

# Phalcon



## tutorialspoint

SIMPLY EASY LEARNING

[www.tutorialspoint.com](http://www.tutorialspoint.com)



<https://www.facebook.com/tutorialspointindia>



<https://twitter.com/tutorialspoint>

## About the Tutorial

---

Phalcon is an open source framework quite popular among developers. It is a combination of PHP and C language. Phalcon is developed by Andres Gutierrez and his group of collaborators.

This tutorial provides an overall idea on Phalcon PHP framework and how you can use it.

## Audience

---

This tutorial is basically developed for those who want to learn Phalcon from ground up. The target audience of learning this framework includes students, PHP developers, web designers and web developers.

## Prerequisites

---

Before starting with this tutorial, the user should have knowledge of HTML, CSS and PHP along with an understanding of MVC framework. It would be an added advantage if you have prior exposure to other traditional frameworks like Laravel, Yii, or Codeigniter.

## Copyright and Disclaimer

---

© Copyright 2016 by Tutorials Point (I) Pvt. Ltd.

All the content and graphics published in this e-book are the property of Tutorials Point (I) Pvt. Ltd. The user of this e-book is prohibited to reuse, retain, copy, distribute or republish any contents or a part of contents of this e-book in any manner without written consent of the publisher.

We strive to update the contents of our website and tutorials as timely and as precisely as possible, however, the contents may contain inaccuracies or errors. Tutorials Point (I) Pvt. Ltd. provides no guarantee regarding the accuracy, timeliness or completeness of our website or its contents including this tutorial. If you discover any errors on our website or in this tutorial, please notify us at [contact@tutorialspoint.com](mailto:contact@tutorialspoint.com)

## Table of Contents

---

About the Tutorial .....	i
Audience .....	i
Prerequisites .....	i
Copyright and Disclaimer .....	i
Table of Contents .....	ii
1. PHALCON – OVERVIEW .....	1
2. PHALCON – ENVIRONMENTAL SETUP .....	3
3. PHALCON – APPLICATION STRUCTURE.....	8
4. PHALCON – FUNCTIONALITY .....	12
5. PHALCON – CONFIGURATION .....	15
6. PHALCON – CONTROLLERS.....	20
7. PHALCON – MODELS.....	23
8. PHALCON – VIEWS .....	29
9. PHALCON – ROUTING .....	33
10. PHALCON – DATABASE CONNECTIVITY .....	35
11. PHALCON – SWITCHING DATABASES.....	37
12. PHALCON – SCAFFOLDING APPLICATION .....	39
Designing the Login Page.....	46
Creating Views .....	49
Category Management.....	51

13.	PHALCON – QUERY LANGUAGE.....	64
	Creating a PHQL Query .....	64
	PHQL Life Cycle.....	66
14.	PHALCON – DATABASE MIGRATION .....	67
	Understanding the Anatomy of Migrated Files .....	69
15.	PHALCON – COOKIE MANAGEMENT .....	72
16.	PHALCON – SESSION MANAGEMENT .....	76
17.	PHALCON – MULTI-LINGUAL SUPPORT .....	79
18.	PHALCON – ASSET MANAGEMENT .....	82
19.	PHALCON – WORKING WITH FORMS .....	86
20.	PHALCON – OBJECT DOCUMENT MAPPER .....	89
21.	PHALCON – SECURITY FEATURES .....	94
	Hashing Password .....	94
	Cross-Site Request Forgery (CSRF) .....	97

# 1. Phalcon – Overview

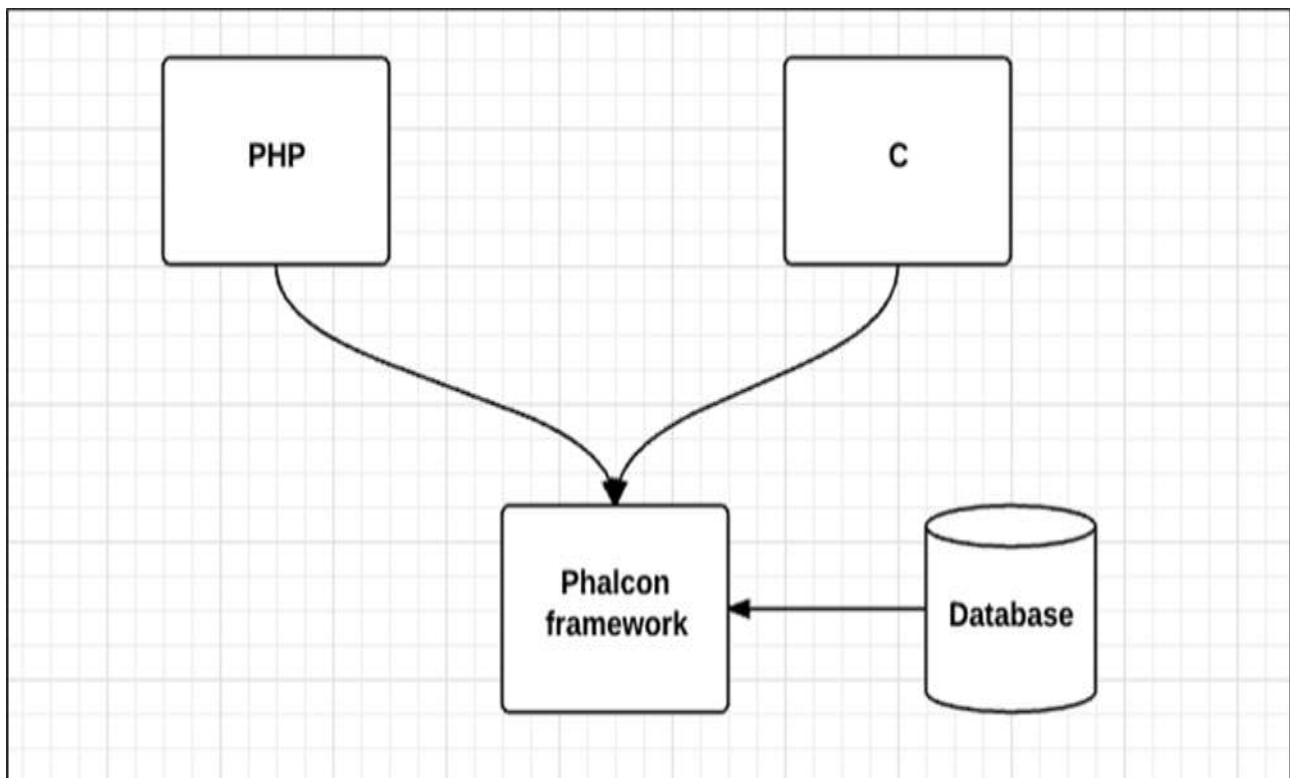
Phalcon is introduced as one of the recent frameworks of PHP, developed by a group of enthusiastic developers. Phalcon is a loosely coupled framework, which means it allows its objects to be treated like glue components, based on the needs of application.

Phalcon offers some unique features in comparison to other frameworks (traditional or existing) in PHP. Following are some of the most prominent features of Phalcon:

- It is a full stack open source framework.
- A user needs quite less amount of code to take advantage of several components.
- It can be used to create an independent framework as required. For example, if we just need Phalcon's Cache component, we can use it in any application written either in pure PHP or using a framework.
- For a developer having knowledge of **Model-View-Controller** (MVC) and **Object-Relational Modeling** (ORM), working with Phalcon is like a cakewalk.

## Performance

The performance of Phalcon is a feature which distinguishes it from other traditional frameworks of PHP. Phalcon has a combination of both PHP and C; each of them can be used as a stand-alone module. The compilation of every request is considered on a higher speed in Phalcon which makes everything seem out-of-the-box.



## C Language

Phalcon is compatible with C which increases the compilation rate. Also, C in combination with Phalcon provides Object Relational Mapping (ORM) which provides consistency with models created. Every model created in Phalcon is associated with the table of relational database. ORM in Phalcon is purely implemented in C.

## Developer Tools

Developer tools are used for developing web application. These tools help in generating scaffold application with a combination of all features (C – Create, R – Read, U – Update, D – Delete). Developer tools also include extensible support for third party libraries to be implemented in Phalcon.

## Object Relational Mapping

Phalcon supports a wide range of databases. It is not limited to access of relational databases. It supports both relational and non-relational databases which is like adding a feather to the cap of Phalcon framework.

## Phalcon Vs Other Frameworks

The following table highlights how Phalcon differs from other popular frameworks such as Yii and Laravel.

	Yii	Laravel	Phalcon
<b>Type of Projects</b>	Yii helps in creating large scale projects like forums, portals, CMS, RESTful web services, etc.	Laravel is used for building web applications. It is known for exquisite and sophisticated syntax.	Phalcon is used to design variety of projects.
<b>Database Support</b>	Yii supports all relational and non-relational databases.	Laravel supports all relational databases.	Phalcon gives equal support to relational and non-relational databases.
<b>Language</b>	Yii framework is purely written in PHP.	Laravel is written in PHP and follows MVC pattern.	Phalcon includes both PHP and C.
<b>Scalability</b>	Yii is quite scalable for small and medium projects.	Scalability is high for Laravel with all kinds of projects.	Good for medium projects.
<b>Performance</b>	Comparatively low.	High but less in comparison with Phalcon.	High performance.

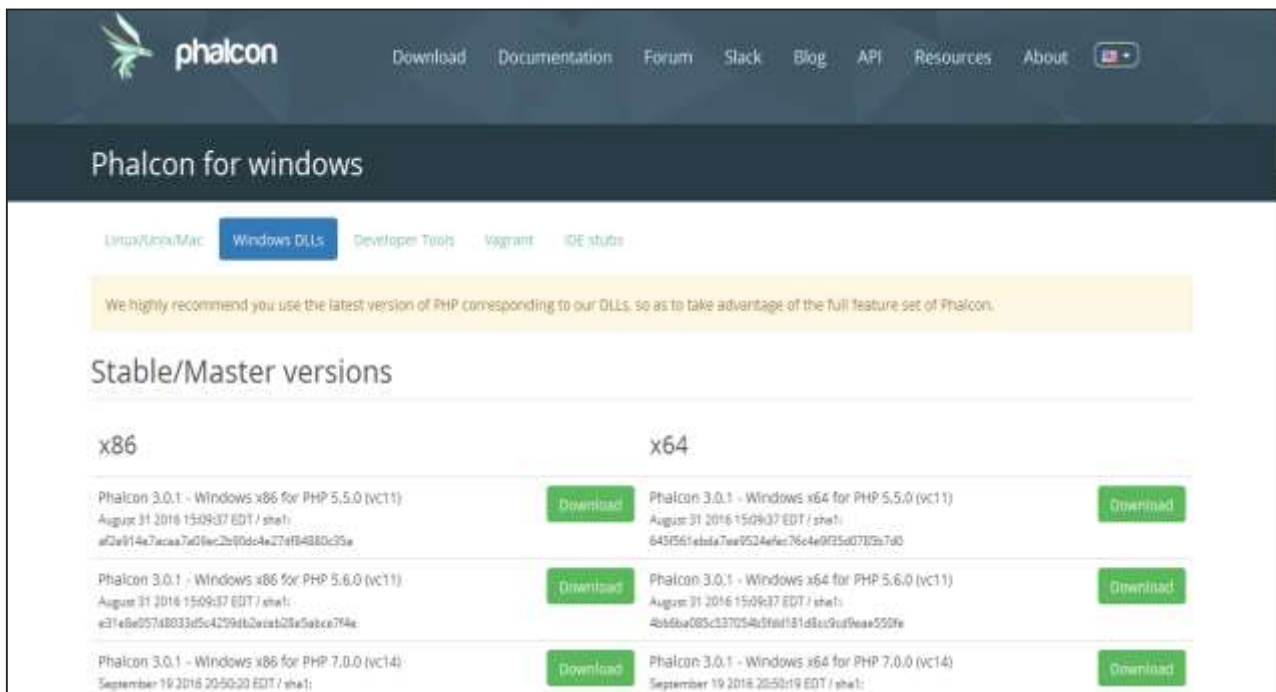
## 2. Phalcon – Environmental Setup

**Prerequisites:** We need WAMP/LAMP/MAMP or XAMPP stack for this framework.

Following are the steps for the installation process of Phalcon framework in Windows.

**Step 1:** Phalcon installation is completely dependent on **dll** file. DLL (Dynamic Link Library) creates the required package and plugins for Phalcon.

The following link is used for downloading dll file:  
<https://phalconphp.com/en/download>



**Step 2:** Download the required dll file. Check for the appropriate configuration of the system and download the required dll file. After downloading the file, extract **phalcon-php.dll** to **/php/ext** in the **xampp** folder.

**Step 3:** Edit the path in **php.ini** file to get it configured in a similar manner of other **.dll** files.



```

863 extension=php_gd2.dll
864 extension=php_gettext.dll
865 ;extension=php_gmp.dll
866 ;extension=php_intl.dll
867 ;extension=php_imap.dll
868 ;extension=php_interbase.dll
869 ;extension=php_ldap.dll
870 extension=php_mbstring.dll
871 extension=php_phalcon.dll
872 extension=php_exif.dll      ; Must be after mbstring as it depends on it
873 extension=php_mysql.dll
874 extension=php_mysqli.dll
875 ;extension=php_oci8.dll      ; Use with Oracle 10gR2 Instant Client
876 ;extension=php_oci8_11g.dll ; Use with Oracle 11gR2 Instant Client
877 ;extension=php_openssl.dll
878 ;extension=php_pdo_firebird.dll
879 extension=php_pdo_mysql.dll
880 ;extension=php_pdo_oci.dll
881 ;extension=php_pdo_odbc.dll
882 ;extension=php_pdo_pgsql.dll
883 extension=php_pdo_sqlite.dll
884 ;extension=php_pgsql.dll
885 ;extension=php_pspell.dll
886 ;extension=php_shmop.dll
887
888
889 ; The MIBS data available in the PHP distribution must be installed.
890 ; See http://www.php.net/manual/en/snmp.installation.php
891 ;extension=php_snmp.dll
892

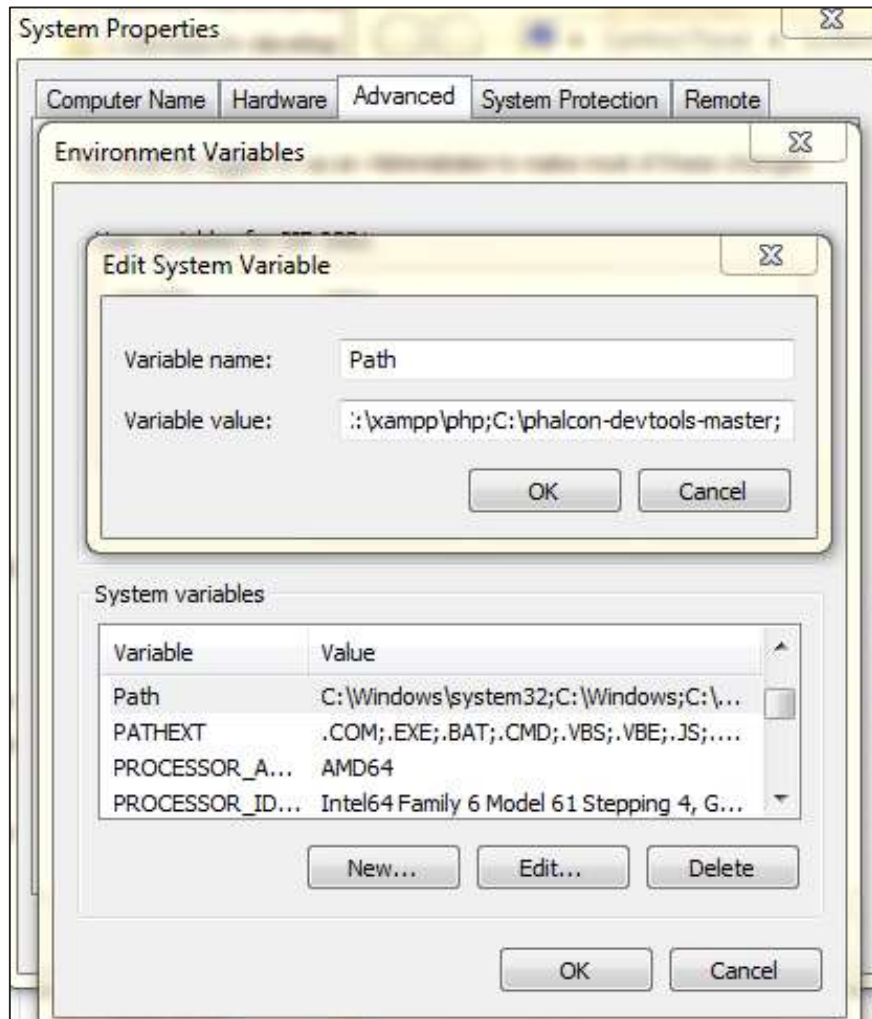
```

**Step 4:** Once the path is edited, restart the **xampp/wamp** stack. It will be clearly visible in the dashboard, once the **dll** file is properly set.

phalcon		
Web framework delivered as a C-extension for PHP		
phalcon	enabled	
Author	Phalcon Team and contributors	
Version	3.0.1	
Build Date	Aug 24 2016 11:24:53	
Powered by Zephir	Version 0.9.4a-dev-7e304ba18c	
Directive	Local Value	Master Value
phalcon.db.escape_identifiers	On	On
phalcon.db.force_casting	Off	Off
phalcon.orm.cast_on_hydrate	Off	Off
phalcon.orm.column_renaming	On	On
phalcon.orm.enable_implicit_joins	On	On
phalcon.orm.enable_literals	On	On
phalcon.orm.events	On	On
phalcon.orm.exception_on_failed_save	Off	Off
phalcon.orm.ignore_unknown_columns	Off	Off
phalcon.orm.late_state_binding	Off	Off
phalcon.orm.not_null_validations	On	On
phalcon.orm.virtual_foreign_keys	On	On



**Step 5:** After downloading the package, set the path variable in the system properties.



**Step 6:** The **dll** files and Phalcon tools together help in creating the project/web application. The user can verify through command prompt whether Phalcon framework has been successfully installed. The output will be displayed as shown in the following screenshot.

```

C:\Users\SIF 2551>phalcon

Phalcon DevTools <3.0.1>

Available commands:
  commands      <alias of: list, enumerate>
  controller     <alias of: create-controller>
  module         <alias of: create-module>
  model          <alias of: create-model>
  all-models     <alias of: create-all-models>
  project        <alias of: create-project>
  scaffold       <alias of: create-scaffold>
  migration      <alias of: create-migration>
  webtools       <alias of: create-webtools>

C:\Users\SIF 2551>

```

**Step 7:** Once this necessary output is received, create a project using the following command:

```
phalcon create-project <project-name>
```

The following output will be displayed.

```

C:\xampp\htdocs>phalcon

Phalcon DevTools <3.0.1>

Available commands:
  commands      <alias of: list, enumerate>
  controller     <alias of: create-controller>
  module         <alias of: create-module>
  model          <alias of: create-model>
  all-models     <alias of: create-all-models>
  project        <alias of: create-project>
  scaffold       <alias of: create-scaffold>
  migration      <alias of: create-migration>
  webtools       <alias of: create-webtools>

C:\xampp\htdocs>phalcon create-project demo1

Phalcon DevTools <3.0.1>

  Success: Controller "index" was successfully created.
C:\xampp\htdocs\demo1\app\controllers\IndexController.php
  Success: Project "demo1" was successfully created.

C:\xampp\htdocs>_

```

**Step 8:** The web application is successfully created. Click the following URL:

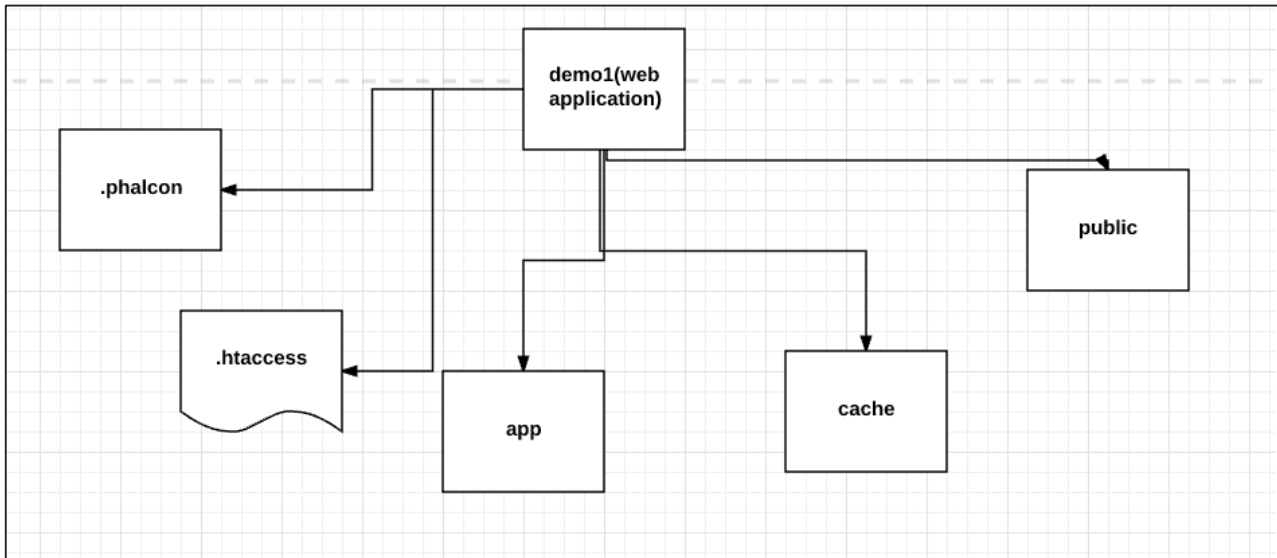
<http://localhost/demo1>

The output will be displayed as shown in the following screenshot. It is the welcome page for Phalcon PHP.



