CODE BRIGHT

YENI BAŞLAYANLAR İÇIN LARAVEL FRAMEWORK *VERSIYON 4* İLE WEB UY-GULAMA GELIŞTIRME

DAYLE REES & SINAN ELDEM

Laravel: Code Bright (TR) Türkçe

Yeni Başlayanlar İçin Laravel Framework Versiyon 4 İle Web Uygulama Geliştirme

Dayle Rees, Sinan Eldem ve Antonio Laguna

Bu kitap http://leanpub.com/codebright-tr adresinde satıştadır.

Bu versiyon, 2015-08-29 tarihinde yayınlanmıştır

Bu Kitabı Tweet'le!

Yazara, Dayle Rees, Sinan Eldem ve Antonio Laguna, destek olmak için bu kitabı Twitter 'da paylaşın!

Bu kitap için önerilen tweet:

Laravel: Code Bright kitabının Türkçe Çevirisi http://leanpub.com/codebright-tr #codebright-tr #laravel @laraveltr

Bu kitap için önerilen hashtag #codebright-tr.

Bu linke tıklayarak, Twitter'da bu kitap hakkında neler paylaşıldığını görebilirsiniz:

https://twitter.com/search?q =#codebright-tr

İçindekiler

Teşekkürler

Her şeyden önce kız arkadaşım Emma'ya teşekkür etmek istiyorum, sadece benim tüm asosyal girişimlerime tahammül ettiği için değil, aynı zamanda her iki kitabım için müthiş kırmızı panda resimleri çektiği için! Seni seviyorum Emma!

Taylor Otwell, geçen yıl inanılmaz oldu, bana ekibin bir parçası olma fırsatı verdiğin için ve dostluğun için teşekkür ederim. Kullanması gerçekten zevk veren bir framework yaptığın için, kodlarımızı şiir okunur gibi yaptığın için ve onun geliştirilmesine bu kadar zaman ve tutku koyduğun için teşekkür ederim. Laravel'in bu yeni versiyonunda seninle çalışmaktan gerçekten zevk aldım ve gelecekteki projelerde tekrar seninle çalışmayı umuyorum!

Eric Barnes, Phill Sparks, Shawn McCool, Jason Lewis, Ian Landsman, çatıyla ilgili tüm destekleriniz ve iyi dostlar olduğunuz için teşekkürler.

Anne ve babama teşekkür ediyorum, yirmi sekiz yıldır benim asosyal çabalarımı destekliyorlar! Ve yine aile üyeleri için benim ilk kitabımdan bir milyar kopya kadar aldıkları için teşekkürler!

İlk kitabım Code Happy almış olan herkese ve Laravel topluluğunun hepsine teşekkür ederim. Sizin desteğiniz olmadan ikinci kitabım asla gerçekleşemezdi.

Çevirenin Notu

Bu kitap, ilk çeviri tecrübem olarak bana son derece keyif verdi. Çeviriyi yaparken bir yandan da öğrendim, bu da işi daha zevkli hale getirdi.

Dayle Rees'in samimi anlatımı ve hemen her konuyu örneklendirmesi öğrenme sürecinde her bilgi seviyesindeki kullanıcıya son derece yardımcı olacak bir kaynağa dönüştürdü bu kitabı.

Öncelikle sevgili eşim Bilge ve gözümün ışığı kızım Tuana Şeyma'ya teşekkürler. İyi ki varsınız!

Laravel Türkiye Forumları[1]'nda oluşturduğumuz dokümantasyon çeviri ekibine, kısa zamanda belgelerin tamamlanmasını sağladığınız ve Laravel'in kapılarını Türkçe dilini kullanan tüm kullanıcılara açtığınız için teşekkürler.

Gerek dokümantasyon, gerekse bu kitabın çevirisinde tüm süreç boyunca yanımda olan ve çok katkı sağlayan değerli Sergin Arı'ya, kattıklarından dolayı minnettarım. Sen olmadan olmazdı!

Çeviri sürecinde ince eleyip sık dokudum ancak yine de hatalar yapmış olabilirim, bu sebeple karşılaşmanız muhtemel hataları bana aşağıdaki kanallardan bildirirseniz sevinirim.

E-posta: sinan@sinaneldem.com.tr

[1] http://www.laravel.gen.tr

Web: www.sinaneldem.com.tr[2]

Twitter: twitter.com/sineld[3]

Diğer Laravel Türkçe Kitapları: leanpub.com/u/sineld[4]

[2] http://www.sinaneldem.com.tr/
[3] http://twitter.com/sineld/
[4] https://leanpub.com/u/sineld/

Giriş

Evet, bir kitap bölümü yazmayalı çok zaman oldu. Code Happy 12 ay kadar önce yayınlandı ve üç bin satış rakamını aştı. Yazı nasıl yazılır hatırlayabilecek miyim bakalım.

