Documentation du système d'attribution de places de parking

Une application développée avec Laravel

Réalisé par :
Yassine TAMANI
Souheyl LABIDI

23 mars 2025

Table des matières

Chapitre 1

Introduction

1.1 Présentation du projet

Ce document décrit le système d'attribution de places de parking développé avec le framework Laravel (PHP). L'application permet la gestion des réservations de places de parking dans un environnement professionnel.

1.2 Objectifs

Le système vise à atteindre les objectifs suivants :
— Permettre aux utilisateurs de réserver des places de parking disponibles
— Gérer une file d'attente pour les demandes de places
— Offrir une interface d'administration pour gérer les utilisateurs, les places et les réservations
— Maintenir un historique des attributions de places
— Automatiser le processus d'attribution en fonction des disponibilités

1.3 Technologies utilisées

Le projet a été développé avec les technologies suivantes :
— PHP 8.x avec le framework Laravel
— Base de données MySQL/MariaDB
— HTML, CSS, JavaScript pour le frontend
— Système d'authentification de Laravel
— ORM Eloquent pour l'accès aux données

Chapitre 2

Architecture du système

2.1 Structure du projet

L'application suit l'architecture MVC (Modèle-Vue-Contrôleur) de Laravel :

- **Modèles** : Représentent les entités de l'application (Utilisateur, Place, Réservation, File d'attente)
- **Vues** : Interfaces utilisateur pour afficher les données
- **Contrôleurs** : Gèrent la logique métier et les requêtes HTTP
- **Routes** : Définissent les points d'entrée de l'application
- **Migrations** : Gèrent la structure de la base de données

2.2 Diagramme de la base de données

Le système utilise une base de données relationnelle avec les tables suivantes :

Structure de la base de données

- **users** : Stocke les informations des utilisateurs
- **places** : Contient les données relatives aux places de parking
- **reservations** : Enregistre les réservations des places
- **file_attentes** : Gère la file d'attente des demandes

2.3 Relations entre les entités

- Un utilisateur peut avoir plusieurs réservations (relation one-to-many)
- Un utilisateur peut être dans la file d'attente (relation one-to-one)

— Une place peut avoir plusieurs réservations dans le temps (relation one-to-many)
— Chaque réservation est liée à un utilisateur et à une place spécifique

Chapitre 3

Modèles et entités

3.1 Modèle User

Le modèle User représente les utilisateurs du système.

```
class User extends Authenticatable
{
    use HasFactory, Notifiable;

    protected $fillable = [
        'name',
        'email',
        'password',
        'est_admin',
        'est_valide',
        'force_password_change',
    ];

    protected $casts = [
        'email_verified_at' => 'datetime',
        'password' => 'hashed',
        'est_admin' => 'boolean',
        'est_valide' => 'boolean',
        'force_password_change' => 'boolean',
    ];

    public function reservations(): HasMany
    {
        return $this->hasMany(Reservation::class);
    }

```

```
27      public function fileAttente(): HasOne
28      {
29          return $this->hasOne(FileAttente::class);
30      }
31
32      public function estAdmin(): bool
33      {
34          return $this->est_admin;
35      }
36  }
```

3.2 Modèle Place

Le modèle Place représente les places de parking disponibles.

```
1   class Place extends Model
2   {
3       use HasFactory;
4
5       protected $fillable = [
6           'numero',
7           'description',
8           'est_disponible',
9       ];
10
11      protected $casts = [
12          'est_disponible' => 'boolean',
13      ];
14
15      public function reservations(): HasMany
16      {
17          return $this->hasMany(Reservation::class);
18      }
19
20      public function reservationsActives()
21      {
22          return $this->reservations()->where('est_active',
                ↪  true);
23      }
24
25      public function estReservee(): bool
26      {
```

```
27         return !$this->est_disponible;
28     }
29 }
```