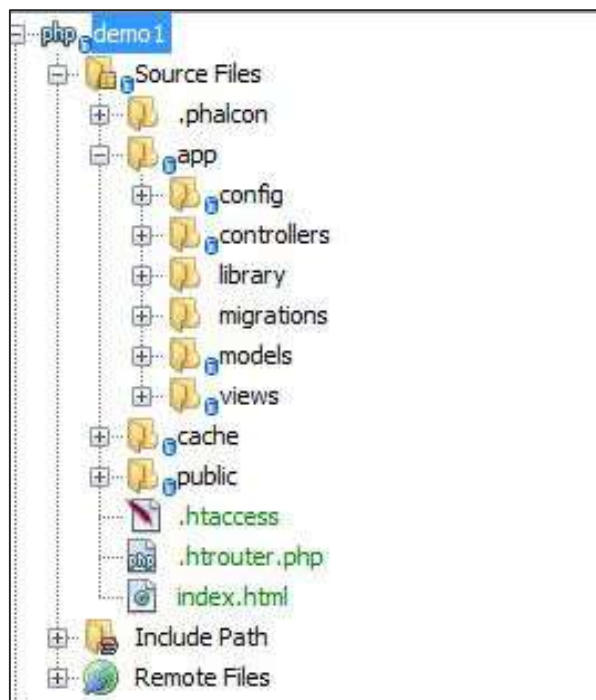
### 3. Phalcon – Application Structure

In this chapter, we will discuss the Application Structure of Phalcon. Following is the complete directory structure of a Phalcon project.



There is one root folder which is considered as the **code base** and is publicly available for the web server. It is also called as **web directory**. Other folders outside the web root directory are considered out of reach for the web server and for Phalcon project.

Once a project is created, the directory structure will be visible as follows in the **wamp/xampp** folder. Consider for the project which we created in the previous chapter.

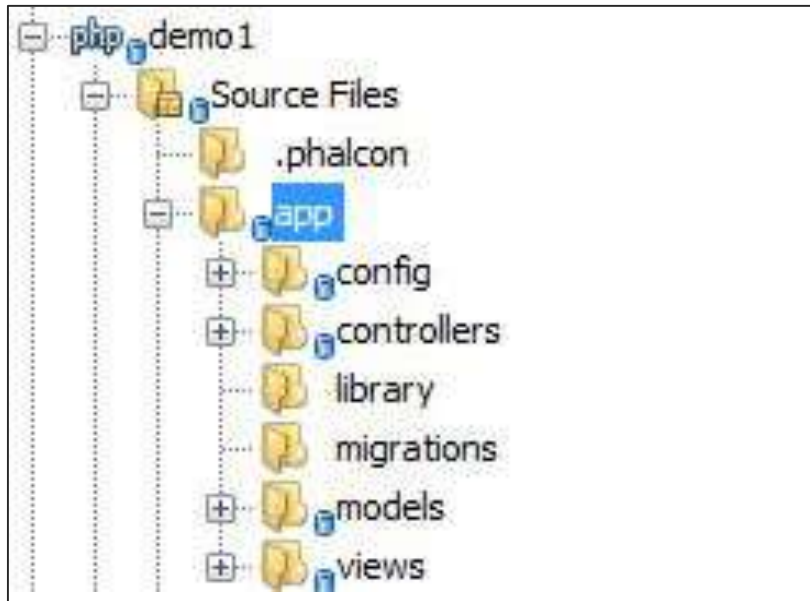


Following are the folders and sub-folders of the project.

## App

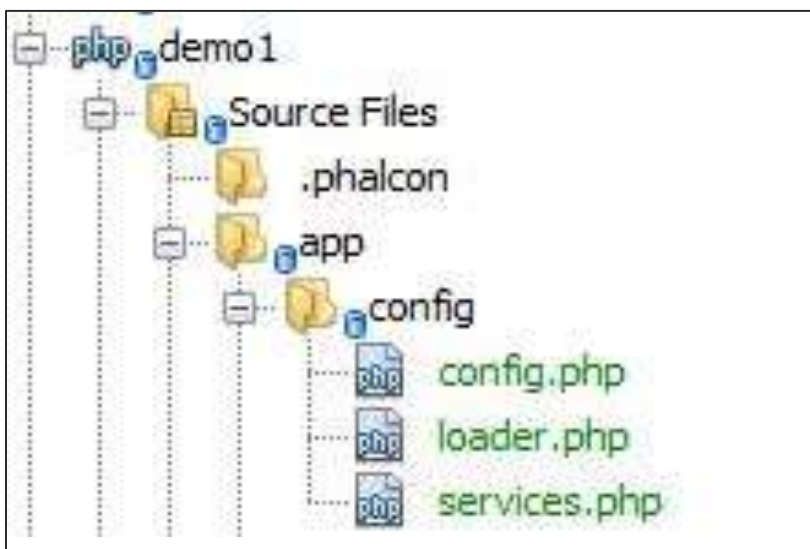
This folder consists of all vital script files and folders. The complete web application is designed on the basis of "app" folder. The configuration files help in assisting the necessary configuration for running the application smoothly.

Following is the detailed view of app folder for the given Phalcon web application.



It consists of config, controllers, library, migrations, models and views.

## Config



All the configuration required for the web application in Phalcon is comprised in this folder. It includes information related to database connectivity, third-party libraries to be added if any, and the services to be included.

## Controllers

All the controllers are included in this folder. They are used for processing requests and generating response.

## Library

Third-party libraries for the web application (apart from the existing Phalcon framework).



## Migrations

This sub-folder consists of all the files associated with data migration, which can also be used in any other framework.

## Models

Models include all the logic required to interact with the database. It is actually used for data representation.

## Views

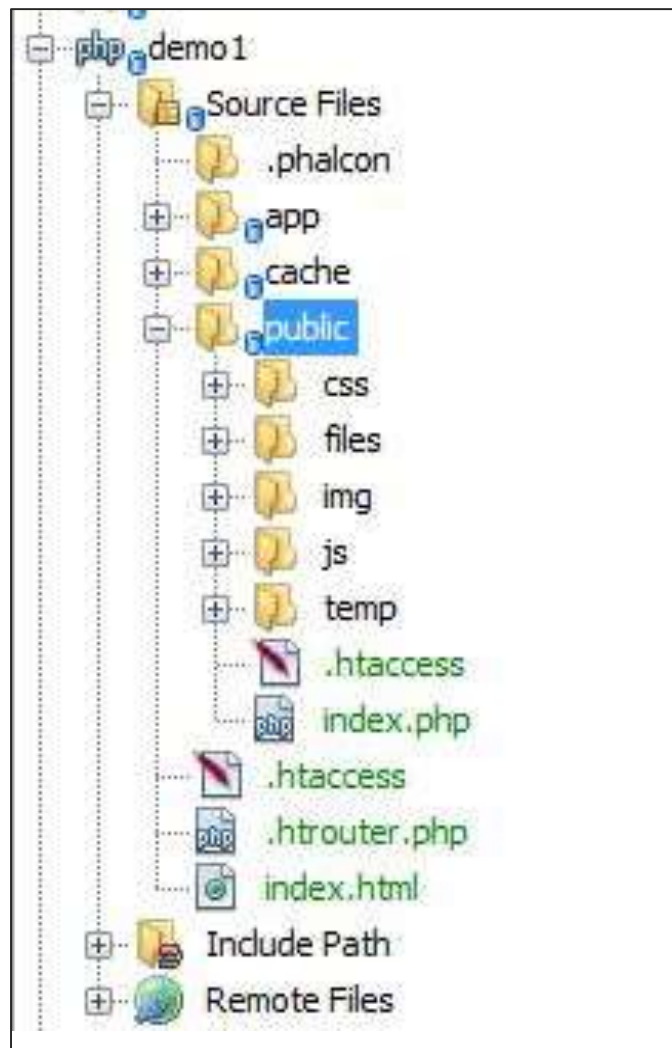
It constitutes all the views related to the web application. These views are displayed to the end users with the help of controllers.

## Cache

This directory includes data related to caching, which helps in improving the performance.

## Public

It includes all the folders for asset management purpose which comprises of CSS, JavaScript, files to be uploaded, and some meta data.



## .htaccess File

Web servers running on Apache Web Server software use **.htaccess** as a configuration file. When it is placed in a directory, all the necessary configuration is loaded as soon as the server is started.

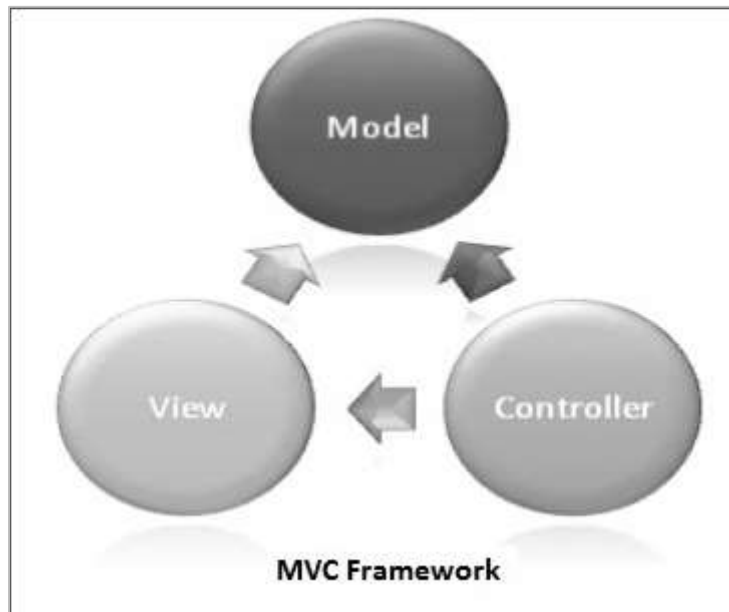
For example, it is possible to configure a website so that it will be available only to specific IP addresses with **.htaccess** file.



## 4. Phalcon – Functionality

**Model View Controller (MVC)** is a software design and structural pattern for developing web-based applications. This software architectural separates the representation of information from the user's interaction with it.

The MVC model defines the web applications with three logic layers.



### Model

Models are objects which represent knowledge. There should be a one-to-one relationship between the model and its parts. It includes all the logic to be used for database connectivity and performing CRUD operations.

### View

A view is a visual representation of its model. View interacts with the model or its parts and gets the data necessary for the presentation from the model. This is achieved by sending requests and receiving appropriate responses. View includes all the data that end user sees.

### Controller

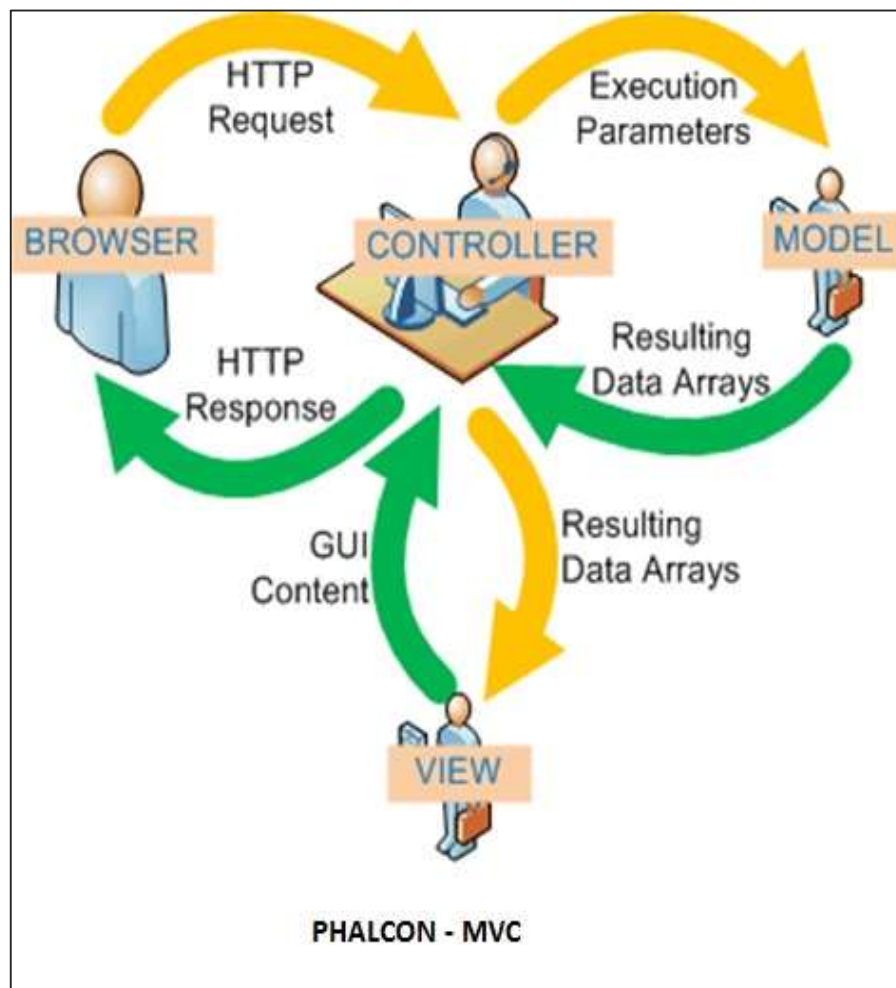
A controller acts as the intermediary between the user and the system (model and view). It accepts the request from the user, through the view sends it to the model. The model manipulates it and sends the response to the controller, which is displayed as the output to the end user through view.

The controller receives such user output and translates it into the appropriate messages. These messages are used by view to display as appropriate responses.

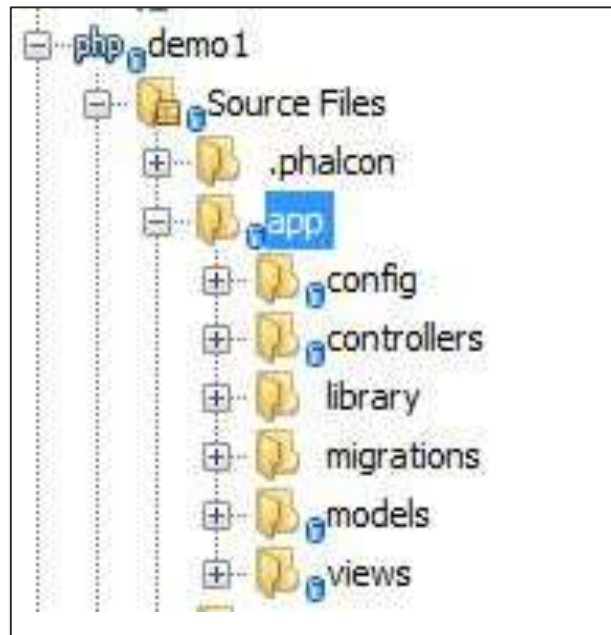
## Workflow in Phalcon

The workflow in Phalcon is as follows:

- The user interacts with the user interface (view) and the interaction is maintained with the help of some method/event.
- These methods and events are handled by the controller.
- The controller accesses the model by updating the user's action.
- View uses the model to generate an appropriate output.
- View fetches data from its model. The model has no direct interaction with view.
- The user interface waits for further user interactions, which starts with a new cycle of request and response.



Phalcon includes directories for Model, View, and Controller. The following screenshot gives a better scenario.



All business logic is described in the controller, and the model interacts with the database which includes all files with respect to each and every table.

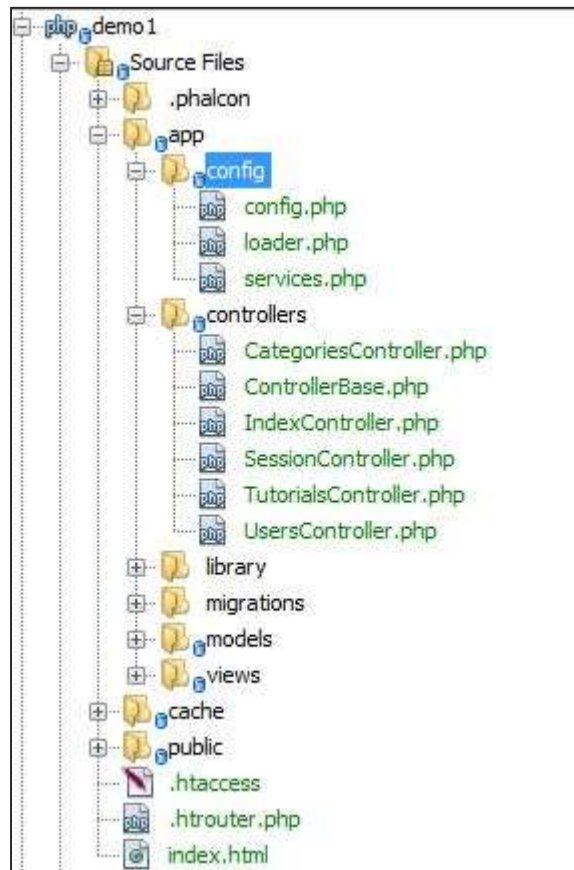
**Note:**

- All the controllers created in Phalcon web application extends **Phalcon\Mvc\Controller**.
- All the models associated with the database tables extends **\Phalcon\Mvc\Model**.

## 5. Phalcon – Configuration

The config folder of the web application includes the following files:

- config.php
- loader.php
- services.php



### config.php

It includes the configurations for database connectivity and routing as per the directory path.

```
<?php
/*
 * Modified: preppend directory path of current file, because of this file own
 different ENV under between Apache and command line.
 * NOTE: please remove this comment.
 */
defined('BASE_PATH') || define('BASE_PATH', getenv('BASE_PATH') ?:
realpath(dirname(__FILE__) . '/../..'));
```

```

defined('APP_PATH') || define('APP_PATH', BASE_PATH . '/app');

return new \Phalcon\Config([
    'database' => [
        'adapter'      => 'Mysql',
        'host'         => 'localhost',
        'username'     => 'root',
        'password'     => '',
        'dbname'       => 'test',
        'charset'      => 'utf8',
    ],
    'application' => [
        'appDir'        => APP_PATH . '/',
        'controllersDir' => APP_PATH . '/controllers/',
        'modelsDir'     => APP_PATH . '/models/',
        'migrationsDir' => APP_PATH . '/migrations/',
        'viewsDir'      => APP_PATH . '/views/',
        'pluginsDir'    => APP_PATH . '/plugins/',
        'libraryDir'    => APP_PATH . '/library/',
        'cacheDir'      => BASE_PATH . '/cache/',
        'baseUri'       => '/demo1/',
    ]
]);

```

## loader.php

It extends the existing class of **\Phalcon\Loader()**. The loader class registers the directories which requires web application.

```

<?php

$loader = new \Phalcon\Loader();

/**
 * We're a registering a set of directories taken from the configuration file
 */
$loader->registerDirs(

```

```

[
    $config->application->controllersDir,
    $config->application->modelsDir
]
)->register();

```

## services.php

This file associates all the functions which implement the services of a web project. It implements **Phalcon\Di** interface. It also implements a dependency injection of the services by loading them.

