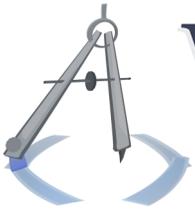# Website Architecture

# Project 3

The specification for Project 3, which tests proficiency with maintaining and modifying basic data-driven websites.

**Michael Serritella**

Summer 2010

# Website Architecture

# Project Goals

This project tests your proficiency in PHP and simple data-driven websites. You don't have to design a site's content or organization; that has already been done. Nor do you have to write any of the HTML or PHP which frames the site. You should only have to write the minimum code necessary to exercise the core skills for this project.

You will display a site whose principal content comes from a database. You will accept user input which changes the database. You will add to the database structure to accommodate new, interactive site features.

# General Description

You will be rendering one forum page, and you will be adding a feature so that everyone can Like everything.

A forum has been outlined for you, and you will pull forum posts from a database and render them. Your job primarily consists of connecting to a SQLite database, populating forum-post PHP objects with its data, and outputting the objects. Additionally, you will accept new forum posts, in which case the user management (i.e. session code) and HTML form handling have been written for you; you must affect the database with this new data. Finally, you will add new structures to the database which allow users to express their liking of forum posts, and you will manage the input of incoming likes, the PHP code of which has been partially written for you.

A shell for each of the pages has been provided, written in PHP and using the MiniArc library, the primary component of which is the HTPPS. You do not need to worry about the aesthetics of the site, the validity of the markup, or subtle browser-compatibility issues. You may modify all of the provided code, as long as you still use PHP and SQLite and satisfy all of the other requirements of this document. You do not need to use MiniArc (which includes the HTPPS) in any of the code that you write, and you may even deconstruct the code given. However, it should make your life much easier.

The project will be graded using Mozilla Firefox 3.0 or newer for Linux. Your submission should not differ significantly when viewed in any version of this browsers; however, if it does, tell the instructor, so that the most favorable version may be used.
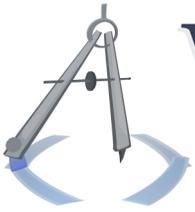
# Website Architecture

## Provided Materials

All provided materials are on the course website in a file called "CIS4930_WAM_2010Su_Project3_Provided.zip". This includes:

- The `public_html` folder, which should be the site's document root. This contains scripts and files that may be requested via `http://localhost`.

- An `includes` folder, which is a sibling to `public_html`.

- A `MiniArc` folder, which is a child of `includes`. `MiniArc` contains the MiniArc distribution, along with a derived/customized pretty printer that writes the pages of this site (e.g. site logo, header, footer) and helps to manage user input. This modified pretty printer is in `MiniArc/Specialized`.

- Site-specific PHP includes inside of `includes`, which provide classes for business objects and functions which use the pretty printer to write the site's pages.

- A `DatabaseSource.sql` file in the root of the archive, which describes the database in its initial condition. This must be used to build a SQLite 3 database which works with the site; it may be hidden or discarded afterwards.

Directly within `includes` are perhaps the most interesting components for this project. The `ForumPost` class describes all the state of a forum post before your modifications, and it corresponds to the data of a forum post stored in the database. The `ForumThread` class strings together a list of The `ForumPost`s. The `ForumUser` class describes a user of the forum, which will be useful in rendering the user's posts.

The PHP class `ForumSession`, defined in `SiteObjects.inc`, contains static methods that will help you to determine which forum user is the current visitor of the page.

**NOTE:** In order to modify the SQLite database, the Web server's Unix account will have to have write permissions on both the SQLite file and its directory. You can easily accomplish this by setting the permissions such that any user can write to the file. From the Unix prompt, go to the directory of your SQLite database and type this:

**chmod ugo+w <TheDatabaseFilename>**
**chmod ugo+w .**
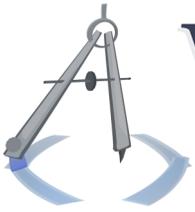
# Requirements

This section details the components of your project along with their requirements, followed by general requirements which apply to the whole project. Each of the specific sections corresponds to one page of the site.

