EBook Gratuito

APPRENDIMENTO Laravel

Free unaffiliated eBook created from
Stack Overflow contributors.

#laravel

Sommario

Di

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: laravel

It is an unofficial and free Laravel ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official Laravel.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Capitolo 1: Iniziare con Laravel

Osservazioni

Laravel StackOverflow Slack Community

Prossimamente

Esercitazione in primo piano

Iniziare con Laravel

Linee guida di contributo

Prossimamente

Guida allo stile di contribuzione

Prossimamente

A proposito di Laravel

Creato da Taylor Otwell come un framework web PHP open-source gratuito, Laravel ha lo scopo di facilitare e accelerare il processo di sviluppo delle applicazioni web con un grande gusto per la semplicità.

Segue lo schema di modello model-view-controller (MVC) nonché lo standard di codifica PSR-2 e lo standard di autoloading PSR-4 .

Esecuzione di un Test Driven Development (TDD) in Laravel è divertente e facile da implementare.

Ospitato su GitHub e disponibile su https://github.com/laravel/laravel , Laravel vanta un'architettura di micro-servizi , che lo rende straordinariamente estensibile e questo, con facilità, con l'uso di terze parti su misura o esistenti pacchi.

Caratteristiche principali

MVC

Laravel utilizza il modello MVC, quindi ci sono tre parti principali del framework che funzionano insieme: modelli, viste e controller. I controller sono la parte principale in cui viene eseguita la maggior parte del lavoro. Si collegano ai modelli per ottenere, creare o aggiornare dati e visualizzare i risultati sulle viste, che contengono la struttura HTML effettiva dell'applicazione.

Blade Templating Engine

Laravel viene fornito con un motore di template conosciuto come Blade. Blade è abbastanza facile da usare, ma potente. Una caratteristica che il motore di template Blade non condivide con altri popolari è la sua permissività; consentendo l'uso di codice PHP semplice in Blade che modella i file del motore.

È importante notare che i file del motore di template di Blade hanno `.blade` aggiunto ai nomi dei file subito prima del solito `.php` che non è altro che l'effettiva estensione del file. In quanto tale, `.blade.php` è l'estensione del file risultante per i file modello Blade. I file del motore modello Blade sono memorizzati nella directory risorse / viste.

Routing e middleware

È possibile definire gli URL della propria applicazione con l'aiuto di percorsi. Questi percorsi possono contenere dati variabili, connettersi a controllori o possono essere integrati in middleware. Middelware è un meccanismo per filtrare le richieste HTTP. Possono essere utilizzati per interagire con le richieste prima che raggiungano i controller e possono quindi modificare o rifiutare le richieste.

Artigiano

Artisan è lo strumento da riga di comando che puoi usare per controllare parti di Laravel. Sono disponibili molti comandi per creare modelli, controller e altre risorse necessarie per lo sviluppo. Puoi anche scrivere i tuoi comandi per estendere lo strumento da riga di comando Artisan.

ORM eloquente

Per collegare i tuoi modelli a vari tipi di database, Laravel offre il proprio ORM con un ampio set di funzioni con cui lavorare. Il framework fornisce anche migrazione e seeding e presenta anche rollback.

Gestione degli eventi

Il framework è in grado di gestire eventi attraverso l'applicazione. È possibile creare listener di eventi e gestori di eventi simili a quelli di NodeJs.

Versioni

Versione	Data di rilascio
1.0	2011-06-09

Versione	Data di rilascio
2.0	2011-11-24
3.0	2012-02-22
3.1	2012-03-27
3.2	2012-05-22
4.0	2013/05/28
4.1	2013/12/12
4.2	2014/06/01
5.0	2015/02/04
5.1 (LTS)	2015/06/09
5.2	2015/12/21
5.3	2016/08/24
5.4	2017/01/24

Examples

Benvenuto nella documentazione dei tag di Laravel!

Laravel è un noto framework PHP. Qui imparerai tutto su Laravel. A partire dal *semplice-come* sapere quale è la programmazione orientata agli oggetti, all'argomento di sviluppo del pacchetto Laravel avanzato.

Questo, come ogni altro tag di documentazione StackOverflow, è una documentazione basata sulla comunità, quindi se hai già esperienze su Laravel, condividi le tue conoscenze aggiungendo i tuoi argomenti o esempi! Non dimenticare di consultare la nostra **guida allo stile di contribuzione** su questo argomento per sapere di più su come contribuire e la guida di stile che abbiamo fatto per assicurarci di dare la migliore esperienza alle persone che vogliono saperne di più su Laravel.

Inoltre, siamo molto contenti che tu venga, spero di vederti spesso qui!

Guida introduttiva

La guida introduttiva è una navigazione personalizzata che abbiamo ordinato da noi stessi per semplificare la navigazione degli argomenti, soprattutto per i principianti. Questa navigazione è ordinata per livello di difficoltà.

Iniziare

Installazione

Viste di Laravel

Lama: introduzione

Lama: variabili e strutture di controllo

O

Installazione da qui

1. Ottieni il compositore da qui e installalo
2. Ottieni Wamp da qui , installalo e imposta la variabile di ambiente di PHP
3. Ottieni il percorso per `www` e digita command:

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

Per installare una versione di Laravel specifica, ottenere il percorso su `www` e digitare command:

```
composer create-project --prefer-dist laravel/laravel=DESIRED_VERSION projectname
```

O

Via Laravel Installer

Per prima cosa, scarica il programma di installazione di Laravel usando Composer:

```
composer global require "laravel/installer"
```

Assicurati di posizionare la `$HOME/.composer/vendor/bin` (o la directory equivalente per il tuo sistema operativo) nel tuo $ PATH in modo che l'eseguibile di `laravel` possa essere localizzato dal tuo sistema.

Una volta installato, il comando `laravel new` creerà una nuova installazione di Laravel nella directory specificata. Ad esempio, il `laravel new blog` creerà una directory denominata `blog` contenente una nuova installazione di Laravel con tutte le dipendenze di Laravel già installate:

```
laravel new blog
```

Leggi Iniziare con Laravel online: https://riptutorial.com/it/laravel/topic/794/iniziare-con-laravel

Capitolo 2: analisi

Examples

introduzione

Scrivere un codice verificabile è una parte importante della costruzione di un progetto robusto, manutenibile e agile. Il supporto per PHP, il framework di test più diffuso, PHPUnit , è integrato direttamente in Laravel. PHPUnit è configurato utilizzando il file `phpunit.xml` , che risiede nella directory radice di ogni nuova applicazione Laravel.

La directory dei `tests` , anche nella directory root, contiene i singoli file di test che contengono la logica per testare ogni parte della tua applicazione. Ovviamente, è responsabilità dell'utente come sviluppatore scrivere questi test mentre costruisci la tua applicazione, ma Laravel include un file di esempio, `ExampleTest.php` , per aiutarti.

```
<?php

use Illuminate\Foundation\Testing\WithoutMiddleware;
use Illuminate\Foundation\Testing\DatabaseMigrations;
use Illuminate\Foundation\Testing\DatabaseTransactions;

class ExampleTest extends TestCase
{
    /**
     * A basic functional test example.
     *
     * @return void
     */
    public function testBasicExample()
    {
        $this->visit('/')
             ->see('Laravel 5');
    }
}
```

Nel metodo `testBasicExample()` , visitiamo la pagina dell'indice del sito e ci assicuriamo di vedere il testo `Laravel 5` da qualche parte su quella pagina. Se il testo non è presente, il test fallirà e genererà un errore.

Test senza middleware e con un nuovo database

Per rendere artigianale la migrazione di un nuovo database prima dell'esecuzione dei test, `use DatabaseMigrations` . Inoltre, se vuoi evitare il middleware come Auth, `use WithoutMiddleware` .

```
<?php

use Illuminate\Foundation\Testing\WithoutMiddleware;
use Illuminate\Foundation\Testing\DatabaseMigrations;
```

```
class ExampleTest extends TestCase
{
    use DatabaseMigrations, WithoutMiddleware;

    /**
     * A basic functional test example.
     *
     * @return void
     */
    public function testExampleIndex()
    {
        $this->visit('/protected-page')
         ->see('All good');
    }
}
```

Transazioni di database per connessione a più server

`DatabaseTransactions` tratto `DatabaseTransactions` consente ai database di eseguire il rollback di tutte le modifiche durante i test. Se si desidera eseguire il rollback di più database, è necessario impostare `$connectionsToTransact` properties

```
use Illuminate\Foundation\Testing\DatabaseMigrations;

class ExampleTest extends TestCase
{
     use DatabaseTransactions;

     $connectionsToTransact =["mysql","sqlite"] //tell Laravel which database need to rollBack

    public function testExampleIndex()
    {
        $this->visit('/action/parameter')
         ->see('items');
    }
}
```

Test di installazione, utilizzando nel database di memoria

La seguente installazione assicura che il framework di test (PHPUnit) usi `:memory:` database.

config / database.php

```
'connections' => [

    'sqlite_testing' => [
        'driver'   => 'sqlite',
        'database' => ':memory:',
        'prefix'   => '',
    ],
    .
    .
    .
```

./phpunit.xml

```
    .
    .
    .
    </filter>
    <php>
        <env name="APP_ENV" value="testing"/>
        <env name="APP_URL" value="http://example.dev"/>
        <env name="CACHE_DRIVER" value="array"/>
        <env name="SESSION_DRIVER" value="array"/>
        <env name="QUEUE_DRIVER" value="sync"/>
        <env name="DB_CONNECTION" value="sqlite_testing"/>
    </php>
</phpunit>
```

Configurazione

Il file phpunit.xml è il file di configurazione predefinito per i test ed è già configurato per il test con PHPUnit.

L'ambiente di test predefinito `APP_ENV` è definito come `testing` con `array` rappresenta il driver cache `CACHE_DRIVER` . Con questa configurazione, nessun dato (sessione / cache) verrà mantenuto durante il test.

Per eseguire test su un ambiente specifico come homestead, i valori predefiniti possono essere modificati in:

```
<env name="DB_HOST" value="192.168.10.10"/>
<env name="DB_DATABASE" value="homestead"/>
<env name="DB_USERNAME" value="homestead"/>
<env name="DB_PASSWORD" value="secret"/>
```

O per utilizzare un database temporaneo *in memoria* :

```
<env name="DB_CONNECTION" value="sqlite"/>
<env name="DB_DATABASE" value=":memory:"/>
```

Un'ultima nota da tenere a mente dalla documentazione di Laravel :

> Assicurati di svuotare la cache di configurazione usando la `config:clear` comando Artisan prima di eseguire i test!

Leggi analisi online: https://riptutorial.com/it/laravel/topic/1249/analisi

Capitolo 3: Artigiano

Sintassi

- php artisan [comando] [opzioni] [argomenti]

Parametri

Comando	Descrizione
chiaro-compilato	Rimuovi il file di classe compilato
giù	Metti l'applicazione in modalità di manutenzione
ENV	Visualizza l'attuale ambiente framework
Aiuto	Visualizza la guida per un comando
elenco	Elenca i comandi
migrare	Esegui le migrazioni del database
ottimizzare	Ottimizza il framework per prestazioni migliori
servire	Servire l'applicazione sul server di sviluppo PHP
rattoppare	Interagisci con la tua applicazione
su	Porta l'applicazione fuori dalla modalità di manutenzione
nome dell'applicazione	Imposta lo spazio dei nomi dell'applicazione
auth: chiaro-reset	Token di reset password scaduti scaduti
cache: clear	Svuota la cache dell'applicazione
Cache: tavolo	Creare una migrazione per la tabella del database della cache
config: la cache	Crea un file di cache per velocizzare il caricamento della configurazione
config: chiaro	Rimuovere il file della cache di configurazione
db: seed	Seme il database con i record
evento: generare	Genera gli eventi e gli ascoltatori mancanti in base alla registrazione

Comando	Descrizione
chiave: generare	Imposta la chiave dell'applicazione
fare: auth	Impalcature di base e viste e percorsi di registrazione
fare: console	Crea un nuovo comando Artisan
fare: Controller	Crea una nuova classe controller
fare: evento	Crea una nuova classe di eventi
fare: lavoro	Crea una nuova classe di lavoro
fare: ascoltatore	Crea una nuova classe listener di eventi
fare: middleware	Crea una nuova classe middleware
fare: la migrazione	Crea un nuovo file di migrazione
fare il modello	Crea una nuova classe di modelli Eloquent
fare: Politica	Crea una nuova classe politica
fare: fornitore	Crea una nuova classe di fornitore di servizi
fare una richiesta	Crea una nuova classe di richiesta di modulo
fare: seminatrice	Crea una nuova classe di seminatrici
fare: Test	Crea una nuova classe di test
migrazione: installare	Creare il repository di migrazione
migrare: aggiornare	Reimposta e riesegui tutte le migrazioni
migrare: reset	Rollback di tutte le migrazioni di database
migrare: rollback	Rollback dell'ultima migrazione del database
migrazione: stato	Mostra lo stato di ogni migrazione
coda: fallito	Elenca tutti i lavori in coda non riusciti
coda: failed-table	Creare una migrazione per la tabella del database dei lavori in coda non riuscita
coda: a filo	Lavare tutti i lavori in coda non riusciti
coda: dimenticare	Elimina un processo in coda non riuscito
coda: ascolta	Ascolta una determinata coda

Comando	Descrizione
coda: restart	Riavvia i daemon dei worker queue dopo il loro attuale lavoro
coda: riprovare	Riprovare un processo di coda non riuscito
coda: tavolo	Creare una migrazione per la tabella del database dei lavori in coda
coda: il lavoro	Elabora il prossimo lavoro su una coda
percorso: la cache	Creare un file di cache del percorso per una registrazione del percorso più veloce
percorso: chiaro	Rimuovi il file della cache del percorso
percorso: lista	Elenca tutti i percorsi registrati
orario: eseguire	Esegui i comandi programmati
sessione: tavolo	Creare una migrazione per la tabella del database di sessione
vendor: pubblicare	Pubblicare le risorse pubblicabili dai pacchetti del fornitore
Vista: chiaro	Cancella tutti i file di visualizzazione compilati

Examples

introduzione

Artisan è un'utilità che può aiutarti a svolgere compiti ripetitivi con comandi bash. Copre molte attività, tra cui: lavorare con **migrazioni di** database e **seeding** , svuotare la **cache** , creare i file necessari per l'impostazione di **autenticazione** , **creare** nuovi *controller, modelli, classi di eventi* e molto altro ancora.

> Artisan è il nome dell'interfaccia della riga di comando inclusa in Laravel. Fornisce una serie di comandi utili per l'utilizzo durante lo sviluppo dell'applicazione.

Per visualizzare un elenco di tutti i comandi Artisan disponibili, è possibile utilizzare il comando di elenco:

```
php artisan list
```

Per saperne di più su qualsiasi comando disponibile, basta precedere il suo nome con la parola chiave **help** :

```
php artisan help [command-name]
```

Elenca tutte le rotte registrate filtrate con più metodi

```
php artisan route:list --method=GET --method=POST
```

Ciò includerà tutte le rotte che accettano simultaneamente i metodi `GET` e `POST` .

Esecuzione di comandi di Laravel Artisan utilizzando il codice PHP

Puoi anche utilizzare i comandi di Laravel Artisan dai tuoi percorsi o controller.

Per eseguire un comando utilizzando il codice PHP:

```
Artisan::call('command-name');
```

Per esempio,

```
Artisan::call('db:seed');
```

Creazione e registrazione di un nuovo comando artigiano

Puoi creare nuovi comandi tramite

```
php artisan make:command [commandName]
```

Quindi questo creerà la classe di comando [commandName] all'interno della directory `app/Console/Commands` .

all'interno di questa classe troverai `protected $signature` e `protected $description` variabili di `protected $description` , che rappresenta il nome e la descrizione del tuo comando che verrà usato per descrivere il tuo comando.

dopo aver creato il comando puoi registrare il tuo comando all'interno della classe `app/Console/Kernel.php` dove troverai la proprietà dei `commands` .

quindi puoi aggiungere il tuo comando all'interno dell'array del comando $ come:

```
protected $commands = [
    Commands\[commandName]::class
];
```

e poi posso usare il mio comando via console.

così come esempio ho chiamato il mio comando come

```
protected $signature = 'test:command';
```

Quindi ogni volta che corro

```
php artisan test:command
```

chiamerà il metodo `handle` all'interno della classe con `test:command` firma `test:command` .

Leggi Artigiano online: https://riptutorial.com/it/laravel/topic/1140/artigiano

Capitolo 4: Autenticazione

Examples

Multi autenticazione

Laravel ti consente di utilizzare più tipi di autenticazione con guardie specifiche.

In laravel 5.3 l'autenticazione multipla è leggermente diversa da Laravel 5.2

Spiegherò come implementare la funzionalità di multiautenticazione in 5.3

Innanzitutto hai bisogno di due diversi modelli utente

```
cp App/User.php App/Admin.php
```

cambia il nome della classe in Admin e imposta lo spazio dei nomi se usi modelli diversi. dovrebbe assomigliare

App \ admin.php

```
<?php

namespace App;

use Illuminate\Foundation\Auth\User as Authenticatable;
use Illuminate\Notifications\Notifiable;

class Admin extends Authenticatable
{
    use Notifiable;

    protected $fillable = ['name', 'email', 'password'];
    protected $hidden   = ['password', 'remember_token'];

}
```

Inoltre è necessario creare una migrazione per l'amministratore

```
php artisan make:migration create_admins_table
```

quindi modificare il file di migrazione con il contenuto della migrazione utente predefinita. Somiglia a questo

```
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;
```

```
class CreateAdminsTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('admins', function (Blueprint $table) {
            $table->increments('id');
            $table->string('name');
            $table->string('email')->unique();
            $table->string('password');
            $table->rememberToken();
            $table->timestamps();

            $table->softDeletes();
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::drop('admins');
    }
}
```

modifica **config / auth.php**

```
'guards'    => [
        'web'   => [
            'driver'   => 'session',
            'provider' => 'users',
        ],

        'api'   => [
            'driver'   => 'token',
            'provider' => 'users',
        ],
        //Add Admin Guard
        'admin' => [
            'driver'   => 'session',
            'provider' => 'admins',
        ],
    ],
```

e

```
'providers' => [
        'users'  => [
            'driver' => 'eloquent',
            'model'  => App\User::class,
        ],
        //Add Admins Provider
```

```
        'admins' => [
            'driver' => 'eloquent',
            'model'  => App\Admin::class,
        ],
    ],
```

Si noti che aggiungiamo due voci. uno in **guardia** variabile uno nella variabile **provider** .

E questo è come usi l'altra guardia, quindi "web"

La mia app \ Http \ Controllers \ Admin \ LoginController

```
<?php

namespace App\Http\Controllers\Admin;

use App\Http\Controllers\Controller;
use Illuminate\Foundation\Auth\AuthenticatesUsers;
use Illuminate\Support\Facades\Auth;

class AuthController extends Controller
{

    use AuthenticatesUsers;

    protected $guard = 'admin';

    protected $redirectTo = '/admin/';

    public function showLoginForm()
    {
        return view('admin.login');
    }

    protected function guard()
    {
        return Auth::guard($this->guard);
    }

}
```

questo ha bisogno di poche spiegazioni.

in poche parole, **Auth :: guard ('admin')** ti consentirà di utilizzare i metodi di autenticazione (come login, logout, registrazione, ecc.) con il tuo amministratore.

Per esempio

```
Auth::guard('admin')->login($user)
```

cercherà $ utente nella tabella amministratori e login con l'utente mentre

```
Auth::login($user)
```

funzionerà normalmente con la tabella degli utenti. La protezione predefinita è specificata in

config / auth.php con l'array *defaults* . In laravel fresco è "web".

Nel controller devi implementare i metodi da AuthenticatesUsers per mostrare i tuoi percorsi di visualizzazione personalizzati. E hai bisogno di implementare altre funzioni come la guardia per usare le tue nuove guardie utente.

In questo esempio il mio login **amministratore** è **admin / login.blade**

E implementando la funzione guard () per restituire **Auth :: guard ('admin')** tutti i metodi dei metodi AuthenticatesUsers funzionano con la guardia "admin".

Nelle versioni precedenti di laravel, questo è leggermente diverso da 5.3

in 5.2 la funzione getGuard restituisce $ guardia variabile dalla classe e dalla funzione principale (login)

```
Auth::guard($guard)->attempt(...)
```

in 5.3 la funzione di guardia restituisce l'intero Auth :: guard () e la funzione principale lo usa come

```
$this->guard()->attempt(...)
```

Leggi Autenticazione online: https://riptutorial.com/it/laravel/topic/7051/autenticazione

Capitolo 5: Autorizzazione

introduzione

Laravel fornisce un modo semplice per autorizzare azioni dell'utente su risorse specifiche. Con Autorizzazione, è possibile consentire in modo selettivo agli utenti l'accesso a determinate risorse negando l'accesso ad altri. Laravel fornisce una semplice API per la gestione delle autorizzazioni degli utenti utilizzando `Gates` e `Policies` . `Gates` forniscono un approccio basato sulla chiusura semplice all'autorizzazione che utilizza `AuthServiceProvider` mentre le `Policies` consentono di organizzare la logica di autorizzazione intorno ai modelli utilizzando le classi.

Examples

Usando i cancelli

`Gates` sono chiusure che determinano se un utente è autorizzato a eseguire una determinata azione su una risorsa. `Gates` sono in genere definite nel metodo di avvio di `AuthServiceProvider` e succintamente nominate per riflettere su ciò che sta facendo. Un esempio di gate che consente solo agli utenti premium di visualizzare alcuni contenuti sarà simile a questo:

```
Gate::define('view-content', function ($user, $content){
    return $user->isSubscribedTo($content->id);
});
```

A `Gate` riceve sempre un'istanza utente come primo argomento, non è necessario passarlo quando si utilizza il gate e può facoltativamente ricevere argomenti aggiuntivi come il modello eloquente in questione.

Autorizzazione di azioni con porte

Per utilizzare l'esempio precedente su un modello blade per nascondere il contenuto dall'utente, in genere si dovrebbe fare qualcosa del genere:

```
@can('view-content', $content)
 <! -- content here -->
@endcan
```

Per impedire completamente la navigazione verso il contenuto, puoi fare quanto segue nel tuo controller:

```
if(Gate::allows('view-content', $content)){
    /* user can view the content */
}

        OR

if(Gate::denies('view-content', $content)){
```

```
    /* user cannot view content */
}
```

Nota: non è necessario passare a questo metodo l'utente attualmente autenticato, Laravel si occupa di ciò per te.

Politiche

Le politiche sono classi che consentono di organizzare la logica di autorizzazione attorno a una risorsa del modello. Utilizzando il nostro esempio precedente, potremmo avere una `ContentPolicy` che gestisce l'accesso degli utenti al modello di `Content` .

Per rendere `ContentPolicy` , laravel fornisce un comando artigiano. Semplicemente corri

```
php artisan make:policy ContentPolicy
```

Questo renderà una classe politica vuota e verrà inserita nella cartella `app/Policies` . Se la cartella non esiste, Laravel la creerà e inserirà la classe all'interno.

Una volta creati, è necessario registrare le politiche per aiutare Laravel a sapere quali politiche utilizzare per autorizzare le azioni sui modelli. L'AuthServiceProvider di `AuthServiceProvider` , fornito con tutte le nuove installazioni di Laravel, ha una proprietà policy che associa i modelli eloquenti alle loro politiche di autorizzazione. Tutto ciò che devi fare è aggiungere la mappatura all'array.

```
protected $policies = [
    Content::class => ContentPolicy::class,
 ];
```

Politiche di scrittura

Le `Policies` Scrittura seguono lo stesso schema della scrittura di `Gates` . Il gate di autorizzazione del contenuto può essere riscritto come un criterio come questo:

```
function view($user, $content)
{
    return $user->isSubscribedTo($content->id);
}
```

Le politiche possono contenere più metodi necessari per occuparsi di tutti i casi di autorizzazione per un modello.

Autorizzazione delle azioni con le politiche

Via il modello utente

Il modello utente Laravel contiene due metodi che aiutano con le autorizzazioni che utilizzano le `Policies` ; `can` e `can't` . Questi due possono essere usati per determinare se un utente ha o meno un'autorizzazione su un modello.

Per verificare se un utente può visualizzare o meno un contenuto, puoi fare quanto segue:

```
if($user->can('view', $content)){
    /* user can view content */
}

        OR

if($user->cant('view', $content)){
    /* user cannot view content */
}
```

Tramite middleware

```
Route::get('/contents/{id}, function(Content $content){
    /* user can view content */
})->middleware('can:view,content');
```

Via controllori

Laravel fornisce un metodo di supporto, chiamato `authorize` che accetta il nome della politica e il modello associato come argomenti, e autorizza l'azione in base alla propria logica di autorizzazione o nega l'azione e lancia una `AuthorizationException` che il gestore di eccezioni di Laravel converte in un `403 HTTP response` .

```
pubic function show($id)
{
    $content = Content::find($id);

    $this->authorize('view', $content);

    /* user can view content */
}
```

Leggi Autorizzazione online: https://riptutorial.com/it/laravel/topic/9360/autorizzazione

Capitolo 6: Autorizzazioni per l'archiviazione

introduzione

Laravel richiede che alcune cartelle siano scrivibili per l'utente del server web.

Examples

Esempio

Abbiamo anche bisogno di impostare le autorizzazioni corrette per `storage` file di `storage` nel `server` . Quindi, dobbiamo dare un permesso di scrittura nella directory di archiviazione come segue:

```
$ chmod -R 777 ./storage ./bootstrap
```

o puoi usare

```
$ sudo chmod -R 777 ./storage ./bootstrap
```

Per Windows

Assicurati di essere un utente amministratore su quel computer con accesso scrivibile

```
xampp\htdocs\laravel\app\storage needs to be writable
```

Il modo NORMAL per impostare le autorizzazioni è avere i file di proprietà del server web:

```
sudo chown -R www-data:www-data /path/to/your/root/directory
```

Leggi Autorizzazioni per l'archiviazione online:
https://riptutorial.com/it/laravel/topic/9797/autorizzazioni-per-l-archiviazione

Capitolo 7: Banca dati

Examples

Più connessioni al database

Laravel consente agli utenti di lavorare su più connessioni di database. Se è necessario connettersi a più database e farli funzionare insieme, si deve fare attenzione alla configurazione della connessione.

Consenti anche di utilizzare diversi tipi di database nella stessa applicazione se richiesto.

Connessione predefinita In `config/database.php` , è possibile visualizzare la chiamata all'elemento di configurazione:

```
'default' => env('DB_CONNECTION', 'mysql'),
```

Questo nome fa riferimento al nome `mysql` qui sotto:

```
'connections' => [

    'sqlite' => [
        'driver' => 'sqlite',
        'database' =>  database_path('database.sqlite'),
        'prefix' => '',
    ],

    'mysql' => [
        'driver' => 'mysql',
        'host' => env('DB_HOST', 'localhost'),
        'port' => env('DB_PORT', '3306'),
        'database' => env('DB_DATABASE', 'forge'),
        'username' => env('DB_USERNAME', 'forge'),
        'password' => env('DB_PASSWORD', ''),
        'charset' => 'utf8',
        'collation' => 'utf8_unicode_ci',
        'prefix' => '',
        'strict' => false,
        'engine' => null,
    ],
],
```

Se non hai menzionato il nome della connessione al database in altri codici o comandi, Laravel prenderà il nome della connessione al database predefinito. tuttavia, in più connessioni di database, anche se si imposta la connessione predefinita, è meglio configurare ovunque la connessione al database utilizzata.

File di migrazione

Nel file di migrazione, se una singola connessione al database, è possibile utilizzare:

```
 Schema::create("table",function(Blueprint $table){
     $table->increments('id');
});
```

Nella connessione a più database, si utilizzerà il metodo `connection()` per comunicare a Laravel quale connessione di database si utilizza:

```
 Schema::connection("sqlite")->create("table",function(Blueprint $table){
     $table->increments('id');
});
```

Artisan Migrate

se utilizzi una singola connessione al database, eseguirai:

```
php artisan migrate
```

Tuttavia, per una connessione a più database, è meglio dire quale connessione al database mantiene i dati di migrazione. quindi eseguirai il seguente comando:

```
php artisan migrate:install --database=sqlite
```

Questo comando installerà la tabella di migrazione nel database di destinazione per preparare la migrazione.

```
php artisan migrate --database=sqlite
```

Questo comando eseguirà la migrazione e salverà i dati di migrazione nel database di destinazione

```
php artisan migrate:rollback --database=sqlite
```

Questo comando eseguirà il rollback della migrazione e salverà i dati di migrazione nel database di destinazione

Modello eloquente

Per specificare una connessione al database usando Eloquent, è necessario definire la proprietà `$connection` :

```
namespace App\Model\Sqlite;
class Table extends Model
{
    protected $table="table";
    protected $connection = 'sqlite';
}
```

Per specificare un'altra (seconda) connessione al database usando Eloquent:

```
namespace App\Model\MySql;
class Table extends Model
{
    protected $table="table";
    protected $connection = 'mysql';
}
```

Laravel utilizzerà la proprietà `$connection` definita in un modello per utilizzare la connessione specificata definita in `config/database.php` . Se la proprietà `$connection` non è definita in un modello, verrà utilizzato il valore predefinito.

Puoi anche specificare un'altra connessione usando il metodo statico `on` :

```
// Using the sqlite connection
Table::on('sqlite')->select(...)->get()
// Using the mysql connection
Table::on('mysql')->select(...)->get()
```

Database / generatore di query

Puoi anche specificare un'altra connessione usando il generatore di query:

```
// Using the sqlite connection
DB::connection('sqlite')->table('table')->select(...)->get()
// Using the mysql connection
DB::connection('mysql')->table('table')->select(...)->get()
```

Test unitario

Laravel fornisce `seeInDatabase($table,$fielsArray,$connection)` per testare il codice di connessione del database. Nel file di test dell'unità, è necessario fare come:

```
$this
    ->json(
        'GET',
        'result1/2015-05-08/2015-08-08/a/123'
    )
     ->seeInDatabase("log", ["field"=>"value"], 'sqlite');
```

In questo modo, Laravel saprà quale connessione di database testare.

Transazioni del database in Unit Test

Laravel consente al database di eseguire il rollback di tutte le modifiche durante i test. Per testare più connessioni di database, è necessario impostare `$connectionsToTransact` properties

```
use Illuminate\Foundation\Testing\DatabaseMigrations;

class ExampleTest extends TestCase
{
     use DatabaseTransactions;

     $connectionsToTransact =["mysql","sqlite"] //tell Laravel which database need to rollBack
```

```
    public function testExampleIndex()
    {
        $this->visit('/action/parameter')
         ->see('items');
    }
}
```

Leggi Banca dati online: https://riptutorial.com/it/laravel/topic/1093/banca-dati

Capitolo 8: Cassiere

Osservazioni

Laravel Cashier può essere utilizzato per la fatturazione dell'abbonamento fornendo un'interfaccia nei servizi di abbonamento di Braintree e Stripe. Oltre alla gestione base dell'abbonamento, può essere utilizzato per gestire coupon, scambiare abbonamenti, quantità, periodi di cancellazione e generazione di fatture PDF.

Examples

Impostazione delle strisce

Configurazione iniziale

Per utilizzare Stripe per la gestione dei pagamenti, è necessario aggiungere quanto segue al `composer.json` quindi eseguire l' `composer update` :

```
"laravel/cashier": "~6.0"
```

La seguente riga deve quindi essere aggiunta a `config/app.php` , il fornitore di servizi:

```
Laravel\Cashier\CashierServiceProvider
```

Impostazione Databse

Per utilizzare la cassa, dobbiamo configurare i database, se una tabella utenti non esiste già, dobbiamo crearne una e dobbiamo anche creare una tabella delle iscrizioni. L'esempio seguente modifica una tabella `users` esistente. Vedi Modelli eloquenti per maggiori informazioni sui modelli.

Per utilizzare la cassa, crea una nuova migrazione e aggiungi quanto segue che conseguirà quanto sopra:

```
// Adjust users table

Schema::table('users', function ($table) {
    $table->string('stripe_id')->nullable();
    $table->string('card_brand')->nullable();
    $table->string('card_last_four')->nullable();
    $table->timestamp('trial_ends_at')->nullable();
});

//Create subscriptions table

Schema::create('subscriptions', function ($table) {
    $table->increments('id');
    $table->integer('user_id');
    $table->string('name');
```

```
    $table->string('stripe_id');
    $table->string('stripe_plan');
    $table->integer('quantity');
    $table->timestamp('trial_ends_at')->nullable();
    $table->timestamp('ends_at')->nullable();
    $table->timestamps();
});
```

Abbiamo quindi bisogno di eseguire `php artisan migrate` per aggiornare il nostro database.

Impostazione del modello

Dovremo quindi aggiungere il tratto fatturabile al modello Utente trovato in `app/User.php` e modificarlo nel modo seguente:

```
use Laravel\Cashier\Billable;

class User extends Authenticatable
{
    use Billable;
}
```

Chiavi a strisce

Per assicurarci di terminare i soldi con il nostro account Stripe, dobbiamo configurarlo nel file `config/services.php` aggiungendo la seguente riga:

```
'stripe' => [
    'model'  => App\User::class,
    'secret' => env('STRIPE_SECRET'),
],
```

Sostituzione di `STRIPE_SECRET` con la propria chiave segreta a strisce.

Dopo aver completato questa operazione, Cashier and Strip è configurato in modo da poter continuare con la configurazione delle iscrizioni.

Leggi Cassiere online: https://riptutorial.com/it/laravel/topic/7474/cassiere

Capitolo 9: Classe CustomException in Laravel

introduzione

Le eccezioni PHP vengono generate quando si verifica un evento o un errore senza precedenti.

Come regola generale, un'eccezione non dovrebbe essere utilizzata per controllare la logica dell'applicazione come if-statement e dovrebbe essere una sottoclasse della classe Exception.

Uno dei principali vantaggi di avere tutte le eccezioni catturate da una singola classe è che siamo in grado di creare gestori di eccezioni personalizzati che restituiscono messaggi di risposta diversi a seconda dell'eccezione.

