, providing the reason for failure.
- Listed items are added to the auction inventory where buyers can browse and bid on them.

Feature: Bidding on Items

User Story: As a bidder, I want to place bids on auction items so that I can win the items I desire.

Acceptance Criteria:

- Bidders can enter their bid amounts which must be higher than the current bid.
- The system provides real-time updates of the current highest bid and auction countdown.
- Bidders receive instant feedback on bid submission, including confirmation or detailed rejection reasons.
- The system handles bid validation, ensuring the auction is still open and the bid is acceptable.
- A clear UI is provided for bid placement, with necessary validation and real-time features using WebSocket or similar technology.

Feature: Auction Management

User Story: As an auction administrator, I want a robust system that maintains auction integrity, especially under high traffic.

Acceptance Criteria:

- The system manages auction data, including current bids, item details, and user accounts.
- A concurrency mechanism ensures bids are processed fairly and accurately.
- Administrators can view and manage auction listings, user accounts, and bids through a backend interface.
- The system reports on auction activity and provides alerts on important auction events.

By focusing on the features, these user stories ensure that the team's efforts are directed toward delivering complete functionalities that integrate the database, backend, and frontend concerns. Each story encapsulates the end-to-end process of a single feature from the user's perspective.

Sprint Retrospectives

Here you will find summaries of what we learned after every sprint. Please look into the pages for that specific information.

Sprint 1

We spent this sprint designing and laying out the foundation of our project. We chose what technologies to use, divided the work by front-end, back-end, and database. We tried out several new things like Flask vs. Django and Jira vs. Excel. We had the mentors help us a lot in this sprint, and with their guidance, we created a project data flow sheet to map out the functions we can start with. We lastly wrote out new tasks for Sprint 2 based off of the things we needed to continue working on and to improve upon this sprint.

Sprint 2

We created tasks to write at least 3 functions: AddUser, AddItem, and SearchItem. We found that these were too many tasks and we could only finish AddUser and part of AddItem. SearchItem (and bidding on items) will have to be saved for Sprint 3. We also had formal review and it went fine. After this sprint, we foresaw that splitting into teams from front-end, back-end, and database did not work out. There was too much overlap and we were going back and forth on what was our task and what was not. Now we plan to work by feature, so every branch is a ticket and every ticket is basically working on a whole feature.

Mansoor Abdulhak on LinkedIn: #ouresearch #mentorshopmatters

Sprint 3

We brought mentors in for the start of Sprint 3 for feedback and advice. For Sprint 3, we have decided to drop search-items and focus on bid-items and testing our features (and documentation). It was recommended that learning how to test our functionality should get more priority over bidding items since we already have experience in making things work, but not making things break. We split up into working on features rather than working on front-end, back-end, and database. There was too much overlap when we were working on Sprint 2 when we had teams split by web development layers. Working by feature improved our production and collaboration.

However, there were still many tasks that were not done in this sprint due to personal issues (laptop breaking) or other class workload, and the time it took to spend learning how to test. These are things we are looking to improve in Sprint 4. Because a lot of the bidding functionality is a little out of reach for us, we are looking to implement the minimum viable product (MVP) for bidding-- this would be a user is able to input a number as a bid and the bid goes up until time runs out.

We also learned that we have to document bugs and new changes and put them in a change log/release notes.

Sprint 4

For Sprint 4, we focused on finishing up documenting testing. We also implemented rudimentary bidding and updated the CSS to make the webpage look nice. We improved since our last 3 sprints making sure not to overload ourselves with more tasks and splitting by functions was the most efficient way to collaborate on the auction website. The presentation is this Friday, so we were a little rushed finishing up the posterboard and preparing for the presentation.

Project Documentation

This section will discuss the structure and set up of the cs_capstone Django project as well as the database details. Please look into the other pages for more information.

Project Design

In this page, we outline the structure of the project and how data will flow amongst each other.

