Raven

GUIDE TO RAVEN FRAMEWORK

RAVEN WEB FRAMEWORK

About

What is Web Framework?
A web framework is a software library or set of libraries that provide developers with a way to build and structure their web applications.
Web framework has tools for **routing**, **security** and **database access**.
Some Popular Web Frameworks include,

- **Laravel**
- **Django**
- **Flask**
- **Ruby on Rails**, etc

Web Frameworks can help developers build complex Web Applications more quickly and easily without having to write all the code from scratch.

What is MVC Architecture?
MVC stands for **Model-View-Controller** which is a software architecture pattern used to separate the different parts of a web application.

- The **Model** is data layer
- The **View** is presentation layer
- The **Controller** is the logic layer that handles requests and responses.
- **MVC** is popular pattern because it promotes separation of concerns, making code more maintainable and easier to test.

What is Raven

Raven is Simple and flexible PHP framework that is suitable for small, medium and complex sized projects, raven is a open-source (OSS) light weight framework under license of MIT.
Easy to learn an use, because its simple and intuitive syntax that is similar to other popular web frameworks like **Laravel**.
It comes with pre-built components and libraries that make it easy to add common feature to your applications.
Raven is highly customizable, extensible, Fast and Secure.
Finally it has a powerful **command line interface** (**CLI**) that make it easy to manage your projects.

TOPICS

- Route Methods
  - GET
  - POST
  - PUT
  - DELETE
  - MATCH
- Route Path
- Route Callbacks
  - using Controllers
  - using Middleware
- Create View
- Create Migration
  - Defining Columns
- Create Model
  - Defining Properties and Attributes
- Create Middleware

Getting Started

Requirements

Composer PHP Dependency Manager
PHP 7.2.5^

Installation

Installation process is simple, use Composer to install Raven.

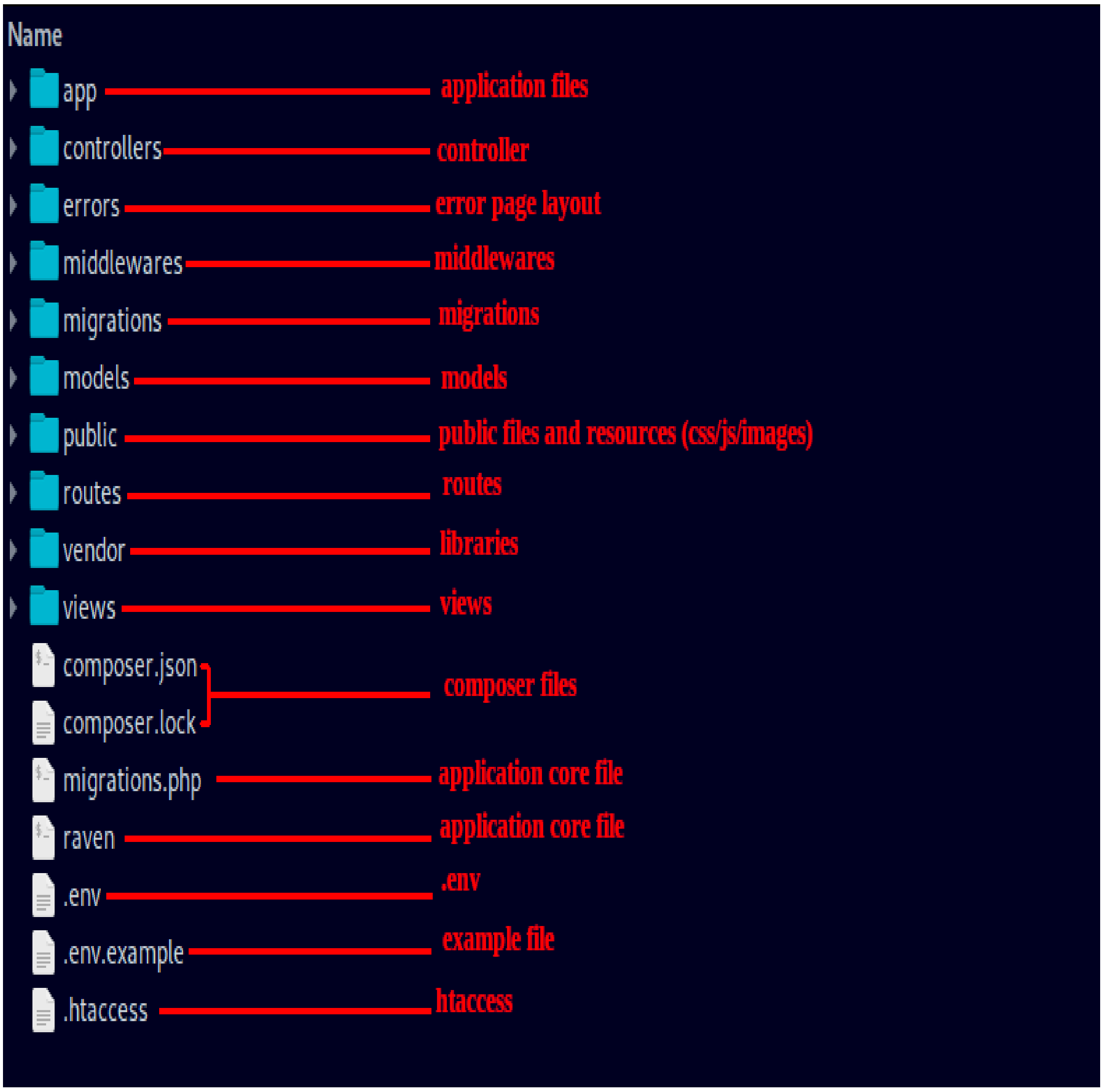
open cmd/terminal and run command:

```
$composer create-project muqadaskk/raven <path>
```

example:

```
$composer create-project muqadaskk/raven myApp
```

Files Structure



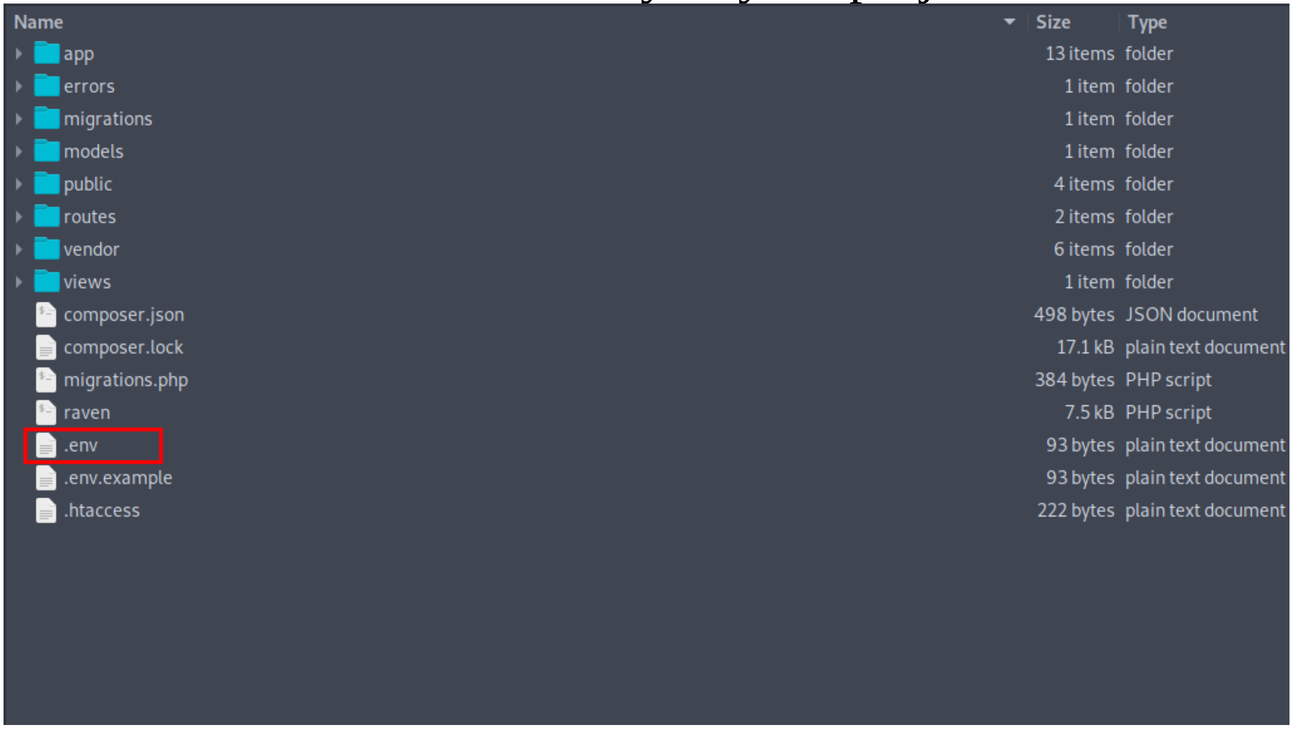
Name	
app	application files
controllers	controller
errors	error page layout
middlewares	middlewares
migrations	migrations
models	models
public	public files and resources (css/js/images)
routes	routes
vendor	libraries
views	views
composer.json	composer files
composer.lock	composer files
migrations.php	application core file
raven	application core file
.env	.env
.env.example	example file
.htaccess	htaccess

Setup

Raven uses vlucas/phpdotenv for database.

Setup .ENV

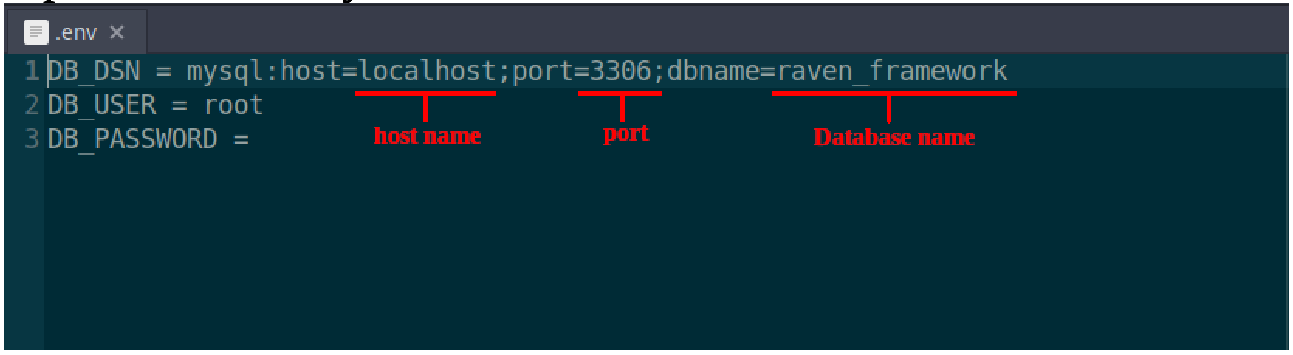
.env is located at root directory of your project.



```
Name                    Size        Type
app                     13 items    folder
errors                  1 item      folder
migrations              1 item      folder
models                  1 item      folder
public                  4 items     folder
routes                  2 items     folder
vendor                  6 items     folder
views                   1 item      folder
composer.json           498 bytes   JSON document
composer.lock           17.1 kB     plain text document
migrations.php          384 bytes   PHP script
raven                   7.5 kB      PHP script
.env                    93 bytes    plain text document
.env.example            93 bytes    plain text document
.htaccess               222 bytes   plain text document
```

<fig 0.1> If you can’t find .env in projects directory press ctrl+h in linux or go to [navbar→view→show hidden files].

Open .env in any text editor.



```
DB_DSN = mysql:host=localhost;port=3306;dbname=raven_framework
DB_USER = root
DB_PASSWORD =
```

<fig 0.1> set your mysql host, port, database name, user and password.

Raven(CLI)

raven is **command line interface** (**CLI**) that comes with **Raven** framework. It is powerful tool that allows you to perform a variety of tasks, such as creating migrations, controllers, models, middlewares and Creating PHP development server.

Usage :- open cmd/terminal in root directory of your project and run command:
for help→ **$php raven -h** or **$php raven --help**

Controllers

A controller is one of three main components of the **Model-View-Controller** (**MVC**) architecture.
Controller is responsible for handling user requests and interacting with model and view components.
It receives user input and sends output to the user, and also interacts with the database and external services. In Raven controller is represented by a class that extends the base controller class, and it usually contains a set of methods for handling user requests.

