Thomas Geraghty

Aston Events CS2410 Coursework Project

Website & Login Details

http://astonevents.tom-geraghty.io/

https://github.com/Thomas-Geraghty/astonevents

Username: admin

Password: brainstorm

Assumptions

Going into this coursework I was under the assumption that we should build an efficient website that was able to connect to a database backend. I decided it was best to use a mixture of HTML, PHP and JavaScript for the project. I used PHP for providing and handling backend information from the MySQL database, whilst I used JS mainly for the dynamic sections of the site like table ordering and filtering. JavaScript's biggest purpose was serving POST data, making GET requests, and then handling the subsequent returned JSON objects. Using AJAX meant I could have a dynamic website, with the ability to update pages on the fly as opposed to PHP having to require a page refresh. This obviously comes with the assumption that all users will be using JavaScript – which is probably likely.

Another assumption I had going into the project was that students would only create accounts when they wanted to be able to host events. Students wouldn't create accounts for the sake of having an account, as all events can be viewed without logging in. This led to a few interesting decisions, especially regarding the like/interested button. Seen as we were not limiting the like button to only logged in users, it is easily spammed. I think this is more an oversight of the actual coursework requirements as opposed to bad design on my end, as it would be easy to stop the button spam if students had to log in to use it. The current design is setup in way that means you cannot like your own event, you can only like other organisers events. This will hopefully help with some of the protentional like button spam.

Description of the System

General Overview

My website was designed with MVC architecture in mind. PHP is the basis for the Model section of the architecture, providing a means of access to the underlying database through classes I made. In the Model package you can see this in classes Database & DBConnection. These classes provide the means for connecting to the database, filtering its data, then returning any requested data to the controller it was called from.

In the controller section we have most of the backend. One class in here is dedicated to adding, viewing, editing and deleting events. This works by intercepting POST or GET requests sent by JavaScript or input forms, then carrying out a task based on the data. This basically acts as a connector between the model and the view.

In the view section we have the main HTML templates. These are filled with content statically by inline PHP, with there DOM edited live by JavaScript. The view contains an interface for generating error pages, the template for the events page, and a collection of structure elements (i.e header, footer) which are added to each page through inline PHP.

Requirements

Students

- Req. S1 is handled by "events.php", 'EventView.php' and the 'eventDisplay.js'. EventView.php gets sent a GET request for a certain data set, then returns a JSON object containing this dataset, to which eventDisplay.js then displays on screen. This is done dynamically without the need for refreshing thanks to AJAX.
- Req. S2 is also handled by these 2 classes, with the category-based filtering function provided in the eventDisplay.js file.
- Req. S3 is met by these also as you can sort the event list by the likeness ranking, thanks to a sorting function written in eventDisplay.js. Same with Req. S4
- Req. S5 is covered by the "eventDisplay.js" and "events.php". 'events.php' is called with a GET that loads onto a certain event, allowing you to share the link to events to people easily. The data for the page is dynamically filled in by eventDisplay.js based on the event ID on aforementioned GET request.
- Req. S6 is handled by "index.php" which has a signup form which sends a POST request to "Signup.php". 'Signup.php' provides methods for controlling the model for adding a user account, hashing and salting the password etc. It validates the entered sign up data, then if its all valid, adds the account to the system and logs you in.

Event Organisers

- Req. E1 is met by the "Auth.php", "Session.php" and "Navbar.php". 'Navbar.php" provides a log-in/log-out button with dropdown box for input depending on the current logged in status. "Auth.php" handles this data, calculating the hash of the password in combination with the salt from the database to determine if the account login is correct. If so, 'Session.php' then modifies the $_SESSION superglobal to keep track of the user ID.
- Req. E2 is fulfilled by "editEvent.js" and the "Events.php" class. editEvents.js provides a form for adding an event, which then passes its data to 'Events.php' which acts as a controller, modifying the model to contain the new event.
- Req. E4 works the same way, except by having the form filled with the current events data, so you can change the elements you want then click submit to change the event.
- Req. E3 is fulfilled by 'eventDisplay.js' and 'EventView.php'. EventView.php provides eventDisplay.js with the events created by the current sessions user. eventDisplay.js then generates a table of these events on screen to view/click on.

Stretch Goals

- Tables can be sorted by clicking on the headers of the table, provided by function in 'eventDisplay.js'.
- Event contains mailto: link with event organisers email, allowing user to email the organiser by just clicking the link.
- Events can have multiple photos (up to 3, as to not clog up page). Photos are uploaded at event creation. Handled by 'Events.php' controller.

Thomas Geraghty

Database Schema