Laravel Boilerplate

Lorsqu'on utilise fréquemment Laravel on est amené à effectuer des tâches répétitives et à utiliser une certain nombre de classes et fonctionnalités à travers différents projets. On pourrait ainsi imaginer une trame de base comportant tout ce qu'on utilise habituellement. C'est en gros ce qui est réalisé par Laravel Boilerplate. Voyons un peu ce qui se cache dans cette librairie qui a obtenu quand même plus de 2600 stars sur Github…

Installation

On dispose d'un site plutôt bien fait :

Il y a une page de démarrage rapide (Quick Start). Là il y a la liste des fonctionnalités, des copies d'écran, des explications pour le téléchargement et l'installation.

Comme c'est juste un Laravel amélioré l'installation est assez classique. On va commencer par récupérer le dépôt :

```
git clone https://github.com/rappasoft/laravel-5-boilerplate.git boilerplate
```

On trouve un fichier **.env.example** plutôt bien garni par rapport à celui de base de Laravel, il faut le renommer en **.env**.

Ensuite il n’y a plus qu’à lancer l’installation :

```
cd boilerplate
composer install
```

Ça dure un moment parce qu’il y a pas mal de package prévus…

Il faut ensuite générer une clé pour le cryptage :

```
php artisan key:generate
```

Ensuite on crée un base de données et on renseigne **.env** :

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=boilerplate
DB_USERNAME=root
DB_PASSWORD=
```

On peut alors lancer les migrations et la population :

```
php artisan migrate --seed
```

On se retrouve avec 12 tables :

Table
cache
jobs
migrations
model_has_permissions
model_has_roles
password_resets
permissions
roles
role_has_permissions
sessions
social_accounts
users
12 tables

Avec 3 utilisateurs par défaut :

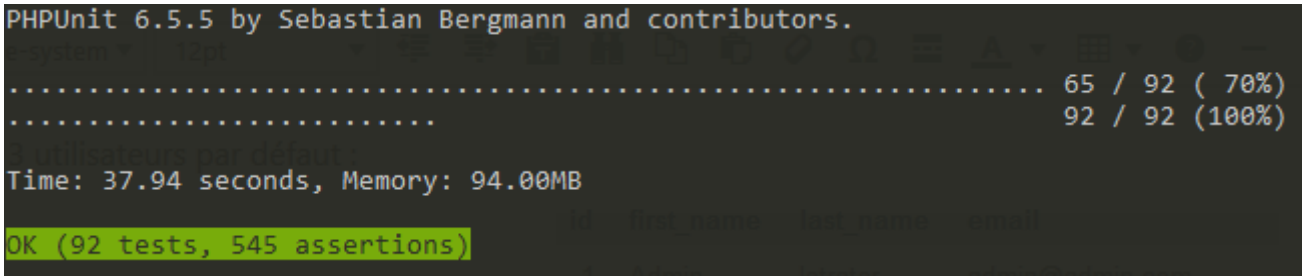
id	first_name	last_name	email
1	Admin	Istrator	admin@admin.com
2	Backend	User	executive@executive.com
3	Default	User	user@user.com

Pour le frontend on a le choix entre **npm** et **yarn**. Personnellement j'utilise **npm**, donc il faut installer :