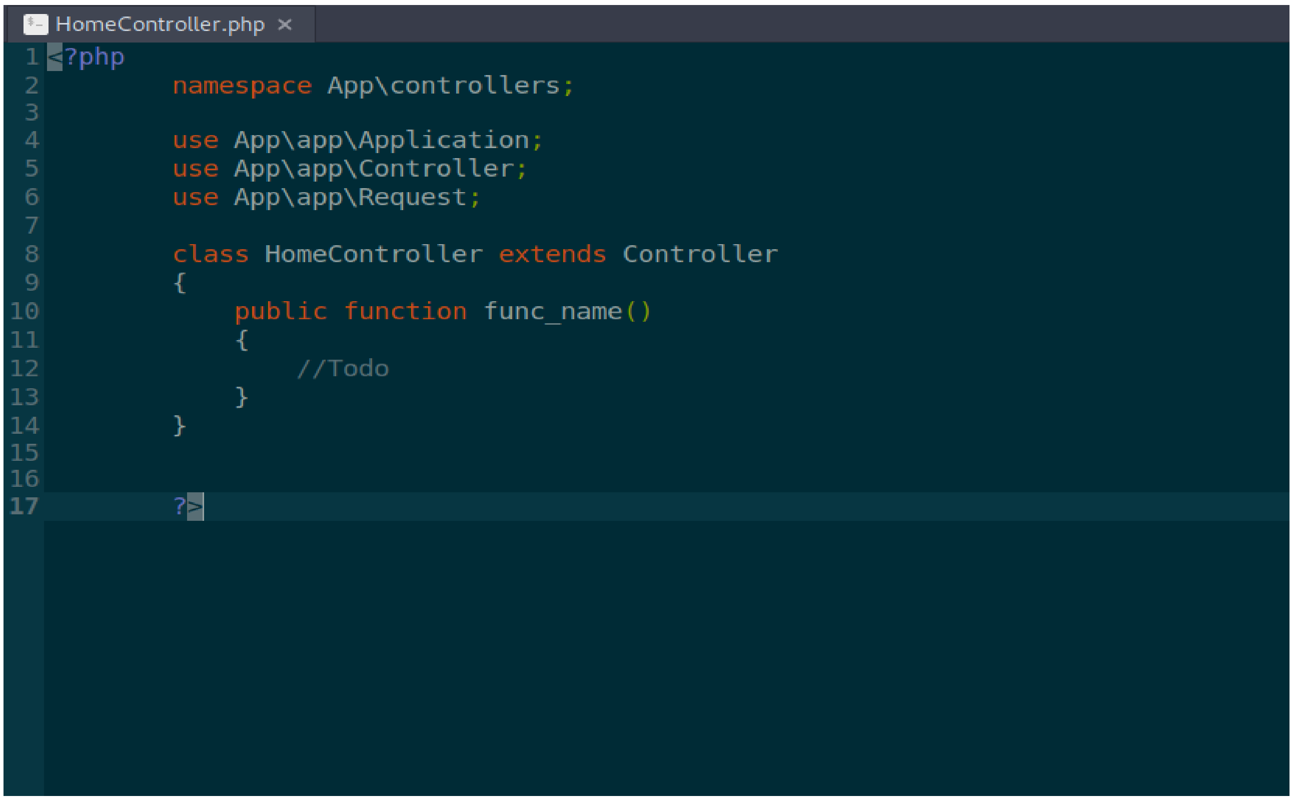
Creating Controller

Controllers are created with help of **raven** (**CLI**) and located in */controllers* directory.

```
$php raven --CreateController <controllername>
$php raven --CreateController HomeController
```

Defining Method

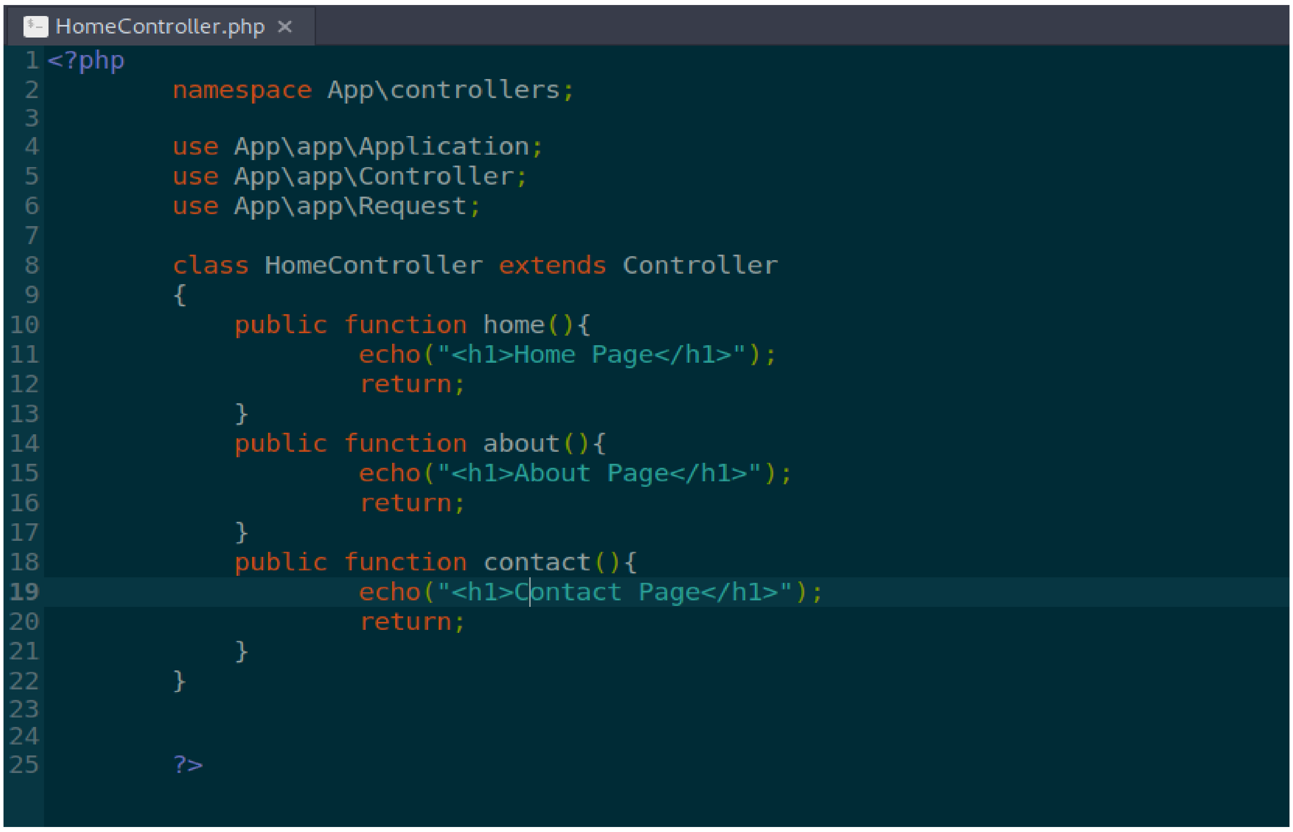
A method is a function that is defined within a controller class in **Raven**. It is used to handle specific user requests, such as creating, reading, updating or deleting data. For example a controller might have a create method that handles the creation of new record in the database, or a read method that retrieve data from database. In Raven methods are typically named in a consistent way to make them easy to identify and understand.
Example:



HomeController.php

```php
<?php
        namespace App\controllers;

        use App\app\Application;
        use App\app\Controller;
        use App\app\Request;

        class HomeController extends Controller
        {
            public function func_name()
            {
                //Todo
            }
        }


        ?>
```

<fig 0.1>We have a new controller class named as Home controller.



HomeController.php

```php
<?php
        namespace App\controllers;

        use App\app\Application;
        use App\app\Controller;
        use App\app\Request;

        class HomeController extends Controller
        {
            public function home(){
                    echo("<h1>Home Page</h1>");
                    return;
            }
            public function about(){
                    echo("<h1>About Page</h1>");
                    return;
            }
            public function contact(){
                    echo("<h1>Contact Page</h1>");
                    return;
            }
        }


        ?>
```

we have created three methods that handles different requests

- method home() handles “/” or “/home” requests and in response print an HTML heading tag <h1> with text “Home Page” on users browser screen.

- method about() handles “/about” request and in response print an HTML heading tag <h1> with text “About Page” on users browser screen.

- method contact() handles “/contact” request and in response print an HTML heading tag <h1> with text “Contact Page” on users browser screen.

Return Methods

The return statement is used to send a response to the user in a controller method in Raven. It can be used to return a view, a redirect, or some other type of response. The return statement is usually the last statement in a controller method, and it is often used in conjunction with the render() or redirect() methods. It is an important part of the controller’s job to return the appropriate response to the user.

- **Render()**

Example: **return $this->render("home");**
this will return a view in response.
#Note: Use only name of view file ignore extension **“.php”**.
If filename is ‘*home.php’* use ‘home’ when returning in response.

- **Abort()**

Example: **return $this->abort("Error 404");**
this will return a Error message in response.
If you want to use custom layout for error page customize layout file it can be found in *./errors* directory.

- **Back()**

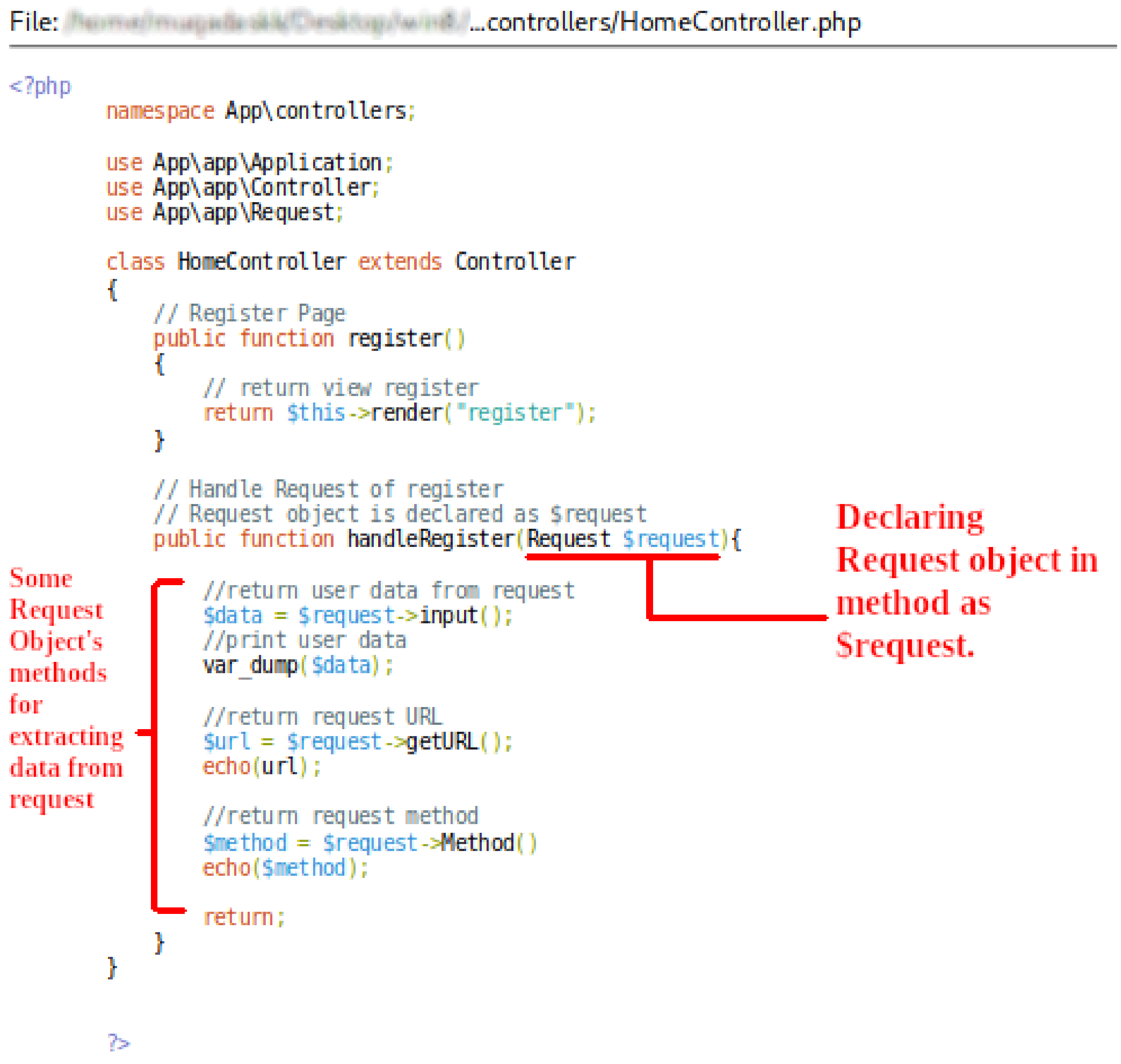
Example: **return $this->back();**
this will return redirect user back to previous path.

- **Redirect()**

Example: **return $this->redirect("/about");**
this will return redirect to /about path.

Handle Requests

The Request is handled with Request object in Raven which is an instance of the “App\app\Request” class. This object contains all of the information about the request, including URL, the method, the header and the query parameters. The Controller use this object to determine how to handle the request.
Request class included in controller class by default.
Request Object is typically declared in method’s parameter.


Declaring Request object in method as $request.
Some Request Object's methods for extracting data from request

```
File: [redacted]...controllers/HomeController.php

<?php
        namespace App\controllers;

        use App\app\Application;
        use App\app\Controller;
        use App\app\Request;

        class HomeController extends Controller
        {
            // Register Page
            public function register()
            {
                // return view register
                return $this->render("register");
            }

            // Handle Request of register
            // Request object is declared as $request
            public function handleRegister(Request $request){

                //return user data from request
                $data = $request->input();
                //print user data
                var_dump($data);

                //return request URL
                $url = $request->getURL();
                echo(url);

                //return request method
                $method = $request->Method()
                echo($method);

                return;
            }
        }


        ?>
```

<fig 0.1> This is a figure of a controller “**HomeController**” which have two methods “**register()**” and “**handleRegister()**”.
Method “**register()**” return a view of register page in */views* directory.
Method “**handleRegister()**” has an Object of Request declared as **$request** in parameter => “**handleRegister(Request $request)**” for handling request,
so **$request** object can be used to extract data from request.
There are some methods are used in <fig 0.1> for extracting data from request like form data, URL, and Method of request.

Request Methods

- **isGet()**

Example: $request->isGet();
return True if request method is GET.

- **IsPost()**

Example: $request->isPost();
return True if request method is Post.

