# Real Estate LLM System - Architecture & Data Ingestion Workflow

## Technical Proposal for Kelly Phillipps

---

## EXECUTIVE SUMMARY

**What you asked for:**

1. System architecture design
2. Data ingestion workflow
3. Recommendation: Simple database vs Google Docs approach

**My recommendation:**

- **Backend:** Python with Django + Django REST Framework
- **Data ingestion:** Simple database with Airtable interface (workers paste in forms, system handles everything else)
- **Timeline:** 10-12 weeks to working demo
- **Monthly cost:** $140-190 operational

**Why this approach:**

- Workers need zero technical skills (just copy/paste in Airtable)
- You get full control of data quality
- System scales from 1 resort to 100+ resorts
- All technology exists and is proven

---

## SYSTEM ARCHITECTURE

### Technology Stack (Python Django-Based)

**Backend Application:**

Component: Web Framework Technology: Django 4.2+ with Django REST Framework Purpose: Handle all API requests, user authentication, database operations Why: Battle-tested, excellent ORM, built-in admin panel, huge ecosystem

Component: Async Task Queue Technology: Celery with Redis Purpose: Handle embedding generation, long-running document processing Why: Django standard for background jobs, reliable

Component: RAG Orchestration Technology: LangChain Python library Purpose: Connect documents with LLMs, manage retrieval logic Why: Industry standard, works perfectly with Django

**Database Layer:**

Component: Primary Database Technology: PostgreSQL 15+ (via Supabase) Purpose: Store users, conversations, properties, all structured data Why: Robust, handles millions of records, managed service

Component: Vector Search Technology: pgvector extension (inside PostgreSQL) Purpose: Semantic search on document embeddings Why: No separate vector database needed, everything in one place

Component: Cache Technology: Redis Purpose: Cache frequent queries, reduce API costs 30-40% Why: Fast, simple, reduces costs significantly

**External APIs:**

Service: Primary LLM Provider: OpenAI GPT-4o-mini Cost: $0.15 per 1M input tokens Usage: 80% of queries (simple questions)

Service: Premium LLM Provider: Anthropic Claude 3.5 Sonnet Cost: $3.00 per 1M input tokens Usage: 20% of queries (complex legal, investment questions)

Service: Embeddings Provider: OpenAI text-embedding-3-small Cost: $0.02 per 1M tokens Purpose: Convert text to vectors for semantic search

**Hosting:**

Component: Backend API Platform: Railway.app or Render.com Cost: $20-25/month Why: Django-friendly, easy deployment, auto-scaling

Component: Database Platform: Supabase Pro Cost: $25/month Includes: PostgreSQL + pgvector + 25GB storage + backups

Component: Redis Platform: Upstash (serverless Redis) Cost: Free tier sufficient initially, $10/month at scale

Component: Frontend Platform: Vercel or Netlify Cost: Free tier works for MVP

## System Flow - How a Query Works

**Step 1: User asks question** User (Buyer role) on Resort Tamarindo website asks: "What's my ROI if I buy Villa Mar?"

**Step 2: Request reaches Django backend** Django receives request with: user authentication token, user role (buyer), tenant_id (tamarindo), query text

**Step 3: Django validates and processes** Check: Is user authenticated? Yes Check: Does user have access to this tenant? Yes Check: What's the user's role? Buyer Action: Log query for analytics

**Step 4: Generate query embedding** Django sends query to OpenAI Embeddings API Receives back: Vector representation of the question (1536 numbers)

**Step 5: Search documents (RAG retrieval)** Django queries PostgreSQL with pgvector:

- Filter by tenant_id = 'tamarindo'
- Filter by user_roles contains 'buyer'
- Find top 5 most similar documents using vector cosine similarity
- Also do keyword search (hybrid approach)

**Step 6: Retrieved documents** System finds:

- Villa Mar property listing (price, features, location)
- Tamarindo market data (average occupancy, ADR)
- ROI calculator methodology document
- Comparable sales in area
- Investment guide for foreign buyers

**Step 7: Assemble context** Django builds prompt:

- System instructions: "You are a real estate advisor for Costa Rica properties..."
- User role context: "User is a potential buyer/investor..."
- Retrieved documents: (the 5 documents found)
- Conversation history: (last 3-4 messages if continuing conversation)
- Current query: "What's my ROI if I buy Villa Mar?"