Examples

Classe CustomException in laravel

tutti gli errori e le eccezioni, sia personalizzati che predefiniti, sono gestiti dalla classe Handler in app / Exceptions / Handler.php con l'aiuto di due metodi.

- rapporto()
- render ()

```
public function render($request, Exception $e)
{
 //check if exception is an instance of ModelNotFoundException.
 if ($e instanceof ModelNotFoundException)
 {
     // ajax 404 json feedback
     if ($request->ajax())
     {
         return response()->json(['error' => 'Not Found'], 404);
     }
     // normal 404 view page feedback
     return response()->view('errors.missing', [], 404);
  }
   return parent::render($request, $e);
}
```

quindi creare la vista correlata all'errore nella cartella errori denominata 404.blade.php

Utente non trovato.

Hai rotto il bilancio di Internet

Leggi Classe CustomException in Laravel online:

https://riptutorial.com/it/laravel/topic/9550/classe-customexception-in-laravel

Capitolo 10: code

introduzione

Le code consentono alla tua applicazione di riservare parti di lavoro che richiedono molto tempo per essere gestite da un processo in background.

Examples

Casi d'uso

Ad esempio, se si sta inviando un'e-mail a un cliente dopo aver avviato un'attività, è meglio reindirizzare immediatamente l'utente alla pagina successiva durante l'accodamento dell'e-mail da inviare in background. Ciò velocizzerà il tempo di caricamento per la pagina successiva, poiché l'invio di un'email può richiedere talvolta diversi secondi o più.

Un altro esempio potrebbe essere l'aggiornamento di un sistema di inventario dopo che un cliente ha effettuato il checkout con il proprio ordine. Anziché attendere il completamento delle chiamate API, che potrebbero richiedere alcuni secondi, è possibile reindirizzare immediatamente l'utente alla pagina di verifica del checkout, accodando le chiamate API in modo che si verifichino in background.

Configurazione del driver della coda

Ciascuno dei driver di coda di Laravel è configurato dal file `config/queue.php` . Un gestore di code è il gestore per la gestione di come eseguire un lavoro in coda, identificando se i lavori hanno avuto esito positivo o negativo, e tentando di nuovo il lavoro se configurato per farlo.

Subito dopo, Laravel supporta i seguenti driver di coda:

sync

Sync, o synchronous, è il driver di coda predefinito che esegue un lavoro in coda all'interno del processo esistente. Con questo driver abilitato, non si ha effettivamente alcuna coda mentre il lavoro in coda viene eseguito immediatamente. Ciò è utile per scopi locali o di test, ma chiaramente non è raccomandato per la produzione in quanto rimuove il vantaggio prestazionale derivante dall'impostazione della coda.

database

Questo driver memorizza i lavori in coda nel database. Prima di abilitare questo driver, è necessario creare tabelle di database per archiviare i lavori in coda e non riusciti:

```
php artisan queue:table
php artisan migrate
```

sqs

Questo driver di coda utilizza il servizio Simple Queue di Amazon per gestire i lavori in coda. Prima di abilitare questo lavoro è necessario installare il seguente pacchetto di composizione: `aws/aws-sdk-php ~3.0`

Inoltre, se si prevede di utilizzare i ritardi per i lavori in coda, Amazon SQS supporta solo un ritardo massimo di 15 minuti.

iron

Questo driver di coda utilizza Iron per gestire i lavori in coda.

redis

Questo driver di coda utilizza un'istanza di Redis per gestire i lavori in coda. Prima di utilizzare questo driver di coda, è necessario configurare una copia di Redis e installare la seguente dipendenza del compositore: `predis/predis ~1.0`

beanstalkd

Questo driver di coda utilizza un'istanza di Beanstalk per gestire i lavori in coda. Prima di utilizzare questo driver di coda, è necessario configurare una copia di Beanstalk e installare la seguente dipendenza del compositore: `pda/pheanstalk ~3.0`

null

Specificando null come driver della coda, si eliminano i lavori in coda.

Leggi code online: https://riptutorial.com/it/laravel/topic/2651/code

Capitolo 11: collezioni

Sintassi

- $ collection = collect (['Value1', 'Value2', 'Value3']); // Chiavi predefinite su 0, 1, 2, ...,

Osservazioni

`Illuminate\Support\Collection` offre un'interfaccia fluida e comoda per gestire array di dati. Potresti averlo usato senza saperlo, ad esempio le query di modello che recuperano più record restituiscono un'istanza di `Illuminate\Support\Collection` .

Per la documentazione aggiornata sulle collezioni è possibile trovare la documentazione ufficiale qui

Examples

Creazione di raccolte

Usando l'helper `collect()` , puoi creare facilmente nuove istanze di raccolta passando un array come:

```
$fruits = collect(['oranges', 'peaches', 'pears']);
```

Se non si desidera utilizzare le funzioni di supporto, è possibile creare una nuova raccolta utilizzando direttamente la classe:

```
$fruits = new Illuminate\Support\Collection(['oranges', 'peaches', 'pears']);
```

Come menzionato nelle note, Modelli per impostazione predefinita restituiscono un'istanza `Collection` , tuttavia sei libero di creare le tue raccolte in base alle esigenze. Se non viene specificato alcun array durante la creazione, verrà creata una raccolta vuota.

dove()

È possibile selezionare determinati elementi da una raccolta utilizzando il metodo `where()` .

```
$data = [
    ['name' => 'Taylor',  'coffee_drinker' => true],
    ['name' => 'Matt', 'coffee_drinker' => true]
];

$matt = collect($data)->where('name', 'Matt');
```

Questo bit di codice selezionerà tutti gli elementi della raccolta in cui il nome è "Matt". In questo

caso, viene restituito solo il secondo elemento.

annidamento

Proprio come la maggior parte dei metodi di array in Laravel, `where()` supporta anche la ricerca di elementi nidificati. Estendiamo l'esempio precedente aggiungendo un secondo array:

```
$data = [
    ['name' => 'Taylor',  'coffee_drinker' => ['at_work' => true, 'at_home' => true]],
    ['name' => 'Matt', 'coffee_drinker' => ['at_work' => true, 'at_home' => false]]
];

$coffeeDrinkerAtHome = collect($data)->where('coffee_drinker.at_home', true);
```

Questo restituirà solo Taylor, mentre beve il caffè a casa. Come puoi vedere, il nesting è supportato usando la notazione a punti.

aggiunte

Quando si crea una raccolta di oggetti invece di matrici, queste possono essere filtrate usando `where()` . La Collezione proverà quindi a ricevere tutte le proprietà desiderate.

5.3

Si noti che dal momento che Laravel 5.3 il metodo `where()` proverà a confrontare liberamente i valori per impostazione predefinita. Ciò significa che durante la ricerca di `(int)1` , verranno restituite tutte le voci contenenti `'1'` . Se non ti piace questo comportamento, puoi usare il metodo `whereStrict()` .

Usando Ottieni valore di ricerca o restituisci valori predefiniti

Ti trovi spesso in una situazione in cui devi trovare un valore corrispondente alle variabili e le raccolte ti hanno coperto.

Nell'esempio seguente abbiamo ottenuto tre diverse localizzazioni in una matrice con un corrispondente codice di chiamata assegnato. Vogliamo essere in grado di fornire un locale e in cambio ottenere il codice di chiamata associato. Il secondo parametro in `get` è un parametro predefinito se il primo parametro non viene trovato.

```
function lookupCallingCode($locale)
{
    return collect([
        'de_DE' => 49,
        'en_GB' => 44,
        'en_US' => 1,
    ])->get($locale, 44);
}
```

Nell'esempio sopra possiamo fare quanto segue

```
lookupCallingCode('de_DE'); // Will return 49
lookupCallingCode('sv_SE'); // Will return 44
```

È anche possibile passare una richiamata come valore predefinito. Il risultato della richiamata verrà restituito se la chiave specificata non esiste:

```
    return collect([
        'de_DE' => 49,
        'en_GB' => 44,
        'en_US' => 1,
    ])->get($locale, function() {
        return 44;
    });
```

Utilizzare Contiene per verificare se una raccolta soddisfa determinate condizioni

Un problema comune è avere una raccolta di elementi che devono soddisfare tutti determinati criteri. Nell'esempio seguente abbiamo raccolto due articoli per un programma di dieta e vogliamo verificare che la dieta non contenga cibi malsani.

```
// First we create a collection
$diet = collect([
    ['name' => 'Banana', 'calories' => '89'],
    ['name' => 'Chocolate', 'calories' => '546']
]);

// Then we check the collection for items with more than 100 calories
$isUnhealthy = $diet->contains(function ($i, $snack) {
    return $snack["calories"] >= 100;
});
```

Nel caso precedente, la variabile `$isUnhealthy` verrà impostata su `true` poiché il cioccolato soddisfa la condizione e la dieta è quindi malsana.

Usando Pluck per estrarre determinati valori da una collezione

Ti troverai spesso con una raccolta di dati in cui sei interessato solo a parti dei dati.

Nell'esempio seguente abbiamo ottenuto un elenco di partecipanti a un evento e vogliamo fornire una guida turistica con un semplice elenco di nomi.

```
// First we collect the participants
$participants = collect([
    ['name' => 'John', 'age' => 55],
    ['name' => 'Melissa', 'age' => 18],
    ['name' => 'Bob', 'age' => 43],
    ['name' => 'Sara', 'age' => 18],
]);
```

```
// Then we ask the collection to fetch all the names
$namesList = $partcipants->pluck('name')
// ['John', 'Melissa', 'Bob', 'Sara'];
```

Puoi anche usare il `pluck` per raccolte di oggetti o array / oggetti nidificati con notazione a punti.

```
$users = User::all(); // Returns Eloquent Collection of all users
$usernames = $users->pluck('username'); // Collection contains only user names

$users->load('profile'); // Load a relationship for all models in collection

// Using dot notation, we can traverse nested properties
$names = $users->pluck('profile.first_name'); // Get all first names from all user profiles
```

Utilizzo di Map per manipolare ciascun elemento in una raccolta

Spesso è necessario cambiare il modo in cui una serie di dati è strutturata e manipolare determinati valori.

Nell'esempio seguente abbiamo ottenuto una raccolta di libri con un importo di sconto allegato. Ma preferiamo avere un elenco di libri con un prezzo già scontato.

```
$books = [
    ['title' => 'The Pragmatic Programmer', 'price' => 20, 'discount' => 0.5],
    ['title' => 'Continuous Delivery', 'price' => 25, 'discount' => 0.1],
    ['title' => 'The Clean Coder', 'price' => 10, 'discount' => 0.75],
];

$discountedItems =  collect($books)->map(function ($book) {
   return ['title' => $book["title"], 'price' => $book["price"] * $book["discount"]];
});

//[
//    ['title' => 'The Pragmatic Programmer', 'price' => 10],
//    ['title' => 'Continuous Delivery', 'price' => 12.5],
//    ['title' => 'The Clean Coder', 'price' => 5],
//]
```

Questo potrebbe anche essere usato per cambiare le chiavi, diciamo che volevamo cambiare il `title` della chiave per `name` questa sarebbe una soluzione adatta.

Usando sum, avg, min o max su una collezione per i calcoli statistici

Le raccolte forniscono anche un modo semplice per eseguire calcoli statistici semplici.

```
$books = [
    ['title' => 'The Pragmatic Programmer', 'price' => 20],
    ['title' => 'Continuous Delivery', 'price' => 30],
    ['title' => 'The Clean Coder', 'price' => 10],
]

$min = collect($books)->min('price'); // 10
$max = collect($books)->max('price'); // 30
$avg = collect($books)->avg('price'); // 20
```

```
$sum = collect($books)->sum('price'); // 60
```

Ordinamento di una collezione

Esistono diversi modi per ordinare una raccolta.

Ordinare()

Il `sort` metodo ordina la collezione:

```
$collection = collect([5, 3, 1, 2, 4]);

$sorted = $collection->sort();

echo $sorted->values()->all();

returns : [1, 2, 3, 4, 5]
```

Il metodo di `sort` consente anche di passare un callback personalizzato con il proprio algoritmo. Sotto il cappuccio usa l' `usort` php.

```
$collection = $collection->sort(function ($a, $b) {
    if ($a == $b) {
        return 0;
    }
    return ($a < $b) ? -1 : 1;
});
```

Ordina per()

Il metodo `sortBy` ordina la collezione con la chiave data:

```
$collection = collect([
['name' => 'Desk', 'price' => 200],
['name' => 'Chair', 'price' => 100],
['name' => 'Bookcase', 'price' => 150],
]);

$sorted = $collection->sortBy('price');

echo $sorted->values()->all();

returns:  [
        ['name' => 'Chair', 'price' => 100],
        ['name' => 'Bookcase', 'price' => 150],
        ['name' => 'Desk', 'price' => 200],
    ]
```

Il metodo `sortBy` consente di utilizzare il formato di notazione a punti per accedere a una chiave più profonda al fine di ordinare una matrice multidimensionale.

```
$collection = collect([
    ["id"=>1,"product"=>['name' => 'Desk', 'price' => 200]],
    ["id"=>2, "product"=>['name' => 'Chair', 'price' => 100]],
    ["id"=>3, "product"=>['name' => 'Bookcase', 'price' => 150]],
    ]);

$sorted = $collection->sortBy("product.price")->toArray();

return: [
  ["id"=>2, "product"=>['name' => 'Chair', 'price' => 100]],
  ["id"=>3, "product"=>['name' => 'Bookcase', 'price' => 150]],
  ["id"=>1,"product"=>['name' => 'Desk', 'price' => 200]],
]
```

SortByDesc ()

Questo metodo ha la stessa firma del metodo `sortBy` , ma `sortBy` la raccolta nell'ordine opposto.

Utilizzo di reduce ()

Il `reduce` metodo riduce la raccolta di un singolo valore, passando il risultato di ogni iterazione nella successiva iterazione. Si prega di vedere ridurre il metodo .

Il metodo di `reduce` esegue il loop di ogni elemento con una raccolta e produce un nuovo risultato alla successiva iterazione. Ogni risultato dell'ultima iterazione viene passato attraverso il primo parametro (negli esempi seguenti, come `$carry`).

Questo metodo può eseguire molta elaborazione su set di dati di grandi dimensioni. Ad esempio i seguenti esempi, utilizzeremo il seguente esempio di dati dello studente:

```
 $student = [
    ['class' => 'Math', 'score' => 60],
    ['class' => 'English', 'score' => 61],
    ['class' => 'Chemistry', 'score' => 50],
    ['class' => 'Physics', 'score' => 49],
];
```

Somma il punteggio totale dello studente

```
$sum = collect($student)
    ->reduce(function($carry, $item){
        return $carry + $item["score"];
    }, 0);
```

Risultato: `220`

Spiegazione:

- `$carry` è il risultato dell'ultima iterazione.
- Il secondo parametro è il valore predefinito per $ carry nel primo round di iterazione. In questo caso, il valore predefinito è 0

Passa uno studente se tutti i loro punteggi sono> = 50

```
$isPass = collect($student)
    ->reduce(function($carry, $item){
        return $carry && $item["score"] >= 50;
    }, true);
```

Risultato: `false`

Spiegazione:

- Il valore predefinito di $ carry è true
- Se tutto il punteggio è superiore a 50, il risultato restituirà true; se inferiore a 50, restituisce falso.

Fallisci uno studente se il punteggio è <50

```
$isFail = collect($student)
    ->reduce(function($carry, $item){
        return $carry || $item["score"] < 50;
    }, false);
```

Risultato: `true`

Spiegare:

- il valore predefinito di $ carry è falso
- se un punteggio è inferiore a 50, restituisce true; se tutti i punteggi sono maggiori di 50, restituire false.

Restituire l'oggetto con il punteggio più alto

```
$highestSubject = collect($student)
    ->reduce(function($carry, $item){
        return $carry === null || $item["score"] > $carry["score"] ? $item : $carry;
    });
```

risultato: `[ "subject" => "English", "score" => 61 ]`

Spiegare:

- Il secondo parametro non è fornito in questo caso.
- Il valore predefinito di $ carry è nullo, quindi verifichiamo che ciò sia nel nostro condizionale.

Uso di macro () per estendere le raccolte

La funzione `macro()` consente di aggiungere nuove funzionalità agli oggetti `Illuminate\Support\Collection`

Uso:

```
Collection::macro("macro_name", function ($parameters) {
    // Your macro
});
```

Per esempio:

```
Collection::macro('uppercase', function () {
    return $this->map(function ($item) {
        return strtoupper($item);
    });
});

collect(["hello", "world"])->uppercase();
```

Risultato: `["HELLO", "WORLD"]`

Utilizzando la sintassi di array

L'oggetto `Collection` implementa l'interfaccia `ArrayAccess` e `IteratorAggregate` , consentendone l'utilizzo come una matrice.

Accedi all'elemento di raccolta:

```
 $collection = collect([1, 2, 3]);
 $result = $collection[1];
```

Risultato: `2`

Assegna nuovo elemento:

```
$collection = collect([1, 2, 3]);
$collection[] = 4;
```

Risultato: `$collection` è `[1, 2, 3, 4]`

Collezione Loop:

```
$collection = collect(["a" => "one", "b" => "two"]);
$result = "";
foreach($collection as $key => $value){
    $result .= "(".$key.": ".$value.") ";
}
```

Risultato: `$result` è `(a: one) (b: two)`

Conversione da matrice a raccolta:

Per convertire una raccolta in un array PHP nativo, utilizzare:

```
$array = $collection->all();
//or
```

```
$array = $collection->toArray()
```

Per convertire una matrice in una raccolta, utilizzare:

```
$collection = collect($array);
```

Utilizzo di collezioni con funzioni di array

Si prega di essere consapevoli del fatto che le collezioni sono oggetti normali che non verranno convertiti correttamente se usati da funzioni che richiedono esplicitamente array, come `array_map($callback)` .

Assicurati di convertire prima la raccolta, oppure, se disponibile, usa il metodo fornito dalla classe `Collection` : `$collection->map($callback)`

Leggi collezioni online: https://riptutorial.com/it/laravel/topic/2358/collezioni

Capitolo 12: Connessioni DB multiple in Laravel

Examples

Passi iniziali

Connessioni multiple di database, di qualsiasi tipo, possono essere definite all'interno del file di configurazione del database (probabile `app/config/database.php`). Ad esempio, per estrarre dati da 2 database MySQL definirli entrambi separatamente:

```
<?php
return array(

    'default' => 'mysql',

    'connections' => array(

        # Our primary database connection
        'mysql' => array(
            'driver'    => 'mysql',
            'host'      => 'host1',
            'database'  => 'database1',
            'username'  => 'user1',
            'password'  => 'pass1'
            'charset'   => 'utf8',
            'collation' => 'utf8_unicode_ci',
            'prefix'    => '',
        ),

        # Our secondary database connection
        'mysql2' => array(
            'driver'    => 'mysql',
            'host'      => 'host2',
            'database'  => 'database2',
            'username'  => 'user2',
            'password'  => 'pass2'
            'charset'   => 'utf8',
            'collation' => 'utf8_unicode_ci',
            'prefix'    => '',
        ),
    ),
);
```

La connessione predefinita è ancora impostata su `mysql` . Ciò significa che, se non diversamente specificato, l'applicazione utilizza la connessione `mysql` .

Utilizzo del builder Schema

All'interno di Schema Builder, utilizzare la facciata dello schema con qualsiasi connessione. Esegui il metodo `connection()` per specificare quale connessione usare:

```
Schema::connection('mysql2')->create('some_table', function($table)
{
    $table->increments('id');
});
```

Utilizzando DB query builder

Simile a Schema Builder, definire una connessione nel Query Builder:

```
$users = DB::connection('mysql2')->select(...);
```

Uso di Eloquent

Esistono diversi modi per definire quale connessione usare nei modelli Eloquent. Un modo è impostare la variabile $ connection nel modello:

```
<?php

class SomeModel extends Eloquent {

    protected $connection = 'mysql2';

}
```

La connessione può anche essere definita in fase di esecuzione tramite il metodo `setConnection` .

```
<?php

class SomeController extends BaseController {

    public function someMethod()
    {
        $someModel = new SomeModel;

        $someModel->setConnection('mysql2');

        $something = $someModel->find(1);

        return $something;
    }
}
```

Da Laravel Documentation

È possibile accedere a ogni singola connessione tramite il metodo di connessione sulla facciata del `DB` , anche quando sono definite più connessioni. Il `name` passato al metodo di `connection` dovrebbe corrispondere a una delle connessioni elencate nel file di configurazione `config/database.php` :

```
$users = DB::connection('foo')->select(...);
```

È anche possibile accedere al raw, l'istanza PDO sottostante utilizzando il metodo getPdo su un'istanza di connessione:

```
$pdo = DB::connection()->getPdo();
```

https://laravel.com/docs/5.4/database#using-multiple-database-connections

Leggi Connessioni DB multiple in Laravel online:
https://riptutorial.com/it/laravel/topic/9605/connessioni-db-multiple-in-laravel

Capitolo 13: Controller

introduzione

Invece di definire tutta la logica di gestione delle richieste come Chiusure nei file di percorso, è possibile che si desideri organizzare questo comportamento utilizzando le classi Controller. I controllori possono raggruppare la logica di gestione delle richieste correlate in una singola classe. I controller sono memorizzati nella directory `app/Http/Controllers` per impostazione predefinita.

Examples

Controller di base

```
<?php

namespace App\Http\Controllers;

use App\User;
use App\Http\Controllers\Controller;

class UserController extends Controller
{
    /**
     * Show the profile for the given user.
     *
     * @param  int  $id
     * @return Response
     */
    public function show($id)
    {
        return view('user.profile', ['user' => User::findOrFail($id)]);
    }
}
```

È possibile definire un percorso per questa azione del controller in questo modo:

```
Route::get('user/{id}', 'UserController@show');
```

Ora, quando una richiesta corrisponde all'URI del percorso specificato, verrà eseguito il metodo `show` sulla classe `UserController` . Naturalmente, i parametri del percorso verranno anche passati al metodo.

Controller Middleware

Il middleware può essere assegnato alle rotte del controller nei file di percorso:

```
Route::get('profile', 'UserController@show')->middleware('auth');
```

Tuttavia, è più conveniente specificare il middleware all'interno del costruttore del controller. Utilizzando il metodo middleware dal costruttore del controller, è possibile assegnare facilmente il

middleware all'azione del controller.

```
class UserController extends Controller
{
    /**
     * Instantiate a new controller instance.
     *
     * @return void
     */
    public function __construct()
    {
        $this->middleware('auth');

        $this->middleware('log')->only('index');

        $this->middleware('subscribed')->except('store');
    }
}
```

Controller di risorse

Il routing delle risorse di Laravel assegna le rotte "CRUD" tipiche a un controller con una singola riga di codice. Ad esempio, potresti voler creare un controller che gestisca tutte le richieste HTTP per "foto" memorizzate dalla tua applicazione. Usando il comando `make:controller` Artisan, possiamo creare rapidamente un controller di questo tipo:

```
php artisan make:controller PhotoController --resource
```

Questo comando genererà un controller su `app/Http/Controllers/PhotoController.php` . Il controller conterrà un metodo per ciascuna delle operazioni di risorse disponibili.

Esempio di aspetto di un controller di risorse

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class PhotoController extends Controller
{
    /**
     * Display a listing of the resource.
     *
     * @return \Illuminate\Http\Response
     */
    public function index()
    {
        //
    }

    /**
     * Show the form for creating a new resource.
     *
     * @return \Illuminate\Http\Response
```

```
     */
    public function create()
    {
        //
    }

    /**
     * Store a newly created resource in storage.
     *
     * @param  \Illuminate\Http\Request  $request
     * @return \Illuminate\Http\Response
     */
    public function store(Request $request)
    {
        //
    }

    /**
     * Display the specified resource.
     *
     * @param  int  $id
     * @return \Illuminate\Http\Response
     */
    public function show($id)
    {
        //
    }

    /**
     * Show the form for editing the specified resource.
     *
     * @param  int  $id
     * @return \Illuminate\Http\Response
     */
    public function edit($id)
    {
        //
    }

    /**
     * Update the specified resource in storage.
     *
     * @param  \Illuminate\Http\Request  $request
     * @param  int  $id
     * @return \Illuminate\Http\Response
     */
    public function update(Request $request, $id)
    {
        //
    }

    /**
     * Remove the specified resource from storage.
     *
     * @param  int  $id
     * @return \Illuminate\Http\Response
     */
    public function destroy($id)
    {
        //
    }
```

```
}
```

L'esempio del controller risorse condivide il nome del metodo di quelli nella tabella seguente.

Successivamente, è possibile registrare un percorso pieno di risorse per il controller:

```
Route::resource('photos', 'PhotoController');
```

Questa dichiarazione di percorso singolo crea più percorsi per gestire una varietà di azioni sulla risorsa. Il controllore generato avrà già metodi stubati per ciascuna di queste azioni, incluse le note che ti informano dei verbi HTTP e degli URI che gestiscono.

Azioni gestite dal controllore delle risorse

Verbo	URI	Azione	Nome del percorso
OTTENERE	`/photos`	indice	photos.index
OTTENERE	`/photos/create`	creare	photos.create
INVIARE	`/photos`	memorizzare	photos.store
OTTENERE	`/photos/{photo}`	mostrare	photos.show
OTTENERE	`/photos/{photo}/edit`	modificare	photos.edit
PUT / PATCH	`/photos/{photo}`	aggiornare	photos.update
ELIMINA	`/photos/{photo}`	distruggere	photos.destroy

Leggi Controller online: https://riptutorial.com/it/laravel/topic/10604/controller

Capitolo 14: costanti

Examples

Esempio

Per prima cosa devi creare un file constants.php ed è una buona pratica creare questo file all'interno di app / config / cartella. Puoi anche aggiungere il file constants.php nel file compose.json.

File di esempio:

> app / config / constants.php

Costanti basate su array all'interno del file:

```
return [
    'CONSTANT' => 'This is my first constant.'
];
```

E puoi ottenere questa costante includendo la `Config` della facciata:

```
use Illuminate\Support\Facades\Config;
```

Quindi ottieni il valore per nome costante `CONSTANT` come di seguito:

```
echo Config::get('constants.CONSTANT');
```

E il risultato sarebbe il valore:

> Questa è la mia prima costante.

Leggi costanti online: https://riptutorial.com/it/laravel/topic/9192/costanti

Capitolo 15: Denominazione dei file durante il caricamento con Laravel su Windows

Parametri

Param / Funzione	Descrizione
upload di file	nome del campo `<input>`
$ samplename	potrebbe anche essere una stringa generata dinamicamente o il nome del file caricato dall'utente
app_path ()	è l'helper di Laravel per fornire il percorso assoluto all'applicazione
getCLientOriginalExtension ()	Laravel wrapper per recuperare l'estensione del file caricato dall'utente come era sul computer dell'utente

Examples

Generazione di nomi di file con data e ora per i file caricati dagli utenti.

Di seguito non funzionerà su una macchina Windows

```
$file = $request->file('file_upload');
$sampleName = 'UserUpload';
$destination = app_path() . '/myStorage/';
$fileName = $sampleName . '-' . date('Y-m-d-H:i:s') . '.' .
$file->getClientOriginalExtension();
$file->move($destination, $fileName);
```

Viene generato un errore come "Impossibile spostare il file in / percorso ..."

Perché? - Funziona perfettamente su un server Ubuntu
Il motivo è che sui `colon ':'` Windows `colon ':'` non è permesso in un nome file dove è consentito su linux. Questa è una cosa così piccola che potremmo non notarlo in anticipo e continuare a chiederci perché un codice che gira bene su Ubuntu (Linux) non funziona?
La nostra prima impressione sarebbe quella di controllare i permessi del file e cose del genere, ma potremmo non notare che `colon ':'` è il colpevole qui.
Quindi, per caricare file su Windows, **non utilizzare i `colon':'` durante la generazione di nomi di file con data e ora** , invece di fare qualcosa di simile di seguito:

```
$filename = $sampleName . '-' . date('Y-m-d-H_i_s') . '.' . $file-
>getClientOriginalExtension();  //ex output UserUpload-2016-02-18-11_25_43.xlsx
```

```
                                    OR

$filename = $sampleName . '-' .date('Y-m-d H i s') . '.' . $file-
>getClientOriginalExtension();  //ex output UserUpload-2016-02-18 11 25 43.xlsx

                                    OR

$filename = $sampleName . '-'.date('Y-m-d_g-i-A').'.' . $file->getClientOriginalExtension();
//ex output UserUpload-2016-02-18_11-25-AM.xlsx
```

Leggi Denominazione dei file durante il caricamento con Laravel su Windows online: https://riptutorial.com/it/laravel/topic/2629/denominazione-dei-file-durante-il-caricamento-con-laravel-su-windows

Capitolo 16: Distribuire l'applicazione Laravel 5 su Shared Hosting su Linux Server

Osservazioni

Per ulteriori informazioni sulla distribuzione del progetto Laravel sull'hosting condiviso, visitare questo repository Github.

Examples

Laravel 5 App su Shared Hosting su Linux Server

Di default, la cartella `public` del progetto Laravel espone il contenuto dell'app che può essere richiesto da qualsiasi luogo da chiunque, il resto del codice dell'app è invisibile o inaccessibile a chiunque non abbia le autorizzazioni appropriate.

Dopo aver sviluppato l'applicazione sul tuo computer di sviluppo, deve essere trasferito su un server di produzione in modo che sia possibile accedervi da Internet da qualsiasi luogo, giusto?

Per la maggior parte delle app / siti Web la prima scelta è quella di utilizzare il pacchetto di hosting condiviso da provider di servizi di hosting come GoDaddy, HostGator ecc. Principalmente a causa del basso costo.

> **nota:** si può chiedere al vostro provider di modificare manualmente **document_root,** quindi tutto quello che dovete fare è caricare la vostra applicazione laravel al server (via FTP), richiesta di modifica di root per **{app} / pubblico** e si dovrebbe essere buono.

Tali pacchetti di hosting condiviso, tuttavia, presentano limitazioni in termini di accesso al terminale e autorizzazioni per i file. Per impostazione predefinita si deve caricare la loro app / codice nella cartella `public_html` sul proprio account di hosting condiviso.

Quindi, se vuoi caricare un progetto Laravel su un account di hosting condiviso, come faresti? Dovresti caricare l'intera app (cartella) nella cartella `public_html` sul tuo account di hosting condiviso? - **Certamente NO**

Perché tutto nella cartella `public_html` è accessibile "pubblicamente cioè da chiunque", il che costituirebbe un grosso rischio per la sicurezza.

Passi per caricare un progetto su un account di hosting condiviso: la via di Laravel

Passo 1
Crea una cartella chiamata laravel (o qualcosa che ti piace) allo stesso livello della cartella `public_html` .

```
Eg:
/
|--var
    |---www
        |----laravel       //create this folder in your shared hosting account
        |----public_html
        |----log
```

Passo 2
Copia ogni cosa tranne la cartella `public` dal tuo progetto laravel (sul computer di sviluppo) nella cartella `laravel` (sull'host del server - account di hosting condiviso).
Puoi usare:

- C-panel: quale sarebbe l'opzione più lenta
- Client FTP: come ***FileZilla*** per collegarti al tuo account di hosting condiviso e trasferire file e cartelle tramite il caricamento FTP
- Connetti unità di rete: puoi anche creare un'unità di rete mappata sul tuo computer di sviluppo per collegarti alla cartella radice del tuo account di hosting condiviso usando " ftp: // nome-dominio " come indirizzo di rete.

Passaggio 3
Apri la cartella `public` del tuo progetto laravel (sul computer di sviluppo), copia tutto e incolla nella cartella `public_html` (sull'host del server - account di hosting condiviso).
Passaggio 4
Ora apri il file `index.php` nella cartella `public_html` sull'account di hosting condiviso (nell'editor di cpanel o in qualsiasi altro editor connesso) e:

Modificare:

```
require __DIR__.'/../bootstrap/autoload.php';
```

A:

```
require __DIR__.'/../laravel/bootstrap/autoload.php';
```

E cambia:

```
$app = require_once __DIR__.'/../bootstrap/app.php';
```

A:

```
$app = require_once __DIR__.'/../laravel/bootstrap/app.php';
```

Salva e chiudi.

Passaggio 5
Ora vai nella cartella `laravel` (sul server di hosting condiviso `laravel`) e apri `server.php` file `server.php`
Modificare

```
require_once __DIR__.'/public/index.php';
```

A:

```
require_once __DIR__.'../public_html/index.php';
```

Salva e chiudi.

Passaggio 6
Imposta i permessi dei file per la cartella `laravel/storage` (in modo ricorsivo) e tutti i file, le sottocartelle e il file all'interno di essi su un account di hosting condiviso - server su `777` .
Nota: fai attenzione ai permessi del file in linux, sono come una spada a doppio taglio, se non usati correttamente, potrebbero rendere la tua app vulnerabile agli attacchi. Per comprendere i permessi dei file Linux puoi leggere https://www.linux.com/learn/tutorials/309527-understanding-linux-file-permissions

Passaggio 7

Poiché il file `.env` del server locale / di sviluppo è Ignorato da git, dovrebbe essere ignorato poiché contiene tutte le variabili di ambiente, tra cui APP_KEY, e non dovrebbe essere esposto al pubblico spingendolo nei repository ". Si può anche vedere che `.gitignore` file è `.env` accennato quindi non caricarlo sul repository.

Dopo aver seguito tutti i passi precedenti, crea un file `.env` nella cartella laravel e aggiungi tutta la variabile d'ambiente che hai usato dal file `.env` del server locale / di sviluppo al file `.env` del server di produzione.

Esistono anche file di configurazione come `app.php` , `database.php` nella cartella config dell'applicazione laravel che definisce queste variabili come predefinite nel secondo parametro di `env()` ma non codificano i valori in questi file in quanto influenzeranno il file di configurazione degli utenti che estraggono il tuo repository. Quindi si consiglia di creare manualmente il file `.env` !

Inoltre laravel fornisce il file `.env-example` che puoi usare come riferimento.

Questo è tutto.

Ora quando visiti l'URL che hai configurato come dominio con il tuo server, la tua app laravel dovrebbe funzionare proprio come ha funzionato sul tuo localhost - macchina di sviluppo, mentre il codice dell'applicazione è ancora sicuro e non accessibile da chiunque senza autorizzazioni appropriate per i file.

Leggi Distribuire l'applicazione Laravel 5 su Shared Hosting su Linux Server online: https://riptutorial.com/it/laravel/topic/2410/distribuire-l-applicazione-laravel-5-su-shared-hosting-su-linux-server

Capitolo 17: Eloquente

introduzione

L'Eloquent è un ORM (Object Relational Model) incluso con il Laravel. Implementa il modello di record attivo e viene utilizzato per interagire con i database relazionali.

Osservazioni

Denominazione della tabella

La convenzione prevede l'uso di "snake_case" pluralizzato per i nomi delle tabelle e singolare "StudlyCase" per i nomi dei modelli. Per esempio:

- Una tabella di `cats` avrebbe un modello `Cat`
- Una tabella `jungle_cats` avrebbe un modello `JungleCat`
- Una tabella `users` avrebbe un modello `User`
- Una tabella `people` avrebbe un modello `Person`

Eloquent proverà automaticamente a legare il tuo modello con una tabella che ha il plurale del nome del modello, come indicato sopra.

Tuttavia, è possibile specificare un nome tabella per sovrascrivere la convenzione predefinita.

