



**Time to be
Updated**
CI3 to CI4 or Laravel

Software Code Base Update

Codeigniter3 To Codeigniter4 or Laravel

MsM Robin
Software Engineer

Datasoft Systems Limited Bangladesh

Codeigniter 3 is a lighter framework with some limited features. It does not support the MVC architecture but HMVC (Hierarchical Model-View-Controller). In contrast, Codeigniter4 is also a lighter and more modern framework than CI3 with more features especially managing packages/tools through composer. Also, CI4 is a complete rewrite of the CI3 framework, supports full features of MVC, and is mostly inspired by Laravel architecture.

As a result, if we want to update our CI3 modules into CI4/Laravel there are some particular sections we need to keep in mind:

1. Configurations

This would be the major concern if we are eager to update CI3 modules into CI4/Laravel. Just because all these 3 frameworks follow different architectures compared to each other.

CodeIgniter 3:

```
php Copy code

// application/config/config.php
$config['base_url'] = 'http://localhost/ci3app/';
$config['encryption_key'] = 'your_encryption_key';
```

CodeIgniter 4:

```
php Copy code

// app/Config/App.php
public $baseURL = 'http://localhost/ci4app/';
public $encryptionKey = 'your_encryption_key';
```

Laravel:

```
env Copy code

// .env
APP_URL=http://localhost/laravelapp
APP_KEY=base64:your_encryption_key
```

Also, CI4 supports the .env file structure as laravel to fulfill the purpose of the configuration.

1.1 Base Controller

Updating a CodeIgniter 3 (CI3) project to CodeIgniter 4 (CI4) or Laravel, one of the key areas that changes is the controller structure. Below are examples demonstrating migrating the base controller from CI3 to CI4 and Laravel.

- **CI3:** Uses a basic structure without strict conventions.
- **CI4/Laravel:** Enforces a more structured approach with a base controller class. For instance, in CI4, you might extend BaseController, whereas, in Laravel, you typically extend Controller.

Codeigniter3 (CI3):

```

php                                                                    Copy code

// application/controllers/BaseController.php
class BaseController extends CI_Controller
{
    public function __construct()
    {
        parent::__construct();
        // Common functionality for all controllers
    }

    // Additional methods shared across multiple controllers
}

```

Codeigniter4 (CI4):

```

base_controller_ci4

<?php namespace App\Controllers;

use App\Models\DropDownModel;
use CodeIgniter\Controller;
use CodeIgniter\HTTP\CLIRequest;
use Config\Services;
use Psr\Log\LoggerInterface;


abstract class BaseController extends Controller
{
    protected $request;
    protected DropDownModel $dropdown;
    protected $validator = null;
    protected $session;
    protected $active_user;
    protected $helpers = ['minifier', 'form', 'number'];

    public function initController($request, $response, LoggerInterface $logger)
    {
        // Do Not Edit This Line
        parent::initController($request, $response, $logger);
        // Preload any models, libraries, etc, here.
        $this->session = Services::session();
        // Base instances
        $this->dropdown = new DropDownModel();
        $this->active_user = session('user_id');
    }
}

```

Laravel:

php

 Copy code

```
// app/Http/Controllers/Controller.php
namespace App\Http\Controllers;

use Illuminate\Foundation\Auth\Access\AuthorizesRequests;
use Illuminate\Foundation\Bus\DispatchesJobs;
use Illuminate\Foundation\Validation\ValidatesRequests;
use Illuminate\Routing\Controller as BaseController;

class Controller extends BaseController
{
    use AuthorizesRequests, DispatchesJobs, ValidatesRequests;

    // Additional methods shared across multiple controllers
}
```

1.2 Database configurations

Database configuration is a crucial aspect of web applications, determining how the application connects to and interacts with the database. CI3, CI4, and Laravel offer different approaches and capabilities for handling database configurations.

CodeIgniter 3 (CI3):

Configuration File: In CI3, database configurations are defined in a single configuration file located at `application/config/database.php`. The setup is straightforward, but it's somewhat limited in terms of flexibility and features.

Environment Handling: CI3 allows for different database configurations based on the environment (e.g., development, production), but it lacks the advanced features found in more modern frameworks.