3.3 Modèle Reservation

Le modèle Reservation gère les réservations des places.

```
1  final class Reservation extends Model
2  {
3      use HasFactory;
4
5      protected $fillable = [
6          'user_id',
7          'place_id',
8          'date_debut',
9          'date_fin',
10         'est_active',
11     ];
12
13     protected $casts = [
14         'date_debut' => 'datetime',
15         'date_fin' => 'datetime',
16         'est_active' => 'boolean',
17     ];
18
19     public function user(): BelongsTo
20     {
21         return $this->belongsTo(User::class);
22     }
23
24     public function place(): BelongsTo
25     {
26         return $this->belongsTo(Place::class);
27     }
28 }
```

3.4 Modèle FileAttente

Le modèle FileAttente gère la file d'attente pour l'attribution des places.

```php
final class FileAttente extends Model
{
    use HasFactory;

    protected $fillable = [
        'user_id',
        'position',
        'date_demande',
    ];

    protected $casts = [
        'date_demande' => 'datetime',
    ];

    public function user(): BelongsTo
    {
        return $this->belongsTo(User::class);
    }
}
```

Chapitre 4

Contrôleurs et logique métier

4.1 ReservationController

Ce contrôleur gère l'ensemble des opérations liées aux réservations de places.

Fonctionnalités principales

- Affichage des réservations de l'utilisateur
- Création de nouvelles réservations
- Visualisation de l'historique des réservations
- Suppression de réservations
- Vérification des disponibilités

4.2 FileAttenteController

Ce contrôleur gère la file d'attente des utilisateurs souhaitant obtenir une place.

Fonctionnalités principales

- Affichage de la file d'attente
- Ajout d'un utilisateur à la file d'attente
- Retrait d'un utilisateur de la file d'attente
- Gestion des positions dans la file d'attente

4.3 PlaceController

Ce contrôleur gère les opérations liées aux places de parking.

Fonctionnalités principales

- Affichage des places disponibles
- Visualisation des détails d'une place
- Vérification de la disponibilité

4.4 Administration

Plusieurs contrôleurs dédiés à l'administration du système :
- **UtilisateurController** : Gestion des utilisateurs
- **HistoriqueController** : Consultation de l'historique des attributions
- **AttributionController** : Attribution manuelle des places
- **DashboardController** : Tableau de bord administratif

Chapitre 5

Routes et points d'entrée

5.1 Routes principales

```php
// Page d'accueil
Route::get('/', function () {
    return view('home');
});

// Routes pour les places
Route::middleware(['auth'])->group(function () {
    Route::get('/places', [PlaceController::class, 'index'])->name('places.index');
    Route::get('/places/{place}', [PlaceController::class, 'show'])->name('places.show');

    // Routes pour les r servations
    Route::get('/reservations', [ReservationController::class, 'index'])->name('reservations.index');
    Route::get('/reservations/create', [ReservationController::class, 'create'])->name('reservations.create');
    Route::get('/reservations/history', [ReservationController::class, 'history'])->name('reservations.history');
    Route::post('/reservations', [ReservationController::class, 'store'])->name('reservations.store');
    Route::get('/reservations/{reservation}', [ReservationController::class, 'show'])->name('reservations.show');
```

```
17      Route::delete('/reservations/{reservation}', [
            ↪ ReservationController::class, 'destroy'])->name
            ↪ ('reservations.destroy');
18
19      // Routes pour la file d'attente
20      Route::get('/file-attente', [FileAttenteController::
            ↪ class, 'index'])->name('file-attente.index');
21      Route::post('/file-attente', [FileAttenteController::
            ↪ class, 'store'])->name('file-attente.store');
22      Route::delete('/file-attente/{fileAttente}', [
            ↪ FileAttenteController::class, 'destroy'])->name
            ↪ ('file-attente.destroy');
23  });
```