## Forum post rendering

These requirements apply to the rendering of each forum post, including your modifications. These items mention CSS classes which have been provided for you. You can write your HTML to conform to these CSS classes, or you could write your own HTML structure and CSS classes, but you would probably be wasting your time. If you design your own styles, each of the below items shall appear legible and distinct from the others.

**Container** The container element of each forum post should have the CSS class "ForumPost" to take advatnage of existing styles.

**Author area** An HTML element that contains the information on the post author; this should have CSS class "ForumPostAuthor".

**Username** The username of the author; it should have the CSS class "ForumPostUsername". This text is guaranteed to be plain without markup, though it may contain characters that should be escaped for HTML.

**Avatar** The avatar image of the author; it should have the CSS class "ForumPostAvatar".

**Message area** The principal message of the forum post, which contains the text of the post and its title, should be contained within an element with the CSS class "ForumPostMessage".

**Title** The title of the forum post; it should have the CSS class "ForumPostTitle". This text is guaranteed to be plain without markup, though it may contain characters that should be escaped for HTML.

**Body** The principal text of the forum post; it should have the CSS class "ForumPostBody". This text is guaranteed to be plain without markup, though it may contain characters that should be escaped for HTML.

**Likes** The container of Like information. This area shall print the usernames of people who Like the forum post, and it shall provide a means for someone to Like the post if he/she has not already done so. A user cannot stop Liking a post. The option to Like a post shall lead to the relative URI `LikePost.php?PostID=<Key>`, where *<Key>* is the database primary key of the forum post.
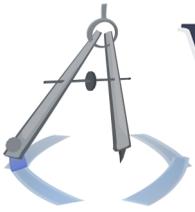
# New replies

These requirements apply to the handling of new replies to the forum post.

**Form input**  The HTML form input for new replies will be handled at the page `Reply.php`. The default implementation of this page can be found in a function within `Pages.inc`; you may change this if you like. By default, the HTML form data is sanitized, and you may assume that it refers to valid data (e.g. the post author's ID). It is sanitized in-place, so that it will be accessible via `$_POST`.

**Database**  The new replies shall be inserted into the database in the same table as the existing replies. The database keys of the replies shall not necessarily imply their order in the rendering, so do not rely on this.

# Like management

These requirements apply to the Like system, including its structure in the database and the handling of Like-related inputs.

**Form input**  The HTML form input for new Likes will be handled at the page `LikePost.php`. The default implementation of this page can be found in a function within `Pages.inc`; you may change this if you like. By default, the HTML form data is sanitized, and you may assume that it refers to valid data (e.g. the post's ID). It is sanitized in-place, so that it will be accessible via `$_GET`.

**Database**  The Like system shall be desiged in the database according to the relational model, so that only the minimal, most sterile information is stored in order to keep track of who Likes what. It should be easy for a user to Unlike a post, should that feature ever become available (i.e. that should be one SQL command).

**PHP**  Your PHP representation of Likes has no specific requirement. You may want to make your own class for them and then make objects of the class available to the class `ForumPost`. You may want to edit `ForumPost` and incorporate the Like information directly. It's your design.

# Database

The database shall be a SQLite 3 database, and it shall be kept out of the document root. It shall originate from the given `DatabaseSource.sql` file.
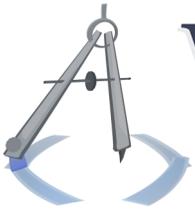
## General requirements

These requirements apply to the project overall.

### Value-added content

For fun or education, any benign content may be added to that which is required - except that which is required to not exist - and it would not incur a penalty.

# Submission

The project must be contained within a folder called "Project3_[Surname]", where [Surname] is your surname, using only alphabetic characters, without spaces, with the first letter of each word in your surname capitalized within the representation. That folder must be contained within a .zip file, such that the folder is the top-level element in the zip file. The zip file must be named "Project3_[Surname].zip".

In Linux, you can achieve this by navigating to the parent of your project directory and executing the command:

**zip -r Project3_[Surname].zip Project3_[Surname]**