

M2L

# Parking

Projet réaliser par le Groupe B de la SIO 2e 2024 – Charles de Foucauld, Paris 18e.

Mohammad-Tourab Syed Zaidi, Abdullah Asci et Vincent Chantraine  
Epreuve E5 - Parking

# PARKING

## Table des matières

<b>Introduction</b> .....	2
<b>Itération 1</b>	
MCD, Maquette & Parking .....	3
<b>Itération 2</b>	
Environnement .....	4
Vue, Routes.....	5
Contrôleurs .....	6
Database, Models .....	7

# PARKING

## INTRODUCTION

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.

Nous pouvons noter les besoins de cette manière :

- 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 attribué, 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.

Nous vous proposons de découvrir notre projet...

# PARKING

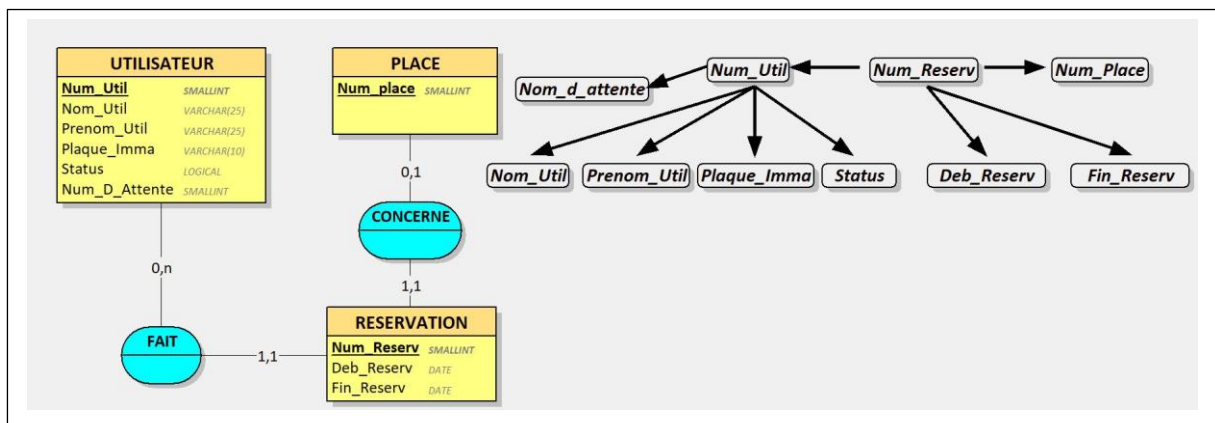
## Itération 1

Le but de cette itération ne porte que sur la documentation. C'est-à-dire nous avons pour objectif de réaliser :

- MCD
- Maquette pour l'application Web
- Plan du site avec URLs

Nous commençons à comprendre comment fonctionne GitHub.

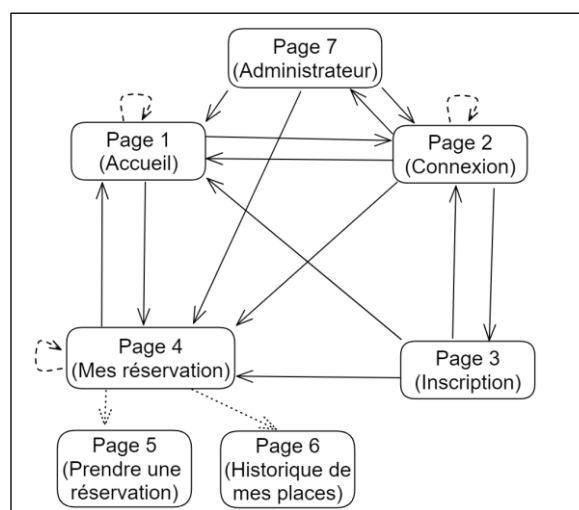
- Voici notre MCD :



- Notre maquette est disponible sur le site suivant :

<https://www.figma.com/file/fH02zofpNsKlqIDQoKggKV/Untitled?type=design&mode=design&t=1LXrAqCg8fQ7MXdC-1#MCD-AP1>

- Enfin, voici notre plan du site avec URLs :