- add_user
- add_item
- get_item
- login

Each function/app that shares a common theme gets its own Django app (a folder basically within the directory).

```
auction_site/                # The root project directory
|
├── add_item/                # Django app for adding auction items
|   ├── migrations/          # Database migrations for propogating changes
|   ├── static/              # CSS style sheets and others needed for front-end
|   ├── templates/           # HTML templates for item addition views
|   ├── admin.py             # Configuration for Django admin interface
|   ├── apps.py              # Application configuration
|   ├── forms.py             # Creating forms, validating user input, and converting input to Python data types
|   ├── models.py            # Data models for auction items
|   ├── tests.py             # Test cases for the add_item app
|   ├── urls.py              # URL patterns for routing within the add_item app
|   └── views.py             # Views for handling item addition requests (contains most of the logic)
|
├── add_user/                # Django app for user registration and profile management
|   ├── ...                  # Similar structure as add_item
|
├── get_items/               # Django app for item retrieval and display
|   ├── ...                  # Similar structure as add_item
|
├── login/                   # Django app for item retrieval and display
|   ├── ...                  # Similar structure as add_item
|
├── auction_site/            # Main project configuration and routing
|   ├── __init__.py          # Indicates that this directory should be treated as a Python package
|   ├── settings.py          # Global settings for the Django project (MySQL login goes here)
|   ├── urls.py              # The top-level URL patterns for the project (where you list each pages' url)
|   ├── wsgi.py              # Entry-point for WSGI-compatible web servers
|   └── asgi.py              # Entry-point for ASGI-compatible web servers
|
├── db.sqlite3               # Default SQLite database file
└── manage.py                # Command-line utility for administrative tasks
```

add_user

Frontend

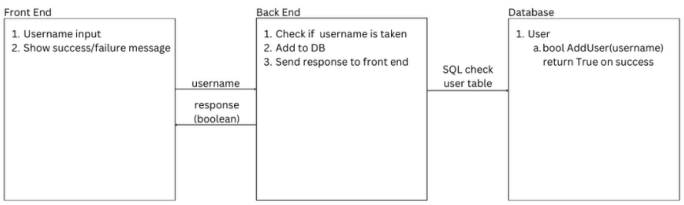
- Collect username from user
- Send collected information to backend (Exact format to be determined)
- Collect response from the backend and display confirmation/rejection to user

Backend

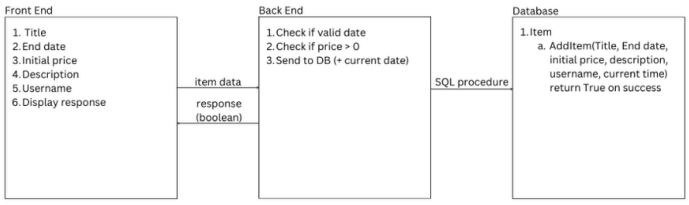
- Collect above information from frontend
- Call AddUser(username). See 3 Database Procedures for more detailed information
- Collect response from database and forward it to the frontend

Database

- Collect username from Backend and add a new user to the table.
- Send True to backend if it was successful, and return False otherwise



add_item

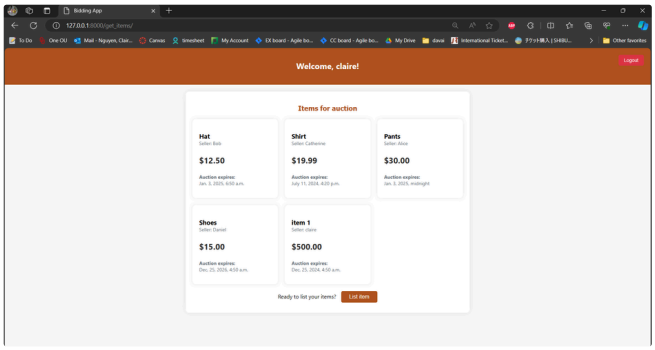

Front End
1. Title
2. End date
3. Initial price
4. Description
5. Username
6. Display response
item data
response
(boolean)
Back End
1. Check if valid date
2. Check if price > 0
3. Send to DB (+ current date)
SQL procedure
Database
1. Item
a. AddItem(Title, End date, initial price, description, username, current time) return True on success

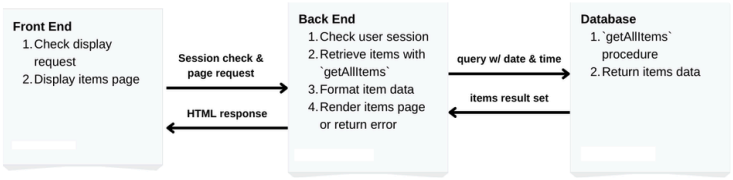
get_item

Purpose: Show what items are available for auction when the user logs in.

This page only shows when a user is logged in.

