ParkingSpots - Technical Documentation

Version: 1.0.0
Date: February 9, 2026
Author: Development Team

Table of Contents

---

1. Executive Summary

1.1 Project Overview

ParkingSpots is a peer-to-peer parking rental marketplace that connects parking space owners with drivers seeking convenient parking solutions. The platform enables property owners to monetize their unused parking spaces while providing users with a seamless way to find, book, and pay for parking.

1.2 Key Features Implemented

Feature	Description	Status
User Authentication	JWT-based auth with role management	[YES] Complete
Location-based Search	Haversine distance calculation for nearby spots	[YES] Complete
Booking System	Full booking lifecycle management	[YES] Complete
Payment Processing	Stripe integration for payments & payouts	[YES] Complete
Reviews & Ratings	5-star rating system with owner responses	[YES] Complete
Real-time Availability	WebSocket/Redis infrastructure	[YES] Complete

1.3 Technology Decisions

Component	Technology	Rationale
Backend Framework	FastAPI	Async support, auto-documentation, high performance
Database	PostgreSQL	ACID compliance, PostGIS compatibility, JSON support
ORM	SQLAlchemy 2.0	Async support, mature ecosystem

Component	Technology	Rationale
Mobile Framework	React Native + Expo	Cross-platform, native performance
State Management	Zustand	Lightweight, TypeScript-first
Payments	Stripe Connect	Marketplace support, global coverage

---

2. System Architecture

2.1 High-Level Architecture

```
+----------------------------------------------------------------------+
|                         CLIENTS                                  |
|----------------------------------------------------------------------|
|  +-------------------+  +------------------+  +--------------------------+   |
|  |  iOS App     |  | Android App  |  |   Admin Dashboard    |   |
|  | (React Native|  |(React Native)|  |      (Future)        |   |
|  \---------+----------+  \---------+----------+  \-------------+--------------+   |
\-----------+-----------------+---------------------+-----------------+
          |                 |                     |
          \-----------------+--+----------------------+
                           | HTTPS/WSS
                           v
+----------------------------------------------------------------------+
|                      API GATEWAY / LOAD BALANCER                 |
\----------------------------------------------------------------------+
                           |
                           v
+----------------------------------------------------------------------+
|                      BACKEND SERVICES                            |
|----------------------------------------------------------------------|
|  +-------------------------------------------------------------+   |
|  |                   FastAPI Application                    |   |
|  |  +-------------+ +-------------+ +-------------+ +---------------------+ |   |
|  |  |  Auth    | | Parking | |Bookings | |    Payments     | |   |
|  |  | Module   | |  Spots  | | Module  | |     Module      | |   |
|  |  \-------------+ \-------------+ \-------------+ \---------------------+ |   |
|  |  +-------------+ +-------------+ +---------------------------------+ |   |
|  |  |  Users   | | Reviews | |   WebSocket Handlers       | |   |
|  |  | Module   | | Module  | |   (Real-time Updates)      | |   |
|  |  \-------------+ \-------------+ \---------------------------------+ |   |
|  \-------------------------------------------------------------+   |
\----------------------------------------------------------------------+
          |                 |                     |
          v                 v                     v
+---------------------+ +---------------------+ +----------------------------+
|   PostgreSQL    | |      Redis      | |    External Services    |
|   (Primary DB)  | |  (Cache/PubSub) | |  +-------------+ +-----------+  |
|                 | |                 | |  | Stripe  | |  AWS  |  |
|                 | |                 | |  |   API   | |  S3   |  |
\---------------------+ \---------------------+ \-----+-----------+---+--------+-----+
```

2.2 Data Flow

```
User Action → Mobile App → API Request → FastAPI Router →
Service Layer → Database → Response → Mobile App → UI Update
```

