

DOCUMENTATION PARKING :

Attribution des places de parking :

Ce projet a été réalisé en groupe de 3 personnes lors de ma 2ème année de BTS SIO durant lequel nous avons dû créer et programmer une application en Laravel.

Afin d'éviter le stationnement sauvage dans le labyrinthe qu'est le parking, il a été décidé d'attribuer à chaque membre qui le demandait une place de parking numérotée.

Le besoin attendu :

- Le front-office doit être sécurisé et n'accepter que les demandes du personnel des ligues. Les inscriptions au service de réservation de place doivent être validées (ou créées) par un administrateur.
- L'administrateur, seul utilisateur du back-office, doit pouvoir éditer la liste des places et gérer les inscriptions des utilisateurs.
- Lorsqu'un utilisateur en fait la demande, une place libre lui est attribuée aléatoirement et immédiatement par l'application, la réservation expire automatiquement au bout d'une durée par défaut déterminée par l'administrateur.
- Si une demande ne peut pas être satisfaite, l'utilisateur est placé en liste d'attente.
- L'utilisateur ne peut pas choisir la date à laquelle une place lui est attribuée, les réservations sont toujours immédiates. Un utilisateur ne peut pas faire une demande de réservation s'il est en file d'attente ou qu'il occupe une place.
- Un utilisateur ou l'administrateur peuvent fermer une réservation avant la date d'expiration prévue. Une fois celle-ci expirée, l'utilisateur doit refaire une demande s'il souhaite obtenir une place.

Caractéristiques à tenir compte lors du projet :

3.1 Sécurité

- Protection des accès par mot de passe.
- Contrôles de saisie des données côté serveur.
- Contrôles de saisie côté client.
- Protection contre les attaques par injection.

3.2 Gestion des mots de passe

- Fonction "mot de passe perdu ?".
- Hachage des mots de passe.

3.3 Espace utilisateur

- Vérification de l'identité par saisie d'un mot de passe.
- Possibilité de visualiser le numéro de place attribuée, ainsi que l'historique des places précédemment attribuées.
- Possibilité de faire une demande de réservation.
- Possibilité de connaître son rang sur la file d'attente.
- Modification du mot de passe.

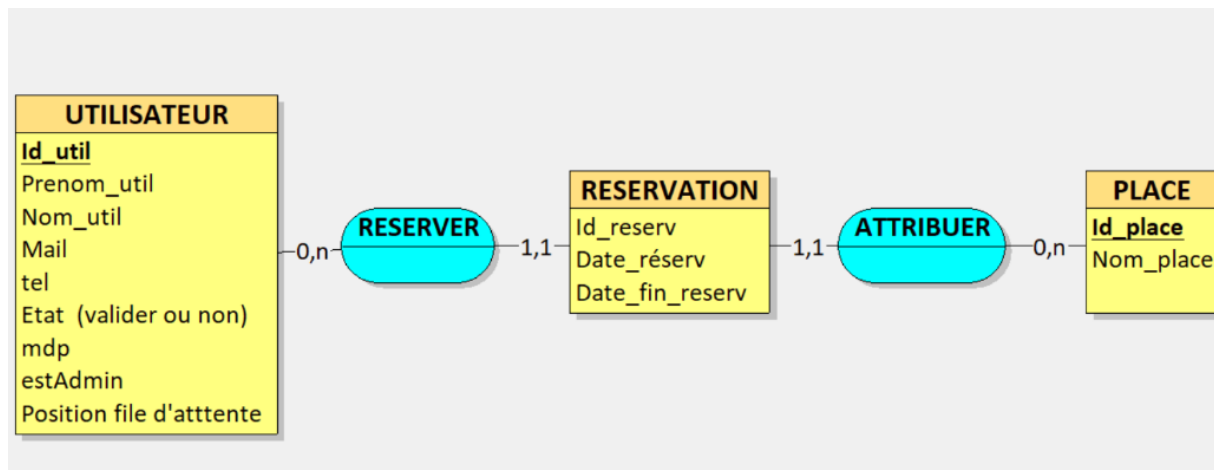
3.4 Espace administrateur

- Protection de l'accès par mot de passe.
- Édition de la liste des utilisateurs, réinitialisation des mots de passe.
- Édition de la liste des places
- Consultation de la liste d'attente.
- Consultation de l'historique d'attribution des places.
- Attribution manuelle des places.
- Édition de la file d'attente (modification de la position des personnes en attente).

