**به نام یگانه برنامه‌نویس هستی**

مستندات

سالوس

نسخه ۱٬۳

# فهرست مطالب

[فهرست مطالب 2](#_Toc448053253)

[مقدمه 3](#_Toc448053254)

[مزایای فریم‌ورک سالوس 4](#_Toc448053255)

[بررسی اجمالی 5](#_Toc448053256)

[اصول اولیه 6](#_Toc448053257)

[بررسی اجمالی لایه‌ها 6](#_Toc448053258)

[مدل 7](#_Toc448053259)

[ویو 7](#_Toc448053260)

[کنترلر 8](#_Toc448053261)

[دیسپلی 8](#_Toc448053262)

[افزودن ماژول جدید 9](#_Toc448053263)

[ماژول مستقل 9](#_Toc448053264)

[ماژول وابسته 12](#_Toc448053265)

[بررسی کامل فایل‌های هر ماژول 13](#_Toc448053266)

[کنترلر 13](#_Toc448053267)

[دیسپلی 14](#_Toc448053268)

[مدل 15](#_Toc448053269)

[ویو 16](#_Toc448053270)

[کنترل صفحه 17](#_Toc448053271)

[روتر 17](#_Toc448053272)

[قوانین مقادیر ورودی روتر 17](#_Toc448053273)

[انتقال دهنده صفحه 20](#_Toc448053274)

# مقدمه

اگرچه زبان پی‌اچ‌پی یکی از قدرتمندترین زبان های برنامه‌نویسی وب می‌باشد و پراستفاده‌ترین آنها در جهان نیز بشمار می‌آید، اما باید گفت برای حرکت به سمت برنامه‌های تجاری بزرگ و مقیاس پذیر به‌تنهایی جواب‌گو نیست. منظور از جوابگو نبودن ضعف پی‌اچ‌پی نیست، استفاده از کدهای استاندارد برای توسعه و پیاده‌سازی برنامه‌های بزرگ بسیار دشوار و وقت‌گیر است( این مسئله در سایر زبان‌ها نیز صادق است). به‌همین دلیل برای بالابردن سطح برنامه‌نویسی و ایجاد برنامه‌های با پویایی بالا نیاز به ابزارها و زیرساخت‌هایی برای این منظور در این زبان‌هاست که یکی از این ابزارها فریم‌ورک‌ها(Framework) هستند.

به‌زبانی ساده و در عین حال فنی، یک فریم ورک (Framework) یک پلت فرم (Platform) اولیه است که به ما امکان توسعه یک برنامه مبتنی بر وب را می دهد. به بیانی دیگر فریم ورک زیرساخت و ساختار برنامه را فراهم می آورد. استفاده از چنین پلتفرمی مزایای بسیاری را بدنبال دارد. به‌عنوان مثال زمان توسعه برنامه را کاهش داده و امکان ساخت سریع یک برنامه مبتنی بر وب را می دهد. پلت فرم ها با ارائه زیر ساختی مناسب از ایجاد کدهای تکراری توسط توسعه‌دهندگان جلوگیری می کنند و امکان استفاده مجدد از کد ها را در برنامه فراهم می‌آورند. یکی از مشکلاتی که در توسعه برنامه ها گریبان گیر برنامه نویسان است این‌است که بدون وجود یک فریم ورک، آنها مجبور هستند در جای جای برنامه خود کدهای تکراری را قرار دهند که این امر روال توسعه نرم افزار و نگهداری آن را با مشکلات عدیده ای روبرو می کند. از دیگر مزایای استفاده از فریم ورک‌ها می‌توان به اتصال به پایگاه داده اشاره کرد. در شرایط عادی شما نیاز دارید اتصال به پایگاه داده را از ابتدا در برنامه پیاده سازی کنید حال آنکه فریم ورک با آماده کردن زیرساختی مناسب کمک بسیار زیادی در این رابطه به شما می‌کند و کار با دیتابیس را به مراتب ساده‌تر از شرایط عادی می نماید.

فریم‌ورک‌ها به یک چارچوب از پیش تعریف شده توسط برنامه‌نویسان حرفه‌ای گفته می‌شود که این کمک را به سایر برنامه نویسان می‌کند تا چرخ را از نو اختراع نکنند!