Example: CI3 Database Configuration (`application/config/database.php`)

```
php Copy code

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

$db['default'] = [
    'dsn'        => '',
    'hostname'   => 'localhost',
    'username'   => 'root',
    'password'   => '',
    'database'   => 'ci3_database',
    'dbdriver'   => 'mysqli',
    'dbprefix'   => '',
    'pconnect'   => FALSE,
    'db_debug'   => (ENVIRONMENT !== 'production'),
    'cache_on'   => FALSE,
    'cachedir'   => '',
```

CodeIgniter 4 (CI4):

- **Configuration Flexibility:** CI4 introduces a more modular and flexible way to configure databases. Configurations are now stored in a dedicated configuration file within the `Config` namespace.
- **Environment-Specific Configurations:** CI4 supports `.env` files, allowing developers to define database configurations for different environments (development, testing, production) in a more secure and manageable way.

Example: CI4 Database Configuration (`app/Config/Database.php`)

```
php Copy code

namespace Config;

use CodeIgniter\Database\Config as BaseConfig;

class Database extends BaseConfig
{
    public $default = [
        'DSN'      => '',
        'hostname' => getenv('database.default.hostname'),
        'username' => getenv('database.default.username'),
        'password' => getenv('database.default.password'),
        'database' => getenv('database.default.database'),
        'DBDriver' => 'MySQLi',
        'DBPrefix' => '',
        'pConnect' => false,
        'DBDebug'  => (ENVIRONMENT !== 'production'),
        'charset'  => 'utf8mb4',
        'DBCollat' => 'utf8mb4_general_ci',
    ];
}
```

Laravel:

- **Configuration Power:** Laravel's database configuration system is highly flexible and powerful. It supports multiple database connections, connection pooling, read/write splitting, and more. Configurations are typically defined in `config/database.php` and can be dynamically handled via `.env` files.
- **Environment Handling:** Laravel relies heavily on the `.env` file for configuring database settings, making it easy to manage different configurations for various environments (e.g., local, staging, production).

Example: Laravel Database Configuration (`config/database.php`)

```
php Copy code

return [

    'default' => env('DB_CONNECTION', 'mysql'),

    'connections' => [

        'mysql' => [
            'driver' => 'mysql',
            'host' => env('DB_HOST', '127.0.0.1'),
            'port' => env('DB_PORT', '3306'),
            'database' => env('DB_DATABASE', 'laravel_database'),
            'username' => env('DB_USERNAME', 'root'),
            'password' => env('DB_PASSWORD', ''),
        ],
    ],
];
```

1.3 Filters

CodeIgniter 3 (CI3): In CI3, form validation and request filtering are typically handled using the **Form Validation** library. While CI3 does not have a dedicated filter system, you can implement pre-processing and validation logic within your controller methods or use hooks. The code you provided is an example of how form data is validated using CI3's Form Validation library. Although not exactly a "filter," this approach is a common way to validate and sanitize input data in CI3.

```
Filters in CodeIgniter3

class EmployeeController extends CI_Controller
{
    /**
     * To validate all the form data ...
     * @return bool
     */
    public function _get_validated_data()
    {
        $this->load->library('form_validation');

        $this->form_validation->set_rules('emp_name', 'Employee Name', 'required');
        $this->form_validation->set_rules('email', 'Email', 'required|
callback_check_valid_email');
        $this->form_validation->set_rules('phone', 'Phone', 'required|
callback_check_valid_phone_number');
        $this->form_validation->set_rules('salary', 'Salary', 'required');
        $this->form_validation->set_rules('address', 'Address', 'required');

        if ($this->form_validation->run() === FALSE) {
            $this->session->set_flashdata('error', validation_errors());
            return false;
        }

        return true;
    }

    /**
     * Check valid phone number ...
     * @return bool
     */
    public function check_valid_phone_number($phone)
    {
        if (!preg_match('/^[0-9]{11}$/', $phone)) {
            $this->form_validation->set_message('check_valid_phone_number', 'Invalid phone
number format');
            return false;
        }
        return true;
    }

    /**
     * Check valid email ...
     * @return bool
     */
    public function check_valid_email($email)
    {
        if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
            $this->form_validation->set_message('check_valid_email', 'Invalid email format');
            return false;
        }
        return true;
    }
}
```

In this example:

- `_get_validated_data()` method handles the overall form validation. It uses callbacks like `check_valid_email()` and `check_valid_phone_number()` for custom validation logic.
- Custom validation messages are set within these callback methods.