3.5 Pages web

- Mise aux normes HTML5 et CSS des pages WEB

MCD Réalisé pour cette application :



Cette application a donc été programmée sur Laravel en PHP lors de différentes étapes :

La création des migrations spécifiques pour chaque table

Voici un exemple de migrations :

```
2014_10_12_000000_create_users_table.php X
database > migrations > 2014_10_12_000000_create_users_table.php > PHP Intelephense > class
1  <?php
2
3  use Illuminate\Database\Migrations\Migration;
4  use Illuminate\Database\Schema\Blueprint;
5  use Illuminate\Support\Facades\Schema;
6
7  return new class extends Migration
8  {
9      /**
10       * Run the migrations.
11       */
12     public function up(): void
13     {
14         Schema::create('users', function (Blueprint $table) {
15             $table->id();
16             $table->string('name');
17             $table->string('email')->unique();
18             $table->timestamp('email_verified_at')->nullable();
19             $table->string('password');
20             $table->boolean('est_valide')->default(false);
21             $table->boolean('est_admin')->default(false);
22             $table->integer('position')->nullable();
23             $table->rememberToken();
24             $table->timestamps();
25         });
26     }
27
28     /**
29      * Reverse the migrations.
30      */
31     public function down(): void
32     {
33         Schema::dropIfExists('utilisateur');
34     }
35 };
36
```

Dans une migration, il faut spécifier le type de données, si il peut être nul, si l'est unique, si il a une valeur par défaut etc.

Mise en place des Factories :

```
UserFactory.php X
database > factories > UserFactory.php
1  <?php
2
3  namespace Database\Factories;
4
5  use Illuminate\Database\Eloquent\Factories\Factory;
6  use Illuminate\Support\Str;
7
8  /**
9   * @extends \Illuminate\Database\Eloquent\Factories\Factory<\App\Models\User>
10  */
11  class UserFactory extends Factory
12  {
13      /**
14       * Define the model's default state.
15       *
16       * @return array<string, mixed>
17       */
18     public function definition(): array
19     {
20         return [
21             'name' => fake()->name(),
22             'email' => fake()->unique()->safeEmail(),
23             'email_verified_at' => now(),
24             'password' => '$2y$10$92IXunpkj08r0Q5byMi.Ye4oKoEa3Ro9llC/.og/at2.uheWG/igi', // password
25             'remember_token' => Str::random(10),
26         ];
27     }
28
29     /**
30      * Indicate that the model's email address should be unverified.
31      */
32     public function unverified(): static
33     {
34         return $this->state(fn (array $attributes) => [
35             'email_verified_at' => null,
36         ]);
37     }
38 }
39
```

Une factorie contient des valeurs par défaut à rentrer pour chaque table créée.

Ensuite il faut mettre en place les seeders :

```
UserSeeder.php X
database > seeders > UserSeeder.php > PHP Intelephense > UserSeeder > run
1  <?php
2
3  namespace Database\Seeders;
4
5  use Illuminate\Database\Seeder;
6  use Illuminate\Support\Facades\DB;
7  use Illuminate\Support\Facades\Hash;
8
9  1 reference | 0 implementations
10 class UserSeeder extends Seeder
11 {
12     /**
13      * Run the database seeds.
14      */
15     0 references | 0 overrides
16     public function run(): void
17     {
18         DB::table('users')->insert([
19             'name' => 'Admin',
20             'email' => 'admin@parking.com',
21             'password' => Hash::make('secret'),
22             'est_valide' => true,
23             'est_admin' => true,
24             'position' => null,
25         ]);
26     }
27 }
```

Le seeder crée un jeu de données, pour l'environnement de développement ou pour les tests.

