CS437/637 Midterm Review

Coverage: Duckett

- Chapter 1-2: Basics: Can skip pp. 53-56
- Chapter 3: Lists: all important
- Chapter 4:Links: all important
- Chapter 5:Images: can skip “old code”
- Chapter 6: Tables: all important
- Chapter 7:Forms: can skip fieldset, legend
- Chapter 8: Extra Markup: can skip iframe, meta info
- Chapter 9: Video and Audio: skip Flash (pp.202-212)
- Chapter 10: Intro CSS: all important

Coverage: Duckett

- Chapter 11: Color: Can skip pp. 255-256
- Chapter 12: Text: can skip @font-face, pp. 282-284, 288-292
- Chapter 13: Boxes: can skip pp. 318-322
- Chapter 14: Lists, tables, and forms: just understand the aligning form controls problem—see slides.
- Chapter 15: Layout: understand float
- Chapter 17: HTML5 Layout: all important
- Last few pages: useful indexes

Coverage: M&H

- Chapter 1: Intro
  - Can skip pp. 10-13, 22-23, 34-on (no exam questions on Netbeans, Notepad, phpAdmin)
- Chapter 2: PHP app basics: all important
  - Pg. 65: note that “isset($var)” requires $var “is set” (i.e. exists) and is not NULL
  - Pg. 73: skip NOT, AND, OR: use !, &&, || as in Java.
- Chapter 3: Intro MySQL
  - Note that we are assuming database identifiers are caseless, as in the SQL standard.
  - No exam questions on how things work in phpMyAdmin, just command line use.

- Chapter 4: PHP + MySQL
  - Note that we want to enable exceptions and put a try-catch around each database access.
  - In this chapter, the examples have PHP files all in the same directory, making includes easy.
  - However, the examples do not follow MVC rules exactly: the index.php on pg. 149 has controller code and then view code all in the same file.
  - It would be easy to fix this up to be MVC.
- Chapter 5: MVC
  - Pg. 161 important diagram
  - Pg. 164: redirection: send special response with new URL for browser to use immediately. Use here: to get the controller to restart its processing.
  - Make sure you understand everything about the two apps in this website, and how the code is subdivided into model, view, and controller code.

- Chapter 6 Testing
  - Types of errors: syntax, runtime, logic
  - PHP has non-fatal runtime errors, unlike Java
- Chapter 7 Forms
  - Also, HTML character entities like <
  - Use of htmlspecialchars() to change < to <, etc., “sanitize” strings
- Chapter 8 Control Statements
  - Control statements themselves are just like Java
  - Type coercion AKA type juggling: can lead to bugs
  - Use of identity operators === and !== to avoid type coercion
  - Dangerous to mix strings and numbers when they can get compared, as in searching and sorting
- Chapter 9: Strings and Numbers
  - Can skip htmlentities() function, sprintf, can always lookup details on a function

- Chapter 10 Dates—skip
- Chapter 11 Arrays
  - Arrays can have gaps, count($a) doesn't count gaps, just non-null elements, array_values($array) returns reindexed array, no gaps
  - Skip end($a) and key($a): use foreach to process an array with gaps (as well as ones without gaps)
  - Arrays of arrays: important way to hold a collection of items, each with attributes, for example.
  - Array assignment copies array contents, and this happens when an array is used as a function argument (without &).
  - PHP is smart about array copying, so not expensive usually.

- Chapter 17 Creating DBs
  - Pg. 569: script my_guitar_shop2.sql
- Chapter 18 SQL: treated as review
- Chapter 19 PDOs
  - Pg. 625: how to turn on exceptions

Homework

- HW1
  - Basic HTML, URLs, SQL
  - Loading mysql db using command line
  - Writing and running command-line PHP
- HW2
  - HTML forms
  - Using XAMPP, book_apps
  - Netbeans project (but not covered on exam)
  - CSS
  - Basics on smartphones vs. desktops and laptops

Homework

- HW3
  - Page flow of pizza project
  - Communications diagram for pizza1 topping management
  - Foreign keys
  - More SQL
  - Start on Pizza1 project (graded separately as another “homework”)

Important Ideas

- HTTP: URLs, request-response cycle, how the browser handles relative URLs
- MVC: How to write a “proper controller”: no HTML, handles all user input and incoming requests
- Associative arrays: so versatile they handle all the "data structures" we need (this capability is also available in Javascript)

The components of an HTTP URL

```
http://www.murach.com/books/index.htm
```

protocol | domain name | path | filename

Apache receives a request for this URL…
What happens if you omit parts of the URL

- If you omit the protocol, the default of http:// will be used.
- If you omit the filename, one of the default filenames for the Apache web server will be used: index.htm, index.html, or index.php.
- If you omit the filename and there is no default file, Apache will display an index of the files and directories in the path.