O kitabı okuduysanız benim öncelikle bir geliştirici, ikinci olarak bir yazar olduğumu zaten biliyorsunuzdur. Bu nedenle, bu kitapta uzun kelimeler göremeyeceksiniz. Shakespeare'i hiçbir şey etkilemeyecektir nasıl olsa (yazım hataları dışında). Laravel çatısını öğrenmek için, basit, düz konuşmalar **alacaksınız**. Ayrıca tutku alacaksınız! Terli yatak çarşafları türünde bir tutku değil, rakipsiz Laravel framework coşkusu. Ben kitaplarımı karşınızda durmuş, sizinle karşılıklı konuşur gibi yazmayı seviyorum. Aslında, eğer gerçekten benimle konuşmak istiyorsanız, o zaman Laravel IRC kanalına gelin ve beni görün!

Şimdi, 'Yazar hakkında bilgi' paragrafına geldik. Burayı kimse okumak istemez, fakat bir miktar egonun kimseye zararı olmaz, öyle değil mi?

Benim adım Dayle Rees (kapakta öyle diyor!) ve ben bir web geliştiricisi ve bir tasarım tutkunuyum. Galler kıyısında küçük bir kasaba olan Aberystwyth'liyim. Son kitabım 'Code Happy'yi yazdığım sırada Aberystwyth'de Galler Milli Kütüphanesinde çalışıyordum, burası Birleşik Krallıktaki üç telif kütüphanesinden biridir.

Galler başkenti Cardiff'e taşındığımdan bu yana BoxUK ile çalışıyorum. BoxUK bir internet danışmanlık ve geliştirme örgütüdür, orada web geliştirme dünyasına meraklı bir geliştiriciler ekibi ile birlikteyim.

Web geliştirme benim sadece işim değil, aynı zamanda hobim. Yararlı ve ilginç kod parçaları ya da güzel tasarımlar bulmak hoşuma gidiyor. Yeteneklerimizin harika şeyler üreteceğine inanıyorum ve hayata geçmiş fikirler görmeyi seviyorum.

Bir yıldan biraz daha önce Laravel topluluğuna kod demetleri, web tasarımları ve yapabildiğim başka yollarla yardımcı olmaya başladım. O zamandan bu yana ilişkim arttı. Laravel artık benim esas açık kaynak projem ve ben şimdi çatının çekirdek geliştirme ekibinin bir üyesiyim.

Laravel 4 (kod adı Illuminate) ile birlikte benim katılımım çok yükseklere çıktı. Bu sürümü, şimdiye dek kullanılabilecek en iyi çatı yapmak için Taylor Otwell ile birlikte çalışıyorum. Laravel 4 ile ilgili bir şey söylemeyin! Onu kullanmaya başlayın ve kod yazarken gülümsemelerinizi durduramadığınızda bize teşekkür edersiniz.

Laravel bir geliştirme aracının ne kadar üretken olabileceğini gösteren bir örnektir. Laravel'in güzelim sözdizimi Taylor Otwell'in rakipsiz dehasından geliyor. O bize şiir gibi okunacak kodlar yazma imkanı vermektedir ve kodlama görevlerimizden zevk almamızı sağlayacaktır.

> Peki Dayle, çatının son sürümüyle ne değişti?

Basit ama kafa karıştırıcı cevap, her şey ve hiçbir şey!

Laravel 4, bir milyar (tam rakam değil, saymadım) yeni özellikler ile birlikte esneklik ve test edilebilirliği artırmak üzere sıfırdan tekrar yazılmıştır. Laravel 3'te kodunuzu yapılandırmak için size bir miktar özgürlük verilmişti, Laravel 4 hackerların vahşi doğaya çıkmalarına ve çatıyı kendi gereksinimlerine uygun şekilde değiştirmelerine olanak sağlayacaktır.

Bir şeyin iyileştirildiğini duyduğumda her zaman bir bityeniği ararım fakat Laravel 4 öyle değil. O hala sevdiğiniz güzel ve ifade edici sözdizimine sahip; belki de onu daha çok sevdiğinizi göreceksiniz!

> Dostum, niye yeni bir kitap yazdın?