از مزایای استفاده از فریم ورک‌ها می‌توان به سرعت بخشیدن به طراحی وب اپلیکیشن، حصول اطمینان از دیده شدن مسائل امنیتی در اسکریپت ها، کاهش باگ در کدها و … اشاره کرد. در عین حال، استفاده از یک فریم ورک، محدودیت‌هایی را برای طراح به ارمغان می‌آورد که شاید به همین دلیل ما گزینه‌های موجود را انتخاب نکردیم.

سالوس توسط ارمایل[[1]](#footnote-1) با پشتوانه بیش از یک دهه تجربه در وب و با نگاه برنامه به عنوان سرویس[[2]](#footnote-2) و برای سرعت‌بخشیدن به ایجاد سرویس‌های مدرن و بدور از مشکلات سنتی ساخت نرم‌افزارها طراحی شده و در حال توسعه است.

## مزایای فریم‌ورک سالوس

* بروزرسانی و انطباق با آخرین نسخه پی‌اچ‌پی - در حال حاضر نسخه ۷
* ایجاد خودکار واسط برنامه‌نویسی استاندارد برای ایجاد یک وب‌سرویس پویا[[3]](#footnote-3)
* استفاده از معماری چندلایه بر پایه ساختار ام‌وی‌سی[[4]](#footnote-4)
* سیستم یکپارچه فرم‌ساز بر اساس داده‌های دیتابیس
* افزایش امنیت ورودی‌های کاربران
* ارتباط هوشمند با دیتابیس
* ابزارهای کنترل و مدیریت عملیات کاربران
* سرعت بالا در رسیدگی به درخواست‌های کاربران
* ساختار بر پایه الگو برای ساخت صفحات پویا
* سامانه اعتبارسنجی خودکار ورودی‌ها
* سیستم هوشمند نمایش خطاها
* نحوه نگارش ساده و زیبا

## بررسی اجمالی

برای راه اندازی پروژه با زبان پی‌اچ‌پی نیاز به یک سری نیازهای اولیه داریم که این نیازها در قالب هسته پیش رو برطرف گردیده است. به جهت حجم پروژه و نیاز برای توسعه و نگهداری آن، سالوس به صورت چند لایه بر مبنای الگوی معماری ام‌وی‌سی طراحی گردیده است. این الگو منطق برنامه را از رابط کاربری به طور کامل جدا ساخته و به توسعه دهندگان این امکان را می‌دهد که توسعه، تست و نگهداری نرم‌افزار را در واحدهای مستقل انجام دهند.

هم‌چنین با توجه به نیاز پیش بینی شده برای ارتباط سیستم با خارج از آن، سیستم بر مبنای واسط برنامه‌نویسی استاندارد بنا شده است. بدین معنا که هر گونه عملیات انجام شده توسط سیستم، قابلیت انجام از طریق خارج از آن را با داشتن سطح دسترسی‌های مورد نیاز دارد.

علاوه بر این به دلیل سبک طراحی صورت گرفته، سیستم قابلیت افزودن کلاس‌های مورد نیاز توسط برنامه نویسان یا جایگزینی کلاس‌های موجود توسط آنان را در بخش‌های مورد نیاز داراست.

# اصول اولیه

در تعریف ما، کلاس به معنی یک آدرس بعد از نام دامنه در مرورگر است. برای مثال آدرس زیر را در نظر بگیرید که اکانت به عنوان یک کلاس شناخته می‌شود.

test.com/account

در ادامه آدرس بعد از کلاس به عنوان ماژول شناخته می‌شود. آدرس زیر را در نظر بگیرید که لاگین به عنوان ماژول شناخته می‌شود.

test.com/account/login

در این مثال لاگین به عنوان یک ماژول در نظر گرفته می‌شود که می‌تواند به صورت مستقل یا وابسته باشد. تمام ماژول‌ها به صورت پیش‌فرض باید در پوشه content موجود در پروژه قرار گیرند. به همین دلیل از این پس آدرس‌ها بر اساس این روش نمایش داده خواهند شد.

ساختار پوشه بندی پیش فرض با وجود ماژول اکانت که شامل ماژول لاگین است، به شکل زیر خواهد بود.

