CodeIgniter 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 familar 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 wich introduce seperation 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

- „*Seperation 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

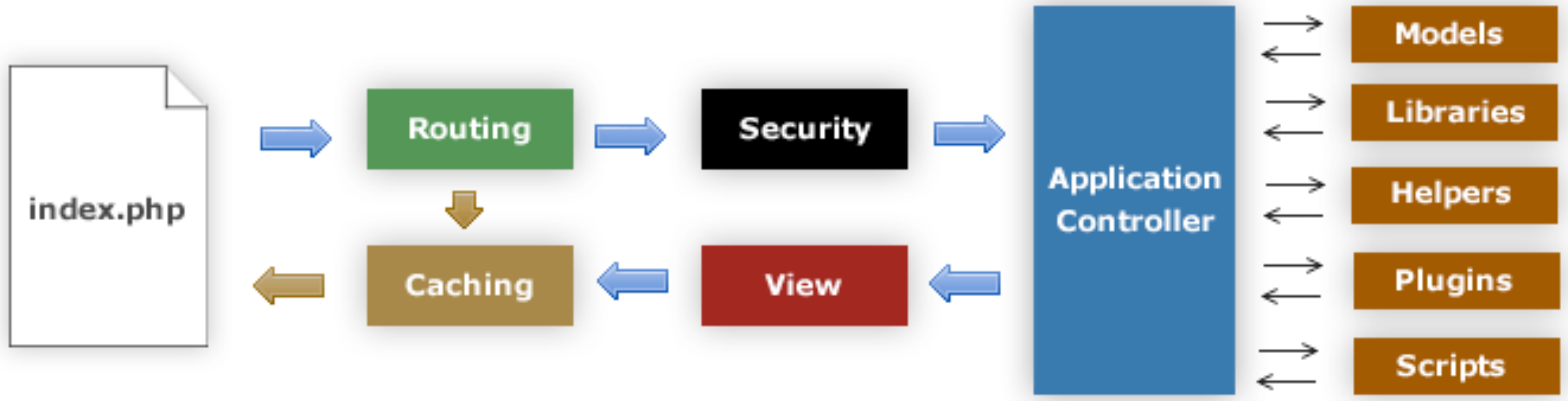
CodeIgniter

- 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.

