PS03CMCA51 Web Application Frameworks (CodeIgniter Part)

Dr. J. V. Smart

Table of Contents

Syllabus

Syllabus with effect from the Academic Year 2021-2022

Course Code	PS03CMCA51	Title of the Course	WEB APPLICATION FRAMEWORKS
Total Credits of the Course	4	Hours per Week	4

Course Objectives:	1. To learn the fundamentals of the Python programming language. 2. To study development of procedural as well as object-

	oriented Python programs. 3. To learn GUI program development using Python. 4. To understand how to access files and databases from Python. 5. To learn client-side web application frameworks. 6. To learn server-side web application frameworks.

Course Content

Unit	Description	Weightage* (%)
1.	**Basic Web Application Development Tools** - Introduction to HTML5, CSS3 - Interactive web pages using JavaScript - The JQuery library - JavaScript user interface library	25
2.	**Web Frameworks for Python** - Introduction to web frameworks - Popular full-stack frameworks and non full-stack frameworks (microframeworks) - Working with Flask and Django frameworks.	25
3.	**Client-side Web Application Frameworks** - Setting up Project, project organization and management - Templates - MVC Architecture - Data binding - Dependency injection - Routing	25
4.	**Server-side Web Application frameworks** - Application structure - MVC Architecture - Routing - Helpers - Libraries - Form validation - Session management - Active record	25

Teaching-Learning Methodology	Blended learning approach incorporating traditional classroom teaching as well as online / ICT-based teaching practices

Evaluation Pattern

Sr. No.	Details of the Evaluation	Weightage
1.	Internal Written / Practical Examination (As per CBCS R.6.8.3)	15%
2.	Internal Continuous Assessment in the form of Practical, Viva-voce, Quizzes, Seminars, Assignments, Attendance (As per CBCS R.6.8.3)	15%
3.	University Examination	70%

Course Outcomes: Having completed this course, the learner will be able to	
1.	develop websites using Django Framework.
2.	manipulate different Python data types.
3.	develop object-oriented programs using Python.
4.	understand the Python package system.
5.	create basic GUI programs as well as Python programs with file handling and database access.

Suggested References:

Sr. No.	References
1.	Dane Cameron, “HTML5, JavaScript and jQuery”, Wrox publication.
2.	David Sawyer McFarland, “CSS3”, O’reilly.
3.	Brad Green and Syham Seshadri, “AngularJS”, O’Reilly.
4.	Python Web Frameworks by Carlos de la Guardia, O'Reilly Media, Inc., March 2016.
5.	Jake Spurlock, “Bootstrap”, O’Reilly.

Sr. No.	References
6.	Thomas Myer, “Professional CodeIgniter”, Wrox Professional Guides.
7.	Karl Swedberg, Jonathan Chaffer, “jQuery 1.4 Reference Guide”, PACKT publishing.
8.	Valeri Karpov, Diego Netto, “Professional AngularJS”, Wrox publication.
9.	Zak Ruvalcaba, Anne Boehm, “HTML5 and CSS3”, Murach.
10.	Bear Bibeault, Yehuda Katz, “jQuery in action”, 2nd edition, Dreamtech press.

On-line resources to be used if available as reference material	
1.	Python documentation.

Introduction to the CodeIgniter Server-side Web Application Framework

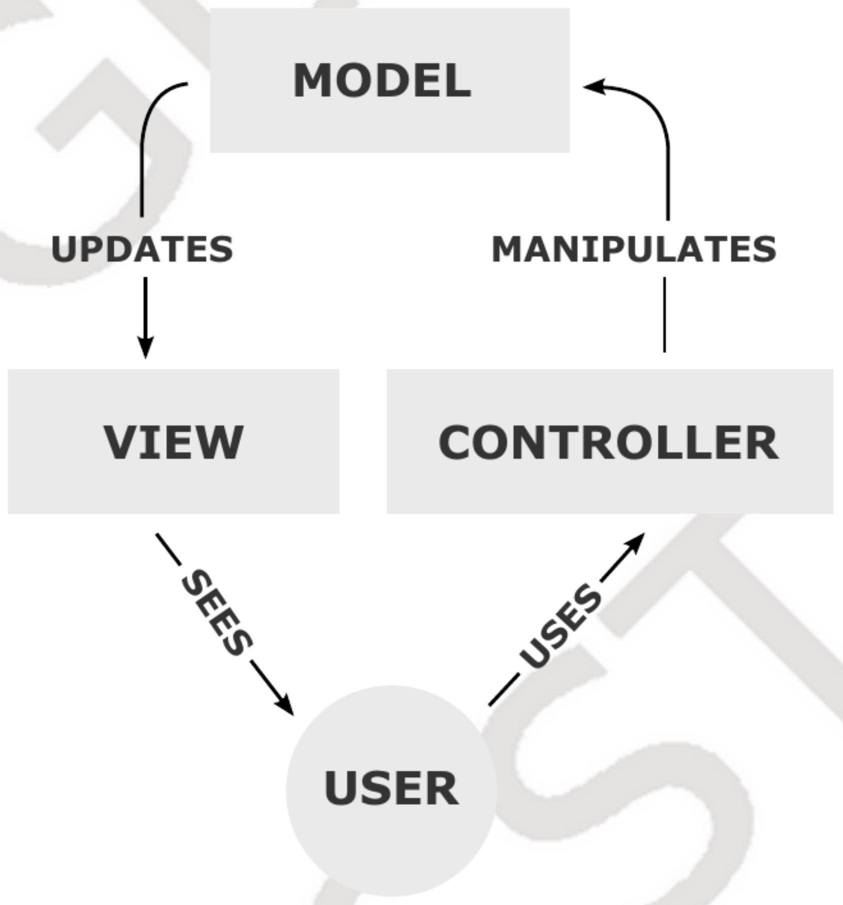
CodeIgniter is an open-source server-side web application framework for developing web applications using PHP.

