WEBD 236
Web Information Systems Programming

Week 3

Copyright © 2012 Todd Whittaker
(todd.whittaker@franklin.edu)

Agenda

• This week’s expected outcomes
• This week’s topics
• This week’s homework
• Upcoming deadlines
• Solutions to homework 1
• Questions and answers
Week 3 Outcomes

• Distinguish between Model 1 and Model 2 architecture web applications
• Employ the Model-View-Controller design pattern in web development
• Utilize RESTful URLs for clean design
• List the advantages to unit testing

Web Architecture

• Model 1 architecture
  – Code for application, database, and display logic is intermixed in a single monolithic page
  – A very tangled set of interactions.
    • Ex: To do application

To do list

1. Write slides for WEBD236
2. Prepare a model 1 architecture example
<?php
    global $db;
    try {
        $db = new PDO('sqlite:ToDoList.db3');
    } catch (PDOException $e) {
        die("Could not open database. " . $e->getMessage());
    }

    if (isset($_POST['description'])) {
        $desc = htmlentities($_POST['description']);
        $statement = $db->prepare("INSERT INTO todo (description, done) values (?, 0)" );
        $statement->bindParam(1, $desc);
        $statement->execute();
    }

    $statement = $db->prepare("SELECT * FROM todo WHERE done = 0 ORDER BY id");
    $statement->execute();
    $rows = $statement->fetchAll();
?>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
    <title>To do list</title>
</head>

Web Architecture

Some logic to handle the database

Some simple validation and security logic

Some logic to generate our list

Some static output
Web Architecture

• Model 1 architecture
  – Works fine for simple applications
  – Becomes horribly messy when
    • Application logic is complicated
    • Error/security reporting is robust
    • You need to debug or make changes
  – Problem: solving three issues in one place.
  – Solution: separate three issues into three places.
MVC-based Architecture

- Model 2:
  - Model-View-Controller (MVC)
    - Model: interacts with the database
    - View: presents data to the user
    - Controller: encapsulates “business logic”
  - Flow of a web request
    - URL maps into a controller
    - Controller updates or retrieves data via model
    - Controller forwards request to a view

- Browser sends a request to a web server
- Web server fires a routing script to determine what other script (based on URL) should handle the request.
- Routing script dispatches the request to the appropriate controller.
- Controller invokes data-oriented actions on the model
MVC-based Architecture

- Model updates/queries the database
- Model returns results to the controller.
- Controller takes results, applies business logic, and forwards results to the view.
- View renders the HTML and returns it to the controller.
- Controller returns HTML to web server and then to browser.

We’ve already done some of this last week:
- Separation of DB instantiation into db.inc
- Separation of model into model_student.inc
- Separation of view into view_student.php

But, we need to structure this better!
MVC-based Architecture

Controller

View

Model

Dispatching
Routing
Web Server

Start here, with better URLs, routing, and dispatching.

Running example – To Do List

To Do List

Description: [ ] Add

Current To Do:

1. [View] [Edit] [Del] Teach class on Wednesday, 7:30 PM EST.

Past To Do:

1. [View] [Edit] [Del] Write slides for WEBD236
2. [View] [Edit] [Del] Prepare a model 1 architecture example
3. [View] [Edit] [Del] Prepare a model 2 architecture example

Copyright © 2012 Todd Whittaker
Pretty URLs, Routing, Dispatching

• GET-based URLs can be ugly

http://localhost/app/view_todo.php?id=23

• “RESTful” URLs are much prettier

http://localhost/app/todo/view/23

Entity  Action  Parameter

But, this URL doesn’t specify a script file to execute! How do we map this into a script?
Pretty URLs, Routing, Dispatching

• .htaccess file
  – Place this in the root of your application folder (i.e. \xampp\htdocs\app)

```plaintext
Options +FollowSymLinks
IndexIgnore */*
# Turn on the RewriteEngine
RewriteEngine On
# Rules
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
RewriteRule . urlrouter.php
```

Must “allow overrides” in the Apache configuration in `c:\xampp\apache\conf\httpd.conf`
Pretty URLs, Routing, Dispatching

• .htaccess file

```html
<Directory "C:/xampp/htdocs">
  Options Indexes FollowSymLinks Includes ExecCGI
  AllowOverride All
  Order allow,deny
  Allow from all
</Directory>

RewriteEngine On
# Rules
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
RewriteRule . urlrouter.php
```

Must “allow overrides” in the Apache configuration in `c:\xampp\apache\conf\httpd.conf`.