Basically, services.php file inside the config folder acts as a container of all services. This interface helps in initializing all the services like database connection, setting up cookies, creating a new session, or connecting with NoSQL database.

```

?php

use Phalcon\Mvc\View;
use Phalcon\Mvc\View\Engine\Php as PhpEngine;
use Phalcon\Mvc\Url as UrlResolver;
use Phalcon\Mvc\View\Engine\Volt as VoltEngine;
use Phalcon\Mvc\Model\MetaData\Memory as MetaDataAdapter;
use Phalcon\Session\Adapter\Files as SessionAdapter;
use Phalcon\Flash\Direct as Flash;

/**
 * Shared configuration service
 */
$di->setShared('config', function () {
    return include APP_PATH . "/config/config.php";
});

/**
 * The URL component is used to generate all kind of urls in the application
 */
$di->setShared('url', function () {
    $config = $this->getConfig();

```

```

        $url = new UrlResolver();

        $url->setBaseUri($config->application->baseUri);

        return $url;
    });

    /**
     * Setting up the view component
     */
    $di->setShared('view', function () {
        $config = $this->getConfig();

        $view = new View();
        $view->setDI($this);
        $view->setViewsDir($config->application->viewsDir);

        $view->registerEngines([
            '.volt' => function ($view) {
                $config = $this->getConfig();

                $volt = new VoltEngine($view, $this);

                $volt->setOptions([
                    'compiledPath' => $config->application->cacheDir,
                    'compiledSeparator' => '_'
                ]);

                return $volt;
            },
            '.phtml' => PhpEngine::class
        ]);

        return $view;
    });

```



```
/**
 * Database connection is created based in the parameters defined in the
 * configuration file
 */
$di->setShared('db', function () {
    $config = $this->getConfig();

    $class = 'Phalcon\Db\Adapter\Pdo\' . $config->database->adapter;
    $connection = new $class([
        'host'      => $config->database->host,
        'username' => $config->database->username,
        'password' => $config->database->password,
        'dbname'   => $config->database->dbname,
        'charset'  => $config->database->charset
    ]);

    return $connection;
});
```

## 6. Phalcon – Controllers

In MVC framework, “C” stands for the Controller which refers to the switchboards of the web application. The actions undertaken by the controller, helps to pass parameters to the view so that it can display and respond to the user input accordingly.

For example, if we register through a sign-up form which includes details of the user such as username, email address and password, and click the Submit button, the data inserted or posted by the user is sent through the controller with the help of associated action or function.

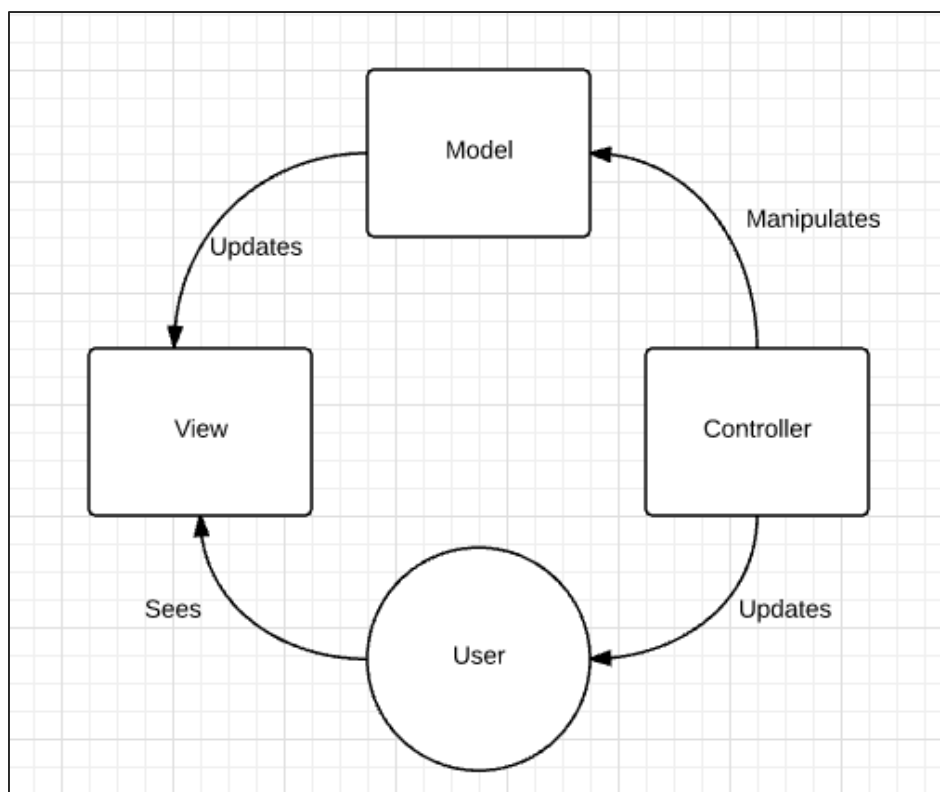
### Features of a Controller

A **controller** accepts inputs from the view and interacts with the associated model.

- It helps in updating the model's state by sending commands to the model. It can also send commands to the associated view, which helps in changing the presentation of the view as per the model's state.
- A controller acts as an intermediary between the model and the view.

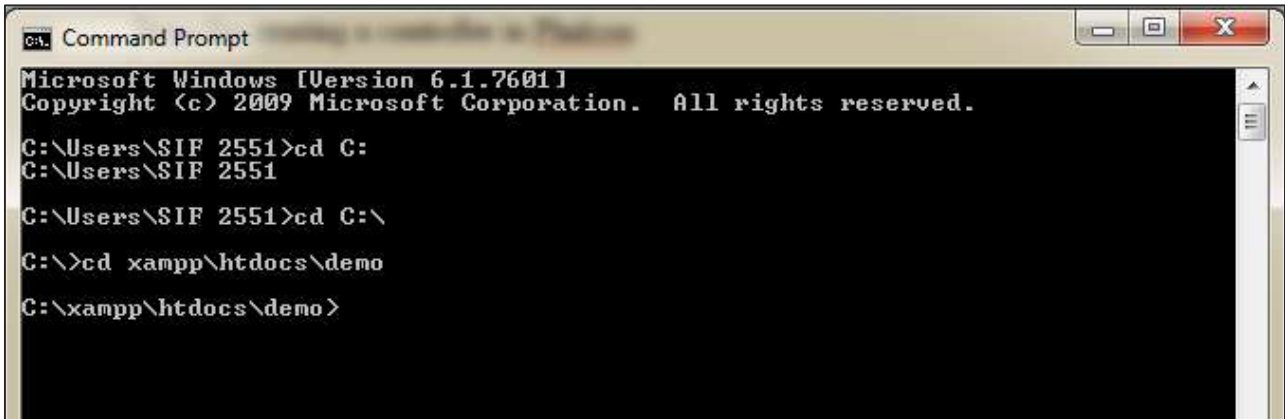
### Workflow of a MVC in Phalcon

The following illustration shows the workflow of MVC in Phalcon:



## Steps to Create a Controller in Phalcon

**Step 1:** Redirect to the project path with the help of command prompt. Refer to the following screenshot.



```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\SIF 2551>cd C:
C:\Users\SIF 2551>

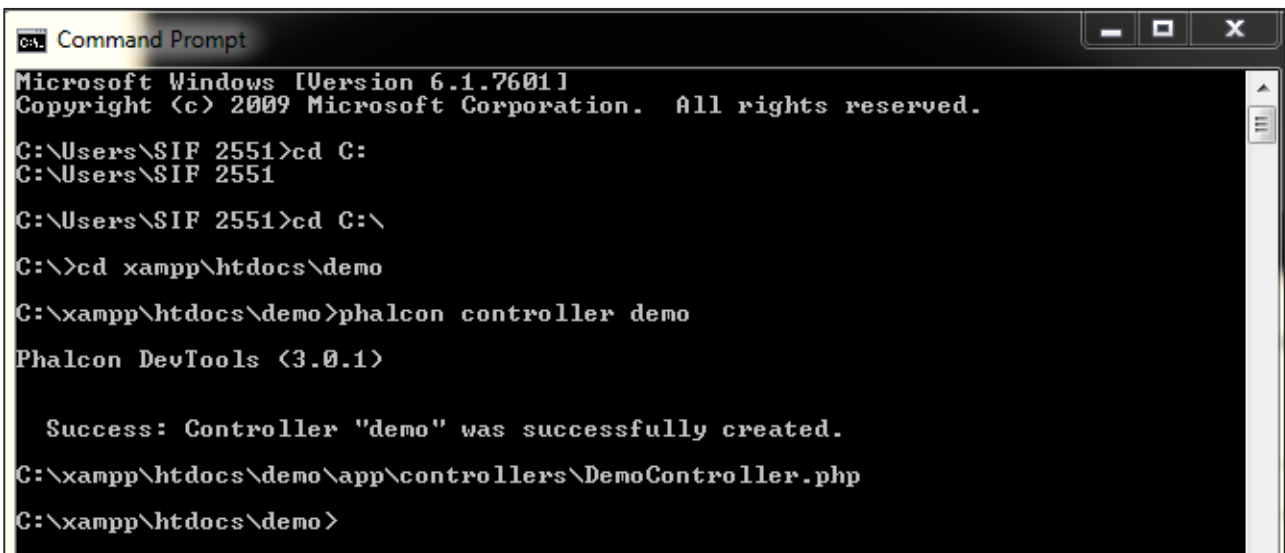
C:\Users\SIF 2551>cd C:\
C:\>cd xampp\htdocs\demo
C:\xampp\htdocs\demo>
```

As referred in the above screenshot, "demo" is the project associated with Phalcon PHP framework.

**Step 2:** Use the following command to create an associated controller.

```
phalcon controller <controller-name>
```

Following is the output on successful execution of the above command.



```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\SIF 2551>cd C:
C:\Users\SIF 2551>

C:\Users\SIF 2551>cd C:\
C:\>cd xampp\htdocs\demo
C:\xampp\htdocs\demo>phalcon controller demo
Phalcon DevTools (3.0.1)

    Success: Controller "demo" was successfully created.
C:\xampp\htdocs\demo\app\controllers\DemoController.php
C:\xampp\htdocs\demo>
```

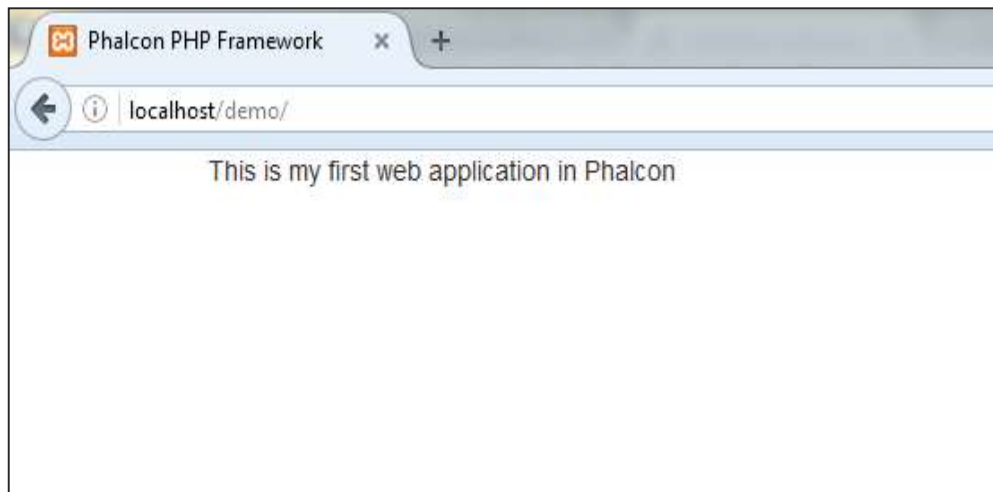
**Note:** The class names of the controllers must have the suffix "controller". This implies a good naming convention which is followed in Phalcon.

By default, when the application is created in Phalcon PHP framework, it includes a controller named "IndexController". This controller is invoked by default to trigger the actions.

This controller is extended by controller-base unlike other controllers which extends **\Phalcon\Mvc\Controller**.

**Code:**

```
<?php
class IndexController extends ControllerBase
{
    public function indexAction()
    {
        echo "This is my first web application in Phalcon";
    }
}
```

**Output:**

## 7. Phalcon – Models

Model in MVC architecture includes the logic of application. Model is the core interaction with the database. It should be able to manage updating, deleting, inserting, and fetching of records as per the user's request.

For understanding the model interaction in Phalcon PHP framework, following steps should be followed.

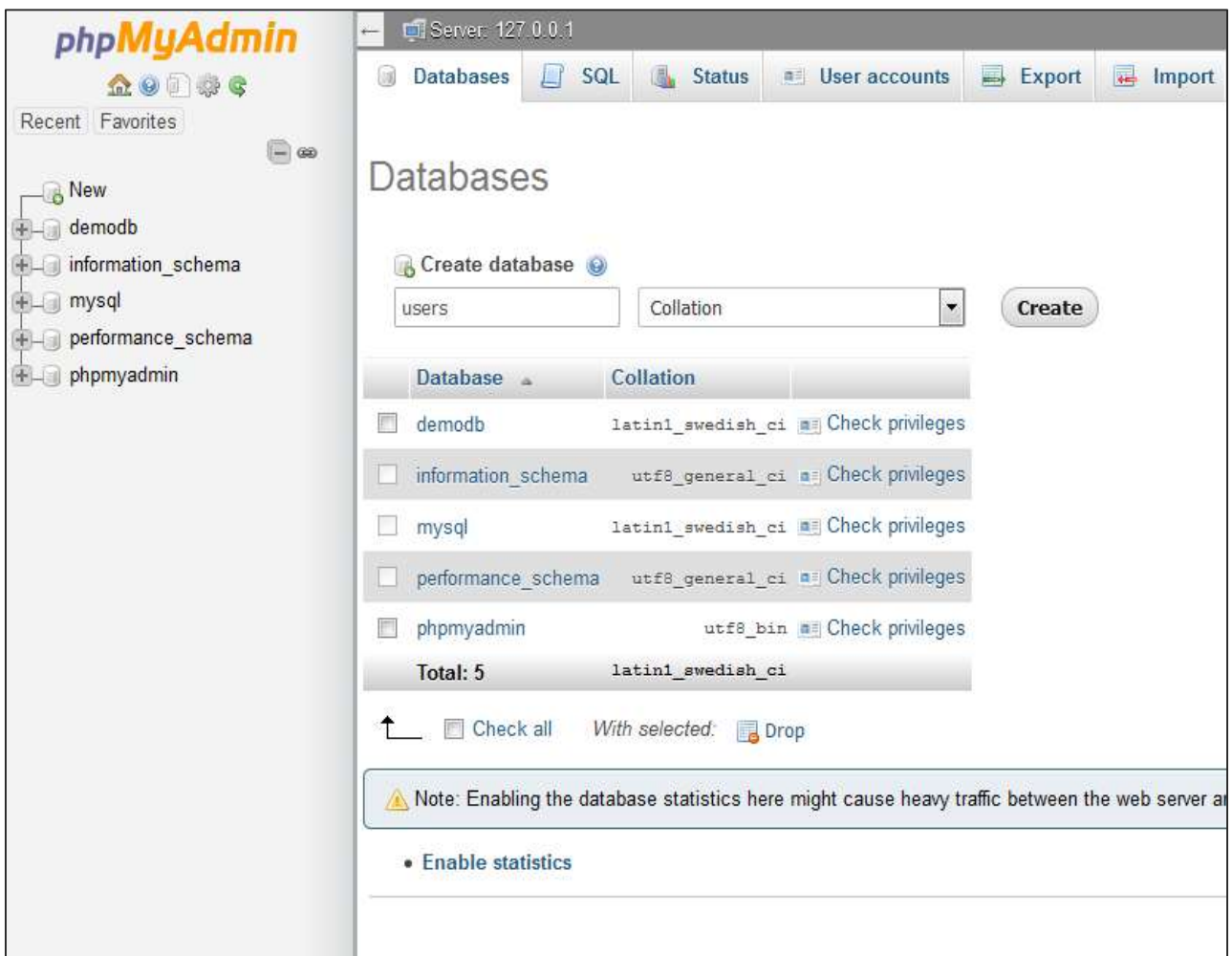
**Step 1:** Creation of database.

For any **LAMP, WAMP, XAMPP** software stack, it is quite easy to create a database with the help of **phpmyadmin** database tool.

Following is the SQL query to create a database.

```
create database <database-name>
```

**Step 2:** In the **phpmyadmin** section, click the Databases tab, mention the database name and further click the Create button as shown in the following screenshot.



**Step 3:** Once the database is created successfully, create a table which will help its association for creating a model in Phalcon framework.

Use the following query to create a new table named "users".

```
DROP TABLE IF EXISTS `users` ;

CREATE TABLE `users` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `name` varchar(25),
  `emailid` varchar(50),
  `contactNumber` number
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

Once the table is created, its structure looks like as shown in the following screenshot.

#	Name	Type	Collation	Attributes	Null	Default	Extra	Action
1	id	int(11)			No	None	AUTO_INCREMENT	Change Drop Primary Unique Index Spatial Fulltext More
2	name	varchar(25)			Yes	NULL		Change Drop Primary Unique Index Spatial Fulltext More
3	emailid	varchar(50)			Yes	NULL		Change Drop Primary Unique Index Spatial Fulltext More
4	contactNumber	int(11)			Yes	NULL		Change Drop Primary Unique Index Spatial Fulltext More

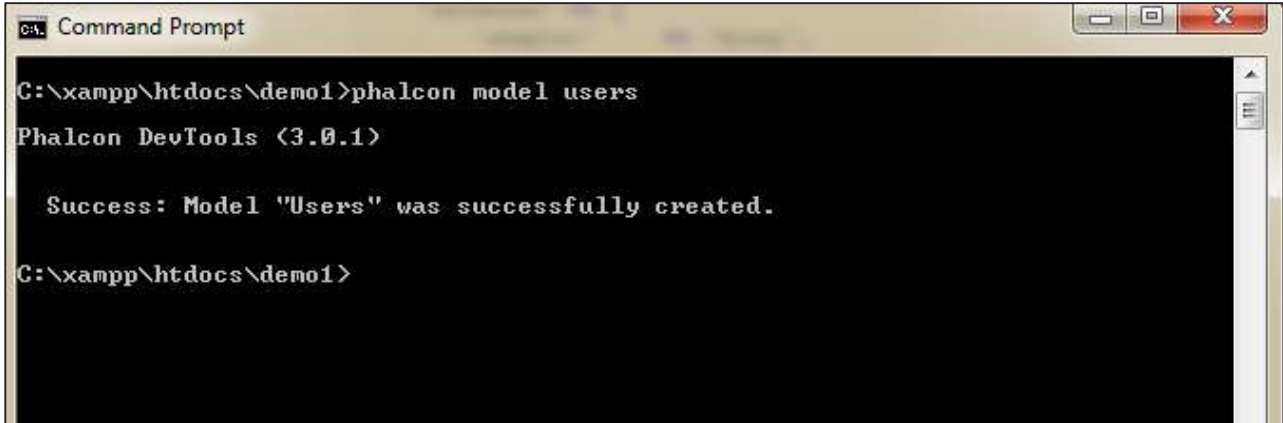
**Step 4:** To create a model associated with the 'Users' table which we created in the above step, open command prompt instance. It is important to redirect to the appropriate project path. Before that, it is vital to check whether the database configuration has been correctly set as shown in the following screenshot.