```
class User extends Model
{
    protected $table = 'customers';
}
```

Examples

introduzione

Eloquent è l' ORM integrato nel framework di Laravel. Consente di interagire con le tabelle del database in modo orientato agli oggetti, utilizzando il pattern ActiveRecord .

Una singola classe di modello di solito si associa ad una singola tabella di database, e possono anche essere definite relazioni di tipi diversi (uno-a-uno , uno-a-molti , molti-a-molti , polimorfico) tra diverse classi di modelli.

Section Making a Model descrive la creazione e la definizione di classi di modelli.

Prima di poter iniziare a utilizzare i modelli Eloquent, assicurati che almeno una connessione al database sia stata configurata nel tuo file di configurazione `config/database.php` .

Per comprendere l'utilizzo di un generatore di query eloquente durante lo sviluppo, è possibile

utilizzare `php artisan ide-helper:generate` command. Ecco il link .

Navigazione sottoargomento

Relazione eloquente

Persistenza

Oltre a leggere i dati con Eloquent, puoi anche usarlo per inserire o aggiornare i dati con il metodo `save()` . Se hai creato una nuova istanza di modello, verrà *inserito* il record; in caso contrario, se si è recuperato un modello dal database e si impostano nuovi valori, questo verrà *aggiornato* .

In questo esempio creiamo un nuovo record `User` :

```
$user = new User();
$user->first_name = 'John';
$user->last_name = 'Doe';
$user->email = 'john.doe@example.com';
$user->password = bcrypt('my_password');
$user->save();
```

È inoltre possibile utilizzare il metodo `create` per popolare i campi utilizzando una matrice di dati:

```
User::create([
    'first_name'=> 'John',
    'last_name' => 'Doe',
    'email'     => 'john.doe@example.com',
    'password'  => bcrypt('changeme'),
]);
```

Quando si utilizza il metodo di creazione, gli attributi devono essere dichiarati nell'array `fillable` all'interno del modello:

```
class User extends Model
{
    protected $fillable = [
         'first_name',
         'last_name',
         'email',
         'password',
    ];
}
```

In alternativa, se si desidera rendere tutti gli attributi assegnabili in massa, è possibile definire la proprietà $ guarded come una matrice vuota:

```
class User extends Model
{
    /**
    * The attributes that aren't mass assignable.
    *
```

```
    * @var array
    */
    protected $guarded = [];
}
```

Ma puoi anche creare un record senza nemmeno modificare l'attributo `fillable` nel tuo modello usando il metodo `forceCreate` piuttosto che `create` metodo

```
User::forceCreate([
    'first_name'=> 'John',
    'last_name' => 'Doe',
    'email'     => 'john.doe@example.com',
    'password'  => bcrypt('changeme'),
]);
```

Di seguito è riportato un esempio di aggiornamento di un modello `User` esistente caricandolo dapprima (utilizzando `find`), modificandolo e quindi salvandolo:

```
$user = User::find(1);
$user->password = bcrypt('my_new_password');
$user->save();
```

Per ottenere lo stesso risultato con una singola chiamata di funzione, è possibile utilizzare il metodo di `update` :

```
$user->update([
    'password' => bcrypt('my_new_password'),
]);
```

I metodi di `create` e `update` rendono molto più semplice l'utilizzo di grandi insiemi di dati rispetto alla necessità di impostare singolarmente ciascuna coppia chiave / valore, come mostrato nei seguenti esempi:

> Notare l'uso `only` e `except` quando si raccolgono i dati della richiesta. È importante specificare le chiavi esatte che si desidera consentire / non consentire l'aggiornamento, altrimenti è possibile che un utente malintenzionato invii ulteriori campi con la loro richiesta e causi aggiornamenti indesiderati.

```
// Updating a user from specific request data
$data = Request::only(['first_name', 'email']);
$user->find(1);
$user->update($data);

// Create a user from specific request data
$data = Request::except(['_token', 'profile_picture', 'profile_name']);
$user->create($data);
```

Eliminazione

È possibile cancellare i dati dopo averli scritti nel database. È possibile eliminare un'istanza del modello se ne è stata recuperata una o specificare le condizioni per i record da eliminare.

Per eliminare un'istanza del modello, recuperarla e chiamare il metodo `delete()` :

```
$user = User::find(1);
$user->delete();
```

In alternativa, è possibile specificare una chiave primaria (o un array di chiavi primarie) dei record che si desidera eliminare tramite il metodo `destroy()` :

```
User::destroy(1);
User::destroy([1, 2, 3]);
```

Puoi anche combinare le query con l'eliminazione:

```
User::where('age', '<', 21)->delete();
```

Questo cancellerà tutti gli utenti che corrispondono alla condizione.

> Nota: Quando si esegue una massa DELETE tramite Eloquente, la `deleting` e `deleted` gli eventi del modello non saranno licenziati per i modelli eliminati. Questo perché i modelli non vengono mai effettivamente recuperati quando si esegue l'istruzione delete.

Cancellazione morbida

Alcune volte non si desidera eliminare definitivamente un record, ma tenerlo in giro per scopi di controllo o di segnalazione. Per questo, Eloquent fornisce funzionalità di *cancellazione soft* .

Per aggiungere funzionalità soft elimina al modello, è necessario importare il tratto di `SoftDeletes` e aggiungerlo alla classe del modello Eloquent:

```
namespace Illuminate\Database\Eloquent\Model;
namespace Illuminate\Database\Eloquent\SoftDeletes;

class User extends Model
{
     use SoftDeletes;
}
```

Quando si elimina un modello, verrà impostato un timestamp su una colonna timestamp `deleted_at` nella tabella per il modello, quindi assicurati di creare prima la colonna `deleted_at` nella tabella. Oppure durante la migrazione dovresti chiamare il metodo `softDeletes()` sul tuo progetto per aggiungere il timestamp `deleted_at` . Esempio:

```
Schema::table('users', function ($table) {
    $table->softDeletes();
});
```

Qualsiasi query ometterà i record eliminati temporaneamente. Puoi forzarli a mostrarli, se lo desideri, usando l'ambito `withTrashed()` :

```
User::withTrashed()->get();
```

Se si desidera consentire agli utenti di *ripristinare* un record dopo l'eliminazione graduale (ad esempio in un'area del tipo di cestino), è possibile utilizzare il metodo `restore()` :

```
$user = User::find(1);
$user->delete();
$user->restore();
```

Per eliminare forzatamente un record usa il metodo `forceDelete()` che rimuoverà veramente il record dal database:

```
$user = User::find(1);
$user->forceDelete();
```

Cambia chiave primaria e data e ora

Per impostazione predefinita, i modelli Eloquent prevedono che la chiave primaria sia denominata `'id'` . Se non è il tuo caso, puoi cambiare il nome della tua chiave primaria specificando la proprietà `$primaryKey` .

```
class Citizen extends Model
{
    protected $primaryKey = 'socialSecurityNo';

    // ...
}
```

Ora, qualsiasi metodo Eloquent che usa la tua chiave primaria (per esempio `find` o `find findOrFail`) userà questo nuovo nome.

Inoltre, Eloquent si aspetta che la chiave primaria sia un intero autoincrementante. Se la chiave primaria non è un numero intero auto-incrementante (ad es. Un GUID), è necessario comunicare a Eloquent aggiornando la proprietà `$incrementing` su `false` :

```
class Citizen extends Model
{
    protected $primaryKey = 'socialSecurityNo';

    public $incrementing = false;

    // ...
}
```

Per impostazione predefinita, Eloquente aspetta `created_at` e `updated_at` colonne di esistere sulle vostre tavole. Se non desideri che queste colonne siano automaticamente gestite da Eloquent, imposta la proprietà `$timestamps` sul tuo modello su false:

```
class Citizen extends Model
{
```

```
    public $timestamps = false;

    // ...
}
```

Se è necessario personalizzare i nomi delle colonne utilizzate per memorizzare i timestamp, è possibile impostare le `CREATED_AT` e `UPDATED_AT` nel modello:

```
class Citizen extends Model
{
    const CREATED_AT = 'date_of_creation';
    const UPDATED_AT = 'date_of_last_update';

    // ...
}
```

Lancia 404 se l'entità non viene trovata

Se si desidera lanciare automaticamente un'eccezione durante la ricerca di un record che non si trova su una modale, è possibile utilizzare entrambi

```
Vehicle::findOrFail(1);
```

o

```
Vehicle::where('make', 'ford')->firstOrFail();
```

Se non viene trovato un record con la chiave primaria di `1` , viene generata una `ModelNotFoundException` . Che è essenzialmente lo stesso della scrittura (vedi sorgente):

```
$vehicle = Vehicle::find($id);

if (!$vehicle) {
    abort(404);
}
```

Modelli di clonazione

Potresti trovarti a dover clonare una riga, magari cambiare alcuni attributi ma hai bisogno di un modo efficace per mantenere le cose ASCIUTTE. Laravel fornisce una sorta di metodo "nascosto" che ti consente di fare questa funzionalità. Sebbene sia completamente non documentato, devi cercarlo attraverso l'API per trovarlo.

Usando `$model->replicate()` puoi facilmente clonare un record

```
$robot = Robot::find(1);
$cloneRobot = $robot->replicate();
// You can add custom attributes here, for example he may want to evolve with an extra arm!
$cloneRobot->arms += 1;
$cloneRobot->save();
```

Quanto sopra troverà un robot con ID 1, quindi lo clonerà.

Leggi Eloquente online: https://riptutorial.com/it/laravel/topic/865/eloquente

Capitolo 18: Eloquente: Accessors & Mutators

introduzione

Gli accessori e i mutatori consentono di formattare i valori degli attributi Eloquenti quando li si recupera o li si imposta sulle istanze del modello. Ad esempio, è possibile utilizzare il crittografia di Laravel per crittografare un valore mentre è archiviato nel database e quindi decrittografarlo automaticamente quando si accede a un modello Eloquent. Oltre agli accessori e ai mutatori personalizzati, Eloquent può anche eseguire automaticamente il cast dei campi data su istanze di Carbon o persino castare campi di testo su JSON.

Sintassi

- imposta l'attributo {ATTRIBUTE} (attributo $) // nel caso cammello

Examples

Definire un Accessors

```
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class User extends Model
{
    /**
     * Get the user's first name.
     *
     * @param  string  $value
     * @return string
     */
    public function getFirstNameAttribute($value)
    {
        return ucfirst($value);
    }
}
```

Ottenere Accessor:

Come puoi vedere, il valore originale della colonna viene passato all'accessorio, permettendoti di manipolare e restituire il valore. Per accedere al valore dell'accessorio, puoi semplicemente accedere all'attributo `first_name` su un'istanza del modello:

```
$user = App\User::find(1);
$firstName = $user->first_name;
```

Definire un Mutatore

```
class User extends Model
{
    public function setPasswordAttribute($password)
    {
        $this->attributes['password'] = bcrypt($password);
    }
    ...
}
```

Sopra il codice fa "criptare" ogni volta che viene impostata la proprietà della password.

```
$user = $users->first();
$user->password = 'white rabbit'; //laravel calls mutator on background
$user->save(); // password is bcrypted and one does not need to call bcrypt('white rabbit')
```

Leggi Eloquente: Accessors & Mutators online:
https://riptutorial.com/it/laravel/topic/8305/eloquente--accessors--amp--mutators

Capitolo 19: Eloquente: modello

Examples

Fare un modello

Creazione del modello

Le classi di modelli devono estendere `Illuminate\Database\Eloquent\Model` . La posizione predefinita per i modelli è la directory `/app` .

Una classe modello può essere facilmente generata dal comando Artisan :

```
php artisan make:model [ModelName]
```

Questo creerà un nuovo file PHP in `app/` per impostazione predefinita, denominato `[ModelName].php` , e conterrà tutto lo standard per il nuovo modello, che include la classe, lo spazio dei nomi e l'uso richiesto per una configurazione di base.

Se si desidera creare un file di migrazione insieme al proprio modello, utilizzare il seguente comando, dove `-m` genererà anche il file di migrazione:

```
php artisan make:model [ModelName] -m
```

Oltre a creare il modello, crea una migrazione del database collegata al modello. Il file PHP di migrazione del database si trova per impostazione predefinita nel `database/migrations/` . Questo, per impostazione predefinita, non include altro che le colonne id e `created_at` / `updated_at` , quindi dovrai modificare il file per fornire colonne aggiuntive.

Nota che dovrai eseguire la migrazione (una volta impostato il file di migrazione) affinché il modello inizi a funzionare usando `php artisan migrate` dalla radice del progetto

Inoltre, se desideri aggiungere una migrazione in un secondo momento, dopo aver creato il modello, puoi farlo eseguendo:

```
php artisan make:migration [migration name]
```

Supponiamo ad esempio di voler creare un modello per i tuoi gatti, avresti due possibilità, da creare con o senza migrazione. Avresti scelto di creare senza migrazione se avevi già una tabella di gatti o non volevi crearne una in questo momento.

Per questo esempio, vogliamo creare una migrazione perché non abbiamo già una tabella, quindi eseguiremo il seguente comando.

```
php artisan make:model Cat -m
```

Questo comando creerà due file:

1. Nella cartella App: `app/Cat.php`
2. Nella cartella del database: `database/migrations/timestamp_creat_cats_table.php`

Il file a cui siamo interessati è il secondo in quanto è questo file che possiamo decidere su cosa vogliamo che la tabella assomigli e includa. Per ogni migrazione predefinita viene fornita una colonna di identificazione automatica incrementale e una colonna di data / ora.

L'esempio seguente di un estratto del file di migrazione include le colonne predefinite di cui sopra e l'aggiunta di un nome di gatto, età e colore:

```
public function up()
    {
        Schema::create('cats', function (Blueprint $table) {

            $table->increments('id');     //Predefined ID
            $table->string('name');       //Name
            $table->integer('age');       //Age
            $table->string('colour');     //Colour
            $table->timestamps();         //Predefined Timestamps

        });
    }
```

Quindi, come puoi vedere, è relativamente facile creare il modello e la migrazione per un tavolo. Quindi per eseguire la migrazione e crearla nel tuo database devi eseguire il seguente comando:

```
php artisan migrate
```

Che migrerà tutte le migrazioni in sospeso nel tuo database.

Posizione del file del modello

I modelli possono essere archiviati ovunque grazie a PSR4 .

Di default i modelli vengono creati nella directory `app` con lo spazio dei nomi di `App` . Per applicazioni più complesse, in genere è consigliabile archiviare i modelli all'interno delle proprie cartelle in una struttura che abbia senso per l'architettura delle app.

Ad esempio, se hai un'applicazione che utilizza una serie di frutti come modelli, puoi creare una cartella chiamata `app/Fruits` e all'interno di questa cartella creerai `Banana.php` (mantenendo la convenzione di denominazione StudlyCase), potresti quindi creare la classe Banana nello spazio dei nomi `App\Fruits` :

```
namespace App\Fruits;

use Illuminate\Database\Eloquent\Model;

class Banana extends Model {
    // Implementation of "Banana" omitted
```

```
}
```

Configurazione del modello

Eloquent segue un approccio "convenzione sulla configurazione". Estendendo la classe `Model` base, tutti i modelli ereditano le proprietà elencate di seguito. Salvo sovrascrittura, si applicano i seguenti valori predefiniti:

Proprietà	Descrizione	Predefinito
`protected $connection`	Nome della connessione DB	Connessione DB predefinita
`protected $table`	Nome della tabella	Per impostazione predefinita, il nome della classe viene convertito in `snake_case` e pluralizzato. Ad esempio, `SpecialPerson` diventa `special_people`
`protected $primaryKey`	Tabella PK	`id`
`public $incrementing`	Indica se gli ID sono a incremento automatico	`true`
`public $timestamps`	Indica se il modello deve essere timestamp	`true`
`const CREATED_AT`	Nome della colonna timestamp di creazione	`created_at`
`const UPDATED_AT`	Nome della colonna timestamp di modifica	`updated_at`
`protected $dates`	Attributi che devono essere modificati in DateTime, oltre agli attributi di timestamp	`[]`
`protected $dateFormat`	Formato in cui gli attributi della data saranno mantenuti	Predefinito per il dialetto SQL corrente.
`protected $with`	Rapporti da caricare con il modello	`[]`
`protected $hidden`	Attributi omessi nella serializzazione del modello	`[]`
`protected $visible`	Attributi consentiti nella serializzazione del modello	`[]`
`protected $appends`	Accessors di attributi aggiunti	`[]`

Proprietà	Descrizione	Predefinito
	alla serializzazione del modello	
`protected $fillable`	Attributi che sono assegnabili in serie	`[]`
`protected $guarded`	Attributi elencati in nero dall'assegnazione di massa	`[*]` (Tutti gli attributi)
`protected $touches`	Le relazioni che dovrebbero essere toccate al salvataggio	`[]`
`protected $perPage`	Il numero di modelli da restituire per l'impaginazione.	`15`

5.0

Proprietà	Descrizione	Predefinito
`protected $casts`	Attributi che dovrebbero essere convertiti in tipi nativi	`[]`

Aggiorna un modello esistente

```
$user = User::find(1);
$user->name = 'abc';
$user->save();
```

È inoltre possibile aggiornare più attributi contemporaneamente utilizzando l' `update` , che non richiede l'utilizzo del `save` seguito:

```
$user = User::find(1);
$user->update(['name' => 'abc', 'location' => 'xyz']);
```

È inoltre possibile aggiornare un modello (i) senza prima averlo richiesto:

```
User::where('id', '>', 2)->update(['location' => 'xyz']);
```

Se non si desidera attivare una modifica al timestamp `updated_at` sul modello, è possibile passare l'opzione `touch` :

```
$user = User::find(1);
$user->update(['name' => 'abc', 'location' => 'xyz'], ['touch' => false]);
```

Leggi Eloquente: modello online: https://riptutorial.com/it/laravel/topic/7984/eloquente--modello

Capitolo 20: Eloquente: relazione

Examples

Interrogare sulle relazioni

Eloquent ti consente anche di eseguire query su relazioni definite, come mostrato di seguito:

```
User::whereHas('articles', function (Builder $query) {
    $query->where('published', '!=', true);
})->get();
```

Ciò richiede che il nome del tuo metodo di relazione sia `articles` in questo caso. L'argomento passato alla chiusura è il Query Builder per il modello correlato, quindi puoi utilizzare qualsiasi query qui che puoi altrove.

Carico

Supponiamo che il modello utente abbia una relazione con il modello di articolo e che tu voglia caricare gli articoli correlati. Ciò significa che gli articoli dell'utente verranno caricati durante il recupero dell'utente.

`articles` è il nome della relazione (metodo) nel modello Utente.

```
User::with('articles')->get();
```

se hai più relazioni. per esempio articoli e post.

```
User::with('articles','posts')->get();
```

e per selezionare le relazioni annidate

```
User::with('posts.comments')->get();
```

Chiama più di una relazione annidata

```
User::with('posts.comments.likes')->get()
```

Inserimento di modelli correlati

Supponiamo che tu abbia un modello `Post` con una relazione `hasMany` con `Comment` . Puoi inserire un oggetto `Comment` relativo a un post procedendo come segue:

```
$post = Post::find(1);

$commentToAdd = new Comment(['message' => 'This is a comment.']);
```

```
$post->comments()->save($commentToAdd);
```

Puoi salvare più modelli contemporaneamente usando la funzione `saveMany` :

```
$post = Post::find(1);

$post->comments()->saveMany([
    new Comment(['message' => 'This a new comment']),
    new Comment(['message' => 'Me too!']),
    new Comment(['message' => 'Eloquent is awesome!'])
]);
```

In alternativa, esiste anche un metodo `create` che accetta un array PHP semplice invece di un'istanza di modello Eloquent.

```
$post = Post::find(1);

$post->comments()->create([
    'message' => 'This is a new comment message'
]);
```

introduzione

Le relazioni eloquenti sono definite come funzioni nelle classi del modello Eloquent. Poiché, come gli stessi modelli Eloquent, le relazioni fungono anche da potenti costruttori di query, definendo le relazioni come funzioni che forniscono potenti funzionalità di concatenamento e interrogazione dei metodi. Ad esempio, possiamo aggiungere ulteriori vincoli alla relazione di questo post:

```
$user->posts()->where('active', 1)->get();
```

Passa all'argomento principale

Tipi di relazione

Uno a molti

Diciamo che ogni post può avere uno o più commenti e ogni commento appartiene a un solo post.

quindi la tabella dei commenti avrà `post_id` . In questo caso le relazioni saranno le seguenti.

Post Model

```
public function comments()
{
   return $this->belongsTo(Post::class);
}
```

Se la chiave esterna è diversa da `post_id` , ad esempio la chiave esterna è `example_post_id` .

```
public function comments()
{
   return $this->belongsTo(Post::class, 'example_post_id');
}
```

e inoltre, se la chiave locale è diversa da `id` , ad esempio la chiave locale è `other_id`

```
public function comments()
{
   return $this->belongsTo(Post::class, 'example_post_id', 'other_id');
}
```

Commenta il modello

definendo l'inverso di uno a molti

```
public function post()
{
   return $this->hasMany(Comment::class);
}
```

Uno a uno

Come associare tra due modelli (esempio: modello `User` e `Phone`)

`App\User`

```
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class User extends Model
{
    /**
     * Get the phone record associated with the user.
     */
    public function phone()
    {
        return $this->hasOne('Phone::class', 'foreign_key', 'local_key');
    }
}
```

`App\Phone`

```
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Phone extends Model
{
```

```
    /**
     * Get the user that owns the phone.
     */
    public function user()
    {
        return $this->belongsTo('User::class', 'foreign_key', 'local_key');
    }
}
```

`foreign_key` : Per default eloquenti assumerà questo valore da `other_model_name_id` (in questo caso `user_id` e `phone_id`), cambiarlo se non è il caso.

`local_key` : per impostazione predefinita, Eloquent assume che questo valore sia `id` (chiave primaria del modello corrente), modificarlo se non è il caso.

> Se il nome del database è archiviato secondo lo standard laravel, non è necessario fornire la chiave esterna e la chiave locale nella dichiarazione della relazione

Spiegazione

Molti a molti

Diciamo che ci sono ruoli e permessi. Ogni ruolo può appartenere a molte autorizzazioni e ogni autorizzazione può appartenere a molti ruoli. quindi ci saranno 3 tavoli. due modelli e un tavolo pivot. `roles` , `users` e tabella `permission_role` .

Modello di ruolo

```
public function permissions()
{
   return $this->belongsToMany(Permission::class);
}
```

Modello di autorizzazione

```
public function roles()
{
   return $this->belongsToMany(Roles::class);
}
```

Nota 1

considerare di seguire mentre si utilizza un nome di tabella diverso per la tabella pivot.

Supponiamo che si desideri utilizzare `role_permission` anziché `permission_role` , poiché eloquent utilizza l'ordine alfabetico per la costruzione dei nomi delle chiavi pivot. sarà necessario passare il nome della tabella pivot come secondo parametro come segue.

Modello di ruolo

```
public function permissions()
{
   return $this->belongsToMany(Permission::class, 'role_permission');
}
```

Modello di autorizzazione

```
public function roles()
{
   return $this->belongsToMany(Roles::class, 'role_permission');
}
```

Nota 2

considerare di seguire durante l'utilizzo di nomi di chiavi diversi nella tabella pivot.

Eloquent presuppone che se non vengono passati tasti come terzo e quarto parametro, saranno i nomi di tabella singolari con `_id` . quindi suppone che il pivot avrà i campi `role_id` e `permission_id` . Se si devono usare chiavi diverse da queste, dovrebbe essere passato come terzo e quarto parametro.

Diciamo se `other_role_id` essere usato `other_role_id` invece di `role_id` e `other_permission_id` invece di `permission_id` . Quindi sarebbe come segue.

Modello di ruolo

```
public function permissions()
{
   return $this->belongsToMany(Permission::class, 'role_permission', 'other_role_id',
'other_permission_id');
}
```

Modello di autorizzazione

```
public function roles()
{
   return $this->belongsToMany(Roles::class, 'role_permission', 'other_permission_id',
'other_role_id');
}
```

polimorfo

Le relazioni polimorfiche consentono a un modello di appartenere a più di un altro modello su una singola associazione. Un buon esempio potrebbero essere le immagini, sia un utente che un prodotto possono avere un'immagine. La struttura della tabella potrebbe avere il seguente aspetto:

```
user
    id - integer
    name - string
    email - string
```

```
product
    id - integer
    title - string
    SKU - string

image
    id - integer
    url - string
    imageable_id - integer
    imageable_type - string
```

Le colonne importanti da guardare sono nella tabella delle immagini. La colonna `imageable_id` conterrà il valore ID dell'utente o del prodotto, mentre la colonna `imageable_type` conterrà il nome classe del modello proprietario. Nei tuoi modelli, hai impostato le relazioni come segue:

```
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Image extends Model
{
    /**
     * Get all of the owning imageable models.
     */
    public function imageable()
    {
        return $this->morphTo();
    }
}

class User extends Model
{
    /**
     * Get all of the user's images.
     */
    public function images()
    {
        return $this->morphMany('Image::class', 'imageable');
    }
}

class Product extends Model
{
    /**
     * Get all of the product's images.
     */
    public function images()
    {
        return $this->morphMany('Image::class', 'imageable');
    }
}
```

Puoi anche recuperare il proprietario di una relazione polimorfica dal modello polimorfico accedendo al nome del metodo che esegue la chiamata a `morphTo` . Nel nostro caso, questo è il metodo `imageable` sul modello Image. Quindi, accederemo a tale metodo come una proprietà

dinamica

```
$image = App\Image::find(1);

$imageable = $image->imageable;
```

Questo `imageable` restituirà un utente o un prodotto.

Molti a molti

Diciamo che ci sono ruoli e permessi. Ogni ruolo può appartenere a molte autorizzazioni e ogni autorizzazione può appartenere a molti ruoli. quindi ci saranno 3 tavoli. due modelli e un tavolo pivot. `roles` , `users` e tabella `permission_role` .

Modello di ruolo

```
public function permissions()
{
   return $this->belongsToMany(Permission::class);
}
```

Modello di autorizzazione

```
public function roles()
{
   return $this->belongsToMany(Roles::class);
}
```

Nota 1

considerare di seguire mentre si utilizza un nome di tabella diverso per la tabella pivot.

Supponiamo che si desideri utilizzare `role_permission` anziché `permission_role` , poiché eloquent utilizza l'ordine alfabetico per la costruzione dei nomi delle chiavi pivot. sarà necessario passare il nome della tabella pivot come secondo parametro come segue.

Modello di ruolo

```
public function permissions()
{
   return $this->belongsToMany(Permission::class, 'role_permission');
}
```

Modello di autorizzazione

```
public function roles()
{
   return $this->belongsToMany(Roles::class, 'role_permission');
}
```

Nota 2

considerare di seguire durante l'utilizzo di nomi di chiavi diversi nella tabella pivot.

Eloquent presuppone che se non vengono passati tasti come terzo e quarto parametro, saranno i nomi di tabella singolari con `_id` . quindi suppone che il pivot avrà i campi `role_id` e `permission_id` . Se si devono usare chiavi diverse da queste, dovrebbe essere passato come terzo e quarto parametro.

Diciamo se `other_role_id` essere usato `other_role_id` invece di `role_id` e `other_permission_id` invece di `permission_id` . Quindi sarebbe come segue.

Modello di ruolo

```
public function permissions()
{
   return $this->belongsToMany(Permission::class, 'role_permission', 'other_role_id',
'other_permission_id');
}
```

Modello di autorizzazione

```
public function roles()
{
   return $this->belongsToMany(Roles::class, 'role_permission', 'other_permission_id',
'other_role_id');
}
```

Accesso alla tabella intermedia utilizzando withPivot ()

Supponiamo di avere una terza colonna ' **permission_assigned_date** ' nella tabella pivot. Per impostazione predefinita, solo le chiavi del modello saranno presenti sull'oggetto pivot. Ora per ottenere questa colonna nel risultato della query è necessario aggiungere il nome con la funzione Pivot ().

```
   public function permissions()
        {
           return $this->belongsToMany(Permission::class, 'role_permission', 'other_role_id',
'other_permission_id')->withPivot('permission_assigned_date');
        }
```

Attaccare / staccare

Eloquent fornisce anche alcuni metodi di supporto aggiuntivi per rendere più conveniente il lavoro con i modelli correlati. Ad esempio, immaginiamo che un utente possa avere molti ruoli e che un ruolo possa avere molte autorizzazioni. Per assegnare un ruolo a un'autorizzazione inserendo un record nella tabella intermedia che unisce i modelli, utilizzare il metodo attach:

```
$role= App\Role::find(1);
$role->permissions()->attach($permissionId);
```

Quando si collega una relazione a un modello, è possibile anche passare una serie di dati aggiuntivi da inserire nella tabella intermedia:

```
$rol->roles()->attach($permissionId, ['permission_assigned_date' => $date]);
```

Allo stesso modo, per rimuovere un'autorizzazione specifica contro un ruolo, utilizzare la funzione di scollegamento

```
$role= App\Role::find(1);
//will remove permission 1,2,3 against role 1
$role->permissions()->detach([1, 2, 3]);
```

Sincronizzazione delle associazioni

Puoi anche utilizzare il metodo di sincronizzazione per costruire associazioni molti a molti. Il metodo di sincronizzazione accetta un array di ID da inserire nella tabella intermedia. Tutti gli ID che non si trovano nell'array specificato verranno rimossi dalla tabella intermedia. Quindi, una volta completata questa operazione, nella tabella intermedia saranno presenti solo gli ID presenti nell'array specificato:

```
//will keep permission id's 1,2,3 against Role id 1

$role= App\Role::find(1)
$role->permissions()->sync([1, 2, 3]);
```

Leggi Eloquente: relazione online: https://riptutorial.com/it/laravel/topic/7960/eloquente--relazione

Capitolo 21: Errore di mancata corrispondenza del token in AJAX

introduzione

Ho analizzato che il rapporto di ottenere l'errore di TokenMismatch è molto alto. E questo errore si verifica a causa di alcuni errori stupidi. Ci sono molte ragioni per cui gli sviluppatori stanno commettendo errori. Ecco alcuni esempi, ad esempio No _token sulle intestazioni, No _token ha passato i dati quando si utilizza Ajax, il problema dei permessi sul percorso di archiviazione, un percorso di archiviazione delle sessioni non valido.

Examples

Token di installazione sull'intestazione

Imposta il token su `<head>` del tuo `default.blade.php` .

```
<meta name="csrf-token" content="{{csrf_token()}}">
```

Aggiungi `ajaxSetup` nella parte superiore del tuo script, che sarà accessibile ovunque. Questo imposterà le intestazioni su ogni chiamata `ajax`

```
$.ajaxSetup({
    headers: {
        'X-CSRF-TOKEN': $('meta[name="csrf-token"]').attr('content')
    }
});
```

Imposta il token etichetta

Aggiungi sotto la funzione al tuo tag `<form>` . Questa funzione genererà un campo nascosto chiamato `_token` e riempirà il valore con il token.

```
{{csrf_field()}}
```

Aggiungi la funzione `csrf_token ()` al tuo `_token` nascosto `_token` value. Questo genererà solo la stringa crittografata.

```
<input type="hidden" name="_token" value="{{csrf_token()}}"/> .
```

Verifica il percorso di archiviazione della sessione e l'autorizzazione

Qui presumo che l'URL dell'app progetto sia `APP_URL=http://project.dev/ts/toys-store`

1. Impostare la autorizzazione scrivibile su percorso_dellattazione `storage_path('framework/sessions')` della cartella.
2. Controlla il percorso del tuo progetto laravel `'path' => '/ts/toys-store',` la radice del tuo progetto laravel.
3. Cambia il nome del tuo cookie `'cookie' => 'toys-store',`

```
return [
    'driver' => env('SESSION_DRIVER', 'file'),
    'lifetime' => 120,
    'expire_on_close' => false,
    'encrypt' => false,
    'files' => storage_path('framework/sessions'),
    'connection' => null,
    'table' => 'sessions',
    'lottery' => [2, 100],
    'cookie' => 'toys-store',
    'path' => '/ts/toys-store',
    'domain' => null,
    'secure' => false,
    'http_only' => true,
];
```

Usa il campo _token su Ajax

Esistono molti modi per inviare `_token` alla chiamata AJAX

1. Ottieni tutto il valore del campo di input all'interno del tag `<form>` utilizzando `var formData = new FormData($("#cart-add")[0]);`
2. Usa `$("form").serialize();` o `$("form").serializeArray();`
3. Aggiungi `_token` manualmente sui `data` di Ajax. usando `$('meta[name="csrf-token"]').attr('content')` o `$('input[name="_token"]').val()` .
4. Possiamo impostare come header su una particolare chiamata Ajax come sotto il codice.