Code Happy 3.0 ile 3.2.x arasında dar bir sürümü kapsıyordu ve bir şeyleri doğru yapmış olmalıyım ki üç binden fazla kopya satıldı. Emin olun, Laravel 4 ile çalışması için çok büyük ihtimalle bütün bir kitabı yeniden işleyecektim. Bununla birlikte, çatının bu versiyonu yeni bir frameworktür. Eğer kitabı güncellemiş olsaydım, hala büyük bir çatı olduğuna inandığım sürüm 3 hakkındaki tüm bilgileri kaybedecektiniz. Birçok insanın Laravel 3'e dayalı projeleri olacaktır ve bu kişiler ihtiyaç duyduklarında Code Happy'deki bilgilere erişebilmelidir diye düşünüyorum.

Ayrıca kendi tecrübelerim var. Code Happy'yi bitirdikten sonra bir kitap yazma konusunda çok şeyler öğrendim. Şimdi kaçınabileceğim, sık yaptığım yanlışları öğrendim. Zaten yaptığım bir şeyi iyiye götürebilirim ve umarım öyle olur.

> Code Happy'yi okumamıştım! Önce onu mu okumalıyım?

İstiyorsanız okuyun, oraya bazı komik şakalar koymuştum. Ancak bu kitap da yeni başlayanlar içindir ve bu nedenle çok temel bilgilerden başlayacağız. Şayet zaten Laravel kullanıyorsanız devam edin ve ne değiştiğini görmek için ilginç parçalara geçin. Çatı için yeniyseniz, bana sadık kalmanızı ve sayfa sayfa okumanızı önereceğim. Merak etmeyin! İlginç tutmaya çalışacağım. Yakında, Laravel ile harika, etkileyici PHP uygulamaları oluşturmuş olacaksınız.

> Kitap ne zaman tamamlanacak?

Önceki kitabımda olduğu gibi, bu kitap da ilerledikçe yayınlanacak. Yani siz her bölümü ben yazdıkça alacaksınız. Kitabın şimdiki durumu tam olmayabilir ancak ek bölümleri ekledikçe bir e-posta alacak, güncellemeleri ücretsiz indirebileceksiniz.

Böyle yazma yönteminin büyük bir esneklik sağladığını düşünüyorum. Yanlışlarım varsa kolayca değiştirebileceğimi bilerek, yazdıklarım hakkında rahat olabiliyorum. Belli bir tarihe kadar yetiştirme telaşı olmadığında, yazacağım kitabın daha büyük kalitede olacağını hissediyorum. Gelecekteki sürümler için ya da ek bilgileri vurgulamak için bu kitabı güncelleyebilirim. Siz içeriğe daha hızlı erişebileceksiniz. Ayrıca, çatının yeni sürümünün piyasaya çıkmasıyla birlikte kitap yayınlayabilmemi de sağlamaktadır.

Sorulardan yoruldum..

İyi! Öyleyse, öğrenme sürecine başlamaya çok hevesli olmalısınız. Hemen atlayın ve Laravel'in keyfini çıkarmaya başlayın. Benimle sohbet etmek isterseniz bir tweet veya IRC'den mesaj göndermekten çekinmeyin!

Filtreler

Birkaç yıl öncesinde Jesse O'brien ve arkadaşlarının yerel hokey takımlarının Laravel Pandalarına karşı oynadıkları son maçlarını seyretmek için özel bir etkinlik planladıkları zamanı hatırladım.

Laravel Pandalarının Londra Şövalyeleri tarafından asla yenilgiye uğratılamayacağını hepimiz biliyoruz, fakat Jesse dinlemedi. Bunun Şövalyeler için zafere doğru giden yolun başlangıcı olacağında ısrar ediyordu.

Etkinliğin Londra'nın merkezindeki Hoser Hut'ta gerçekleştirilmesi planlanmıştı. 'Çok kuzey' Amerika'da (Maple şurubu ülkesi) doğmuş biri için dostça konuksever bir yer.

Ne yazık ki, Hoser Hut sınırdan gelenlere karşı öyle konuksever olmamakla bilinen bir üne sahipti. Amerikalıların düzenli olarak Hoser Hut pencerelerinin dışına atıldığı bilinen bir gerçekti. Kötü Amerikalıları dışarda tutmak için bir çeşit kapı filtresine ihtiyacı olduğu hükmüne varması bu yüzdendi. Tabii ki, iyi ingiliz adamı Dayle Rees Hoser Hut'ta her zaman iyi karşılanırdı. O her yerde iyi karşılanır.

