Vecto Pilot™ - Architecture V2 Reference

Vecto Pilot™ - Architecture V2 Reference
 Table of Contents
 Vision & Mission
Vision Statement
Mission Statement
Business Model
 Core Principles & Non-Negotiables
1. **Accuracy Before Expense**
2. **Zero Hardcoding Policy**
3. **Single-Path Triad (No Fallbacks in Production Flow)**
4. **Database-Driven Reconciliation**
5. **Complete Snapshot Gating**
6. **Deterministic Logging for ML**
 System Architecture
Multi-Server Design
Technology Stack
 Project File Structure
Complete Directory Layout
Key File Descriptions
 Environment Variables Reference
Complete .env Configuration
Environment Variable Validation
 API Endpoint Reference
Base URL
Authentication
Common Headers
1. Location Resolution
2. Recommendations (Triad Pipeline)
3. User Actions & Tracking
4. Feedback
5. Health & Monitoring
⚛ Frontend Architecture
React Component Hierarchy
State Management Architecture
Routing Architecture (Wouter)
Form Handling Pattern
Styling System
 Data Flow & Mapping
1. Location Resolution Flow
2. Strategy Generation Flow (Background Job)
3. Smart Blocks Recommendation Flow (Triad Pipeline)
4. User Action Logging Flow
 AI Model Strategy & Risk Management
Current Production Models
Model Parameters (Environment-Driven)
Critical Model Risks

Model Monitoring

 External API Constraints
Rate Limits & Quotas

Cost Optimization Strategies

API Error Handling Patterns

 Database Schema & Field Mapping
Table Relationships

Complete Field Mapping

 Testing Strategy
1. **Unit Testing** (Future)

2. **Integration Testing** (Current)

3. **End-to-End Testing** (Current)

4. **Performance Testing** (Future)

5. **Model Verification Testing** (Monthly)

6. **Database Migration Testing**

⚠ Known Risks & Mitigations
RISK MATRIX

DETAILED RISK ANALYSIS

 AI Development Guardrails
Rules for AI-Assisted Development

 Quick Start Implementation Guides
Guide 1: Adding a New API Endpoint

Guide 2: Adding a New Database Field

Guide 3: Modifying the Triad Pipeline

Guide 4: Adding a New React Page

Guide 5: Debugging Common Issues

 Deployment & Infrastructure
Replit Autoscale Deployment

Database Configuration (Neon Serverless)

Performance Optimization

Monitoring & Logging

Security Checklist

Scaling Considerations

 Decision Log

 Lessons Learned for AI-Assisted App Development
LESSON #1: Documentation Is Code

LESSON #2: Trust But Verify AI Suggestions

LESSON #3: Model IDs Expire

LESSON #4: Database Schema Is ML Contract

LESSON #5: Environment Variables Over Hardcoded Timeouts

 Future Enhancements (Roadmap)
Phase 1: Production Hardening (Q4 2025)

Phase 2: ML Model Training (Q1 2026)

Phase 3: Multi-Market Expansion (Q2 2026)

Phase 4: Mobile Apps (Q3 2026)

 Related Documentation

 Contributing Guidelines
For Human Developers

For AI Assistants

Vecto Pilot™ - Architecture V2 Reference

Comprehensive System Design & AI Development Blueprint

Last Updated: 2025-10-13
Version: 2.1 (Enhanced for Model-to-Model Replication)
Status: Production Architecture with AI Development Guardrails + Complete Implementation Guides
Purpose: Single source of truth for system design, AI-assisted development patterns, production readiness validation, and autonomous feature development

Table of Contents

1. Vision & Mission
2. Core Principles & Non-Negotiables
3. System Architecture
4. Project File Structure
5. Environment Variables Reference
6. API Endpoint Reference
7. Frontend Architecture
8. Data Flow & Mapping
9. AI Model Strategy & Risk Management
10. Database Schema & Field Mapping
11. External API Constraints
12. Testing Strategy
13. Known Risks & Mitigations
14. AI Development Guardrails
15. Quick Start Implementation Guides
16. Deployment & Infrastructure
17. Decision Log

Vision & Mission

Vision Statement

“Empower rideshare drivers with AI-powered strategic positioning intelligence that eliminates guesswork and maximizes earnings through data-driven recommendations.”

Mission Statement

Drivers don’t lose money because they can’t drive. They lose it in the gaps—time with no passenger, miles with no rider, and opaque pricing that shifts under their feet. In big markets, as much as 40% of rideshail miles are “deadhead” miles between trips, which drags down earnings even when the per-trip payout looks decent.

This app solves that problem by removing guesswork and grounding every recommendation in verified coordinates and business hours from Google Places—not model hallucinations. It computes distance and earnings-per-mile from the server side with real navigation distance, not rough client math.

Core Value Propositions: 1. **Higher Utilization** - Reduce deadhead miles through strategic positioning 2. **Verified Data** - No hallucinated venues; all locations validated via Google Places API 3. **Real-Time Intelligence** - Traffic-aware routing, airport

delays, weather integration 4. **ML-Ready Capture** - Every recommendation tracked for counterfactual learning 5. **Safety First** - Reduce fatigue from aimless driving, get home faster with fewer total miles

Business Model

- **Driver-First SaaS** - Subscription model ($9.99/month or $89.99/year)
- **No Commission Cuts** - We don't touch driver earnings
- **Privacy-Preserving** - Anonymous usage tracking, no PII required
- **Platform Agnostic** - Works with Uber, Lyft, any rideshare service

---

Core Principles & Non-Negotiables

1. Accuracy Before Expense

Cost matters but cannot override correctness for drivers. When tension exists, we resolve in favor of accuracy and transparent failure.

Example: If Google Places API quota is exhausted, we fail-closed with clear error message rather than using stale/cached data.

2. Zero Hardcoding Policy

All location data, venue information, and model configurations MUST come from: - **Database** (PostgreSQL) - Venue catalog, metrics, feedback - **Environment Variables** (.env) - Model names, API endpoints, timeouts - **External APIs** (Google Maps, Weather, FAA) - Real-time context

Forbidden:

```
// ❌ NEVER DO THIS
if (city === "Dallas") { ... }
const model = "gpt-4"; // Hardcoded model name

// ✅ ALWAYS DO THIS
const venue = await db.select().from(venue_catalog).where(eq(venue_catalog.city, city));
const model = process.env.OPENAI_MODEL;
```

3. Single-Path Triad (No Fallbacks in Production Flow)

The recommendation pipeline uses a deterministic three-stage process: 1. **Claude Sonnet 4.5** (Strategist) - Strategic overview 2. **GPT-5** (Planner) - Tactical venue selection with deep reasoning 3. **Gemini 2.5 Pro** (Validator) - JSON validation and ranking

No fallback models - If any stage fails, the entire request fails with clear error. This preserves ML training data integrity (we know exactly which model produced each output).

Exception: Agent Override (workspace operations) uses fallback chain for operational resilience.

4. Database-Driven Reconciliation

Every venue recommendation MUST reconcile to: - `venue_catalog` table (seeded venues) - Google Places API (validation of coordinates/hours) - `ranking_candidates` table (ML training data)

Flow:

```
GPT-5 suggests "Stonebriar Centre"
  → Lookup place_id via Google Places Find Place API
  → Validate coordinates via Google Geocoding API
  → Fetch hours via Google Places Details API
  → Store in venue_catalog (if new)
  → Create ranking_candidates row (ML capture)
  → Return to user with verified data
```

5. Complete Snapshot Gating

No LLM call without a complete location snapshot: - GPS coordinates (lat, lng, accuracy) - City, state, timezone - Weather, AQI - Time context (daypart, hour, dow, is_weekend) - H3 geospatial index

If any core field is missing, return `400 Bad Request` with `refresh_required` status.

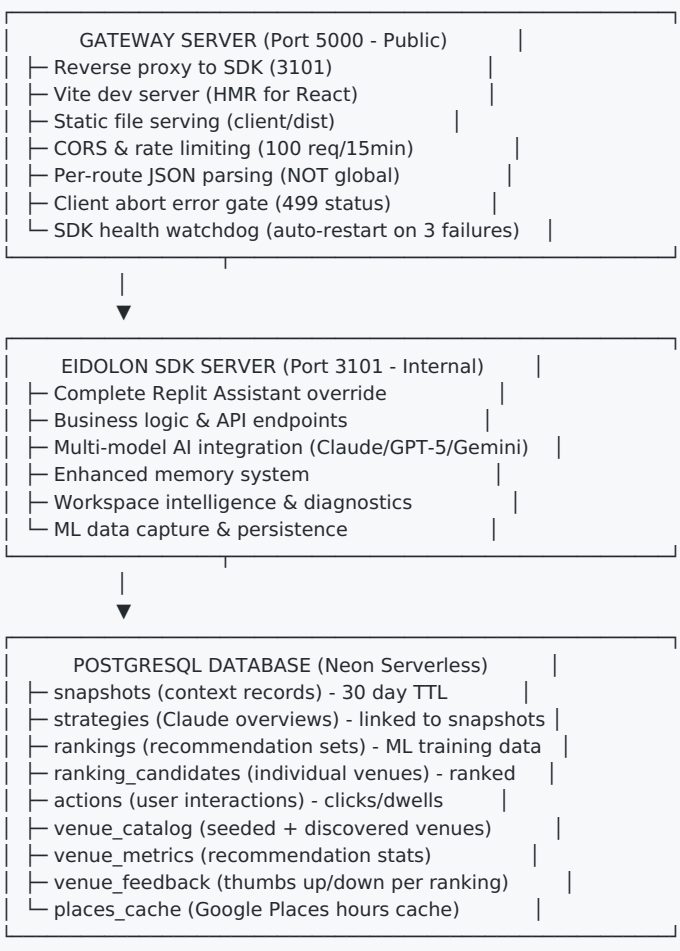
6. Deterministic Logging for ML

For every recommendation set served: - Input: `snapshot_id`, context hash, H3 hex - Process: `ranking_id`, model names, token usage, latency - Output: `ranking_candidates` (6 venues with rank, earnings, distance) - Outcome: `actions` (view, dwell, click, dismiss)

This enables counterfactual learning: “Given context X, model suggested Y, user chose Z.”

□ System Architecture

Multi-Server Design



Technology Stack

Frontend: - React 18 (single render in production, StrictMode in dev) - TypeScript 5.x - TanStack Query v5 (data fetching with smart caching) - Wouter (lightweight routing) - Tailwind CSS + shadcn/ui

Backend: - Node.js 22.x (ESM modules) - Express.js (per-route JSON parsing) - Drizzle ORM (type-safe database queries) - PostgreSQL (Neon serverless)

AI Models: - Claude Sonnet 4.5 (`claude-sonnet-4-5-20250929`) - GPT-5 (`gpt-5` - reasoning_effort: high) - Gemini 2.5 Pro (`gemini-2.5-pro-latest`)

External APIs: - Google Maps (Geocoding, Directions, Timezone, Places) - Google Air Quality API - OpenWeather API - FAA ASWS (Airport Status Web Service)