While CI3 doesn't have a native filter system, you can still achieve similar results by organizing validation logic in controller methods and using callbacks for custom validation.

CodeIgniter 4 (CI4):

CI4 introduces a more structured and flexible **filter system** that can be used for pre-and post-processing of requests. Filters are configured in the `app/Config/Filters.php` file and can be applied globally, per-controller, or per-method.

```
Filters in Codeigniter4

// app/Filters/ValidationFilter.php
<?php
namespace App\Filters;

use CodeIgniter\HTTP\RequestInterface;
use CodeIgniter\HTTP\ResponseInterface;
use CodeIgniter\Filters\FilterInterface;

class ValidationFilter implements FilterInterface
{
    public function before(RequestInterface $request, $arguments = null)
    {
        $validation = \Config\Services::validation();
        $validation->setRules([
            'emp_name' => 'required',
            'email'    => 'required|valid_email',
            'phone'    => 'required|regex_match[/^[0-9]{11}$/]/',
            'salary'  => 'required',
            'address'  => 'required'
        ]);

        if (!$validation->withRequest($request)->run()) {
            return redirect()->back()->withInput()->with('errors', $validation->getErrors());
        }
    }

    public function after(RequestInterface $request, ResponseInterface $response, $arguments = null)
    {
        // Post-processing logic can go here
    }
}
```



```

To apply the filter in Filters.php:
<?php
namespace Config;

use CodeIgniter\Config\BaseConfig;

class Filters extends BaseConfig
{
    public $aliases = [
        'validation' => \App\Filters\ValidationFilter::class,
    ];

    public $globals = [
        'before' => [
            'validation' => ['except' => ['login/*', 'register/*']],
        ],
    ];
}

```

Laravel: Utilizes middleware for handling request filtering, offering a more granular and powerful approach.

```

filters_laravel

// app/Http/Requests/EmployeeRequest.php

namespace App\Http\Requests;

use Illuminate\Foundation\Http\FormRequest;

class EmployeeRequest extends FormRequest
{
    /**
     * Determine if the user is authorized to make this request.
     *
     * @return bool
     */
    public function authorize()
    {
        // You can add authorization logic here if needed
        return true;
    }

    /**
     * Get the validation rules that apply to the request.
     *
     * @return array
     */
    public function rules()
    {
        return [
            'emp_name' => 'required|string|max:255',
            'email' => 'required|email|unique:employees,email',
            'phone' => 'required|regex:/^[0-9]{11}$//',
            'salary' => 'required|numeric|min:0',
            'address' => 'required|string|max:500',
        ];
    }

    /**
     * Get the custom messages for validator errors.
     *
     * @return array
     */
    public function messages()
    {
        return [
            'emp_name.required' => 'Employee name is required.',
            'email.required' => 'Email is required.',
            'email.email' => 'Please provide a valid email address.',
            'phone.required' => 'Phone number is required.',
            'phone.regex' => 'Invalid phone number format. It should be 11 digits.',
            'salary.required' => 'Salary is required.',
            'salary.numeric' => 'Salary must be a number.',
            'address.required' => 'Address is required.',
        ];
    }
}

```

1.4 Session and Token Management

- **CI3:** Basic session handling with limited token management.

```
session_ci3

// Loading the session library
$this->load->library('session');

// Setting session data
$this->session->set_userdata('username', 'john_doe');

// Getting session data
$username = $this->session->userdata('username');

// Destroying a session
$this->session->sess_destroy();
```

Example: Basic CSRF Protection CI3 provides basic CSRF protection, which can be enabled in the configuration file:

```
php Copy code

// application/config/config.php

$config['csrf_protection'] = TRUE;
$config['csrf_token_name'] = 'csrf_test_name';
$config['csrf_cookie_name'] = 'csrf_cookie_name';
```

- **CI4:** Improved session management with better security features.

