# MVC Framework Introduction

## lesson #lesson02

**James L. Parry**
**B.C. Institute of Technology**

## Agenda

## Setup

## Setup

This lesson mentions source code files from the Example Contacts demo webapp. Please download that project first, and extract it locally, so that you can refer to the source files during the lesson.

## Frameworks

## Frameworks

An MVC framework is one with conventions and pre-built components that encourage or enforce good programming practices.

CodeIgniter is one such framework.

## Design Pattern Driven!

Model-View-Controller is a design pattern.

It is an "industry-accepted best practice", that says that it is good to keep separate concerns apart from each other.

CodeIgniter has base classes for models and controllers, and it has components to build presentation content separate from these. This is not conventional PHP scripting!

A typical MVC framework incorporates many other design patterns:

- DAO (DB access)
- Business delegate (framework itself)
- Session facade (session library)
- Front controller (index.php)
- Intercepting filter (hooks)
- Active record (model)

## Categorizing MVC Frameworks

Frameworks come in all sizes.

They differ in philosophy, scope, management tools & included plugins.

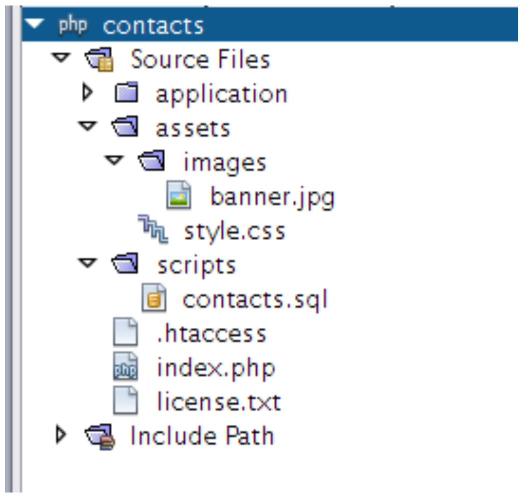
| Category | Adds | Size | Example |
|---|---|---|---|
| Micro | MVC, plugins, routing, instance | 0.3MB | Fat-Free |
| Lean | Config, templates | 2MB | CodeIgniter |
| Normal | Scaffolding, auth, ORM, CLI, IDE plugin | 10MB | CakePHP |
| Enterprise | Engines (templating, ORM), included plugins, installer, starter, generators | 24MB+ | Symfony2 |

## Project Files

A CodeIgniter project contains application logic and support files.

"index.php", in the document root, *is* the front controller, i.e. entry point.

".htaccess" is an Apache configuration file, to eliminate the need to explicitly have "index.php" in URLs.

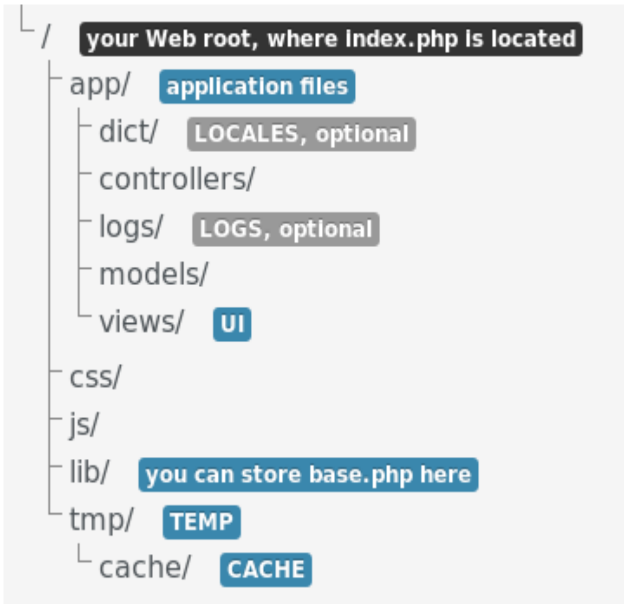


```
contacts
  Source Files
    application
    assets
      images
        banner.jpg
      style.css
    scripts
      contacts.sql
    .htaccess
    index.php
    license.txt
  Include Path
```

