TP 6 : accès à une base de données avec *Laravel*

Dans ce TP, nous allons utiliser *Laravel* pour interagir avec une base de données. Pour cela, nous allons créer des *migrations* et des *modèles*, ainsi que des *contrôleurs*. Vous devez réutiliser l'installation de *Laravel* que vous avez faite dans le TP précédent.

I. Migrations, modèles et peuplement d'une base de données

1) Configuration de *Laravel*

Dans un premier temps, il est nécessaire de spécifier la configuration de la base de données à l'application *Laravel*.

1. Si ce n'est pas déjà fait, créez une base de données.
2. Éditez le fichier `.env` situé à la racine de votre projet *Laravel* et cherchez les lignes ci-dessous. Modifiez-les en fonction de la configuration de votre SGBD et de votre base de données.

```
DB_CONNECTION=mysql
DB_HOST=localhost
DB_PORT=3306
DB_DATABASE=???
DB_USERNAME=???
DB_PASSWORD=???
```

Laravel est maintenant prêt pour interagir avec votre base de données

2) Création d'un modèle et d'une migration

Le but d'un modèle est de pouvoir interagir de manière transparente avec la base de données. *Laravel* exploite la notion de persistance des objets : le développeur manipule des objets qui sont automatiquement récupérés ou créés depuis ou dans la base de données. La création d'un modèle passe par l'outil *artisan*.

1. Créez une migration et un modèle pour les actualités avec la commande suivante :

```
php artisan make:model Actualite --migration
```

Un fichier a été créé dans le répertoire `app/models` nommé `Actualite.php`, correspondant au modèle, de même que le fichier `2020_XX_XX_XXXX_create_actualites_table.php`, correspondant à la migration, situé dans le répertoire `database/migrations`. Dans le fichier migration, il y a deux méthodes : la méthode `up` qui est utilisée lors de création/mise-à-jour de la table et la méthode `down` qui est utilisée lors de l'annulation de la migration (comme vu en cours, une migration ne sert pas uniquement à la création de tables, mais peut également les mettre à jour).

2. Dans la méthode `up`, ajoutez le code suivant :

```
Schema::create('actualites', function (Blueprint $table) {
      $table->id('id');
      $table->string('titre', 100);
      $table->text('message');
      $table->datetime('date');
   });
```

3. Dans la méthode `down`, ajoutez ou vérifiez que le code suivant est présent (pour supprimer la table) :

```
Schema::dropIfExists('actualites');
```

4. Testez la migration en tapant la commande suivante :

```
php artisan migrate
```

5. Vérifiez dans votre base de données (via *phpmyadmin*) que la table *actualites* a été créée.

La commande suivante permet de revenir en arrière depuis la dernière migration :

```
php artisan migrate:rollback
```

6. Testez cette commande et vérifiez ses effets dans votre base de données. N'oubliez pas de refaire la migration ensuite pour recréer la table.

3) Peuplement de la base de données

Avec *artisan, Laravel* peut peupler la base de données pour insérer des informations pour le fonctionnement minimal de l'application ou pour générer des données de test.

1. Créez la classe `ActualiteTableSeeder` pour peupler la table *actualites* avec la commande suivante :

```
php artisan make:seed ActualiteTableSeeder
```

Le fichier `ActualiteTableSeeder.php` est créé dans le répertoire `database/seeds`. Le code pour le peuplement doit être placé dans la méthode `run`.

2. Ajoutez le code suivant dans la méthode `run` pour créer une actualité (`truncate` permet de vider la table) :

```
DB::table('actualites')->truncate();
\App\Models\Actualite::create([
      'titre' => 'Actualité 1',
      'message' => "Ceci est l'actualité numéro 1. C'est cool !!!",
      'date' => '2020-10-09'
  ]);
```

Comme vu en cours, pour éviter d'utiliser le chemin absolu de la classe `Actualite`, vous pouvez spécifier avant votre classe l'instruction `use App\Models\Actualite;`.

Pour que cette méthode soit appelée au moment du peuplement, il faut ensuite modifier le fichier `DatabaseSeeder.php` qui est appelé par défaut lors du peuplement de la base.