1. **Check User Login**: The function first checks if a user is logged in by looking for 'username' in the session. If the user isn't logged in, it redirects them to the login page.
2. **Retrieve Data**:
   - The function connects to the database and runs a stored procedure called `getAllItems`, passing the current date and time as an argument. This procedure retrieves all items from the database.
   - The results are fetched and stored in a list.
3. **Format Data**: Each item's price, taken from the database in cents, is converted to dollars and formatted as a string (e.g., $10.00). Other item details are kept as they are.
4. **Render and Send to User**: The function then prepares an HTML page by placing the formatted items and the username into the template. This page is sent back to the user's browser for display.
5. **Handle Errors**: If any errors occur during the process, the function sends back an error message in a JSON format.



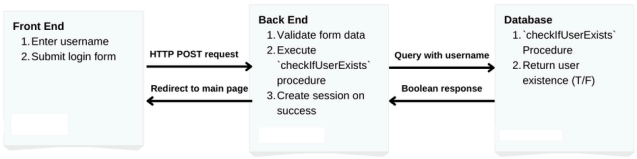


login

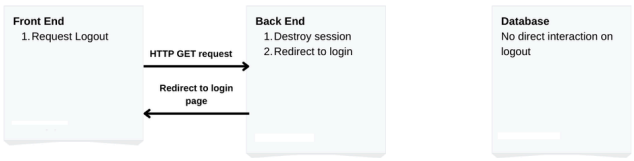
Purpose: Allow users to log in and log out, and have a session for them to keep track of their activity.

1. **Login Process**:
   - **User Input**: Users enter their username through a form on the frontend.
   - **Username Validation**: When the form is submitted, Django checks if the form data is valid using `form.is_valid()`. If valid, it retrieves the cleaned username.
   - **Database Check**: The backend then sends this username to a stored procedure in the database called `checkIfUserExists`, which checks if the username exists and returns true or false.
   - **Session Creation**: If the username exists, Django starts a session for the user by saving the username in the session data. It then redirects the user to the `get_items` page, where they can view items.
   - **Error Handling**: If any errors occur during the process, such as during the database check, an error message is displayed on the login page.
2. **Logout Process**:
   - **Session Termination**: When a user chooses to log out, Django deletes their session information, effectively logging them out.
   - **Redirection**: The user is then redirected to the login page and shown a message that they have been logged out.
3. **Handling Already Logged-in Users**:
   - If a user who is already logged in tries to access the login page, Django will redirect them directly to the `get_items` page, preventing them from logging in again.

Logging In



Logging out



Setting up Django

We chose Django because of its capabilities to handle high amounts of traffic and data.

Run this command to begin a Django project:

`start project django-admin startproject yourprojectname`

When the command is run for a project named `auction-site`, Django will automatically have this file structure:

```
auction_site/
    manage.py
    auction_site/
        __init__.py
        settings.py
        urls.py
        asgi.py
        wsgi.py
```

There should be no need to run this command if you are building off of the pre-existing project. Please refer to README for GitHub to run the Django project.

1. **Top-level Directory (`AUCTION_SITE`)**: This is the root directory the entire Django project. It's named after the project and contains all the components of your Django application, including all the Django apps (like `add_item`, `add_user`, `get_items`), project-wide files (like `manage.py`), and the project's configuration directory (which has the same name as the project by convention).
    a. `manage.py`:
2. **Inner** `auction_site` Directory: This inner directory is the actual Python package for your project. It usually contains settings, URL configurations, WSGI and ASGI applications for deployment, and sometimes static files and templates that are used project-wide. Here's a breakdown:
    - `__init__.py`: This empty file tells Python that the directory should be considered a Python package.
    - `settings.py`: Contains all the project settings, including database configurations, middleware, installed apps, template settings, static files settings, etc.
    - `urls.py`: The URL declarations for the project; a “table of contents” of the Django-powered site.
    - `wsgi.py` / `asgi.py`: These files are used to help your Django application communicate with the web server. You would use `wsgi.py` for WSGI-compatible web servers and `asgi.py` for ASGI-compatible servers.

The reason for having a project package with the same name inside the top-level directory is to keep the project's package namespace clean and to avoid name clashes with other packages. This pattern is also good for encapsulating the settings and main configurations of the project, so it is clear to show where the configuration starts and ends.

Database

This section highlights our database documentation. This includes how to set up the database, the database schema, and the database procedures we have. Please look into the other pages for that information.