```
return new \Phalcon\Config([
    'database' => [
        'adapter'      => 'Mysql',
        'host'         => 'localhost',
        'username'     => 'root',
        'password'     => '',
        'dbname'       => 'users',
        'charset'      => 'utf8',
    ],
]);
```

**Step 5:** Use the following command to create any model in Phalcon framework.

```
phalcon model <model-name>
```

Following is the output on execution of the above command.



```
Command Prompt
C:\xampp\htdocs\demo1>phalcon model users
Phalcon DevTools <3.0.1>

Success: Model "Users" was successfully created.

C:\xampp\htdocs\demo1>
```

This implies that the model has been created successfully.

**Step 6:** Model created successfully is present in the models folder. Use the following path to view where the model is created.

```
C:\xampp\htdocs\demo1\app\models
```

Following is the complete code for **Users.php**.

```
<?php

class Users extends \Phalcon\Mvc\Model
{
    /**
     *
     * @var integer
     * @Primary
     * @Identity
     * @Column(type="integer", length=11, nullable=false)
     */

    public $id;

    /**
     *
     * @var string
     * @Column(type="string", length=25, nullable=true)
     */
}
```



```
public $name;

/**
 *
 * @var string
 * @Column(type="string", length=50, nullable=true)
 */

public $emailid;

/**
 *
 * @var integer
 * @Column(type="integer", length=11, nullable=true)
 */

public $contactNumber;

/**
 * Returns table name mapped in the model.
 *
 * @return string
 */

public function getSource()
{
    return 'users';
}

/**
 * Allows to query a set of records that match the specified conditions
 *
 * @param mixed $parameters
 * @return Users[]
 */

public static function find($parameters = null)
{
    return parent::find($parameters);
}

/**
 * Allows to query the first record that match the specified conditions
 *
 * @param mixed $parameters
 * @return Users
 */
```

```

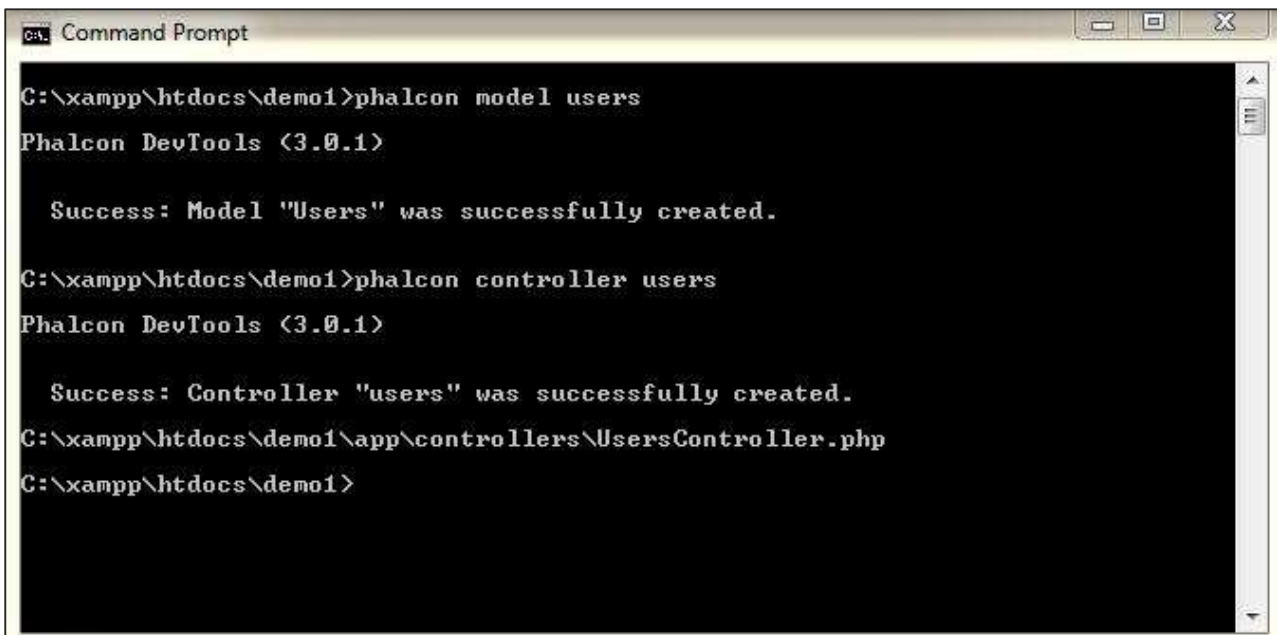
public static function findFirst($parameters = null)
{
    return parent::findFirst($parameters);
}
}

```

**Step 7:** The Controller interacts with the model and the view to get the necessary output. As with the model, use the following command terminal to create a controller.

```
Phalcon controller <controller-name>
```

On successful execution of the above command, following is the output.



```

C:\xampp\htdocs\demo1>phalcon model users
Phalcon DevTools <3.0.1>

Success: Model "Users" was successfully created.

C:\xampp\htdocs\demo1>phalcon controller users
Phalcon DevTools <3.0.1>

Success: Controller "users" was successfully created.
C:\xampp\htdocs\demo1\app\controllers\UsersController.php
C:\xampp\htdocs\demo1>

```

Following is the code for **UserController.php**

```

<?php

class UsersController extends \Phalcon\Mvc\Controller
{
    public function indexAction()
    {
        echo "Users Controller has been called";
    }
}

```

The output will be displayed if we hit the following URL:

<http://localhost/demo1/users>



## 8. Phalcon – Views

Views are information being presented to the end user. A view can be considered as a web page with the appropriate response to be displayed. The response is received through the controller which interacts with the model.

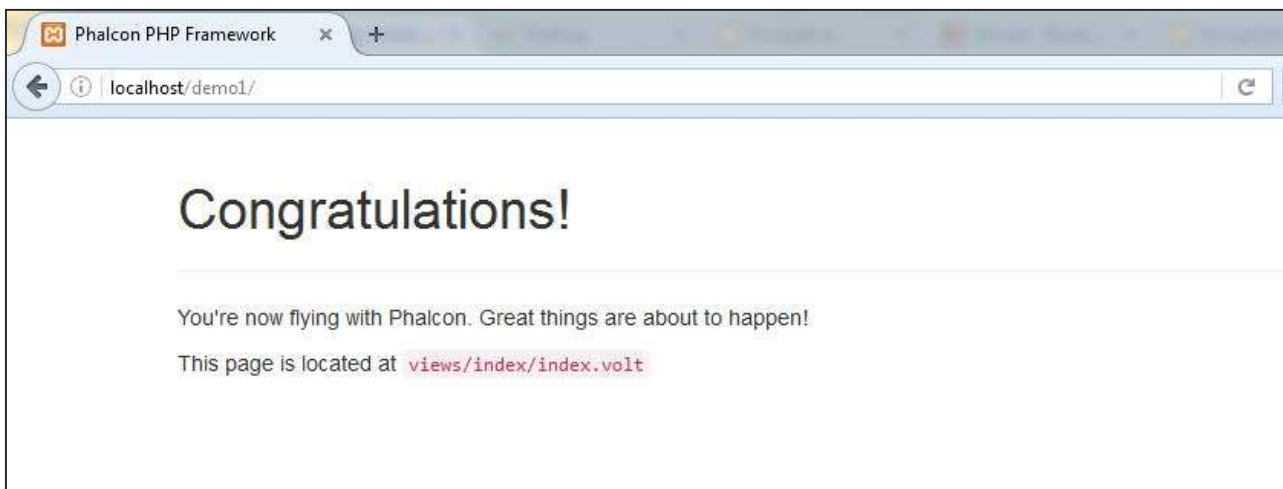
Specifically in Phalcon, the view consists of Volt code, PHP and HTML. A set of special delimiters is available to enter in Volt mode. `{% ... %}` is used to execute statements such as for-loops or assign values, and `{{ ... }}` prints the result of an expression to the template.

Views in Phalcon are basically classified into two types:

- Volt
- phtml

### Volt

Following is the screenshot of the output we had created for the project **demo1** in the previous chapter.



This output is achieved with the help of file **views/index/index.volt**.

### Features of Volt Files

- It is a template written in C language and is considerably fast as compared to other languages.
- It includes a set of highly integrated components, which are very beneficial in Phalcon.
- It can also be used as a stand-alone component.
- Volt is compiled to pure PHP code.

Following is the code for **index.volt** which loads by default for any project.

```
<!--<div class="page-header">
    <h1>Congratulations!</h1>
</div>-->
<p>This is my first web application in Phalcon </p>
<!--<p>You're now flying with Phalcon. Great things are about to happen!</p>

<p>This page is located at <code>views/index/index.volt</code></p>-->
```

## Hierarchical Rendering

Views in Phalcon support hierarchical rendering and **Phalcon\Mvc\View** is used as default rendering component. This component uses PHP as template engine in comparison with volt files which uses C as a template language.

These views should have **.phtml** extension. The default directory of views for the given project consists of the following three files:

- **Action view:** This view is called to execute a particular action. It is called when "show" action is executed.
- **Controller layout:** This view is present inside the layouts folder. For example, **C:\xampp\htdocs\demo\app\views\layouts**. It invokes the method calls associated with the appropriate controller. The code implemented in layout will be implemented as and when required.
- **Main layout:** This layout view will invoke the main action and it will be shown for every controller or action within the web application.

## Difference between .volt and .phtml Files

.volt	.phtml
.volt extension is used when the template engine set up in the application is written in C	.phtml is used when the template engine is PHP itself
It can be used as a stand-alone component	It cannot be used as a stand-alone component
Volt views are compiled to PHP code	phtml files itself includes PHP code so there is no need of compilation in Phalcon framework

## Variables

Variables are assigned and changed in the template using 'set'.

### Declaring an array

```
{% set fruits = ['Apple', 'Banana', 'Orange'] %}
```

### Declaring a string

```
{% set name ="John Kennedy" %}
```

## Comments

Comments may also be added to a template using the `{# ... #}` delimiters. All text inside them is just ignored in the final output.

```
{# note: this is a comment  
    {% set price = 100; %}  
#}
```

## Example

```
{% set fruits = ['Apple', 'Banana', 'Orange'] %}  
<h1>Fruits</h1>  
<ul>  
{% for fruit in fruits %}  
    <li>{{ fruit|e }}</li>  
{% endfor %}  
</ul>  
  
{% set robots = ['Voltron', 'Astro Boy', 'Terminator', 'C3PO'] %}  
  
<ul>  
{% for robot in robots %}  
    <li>{{ robot }}</li>  
{% endfor %}  
</ul>
```

## Output

The code will produce the following output screen:





## 9. Phalcon – Routing

The router component allows to define routes that are mapped to the controllers or handlers that should receive the request. A router parses a URI as per the information received.

Every router in the web application has two modes:

- MVC mode
- Match-only mode

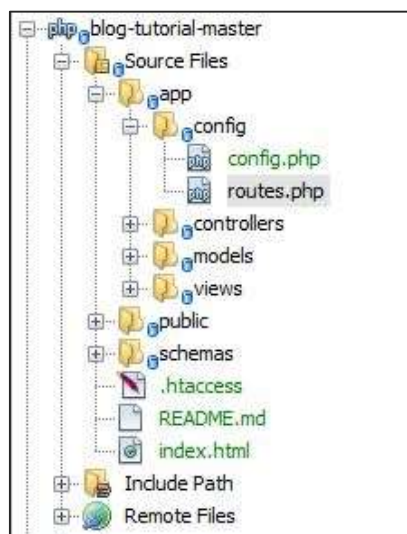
The first mode is ideal for working with MVC applications. Following is the syntax to define a route in Phalcon.

```
$router = new Router();

// Define a route
$router->add(
    "<URI-Name>",
    [
        "controller" => "<controller-name>",
        "action"      => "<action-name>",
    ]
);
```

### Example

For searching a category, let us create a route in **routes.php** of config folder.



Consider creating a route which will call a method login as we invoke “**UsersController**”. In such a case, it is suggested to create a route which maps the given URL.

```
<?php

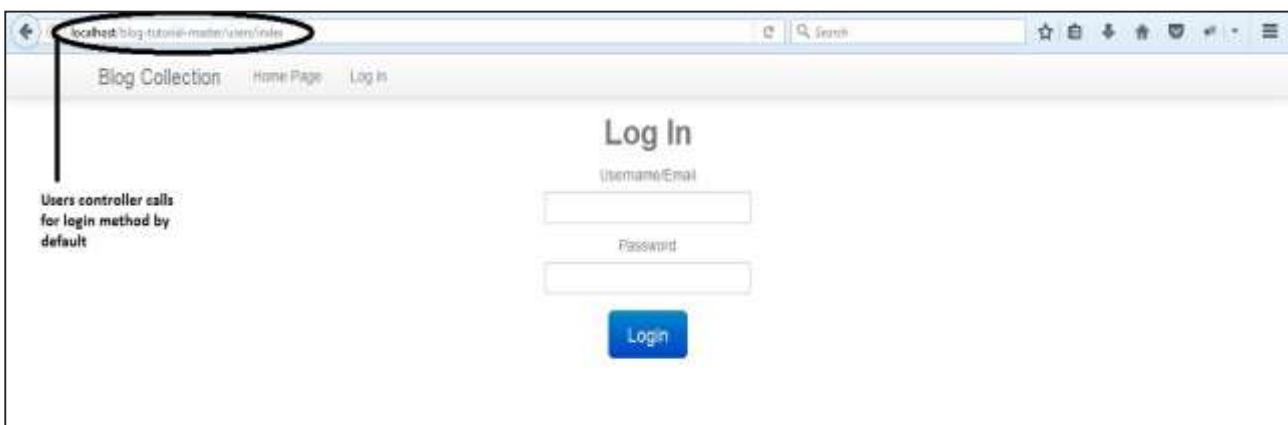
$router = new Phalcon\Mvc\Router();

$router->add('/login', array(
    'controller' => 'users',
    'action' => 'login',
));

return $router;
```

## Output

The code will produce the following output:



## 10. Phalcon – Database Connectivity

In this chapter, we will discuss the database connectivity related to Phalcon.

### Creation of Database and Design

We will focus on creating a database for blogs which maintains posts along with the categories as per the entries of users.

Database Name: blog-tutorial

Query used for creating the database:

```
drop database blog-tutorial (if exists)
create database blog-tutorial
```

After creation of the database, the database will be listed as shown in the following screenshot.



Phalcon uses commands for creation of **models**, **controllers**, and even projects. Let's see how it works.

**Step 1:** Create a project named blog-tutorial.



```

C:\xampp\htdocs>phalcon create-project blog-tutorial
Phalcon DevTools (3.0.1)

Success: Controller "index" was successfully created.
C:\xampp\htdocs\blog-tutorial\app\controllers\IndexController.php
Success: Project "blog-tutorial" was successfully created.
C:\xampp\htdocs>

```

**Step 2:** Configure the web application that connects to the database which we created for managing blogs.

```

<?php

return new \Phalcon\Config(array(
    'database' => array(
        'adapter' => 'Mysql',
        'host' => 'localhost',
        'username' => 'root',
        // 'dbname' => 'blog_tutorial',
        'password' => '',
        'name' => 'blog_tutorial',
    ),
    'application' => array(
        'controllersDir' => __DIR__ . '/../..app/controllers/',
        'modelsDir' => __DIR__ . '/../..app/models/',
        'viewsDir' => __DIR__ . '/../..app/views/',
        'baseUri' => '/blog-tutorial/',
    )
));

```

## 11. Phalcon – Switching Databases

We have used a MySQL database in our application. If we wanted to change the database software midstream, it would not be too hard, as long as we have the same data structure in our new database.

### PostgreSQL

Configure the web application which will connect to PostgreSQL database.

This can be achieved using the following code. The services will include **Phalcon\Db\Adapter\Pdo\Postgresql**

```
use Phalcon\Db\Adapter\Pdo\Postgresql;

$config = [
    'host'      => 'localhost',
    'dbname'    => 'blog_tutorial',
    'port'      => 5432,
    'username'  => 'root',
    'password'  => ''
];

$connection = new Postgresql($config);
```

### SQLite

For implementing SQLite connection, the configuration should be extended with **Phalcon\Db\Adapter\Pdo\Sqlite** abstract class.

```
<?php

use Phalcon\Db\Adapter\Pdo\Sqlite;

$connection = new Sqlite(['dbname' => '/tmp/blog_tutorial.sqlite']);
```

## Oracle

For implementing Oracle database connection in Phalcon, the configuration should be extended with **Phalcon\Db\Adapter\Pdo\Oracle** abstract class.

```
<?php

use Phalcon\Db\Adapter\Pdo\Oracle;

$config = array(
    "dbname" => "//localhost/blog_tutorial",
    "username" => "root",
    "password" => ""
);

$connection = new Phalcon\Db\Adapter\Pdo\Oracle($config);
```

## 12. Phalcon – Scaffolding Application

Scaffolding usually refers to a type of code generation where we point it to a web application database, which results in creating a basic CRUD (Create, Read, Update, Delete) application.

Before designing a CRUD application, it is important to design database tables as per the need of the application.

**Step 1:** Create a scaffolding application which will include all crud operations.

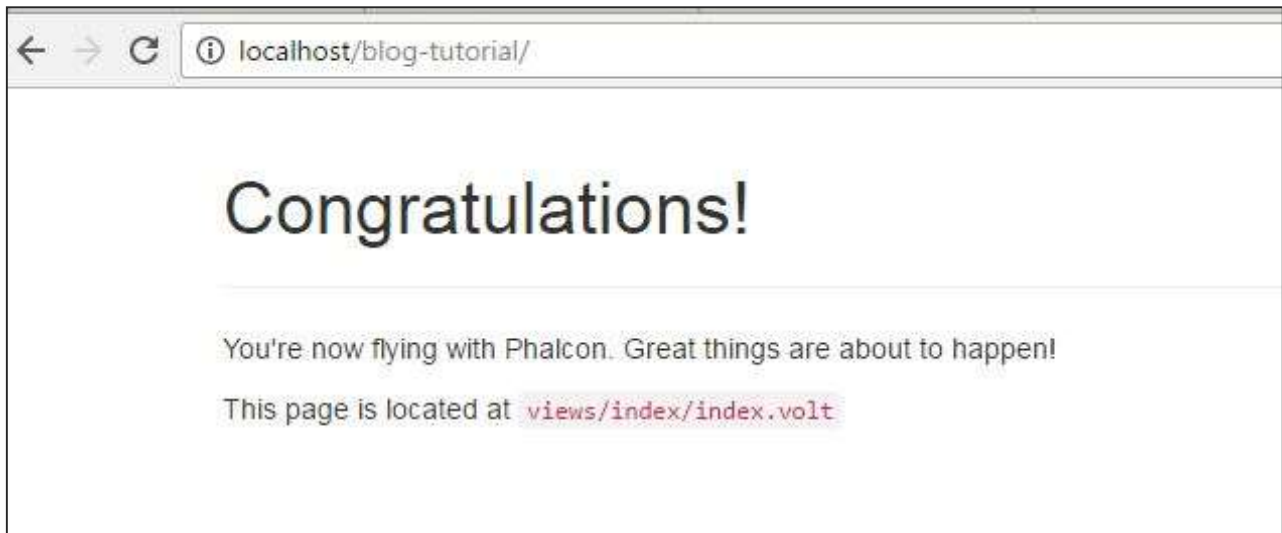
```
Command: phalcon scaffold <table-name>
```

```
C:\xampp\htdocs\blog-tutorial>phalcon scaffold users
Phalcon DevTools (3.0.1)

    Success: Model "Users" was successfully created.
C:\xampp\htdocs\blog-tutorial/app/controllers/UsersController.php
C:\xampp\htdocs\blog-tutorial/app/views/layouts/users.phtml
C:\xampp\htdocs\blog-tutorial/app/views/users/index.phtml
C:\xampp\htdocs\blog-tutorial/app/views/users/search.phtml
C:\xampp\htdocs\blog-tutorial/app/views/users/new.phtml
C:\xampp\htdocs\blog-tutorial/app/views/users/edit.phtml
C:\xampp\htdocs\blog-tutorial>phalcon scaffold categories
Phalcon DevTools (3.0.1)

    Success: Model "Categories" was successfully created.
C:\xampp\htdocs\blog-tutorial/app/controllers/CategoriesController.php
C:\xampp\htdocs\blog-tutorial/app/views/layouts/categories.phtml
C:\xampp\htdocs\blog-tutorial/app/views/categories/index.phtml
C:\xampp\htdocs\blog-tutorial/app/views/categories/search.phtml
C:\xampp\htdocs\blog-tutorial/app/views/categories/new.phtml
C:\xampp\htdocs\blog-tutorial/app/views/categories/edit.phtml
C:\xampp\htdocs\blog-tutorial>phalcon scaffold posts
Phalcon DevTools (3.0.1)

    Success: Model "Posts" was successfully created.
C:\xampp\htdocs\blog-tutorial/app/controllers/PostsController.php
C:\xampp\htdocs\blog-tutorial/app/views/layouts/posts.phtml
C:\xampp\htdocs\blog-tutorial/app/views/posts/index.phtml
C:\xampp\htdocs\blog-tutorial/app/views/posts/search.phtml
C:\xampp\htdocs\blog-tutorial/app/views/posts/new.phtml
C:\xampp\htdocs\blog-tutorial/app/views/posts/edit.phtml
```



The scaffold generator of Phalcon once executed will create files and folders which are described in the following table.

app/controllers/CategoriesController.php	The Categories controller
app/models/Categories.php	The Categories model
app/views/layout/Categories.phtml	Controller layout for Categories
app/views/categories/new.phtml	View for the action "new"
app/views/categories/edit.phtml	View for the action "edit"
app/views/categories/search.phtml	View for the action "search"

**Step 2:** Create an index page (Combination of phtml and volt).

Code to be included in index.phtml in users folder.

