Python

unknown

Aug 20, 2023

CONTENTS

CHAPTER

ONE

SERVER REQUIREMENTS

The CodeCube framework has a few system requirements. Installing Apache distribution packages such as Xampp, Wamp, Laragon (for windows) or a standard Lamp stack setup (for Linux) will satisfy most if not all requirements. If you are setting up an Apache server manually, make sure your server meets the following requirements:

- PHP >= 7.2.0
- BCMath PHP Extension
- Ctype PHP Extension
- JSON PHP Extension
- Mbstring PHP Extension
- OpenSSL PHP Extension
- PDO PHP Extension
- Tokenizer PHP Extension
- XML PHP Extension

CHAPTER

TWO

ADDITIONAL REQUIREMENTS

- Relational database such as MySQL
- Git
- Composer
- A standard HTML supported browser

You can follow this tutorial while setting up Lamp server in Debian based Linux distro to ensure your server setup is up to the standard.

CHAPTER

THREE

INSTALLING CODECUBE

First make sure you have Git installed. Go to your server root directory (for example **var/www/html** in Lamp depending on your settings), open it in terminal (or gitbash if you're in windows) and run the following command-

```
composer create-project codecube/codecube
```

Running this command will automatically setup all composer dependencies.

Alternatively, you can run the following command to clone from the GitHub repository directly.

```
git clone https://github.com/bappychanting/codecube.git
```

You will see it has created a folder titled **codecube**. Customize the folder name to what you want to set your project name as.

You can run the following command and change **my-folder-name** to set your desired folder name directly-

```
git clone https://github.com/bappychanting/codecube.git my-folder-name
```

Once done navigate to the folder and update composer via following command-

```
composer update
```

Afterwards, go to the terminal, make sure the path is set to your project folder and run below command to start your project-

```
php -S localhost:8000
```

CHAPTER

FOUR

CONFIGURATION

4.1 Environment Configuration

If you go to project folder, you'll find there is a project configuration file titled **env.example.php**. Make a copy of the file and rename it to **env.php**. You can either do it manually via file manager or run the following command in the previously opened terminal/git bash window-

```
cp env.example.php env.php
```

Open the **env.php** file in a text editor and change the values of the constants to set up your project. The setup constants are explained below-

4.1.1 Set Application Values

- `APP_NAME`: Defines application name e.g. if you are creating a blog you can rename the constant into "My Blog".
- `APP_URL`: Defines the application default URL. By default it is set to local server address *http://localhost:8000*. You should set it to domain/subdomain address once deployed in an online server. You can set the protocol dynamically using this code: `(isset($_SERVER['HTTPS']) && $_SERVER['HTTPS'] == 'on' ? 'https' : 'http').'://localhost:8000'`.
- `APP_ENV`: Defines the current application environment. BY default it is set to development environment- `dev`. You should change the value once you deploy the project in production server.

4.1.2 Set Database Values

- `DB_CONNECTION`: Defines your database driver/what database you are using. Default value is 'mysql'. You can change it if you are using any other database instead such as MariaDB or SQLite.
- `DB_HOST`: Defines your database host address. Default value is '127.0.0.1'.
- `DB_PORT`: Defines the port of your database host address. Default value is '3306'.
- `DB_DATABASE`: Defines the database name. Default value is 'homestead'. Change it to the name of the database schema you are going to use for your project.
- `DB_USERNAME`: Defines the database username. Default value is 'homestead'. Change it to the username to access your database schema, for example 'root'.
- `DB_PASSWORD`: Defines the database password. Change it to the password to access your database schema. If there is no password keep it empty.

4.1.3 Set Mail Values

- `MAIL_DRIVER`: Defines you mail driver. By default it is set as 'smtp'. If you are using any other mail driver such as 'pop3', change it accordingly.
- `MAIL_HOST`: Defines your mail host URL. Default value is 'smtp.mailtrap.io'.
- `MAIL_PORT`: Defines the port of your mail host URL. Default value is '2525'.
- `MAIL_USERNAME`: Defines the username for your mail settings.
- `MAIL_PASSWORD`: Defines the password for your mail settings.
- `MAIL_ENCRYPTION`: Defines the type of encryption you will use for sending mails. Default value is 'ssl'.

4.2 Application Configuration

Before starting up your application, you may want to checkout the application configuration. Open **config/app.php** in a text editor to checkout or change the basic configuration of application. Below the keys of the array returned from the file are explained-

- `auth_time`: Declares how long a login session will last. To update the value, change the default integer and define whether it will be in hours, minutes or seconds.
- `remember_me`: Declares how long the remember me cookie will last. Follow the same strategy as `auth_time` settings to update this value.
- `update_session_cookie_settings`: Declares whether the previously declared auth time will be actually activated and updated in php settings. Default value is 'no', setting it to 'yes' will update the php session settings.
- `upload`: Declares where the uploaded files will be saved. By default the system will upload the files in **storage/app/public** folder.
- `auto_logging`: Declares whether the system will save various errors, warning etc. messages from the system or logs declared by you in various log files. By default it is `on`, changing the value to anything else will turn off auto-logging. The system saves the log files in **storage/logs** folder.
- `locale`: Declares the system locale. By default it is `en` (English).
- `placeholder`: Declares the default placeholder for all images in the website. By default the system uses images from Lorem Picsum as placeholder.

