Laravel 4

Türkçe Dokümantasyon

Hazırlayan ve E-kitap'a dönüştüren

Sinan Eldem

Laravel 4 Türkçe Dokümantasyon

Laravel 4 Türkiye Forumları Çeviri Ekibi tarafından yapılan çeviriler

Sinan Eldem

Bu kitap şu adreste satılmaktadır http://leanpub.com/laravel4-tr

Bu versiyon şu tarihte yayımlandı 2013-10-10

Kitabı tweetleyin!

Sinan Eldem'a kitabını şu adresten Twitter tanıtarak yardımcı olun!

Kitap için önerilen tweet:

Laravel 4 Türkçe Dokümantasyon kitaba dönüştürüldü. #laravel4tr @laraveltr @laravelphp

Kitap için önerilen hashtag #laravel4-tr.

Kitap için diğerleri ne demiş merak ediyorsanız bağlantıya tıklayarak hashtagları arayabilirsiniz:

https://twitter.com/search?q =#laravel4-tr

İçindekiler

Laravel 4 Türkçe Dokümantasyon

Nedir?

Laravel etkileyici ve zarif sözdizimine sahip bir web uygulama çatısıdır (framework'tür). Geliştiriciliğin gerçekte eğlenceli, üretken deneyimlere dayanarak yerine getirilmesi gerektiğine inanır. Laravel birçok web uygulamasında kullanılan yetkilendirme, rotalama, oturum yönetimi ve kaşeleme gibi ortak görevleri kolaylaştırarak geliştiriciliğin zorluklarını ortadan kaldırmak amacını gütmektedir.

Laravel, geliştiriciler için, uygulama işlevselliğinden ödün vermeden geliştirme aşamasını memnuniyet verici hale getirmeyi amaç edinmiştir. En iyi kodu mutlu geliştiriciler yazar. Bu hedef için Ruby on Rails, ASP.NET MVC, ve Sinatra dilleri de dahil olmak üzere, diğer çatılardaki güzel özellikler bir araya getirilmeye çalışılmıştır.

Laravel büyük ve kapsamlı uygulamalar için gereken araçları içeren erişilebilir, aynı zamanda güçlü bir çatıdır. Mükemmel ters kontrol kapsayıcısı, etkileyici göç sistemi ve sağlam yerleşik ünite test desteği size geliştirmeyi amaçladığınız uygulama için gerekli araçları sağlayacaktır.

Çeviri

Laravel 4 Türkçe Dokümantasyonu[1], İngilizce bilmeyen kullanıcıların da istifade edebilmesi amacıyla Laravel Türkiye Forumları[2] kullanıcılarından oluşan gönüllü çeviri ekibi tarafından yapılmış çevirileri içermektedir.

Çeviri Ekibi

Katılım Sırasına Göre

- sineld[3]
- mecit[4]
- smh[5]
- Aristona[6]
- KenMilabel[7]

[1] http://dokuman.laravel.gen.tr/docs
[2] http://forum.laravel.gen.tr/
[3] https://github.com/sineld
[4] https://github.com/mecit
[5] https://github.com/smhayhan
[6] https://github.com/Aristona
[7] https://github.com/KenMilabel

- usirin[8]
- serginari[9]
- candelibas[10]
- ersinkandemir[11]
- ekrembk[12]

Çeviriye katkıda bulunmak isterseniz bu repo'yu fork edip, değişiklikleri yaptıktan sonra pull-request edebilirsiniz.

Çeviri bilgi forumu[13]

[8] https://github.com/usirin
[9] https://github.com/serginari
[10] https://github.com/candelibas
[11] https://github.com/ersinkandemir
[12] https://github.com/ekrembk
[13] http://forum.laravel.gen.tr/viewtopic.php?id=125

Editörün Notu

Bu kitap, orijinal Laravel dokümantasyonunun (belgelerinin), Laravel Türkiye Forumları'nda oluşturan çeviri ekibi tarafından Türkçe'ye çevirilen kullanma klavuzudur.

Laravel ile tanıştıktan kısa bir süre sonra çatının Php kullanıcılarına sağladığı kolaylıkları gördüm ve bunun Türkiye'de kullanılması için gereken adımları attım. Öncesinde bir forum, ardından da dokümantasyonun çevirilmesi geldi. Her şey beklediğimden daha hızlı gerçekleşti ve bu dokümantasyon haricinde üç kitabın da çevirisini tamamladım.

Bütün süreç boyunca yanımda olan sevgili eşim Bilge ve gözümün ışığı kızım Tuana Şeyma'ya teşekkürler. İyi ki varsınız!

Çeviri ekibine tek tek teşekkür eder, kattıklarından dolayı minettarlığımı bildiririm. Gerek dokümantasyonun, gerekse bu kitapların çevirisinde tüm süreç boyunca yanımda olan ve çok katkı sağlayan değerli Sergin Arı'ya, kattıklarından dolayı minnettarım.

Çeviri sürecinde ekibimiz çok ince eleyip sık dokudu ancak yine de hatalar yapmış olabiliriz, bu sebeple karşılaşmanız muhtemel hataları bana aşağıdaki kanallardan bildirirseniz sevinirim. Dilerseniz Github ambarından[14] değişiklikleri kendiniz de uygulayabilirsiniz.

Sinan Eldem

E-posta: sinan@sinaneldem.com.tr

Web: www.sinaneldem.com.tr[15]

Twitter: twitter.com/sineld[16]

[14] https://github.com/laravel-tr/docs

[15] http://www.sinaneldem.com.tr/

[16] http://twitter.com/sineld

Tanıtım

Laravel Felsefesi

Laravel etkileyici ve zarif sözdizimine sahip bir web uygulama çatısıdır (framework). Bizler geliştirmenin gerçekten tatmin edici olması için keyifli ve üretken bir deneyim olması gerektiğine inanıyoruz. Laravel birçok web uygulamasında kullanılan yetkilendirme, rotalama, oturum yönetimi ve ön bellekleme gibi ortak görevleri kolaylaştırarak geliştiriciliğin zorluklarını ortadan kaldırmak amacını gütmektedir.

Laravel, geliştiriciler için, uygulama işlevselliğinden ödün vermeden geliştirme aşamasını memnuniyet verici hale getirmeyi amaç edinmiştir. En iyi kodu mutlu geliştiriciler yazar. Bu hedefe varmak için, başka dillerde yazılmış Ruby on Rails, ASP.NET MVC ve Sinatra gibi çatılar da dahil olmak üzere, diğer çatılarda gördüğümüz en iyi özellikleri birleştirmeye çalıştık.

Laravel büyük, kapsamlı uygulamalar için gereken araçları içeren erişilebilir, aynı zamanda güçlü bir çatıdır. Mükemmel ters kontrol kapsayıcısı, etkileyici göç sistemi ve sağlam yerleşik ünite test desteği size geliştirmeyi amaçladığınız uygulama için gerekli araçları sağlayacaktır.

Laravel'i Öğrenmek

Laravel öğrenmenin en iyi yollarından biri tüm dokümantasyonunu dikkatlice okumaktır. Bu kılavuz size çatının çehresi ve uygulamanızda nasıl kullanacağınız konusunda rehber olur.

Bu kılavuza ek olarak Laravel kitapları[17]'na gözatabilirsiniz. Laravel topluluğunun yazdığı bu kitaplar çatıyı öğrenmek için çok iyi tamamlayıcı kaynaklar olarak hizmet edecektir:

- Code Bright (Türkçe Çeviri)[18], yazar: Dayle Rees ve Sinan Eldem
- Implementing Laravel (Türkçe Çeviri)[19], yazar: Chris Fidao ve Sinan Eldem
- Laravel: From Apprentice To Artisan (Türkçe Çeviri)[20], yazar: Taylor Otwell ve Sinan Eldem
- Laravel 4 Türkçe Dokümantasyon (Bu Kitap)[21], yazarlar: Türkiye Forumları Çeviri Ekibi
- Laravel Testing Decoded (İngilizce)[22], yazar: Jeffrey Way
- Getting Stuff Done With Laravel 4 (İngilizce)[23] by Chuck Heintzelman

[17] http://wiki.laravel.io/Books
[18] https://leanpub.com/codebright-tr
[19] https://leanpub.com/implementinglaravel-tr
[20] https://leanpub.com/laravel-4-tr
[21] https://leanpub.com/laravel4-tr
[22] https://leanpub.com/laravel-testing-decoded
[23] https://leanpub.com/gettingstuffdonelaravel

Geliştirme Ekibi

Laravel, çatının geliştirilmesi liderliğini sürdüren Taylor Otwell[24] tarafından oluşturuldu. Önde gelen diğer topluluk üyeleri ve katkıda bulunan kişiler Dayle Rees[25], Shawn McCool[26], Jeffrey Way[27], Jason Lewis[28], Ben Corlett[29], Franz Liedke[30], Dries Vints[31], Mior Muhammed Zaki[32] ve Phil Sturgeon[33]'dır.

Çatı Sponsorları

Aşağıdaki kuruluşlar Laravel framework geliştirilmesine mali katkılarda bulunmuştur:

- UserScape
- Cartalyst
- Elli Davis - Toronto Realtor
- Jay Banks - Vancouver Lofts & Condos
- Julie Kinnear - Toronto MLS
- Jamie Sarner - Toronto Real Estate

[24] https://github.com/taylorotwell
[25] https://github.com/daylerees
[26] https://github.com/ShawnMcCool
[27] https://github.com/JeffreyWay
[28] https://github.com/jasonlewis
[29] https://github.com/bencorlett
[30] https://github.com/franzliedke
[31] https://github.com/driesvints
[32] https://github.com/crynobone
[33] https://github.com/philsturgeon

Laravel Hızlı Başlangıç

Kurulum

Laravel Framework'ü kurmak için aşağıdaki komutu, komut işleyici uygulamanızdan çalıştırabilirsiniz:

```
composer create-project laravel/laravel your-project-name --prefer-dist
```

veya Laravel Framework'ü kurmak için Github Kaynağı'nı[34] indirmelisiniz. Daha sonra Composer'i kurup[35], `composer install` komutunu projenizin root (ana) dizininde çalıştırmalısınız. Bu komutu çalıştırmak, Laravel'i ve Laravel'in gereksinimlerini (dependencies) indirip kuracaktır.

Laravel kurulduktan sonra dizin yapısına göz gezdirin ve Laravel'in nasıl bir yapısı olduğuna bakın. `app` dizini içerisinde `views`, `controllers`, ve `models` gibi dizinler bulunmaktadır. Projenizi geliştirirken yazacağınız kodların çok büyük bir kısmı bu dizinlerin içine yazılacaktır. Ayrıca `app/config` dizini içerisine bakıp size ne tür ayar değerleri tanımlandığını görebilirsiniz.

Routing (Yönlendirme)

Başlangıç olarak Laravel'de ilk Route'umuzu yazalım. Laravel'de Route oluşturmak için en basit yol bir closure (anonim fonksiyon) kullanmaktır. `app/routes.php` dosyasını açın ve aşağıdaki kod parçacığını sayfanın en altına yapıştırın:

```
Route::get('kullanicilar', function()
{
        return 'Kullanıcılar!';
});
```

Şimdi, eğer web tarayıcınızda `/kullanicilar` adresine girerseniz, ekranda `Kullanıcılar!` yazısını görmüş olmanız gerekir. Eğer gördüyseniz çok iyi! İlk Route'unuzu başarıyla oluşturdunuz.

Route'lar ayrıca controller sınıflarına da bağlanabilir. Örneğin:

[34] https://github.com/laravel/laravel/archive/master.zip
[35] http://getcomposer.org

```
Route::get('kullanicilar', 'KullaniciController@getIndex');
```

Bu Route Laravel'e şunu belirtiyor: `/kullanicilar` rotasına yapılan bir istek `KullaniciController` sınıfındaki `getIndex` metodunu çağırmalıdır. Controller Routing hakkında daha fazla bilgi almak için Controller Dökümantasyonu'na[36] bir göz atın.

Bir View Oluşturma

Şimdi basit bir view dosyası oluşturup, kullanıcı bilgilerini ekrana view üzerinden yazdıracağız. View dosyaları `app/views` dizini içerisinde bulunmakta olup projenizin HTML dosyalarını barındırır. Şimdi bu dizin içerisine 2 tane dosya oluşturacağız: `layout.blade.php` ve `kullanicilar.blade.php`. Önce `layout.blade.php` dosyamızı oluşturalım:

```
<html>
        <body>
                <h1>Laravel Hızlı Başlangıç</h1>

                @yield('content')
        </body>
</html>
```

Şimdiki adımda ise `kullanicilar.blade.php` view dosyasını oluşturalım:

```
@extends('layout')

@section('content')
        Kullanıcılar!
@stop
```

Bu syntax size ilk etapta biraz yabancı gelebilir. Bunun sebebi Laravel'in güçlü templating sisteminin (Blade) kullanılmasıdır. Blade son derece hızlı çalışır çünkü sadece birkaç tane regex kodları kullanıp Blade syntaxını PHP scriptlerine dönüştürür. Blade kullanıcılarına çok büyük fonksiyonellik sağlar. Şablon kalıtımı (Template inheritance) ve PHP'nin `if` ve `for` gibi temel kontrol yapılarını Blade üzerinden kullanabilirsiniz. Daha fazla bilgi için Blade Dökümantasyonu'na[37] bakınız.

Şimdi gerekli view dosyalarımızı oluşturduğumuza göre, oluşturduğumuz viewi `/kullanicilar` isteğine bir cevap olarak döndürelim. `Kullanıcılar!` stringini döndürmek yerine, bu kez oluşturduğumuz view dosyalarını döndüreceğiz:

[36] /docs/controllers
[37] /docs/templates

```
Route::get('kullanicilar', function()
{
        return View::make('kullanicilar');
});
```

Harika! Bir layoutu genişleten bir view oluşturdunuz. Bir sonraki bölümümümüzde Veritabanı Katmanı (Database Layer) üzerinde duracağız.

Bir Migration Oluşturma

Bir veritabanı tablosu oluşturmak için Laravel'in migration özelliğini kullanacağız. Migrationlar çok kolay bir şekilde veritabanında değişiklikler yapmayı ve bunları takım arkadaşlarınızla paylaşmanızı sağlar.

Öncelikle bir veritabanı konfigürasyonu ayarlayalım. Tüm veritabanı konfigürasyonlarınızı `app/config/database.p` dosyası içerisinde değiştirebilirsiniz. Laravel öntanımlı olarak MySQL kullanmaya ayarlanmıştır, veritabanı konfigürasyonlarınızı `app/config/database.php` dosyası içerisine tanımlamanız gerekecektir. Dilerseniz `driver` değerini `sqlite` yapıp `app/database` dizininde bulunan SQLite veritabanını kullanabilirsiniz.

Sonra, bir migration oluşturmak için Artisan CLI[38] kullanacağız. Projenizin ana dizinine gelerek, aşağıdaki kodu terminal üzerinde yazın:

```
php artisan migrate:make create_users_table
```

Şimdi, oluşturulan migration dosyasını `app/database/migrations` dizininde bulun. Bu dosya 2 methoddan oluşmaktadır: `up` ve `down`. `up` methodunda, tablonuzdaki değişiklikleri yapmalısınız. `down` methodunda ise yaptığınız değişiklikleri geri almalısınız.

Şuna benzeyen bir migration oluşturalım:

```
public function up()
{
        Schema::create('users', function($table)
        {
                $table->increments('id');
                $table->string('email')->unique();
                $table->string('name');
                $table->timestamps();
        });
}
```

[38] /docs/artisan

```
11
12  public function down()
13  {
14          Schema::drop('users');
15  }
```

Şimdi bu migrationu Artisan CLI üzerinde `migrate` komutu kullanarak çalıştıralım. Projenizin ana dizinine gelip aşağıdaki kodu çalıştırın:

```
1  php artisan migrate
```

Eğer bir migrationu geri almak isterseniz `migrate:rollback` komutunu çalıştırmanız yeterli olacaktır. Şimdi bir veritabanı tablosu oluşturduğumza göre, tablomuzdan veri çekmeyi öğrenerek devam edelim!

Eloquent ORM

Laravel mükemmel bir ORM aracıyla beraber gelmektedir: Eloquent. Eğer daha önce Ruby on Rails frameworkü üzerinde çalıştıysanız Eloquent size çok tanıdık gelecektir, çünkü veritabanı işlemleri için ActiveRecord stilini kullanır.

Öncelikle, modeli tanımlayalım. Bir Eloquent modeli ilgili veritabanı tablosunu sorgulamak için kullanılabilir, aynı zamanda bu tablo içindeki belirli bir satırı temsil eder.. Merak etmenize gerek yok, örnekleri görünce ne kadar kolay olduğunu anlayacaksınız! Model dosyaları `app/models` dizininde bulunmaktadır. Şimdi o dizinde bir `User.php` modeli oluşturalım:

```
1  class User extends Eloquent {}
```

Lütfen dikkat edin, herhangi bir veritabanı tablosu belirtmedik.

Eloquent'in içerisinde birçok hüküm vardır, bunlardan birisi model adının çoğul yapısını veritabanı tablosu olarak kullandırmaktır. Kullanışlı, değil mi?

Tercih ettiğiniz veritabanı yönetim aracını kullanarak, `users` tablosuna birkaç satır ekleyin. Ondan sonra Eloquent'i kullanarak o tablodan bazı verileri çekip view dosyamıza göndereceğiz.

Şimdi `/users` routemizi editleyelim, ve şuna benzer bir hale getirelim:

```
Route::get('users', function()
{
        $users = User::all(); //Users tablosundaki tüm verileri $users değişkenine atar

        return View::make('users')->with('users', $users);
});
```

Şimdi bu scripti biraz inceleyelim. Öncelikle, `User` modelindeki `all` methodu `users` tablosundaki tüm verileri çekecektir. Daha sonra bu veriler `with` methodu kullanılarak view dosyasına gönderilir. `with` methodu bir anahtar ve bir değer almaktadır, böylece gönderilen veriyi view dosyası tanıyabilir.

Harika. Artık kullanıcıları view dosyamızda göstermeye hazırız!

Veri Gösterme

Artık `users` objesini view dosyamıza yönlendirdiğimiz için ekrana bastırabiliriz:

```
@extends('layout')

@section('content')
        @foreach($users as $user)
                <p>{{ $user->name }}</p>
        @endforeach
@stop
```

`echo` ifadesinin nerede olduğunu merak ediyor olabilirsiniz. Blade kullanırken, küme parantezi arasına yazılan değişkenler aynı `echo` ifadesindeki gibi ekrana bastırılır. Şimdi `/users` adresine girip veritabanınızda kayıtlı olan tüm kullanıcıların listesinin ekrana bastırıldığını görebilirsiniz.

Bu sadece bir başlangıç. Bu derste Laravel'in en temel konularını gördünüz, ancak daha göreceğiniz birçok heyecan verici özellikler var! Dökümantasyonu okumaya devam edin ve Laravel içerisinde gelen birçok farklı özellik hakkında daha fazla bilgiye sahip olun. Örneğin Eloquent[39] ve Blade[40]. Belkide sizin ilginizi Queues[41] ve Unit Testing[42] çekiyordur?. Yada IoC Container[43] kullanarak uygulamanızın mimarisini güçlendirmek istiyorsunuzdur? Seçim sizin!

[39] /docs/eloquent
[40] /docs/templates
[41] /docs/queues
[42] /docs/testing
[43] /docs/ioc

Laravel'e Katkıda Bulunulması

Giriş

Laravel ücretsiz ve açık kaynak bir yazılım olup, geliştirilmesine ve ilerletilmesine katkıda bulunabilinir. Laravel kaynak kodu Github[44] da bulunmakta olup, oradan projeye kolayca bir çatal açılarak (forking), katkılarınız birleştirilebilir (merging).

Alınsın Talepleri (Pull Requests)

Alınsın talebinin işleyişi, yeni özellikler için veya yazılım hataları için olmasına bağlı olarak farklılık gösterir. Yeni bir özellik için yapılacak bir alınsın talebi göndermeden önce, başlığında bir teklif `[Proposal]` konusu oluşturmanız gerekir. Teklif, yeni özellik yanında uygulama fikirlerini de açıklamalıdır. Bu teklif daha sonra gözden geçirilecek ve ya kabul ya da red edilecektir. Bir teklif onaylandıktan sonra, bu yeni özelliği uygulamaya koyan bir alınsın talebi oluşturulabilir. Bu ilkeye uyulmamış olan alınsın talepleri hemen kapatılacaktır.

Yazılım hataları için gönderilecek olan alınsın talepleri, bir teklif oluşturulmadan gönderilebilir. Eğer Github'da dosyalanmış olan bir yazılım hatası çözümünü bildiğinizi düşünüyorsanız, o durumda lütfen önerilen düzeltmenin detaylarını belirten bir not giriniz.

Dokümantasyon için eklemeler ve düzeltmeleri Github üzerindeki dokümantasyon ambarı[45]'na ilave edebilirsiniz.

Özellik Talepleri

Eğer Laravel'e ilave edildiğini görmek isteyeceğiniz bir 'yeni özellik' fikriniz varsa, Github'da başlığında 'Talep' `[Request]` olacak bir konu oluşturabilirsiniz. Özellik talebi, bir ana katılımcı tarafından gözden geçirilecektir.

Kodlama İlkeleri

Laravel, PSR-0[46] ve PSR-1[47] kodlama standartlarını takip eder. Bunlara ilave olarak, takip edilmesi gereken diğer standartların listesi şöyledir:

[44] http://github.com
[45] https://github.com/laravel-tr/docs
[46] https://github.com/php-fig/fig-standards/blob/master/accepted/PSR-0.md
[47] https://github.com/php-fig/fig-standards/blob/master/accepted/PSR-1-basic-coding-standard.md

- 'Namespace' deklarasyonlarının `<?php` ile aynı satırda olması gerekir.
- Sınıf (Class) açılışlarının `{` , sınıf ismi ile aynı satırda olması gerekir.
- Fonksiyon (Function) ve kontrol bloğu (control structure) açılışlarının `{`, farklı satırlarda olması gerekir.
- Arayüz (Interface) isimleri `Interface` son ekini alırlar, örneğin (`FalancaInterface`).

Kurulum

Composer Kurulumu

Laravel bağımlılıklarını yönetmek için Composer[48] kullanır. Öncelikle `composer.phar` dosyasını indiriniz. PHAR arşivini yerel proje dosyanızda tutabileceğiniz gibi `usr/local/bin` içerisine taşıyarak sisteminizde evrensel olarak da kullanabilirsiniz. Windows'ta Composer Windows kurulumu[49]nu kullanabilirsiniz. Setup Composer'i PATH değişkeni olarak kaydedecektir, böylece terminal üzerinde `composer` yazdığınızda Composer'i direkt olarak kullanabilirsiniz.

Laravel Yükleme

Composer'ın Create-Project Komutuyla

Terminalinizde Composer create-project komutunu yayınlayarak Laravel'i yükleyebilirsiniz:

```
composer create-project laravel/laravel --prefer-dist
```

Elle İndirerek

Composer yüklendikten sonra, Laravel framework'ün son sürümü[50]nü indirip, içeriğini sunucunuzdaki bir dizine çıkarınız. Ardından, Laravel uygulamanızın ana dizininde, Laravel gereksinimlerini yüklemek için, `php composer.phar install` (veya `composer install`) komutunu çalıştırınız. Bu işlemin başarıyla tamamlanabilmesi için sunucunuzda Git[51] yüklü olması gerekmektedir.

Laravel'i güncellemek isterseniz `php composer.phar update` komutunu yayınlayabilirsiniz.

- Composer kurulumu için Türkçe kaynak: Composer'ı Evrensel Olarak Kuralım[52]
- Laravel 4 kurulumu için Türkçe kaynak: Laravel Framework Kurulumu[53]

[48] http://getcomposer.org
[49] https://getcomposer.org/Composer-Setup.exe
[50] https://github.com/laravel/laravel/archive/master.zip
[51] http://git-scm.com/downloads
[52] http://www.sinaneldem.com.tr/composeri-evrensel-olarak-kuralim/
[53] http://www.sinaneldem.com.tr/laravel-framework-kurulumu/

Sunucu Gereksinimleri

Laravel framework'un birkaç sistem gereksinimi bulunmaktadır:

- PHP >= 5.3.7
- MCrypt PHP Eklentisi

Yapılandırma

Laravel'in çalışabilmesi için neredeyse hiç yapilandırma ayarı gerekmez. Geliştirmeye başlamak için serbestsiniz! Ancak `app/config/app.php` dosyasını ve dokümantasyonunu gözden geçirebilirsiniz. Buradaki `timezone` (saat dilimi) ve `locale` (lisan) gibi değerleri uygulamanızın ihtiyaçlarına göre düzenleyebilirsiniz.

> **Not**: Mutlaka ayarlamanız gereken bir yapılandırma seçeneneği `app/config/app.php` dosyasındaki `key` seçeneğidir. Bu değer rastgele seçilmiş 32 karakterden oluşmalıdır. Bu anahtar, değerler kriptolanacağı zaman kullanılmaktadır ve eğer doğru ayarlanmazsa kriptolanmış değerler güvenli olmayacaktır. `php artisan key:generate` artisan komutu ile bu değeri kolayca ayarlayabilirsiniz.

İzinler

Laravel `app/storage` dizin içeriğinin web sunucu tarafından yazılabilir olmasını gerektirmektedir.

Dosya Yolları

Framework dizin yollarının birkaçı yapılandırılabilirdir. Bu dizin yollarını değiştirebilmek için `bootstrap/paths.php` dosyasını gözden geçiriniz.

> **Not**: Laravel, public klasörüne sadece public olması zorunlu dosyaları koymak suretiyle, uygulama kodunuzu ve lokal depolamanızı koruyacak şekilde tasarlanmıştır. Ya public klasörünü sitenizin documentRoot (web root olarak da bilinmektedir)'u olarak ayarlamanız, ya da public klasörünün içeriğini sitenizin kök dizinine koymanız ve Laravelin diğer tüm dosyalarını kök dizini dışına koymanız önerilir.

Zarif URL'ler

Framework ile beraber gelen `public/.htaccess` dosyası URL'lerin `index.php` olmadan kullanımına olanak sağlamaktadır. Laravel uygulamanızın sunumu için Apache kullanıyorsanız `mod_rewrite` modülünün etkin olduğundan emin olunuz.

Eğer Laravel ile birlikte gelen `.htaccess` dosyası Apache kurulumunuz ile işlev göstermezse, bunu deneyiniz:

```
Options +FollowSymLinks
RewriteEngine On

RewriteCond %{REQUEST_FILENAME} !-d
RewriteCond %{REQUEST_FILENAME} !-f
RewriteRule ^ index.php [L]
```

Yapılandırma

Giriş

Laravel'in tüm yapılandırma dosyaları `app/config` dizini içindedir. Tüm dosyalardaki yapılandırma seçenekleri açıklanmıştır, dosyalara göz gezdirip size sunulan seçeneklere göz atabilirsiniz.

Bazen yapılandırma değerlerine run-time (çalışma anı) esnasında erişmeniz gerekir. Bunu `Config` sınıfını kullanarak yapabilirsiniz:

Bir Yapılandırma Değerine Erişmek

```
1  Config::get('app.timezone');
```

Eğer yapılandırma değeri bulunamazsa dönecek değeri ise ikinci bir parametreyle belirleyebilirsiniz:

```
1  $timezone = Config::get('app.timezone', 'UTC');
```

Lütfen dikkat edin, "nokta" şeklindeki kullanım biçimi tüm yapılandırma dosyalarına erişmenizi sağlar. Dilerseniz yapılandırma değerlerini run-time (çalışma anı) esnasında da ayarlayabilirsiniz:

Bir Yapılandırma Değeri Ayarlamak

```
1  Config::set('database.default', 'sqlite');
```

Çalışma zamanında ayarlanan yapılandırma değerleri sadece güncel istek süresince ayarlanırlar ve sonraki isteklere aktarılmayacaklardır.

Ortam Yapılandırması

Uygulamanın çalışma ortamına göre farklı yapılandırma değerlerine sahip olmak çoğu zaman iyidir. Örneğin, kişisel bilgisayarınızda, sunucudan farklı bir önbellekleme uygulaması kullanmak isteyebilirsiniz. Bunu ortam tabanlı yapılandırmalar oluşturarak sağlayabilirsiniz.

Bunu yapmak çok basit! `config` dizini içerisinde, ortam isminizi kullandığınız (örneğin `local`) bir dizin daha oluşturun. Şimdi, belirttiğiniz ortam için üzerine yazmak istediğiniz yapılandırma dosyalarınızı ve seçeneklerinizi geçirin. Örneğin, önbellekleme yapılandırmasının üzerine yazmak için, `app/config/local` dizini içerisinde `cache.php` dosyası oluşturmanız gerekir. Oluşturduğunuz dosyanın içerisine şunları yazın:

```php
<?php

return array(

        'driver' => 'file',

);
```

> **Not**: 'testing' adını ortam ismi olarak kullanmayın. Bu isim Unit Testing amacıyla rezerve edilmiştir.

Dikkat ederseniz, bu dosyada *bütün* değerleri yazmanıza gerek yok. Sadece üzerine yazmak istediklerinizi eklemeniz yeterli. Geri kalan değerler, öntanımlı yapılandırma değerlerinden alınacaktır.

Şimdi yapmamız gereken Laravel'e hangi ortamda çalıştığını belirtmek. Öntanımlı ortam daima `production` ortamıdır. Ancak ana dizindeki `bootstrap/start.php` dosya içerisine eklemeler yaparak farklı ortamlar oluşturmak mümkündür. Bu dosya içerisinde `$app->detectEnvironment` adında bir tanım bulacaksınız. Bu methoda eklenen bir parametre ile Laravel'e hangi ortamda çalıştığını belirtebilirsiniz. Hatta ihtiyacınız olursa, diğer ortam ve makine isimlerini de dizi olarak ekleyebilirsiniz:

```php
<?php

$env = $app->detectEnvironment(array(

    'local' => array('your-machine-name'),

));
```

Dilerseniz, `detectEnvironment` methoduna `Closure` ekleyip ortam algılama özelliğini kendiniz de yazabilirsiniz:

```php
$env = $app->detectEnvironment(function()
{
        return $_SERVER['MY_LARAVEL_ENV'];
});
```

Şuanki uygulama ortamına `environment` methoduyla erişebilirsiniz:

Şuanki Uygulama Ortamına Erişmek

```
$environment = App::environment();
```

Bakım Modu

Uygulamanız bakım modundayken, her istek için standart bir view gösterilir. Böylece uygulamanız güncellenirken, bir süreliğine uygulamayı "çalışmaz hale" getirebilirsiniz. Halihazırda `App::down` methoduna yapılan bir istek `app/start/global.php` dosyasında bulunmaktadır. Uygulamanız bakım modunda olduğunda, kullanıcılara bu metoddan dönen yanıt gönderilecektir.

Bakım modunu açmak için `down` komutunu Artisan üzerinde çalıştırın:

```
php artisan down
```

Bakım modunu kapatmak içinse, `up` komutunu çalıştırabilirsiniz:

```
php artisan up
```

Uygulamanız bakım modundayken kullanıcılara özel bir view göstermek için `app/start/global.php` dosyası içerisindeki `down` methodunu dilediğiniz gibi değiştirebilirsiniz:

```
App::down(function()
{
        return Response::view('bakim_sayfasi', array(), 503);
})
```

İstek Yaşam Döngüsü

Genel Bakış

Laravel'de İstek Yaşam Döngüsü (Request Lifecycle) gerçekten çok basit bir yapıdadır. Bir istek uygulamanıza girer ve uygun rota veya kontrolöre gönderilir. Bu rotadan gelen cevap daha sonra tarayıcıya geri gönderilir ve ekranda görüntülenir. Bazen, rotalarınız gerçekten çağrılmadan önce ya da sonra bazı işlemler yapmak isteyebilirsiniz. Laravel'de bunu yapmanın birkaç yolu vardır. Bu yollardan ikisi "start" dosyaları ve application olaylarıdır (events).

Start Dosyaları

Uygulamanızın start dosyaları `app/start` dizininde bulunmaktadır. Varsayılan olarak bunlardan üçü uygulamanızın içine dahil edilmiştir. Bunlar `global.php`, `local.php`, ve `artisan.php`'dir. `artisan.php` hakkında daha fazla bilgiye sahip olmak için Artisan komut satırı[54] dökümanlarına bakınız.

Bunlardan `global.php` start dosyası Günlüklerin[55] kayda geçirilmesi ve `app/filters.php` dosyanızın dahil edilmesi gibi ön tanımlı birkaç temel öğe içerir. Ancak, bu dosyaya istediğiniz her şeyi ekleyebilirsiniz. Bu dosya ortam ne olursa olsun uygulamanıza gelen *her* istekte otomatik olarak dahil edilecektir. Öte yandan `local.php` dosyası yalnızca uygulamanız `local` ortamda çalışırken çağrılır. Ortamlar hakkında daha fazla bilgi için Yapılandırma[56] belgelerine bakınız.

`local`'e ilaveten başka ortamlarınız da varsa, pek tabii bu ortamlar için de start dosyaları oluşturabilirsiniz. Uygulamanız o ortamda çalıştığı zaman bunlar otomatik olarak dahil edileceklerdir.

Application Olayları (Events)

Bunlara ek olarak `before`, `after`, `close`, `finish` ve `shutdown` uygulama olaylarını kayda geçirmek suretiyle istek öncesinde ve sonrasında bazı işlemler de yapabilirsiniz:

Uygulama Olaylarının Kayda Geçirilmesi

[54] /docs/commands#registering-commands
[55] /docs/errors
[56] /docs/configuration

```
App::before(function($request)
{
        //İstek öncesi olayları
});

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

Bu olay dinleyicileri, uygulamanıza yapılan her istek öncesinde (`before`) ve sonrasında (`after`) çalışacaktır.

Rotalar

Temel Rotalandırma

Uygulamanızdaki rotaların çoğu `app/routes.php` dosyasında tanımlanır. En basit Laravel rotası "URL deseni" ve "closure (anonim fonksiyon)"'dan oluşur.

Temel GET Rotası

```
Route::get('/', function()
{
        return 'Merhaba Laravel!';
});
```

Temel POST Rotası

```
Route::post('bir/sey', function()
{
        return 'Merhaba Laravel!';
});
```

Tüm HTTP Metodları (GET, POST gibi) İçin Rota Yazımı

```
Route::any('birsey', function()
{
        return 'Merhaba Laravel!';
});
```

Rotanın Zorunlu Olarak HTTPS Üzerinden Kullanılmasını Sağlamak

```
Route::get('birsey', array('https', function()
{
        return 'HTTPS üzerinde olmalı!';
}));
```

Çoğu zaman, rotalarınız için URL'ler üretmeniz gerekecek. Bunu `URL::to` metoduyla yapabilirsiniz:

```
$url = URL::to('birsey');
```

Rota Parametreleri

```
Route::get('kullanici/{id}', function($id)
{
        return 'Kullanıcı NO: '.$id;
});
```

İsteğe Bağlı Rota Parametreleri

```
Route::get('kullanici/{isim?}', function($isim = null)
{
        return $isim;
});
```

Öntanımlı Değerli İsteğe Bağlı Rota Parametreleri

```
Route::get('kullanici/{isim?}', function($isim = 'Ali')
{
        return $isim;
});
```

Rotalarda Düzenli İfade Kontrolü

```
Route::get('kullanici/{isim}', function($isim)
{
        //
})
->where('isim', '[A-Za-z]+');

Route::get('kullanici/{id}', function($id)
{
        //
})
->where('id', '[0-9]+');
```

Tabii ki kuralları bir dizi hâlinde tanımlayabilirsiniz:

```
Route::get('kullanici/{id}/{isim}', function($id, $isim)
{
        //
})
->where(array('id' => '[0-9]+', 'isim' => '[a-z]+'))
```

Rota Filtreleri

Rota filtreleri, sitenizin yetkilendirme gereken alanlarına erişimi kısıtlamak için uygun bir yoldur. Laravel'de `auth`, `auth.basic`, `guest`, `csrf` gibi `app/filters.php` dosyasında tanımlı filtreler vardır.

Rota Filtresi Tanımlama

```
Route::filter('yas', function()
{
        if (Input::get('yas') < 18)
        {
                return Redirect::to('anasayfa');
        }
});
```

Eğer filtreden bir yanıt (`Redirect::to` gibi) döndürülürse, bu cevap olarak kabul edilecek ve rota çalıştırılmayacaktır, rotada olabilecek `after` filtreleri de iptal edilecektir.

Rotaya Filtre Ekleme

```
Route::get('kullanici', array('before' => 'yas', function()
{
        return '18 yaş üzerisin!';
}));
```

Rotaya Birden Çok Filtre Ekleme

```
Route::get('user', array('before' => 'auth|yas', function()
{
        return '18 yaşın üzerisin ve giriş yetkin var!';
}));
```

Filtre Parametrelerini Belirtme

```
Route::filter('yas', function($rota, $istek, $deger)
{
        //
});

Route::get('kullanici', array('before' => 'yas:18', function()
{
        return 'Merhaba Laravel!';
}));
```

'After' filtreleri filtreye 3. parametre olarak geçilen bir `$yanit` parametresi alırlar.