```
$.ajax({
    url: $("#category-add").attr("action"),
    type: "POST",
    data: formData,
    processData: false,
    contentType: false,
    dataType: "json",
    headers: {
        'X-CSRF-TOKEN': $('meta[name="csrf-token"]').attr('content')
    }
});
```

Leggi Errore di mancata corrispondenza del token in AJAX online:
https://riptutorial.com/it/laravel/topic/10656/errore-di-mancata-corrispondenza-del-token-in-ajax

Capitolo 22: Eventi e ascoltatori

Examples

Utilizzo di eventi e ascoltatori per l'invio di e-mail a un nuovo utente registrato

Gli eventi di Laravel permettono di implementare il modello di Osservatore. Questo può essere usato per inviare una email di benvenuto ad un utente ogni volta che si registrano sulla tua applicazione.

Nuovi eventi e ascoltatori possono essere generati utilizzando l'utilità della riga di comando artisan dopo aver registrato l'evento e il loro particolare listener nella classe `App\Providers\EventServiceProvider` .

```
protected $listen = [
    'App\Events\NewUserRegistered' => [
        'App\Listeners\SendWelcomeEmail',
    ],
];
```

Notazione alternativa:

```
protected $listen = [
    \App\Events\NewUserRegistered::class => [
        \App\Listeners\SendWelcomeEmail::class,
    ],
];
```

Ora esegui `php artisan generate:event` . Questo comando genererà tutti gli eventi e gli ascoltatori corrispondenti menzionati sopra rispettivamente nelle directory `App\Events` e `App\Listeners` .

Possiamo avere più ascoltatori per un singolo evento come

```
protected $listen = [
    'Event' => [
        'Listner1', 'Listener2'
    ],
];
```

`NewUserRegistered` è solo una classe wrapper per il modello Utente appena registrato:

```
class NewUserRegistered extends Event
{
    use SerializesModels;

    public $user;

    /**
     * Create a new event instance.
     *
```

```
     * @return void
     */
    public function __construct(User $user)
    {
        $this->user = $user;
    }
}
```

Questo `Event` sarà gestito dal listener `SendWelcomeEmail` :

```
class SendWelcomeEmail
{
    /**
     * Handle the event.
     *
     * @param  NewUserRegistered  $event
     */
    public function handle(NewUserRegistered $event)
    {
        //send the welcome email to the user
        $user = $event->user;
        Mail::send('emails.welcome', ['user' => $user], function ($message) use ($user) {
                $message->from('hi@yourdomain.com', 'John Doe');
                $message->subject('Welcome aboard '.$user->name.'!');
                $message->to($user->email);
        });
    }
}
```

L'ultimo passaggio consiste nel chiamare / attivare l'evento ogni volta che si registra un nuovo utente. Questo può essere fatto nel controller, nel comando o nel servizio, ovunque sia implementata la logica di registrazione dell'utente:

```
event(new NewUserRegistered($user));
```

Leggi Eventi e ascoltatori online: https://riptutorial.com/it/laravel/topic/4687/eventi-e-ascoltatori

Capitolo 23: Filesystem / Cloud Storage

Examples

Configurazione

Il file di configurazione del filesystem si trova in `config/filesystems.php` . All'interno di questo file puoi configurare tutti i tuoi "dischi". Ogni disco rappresenta un particolare driver di archiviazione e posizione di archiviazione. Configurazioni di esempio per ciascun driver supportato sono incluse nel file di configurazione. Quindi, modifica semplicemente la configurazione per riflettere le tue preferenze e credenziali di archiviazione!

Prima di utilizzare i driver S3 o Rackspace, è necessario installare il pacchetto appropriato tramite Composer:

- Amazon S3: `league/flysystem-aws-s3-v2 ~1.0`
- Rackspace: `league/flysystem-rackspace ~1.0`

Naturalmente, è possibile configurare quanti dischi si desidera e possono anche avere più dischi che utilizzano lo stesso driver.

Quando si utilizza il driver locale, si noti che tutte le operazioni sui file sono relative alla directory root definita nel file di configurazione. Per impostazione predefinita, questo valore è impostato sulla `storage/app directory` . Pertanto, il seguente metodo memorizzerebbe un file in `storage/app/file.txt` :

```
Storage::disk('local')->put('file.txt', 'Contents');
```

Uso di base

La facciata di `Storage` può essere utilizzata per interagire con qualsiasi disco configurato. In alternativa, è possibile digitare -intensificare il `Illuminate\Contracts\Filesystem\Factory` su qualsiasi classe che viene risolta tramite il contenitore del servizio Laravel.

Recupero di un disco particolare

```
$disk = Storage::disk('s3');

$disk = Storage::disk('local');
```

Determinare se esiste un file

```
$exists = Storage::disk('s3')->exists('file.jpg');
```

Metodi di chiamata sul disco predefinito

```
if (Storage::exists('file.jpg'))
{
    //
}
```

Recupero del contenuto di un file

```
$contents = Storage::get('file.jpg');
```

Impostazione del contenuto di un file

```
Storage::put('file.jpg', $contents);
```

Prepend to A File

```
Storage::prepend('file.log', 'Prepended Text');
```

Aggiungi a un file

```
Storage::append('file.log', 'Appended Text');
```

Elimina un file

```
Storage::delete('file.jpg');

Storage::delete(['file1.jpg', 'file2.jpg']);
```

Copia un file in una nuova posizione

```
Storage::copy('old/file1.jpg', 'new/file1.jpg');
```

Sposta un file in una nuova posizione

```
Storage::move('old/file1.jpg', 'new/file1.jpg');
```

Ottieni dimensioni file

```
$size = Storage::size('file1.jpg');
```

Ottieni l'ultimo tempo di modifica (UNIX)

```
$time = Storage::lastModified('file1.jpg');
```

Ottieni tutti i file in una directory

```
$files = Storage::files($directory);

// Recursive...
```

```
$files = Storage::allFiles($directory);
```

Ottieni tutte le directory in una directory

```
$directories = Storage::directories($directory);

// Recursive...
$directories = Storage::allDirectories($directory);
```

Crea una directory

```
Storage::makeDirectory($directory);
```

Elimina una directory

```
Storage::deleteDirectory($directory);
```

File system personalizzati

L'integrazione di Laravel Flysystem fornisce driver per diversi "driver" pronti all'uso; tuttavia, Flysystem non è limitato a questi e ha adattatori per molti altri sistemi di archiviazione. È possibile creare un driver personalizzato se si desidera utilizzare uno di questi adattatori aggiuntivi nell'applicazione Laravel. Non preoccuparti, non è troppo difficile!

Per configurare il filesystem personalizzato è necessario creare un fornitore di servizi come `DropboxFilesystemServiceProvider` . Nel metodo di `boot` del provider, è possibile iniettare un'istanza del `Illuminate\Contracts\Filesystem\Factory` e chiamare il metodo di `extend` dell'istanza iniettata. In alternativa, è possibile utilizzare il metodo di `extend` della facciata del `Disk` .

Il primo argomento del metodo `extend` è il nome del driver e il secondo è una chiusura che riceve le variabili `$app` e `$config` . La chiusura del resolver deve restituire un'istanza di `League\Flysystem\Filesystem` .

> Nota: la variabile $ config conterrà già i valori definiti in `config/filesystems.php` per il disco specificato. Esempio di Dropbox

```
<?php namespace App\Providers;

use Storage;
use League\Flysystem\Filesystem;
use Dropbox\Client as DropboxClient;
use League\Flysystem\Dropbox\DropboxAdapter;
use Illuminate\Support\ServiceProvider;

class DropboxFilesystemServiceProvider extends ServiceProvider {

    public function boot()
    {
        Storage::extend('dropbox', function($app, $config)
        {
            $client = new DropboxClient($config['accessToken'], $config['clientIdentifier']);
```

```
            return new Filesystem(new DropboxAdapter($client));
        });
    }

    public function register()
    {
        //
    }

}
```

Creazione di collegamenti simbolici in un server Web tramite SSH

Nella documentazione di Laravel, un collegamento simbolico (link simbolico o soft link) da pubblico / storage a storage / app / public dovrebbe essere creato per rendere i file accessibili dal web.

(QUESTA PROCEDURA CREERÀ IL LINK SIMBOLICO NELL'ELENCO DEL PROGETTO LARAVEL)

Ecco i passaggi su come creare un collegamento simbolico nel tuo server web Linux usando il client SSH:

1. Collegati e accedi al tuo server web usando il client SSH (es. PUTTY).
2. Collega **memoria / app / pubblico** a **pubblico / archiviazione** usando la sintassi

   ```
   ln -s target_path link_path
   ```

   Esempio (nella directory dei file di CPanel)

   ```
   ln -s /home/cpanel_username/project_name/storage/app/public
   /home/cpanel_sername/project_name/public/storage
   ```

*(Verrà creata una cartella denominata **memoria** per collegare il percorso con un indicatore **>>>** sull'icona della cartella.)*

Leggi Filesystem / Cloud Storage online: https://riptutorial.com/it/laravel/topic/3040/filesystem---cloud-storage

Capitolo 24: Funzione Helper personalizzata

introduzione

L'aggiunta di helper personalizzati può aiutarti con la tua velocità di sviluppo. Ci sono alcune cose da tenere in considerazione durante la scrittura di tali funzioni di supporto, quindi, questo tutorial.

Osservazioni

Solo alcuni suggerimenti:

- Abbiamo inserito le definizioni delle funzioni all'interno di un controllo (`function_exists`) per evitare eccezioni quando il provider di servizi viene chiamato due volte.
- Un modo alternativo è la registrazione del file helpers dal file `composer.json` . È possibile copiare la logica dal framework laravel stesso .

Examples

document.php

```
<?php

if (!function_exists('document')) {
    function document($text = '') {
        return $text;
    }
}
```

Crea un file helper.php, supponiamo per ora che risieda in `app/Helpers/document.php` . Puoi mettere molti helper in un file (questo è come lo fa Laravel) o puoi dividerli per nome.

HelpersServiceProvider.php

Ora creiamo un fornitore di servizi. Mettiamolo sotto `app/Providers` :

```
<?php

namespace App\Providers;

class HelpersServiceProvider extends ServiceProvider
{
    public function register()
    {
        require_once __DIR__ . '/../Helpers/document.php';
    }
}
```

Il fornitore di servizi sopra riportato carica il file degli helper e registra automaticamente la tua

funzione personalizzata. Assicurati di registrare questo HelpersServiceProvider nel tuo `config/app.php` sotto i `providers` :

```
'providers' => [
     // [..] other providers
     App\Providers\HelpersServiceProvider::class,
]
```

Uso

Ora puoi usare la funzione `document()` ovunque nel tuo codice, ad esempio nei template blade. Questo esempio restituisce solo la stessa stringa che riceve come argomento

```
<?php
Route::get('document/{text}', function($text) {
    return document($text);
});
```

Ora vai su `/document/foo` nel tuo browser (usa `php artisan serve` o `valet`), che restituirà `foo` .

Leggi Funzione Helper personalizzata online: https://riptutorial.com/it/laravel/topic/8347/funzione-helper-personalizzata

Capitolo 25: Gestione degli errori

Osservazioni

Ricordarsi di impostare la propria applicazione per l'invio di e-mail assicurando la corretta configurazione di `config/mail.php`

Verificare inoltre che le variabili ENV siano impostate correttamente.

Questo esempio è una guida ed è minimo. Esplora, modifica e modella la vista come desideri. Modifica il codice per soddisfare le tue esigenze. Ad esempio, imposta il destinatario nel tuo file *.env*

Examples

Invia messaggio di errore

Le eccezioni in Laravel sono gestite da *App \ Exceptions \ Handler.php*

Questo file contiene due funzioni per impostazione predefinita. Segnala e Renderizza. Useremo solo il primo

```
public function report(Exception $e)
```

> Il metodo del report viene utilizzato per registrare le eccezioni o inviarle a un servizio esterno come BugSnag. Per impostazione predefinita, il metodo del report passa semplicemente l'eccezione alla classe base in cui viene registrata l'eccezione. Tuttavia, sei libero di registrare eccezioni come desideri.

Essenzialmente questa funzione inoltra semplicemente l'errore e non fa nulla. Pertanto, possiamo inserire la logica di business per eseguire operazioni basate sull'errore. Per questo esempio, invieremo un'email contenente le informazioni sull'errore.

```
public function report(Exception $e)
{
    if ($e instanceof \Exception) {
        // Fetch the error information we would like to
        // send to the view for emailing
        $error['file']    = $e->getFile();
        $error['code']    = $e->getCode();
        $error['line']    = $e->getLine();
        $error['message'] = $e->getMessage();
        $error['trace']   = $e->getTrace();

        // Only send email reports on production server
        if(ENV('APP_ENV') == "production"){
            #1. Queue email for sending on "exceptions_emails" queue
            #2. Use the emails.exception_notif view shown below
            #3. Pass the error array to the view as variable $e
```

```
            Mail::queueOn('exception_emails', 'emails.exception_notif', ["e" => $error],
function ($m) {
                $m->subject("Laravel Error");
                $m->from(ENV("MAIL_FROM"), ENV("MAIL_NAME"));
                $m->to("webmaster@laravelapp.com", "Webmaster");
            });

        }
    }

    // Pass the error on to continue processing
    return parent::report($e);
}
```

La vista per l'email ("emails.exception_notif") è sotto

```
<?php
$action = (\Route::getCurrentRoute()) ? \Route::getCurrentRoute()->getActionName() : "n/a";
$path = (\Route::getCurrentRoute()) ? \Route::getCurrentRoute()->getPath() : "n/a";
$user = (\Auth::check()) ? \Auth::user()->name : 'no login';
?>

There was an error in your Laravel App<br />

<hr />
<table border="1" width="100%">
    <tr><th >User:</th><td>{{ $user }}</td></tr>
    <tr><th >Message:</th><td>{{ $e['message'] }}</td></tr>
    <tr><th >Action:</th><td>{{ $action }}</td></tr>
    <tr><th >URI:</th><td>{{ $path }}</td></tr>
    <tr><th >Line:</th><td>{{ $e['line'] }}</td></tr>
    <tr><th >Code:</th><td>{{ $e['code'] }}</td></tr>
</table>
```

Catching application Wide ModelNotFoundException

app \ Eccezioni \ Handler.php

```
public function render($request, Exception $exception)
{
    if ($exception instanceof ModelNotFoundException) {
        abort(404);
    }

    return parent::render($request, $exception);
}
```

Puoi prendere / gestire qualsiasi eccezione che viene lanciata in Laravel.

Leggi Gestione degli errori online: https://riptutorial.com/it/laravel/topic/2858/gestione-degli-errori

Capitolo 26: Guida d'installazione

Osservazioni

Questa sezione fornisce una panoramica di ciò che laravel-5.4 è, e perché uno sviluppatore potrebbe voler usarlo.

Dovrebbe anche menzionare tutti i soggetti di grandi dimensioni all'interno di laravel-5.4 e collegarsi agli argomenti correlati. Poiché la documentazione di laravel-5.4 è nuova, potrebbe essere necessario creare versioni iniziali di tali argomenti correlati.

Examples

Installazione

Istruzioni dettagliate su come installare o installare laravel.

il compositore è richiesto per installare facilmente laravel.

Ci sono 3 metodi per installare laravel nel tuo sistema:

1. **Via Laravel Installer**

   Scarica l'installer di Laravel usando il `composer`

   ```
   composer global require "laravel/installer"
   ```

   Prima di usare il compositore dobbiamo aggiungere `~/.composer/vendor/bin` a `PATH` . Una volta terminata l'installazione, possiamo usare il comando `laravel new` per creare un nuovo progetto in `Laravel` .

   Esempio:

   ```
   laravel new {folder name}
   ```

   Questo comando crea una nuova directory denominata come `site` e una nuova installazione di `Laravel` con tutte le altre dipendenze installate nella directory.

2. **Tramite Composer Create-Project**

   Puoi usare il comando nel `terminal` per creare una nuova `Laravel app` :

   ```
   composer create-project laravel/laravel {folder name}
   ```

3. **Tramite download**

Scarica Laravel e decomprimilo.

1. `composer install`
2. Copia `.env.example` in `.env` via `teminal` o manualmente.

```
cp .env.example .env
```

3. Apri il file `.env` e imposta il tuo database, email, pusher, ecc. (Se necessario)
4. `php artisan migrate` (se il database è configurato)
5. `php artisan key:generate`
6. `php artisan serve`
7. Vai a localhost: 8000 per visualizzare il sito

Laravel docs

Hello World Example (Base)

L'accesso alle pagine e l'output dei dati è abbastanza facile in Laravel. Tutti i percorsi delle pagine si trovano in `app/routes.php` . Di solito ci sono alcuni esempi per iniziare, ma creeremo una nuova rotta. Apri la tua `app/routes.php` e incolla il seguente codice:

```
Route::get('helloworld', function () {
    return '<h1>Hello World</h1>';
});
```

Questo dice a Laravel che quando qualcuno accede a `http://localhost/helloworld` in un browser, dovrebbe eseguire la funzione e restituire la stringa fornita.

Hello World Example With Views and Controller

Supponendo di avere un'applicazione funzionante di laravel in esecuzione, ad esempio, "mylaravel.com", vogliamo che la nostra applicazione mostri un messaggio "Hello World" quando raggiungiamo l'URL `http://mylaravel.com/helloworld` . Implica la creazione di due file (la vista e il controller) e la modifica di un file esistente, il router.

La vista

Prima di tutto, apriamo un nuovo file di visualizzazione blade denominato `helloview.blade.php` con la stringa "Hello World". Crealo nell'app directory / risorse / viste

```
<h1>Hello, World</h1>
```

Il controller

Ora creiamo un controller che gestirà la visualizzazione di quella vista con la stringa "Hello World". Useremo artigiano nella riga di comando.

```
$> cd your_laravel_project_root_directory
```

```
$> php artisan make:controller HelloController
```

Questo creerà solo un file (`app/Http/Controllers/HelloController.php`) contenente la classe che è il nostro nuovo controller `HelloController` .

Modifica quel nuovo file e scrivi un nuovo metodo `hello` che visualizzerà la vista che abbiamo creato in precedenza.

```
public function hello()
{
    return view('helloview');
}
```

L'argomento 'helloview' nella funzione di visualizzazione è solo il nome del file di visualizzazione senza il trailing ".blade.php". Laravel saprà come trovarlo.

Ora quando chiamiamo il metodo `hello` del controller `HelloController` verrà visualizzato il messaggio. Ma come lo colleghiamo a una chiamata a `http://mylaravel.com/helloworld` ? Con il passaggio finale, il routing.

Il router

Apri l' `app/routes/web.php` file esistente `app/routes/web.php` (nelle vecchie versioni di laravel `app/Http/routes.php`) e aggiungi questa riga:

```
Route::get('/helloworld', 'HelloController@hello');
```

che è un comando molto esplicativo che dice alla nostra app di laravel: "Quando qualcuno usa il verbo `GET` per accedere a" / helloworld "in questa app di laravel, restituisce i risultati della chiamata alla funzione `hello` nel controller `HelloController` .

Leggi Guida d'installazione online: https://riptutorial.com/it/laravel/topic/2187/guida-d-installazione

Capitolo 27: Helpers

introduzione

Gli helper di Laravel sono le funzioni accessibili globalmente definite dal framework. Può essere chiamato direttamente e utilizzato indipendentemente ovunque all'interno dell'applicazione senza dover istanziare un oggetto o importare una classe.

Ci sono helper per manipolare *array* , *percorsi* , *stringhe* , *URL* , ecc

Examples

Metodi di matrice

array_add ()

Questo metodo viene utilizzato per aggiungere nuove coppie di valori chiave a un array.

```
$array = ['username' => 'testuser'];

$array = array_add($array, 'age', 18);
```

risultato

```
['username' => 'testuser', 'age' => 18]
```

Metodi di stringa

camel_case ()

Questo metodo cambia una stringa in un caso di cammello

```
camel_case('hello_world');
```

risultato

```
HelloWorld
```

Path mehods

I metodi Path facilitano l'accesso ai percorsi relativi alle applicazioni facilmente da qualsiasi luogo.

public_path ()

Questo metodo restituisce il percorso pubblico completo dell'applicazione. che è la directory

pubblica.

```
$path = public_path();
```

urls

url ()

La funzione url genera un URL completo del percorso specificato.

se il tuo sito è `hello.com`

```
echo url('my/dashboard');
```

sarebbe tornato

```
hello.com/my/dashboard
```

se non viene passato nulla al metodo url, viene restituita un'istanza di `Illuminate\Routing\UrlGenerator` e potrebbe essere utilizzata in questo modo

restituirebbe l'url corrente

```
echo url()->current();
```

restituirebbe l'url completo

```
echo url()->full();
```

restituirebbe l'url precedente

```
echo url()->previous();
```

Leggi Helpers online: https://riptutorial.com/it/laravel/topic/8827/helpers

Capitolo 28: HTML e Form Builder

Examples

Installazione

HTML e Form Builder non sono un componente principale da Laravel 5, quindi è necessario installarlo separatamente:

```
composer require laravelcollective/html "~5.0"
```

Infine in `config/app.php` dobbiamo registrare il fornitore di servizi e gli alias delle facciate come questo:

```
'providers' => [
    // ...
    Collective\Html\HtmlServiceProvider::class,
    // ...
],

'aliases' => [
   // ...
  'Form' => Collective\Html\FormFacade::class,
  'Html' => Collective\Html\HtmlFacade::class,
   // ...
],
```

I documenti completi sono disponibili su moduli e HTML

Leggi HTML e Form Builder online: https://riptutorial.com/it/laravel/topic/3672/html-e-form-builder

Capitolo 29: Iniziare con laravel-5.3

Osservazioni

Questa sezione fornisce una panoramica di cosa è laravel-5.3 e perché uno sviluppatore potrebbe volerlo utilizzare.

Dovrebbe anche menzionare eventuali soggetti di grandi dimensioni all'interno di laravel-5.3 e collegarsi agli argomenti correlati. Poiché la documentazione di laravel-5.3 è nuova, potrebbe essere necessario creare versioni iniziali di tali argomenti correlati.

Examples

Installazione di Laravel

Requisiti:
Hai bisogno di `PHP >= 5.6.4` e `Composer` installato sul tuo computer. Puoi controllare la versione di entrambi usando il comando:
Per PHP:

```
php -v
```

Uscita come questa:

```
PHP 7.0.9 (cli) (built: Aug 26 2016 06:17:04) ( NTS )
Copyright (c) 1997-2016 The PHP Group
Zend Engine v3.0.0, Copyright (c) 1998-2016 Zend Technologies
```

Per compositore
È possibile eseguire il comando sul terminale / CMD:

```
composer --version
```

Uscita come questa:

```
composer version 1.2.1 2016-09-12 11:27:19
```

Laravel utilizza Composer per gestire le sue dipendenze. Quindi, prima di usare Laravel, assicurati di aver installato Composer sul tuo computer.

Via Laravel Installer

Per prima cosa, scarica il programma di installazione di Laravel usando Composer:

```
composer global require "laravel/installer"
```

Assicurati di posizionare la `$HOME/.composer/vendor/bin` (o la directory equivalente per il tuo sistema operativo) nel tuo $ PATH in modo che l'eseguibile di `laravel` possa essere localizzato dal tuo sistema.

Una volta installato, il comando `laravel new` creerà una nuova installazione di Laravel nella directory specificata. Ad esempio, il `laravel new blog` creerà una directory denominata `blog` contenente una nuova installazione di Laravel con tutte le dipendenze di Laravel già installate:

```
laravel new blog
```

Tramite Composer Create-Project

In alternativa, è possibile installare Laravel emettendo il comando `create-project` Composer nel proprio terminale:

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

Impostare

Dopo aver completato l'installazione di Laravel, sarà necessario impostare le `permissions` per le cartelle di archiviazione e Bootstrap.

> **Nota: l'** impostazione delle `permissions` è uno dei processi più importanti da completare durante l'installazione di Laravel.

Server di sviluppo locale

Se hai PHP installato localmente e vorresti usare il server di sviluppo integrato di PHP per servire la tua applicazione, puoi usare il comando di `serve` Artisan. Questo comando avvierà un server di sviluppo all'indirizzo `http://localhost:8000` :

```
php artisan serve
```

Apri l'url di richiesta del browser `http://localhost:8000`

Requisiti del server

Il framework Laravel ha alcuni requisiti di sistema. Naturalmente, tutti questi requisiti sono soddisfatti dalla macchina virtuale Laravel Homestead , quindi è altamente raccomandato l'utilizzo di Homestead come ambiente di sviluppo Laravel locale.

Tuttavia, se non si utilizza Homestead, è necessario assicurarsi che il server soddisfi i seguenti requisiti:

- PHP> = 5.6.4

- Estensione PHP OpenSSL
- Estensione PHP PDO
- Estensione PHP di Mbstring
- Estensione PHP Tokenizer
- Estensione PHP XML

Server di sviluppo locale

Se hai PHP installato localmente e vorresti usare il server di sviluppo integrato di PHP per servire la tua applicazione, puoi usare il comando di `serve` Artisan. Questo comando avvierà un server di sviluppo all'indirizzo `http://localhost:8000` :

```
php artisan serve
```

Ovviamente, sono disponibili opzioni di sviluppo locale più robuste tramite Homestead e Valet .

Inoltre è possibile utilizzare una porta personalizzata, qualcosa come `8080` . Puoi farlo con l'opzione `--port` .

```
php artisan serve --port=8080
```

Se hai un dominio locale nel tuo file hosts, puoi impostare il nome host. Questo può essere fatto con l'opzione `--host` .

```
php artisan serve --host=example.dev
```

È anche possibile eseguire su un host e una porta personalizzati, questo può essere fatto con il seguente comando.

```
php artisan serve --host=example.dev --port=8080
```

Ciao World Example (di base) e con l'utilizzo di una vista

L'esempio di base
Aprire il file `routes/web.php` e incollare il seguente codice nel file:

```
Route::get('helloworld', function () {
    return '<h1>Hello World</h1>';
});
```

qui ' **helloworld** ' fungerà da nome della pagina a cui desideri accedere,

e se non si desidera creare un file blade e si desidera comunque accedere direttamente alla pagina, è possibile utilizzare il routing di laravel in questo modo

ora digita `localhost/helloworld` nella barra degli indirizzi del browser e puoi accedere alla pagina che mostra Hello World.

Il prossimo passo.
Quindi hai imparato come creare un Hello World molto semplice! pagina restituendo un ciao frase mondiale. Ma possiamo renderlo un po 'più bello!

Passo 1.
Inizieremo di nuovo sul nostro `routes/web.php` file `routes/web.php` ora invece di utilizzare il codice sopra utilizzeremo il seguente codice:

```
Route::get('helloworld', function() {
    return view('helloworld');
});
```

Il valore di ritorno questa volta non è solo un semplice testo di helloworld ma una vista. Una vista in Laravel è semplicemente un nuovo file. Questo file "helloworld" contiene l'HTML e forse in seguito anche alcuni PHP del testo di Helloworld.

Passo 2.
Ora che abbiamo modificato il nostro percorso per richiamare una vista, faremo la vista. Laravel lavora con i file blade.php nelle viste. Quindi, in questo caso, il nostro percorso si chiama helloworld. Quindi la nostra vista si chiamerà `helloworld.blade.php`

Creeremo il nuovo file nella directory `resources/views` e lo chiameremo helloworld.blade.php

Ora apriremo questo nuovo file e lo modificheremo creando la nostra frase Hello World. Possiamo aggiungere diversi modi per ottenere la nostra frase come nell'esempio qui sotto.

```
<html>
    <body>
        <h1> Hello World! </h1>

        <?php
            echo "Hello PHP World!";
        ?>

    </body>
</html>
```

ora vai al tuo browser e digita di nuovo il percorso come nell'esempio di base: `localhost/helloworld` vedrai la tua nuova vista creata con tutti i contenuti!

Hello World Example (Base)

Apri il file delle rotte. Incolla il seguente codice in:

```
Route::get('helloworld', function () {
    return '<h1>Hello World</h1>';
});
```

dopo aver percorso `http://localhost/helloworld` visualizza Hello World.

Il file delle rotte si trova in `/routes/web.php`

Configurazione del server Web per Pretty URL

Se hai installato `Laravel` tramite `Composer or the Laravel installer` , sotto la configurazione ti servirà.

Configurazione per Apache Laravel include un file `public/.htaccess` che viene utilizzato per fornire URL senza il front controller `index.php` nel percorso. Prima di servire Laravel con Apache, assicurati di abilitare il modulo `mod_rewrite` modo che il file `.htaccess` sia onorato dal server.

Se il file `.htaccess` fornito con Laravel non funziona con l'installazione di Apache, prova questa alternativa:

```
Options +FollowSymLinks
RewriteEngine On

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

Configurazione per Nginx Se stai utilizzando Nginx, la seguente direttiva nella configurazione del tuo sito indirizzerà tutte le richieste al front controller `index.php` :

```
location / {
    try_files $uri $uri/ /index.php?$query_string;
}
```

Naturalmente, quando si utilizza Homestead o Valet , gli URL graziosi verranno automaticamente configurati.

Leggi Iniziare con laravel-5.3 online: https://riptutorial.com/it/laravel/topic/8602/iniziare-con-laravel-5-3

Capitolo 30: Installazione

Examples

Installazione

Le applicazioni Laravel sono installate e gestite con Composer , un popolare gestore di dipendenze PHP. Ci sono due modi per creare una nuova applicazione Laravel.

Via Compositore

```
$ composer create-project laravel/laravel [foldername]
```

O

```
$ composer create-project --prefer-dist laravel/laravel [foldername]
```

Sostituisci **[nomecartella]** con il nome della directory in cui vuoi installare la tua nuova applicazione Laravel. Non deve esistere prima dell'installazione. Potrebbe anche essere necessario aggiungere l'eseguibile del Composer al tuo percorso di sistema.

Se si desidera creare un progetto Laravel utilizzando una versione specifica del framework, è possibile fornire un modello di versione, altrimenti il progetto utilizzerà l'ultima versione disponibile.

Ad esempio, se volessi creare un progetto in Laravel 5.2, eseguiresti:

```
$ composer create-project --prefer-dist laravel/laravel 5.2.*
```

Perché --prefer-dist

Esistono due modi per scaricare un pacchetto: `source` e `dist` . Per le versioni stabili, il compositore utilizzerà la `dist` per impostazione predefinita. L' `source` è un repository di controllo della versione. Se `--prefer-source` è abilitato, Composer installerà dal sorgente se ce n'è uno.

`--prefer-dist` è l'opposto di `--prefer-source` e dice a Composer di installare da `dist` se possibile. Ciò può accelerare le installazioni in modo sostanziale sui server di generazione e in altri casi di utilizzo in cui in genere non vengono eseguiti gli aggiornamenti dei fornitori. Permette anche di evitare problemi con Git se non si dispone di una configurazione corretta.

Tramite l'installazione di Laravel

Laravel fornisce un'utile utility da riga di comando per creare rapidamente le applicazioni Laravel. Innanzitutto, installa l'installer:

```
$ composer global require laravel/installer
```

> Devi assicurarti che la cartella dei binari Composer sia nella variabile $ PATH per eseguire il programma di installazione di Laravel.
>
> Per prima cosa, guarda se è già nella tua variabile $ PATH
>
> ```
> echo $PATH
> ```
>
> Se tutto è corretto, l'output dovrebbe contenere qualcosa come questo:
>
> ```
> Users/yourusername/.composer/vendor/bin
> ```
>
> In caso contrario, modifica il tuo `.bashrc` o, se usi ZSH, il tuo `.zshrc` modo che contenga il percorso della directory del fornitore Composer.

Una volta installato, questo comando creerà una nuova installazione di Laravel nella directory specificata.

```
laravel new [foldername]
```

È inoltre possibile utilizzare `.` (un punto) al posto di **[nomecompa]** per creare il progetto nella directory di lavoro corrente senza creare una sottodirectory.

Esecuzione dell'applicazione

Laravel viene fornito in bundle con un server web basato su PHP che può essere avviato eseguendo

```
$ php artisan serve
```

Per impostazione predefinita, il server HTTP utilizzerà la porta 8000, ma se la porta è già in uso o se si desidera eseguire più applicazioni Laravel contemporaneamente, è possibile utilizzare il flag `--port` per specificare una porta diversa:

```
$ php artisan serve --port=8080
```

Il server HTTP utilizzerà `localhost` come dominio predefinito per l'esecuzione dell'applicazione, ma è possibile utilizzare il flag `--host` per specificare un indirizzo diverso:

```
$ php artisan serve --host=192.168.0.100 --port=8080
```

Utilizzando un server diverso

Se si preferisce utilizzare un software server Web diverso, alcuni file di configurazione vengono forniti all'interno della directory `public` del progetto; `.htaccess` per Apache e `web.config` per ASP.NET. Per altri software come NGINX, è possibile convertire le configurazioni di Apache utilizzando vari strumenti online.

Il framework ha bisogno che l'utente del web server abbia i permessi di scrittura sulle seguenti directory:

- `/storage`
- `/bootstrap/cache`

Su * nix sistemi operativi questo può essere raggiunto da

```
chown -R www-data:www-data storage bootstrap/cache
chmod -R ug+rwx storage bootstrap/cache
```

(dove `www-data` è il nome e il gruppo dell'utente del web server)

Il server Web di tua scelta dovrebbe essere configurato per servire il contenuto dalla directory `/public` del tuo progetto, che di solito viene eseguita impostandola come root del documento. Il resto del tuo progetto non dovrebbe essere accessibile attraverso il tuo server web.

Se imposti correttamente tutto, la navigazione verso l'URL del tuo sito web dovrebbe mostrare la pagina di destinazione predefinita di Laravel.

Requisiti

Il framework Laravel ha i seguenti requisiti:

5.3

- PHP> = 5.6.4
- Estensione PHP XML
- Estensione PHP PDO
- Estensione PHP OpenSSL
- Estensione PHP di Mbstring
- Estensione PHP Tokenizer

5.1 (LTS) 5.2

- PHP> = 5.5.9
- Estensione PHP PDO
- Laravel 5.1 è la prima versione di Laravel a supportare PHP 7.0.