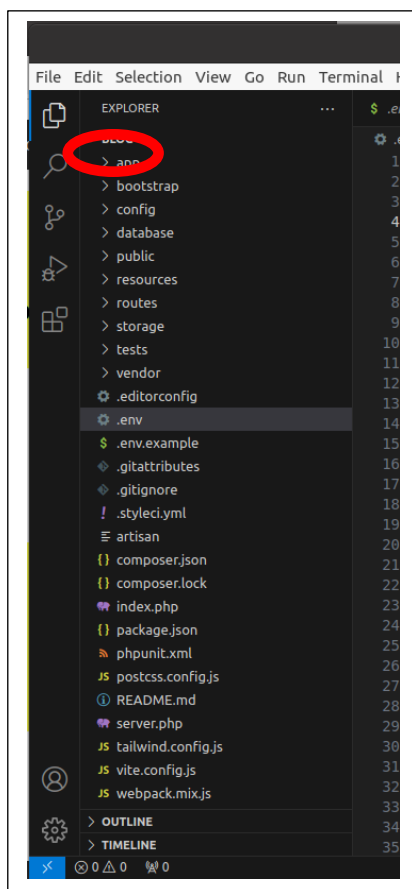
# PARKING

## Itération 2

Une fois les éléments de documentations effectués, on peut commencer à coder. Pour cela, il faut préparer l'environnement de développement en installant ou en ayant les éléments suivants :

- Machine Linux (préférence Ubuntu)  
Avoir PHP & Composer & MySQL
- Avoir Laravel
- Avoir un bon IDE : VSCode
- Avoir GitHub-Desktop

Un fois installé, nous devons apprendre ce qu'est Laravel et comment cela fonctionne.



Le répertoire "BLOG" englobe l'intégralité de notre projet. Ce dossier est soumis à un versionnage, et toute information que nous ne souhaitons pas inclure dans le versionnage est spécifiée dans un fichier « .env ».

# PARKING

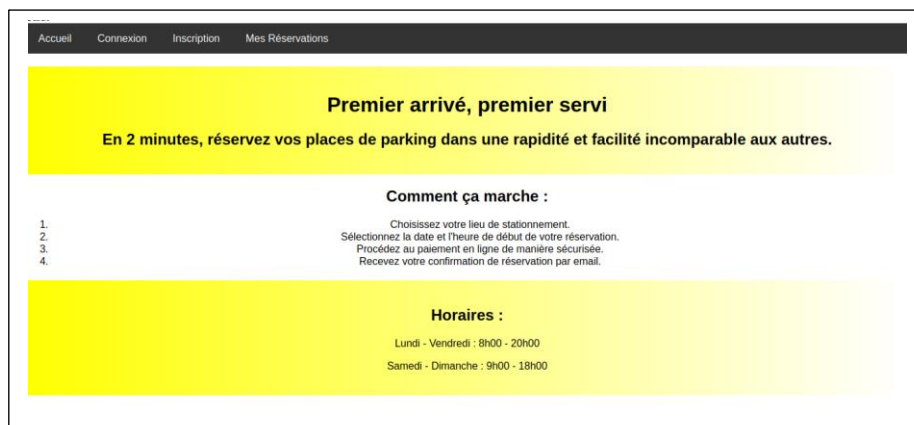
Nous avons compris la théorie du système de routes, vues, model et de Controller. Mais cela fut complexe de gérer aux premiers abords. Pour cela nous avons regardé des tutoriels, notamment un avec youtubeur « Garfikart » qui nous explique comment fonctionne Laravel.

Nous avons commencé par créer une page index.html :

## VUES :

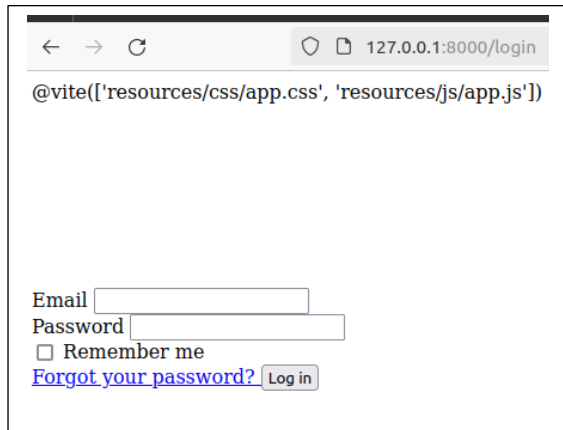
Les vues servent à afficher les données d'une application web à l'utilisateur final, en fournissant une interface utilisateur.