## Folder Structures

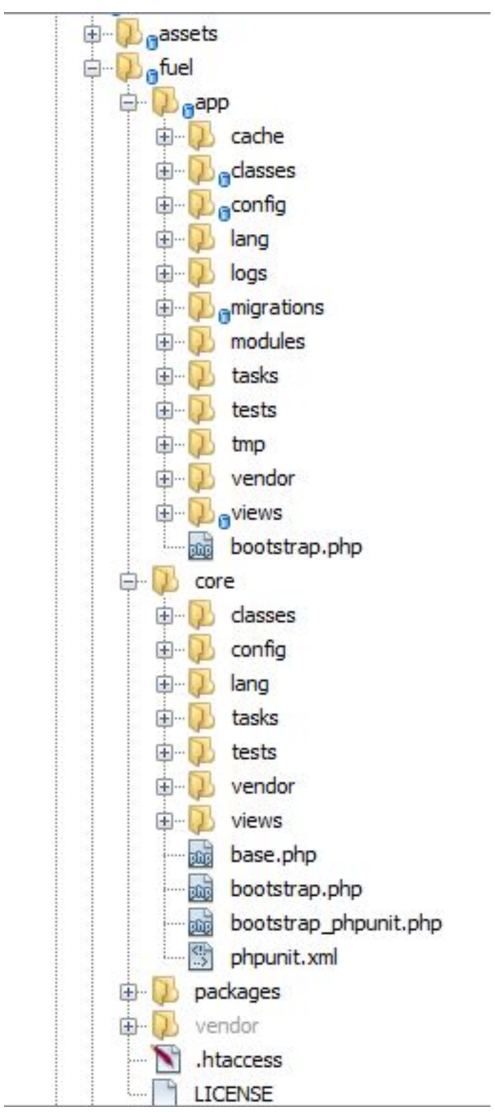
Other frameworks follow similar conventions.

*Fat-Free*



```
└ /  your Web root, where index.php is located
  ├ app/  application files
  │ ├ dict/  LOCALES, optional
  │ ├ controllers/
  │ ├ logs/  LOGS, optional
  │ ├ models/
  │ └ views/  UI
  ├ css/
  ├ js/
  ├ lib/  you can store base.php here
  └ tmp/  TEMP
    └ cache/  CACHE
```

*FuelPHP*



```
assets
fuel
  app
    cache
    classes
    config
    lang
    logs
    migrations
    modules
    tasks
    tests
    tmp
    vendor
    views
    bootstrap.php
  core
    classes
    config
    lang
    tasks
    tests
    vendor
    views
    base.php
    bootstrap.php
    bootstrap_phpunit.php
    phpunit.xml
  packages
  vendor
  .htaccess
  LICENSE
```

## Class Loader

CodeIgniter has its own class loader.

It uses the folder structure to locate classes (per MVC)

```
$this->load->model('customers');
$this->customers->get(...);
```

It enforces naming conventions too.

## Routing

CodeIgniter normally routes requests to controllers by convention, i.e. using the URI segment and looking in the controllers folder.Other frameworks often handle routing a bit differently.

*Kohana*

```
Route::set(‘default’,
    ‘(<controller>(/<action>(/<id>)))’)
    ->defaults(array(
        ‘controller’ => ‘welcome’,
        ‘action’=>‘index’, ))
```

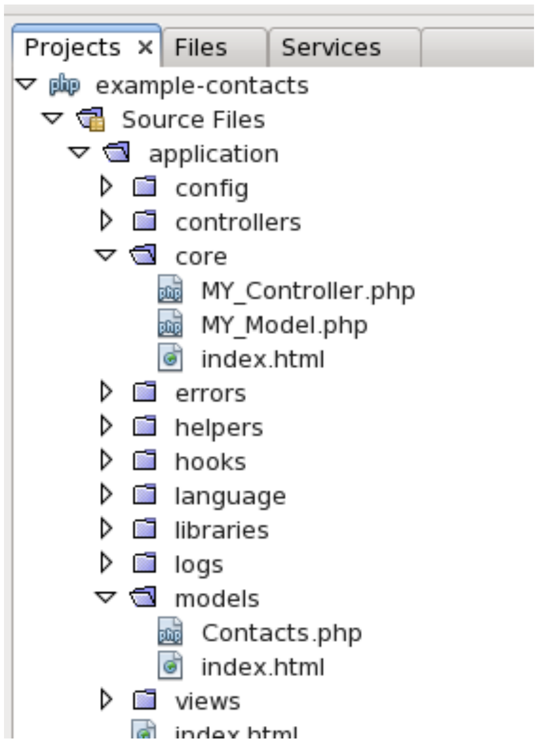
*Laravel*

```
//Basic POST Route
Route::post(‘/’, function()
{
        return ‘example content’;
});
```