```
npm install
```

Les 1432 packages mettent un petit moment à s'installer…

On trouve aussi de nombreux tests qu'on peut lancer avec **phpUnit** :

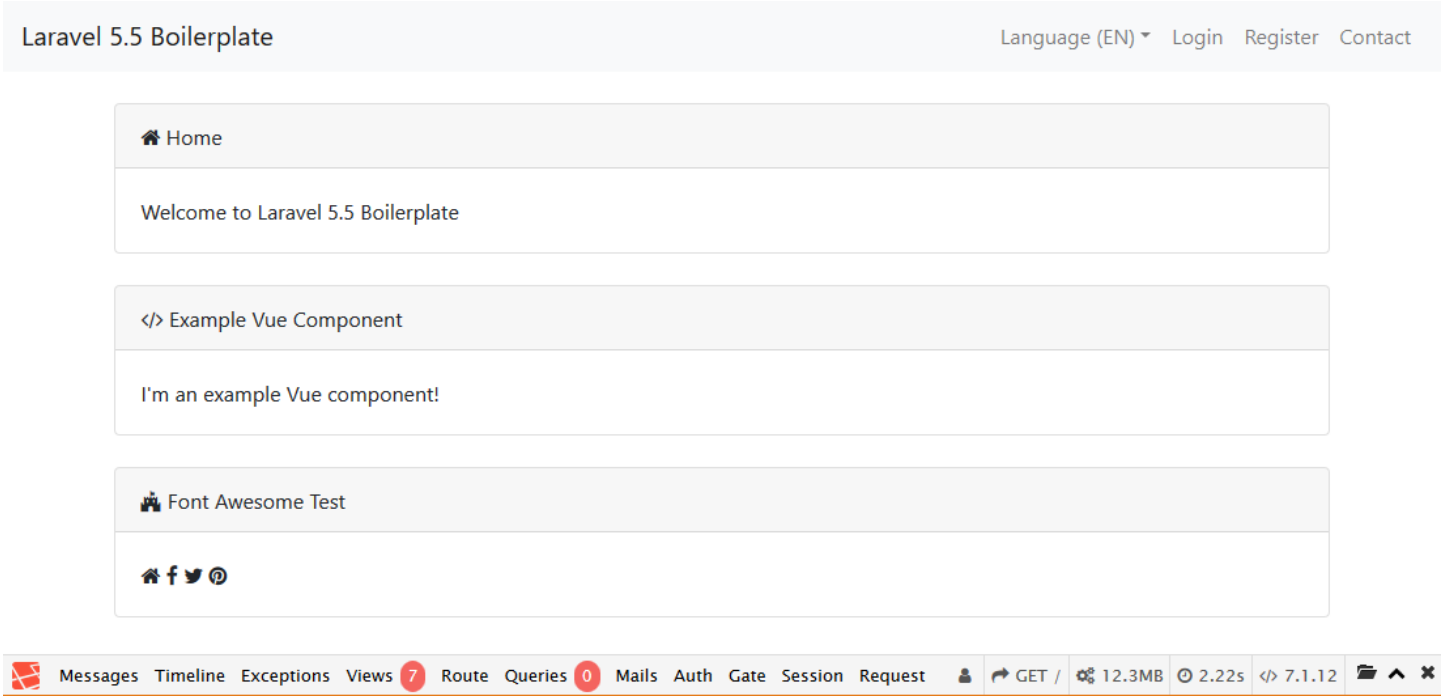


*Il faut prévoir l'extension **pdo_sqlite** de PHP et également renseigner la configuration des mails pour éviter de tomber sur*

une erreur.

État des lieux

Tous semble correct alors je lance :