Permits the rewrite engine to be active on a per-directory basis.

Pretty URLs, Routing, Dispatching

• URL routing and dispatching

  Three critical pieces of information for a URL like http://localhost/app/todo/view/5

```php
print_r($_SERVER['REQUEST_METHOD']);
print_r($_SERVER['REQUEST_URI']);
print_r($_SERVER['SCRIPT_NAME']);
```

GET
/app/todo/view/5
/app/urlrouter.php

Can use this to figure out the directory (app), the entity (todo), the action (view) the ID (5) and the method (GET).
Pretty URLs, Routing, Dispatching

- urlrouter.php

```php
<?php
function routeUrl() {
    $method = $_SERVER['REQUEST_METHOD'];
    $requestURI = explode('/', $_SERVER['REQUEST_URI']);
    $scriptName = explode('/', $_SERVER['SCRIPT_NAME']);

    for ($i = 0; $i < sizeof($scriptName); $i++) {
        if ($requestURI[$i] == $scriptName[$i]) {
            unset($requestURI[$i]);
        }
    }
    # continued...
}
```
Pretty URLs, Routing, Dispatching

• urlrouter.php

```php
<?php
function routeUrl() {
    $method = $_SERVER['REQUEST_METHOD'];
    $requestURI = explode('/', $_SERVER['REQUEST_URI']);
    $scriptName = explode('/', $_SERVER['SCRIPT_NAME']);

    for ($i = 0; $i < sizeof($scriptName); $i++) {
        if ($requestURI[$i] == $scriptName[$i]) {
            unset($requestURI[$i]);
        }
    }
    # continued...
}
$command = array_values($requestURI);
$controller = 'controllers/' . $command[0] . '.inc';
$func = strtolower($method) . '_' . (isset($command[1]) ? $command[1] : 'index');
$params = array_slice($command, 2);
# continued...
```
Pretty URLs, Routing, Dispatching

• urlrouter.php

```php
if (file_exists($controller)) {
    require $controller;
    if (function_exists($func)) {
        $func($params);
        exit();
    }
    else {
        die("Command '$func' doesn't exist.");
    }
} else {
    die("Controller '$controller' doesn't exist.");
}
routeURL();
```

Pretty URLs, Routing, Dispatching

• What we’ve done:
  – Given a URL `http://localhost/app/todo/view/5`
    • Included the file 'controllers/todo.inc'
    • Called the function 'get_view' with a parameter array containing the value '5' at index 0.

  – What should get_view do?
    • It’s a controller.
      – Validate parameters
      – Query the model
      – Render the view
Controllers

Controllers tie together the model (DB), view (HTML), and the business logic. Also commonly perform validation.

```
<?php
include_once "include/util.inc";
include_once "models/todo.inc";

function safeParam($arr, $index, $default) {
    if ($arr && isset($arr[$index])) {
        return $arr[$index];
    }
    return $default;
}

# continued ...
```
Controllers

• controllers/todo.inc

```php
<?php
include_once "include/util.inc";
include_once "models/todo.inc";

function safeParam($arr, $index, $default) {
    if ($arr && isset($arr[$index])) {
        return $arr[$index];
    }
    return $default;
}

# continued ...
```

Models contain functions that interact with the database.

```php
function get_view($params) {
    $id = safeParam($params, 0, false);
    if ($id === false) {
        die("No todo id specified");
    }
    $todo = findToDoById($id);
    if (!$todo) {
        die("No todo with id $id found.");
    }
    renderTemplate("views/todo_view.inc",
                  array(
                      'title' => 'Viewing To Do',
                      'todo' => $todo));
```
Controllers

- controllers/todo.inc

```php
function get_view($params) {
    $id = safeParam($params, 0, false);
    if ($id === false) {
        die("No todo id specified");
    }
    $todo = findToDoById($id);
    if (!$todo) {
        die("No todo with id $id found.");
    }
    renderTemplate(
        "views/todo_view.inc",
        array(
            'title' => 'Viewing To Do',
            'todo' => $todo));
}
```

There are more controllers in this file, i.e. get_list, get_edit, post_add, post_edit, etc.