- **IsPut()**

Example: $request->isPut();
return True if request method is PUT.

- **IsDelete()**

Example: $request->isDelete();
return True if request method is DELETE.

- **Method()**

Example: $request->Method();
return request method type ‘get’,‘post’‘put’ or ‘delete’.

- **input()**

Example: $request->input();
retrieve data from query parameters of request.

Validate
validate()
Check validation
isValidate()

Hashing

• **hash()**

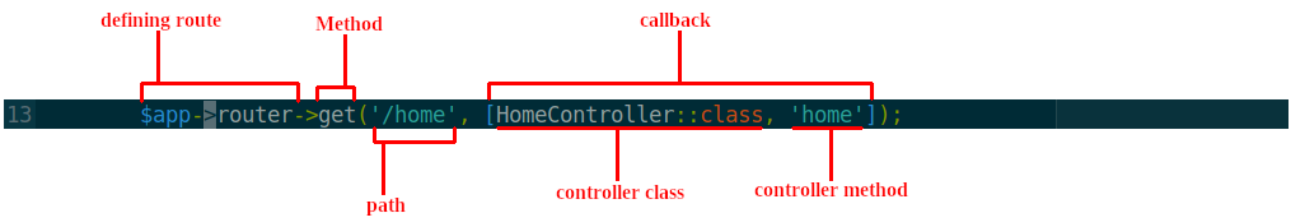
Example: **$this->hash(“secret”);**

return hashed value of “secret” word mostly used for password hashing.

Routes

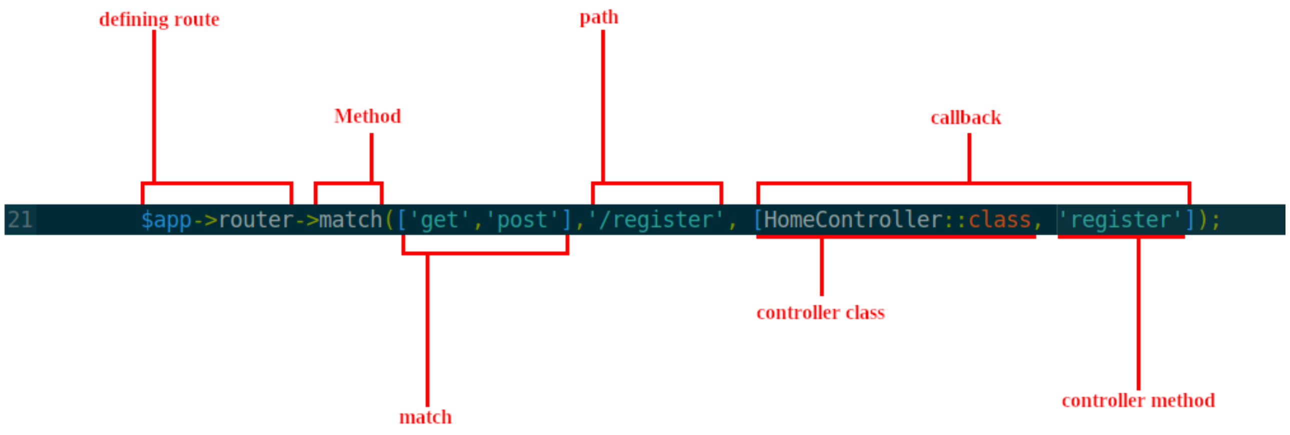
Routes are the heart of the **Raven** framework. They allow you to define the **paths** that will be used to access your application’s **controllers** and **views**.
Raven uses a simple and intuitive syntax for defining routes, which make it easy to get started.

Route Structure:-



```
$app->router->get('/home', [HomeController::class, 'home']);
```

<fig 0.1> Single Method (get, post, put, delete)



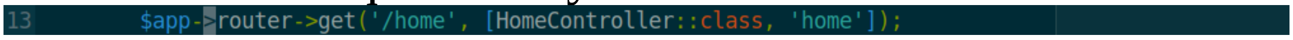
```
$app->router->match(['get','post'],'/register', [HomeController::class, 'register']);
```

<fig 0.2> Multiple Methods match(get, post, put, delete)

Route Methods

- **GET**

  handle Get requests only.

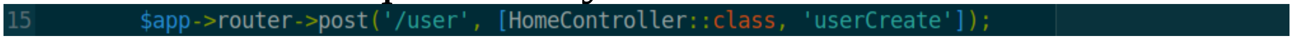


```php
$app->router->get('/home', [HomeController::class, 'home']);
```

<fig 0.1> example get route

- **POST**

  handle Post requests only.

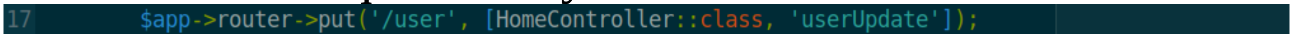


```php
$app->router->post('/user', [HomeController::class, 'userCreate']);
```

<fig 0.1> example post route

- **PUT**

  handle Put requests only.

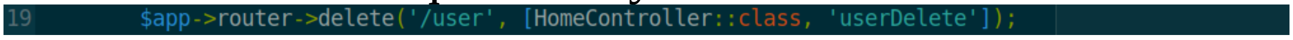


```php
$app->router->put('/user', [HomeController::class, 'userUpdate']);
```

<fig 0.1> example put route

- **DELETE**

  handle Delete requests only

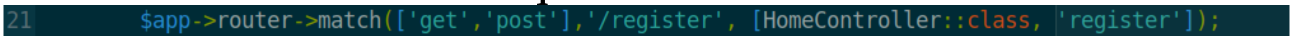


```php
$app->router->delete('/user', [HomeController::class, 'userDelete']);
```

<fig 0.1> example delete route

- **MATCH**

  handle all methods requests.



```php
$app->router->match(['get','post'],'/register', [HomeController::class, 'register']);
```

<fig 0.1> example match route

Route Path

Route Path is URL path that is associated with a particular route.

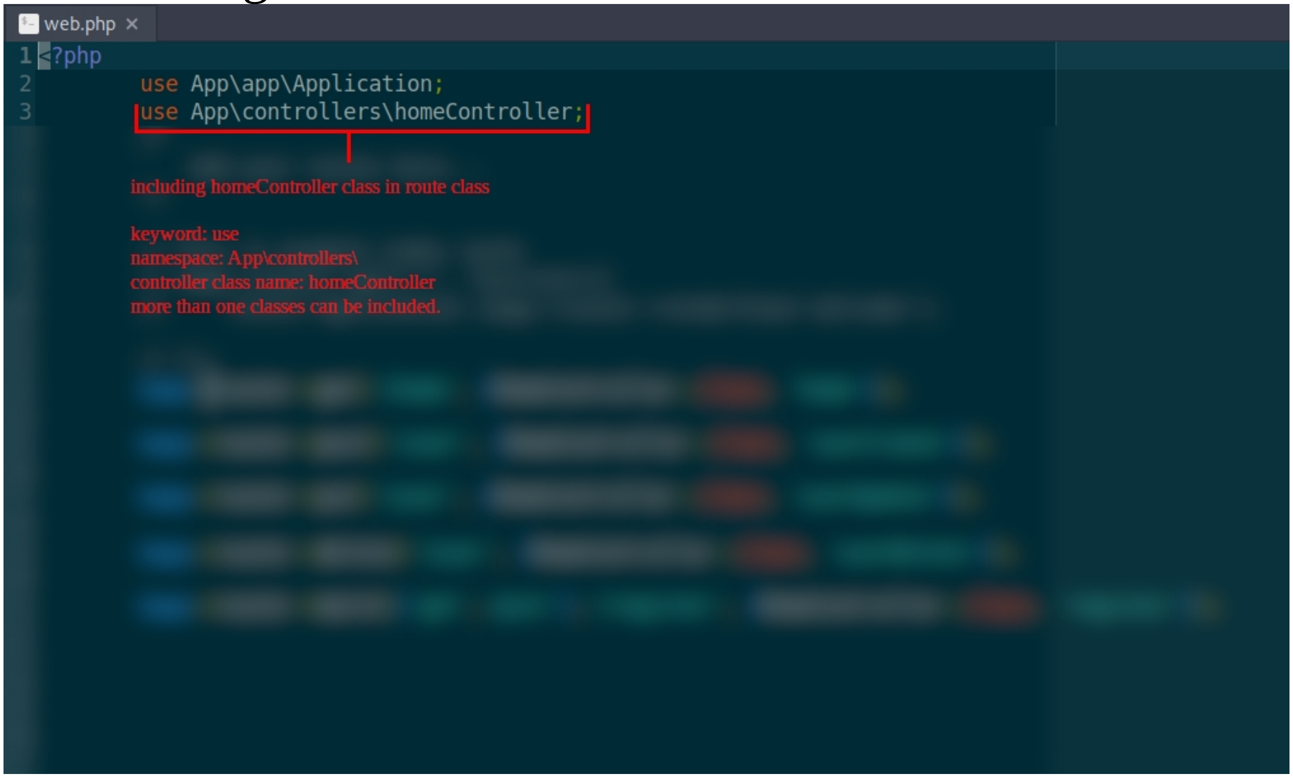
Route Callback

Route Callback is the function that is called when a route is matched. Callback have two parts controller class and method.

The method from controller class will be called when user visit path.

using Controllers

It is important to include controller class in routes class before using controller in route.



```php
<?php
        use App\app\Application;
        use App\controllers\homeController;
```

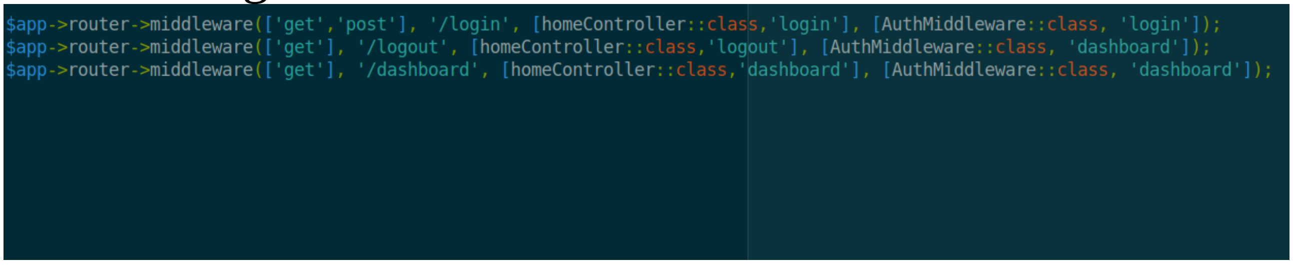
including homeController class in route class

keyword: use
namespace: App\controllers\
controller class name: homeController
more than one classes can be included.

<fig 0.1> including **homeController** class in route class.

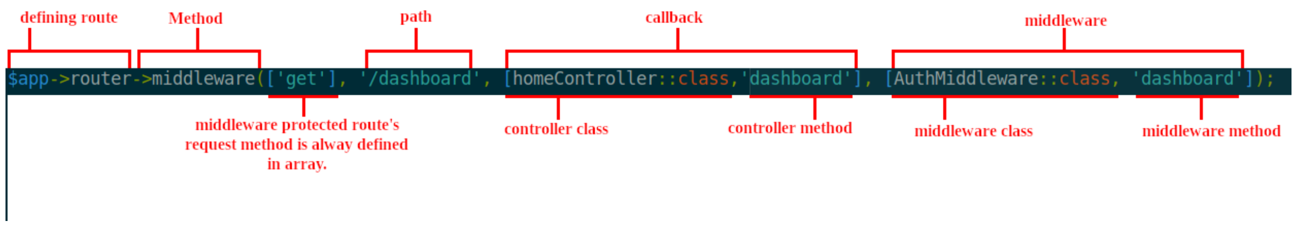
using Middlewares

It is important to include Middleware class in routes class before using Middleware in route same as controller.



```php
$app->router->middleware(['get','post'], '/login', [homeController::class,'login'], [AuthMiddleware::class, 'login']);
$app->router->middleware(['get'], '/logout', [homeController::class,'logout'], [AuthMiddleware::class, 'dashboard']);
$app->router->middleware(['get'], '/dashboard', [homeController::class,'dashboard'], [AuthMiddleware::class, 'dashboard']);
```

<fig 0.1> In above figure we have routes protected with AuthMiddleware.



