WEBD 236

Web Information Systems Programming

Week 11

Todd Whittaker and Scott Sharkey
(sharkesc@franklin.edu)

FRANKLIN UNIVERSITY

Agenda

- This week’s expected outcomes
- This week’s topics
- This week’s homework
- Upcoming deadlines
- Questions and answers

Week 11 Outcomes

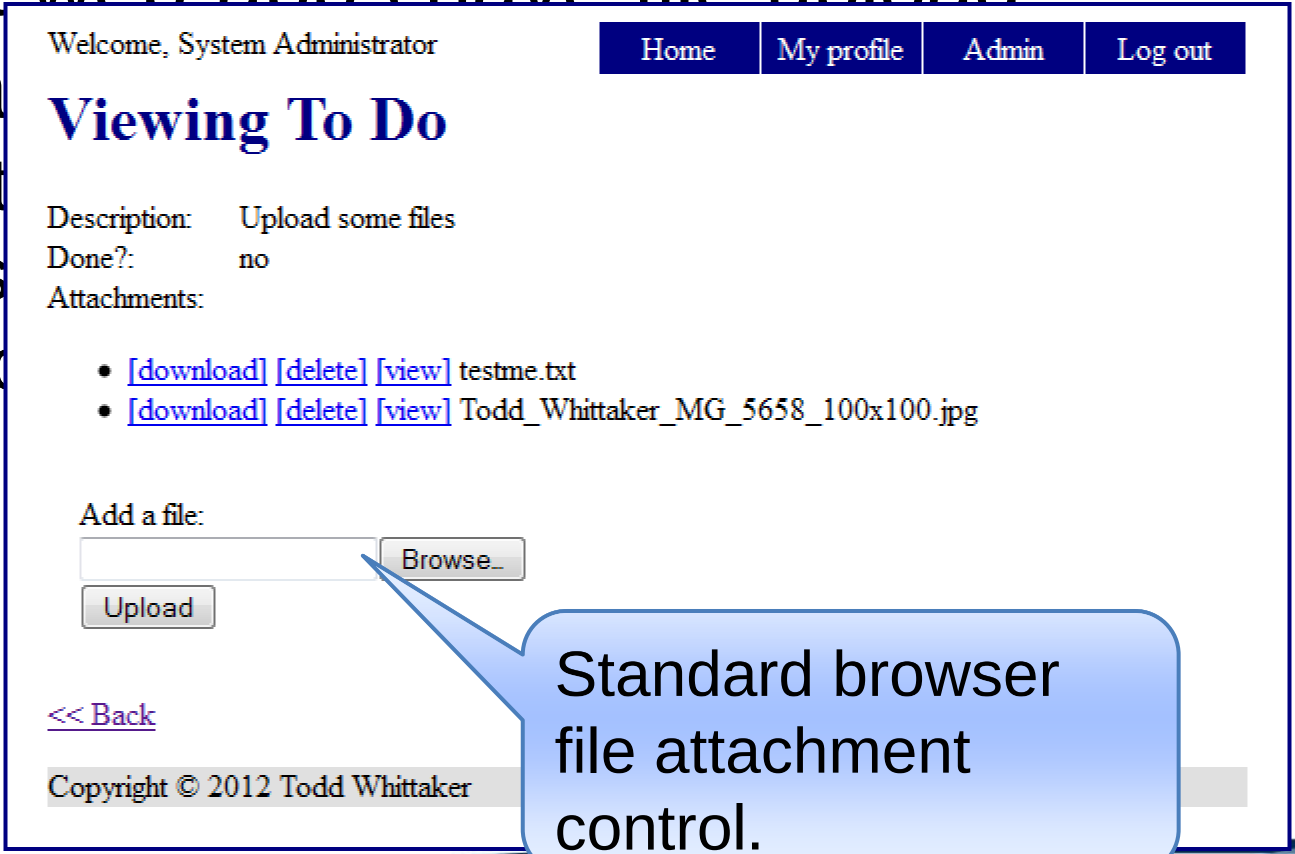
- Explore the security implications of file uploads
- Write code that receives, stores, processes, and transmits files uploaded via the web browser.
- List the advantages of using a web-framework for application development
- List and explain the typical features of a web-framework

File Uploads

- Most web-apps have file upload capabilities
  - Attachments to items
  - User profile pictures
  - Etc.

