Ad Lister Project

Congratulations! You've just gotten your first web development job. Ad Lister LLC had originally hired an Elbonian subcontractor to build their Craigslist clone but the project was never finished. Now they've hired you and another junior developer or two (your classmates) to come in and finish the project.

You'll need all of the topics you've learned so far in class including HTML, CSS, JavaScript, and PHP. You will be working on this project in teams of two to three.

Here are the primary goals of the project:

1. Gain experience in building a well-designed in PHP site without the use of a framework.
2. Learn how to use Git in a team environment.
3. Test your resourcefulness in solving problems.

Project Tasks Outline

1. Build your site layout from mockups provided
2. Generate and render PHP mock data
3. Design your database
4. Implement Models
5. Replace mock data with live data
6. Implement Ad creation w/ image upload
7. Implement User Login
8. Protect your sensitive configuration
9. Implement a Front Controller

After each step, you will need to provide a demo to an instructor so they can approve the work and review your code.

Step 1: Clone the site so far.

It looks like the Elbonians have already set up most of the files you'll need. A few of them may even have notes about what belonged where.

1. Start by cloning the repository onto your local machine. Explore the files that are already there and look at what still needs to be completed.

```
database/
    migration.php
    seeder.php

models/
    Ad.php
    BaseModel.php
    User.php

public/
    css/
        main.css       // site styles
    img/
        logo.png
    js/
        main.js        // site javascript

    ads.create.php     // ad creation form
    ads.edit.php       // ad editing form
    ads.index.php      // listing of all ads
    ads.show.php       // view of individual ad

    auth.login.php     // login form
    auth.logout.php    // logout action

    index.php          // marketing homepage
```

```
    users.create.php   // user signup
    users.edit.php     // user editing form
    users.show.php     // user profile

utils/
    Auth.php
    Input.php
    Logger.php

views/
    partials/
        header.php
        footer.php
        navbar.php

bootstrap.php          // site initialization
```

1. Next, build an HTML template for your site according to the wireframes provided. Once complete, break down the template into a header, footer, navbar, etc and populate the PHP files in the `views/partials` directory accordingly.
2. Then, begin adding content to the `public/index.php` file and include your partial views to make your PHP site look like your HTML mockup. Make sure to put any site resources (JS, images, and CSS) in the appropriate locations.
3. Continue building out your site with HTML so that all of the PHP files in the public directory have content.
4. Add links between the pages so that you can browse the site and it feels like it is complete. All the data for each page should just be hard-coded HTML. For example, in the `ads.index.php` file, you should have a listing of 3-5 hard-coded sample ads. The `ads.show.php` page will have hard-coded data showing one single ad, etc.

When you think you have completed all the above tasks, please call over an instructor and give them a demonstration before continuing.

Step 2: Design your database

Good news! The Elbonians built a database migration for your users. Starting on paper, think about how you'll map out your database. Since you have a user table already, think about how you'll store your items. How do those two tables relate?

Make a UML-style map of the class on paper before you begin.

Ultimately, the resulting code returned from your model will look something like this:

```
$ads = [
    [
        'id'          => 100,
        'user_id'     => 100,
        'name'        => 'SNES',
        'description' => 'Plays like new! Includes mario kart.',
        'price'       => 150.00,
        'image_url'   => '/img/uploads/100.png'
    ],
    // ...
];
```

Create the database tables and also enter some sample data into the database. Fill in your `database/seeder.php`; using SQL statements and `PDO::exec()` it should first drop all your database's tables if they exist, and then recreate them.
Your `database/seeder.php` file should truncate the same tables and then use prepared

statements to insert test data for your application. Be sure to create enough data to effectively test all the relations and models in your application.

Step 3: Implement your models

Now that we have some data in the database, it is time to make that accessible via PHP. To do so, we will use the Model class provided.

While the Elbonians left you a model for the Users, you will need at least one more for the Ad. Your Ad model will be a class that inherits from the `BaseModel` that provides much of the reusable functionality.

When complete, test the models by using the PHP interactive shell or by writing a small test PHP file.