Example: Session Handling

```
php Copy code

// Setting session data
session()->set('username', 'john_doe');

// Getting session data
$username = session()->get('username');

// Removing session data
session()->remove('username');

// Destroying a session
session()->destroy();
```

Example: CSRF Protection

In CI4, CSRF protection is enabled by default and is automatically integrated with forms and AJAX requests.

```
php Copy code

// CSRF token is automatically added to forms
<form method="post" action="/submit">
    <?= csrf_field() ?>
    <!-- Other form fields -->
</form>
```

Token Management: CI4 has better built-in support for token management, including support for JWT (JSON Web Tokens) via third-party libraries, but it still requires manual implementation for advanced token-based systems.

- **Laravel:** Provides comprehensive session & token management, including out-of-the-box CSRF protection & advanced API token management through tools like Passport & Sanctum.

Example: Session Handling

```
php Copy code

// Setting session data
session(['username' => 'john_doe']);

// Getting session data
$username = session('username');

// Removing session data
session()->forget('username');

// Destroying a session
session()->flush();
```

Example: CSRF Protection

Laravel automatically includes CSRF protection for all POST, PUT, PATCH, and DELETE requests.

```
php Copy code

// CSRF token is automatically added to forms
<form method="POST" action="/submit">
    @csrf
    <!-- Other form fields -->
</form>
```

Token Management: Laravel offers robust token management, particularly for API development, through tools like **Laravel Passport** and **Laravel Sanctum**.

Example: API Tokens with Laravel Sanctum Laravel Sanctum provides a lightweight authentication system for SPAs (single-page applications), simple APIs, and mobile applications. It allows you to issue API tokens to your users without the overhead of OAuth.

```
php Copy code

// Generating an API token
$token = $user->createToken('my-app-token')->plainTextToken;

// Protecting routes with Sanctum middleware
Route::middleware('auth:sanctum')->get('/user', function (Request $request) {
    return $request->user();
});
```

1.5 Usages of Constants

- **CI3:** Constants are often defined in configuration files or manually within code.

```
php Copy code

// application/config/constants.php

defined('BASEPATH') OR exit('No direct script access allowed');

define('SITE_NAME', 'MyWebsite');
define('API_KEY', 'your-api-key-here');
define('DB_TABLE', 'users');
```

- **CI4/Laravel:** Encourages the use of environment files (.env) for defining constants and configuration settings, promoting better security and flexibility.

Example: Defining Constants in CI4

```
php Copy code

// .env

app.baseURL = 'http://localhost:8080'
database.default.hostname = 'localhost'
database.default.database = 'my_database'
database.default.username = 'root'
database.default.password = ''
```

```
ci4_constant

// app/Config/Constants.php

namespace Config;

class Constants
{
    const SITE_NAME = 'MyWebsite';
    const API_KEY = 'your-api-key-here';
}

// In a controller
use Config\Constants;

class ExampleController extends BaseController
{
    public function index()
    {
        echo 'Welcome to ' . Constants::SITE_NAME;
        // Accessing environment variables
        echo 'Base URL: ' . env('app.baseURL');
    }
}
```

Laravel:

Example: Defining Constants in Laravel

```
env Copy code

// .env

APP_NAME=MyWebsite
API_KEY=your-api-key-here
DB_TABLE=users
```

```
laravel_constant

// config/app.php

return [
    'name' => env('APP_NAME', 'Laravel'),
    'api_key' => env('API_KEY', ''),
    'db_table' => env('DB_TABLE', 'users'),
];

// In a controller
use Illuminate\Support\Facades\Config;

class ExampleController extends Controller
{
    public function index()
    {
        echo 'Welcome to ' . config('app.name');
        echo 'API Key: ' . config('app.api_key');
        $users = DB::table(config('app.db_table'))->get();
    }
}
```

2. Routes

Routing is a crucial aspect of web frameworks, determining how URLs are mapped to controllers and methods. Routing differs significantly between CI3, CI4, and Laravel. Each version of CodeIgniter and Laravel offers distinct approaches to routing, reflecting the evolution of web application development.