The URL can have parameters as well:
http://localhost/book_apps/ch04_product_viewer/?category_id=2
For this URL, Apache accesses the file
    book_apps/ch04_product_viewer/index.php
And provides it to PHP along with the URL parameter category_id=2

HTTP request-response cycle

- A HTTP request-response cycle uses one TCP connection in two-way interchange from client to server and back--
  - Client sends GET or POST request over TCP connection
  - Server receives request, interprets URL, parameters, sends back response (HTML, image file (etc.), CSS file, or redirect)
  - Client receives response, browser renders it (or immediately issues another GET if the response was a redirect)
- Server is “stateless”: treats each request as a completely new thing, spec’d by URL, parameters
- So it’s our job as programmers to pass info from one request (server execution of PHP) to the next in our app if needed, or use a database for long-term storage.
  - Or use session variables, but this will not be on the midterm exam

HTTP request in server: handed off to PHP or not

- HTTP request-response cycle:
  - Client sends GET or POST command over TCP connection
  - Server receives command, interprets URL, parameters, sends back response (HTML, image file (etc.), CSS file, or redirect)
  - Client receives response, browser renders it
- Server interprets URL:
  - ✓ If the URL ends in .html, .css, .jpg, etc., the server just reads the corresponding file and sends it back (static web server behavior)
  - ✓ If the URL ends in .php, the server hands its processing over the the PHP system:
    - ❑ The PHP system puts the request parameters in global arrays _GET[] or _POST[], runs the PHP program
    - ❑ The request parameters contain the user input from forms and links
    - ❑ The PHP program generates the response, usually HTML or redirect

A simple HTTP request

```
GET / HTTP/1.1
Host: www.example.com
```

A simple HTTP response

```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 136
Server: Apache/2.2.3

<html>
<head>
    <title>Example Web Page</title>
</head>
<body>
    <p>This is a sample web page</p>
</body>
</html>
```

Example of a POST Request

```
POST /path/script.php HTTP/1.1
Host:jmarshall.com
User-Agent: HTTPTool/1.0
Content-Type: application/x-www-form-urlencoded
Content-Length: 32

home=Coston&favorite_flavor=flies
```

Blank line after headers

Form data in body of POST request

- We never see this body text in practice, only the resulting name-value information it carries.
- PHP parses the body text into the $_POST array automatically
- We can see the “form data” home=Coston, favorite_flavor=flies using Chrome’s Inspect>Network facility

HTTP parameters

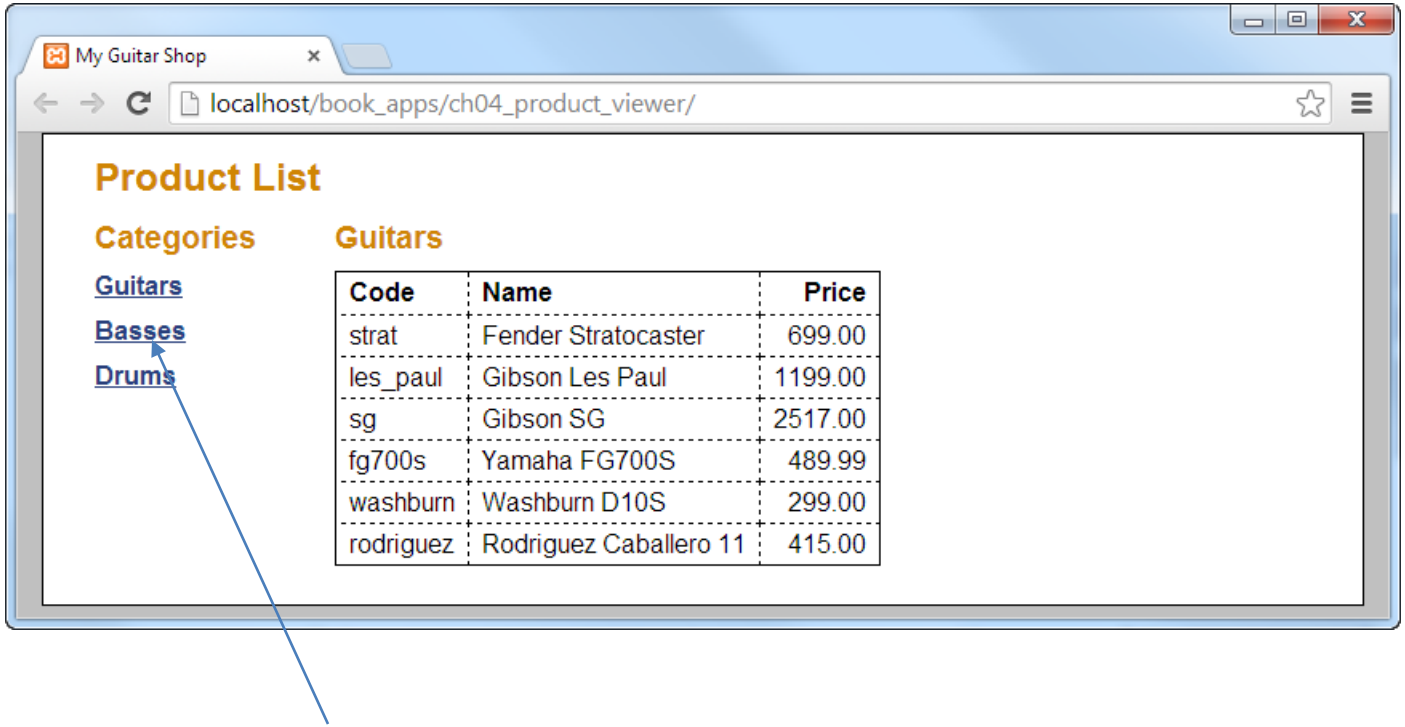
- GET case: the parameters are part of the URL
  Example: GET request with parameters

```
GET /myapp/index.php?action=add&name=foo HTTP/1.1
(headers, then blank line)
```

