Plan du projet Laravel ePark

1. Initialisation et configuration du projet

- Initialisation du dépôt Git (`git init`)
- Création du fichier `.gitignore` (exclusion des fichiers/dossiers sensibles et inutiles)
- Création du projet Laravel
- Configuration de la base de données
- Installation de Breeze/Jetstream pour l'authentification

2. Modélisation des données (migrations & modèles)

- Utilisateur (User) avec gestion des rôles (Locataire, Propriétaire, ou les deux)
- Site (nom, adresse, propriétaire)
- Place (site, disponibilité, caractéristiques)
- Réservation (utilisateur, place, date, statut)

3. Authentification et gestion des rôles

- Inscription/connexion
- Attribution et gestion des rôles
- Middleware pour sécuriser les routes selon le rôle

4. Fonctionnalités principales (contrôleurs & vues)

- Tableau de bord selon le rôle
- Réservation d'une place (locataire)
- Ajout de place à un site existant (propriétaire)
- Création d'un nouveau site (propriétaire)
- Gestion des réservations, places, sites
- Pénalités de dépassement (tranches 1h/3h/1 jour)

5. Interface utilisateur (Blade ou API)

- Pages responsives pour mobile (Blade + Bootstrap/Tailwind)
- (Optionnel) API REST pour front mobile séparé

6. Fonctionnalités avancées (optionnel)

- Carte interactive (Leaflet/Google Maps)
- Notifications (email, in-app)

Plan du projet Laravel ePark — état et prochaines étapes (11-02-2026)

Ce plan liste les phases réalisées et les actions prioritaires pour stabiliser la production et automatiser le déploiement.

Phase actuelle — stabilisation & release

- Changements réalisés : UI harmonisee (palette/composants), logo ePark, accessibilite de base (aria/contrastes), stats admin, annulation/modification reservation, soft delete places.

Tâches immédiates (priorité haute)

1. Exécuter les migrations en production
   - `php artisan migrate --force`
   - Vérifier que la table `notifications` existe et que les anciennes notifications sont maintenant dans `app_notifications` si présent.
   - Verifier la colonne `places.cancel_deadline_hours` et `places.deleted_at`.
2. Nettoyer caches et vues
   - `php artisan config:clear && php artisan cache:clear && php artisan view:clear`
3. Vérifier les workers/queues (Supervisor/systemd)
   - Redémarrer si nécessaire.

Déploiement (procédure recommandée)

1. Local build
   - `npm ci && npm run build`
2. Préparer `_deploy` (scripts le font automatiquement)
3. Transfert
   - Préférer `rsync` : `rsync -az --delete --exclude vendor --exclude node_modules _deploy/ user@host:/var/www/app`
   - Fallback : utiliser `--allow-scp` avec `scp` mais prévoir temps de transfert plus long.
4. Post-deploy (SSH)
   - `php artisan migrate --force`
   - `php artisan cache:clear && php artisan view:clear`
   - Redémarrer services (php-fpm/nginx, supervisor)

Tests à lancer avant release

- Unitaires : modèles et règles de conflit (locaux/CI).
- Intégration : webhook paiement (staging).
- E2E : parcours réservation complet en staging.
- E2E : scenario annulation/reschedule selon delai (12h/24h).

Automatisation & CI/CD (proposition)

- Ajouter une étape `deploy` dans CI qui : build assets, transfert via rsync, exécute migrations et clear caches via SSH.
- Prévoir variables sécurisées pour l'hôte, utilisateur et clé SSH.

Rollback & sécurité

- Garder backups DB avant migration en prod (snapshot ou export SQL).
- Documenter procédure de rollback (DB + assets).

To-do de suivi

- Documenter procédures d'installation d'`rsync` sur les environnements.

- Ajouter post-deploy automatique (optionnel, à condition d'avoir clés et permissions).
- Valider en staging le script `deploy-full.sh` avec `--allow-scp` si rsync non disponible.
- Ajouter un audit accessibilite plus complet (modales, contrastes, focus).

Mis à jour le : 2026-02-11