3. Ajoutez la ligne suivante (ou modifiez celle existante) dans la méthode `run` de la classe `DatabaseSeed` :

```
$this->call(Database\Seeders\ActualiteTableSeeder::class);
```

4. Exécutez le peuplement de la base avec la commande suivante :

```
php artisan db:seed
```

5. Vérifiez dans la table *actualites* qu'un enregistrement a bien été créé.

4) Utilisation du *faker*

Lorsque l'on a besoin de générer une grande quantité de données pour réaliser des tests, il est possible d'utiliser le *faker* qui permet de générer des données aléatoires. Pour cela, il faut simplement créer une instance via la classe `Factory`. Pour connaître les méthodes et les possibilités de cette classe, consultez la documentation :

https://github.com/fzaninotto/Faker

1. Ajoutez les instructions suivantes dans le *seeder* (pour le moment, nous utilisons uniquement la génération aléatoire de la date) :

```
$faker = \Faker\Factory::create();
Actualite::create([
                'titre' => 'Actualité 1',
                'message' => "Ceci est l'actualité numéro 1. C'est cool !!!",
                'date' => $faker->date('Y-m-d')
            ]);
```

2. Lancez le peuplement et observez la date générée dans votre base de données.
3. Modifiez votre classe pour générer 20 actualités avec un titre et un message aléatoires, des dates aléatoires de moins d'un an.

II. Développement d'un contrôleur

1) Création d'un contrôleur

Pour le moment, nous avons une base de données qui contient une table, mais nous n'avons toujours pas la possibilité d'ajouter ou supprimer des éléments dans celle-ci. Il est possible de réaliser ces opérations manuellement. Ici, nous allons utiliser un contrôleur généré automatiquement qui nous permettra de réaliser les actions de base CRUD (*Create*, *Read*, *Update* et *Delete*).

1. Créez un contrôleur via *artisan* avec l'instruction suivante :

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

Le fichier `ActualitesController` a été créé dans le répertoire `app/Http/Controllers`. Il contient les méthodes classiques CRUD et quelques autres. Pour qu'il soit accessible, n'oubliez pas d'ajouter la route suivante :

```
Route::resource('actualites', 'ActualitesController');
```

2. Vérifiez que la page `localhost/actualites/` fonctionne (une page vide est affichée).

2) Ajout et affichage

Pour manipuler ou afficher les actualités, nous allons avoir besoin de plusieurs vues.

1. Créez un répertoire `actualites` dans le répertoire des vues.

Pour ajouter une nouvelle actualité, nous avons besoin de créer une vue qui va contenir un formulaire, permettant de saisir les informations. Nous considérons le *template* (fichier `template.blade.php`) et la vue (fichier `create.blade.php`) donnés en annexes.

2. Vérifiez le bon fonctionnement de l'URL http://localhost/actualites/create. Le formulaire doit s'afficher normalement.
3. Que se passe-t-il lorsque vous cliquez sur le bouton *Ajouter* ?

Lavarel gère automatiquement la sécurité lors de la récupération des données du formulaire. Par défaut, vous obtiendrez une erreur 419 pour « Page expirée ». Il faut ajouter un champ spécifique dans chaque formulaire pour qu'il soit considéré comme valide par *Laravel*.

4. Ajoutez l'annotation *blade* `@csrf` après la balise `form` du formulaire. Vérifiez que l'erreur n'est plus affichée.

Lorsque le formulaire est validé, la méthode `store` du contrôleur est appelée automatiquement.

5. Ajoutez le code suivant pour récupérer l'actualité et la sauvegarder dans la base :

```
$actualite = new Actualite();
$actualite->titre = $request->titre;
$actualite->message = $request->message;
$actualite->date = $request->date;
```

```
$actualite->save();
return redirect()->route('actualites.show', ['actualite' => $actualite]);
```

6. Dans la méthode `show`, ajoutez le code suivant :

```
return view('actualites.show', ['actualite' => Actualite::findOrFail($id)]);
```

7. Testez maintenant l'ajout et l'affichage d'une actualité.

Pour récupérer un modèle depuis le formulaire, il est possible de récupérer automatiquement tous les champs. Pour cela, il faut modifier la classe `Actualite.php` et spécifier les champs qui peuvent être remplis.

8. Ajoutez le code suivant dans la classe `Actualite.php` :

```
protected $fillable = [
        'titre',
        'message',
        'date'
    ];
```

9. Modifiez maintenant le code de la méthode `store` du contrôleur :

```
$actualite = Actualite::create($request->input());
$actualite->save();
return redirect()->route('actualites.show', ['actualite' => $actualite]);
```

10. Vérifiez que le fonctionnement est identique.

3) Lister les actualités

La méthode `index` est appelée par défaut lorsqu'aucune action n'est spécifiée. Nous allons l'utiliser pour afficher la liste des actualités.

1. Ajoutez le code suivant dans la méthode `index` :

```
$actualites = Actualite::all();
return view('actualites.list', ['actualites' => $actualites]);
```

2. Ajoutez la vue `list.blade.php` présentée en annexes.
3. Vérifiez que la liste des actualités s'affiche correctement.

Il faut noter que vous pouvez choisir le nombre d'actualités affichées, l'ordre, *etc.*

4. Remplacez la première instruction par les instructions suivantes et observez le résultat :

```
$actualites = Actualite::orderBy('date', 'desc')->take(10)->get();
```

Pour plus de fonctionnalités, consultez la documentation d'*Eloquent* :

https://laravel.com/docs/8.x/eloquent

Dans la vue qui représente la liste des actualités, des liens ont été ajoutés pour éditer et supprimer les actualités.