Additionally you can run the migration files if you want to checkout the demo application ported with the framework. Visit *Database Migration* for more information.

CHAPTER

FIVE

DIRECTORY STRUCTURE

CodeCube follows a similar directory structure to Laravel framework. The directories are structured in a way to provide a standard starting point to use the framework, but with a thorough understanding of the directories, the developers are free to customize the directories anyway he likes without breaking the system:

Listing 1: CodeCube Framework Directory Structure

```
/app
    /Helpers
    /Http
         /Controllers
         /Guards
    /Models
/config
/database
/resources
      /assets
      /locale
      /markups
      /views
/routes
/storage
      /app
         /public
      /logs
/vendor
```

Below the directories are briefly explained-

- The **app** directory: This is the directory to store all your model, controller, guard and rest of the classes. This directory has three sub directories-
  - **Helpers:** This is where you store all your helper classes.
  - **Http:** This folder has two sub-classes-
  - **Controllers:** All the controller classes are stored here.
  - **Guards:** All the classes that contain various methods to maintain authentication and access control logic are stored here.
  - **Models:** All your model classes will be included here.
- The **config** directory: This is where you will store various project configuration files. All of these configuration files will return hard-coded data declared by the developer to be used wherever necessary within the system, as array.

- The **database** directory: In this directory all the files that will be used to run migration are stored. Just like configs, the files will return arrays, but the arrays have various database commands as values instead.
- The **resource** directory: In the directory all the files necessary to generate your views are stored. The directory contains three sub-directories:
- **Assets:** Contains all the project assets such as images, JavaScript and CSS files.
- **Markups:** Contains various XML files to provide necessary design classes, layouts etc. to the system to generate general views such as mail and pagination.
- **Views:** Contains all your view files.
- The **route** directory: This directory contains all the route definitions for your system. Like config and database files, these files will return arrays with keys for route name and values for URL.
- The **storage** directory: This is where all the uploaded files (by default) and log files are stored.
- The **vendor** directory: This folder will be generated once the composer update command is run. This folder contains all the composer dependencies.

CHAPTER

SIX

ARCHITECTURE CONCEPT

The entry and exit point of the whole framework is **index.php** file. This file includes the application configuration files to configure the settings, autoload files to include all the base classes and functions, and route files to call method of a controller class defined for the current URL. All of these are done within a **try catch** function, which reports error if any exception is found while performing an action.

The framework follows **MVC** (Model, View, Controller) pattern. How the general flow of activities follows this design pattern within this framework is explained below-

- Models are where all the classes based on the real life entities associated with the program, their relations, objects and methods are defined, and **OOP** (Object Oriented Programming) concept is applied on top of them. This is where different **CRUD** (create, retrieve, update, delete) actions are generally performed and passed to the Controller.
- Controllers retrieve data returned from model, perform necessary actions on them and pass it to the view. Generally, access control logic is applied here as well.
- Views are used to construct visual layout and present the data to the user.

Additionally, apart from working on models, views and controllers:

- Various base classes and functions from **base** folder (located in **vendor/codecube** directory) can be called to follow the standard way of accessing database, input validation, defining layouts, connecting models , controllers and views etc.
- Methods from helper classes, or classes included by composer dependency can be called to perform certain actions on the data anywhere within the system.
- Controller will call method from guard classes to implement access control logic.

CHAPTER

SEVEN

ROUTING

Routes are defined in **routes/web.php** file. As mentioned earlier, routes are basically an array with key names for the URL, and the name of the **Controller** class and method to be called for the URL as value.

7.1 Default Routes

Before defining routes, you may need to declare the default route for landing and error pages.

To do that, open **config/url.php** file and modify the value of `landing` key for the default landing page, and `error` key for the default error page of the project.

As value, you must type the name of the controller class first, then method of the class to be called after @ sign-

```
HomeController@welcome
```

Above, 'HomeController' is the name of the controller class and 'welcome' is the name of the method.

You can define the URL for accessing database migration page as the value of `migration_url` key. The URL must exclude the domain name and first forward slash. Once you go into your browser and access the URL (by adding a forward slash and the value of the aforementioned key after your domain name) you will see the database migration page.

You can also define the **prefix** for the URLs that will receive API calls in the `api_url` key. The framework will automatically include the prefix at the beginning of the API routes that you have defined.

7.2 Defining Web Routes

To define web routes, open **routes/web.php** file. There, you can define the URL for the route as a key of the array returned, and name of the controller class along with the method called from the class as its value-

```
'welcome' => 'HomeController@welcome'
```

In the above example, for the *your-site-url/welcome* URL, `welcome()` method from the **HomeController** class is executed. Like default routes, you have to type the method name after the controller class name and @ sign.

To include SEO friendly URL parameters to your web routes, include curly brackets to each of the route keywords that you want to use as a URL parameter like below-

```
'blog/show/{blog_id}' => 'BlogController@show'
```

Note that, here, the generated **$_GET** parameter for `{blog_id}` will simply exclude the curly brackets, i.e., it will be `blog_id`. Furthermore, to identify such a route, the route must include at least one route keyword that is not a URL parameter. Finally, if there are a list of routes that only differ by the name of URL parameter(s) inside the curly braces, always the first route from that list will be called to execute the assigned controller method.

