Convert Static Site to CodeIgniter

tutorial #tutorial02c

James L. Parry
B.C. Institute of Technology

Tutorial Goals

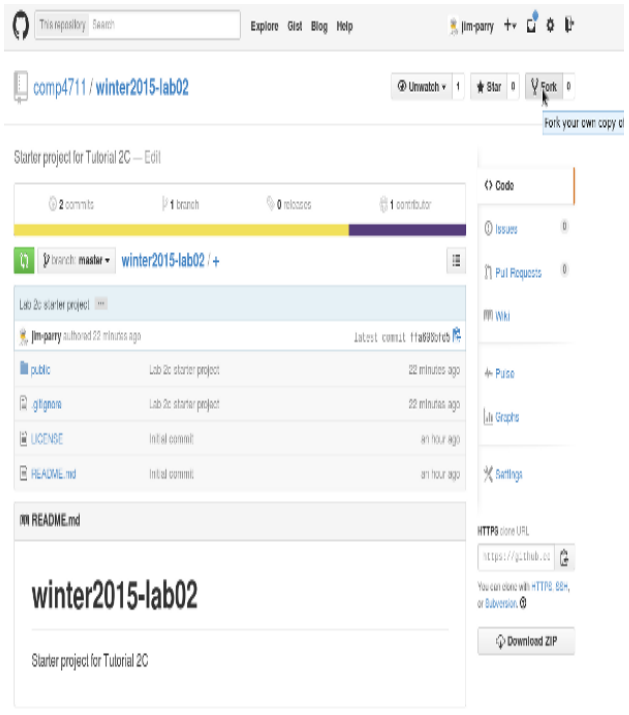
This tutorial will walk you through converting a static HTML website into a CodeIgniter webapp.

Suggestion: you may want to skim the slideshow first, before working your way through it.

Get The Starter

Fork the lab starter project, and clone it to your local system.

You can clone it to any folder inside your htdocs, as long as you have a virtual host mapping to it for testing. I will assume "comp4711.local" for this writeup.



comp4711 / winter2015-lab02

Starter project for Tutorial 2C — Edit

winter2015-lab02 / +

Lab 2c starter project

jim-parry authored 22 minutes ago latest commit ffa698bfdb

public	Lab 2c starter project	22 minutes ago
.gitignore	Lab 2c starter project	22 minutes ago
LICENSE	Initial commit	an hour ago
README.md	Initial commit	an hour ago

README.md

winter2015-lab02

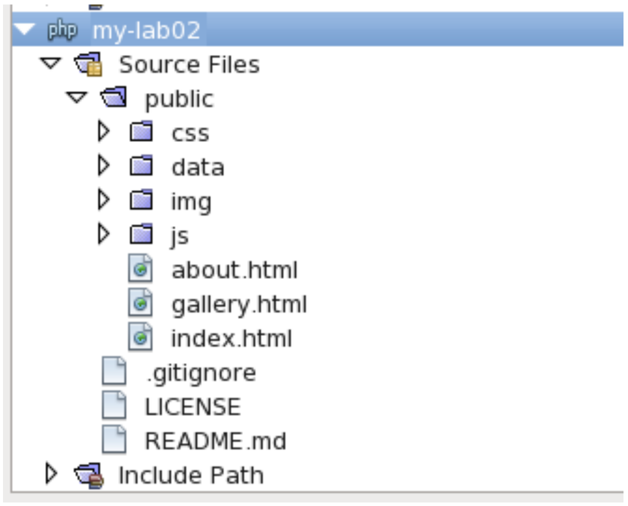
Starter project for Tutorial 2C

You can clone with HTTPS, SSH, or Subversion.

What Do You Have?

I cloned mine to a "my-lab02" project/folder on my system. Substitute you choices whenever I refer to my folder.

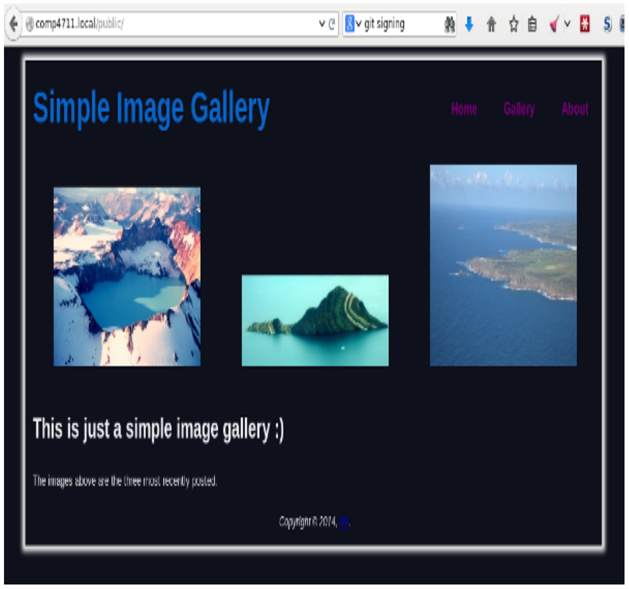
You should see a "public" folder, which contains an entire static website.



```
my-lab02
  Source Files
    public
      css
      data
      img
      js
      about.html
      gallery.html
      index.html
    .gitignore
    LICENSE
    README.md
  Include Path
```

What Does It Look Like?

Assuming that your new project is mapped to "comp4711.local", then the URL "comp4711.local/public" should give you the homepage of the static site, shown right.



This is just a simple image gallery :)

The images above are the three most recently posted.