4) Editer des actualités

Pour éditer une actualité, il suffit d'utiliser le lien créé via la fonction `route` (`id` représente l'identifiant de l'actualité à éditer) :

```
{{route('actualites.edit', id)}}
```

La méthode `edit` du contrôleur possède le code suivant :

```
return view('actualites.edit', ['actualite' => Actualite::findOrFail($id)]);
```

Pour finir, nous avons besoin d'une vue qui est représentée en annexes.

1. Modifiez votre contrôleur et ajoutez la vue `edit.blade.php`.
2. Vérifiez que l'édition est fonctionnelle.

5) Supprimer des actualités

Pour supprimer une actualité, il faut utiliser un formulaire car il faut passer la méthode `DELETE` au contrôleur à l'aide de l'annotation `@method('DELETE')`. Voici un formulaire basique :

```
<form action="{{route('actualites.destroy', $actualite->id)}}" method="POST">
    @method('DELETE')
    @csrf
    <button type="submit" class="btn btn-sm btn-danger">Supprimer</button>
</form>
```

Le contenu de la méthode `destroy` est le suivant :

```
$actualite = Actualite::findOrFail($id);
$actualite->delete();
return redirect()->route('actualites.index');
```

1. Modifiez votre contrôleur et vérifiez la suppression.
2. Comment pourrions-nous demander une confirmation de suppression avant la suppression définitive ?

III. Gestion des utilisateurs

- Pour cette section, il est conseillé d'utiliser la version de PHP 7.4 minimum. Au moment de l'écriture de ce support, la version d'uWamp ne supporte pas les versions supérieures à 7.2.7.
- Il est nécessaire d'installer *node.js* afin de pouvoir utiliser le gestionnaire `npm`.

1) Installation des routes

Par défaut, vous pouvez remarquer qu'une table `users` est créée automatiquement. Elle correspond aux utilisateurs du site. Par contre, il n'y a aucune route pour gérer les connexions/déconnexions, *etc*. Dans un premier temps, il faut installer le package *jetstream* (d'autres sont disponibles dans *Laravel*) :

```
composer require laravel/jetstream --dev
```

Il faut ensuite créer les vues via les commandes suivantes (d'autres sont également possible) :

```
php artisan jetstream:install livewire
```

L'installation ne peut pas fonctionner avec la version PHP proposée dans *uWamp* (7.2.7). Vous pouvez remplacer le package `jetstream` par le package `ui`. Dans ce cas, la première commande devient :

```
composer require laravel/jetstream --dev
```

Et la commande pour créer les vues :

```
php artisan ui vue --auth
```

Comme indiqué dans l'invite de commandes, il faut ensuite exécuter ces commandes pour générer le *Javascript* et le CSS (`npm` fait partie de *node.js* https://nodejs.org/fr/) :

```
npm install
npm run dev
```

Vous pouvez voir que dans le fichier `web.php`, des lignes ont été ajoutées. Vous pouvez consulter les routes créées à l'aide de la commande suivante (le -c signifie l'affichage compact) :

```
php artisan route:list -c
```

Il est maintenant possible de créer des utilisateurs, de se connecter, etc.

2) Gestion de l'authentification

1. Testez la route `/register` et créez un compte. Si tout se passe bien, vous êtes maintenant connecté. Vous êtes redirigé vers la vue `dashboard` (tout est modifiable).
2. Allez maintenant sur la liste des actualités.

Nous souhaiterions empêcher l'édition et la suppression des actualités pour les utilisateurs non connectés.

3. Modifiez la méthode `index` du contrôleur des actualités :

```
return view('actualites.list',['actualites' => $actualites, 'logged' => Auth::check()]);
```

La méthode `Auth::check()` retourne *vrai* si l'utilisateur est connecté. Cette variable peut maintenant être utilisée dans la vue `actualites/list.blade.php`.

4. Ajoutez les instructions *blade* pour protéger les boutons *Modifier* et *Supprimer* :

```
@if($logged)
…
@endif
```

5. Déconnectez-vous et vérifiez que les boutons ne sont plus accessibles.
6. Si vous êtes déconnecté, vous pouvez encore accéder à la page de création. Faites le test.
7. Dans la méthode `create`, ajoutez l'instruction suivante et vérifiez le comportement de votre application :

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

8. Protégez toutes les pages et les boutons de l'ensemble des vues avec les techniques précédentes.

Laravel propose des solutions complètes pour gérer l'authentification et les droits associés aux utilisateurs (les autorisations). Consultez la documentation *Laravel*.