```
Route::filter('log', function($rota, $istek, $yanit, $deger)
{
        //
});
```

Desenli Filtreler

URL desenine göre de rotalara filtre ataması yapabilirsiniz.

```
Route::filter('admin', function()
{
        //
});

Route::when('admin/*', 'admin');
```

Yukarıdaki örnekte, `admin` filtresi `admin/` ile başlayan tüm rotalara uygulanacaktır. * karakteri tüm karakterleri yakalamak için kullanılır.

Filtreleri HTTP metodlarına (GET, POST gibi) göre uygulayabilirsiniz.

```
Route::when('admin/*', 'admin', array('post'));
```

Filtre Sınıfları

Daha gelişmiş filtreler için closure yerine sınıfları kullanmak isteyebilirsiniz. Filtre sınıfları uygulama IoC Konteyneri[57]nde çözümlendikleri için, daha fazla test edilebilirlik için bu filtrelerde bağımlılık enjeksiyonu kullanmanız mümkün olabilecektir.

Filtre Sınıfı Oluşturma

[57] /docs/ioc

```
class BirSeyFiltresi {

        public function filter()
        {
                // Filtre işlemleri...
        }

}
```

Filtre Sınıfını Tanımlamak

```
Route::filter('birsey', 'BirSeyFiltresi');
```

İsimli Rotalar

İsimli rotalar link veya yönlendirme oluştururken kolaylık sağlar. Bir rotayı şöyle isimlendirebilirsiniz:

```
Route::get('kullanici/profil', array('as' => 'profil', function()
{
        //
}));
```

Denetçi metodları için de rota isimleri belirleyebilirsiniz:

```
Route::get('kullanici/profil', array('as' => 'profil', 'uses' => 'KullaniciContro\
ller@profilGoster'));
```

Şimdi, rota isimlerini link veya yönlendirme oluştururken kullanabilirsiniz:

```
$url = URL::route('profil');

$yonlendirme = Redirect::route('profil');
```

Çalışan rotanın ismine `currentRouteName` metoduyla ulaşabilirsiniz:

```
$isim = Route::currentRouteName();
```

Rota Grupları

Bazen bir grup rotaya filtre atamanız gerekebilir. Her birine ayrı filtre atamaktansa, rota gruplarını kullanabilirsiniz:

```
Route::group(array('before' => 'auth'), function()
{
        Route::get('/', function()
        {
                // Yetki gerekir. ("auth" filtresi)
        });

        Route::get('user/profile', function()
        {
                // Yetki gerekir. ("auth" filtresi)
        });
});
```

Alt Alanadı (Subdomain) Rotalandırması

Laravel rotaları ile alt-alanadlarını yakalayabilir ve parametre olarak kullanabilirsiniz.

Alt-alanadı Rotası Tanımlama

```
Route::group(array('domain' => '{hesapadi}.uygulamam.com'), function()
{

        Route::get('kullanici/{id}', function($hesapadi, $id)
        {
                //
        });

});
```

Rotalarda Ön-ek

`prefix` seçeneğini kullanarak gruptaki rotalara ön-ek ekleyebilirsiniz:

Gruplanmış Rotalara Ön-ek Ekleme

```
Route::group(array('prefix' => 'admin'), function()
{

        Route::get('kullanici', function()
        {
                //
        });

});
```

Rotalara Model Ataması

Model ataması model sınıflarının rotalarda kullanılması için kolaylık sağlar. Mesela, bir kullanıcının ID'sinin aktarılması yerine, ID ile eşleşen Kullanici modelini aktarabilirsiniz. İlk olarak, girilen parametreler için kullanılacak modelleri `Route::model` metoduyla belirleyin:

Parametrelere Model Atanması

```
Route::model('kullanici', 'Kullanici');
```

Daha sonra, `{kullanici}` parametresini içeren bir rota belirleyin:

```
Route::get('profil/{kullanici}', function(Kullanici $kullanici)
{
        //
});
```

`{kullanici}` parametresi ile `Kullanici` modelini eşleştirdiğimizden, bir `Kullanici` nesnesi rotaya aktarılacaktır. Yani, `profil/1` şeklindeki istek, ID'si 1 olan `Kullanici` nesnesini aktaracaktır.

> **Not**: Eğer model için veritabanında eşleşme yapılamazsa, 404 hatası fırlatılır.

Eğer eşleşmeme durumunda yapılacak işlemi kendiniz belirlemek istiyorsanız, `model` metoduna 3. argüman olarak bir closure ekleyebilirsiniz:

```
Route::model('kullanici', 'Kullanici', function()
{
        throw new NotFoundException;
});
```

Modeller yerine kendi tanımlayıcınızı kullanmak isteyebilirsiniz. Bunun için `Route::bind` metodu kullanılır:

```
Route::bind('kullanici', function($deger, $rota)
{
        return Kullanici::where('isim', $deger)->first();
});
```

404 Hatası Fırlatma

404 hatasını tetiklemenin iki yolu vardır. İlki, `App::abort` metodu.

```
App::abort(404);
```

İkinci, `Symfony\Component\HttpKernel\Exception\NotFoundHttpException` nesnesi oluşturmaktır. 404 hatalarının yakalanması ve özel yanıtla oluşturulması hakkında daha fazla bilgiye dokümantasyonun hatalar[58] bölümünden ulaşabilirsiniz.

Denetçilere Rotalama

Laravel sadece closure'lara değil, aynı zamanda denetçi sınıflarına rotalamaya da imkan verir ve hatta kaynak denetçileri[59] oluşturulması da mümkündür.

Daha fazla bilgi için Denetçiler[60] konusunu inceleyin.

[58] /docs/errors#handling-404-errors
[59] /docs/controllers#resource-controllers
[60] /docs/controllers

İstekler (Requests) ve Girdi (Input)

Basit Girdi

Tüm kullanıcı girdisine birkaç basit metodla erişebilirsiniz. İstek için kullanılmış olan HTTP eylemi için endişe etmenize gerek yoktur, bütün eylemler için girdi bilgisine erişim aynıdır.

Bir Girdi Değerinin Öğrenilmesi

```
$ismi = Input::get('ismi');
```

Bir Girdi Değerinin (Eksik Olması Durumunda Varsayılacak Olan Bir "Ön Değer" Belirtilerek) Öğrenilmesi

```
$ismi = Input::get('ismi', 'Saliha');
```

Bir Girdi Değerinin Mevcut Olduğunun Test Edilmesi

```
if (Input::has('ismi'))
{
        //
}
```

İstekteki Tüm Girdi Değerlerinin Birden Öğrenilmesi

```
$girdi = Input::all();
```

İstek Girdisinin Sadece Bazı Değerlerinin Öğrenilmesi

```
$girdi = Input::only('kullaniciadi', 'sifre');        //sadece belirtilenler

$girdi = Input::except('kredi_karti');        //belirtilenler hariç
```

Bazı JavaScript kütüphaneleri, örneğin Backbone, girdi bilgisini uygulamaya JSON olarak gönderir. Bu girdi verisine de yine normal şekilde `Input::get` ile erişebilirsiniz.

Çerezler (Cookies)

Laravel çerçevesi tarafından oluşturulan tüm çerezler, bir kimlik doğrulama kodu ile şifrelenir ve imzalanır. Kullanıcı tarafından değiştirilmesi halinde geçersiz kabul edilecektir.

Bir Çerez Değerinin Öğrenilmesi

```
$deger = Cookie::get('ismi');
```

Yanıta(Response) Yeni Bir Çerez İliştirilmesi

```
$yanıt= Response::make('Merhaba Dünya');

$yanıt->withCookie(Cookie::make('ismi', 'degeri', $dakika));
```

Süresiz Bir Çerez Oluşturulması

```
$cerez = Cookie::forever('ismi', 'degeri');
```

Önceki Girdi

Bazı durumlarda bir isteğin girdisini bir sonraki isteğe kadar tutmanız gerekebilir. Örneğin, doğrulama hataları için kontrol ettikten sonra bir formu yeniden bu önceki girdi bilgisi ile doldurmak gerekebilir.

Girdinin Oturuma(Session) Geçici Olarak Yansıtılması (flash)

```
Input::flash();
```

Girdinin Sadece Bazı Değerlerinin Oturuma Geçici Olarak Yansıtılması

```
Input::flashOnly('kullaniciadi', 'email');        //sadece belirtilenler

Input::flashExcept('sifre');        //belirtilenler hariç
```

Girdinin geçici olarak oturuma yansıtılmasını, sık şekilde bir önceki sayfaya tekrar-yönlendirme (redirect) ile birlikte yapacağınız için, bu yansıtmayı (redirect)'e zincir ek yapabilirsiniz.

```
return Redirect::to('form')->withInput();        //tüm girdi değerleri ile beraber

return Redirect::to('form')->withInput(Input::except('sifre'));        //belirtilenler h\
ariç
```

> **Not**: Diğer verilerin istekler arasında geçici yansıtmasını (flash), Oturum Session[61] sınıfını kullanarak yapabilirsiniz.

Önceki Girdi Verisinin Elde Edilmesi

[61] /docs/session

```
Input::old('kullaniciadi');
```

Dosyalar

Yollanan Bir Dosyanın Öğrenilmesi

```
$dosya = Input::file('foto');
```

Bir Dosya Yollanmış Olup Olmadığının Belirlenmesi

```
if (Input::hasFile('foto'))
{
        //
}
```

Dosya `file` metodu tarafından döndürülen nesne (object), PHP `SplFileInfo` sınıfının bir uzantısı olan `Symfony\Component\HttpFoundation\File\UploadedFile` sınıfının bir "üyesidir", ve bu sayede dosya ile etkileşim için çeşitli metodlar sağlar.

Yüklenmiş Olan Bir Dosyanın Taşınması

```
Input::file('foto')->move($hedefDizinPatikasi);

Input::file('foto')->move($hedefDizinPatikasi, $dosyaAdi);
```

Yüklenmiş Olan Bir Dosyanın Dosya Yolunun Öğrenilmesi

```
$patika = Input::file('foto')->getRealPath();
```

**Yüklenmiş Olan Bir Dosyanın Orijinal Adının Öğrenilmesi

```
$name = Input::file('foto')->getClientOriginalName();
```

Yüklenmiş Olan Bir Dosyanın Uzantısının Öğrenilmesi

```
$uzanti = Input::file('foto')->getClientOriginalExtension();
```

Yüklenmiş Olan Bir Dosyanın Boyutunun Öğrenilmesi

```
$buyukluk = Input::file('foto')->getSize();
```

Yüklenmiş Olan Bir Dosyanın MIME Türünün Öğrenilmesi

```
$mime = Input::file('foto')->getMimeType();
```

İstek Bilgileri

İstek `Request` sınıfı, uygulamanıza gelecek olan HTTP isteğini incelemeniz için birçok metod sunar ve `Symfony\Component\HttpFoundation\Request` sınıfının bir uzantısıdır. Bunlardan bazıları şöyledir.

İstek URI'nın Öğrenilmesi

```
$uri = Request::path();
```

İstek Patikasının Bir Şablona Uygunluğunun Test Edilmesi

```
if (Request::is('admin/*'))
{
        //
}
```

İstek URL'nin Öğrenilmesi

```
$url = Request::url();
```

İstek URI'nın Herhangi Bir Bölümünün Öğrenilmesi

```
$segment = Request::segment(1);
```

Bir İstek Başlığı(Header) Değerinin Öğrenilmesi

```
$deger = Request::header('Content-Type');
```

Sunucu bilgileri için $_SERVER Değerlerinin Öğrenilmesi

```
$deger = Request::server('PATH_INFO');
```

İsteğin AJAX Kullandığının Test Edilmesi

```
if (Request::ajax())
{
        //
}
```

İsteğin HTTPS Üzerinden Olduğunun Test Edilmesi

```
if (Request::secure())
{
        //
}
```

Görünümler ve Cevaplar (Views & Responses)

Basit Cevaplar

Rotalardan String Döndürme

```
Route::get('/', function()
{
        return 'Merhaba dünya!';
});
```

Özel Cevaplar Oluşturma

Bir cevap (`Response`) olgusu `Symfony\Component\HttpFoundation\Response` sınıfından türer ve HTTP cevapları oluşturmak için çeşitli metodlar sağlar.

```
$cevap = Response::make($contents, $statusCode);

$cevap->header('Content-Type', $deger);

return $cevap;
```

Cevaplara Çerez Bağlanması

```
$cerez = Cookie::make('isim', 'deger');

return Response::make($content)->withCookie($cerez);
```

Yön Değiştirtmeler (Redirects)

Bir Yön Değiştirtme Döndürme

```
return Redirect::to('uye/giris');
```

Flaş Veri Eşliğinde Bir Yön Değiştirtme Döndürme

```
return Redirect::to('uye/giris')->with('mesaj', 'Giriş başarısız!');
```

İsimli Bir Rotaya Yön Değiştirme Döndürme

```
return Redirect::route('giris');
```

Parametre Geçerek İsimli Bir Rotaya Yön Değiştirme Döndürme

```
return Redirect::route('profil', array(1));
```

İsimli Parametre Kullanarak İsimli Bir Rotaya Yön Değiştirme Döndürme

```
return Redirect::route('profil', array('uye' => 1));
```

Bir Kontrolör Eylemine Yön Değiştirme Döndürme

```
return Redirect::action('HomeController@index');
```

Parametre Geçerek Bir Kontrolör Eylemine Yön Değiştirme Döndürme

```
return Redirect::action('UserController@profil', array(1));
```

İsimli Parametre Kullanarak Bir Kontrolör Eylemine Yön Değiştirme Döndürme

```
return Redirect::action('UserController@profil', array('uye' => 1));
```

Görünümler (Views)

Görünümler tipik olarak uygulamanızın HTML'sini içerirler ve kontrolörünüzün ve etki alanı mantığınızın gösterim mantığınızdan ayrı tutulmasının uygun bir yoludur. Görünümler `app/views` dizininde saklanmaktadır.

Basit bir görünüm şuna benzer:

```
<!-- Görünüm app/views/selamlama.php dosyasında bulunsun-->

<html>
        <body>
                <h1>Merhaba <?php echo $isim; ?></h1>
        </body>
</html>
```

Bu görünüm web tarayıcısına şu şekilde döndürülebilir:

```
Route::get('/', function()
{
        return View::make('selamlama', array('isim' => 'Tuana Şeyma'));
});
```

`View::make` metodundaki ikinci parametre görünümde kullanılması gereken bir veri dizisidir.

Görünümlere Veri Geçilmesi

Dilerseniz, `make` metoduna ikinci parametre olarak veriler dizisi geçebilirsiniz:

```
$view = View::make('selamlama', $dizi);

$view = View::make('selamlama')->with('isim', 'Tuana Şeyma');
```

Yukarıdaki örnekte `$isim` değişkeni görünümden erişilebilir olacak ve `Tuana Şeyma` bilgisini taşıyacaktır.

Bir parça veriyi tüm görünümler arasında paylaşmanız da mümkündür:

```
View::share('isim', 'Tuana Şeyma');
```

Bir Görünüme Bir Alt Görünüm Geçirilmesi

Bazen bir görünümü başka bir görünümün içine geçirmek isteyebilirsiniz. Örneğin, `app/views/evlat/view.php`'de saklanan belli bir görünüm olsun ve biz bunu şu şekilde başka bir görünüme geçirebiliriz:

```
$view = View::make('selamlama')->nest('evlat', 'evlat.view');

$view = View::make('selamlama')->nest('evlat', 'evlat.view', $veri);
```

Bundan sonra bu alt görünüm ebeveyn görünümde gösterilebilir:

```
<html>
        <body>
                <h1>Merhaba!</h1>
                <?php echo $evlat; ?>
        </body>
</html>
```

Görünüm Kompozitörleri

Görünüm kompozitörleri görünüm oluşturulduğu zaman çağrılan bitirme fonksiyonları veya sınıf metodlarıdır. Eğer belli bir görünüm, uygulamanız boyunca her oluşturulduğunda bu görünüme bağlamak istediğiniz bir veri varsa, bir görünüm kompozitörü kodun tek bir yere koyulabilmesi imkanı verebilir. Bu nedenle, görünüm kompozitörleri "görünüm modelleri" veya "sunum yapıcı" gibi iş görürler.

Bir Görünüm Kompozitörü Tanımlanması

```
View::composer('profil', function($view)
{
        $view->with('navigasyon', Sayfa::all());
});
```

Şimdi `profil` görünümü her oluşturulduğunda, `navigasyon` verisi bu görünüme bağlanacaktır.

Bir görünüm kompozitörüne bir defada birden çok görünüm bağlamanız da mümkündür:

```
View::composer(array('profil','pano','bildirim'), function($view)
{
    $view->with('navigasyon', Sayfa::all());
});
```

> **Not**: Görünüm Kompozitörlerini tüm sayfalara bağlamanız gerektiğinde, dizi olarak tüm sayfaları tek tek vermek yerine, `layout` olarak kullandığınız görünümünü verebilirsiniz. Böylece o layoutu extend eden her sayfa otomatik olarak kompozitörden gelen verilere ulaşabillir.

Bunun yerine sınıf tabanlı bir kompozitör kullanmak isterseniz, ki uygulama IoC Konteyneri[62] ile çözümlenebilme yararı sağlar, şöyle yapabilirsiniz:

[62] /docs/ioc

```
View::composer('profil', 'ProfileComposer');
```

Bir görünüm kompozitörü sınıfı şöyle tanımlanmalıdır:

```
class ProfileComposer {

        public function compose($view)
        {
                $view->with('adet', Uye::count());
        }

}
```

Kompozitör sınıfının nerede saklanacağı konusunda bir adet olmadığına dikkat edin. `composer.json` dosyanızdaki yönergeleri kullanarak otomatik yüklenebildikleri sürece, bunları istediğiniz yerde depolayabilirsiniz.

Görünüm Oluşturucular

Görünüm **oluşturucuları* tam olarak görünüm kompozitörleri gibi çalışırlar; ancak bunlar görünüm oluşturulur oluşturulmaz aktifleştirilirler. Görünüm oluşturucusu kaydetmek için, basitçe `creator` metodunu kullanınız:

```
View::creator('profil', function($view)
{
        $view->with('adet', Uye::count());
});
```

Özel Cevaplar

Bir JSON Cevabı Oluşturma

```
return Response::json(array('isim' => 'Tuana Şeyma', 'il' => 'Bursa'));
```

Bir JSONP Cevabı Oluşturma

```
return Response::json(array('isim' => 'Tuana Şeyma', 'il' => 'Bursa'))->setCallba\
ck(Input::get('callback'));
```

Bir Dosya İndirme Cevabı Oluşturma

```php
return Response::download($indirilecekDosyaYolu);

return Response::download($indirilecekDosyaYolu, $isim, $basliklar);
```

Denetçiler (Controllers)

Temel Denetçiler

Bütün rotalandırma mantığını, tüm rotaları tek tek `routes.php` dosyasında tanımlamak yerine, bu davranışlarını Denetçiler (Controllers) sınıflarını kullanarak organize edebilirsiniz. Denetçiler, ilişkin oldukları rotaların mantığını bir sınıfta gruplar. Aynı zamanda, daha ileri çerçeve (framework) özelliklerini kullanma avantajına sahiptirler, örneğin otomatik dependency injection[63] (bağımlılık enjeksiyonu) gibi.

Denetçiler genelde `app/controllers` dizininde konumlandırılır ve `composer.json` dosyanızın sınıf haritası `classmap` seçeneğinde, varsayılan olarak bu dizin belirlenmiştir.

Basit bir denetçi (controller) sınıfı örneği şöyledir:

```
1  class KullaniciController extends BaseController {
2
3          /**
4           * Verilen kullanıcının profilini göster.
5           */
6          public function showProfile($id)
7          {
8                  $kullanici = Kullanici::find($id);
9
10                 return View::make('kullanici.profil', array('kullanici' => $kullanici));
11         }
12
13 }
```

Bütün denetçilerin `BaseController` sınıfından türetilmiş olması gerekir. `BaseController` ın kendisi de `app/controllers` dizininde bulunur ve bütün denetçiler için geçerli olacak ortak mantığın içine yerleştirilmesinde kullanılabilir. `BaseController` sınıfı, çerçevenin `Controller` sınıfının uzantısıdır. Bu durumda, oluşturmuş olduğumuz denetçi fonksiyonuna rotalandırmayı şu şekilde yapabiliriz:

```
1  Route::get('kullanici/{id}', 'KullaniciController@showProfile');
```

Eğer bir denetçinizi, dizin içerisinde yuvalandırarak (nest) veya PHP isim-alanları (namespaces) kullanarak organize etmek isterseniz, bu durumda rotayı tanımlarken, tam nitelendirilmiş (fully qualified) sınıf adını kullanınız:

[63] /docs/ioc

```
Route::get('falanca', 'Namespace\FalancaController@yontemAdi');
```

Denetçi rotalarına isimler de verebilirsiniz:

```
Route::get('falanca', array('uses' => 'FalancaController@yontemAdi',
        'as' => 'rotaAdi'));
```

Herhangi bir denetçi eylemine ait bir URL üretmek için, `URL::action` metodunu kullanabilirsiniz:

```
$url = URL::action('FalancaController@yontemAdi');
```

Çalıştırılmakta olan bir denetçi eyleminin ismine `currentRouteAction` metodu ile erişebilirsiniz:

```
$action = Route::currentRouteAction();
```

Denetçi Filtreleri

Denetçi rotalarına, diğer rotalarda olduğuna benzer şekilde, filtreler Filters[64] belirlenebilir:

```
Route::get('profile', array('before' => 'auth',
                        'uses' => 'KullaniciController@showProfile'));
```

Filtreleri, denetçinizin içerisinden de belirtebilirsiniz:

```
class KullaniciController extends BaseController {

        /**
         * Yeni bir KullaniciController sureti (instance) oluştur. (new KullaniciControl\
ler)
         */
        public function __construct()
        {
                $this->beforeFilter('auth', array('except' => 'getGiris'));

                $this->beforeFilter('csrf', array('on' => 'post'));

                $this->afterFilter('log', array('only' =>
                        array('falancaYontem', 'filancaYontem')));
        }

}
```

Controller filtrelerini bir Closure kullanarak da belirtebilirsiniz:

[64] /docs/routing#route-filters

```
class KullaniciController extends BaseController {

        /**
         * Yeni bir KullaniciController sureti (instance) oluştur. (new KullaniciControl\
ler)
         */
        public function __construct()
        {
                $this->beforeFilter(function()
                {
                        //
                });
        }

}
```

TEDA-uyumlu (Temsili Durum Aktarma uyumlu, RESTful) Denetçiler

Laravel size, basit TEDA (REST) isimlendirme tüzüklerini (naming conventions) kullanarak, belirleyeceğiniz tek bir rota ile, denetçilerinizin içindeki her eylemi kullanabilme imkanını tanır. İlk olarak, (rota : denetçi) `Route::controller` metodu ile bu rotayı tanımlayınız:

TEDA-uyumlu Bir Denetçi Oluşturulması

```
Route::controller('kullanicilar', 'KullaniciController');
```

`controller` (denetçi) metodu iki argüman alır. Birincisi denetçinin yöneteceği baz URI olup, ikincisi denetçinin sınıf ismidir. Akabinde sadece, isimlerine HTTP eyleminin ön ek olarak ekleneceği ve bunlara cevap verecek olan metodlarınızı denetçinize ilave ediniz:

```
class KullaniciController extends BaseController {

        public function getIndex()
        {
                //
        }

        public function postProfile()
        {
                //
```

```
11          }
12
13  }
```

`index` (fihrist) metodları, denetçi tarafından yönetilmekte olan kök URI 'a cevap verir. Örneğimizde bu, `kullanicilar` dır.

Denetçinizdeki bir eylem metodunun ismi birden fazla kelimeden oluşuyorsa, bu eylem metoduna, URI da kelime aralarına tire işareti "-" eklenmiş şekilde yazarak erişebilirsiniz. Örneğin, 'Kullanici-Controller' denetçimizdeki aşağıdaki şekilde isimlendirilmiş olan metod, `kullanici/yonetici-profili` URI 'na cevap verecektir.

```
1  public function getYoneticiProfili() {}
```

Kaynak (Resource) Denetçileri

Kaynak denetçileri, kaynaklar etrafında TEDA-uyumlu denetçiler oluşturulmasını kolaylaştırır. Artisan KSA'daki (Artisan Komut Satırı Arayüzü) `controller:make` komutunu ve de `Route::resource` metodunu kullanmak sureti ile böyle bir denetçiyi çabucak oluşturabiliriz.

Denetçiyi komut satırını kullanarak oluşturmak için şu komutu kullanınız:

```
1  php artisan controller:make FotoController
```

Bu denetçinin TEDA-uyumlu rotasını (routes.php) dosyasında kayıt ettiriniz:

```
1  Route::resource('foto', 'FotoController');
```

Bu tek bir rota deklarasyonu, foto kaynağınız üzerinde çalıştıracağınız çeşitli TEDA-uyumlu eylem metodlarına erişeceğiniz rotalar oluşturur. Aynı zamanda, oluşturulmuş olan denetçide, bu eylemlerin her biri için metodları hazır olarak oluşturulmuş ve hangi URI'ı ve eylemi yönettikleri yanlarına not olarak yazılmış olacaktır.

Kaynak Denetçisinin Yöneteceği Eylemler

```
HTTP Fiili | Patika                | Eylem            | Rota İsmi
-----------|-----------------------|------------------|---------------
GET        | /kaynak               | index            | kaynak.index
GET        | /kaynak/create        | create (oluştur) | kaynak.create
POST       | /kaynak               | store (kaydet)   | kaynak.store
GET        | /kaynak/{id}          | show (göster)    | kaynak.show
GET        | /kaynak/{id}/edit     | edit (düzenle)   | kaynak.edit
PUT/PATCH  | /kaynak/{id}          | update (güncelle)| kaynak.update
DELETE     | /kaynak/{id}          | destroy (imha et)| kaynak.destroy
```

Bazen bu eylemlerin sadece bazılarına ihtiyaç duyabilirsiniz:

```
php artisan controller:make FotoController --only=index,show   //sadece belirtile\
nleri

php artisan controller:make FotoController --except=index     //belirtilenler har\
iç
```

Ve, rotasında da eylemlerin sadece bazılarını yönetmesini belirleyebilirsiniz:

```
Route::resource('foto', 'FotoController',
                array('only' => array('index', 'show')));

Route::resource('photo', 'PhotoController',
                array('except' => array('create', 'store', 'update', delete')));
```

Eksik Olan Metodların Yönetilmesi

Denetçide tanımlanmamış olan metodlara gelecek olan çağrıları yönetmek için bir "hepsini-yakala" metodu tanımlanabilir. Bu metodun isminin `missingMethod` (eksik metod) olması gerekir ve tek argümanı olarak gelen isteğin (request) parametrelerini alır:

Bir Hepsini-Yakala Metodunun Tanımlanması

```
public function missingMethod($parameters)
{
        //
}
```

Hatalar ve Günlüğe Ekleme

Hata Ayrıntısı

Ön tanımlı olarak hata ayrıntısı uygulamanızda etkindir. Yani bir hata oluştuğu zaman ayrıntılı bir sorun listesi ve hata iletisi gösterebileceksiniz. `app/config/app.php` dosyanızdaki `debug` seçeneğini `false` ayarlayarak hata ayrıntılarını devre dışı bırakabilirsiniz.

> **Not**: Bir üretim (production) ortamında hata ayrıntılarını devre dışı bırakmanız şiddetle önerilir.

Hataların İşlenmesi

Ön tanımlı olarak, `app/start/global.php` dosyasında tüm istisnalar için bir hata işleyici bulunmaktadır:

```
App::error(function(Exception $exception)
{
        Log::error($exception);
});
```

En temel hata işleyici budur. Ancak siz gerektiği kadar işleyici belirleyebilirsiniz. İşleyicilere işledikleri istisnaların tipine işaret eden isimler verilir. Örneğin, sadece `RuntimeException` olgularını işleyen bir işleyici oluşturabilirsiniz:

```
App::error(function(RuntimeException $exception)
{
        // İstisnayı işle...
});
```

Bir istisna işleyicisinin bir cevap döndürmesi halinde, bu cevap tarayıcıya gönderilecek ve başka bir hata işleyici çağrılmayacaktır:

```
App::error(function(InvalidUserException $exception)
{
        Log::error($exception);

        return 'Maalesef bu hesapla ilgili yanlış bir şeyler var!';
});
```

PHP'nin önemli hatalarını (fatal error) izlemek için, `App::fatal` metodunu kullanabilirsiniz:

```
App::fatal(function($exception)
{
        //
});
```

HTTP İstisnaları

HTTP istisnaları bir istemci isteği sırasında oluşabilecek hatalar demektir. Bu bir sayfa bulunamadı hatası (404), bir yetkisizlik hatası (401), hatta genel 500 hatası olabilir. Böyle bir cevap döndürmek için aşağıdaki biçimi kullanın:

```
App::abort(404, 'Sayfa bulunamadı');
```

Buradaki ilk parametre HTTP durum kodu, ikinci parametre ise bu hata durumunda göstermek istediğiniz özel bir mesajdır.

401 Yetkisizlik istisnası çıkarmak için tek yapacağınız şudur:

```
App::abort(401, 'Yetkili değilsiniz.');
```

Bu istisnalar, isteğin yaşam döngüsü boyunca her an çalışabilecektir.

404 Hatalarının İşlenmesi

Uygulamanızdaki tüm "404 Not Found" hatalarını işleyerek özel 404 hata hata sayfaları döndürmenize imkan veren bir hata işleyici kaydı yapabilirsiniz:

```
App::missing(function($exception)
{
        return Response::view('errors.missing', array(), 404);
});
```

Günlüğe Ekleme

Laravel'in günlüğe ekleme imkanlanları güçlü Monolog[65] üstünde basit bir katman sağlar. Laravel, ön tanımlı olarak uygulamanız için günlük dosyaları oluşturacak şekilde yapılandırılmıştır ve bu dosyalar `app/storage/logs` klasörü içinde tutulmaktadır. Bu dosyalara aşağıdakilere benzer şekilde bilgi yazabilirsiniz:

```
Log::info('İşte bu yararlı bir bilgidir.');

Log::warning('Yanlış giden bir şeyler olabilir.');

Log::error('Gerçekten yanlış giden bir şey var.');
```

Günlük tutucu, RFC 5424[66]'de tanımlanmış yedi günlük ekleme düzeyi sağlamaktadır: **debug**, **info**, **notice**, **warning**, **error**, **critical** ve **alert**.

Monolog, günlüğe ekleme için kullanabileceğiniz bir takım başka işleyicilere de sahiptir. Gerektiğinde, Laravel tarafından kullanılan Monolog olgusuna şu şekilde ulaşabilirsiniz:

```
$monolog = Log::getMonolog();
```

Ayrıca, günlüğe geçirilen tüm mesajları yakalamak için bir olay kaydı da yapabilirsiniz:

Bir günlük izleyici kaydı yapılması

```
Log::listen(function($level, $message, $context)
{
        //
});
```

[65] http://github.com/seldaek/monolog
[66] http://tools.ietf.org/html/rfc5424

Önbellekleme (Cache)

Yapılandırma

Laravel, çeşitli önbellekleme sistemleri için tümleşik bir API sağlar. Önbellekleme yapılandırma ayarları `app/config/cache.php` dosyasında bulunmaktadır. Bu dosyada uygulamanızda varsayılan olarak hangi önbellekleme sürücüsünü kullanmak istediğinizi belirtebilirsiniz. Laravel, Memcached[67] ve Redis[68] gibi popüler önbellekleme sürücülerini barındırır.

Önbellekleme yapılandırma dosyası ayrıca dosyanın içinde açıklanmış çeşitli seçenekleri de içerir, bu yüzden o seçenekleri de okuduğunuzdan emin olun. Varsayılan olarak, Laravel, sıralanarak önbelleklenmiş öğeleri dosya sisteminde depolayan `file` (dosya) önbellekleme sürücüsünü kullanmak üzere ayarlanmıştır. Daha büyük uygulamalar için, Memcached ve APC gibi bir önbellekleme uygulaması kullanmanız önerilir.

Önbellekleme Kullanımı

Bir Öğeyi Önbelleğe Koymak

```
1  Cache::put('key', 'value', $minutes);
```

Eğer Öğe Önbellekte Yoksa, Öğeyi Önbelleğe Koymak

```
1  Cache::add('key', 'value', $minutes);
```

Öğenin Önbellekte Var Olup Olmadığını Kontrol Etmek

```
1  if (Cache::has('key'))
2  {
3          //
4  }
```

Önbellekten Bir Öğeyi Almak

[67] http://memcached.org
[68] http://redis.io

```
$value = Cache::get('key');
```

Bir Önbellek Değeri Almak Veya Varsayılan Bir Değer Döndürmek

```
$value = Cache::get('key', 'varsayılanDeğer');

$value = Cache::get('key', function() { return 'varsayılanDeğer'; });
```

Bir Öğeyi Kalıcı Olarak Önbelleğe Koymak

```
Cache::forever('key', 'value');
```

Bazen, önbellekten bir öğeyi almak isteyebilir ve ayrıca talep edilen öğe yoksa önbellekte varsayılan bir değer saklayabilirsiniz. Bunu, `Cache::remember` metodunu kullanarak yapabilirsiniz:

```
$value = Cache::remember('kullanicilar', $minutes, function()
{
        return DB::table('kullanicilar')->get();
});
```

Ayrıca, `remember` ve `forever` methodlarını birlikte kullanabilirsiniz.