File Uploads

- Most web-apps have file upload capa
  - Att
  - Us
  - Etc



Welcome, System Administrator

Viewing To Do

Description: Upload some files
Done?: no
Attachments:

- [download] [delete] [view] testme.txt
- [download] [delete] [view] Todd_Whittaker_MG_5658_100x100.jpg

Add a file:

<< Back

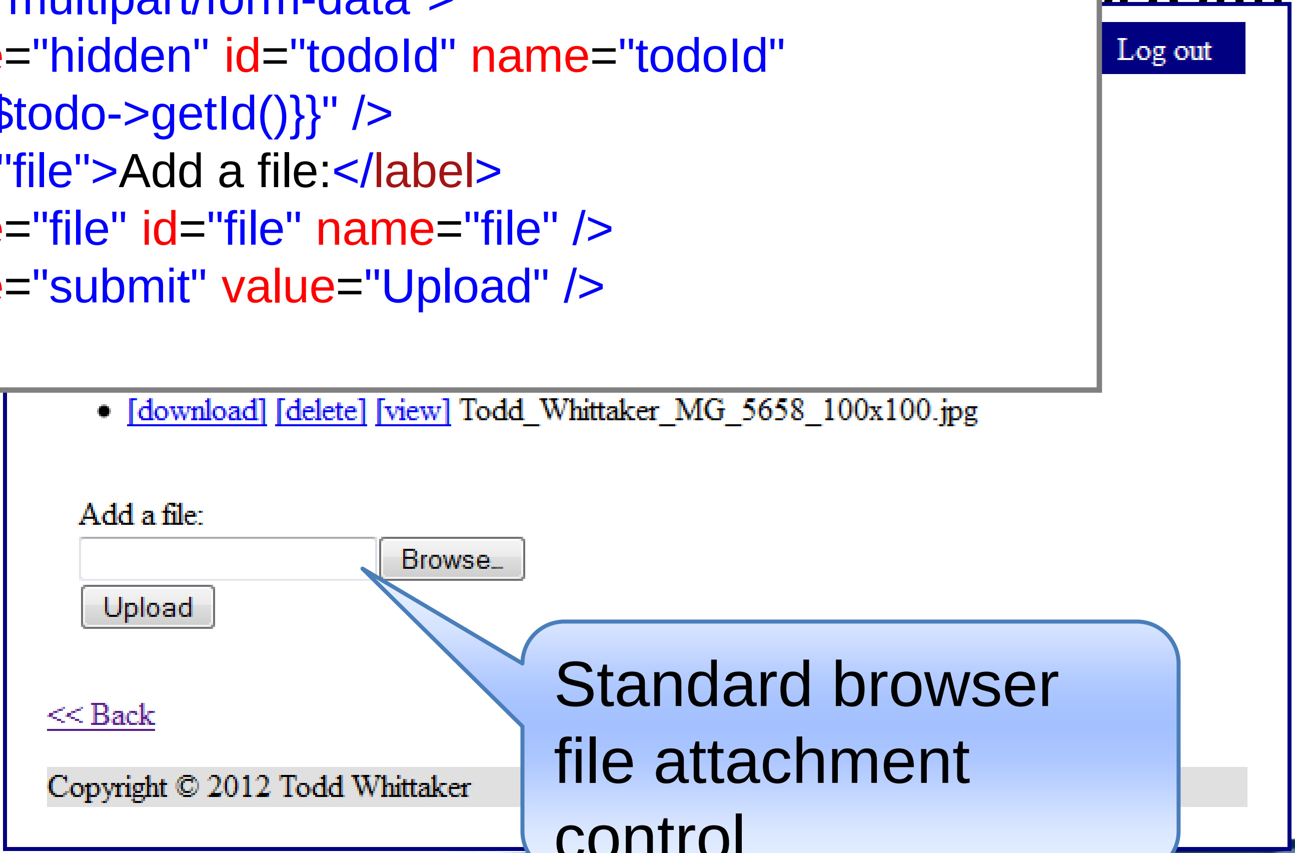
Copyright © 2012 Todd Whittaker

Standard browser file attachment control.

File Uploads

abilities

```
<form action="@@file/add@@" method="post"
    enctype="multipart/form-data">
  <input type="hidden" id="todoId" name="todoId"
    value="{{$todo->getId()}}" />
  <label for="file">Add a file:</label>
  <input type="file" id="file" name="file" />
  <input type="submit" value="Upload" />
</form>
```



- [download] [delete] [view] Todd_Whittaker_MG_5658_100x100.jpg

Add a file:

Copyright © 2012 Todd Whittaker

Standard browser file attachment control.

File Uploads

- On the server side, the $_FILES super-global contains information about all uploaded files.
  - Simplest code:

```
<html>
    <head><title>File upload</title></head>
    <body>
        <form action="fileupload.php" method="post“
            enctype="multipart/form-data">
            <input type="file" name="myFile" />
            <input type="submit" value="Upload!" />
        </form>
        <pre><?php print_r($_FILES, true); ?></pre>
    </body>
</html>
```

File Uploads

- On the server side, the $_FILES super-
  g
  u

```
<html>
    <he
    <bo
        <
```

```
Array
(
  [myFile] => Array
  (
    [name] => Scott_Sharkey_MG_5658_100x100.jpg
    [type] => image/jpeg
    [tmp_name] => C:\xampp\tmp\phpD476.tmp
    [error] => 0
    [size] => 6632
  )
)
```

```
        </form>
        <pre><?php print_r($_FILES, true); ?></pre>
    </body>
</html>
```

File Uploads

- On the server side
  g
  u

```
Array
(
  [myFile] => Array
  (
    [name] => Scott_Sharkey_MG_5658_100x100.jpg
    [type] => image/jpeg
    [tmp_name] => C:\xampp\tmp\phpD476.tmp
    [error] => 0
    [size] => 6632
  )
)
```

MIME (Multipurpose Internet Mail Extensions) type. Unchecked, provided by browser.