Ils se basent sur la définition des factories pour créer ses objets et remplir la base de données.

Et ici ajout de l'admin dans le seeder.

```

resources > views > pages > place > create.blade.php > ...
1
2 @extends('layouts.app')
3
4 @section('content')
5 <div class="container">
6 <div class="row justify-content-center">
7 <div class="col-md-8">
8 <div class="card">
9 <div class="card-header">Ajouter</div>
10
11 <div class="card-body">
12 <form method="POST" action="{{ route('places.store') }}">
13 @csrf
14
15 <div class="row mb-3">
16 <label for="numero" class="col-md-4 col-form-label text-md-end">Numero</label>
17
18 <div class="col-md-6">
19 <input id="numero" type="text" class="form-control @error('numero') is-invalid @enderror" name="numero" value="{{ old('numero') }}" required
20
21 @error('numero')
22 <span class="invalid-feedback" role="alert">
23 <strong>{{ $message }}</strong>
24 </span>
25 @enderror
26 </div>
27 </div>
28
29 <div class="row mb-0">
30 <div class="col-md-6 offset-md-4">
31 <button type="submit" class="btn btn-primary">
32 Ajouter
33 </button>
34 </div>
35 </div>
36 </form>
37 </div>
38 </div>
39 </div>
40 </div>
41 </div>
42 @endsection

```

Initialisation et création des vues :

Les vues contiennent le code HTML du site, une vue c'est donc créer un fichier contenant du HTML qui sera retourné par une route.

Il faut ensuite mettre en place les routes :

```

use Illuminate\Support\Facades\Route;

/*
|-----
| Web Routes
|-----
|
| Here is where you can register web routes for your application. These
| routes are loaded by the RouteServiceProvider and all of them will
| be assigned to the "web" middleware group. Make something great!
|
*/

Auth::routes();
Route::post('/login', [App\Http\Controllers\Auth\LoginController::class, 'authenticate']);

Route::middleware(['auth'])->group(function () {
    Route::get('/', [App\Http\Controllers\HomeController::class, 'index'])->name('home');

    Route::resource('utilisateurs', App\Http\Controllers\UserController::class)->except(['edit']);

    Route::resource('reservations', App\Http\Controllers\ReservationController::class)->except(['edit', 'show', 'update']);
    Route::post('reservations/position/{utilisateur}', [App\Http\Controllers\ReservationController::class, 'changePosition'])->name('reservations.position');
    Route::post('reservations/force', [App\Http\Controllers\ReservationController::class, 'force'])->name('reservations.force');
    Route::get('reservations/browse', [App\Http\Controllers\ReservationController::class, 'browse'])->name('reservations.browse');

    Route::resource('places', App\Http\Controllers\PlaceController::class)->except(['edit']);
});

```

Le système de routes permet de faire correspondre un morceau de code aux différentes URL de l'application.