CodeIgniter: 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

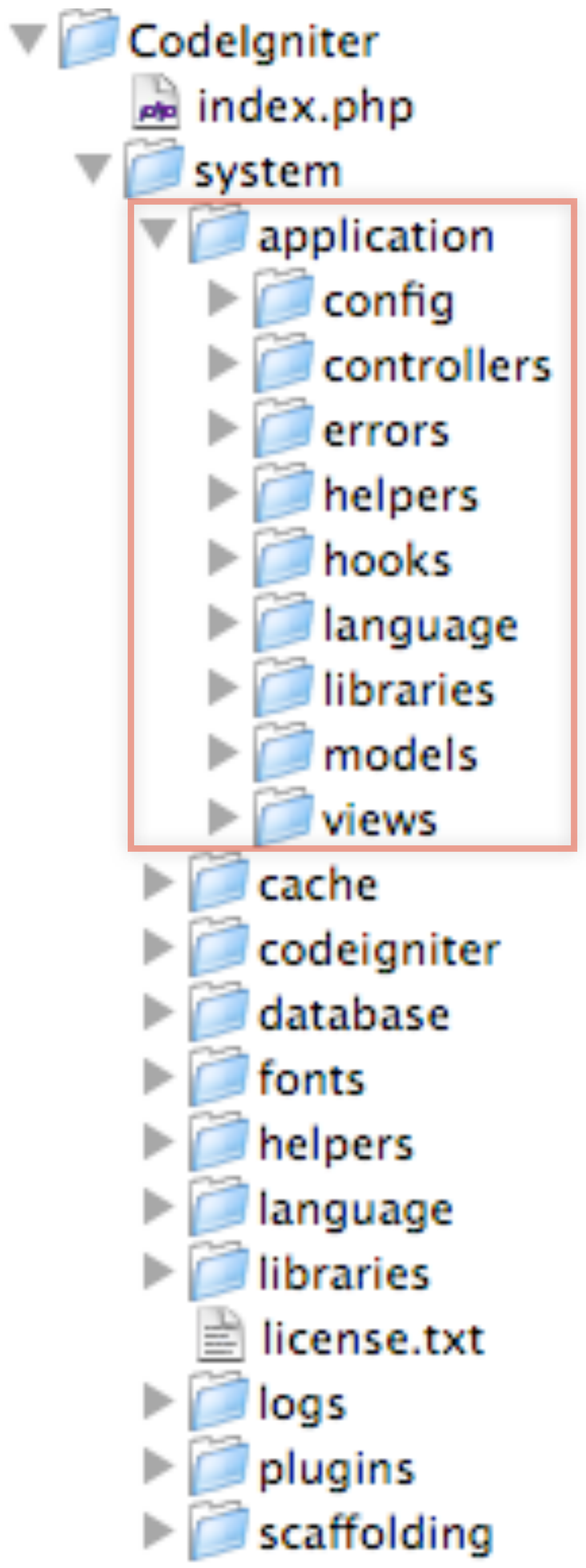
Application Flow Chart



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

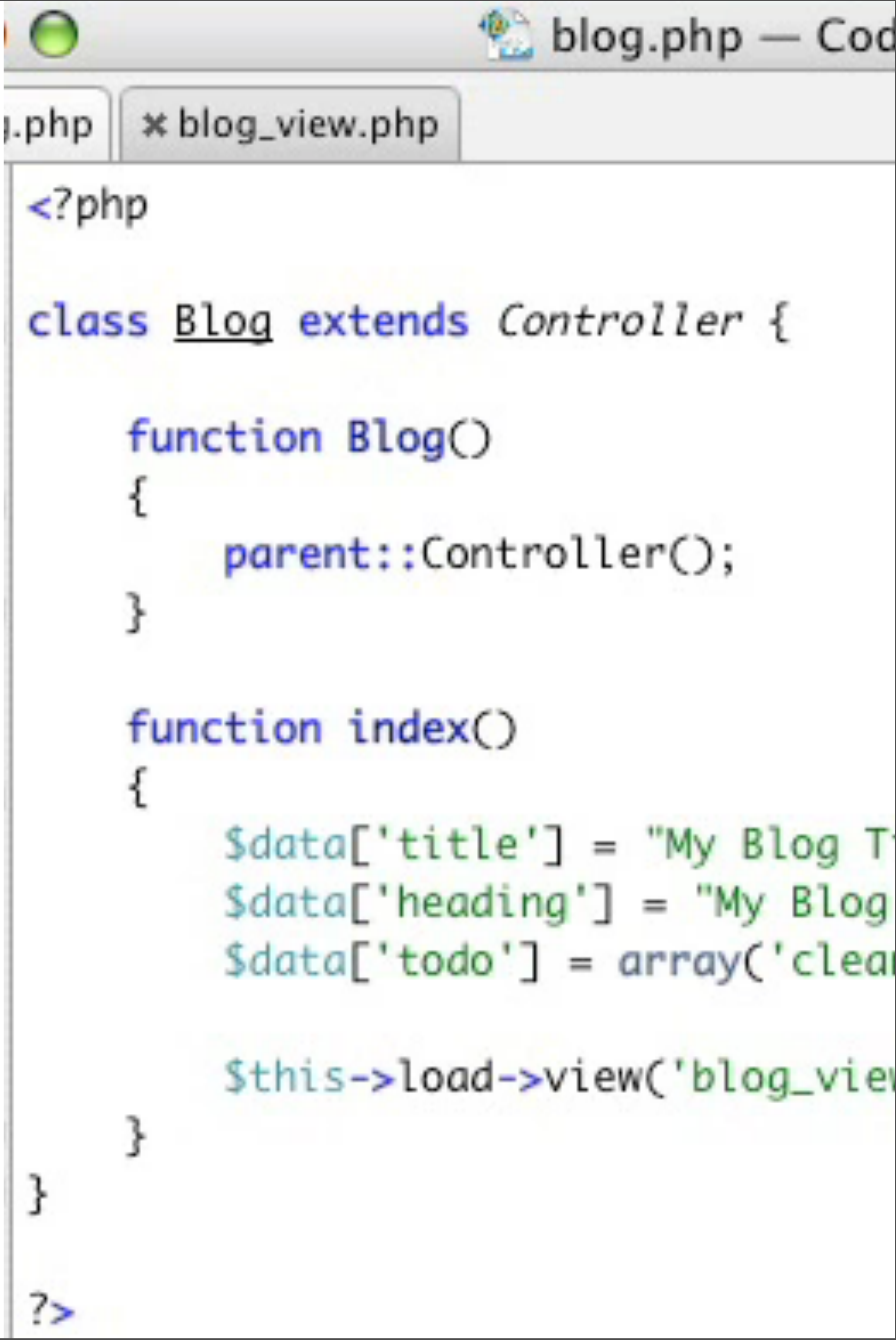
- **Directory Structure of CodeIgniter:**
  - **index.php** - recieves all requests and routes to the right controllers classes and actions, parameters are included in the URL
  - **/system** - contains all CodeIgniter classes and libraries provided by the framework
  - **/application** - this is where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



```
CodeIgniter
    index.php
    system
        application
            config
            controllers
            errors
            helpers
            hooks
            language
            libraries
            models
            views
        cache
        codeigniter
        database
        fonts
        helpers
        language
        libraries
        license.txt
        logs
        plugins
        scaffolding
```

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 redering a HTML page
- „*index*“ is the default action



```
<?php

class Blog extends Controller {

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

    function index()
    {
        $data['title'] = "My Blog T
        $data['heading'] = "My Blog
        $data['todo'] = array('clea

        $this->load->view('blog_vie
    }
}

?>
```

