ITECH3108 Dynamic Web Programming

Assignment 2 - Hamburger Gossip

"*It's like Yik Yak, but for opinions about Hamburgers*"

Overview

This is an individual assignment in which you will explore the use of data interchange between web servers and browsers, dynamic construction of page content, and API design, using PHP, JavaScript, and JSON.

Important: This assignment specification is generated just for you. Do not distribute this specification.

Timelines and Expectations

Percentage value of task: 20%

Due: Refer to Course Description

Learning Outcomes Assessed

The following course learning outcomes are assessed by completing this assessment:

- **K2.** Contrast the capabilities and limitations of client-side and server-side web code
- **K3.** Detect opportunities for increasing security and privacy of web applications
- **S1.** Develop client/server web applications using client-side and server-side code
- **S2.** Connect to and manipulate a database management system programmatically using server-side code
- **S3.** Design and implement a RESTful web application programming interface (API)
- **S4.** Implement a client-side web application which uses a client-side programming language to access a web API
- **A1.** Design, develop, test, and debug client/server web applications to provided specifications

Assessment Details

Introduction

Hamburger Gossip is a single-page web app that allows users to post their opinions about hamburgers online, and reply to other people's opinions.

Users can post new opinions, which include some review text and, optionally, their name. Opinions submitted without a name are labelled "*anonymous*" and should have distinct formatting.

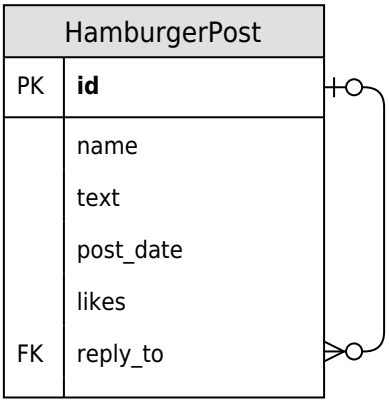
Users can additionally "like" other opinion posts.

Database

A simple database with one table is sufficient to model the database requirements for this assignment, however you may if you wish extend and/or normalize this database.

The database has the following structure:

- HamburgerPost (id, name, text, post_date, likes, *reply_to*)



Each record in the HamburgerPost table represents a single opinion post.

The *post_date* field should be a MySQL `TIMESTAMP` field, and contains the date and time that the post was created. You may achieve this using the `DEFAULT CURRENT_TIMESTAMP` declaration in your `CREATE TABLE` statement.

The *likes* field is an integer value indicating how many people have "liked" the opinion.

The *reply_to* field is a nullable foreign key, which indicates which post, if any, each post is a reply to. A `NULL` *reply_to* indicates that the post is not a reply.

The assignment tasks are closely associated with the lab work of topics 7 to 10. Code and examples from lectures and labs should be a useful guide throughout this assignment. The assignment requires a number of files and a report to be produced. The report should respond to written tasks that are included below

Initial task

Create the above database using your existing SQL skills. You will need at least **eight** opinions about hamburgers. At least **three** posts should be replies.

At least one of the posts should be written by somebody called *user30344274*.

Invent or discover your own data. Cite data sources appropriately in your report, or alternatively include a source field in your database.

Submit your SQL file as part of your assignment.

JSON markup

Mark up the complete data using JSON and save it as a `.json` file. Check that the file is valid JSON and report the method used to validate.

Submit this file as part of your assignment.

Back-end API

Create a RESTful JSON API using PHP, implementing at least the following functionality:

- List all top-level opinion posts *(GET)*
    - Should include id, name, text, likes for posts with a `NULL` *reply_to* value.

    - Use an optional querystring/GET parameter to allow sorting by post_date and likes.
- Retrieve all details for a particular post *(GET)*
    - including replies
- Create a new post *(POST)*
- Increment the *likes* value for an opinion *(POST)*
- Decrement the *likes* value for an opinion *(POST)*

Note that it is ok to unlike a post that you never liked - this may result in the *likes* field being negative.

> *The Like and Unlike methods are not strictly following RESTful practices, as they are not implemented by transferring state. You may choose to use the PUT method instead.*

Follow HATEOAS (Hypermedia as the Engine of Application State) practices. Include a note in your report about how you have followed HATEOAS.