2.3 Directory Structure

```
ParkingSpots/
|---- backend/
|   |---- app/
|   |   |---- __init__.py
|   |   |---- main.py                 # Application entry point
|   |   |---- api/
|   |   |   |---- __init__.py
|   |   |   |---- deps.py             # Dependency injection
|   |   |   \---- v1/
|   |   |       |---- __init__.py
|   |   |       |---- router.py       # API router aggregation
|   |   |       \---- endpoints/
|   |   |           |---- auth.py     # Authentication endpoints
|   |   |           |---- users.py    # User management
|   |   |           |---- parking_spots.py
|   |   |           |---- bookings.py
|   |   |           |---- reviews.py
|   |   |           \---- payments.py
|   |   |---- core/
|   |   |   |---- __init__.py
|   |   |   |---- config.py           # Settings management
|   |   |   \---- security.py         # JWT & password hashing
|   |   |---- db/
|   |   |   |---- __init__.py
|   |   |   |---- base.py             # Base model class
|   |   |   \---- session.py          # Database session
|   |   |---- models/                 # SQLAlchemy models
|   |   |   |---- user.py
|   |   |   |---- parking_spot.py
|   |   |   |---- booking.py
|   |   |   |---- review.py
|   |   |   \---- payment.py
|   |   \---- schemas/                # Pydantic schemas
|   |       |---- user.py
|   |       |---- parking_spot.py
|   |       |---- booking.py
|   |       |---- review.py
|   |       \---- payment.py
|   |---- requirements.txt
|   |---- .env.example
|   \---- README.md
|
|---- mobile/
|   |---- src/
|   |   |---- components/             # Reusable UI components
|   |   |---- navigation/
|   |   |   |---- index.ts
|   |   |   \---- AppNavigator.tsx    # Navigation configuration
|   |   |---- screens/
|   |   |   |---- auth/
|   |   |   |   |---- LoginScreen.tsx
|   |   |   |   \---- RegisterScreen.tsx
|   |   |   |---- home/
|   |   |   |   \---- HomeScreen.tsx
|   |   |   |---- parking/
|   |   |   |   \---- ParkingSpotDetailScreen.tsx
|   |   |   |---- bookings/
|   |   |   |   \---- BookingsScreen.tsx
|   |   |   \---- profile/
|   |   |       \---- ProfileScreen.tsx
|   |   |---- services/              # API service layer
|   |   |   |---- api.ts             # Axios client
|   |   |   |---- auth.ts
|   |   |   |---- parkingSpot.ts
|   |   |   |---- booking.ts
|   |   |   |---- review.ts
|   |   |   \---- payment.ts
|   |   |---- stores/                # Zustand state stores
|   |   |   |---- authStore.ts
|   |   |   |---- parkingSpotStore.ts
|   |   |   \---- bookingStore.ts
|   |   \---- types/                 # TypeScript definitions
|   |       |---- user.ts
|   |       |---- parkingSpot.ts
|   |       |---- booking.ts
|   |       \---- review.ts
|   |---- App.tsx                    # Application root
|   |---- app.json                   # Expo configuration
|   |---- package.json
|   \---- tsconfig.json
|
\---- README.md                      # Project documentation
```

---

3. Backend Implementation

3.1 FastAPI Application Structure

Main Application (`app/main.py`)

```
# Key components:
- Lifespan context manager for startup/shutdown
- CORS middleware configuration
- API router inclusion
- Health check endpoints
```

Features: - Async database table creation on startup - Graceful shutdown with connection cleanup - Configurable CORS for mobile app access

Configuration (`app/core/config.py`) Uses Pydantic Settings for type-safe configuration:

Setting	Type	Purpose
`DATABASE_URL`	str	PostgreSQL connection string
`SECRET_KEY`	str	JWT signing key
`ACCESS_TOKEN_EXPIRE_MINUTES`	int	Token validity (default: 30)
`REFRESH_TOKEN_EXPIRE_DAYS`	int	Refresh token validity (default: 7)
`STRIPE_SECRET_KEY`	str	Stripe API authentication
`REDIS_URL`	str	Redis connection for caching

3.2 Database Layer

Session Management (`app/db/session.py`)

```
# Async SQLAlchemy engine with:
- Connection pooling
- Automatic session cleanup
- Transaction management via dependency injection
```

Base Model (`app/db/base.py`) Provides: - `Base` - SQLAlchemy declarative base - `TimestampMixin` - Automatic `created_at`/`updated_at` columns