□ Project File Structure

Complete Directory Layout

```
vecto-pilot/
├── .env                          # Environment configuration (NOT in git)
├── .env.example                  # Template for required env vars
├── .gitignore                    # Git exclusions
├── package.json                  # Node.js dependencies
├── tsconfig.json                 # TypeScript configuration
├── vite.config.ts                # Vite build configuration
├── tailwind.config.ts            # Tailwind CSS configuration
├── drizzle.config.ts             # Database ORM configuration
├── gateway-server.js             # Main entry point (Port 5000)
│
├── client/                       # Frontend React application
│   ├── index.html                # HTML entry point
│   ├── public/                   # Static assets
│   │   └── vite.svg              # Favicon
│   └── src/
│       ├── main.tsx              # React app initialization
│       ├── App.tsx               # Root component with routing
│       ├── index.css             # Global styles + Tailwind
│       │
│       ├── components/           # React components
│       │   ├── ui/               # shadcn/ui components
│       │   │   ├── button.tsx
│       │   │   ├── card.tsx
│       │   │   ├── dialog.tsx
│       │   │   ├── form.tsx
│       │   │   ├── input.tsx
│       │   │   ├── label.tsx
│       │   │   ├── select.tsx
│       │   │   ├── skeleton.tsx
│       │   │   ├── toast.tsx
│       │   │   └── toaster.tsx
│       │   ├── BlockCard.tsx         # Venue recommendation card
│       │   ├── StrategyCard.tsx      # Claude strategy display
│       │   └── LocationHeader.tsx    # GPS status header
│       │
│       ├── contexts/             # React Context providers
│       │   └── location-context-clean.tsx  # GPS + location state management
│       │
│       ├── hooks/                # Custom React hooks
│       │   ├── use-toast.ts          # Toast notifications
│       │   └── useGeoPosition.ts     # Browser geolocation wrapper
│       │
│       ├── lib/                  # Utilities and helpers
│       │   ├── queryClient.ts        # TanStack Query setup + apiRequest
│       │   ├── snapshot.ts           # Snapshot creation logic
│       │   ├── utils.ts              # General utilities (cn, etc.)
│       │   └── validators.ts         # Zod schemas for client
│       │
│       └── pages/                # Route components
│           ├── home.tsx          # Landing page (/)
│           ├── co-pilot.tsx      # Main app (/co-pilot)
│           └── settings.tsx      # User settings (/settings)
│
├── server/                       # Backend Node.js application
│   ├── index.js                  # SDK server entry point (Port 3101)
│   ├── vite.ts                   # Vite dev server setup
│   │
│   ├── config/                   # Configuration files
│   │   ├── [illegible]           # Fi[illegible]
```

```
│   │   └── assistant-policy.json          # Eidolon assistant policy
│   │
│   ├── lib/                              # Core business logic
│   │   ├── adapters/                     # AI provider adapters
│   │   │   ├── anthropic-claude.js       # Claude Sonnet 4.5 adapter
│   │   │   ├── openai-gpt5.js            # GPT-5 adapter
│   │   │   └── google-gemini.js          # Gemini 2.5 Pro adapter
│   │   │
│   │   ├── gpt5-tactical-planner.js      # GPT-5 venue selection
│   │   ├── gemini-validator.js           # Gemini JSON validation
│   │   ├── strategy-generator.js         # Claude strategy background job
│   │   ├── persist-ranking.js            # Atomic DB persistence
│   │   ├── job-queue.js                  # Background job queue
│   │   ├── venue-resolver.js             # Google Places resolution
│   │   ├── distance-calculator.js        # Routes API integration
│   │   └── snapshot-validator.js         # Snapshot completeness check
│   │
│   ├── routes/                           # Express route handlers
│   │   ├── blocks.js                     # POST /api/blocks (Triad pipeline)
│   │   ├── blocks-fast.js                # POST /api/blocks/fast (Quick Picks)
│   │   ├── location.js                   # Location resolution endpoints
│   │   ├── actions.js                    # POST /api/actions (user tracking)
│   │   ├── feedback.js                   # Venue/strategy feedback
│   │   └── health.js                     # GET /health (monitoring)
│   │
│   ├── util/                             # Server utilities
│   │   ├── db.js                         # PostgreSQL connection pool
│   │   ├── logger.js                     # Structured logging
│   │   └── error-handler.js              # Express error middleware
│   │
│   └── storage.ts                        # Storage interface (future)
│
├── shared/                               # Shared between client/server
│   └── schema.ts                         # Drizzle database schema + Zod types
│
├── data/                                 # Data storage (gitignored)
│   ├── context-snapshots/                # Snapshot JSON backups
│   └── logs/                             # Application logs
│
├── tools/                                # Developer tools
│   └── research/                         # Model research scripts
│       └── model-discovery.mjs           # Monthly model verification
│
└── docs/                                 # Documentation
    ├── ARCHITECTUREV2.md                 # This document
    ├── ARCHITECTURE.md                   # V1 architecture (historical)
    ├── MODEL.md                          # AI model specifications
    ├── ISSUES.md                         # Issue tracking + root cause analysis
    └── README.md                         # Quick start guide
```

Key File Descriptions

Entry Points

- **`gateway-server.js`** - Main application server (Port 5000)
  - Proxies `/api/*` to SDK server (3101)
  - Serves Vite dev server (development)
  - Serves `client/dist` static files (production)
  - Handles CORS, rate limiting, error logging
- **`server/index.js`** - SDK server (Port 3101)
  - All business logic and AI integration
  - Database operations
  - Background job processing

Frontend Core Files

- **`client/src/App.tsx`** - Route configuration (Wouter)

- `client/src/contexts/location-context-clean.tsx` - GPS state management
- `client/src/lib/queryClient.ts` - TanStack Query configuration
- `client/src/pages/co-pilot.tsx` - Main recommendation interface

Backend Core Files

- `server/routes/blocks.js` - Triad pipeline orchestration
- `server/lib/gpt5-tactical-planner.js` - GPT-5 venue generation
- `server/lib/persist-ranking.js` - Atomic database writes
- `shared/schema.ts` - Database schema (single source of truth)

Configuration Files

- `.env` - Environment variables (create from `.env.example`)
- `drizzle.config.ts` - Database connection and migrations
- `vite.config.ts` - Frontend build configuration
- `tailwind.config.ts` - Styling configuration

Environment Variables Reference

Complete .env Configuration

```
# ================================================================================
# DATABASE CONFIGURATION
# ================================================================================
DATABASE_URL=postgresql://user:password@host:5432/database?sslmode=require
# Required: PostgreSQL connection string (Neon serverless recommended)
# Format: postgresql://[user]:[password]@[host]:[port]/[database]?sslmode=require
# Used in: server/util/db.js, drizzle.config.ts

# ================================================================================
# AI MODEL CONFIGURATION (Triad Pipeline)
# ================================================================================

# --- Claude Sonnet 4.5 (Strategist) ---
ANTHROPIC_API_KEY=sk-ant-api03-...
# Required: Anthropic API key for Claude
# Get from: https://console.anthropic.com/settings/keys
# Used in: server/lib/adapters/anthropic-claude.js

CLAUDE_MODEL=claude-sonnet-4-5-20250929
# Required: Exact model ID for Claude
# Default: claude-sonnet-4-5-20250929
# VERIFY MONTHLY: This model may be deprecated
# Used in: server/lib/strategy-generator.js

CLAUDE_TIMEOUT_MS=12000
# Optional: Claude API timeout in milliseconds
# Default: 12000 (12 seconds)
# Used in: server/lib/strategy-generator.js

# --- OpenAI GPT-5 (Planner) ---
OPENAI_API_KEY=sk-proj-...
# Required: OpenAI API key for GPT-5
# Get from: https://platform.openai.com/api-keys
# Used in: server/lib/adapters/openai-gpt5.js

OPENAI_MODEL=gpt-5
# Required: GPT-5 model ID
# Default: gpt-5
# ⚠ DO NOT use: gpt-4, gpt-4-turbo (deprecated for this app)
# Used in: server/lib/gpt5-tactical-planner.js
```

```
PLANNER_DEADLINE_MS=120000
# Required: GPT-5 timeout in milliseconds
# Default: 120000 (2 minutes)
# ⚠ CRITICAL: This is the canonical timeout for GPT-5
# Used in: server/lib/gpt5-tactical-planner.js

GPT5_REASONING_EFFORT=high
# Required: GPT-5 reasoning effort level
# Options: minimal | low | medium | high
# Default: high (best quality, slower)
# Used in: server/lib/gpt5-tactical-planner.js

# --- Google Gemini 2.5 Pro (Validator) ---
GOOGLE_GENERATIVE_AI_API_KEY=AIza...
# Required: Google AI API key for Gemini
# Get from: https://makersuite.google.com/app/apikey
# Used in: server/lib/adapters/google-gemini.js

GEMINI_MODEL=gemini-2.5-pro-latest
# Required: Gemini model ID
# Default: gemini-2.5-pro-latest
# Options: gemini-2.5-pro-latest | gemini-2.5-flash
# Used in: server/lib/gemini-validator.js

VALIDATOR_DEADLINE_MS=60000
# Optional: Gemini API timeout in milliseconds
# Default: 60000 (60 seconds)
# Used in: server/lib/gemini-validator.js

# ================================================================================
# GOOGLE MAPS API CONFIGURATION
# ================================================================================
GOOGLE_MAPS_API_KEY=AIza...
# Required: Google Maps Platform API key
# Enable APIs: Geocoding, Places, Routes, Timezone
# Get from: https://console.cloud.google.com/apis/credentials
# Used in: server/routes/location.js, server/lib/venue-resolver.js

# ================================================================================
# WEATHER & AIR QUALITY APIS
# ================================================================================
OPENWEATHER_API_KEY=abc123...
# Required: OpenWeather API key for weather data
# Get from: https://openweathermap.org/api
# Free tier: 1000 calls/day
# Used in: server/routes/location.js

GOOGLE_AIR_QUALITY_API_KEY=AIza...
# Optional: Google Air Quality API key (can use GOOGLE_MAPS_API_KEY)
# Get from: https://console.cloud.google.com/apis/credentials
# Used in: server/routes/location.js

# ================================================================================
# SERVER CONFIGURATION
# ================================================================================
NODE_ENV=development
# Required: Environment mode
# Options: development | production
# Default: development
# Used in: gateway-server.js, server/index.js

PORT=5000
# Optional: Gateway server port (public-facing)
# Default: 5000 (Replit firewall requirement)
# ⚠ MUST be 5000 on Replit (only port not firewalled)
# Used in: gateway-server.js

SDK_PORT=3101
# Optional: SDK server port (internal)
# Default: 3101
# Used in: server/index.js
```

```
VITE_PORT=3003
# Optional: Vite dev server port (development only)
# Default: 3003
# Used in: server/vite.ts

# ============================================================================
# TOTAL BUDGET & RATE LIMITS
# ============================================================================
LLM_TOTAL_BUDGET_MS=200000
# Optional: Total AI pipeline timeout
# Default: 200000 (200 seconds = 3m 20s)
# Sum of: Claude (12s) + GPT-5 (120s) + Gemini (60s) + overhead
# Used in: server/routes/blocks.js

RATE_LIMIT_WINDOW_MS=900000
# Optional: Rate limit window
# Default: 900000 (15 minutes)
# Used in: gateway-server.js

RATE_LIMIT_MAX_REQUESTS=100
# Optional: Max requests per window
# Default: 100
# Used in: gateway-server.js

# ============================================================================
# LOGGING & MONITORING
# ============================================================================
LOG_LEVEL=info
# Optional: Logging verbosity
# Options: debug | info | warn | error
# Default: info
# Used in: server/util/logger.js

ENABLE_REQUEST_LOGGING=true
# Optional: Log all HTTP requests
# Default: true
# Used in: gateway-server.js

# ============================================================================
# FEATURE FLAGS (Future)
# ============================================================================
ENABLE_ML_RERANKING=false
# Optional: Use ML model instead of Triad (future)
# Default: false
# Planned for: Q1 2026

ENABLE_FAST_PATH=true
# Optional: Enable /api/blocks/fast endpoint
# Default: true
# Used in: server/routes/blocks-fast.js
```

Environment Variable Validation

On Startup, the application checks: 1. □ All required API keys present 2. □ Model IDs are non-empty strings 3. □ Timeout values are positive integers 4. □ Database URL is valid PostgreSQL connection string

If validation fails: - Server logs detailed error message - Application exits with code 1 - Clear instructions printed to console

Example validation error:

```
□ CONFIGURATION ERROR:
Missing required environment variable: ANTHROPIC_API_KEY
Please add to .env file:

ANTHROPIC_API_KEY=sk-ant-api03-...

Get your API key from: https://console.anthropic.com/settings/keys
```

API Endpoint Reference

Base URL

- **Development:** `http://localhost:5000/api`
- **Production:** `https://[your-deployment].replit.app/api`

Authentication

Currently anonymous (user_id optional). Future: JWT tokens.

Common Headers

```
Content-Type: application/json
X-Correlation-Id: <uuid>          # Optional: Request tracing
X-Snapshot-Id: <uuid>             # Required for /api/blocks
X-Idempotency-Key: <uuid>         # Required for /api/actions
```

---

1. Location Resolution

`POST /api/location/resolve`

Resolve GPS coordinates to city, timezone, weather, and air quality.

Request:

```
{
  "lat": 33.12854,
  "lng": -96.87551,
  "accuracy": 97
}
```

Response (200 OK):

```
{
  "city": "Frisco",
  "state": "TX",
  "country": "United States",
  "formatted_address": "Frisco, TX 75034, USA",
  "timezone": "America/Chicago",
  "utc_offset": -18000,
  "weather": {
    "tempF": 75,
    "conditions": "clear sky",
    "description": "Clear"
  },
  "air": {
    "aqi": 71,
    "category": "Moderate",
    "dominantPollutant": "pm25"
  }
}
```

Error (400 Bad Request):

```
{
  "error": "Invalid coordinates",
  "details": "Latitude must be between -90 and 90"
}
```

POST /api/location/snapshot

Create a location context snapshot for recommendations.