```
<?php use Phalcon\Tag as Tag ?>
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
```



```

<title>Blog Tutorial</title>

<link rel="stylesheet" type="text/css"
href="http://netdna.bootstrapcdn.com/twitter-bootstrap/2.2.1/css/bootstrap-
combined.min.css"/>

<meta name="viewport" content="width=device-width, initial-scale=1.0">
</head>
<body>

<div class="navbar navbar-fixed-top">
    <div class="navbar-inner">
        <div class="container">
            <a class="btn btn-navbar" data-toggle="collapse" data-
target=".nav-collapse">
                <span class="icon-bar"></span>
                <span class="icon-bar"></span>
                <span class="icon-bar"></span>
            </a>
            <a class="brand" href="#">Blog Collection</a>
            <div class="nav-collapse">
                <ul class="nav pull-left">
                    <li>
                        <?php echo Phalcon\Tag::linkTo('index', 'Home Page') ?>
                    </li>
                    <?php if ($this->session->has('auth')) { ?>
                        <li>
                            <?php echo Phalcon\Tag::linkTo('posts/index',
'+Posts') ?>
                        </li>
                        <li>
                            <?php echo
Phalcon\Tag::linkTo('categories/index', '+Categories') ?>
                        </li>
                        <li>
                            <?php echo Phalcon\Tag::linkTo('users/logout',
'Log out') ?>
                        </li>
                    <?php } else { ?>
                        <li>

```

```

                                <?php echo Phalcon\Tag::linkTo('users/index',
'Log in') ?>

                                </li>
                                <?php } ?>
                            </ul>
                        </div>
                    </div>
                </div>
            </div>

            <?php echo $this->getContent() ?>

            <script src="http://netdna.bootstrapcdn.com/twitter-
bootstrap/2.2.1/js/bootstrap.min.js"></script>
        </body>
    </html>

```

Default file **index.volt** will include the following code.

```

<?php echo $this->getContent() ?>

<div align="center">

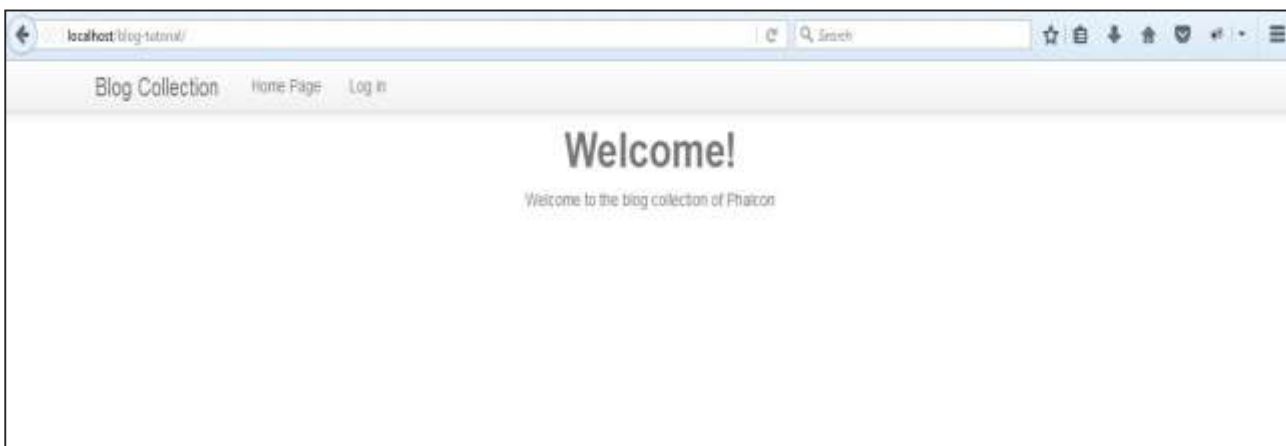
    <h1>Welcome!</h1>

    <p>Welcome to the blog collection of Phalcon</p>

</div>

```

Successful execution of the above code produces the following output.



**Step 3:** Change with the respective models.

## Users.php

```
<?php

class Users extends \Phalcon\Mvc\Model
{

    /**
     * @var integer
     *
     */
    public $id;

    /**
     * @var string
     *
     */
    public $login;

    /**
     * @var string
     *
     */
    public $password;

    /**
     * Initializer method for model.
     */
    public function initialize()
    {
        $this->hasMany("id", "Posts", "users_id");
    }
}
```

The function named '**initialize**' helps in implementing relationship between id and users\_id in Posts table, which means that each unique user has many posts associated in the table.

## Posts.php

```
<?php

class Posts extends \Phalcon\Mvc\Model
{

    /**
     * @var integer
     *
     */
    public $id;

    /**
     * @var string
     *
     */
    public $title;

    /**
     * @var string
     *
     */
    public $slug;

    /**
     * @var string
     *
     */
    public $content;

    /**
     * @var string
     *
     */
    public $created;
```

```

    /**
     * @var integer
     *
     */
    public $users_id;

    /**
     * @var integer
     *
     */
    public $categories_id;

    /**
     * Initializer method for model.
     */
    public function initialize()
    {
        $this->belongsTo("users_id", "Users", "id");
        $this->belongsTo("categories_id", "Categories", "id");
    }
}

```

The function '**initialize**' includes relationship constraint mentioning the foreign key and primary key relationship with the table.

**users\_id** refers to id in "Users" table.

**categories\_id** refers to id in "Categories" table.

## Categories.php

```

<?php

class Categories extends \Phalcon\Mvc\Model
{

    /**
     * @var integer

```

```

    *
    */

    public $id;

    /**
     * @var string
     *
     */
    public $name;

    /**
     * @var string
     *
     */
    public $slug;

    /**
     * Initializer method for model.
     */
    public function initialize()
    {
        $this->hasMany("id", "Posts", "categories_id");
    }
}

```

Similar to Users model, the **'initialize'** function specifies that it includes many **categories\_id** for the given post.

## Designing the Login Page

### UsersController.php

```

<?php

class UsersController extends Phalcon\Mvc\Controller
{
    public function indexAction()

```

```
{

}

public function loginAction()
{

    if ($this->request->isPost()) {
        $user = Users::findFirst(array(
            'login = :login: and password = :password:',
            'bind' => array(
                'login' => $this->request->getPost("login"),
                'password' => $this->request->getPost("password")
            )
        ));

        if ($user === false){
            $this->flash->error("Incorrect credentials");
            return $this->dispatcher->forward(array(
                'controller' => 'users',
                'action' => 'index'
            ));
        }
        $this->session->set('auth', $user->id);

        $this->flash->success("You've been successfully logged in");
    }
    return $this->dispatcher->forward(array(
        'controller' => 'posts',
        'action' => 'index'
    ));
}

public function logoutAction()
{
    $this->session->remove('auth');
```

```

        return $this->dispatcher->forward(array(
            'controller' => 'posts',

            'action' => 'index'

        ));
    }
}

```

The **UsersController** includes functionality with log in and log out features. It checks for the associated value in the records for “Users” table. If the value gets authenticated, the user successfully logs in or else gets an error message.

Following is the output of the above code.

Once logged in to the web application, the output will look as shown in the following screenshot.

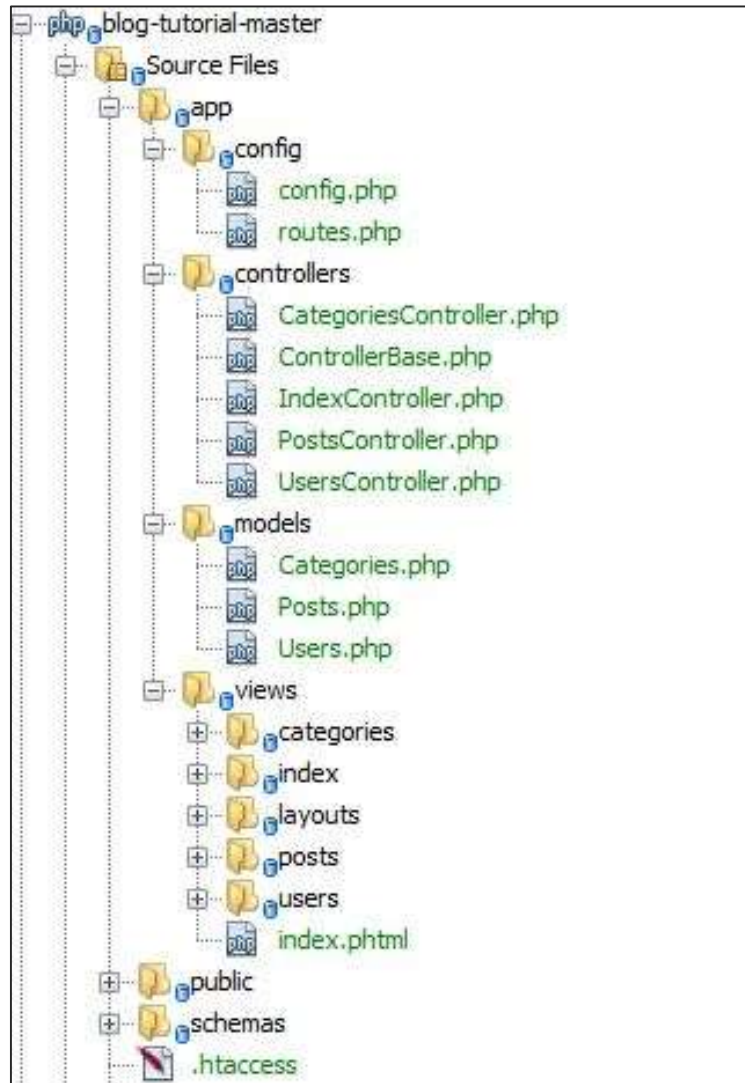
Title	Created	
First Post	1	<a href="#">Details</a>
ABC	12	<a href="#">Details</a>

We will look at implementing views in the next chapter which will focus on categories and posts management.



## Creating Views

Following is the complete structure of Blog-tutorial-master project.



The associated view for displaying the home page after the user successfully logs in is **"index.phtml"**.

```
<?php use Phalcon\Tag as Tag ?>
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Blog Tutorial</title>
        <link rel="stylesheet" type="text/css"
href="http://netdna.bootstrapcdn.com/twitter-bootstrap/2.2.1/css/bootstrap-
combined.min.css"/>
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```

</head>
<body>

    <div class="navbar navbar-fixed-top">
        <div class="navbar-inner">
            <div class="container">
                <a class="btn btn-navbar" data-toggle="collapse" data-
target=".nav-collapse">
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                </a>
                <a class="brand" href="#">Blog Collection</a>
                <div class="nav-collapse">
                    <ul class="nav pull-left">
                        <li>
                            <?php echo Phalcon\Tag::linkTo('index', 'Home Page') ?>
                        </li>
                        <?php if ($this->session->has('auth')) { ?>
                            <li>
                                <?php echo Phalcon\Tag::linkTo('posts/index',
'+Posts') ?>
                            </li>
                            <li>
                                <?php echo
Phalcon\Tag::linkTo('categories/index', '+Categories') ?>
                            </li>
                            <li>
                                <?php echo Phalcon\Tag::linkTo('users/logout',
'Log out') ?>
                            </li>
                        <?php } else { ?>
                            <li>
                                <?php echo Phalcon\Tag::linkTo('users/index',
'Log in') ?>
                            </li>
                        <?php } ?>
                    </ul>
                </div>
            </div>
        </div>
    </div>

```

```

        </div>
    </div>
</div>

<?php echo $this->getContent() ?>

    <script src="http://netdna.bootstrapcdn.com/twitter-
bootstrap/2.2.1/js/bootstrap.min.js"></script>
</body>
</html>

```

## Category Management

---

### index.phtml

```

<?php echo $this->getContent(); ?>

<div align="right">
    <?php echo \Phalcon\Tag::linkTo(array("categories/new", "Create Categories",
"class" => "btn")) ?>
</div>

<div align="center">

    <h1>Search categories</h1>

    <?php echo \Phalcon\Tag::form(array("categories/search", "autocomplete" => "off")) ?>
    <table align="center">

```

```

        <tr>
            <td align="right">
                <label for="id">Id</label>
            </td>
            <td align="left">
                <?php echo \Phalcon\Tag::textField(array("id", "type" => "numeric")) ?>
            </td>
        </tr>
        <tr>
            <td align="right">
                <label for="name">Name</label>
            </td>
            <td align="left">
                <?php echo \Phalcon\Tag::textField(array("name", "size" => 30)) ?>
            </td>
        </tr>
        <tr>
            <td align="right">
                <label for="slug">Slug</label>
            </td>
            <td align="left">
                <?php echo \Phalcon\Tag::textField(array("slug", "size" => 30)) ?>
            </td>
        </tr>
        <tr>
            <td></td>
            <td><?php echo \Phalcon\Tag::submitButton(array("Search", "class" =>
"btn")) ?></td>
        </tr>
    </table>
</form>
</div>

```

**Edit.phtml**

```

<?php use Phalcon\Tag as Tag; ?>

<?php echo $this->getContent(); ?>

<?php echo \Phalcon\Tag::form("categories/save") ?>

    <table width="100%">
        <tr>
            <td align="left"><?php echo Tag::linkTo(array("categories",
"Back", "class" => "btn")) ?></td>
            <td align="right"><?php echo Tag::submitButton(array("Save",
"class" => "btn")) ?></td>
        <tr>
        </table>

    <div align="center">
        <h1>Edit categories</h1>
    </div>

    <table align="center">
        <tr>
            <td align="right">
                <label for="name">Name</label>
            </td>
            <td align="left">
                <?php echo Tag::textField(array("name", "size" => 30)) ?>
            </td>
        </tr>
        <tr>
            <td align="right">
                <label for="slug">Slug</label>
            </td>
            <td align="left">

```

```

                <?php echo Tag::textField(array("slug", "size" => 30)) ?>
            </td>
        </tr>
    </table>

</form>

```

## Search.phtml

```

<?php $this->getContent(); ?>

<table width="100%">
    <tr>
        <td align="left">
            <?php echo \Phalcon\Tag::linkTo(array("categories/index", "Go
Back", "class" => "btn")); ?>
        </td>
        <td align="right">
            <?php echo \Phalcon\Tag::linkTo(array("categories/new", "Create a
Category", "class" => "btn")); ?>
        </td>
    <tr>
    </table>

<table class="browse table" align="center" width="60%">
    <thead>
        <tr>
            <th>Id</th>
            <th>Name</th>
            <th>Slug</th>
        </tr>
    </thead>
    <tbody>
        <?php
            if(isset($page->items)){
                foreach($page->items as $categories){ ?>

```

```

        <tr>
            <td><?php echo $categories->id ?></td>
            <td><?php echo $categories->name ?></td>
            <td><?php echo $categories->slug ?></td>
            <td><?php echo
\Phalcon\Tag::linkTo(array("categories/edit/".$categories->id, "Edit")); ?></td>
            <td><?php echo
\Phalcon\Tag::linkTo(array("categories/delete/".$categories->id, "Delete")); ?></td>
        </tr>
    <?php }
    } ?>
</tbody>
</table>

<div class="pagination" align="center">
    <ul>
        <li><?php echo \Phalcon\Tag::linkTo("categories/search", "First") ?></li>
        <li><?php echo \Phalcon\Tag::linkTo("categories/search?page=".$page-
>before, "Previous") ?></li>
        <li><?php echo \Phalcon\Tag::linkTo("categories/search?page=".$page-
>next, "Next") ?></li>
        <li><?php echo \Phalcon\Tag::linkTo("categories/search?page=".$page-
>last, "Last") ?></li>
    </ul>
</div>

```

## Search.phtml

```

<?php $this->getContent(); ?>

<table width="100%">
    <tr>
        <td align="left">
            <?php echo \Phalcon\Tag::linkTo(array("categories/index", "Go
Back", "class" => "btn")); ?>
        </td>
        <td align="right">
            <?php echo \Phalcon\Tag::linkTo(array("categories/new", "Create a
Category", "class" => "btn")); ?>
        </td>
    </tr>
</table>

```

```

        </td>

    <tr>
</table>

<table class="browse table" align="center" width="60%">
    <thead>
        <tr>
            <th>Id</th>
            <th>Name</th>
            <th>Slug</th>
        </tr>
    </thead>
    <tbody>
    <?php
        if(isset($page->items)){
            foreach($page->items as $categories){ ?>
                <tr>
                    <td><?php echo $categories->id ?></td>
                    <td><?php echo $categories->name ?></td>
                    <td><?php echo $categories->slug ?></td>
                    <td><?php echo
\Phalcon\Tag::linkTo(array("categories/edit/".$categories->id, "Edit")); ?></td>
                    <td><?php echo
\Phalcon\Tag::linkTo(array("categories/delete/".$categories->id, "Delete")); ?></td>
                </tr>
            <?php }
        } ?>
    </tbody>
</table>
<div class="pagination" align="center">
    <ul>
        <li><?php echo \Phalcon\Tag::linkTo("categories/search", "First") ?></li>
        <li><?php echo \Phalcon\Tag::linkTo("categories/search?page=".$page-
>before, "Previous") ?></li>
        <li><?php echo \Phalcon\Tag::linkTo("categories/search?page=".$page-
>next, "Next") ?></li>
        <li><?php echo \Phalcon\Tag::linkTo("categories/search?page=".$page-
>last, "Last") ?></li>
    </ul>

```



```

    </ul>
</div>

```

Business logic is developed in **CategoriesController** which includes methods for creating, editing, and searching the categories.