7.3 Defining API Routes

To define API routes, open **routes/api.php** file. There, you can define the API routes similarly to how the web routes are defined above-

```
'test_api' => 'HomeController@testApi',
```

It works almost exactly the way as the web routes, except it doesn't support SEO friendly URL parameters! Notice that you must exclude the API prefix defined in **config/url.php** while defining your API routes. Call *your-site-url/api-prefix/test_api* url from your browser/postman to receive your API data.

7.4 Using Routes

CodeCube provides some default functions for working with routes and URLs.

To show a route link in a view, use the `route()` method. The route must be defined in the **routes/web.php** file as a route key, otherwise the framework will report error.

```
echo route('signup');
```

You can add additional parameters to the route as an associative array like below-

```
echo route('items/show', ['id' => $item['id']]);
```

If your web route includes SEO friendly URL parameters, you can set the value for those parameters defined in the route using the passed associative array like below-

```
echo route('blog/show/{blog_id}', ['blog_id' => 5]);
```

Here, `{blog_id}` will be replaced by the value of `blog_id` from the associative array that has been passed as the function's second parameter. Note that, to replace the route keyword with a value from the array, the key name MUST BE the same as the curly bracket excluded keyword from the route. The generated URL will be *your-site-url/blog/show/5*.

To generate API links, use the `api_route()` function. It works almost the same way as the `route()` function, except it doesn't support SEO friendly URL parameters.

```
echo api_route('sitemap', ['id' => 5]);
```

It will generate the link *your-site-url/api/sitemap?id=5*. Notice that, similarly to web routes, you can add URL parameters by passing the list of parameters and their names as an associated array as the second argument of the function.

Sometimes during routing you might need to include **$_GET** parameters of the current URL along with additional **$_GET** parameters you want to pass to a URL. You can do this using `routeUrl()` method. The third parameter takes the list of **$_GET** parameters of the current URL you want to exclude, as an array-

```
echo routeUrl('items/show', ['id' => $item['id']], ['item_token]);
```

If your web route includes SEO friendly URL parameters, in above function, you can set the value for those parameters defined in the route in a similar fashion as the `route()` function-

```
echo routeUrl('blog/show/{blog_id}', ['blog_id' => 5, 'keyword' => 'author'], ['blog_id
↪']);
```

It URL it will generate is *your-site-url/blog/show/5?keyword=author*.

To check whether the current URL is a specific route, use the `route_is()` method.

```
echo route_is('home') ? 'active' : '';
```

In the above code, the framework checks whether the current route is **home** and if it is, shows **active**.

To exclude specific keywords from route during checking, add curly braces to the route keywords like below-

```
echo route_is('blog/show/{blog_id}/author') ? 'active' : '';
```

In the above example, the function will automatically exclude the `{blog_id}` keyword and check whether the position of rest of the parameters match with current URL keywords, and return **TRUE** if they do.

To get the route from the current URL, use `get_route()` method. For URL *your-site-url/home*, this method will extract the route **home**-

```
echo get_route();
```

You can replace certain keywords of the URL by passing the list of keywords and their position as parameter to get a more precise route-

```
echo get_route(['2' => '{blog_id}']);
```

Above for URL *your-site-url/blog/show/45*, the function will return "blog/show/{blog_id}".

To get the URL of the current page use the following function-

```
echo get_url();
```

To redirect from controller to the last visited URL of the site from where some form data was submitted, use `back()` function-

```
echo back();
```

CHAPTER

EIGHT

GUARDS

Guard classes allow to perform actions such as request checking, filtering, redirection etc. conveniently before executing a controller method. It encapsulates these filtering logic, and as such, helps to perform actions such as authentication or access control while called, without repeating codes in different controller classes.

```
<?php
namespace App\Http\Guards;
use Base\Request;
use Base\BaseController as base;
class CheckGuest
{
    public function __construct()
    {
        $request = new Request();
        if($request->auth()){
            base::redirect('home');
        }
    }
}
```

Guard classes are defined in **app/Http/Guards** folder. Below is an example of a guard class that checks whether a user is authenticated, and if so, redirects him to home page, thus preventing him from accessing the function.

This guard can be called within a controller method like below-

```
$this->guard('CheckGuest');
```

CHAPTER

NINE

CSRF PROTECTION

CodeCube makes it easy to protect your application from cross-site request forgery(CSRF) attacks. Cross-site request forgeries are a type of malicious exploit whereby unauthorized commands are performed on behalf of an authenticated user.

CodeCube automatically generates a **CSRF token** for each active user session managed by the application. This token is used to verify that the authenticated user is the one actually making the requests to the application.