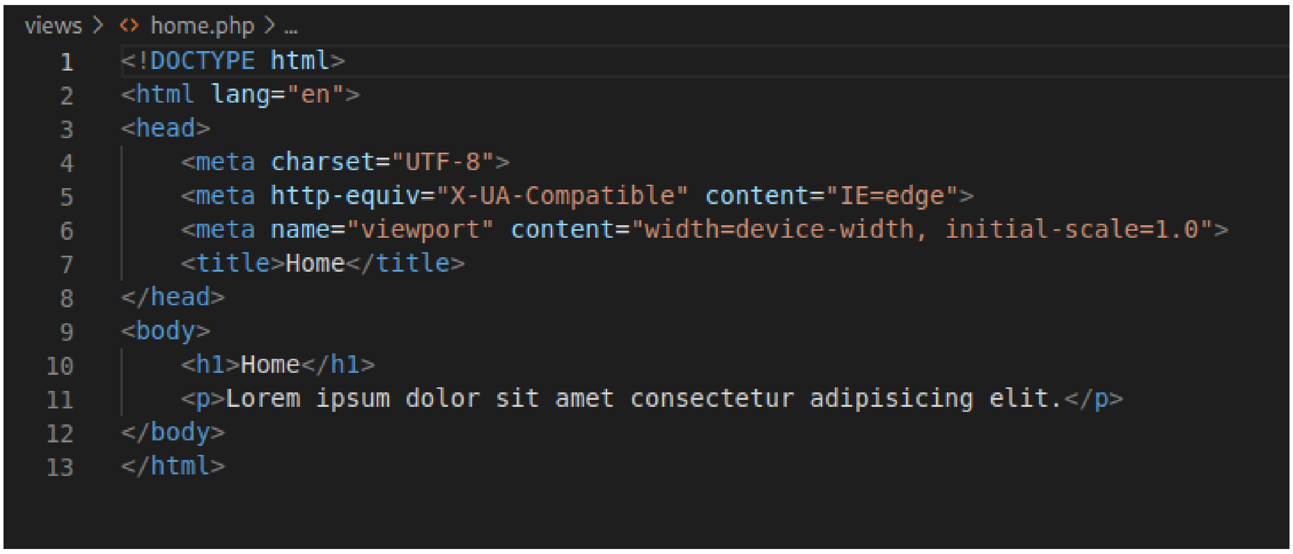
```php
$app->router->middleware(['get'], '/dashboard', [homeController::class,'dashboard'], [AuthMiddleware::class, 'dashboard']);
```

<fig 0.1> middleware protected route defining method and structure.

Views

A View is a file contains the **HTML** and other data that is displayed to the user. It is typically stored in */views* directory and has **.php** extension.

When a **controller** returns a response to the user, it will typically include a **view** that contains the **HTML** that is sent to the user's browser. View can also contain dynamic data such as data from the **request** or **database**, which is inserted into **HTML** using **PHP**. View are important part of **MVC** patterns, as they separate the presentation logic from business logic.

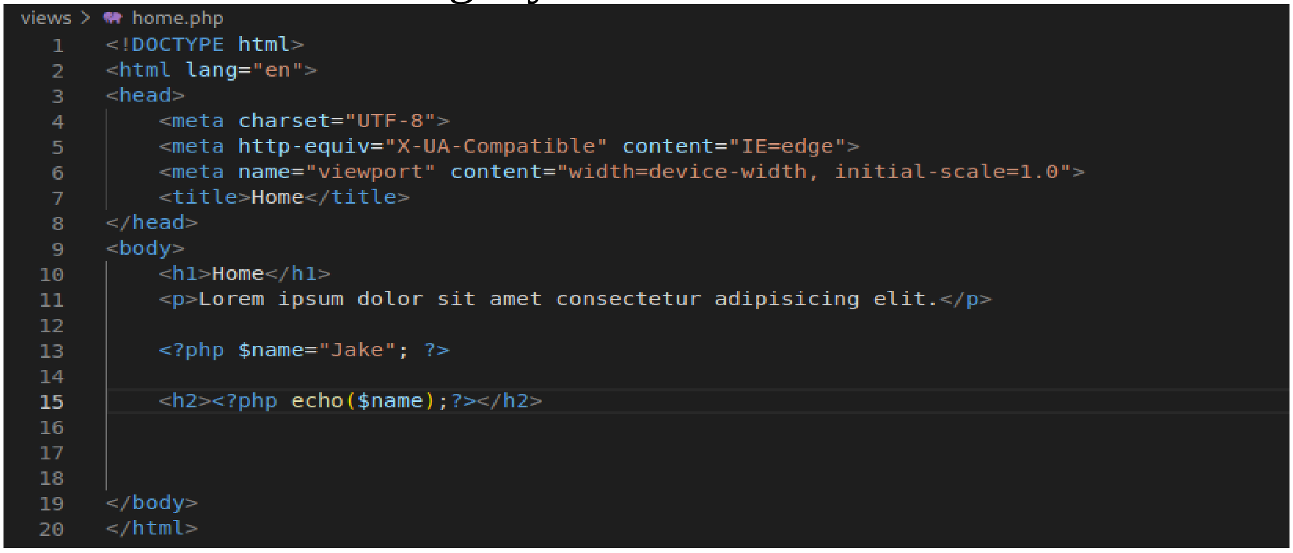


```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Home</title>
</head>
<body>
    <h1>Home</h1>
    <p>Lorem ipsum dolor sit amet consectetur adipisicing elit.</p>
</body>
</html>
```

<fig 0.1> A home page view file in */view* directory.

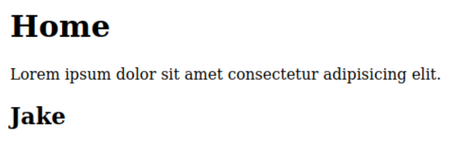
Using Dynamic Data

PHP is used for using Dynamic data in **View** file.



```
views > home.php
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Home</title>
</head>
<body>
    <h1>Home</h1>
    <p>Lorem ipsum dolor sit amet consectetur adipisicing elit.</p>

    <?php $name="Jake"; ?>

    <h2><?php echo($name);?></h2>



</body>
</html>
```

<fig 0.1> Some php code in view.

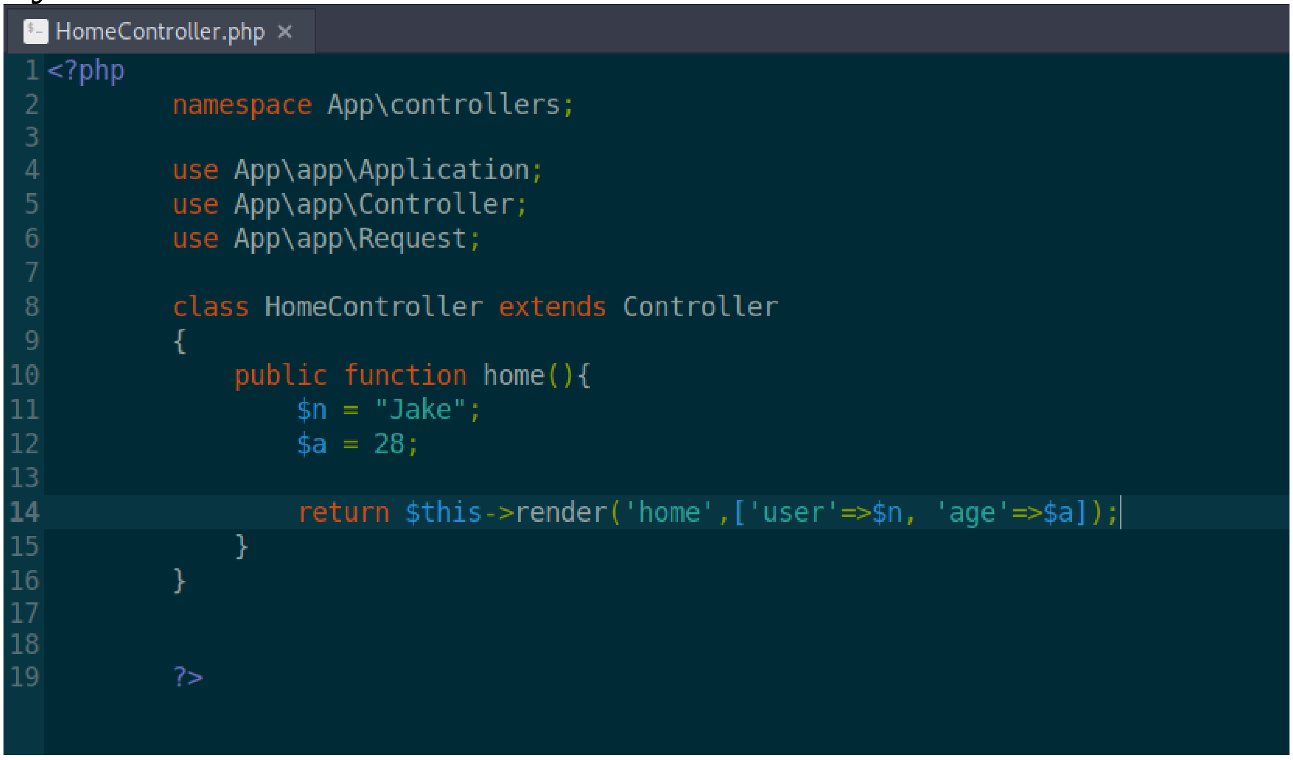


```
Home
Lorem ipsum dolor sit amet consectetur adipisicing elit.
Jake
```

<fig 0.2> output.

Passing Data to View

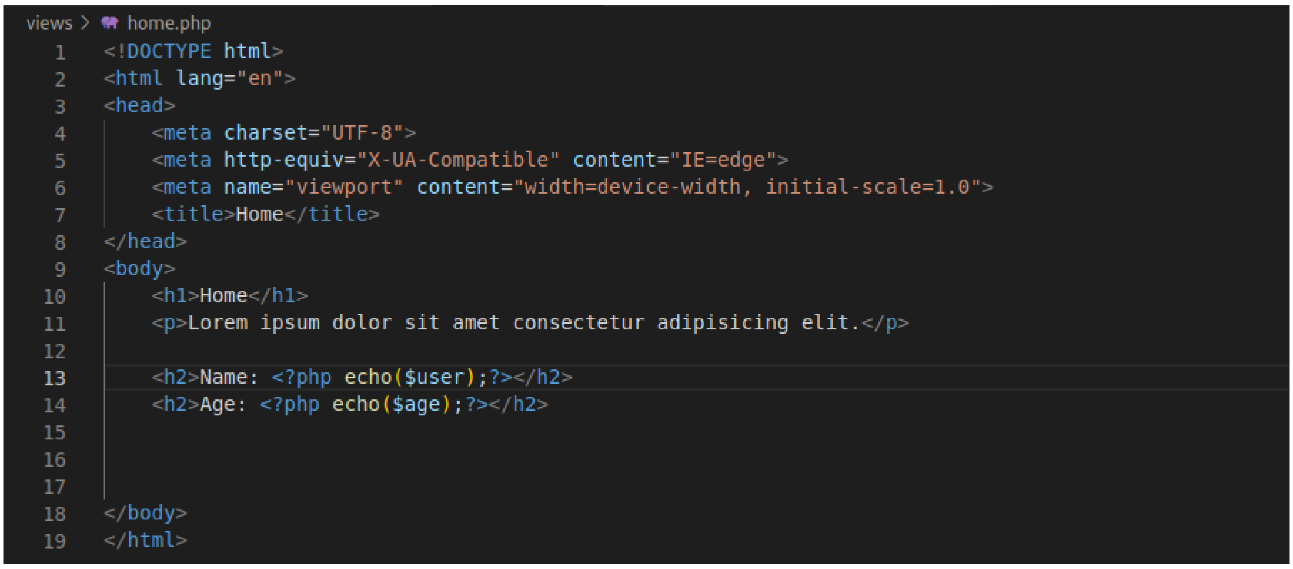
passing data from **controller** to **view** is process to use dynamic data in **HTML**.



HomeController.php

```php
<?php
        namespace App\controllers;

        use App\app\Application;
        use App\app\Controller;
        use App\app\Request;

        class HomeController extends Controller
        {
            public function home(){
                $n = "Jake";
                $a = 28;

                return $this->render('home',['user'=>$n, 'age'=>$a]);
            }
        }


        ?>
```

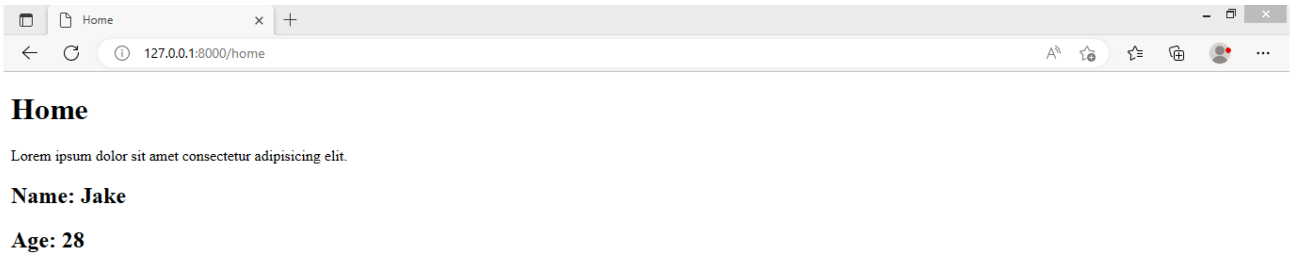
<fig 0.1> Two variable (**$n** and **$a**) in **controller** having values **string** name and **int** age.
Variable are passed to **view** with **render** method can be accessed in **view** with variables (‘**$user**’ and “**age**”).



views > home.php

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Home</title>
</head>
<body>
    <h1>Home</h1>
    <p>Lorem ipsum dolor sit amet consectetur adipisicing elit.</p>