```

<?php

use \Phalcon\Tag as Tag,
    \Phalcon\Mvc\Model\Criteria;

class CategoriesController extends ControllerBase
{

    public function indexAction()
    {
        $this->session->conditions = null;
    }

    public function searchAction()
    {

        $numberPage = 1;
        if ($this->request->isPost()) {
            $query = Criteria::fromInput($this->di, "Categories", $_POST);
            $this->session->conditions = $query->getConditions();
        } else {
            $numberPage = $this->request->getQuery("page", "int");
            if ($numberPage <= 0) {
                $numberPage = 1;
            }
        }

        $parameters = array();
        if ($this->session->conditions) {
            $parameters["conditions"] = $this->session->conditions;
        }
    }
}

```

```

// $parameters["order"] = "id";

$categories = Categories::find($parameters);
if (count($categories) == 0) {
    $this->flash->notice("The search did not find any categories");
    return $this->dispatcher->forward(array(
        "controller" => "categories",
        "action" => "index"
    ));
}

$paginator = new \Phalcon\Paginator\Adapter\Model(array(
    "data" => $categories,
    "limit"=> 10,
    "page" => $numberPage
));
$page = $paginator->getPaginate();

$this->view->setVar("page", $page);
}

public function newAction()
{

}

public function editAction($id)
{
    $request = $this->request;
    if (!$request->isPost()) {

        $categories = Categories::findFirst(array(
            'id = :id:',
            'bind' => array('id' => $id)
        ));
        if (!$categories) {

```

```

        $this->flash->error("The category was not found");
        return $this->dispatcher->forward(array(

            "controller" => "categories",
            "action" => "index"

        ));
    }
    $this->view->setVar("id", $categories->id);

    Tag::displayTo("id", $categories->id);
    Tag::displayTo("name", $categories->name);
    Tag::displayTo("slug", $categories->slug);
}
}
public function createAction()
{
    if (!$this->request->isPost()) {
        return $this->dispatcher->forward(array(
            "controller" => "categories",
            "action" => "index"
        ));
    }

    $categories = new Categories();
    $categories->id = $this->request->getPost("id");
    $categories->name = $this->request->getPost("name");
    $categories->slug = $this->request->getPost("slug");
    if (!$categories->save()) {
        foreach ($categories->getMessages() as $message) {
            $this->flash->error((string) $message);
        }
        return $this->dispatcher->forward(array(
            "controller" => "categories",
            "action" => "new"
        ));
    } else {
        $this->flash->success("The category was created successfully");
    }
}

```

```

        return $this->dispatcher->forward(array(
            "controller" => "categories",

            "action" => "index"
        ));
    }
}

public function saveAction()
{
    if (!$this->request->isPost()) {
        return $this->dispatcher->forward(array(
            "controller" => "categories",
            "action" => "index"
        ));
    }

    $category = Categories::findFirst(array(
        'id = :id:',
        'bind' => array('id' => $this->request->getPost("id"))
    ));

    if (!$category) {
        $this->flash->error("The category does not exist");
        return $this->dispatcher->forward(array(
            "controller" => "categories",
            "action" => "index"
        ));
    }

    $categories->id = $this->request->getPost("id");
    $categories->name = $this->request->getPost("name");
    $categories->slug = $this->request->getPost("slug");

    if (!$categories->save()) {
        foreach ($categories->getMessages() as $message) {
            $this->flash->error((string) $message);
        }
    }
}

```

```

        return $this->dispatcher->forward(array(
            "controller" => "categories",

            "action" => "edit",
            "params" => array($categories->id)
        ));

    } else {
        $this->flash->success("categories was updated successfully");
        return $this->dispatcher->forward(array(
            "controller" => "categories",
            "action" => "index"
        ));
    }
}

public function deleteAction($id)
{
    $categories = Categories::findFirst(array(
        'id = :id:',
        'bind' => array('id' => $id)
    ));

    if (!$categories) {
        $this->flash->error("The category was not found");
        return $this->dispatcher->forward(array(
            "controller" => "categories",
            "action" => "index"
        ));
    }

    if (!$categories->delete()) {
        foreach ($categories->getMessages() as $message){
            $this->flash->error((string) $message);
        }
        return $this->dispatcher->forward(array(
            "controller" => "categories",
            "action" => "search"
        ));
    }
}

```

```

    ));
} else {

    $this->flash->success("The category was deleted");
    return $this->dispatcher->forward(array(
        "controller" => "categories",
        "action" => "index"
    ));
}
}
}

```

The above code produces the following output.

id	Name	Slug	Edit	Delete
1	Radhika	Slug N	Edit	Delete
2	hi	hi	Edit	Delete
3	slug	slug	Edit	Delete
4	hello	hello	Edit	Delete

The screenshot shows a web interface for creating a new category. At the top, there is a navigation bar with links: 'Blog Collection', 'Home Page', '+Posts', '+Categories', and 'Log out'. Below the navigation bar, on the left, is a 'Go Back' button, and on the right, is a 'Save' button. The main heading is 'Create categories'. Below this heading, there are two input fields: 'Name' and 'Slug', each with a text input box.

The list of categories will be displayed as seen in the following screenshot.

The screenshot shows a web interface displaying a list of categories. At the top, there is a navigation bar with links: 'Blog Collection', 'Home Page', '+Posts', '+Categories', and 'Log out'. Below the navigation bar, on the left, is a 'Go Back' button, and on the right, is a 'Create a Category' button. The main content is a table with the following columns: 'Id', 'Name', 'Slug', 'Edit', and 'Delete'.

Id	Name	Slug	Edit	Delete
1	Radhika	Radhika	Edit	Delete
2	hi	hi	Edit	Delete
3	slug	slug	Edit	Delete
4	hello	hello	Edit	Delete
5	hi	hi	Edit	Delete
6	hi	hi	Edit	Delete
7	hi	hi	Edit	Delete
8	hi hi	hihi	Edit	Delete

At the bottom of the table, there are pagination controls: 'First', 'Previous', 'Next', and 'Last'.

## 13. Phalcon – Query Language

Phalcon Query Language (PHQL) also called as **PhalconQL** is a high-level SQL dialect which standardizes SQL queries for the database systems supported by Phalcon.

It includes a parser, written in C, which translates the syntax in target RDBMS.

Here is a list of some of the prominent features of Phalcon query language:

- For security of web application, it uses bound parameters.
- Tables are treated as models whereas columns are treated as class attributes.
- All the data manipulation statements are used to prevent data loss which can occur.
- SQL injection is prevented by keeping SQL query call one at a time.

### Creating a PHQL Query

---

Queries are created by instantiating class **Phalcon\Mvc\Model\Query**

#### Example

```
// Instantiate the Query
$query = new Query(
    "SELECT * FROM Users",
    $this->getDI()
);

// Execute the query returning a result if any
$cars = $query->execute();
```

In the previous chapters, we have seen the working of scaffold web application named blog tutorial. It included searching categories as per name or slug.

Following is the code included for searchAction.

```
public function searchAction()
{

    $numberPage = 1;
    if ($this->request->isPost()) {
        $query = Criteria::fromInput($this->di, "Categories", $_POST);
        $this->session->conditions = $query->getConditions();
    } else {
```



```

        $numberPage = $this->request->getQuery("page", "int");
        if ($numberPage <= 0) {
            $numberPage = 1;
        }
    }
    $parameters = array();
    if ($this->session->conditions) {
        $parameters["conditions"] = $this->session->conditions;
    }
    // $parameters["order"] = "id";
    $categories = Categories::find($parameters);
    if (count($categories) == 0) {
        $this->flash->notice("The search did not find any categories");
        return $this->dispatcher->forward(array(
            "controller" => "categories",
            "action" => "index"
        ));
    }
    $paginator = new \Phalcon\Paginator\Adapter\Model(array(
        "data" => $categories,
        "limit"=> 10,
        "page" => $numberPage
    ));
    $page = $paginator->getPagate();
    $this->view->setVar("page", $page);
}

```

The PHQL query executed (highlighted) in the controller will fetch all the results as per the search condition. The result of any search query as per condition will be displayed as in the screenshot.

Following is the output received on successful execution of the above code.

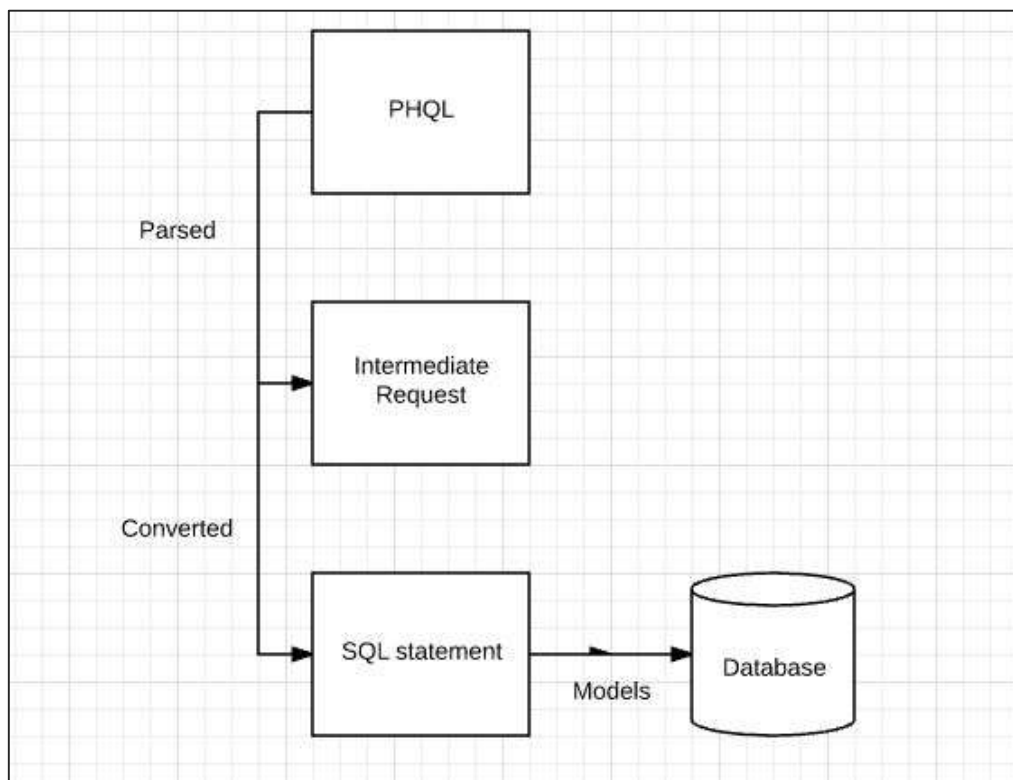
id	Name	Slug	Edit	Delete
1	Rodhika	slug R	Edit	Delete
2	R	R	Edit	Delete
3	slug	slug	Edit	Delete
4	beko	beko	Edit	Delete

## PHQL Life Cycle

Being a high-level language, PHQL provides the ability to the developers to personalize and customize various aspects as per requirements.

Following is the life cycle of each PHQL statement executed in Phalcon:

- Every PHQL statement is parsed and converted as an Intermediate Representation (IR) which is completely independent of the SQL implemented by the database system.
- The IR is converted to SQL statement as per the database system which is used in the web application. SQL statements generated are associated with the model.
- All PHQL statements are parsed once and cached in the memory. If the same statement result is executed, it will help in faster performance.



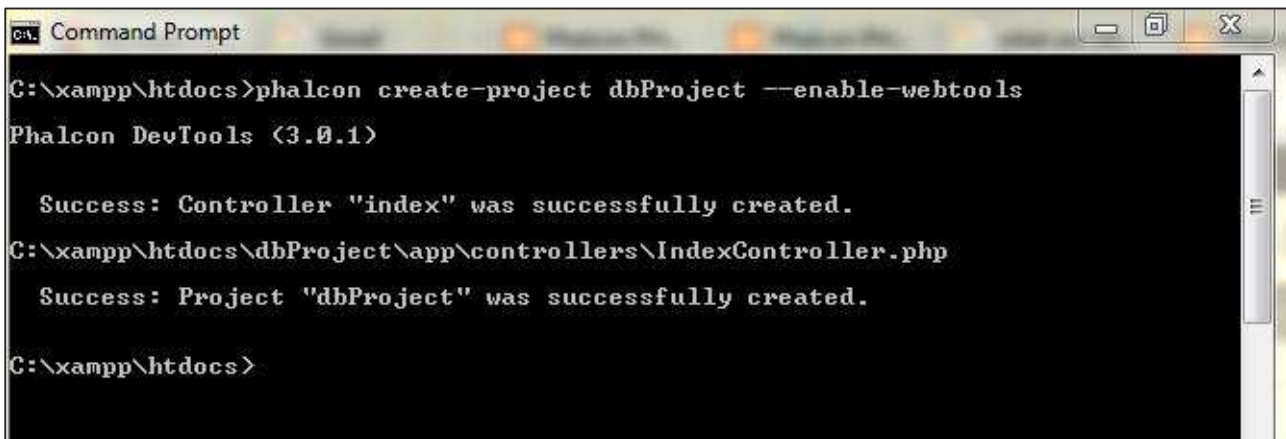
## 14. Phalcon – Database Migration

Database migration is important for the following reasons:

- Database migration helps in transferring data between the specified storage types.
- Database migration refers to the context of web-based applications migrating from one platform to another.
- This process usually takes place to keep a track of data which is being outdated.

Phalcon performs database migration process in the following way:

**Step 1:** Create a project named “dbProject” in **xampp/wamp** directory.



```
Command Prompt
C:\xampp\htdocs>phalcon create-project dbProject --enable-webtools
Phalcon DevTools (3.0.1)

Success: Controller "index" was successfully created.
C:\xampp\htdocs\dbProject\app\controllers\IndexController.php
Success: Project "dbProject" was successfully created.
C:\xampp\htdocs>
```

**Step 2:** Configure the project with the appropriate database connectivity.

```
<?php
/*
 * Modified: prepend directory path of current file, because of this file own
 * different ENV under between Apache and command line.
 * NOTE: please remove this comment.
 */
defined('BASE_PATH') || define('BASE_PATH', getenv('BASE_PATH') ?
realpath(dirname(__FILE__) . '/../..'));
defined('APP_PATH') || define('APP_PATH', BASE_PATH . '/app');

return new \Phalcon\Config([
    'database' => [
        'adapter'    => 'Mysql',
        'host'       => 'localhost',
        'username'   => 'root',
        'password'   => '',
        'dbname'     => 'demodb',
        'charset'    => 'utf8',
    ],
]);
```

```

    'application' => [
        'appDir'      => APP_PATH . '/',
        'controllersDir' => APP_PATH . '/controllers/',
        'modelsDir'    => APP_PATH . '/models/',
        'migrationsDir' => APP_PATH . '/migrations/',
        'viewsDir'     => APP_PATH . '/views/',
        'pluginsDir'   => APP_PATH . '/plugins/',
        'libraryDir'   => APP_PATH . '/library/',
        'cacheDir'     => BASE_PATH . '/cache/',
        'baseUri'      => '/dbProject/',
    ]
});

```

**Step 3:** Execute the command for migration of tables included within the database “demodb”. For now, it includes one table “users”.

```

C:\xampp\htdocs\dbProject>phalcon migration generate

Phalcon DevTools (3.0.1)

1477906916.045: SHOW TABLES => 1477906916.046 <0.00099992752075195>
1477906916.048: DESCRIBE `demodb`.`user` => 1477906916.051 <0.0030009746551514>
1477906916.052: SHOW INDEXES FROM `demodb`.`user` => 1477906916.053 <0.00099992752075195>
1477906916.054: SELECT KCU.TABLE_NAME, KCU.COLUMN_NAME, KCU.CONSTRAINT_NAME, KCU.REFERENCED_TABLE_SCHEMA, KCU.REFERENCED_TABLE_NAME, KCU.REFERENCED_COLUMN_NAME, RC.UPDATE_RULE, RC.DELETE_RULE FROM INFORMATION_SCHEMA.KEY_COLUMN_USAGE AS KCU LEFT JOIN INFORMATION_SCHEMA.REFERENTIAL_CONSTRAINTS AS RC ON RC.CONSTRAINT_NAME = KCU.CONSTRAINT_NAME WHERE KCU.REFERENCED_TABLE_NAME IS NOT NULL AND KCU.CONSTRAINT_SCHEMA = 'demodb' AND KCU.TABLE_NAME = 'user' => 1477906916.433 <0.37902188301086>
1477906916.437: SELECT TABLES.TABLE_TYPE AS table_type, TABLES.AUTO_INCREMENT AS auto_increment, TABLES.ENGINE AS engine, TABLES.TABLE_COLLATION AS table_collation FROM INFORMATION_SCHEMA.TABLES WHERE TABLES.TABLE_SCHEMA = 'demodb' AND TABLES.TABLE_NAME = 'user' => 1477906916.44 <0.003000020980835>

Success: Version 1.0.0 was successfully generated

C:\xampp\htdocs\dbProject>phalcon migration run

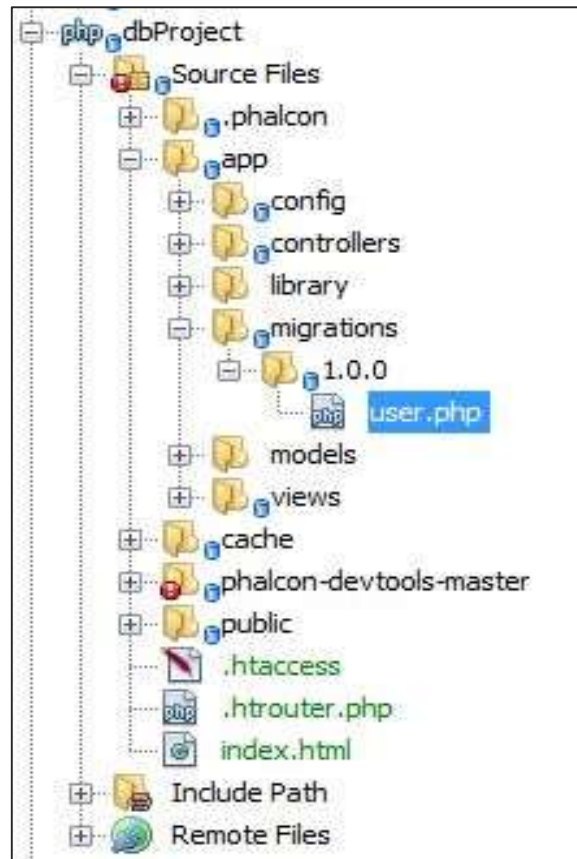
Phalcon DevTools (3.0.1)

1477907364.3537: SELECT IF(COUNT(*) > 0, 1, 0) FROM `INFORMATION_SCHEMA`.`TABLES` WHERE `TABLE_NAME` = 'user' AND `TABLE_SCHEMA` = 'demodb' => 1477907364.3547 <0.00099992752075195>
1477907364.3547: DESCRIBE `demodb`.`user` => 1477907364.3567 <0.002000093460083>
1477907364.3577: SHOW INDEXES FROM `demodb`.`user` => 1477907364.3577 <0>

Success: Version 1.0.0 was successfully migrated

```

**Step 4:** The database files migrated are stored inside the migrations directory within “app” folder.



Thus, the tables are successfully migrated.

## Understanding the Anatomy of Migrated Files

The migrated file has a unique class which extends **Phalcon\Mvc\Model\Migration** class. The Migration class in Phalcon includes the methods **up()** and **down()**. The **up()** method is used for performing migration while the down method rolls back the operation.

### Users.php

```
<?php
use Phalcon\Db\Column;
use Phalcon\Db\Index;
use Phalcon\Db\Reference;
use Phalcon\Mvc\Model\Migration;

/**
 * Class UserMigration_100
 */
class UserMigration_100 extends Migration
{
    /**
     * Define the table structure
     *
     * @return void
```

```

*/

public function morph()
{
    $this->morphTable('user', [
        'columns' => [
            new Column(
                'Id',
                [
                    'type' => Column::TYPE_INTEGER,
                    'notNull' => true,
                    'autoIncrement' => true,
                    'size' => 11,
                    'first' => true
                ]
            ),
            new Column(
                'username',
                [
                    'type' => Column::TYPE_VARCHAR,
                    'notNull' => true,
                    'size' => 40,
                    'after' => 'Id'
                ]
            ),
            new Column(
                'email',
                [
                    'type' => Column::TYPE_VARCHAR,
                    'notNull' => true,
                    'size' => 40,
                    'after' => 'username'
                ]
            ),
            new Column(
                'password',
                [
                    'type' => Column::TYPE_VARCHAR,
                    'notNull' => true,
                    'size' => 10,
                    'after' => 'email'
                ]
            )
        ],
        'indexes' => [
            new Index('PRIMARY', ['Id'], 'PRIMARY')
        ],
        'options' => [
            'TABLE_TYPE' => 'BASE TABLE',
            'AUTO_INCREMENT' => '3',
            'ENGINE' => 'InnoDB',

```

```

        'TABLE_COLLATION' => 'latin1_swedish_ci'
    ],

    ];

}

/**
 * Run the migrations
 *
 * @return void
 */
public function up()
{

}

/**
 * Reverse the migrations
 *
 * @return void
 */
public function down()
{

}

}

```

The class **UserMigration\_100** as shown in the example above includes associative array with four sections which are:

- **Columns** - Includes a set of table columns.
- **Indexes** - Includes a set of table indexes.
- **References** - Includes all the referential integrity constraints (foreign key).
- **Options** - Array with a set of table creation options.

As seen in the example above, version 1.0.0 of the database was successfully migrated. Phalcon may include and run multiple migration processes depending on how the database content is kept.

## 15. Phalcon – Cookie Management

Cookies also known as **browser cookies** are small text files stored in the browser. It saves all the information related to user identity. This information is used to validate the users once they browse through different pages.

There are two different types of Cookies:

- **Session Cookies** - These type of cookies stay on the browser and retain information until the browser is closed. As soon as the browser is opened, it will be treated as a new session for the same user.
- **Persistent Cookies** - It includes a stipulated lifespan and remains in the browser within the given lifespan. Those websites which use persistent cookies keep track of each and every user, even if the browser is closed by the user.

Let us now discuss how cookies work in Phalcon.

### Cookies in Phalcon

Phalcon uses **Phalcon\Http\Response\Cookies** as a global storage for cookies. Cookies are stored in Phalcon while sending a request to the server.

Following is the syntax for setting up a Cookie:

```
$this->cookies->set(  
    "<cookie-name>",  
    "<cookie-value>",  
    time  
);
```

Consider the following example. Using the following code, we will create cookies of the user when the user logs in to the web application.

