Cours Laravel 5.5 – les notifications

On a vu dans ce cours comment envoyer un email avec Laravel. Mais on dispose aussi d'un système complet de notifications, par exemple par SMS, qui inclue aussi les emails ou même la base de données.

Classiquement une notification est un message court pour informer un utilisateur qu'il s'est passé quelque chose qui le concerne dans l'application.

Par exemple une donnée sensible a été mise à jour, on envoie un SMS par sécurité en informant l'utilisateur de ce changement et, si ce n'est pas lui qui l'a effectué, il peut alors intervenir.

Évidemment pour tout ce qui n'est pas email ou base de données il faut utiliser un service externe. Il y a un site dédié pour tous les drivers existants et la liste est déjà longue !

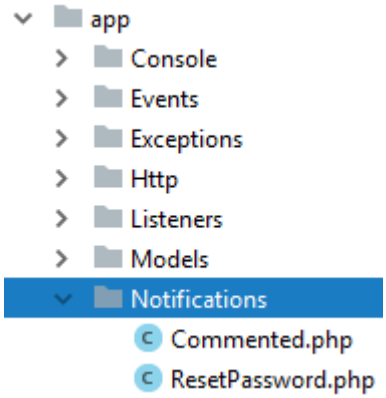
L'application d'exemple utilise les notifications pour l'envoi d'email pour :

- la confirmation de l'adresse email lors de l'enregistrement (par l'intermédiaire d'un package),
- le renouvellement du mot de passe.

Elle utilise également les notifications en base de données pour prévenir les rédacteurs que de nouveaux commentaires ont été ajoutés à leurs articles.

Organisation du code

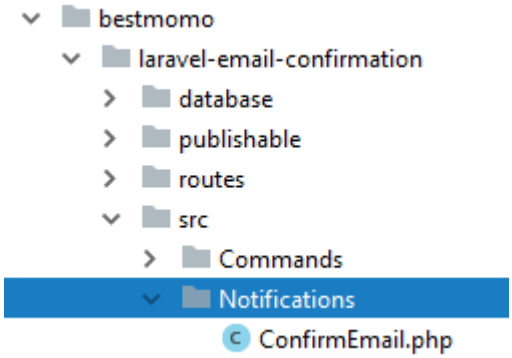
Les notifications se trouvent dans le dossier **app/Notifications** :



Ce dossier n'existe pas dans l'installation de base de Laravel, il est ajouté lorsqu'on crée la première notification avec Artisan qui dispose à cet effet d'une commande :

```
make:notification          Create a new notification class
```

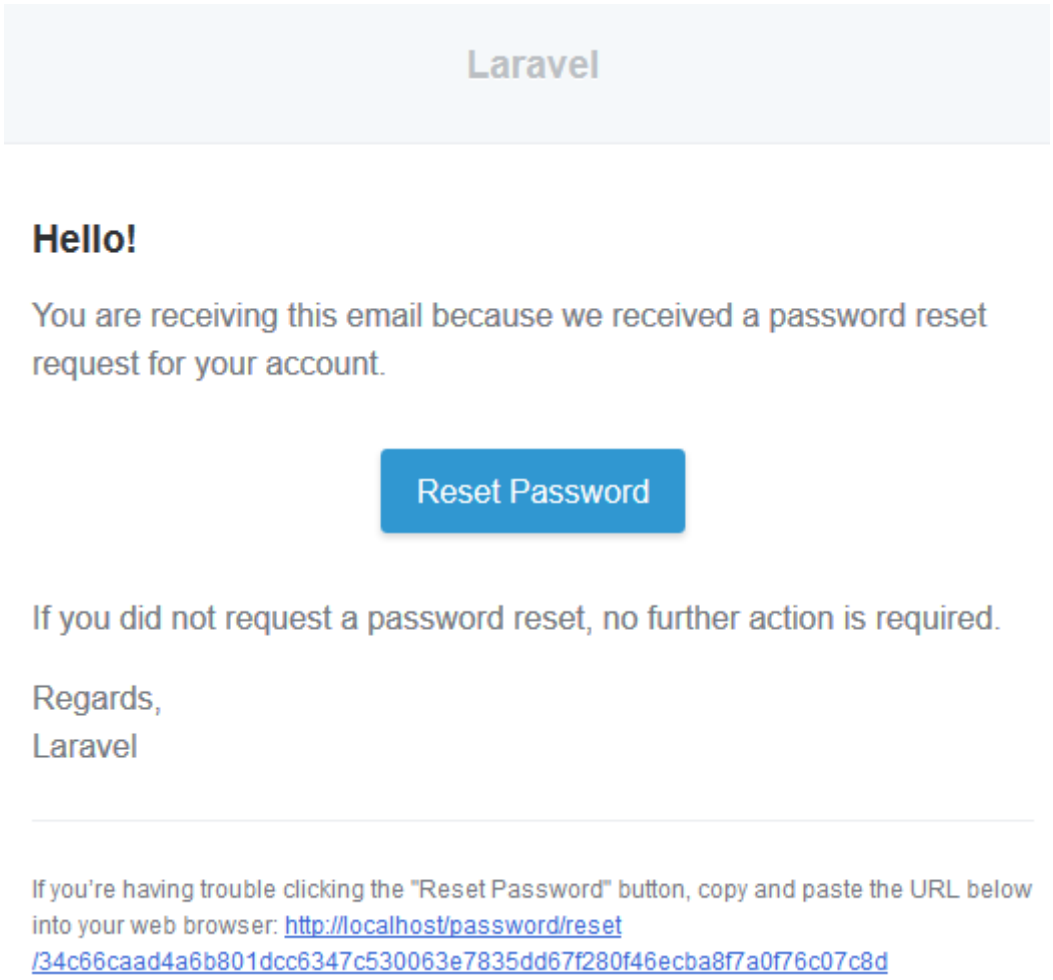
Évidemment pour les packages les notifications se situent dans un dossier du package, par exemple :