Routing Requests

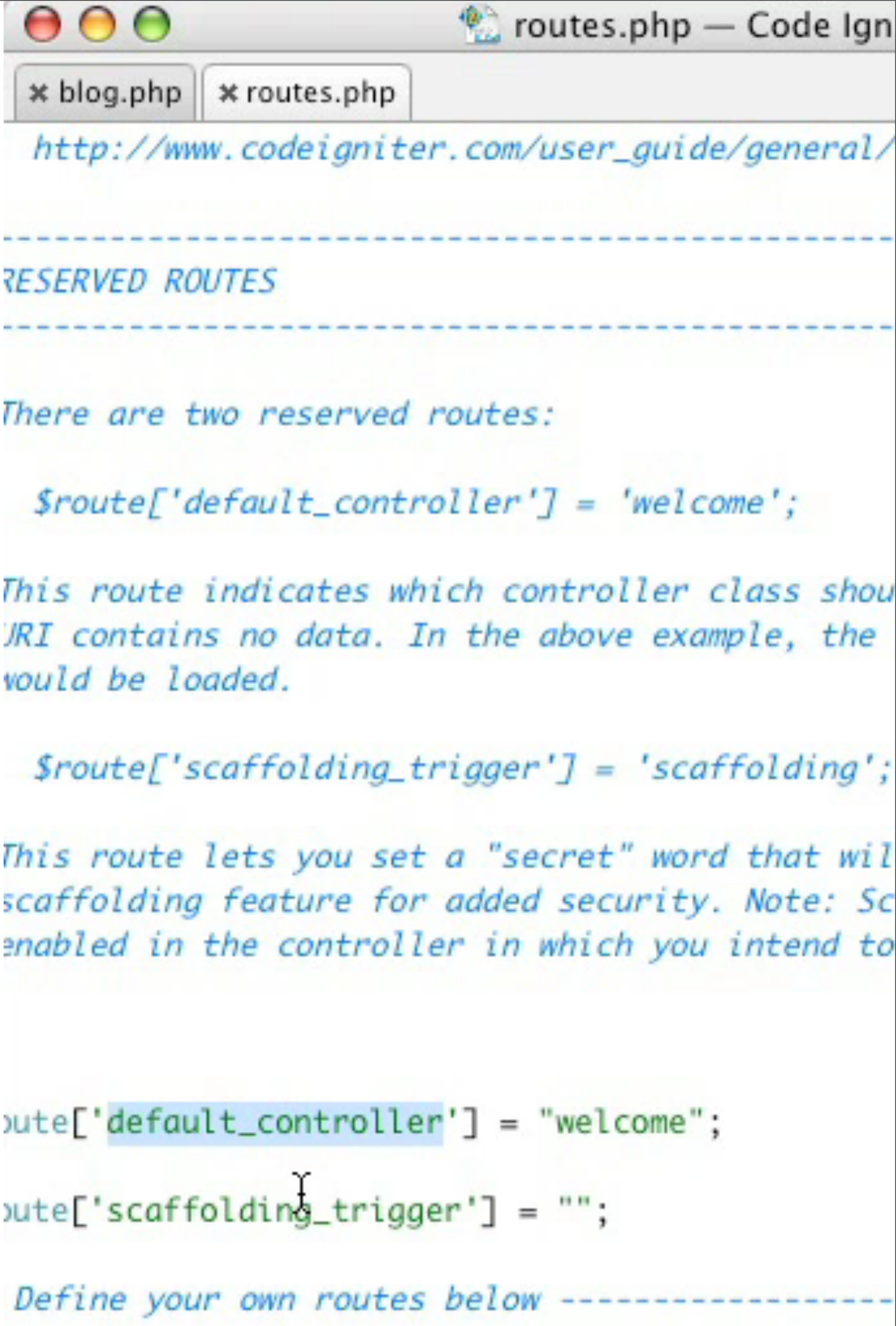
- Per default CodeIgniter maps URL to controller actions:

  /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



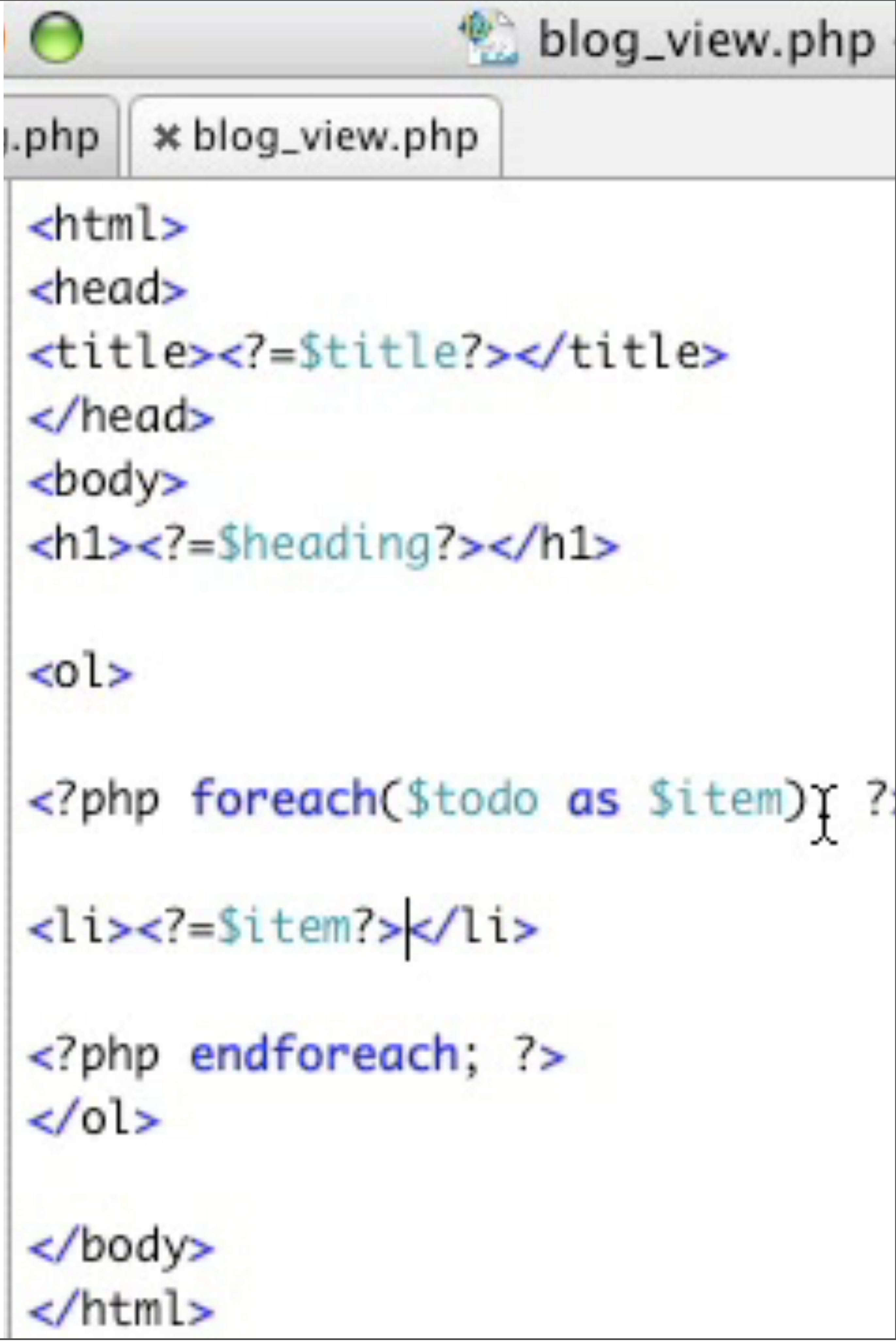
```
 http://www.codeigniter.com/user_guide/general/

-----------------------------------------------
RESERVED ROUTES
-----------------------------------------------

There are two reserved routes:

 $route['default_controller'] = 'welcome';

This route indicates which controller class shou
URI contains no data. In the above example, the
would be loaded.

 $route['scaffolding_trigger'] = 'scaffolding';

This route lets you set a "secret" word that wil
scaffolding feature for added security. Note: Sc
enabled in the controller in which you intend to


oute['default_controller'] = "welcome";

oute['scaffolding_trigger'] = "";

 Define your own routes below ------------------
```

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:

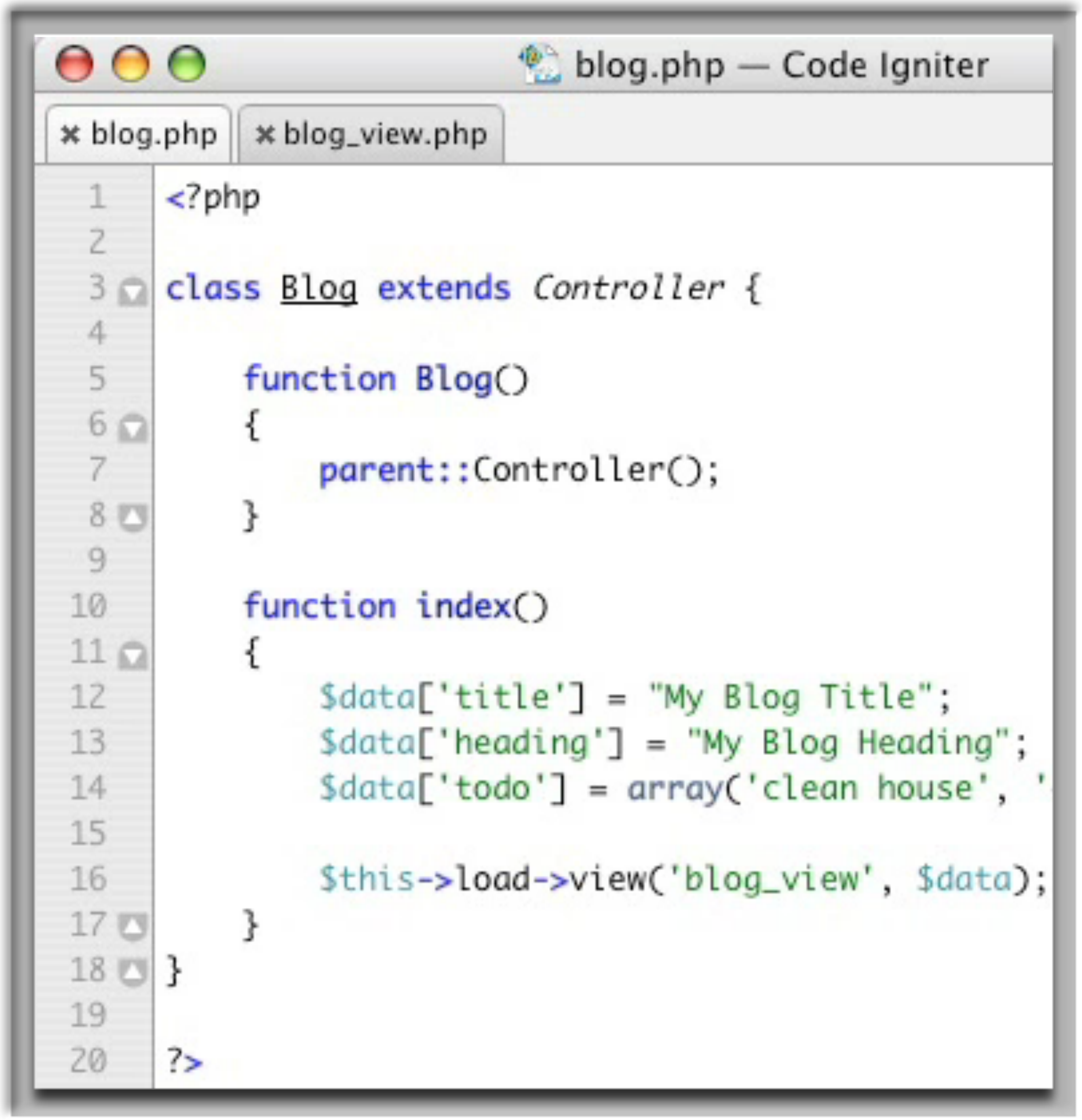
  $this->load->view('blog_view');