```
<?php  
  
class UsersController extends \Phalcon\Mvc\Controller  
{  
    public function indexAction()  
    {  
        if ($this->cookies->has("login-action")) {  
            // Get the cookie  
            $loginCookie = $this->cookies->get("login-action");  
        }  
    }  
}
```



```

        // Get the cookie's value
        $value = $loginCookie->getValue();

        echo($value);
    }

    $this->cookies->set(
        "login-action",
        "abc",
        time() + 15 * 86400
    );
}
}

```

The encrypted cookies will be displayed as output.



## Description

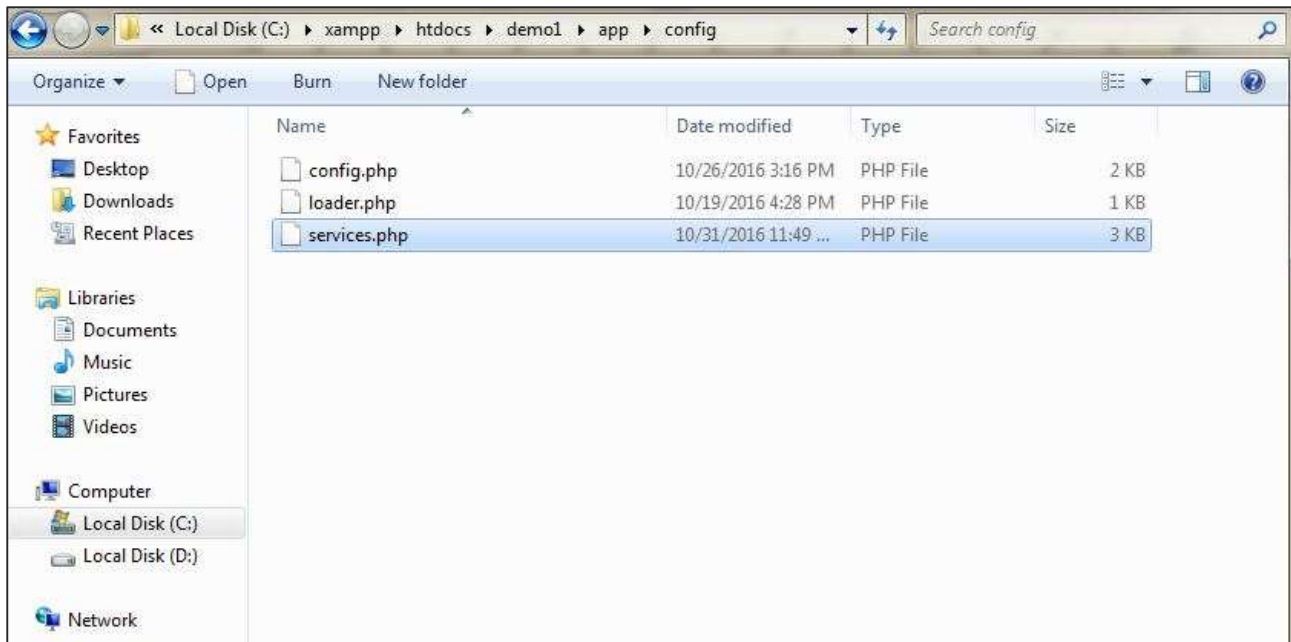
Cookie named "**loginAction**" has been created with value "**abc**".

The method "**indexAction**" checks whether the cookie exists and prints the value accordingly.

## Encryption of Cookies

Cookies in Phalcon are encrypted before being sent to the server as a request and decrypted as soon as we get an appropriate response from the server. This assures security of the authorized users.

It is always suggested to avoid storing sensitive data in cookies, despite the functionality of encryption and decryption. The configuration for encryption of cookies is included in **services.php** file.



```
/**
 * Enable encryption key for setting values of cookies
 */

$di->set(
    "cookies",
    function () {
        $cookies = new Cookies();

        $cookies->useEncryption(false);

        return $cookies;
    }
);

/**
 * Set encryption key
 */
$di->set(
    "crypt",
    function () {
        $crypt = new Crypt();
```

```
$crypt->setKey('AED@!sft56$'); // Use a unique Key!  
  
return $crypt;  
}  
);
```

**Note:**

- It is always suggested to use encryption while sending cookies to the server.
- If encryption is not used, all the internal application will be exposed to the attacker.
- It is also recommended to store small data and literals in cookies.

## 16. Phalcon – Session Management

Sessions are server-side information storage which helps in user interaction with the website or web application. Each session is uniquely defined with a session ID, which is passed to the web server whenever the browser makes an HTTP request. The session ID is paired every time with the internal database such that all stored variables are retrieved.

### Sessions in Phalcon

Phalcon uses session components which includes the wrappers to access the session data.

Following are the features in Phalcon:

- Session data can be isolated from other components on the same domain.
- According to the application needs, the session value can be changed with the help of the session adapter.

### Starting a Session in Phalcon

All the session activities are associated with the adapter files which are declared in **Services.php** file inside the **/config** folder of the web application.

```
/**
 * Start the session the first time some component requests the session service
 */
$di->setShared('session', function () {
    $session = new SessionAdapter();
    $session->start();

    return $session;
});
```

### Creating a Session

**Step 1:** Create a session controller for instantiating a session such that data can be retrieved appropriately.

```

CA: Command Prompt
C:\xampp\htdocs\demo1>phalcon controller Session
Phalcon DevTools <3.0.1>

    Success: Controller "Session" was successfully created.
C:\xampp\htdocs\demo1\app\controllers\SessionController.php
C:\xampp\htdocs\demo1>

```

**Step 2:** Create a session with a name and value.

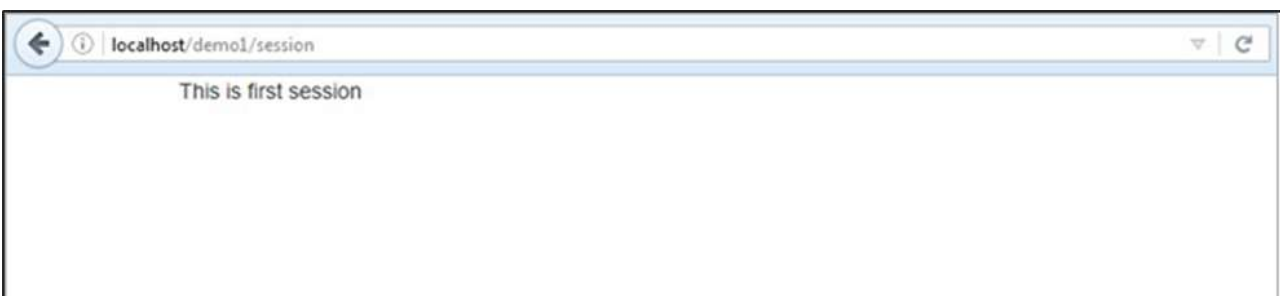
```

<?php

class SessionController extends \Phalcon\Mvc\Controller
{
    public function indexAction()
    {
        //Define a session variable
        $this->session->set("user-name", "Omkar");
        //Check if the variable is defined
        if ($this->session->has("user-name")) {
            //Retrieve its value
            $name = $this->session->get("user-name");
            echo($name);
        }
    }
}

```

The above code produces the following output.



## Removing a Session

It is possible to destroy the session or unset some variable values within the session in Phalcon. Following is the syntax to unset variable values in session.

```
$this->session->remove(<variable-name>);
```

As shown in the example above, the variable name created in the session is "**data-content**" which can be removed using the following code.

```
public function removeAction()
{
    // Remove a session variable with associated session
    $this->session->remove("data-content");
}
```

Following is the syntax to destroy the complete session.

```
$this->session->destroy();
```

## 17. Phalcon – Multi-Lingual Support

Phalcon includes a component **Phalcon\Translate** which provides multi-lingual support and it is very helpful to create web pages, which gets translated in multiple languages.

It includes an adapter which helps in binding arrays and assists in reading translation messages.

### Example

Let us create an output with the help of Translate component in Phalcon, which will help to display the output as per the language suggested.

**Step 1:** Phalcon gives freedom to every developer to organize translation strings. Consider keeping two different files namely: **en.php** (for English strings) and **fr.php** (for French strings).

The file will contain an array of key-value pair, where the keys are unique and values will differ as per the translation needed.

#### en.php

```
<?php

// app/messages/en.php
$messagesContent = [
    "bye"      => "Good Bye",
    "hi-name"  => "Hello %name%",
    "song"     => "Your favorite song is %song%",
];
```

#### fr.php

```
<?php

// app/messages/fr.php
$messagesContent = [

    "bye"      => "Au revoir",
    "hello-name" => "Bonjour %name%",
    "song"     => "Votre chanson préférée est %song%",
];
```

**Step 2:** In an application, create a **UserController** which will take parameters as to which file should be used for translation.

```
<?php
use Phalcon\Translate\Adapter\NativeArray;
class UserController extends \Phalcon\Mvc\Controller
{

    protected function getMessageTransalation()
    {
        // Ask for the best language
        // Display the output in desired language
        require "en.php";

        // Return a translation object
        return new NativeArray(
            [
                "content" => $messagesContent,
            ]
        );
    }

    public function indexAction()
    {
        $this->view->name = "Radhika";
        $this->view->song= "Ton sourire m'ensorcelle Je suis fou de toi Le désir
coule dans mes veines Guidé par ta voix";
        $this->view->t     = $this->getMessageTransalation();
    }
}
```



For the default method, two parameters are taken, first is name and the second is the favorite song of the user. Later, the function **getMessageTranslation** is being called which returns the desired output.

For now, we want the output in English.

**Step 3:** The associated **code view demo\app\views\User\index.volt** will include the following code:

```
<p><?php echo $t->_("hello-name", ["name" => $name]); ?></p>
<p><?php echo $t->_("song", ["song" => $song]); ?></p>
```



If we want the complete output to be displayed in French, we only need to change the file name.

```
require "fr.php";
```

Following is the output in French.



## 18. Phalcon – Asset Management

Assets are all about the additional components apart from the existing framework in Phalcon. Phalcon has an asset manager which helps to manage all the asset components like CSS or JS files.

The common methods used are:

Method	Importance
__construct(variable \$options)	Initializes the component Phalcon\Assets\Manager
addCss(string \$path, variable \$local, variable \$filter, variable \$attributes)	Adds a CSS resource from the 'css' collection to a particular view
addJs(string \$path, variable \$local, variable \$filter, variable \$attributes)	Adds a JavaScript resource to the 'js' collection

### Example

Consider the sample project of Phalcon “**vokuro**” which is the best illustration for adding **css** files. It will include assets/Manager for invoking all the **css** files.

The default controller for the project will invoke all the **css** files.

```
<?php
namespace Vokuro\Controllers;
use Phalcon\Assets\Manager;

/**
 * Display the default index page.
 */
class IndexController extends ControllerBase
{

    /**
     * Default action. Set the public layout (layouts/public.volt)
     */
    public function indexAction()
    {
```

```

        $this->assets->addCss("public/style.css");

        $this->view->setVar('logged_in', is_array($this->auth->getIdentity()));
        $this->view->setTemplateBefore('public');
    }
}

```

## Style.css

```

div.remember {
    margin-top: 7px;
    color: #969696;
}

div.remember label {
    padding-top: 15px;
}

div.forgot {
    margin-top: 7px;
    color: #dadada;
}

footer {
    background: url("../img/feature-gradient.png") no-repeat scroll center 100%
white;
    color: #B7B7B7;
    font-size: 12px;
    padding: 30px 0;
    text-align: center;
}

footer a {
    margin-left: 10px;
    margin-right: 10px;
}

```

```

table.signup td {
    padding: 10px;
}

table.signup .alert {
    margin-bottom: 0;
    margin-top: 3px;
}

table.perms select {
    margin-top: 5px;
    margin-right: 10px;
}

table.perms label {
    margin-right: 10px;
}

div.main-container {
    min-height: 450px;
}

```

The assets will be managed inside views, which will display **css** files as an output.

## Index.volt

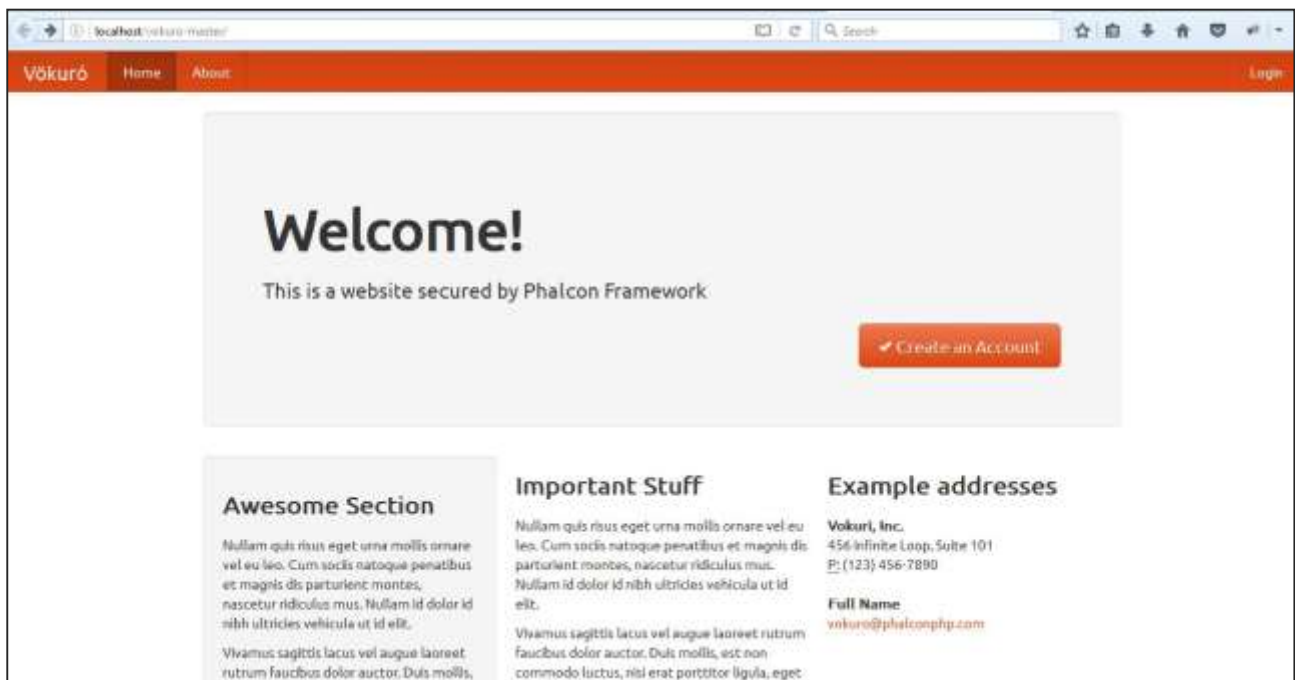
```

{{ content() }}
{{ assets.outputCss() }}
<header class="jumbotron subhead" id="overview">
    <div class="hero-unit">
        <h1>Welcome!</h1>
        <p class="lead">This is a website secured by Phalcon Framework</p>
        <div align="right">
            {{ link_to('session/signup', '<i class="icon-ok icon-white"></i> Create an
Account', 'class': 'btn btn-primary btn-large') }}
        </div>
    </div>
</header>

```

**Output:**

It will produce the following output:



## 19. Phalcon – Working with Forms

Forms are used in all web applications to accept inputs from the user as request. The data is accepted as an input, then manipulated and saved in the database or any other operation is being performed.

Phalcon includes a component named **Phalcon\Forms** which helps in the creation and maintenance of forms.

Consider the example of Blog-tutorial, which we created in the previous chapters. It includes a form which is used to create a new category.

```
<?php echo \Phalcon\Tag::form(array("categories/create", "autocomplete" => "off")) ?>

<table width="100%">
    <tr>
        <td align="left"><?php echo \Phalcon\Tag::linkTo(array("categories", "Go
Back", "class" => "btn")) ?></td>
        <td align="right"><?php echo \Phalcon\Tag::submitButton(array("Save",
"class" => "btn")) ?></td>
    <tr>
    </table>

<?php echo $this->getContent(); ?>

<div align="center">
    <h1>Create categories</h1>
</div>

<table align="center">
    <tr>
        <td align="right">
            <label for="name">Name</label>
        </td>
        <td align="left">
            <?php echo \Phalcon\Tag::textField(array("name", "size" => 30)) ?>
        </td>
    </tr>
    <tr>
```

```

        <td align="right">
            <label for="slug">Slug</label>
        </td>
        <td align="left">
            <?php echo \Phalcon\Tag::textField(array("slug", "size" => 30)) ?>
        </td>
    </tr>
</table>
</form>

```

**Output:** It will produce the following output:

The input fields of form are rendered with the help of **Phalcon/tag** component. Each element in the form can be rendered as per the requirement of the developer.

Following is the syntax for rendering value.

```
echo $form->render(element-name)
```

### Validation:

Once the values are rendered in the controller, the values will be entered in the database with the help of models. Phalcon forms are integrated with the validation component to offer instant validation. Built-in or custom validators can be set to each element.

```

<?php

use Phalcon\Forms\Element\Text;
use Phalcon\Validation\Validator\PresenceOf;
use Phalcon\Validation\Validator\StringLength;

$name = new Text(
    "Name"
);

```

```
$name->addValidator(  
    new PresenceOf(  
        [  
            "message" => "name is required",  
        ]  
    )  
);  
$form->add($name);
```

**Output:** It will produce the following output:



The screenshot shows a web browser window with a navigation bar at the top containing links: "Blog Collection", "Home Page", "Posts", "Categories", and "Log out". Below the navigation bar, there is a "Go Back" button on the left and a "Save" button on the right. A red horizontal banner displays the message "name is required". Below this banner, the heading "Create categories" is centered. Under the heading, there are two input fields: "Name" and "Slug". The "Name" field is empty, and the "Slug" field contains the text "xxx".



## 20. Phalcon – Object Document Mapper

Before starting with the concepts of Object Relational Mapper (ORM) and **Object Document Mapper (ODM)**, it is important to understand the difference between SQL and NoSQL databases.

The following table highlights the differences between SQL and NoSQL:

SQL	NoSQL
They are also termed as Relational Databases (RDBMS)	They are called as non-relational or distributed database
The structure of database is constituted as tables and views	It consists of document based and graph databases
It includes a predefined schema	It has a dynamic schema
It is very powerful for defining and manipulating data	It is powerful in maintaining data as collection of documents

Phalcon has the ability to map with SQL and NoSQL databases. This is achieved with the help of Object Document Mapper (ODM) for NoSQL database and Object Relational Mapper (ORM) for SQL database.

In Phalcon, ORM concept comprises of creating a model associated with the given table-name as we have seen in the previous chapters. It follows all the referential integrity constraints.

### Object Document Mapper (ODM)

It is an object associated with NoSQL database. As the name suggests it maps the document related module. Phalcon uses it to map with databases like MongoDB

### Example

**Step 1:** Create a database of MongoDB named **"test"**. We will use this database to map with and get the appropriate response.