```
$value = Cache::rememberForever('kullanicilar', function()
{
        return DB::table('kullanicilar')->get();
});
```

Önbellekte bütün öğelerin sıralanmış şekilde saklandığını unutmayın, yani her türlü veriyi saklayabilirsiniz.

Önbellekten Bir Öğeyi Silmek

```
Cache::forget('key');
```

Arttırma & Azaltma

`file` ve `database` hariç tüm sürücüler `increment` (artma) ve `decrement` (azalma) işlemlerini destekler:

Bir Değeri Arttırmak

```
Cache::increment('key');

Cache::increment('key', $miktar);
```

Bir Değeri Azaltmak

```
Cache::decrement('key');

Cache::decrement('key', $miktar);
```

Önbellek Bölümleri

> **Not**: Önbellek bölümleri `dosya` ve `veritabanı` önbellekleme sürücüleri kullanılırken desteklenmemektedir.

Önbellek bölümleri, önbellekteki ilişkili öğeleri gruplamanıza ve tüm bölümü temizlemenize olanak sağlar. Bölüme erişim için `section` metodu kullanılır:

Bir Önbellek Bölümününe Erişim

```
Cache::section('insanlar')->put('Mehmet', $mehmet, $dakika);

Cache::section('insanlar')->put('Şeyma', $ayse, $dakika);
```

Ayrıca bölümlerde önbelleklenmiş öğelere, diğer önbellek metodlarında olduğu gibi `increment` ve `decrement` ile de erişebilirsiniz.

Önbellek Bölümündeki Öğelere Erişmek

```
$anne = Cache::section('insanlar')->get('Mehmet');
```

Önbellek bölümünü bu şekilde temizleyebilirsiniz:

```
Cache::section('insanlar')->flush();
```

Veritabanı Önbelleği

Veritabanı önbellek sürücüsü kullanırken, önbellek öğelerini içeren bir tablo kurulumu gerekir. Bu tablo için örnek bir şema aşağıda gösterilmiştir:

```php
Schema::create('cache', function($table)
{
        $table->string('key')->unique();
        $table->text('value');
        $table->integer('expiration');
});
```

Olaylar (Events)

Basit Kullanım

Laravel'in `Event` sınıfı, uygulamanızdaki olaylara abone olmanıza ve dinlemenize imkan veren basit bir gözlemci aracıdır.

Bir Olaya Abone Olma

```
Event::listen('uye.login', function($uye)
{
        $uye->last_login = new DateTime;

        $uye->save();
});
```

Bir Olayı Ateşleme

```
$olay = Event::fire('uye.login', array($uye));
```

Olaylara abone olurken bir öncelik de belirtebilirsiniz. Daha yüksek önceliği olan dinleyiciler daha önce çalışacak, aynı önceliğe sahip dinleyiciler ise abonelik sırasına göre çalışacaklardır.

Bir Olaya Abone Olurken Öncelik Belirtme

```
Event::listen('uye.login', 'LoginHandler', 10);

Event::listen('uye.login', 'DigerHandler', 5);
```

Bazen bir olayın diğer dinleyicilere yayılmasını durdurmak isteyebilirsiniz. Dinleyicinizden `false` döndürerek bunu gerçekleştirebilirsiniz:

Bir Olayın Yayılımının Durdurulması

```
Event::listen('uye.login', function($event)
{
        // Olayı işle...

        return false;
});
```

Joker Dinleyiciler

Bir olay dinleyiciyi kayda geçirirken, joker dinleyicileri belirtmek üzere yıldız işareti kullanabilirsiniz:

Joker Olay Dinleyicilerin Kayda Geçirilmesi

```
Event::listen('falan.*', function($param, $event)
{
        // Olayı işle...
});
```

Bu dinleyici `falan.` ile başlayan tüm olayları işleyecektir. Tam olay adının işleyiciye son parametre olarak geçildiğine dikkat ediniz.

Dinleyici Olarak Sınıfları Kullanma

Bazı durumlarda, bir olayı işlemek için bir bitirme fonksiyonu yerine bir sınıf kullanmak isteyebilirsiniz. Sınıf olay dinleyileri Laravel'in IoC konteyneri[69] ile çözümlenecek, böylece size dinleyicileriniz üzerinde tam bir koloni enjeksiyonu gücü verecektir.

Bir Sınıf Dinleyicinin Kayda Geçirilmesi

```
Event::listen('uye.login', 'LoginIsleyici');
```

Ön tanımlı olarak, `LoginHandler` sınıfındaki `handle` metodu çağrılacaktır:

Bir Olay Dinleyici Sınıfının Tanımlanması

[69] /docs/ioc

```
class LoginIsleyici {

        public function handle($data)
        {
                //
        }

}
```

Eğer ön tanımlı `handle` metodunu kullanmak istemiyorsanız, abone olunacak metodu belirleyebilirsiniz:

Hangi Metoda Abone Olunduğunun Tanımlanması

```
Event::listen('uye.login', 'LoginIsleyici@onLogin');
```

Olayları Sıraya Sokma

`queue` ve `flush` metodlarını kullanarak, bir olayı hemen ateşlemeyip, ateşlenmek üzere "sıraya" sokabilirsiniz:

Sıralı Bir Olayın Kayda Geçirilmesi

```
Event::queue('falan', array($uye));
```

Bir Olay Flusher'ın Kayda Geçirilmesi

```
Event::flusher('falan', function($uye)
{
        //
});
```

Son olarak, ilgili "flusher"ı çalıştırabilir ve `flush` metodunu kullanarak sıradaki tüm olayları harekete geçirebilirsiniz:

```
Event::flush('falan');
```

Olay Aboneleri

Olay aboneleri, sınıfın kendi içinden birden çok olaya abone olabilen sınıflardır. Aboneler bir `subscribe` metodu ile tanımlanırlar ve bu metoda parametre olarak bir olay sevkiyatçısı olgusu geçilecektir:

Bir Olay Abonesi Tanımlanması

```
class UyeOlayIsleyici {

        /**
         * Uye login olaylarını işle.
         */
        public function onUyeLogin($event)
        {
                //
        }

        /**
         * Uye logout olaylarını hallet.
         */
        public function onUyeLogout($event)
        {
                //
        }

        /**
         * Abone dinleyicilerini kayda geçir.
         *
         * @param  Illuminate\Events\Dispatcher  $events
         * @return array
         */
        public function subscribe($events)
        {
                $events->listen('uye.login', 'UyeOlayIsleyici@onUyeLogin');

                $events->listen('uye.logout', 'UyeOlayIsleyici@onUyeLogout');
        }

}
```

Abone tanımlandıktan sonra, `Event` sınıfı kullanılarak kayda geçirilebilir.

Bir Olay Abonesinin Kayda Geçirilmesi

```
$abone = new UyeOlayIsleyici;

Event::subscribe($abone);
```

Frameworkün Genişletilmesi

Giriş

Laravel, frameworkün çekirdek bileşenlerinin davranışlarını isteğinize göre özelleştirebilmeniz, hatta tümden değiştirebilmeniz için size birçok genişletme noktası sağlar. Örneğin, hash yapma araçları bir `HasherInterface` sözleşmesi ile sağlanmış olup, kendi uygulamanızın gereksinimlerine dayalı olarak implemente edebilirsiniz. Ayrıca, sizin kendi uygun "helper" metodlarınızı eklemenize imkan vermek üzere `Request` nesnesini de genişletebilirsiniz. Hatta tamamen yeni kimlik doğrulama, cache ve session sürücüleri bile ekleyebilirsiniz!

Laravel bileşenleri genel olarak iki şekilde genişletilir: IoC konteynerinde yeni implementasyonlar bağlayarak veya bir ekstensiyonu "Factory" tasarım deseninin implementasyonları olan bir `Manager` sınıfı ile register ederek. Bu bölümde, frameworkün genişletilmesi için çeşitli metodları keşfedeceğiz ve gerekli kodları inceleyeceğiz.

Genişletme Metodları

Aklınızda tutun, Laravel bileşenleri tipik olarak iki yoldan biriyle genişletilir: IoC bağlamaları ve `Manager` sınıfları. Manager sınıfları "factory" tasarım deseninin bir implementasyonu olarak hizmet eder ve cache ve session gibi sürücü temelli araçların başlatılmasından sorumludur.

Manager'lar & Factory'ler

Laravel sürücü temelli bileşenlerin oluşturulmasını yöneten birkaç `Manager` sınıfıyla gelir. Bunlar cache, session, authentication ve queue bileşenleridir. Manager sınıfı, uygulamanın yapılandırmasına dayalı olarak belirli bir sürücü implementasyonunun oluşturulmasından sorumludur. Örneğin, `CacheManager` sınıfı APC, Memcached, Native ve diğer cache sürücü implementasyonlarını oluşturabilir.

Bu managerlerin her birisinde, managere kolaylıkla yeni sürücü çözünürlük işlevselliği enjekte edilmesi için kullanılabilen bir `extend` metodu bulunmaktadır. Aşağıda, bu managerlerin her birini, onların her birine özel bir sürücü desteğinin nasıl enjekte edildiğinin örnekleriyle birlikte göreceğiz.

Managerlarınız Hakkında Bilgi Edinin

Laravel'le gelen `CacheManager` ve `SessionManager` gibi çeşitli `Manager` sınıflarını keşfetmek için biraz zaman ayırın. Bu sınıfların baştan sona okunması Laravel'in örtü altında nasıl çalıştığı konusunda

size daha kapsamlı bir anlayış verecektir. Tüm manager sınıfları `Illuminate\Support\Manager` taban sınıfını genişletir, bu taban sınıf her manager için yararlı, ortak bazı işlevsellik sağlar.

Cache

Laravel cache aracını genişletmek için, `CacheManager`deki managera özel bir sürücü çözümleyicisi bağlamak için kullanılan ve tüm manager sınıfları çapında ortak olan `extend` metodunu kullanacağız. Örneğin, "mongo" adında yeni bir cache sürücüsü register etmek için, şöyle yapacağız:

```
Cache::extend('mongo', function($app)
{
        // Illuminate\Cache\Repository olgusu döndür...
});
```

`extend` metoduna geçilen ilk parametre sürücünün adıdır. Bu, sizin `app/config/cache.php` yapılandırma dosyanızdaki `driver` opsiyonuna tekabül edecektir. İkinci parametre bir `Illuminate\Cache\Repository` olgusu döndürmesi gereken bir Closure'dur. Bu Closure'a `Illuminate\Foundation\Application`in bir olgusu ve bir IoC konteyneri olan bir `$app` olgusu geçilecektir.

Özel cache sürücümüzü oluşturmak için, öncelikle `Illuminate\Cache\StoreInterface` sözleşmesini implemente etmemiz gerekiyor. Yani, bizim MongoDB cache implementasyonumuz şöyle bir şey olacaktır:

```
class MongoStore implements Illuminate\Cache\StoreInterface {

        public function get($key) {}
        public function put($key, $value, $minutes) {}
        public function increment($key, $value = 1) {}
        public function decrement($key, $value = 1) {}
        public function forever($key, $value) {}
        public function forget($key) {}
        public function flush() {}

}
```

Sadece bir MongoDB bağlantısı kullanarak bu metodların her birini implemente etmemiz gerekiyor. Implementasyonumuzu tamamladıktan sonra, özel sürücümüzün kaydını bitirebiliriz:

```
use Illuminate\Cache\Repository;

Cache::extend('mongo', function($app)
{
        return new Repository(new MongoStore);
});
```

Yukarıdaki örnekte görebileceğiniz gibi, özel cache sürücüleri oluştururken taban `Illuminate\Cache\Repository` sınıfını kullanabilirsiniz. Tipik olarak kendi repository sınıfınızı oluşturma zorunluğu söz konusu değildir.

Özel cache sürücü kodunuzu nereye koyacağınızı merak ediyorsanız, onu Packagist'te bulundurmayı düşünün! Veya, uygulamanızın birincil klasörü içinde bir `Extensions` aduzayı oluşturabilirsiniz. Örneğin, uygulama `Snappy` adındaysa, cache ekstensiyonunu `app/Snappy/Extensions/MongoStore.php` içine koyabilirsiniz. Bununla birlikte, Laravel'in rijit bir uygulama yapısına sahip olmadığını ve uygulamanızı sizin tercihlerinize göre organize etmekte özgür olduğunuzu aklınızda tutun.

Nereye Genişletilecek

Şayet kod parçalarınızı nereye koyacağınızı merak ediyorsanız, her zaman bir hizmet sağlayıcı düşünün. Daha önce tartıştığımız gibi, framework ekstensiyonlarını organize etmek için bir hizmet sağlayıcı kullanmak kodunuzu organize etmek için harika bir yoldur.

Session

Laravel'i özel bir session sürücüsü ile genişletmek, tıpkı cache sisteminin genişletilmesi kadar kolaydır. Aynı şekilde, özel kodumuzu register etmek için `extend` metodunu kullanacağız:

```
Session::extend('mongo', function($app)
{
        // SessionHandlerInterface'in implementasyonunu döndür
});
```

Dikkat ederseniz bizim özel session sürücümüz `SessionHandlerInterfacei` implemente edecektir. Bu interface PHP 5.4+ çekirdeğine dahil edilmiştir. Eğer siz PHP 5.3 kullanıyorsanız, ileriye yönelik uyumluluğa sahip olmanız için bu interface Laravel tarafından sizin için tanımlanmış olacaktır. Bu interface, implemente etmemiz gereken sadece birkaç basit metod içermektedir. Bir MongoDB implementation kalıbı şöyle bir şeydir:

```
class MongoHandler implements SessionHandlerInterface {

        public function open($savePath, $sessionName) {}
        public function close() {}
        public function read($sessionId) {}
        public function write($sessionId, $data) {}
        public function destroy($sessionId) {}
        public function gc($lifetime) {}

}
```

Bu metodlar cache `StoreInterface` kadar kolay anlaşılabilir olmadıklarından, en iyisi bu metodların her birinin yaptıklarını kısaca keşfedelim:

- `open` metodu tipik olarak dosya tabanlı oturum depolama sistemlerinde kullanılacaktır. Laravel oturumlar için PHP'nin natif dosya depolamasını kullanan `native` bir session sürücüsü ile geldiğinden, bu metoda neredeyse hiçbir şey koymanız gerekmeyecektir. Onu boş bir kalıp olarak bırakabilirsiniz. PHP'nin bizden bu metodu implemente etmemizi istemesi, gerçekte sadece kötü bir interface tasarımıdır (bunu ileride tartışacağız).
- `close` metodu, `open` metoduna benzer şekilde genellikle gözardı edilebilir. Çoğu sürücü için, bu metod gerekli değildir.
- `read` metodu verilen bir `$sessionId` ile eşlik eden oturum verisinin string versiyonunu döndürecektir. Serileştirmeyi Laravel sizin yerinize yapacağı için, sürücünüzde oturum verisini elde ederken veya depolarken herhangi bir serileştirme veya başka kodlamalar yapmanıza gerek yoktur.
- `write` metodu `$sessionId` ile eşlik eden belirli bir `$data` stringini MongoDB, Dynamo vb gibi kalıcı depo sistemlerine yazacaktır.
- `destroy` metodu `$sessionId` ile eşlik eden veriyi kalıcı depodan kaldıracaktır.
- `gc` metodu bir UNIX timestamp türünde verilen bir `$lifetime` süresinden daha eski tüm oturum verisini imha edecektir. Memcached ve Redis gibi süresi kendiliğinden dolan sistemler için bu metod boş bırakılabilir.

`SessionHandlerInterface` implemente edildikten sonra, onu Session manager ile register etmeye hazırız:

```
Session::extend('mongo', function($app)
{
        return new MongoHandler;
});
```

Session sürücüsü kayda geçirildikten sonra `app/config/session.php` yapılandırma dosyamızda `mongo` sürücüsünü kullanabiliriz.

Bilginizi Paylaşın

Unutmayın, özel bir session işleyici yazarsanız, onu Packagist'te paylaşın!

Authentication

Authentication (kimlik doğrulama) da cache ve session araçlarıyla aynı yolla genişletilebilir. Burada da yine aşina olduğumuz `extend` metodunu kullanacağız:

```
Auth::extend('riak', function($app)
{
        // Illuminate\Auth\UserProviderInterface'un implementasyonunu döndür
});
```

Bu `UserProviderInterface` implementasyonlarının tek sorumluluğu MySQL, Riak vb gibi kalıcı bir depolama sisteminin bir `UserInterface` implementasyonunu getirmektir. Bu iki interface Laravel authentication mekanizmalarının kullanıcı verisinin nerede depolandığına veya onu temsil etmek için kullanılan sınıf tipine bakılmaksızın fonksiyon görmeye devam etmesini sağlar.

`UserProviderInterfacee` bir göz atalım:

```
interface UserProviderInterface {

        public function retrieveById($identifier);
        public function retrieveByCredentials(array $credentials);
        public function validateCredentials(UserInterface $user, array $credentials);

}
```

`retrieveById` fonksiyonu tipik olarak kullanıcıyı temsil eden sayısal bir anahtar alır (örneğin bir MySQL veritabanındaki otomatik artan ID gibi). Metod tarafından, bu ID'e uyan `UserInterface` implementasyonu getirilecek ve döndürülecektir.

`retrieveByCredentials` metodu bir uygulamaya giriş yapma girişiminde bulunulduğu zaman `Auth::attempt` metoduna geçilen kimlik bilgilerinden oluşan diziyi alır. Bu metod daha sonra bu kimlik bilgilerine uyan kullanıcıyı altta yatan kalıcı depolama sisteminden "sorgulamalıdır". Tipik olarak, bu metod `$credentails['username']` üzerine bir "where" koşulu olan bir sorgu çalıştıracaktır. **Bu metod herhangi bir şifre doğrulaması veya authentication yapmaya kalkışmamalıdır.**

`validateCredentials` metodu kullanıcı kimliğini doğrulamak için verilen bir `$user` ile `$credentialsi` karşılaştırır. Örneğin, bu metod `$user->getAuthPassword()` stringini `$credentials['password']`in bir `Hash::make` hali ile karşılaştırabilir.

Artık `UserProviderInterface` metodlarının her birini keşfettiğimize göre, bir de `UserInterfacee` göz atalım. Hatırlayınız, providerin `retrieveById` ve `retrieveByCredentials` metodları bu interface'in implementasyonlarını döndürecektir:

```
1  interface UserInterface {
2
3          public function getAuthIdentifier();
4          public function getAuthPassword();
5
6  }
```

Bu interface basittir. `getAuthIdentifier` metodu kullanıcının "birincil anahtarını" döndürmelidir. Bir MySQL back-endinde, bu yine otomatik artan birincil anahtar olacaktır. `getAuthPassword` kullanıcının hash'lenmiş şifresini döndürmelidir. Bu interface, sizin kullandığınız ORM veya depolama soyutlama katmanı ne olursa olsun, authentication sisteminin herhangi bir User sınıfı ile çalışmasına imkan verir. Ön tanımlı olarak Laravel `app/models` klasörü içinde bu interface'i implemente eden bir `User` sınıfı bulundurur, bu yüzden bir implementasyon örneğini görmek için bu sınıfa başvurabilirsiniz.

Son olarak, `UserProviderInterface` implemente edildikten sonra, genişletmemizi `Auth` facade'ı ile kayda geçirmeye hazırız:

```
1  Auth::extend('riak', function($app)
2  {
3          return new RiakUserProvider($app['riak.connection']);
4  });
```

Sürücüyü `extend` metodu ile register ettikten sonra, `app/config/auth.php` yapılandırma dosyanızda yeni sürücüyü belirtin.

IoC Temelli Genişletme

Laravel frameworke dahil edilen hemen her hizmet sağlayıcı IoC konteynerine nesneler bağlar. Uygulamanızın hizmet sağlayıcılarının bir listesini `app/config/app.php` yapılandırma dosyasında bulabilirsiniz. Vaktiniz oldukça bu sağlayıcıların her birinin kaynak koduna baştan sona göz gezdiriniz. Bunu yapmakla, her bir sağlayıcının frameworke neler eklediğini çok daha iyi anlayacaksınız, bunun yanı sıra IoC konteynerine çeşitli hizmetleri bağlamak için hangi anahtarların kullanıldığını da öğreneceksiniz.

Örneğin, `PaginationServiceProvider` IoC konteynerine bir `paginator` anahtarı bağlamaktadır ve bu anahtar bir `Illuminate\Pagination\Environment` olgusuna çözümlenmektedir. Bu IoC bağlamasını override etmek suretiyle kendi uygulamanızda bu sınıfı kolaylıkla genişletebilir ve override edebilirsiniz. Örneğin, taban `Environment` sınıfını genişleten bir sınıf oluşturabilirsiniz:

```
namespace Snappy\Extensions\Pagination;

class Environment extends \Illuminate\Pagination\Environment {

        //

}
```

Sınıf genişletmenizi oluşturduktan sonra, yeni bir `SnappyPaginationProvider` hizmet sağlayıcı sınıfı oluşturarak bunun `boot` metodunda paginator'u override edebilirsiniz:

```
class SnappyPaginationProvider extends PaginationServiceProvider {

        public function boot()
        {
                App::bind('paginator', function()
                {
                        return new Snappy\Extensions\Pagination\Environment;
                });

                parent::boot();
        }

}
```

Bu sınıfın ön tanımlı `ServiceProvider` taban sınıfını değil `PaginationServiceProvider` sınıfını genişlettiğine dikkat ediniz. Service providerinizi genişlettikten sonra, `app/config/app.php` yapılandırma dosyanızdaki `PaginationServiceProvider`ı sizin genişletilmiş providerin ismi ile takas edin.

Konteynerde bağlanan herhangi bir çekirdek sınıfın genişletilmesi için genel yöntem budur. Esasında, her çekirdek sınıf konteynerde bu tarzda bağlanır ve override edilebilir. Tekrar ifade edeyim, frameworkte yer alan hizmet sağlayıcılarının baştan sona okunması çeşitli sınıfların konteynerde nerede bağlandığı ve onu bağlamak için hangi anahtarın kullanıldığı konusunda sizi bilgilendirecektir. Laravelin nasıl biraraya getirildiğini daha çok öğrenmek için harika bir yoldur.

Request Genişletmesi

Request frameworkün çok temel bir parçası olduğu ve istek döngüsünde çok erken başlatıldığı için, `Request` sınıfının genişletilmesi önceki örneklerden biraz farklı yapılır.

İlk olarak, sınıfı normaldeki gibi genişletin:

```
<?php namespace QuickBill\Extensions;

class Request extends \Illuminate\Http\Request {

        // Burada özel, yararlı metodlar olacak...

}
```

Sınıfı genişlettikten sonra, `bootstrap/start.php` dosyasını açın. Bu dosya, uygulamanıza yapılan her istekte en başta dahil edilen dosyalardan biridir. Dikkat ederseniz, bu dosyada yapılan ilk eylem Laravel'in `$app` olgusunun oluşturulmasıdır:

```
$app = new \Illuminate\Foundation\Application;
```

Yeni bir application olgusu oluşturulduğu zaman, yeni bir `Illuminate\Http\Request` olgusu oluşturacak ve `request` anahtarını kullanarak onu IoC konteynerine bağlayacaktır. Bu yüzden, "default" istek tipi olarak kullanılması gereken özel bir sınıfı belirten bir yola ihtiyacımız var, değil mi? Ve, ne mutlu ki, application olgusundaki `requestClass` metodu tam bunu yapar! Yani, `bootstrap/start.php` dosyamızın en üstüne şu satırı ekleyebiliriz:

```
use Illuminate\Foundation\Application;

Application::requestClass('QuickBill\Extensions\Request');
```

Özel istek sınıfı belirtildikten sonra, Laravel bir `Request` olgusu oluşturduğu her zaman bu sınıfı kullanacaktır, böylece sizin özel request sınıfı olgunuzun, unit testlerde bile, her zaman kullanılabilir olmasına imkan verecektir!

Cepheler (Facades)

Giriş

Cepheler uygulamanızın IoC konteynerinde[70] bulunan sınıflar için "statik" bir arayüz sağlar. Laravel birçok cephe ile gelmektedir ve büyük bir ihtimalle daha ne olduklarını bilmeden onları kullanıyorsunuzdur!

Zaman zaman, uygulama ve paketleriniz için kendi cephelerinizi oluşturmak isteyebilirsiniz, bu itibarla bu sınıfların kavramlarını, geliştirilmesini ve kullanımını inceleyelim.

> **Not**: Cepheler konusunu incelemeden önce Laravel IoC konteyneri[71] ile çok aşina olmanız kuvvetle önerilir.

Açıklama

Bir Laravel uygulaması bağlamında bir cephe bir nesneye onun konteynerinden erişim sağlayan bir sınıf demektir. Bu işi yapan mekanizmalar `Facade` sınıfında tanımlıdır. Laravel'in cepheleri ve sizin oluşturduğunuz kendi cepheleriniz bu temel `Facade` sınıfından türeyecektir.

Sizin cephe sınıfınız sadece tek bir metoda tatbikat getirmesi gerekiyor: `getFacadeAccessor`. `getFacadeAccessor` methodunun tanımlayacağı iş konteynerden ne çözeceğidir. `Facade` temel sınıfı sizin cephelerinizden, çözülmüş nesneye yapılan çağrıları ertelemek için `__callStatic()` sihirli-metodunu kullanır.

Pratik Kullanım

Aşağıdaki örnekte, Laravel önbellekleme sistemine bir çağrı yapılmış. Bu koda göz attığınızda, `Cache` sınıfında statik bir metod olan `get`'in çağrılıyor olduğunu düşünebilirsiniz.

```
1  $deger = Cache::get('anahtar');
```

Ancak, eğer `Illuminate\Support\Facades\Cache` sınıfına bakacak olursak, orada `get` adında statik bir metod olmadığını görürüz:

---

[70] /docs/ioc
[71] /docs/ioc

```
class Cache extends Facade {

        /**
         * Get the registered name of the component.
         *
         * @return string
         */
        protected static function getFacadeAccessor() { return 'cache'; }

}
```

Bu Cache sınıfı temel `Facade` sınıfından türetilmiş ve `getFacadeAccessor()` adında bir metod tanımlamış. Bu metodun işinin bir IoC bağlayıcısının adını döndürmek olduğunu hatırlayın.

Bir kullanıcı `Cache` cephesinde herhangi bir statik metoda başvurduğunda, Laravel, IoC konteynerinden `cache` bağlayıcısını çözecek ve istenen metodu (bu örnekte `get`) bu nesneye karşı çalıştıracaktır.

Yani bizim `Cache::get` çağrımız şu şekilde yeniden yazılabilir:

```
$value = $app->make('cache')->get('anahtar');
```

Cephe Oluşturma

Kendi uygulama veya paketiniz için bir cephe oluşturulması kolaydır. Sadece üç şeye ihtiyacınız vardır:

- Bir IoC bağlayıcısı
- Bir cephe sınıfı.
- Bir cephe takma adı yapılandırması.

Bir örnek bakalım. Burada, `OdemeGecidi\Odeme` olarak tanımlanmış bir sınıfımız var.

```
namespace OdemeGecidi;

class Odeme {

        public function process()
        {
                //
        }

}
```

Bu sınıfı IoC konteynerinden çözebiliyor olmamız lazım. Öyleyse, bir bağlayıcı ekleyelim:

```
App::bind('odeme', function()
{
        return new \OdemeGecidi\Odeme;
});
```

Bu bağlayıcıyı kayda geçirmek için harika bir yer `OdemeServiceProvider` adında yeni bir hizmet sağlayıcı[72] oluşturmak ve bu bağlayıcıyı `register` metoduna eklemek olacaktır. Daha sonra Laravel'i sizin hizmet sağlayıcınızı `app/config/app.php` yapılandırma dosyasından yükleyecek şekilde yapılandırın.

Daha sonra, kendi cephe sınıfımızı oluşturabiliriz:

```
use Illuminate\Support\Facades\Facade;

class Odeme extends Facade {

        protected static function getFacadeAccessor() { return 'odeme'; }

}
```

Son olarak, eğer istiyorsak, `app/config/app.php` yapılandırma dosyasındaki `aliases` dizisine kendi cephe'miz için bir takma ad ekleyebiliriz. Artık, `process` metodunu `Odeme` sınıfının bir olgusunda çağırabiliriz.

```
Odeme::process();
```

Cepheleri Taklit Etme

Ünite testi cephelerin nasıl çalıştıkları konusunda önemli bir husustur. Gerçekten, cephelerin varlıkları için bile primer neden test edilebilirliktir. Daha fazla bilgi için, belgelerdeki Cepheleri Taklit Etme[73] kesimine bakın.

[72] /docs/ioc#service-providers
[73] /docs/testing#mocking-facades

Formlar & HTML

Form Açmak

Form Açmak

```
{{ Form::open(array('url' => 'falan/filan')) }}
        //
{{ Form::close() }}
```

Varsayılan olarak, `POST` metodu kullanılır; ancak, istediğiniz bir metodu da belirtebilirsiniz:

```
echo Form::open(array('url' => 'falan/filan', 'method' => 'put'))
```

> **Not**: HTML formları, sadece `POST` ve `GET` metotlarını desteklediği için, `PUT` ve `DELETE` metotları formunuza otomatik olarak bir `_method` gizli alanı eklenmek suretiyle taklit edilecektir.

Ayrıca, isimlendirilmiş rotalar veya denetçi aksiyonlarına yönlendirilen formlar da açabilirsiniz:

```
echo Form::open(array('route' => 'route.name'))

echo Form::open(array('action' => 'Controller@method'))
```

Formunuz dosya yüklemelerini kabul edecekse, diziye `files` seçeneğini ekleyin:

```
echo Form::open(array('url' => 'falan/filan', 'files' => true))
```

CSRF Koruması

Laravel, uygulamanızı siteler arası istek sahtekarlıklarından korumak için kolay bir metot sunar. Öncelikle, kullanıcının oturumuna rastgele bir değer yerleştirilir. Merak etmeyin, bu otomatik olarak yapılır. CSRF değeri, formlarınıza gizli bir alan olarak otomatik olarak yerleştirilir. Yine de, gizli alan için HTML kodunu oluşturmak isterseniz, `token` metodunu kullanabilirsiniz:

Bir Forma CSRF Değeri Eklemek

```
echo Form::token();
```

Bir Rotaya CSRF Filtresi Eklemek

```
Route::post('profil', array('before' => 'csrf', function()
{
        //
}));
```

Forma Model Bağlanması

Sıklıkla, bir modelin içeriğine dayanan bir form oluşturmak isteyebilirsiniz. Bunu yapmak için, `Form::model` metodunu kullanın:

Bir Model Formu Açmak

```
echo Form::model($user, array('route' => array('user.update', $user->id)))
```

Şimdi, bir form elementi oluşturduğunuzda, mesela bir text input, elementin ismiyle eşleşen modelin değeri, otomatik olarak alanın değeri olarak belirlenir. Yani, örneğin, `email` ismine sahip bir text alanı için, kullanıcı modelinin `email` niteliği değer olarak atanır. Bununla birlikte, dahası da var! Oturum flaş verisinde inputa uyan bir öğe mevcutsa, bu değer, model'in değerine nazaran öncelik alacaktır. Yani, öncelik şu şekildedir:

1. Oturum Flaş Verisi (Önceki Girdi)
2. Doğrudan Atanmış Değer
3. Model Nitelik Değeri

Bu size model değerlerine bağlanan formları sadece çabukça oluşturmanıza imkan vermekle kalmaz, sunucu tarafında bir geçerlilik hatası olduğunda tekrar kolayca doldurmanızı da sağlayacaktır!

> **Not:** `Form::model` kullanıyor olduğunuzda, `Form::close` ile formunuzu kapatmayı unutmayın!

Label

Bir Label Elementi Üretilmesi

```
echo Form::label('email', 'E-Mail Adresi');
```

Ek HTML Nitelikleri Belirtme

```
echo Form::label('email', 'E-Mail Adresi', array('class' => 'awesome'));
```

> **Not**: Bir label oluştururken, label ismiyle aynı isimde oluşturduğunuz bir form elemanı otomatik olarak label ile aynı isimde bir ID de alacaktır.

Text, Textarea, Password & Hidden Alanlar

Bir Text Inputu Üretilmesi

```
echo Form::text('uyeadi');
```

Ön Tanımlı Bir Değer Belirtilmesi

```
echo Form::text('email', 'ornek@gmail.com');
```

> **Not**: *hidden* ve *textarea* metodları *text* metodu ile aynı şekilde yazılır.

Bir Password Inputu Üretilmesi

```
echo Form::password('parola');
```

Onay Kutuları ve Seçenek Düğmeleri

Bir Checkbox Veya Radio Inputu Üretilmesi

```
echo Form::checkbox('isim', 'deger');

echo Form::radio('isim', 'deger');
```

Seçilmiş Bir Checkbox Veya Radio Inputu Üretilmesi

```
echo Form::checkbox('isim', 'deger', true);

echo Form::radio('isim', 'deger', true);
```

File Inputu

Bir File Inputu Üretilmesi

```
echo Form::file('resim');
```

Aşağı Açılır Listeler

Aşağı Açılır Bir Liste Üretilmesi

```
echo Form::select('boyut', array('B' => 'Büyük', 'K' => 'Küçük'));
```

Ön Tanımlı Seçilmiş Bir Aşağı Açılır Liste Üretilmesi

```
echo Form::select('size', array('B' => 'Büyük', 'K' => 'Küçük''), 'K');
```

Gruplanmış Bir Liste Üretilmesi

```
echo Form::select('hayvan', array(
        'Kediler' => array('tekir' => 'Tekir'),
        'Köpekler' => array('kangal' => 'Kangal'),
));
```

Düğmeler

Bir Submit Düğmesinin Üretilmesi

```
echo Form::submit('Tıkla beni!');
```

> **Not**: Bir button elamanı üretmeniz gerekiyorsa, *button* metodunu kullanın. Bu aynı *submit* gibi yazılır.

Özel Makrolar

"Makrolar" denen kendi özel Form sınıf yardımcılarınızı tanımlamak kolaydır. Nasıl çalıştığını görün: Önce belli bir isim ve Closure fonksiyonu ile makroyu kayda geçirin:

Bir Form Makrosunun Kayda Geçirilmesi

```
Form::macro('makAlan', function()
{
        return '<input type ="awesome">';
});
```

Şimdi adını kullanarak makronuzu çağırabilirsiniz:

Özel Bir Form Makrosunun Çağırılması

```
echo Form::makAlan();
```

URL Oluşturma

URL oluşturma ile ilgili detaylı bilgi için dokümantasyonun `Yardımcı (Helper) Fonksiyonları` bölümüne bakınız.

Yardımcı (Helper) Fonksiyonları

Arrayler (Diziler)

array_add

`array_add` fonksiyonu, verilen anahtar / değer çiftini, eğer daha önce eklenmemişse array'e eklemeye yarar.

```
$array = array('foo' => 'bar');

$array = array_add($array, 'key', 'value');
```

array_divide

`array_divide` fonksiyonu, birincisi anahtarlar, ikincisi değerler olacak şekilde iki farklı array döndürür.

```
$array = array('foo' => 'bar');

list($keys, $values) = array_divide($array);
```

array_dot

`array_dot` fonksiyonu, çok boyutlu bir array'i derinlikleri 'nokta (dot)' notasyonunu sağlayacak şekilde 1 boyutlu array'e çevirir.

```
$array = array('foo' => array('bar' => 'baz'));

$array = array_dot($array);

// array('foo.bar' => 'baz');
```

array_except

`array_except` fonksiyonu, verilen anahtar / değer çiftini array'den siler.