3.3 API Endpoints Summary

Module	Endpoints	Purpose
Auth	4	Registration, login, token refresh, password reset
Users	5	Profile CRUD, password change
Parking Spots	8	Listing CRUD, search, availability management
Bookings	8	Booking CRUD, pricing, check-in/out
Reviews	7	Review CRUD, summaries, helpful votes
Payments	8	Payment intents, confirmations, refunds, payouts

---

4. Mobile Application

4.1 Technology Stack

Package	Version	Purpose
`expo`	~50.0.6	Development platform
`react-native`	0.73.2	Core framework
`@react-navigation/native`	^6.1.9	Navigation
`react-native-maps`	1.10.0	Map integration
`zustand`	^4.5.0	State management
`axios`	^1.6.5	HTTP client
`@stripe/stripe-react-native`	^0.35.1	Payment UI

4.2 State Management Architecture

Using Zustand for lightweight, TypeScript-first state management:

Auth Store (`stores/authStore.ts`)

```
interface AuthState {
  user: User | null;
  isAuthenticated: boolean;
  isLoading: boolean;
  error: string | null;

  // Actions
  login: (email, password) => Promise<void>;
  register: (data) => Promise<void>;
  logout: () => Promise<void>;
  fetchUser: () => Promise<void>;
  checkAuth: () => Promise<boolean>;
}
```

Parking Spot Store (`stores/parkingSpotStore.ts`)

```
interface ParkingSpotState {
  searchResults: ParkingSpotListItem[];
  currentLocation: Location | null;
  selectedSpot: ParkingSpot | null;
  mySpots: ParkingSpot[];
  searchFilters: Partial<ParkingSpotSearch>;

  // Actions
  searchNearby: (params) => Promise<void>;
  getSpotDetails: (id) => Promise<ParkingSpot>;
  createSpot: (data) => Promise<ParkingSpot>;
}
```

4.3 API Service Layer

The service layer (`src/services/`) provides:

1. **Centralized API Client** - Axios instance with interceptors
2. **Automatic Token Refresh** - Transparent retry on 401
3. **Type-safe Responses** - Generic TypeScript return types
4. **Error Handling** - Consistent error transformation

Token Refresh Flow:

```
Request fails (401) → Check if refreshing →
Queue request → Refresh token →
Retry queued requests → Return responses
```

4.4 Navigation Structure

```
Root Navigator
|---- Auth Stack (unauthenticated)
|    |---- Login Screen
|    \---- Register Screen
|
\---- Main Tabs (authenticated)
     |---- Home Tab
     |    |---- Home Screen (Map)
```

```
    |   |---- Search Screen
    |   \---- Parking Spot Detail
    |
    |---- Bookings Tab
    |   |---- Bookings List
    |   \---- Booking Detail
    |
    \---- Profile Tab
        |---- Profile Screen
        |---- My Spots (owners)
        |---- Settings
        \---- Payment Methods
```

---

5. Database Design

5.1 Entity Relationship Diagram

```
+------------------+       +----------------------+       +------------------+
|    USERS     |       |  PARKING_SPOTS  |       |   BOOKINGS  |
|------------------|       |----------------------|       |------------------|
| id (PK)      |----+    | id (PK)         |----+    | id (PK)     |
| email        |  |     | owner_id (FK)---|----+    | user_id(FK) |----+
| password     |  |     | title           |       | spot_id(FK) |----|
| full_name    |  |     | description     |       | start_time  |  |
| phone        |  |     | latitude        |       | end_time    |  |
| role         |  |     | longitude       |       | status      |  |
| stripe_id    |  |     | hourly_rate     |       | total       |  |
| created_at   |  |     | daily_rate      |<------| created_at  |  |
\------------------+  |    | is_available    |       \------------------+  |
                 |    | average_rating  |                   |
                 |    | created_at      |                   |
                 |    \----------------------+                       |
                 |                                        |
                 |    +----------------------+       +------------------+  |
                 |    |    REVIEWS      |       |  PAYMENTS   |  |
                 |    |----------------------|       |------------------|  |
                 |    | id (PK)         |       | id (PK)     |  |
                 \----->| reviewer_id(FK) |       | booking_id  |<---|
                      | spot_id (FK)    |       | user_id(FK) |----+
                      | booking_id (FK) |       | amount      |
                      | overall_rating  |       | status      |
                      | comment         |       | stripe_id   |
                      | owner_response  |       | created_at  |
                      | created_at      |       \------------------+
                      \----------------------+
```