Request:

```
{
  "snapshot_id": "51a60564-c6d5-4360-8ed1-24f5c1cc1df8",
  "user_id": "ce372d10-30b7-4f14-af77-d1cada207d27",
  "device_id": "abc-123-def",
  "session_id": "session-xyz",
  "lat": 33.12854,
  "lng": -96.87551,
  "accuracy_m": 97,
  "coord_source": "gps",
  "city": "Frisco",
  "state": "TX",
  "country": "United States",
  "timezone": "America/Chicago",
  "local_iso": "2025-10-13T11:28:00",
  "weather": {...},
  "air": {...}
}
```

Response (201 Created):

```
{
  "snapshot_id": "51a60564-c6d5-4360-8ed1-24f5c1cc1df8",
  "h3_r8": "8823674bfffffff",
  "airport_context": {
    "airport_code": "DFW",
    "distance_miles": 12.5,
    "delay_minutes": 15
  },
  "status": "ok"
}
```

---

2. Recommendations (Triad Pipeline)

POST /api/blocks

Get AI-powered venue recommendations (Triad: Claude → GPT-5 → Gemini).

Headers:

```
X-Snapshot-Id: 51a60564-c6d5-4360-8ed1-24f5c1cc1df8
```

Request:

```
{
  "userId": "ce372d10-30b7-4f14-af77-d1cada207d27"
}
```

Response (200 OK):

```
{
 "blocks": [
  {
   "id": "ChIJPXYJW648TIYRjtIg0_rKXeo",
   "name": "Legacy West",
   "address": "7401 Windrose Ave, Plano, TX 75024",
   "lat": 33.0795292,
   "lng": -96.8265703,
   "distance": 6.3,
   "driveTime": 14,
   "value_per_min": 0.52,
   "value_grade": "C",
   "not_worth": false,
   "rank": 1,
   "category": "shopping",
   "hours": "Open until 9:00 PM",
   "reasoning": "High-end mixed-use shopping district with corporate lunch traffic..."
  }
 ],
 "staging": {
  "name": "Shops at Legacy North Parking Garage",
  "lat": 33.0798,
  "lng": -96.8209,
  "address": "7300 Lone Star Dr, Plano, TX 75024"
 },
 "strategy": {
  "text": "Reposition to Legacy/Granite corridor for lunch rush...",
  "model": "claude-sonnet-4-5-20250929"
 },
 "meta": {
  "ranking_id": "389c4944-e7da-40cb-9581-4471ffbfbe52",
  "snapshot_id": "51a60564-c6d5-4360-8ed1-24f5c1cc1df8",
  "total_ms": 257594,
  "model_name": "claude-sonnet-4-5→gpt-5→gemini-2.5-pro"
 }
}
```


Response (202 Accepted - Strategy Pending):


```
{
 "status": "pending",
 "message": "Strategy generation in progress",
 "retry_after": 5
}
```


Error (400 Bad Request - Incomplete Snapshot):


```
{
 "error": "Incomplete snapshot",
 "status": "refresh_required",
 "missing_fields": ["timezone", "weather"]
}
```

`GET /api/blocks/strategy/:snapshotId`

Get Claude strategic overview for a snapshot (background job result).

Response (200 OK):

```
{
 "strategy": "Today is Monday at 11:28 AM in Frisco. With perfect weather...",
 "status": "ok",
 "model": "claude-sonnet-4-5-20250929",
 "latency_ms": 3450
}
```


Response (202 Accepted - Still Generating):


```
{
 "status": "pending",
 "attempt": 1,
 "next_retry_at": "2025-10-13T11:28:15Z"
}
```


---

3. User Actions & Tracking

POST /api/actions

Log user interaction with venue card.

Headers:

```
X-Idempotency-Key: action-abc-123-def
```

Request:


```
{
 "action": "block_clicked",
 "snapshot_id": "51a60564-c6d5-4360-8ed1-24f5c1cc1df8",
 "ranking_id": "389c4944-e7da-40cb-9581-4471ffbfbe52",
 "block_id": "ChIJPXYJW648TIYRjtIg0_rKXeo",
 "from_rank": 1,
 "dwell_ms": 5432
}
```


Action Types: - `view` - Card appeared in viewport (IntersectionObserver) - `dwell` - User stayed on card >2 seconds - `block_clicked` - User tapped “Navigate” button - `block_dismissed` - User swiped away or closed

Response (201 Created):


```
{
 "action_id": "action-uuid-123",
 "status": "logged"
}
```


Error (409 Conflict - Duplicate):


```
{
 "error": "Duplicate action",
 "idempotency_key": "action-abc-123-def"
}
```


---

4. Feedback

POST /api/feedback/venue

Submit thumbs up/down for a venue.

Request:

```
{
  "snapshot_id": "51a60564-c6d5-4360-8ed1-24f5c1cc1df8",
  "ranking_id": "389c4944-e7da-40cb-9581-4471ffbfbe52",
  "place_id": "ChIJPXYJW648TIYRjtIg0_rKXeo",
  "sentiment": "up",
  "comment": "Great lunch spot, easy pickup at north entrance"
}
```

Sentiment: `"up"` or `"down"`

Response (201 Created):

```
{
  "feedback_id": "feedback-uuid",
  "status": "recorded"
}
```

Rate Limit: 10 feedback submissions per 1 minute per user

POST /api/feedback/strategy

Submit feedback on Claude's strategic overview.

Request:

```
{
  "snapshot_id": "51a60564-c6d5-4360-8ed1-24f5c1cc1df8",
  "sentiment": "down",
  "comment": "Strategy suggested north side but surge was south"
}
```

Response (201 Created):

```
{
  "feedback_id": "strategy-feedback-uuid",
  "status": "recorded"
}
```

---

5. Health & Monitoring

GET /health

Server health check.

Response (200 OK):

```
{
  "status": "healthy",
  "timestamp": "2025-10-13T11:28:00Z",
  "uptime_seconds": 3600,
  "database": "connected",
  "services": {
    "claude": "operational",
    "gpt5": "operational",
    "gemini": "operational"
  }
}
```

⚛ Frontend Architecture

React Component Hierarchy

```
<App>                              # Root component (Wouter routing)
 <LocationProvider>                    # GPS + location state
  <QueryClientProvider>                 # TanStack Query
   <Toaster />                       # Toast notifications

   <Route path="/">
    <HomePage />
   </Route>

   <Route path="/co-pilot">
    <CoPilotPage>                     # Main app interface
     <LocationHeader />                  # GPS status, city, weather

     <StrategyCard                    # Claude strategic overview
      strategy={strategyQuery.data}
      isLoading={strategyQuery.isLoading}
     />

     {blocksQuery.data?.blocks.map(block => (
      <BlockCard                      # Venue recommendation
       key={block.id}
       venue={block}
       onNavigate={handleNavigate}
       onFeedback={handleFeedback}
      />
     ))}
    </CoPilotPage>
   </Route>

   <Route path="/settings">
    <SettingsPage />
   </Route>
  </QueryClientProvider>
 </LocationProvider>
</App>
```

State Management Architecture

1. GPS & Location State (LocationContext)

File: `client/src/contexts/location-context-clean.tsx`

Responsibilities: - Manages GPS coordinates from browser Geolocation API - Triggers location resolution (city, weather, timezone) - Creates snapshots for recommendation requests - Broadcasts location changes via custom events

State Shape:

```
{
  // GPS Coordinates
  coords: {
    latitude: number;
    longitude: number;
    accuracy: number;
  } | null;

  // Resolved Location
  city: string | null;
  state: string | null;
  timezone: string | null;

  // Context Data
  weather: {
    tempF: number;
    conditions: string;
  } | null;
  air: {
    aqi: number;
    category: string;
  } | null;

  // Snapshot
  currentSnapshot: {
    snapshot_id: string;
    h3_r8: string;
  } | null;

  // Loading States
  isLoadingGPS: boolean;
  isLoadingEnrichment: boolean;
  error: string | null;
}
```

Key Pattern - Object Reference Creation:

```
// ✅ CORRECT - Creates new reference to trigger React re-render
setLocationState(prev => ({
  ...prev,
  coords: { ...coords }  // New object reference
}));

// ❌ WRONG - Mutates existing object (React won't re-render)
setLocationState(prev => {
  prev.coords = coords;  // Mutation
  return prev;
});
```

2. Server State (TanStack Query)

File: `client/src/lib/queryClient.ts`

Configuration:

```
export const queryClient = new QueryClient({
  defaultOptions: {
    queries: {
      staleTime: 1000 * 60 * 5,       // 5 minutes
      gcTime: 1000 * 60 * 10,         // 10 minutes (formerly cacheTime)
      retry: 3,
      retryDelay: attemptIndex => Math.min(1000 * 2 ** attemptIndex, 30000),
      refetchOnWindowFocus: false,    // Prevent refetch on tab switch
    },
  },
});
```

Custom Fetcher:

```
export async function apiRequest<T>(
  url: string,
  options?: RequestInit
): Promise<T> {
  const res = await fetch(url, {
    ...options,
    headers: {
      'Content-Type': 'application/json',
      ...options?.headers,
    },
  });

  if (!res.ok) {
    const error = await res.json();
    throw new Error(error.message || `HTTP ${res.status}`);
  }

  return res.json();
}
```

Query Patterns:

```
// Blocks recommendation query
const blocksQuery = useQuery({
  queryKey: ['/api/blocks', snapshotId],  // Hierarchical key
  queryFn: async () => {
    return apiRequest<BlocksResponse>('/api/blocks', {
      method: 'POST',
      headers: { 'X-Snapshot-Id': snapshotId },
      body: JSON.stringify({ userId }),
    });
  },
  enabled: !!snapshotId,                // Only run if snapshot exists
  staleTime: Infinity,                  // Never auto-refetch (snapshot-specific)
});

// Strategy query (background job polling)
const strategyQuery = useQuery({
  queryKey: ['/api/blocks/strategy', snapshotId],
  queryFn: () => apiRequest(`/api/blocks/strategy/${snapshotId}`),
  refetchInterval: (data) => {
    return data?.status === 'pending' ? 2000 : false;  // Poll every 2s if pending
  },
  enabled: !!snapshotId,
});
```

Mutation Patterns:

```
// Action logging mutation
const actionMutation = useMutation({
  mutationFn: (action: ActionPayload) => {
    return apiRequest('/api/actions', {
      method: 'POST',
      headers: {
        'X-Idempotency-Key': crypto.randomUUID(),
      },
      body: JSON.stringify(action),
    });
  },
  onSuccess: () => {
    // Invalidate related queries
    queryClient.invalidateQueries({
      queryKey: ['/api/blocks', action.snapshot_id]
    });
  },
});
```

Routing Architecture (Wouter)

File: `client/src/App.tsx`

```
import { Route, Switch, useLocation } from 'wouter';

function App() {
  const [location, setLocation] = useLocation();

  return (
    <Switch>
      <Route path="/">
        <HomePage />
      </Route>

      <Route path="/co-pilot">
        <CoPilotPage />
      </Route>

      <Route path="/settings">
        <SettingsPage />
      </Route>

      <Route>
        <NotFoundPage />
      </Route>
    </Switch>
  );
}
```

Navigation:

```
import { Link, useLocation } from 'wouter';

// Declarative navigation
<Link href="/co-pilot">
  <Button>Get Recommendations</Button>
</Link>

// Programmatic navigation
const [, setLocation] = useLocation();
setLocation('/co-pilot');
```