```
$array = array_except($array, array('keys', 'to', 'remove'));
```

array_fetch

`array_fetch` metodu seçilen bir iç elemanı içeren düz bir dizi döndürür.

```
$array = array(array('name' => 'Taylor'), array('name' => 'Dayle'));

var_dump(array_fetch($array, 'name'));

// array('Taylor', 'Dayle');
```

array_first

`array_first` fonksiyonu, verilen doğruluk testine uyan ilk array elemanını döndürür.

```
$array = array(100, 200, 300);

$value = array_first($array, function($key, $value)
{
        return $value >= 150;
});
```

Ayrıca varsayılan bir değer, üçüncü eleman olarak verilebilir.

```
$value = array_first($array, $callback, $default);
```

array_flatten

`array_flatten` metodu çok boyutlu bir diziyi tek düzey halinde düzleştirir.

```
$array = array('name' => 'Joe', 'languages' => array('PHP', 'Ruby'));

$array = array_flatten($array);

// array('Joe', 'PHP', 'Ruby');
```

array_forget

`array_forget` metodu "dot" notasyonu kullanarak, derin bir iç içe diziden belirli bir anahtar / değer çiftini kaldıracaktır.

```
$array = array('names' => array('joe' => array('programmer')));

$array = array_forget($array, 'names.joe');
```

array_get

`array_get` metodu nokta notasyonu kullanarak derin bir iç içe diziden belirli bir değeri döndürür.

```
$array = array('names' => array('joe' => array('programmer')));

$value = array_get($array, 'names.joe');
```

array_only

`array_only` fonksiyonu, array'den sadece verilen anahtar / değer çiftlerini döndürür.

```
$array = array('name' => 'Joe', 'age' => 27, 'votes' => 1);

$array = array_only($array, array('name', 'votes'));
```

array_pluck

`array_pluck` metodu verilen bir anahtar / değer çiftleri listesini diziden koparacaktır.

```
$array = array(array('name' => 'Taylor'), array('name' => 'Dayle'));

$array = array_pluck($array, 'name');

// array('Taylor', 'Dayle');
```

array_pull

`array_pull` metodu diziden belirli bir anahtar / değer çifti döndürecek, aynı zamanda bu çifti diziden çıkartacaktır.

```
$array = array('name' => 'Taylor', 'age' => 27);

$name = array_pull($array, 'name');
```

array_set

`array_set` metodu nokta notasyonu kullanarak, derin bir iç içe dizide bir değer ayarlar.

```
$array = array('names' => array('programmer' => 'Joe'));

array_set($array, 'names.editor', 'Taylor');
```

head

Dizideki ilk elemanı döndürür. PHP 5.3.x'deki metod zincirleme işine yarar.

```
$first = head($this->returnsArray('foo'));
```

last

Dizideki son elemanı döndürür. Metod zincirlemesinde işe yarar.

```
$last = last($this->returnsArray('foo'));
```

Dosya Yolları

app_path

`app` dizininin tam dosya yolunu getirir.

base_path

Uygulamanın ana dizininin tam dosya yolunu getirir.

public_path

`public` dizininin tam dosya yolunu getirir.

storage_path

`app/storage` dizininin tam dosya yolunu getirir.

Yazı İşlemleri

camel_case

Yazıyı `camelCase` olacak şekilde düzenler.

```php
$camel = camsel_case('foo_bar');

// fooBar
```

class_basename

Girilen class'ın namespace'ler olmadan sadece adını dondürür.

```php
$class = class_basename('Foo\Bar\Baz');

// Baz
```

e

Girilen yazıya UTF-8 desteğiyle `htmlentities` fonksiyonunu uygular.

```php
$entities = e('<html>foo</html>');
```

ends_with

Girilen yazının verilen değerle bitip bitmediğine karar verir.

```php
$value = ends_with('This is my name', 'name');
```

snake_case

Yazıyı `snake_case` olacak şekilde düzenler.

```php
$snake = snake_case('fooBar');

// foo_bar
```

starts_with

Girilen yazının verilen değerle başlayıp başlamadığına karar verir.

```php
$value = starts_with('This is my name', 'This');
```

str_contains

Girilen yazının içinde verilen değerin olup olmadığına karar verir.

```php
$value = str_contains('This is my name', 'my');
```

str_finish

Girilen yazının sonuna verilen değeri ekler. Verilen değerden oluşan ekstraları yok eder.

```php
$string = str_finish('this/string', '/');

// this/string/
```

str_is

Girilen yazıyla verilen değerin eşleşip eşleşmediğine karar verir. Yıldız işareti (*) genel arama karakteri olarak kullanılabilir.

```php
$value = str_is('foo*', 'foobar');
```

str_plural

Girilen kelimeyi çoğul hale getirir (Sadece ingilizce için geçerli).

```php
$plural = str_plural('car');
```

str_random

Girilen değer kadar uzunlukta rastgele karakterlerden oluşan bir yazı üretir.

```php
$string = str_random(40);
```

str_singular

Girilen kelimeyi tekil hale getirir (Sadece ingilizce için geçerli).

```php
$singular = str_singular('cars');
```

studly_case

Girilen yazıyı `StudlyCase` olacak şekilde düzenler.

```
$value = studly_case('foo_bar');

// FooBar
```

trans

Girilen dil satırını çevirir. `Lang::get` fonksiyonunun kısayolu.

```
$value = trans('validation.required');
```

trans_choice

Girilen dil satırını çekimli çevirir. `Lang::choice` fonksiyonunun kısayolu.

```
$value = trans_choice('foo.bar', $count);
```

URL İşlemleri

action

Belirli bir denetçi eylemi için bir URL üretir.

```
$url = action('HomeController@getIndex', $params);
```

route

Verilen isimli rota için URL oluştur.

```
$url = route('routeName', $params);
```

asset

Bir varlık için bir URL üretir.

```
$url = asset('img/photo.jpg');
```

link_to

Girilen URL'e gerekli HTML linkini oluşturur.

```
echo link_to('foo/bar', $title, $attributes = array(), $secure = null);
```

link_to_asset

Verilen varlık için bir HTML bağlantısı üretir.

```
echo link_to_asset('foo/bar.zip', $title, $attributes = array(),
        $secure = null);
```

link_to_route

Girilen rota için gerekli HTML linkini oluşturur.

```
echo link_to_route('route.name', $title,
        $parameters = array(), $attributes = array());
```

link_to_action

Verilen bir denetçi eylemi için bir HTML linki oluşturur.

```
echo link_to_action('HomeController@getIndex', $title,
        $parameters = array(), $attributes = array());
```

secure_asset

Girilen eleman için gerekli HTML linkini HTTPS kullanarak oluşturur.

```
echo secure_asset('foo/bar.zip', $title, $attributes = array());
```

secure_url

Girilen URL'e gerekli HTML linkini HTTPS kullanarak oluşturur.

```
echo secure_url('foo/bar', $parameters = array());
```

url

Verilen bir dosya yolu için tam kalifiye bir URL üretir.

```
echo url('foo/bar', $parameters = array(), $secure = null);
```

Diğer

csrf_token

CSRF token'inin güncel değerini döndürür.

```
$token = csrf_token();
```

dd

Girilen veriyi ekrana basar ve uygulamayı durdurur.

```
dd($value);
```

value

Eğer girilen değer anonim bir fonksiyonsa, değer olarak anonim fonksiyonun döndürdüğü değer döndürür. Eğer değilse direk değeri döndürür.

```
$value = value(function() { return 'bar'; });
```

with

Girilen objeyi döndürür. PHP 5.3.x kullanımında metod zincirleme işlemi için çok yararlı.

```
$value = with(new Foo)->doWork();
```

IoC Konteyneri

Giriş

Laravel'in "inversion of control" konteyneri, sınıf bağımlılıklarının yönetiminde güçlü bir araçtır. Bağımlılık enjeksiyonu ağır kodlanmış sınıf bağımlılıklarının kaldırılması için bir yöntemdir. Bunun yerine, bağımlılıklar çalışma zamanında enjekte edilmekte, bağımlılık işlemleri kolayca takas edilebildiği için daha büyük esneklik sağlamaktadır.

Laravel IoC konteyner'inin anlaşılması hem güçlü, büyük bir uygulama oluşturmak için hem de Laravel'in kendi çekirdeğine katkıda bulunmak için esastır.

Basit Kullanım

IoC konteyneri bağımlılıkları iki yolla çözebilmektedir: ya Closure geri çağrıları yoluyla ya da otomatik çözülüm yoluyla. Önce Closure geri çağrılarını ele alalım. Birincisi, bir "tip", konteynere bağlanabilir:

Bir Tipin Konteynere Bağlanması

```
App::bind('falan', function($app)
{
        return new FalanFilan;
});
```

Bir Tipin Konteynerden Dönüştürülmesi

```
$deger = App::make('falan');
```

`App::make` metodu çağrıldığı zaman, ilgili Closure callback'i çalıştırılacak ve sonuç döndürülecektir.

Bazen, konteyner içine sadece bir kez çözümlenmesi ve aynı olgunun konteynere sonraki çağrılarda döndürülmesi gereken bir şeyler bağlamak isteyebilirsiniz:

Konteynere "Paylaşılan" Bir Tip Bağlama

```
App::singleton('falan', function()
{
        return new FalanFilan;
});
```

`instance` metodunu kullanarak, konteynere mevcut bir nesne olgusunu da bağlayabilirsiniz:

Mevcut Bir Olgunun Konteynere Bağlanması

```
$falan = new Falan;

App::instance('falan', $falan);
```

Otomatik Çözümleme

IoC konteyneri birçok durumda hiçbir yapılandırmaya gerek kalmadan sınıfları çözümleyecek kadar güçlüdür. Örneğin:

Bir Sınıfın Çözümlenmesi

```
class FalanFilan {

        public function __construct(Baz $baz)
        {
                $this->baz = $baz;
        }

}

$falanFilan = App::make('FalanFilan');
```

Dikkat ederseniz, FalanFilan sınıfını konteynerde kayıt etmemiş olsak bile konteyner bu sınıfı hala çözümleyecek, hatta `Baz` bağımlılığını otomatik olarak enjekte edebilecektir!

Bir tipin konteynerde bağlı olmadığı durumlarda, sınıfı görmek ve sınıf yapıcısının tip ipuçlarını okumak için PHP'nin Reflection araçlarını kullanacaktır. Konteyner bu bilgiyi kullanmak suretiyle sınıfın bir olgusunu otomatik olarak inşa edecektir.

Buna karşın, bazı durumlarda, bir sınıf "somut tipte" olmayıp, arayüz tatbikatına (implementasyonuna) bağımlı olabilir. Böyle olduğu takdirde, hangi arayüz tatbikatının enjekte edileceği konusunda konteyneri bilgilendirmek için `App::bind` metodu kullanılmalıdır:

Bir Implementasyona Bir Interface Bağlanması

```
App::bind('UyeRepositoryInterface', 'DbUyeRepository');
```

Şimdi şu denetçiyi ele alalım:

```
class UyeController extends BaseController {

        public function __construct(UyeRepositoryInterface $uyeler)
        {
                $this->uyeler = $uyeler;
        }

}
```

Biz `UyeRepositoryInterface`'i somut bir tipe bağladığımız için, `DbUserRepository` oluşturulduğu zaman otomatik olarak bu denetçiye enjekte edilecektir.

Pratik Kullanım

Laravel uygulamanızın esneklik ve test edilebilirliğini artırmak amacıyla IoC konteyneri kullanmak için çeşitli fırsatlar sağlar. En başta gelen örnek, denetçilerin çözümlenmesidir. Bütün denetçiler IoC kenteyneri tarafından bir kontroller sınıf yapıcısındaki tip ipuçları bağımlılığı ile çözümlenir ve bunlar otomatik olarak enjekte edilecektir.

Tipe Özgü İpucu Denetçi Bağımlılıkları

```
class SiparisController extends BaseController {

        public function __construct(SiparisRepository $siparisler)
        {
                $this->siparisler = $siparisler;
        }

        public function getIndex()
        {
                $all = $this->siparisler->all();

                return View::make('siparisler', compact('all'));
        }

}
```

Bu örnekteki `SiparisRepository` sınıfı otomatik olarak kontroller'e enjekte edilecektir. Bu şu anlama gelir: unit testi[74] sırasında "hayali" bir `SiparisRepository` konteynere bağlanabilir ve denetçiye enjekte edilebilir, böylece sorunsuz bir veritabanı katmanı etkileşimi mümkün olur.

Filtreler[75], kompozitörler[76] ve olay işleyicileri[77] de IoC konteynerinde çözülebilirler . Bunları kayda geçirdiğiniz zaman, sadece kullanılması gereken sınıfın adını vermeniz yeterlidir:

Diğer IoC Kullanım Örnekleri

```
Route::filter('falan', 'FalanFilter');

View::composer('falan', 'FalanComposer');

Event::listen('falan', 'FalanHandler');
```

Hizmet Sağlayıcıları

Hizmet Sağlayıcıları birbirine yakın IoC kayıtlarını tek bir yerleşimde gruplamak için harika bir yoldur. Bunları uygulamanızdaki bileşenleri önceden yüklemenin bir yolu olarak düşünün. Bir hizmet sağlayıcısının içinde özel kimlik doğrulama sürücünüzü kayda geçirebilir, uygulamanızın ambar sınıflarını IoC konteyneri ile kayda geçirebilir, hatta özel bir Artisan komutu dahi kurabilirsiniz.

Aslında, çekirdek Laravel bileşenlerinin pek çoğu hizmet sağlayıcıları içermektedir. Uygulamanızdaki kayıtlı hizmet sağlayıcılarının hepsi, `app/config/app.php` yapılandırma dosyasının `providers` dizisinde listelenmektedir.

Bir hizmet sağlayıcı oluşturmak için, sadece `Illuminate\Support\ServiceProvider` sınıfını genişletin ve bir `register` metodu tanımlayın:

Bir Hizmet Sağlayıcı Tanımlanması

```
use Illuminate\Support\ServiceProvider;

class FalanServiceProvider extends ServiceProvider {

        public function register()
        {
                $this->app->bind('falan', function()
                {
                        return new Falan;
```

[74] /docs/testing
[75] /docs/routing#route-filters
[76] /docs/responses#view-composers
[77] /docs/events#using-classes-as-listeners

```
10                  });
11          }
12
13  }
```

Bu `register` metodunda, uygulama IoC konteynerinin `$this->app` özelliği aracılığıyla kullanılabildiğini unutmayın. Bir sağlayıcı oluşturdunuz ve uygulamanızla kayda geçirmeye hazırsanız, yapmanız gereken şey onu `app` yapılandırma dosyanızdaki `providers` dizisine eklemektir.

Bir hizmet sağlayıcıyı `App::register` metodunu kullanarak çalışma zamanında da kayda geçirebilirsiniz:

Bir Hizmet Sağlayıcının Çalışma Zamanında Kayda Geçirilmesi

```
1  App::register('FalanServiceProvider');
```

Konteyner Olayları

Konteyner ne zaman bir nesne çözümlese bir olay ateşler. `resolving` metodunu kullanarak bu olayı dinleyebilirsiniz:

Bir Resolving Dinleyicisinin Kayda Geçirilmesi

```
1  App::resolving(function($nesne)
2  {
3          //
4  });
```

Çözülen nesnenin geri çağrıya geçirileceğini unutmayın.

Yerelleştirme

Giriş

Laravel'in `Lang` sınıfı farklı dillerdeki yazılara ulaşabileceğiniz bir hizmet verir, bu sayede uygulamanızda rahatlıkla çoklu dil desteği verebilirsiniz.

Dil Dosyaları

Diller için kayıtlar `app/lang` dizininin içerisindeki dosyalarda tutulur. Bu dizin içerisinde desteklenen her dil için bir klasör oluşturulmalıdır.

```
/app
        /lang
                /en
                        mesajlar.php
                /tr
                        mesajlar.php
```

Dil dosyaları basitçe anahtarlı bir şekilde kayıtları barındıran bir dizi döndürür. Örneğin:

Örnek Dil Dosyası

```php
<?php

return array(
        'hosgeldiniz' => 'Uygulamamıza hoş geldiniz!'
);
```

Uygulamanız için varsayılan dil `app/config/app.php` ayar dosyasında tutulmaktadır. Bunun dışında, aktif dili `App::setLocale` metoduyla çalışma esnasında da değiştirebilirsiniz.

Varsayılan Dili Çalışma Esnasında Değiştirmek

```
App::setLocale('tr');
```

Temel Kullanım

Bir Dil Dosyasından Satırları Almak

```
echo Lang::get('mesajlar.hosgeldin');
```

`get` metoduna verilen parametrenin ilk kısmı dil dosyasının adını, ikinci kısım ise alınmak istenen satırın anahtarını içerir.

> **Not**: Eğer istenen dil satırı bulunmuyorsa, `get` metodu anahtarı döndürecektir.

`Lang::get()` ile aynı parametreleri kullanan ve bunun kısaltması olan `trans()` yardımcı metodunu kullanabilirsiniz:

```
echo trans('mesajlar.hosgeldin');
```

Satırlarda Değişiklik Yapmak

Ayrıca dil satırlarınızda yertutucular tanımlayabilirsiniz:

```
'hosgeldin' => 'Hoşgeldin, :isim',
```

Daha sonra, `Lang::get` metoduna ikinci bir parametreyle yapılacak değişiklikleri belirtin:

```
echo Lang::get('mesajlar.hosgeldin', array('isim' => 'Ekrem'));
```

Bir Dil Dosyasının İstenen Satıra Sahip Olup Olmadığını Kontrol Etmek

```
if (Lang::has('mesajlar.hosgeldin'))
{
        //
}
```

Çoğullaştırma

Çoğullaştırma karmaşık bir problemdir, çünkü her dilin farklı ve karmaşık çoğullaştırma kuralları vardır. Dil dosyalarınızda bunu kolaylıkla yönetebilirsiniz. `dik çubuk` karakteri ile, bir çevirinin tekil ve çoğul hallerini birbirinden ayırabilirsiniz:

```
'elmalar' => 'Bir elma var|Bir sürü elma var',
```

Daha sonra `Lang::choise` metoduyla satırı alabilirsiniz:

```
echo Lang::choice('mesajlar.elmalar', 10);
```

Laravel'in tercüme sınıfı gücünü Symfony'nin tercüme bileşeninden aldığı için, daha belirgin çoğullaştırma kuralları da belirleyebilirsiniz:

```
'elmalar' => '{0} Hiç elma yok|[1,19] Bir kaç elma var|[20,Inf] Çok fazla elma va\
r',
```

Validation (Geçerlilik Denetimi)

Validation hatalarının ve mesajlarının yerelleştirmesi için dokümantasyonun `Validation` bölümüne bakınız.

Posta

Yapılandırma

Laravel popüler SwiftMailer[78] kitaplığı üzerinden temiz ve basit bir API sağlamaktadır. Posta yapılandırma dosyası `app/config/mail.php`'dir ve sizin SMTP host, port ve kimlik bilgilerizi değiştirmenize, bunun yanında bu kitaplığın yolladığı tüm mesajlar için global bir `from` adresi ayarlamanıza imkan veren seçenekler içermektedir. İstediğiniz herhangi bir SMTP sunucusunu kullanabilirsiniz. Posta göndermek için şayet PHP'nin `mail` fonksiyonunu kullanmak istiyorsanız, yapılandırma dosyasındaki `driver`'ı `mail`'e değiştiriniz. Bir `sendmail` sürücüsü de bulunmaktadır.

Basit Kullanım

Bir e-posta mesajı göndermek için `Mail::send` metodu kullanılabilir:

```
1  Mail::send('emails.welcome', $data, function($message)
2  {
3          $message->to('falan@numune.com', 'Can Simitci')->subject('Hoş geldiniz!');
4  });
```

Burada `send` metoduna geçilen ilk parametre e-posta gövde metni olarak kullanılacak görünümün ("view"in) ismidir ve ikinci parametre `$data` ise bu görünüme geçilecek veriyi temsil eder. Üçüncü parametremiz e-posta mesajında çeşitli seçenekler belirlemize imkan veren bir bitirme fonksiyonudur.

> **Not**: E-posta görünümlerine mutlaka bir `$message` değişkeni geçilir ve bu değişken bize ataşmanların yazı içine gömülmesi imkanı verir. Dolayısıyla sizin görünüm elemanlarınız arasında bir `message` değişkeni olmaması iyi olur.

Bir HTML görünümüne ek olarak düz metin görünümü kullanmayı da belirtebilirsiniz:

```
1  Mail::send(array('html.view', 'text.view'), $data, $callback);
```

Veya, `html` ya da `text` keylerini kullanmak suretiyle sadece bir tip görünüm belirleyebilirsiniz:

[78] http://swiftmailer.org

```
Mail::send(array('text' => 'view'), $data, $callback);
```

E-posta mesajınızda bunlar yanında, karbon kopyalar veya ataşmanlar gibi başka seçenekler de belirtebilirsiniz:

```
Mail::send('emails.welcome', $data, function($message)
{
        $message->from('bizden@numune.com', 'Laravel');

        $message->to('falan@numune.com')->cc('filan@numune.com');

        $message->attach($eklenecekDosya);
});
```

Bir mesaja dosya eklediğinizde, bir MIME tipi ve / veya ne adla görüneceğini de belirleyebilirsiniz:

```
$message->attach($eklenecekDosya, array('as' => $gorunecekAd, 'mime' => $mime));
```

> **Not**: Bir `Mail::send` bitirme fonksiyonuna geçilen "message" olgusu, SwiftMailer'in message sınıfını genişleterek, e-posta mesajlarınızı oluşturmak için sınıf üzerinden her türlü metodu çağırabilmenize imkan verir.

Ataşmanların Yazı İçine Gömülmesi

Ataşmanların yazı içine gömülmesi tipik olarak zahmetlidir; ama Laravel size e-postalarınıza resimler eklemek ve uygun CID elde etmeniz için pratik bir yol sağlar.

Bir E-Posta Görünümüne Bir Resim Gömülmesi

```
<body>
        İşte bir resim:

        <img src ="<?php echo $message->embed($resimDosyaYolu); ?>">
</body>
```

Bir E-Posta Görünümüne Ham Veri Gömülmesi

```
<body>
        Burada ise ham veriden elde edilen resim görüyoruz:

        <img src ="<?php echo $message->embedData($data, $name); ?>">
</body>
```

`Mail` sınıfı tarafından e-mail görünümlerine bu `$message` değişkeninin MUTLAKA geçileceğini unutmayın.

Postaların Sıraya Sokulması

E-mail mesajlarının gönderilmesi uygulamanızın cevap zamanını önemli ölçüde uzatabileceğin için, birçok geliştirici e-posta mesajlarını arka planda gönderilmek üzere kuyruğa sokmayı tercih eder. Laravel, dahili tekleşmiş kuyruk API[79]'sini kullanarak bunu kolaylaştırır. Bir e-posta mesajını kuyruğa sokmak için tek yapmanız gereken şey, `Mail` sınıfının `queue` metodunu kullanmaktır:

Bir Mail Mesajının Kuyruğa Sokulması

```
Mail::queue('emails.welcome', $data, function($message)
{
        $message->to('falan@numune.com', 'Can Simitci')->subject('Hoş geldiniz!');
});
```

`later` metodunu kullanarak mail mesajınızın gönderilmek için bekleyeceği saniye sayısını da belirleyebilirsiniz:

```
Mail::later(5, 'emails.welcome', $data, function($message)
{
        $message->to('falan@numune.com', 'Can Simitci')->subject('Hoş geldiniz!');
});
```

Mesajı yollamak için belirli bir kuyruk veya "tüpgeçit" belirlemek istiyorsanız bunu `queueOn` ve `laterOn` metodlarını kullanarak gerçekleştirebilirsiniz:

```
Mail::queueOn('queue-name', 'emails.welcome', $data, function($message)
{
        $message->to('falan@numune.com', 'Can Simitci')->subject('Hoş geldiniz!');
});
```

[79] /docs/queues

Posta & Yerel Geliştirme

E-posta gönderen bir uygulama geliştirilirken, genelde lokal veya geliştirme ortamında mesaj göndermenin devre dışı bırakılması arzu edilmektedir. bunu yapmak için, ya `Mail::pretend` metodunu çağırın ya da `app/config/mail.php` yapılandırma dosyanızdaki `pretend` seçeneğini `true` olarak ayarlayın. Mailer `pretend` modunda olduğu zaman, mesajlar alıcıya gönderilmek yerine uygulamanızın günlük dosyalarına yazılacaktır.

Taklit Posta Modunun Etkinleştirilmesi

```
1  Mail::pretend();
```

Paket Geliştirme

Giriş

Paketler Laravel'e işlevsellik eklemenin esas yollarıdır. Paketler tarihlerle çalışmanın harika bir yolu olan Carbon[80] gibi bir şey ya da Behat[81] gibi tam bir BDD test framework'ı olabilir.

Farklı paket türleri bulunmaktadır. Bazı paketler kendi başınadır, yani sadece Laravel değil herhangi bir framework ile çalışırlar: Carbon ve Behat her ikisi de bu tür stand-alone paket örnekleridir. Bu paketler sadece `composer.json` dosyasında istek yapılmak suretiyle Laravel'le kullanılabilmektedir.

Öte yandan, diğer bazı paketler özellikle Laravel ile kullanım için tasarlanmıştır. Önceki Laravel sürümlerinde, bu tip paketlere "bundle" deniyordu. Bu paketlerde özellikle bir Laravel uygulamasını güçlendirmeyi amaçlamış rotalar, denetçiler (controllers), görünümler, yapılandırmalar ve migrasyonlar olabilir. Kendi başına türde bir paket geliştirmek için gerekli özel bir süreç olmadığı için, bu kılavuz esas itibarıyla Laravel'e özgü olanların geliştirilmesini kapsamaktadır.

Tüm Laravel paketleri Packagist[82] ve Composer[83] aracılığıyla dağıtılır, bu yüzden bu harika PHP paket dağıtım araçlarını öğrenmek esastır.

Bir Paket Oluşturma

Laravel'le kullanmak üzere yeni bir paket oluşturmanın en kolay yolu `workbench` Artisan komutudur. Öncelikle, `app/config/workbench.php` dosyasında birkaç seçeneği ayarlamanız gerekiyor. Bu dosyada, bir `name` ve `email` seçeneği bulacaksınız. Bu değerler sizin yeni paketiniz için bir `composer.json` dosyası üretmekte kullanılacaktır. Bu değerleri girdikten sonra, bir tezgah (workbench) paketi oluşturmaya hazırsınız!

Workbench Artisan Komutunun Verilmesi

```
1  php artisan workbench satıcıadı/paketadı --resources
```

Satıcıadı sizin paketinizi farklı yazarlardan gelen aynı isimli diğer paketlerden ayırt etmenin bir yoludur. Örneğin ben (Taylor Otwell) "Zapper" adında yeni bir paket oluşturacaksam, satıcıadı `Taylor`, paketadı ise `Zapper` olacaktır. Ön tanımlı olarak, bu workbench komutu framework

[80] https://github.com/briannesbitt/Carbon
[81] https://github.com/Behat/Behat
[82] http://packagist.org
[83] http://getcomposer.org

bilinemez paketler oluşturur; ancak, `resources` komutu workbench'e `migrations`, `views`, `config` ve bunlar gibi Laravel'e özgü dizinleri olan paketler üretmesini söyler.

`workbench` komutu çalıştırıldıktan sonra sizin paketiniz Laravel kurulumunuzun `workbench` dizini içerisinde hazırlanmış olacaktır. Daha sonra, paketiniz için oluşturulmuş olan `ServiceProvider`'i kayda geçireceksiniz. Bu hizmet sağlayıcının adını `app/config/app.php` dosyasındaki `providers` dizisine ekleyerek kayda geçirebilirsiniz. Bu, Laravel'e uygulamanız başladığı zaman sizin paketinizi yüklemesi talimatı verecektir. Hizmet sağlayıcıları `[Paket]ServiceProvider` şeklinde bir isimlendirme geleneği kullanırlar. Öyleyse, yukarıdaki örnek için `providers` dizisine `Taylor\Zapper\ZapperServiceProvid` ekleyeceğiz.

Sağlayıcıyı kayda geçirdikten sonra artık paketinizi geliştirmeye başlayabilirsiniz! Bununla birlikte, bu konuya geçmeden önce, paket yapısı ve geliştirme iş akışını daha yakından tanımak için aşağıdaki kesimleri gözden geçirmenizde yarar var.

> **Not**: Servis sağlayıcınız bulunamıyorsa uygulamanızın ana dizininde `php artisan dump-autoload` komutunu çalıştırınız.

Paket Yapısı

`workbench` komutu kullanılırken, paketiniz, paketinizin Laravel frameworkün diğer kısımlarıyla iyi bütünleşmesine imkan veren geleneklerle kurulur:

Temel Paket Dizin Yapısı

```
/src
        /Satici
                /Paket
                        PaketServiceProvider.php
        /config
        /lang
        /migrations
        /views
/tests
/public
```

Bu yapıyı biraz daha açalım. Buradaki `src/Satici/Paket` dizini sizin paketinizin `ServiceProvider` de dahil olmak üzere tüm sınıflarının evidir. `config`, `lang`, `migrations` ve `views` dizinleri ise, tahmin edebileceğiniz gibi paketinizdeki kaynakların kendilerine tekabül edenlerini içermektedir. Paketlerde, tıpkı "normal" uygulamalarda olduğu gibi bu kaynaklardan birileri olabilir.

Hizmet Sağlayıcıları

Hizmet sağlayıcıları paketleriniz için tamamen önceden yükleme (bootstrap) sınıflarıdır. Ön tanımlı olarak bunlar iki metod taşırlar: `boot` ve `register`. Bu metodların içerisinde, istediğiniz her şeyi yapabilirsiniz: bir rota dosyası dahil etmek, IoC konteynerinde bağlayıcı kayda geçirmek, olaylara tutturmak veya istediğiniz daha başka bir şey.

Bunlardan `register` metodu, hizmet sağlayıcı kayıt edilir edilmez çağrılır, `boot` komutu ise sadece bir istek yönlendirilmeden önce çağrılır. Bu nedenle, eğer sizin hizmet sağlayıcınızdaki eylemler, zaten kaydı yapılmış başka bir hizmet sağlayıcısına dayanıyorsa veya başka bir sağlayıcı tarafından bağlanan hizmetleri geçersiz bırakıyorsanız, `boot` metodunu kullanmalısınız.

`workbench` kullanarak bir paket oluşturulurken, `boot` komutu zaten bir eylem içerir:

```
1 $this->package('satici/paket');
```

Bu metod Laravel'in uygulamanız için görünüm, konfigürasyon ve diğer kaynakları nasıl düzgünce yükleyeceğini bilmesine imkan verir. Genelde, paket kurulumunu workbench gelenekleri kullanarak yapacağı için bu kod satırını değiştirmenin bir gereği yoktur.

Servis sağlayıcı sınıfları için "varsayılan yer" mevcut değildir. Bunları istediğiniz yere konumlandırabilirsiniz, belki bunları `app` dizini içinde `Providers` aduzayı ile organize edersiniz. Dosya, Composer'ın otomatik-yükleme olanakları[84] sınıfı yükleyebilmek için dosyanın nerede bulunduğunu bildiği sürece istediğiniz yere konumlandırılabilir.

Paket Gelenekleri

Bir paketten gelen kaynaklar kullanılırken, örneğin yapılandırma öğeleri veya görünümler için genelde çift iki nokta üst üste söz dizimi kullanılır:

Bir Paketteki Bir Görünümü Yükleme

```
1 return View::make('package::gorunum.isim');
```

Bir Paket Yapılandırma Öğesinin Öğrenilmesi

```
1 return Config::get('package::grup.secenek');
```

> **Not**: Eğer paketinizde migrasyonlar varsa, sınıf adının başka paketlerle olası sınıf adı çatışmalarını önlemek amacıyla migrasyon isimlerine paketinizin adını ön ek vermeyi düşünün.

[84] http://getcomposer.org/doc/01-basic-usage.md#autoloading

Geliştirme İş Akışı

Bir paket geliştirirken bir uygulama kapsamı içerisinde geliştiriyor olabilmek yararlı olur ve size kolaylıkla görmek ve şablonlarınızla denemek ve benzeri imkanlar verir. Bu nedenle, bu işe başlarken Laravel'in yeni bir kopyasını yükleyin, sonra da paket yapınızı oluşturmak için `workbench` komutunu kullanın.

`workbench` komutunun paketinizi oluşturmasından sonra, `workbench/[satici]/[paket]` dizininden `git init` yapabilir ve paketinizi doğrudan workbench'tan `git push` yapabilirsiniz! Bu size sürekli `composer update` komutlarıyla batağa saplanmaksızın bir uygulama bağlamında uygun bir şekilde paket geliştirmenize imkan verecektir.

Sizin paketleriniz `workbench` dizininde olduğundan, Composer'in sizin paketinizin dosyalarını otomatik yüklemeyi nereden bileceğini merak ediyor olabilirsiniz. Bu `workbench` dizini mevcut olduğu zaman, Laravel akıllı bir şekilde paket var mı diye bu dizini tarayacak, uygulama başladığında bunların Composer autoload dosyalarını yükleyecektir!

Eğer paketinizin autoload dosyalarını tekrar üretmeniz gerekirse, `php artisan dump-autoload` komutunu kullanabilirsiniz. Bu komut, sizin kök projenizdekiler yanında, oluşturmuş olduğunuz workbench'lerdeki autoload dosyalarını da tekrardan üretecektir.

Artisan Autoload Komutunun Çalıştırılması

```
1 php artisan dump-autoload
```

Paket Yönlendirme (Routing)

Laravel'in önceki sürümlerinde, bir paketin hangi URI'lere cevap vereceğini belirtmek için `handles` cümleciği kullanılırdı. Ancak, Laravel 4'te, bir paket her URI'ye cevap verebilir. Paketiniz için bir rota dosyasını yüklemek için, hizmet sağlayıcınızın `boot` metodu içerisinde onu `include` etmeniz yeterlidir.

Bir Hizmet Sağlayıcısından Bir Rota Dosyasının Dahil Edilmesi

```
1 public function boot()
2 {
3         $this->package('satici/paket');
4 
5         include __DIR__.'/../../routes.php';
6 }
```

> **Not**: Şayet paketiniz denetçiler (controllers) kullanıyorsa, bunların sizin `composer.json` dosyanızın auto-load kesiminde düzgün bir şekilde yapılandırılmış olduğundan emin olun.

Paket Yapılandırması

Bazı paketler yapılandırma dosyaları gerektirebilir. Bu dosyalar tipik uygulama yapılandırma dosyalarıyla aynı şekilde tanımlanmalıdır. Ve, hizmet sağlayıcınızda kaynakları kayda geçirmede ön tanımlı `$this->package` metodunu kullanıyorken, olağan "çift iki nokta üst üste" söz dizimini kullanarak erişebilirsiniz:

Paket Yapılandırma Dosyalarına Erişme

```
Config::get('paket::dosya.secenek');
```

Ancak eğer paketiniz tek bir yapılandırma dosyası içeriyorsa, adına sadece `config.php` diyebilirsiniz. Böyle yapmışsanız, dosya adını belirtmenize gerek kalmadan seçeneklere doğrudan erişebilirsiniz:

Tek Dosyalı Paket Yapılandırmasına Erişme