Content/home

Content/account/home

Content/account/login

پوشه home همان index است، یعنی در صورتی که هیچ چیزی وجود نداشته باشد، کدهای موجود در این پوشه خوانده می‌شود. برای ماژول اکانت پوشه account/home و برای ماژول لاگین پوشه account/login خوانده می‌شود. در صورتی که فایل‌های مورد نیاز در هر یک از ماژول‌ها وجود نداشته باشد، به ترتیب به یک پوشه بالاتر مراجعه می‌شود و در این مدل پوشه home به عنوان آخرین آدرس مورد بررسی قرار می‌گیرد.

## بررسی اجمالی لایه‌ها

هر ماژول می تواند باید شامل لایه‌های زیر باشد که در ادامه به تشریح آن می‌پردازیم.

model.php

view.php

controller.php

display.html

### **مدل**

این لایه تمامی پروسه‌ها و عملگرهای سمت سرور از قبیل ایجاد و حذف فایل، فولدر، اتصال به پایگاه داده ها. تغییر سایز عکس‌ها و ... را کنترل می‌کند. توابعی که در مدل ساخته می‌شوند را می‌توان در ویو و کنترلر فراخوانی کرد. اما بهترین راه فراخوانی استفاده از روترهاست.

مدل لایه‌ای برای ارتباط با دیتابیس و ذخیره سازی داده‌های دائمی است. نکته مهم این است که این بخش از اتفاقاتی که در بخش های کنترلر و ویو می‌افتد هیچ اطلاعاتی ندارد. این لایه به دنبال پاسخ از دیگر لایه‌ها نیست و تنها هدف آن ذخیره سازی دائمی داده‌ها یا آماده‌سازی آنها برای انتقال به لایه‌های دیگر است.

هم‌چنین این لایه تنها زمانی فراخوانی می‌شود که دستوری از نوع هدر شامل post, put, add, delete به صفحه ارسال شود و یا نیاز داشته باشیم در ویو از کوئری خاصی استفاده کنیم، قابل توجه است که اگر قصد هدایت به این لایه را داشتیم و دستور ما از نوع هدر نبود، می‌توان یک نمونه مجازی از آن ایجادکرد.

برای کسب اطلاعات بیشتر در رابطه با مدل به صفحه ویکی‌پدیای آن مراجعه کنید.

### **ویو**

این لایه داده‌ها را از مدل درخواست کرده و داده‌های نهایی را برای ارائه به خروجی نمایش آماده می‌کند. به عبارت دیگر این لایه تنظیمات پویا جهت آماده‌سازی حالت نمایشی و ارسال به فایل display.html را بر عهده دارد. این لایه به کمک فایل دیسپلی عمل می‌کند و اگر از روترها استفاده کرده باشید به صورت اتوماتیک داده‌های بازگشتی مدل به تابع های خاص این کلاس ارسال می شود. در این لایه می‌توان تمامی حالت‌های نمایشی صفحه را کنترل کرد و به عهده گرفت.

### **کنترلر**

فایل مادر هر ماژول است، و بودنش به معنی استقلال ماژول به حساب می‌آید. اگر ماژولی دارای این فایل نباشد به معنی عدم استقلال ماژول است و نمی توان از آدرس‌های تعریف شده استفاده کرد، به گونه دیگر نمی‌توان گفت که پوشه ما ماژول است بلکه حکم زیرماژول را دارد. اگر ماژولی دارای این فایل نباشد از کنترلر اولین ماژول home استفاده می‌کند. به عنوان مثال ماژول user/logout دارای فایل controller نیست. پس سیستم در فایل user/home به دنبال کنترلر می‌گردد و اگر این پوشه نیز وجود نداشت در پوشه user به دنبال آن می‌گردد و در صورت عدم وجود فایل controller، به پوشه home منتقل می‌شود و از آن پوشه کنترلر را درخواست می‌کند.

همانگونه که از اسم این فایل مشخص است، وظیفه این فایل به عهده گیرنده پردازش‌های اصلی و کنترل کننده فایل است، از قبیل توابع مشترک در model, view و کنترل سطوح دسترسی، کنترل آدرس‌بار صفحه و دیگر خصیصه‌های اصلی. در هنگام فراخوانی یک صفحه قبل از هر اتفاقی کنترلر فرخوانی می‌شود.