**Step 8: Route to appropriate LLM** Django analyzes query complexity:

- Contains keywords: "ROI", "investment", "buy"
- Complexity score: HIGH
- Decision: Route to Claude 3.5 Sonnet (premium model)

**Step 9: LLM generates response** Claude receives full context, generates response: "Based on current market data for Tamarindo, Villa Mar ($450,000) could generate approximately 45% occupancy at $320/night ADR, yielding gross annual revenue of $52,560. After expenses..."

**Step 10: Django processes response**

- Extracts citations (which documents were used)
- Counts tokens used (for billing)
- Stores conversation in database
- Returns to user

**Step 11: User sees response** Chat interface displays answer with sources: "According to Tamarindo market analysis (updated Dec 2024)..."

## Multi-Tenant Architecture

**What is multi-tenancy:** One system serves multiple resorts. Each resort (tenant) sees only their data. Like separate databases but more efficient.

**How it works in practice:**

Every table in database has tenant_id column:

Table: documents

- id: 12345
- tenant_id: 'tamarindo'
- content: "Villa Mar has 3 bedrooms..."
- user_roles: ['buyer', 'tourist']

Table: conversations

- id: 67890
- tenant_id: 'tamarindo'
- user_id: 'abc123'
- created_at: 2024-12-31

**Automatic filtering:**

When user from Tamarindo queries, Django automatically adds: WHERE tenant_id = 'tamarindo'

When user from Jacó queries, Django automatically adds: WHERE tenant_id = 'jaco'

**Security layers:**

Layer 1: PostgreSQL Row-Level Security Database itself enforces: "You can only SELECT rows where tenant_id matches your session tenant_id"

Layer 2: Django ORM filters Every query in Django code includes tenant filter automatically

Layer 3: Audit logging Every data access logged with: user_id, tenant_id, timestamp, action

**Benefits for you:**

One system to maintain instead of separate systems per resort Shared infrastructure = lower costs per resort Easy to add new resorts (just create new tenant_id) All resorts get updates simultaneously

## Role-Based Access Control

**Four main user roles:**

**Role: Buyer/Investor** Can see: Property prices, ROI calculators, market data, investment guides, legal info for foreigners Cannot see: Staff procedures, vendor contacts, other buyers' conversations LLM instructions: "Focus on investment metrics, ROI, comparables, legal requirements"

**Role: Tourist/Guest** Can see: Area guides, activities, restaurants, safety info, property amenities (not prices) Cannot see: Investment data, property prices, staff procedures LLM instructions: "Focus on experiences, recommendations, logistics, safety"

**Role: Vendor (Local service provider)** Can see: Demand patterns (what tourists ask for), pricing benchmarks, service opportunities Cannot see: Guest personal info, property financial data, other vendors' details LLM instructions: "Share market insights, protect guest privacy, encourage quality service"

**Role: Staff/Property Manager** Can see: SOPs, vendor contacts, procedures, maintenance schedules Cannot see: Owner financials (depending on settings), other properties' internal data LLM instructions: "Provide operational guidance, maintain professional boundaries"

**Implementation in code:**

Django model: User has field "role" Django model: Document has field "allowed_roles" (array)

When querying documents: Filter by: tenant_id = user.tenant_id AND user.role IN document.allowed_roles

**Dynamic prompts by role:**

```
SYSTEM_PROMPTS = {
    'buyer': "You are a real estate investment advisor for Costa Rica properties. Focus on ROI, market analysis, legal requirements for foreign buyers, and comparable properties. Always cite data sources with dates. If asked about non-investment topics, redirect to investment considerations.",
    'tourist': "You are a concierge for Costa Rica vacation properties. Focus on experiences, local
recommendations, activities, safety, and logistics. Never discuss property prices or investment
returns. For booking questions, direct to property managers.",

    'vendor': "You are a marketplace analyst helping local service providers understand tourist
demand. Share insights about what tourists want and pricing benchmarks. Never share guest
personal information. Encourage high-quality service aligned with tourist expectations.",

    'staff': "You are an operations assistant for property management. Provide SOPs, vendor
contacts, and procedural guidance. Do not share guest personal data or financial information
beyond your authorization level."
}
```

---

# DATA INGESTION WORKFLOW

## Recommended Approach: Airtable Interface + Automated Processing

**Why this approach:**

- Workers see familiar spreadsheet interface
- No coding required to add content
- Automatic processing in background
- Quality control built in
- Costs $30-40/month total