```
Config::get('paket::secenek');
```

Bazen, görünümler gibi paket kaynaklarınızı tipik `$this->package` metodundan başka türlü kayda geçirmek isteyebilirsiniz. Tipik olarak bu sadece kaynaklar konvansiyonel bir yerleşimde olmadıkları takdirde yapılacaktır. Bu kaynakları elle kayda geçirmek için `View`, `Lang` ve `Config` sınıflarının `addNamespace` metodunu kullanabilirsiniz:

Bir Kaynak Aduzayının Elle Kayda Geçirilmesi

```
View::addNamespace('paket', __DIR__.'/path/to/views');
```

Aduzayı kayda geçirildikten sonra, kaynağa erişmek için aduzayının adını ve "çift iki nokta üst üste" söz dizimini kullanabilirsiniz:

```
return View::make('paket::view.isim');
```

`View`, `Lang` ve `Config` sınıflarında `addNamespace` için metod biçimi aynıdır.

Basamaklı Yapılandırma Dosyaları

Diğer geliştiriciler sizin paketlerinizi yükledikleri zaman yapılandırma seçeneklerinden bir kısmını geçersiz kılmak ve değiştirmek isteyebilirler. Ancak, eğer sizin paket kaynak kodunuzdaki değerleri değiştirirlerse, Composer'in daha sonraki paket güncellemesinde bunun üzerine yazılacaktır, tekrar sizin yazdığınız hale gelecektir. O yüzden, bunun yerine `config:publish` artisan komutu kullanılmalıdır:

Config Publish Komutunun Çalıştırılması

```
php artisan config:publish satici/paket
```

Bu komut çalıştırıldığında, sizin uygulamanız için olan konfigürasyon dosyaları `app/config/packages/satici/pake` dizinine kopyalanacak, burada geliştiriciler tarafından güvenle değiştirilebilecektir!

> **Not:** Geliştiriciler ayrıca onları `app/config/packages/satici/paket/environment`'e koyarak sizin paketiniz için ortama özgü yapılandırma dosyaları da oluşturabilirler.

Paket Migrasyonları

Paketleriniz için kolayca migrasyon oluşturabilir ve çalıştırabilirsiniz. workbench'de bir paket için bir migrasyon oluşturmak için `--bench` seçeneğini kullanın:

Workbench Paketleri İçin Migrasyon Oluşturulması

```
php artisan migrate:make create_users_table --bench="satici/paket"
```

Workbench Paketleri İçin Migrasyonların Çalıştırılması

```
php artisan migrate --bench="satici/paket"
```

`vendor` dizinine Composer tarafından yüklenmiş bitmiş bir paket için migrasyonlar çalıştırmak için `--package` yönergesini kullanabilirsiniz:

Yüklenmiş Bir Paket İçin Migrasyonların Çalıştırılması

```
php artisan migrate --package="satici/paket"
```

Paket Varlıkları

Bazı paketlerde JavaScript, CSS ve resimler gibi varlıklar olabilir. Ancak biz `satici` veya `workbench` dizinlerinde varlıklara bağlanamayız, öyleyse bu varlıkları uygulamamızın `public` dizinine taşıyacak bir yola ihtiyacımız var. Sizin için bununla `asset:publish` komutu ilgilenecektir:

Paket Varlıklarının Public Dizinine Taşınması

```
php artisan asset:publish

php artisan asset:publish satici/paket
```

Eğer paket hala `workbench`'de ise, `--bench` yönergesini kullanın:

```
1 php artisan asset:publish --bench="satici/paket"
```

Bu komut varlıkları satıcı ve paket ismine göre `public/packages` dizinine taşıyacaktır. Yani, `userscape/kudos` adındaki bir paket kendi varlıklarını `public/packages/userscape/kudos` dizinine taşıyacaktır. Bu varlık yayımlama geleneğinin kullanılması, kendi paketlerinizin görünümlerinde varlık path'lerini güvenle kodlamanıza imkan verir.

Paketlerin Yayımlanması

Paketiniz yayımlanmaya hazır olduğunda, paketi Packagist[85] ambarına yollayacaksınız. Eğer paketiniz Laravel'e özgü ise, paketinizin `composer.json` dosyasına bir `laravel` etiketi eklemeyi düşünün.

Ayrıca, geliştiriciler kendi `composer.json` dosyalarında sizin paketinize istek yaptıklarında stabil sürümlere bağlı olabilmeleri için sürümlerinizi de etiketlemeniz hoş ve yardımcı olacaktır. Şayet stabil bir sürüm hazır değilse, `branch-alias` Composer direktifini kullanmayı düşünün.

Paketinizi yayımladıktan sonra, `workbench` tarafından oluşturulan uygulama bağlamı içinde onu geliştirmeye devam etmekte özgürsünüz. Bu, paketinizi yayımladıktan sonra bile rahat bir şekilde geliştirmek için muazzam bir yoldur.

Bazı kuruluşlar kendi geliştiricileri için paketlerini kendi özel ambarlarında barındırmayı tercih ediyorlar. Siz de böyle yapmayı düşünürseniz, Composer ekibi tarafından sağlanan Satis[86] belgelerini inceleyin.

[85] http://packagist.org
[86] http://github.com/composer/satis

Sayfalandırma

Yapılandırma

Diğer çatılarda (frameworkler'de), sayfalandırma oldukça sıkıntılı olabilir. Laravel bu işi çocuk oyuncağı gibi yapar. `app/config/view.php` dosyasında bir tek yapılandırma seçeneği bulunmaktadır. `pagination` seçeneği sayfalandırma bağlantıları (links) oluşturmak için kullanılması gereken görünümü (view) belirtir. Varsayılan olarak, Laravel iki görünüm içerir.

`pagination::slider` görünümü mevcut sayfaya dayalı olarak akıllı bir bağlantı aralığı gösterirken, `pagination::simple` görünümü sadece "önceki" ve "sonraki" butonlarını gösterecektir. **Her iki görünüm de Twitter Bootstrap ile uyumludur**

Kullanım

Öğeleri sayfalandırmak için çeşitli yollar vardır. En basiti sorgu oluşturucusunda veya bir Eloquent modelinde `paginate` metodunu kullanmaktır.

Veritabanı Sonuçlarının Sayfalandırılması

```
1  $uyeler = DB::table('uyeler')->paginate(15);
```

Eloquent[87] modellerini de sayfalandırabilirsiniz:

Bir Eloquent Modelinin Sayfalandırılması

```
1  $uyeler = User::where('oylar', '>', 100)->paginate(15);
```

`paginate` metodundan geçen argüman sayfa başı görüntülemek istediğiniz öğelerin sayısıdır. Bir kez sonuçları aldıktan sonra görünümde görüntüleyebilir ve `links` metodunu kullanarak sayfalandırma bağlantıları oluşturabilirsiniz:

[87] /docs/eloquent

```
<div class ="container">
        <?php foreach ($uyeler as $uye): ?>
                <?php echo $uye->isim; ?>
        <?php endforeach; ?>
</div>

<?php echo $uyeler->links(); ?>
```

Sayfalandırma sistemi oluşturmak işte bu kadar! Unutmayın, mevcut sayfa için çatıya bilgi vermedik. Laravel bunu sizin için otomatik olarak belirledi.

Ayrıca aşağıdaki metodlarla ek olarak sayfalandırma bilgisine erişebilirsiniz:

- getCurrentPage
- getLastPage
- getPerPage
- getTotal
- getFrom
- getTo

Bazen bir sayfalandırma olgusunu kendiniz bir öğeler dizisi geçerek oluşturmak isteyebilirsiniz. Bunu yapmak için `Paginator::make` methodunu kullanınız:

Elle Sayfalandırıcı Oluşturmak

```
$sayfalandirici = Paginator::make($ogeler, $toplamOgeler, $sayfaBasi);
```

Sayfalandırma URI'ını Özelleştirmek

Sayfalandırma tarafından kullanılan `setBaseUrl` methodunu da özelleştirebilirsiniz:

```
$uyeler = Uye::paginate();

$uyeler->setBaseUrl('ozel/url');
```

Yukarıdaki örnek böyle bir URL oluşturacaktır: http://ornek.com/ozel/url?page =2

Sayfalandırma Linklerine Ekleme Yapmak

Sayfalandırıcı üzerinde `appends` methodunu kullanarak sayfalandırma linklerinize sorgu katarı (query string) ekleyebilirsiniz:

```php
<?php echo $uyeler->appends(array('sira' => 'oylar'))->links(); ?>
```

Bu kod, sayfalandırma linkine “&sira =oylar” ekleyecek ve şöyle bir URL üretecektir:

```
http://ornek.com/birsey?sayfa =2&sira =oylar
```

Kuyruklar

Yapılandırma

Laravel'in Queue (kuyruk) bileşeni bir takım farklı kuyruk servisleri için tek bir API sağlamaktadır. Kuyruklar e-mail göndermek gibi zaman harcayan görevleri ileri bir zamana kadar ertelemenize imkan verir ve böylece uygulamanıza yapılan web istekleri büyük ölçüde hızlanır.

Kuyruk yapılandırma dosyası `app/config/queue.php` olarak saklanır. Bu dosyada frameworke dahil edilmiş kuyruk sürücülerinin her birisi için bağlantı yapılandırmaları bulacaksınız. Laravel'deki kuyruk sürücüleri arasında Beanstalkd[88], IronMQ[89], Amazon SQS[90] ve senkronize (lokal kullanım için) sürücü yer almaktadır.

Listelenen bu kuyruk sürücüleri için aşağıdaki bağımlılıklar gereklidir:

- Beanstalkd: `pda/pheanstalk`
- Amazon SQS: `aws/aws-sdk-php`
- IronMQ: `iron-io/iron_mq`

Basit Kullanım Şekli

Kuyruğa yeni bir iş itmek için `Queue::push` metodunu kullanın:

Bir İşin Kuyruğa Sokulması

```
1  Queue::push('SendEmail', array('message' => $message));
```

`push` metoduna girilen ilk parametre işi yapmak için kullanılacak sınıfın adıdır. İkinci parametre işleyiciye geçirilecek veri dizisidir. Bir iş işleyicisi şu şekilde tanımlanmalıdır:

Bir İş İşleyicisinin Tanımlanması

[88] http://kr.github.com/beanstalkd
[89] http://iron.io
[90] http://aws.amazon.com/sqs

```
class SendEmail {

        public function fire($is, $veri)
        {
                //
        }

}
```

Gerekli olan tek metodun `fire` olduğuna dikkat edin. Bu metod bir `iş` olgusu ve bir de kuyruğa sokulacak `veri` dizisi parametrelerini alır.

Eğer iş'in `fire`'den başka bir metod kullanmasını istiyorsanız, işi sokarken (yani push metodunda) metodu belirleyebilirsiniz:

Özel Bir İşleyici Metodunun Belirlenmesi

```
Queue::push('SendEmail@send', array('message' => $message));
```

Bir iş işlendikten sonra kuyruktan silinmelidir. Silme işlemi ilgili `iş` olgusunda `delete` metodu kullanılarak yapılabilir:

İşlenmiş Bir İşin Silinmesi

```
public function fire($is, $veri)
{
        // İşi işle...

        $is->delete();
}
```

Bir işi tekrar kuyruğa devretmek isterseniz, bunu `release` metodu aracılığıyla yapabilirsiniz:

Bir İşin Tekrar Kuyruğa Koyulması

```
public function fire($is, $veri)
{
        // İş sürecini yürüt...

        $is->release();
}
```

İş tekrar salınmadan önce kaç saniye bekleneceğini de belirleyebilirsiniz:

```
$is->release(5);
```

İş işlenirken bir istisna oluşursa, otomatik olarak kuyruğa tekrar salınacaktır. `attempts` metodunu kullanarak, işi çalıştırmak için yapılmış olan girişim sayısını da yoklayabilirsiniz:

Çalıştırma Girişimlerinin Sayısını Yoklama

```
if ($is->attempts() > 3)
{
        //
}
```

İş tanımlayıcılarına da erişebilirsiniz:

Bir İşin ID'ine Erişme

```
$is->getJobId();
```

Kuyruğa Closure Fonksiyonu Sokma

Kuyruğa bir Closure de push edebilirsiniz. Bu, kuyruğa sokulması gerekecek hızlı, basit görevler için çok uygundur:

Kuyruğa Bir Closure Sokulması

```
Queue::push(function($is) use ($id)
{
        Account::delete($id);

        $is->delete();
});
```

> **Not**: Kuyruğa bir Closure sokarken __DIR__ ve __FILE__ sabitleri kullanılmamalıdır.

Iron.io push kuyrukları kullanılıyorken, Closure'ların kuyruğa sokulmasında daha fazla önlem almalısınız. Kuyruk mesajlarızı alan son nokta, isteğin gerçekten Iron.io'den mi geldiğini doğrulayacak bir jeton yoklaması yapmalıdır. Örneğin, sizin push kuyruk son noktanız şuna benzer bir şey olmalıdır: `https://uygulamaniz.com/queue/receive?token =SecretToken`. Böylece, kuyruk istek sıralamasından önce uygulamanızdaki gizli jetonun değerini kontrol edebilirsiniz.

Kuyruk Dinleyicileri Çalıştırma

Laravel, kuyruğa itildikçe yeni işler çalıştıran bir Artisan görevi içermektedir. Bu görevi çalıştırmak için `queue:listen` komutunu kullanabilirsiniz:

Kuyruk Dinleyici Başlatılması

```
php artisan queue:listen
```

Ayrıca dinleyicinin kullanacağı kuyruk bağlantısını da belirtebilirsiniz:

```
php artisan queue:listen connection
```

Unutmamanız gereken şey, bu görev başlatıldıktan sonra elle durdurulana kadar çalışmaya devam edeceğidir. Kuyruk dinleyicinin çalışmayı durdurmamasından emin olmak için Supervisor[91] gibi bir süreç monitörü kullanabilirsiniz.

Ayrıca her işin çalışmasına izin verilecek zaman süresini (saniye cinsinden) de ayarlayabilirsiniz:

İş Zaman Aşımı Parametresi Belirleme

```
php artisan queue:listen --timeout=60
```

Kuyruktaki sadece ilk sıradaki işi yürütmek için `queue:work` komutunu kullanabilirsiniz:

Kuyruktaki İlk İşin İşleme Geçirilmesi

```
php artisan queue:work
```

Push Kuyrukları

Push kuyrukları size herhangi bir art alan veya arka plan dinleyici çalıştırmaksızın güçlü Laravel 4 kuyruk araçlarını kullanmanıza imkan verir. Push kuyrukları şu anda sadece Iron.io[92] sürücüsü tarafından desteklenmektedir. Başlamak için önce bir Iron.io hesabı oluşturun ve Iron kimlik bilgilerinizi `app/config/queue.php` yapılandırma dosyasına ekleyin.

Daha sonra, yeni push edilmiş kuyruk işlerini alacak bir URL son noktasını kayda geçirmek için `queue:subscribe` Artisan komutunu kullanabilirsiniz:

Bir Push Kuyruk Aboneliğinin Kayda Geçirilmesi

```
php artisan queue:subscribe queue_name http://falan.com/queue/receive
```

Şimdi, sizin Iron panonuza giriş yaptığınız zaman, yeni push kuyruğunuzu ve abone olunan URL'yi göreceksiniz. Verilen bir kuyruk için istediğiniz kadar çok URL kaydedebilirsiniz. Sonra da, `queue/receive` son noktanız için bir rota oluşturun ve `Queue::marshal` metodundan cevap döndürün:

[91] http://supervisord.org/
[92] http://iron.io

```
Route::post('queue/receive', function()
{
        return Queue::marshal();
});
```

Doğru iş işleyici sınıfının ateşlenmesiyle `marshal` metodu ilgilenecektir. Push kuyruğundaki işleri ateşlemek için, konvansiyonal kuyruklar için kullanılan aynı `Queue::push` metodunu kullanmanız yeterlidir.

Güvenlik

Yapılandırma

Laravel, kimlik doğrulanması işlerini çok basit hale getirmeyi amaçlamaktadır. Aslında, hemen her şey hazır yapılandırılmış durumdadır. Kimlik doğrulaması yapılandırma dosyası `app/config/auth.php` yerleşiminde bulunmaktadır ve kimlik doğrulama araçlarının davranışlarına nasıl ince ayarlar yapılacağı üzerine iyi belgelenmiş çeşitli seçenekler barındırır.

Ön tanımlı olarak, Laravel `app/models` dizininde bir `User` modeli içermektedir ve bu model ön tanımlı Eloquent kimlik doğrulama sürücüsü ile kullanıma hazırdır. Bu modelin şemasını oluştururken şifre alanının en az 60 karakter olmasını temin etmeniz gerektiğini unutmayın.

Şayet sizin uygulamanız Eloquent kullanmıyorsa, Laravel sorgu oluşturucusunu kullanan `database` kimlik doğrulama sürücüsünü kullanabilirsiniz.

Şifrelerin Saklanması

Laravel'deki `Hash` sınıfı güvenli Bcrypt karıştırması (hashing) sağlar:

Bcrypt Kullanılarak Bir Şifrenin Karıştırılması

```
1  $parola = Hash::make('secret');
```

Bir Şifrenin Karıştırılmışa Göre Doğrulanması

```
1  if (Hash::check('secret', $karistirilmisParola))
2  {
3          // Parola doğrulanmıştır...
4  }
```

Bir Şifrenin Yeniden Karıştırılması Gerekip Gerekmediğinin Yoklanması

```
if (Hash::needsRehash($karistirilmis))
{
        $karistirilmis = Hash::make('secret');
}
```

Kullanıcı Kimliklerinin Doğrulanması

Bir kullanıcının uygulamanıza girişi için `Auth::attempt` metodunu kullanabilirsiniz.

```
if (Auth::attempt(array('email' => $email, 'password' => $parola)))
{
        return Redirect::intended('pano');
}
```

Buradaki `email`'in gerekli bir seçenek değil, sadece örnek olsun diye kullanılmış olduğunu bilin. Veritabanınızda bir "kullanıcı adı"na ("username"e) karşılık gelen sütunu kullanmanız gerekiyor. `Redirect::intended` fonksiyonu, kullanıcıları kimlik doğrulama filtresi tarafından yakalanmadan önce erişmeye çalıştıkları URL'ye yönlendirecektir. Kullanıcının önceden girmeye çalıştığı bir url olmayan durumlarda kullanılabilsin diye bu metoda bir dönüş URI parametresi verilebilir.

`attempt` metodu çağrıldığında, `auth.attempt` olayı[93] ateşlenecektir. Şayet kimlik doğrulama girişimi başarılı olur ve kullanıcı giriş yapmış olursa, `auth.login` olayı da ateşlenecektir.

Bir kullanıcının uygulamanıza zaten giriş yapmış olduğunu tayin etmek için `check` metodunu kullanabilirsiniz:

Bir Kullanıcının Doğrulanmış Olup Olmadığının Tayin Edilmesi

```
if (Auth::check())
{
        // Kullanıcı giriş yapmıştır...
}
```

Şayet uygulamanıza "beni hatırla" işlevselliği vermek istiyorsanız, `attempt` metoduna ikinci parametre olarak `true` geçebilirsiniz, böylece bu kullanıcı süresiz olarak "doğrulanmış" tutulacaktır (yada manuel olarak çıkış işlemi yapıncaya kadar):

Bir Kullanıcının Kimliğinin Doğrulanması ve "Hatırlanması"

[93] /docs/events

```
if (Auth::attempt(array('email' => $email, 'password' => $parola), true))
{
        // Bu kullanıcı hatırlanacak...
}
```

Not: `attempt` metodu `true` döndürürse, kullanıcı uygulamanıza girmiş kabul edilir.

Kimlik doğrulama sorgusuna ekstra şartlar da ekleyebilirsiniz:

Bir Kullanıcının Ek Şartlara Göre Doğrulanması

```
if (Auth::attempt(array('email' => $email, 'password' => $parola, 'aktif' => 1)))
{
    // Bu kullanıcı aktiftir, üyeliği askıya alınmış değildir ve mevcuttur.
}
```

Bir kullanıcının kimliği doğrulandıktan sonra, bu kullanıcının model / kaydına ulaşabilirsiniz:

Login Yapmış Kullanıcıya Erişme

```
$email = Auth::user()->email;
```

Bir kullanıcıyı sadece ID'i ile uygulamanıza giriş yaptırtmak için `loginUsingId` metodunu kullanın:

```
Auth::loginUsingId(1);
```

`validate` metodu gerçekte uygulamaya giriş yapılmaksızın bir kullanıcının kimlik bilgilerinin geçerlilik denetiminden geçirilmesine imkan verir:

Login Olmaksızın Kullanıcı Bilgilerinin Geçerlilik Denetimi

```
if (Auth::validate($kimlikbilgileri))
{
        //
}
```

Bir kullanıcıyı uygulamanıza tek bir istek için giriş yapmak için de `once` metodunu kullanabilirsiniz. Bu durumda oturum veya çerezler kullanılmayacaktır.

Bir Kullanıca Tek Bir İstek İçin Giriş Yapma

```
if (Auth::once($kimlikbilgileri))
{
        //
}
```

Bir Kullanıcıya Uygulamadan Çıkış Yapma

```
Auth::logout();
```

Elle Kullanıcı Girişi

Şayet, mevcut bir kullanıcı olgusunu uygulamanıza giriş yaptırmak istiyorsanız, bu olguda `login` metodunu çağırmanız yeterlidir:

```
$uye = Uye::find(1);

Auth::login($uye);
```

Bu metod, bir kullanıcıyı `attempt` metodu kullanarak kimlik bilgileri ile giriş yaptırmaya eşdeğerdir.

Rotaların Korunması

Belli bir rotaya sadece kimliği doğrulanmış kullanıcıların erişebilmesini sağlamak amacıyla rota filtreleri kullanılabilir. Laravel ön tanımlı olarak `auth` filtresi sağlamıştır ve `app/filters.php` içinde tanımlanmıştır.

Bir Rotanın Korunması

```
Route::get('profil', array('before' => 'auth', function()
{
        // Sadece kimliği doğrulanmış üyeler girebilir...
}));
```

CSRF Koruması

Laravel, uygulamanızı siteler arası istek sahtekarlıklarından (cross-site request forgeries [CSRF]) korumak için kolay bir metod sağlamaktadır.

Forma CSRF Jetonunun Eklenmesi

```
<input type="hidden" name="_token" value="<?php echo csrf_token(); ?>">
```

Gönderilmiş CSRF Jetonunun Geçerlilik Yoklaması

```
Route::post('register', array('before' => 'csrf', function()
{
    return 'Geçerli bir CSRF jetonu verdiniz!';
}));
```

HTTP Basit Kimlik Doğrulaması

HTTP Basit Kimlik Doğrulaması, kullanıcıları özel bir "giriş" sayfası açmadan uygulamanıza giriş yapabilmeleri için hızlı bir yoldur. Bunun için, rotanıza `auth.basic` filtresi tutturun:

HTTP Basit İle Bir Rotanın Korunması

```
Route::get('profil', array('before' => 'auth.basic', function()
{
        // Sadece kimliği doğrulanmış üyeler girebilir...
}));
```

Ön tanımlı olarak, bu `basic` filtresi kimlik doğrulaması yaparken kullanıcı kaydındaki `email` sütununu kullanacaktır. Siz başka bir sütunu kullanmak istiyorsanız, `basic` metoduna birinci parametre olarak bu sütunun adını geçirin:

```
return Auth::basic('uyeismi');
```

HTTP Basit Kimlik Doğrulamasını oturumda kullanıcı tanıtıcı bir çerez ayarlamadan da kullanabilirsiniz, bu daha çok API kimlik doğrulamalarında işe yarayacaktır. Bunu yapmak için, `onceBasic` metodu döndüren bir filtre tanımlayın:

Durum Bilgisi Olmaksızın Bir HTTP Basit Filtresi Ayarlanması

```
Route::filter('basic.once', function()
{
        return Auth::onceBasic();
});
```

Şifre Hatırlatıcıları & Sıfırlama

Şifre Hatırlatıcı Göndermek

Çoğu web uygulaması, kullanıcılarına unutulmuş şifrelerini sıfırlayacak bir yol verir. Her uygulamada bunu tekrar tekrar yapmaya zorlamak yerine Laravel size şifre hatırlatıcı mektup gönderme ve şifre sıfırlaması yapılması için pratik metodlar sağlar. Başlamak için sizin `User` modelinizin `Illuminate\Auth\Reminders\RemindableInterface` sözleşmesini yerine getirdiğini doğrulayın. Tabii ki, Laravel'le gelen `User` modeli bu arayüz kontratını zaten yerine getirmektedir.

RemindableInterface Yürütme İşlemi

```
class User extends Eloquent implements RemindableInterface {

        public function getReminderEmail()
        {
                return $this->email;
        }

}
```

Daha sonra, şifre sıfırlama jetonlarının saklanacağı bir tablo oluşturulmalıdır. Bu tablo için bir migrasyon üretmek için yapacağınız tek şey `auth:reminders` Artisan komutunu çalıştırmaktır:

Hatırlatıcı Tablo Migrasyonunun Üretilmesi

```
php artisan auth:reminders

php artisan migrate
```

Bir şifre hatırlatıcı göndermek için, `Password::remind` metodunu kullanabiliriz:

Bir Şifre Hatırlatıcı Gönderme

```
Route::post('password/remind', function()
{
        $kimlikbilgileri = array('email' => Input::get('email'));

        return Password::remind($kimlikbilgileri);
});
```

`Password::remind` metoduna geçirilen parametrelerin `Auth::attempt` metoduna geçirilenle aynı olduğuna dikkat edin. Bu metod `User`'ı getirecek ve e-mail aracılığı ile ona bir şifre sıfırlama linki gönderecektir. Bu e-mail görünümüne, şifre sıfırlama formuna link oluşturmakta kullanılabilcek bir `token` değişkeni geçilecektir. Bu görünüme `user` nesnesi de geçilecektir.

> **Not:** `auth.reminder.email` yapılandırma seçeneğini değiştirmek suretiyle e-mail mesajı olarak hangi görünümün kullanılacağını belirleyebilirsiniz. Tabii ki, ön tanımlı bir görünüm mevcuttur.

`remind` metoduna ikinci bir parametre olarak bir bitirme fonksiyonu (Closure) geçerek, kullanıcıya gönderilecek mesaj olgusunu değiştirebilirsiniz:

```
return Password::remind($kimlikbilgileri, function($mesaj, $uye)
{
        $mesaj->subject('Şifre Hatırlatıcınız');
});
```

Ayrıca, `remind` metodunun sonuçlarını doğrudan bir rotadan döndürdüğümüze dikkat ediniz. Ön tanımlı olarak, `remind` metodu mevcut URI'ye bir `Redirect` döndürecektir. Şifre sıfırlamaya çalışılırken eğer bir hata oluşursa, oturuma bir `error` değişkeni, bir de `reminders` dil dosyasından bir dil satırı çekmekte kullanılabilecek bir `reason` değişkeni flaş tarzında gönderilir. Şifre sıfırlama başarılı olursa bu sefer oturuma bir `success` değişkeni gönderilecektir. Bu durumda şifre sıfırlama form görünümünüz şöyle bir şey olacaktır:

```
@if (Session::has('error'))
        {{ trans(Session::get('reason')) }}
@elseif (Session::has('success'))
        Şifre sıfırlaması olan bir e-mail gönderildi.
@endif

<input type="text" name="email">
<input type="submit" value="Hatırlatıcı Gönder">
```

Şifrelerin Sıfırlanması

Bir kullanıcı hatırlatma e-mailindeki sıfırlama linkini tıkladıktan sonra, bir `password` ve `password_confirmation` alanı yanında gizli bir `token` alanı da olan bir forma yönlendirilmelidir. Aşağıda şifre sıfırlama formu için bir rota örneği görülüyor:

```
Route::get('password/reset/{token}', function($token)
{
        return View::make('auth.reset')->with('token', $token);
});
```

Ve, bir şifre sıfırlama formu görünümü de şuna benzeyebilir:

```
@if (Session::has('error'))
        {{ trans(Session::get('reason')) }}
@endif

<input type="hidden" name="token" value="{{ $token }}">
<input type="text" name="email">
<input type="password" name="password">
<input type="password" name="password_confirmation">
```

Tekrar hatırlatmakta yarar var, şifre sıfırlaması sırasında Laravel tarafından saptanabilen herhangi bir hatayı göstermek için `Session`'u kullanıyoruz. Artık sıfırlama işini yapacak bir `POST` rotası tanımlayabiliriz:

```
Route::post('password/reset/{token}', function()
{
        $kimlikbilgileri = array('email' => Input::get('email'));

        return Password::reset($kimlikbilgileri, function($uye, $password)
        {
                $uye->password = Hash::make($password);

                $uye->save();

                return Redirect::to('home');
        });
});
```

Şifre sıfırlama başarılı olursa `User` (üye) olgunuz ve şifre sizin bitirme fonksiyonunuza geçilecek, böylece burada gerçek save oparasyonu yapabileceksiniz. Daha sonra, `reset` metodu tarafından döndürülecek olan bitirme fonksiyonundan ya bir `Redirect` döndürebilirsiniz veya başka bir tipte cevap döndürebilirsiniz. Bu `reset` metodunun istekte geçerli bir `token`, geçerli kimlik bilgileri ve birbirine uyan şifreler olup olmadığını otomatik olarak kontrol ettiğini unutmayın.

Ayrıca, `remind` metoduna benzer şeklilde, şifre resetlemesi sırasında bir hata oluşması durumunda `reset` metodu da bir `error` ve bir `reason` eşliğinde mevcut URI'ye bir `Redirect` döndürecektir.

Kriptolama

Laravel, mcrypt PHP uzantısı aracılığıyla güçlü AES-256 kriptolama imkanı sağlamaktadır:

Bir Değerin Kriptolanması

```
$kriptolu = Crypt::encrypt('secret');
```

> **Not:** `app/config/app.php` dosyasının `key` seçeneğinde 32 karakterli rasgele string ayarladığınızdan emin olun. Aksi Takdirde kriptolanmış değerler güvenli olmayacaktır.

Kriptolu Bir Değerin Çözülmesi

```
$cozuk = Crypt::decrypt($kriptoluDeger);
```

Ayrıca, kriptocu tarafından kullanılan cipher ve mod da ayarlayabilirsiniz

Cipher ve Mod Ayarlanması

```
Crypt::setMode('crt');

Crypt::setCipher($cipher);
```

Oturum

Yapılandırma

HTTP odaklı uygulamalar durum bilgisi taşımadıkları için, oturumlar istekler arasında kullanıcı hakkında bilgi saklamak için bir yol sağlar. Laravel temiz, tek bir API aracılığıyla kullanılabilen çeşitli oturum back-endleri ile birlikte gelir. İçerisinde Memcached[94], Redis[95] ve veritabanları gibi popüler back-end desteği yer almaktadır.

Oturum yapılandırma ayarları `app/config/session.php` dosyasında bulunmaktadır. Bu belgede size sunulan iyi belgelenmiş seçenekleri gözden geçirmeyi unutmayın. Ön tanımlı olarak, Laravel `native` oturum sürücüsünü kullanmak üzere yapılandırılmıştır ve bu yapılandırma uygulamaların çoğunda iyi çalışacaktır.

Oturum Kullanımı

Oturumda Bir Öğe Saklamak

```
1  Session::put('anahtar', 'deger');
```

Dizi Oturum Değerine Bir Değer Eklemek

```
1  Session::push('uyeler.takimlar', 'gelistiriciler');
```

Oturumdaki Bir Öğeyi Öğrenmek

```
1  $deger = Session::get('anahtar');
```

Bir Öğe Almak Veya Varsayılan Bir Değer Döndürmek

[94] http://memcached.org
[95] http://redis.io

```
$deger = Session::get('anahtar', 'default');

$deger = Session::get('anahtar', function() { return 'default'; });
```

Oturumdaki Tüm Verileri Almak

```
$veri = Session::all();
```

Oturumda Bir Öğenin Olup Olmadığını Tespit Etmek

```
if (Session::has('uyeler'))
{
        //
}
```

Oturumdan Bir Öğeyi Çıkartmak

```
Session::forget('anahtar');
```

Oturumdaki Tüm Öğeleri Çıkartmak

```
Session::flush();
```

Tekrar Oturum ID Üretmek

```
Session::regenerate();
```

Flaş Verisi

Bazen oturumda sadece sonraki istek için öğeler saklamak isteyebilirsiniz. Bunu `Session::flash` metodunu kullanarak gerçekleştirebilirsiniz:

```
Session::flash('anahtar', 'deger');
```

Mevcut Flaş Verinin Bir Başka İstek İçin Yeniden Flaşlanması

```
Session::reflash();
```

Flaş Verinin Sadece Bir Alt Kümesinin Yeniden Flaşlanması

```
Session::keep(array('uyeadi', 'email'));
```

Veritabanı Oturumları

`database` oturum sürücüsü kullanıyorken, oturum öğelerini taşıyan bir tablo kurulumu gerekecek. Aşağıda, bu tablo için örnek bir `Şema` deklarasyonu gösterilmektedir:

```
Schema::create('sessions', function($table)
{
        $table->string('id')->unique();
        $table->text('payload');
        $table->integer('last_activity');
});
```

Tabii ki, bu migrasyonu üretmek için `session:table` Artisan komutunu kullanabilirsiniz!

```
php artisan session:table

composer dump-autoload

php artisan migrate
```

Oturum Sürücüleri

Oturum "driver'ı" her istek için oturum verisinin nerede saklanacağını tanımlamaktadır. Laravel çeşitli harika sürücülerle birlikte gelmektedir.

- `native` - oturumlar dahili PHP oturum araçları tarafından halledilecektir.
- `cookie` - oturumlar güvenli, kitiptolanmış çerezlerde saklanacaktır.
- `database` - oturumlar kendi uygulamanızın kullandığı bir veritabanında saklanacaktır.
- `memcached` / `redis` - oturumlar bu hızlı, önbellekleme tabanlı depolardan birisinde saklanacaktır.
- `array` - oturumlar basit bir PHP dizisinde saklanacak ve istekler arasında sebat etmeyecektir.

> **Not**: Array sürücüsü tipik olarak unit testleri için kullanılır, bu yüzden oturum verileri sürdürülmeyecektir.

Şablonlar

Denetçi Düzenleri

Laravel'de şablon kullanma yöntemlerinden birisi denetçi düzenleri üzerinden gerçekleştirilir. İlgili denetçideki `layout` özelliğinin belirlenmesiyle, belirlemiş olduğunuz görünüm oluşturulacak ve eylemlerden dönmüş cevap olarak kabul edilecektir.

Bir Denetçide Bir Düzen Tanımlanması

```
1  class UyeController extends BaseController {
2
3          /**
4           * Cevaplar için kullanılacak olan düzen.
5           */
6          protected $layout = 'layouts.master';
7
8          /**
9           * Uye profilini göster.
10          */
11         public function showProfil()
12         {
13                 $this->layout->content = View::make('uye.profil');
14         }
15
16 }
```

Blade Şablonları

Blade Laravel'le gelen basit ama güçlü bir şablon motorudur. Denetçi düzenlerinden farklı olarak, Blade *şablon kalıtımı* ve *kesimler* (sections) ile yürütülür. Tüm Blade şablonlarının uzantısı `.blade.php` olmalıdır.

Bir Blade Düzeninin Tanımlanması

```
<!-- app/views/layouts/master.blade.php 'de bulunmaktadır-->

<html>
        <body>
                @section('sidebar')
                        This is the master sidebar.
                @show

                <div class ="container">
                        @yield('content')
                </div>
        </body>
</html>
```

Bir Blade Düzeninin Kullanılması

```
@extends('layouts.master')

@section('sidebar')
        @parent

        <p>Burası master sidebar'a eklenmiştir.</p>
@stop

@section('content')
        <p>Burası kendi content bölümümdür.</p>
@stop
```

Bir Blade düzenini genişleten (`extend`) görünümlerin, düzenden gelen kesimleri değiştirmekten başka bir şey yapmadığını unutmayın. İlgili düzenin içeriği bir kesimde `@parent` direktifi kullanılarak çocuk görünüme katılabilir, böylece bir kenar çubuğu veya altbilgi gibi bir düzen kesimine eklemeler yapabilirsiniz.

Kimi zaman, örneğin bir kesimin tanımlanmış olup olmadığından emin olmadığınız durumlarda, `@yield` direktifine ön tanımlı bir değer geçmek isteyebilirsiniz. Bu ön tanımlı değer ikinci parametre olarak geçilebilir.

