Laravel 5

J. ROMAGNY

Table des matières

1. EDI

PHPStorm dispose de plugins pour **Laravel**. Chercher dans « **settings** » … « **plugins** »

2. Installation de Laravel

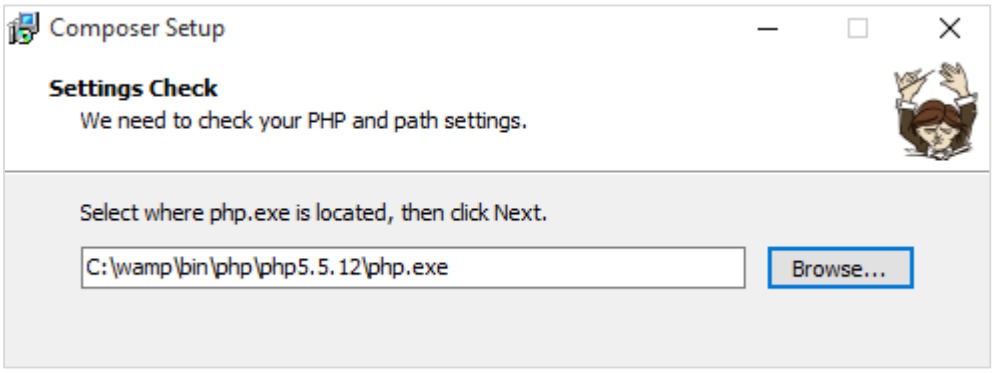
Documentation

Exemple pour **Windows**

a. Installer **Composer** (si ce n'est pas fait)

Documentation

Sur **Windows** télécharger « Composer-Setup.exe », durant l'installation renseigner le chemin vers **PHP** (exemple avec WAMP)



Depuis une invite de commande, saisir « composer » pour voir si l'installation s'est correctement passée.

b. Installer **Laravel**

Depuis une invite de commande, installer Laravel grâce à Composer

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

Ajouter à la variable d'environnement **Path** de l'utilisateur le chemin vers le répertoire « bin » de « composer » du dossier « AppData »

```
C:\Users\[user]\AppData\Roaming\Composer\vendor\bin
```

Depuis une invite de invite entrer « laravel » pour voir si tout est bon.

3. Créer un nouveau projet Laravel

- Soit … Depuis une invite de commande, naviguer jusqu'au dossier ou le projet devra être créé (le dossier « www » de WAMP par exemple) puis on crée un projet « laravelcms » par exemple

```
laravel new laravelcms
```

- Soit avec **Composer**, permet de **choisir la version de Laravel** utilisée

```
composer create-project laravel/laravel=5.1 laravelcms --prefer-dist
```

a. Packages pour Laravel

Packalyst

b. Configurer le site

URL

Pour permettre à Artisan de générer son code correctement

```
#config/app.php

'url' => 'http://localhost/laravelcms',
```

Base de données

Choix de la base de données utilisée

```
#config/app.php

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

Renseigner les informations de la base plus bas dans le fichier

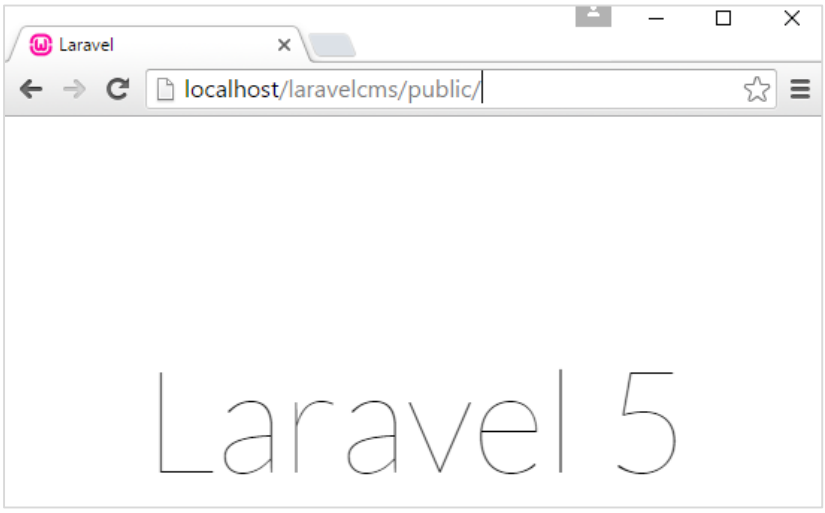
```
'mysql' => [
    'driver'    => 'mysql',
    'host'      => env('DB_HOST', 'localhost'),
    'database'  => env('DB_DATABASE', 'laraveldb'),
    'username'  => env('DB_USERNAME', 'root'),
    'password'  => env('DB_PASSWORD', ''),
    'charset'   => 'utf8',
    'collation' => 'utf8_unicode_ci',
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

Il faut également changer les informations de base de données dans le fichier « **.env** » à la racine du projet.

```
DB_HOST=localhost
DB_DATABASE=laraveldb
DB_USERNAME=root
DB_PASSWORD=
```

c. Afficher son site

a. On peut aller « **http://localhost/laravelcms/public/** » pour tester son site.



b. Il est possible de créer un « **Virtual Host** » (pour avoir « local.dev » au lieu de « http://localhost/.../public/ »)

c. On peut aussi lancer un **serveur** avec **Artisan**.

Depuis une invite de commande, naviguer jusqu'au dossier du projet puis

```
php artisan serve
```

On peut alors se rendre « **http://localhost:8000/** » pour afficher son site.