Anytime you define an HTML form in your application, you should include a hidden `_token` field in the form so that the framework can validate the request-

```
<form method="POST" action="<?php echo route('items/store'); ?>">
<input type="hidden" name="_token" value="<?php echo csrf_token(); ?>">
</form>
```

9.1 X-CSRF-TOKEN

In addition to checking for the **CSRF** token as a POST parameter, the framework will also check for the **X-CSRF-TOKEN** request header. You could, for example, store the token in an HTML meta tag:

```
<meta name="csrf-token" content="<?php echo csrf_token(); ?>">
```

Then, once you have created the meta tag, you can instruct a library like jQuery to automatically add the token to all request headers. This provides simple, convenient **CSRF** protection for your AJAX based applications-

```
$.ajaxSetup({
    headers: {
        'X-CSRF-TOKEN': $('meta[name="csrf-token"]').attr('content')
    }
});
```

CHAPTER

TEN

CONTROLLERS

Controller classes are stored in **app/Https/Controllers** folder. All new controller classes declared by the developer must extend the default **Controller** class in the folder which itself inherits the **BaseController** class to access various **base** controller methods. Default **Controller** class can also be used to write custom methods that will be used throughout different controller classes.

Below is an example of a basic controller class-

```
<?php

namespace App\Http\Controllers;

class HomeController extends Controller
{
    public function home()
    {
        $this->guard('CheckAuth');
        return $this->view('home');
    }
}
```

Here are the **BaseController** class methods that can be used in controller classes:

- The `config()` Method: This method can be used to call configuration files from the **config** directory. If the config file is nested inside directories, the directory tree of the config file is represented via `.` dots. Below the **config/dev/app.php** file is called by this method and saved in `$app` variable.

```
$app = $this->config('dev.app');
```

- The `view()` Method: This method is used for calling view files at the end of controller method and pass data to them. The directory tree of the view file is represented via `.` dots.

```
    public function home()
{
    $auth = $request->auth();
    return $this->view('front.home', compact('auth'));
}
```

In the above example, **views/front/home.php** view file is called in the `HomeController::home()` method and the authentication data is passed in the view.

- The `redirect()` Method: Used for redirecting to different route urls.

```
    public function redirect()
{
        $this->redirect('items/show', ['id' => 1]);
}
```

In the above example, the `HomeController::redirect()` method redirects to **items/show** route with id parameter.

- The `abort()` Method: This method is used to redirect to error pages.

```
    public function error()
{
    $this->abort(404);
}
```

In the above example, the `HomeController::error()` method shows the **404** error page to the user.

CHAPTER

ELEVEN

SESSION REQUESTS

Base **Request** class is used to access various session data. All data that are controlled using **Request** class are stored in the `request` key of session array. To use the **Request** class, call it on top of your class like below-

```
use Base\Request;
```

Once called, you can access various methods of the class to control session data. Below how the session data can be accessed and modified using **Request** class is explained-

11.1 Reading and Writing Session data

You can input new data into session using the `Request::setData()` method like below-

```
Request::setData('color-green', 'green);
```

Here, `color` is the name (key) of session and `green` is the value. To read the input value you can use `Request::getData()` method like below-

```
echo Request::getData('color-green');
```

Additionally you can use `Request::put()` method to store associative arrays in session like below-

```
Request::put('winners', ['first' => 'John', 'second'=> 'doe']);
```

You can get the array using `Request::show()` method. The data will be returned as a PHP stdClass Object-

```
<?php
        $winners = Request::show(winners);
        echo winners->first;
?>
```

11.2 Destroying Session Data

To destroy a session data, use `Request::destroy()` method like below-

```
Request::destroy('winners');
```

11.3 Flashing Session Data

Sometimes you might need to flash session data to display alerts, warnings etc. Flashing session data means destroying session data automatically right after it has been used. To set session flash data, use `Request::setFlash()` like below-

```
Request::setFlash(['success' => 'Item added!']);
```

To flash the session data, use `Request::getFlash()` like below-

```
print_r(Request::getFlash());
```

11.4 Handling Cookies

Base Request class provides you with some built in methods to conveniently handle cookies.

To add a new cookie, use `Request::setCookie()` like below-

```
Request::setCookie('hello', 'Hello World', 300);
```

Here, 'hello' is the name of the cookie, 'Hello World' is the value and 300 is how long the cookie will last in seconds.

To get the cookie value, use `Request::getCookie()` like below-

```
echo Request::getCookie('hello');
```

To delete a cookie, use `Request::deleteCookie()` like below

```
Request::deleteCookie('hello');
```

11.5 Get HTTP Headers

You can access all the HTTP headers using `Request::headers()` method

```
print_r(Request::headers());
```

Above method will show all the information related to HTTP headers.

CHAPTER

TWELVE

FRONTEND\VIEWS

To help generating views, CodeCube uses PHP Template Inheritance plugin, as well as various custom methods.

As explained in *Directory Structure*, all the files necessary to generate views are stored in **resources** directory. There, you can store css, js, images and various asset files in **resources/assets** directory, layouts and class-names for various PHP tags for the views that will be automatically generated by the system in **resources/xml** directory, and necessary PHP view files in **resources/views** directory.

