# Projet Réservations – Laravel 7 Roadmap

## Chapitre 1 – Comment démarrer

### Exigences

Avant d’installer Laravel 7, il faudra veiller à respecter les exigences suivantes :

* PHP >= 7.2.5
* Extensions PHP : BCMath, Ctype, Fileinfo, JSON, Mbstring, OpenSSL, PDO, Tokenizer, XML.

### Installation et configuration

1. Installer Laravel
   1. Mettre à jour le gestionnaire de dépendances *composer*

composer self-update

* 1. Télécharger le framework au moyen du gestionnaire de dépendances *composer*

composer create-project --prefer-dist laravel/laravel reservations

<https://laravel.com/docs/7.x>

*Notez qu’il est aussi possible d’utiliser l’installeur de Laravel pour créer des projets. Il est aussi vivement conseillé d’installer le système d’authentification dès le début du projet.*

[*https://laravel.com/docs/7.x/authentication*](https://laravel.com/docs/7.x/authentication)

*laravel new reservations --auth*

* 1. Tester l’installation

Lancer le serveur PHP interne et accéder à l’URL [**http://localhost:8000/**](http://localhost:8000/)

Entrer dans le dossier du projet et taper la commande de lancement du serveur

cd reservations

php artisan serve

Pour arrêter le server, il faut taper CTRL+C.

Vous pouvez aussi créer un Virtual Host dans le fichier de configuration du serveur Apache *httpd.conf* en ajoutant le code suivant :

<VirtualHost \*:80>

DocumentRoot "C:/UwAmp/www/**reservations**/public/"

**ServerName "laravel.localhost"**

<Directory "C:/UwAmp/www/**reservations**/public/">

AllowOverride All

Options FollowSymLinks Includes Indexes MultiViews

</Directory>

</VirtualHost>

Vous pouvez adapter le *ServerName* en changeant le préfixe de *localhost*. Dès lors, vous pouvez accéder au site dans le navigateur en tapant l’URL [**http://laravel.localhost/**](http://laravel.localhost/)

N.B. : Vous devez adapter le chemin spécifié dans *DocumentRoot* et *Directory* en fonction du paquetage AMP que vous utilisez (UwAMP, *XAMPP,* *WampServer*,…).

1. Initialiser le système de versioning du code

Arrêter le serveur et taper la commande d’initialisation du versioning

git init

Ajouter tous les fichiers du dossier au *stage*, puis faites un premier *commit*.

git add .

git commit -m "After framework installation"

1. Configurer le framework
   1. Définir le nom de l’application

Ouvrir le fichier .env et modifier l’entrée APP\_NAME

APP\_NAME="Projet Réservations"

APP\_ENV=local

APP\_KEY=base64:fN8i+OQ4pnLdG4qxDRG0aknTL6fmkwBT2Mc39+A66Og=

APP\_DEBUG=true

APP\_URL=http://localhost

Vérifiez que l’entrée APP\_KEY possède bien une valeur. Dans le cas contraire, vous pouvez générer une clé dans la console au moyen de la commande suivante :

php artisan key:generate

* 1. Définir le chemin d’accès à la base de données

Ouvrir le fichier .env et modifier les entrées DB\_DATABASE, DB\_USERNAME et DB\_PASSWORD

DB\_CONNECTION=mysql

DB\_HOST=127.0.0.1

DB\_PORT=3306

DB\_DATABASE=reservations

DB\_USERNAME=root

DB\_PASSWORD=root

N.B. : Par défaut, le driver configuré est mysql. Si nécessaire vous pouvez changer de driver en modifiant l’entrée DB\_CONNECTION. Pour plus de configurations, voir le fichier *config/database.php*