5.2 Model Specifications

Users Model

Column	Type	Constraints	Description
`id`	UUID	PK, default uuid4	Unique identifier
`email`	VARCHAR(255)	UNIQUE, NOT NULL, INDEX	User email

Column	Type	Constraints	Description
`hashed_password`	VARCHAR(255)	NOT NULL	Bcrypt hash
`full_name`	VARCHAR(255)	NOT NULL	Display name
`phone_number`	VARCHAR(20)	NULLABLE	Contact number
`role`	ENUM	NOT NULL, default 'renter'	owner/renter/admin
`is_active`	BOOLEAN	default TRUE	Account status
`is_verified`	BOOLEAN	default FALSE	Email verification
`stripe_customer_id`	VARCHAR(255)	NULLABLE	Stripe reference
`latitude`	FLOAT	NULLABLE	User location
`longitude`	FLOAT	NULLABLE	User location
`created_at`	TIMESTAMP	NOT NULL	Record creation
`updated_at`	TIMESTAMP	NOT NULL	Last modification

Parking Spots Model

Column	Type	Constraints	Description
`id`	UUID	PK	Unique identifier
`owner_id`	UUID	FK → users.id	Spot owner
`title`	VARCHAR(255)	NOT NULL	Listing title
`description`	TEXT	NULLABLE	Detailed description
`spot_type`	ENUM	NOT NULL	indoor/outdoor/covered/garage/driveway/lot
`vehicle_size`	ENUM	NOT NULL	motorcycle/compact/standard/large/oversized
`address`	VARCHAR(500)	NOT NULL	Street address
`city`	VARCHAR(100)	NOT NULL	City name
`state`	VARCHAR(100)	NOT NULL	State/province
`zip_code`	VARCHAR(20)	NOT NULL	Postal code
`latitude`	FLOAT	NOT NULL	GPS latitude
`longitude`	FLOAT	NOT NULL	GPS longitude
`hourly_rate`	INTEGER	NOT NULL	Price in cents
`daily_rate`	INTEGER	NULLABLE	Daily price (cents)
`monthly_rate`	INTEGER	NULLABLE	Monthly price (cents)
`is_covered`	BOOLEAN	default FALSE	Has cover
`has_ev_charging`	BOOLEAN	default FALSE	EV capable
`has_security`	BOOLEAN	default FALSE	Security camera/guard
`has_lighting`	BOOLEAN	default FALSE	Well lit
`is_handicap_accessible`	BOOLEAN	default FALSE	ADA accessible
`images`	ARRAY[VARCHAR]	default []	Image URLs
`is_active`	BOOLEAN	default TRUE	Listing active
`is_available`	BOOLEAN	default TRUE	Currently available
`operating_hours`	JSON	NULLABLE	Weekly schedule
`access_instructions`	TEXT	NULLABLE	How to access
`average_rating`	FLOAT	default 0.0	Computed rating
`total_reviews`	INTEGER	default 0	Review count
`total_bookings`	INTEGER	default 0	Booking count

Bookings Model

Column	Type	Constraints	Description
`id`	UUID	PK	Unique identifier
`user_id`	UUID	FK → users.id	Booking user

Column	Type	Constraints	Description
`parking_spot_id`	UUID	FK → parking_spots.id	Reserved spot
`start_time`	TIMESTAMP	NOT NULL	Reservation start
`end_time`	TIMESTAMP	NOT NULL	Reservation end
`status`	ENUM	NOT NULL	pending/confirmed/in_progress/completed/cancel
`total_amount`	INTEGER	NOT NULL	Total in cents
`service_fee`	INTEGER	default 0	Platform fee (cents)
`owner_payout`	INTEGER	default 0	Owner earnings (cents)
`payment_intent_id`	VARCHAR(255)	NULLABLE	Stripe reference
`payment_status`	VARCHAR(50)	default 'pending'	Payment state
`vehicle_plate`	VARCHAR(20)	NULLABLE	License plate
`vehicle_make`	VARCHAR(50)	NULLABLE	Car manufacturer
`vehicle_model`	VARCHAR(50)	NULLABLE	Car model
`vehicle_color`	VARCHAR(30)	NULLABLE	Car color
`special_requests`	TEXT	NULLABLE	User notes
`cancellation_reason`	TEXT	NULLABLE	Why cancelled
`checked_in_at`	TIMESTAMP	NULLABLE	Actual arrival
`checked_out_at`	TIMESTAMP	NULLABLE	Actual departure