Copyright © 2014,

How Do We Make This Into a Webapp?

The basic steps we will follow are shown to the right.

This isn't the only way to do this job, but is one that makes sense. After each stage of the conversion, we should still have a working site.

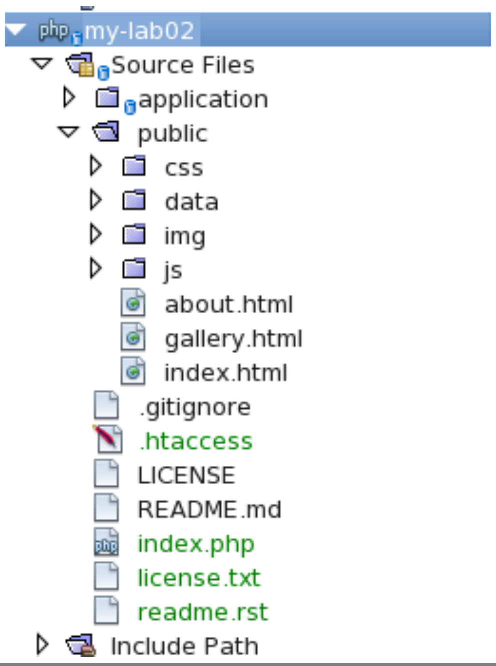
Er, it will behave like it is working, even though it won't be a proper webapp until we are done.

1. Copy our CI starter into our project
2. Do a trivial conversion
3. Setup base controller and view template
4. Fix the pages to use these
5. Add a model
6. Revisit the pages

Copy Our CI Starter

Copy the contents of your CodeIgniter starter project (from tutorial 2B) into your project folder.

Now, asking for comp4711.local in your browser should give you the "Welcome to CodeIgniter!" page.

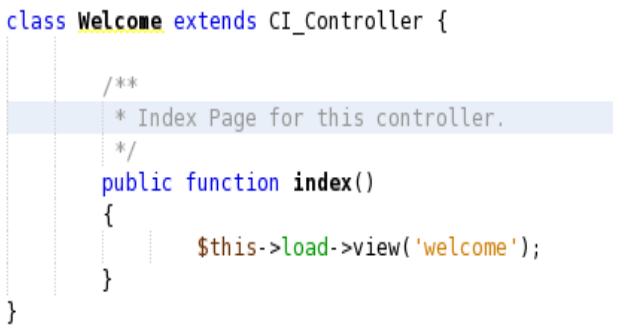


```
my-lab02
  Source Files
    application
    public
      css
      data
      img
      js
      about.html
      gallery.html
      index.html
    .gitignore
    .htaccess
    LICENSE
    README.md
    index.php
    license.txt
    readme.rst
  Include Path
```

Create Trivial Controllers

Our static site has 3 pages, index.html, gallery.html and about.html. We can make trivial controllers for each of these, which simply load the corresponding "view".

Fix application/controllers/Welcome.php (your homepage) to load the "welcome" view. Create similar controllers Gallery.php and About.php, which load the "gallery" and "about" views, respectively.



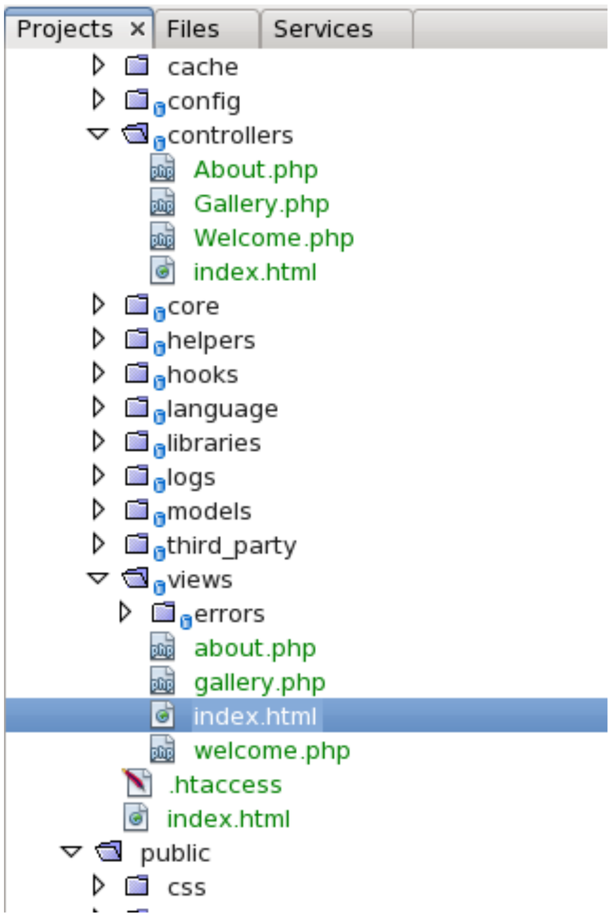
```
class Welcome extends CI_Controller {

    /**
     * Index Page for this controller.
     */
    public function index()
    {
        $this->load->view('welcome');
    }
}
```

Create Trivial Views

We can make trivial views for each of the static pages, by copying the static html files to our application/views folder, and renaming them to have a ".php" extension. Note that index.html would become welcome.php.

You may have to rename these using your file manager, if your IDE won't let you change the extension.

We no longer need the views/welcome_message.php.