5.2 Routes d'administration

```
1   // Routes pour l'administration
2   Route::prefix('admin')->group(function () {
3       Route::get('/dashboard', [DashboardController::class,
            ↪  'index'])
4           ->middleware(CheckAdmin::class)
5           ->name('admin.dashboard');
6
7       // Routes pour la gestion des utilisateurs
8       Route::resource('utilisateurs', UtilisateurController
            ↪ ::class)
9           ->middleware(CheckAdmin::class)
10          ->names([
11              'index' => 'admin.utilisateurs.index',
12              'create' => 'admin.utilisateurs.create',
13              'store' => 'admin.utilisateurs.store',
14              'show' => 'admin.utilisateurs.show',
15              'edit' => 'admin.utilisateurs.edit',
16              'update' => 'admin.utilisateurs.update',
17              'destroy' => 'admin.utilisateurs.destroy',
18          ]);
19
20      // Routes pour la gestion des places
21      Route::resource('places', AdminPlaceController::class
            ↪ )
22          ->middleware(CheckAdmin::class);
```

```
23
24      // Routes pour l'historique des attributions
25      Route::get('/historique', [HistoriqueController::
            ↪ class, 'index'])
26          ->middleware(CheckAdmin::class)
27          ->name('admin.historique.index');
28  });
```

5.3 Routes d'authentification

```
1   // Routes pour l'authentification
2   Route::middleware(['web', 'throttle:auth'])->group(
        ↪ function () {
3       Route::get('/login', [LoginController::class, '
            ↪ showLoginForm'])->name('login');
4       Route::post('/login', [LoginController::class, 'login
            ↪ ']);
5       Route::post('/logout', [LoginController::class, '
            ↪ logout'])->name('logout');
6       Route::get('/register', [RegisterController::class, '
            ↪ showRegistrationForm'])->name('register');
7       Route::post('/register', [RegisterController::class,
            ↪ 'register']);
8   });
9
10  // Routes pour la r initialisation de mot de passe
11  Route::middleware(['web', 'throttle:password'])->group(
        ↪ function () {
12      Route::get('/password/reset', [
            ↪ PasswordResetController::class, '
            ↪ showLinkRequestForm'])->name('password.request'
            ↪ );
13      Route::post('/password/email', [
            ↪ PasswordResetController::class, '
            ↪ sendResetLinkEmail'])->name('password.email');
14      Route::get('/password/reset/{token}', [
            ↪ PasswordResetController::class, 'showResetForm'
            ↪ ])->name('password.reset');
15      Route::post('/password/reset', [
            ↪ PasswordResetController::class, 'reset'])->name
            ↪ ('password.update');
```

```
16 });
```

Chapitre 6

Migrations et structure de la base de données

6.1 Migrations principales

```
// Migration pour la table des places
Schema::create('places', function (Blueprint $table) {
    $table->id();
    $table->string('numero')->unique();
    $table->boolean('est_disponible')->default(true);
    $table->text('description')->nullable();
    $table->timestamps();
});

// Migration pour la table des r servations
Schema::create('reservations', function (Blueprint $table
    ↪ ) {
    $table->id();
    $table->foreignId('user_id')->constrained('users');
    $table->foreignId('place_id')->constrained('places');
    $table->dateTime('date_debut');
    $table->dateTime('date_fin');
    $table->boolean('est_active')->default(true);
    $table->timestamps();
});

// Migration pour la table de la file d'attente
Schema::create('file_attentes', function (Blueprint
    ↪ $table) {
    $table->id();
```

```
24      $table->foreignId('user_id')->constrained('users');
25      $table->integer('position');
26      $table->dateTime('date_demande');
27      $table->timestamps();
28  });
```

6.2 Évolution de la structure

Le projet a connu plusieurs évolutions de sa structure de données :

- Ajout du champ `force_password_change` pour forcer le changement de mot de passe
- Mise à jour du statut de disponibilité des places
- Migration des utilisateurs du modèle Utilisateur vers le modèle User standard de Laravel
- Ajout des champs d'administration et de validation pour les utilisateurs

Chapitre 7

Fonctionnalités

7.1 Système de réservation

Processus de réservation

1. L'utilisateur consulte les places disponibles
2. Il sélectionne une place et une période de réservation
3. Le système vérifie la disponibilité de la place pendant la période demandée
4. Si la place est disponible, une réservation est créée
5. Sinon, l'utilisateur peut s'inscrire en file d'attente