### **دیسپلی**

این فایل که با پسوند html ذخیره می‌شود تم و حالت نمایشی صفحه را به عهده می‌گیرد که با ساختار نوشتاری twig به متغیرهای view متصل می‌شود.

برای کسب اطلاعات بیشتر در مورد توئیگ به مستندات آن مراجعه شود.

# افزودن ماژول جدید

اگر درون پوشه مربوطه به ماژول فایل controller.php وجود داشته باشد، آن ماژول به عنوان یک ماژول مستقل شناخته شده و در غیر این‌صورت در تعریف ما یک ماژول وابسته است. هر ماژول مستقل علاوه بر فایل controller.php که اجباری است و مشخص کننده استقلال ماژول است می تواند فایل‌های زیر را نیز داشت باشد.

model.php

view.php

controller.php

display.html

پس از درخواست کاربر به هر صفحه اولین لایه‌ای که مورد خوانده می‌شود، کنترلر است. سلسه مراتب دسترسی به کنترلر در پوشه ها به شرح زیر است.

content/account/login/controller.php => content/account/login/controller.php

content/account/login/no controller! => content/account/home/controller.php

content/account/home/no controller! => content/account/controller.php

content/account/no controller! => content/home/controller.php

content/home/no controller! => bigbang :(

## ماژول مستقل

اطلاعات هر ماژول درون یک پوشه با نام دلخواه ذخیره می‌شود. البته می‌توان برای سهولت کار ماژول ها را دسته‌بندی کرد تا چند ماژول درون یک پوشه قرار گیرد. با توجه به نیاز موجود، ما می‌توانیم تا دو ماژول مستقل تو در تو داشته باشیم و از آدرس سوم به بعد باید از طریق ماژول وابسته پیاده سازی شود.

برای مثال اگر شما ماژولی برای ورود کاربران دارید، می‌توانید به دو صورت زیر عمل کنید.

content/login

content/account/login

اگر در سه کلاس کنترلر، مدل و ویو نیاز به تابع سازنده (constructor) داشتید، می‌توانید به صورت زیر عمل کنید

function \_counstruct($\_controller = null)

{

…

parent::\_counstruct(...func\_get\_args());

…

}

برای ساخت یک ماژول مستقل، یک پوشه درون پوشه content پروژه خود ساخته و چهار فایل زیر را درون آن قرار دهید.

/content/home/controller.php

<?php

namespace content\home;

class controller extends lib\mvc\controller

{

}

?>

/content/home/view.php

<?php

Namespace content\home;

class view extends \lib\mvc\view

{

}

?>

/content/home/model.php

<?php

namespace content\home;

class model extends \lib\mvc\model

{

}

?>

/content/home/display.html

{%extends display.mvc%}

{%block content%}

{%endblock%}

## ماژول وابسته

ماژول وابسته بدین معناست که این ماژول برای اجرا نیاز به یک ماژول دیگر دارد و به تنهایی قابل اجرا نیست. در ماژول وابسته، ماژول دیگر فایل کنترلر را ندارد، البته با توجه به سبک طراحی باید یک کنترلر وظایف مورد نیاز را بر عهده گیرد که در ادامه روش کار را شرح خواهیم داد. البته ماژول می تواند سایر لایه ها را هم نداشته باشد و یا حتی به طور کل مجازی بوده و مثل لاگ اوت توسط یک تابع کنترل شود.

می خواهیم صفحات پویای زیر را درست کنیم.

/javad/noshahr

/saman=ramhormoz

/hasan/cityNumber=12

/javad/cityNumber=13

/saman/cityNumber=14

اگر آدرس های بالا را در آدرس بار مرورگر وارد کنید متوجه می شوید که صفحه پیغام۴۰۴ را داده؛ چون این ماژول ها کنترلر و حتی شاید پوشه نداشته باشند. اگر درست متوجه شده باشید، قرار است همه این ماژول های وابسته از کنترلر home استفاده کنند. پس در کنترلر قواعدی را می سازیم تا دیگر خطای ۴۰۴ تولید نشود، و آدرس های ما معتبر باشد.

# بررسی کامل فایل‌های هر ماژول

## کنترلر

فایل controller.php همان‌طور که از اسم آن مشخص است وظیفه کنترل و مدیریت بین ماژول‌ها را به عهده دارد. در این فایل شما می‌تواند مشخص کنید که چه آدرس‌هایی به چه پوشه‌هایی منتقل شوند و یا سینتکس درست آدرس‌ها چیست و یا اینکه سطح دسترسی به چه صورت می‌باشد.