CTRL + C pour **arrêter le serveur**

d. Les commandes avec Artisan

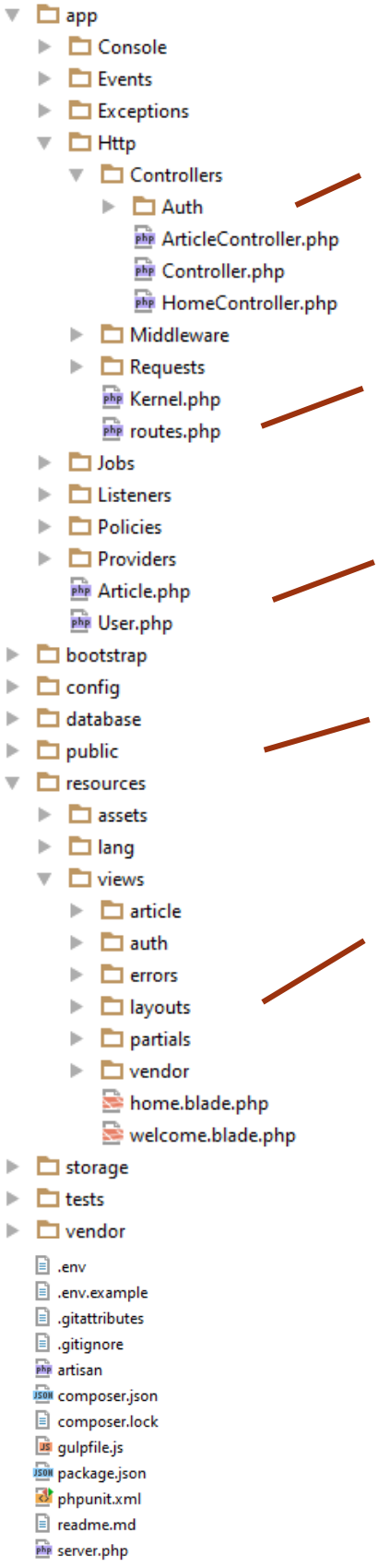
Obtenir la liste des commandes disponibles

```
php artisan list
```

Les commandes « make » permettent de générer du code.

4. Organisation du projet

Contrairement à Symfony, le projet n'est pas organisé par « bundles »



Contrôleurs

« app/Http/Controllers/ »

- « Auth » contrôleurs pour authentification

Routes

« app/Http/routes.php »

Modèles

« app/»

Dossier « public » contient les feuilles de Styles, scripts, images, etc.

Vues

« app/resources/views »

- Master Pages dans le dossier « layouts »
- vues partielles dans le dossier « partials »

5. Authentification à prendre en considération

Si le site nécessite la connexion des utilisateurs. Il peut être bon d'installer dès le début la Master Page, les vues pour l'authentification, les routes, etc.

```
php artisan make:auth
```

6. Routes

Documentation, responses

Note : en cas de problème avec les routes, vérifier que le module « **rewrite_module** » d'**Apache** est **activé**. Si ce n'est pas le cas, l'activer et relancer le service.

Définies dans le fichier « **routes.php** » du dossier « app/Http »

Avec Laravel 5 Celles-ci seront à placer entre …

```
Route::group(['middleware' => ['web']], function () {

});
```

On peut **afficher une vue** directement

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

Avec **passage de paramètre**

```
Route::get('/', function () {
    return view('welcome',array('message' => 'Bienvenue sur le site!'));
});
```

Route **nommée** avec **vue**

```
Route::get('/',array('as'=> 'homepage', function () {
    return view('welcome');
}));
```

Mais le plus souvent on **lie à une action d'un contrôleur**, et on nommera la route de manière à pouvoir l'utiliser depuis les vues (avec la méthode « route ») ou pour les redirections depuis les contrôleurs

Route « /articles » nommée « articles_index » pointant l'action « index » du contrôleur « ArticleController »

```
Route::get('articles', [
    'as'=> 'articles_index',
    'uses' => 'ArticleController@index'
]);
```

Sans nom avec contrôleur

```
Route::get('articles','ArticleController@index');
```

Avec paramètre

```
Route::get('articles/view/{id}', [
    'as'=> 'articles_view',
    'uses' => 'ArticleController@view'
]);
```

On peut définir plusieurs verbes http pour une route avec « **match** ». A noter ici également que c'est une route protégée, qui demande à l'utilisateur d'être connecté (avec le middleware « auth »)

```
Route::match(['get','post'],'articles/create', [
    'as'=> 'articles_create',
    'uses' => 'ArticleController@create',
    'middleware' => 'auth'
]);
```

… mais aussi « **any** » pour tous les verbes

```
Route::any('test',function(){
    // etc.
});
```

Route pour « **delete** »

```
Route::get('/articles/{id}', [
    'as'=> 'articles_delete',
    'uses' => 'ArticleController@delete',
    'middleware' => 'auth'
]);
```

Il est possible de définir automatiquement toutes les routes pour un contrôleur. On utilisera cette approche pour les routes ne nécessitant pas d'êtres « propres » (partie admin du site par exemple). En premier paramètre la route, en second le contrôleur.

```
Route::controller('welcome','WelcomeController');
```

Groupe

Ici toutes les routes commenceront par « admin »

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

    Route::get('test',function(){
        return 'hello';
    });

});
```

Redirection

```
Route::get('test',function(){
    return redirect('/');
});
```

Afficher un message. Soyons honnête ça sert surtout en démonstration mais …

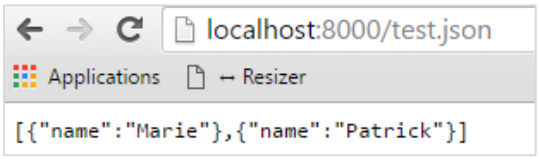
```
Route::get('test',function(){
    return "Ma route !";
});
```

Contrainte

```
Route::get('article/{id}', 'ArticleController@view')->where('id', '[0-9]+');
```

JSON

