

COMP 3512 Assignment #2

Version 2.0, November 21

Overview

This assignment is a group project that expands your first assignment in functionality and scope. It is broken into several milestones with different dates. The milestones are in place to ensure your group is progressing appropriately.

Some of the specific details for some of the milestones and pages will develop over time; that is, this assignment specification is a living document that will grow over the next several weeks.

Composition

You will be working in groups of three or four for this assignment. You will be grouped up as with the first assignment – based on your submission time in the first milestone, but there will be a few additional restrictions:

- I will do my best to make sure you work with someone you did **not** work with on the first assignment, and
- As with the first assignment, I will be asking you for a list of peers you would prefer **not** to work with. I will do my utmost to honour those requests.

When working in a group, each member needs to take responsibility for and complete an appropriate amount of the project work. **Be sure to consult the instructor at least one week prior to any due date if your group is experiencing serious problems in this regard (but by then it's almost too late). If you are having problems two weeks before the due date with a group member, then there is probably a sufficient amount of time to rectify the problems!**

I feel foolish saying this in a third-year university course, but it is your responsibility to read all the assignment instructions thoroughly and carefully. If you are unclear about something, ask me. But before you do, read the material in question again!

Functionality

Your second assignment will replicate some of the functionality from assignment 1, but is not a single-page application; instead multiple pages in PHP are needed. Some of these pages also use JavaScript but others don't.

Visual Design: Your pages will be marked on a desktop computer, but your site needs to be usable at mobile sizes. That is, I will be mainly testing and evaluating this assignment using a browser with a large browser width, say 1200 pixels wide. You should still use media queries as I will also test your pages with a smaller-width browser size to ensure it looks reasonable at the mobile sizes.

Header: Each page will have a header at the top that will contain some type of logo and a navigation menu. For smaller browser sizes, you will instead need a "hamburger" menu, that is, a responsive navigation bar. There are many examples online of the necessary CSS and JavaScript for this to work. If you make use of CSS+JavaScript you find online, please be sure to document this in the About page. The header/hamburger menu should have the following links/options:

- Home
- About
- Browse/Search
- Countries
- Cities
- Profile. Should only be available once user is logged in.
- Favorites. Should only be available once user is logged in.
- Login/Logout. If user is not logged in, then the option should be Login; if user is already logged in, then option should be Logout.
- Sign Up. Should only be available if the user isn't logged in.

Note: the sketches in this assignment specification are meant to show functionality, not design. Here I've shown content as boxes, but you could do them as rectangles, circles, icons, simple links, etc. Make your pages look nicer than these sketches!

Home (Not Logged In): The main page for the assignment. This file **must** be named `index.php`. This must have the functionality shown in the following sketches. The first shows the home page when user hasn't logged in; the second show the home page after a user has logged in.

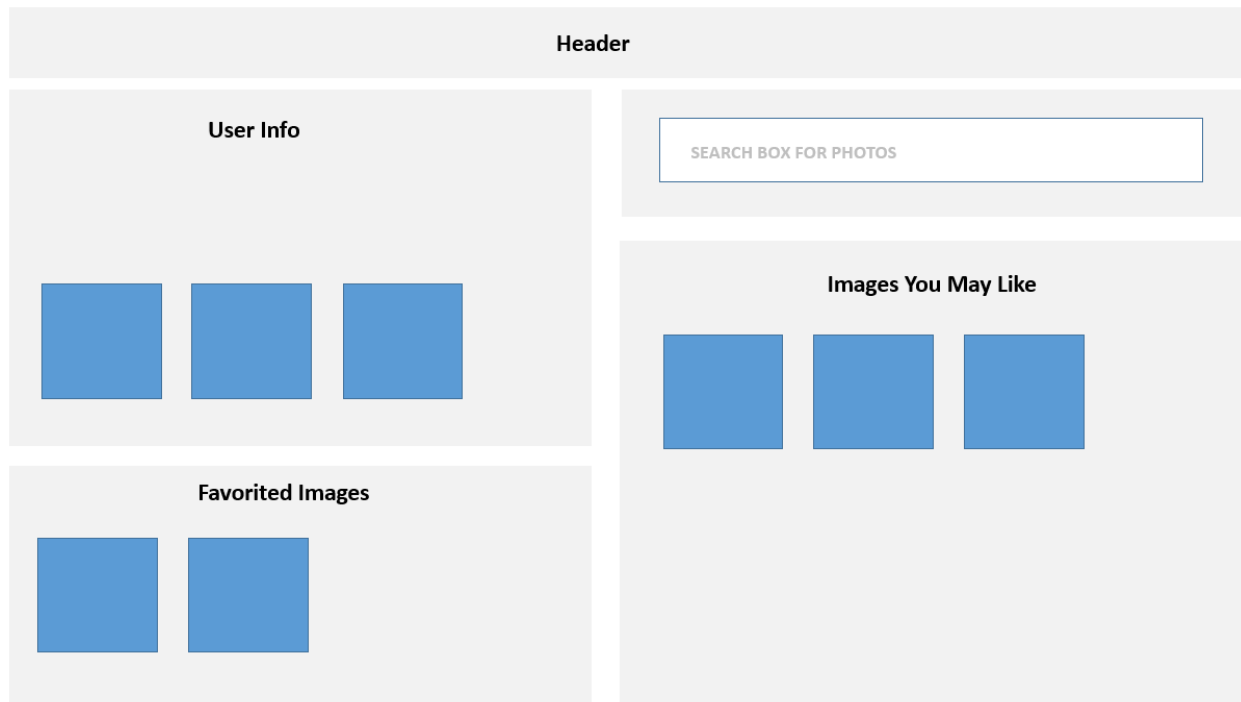
A hero image is a large banner image: you can find attractive very large images on unsplash.com. The search box should take the user to the browse/search page (i.e., it should show results as if the user performed a photo title filter action).