File temporarily held at this location.

Max file upload size controlled by php.ini.

```
<html>
  <he
  <bo
    <

    </f
    <p                              ); ?></pre>
  </bod
</html>
```

File Uploads

- Moving the file to the right location

```
foreach ($_FILES as $file) {
    $path = getcwd() . DIRECTORY_SEPARATOR . 'uploads' .
        DIRECTORY_SEPARATOR;
    $success = move_uploaded_file($file['tmp_name'], $path .
        $file['name']);
    if (!$success) {
        die("Problem moving file.");
    }
}
```

File Uploads

- Moving the file to the right location

```
foreach ($_FILES as $file) {
    $path = getcwd() . DIRECTORY_SEPARATOR . 'uploads' .
        DIRECTORY_SEPARATOR;
    $success = move_uploaded_file($file['tmp_name'], $path .
        $file['name']);
    if (!$success) {
        die("Problem moving file.");
    }
}
```

What is wrong with this code from a security point of view?

File Uploads

- Moving the file to the right location
  - Never, *ever* trust user input of any kind.
  - What if the user somehow changed the original file name to be “..\index.php”?
- Several solutions:
  - Generate your own file name
  - Sanitize the existing name somehow

File Uploads

- Sanitizing the existing file name

```
function sanitizeFileName($str) {
    // get rid of consecutive dots
    $str = preg_replace('/\.\.+/', '.', $str);
    // get rid of trailing dots
    $str = preg_replace('/\.+$/', '', $str);
    // get rid of leading dots
    $str = preg_replace('/^\.+/', '', $str);
    // get rid of other nasty characters
    return preg_replace('/[^0-9a-zA-Z_\.-]/', '_', $str);
}
```

File Uploads

- Sanitizing the existing file name

```
function sanitizeFileName($str) {
    // get rid of consecutive dots
    $str = preg_replace('/\.\.+/', '.', $str);
    // get rid of trailing dots
    $str = preg_replace('/\.+$/', '', $str);
    // get rid of leading dots
    $str = preg_replace('/^\.+/', '', $s    r);
    // get rid of other nasty characters
    return preg_replace('/[^0-9a-zA-Z_\.          tr);
}
```

Even if you choose to generate your own file name, but yet display this one to the user, you should sanitize. Avoids injection.

File Uploads

- Generating a new file name

```
function generateName($dir) {
    do {
        $name = uniqid('upload');
    } while (is_file($dir . DIRECTORY_SEPARATOR . $name));
    return $name;
}
```

File Uploads

- Generating a new file name

```
function generateName($dir) {
    do {
        $name = uniqid('upload');
    } while (is_file($dir . DIRECTOR SEPARATOR . $name));
    return $name;
}
```

This is the prefix string prepended onto the 13 character hex identifier returned by uniqid. Really useful when multiple servers could be generating unique IDs concurrently.

File Uploads

- Don’t we likely want to store file references in the database somewhere?
  - Two approaches:
    - Store file metadata in the DB, file contents on disk
    - Store metadata and contents in the DB

File Uploads

- Don’t we likely want to store file references in the database somewhere?
  - Two approaches:
    - Store file metadata in the DB, file contents on disk
    - Store metadata and contents in the DB

- Advantages: smaller DB, more easily backed up
- Disadvantages: Can “orphan” files if rows are deleted, but not the disk files (cascading deletes).

File Uploads

- Don’t we likely want to store file references in the database somewhere?
  - Two approaches:
    - Store file metadata in the DB, file contents on disk
    - Store metadata and contents in the DB

File contents become BLOBs

- Advantages: No orphaned files
- Disadvantages: Large, unwieldy databases; ETL is more difficult.

File Uploads

- Don’t we likely want to store file references in the database somewhere?
  - Two approaches:
    - Store file metadata in the DB, file contents on disk
    - Store metadata and contents in the DB

We will choose option 1. Would need to periodically clean the uploads directory to get rid of orphans.