- There is no application logic in the views, only display logic (to some degree)
- **<?=** is short form for **<?php echo**

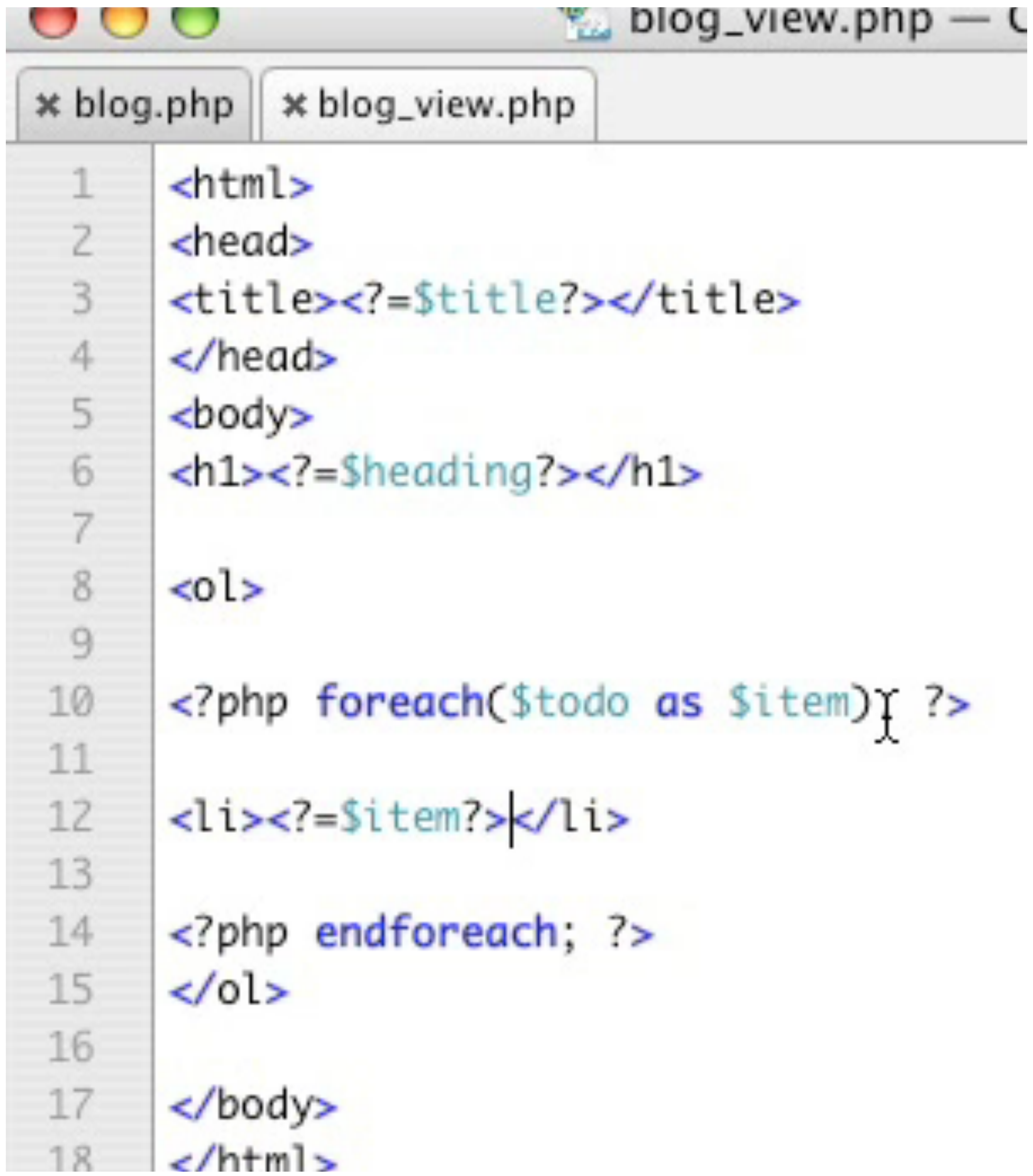


```
<html>
<head>
<title><?=$title?></title>
</head>
<body>
<h1><?=$heading?></h1>

<ol>

<?php foreach($todo as $item): ?

<li><?=$item?></li>

<?php endforeach; ?>
</ol>

</body>
</html>
```