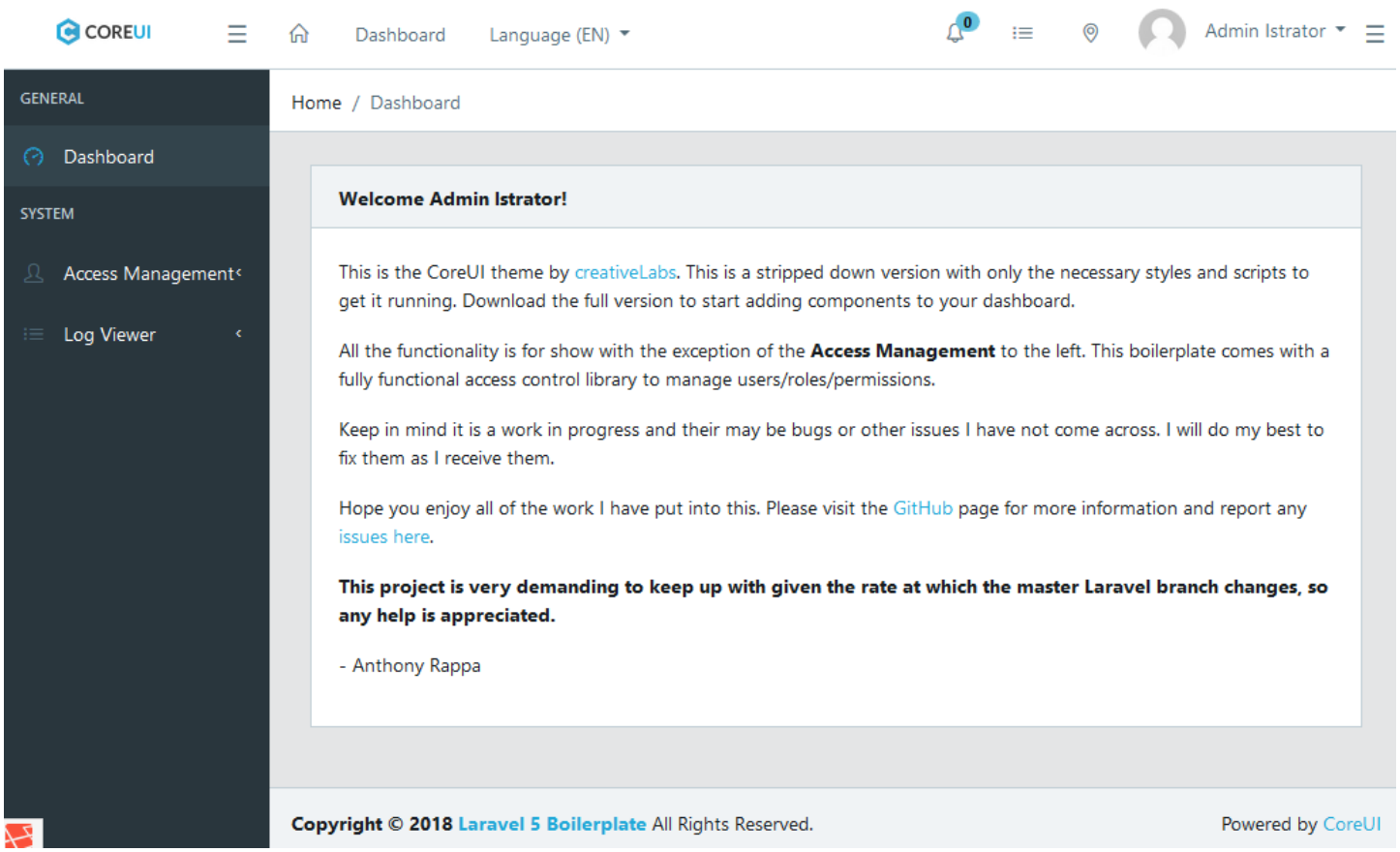


Une page d'accueil sommaire avec une barre de navigation, un accès au login et à l'enregistrement, un large choix de langues, une page de contact avec un formulaire et la barre de débogage.

On peut accéder à l'administration avec :

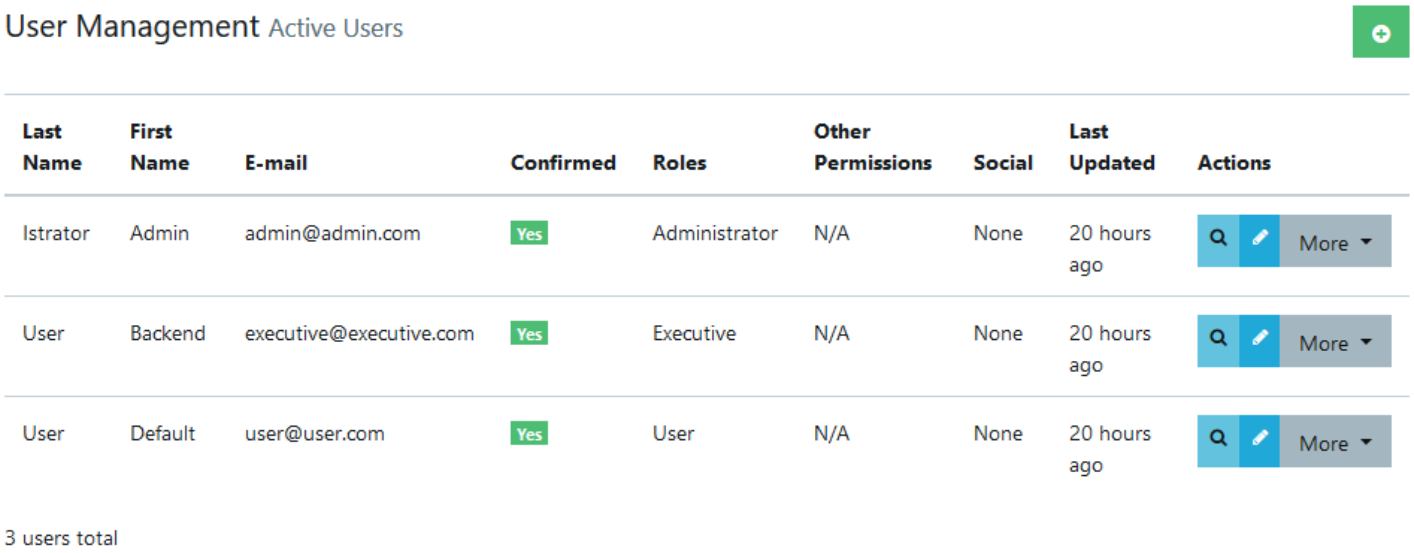
Username: admin@admin.com

Password: 1234



