Assignment 1: A Small PHP MVC Web Application

Warning:

Start this assignment immediately once you receive the statement. Students new to programming PHP MVC Web applications will succeed only if they work for the entire time allocated for this assignment. Slow and steady wins the race.

Purpose

Before we build a complete database-driven Web application, we must understand how MVC (Model View Controller) Web applications work, how controllers are invoked by routing, how controllers call views and how views can supply correct URLs (Uniform Resource Locator) for navigation through your browser.

Learning objectives

In this assignment, you will build a PHP MVC Web application which will not interact with databases. However, we will still read from and store data in files. You will

- Understand application bootstrapping
- Understand URLs and the Routing mechanism
- Understand the role of controllers and their methods
- Understand the role of views
- Understand the role of models for data access
- Build correct controllers to invoke views
- Build correct views with correct HTML (HyperText Markup Language), forms, and hyperlinks
- Build controller methods that receive form data
- Open, lock, read, write, and close text files
- Parse and generate JSON strings with PHP built-in functions
- Understand and use associative arrays
- Understand and use GitHub repositories
- Reading instructions

Requirements

1- You will work in teams of 2 and use GitHub to store your project code from the beginning. Uploading to GitHub once the project is complete is not an acceptable work method.
2- Follow the instructions on https://getbootstrap.com/ to include the CSS and JavaScript Bootstrap resources in all your views using a CDN (Content Delivery Network).
3- Each page will have a full navigation menu to all pages.
4- Implement all views, controller methods, and routes to support functionality explained in the following section (Instructions).

5- Use the code framework that we built together in class tutorials. Source this code from https://github.com/paquettm/eComH24S1.
6- BONUS: You will use the `getJSON` function from the jQuery library to get, parse and process JSON data from the /Counter/index method defined below. You must include the jQuery library. Follow the instructions on code.jquery.com for the most recent version.

Hint: placing JavaScript/CSS code in the header or footer sub views will make it available in all pages.

Instructions

In teams of 2, build a Web application with the views called through matching controller methods invoked by requests to the following URLs:

1- localhost/Main/index -> site landing view
2- Localhost/Main/about_us -> the about us view
3- localhost/Contact/index -> the contact us form view
4- localhost/Contact/read -> the listing of all messages
5- localhost/Count/index -> a page load counter

All views will be written in PHP files that you will organize in the views subfolder under the app folder as follows:

1- app/views/Main/index.php -> site landing page view
2- App/views/Main/about_us.php -> the about us page view
3- app/views/Contact/index.php -> the contact us form view
4- app/views/Contact/read.php -> the message listing view
5- Contrary to the other controller methods, the page load counter will simply return JSON-encoded data through one echo command

Hint: Use a naming convention for your controllers and their methods such that you find it easy to make the correspondence between URL, controllers and methods, and view names.

BONUS: Page load counter

You will not be penalized if you don't implement this, however, if you do, you get the gratification of knowing you can implement a simple AJAX/PHP page counter.

Read the PHP documentation for the following functions:

- `fopen`: to open files in 'r' read mode and 'w' write mode
- `fread`: to read an open file up to a specified number of bytes
- `fwrite`: to write to an open file, the contents of a string
- `flock`: to lock a file for exclusive writing
- `fclose`: to close an open file
- `json_decode`: to read a JSON string and output an object
- `json_encode`: to read an object and output a JSON string

Counter Model

Create the Counter model class in the app\models namespace.

The Counter model class has one property, `count`, and 4 functions: `__construct`, `increment`, `write`, and `__toString`.

The `_construct` method has the following algorithm:

1. If the /resources/counter.txt file exists (use the `file_exists` function)
    a. Open it for reading (use `fopen`);
    b. Lock the file (use `flock`);
    c. read the file into the `$count` variable;
    d. Close the file (use `fclose`);
2. Else
    a. Set the `$count` variable to `'{"count":0}';`
3. Decode the JSON in `$count` and copy the resulting object's `count` property to `this` object's `count` property.

The `increment` method adds 1 to `this` object's `count` property.

The `write` method has the following algorithm:

1. `json_encode this` object into `$count`;
2. Open the counter.txt file for writing (use `fopen`);
3. Lock the file for writing (use `flock`);
4. Overwrite the file contents with `$count` (use `fwrite`).
5. Close the file (use `fclose`)

The `__toString` method returns the json-encoded value of `this` object.

Count Controller

The Count controller class will have an index method which will perform the following operations:

1. Make a new Counter model object
2. Call increment on the Counter object
3. Call write the Counter object
4. Echo the Counter object

Each of the following views will call the Page load counter through an AJAX request and display the page load count at the bottom of the page, to the right of the page.