Controller & View sample



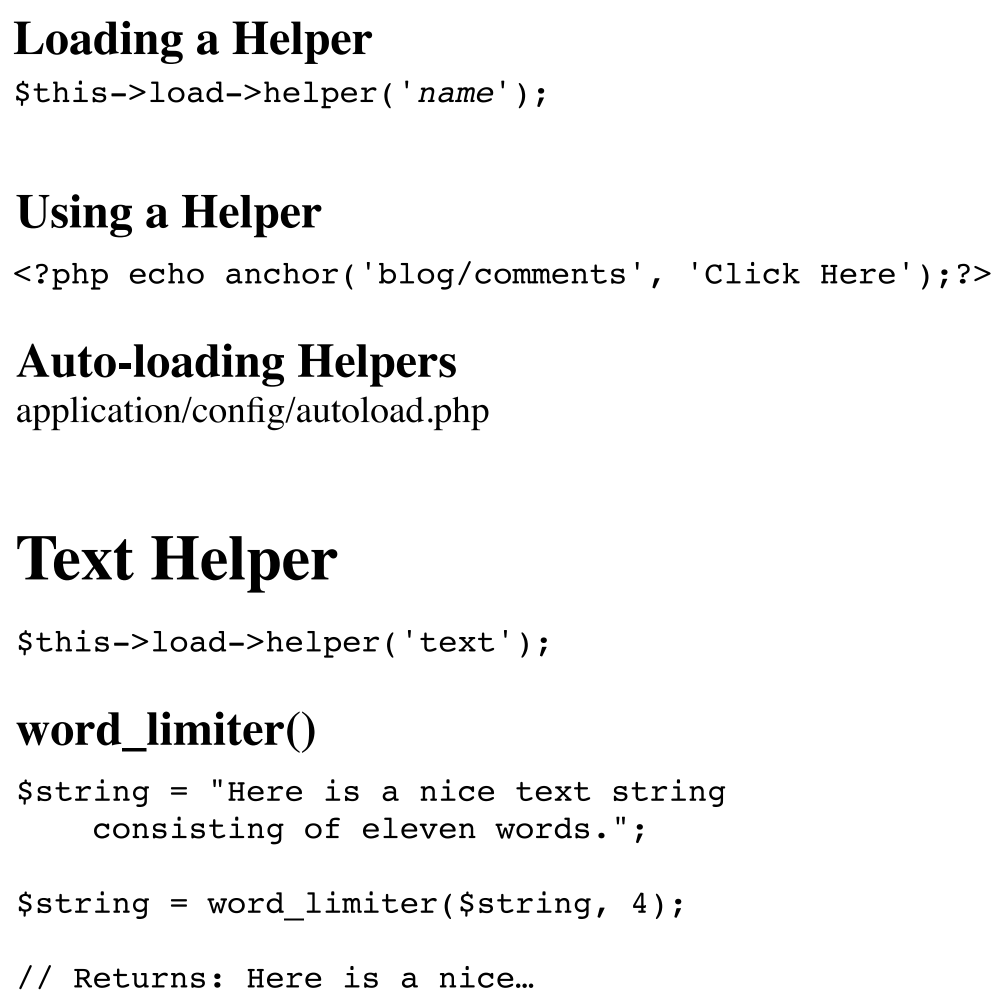
```php
<?php

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);
    }
}

?>
```



```html
<html>
<head>
<title><?=$title?></title>
</head>
<body>
<h1><?=$heading?></h1>

<ol>

<?php foreach($todo as $item): ?>

<li><?=$item?></li>

<?php endforeach; ?>
</ol>

</body>
</html>
```