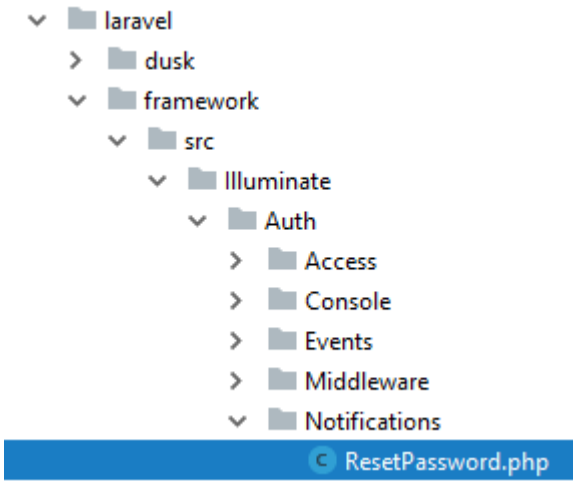
Le renouvellement du mot de passe

On a vu en détail le renouvellement du mot de passe dans ce chapitre. J'ai alors précisé qu'on envoyait un email par le système de notification en précisant que je vous en parlerai plus tard. C'est ce que je vais faire maintenant.

On a vu que Laravel, après la demande de renouvellement de l'utilisateur, expédie ce genre d'email :



Par défaut la classe de notification se situe dans le framework :



Avec ce code :

```
<?php

namespace Illuminate\Auth\Notifications;

use Illuminate\Notifications\Notification;
use Illuminate\Notifications\Messages\MailMessage;
```

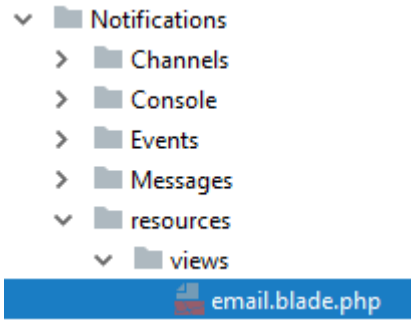
```
class ResetPassword extends Notification
{
    /**
     * The password reset token.
     *
     * @var string
     */
    public $token;

    /**
     * Create a notification instance.
     *
     * @param  string  $token
     * @return void
     */
    public function __construct($token)
    {
        $this->token = $token;
    }

    /**
     * Get the notification's channels.
     *
     * @param  mixed  $notifiable
     * @return array|string
     */
    public function via($notifiable)
    {
        return ['mail'];
    }

    /**
     * Build the mail representation of the notification.
     *
     * @param  mixed  $notifiable
     * @return \Illuminate\Notifications\Messages\MailMessage
     */
    public function toMail($notifiable)
    {
        return (new MailMessage)
                    ->line('You are receiving this email because we
received a password reset request for your account.')
                                ->action('Reset  Password',
```

```
url(config('app.url').route('password.reset',    $this->token, false)))
            ->line('If you did not request a password reset, no further action is required.');
    }
}
```

Comme c'est une notification par email on a une méthode **toMail**. D'autre part on précise dans la méthode **via** qu'on retourne un email. La classe **MailMessage** offre un certain nombre de méthodes bien pratiques comme :

- **line** : pour écrire une ligne
- **action** : pour afficher un bouton
- **from** : pour définir l'adresse de l'expéditeur
- **subject** : pour définir le sujet
- **cc** et **bcc** : pour faire des copies
- **attach** : pour joindre un document
- **priority** : pour fixer la priorité…

Vous constatez qu'on a une mise en forme de l'email plutôt esthétique. cette mise en forme est issues d'une vue par défaut dans le framework :



Si vous voulez modifier cette mise en forme il faut publier cette vue :

```
php artisan vendor:publish --tag=laravel-notifications
```

Vous retrouvez alors la vue dans le dossier **resources/views/vendor/notifications**.