## Models

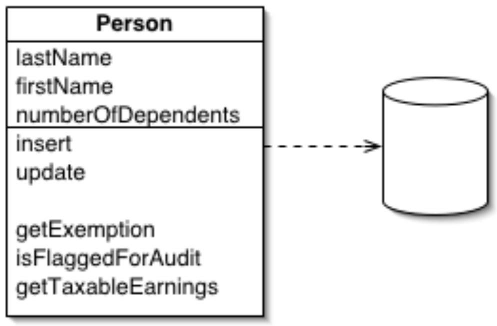
Models encapsulate data sources or entities.

Each model is a class that wraps a data source.



## Active Record

CodeIgniter encourages the Active Record design pattern which says you build CRUD methods into models.



## Query Builder

CodeIgniter comes with a QueryBuilder library, to make it easier to access your databases.

```
$this->db->select('*')->from('my_table')
    ->group_start()
    ->where('a', 'a')
    ->or_group_start()
    ->where('b', 'b')
    ->where('c', 'c')
    ->group_end()
    ->group_end()
    ->where('d', 'd')
    ->get();
```

## Model Classes

In CodeIgniter, models are classes that extend CI_Model.

Each data source, eg. RDB table, has its own model.

```
public class Orders extends CI_Model {
    public function add_item($code, $quant
    public function calc_total() {...}
}
```

## Model Methods

CRUD

Some business logic

Work with objects or associative arrays

* not * beans, i.e. entity models

Do not generate views, but generate or provide data to be passed to views

Usage: $this->load->model('goodies');

## Base Model

You can provide your own base model, MY_Model, with common methods. See core/MY_Model in the example webapp.

This could be used to provide a simple object-relational map.

Sample properties: $_tableName, $_keyField

Sample methods: create() get($key), add($record), update($record), delete($key), exists($key)

## Other Model Examples

Other frameworks have similar functionality.

*Fat-Free*

```
$db = new DB\SQL('mysql:host=localhost';
    port=3306;dbname=mystuff',
    'myuserid','mypassword');
$user = new DB\SQL\Mapper($db,'users');
$user->load('id=1');
$user->name='Pointy haired boss';
$user->save();
```

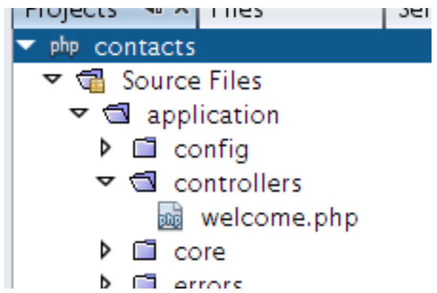
*CakePHP*

```
App::uses(’AppModel’, ’Model’);
class Ingredient extends AppModel {
    public $name = ’Ingredient’;
}
```

## Controllers

Controllers handle incoming requests.

A Controller serves as an intermediary between the Model, the View, and any other resources needed to process an HTTP request and generate a web page.

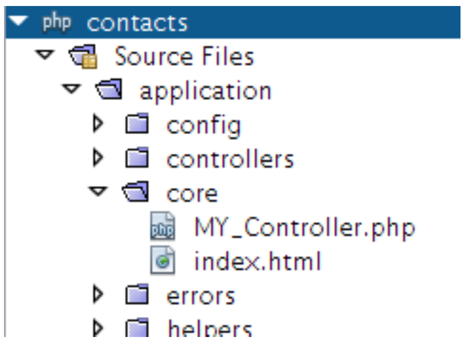


## CI Controllers

In CI, controllers are classes that extend CI_Controller or your base controller

Our base controller implements view templating, through render()