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

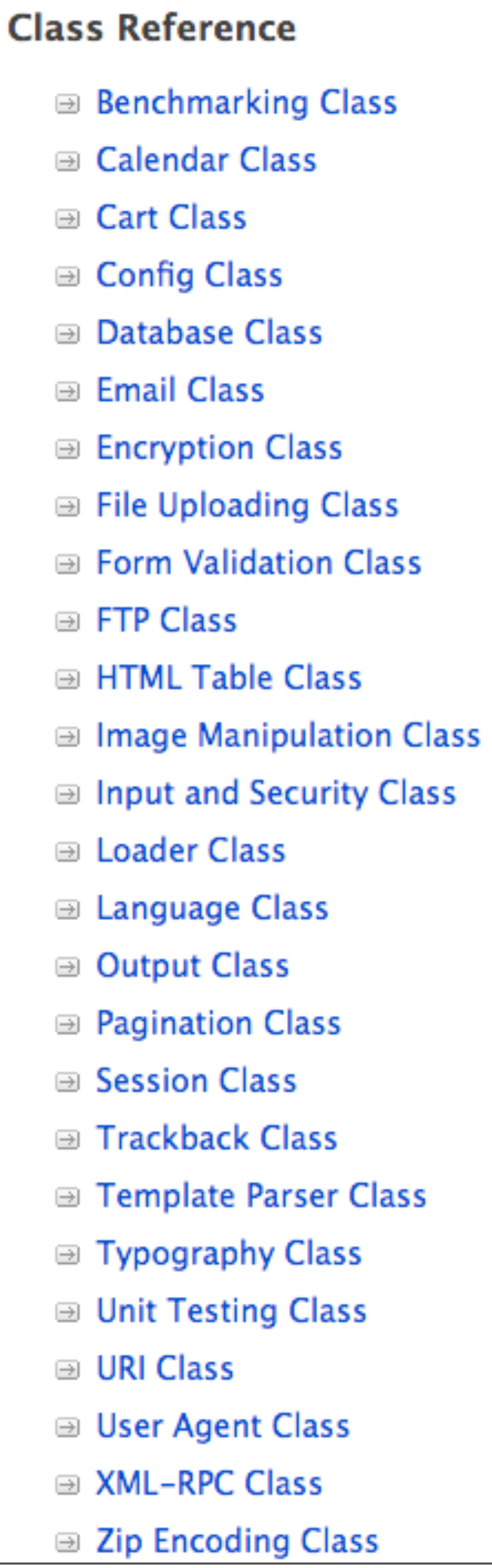
```
$this->load->helper('text');
```

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



Class Reference

- Benchmarking Class
- Calendar Class
- Cart Class
- Config Class
- Database Class
- Email Class
- Encryption Class
- File Uploading Class
- Form Validation Class
- FTP Class
- HTML Table Class
- Image Manipulation Class
- Input and Security Class
- Loader Class
- Language Class
- Output Class
- Pagination Class
- Session Class
- Trackback Class
- Template Parser Class
- Typography Class
- Unit Testing Class
- URI Class
- User Agent Class
- XML-RPC Class
- Zip Encoding Class

Database Handling

- CodeIgniter provides a simple way to access the **database**.
- It **encapsulates** the underlying database (MySQL, Oracle, etc)
- The **connection** handling is done by CodeIgniter 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

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

Standard Query With Multiple Results

```
$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;
}

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

Database Configuration

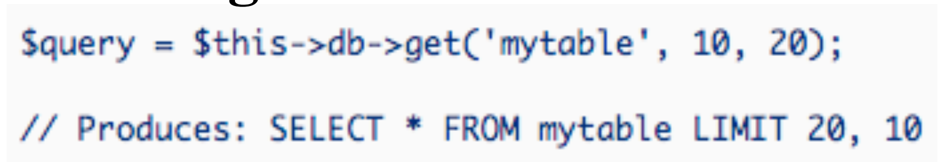
```
$db['test']['hostname'] = "localhost";
$db['test']['username'] = "root";
$db['test']['password'] = "";
$db['test']['database'] = "database_name";
$db['test']['dbdriver'] = "mysql";
$db['test']['dbprefix'] = "";
$db['test']['pconnect'] = TRUE;
$db['test']['db_debug'] = FALSE;
$db['test']['cache_on']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

Active Records

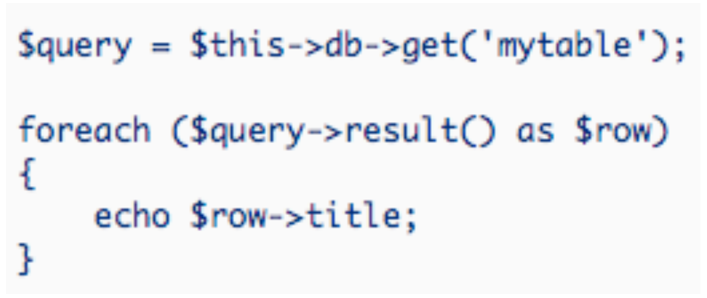
- CodeIgniter uses a modified version of the *Active Record Database* Pattern.

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

Selecting Data

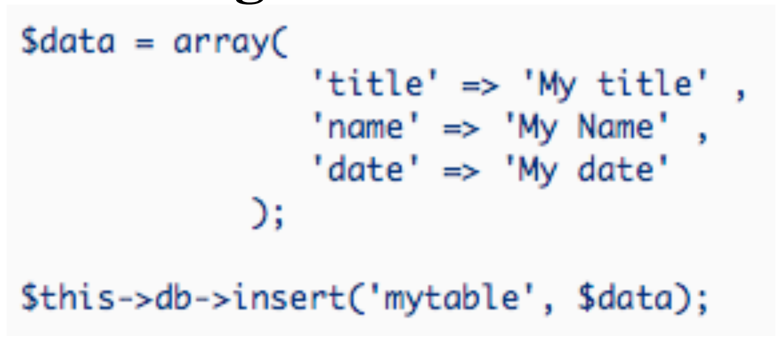


```
$query = $this->db->get('mytable', 10, 20);

// Produces: SELECT * FROM mytable LIMIT 20, 10
```