```
Route::get('test.json', function(){
    $people=[
        ['name'=>"Marie"],
        ['name'=>'Patrick']
    ];
    return response()->json($people);
});
```



Tester ses routes

On peut afficher toutes les routes définies dans la console avec

```
php artisan route:list
```

On peut utiliser **Tinker** pour afficher l'url d'une route

```
php artisan tinker
>>>route('articles')
>>>exit
```

7. Créer un contrôleur

Documentation

Exemple création d'un contrôleur nommé « ArticleController ». Il sera jouté dans « **app/Htpp/Controllers** »

```
php artisan make:controller ArticleController
```

8. Migration et modèle

a. Créer un fichier de migration

Documentation

a. Modifier les **informations de base de données** dans le fichier « **database.php** » (du dossier « config) et « **.env** » (à la racine) si ce n'est fait.
b. Créer une migration permettant de définir

```
php artisan make:migration create_article_table --create=article
```

- « create_article_table » c'est le nom du fichier généré
- « --create=article » c'est le nom de la table

… Le fichier est généré dans le dossier « **database/migrations** »

c. On renseigne les colonnes de la table.

```php
<?php
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;

class CreateArticleTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('article', function (Blueprint $table) {
            $table->increments('id');

            $table->string('title');
            $table->longText('content');
            $table->integer('user_id')->unsigned();
            $table->foreign('user_id')->references('id')->on('users');
            $table->timestamps();
        });
    }
    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::drop('article');
    }
}
```

- « **up** » sert à mettre à jour la base, « **down** » sert à annuler les oprations sur la base.
- Les colonnes « **created_at** » et « **updated_at** » sont automatiquement **ajoutées** avec « **$table->timestamps** »
- On définit une colonne « **user_id** » puis on lui ajoute une **contrainte de clé étrangère** sur l'id de la table « users »

d. Mise à jour de la base de données (ici la table « article » sera créée ainsi que les tables liées à l'authentification étant donné que c'est la première migration effectuée)

```
php artisan migrate
```

Si on veut annuler

```
php artisan migrate :rollback
```

Si on modifie encore le fichier de migration après la création des tables. Attention toutefois les tables seront vidées

```
php artisan migrate:refresh
```

b. Créer le modèle Eloquent

Documentation

```
php artisan make:model Article
```

Le modèle est généré à la racine du dossier « app »

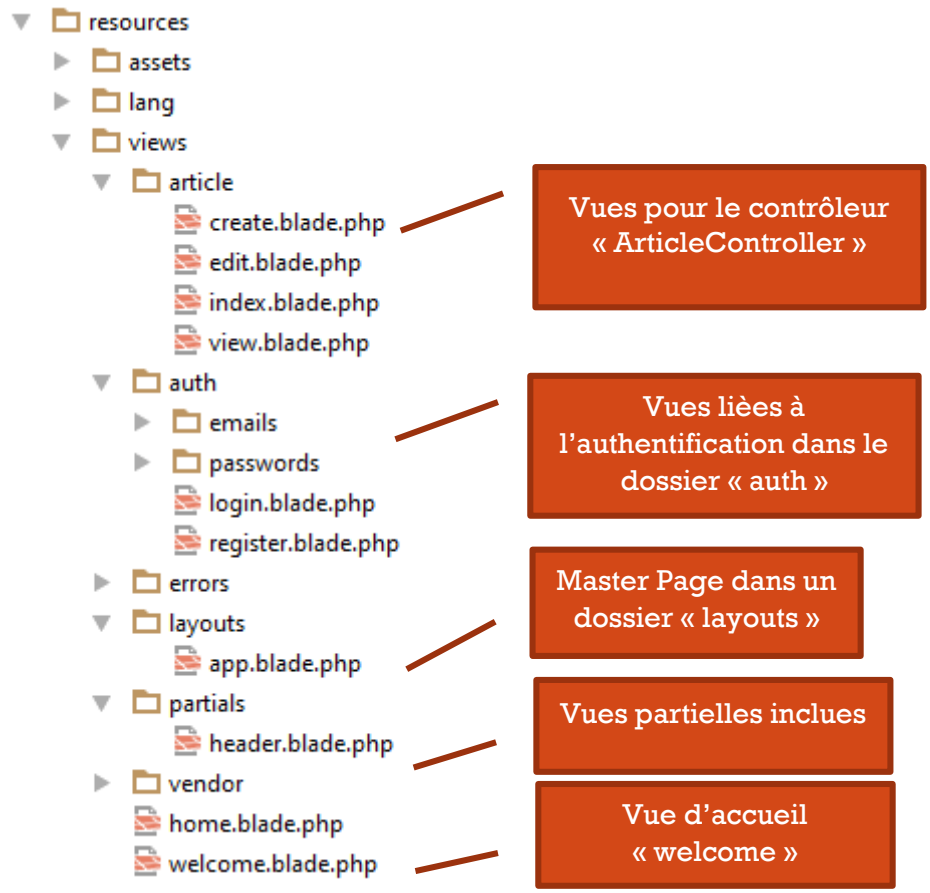
- On renseigne le nom de la table correspondante en base avec « **$table** »
- Et les colonnes qui seront récupérables avec « **$fillable** »
- On définit ici en plus une fonction permettant de charger automatiquement l'utilisateur correspondant à la clé étrangère