See application/core/MY_Controller.php



## App Controllers

Each of your webapp controllers builds on your base controller

Normally, each usecase has its own controller

Additionally, controllers or subcontrollers are used for services

See application/controllers/Welcome.php

## Controller Methods

URLs are segment based

example.com/class/function/parm1/parm2

index() is the default handler function

Other functions are treated as sub-controllers

Parameters are passed positionally

Confusion is possible ... application/controllers/business/work

## Example Base Controller

core/MY_Controller.php

```
class Application extends CI_Controller {
// fields

function __construct() {
    parent::__construct();
    // make sure we play the game nicely
    $this->load->helper('common');
    $this->load->helper('url');
}

function render() {...}
}
```

## Other Framework Examples

*FuelPHP*

```
class Controller_Example extends Controller {

    public function action_index() {
        $data['css'] = Asset::css(array
            ('reset.css','960.css','main.css'));
        return Response::forge(View::forge('welcome/index'));
    }
}
```

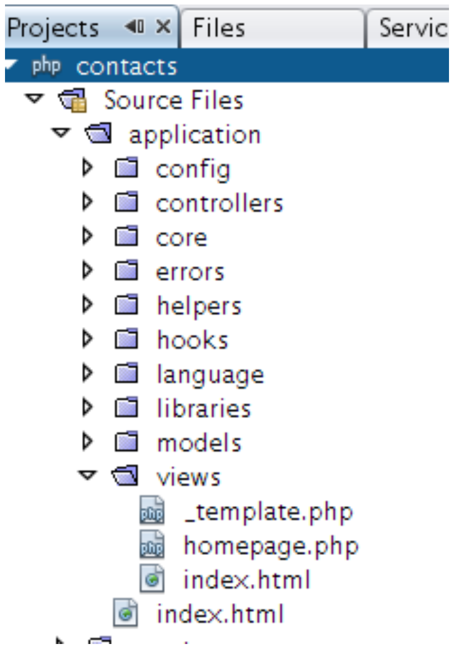
*Laravel*

```
class Controller extends BaseController {
    public function home() {
        return View::make(‘home’);
    }
}
...
Route::get(‘/’, ‘Controller@home’)
```

## Views

Views are the components that deal with presentation, typically for a web browser.

Views generate the information presented to a user, for instance a web page.



## View Templates

A normal practice is to build a view template, which wraps the "real" webpage body, whose name is passed as a parameter; this is managed by the base controller

CI has a template parser built-in, which lets us eliminate most PHP script from a view.

See application/views/_template.php

## View Fragments

Each of your view fragments is stored in its own file, using the substitution fields for passed parameters.

Substitution fields are enclosed in braces inside your view file.

See application/views/homepage.php

## View Usage

Javascript libraries are not well-supported in CI; you can use them, but effectively client-side

Do use CSS to control the appearance

Usage: $this->load->view('showoff',$data); or $this->parser->load('showoff',$data);

;

Spoiler: JS/CSS integration coming :)

## View Examples From Other Frameworks

Other frameworks have similar functionality.

*Fat-Free*

```
$f3->set('name','Jim');
$f3->set('title','Sir');
...
$view = new Template;
echo $view->render('template.html');

where the HTML fragment includes
... <p>Hello {{ @title }} {{ @name }}</p>
```

*FuelPHP*

```
fuel/app/views/layout.php ...
<html>
    <head>
        <?php echo $head; ?>
    </head>
    <body>
        <?php echo $header; ?>
        <?php echo $content; ?>
        <?php echo $footer; ?>
    </body>
</html>
```

## View Methods

Seriously?

If you need functionality, use helpers

## Support Components

## Support Components

An MVC framework comes with a number of pre-built classes or scripts to make your webapp development easier and more consistent.