- POST case: the parameters are in the POST body
  Example: POST request with parameters

```
POST /myapp/index.php HTTP/1.1
(request headers, then blank line)
(POST body with encoded parameters)
```

From Chap. 4 slides: How category_id=2 got into the URL:
look at the link in first page again:

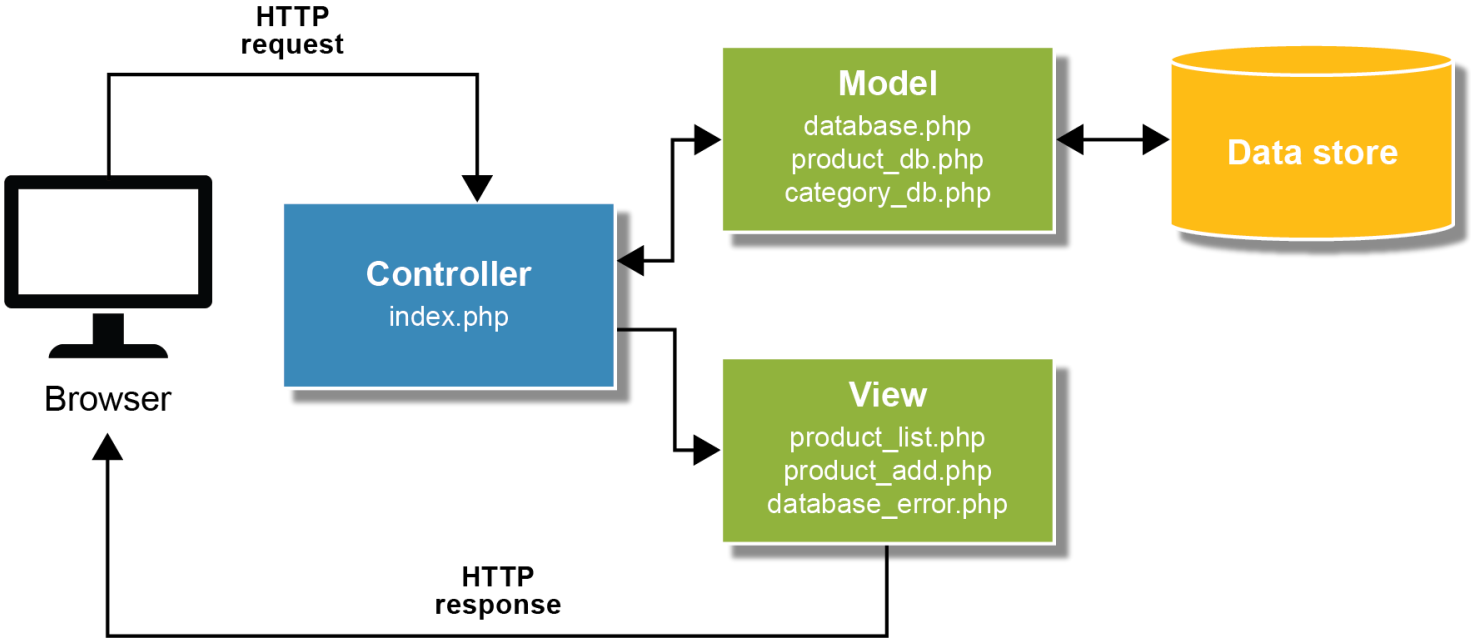


In HTML: Basses
This href is a relative URL because it has no "http://server" part
As explained next slide, this causes a GET request back to this directory:
GET /book_apps/ch04_product_viewer/?category_id=2 HTTP/1.1

Relative URL handling by Browser

- The browser finds a *relative URL* in the HTML, here “?category_id=2”
- The browser remembers where (what URL, less filename) it got the current page from, here http://localhost/book_apps/ch04_product_viewer/
- The browser generates a *full URL* relative to this location, to send to the web server (which itself has no idea where the last request came from)
- Last slide case: HTML has
  Basses
  so browser generates
  GET http://localhost/book_apps/ch04_product_viewer/?category_id=2
- The web server (Apache) receives this GET: how does it handle it?
  - Sees no filename, so looks for index.html or index.php, finds index.php, hands off to PHP with parameter category_id=2

HTTP and the webapp: The MVC pattern



- The request comes into the controller, index.php
- The controller code interprets user input from _GET[] or _POST[]
- The controller code calls model functions to get needed data
- When the controller has all the data needed for the response, it forwards the request to the view file for composing the needed HTML.

✓ In some cases, the controller redirects back to itself or another controller

MVC (model-view-controller) :

Each request coming in makes the server

1. Execute PHP code (“the controller”), in a file usually named index.php, sets variables up, accesses the DB (the “model”), then forwards to the view code.