1 Setting up Database

Installing MySQL

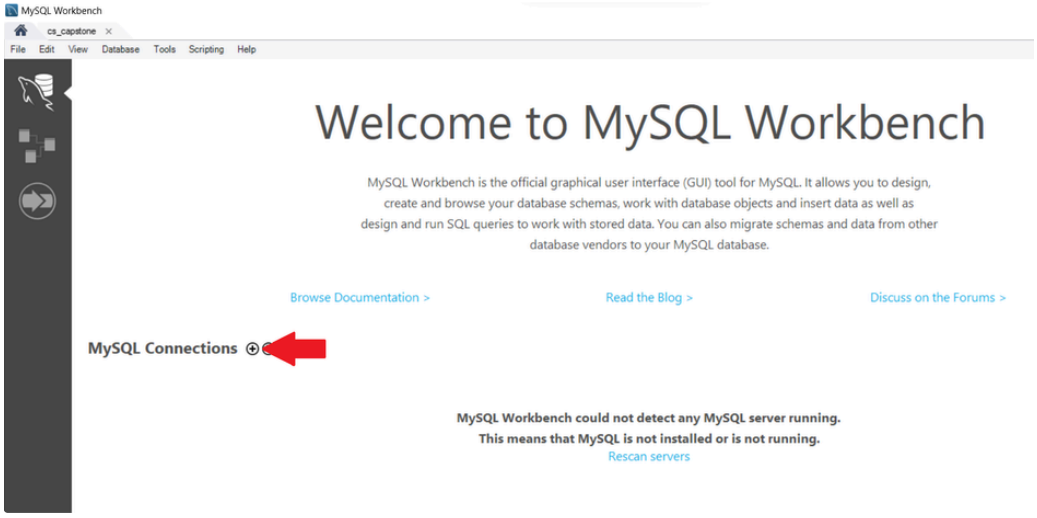
In order to set up the database for the project, you must first install MySQL Server (8.0.31 or later) and MySQL Workbench.

Windows

For Windows, the easiest way to do this is to install the MySQL installer from this page. From there, you can use the installer to install MySQL Server and MySQL Workbench. You can leave most options as default, but when downloading MySQL Server, you will be asked to enter a root password. Make sure to remember what password you enter.

Setting up MySQL Workbench

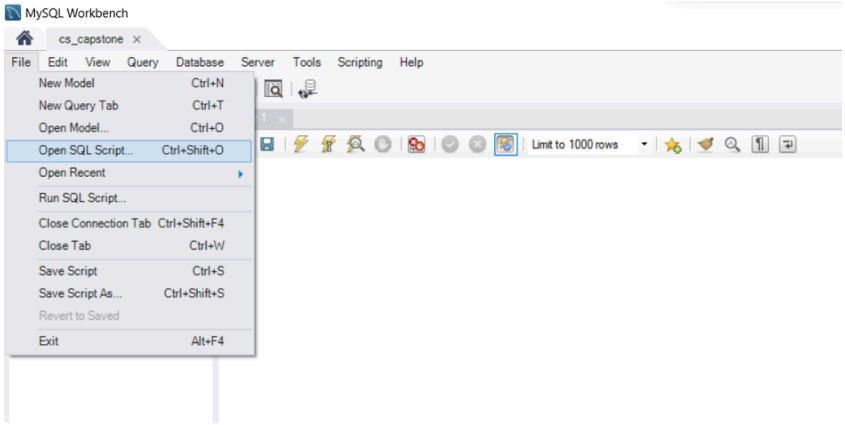
Once you've installed MySQL server and MySQL Workbench, open MySQL Workbench. On the home page next to the text "MySQL Connections" you should click the small plus button.



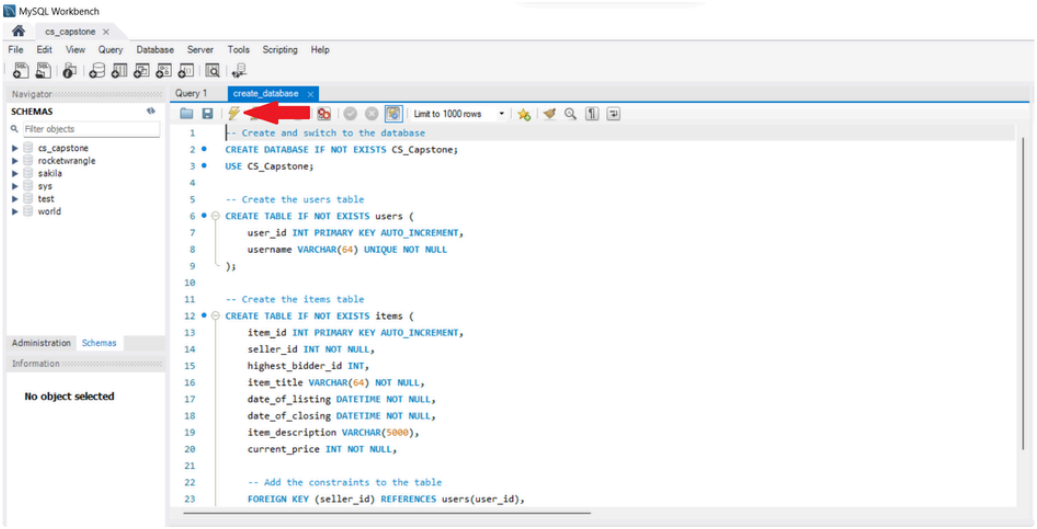
This will open a page to create a new connection. Enter a name for the connection (e.g. "cs_capstone") and then click OK. You should then have the option to connect to this on the homepage in the future. If you click on the connection, it will ask you to enter the password that you entered before. After you enter it, it should open up a new tab.