- **CI3:** Primarily uses auto-routing, which can be less secure and less flexible.
- **CI4:** Introduces a more structured routing system with support for both manual and auto-routing.
- **Laravel:** Offers a highly flexible and powerful routing system, including named routes, route groups, and middleware.

CodeIgniter 3 (CI3):

- **Auto-Routing:** CI3 primarily relies on a conventional, auto-routing system where URLs map directly to controller classes and their methods. While simple and convenient, this approach can lead to security vulnerabilities and lacks flexibility.
- **Manual Routing:** CI3 also supports manual routing through the `routes.php` configuration file, allowing developers to define specific routes for better control.

Example: Auto-Routing in CI3

plaintext

Copy code

```
http://example.com/index.php/controller/method/param1/param2
```

In this example, the URL automatically maps to `Controller::method($param1, $param2)`.

Example: Manual Routing in CI3

php

Copy code

```
// application/config/routes.php

$route['default_controller'] = 'welcome';
$route['404_override'] = '';
$route['translate_uri_dashes'] = FALSE;

// Custom route
$route['products/(:any)'] = 'catalog/product_lookup/$1';
```

Here, a URL like `http://example.com/products/shoes` will route to the `Catalog::product_lookup('shoes')` method.

CodeIgniter 4 (CI4):

- **Structured Routing:** CI4 introduces a more structured and secure routing system. Developers can define routes explicitly, and auto-routing is disabled by default, reducing security risks.
- **Manual Routing:** CI4 encourages manual routing for better security and control, supporting complex routing configuration.

Example: Routing in CI4

```
php Copy code

// app/Config/Routes.php

$routes->get('/', 'Home::index');
$routes->get('products/(:any)', 'Catalog::productLookup/$1');
$routes->post('submit', 'Form::submit');

// Named routes
$routes->get('profile', 'User::profile', ['as' => 'user.profile']);
```

Auto-Routing: CI4 supports auto-routing, but it's not recommended due to security concerns. You can enable it explicitly:

```
php Copy code

$routes->setAutoRoute(true);
```

Route Groups: CI4 allows grouping of routes, applying filters (like authentication), and setting namespaces for better organization.

Laravel:

Highly Flexible Routing System: Laravel's routing system is both powerful and intuitive, providing extensive features like named routes, route groups, middleware, and more.

Expressive Syntax: Laravel's routing syntax is designed to be simple yet expressive, allowing developers to define routes with ease.

```
laravel_routes

// routes/web.php

use App\Http\Controllers\HomeController;
use App\Http\Controllers\ProductController;

// Basic route
Route::get('/', [HomeController::class, 'index']);

// Route with parameters
Route::get('products/{id}', [ProductController::class, 'show']);

// Named route
Route::get('user/profile', [UserController::class, 'profile'])->name('profile');

// Route with middleware
Route::get('dashboard', [DashboardController::class, 'index'])->middleware('auth');
```

- **Named Routes:** Allows for easy route generation and redirection using names instead of URIs.

```
php Copy code

return redirect()->route('profile');
```

- **Route Groups:** Routes can be grouped together, sharing common attributes like middleware, namespaces, or prefixes.

```
php Copy code

Route::middleware(['auth'])->group(function () {
    Route::get('dashboard', [DashboardController::class, 'index']);
    Route::get('settings', [SettingsController::class, 'index']);
});
```

3. Magic method to Constructors

In CI3 Controllers and Models are to load/call/invoke by its magic method called load. But in CI4/Laravel can be called. In CodeIgniter 3, it's not straightforward to call a method of one controller from another controller directly due to the way the framework is designed. However, there are a few ways you can achieve similar functionality:

1. **Libraries:** Common code can be moved to libraries.
2. **Helpers:** Common functions can be moved to helpers.
3. **Models:** Business logic can be moved to models.
4. **Modular Separation:** Use HMVC (Hierarchical Model-View-Controller) to enable modular separation.

Whereas CI4/Laravel follows the same approach to create a class instance directly on the construction or a method.