    <h2>Name: <?php echo($user);?></h2>
    <h2>Age: <?php echo($age);?></h2>



</body>
</html>
```

<fig 0.2> In **view** we can access data of **$n** and **$a** variable as **$user** and **$age**.



Home

Lorem ipsum dolor sit amet consectetur adipisicing elit.

Name: Jake

Age: 28

<fig 0.3> Output

You can pass **string**, **int**, and **array** from **controller** to **view** with that method.

Migration

Migration is a process of moving data from one database schema to another. This is typically done when structure of database needs to be changed, such as adding new column or table.

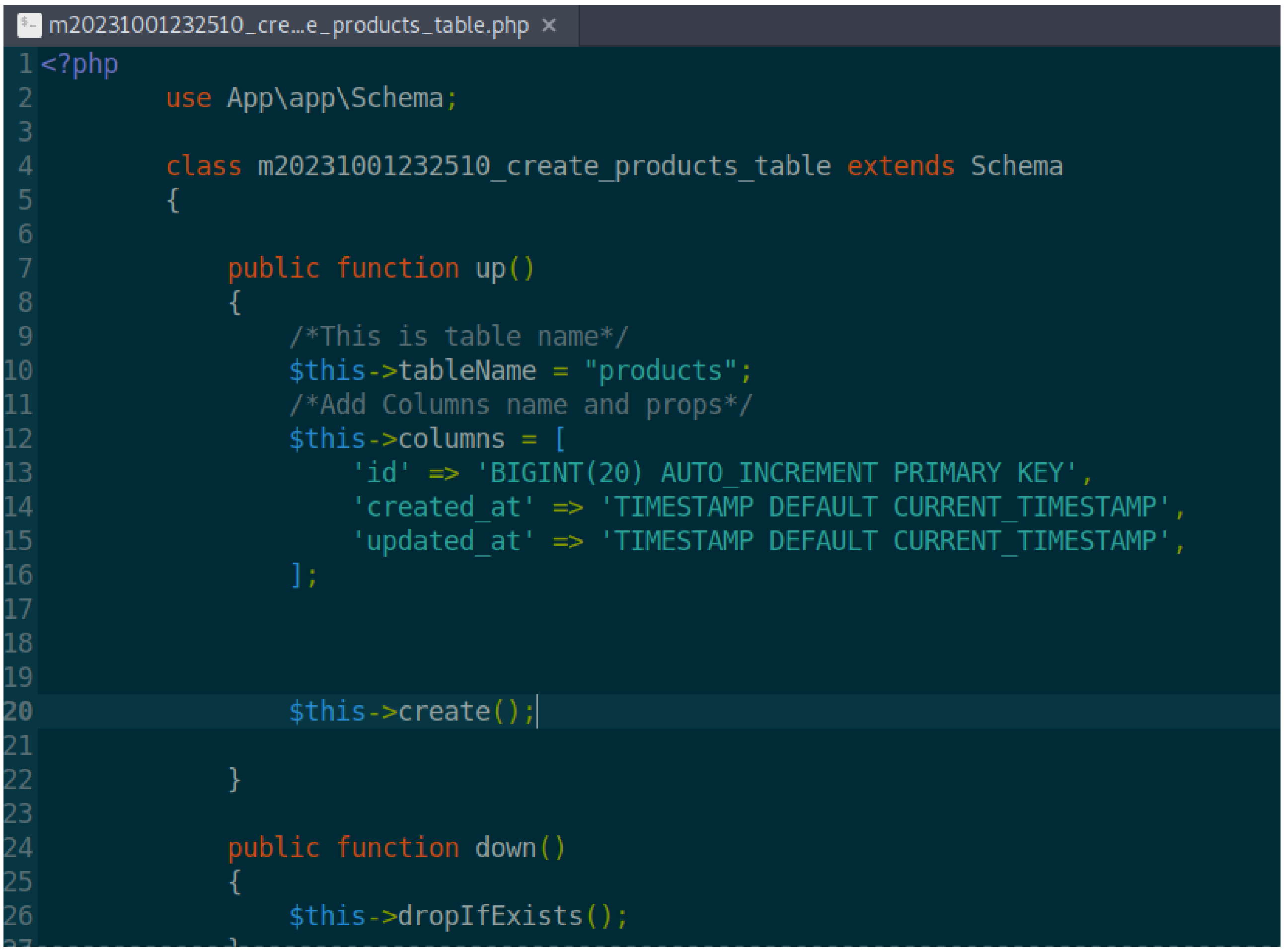
Creating Migration Class

Migration are created with help of **raven** (**CLI**) and located in */migrations* directory.

```
$php raven --CreateMigration <Migrationname>
$php raven --CreateMigration products
```

Defining Columns

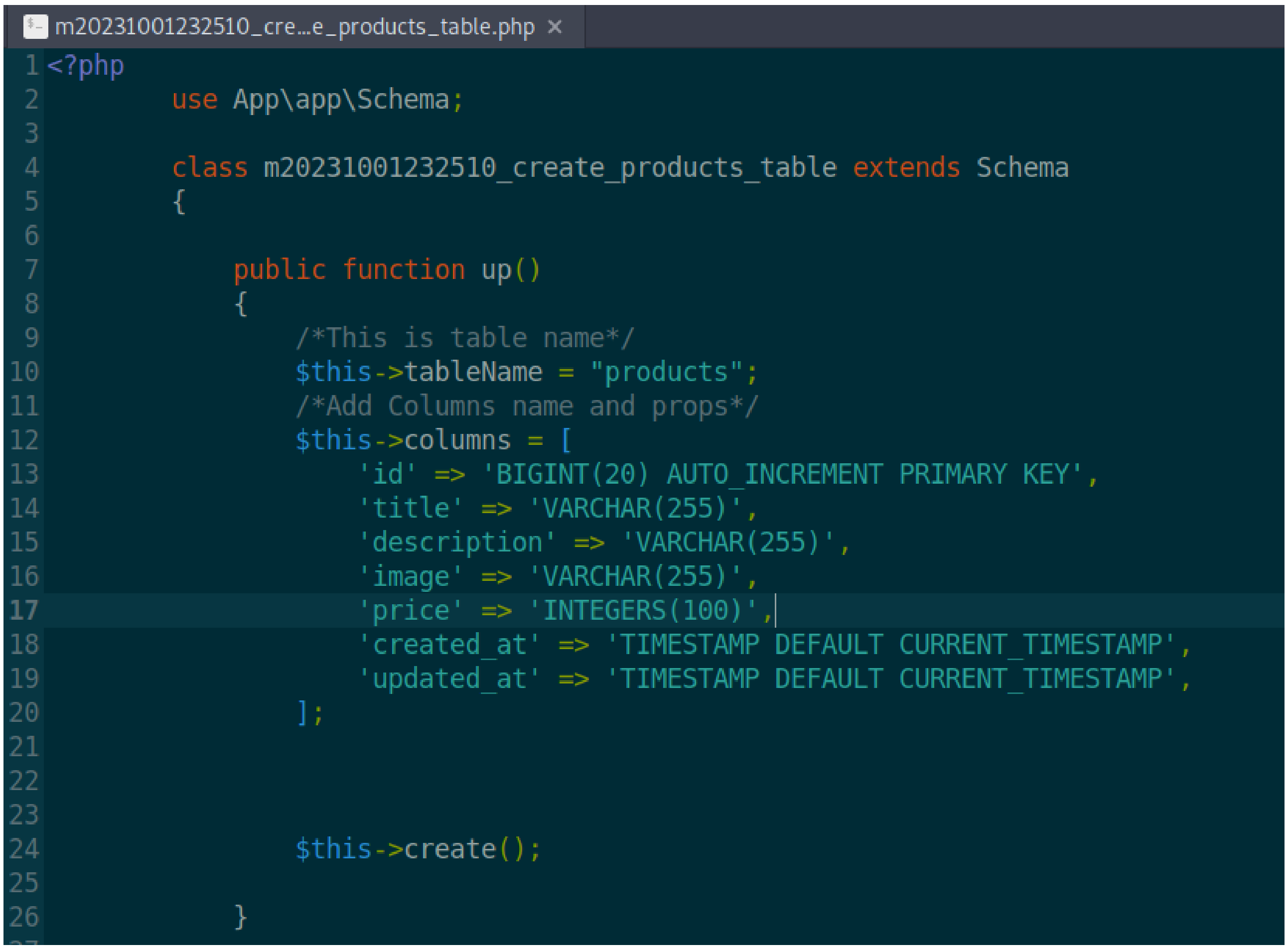
It is important to define **Columns** in **migration's** class before **migrating database**. Your **migrations** classes are stored in *./migrations* directory.



```php
<?php
        use App\app\Schema;

        class m20231001232510_create_products_table extends Schema
        {

            public function up()
            {
                /*This is table name*/
                $this->tableName = "products";
                /*Add Columns name and props*/
                $this->columns = [
                    'id' => 'BIGINT(20) AUTO_INCREMENT PRIMARY KEY',
                    'created_at' => 'TIMESTAMP DEFAULT CURRENT_TIMESTAMP',
                    'updated_at' => 'TIMESTAMP DEFAULT CURRENT_TIMESTAMP',
                ];



                $this->create();

            }

            public function down()
            {
                $this->dropIfExists();
```

<fig 0.1> A newly created migration class with default columns.
Default columns are includes:

1. id
2. created_at
3. updated_at



m20231001232510_cre…e_products_table.php

```php
<?php
        use App\app\Schema;

        class m20231001232510_create_products_table extends Schema
        {

            public function up()
            {
                /*This is table name*/
                $this->tableName = "products";
                /*Add Columns name and props*/
                $this->columns = [
                    'id' => 'BIGINT(20) AUTO_INCREMENT PRIMARY KEY',
                    'title' => 'VARCHAR(255)',
                    'description' => 'VARCHAR(255)',
                    'image' => 'VARCHAR(255)',
                    'price' => 'INTEGERS(100)',
                    'created_at' => 'TIMESTAMP DEFAULT CURRENT_TIMESTAMP',
                    'updated_at' => 'TIMESTAMP DEFAULT CURRENT_TIMESTAMP',
                ];



                $this->create();

            }
```

<fig 0.2> After defining required **columns** in **migration** class.

Migrating

Migrating is final step to make change in database.
Migrating is done with help of **raven** (**CLI**).
Example: **$php raven --Migrate**

Model

In **raven** a **Model** is an object that represent he data stored in the database.
It is used to interact with the **database** and provides a clean interface for accessing and manipulating the data. A **model** is typically defined in a **PHP** class that extends ‘App\app\DbModel’ Class. This class provides a number of built-in methods for querying updating and deleting the **database**.

Models can be used in controller by including **namespace** “**use App\models\<Model Name>**”.

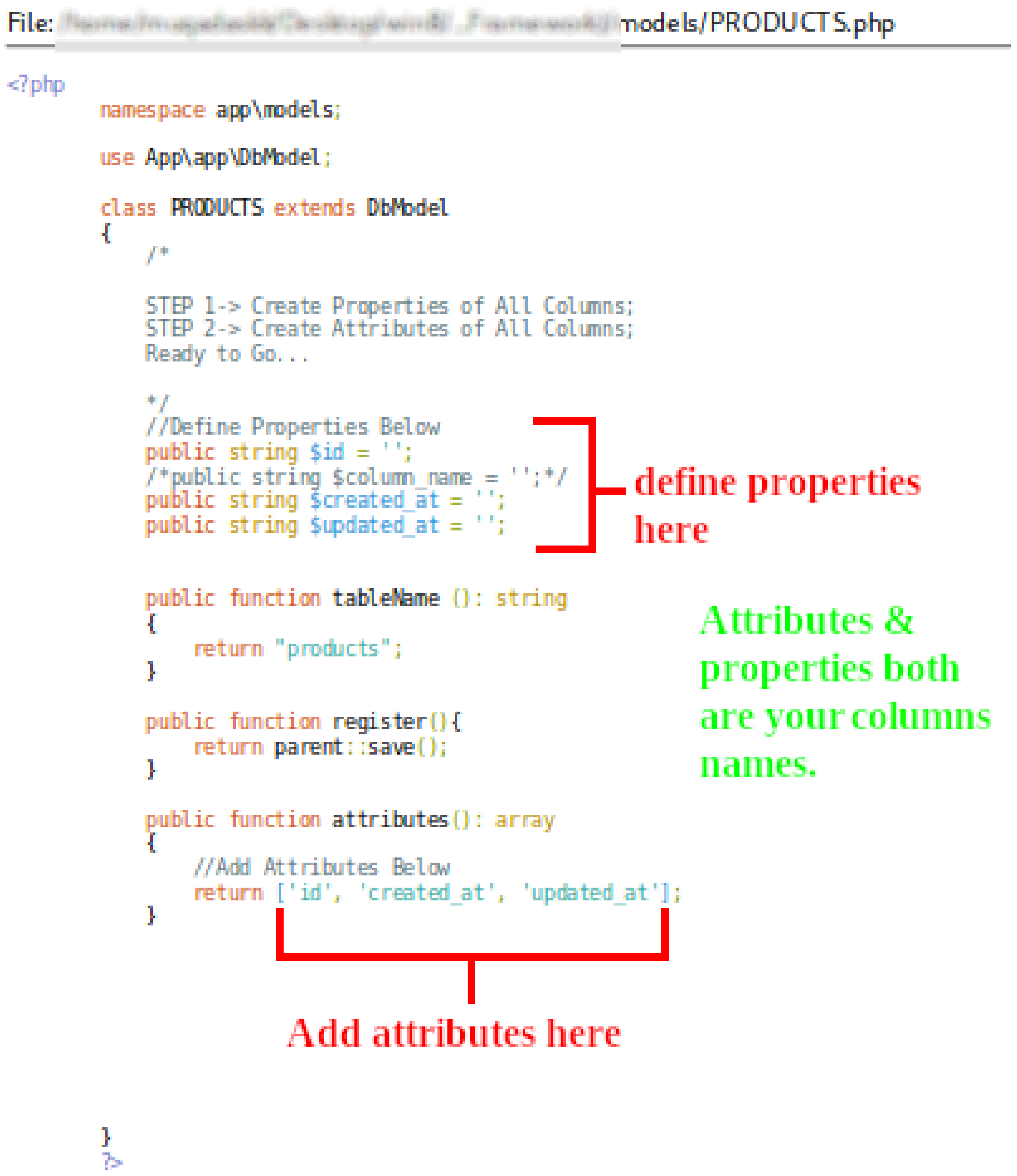
Creating Model

After creating table in database we can create **models** class. **Models** are created with help of **raven** (**CLI**) and located in */models* directory.