5.3 Indexes

```sql
-- Performance indexes
CREATE INDEX idx_users_email ON users(email);
CREATE INDEX idx_parking_spots_owner ON parking_spots(owner_id);
CREATE INDEX idx_parking_spots_location ON parking_spots(latitude, longitude);
CREATE INDEX idx_parking_spots_available ON parking_spots(is_active, is_available);
CREATE INDEX idx_bookings_user ON bookings(user_id);
CREATE INDEX idx_bookings_spot ON bookings(parking_spot_id);
CREATE INDEX idx_bookings_status ON bookings(status);
CREATE INDEX idx_reviews_spot ON reviews(parking_spot_id);
```

---

6. API Specification

6.1 Authentication Endpoints

POST `/api/v1/auth/register` Request:

```json
{
  "email": "user@example.com",
  "password": "securePassword123",
  "full_name": "John Doe",
  "phone_number": "+1234567890",
  "role": "renter"
}
```

Response (201):

```json
{
  "id": "uuid",
  "email": "user@example.com",
  "full_name": "John Doe",
  "role": "renter",
  "is_active": true,
  "is_verified": false,
  "created_at": "2026-02-09T12:00:00Z"
}
```

POST `/api/v1/auth/login` Request:

```
{
  "email": "user@example.com",
  "password": "securePassword123"
}
```

Response (200):

```
{
  "access_token": "eyJhbGciOiJIUzI1NiIs...",
  "refresh_token": "eyJhbGciOiJIUzI1NiIs...",
  "token_type": "bearer"
}
```

6.2 Parking Spots Endpoints

GET `/api/v1/parking-spots` Query Parameters: | Parameter | Type | Description | |———–|——|————-| | `latitude` | float | Search center latitude | | `longitude` | float | Search center longitude | | `radius_km` | float | Search radius (default: 10) | | `spot_type` | string | Filter by type | | `max_hourly_rate` | int | Maximum price filter | | `has_ev_charging` | bool | EV filter | | `page` | int | Pagination page | | `page_size` | int | Results per page |

Response (200):

```
[
  {
    "id": "uuid",
    "title": "Downtown Parking Spot",
    "address": "123 Main St",
    "city": "New York",
    "state": "NY",
    "latitude": 40.7128,
    "longitude": -74.0060,
    "hourly_rate": 500,
    "spot_type": "garage",
    "is_available": true,
    "average_rating": 4.5,
    "total_reviews": 23,
    "images": ["https://..."],
    "distance_km": 0.5
  }
]
```

6.3 Booking Endpoints

POST `/api/v1/bookings/calculate-price` Request:

```
{
  "parking_spot_id": "uuid",
  "start_time": "2026-02-10T09:00:00Z",
  "end_time": "2026-02-10T17:00:00Z"
}
```

Response (200):

```
{
  "subtotal": 4000,
  "service_fee": 400,
  "total": 4400,
  "owner_payout": 4000,
  "duration_hours": 8.0
}
```

POST `/api/v1/bookings` **Request:**

```
{
  "parking_spot_id": "uuid",
  "start_time": "2026-02-10T09:00:00Z",
  "end_time": "2026-02-10T17:00:00Z",
  "vehicle_plate": "ABC123",
  "vehicle_make": "Toyota",
  "vehicle_model": "Camry",
  "vehicle_color": "Silver"
}
```

Response (201):

```
{
  "id": "uuid",
  "user_id": "uuid",
  "parking_spot_id": "uuid",
  "start_time": "2026-02-10T09:00:00Z",
  "end_time": "2026-02-10T17:00:00Z",
  "status": "pending",
  "total_amount": 4400,
  "service_fee": 400,
  "payment_status": "pending",
  "created_at": "2026-02-09T12:00:00Z"
}
```