2. The view code, in its own .php file, generates response HTML (the “view”), with help of some echo $whatever and PHP loops.
   - ✓ In some cases, the controller sends back a REDIRECT rather than HTML, causing the controller to be executed in a new request/response cycle

A simple HTTP request

```
GET /apps/foo.php HTTP/1.1
Host: www.example.com
```

A Redirect HTTP response (absolute URL)

```
HTTP/1.1 302 Found                    ← Response code 302
Location: http://www.example.com/spec/← "Location" header here
Server: Apache/2.2.3
                                      ← No Response body
```

--or—

A Redirect HTTP response (relative URL)

```
HTTP/1.1 302 Found
Location: .
Server: Apache/2.2.3
```

The browser sees the redirect response, immediately sends a GET using the absolute URL or the implied URL, relative to the earlier GET's URL's path (here http://www.example.com/apps/)

A built-in function for redirecting a request

```
header($header)
```

The header function

```
header('Location: .');          // the current directory
header('Location: ..');         // up one directory
header('Location: ./admin');    // down one directory
header('Location: error.php');
header('Location: http://www.murach.com/');
```

How to redirect a request

back to this same directory, thus index.html

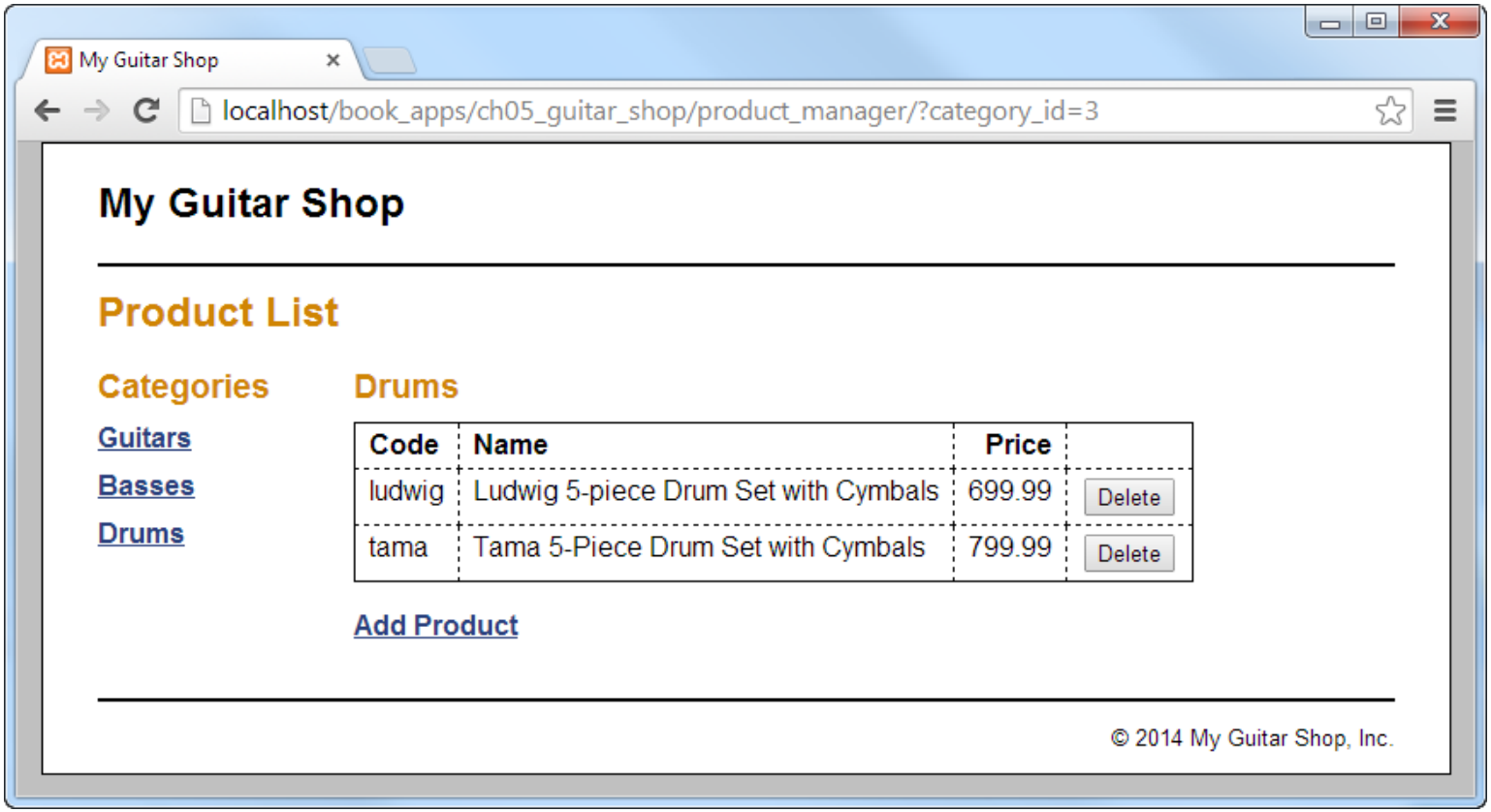
```
if ($action == 'delete_product') {
    $product_id = filter_input(INPUT_POST, 'product_id',
            FILTER_VALIDATE_INT);
    if ($product_id != NULL || $product_id != FALSE) {
        delete_product($product_id);
        header("Location: .");
    }
}
```

Note: This innocuous-looking call `header("Location: …")` sets the HTTP *response code* to REDIRECT (302), not just the Location header!