5.0

- PHP> = 5.4, PHP <7
- Estensione PHP OpenSSL
- Estensione PHP Tokenizer
- Estensione PHP di Mbstring
- Estensione PHP JSON (solo su PHP 5.5)

4.2

- PHP> = 5.4
- Estensione PHP di Mbstring
- Estensione PHP JSON (solo su PHP 5.5)

Ciao World Example (Usando Controller e View)

1. Creare un'applicazione Laravel:

```
$ composer create-project laravel/laravel hello-world
```

2. Passare alla cartella del progetto, ad es

```
$ cd C:\xampp\htdocs\hello-world
```

3. Crea un controller:

```
$ php artisan make:controller HelloController --resource
```

Questo creerà l' **app** file **/ Http / Controllers / HelloController.php** . L'opzione `--resource` genererà i metodi CRUD per il controller, ad esempio index, create, show, update.

4. Registra un percorso per il metodo `index` di HelloController. Aggiungi questa linea **all'app / Http / routes.php** *(versione 5.0 a 5.2)* o **routes / web.php** *(versione 5.3)* :

```
Route::get('hello', 'HelloController@index');
```

Per vedere i tuoi percorsi appena aggiunti, puoi eseguire `$ php artisan route:list`

5. Crea un modello Blade nella directory `views` :

Resources / views / hello.blade.php:

```
<h1>Hello world!</h1>
```

6. Ora diciamo al metodo index per visualizzare il template **hello.blade.php** :

App / HTTP / Controller / HelloController.php

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

use App\Http\Requests;

class HelloController extends Controller
{
    /**
```

```
         * Display a listing of the resource.
         *
         * @return \Illuminate\Http\Response
         */
        public function index()
        {
            return view('hello');
        }

        // ... other resources are listed below the index one above
```

Puoi servire la tua app utilizzando il seguente comando PHP Artisan: **`php artisan serve`** ; ti mostrerà l'indirizzo a cui puoi accedere alla tua applicazione *(di solito su http: // localhost: 8000 per impostazione predefinita)* .

In alternativa, puoi andare direttamente alla posizione appropriata nel tuo browser; *nel caso tu stia utilizzando un server come XAMPP (http: // localhost / hello-world / public / hello dovresti aver installato l'istanza di Laravel,* ***`hello-world`*** *, direttamente nella tua directory* ***xampp / htdocs*** *come in: avendo eseguito il passaggio 1 di questo Hello Word dall'interfaccia della riga di comando, che punta alla directory* ***xampp / htdocs*** *)* .

Hello World Example (Base)

Apri il file delle rotte. Incolla il seguente codice in:

```
Route::get('helloworld', function () {
    return '<h1>Hello World</h1>';
});
```

dopo aver percorso il `localhost/helloworld` visualizza **Hello World** .

Il file di rotte si trova:

5.3

Per il Web

```
routes/web.php
```

Per le API

```
routes/api.php
```

5.2 5.1 (LTS) 5.0

```
app/Http/routes.php
```

4.2

```
app/routes.php
```

Installazione usando LaraDock (Laravel Homestead per Docker)

LaraDock è un ambiente di sviluppo simile a Laravel Homestead ma per Docker invece di Vagrant. https://github.com/LaraDock/laradock

Installazione

* Richiede Git e Docker

Clona il repository LaraDock:

R. Se hai già un progetto Laravel, clona questo repository sulla tua directory principale di Laravel:

```
git submodule add https://github.com/LaraDock/laradock.git
```

B. Se non hai un progetto Laravel e vuoi installare Laravel da Docker, clona questo repository ovunque sul tuo computer:

```
git clone https://github.com/LaraDock/laradock.git
```

Uso di base

1. Esegui contenitori: (Assicurati di essere nella cartella laradock prima di eseguire i comandi di composizione docker).

   Esempio: esecuzione di NGINX e MySQL: `docker-compose up -d nginx mysql`

   C'è una lista di contenitori disponibili che puoi selezionare per creare le tue combinazioni.

   `nginx` , `hhvm` , `php-fpm` , `mysql` , `redis` , `postgres` , `mariadb` , `neo4j` , `mongo` , `apache2` , `caddy` , `memcached` , `beanstalkd` , `beanstalkd-console` , `workspace`

2. Entra nel contenitore Workspace, per eseguire comandi come (Artisan, Composer, PHPUnit, Gulp, ...).

   ```
   docker-compose exec workspace bash
   ```

3. Se non hai ancora installato un progetto Laravel, segui il passaggio per installare Laravel da un contenitore Docker.

   un. Inserisci il contenitore dell'area di lavoro.

   b. Installa Laravel. `composer create-project laravel/laravel my-cool-app "5.3.*"`

4. Modifica le configurazioni di Laravel. Apri il file .env di Laravel e imposta DB_HOST sul tuo mysql:

```
DB_HOST=mysql
```

5. Apri il tuo browser e visita il tuo indirizzo localhost.

Leggi Installazione online: https://riptutorial.com/it/laravel/topic/7961/installazione

Capitolo 31: Integrazione Sparkpost con Laravel 5.4

introduzione

Laravel 5.4 viene preinstallato con sparkpost api lib. Sparkpost lib richiede una chiave segreta che si può trovare dal proprio account sparkpost.

Examples

ESEMPIO dati del file .env

Per creare correttamente una configurazione di API e-mail sparkpost, aggiungi i dettagli di seguito al file ENV e la tua applicazione sarà buona per iniziare a inviare e-mail.

MAIL_DRIVER = sparkpost
SPARKPOST_SECRET =

NOTA: i dettagli di cui sopra non ti danno il codice scritto nel controller che ha la logica di business per inviare e-mail usando la funzione Mail :: send di laravels.

Leggi Integrazione Sparkpost con Laravel 5.4 online:
https://riptutorial.com/it/laravel/topic/10136/integrazione-sparkpost-con-laravel-5-4

Capitolo 32: Introduzione a laravel-5.2

introduzione

Laravel è un framework MVC con bundle, migrazioni e Artis CLI. Laravel offre un robusto set di strumenti e un'architettura applicativa che incorpora molte delle migliori caratteristiche di framework come CodeIgniter, Yii, ASP.NET MVC, Ruby on Rails, Sinatra e altri. Laravel è un framework Open Source. Ha un ricco set di funzionalità che aumenterà la velocità di sviluppo Web. Se hai familiarità con Core PHP e Advanced PHP, Laravel renderà più semplice il tuo compito. Risparmierà molto tempo.

Osservazioni

Questa sezione fornisce una panoramica di cosa è laravel-5.1 e perché uno sviluppatore potrebbe volerlo usare.

Dovrebbe anche menzionare eventuali soggetti di grandi dimensioni all'interno di laravel-5.1 e collegarsi agli argomenti correlati. Poiché la documentazione di laravel-5.1 è nuova, potrebbe essere necessario creare versioni iniziali di tali argomenti correlati.

Examples

Installazione o configurazione

Istruzioni per l'installazione di Laravel 5.1 su una macchina Linux / Mac / Unix.

Prima di iniziare l'installazione, verificare se sono soddisfatti i seguenti requisiti:

- PHP> = 5.5.9
- Estensione PHP OpenSSL
- Estensione PHP PDO
- Estensione PHP di Mbstring
- Estensione PHP Tokenizer

Iniziamo l'installazione:

1. Installa compositore. Documentazione del compositore
2. Esegui `composer create-project laravel/laravel <folder-name> "5.1.*"`
3. Assicurarsi che la cartella di `storage` e la cartella `bootstrap/cache` siano scrivibili.
4. Apri il file `.env` e imposta le informazioni di configurazione come credenziali del database, stato di debug, ambiente applicativo, ecc.
5. Esegui `php artisan serve` e indirizza il browser a `http://localhost:8000` . Se tutto va bene allora dovresti prendere la pagina

Installa Laravel 5.1 Framework su Ubuntu 16.04, 14.04 e LinuxMint

Passaggio 1: installare LAMP

Per iniziare con Laravel, abbiamo prima bisogno di configurare un server LAMP in esecuzione. Se hai già in esecuzione stack LAMP, salta questo passaggio altrimenti usa i seguenti comandi per impostare la lampada sul sistema Ubuntu.

Installa PHP 5.6

```
$ sudo apt-get install python-software-properties
$ sudo add-apt-repository ppa:ondrej/php
$ sudo apt-get update
$ sudo apt-get install -y php5.6 php5.6-mcrypt php5.6-gd
```

Installa Apache2

```
$ apt-get install apache2 libapache2-mod-php5
```

Installa MySQL

```
$ apt-get install mysql-server php5.6-mysql
```

Passaggio 2: installa Composer

Il compositore è richiesto per l'installazione delle dipendenze di Laravel. Quindi utilizzare i comandi sottostanti per scaricare e utilizzare come comando nel nostro sistema.

```
$ curl -sS https://getcomposer.org/installer | php
$ sudo mv composer.phar /usr/local/bin/composer
$ sudo chmod +x /usr/local/bin/composer
```

Passaggio 3: installa Laravel

Per scaricare l'ultima versione di Laravel, utilizzare il comando riportato di seguito per clonare il repository principale di laravel da github.

```
$ cd /var/www
$ git clone https://github.com/laravel/laravel.git
```

Passare alla directory del codice di Laravel e utilizzare composer per installare tutte le dipendenze richieste per il framework Laravel.

```
$ cd /var/www/laravel
$ sudo composer install
```

L'installazione delle dipendenze richiederà del tempo. Dopo aver impostato le autorizzazioni appropriate per i file.

```
$ chown -R www-data.www-data /var/www/laravel
$ chmod -R 755 /var/www/laravel
```

```
$ chmod -R 777 /var/www/laravel/app/storage
```

Passaggio 4: impostare la chiave di crittografia

Ora imposta la chiave di crittografia del numero casuale a 32 bit, utilizzata dal servizio di crittografia di Illuminate.

```
$ php artisan key:generate

Application key [uOHTNu3Au1Kt7Uloyr2Py9blU0J5XQ75] set successfully.
```

Ora modifica il file di configurazione `config/app.php` e aggiorna la chiave dell'applicazione sopra descritta come segue. Assicurati anche che il cipher sia impostato correttamente.

```
'key' => env('APP_KEY', 'uOHTNu3Au1Kt7Uloyr2Py9blU0J5XQ75'),

'cipher' => 'AES-256-CBC',
```

Passaggio 5: creare Apache VirtualHost

Ora aggiungi un host virtuale nel tuo file di configurazione Apache per accedere al framework Laravel dal browser web. Creare il file di configurazione di Apache nella directory `/etc/apache2/sites-available/` e aggiungere sotto il contenuto.

```
$ vim /etc/apache2/sites-available/laravel.example.com.conf
```

Questa è la struttura del file dell'host virtuale.

```
<VirtualHost *:80>

        ServerName laravel.example.com
        DocumentRoot /var/www/laravel/public

        <Directory />
                Options FollowSymLinks
                AllowOverride None
        </Directory>
        <Directory /var/www/laravel>
                AllowOverride All
        </Directory>

        ErrorLog ${APACHE_LOG_DIR}/error.log
        LogLevel warn
        CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

Infine, consente di abilitare il sito Web e di ricaricare il servizio Apache utilizzando il comando riportato di seguito.

```
$ a2ensite laravel.example.com
$ sudo service apache2 reload
```

Passaggio 6: accedi a Laravel

A questo punto hai completato con successo il framework PHP laravel 5 sul tuo sistema. Ora fai la voce del file host per accedere alla tua applicazione Laravel nel browser web. Cambia `127.0.0.1` con il tuo server ip e `laravel.example.com` con il tuo nome di dominio configurato in Apache.

```
$ sudo echo "127.0.0.1  laravel.example.com" >> /etc/hosts
```

E accedi a http://laravel.example.com nel tuo browser web preferito come sotto.

Leggi Introduzione a laravel-5.2 online: https://riptutorial.com/it/laravel/topic/1987/introduzione-a-laravel-5-2

Capitolo 33: Introduzione a laravel-5.3

introduzione

Nuove funzionalità, miglioramenti e modifiche da Laravel 5.2 a 5.3

Examples

La variabile $ loop

È noto da un po 'che il trattamento dei loop in Blade è stato limitato, a partire da 5.3 c'è una variabile chiamata `$loop` disponibile

```
@foreach($variables as $variable)

    // Within here the `$loop` variable becomes available

    // Current index, 0 based
    $loop->index;

    // Current iteration, 1 based
    $loop->iteration;

    // How many iterations are left for the loop to be complete
    $loop->remaining;

    // Get the amount of items in the loop
    $loop->count;

    // Check to see if it's the first iteration ...
    $loop->first;

    // ... Or last iteration
    $loop->last;

    //Depth of the loop, ie if a loop within a loop the depth would be 2, 1 based counting.
    $loop->depth;

    // Get's the parent `$loop` if the loop is nested, else null
    $loop->parent;

@endforeach
```

Leggi Introduzione a laravel-5.3 online: https://riptutorial.com/it/laravel/topic/9231/introduzione-a-laravel-5-3

Capitolo 34: Laravel Docker

introduzione

Una sfida affrontata da ogni sviluppatore e team di sviluppo è la coerenza ambientale. Laravel è oggi uno dei framework PHP più popolari. DDocker, d'altra parte, è un metodo di virtualizzazione che elimina i problemi di *"funziona sulla mia macchina"* quando si collabora sul codice con altri sviluppatori. I due insieme creano una fusione di **utili** e **potenti** . Sebbene entrambi facciano cose molto diverse, entrambi possono essere combinati per creare prodotti sorprendenti.

Examples

Utilizzando Laradock

Laradock è un progetto che fornisce un ready-to-go contiene su misura per l'uso Laravel.

Scarica o clona Laradock nella cartella principale del tuo progetto:

```
git clone https://github.com/Laradock/laradock.git
```

Cambia la directory in Laradock e genera il file `.env` necessario per eseguire le tue configurazioni:

```
cd laradock
cp .env-example .env
```

Ora sei pronto per eseguire la finestra mobile. La prima volta che si esegue il contenitore verranno scaricati tutti i pacchetti necessari da Internet.

```
docker-compose up -d nginx mysql redis beanstalkd
```

Ora puoi aprire il tuo browser e visualizzare il tuo progetto su `http://localhost` .

Per la documentazione completa e la configurazione di Laradock clicca qui .

Leggi Laravel Docker online: https://riptutorial.com/it/laravel/topic/10034/laravel-docker

Capitolo 35: Link utili

introduzione

In questo argomento, puoi trovare link utili per migliorare le tue abilità Laravel o estendere le tue conoscenze.

Examples

Ecosistema Laravel

- Laravel Scout - Laravel Scout fornisce una semplice soluzione basata su driver per l'aggiunta della ricerca full-text ai modelli Eloquent.
- Laravel Passport : autenticazione API senza mal di testa. Passport è un server OAuth2 pronto in pochi minuti.
- Homestead - L'ambiente di sviluppo ufficiale di Laravel. Realizzato da Vagrant, Homestead porta l'intero team sulla stessa pagina con gli ultimi PHP, MySQL, Postgres, Redis e altri.
- Cassa di Laravel : rende la fatturazione dell'abbonamento indolore con integrazioni integrate a strisce e Braintree. Coupon, abbonamenti, cancellazioni e persino fatture PDF sono pronti all'uso.
- Forge : fornisce e distribuisce applicazioni PHP illimitate su DigitalOcean, Linode e AWS.
- Envoyer - Zero downtime PHP Deployment.
- Valet - Un ambiente di sviluppo Laravel per i minimalisti di Mac. No Vagrant, no Apache, nessun clamore.
- Spark - Potente impalcatura per applicazioni SaaS. Smetti di scrivere hotplatter e concentrati sulla tua applicazione.
- Lumen - Se tutto ciò di cui hai bisogno è un'API e una velocità fulminea, prova Lumen. È Laravel super leggero.
- Statamic - Un vero CMS progettato per rendere le agenzie redditizie, gli sviluppatori felici e i clienti ti abbracciano.

Formazione scolastica

- Laracast - Impara lo sviluppo web pratico e moderno, attraverso screencast esperti.
- Laravel News - Rimani aggiornato con Laravel con Laravel News.
- Laravel.io - Forum con codice open source.

podcast

- Podcast di Laravel News
- I podcast di Laravel

Leggi Link utili online: https://riptutorial.com/it/laravel/topic/9957/link-utili

Capitolo 36: Macro in relazione eloquente

introduzione

Abbiamo nuove funzionalità per Eloquent Relationship in Laravel versione 5.4.8. Possiamo recuperare una singola istanza di una relazione hasMany (è solo un esempio) definendola sul posto e funzionerà per tutte le relazioni

Examples

Possiamo recuperare una istanza di hasMany relationship

Nel nostro AppServiceProvider.php

```
public function boot()
{
  HasMany::macro('toHasOne', function() {
      return new HasOne(
          $this->query,
          $this->parent,
          $this->foreignKey,
          $this->localKey
      );
  });
}
```

Supponiamo di avere un negozio modale e stiamo ricevendo l'elenco dei prodotti acquistati. Supponiamo di avere una relazione interamente acquistata per Negozio modale

```
public function allPurchased()
{
    return $this->hasMany(Purchased::class);
}

public function lastPurchased()
{
    return $this->allPurchased()->latest()->toHasOne();
}
```

Leggi Macro in relazione eloquente online: https://riptutorial.com/it/laravel/topic/8998/macro-in-relazione-eloquente

Capitolo 37: middleware

introduzione

Il middleware è una classe, che può essere assegnata a uno o più percorsi e utilizzata per compiere azioni nelle fasi iniziali o finali del ciclo di richiesta. Possiamo considerarli come una serie di livelli che una richiesta HTTP deve attraversare mentre viene eseguita

Osservazioni

Un middleware "Prima" verrà eseguito prima del codice di azione del controller; mentre un middleware "After" viene eseguito dopo che la richiesta viene gestita dall'applicazione

Examples

Definire un middleware

Per definire un nuovo middleware dobbiamo creare la classe middleware:

```
class AuthenticationMiddleware
{
    //this method will execute when the middleware will be triggered
    public function handle ( $request, Closure $next )
    {
        if ( ! Auth::user() )
        {
            return redirect('login');
        }

        return $next($request);
    }
}
```

Quindi dobbiamo registrare il middleware: se il middleware deve essere associato a tutti i percorsi dell'applicazione, dovremmo aggiungerlo alla proprietà middleware `app/Http/Kernel.php` :

```
protected $middleware = [
        \Illuminate\Foundation\Http\Middleware\CheckForMaintenanceMode::class,
        \App\Http\Middleware\AuthenticationMiddleware::class
];
```

mentre se vogliamo solo associare il middleware ad alcuni dei percorsi, possiamo aggiungerlo a `$routeMiddleware`

```
//register the middleware as a 'route middleware' giving it the name of 'custom_auth'
protected $routeMiddleware = [
    'custom_auth' => \App\Http\Middleware\AuthenticationMiddleware::class
];
```

e quindi legarlo alle singole rotte come questa:

```
//bind the middleware to the admin_page route, so that it will be executed for that route
Route::get('admin_page', 'AdminController@index')->middleware('custom_auth');
```

Prima e dopo il middleware

Un esempio di middleware "prima" sarebbe il seguente:

```
<?php

namespace App\Http\Middleware;

use Closure;

class BeforeMiddleware
{
    public function handle($request, Closure $next)
    {
        // Perform action

        return $next($request);
    }
}
```

mentre il middleware "after" apparirebbe così:

```
<?php

namespace App\Http\Middleware;

use Closure;

class AfterMiddleware
{
    public function handle($request, Closure $next)
    {
        $response = $next($request);

        // Perform action

        return $response;
    }
}
```

La differenza chiave sta nel modo in cui viene gestito il parametro `$request` . Se le azioni vengono eseguite prima che `$next($request)` avvenga prima che il codice del controller venga eseguito mentre si chiama `$next($request)` , le azioni vengono eseguite dopo che il codice del controller è stato eseguito.

Installa il middleware

Qualsiasi middleware registrato come `routeMiddleware` in `app/Http/Kernel.php` può essere assegnato a un percorso.

Ci sono diversi modi per assegnare il middleware, ma tutti fanno lo stesso.

```
Route::get('/admin', 'AdminController@index')->middleware('auth', 'admin');
Route::get('admin/profile', ['using' => 'AdminController@index', 'middleware' => 'auth']);
Route::get('admin/profile', ['using' => 'AdminController@index', 'middleware' => ['auth',
'admin']);
```

In tutti gli esempi precedenti, è anche possibile passare nomi di classi pienamente qualificati come middleware, indipendentemente dal fatto che sia stato registrato come middleware di route.

```
use App\Http\Middleware\CheckAdmin;
Route::get('/admin', 'AdminController@index')->middleware(CheckAdmin::class);
```

Leggi middleware online: https://riptutorial.com/it/laravel/topic/3419/middleware

Capitolo 38: Migrazioni del database

Examples

migrazioni

Per controllare il tuo database in Laravel è usando le migrazioni. Crea migrazione con artigiano:

```
php artisan make:migration create_first_table --create=first_table
```

Questo genererà la classe CreateFirstTable. All'interno del metodo up puoi creare le tue colonne:

```
<?php

use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;

class CreateFirstTable extends Migration
{
    public function up()
    {
        Schema::create('first_table', function (Blueprint $table) {
            $table->increments('id');
            $table->string('first_string_column_name');
            $table->integer('secont_integer_column_name');
            $table->timestamps();
        });
    }

    public function down()
    {
        Schema::drop('first_table');
    }
}
```

Alla fine di eseguire tutte le classi di migrazione è possibile eseguire il comando artisan:

```
php artisan migrate
```

Questo creerà le tue tabelle e le tue colonne nel tuo database. Altri comandi di migrazione utili sono:

- `php artisan migrate:rollback` - Rollback l'ultima migrazione del database
- `php artisan migrate:reset` - Ripristina tutte le migrazioni di database
- `php artisan migrate:refresh` - Reimposta e riesegui tutte le migrazioni
- `php artisan migrate:status` : mostra lo stato di ciascuna migrazione

Modifica delle tabelle esistenti

A volte, è necessario modificare la struttura della tabella esistente come `renaming/deleting`

colonne. Che puoi realizzare creando una nuova migrazione. E nel metodo `up` della tua migrazione.

```
//Renaming Column.

public function up()
{
    Schema::table('users', function (Blueprint $table) {
        $table->renameColumn('email', 'username');
    });
}
```

L'esempio sopra ridenomina la `email column` della `users table` al `username` . Mentre il codice sottostante rilascia un `username` colonna dalla tabella degli `users` .

IMPROTANT: per modificare le colonne è necessario aggiungere la dipendenza `doctrine/dbal` al file `composer.json` del progetto ed eseguire l' `composer update` per riflettere le modifiche.

```
//Droping Column
public function up()
{
    Schema::table('users', function (Blueprint $table) {
        $table->dropColumn('username');
    });
}
```

I file di migrazione

Le migrazioni in un'applicazione Laravel 5 risiedono nella directory `database/migrations` . I loro nomi di file sono conformi a un formato particolare:

```
<year>_<month>_<day>_<hour><minute><second>_<name>.php
```

Un file di migrazione dovrebbe rappresentare un aggiornamento dello schema per risolvere un particolare problema. Per esempio:

```
2016_07_21_134310_add_last_logged_in_to_users_table.php
```

Le migrazioni dei database sono mantenute in ordine cronologico in modo che Laravel sappia in quale ordine eseguirle. Laravel eseguirà sempre le migrazioni dal più vecchio al più recente.

Generazione di file di migrazione

Creare un nuovo file di migrazione con il nome file corretto ogni volta che è necessario modificare lo schema sarebbe un lavoro ingrato. Per fortuna, il comando `artisan` di Laravel può generare la migrazione per voi:

```
php artisan make:migration add_last_logged_in_to_users_table
```

È inoltre possibile utilizzare le `--table` e `--create` bandiere con il comando precedente. Questi

sono opzionali e solo per comodità e inseriranno il codice boilerplate pertinente nel file di migrazione.

```
php artisan make:migration add_last_logged_in_to_users_table --table=users

php artisan make:migration create_logs_table --create=logs
```

È possibile specificare un percorso di output personalizzato per la migrazione generata utilizzando l'opzione `--path` . Il percorso è relativo al percorso di base dell'applicazione.

```
php artisan make:migration --path=app/Modules/User/Migrations
```

All'interno di una migrazione del database

Ogni migrazione dovrebbe avere un metodo `up()` e un metodo `down()` . Lo scopo del metodo `up()` è eseguire le operazioni richieste per mettere lo schema del database nel suo nuovo stato, e lo scopo del metodo `down()` è di invertire qualsiasi operazione eseguita dal metodo `up()` . Garantire che il metodo `down()` annulli correttamente le operazioni è fondamentale per poter ripristinare le modifiche dello schema del database.

Un esempio di file di migrazione potrebbe essere simile a questo:

```
<?php

use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;

class AddLastLoggedInToUsersTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::table('users', function (Blueprint $table) {
            $table->dateTime('last_logged_in')->nullable();
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::table('users', function (Blueprint $table) {
            $table->dropColumn('last_logged_in');
        });
    }
}
```

Durante l'esecuzione di questa migrazione, Laravel genererà il seguente SQL da eseguire sul tuo database:

```
ALTER TABLE `users` ADD `last_logged_in` DATETIME NULL
```

Esecuzione di migrazioni

Una volta scritta la migrazione, eseguendola verranno applicate le operazioni al database.

```
php artisan migrate
```

Se tutto è andato bene, vedrai un risultato simile al seguente:

```
Migrated: 2016_07_21_134310_add_last_logged_in_to_users_table
```

Laravel è abbastanza intelligente da sapere quando esegui migrazioni nell'ambiente di produzione. Se rileva che stai eseguendo una migrazione distruttiva (ad esempio, una che rimuove una colonna da una tabella), il comando `php artisan migrate` ti chiederà conferma. In ambienti di consegna continua questo potrebbe non essere desiderato. In tal caso, usa il flag `--force` per saltare la conferma:

```
php artisan migrate --force
```

Se hai appena eseguito le migrazioni, potresti essere confuso nel vedere la presenza di una tabella delle `migrations` nel tuo database. Questa tabella è ciò che Laravel utilizza per tenere traccia di ciò che le migrazioni sono già state eseguite. Quando si esegue il comando `migrate` , Laravel determinerà quali migrazioni devono ancora essere eseguite, quindi le eseguirà in ordine cronologico, quindi aggiornerà la tabella delle `migrations` in base alle esigenze.

Non dovresti mai modificare manualmente la tabella delle `migrations` meno che tu non sappia assolutamente cosa stai facendo. È molto facile lasciare inavvertitamente il database in uno stato di errore in cui le migrazioni falliscono.

Rolling Back Migrations

Che cosa succede se si desidera eseguire il rollback dell'ultima migrazione, ad esempio un'operazione recente, è possibile utilizzare il comando di `rollback` fantastico. Ma ricorda che questo comando ripristina solo l'ultima migrazione, che può includere più file di migrazione

```
php artisan migrate:rollback
```

Se sei interessato a ripristinare tutte le migrazioni delle applicazioni, puoi utilizzare il seguente comando

```
php artisan migrate:reset
```

Inoltre se sei pigro come me e vuoi eseguire il rollback e migrare con un solo comando, puoi usare questo comando

```
php artisan migrate:refresh
php artisan migrate:refresh --seed
```

È anche possibile specificare il numero di passaggi per il rollback con l'opzione `step` . In questo modo verrà eseguito il rollback di 1 passaggio.

```
php artisan migrate:rollback --step=1
```

Leggi Migrazioni del database online: https://riptutorial.com/it/laravel/topic/1131/migrazioni-del-database

Capitolo 39: Modelli di lama

introduzione

Laravel supporta il motore di templatura della lama fuori dalla scatola. Il motore di template di Blade ci consente di creare modelli master e modelli child caricando il contenuto da modelli master, possiamo avere variabili, cicli e istruzioni condizionali all'interno del file blade.

Examples

Viste: Introduzione

Le viste, in uno schema MVC, contengono la logica su *come* presentare i dati all'utente. In un'applicazione Web, in genere vengono utilizzati per generare l'output HTML che viene restituito agli utenti con ciascuna risposta. Per impostazione predefinita, le viste in Laravel sono memorizzate nella directory `resources/views` .

Una vista può essere chiamata usando la funzione helper `view` :

```
view(string $path, array $data = [])
```

Il primo parametro dell'helper è il percorso di un file di visualizzazione e il secondo parametro è una matrice facoltativa di dati da passare alla vista.

Pertanto, per chiamare le `resources/views/example.php` , si utilizzerà:

```
view('example');
```

Visualizza i file in sottocartelle all'interno della directory `resources/views` , come `resources/views/parts/header/navigation.php` , può essere chiamato usando dot notation: `view('parts.header.navigation');`

All'interno di un file di visualizzazione, come `resources/views/example.php` , sei libero di includere sia HTML che PHP insieme:

```
<html>
    <head>
        <title>Hello world!</title>
    </head>
    <body>
        <h1>Welcome!</h1>
        <p>Your name is: <?php echo $name; ?></p>
    </body>
</html>
```

Nell'esempio precedente (che non utilizza alcuna sintassi specifica di Blade), emettiamo la variabile `$name` . Per passare questo valore alla nostra vista, passeremmo una matrice di valori

quando chiamiamo l'helper della vista:

```
view('example', ['name' => $name]);
```

o in alternativa, utilizzare l'helper `compact()` . In questo caso, la stringa passata a compact () corrisponde al nome della variabile che vogliamo passare alla vista.

```
view('example', compact('name'));
```

CONVENZIONE NOMINATIVO PER VARIABILI DI BLADE

Durante l'invio dei dati alla vista. Puoi usare `underscore` per `variable` multi-words ma con `-` laravel dà errore.

Come questo darà errore (notare il `hyphen ( - )` all'interno `user-address`

```
view('example',['user-address' => 'Some Address']);
```

Il **modo corretto** di farlo sarà

```
view('example', ['user_address' => 'Some Address']);
```

Strutture di controllo

Blade fornisce una comoda sintassi per strutture di controllo PHP comuni.

Ciascuna delle strutture di controllo inizia con `@[structure]` e termina con `@[endstructure]` . Si noti che all'interno dei tag, stiamo semplicemente digitando normale HTML e includendo variabili con la sintassi di Blade.

Condizionali

'Se' dichiarazioni

```
@if ($i > 10)
    <p>{{ $i }} is large.</p>
@elseif ($i == 10)
    <p>{{ $i }} is ten.</p>
@else
    <p>{{ $i }} is small.</p>
@endif
```

'A meno che non dichiarazioni'

(Sintassi breve per "se non").

```
@unless ($user->hasName())
    <p>A user has no name.</p>
@endunless
```

Loops

Ciclo 'While'

```
@while (true)
    <p>I'm looping forever.</p>
@endwhile
```

Ciclo 'Foreach'

```
@foreach ($users as $id => $name)
    <p>User {{ $name }} has ID {{ $id }}.</p>
@endforeach
```

Ciclo 'Forelse'

(Come il ciclo 'foreach', ma aggiunge una direttiva `@empty` speciale, che viene eseguita quando l'espressione dell'array iterata su è vuota, come modo per mostrare il contenuto predefinito.)

```
@forelse($posts as $post)
    <p>{{ $post }} is the post content.</p>
@empty
    <p>There are no posts.</p>
@endforelse
```

All'interno dei loop, sarà disponibile una speciale variabile `$loop` , contenente informazioni sullo stato del loop:

Proprietà	Descrizione
`$loop->index`	L'indice dell'iterazione del loop corrente (inizia da 0).
`$loop->iteration`	L'iterazione del loop corrente (inizia da 1).
`$loop->remaining`	Le restanti iterazioni del ciclo.
`$loop->count`	Il numero totale di elementi nell'array che viene iterato.
`$loop->first`	Se questa è la prima iterazione del ciclo.
`$loop->last`	Se questa è l'ultima iterazione attraverso il ciclo.

Proprietà	Descrizione
`$loop->depth`	Il livello di annidamento del loop corrente.
`$loop->parent`	Quando si trova in un ciclo annidato, la variabile del ciclo del genitore.

Esempio:

```
@foreach ($users as $user)
  @foreach ($user->posts as $post)
        @if ($loop->parent->first)
            This is first iteration of the parent loop.
        @endif
    @endforeach
@endforeach
```

Dal momento che Laravel 5.2.22, possiamo anche usare le direttive `@continue` e `@break`

Proprietà	Descrizione
`@continue`	Interrompe l'iterazione corrente e avvia quella successiva.
`@break`	Interrompe il loop corrente.

Esempio :

```
@foreach ($users as $user)
    @continue ($user->id == 2)
        <p>{{ $user->id }} {{ $user->name }}</p>
    @break ($user->id == 4)
@endforeach
```

Quindi (*supponendo che più di 5 utenti siano ordinati per ID e non sia presente alcun ID*) la pagina verrà visualizzata

```
1 Dave
3 John
4 William
```

Facendo eco alle espressioni PHP

Qualsiasi espressione PHP all'interno di doppie parentesi graffe `{{ $variable }}` verrà `echo` dopo essere stata eseguita attraverso la funzione `e` helper . (Quindi i caratteri speciali html (`<` , `>` , `"` , `'` , `&`) vengono tranquillamente sostituiti per le entità html corrispondenti.) (L'espressione PHP deve valutare la stringa, altrimenti verrà generata un'eccezione).

Echoing una variabile

```
{{ $variable }}
```

Echoing di un elemento in una matrice

```
{{ $array["key"] }}
```

Echoing di una proprietà dell'oggetto

```
{{ $object->property }}
```

Riprendendo il risultato di una chiamata di funzione

```
{{ strtolower($variable) }}
```

Controllo dell'esistenza

Normalmente, in PHP, per verificare se una variabile è impostata e stamparla si farebbe

- Prima di PHP 7

```
<?php echo isset($variable) ? $variable : 'Default'; ?>
```

- Dopo PHP 7 (utilizzando "Null coalescing operator")

```
<?php echo $variable ?? 'Default'; ?>
```

L'operatore della lama `or` lo semplifica:

```
{{ $variable or 'Default' }}
```

Echi grezzi

Come accennato, la normale sintassi delle doppie parentesi `{{ }}` , viene filtrata tramite la funzione `htmlspecialchars` di PHP, per `htmlspecialchars` di sicurezza (impedendo l'iniezione dannosa di HTML nella vista). Se si desidera ignorare questo comportamento, ad esempio se si sta tentando di generare un blocco di contenuto HTML risultante da un'espressione PHP, utilizzare la seguente sintassi:

```
{!! $myHtmlString !!}
```

Si noti che è considerata una buona pratica utilizzare la sintassi standard `{{ }}` per evitare i propri

dati, a meno che non sia assolutamente necessario. Inoltre, quando si riproducono contenuti non fidati (ad es. Contenuto fornito dagli utenti del proprio sito), si dovrebbe evitare di usare `{!! !!}` sintassi.