Creating the Database

On this new tab, you should be able to go to File->Open SQL Script. Here you can open the SQL scripts from GitHub.



When you open these scripts, first pull up the “create_database.sql” file. You can run this by hitting the yellow lightning bolt symbol in the upper left of the script while having no text selected. This script creates the database and the tables.



Next, you should run “database_procedures.sql“ in the same way as before. You may get some warnings, but these are fine to ignore. This script sets up the procedures used to interface with the backend. Finally, if you would like you can populate the database with some example data by running “test_data.sql”.

Macbook Instructions

Prerequisites

- macOS running on your MacBook
- An active internet connection
- Administrative access to your MacBook

Step 1: Download MySQL

- Navigate to the MySQL Downloads Page: Open your preferred web browser and go to the official MySQL downloads page at MySQL :: Download MySQL Community Server .
- Select macOS Version: Scroll down to the "Select Operating System" section, and choose "macOS" from the list of operating systems.
- Choose a Version: Select the version of MySQL that you wish to download. If you're unsure which version to choose, the most recent GA (General Availability) version is recommended for most users.

- Download the Installer: After selecting the macOS version, you will be presented with the option to download the DMG archive. Click the "Download" button for the DMG file. You may be prompted to sign up for an Oracle account or sign in, but this is usually optional, and you can skip it by clicking the "No thanks, just start my download" link.

Step 2: Install MySQL

- Open the Downloaded DMG File: Once the download is complete, locate the DMG file in your "Downloads" folder and double-click it to open the installer.
- Run the MySQL Installer: Within the opened DMG file, you'll find a .pkg file—this is the installer package. Double-click it to start the installation process.
- Follow the Installation Prompts: The installer will guide you through the installation process. You'll need to agree to the license terms, select an install location (the default should be fine for most users), and potentially enter your administrator password to allow the installation.
- Note the Temporary Password: During the installation, a temporary root password will be generated. Be sure to note this password down, as you will need it to secure your MySQL installation later.

Step 3: Configure MySQL

- Open System Preferences: After installation, open System Preferences and look for the MySQL icon. Click it to open the MySQL preferences pane.
- Start MySQL Server: Click the "Start MySQL Server" button to start the MySQL server. You may need to enter your macOS administrator password.
- Secure MySQL Installation: Open the Terminal application (found in Applications > Utilities) and enter the following command to secure your MySQL installation:
- sudo mysql_secure_installation `sudo mysql_secure_installation`
- Follow the prompts to set a root password, remove anonymous users, disallow root login remotely, remove the test database and access to it, and reload privilege tables.

Step 4: Connect to MySQL

- Open Terminal: Open the Terminal application again if it's not already open.
- Connect to MySQL: Enter the following command to connect to your MySQL server as the root user:
- mysql -u root -p
- You'll be prompted to enter the root password you set during the secure installation process.