The chapter 5 website

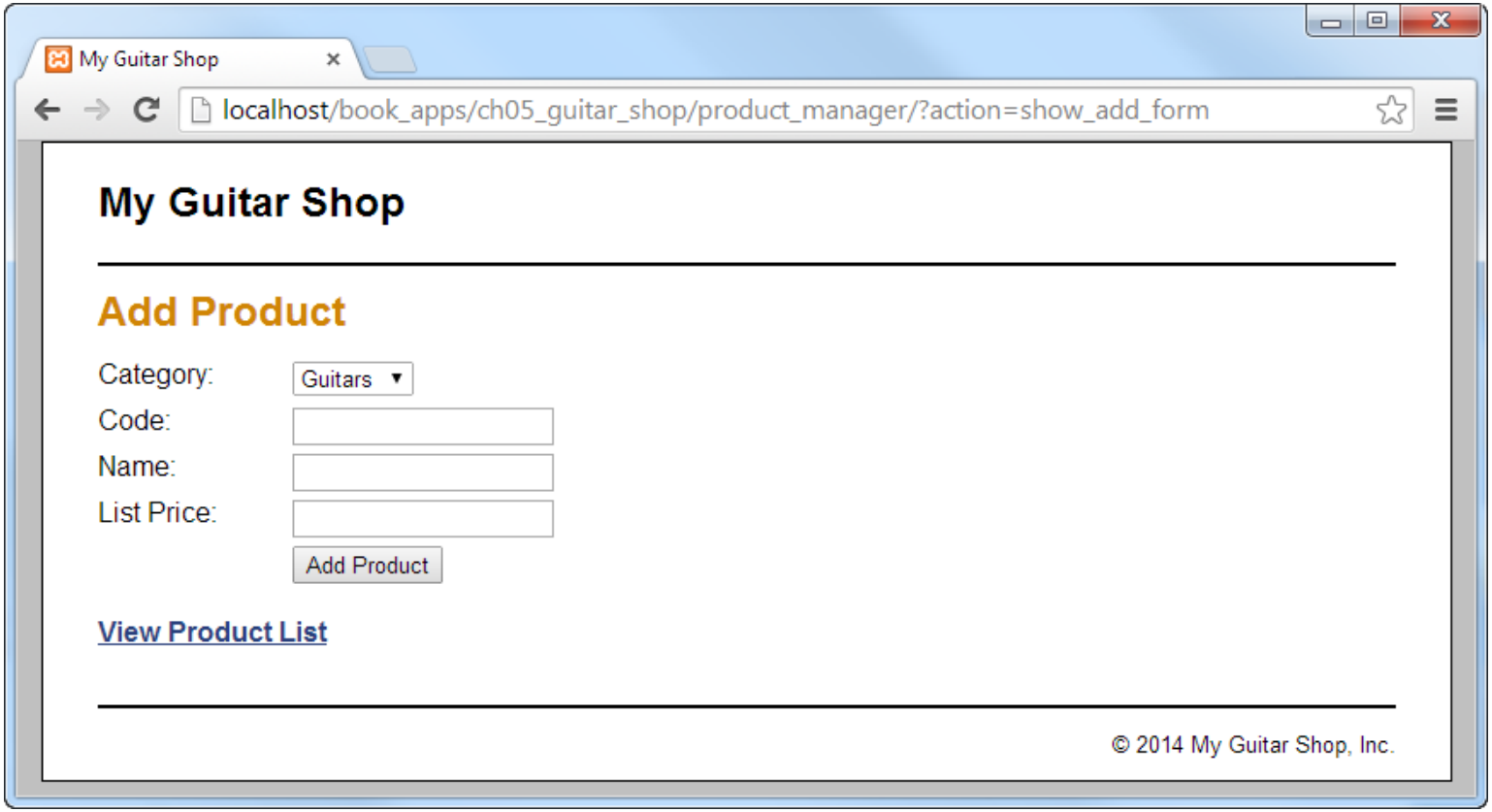
- The apps here look like the chapter 4 apps, Product Catalog and Product Manager, but their implementation uses MVC principles
- Further, there are subdirectories for each app, and other subdirectories for common files, just as we will use for the pizza project.
- Product Manager's index.php is now its "controller", and handles all UI and decides what to do
- Similarly Product Catalog's index.php is its controller, in its own directory.
- This project is a model for Project 1, i.e., pizza1.