```
@yield('section', 'Ön Tanımlı İçerik');
```

Diğer Blade Kontrol Yapıları

Veri Yazdırılması

```
Merhaba {{ $isim }}.

Şu anki UNIX zaman damgası {{ time() }}'dır.
```

Tabii ki, kullanıcılardan gelen tüm veriler escape edilmeli ya da arındırılmalıdır. Çıktıyı escape etmek için, üçlü küme parantezi sözdizimini kullanabilirsiniz:

```
Merhaba {{{ $isim }}}.
```

> **Not**: Uygulamanızın kullanıcılarından gelen verileri yazdıracağınız zaman çok dikkatli olun. İçerikte olabilecek HTML antitelerini escape etmek amacıyla her zaman için üçlü küme parantezi sözdizimi kullanın.

If Cümleleri

```
@if (count($records) === 1)
        Tek kayıt var!
@elseif (count($records) > 1)
        Birden çok kayıt var!
@else
        Hiç kayıt yok!
@endif

@unless (Auth::check())
        Giriş yapmadınız.
@endunless
```

Döngüler

```
@for ($i = 0; $i < 10; $i++)
        Şu anki değer {{ $i }}'dir.
@endfor

@foreach ($uyeler as $uye)
        <p>Bu, üye {{ $uye->id }}'dir.</p>
@endforeach

@while (true)
        <p>Sonsuz döngüdeyim.</p>
@endwhile
```

Alt Görünümlerin Dahil Edilmesi

```
@include('view.ismi')
```

Dahil edilen görünüme bir veri dizisi de geçebilirsiniz:

```
@include('view.ismi', array('birsey'=>'veri'))
```

Kesimlerin Üzerine Yazmak

Ön tanımlı olarak, section'lar sectionda önceden mevcut olan bir içeriğe eklenirler. Bir section'u, öncekini geçersiz kılarak tümden üzerine yazmak için `overwrite` cümlesini kullanabilirsiniz:

```
@extends('list.item.container')

@section('list.item.content')
    <p>Bu {{ $item->type }} tipinde bir öğedir</p>
@overwrite
```

Dil Satırlarının Gösterilmesi

```
@lang('language.line')

@choice('language.line', 1);
```

Yorumlar

```
{{-- Bu yorum, gösterilen HTML içerisinde olmayacaktır --}}
```

Unit Testing

Giriş

Laravel hazırlanırken birim testler ile hazırlandı. Açıkçası, PHPUnit ile test desteği halihazırda var ve uygulamanız için hazırlanmış `phpunit.xml` dosyası da Laravel ile birlikte geliyor. PHPUnit'in haricinde, Laravel ayrıca Symfony HttpKernel, DomCrawler ve BrowserKit bileşenlerinden de yararlanıyor ki, kullanıcılar testler sırasında görünümleri inceleyebilsin ve müdahale edebilsin. Bu bileşenler ile temsili bir tarayıcıya sahip oluyorsunuz.

Örnek bir test dosyası `app/tests` dizininde bulunmaktadır. Yeni bir Laravel uygulaması kurulumundan sonra, komut satırında `phpunit` komutuyla testlerinizi çalıştırabilirsiniz.

Testleri Tanımlamak ve Çalıştırmak

Yeni bir test durumu oluşturmak için, `app/test` dizini içerisinde yeni bir test dosyası oluşturmanız yeterli. Test sınıflarınız `TestCase` sınıfını extend ediyor olmalıdır. Bu şekilde normalde PHPUnit ile hazırladığınız test metodlarını aynı şekilde oluşturabilirsiniz.

Örnek Bir Test Sınıfı

```
class FooTest extends TestCase {

        public function testSomethingIsTrue()
        {
                $this->assertTrue(true);
        }

}
```

Daha sonra komut satırında `phpunit` ile uygulamanızın tüm testlerini çalıştırabilirsiniz.

> **Not**: Eğer kendi `setUp` methodunuzu tanımlarsanız, `parent::setUp` kodunu çalıştırdığınızdan emin olun.

Test Ortamı

Testleri çalıştırıken, Laravel otomatik olarak ortam yapılandırmasını `testing`'e alacaktır. Ayrıca, Laravel'de test ortamında `önbellekleme` ve `oturum` için özel ayar dosyaları bulunmaktadır. İki sürücü de bir `dizi` olacak şekilde ayarlanmış olup, test yaparkenki oturum ve önbellek verilerinin kalıcı olmaması sağlanmıştır. Test ortamı için gerektiğinde başka ayarlar yapmakta özgürsünüz.

Testlerin İçerisinde Rotaları Çağırmak

Testleriniz içerisinde `call` metodu ile rahatlıkla rotaları çağırabilirsiniz:

Test Dosyasından Bir Rota Çağırmak

```
$response = $this->call('GET', 'user/profile');

$response = $this->call($method, $uri, $parameters, $files, $server, $content);
```

Daha sonra `Illuminate\Http\Response` nesnesini inceleyebilirsiniz:

```
$this->assertEquals('Hello World', $response->getContent());
```

Ayrıca bir test dosyasından denetçileri de çağırabilirsiniz:

Test Dosyasından Bir Denetçi Çağırmak

```
$response = $this->action('GET', 'HomeController@index');

$response = $this->action('GET', 'UserController@profile', array('user' => 1));
```

`getContent` metodu cevap olarak değerlendirilmiş string içeriğini döndürecektir. Eğer rotanız bir `Görünüm` döndürüyorsa, `original` özelliği ile buna ulaşabilirsiniz:

```
$view = $response->original;

$this->assertEquals('John', $view['name']);
```

Bir HTTPS rotayı çağırmak için, `callSecure` metodunu kullanabilirsiniz.

```
$response = $this->callSecure('GET', 'foo/bar');
```

> **Not**: Testing ortamında olduğunuzda rota filtreleri devre dışı bırakılır. Bunları etkinleştirmek için testinize `Route::enableFilters()` ekleyin.

DOM Böceği

Ayrıca bir rota çağırıp, bir DOM Böceği nesnesi alarak içeriği inceleyebilirsiniz:

```
$crawler = $this->client->request('GET', '/');

$this->assertTrue($this->client->getResponse()->isOk());

$this->assertCount(1, $crawler->filter('h1:contains("Hello World!")'));
```

Böceği nasıl kullanacağınız hakkında detaylı bilgi için kendi dökümantasyonunu[96] okuyabilirsiniz.

Facade'ları Taklit Etmek

Test yaparken, sabit Laravel facadelarını taklit etmeniz gerekecektir. Örneğin, şu denetçi aksiyonunu varsayalım:

```
public function getIndex()
{
        Event::fire('foo', array('name' => 'Dayle'));

        return 'All done!';
}
```

`Event` sınıfına yapılan çağrıyı taklit edebilmek için Facade üzerinde `shouldReceive` metodunu kullanabilirsiniz, bu metod bir Mockery[97] örneği döndürecek.

Bir Facade'ı Taklit Etmek

```
public function testGetIndex()
{
        Event::shouldReceive('fire')->once()->with(array('name' => 'Dayle'));

        $this->call('GET', '/');
}
```

> **Note:** `Request` metodunu taklit etmemelisiniz. Bunun yerine, testlerinizi çalıştırırken istediğiniz girdileri `call` metodunda belirtin.

Laravel'e Özel `Assert` Metodları

Laravel test yapımını kolaylaştırmak için halihazırda bazı `assert` metodlarıyla gelir:

Yanıtın Başarıyla Geldiği İspatlamak

[96] http://symfony.com/doc/master/components/dom_crawler.html
[97] https://github.com/padraic/mockery

```
public function testMethod()
{
        $this->call('GET', '/');

        $this->assertResponseOk();
}
```

Yanıt Kodlarını İspatlamak

```
$this->assertResponseStatus(403);
```

Yanıtın Bir Yönlendirme Olduğunu İspatlamak

```
$this->assertRedirectedTo('foo');

$this->assertRedirectedToRoute('route.name');

$this->assertRedirectedToAction('Controller@method');
```

Bir Görünümde Veri Olduğunu İspatlamak

```
public function testMethod()
{
        $this->call('GET', '/');

        $this->assertViewHas('name');
        $this->assertViewHas('age', $value);
}
```

Oturumda Bir Verinin Kayıtlı Olduğunu İspatlamak

```
public function testMethod()
{
        $this->call('GET', '/');

        $this->assertSessionHas('name');
        $this->assertSessionHas('age', $value);
}
```

Yardımcı Metodlar

Test yapımını kolaylaştırmak için `TestCase` sınıfı bazı yardımcı metodlarla birlikte gelir. Mevcut oturum açmış kullanıcıyı `be` metodu ile belirleyebilirsiniz.

Oturum Açmış Kullanıcıyı Belirleme

```
$user = new User(array('name' => 'John'));

$this->be($user);
```

Bir test içerisinden `seed` metoduyla veritabanınızı yeniden filizlendirebilirsiniz:

Test İçerisinden Veritabanını Yeniden Filizlendirmek

```
$this->seed();

$this->seed($connection);
```

Filizlendirmeyle ilgili daha fazla bilgiyi dökümantasyonun yerleşimler ve filizlendirme[98] bölümünde bulabilirsiniz.

[98] /docs/migrations#database-seeding

Geçerlilik Denetimi

Basit Kullanım

Laravel, `Validation` sınıfı aracığıyla verilerin geçerlilik denetimi ve geçerlilik hata mesajlarının gösterilmesi için basit ve kullanışlı bir araçla birlikte gelmektedir.

Basit Bir Geçerlilik Denetimi Örneği

```
$geçerlilikyoklayici = Validator::make(
        array('isim' => 'Tuana Şeyma'),
        array('yas'  => 'required|min:5')
);
```

Buradaki `make` metoduna geçilen ilk parametre, geçerli olup olmadığına bakılacak veridir. İkinci parametre ise, bu veriye tatbik edilecek geçerlilik kurallarıdır.

Birden çok kural ya bir "pipe" karakteri (|) ile ayrılır, ya da ayrı dizi elemanları olarak verilebilir.

Kuralları Belirtmek İçin Dizi Kullanımı

```
$geçerlilikyoklayici = Validator::make(
        array('isim' => 'Tuana Şeyma'),
        array('yas'  => array('required', 'min:5'))
);
```

Birçok Alana Tek Seferde Geçerlilik Denetimi Yapmak

```
$validator = Validator::make(
    array(
        'name' => 'Tuana Şeyma',
        'password' => 'aksakşifre',
        'email' => 'tuanaseyma@eldem.com'
    ),
    array(
        'name' => 'required',
        'password' => 'required|min:8',
        'email' => 'required|email|unique'
    )
);
```

Bir `Validator` olgusu oluşturulduktan sonra, geçerlilik denetimi yapmak için `fails` veya `passes` metodları kullanılabilir.

```
if ($geçerlilikyoklayici->fails())
{
        // İlgili veri geçerlik denetimini geçememiştir
}
```

Şayet geçerlilik denetimi başarısız olursa, geçerlik yoklayıcısından hata mesajları alabilirsiniz:

```
$mesajlar = $geçerlilikyoklayici->messages();
```

Ayrıca, başarısız olan geçerlilik kurallarına bir dizi olarak da erişebilirsiniz. Bunu yapmak için `failed` metodunu kullanabilirsiniz:

```
$kalanlar = $geçerlilikyoklayici->failed();
```

Dosyalar İçin Geçerlilik Denetimi

`Validator` sınıfı dosyaların geçerliliği konusunda `size`, `mimes` ve benzeri kurallar sağlar. Dosyaları geçerlilikten geçirirken, tıpkı diğer verilerde olduğu gibi bunları da geçerlilik denetçisine parametre olarak geçersiniz.

Hata Mesajlarıyla Çalışmak

Bir `Validator` olgusunda `messages` metodunu çağırdıktan sonra, bir `MessageBag` olgusu alacaksınız. MessageBag sınıfında hata mesajlarıyla çalışmak için bir takım yararlı metodlar vardır.

Bir Alan İçin İlk Hata Mesajının Elde Edilmesi

```
echo $mesajlar->first('email');
```

Bir Alan İçin Tüm Hata Mesajlarının Elde Edilmesi

```
foreach ($mesajlar->get('email') as $mesaj)
{
        //
}
```

Tüm Alanlar İçin Tüm Hata Mesajlarının Elde Edilmesi

```
foreach ($mesajlar->all() as $mesaj)
{
        //
}
```

Bir Alan İçin Hata Mevcut Olup Olmadığının Tespiti

```
if ($mesajlar->has('email'))
{
        //
}
```

Bir Hata Mesajının Biçimlendirilmiş Olarak Alınması

```
echo $mesajlar->first('email', '<p>:mesaj</p>');
```

> **Not**: Ön tanımlı olarak, mesajlar Bootstrap'a uyumlu bir söz dizimiyle biçimlendirilir.

Tüm Hata Mesajlarının Biçimlendirilmiş Olarak Alınması

```
foreach ($mesajlar->all('<li>:mesaj</li>') as $mesaj)
{
        //
}
```

Hata Mesajları & Görünümler

Geçerlilik denetimi yaptıktan sonra aldığınız hata mesajlarını görünümlerinize gönderecek kolay bir yola ihtiyacınız olacak. Bu iş Laravel tarafından pratik bir şekilde halledilmektedir. Bir örnek olarak şu rotaları ele alalım:

```
Route::get('kayit', function()
{
        return View::make('uye.kayit');
});

Route::post('kayit', function()
{
        $kurallar = array(...);

        $geçerlilikyoklayici = Validator::make(Input::all(), $kurallar);

        if ($geçerlilikyoklayici->fails())
        {
                return Redirect::to('kayit')->withErrors($geçerlilikyoklayici);
        }
});
```

Dikkat ederseniz, geçerlilik başarısız olduğunda, `Validator` olgusunu `withErrors` metodunu kullanarak Redirect'e geçiriyoruz. Bu metod, hata mesajlarını oturuma flaş tipinde aktaracak, yani bir sonraki isteğe kadar kullanılabilir olacaktır.

Buna karşın, yine dikkat ederseniz GET rotamızda hata mesajlarını görünüme açık olarak bağlamak zorunda değiliz. Bunun nedeni, Laravel'in oturum verisinde hatalar olup olmadığını her zaman yoklaması ve olduğunu tespit etmesi halinde bunları otomatik olarak görünüme bağlamasıdır. **Bu itibarla, her istekte tüm görünümleriniz için bir `$errors` değişkeni mevcut olacağını unutmayın**, dolayısıyla siz `$errors` değişkeninin her zaman tanımlanmış olduğunu iç rahatı ile varsayıp, güvenle kullanabilirsiniz. Bu `$errors` değişkeni `MessageBag` sınıfından bir olgu olacaktır.

Bu durumda, yeniden yön verme sonrasında, otomatikman bağlanan `$errors` değişkenini görünümlerinizde kullanabilirsiniz:

```
<?php echo $errors->first('email'); ?>
```

Mevcut Geçerlilik Kuralları

Mevcut tüm geçerlilik kuralları ve bunların işlevleri aşağıda verilmiştir:

- Accepted
- Active URL
- After (Tarih)
- Alpha
- Alpha Dash

accepted

Geçerlilik bakılan alan *yes*, *on* veya *1* olmalıdır. Bu, "Hizmet Şartlarının" kabul edildiğinin doğrulanmasında işe yarar.

active_url

Geçerlilik bakılan alan `checkdnsrr` PHP fonksiyonuna göre geçerli bir URL olmalıdır.

after: *tarih*

Geçerlilik bakılan alan verilen bir tarihten sonraki bir değer olmalıdır. Tarihler PHP `strtotime` fonksiyonuna geçirilecektir.

alpha

Geçerlilik bakılan alan tamamen alfabe harfleri olmalıdır.

alpha_dash

Geçerlilik bakılan alan alfa-numerik karakterler yanında tire ve alt tire de olabilir.

alpha_num

Geçerlilik bakılan alan tamamen alfa-numerik karakterler olmalıdır.

before: *tarih*

Geçerlilik bakılan alan verilen bir tarihten önceki bir değer olmalıdır. Tarihler PHP `strtotime` fonksiyonuna geçirilecektir.

between: *min*, *max*

Geçerlilik bakılan alan verilen *min* ile *max* arasında bir büyüklükte olmalıdır. Stringler, sayılar ve dosyalar `size` kuralıyla aynı tarzda değerlendirilir.

confirmed

Geçerlilik bakılan alana uyan eden bir `falan_confirmation` alanı olmalıdır. Örneğin, geçerlilik bakılan alan `parola` ise, inputta karşılık gelen bir `parola_confirmation` alanı olmalıdır.

date

Geçerlilik bakılan alan `strtotime` PHP fonksiyonua göre uygun bir tarih olmalıdır.

date_format: *format*

Geçerlilik bakılan alan `date_parse_from_format` PHP fonksiyona göre tanımlanmış bir *format*'a uygun olmalıdır.

different: *alan*

Verilen *alan*, geçerlilik bakılan alandan farklı olmalıdır.

email

Geçerlilik bakılan alan bir e-mail adresi şeklinde biçimlendirilmiş olmalıdır.

exists: *tablo*, *sütun*

Geçerlilik bakılan alan verilen bir veritabanı tablosunda mevcut olmalıdır.

Exists Kuralının Basit Kulllanım Şekli

```
'il' => 'exists:iller'
```

Özel Bir Sütun İsminin Belirtilmesi

```
'il' => 'exists:iller,kisa_hali'
```

Sorguya "where" cümleciği olarak eklenecek daha fazla şart da belirtebilirsiniz:

```
'email' => 'exists:personel,email,hesap_id,1'
```

image

Geçerlilik bakılan alan bir imaj (jpeg, png, bmp veya gif) olmalıdır.

in: *falan*, *filan*,...

Geçerlilik bakılan alan verilen bir değerler listesinde olmalıdır.

integer

Geçerlilik bakılan alan bir tamsayı olmalıdır.

ip

Geçerlilik bakılan alan bir IP adresi olarak biçimlendirilmiş olmalıdır.

max: *deger*

Geçerlilik bakılan alan bir maksimum *deger*'den az olmalıdır. Stringler, sayılar ve dosyalar `size` kuralıyla aynı tarzda değerlendirilir.

mimes: *falan*, *filan*,...

Geçerlilik bakılan alan listelenen uzantılardan birine tekabül eden bir MIME tipinde olmalıdır.

MIME Kuralının Basit Kullanım Şekli

```
'resim' => 'mimes:jpeg,bmp,png'
```

min: *deger*

Geçerlilik bakılan alan bir asgari *deger*'den büyük olmalıdır. Stringler, sayılar ve dosyalar `size` kuralıyla aynı tarzda değerlendirilir.

not_in: *falan*,*filan*,...

Geçerlilik bakılan alan verilen değerler listesinde yer almamalıdır.

numeric

Geçerlilik bakılan alan sayısal bir değer olmalıdır.

regex: *desen*

Geçerlilik bakılan alan verilen düzenli ifadeye uygun olmalıdır.

Not: `regex` deseni kullanırken, özellikle düzenli ifade bir pipe karakteri (|) içeriyorsa, kuralları belirtmek için pipe ayıracı kullanmak yerine bir dizide belirtmek gerekli olabilir.

required

Geçerlilik bakılan alan input verisinde bulunmak zorundadır.

required_if: *alan*, *deger*

Şayet *alan* alanı *deger*'e eşit ise, geçerlilik bakılan alan girilmek zorundadır.

required_with: *falan*, *filan*,...

Geçerlilik bakılan alan, sadece belirtilen alanların bulunması durumunda bulunmak zorundadır.

required_without: *falan*, *filan*,...

Geçerlilik bakılan alan, sadece diğer belirtilen alanlar olmadığı takdirde bulunmak zorundadır.

same: *alan*

Verilen *alan* geçerlilik bakılan alanla aynı olmalıdır.

size: *deger*

Geçerlilik bakılan alan verilen *deger*'le aynı büyüklükte olmalıdır. String veriler için, *deger* harf sayısı anlamına gelir. Numerik veriler için, *deger* verilen bir tamsayı değeridir. Dosyalar için, *size* kilobayt cinsinden dosya boyutuna karşılık gelir.

unique: *tablo*, *sütun*, *haric*, *idSütunu*

Geçerlilik bakılan alan verilen bir veritabanı tablosunda benzersiz olmalıdır. Eğer `sütun` seçeneği belirtilmemişse, geçerlilik bakılan alan aynı zamanda sütun adı olarak kabul edilecektir.

Unique Kuralının Basit Kullanım Şekli

```
'email' => 'unique:uyeler'
```

Özel Bir Sütun Adının Belirtilmesi

```
'email' => 'unique:uyeler,email_adresi'
```

Verilen Bir ID İçin Unique Kuralının Göz Ardı Edilmesi

```
'email' => 'unique:uyeler,email_adresi,10'
```

Ek olarak Where Cümlecikleri Ekleme

Bir sorguya "where" cümlecikleri ekler gibi daha fazla şart ekleyebilirsiniz:

```
'email' => 'unique:users,email_address,NULL,id,account_id,1'
```

Yukarıdaki kuralda, sadece `account_id`'si `1` olan satırlar unique yoklamasına dahil edilecektir.

url

Geçerlilik bakılan alan bir URL şeklinde biçimlendirilmiş olmalıdır.

Duruma Göre Kurallar Ekleme

Bazen belli bir alanın başka bir alan 100'den fazla bir değere sahip olduğunda gerekli olmasını isteyebilirsiniz. Bu geçerlilik kurallarının eklenmesi sorun oluşturmak zorunda değildir. Öncelikle, asla değişmeyecek *statik kuralları* nızın olduğu bir `Validator` olgusu oluşturun:

```
$v = Validator::make($data, array(
        'email' => 'required|email',
        'games' => 'required|numeric',
));
```

Diyelim ki, game toplayıcıları için bir web uygulamamız var. Eğer bir game toplayıcısı uygulamamıza üye olursa ve onun 100'den fazla oyunu varsa, neden bu kadar çok oyunu olduğunu açıklamasını isteyelim. Örneğin, belki oyunları yeniden satan bir dükkanı vardır veya sadece toplamaktan hoşlanıyor olabilir. Bu gereksinimi, duruma göre eklemek için `Validator` olgusunda `sometimes` metodunu kullanabiliriz.

```
$v->sometimes('reason', 'required|max:500', function($input)
{
        return $input->games >= 100;
});
```

Bu `sometimes` metoduna geçirilen ilk parametre duruma bağlı olarak geçerlilik denetiminden geçireceğimiz alanın adıdır. İkinci parametre ise, eklemek istedğimiz kurallardır. Üçüncü parametre olarak geçirilen `Closure` `true` döndürürse, ilgili kural eklenecektir. Bu metod, karmaşık şartlı geçerlilikler inşa edilmesini çok kolaylaştırır. Bir defada birden çok alan için şartlı geçerlilik eklemeniz de mümkündür:

```
$v->sometimes(array('reason', 'cost'), 'required', function($input)
{
        return $input->games >= 100;
});
```

> **Not:** `Closure`'a geçirilen `$input` parametresi `Illuminate\Support\Fluent`'ın bir olgusu olacaktır ve input ve dosyalarınıza erişeceğiniz bir nesne olarak kullanılabilir.

Özel Hata Mesajları

Gerek duyduğunuzda, geçerlilik için ön tanımlı hata mesajları yerine özel hata mesajları kullanabilirsiniz. Özel mesaj belirtmek için birkaç yol var.

Validator'e Özel Mesaj Geçilmesi

```
$mesajlar = array(
        'required' => ':attribute alanı gereklidir.',
);

$geçerlilikyoklayici = Validator::make($input, $kurallar, $mesajlar);
```

Not: Buradaki `:attribute` yer tutucusu geçerlilik bakılan alanın gerçek adıyla değiştirilecektir. Geçerlilik mesajlarınızda diğer yer tutucuları da kullanabilirsiniz.

Diğer Geçerlilik Yer Tutucuları

```
$mesajlar = array(
        'same'    => ':attribute ve :other aynı olmalıdır.',
        'size'    => ':attribute tam olarak :size olmalıdır.',
        'between' => ':attribute :min ile :max arasında olmalıdır.',
        'in'      => ':attribute şu tiplerden birisi olmalıdır: :values',
);
```

Bazen sadece belirli bir alan için özel hata mesajları belirlemek isteyebilirsiniz:

Belli Bir Attribute İçin Özel Mesaj Belirlenmesi

```
$mesajlar = array(
        'email.required' => 'e-mail adresinizi bilmemiz gerekiyor!',
);
```

Bazı durumlarda, özel hata mesajlarınızı doğrudan `Validator`'e geçirmek yerine bir dil dosyasında belirtmek isteyebilirsiniz. Bunu yapmak için, mesajlarınızı `app/lang/xx/validation.php` dil dosyasındaki `custom` dizisine ekleyiniz.

Özel Mesajların Dil Dosyalarında Belirtilmesi

```
'custom' => array(
        'email' => array(
                'required' => 'e-mail adresinizi bilmemiz gerekiyor!',
        ),
),
```

Özel Geçerlilik Kuralları

Laravel'de her biri yararlı çok sayıda geçerlilik kuralı bulunmaktadır; bununla birlikte siz kendiniz de bazı kurallar belirlemek isteyebilirsiniz. Özel geçerlilik kuralı kayda geçirmenin bir yolu `Validator::extend` metodunu kullanmaktır:

Özel Bir Geçerlilik Kuralını Kayda Geçirme

```
Validator::extend('falan', function($attribute, $value, $parameters)
{
        return $value === 'falan';
});
```

> **Not**: `extend` metoduna geçilen kuralın adı "yılan tipi" (kelimeler boşluk olmaksızın, küçük harfli ve alt tire ile birleştirilmiş) olmalıdır.

Özel bir geçerlilik bitirme fonksiyonu (Closure) üç parametre alır: geçerlilik bakılacak `$attribute`'ın adı, bu niteliğin `$value`'i ve kurala geçilecek bir `$parameters` dizisi.

Bu `extend` metoduna bir bitirme fonksiyonu yerine bir sınıf ve metod da geçebilirsiniz:

```
Validator::extend('falan', 'FalanValidator@validate');
```

Özel kurallarınız için aynı zamanda bir hata mesajı da tanımlamanız gerekeceğini unutmayın. Bunu, ya aynı satırda özel hata mesaj dizisi kullanarak ya da geçerlilik dil dosyasına bir giriş eklemek suretiyle yapabilirsiniz.

Validator'ü genişletmek için bir bitirme fonksiyonu çağrısı kullanmak yerine, Validator sınıfının kendisini de genişletebilirsiniz. Bunu yapmak için, `Illuminate\Validation\Validator`'ü genişleten bir Validator sınıfı yazın. Validation metodlarınızı, başına `validate` getirerek bu sınıfa ekleyebilirsiniz:

Validator Sınıfının Genişletilmesi

```
<?php

class OzelValidator extends Illuminate\Validation\Validator {

        public function validateFalan($attribute, $value, $parameters)
        {
                return $value === 'falan';
        }

}
```

Daha sonra, özel Validator uzantınızı kayda geçirmeniz gerekiyor:

Özel Bir Validator Çözümleyicisinin Kayda Geçirilmesi

```
Validator::resolver(function($translator, $data, $rules, $messages)
{
        return new OzelValidator($translator, $data, $rules, $messages);
});
```

Özel bir geçerlilik kuralı oluştururken, bazı durumlarda, hata mesajlarıyla değiştirilecek özel yer tutucular tanımlamanız gerekebilir. Bunu, aynen yukarıda tarif edildiği gibi özel bir Validator oluşturup, bu validatore bir `replaceXXX` fonksiyonu ekleyerek gerçekleştirebilirsiniz.

```
protected function replaceFalan($message, $attribute, $rule, $parameters)
{
        return str_replace(':falan', $parameters[0], $message);
}
```

Temel Veritabanı Kullanımı

Yapılandırma

Laravel, veritabanı bağlantısını ve sorguları çalıştırmayı fazlasıyla kolay kılar. Veritabanı yapılandırma ayarları `app/config/database.php` dosyasında bulunmaktadır. Bu dosyada hem tüm veritabanı bağlantılarını tanımlayabilir, hem de hangi bağlantının varsayılan olarak kullanılacağını seçebilirsiniz. Bu dosyada, desteklenen veritabanı sistemlerinin tümü için örnekler verilmiştir.

Laravel tarafından desteklenen veritabanı sistemleri: MySQL, Postgres, SQLite, ve SQL Server.

Sorguları Çalıştırma

Veritabanı bağlantılarını bir kere yapılandırdıktan sonra `DB` sınıfını kullanarak sorgularını çalıştırabilirsiniz.

Kayıt Çekme (Select)

```
1  $sonuclar = DB::select('select * from uyeler where id = ?', array(1));
```

`select` metodu sonuçları her zaman `dizi` tipinde döndürür.

Yeni Kayıt Ekleme (Insert)

```
1  DB::insert('insert into uyeler (id, isim) values (?, ?)', array(1, 'Emre'));
```

Kayıt Güncelleme (Update)

```
1  DB::update('update uyeler set oy = 100 where isim = ?', array('Hakan'));
```

Kayıt Silme (Delete)

```
1  DB::delete('delete from uyeler');
```

> **Not:** `update` ve `delete` sorguları, bu işlemlerden etkilenen satır sayısını döndürür.

Genel Bir Sorgu Çalıştırma

```
DB::statement('drop table uyeler');
```

`DB::listen` metodunu kullanarak sorgu olaylarını dinleyebilirsiniz:

Sorgu Olaylarını Dinleme

```
DB::listen(function($sql, $bindings, $time)
{
        //
});
```

Veritabanı İşlemleri

Bir veritabanı işleminde, birden fazla işlemi birden gerçekleştirmek için, 'transaction' metodunu kullanabilirsiniz:

```
DB::transaction(function()
{
        DB::table('uyeler')->update(array('votes' => 1));

        DB::table('posts')->delete();
});
```

Bağlantılara Erişme

Birden fazla bağlantı kullandığınız durumlarda, bu bağlantılara `DB::connection` metodu aracılığı ile ulaşabilirsiniz.

```
$uyeler = DB::connection('foo')->select(...);
```

Ayrıca temel PDO örneğine de ulaşabilirsiniz:

```
$pdo = DB::connection()->getPdo();
```

Bazen veritabanına tekrar bağlanmaya ihtiyacınız olabilir.

```
DB::reconnect('foo');
```

Sorgu Günlükleri

Varsayılan olarak, Laravel güncel istek için çalıştırılabilecek tüm sorgular için bellekte bir günlük tutar. Bununla birlikte, bu bazı durumlarda, örneğin çok sayıda satır eklerken, uygulamanın aşırı bellek kullanmasına neden olabilir. Günlüğü devre dışı bırakmak için `disableQueryLog` metodunu kullanabilirsiniz.