**The complete workflow:**

**Step 1: Worker opens Airtable** They see a form with simple fields to fill out

**Step 2: Worker fills form** Example for property listing:

- Property Name: "Villa Mar"
- Description: (paste full description)
- Price: 450000
- Bedrooms: 3
- Location: Tamarindo
- Target Audience: (checkboxes) Buyer, Tourist
- Category: Property Listing

**Step 3: Worker saves** Clicks "Submit" button in Airtable

**Step 4: Automation triggers (Make.com)** Detects new record in Airtable Starts workflow automatically

**Step 5: Data transformation** Make.com combines relevant fields into text for embedding: "Villa Mar - Tamarindo - 3 bedroom luxury villa for $450,000. Oceanview, private pool, walking distance to beach..."

**Step 6: Generate embedding** Make.com calls OpenAI API: Input: Combined text Output: Vector (1536 numbers representing meaning)

**Step 7: Store in database** Make.com calls Django API: Sends: Content, embedding, metadata (price, location, roles, category) Django stores in PostgreSQL

**Step 8: Document is live** Within 2-3 minutes of worker clicking "Submit", content is searchable in chatbot

## Airtable Structure

**Base: Real Estate Knowledge Base**

**Table 1: Property Listings**

| Field Name | Type | Purpose | Example |
|---|---|---|---|
| Property_Name | Single line text | Property identifier | "Villa Mar" |
| Description | Long text | Full property description | "Luxury 3-bedroom villa..." |
| Price_USD | Number | Property price | 450000 |
| Bedrooms | Number | Number of bedrooms | 3 |
| Bathrooms | Number | Number of bathrooms | 2.5 |
| Location | Single select | Area/region | Tamarindo |
| Amenities | Multiple select | Property features | Pool, Ocean View, WiFi |
| Photos | Attachments | Property images | (uploaded files) |
| Target_Roles | Multiple select | Who can see this | Buyer, Tourist |
| Status | Single select | Availability | Active, Pending, Sold |
| Last_Updated | Date | When info updated | 2024-12-31 |
| Tenant_ID | Single select | Which resort | tamarindo |

**Table 2: Area Guides**

| Field Name | Type | Purpose | Example |
|---|---|---|---|
| Region | Single select | Geographic area | Tamarindo |
| Category | Single select | Type of guide | Restaurants |
| Title | Single line text | Guide title | "Best Restaurants in Tamarindo" |
| Content | Long text | Full guide text | "1. Pangas Beach Club..." |
| Target_Roles | Multiple select | Who can see | Tourist, Buyer |
| Freshness_Date | Date | When to review | 2025-03-31 |
| Source_URL | URL | Original source if any | (url) |
| Tenant_ID | Single select | Which resort | tamarindo |

**Table 3: FAQs**

| Field Name | Type | Purpose | Example |
|---|---|---|---|
| Question | Long text | The question | "Is Uber safe in Costa Rica?" |
| Answer | Long text | The answer | "Yes, Uber is widely used..." |
| Category | Single select | Topic area | Transportation |
| Target_Roles | Multiple select | Who can see | Tourist, Buyer, |

Staff Language | Single select | EN or ES | EN Priority | Single select | How important | High
Tenant_ID | Single select | Which resort | tamarindo

**Table 4: Legal/Policy Documents**

Field Name | Type | Purpose | Example Document_Type | Single select | Category | Foreign Buyer Guide Title | Single line text | Document name | "Buying Property as Foreigner" Content | Long text | Full text | "Foreigners have equal rights..." Effective_Date | Date | When this became true | 2024-01-01 Review_Date | Date | When to review | 2025-01-01 Target_Roles | Multiple select | Who can see | Buyer Tenant_ID | Single select | Which resort | tamarindo

## Make.com Automation Setup

**Scenario 1: New Property Listing**

Trigger: New record in "Property Listings" table Frequency: Instant (webhook)

Module 1: Get Airtable Record Action: Fetch the new record with all fields

Module 2: Transform Data Action: JavaScript code module Code does:

- Combine Property_Name + Location + Description + Amenities into single text
- Build metadata JSON: price, bedrooms, location, etc
- Validate required fields exist