File Uploads

- Let’s create a class to encapsulate this.

```
class UploadDir {
    private $dir;
    function __construct($dir = 'uploads') {
        $this -> dir = getcwd() . DIRECTORY_SEPARATOR . $dir;
        if (!is_dir($this -> dir)) {
            mkdir($this -> dir);
        }
    }
    private static function sanitizeFileName($str) {
        $str = preg_replace('/\.\.+/', '.', $str);
        $str = preg_replace('/\.+$/', '', $str);
        $str = preg_replace('/^\.+/', '', $str);
        return preg_replace('/[^0-9a-zA-Z_\.-]/', '_', $str);
    }
```

File Uploads

- Let’s create a class to encapsulate this.

```
private function generateName() {
    do {
        $name = uniqid('upload');
    } while (is_file($this->dir . DIRECTORY_SEPARATOR .
        $name));
    return $name;
}
public function getAllUploads() {
    $result = array();
    foreach ($_FILES as $key => $meta) {
        $result[] = $this -> getUpload($key);
    }
    return $result;
}
```

File Uploads

- Let’s create a class to encapsulate this.

```
public function getUpload($key) {
    $file = null;
    if (isset($_FILES[$key])) {
        $tmp_name = $_FILES[$key]['tmp_name'];
        $nameOnDisk = $this -> generateName();
        $path = $this -> dir . DIRECTORY_SEPARATOR .
            $nameOnDisk;
        $success = move_uploaded_file($tmp_name, $path);
        if (!$success) {
            throw new Exception("Problem with file.");
        }
        // ...continued...
```

File Uploads

- Let’s create a class to encapsulate this.

```
    public function getUpload($key) {
            // ...continued...
            $params = array(
               'dir' => $this->dir,
               'nameOnDisk' => $nameOnDisk,
               'origName' => self::sanitizeFileName(
                  $_FILES[$key]['name']),
               'type' => $_FILES[$key]['type'],
               'size' => $_FILES[$key]['size']);
            $file = new File($params);
         }
         return $file;
    }
} // end class UploadDir
```

File Uploads

- Using UploadDir

```
$dir = new UploadDir('files');
$files = $dir -> getAllUploads();
foreach ($files as $file) {
    $file -> insert();
}
```

Looks like a File object subclasses Model.

File Uploads

- DDL for files table

```
CREATE TABLE file (
    id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
    dir VARCHAR(150) NOT NULL,
    origName VARCHAR(50) NOT NULL,
    nameOnDisk VARCHAR(50) NOT NULL,
    type VARCHAR(50) NOT NULL,
    size INTEGER NOT NULL,
    todoId INTEGER NOT NULL,
    FOREIGN KEY(todoId) REFERENCES todo(id) ON DELETE CASCADE
)
```

File Uploads

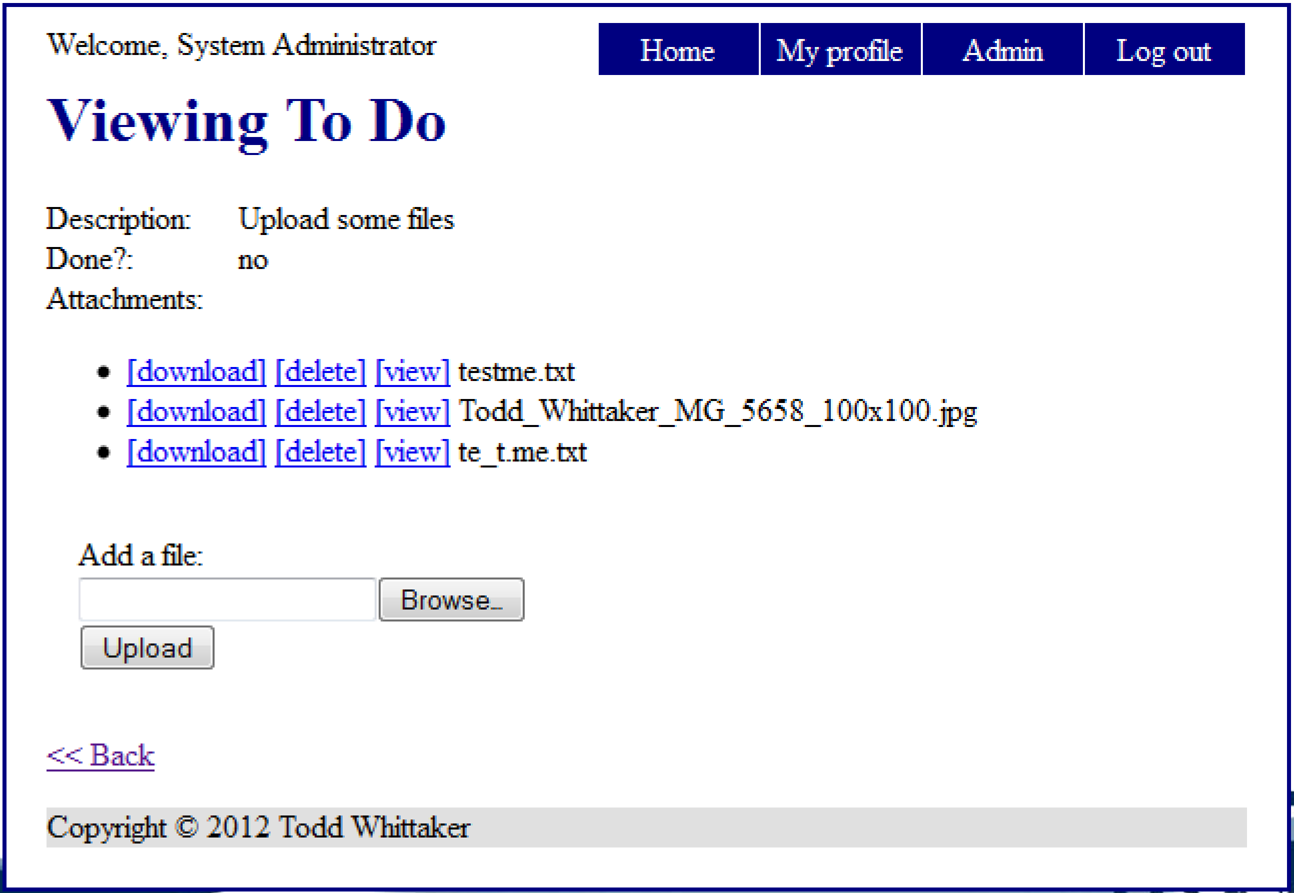
- DDL for files table

```
CREATE TABLE file (
    id INTEGER NOT NULL PRIMARY KEY A
    dir VARCHAR(150) NOT NULL,
    origName VARCHAR(50) NOT NULL,
    nameOnDisk VARCHAR(50) NOT NULL
    type VARCHAR(50) NOT NULL,
    size INTEGER NOT NULL,
    todoId INTEGER NOT NULL,
    FOREIGN KEY(todoId) REFERENCES todo(id) ON DELETE CASCADE
)
```

