Hedvi Nadenane

Année
2024-2025

Documentation AP Parking

Sommaire :

2.4.7 *app/Http/Controllers/adminController.php*.....

Présentation du Projet :

1.1 En quoi consiste ce Projet ? :

Le projet consiste à développer une **application de gestion des places de parking** permettant aux membres du personnel de **réserver une place numérotée** de manière automatisée.

Lorsqu'un utilisateur fait une demande, une place libre lui est attribuée immédiatement, et en cas d'indisponibilité, il est placé en file d'attente.

Un **administrateur** supervise l'ensemble du système, valide les inscriptions, gère les réservations et peut attribuer des places manuellement. L'application devra être **sécurisée, accessible via un réseau local** et proposer une interface claire et responsive. Elle inclura également une **base de données** pour stocker les informations des utilisateurs, des réservations et de l'historique des attributions.

Présentation du Projet :

1.2 Explication des différentes Pages :

Explication d'une route :

En Laravel, une **route** permet de définir quelle action doit être exécutée lorsqu'un utilisateur accède à une URL spécifique. Elle associe une URL à une fonction ou à un contrôleur.Les routes sont définies dans les fichiers de routes situés dans le dossier routes/, comme Web.php

Explication d'un Controller :

Un **Controller** en Laravel est une classe qui gère la logique du site. Il fait le lien entre les routes (les URL) et les vues (les pages affichées).

Explication simple :

Quand un utilisateur fait une action (comme cliquer sur un bouton), la requête est envoyée à un **Controller**. Ce dernier traite la demande, récupère les données nécessaires, et renvoie la réponse appropriée

Explication d'une vue :

Une **Vue** en Laravel est un fichier qui contient le code HTML affiché à l'utilisateur. Elle sert à séparer l'affichage de la logique du projet, ce qui permet d'avoir un code plus propre et organisé.

Explication d'une migration (BDD) :

Une **migration** en Laravel est un fichier qui permet de créer, modifier ou supprimer des tables dans la base de données. Elle sert à gérer la structure de la base de données de manière organisée et versionnée, un peu comme un "git" pour la base de données.

Explication d'un seeders (BDD) :

Un **seeder** en Laravel est un fichier qui permet d'insérer des données automatiquement dans la base de données. Il sert à remplir les tables avec des valeurs par défaut, utiles pour tester l'application sans devoir entrer les données manuellement.

Descriptif des différentes Pages :

2.1.1 Contenue de la page : routes/auth.php:

Ce fichier utilise **Laravel Breeze** pour gérer l'authentification. Il définit des groupes de routes en fonction de l'état de connexion de l'utilisateur :

- **middleware('guest') :** Ces routes sont accessibles uniquement aux utilisateurs non connectés (ex. connexion, inscription).
- **middleware('auth') :** Ces routes sont accessibles uniquement aux utilisateurs connectés (ex. vérification d'email, déconnexion).

1.Routes accessibles aux invités (`guest`)

```php
Route::middleware('guest')->group(function () {
    Route::get('register', [RegisteredUserController::class, 'create'])
                ->name('register');

    Route::post('register', [RegisteredUserController::class, 'store']);
```

GET /register → Affiche le formulaire d'inscription.

POST /register → Enregistre un nouvel utilisateur

[RegisteredUserController::class, 'create'] : Appelle la méthode **create()** dans le contrôleur **RegisteredUserController** pour afficher le formulaire d'inscription.

2.Routes accessibles aux client (`login`)

```php
Route::post('login', [AuthenticatedSessionController::class, 'store']);

Route::get('forgot-password', [PasswordResetLinkController::class, 'create'])
            ->name('password.request');
```

GET `/login` → Affiche le formulaire de connexion.

POST `/login` → Authentifie l'utilisateur et le connecte

3.Mot de passe oublié (`forgot-password`)

```php
Route::get('forgot-password', [PasswordResetLinkController::class, 'create'])
    ->name('password.request');

Route::post('forgot-password', [PasswordResetLinkController::class, 'store'])
    ->name('password.email');
```

GET `/forgot-password` → Affiche le formulaire pour entrer son email et recevoir un lien de réinitialisation.

POST `/forgot-password` → Envoie un email contenant le lien pour réinitialiser le mot de passe.

4. Réinitialisation du mot de passe (`reset-password`)

```php
Route::get('reset-password/{token}', [NewPasswordController::class, 'create'])
    ->name('password.reset');

Route::post('reset-password', [NewPasswordController::class, 'store'])
    ->name('password.store');
```

GET `/reset-password/{token}` → Affiche le formulaire de réinitialisation avec le token.

POST `/reset-password` → Met à jour le mot de passe dans la base de données.

5. Routes accessibles aux utilisateurs connectés (`auth`)

Ces routes permettent de gérer la vérification d'email, le changement de mot de passe et la déconnexion.

```php
Route::middleware('auth')->group(function () {
    Route::get('verify-email', EmailVerificationPromptController::class)
                ->name('verification.notice');

    Route::get('verify-email/{id}/{hash}', VerifyEmailController::class)
                ->middleware(['signed', 'throttle:6,1'])
                ->name('verification.verify');

    Route::post('email/verification-notification', [EmailVerificationNotification
                ->middleware('throttle:6,1')
                ->name('verification.send');
```

GET `/verify-email` → Invite l'utilisateur à vérifier son email.