The Product List page of the Product Manager app

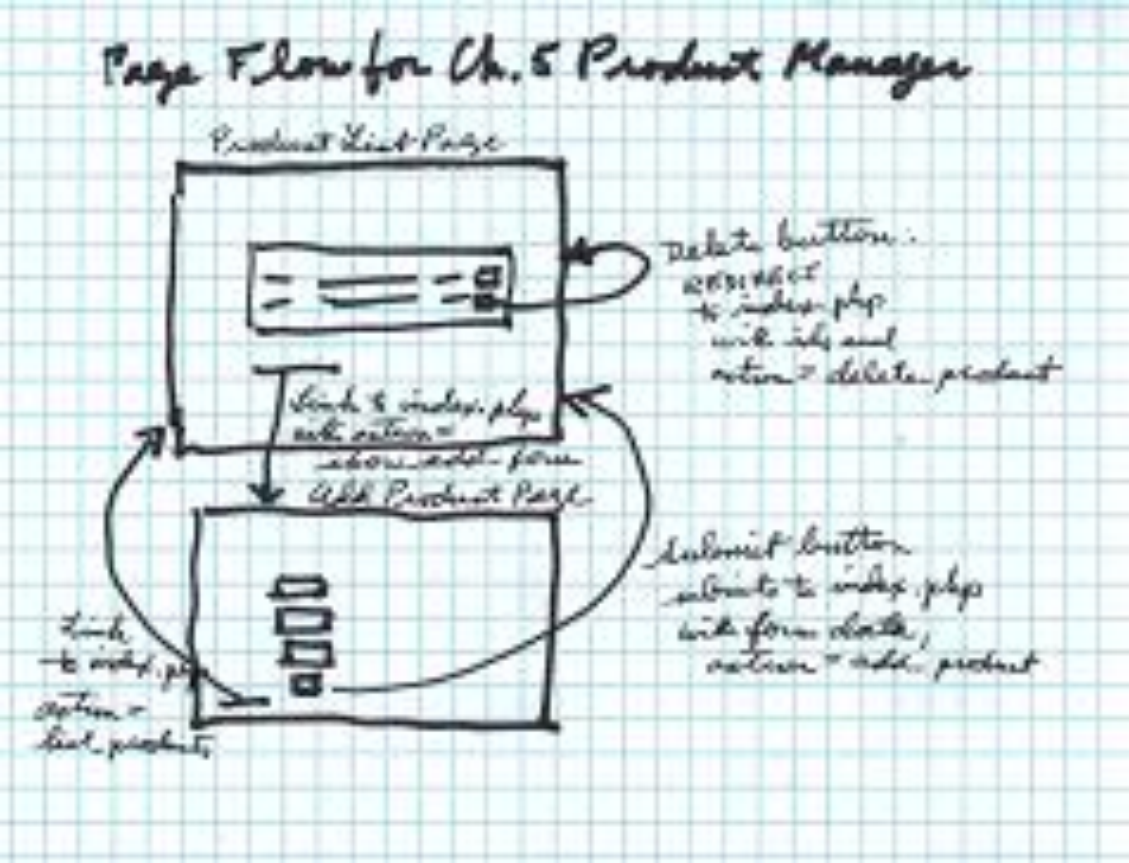


online

The Add Product page



Page Flow for Chap 5 Product Manager

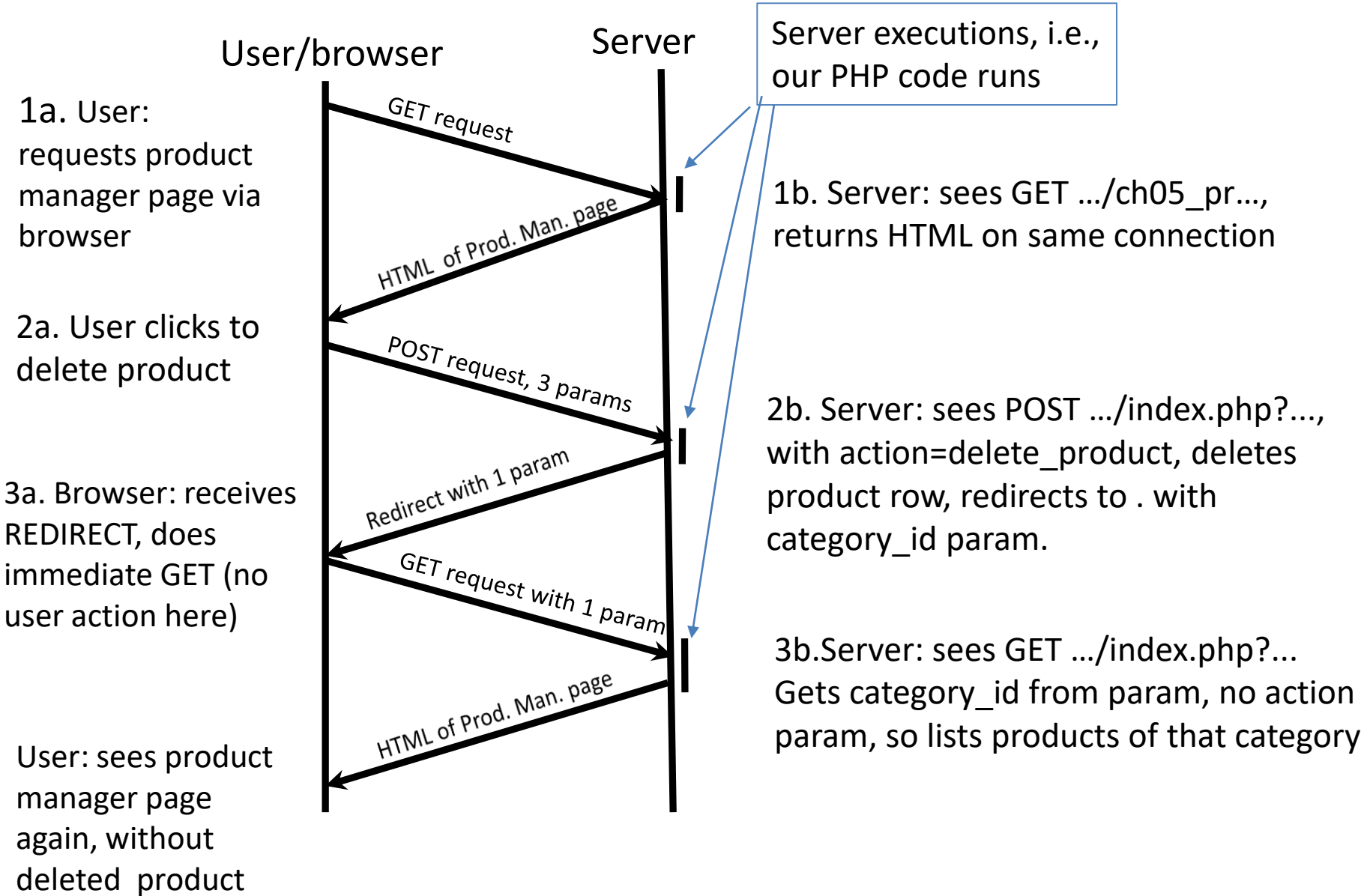


(not including error pages)

Delete a product in this app

- Unlike Chap. 4's Product Manager, this implementation does not have a separate delete_project.php for the delete action, because following MVC, all UI is handled in the controller.
- It does the delete action in index.php, and then wants to use index.php to redisplay the product list.
- But it's running in index.php, and can't include itself to do the redisplay.
- So it does the REDIRECT to . (itself) to do the redisplay in another request-response cycle.
- Similarly, after the add-product form executes, the controller accepts the form data, does the add-product in the DB, and then redirects to itself to redisplay the product list.