Form Handling Pattern

Using react-hook-form + Zod:

```tsx
import { useForm } from 'react-hook-form';
import { zodResolver } from '@hookform/resolvers/zod';
import { z } from 'zod';

// Define schema
const feedbackSchema = z.object({
  sentiment: z.enum(['up', 'down']),
  comment: z.string().min(10).max(500).optional(),
});

type FeedbackForm = z.infer<typeof feedbackSchema>;

function FeedbackDialog() {
  const form = useForm<FeedbackForm>({
    resolver: zodResolver(feedbackSchema),
    defaultValues: {                    // ⚠ REQUIRED for controlled form
      sentiment: 'up',
      comment: '',
    },
  });

  const onSubmit = async (data: FeedbackForm) => {
    await apiRequest('/api/feedback/venue', {
      method: 'POST',
      body: JSON.stringify(data),
    });
  };

  return (
    <Form {...form}>
      <form onSubmit={form.handleSubmit(onSubmit)}>
        <FormField
          control={form.control}
          name="sentiment"
          render={({ field }) => (
            <FormItem>
              <FormLabel>How was this venue?</FormLabel>
              <FormControl>
                <RadioGroup {...field}>
                  <RadioGroupItem value="up">👍 Good</RadioGroupItem>
                  <RadioGroupItem value="down">👎 Not Worth It</RadioGroupItem>
                </RadioGroup>
              </FormControl>
            </FormItem>
          )}
        />

        <Button type="submit">Submit</Button>
      </form>
    </Form>
  );
}
```

Styling System

Tailwind CSS + shadcn/ui:

```
// Global styles (client/src/index.css)
@tailwind base;
@tailwind components;
@tailwind utilities;

@layer base {
  :root {
    --background: 210 11% 98%;       /* #F5F7FA */
    --foreground: 222 47% 11%;       /* #0F172A */
    --primary: 221 83% 53%;          /* #3B82F6 */
    /* ... */
  }

  .dark {
    --background: 222 47% 11%;
    --foreground: 210 40% 98%;
    /* ... */
  }
}

// Component usage
<Card className="bg-background border-border">
  <CardHeader>
    <CardTitle className="text-foreground">Legacy West</CardTitle>
  </CardHeader>
</Card>

// Utility classes
<Button className="bg-primary hover:bg-primary/90">
  Navigate
</Button>
```

□ Data Flow & Mapping

1. Location Resolution Flow

```
User Opens App / Clicks GPS Refresh
    │
    ├──→ Browser Geolocation API (navigator.geolocation.getCurrentPosition)
    │
    ├──→ useGeoPosition Hook (client/src/hooks/useGeoPosition.ts)
    │    └─ Returns: { latitude, longitude, accuracy }
    │
    ├──→ LocationContext (client/src/contexts/location-context-clean.tsx)
    │    ├─ Creates new object reference via {...prev, coords: {...coords}}
    │    ├─ Triggers React re-render of all consumers
    │    └─ Dispatches "location-changed" event
    │
    ├──→ POST /api/location/resolve (parallel calls to Google APIs)
    │    ├─ Google Geocoding API → city, state, country, formatted_address
    │    ├─ Google Timezone API → timezone, UTC offset
    │    ├─ OpenWeather API → weather, temp, conditions
    │    └─ Google AQ API → AQI, category, pollutants
    │
    ├──→ createSnapshot() (client/src/lib/snapshot.ts)
    │    └─ Combines GPS + resolved data + time context → SnapshotV1 format
    │
    └──→ POST /api/location/snapshot (server/routes/location.js)
         ├─ Calculate H3 geospatial index (resolution 8 = ~0.46 km² hexagons)
         ├─ Fetch nearby airport delays (FAA ASWS API)
         ├─ Save to snapshots table (PostgreSQL)
         ├─ Save to filesystem (data/context-snapshots/*.json backup)
         ├─ Enqueue strategy generation job (background)
         └─ Return: { snapshot_id, h3_r8, airport_context }
```

2. Strategy Generation Flow (Background Job)

```
Snapshot Created Event
   │
   ├──→ Job Queue (server/lib/job-queue.js)
   │    └─ Enqueues strategy generation for snapshot_id
   │
   ├──→ generateStrategyForSnapshot() (server/lib/strategy-generator.js)
   │    ├─ Load snapshot from database
   │    ├─ Build prompt: "You are in {city} at {time}, weather is {conditions}..."
   │    ├─ Call Claude Sonnet 4.5 API (max_tokens: 500, temperature: 0.0)
   │    └─ Returns: 2-3 sentence strategic overview
   │
   ├──→ INSERT INTO strategies
   │    ├─ snapshot_id (FK to snapshots.snapshot_id)
   │    ├─ strategy (text)
   │    ├─ status: 'ok' | 'pending' | 'failed'
   │    ├─ latency_ms, tokens, attempt
   │    └─ model_name: 'claude-sonnet-4-5-20250929'
   │
   └──→ Client polls GET /api/blocks/strategy/:snapshotId
        ├─ Returns 202 Accepted (status: 'pending') while generating
        ├─ Returns 200 OK with strategy text when complete
        └─ Returns 304 Not Modified (ETag caching) on subsequent requests
```

3. Smart Blocks Recommendation Flow (Triad Pipeline)

```
User Navigates to Co-Pilot Tab
   │
   ├──→ POST /api/blocks (server/routes/blocks.js)
   │    ├─ Headers: X-Snapshot-Id (anchors to specific context)
   │    ├─ Load snapshot from database (ONLY source of origin coordinates)
   │    └─ Validate snapshot completeness (lat, lng, city, timezone required)
   │
   ├──→ TRIAD STEP 1/3: Wait for Claude Strategy (gating requirement)
   │    ├─ Query strategies table WHERE snapshot_id = :id
   │    ├─ If status = 'pending': Return 202 Accepted (strategy_required)
   │    ├─ If status = 'failed': Return 500 with error details
   │    └─ If status = 'ok': Proceed to Step 2 with strategy text
   │
   ├──→ TRIAD STEP 2/3: GPT-5 Tactical Planner (requires Claude strategy)
   │    ├─ Input: Claude's strategy + snapshot context + venue catalog shortlist
   │    ├─ Model: gpt-5 (reasoning_effort: high, max_completion_tokens: 32000)
   │    ├─ Timeout: 120 seconds (PLANNER_DEADLINE_MS from .env)
   │    ├─ Output: 4-6 venue recommendations with:
   │    │   - name, lat, lng, category
   │    │   - approach_roads, best_pickup_point
   │    │   - peak_windows_local, avoid_zones
   │    │   - reasoning (why this venue now)
   │    └─ best_staging_location: { name, lat, lng, tactical_summary }
   │
   ├──→ VENUE RESOLUTION (Google APIs - Geocoding + Places split)
   │    ├─ For each GPT-5 venue:
   │    │   ├─ IF has lat/lng: Reverse geocode → get place_id + address
   │    │   ├─ ELSE: Places Find Place by name → get place_id + coords
   │    │   ├─ Places Details (hours only) → get business hours
   │    │   ├─ Cache to places table (coords/address)
   │    │   ├─ Cache to places_cache table (hours separately)
   │    │   └─ Skip venue if resolution fails (e.g., generic districts)
   │    └─ Enrich staging location (same flow)
   │
   ├──→ TRAFFIC-AWARE DISTANCE & ETA (Google Routes API)
   │    ├─ For each resolved venue:
   │    │   ├─ Call Routes API with TRAFFIC_AWARE_OPTIMAL mode
   │    │   ├─ Input: snapshot coords → venue coords
   │    │   ├─ Returns: distanceMeters, durationSeconds
   │    │   └─ Calculate: distance_miles, drive_minutes
   │    └─ NO FALLBACK: If Routes API fails, fail request (no Haversine approximation)
   │
```

```
    │
    ├──→ TRIAD STEP 3/3: Gemini 2.5 Pro Validator
    │    ├─ Input: Enriched venues with real distances + drive times
    │    ├─ Model: gemini-2.5-pro-latest (temperature: 0.2, maxOutputTokens: 2048)
    │    ├─ Timeout: 60 seconds (VALIDATOR_DEADLINE_MS from .env)
    │    ├─ Tasks:
    │    │   ├─ Validate JSON structure (placeId, name, coordinates present)
    │    │   ├─ Calculate value_per_minute = earnings / (drive + trip + wait)
    │    │   ├─ Assign value_grade (A/B/C/D/F based on value_per_min thresholds)
    │    │   ├─ Set not_worth flag (value_per_min < 0.50)
    │    │   └─ Rank by value_per_minute descending
    │    └─ Output: Validated + ranked venues (top 6)
    │
    ├──→ ATOMIC DATABASE PERSISTENCE (server/lib/persist-ranking.js)
    │    ├─ BEGIN TRANSACTION
    │    ├─ INSERT INTO rankings (snapshot_id, model_name, correlation_id)
    │    ├─ INSERT INTO ranking_candidates (6 rows):
    │    │   - ranking_id, place_id, name, lat, lng
    │    │   - rank (1-6), distance_miles, drive_minutes
    │    │   - value_per_min, value_grade, not_worth
    │    │   - estimated_distance_miles, distance_source: 'routes_api'
    │    ├─ COMMIT
    │    └─ On error: ROLLBACK (all or nothing)
    │
    ├──→ FEEDBACK ENRICHMENT (non-blocking)
    │    └─ JOIN venue_feedback ON place_id → add thumbs_up/down counts
    │
    └──→ RETURN to Client (client/src/pages/co-pilot.tsx)
         └─ Display: 6 venue cards sorted by value_per_minute
```

4. User Action Logging Flow

```
User Interacts with Venue Card
    │
    ├──→ Action Types:
    │    ├─ 'view' - Card appears on screen (IntersectionObserver)
    │    ├─ 'dwell' - User stays on card >2 seconds
    │    ├─ 'block_clicked' - User taps "Navigate" button
    │    └─ 'block_dismissed' - User swipes away or closes
    │
    ├──→ POST /api/actions (server/routes/actions.js)
    │    ├─ Headers: X-Idempotency-Key (prevent duplicate logging)
    │    ├─ Body: {
    │    │   action: 'dwell',
    │    │   snapshot_id: 'uuid',
    │    │   ranking_id: 'uuid',
    │    │   block_id: 'place_id',
    │    │   dwell_ms: 5432,
    │    │   from_rank: 1
    │    │ }
    │    └─ Retry logic: 5 attempts with exponential backoff (handles DB replication lag)
    │
    └──→ INSERT INTO actions
         └─ Links: snapshot → ranking → candidate → action
              └─ Enables ML training: "Given context X, user chose block Y at rank Z"
```

AI Model Strategy & Risk Management

Current Production Models

Stage	Provider	Model ID	Released	Timeout	Purpose
Strategist	Anthropic	`claude-sonnet-4-5-20250929`	Sep 2025	12s	Strategic overview (2-3 sentences)
Planner	OpenAI	`gpt-5`	Aug 2025	120s	Tactical venue selection with deep reasoning
Validator	Google	`gemini-2.5-pro-latest`	Jun 2025	60s	JSON validation + value-per-minute ranking

Model Parameters (Environment-Driven)

```
# Claude Sonnet 4.5 (Strategist)
CLAUDE_MODEL=claude-sonnet-4-5-20250929
CLAUDE_TIMEOUT_MS=12000
# Uses: temperature=0.0 (most deterministic), max_tokens=500

# GPT-5 (Planner)
OPENAI_MODEL=gpt-5
GPT5_TIMEOUT_MS=120000
GPT5_REASONING_EFFORT=high
# ⚠ GPT-5 does NOT support: temperature, top_p, frequency_penalty, presence_penalty
# Must use: reasoning_effort (minimal|low|medium|high), max_completion_tokens

# Gemini 2.5 Pro (Validator)
GEMINI_MODEL=gemini-2.5-pro-latest
GEMINI_TIMEOUT_MS=60000
# Uses: temperature=0.2, maxOutputTokens=2048

# Total Budget
LLM_TOTAL_BUDGET_MS=200000  # 200 seconds total
```

Critical Model Risks

RISK #1: Post-Training-Cutoff Models

Issue: GPT-5 (released Aug 2025) and Claude Sonnet 4.5 (released Sep 2025) are NEWER than most AI assistant training data cutoffs (typically April 2024 - June 2025).

Impact: - AI assistants may not know these models exist - May suggest deprecated alternatives (gpt-4, claude-opus-3) - Parameter constraints may be incorrect (e.g., suggesting `temperature` for GPT-5)

Mitigation: 1. **Trust MODEL.md over AI suggestions** - This document is manually verified against live APIs 2. **Run monthly model research** - `node tools/research/model-discovery.mjs` (uses Perplexity AI to fetch latest docs) 3. **API verification tests** - Curl tests in MODEL.md confirm each model works 4. **Model assertion in adapters** - Code validates response echoes requested model

Verification Command:

```
# Test GPT-5 is working
curl -X POST "https://api.openai.com/v1/chat/completions" \
  -H "Authorization: Bearer $OPENAI_API_KEY" \
  -H "Content-Type: application/json" \
  -d '{
    "model": "gpt-5",
    "messages": [{"role": "user", "content": "ping"}],
    "reasoning_effort": "medium",
    "max_completion_tokens": 64
  }'

# Expected: {"model":"gpt-5",...}
# If 404: Model ID is wrong or not available to your API key
```

RISK #2: Silent Model Swaps

Issue: API providers may return different model than requested without error (e.g., requesting `gpt-5` but getting `gpt-4o`).