The sketch shows a web page layout for a home page when a user is not logged in. It consists of a light gray header bar at the top with the text "Header" centered. Below the header is a large, light gray rectangular area representing the hero image, with the text "Hero Image" centered. In the center of the hero image area, there are two rounded rectangular buttons: "LOGIN" on the left and "JOIN" on the right. Below these buttons is a white rectangular search box with a thin blue border, containing the placeholder text "SEARCH BOX FOR PHOTOS".

Home (Logged In): Display the user info (first name, last name, city, country). Also display any photos that belong to them, and of their favorited photos (if none yet, be able to handle that).

Also display maybe 10-12 recommended photos the user “may” like. How to do this? Ideally, if you had several additional months to work on this assignment, you would create a recommendation engine using some type of Machine Learning algorithm. But given the time constraints, simply show: images with the same country and city to those they own or have favorited, or for favorited images by the same user. It’s possible for a new user who hasn’t uploaded or favorited any images, that you don’t have anything to base this Like algorithm on. In that case, show the most recent (i.e., the last) 10-12 photos in the imagedetails table. If your Like algorithm has fewer than 10-12 images, fill it up with the most recent images.

Each of these photo images should be links to the `single-photo.php` page with the image id in the query string.



Country Page: will display a sorted list of all countries as well as provide details about a country. This file must be named `single-country.php`. This list must be populated in JavaScript from an API you created in your third milestone named `api-countries.php`. Each country in this list should be a hyperlink, with the URL being the same `single-country.php` page, but with a querystring parameter indicating which country to display (e.g., `single-country.php?iso=ca`).

To improve the performance of your assignment, you must store the list of countries in local storage after you fetch it. Your page should thus check if country data is already saved in local storage: if it is then use it, otherwise fetch it and store it in local storage.

The country filters have the exact same requirements as the first assignment, and use JavaScript to perform the filtering.

When the user clicks on a country (that is, the page is requested with a querystring parameter), display the details of the country (name, area, population, capital name, currency, domain, languages, neighbouring countries, and description), a list of photos for that country, and a list of cities in that country. Each photo image should be a link to `single-photo.php` with the appropriate query string parameter. Each city should be a link to `single-city.php` with the appropriate query string parameter.



City Page: This page must be named `single-city.php`. It will display information for a single city. Which city? The city whose id was passed as a query string to the page.

This page will allow the user to view the following city fields: name, population, elevation, and timezone.

You will also have to display a static country map that includes a marker for the city (you will use the Google Maps Static API). Like on the previous page, the city photos will be links to `single-photo.php`.

Notice that this page has the same country list+filter as the country page. Duplicate the functionality but don't duplicate the code+markup!



Photo Page: This page should be named `single-photo.php`. It should display the single photo indicated by the passed `querystring` parameter. It should have the same functionality as the Single Photo View in the first assignment (tabs, map, hover photo, but no speaking or close buttons). Country and city fields should be appropriate links to `single-country.php` and `single-city.php`.