```php
<?php
namespace App;

use Illuminate\Database\Eloquent\Model;

class Article extends Model
{
    protected $table='article';
    protected $fillable = ['title','content'];


    public function user()
    {
        return $this->belongsTo('App\User');
    }
}
```

Note on peut cacher des colonnes avec « $hidden »

9. Vues

Le dossier « **resources** » contenant les **vues**



Master Page

On peut créer sa propre Master Page ou utiliser celle générée avec l'authentification.

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Laravel demo - @yield('title')</title>
    <link type="text/css" rel="stylesheet" href="{{ URL::asset('bootstrap/font-
awesome.css') }}">
    <link type="text/css" rel="stylesheet" href="{{ URL::asset('bootstrap/bootstrap.css')
}}">
    <link type="text/css" rel="stylesheet" href="{{ URL::asset('css/style.css') }}">
</head>
<body>
@include('partials.header')

<section class="container">
    @yield('content')
</section>

<script src="https://ajax.googleapis.com/ajax/libs/jquery/2.2.0/jquery.min.js"></script>
<script src="{{ URL::asset('js/bootstrap.min.js') }}"></script>
</body>
</html>
```

Assets du dossier « public »

Vue partielle

Section

Une **vue** simple (welcome.blade.php)

```
@extends('layouts.app')
```

```
@section('content')

    <h2>Accueil</h2>
    <hr>
    <div class="alert alert-warning">
        <p>Bienvenue sur le site!</p>
    </div>

@endsection
```

Le moteur de vues utilisé avec **Laravel** est **BLADE** (Il est possible également d'utiliser PHP).

Documentation

Variables avec $ devant et entre accolades {{ }}

```
{{ $article->title }}
```

@if, **@foreach**, etc.

```
@if(isset($articles) and count($articles)> 0)

@foreach($articles as $article)

<article>
  <h2><a href="{{ route('articles_view',['id' => $article->id ]) }}">{{
$article->title  }}</a></h2>
  <div class="entry-content">
        {{ $article->content  }}
  </div>
</article>

@endforeach
@else
<div class="alert alert-warning text-center">
    Voulez-vous ajouter <a href="{{ route('articles_create') }}">le
premier article</a>?
</div>
@endif
```

Faire des liens

Lien simple

```
<a href="/login">
```

Son équivalent avec la fonction « **url** »

```
<a href="{{ url('/login') }}">
```

Fonction « **route** » pointe sur une route nommée

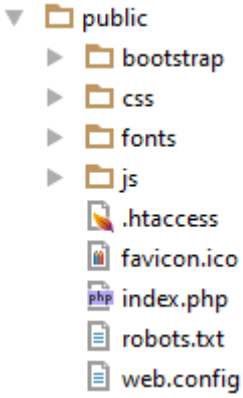
```
<a href="{{ route('articles_create') }}">Ajouter un nouvel article</a>
```

Avec **paramètre passé**

```
<a href="{{ route('articles_delete', ['id'=> $article->id])}}"
class="btn btn-primary">Supprimer</a>
```

Le dossier « public »

Peut contenir les feuilles de styles, scripts, images, etc du site



Pour accéder à un fichier du dossier public

```
<link type="text/css" rel="stylesheet" href="{{ URL::asset('css/style.css') }}">
```

Il est possible également de faire un « echo » PHP si on n'utilise pas Blade.

```
<link rel="stylesheet" type="text/css" href="<?php echo asset('css/style.css'); ?>">
```

10. Crud / Routes et vues

a. Liste

Route

```
Route::get('articles', [
    'as'=> 'articles_index',
    'uses' => 'ArticleController@index'
]);
```

Action du **contrôleur**

```
public function index()
{
    $articles = Article::all();
    return view('article/index')->with('articles',$articles);
}
```

Récupération des données grâce au modèle

Passage à la vue

Vue « index.blade.html »

```
@extends('layouts.app')

@section('title', 'Liste des articles')

@section('content')
    <h1>Articles</h1>
    <hr>
    @if(isset($articles) and count($articles)> 0)
        <a href="{{ route('articles_create') }}" class="btn btn-primary">Ajouter un
nouvel article</a>

        @foreach($articles as $article)
            <article>
                <header class="entry-header">
                    <h2><a href="{{ route('articles_view',['id' => $article->id ])
}}">{{ $article->title }}</a></h2>
                    <small>
                        <em class="text-muted">
                            Posté le <span class="created">{{ date('d M Y',
strtotime($article->created_at)) }}</span>
                            par <span class="author">{{ $article->user->name }}</span>
                        </em>
                    </small>
                </header>
                <div class="entry-content">
                    {{ $article->content }}
                </div>
            </article>

        @endforeach
    @else
        <div class="alert alert-warning text-center">
            Voulez-vous ajouter <a href="{{ route('articles_create') }}">le premier
article</a>?
        </div>
    @endif

@endsection
```

S'il y a des articles on les affiche sinon affihage d'un message pour afficher le premier article

Formater une date

```
{{ date('d M Y', strtotime($article->created_at)) }}
```

b. Vue détails

Route

```
Route::get('articles/view/{id}', [
    'as'=> 'articles_view',
    'uses' => 'ArticleController@view'
]);
```

Action du **contrôleur**

On passe l'id, on récupère l'article correspondant et on le passe à la vue

```
public function view($id)
{
    $article = Article::where('id','=', $id)->first();
    return view('article/view')->with('article',$article);
}
```

Vue « view.blade.html »

```
@extends('layouts.app')

@section('title', 'Détails de l\'article')

@section('content')

    <section>
        <header class="entry-header">
            <h1>{{ $article->title }}</h1>
            <small>
                <em class="text-muted">
                    Posté le <span class="created">{{ date('d M Y',
strtotime($article->created_at)) }}</span>
                    par <span class="author">{{ $article->user->name }}</span>
                </em>
            </small>
        </header>
        <div class="pull-right">
            <a href="{{ route('articles_edit',  ['id'=> $article->id])}}" class="btn
btn-default">Modifier</a>
            <a href="{{ route('articles_delete',  ['id'=> $article->id])}}" class="btn
btn-primary">Supprimer</a>
        </div>
        <div class="entry-content">
            {{  $article->content }}
        </div>
    </section>

@endsection
```