2 Database Schema

users Table

Column Name	Data Type	Properties	Description
user_id	INT	Primary Key, Auto Increment	A unique integer that identifies users
username	VARCHAR(64)	Unique, Not Null	The username that is used to login and displayed to users

items Table

Column Name	Data Type	Properties	Description
item_id	INT	Primary Key, Auto Increment	A unique integer that identifies items
seller_id	INT	Foreign Key, Not Null	The user_id of the user who is selling the item
highest_bidder_id	INT	Foreign Key	The user_id of the user who currently holds the highest bid on the item
item_title	VARCHAR(64)	Not Null	The title of the item that is displayed to users
date_of_listing	DATETIME	Not Null	The time the item was initially listed
date_of_closing	DATETIME	Not Null	The time that the auction for the item ends
item_description	VARCHAR(5000)		The text description of the item entered by the seller
current_price	INT	Not Null	The current price the item is being listed for
is_canceled	BOOLEAN		Whether or not the seller has cancelled the auction for this item

3 Database Procedures

addUser

```
addUser(VARCHAR(64) username)
```

- Description
  - Adds a new user with the given username. Username must not already be taken.
- Return
  - True on success, False on failure
- Parameters
  - username - The username of the new user

checkIfUserExists

```
checkIfUserExists(VARCHAR(64) username)
```

- Description
  - Checks to see if the given user already exists.
- Return
  - True if user exists, False if they don't or on error
- Parameters
  - username - The username of the user to search for

addItem

```
addItem(VARCHAR(64) item_title, VARCHAR(5000) item_description, VARCHAR(64) seller_username, INT initial_price, D
```

- Description
  - Adds a new item associated with a given seller.
- Return
  - True on success, False on failure
- Parameters
  - item_title - The title of the new item
  - item_description - A description of the item. May be NULL
  - seller_username - The username of the user who is selling the item
  - initial_price - The initial bidding price for the item
  - date_of_listing - The time the item was initially added
  - date_of_closing - The time in which the auction for the item should end

getItemsBySearch

```
findItems(VARCHAR(64) search_text)
```

- Description
  - Finds all items that contain the search text in the seller username, the item description, or the item title.

- Return
  - Table containing the items. For each item, the table contains the item title, the seller username, the current price of the object, and the end date of the auction for the item.
- Parameters
  - search_text - The text to use to search for items

getAllItems

```
getAllItems(DATETIME filter_date)
```

- Description
  - Finds all the items that have not been canceled and that have an end date to their auction after the filter_date
- Return
  - Table containing the items. For each item, the table contains the item title, the seller username, the current price of the object, and the end date of the auction for the item.
- Parameters
  - filter_date - Only items whose close date is after this date are returned.

getItemDetails

```
addUser(INT itemID)
```

- Description
  - Finds the specific details of the item with the given ID
- Return
  - Returns the item id, item title, seller username, current price, date of listing, date of closing, and the item description
- Parameters
  - itemID- The ID of the item to get the details of

placeBid

```
addUser(INT item_id, INT bid_price, VARCHAR(64) bidder_username)
```

- Description
  - Sets the price of an item to the bid price.
- Return
  - Returns the number of rows changed (0 if not updated, 1 otherwise)
- Parameters
  - item_id- The ID of the item to bid on
  - bid_price - The price to set the current price to
  - bidder_username - The username of the user placing the bid

Testing

Front-end testing

Selenium

Back-end testing

Unit Testing

Database testing

Stress Testing

The database stress tests are located in tests/database/stress. The tests are labeled by the functionality that they test, and each functionality has 2 files. There is a .sql file, which contains the SQL code that is executed during the test, and there is a .ps1 file which contains a PowerShell script to execute the test on Windows.

In order to run the tests:

1. Delete the database and set it up from scratch (only do this on a test server, not production). See 1 Setting up Database for more details.
2. Edit the PowerShell script of whatever functionality that you want to test. Enter your username at the “user” argument, your password at the “password” argument, and you can edit the arguments for concurrency and iterations. Concurrency controls how many simultaneous requests are issued to the database, and iterations controls how many times the tests are run and averaged.
3. On the command line, execute the PowerShell script of the test you want to run. The test results should be printed out in the terminal.

Currently, the implemented tests are:

- add_item: This tests checks how long it takes to add an item to the database. For this test to run, at least 1 user must already be added to the database. The test defaults to the user “Alice”
- add_user: This test checks how long it takes to add a new user to the database. The usernames are randomly generated strings.
- get_items: This test checks how long it takes to retrieve all currently listed items from the database.

Known Issues
Depending on machine hardware, some may take much longer than others.
Example:

```
PS C:\Users\hotch\School\cs_capstone\tests\database\stress> .\add_user_test.ps1
mysqlslap: [Warning] Using a password on the command line interface can be insecure.
Benchmark
        Average number of seconds to run all queries: 11.937 seconds
        Minimum number of seconds to run all queries: 2.687 seconds
        Maximum number of seconds to run all queries: 32.032 seconds
        Number of clients running queries: 20
        Average number of queries per client: 1
```

Selenium Front End Testing Documentation

Introduction

Selenium is a powerful tool for automating web browsers, primarily used for testing web applications. This documentation will guide you through setting up and using Selenium for front-end testing of our auctioning web application.