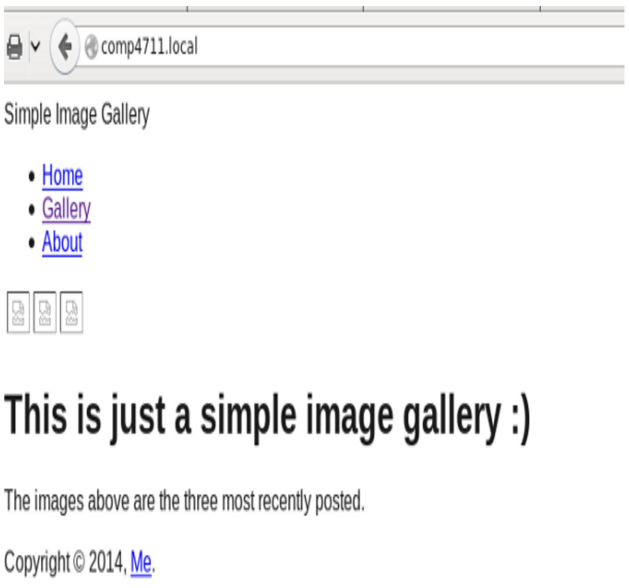


```
cache
config
controllers
    About.php
    Gallery.php
    Welcome.php
    index.html
core
helpers
hooks
language
libraries
logs
models
third_party
views
    errors
    about.php
    gallery.php
    index.html
    welcome.php
.htaccess
index.html
public
    css
```

We Now Have a Trivial Site

Checkout your site now, using the comp4711.local URL.

Oops, our image links are broken!



comp4711.local

Simple Image Gallery

- Home
- Gallery
- About

This is just a simple image gallery :)

The images above are the three most recently posted.

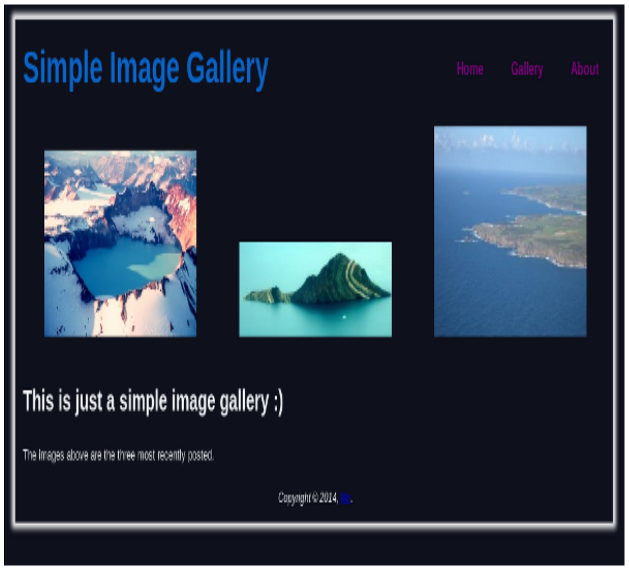
Copyright © 2014, Me.

Fix Our Image Links

Examining views/welcome.php, you can see that the image references no longer work. We can solve this by copying/moving those folders up one level, alongside "applications".

A better practice is to keep your "assets" (like these) in a separate folder, which could be called "public" or "assets". We can address the better practice in a later refactoring.

For now, let's copy the folders up one level.



Simple Image Gallery

Home Gallery About

This is just a simple image gallery :)

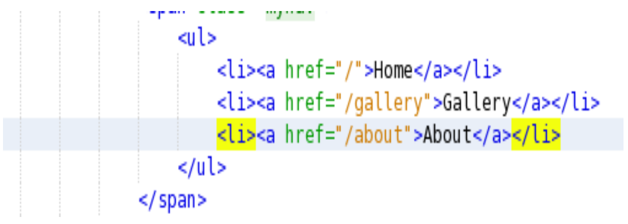
The images above are the three most recently posted.

Copyright © 2014, Me.

Fix Our Menu

Yay, our site *looks* better, but what about the menu? It still refers to the original HTML files. Fix this by changing the menu links to refer to our CI controllers instead. Apply this change to each of our three views.

Notice that the homepage is referred to as just "/", and that each of the other references start with a slash.



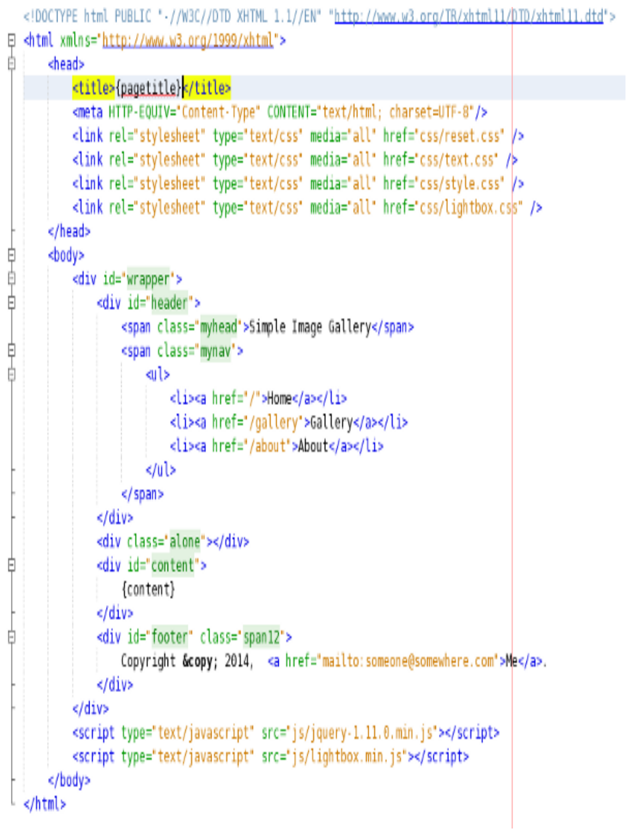
```
        <ul>
            <li><a href="/">Home</a></li>
            <li><a href="/gallery">Gallery</a></li>
            <li><a href="/about">About</a></li>
        </ul>
    </span>
```

Build a Master View Template

Eliminate duplicate code in our views by extracting the common opening and closing HTML into views/template.php.