CodeIgniter

The CodeIgniter Logo

MVC (Model-View-Controller)

MVC (Model-View-Controller) is an architectural pattern for software design. As the name suggests, an MVC application consists of three parts - Model, View and Controller.

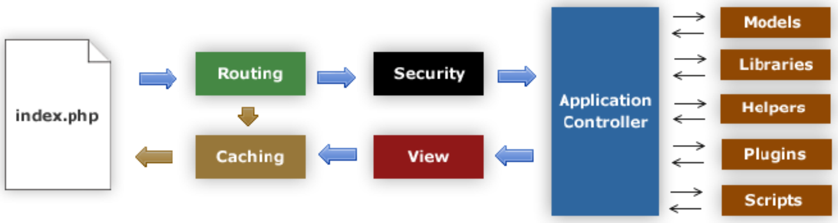


The MVC Pattern

- **Model** The model is responsible for storing and manipulating the data of the application. It is also responsible for implementing the business logic
- **View** The view is a representation of (part of) the data that is presented to the user. Same data may be presented in different form by different views (e.g. table v/s chart)
- **Controller** The controller accepts user input in the form of events, optionally validate it and then either sends it to the model for action or selects the appropriate view to be displayed. Often, it is also responsible for fetching the data needed by the view from the model

Major advantages of the MVC pattern include separation between presentation and logic, possibility of simultaneous independent development of the model, view and controller components and the ability to have multiple views for presenting the same data in different ways. MVC pattern was conceived for GUI application development, but it is heavily used for web application development. Different frameworks implement the pattern in different ways. Several software architectural patterns based on MVC have been developed.

CodeIgniter Application Flowchart



CodeIgniter Application Flowchart

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, 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.

CodeIgniter at a Glance

(From CodeIgniter Documentation)

CodeIgniter is an Application Framework

CodeIgniter is a toolkit for people who build web applications using PHP. Its goal is to enable you to develop projects much faster than you could if you were writing code from scratch, by providing a rich set of libraries for commonly needed tasks, as well as a simple interface and logical structure to access these libraries. CodeIgniter lets you creatively focus on your project by minimizing the amount of code needed for a given task.

CodeIgniter is Free

CodeIgniter is licensed under the MIT license so you can use it however you please. For more information please read the license agreement.

CodeIgniter is Light Weight

Truly light weight. The core system requires only a few very small libraries. This is in stark contrast to many frameworks that require significantly more resources. Additional libraries are loaded dynamically upon request, based on your needs for a given process, so the base system is very lean and quite fast.

CodeIgniter Uses M-V-C

CodeIgniter uses the Model-View-Controller approach, which allows great separation between logic and presentation. This is particularly good for projects in which designers are working with your template files, as the code these files contain will be minimized. We describe MVC in more detail on its own page.

CodeIgniter Generates Clean URLs

The URLs generated by CodeIgniter are clean and search-engine friendly. Rather than using the standard "query string" approach to URLs that is synonymous with dynamic systems, CodeIgniter uses a segment-based approach:

```
example.com/news/article/345
```

Note: By default the index.php file is included in the URL but it can be removed using a simple .htaccess file.

CodeIgniter Packs a Punch

CodeIgniter comes with full-range of libraries that enable the most commonly needed web development tasks, like accessing a database, sending email, validating form data, maintaining sessions, manipulating images, working with XML-RPC data and much more.