Installation

Before getting started, ensure you have Python installed on your system. Install the Selenium library using pip:

```
pip install selenium
```

You'll also need to download the appropriate WebDriver for your browser. In this documentation, we'll be using ChromeDriver for Google Chrome. You can download ChromeDriver from the official website: ChromeDriver Downloads

WebDriver Location: Ensure that the WebDriver executable file is located in the `test/front-end` directory.

Setting Up Selenium

Importing Libraries

```
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
import time
```

Initialize WebDriver

```
# Specify the WebDriver executable filename
webdriver_filename = './chromedriver.exe'  # the driver in our directory

# Initialize the WebDriver with the specified filename
options = webdriver.ChromeOptions()
options.binary_location = webdriver_filename
driver = webdriver.Chrome(options=options)
```

Testing Front End with Selenium

Step 1: Create User

Before adding an item, make sure to create a user by running `adduser_test.py` script.

```
python3 adduser_test.py
```

This script will open a Chrome window controlled by Selenium to create a user.

Step 2: Add Item

After creating the user, run the `additem_test.py` script to add an item.

```
python3 additem_test.py
```

This script will open a Chrome window controlled by Selenium to add an item. It will also populate the database with the test input provided to test our web application.

Logging in and Creating User

```
# Navigate to the login page and fill in the credentials
driver.get("<http://127.0.0.1:8000/>")  # login page URL
username_input = driver.find_element(By.NAME, 'username')

username_input.send_keys('test_user')

# Submit the login form
login_button = driver.find_element(By.XPATH, "//button[text()='Create account']")
login_button.click()

# Wait for a bit to see the result
time.sleep(2)
```

Open Webpage

```
# Open the webpage containing the forms
driver.get("<<http://127.0.0.1:8000/add_item/>>")  # for add_items page
```

Interacting with Elements

```
# Find elements by different locators
username_input = driver.find_element(By.NAME, 'username')
title_input = driver.find_element(By.NAME, 'title')
end_date_input = driver.find_element(By.NAME, 'end_date')
initial_price_input = driver.find_element(By.NAME, 'initial_price')
description_input = driver.find_element(By.NAME, 'description')

# Interact with elements
username_input.send_keys('jazmin')
title_input.send_keys('Test Item')
end_date_input.send_keys('2024-04-20')
initial_price_input.send_keys('10000')
description_input.send_keys('This is a test description.')
```

Submitting Form

```
# Submit the form
submit_button = driver.find_element(By.XPATH, "//button[text()='Add Item']")
submit_button.click()
```

Handling Wait

```
# Wait for a bit to see the result
time.sleep(2)
```

Closing Browser Window

```
# Close the browser window
driver.quit()
```

Conclusion

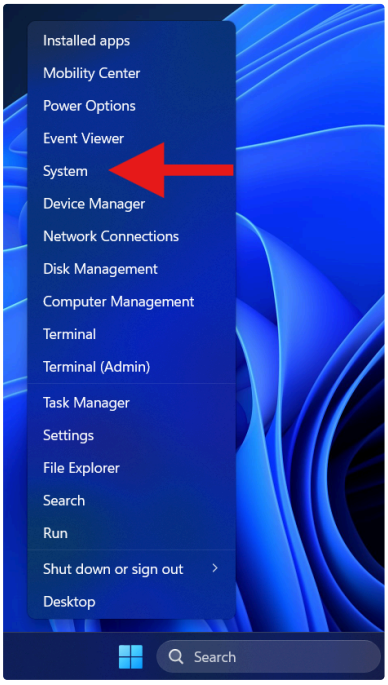
Selenium provides a robust framework for front-end testing, allowing us to automate interactions with our web application and verify its behavior. By following the guidelines outlined in this documentation, you can effectively use Selenium to ensure the quality and reliability of our web application's front end.