```
DB::connection()->disableQueryLog();
```

Sorgu Oluşturucusu

Giriş

Veritabanı sorgu oluşturucusu veritabanı sorguları oluşturulması ve çalıştırılması için kullanışlı ve akıcı bir arayüz sağlar. Uygulamanızdaki pek çok veritabanı işlemini bununla gerçekleştirebilirsiniz ve desteklenen tüm veritabanı sistemlerinde çalışmaktadır.

> **Not:** Laravel sorgu oluşturucusu, uygulamanızı SQL enjeksiyon saldırılarına karşı korumak için PDO parametre bağlayıcı kullanmaktadır. Bağlayıcı olarak geçirilen yazıların temizlenmesine gerek yoktur. Temizlenmeyen yalnızca iki metod vardır, bunlar `groupBy` ve `orderBy`'dır. Kullanıcı girdilerini bu metodlara doğrudan geçmemelisiniz.

Seçmeler

Bir Tablonun Bütün Satırlarının Alınması

```
$uyeler = DB::table('uyeler')->get();

foreach (uyeler as $uye)
{
        var_dump($uye->isim);
}
```

Bir Tablonun Tek Bir Satırının Alınması

```
$uye = DB::table('uyeler')->where('isim', 'Can')->first();

var_dump($uye->isim);
```

Bir Satırın Tek Bir Sütununun Alınması

```
$isim = DB::table('uyeler')->where('isim', 'Can')->pluck('isim');
```

Sütun Değerlerinden Oluşan Bir Liste Elde Edilmesi

```
$roller = DB::table('roller')->lists('unvan');
```

Bu metod rol ünvanlarından oluşan bir dizi döndürecektir. Döndürülen dizi için özel bir anahtar sütun da belirleyebilirsiniz:

```
$roller = DB::table('roller')->lists('unvan', 'isim');
```

Bir Select Bendinin Belirlenmesi

```
$uyeler = DB::table('uyeler')->select('isim', 'email')->get();

$uyeler = DB::table('uyeler')->distinct()->get();

$uyeler = DB::table('uyeler')->select('isim as uye_adi')->get();
```

Mevcut Bir Sorguya Bir Select Bendinin Eklenmesi

```
$sorgu = DB::table('uyeler')->select('isim');

$uyeler = $query->addSelect('yas')->get();
```

Where Kullanımı

```
$uyeler = DB::table('uyeler')->where('puan', '>', 100)->get();
```

Or Cümleleri

```
$uyeler = DB::table('uyeler')
                    ->where('puan', '>', 100)
                    ->orWhere('isim', 'Can')
                    ->get();
```

Where Between Kullanımı

```
$uyeler = DB::table('uyeler')
                    ->whereBetween('puan', array(1, 100))->get();
```

Bir Dizi Aracılığıyla Where In Kullanımı

```
$uyeler = DB::table('uyeler')
                    ->whereIn('id', array(1, 2, 3))->get();

$uyeler = DB::table('uyeler')
                    ->whereNotIn('id', array(1, 2, 3))->get();
```

Değer Girilmemiş Kayıtları Bulmak için Where Null Kullanımı

```
$uyeler = DB::table('uyeler')
                    ->whereNull('guncelleme_vakti')->get();
```

Order By, Group By ve Having

```
$uyeler = DB::table('uyeler')
                    ->orderBy('name', 'desc')
                    ->groupBy('count')
                    ->having('count', '>', 100)
                    ->get();
```

Offset ve Limit

```
$uyeler = DB::table('uyeler')->skip(10)->take(5)->get();
```

Joinler

Sorgu oluşturucusu join cümleleri yazmak için de kullanılabilir. Şu örneklere bir bakın:

Temel Join Cümleleri

```
DB::table('uyeler')
            ->join('kisiler', 'uyeler.id', '=', 'kisiler.uye_id')
            ->join('siparisler', 'uyeler.id', '=', 'siparisler.uye_id')
            ->select('uyeler.id', 'kisiler.telefon', 'siparisler.fiyat');
```

Left Join Cümlesi

```
DB::table('uyeler')
            ->leftJoin('makaleler', 'uyeler.id', '=', 'makaleler.uye_id')
            ->get();
```

Daha ileri join cümleleri de tanımlayabilirsiniz:

```
DB::table('uyeler')
        ->join('kisiler', function($join)
        {
                $join->on('uyeler.id', '=', 'kisiler.uye_id')->orOn(...);
        })
        ->get();
```

İleri Where Cümleleri

Kimi zaman "where exists" veya içi içe parametre gruplaması gibi daha ileri where cümleleri oluşturmanız gerekebilir. Laravel sorgu oluşturucusu bunu da halledecektir:

Parametre Gruplaması

```
DB::table('uyeler')
            ->where('isim', '=', 'Can')
            ->orWhere(function($query)
            {
                    $query->where('puan', '>', 100)
                          ->where('unvan', '<>', 'Müdür');
            })
            ->get();
```

Yukardaki sorgu aşağıdaki SQL cümlesini oluşturacaktır:

```
select * from uyeler where isim = 'Can' or (puan > 100 and unvan <> 'Müdür')
```

Exists Cümleleri

```
DB::table('uyeler')
            ->whereExists(function($query)
            {
                    $query->select(DB::raw(1))
                          ->from('siparisler')
                          ->whereRaw('siparisler.uye_id = uyeler.id');
            })
            ->get();
```

Yukardaki sorgu aşağıdaki SQL cümlesini oluşturacaktır:

```
select * from uyeler
where exists (
        select 1 from siparisler where siparisler.uye_id = uyeler.id
)
```

Kümeleme (Aggregate) İşlemleri

Sorgu oluşturucusu; `count`, `max`, `min`, `avg` ve `sum` gibi çeşitli kümeleme metodları da sağlamaktadır.

Aggregate Metodlarının Kullanımı

```
$uyeler = DB::table('uyeler')->count();

$fiyat = DB::table('siparisler')->max('fiyat');

$fiyat = DB::table('siparisler')->min('fiyat');

$fiyat = DB::table('siparisler')->avg('fiyat');

$toplam = DB::table('uyeler')->sum('puan');
```

Ham İfadeler

Bazen bir sorguda ham ifade kullanma ihtiyacı duyabilirsiniz. Bu ifadeler sorguya doğrudan yazı olarak enjekte edileceğinden, bir SQL enjeksiyon noktası oluşturmamaya özen gösteriniz. Ham ifade oluşturmak için `DB::raw` metodunu kullanabilirsiniz:

Ham İfade Kullanımı

```
$users = DB::table('uyeler')
        ->select(DB::raw('count(*) as uye_adedi, durum'))
        ->where('durum', '<>', 1)
        ->groupBy('durum')
        ->get();
```

Bir Sütun Değerinin Artırılması veya Azaltılması

```
DB::table('uyeler')->increment('puan');

DB::table('uyeler')->increment('puan', 5);

DB::table('uyeler')->decrement('puan');

DB::table('uyeler')->decrement('puan', 5);
```

Ayrıca, güncellenecek ek sütunlar belirtebilirsiniz:

```
DB::table('uyeler')->increment('puan', 1, array('isim' => 'Tuana Şeyma'));
```

Eklemeler

Bir Tabloya Kayıt Eklenmesi

```
DB::table('uyeler')->insert(
        array('email' => 'can@numune.com', 'puan' => 0)
);
```

Şayet tabloda otomatik artan bir id alanı varsa, bir kayıt eklemek ve oluşan otomatik id'i öğrenmek için `insertGetId` metodu kullanılabilir:

Otomatik Artan Bir Id Alanı Olan Tabloya Kayıt Eklenmesi

```
$id = DB::table('uyeler')->insertGetId(
        array('email' => 'can@numune.com', 'puan' => 0)
);
```

> **Not**: PostgreSQL ile kullanıldığı zaman insertGetId metodu, otomatik artan alanın adının da "id" olmasını bekler.

Bir Tabloya Birden Çok Kayıt Eklenmesi

```
DB::table('uyeler')->insert(array(
        array('email' => 'sinan@numune.com', 'puan' => 0),
        array('email' => 'ozan@numune.com', 'puan' => 0),
));
```

Güncellemeler

Bir Tablodaki Kayıtların Güncellenmesi

```
DB::table('uyeler')
            ->where('id', 1)
            ->update(array('puan' => 1));
```

Silmeler

Bir Tablodaki Kayıtların Silinmesi

```
DB::table('uyeler')->where('puan', '<', 100)->delete();
```

Bir Tablodaki Tüm Kayıtların Silinmesi

```
DB::table('uyeler')->delete();
```

Bir Tablonun Budanması

```
DB::table('uyeler')->truncate();
```

Birleştirmeler

Sorgu oluşturucusu, iki ayrı sorgunun tek bir "birlik" haline getirilmesi için de hızlı bir yol sağlamaktadır:

Bir Birleştirme Sorgusu Yapılması

```
$ilksorgu = DB::table('uyeler')->whereNull('ismi');

$users = DB::table('uyeler')->whereNull('soy_ismi')->union($ilksorgu)->get();
```

Ayrıca `unionAll` metodu da mevcut olup, aynı `union` gibi kullanılır.

Sorguların Bellekte Saklanması

Bir sorgunun sonuçları `remember` metodu kullanılarak bellekte saklanabilir:

Bir Sorgu Sonucunun Bellekte Saklanması

```
$uyeler = DB::table('uyeler')->remember(10)->get();
```

Bu örnekte, sorgunun sonuçları on dakika süreyle bellekte saklanacaktır. Sonuçlar bellekte tutulduğu süre boyunca bu sorgu artık veritabanında çalıştırılmayacak, onun yerine sonuçlar uygulamanız için belirlediğiniz ön tanımlı bellekleme sürücüsü tarafından yüklenecektir.

Eloquent ORM

Giriş

Laravelle gelen "Eloquent ORM", veritabanınızla çalışırken kullanacağınız güzel ve sade bir ActiveRecord uygulaması sağlamaktadır. Her veritabanı tablosu, bu tabloyla etkilişim için kullanıcak kendine has bir "Model" sahibidir.

Başlamadan önce, `app/config/database.php`'de bir veritabanı bağlantısı yapılandırmış olun.

Temel Kullanım

Öncelikle bir Eloquent modeli oluşturunuz. Modeller tipik olarak `app/models` klasöründe yer alır, fakat siz modellerinizi `composer.json` dosyanıza göre otomatik yükleme yapabileceğiniz başka bir yere de koyabilirsiniz.

Bir Eloquent Modelinin Tanımlanması

```
1  class Uye extends Eloquent {}
```

Dikkat ederseniz Eloquent'e `Uye` modelimiz için hangi tabloyu kullanacağımızı söylemedik. Eğer açıkça başka bir isim belirtilmezse tablo isimi olarak sınıf adının ingilizde çoğulunun küçük harf hali kullanılacaktır. Dolayısıyla bizim örneğimizde Eloquent, `Uye` modelinin `uyes` tablosundaki kayıtları tutacağını varsayacaktır. Tablo ismini açıkça belirtmek için modelinizde bir `table` özelliği tanımlayınız:

```
1  class Uye extends Eloquent {
2
3          protected $table = 'uyeler';
4
5  }
```

> **Not**: Eloquent'in başka bir ön kabulü de her tablonun `id` adında bir primer key sütunu olduğudur. Bu kuralı aşmak için de bir `primaryKey` özelliği tanımlamanız gerekecek. Benzer şekilde, modeliniz kullanılacağı zaman kullanılacak veritabanı bağlantısının adını değiştirmek için bir `connection` özelliği tanımlayabilirsiniz.

Bir model tanımladıktan sonra artık tablonuzda kayıt oluşturmaya ve ondan kayıt getirmeye başlayabilirsiniz. Tablolarınıza ön tanımlı olarak `updated_at` ve `created_at` sütunları koymanız gerektiğine dikkat ediniz. Şayet bu sütunların otomatik olarak tutulmasını istemiyorsanız, modelinizdeki `$timestamps` özelliğini `false` olarak ayarlayınız.

Tüm Modellerin Alınması

```
1 $uyeler = Uye::all();
```

Birincil Alana Göre Bir Kaydın Alınması

```
1 $uye = Uye::find(1);
2
3 var_dump($uye->isim);
```

> **Not:** Sorgu Oluşturucusu[99]'nda bulunan tüm metodlar Eloquent modellerini sorgularken de kullanılabilir.

Birincil Alana Göre Bir Model Alınması ya da Ortaya Bir İstisna Çıkartılması

Bazı durumlarda bir model bulunamadığında bir istisna çıkartmak, böylece bir `App::error` işleyicisi kullanarak istisnayı yakalayabilmek ve bir 404 sayfası göstermek isteyebilirsiniz.

```
1 $model = Uye::findOrFail(1);
```

Bu hata işleyicinin kaydını yapmak için `ModelNotFoundException`'i dinlemek gerekir.

```
1 use Illuminate\Database\Eloquent\ModelNotFoundException;
2
3 App::error(function(ModelNotFoundException $e)
4 {
5         return Response::make('Bulunamadı', 404);
6 });
```

Eloquent Modelleri Kullanarak Sorgu Yapma

[99] /docs/queries

```
$uyeler = Uye::where('puan', '>', 100)->take(10)->get();

foreach ($uyeler as $uye)
{
        var_dump($uye->isim);
}
```

Tabi ki, sorgu oluşturucusunun kümeleme fonksiyonlarını da kullanabilirsiniz.

Eloquent Küme Metodları

```
$adet = Uye::where('puan', '>', 100)->count();
```

Gereken sorguyu fluent arayüzüyle üretemediğiniz zaman `whereRaw` kullanabilirsiniz:

```
$uyeler = Uye::whereRaw('yas > ? and puan = 100', array(25))->get();
```

Query Bağlantısının Belirtilmesi

Bir Eloquent sorgusu çalıştırırken hangi bağlantının kullanılacağını da belirleyebilirsiniz. `On` metodunu kullanmanız yeterlidir:

```
$uyeler = Uye::on('bağlantı-adı')->find(1);
```

Toplu Atama

Yeni bir model oluşturulurken model oluşturucuya niteliklerden oluşan bir dizi geçersiniz. Bu nitelikler bu durumda modele "toplu atama" aracılığıyla atanır. Bu gayet uygun bir yaklaşımdır, fakat bir kullanıcı girdisi bir modele körleme geçirildiği takdirde **ciddi (serious)** bir güvenlik sorunu olabilecektir. Kullanıcı girdisi bir modele körlemesine geçirilirse, bu kullanıcı modelin niteliklerinin **birisini (any)** ve **hepsini (all)** değiştirebilecektir. Bu sebepler yüzünden, tüm Eloquent modelleri ön tanımlı olarak toplu atamaya karşı koyar.

Başlamak için modelinizde `fillable` veya `guarded` özelliğini ayarlayınız.

Bunlardan `fillable` özelliği hangi niteliklerin toplu atanacaklarını belirler. Bu işlem sınıf ya da olgu düzeyinde ayarlanabilir.

Bir Modelde Fillable Niteliklerin Tanımlanması

```
class Uye extends Eloquent {

        protected $fillable = array('ismi', 'soy_ismi', 'email');

}
```

Bu örnekte, sadece belirttiğimiz üç nitelik toplu atanabilecektir.

`fillable`'in tersi `guarded`'dir ve bir "beyaz-liste" yerine bir "kara-liste" olarak iş görür:

Bir Modelde Guarded Niteliklerin Tanımlanması

```
class Uye extends Eloquent {

        protected $guarded = array('id', 'parola');

}
```

Yukardaki örneğe göre `id` ve `parola` nitelikleri toplu atana **mayacaktır**. Diğer tüm nitelikler toplu atanabilecektir. Toplu atamayı niteliklerin **hepsi (all)** için bloke etmeyi de seçebilirsiniz:

Toplu Atamanın Tüm Nitelikler İçin Engellenmesi

```
protected $guarded = array('*');
```

Ekleme, Güncelleme, Silme

Veritabanında bir modelden yeni bir kayıt oluşturmak için, yeni bir model olgusu oluşturun ve `save` metodunu çağırın.

Yeni Bir Modelin Kaydedilmesi

```
$uye = new Uye;

$uye->isim = 'Can';

$uye->save();
```

> **Not**: Tipik olarak, Eloquent modellerinizde otomatik artan anahtarlar olacaktır. Ama siz kendi keylerinizi belirlemek isterseniz, modelinizdeeki `incrementing` özelliğini `false` olarak ayarlayın.

Yeni bir modeli tek satırda kaydetmek için `create` metodunu kullanabilirsiniz. Eklenen model olgusu bu metoddan döndürülecektir. Ancak, tüm Elequent modelleri toplu atamaya karşı korunumlu oldukları için, bunu yapmadan önce modelinizde bir `fillable` veya `guarded` özelliği belirlemeniz gerekecektir.

Modeldeki Korunumlu Niteliklerin Ayarlanması

```
class Uye extends Eloquent {

        protected $guarded = array('id', 'hesap_no');

}
```

Model Create Metodunun Kullanımı

```
$uye = Uye::create(array('isim' => 'Can'));
```

Bir modeli güncellemek için onu getirir, bir niteliğini değiştirir, sonra da `save` metodunu kullanabilirsiniz:

Getirilen Bir Modelin Güncellenmesi

```
$uye = Uye::find(1);

$uye->email = 'can@filan.com';

$uye->save();
```

Bazen sadece bir modeli değil, onun bütün ilişkilerini de kaydetmek isteyebilirsiniz. Bunu yapmak için `push` metodunu kullanın:

Bir Model ve İlişkilerinin Kaydedilmesi

```
$uye->push();
```

Ayrıca, bir modeller kümesinde güncelleme sorguları da çalıştırabilirsiniz:

```
$satirSayisi = Uye::where('puan', '>', 100)->update(array('durum' => 2));
```

Bir modeli silmek için olgu üzerinde `delete` metodunu çağırın:

Mevcut Bir Modelin Silinmesi

```
$uye = Uye::find(1);

$uye->delete();
```

Mevcut Bir Modelin Key Aracılığıyla Silinmesi

```
Uye::destroy(1);

Uye::destroy(array(1, 2, 3));

Uye::destroy(1, 2, 3);
```

Gayet tabii, bir modeller kümesinde bir silme sorgusu da çalıştırabilirsiniz:

```
$satirSayisi = Uye::where('puan', '>', 100)->delete();
```

Eğer bir modelde sadece zaman damgalarını güncellemek istiyorsanız, `touch` metodunu kullanabilirsiniz:

Bir Modelin Sadece Zaman Damgalarının Güncellenmesi

```
$uye->touch();
```

Zaman Damgaları

Ön tanımlı olarak, veritabanı tablonuzdaki `created_at` ve `updated_at` sütunlarının idamesini otomatik olarak Eloquent yapacaktır. Size tek düşen `datetime` tipindeki bu iki alanı tablonuza eklemektir, geri kalan işleri Eloquent üstlenecektir. Şayet siz bu sütunların idamesini Eloquent'in yapmasını istemiyorsanız, modelinize şu özelliği eklemeniz gerekir:

Otomatik Zaman Damgalarının Devre Dışı Bırakılması

```
class Uye extends Eloquent {

        protected $table = 'uyeler';

        public $timestamps = false;

}
```

Zaman damgalarınızın biçimini özelleştirmek isterseniz, modelinizdeki `freshTimestamp` metodunu ezebilirsiniz(override):

Özel Bir Zaman Damgası Biçiminin Şart Koşulması

```
class Uye extends Eloquent {

        public function freshTimestamp()
        {
                return time();
        }

}
```

Belirsiz Silme

Bir model belirsiz silindiğinde, aslında veritabanınızdan çıkartılmaz. Onun yerinde kayıttaki bir `deleted_at` zaman damgası ayarlanır. Bir modeli için belirsiz silmeler yapılabilmesi için modelinizde `softDelete` özelliğine atama yapmanız gerekir:

```
class Uye extends Eloquent {

        protected $softDelete = true;

}
```

Tablonuza bir `deleted_at` sütunu eklemek için ise, bir migrasyondan `softDeletes` metodunu kullanabilirsiniz:

```
$table->softDeletes();
```

Şimdi, artık modelinizde `delete` metodunu çağırdığınız zaman, bu `deleted_at` sütunu güncel zaman damgasına ayarlanacaktır. Belirsiz silme kullanılan bir model sorgulandığında, "silinmiş olan" modeller sorgu sonuçlarına dahil edilmeyecektir. Bir sonuç kümesinde belirsiz silinmiş modellerin gözükmesini zorlamak için sorgunuzda `withTrashed` metodunu kullanınız:

Belirsiz Silinmiş Modelleri Sonuçlara Girmeye Zorlama

```
$uyeler = Uye::withTrashed()->where('hesap_no', 1)->get();
```

Sonuç kümenizde **sadece** belirsiz silinmiş modellerin olmasını istiyorsanız, `onlyTrashed` metodunu kullanabilirsiniz:

```
$uyeler = Uye::onlyTrashed()->where('hesap_no', 1)->get();
```

Belirsiz silinmiş bir modeli tekrar etkin hale getirmek için, `restore` metodunu kullanın:

```php
$uye->restore();
```

`restore` metodunu bir sorguda da kullanabilirsiniz:

```php
Uye::withTrashed()->where('hesap_no', 1)->restore();
```

`restore` metodu ilişkilerde de kullanılabilir:

```php
$uye->postalar()->restore();
```

Bir modeli veritabanından gerçekten çıkartmak istediğinizde, `forceDelete` metodunu kullanabilirsiniz:

```php
$uye->forceDelete();
```

`forceDelete` metodu ilişkilerde de çalışır:

```php
$uye->postalar()->forceDelete();
```

Belli bir model olgusunun belirsiz silme özelliğine sahip olup olmadığını öğrenmek için, `trashed` metodunu kullanabilirsiniz:

```php
if ($uye->trashed())
{
        //
}
```

Sorgu Kapsamları

Kapsamlar size sorgu mantığınızı modellerinizde tekrar tekrar kullanma imkanı verir. Bir kapsam tanımlamak için bir model metodunun başına `scope` getirmeniz yeterlidir:

Bir Sorgu Kapsamının Tanımlanması

```
class Uye extends Eloquent {

        public function scopePopular($query)
        {
                return $query->where('puan', '>', 100);
        }

        public function scopeKadin($query)
        {
                return $query->whereGender('W');
        }

}
```

Bir Sorgu Kapsamının Kullanılması

```
$uyeler = Uye::popular()->kadin()->orderBy('created_at')->get();
```

Dinamik Kapsamlar

Bazen parametreler kabul eden kapsam tanımlamak isteyebilirsiniz. Yapmanız gereken kapsam metoduna parametrelerinizi eklemek:

```
class Uye extends Eloquent {

        public function scopeOfType($query, $type)
        {
                return $query->whereType($type);
        }

}
```

Parametreyi kapsamın çağrısına geçin:

```
$uyeler = Uye::ofType('moderator')->get();
```

İlişkiler

Pek tabii, veritabanı tablolarınız büyük ihtimalle bir diğeriyle ilişkilidir. Örneğin bir blog yazısında çok sayıda yorum olabilir veya bir sipariş onu ısmarlayan kullanıcı ile ilişkili olacaktır. Eloquent bu ilişkileri kolayca yönetmenizi ve rahat çalışmanızı sağlar. Laravel dört tip ilişkiyi desteklemektedir:

Birden Bire

Birden bire şeklindeki bir ilişki çok basit bir ilişikidir. Örneğin, bir `Uye` modelinin bir `Telefon`'u olabilir. Eloquent'de bu ilişkiiyi şöyle tanımlayabiliriz:

Birden Bire Tarzı İlişki Tanımlama

```
class Uye extends Eloquent {

        public function tel()
        {
                return $this->hasOne('Telefon');
        }

}
```

`hasOne` metoduna geçirilen ilk parametre ilişkili modelin adıdır. İlişki tanımlandıktan sonra onu Eloquent'in dinamik özellikler'ini kullanarak elde edebiliriz:

```
$tel = Uye::find(1)->tel;
```

Bu cümlenin gerçekleştirdiği SQL şunlardır (tablo isimleri model tanımında özel olarak belirtilmedi ise tablo ismi olarak model isminin küçük harfli çoğul halinin kullanıldığını hatırlayınız):

```
select * from uyes where id = 1

select * from telefons where uye_id = 1
```

Eloquent'in ilişkideki yabancı key'in ne olduğuna model adına göre karar verdiğine dikkat ediniz. Şimdiki örnekte `Telefon` modelinin `uye_id` adlı bir yabancı key kullandığı varsayılmaktadır. Siz nu ön kuralı değiştirmek istiyorsanız `hasOne` metoduna ikinci bir parametre geçebilirsiniz:

```
return $this->hasOne('Telefon', 'mahsus_key');
```

`Telefon` modeli üzerinde ilişkinin tersini tanımlamak için, `belongsTo` metodunu kullanınız:

Bir İlişkinin Tersinin Tanımlanması

```
class Telefon extends Eloquent {

        public function uye()
        {
                return $this->belongsTo('Uye');
        }

}
```

Birden Birçoğa

Birde birçoğa ilişki örneği olarak birçok yorum yapılmış bir blog yazısı verilebilir. Bu ilişkiyi de şöyle modelleyebiliriz:

```
class Makale extends Eloquent {

        public function yorumlar()
        {
                return $this->hasMany('Yorum');
        }

}
```

Şimdi artık bir makalenin yorumlarına dinamik özellik aracılığıyla ulaşabiliriz:

```
$yorumlar = Makale::find(1)->yorumlar;
```

Hangi yorumların alınacağını daha da kısıtlamak için `yorumlar` metodunu çağırabilir ve şartlar koşmayı sürdürebilirsiniz:

```
$yorumlar = Makale::find(1)->yorumlar()->where('baslik', '=', 'bu')->first();
```

Tıpkı hasOne'de olduğu gibi konvansiyonel yabancı key varsayımını `hasMany` metoduna ikinci bir parametre geçerek değiştirebilirsiniz:

```
return $this->hasMany('Yorum', 'mahsus_key');
```

İlişkinin tersini `Yorum` modelinde tanımlamak için, `belongsTo` metodu kullanılmaktadır:

Bir İlişkinin Tersinin Tanımlanması

```
class Yorum extends Eloquent {

        public function makale()
        {
                return $this->belongsTo('Makale');
        }

}
```

Birçoktan Birçoğa

Birçoktan birçoğa ilişkiler daha karmaşık bir ilişki tipidir. Bu tarz bir ilişki örneği bir üyenin birçok rolü olması, aynı zamanda bu rollerin başka kullanıcılar tarafından da paylaşılmasıdır. Örneğin birçok üye "Müdür" rolünde olabilir. Bu ilişki için üç veritabanı tablosu gereklidir: `uyeler`, `roller` ve `rol_uye`. Bu `rol_uye` tablosu ilişkili model isimlerinin alfabetik sıralamasına göre adlandırılır ve `uye_id` ve `rol_id` sütunlarına sahip olmalıdır (model isimlerine alttire ve id eklenmiş iki alan).

Birçoktan birçoğa ilişikileri `belongsToMany` metodunu kullanarak tanımlayabiliyoruz:

```
class Uye extends Eloquent {

        public function roller()
        {
                return $this->belongsToMany('Rol');
        }

}
```

Artık rolleri `Uye` modeli aracılığıyla getirebiliriz:

```
$roller = Uye::find(1)->roller;
```

Pivot tablo ismi olarak ön kabullü tablo ismi yerine başka bir isim kullanmak isterseniz, bunu `belongsToMany` metoduna ikinci bir parametre geçerek gerçekleştirebilirsiniz:

```
return $this->belongsToMany('Rol', 'uye_rollleri');
```

İlişkili key için konvansiyonel yaklaşımı da değiştirebilirsiniz:

```
    return $this->belongsToMany('Rol', 'uye_rollleri', 'user_id', 'foo_id');
```

Ve tabii ki ilişkinin tersini `Rol` modelinde de tanımlayabilirsiniz:

```
class Rol extends Eloquent {

        public function uyeler()
        {
                return $this->belongsToMany('Uye');
        }

}
```

Çokbiçimli İlişkiler

Çokbiçimli (Polimorfik) İlişkiler bir modelin tek bir ilişkilendirme ile birden çok modele ait olmasına imkan verir. Örneğin, kendisi ya bir personel modeline ya da bir siparis modeline ait olan bir foto modeliniz olduğunu düşünün. Bu ilişkiyi şu şekilde tanımlayacağız:

```
class Foto extends Eloquent {

        public function resim()
        {
                return $this->morphTo();
        }

}

class Personel extends Eloquent {

        public function fotolar()
        {
                return $this->morphMany('Foto', 'resim');
        }

}

class Siparis extends Eloquent {

        public function fotolar()
        {
                return $this->morphMany('Foto', 'resim');
        }

}
```

Artık bir personel ya da siparişe ait fotoları elde edebiliriz:

Çokbiçimli Bir İlişkinin Getirilmesi

```
$personel = Personel::find(1);

foreach ($personel->fotolar as $foto)
{
        //
}
```

Ancak, "çokbiçimli" ilişkinin gerçek farkını bir personel veya siparişe `Foto` modelinden erişebilmekle görürsünüz:

Çokbiçimli Bir İlişkinin Sahibinin Getirilmesi

```
$foto = Foto::find(1);

$resim = $foto->resim;
```

`Foto` modelindeki `resim` ilişkisi, fotonun sahibi olan modele bağlı olarak ya bir `Personel` ya da bir `Siparis` olgusu döndürecektir.

Bunun nasıl çalıştığını anlamanıza yardımcı olmak için veritabanı yapımızı polimorfik bir ilişkiye açalım:

Polymorphic Relation Table Structure

```
personel
        id - integer
        isim - string

siparisler
        id - integer
        fiyat - integer

fotolar
        id - integer
        dosyayolu - string
        resim_id - integer
        resim_type - string
```

Buradaki anahtar alanların The key fields to notice here are the on the `fotolar` tablosundaki `resim_id` and `resim_type` olduğuna dikkat ediniz. Buradaki ID, fotonun sahibi olan personel veya siparişin ID'ini, TYPE ise sahip olan modelin sınıf adını tutacaktır. Böylece ORM, `resim` ilişkisiyle erişildiğinde döndürülecek sahip modelin hangisi olduğunu tespit edebilecektir.

İlişkilerin Sorgulanması

Bir modelin kayıtlarına erişirken, sonuçları bir ilişki varlığına göre sınırlamak isteyebilirsiniz. Diyelim ki, en az bir yorum yapılmış tüm blog makalelerini çekmek istediniz. Bunu yapmak için `has` metodunu kullanabilirsiniz:

Seçerken İlişkilerin Yoklanması

```
$makaleler = Makale::has('yorumlar')->get();
```

Ayrıca, bir işlemci ve bir sayı da belirleyebilirsiniz, örneğin üç ve daha çok yorum almış makaleleri getirmek için:

```
$makaleler = Makale::has('yorumlar', '>=', 3)->get();
```

Dinamik Özellikler

Eloquent, ilişkilerinize dinamik özellikle yoluyla erişme imkanı verir. Eloquent ilişkiyi sizin için otomatik olarak yükleyecektir. Hatta, `get` (birden birçoğa ilişkiler için) metodunun mu yoksa `first` (birden bire ilişkiler için) metodunun mu çağırılacağını bilecek kadar akıllıdır. İlişkiyle aynı isimli dinamik bir özellik aracılığı ile erişileblir olacaktır. Örneğin, şu `$telefon` modelinde:

```
class Telefon extends Eloquent {

        public function uye()
        {
                return $this->belongsTo('Uye');
        }

}

$telefon= Telefon::find(1);
```

Bu kullanının email'ini şu şekilde göstermek yerine:

```
echo $telefon->uye()->first()->email;
```

Buradaki gibi basit bir hale kısaltılabilir:

```
echo $telefon->uye->email;
```

Ateşli Yüklemeler

Ateşli yükleme N + 1 sorgu problemini gidermek içindir. Örnek olarak, `Yazar` ile ilişkilendirilmiş bir `Kitap` modelini düşünün. İlişki de şöyle tanımlanmış olsun:

```
class Kitap extends Eloquent {

        public function yazar()
        {
                return $this->belongsTo('Yazar');
        }

}
```

Şimdi, şu kodu ele alalım:

```
foreach (Kitap::all() as $kitap)
{
        echo $kitap->yazar->isim;
}
```

Bu döngü tablodaki kitapların hepsini almak için 1 sorgu çalıştıracak, sonra da yazarını elde etmek için her bir kitabı sorgulayacaktır. Yani, eğer 25 kitabımız varsa bu döngü 26 sorgu çalıştıracaktır.

Neyseki, sorgu sayısını büyük ölçüde azaltan ateşli yükleme kullanabiliriz. Ateşli yüklenecek ilişkiler `with` metodu aracılığıyla belirlenebilmektedir:

```
foreach (Kitap::with('yazar')->get() as $kitap)
{
        echo $kitap->yazar->isim;
}
```

Yukardaki döngüde sadece iki sorgu çalıştırılacaktır (model tanımında tablo isimleri açıkça belirtilmediyse ingilizce küçük harf çoğul kabulünü hatorlayınız):

```
select * from kitaps

select * from yazars where id in (1, 2, 3, 4, 5, ...)
```

Ateşli yüklemenin akıllıca kullanımı uygulamanızın performansını önemli ölçüde artırabilir.

Tabii ki, bir defada birden çok ilişkiyi ateşli yükleyebilirsiniz:

```
$kitaplar = Kitap::with('yazar', 'kitabevi')->get();
```

Hatta içi içe ilişkileri de ateşleyebilirsiniz:

```
$kitaplar = Kitap::with('yazar.kisiler')->get();
```

Yukarıdaki örnekte `yazar` ilişkisi ateşli yüklenecektir ve yazarın `kisiler` ilişkisi de ateşli yüklenecektir.

Ateşli Yükleme Sınırlamaları

Bazen bir ilişkiyi ateşli yüklemek, ama ateşli yükleme için de bir şart belirlemek isteyebiliriz. İşte bir örnek:

```
$uyeler = Uye::with(array('makaleler' => function($query)
{
        $query->where('baslik', 'like', '%birinci%');
}))->get();
```

Bu örnekte üyenin makalelerinden sadece baslik alaninda "birinci" kelimesi geçen makalelerini ateşli yüklüyoruz.

Tembel Ateşli Yükleme

İlişkili modelleri, direkt olarak önceden mevcut model koleksiyonundan ateşli yüklemek de mümkündür. Bu özellikle ilişkili modeli önbellekleme ile birlikte yükleyip yüklememeye dinamik karar vereceğiniz zaman işe yarayabilir.