CodeIgniter 3:

```
php Copy code

// application/controllers/Welcome.php
class Welcome extends CI_Controller {
    public function __construct() {
        parent::__construct();
        $this->load->model('user_model');
    }

    public function index() {
        $data['users'] = $this->user_model->get_users();
        $this->load->view('welcome_message', $data);
    }
}
```

Laravel:

```
php Copy code

// app/Http/Controllers/WelcomeController.php
namespace App\Http\Controllers;

use App\Models\User;

class WelcomeController extends Controller {
    protected $user;

    public function __construct(User $user) {
        $this->user = $user;
    }

    public function index() {
        $data['users'] = $this->user->all();
        return view('welcome_message', $data);
    }
}
```

3.1 Calling/Invoking Another Controller to another.

CodeIgniter 4:

```
php Copy code

// app/Controllers/Welcome.php
namespace App\Controllers;

use App\Models\UserModel;

class Welcome extends BaseController {
    protected $userModel;

    public function __construct() {
        $this->userModel = new UserModel();
    }

    public function index() {
        $data['users'] = $this->userModel->findAll();
        return view('welcome_message', $data);
    }
}
```

CodeIgniter 4:

```
php Copy code

// app/Controllers/AnotherController.php
namespace App\Controllers;

class AnotherController extends BaseController {
    public function index() {
        // Logic here
    }
}

// app/Controllers/Welcome.php
namespace App\Controllers;

class Welcome extends BaseController {
    public function callAnother() {
        $anotherController = new AnotherController();
        $anotherController->index();
    }
}
```

3.2 Calling/Invoking Another model to controller/model.

CodeIgniter 3:

```
php Copy code

// application/models/Another_model.php
class Another_model extends CI_Model {
    public function get_data() {
        return $this->db->get('table')->result();
    }
}

// application/controllers/Welcome.php
class Welcome extends CI_Controller {
    public function index() {
        $this->load->model('another_model');
        $data = $this->another_model->get_data();
        // Logic here
    }
}
```

CodeIgniter 4:

```
php Copy code

// app/Models/AnotherModel.php
namespace App\Models;

use CodeIgniter\Model;

class AnotherModel extends Model {
    protected $table = 'table';

    public function getData() {
        return $this->findAll();
    }
}

// app/Controllers/Welcome.php
namespace App\Controllers;

use App\Models\AnotherModel;


class Welcome extends BaseController {
    protected $anotherModel;

    public function __construct() {
        $this->anotherModel = new AnotherModel();
    }

    public function index() {
        $data = $this->anotherModel->getData();
        // Logic here
    }
}
```

Laravel:

php

 Copy code

```
// app/Models/AnotherModel.php
namespace App\Models;

use Illuminate\Database\Eloquent\Model;

class AnotherModel extends Model {
    protected $table = 'table';
}

// app/Http/Controllers/WelcomeController.php
namespace App\Http\Controllers;

use App\Models\AnotherModel;

class WelcomeController extends Controller {
    protected $anotherModel;

    public function __construct(AnotherModel $anotherModel) {
        $this->anotherModel = $anotherModel;
    }

    public function index() {
        $data = $this->anotherModel->all();
        // Logic here
    }
}
```

4. Query builder methods:

The query builder is an essential component in web frameworks, allowing developers to interact with the database abstractly and fluently. The query builder methods in CodeIgniter 3 (CI3), CodeIgniter 4 (CI4), and Laravel provide different levels of flexibility, ease of use, and functionality.

CodeIgniter 3 (CI3):

CI3 offers a straightforward query builder that provides basic methods for creating, reading, updating, and deleting (CRUD) operations. It's easy to use but lacks some of the more advanced features available in modern frameworks like CI4 and Laravel.

CodeIgniter 3

```
php Copy code

/**
 * To get employee details ...
 * @param $employee_id
 * @return mixed
 * @author MsM Robin
 * @date 2024-08-01
 */
public function getEmployeeDetails($employee_id = null)
{
    $this->db->select('employees.*, departments.name as department_name, designations.name
        ->from('employees')
        ->join('departments', 'employees.department_id = departments.ID')
        ->join('designations', 'employees.designation_id = designations.ID')
        ->join('cities', 'employees.city_id = cities.ID');

    if ($employee_id) {
        return $this->db->where('employees.ID', $employee_id)->order_by('employees.ID', 'D
    } else {
        return $this->db->order_by('employees.ID', 'DESC')->get()->result();
    }
}
```

Common Methods:

- `select()`, `from()`, `where()`, `join()`, `insert()`, `update()`, `delete()`, `get()`, `result()`, `row()`

Limitations:

- Lacks support for advanced features like eager loading, dynamic relationships, and more expressive query methods.