6.4 Error Response Format

```
{
  "detail": "Error message describing what went wrong"
}
```

HTTP Status Codes:

Code	Meaning
200	Success
201	Created
400	Bad Request
401	Unauthorized
403	Forbidden
404	Not Found
409	Conflict
422	Validation Error
500	Server Error

---

7. Authentication & Security

7.1 JWT Token Structure

Access Token Payload:

```
{
  "sub": "user-uuid",
  "exp": 1707480000,
  "type": "access"
}
```

Refresh Token Payload:

```json
{
  "sub": "user-uuid",
  "exp": 1708084800,
  "type": "refresh"
}
```

7.2 Password Security

- **Algorithm:** Bcrypt
- **Work Factor:** Default (12 rounds)
- **Minimum Length:** 8 characters

7.3 Authorization Flow

```
+-----------------------------------------------------------------+
|                     Authorization Flow                       |
|-----------------------------------------------------------------|
|                                                             |
|  Client                    Server                           |
|    |                         |                              |
|    |  POST /auth/login       |                              |
|    |  {email, password}      |                              |
|    | ----------------------->|                              |
|    |                         | Verify credentials           |
|    |                         | Generate tokens              |
|    |  {access, refresh}      |                              |
|    | <-----------------------|                              |
|    |                         |                              |
|    |  GET /users/me          |                              |
|    |  Authorization: Bearer  |                              |
|    | ----------------------->|                              |
|    |                         | Validate token               |
|    |                         | Extract user_id              |
|    |  {user data}            |                              |
|    | <-----------------------|                              |
|    |                         |                              |
|    |  (Token expired)        |                              |
|    |  POST /auth/refresh     |                              |
|    |  {refresh_token}        |                              |
|    | ----------------------->|                              |
|    |                         | Validate refresh token       |
|    |  {new access, refresh}  |                              |
|    | <-----------------------|                              |
|                                                             |
\-----------------------------------------------------------------+
```

7.4 Role-Based Access Control

Role	Permissions
`renter`	Search spots, create bookings, write reviews
`owner`	All renter + create spots, manage bookings, respond to reviews
`admin`	All owner + user management, system configuration

8. Payment Integration

8.1 Stripe Integration Overview

```
+------------------------------------------------------------------+
|                    Payment Flow                               |
|------------------------------------------------------------------|
|                                                              |
|  Mobile App              Backend               Stripe        |
|      |                      |                    |           |
|      | Create Booking       |                    |           |
|      |--------------------->|                    |           |
|      |                      | Create Payment     |           |
|      |                      | Intent             |           |
|      |                      |------------------->|           |
|      |                      |                    |           |
|      |                      | client_secret      |           |
|      | client_secret        |<-------------------|           |
|      |<---------------------|                    |           |
|      |                      |                    |           |
|      | confirmPayment()     |                    |           |
|      |----------------------------------------->|           |
|      |                      |                    |           |
|      |                      |   Webhook:         |           |
|      |                      |   payment_intent.  |           |
|      |                      |   succeeded        |           |
|      |                      |<-------------------|           |
|      |                      |                    |           |
|      |                      | Update booking     |           |
|      |                      | status             |           |
|      |                      |                    |           |
|      | Booking confirmed    |                    |           |
|      |<---------------------|                    |           |
|                                                              |
\------------------------------------------------------------------+
```

8.2 Fee Structure

Component	Amount	Description
Subtotal	100%	Hourly rate × hours
Service Fee	10% + $0.50	Platform commission + transaction fee
Total	110% + $0.50	User pays this amount
Owner Payout	90% - $0.50	Subtotal minus service fee and Stripe fees

Example: - Parking spot rate: $40 for 8 hours - Service fee: (10% × $40) + $0.50 = $4.00 + $0.50 = $4.50 - Total charged to user: $44.50 - Owner receives: $40.00 - Stripe fees (~2.9% + $0.30)

8.3 Stripe Connect for Owners

Owners receive payouts via Stripe Connect Express accounts:

1. Owner initiates onboarding

2. Backend creates Express account
3. Owner completes Stripe onboarding
4. Payouts automatically deposited

---

9. Real-time Features

9.1 Architecture

```
+-----------------------------------------------------------------+
|                Real-time System Architecture                 |
|-----------------------------------------------------------------|
|                                                             |
|  +--------------+    +--------------+    +-------------------------+   |
|  | Client  |---->| FastAPI |---->|  Redis Pub/Sub      |   |
|  |   WS    |<----|   WS    |<----|                     |   |
|  \--------------+    \--------------+    \-------------------------+   |
|                                              |              |
|                                              v              |
|                                  +-------------------------+   |
|                                  |   Event Types:      |   |
|                                  |   - availability    |   |
|                                  |   - booking_update  |   |
|                                  |   - new_review      |   |
|                                  \-------------------------+   |
|                                                             |
\-----------------------------------------------------------------+
```

9.2 Event Types

Event	Payload	Trigger
`spot_availability`	`{spot_id, is_available}`	Availability toggle
`booking_update`	`{booking_id, status}`	Status change
`new_booking`	`{booking_id, spot_id}`	New reservation
`new_review`	`{review_id, spot_id, rating}`	Review posted

---

10. Deployment Guide

10.1 Backend Deployment

Docker Configuration

```dockerfile
FROM python:3.11-slim

WORKDIR /app

# Install dependencies
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

# Copy application
COPY . .
```

```
# Run with Gunicorn + Uvicorn workers
CMD ["gunicorn", "app.main:app", "-w", "4", "-k", "uvicorn.workers.UvicornWorker", "-b", "0.0.0.0:8000"
```

Docker Compose

```yaml
version: '3.8'

services:
  api:
    build: ./backend
    ports:
      - "8000:8000"
    environment:
      - DATABASE_URL=postgresql+asyncpg://postgres:password@db:5432/parkingspots
      - REDIS_URL=redis://redis:6379
    depends_on:
      - db
      - redis

  db:
    image: postgres:15
    volumes:
      - postgres_data:/var/lib/postgresql/data
    environment:
      - POSTGRES_DB=parkingspots
      - POSTGRES_PASSWORD=password

  redis:
    image: redis:7-alpine
    volumes:
      - redis_data:/data

volumes:
  postgres_data:
  redis_data:
```

10.2 Mobile App Deployment

EAS Build Configuration

```json
{
  "cli": {
    "version": ">= 5.0.0"
  },
  "build": {
    "development": {
      "developmentClient": true,
      "distribution": "internal"
    },
    "preview": {
      "distribution": "internal"
    },
    "production": {}
  },
  "submit": {
    "production": {}
  }
}
```

Build Commands

```
# Install EAS CLI
npm install -g eas-cli

# Configure project
eas build:configure

# Build for iOS
eas build --platform ios --profile production

# Build for Android
eas build --platform android --profile production

# Submit to stores
eas submit --platform ios
eas submit --platform android
```

10.3 Environment Checklist

- ☐ PostgreSQL database created
- ☐ Redis instance running
- ☐ Environment variables configured
- ☐ Stripe account set up with webhook
- ☐ AWS S3 bucket for images
- ☐ SSL certificates configured
- ☐ DNS records pointing to servers
- ☐ Monitoring and logging enabled

Appendix A: API Response Codes

Code	Meaning	Common Causes
200	OK	Successful request
201	Created	Resource created
400	Bad Request	Invalid input
401	Unauthorized	Missing/invalid token
403	Forbidden	Insufficient permissions
404	Not Found	Resource doesn't exist
409	Conflict	Duplicate or conflicting state
422	Unprocessable	Validation failed
500	Server Error	Internal error

Appendix B: Location Search Algorithm

The Haversine formula is used for distance calculations:

```python
def haversine(lon1, lat1, lon2, lat2):
    """Calculate great circle distance in kilometers."""
    lon1, lat1, lon2, lat2 = map(radians, [lon1, lat1, lon2, lat2])
    dlon = lon2 - lon1
    dlat = lat2 - lat1
    a = sin(dlat/2)**2 + cos(lat1) * cos(lat2) * sin(dlon/2)**2
    c = 2 * asin(sqrt(a))
    r = 6371  # Earth radius in km
    return c * r
```

Document End

For questions or support, contact the development team.