Module 3: Generate Embedding Action: HTTP Request to OpenAI Endpoint: https://api.openai.com/v1/embeddings Body:

```
{
  "model": "text-embedding-3-small",
  "input": "(combined text from Module 2)"
}
```

Response: Vector array (1536 numbers)

Module 4: Store in Database Action: HTTP Request to Django API Endpoint: https://your-backend.com/api/documents/ Method: POST Body:

```
{
  "tenant_id": "tamarindo",
  "content": "(combined text)",
  "embedding": "(vector from Module 3)",
  "metadata": {
    "property_id": "(Airtable record ID)",
    "price": 450000,
    "bedrooms": 3,
    "location": "Tamarindo",
    "content_type": "property_listing"
  },
  "user_roles": ["buyer", "tourist"],
  "source_url": "(Airtable record URL)"
}
```

Module 5: Update Airtable Status Action: Update record with "Processing Status" = "Indexed"

Module 6: Error Handling If any module fails:

- Log error to separate Airtable table
- Send Slack/email notification
- Mark record as "Failed - Needs Review"

**Cost:** Make.com Starter plan = $9/month (10,000 operations = enough for 5,000+ documents/month)

## Alternative: Direct Database Entry (Not Recommended Initially)

**If you wanted workers to use database directly:**

You'd build custom Django admin interface Workers log into Django admin Fill forms there instead of Airtable

**Why I don't recommend this for start:**

- Takes 2-3 weeks to build good UI
- Workers need to learn new interface
- More bugs to fix
- Airtable is proven, stable, familiar

**When to switch:** If you need features Airtable can't provide:

- Complex validation logic
- Multi-step workflows
- Custom calculations
- Integration with specific internal tools

Then build custom admin panel. But start with Airtable.

## Content Update Workflow

**Scenario: Price changes for Villa Mar**

Worker in Airtable:

1. Finds Villa Mar record
2. Changes Price field: 450000 → 425000
3. Clicks Save

Make.com detects update:

1. Fetches updated record
2. Regenerates embedding (content changed)
3. Calls Django API: Update document with ID X
4. Django updates database
5. Old price overwritten, new price live

Chatbot immediately uses new price in responses.

**Scenario: Document becomes outdated**

Worker marks "Freshness_Date" as past:

- Document still exists in database
- But flagged as "stale"
- Chatbot can mention: "Note: This information is from [date] and may need updating"
- Or exclude from search if too old

## Data Quality Control

**Validation rules in Airtable:**

Required fields marked with asterisk

- Property listings MUST have: Name, Description, Price, Location, Target_Roles
- FAQs MUST have: Question, Answer, Category, Target_Roles

Field types enforce correct data

- Price must be number (can't enter text)
- Date fields only accept dates
- Single select fields prevent typos (dropdown only)

**Validation in Make.com:**

Before generating embedding:

- Check text length (min 50 characters, max 10,000)
- Check required metadata exists
- Check price is reasonable (if property: between $50k - $5M)

- Reject if validation fails, log error

**Quality metrics tracking:**

Django tracks per document:

- Times retrieved (how often document used in responses)
- User feedback (thumbs up/down on answers citing this doc)
- Last updated date
- Usage count

Admin dashboard shows:

- Documents with low usage = maybe not useful, review/delete
- Documents with negative feedback = wrong information, review/fix
- Documents over 6 months old = flag for review

---

# DATABASE SCHEMA

## Django Models (Simplified)

**Model: Tenant** Purpose: Represents each resort/property group

Fields:

- id: UUID (unique identifier)
- name: CharField (resort name)
- domain: URLField (their website)
- settings: JSONField (custom configurations)
- created_at: DateTimeField
- is_active: BooleanField

**Model: Document** Purpose: Stores all knowledge base content

Fields:

- id: UUID
- tenant: ForeignKey to Tenant
- content: TextField (the actual text)
- embedding: Vector field (1536 dimensions for OpenAI)
- metadata: JSONField (flexible storage for price, location, etc)
- user_roles: ArrayField (which roles can see: buyer, tourist, vendor, staff)
- content_type: CharField (property_listing, area_guide, faq, legal, etc)

- freshness_date: DateField (when content should be reviewed)
- source_url: URLField (where this came from)
- is_active: BooleanField
- times_retrieved: IntegerField (usage tracking)
- created_at: DateTimeField
- updated_at: DateTimeField

**Model: Conversation** Purpose: Tracks chat sessions

Fields:

- id: UUID
- tenant: ForeignKey to Tenant
- user: ForeignKey to User
- user_role: CharField (buyer, tourist, vendor, staff)
- title: CharField (auto-generated from first message)
- created_at: DateTimeField
- updated_at: DateTimeField

**Model: Message** Purpose: Individual messages in conversations

Fields:

- id: UUID
- conversation: ForeignKey to Conversation
- role: CharField (user, assistant, system)
- content: TextField
- tokens_input: IntegerField
- tokens_output: IntegerField
- model_used: CharField (gpt-4o-mini, claude-3-5-sonnet, etc)
- retrieved_documents: JSONField (IDs of docs used to generate this response)
- latency_ms: IntegerField (how long to generate)
- created_at: DateTimeField

**Model: CustomUser (extends Django User)** Purpose: User accounts with roles

Fields:

- (inherits from Django User: email, password, etc)
- tenant: ForeignKey to Tenant
- role: CharField (buyer, tourist, vendor, staff, admin)
- preferences: JSONField (language, notification settings, etc)

## Database Indexes (Critical for Performance)

**Vector similarity index:**

```
CREATE INDEX ON documents USING hnsw (embedding vector_cosine_ops);
```

Purpose: Makes semantic search fast (milliseconds instead of seconds)

**Tenant filtering index:**

```
CREATE INDEX ON documents (tenant_id);
CREATE INDEX ON conversations (tenant_id);
CREATE INDEX ON messages (conversation_id);
```

Purpose: Fast filtering by resort

**Role filtering index:**

```
CREATE INDEX ON documents USING gin(user_roles);
```

Purpose: Fast filtering by who can see document

**Full-text search index:**

```
CREATE INDEX ON documents USING gin(to_tsvector('english', content));
```

Purpose: Keyword search (hybrid with vector search)

---

## SOURCES OF DATA - WHERE TO GET INFORMATION

### Property Listings Data

**Source 1: Encuentra24.com** What it is: Largest classifieds platform in Costa Rica Data available: Prices, photos, descriptions, agent contacts, location How to get it: Web scraping with Apify ($0.003 per listing) Legal: Public data, respect robots.txt, rate limit to 1 request/second Update frequency: Weekly for active listings Quality: Medium (some listings outdated, must verify)

**Source 2: RE.CR (Costa Rica MLS)** What it is: Independent MLS network, verified listings Data available: Similar to Encuentra24 but higher quality How to get it: Web scraping (more restrictive) Legal: Public data, be respectful Update frequency: Weekly Quality: High (verified against Registro Nacional)

**Source 3: Coldwell Banker Costa Rica** What it is: Major real estate agency, luxury focus Data available: Premium properties with detailed descriptions How to get it: Web scraping Update frequency: Monthly Quality: Very high (professional photography, accurate info)

**Source 4: Registro Nacional** What it is: Official government property records Data available: Title info, transaction history, liens How to get it: Manual lookup only (no bulk API) Legal: Official source, free Use case: Verify specific properties, not for mass data collection Quality: Authoritative

## Vacation Rental Analytics

**Source: AirDNA** Cost: $34/month (Research plan) Data provided:

- Occupancy rates by market (daily updates)
- Average Daily Rate (ADR)
- Revenue estimates
- Seasonality patterns
- Competitive analysis
- Market saturation metrics

Example current data (December 2024): Location: Tamarindo

- Occupancy: 49%
- ADR: $355/night
- Estimated monthly revenue: $35,379
- Peak season (Dec-Apr): 65% occupancy, $450 ADR
- Low season (Jun-Oct): 35% occupancy, $250 ADR

Location: Jacó

- Occupancy: 40%
- ADR: $318/night
- Peak season: 55% occupancy, $390 ADR

How to integrate:

- Download CSV exports monthly
- Upload to Airtable as "Market Data" table
- System uses for ROI calculations

**Alternative: Inside Airbnb** Cost: Free Data: Quarterly dumps of Airbnb listings (anonymized) Limitation: 1-3 month lag Use: Historical analysis, trends

## Tourism Information

**Source: ICT (Instituto Costarricense de Turismo)** Website: ict.go.cr/en/statistics.html Cost: Free Data available:

- Monthly tourist arrivals by country
- Average spending per tourist
- Average stay duration
- Most visited destinations
- Official high/shoulder/low season definitions

Current seasonal patterns (2024):