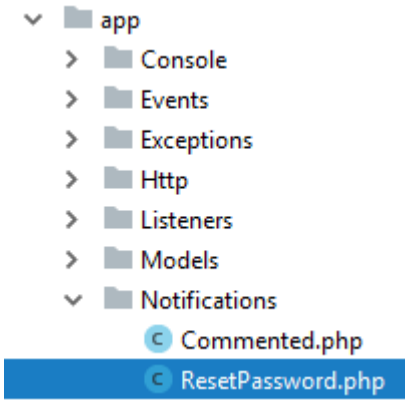
Vous pouvez vous demander maintenant comment est effectivement commandée cette notification. Ça se passe dans le trait **CanResetPassword** avec la méthode **notify** :

```php
<?php

namespace Illuminate\Auth\Passwords;

use Illuminate\Auth\Notifications\ResetPassword as ResetPasswordNotification;

trait CanResetPassword
{
    ...

    public function sendPasswordResetNotification($token)
    {
        $this->notify(new ResetPasswordNotification($token));
    }
}
```

Dans l’application d’exemple on n’utilise pas directement la classe de notification prévue dans le framework parce qu’on veut localiser le texte :

```php
public function toMail()
{
    return (new MailMessage)
          ->line(__('You are receiving this email because we received a password reset request for your account.'))
          ->line(__('Click the button below to reset your password:'))
          ->action(__('Reset Password'), url('password/reset', $this->token))
          ->line(__('If you did not request a password reset, no further action is required.'));
}
```

On a donc cette classe dans l’application :



Et pour informer Laravel d'utiliser cette classe on surcharge la méthode **sendPasswordResetNotification** qu'on a vue ci-dessus présente dans le trait **CanResetPassword** au niveau de notre modèle **User** :

```
<?php

...

use App\Notifications\ResetPassword as ResetPasswordNotification;

class User extends Authenticatable
{
     use Notifiable, IngoingTrait;

    ...

    public function sendPasswordResetNotification($token)
    {
        $this->notify(new ResetPasswordNotification($token));
    }

    ...
}
```

La notification des rédacteurs pour les nouveaux commentaires

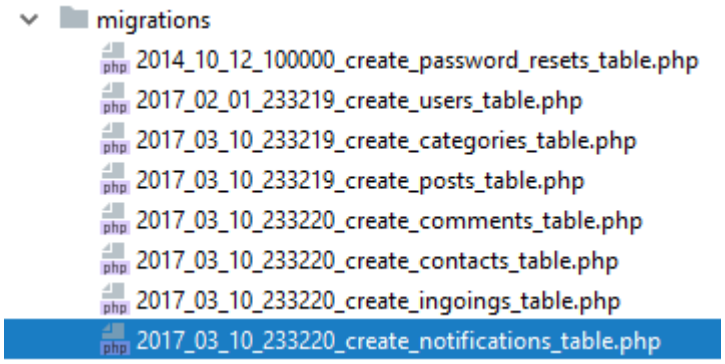
On peut utiliser les notifications en base de données. Elles sont ainsi stockées en attendant d'être lues.

La table des notifications

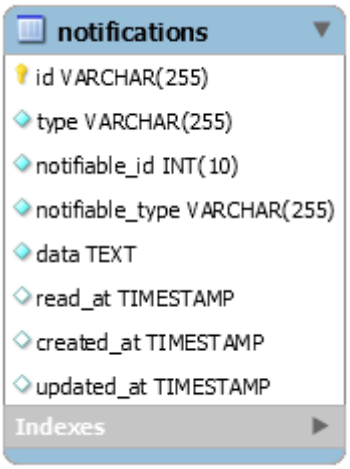
Pour que ça fonctionne il faut ajouter une table avec cette commande d'Artisan :

```
php artisan notifications:table
```

Ce qui a pour effet d'ajouter une migration :



Après avoir effectué la migration on se retrouve avec cette table notifications :



C'est dans cette table que sont stockées les notifications.

