PROGRAMMING NOTES

NERDERY ASSESSMENT TEST – PHP

Asif R Choudhury
choudhury.asif@gmail.com
262-909-4828

OBJECTIVES

I approached this project hoping to accomplish the following objectives:

- Demonstrate a working knowledge of PHP
- Provide insight into my coding style and habits
- Create readable and easily extensible code
- Illustrate familiarity and flexibility with MVC frameworks/patterns

PROGRAMMATIC DECISIONS

Some of the decisions I made while programming are listed below, accompanied by explanations of how it resembles or deviates from my usual coding choices.

CODE STRUCTURE

FRAMEWORK PREFERENCE

When starting from scratch, I personally prefer not to use existing frameworks or CMSs. Using an established framework goes against my objective of creating readable and easily extensible code in most cases that I encounter, and specifically for this project. By starting off with a framework, it would be limiting extensibility immediately in two ways: the extensibility would be constrained to the built-in features of that framework; and the number of people who can contribute to the project would then be constrained to those familiar with the specific framework. Additionally, adding hundreds of files to a project where only a handful add functionality makes everything less readable and less comprehensible.

That being said, I understand the necessity and appropriateness of using frameworks for larger projects and when working in teams. I used CakePHP when working for my school's web development department, and am prepared to learn and adopt whatever frameworks are used at The Nerdery.

USUAL STRUCTURE

In general, when viewed within an MVC paradigm, the vast majority of my controller logic is in JavaScript. As my backend code is primarily model logic, I usually do not strongly distinguish controller and model classes within PHP, but rather organize the logic into pertinent objects, using a single controller file often paired with mod_rewrite to map requests to those objects. I have found that creating a JSON API-like backend from the start allows for easy extensibility to the front end, such as creating a mobile version or a PhoneGap app.

STRUCTURE FOR THIS PROJECT

I structured this project in a way that can be fully functional and extended as is, or be migrated to either CodeIgniter or CakePHP and fully functional within ten minutes. I believe that this makes my code more extensible in that you have the freedom to choose what framework is best suited for how you wish to extend the application.

There are three things I hoped to accomplish in approaching this project with this structure. First, I hoped to demonstrate that I know how to code in PHP, and that I am not just muddling my way through PHP using a framework as a crutch. Second, I hoped to illustrate that I am comfortable with the MVC pattern, and that I have an understanding of MVC frameworks. And lastly, I hoped to provide insight into my coding habits: primarily my tendency to consider how I might leave both the developer and the end-user with as many options as possible.

REUSED UI

Due to time constraints, and because this is not a frontend challenge, I chose to reuse the frontend code from a recent C# .NET challenge. The CSS was untouched; I updated one line in the footer of the HTML to reflect that this is a PHP challenge; and I introduced the xbox.url object in /scripts/xbox.js to better handle changes in frameworks and directory structures. When I initially created the UI, I used JQuery UI with its default theme and razorjack's Quicksand library in order to quickly get things up and running. I usually spend more time with styles and theming, and have a personal fork of quicksand that gets along with my code better.

DEVIATION FROM NAT SPECIFICATIONS

SOAP SERVICE

My initial call to the SOAP service described in the PHP NAT specification failed, so I thought that I might need a different API key. Since I had a functional API key for the .NET SOAP service, and it worked fine when I called it from PHP, I chose to move forward with that. Further down the road, I realized that the API key was valid, and that the PHP SOAP service merely expected string format for the request where I was using arrays. Due to time constraints, I have opted not to rewrite the code, as my application still accesses the same API and the functionality has not been compromised in any way.

UI: PURCHASE ON SAME PAGE

A minor deviation from the specifications is that I allow the end-user to purchase games on the same page (accessed by the menu on the right) rather than through a separate page as indicated by the specs. I made this decision when initially developing the UI for another challenge, and decided to keep it the same in the interest of time.

