Lab 1: View Classes

In this lab you will apply the Model-View-Controller pattern to divide your static HTML files into View classes that encapsulate common page and site elements, such as headers, footers, navigation, etc.

Objectives & Outcomes

Successful completion of this activity will show that you can:

- Extract HTML for common elements into reusable templates
- Create classes in server-side languages such as PHP and ColdFusion
- Create classes that implement the View part of the server-side MVC pattern
- Explain the difference between the server-side and client-side implementations of the MVC pattern

Level of Effort

This activity should take approximately 4h to complete. It will require:

- 30m Research
- 5m Prep & Delivery
- 3h Work

If you find that this activity takes you significantly less or more time than this estimate, please contact me for guidance.

Reading & Resources

Books

PHP Objects, Patterns, and Practice
Chapters:
2-3
Pages:
11-39
Author:
Matt Zandstra

Publisher:
Apress
Edition:
3 (2010)
The introductory chapters on object basics will be the most helpful for now.
Object-Oriented Programming in ColdFusion

Chapters:
1
Pages:
10-35
Author:
Matt Gifford

Publisher:
Packt
Edition:
1 (2009)
This introductory chapter covers how to code, instantiate, and use objects in ColdFusion.

Web link

Sproutcore MVC vs Rails MVC
This post contains an excellent explanation of the differences between the data flow in a server-side app and a client-side app. While the title says it is about Rails (server-side) and Sproutcore (client-side), it could just as easily apply to PHP/ColdFusion versus Flex/jQuery.

Instructions

For this lab you will be converting static HTML to PHP and ColdFusion. You will need your HTML and CSS from your Web Standards Project site. If you do not have an archive of this material, you may use another student's HTML and CSS or may work with a local copy of a public web site and assets approved by the instructor.

Development

In the class Git repository, find the `ssl_labs` folder. Make sure you have a subfolder with your name, such as `rosborne`, inside it. Inside your folder, create a new folder named `01_views`. Copy your HTML and CSS assets into this folder. If you like, you may recreate an existing folder structure, such as `assets`, etc, inside your `01_views` folder. You should end up with a folder structure like:

`ssl_labs/yourname/01_views`
Choose three pages from your site that use distinctly different layouts. For example, you might choose the home page, a category page, and a detail page. You will create two versions, one PHP and one CFML, of each of these three pages, for a total of size items. You must create View classes and templates to encapsulate each of the presentation items on the pages. For example:

`index.html`	`index.php`	`index.cfm`
`category.html`	`category.php`	`category.cfm`
`detail.html`	`detail.php`	`detail.cfm`
	`SiteView.php`	`SiteView.cfc`
	`views/`	`views/`

	`category.php`	
`category.cfm`		
	`detail.php`	`detail.cfm`
	`footer.php`	`footer.cfm`
	`header.php`	`header.cfm`
	`home.php`	`home.cfm`
	`topnav.php`	`topnav.cfm`

In the above table, the static site has 3 pages. PHP and ColdFusion versions of those landing pages have been created, as well as View classes to wrap the presentation layer functionality. A `views` folder has been created to hold the templates for each presentation item. In this way, the landing pages should be very simple, just instantiating the View class and calling the `show` method to include the appropriate templates for that page.

You **do not** have to choose these 3 exact pages, or these exact templates. But you should make an effort to ensure that you have extracted common elements into shared templates to reduce repeated code.

Git Commits

Part of your grade for this project is based on regular, meaningful commits to the Git repository. You are encouraged to use a Pomodoro timer or some other method to remind yourself to commit often. Your commit messages should explain exactly what you have done with enough detail that anyone else would be able to follow along without having to be familiar with your code.

You should aim for no more than 30 minutes between commits, or no more than 2 or 3 files changed per commit.

Video Reflection

This lab also includes a video reflection assignment. You will need to register for a video sharing service (such as YouTube, Vimeo, Jing, Viddler, etc). After you have completed the development portion of the assignment you will record a video, screencast or talking head, which answers these questions:

1. What didn't go as well as you had hoped? Where did you run into problems with the lab?
2. What went better or was easier than you expected? What went right?
3. Do you think the MVC pattern and View classes are worth the time to develop? Why or why not?
4. How do you think what you did in this lab is going to help you in your future school work, career, or life?

You should show off your end result, working or not. Shoot for your videos to be 2-3 minutes long. The intent of the videos isn't a code review, so please don't just walk line by line through your code, but if you did something you feel is clever and worth showing off then please include it.

These videos will be shown in class before the next lecture. Upload your video to your sharing service and get the link to the watchable video. Create a file in the `ssl_videos` folder named `yourname.txt` and include the URL for your video. Commit this file to the repository.

Do not put your video file in the Git repository.

Where should you be?

By the end of this assignment you should feel comfortable extracting HTML into PHP and ColdFusion templates for views. You may not be able to create PHP or ColdFusion objects off of the top of your head, or may not feel 100% comfortable with the syntax of each, but you should be able to code basic functionality in each with the aid of Internet or book resources. You should be able to follow the server-side page request lifecycle.

Deliverables

Make sure your source code, assets, and reflection video URL are committed to the repository. Push your changes to the server before the start of the next lecture.