Let's do that, and add a placeholder, "{content}", which will get substituted appropriately in our next tutorial step. Add another placeholder, "{pagetitle}", for the page title.

I would do this by copying the about view, renaming it to "template.php", and then zapping its middle.

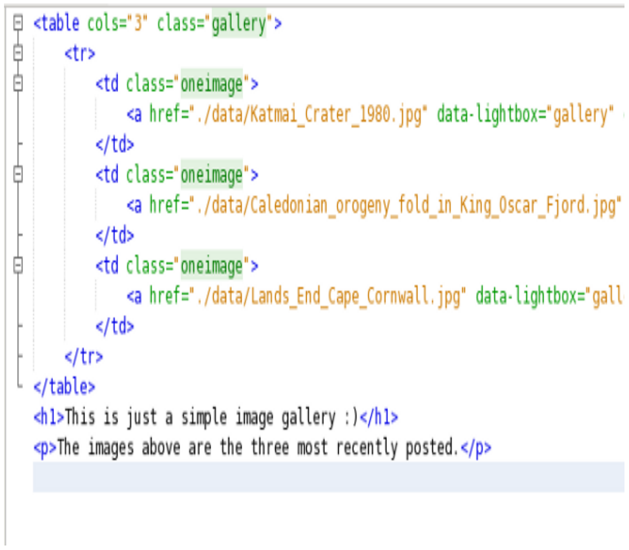


```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <title>{pagetitle}</title>
        <meta HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=UTF-8"/>
        <link rel="stylesheet" type="text/css" media="all" href="css/reset.css" />
        <link rel="stylesheet" type="text/css" media="all" href="css/text.css" />
        <link rel="stylesheet" type="text/css" media="all" href="css/style.css" />
        <link rel="stylesheet" type="text/css" media="all" href="css/lightbox.css" />
    </head>
    <body>
        <div id="wrapper">
            <div id="header">
                <span class="myhead">Simple Image Gallery</span>
                <span class="mynav">
                    <ul>
                        <li><a href="/">Home</a></li>
                        <li><a href="/gallery">Gallery</a></li>
                        <li><a href="/about">About</a></li>
                    </ul>
                </span>
            </div>
            <div class="alone"></div>
            <div id="content">
                {content}
            </div>
            <div id="footer" class="span12">
                Copyright &copy; 2014,  <a href="mailto:someone@somewhere.com">Me</a>.
            </div>
        </div>
        <script type="text/javascript" src="js/jquery-1.11.0.min.js"></script>
        <script type="text/javascript" src="js/lightbox.min.js"></script>
    </body>
</html>
```

Simplify Our Views

We can now eliminate the common opening and closing HTML from our three original views, leaving just the "meat".

Our revised views/welcome.php is shown to the right.

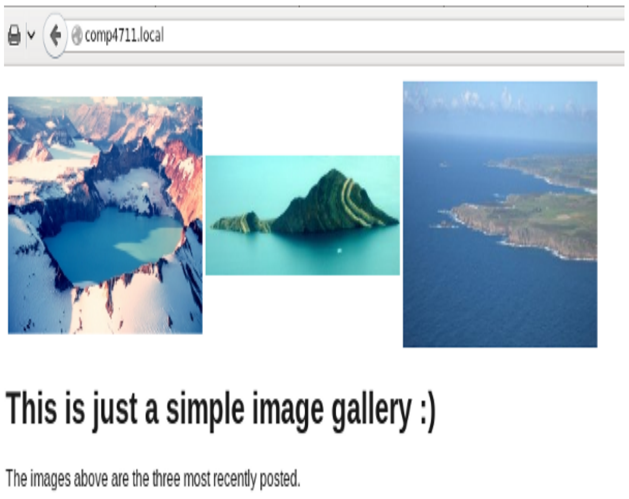


```
<table cols="3" class="gallery">
    <tr>
        <td class="oneimage">
            <a href="./data/Katmai_Crater_1980.jpg" data-lightbox="gallery"
        </td>
        <td class="oneimage">
            <a href="./data/Caledonian_orogeny_fold_in_King_Oscar_Fjord.jpg"
        </td>
        <td class="oneimage">
            <a href="./data/Lands_End_Cape_Cornwall.jpg" data-lightbox="gall
        </td>
    </tr>
</table>
<h1>This is just a simple image gallery :)</h1>
<p>The images above are the three most recently posted.</p>
```

Is This Better?

The site still "works", but looks funny without the proper styling :-/

We will fix that next!