```
$php raven --CreateModel <ModelName>
$php raven --CreateModel Products
```

Defining Properties and Attributes

Defining properties and attributes sounds little complex but it is very easy process. It is process where we define which columns we want to access through this model.

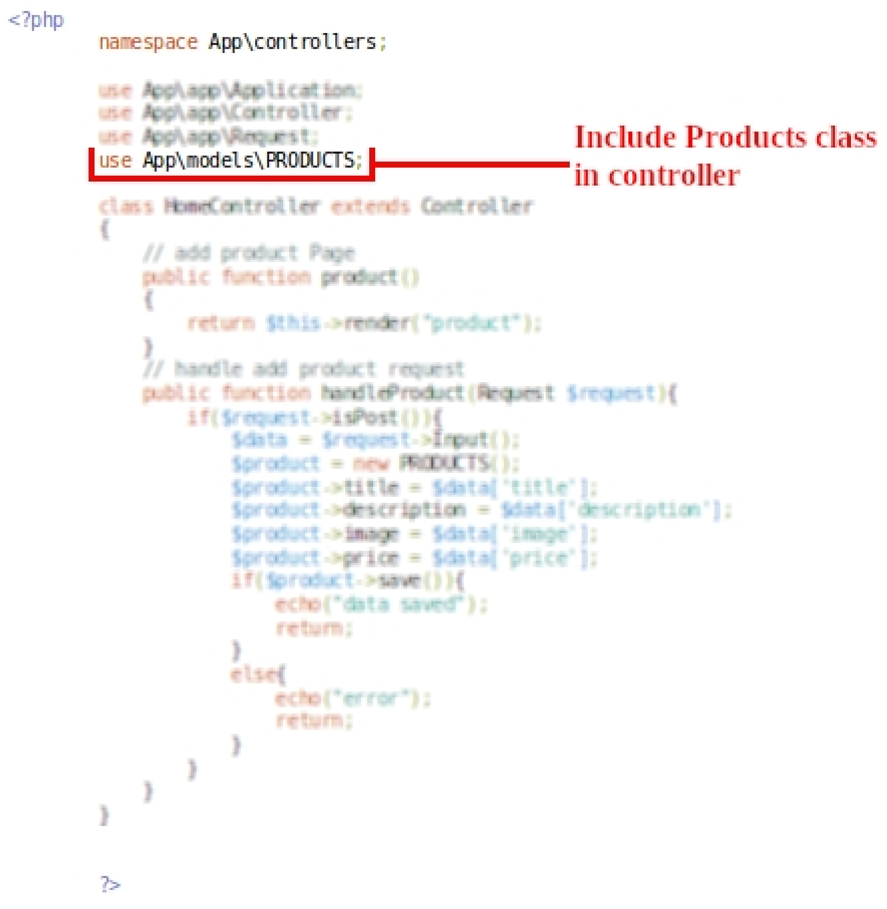

File: [redacted] models/PRODUCTS.php

```
<?php
        namespace app\models;

        use App\app\DbModel;

        class PRODUCTS extends DbModel
        {
            /*

            STEP 1-> Create Properties of All Columns;
            STEP 2-> Create Attributes of All Columns;
            Ready to Go...

            */
            //Define Properties Below
            public string $id = '';
            /*public string $column_name = '';*/
            public string $created_at = '';
            public string $updated_at = '';

            public function tableName (): string
            {
                return "products";
            }

            public function register(){
                return parent::save();
            }

            public function attributes(): array
            {
                //Add Attributes Below
                return ['id', 'created_at', 'updated_at'];
            }


        }
        ?>
```

<fig 0.1> In above figure we can see a newly created **products model** class which have some default **properties** and **attributes** these properties and attributes are column’s names in table **prdoucts**. We can add rest of column names in this class for accessing through this model.

Using Model in Controller

After defining attributes and properties we can use model in controller.



```php
<?php
    namespace App\controllers;

    use App\app\Application;
    use App\app\Controller;
    use App\app\Request;
    use App\models\PRODUCTS;

    class HomeController extends Controller
    {
        // add product Page
        public function product()
        {
            return $this->render("product");
        }
        // handle add product request
        public function handleProduct(Request $request){
            if($request->isPost()){
                $data = $request->Input();
                $product = new PRODUCTS();
                $product->title = $data['title'];
                $product->description = $data['description'];
                $product->image = $data['image'];
                $product->price = $data['price'];
                if($product->save()){
                    echo("data saved");
                    return;
                }
                else{
                    echo("error");
                    return;
                }
            }
        }
    }


    ?>
```

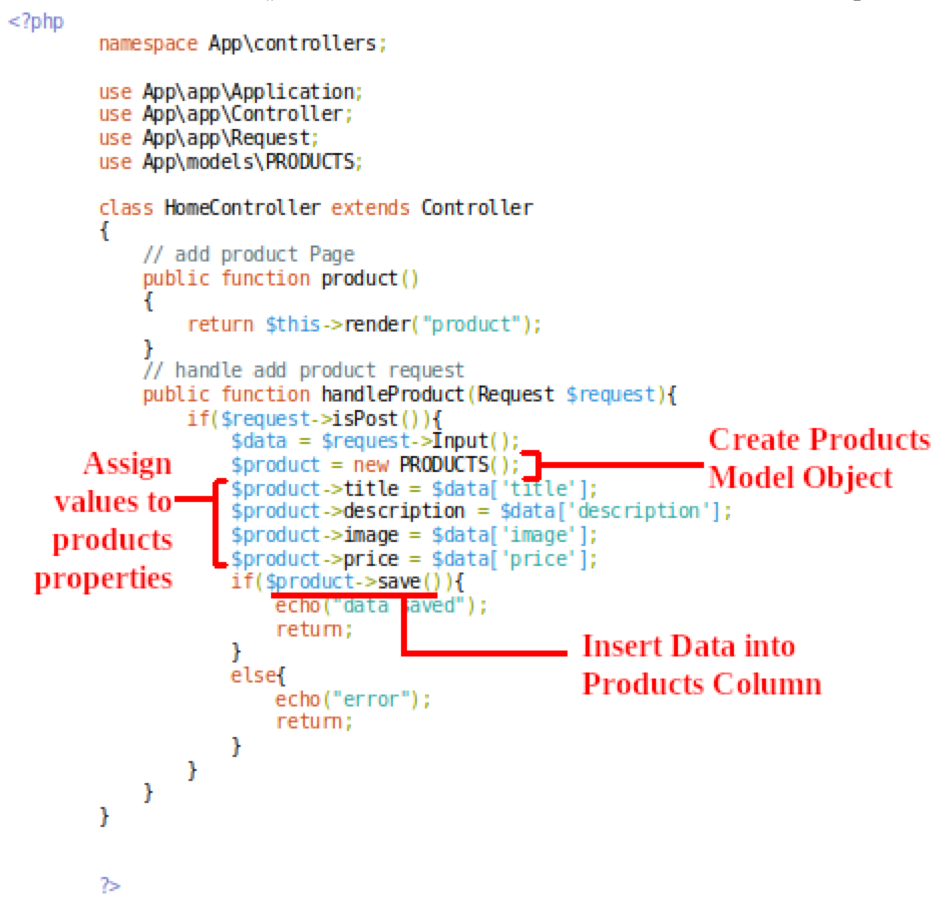
<fig 0.1> Import model.

Model Methods

4. **Save()**
5. **Fetch()**
6. **FetchAll()**
7. **Update()**
8. **Delete()**
9. **Where() “Condition”**

Save()

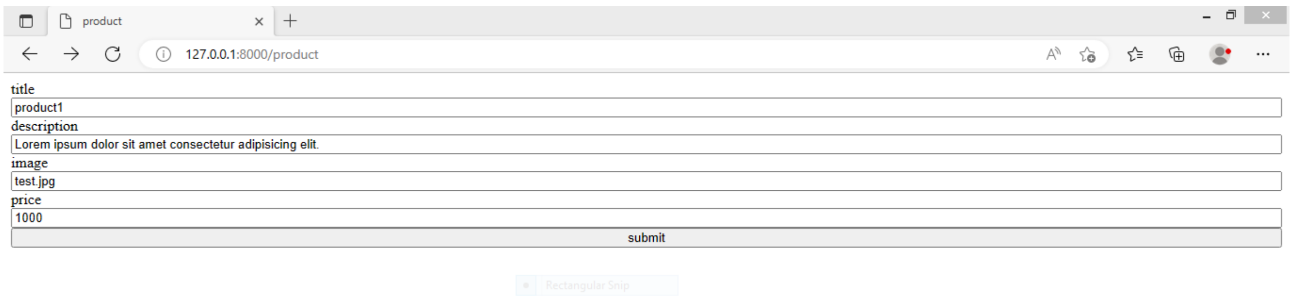
save() is method of Model, used to save data in table.
Typically this method is called after assigning values to model properties.
Method save() returns true or error “if any”.



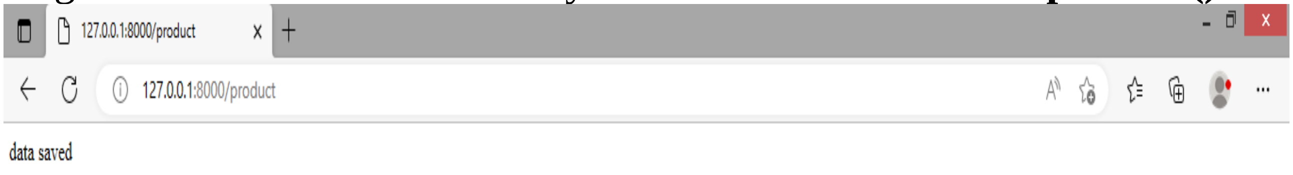
```php
<?php
        namespace App\controllers;

        use App\app\Application;
        use App\app\Controller;
        use App\app\Request;
        use App\models\PRODUCTS;

        class HomeController extends Controller
        {
            // add product Page
            public function product()
            {
                return $this->render("product");
            }
            // handle add product request
            public function handleProduct(Request $request){
                if($request->isPost()){
                    $data = $request->Input();
                    $product = new PRODUCTS();
                    $product->title = $data['title'];
                    $product->description = $data['description'];
                    $product->image = $data['image'];
                    $product->price = $data['price'];
                    if($product->save()){
                        echo("data saved");
                        return;
                    }
                    else{
                        echo("error");
                        return;
                    }
                }
            }
        }

        ?>
```

Assign values to products properties
Create Products Model Object
Insert Data into Products Column

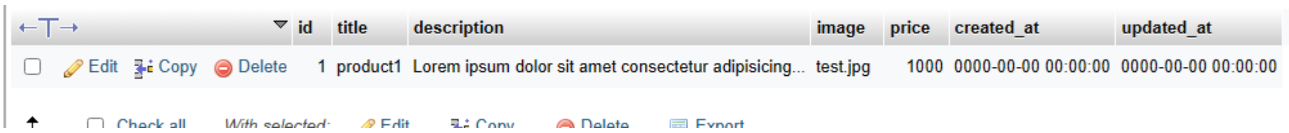
<fig 0.1> A method **(handleProduct)** in controller **(HomeController)** is handling request.
Method is using **Products model**.
Retrieve form data from **request** and store in **$data** variable.
Assign values of **$data** to **Products Model Object’s** properties.
Call method save() in condition if save() return true prints “data saved” else prints “error” in user’s browser.


title
product1
description
Lorem ipsum dolor sit amet consectetur adipisicing elit.
image
test.jpg
price
1000
submit

<fig 0.2> A form **view** handled by **HomeController** method **product()**.


data saved

<fig 0.3> Output of request handled by **HomeController** method **handleProduct()**.

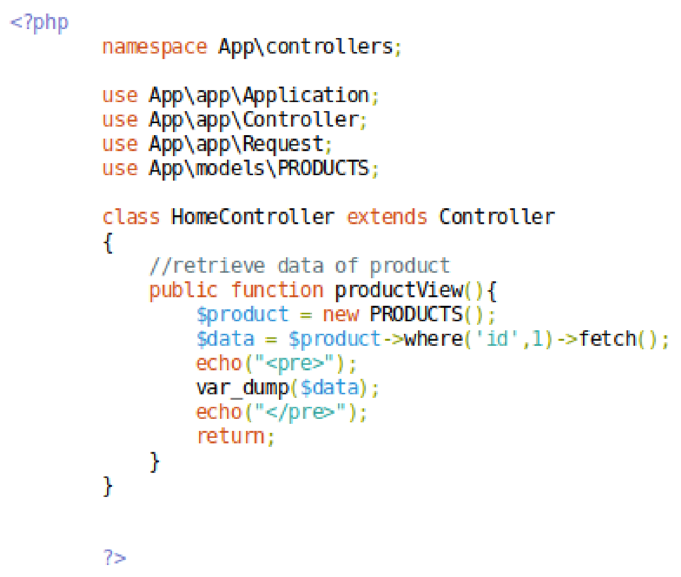


	id	title	description	image	price	created_at	updated_at
Edit Copy Delete	1	product1	Lorem ipsum dolor sit amet consectetur adipisicing...	test.jpg	1000	0000-00-00 00:00:00	0000-00-00 00:00:00

<fig 0.3> Result of data saved in table **Products.**

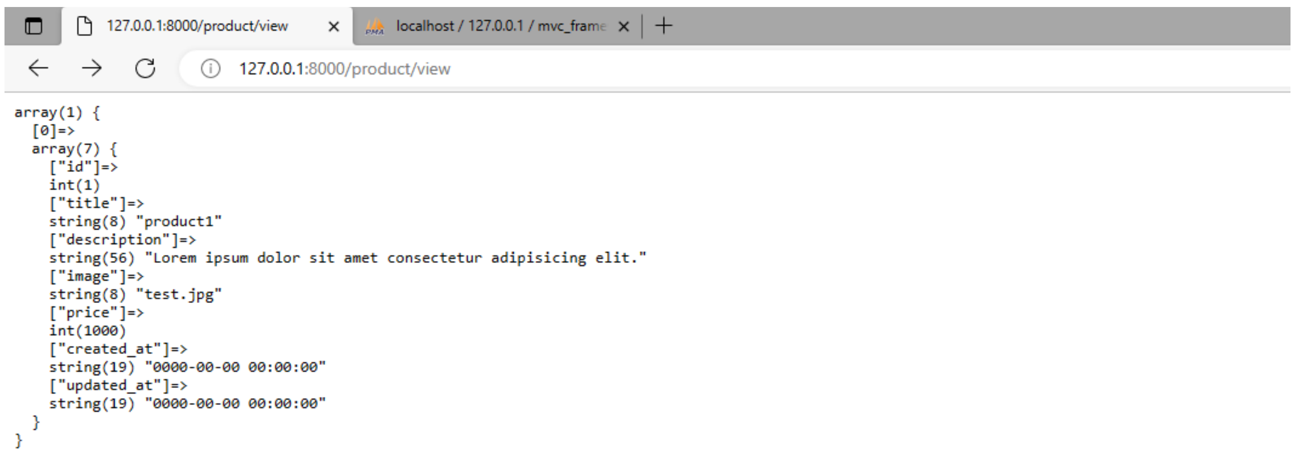
Fetch()

fetch() is method of Model, used to retrieve single column data from table. Typically this method is always used with combination of **where()** method (**$obj->where(prop,value)->fetch()**).
Method fetch() returns array or error “if any”.