5. Routes accessibles aux utilisateurs connectés (`auth`)

6. Confirmation du mot de passe (`confirm-password`)

```php
Route::get('confirm-password', [ConfirmablePasswordController::class, 'show'])
            ->name('password.confirm');

Route::post('confirm-password', [ConfirmablePasswordController::class, 'store']);
```

GET `/confirm-password` → Demande de saisir à nouveau son mot de passe pour des actions sensibles (ex. modifier email).

POST `/confirm-password` → Vérifie si le mot de passe est correct.

7. Mise à jour du mot de passe (`password`)

```php
Route::put('password', [PasswordController::class, 'update'])->name('password.up
```

PUT `/password` → Permet de modifier son mot de passe depuis son profil.

.

Descriptif des différentes Pages :

2.1.2 Contenue de la page : routes/web.php:

Ce fichier définit toutes les **routes web** de votre application Laravel. Les routes sont utilisées pour lier les URL aux actions ou **méthodes** des contrôleurs. Voici une explication détaillée de chaque partie de ce fichier.

1.Routes pour la page d'accueil (' / ')

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

- Cette route est associée à l'URL `/`, la page d'accueil de votre application.
- Lorsque l'utilisateur accède à cette URL, Laravel affiche la vue **`welcome`**. Il s'agit généralement de la première page de l'application ou d'une page d'accueil par défaut.

2. Dashboard sécurisé (utilisateur connecté obligatoire)

```php
Route::middleware(['auth'])->group(function () {
    Route::get('/dashboard', [ProfileController::class, 'index'])->name('dashboar
    Route::post('/profile/password', [ProfileController::class, 'updatePassword']
});
```

`Route::middleware(['auth'])` : Ce groupe de routes est accessible uniquement aux utilisateurs authentifiés (connectés). Si un utilisateur n'est pas connecté, il sera redirigé vers la page de connexion.

Route `/dashboard` : Affiche le tableau de bord de l'utilisateur connecté. La méthode `index` du `ProfileController` gère l'affichage de cette page.

Route `/profile/password` : Permet à l'utilisateur de modifier son mot de passe via la méthode `updatePassword` du même contrôleur.

3. Routes nécessitant une authentification

Les routes suivantes sont également protégées par le middleware **`auth`,** donc elles ne sont accessibles qu'aux **utilisateurs connectés.**

```php
Route::middleware(['auth'])->group(function () {
    // Routes pour les utilisateurs
    Route::resource('users', UserController::class);

    // Routes pour le parking
    Route::resource('parking', ParkingController::class);

    // Routes pour les réservations
    Route::resource('reservations', ReservationController::class);

    // Routes pour l'historique
    Route::resource('historiques', HistoriqueController::class);

    // Routes pour l'attente
    Route::resource('attente', AttenteController::class);
```

Toutes ces **routes** permettent d'afficher une action selon le commentaire qui lui est inscrit.

4.Gestion du profil utilisateur

```php
Route::get('/profile', [ProfileController::class, 'edit'])->name('profile.edit')
Route::patch('/profile', [ProfileController::class, 'update'])->name('profile.up
Route::delete('/profile', [ProfileController::class, 'destroy'])->name('profile.
```

- Ces routes permettent à l'utilisateur de modifier ou supprimer son profil (nom, email, etc.).

5. Routes spécifiques aux administrateurs

```php
Route::resource('admin', AdminController::class)
    ->middleware('can:viewAny,App\Models\User');
```

Ces **routes** sont utilisées pour gérer les utilisateurs (ex. liste des utilisateurs, suppression, modification).

La route est protégée par le middleware **`can:viewAny,App\Models\User`**, qui garantit que seuls les utilisateurs ayant la permission appropriée peuvent y accéder.

6.Gestion des places de parking (Admin seulement)

```
Route::post('/parking/{parking}/occuper', [ParkingController::class, 'marquerOccu
Route::post('/parking/{parking}/liberer', [ParkingController::class, 'marquerLibre
```

Ces routes permettent aux administrateurs de marquer les places de parking comme occupées ou libres. Ces actions sont contrôlées par la méthode **marquerOccupee** et **marquerLibre** du **ParkingController**.

7.Gestion de la liste d'attente

```
Route::post('/attente/{id}/update-position', [AttenteController::class, 'updatePo
    ->name('attente.updatePosition');
```

Cette **route** permet de mettre à jour la position d'un utilisateur dans la liste d'attente.

8.Historique des attributions

```
Route::resource('historique', HistoriqueController::class);
```

Ces **routes** permettent de consulter et gérer l'historique des actions concernant les parkings et les réservations.

9.Réservations

```
Route::post('/reservation', [ReservationController::class, 'store'])->name
```

Cette **route** permet à un utilisateur de faire une nouvelle réservation pour un parking.

10.Routes pour l'inscription (inscription de nouveaux utilisateurs)

```
Route::get('/register', [RegisteredUserController::class, 'create'])->name('regis
Route::post('/register', [RegisteredUserController::class, 'store']);
```

Route GET **`/register`** : Affiche le formulaire d'inscription, géré par la méthode `create` du **`RegisteredUserController`.**

Route POST **`/register`** : Envoie les informations saisies pour créer un nouvel utilisateur, géré par la méthode `store` du même contrôleur.