'connections' => [

//…

'mysql' => [

'driver' => 'mysql',

'host' => env('DB\_HOST', '127.0.0.1'),

'port' => env('DB\_PORT', '3306'),

'database' => env('DB\_DATABASE', 'reservations'),

'username' => env('DB\_USERNAME', 'root'),

'password' => env('DB\_PASSWORD', 'root'),

//…

],

//…

],

1. Versionner le projet

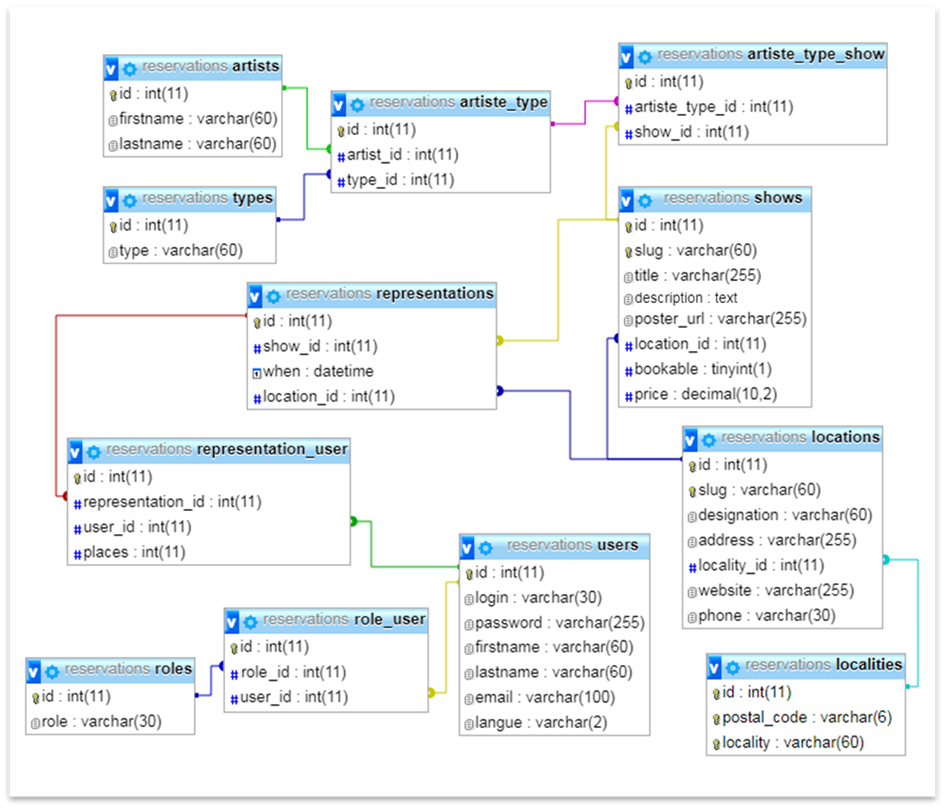
git status

git add .

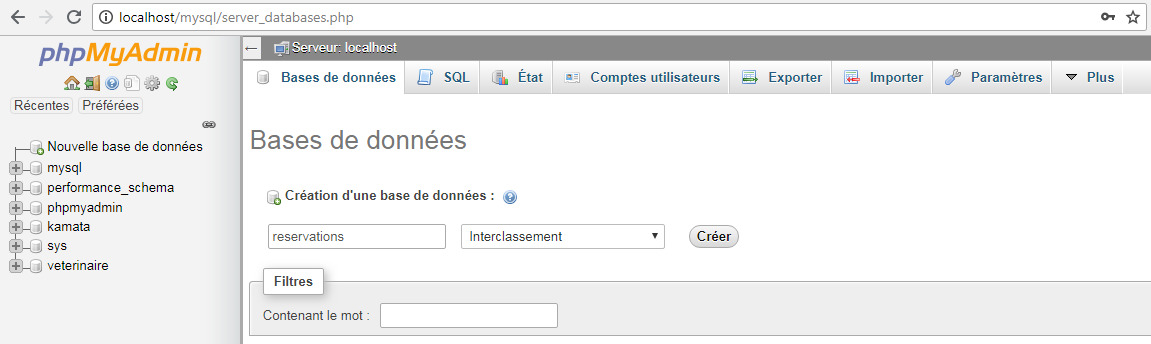
git commit -m "After framework configuration"

## Chapitre 2 – Modéliser les données

### Base de données : tables et modèles



1. Créer la base de données
   1. via *PhpMyAdmin*



* 1. ou la ligne de commande

C:\Uwamp\bin\database\mysql-5.7.11\bin> **mysql –u root –p**

Enter password: \*\*\*\*

mysql> **CREATE DATABASE reservations;**

Query OK, 1 row affected (0.02 sec)

mysql> **\q**

Bye

<https://laravel.com/docs/5.6/eloquent>

1. Créer un modèle

Retourner dans le dossier du projet et taper la commande du générateur *make*

**php artisan make:model Artist**

Le fichier créé: *Artist.php* (dans le dossier *app*).

Dans le modèle, indiquez les champs qui pourront être modifiés avec la propriété *$fillable* ou inversement ceux qui ne devraient pas l’être avec la propriété *$guarded*.