Initialisation et mise en place des controller :

```
<?php

namespace App\Http\Controllers\PlaceController;

namespace App\Http\Controllers;

use App\Models\Place;
use Illuminate\Http\Request;

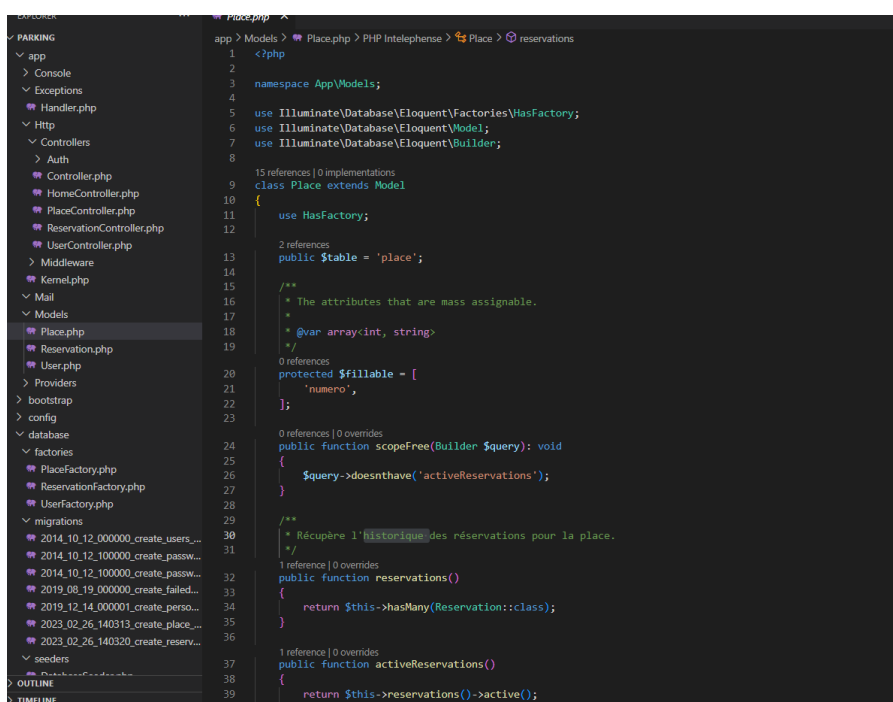
1 reference | 0 implementations
class PlaceController extends Controller
{
    /**
     * Display a listing of the resource.
     */
    0 references | 0 overrides
    public function index()
    {
        return view('pages.place.index', [
            'places' => Place::orderBy('numero', 'ASC')->get(),
        ]);
    }

    /**
     * Show the form for creating a new resource.
     */
    0 references | 0 overrides
    public function create()
    {
        return view('pages.place.create');
    }

    /**
     * Store a newly created resource in storage.
     */
    0 references | 0 overrides
    public function store(Request $request)
    {
        $request->validate([
            'numero' => ['required', 'string', 'max:255', 'unique:place'],
        ]);

        $place = Place::create([
            'numero' => $request->numero,
        ]);
    }
}
```

Un controller est une classe qui va contenir différentes méthodes. Chaque méthode correspondant généralement à une opération (URL) de votre application. Il est ensuite possible de faire appelle à ce controller et cette méthode (appelée "action") en utilisant les routes.



```
app > Models > Place.php > PHP Intelliphense > Place > reservations

1  <?php
2
3  namespace App\Models;
4
5  use Illuminate\Database\Eloquent\Factories\HasFactory;
6  use Illuminate\Database\Eloquent\Model;
7  use Illuminate\Database\Eloquent\Builder;
8
9  15 references | 0 implementations
10 class Place extends Model
11 {
12     use HasFactory;
13
14     2 references
15     public $table = 'place';
16
17     /**
18      * The attributes that are mass assignable.
19      *
20      * @var array<int, string>
21      */
22     0 references
23     protected $fillable = [
24         'numero',
25     ];
26
27     0 references | 0 overrides
28     public function scopeFree(Builder $query): void
29     {
30         $query->doesntHave('activeReservations');
31     }
32
33     /**
34      * Récupère l'historique des réservations pour la place.
35      */
36     1 reference | 0 overrides
37     public function reservations()
38     {
39         return $this->hasMany(Reservation::class);
40     }
41
42     1 reference | 0 overrides
43     public function activeReservations()
44     {
45         return $this->reservations()->active();
46     }
47 }
```

Initialisation et mise en place des model :

Dans Laravel, chaque table de base de données a un "modèle" correspondant qui nous permet d'interagir avec cette table.