Communications Diagram for product deletion, chap. 5



Requests and responses on the wire

Here gs stands for guitar_shop to shorten URLs

```
GET /book_apps/ch05_gs/product_manager HTTP/1.1
Host: …
```

request

```
HTTP/1.1 200 OK
<HTML of product manager page>
```

response

```
POST /book_apps/ch05_gs/product_manager HTTP/1.1
Host:…
In body of POST: 3 params
```

request

```
HTTP/1.1 302 Found
Location: .?category_id=1
```

REDIRECT response

```
GET /book_apps/ch05_gs/product_manager?category_id=1 HTTP/1.1
Host:…
```

Browser-generated request

```
HTTP/1.1 200 OK
<HTML of product manager page>
```

response

Associative Arrays

The syntax for creating an associative array

```
array([key1 => value1, key2 => value2, ... ])
```

How to create an associative array of tax rates

With one statement

```
$tax_rates =
    array('NC' => 7.75, 'CA' => 8.25, 'NY' => 8.875);
```

With multiple statements

```
$tax_rates = array();
$tax_rates['NC'] = 7.75;
$tax_rates['CA'] = 8.25;
$tax_rates['NY'] = 8.875;
```

How to set a value with a specific key

```
$name = array('first' => 'Ray', 'last' => 'Harris');
$name['middle'] = 'Thomas';
```

What happens when you omit the key when adding a value

```
$name = array('first' => 'Ray', 'last' => 'Harris');
$name[] = 'Thomas';                    // key is 0
```

How to get a value at a specified key

```
$name = array('first' => 'Ray', 'last' => 'Harris');
$first_name = $name['first'];
$last_name = $name['last'];
```

- A regular array is just a PHP array with integer keys from 0 to n-1.
- An associative array is a PHP array with string keys.
- The book says an array with gaps in numeric indexes may also be considered “associative” (pg. 321). Or it may not, a marginal case.
- It is possible to have both types of indexes in one array.
- But it’s all one array type.
- Values can be any PHP type, even arrays…

The syntax of a foreach loop

```
foreach ($array_name as [ $key => ] $value) {
    // Statements that use $key and $value
}
```

A loop that displays the values in an array

```
$tax_rates = array('NC' => 7.75, 'CA' => 8.25,
                   'NY' => 8.875);
echo '<ul>';
foreach ($tax_rates as $rate) {
    echo "<li>$rate</li>";
}
echo '</ul>';
```

The result displayed in a browser

- 7.75
- 8.25
- 8.875

A foreach loop that displays the keys and values

```
$tax_rates = array('NC' => 7.75, 'CA' => 8.25,
                   'NY' => 8.875);
echo '<ul>';
foreach ($tax_rates as $state => $rate) {
    echo "<li>$state ($rate)</li>";
}
echo '</ul>';
```

The result displayed in a browser

- NC (7.75)
- CA (8.25)
- NY (8.875)

A simple array

[0]	[1]	[2]	[3]	[4]
‘Mike’	‘Joel’	‘Anne’	‘Ray’	‘Pren’

Suppose this array is the value of variable $group

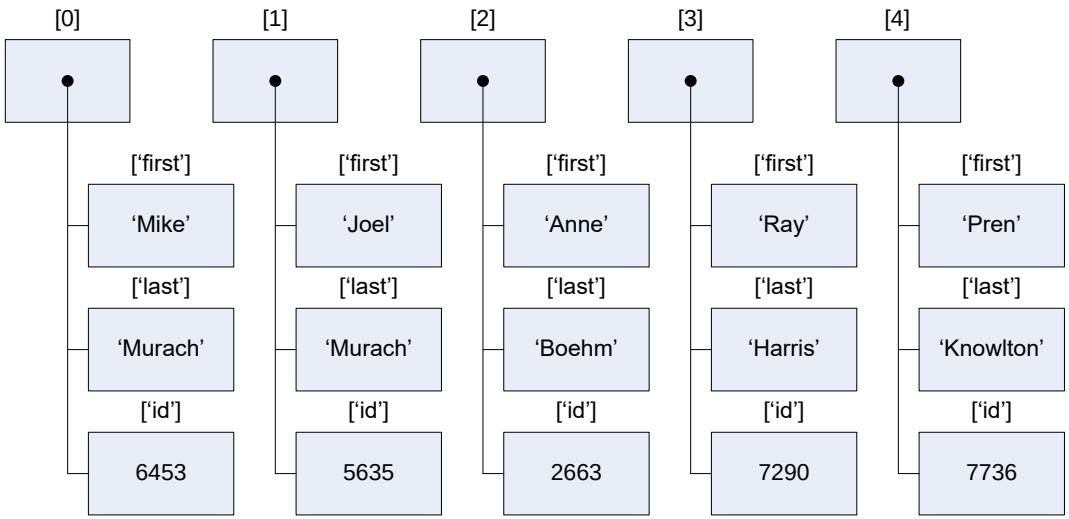
- Add Mary to group:

```
$group[] = ‘Mary’;
```