Ensuite, précisez le nom de la table avec la propriété *$table*.

Par défaut, Laravel crée des champs created\_at et updated\_at dans la table. Pour empêcher cela, vous pouvez assigner la valeur *false* à la propriété *$timestamps*.

*app\Artist.php*

<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Artist extends Model

{

/\*\*

\* The attributes that are mass assignable.

\*

\* @var array

\*/

**protected $fillable = ['firstname', 'lastname'];**

/\*\*

\* The table associated with the model.

\*

\* @var string

\*/

**protected $table = 'artists';**

/\*\*

\* Indicates if the model should be timestamped.

\*

\* @var bool

\*/

**public $timestamps = false;**

}

1. Créer les migrations

Une migration permet de définir les modifications entre le modèle et la base de données. Puisque nous avons créé le modèle *Artist* et spécifié les champs *firstname* et *lastname*, la migration définira la méthode de création (et de suppression) de cette table dans la base de données.

php artisan make:migration create\_artists\_table

Modifier le fichier généré : ajouter les champs *firstname* et *lastname* limité à 60 caractères, supprimer la création des champs de *created\_at* et *updated\_at* (timestamps).

*database\migrations\YYYY\_MM\_DD\_HHMMSS\_create\_artists\_table.php*

<?php

use Illuminate\Support\Facades\Schema;

use Illuminate\Database\Schema\Blueprint;

use Illuminate\Database\Migrations\Migration;

class CreateArtistsTable extends Migration

{

/\*\*

\* Run the migrations.

\*

\* @return void

\*/

public function up()

{

Schema::create('artists', function (Blueprint $table) {

$table->increments('id');

**$table->string('firstname')->length(60);**

**$table->string('lastname')->length(60);**

});

}

/\*\*

\* Reverse the migrations.

\*

\* @return void

\*/

public function down()

{

Schema::dropIfExists('artists');

}

}

1. Exécuter les migrations

php artisan migrate

Dans la base de données, la table a été créée.

1. Insérer un jeu de données de test dans la table

Les *seeders* sont des classes qui permettent de définir des données à insérer dans les tables de la base de données. Bien souvent, il s’agit de données fictives pour tester l’application.

<https://laravel.com/docs/7.x/seeding>

* 1. Créer une classe de *seeders*

php artisan make:seeder ArtistsTableSeeder

Le fichier créé:

*database\seeds\ArtistsTableSeeder.php*

<?php

use Illuminate\Database\Seeder;

class ArtistsTableSeeder extends Seeder

{

//

}

* 1. Modifier le fichier de seeders

<?php

use Illuminate\Database\Seeder;

**use App\Artist;**

class ArtistsTableSeeder extends Seeder

{

/\*\*

\* Run the database seeds.

\*

\* @return void

\*/

public function run()

{

//Empty the table first

Artist::truncate();

//Define data

$artists = [

['firstname'=>'Daniel','lastname'=>'Marcelin'],

['firstname'=>'Philippe','lastname'=>'Laurent'],

['firstname'=>'Marius','lastname'=>'Von Mayenburg'],

['firstname'=>'Olivier','lastname'=>'Boudon'],

['firstname'=>'Anne Marie','lastname'=>'Loop'],

['firstname'=>'Pietro','lastname'=>'Varasso'],

['firstname'=>'Laurent','lastname'=>'Caron'],

['firstname'=>'Élena','lastname'=>'Perez'],

['firstname'=>'Guillaume','lastname'=>'Alexandre'],

['firstname'=>'Claude','lastname'=>'Semal'],

['firstname'=>'Laurence','lastname'=>'Warin'],

['firstname'=>'Pierre','lastname'=>'Wayburn'],

['firstname'=>'Gwendoline','lastname'=>'Gauthier'],

];

//Insert data in the table

foreach ($artists as $data) {

DB::table('artists')->insert([

'firstname' => $data['firstname'],

'lastname' => $data['lastname'],

]);

}

}

}

<https://laravel.com/docs/7.x/seeding#using-model-factories>

* 1. Exécuter les seeders
     1. Dans le fichier *DatabaseSeeder.php*, il faut d’abord ajouter la classe *ArtistsTableSeeder* à la liste des seeders à exécuter :

*database\seeds\DatabaseSeeder.php*

<?php

use Illuminate\Database\Seeder;

class DatabaseSeeder extends Seeder

{

/\*\*

\* Seed the application's database.