Jesse, Hoser Hut'un önünde durup, gelen misafirlerin Kanada'lı olup olmadığını teyit etmek için kimliklerini göstermelerini istemek üzere bir fedai tuttu.

Görüyorsunuz ki, Jesse'nin yaptığı bir filtre uygulamaktı. Filtrenin gereksinimlerini geçenler Laravel Pandalarının Londra Şövalyelerini mahvettiğini görmek için sıcak ve rahat Hoser Hut'a giriş elde edecekti. Buna karşın bara girmeye çalışan Amerikalılar filtreyi karşılayamayacak ve kendilerine çizmenin parlak tarafı gösterilecekti.

Jesse'yi oyununa bırakalım ve uygulama rotalarımızı korumak için filtreleri nasıl kullanacağımızı görelim.

Basit Filtreler

Filtreler bir rotaya tatbik edilebilecek belirli kurallar veya eylemler kümesidir. Bunlar bir rota mantığının çalıştırılmasından önce veya sonra yapılabilirler ancak before filtrelerini daha yararlı bulacaksınız. Before filtrelerini kullanarak, eğer belirli kurallar veya kriterler karşılanmazsa, uygulamanın akışını değiştirebiliriz. Bu filtreler rotalarımızı korumanın mükemmel bir yoludur.

Her zaman olduğu gibi, bir örnek bin kelime konuşmaktan iyidir. Bir filtreyi inceleyelim ancak önce başka bir şeye ihtiyacımız var. Görelim:

```
<!-- app/views/dogumgunu.php -->

<h1>Mutlu yıllar!</h1>
<p>Mutlu yıllar Dayle, yaşa, varol!</p>
```

Süper! Artık doğum günü kutlama görünümümüz olduğuna göre, ilk filtremizi oluşturabiliriz. İşte başlıyoruz:

```
<?php

// app/filters.php

Route::filter('dogumgunu', function()
{
    if (date('d/m/y') == '12/12/84') {
        return View::make('dogumgunu');
    }
});
```

Bu ilk filtremiz oldu. Laravel, filtrelerimiz için genel bir yer olarak `app/filters.php` dosyasını sağlar ancak aslında bunu istediğiniz yere koyabilirsiniz.

Yeni bir filtre oluşturmak için `Route::filter()` metodunu kullanıyoruz. Birinci parametresi dostça bir isim olup, biraz sonra onu bir rota için filtre olarak atamak için kullanacağız. Bu örnekte ben 'dogumgunu' filtresi adını verdim. Rotaya ikinci parametre bir geriçağrı (callback) fonksiyonudur ve örneğimizde bu bir anonim fonksiyondur (Closure).

Callback filtre çalıştığı zaman çağrılan bir fonksiyondur. Bu fonksiyon tıpkı bizim rota mantığımızda kullandığımız gibi cevap tipinde bir nesne döndürürse, bu cevap döndürülecek ve rota mantığının sonucunun yerine bu sunulacaktır. Şayet filtre geriçağrı fonksiyonundan hiçbir cevap döndürülmezse, o zaman rota mantığı normal şekilde devam edecektir.

Bu bize büyük bir güç verir, öyleyse ilerleyin ve kötü kahkahanızı atın. Ciddiyim, bu önemli bir iş.

> Muahahahah!

Güzel, yapacağınız her şeyi ben söyleyeceğim. Gördüğünüz gibi ya uygulamanın akışını değiştirebiliriz veya bir eylem yapıp rota mantığının çalışmaya devam etmesine izin verebiliriz. Örneğin, biz web sitemizde belirli tipteki bir kullanıcıya sadece belirli tipte bir içerik göstermek isteyebiliriz. Bu başka bir sayfaya bir yönlendirme cevabı döndürmek yoluyla olabilir. Alternatif olarak, hangi sayfaların ziyaret edildiğini görmek için filtre her çalıştığında bir günlük tutabiliriz. Belki de kendimi öne alıyorum, örnek filtremize bir daha bakalım.

```php
<?php

// app/filters.php

Route::filter('dogumgunu', function()
{
    if (date('d/m') == '12/12') {
        return View::make('dogumgunu');
    }
});
```

Closure fonksiyonuna yakından baktığımızda, bir şart ve bir cevabımız olduğunu görüyoruz. Filtremizde, eğer şu andaki tarih '12/12/84'e, yani evrendeki en önemli kişinin doğduğu tarihe eşitse, closure o zaman bir cevap döndürecektir. Şayet Closure'den cevap dönerse mutlu yıllar görünümüne yönlendirileceğiz. Aksi takdirde rota mantığımız normal şekilde devam edecektir.