La notification

Comme pour les emails il nous faut une classe de notification ;

```
<?php

namespace App\Notifications;

use Illuminate\Notifications\Notification;
```

```
use App\Models\Post;

class Commented extends Notification
{
    /**
     * Post property.
     *
     * @var \App\Models\Post
     */
    protected $post;

    /**
     * User id property.
     *
     * @var integer
     */
    protected $user_id;

    /**
     * Create a new notification instance.
     *
     * @param Post $post
     * @param integer $user_id
     */
    public function __construct(Post $post, $user_id)
    {
        $this->post = $post;
        $this->user_id = $user_id;
    }

    /**
     * Get the notification's delivery channels.
     *
     * @param  mixed  $notifiable
     * @return array
     */
    public function via($notifiable)
    {
        return ['database'];
    }

    /**
     * Get the array representation of the notification.
```

```
     *
     * @param  mixed  $notifiable
     * @return array
     */
    public function toArray($notifiable)
    {
        return [
            'title' => $this->post->title,
            'slug' => $this->post->slug,
            'user_id' => $this->user_id,
        ];
    }
}
```

Cette fois dans la méthode **via** on a **database**.

On voit qu'on doit transmettre dans le constructeur une instance de l'article concerné (**$post**) ainsi que l'identifiant de l'auteur (**user_id**). La mise en forme se fait dans la méthode **toArray** où on retourne un tableau de données qui sera transformé en **JSON** dans la base.

On va ainsi transmettre le titre, le slug (pour générer l'url) de l'article, ainsi que l'identifiant de l'auteur.

Création d'une notification

Une notification sera créée quand un commentaire le sera. Voici la méthode **store** du contrôleur des commentaires (**App\Http\Controllers\Front\CommentController**) :

```
public function store(CommentRequest $request, Post $post, $comment_id = null)
{
    Comment::create ([
        'body' => $request->input('message' . $comment_id),
        'post_id' => $post->id,
        'user_id' => $request->user()->id,
        'parent_id' => $comment_id,
    ]);

            $post->user->notify(new   Commented($post, $request->user()->id));
```

```
    if (!$request->user()->valid) {
        $request->session()->flash('warning', __('Thanks for your comment. It will appear when an administrator has validated it.<br>Once you are validated your other comments immediately appear.'));
    }

    if($request->ajax()) {
        return response()->json();
    }

    return back();
}
```

La ligne qui crée la notification est celle-ci :

```
$post->user->notify(new Commented($post, $request->user()->id));
```

Exactement comme on l'a vu pour les email ci-dessus.

On trouve cet enregistrement dans la table des notifications :

id	type	notifiable_id	notifiable_type	data	read_at
e7a30d70-cde9-4afe-a1b9-3022fd31a6d5	App\Notifications\Commented	1	App\Models\User	{"title":"Post 2","slug":"post-2","user_id":1}	*NULL*

La clé étrangère qui permet de connaître l'utilisateur est notifiable_id.

On a le type de notification pour les distinguer si on en a plusieurs, ce qui n'est pas le cas de l'application d'exemple. La colonne **read_at** est à **NULL** parce que la notification n'a pas encore été lue. Dans la colonne **data** on retrouve sous forme de **JSON** les informations transmises (title, slug et user_id).

Les routes et le contrôleur

On a deux routes :

```
admin/notifications/{notification}  | notifications.update  | App\Http\Controllers\Back\NotificationController@update
admin/notifications/{user}          | notifications.index   | App\Http\Controllers\Back\NotificationController@index
```

La première est pour l'affichage des notifications.

La seconde est pour marquer une notification comme lue.

Voici le contrôleur (**App\Http\Controllers\Back\NotificationController**) :

```
<?php

namespace App\Http\Controllers\Back;

use App\ {
    Models\User,
    Http\Controllers\Controller
};
use Illuminate\ {
    Http\Request,
    Notifications\DatabaseNotification
};

class NotificationController extends Controller
{
    /**
     * Display a listing of the resource.
     *
     * @param  \App\Models\User  $user
     * @return \Illuminate\Http\Response
     */
    public function index(User $user)
    {
        return view('back.notifications.index', compact('user'));
    }

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