Models

Models abstract the access to the database. Handle create, read, update, delete (CRUD) of rows. Frequently contain many `findByXXX()` functions: one update, one insert, one delete function.
Models

- models/todo.inc

```php
<?php
include_once 'models/db.inc';

function findToDoById($id) {
    global $db;
    $st = $db -> prepare('SELECT * FROM todo WHERE id = ?');
    $st -> bindParam(1, $id);
    $st -> execute();
    return $st -> fetch(PDO::FETCH_ASSOC);
}
?>
```

- models/db.inc

```php
<?php
global $db;
try {
    $db = new PDO('sqlite:ToDoList.db3');
} catch (PDOException $e) {
    die("Could not open database. " . $e->getMessage());
}
?>
```
Views allow us to return pages back to the user. But, we want to keep programming logic in views very simple.

- **Programming logic in views**
  - We don’t want any validation, business rules, etc. in our views
  - Only want *display-based* logic, i.e. a loop over the rows in a result set

- **Templating**
  - Can use includes in PHP to pull in sections but...
  - PHP is an ugly templating language
Views

• Programming logic in views
  – We don’t want validation, business rules, etc. in our views
  – Only want display-based logic, i.e. a loop over the rows in a result set

• Templating
  – Can use includes in PHP to pull in sections but...
  – PHP is an ugly templating language

Ugly code just to render the value of a variable:
```php
<?php echo $someVar ?>
```
This would be nicer:
```html
{{$someVar}}
```

Views

• Solution: build a simple templating engine

```html
%% views/header.html %%
<h1>{{$title}}</h1>
<div class='display'>
  <label>Description:</label>
  <div class='value'>{{$todo['description']}}</div>
  <label>Done?:</label>
  <div class='value'>{{$todo['done'] ? 'yes' : 'no'}}\</div>
</div>
<p><a href="@@index@@"><< Back</a></p>
%% views/footer.html %%
```
Views

• Solution: build a simple templating engine

```html
%% views/header.html %%
<h1>{{$title}}</h1>
<div class='display'>
  <label>Description:</label>
  <div class='value'>{{$todo[description]}}</div>
  <label>Done?:</label>
  <div class='value'>{{$todo['done'] ? 'yes' : 'no'}}</div>
</div>
<p><a href='@@index@@'><< Back</a></p>
%% views/footer.html %%
```

Things enclosed in `%%` are imported.

Things enclosed in `{{ }}` are variables to echo to output.

`Standard PHP can be enclosed in `[]`. (not shown)`

Things enclosed in `@@ @@` are "relative" URLs.

This becomes...
Views

• Solution: build a simple templating engine

<!-- views/footer.html -->

```
<p>Back</p>
<a href="/app/index"> <-- content div --></a>
```

%v view/head
<h1>{{$title}}</h1>
<div class="d">
  <label>Description:</label>
  <div class="value">
    <?php echo $todo['description']]); ?></div>
  <label>Done?:</label>
  <div class="value">
    <?php echo $todo['done']; ?></div>
</div>

</p>
<a href="/app/index"></a></p>
<div class="footer">
  Copyright © 2012 Todd Whittaker
</div>
</div>
</body>
</html>

But, where do variables like $title come from?
Views

- Solution: build a simple templating engine

```php
views/footer.html
%%
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
  <title><?php echo($title); ?></title>
  <link rel="stylesheet" href="/app/static/style.css" />
</head>
<body>
<div class="content">
  <h1><?php echo($title); ?></h1>
  <div class="display">
    <label>Description:</label>
    <div class="value">(<?php echo($todo['description']); ?>)
    <label>Done?:</label>
    <div class="value">(<?php echo($todo['done'] ? 'yes' : 'no'); ?>)
  </div>
</div>
<p><a href="/app/index"><< Back</a></p>
</div>
<div class="footer">
  Copyright © 2012 Todd Whittaker
</div>
</body>
</html>
```

But, where do variables like $title come from?

From the array of variables passed into renderTemplate from the controller.

```php
renderTemplate("views/todo_view.inc", array(
    'title' => 'Viewing To Do',
    'todo' => $todo
));
```

default pseudocode.