Impact: ML training data corruption (don't know which model generated output).

Mitigation: - **Model assertion in all adapters**:

```
// server/lib/adapters/openai-gpt5.js
const responseModel = data.model;
if (responseModel !== requestedModel) {
  throw new Error(`Model mismatch: requested ${requestedModel}, got ${responseModel}`);
}
```

RISK #3: Parameter Incompatibility

Issue: GPT-5 deprecated `temperature` and `top_p` parameters. Using them causes API errors.

Impact: Recommendation pipeline fails if old parameters used.

Mitigation: - **Adapter-specific parameter validation**:

```
// ✅ CORRECT for GPT-5
{ reasoning_effort: 'high', max_completion_tokens: 32000 }

// ❌ WRONG for GPT-5 (will error)
{ temperature: 0.7, top_p: 0.95 }
```

Model Monitoring

Monthly Verification Checklist: - [ ] Run `node tools/research/model-discovery.mjs` - [ ] Review `tools/research/model-research-YYYY-MM-DD.json` - [ ] Update MODEL.md with any changes - [ ] Test each model via curl (examples in MODEL.md) - [ ] Update adapters if parameter constraints changed - [ ] Update .env.example with new defaults

---

🌐 External API Constraints

Rate Limits & Quotas

Google Maps Platform APIs

Geocoding API: - **Free Tier:** 40,000 requests/month - **Cost:** $5.00 per 1,000 requests (after free tier) - **Rate Limit:** 50 requests/second - **Usage:** Location resolution, reverse geocoding - **Mitigation:** Cache coordinates in `places` table

Places API (Find Place, Details): - **Free Tier:** None (pay-as-you-go) - **Cost:** - Find Place: $17 per 1,000 requests - Place Details (Basic): $17 per 1,000 requests - Place Details (Hours): Additional $5 per 1,000 requests - **Rate Limit:** 100 queries/second - **Usage:** Venue validation, business hours - **Mitigation:** Cache hours in `places_cache` table (24-hour TTL)

Routes API (Directions): - **Free Tier:** $200 credit/month (~7,500 requests) - **Cost:** $10.00 per 1,000 requests (TRAFFIC_AWARE mode) - **Rate Limit:** 100 elements/second - **Usage:** Traffic-aware distance & ETA - **Mitigation:** - Use only for final rankings (not shortlist) - No Haversine fallback (fail-fast on quota exhaustion)

Timezone API: - **Free Tier:** 40,000 requests/month - **Cost:** $5.00 per 1,000 requests (after free tier) - **Rate Limit:** 100 requests/second - **Usage:** Timezone resolution for snapshots - **Mitigation:** Cache timezone per GPS precision (H3 resolution 8)

Air Quality API: - **Free Tier:** $200 credit/month (~40,000 requests) - **Cost:** $5.00 per 1,000 requests - **Rate Limit:** 10 requests/second - **Usage:** AQI for context snapshots - **Mitigation:** Single request per snapshot (not per venue)

AI Model APIs

Anthropic Claude (Sonnet 4.5): - **Rate Limits:** - Tier 1: 50 requests/minute, 40K tokens/minute - Tier 2: 1,000 requests/minute, 100K tokens/minute - Tier 3: 2,000 requests/minute, 400K tokens/minute - **Cost:** $3.00 per 1M input tokens, $15.00 per 1M output tokens - **Usage:** Strategic overview (2-3 sentences, ~150 tokens output) - **Mitigation:** Background job

queue (non-blocking)

OpenAI GPT-5: - **Rate Limits:** - Tier 1: 500 requests/minute, 30K tokens/minute - Tier 2: 5,000 requests/minute, 450K tokens/minute - **Cost:** $10.00 per 1M input tokens, $30.00 per 1M output tokens - **Reasoning Cost:** ~5x higher token usage (includes reasoning tokens) - **Usage:** Tactical venue selection (~1,000 output tokens) - **Mitigation:** - Single request per recommendation (no retries on success) - Timeout at 120 seconds (fail-fast)

Google Gemini 2.5 Pro: - **Rate Limits:** - Free Tier: 15 requests/minute, 1M tokens/minute - Paid Tier: 360 requests/minute, 4M tokens/minute - **Cost:** $1.25 per 1M input tokens, $5.00 per 1M output tokens - **Usage:** JSON validation + ranking (~500 output tokens) - **Mitigation:** Final stage only (post-validation)

Weather & Aviation APIs

OpenWeather API: - **Free Tier:** 1,000 calls/day, 60 calls/minute - **Cost:** - Startup: $40/month (100K calls) - Developer: $120/month (1M calls) - **Usage:** Weather conditions for snapshots - **Mitigation:** Single request per snapshot creation

FAA ASWS (Airport Status Web Service): - **Free Tier:** Yes (public service) - **Rate Limit:** Not officially documented (~10 requests/second) - **Availability:** ~95% uptime (government API) - **Usage:** Airport delay context - **Mitigation:** - Fallback to preserve proximity without delay data (Issue #29) - Non-blocking (optional enrichment)

Cost Optimization Strategies

1. Database Caching

- **Place Coordinates:** Cache in `places` table (permanent)
- **Business Hours:** Cache in `places_cache` table (24-hour TTL)
- **Weather Data:** No caching (real-time requirement)
- **Timezone:** Cache per H3 hex (permanent)

Savings: - Geocoding: 80% reduction (only new venues) - Places API: 60% reduction (cached hours) - Routes API: No reduction (must be real-time)

2. Request Batching

- **Parallel API Calls:** Location resolution (Geocoding + Timezone + Weather + AQ)
- **Sequential Venue Resolution:** Geocode → Places → Routes (fail early)

3. Quota Monitoring

- **Daily Check:** Google Cloud Console quota dashboard
- **Alert Threshold:** 80% of monthly quota
- **Fallback Plan:** Graceful degradation (clear error messages)

4. Model Selection Strategy

- **Claude Sonnet 4.5:** Most cost-effective for strategy ($3/$15 per 1M tokens)
- **GPT-5:** Most expensive but required for reasoning ($10/$30 + reasoning overhead)
- **Gemini 2.5 Pro:** Cheapest validator ($1.25/$5 per 1M tokens)

Total Cost Per Recommendation (Estimated): - Claude: ~$0.002 (150 tokens × $0.000015) - GPT-5: ~$0.030 (1,000 tokens × $0.000030) - Gemini: ~$0.003 (500 tokens × $0.000005) - Google Maps: ~$0.040 (4 API calls × ~$0.010) - **Total: ~$0.075 per recommendation**

At $9.99/month subscription: - Break-even: ~133 recommendations/month (~4 per day) - Target: 200 recommendations/month for 40% margin

---

API Error Handling Patterns

Circuit Breaker Pattern

```
// server/lib/circuit-breaker.js
class CircuitBreaker {
  constructor(threshold = 5, timeout = 60000) {
    this.failureCount = 0;
    this.threshold = threshold;
    this.timeout = timeout;
    this.state = 'CLOSED'; // CLOSED | OPEN | HALF_OPEN
  }

  async execute(fn) {
    if (this.state === 'OPEN') {
      throw new Error('Circuit breaker OPEN - service unavailable');
    }

    try {
      const result = await fn();
      this.onSuccess();
      return result;
    } catch (error) {
      this.onFailure();
      throw error;
    }
  }

  onSuccess() {
    this.failureCount = 0;
    this.state = 'CLOSED';
  }

  onFailure() {
    this.failureCount++;
    if (this.failureCount >= this.threshold) {
      this.state = 'OPEN';
      setTimeout(() => {
        this.state = 'HALF_OPEN';
        this.failureCount = 0;
      }, this.timeout);
    }
  }
}
```

Used for: - Google Maps APIs (5 failures → 60s cooldown) - OpenWeather API (5 failures → 60s cooldown) - Not used for AI models (fail-fast, no retries)

Retry Strategy

```
// server/util/retry.js
async function retryWithBackoff(fn, maxRetries = 3) {
  for (let i = 0; i < maxRetries; i++) {
    try {
      return await fn();
    } catch (error) {
      if (i === maxRetries - 1) throw error;

      const isRetryable =
        error.status === 429 || // Rate limit
        error.status === 502 || // Bad gateway
        error.status === 503 || // Service unavailable
        error.status === 504;   // Gateway timeout

      if (!isRetryable) throw error;

      const delay = Math.min(1000 * 2 ** i, 30000); // Exponential backoff (max 30s)
      await new Promise(resolve => setTimeout(resolve, delay));
    }
  }
}
```

Applied to: - Google Maps API errors (429, 502, 503, 504) - Database replication lag (action logging) - Not applied to: AI model errors (fail-fast)

□ Database Schema & Field Mapping

Table Relationships

```
snapshots (context)
  ├──→ strategies (1:1) - Claude's strategic overview
  ├──→ rankings (1:N) - Each snapshot can have multiple recommendation sets
  └──→ actions (1:N) - User interactions tied to snapshot context

rankings (recommendation sets)
  ├──→ ranking_candidates (1:N) - Individual venues in the ranking (6 per ranking)
  └──→ venue_feedback (1:N) - Thumbs up/down per ranking

ranking_candidates (venues)
  ├──→ actions (1:N) - User interactions with this specific venue
  └──→ venue_catalog (N:1) - Links to master venue record via place_id

venue_catalog (master venue list)
  ├──→ venue_metrics (1:1) - Aggregated stats (times_recommended, positive_feedback)
  ├──→ llm_venue_suggestions (1:N) - GPT-5 suggested this, validation status
  └──→ places_cache (1:1) - Google Places hours cache via place_id
```

Complete Field Mapping

snapshots (Context Records)

Field	Type	Source	Purpose	Required
snapshot_id	UUID	Client (crypto.randomUUID)	Primary key, links to all downstream tables	□
created_at	TIMESTAMPTZ	Client (new Date().toISOString())	Snapshot creation time (UTC)	□
user_id	UUID	Client (localStorage or null)	Anonymous user tracking	□
device_id	UUID	Client (localStorage)	Device fingerprint	□
session_id	UUID	Client (rotates after 30min)	Session tracking	□
lat	DOUBLE PRECISION	Browser Geolocation API	GPS latitude	□
lng	DOUBLE PRECISION	Browser Geolocation API	GPS longitude	□
accuracy_m	DOUBLE PRECISION	Browser Geolocation API	GPS accuracy in meters	□
coord_source	TEXT	Client ('gps' or 'manual_city_search')	How coords were obtained	□
city	TEXT	Google Geocoding API	Resolved city name	□
state	TEXT	Google Geocoding API	State abbreviation (e.g., "TX")	□
country	TEXT	Google Geocoding API	Country name	□

Field	Type	Source	Purpose	Required
formatted_address	TEXT	Google Geocoding API	Full address string	✅
timezone	TEXT	Google Timezone API	IANA timezone (e.g., "America/Chicago")	✅
local_iso	TIMESTAMP	Calculated from UTC + timezone	Local time (no timezone offset)	✅
dow	INTEGER	Calculated (0=Sunday, 6=Saturday)	Day of week for ML features	✅
hour	INTEGER	Calculated (0-23)	Hour of day for ML features	✅
day_part_key	TEXT	Calculated (morning/afternoon/evening)	Daypart classification	✅
h3_r8	TEXT	Calculated via h3-js (latLngToCell)	Geospatial hex index (resolution 8 = ~0.46 km²)	✅
weather	JSONB	OpenWeather API	{tempF, conditions, description}	✅
air	JSONB	Google AQ API	{aqi, category, dominantPollutant}	✅
airport_context	JSONB	FAA ASWS API	{airport_code, distance_miles, delay_minutes}	✅
device	JSONB	Client (navigator.userAgent)	{ua, platform}	✅
permissions	JSONB	Client (navigator.permissions)	{geolocation: 'granted'}	✅
extras	JSONB	Reserved	Future metadata	✅