Context is a one-to-many relationship with a “ToDo” object. Your schema may vary.

File Uploads

- Goal: attach files to ToDo



Welcome, System Administrator

Viewing To Do

Description: Upload some files
Done?: no
Attachments:

- [download] [delete] [view] testme.txt
- [download] [delete] [view] Todd_Whittaker_MG_5658_100x100.jpg
- [download] [delete] [view] te_t.me.txt

Add a file:

<< Back

Copyright © 2012 Todd Whittaker

File Uploads

- Lib/File.inc class

```
class File extends Model {

   private $dir;
   private $origName;
   private $nameOnDisk;
   private $type;
   private $size;
   private $todoId;
```

File Uploads

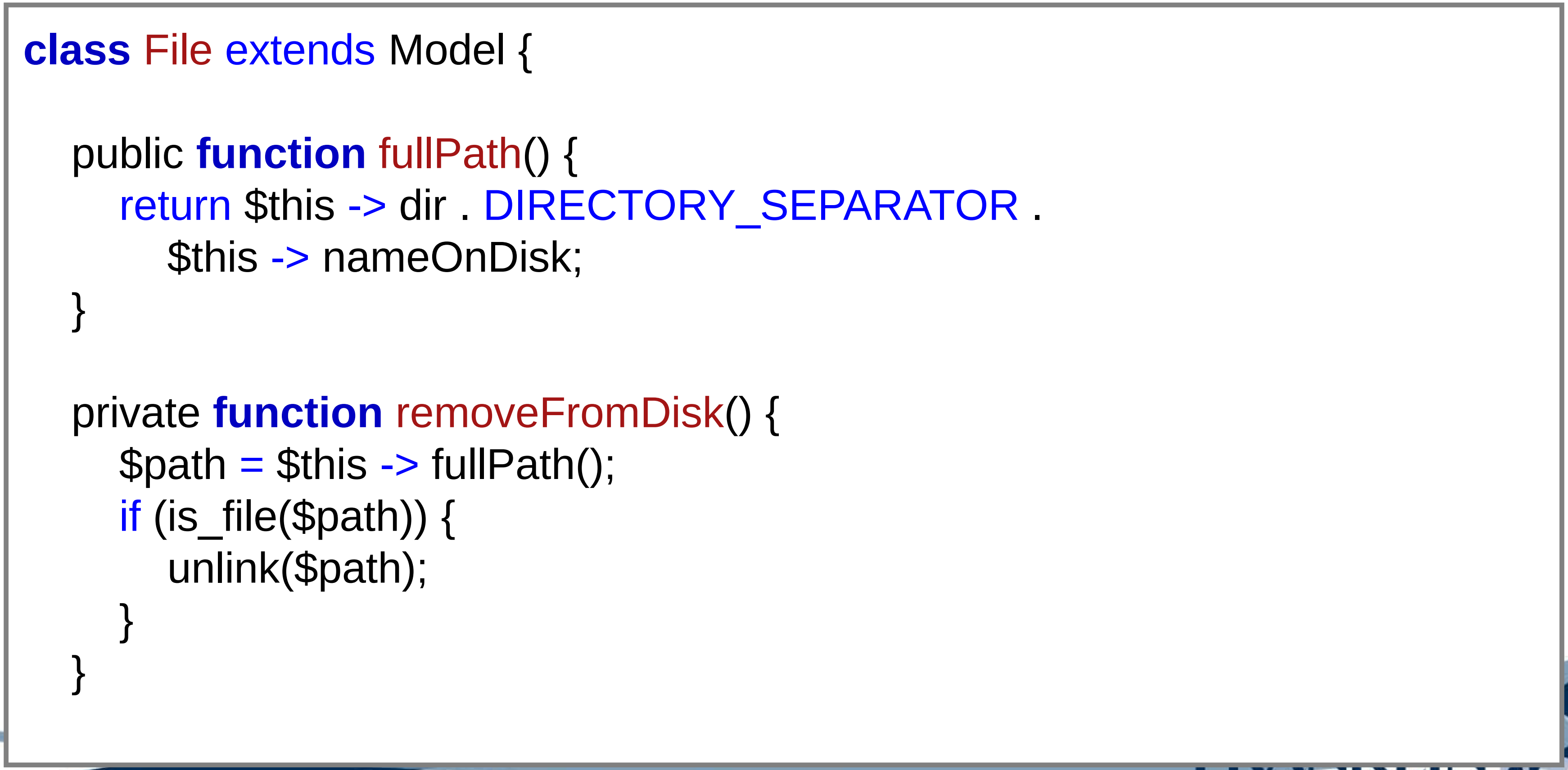
- Lib/File.inc class

```
class File extends Model {

    public function __construct($fields) {
        parent::__construct($fields);
        $this -> setDir(safeParam($fields, 'dir'));
        $this -> setOrigName(safeParam($fields, 'origName'));
        $this -> setNameOnDisk(safeParam($fields,
            'nameOnDisk'));
        $this -> setType(safeParam($fields, 'type'));
        $this -> setSize(safeParam($fields, 'size'));
        $this -> setTodoId(safeParam($fields, 'todoId'));
    }
```

