

Architecture Decision Records (ADRs)

Système de gestion de parking - Flutter + Spring Boot

ADR-001 : Application Mobile et Web Flutter	2
ADR-002: Clean Architecture + DDD	2
ADR-003: Stack Technique Flutter + Spring Boot	2
ADR-004: Authentification JWT + Spring Security	3
ADR-005: Modular Monolith	3
ADR-006: BLoC Pattern + Repository Pattern	3
ADR-007: l'API externe QR Code	3
ADR-008: Containerisation Docker	4
ADR-009: Stratégie de Tests	4
ADR-010: Architecture Hexagonale (Ports & Adapters)	5
ADR-011: Service QR Code Externe	6

ADR-001 : Application Mobile et Web Flutter

Statut: Accepté

Décision

Justification du choix pour la gestion de parking

Avantages

- QR code natif
- Notifications push
- Performance native
- Cross-platform avec un seul codebase

ADR-002: Clean Architecture + DDD

Statut: Accepté

Décision

L'Adoption de Clean Architecture avec Domain Driven Design sont les premiers choix qui nous avons pensé car c'est ce que nous avons eu en cours

Avantages

- Séparation claire des responsabilités
- Business logic indépendante du framework
- Testabilité et maintenabilité améliorées

ADR-003: Stack Technique Flutter + Spring Boot

Nous avons listé Flutter, [React.js](#), Postgres

Statut: Accepté

Stack retenue

- **Frontend** : Flutter/Dart pour performance native multiplateforme
- **Backend** : Java 17+ Spring Boot 3.x pour robustesse entreprise
- **Base de données** : MySQL

Dépendances Spring Boot principales :

- **Spring Boot DevTools** - Redémarrage automatique en développement
- **Spring Web** - Création d'API REST
- **Spring Security** - Authentification JWT et autorisation
- **Spring Data JPA** - ORM pour communication base de données
- **MySQL Connector/J** - Driver MySQL pour production

ADR-004: Authentification JWT + Spring Security

Statut : Accepté

Solution d'authentification

- Spring Security pour le backend (battle-tested)

ADR-005: Modular Monolith

Statut: Accepté

Architecture

- Single database avec séparation par domaines
- Simplicité opérationnelle vs complexité microservices
- ACID transactions cross-domain

ADR-006: BLoC Pattern + Repository Pattern

Statut: Accepté

Patterns Flutter

- State management prévisible pour Flutter
- Séparation UI/Business Logic
- Architecture testable et réactive

ADR-007: l'API externe QR Code

Statut: Accepté

Décision Implémentation QR codes statiques pour les 60 places de parking (A01-F10)

Solution technique

- QR codes générés via API externe spécialisée
- Format QR : URL vers endpoint /checkin/{placeId}/{reservationToken}
- Service externe appelé pour générer les 60 QR codes (A01-F10)
- Scanner Flutter avec package qr_code_scanner
- Check-in automatique lors du scan
- Job automatique de libération des places à 11h si pas de check-in

Avantages

- UX fluide : scan rapide et confirmation immédiate
- Sécurité : token unique par réservation
- Automatisation : libération automatique des places non utilisées
- Traçabilité : historique complet des check-ins

ADR-008: Containerisation Docker

Statut: Accepté

Décision Containerisation obligatoire avec Docker pour tous les composants

Architecture conteneurs

- Container Spring Boot (backend + API)
- Container MySQL (base de données)
- Container Nginx (proxy pour Flutter web)
- Docker Compose pour orchestration locale

Avantages

- Portabilité : fonctionnement identique sur tous les environnements
- Isolation : pas de conflits de dépendances
- Déploiement simplifié : configuration reproductible
- Contributions facilitées : onboarding équipe rapide
- Conformité projet : containerisation requise par les spécifications

ADR-009: Stratégie de Tests

Statut: Accepté

Décision Implémentation de tests significatifs à tous les niveaux

Tests Backend (Spring Boot)

- Tests unitaires : JUnit 5 + Mockito pour la logique métier
- Tests d'intégration : @SpringBootTest avec TestContainers MySQL
- Tests API : MockMvc pour les endpoints REST
- Tests sécurité : Spring Security Test

Tests Frontend (Flutter)

- Tests unitaires : flutter_test pour la logique business
- Tests widgets : Widget testing pour l'UI
- Tests d'intégration : Integration testing end-to-end
- Tests QR : Mock caméra pour scanner

Avantages

- Qualité : détection précoce des bugs
- Régression : protection contre les régressions
- Documentation vivante : tests comme spécifications
- Confiance : déploiements sécurisés
- TestContainers : tests avec vraie base de données

ADR-010: Architecture Hexagonale (Ports & Adapters)

Statut: Accepté

Décision Adoption de l'architecture hexagonale plutôt qu'une architecture en couches traditionnelle

Structure Hexagonale

Cœur Métier (Domain Core)

- **Entités métier** : User, Reservation, ParkingSpot, Analytics
- **Services métier** : ReservationService, UserService, AnalyticsService
- **Règles business** : Validation réservations, libération automatique 11h, gestion des rôles

Ports (Interfaces)

Ports d'entrée (Primary Ports)

- **ReservationManagementPort** - Gestion des réservations
- **UserManagementPort** - Authentification et profils
- **AnalyticsPort** - Dashboard et statistiques
- **QRCodeValidationPort** - Check-in via QR code

Ports de sortie (Secondary Ports)

- **ReservationRepositoryPort** - Persistance des réservations
- **UserRepositoryPort** - Persistance des utilisateurs
- **EmailNotificationPort** - Envoi d'emails
- **QRCodeGenerationPort** - Génération QR codes externes

Adaptateurs (Adapters)

Adaptateurs d'entrée (Primary Adapters)

- **REST Controllers** : Exposition des APIs pour Flutter
- **Security Filters** : Authentification JWT
- **Scheduled Tasks** : Jobs automatiques (libération 11h)

Adaptateurs de sortie (Secondary Adapters)

- **JPA Repositories** : Persistance MySQL
- **SMTP Adapter** : Service d'envoi d'emails
- **QR Code API Adapter** : Intégration service externe QR
- **Cache Adapter** : Gestion du cache (si nécessaire)

Avantages

Testabilité

- Logique métier isolée des frameworks
- Tests unitaires du domain sans dépendances externes
- Mock facile des adaptateurs pour les tests

Maintenabilité

- Séparation claire des responsabilités
- Changement d'adaptateurs sans impact sur le métier
- Code métier indépendant de Spring Boot

Flexibilité

- Possibilité de changer de base de données sans impact
- Remplacement facile des services externes
- Évolution vers d'autres frameworks si nécessaire

Conformité DDD

- S'aligne parfaitement avec Domain Driven Design (ADR-002)

- Préservation de la logique métier au centre
- Isolation des préoccupations techniques

ADR-011: Service QR Code Externe

Statut: Accepté

Décision: Utilisation d'une API externe pour génération QR codes

Justification:

- Séparation des responsabilités
- Service spécialisé plus fiable
- Pas de dépendance dans le code métier
- Possibilité de changer de provider facilement