Liens pour l'édition et suppression de l'article

c. Formulaires et validation

Documentation validation

Formulaire d'ajout

Route

```
Route::match(['get','post'],'articles/create', [
    'as'=> 'articles_create',
    'uses' => 'ArticleController@create',
    'middleware' => 'auth'
]);
```

Action du **contrôleur**

- On récupère les informations du formulaire avec « **$request** »
- Validation de formulaire. Si le formulaire n'est pas validé automatiquement la page du formulaire est affichée avec les erreurs.
- On sauvegarde le nouvel article avec la fonction « **save** » du modèle
- **Redirection** vers la page détails de l'article avec **message flash**

```
public function create(Request $request)
{
    if($request->isMethod('post')) {
        $this->validate($request,
            [
                'title' => 'required',
                'content' => 'required'
            ],
            [
                'title.required'=>'Vous devez renseigner un titre',
                'content.required'=>'Vous devez ajouter un contenu'
            ]);
        $article = new Article();
        $article->title = $request->input('title');
        $article->content = $request->input('content');
        $article->user_id = $request->user()->id;
        $article->save();

        return redirect()->route('articles_view',array('id' => $article->id))-
>with('success', 'L\'article a été ajouté.');
    }
    else{
        return view('article/create');
    }
}
```

Règles de validation

Personnalisation des messages d'erreur

```
<form method="post">
    {{ csrf_field() }}
    <div class="form-group">
        <label for="title">Titre</label>
        <input type="text" id="title" name="title" class="form-control"
placeholder="Saisissez votre titre ici">
    </div>
    <div class="form-group">
        <textarea rows="10" id="content" name="content" class="form-
control"></textarea>
    </div>
    <input type="submit" value="Publier" class="btn btn-default">
</form>
```

```
@if (count($errors) > 0)
    <div class="alert alert-danger">
        <ul>
            @foreach ($errors->all() as $error)
                <li>{{ $error }}</li>
            @endforeach
        </ul>
    </div>
@endif
```

Affichage des erreurs de validation

Formulaire d'édition

Route

```
Route::match(['get','post'],'articles/edit/{id}', [
    'as'=> 'articles_edit',
    'uses' => 'ArticleController@edit',
    'middleware' => 'auth'
]);
```

Action du contrôleur

Identifiant de l'article passé

```
public function edit(Request $request, $id)
{
    $article = Article::where('id','=', $id)->first();
    if($request->isMethod('post')) {

        $this->validate($request,
            [
                'title' => 'required',
                'content' => 'required'
            ],
            [
                'title.required'=>'Vous devez renseigner un titre',
                'content.required'=>'Vous devez ajouter un contenu'
            ]);
        $article->title = $request->input('title');
        $article->content = $request->input('content');
        $article->save();

        return redirect()->route('articles_view',array('id' => $article->id))->with('success', 'L\'article a été modifié.');
     }
    else{
        return view('article/edit')->with('article',$article);
    }
}
```

Passage de l'article à la vue

Formulaire

```
<form method="post">
    {{ csrf_field() }}
    <div class="form-group">
        <label for="title">Titre : </label>
        <input type="text" id="title" name="title" class="form-control" value="{{ $article->title }}">
    </div>
    <div class="form-group">
        <textarea rows="10" id="content" name="content" class="form-control">{{ $article->content }}</textarea>
    </div>
    <input type="submit" value="Publier" class="btn btn-default">
</form>
```

d. Suppresion

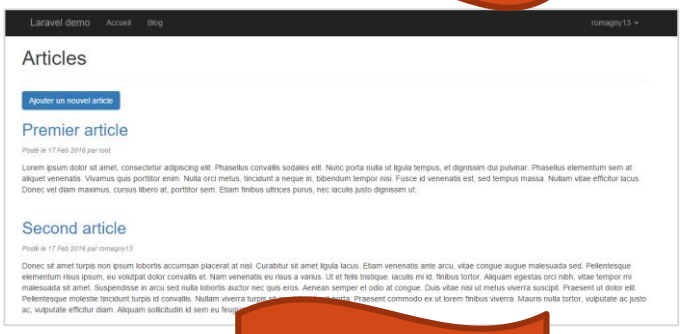
Route

```
Route::get('/articles/{id}', [
    'as'=> 'articles_delete',
    'uses' => 'ArticleController@delete',
    'middleware' => 'auth'
]);
```

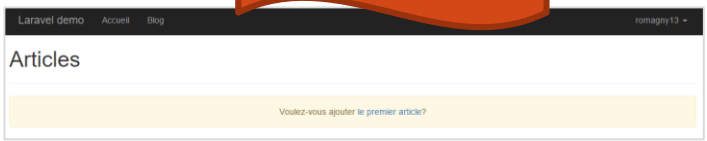
Action du **contrôleur**

```
public function delete($id)
{
    $article = Article::where('id','=', $id)->first();
    $article->delete();
    return redirect()->route('articles_index')-
>with('success','L\'article a été supprimé.');
}
```