The Add to Favorites button will add the photo to a session-based list and provide feedback that it has been added. This can be achieved in two ways:

- Redirecting to a PHP page that adds the photo to the session list, and then redirects back to this page. This is easiest but old fashioned and inefficient; it results in two redirects (two request+response cycles).
- Do it asynchronously using JavaScript fetch. The fetched page would return simple JSON that indicates whether add to favorites was successful or not. This is a better approach and will get a slightly better mark.

This add to favorites button will only be visible if the user is logged in. If a photo is already favorited, you shouldn't be able to add it again. Ideally, you do this by hiding/disabling the button. Alternately, you display a message telling the user it's already been favorited.

Header

Photo

Photo Title

User Name

Country, City

Description

Details

Map

Description



Add to Favorites

Favorites Page: will display list of logged-in user's favorited photos (implemented via PHP sessions). If none yet, be able to handle that with a message. This page must be named `favorites.php`. The user should be able to remove photos singly or all at once from this list. This page will be entirely in PHP. The list should display a small version of the photo and its name. They both should be links to the appropriate single photo page.

About Page: Provide brief description for the site, by mentioning class name, university, professor name, semester+year, and technologies used. Also display the names and github repos (as links) for each person in the group. Add a link to the main assignment github repo. Every year, for unknown reasons, students lose easy marks by not fulfilling these simple requirements.

Just like in assignment 1, be sure to provide credit for any external/online CSS/JavaScript/PHP you have made use of in this assignment.

Browse/Search Page: display a list of photos with the ability to filter the list. Sort the list by title. In the filters, allow the user to select a city or country (using <select> lists, but only include cities and countries that have photos). Also provide a text box to allow the user to search for photos that contain the entered text anywhere in the title. Finally, the search/filtering will need some type of submit/filter button. This page doesn't use JavaScript but PHP which will require the programmatic construction of a SQL WHERE using this form data. To simplify your coding, assume that each filter is mutually exclusive (no ANDs or ORs needed). Provide a way to reset/remove the filters.

Header	
Photo Filters	Browse/Search Results
	<div>Title<div>ViewAdd to Favorites</div></div>
	<div>Title<div>ViewAdd to Favorites</div></div>

Login Page: will display a login screen with email and password fields, plus a login button and a link or button for signing up if the user doesn't have an account yet. Some of the information below about hashing algorithms will be explained in class when we cover security.

Your database has a table named Users. It contains the following fields: id, firstname, lastname, city, country, email, password, salt, password_sha256.

The actual password for each user is **mypassword**, but that's not what is stored in the database. Instead, the actual password has been subjected to a bcrypt hash function with a cost value of 12. The resulting digest from that hash function is what has been stored in the password field. That means to check for a correct login, you will have to perform something equivalent to the following:

```
$digest = password_hash( $_POST['pass'], PASSWORD_BCRYPT, ['cost' => 12] );
if ($digest == $password_field_from_database_table && emails also match) {
    // we have a match, log the user in
}
```

For successful matches, you will need to preserve in PHP session state the log-in status of the user and the user's id. As well, after logging in, redirect to the **Home** page.

Registration/Signup Page: will display a registration form. Perform the following client-side validation checks using JavaScript: first name, last name, city, and country can't be blank, email must be in proper format (use regular expressions), and the two passwords must match and be at least 8 characters long. Be sure to display any error messages nicely.

Once the user clicks Sign Up/Register button, you will have to check to ensure that the email doesn't already exist in the users table. If it does, return to form (with the user's data still there) and display appropriate error message. If email is new, then add new record to the Users table, log them in, and redirect to **Home** page.

You will **not** store the password as plain text in the database. Instead, you will save the bcrypt hash with a cost value of 12.

A hand-drawn sketch of a web form titled "SIGN UP". The form is enclosed in a rounded rectangle. At the top left is a "Logo" placeholder, and at the top right is a hamburger menu icon. Below the title, there are seven input fields stacked vertically, each with a label to its left: "First Name", "Last Name", "City", "Country", "email", "password", and "confirm password". A "SIGN UP" button is located at the bottom of the form. To the right of the form, there are three handwritten annotations with arrows pointing to the input fields: a bracket spanning the first four fields with the text "can't be left blank", an arrow pointing to the "email" field with the text "proper format", and a bracket spanning the "password" and "confirm password" fields with the text "must match".

Logout. If the user is logged in, then modify your session state information so your program knows a user is no longer logged in. After logging out, redirect to **Home** page.