Database Stress Testing

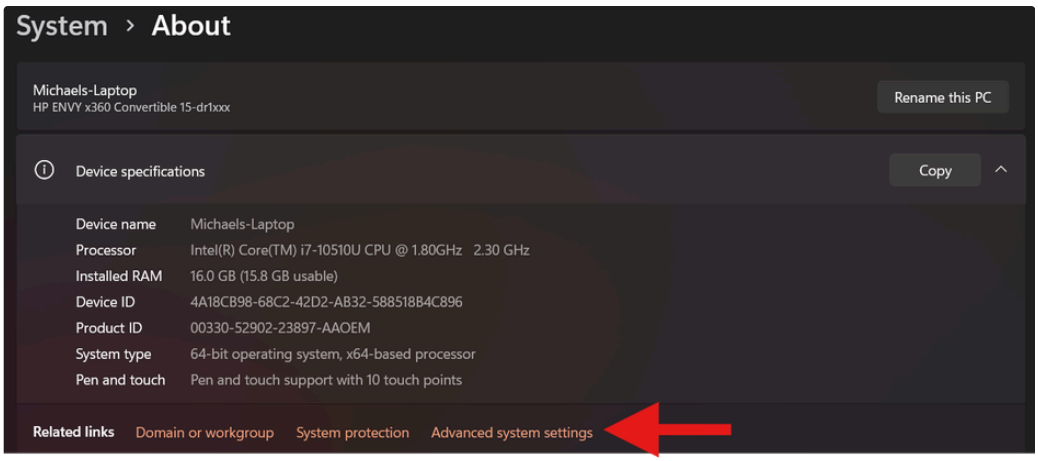
Setup

In order to run the tests, we need to add mysqlslap to the system path.

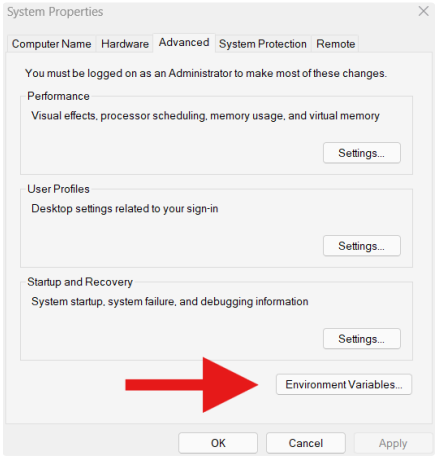
1. Right click the home button and select “System”



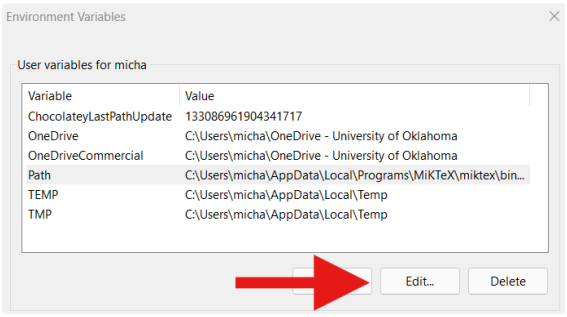
2. Select “Advanced System Settings”



3. Select “Environment Variables…”



4. Select “Path” from list and select “Edit



5. Open your file browser and find the folder which contains mysqlslap.exe. On most systems this will be “C:\Program Files\MySQL\MySQL Server 8.0\bin” or a similar path.
6. Add the above file path to the Path environment variable.
7. If you move to running the tests immediately after this, make sure that your PowerShell path variable is updated. You can do this by restarting your device, or by following instructions here.

Running the Tests

1. It is recommended that before running the tests that you have a fresh instantiation of the database with the test data. Further details on doing this can be found at 1 Setting up Database
2. In “tests/database/stress” you can find .sql and .ps1 files for each type of test. Go to the .ps1 file of the test you intend to run and edit the parameters
   a. Set “user” to be your database username
   b. Set “password” to be your database password
   c. Set “concurrency” to be the number of simultaneous users that you want to test
   d. Set “iterations“ to be the number of times you want the test to be repeated
3. Once the script is edited, navigate to the folder containing the script on PowerShell. Execute the script by typing “.\script_name.ps1“. The script may take significant time to run depending on your computer and the settings for concurrency and iterations

Implemented Tests

Currently the implemented tests are:

1. add_item_test.ps1
    a. This test measures how long it takes to add an item to the database. In order for this test to work, at least 1 user must be added to the database. The default setting for this user is “Alice”
2. add_user_test.ps1
    a. This test measures how long it takes to add a new user to the database. The users added are randomly generated strings
3. get_items_test.ps1
    a. This test measures how long it takes to retrieve all of the items currently available for auction.

Additional Notes

- In some systems you may need to edit the execution policy on PowerShell. See here for details on how to change the policy.
- When creating new tests, mysqlslap will not accept any comments in the .sql files.