- High Season: December - April (60% of annual arrivals, premium pricing)
- Shoulder: May, November (moderate arrivals, 30-50% premium)
- Green Season: June - October (lowest arrivals, best deals)

How to use:

- Manual review monthly
- Create "Tourism Insights" documents in Airtable
- Use in market trend conversations

## Legal & Regulatory Information

**Key facts for foreign buyers:**

Property Rights:

- Foreigners have SAME rights as Costa Ricans
- No citizenship or residency required
- Can own 100% freehold (titled property)

Maritime Zone restrictions:

- 0-50 meters from high tide: Public domain, cannot own
- 50-200 meters: Concessions only, max 49% foreign ownership (need CR corporation with 51% local)
- 200+ meters: No restrictions

Transaction costs (3-5% total):

- Transfer tax: 1.5% of registered value
- Registry fees: 0.2-0.5%
- Notary fees: 1-1.5%
- Documentary stamps: ~0.25%
- Attorney fees: 1-1.5%

**Data sources:**

- Invest in Costa Rica (official government site)
- Costa Rica Legal Blog (updated frequently)
- Law firm white papers (Garnier RSVM, BLP, etc)
- Create once, updates as laws change

## Safety & Crime Statistics

**Current context (2024):**

National statistics:

- Homicide rate: 16.6 per 100,000 residents
- Trend: Increased since 2020 (COVID impact + drug trafficking)
- Comparison: Higher than Panama (9.2), lower than El Salvador

Geographic distribution:

- 47% of districts have under 50 incidents annually
- Concentrated in: San José urban center, port cities (Limón, Puntarenas)
- Tourist areas: Generally safer than urban centers

Tourist-specific:

- Violent crime against tourists: Rare
- Property crime (theft from cars, pickpockets): Moderate
- Scams: Common (taxi overcharging, timeshare pitches)

**Example real issue: Uber in Costa Rica**

- Status: Technically "illegal" but widely used
- Reality: 100,000+ Uber rides DAILY
- Incidents: Only 3 rock-throwing incidents in history
- Problem: Outdated online info scares tourists unnecessarily
- Your system's value: Provide CURRENT, accurate context

**Data sources:**

- OIJ (Organismo de Investigación Judicial) - official stats
- US State Department travel advisories
- Canada travel advisories
- Local news (La Nación, CRHoy)

**How to present:** Balance honesty with context. Frame: "Like any tourist destination, standard precautions apply. Here's what you need to know..." Then provide specific, actionable advice.

## COST BREAKDOWN

### Development Phase (One-Time)

Item: OpenAI API (testing/development) Cost: $50-100 Purpose: Testing embeddings, prompts during development

Item: Airtable Pro (3 months) Cost: $60 ($20/mo x 3) Purpose: Setting up data entry system

Item: Make.com (3 months) Cost: $27 ($9/mo x 3) Purpose: Building automations

Item: Hosting (Railway/Render, 3 months) Cost: $60-75 ($20-25/mo x 3) Purpose: Hosting Django backend during development

Item: Miscellaneous (domains, testing tools) Cost: $50

Total Development Cost: $250-350

### Monthly Operational Costs (After Launch)

**Infrastructure:**

Item: Backend Hosting (Railway.app or Render.com) Cost: $20-25 Includes: Python/Django hosting, auto-scaling, SSL

Item: Database (Supabase Pro) Cost: $25 Includes: PostgreSQL + pgvector, 25GB storage, daily backups

Item: Redis Cache (Upstash) Cost: $0-10 (free tier works initially) Purpose: Reduce API costs through caching

**Data Entry Tools:**

Item: Airtable Pro Cost: $20 Purpose: Data entry interface for workers

Item: Make.com Starter Cost: $9 Purpose: Automations (Airtable → Database)

**LLM APIs (estimated for 500 conversations/day, 3,500/week):**

Assumptions per conversation:

- Average 7 message exchanges
- 200 tokens per user message
- 400 tokens per assistant response

- 5 documents retrieved per query (3,000 tokens total context)

Monthly token usage:

- Input tokens: 500 conv/day × 7 exchanges × (200 + 3,000) = 11.2M tokens/month
- Output tokens: 500 conv/day × 7 exchanges × 400 = 4.2M tokens/month

With 80/20 split (cheap/premium models):

GPT-4o-mini (80% of traffic):