- Under Apache, this step will require configuring a .htaccess file to allow using clean URLs (without a .php extension)

Front-end

Create an HTML/CSS/JS page which uses JavaScript, AJAX, the DOM and your back-end API to:

- Display a global timeline of all top level hamburger opinion posts. Sort the list by date, with the most recent first.
- Allow the user to submit a new top-level post, which will be displayed *immediately* in the review list.
- Display the details of an individual post when clicked, including all direct replies.
- Allow the user to reply to a top-level post. The reply should be displayed immediately.
- Allow the user to "like" a post by clicking on a link or button.
- Periodically (every 5 seconds), update the page with new posts or replies and updated "like" counts. This should **not** remove any text currently being entered in any forms.

The last requirement can be tested by opening your site in multiple browser tabs at the same time

All functionality should be implemented using JavaScript, the DOM and the backend API, without reloading/refreshing the browser page.

Bonus tasks (Completely optional!)

These tasks are 100% optional, and you can receive full marks without attempting or completing them. They are intended to be a challenge if you are interested in such things, and the marks available do not reflect the significant research and effort required to implement them correctly.

Bonus task one: WebSockets

Instead of polling every 5 seconds, use WebSockets to notify browsers when the content they are viewing has changed.

Implement your WebSocket server using either Python or Node.js. You may use an existing WebSocket library.

Bonus task two: Token-based authentication

Implement optional signup/login, also using AJAX techniques, which allows deleting posts and remembering likes.

Use JWT token-based authentication, rather than cookies / PHP sessions.

Bonus task three: GraphQL in Python

Using Python and the Graphene and SQLAlchemy libraries, implement a new API using GraphQL.

Create a variant of your front-end that uses your new GraphQL API instead of REST.

Further details

Please note that there are no marks for aesthetics, so please spend your time appropriately. It is acceptable to use third-party CSS frameworks such as Bootstrap, Skeleton, Bourbon or similar if you wish, as long as you reference appropriately in your report. Third-party JavaScript is not acceptable, nor is code obtained through online Q&A sites such as StackOverflow.

Documentation

Include a written report containing:

- A statement of completion;
- Details of specific assistance you received from people other than your lecturer or tutor, and the names of those assisting;
- References to any third-party CSS frameworks if applicable;
- Anything interesting or cool you'd like to draw to your marker's attention.

Submission

All files should be submitted to Moodle by the due date and time. Check with your tutor as to whether a hard copy is required in addition to the electronic submission.

Marking Criteria/Rubric

Refer to the attached marking guide.

Feedback

Feedback will be supplied through Moodle. Authoritative marks will be published through fdlMarks

Plagiarism

Plagiarism is the presentation of the expressed thought or work of another person as though it is one's own without properly acknowledging that person. You must not allow other students to copy your work and must take care to safeguard against this happening. More information about the plagiarism policy and procedure for the university can be found at http://federation.edu.au/students/learning-and-study/online-help-with/plagiarism.

Marking Guide: Assignment 2

Feature	Criteria	Maximum	Obtained
SQL file	Requirements satisfied	1	
Data Interchange	JSON is accurate, well-structured and syntactically valid	1	
Back-end API	RESTful (resource-based) design	2	
	Uses HATEOAS approach	1	
	List and details (GET)	2	
	New post (POST)	2	
	Like and Unlike	1	
	Sort based on query parameter	1	
Front-end	Global timeline	1	
	Details page, including replies and likes	2	
	Submit a new post	1	
	Submit a reply	1	
	Automatic data reload on interval	2	
Bonus optional challenge tasks	WebSocket update channel	(+1)	
	JWT token-based authentication and admin	(+2)	
	GraphQL API	(+2)	
Documentation	Description of JSON validation approaches	1	
	Description of HATEOAS approach	1	
	Completion of tasks, Assistance statement (lose 1 mark each if not included)	(-2)	
Quality of code (lose marks if criteria not met)	Layout, structure, indentation	(-1)	
	Appropriate and consistent naming scheme	(-1)	
	Appropriate use of comments, including quality and accuracy. Comments do not simply narrate code but illuminate intent and design decisions. No commented-out code.	(-1)	
	Valid HTML5	(-1)	
	Total:	**20**	