Step 4: Implement Ad creation w/ image upload

So far, we have a site that appears to work. We can browse ads and look at individual ads and users. Now, it is time to allow users to create ads so we will have more than just our seeded data.

The Elbonians started to build the backend of an image uploader in the helper_functions.php file under the utils directory. Take a look and see if you can figure out how to use it. You might find the documentation on the $_FILES superglobal useful.

Step 5: Implement User Login

Now that we have introduced editing features, we really need to have a login system to make sure that users only have access to what they should. Here is what we need:

1. Only authenticated users can create ads.
2. Ads can only be edited by the user that created them.

You should be able to use your `Auth` helper class along with the provided User model to help with these tasks.

Step 6: Protect your sensitive configuration

Now that we have incorporated database access and user login, you may have noticed that we have some important things like usernames and passwords stored in our code. If we check these things into a public repository on GitHub then they are there for all the world to see, which certainly is not ideal.

We'll use an *env* file to manage this risk. In the root directory, you may have noticed a gitignore file that tells git to ignore a file titled "*.env.php"*. This file will contain our database and configuration information.

Thankfully, there is already a file called `.env.template.php`, also in the site root, that contains placeholder information. This is a common practice. Fill in the appropriate information ad rename to ".env.php". Note the leading period.

Notice that this PHP file is essentially just an array that gets returned. There is a neat trick that will allow you to use the contents of this file to initialize a variable.

So, we can use a PHP `include` statement to read our `.env.php` file and store the contents in a variable of our choosing.

We should put this code in a place where it will always be accessible. A great place to do this would be the `bootstrap.php` file in the sites root folder. Then, the `bootstrap.php` file should get required in all other PHP files. You can also put other important initialization code there and you will always know that it got done.

For example, if you wanted to make logging available across your application, you may initialize your site logger there.
Now, you will have access to variables such as your database name by doing the following:

```
echo $_ENV['DB_NAME'];
```

Try replacing all your sensitive configuration currently in your code with references to the entries in your `$_ENV` array. Since we ignored the `.env.php` file using `.gitignore`, we don't have to worry about those appearing in our repository and now we are much better off.

Step 10: Implement a Front Controller

On the web, you will rarely see URLs like `http://adlister.dev/ads.index.php`. Instead, you are more likely to see URLs like `http://adlister.dev/ads`. These pretty-URLs look much nicer. We can implement these in our project using a Front Controller.
The Front Controller will be our `index.php` file. It will look at each request that comes in, and based on the path, it will require or include other PHP files. Here is an example:

```
<?php


switch ($_SERVER['REQUEST_URI']) {
    case '/ads':
        include 'ads/index.php';
        break;
    case '/ads/show':
        include 'ads/show.php';
```

```
        break;
    default:
        include 'home.php';
        break;
}
```

In the simple example above, you can get the basic idea of what is going on. We use the superglobal `$_SERVER` array to determine what the request path is. We then conditionally load one of our other PHP files based on that value.

At this point, it would be a good thing to refactor your file structure so that `ads.index.php` in the public directory becomes `ads/index.php` within the views directory. Follow this same pattern for all the other files as well.

Where to go from here

There are so many places to go. If you have time, try some of these but feel free to come up with your own ideas too!

1. Allow a user to edit their profile, update their password, etc.
2. Allow users to delete ads that they have created.
3. Implement a user dashboard view where they will have quick access to all their ads.
4. Use composer to pull in a third party libraries for things like email (SwiftMailer), generating fake data (Faker), or any other feature you may want.
5. Send emails on important events like user signup.
6. Allow users to mark their "favorite" ads and make these show up on the dashboard.
7. Implement a "flagging" feature where users can mark ads as inappropriate.
8. Implement an admin user that can moderate ads and remove them.
9. Implement a user feedback system.
10. Implement a forgot-password feature.
11. Once nice feature for a user of the Ad Lister app would be to create some ads in a CSV and then do a bulk upload. Implement this feature, but don't worry about the image upload.
12. Be creative, and most of all have fun!