\*

\* @return void

\*/

public function run()

{

$this->call(**[**

ArtistsTableSeeder::class,

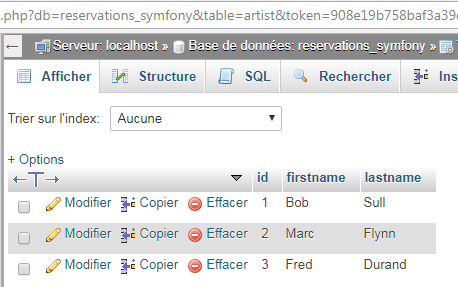
**]**);

}

}

* + 1. Ensuite, il faut regénérer le fichier autoloader de Composer :

composer dump-autoload

* + 1. Enfin, nous pouvons exécuter les seeders :

php artisan db:seed

1. Versionner le projet

git status

git add .

git commit -m "Modèle Artist +migration +seeders"

### Contrôleurs et templates

<https://laravel.com/docs/7.x/controllers>

1. Créer un contrôleur (et un template)

php artisan make:controller ArtistController --resource

Cette commande crée un contrôleur, et l’option --resource ajoute les méthodes (*index*, *create*, *store*, *show*, *edit*, *update* et *destroy*).

*App\Http\Controllers\ArtistController.php*

<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class ArtistController 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)

{

//

}

}

1. Modifier la méthode index de façon à récupérer la liste des artistes et envoyer la liste des artistes à un template spécifique.  
   Vous pouvez utiliser une syntaxe SQL avec la méthode DB::select() ou en encore passer par le modèle Artist de l’ORM Eloquent avec la méthode Artist::all().

<https://laravel.com/docs/7.x/queries>

<https://laravel.com/docs/7.x/eloquent#retrieving-models>

<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

**use App\Artist;**

class ArtistController extends Controller

{

/\*\*

\* Display a listing of the resource.

\*

\* @return \Illuminate\Http\Response

\*/

public function index()

{

**$artists = Artist::all();**

**return view('artist.index',[**

**'artists' => $artists,**

**'resource' => 'artistes',**

**]);**

}

//…

}

1. Modifier la méthode show de façon à récupérer l’artiste dont l’id est donnée en paramètre. Envoyer l’objet $artist trouvé à un template spécifique.  
   Vous pouvez utiliser une syntaxe SQL avec la méthode DB::select() ou en encore passer par le modèle Artist de l’ORM Eloquent avec la méthode Artist::find($id).

//…

/\*\*

\* Display the specified resource.

\*

\* @param int $id

\* @return \Illuminate\Http\Response

\*/

public function show($id)

{

**$artist = Artist::find($id);**

**return view('artist.show',[**

**'artist' => $artist,**

**]);**

}

//…

1. Créer un layout principal et deux templates
   1. Il faut d’abord créer un fichier qui reprendra l’ossature principale tel que repris dans la documentation du moteur de template Blade. N’oubliez pas de créer le dossier *layouts*.

<https://laravel.com/docs/7.x/blade>

*ressources/views/layouts/app.****blade****.php*

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title>Projet réservations - @yield('title')</title>

</head>

<body>

<div class="container">

@yield('content')

</div>

</body>

</html>

* 1. Ensuite, nous créons le template *index.****blade****.php* qui devra afficher la liste des artistes. Sans oublier de créer le dossier *artist*.

*ressources/views/artist/index.blade.php*

@extends('layouts.app')

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

@section('content')

<h1>Liste des {{ $resource }}</h1>

<table>

<thead>

<tr>

<th>Firstname</th>

<th>Lastname</th>

</tr>

</thead>

<tbody>

@foreach($artists as $artist)

<tr>

<td>{{ $artist->firstname }}</td>

<td>{{ $artist->lastname }}</td>

</tr>

@endforeach

</tbody>

</table>

@endsection

<https://laravel.com/docs/7.x/blade#loops>

* 1. Ensuite, nous créons le template *show.****blade****.php* qui devra afficher les données de l’artiste dont l’id est donné dans l’URL.

*ressources/views/artist/show.blade.php*

@extends('layouts.app')

@section('title', 'Fiche d\'un artiste')

@section('content')

<h1>{{ $artist->firstname }} {{ $artist->lastname }}</h1>

@endsection

1. Enfin, nous devons ajouter les routes afin de pouvoir accéder à ces pages à travers notre application. Dans Laravel, les routes sont définies dans le fichier *routes/web.php*.