This is just a simple image gallery :)

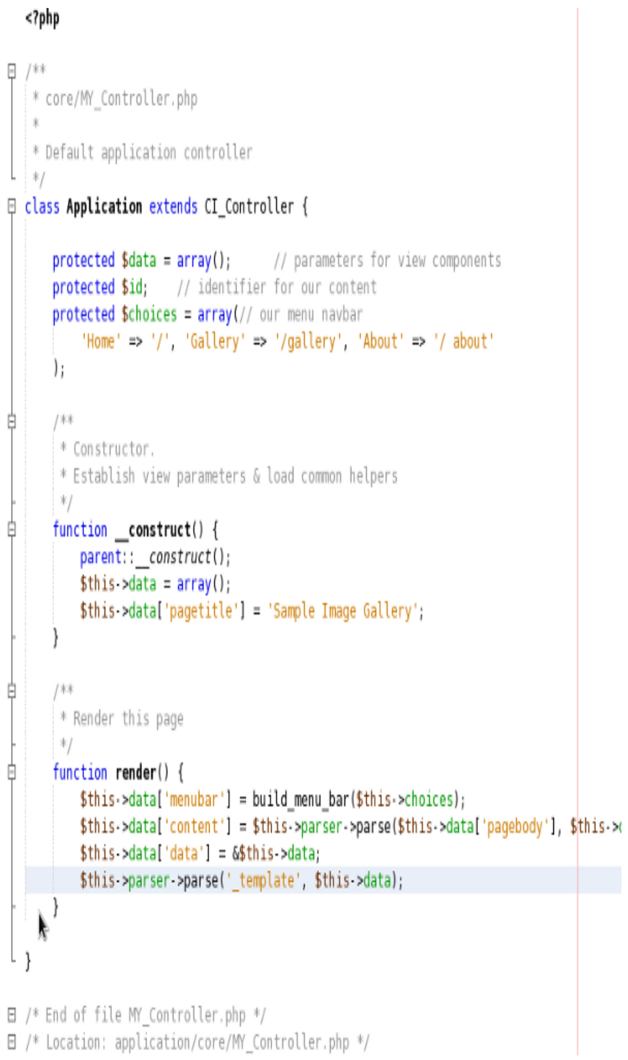
The images above are the three most recently posted.

Build a base controller

We will make a base controller to pull the view pieces together consistently, using our template.

This will be similar to the one referenced in lesson 2, from the contacts sample webapp. Copy yours from the contacts project to core/MY_Controller.php inside your lab 2 project.

Fix the menu to suit our app, and change the ['title'] reference to ['pagetitle'], to match what we already setup in our template.



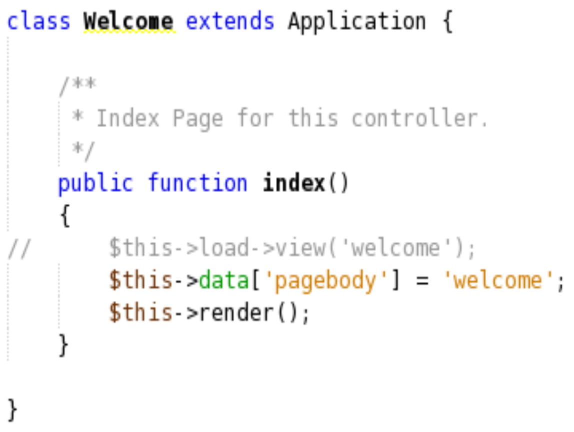
```php
<?php

/**
 * core/MY_Controller.php
 *
 * Default application controller
 */
class Application extends CI_Controller {

    protected $data = array();      // parameters for view components
    protected $id;    // identifier for our content
    protected $choices = array(// our menu navbar
        'Home' => '/', 'Gallery' => '/gallery', 'About' => '/ about'
    );

    /**
     * Constructor.
     * Establish view parameters & load common helpers
     */
    function __construct() {
        parent::__construct();
        $this->data = array();
        $this->data['pagetitle'] = 'Sample Image Gallery';
    }

    /**
     * Render this page
     */
    function render() {
        $this->data['menubar'] = build_menu_bar($this->choices);
        $this->data['content'] = $this->parser->parse($this->data['pagebody'], $this->
        $this->data['data'] = &$this->data;
        $this->parser->parse('_template', $this->data);
    }

}

/* End of file MY_Controller.php */
/* Location: application/core/MY_Controller.php */
```

Apply our base controller

Modify our three controllers to extend our base controller, Application, instead of CI_Controller.

In these controllers, instead of $this->load->view('x') we should set a view parameter instead, and invoke the render method of the base controller.

$this->data['pagebody'] = 'x'; $this->render();



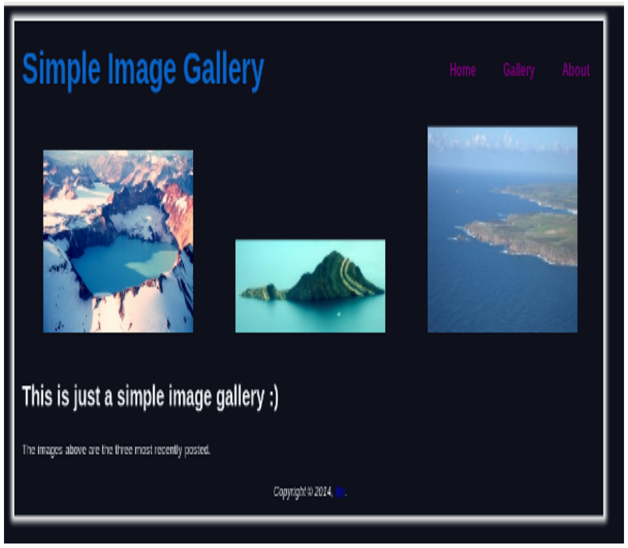
```php
class Welcome extends Application {

    /**
     * Index Page for this controller.
     */
    public function index()
    {
//      $this->load->view('welcome');
        $this->data['pagebody'] = 'welcome';
        $this->render();
    }

}
```

What Do You Think Now?