12.1 Defining Layouts

To define a layout, you can create a **layout.php** file, and use `startblock()` & `endblock()` methods to define each section for the layout like below-

```
<html>
<head>
<title><?php startblock('title') ?><?php endblock() ?></title>
</head>
<body>
<?php startblock('content') ?> <?php endblock() ?>
</body>
</html>
```

12.2 Extending a Layout

Once defined you can extend the layout file in **home.php** file like below-

```
<?php inherits('layout'); ?>

<?php startblock('title') ?>Home<?php endblock() ?>

<?php startblock('content') ?>This is my content!<?php endblock() ?>
```

Above, the `inherits()` method is used to call the parent view the child view is extending form.

12.3 Including other views & assets

To include other views, use the `append()` method like below-

```
append('front.leftbar', ['items' => $items]);
```

Above, **views/front/leftbar.php** file is included in the **home.php** file and items array is passed to it.

To include css files, use the `style()` method like below-

```
echo style('style.css');
```

To include js files, use the `script()` method like below-

```
echo script('script.js');
```

To show the title of the application, use the `title()` method. To show any other text other then title, pass the text as parameter-

```
echo title('My Title');
```

It will show the generated title using the text along with project name given in the `APP_NAME` constant, like this-

```
My Title || Project Name
```

To get the source of a file uploaded in the directory defined in the `upload` key in **config/app.php** file in a view, use the `asset()` method like below-

```
echo asset('document.pdf');
```

To show image in view, use the `image()` method like below-

```
echo image('my_image.jpg');
```

To show an image's thumbnail in view that was saved during upload, pass the thumbnail extension text as an argument to the `image()` method like below-

```
echo image('my_image.jpg', 'alt', [], '_thumb');
```

The framework will automatically detect whether the image exists, if not it will use the value of `placeholder` key set in **config/app.php** instead.

To show the default icon of the website, use the `icon()` method like below-

```
echo icon('favicon.ico');
```

CHAPTER

THIRTEEN

LOCALIZATION

13.1 Introduction

CodeCube's localization features provide a convenient way to retrieve strings in various languages, allowing you to easily support multiple languages within your application. Language strings are stored in files within the **resources/locale** directory. Within this directory there should be a subdirectory for each language supported by the application:

Listing 1: Locale Structure

```
/resources
    /locale
        /en
            message.php
        /es
            message.php
```

All language files return an array of keyed strings. For example:

```
<?php
    return [
        'info' => 'This is a info alert!',
    ];
?>
```

13.2 Configuring The Locale

The default language for your application is stored in the **config/app.php** configuration. You can change your preferred language by changing value of the `locale` key, which is set as `en` (English) by default. Here **en** is the name of the folder containing your language files.

```
'locale' => 'en', // configuration for default locale
```

13.3 Retrieving Translation Strings

You can retrieve texts from the language files using the `locale()` method. Pass the name of the file as the first parameter and the name of the key as the second parameter.

```
echo locale('message', 'info');
```

Above code will display 'This is a info alert!' text from the **message** language file.

If you wish, you may define placeholders inside any of the array of strings inside a locale file. All placeholders are prefixed with a `:`. For example, you may define a validation message with a placeholder `:data`:

```
'empty' => ':data can not be empty!',
```

To replace the placeholders when retrieving the string, pass an array of replacements as the second argument to the `locale()` method.

```
echo locale('validation', 'empty', ['data' => 'Name']);
```

Above code will display "Name cannot be empty", replacing the `:data` placeholder.

CHAPTER

FOURTEEN

FORM VALIDATION

You can define form validation roles in either model or controller. To set the validation roles, you can declare an array, and then define each validation error text as a key of the array like below-

```
$errors = array();

if($_POST[‘name’]){
 $errors['name'] = "Name can not be empty!";
}

if($_POST[price]){
 $errors['price'] = "Price can not be empty!";
}
```

Once you’ve defined each validation roles, you have to pass the array to `setErrors()` method as parameter to get the errors in the view page.

To show the error in the view page, use the `field_err()` method like below-

```
<span><?php echo field_err('name'); ?></span>
```

To get all the errors found in the view page they were submitted from, use the `getErrors()` method.

```
<p><?php print_r(getErrors()); ?></p>
```

14.1 Get Previously Submitted Values

To get the submitted post values back in the view page they were submitted from, use the `field_val()` method.

```
<input type="text" name="name" value="<?php echo field_val('name'); ?>"
```

CHAPTER

FIFTEEN

DATABASE

CodeCube provides some built in mechanisms for easier database interaction.

15.1 Configuring Database

As discussed in *Configuration* chapter, you can configure the database connection by changing the `DB_` constants in **env.php** file like below-

```
const DB_CONNECTION = 'mysql';
const DB_HOST = '127.0.0.1';
const DB_PORT = '3306';
const DB_DATABASE = 'codecube';
const DB_USERNAME = 'root';
const DB_PASSWORD = '1234'
```

15.2 Query Builders

You can use CodeCube's built in functionalities to perform **CRUD** (Create, Retrieve, Update, Delete) functions on database. To access those functionalities, you must use the base **DB** class at the top of your PHP file.

```
use Base\DB;
```

15.3 Retrieving Results

To read a table from database, use the `table()` method of the **DB** class to begin query, and `read()` method to get the result.

```
$db = new DB;
$items = $db->table('items')->read();
```