CodeIgniter 4 (CI4):

CI4 enhances the query builder with more expressive methods and better support for modern database features. It's more powerful and flexible compared to CI3.

CodeIgniter 4

```
php Copy code

/**
 * To get employee details ...
 * @param $employee_id
 * @return mixed
 */
public function getEmployeeDetails($employee_id = null)
{
    $builder = $this->db->table('employees')
        ->select('employees.*, departments.name as department_name, designations.name as d
        ->join('departments', 'employees.department_id = departments.ID')
        ->join('designations', 'employees.designation_id = designations.ID')
        ->join('cities', 'employees.city_id = cities.ID');

    if ($employee_id) {
        return $builder->where('employees.ID', $employee_id)->orderBy('employees.ID', 'DES
    } else {
        return $builder->orderBy('employees.ID', 'DESC')->get()->getResult();
    }
}
```

New Features in CI4:

- Query Builders: More expressive, allowing for fluent method chaining.
- Better Integration with Models: CI4 models come with built-in query builder methods, making it easier to interact with the database.
- Enhanced Result Handling: CI4 offers better methods for handling query results, such as `getResult()`, `getRow()`, and `getNumRows()`.

Additional Methods:

- `like()`, `orWhere()`, `groupBy()`, `having()`, `orderBy()`, `limit()`, `offset()`, `countAllResults()`, `getCompiledSelect()`

Laravel:

Laravel's query builder is one of the most powerful and expressive in the PHP ecosystem. It offers a fluent interface and integrates seamlessly with Eloquent ORM, making database interactions both simple and highly customizable.

Advanced Features:

- **Aggregates:** Methods like `count()`, `max()`, `min()`, `avg()`, and `sum()` are available for performing aggregate calculations.

```
$count = DB::table('users')->where('status', 'active')->count();
```

Joins and Subqueries: Laravel supports complex joins and subqueries within the query builder.

```
php Copy code

$users = DB::table('users')
    ->join('posts', 'users.id', '=', 'posts.user_id')
    ->select('users.*', 'posts.title')
    ->get();
```


Eloquent ORM Integration: Laravel's query builder works seamlessly with Eloquent, providing an easy way to interact with the database using models.

Advantages:

- **Expressive Syntax:** The query builder is designed to be readable and easy to understand, even for complex queries.
- **Integration with Eloquent:** The tight integration with Eloquent ORM allows developers to choose between raw SQL queries, fluent query builder methods, or Eloquent models depending on the need.
- **Advanced Features:** Laravel's query builder offers extensive features like pagination, chunking results, and more, making it suitable for complex applications.

Laravel ORM (Eloquent)

php

 Copy code

```
namespace App\Models;

use Illuminate\Database\Eloquent\Model;

class Employee extends Model
{
    protected $table = 'employees';

    public function department()
    {
        return $this->belongsTo(Department::class, 'department_id', 'ID');
    }

    public function designation()
    {
        return $this->belongsTo(Designation::class, 'designation_id', 'ID');
    }


    public function city()
    {
        return $this->belongsTo(City::class, 'city_id', 'ID');
    }
}
```

```
/**
 * To get employee details ...
 * @param $employee_id
 * @return mixed
 */
public static function getEmployeeDetails($employee_id = null)
{
    $query = self::with(['department', 'designation', 'city']);

    if ($employee_id) {
        return $query->where('ID', $employee_id)->orderBy('ID', 'DESC')->first();
    } else {
        return $query->orderBy('ID', 'DESC')->get();
    }
}
```