Inclusione di viste parziali

Con Blade, puoi anche includere viste parziali (chiamate "parziali") direttamente in una pagina come questa:

```
@include('includes.info', ['title' => 'Information Station'])
```

Il codice sopra includerà la vista in "views / includes / info.blade.php". Passerà anche in un `$title` variabile avente valore 'Stazione informazioni'.

In generale, una pagina inclusa avrà accesso a qualsiasi variabile a cui ha accesso la pagina chiamante. Ad esempio, se abbiamo:

```
{{$user}} // Outputs 'abc123'
@include('includes.info')
```

E 'include / info.blade.php' ha il seguente:

```
<p>{{$user}} is the current user.</p>
```

Quindi la pagina renderà:

```
abc123
abc123 is the current user.
```

Includi Ciascuno

A volte, si desidera combinare un'istruzione `include` con un'istruzione `foreach` e accedere alle variabili dall'interno del ciclo foreach nell'include. In questo caso, usa la direttiva `@each` di Blade:

```
@each('includes.job', $jobs, 'job')
```

Il primo parametro è la pagina da includere. Il secondo parametro è la matrice da iterare sopra. Il terzo parametro è la variabile assegnata agli elementi dell'array. La dichiarazione di cui sopra è equivalente a:

```
@foreach($jobs as $job)
    @include('includes.job', ['job' => $job])
@endforeach
```

È anche possibile passare un quarto argomento facoltativo alla direttiva `@each` per specificare la vista da mostrare quando la matrice è vuota.

```
@each('includes.job', $jobs, 'job', 'includes.jobsEmpty')
```

Ereditarietà del layout

Un layout è un file di visualizzazione, che viene esteso da altre viste che iniettano blocchi di codice nel loro genitore. Per esempio:

parent.blade.php:

```
<html>
    <head>
        <style type='text/css'>
        @yield('styling')
        </style>
    </head>
    <body>
        <div class='main'>
        @yield('main-content')
        </div>
    </body>
</html>
```

child.blade.php:

```
@extends('parent')

@section('styling')
.main {
    color: red;
}
@stop

@section('main-content')
This is child page!
@stop
```

otherpage.blade.php:

```
@extends('parent')

@section('styling')
.main {
    color: blue;
}
@stop

@section('main-content')
This is another page!
@stop
```

Qui vedete due pagine figlio di esempio, che estendono ciascuna il genitore. Le pagine `@section` definiscono una `@section` , che viene inserita nel genitore `@yield` appropriata.

Quindi la vista renderizzata da `View::make('child')` dirà " **Questa è una pagina figlio!** " In rosso, mentre `View::make('otherpage')` produrrà lo stesso html, tranne con il testo " **Questa è un'altra pagina!** "in blu invece.

È comune separare i file di visualizzazione, ad esempio disporre di una cartella di layout specifica per i file di layout e una cartella separata per le varie visualizzazioni individuali specifiche.

I layout hanno lo scopo di applicare un codice che dovrebbe apparire su ogni pagina, ad esempio aggiungendo una barra laterale o un'intestazione, senza dover scrivere tutto il codice html in ogni singola visualizzazione.

Le viste possono essere estese ripetutamente, ad esempio **page3** can `@extend('page2')` , e **page2** can `@extend('page1')` .

Il comando estendi usa la stessa sintassi usata per `View::make` e `@include` , quindi il `layouts/main/page.blade.php` file `layouts/main/page.blade.php` è accessibile come `layouts.main.page` .

Condivisione di dati a tutte le viste

A volte è necessario impostare gli stessi dati in molte delle visualizzazioni.

Utilizzando View :: share

```
// "View" is the View Facade
View::share('shareddata', $data);
```

Dopodiché, il contenuto di `$data` sarà disponibile in tutte le visualizzazioni sotto il nome `$shareddata` .

`View::share` viene in genere chiamato in un fornitore di servizi, o forse nel costruttore di un controller, quindi i dati saranno condivisi nelle viste restituite solo da quel controller.

Utilizzando View :: compositore

I compositori di vista sono callback o metodi di classe chiamati quando viene eseguito il rendering di una vista. Se si dispone di dati che si desidera associare a una vista ogni volta che viene visualizzata la vista, un compositore di viste può aiutare a organizzare quella logica in un'unica posizione. È possibile associare direttamente una variabile a una vista specifica oa tutte le viste.

Compositore di chiusura

```
use Illuminate\Support\Facades\View;

// ...

View::composer('*', function ($view) {
    $view->with('somedata', $data);
});
```

Compositore di classe

```
use Illuminate\Support\Facades\View;

// ...

View::composer('*', 'App\Http\ViewComposers\SomeComposer');
```

Come con `View::share` , è meglio registrare i compositori in un fornitore di servizi.

Se vai con l'approccio della classe del compositore, allora avresti `App/Http/ViewComposers/SomeComposer.php` con:

```
use Illuminate\Contracts\View\View;

class SomeComposer
{
    public function compose(View $view)
    {
        $view->with('somedata', $data);
    }
}
```

Questi esempi usano `'*'` nella registrazione del compositore. Questo parametro è una stringa che corrisponde ai nomi di vista per i quali registrare il compositore (`*` essendo un carattere jolly). Puoi anche selezionare una vista singola (ad es. `'home'`) di un gruppo di percorsi in una sottocartella (ad es. `'users.*'`).

Esegui codice PHP arbitrario

Anche se potrebbe non essere corretto farlo in una vista se si intende separare strettamente le preoccupazioni, la direttiva `php` Blade consente un modo per eseguire codice PHP, ad esempio, per impostare una variabile:

```
@php($varName = 'Enter content ')
```

(uguale a:)

```
@php
    $varName = 'Enter content ';
@endphp
```

dopo:

```
{{ $varName }}
```

Risultato:

> Inserisci il contenuto

Leggi Modelli di lama online: https://riptutorial.com/it/laravel/topic/1407/modelli-di-lama

Capitolo 40: Modifica il comportamento di routing predefinito in Laravel 5.2.31 +

Sintassi

- mappa delle funzioni pubbliche (router $ router) // Definire i percorsi per l'applicazione.
- funzione protetta mapWebRoutes (router $ router) // Definire i percorsi "web" per l'applicazione.

Parametri

Parametro	Intestazione
Router $ router	\ Illuminate \ Routing \ Router $ router

Osservazioni

Middleware significa che ogni chiamata a un percorso passerà attraverso il middleware prima di colpire effettivamente il codice specifico del percorso. In Laravel il middleware web viene utilizzato per garantire la gestione delle sessioni o il controllo del token csrf, ad esempio.

Di default ci sono altri media come auth o api. Puoi anche creare facilmente il tuo middleware.

Examples

Aggiunta di api-rotte con altri middleware e mantiene il middleware Web predefinito

Dal momento che Laravel versione 5.2.31 il middleware web è applicato di default all'interno del RouteServiceProvider (https://github.com/laravel/laravel/commit/5c30c98db96459b4cc878d085490e4677b0b67ed)

In app / Provider / RouteServiceProvider.php troverai le seguenti funzioni che applicano il middleware su ogni percorso all'interno della tua app / Http / routes.php

```
public function map(Router $router)
{
    $this->mapWebRoutes($router);
}

// ...

protected function mapWebRoutes(Router $router)
{
```

```
    $router->group([
        'namespace' => $this->namespace, 'middleware' => 'web',
    ], function ($router) {
        require app_path('Http/routes.php');
    });
}
```

Come puoi vedere il web del **middleware** è applicato. Puoi cambiarlo qui. Tuttavia, puoi anche aggiungere facilmente un'altra voce per poter inserire i tuoi percorsi API per esempio in un altro file (ad esempio routes-api.php)

```
public function map(Router $router)
{
    $this->mapWebRoutes($router);
    $this->mapApiRoutes($router);
}

protected function mapWebRoutes(Router $router)
{
    $router->group([
        'namespace' => $this->namespace, 'middleware' => 'web',
    ], function ($router) {
        require app_path('Http/routes.php');
    });
}

protected function mapApiRoutes(Router $router)
{
    $router->group([
        'namespace' => $this->namespace, 'middleware' => 'api',
    ], function ($router) {
        require app_path('Http/routes-api.php');
    });
}
```

Con questo puoi facilmente separare i percorsi API dai percorsi delle applicazioni senza il wrapper di gruppo disordinato all'interno del tuo route.php

Leggi Modifica il comportamento di routing predefinito in Laravel 5.2.31 + online: https://riptutorial.com/it/laravel/topic/4285/modifica-il-comportamento-di-routing-predefinito-in-laravel-5-2-31-plus

Capitolo 41: Nozioni di base di Cron

introduzione

Cron è un demone del programma di pianificazione delle attività che esegue le attività pianificate a determinati intervalli. Cron utilizza un file di configurazione chiamato crontab, noto anche come cron table, per gestire il processo di pianificazione.

Examples

Crea un processo cron

Crontab contiene lavori cron, ognuno relativo a un'attività specifica. I lavori Cron sono composti da due parti, l'espressione cron e un comando shell da eseguire:

```
* * * * * comando / per / esecuzione
```

Ogni campo nell'espressione sopra `* * * * *` è un'opzione per impostare la frequenza del programma. È composto da minuto, ora, giorno del mese, mese e giorno della settimana in ordine del posizionamento. Il simbolo asterisco si riferisce a tutti i valori possibili per il rispettivo campo. Di conseguenza, il processo cron di cui sopra verrà eseguito ogni minuto del giorno.

Il seguente cron job viene eseguito alle **12:30** ogni giorno:

```
30 12 * * * command / to / run
```

Leggi Nozioni di base di Cron online: https://riptutorial.com/it/laravel/topic/9891/nozioni-di-base-di-cron

Capitolo 42: Osservatore

Examples

Creare un osservatore

Gli osservatori sono usati per ascoltare i callback del ciclo di vita di un determinato modello in Laravel.
Questi ascoltatori possono ascoltare una delle seguenti azioni:

- la creazione di
- creato
- in aggiornamento
- aggiornato
- Salvataggio
- salvato
- eliminazione
- cancellato
- ripristino
- restaurato

Ecco un esempio di un osservatore.

UserObserver

```
<?php

namespace App\Observers;

/**
 * Observes the Users model
 */
class UserObserver
{
    /**
     * Function will be triggerd when a user is updated
     *
     * @param Users $model
     */
     public function updated($model)
     {
         // execute your own code
     }
}
```

Come mostrato nell'osservatore utente, ascoltiamo l'azione aggiornata, tuttavia prima che questa classe ascolti effettivamente il modello utente, dobbiamo prima registrarlo all'interno di `EventServiceProvider` .

EventServiceProvider

```php
<?php

namespace App\Providers;

use Illuminate\Contracts\Events\Dispatcher as DispatcherContract;
use Illuminate\Foundation\Support\Providers\EventServiceProvider as ServiceProvider;

use App\Models\Users;
use App\Observers\UserObserver;

/**
 * Event service provider class
 */
class EventServiceProvider extends ServiceProvider
{
    /**
     * Boot function
     *
     * @param DispatcherContract $events
     */
    public function boot(DispatcherContract $events)
    {
        parent::boot($events);

        // In this case we have a User model that we want to observe
        // We tell Laravel that the observer for the user model is the UserObserver
        Users::observe(new UserObserver());
    }
}
```

Ora che abbiamo registrato il nostro osservatore, la funzione aggiornata verrà richiamata ogni volta dopo aver salvato il modello dell'utente.

Leggi Osservatore online: https://riptutorial.com/it/laravel/topic/7128/osservatore

Capitolo 43: Pacchetti Laravel

Examples

laravel-ide-helper

Questo pacchetto genera un file che l'IDE comprende, quindi può fornire un completamento automatico accurato. La generazione viene eseguita in base ai file nel progetto.

Leggi di più su questo qui

laravel-DataTable

Questo pacchetto è stato creato per gestire i lavori sul lato server di DataTables Plugin jQuery tramite l'opzione AJAX utilizzando ORI, Fluent Query Builder o Raccolta.

Leggi di più su questo qui o qui

Immagine di intervento

Intervention Image è una libreria di manipolazione e manipolazione di immagini PHP open source. Fornisce un modo più semplice ed espressivo per creare, modificare e comporre immagini e supporta attualmente le due librerie di elaborazione immagini più comuni GD Library e Imagick.

Leggi di più su questo qui

Generatore di laravel

Ottieni le tue API e il pannello di amministrazione pronto in pochi minuti. Generatore di araravel per generare CRUD, API, casi di test e documentazione Swagger

Leggi di più su questo qui

Laravel Socialite

Laravel Socialite fornisce un'interfaccia espressiva e fluente all'autenticazione OAuth con Facebook, Twitter, Google, LinkedIn, GitHub e Bitbucket. Gestisce quasi tutto il codice di autenticazione sociale boilerplate che stai temendo di scrivere.

Leggi di più su questo qui

Pacchetti ufficiali

Cassiere

Laravel Cashier fornisce un'interfaccia espressiva e fluente ai servizi di fatturazione in

abbonamento di Stripe e Braintree . Gestisce quasi tutto il codice di fatturazione dell'abbonamento che stai temendo di scrivere. Oltre alla gestione base degli abbonamenti, la Cassa può gestire i coupon, scambiare abbonamento, "quantità" di sottoscrizione, periodi di cancellazione e persino generare PDF di fatturazione.

Ulteriori informazioni su questo pacchetto sono disponibili qui .

Inviato

Laravel Envoy fornisce una sintassi pulita e minimale per la definizione delle attività comuni eseguite sui server remoti. Utilizzando la sintassi in stile Blade, puoi facilmente configurare le attività per la distribuzione, i comandi Artisan e altro. Attualmente, Envoy supporta solo i sistemi operativi Mac e Linux.

Questo pacchetto può essere trovato su Github .

Passaporto

Laravel facilita già l'esecuzione dell'autenticazione tramite moduli di accesso tradizionali, ma per quanto riguarda le API? Le API utilizzano in genere token per autenticare gli utenti e non mantengono lo stato di sessione tra le richieste. Laravel rende l'autenticazione API un gioco da ragazzi con Laravel Passport, che fornisce un'implementazione completa del server OAuth2 per la tua applicazione Laravel in pochi minuti.

Ulteriori informazioni su questo pacchetto sono disponibili qui .

esploratore

Laravel Scout offre una soluzione semplice e basata su driver per l'aggiunta della ricerca full-text ai modelli Eloquent. Usando osservatori di modelli, Scout manterrà automaticamente gli indici di ricerca in sincronia con i tuoi record di Eloquent.

Attualmente, Scout viene fornito con un autista Algolia; tuttavia, scrivere driver personalizzati è semplice e sei libero di estendere Scout con le tue implementazioni di ricerca.

Ulteriori informazioni su questo pacchetto sono disponibili qui .

persona mondana

Laravel Socialite fornisce un'interfaccia espressiva e fluente all'autenticazione OAuth con Facebook, Twitter, Google, LinkedIn, GitHub e Bitbucket. Gestisce quasi tutto il codice di autenticazione sociale boilerplate che stai temendo di scrivere.

Questo pacchetto può essere trovato su Github .

Leggi Pacchetti Laravel online: https://riptutorial.com/it/laravel/topic/8001/pacchetti-laravel

Capitolo 44: paginatura

Examples

Impaginazione in Laravel

In altri quadri l'impaginazione è mal di testa. Laravel rende più semplice, può generare l'impaginazione aggiungendo poche righe di codice in Controller e View.

Uso di base

Esistono molti modi per impaginare gli elementi, ma il più semplice è l'utilizzo del metodo paginate sul generatore di query o su una query di Eloquent . Laravel out of the box si occupa di impostare il limite e l'offset in base alla pagina corrente visualizzata dall'utente. Per impostazione predefinita, la pagina corrente viene rilevata dal valore dell'argomento della stringa di query della pagina sulla richiesta HTTP. E di sicuro, questo valore viene rilevato automaticamente da Laravel e inserito nei collegamenti generati da paginator.

Ora diciamo che vogliamo chiamare il metodo `paginate` su query. Nel nostro esempio l'argomento passato a paginare è il numero di elementi che si desidera visualizzare "per pagina". Nel nostro caso, diciamo che vogliamo visualizzare 10 elementi per pagina.

```
<?php

namespace App\Http\Controllers;

use DB;
use App\Http\Controllers\Controller;

class UserController extends Controller
{
    /**
     * Show all of the users for the application.
     *
     * @return Response
     */
    public function index()
    {
        $users = DB::table('users')->paginate(10);

        return view('user.index', ['users' => $users]);
    }
}
```

> **Nota:** attualmente, le operazioni di impaginazione che utilizzano un'istruzione `groupBy` non possono essere eseguite in modo efficiente da Laravel. Se è necessario utilizzare un `groupBy` con un set di risultati impaginato, si consiglia di interrogare il database e creare manualmente un cercapersone.

Impaginazione semplice

Diciamo che vuoi solo visualizzare i link Avanti e Indietro sulla tua vista di impaginazione. Laravel ti offre questa opzione usando il metodo `simplePaginate` .

```
$users = DB::table('users')->simplePaginate(10);
```

Visualizzazione dei risultati in una vista

Ora consente di visualizzare l'impaginazione in vista. In realtà quando chiamate i metodi `paginate` o `simplePaginate` su query Eloquent, ricevete un'istanza cercapersone. Quando viene chiamato il metodo paginate, si riceve un'istanza di `Illuminate\Pagination\LengthAwarePaginator` , mentre quando si chiama il metodo `simplePaginate` , si riceve un'istanza di `Illuminate\Pagination\Paginator` . Queste istanze / oggetti vengono forniti con diversi metodi che spiegano il set di risultati. Inoltre, oltre a questi metodi di helpers, le istanze di impaginatore sono iteratori e possono essere collegate in loop come una matrice.

Una volta ricevuti i risultati, è possibile rendere facilmente i collegamenti della pagina utilizzando blade

```
<div class="container">
    @foreach ($users as $user)
        {{ $user->name }}
    @endforeach
</div>

{{ $users->links() }}
```

Il metodo dei `links` renderizza automaticamente i collegamenti ad altre pagine nella serie di risultati. Ciascuno di questi collegamenti conterrà il numero di pagina specifico, ovvero la variabile di stringa della query della `?page` . L'HTML generato dal metodo links è perfettamente compatibile con il framework CSS Bootstrap.

Modifica delle visualizzazioni di impaginazione

Durante l'utilizzo della paginazione di laravel sei libero di usare le tue visualizzazioni personalizzate. Quindi, quando chiami il metodo dei link su un'istanza di impaginatore, passa il nome della vista come primo argomento al metodo come:

```
{{ $paginator->links('view.name') }}
```

o

È possibile personalizzare le visualizzazioni di impaginazione esportandole nella directory `resources/views/vendor` utilizzando il fornitore: comando di pubblicazione:

```
php artisan vendor:publish --tag=laravel-pagination
```

Questo comando posiziona le viste nella directory `resources/views/vendor/pagination` . Il file `default.blade.php` all'interno di questa directory corrisponde alla vista dell'impostazione predefinita. Modifica questo file per modificare l'HTML di impaginazione.

Leggi paginatura online: https://riptutorial.com/it/laravel/topic/2359/paginatura

Capitolo 45: persona mondana

Examples

Installazione

```
composer require laravel/socialite
```

Questa installazione presuppone che tu stia utilizzando Composer per gestire le tue dipendenze con Laravel, che è un ottimo modo per affrontarlo.

Configurazione

Nel tuo `config\services.php` puoi aggiungere il seguente codice

```
'facebook' => [
    'client_id' => 'your-facebook-app-id',
    'client_secret' => 'your-facebook-app-secret',
    'redirect' => 'http://your-callback-url',
],
```

Dovrai anche aggiungere il Provider al tuo `config\app.php`

Cerca la matrice `'providers' => []` e, alla fine, aggiungi quanto segue

```
'providers' => [
      ...

      Laravel\Socialite\SocialiteServiceProvider::class,
]
```

Una confezione è fornita anche con il pacchetto. Se vuoi farne un uso assicurati che l'array di `aliases` (anche nella tua `config\app.php`) abbia il seguente codice

```
'aliases' => [
    ....
    'Socialite' => Laravel\Socialite\Facades\Socialite::class,
]
```

Uso di base - Facciata

```
return Socialite::driver('facebook')->redirect();
```

Questo reindirizzerà una richiesta in arrivo all'URL appropriato per essere autenticato. Un esempio di base sarebbe in un controller

```
<?php
```

```
namespace App\Http\Controllers\Auth;

use Socialite;

class AuthenticationController extends Controller {

    /**
     * Redirects the User to the Facebook page to get authorization.
     *
     * @return Response
     */
    public function facebook() {
        return Socialite::driver('facebook')->redirect();
    }

}
```

assicurati che il tuo file `app\Http\routes.php` abbia un percorso per consentire una richiesta in arrivo qui.

```
Route::get('facebook', 'App\Http\Controllers\Auth\AuthenticationController@facebook');
```

Uso di base - Iniezione delle dipendenze

```
/**
 * LoginController constructor.
 * @param Socialite $socialite
 */
public function __construct(Socialite $socialite) {
    $this->socialite = $socialite;
}
```

All'interno del costruttore del tuo Controller, ora sei in grado di iniettare la classe `Socialite` che ti aiuterà a gestire i login con i social network. Questo sostituirà l'uso della facciata.

```
/**
 * Redirects the User to the Facebook page to get authorization.
 *
 * @return Response
 */
public function facebook() {
    return $this->socialite->driver('facebook')->redirect();
}
```

Socialite per API - Stateless

```
public function facebook() {
    return $this->socialite->driver('facebook')->stateless()->redirect()->getTargetUrl();
}
```

Ciò restituirà l'URL che il consumatore dell'API deve fornire all'utente finale per ottenere l'autorizzazione da Facebook.

Leggi persona mondana online: https://riptutorial.com/it/laravel/topic/1312/persona-mondana

Capitolo 46: Pianificazione delle attività

Examples

Creare un'attività

Puoi creare un'attività (Console Command) in Laravel usando Artisan. Dalla tua riga di comando:

```
php artisan make:console MyTaskName
```

Questo crea il file in **app / Console / Comandi / MyTaskName.php** . Sembrerà così:

```
<?php

namespace App\Console\Commands;

use Illuminate\Console\Command;

class MyTaskName extends Command
{
    /**
     * The name and signature of the console command.
     *
     * @var string
     */
    protected $signature = 'command:name';

    /**
     * The console command description.
     *
     * @var string
     */
    protected $description = 'Command description';

    /**
     * Create a new command instance.
     *
     * @return void
     */
    public function __construct()
    {
        parent::__construct();
    }

    /**
     * Execute the console command.
     *
     * @return mixed
     */
    public function handle()
    {
        //
    }
}
```

Alcune parti importanti di questa definizione sono:

- La proprietà `$signature` è ciò che identifica il tuo comando. Sarai in grado di eseguire questo comando in un secondo momento tramite la riga di comando usando Artisan eseguendo il `php artisan command:name` (dove `command:name` corrisponde alla `$signature` del tuo comando)
- La proprietà `$description` è l'help / display di utilizzo di Artisan accanto al tuo comando quando viene reso disponibile.
- Il metodo `handle()` è dove scrivi il codice per il tuo comando.

Alla fine, il tuo compito sarà reso disponibile alla linea di comando attraverso Artisan. The `protected $signature = 'command:name';` la proprietà di questa classe è ciò che useresti per eseguirla.

Rendere disponibile un compito

Puoi rendere un compito disponibile ad Artisan e alla tua applicazione nel file **app / Console / Kernel.php** .

La classe `Kernel` contiene un array chiamato `$commands` che rende i tuoi comandi disponibili per la tua applicazione.

Aggiungi il tuo comando a questo array, per renderlo disponibile ad Artisan e alla tua applicazione.

```
<?php

namespace App\Console;

use Illuminate\Console\Scheduling\Schedule;
use Illuminate\Foundation\Console\Kernel as ConsoleKernel;

class Kernel extends ConsoleKernel
{
    /**
     * The Artisan commands provided by your application.
     *
     * @var array
     */
    protected $commands = [
        Commands\Inspire::class,
        Commands\MyTaskName::class // This line makes MyTaskName available
    ];

    /**
     * Define the application's command schedule.
     *
     * @param  \Illuminate\Console\Scheduling\Schedule  $schedule
     * @return void
     */
    protected function schedule(Schedule $schedule)
    {

    }
}
```

Una volta fatto, puoi ora accedere al tuo comando tramite la riga di comando, usando Artisan.

Supponendo che il tuo comando abbia la proprietà `$signature` impostata su `my:task` , puoi eseguire il seguente comando per eseguire l'attività:

```
php artisan my:task
```

Pianificazione del tuo compito

Quando il comando è reso disponibile per la tua applicazione, puoi utilizzare Laravel per programmarlo per l'esecuzione a intervalli predefiniti, proprio come faresti con un CRON.

Nel file **App / Console / Kernel.php** troverai un metodo di `schedule` che puoi utilizzare per pianificare il tuo compito.

```
<?php

namespace App\Console;

use Illuminate\Console\Scheduling\Schedule;
use Illuminate\Foundation\Console\Kernel as ConsoleKernel;

class Kernel extends ConsoleKernel
{
    /**
     * The Artisan commands provided by your application.
     *
     * @var array
     */
    protected $commands = [
        Commands\Inspire::class,
        Commands\MyTaskName::class
    ];

    /**
     * Define the application's command schedule.
     *
     * @param  \Illuminate\Console\Scheduling\Schedule  $schedule
     * @return void
     */
    protected function schedule(Schedule $schedule)
    {
        $schedule->command('my:task')->everyMinute();
        // $schedule->command('my:task')->everyFiveMinutes();
        // $schedule->command('my:task')->daily();
        // $schedule->command('my:task')->monthly();
        // $schedule->command('my:task')->sundays();
    }
}
```

Supponendo che la `$signature` della tua attività sia `my:task` puoi programmarla come mostrato sopra, usando l'oggetto `Schedule $schedule` . Laravel offre molti modi diversi per programmare il tuo comando, come mostrato nelle righe sopra commentate.

Impostazione dello scheduler da eseguire

Lo scheduler può essere eseguito utilizzando il comando:

```
php artisan schedule:run
```

Lo scheduler deve essere eseguito ogni minuto per funzionare correttamente. È possibile impostarlo creando un cron job con la seguente riga, che esegue lo scheduler ogni minuto in background.

```
* * * * * php /path/to/artisan schedule:run >> /dev/null 2>&1
```

Leggi Pianificazione delle attività online: https://riptutorial.com/it/laravel/topic/4026/pianificazione-delle-attivita

Capitolo 47: Politiche

Examples

Creazione di politiche

Poiché la definizione di tutta la logica di autorizzazione in `AuthServiceProvider` potrebbe diventare ingombrante in applicazioni di grandi dimensioni, Laravel consente di suddividere la logica di autorizzazione in classi "Politica". Le politiche sono semplici classi PHP che raggruppano la logica di autorizzazione in base alla risorsa che autorizzano.

È possibile generare una politica utilizzando il comando `make:policy` artisan. La politica generata verrà inserita nella directory `app/Policies` :

```
php artisan make:policy PostPolicy
```

Leggi Politiche online: https://riptutorial.com/it/laravel/topic/7344/politiche

Capitolo 48: posta

Examples

Esempio di base

Puoi configurare Mail semplicemente aggiungendo / modificando queste righe nel file **.ENV** dell'app con i dettagli di accesso del tuo provider email, ad esempio per utilizzarlo con gmail puoi utilizzare:

```
MAIL_DRIVER=smtp
MAIL_HOST=smtp.gmail.com
MAIL_PORT=587
MAIL_USERNAME=yourEmail@gmail.com
MAIL_PASSWORD=yourPassword
MAIL_ENCRYPTION=tls
```

Quindi puoi iniziare a inviare e-mail usando Mail, ad esempio:

```
$variable = 'Hello world!'; // A variable which can be use inside email blade template.
Mail::send('your.blade.file', ['variable' => $variable], function ($message) {
            $message->from('john@doe.com');
            $message->sender('john@doe.com');
            $message->to(foo@bar.com);
            $message->subject('Hello World');
        });
```

Leggi posta online: https://riptutorial.com/it/laravel/topic/8014/posta

Capitolo 49: Problemi comuni e soluzioni rapide

introduzione

Questa sezione elenca i problemi comuni e gli sviluppatori di soluzioni rapide (in particolare i principianti).

Examples

Eccezione TokenMisMatch

Questa eccezione si ottiene principalmente con l'invio di moduli. Laravel protegge l'applicazione da `CSRF` e convalida ogni richiesta e garantisce che la richiesta provenga dall'interno dell'applicazione. Questa convalida viene eseguita utilizzando un `token` . Se questo token non corrisponde, viene generata questa eccezione.

Soluzione rapida

Aggiungi questo all'interno del tuo elemento del modulo. Questo invia `csrf_token` generato da laravel insieme ad altri dati del modulo in modo che laravel sappia che la tua richiesta è valida

```
<input type="hidden" name="_token" value="{{ csrf_token() }}">
```

Leggi Problemi comuni e soluzioni rapide online:
https://riptutorial.com/it/laravel/topic/9971/problemi-comuni-e-soluzioni-rapide

Capitolo 50: quadro di lumen

Examples

Iniziare con Lumen

Nell'esempio seguente viene illustrato l'uso di `Lumen` negli ambienti `WAMP / MAMP / LAMP` .

Per lavorare con `Lumen` è necessario impostare prima un paio di cose.

- Compositore
- PHPUnit
- git (non richiesto ma fortemente raccomandato)

Supponendo di aver installato tutti questi tre componenti (almeno è necessario comporre il compositore), per prima cosa accedere al documento root dei server Web utilizzando il terminale. MacOSX e Linux hanno un ottimo terminale. Puoi usare `git bash` (che in realtà è `mingw32` o `mingw64`) in windows.

```
$ cd path/to/your/document/root
```

Quindi è necessario utilizzare compose per installare e creare il progetto `Lumen` . Esegui il seguente comando.

```
$ composer create-project laravel/lumen=~5.2.0 --prefer-dist lumen-project
$ cd lumen-project
```

`lumen-app` nel codice sopra è il nome della cartella. Puoi cambiarlo come preferisci. Ora è necessario impostare l'host virtuale in modo che punti al `path/to/document/root/lumen-project/public` folder. Supponiamo che abbia mappato `http://lumen-project.local` a questa cartella. Ora se vai a questo URL dovresti vedere un messaggio come segue (a seconda della versione `Lumen` installata, nel mio caso era 5.4.4) -

```
Lumen (5.4.4) (Laravel Components 5.4.*)
```

Se apri il file `lumen-project/routers/web.php` , dovresti vedere quanto segue:

```
$app->get('/', function () use($app) {
    return $app->version();
});
```

Congratulazioni! Ora hai un'installazione `Lumen` funzionante. No, puoi estendere questa app per ascoltare i tuoi endpoint personalizzati.

Leggi quadro di lumen online: https://riptutorial.com/it/laravel/topic/9221/quadro-di-lumen

Capitolo 51: Richiesta / e di modulo

introduzione

Le richieste personalizzate (o richieste di modulo) sono utili in situazioni in cui si desidera **autorizzare** e **convalidare** una richiesta prima di colpire il metodo del controllore.

Si può pensare a due usi pratici, **creando** e **aggiornando** un record mentre ogni azione ha un diverso insieme di regole di validazione (o autorizzazione).

L'utilizzo delle richieste di modulo è banale, si deve digitare la classe di richiesta nel metodo.

Sintassi

- php artisan make: richiesta name_of_request

Osservazioni

Le richieste sono utili quando si separa la convalida dal Controller. Consente inoltre di verificare se la richiesta è autorizzata.

Examples

Creazione di richieste

```
php artisan make:request StoreUserRequest

php artisan make:request UpdateUserRequest
```

> **Nota** : puoi anche considerare l'utilizzo di nomi come **StoreUser** o **UpdateUser** (senza Appendice di **richiesta**) poiché i tuoiRichiesta sono inseriti nell'app di cartella `app/Http/Requests/` .

Utilizzando la richiesta di modulo

Diciamo continuate con l'esempio utente (potreste avere un controller con metodo store e metodo di aggiornamento). Per utilizzare FormRequests, si utilizza type-suggerendo la richiesta specifica.

```
...

public function store(App\Http\Requests\StoreRequest $request, App\User $user) {
    //by type-hinting the request class, Laravel "runs" StoreRequest
    //before actual method store is hit

    //logic that handles storing new user
    //(both email and password has to be in $fillable property of User model
```

```
    $user->create($request->only(['email', 'password']));
    return redirect()->back();
}

...

public function update(App\Http\Requests\UpdateRequest $request, App\User $users, $id) {
    //by type-hinting the request class, Laravel "runs" UpdateRequest
    //before actual method update is hit

    //logic that handles updating a user
    //(both email and password has to be in $fillable property of User model
    $user = $users->findOrFail($id);
    $user->update($request->only(['password']));
    return redirect()->back();
}
```

Gestione dei reindirizzamenti dopo la convalida

A volte potresti voler avere qualche login per determinare dove l'utente viene reindirizzato dopo aver inviato un modulo. Le richieste di modulo offrono una varietà di modi.

Di default ci sono 3 variabili dichiarate nella richiesta `$redirect` , `$redirectRoute` e `$redirectAction` .

Oltre a queste 3 variabili, è possibile eseguire l'override del gestore di reindirizzamento principale `getRedirectUrl()` .