File Uploads

- Lib/File.inc class



```
class File extends Model {

    public function fullPath() {
        return $this -> dir . DIRECTORY_SEPARATOR .
            $this -> nameOnDisk;
    }

    private function removeFromDisk() {
        $path = $this -> fullPath();
        if (is_file($path)) {
            unlink($path);
        }
    }
```

File Uploads

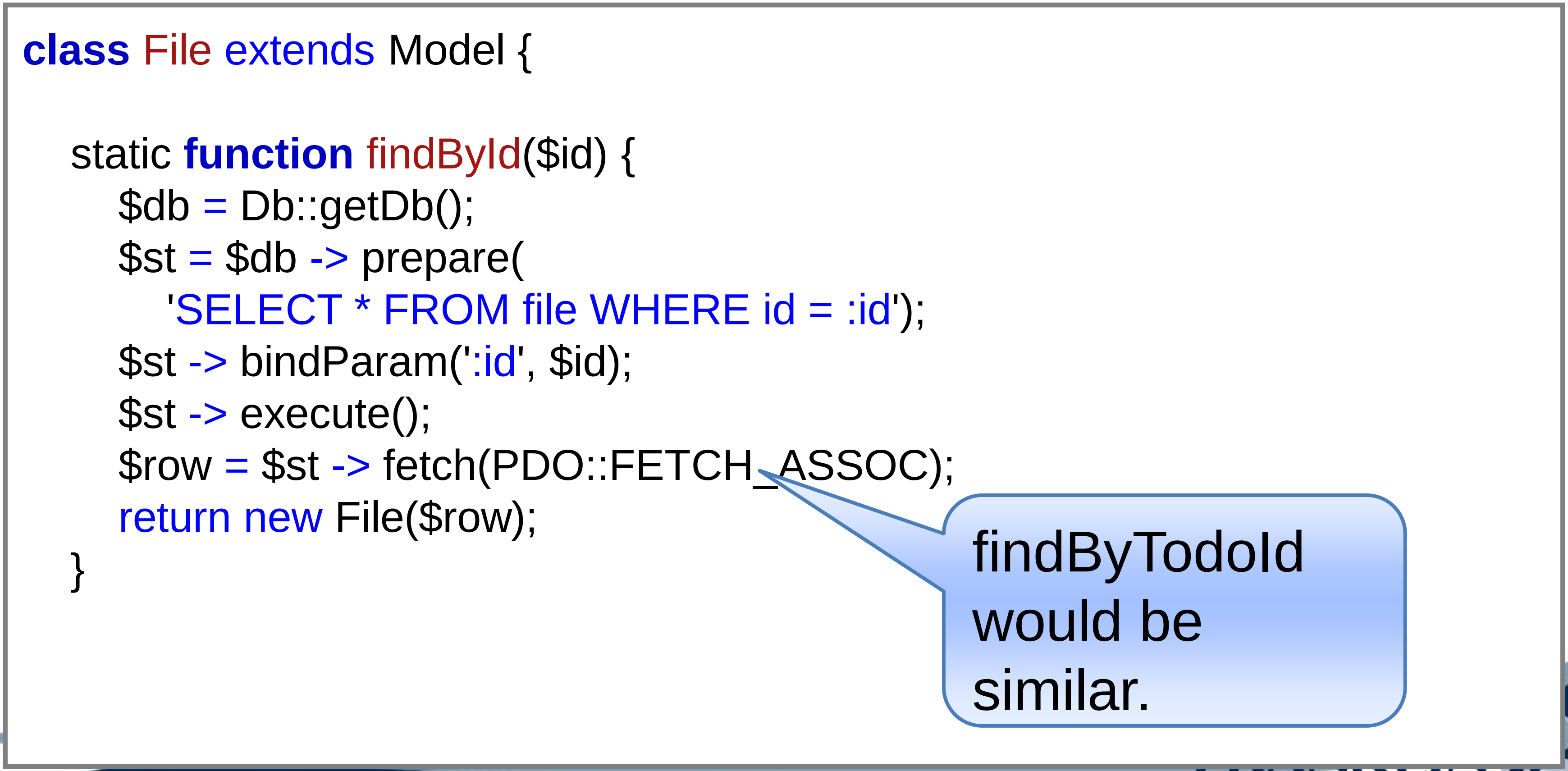
- Lib/File.inc class

```
class File extends Model {

  private function moveOnDisk($to) {
    $old = $this -> fullPath();
    $new = $this -> dir . DIRECTORY_SEPARATOR . $to;
    if (is_file($old)) {
      rename($old, $new);
    }
    $this -> nameOnDisk = $to;
    return $this;
  }
```