- Print out members of group:

```
foreach ($group as $member) {
  echo $member . ‘ ’;
}
```

A rectangular array of arrays



Suppose this array is the value of variable $rows

- Add new row:

```
$rows[] = array('first'=>'Mary','last'=>'Chen','id'=>8456);
```

- Print out rows, one way:

```
foreach($rows as $row) {
 echo "$row[first] $row[last], id= $row[id]";
}
```

Note we don't put quotes around array keys here– see pg. 323 either edition

Project 1: a website like Ch05's

- Note solution will be available Thursday for study, use for exam.
- MVC: all HTML-generating PHP (the view code) is in separate source files from controller code, itself in "index.php" files.
- All database access is handled in the model code, in the model directory.
  - All shared (between users) variable data is held in the database.
- View code is stored in the same directory as the index.php that controls it, unless it is view code that is included from code in more than one directory, like header.php and footer.php, in which case it is stored in the view directory.
- There is one top-level index.php that has links to the five major web apps of this website: for student operations and the various admin operations.
  - This index.php is not a "controller" because it does not use PHP code to interpret user input or access the database.
  - So it's OK that it has HTML in it.

Project 1 web apps

- Like the web app “product_catalog” in the website “ch05_guitar_shop”, there is an “pizza” directory, containing an index.php to control the actions for students, i.e. the student webapp. Views involved in this activity are in other files in this directory.
- Like product_manager in ch05_guitar_shop, there is a topping directory, containing an index.php to implement the topping actions for admins, i.e., the topping webapp. Views involved in this activity are in other files in this directory.
- Similarly, also like product_manager in ch05_guitar_shop, there is a user directory, containing an index.php to implement the user actions for admins. Views involved in this activity are in other files in this directory.
- Similarly for order and day, more admin actions, with index.php controllers.
- So we have five cooperating web apps in this pizza1 website
- Users can navigate the site with help of setup in header.php, with a <nav> that holds links to the site home page and the student web app.

Pizza1 Database Data

- Each student order causes inserts in the database, easily retrievable in future requests.
- Student order status can be read from the DB given the user id.
- Each admin action changes or reads database data.
- The retrieved database data (and other data) is held in variables only during a single request. This is an important part of the PHP execution model.
- Each request starts fresh getting needed DB data.
- This is important to make sure DB data isn't "stale".
- It also means the server only needs to allocate memory for database data for the duration of the request (maybe 50 ms).

Pizza1 User-private data

- The userid is like the current category in ch05_guitar_shop: it is specified by the user and not stored in the database.
- In the posted solution, the userid follows the user from request to request via a request parameter, like the current category in ch05_guitar_shop.
- It's OK if your solution requires the user to specify the userid each time, since we only recently covered this point properly.
  - But be sure you understand the userid handling of the solution, for the midterm exam.

Pizza1 Controllers

- The controllers are coded in the index.php files of each webapp directory: pizza, topping, user, order, day
- The controllers are the most central code, defining what the app does.
- The controller interprets user input and specifically takes the user “command” from the “action” parameter. This parameter could be named otherwise of course, but M&H sticks with “action” consistently through the book.
- All calls to the model functions are from the controllers, not the view code (MVC rule).
- The controller has code specific for an action value, and then either forwards to a view file (include …) or redirects back to itself, with needed info for that execution in parameters.
- Variable values persist through a PHP forward (include …), but not a redirect, since that entails another request-response cycle.

Pizza1 View Files

- View code will be stored in the same directory as the index.php that controls it, unless it is view code that is included from code in more than one directory, like header.php and footer.php, in which case it is stored in the view directory.
- View code in the app directories contains forms and links to capture user commands for that app.
  - A form has an action=URL, a link has href=URL
  - These URLs all point back to the controller: “index.php” or “.” or “”, by the MVC rule that each incoming request is received by a controller.
  - After the user clicks on the link or button, the controller will execute in a new request cycle, and the related user input is in request parameters.
- Thus the user stays with a certain app until they opt out by following a link provided by header.php, which is included in each app view file.

Pizza1 model code

- Each needed database action has a function in a file in the model directory.
- The function headers make up the database API, and their code is separate from the controller as expected in MVC.
- Although database actions can cause exceptions, no try { } catch {} is coded in model files.
- Instead, the *calling code in the controllers* has try {} around the model calls and a catch {} that forwards to database_error.php:

```
include ('../errors/database_error.php');
```

- The idea is that the controller better knows what to do when things go wrong than the model code itself.

Summary on Pizza1

This project is meant to be a model for a mid-size PHP project:

- It follows MVC.
- It uses the database for all shared, changeable data, and some shared unchanging data (pizza sizes for ex.)
- It uses only one level of directories holding PHP code
  - Multilevel source directories make includes difficult, usually requiring an “include path” setup.
- It uses shared header.php and footer.php and CSS to keep a consistent look across the website, and do common setup.
- It doesn’t use session variables, but another mid-size project might.
  - It shows how we can use URL parameters to transfer information (userid) from one request to the next.