```

C:\Users\SIF 2551>mongod
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\SIF 2551>mongod
2016-11-07T17:36:56.034+0530 I CONTROL [main] Hotfix KB2731284 or later update
is not installed, will zero-out data files
2016-11-07T17:36:56.035+0530 I CONTROL [initandlisten] MongoDB starting : pid=6
996 port=27017 dbpath=C:\data\db\ 64-bit host=SIF2551-PC
2016-11-07T17:36:56.035+0530 I CONTROL [initandlisten] targetMinOS: Windows 7/W
indows Server 2008 R2
2016-11-07T17:36:56.035+0530 I CONTROL [initandlisten] db version v3.2.10
2016-11-07T17:36:56.035+0530 I CONTROL [initandlisten] git version: 79d9b3ab5ce
20f51c272b4411202710a082d0317
2016-11-07T17:36:56.035+0530 I CONTROL [initandlisten] OpenSSL version: OpenSSL
1.0.1t-fips 3 May 2016
2016-11-07T17:36:56.035+0530 I CONTROL [initandlisten] allocator: tcmalloc
2016-11-07T17:36:56.035+0530 I CONTROL [initandlisten] modules: none
2016-11-07T17:36:56.035+0530 I CONTROL [initandlisten] build environment:
2016-11-07T17:36:56.035+0530 I CONTROL [initandlisten] distmod: 2008plus-ss
l
2016-11-07T17:36:56.036+0530 I CONTROL [initandlisten] distarch: x86_64
2016-11-07T17:36:56.036+0530 I CONTROL [initandlisten] target_arch: x86_64
2016-11-07T17:36:56.036+0530 I CONTROL [initandlisten] options: {}
2016-11-07T17:36:56.036+0530 I - [initandlisten] Detected data files in C
:\data\db\ created by the 'wiredTiger' storage engine, so setting the active sto
rage engine to 'wiredTiger'.
2016-11-07T17:36:56.037+0530 I STORAGE [initandlisten] wiredtiger_open config:
create,cache_size=4G,session_max=20000,eviction=(threads_max=4),config_base=false,statistics=(fast),log=(enabled=true,archive=true,path=journal,compressor=snappy),file_manager=(close_idle_time=100000),checkpoint=(wait=60,log_size=2GB),statistics_log=(wait=0),
2016-11-07T17:36:56.883+0530 I NETWORK [HostnameCanonicalizationWorker] Startin
g hostname canonicalization worker
2016-11-07T17:36:56.883+0530 I FTDC [initandlisten] Initializing full-time d
iagnostic data capture with directory 'C:\data\db\diagnostic.data'
2016-11-07T17:36:56.888+0530 I NETWORK [initandlisten] waiting for connections
on port 27017

```

```

C:\Users\SIF 2551>mongo
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\SIF 2551>mongo
2016-11-08T09:22:41.071+0530 I CONTROL [main] Hotfix KB2731284 or later update
is not installed, will zero-out data files
MongoDB shell version: 3.2.10
connecting to: test
> use test;
switched to db test
>

```

**Step 2:** Check for the inserted records in the database. The command associated with it is:

```
db.collection.find()
```

```

C:\Users\SIF 2551>mongo
2016-11-08T09:22:41.071+0530 I CONTROL [main] Hotfix KB2731284 or later update
is not installed, will zero-out data files
MongoDB shell version: 3.2.10
connecting to: test
> use test;
switched to db test
> db.test.find()
{ "_id" : ObjectId("5819ab6cfce9c70ac6087821"), "data" : [ { "country_id" : "AFG",
"country_name" : "Afghanistan" }, { "country_id" : "ALB", "country_name" : "A
lbania" }, { "country_id" : "DZA", "country_name" : "Algeria" }, { "country_id"
: "AND", "country_name" : "Andorra" }, { "country_id" : "AGO", "country_name" :
"Angola" }, { "country_id" : "ATG", "country_name" : "Antigua & Barbuda" }, { "c
ountry_id" : "ARG", "country_name" : "Argentina" }, { "country_id" : "ARM", "cou
untry_name" : "Armenia" }, { "country_id" : "AUS", "country_name" : "Australia" },
{ "country_id" : "AUT", "country_name" : "Austria" }, { "country_id" : "AZE",
"country_name" : "Azerbaijan" }, { "country_id" : "BHR", "country_name" : "Bahra
in" }, { "country_id" : "BGD", "country_name" : "Bangladesh" }, { "country_id" :
"BRB", "country_name" : "Barbados" }, { "country_id" : "BLR", "country_name" :
"Belarus" }, { "country_id" : "BEL", "country_name" : "Belgium" }, { "country_id
" : "BLZ", "country_name" : "Belize" }, { "country_id" : "BEN", "country_name" :
"Benin" }, { "country_id" : "BTN", "country_name" : "Bhutan" }, { "country_id"

```

It is observed that every document is mapped with ObjectId which is a feature of ODM. The value of **ObjectId** is unique and later used to fetch all the data stored with respect to that particular Id.

**Step 3:** Set up model for the database created. A model is a class which extends **Phalcon\Mvc\Collection**. **Test.php** model will include the following code.

```

<?php
use Phalcon\Mvc\Collection;

class Test extends Collection
{
    public function initialize()
    {
        $this->setSource("test");
    }
}

```

**Step 4:** Configure the project including database connectivity in **services.php**

```
// Simple database connection to localhost
$di->set(
    "mongo",
    function () {
        $mongo = new MongoClient();

        return $mongo->selectDB("test");
    },
    true
);
// Connecting to a domain socket, falling back to localhost connection
$di->set(
    "mongo",
    function () {
        $mongo = new MongoClient(
            "mongodb:///tmp/mongodb-27017.sock,localhost:27017"
        );

        return $mongo->selectDB("test");
    },
    true
);
```

**Step 5:** Print the values with respect to **ObjectId** with the help of **TestController.php**

```
<?php

use Phalcon\Mvc\Controller;

class TestController extends Controller
{
    public function index()
    {
        // Find record with _id = "5087358f2d42b8c3d15ec4e2"
        $test = Test::findById("5819ab6cfce9c70ac6087821");
    }
}
```



```

    echo $test->data;
}
}

```

The output will display data which matches the objectId. If the objectId is not matched as per the records in the documents, then the appropriate output will not be displayed as the number of records is fetched.



```

{
  "type": "FeatureCollection",
  "features": [
    {
      "type": "Feature", "properties": {
        "name": "Italy", "SOVEREIGN": "Italy", "iso_a3": "ITA", "iso_a2": "IT", "UN_CODE": 388, "WHO_CODE": 4180, "WHO_REGION": "EURO", "WHO_STATUS": "Member state", "REAL_VALUE": 0
      }, "geometry": {
        "type": "MultiPolygon", "coordinates": [
          [
            [
              [13.95858, 46.31], [13.44, 45.64], [12.95, 45.64], [12.48, 44.81], [13.9, 42.59], [16.1, 41.67], [16.54, 41.39], [17.0, 41.39], [18.57, 40.21], [18.35, 39.82], [17.82, 40.17], [16.9, 39.88], [17.29, 38.97], [16.34, 38.09], [15.76, 37.91], [15.58, 38.88], [15.14, 40.19], [12.47, 41.41], [19.27, 43.52], [9.35, 43.83], [8.42, 43.53], [7.33, 43.53], [7.07, 44.72], [7.54, 45.62], [7.54, 45.93], [8.45, 45.92], [8.45, 46.23], [8.92, 45.92], [9.61, 46.29], [10.33, 46.22], [10.35, 46.94], [12.21, 46.82], [12.97, 46.57], [14.03, 46.51], [13.95, 46.31], [12.74, 38.14], [15.51, 38.22], [15.21, 36.08], [14.96, 36.58], [12.43, 37.67], [12.74, 38.14]
            ]
          ]
        ]
      }
    },
    {
      "type": "Feature", "properties": {
        "name": "Jammu and Kashmir", "SOVEREIGN": "Jammu and Kashmir", "iso_a3": "J&K", "iso_a2": "J&K", "UN_CODE": null, "WHO_CODE": null, "WHO_REGION": null, "WHO_STATUS": null, "REAL_VALUE": null
      }, "geometry": {
        "type": "MultiPolygon", "coordinates": [
          [
            [
              [72.22, 36.32], [75.53, 36.74], [78.7, 34.07], [79.32, 34.05], [78.97, 33.83], [79.25, 33.88], [79.25, 32.74], [79.04, 32.39], [70.63, 33.7], [77.32, 33.15], [74.7, 32.88], [72.22, 36.32]
            ]
          ]
        ]
      }
    },
    {
      "type": "Feature", "properties": {
        "name": "Israel", "SOVEREIGN": "Israel", "iso_a3": "ISR", "iso_a2": "IL", "UN_CODE": 376, "WHO_CODE": 3150, "WHO_REGION": "EURO", "WHO_STATUS": "Member state", "REAL_VALUE": 0
      }, "geometry": {
        "type": "MultiPolygon", "coordinates": [
          [
            [
              [35.89, 33.88], [35.47, 33.87], [35.68, 33.31], [35.6, 32.83], [35.63, 32.67], [35.56, 32.38], [35.15, 32.53], [35.01, 32.27], [34.99, 32.05], [34.97, 31.84], [35.11, 31.83], [35.21, 31.76], [34.94, 31.58], [34.87, 31.38], [34.93, 31.34], [35.23, 31.37], [35.46, 31.49], [35.46, 31.39], [35.47, 31.18], [35.2, 30.54], [35.14, 30.17], [34.95, 29.52], [34.89, 29.48], [34.82, 30.38], [34.28, 31.2], [34.56, 31.54], [34.49, 31.8], [34.9, 32.73], [35.09, 33.88]
            ]
          ]
        ]
      }
    },
    {
      "type": "Feature", "properties": {
        "name": "Saudi Arabia", "SOVEREIGN": "Saudi Arabia", "iso_a3": "SAU", "iso_a2": "SA", "UN_CODE": 682, "WHO_CODE": 3340, "WHO_REGION": "EMRO", "WHO_STATUS": "Member state", "REAL_VALUE": 1
      }, "geometry": {
        "type": "MultiPolygon", "coordinates": [
          [
            [
              [37.44, 29.97], [38.02, 30.49], [37.04, 31.51], [39.18, 32.09], [40.46, 31.98], [44.75, 29.22], [46.53, 29.06], [47.4, 28.97], [48.42, 28.53], [49.1, 27.57], [50.21, 26.71], [50.08, 25.8], [50.75, 25.09], [50.86, 24.78], [51.43, 24.62], [51.49, 24.36], [51.54, 24.15], [52.52, 22.93], [55.13, 22.64], [55.73, 22.12], [55.06, 20.0], [51.99, 19.01], [48.19, 18.33], [47.16, 17.1], [45.21, 17.12], [43.32, 17.47], [42.89, 16.3], [40.09, 20.18], [39.09, 21.12], [39.16, 22.37], [38.49, 23.77], [37.58, 24.29], [35.1, 28.17], [34.61, 28.08], [34.89, 29.48], [36.09, 29.19], [36.65, 29.72], [37.44, 29.97]
            ]
          ]
        ]
      }
    },
    {
      "type": "Feature", "properties": {
        "name": "Jordan", "SOVEREIGN": "Jordan", "iso_a3": "JOR", "iso_a2": "JO", "UN_CODE": 408, "WHO_CODE": 3170, "WHO_REGION": "EMRO", "WHO_STATUS": "Member state", "REAL_VALUE": 1
      }, "geometry": {
        "type": "MultiPolygon", "coordinates": [
          [
            [
              [36.83, 32.28], [36.77, 33.4], [39.18, 32.09], [37.04, 31.51], [38.02, 30.49], [37.44, 29.97], [36.85, 29.72], [36.86, 29.19], [34.89, 29.48], [34.95, 29.52], [35.14, 30.17], [35.2, 30.54], [35.47, 31.18], [35.46, 31.39], [35.48, 31.49], [35.55, 31.78], [35.5, 32.11], [35.56, 32.38], [35.63, 32.67], [35.89, 32.73], [36.1, 32.51], [36.83, 32.28]
            ]
          ]
        ]
      }
    },
    {
      "type": "Feature", "properties": {
        "name": "United Arab Emirates", "SOVEREIGN": "United Arab Emirates", "iso_a3": "ARE", "iso_a2": "AE", "UN_CODE": 784, "WHO_CODE": 3405, "WHO_REGION": "EMRO", "WHO_STATUS": "Member state", "REAL_VALUE": 0
      }, "geometry": {
        "type": "MultiPolygon", "coordinates": [
          [
            [
              [51.49, 24.36], [53.88, 24.84], [55.54, 25.54], [56.06, 26.03], [56.33, 25.83], [56.38, 25.47], [55.8, 24.87], [55.92, 24.87], [55.56, 23.73], [55.13, 22.64], [52.52, 22.93], [51.54, 24.15], [51.49, 24.36]
            ]
          ]
        ]
      }
    },
    {
      "type": "Feature", "properties": {
        "name": "Yemen", "SOVEREIGN": "Yemen", "iso_a3": "YEM", "iso_a2": "YE", "UN_CODE": 887, "WHO_CODE": 3420, "WHO_REGION": "EMRO", "WHO_STATUS": "Member state", "REAL_VALUE": 15
      }, "geometry": {
        "type": "MultiPolygon", "coordinates": [
          [
            [
              [42.89, 16.3], [43.32, 17.47], [45.21, 17.12], [47.16, 17.1], [48.19, 18.33], [51.99, 19.01], [52.38, 18.26], [53.02, 16.73], [52.13, 15.64], [49.14, 14.33], [45.28, 13.02], [44.43, 12.57], [43.46, 12.66], [43.22, 13.24], [43.26, 13.73], [42.72, 15.22], [42.89, 16.3]
            ]
          ]
        ]
      }
    },
    {
      "type": "Feature", "properties": {
        "name": "India", "SOVEREIGN": "India", "iso_a3": "IND", "iso_a2": "IN", "UN_CODE": 356, "WHO_CODE": 3108, "WHO_REGION": "SEARO", "WHO_STATUS": "Member state", "REAL_VALUE": 534
      }, "geometry": {
        "type": "MultiPolygon", "coordinates": [
          [
            [
              [75.41, 32.7], [77.54, 33.14], [78.48, 32.67], [78.83, 32.5], [78.53, 32.47], [78.59, 32.35], [78.85, 32.36], [79.59, 31.34], [79.85, 31.34], [80.96, 30.54], [80.08, 28.84], [83.27, 27.33], [84.58, 27.43], [84.79, 27.02], [86.13, 26.48], [88.03, 26.4], [88.04, 26.09], [88.24, 27.92], [88.78, 27.89], [89.78, 28.3], [89.07, 27.56], [88.8, 27.02], [89.68, 26.66], [90.91, 26.80], [92.09, 26.84], [91.74, 27.87], [92.74, 28.95], [94.33, 29.2], [94.73, 29.3], [96.34, 29.36], [96.05, 28.38], [97.38, 28.23], [96.96, 27.33], [95.21, 26.63], [94.18, 23.88], [93.28, 24.06], [93.18, 22.16], [92.64, 21.85], [92.25, 23.78], [91.58, 22.95], [91.17, 24.03], [92.82, 24.19], [92.43, 25.11], [89.88, 25.32], [89.63, 26.21], [88.52, 26.43], [87.93, 25.89], [88.67, 25.35], [88.08, 24.62], [88.05, 24.14], [88.91, 23.18], [89.16, 21.67], [87.16, 21.54], [87.0, 20.69], [86.35, 20.07], [84.12, 18.3], [82.35, 17.01], [82.35, 16.57], [83.32, 16.32], [80.6, 15.96], [80.24, 14.62], [79.82, 18.3], [79.31, 9.83], [79.16, 9.29], [78.15, 8.57], [77.48, 7.9], [76.68, 8.7], [75.22, 11.63], [73.47, 16.85], [72.67, 18.76], [72.93, 30.79], [72.42, 22.36], [72.06, 21.1], [70.77, 20.66], [68.78, 22.31], [68.84, 22.55], [70.56, 23.03], [69.3, 22.72], [67.73,
            ]
          ]
        ]
      }
    }
  ]
}

```

## 21. Phalcon – Security Features

Phalcon provides security features with the help of Security component, which helps in performing certain tasks like password hashing and **Cross-Site Request Forgery** (CSRF).

### Hashing Password

**Hashing** can be defined as the process of converting a fixed length bit string into a specified length in such a way that it cannot be reversed. Any change in the input string will change the value of hashed data.

Decryption of hashed data takes place by taking the value entered by the user as input and comparing hash form of the same. Usually for any web-based applications, storing passwords as plain text is a bad practice. It is prone to third-party attacks as those who have access to the database can easily procure passwords for any user.

Phalcon provides an easy way to store passwords in encrypted form which follows an algorithm like **md5**, **base64** or **sh1**.

As seen in the previous chapters, where we created a project for blogs. The login screen accepts input as username and password for the user. To receive the passwords from the user and decrypt it in a particular form, the following code snippet is used.

The decrypted password is then matched with the password accepted as input from the user. If the value matches, the user can successfully log in to web application else an error message is displayed.

```
<?php

class UsersController extends Phalcon\Mvc\Controller
{

    public function indexAction()
    {

    }

    public function registerUser()
    {
        $user = new Users();

        $login    = $this->request->getPost("login");
        $password = $this->request->getPost("password");
```

```

        $user->login = $login;

        // Store the hashed password
        $user->password = $this->security->sh1($password);

        $user->save();
    }

    public function loginAction()
    {

        if ($this->request->isPost()) {

            $user = Users::findFirst(array(
                'login = :login: and password = :password:',
                'bind' => array(
                    'login' => $this->request->getPost("login"),
                    'password' => sha1($this->request->getPost("password"))
                )
            ));

            if ($user === false){
                $this->flash->error("Incorrect credentials");
                return $this->dispatcher->forward(array(
                    'controller' => 'users',
                    'action' => 'index'
                ));
            }

            $this->session->set('auth', $user->id);

            $this->flash->success("You've been successfully logged in");
        }

        return $this->dispatcher->forward(array(

```

```

        'controller' => 'posts',
        'action' => 'index'
    ));
}

public function logoutAction()
{
    $this->session->remove('auth');
    return $this->dispatcher->forward(array(
        'controller' => 'posts',
        'action' => 'index'
    ));
}
}

```

The passwords stored in the database are in an encrypted format of **sh1** algorithm.

+ Options					id	login	password
<input type="checkbox"/>		Edit		Copy		Delete	1 Radhika f1404e8a9e9d28a68281a46ac8f6f9bd7ba646ba
<input type="checkbox"/>		Edit		Copy		Delete	2 Omkar a2c065e66cf54b22828dacb559ea340ba9bc8fd5
<input type="checkbox"/>		Edit		Copy		Delete	3 admin d033e22ae348aeb5660fc2140aec35850c4da997

Once the user enters an appropriate username and password, the user can access the system, else an error message is displayed as a validation.

[Blog Collection](#)
[Home Page](#)
[Log In](#)

incorrect credentials

### Log In

Username/Email

Password



## Cross-Site Request Forgery (CSRF)

It is an attack which forces authenticated users of web application to perform certain unwanted actions. Forms which accept inputs from the users are vulnerable to this attack. Phalcon tries to prevent this attack by protecting the data which is sent through forms outside the application.

The data in each form is secured with the help of token generation. The token generated is random and it is matched with the token to which we are sending the form data (mostly outside the web application through POST method).

### Code:

```
<?php echo Tag::form('session/login') ?>

<!-- Login and password inputs ... -->

<input type="hidden" name="<?php echo $this->security->getTokenKey() ?>"
       value="<?php echo $this->security->getToken() ?>"/>

</form>
```

**Note:** It is important to use session adapter while sending tokens of form, as all the data will be maintained in the session.

Include session adapter in **services.php** using the following code.

```
/**
 * Start the session the first time some component request the session service
 */
$di->setShared('session', function () {
    $session = new SessionAdapter();
    $session->start();

    return $session;
});
```