File Uploads

- Lib/File.inc class



```
class File extends Model {

    static function findById($id) {
        $db = Db::getDb();
        $st = $db -> prepare(
            'SELECT * FROM file WHERE id = :id');
        $st -> bindParam(':id', $id);
        $st -> execute();
        $row = $st -> fetch(PDO::FETCH_ASSOC);
        return new File($row);
    }
```

findByTodoId would be similar.

File Uploads

- Lib/File.inc class

insert and update are fairly standard.

```
class File extends Model {

    function delete() {
      $db = Db::getDb();
      $statement = $db -> prepare(
         "DELETE FROM file WHERE id = :id");
      $statement -> bindParam(':id', $this -> id);
      $statement -> execute();
      $this->removeFromDisk();
   }
}
```

File Uploads

- controllers/file.inc
  - Need to handle create, delete, download and view capabilities

File Uploads

- controllers/file.inc

```
// uploading a file
function post_add($params) {
    Authenticator::instance() -> ensure('edit_todo');
    $todoId = safeParam($_REQUEST, 'todoId', false);
    $todo = Todo::findById($todoId);
    if (!$todo) {
        die("No todo with that ID found");
    }
    $dir = new UploadDir();
    $file = $dir -> getUpload('file');
    $file -> setTodoId($todo -> getId());
    $file -> insert();
    redirectRelative("todo/view/{$todo->getId()}");
}
```

File Uploads

- controllers/file.inc

```
// deleting a file

function get_delete($params) {
    Authenticator::instance() -> ensure('edit_todo');
    $fileId = safeParam($params, 0);
    $todoId = safeParam($params, 1);
    $file = File::findById($fileId);
    $file -> delete();
    redirectRelative("todo/view/$todoId");
}
```

Needs a second parameter of what Todo to redirect to after deleting.

File Uploads

- controllers/file.inc

```
// download a file
function get_download($params) {
    Authenticator::instance() -> ensure('view_todo');
    $fileId = safeParam($params, 0);
    $file = File::findById($fileId);
    header('Content-Description: File Transfer');
    header('Content-Type: ' . $file -> getType());
    header('Content-Disposition: attachment; filename=' .
        $file -> getOrigName());
    header('Content-Transfer-Encoding: binary');
    // ...continued...
```

app/file/download/1 will trigger a download

File Uploads

- controllers/file.inc

```
    // ...continued...
    header('Cache-Control: must-revalidate');
    header('Pragma: public');
    header('Content-Length: ' . $file -> getSize());
    ob_clean();
    flush();
    readfile($file -> fullPath());
    exit;
}
```

File Uploads

- controllers/file.inc

```
// view a file (inline, not "download")
function get_view($params) {
    Authenticator::instance() -> ensure('view_todo');
    $file = File::findById(safeParam($params, 0));
    header('Last-Modified: ' . date('r'));
    header('Accept-Ranges: bytes');
    header('Content-Length: ' . $file -> getSize());
    header('Content-Type: ' . $file -> getType());
    header('Content-Disposition: inline; filename=' .
        $file -> getOrigName());
    ob_clean();
    flush();
    readfile($file -> fullPath());
    exit;
}
```

app/file/view/1 is the URL for viewing inline.