Helpful hint: Since the views are called into objects that are in Controller subclasses, it is valid to use the view method within views to include sub views, such as an Ajax-based page counter, or a navigation menu.

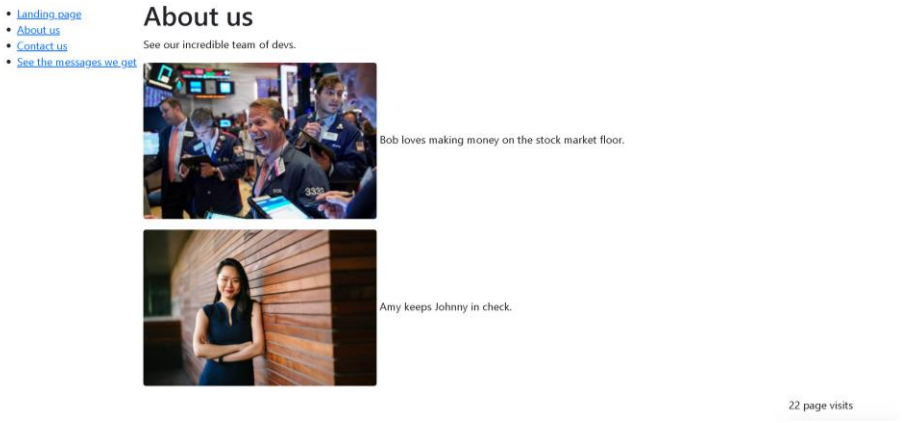
Views

The application must have views as follows:

Site landing

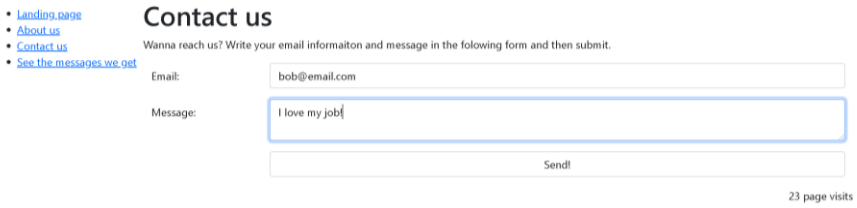
Introduce the Web Application that you propose to develop this semester for your term project.

About Us page



Introduce your team – put your pictures in images, in figures, with captions.

Contact Us



Create a view that has a form to send a message to the company. The form will have labels and fields for the email (use an HTML input of type email) and for the message (use an HTML textarea element).

Message Model

Create the Message model class in the app\models namespace.

The Message model class has three properties: `name`, `email`, and `IP`, and 2 functions: `read` and `write`.

The `read` method opens the /resources/messages.txt file with the `file()` function and returns the result.

The `write` method has the following algorithm (like an earlier method):

1. `json_encode` this object into `$message`;
2. Open the /resources/messages.txt file for appending (use `fopen`);
3. Lock the file for writing (use `flock`);
4. write contents of `$message` and concatenate with a `\n` (use `fwrite`).
5. Close the file handler (use `fclose`)

Contact Controller

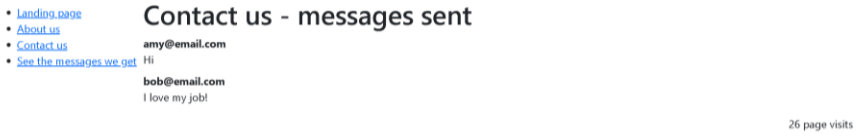
To receive the data from the form discussed earlier, your program must extract the right data from $_POST and use it to set a new Message model object's properties.

To the same object, your program must set the IP address obtained from accessing `$_SERVER['REMOTE_ADDR']`.

Then, your program must call the write method of that message model object.

Once the writing is complete, the program must redirect to the `localhost/Contact/read` URL with the following instruction: `header('location:/Contact/read');`

Read Message log page



Read and display the messages by making a new message model object and using its read method. Call the view and pass the data obtained from the call to `read()`. In the view, use a foreach loop and output each element of the array.

In this example, messages.txt contains the following:

```
{"email":"amy@email.com","message":"Hi","IP":"143.119.66.25"}
{"email":"bob@email.com","message":"I love my job!",
"IP":"141.78.73.72"}
```

Submission instructions

Your completed assignment code must be stored in a public or private GitHub repository. You must invite your teacher(s) as a contributor(s) to this repository (If Ronald teaches your lab section, invite Ronald and Michel as contributors).

Do not submit your code through Omnivox. Instead, submit a text file containing

- your name and
- your teammate's name along with
- the URL of your GitHub repository.

You will demonstrate your assignment to the teacher during the lab class following the submission deadline.