Tabii bir filtrenin işe yaraması için bir rotaya bağlamamız gerekiyor. Ancak, bunu yapmadan önce rotanın yapısını biraz değiştirmemiz gerekiyor. Rotalama metodlarının ikinci parametre olarak bir closure aldığını söylediğimi hatırlıyor musunuz? Pekala, ben yine beyaz bir yalan söyledim. Kusura bakmayın.

Gördüğünüz gibi, rota metodları ikinci parametre olarak bir dizi de kabul edebilmektedir. Rotaya ek parametreler atamak için bu diziyi kullanabiliriz. İkinci parametre olarak bir dizi verildiğinde bir rotanın nasıl göründüğüne bir bakalım.

```php
<?php

// app/routes.php

Route::get('/', array(function()
{
    return View::make('hello');
}));
```

Görüyorsunuz, oldukça benzer. Yaptığımız Closure'ı diziye çevirmek. O aynen önceki yaptığı işi görür. Aslında, closure'ı dizide tuttuğumuz sürece, başka değerler dahil edebiliriz. (Çevirenin notu: hello görünümünün Laravel ilk kurulduğunda ön tanımlı olarak açılış sayfası göstermek için oluşturulan görünüm olduğunu biliyorsunuz.) Şimdi filtreyi nasıl bağlayacağımıza geçiyoruz. 'before' filtre seçeneğine bakarak başlayalım.

```php
<?php

// app/routes.php

Route::get('/', array(
    'before' => 'dogumgunu:12/12',
    function()
    {
       return View::make('hello');
    }
));
```

Görebileceğiniz gibi, dizimizin içinde başka bir seçenek oluşturduk. Dizideki 'before' anahtarı framework'e rota mantığı çalıştırılmadan önce 'dogumgunu' filtresini çalıştırmak istediğimizi söyler. 'dogumgunu' değeri filtremize verdiğimiz takma ad ile eşleşmektedir.

İlerleyelim ve /'yi ziyaret ederek rotamızı çalıştıralım. Şimdi, bu günün Aralık'ın 12'si olmadığını varsayarsak, bu durumda Laravel karşılama sayfasını göreceksiniz. Çünkü filtrenin şartlı mantığından kalınmış ve bir cevap döndürülmemiştir.

Pekiyi, filtre şartı geçip de cevap döndürdüğü zaman ne olacağını görmek için 12 Aralık olana kadar bekleyelim.

Şaka yapıyorum, en iyisi filtreyi geçmeye zorlayacak şekilde değiştirelim. Şartı, boolean değer `true` olarak değiştirebiliriz.

```php
<?php

// app/filters.php

Route::filter('dogumgunu', function()
{
    if (true) {
        return View::make('dogumgunu');
    }
});
```

Başlayalım, bir şeylerin değişip değişmediğini görmek için /'i ziyaret edelim. Yaşasın, bu benim doğum günüm! Benim için mutlu yıllar şarkısı söyleyelim. Aslında, Aralık'a kadar beklemek lazım. O zaman doğumgünü filtre mantığının geçtiğini ve mutlu yıllar görünümü döndürüldüğünü görebileceğiz.

Bir rota dizisinin 'after' seçeneğini kullanarak bir filtre bağlayabiliriz, bu tür filtre rota mantığınızdan sonra çalıştırılacaktır. İşte bir örnek:

```php
<?php

// app/routes.php

Route::get('/', array(
    'after' => 'dogumgunu',
    function()
    {
       return View::make('hello');
    }
));
```

Ancak, aklınızda tutmanız gereken bir şey var, after filtresi cevabın yerine ikame edilemez. Dolayısıyla, 'after' kullanıldığı zaman bizim dogumgunu filtresi anlamsız olacaktır. Yine de bazı günlüğe yazma işleri veya temizleme operasyonları yapabilirsiniz. İhtiyacınız olduğunda onun orada olduğunu unutmayın yeter!

Çoklu Filtreler

Bilmeniz gereken başka bir şey de, bir rotaya istediğiniz sayıda filtre uygulayabileceğinizdir. Bu eylemin bir örneğini görelim. İlk olarak, çoklu before filtreleri bağlayalım:

```php
<?php

// app/routes.php

Route::get('/', array(
    'before' => 'dogumgunu|yilbasi',
    function()
    {
       return View::make('hello');
    }
));
```

Burada rotaya hem 'dogumgunu' hem de 'yilbasi' before filtreleri bağladık. Yeni 'yilbasi' filtresinin ne yapacağının mantığını senin hayal gücüne bırakıyorum ancak marifetli bir şey yapacağından eminim.

