Codelgniter and MVC

Enterprise class web application development
Motivation

• You have worked with PHP, for small sites this works very well. HTML files can be easily extended with dynamic content from the database, form processing, etc.

• When sites grow, you might have realized that across multiple pages lots of code repetition occurs. This is a problem when you need to change certain parts of a page, that affects many or all pages.

• Furthermore, its hard to introduce new developers to code someone else has written. It takes a long time to get familiar with the code.
Motivation

- Enterprise class (big) web applications need structure, one HTML file per page with PHP code is not optimal => things get confusing, hard to work in teams, etc.

- Web Application Frameworks provide such a structure which introduce separation of concern, etc.

- Most common for Web application development are frameworks that are based on the Model-View-Controller design pattern

- Motivation video for a MVC framework: http://www.youtube.com/watch?v=p5EIrSM8dCA
Model-View-Controller

- "Separation of concerns" of Logic and Presentation

- **Controller**: Handles all incoming HTTP requests, passes data to the views

- **View**: Renders the HTML output

- **Models**: Encapsulate Business Logic, such as interaction with the database

  - For PHP we introduce CodeIgniter
• **CodeIgniter** is a PHP-based MVC framework that helps structure your code and make redundant tasks less tedious.

• There are countless similar PHP frameworks, the most popular ones being **CakePHP** and **symfony**. Ruby on Rails is famous in the Ruby world.

• CodeIgniter is very light weight. It doesn‘t force any convention but provides many commonly required features through a set of build in libraries.

• CodeIgniter has a low learning curve and is one of the best documented PHP web frameworks.
Codelgniter: Features

- Model-View-Controller Based System

- Extremely Light Weight, does not force any convention

- Full Featured database classes with support for several platforms, Active Record Database Support (ORM)

- Custom Routing

- Form and Data Validation

- Security and XSS Filtering
1. The index.php serves as the front controller, initializing the base resources needed to run CodeIgniter.
2. The Router examines the HTTP request to determine what should be done with it.
3. If a cache file exists, it is sent directly to the browser, bypassing the normal system execution.
4. Security. Before the application controller is loaded, the HTTP request and any user submitted data is filtered for security.
5. The Controller loads the model, core libraries, plugins, helpers, and any other resources needed to process the specific request.
6. The finalized View is rendered then sent to the web browser to be seen. If caching is enabled, the view is cached first so that on subsequent requests it can be served.
Getting Started

• **Directory Structure of CodeIgniter:**
  
  • **index.php** - receives all requests and routes to the right controllers classes and actions, parameters are included in the URL
  
  • **/system** - contains all CodeIgniter classes and libraries provided by the framework
  
  • **/application** - this is where your application code is located, including the model, view and controller classes
Controllers

- Take **incoming HTTP requests** and process them
- Must be the same **filename** as the capitalized **class name**
- Must **extend** the main **Controller class** of CodeIgniter
- Each class function represents an **controller action**, which is rendering a HTML page
- „**index**“ is the default action
Routing Requests

- Per default CodeIgniter maps URL to controller actions:

  /index.php/controller/action

- The **default controller** is "welcome" and the **default action** is "index".

- Custom routing can be configured through:

  /application/config/routes.php
Views

- Are HTML pages or page fragments
- Those are load and sent by the Controller to the Browser by the use of the following code in den Ctrl:
  ```php
  $this->load->view('blog_view');
  ```
- There is no application logic in the views, only display logic (to some degree)
- `<?=` is short form for `<?php echo`
Controller & View sample

```php
<?php

class Blog extends Controller {

    function Blog()
    {
        parent::Controller();
    }

    function index()
    {
        $data['title'] = "My Blog Title";
        $data['heading'] = "My Blog Heading";
        $data['todo'] = array('clean house', '....);

        $this->load->view('blog_view', $data);
    }

?>
```
Helpers

- Helper functions are located in /application/helpers/
- These are small PHP functions that provide shortcuts, to outsource often used code
- For example formatting, text, url or form helpers
- Needs to be loaded through: $this->load->helper('name');

Loading a Helper
$this->load->helper('name');

Using a Helper
<?php echo anchor('blog/comments', 'Click Here');?>

Auto-loading Helpers
application/config/autoload.php

Text Helper
$this->load->helper('text');

word_limiter()
$string = "Here is a nice text string consisting of eleven words.";
$string = word_limiter($string, 4);
// Returns: Here is a nice...
Libraries

• Libraries are similar to helpers

• The also need to be loaded through: $this->load->library('classname');

• The difference is that they **encapsulate** more complex functionality, such as *image processing*, *form validation handling*, *caching*, etc.

• Libraries allow to dynamically extend CodeIgniters functionality and extendability
Database Handling

• Codelgniter provides a simple way to access the database.

• It encapsulates the underlying database (MySQL, Oracle, etc)

• The connection handling is done by Codelgniter and configured through:

/application/config/database.php

• Provides security through automatic escaping of inserted data, avoids Cross-Side Scripting (CSS) attacks

Initializing the Database Class

```php
$this->load->database();
```

Standard Query With Multiple Results

```php
$query = $this->db->query('SELECT name, title, email FROM my_table');
foreach ($query->result() as $row)
{
    echo $row->title;
    echo $row->name;
    echo $row->email;
}
```

```php
echo 'Total Results: ' . $query->num_rows();
```

Database Configuration

```php
$db['test']['hostname'] = "localhost";
$db['test']['username'] = "root";
$db['test']['password'] = "";
$db['test']['database'] = "database_name";
$db['test']['dbdriver'] = "mysql";
$db['test']['dbprefix'] = "";
$db['test']['pconnect'] = TRUE;
$db['test']['db_debug'] = FALSE;
$db['test']['cache_on'] = FALSE;
$db['test']['cachedir'] = "";
$db['test']['char_set'] = "utf8";
$db['test']['dbcollat'] = "utf8_general_ci";
```
Active Records

- CodeIgniter uses a modified version of the **Active Record Database Pattern**.

- This pattern allows information to be **retrieved**, **inserted**, and **updated** in your database with minimal scripting.

---

**Selecting Data**

```php
$query = $this->db->get('mytable', 10, 20);
// Produces: SELECT * FROM mytable LIMIT 20, 10

foreach ($query->result() as $row) {
    echo $row->title;
}
```

**Inserting Data**

```php
$data = array(
    'title' => 'My title',
    'name' => 'My Name',
    'date' => 'My date'
);

$this->db->insert('mytable', $data);
```

**Method Chaining**

```php
$this->db->select('title')->from('mytable')->where('id', $id)
$query = $this->db->get();
```
Application Profiling

- The Profiler Class will display benchmark results, queries you have run, and $_POST data at the bottom of your pages.

- This information can be useful during development in order to help with debugging and optimization.

- Enabling the Profiler somewhere in the Controller:

```
$this->output->enable_profiler(TRUE);
```
Further Reading and Resources

- **CodeIgniter User Guide:**

- **Tutorials and Video Tutorials:**

- **Model-View-Controller and CodeIgniter:**