Le template d'administration est CoreUI dans sa version gratuite, c'est à dire pratiquement sans plugins. Personnellement je préfère AdminLTE.

On trouve une gestion des utilisateurs :



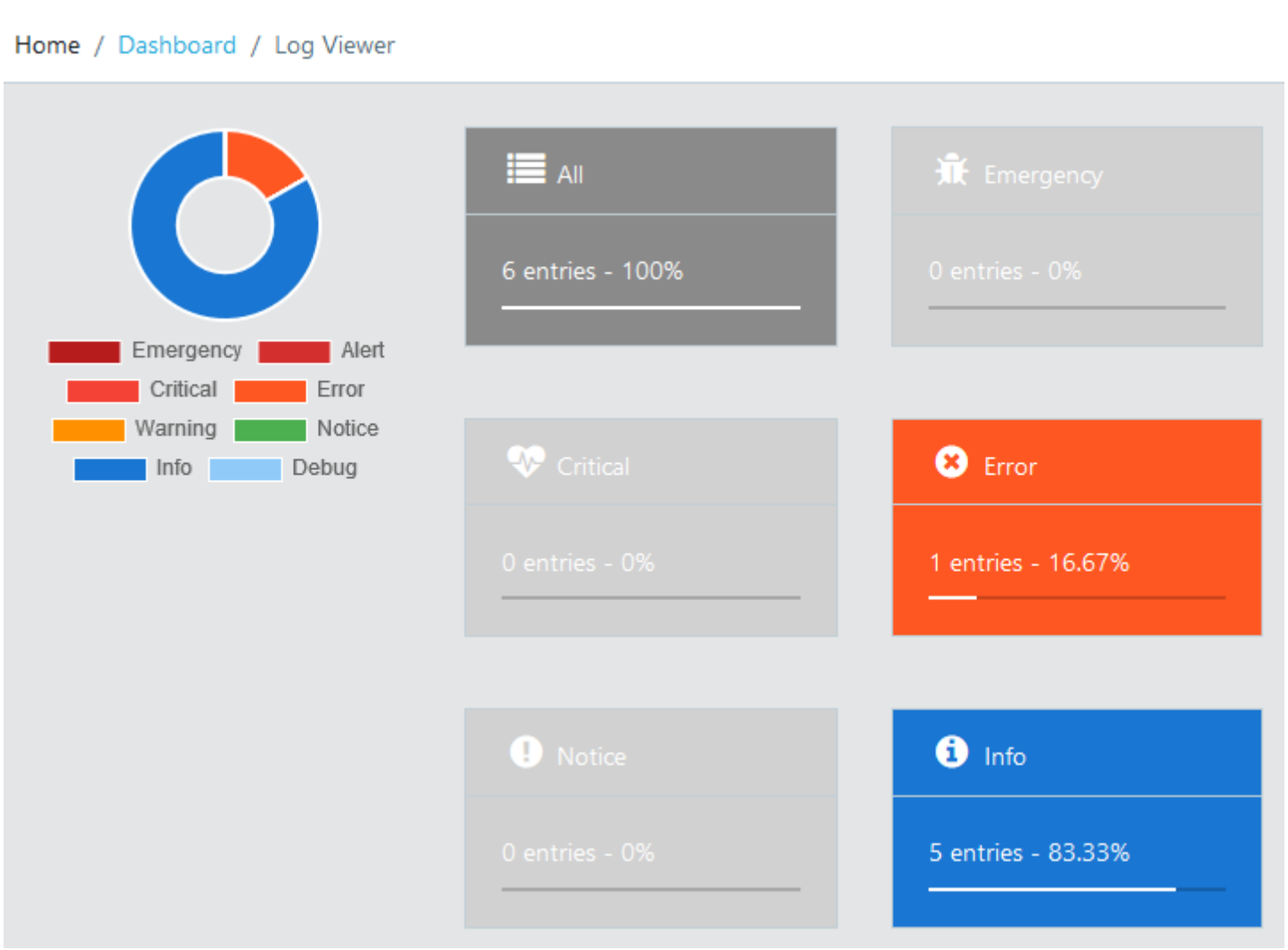
Une gestion des rôles :

Role Management

Role	Permissions	Number of Users	Actions
Administrator	All	1	N/A
Executive	View Backend	1	
User	None	1	

3 roles total

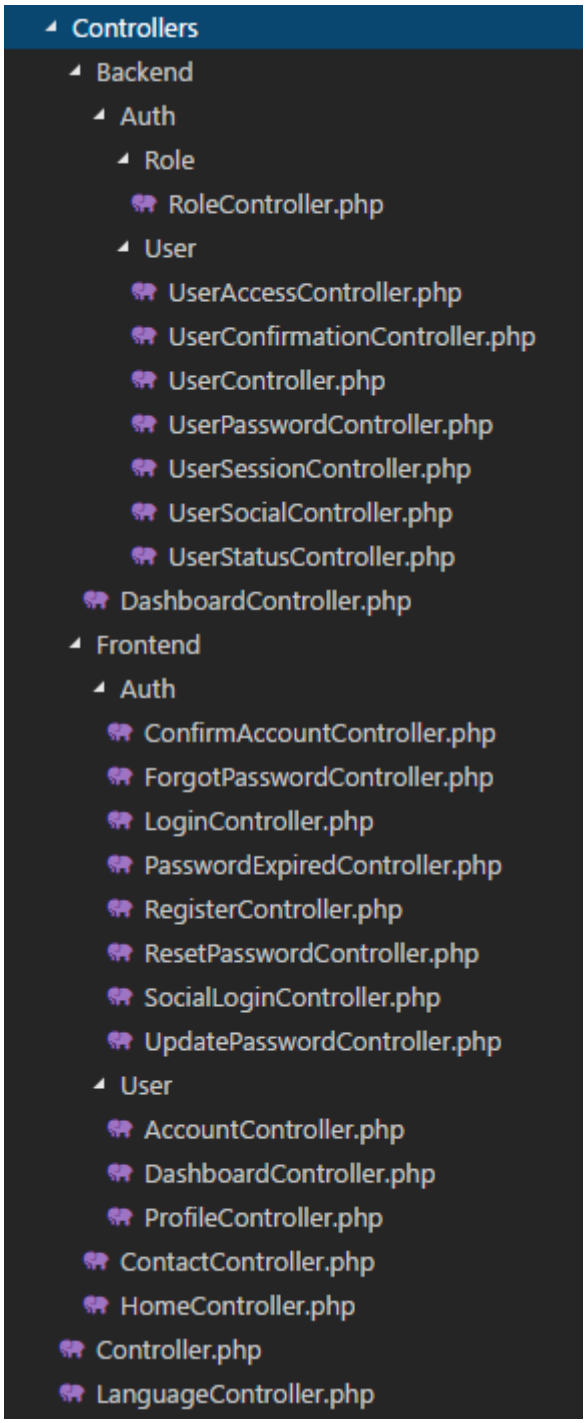
Une visualisation des logs :



Pour en apprendre plus il faut plonger dans la documentation ou fouiller un peu le code…

Les contrôleurs

On trouve de nombreux contrôleur rangés dans leur dossier et sous-dossiers :



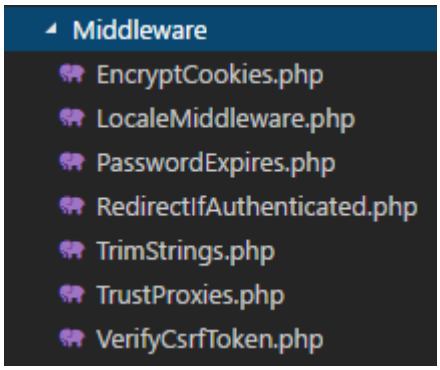
Les validations sont systématiquement réalisées par des **Form Request** et la gestion des données au travers de **repositories**. Du coup le code est propre :