Boru | karakteri bir filtre listesini ayırmakta kullanılır. Liste soldan sağa doğru çalıştırılır ve bir cevap döndüren ilk filtre, isteği sonlandıracak ve o cevap sonuç olarak sunulacaktır.

İsterseniz çoklu filtre vermek yerine bir dizi de kullanabilirsiniz. Bu size belki daha 'phpemsi' gelebilir.

```php
<?php

// app/routes.php

Route::get('/', array(
    'before' => array('dogumgunu', 'yilbasi'),
    function()
    {
       return View::make('hello');
    }
));
```

Size hangisi uygunsa onu kullanın, ben şahsen dizileri seviyorum. İsterseniz bir 'before' ve 'after' filtresini aynı anda da ekleyebilirsiniz, bunun gibi:

```php
<?php

// app/routes.php

Route::get('/', array(
    'before'    => 'dogumgunu',
    'after'     => 'yilbasi',
    function()
    {
       return View::make('hello');
    }
));
```

Doğal olarak, ilk önce before filtresi çalışacak, sonra rota mantığı ve son olarak da after filtresi çalışacaktır.

İyi, filtreler tamam mı diyorsunuz? Bırakıp gitmek yok!

Filtre Parametreleri

Tıpkı PHP fonksiyonları gibi, filtreler de parametre kabul edebilirler. Bu, tekrarlardan kaçınmak için harika bir yoldur ve esneklik artışı imkanı verir. Her zaman olduğu gibi, bir örnekle gidelim.

```php
<?php

// app/filters.php

// before

Route::filter('test', function($route, $request)
{

});

// after

Route::filter('test', function($route, $request, $response)
{

});
```

Bir dakika, neden orada iki filtre var?

İyi fark ettiniz! Aslında onlar aynı filtre, ama yine de sizin sorunuz geçerli. Gördüğünüz gibi Laravel 'before' ve 'after' filtreleri için farklı parametre setleri sunmaktadır. Her iki filtrenin de `$route` ve `$request` değişkenleri aldığını unutmayın. Aslında bunlara istediğiniz ismi verebilirsiniz ancak bu şekilde isim vermemin bir nedeni var.

Eğer ilk parametreye `var_dump()` yaparsanız onun bir `Illuminate\Routing\Route` olgusu olduğunu göreceksiniz. Hatırlayacaksınız, 'Illuminate' Laravel 4 bileşenleri için kullanılan kod adıdır. 'Route' sınıfı rotalama katmanı tarafından kullanılan bir rotayı temsil eder. Bu olgu, çalışmakta olan güncel rotayı temsil eder. Zekice değil mi? 'Route' olgusu dev gibidir, bu Gallerli kurnaz adama inanmıyorsanız gidin onu `var_dump()` yapın. İçindeki bilgilerin ayrıntısını sorgulayabilir, hatta framework'ü manipüle etmek için bazı değerleri değiştirebilirsiniz. Bununla birlikte, bu ileri bir konudur ve bu bölümün kapsamı içinde değildir, o yüzden en iyisi biz sonraki parametreye bakalım.

Tahmin edebileceğiniz gibi, sonraki parametre güncel istek nesnesinin bir olgusudur. Web sunucunuza gönderilen isteğin durumunu `Illuminate\Http\Request` olgusu temsil eder. Bu olgu zengin ek bilgilerle birlikte URL'yi ve istekle geçirilen veriyi taşır.

After filtresi ek bir parametre alır, eylemi yapan rota filtresinden dönen bir cevap nesnesi olgusunu. Bu olgu güncel isteğin cevabı olarak sunulan neyse odur.

Pekiyi, Laravel'in bize verdiği bu parametreler frameworkun ileri kullanıcıları için yararlı olabilir ancak biz rota filtrelerimize kendi parametrelerimizi verebilsek harika olmaz mıydı? Bunu nasıl yapabileceğimizi bir görelim.

İlk olarak bizim filtre Closure'una yer tutucu bir değişken eklememiz gerekiyor, bu değişken Laravel'in kendi sağladığından daha sonra gelmelidir, bunun gibi:

```php
<?php

// app/filters.php

Route::filter('dogumgunu', function($route, $request, $tarih)
{
    if (date('d/m') == $tarih) {
        return View::make('dogumgunu');
    }
});
```

Bizim dogumgunu filtremiz bir `$tarih` parametresi kabul edecek şekilde değişmiş oldu. Eğer güncel tarih verilen tarihe uyarsa bu durumda dogumgunu filtresi çalışacaktır.