*routes/web.php*

<?php

/\*

|--------------------------------------------------------------------------

| Web Routes

|--------------------------------------------------------------------------

|

| Here is where you can register web routes for your application. These

| routes are loaded by the RouteServiceProvider within a group which

| contains the "web" middleware group. Now create something great!

|

\*/

Route::get('/', function () {

return view('welcome');

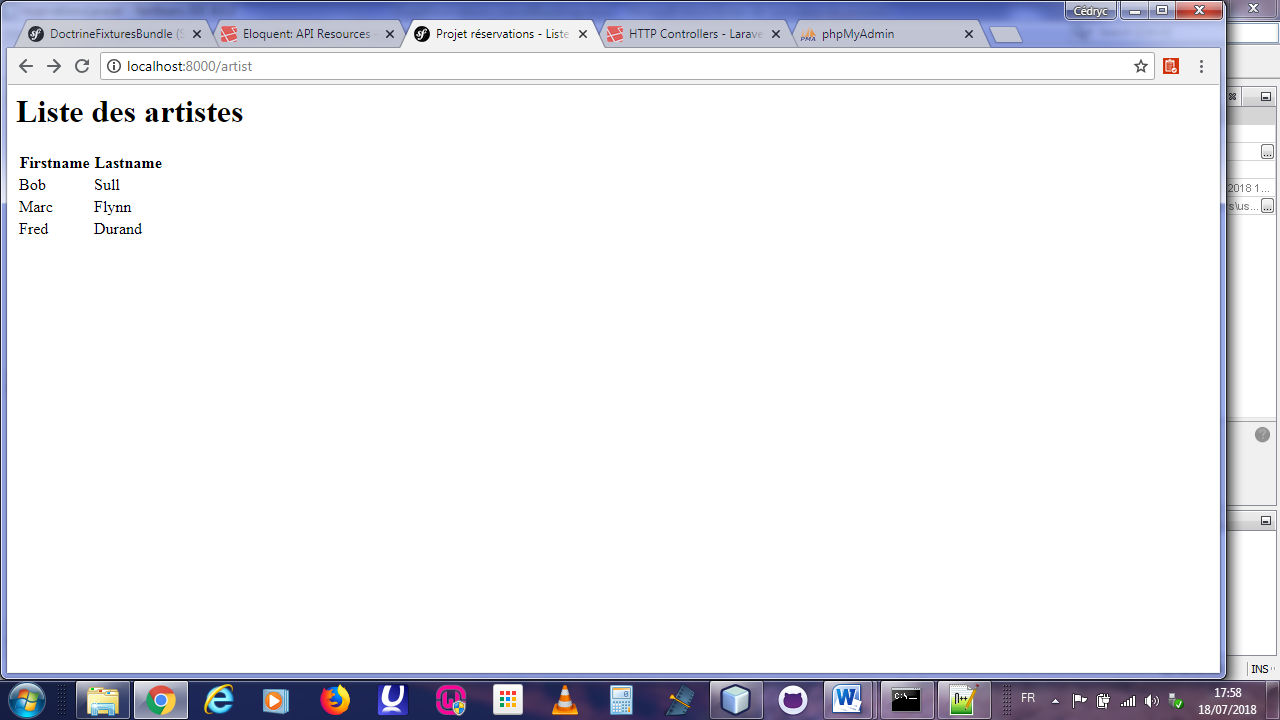
});

Route::get('artist', 'ArtistController@index');

Route::get('artist/{id}', 'ArtistController@show');

* Lancer le serveur interne (ou accéder directement à l’URL de votre Virtual Host)

php artisan serve

* Accéder à l’URL <http://localhost:8000/artist> (ou <http://laravel.localhost/artist>)
* Accéder à l’URL <http://localhost:8000/artist/1> (ou <http://laravel.localhost/artist/1>)

1. Versionner le projet

git status

git add .

git commit -m "Controller Artist +templates +routes"

### A vous de jouer !

1. **Répétez ces opérations pour les modèles *Type*, *Locality* et *Role* (correspondant aux tables *types*, *localities* et *roles*).**Valeurs pour la table *types* : auteur, scénographe, comédien...  
   Valeurs pour la table *roles* : admin, member, affiliate…
   1. Créer les modèles
   2. Créer les migrations
   3. Exécuter les migrations
   4. Générer la classe de seeders
   5. Définir les données test (seeders)
   6. Charger les données test (seeders) dans la table
   7. Créer les contrôleurs, les templates (index et show) et définissez les routes
2. Versionner le projet