```
$kitaplar= Kitap::all();

$kitaplar->load('yazar', 'kitabevi');
```

İlişkili Modelleri Ekleme

Yeni ilişkili model ekleme ihtiyacınız çok olacaktır. Örneğin, bir makale için yeni bir yorum eklemek isteyebilirsiniz. Model üzerinde `makale_id` yabancı key alanını elle ayarlamak yerine, doğrudan ebeveyn `Makale` modelinden yeni yorum ekleyebilirsiniz:

İlişkili Bir Modelin Eklenmesi

```
1  $yorum = new Yorum(array('mesaj' => 'Yeni bir yorum.'));
2
3  $makale = Makale::find(1);
4
5  $yorum = $makale->yorumlar()->save($yorum);
```

Bu örnekte eklenen yorumdaki `makale_id` alanı otomatik olarak ayarlanmaktadır.

İlişkili Model Ekleme (Birçoktan Birçoğa)

Birçoktan birçoğa ilişkilerle çalışırken de ilişkili model ekleyebilirsiniz. Daha önceki örneğimiz `Uye` ve `Rol` modellerini kullanamaya devam edelim. Bir uyeye yeni roller eklemeyi `attach` metodu ile yapabiliriz:

Birçoktan Birçoğa Modellerinin Eklenmesi

```
1  $uye = Uye::find(1);
2
3  $uye->roller()->attach(1);
```

İlişkiler için pivot tabloda tutulan nitelelikleri bir diz olarak da geçebilirsiniz:

```
1  $uye->roller()->attach(1, array('sonaerme' => $sonaerme));
```

Tabii, `attach`'in ters işlemi `detach`'tir:

```
1  $uye->roller()->detach(1);
```

İlişkili modelleri bağlamak için `sync` metodunu da kullanabilirsiniz. Bu `sync` metodu parametre olarak pivot tablodaki yerlerin id'lerinden oluşan bir dizi geçirilmesini ister. Bu işlem tamamlandıktan sonra, model için kullanıcak ara tabloda sadece bu id'ler olacaktır:

Birçoktan Birçoğa Model Bağlamak İçin Sync Kullanımı

```
$uye->roller()->sync(array(1, 2, 3));
```

Belli id değerleri olan başka pivot tabloyu da ilişkilendirebilirsiniz:

Sync Yaparken Pivot Veri Eklenmesi

```
$uye->roller()->sync(array(1 => array('sonaerme' => true)));
```

Bazen yeni bir ilişkili model oluşturmak ve tek bir komutla bunu eklemek isteyebilirsiniz. Bu işlem için, `save` metodunu kullanabilirsiniz:

```
$rol = new Rol(array('isim' => 'Editor'));

Uye::find(1)->roller()->save($rol);
```

Bu örnekte, yeni bir `Rol` modeli kaydedilecek ve uye modeline eklenecektir. Bu işlem için bağlı tablolardaki niteliklerden oluşan bir dizi de geçebilirsiniz:

```
Uye::find(1)->roller()->save($rol, array('sonaerme' => $sonaerme));
```

Ebeveyn Zaman Damgalarına Dokunma

Bir `Yorum`'un bir `Makale`'ye ait olması örneğimizdeki gibi, bir model başka bir modele ait (`belongsTo`) olduğu takdirde, çocuk modeli güncellediğiniz zaman ebeveyn zaman damgasını da güncellemek iyidir. Örneğin, bir `Yorum` güncellendiğinde, bunun sahibi olan `Makale`'nin `updated_at` zaman damgasını otomatikman güncellemek isteyebilirsiniz. Bunu gerçekleştirmek için tek yapacağınız şey, çocuk modele ilişkilerin isimlerini içeren bir `touches` özelliği eklemektir:

```
class Yorum extends Eloquent {

        protected $touches = array('makale');

        public function makale()
        {
                return $this->belongsTo('Makale');
        }

}
```

Bunu yaptıktan sonra artık bir `Yorum` güncellediğinizde, sahibi olan `Makale` de güncellenmiş bir `updated_at` sütununa sahip olacaktır:

```
$yorum = Yorum::find(1);

$yorum->text = 'Bu yorumu düzelt!';

$yorum->save();
```

Pivot Tablolarla Çalışmak

Daha önce öğrendiğiniz gibi, birçoktan birçoğa ilişkilerle çalışmak bir ara tablonun olmasını gerektirir. Eloquent işte bu tablo ile etkileşim için çok yararlı bazı yollar sağlamaktadır. Örneğin bizim bir `Uye` nesnemiz, bir de onun bağlı olduğu birçok `Rol` nesnelerimiz olsun. Bu ilişkiye eriştikten sonra, `pivot` tabloya modellerimiz üzerinden erişebiliriz:

```
$uye = Uye::find(1);

foreach ($uye->roller as $rol)
{
        echo $rol->pivot->created_at;
}
```

Dikkat ederseniz, elde ettiğimiz her bir `Rol` modeline otomatikman bir `pivot` niteliği atanmıştır. Bu nitelik, ara tabloyu temsil eden bir modeli taşır ve herhangi bir Eloquent modeli gibi kullanılabilir.

Ön tanımlı olarak, `pivot` nesnesinde sadece keyler olacaktır. Şayet pivot tablonuzda bunlardan başka nitelikler varsa, bunları ilişki tanımlama sırasında belirtmelisiniz:

```
return $this->belongsToMany('Rol')->withPivot('falan', 'filan');
```

Şimdi `Rol` modelinin `pivot` nesnesinde `falan` ve `filan` nitelikleri erişilebilir olacaktır.

Eğer pivot tablonuzun `created_at` ve `updated_at` zaman damgalarını otomatik olarak halletmesini istiyorsanız, ilişki tanımlamasında `withTimestamps` metodunu kullanın:

```
return $this->belongsToMany('Rol')->withTimestamps();
```

Bir modelin pivot tablosundaki tüm kayıtları silmek için, `detach` metodunu kullanabilirsiniz:

Bir Pivot Tablodaki Tüm Kayıtların Silinmesi

```
Uye::find(1)->roller()->detach();
```

Bu operasyonun `roller` tablosundan kayıt silmediğine, sadece pivot tablodan sildiğine dikkat ediniz.

Koleksiyonlar

Eloquent tarafından döndürülen tüm çoklu sonuç kümeleri ya `get` metodu aracılığıyla döndürülür veya bir ilişki bir Eloquent `Collection` nesnesi döndürür. Bu nesne PHP'nin `IteratorAggregate` arayüzünün bir uygulama biçimidir ve tıpkı bir dizide dolaşır gibi dolaşılabilinmektedir. Bunun yanında, bu nesne sonuç kümeleriyle çalışırken işe yarayan başka bir takım metodlara da sahiptir.

Örneğin biz `contains` metodunu kullanarak bir sonuç kümesinin belli bir primer key içerip içermediğini tespit edebiliriz:

Bir Koleksiyonun Bir Key Taşıyıp Taşımadığının Yoklanması

```
1  $roller = Uye::find(1)->roller;
2
3  if ($roller->contains(2))
4  {
5          //
6  }
```

Koleksiyonlar aynı zamanda bir dizi ya da JSON'a dünüştürülebilmektedir:

```
1  $roller = Uye::find(1)->roller->toArray();
2
3  $roller = Uye::find(1)->roller->toJson();
```

Eğer bir koleksiyon bir string kalıbına çevrilirse JSON olarak döndürülecektir:

```
1  $roller = (string) Uye::find(1)->roller;
```

Eloquent koleksiyonları içerdikleri elemanları dolaşmak ve filtre etmekle ilgili bazı metodlara da sahiptir:

Koleksiyonlarda Tekrarlı İşlemler

```
1  $roller = $uye->roller->each(function($rol)
2  {
3
4  });
```

Koleksiyonlarda Filtreleme

Verilen dönüş (callback) array_filter()[100] için dönüş olarak kullanılacak.

[100] http://php.net/manual/en/function.array-filter.php

```
$uyeler = $uye->filter(function($uyeler)
{
        if($uye->isAdmin())
        {
                return $uye;
        }
});
```

Her Bir Koleksiyon Nesnesine Bir Dönüş (Callback) Yapmak

```
$roller = Uye::find(1)->roller;

$roller->each(function($rol)
{
        //
});
```

Bir Koleksiyonu Bir Değere Göre Sıralama

```
$roller = $roller->sortBy(function($rol)
{
        return $rol->created_at;
});
```

Bazen de, kendi eklediğiniz metodları olan özel bir koleksiyon nesnesi döndürmek isteyebilirsiniz. Bunu, Eloquent modeliniz üzerinde `newCollection` metodunu ezerek yapabilirsiniz:

Özel Bir Koleksiyon Tipinin Döndürülmesi

```
class Uye extends Eloquent {

        public function newCollection(array $models = array())
        {
                return new CustomCollection($models);
        }

}
```

Erişimciler & Değiştiriciler (Accessors & Mutators)

Eloquent model niteliklerini alıp getirirken veya onları ayarlarken dönüşüm yapmak için uygun bir yol sağlar. Bir erişimci beyan etmek için modeliniz üzerinde sadece bir `getFilanAttribute` metodu tanımlamak yeterlidir. Yalnız unutmamanız gereken şey, veritabanı sütunlarınızın isimleri yılan tarzı (küçük harfli kelimelerin boşluk olmaksızın alt tire ile birbirine bağlanması) olsa dahi, metodlarınızın deve tarzı (birinci kelimenin tümü küçük harf olmak ve sonraki kelimelerin ilk harfi büyük diğer hafleri küçük olmak üzere boşluk olmaksızın kelimelerin yanyana dizilmesi) olması gerektiğidir:

Bir Erişimci Tanımlanması

```
1  class Uye extends Eloquent {
2
3          public function getSoyAdiAttribute($value)
4          {
5                  return ucfirst($value);
6          }
7
8  }
```

Yukarıdaki örnekte `soy_adi` sütununun bir erişimcisi vardır. Niteliğin değerinin erişimciye geçildiğine dikkat ediniz.

Değiştiriciler de benzer şekilde deklare edilir:

Bir Değiştirici Tanımlanması

```
1  class Uye extends Eloquent {
2
3          public function setSoyAdiAttribute($value)
4          {
5                  $this->attributes['soy_adi'] = strtolower($value);
6          }
7
8  }
```

Tarih Değiştiricileri

Ön tanımlı olarak, Eloquent will convert the `created_at`, `updated_at`, and `deleted_at` columns to instances of Carbon[101], olgularına çevirecektir. Carbon çeşitli yardımcı metodlar sağlar ve PHP'nin `DateTime` sınıfını genişletir.

[101] https://github.com/briannesbitt/Carbon

Siz hangi alanların otomatik olarak değiştirileceğini isteğinize göre ayarlayabilirsiniz, hatta modeldeki `getDates` metodunu ezmek suretiyle bu motasyonu tamamen devre dışı bırakabilirsiniz:

```
public function getDates()
{
        return array('created_at');
}
```

Bir sütun bir tarih olarak kabul edildiğinde, bunun değerini bir UNIX timetamp, date string (`Y-m-d`), date-time string ve tabii ki bir `DateTime` / `Carbon` olgusuna ayarlayabilirsiniz.

Tarih değiştiricilerini tümden devre dışı bırakmak için `getDates` metodunda boş bir dizi döndürünüz:

```
public function getDates()
{
        return array();
}
```

Model Olayları

Eloquent modelleri bazı olayları tetikleyerek, modelin yaşam döngüsündeki çeşitli noktalarda müdahale etmenize imkan verir. Bu amaçla şu metodlar kullanılmaktadır: `creating`, `created`, `updating`, `updated`, `saving`, `saved`, `deleting`, `deleted`. Eğer `creating`, `updating` veya `saving` olaylarından `false` döndürülürse, eylem iptal edilecektir:

Saklama Operasyonlarının Olaylar Aracığıyla İptal Edilmesi

```
Uye::creating(function($uye)
{
        if ( ! $uye->isValid()) return false;
});
```

Eloquent modelleri bunun dışında static bir `boot` metodu içermekte olup, olay bağlamanızı kayıt etmeniz için uygun bir yerdir.

Bir Model Boot Metodunun Ayarlanması

```
class Uye extends Eloquent {

        public static function boot()
        {
                parent::boot();

                // Olay bağlamayı ayarla...
        }

}
```

Model Gözlemcileri

Model olaylarının işlenmesini pekiştirmek için, bir model gözlemcisi kaydı yapabilirsiniz. Bir gözlemci sınıfında çeşitli model olaylarına tekabül eden metodlar bulunabilir. Örneğin bir gözlemcide, diğer model olay isimlerine ek olarak `creating`, `updating`, `saving` metodları olabilir.

Yani, bir model gözlemcisi şöyle olabilir:

```
class UyeGozlemcisi {

        public function saving($model)
        {
                //
        }

        public function saved($model)
        {
                //
        }

}
```

Modelinizde `observe` metodunu kullanarak bir gözlemci olgusu kaydı yapabilirsiniz:

```
Uye::observe(new UyeGozlemcisi);
```

Diziye / JSON'a Çevirme

JSON APIler oluşturulurken, çoğu defa modellerinizi ve ilişkilerini dizilere veya JSON'a çevirmeniz gerekecektir. Bu yüzden Eloquent bunları yapacak metodlar içermektedir. Bir modeli ve onun yüklenen ilişkilerini bir diziye çevirmek için `toArray` metodunu kullanabilirsiniz:

Bir Modelin Bir Diziye Çevrilmesi

```
$uye = Uye::with('roller')->first();

return $uye->toArray();
```

Modellerin koleksiyonlarının da bütün olarak dizilere dönüştürülebildiğini unutmayın:

```
return Uye::all()->toArray();
```

Bir Modeli JSON'a çevirmek için, `toJson` metodunu kullanabilirsiniz:

Bir Modelin JSON'a Çevrilmesi

```
return Uye::find(1)->toJson();
```

Bir model veya koleksiyon bir string kalıbına sokulduğu takdirde, JSON'a çevrileceğine dikkat ediniz. Yani Elequent nesnelerini direkt olarak uygulamanızın rotalarından döndürebilirsiniz!

Bir Modelin Bir Rotadan Döndürülmesi

```
Route::get('uyeler', function()
{
        return Uye:all();
});
```

Bazen bazı nitelikleri (örneğin şifreleri) modelinizin dizi veya JSON biçimlerinden hariç tutmak isteyebilirsiniz. Bunu yapmak için modelinize bir `hidden` özelliği ekleyiniz:

Niteliklerin Dizi veya JSON'a Çevrilmekten Saklanması

```
class Uye extends Eloquent {

        protected $hidden = array('parola');

}
```

Alternatif olarak, beyaz bir liste tanımlamak için `visible` özelliğini kullanabilirsiniz:

```
protected $visible = array('adi', 'soy_adi');
```

Kimi durumlarda, veritabanınızdaki bir sütuna tekabül etmeyen dizi nitelikleri eklemeniz gerekebilir. Bunu yapmak için, değer için bir erişimci tanımlamanız yeterlidir:

```php
public function getIsAdminAttribute()
{
        return $this->attributes['admin'] == 'yes';
}
```

Erişimciyi oluşturduktan sonra, ilgili modeldeki `appends` özelliğine değeri ekleyin:

```php
protected $appends = array('is_admin');
```

Nitelik `appends` listesine eklendikten sonra modelin hem dizi hem de JSON formlarına dahil edilecektir.

Şema Oluşturucusu

Giriş

Laravel'in `Schema` sınıfı tablolara müdahale etmekte veritabanı bilinmesine gerek kalmaz bir yol sağlar. Laravel'in desteklediği tüm veritabanlarıyla sağlıklı çalışır ve bu sistemlerin tümünde aynı olan bir API'ye sahiptir.

Tabloların Oluşturulması ve Yok Edilmesi

Yeni bir veritabanı tablosu oluşturmak için `Schema::create` metodu kullanılır:

```
Schema::create('uyeler', function($table)
{
        $table->increments('id');
});
```

Bu `create` metoduna geçilen ilk parametre tablonun adıdır ve ikincisi bu yeni tabloyu tanımlamakta kullanılabilecek bir proje (`Blueprint`) nesnesi alacak bir bitirme (`Closure`) fonksiyonudur.

Mevcut bir veritabanı tablosunun adını değiştirmek için `rename` metodu kullanılabilir:

```
Schema::rename($eskisinden, $yeniye);
```

Şema operasyonunun gerçekleştirileceği bağlantıyı belirlemek için `Schema::connection` metodunu kullanınız:

```
Schema::connection('falan')->create('uyeler', function($table)
{
        $table->increments('id');
});
```

Bir tabloyu yok etmek için, `Schema::drop` metodunu kullanabilirsiniz:

```
Schema::drop('uyeler');

Schema::dropIfExists('uyeler');
```

Sütunların Eklenmesi

Mevcut bir tabloda sütun ekleme için `Schema::table` metodunu kullanıyoruz:

```
Schema::table('uyeler', function($table)
{
        $table->string('email');
});
```

Tablo oluşturma zamanında ise tablo oluşturucusunda bulunan çeşitli sütun tiplerini kullanabilirsiniz:

```
Komut          | Açıklama
-------------- | -------------
`$table->increments('id');`  |  Giderek artan ID alanı ekler (esas key).
`$table->bigIncrements('id');`  |  "big integer" eşdeğeri.
`$table->string('email');`  |  VARCHAR eşdeğeri sütun
`$table->string('isim', 100);`  |  belli uzunlukta VARCHAR eşdeğeri sütun
`$table->integer('puan');`  |  INTEGER eşdeğeri sütun
`$table->bigInteger('puan');`  |  BIGINT eşdeğeri sütun
`$table->smallInteger('puan');`  |  SMALLINT eşdeğeri sütun
`$table->float('miktar');`  |  FLOAT eşdeğeri sütun
`$table->decimal('miktar', 5, 2);`  |  basamak ve ondalık basamak
                                                sayısı belirlenmiş DECIMAL eşdeğeri sütun
`$table->boolean('teyit');`  |  BOOLEAN eşdeğeri sütun
`$table->date('created_at');`  |  DATE eşdeğeri sütun
`$table->dateTime('created_at');`  |  DATETIME eşdeğeri sütun
`$table->time('ikindi');`  |  TIME eşdeğeri sütun
`$table->timestamp('eklenme_vakti');`  |  TIMESTAMP eşdeğeri sütun
`$table->timestamps();`  |  **created\_at** ve **updated\_at** sütunlarını ekler
`$table->softDeletes();`  |  Belirsiz silmeler için **deleted\_at** sütunu ekler
`$table->text('izahat');`  |  TEXT eşdeğeri sütun
`$table->binary('veri');`  |  BLOB eşdeğeri sütun
`$table->enum('tercihler', array('falan', 'filan'));` | ENUM eşdeğeri sütun
`->nullable()`  |  İlgili sütunun NULL değerleri olabilir demektir
`->default($deger)`  |  Bir sütun için ön tanımlı bir değer tanımlar
`->unsigned()`  |  INTEGER'i UNSIGNED olarak ayarlar
```

Şayet MySQL veritabanı kullanıyorsanız, sütunların sıralamasını belirlemek için `after` metodunu kullanabilirsiniz:

MySQL Veritabanında After Kullanımı

```
$table->string('isim')->after('email');
```

Sütün İsimlerinin Değiştirilmesi

Bir sütun ismini değiştirmek için Şema Oluşturucusunda `renameColumn` metodunu kullanabilirsiniz:

Bir Sütun İsminin Değiştirilmesi

```
Schema::table('uyeler', function($table)
{
        $table->renameColumn('eskiden', 'yeniye');
});
```

Not: `enum` sütun tipleri için isim değiştirme desteklenmemektedir.

Sütunların Yok Edilmesi

Bir Veritabanı Tablosundan Bir Sütunun Yok Edilmesi

```
Schema::table('uyeler', function($table)
{
        $table->dropColumn('puan');
});
```

Bir Veritabanı Tablosundan Birden Çok Sütunun Yok Edilmesi

```
Schema::table('uyeler', function($table)
{
        $table->dropColumn('puan', 'avatar', 'ikametgah');
});
```

Mevcutluk Yoklanması

`hasTable` ve `hasColumn` metodlarını kullanarak bir tablo ya da sütunun var olup olmadığını kolayca yoklayabilirsiniz:

Tablonun Var Olduğunun Yoklanması

```
if (Schema::hasTable('uyeler'))
{
        //
}
```

Sütunların Var Olduğunun Yoklanması

```
if (Schema::hasColumn('uyeler', 'email'))
{
        //
}
```

İndeks Eklenmesi

Şema oluşturucusu çeşitli indeks tiplerini desteklemektedir. Bunları iki şekilde ekleyebilirsiniz. Birinci yol bir sütun tanımı sırasında tanımlamak, ikinci yol ise ayrıca eklemektir:

Bir Sütun ve İndeksin Birlikte Oluşturulması

```
$table->string('email')->unique();
```

Ya da, ayrı satırlarda indeks ekleme yolunu seçebilirsiniz. Aşağıda, kullanılabilecek tüm indeks tiplerinin bir listesi verilmiştir:

```
Komut          | Açıklama
-------------- | -------------
`$table->primary('id');`  |  Bir esas key eklenmesi
`$table->primary(array('ilk', 'son'));`  |  Bileşik keylerin eklenmesi
`$table->unique('email');`  |  Benzersiz bir indeks eklenmesi
`$table->index('il');`  |  Basit bir indeks eklenmesi
```

Yabancı Key

Laravel, tablolarınıza yabancı key sınırlaması eklemeniz için de destek verir:

Bir Tabloya Bir Yabcancı Key Eklenmesi

```
$table->foreign('uye_id')->references('id')->on('uyeler');
```

Bu örnekte, `uye_id` sütununun `uyeler` tablosundaki `id` sütununu referans aldığını beyan ediyoruz.

Ayrıca, güncelleme ve silme (“on delete” ve “on update”) eylemi sınırlamaları için seçenekler de belirleyebilirsiniz:

```
$table->foreign('uye_id')
      ->references('id')->on('uyeler')
      ->onDelete('cascade');
```

Bir yabancı keyi yok etmek için, `dropForeign` metodunu kullanabilirsiniz. Yabancı key için de diğer indeksler için kullanılan isimlendirme geleneği kullanılır:

```
$table->dropForeign('makaleler_uye_id_foreign');
```

> **Not**: Otomatik artan bir tam sayıya başvuran bir foreign key oluşturulurken, foreign key sütununu her zaman için `unsigned` yapmayı unutmayın.

İndekslerin Yok Edilmesi

Bir indeksi yok etmek için indeksin adını belirtmelisiniz. Laravel, ön tanımlı olarak indekslere makul bir isim tahsis eder. Tablo adını, indekslenen alan adlarını ve indeks tipini art arda ekler. İşte bazı örnekler:

```
Komut          | Açıklama
-------------- | --------------
`$table->dropPrimary('uyeler_id_primary');`  |  "uyeler" tablosundan primer key'i\
n yok edilmesi
`$table->dropUnique('uyeler_email_unique');`  |  "uyeler" tablosundan benzersiz b\
ir indeksin yok edilmesi
`$table->dropIndex('geo_il_index');`  |  "geo" tablosundan basit bir indeksin yok\
 edilmesi
```

Depolama Motorları

Bir tablo için depolama motoru ayarlamak için, şema oluşturucusunda `engine` özelliğini ayarlayınız:

```
Schema::create('uyeler', function($table)
{
    $table->engine = 'InnoDB';

    $table->string('email');
});
```

Yerleşimler (Migrations) ve Filizlendirme (Seeding)

Giriş

Yerleşimler veritabanı için bir sürüm kontrol türüdür. Bir ekibin veritabanı şemasını değiştirmesine ve son şema durumuna güncel kalmalarına imkan verir. Yerleşimler, uygulama şemasını kolayca yönetmek amacıyla tipik olarak Şema (Schema) Kurucu[102] ile eşleştirilirler.

Yerleşimlerin Oluşturulması

Bir yerleşim oluşturmak için, Artisan KSA 'da (Artisan Komut Satırı Arayüzü) `migrate:make` komutunu kullanabilirsiniz:

Bir Yerleşim Oluşturulması

```
php artisan migrate:make kullanicilar_tablosunu_olustur
```

Yerleşim `app/database/migrations` dizininize konumlandırılır ve Laravel'in yerleşimlerin sırasını belirlemesine imkan veren bir zaman damgası içerir.

Yerleşimi oluştururken bir patika `--path` seçeneği de belirtebilirsiniz. Patika, kurulum kök dizinine göreceli olmalıdır:

```
php artisan migrate:make falancaYerlesim --path=app/migrations
```

Tablo ismini ve yeni bir tablonun oluşturulacagını da, tablo `--table` ve oluştur `--create` seçeneklerini kullanarak belirtebilirsiniz:

```
php artisan migrate:make kullanicilar_tablosunu_olustur --table=kullanicilar --cr\
eate
```

Yerleşimlerin Çalıştırılması

Bekleyen Yerleşimlerin Hepsinin Birden Çalıştırılması

[102] /docs/schema

```
php artisan migrate
```

Bir Patikadaki Yerleşimlerin Çalıştırılması

```
php artisan migrate --path=app/falancaDizin/migrations
```

Bir Paketin Tüm Bekleyen Yerleşimlerinin Çalıştırılması

```
php artisan migrate --package=vendor/package
```

> **Not**: Yerleşimleri çalıştırırken, "class not found" (sınıf bulunamadı) hatası veririse, `composer update` (composer güncelle) komutunu çalıştırarak deneyiniz.

Yerleşimlerin Geriye Döndürülmesi

Son Yerleşim İşleminin Geriye Döndürülmesi

```
php artisan migrate:rollback
```

Tüm Yerleşim İşlemlerinin Geriye Döndürülmesi

```
php artisan migrate:reset
```

Tüm Yerleşim İşlemlerinin Geriye Döndürülmesi ve Hepsinin Tekrardan Çalıştırılması

```
php artisan migrate:refresh                //filizlendirmeler dahil edilmeden

php artisan migrate:refresh --seed        //filizlendirmeler dahil edilerek
```

Veritabanı Filizlendirmesi

Filizlendirme (seeding), yerleşim ile oluşturulacak veritabanı tablosunda gerekli olacak ilk veri kayıtlarının (seed data) oluşturulması işlemidir(:çevirenin notu). Laravel, veritabanınızın deneme verisi ile filizlendirilmesi için kolaylık sağlayacak olan filizlendirme (seed) sınıflarını bulundurur. Bütün filizlendirme sınıfları `app/database/seeds` dizininde konumlandırılır. Filizlendirme sınıflarına istediğiniz isimleri verebilirsiniz. Fakat isimlendirirken anlaşılacak belli bir düzene (convention) uyulması lehinizedir, örneğin `KullanicilarTablosuFilizlendiricisi`, vb. Ön tanımlı olarak, sizin için bir DatabaseSeeder sınıfı tanımlanmıştır. Filizlendirme sırasını denetlemenize imkan verecek olan, bu sınıfın 'çağır' `call` metodunu kullanarak diğer filizlendirme sınıflarınızı çalıştırabilirsiniz.

Veritabanı Filizlendirme Sınıfı Örneği

```
class DatabaseSeeder extends Seeder {

        public function run()
        {
                $this->call('KullanicilarTablosuFilizlendiricisi');

                $this->command->info('Kullanıcı tablosu filizlendirildi!');
        }

}

class KullanicilarTablosuFilizlendiricisi extends Seeder {

        public function run()
        {
                DB::table('kullanicilar')->delete();

                User::create(array('email' => 'falanca@filanca.com'));
        }

}
```

Veritabanınızı filizlendirmek için, Artisan KSA'da `db:seed` (filizlendir) komutunu kullanabilirsiniz:

```
php artisan db:seed
```

Veritabanınızı `migrate:refresh` (yenile) komutunu kullanarak da filizlendirebilirsiniz, bu komut aynı zamanda bütün yerleşimleri geriye döndürüp, hepsini tekrardan çalıştıracaktır:

```
php artisan migrate:refresh --seed
```

Redis

Giriş

Redis[103] açık kaynak, gelişmiş bir anahtar-değer deposudur. Anahtarlar stringler[104], hashler[105], listeler[106], kümeler[107] ve sıralı kümeler[108] taşıyabildikleri için sıklıkla bir veri yapısı sunucusu olarak da ifade edilmektedir.

> **Not**: Eğer PECL aracılığıyla yüklenmiş Redis PHP eklentiniz varsa, `app/config/app.php` dosyanızda Redis için kullanılan lakabın ismini değiştirmeniz gereklidir.

Yapılandırma

Uygulamanızdaki Redis yapılandırması **app/config/database.php** dosyasında saklanır. Bu dosya içerisinde, uygulamanız tarafından kullanılan Redis sunucularını içeren bir **redis** dizisi göreceksiniz:

```
'redis' => array(

        'cluster' => true,

        'default' => array('host' => '127.0.0.1', 'port' => 6379),

),
```

Geliştirme için bu "default" sunucu yapılandırması yeterlidir. Yine de siz ortamınıza göre bu diziyi değiştirmekte serbestsiniz. Sadece her Redis sunucusuna bir ad verin ve bu sunucu tarafından kullanılan ana bilgisayarı (host) ve bağlantı noktasını (port) belirtin.

Buradaki `cluster` seçeneği Laravel Redis istemcisine Redis düğümleriniz arasında istemci taraflı bölümlendirme (sharding) yapmasını söylemektedir. Böylece siz düğüm havuzu ve büyük miktarda kullanılabilir RAM oluşturabilirsiniz. Bununla birlikte istemci taraflı bölümlendirmenin başarısızlık

[103] http://redis.io
[104] http://redis.io/topics/data-types#strings
[105] http://redis.io/topics/data-types#hashes
[106] http://redis.io/topics/data-types#lists
[107] http://redis.io/topics/data-types#sets
[108] http://redis.io/topics/data-types#sorted-sets

durumlarını halledemediğini unutmayın. Bu nedenle, istemci taraflı bölümlendirme, esasında başka bir asıl veri deposunda olup da önbelleğe alınmış veriler için uygundurlar.

Şayet sizin Redis serveriniz authentication istiyorsa, Redis server yapılandırma dizinize bir `password` anahtar / değer çifti eklemek suretiyle bir şifre sağlayabilirsiniz.

Kullanım

Bir Redis olgusunu `Redis::connection` metodunu çağırarak getirebilirsiniz:

```
1  $redis = Redis::connection();
```

Bu size "default" Redis sunucusunun bir olgusunu verecektir. Eğer sunucu öbekleme (clustering) kullanmıyorsanız, Redis yapılandırmanızda tanımlanan belirli bir sunucuyu getirmek için `connection` metodunda parametre olarak o sunucunun adını geçersiniz:

```
1  $redis = Redis::connection('digerbirsunucu');
```

Redis istemci olgusu oluşturduktan sonra, artık bu olguya her türlü Redis komutu[109] verebiliriz. Laravel Redis sunucusuna komut geçerken sihirli metodlar tekniğini kullanır:

```
1  $redis->set('isim', 'Taylor');
2
3  $isim = $redis->get('isim');
4
5  $degerler = $redis->lrange('isimler', 5, 10);
```

Görüldüğü gibi komut parametreleri basitçe sihirli metodlara geçilmektedir. Tabii ki siz sihirli metod tekniğini kullanmak zorunda değilsiniz, `command` metodunu kullanarak da sunucuya komut geçebilirsiniz:

```
1  $degerler = $redis->command('lrange', array(5, 10));
```

Komutlarınızı sadece "default" bağlantıda çalıştıracağınız zaman, direkt `Redis` sınıfındaki statik sihirli metodları kullanın:

[109] http://redis.io/commands

```
Redis::set('isim', 'Taylor');

$isim = Redis::get('isim');

$degerler = Redis::lrange('isimler', 5, 10);
```

Not: Redis Önbellekleme[110] ve Oturum[111] sürücüleri Laravel'de mevcuttur.

Pipeline Kullanma

Bir operasyonda sunucuya birçok komut göndermeniz gerektiğinde pipeline kullanılmalıdır. Bunu yapmak için `pipeline` komutunu kullanın:

Sunucularınıza Birden Çok Komutun Döşenmesi

```
Redis::pipeline(function($pipe)
{
        for ($i = 0; $i < 1000; $i++)
        {
                $pipe->set("key:$i", $i);
        }
});
```

[110] /docs/cache
[111] /docs/session

Artisan CLI

Giriş

Artisan, Laravel içerisinde gelen CLI'ın (Command-line Interface) adıdır. Artisan size uygulamanızı geliştirirken birçok yardımcı komut sağlar. Artisan, güçlü Symfony Console bileşeni üzerinden geliştirilmiştir.

Kullanım

Tüm Artisan komutlarını görmek için `list` komutunu kullanabilirsiniz:

Tüm Kullanılabilir Komutları Listelemek

```
php artisan list
```

Tüm komutların özel bir "yardım" ekranı vardır ve komut hakkındaki argüman sırası ile ayarlar gibi bilgilerin açıklanmasını sağlar. Bir yardım ekranını görüntülemek için komut adından önce `help` yazın:

Bir Komut için Yardım Ekranını Görmek

```
php artisan help migrate
```

Bir komut kullanırken kullanılacak olan Ortam Ayarları'nı `--env` komutuyla belirleyebilirsiniz:

Ortam Ayarlarını Belirlemek

```
php artisan migrate --env=local
```

Ayrıca şu an kullanmakta olduğunuz Laravel'in sürümünü de `--version` seçeneğini kullanarak Artisan üzerinden görebilirsiniz:

Laravel'in Sürümünü Görmek

```
php artisan --version
```

Artisan'ın Geliştirilmesi

Giriş

Artisan'da mevcut olan komutlara ilaveten, uygulamanız ile çalışacak olan kendi özel komutlarınızı inşa edebilirsiniz. Bu özel komutlarınızı `app/commands` dizininde depolayabilirsiniz. Komutlarınızı kendi istediğiniz başka bir dizinde de depolayabilirsiniz. Bunun için, bu komutlarınızın `composer.json` ayarlarınız bazında "autoload" edilebiliyor olması gerekmektedir.

Komut Oluşturulması

Sınıfının Oluşturulmması

Yeni bir komut oluşturmak için, `command:make` Artisan komutunu kullanabilirsiniz. Bu komut, başlamanızda size yardımcı olmak için yeni bir komut taslağı oluşturacaktır.

Yeni bir Komut Sınıfının Oluşturulması

```
1  php artisan command:make FalancaKomut
```

Ön tanımlı olarak, oluşturulan komutlar `app/commands` dizininde depolanırlar. Fakat, siz başka bir dizin veya bir 'namespace' de belirleyebilirsiniz.

```
1  php artisan command:make FalancaKomut --path=app/classes --namespace=Siniflar
```

Komutun Yazılışı

Komut oluşturulduktan sonra, komutu `list` ekranında görüntülerken kullanılacak olan, sınıf ismi `name` ve tanımı `description` özellikleri doldurulmalıdır.

Komut çalıştırıldığında `fire` (ateşle) metodu çağırılmaktadır. Bu metoda, istenecek herhangi bir komut mantığı yerleştirebilinir.

Argümanlar & Seçenekler

Komutunuzun alacağı argüman veya seçenekleri tanımlayabileceğiniz yerler `getArguments` ve `getOptions` metodlarıdır. Bu metodların her ikisi de birer komut dizisi verirler. Bu komut dizileri, bir 'dizi seçenekleri listesi' ile tarif edilirler.

Argümanları `arguments` belirlerlerken, dizi tanımı değerleri şunları belirler: (ismi, modu, tanımı, ön değeri)

```
array($name, $mode, $description, $defaultValue)
```

Modu argümanı `mode` şunlardan herhangi biri olabilir: `InputArgument::REQUIRED` (mecburi) veya `InputArgument::OPTIONAL` (isteğe bağlı).

Seçenekleri `options` belirlerken, dizi tanımı değerleri şunları belirler: (ismi, kısayolu, modu, tanımı, ön değeri)

```
array($name, $shortcut, $mode, $description, $defaultValue)
```

Seçenekler için, modu argümanı `mode` şunlardan biri olabilir: `InputOption::VALUE_REQUIRED` (Girdi Seçeneği: mecburi), `InputOption::VALUE_OPTIONAL` (isteğe bağlı), `InputOption::VALUE_IS_ARRAY` (dizi), `InputOption::VALUE_NONE` (yok).

`VALUE_IS_ARRAY` (Girdi Seçeneği: dizi) modu, komut çağırılırken, anahtarın birden çok kez kullanılabilir olduğunu belirtir:

```
php artisan falan --option=filan --option=fesmekan
```

`VALUE_NONE` (Girdi Seçeneği: yok) modu, seçeneğin sadece bir "anahtar" olarak kullanıldığını belirtir.

```
php artisan falan --option
```

Girdilerin Çağırılması

Komutunuz çalışırken, uygulamanızın kabul edeceği argüman ve seçenek değerlerine ulaşabilmeniz gerekecektir. Bunu yapabilmek için argüman `argument` ve seçenek `option` metodlarını kullanabilirsiniz:

Bir Komut Argüman Değerinin Çağırılması

```
$value = $this->argument('ismi');
```

Tüm Argümanların Birden Çağırılması

```
$arguments = $this->argument();
```

Bir Komut Seçeneği Değerinin Çağırılması

```
$value = $this->option('ismi');
```

Tüm Seçeneklerin Birden Çağırılması

```
$options = $this->option();
```

Çıktı Yazılışı

Çıktının konsola gönderilmesi için, `info` (bilgi), `comment` (not), `question` (soru) ve `error` (hata) metodlarını kullanabilirsiniz. Bu metodların her biri, kendi amaçlarına uygun olan ANSI renklerini kullanacaktır.

Konsola Bilgi Gönderilmesi

```
$this->info('Bunu ekranda göster');
```

Konsola Bir Hata Mesajı Gönderilmesi

```
$this->error('Bir hata oluştu!');
```

Soruların Soruluşu

Kullanıcıdan bir girdi talep etmek için, `ask` (sor) ve `confirm` (onayla) metodlarını kullanabilirsiniz.

Kullanıcıya Girdi Bilgisinin Soruluşu

```
$name = $this->ask('İsminiz nedir?');
```

Kullanıcıya Gizli Şifre Bilgisinin Soruluşu

```
$password = $this->secret('Lütfen şifrenizi giriniz!');
```

Kullanıcıya Onayının Soruluşu

```
if ($this->confirm('Devam etmek istiyor musunuz? [evet|hayır]'))
{
        //
}
```

İsterseniz `confirm` (onayla) metoduna, `true` (evet) ve `false` (hayır) seçeneklerinden birini varsayılan ön değer olarak belirleyebilirsiniz :

```
$this->confirm($soru, true);
```

Komutların Kayıt Ettirilmesi

Komutunuzun inşa edilmesi tamamlandığında, kullanılmaya hazır olabilmesi için, Artisan'da kayıt ettirmeniz gerekir. Bu, genelde `app/start/artisan.php` dosyası içerisinde yapılır. Bu dosya içerisinde, kayıt ettirmek için `Artisan::add` (Artisan::ekle) metodunu kullanabilirsiniz.

Bir Artisan Komutunun Kayıt Ettirilişi

```
Artisan::add(new FalancaKomut);
```

Eğer komutunuz IoC container[112] uygulamasında kayıtlı ise, Artisan'da da kullanılabilir olması için `Artisan::resolve` metodunu kullanabilirsiniz.

IoC Container'da Olan Bir Komutun Kayıt Ettirilişi

```
Artisan::resolve('binding.ismi');
```

Diğer Komutların Çağırılması

Bazı durumlarda, komtunuzun içerisinden diğer başka bir komutu çağırmak isteyebilirsiniz. Bunu, `call` (çağır) metodunu kullanarak yapabilirsiniz:

Başka Bir Komutun Çağırılışı

```
$this->call('command.ismi', array('argument' => 'falan', '--option' => 'filan'));
```

[112] /docs/ioc