Di seguito viene fornita una richiesta di esempio che spiega cosa è possibile fare.

```
<?php namespace App;

use Illuminate\Foundation\Http\FormRequest as Request;

class SampleRequest extends Request {

    // Redirect to the given url
    public $redirect;

    // Redirect to a given route
    public $redirectRoute;

    // Redirect to a given action
    public $redirectAction;


    /**
     * Get the URL to redirect to on a validation error.
     *
     * @return string
     */
    protected function getRedirectUrl()
    {

        // If no path is given for `url()` it will return a new instance of
`Illuminate\Routing\UrlGenerator`

        // If your form is down the page for example you can redirect to a hash
```

```
        return url()->previous() . '#contact';

        //`url()` provides several methods you can chain such as

        // Get the current URL
        return url()->current();

        // Get the full URL of the current request
        return url()->full();

        // Go back
        return url()->previous();

        // Or just redirect back
        return redirect()->back();
    }


    /**
     * Get the validation rules that apply to the request.
     *
     * @return array
     */
    public function rules()
    {
        return [];
    }

    /**
     * Determine if the user is authorized to make this request.
     *
     * @return bool
     */
    public function authorize()
    {
        return true;
    }
}
```

Leggi Richiesta / e di modulo online: https://riptutorial.com/it/laravel/topic/6329/richiesta---e-di-modulo

Capitolo 52: Richiesta di dominio incrociato

Examples

introduzione

A volte abbiamo bisogno di una richiesta cross-domain per le nostre API in laravel. Dobbiamo aggiungere intestazioni appropriate per completare correttamente la richiesta di dominio incrociato. Quindi dobbiamo assicurarci che qualunque intestazione che stiamo aggiungendo sia accurata, altrimenti le nostre API diventano vulnerabili. Per aggiungere intestazioni è necessario aggiungere il middleware in laravel che aggiungerà le intestazioni appropriate e inoltrerà le richieste.

CorsHeaders

```
<?php

namespace laravel\Http\Middleware;

class CorsHeaders
{
  /**
   * This must be executed _before_ the controller action since _after_ middleware isn't
executed when exceptions are thrown and caught by global handlers.
   *
   * @param $request
   * @param \Closure $next
   * @param string [$checkWhitelist] true or false Is a string b/c of the way the arguments
are supplied.
   * @return mixed
   */
  public function handle($request, \Closure $next, $checkWhitelist = 'true')
  {
    if ($checkWhitelist == 'true') {
      // Make sure the request origin domain matches one of ours before sending CORS response
headers.
      $origin = $request->header('Origin');
      $matches = [];
      preg_match('/^(https?:\/\/)?([a-zA-Z\d]+\.)*(?<domain>[a-zA-Z\d-\.]+\.[a-z]{2,10})$/',
$origin, $matches);

      if (isset($matches['domain']) && in_array($matches['domain'], ['yoursite.com']) {
        header('Access-Control-Allow-Origin: ' . $origin);
        header('Access-Control-Expose-Headers: Location');
        header('Access-Control-Allow-Credentials: true');

        // If a preflight request comes then add appropriate headers
        if ($request->method() === 'OPTIONS') {
          header('Access-Control-Allow-Methods: GET, POST, PUT, OPTIONS, DELETE, PATCH');
          header('Access-Control-Allow-Headers: ' . $request->header('Access-Control-Request-
Headers'));
            // 20 days
          header('Access-Control-Max-Age: 1728000');
        }
```

```
        }
    } else {
        header('Access-Control-Allow-Origin: *');
    }

    return $next($request);
  }
}
```

Leggi Richiesta di dominio incrociato online: https://riptutorial.com/it/laravel/topic/7425/richiesta-di-dominio-incrociato

Capitolo 53: richieste

Examples

Ottenere input

Il modo principale per ottenere input consiste nell'iniezione di `Illuminate\Http\Request` nel controller, dopo che ci sono numerosi modi per accedere ai dati, 4 dei quali sono nell'esempio seguente.

```
<?php
namespace App\Http\Controllers;

use Illuminate\Http\Request;

class UserController extends Controller
{
    public function store(Request $request)
    {
        // Returns the username value
        $name = $request->input('username');

        // Returns the username value
        $name = $request->username;

        // Returns the username value
        $name = request('username');

        // Returns the username value again
        $name = request()->username;

    }

}
```

Quando si utilizza la funzione di `input` , è anche possibile aggiungere un valore predefinito per quando l'input della richiesta non è disponibile

```
$name = $request->input('username', 'John Doe');
```

Leggi richieste online: https://riptutorial.com/it/laravel/topic/3076/richieste

Capitolo 54: richieste

Examples

Ottieni un'istanza della richiesta HTTP

Per ottenere un'istanza di una richiesta HTTP, la classe `Illuminate\Http\Request` deve essere di tipo hint nel costruttore o nel metodo del controller.

Codice di esempio:

```
 <?php

namespace App\Http\Controllers;

/* Here how we illuminate the request class in controller */
use Illuminate\Http\Request;

use Illuminate\Routing\Controller;

class PostController extends Controller
{
 /**
  * Store a new post.
  *
  * @param  Request  $request
  * @return Response
  */
 public function store(Request $request)
 {
     $name = $request->input('post_title');

     /*
     * so typecasting Request class in our method like above avails the
     * HTTP GET/POST/PUT etc method params in the controller to use and
     * manipulate
     */
 }
}
```

Richiedi l'istanza con altri parametri dalle rotte nel metodo controller

A volte è necessario accettare i parametri del percorso e accedere ai parametri della richiesta HTTP. Possiamo ancora suggerire la classe Requests nel controller di laravel e ottenere ciò come spiegato di seguito

Ad esempio, abbiamo un percorso che aggiorna un determinato post come questo (passaggio post id id route)

```
Route::put('post/{id}', 'PostController@update');
```

Inoltre, poiché l'utente ha modificato altri campi del modulo di modifica, sarà disponibile nella richiesta HTTP

Ecco come accedere a entrambi nel nostro metodo

```
    public function update(Request $request,$id){
        //This way we have $id param from route and $request as an HTTP Request object

    }
```

Leggi richieste online: https://riptutorial.com/it/laravel/topic/4929/richieste

Capitolo 55: Rimuovi pubblico dall'URL in laravel

introduzione

Come rimuovere `public` dall'URL in Laravel, ci sono molte risposte su Internet ma il modo più semplice è descritto di seguito

Examples

Come farlo?

Attenersi alla seguente procedura per rimuovere `public` dall'URL

1. Copia il file `.htaccess` da `/public` directory a `Laravel/project` cartella principale del `Laravel/project` .
2. Rinominare il `server.php` nella cartella `Laravel/project` root su `index.php` .

Saluti sarai buono ora.

Nota: è stato testato su *Laravel 4.2* , *Laravel 5.1* , *Laravel 5.2* , *Laravel 5.3* .

Penso che questo sia il modo più semplice per rimuovere il `public` dall'URL.

Rimuovi il pubblico dall'URL

1. Rinominare il `server.php` in `index.php`
2. Copia il `.htaccess` dalla cartella `public` alla cartella `root`
3. Modifica `.htaccess` un po 'come segue per la statica:

```
RewriteEngine On

RewriteCond %{REQUEST_FILENAME} !-d
RewriteRule ^(.*)/$ /$1 [L,R=301]

RewriteCond %{REQUEST_URI} !(\.css|\.js|\.png|\.jpg|\.gif|robots\.txt)$ [NC]
RewriteCond %{REQUEST_FILENAME} !-d
RewriteCond %{REQUEST_FILENAME} !-f
RewriteRule ^ index.php [L]

RewriteCond %{REQUEST_FILENAME} !-d
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_URI} !^/public/
RewriteRule ^(css|js|images)/(.*)$ public/$1/$2 [L,NC]
```

A volte ho usato questo metodo per rimuovere `public` URL del modulo `public` .

Leggi Rimuovi pubblico dall'URL in laravel online:

https://riptutorial.com/it/laravel/topic/9786/rimuovi-pubblico-dall-url-in-laravel

Capitolo 56: Route Model Binding

Examples

Legame implicito

Laravel risolve automaticamente i modelli Eloquent definiti in rotte o azioni del controllore i cui nomi di variabili corrispondono a un nome di segmento di percorso. Per esempio:

```
Route::get('api/users/{user}', function (App\User $user) {
    return $user->email;
});
```

In questo esempio, poiché la variabile utente Eloquent $ definita sulla route corrisponde al segmento {user} nell'URI della rotta, Laravel inietterà automaticamente l'istanza del modello che ha un ID corrispondente al valore corrispondente dall'URI della richiesta. Se nel database non viene trovata un'istanza di modello corrispondente, verrà generata automaticamente una risposta HTTP 404.

Se il nome della tabella del modello è composto da più parole, per rendere funzionante il binding del modello implicito la variabile di input dovrebbe essere in minuscolo;
Ad esempio, se l'utente può eseguire un qualche tipo di *azione* e vogliamo accedere a questa azione, il percorso sarà:

```
Route::get('api/useractions/{useraction}', function (App\UserAction $useraction) {
    return $useraction->description;
});
```

Legame esplicito

Per registrare un'associazione esplicita, utilizzare il metodo del modello del router per specificare la classe per un determinato parametro. È necessario definire i collegamenti del modello esplicito nel metodo di avvio della classe RouteServiceProvider

```
public function boot()
{
   parent::boot();

   Route::model('user', App\User::class);
}
```

Successivamente, possiamo definire una rotta che contiene il parametro {utente}.

```
$router->get('profile/{user}', function(App\User $user) {

});
```

Dal momento che abbiamo associato tutti `{user}` parametri `{user}` al modello `App\User` , un'istanza User verrà inserita nella route. Quindi, per esempio, una richiesta al `profile/1` inietterà l'istanza Utente dal database che ha un **ID** di **1** .

Se un'istanza del modello corrispondente non viene trovata nel database, verrà generata automaticamente una risposta **HTTP 404** .

Leggi Route Model Binding online: https://riptutorial.com/it/laravel/topic/7098/route-model-binding

Capitolo 57: Routing

Examples

Routing di base

Il routing definisce una mappa tra i metodi HTTP e gli URI su un lato e le azioni sull'altro. Le rotte sono normalmente scritte nel file `app/Http/routes.php` .

Nella sua forma più semplice, una rotta viene definita chiamando il corrispondente metodo HTTP sulla facciata Route, passando come parametri una stringa che corrisponde all'URI (relativo alla root dell'applicazione) e un callback.

Ad esempio: un percorso verso l'URI radice del sito che restituisce una vista a `home` assomiglia a questo:

```
Route::get('/', function() {
    return view('home');
});
```

Un percorso per una richiesta di posta che richiama semplicemente le variabili del post:

```
Route::post('submit', function() {
   return Input::all();
});

//or

Route::post('submit', function(\Illuminate\Http\Request $request) {
   return $request->all();
});
```

Rotte che puntano a un metodo di controllo

Invece di definire il callback in linea, il percorso può fare riferimento a un metodo controller nella sintassi [ControllerClassName @ Method]:

```
Route::get('login', 'LoginController@index');
```

Un percorso per più verbi

Il metodo di `match` può essere utilizzato per abbinare una matrice di metodi HTTP per una determinata rotta:

```
Route::match(['GET', 'POST'], '/', 'LoginController@index');
```

Inoltre puoi utilizzare `all` per abbinare qualsiasi metodo HTTP per una determinata rotta:

```
Route::all('login', 'LoginController@index');
```

Instradare i gruppi

Le rotte possono essere raggruppate per evitare la ripetizione del codice.

Diciamo che tutti gli URI con un prefisso di `/admin` utilizzano un certo middleware chiamato `admin` e vivono tutti nello spazio dei nomi `App\Http\Controllers\Admin` .

Un modo pulito di rappresentarlo utilizzando i gruppi di percorsi è il seguente:

```
Route::group([
    'namespace' => 'Admin',
    'middleware' => 'admin',
    'prefix' => 'admin'
    ], function () {

        // something.dev/admin
        // 'App\Http\Controllers\Admin\IndexController'
        // Uses admin middleware
        Route::get('/', ['uses' => 'IndexController@index']);

        // something.dev/admin/logs
        // 'App\Http\Controllers\Admin\LogsController'
        // Uses admin middleware
        Route::get('/logs', ['uses' => 'LogsController@index']);
});
```

Percorso con nome

I percorsi denominati vengono utilizzati per generare un URL o reindirizzare a un percorso specifico. Il vantaggio dell'utilizzo di una rotta denominata è, se in futuro cambieremo l'URI di una rotta, non avremmo bisogno di cambiare l'URL o i reindirizzamenti che puntano a quella rotta se stiamo usando una rotta con nome. Ma se i collegamenti sono stati generati usando l'url [es. `url('/admin/login')`], quindi dovremmo cambiare ovunque dove viene utilizzato.

I percorsi denominati vengono creati utilizzando `as` chiave di array

```
Route::get('login', ['as' => 'loginPage', 'uses' => 'LoginController@index']);
```

o usando il `name` metodo

```
Route::get('login', 'LoginController@index')->name('loginPage');
```

Genera URL usando la rotta denominata

Per generare un url usando il nome del percorso

```
$url = route('loginPage');
```

Se si sta utilizzando il nome della rotta per il reindirizzamento

```
$redirect = Redirect::route('loginPage');
```

Parametri di percorso

È possibile utilizzare i parametri del percorso per ottenere la parte del segmento URI. È possibile definire un parametro / i di percorso facoltativo o richiesto durante la creazione di un percorso. I parametri opzionali hanno un `?` aggiunto alla fine del nome del parametro. Questo nome è racchiuso tra parentesi graffe `{}`

Parametro opzionale

```
Route::get('profile/{id?}', ['as' => 'viewProfile', 'uses' => 'ProfileController@view']);
```

Questo percorso è accessibile da `domain.com/profile/23` dove 23 è il parametro id. In questo esempio l' `id` viene passato come parametro nel metodo `view` di `ProfileController` . Poiché si tratta di un parametro facoltativo, l'accesso a `domain.com/profile` funziona `domain.com/profile` .

Parametro richiesto

```
Route::get('profile/{id}', ['as' => 'viewProfile', 'uses' => 'ProfileController@view']);
```

Si noti che il nome del parametro richiesto non ha un `?` alla fine del nome del parametro.

Accesso al parametro nel controller

Nel tuo controller, il tuo metodo di visualizzazione prende un parametro con lo **stesso** nome di `routes.php` e può essere usato come una variabile normale. Laravel si prende cura di iniettare il valore:

```
public function view($id){
    echo $id;
}
```

Prendi tutti i percorsi

Se vuoi catturare tutti i percorsi, puoi usare un'espressione regolare come mostrato:

```
Route::any('{catchall}', 'CatchAllController@handle')->where('catchall', '.*');
```

Importante: se hai altri percorsi e non vuoi che il catch-all interferisca, dovresti metterlo alla fine. Per esempio:

Cattura tutti i percorsi tranne quelli già definiti

```
Route::get('login',  'AuthController@login');
Route::get('logout', 'AuthController@logout');
Route::get('home',   'HomeController@home');

// The catch-all will match anything except the previous defined routes.
Route::any('{catchall}', 'CatchAllController@handle')->where('catchall', '.*');
```

I percorsi sono abbinati nell'ordine in cui sono dichiarati

Questo è un trucco comune con le rotte di Laravel. I percorsi sono abbinati nell'ordine in cui sono dichiarati. La prima route corrispondente è quella che viene utilizzata.

Questo esempio funzionerà come previsto:

```
Route::get('/posts/{postId}/comments/{commentId}', 'CommentController@show');
Route::get('/posts/{postId}', 'PostController@show');
```

Una richiesta get a `/posts/1/comments/1` invocherà `CommentController@show` . Una richiesta get a `/posts/1` invocherà `PostController@show` .

Tuttavia, questo esempio non funzionerà nello stesso modo:

```
Route::get('/posts/{postId}', 'PostController@show');
Route::get('/posts/{postId}/comments/{commentId}', 'CommentController@show');
```

Una richiesta get a `/posts/1/comments/1` invocherà `PostController@show` . Una richiesta get a `/posts/1` invocherà `PostController@show` .

Poiché Laravel utilizza il primo percorso abbinato, la richiesta a `/posts/1/comments/1` corrisponde a `Route::get('/posts/{postId}', 'PostController@show');` e assegna la variabile `$postId` al valore `1/comments/1` . Ciò significa che `CommentController@show` non verrà mai richiamato.

Percorsi insensibili alle maiuscole

Le rotte in Laravel fanno distinzione tra maiuscole e minuscole. Significa che un percorso come

```
Route::get('login', ...);
```

abbinerà una richiesta GET a `/login` ma non corrisponderà a una richiesta GET a `/Login` .

Per rendere i tuoi percorsi senza distinzione tra maiuscole e minuscole, devi creare una nuova classe di validatore che corrisponda agli URL richiesti rispetto alle route definite. L'unica differenza tra il nuovo validatore e quello esistente è che aggiungerà il modificatore **i** alla fine dell'espressione regolare per il percorso compilato per attivare la corrispondenza senza distinzione tra maiuscole e minuscole.

```
<?php namespace Some\Namespace;

use Illuminate\Http\Request;
use Illuminate\Routing\Route;
use Illuminate\Routing\Matching\ValidatorInterface;

class CaseInsensitiveUriValidator implements ValidatorInterface
{
  public function matches(Route $route, Request $request)
  {
    $path = $request->path() == '/' ? '/' : '/'.$request->path();
    return preg_match(preg_replace('/$/','i', $route->getCompiled()->getRegex()),
rawurldecode($path));
  }
}
```

Affinché Laravel possa utilizzare il nuovo validatore, è necessario aggiornare l'elenco di dispositivi di corrispondenza utilizzati per associare l'URL a un percorso e sostituire l'UriValidator originale con il proprio.

Per fare ciò, aggiungi quanto segue all'inizio del tuo file routes.php:

```
<?php
use Illuminate\Routing\Route as IlluminateRoute;
use Your\Namespace\CaseInsensitiveUriValidator;
use Illuminate\Routing\Matching\UriValidator;

$validators = IlluminateRoute::getValidators();
$validators[] = new CaseInsensitiveUriValidator;
IlluminateRoute::$validators = array_filter($validators, function($validator) {
  return get_class($validator) != UriValidator::class;
});
```

Questo rimuoverà il validatore originale e aggiungerà il tuo alla lista dei validatori.

Leggi Routing online: https://riptutorial.com/it/laravel/topic/1284/routing

Capitolo 58: Semina

Osservazioni

Il seeding del database consente di inserire dati, dati di test generali nel database. Di default c'è una classe `DatabaseSeeder` sotto `database/seeds` .

L'esecuzione di seminatrici può essere eseguita con

```
php artisan db:seed
```

O se vuoi solo elaborare una singola classe

```
php artisan db:seed --class=TestSeederClass
```

Come per tutti i comandi artisan, è possibile accedere a una vasta gamma di metodi che possono essere trovati nella documentazione API

Examples

Inserimento di dati

Esistono diversi modi per inserire i dati:

Utilizzando la facciata DB

```
public function run()
{
    DB::table('users')
        ->insert([
            'name' => 'Taylor',
            'age'  => 21
        ]);
}
```

Tramite l'istanziazione di un modello

```
public function run()
{
    $user = new User;
    $user->name = 'Taylor';
    $user->save();
}
```

Utilizzando il metodo di creazione

```
public function run()
{
    User::create([
        'name' => 'Taylor',
        'age'  => 21
    ]);
}
```

Utilizzando la fabbrica

```
public function run()
{
    factory(App\User::class, 10)->create();
}
```

Semina && elimina vecchi dati e resetta l'auto-incremento

```
public function run()
{
    DB::table('users')->delete();
    DB::unprepared('ALTER TABLE users AUTO_INCREMENT=1;');
    factory(App\User::class, 200)->create();
}
```

Vedere l'esempio Persistente per ulteriori informazioni sull'inserimento / aggiornamento dei dati.

Chiamando altri seminatori

All'interno della classe `DatabaseSeeder` puoi chiamare altri seeder

```
$this->call(TestSeeder::class)
```

Questo ti permette di conservare un file dove puoi trovare facilmente i seeder. Tieni presente che è necessario prestare attenzione all'ordine delle chiamate in merito ai vincoli delle chiavi esterne. Non puoi fare riferimento a una tabella che non esiste ancora.

Creare una seminatrice

Per creare seminatrici, puoi usare il comando `make:seeder` Artisan. Tutti i seeders generati verranno posizionati nella directory `database/seeds` .

```
$ php artisan make:seeder MoviesTableSeeder
```

I seeders generati conterranno un metodo: `run` . Puoi inserire dati nel tuo database con questo metodo.

```
<?php

use Illuminate\Database\Seeder;
```

```
use Illuminate\Database\Eloquent\Model;

class MoviesTableSeeder extends Seeder
{
    /**
     * Run the database seeds.
     *
     * @return void
     */
    public function run()
    {
        App\Movie::create([
            'name' => 'A New Hope',
            'year' => '1977'
        ]);

        App\Movie::create([
            'name' => 'The Empire Strikes Back',
            'year' => '1980'
        ]);
    }
}
```

Generalmente vuoi chiamare tutti i tuoi seeder all'interno della classe `DatabaseSeeder` .

Una volta che hai finito di scrivere i seeder, usa il comando `db:seed` . Questo farà eseguire `DatabaseSeeder` s' `run` funzione.

```
$ php artisan db:seed
```

È anche possibile specificare di eseguire una classe di seminatrice specifica da eseguire singolarmente utilizzando l'opzione `--class` .

```
$ php artisan db:seed --class=UserSeeder
```

Se si desidera eseguire il rollback e rieseguire tutte le migrazioni, quindi riavviare:

```
$ php artisan migrate:refresh --seed
```

> Il comando `migrate:refresh --seed` è un collegamento a questi 3 comandi:
>
> ```
> $ php artisan migrate:reset     # rollback all migrations
> $ php artisan migrate           # run migrations
> $ php artisan db:seed           # run seeders
> ```

Rianimazione sicura

Potresti voler ri-seminare il tuo database senza influenzare i tuoi semi precedentemente creati. A tale scopo, puoi utilizzare `firstOrCreate` nella tua seminatrice:

```
EmployeeType::firstOrCreate([
    'type' => 'manager',
```

```
]);
```

Quindi puoi seminare il database:

```
php artisan db:seed
```

In seguito, se vuoi aggiungere un altro tipo di dipendente, puoi semplicemente aggiungere quello nuovo nello stesso file:

```
EmployeeType::firstOrCreate([
    'type' => 'manager',
]);
EmployeeType::firstOrCreate([
    'type' => 'secretary',
]);
```

E semina nuovamente il tuo database senza problemi:

```
php artisan db:seed
```

Notare che nella prima chiamata si sta recuperando il record ma non si fa nulla con esso.

Leggi Semina online: https://riptutorial.com/it/laravel/topic/3272/semina

Capitolo 59: Semina del database

Examples

Esecuzione di una seminatrice

Puoi aggiungere la tua nuova seminatrice alla classe DatabaseSeeder.

```
    /**
     * Run the database seeds.
     *
     * @return void
     */
    public function run()
    {
        $this->call(UserTableSeeder::class);
    }
```

Per eseguire una seminatrice di database, utilizzare il comando Artisan

```
php artisan db:seed
```

Questo eseguirà la classe DatabaseSeeder. Puoi anche scegliere di usare l'opzione `--class=` per specificare manualmente quale seminatrice eseguire.

* Nota, potrebbe essere necessario eseguire dumpautoload di composer se non è possibile trovare la classe Seeder. Ciò accade in genere se si crea manualmente una classe seminatrice invece di utilizzare il comando artisan.

Creare un seme

I semi del database sono memorizzati nella directory / database / semi. Puoi creare un seme usando un comando Artisan.

```
php artisan make:seed UserTableSeeder
```

In alternativa puoi creare una nuova classe che estende `Illuminate\Database\Seeder` . La classe deve avere una funzione pubblica chiamata `run()` .

Inserimento di dati utilizzando una seminatrice

Puoi fare riferimento a modelli in una seminatrice.

```
use DB;
use App\Models\User;

class UserTableSeeder extends Illuminate\Database\Seeder{
```

```
    public function run(){
        # Remove all existing entrie
        DB::table('users')->delete() ;
        User::create([
            'name' => 'Admin',
            'email' => 'admin@example.com',
            'password' => Hash::make('password')
        ]);

    }
}
```

Inserimento di dati con una fabbrica modello

Potresti usare le Fabbriche di modelli nei tuoi semi. Questo creerà 3 nuovi utenti.

```
use App\Models\User;

class UserTableSeeder extends Illuminate\Database\Seeder{

    public function run(){
        factory(User::class)->times(3)->create();
    }
}
```

Ad esempio, potresti anche voler definire campi specifici sul tuo seeding come una password. Questo creerà 3 utenti con la stessa password.

```
factory(User::class)->times(3)->create(['password' => '123456']);
```

Semina con MySQL Dump

Segui l'esempio precedente di creazione di un seme. In questo esempio viene utilizzato un dump MySQL per eseguire il seeding di una tabella nel database del progetto. La tabella deve essere creata prima della semina.

```
<?php

use Illuminate\Database\Seeder;

class UserTableSeeder extends Seeder
{

    /**
     * Run the database seeds.
     *
     * @return void
     */
     public function run()
     {
         $sql = file_get_contents(database_path() . '/seeds/users.sql');

         DB::statement($sql);
      }
```

```
}
```

Il nostro $ sql sarà il contenuto del nostro dump di users.sql. Il dump dovrebbe avere un'istruzione INSERT INTO. Spetterà a te dove conservi le discariche. Nell'esempio precedente, è memorizzato nella directory di progetto `\database\seeds` . Utilizzando la funzione helper di `database_path()` di laravel, `database_path()` e aggiungendo la directory e il nome del file del dump.

```
INSERT INTO `users` (`id`, `name`, `email`, `password`, `remember_token`, `created_at`,
`updated_at`) VALUES
(1, 'Jane', 'janeDoe@fakemail.com', 'superSecret', NULL, '2016-07-21 00:00:00', '2016-07-21
00:00:00'),
(2, 'John', 'johnny@fakemail.com', 'sup3rS3cr3t', NULL, '2016-07-21 00:00:00', '2016-07-21
00:00:00');
```

`DB::statement($sql)` eseguirà gli inserti una volta eseguita la Seeder. Come negli esempi precedenti, puoi inserire `UserTableSeeder` nella classe `DatabaseSeeder` fornita da laravel:

```
<?php

use Illuminate\Database\Seeder;

class DatabaseSeeder extends Seeder
{
    /**
     * Run the database seeds.
     *
     * @return void
     */
    public function run()
    {
        $this->call(UserTableSeeder::class);
    }
}
```

ed eseguire dalla CLI nella directory di progetto `php artisan db:seed` . Oppure puoi eseguire la Seminatrice per una singola classe usando `php artisan db:seed --class=UsersTableSeeder`

Usare faker e ModelFactories per generare semi

1) MODO SEMPLICE DI BASE

Le applicazioni basate su database spesso richiedono dati pre-seminati nel sistema per scopi di test e dimostrativi.

Per rendere tali dati, prima creare la classe seminatrice

ProductTableSeeder

```
use Faker\Factory as Faker;
use App\Product;

class ProductTableSeeder extends DatabaseSeeder {
```

```
public function run()
{
    $faker = $this->getFaker();

    for ($i = 0; $i < 10; $i++)
    {
        $name =         $faker->word;
        $image =        $faker->imageUrl;

        Modelname::create([
            'name' => $name,
            'image' => $image,
        ]);
      }
    }
 }
```

Per chiamare un essere in grado di eseguire una classe di seminatrice, è necessario chiamarlo dalla classe DatabaseSeeder, semplicemente passando il nome della seminatrice che si desidera eseguire:

usa Illuminate \ Database \ Seeder;

```
class DatabaseSeeder extends Seeder {

    protected $faker;

    public function getFaker() {
        if (empty($this->faker)) {
            $faker = Faker\Factory::create();
            $faker->addProvider(new Faker\Provider\Base($faker));
            $faker->addProvider(new Faker\Provider\Lorem($faker));
        }
        return $this->faker = $faker;
    }
    public function run() {
        $this->call(ProductTableSeeder::class);
    }
}
```

Non dimenticare di eseguire `$ composer dump-autoload` dopo aver creato la Seeder, dal momento che non vengono automaticamente caricati automaticamente dal compositore (a meno che tu non abbia creato la seminatrice tramite il comando artisan `$ php artisan make:seeder Name`)

Ora sei pronto per seminare eseguendo questo comando `php artisan db:seed`

2) UTILIZZO DELLE FABBRICHE DEL MODELLO

Prima di tutto devi definire un set predefinito di attributi per ciascun modello in `App/database/factories/ModelFactory.php`

Assunzione di un modello Utente come esempio, Ecco come appare ModelFactory

```
$factory->define(App\User::class, function (Faker\Generator $faker) {
    return [
        'name' => $faker->name,
```

```
        'email' => $faker->email,
        'password' => bcrypt(str_random(10)),
        'remember_token' => str_random(10),
    ];
});
```

Ora crea una seminatrice da tavolo `php artisan make:seeder UsersTableSeeder`

E aggiungi questo

```
public function run()
{
    factory(App\User::class, 100)->create()
}
```

quindi aggiungere questo al `DatabaseSeeder`

```
public function run()
{
    $this->call(UsersTableSeeder::class);
}
```

Questo semina la tabella con 100 record.

Leggi Semina del database online: https://riptutorial.com/it/laravel/topic/1118/semina-del-database

Capitolo 60: Servizi

Examples

introduzione

Laravel consente l'accesso a una varietà di classi chiamate Servizi. Alcuni servizi sono disponibili immediatamente, ma è possibile crearli da soli.

Un servizio può essere utilizzato in più file dell'applicazione, come i controller. Immaginiamo un servizio `OurService` implementa un metodo `getNumber()` restituendo un numero casuale:

```
class SomeController extends Controller {

    public function getRandomNumber()
    {
        return app(OurService::class)->getNumber();
    }
}
```

Per creare un servizio, è necessario solo creare una nuova classe:

```
# app/Services/OurService/OurService.php

<?php
namespace App\Services\OurService;

class OurService
{
    public function getNumber()
    {
        return rand();
    }
}
```

A questo punto, potresti già utilizzare questo servizio in un controller, ma dovresti creare un'istanza di un nuovo oggetto ogni volta che ne hai bisogno:

```
class SomeController extends Controller {

    public function getRandomNumber()
    {
        $service = new OurService();
        return $service->getNumber();
    }

    public function getOtherRandomNumber()
    {
        $service = new OurService();
        return $service->getNumber();
    }
}
```

Questo è il motivo per cui il prossimo passo è registrare il tuo servizio nel **Service Container** . Quando registri il servizio nel contenitore del servizio, non è necessario creare un nuovo oggetto ogni volta che ne hai bisogno.

Per registrare un servizio nel contenitore del servizio, è necessario creare un **fornitore di servizi** . Questo fornitore di servizi può:

1. Registra il tuo servizio nel contenitore di servizio con *il metodo di registrazione*)
2. Iniezione di altri servizi nel servizio (dipendenze) con *il metodo di avvio*

Un **fornitore di servizi** è una classe che estende la classe astratta `Illuminate\Support\ServiceProvider` . È necessario implementare il metodo `register()` per **registrare** un servizio nel **Service Container** :

```
# app/Services/OurService/OurServiceServiceProvider.php

<?php
namespace App\Services\OurService;

use Illuminate\Support\ServiceProvider;

class OurServiceServiceProvider extends ServiceProvider
{
    public function register()
    {
        $this->app->singleton('OurService', function($app) {
            return new OurService();
        });
    }
}
```

Tutti i **fornitori di servizi** sono salvati in un array in `config/app.php` . Quindi abbiamo bisogno di registrare il nostro fornitore di servizi in questo array:

```
return [

    ...

    'providers' => [

        ...

        App\Services\OurService\OurServiceServiceProvider::class,

        ...

    ],

    ...

];
```

Ora possiamo accedere al nostro servizio in un controller. Tre possibilità:

1. Iniezione di dipendenza:

```
<?php
namespace App\Http\Controllers;

use App\Services\OurService\OurService;

class SomeController extends Controller
{
    public function __construct(OurService $our_service)
    {
        dd($our_service->getNumber());
    }
}
```

2. Accesso tramite l'helper `app()` :

```
<?php
namespace App\Http\Controllers;

use App\Services\OurService\OurService;

class SomeController extends Controller
{
    public function getRandomNumber()
    {
        return app('OurService')->getNumber();
    }
}
```

Laravel fornisce facciate, classi immaginarie che puoi usare in tutti i tuoi progetti e riflettere un servizio. Per accedere più facilmente al tuo servizio, puoi creare una facciata:

```
<?php
namespace App\Http\Controllers;

use Randomisator;

class SomeController extends Controller
{
    public function getRandomNumber()
    {
        return Randomisator::getNumber();
    }

}
```

Per creare una nuova facciata, è necessario creare una nuova classe che estende `Illuminate\Support\Facades\Facade` . Questa classe deve implementare il metodo `getFacadeAccessor()` e restituire il nome di un servizio registrato da un **fornitore di servizi** :

```
# app/Services/OurService/OurServiceFacade.php

<?php
namespace App\Services\OurService;

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

```
class OurServiceFacade extends Facade
{
    protected static function getFacadeAccessor()
    {
        return 'OurService';
    }
}
```

Devi anche registrare la tua facciata in `config/app.php` :

```
return [

    ...

    'aliases' => [

        ....

        'Randomisator' => App\Services\OurService\OurServiceFacade::class,
    ],

];
```

The Facade è ora accessibile ovunque nel tuo progetto.

Se si desidera accedere al servizio dalle proprie visualizzazioni, è possibile creare una funzione di supporto. Laravel viene fornito con alcuni helper che funzionano fuori dalla scatola, come la funzione `auth()` o la funzione `view()` . Per creare una funzione di supporto, crea un nuovo file:

```
# app/Services/OurService/helpers.php

if (! function_exists('randomisator')) {
    /**
     * Get the available OurService instance.
     *
     * @return \App\ElMatella\FacebookLaravelSdk
     */
    function randomisator()
    {
        return app('OurService');
    }
}
```

Devi anche registrare questo file, ma nel tuo file `composer.json` :

```
{

    ...