Laravel Query Builder

php

 Copy code

```
use Illuminate\Support\Facades\DB;

/**
 * To get employee details ...
 * @param $employee_id
 * @return mixed
 */
public function getEmployeeDetails($employee_id = null)
{
    $query = DB::table('employees')
        ->select('employees.*', 'departments.name as department_name', 'designations.name')
        ->join('departments', 'employees.department_id', '=', 'departments.ID')
        ->join('designations', 'employees.designation_id', '=', 'designations.ID')
        ->join('cities', 'employees.city_id', '=', 'cities.ID');

    if ($employee_id) {
        return $query->where('employees.ID', $employee_id)->orderBy('employees.ID', 'DESC')
    } else {
        return $query->orderBy('employees.ID', 'DESC')->get();
    }
}
```

5. Helpers, libraries to services

In web frameworks, helpers, libraries, and services are essential components for organizing reusable code. The evolution from CI3 to CI4 and Laravel reflects a shift toward more structured and modular approaches, providing better flexibility, maintainability, and scalability.

CodeIgniter 3 (CI3):

- **Helpers:** CI3 helpers are simple, procedural functions that provide utility methods. They are not object-oriented and are loaded manually.
- **Libraries:** Libraries in CI3 are more complex than helpers. They are typically classes that encapsulate specific functionality, such as session management, email handling, or form validation. Libraries can be core libraries provided by CodeIgniter or custom ones developed by the user.
- **Loading Helpers and Libraries:** Helpers and libraries are loaded manually when needed, either in controllers or models.

Example: Using Helpers and Libraries in CI3

```
php Copy code

// Loading a helper
$this->load->helper('url');
echo base_url();

// Loading a library
$this->load->library('session');
$this->session->set_userdata('user_id', 1);

// Custom library usage
$this->load->library('my_custom_library');
$this->my_custom_library->customMethod();
```

CodeIgniter 4 (CI4):

- **Helpers:** CI4 still supports procedural helpers but encourages the use of more organized and modular code through services and classes.
- **Libraries to Services:** CI4 introduces the concept of services, replacing libraries in many cases. Services are singletons, meaning they are instantiated only once and can be accessed globally throughout the application. This approach improves performance and provides a more modern structure.
- **Loading Helpers and Services:** Helpers are still loaded manually, but services are accessed through a service container, providing better organization and control.

Example: Using Helpers and Services in CI4

```
php Copy code  
  
// Loading a helper  
helper('url');  
echo base_url();  
  
// Using a service  
$session = \Config\Services::session();  
$session->set('user_id', 1);  
  
// Custom service usage  
$myService = \Config\Services::myCustomService();  
$myService->customMethod();
```

Advantages:

- **Service Container:** CI4's service container allows for better dependency management and more modular code.
- **Singleton Services:** Services are singletons, ensuring they are instantiated only once, reducing memory usage and increasing efficiency.

Laravel:

- **Helpers:** Laravel provides a rich set of global helpers that are available throughout the application without the need for manual loading. These helpers are designed to be convenient utility functions.
- **Service Container:** Laravel's service container is a powerful tool that handles dependency injection and service management. Services in Laravel are typically provided by classes and are bound to the service container, which manages their lifecycle and dependencies.
- **Service Providers:** Service providers in Laravel are the central place to register and configure services. They allow you to bind classes to the service container, ensuring that services are only loaded when needed.
- **Facades:** Laravel provides facades as a way to access services in a convenient, expressive way. Facades provide a static interface to classes that are bound in the service container.

Advantages:

- **Dependency Injection:** Laravel's service container allows for automatic dependency injection, making it easy to manage complex dependencies.
- **Service Providers:** Service providers centralize service registration, making it easy to configure and extend the application.
- **Facades:** Facades provide a convenient and expressive way to interact with services without needing to pass around service objects.

Example: Using Helpers and Services in Laravel

```
php Copy code  
  
// Using a helper  
echo url('/');  
  
// Accessing services via Facades  
use Illuminate\Support\Facades\Session;  
  
Session::put('user_id', 1);  
  
// Custom service via Service Provider  
// Binding a service in a Service Provider  
$this->app->bind('MyCustomService', function ($app) {  
    return new \App\Services\MyCustomService();  
});  
  
// Accessing the custom service  
$myService = app('MyCustomService');  
$myService->customMethod();
```

THE END