CodeIgniter is Extensible

The system can be easily extended through the use of your own libraries, helpers, or through class extensions or system hooks.

CodeIgniter Does Not Require a Template Engine

Although CodeIgniter does come with a simple template parser that can be optionally used, it does not force you to use one. Template engines simply can not match the performance of native PHP, and the syntax that must be learned to use a template engine is usually only marginally easier than learning the basics of PHP. Consider this block of PHP code:

```
<ul>
<?php foreach ($addressbook as $name):?>
        <li><?=$name?></li>
<?php endforeach; ?>
</ul>
```

Contrast this with the pseudo-code used by a template engine:

```
<ul>
{foreach from=$addressbook item="name"}
        <li>{$name}</li>
{/foreach}
</ul>
```

Yes, the template engine example is a bit cleaner, but it comes at the price of performance, as the pseudo-code must be converted back into PHP to run. Since one of our goals is maximum performance, we opted to not require the use of a template engine.

CodeIgniter is Thoroughly Documented

Programmers love to code and hate to write documentation. We're no different, of course, but since documentation is as important as the code itself, we are committed to doing it. Our source code is extremely clean and well commented as well.

CodeIgniter has a Friendly Community of Users

Our growing community of users can be seen actively participating in our Community Forums.

Installation

- Download CodeIgniter as a zip file from https://codeigniter.com
- Unzip the contents of the `framework-4.1.4` directory in the document root of the web server or a subdirectory
- If you use a subdirectory of the document root, modify `public/.htaccess`

file

Change the line

```
# RewriteBase /
```

to

```
RewriteBase /CodeIgniterDemo/
```

where `CodeIgniterDemo` is the subdirectory of the document root where you installed the framework.

- Modify `app/Config/App.php`

Change the line

```
public $baseURL = 'http://localhost:8080';
```

to

```
public $baseURL = 'http://localhost/CodeIgniterDemo';
```

where `CodeIgniterDemo` is the subdirectory of the document root where you installed the framework.

- Give write permission to all users / everyone on the `writable` directory
- Now you can open your demo app in the browser. By default, it shows a predefined welcome page
- To change the welcome page
    - Define a home view (page) in `app/Views/home.php`
    - In the home controller (`app/Controllers/Home.php`), modify the function `index()` to return `view('home')`
- Copy the sample environment file `env` to actual environment file `.env` . Add the line

```
CI_ENVIRONMENT = development
```

at the end of the `.env` file.

CodeIgniter Directory Structure

```
.
├── app
│   ├── Config
│       ├── App.php
│       ├── Database.php
│       └── Routes.php
│   ├── Controllers
│   ├── Database
│   ├── Helpers
│   ├── Libraries
│   ├── Models
│   └── Views
│       ├── errors
│       ├── pages
│       └── templates
├── public
├── system
└── writable
```

- **app** Contains the application files
  - **Config** Contains `App.php`, the main CodeIgniter app configuration file; and `Database.php`, the database configuration file
    - **App.php** The main CodeIgniter app configuration file
    - **Database.php** The database configuration file
    - **Routes.php** The CodeIgniter routing configuration file
  - **Controllers** Contains the controller classes
  - **Models** Contains the model classes. Usually, each model corresponds to a table in the database
  - **Views** Contains the view files
    - **pages** Contains static (fixed-content) pages
    - **templates** Contains templates like headers and footers that can be included in other pages
    - **errors** Contains the error-handling pages
- **public** Contains the main `index.php` file
- **system** Contains the CodeIgniter system files
- **writable** The web server must have write permission on this directory. It is used for log files, uploaded files, etc.