Annexes

Le *template* nommé `template.blade.php` (répertoire `resources/views/`) utilisé dans l'exercice 2 :

```
<!DOCTYPE html>
<html lang="fr">
  <head>
    <title>@yield('titre')</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link                                                    rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css">
  </head>
  <body>
    <div class="container">
      <div class="card">
        <div class="card-header bg-primary text-white text-center">@yield('titre')</div>
        <div class="card-body bg-light">@yield('contenu')</div>
      </div>
    </div>
  </body>
 </html>
```

La vue `create.blade.php` (répertoire `ressources/views/actualites/`) utilisée dans l'exercice 2 :

```
@extends('template')
@section('titre') Ajouter une actualité @endsection
@section('contenu')
<form action="{{url('actualites')}}" method="post">
  <div class="form-group row">
    <label for="titre" class="col-sm-2 col-form-label">Titre</label>
    <div class="col-sm-10">
      <input     type="text"     class="form-control"     name="titre"     id="titre"
placeholder="Saisir le titre de l'actualité"/>
    </div>
  </div>
  <div class="form-group row">
    <label for="message" class="col-sm-2 col-form-label">Message</label>
    <div class="col-sm-10">
      <textarea    class="form-control"    id="message"    name="message"    rows="3"
placeholder="Saisir le message de l'actualité"></textarea>
    </div>
  </div>
  <div class="form-group row">
    <label for="date" class="col-sm-2 col-form-label">Date</label>
    <div class="col-sm-10">
      <input   type="datetime-local"   class="form-control"   name="date"   id="date"
placeholder="Saisir la date de l'actualité"/>
    </div>
  </div>
  <div class="form-group row">
    <button class="btn btn-primary mb-1 mr-1" type="submit">Ajouter</button>
    <a href="{{url('actualites')}}" class="btn btn-danger mb-1">Annuler</a>
  </div>
</form>
@endsection
```

La vue `list.blade.php` (répertoire `ressources/views/actualites/`) utilisée dans l'exercice 2 :

```
@extends('template')
@section('titre') Liste des actualités @endsection
@section('contenu')
  <div class="d-flex justify-content-center">
    <a   href="{{route('actualites.create')}}"   class="btn   btn-sm   btn-primary   mb-
1">Création d'une nouvelle actualité</a>
  </div>
  <ul class="list-group">
<?php
foreach($actualites as $actualite) {
?>
    <li class="list-group-item d-flex justify-content-between align-items-center">
      <div class="col-8">
        <span     class="badge     badge-primary     badge-pill">{{strftime('%d/%m/%Y',
strtotime($actualite->date))}}</span>
        <strong>{{$actualite->titre}}</strong>
        {{$actualite->message}}
      </div>
      <div>
        <a  href="{{route('actualites.show',$actualite->id)}}"  class="btn  btn-sm  btn-
primary mb-1">Consulter</a>
        <a  href="{{route('actualites.edit',$actualite->id)}}"  class="btn  btn-sm  btn-
primary mb-1">Editer</a>
          @method('DELETE')
          @csrf
        <form action="{{route('actualites.destroy', $actualite->id)}}" method="POST">
          <button type="submit" class="btn btn-sm btn-danger mb-1">Supprimer</button>
        </form>
      </div>
    </li>
<?php
}
?>
  </ul>
@endsection
```

La vue `edit.blade.php` (répertoire `ressources/views/actualites/`) utilisée dans l'exercice 2 :

```
@extends('template')
@section('titre') Modifier une actualité @endsection
@section('contenu')
<form action="{{url('actualites')}}" method="post">
  @csrf
  <div class="form-group row">
    <label for="titre" class="col-sm-2 col-form-label">Titre</label>
    <div class="col-sm-10">
      <input     type="text"     class="form-control"     name="titre"     id="titre"
placeholder="Saisir le titre de l'actualité" value="{{$actualite->titre}}"/>
    </div>
  </div>
  <div class="form-group row">
    <label for="message" class="col-sm-2 col-form-label">Message</label>
    <div class="col-sm-10">
      <textarea    class="form-control"    id="message"    name="message"    rows="3"
placeholder="Saisir le message de l'actualité">{{$actualite->message}}</textarea>
    </div>
  </div>
  <div class="form-group row">
    <label for="date" class="col-sm-2 col-form-label">Date</label>
    <div class="col-sm-10">
      <input   type="datetime-local"   class="form-control"   name="date"   id="date"
placeholder="Saisir    la    date    de    l'actualité"    value="{{date('Y-m-d\TH:i',
strtotime($actualite->date))}}"/>
    </div>
  </div>
  <div class="form-group row">
    <button class="btn btn-primary mb-1 mr-1" type="submit">Modifier</button>
    <a  href="{{route('actualites.show',$actualite->id)}}"  class="btn  btn-danger  mb-
1">Annuler</a>
  </div>
</form>
</div>
@endsection
```