15.3.1 Aggregation

To get the sum of the last set of values returned by `read()` method, use the `getTotal()` method-

```
$total_items = $db->getTotal();
```

15.3.2 Add Conditions

To apply the SQL **where** clause on records, combine the `where()` method with other clauses-

```
$items = $db->table('items')->where('user_id', '=',  1)->read();
```

You can use the `or()` method, `and()` method or `not()` method along with `where()` method like below-

```
$items = $db->table('items')->where('user_id', '=',  1)->and('price', '<', '250')->
↪or('price', '>', '50')->where()->not('name', '=', 'buy')->and()->not('name', '=',
↪'buying')->read();
```

To add extra conditions use the `condition()` method-

```
$db->table('items')->where('user_id', '=', 1)->condition('AND price > 50')->read();
```

15.3.3 Ordering

Use the `orderBy()` method at the end of your conditions to order your results-

```
$items = $db->table('items')->where('user_id', '=', 1)->orderBy('created_at', 'desc')->
↪read();
```

15.3.4 Pagination

To get paginated data from the database, use the `limit()` method before `read()` method-

```
$items = $db->table('items')->limit(2)->read();
```

Afterwards, you can get a pagination of your query using the `pagination()` method-

```
$pagination = $db->pagination();
```

Additionally, you can use the `paginationData()` method to get necessary pagination data as array-

```
$data = $db->paginationData()
```

Afterwards, pass the array values as parameters to `generatePages()` method to generate the pagination-

```
$pagination = $db->generatePages($data['page'], $data['totalPages'], $data['url'])
```

15.4 Inserting Data

To insert data in the database, use the `create()` method in combination with `table()` method, and `data()` method to pass input data array, where keys are the name of the table columns and their values are the input values-

```
$db->table('items')->data(['name' => 'Human', 'price' => '500'])->create();
```

15.5 Updating Data

To update data of a row in the database, use the `update()` method in combination with `table()` method, and `set()` method to pass update data array, where keys are the name of the table columns and their values are the updated values-

```
$db->table('items')->set(['name' => 'update name'])->where('id', '=', 1)->update();
```

15.6 Deleting Data

Before deleting, CodeCube checks whether `deleted_at` column exists in a table. If it does, instead of deleting the column completely, the framework updates the column value to current date-time, and ignores the rows with NOT NULL values for `deleted_at` column in `read()` method. If the column doesn't exist, the framework deletes the row completely. To delete a row from a table, use the `delete()` method.

```
$db->table('items')->where('id', '=', 1)->delete();
```

15.7 Raw SQL Query

You can use the `get()` method to directly input a SQL read command-

```
$db->get('SELECT name, username FROM users');
```

You can use the `write()` method to write raw SQL command to insert, update or delete data into the table.

```
$db->write('INSERT INTO items ('name', 'price') VALUES ('New Item', '100')');
```

15.8 Using SQL Views

The framework puts extra emphasis on database views for more effective retrieval of database tables. You can use the SQL views in combination with CodeCube built-in query builder functions to perform more complex queries.

As an example, you can join **users** and **items** table and save the query as a SQL view-

```
CREATE VIEW `items_view` AS SELECT i.id AS id, i.name as name, i.price as price, i.user_
↪id as user_id, u.username AS username, i.created_at AS created_at, i.updated_at AS␣
↪updated_at, i.deleted_at AS deleted_at FROM items i, users u WHERE i.user_id = u.id
```

And later, use query builder to read data from the view-

```
$items = $db->table('items_view')->where('user_id', '=', 1)->orderBy('created_at', 'desc
↪')->limit(2)->read();
```

15.9 Return SQL Command

To find the last executed SQL command string, use the `getLastSQL()` method-

```
logger($db->getLastSQL());
```

15.10 Migration

Migrations are like version control for your database, allowing your team to easily modify and share the application's database schema. CodeCube provides a convenient way of migrating your database.

Before migration, you have to create/modify your migration files. To start working on migration files, go to **database** folder. There you will find 4 files for creating and removing tables and views-

```
1_drop_view_statements

2_drop_table_statements

3_create_table_statements

4_create_table_statements
```

There migration files return SQL commands as an associative array and each command with an identifying key. Write your SQL create and drop table/view commands there for all the tables and views you want to create and drop. Make sure to maintain proper order while writing your SQL commands so that the system won't face conflicts related to foreign key checks while performing migration.

```
'create_users' => "CREATE TABLE `users` (`id` int(11) unsigned NOT NULL AUTO_INCREMENT,␣
↪`name` varchar(255) COLLATE utf8_unicode_ci NOT NULL, `username` varchar(255) COLLATE␣
↪utf8_unicode_ci NOT NULL, `email` varchar(255) COLLATE utf8_unicode_ci NOT NULL,␣
↪`password` varchar(255) COLLATE utf8_unicode_ci NOT NULL, `created_at` timestamp NULL␣
↪DEFAULT NULL, `updated_at` timestamp NULL DEFAULT NULL, PRIMARY KEY (`id`), UNIQUE KEY␣
↪`users_username_unique` (`username`), UNIQUE KEY `users_email_unique` (`email`))"
```

To insert values to your tables, create new insert files, with an identifying number at the beginning of migration file name to maintain order.

```
5_insert_users
```

Like the create/drop migration commands, place your insert commands inside a returning array from the migration files with identifying keys for each.

```
'insert_users' => 'INSERT INTO `users` VALUES (1,"Default User","codecube",
↪"codecube@gmail.com","secret", NULL, NULL);',
```

To migrate your database, open terminal in your project root directory and execute the below command.

```
php index.php migrate reset
```

To note, add the second argument `reset` only if you want to reset your previously migrated table.

