M2L

Parking

Projet réaliser par le groupe C

SIO2– Charles de Foucauld, Paris 18e.

PARKING

Table des matières

PARKING

INTRODUCTION

Pour éviter que le parking ne devienne un vrai casse-tête avec des voitures mal garées partout, nous avons décidé d'attribuer à chaque personne qui le demande une place de parking numérotée. Voici ce dont nous avons besoin :

- 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 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'administateur 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.

Voilà ce qu'on vous propose pour notre projet...

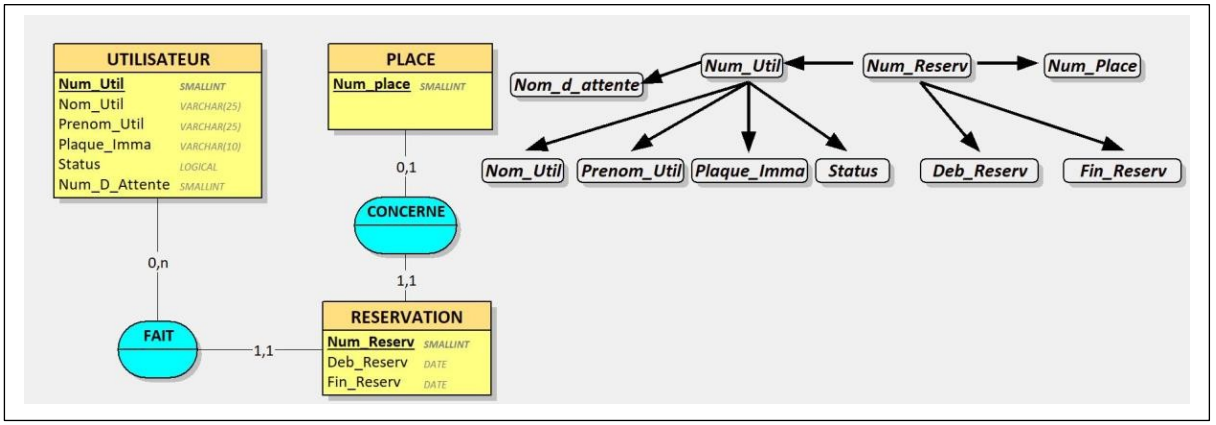
PARKING

Itération 1

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

- MCD
- Maquette de l'application Web qu'on a faite sur Figma
- Plan du site avec URLs

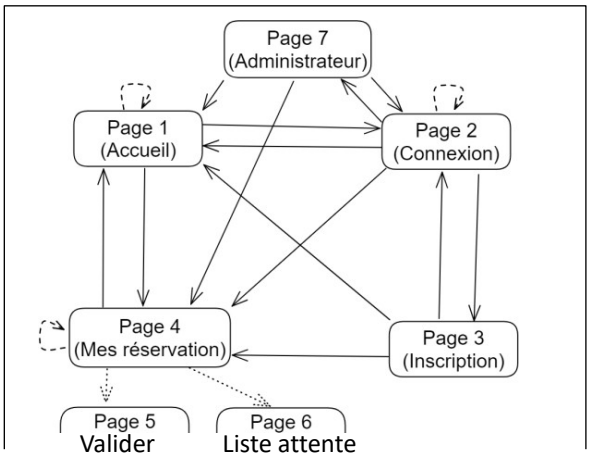
- Voici notre MCD :



- 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 :



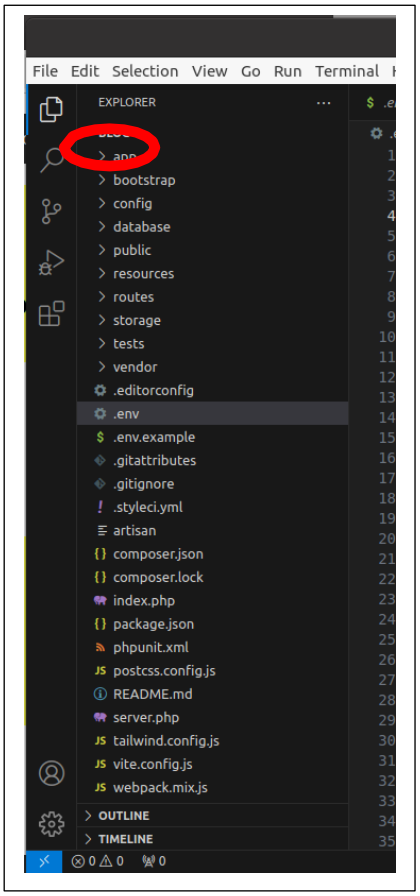
PARKING

Itération 2

Une fois les éléments de documentions effectués, on peut commencer à coder. Pour cela, on a due préparer notre environnement de développement en installant et configurant les éléments suivants :

- Machine virtuel sous Linux
- Installer PHP & MySQL
- Avoir Laravel(Framework PHP)
- Avoir un IDE : VSCode
- Installer GitHub-Desktop

Un fois installé, nous devions 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 pour partager nos modification il faut que nous fêtions des push de modifications et pull pour recevoir c'elle des autres .