SORTING GAMES: REDUNDANCY AND UI DEVIATION

The .NET challenge specifications only required that the games be sorted by votes, but I had created both sort by vote/alphabetically and search functions within the UI, as I saw them as convenient features that would not take too much effort to implement. The extra effort paid dividends by giving me the ability to sort owned games alphabetically right away per the PHP NAT specifications. I recognize that requesting that the games be sorted in different ways is to assess my ability to translate the specifications into PHP, so I implemented this in PHP as well. Hence there is a certain level of redundancy in game sorting between the JavaScript and the PHP that usually would not be present in my code. Additionally, while this is not an explicit deviation from the PHP NAT specifications, I just thought I would note that the UI does not automatically toggle how it is sorted based on which games are displayed, but rather lets the end-user choose when and how to sort.

APPLICATION CONFIGURATION

FILES

The `master` branch contains the code that I am officially submitting for review. The `CodeIgniter` and `CakePHP` branches are there to illustrate the ease of migration, which can be observed in the appendix or by reviewing the commits on BitBucket. The following files from the master branch are the only ones in which I wrote code:

- /index.php
- /Game.php
- /GamesController.php
- /User.php
- /UsersController.php
- /fakeMVCController.php
- /scripts/docReady.js
- /scripts/xbox.js
- /styles/global.css

GIT DOWNLOAD (VIA BASH)

Clone Git:

```
git clone https://asifrc@bitbucket.org/asifrc/nerdphp.git
```

Optionally, fetch CakePHP or CodeIgniter Branches:

```
git checkout –b CakePHP origin/CakePHP
git checkout –b CodeIgniter origin/CodeIgniter
```

ZIP DOWNLOAD

Go to https://bitbucket.org/asifrc/nerdphp/downloads
Click on Branches, and download `master`
Optionally, download `CakePHP` or `CodeIgniter` branches

Appendix: Migration Guide

CodeIgniter

Step 1
Paste all CodeIgniter files EXCEPT /index.php

Step 2
Move /index.php to /application/views/index.php

Step 3
Paste CodeIgniter's /index.php into /

Step 4
Edit /scripts/xbox.js
```
rewrite: true,
games: "index.php/GamesController",
user: "index.php/UsersController",
```

Step 5
Move /User.php to /application/models/User.php

Step 6
Edit User.php: `class User extends CI_Model`

Step 7
Move /Game.php to /application/models/Game.php

Step 8
Edit Game.php: `class Game extends CI_Model`
Uncomment call to `parent::_construct()`

Step 9
Move /UsersController.php to /application/controllers/UsersController.php

Step 10
Edit UsersController.php
`class UsersController extends CI_Controller`

Step 11
Move /GamesController.php to /application/controllers/GamesController.php

Step 12
Edit GamesController.php
`class GamesController extends CI_Controller`

Step 13
Final Step: Edit /application/controllers/welcome.php
Change `'welcome_message'` to `'index'` in `$this->load->view('welcome_message');`

CakePHP

Step 1
Paste all CakePHP files EXCEPT /index.php

Step 2
Move and Rename /index.php to /app/View/Pages/index.ctp

Step 3
Copy CakePHP's /index.php into /

Step 4
Move /scripts and /styles folders into /app/webroot/

Step 5
Edit app/webroot/scripts/xbox.js
```
rewrite: true,
games: "Games ",
user: "Users",
```

Step 6
Move /User.php to /app/Model/User.php

Step 7
Edit User.php: `class User extends AppModel`

Step 8
Move /Game.php to /app/Model/Game.php

Step 9
Edit Game.php: `class Game extends AppModel`
Uncomment call to `parent::_construct()`

Step 10
Move /UsersController.php to /app/Controller/UsersController.php

Step 11
Edit UsersController.php
`class UsersController extends AppController`
Delete entire `__construct()` function

Step 12
Move /GamesController.php to /app/Controller/GamesController.php

Step 13
Edit GamesController.php
`class GamesController extends AppController`
Uncomment `public $uses = array("Game", "User");`
Delete `$this->load->model('Game');`
Delete `$this->load->model('User');`

Step 14
Edit /app/View/Layouts/default.ctp
Delete everything and replace with
`<?php echo $this->fetch('content'); ?>`

Step 15
Edit /app/Config/routes.php
Change `'home'` to `'index'` in `Router::connect('/', array('controller' => 'pages', 'action' => 'display', 'home'));`

Step 16
Final Step: Rename /app/Config/database.php.default to /app/Config/database.php

NOTE: You will want to clean up the comments after migration