CHAPTER

SIXTEEN

AUTHENTICATION

CodeCube provides built-in Authentication class for easier implementation of authentication. To access its functionalities, call the class at the top of your PHP file like below-

```
use Base\Authenticable;
```

16.1 Sign In/Sign out

Once called, you can use its `signin()` function to authenticate a user to the system. The method will check whether any user's email or username matches with the passed `identity`, and if found, signs the user in to the system.

```
$auth = new Authenticable;
$auth->signin('identity', 'password');
```

You can pass in the name of your *remember me* checkbox as the third parameter to the `signin()` method if you want to enable persistent login cookies for your authentication.

```
$auth->signin('identity', 'password', 'remember');
```

Once authenticated, you can use the `getAuth()` method to get authenticated user data.

```
$auth_user = $auth->getAuth();
```

You can use the `check()` method to check whether a user is authenticated.

```
echo $auth->check() ? 'authenticated' : 'not authenticated';
```

You can sign out the user using the `signout()` method.

```
auth->signout();
```

16.2 Password Reset

While resetting password, use the `storelink()` method with a unique string and user Id as its parameters to store reset links. The unique string will be stored as password reset link parameter to make the link only accessible by the requesting user-

```
$auth->storeLink(md5(uniqid()), 1);
```

Pass the token as a parameter to the `getLink()` method to check whether the token exists and get other information for the requesting user once the user clicks on the unique password reset link.

```
$auth->getLink($_GET['token']);
```

Once the user resets his password, you can set the unique token invalid using the `updateValidity()` method so that it won't be accessible anymore.

```
$auth->updateValidity($token);
```

CHAPTER

SEVENTEEN

MAIL

CodeCube provides a base **Mail** (built on top of PHPMailer class) class to make sending emails easier. Before sending mail, you must provide your mail configuration values to the `MAIL_` Constants in the **env.php** file to configure mail.

```
const MAIL_DRIVER='smtp';
const MAIL_HOST='smtp.mailtrap.io';
const MAIL_PORT='2525';
const MAIL_USERNAME= 'codecube';
const MAIL_PASSWORD= 'pass';
const MAIL_ENCRYPTION= 'ssl';
```

You can also provide values for the XML tags in the **resources/markups/mail.xml** file to send a stylized email with header and footer.

```
<mail>
<style>body {background-color: lightblue;}</style>
<header>Email Header</header>
<footer>Email Footer</footer>
</mail>
```

Once everything is set up, use the setter methods to set up mail receivers, sender, carbon copies, blind carbon copies, subject and message like below. You need to pass an array consisting of appropriate emails as parameter for setting up receivers, carbon copies or blind carbon copies. Use the `createMessage()` method to include the XML layout. If you want to include any XML file other than **resources/markups/mail.xml** containing tags for mail (the file must be in the **resources/markups** directory), you can pass the name of the file as the parameter of this method. Finally use `send()` method to send the mail.

```
use Base\Mail;
$mail = new Mail;
$mail->setReceivers($receivers);
$mail->setSender($sender);
$mail->setCarbonCopies($cc);
$mail->setBlindCarbonCopies($bcc);
$mail->setSubject($subject);
$mail->setMessage($message);
$mail->createMessage()->send();
```

CHAPTER

EIGHTEEN

HELPERS

Helpers are classes with functions that help you to avoid repetition of codes and perform certain tasks easier within the application. CodeCube supplies some built-in helper functions and developers can easily add helpers of their own. All helper classes are stored in **app/Helpers** directory.

By default, CodeCube provides tree helper classes- **ApiHelper**, **Misc** & **Upload**. To access their functionalities, include the classes at the top of your PHP file.

```
use App\Helpers\ApiHelper;
use App\Helpers\Misc;
use App\Helpers\Upload;
```

Their functions are explained below

- `ApiHelper::request()` Method: Returns all the parameters received from an API call (headers, params and body) as an array.

```
print_r(ApiHelper::request());
```

- `ApiHelper::validator()` Method: Returns the input validation messages in intended API format.

```
ApiHelper::validator(['email' => 'email required'], 'Validation failed!');
```

- `ApiHelper::success` Method: Returns the requested data along with a success message in intended API format.

```
ApiHelper::success(['users' => $all_users], 'Success!');
```

- `ApiHelper::fail()` Method: Returns a failure message in intended API format.

```
ApiHelper::fail('Exception found!', ['auth' => 'api authentication failed'], 5001, 401);
```

- `ApiHelper::response()` Method: Here you can define the format for showing your API results. You can also call this method directly if you need to show data that are not compatible with aforementioned helper methods.

```
ApiHelper::response(['message' => 'The requested user has not been verified!', 'error' =>
↪ ['email' => 'email address not found'], false, 400);
```

- `Misc::pluck()` Method: Returns an array consisting of values of a key from a multidimensional array.

```
Misc::pluck($users, 'name');
```

- `Misc::randInt()` Method: Creates a random number of specific length

```
Misc::randInt(10)
```

- `Misc::randStr()` Method: Creates a random string of given length.

```
Misc::randStr(10);
```

- `Misc::urlString()` Method: Creates an SEO friendly URL of a given text.

```
Misc::urlString('New Item', '-');
```

This will return string `new-item`.