Indexes:

```
CREATE INDEX idx_snapshots_user_id ON snapshots(user_id);
CREATE INDEX idx_snapshots_h3_r8 ON snapshots(h3_r8);
CREATE INDEX idx_snapshots_created_at ON snapshots(created_at);
```

strategies (Claude Strategic Overviews)

Field	Type	Source	Purpose	Required
id	UUID	Auto-generated	Primary key	□
snapshot_id	UUID	FK to snapshots.snapshot_id	Links strategy to context	□
correlation_id	UUID	Request tracing ID	Debugging and log correlation	□
strategy	TEXT	Claude Sonnet 4.5 API response	2-3 sentence strategic overview	□ (null while pending)
status	TEXT	Job queue (‘pending’, ‘ok’, ‘failed’)	Generation status	□
error_code	INTEGER	HTTP status if failed	Error classification	□
error_message	TEXT	Error details	Debugging	□
attempt	INTEGER	Retry counter (default 1)	Retry tracking	□
latency_ms	INTEGER	Response time	Performance monitoring	□
tokens	INTEGER	Claude token usage	Cost tracking	□
next_retry_at	TIMESTAMPTZ	Scheduled retry time	Exponential backoff	□
created_at	TIMESTAMPTZ	Auto (defaultNow)	Record creation	□
updated_at	TIMESTAMPTZ	Auto (defaultNow)	Last status change	□
model_name	TEXT	‘claude-sonnet-4-5-20250929’	Model version tracking	□
model_params	JSONB	{temperature: 0.0, max_tokens: 500}	A/B testing	□
prompt_version	TEXT	Prompt template version	Prompt iteration tracking	□

Unique Constraint:

```
ALTER TABLE strategies ADD CONSTRAINT strategies_snapshot_id_unique UNIQUE(snapshot_id);
```

rankings (Recommendation Sets)

Field	Type	Source	Purpose	Required
ranking_id	UUID	Server (crypto.randomUUID)	Primary key	✅
created_at	TIMESTAMPTZ	Server (new Date())	Recommendation time	✅
snapshot_id	UUID	FK to snapshots.snapshot_id	Links to context	✅
correlation_id	UUID	Request tracing	End-to-end trace	✅
user_id	UUID	From snapshot	User tracking	✅
city	TEXT	From snapshot	City for this ranking	✅
ui	JSONB	{maxDistance, filters}	UI state at request time	✅
model_name	TEXT	'claude-sonnet-4-5→gpt-5→gemini-2.5-pro'	Triad pipeline version	✅
scoring_ms	INTEGER	Performance metric	Scoring engine latency	✅
planner_ms	INTEGER	Performance metric	GPT-5 latency	✅
total_ms	INTEGER	Performance metric	End-to-end latency	✅
timed_out	BOOLEAN	Error tracking	Pipeline timeout flag	✅
path_taken	TEXT	'triad' or 'fallback'	Which pipeline executed	✅

ranking_candidates (Individual Venues)

Field	Type	Source	Purpose	Required
id	UUID	Auto-generated	Primary key	✅
ranking_id	UUID	FK to rankings.ranking_id	Links to recommendation set	✅
block_id	TEXT	Legacy field (venue name)	Backward compatibility	✅
place_id	TEXT	Google Places API	Stable Google identifier	⚠ Highly Recommended
name	TEXT	GPT-5 or Google Places	Venue display name	✅
lat	DOUBLE PRECISION	Google Geocoding/Places API	Venue latitude	✅
lng	DOUBLE PRECISION	Google Geocoding/Places API	Venue longitude	✅
drive_time_min	INTEGER	Legacy field	Deprecated (use drive_minutes)	✅
drive_minutes	INTEGER	Google Routes API	Traffic-aware drive time	⚠ Recommended
distance_miles	DOUBLE PRECISION	Google Routes API (distanceMeters/1609.344)	Road distance (NOT Haversine)	⚠ Recommended
estimated_distance_miles	DOUBLE PRECISION	Workflow trace	Pre-validation estimate	✅

Field	Type	Source	Purpose	Required
distance_source	TEXT	'routes_api' or 'haversine'	How distance was calculated	□
straight_line_km	DOUBLE PRECISION	Haversine formula	As-the-crow-flies distance	□
est_earnings_per_ride	DOUBLE PRECISION	Gemini validation	Estimated earnings for this trip	□
value_per_min	DOUBLE PRECISION	Calculated: earnings/(drive+trip+wait)	Earnings per minute metric	⚠ Recommended
value_grade	TEXT	'A', 'B', 'C', 'D', 'F'	Letter grade for value	□
not_worth	BOOLEAN	value_per_min < 0.50 threshold	Low-value flag	□
rate_per_min_used	DOUBLE PRECISION	Calculation input	Rate assumption used	□
trip_minutes_used	INTEGER	Calculation input	Trip duration assumption	□
wait_minutes_used	INTEGER	Calculation input	Wait time assumption	□
model_score	DOUBLE PRECISION	GPT-5 raw score	Model confidence	□
rank	INTEGER	Gemini ranking (1-6)	Display position	□
exploration_policy	TEXT	'none' or 'epsilon_greedy'	Exploration strategy	□
epsilon	DOUBLE PRECISION	Exploration parameter	Random exploration rate	□
was_forced	BOOLEAN	Exploration bumped this up	Exploration flag	□
propensity	DOUBLE PRECISION	P(shown | context)	Inverse propensity weighting for ML	□
features	JSONB	Model features	ML feature vector	□
h3_r8	TEXT	Venue geospatial hex	Spatial index	□
snapshot_id	UUID	FK to snapshots.snapshot_id	Direct link to context	□

Indexes (Performance Critical):

```
CREATE INDEX idx_ranking_candidates_ranking_id ON ranking_candidates(ranking_id);
CREATE INDEX idx_ranking_candidates_snapshot_id ON ranking_candidates(snapshot_id);
CREATE INDEX idx_ranking_candidates_place_id ON ranking_candidates(place_id);
```

actions (User Interactions)

Field	Type	Source	Purpose	Required
action_id	UUID	Auto-generated	Primary key	✓
created_at	TIMESTAMPTZ	Server timestamp	Action time	✓
ranking_id	UUID	FK to rankings.ranking_id	Which recommendation set	✓
snapshot_id	UUID	FK to snapshots.snapshot_id	Context at action time	✓
user_id	UUID	From snapshot	User tracking	✓
action	TEXT	'view', 'dwell', 'block_clicked', 'block_dismissed'	Action type	✓
block_id	TEXT	Venue name or place_id	Which venue	✓
dwell_ms	INTEGER	Frontend timer	Dwell time in milliseconds	✓
from_rank	INTEGER	Position clicked (1-6)	Ranking position	✓
raw	JSONB	Extra metadata	Future expansion	✓

Indexes:

```
CREATE INDEX idx_actions_snapshot_id ON actions(snapshot_id);
CREATE INDEX idx_actions_ranking_id ON actions(ranking_id);
```

venue_catalog (Master Venue List)

Field	Type	Source	Purpose	Required
venue_id	UUID	Auto-generated	Primary key	□
place_id	TEXT	Google Places API	Google stable ID (UNIQUE)	⚠ Highly Recommended
name	TEXT	Seed data or GPT-5	Display name	□
address	TEXT	Google Places/Geocoding	Full address	□
lat	DOUBLE PRECISION	Google Places/Geocoding	Venue latitude	□
lng	DOUBLE PRECISION	Google Places/Geocoding	Venue longitude	□
category	TEXT	Seed data classification	'airport', 'entertainment', 'dining', etc.	□
dayparts	TEXT[]	Seed data	['morning', 'afternoon', 'evening', 'all_day']	□
staging_notes	JSONB	Seed data	{pickup_zone, approach_roads, avoid_zones}	□
city	TEXT	Extracted from address	City name	□
metro	TEXT	Seed data	Metro area (e.g., "DFW")	□
ai_estimated_hours	TEXT	Deprecated	Old hours format	□
business_hours	JSONB	Google Places Details	Full hours array	□
discovery_source	TEXT	'seed', 'llm_suggested', 'user_added'	How venue entered catalog	□
validated_at	TIMESTAMPTZ	Places API call time	Last validation timestamp	□
suggestion_metadata	JSONB	GPT-5 reasoning	Why model suggested this	□
created_at	TIMESTAMPTZ	Auto (defaultNow)	Record creation	□
last_known_status	TEXT	Google Places Details	'open', 'closed', 'temporarily_closed', 'permanently_closed'	□
status_checked_at	TIMESTAMPTZ	Last Places API check	Staleness tracking	□
consecutive_closed_checks	INTEGER	Failure counter	Auto-suppression trigger	□
auto_suppressed	BOOLEAN	Suppression flag	Hide from recommendations	□
suppression_reason	TEXT	Why suppressed	Debugging	□

venue_metrics (Aggregated Stats)

Field	Type	Source	Purpose	Required
venue_id	UUID	FK to venue_catalog.venue_id	Primary key	□
times_recommended	INTEGER	Count from ranking_candidates	Exposure tracking	□
times_chosen	INTEGER	Count from actions WHERE action='block_clicked'	Click-through rate	□
positive_feedback	INTEGER	Count from venue_feedback WHERE sentiment='up'	Thumbs up	□
negative_feedback	INTEGER	Count from venue_feedback WHERE sentiment='down'	Thumbs down	□
reliability_score	DOUBLE PRECISION	Calculated: positive/(positive+negative)	Quality metric	□
last_verified_by_driver	TIMESTAMPTZ	Latest venue_feedback timestamp	Freshness indicator	□

□ Testing Strategy

1. Unit Testing (Future)

Framework: Vitest
Coverage Target: 80% for lib/ functions

Priority Test Suites: - `lib/scoring-engine.js` - Venue ranking logic - `lib/driveTime.js` - Drive time prediction - `util/validate-snapshot.js` - Snapshot validation - `lib/exploration.js` - Exploration policy

2. Integration Testing (Current)

Framework: Manual curl + Node.js scripts

Test Scenarios:

```
# Test snapshot creation
curl -X POST http://localhost:5000/api/location/snapshot \
  -H "Content-Type: application/json" \
  -d @tests/fixtures/snapshot-minimal.json

# Test blocks recommendation
curl -X POST http://localhost:5000/api/blocks \
  -H "X-Snapshot-Id: <uuid>" \
  -H "Content-Type: application/json" \
  -d '{"userId":"test-user"}'

# Test strategy generation
curl http://localhost:5000/api/blocks/strategy/<snapshot-id>
```

3. End-to-End Testing (Current)

Tool: Browser DevTools + Manual QA

Test Flow: 1. Open app → GPS permission granted 2. Wait for snapshot creation 3. Navigate to Co-Pilot tab 4. Verify 6 venue cards appear 5. Click "Navigate" → Opens Google Maps 6. Check database: `actions` table has 'block_clicked' row

4. Performance Testing (Future)

Target Metrics: - Snapshot creation: <3 seconds (parallel API calls) - Strategy generation: <12 seconds (Claude timeout) - Blocks recommendation: <200 seconds total (triad pipeline) - Database query latency: <100ms (with indexes)

5. Model Verification Testing (Monthly)

Script: `node tools/research/model-discovery.mjs`

Validates: - Model IDs still exist - Parameter constraints haven't changed - Pricing hasn't increased significantly - New flagship models released

6. Database Migration Testing

Process: 1. Run migration on local dev database 2. Verify schema changes via `pg_dump` 3. Test rollback script 4. Run migration on staging 5. Production migration during low-traffic window

---

⚠ Known Risks & Mitigations

RISK MATRIX

Risk ID	Category	Severity	Likelihood	Mitigation Status
R1	Post-training-cutoff models	HIGH	HIGH	□ Documented + monthly verification
R2	Silent model swaps	MEDIUM	LOW	□ Model assertion in adapters
R3	Google API quota exhaustion	HIGH	MEDIUM	□ Circuit breakers + fail-fast
R4	Database replication lag	MEDIUM	MEDIUM	□ Retry logic with exponential backoff
R5	GPT-5 timeout (5min limit)	HIGH	MEDIUM	□ **ACTIVE ISSUE** - See Issue #22-27
R6	Hard-coded timezone fallbacks	LOW	LOW	□ FIXED - Issue #21
R7	Missing database indexes	MEDIUM	LOW	□ Added FK indexes - Issue #28
R8	Venue catalog staleness	MEDIUM	MEDIUM	□ Auto-suppression logic planned
R9	Client-side GPS denial	LOW	HIGH	□ Clear error message + refresh prompt
R10	FAA API unavailability	LOW	MEDIUM	□ Graceful degradation (no airport context)

DETAILED RISK ANALYSIS

R1: Post-Training-Cutoff Models

Description: GPT-5 (Aug 2025) and Claude Sonnet 4.5 (Sep 2025) are newer than most AI assistant training cutoffs.