Liste



Si aucun article



Détails

Formulaire avec validation

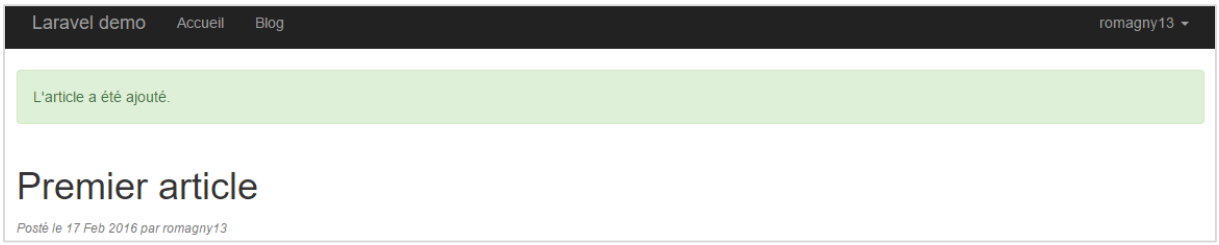
11. Messages Flash

Depuis les **actions** d'un **contrôleur** on fait une redirection avec un message par exemple

```
return redirect()->route('articles_view',array('id' => $article->id))->with('success',
'L\'article a été ajouté.');
```

Affichage des messages dans la **Master page** (ou une vue partielle comme le header)

```
<div class="container">
    @if(Session::has('success'))
        <div class="alert alert-success">{{ Session::get('success')
}}</div>
    @endif
    @if(Session::has('error'))
        <div class="alert alert-danger">{{ Session::get('error') }}</div>
    @endif
</div>
```



12. Authentification

Documentation authentification et autorisation

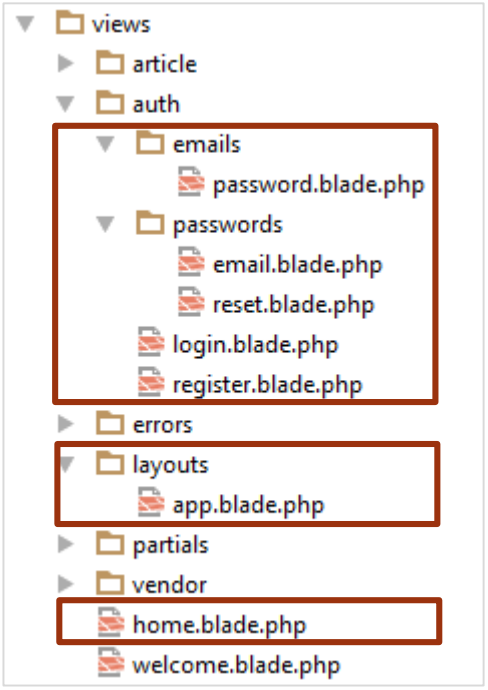
a. Installation

Si ce n'est pas fait installer les vues d'authentification, master pages, routes, etc.

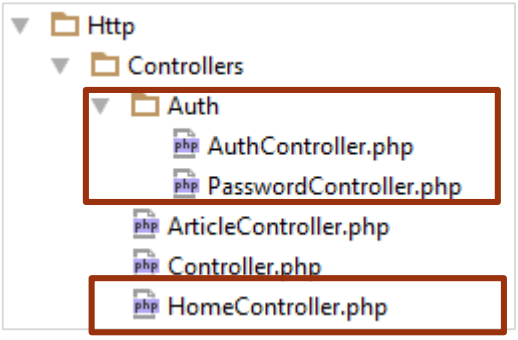
Attention certaines vues (comme « welcome ») ou la Master Page si elle se nomme « app.blade.php » risquent d'être remplacées. Voilà pourquoi il vaut mieux faire cette action avec un projet venant d'être créé.

```
php artisan make:auth
```

Vues



Contrôleurs



b. Utilsateur

Depuis le contrôleur

Savoir si l'utilisateur est **connecté**

```
if (Auth::check()) {

}
```

Récupérer l'utilisateur connecté

Soit avec

```
$user = Auth::user();
```

Soit

```
$request->user()
```

c. Protéger une route

On ajoute le **middleware** « **auth** » à la **route**. Automatiquement l'utilisateur sera redirigé vers la page de connexion, puis une fois cela fait il sera redirigé vers la page qu'il voulait consulter.

```
Route::match(['get','post'],'articles/create', [
    'as'=> 'articles_create',
    'uses' => 'ArticleController@create',
    'middleware' => 'auth'
]);
```

Il est possible de l'application à un **contrôleur** également

```
public function __construct()
{
    $this->middleware('auth');
}
```

d. Menu connexion, inscription

Dans la Master Page ou dans une vue partielle « header »

```
<div class="navbar-inverse">
    <div class="container">
        <div class="navbar-header">
            <button type="button" class="navbar-toggle" data-toggle="collapse" data-
target=".navbar-collapse">
                <span class="icon-bar"></span>
                <span class="icon-bar"></span>
                <span class="icon-bar"></span>
            </button>
            <a href="{{ route('homepage') }}" class="navbar-brand">Laravel demo</a>
        </div>
        <div class="navbar-collapse collapse">
            <ul class="nav navbar-nav">
                <li><a href="{{ route('homepage') }}">Accueil</a></li>
                <li><a href="{{ route('articles_index') }}">Blog</a></li>
            </ul>

            <ul class="nav navbar-nav navbar-right">
                @if (Auth::guest())
                    <li><a href="{{ url('/login') }}">Se connecter</a></li>
                    <li><a href="{{ url('/register') }}">S'inscrire</a></li>
                @else
                    <li class="dropdown">
                        <a href="#" class="dropdown-toggle" data-toggle="dropdown"
role="button" aria-expanded="false">
                            {{ Auth::user()->name }} <span class="caret"></span>
                        </a>
                        <ul class="dropdown-menu" role="menu">
                            <li><a href="{{ url('/home') }}">Dashboard</a></li>
                            <li><a href="{{ url('/logout') }}">Se déconnecter</a></li>
                        </ul>
                    </li>
                @endif
            </ul>

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

Si utilisateur non connecté

Utilisateur connecté

e. Personnaliser les formulaires générés

Exemple avec « login.blade.php »

```
@extends('layouts.app')

@section('content')