Sample Code

```php
# app/Config/Database.php
<?php
...
...
class Database extends Config
{
    ...
    ...
    public $default = [
        'hostname' => 'localhost',
        'username' => 'root',
        'password' => 'gdcst',
        'database' => 'classicmodels',
        'DBDriver' => 'MySQLi',
        'port'     => 3306,
        ...
        ...
    ];
    ...
    ...
}
```

```php
# app/Models/EmployeeModel.php
<?php
namespace App\Models;
use CodeIgniter\Model;

class EmployeeModel extends Model
{
    protected $table = 'employees';
    protected $primaryKey = 'employeeNumber';
    protected $allowedFields = [ 'employeeNumber', 'lastName',
         'firstName', 'extension', 'email', 'officeCode',
         'reportsTo', 'jobTitle' ];
}
```

```php
# app/Controllers/Employee.php
# A RESTful web service controller

<?php
...
...
class Employee extends ResourceController
{
    protected $modelName = 'App\Models\EmployeeModel';
    protected $format    = 'json';
...
...
```

```php
    // Retrieve list of resources
    // GET method
    public function index(){
        $model = new EmployeeModel();
        $data['employees'] = $model->orderBy('employeeNumber',
        'ASC')->findAll();
        return $this->respond($data);
    }

    // Insert a resource
    // POST method
    public function create() {
        $model = new EmployeeModel();

        // For JSON data
        $data = file_get_contents("php://input");
        $data = json_decode($data);

        $retval = $model->insert($data);
        if ($retval === False) {
            $response = 'ERROR: The employee was not created';
        } else {
            $response = 'Success: Employee created successfully';
        }
        return $this->respondCreated($response);
    }

    // Retrieve a single resource
    // GET method
    public function show($employeeNumber = null){
        $model = new EmployeeModel();
        $data = $model->where('employeeNumber',
        $employeeNumber)->first();
        if ($data) {
            return $this->respond($data);
        } else {
            return $this->failNotFound('No employee found');
        }
    }

    // Update a resource
    // PUT method
    public function update($employeeNumber = null){
        $model = new EmployeeModel();

        // For JSON data
        $data = file_get_contents("php://input");
        $data = json_decode($data);

        $model->update($employeeNumber, $data);

        $response = 'Success: Employee updated successfully';
        return $this->respond($response);
    }

    // Delete a resource
    // DELETE method
    public function delete($employeeNumber = null){
        $model = new EmployeeModel();
        $data = $model->where('employeeNumber',
        $employeeNumber)->delete($employeeNumber);
        if ($data) {
            $response = 'Employee successfully deleted';
            return $this->respondDeleted($response);
        } else {
            return $this->failNotFound('No employee found');
        }
    }

}
```

```
# app/Views/pages/about.php
# A static page
<h3>About CodeIgniter</h3>
<div>CodeIgniter is ... </div>
```

```
# app/Views/templates/header.php
# A template for a page header
<!doctype html>
<html>
<head>
    <title>CodeIgniter Tutorial</title>
</head>
<body>

    <h1><?= esc($title) ?></h1>
```

```
# app/Views/templates/footer.php
# A template for a page footer
    <em>&copy; 2021</em>
</body>
</html>
```

```
# app/Views/news/overview.php
# a sample view for a list of news
<h2><?= $title ?></h2>

<?php
```

```
foreach ($news as $news_item) {
?>
    <h3><?= $news_item['title'] ?></h3>
    <div class="main">
        <?= $news_item['body'] ?>
    </div>
<?php
}
?>
```

```
# app/Config/Routes.php

<?php

...
...
$routes->setDefaultController('Home');
$routes->setDefaultMethod('index');

$routes->get('/', 'Home::index');
$routes->get('news/(:segment)', 'News::view/$1');
$routes->get('news', 'News::index');
$routes->get('(:any)', 'Pages::view/$1');
```

Helpers

Helpers are collections of useful procedural functions.

- Helpers
    - Array Helper
    - Date Helper
    - Filesystem Helper
    - Form Helper
    - HTML Helper
    - Number Helper
    - Test Helper
    - Text Helper
    - URL Helper
    - XML Helper

Libraries

CodeIgniter comes with a set of built-in libraries for common tasks.

- Libraries
  - Caching
  - Cookies
  - Email
  - Encryption
  - Image Manipulation
  - Pagination
  - Security
  - Session handling
  - Validation

Active Record

Active record is a software architectural pattern. It is used when using Object-Oriented Programming with RDBMS.

```
RDBMS:

Table:

-------------------------
|    |      |     |      |
-------------------------
|    |      |     |      |
-------------------------
|    |      |     |      |
-------------------------
|    |      |     |      |
-------------------------
|    |      |     |      |
-------------------------

A table in the RDBMS is mapped to a class.

Each row corresponds to an object.

CRUD functionality on a table is implemented as methods on the
corresponding class.

Table: News => class News
```

```
     | ID | Title     | Text  | Slug      |
     --------------------------------------
Row: | 1  | QQQQ WWWW | asdfg | qqqq-wwww |  => object of class
News
Row: | 2  | RRRR      | zzzzz | rrrr      |  => object of class
News
Row: | 3  | TTTT      | ggggg | tttt      |  => object of class
News

CRUD:
    Create (INSERT)   => News->add(): creates new object and
inserts a row in the table
    Retrieve (SELECT) => News->get(): fetches one or more rows
from the table as objects
    Update (UPDATE)   => News->update(): updates both the table
and the object
    Delete (DELETE)   => News->delete(): deletes the row in the
table and frees the object
```