Impact if Unmitigated: - AI assistants suggest wrong model IDs (`gpt-4` instead of `gpt-5`) - Parameter errors (using `temperature` for GPT-5 causes API errors) - Developer confusion from conflicting documentation

Mitigation Strategy: 1. **Primary:** Trust MODEL.md over AI suggestions (manually verified against live APIs) 2. **Secondary:** Monthly research via Perplexity AI (`node tools/research/model-discovery.mjs`) 3. **Tertiary:** API verification tests (curl examples in MODEL.md)

Residual Risk: LOW (requires manual diligence to run monthly verification)

R5: GPT-5 Timeout (5-Minute Hard Limit)

Description: GPT-5 has hard-coded 5-minute timeout (300,000ms) despite environment variable `PLANNER_DEADLINE_MS=120000` (2 minutes).

Evidence:

```
// server/lib/gpt5-tactical-planner.js:183
setTimeout(() => {
  controller.abort();
}, 300000); // 5 min timeout - HARDCODED!
```

Impact: - Environment variable `PLANNER_DEADLINE_MS` is ignored - Requests can hang for up to 5 minutes - Affects user experience (long wait times)

Fix Required:

```
// ✅ CORRECT - Use environment variable
const timeout = parseInt(process.env.PLANNER_DEADLINE_MS || '120000', 10);
setTimeout(() => {
  controller.abort();
}, timeout);
```

Status: ⚠️ ACTIVE ISSUE - Tracked in ISSUES.md #22, #27

R8: Venue Catalog Staleness

Description: Venues can close permanently but remain in catalog.

Impact: - Recommending closed venues wastes driver time - Negative user experience - Low reliability_score over time

Planned Mitigation: 1. **Field:** `consecutive_closed_checks` counter (increments on each “permanently_closed” status) 2. **Logic:** If `consecutive_closed_checks >= 3` , set `auto_suppressed = true` 3. **Filter:** Exclude `auto_suppressed = true` from shortlist generation 4. **Manual Override:** Admin can clear suppression flag if venue reopens

Status: 📋 PLANNED (Issue #32 in ISSUES.md)

🛡️ AI Development Guardrails

Rules for AI-Assisted Development

These constraints prevent common mistakes when using AI assistants to modify this codebase:

GUARDRAIL #1: Never Remove Error Handling

❌ **WRONG:** “This try-catch is too verbose, let’s remove it”
✅ **CORRECT:** Keep all error handling, improve error messages if needed

Rationale: We fail-fast with clear errors, never silently swallow exceptions.

GUARDRAIL #2: Never Add Fallbacks to Triad Pipeline

❌ **WRONG:** “If GPT-5 fails, let’s fall back to GPT-4”
✅ **CORRECT:** Fail the entire request, log the error, return 500 with details

Rationale: Single-path triad preserves ML training data integrity.

GUARDRAIL #3: Never Hardcode Location Data

 WRONG: `if (city === "Dallas") { /* special logic */ }`
 CORRECT: `const cityData = await db.select().from(venue_catalog).where(eq(...));`

Rationale: Zero hardcoding policy - all data from DB or env vars.

GUARDRAIL #4: Never Use Old Model IDs

 WRONG: `model: "gpt-4"` or `model: "claude-opus-3"`
 CORRECT: `model: process.env.OPENAI_MODEL` (reads `gpt-5` from .env)

Rationale: Post-training-cutoff models must be environment-driven.

GUARDRAIL #5: Never Skip Database Indexes

 WRONG: Adding foreign key without index
 CORRECT:

```
ALTER TABLE ranking_candidates ADD FOREIGN KEY (snapshot_id) REFERENCES snapshots(snapshot_id);
CREATE INDEX idx_ranking_candidates_snapshot_id ON ranking_candidates(snapshot_id);
```

Rationale: Foreign keys without indexes cause slow queries at scale.

GUARDRAIL #6: Never Disable GPS Validation

 WRONG: Accepting incomplete snapshots to “make it work”
 CORRECT: Return `400 Bad Request` with `refresh_required` status

Rationale: Incomplete snapshots corrupt ML training data.

GUARDRAIL #7: Always Verify AI Suggestions Against Live APIs

 WRONG: Trust AI assistant’s model parameter suggestions
 CORRECT: Cross-reference with MODEL.md and test via curl

Rationale: AI training data is outdated for 2025 models.

Quick Start Implementation Guides

Guide 1: Adding a New API Endpoint

Scenario: You want to add a new endpoint to fetch user’s recommendation history.

Step-by-Step:

1. **Define the database query (if needed)**

```
// server/util/db.js
export async function getUserRecommendationHistory(userId, limit = 20) {
  const result = await db
    .select({
      ranking_id: rankings.ranking_id,
      created_at: rankings.created_at,
      city: rankings.city,
      venue_count: sql<number>`count(${rankingCandidates.id})`,
    })
    .from(rankings)
    .leftJoin(rankingCandidates, eq(rankings.ranking_id, rankingCandidates.ranking_id))
    .where(eq(rankings.user_id, userId))
    .groupBy(rankings.ranking_id, rankings.created_at, rankings.city)
    .orderBy(desc(rankings.created_at))
    .limit(limit);

  return result;
}
```

2. **Create the route handler**

```
// server/routes/history.js
import express from 'express';
import { getUserRecommendationHistory } from '../util/db.js';

const router = express.Router();

router.get('/history/:userId', async (req, res) => {
  try {
    const { userId } = req.params;
    const limit = parseInt(req.query.limit) || 20;

    const history = await getUserRecommendationHistory(userId, limit);

    res.json({
      user_id: userId,
      total: history.length,
      recommendations: history,
    });
  } catch (error) {
    console.error('[GET /api/history/:userId] Error:', error);
    res.status(500).json({
      error: 'Failed to fetch recommendation history',
      details: error.message,
    });
  }
});

export default router;
```

3. **Register the route in the main app**

```
// server/index.js
import historyRoutes from './routes/history.js';

// ... other imports

app.use('/api', historyRoutes);
```

4. **Test with curl**

```
curl http://localhost:5000/api/history/ce372d10-30b7-4f14-af77-d1cada207d27?limit=10
```

5. **Create frontend hook (optional)**

```
// client/src/hooks/useRecommendationHistory.ts
import { useQuery } from '@tanstack/react-query';
import { apiRequest } from '@/lib/queryClient';

export function useRecommendationHistory(userId: string, limit = 20) {
  return useQuery({
    queryKey: ['/api/history', userId, limit],
    queryFn: () => apiRequest(`/api/history/${userId}?limit=${limit}`),
    enabled: !!userId,
  });
}
```

Guide 2: Adding a New Database Field

Scenario: You want to add a `favorite` boolean flag to venue recommendations.

Step-by-Step:

1. **Update the Drizzle schema**

```ts
// shared/schema.ts
export const rankingCandidates = pgTable('ranking_candidates', {
  // ... existing fields

  favorite: boolean('favorite').default(false),  // NEW FIELD
});
```

2. **Push schema changes to database**

```
npm run db:push
# If there's a data-loss warning:
npm run db:push --force
```

3. **Update insert schema (if needed for validation)**

```ts
// shared/schema.ts
export const insertRankingCandidateSchema = createInsertSchema(rankingCandidates)
  .omit({ id: true })
  .extend({
    favorite: z.boolean().optional(),
  });
```

4. **Update the persistence layer**

```ts
// server/lib/persist-ranking.js
const candidateRow = {
  // ... existing fields
  favorite: candidate.favorite || false,  // Include new field
};
```

5. **Update frontend types (if using TypeScript)**

```ts
// client/src/types/venue.ts
export type Venue = {
  // ... existing fields
  favorite?: boolean;
};
```

6. **Verify the change**

```
# Check database schema
psql $DATABASE_URL -c "\d ranking_candidates"

# Should show:
#  favorite | boolean | | default false
```

Guide 3: Modifying the Triad Pipeline

Scenario: You want to add a new AI model (e.g., Gemini as reranker instead of validator).

Step-by-Step:

1. **Create adapter for the new model**

```
// server/lib/adapters/google-gemini-reranker.js
import { GoogleGenerativeAI } from '@google/generative-ai';

const genAI = new GoogleGenerativeAI(process.env.GOOGLE_GENERATIVE_AI_API_KEY);

export async function rerankVenues(venues, context) {
  const model = genAI.getGenerativeModel({
    model: process.env.GEMINI_MODEL
  });

  const prompt = `Rerank these venues based on driver earnings potential...`;

  const result = await model.generateContent(prompt);
  const text = result.response.text();

  return JSON.parse(text);
}
```

2. **Update environment variables**

```
# .env
GEMINI_MODEL=gemini-2.5-pro-latest
GEMINI_RERANK_TIMEOUT_MS=30000
```

3. **Integrate into Triad pipeline**

```
// server/routes/blocks.js

// Step 3: Gemini Reranking (NEW)
const rerankedVenues = await callGeminiReranker(enrichedVenues, snapshot);

// Update model_name to reflect new pipeline
const modelName = 'claude-sonnet-4-5→gpt-5→gemini-2.5-pro-rerank';
```

4. **Update MODEL.md documentation**

```
### Gemini 2.5 Pro (Reranker)
- **Purpose:** Rerank venues based on earnings potential
- **Timeout:** 30 seconds
- **Input:** Enriched venues + context
- **Output:** Reranked venues array
```

5. **Update ARCHITECTUREV2.md**

- Add new model to “AI Model Strategy” section
- Update data flow diagram
- Add to Decision Log with rationale

6. **Test the new pipeline**

```
# Make recommendation request
curl -X POST http://localhost:5000/api/blocks \
  -H "X-Snapshot-Id: <uuid>" \
  -H "Content-Type: application/json" \
  -d '{"userId":"test-user"}'

# Verify model_name in response
# Expected: "claude-sonnet-4-5→gpt-5→gemini-2.5-pro-rerank"
```

Guide 4: Adding a New React Page

Scenario: You want to add an Analytics dashboard page.

Step-by-Step:

1. **Create the page component**

```
// client/src/pages/analytics.tsx
import { useQuery } from '@tanstack/react-query';
import { Card, CardHeader, CardTitle, CardContent } from '@/components/ui/card';

export default function AnalyticsPage() {
  const statsQuery = useQuery({
    queryKey: ['/api/analytics/stats'],
    queryFn: () => fetch('/api/analytics/stats').then(r => r.json()),
  });

  if (statsQuery.isLoading) {
    return <div>Loading analytics...</div>;
  }

  return (
    <div className="container mx-auto p-4">
      <h1 className="text-2xl font-bold mb-4">Analytics Dashboard</h1>

      <div className="grid grid-cols-1 md:grid-cols-3 gap-4">
        <Card>
          <CardHeader>
            <CardTitle>Total Recommendations</CardTitle>
          </CardHeader>
          <CardContent>
            <p className="text-3xl">{statsQuery.data?.total_recommendations}</p>
          </CardContent>
        </Card>

        {/* Add more cards */}
      </div>
    </div>
  );
}
```

2. **Register the route**

```
// client/src/App.tsx
import AnalyticsPage from './pages/analytics';

function App() {
  return (
    <Switch>
      {/* ... existing routes */}

      <Route path="/analytics">
        <AnalyticsPage />
      </Route>
    </Switch>
  );
}
```

3. **Add navigation link**

```
// client/src/components/LocationHeader.tsx
<Link href="/analytics">
  <Button variant="ghost">Analytics</Button>
</Link>
```

4. **Create backend endpoint**

```
// server/routes/analytics.js
import express from 'express';
const router = express.Router();

router.get('/analytics/stats', async (req, res) => {
  const stats = await db.select({
    total_recommendations: sql<number>`count(distinct ${rankings.ranking_id})`,
    total_actions: sql<number>`count(${actions.action_id})`,
    avg_venues_per_ranking: sql<number>`avg(venue_count)`,
  })
  .from(rankings)
  .leftJoin(actions, eq(rankings.ranking_id, actions.ranking_id))
  .execute();

  res.json(stats[0]);
});

export default router;
```