- Input: 9M tokens × $0.15/1M = $1.35
- Output: 3.4M tokens × $0.60/1M = $2.04
- Subtotal: $3.39

Claude 3.5 Sonnet (20% of traffic):

- Input: 2.2M tokens × $3.00/1M = $6.60
- Output: 840K tokens × $15.00/1M = $12.60
- Subtotal: $19.20

OpenAI Embeddings:

- New documents: ~100/month × 500 tokens = 50K tokens
- Cost: 50K × $0.02/1M = negligible
- Subtotal: $1

Total LLM costs: $24 (with caching saving ~30%)

**After semantic caching (30% reduction):** Actual LLM costs: ~$17-20/month

**Monitoring & Tools:**

Item: Sentry (error tracking) Cost: $0 (free tier sufficient)

Item: PostHog (analytics, optional) Cost: $0 (free tier sufficient initially)

**Total Monthly Operational: $111-129**

## Scaling Costs (At 2,000 conversations/day)

LLM APIs would increase to: $60-80/month Infrastructure might need upgrade: $40-50/month
Total at higher scale: $180-220/month

---

# TIMELINE - 12 WEEK PLAN

## Weeks 1-2: Foundation & Setup

**Django Project Setup:**

- Install Python 3.11, Django 4.2, Django REST Framework
- Create project structure
- Setup PostgreSQL database locally
- Create basic models (Tenant, User, Document, Conversation, Message)
- Setup Supabase account and production database
- Install pgvector extension
- Run migrations

**Authentication:**

- Implement JWT authentication with Django REST Framework
- User registration/login endpoints
- Role assignment system
- Basic permission checks

**LLM Integration:**

- OpenAI API setup (embeddings + GPT-4o-mini)
- Anthropic Claude API setup
- Test basic completions
- Implement streaming responses

**Deliverable:** Django API running locally, can authenticate users, can call OpenAI

## Weeks 3-4: RAG Implementation

**Document Ingestion:**

- LangChain setup with Django
- Document loader (accept text, PDF, DOCX)
- Chunking strategy implementation
- Embedding generation pipeline
- Store in PostgreSQL with pgvector

**Vector Search:**

- Similarity search implementation (cosine distance)
- Hybrid search (vector + keyword BM25)
- Metadata filtering

- Test retrieval accuracy with 30-50 sample documents

**API Endpoints:**

- POST /api/documents/ (create document)
- GET /api/documents/ (list documents)
- POST /api/documents/search/ (test retrieval)

**Deliverable:** Can upload documents, generate embeddings, retrieve relevant docs for query

## Weeks 5-6: Chat Implementation & RBAC

**Chat Endpoints:**

- POST /api/chat/ (send message, get response)
- GET /api/conversations/ (list user's conversations)
- GET /api/conversations/{id}/messages/ (get conversation history)

**RAG Pipeline:**

- Query → Embedding → Vector search → Context assembly → LLM → Response
- Implement conversation memory (last 5 messages)
- Citation tracking (which docs used)
- Token counting for cost tracking

**Role-Based Access:**

- Dynamic system prompts per role
- Document filtering by user_roles
- Test with different user types

**Model Routing:**

- Complexity detection (keywords: "invest", "legal", "analyze")
- Route simple → GPT-4o-mini, complex → Claude
- Fallback logic if API fails

**Deliverable:** Working chat that responds differently based on user role

## Weeks 7-8: Multi-Tenancy & Admin

**Multi-Tenant Implementation:**

- Automatic tenant_id filtering in all queries
- PostgreSQL Row-Level Security policies
- Test isolation (Tenant A cannot see Tenant B data)

- Admin can switch tenants (for management)

**Django Admin Panel:**

- Customize Django admin for Documents
- Add filters (tenant, content_type, user_roles)
- Bulk actions (mark as outdated, delete)
- Document preview
- Usage statistics per document

**Frontend (Basic):**

- Simple HTML/JavaScript chat interface
- Message display with citations
- Conversation history
- Can embed in iframe for resort websites

**Deliverable:** Multi-tenant system working, basic admin panel functional

## Weeks 9-10: Data Pipeline & Quality

**Airtable Setup:**

- Create base with tables (Property Listings, Area Guides, FAQs, Legal)
- Define all fields with validation
- Add sample records