    "autoload": {
        "files": [
            "app/Services/OurService/helpers.php"
        ],

        ...
    }
```

```
}
```

Ora puoi usare questo helper in una vista:

```
<h1>Here is a random number: {{ randomisator()->getNumber() }}</h1>
```

Leggi Servizi online: https://riptutorial.com/it/laravel/topic/1907/servizi

Capitolo 61: Servizi

Examples

Associazione di un'interfaccia all'implementazione

In un metodo di `register` Service Provider possiamo associare un'interfaccia a un'implementazione:

```
public function register()
{
    App::bind( UserRepositoryInterface::class, EloquentUserRepository::class );
}
```

D'ora in poi, ogni volta che l'applicazione avrà bisogno di un'istanza di `UserRepositoryInterface` , Laravel eseguirà automaticamente l'iniezione di una nuova istanza di `EloquentUserRepository` :

```
//this will get back an instance of EloquentUserRepository
$repo = App::make( UserRepositoryInterface:class );
```

Associazione di un'istanza

Possiamo utilizzare il contenitore di servizi come registro associando un'istanza di un oggetto e recuperandolo quando ne avremo bisogno:

```
// Create an instance.
$john = new User('John');

// Bind it to the service container.
App::instance('the-user', $john);

// ...somewhere and/or in another class...

// Get back the instance
$john = App::make('the-user');
```

Associazione di un Singleton al contenitore di servizi

Possiamo legare una classe come Singleton:

```
public function register()
{
    App::singleton('my-database', function()
    {
        return new Database();
    });
}
```

In questo modo, la prima volta che un'istanza di `'my-database'` verrà richiesta al contenitore del

servizio, verrà creata una nuova istanza. Tutte le successive richieste di questa classe recupereranno la prima istanza creata:

```
//a new instance of Database is created
$db = App::make('my-database');

//the same instance created before is returned
$anotherDb = App::make('my-database');
```

introduzione

Il **contenitore del servizio** è l'oggetto principale dell'applicazione. Può essere utilizzato come contenitore di iniezione delle dipendenze e un registro per l'applicazione definendo i binding nei provider di servizi

I Service Provider sono classi in cui definiamo il modo in cui le nostre classi di servizio verranno create attraverso l'applicazione, riavviaranno la loro configurazione e collegheranno le interfacce alle implementazioni

I servizi sono classi che racchiudono una o più attività correlate alla logica

Utilizzo del contenitore di servizi come contenitore di iniezione delle dipendenze

Possiamo utilizzare il contenitore di servizi come contenitore di iniezione delle dipendenze legando il processo di creazione degli oggetti con le loro dipendenze in un punto dell'applicazione

Supponiamo che la creazione di un `PdfCreator` bisogno di due oggetti come dipendenze; ogni volta che abbiamo bisogno di costruire un'istanza di `PdfCreator` , dovremmo passare queste dipendenze a che costruttore. Utilizzando il Service Container come DIC, definiamo la creazione di `PdfCreator` nella definizione del binding, prendendo la dipendenza richiesta direttamente dal contenitore del servizio:

```
App:bind('pdf-creator', function($app) {

    // Get the needed dependencies from the service container.
    $pdfRender = $app->make('pdf-render');
    $templateManager = $app->make('template-manager');

    // Create the instance passing the needed dependencies.
    return new PdfCreator( $pdfRender, $templateManager );
});
```

Quindi, in ogni punto della nostra app, per ottenere un nuovo `PdfCreator` , possiamo semplicemente fare:

```
$pdfCreator = App::make('pdf-creator');
```

E il contenitore del Servizio creerà una nuova istanza, insieme alle dipendenze necessarie per noi.

Leggi Servizi online: https://riptutorial.com/it/laravel/topic/1908/servizi

Capitolo 62: Struttura della directory

Examples

Cambia directory delle app predefinite

Ci sono casi d'uso quando potresti voler rinominare la tua directory app in qualcos'altro. In Laravel4 puoi semplicemente modificare una voce di configurazione, ecco un modo per farlo in Laravel5.

In questo esempio, rinomineremo la directory `app` in `src` .

Sostituisci la classe dell'applicazione

L' `app` name name è hardcoded nella classe Application principale, quindi deve essere sovrascritta. Crea un nuovo file `Application.php` . Preferisco mantenere il mio nella directory `src` (quella che sostituiremo con l'app), ma puoi posizionarlo altrove.

Ecco come dovrebbe apparire la classe sostituita. Se vuoi un nome diverso, cambia la stringa `src` in qualcos'altro.

```
namespace App;

class Application extends \Illuminate\Foundation\Application
{
    /**
     * @inheritdoc
     */
    public function path($path = '')
    {
        return $this->basePath . DIRECTORY_SEPARATOR . 'src' . ($path ? DIRECTORY_SEPARATOR .
$path : $path);
    }
}
```

Salva il file. Abbiamo finito con questo.

Chiamando la nuova classe

Apri il `bootstrap/app.php` e individua

```
$app = new Illuminate\Foundation\Application(
    realpath(__DIR__.'/../')
);
```

Lo sostituiremo con questo

```
$app = new App\Application(
```

```
    realpath(__DIR__.'/../')
);
```

Compositore

Apri il tuo file `composer.json` e cambia autoloading in modo che corrisponda alla tua nuova posizione

```
"psr-4": {
    "App\\": "src/"
}
```

E infine, nella riga di comando esegui `composer dump-autoload` e la tua app dovrebbe essere pubblicata dalla directory `src` .

Cambia la directory Controller

se vogliamo cambiare la directory `Controllers` abbiamo bisogno di:

1. Sposta e / o rinomina la directory `Controllers` predefinita dove vogliamo. Ad esempio da `app/Http/Controllers` a `app/Controllers`
2. Aggiorna tutti gli spazi dei nomi dei file all'interno della cartella `Controllers` , facendo aderire al nuovo percorso, rispettando lo specifico PSR-4.
3. Cambia lo spazio dei nomi che viene applicato al file `routes.php` , modificando `app\Providers\RouteServiceProvider.php` e modifica questo:

```
protected $namespace = 'App\Http\Controllers';
```

a questo:

```
protected $namespace = 'App\Controllers';
```

Leggi Struttura della directory online: https://riptutorial.com/it/laravel/topic/3153/struttura-della-directory

Capitolo 63: usa gli alias dei campi in Eloquent

Leggi usa gli alias dei campi in Eloquent online: https://riptutorial.com/it/laravel/topic/7927/usa-gli-alias-dei-campi-in-eloquent

Capitolo 64: Validazione

Parametri

Parametro	Dettagli
necessario	Il campo è obbligatorio
qualche volta	Eseguire controlli di convalida su un campo solo se tale campo è presente nell'array di input
e-mail	L'input è un'e-mail valida
max: Valore	Il valore di input dovrebbe essere inferiore al valore massimo
unico nel suo genere: db_table_name	Il valore di input dovrebbe essere univoco nel nome della tabella del database fornito
accettato	Sì / On / 1 vero, utile per controllare i TOS
active_url	Deve essere un URL valido in base a checkdnsrr
dopo *: data*	Il campo sotto convalida deve fornire un valore dopo la data specificata
alfa	Il campo sotto convalida deve essere interamente caratteri alfabetici.
alpha_dash	Il campo in fase di validazione può contenere caratteri alfanumerici, nonché trattini e caratteri di sottolineatura.
alpha_num	Il campo sotto convalida deve essere composto esclusivamente da caratteri alfanumerici.
schieramento	Deve essere un array PHP
prima *: data*	Il campo deve essere un valore sotto la data specificata
tra: min, max	Il valore di input deve essere compreso tra il valore minimo (minimo) e massimo (massimo)
booleano	Il campo sotto convalida deve poter essere lanciato come booleano. Gli input accettati sono `true` , `false` , `1` , `0` , `"1"` e `"0"` .
confermato	Il campo in fase di validazione deve avere un campo corrispondente di `foo_confirmation` . Ad esempio, se il campo in fase di validazione è `password` , nell'input deve essere presente un campo `password_confirmation` corrispondente.

Parametro	Dettagli
Data	Il campo in fase di validazione deve essere una data valida in base alla funzione PHP strtotime .
numero intero	Il campo sotto convalida deve essere un numero intero
stringa	Il campo sotto convalida deve essere un tipo di stringa .

Examples

Esempio di base

È possibile convalidare i dati della richiesta utilizzando il metodo di `validate` (disponibile nel controller di base, fornito dalla caratteristica `ValidatesRequests`).

Se le regole passano, il tuo codice continuerà ad essere eseguito normalmente; tuttavia, se la convalida fallisce, verrà inviata automaticamente una risposta di errore contenente gli errori di convalida:

- per richieste di moduli HTML tipiche, l'utente verrà reindirizzato alla pagina precedente, con il modulo che mantiene i valori inviati
- per le richieste che prevedono una risposta JSON, verrà generata una risposta HTTP con codice 422

Ad esempio, nel tuo `UserController` , potresti salvare un nuovo utente nel metodo `store` , che avrebbe bisogno di essere convalidato prima di salvare.

```
/**
 * @param  Request  $request
 * @return Response
 */
public function store(Request $request) {
    $this->validate($request, [
        'name' => 'required',
        'email' => 'email|unique:users|max:255'
    ],
    // second array of validation messages can be passed here
    [
        'name.required' => 'Please provide a valid name!',
        'email.required' => 'Please provide a valid email!',
    ]);

    // The validation passed
}
```

Nell'esempio sopra, convalidiamo che il campo del `name` esiste con un valore non vuoto. In secondo luogo, controlliamo che il campo di `email` abbia un formato di posta elettronica valido, sia univoco nella tabella del database "utenti" e abbia una lunghezza massima di 255 caratteri.

Il `|` Il carattere (pipe) combina diverse regole di convalida per un campo.

A volte potresti voler interrompere l'esecuzione delle regole di convalida su un attributo dopo il primo errore di convalida. Per fare ciò, assegna la regola di `bail` all'attributo:

```
$this->validate($request, [
    'name' => 'bail|required',
    'email' => 'email|unique:users|max:255'
]);
```

L'elenco completo delle regole di convalida disponibili è disponibile nella sezione dei parametri riportata di seguito .

Convalida matrice

La convalida dei campi di input del modulo matrice è molto semplice.

Supponiamo di dover convalidare ogni nome, e-mail e nome padre in un determinato array. Potresti fare quanto segue:

```
$validator = \Validator::make($request->all(), [
    'name.*'       => 'required',
    'email.*'      => 'email|unique:users',
    'fatherName.*' => 'required'
]);

if ($validator->fails()) {
    return back()->withInput()->withErrors($validator->errors());
}
```

Laravel visualizza i messaggi predefiniti per la convalida. Tuttavia, se desideri messaggi personalizzati per campi basati su array, puoi aggiungere il seguente codice:

```
[
    'name.*' => [
        'required' => 'Name field is required',
    ],
    'email.*' => [
        'unique'   => 'Unique Email is required',
    ],
    'fatherName.*' => [
        'required' => 'Father Name required',
    ]
]
```

Il tuo codice finale sarà simile a questo:

```
$validator = \Validator::make($request->all(), [
    'name.*'       => 'required',
    'email.*'      => 'email|unique:users',
    'fatherName.*' => 'required',
], [
    'name.*'       => 'Name Required',
    'email.*'      => 'Unique Email is required',
    'fatherName.*' => 'Father Name required',
]);
```

```
if ($validator->fails()) {
    return back()->withInput()->withErrors($validator->errors());
}
```

Altri approcci di convalida

1) Convalida della richiesta di modulo

È possibile creare una "richiesta di modulo" che può contenere la logica di autorizzazione, le regole di convalida e i messaggi di errore per una particolare richiesta nell'applicazione.

La `make:request` comando CLI Artisan genera la classe e la inserisce nella directory `app/Http/Requests` :

```
php artisan make:request StoreBlogPostRequest
```

Il metodo `authorize` può essere sovrascritto con la logica di autorizzazione per questa richiesta:

```
public function authorize()
{
    return $this->user()->can('post');
}
```

Il metodo delle `rules` può essere sovrascritto con le regole specifiche per questa richiesta:

```
public function rules()
{
    return [
        'title' => 'required|unique:posts|max:255',
        'body' => 'required',
    ];
}
```

Il metodo dei `messages` può essere sovrascritto con i messaggi specifici per questa richiesta:

```
public function messages()
{
    return [
        'title.required' => 'A title is required',
        'title.unique' => 'There is another post with the same title',
        'title.max' => 'The title may not exceed :max characters',
        'body.required' => 'A message is required',
    ];
}
```

Per convalidare la richiesta, basta digitare-suggerire la classe di richiesta specifica sul metodo del controller corrispondente. Se la convalida fallisce, verrà inviata una risposta di errore.

```
public function store(StoreBlogPostRequest $request)
{
    // validation passed
```

```
}
```

---

2) Creazione manuale dei validatori

Per maggiore flessibilità, è possibile creare manualmente un validatore e gestire direttamente la convalida non riuscita:

```
<?php
namespace App\Http\Controllers;

use Validator;
use Illuminate\Http\Request;
use App\Http\Controllers\Controller;

class PostController extends Controller
{
    public function store(Request $request)
    {
        $validator = Validator::make($request->all(), [
            'title' => 'required|unique:posts|max:255',
            'body' => 'required',
        ]);

        if ($validator->fails()) {
            return redirect('post/create')
                    ->withErrors($validator)
                    ->withInput();
        }

        // Store the blog post...
    }
}
```

2) Creazione fluente di regole

Occasionalmente potrebbe essere necessario creare regole univoche al volo, lavorare con il metodo `boot()` all'interno di un Service Provider potrebbe essere esagerato, a partire da Laravel 5.4 è possibile creare fluentemente nuove regole usando la classe `Rule` .

Ad esempio, lavoreremo con `UserRequest` per quando si desidera inserire o aggiornare un utente. Per ora vogliamo che sia richiesto un nome e che l'indirizzo email sia univoco. Il problema con l'utilizzo della regola `unique` è che se stai modificando un utente, potrebbero mantenere la stessa email, quindi devi escludere l'utente corrente dalla regola. L'esempio seguente mostra come puoi farlo facilmente utilizzando la nuova classe `Rule` .

```
<?php
namespace App\Http\Requests;
use Illuminate\Foundation\Http\FormRequest;
use Illuminate\Http\Request;
use Illuminate\Validation\Rule;

class UserRequest extends FormRequest
{
    /**
```

```
     * Determine if the user is authorized to make this request.
     *
     * @return bool
     */
    public function authorize()
    {
        return true;
    }

    /**
     * Get the validation rules that apply to the request.
     *
     * @return array
     */
    public function rules(Request $request)
    {
        $id = $request->route()->getParameter('user');

        return [
            'name'           =>  'required',

            // Notice the value is an array and not a string like usual
            'email'         =>  [
                'required',
                Rule::unique('users')->ignore($id)
            ]
        ];
    }
}
```

Classe di richiesta modulo singolo per POST, PUT, PATCH

Seguendo l'esempio di convalida della richiesta di form , è possibile utilizzare la stessa classe di richiesta per `POST` , `PUT` , `PATCH` modo da non dover creare un'altra classe utilizzando le stesse convalide / simili. Questo è utile se hai attributi nella tua tabella che sono unici.

```
/**
 * Get the validation rules that apply to the request.
 *
 * @return array
 */
public function rules() {
    switch($this->method()) {
        case 'GET':
        case 'DELETE':
            return [];
        case 'POST':
            return [
                'name'     => 'required|max:75|unique',
                'category' => 'required',
                'price'    => 'required|between:0,1000',
            ];
        case 'PUT':
        case 'PATCH':
            return [
                'name'     => 'required|max:75|unique:product,name,' . $this->product,
                'category' => 'required',
                'price'    => 'required|between:0,1000',
```

```
            ];
        default:break;
    }
}
```

Partendo dall'alto, la nostra istruzione switch analizzerà il tipo di metodo della richiesta (`GET` , `DELETE` , `POST` , `PUT` , `PATCH`).

A seconda del metodo restituirà la serie di regole definite. Se si dispone di un campo univoco, ad esempio il campo del `name` nell'esempio, è necessario specificare un particolare ID per la convalida da ignorare.

```
'field_name' => 'unique:table_name,column_name,' . $idToIgnore`
```

Se si ha una chiave primaria etichettata diversa da `id` , si specifica la colonna chiave primaria come quarto parametro.

```
'field_name' => 'unique:table_name,column_name,' . $idToIgnore . ',primary_key_column'
```

In questo esempio, stiamo utilizzando `PUT` e passando alla route (`admin/products/{product}`) il valore dell'id del prodotto. Quindi `$this->product` sarà uguale `id` da ignorare.

Ora le tue regole di convalida `PUT|PATCH` e `POST` non devono essere le stesse. Definisci la tua logica che si adatta alle tue esigenze. Questa tecnica ti consente di riutilizzare i messaggi personalizzati che potresti aver definito all'interno della classe di richiesta modulo personalizzata.

Messaggio di errore

Personalizzazione dei messaggi di errore

I file `/resources/lang/[lang]/validation.php` contengono i messaggi di errore che devono essere utilizzati dal validatore. Puoi modificarli secondo necessità.

La maggior parte di essi ha segnaposti che verranno automaticamente sostituiti durante la generazione del messaggio di errore.

Ad esempio, in `'required' => 'The :attribute field is required.'` , il segnaposto `:attribute` sarà sostituito dal nome del campo (in alternativa, è possibile personalizzare anche il valore di visualizzazione di ciascun campo nell'array degli `attributes` nello stesso file).

Esempio

configurazione del messaggio:

```
'required' => 'Please inform your :attribute.',
//...
'attributes => [
    'email' => 'E-Mail address'
]
```

regole:

```
`email' => `required`
```

messaggio di errore risultante:

> "Per favore informa il tuo indirizzo e-mail."

Personalizzazione dei messaggi di errore all'interno di una classe di richiesta

La classe Request ha accesso a un metodo `messages()` che dovrebbe restituire un array, questo può essere usato per sovrascrivere i messaggi senza dover accedere ai file lang. Ad esempio, se abbiamo una convalida personalizzata di `file_exists` puoi inviare messaggi come di seguito.

```
class SampleRequest extends Request {

    /**
     * Get the validation rules that apply to the request.
     *
     * @return array
     */
    public function rules()
    {
        return [
            'image' =>  'required|file_exists'
        ];
    }

    /**
     * Determine if the user is authorized to make this request.
     *
     * @return bool
     */
    public function authorize()
    {
        return true;
    }

    public function messages()
    {
        return [
            'image.file_exists' =>  'That file no longer exists or is invalid'
        ];
    }

}
```

Visualizzazione dei messaggi di errore

Gli errori di convalida vengono visualizzati sulla sessione e sono disponibili anche nella variabile `$errors`

, che viene condivisa automaticamente su tutte le visualizzazioni.

Esempio di visualizzazione degli errori in una vista Blade:

```
@if (count($errors) > 0)
    <div class="alert alert-danger">
        <ul>
            @foreach ($errors->all() as $error)
                <li>{{ $error }}</li>
            @endforeach
        </ul>
    </div>
@endif
```

Regole di convalida personalizzate

Se si desidera creare una regola di convalida personalizzata, è possibile farlo ad esempio nel metodo di `boot` di un fornitore di servizi, tramite la facciata Validator.

```
<?php
namespace App\Providers;

use Illuminate\Support\ServiceProvider;
use Validator;

class AppServiceProvider extends ServiceProvider
{
    public function boot()
    {
        Validator::extend('starts_with', function($attribute, $value, $parameters, $validator)
{
            return \Illuminate\Support\Str::startsWith($value, $parameters[0]);
        });

        Validator::replacer('starts_with', function($message, $attribute, $rule, $parameters)
{
            return str_replace(':needle', $parameters[0], $message);
        });
    }
}
```

Il metodo `extend` prende una stringa che sarà il nome della regola e una funzione che a sua volta sarà passata il nome dell'attributo, il valore che viene convalidato, una matrice dei parametri della regola e l'istanza del validatore, e dovrebbe restituire se la convalida passa. In questo esempio, stiamo controllando se la stringa del valore inizia con una sottostringa data.

Il messaggio di errore per questa regola personalizzata può essere impostato normalmente nel file `/resources/lang/[lang]/validation.php` e può contenere segnaposti, ad esempio, per i valori dei parametri:

```
'starts_with' => 'The :attribute must start with :needle.'
```

Il metodo `replacer` prende una stringa che è il nome della regola e una funzione che a sua volta verrà passata al messaggio originale (prima della sostituzione), il nome dell'attributo, il nome della

regola e una matrice dei parametri della regola, e dovrebbe restituire il messaggio dopo aver sostituito i segnaposto secondo necessità.

Usa questa regola come qualsiasi altra:

```
$this->validate($request, [
    'phone_number' => 'required|starts_with:+'
]);
```

Leggi Validazione online: https://riptutorial.com/it/laravel/topic/1310/validazione

Capitolo 65: valletto

introduzione

Valet è un ambiente di sviluppo su misura per macOS. Elimina la necessità di macchine virtuali, Homestead o Vagrant. Non è più necessario aggiornare costantemente il `/etc/hosts` . Puoi persino condividere i tuoi siti pubblicamente usando tunnel locali.

Laravel Valet rende tutti i siti disponibili su un dominio `*.dev` legando i nomi delle cartelle ai nomi di dominio.

Sintassi

- comando valet [opzioni] [argomenti]

Parametri

Parametro	Set di valori
comando	dominio , fetch-share-url, dimentica, guida, installa, collega , collega , elenca, registra, on-ultima versione, apre, parcheggia , percorsi, riavvia, sicuro, avvia, ferma, disinstalla, scollega, non sicuro, quale
opzioni	-h, --help, -q, --quiet, -V, --version, --ansi, --no-ansi, -n, --no-interaction, -v, -vv, -vvv, - -verbose
argomenti	*(opzionale)*

Osservazioni

Poiché Valet per Linux e Windows non sono ufficiali, non ci sarà supporto al di fuori dei rispettivi repository Github.

Examples

Collegamento

Questo comando è utile se vuoi servire un singolo sito in una directory e non l'intera directory.

```
cd ~/Projects/my-blog/
valet link awesome-blog
```

Valet creerà un link simbolico in `~/.valet/Sites` che punta alla directory di lavoro corrente.

Dopo aver eseguito il comando link, puoi accedere al sito nel tuo browser su `http://awesome-blog.dev` .

Per vedere un elenco di tutte le tue directory collegate, esegui il comando valet links . È possibile utilizzare `valet unlink awesome-blog` per distruggere il collegamento simbolico.

Parcheggio

```
cd ~/Projects
valet park
```

Questo comando registrerà la directory di lavoro corrente come un percorso che Valet dovrebbe cercare per i siti. Ora, qualsiasi progetto Laravel che crei all'interno della tua directory "parcheggiata" verrà automaticamente servito usando la convenzione `http://folder-name.dev` .

Collegamenti

Questo comando mostrerà tutti i collegamenti di Valet registrati che hai creato e i loro percorsi di file corrispondenti sul tuo computer.

Comando:

```
valet links
```

Uscita di esempio:

```
...
site1 -> /path/to/site/one
site2 -> /path/to/site/two
...
```

Nota 1: è possibile eseguire questo comando da qualsiasi posizione, non solo all'interno di una cartella collegata.

Nota 2: i siti saranno elencati senza la fine **.dev** ma continuerai a utilizzare **site1.dev** per accedere alla tua applicazione dal browser.

Installazione

***IMPORTANTE!!* Valet è uno strumento progettato solo per macOS.**

Prerequisiti

- Valet utilizza la porta HTTP della propria macchina locale (porta 80), pertanto, non sarà possibile utilizzarla se *Apache* o *Nginx* sono installati e in esecuzione sulla stessa macchina.
- MacOS' direttore non ufficiale pacchetto Homebrew è necessario usare correttamente Valet.
- Assicurati che Homebrew sia aggiornato all'ultima versione eseguendo l' `brew update` nel terminale.

Installazione

- Installa PHP 7.1 usando Homebrew tramite `brew install homebrew/php/php71` .
- Installare Valet con Composer via `composer global require laravel/valet` .
- Aggiungi `~/.composer/vendor/bin` alla "PATH" del tuo sistema se non è già presente.
- Esegui il comando di `valet install` .

Post Install Durante il processo di installazione, Valet ha installato *DnsMasq* . Ha inoltre registrato il daemon di Valet per l'avvio automatico all'avvio del sistema, quindi non è necessario eseguire l' `valet start` o l' `valet install` ogni volta che si riavvia il computer.

Valet domain

Questo comando consente di modificare o visualizzare il dominio di *primo livello (dominio di primo livello)* utilizzato per associare i domini al computer locale.

Ottieni il TLD attuale

```
$ valet domain
> dev
```

Imposta il TLD

```
$ valet domain local
> Your Valet domain has been updated to [local].
```

Installazione (Linux)

IMPORTANTE!! Valet è uno strumento progettato per macOS, la versione di seguito viene portata su Linux OS.

Prerequisiti

- **Non** installare valet come `root` o usando il comando `sudo` .
- Valet utilizza la porta HTTP della propria macchina locale (porta 80), pertanto, non sarà possibile utilizzarla se Apache o Nginx sono installati e in esecuzione sulla stessa macchina.
- È richiesta una versione aggiornata del `composer` per installare ed eseguire Valet.

Installazione

- Eseguire `composer global require cpriego/valet-linux` per installare Valet a livello globale.
- Eseguire il comando di `valet install` per completare l'installazione.

Post installazione

Durante il processo di installazione, Valet ha installato DnsMasq. Ha inoltre registrato il daemon di Valet per l'avvio automatico all'avvio del sistema, quindi non è necessario eseguire l' `valet start` o l' `valet install` ogni volta che si riavvia il computer.

La documentazione ufficiale può essere trovata qui .

Leggi valletto online: https://riptutorial.com/it/laravel/topic/1906/valletto

Titoli di coda

S. No	Capitoli	Contributors
1	Iniziare con Laravel	alepeino, Alphonsus, boroboris, Colin Herzog, Community, Ed Rands, Evgeniy Maynagashev, Gaurav, Imam Assidiqqi, James, Ketan Akbari, Kovah, Lance Pioch, Marek Skiba, Martin Bean, Misa Lazovic, nyedidikeke, Oliver Adria, Prakash, rap-2-h, Ru Chern Chong, SeinopSys, Tatranskymedved, Tim
2	analisi	Alessandro Bassi, Brayniverse, caoglish, Julian Minde, Kyslik, rap-2-h, Sven
3	Artigiano	Alessandro Bassi, Gaurav, Harshal Limaye, Himanshu Raval, Imam Assidiqqi, Kaspars, Laurel, Rubens Mariuzzo, Safoor Safdar, Sagar Naliyapara, SeinopSys
4	Autenticazione	Aykut CAN, Imam Assidiqqi
5	Autorizzazione	Daniel Verem
6	Autorizzazioni per l'archiviazione	A. Raza
7	Banca dati	A. Raza, adam, caoglish, Ian, Iftikhar uddin, Imam Assidiqqi, liamja, Panagiotis Koursaris, RamenChef, Rubens Mariuzzo, Sanzeeb Aryal, Vucko
8	Cassiere	littleswany, RamenChef
9	Classe CustomException in Laravel	ashish bansal
10	code	Alessandro Bassi, Kyslik
11	collezioni	A. Raza, Alessandro Bassi, Alex Harris, bhill77, caoglish, Dummy Code, Gras Double, Ian, Imam Assidiqqi, Josh Rumbut, Karim Geiger, matiaslauriti, Nicklas Kevin Frank, Ozzy, rap-2-h, simonhamp, Vucko
12	Connessioni DB multiple in Laravel	4444, A. Raza, Rana Ghosh
13	Controller	Ru Chern Chong
14	costanti	Mubashar Iqbal, Oscar David, Zakaria Acharki

15	Denominazione dei file durante il caricamento con Laravel su Windows	Donkarnash, RamenChef
16	Distribuire l'applicazione Laravel 5 su Shared Hosting su Linux Server	Donkarnash, Gayan, Imam Assidiqqi, Kyslik, PassionInfinite, Pete Houston, rap-2-h, Ru Chern Chong, Stojan Kukrika, ultrasamad
17	Eloquente	aimme, alepeino, Alessandro Bassi, Alex Harris, Alfa, Alphonsus, andretzermias, andrewtweber, Andrey Lutskevich, aynber, Buckwheat, Casper Spruit, Dancia, Dipesh Poudel, Ian, Imam Assidiqqi, James, James, jedrzej.kurylo, John Slegers, Josh Rumbut, Kaspars, Ketan Akbari, KuKeC, littleswany, Lykegenes, Maantje, Mahmood, Marco Aurélio Deleu, marcus.ramsden, Marek Skiba, Martin Bean, matiaslauriti, MM2, Nicklas Kevin Frank, Niklas Modess, Nyan Lynn Htut, patricus, Pete Houston, Phroggyy, Prisoner Raju, RamenChef, rap-2-h, Rubens Mariuzzo, Sagar Naliyapara, Samsquanch, Sergio Guillen Mantilla, Tim, tkausl, whoan, Yasin Patel
18	Eloquente: Accessors & Mutators	Diego Souza, Kyslik
19	Eloquente: modello	Aeolingamenfel, alepeino, Alex Harris, Imam Assidiqqi, John Slegers, Kaspars, littleswany, Marco Aurélio Deleu, marcus.ramsden, Marek Skiba, matiaslauriti, Nicklas Kevin Frank, Samsquanch, Tim
20	Eloquente: relazione	Advaith, aimme, Alex Harris, Alphonsus, bhill77, Imam Assidiqqi , Ketan Akbari, Phroggyy, rap-2-h, Ru Chern Chong, Zulfiqar Tariq
21	Errore di mancata corrispondenza del token in AJAX	Pankaj Makwana
22	Eventi e ascoltatori	Bharat Geleda, matiaslauriti, Nauman Zafar
23	Filesystem / Cloud Storage	Imam Assidiqqi, Nitish Kumar, Paulo Laxamana
24	Funzione Helper personalizzata	Ian, Luceos, rap-2-h, Raunak Gupta
25	Gestione degli errori	Isma, Kyslik, RamenChef, Rubens Mariuzzo

26	Guida d'installazione	Advaith, Amarnasan, aynber, Community, davejal, Dov Benyomin Sohacheski, Imam Assidiqqi, PaladiN, rap-2-h, Ru Chern Chong
27	Helpers	aimme
28	HTML e Form Builder	alepeino, Casper Spruit, Himanshu Raval, Prakash
29	Iniziare con laravel-5.3	A. Raza, Advaith, Community, davejal, Deathstorm, Manish, Matthew Beckman, Robin Dirksen, Shital Jachak
30	Installazione	A. Raza, alepeino, Alphonsus, Black, boroboris, Gaurav, Imam Assidiqqi, James, Ketan Akbari, Lance Pioch, Marek Skiba, Martin Bean, nyedidikeke, PaladiN, Prakash, rap-2-h, Ru Chern Chong, Sagar Naliyapara, SeinopSys, Tim
31	Integrazione Sparkpost con Laravel 5.4	Alvin Chettiar
32	Introduzione a laravel-5.2	A. Raza, ashish bansal, Community, Edward Palen, Ivanka Todorova, Shubhamoy
33	Introduzione a laravel-5.3	Ian
34	Laravel Docker	Dov Benyomin Sohacheski
35	Link utili	Jakub Kratina
36	Macro in relazione eloquente	Alex Casajuana, Vikash
37	middleware	Alex Harris, Kaspars, Kyslik, Moppo, Pistachio
38	Migrazioni del database	Chris, Chris White, Hovsep, hschin, Iftikhar uddin, Imam Assidiqqi, Kaspars, liamja, littleswany, mnoronha, Nauman Zafar, Panagiotis Koursaris, Paulo Freitas, Vucko
39	Modelli di lama	A. Raza, agleis, Akshay Khale, alepeino, Alessandro Bassi, Benubird, cbaconnier, Christophvh, Imam Assidiqqi, matiaslauriti, Nauman Zafar, rap-2-h, Safoor Safdar, Tosho Trajanov, yogesh
40	Modifica il comportamento di routing predefinito in Laravel 5.2.31 +	Frank Provost

41	Nozioni di base di Cron	A. Raza
42	Osservatore	matiaslauriti, Szenis
43	Pacchetti Laravel	Casper Spruit, Imam Assidiqqi, Ketan Akbari, rap-2-h, Ru Chern Chong, Tosho Trajanov
44	paginatura	Himanshu Raval, Iftikhar uddin
45	persona mondana	Jonathon, Marco Aurélio Deleu
46	Pianificazione delle attività	Jonathon
47	Politiche	Tosho Trajanov
48	posta	Yohanan Baruchel
49	Problemi comuni e soluzioni rapide	Nauman Zafar
50	quadro di lumen	maksbd19
51	Richiesta / e di modulo	Bookeater, Ian, John Roca, Kyslik, RamenChef
52	Richiesta di dominio incrociato	Imam Assidiqqi, Suraj
53	richieste	Ian, Jerodev, RamenChef, Rubens Mariuzzo
54	Rimuovi pubblico dall'URL in laravel	A. Raza, Rana Ghosh, ultrasamad
55	Route Model Binding	A. Raza, GiuServ, Vikash
56	Routing	A. Raza, alepeino, Alessandro Bassi, Alex Juchem, beznez, Dwight, Ilker Mutlu, Imam Assidiqqi, jedrzej.kurylo, Kyslik, Milan Maharjan, Rubens Mariuzzo, SeinopSys, Vucko
57	Semina	A. Raza, Alphonsus, Ian, Imam Assidiqqi, Kyslik, SupFrost, whoan
58	Semina del database	Achraf Khouadja, Andrew Nolan, Dan Johnson, Isma, Kyslik, Marco Aurélio Deleu
59	Servizi	A. Raza, El_Matella
60	Struttura della directory	Kaspars, Moppo, RamenChef

61	usa gli alias dei campi in Eloquent	MM2
62	Validazione	A. Raza, alepeino, Alessandro Bassi, Alex Harris, Andrew Nolan , happyhardik, Himanshu Raval, Ian, Iftikhar uddin, John Slegers, Marco Aurélio Deleu, matiaslauriti, rap-2-h, Rubens Mariuzzo, Safoor Safdar, Sagar Naliyapara, Stephen Leppik, sun, Vucko
63	valletto	David Lartey, Dov Benyomin Sohacheski, Imam Assidiqqi, Misa Lazovic, Ru Chern Chong, Shog9