5. **Register analytics routes**

```
// server/index.js
import analyticsRoutes from './routes/analytics.js';
app.use('/api', analyticsRoutes);
```

Guide 5: Debugging Common Issues

Issue 1: “GPS Required” Stuck on Frontend

Diagnosis:

```
// Check LocationContext state mutation
console.log('Setting coords:', coords);
```

Fix:

```
// ✅ CORRECT - Create new object reference
setLocationState(prev => ({
  ...prev,
  coords: { ...coords }  // NEW object
}));

// ❌ WRONG - Mutates existing object
setLocationState(prev => {
  prev.coords = coords;
  return prev;
});
```

Issue 2: GPT-5 Timeout Ignored

Diagnosis:

```
# Check if environment variable is set
echo $PLANNER_DEADLINE_MS
```

Fix:

```
// server/lib/gpt5-tactical-planner.js

// ❌ WRONG - Hardcoded timeout
setTimeout(() => controller.abort(), 300000);

// ✅ CORRECT - Use environment variable
const timeout = parseInt(process.env.PLANNER_DEADLINE_MS || '120000', 10);
setTimeout(() => controller.abort(), timeout);
```

Issue 3: Database Query Slow

Diagnosis:

```
-- Check for missing indexes
SELECT schemaname, tablename, indexname
FROM pg_indexes
WHERE tablename = 'ranking_candidates';
```

Fix:

```
-- Add index on foreign key
CREATE INDEX idx_ranking_candidates_ranking_id
ON ranking_candidates(ranking_id);
```

Issue 4: TanStack Query Not Refetching

Diagnosis:

```
// Check if query is enabled and stale
console.log({
  enabled: blocksQuery.enabled,
  isStale: blocksQuery.isStale,
  dataUpdatedAt: blocksQuery.dataUpdatedAt,
});
```

Fix:

```
// Invalidate queries after mutation
queryClient.invalidateQueries({
  queryKey: ['/api/blocks', snapshotId]
});

// Or refetch explicitly
blocksQuery.refetch();
```

🚀 Deployment & Infrastructure

Replit Autoscale Deployment

Configuration File: `.replit`

```
[deployment]
deploymentTarget = "autoscale"
run = ["npm", "start"]
build = ["npm", "run", "build"]

[[ports]]
localPort = 5000
externalPort = 80
```

Environment Setup: 1. **Production Environment Variables** - Copy all values from `.env` to Replit Secrets - Set `NODE_ENV=production` - Verify `PORT` is set to 5000 (Replit requirement)

2. **Build Command:**

```
npm run build
# Compiles TypeScript + Vite production build
# Output: client/dist/ (static files)
```

3. **Start Command:**

```
npm start
# Runs: NODE_ENV=production node gateway-server.js
```

Port Configuration: - **Gateway Server:** Port 5000 (public-facing, only non-firewalled port on Replit) - **SDK Server:** Port 3101 (internal, proxied through gateway) - **Vite Dev Server:** Port 3003 (development only, not in production)

Health Checks: - Endpoint: `GET /health` - Interval: 30 seconds - Timeout: 5 seconds - Healthy Threshold: 2 consecutive successes - Unhealthy Threshold: 3 consecutive failures

---

Database Configuration (Neon Serverless)

Connection String Format:

```
postgresql://[user]:[password]@[host].neon.tech:5432/[database]?sslmode=require
```

Connection Pooling:

```
// server/util/db.js
import { Pool } from 'pg';

export const pool = new Pool({
  connectionString: process.env.DATABASE_URL,
  max: 20,                    // Maximum connections
  idleTimeoutMillis: 30000,   // Close idle connections after 30s
  connectionTimeoutMillis: 5000,  // Wait 5s for connection
});
```

Migration Strategy:

```
# Never manually write SQL migrations
# Use Drizzle push to sync schema

npm run db:push          # Sync schema (checks for data loss)
npm run db:push --force  # Force sync (accepts data loss)
```

Backup Strategy: - **Neon Automatic Backups:** 7-day point-in-time recovery (Enterprise plan) - **Manual Exports:** Weekly full dumps via `pg_dump`

```
pg_dump $DATABASE_URL > backup-$(date +%Y%m%d).sql
```

Performance Optimization

Frontend (Vite Build)

```
// vite.config.ts
export default defineConfig({
  build: {
    rollupOptions: {
      output: {
        manualChunks: {
          'vendor': ['react', 'react-dom'],
          'ui': ['@radix-ui/react-dialog', '@radix-ui/react-select'],
          'query': ['@tanstack/react-query'],
        },
      },
    },
  },
});
```

Result: - Initial load: <100KB gzipped - Code splitting: Lazy load pages - Tree shaking: Remove unused Radix UI components

---

Backend (Express)

```
// gateway-server.js
import compression from 'compression';
import helmet from 'helmet';

app.use(compression());  // Gzip responses
app.use(helmet());       // Security headers
```

Monitoring & Logging

Application Logs

```
// server/util/logger.js
export function log(level, message, meta = {}) {
  const timestamp = new Date().toISOString();
  const logEntry = {
    timestamp,
    level,
    message,
    correlation_id: meta.correlation_id,
    ...meta,
  };

  console.log(JSON.stringify(logEntry));
}
```

Log Levels: - `DEBUG` - Detailed diagnostic info - `INFO` - General informational messages - `WARN` - Warning conditions (degraded but operational) - `ERROR` - Error conditions (failed requests, API errors)

Performance Metrics

```
// Track key metrics
const metrics = {
  snapshot_creation_ms: [],
  strategy_generation_ms: [],
  triad_total_ms: [],
  db_query_ms: [],
};

// Log P95 every hour
setInterval(() => {
  const p95 = calculatePercentile(metrics.triad_total_ms, 95);
  log('INFO', 'Performance P95', { triad_p95_ms: p95 });
}, 3600000);
```

Target Metrics: - Snapshot creation: P95 <3s - Strategy generation: P95 <12s - Triad pipeline: P95 <200s - Database queries: P95 <100ms

---

Security Checklist

- [ ] All API keys in Replit Secrets (not `.env` file)
- [ ] CORS restricted to production domain
- [ ] Rate limiting enabled (100 requests/15min)
- [ ] Helmet.js security headers configured
- [ ] Database connections use SSL (`sslmode=require`)
- [ ] No sensitive data in logs (API keys, user PII)
- [ ] Input validation with Zod on all endpoints
- [ ] SQL injection prevention (parameterized queries only)

Scaling Considerations

Current Limits (Autoscale): - **Concurrent Requests:** 100 (Replit Autoscale limit) - **Database Connections:** 20 (Neon free tier) - **AI Model Rate Limits:** See External API Constraints

When to Scale Up: 1. **Database:** Upgrade to Neon Pro when >100 concurrent connections 2. **AI Models:** Upgrade to Tier 2/3 when hitting rate limits 3. **Compute:** Move to dedicated VM when Autoscale insufficient

Cost Projection: - **0-1,000 users:** Replit Autoscale + Neon Free (~$50/month) - **1,000-10,000 users:** Replit VM + Neon Pro (~$200/month) - **10,000+ users:** Dedicated infrastructure (~$1,000/month)

□ Decision Log

Date	Decision	Rationale	Impact	Status
2025-10-03	Single-path triad (no fallbacks)	ML training integrity, consistent quality	No automatic failover	□ LOCKED
2025-10-06	Zero hardcoding policy	Global scaling, multi-market support	All data from DB/env	□ ENFORCED
2025-10-07	Per-route JSON parsing	Prevent abort errors, cleaner logs	No global body parser	□ IMPLEMENTED
2025-10-08	Model assertion in adapters	Catch silent model swaps	API response validation	□ IMPLEMENTED
2025-10-08	H3 geospatial indexing	Spatial queries for ML features	Added h3_r8 field	□ INDEXED
2025-10-09	Database-first venue resolution	Reduce API calls, faster response	Cache coords/hours separately	□ IMPLEMENTED
2025-10-13	Fix GPS state sync bug (Issue #21)	Frontend stuck on “GPS Required”	Create new object reference	□ FIXED
2025-10-13	Strategy background job	Decouple from blocks request	Async strategy generation	□ IMPLEMENTED
2025-10-13	Atomic ranking persistence	Prevent partial data corruption	Transaction with BEGIN/COMMIT	□ IMPLEMENTED
PENDING	Fix GPT-5 timeout (Issue #22)	Environment variable ignored	Use `process.env.PLANNER_DEADLINE_MS`	□ ACTIVE
PENDING	Auto-suppress closed venues (Issue #32)	Stop recommending closed places	`consecutive_closed_checks >= 3`	□ PLANNED

□ Lessons Learned for AI-Assisted App Development

LESSON #1: Documentation Is Code

This document (ARCHITECTUREV2.md) is as important as the codebase itself. It prevents: - □ Rework from misunderstanding design decisions - □ Introducing bugs by removing “unnecessary” code that has critical purpose - □ Breaking ML pipelines by adding fallbacks

LESSON #2: Trust But Verify AI Suggestions

When AI assistants suggest code changes: 1. □**Verify against this document** - Does it violate a guardrail? 2. □ **Check for hardcoding** - Does it introduce magic strings/numbers? 3. □ **Test the model ID** - Is it a real, available model (post-training-cutoff risk)? 4. □ **Review error handling** - Does it preserve fail-fast behavior?

LESSON #3: Model IDs Expire

Unlike traditional software dependencies, AI model IDs: - Have unpredictable deprecation timelines (6 months to 2 years) - May be renamed without backward compatibility (`gpt-4` → `gpt-4-turbo` → `gpt-5`) - Require monthly verification via research tools (Perplexity AI)

LESSON #4: Database Schema Is ML Contract

Every field in `snapshots` , `rankings` , `ranking_candidates` , `actions` is a contract with future ML models. Changing schema breaks training pipelines. **Rule:** Never drop fields without migration plan.

LESSON #5: Environment Variables Over Hardcoded Timeouts

Bad:

```
setTimeout(() => abort(), 300000); // 5 minutes - what if requirements change?
```

Good:

```
const timeout = parseInt(process.env.PLANNER_DEADLINE_MS || '120000', 10);
setTimeout(() => abort(), timeout);
```

Why: Requirements change, production vs staging may need different values, A/B testing requires dynamic configuration.

Future Enhancements (Roadmap)

Phase 1: Production Hardening (Q4 2025)

- [ ] Fix GPT-5 timeout environment variable (Issue #22)
- [ ] Add auto-suppression for closed venues (Issue #32)
- [ ] Implement unit tests for scoring engine (80% coverage target)
- [ ] Add Redis caching for Google Places API responses (reduce API costs)

Phase 2: ML Model Training (Q1 2026)

- [ ] Export training dataset (snapshots → rankings → actions)
- [ ] Train LightGBM ranking model (predict click probability)
- [ ] A/B test: Triad pipeline vs ML-reranked pipeline
- [ ] Implement model serving endpoint (`POST /api/ml/rerank`)

Phase 3: Multi-Market Expansion (Q2 2026)

- [ ] Seed venue catalogs for top 20 US metro areas
- [ ] Add international airport support (IATA code lookup)
- [ ] Localize currency (EUR, GBP support)
- [ ] Multi-language support (Spanish, French)

Phase 4: Mobile Apps (Q3 2026)

- [ ] React Native iOS app
- [ ] React Native Android app
- [ ] Push notifications for surge alerts
- [ ] Offline mode with cached venues

Related Documentation

- **MODEL.md** - AI model specifications, API details, verification tests
- **ISSUES.md** - Complete issue tracker with root cause analysis
- **README.md** - Quick start guide, API reference, development commands
- **ARCHITECTURE.md** - V1 architecture (kept for historical reference)

Contributing Guidelines

For Human Developers

1. Read this document BEFORE making changes
2. Check ISSUES.md for known problems
3. Verify changes don't violate guardrails
4. Test locally before committing
5. Update this document if architectural decisions change

For AI Assistants

1. **ALWAYS** reference this document when suggesting code changes
2. **NEVER** suggest removing error handling or adding fallbacks
3. **VERIFY** model IDs against MODEL.md (not training data)
4. **CHECK** for hardcoding violations before proposing edits
5. **ASK** user if uncertain about design decisions

End of Document
Maintained by: Development Team
Review Cadence: Monthly (after model verification runs)
Next Review: 2025-11-13