Voici l'accueil de notre site codé en HTML & CSS se trouvant sur la vue « toto.blade.php » :

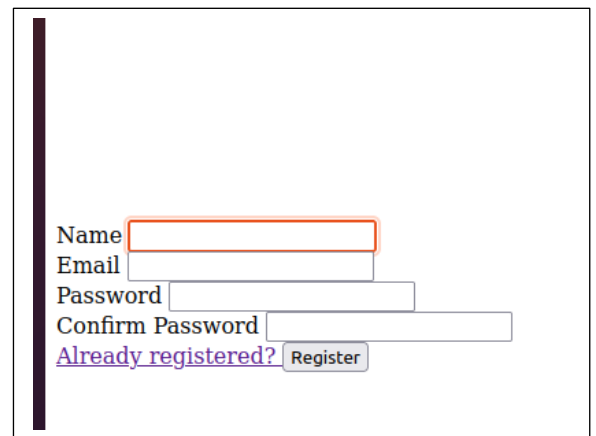


# PARKING

Pour notre page connexion, nous avons utilisé le package Breeze :



Page de connexion



Page d'inscription

## ROUTES

Les routes dans Laravel sont comme des panneaux indicateurs sur une autoroute. Elles indiquent à Laravel quelle action doit être exécutée lorsque quelqu'un visite une URL spécifique sur votre site Web. Voici nos routes :

```
Route::get('/', function () {
    return view('toto');
});

Route::get('/dashboard', function () {
    return view('dashboard');
})->middleware(['auth'])->name('dashboard');

require __DIR__.'/auth.php';
Route::get('/reserve', function () {
    return view('reserve');
});

require __DIR__.'/auth.php';
Route::get('/create', function () {
    return view('create');
});

Route::resource('reservations', ReservationController::class);
Route::post('/reservations', 'App\Http\Controllers\ReservationController@store');
Route::post('/reservations', 'App\Http\Controllers\ReservationController@store')->name('reservations.store');
```

# PARKING

## CONTROLEURS

Voici notre contrôleur, qui gère le processus de création de nouvelles réservations dans votre application. Elle assure également que les réservations antérieures sont nettoyées, évitant ainsi toute confusion ou problème avec les réservations expirées.

```
class ReservationController extends Controller
{
    // Enregistrer une nouvelle réservation dans la base de données
    public function store(Request $request)
    {
        // Récupérer l'ID de l'utilisateur authentifié
        $userID = auth()->user()->id;

        // Supprimer les réservations antérieures à la date actuelle
        Reservation::where('Fin_Reserv', '<', now())->delete();

        // Récupérer un ID_Place disponible
        $availablePlace = Place::whereNotExists(function ($query) {
            $query->select(DB::raw(1))
                ->from('reservations')
                ->whereColumn('reservations.place_id', 'places.id')
                ->orWhere('reservations.Fin_Reserv', '<', now());
        })->value('id');

        if (!$availablePlace) {
            // Aucune place disponible, rediriger vers la liste d'attente
            return ('error, Aucune place disponible pour effectuer la
réservation.');
```



# PARKING

## DATABASE

Bien sûr nous devons créer notre base de données. Nous la nommerons « parking ». Nous créerons les tables grâce à la commande suivante :

php artisan make:migration nom\_de\_la\_table

Ce dossier sera visible sur `blog/database/migration/*nom du fichier*.php`

```
class CreateTablePlace extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('places', function (Blueprint $table) {
            $table->id();
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::dropIfExists('places');
    }
}
```

## MODELS

Un modèle dans Laravel agit comme un gestionnaire de données pour une application web. Il permet à l'application d'interagir avec la base de données en récupérant, en enregistrant et en modifiant les données. Voici un exemple de notre model « réservation.php »

```
class Reservation extends Model
{
    // Nom de la table dans la base de données
    protected $table = 'reservations';
    public $timestamps = false; // Désactiver les horodatages
    // Colonnes pouvant être mass assignable
    protected $fillable = [
        'Deb_Reserv',
        'Fin_Reserv',
        'user_id',
        'place_id',
    ];

    // Ajoutez vos relations ici, par exemple :
    public function user()
    {
        return $this->belongsTo(User::class, 'user_id');
    }

    public function place()
    {
        return $this->belongsTo(Place::class, 'place_id');
    }
}
```

# PARKING

Nous avons également pu créer une page pour réserver et qui est coordonnée avec la base de données :



**\*\*\* LE PROJET N'EST PAS ENCORE FINI \*\*\***