دو تابع اصلی در کنترلر به نام‌های config, \_route موجود است. تابع config بعد از سازنده کلاس اجرا می‌شود، معمولا در کنترلر ما مستقیما تابع سازنده را نمی‌سازیم. پس اگر نیاز بود بعد از ساخت کنترلر عملیاتی را انجام دهید از این تابع استفاده می‌کنید.

تابع \_route وظیفه سازماندهی آدرس‌ها و مسیرهای این کنترلر یا به اصطلاح این ماژول می‌باشد،‌ این تابع قبل از تابع config اجرا می‌گردد،‌ پس می‌توانید مطمئن باشید که اگر کاربر اجازه اجرای این صفحه را در بخش \_route نگرفته است، تابع config شما اجرا نخواهد شد.

شما می‌توانید درون این تابع آدرس اصلی کنترلر خود را تغییر دهید، آدرس model, view, display خود را سازماندهی کنید.

اگر در این تابع نام کنترلر عوض شد و شما سیستم را به کنترلر دیگری هدایت کردید، سیستم دوباره عملیات پیداکردن،‌ اجرا کردن و سازماندهی آن کنترلر را اجرا می‌کند.

content/home/controller.php

<?php

namespace content\home;

class model extends \lib\mvc\model

{

function \_route()

{

$this->get(‘check’, ‘vcheck’)->ALL();

$this->post('login')->ALL();

}

}

## دیسپلی

کلیه خروجی قابل نمایش به کاربر در فایل display.html آماده خواهد شد. سیستم توئیگ[[5]](#footnote-5) به این فایل اضافه شده و تمام کنترل‌ها بر روی مقادیر از طریق توئیگ امکان‌پذیر است. برای آشنایی بیشتر با توئیگ به داکیومنت آن مراجعه کنید.

یکی از ویژگی‌های توئیگ، استفاده از توابع است. برای مثال با استفاده از تابع فرم‌ساز به راحتی می‌توانید فرم مورد نظر خود را بسازید که نمونه آن را در بخش مدل دیدید.

/content/home/display.html

{%extends display.mvc%}

{%block content%}

<h1>{{header\_test}}</h1>

<form method="post">

<input type=”text” name=”username” placeholder=”Users Name”>

<input type=”submit”>

</form>

{%endblock%}

## مدل

فایل model.php وظیفه ارتباط با دیتابیس و تهیه خروجی برای استفاده در سایر لایه‌ها را بر عهده دارد. برای آشنایی بیشتر با طریقه ارتباط با دیتابیس مستندات بخش sql را مطالعه فرمایید. در ساختار مهندسی MVC و به دلیل یکپارچه بودن سیستم تنها کانال انتقال و دریافت کلاس model می‌باشد. پس ناگزیر باید ساختاری طراحی شود که شما در هرکجای سیستم که هستید بتوانید از model استفاده کنید و اطلاعات خود را ارسال و یا دریافت نمایید.

/content/users/add/model.php

<?php

namespace content\home;

class model extends \lib\mvc\model

{

function get\_check($object)

{

return (array) $object;

}

function post\_login($object)

{

\lib\debug::true("life login");

}

}

?>

## ویو

فایل view.php وظیفه این فایل ایجاد و مدیریت آیتم ها و مقادیری است که در فایل display.html قرار است نمایش داده شود. فرض کنید می‌خواهید لیستی از کاربران خود را ایجاد کرده و نمایش دهید، ایجاد این لیست در این فایل صورت می‌پذیرد و مقادیر آن به فایل display.html ارسال شده تا در تگ ها جایگزاری شود. به مثال زیر توجه کنید.

/content/users/list/view.php

<?php

namespace content\users\list;

class view extends \lib\mvc\view

{

function view\_vcheck($model\_callback)

{

$this->data->header\_test = “I Love IRAN”;

}

?>

در این مثال ما در روتر کنترلر خود دو مسیر تعریف کردیم، یکی مسیر برای اجرای عادی صفحه که در حالت api همان درخواست get می‌باشد. به صورت پیشفرض نیاز به مسیرسازی برای این حالت نمی‌باشد، اما از آن‌جا که شاید نیاز باشد در اجرای صفحه درخواستی از سمت پایگاه داده‌ها و یا آنالیزهای سمت سروری داشته باشیم از طریق این مسیرسازی درخواست‌های سمت سروری را به سمت تابع check و در بخش view به سمت تابع vcheck ارسال می‌نماییم.

