Max Stetter

CS/WEB 3200: Web Application Development I

Fall 2021 Final Exam (Practical)

Worth 20%

Monday, November 29–
Monday, December 6, 2021
3:00pm

Details and instructions:

- Do not start the exam prior to the start date indicated above. To receive credit, you must pass off and submit your final solution on or before the end date indicated above.
- During the exam, you may reference any resource published prior to the start of the exam. Any material made available following the start of the exam is prohibited.
- During the exam, do not discuss the exam or its contents with any individual other than the course instructor. Collaboration or discussion with other individuals is strictly prohibited.
- The solution for this exam should be written from scratch. While you may reference other projects as examples, directly copying/pasting nontrivial amounts of existing code from other projects (written by you, or otherwise) is not permitted. Any superfluous code that is nonessential to a working solution and left as part of your final submission *will result in a significant loss of points*.
- You may not use software frameworks or libraries as part of your submitted solution, unless explicitly specified by the exam. Any standard libraries included with a language are allowed.
- Your solution will be graded based on completion. Completion is defined as having met all requirements stated by the exam, having produced a working solution, and having conformed to all necessary conventions and standards (e.g. HTTP, REST).
- To receive credit, pass off your final solution with the instructor and also submit your final solution using the designated private GitHub repository provided by the instructor.

For this exam, you will create a web application to aid Mr. Willy Wonka in managing his highly-coveted Golden Tickets. Your application will support the creation of new tickets, the random designation of winning (golden) tickets, the restriction of repeat ticket submissions, and the retrieval of tickets and their attributes.

Resource

Throughout the exam, refer to the provided resource definition below.

Resource name:

`Ticket`

Resource attributes:

`entrant_name` (a string value)
`entrant_age` (an integer value)
`guest_name` (a string value)
`random_token` (an integer value)

Database

Using SQLite, create a database schema to represent the resource defined above, according to the following specification:

1. In addition to the fields for the resource attributes listed, include a correctly-named integer primary key field.

2. This database schema should be used by the server application (described below) to persist and retrieve data for the resource.

3. Create a Python class (separate from your server implementation) to encapsulate all database logic for your application.

4. Consider using a row factory which returns rows as dictionaries rather than tuples (see the server requirements below).

Server

Using Python, create a server application that implements an API according to the following specification:

1. Implement a RESTful API endpoint which creates a new member within the collection for the ticket resource, given the required data attributes, and returns an appropriate response upon success.

    a. The data attributes given by the client should include `entrant_name`, `entrant_age`, and `guest_name`. The attributes should be represented as `x-www-form-urlencoded` data.

    b. Before creating a new record, a random value should be generated for the `random_token` attribute. The random value should be an integer between 0-6, inclusive. This value should *not* be provided by the client.

2. Implement a RESTful API endpoint which returns an appropriate response that contains a data representation of all members within the collection for the ticket resource.

    a. The data returned should be represented using JSON, as a list of objects, where each object represents one member of the collection and includes all attributes of the resource, by name. Do *not* represent the collection as a list of lists; a list of objects is required.

3. Restrict a client to creating only one ticket per session, using a cookie (Mr. Wonka acknowledges and discretely permits this security vulnerability).

    a. When a ticket is successfully created, the server should return a cookie to the client. The cookie should be named `oompa` and its value should be `loompa`. While the value of the cookie is insignificant, the presence of the cookie is.

    b. If a subsequent request is received to create a ticket, and the `oompa` cookie is present in the request, then a ticket should *not* be created, and a response with status `403 Forbidden` should be returned, with the response body containing a short explanation: `The Oompa Loompas have already received your ticket. Please try again tomorrow.`

    c. Note that, while cookies are necessary in order to implement this functionality, a session store is *not* required and would introduce unnecessary complexity.

Client

Using JavaScript (and HTML and CSS), create a single-page client application that implements a user interface according to the following specification:

1. Implement a form which captures information for a new ticket, and, when submitted, sends its data to the server API using an Ajax request.

    a. The information captured from the form should include `entrant_name`, `entrant_age`, and `guest_name`. These attributes should be sent to the server API represented as `x-www-form-urlencoded` data.

    b. If the server API creates the ticket successfully, clear the form inputs and automatically refresh the list of tickets (see below).

    c. If, however, the server API refuses to create the ticket (see above), display the response text from the server to the user as a simple `alert()` message.

2. Implement a list which displays a collection of tickets requested from the server API using an Ajax request.

    a. The list should accommodate any number of tickets and should include the entrant's name, entrant's age, and guest's name for each ticket.

    b. In addition, each ticket in the list should be styled (visually) differently if it is a winning (golden) ticket. A ticket should be considered a winning ticket if its `random_token` value equals the current day of the week. Consider using the JavaScript code below which returns the current day of the week as an integer value between 0-6 inclusive.

    ```
    var dayOfWeek = new Date().getDay();
    ```

    c. The list should be populated automatically when the page initially loads.

3. Use *minimal* styling to make your application appear clean and professional. Use valid HTML and CSS to structure and style your application.

General

1. Your client and server applications should strictly conform to HTTP and REST standards. Consider headers, methods, paths, status codes, etc. when implementing your requests and responses. With very few exceptions, exactness is expected.

2. If a request is received for a method/path pair which does not match the endpoints specified above, then an appropriate `Not Found` response should be returned by the server, with the response body containing a short explanation: `It seems that this resource has been lost in the chocolate pipes. An Oompa Loompa will be dispatched promptly to recover the artifact.`

3. Be sure to implement CORS as necessary throughout your client and server applications. While preflighted requests should not be required for this application, it is critical that you allow credentials (cookies) to be sent between the client and server.

Check announcement for cookie code.
- postman will not work with the two lines.