Working code in include/util.inc

```php
if the template doesn't exist:
  die
if there is a cached version of the view:
  load cached version
else:
  load the template into contents string
  resolve any %% template %% imports recursively
  replace any @@ URL @@ with an absolute URL
  replace any "{{" with "<?php echo(" 
  replace any "}" with "); ?>"
  replace any "[" with "<?php "
  replace any "]" with "?>"
  write out the new contents to a cache file
  extract the array of template parameters
  make PHP interpret the contents
  echo the interpreted contents
```
Views

if the template doesn't exist:
    die
if there is a cached version of the view:
    load cached version
else:
    load the template into contents string
    resolve any %% template %% imports recursively
    replace any @@ URL @@ with an absolute URL
    replace any "{{" with "<?php echo(""
    replace any "}}" with 
    replace any "[[" with "<?php 
    replace any "]]]" with "?>"
    write out the new contents to a cache file
    extract the array of template parameters
    make PHP interpret the contents
    echo the interpreted contents

Plain relative URLs don’t work because accessing “todo/delete/5” in the browser from the page “/app/todo/view/1” would resolve to “app/todo/view/todo/delete/5”

Last Words on MVC

• Remember the flow:
  – Router/dispatcher loads the controller and calls the right function
  – Controller function invokes the model for data access or update and then returns a View by:
    • Rendering a template (for GET requests)
    • Redirecting to a URL (for a POST request)
Testing and Debugging

• Three kinds of errors
  – Syntax errors: show up as ❌ in Aptana. Relatively easy to fix (missing semi-colons, quotation marks, parentheses, curly braces, etc). Also show up as parse errors if you try to run the bad script:

  **Parse error**: syntax error, unexpected T_FUNCTION in \C:\xampp\htdocs\model2\include\util.inc** on line **8**

• Three kinds of errors
  – Runtime errors: only show up when you’re running the code (i.e. clicking ‘refresh’). Some are fatal, others are warnings. Also relatively easy to fix.

  **Warning**: include() [**function.include**]: Failed opening 'foo.inc' for inclusion (include_path='.;\C:\xampp\php\PEAR') in **\C:\xampp\htdocs\model2\include\util.inc** on line **7**
Testing and Debugging

• Three kinds of errors
  – Logic errors: The silent killers; very difficult to find and fix. Your algorithm doesn’t do what you think it should do. Now you need to debug.
  • Use `echo`, `print`, and `print_r` in conjunction with your browser’s “view source” option to see the values of variables.

• There is an easier way.
  – Write testing code to make sure that your production code works like it should.
  – Can be very simple, or you can use a testing framework.
<?php
function assertEquals($expected, $actual) {
    if ($expected != $actual) {
        die("Expected: $expected but got $actual");
    }
}
function calcGasMileage($milesDriven, $gallonsUsed) {
    return $milesDriven / $gallonsUsed;
}
function testCalcGasMileage() {
    $miles = 300;
    $gallons = 15;
    assertEquals(20, calcGasMileage($miles, $gallons));
}
testCalcGasMileage();
print("Everything is fine!");
?>
Testing and Debugging

```php
<?php
function assertEquals($expected, $actual) {
    if ($expected != $actual) {
        die("Expected: $expected but got $actual");
    }
}

function calcGasMileage($milesDriven, $gallonsUsed) {
    return $milesDriven / $gallonsUsed;
}

function testCalcGasMileage() {
    $miles = 300;
    $gallons = 15;
    assertEquals(20, calcGasMileage($miles, $gallons));
}

testCalcGasMileage();
print("Everything is fine!");
?>
```

Called a “unit test” since we’re testing a single small unit of code (one function in this case). What other numbers should we try?

Testing and Debugging

• Advantages of unit testing
  – Tests run on the server for your business logic
  – Tests build confidence that you are building the system right
  – Tests can be rerun after every change you make to ensure that they still pass.
  – Writing testable code is similar to writing code others can easily read.
Testing and Debugging

• Some PHP unit testing frameworks
  – PHPUnit: https://github.com/sebastianbergmann/phpunit
  – SimpleTest: http://www.simpletest.org/

• Note, testing your JavaScript or GUI is vastly different
  – Jasmine for JavaScript
  – Selinium for GUIs/request-response

Upcoming Deadlines

• Readings for next week
  – Chapters 7 and 8 in *PHP and MySQL*

• Assignments
  – Homework 2 due January 22
  – Lab 1 due January 22

• Next week:
  – Forms and data, control statements
Solutions to HW 1

General Q & A

- Questions?
- Comments?
- Concerns?