Şimdi de rota filtrelerine parametrelerin nasıl verileceğini öğrenmemiz gerekiyor. Bir bakalım.

```php
<?php

// app/routes.php

Route::get('/', array(
    'before'    => 'dogumgunu:12/12',
    function()
    {
       return View::make('hello');
    }
));
```

Rotaya atadığımızda iki nokta üst üste : karakterinden sonra gelen parametre filtremize geçirilir. Şimdi bunu test edelim, tarihi bu günkü tarihe değiştirelim ve filtrenin ateşlendiğini izleyelim.

Eğer ek parametreler vermek istersek, Closure'de fazladan yer tutucu değişkenler vermemiz gerekiyor. Bu şöyle bir şey olacaktır.

```php
<?php

// app/filters.php

Route::filter('dogumgunu', function($route, $request, $birinci, $ikinci, $ucuncu)
{
    return "{$birinci} - {$ikinci} - {$ucuncu}";
});
```

İstediğimiz kadar parametre alabiliriz. Birden çok parametre vermek için öncelikle filtre adı ile filtrenin parametreleri arasına iki nokta üst üste : eklemeliyiz. Parametrelerin kendileri de virgül ile , ayrılmalıdır. İşte bir örnek:

```
1  <?php
2
3  // app/routes.php
4
5  Route::get('/', array(
6      'before'    => 'dogumgunu:falan,filan,gibi',
7      function()
8      {
9          return View::make('hello');
10     }
11 ));
```

'falan', 'filan' ve 'gibi' değerleri filtreye eklediğimiz yer tutuculara geçirilecektir. Tıpkı fonksiyonlarda olduğu gibi filtre parametrelerine ön tanımlı değerler atayabiliriz, böylece onları opsiyonel yapmış oluruz. İşte bir örnek:

```
1  <?php
2
3  // app/filter.php
4
5  Route::filter('ornek', function($route, $request, $opsiyonel = 'Aynen!')
6  {
7      return $opsiyonel;
8  });
```

Opsiyonel parametreyi vermek veya vermemek. Bu size kalmış, O sizin frameworkünüz!

Filtrenizi daha verimli yapmak için istediğiniz kadar parametre kullanmakta serbestsiniz. Bu harika özelliğin avantajını kullanın.

Filtre Sınıfları

Closure'ler harika. Bunlar gerçekten kullanışlıdır ve örneklerimde iyi iş yapar. Bununla birlikte, bunlar yazdığımız mantığa sarılı kalırlar. Onları başlatamayız, bu da onların test edilmesini zorlaştırır.

İşte bu sebeple, bir Closure gerektiren her Laravel özelliği bir alternatife de sahiptir. Bir PHP sınıfına. Filtrelerimizi temsil etmek üzere bir sınıfı nasıl kullanacağımızı görelim.

Sınıf yapmadan önce, onu koyacak bir yere ihtiyacımız var. Şimdi `/app` klasöründe `filters` denen yeni bir klasör oluşturalım ve bu yeni klasörü de içermesi için `composer.json` classmap'ımızı güncelleyelim.

```
"autoload": {
    "classmap": [
        "app/commands",
        "app/controllers",
        "app/models",
        "app/filters",
        "app/database/migrations",
        "app/database/seeds",
        "app/tests/TestCase.php"
    ]
}
```

Şimdi bizim dogumgunu filtremiz için yeni bir sınıf oluşturalım. İşte yapıyoruz:

```
<?php

// app/filters/Dogumgunu.php

class DogumgunuFilter
{
    public function filter($route, $request, $tarih)
    {
        if (date('d/m') == $tarih) {
            return View::make('dogumgunu');
        }
    }
}
```

Ben bu sınıfa 'DogumgunuFilter' adını verdim, siz 'Filter' son ekini kullanmak zorunda değilsiniz, fakat ben böyle yapmayı seviyorum, gerisi size kalmış. Fakat zorunda olduğunuz bir şey var, `filter()` metodu. Bu metod tıpkı bir Closure gibi çalışır. Aslında, tıpkı Closure gibi çalıştığı için onu tekrar açıklamaya gerek yok. Öyle yapmak yerine bir filtrenin bir rotaya nasıl takılacağını görebiliriz.