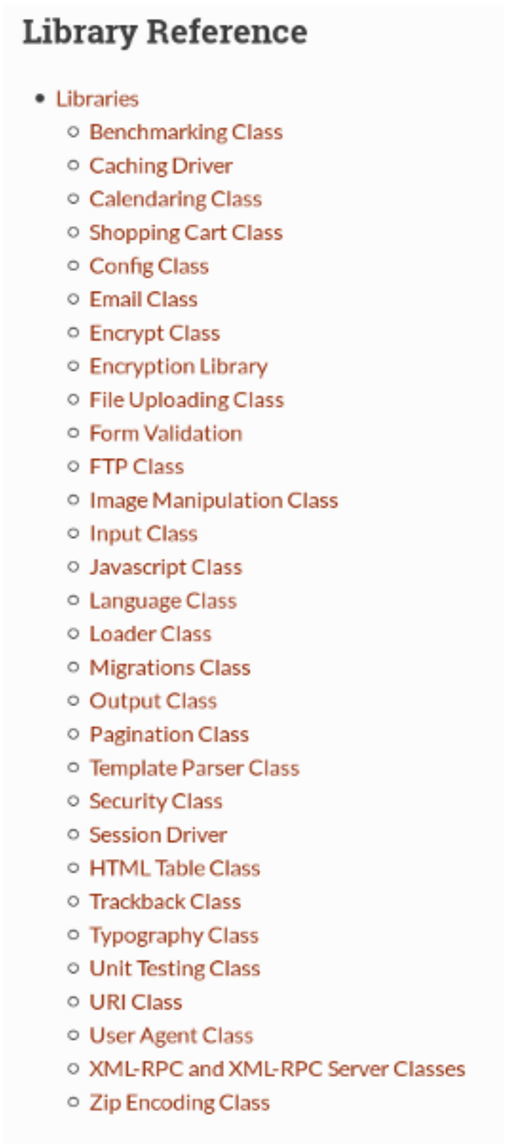
These are collectively referred to as support components

## Libraries

Any classes that you would like to use can be put into the application/libraries folder

Usage: $this->load->library('whatever');

Examples: third party classes, bean-like classes?



**Library Reference**

- Libraries
  - Benchmarking Class
  - Caching Driver
  - Calendaring Class
  - Shopping Cart Class
  - Config Class
  - Email Class
  - Encrypt Class
  - Encryption Library
  - File Uploading Class
  - Form Validation
  - FTP Class
  - Image Manipulation Class
  - Input Class
  - Javascript Class
  - Language Class
  - Loader Class
  - Migrations Class
  - Output Class
  - Pagination Class
  - Template Parser Class
  - Security Class
  - Session Driver
  - HTML Table Class
  - Trackback Class
  - Typography Class
  - Unit Testing Class
  - URI Class
  - User Agent Class
  - XML-RPC and XML-RPC Server Classes
  - Zip Encoding Class

## Example Library: Product

libraries/Product.php

Use for object model for entities stored in an XML document or RDB table

"models/products" would be the aggregation of these

Again, just one way to do this

## Libraries in Other Frameworks

Laravel

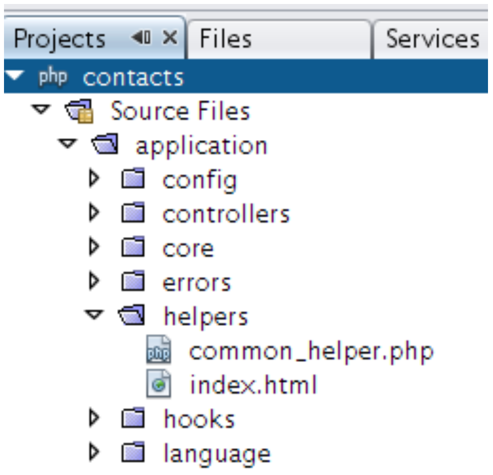
- Auth
- Validation
- Pagination
- Database

Symfony

- Browserkit
- Config
- DomCrawler
- Finder
- OptionsResolver
- ...