```php
<?php
        namespace App\controllers;

        use App\app\Application;
        use App\app\Controller;
        use App\app\Request;
        use App\models\PRODUCTS;

        class HomeController extends Controller
        {
            //retrieve data of product
            public function productView(){
                $product = new PRODUCTS();
                $data = $product->where('id',1)->fetch();
                echo("<pre>");
                var_dump($data);
                echo("</pre>");
                return;
            }
        }


        ?>
```

<fig 0.1> fetching data from **products** table **where id=1**.



```
array(1) {
  [0]=>
  array(7) {
    ["id"]=>
    int(1)
    ["title"]=>
    string(8) "product1"
    ["description"]=>
    string(56) "Lorem ipsum dolor sit amet consectetur adipisicing elit."
    ["image"]=>
    string(8) "test.jpg"
    ["price"]=>
    int(1000)
    ["created_at"]=>
    string(19) "0000-00-00 00:00:00"
    ["updated_at"]=>
    string(19) "0000-00-00 00:00:00"
  }
}
```

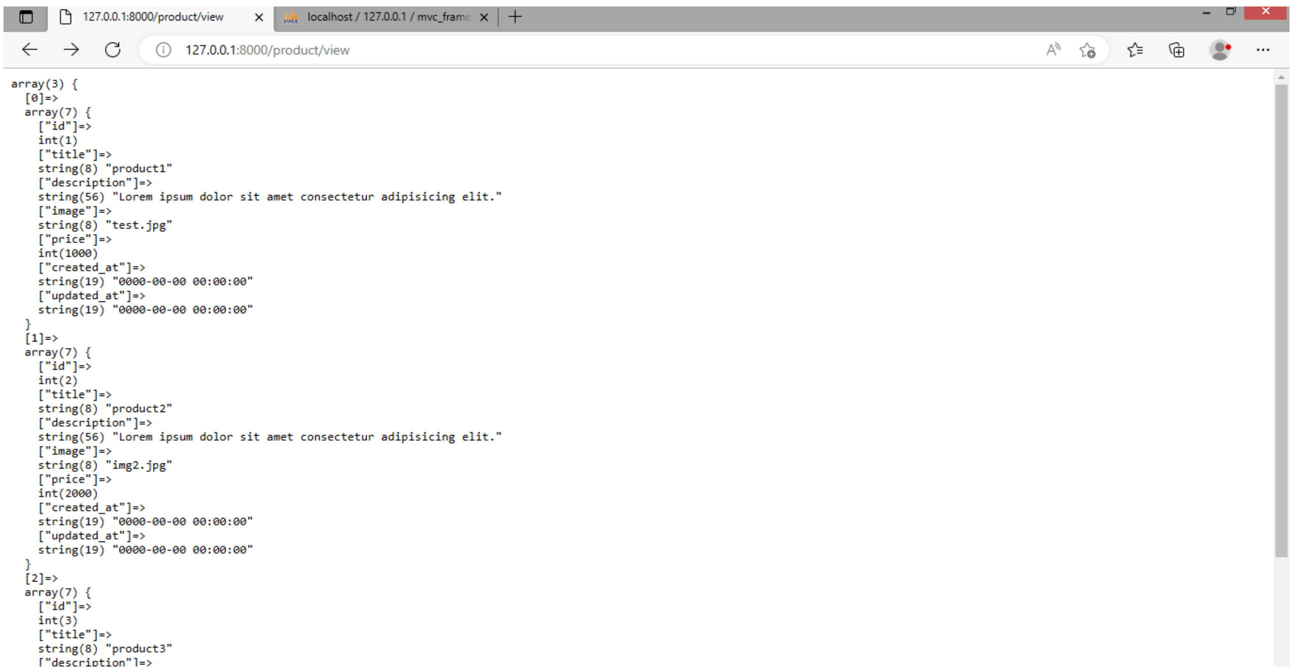
<fig 0.2> Output.

FetchAll()

fetchAll() is method of Model, used to retrieve all columns data from table.
Method fetchAll() returns array or error "if any".

```php
<?php
        namespace App\controllers;

        use App\app\Application;
        use App\app\Controller;
        use App\app\Request;
        use App\models\PRODUCTS;

        class HomeController extends Controller
        {
            //retrieve data of product
            public function productView(){
                $product = new PRODUCTS();
                $data = $product->fetchAll();
                echo("<pre>");
                var_dump($data);
                echo("</pre>");
                return;
            }
        }


        ?>
```

<fig 0.1> fetching all data from **products** table.


127.0.0.1:8000/product/view
localhost / 127.0.0.1 / mvc_frame
127.0.0.1:8000/product/view

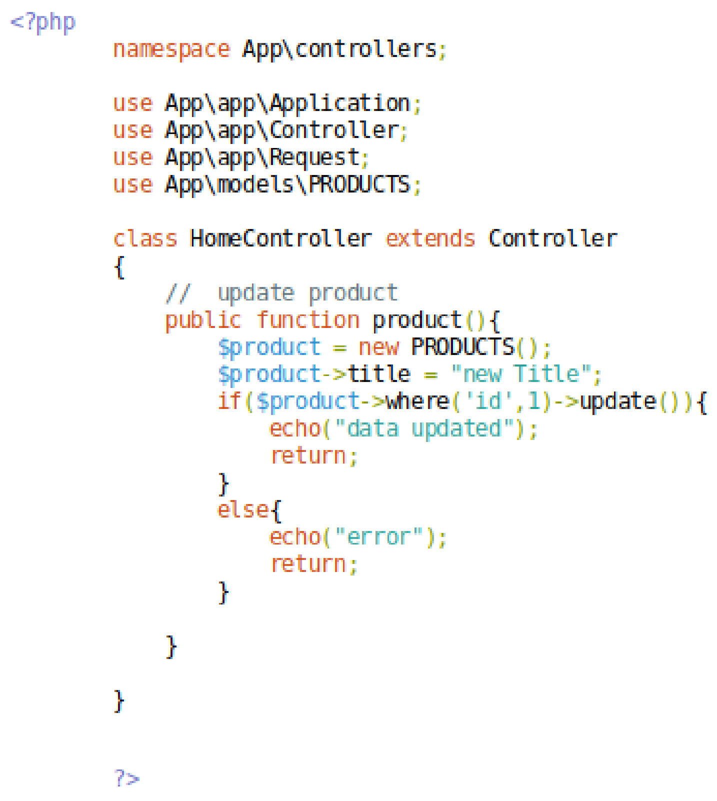
```
array(3) {
  [0]=>
  array(7) {
    ["id"]=>
    int(1)
    ["title"]=>
    string(8) "product1"
    ["description"]=>
    string(56) "Lorem ipsum dolor sit amet consectetur adipisicing elit."
    ["image"]=>
    string(8) "test.jpg"
    ["price"]=>
    int(1000)
    ["created_at"]=>
    string(19) "0000-00-00 00:00:00"
    ["updated_at"]=>
    string(19) "0000-00-00 00:00:00"
  }
  [1]=>
  array(7) {
    ["id"]=>
    int(2)
    ["title"]=>
    string(8) "product2"
    ["description"]=>
    string(56) "Lorem ipsum dolor sit amet consectetur adipisicing elit."
    ["image"]=>
    string(8) "img2.jpg"
    ["price"]=>
    int(2000)
    ["created_at"]=>
    string(19) "0000-00-00 00:00:00"
    ["updated_at"]=>
    string(19) "0000-00-00 00:00:00"
  }
  [2]=>
  array(7) {
    ["id"]=>
    int(3)
    ["title"]=>
    string(8) "product3"
    ["description"]=>
```

<fig 0.2> Output.

Update()

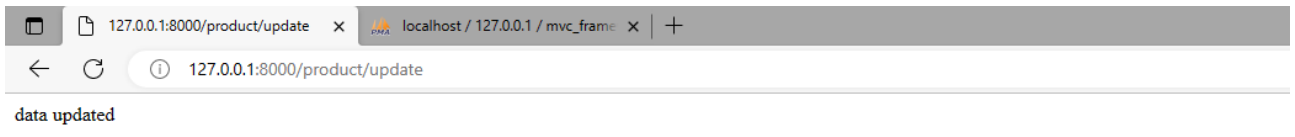
update() is method of **Model**, used to update column data of table. Typically this method is always used with combination of **where()** method (**$obj->where(prop,value)->update()**).

Method update() returns true or error “if any”.



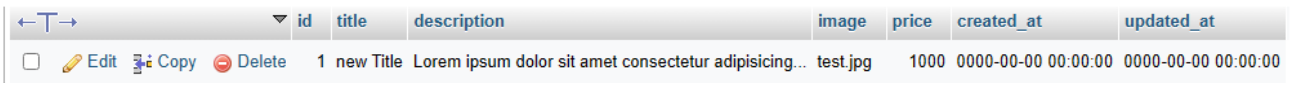
```php
<?php
    namespace App\controllers;

    use App\app\Application;
    use App\app\Controller;
    use App\app\Request;
    use App\models\PRODUCTS;

    class HomeController extends Controller
    {
        //  update product
        public function product(){
            $product = new PRODUCTS();
            $product->title = "new Title";
            if($product->where('id',1)->update()){
                echo("data updated");
                return;
            }
            else{
                echo("error");
                return;
            }

        }

    }


    ?>
```

<fig 0.1> Assign value to property in model of table for which column you want to update and call method **update()** with condition **where()**.


127.0.0.1:8000/product/update
localhost / 127.0.0.1 / mvc_frame
127.0.0.1:8000/product/update

```
data updated
```

<fig 0.2> Output.

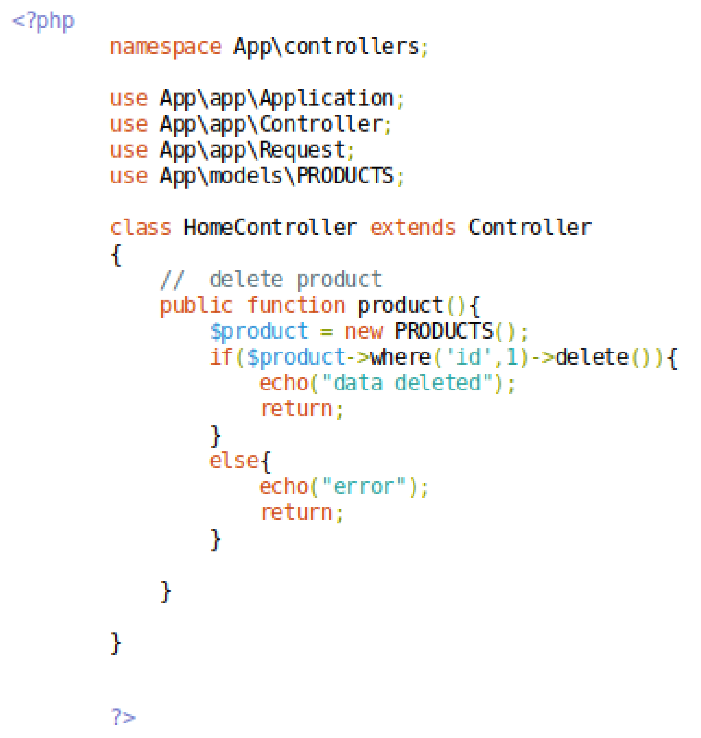


	id	title	description	image	price	created_at	updated_at
Edit Copy Delete	1	new Title	Lorem ipsum dolor sit amet consectetur adipisicing...	test.jpg	1000	0000-00-00 00:00:00	0000-00-00 00:00:00

<fig 0.3> Output in Database.

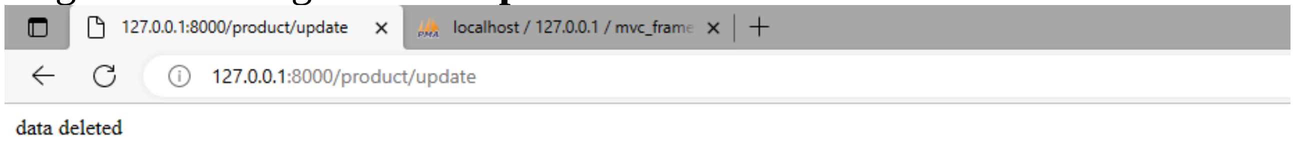
Delete()

delete() is method of **Model**, used to delete column data from table. Typically this method is always used with combination of where() method (**$obj->where(prop,value)->delete()**).
Method delete() returns true or error “if any”.