We also need to copy the common_helper from the contacts starter, so we can use its build_menu function.

We should autoload the common and url helpers, and the parser library, since they are used in Application.

Looking better, although the image references are still hard-coded. We'll fix that next.



This is just a simple image gallery :)

The images above are the three most recently posted.

Copyright © 2014, Me.

```
$autoload['helper'] = array('common', 'url');
```

Add a Model?

We already have a SQL script in the data/setup folder of our starter. How convenient! It's like someone planned to use a database eventually :-/

The first step is to create a database, and import the data/setup/images.sql script to create a table we can then use.

In config/autoload.php, we should add the database library, and in config/database.php you should adjust your settings for your system.

```
$active_group = 'default';
$query_builder = TRUE;

$db['default'] = array(
        'dsn'   => '',
        'hostname' => 'localhost',
        'username' => 'root',
        'password' => '',
        'database' => 'picassos',
        'dbdriver' => 'mysqli',
        'dbprefix' => '',
        'pconnect' => TRUE,
        'db_debug' => TRUE,
        'cache_on' => FALSE,
        'cachedir' => '',
        'char_set' => 'utf8',
        'dbcollat' => 'utf8_general_ci',
        'swap_pre' => '',
        'autoinit' => TRUE,
        'encrypt' => FALSE,
        'compress' => FALSE,
        'stricton' => FALSE,
        'failover' => array(),
        'save_queries' => TRUE
);
```

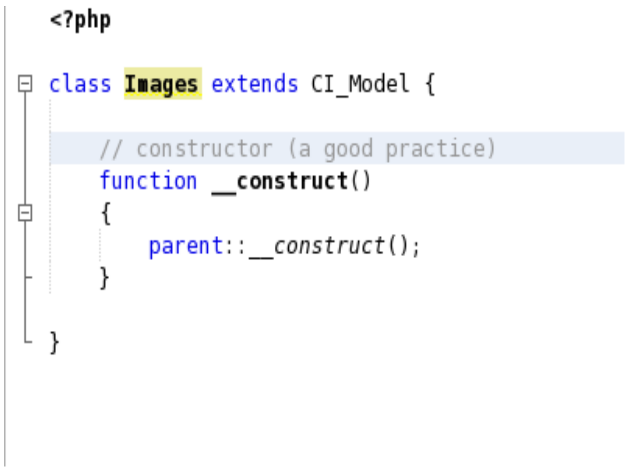
```
$autoload['libraries'] = array('database', 'parser');
```

Now We Can Add a Model

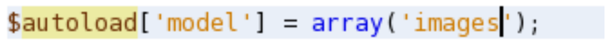
The contacts example had a base model, but that feels like overkill for this simple app. We only need two methods in our model: get all images, ordered properly, and get the three newest images.

Of course, if you want a complete MY_Model in your starter app, for convenience, feel free to use that here.

We just need a simple model, which I suggest calling "Images". It is used on most of our pages, hence is safe to autoload in our configuration.



```php
<?php

class Images extends CI_Model {

    // constructor (a good practice)
    function __construct()
    {
        parent::__construct();
    }

}
```

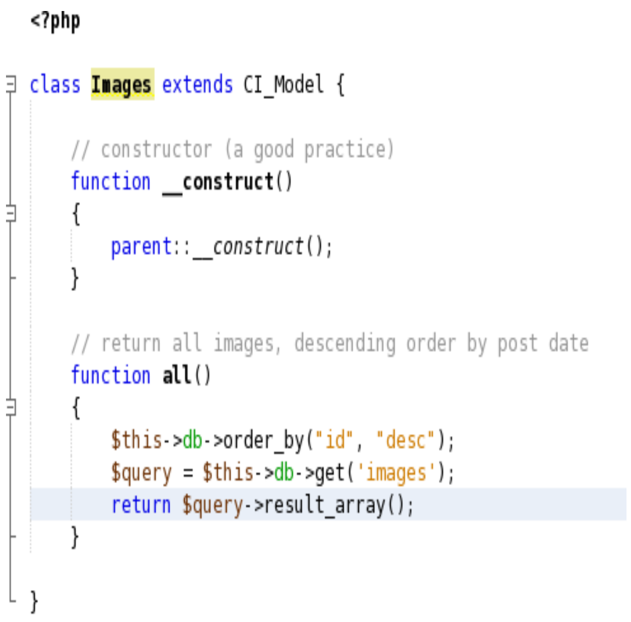


```php
$autoload['model'] = array('images');
```

Revisit the gallery page

Instead of hard-coding image names like the original static site, we can generate the HTML for those programmatically, in our Gallery controller.

We will need to add a suitable method to our Images model to provide all of the image data. Let's call it *all()*, and have it return an array of images, in reverse posting order (newest ones first).



```php
<?php

class Images extends CI_Model {

    // constructor (a good practice)
    function __construct()
    {
        parent::__construct();
    }

    // return all images, descending order by post date
    function all()
    {
        $this->db->order_by("id", "desc");
        $query = $this->db->get('images');
        return $query->result_array();
    }

}
```

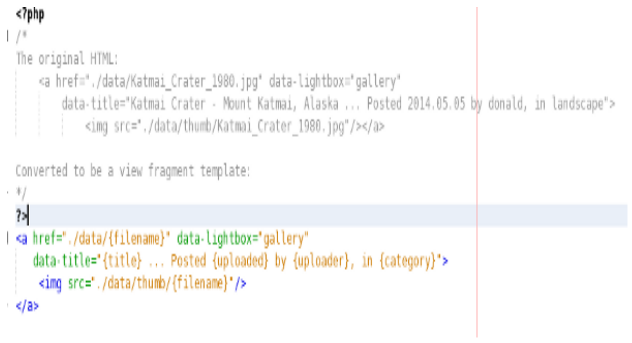
Let's Handle a Single Image

We can make a view fragment to handle the HTML for a single image.

