HireSpace - Backend Requirements Document

Created by	Saif Ashraf
Category	

Version: 1.0

Product: Workspace booking platform backend

Status: Planning

Overview

HireSpace backend provides APIs for a workspace booking platform. Users can search for spaces, book them with a 15-minute hold system, and pay online. Space owners can list their spaces and manage bookings.

Backend Success Metrics

- **Zero double-bookings** (core system requirement)
- **15-minute hold system** works reliably without manual intervention
- **>90% booking completion rate** (holds that convert to confirmed bookings)
- **<200ms API response times** for search queries
- **Payment-booking consistency** (no orphaned payments or bookings)

User Types & Backend Needs

Freelancer (Sarah)

Backend Requirements: Fast search APIs, reliable booking system, payment processing

Space Owner (Marcus)

Backend Requirements: Space management APIs, booking notifications, dashboard data

Small Team (Alex)

Backend Requirements: Multi-user booking support, transparent pricing calculations

---

Core Backend Features by Phase

Phase 1: Basic Booking System

Goal: Build core booking engine

- User authentication (JWT-based)
- Space CRUD operations with photo uploads
- Search spaces by location, date, capacity
- 15-minute booking hold system with auto-expiration
- Stripe payment integration
- Email notifications (booking confirmations)
- Basic space owner management endpoints

Phase 2: Enhanced Operations

Goal: Improve search and business logic

- Advanced search filters (amenities, price range)
- Geolocation-based space discovery
- Calendar availability views
- User booking history and profiles
- Messaging system between users and owners
- Review system (post-booking ratings)

Phase 3: Scale & Polish

Goal: Production-ready operations

- Owner analytics (bookings, revenue)
- Customer support tools and admin panel
- Refund and cancellation handling
- Background job processing for cleanup
- Comprehensive logging and monitoring

Critical Backend Challenges

1. Preventing Double-Bookings

Problem: Multiple users trying to book the same time slot

Approach: Database constraints + transaction locking + proper error handling

2. Hold System Timing

Problem: 15-minute holds must expire automatically

Approach: Background jobs + database triggers + cleanup processes

3. Payment-Booking Consistency

Problem: Payment succeeds but booking fails (or vice versa)

Approach: Stripe webhooks + proper transaction rollback

4. Real-time Availability

Problem: Fast availability checking across date ranges

Approach: Efficient database queries + caching for popular spaces

---

Key Backend Flows

Booking Process

1. **Hold Creation:** Create temporary booking record with 15-min expiration
2. **Payment Processing:** Stripe payment with booking metadata
3. **Confirmation:** Convert hold to confirmed booking on payment success
4. **Cleanup:** Auto-expire holds that aren't paid

Space Management

1. **Listing Creation:** Upload photos to S3, store metadata in database
2. **Availability Management:** Owner sets available time slots
3. **Booking Notifications:** Real-time alerts when space is booked

---

Technical Stack

- **Framework:** NestJS + TypeScript
- **Database:** PostgreSQL with Prisma ORM
- **Payments:** Stripe API integration
- **File Storage:** AWS S3 for space photos
- **Email:** SendGrid for notifications
- **Background Jobs:** Bull Queue + Redis

Open Questions

Business Logic

- **Pricing model:** How do we calculate and collect platform fees?
- **Cancellation policy:** How far in advance? Refund percentages?
- **Verification:** How to ensure space photos/descriptions are accurate?

Technical Decisions

- **Search technology:** PostgreSQL queries vs Elasticsearch for complex search?
- **Real-time features:** WebSockets for live availability updates?
- **File uploads:** Direct S3 upload vs server proxy?
- **Geographic scope:** Single timezone initially or multi-timezone support?

Success Criteria Per Phase

Phase 1: End-to-end booking works without conflicts

Phase 2: Users can easily discover and book spaces they need

Phase 3: System handles production load and edge cases reliably

Assumptions

- Credit card payments only initially
- English language support only
- Web-only (no mobile app APIs initially)
- Single currency support (USD)
- Space owners comfortable with digital payment processing