```
        $notification->markAsRead();

        if($request->user()->unreadNotifications->isEmpty()) {
            return redirect()->route('posts.index');
        }

        return back();
    }
}
```

La première méthode (**index**) ne pose pas de problème puisqu'on se contente d'utiliser une vue en transmettant une instance du modèle pour l'utilisateur connecté.

La seconde méthode (**update**) est destinée à marquer une notification comme lue, ce qui se fait avec la méthode **markAsRead**.

Remarquez que pour les deux méthodes on utilise la liaison automatique entre le paramètre de la route et un modèle. De cette manière on a directement une instance du modèle dans la méthode.

La vue index

Voici la vue pour afficher les notifications (**resources/views/back/notifications/index.blade.php**) :

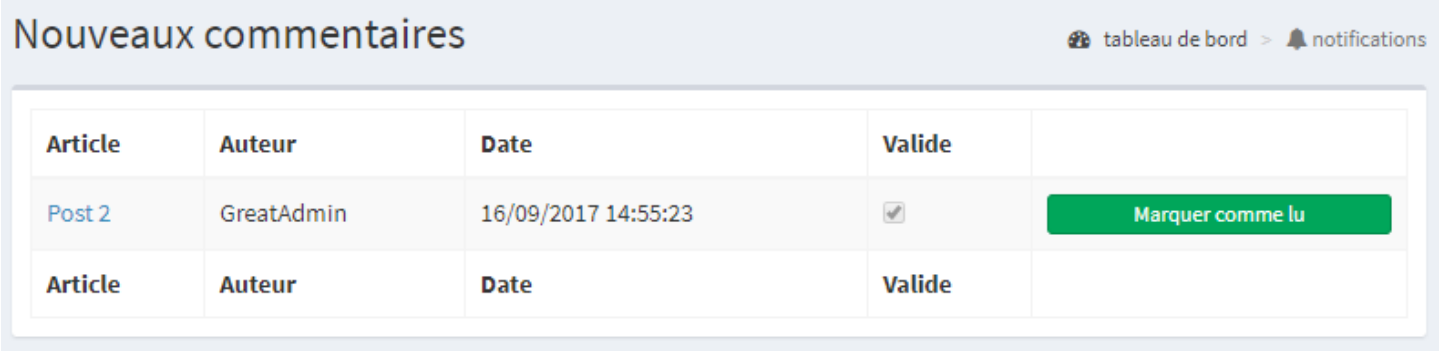
```
@extends('back.layout')

@section('main')

    <div class="row">
        <div class="col-md-12">
            <div class="box">
                <div class="box-body table-responsive">
                      <table class="table table-striped table-
bordered">
                        <thead>
                        <tr>
                            <th>@lang('Post')</th>
                            <th>@lang('Author')</th>
                            <th>@lang('Date')</th>
                            <th>@lang('Valid')</th>
                            <th></th>
```

```
                        </tr>
                        </thead>
                        <tfoot>
                        <tr>
                            <th>@lang('Post')</th>
                            <th>@lang('Author')</th>
                            <th>@lang('Date')</th>
                            <th>@lang('Valid')</th>
                            <th></th>
                        </tr>
                        </tfoot>
                        <tbody>
                             @foreach ($user->unreadNotifications
as $notification)
                                <tr>
                                                @php $user =
user($notification->data['user_id']) @endphp
                                                <td><a href="{{
route('posts.display',  [$notification->data['slug']])  }}">{{
$notification->data['title'] }}</a></td>
                                    <td>{{ $user->name }}</td>
                                                    <td>{{
$notification->created_at->formatLocalized('%c') }}</td>
                                     <td><input type="checkbox"
name="valid" {{ $user->valid ? 'checked' : '' }} disabled></td>
                                    <td>
                                                <form action="{{
route('notifications.update',   [$notification->id])   }}"
method="POST">
                                                    {{ csrf_field() }}
                                                    {{ method_field('PUT')
}}
                                                 <input type="submit"
class="btn btn-success btn-xs btn-block" value="@lang('Mark as
read')">
                                            </form>
                                    </td>
                                </tr>
                            @endforeach
                        </tbody>
                    </table>
                </div>
            </div>
```

```
            <!-- /.box -->
        </div>
        <!-- /.col -->
    </div>
    <!-- /.row -->

@endsection
```

On voit qu'on peut récupérer toutes les notifications non lues de l'utilisateur avec **$user->unreadNotifications**.

Voici l'aspect de la vue :



Si on clique sur « Marquer comme lue » on appelle la méthode **update** du contrôleur vue ci-dessus et la notification est marquée comme lue (mise à jour de la colonne **read_at**) et elle disparaît de l'affichage.

Il y aurait encore beaucoup à dire sur les notifications au-delà des deux exemples de ce chapitre, reportez-vous à la documentation officielle pour en savoir plus.

En résumé

- Laravel comporte un système complet et performant de notifications.
- On peut envoyer des emails avec les notifications, leur mise en forme est facilitée par la présence de puissantes méthodes et d'un template qu'on peut personnaliser.
- On peut stocker les notifications en base de données pour une utilisation ultérieure.