Öncelikle bir filtre takma adı oluşturmamız gerekiyor, Bir kez daha biz `Route::filter()` metodunu kullanacağız. Bununla birlikte, bu sefer ikinci parametre olarak bir closure yerine bir string geçeceğiz. Bunun gibi:

```php
<?php

// app/routes.php

Route::filter('dogumgunu', 'DogumgunuFilter');
```

Bu metodun ikinci parametresi kullanacağımız filtre sınıfını tanımlayan bir stringtir. Eğer filtre sınıfı bir aduzayı içinde ise, o zaman aduzayını da vermemiz gerekiyor.

Artık filtre takma adı oluşturduğumuza göre, rotaya bunu aynen daha önce yaptığımız gibi ekleyebiliriz.

```php
<?php

// app/routes.php

Route::get('/', array(
    'before'    => 'dogumgunu',
    function()
    {
       return View::make('hello');
    }
));
```

Tabii ki, Composer ve Laravel'in bizim filtre sınıfımızı bulabilmeleri için öncelikle `composer dump-autoload` komutunu çalıştırmamız gerektiğini unutmayın.

Eğer kodunuzu tam olarak test etmek amacındaysanız, işiniz için en iyisi filtreleri sınıf olarak yazmaktır. İlerideki bir bölümde test konusunu daha ayrıntılı göreceğiz.

Evrensel Filtreler

Eğer `/app/filters.php` dosyasının içine bakarsanız garip görünen iki filtre göreceksiniz. Bunlar evrensel filtrelerdir ve uygulamanıza yapılan her istekten önce ve sonra çalışırlar.

```php
<?php

// app/filters.php

App::before(function($request)
{
    //
});


App::after(function($request, $response)
{
    //
});
```

Bunlar ön tanımlı olarak tüm rotalara uygulanmaları dışında tam olarak normal filtreler gibi çalışırlar. Yani bizim rotalarımızın `before` ve `after` dizi indekslerine bunları eklememize gerek yoktur.

Default Filtreler

`app/filters.php`de sizin için zaten oluşturulmuş bazı filtreler vardır. Bunlardan ilk üçüne bakalım.

```php
<?php

// app/filters.php

Route::filter('auth', function()
{
    if (Auth::guest()) return Redirect::guest('login');
});


Route::filter('auth.basic', function()
{
    return Auth::basic();
});

Route::filter('guest', function()
{
    if (Auth::check()) return Redirect::to('/');
});
```

Bu filtrelerin hepsi de Laravel'in kimlik doğrulama katmanı ile ilgilidir. Bunlar web uygulamamıza o anda giriş yapmış veya yapmamış kullanıcılara rota erişimini kısıtlamak için kullanılabilir.

Sonraki bölümlerin birinde kimlik doğrulama katmanına daha yakından bakacağız ve bu filtrelerin içeriği daha anlaşılır olacaktır. Şimdilik, bunların sizi orada beklediğini bilmeniz yeterli!

Dördüncü filtre siteler arası istek sahtekarlığı filtresidir ve şöyle görünmektedir:

```
<?php

// app/filters.php

Route::filter('csrf', function()
{
    if (Session::token() != Input::get('_token'))
    {
        throw new Illuminate\Session\TokenMismatchException;
    }
});
```

Rotalarınızı sizin uygulamanızdan başka bir kaynaktan post edilen isteklerden korumak için bu filtreyi ilgili rotalarınıza bağlayabilirsiniz. Bu çok yararlı bir güvenlik önlemi olup, esas olarak formlar veya veri gönderimi rotalarını korumak için kullanılmaktadır.

Laravel'in sağladığı filtrelerin avantajını kullanmaktan çekinmeyin, onlar size zaman kazandırmak için oradalar.

Desen Filtreleri

Filtreyi tüm rotalarınıza elle bağlamak istemezsiniz değil mi? Hayır sizi suçlamıyorum. Parmaklar yorulur, bir kitap yazdığım için bunu biliyorum. Zavallı küçük parmaklarınızı korumanın bir yolunu bulmaya çalışayım. Burada bir desen filtresi var.

Desen filtreleri jokerli bir rota deseni vererek bir before filtresini çok sayıda rotaya eşlemenize imkan verecektir. Bunu eylemde görelim.

```
<?php

// app/routes.php

Route::when('profile/*', 'dogumgunu');
```

Yukarıdaki `Route::when()` metodu, 'profile/' ile başlayan tüm rota URI'lerinde 'dogumgunu' filtresini çalıştıracaktır. İlk parametredeki yıldız bir joker olarak davranacaktır. Bu, bir before filtresini çok sayıda farklı rotaya bir seferde bağlamak için harika bir yoldur.