```
/**
 * @param ManageUserRequest $request
```

```
     *
     *                                  @return
\Illuminate\Contracts\View\Factory|\Illuminate\View\View
     */
    public function index(ManageUserRequest $request)
    {
        return view('backend.auth.user.index')
            ->withUsers($this->userRepository->getActivePaginated(25,
'id', 'asc'));
    }
```

Les middlewares

On trouve ces middlewares :



On remarque l'ajout de :

- **LocaleMiddleware** pour la gestion des locales associé à la configuration **config/locale.php**
- **PasswordExpires** si on veut une expiration du mot de passe au bout d'un certain délai

D'autre part on a aussi les middlewares du package spatie/laravel-permission pour la gestion des rôles :

- **RoleMiddleware**
- **PermissionMiddleware**

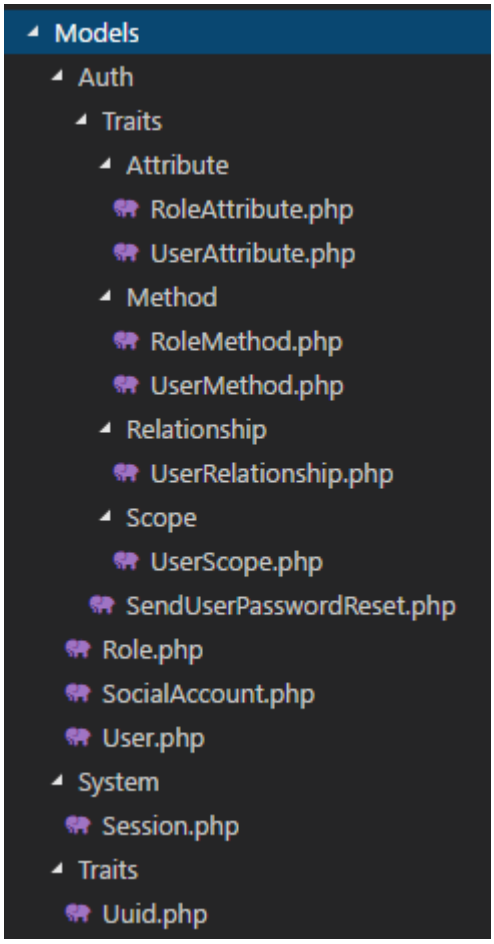
L'ajout de ces middlewares permet de protéger facilement les routes :

```
Route::group([
    'middleware' => 'role:administrator',
```

], function () {

Les modèles

Les modèles se trouvent dans le dossier **app/Models** :



On trouve de nombreux traits. Ils servent pour gérer les attributs, les relations, les scopes et les méthodes spécifiques. Ça fait au final pas mal de fichiers. Par exemple pour le modèle **User** on trouve une rafale de traits :

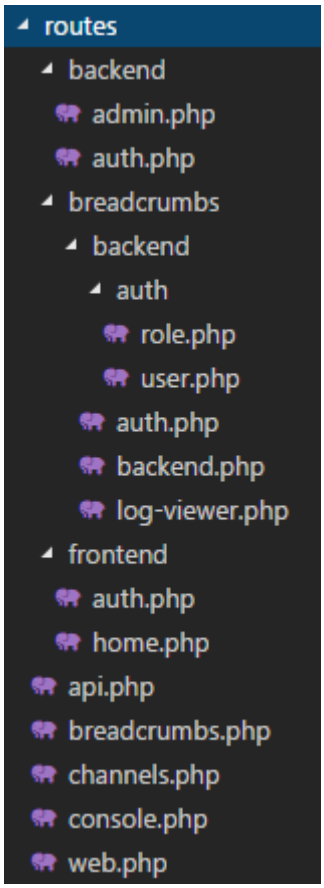
```
class User extends Authenticatable
{
    use HasRoles,
        Notifiable,
        SendUserPasswordReset,
        SoftDeletes,
        UserAttribute,
        UserMethod,
        UserRelationship,
        UserScope,
```

```
        Uuid;
```

C’est un choix architectural…

Les routes

Là aussi on trouve de nombreux fichiers :



Pour comprendre l’organisation il faut regarder le code dans **routes/web.php** :

```
/*
 * Frontend Routes
 * Namespaces indicate folder structure
 */
Route::group(['namespace' => 'Frontend', 'as' => 'frontend.'],
function () {
    include_route_files(__DIR__.'/frontend/');
});
```

L’espace de nom est analogue au chemin des dossiers. Ici on va dans le dossier **frontend**. Dans ce dossier dans le fichier **auth.php** on trouve par exemple :