## Helpers

Put commonly used functions inside a "helper" script

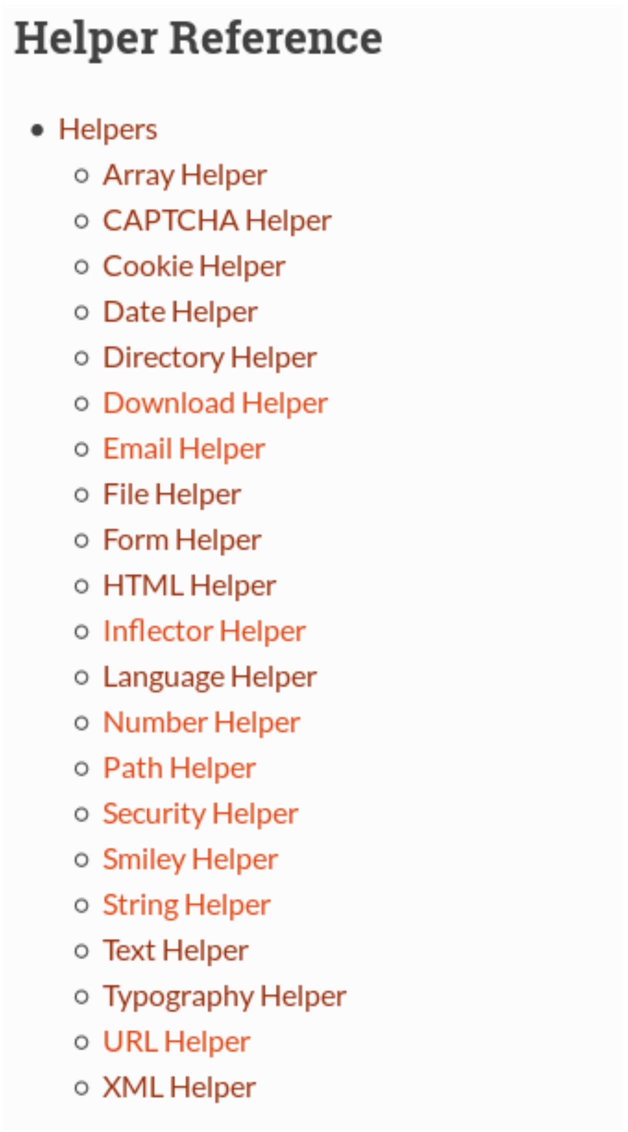


## Helpers are "Classic"

Functions in any such loaded helpers are available everywhere

Usage: $this->load->helper('common');

See application/helpers/common_helper.php



Helper Reference

- Helpers
  - Array Helper
  - CAPTCHA Helper
  - Cookie Helper
  - Date Helper
  - Directory Helper
  - Download Helper
  - Email Helper
  - File Helper
  - Form Helper
  - HTML Helper
  - Inflector Helper
  - Language Helper
  - Number Helper
  - Path Helper
  - Security Helper
  - Smiley Helper
  - String Helper
  - Text Helper
  - Typography Helper
  - URL Helper
  - XML Helper

## Example Helper: common_helper

Form field handling, using associative array as data transfer object/buffer

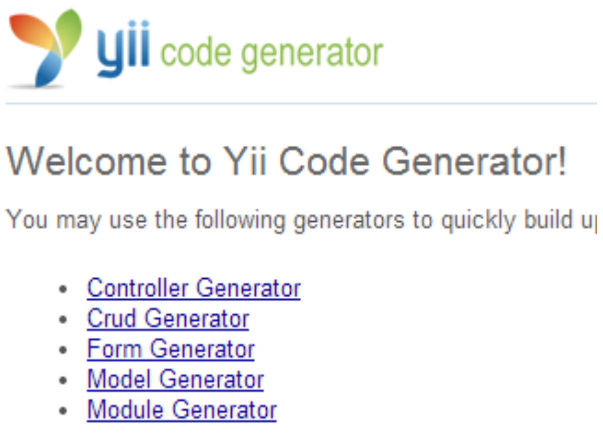
fieldExtract($source,$target,$fields)
-$source – assoc array from form
-$target – corresponding object
-$fields – names of fields in this form

fieldInject($source,$target,$fields)
-$source – original data object
-$target – assoc array for parameters to view
-$fields – names of fields in this form

## Utilities Examples

Many frameworks come with bundled utilities.

*Yii*



Welcome to Yii Code Generator!