```
<?php
        namespace App\controllers;

        use App\app\Application;
        use App\app\Controller;
        use App\app\Request;
        use App\models\PRODUCTS;

        class HomeController extends Controller
        {
            //  delete product
            public function product(){
                $product = new PRODUCTS();
                if($product->where('id',1)->delete()){
                    echo("data deleted");
                    return;
                }
                else{
                    echo("error");
                    return;
                }

            }

        }


        ?>
```

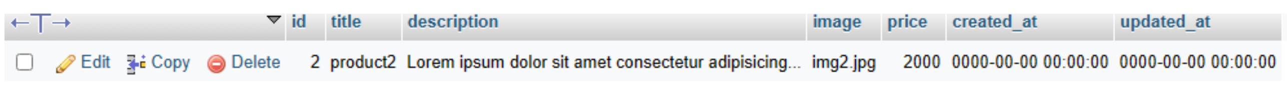
<fig 0.1> deleting data from **products** table **where id=1**.



127.0.0.1:8000/product/update

```
data deleted
```

<fig 0.2> Output.



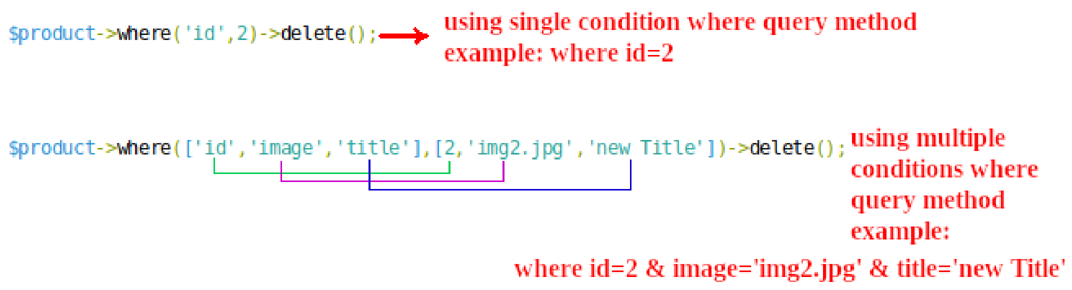
←T→	id	title	description	image	price	created_at	updated_at
Edit Copy Delete	2	product2	Lorem ipsum dolor sit amet consectetur adipisicing...	img2.jpg	2000	0000-00-00 00:00:00	0000-00-00 00:00:00

<fig 0.3> Output in Database column with **id=1** is deleted.

Where()

where() is method of **Model**, used to set condition before running query. Typically this method is always used with combination of **fetch()**, **update()** and **delete()** method (**$obj->where(prop,value)->delete()**, **$obj->where(prop,value)->update()**, **$obj->where(prop,value)->fetch()**).

Where method can have **single** and **multiple** conditions.



```
$product->where('id',2)->delete();
```

using single condition where query method example: where id=2

```
$product->where(['id','image','title'],[2,'img2.jpg','new Title'])->delete();
```

using multiple conditions where query method example:

where id=2 & image='img2.jpg' & title='new Title'

<fig 0.1> Delete method with single and multiple where conditions.

Multiple where condition are declared in two arrays.

Example: (**[prop1, prop2]**,**[val2,val2]**).

1st prop condition will have **1st value**.

You can assign multiple prop same condition value with array of prop and int or string type value.

Example: (**[prop1,prop2]**, **val**).

Middleware

~~**Middleware** is responsible for protecting Routes from un user requests and interacting with model and view components.~~
~~It receives user input and sends output to the user, and also interacts with the database and external services. In Raven controller is represented by a class that extends the base controller class, and it usually contains a set of methods for handling user requests.~~

Creating Middleware Class

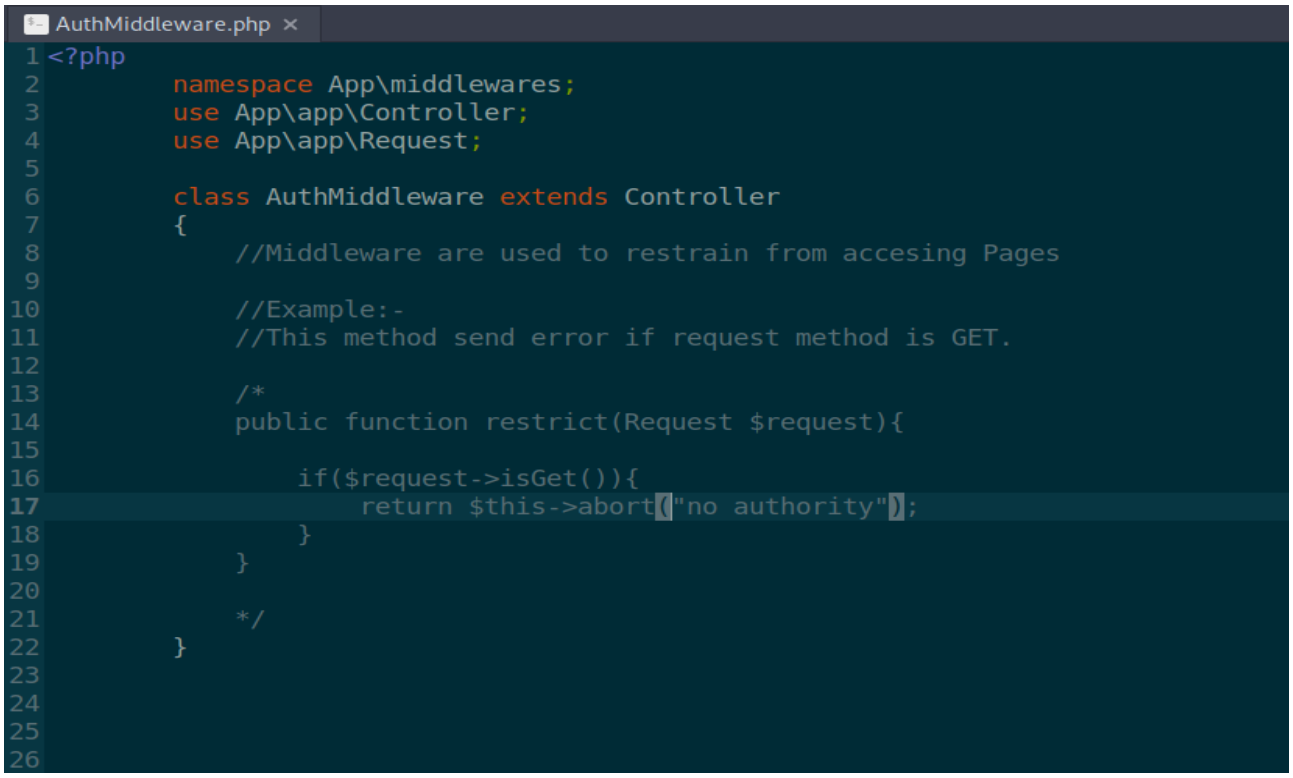
Middlewares are created with help of **raven** (**CLI**) and located in */middlewares* directory.

$php raven --CreateMiddleware <Middlewarename>
$php raven --CreateMiddleware AuthMiddleware

Defining Method

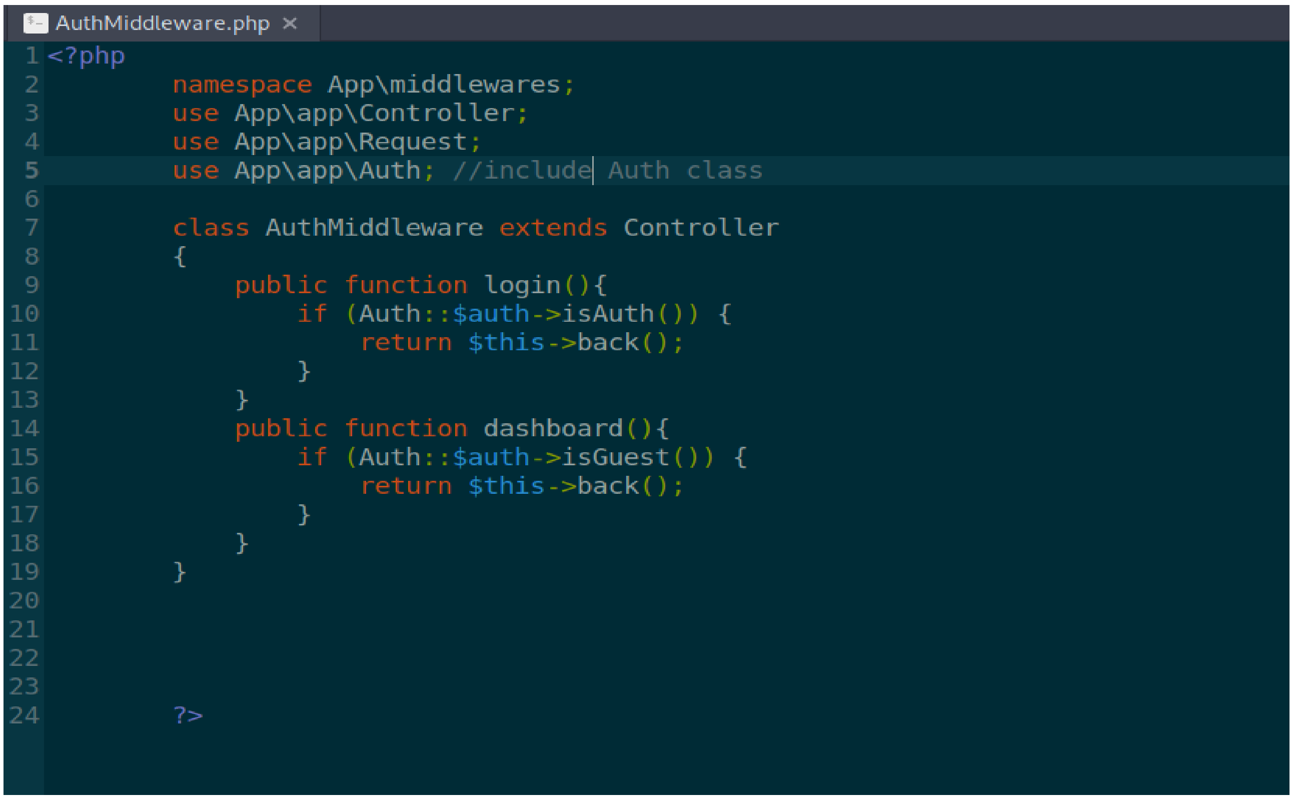
A method is a function that is defined within a Middleware class in **Raven**. It is used to handle specific user routes for restricting and aborting request in specific conditions. For example a Middleware might have a login method that prevent from user after user is logged in, or a dashboard method that protect dashboard page from unauthorized user. In Raven Middleware methods are same as Controller's methods and typically named in a consistent way to make them easy to identify and understand.

Example:



```php
<?php
        namespace App\middlewares;
        use App\app\Controller;
        use App\app\Request;

        class AuthMiddleware extends Controller
        {
            //Middleware are used to restrain from accesing Pages

            //Example:-
            //This method send error if request method is GET.

            /*
            public function restrict(Request $request){

                if($request->isGet()){
                    return $this->abort("no authority");
                }
            }

            */
        }



```

<fig 0.1>We have a new Middleware class named as AuthMiddleware.



```php
<?php
        namespace App\middlewares;
        use App\app\Controller;
        use App\app\Request;
        use App\app\Auth; //include Auth class

        class AuthMiddleware extends Controller
        {
            public function login(){
                if (Auth::$auth->isAuth()) {
                    return $this->back();
                }
            }
            public function dashboard(){
                if (Auth::$auth->isGuest()) {
                    return $this->back();
                }
            }
        }




        ?>
```

<fig 0.2> We have included App\app\Auth Class for checking authentication and created two methods that handles different routes login method return back if user authorized and dashboard method return back if user if guest or unauthorized.

Auth

Auth is Special class mostly used for authentication and in middleware.
Auth is Raven’s built-in simple authentication class uses PHP’s session variable to authenticate users.

Auth Methods

- **isGuest()**

Example: **Auth::$auth->isGuest();**
return True if user is not authenticated.

- **isAuth()**

Example: **Auth::$auth->isAuth();**
return True if user is authenticated.

- **Attempt()**

Example: **Auth::$auth->Attempt();**
return True if authentication is successful else return error.

- **user()**

Example: **Auth::$auth->user();**
return authenticated user data.

- **logout()**

Example: **Auth::$auth->logout();**
return true if user is authenticate and logout.