7.2 Gestion de la file d'attente

La file d'attente fonctionne selon les principes suivants :
— Les utilisateurs sont positionnés selon l'ordre d'inscription
— Lorsqu'une place se libère, le système peut notifier le premier utilisateur en file d'attente
— L'administrateur peut également attribuer manuellement les places
— Un utilisateur peut se retirer de la file d'attente à tout moment

7.3 Administration du système

Le panneau d'administration permet de :
— Gérer les utilisateurs (création, modification, validation, suppression)
— Gérer les places (ajout, modification, suppression)

— Consulter et filtrer l'historique des réservations
— Attribuer manuellement des places aux utilisateurs
— Gérer la file d'attente

7.4 Sécurité

L'application intègre plusieurs mécanismes de sécurité :
— Authentification des utilisateurs
— Protection contre les attaques CSRF
— Limitation de débit sur les routes sensibles
— Middleware de vérification des droits administrateur
— Hachage sécurisé des mots de passe
— Mécanisme de récupération de mot de passe
— Forçage du changement de mot de passe

Chapitre 8

Installation et configuration

8.1 Prérequis

Pour installer et exécuter l'application, les éléments suivants sont nécessaires :

- PHP 8.1 ou supérieur
- Composer (gestionnaire de dépendances PHP)
- MySQL ou MariaDB
- Serveur web (Apache, Nginx, etc.)
- Extensions PHP requises par Laravel

8.2 Installation

1. Cloner le dépôt Git du projet
2. Installer les dépendances PHP avec Composer
3. Configurer le fichier .env avec les paramètres de base de données
4. Créer la base de données
5. Exécuter les migrations
6. Démarrer le serveur de développement ou configurer un serveur web

8.3 Configuration de la base de données

```
1 # Cr er la base de donn es et l'utilisateur
2 CREATE DATABASE parking_attribution CHARACTER SET utf8mb4
    ↪  COLLATE utf8mb4_unicode_ci;
```

```
CREATE USER 'parking_user'@'localhost' IDENTIFIED BY '
    ↪ votre_mot_de_passe_securise';
GRANT ALL PRIVILEGES ON parking_attribution.* TO '
    ↪ parking_user'@'localhost';
FLUSH PRIVILEGES;
```

8.4 Variables d'environnement

Configuration des variables d'environnement dans le fichier .env :

```
APP_NAME="Gestion de parking"
APP_ENV=production
APP_KEY=base64:...
APP_DEBUG=false
APP_URL=http://votre-domaine.com

DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=parking_attribution
DB_USERNAME=parking_user
DB_PASSWORD=votre_mot_de_passe_securise
```

Chapitre 9

Maintenance et évolutions

9.1 Maintenance du système

Pour maintenir le système en bon état de fonctionnement :
— Effectuer régulièrement des sauvegardes de la base de données
— Mettre à jour les dépendances PHP via Composer
— Surveiller les journaux d'erreurs de l'application
— Tester régulièrement les fonctionnalités critiques

9.2 Évolutions possibles

Le système pourrait être amélioré avec les fonctionnalités suivantes :
— Système de notifications par email pour les attributions
— Interface mobile responsive ou application mobile dédiée
— Statistiques d'utilisation des places
— Système de cartographie des places de parking
— Intégration avec des systèmes de contrôle d'accès physique
— Réservations récurrentes pour les utilisateurs réguliers
— API pour l'intégration avec d'autres systèmes d'entreprise

Chapitre 10

Conclusion

10.1 Récapitulatif

Le système d'attribution de places de parking répond aux besoins de gestion des places dans un environnement professionnel. Il permet :

- Une gestion efficace des réservations
- Un suivi transparent de la file d'attente
- Une administration simplifiée du parc de stationnement
- Une traçabilité des attributions

10.2 Remerciements

Nous tenons à remercier toutes les personnes ayant contribué à ce projet, ainsi que l'équipe de Laravel pour la qualité de leur framework.

Réalisé par Yassine TAMANI et Souheyl LABIDI