**Make.com Automation:**

- Scenario 1: Property Listing → Embedding → Database
- Scenario 2: FAQ → Embedding → Database
- Error handling and notifications
- Test full pipeline end-to-end

**Quality Features:**

- Document usage tracking
- Freshness date checking
- Duplicate detection
- Admin dashboard showing:
  - Total documents per tenant
  - Most/least used documents
  - Documents needing review

**Deliverable:** Workers can add content via Airtable, appears in chatbot within minutes

### Weeks 11-12: Polish & Demo Prep

**Performance Optimization:**

- Implement Redis caching for frequent queries
- Database query optimization
- Response time profiling (target < 3 seconds)

**Production Deployment:**

- Deploy Django to Railway/Render
- Configure production database (Supabase)
- Setup environment variables
- SSL certificates
- Monitoring (Sentry)

**Content Loading:**

- 5-10 real property listings
- 2-3 area guides (different regions)
- 20-30 FAQs
- Legal guide for foreign buyers
- Safety information

**Demo Preparation:**

- Test accounts (buyer, tourist, vendor, staff)
- Sample conversations scripted
- Screenshots/videos
- Known limitations documented

**Documentation:**

- Architecture diagram
- API documentation (Django REST auto-generates)
- Admin user guide
- Troubleshooting guide

**Deliverable:** Production system live, with real content, demo-ready

---

## RECOMMENDATION: DATABASE VS GOOGLE DOCS

### Why I Recommend Database (with Airtable interface)

**Advantages:**

Control & Quality:

- You control data structure (enforced fields, validation)
- Can track who added what, when
- Can mark documents as outdated
- Can measure which content is actually useful

Flexibility:

- Can query data in multiple ways
- Can generate reports (most viewed properties, popular questions)
- Can do analytics (which topics get most questions)

Scalability:

- Works for 1 resort or 100 resorts
- No performance degradation as you grow
- Can handle millions of documents

Search Quality:

- Vector similarity works much better with structured data
- Can filter by multiple criteria (role + location + price range)
- Hybrid search (semantic + keyword) requires database

Multi-Tenancy:

- Each resort's data completely isolated
- Easy to manage permissions
- Can white-label per resort

**Why NOT Google Docs:**

No Structure:

- Workers could paste anything anywhere
- No validation (prices could be text, dates could be wrong)
- Hard to find duplicates

No Querying:

- Can't filter by role, location, price range
- Can't track usage
- Can't measure quality

Performance:

- Google Docs API has rate limits
- Slow to search through hundreds of documents
- Would need to download all docs, process locally

Maintenance Nightmare:

- How do you know if content is current?
- How do you find which doc to update when price changes?
- How do you prevent workers from accidentally deleting things?

Cost:

- Google Workspace per user adds up
- Need Google Drive API quota (costs money at scale)

**When Google Docs WOULD make sense:**

If you want absolute simplicity for 2-3 weeks prototyping If you have <20 documents total If you don't care about quality control If this is proof-of-concept only, not production

**My strong recommendation:** Use database with Airtable interface. The upfront setup (2-3 days) saves months of headaches later.

---

## NEXT STEPS

**This week (for you to decide):**

1. Review this proposal
2. Questions about anything unclear?
3. Approve technology choices (Django, Airtable, etc)
4. Confirm budget ($250-350 development + $110-130/month operational)
5. Confirm timeline (12 weeks acceptable?)

**Week 1 (if approved):**

Me:

- Setup development environment
- Create Django project
- Setup Supabase database
- Basic models and authentication working

You:

- Create OpenAI account, get API key ($50 credit to start)
- Create Anthropic account, get API key ($5 credit to start)
- Start gathering sample content (5-10 properties, FAQs, area info)

**Weekly check-ins:**

Format: 30-minute call or detailed written update Content:

- What I completed this week
- Any blockers or decisions needed from you
- What's planned for next week
- Budget status (API costs, any overages)

**Questions I need answered:**

1. Do you have access to 50-100 real property listings I can use for initial content?
2. Do you have existing FAQs or area guides, or should I help create them?
3. For the demo, which resort(s) should I prioritize (Tamarindo, Jacó, other)?
4. Do you need Spanish language support from day 1, or can that be V2?
5. Any specific legal/compliance concerns beyond Fair Housing Act?

This proposal gives you everything you asked for: architecture, workflow, and clear recommendation. The system is viable, costs are reasonable, and timeline is realistic for 30 hours/week.