Copy the code for one image table cell, and tailor it to use the appropriate field names from our database as substitution fields.

Save this as views/_cell.php. The underscore at the front of the name reinforces that this is for internal use.

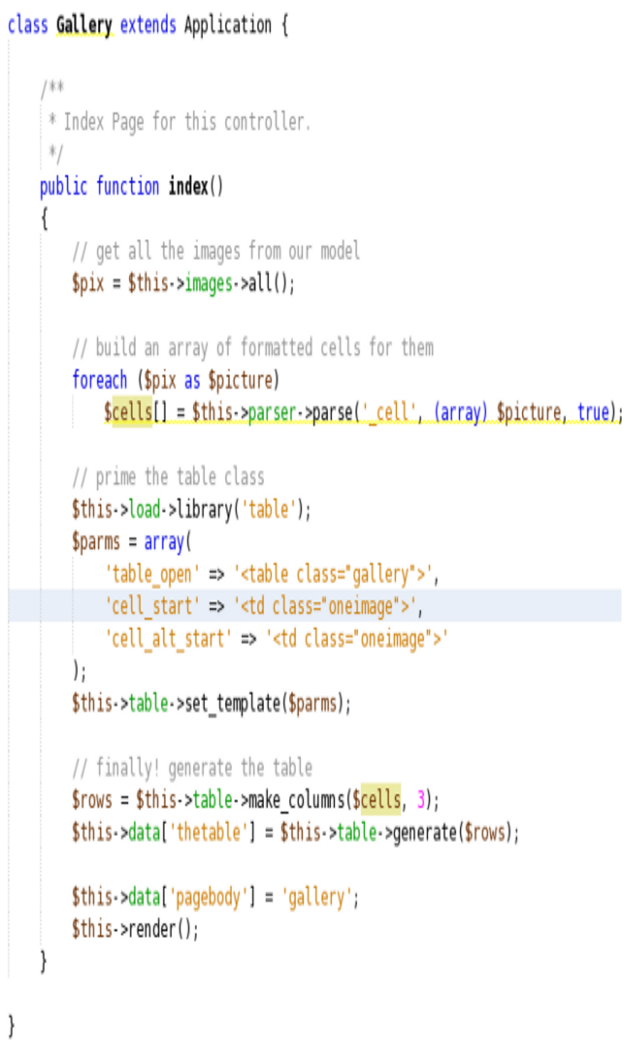
Notice that there is no PHP in the view fragment.



```
<?php
/*
The original HTML:
    <a href="./data/Katmai_Crater_1980.jpg" data-lightbox="gallery"
        data-title="Katmai Crater - Mount Katmai, Alaska ... Posted 2014.05.05 by donald, in landscape">
            <img src="./data/thumb/Katmai_Crater_1980.jpg"/></a>

Converted to be a view fragment template:
*/
?>
<a href="./data/{filename}" data-lightbox="gallery"
   data-title="{title} ... Posted {uploaded} by {uploader}, in {category}">
    <img src="./data/thumb/{filename}"/>
</a>
```

Let's Generate Our Gallery

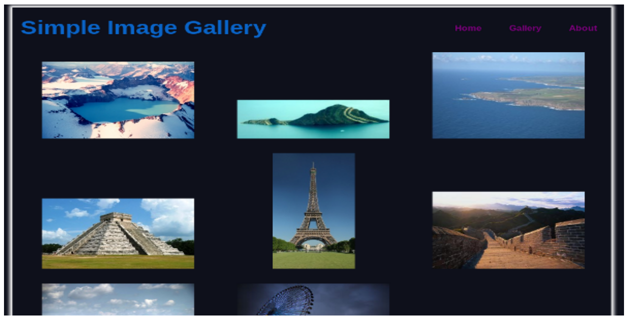
We can use the HTML table class to generate the HTML for our table now. This logic would go inside our Gallery controller.



```
class Gallery extends Application {

    /**
     * Index Page for this controller.
     */
    public function index()
    {
        // get all the images from our model
        $pix = $this->images->all();

        // build an array of formatted cells for them
        foreach ($pix as $picture)
            $cells[] = $this->parser->parse('_cell', (array) $picture, true);

        // prime the table class
        $this->load->library('table');
        $parms = array(
            'table_open' => '<table class="gallery">',
            'cell_start' => '<td class="oneimage">',
            'cell_alt_start' => '<td class="oneimage">'
        );
        $this->table->set_template($parms);

        // finally! generate the table
        $rows = $this->table->make_columns($cells, 3);
        $this->data['thetable'] = $this->table->generate($rows);

        $this->data['pagebody'] = 'gallery';
        $this->render();
    }

}
```

What Does It Look Like?

Can you see a difference?



Revisit the Homepage

The homepage shows the 3 most recent images posted. This is very similar to the gallery, and we can leverage what we did for it.