File Uploads

- Let’s say a user uploads a profile picture
  - Is there any security implication for permitting that to be viewed?

File Uploads

- Let’s say a user uploads a profile picture
  - Is there any security implication for permitting that to be viewed?

Absolutely! http://news.cnet.com/JPEG-exploit-could-beat-antivirus-software/2100-7349_3-5388633.html

File Uploads

- Let’s say a user uploads a profile picture
  - Is there any security implication for permitting that to be viewed?
    - Browsers have security vulnerabilities.
    - JPEG, GIF, PNG rendering libraries have had vulnerabilities.
    - A carefully crafted picture, sent back to the browser, could contain malware.
  - Solution: Never, *ever* trust user input

File Uploads

- Let’s say a user uploads a profile picture
  - Never, *ever* trust user input of any kind.
    - Resample the image using PHP functions (pages 760-761 of your textbook).
    - Store and transmit the resampled image.

Show me the code!

- Full source code for the upload-enabled “Todo” application is found in the standard location: http://cs.franklin.edu/~sharkesc/webd236/

Other file-related functions

- A list of functions worth exploring
  - is_file($path): returns true if $path is a file
  - is_dir($path): returns true if $path points to a dir
  - file_exists($path): is_file($path) || is_dir($path)
  - getcwd(): current working directory
  - scandir($path): returns an array of files in $path

Other file-related functions

- A list of functions worth exploring
  - file($name): returns an array of file contents, one entry per line.
  - file_get_contents($name): returns one big string containing all data from the file.
  - read_file($name): dumps the entire contents of the file to the output stream.
  - file_put_contents($name, $data): writes the data to the file (overwriting by default).

Other file-related functions

- A list of functions worth exploring
  - fopen($path, $mode): opens a file, returning a “handle.”
  - feof($handle): returns true if at end of file.
  - fclose($handle): closes a file handle.
  - fread($handle, $length): reads bytes from the file.
  - fwrite($handle, $data): writes bytes to the file.
  - fgets($handle): read one line from the file.

Other file-related functions

- A list of functions worth exploring
  - copy($old, $new): copies a file
  - rename($old, $new): renames a file
  - unlink($name): deletes a file
  - fgetcsv($handle): reads in one line of CSV, returning an array of the data. Useful for importing data. First line has “keys,” usually.
  - fputcsv($handle, $array): writes an array to file as CSV. Useful for exporting data.

Not covered today

- Image manipulation (pgs 756-763)

Application Frameworks

- You may have noticed
  - Writing the same kind of code repeatedly
    - Models: getters, setters, findByX, insert, delete, update.
    - Controllers: retrieve parameters, validate, check permissions, update model, render a page or redirect
    - Views: using the same header/footer, printing variables
  - We have built a small application framework that takes some of the drudgery out

Application Frameworks

- Application Frameworks
  - Provide the infrastructure for building apps, so you can concentrate on the problem
    - Routing/dispatching
    - Flexible MVC
    - Caching
    - Localization
    - Validation/sanitization
    - Security

All designed to work together, the foundations for any app. A production app framework provides much more.

Application Frameworks

- “Convention over configuration”
  - – Frameworks impose a way of doing things
    - Ruby/Rails vs. Java/JEE
  - – Examples:
    - Model class names are singular and camel case (e.g. BlogPost), but the table in the database is plural and underscored (e.g. blog_posts).
    - Routes like /app/user/edit/5 get mapped to a class called UserController, calling the method edit($id), and 5 becomes the value of $id.

Many more features

- Additional features:
  - ACL based authorization
  - Object relational mapping of one-to-many (hasMany) and many-to-many (hasAndBelongsToMany) relationships
  - Code generation from an existing DB
  - Email components
  - Pagination, cookies, sessions, security, etc.

Learning curve

- Learning a framework is hard
  - – Resources
    - Online books and tutorials:
    - Free ebooks:
  - – The payoff in development time is huge.

Other frameworks

- PHP has many frameworks
  - E.g. Laravel, Cake, Yii, CodeIgniter, Zend, Symfony
  - Each has a different philosophy (monolithic vs. plugin-based) and provides different features.
- And several templating languages
  - E.g. Smarty, Dwoo, Rain

Lab 4

- You will be again modifying the forum application
  - Add Mini-markdown to questions and answers
  - Add RBAC (Users, Posters, Moderators, Administrators)
  - Attach files to questions
  - Bonus: send “forgot my password” email

Upcoming Deadlines

- Readings for next week
  - Chapter 24 in *PHP and MySQL*
- Assignments

  Lab 4 due end of week 12

  Final due end of week 12

General Q & A

- Questions?
- Comments?
- Concerns?