در مسیرسازی دوم ما متد ارسال http را برر روی post مسیرسازی می‌کنیم و model را به سمت تابع login هدایت می‌نماییم، از آنجا که متدهای post, put و delete نیاز به اجرای بخش view و display ندارند پس نیازی نیست که در مسیرسازی اشاره‌ای به تابعی در view انجام دهیم.

# کنترل صفحه

برای کنترل آدرس‌های اضافی موجود در آدرس بار مرورگر از این تابع استفاده می شود.

## روتر

برای روتر کردن می توان به صورت مستقیم از خود کلاس روتر استفاده کرد.

new \lib\router\route(conditions, [[afterTrue]..args]);

اما به یاد داشته باشین که اگر در ساختار mvc بودید. یعنی در یکی از کلاسهای مدل ویو و یا کنترلر بودید باید از حالت زیر استفاده کنید.

$this->route(conditions, [[afterTrue]..args]);

پارامتر اول قوانین روتر است که اگر حالت صفحه شما با آن قوانین هماهنگ بود روتر مقدار true را بر می گرداند.

پارامتر دوم یک تابع دلخواه می باشد که اگر روتر مقدار true را برگشت داد این تابع اجرا می‌شود. پارامترهای دوم به بعد مقادیر ارسالی به تابع دلخواه هستند و اگر تابع وجود داشته باشند به آرگومان‌های آن تابع ارسال می‌گردند.

### **قوانین مقادیر ورودی روتر**

در حالت رشته‌ای اگر مقدار قانون رشته باشد، هماهنگ بودن آن با آدرس صفحه چک می‌شود.

در حالت عبارت ویژه اگر مقدار قانون عبارت ویژه باشد، آدرس صفحه با آن عبارت چک می‌شود.

از حالت آرایه ای در گاهی از اوقات که لازم است چندین مقدار برای قانون گذاری دخیل باشند، استفاده می‌شود. برای این کار از آرایه استفاده می کنیم. به نمونه زیر دقت کنید.

url => array() or strign or RegExp

real\_url => array() or strign or RegExp

fn => array(function, ...args) or function

max => int

min => int

sub\_domain => array(), strign, RegExp

domain => array(), strign, RegExp

property => array()

get =>array()

post =>array()

url : مقدار آدرس صفحه را مشخص می‌کند، البته آدرس مجازی. بهتر است در از این مقدار به جای مقدار real\_url استفاده کنید. چرا که ممکن است در مسیر اجرای صفحه جایی real\_url دست خورده باشد و در اصل ما نیاز به url داشته باشیم. مثلا فرض کنید ما سیستم api خود را فعال کرده‌ایم. سیستم api زمانی کار می‌کند که ابتدای همه آدرس‌ها عبارت api نوشته شود. مانند مثال زیر

site.com/api/post

site.com/api/post/10

در این حالت برای اینکه مسیریابی صحیح باشد ما بعد از تشخیص اینکه سیستم api فعال است عبارت api را از url حذف می‌کنیم. حال اگر شما در قوانین از real\_url استفاده کرده باشید باید دو حالت روتر برای حالت api و غیر آن داشته باشید

default router

real\_url => array(“post”, “/\d+/”);

api router

real\_url => array(“api”, “post”, “/\d+”)

اما اگر از حالت url استفاده کرده باشید، فقط یک حالت کافی است. چرا که عبارت api از url حذف شده است.

url => array(“post”, “/\d+/”);

برای حالت‌های url ,real\_url, sub\_domain , domain می توان از سه حالت استفاده کرد.

حالت اول رشته است که دقیقا مقادیر را مطابق آن می‌سنجد. حالت دوم عبارت ویژه است که مقدار درخواستی را با عبارت ویژه می سنجد و حالت سوم آرایه است که درون آرایه ها می‌توان از رشته و عبارت ویژه استفاده کرد. خانه اول آرایه اولین مقدار درخواستی است. مثلا آدرس صفحه ما به صورت زیر است.