PARKING

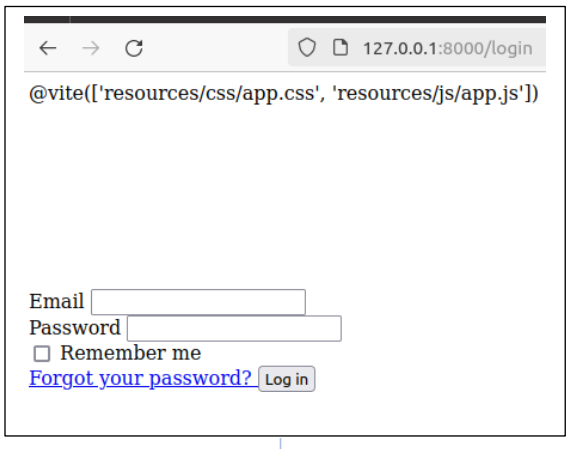
Nous avions compris la théorie du système de routes, vues, model et de Controller.
Nous avons commencé par créer une page index.html :

Les VUES :

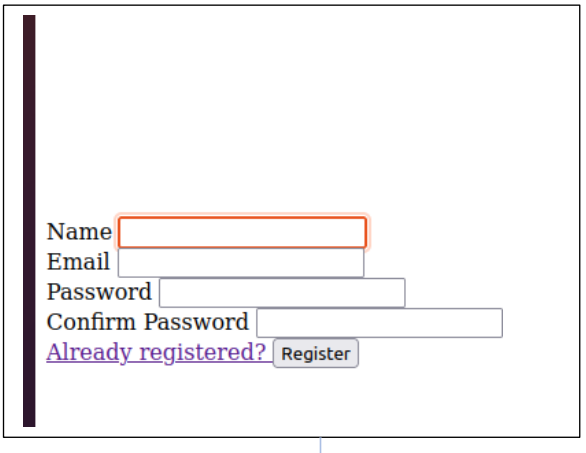
Les vues servent à afficher les données d'une application web à l'utilisateur final, en fournissant une interface à l'utilisateur.

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

Pour notre page connexion, nous avons utilisé le package breeze qui fourni une page de connexion et d'inscription a notre application web, les informations d'indentification sont stocker dans une base de données (Parking) sur MY SQL:



Page de connexion



Page d'inscription

PARKING

LES 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.

Nos routes actuels que l’on retrouve dans le fichier web.php :

parkinggg / routes / **web.php**

Code | Blame | 37 lines (29 loc) · 1.03 KB | Code 55% faster with GitHub Copilot | Raw

```
4
5   /*
6   |--------------------------------------------------------------------------
7   | Web Routes
8   |--------------------------------------------------------------------------
9   |
10  | Here is where you can register web routes for your application. These
11  | routes are loaded by the RouteServiceProvider within a group which
12  | contains the "web" middleware group. Now create something great!
13  |
14  */
15
16  Route::get('/', function () {
17      return view('test');
18  });
19
20  Route::get('/dashboard', function () {
21      return view('dashboard');
22  })->middleware(['auth'])->name('dashboard');
23
24  require __DIR__.'/auth.php';
25  Route::get('/reserve', function () {
26      return view('reserve');
27  });
28
29  require __DIR__.'/auth.php';
30  Route::get('/create', function () {
31      return view('create');
32  });
33
34
35  Route::resource('reservations', ReservationController::class);
36  Route::post('/reservations', 'App\Http\Controllers\ReservationController@store');
37  Route::post('/reservations', 'App\Http\Controllers\ReservationController@store')->name('reservations.store');
```

PARKING

CONTROLLEURS

Les contrôleurs gère le processus de création de nouvelles réservations dans notre application. Elle assure également que les réservations antérieures sont nettoyées, évitant ainsi toute confusion ou problème avec les réservations expirées. En créant des enregistrements dans chaque tables(Place, Réservation).

```
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.');
        }

        // Calculer la date et l'heure actuelles
        $currentDateTime = now();
        $expirationDateTime = $currentDateTime->copy()->addMinutes(1);

        // Créer une nouvelle réservation avec l'ID de la place disponible
        $reservation = new Reservation();
        $reservation->Deb_Reserv = $currentDateTime;
        $reservation->Fin_Reserv = $expirationDateTime;
        $reservation->user_id = $userID;
        $reservation->place_id = $availablePlace;
        $reservation->save();

        // Rediriger avec un message de succès
        return ('success, Réservation ajoutée avec succès.');
    }
```

PARKING

DATABASE

Notre base de donnes s'appelle « 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

```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');
    }
}
```

PARKING

LES 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 »

```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');
    }
}
```

Nous avons également pue créer une page de réservation qui est coordonnées avec la base de données.
Lorsque que l'utilisateur est connecter et qu'il veut réserver une place il clique sur le bouton jaune pour que s'il reste des places , une place lui soit attribuer aléatoirement sinon il ya un message lui disant d'attendre car aucune place n'est encore disponible.

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