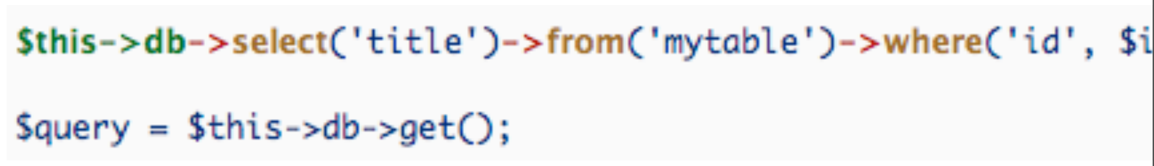
```
$query = $this->db->get('mytable');

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

Inserting Data



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

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

Method Chaining

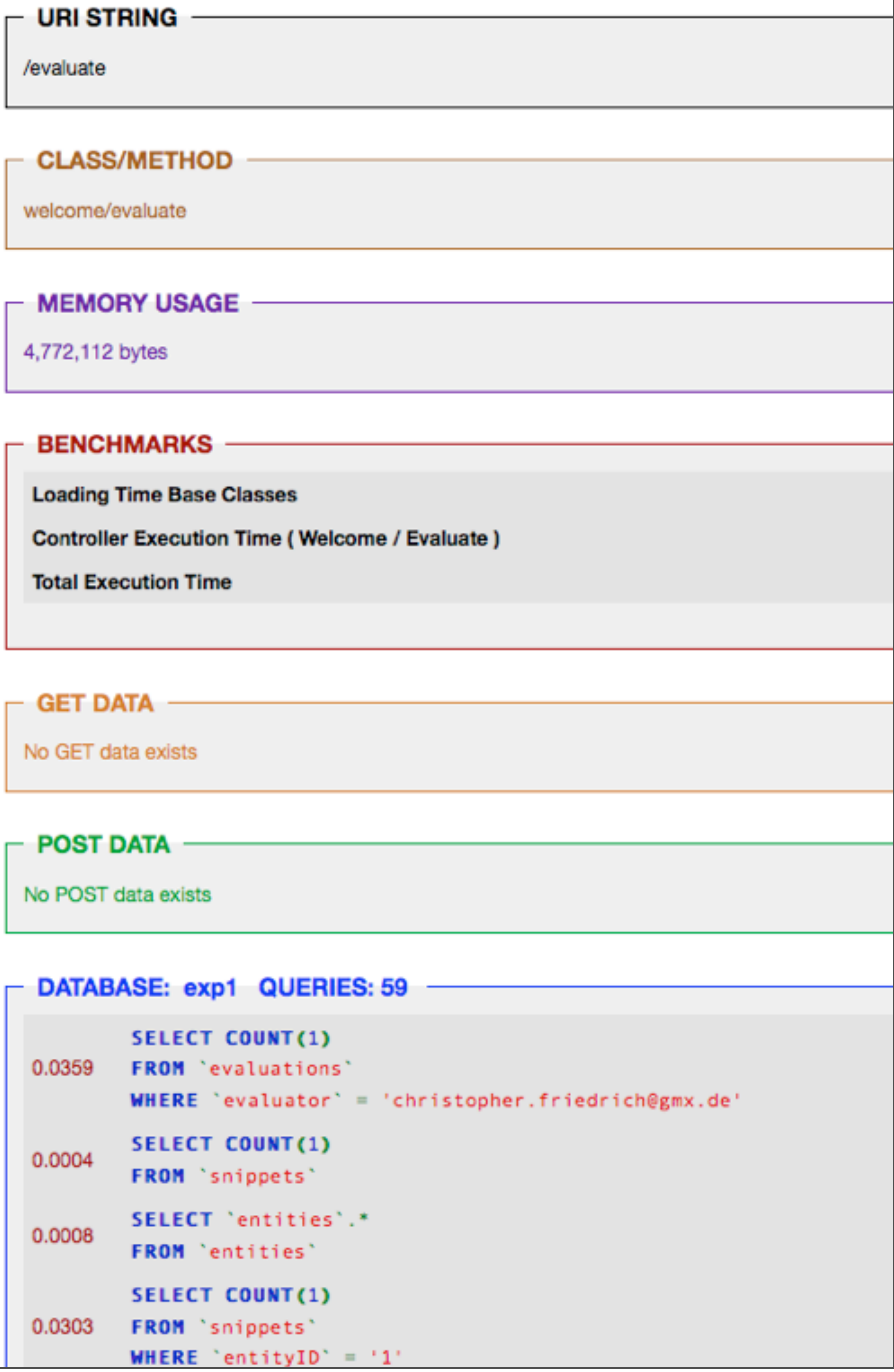


```
$this->db->select('title')->from('mytable')->where('id', $i

$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);
```



URI STRING

/evaluate

CLASS/METHOD

welcome/evaluate

MEMORY USAGE

4,772,112 bytes

BENCHMARKS

Loading Time Base Classes

Controller Execution Time (Welcome / Evaluate)

Total Execution Time

GET DATA

No GET data exists

POST DATA

No POST data exists

DATABASE: exp1 QUERIES: 59

```
0.0359  SELECT COUNT(1)
        FROM `evaluations`
        WHERE `evaluator` = 'christopher.friedrich@gmx.de'
0.0004  SELECT COUNT(1)
        FROM `snippets`
0.0008  SELECT `entities`.*
        FROM `entities`
0.0303  SELECT COUNT(1)
        FROM `snippets`
        WHERE `entityID` = '1'
```

Further Reading and Resources

- **CodeIgniter User Guide:**
  http://codeigniter.com/user_guide/index.html
- **Tutorials and Video Tutorials:**
  http://codeigniter.com/wiki/Category:Help::Tutorials
- **Model-View-Controller and CodeIgniter:**
  http://www.zenperfect.com/2007/07/12/model-view-controller-and-codeigniter/