Harman, Stephanie, and Davey

DubDate

Design

Our project is designed in PHP and uses SQL heavily. Pset 7's C$50 Finance gave us the framework to build off of for this project. Our website is very similar to C$50 Finance in that we allow users to create profiles, login, and view their information. We used/made changes to the following from pset7, utilizing both code from solution sets and our own code as the basis: *config.php* (now redirects to our first page), *helpers.php* (the functions *render*, *redirect*, *apologize*, *dump*, and *logout*; additionally here, we added prof_lookup and render_next--but more on those later!), *register.php* (now has the user create a couple name, and add in both partner's names), *login.php*, *index.php* (now the controller for our home page), *header.php* (we rewrote this so that it is now the HTML for the top of our website), *footer.php* (we rewrote this this to be the html for the bottom of our webpage), *config.json*, and the bootstrap as well as the CS50 style sheets. We designed a database called *DubDate* in PHPMyAdmin with the tables *users* (with the columns id, username, hash, viewnum, name1, and name2), *interests* (with the columns id, user_id [foreign key to users table], and interest), *proposals* (with the columns id, user_id [foreign key to users tables], and proposee_id), *matches* (with the columns id, user1_id, and user2_id), and *messages* (with the columns id, match_id, sender_id, and message).

When you first register, you are taken to prof_page. Two buttons take you to two controllers: prof_edit.php and photo_add.php. Prof_edit just inserts a typed interest into the interests table, associating the user id with the interest for later access. Photo_add

stores an uploaded photo on the server. We looked up this code and the error checks online. We modified the code to allow for multiple submissions, in which later photos overwrite what's currently there. Just like that, you've made a profile.

For the home page, the main controller used in our project is *index.php*, solely because it is used to render the homepage, which allows the user to go through a slideshow of other couple's profiles in a similar fashion to Tinder. This controller queries the users table of the database for the "viewnum" of the current user -- the id of the next couple to show, passing viewnum into the function prof_lookup, and that into render. But what does prof_lookup do? This function, defined in helpers, simply takes an id as an argument and then queries the database to find all the relevant profile information related to that id. It outputs an associative array that the view prof_template uses. Many views require prof_template at some point -- we've thus created a social app! But we don't stay in the index controller for long. Yes and No buttons on the homescreen each link to respective controllers. "No" is a simpler controller than "Yes" -- just show the next person via the render_next function. What does render_next do? Also defined in helpers, this function updates the viewnum to the next viable id, apologizing if there are no users left to see. It then renders the profile of that next id. "Yes" controller is more complicated. The fundamental logic of proposals and matches happens here. It tries to delete the "compliment" of the current yes proposal from the proposals table. If it succeeds, it means this person has already proposed to you, and so there is a match, and we insert into the matches database. If it fails, it inserts this proposal into the proposals database. And that is the logic of the slideshow on the home page!

Over in the matches page, the main controller is match.php. This queries the matches table, deriving whom you have matched with. Notice, because of the clever logic of Yes, using a separate table for matches and proposals, all of these queries happen in O(n); or even O(logn) as we index on the ids. That means that as our user base grows, our queries won't become overly expensive. Thanks to Charlie Proctor for inspiring this streamlined backend technique. Before, we just had a proposals table, no matches table, and each time we derived matches in O(n^2), since we had a double for loop. It worked, but the algorithm was complicated and expensive.

Match_page.php, called by match.php controller, displays your matches in the form of buttons -- one to view the profile (match_profile.php) and one to chat with them (match_chat.php). This button clicking is handled by another controller -- match_prof.php.

And that brings us to the chatroom -- some of the coolest logic in the site. We use one more table, messages, to store all the messages, remembering the info of the sender, the match_id, and the message itself. The query statement in messages.php returns an associative array of messages that are then printed iteratively left, right, based on sender_id, in match_chat.php. It looks like a pretty legit IM service!

Logout kills the session; log back in whenever. Logging in, unlike registering, brings you right to the home page--it's assumed you've already done the profile creation you wanted to when you registered. Of course, you can always return to any of the nav pills as desired.

One last note. We experimented with AngularJS in our registration view, and it worked out fine. Using AngularJS allowed us to have dynamic text that we typed into a

textfield appear on a different location on the webpage. This process involves no reloading of the webpage.

We attempted using the FireBase database and its library in our public chatroom. It worked for the most part, but it was too complex for us to change the code enough to allow us to use your session ids as our names. We were only able to input our names into the public chatroom -- we wanted this process to be automated by the server as it would prevent users from making silly, perhaps rude anonymous comments into the public chatroom. For example, any user could use the name of another couple or a string of random characters as their username, and type messages that would just not be pleasant for the community to read. As such, we just discarded our public chatroom in its entirety. (See scrap folder and specifically chatroom.php if interested.)

We hope you enjoyed this walkthrough of DubDate!