Add a newest() method to our model, which just returns the 3 newest images.

```
class Images extends CI_Model {

    // constructor (a good practice)
    function __construct()
    {
        parent::__construct();
    }

    // return all images, descending order by post date
    function all()
    {
        $this->db->order_by("id", "desc");
        $query = $this->db->get('images');
        return $query->result_array();
    }

    // return just the 3 newest images.
    function newest()
    {
        $this->db->order_by("id", "desc");
        $this->db->limit(3);
        $query = $this->db->get('images');
        return $query->result_array();
    }

}
```

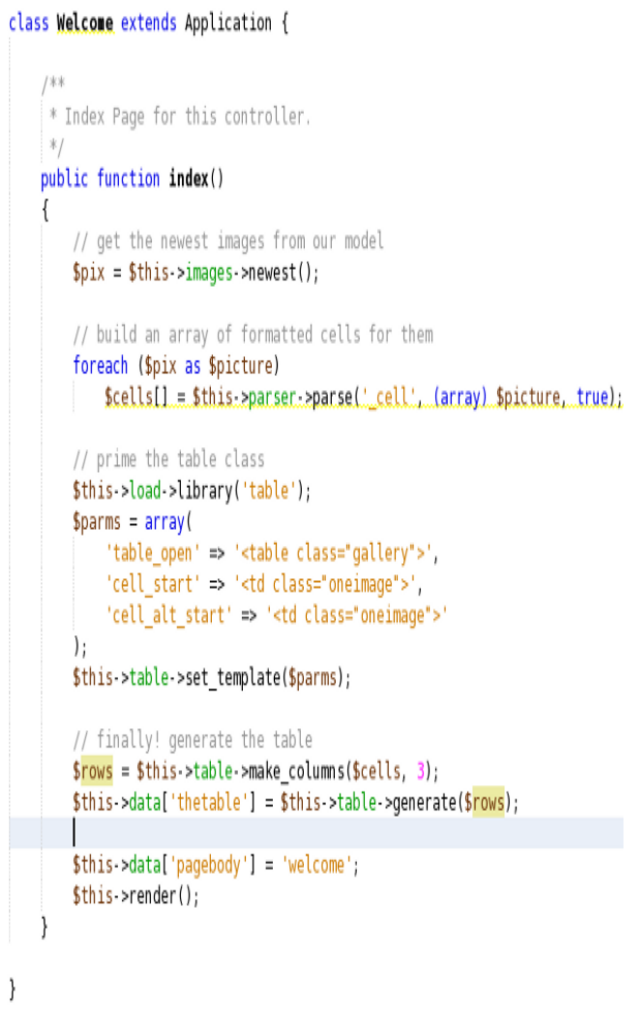
Fix the Homepage View

Adjust the "welcome" view, the same way we changed the "gallery" view.

```
{thetable}
<h1>This is just a simple image gallery :)</h1>
<p>The images above are the three most recently posted.</p>
```

Fix the Homepage Controller

Finally, update the "Welcome" controller, similarly to the Gallery one.



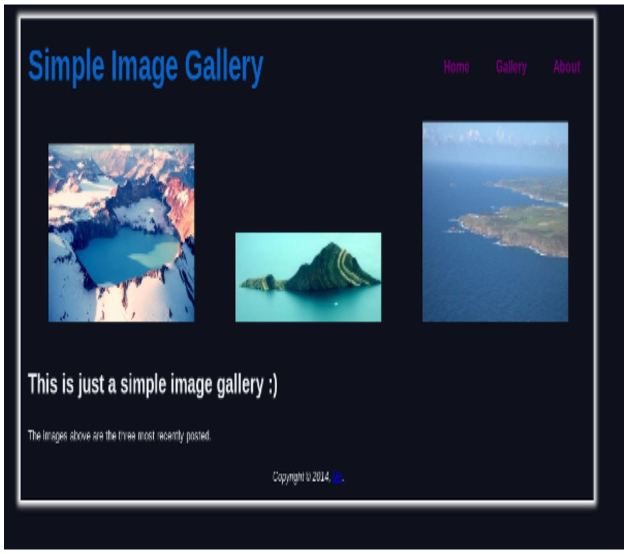
```
class Welcome extends Application {

    /**
     * Index Page for this controller.
     */
    public function index()
    {
        // get the newest images from our model
        $pix = $this->images->newest();

        // build an array of formatted cells for them
        foreach ($pix as $picture)
            $cells[] = $this->parser->parse('_cell', (array) $picture, true);

        // prime the table class
        $this->load->library('table');
        $parms = array(
            'table_open' => '<table class="gallery">',
            'cell_start' => '<td class="oneimage">',
            'cell_alt_start' => '<td class="oneimage">'
        );
        $this->table->set_template($parms);

        // finally! generate the table
        $rows = $this->table->make_columns($cells, 3);
        $this->data['thetable'] = $this->table->generate($rows);

        $this->data['pagebody'] = 'welcome';
        $this->render();
    }

}
```

Are We There Yet?

We are indeed there:)

We have successfully converted a static website into a CodeIgniter webapp.

There are many efficiencies that could be achieved with re-factoring, but the idea and the process should be clear.



This is just a simple image gallery :)

The images above are the three most recently posted.

Copyright © 2014, Me.

Congratulations!

You have completed tutorial #tutorial02c: Convert Static Site to CodeIgniter

If you would take a minute to provide some feedback, we would appreciate it!

The next activity in sequence is: lab02 CodeIgniter Intro 2015.01.18 17:30

You can use your browser's back button to return to the page you were on before starting this activity, or you can jump directly to the course homepage, organizer, or reference page.