You may use the following generators to quickly build u

- Controller Generator
- Crud Generator
- Form Generator
- Model Generator
- Module Generator

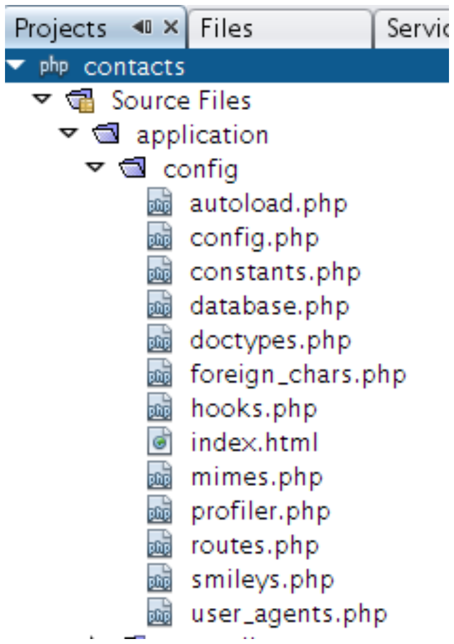
*FuelPHP*

```
$ php oil g scaffold monkey name:string de

Created model: APPPATH/classes/model/monke
Created migration: APPPATH/migrations/003_
Created controller: APPPATH/classes/contro
Created view: APPPATH/views/monkeys/index.
Created view: APPPATH/views/monkeys/view.p
Created view: APPPATH/views/monkeys/create
Created view: APPPATH/views/monkeys/edit.p
Created view: APPPATH/views/monkeys/_form.
```

## Configuration

CodeIgniter provides a "config" folder to let you customize your web app



```
Projects | Files | Servic
php contacts
  Source Files
    application
      config
        autoload.php
        config.php
        constants.php
        database.php
        doctypes.php
        foreign_chars.php
        hooks.php
        index.html
        mimes.php
        profiler.php
        routes.php
        smileys.php
        user_agents.php
```

## Configuration Practices

index.php points to your system folder

application/config/ contains...

- autoload – specify components to pre-load
- config – specify application parameters
- database – specify RDB parameters
- routes – specify default controller

## Configuration Examples From Other Frameworks

*Laravel*

```
composer create-project laravel/laravel project_name --prefer-dist
Composer.json:
"require": {
    "laravel/framework": "4.1.*",
    "intervention/image": "1.*"
}

Composer update
```

*Kohana*

```
Kohana::modules(array(
   'auth'       => MODPATH.'auth',       //
   'cache'      => MODPATH.'cache',      //
   'codebench'  => MODPATH.'codebench',  /
   'database'   => MODPATH.'database',   /
   'image'      => MODPATH.'image',      //
   'orm'        => MODPATH.'orm',        //
   'oauth'      => MODPATH.'oauth',      //
   'pagination' => MODPATH.'pagination', /
   'unittest'   => MODPATH.'unittest',   /
   'userguide'  => MODPATH.'userguide',  /
   ));
```

## Your CodeIgniter Webapp!

## Your CodeIgniter Webapp!

This is a simplified look at where stuff goes inside your webapp, and the general steps you would follow to build a simple CodeIgniter webapp.

## Your Folder Structure

CI system → /xampp/htdocs/system3

CI user guide → desktop?

CI starter → copy to /xampp/htdocs and tailor for a new webapp

NOTE: the starter project differs from the official CI download in several ways: base controller with templating, simple "ORM: model, standard config for Apache

## Setup Your Webapp

Copy/extract the starter to a folder in htdocs

Open it with NB & rename (proj & folder)

Ensure virtual host mapping

Modify NB project run config?

Create RDB if needed

## Configure Your Webapp

config/constants.php – adjust as needed

config/config.php – define menu navbar

## Setup Your View Template

Make a wireframe model for the website.

Tailor views/_template.php to match it

This template contains only the common page elements for the site, with substitutable fields for the menu and content.

## Tailor Your Webapp

controllers/?.php – controller per navbar item, bound to corresponding view

models/?.php – model per RDB table

views/welcome.php – tailor your homepage

views/?.php – view per controller

## Manage Your Assets

assets/css/style.css – your CSS

assets/images/ - your webapp images

assets/css/ - any imported CSS frameworks

assets/js/ - any imported JS frameworks

## Congratulations!

You have completed lesson #lesson02: MVC Framework Introduction

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

The next activity in sequence is: contacts Example webapp - contacts

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.