- `Misc::substrwords()` Method: Will cut a string to the given length.

```
Misc::substrwords('Keeps the first five letters', 5);
```

- `Misc::generateColor()` Method: Generates random html colors.
- `Misc::generateYearArray()` Method: Returns start and end UNIX timestamp of all the months from this year to the number of previous years passed as parameter.

```
Misc::generateYearArray(1);
```

- `Misc::createCalendarDateRange()` Method: Returns an array containing all the days of the week and all the dates of a given date range. This method helps to create a PHP calendar.

```
Misc::createCalendarDateRange('2019-01-01', '2019-12-31');
```

- `Upload::fileUpload()` Method: Will upload files to the server and return the upload directory.

```
Upload::fileUpload($_FILES['file'], $app['upload'].'/myfiles');
```

- `Upload::imageUpload()` Method: Resizes and uploads image while checking file format. If you want to keep the original image aspect ratio while resizing, set the final parameter as `true`. It returns the upload directory.

```
Upload::imageUpload($_FILES['file'], $app['upload'].'/myfiles', 640, 480, true);
```

- `Upload::imageUploadWithThumb()` Method: Resizes and uploads full size image while checking file format and adds a thumbnail of the image of given size afterwards. Set the thumbnail extension word, thumbnail width and height by passing arguments. If you want to keep the original image aspect ratio while resizing, set the final parameter as `true`. It returns the upload directory.

```
Upload::imageUploadWithThumb($_FILES['file'], $app['upload'].'/myfiles', '_thumb', 640,␣
↪480, true);
```

- `Upload::resizeImage()` Method: Resizes an uploaded image.

```
Upload::fileUpload($app['upload'].'/myimages/uploaded.jpeg', 640, 480, true);
```

Aside from the helper functions provided with helper classes, **CodeCube** also provides some additional built-in functions to help with development and debugging. For example:

- `dd()` Method: Dumps the variable passed as the only parameter and ends execution of the script.
- `logger()` Method: To create logs. Simply pass the variable whose content you want to written in your log.

CHAPTER

NINETEEN

SITEMAP

A **sitemap.xml** is a file where you provide information about the pages, videos, and other files on your site, and the relationships between them. CodeCube framework provides a default sitemap. You can manually update the sitemap if you want, but the CodeCube framework provides built-in functionalities to manage the sitemap.

To manage the sitemap, include the base Sitemap class in your .php file like below-

```
use Base\Sitemap;
```

19.1 Setting up Sitemap

To setup Sitemap for the first time or to add multiple new URLs to sitemap, use the `updateNodes()` method like below-

```
$sitemap = new Sitemap;
$urls = array();
$blogs = $this->blog->getBlogs();
foreach ($blogs as $blog) {
array_push($urls, [ route('blog/show', ['id' => $blog['id']]), date("Y-m-d", strtotime(
↪$blog['updated_at'])) ]);
}
$sitemap->updateNodes(['show'], $urls);
```

19.2 Adding an URL to Sitemap

To add a new URL to the sitemap, use the `addNode()` method with `route()` method like below-

```
$sitemap = new Sitemap;
$sitemap->addNode(route('blog/show', ['id' => 1]), date('Y-m-d'));
```

19.3 Deleting all URLs in Sitemap

To delete all URLs in the **sitemap.xml**, use the `deleteNodes()` method like below-

```
$sitemap = new Sitemap;
$sitemap->deleteNodes();
```

19.4 Refreshing Sitemap

To refresh the sitemap, use the `refreshSitemap()` method like below-

```
$sitemap->refreshSitemap(['show'], $urls);
```

CHAPTER

TWENTY

BUILT-IN APPLICATION

To help you give a clear understanding of the standard way of managing the Models, Controllers, Guards, Views, migrations, markups, assets etc. within the framework, CodeCube provides a demo mini-application. To check the application, *run migration* at first. Afterwards, go to welcome page and click on Demo App link to check out the application. You will find all their model, controller and view files in appropriate folders in the application. Check those codes out and afterwards, you can start developing your new application from there.

CHAPTER

TWENTYONE

CODECUBE FRAMEWORK

An easy & lightweight PHP framework inspired by Laravel.

Aims of this framework are to-

- Provide a foundation to creative PHP developers for faster dynamic website development. Small, hand-built libraries that the framework is based on guarantee minimum performance overhead.
- Assist PHP newcomers who are looking forward to begin coding in Laravel. Due to familiar, pure PHP coding in a similar structure to Laravel framework, it will be easier for a PHP beginner to adopt this framework, evaluate how the basics of a web framework work and switch to Laravel later on.