    <div class="col-md-offset-4 col-md-4">
        <h3>Se connecter</h3>
        <hr>
        <form method="post" action="{{ url('/login') }}" class="form-horizontal">
            {!! csrf_field() !!}
            <div class="form-group{{ $errors->has('email') ? ' has-error' : '' }}">
                <label for="email">Email</label>
                <input type="email" class="form-control" name="email" value="{{
old('email') }}">
                @if ($errors->has('email'))
                    <span class="help-block">
                        <strong>{{ $errors->first('email') }}</strong>
                     </span>
                @endif
            </div>
            <div class="form-group{{ $errors->has('password') ? ' has-error' : '' }}">
                <label for="password">Mot de passe</label>
                <input type="password" class="form-control" name="password">
                @if ($errors->has('password'))
                    <span class="help-block">
                        <strong>{{ $errors->first('password') }}</strong>
                    </span>
                @endif
            </div>
            <div class="text-center form-group">
                <input type="checkbox" name="remember"> Se souvenir de moi
            </div>

            <div class="text-center form-group">
                <button type="submit" class="btn btn-primary">Se connecter</button>
                  ou  
                <a href="/register">S'inscrire</a>
            </div>
            <div class="text-center form-group">
                <a class="btn btn-link" href="{{ url('/password/reset') }}">Mot de passe
oublié?</a>
            </div>
        </form>
    </div>

@endsection
```

Et « register.blade.php »

```
@extends('layouts.app')

@section('content')

    <div class="col-md-offset-4 col-md-4">
        <h3>Inscription</h3>
        <hr>
        <form class="form-horizontal" role="form" method="POST" action="{{
url('/register') }}">
```

```
            {!! csrf_field() !!}
            <fieldset>
            <div class="form-group{{ $errors->has('name') ? ' has-error' : '' }}">
                <label for="username">Nom d'utilisateur</label>
                <div class="controls">
                    <input type="text" class="form-control" name="name" value="{{
old('name') }}">
                    @if ($errors->has('name'))
                        <span class="help-block">
                            <strong>{{ $errors->first('name') }}</strong>
                        </span>
                    @endif
                </div>
            </div>
            <div class="form-group{{ $errors->has('email') ? ' has-error' : '' }}">
                <label for="email">Email</label>
                <div class="controls">
                    <input type="email" class="form-control" name="email" value="{{
old('email') }}">
                    @if ($errors->has('email'))
                        <span class="help-block">
                                    <strong>{{ $errors->first('email') }}</strong>
                                </span>
                    @endif
                </div>
            </div>
            <div class="form-group{{ $errors->has('password') ? ' has-error' : '' }}">
                <label for="password">Mot de passe</label>
                <div class="controls">
                    <input type="password" class="form-control" name="password">
                    @if ($errors->has('password'))
                        <span class="help-block">
                                    <strong>{{ $errors->first('password') }}</strong>
                                </span>
                    @endif
                </div>
            </div>
            <div class="form-group{{ $errors->has('password_confirmation') ? ' has-error'
: '' }}">
                <label for="username">Confirmer le mot de passe</label>
                <div class="controls">
                    <input type="password" class="form-control"
name="password_confirmation">
                    @if ($errors->has('password_confirmation'))
                        <span class="help-block">
                            <strong>{{ $errors->first('password_confirmation') }}</strong>
                         </span>
                    @endif
                </div>
            </div>
            <div class="text-center form-group">
                <button type="submit" class="btn btn-primary">S'inscrire</button>
                  ou  
                <a href="/login">Se connecter</a>
            </div>
        </fieldset>
      </form>
    </div>

@endsection
```

f. Personnaliser les messages d'erreur de validation

Dans « **AuthController** »

```php
/**
 * Get a validator for an incoming registration request.
 *
 * @param  array  $data
 * @return \Illuminate\Contracts\Validation\Validator
 */
protected function validator(array $data)
{
    return Validator::make($data, [
        'name' => 'required|max:255',
        'email' => 'required|email|max:255|unique:users',
        'password' => 'required|confirmed|min:6',
    ],[
        'name.required' => 'Vous devez indiquer un nom d\'utilisateur',
        'email.required' => 'Un email est requis',
        'password.required' => 'Vous devez indiquer un mot de passe',
        'password.confirmed' => 'Les mots de passe ne correspondent pas',
        'password.min' => 'Le mot de passe doit faire au moins 6 caractères'
    ]);
}
```

13. Connexion avec les réseaux sociaux et « Socialite »

Documentation

a. Installation

```
composer require laravel/socialite
```

b. Configuration

```
#config/app.php

'providers' => [
    // etc.
    App\Providers\EventServiceProvider::class,
    App\Providers\RouteServiceProvider::class,
    Laravel\Socialite\SocialiteServiceProvider::class,
],

'aliases' => [
    // etc.
    'Socialite' => Laravel\Socialite\Facades\Socialite::class,
],
```

Créer des applications :

- Facebook
- Google :
  - Activer l'API Google +
  - Créer les identifiants avec « ID client OAuth ». Indiquer l'url d'origine et de redirection autorisées
  - Récupérer les identifiants

Dans le fichier « **services.php** » dans le dossier « config »

```
'facebook' => [
    'client_id'     => '1748…7764',
    'client_secret' => '792eca40770…d26ed0cc1fcf',
    'redirect'      =>
'http://localhost/laravelcms/public/login/callback/facebook',
],
'google' => [
    'client_id'     => '1007…93apps.googleusercontent.com',
    'client_secret' => 'RD3uwlbz…ipjyxr1l',
    'redirect'      =>
'http://localhost/laravelcms/public/login/callback/google',
],
```

Migration et mise à jour de la base

« Socialite » ne fait pas tout, il faut soi-même modifier la base de données et la table « users »

Créer une migration pour modifier la table « users » (exemple « update_users_table »

```
php artisan make:migration update_users_table
```

```php
<?php

use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;

class UpdateUsersTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::table('users', function ($table) {
            $table->string('provider');
            $table->text('social_id');
            $table->string('role')->default('role_user');
        });
    }

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

Note on pourrait revoir et améliorer la structure, créer une table « role » par exemple mais ici c'est un simple exemple.