```
Route::group(['namespace' => 'Auth', 'as' => 'auth.'], function ()
{
```

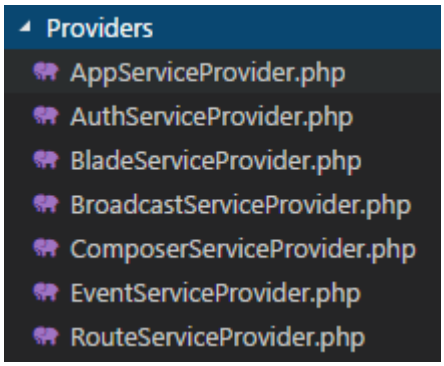
Du coup les routes seront préfixées par **frontend.auth** :

```
frontend.auth.login
frontend.auth.login.post
frontend.auth.social.login
frontend.auth.
frontend.auth.logout
frontend.auth.logout-as
frontend.auth.password.email.post
frontend.auth.password.expired.update
frontend.auth.password.expired
frontend.auth.password.email
frontend.auth.password.reset
frontend.auth.password.reset.form
frontend.auth.password.update
frontend.user.profile.update
frontend.auth.register
frontend.auth.register.post
```

Il faut juste un peu s'habituer au système…

Les providers

On trouve ces providers :



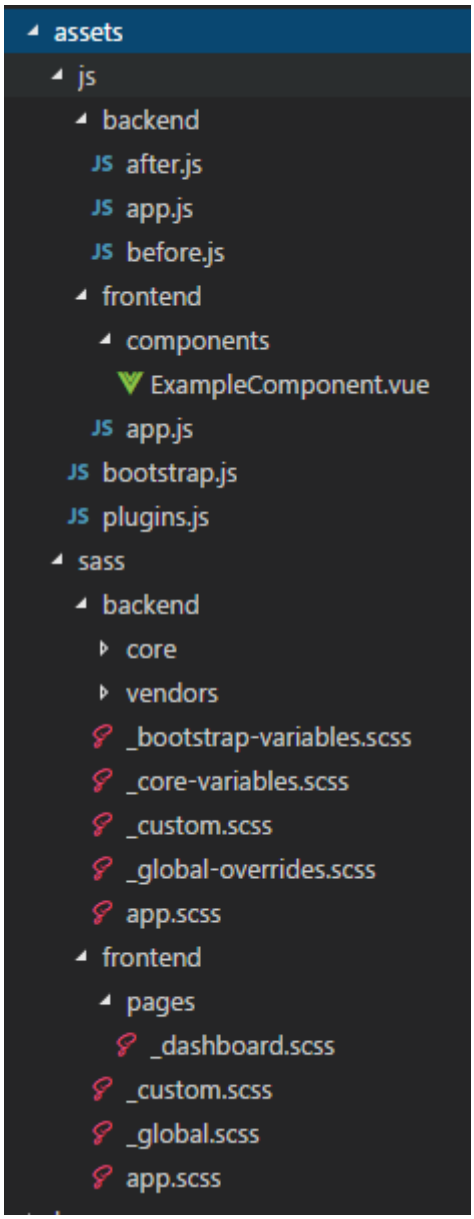
On a donc l'ajout de :

- **BladeServiceProvider** pour ajouter des directives Blade
- **ComposerServiceProvider** pour ajouter des composeurs de vues

Il y a quelques réglages dans **AppServiceProvider** : la locale, le forçage éventuel en **HTTPS**, les templates pour Bootstrap 4…

Les assets

Les assets sont fournis :



On peut mieux comprendre l'organisation en allant jeter un œil dans **webpack.mix.js** :

```
mix.sass('resources/assets/sass/frontend/app.scss',
'public/css/frontend.css')
         .sass('resources/assets/sass/backend/app.scss',
'public/css/backend.css')
             .js('resources/assets/js/frontend/app.js',
'public/js/frontend.js')
    .js([
        'resources/assets/js/backend/before.js',
        'resources/assets/js/backend/app.js',
```

```
        'resources/assets/js/backend/after.js'
    ], 'public/js/backend.js');

if (mix.inProduction() || process.env.npm_lifecycle_event !==
'hot') {
    mix.version();
}
```

Conclusion

Ce boilerplate est bien structuré et codé, c’est une bonne base de départ pour un projet à condition d’accepter l’organisation imposée. Il peut faire gagner du temps au niveau de la constitution du backend. Il est équipé d’une batterie complète de tests. Je regrette juste le choix de CoreUI.