/admin/post/10

برای قانون گذاری روتر آدرس url به صورت زیر عمل میشود.

array(“admin”, “post”, “/\d+/”)

min ,max: مقدار max به صورت پیشفرض صفر است. اگر شما مقدار url را عبارت ویژه قرار دهید مقدار max خوانده نشده و بستگی به قانون‌گذاری شما در عبارت ویژه دارد. اما اگر مقدار رشته بود باید max را تعریف کنید و اگر آرایه بود مقدار max از طول آرایه گرفته می‌شود. همه این‌ها زمانی است که شما مقداری برای max تعریف نکرده اید اما اگر تعریف کنید برابر با مقدار شما می‌گردد. مقدار max , min تعداد دایرکتوری های شما در آدرس صفحه می باشد.

Fn: می تواند دو حالت مقداری بگیرید. اگر تابع گرفت تابع اجرا میشود و اگر بازگشت true داشت که عملیات صحیح است اما اگر بازگشت false داشت قانون شما اشتباه است و روتر مقدار false بر می‌گرداند. اما اگر آرایه بود مقدار اول باید تابع باشد که همانند بالا عمل می‌شود و بقیه مقادیر همان آرگومان های تابع هستند.

توجه داشته باشین که هسته در حین بارگذاری قبل از اینکه وارد ساختار mvc شما شود اگر کلاسی به اسم cls/router داشته باشین وارد کلاس شده و اگر در کلاس تابع \_before باشد قبل از روتر اصلی هسته اجرا می‌شود و اگر \_after باشد بعد از روتر اصلی هسته اجرا میشود.

# انتقال دهنده صفحه

ریداریکتور یا همان انتقال دهنده صفحه به ما کمک می کند که صفحه خودمان را به صفحه ای دیگر که مورد نیاز است انتقاد دهیم.

برای ریدایرکت کردن می توان به صورت مستقیم از خود کلاس ریدایرکت استفاده کرد.

$redirector = new \lib\redirector($url = null, $php = true);

آرگومان اول می‌تواند رشته یک آدرس خاص باشد و آرگومان دوم یک مقدار منطقی است که اگر true بود ریدایرکت کردن صفحه از طریق php و با متد header صورت می‌گیرد، اما اگر false بود از طریق تگ meta در html صورت می‌گیرد.

اما اگر قصد دارید در داخل وب‌سایت خود از صفحه‌ای به صفحه دیگر ریدایرکت شوید و یا از ماژولی به ماژول دیگر می‌توانید مقدار url را null گذاشته و از طریق متدهای این کلاس آدرس دهی کنید.

$redirector->set\_protocol(‘ftp’);

این متد پروتکل آدرس صفحه را تنظیم می‌کند. به عنوان مثال اگر آدرس صفحه فعلی شما <http://saloos.ir> باشد، اگر مثال بالا را اجرا کنید، آدرس صفحه به <ftp://saloos.ir> تغییر پیدا می‌کند، بدون نیاز به نام دهی دومین.

$redirector->set\_domain($domain = null)

این متد نام دامنه ریدایرکتور را مشخص می‌کند. اگر نام دامنه null باشد، زیر دامنه‌ها را حذف کرده و نام خالص دامنه را جایگزین می‌کند. به مثال زیر توجه کنید.

url: http://portal.saloos.ir/login

$redirector->set\_domain();

url: <http://saloos.ir/login>

همانطور که مشاهده می‌کنید تنها نام دامنه را از زیر دامنه به دامنه اصلی تغییر داده و هیچ تغییر دیگری در آدرس صفحه داده نشده است، که باید به این امر توجه شود.

$redirector->set\_url($value = '')

این متد آدرس دایرکتوری‌های صفحه را تغییر می‌دهد که می‌تواند دو حالت رشته‌ای و آرایه‌ای بگیرد.

1. <http://ermile.com> [↑](#footnote-ref-1)
2. <https://en.wikipedia.org/wiki/Software_as_a_service> [↑](#footnote-ref-2)
3. <https://en.wikipedia.org/wiki/Representational_state_transfer> [↑](#footnote-ref-3)
4. <https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller> [↑](#footnote-ref-4)
5. http://twig.sensiolabs.org/documentation [↑](#footnote-ref-5)