Mettre à jour la base (attention les tables seront vidées)

```
php artisan migrate:refresh
```

Routes

```php
Route::get('/login/{provider}',[
    'as'   => 'auth_social_login',
    'uses' => 'Auth\AuthController@redirectToProvider'
]);


Route::get('/login/callback/{provider}',[
    'as'   => 'auth_social_callback',
    'uses' => 'Auth\AuthController@handleProviderCallback'
]);
```

Ajouter deux actions au contrôleur « **AuthController** »

« Socialite » ne fait pas tout, il faut mettre à jour la base de données

Redirection vers le réseau social. L'utilisateur accepte les permissions la première fois

Callback avec les informations de l'utilisateur

```
  public function redirectToProvider($provider)
    {
        return Socialite::driver($provider)->redirect();
    }

    public function handleProviderCallback($provider)
    {

        $user = Socialite::driver($provider)->user();
        $userCheck = User::where('email', '=', $user->email)->first();
        if(empty($userCheck))
        {
            // register
            $newUser = new User;
            $newUser->email = $user->email;
            $parts = explode("@", $user->email);
            $username = $parts[0];
            $newUser->name = $username;
            $newUser->provider = $provider;
            $newUser->social_id = $user->id;
            $newUser->password = bcrypt(str_random(20));
            $newUser->save();

            Auth::login($newUser, true);
            return redirect()->route('articles_index')->with('success', 'Vous êtes
connecté.');
        }
        else
        {
            // verifier provider
            if($userCheck->provider == $provider)
            {
                Auth::login($userCheck, true);
                return redirect()->route('articles_index')->with('success', 'Vous êtes
connecté.');
            }
            else
            {
                return redirect('/login')->with('error', 'Un utilisateur est déja
enregistré avec cet email.');
            }
        }
    }
```

Ne pas oublier

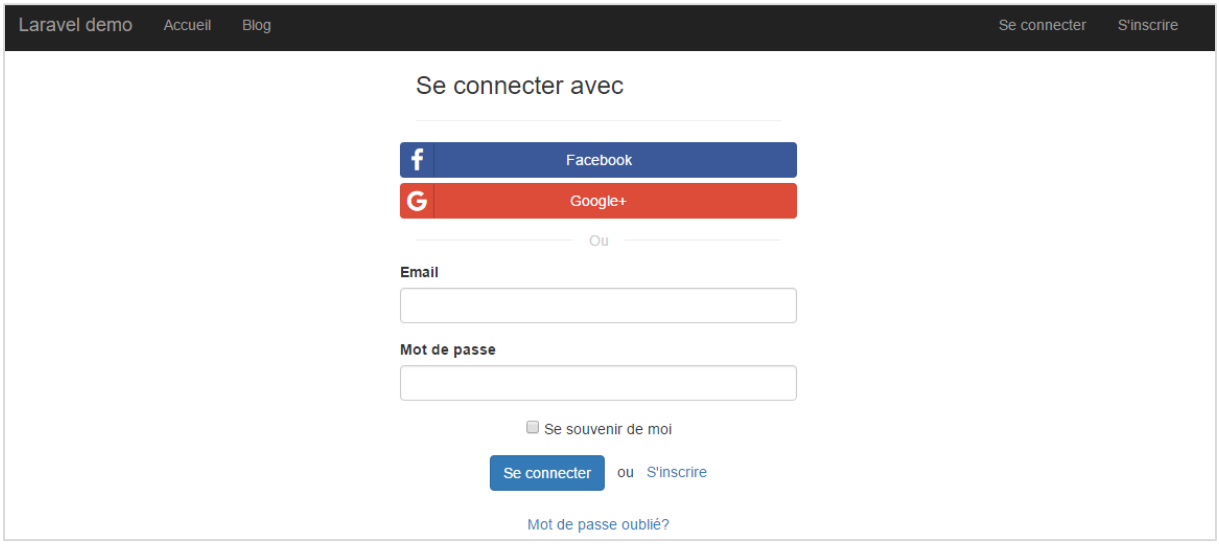
```
use Socialite;
```

Ajout de liens dans la vue « login.blade.php »

```
<a href="{{ route('auth_social_login',['provider'=>'facebook']) }}">Facebook</a>
<a href="{{ route('auth_social_login',['provider'=>'google']) }}">Google+</a>
```

On peut utiliser Bootstrap social si on veut mettre en forme ses liens.

```
<a class="btn btn-block btn-social btn-facebook" href="{{
route('auth_social_login',['provider'=>'facebook']) }}">
<span class="fa fa-facebook"></span>
Facebook
</a>
<a class="btn btn-block btn-social btn-google" href="{{
route('auth_social_login',['provider'=>'google']) }}">
    <span class="fa fa-google"></span>
    Google+
</a>
```



Problème de certificat SSL en local (exemple avec Wamp)

- Télécharger le fichier cacert.pem et le copier à la racine de Wamp.
- Ouvrir **PHP.ini** (exemple « C:\wamp\bin\apache\apache2.4.9\bin », sinon on peut obtenir le chemin avec « phpinfo() ») et rajouter la ligne tout en bas du fichier dans la partie « [curl] » (on indique le chemin vers le fichier cacert.pem)

```
curl.cainfo="c:/wamp/cacert.pem"
```