Contents

Property Witch (AIPA) - Developer Manual

Version: 2.0
Last Updated: February 6, 2026
Platform: AI-Powered Real Estate Search Assistant for Portugal

---

Table of Contents

---

1. Executive Summary

1.1 What is Property Witch?

Property Witch (internally codenamed AIPA - AI Property Assistant) is an AI-powered real estate search assistant specifically designed for the Portuguese property market. It combines natural language processing, computer vision, and intelligent analysis to help users find their ideal properties.

1.2 Core Capabilities

Feature	Description
□ **Natural Language Search**	Users search in plain English/Portuguese: “Find a 2-bedroom apartment in Lisbon under €200k”
□ **AI Intent Detection**	Understands context, follow-ups, refinements, and conversation flow
□ **Smart Analysis**	Provides detailed reasoning for each listing’s relevance
□ **Vision AI**	Analyzes property photos to detect pools, sea views, gardens, etc.
□ **Knowledge Base**	Built-in Portuguese real estate knowledge (taxes, regions, visa programs)
□ **Auto-Indexing**	Scheduled indexing of 8,000+ listings every 4 hours
□ **Conversation Memory**	Persistent chat threads with context retention

1.3 Key Statistics

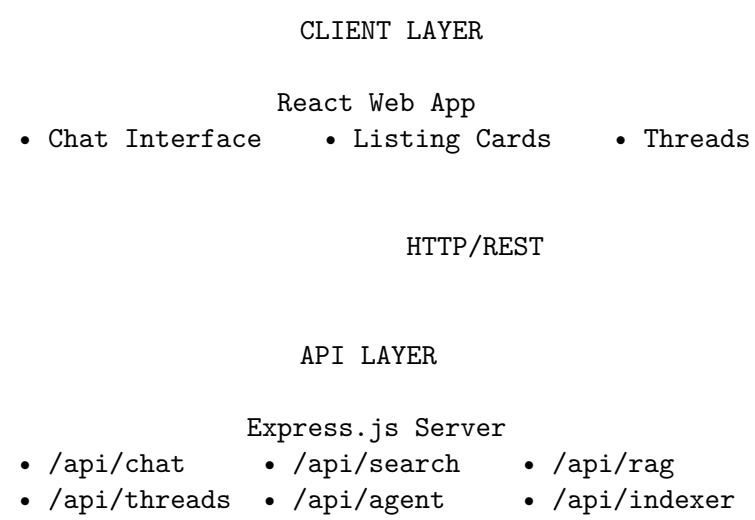
- **Codebase Size:** ~10,000 lines of TypeScript

- **AI Service:** 2,000+ lines (core intelligence)
- **Knowledge Entries:** 50+ Portuguese real estate topics
- **Supported Regions:** All Portuguese districts
- **Photo Analysis:** Up to 15 listings per search with visual features

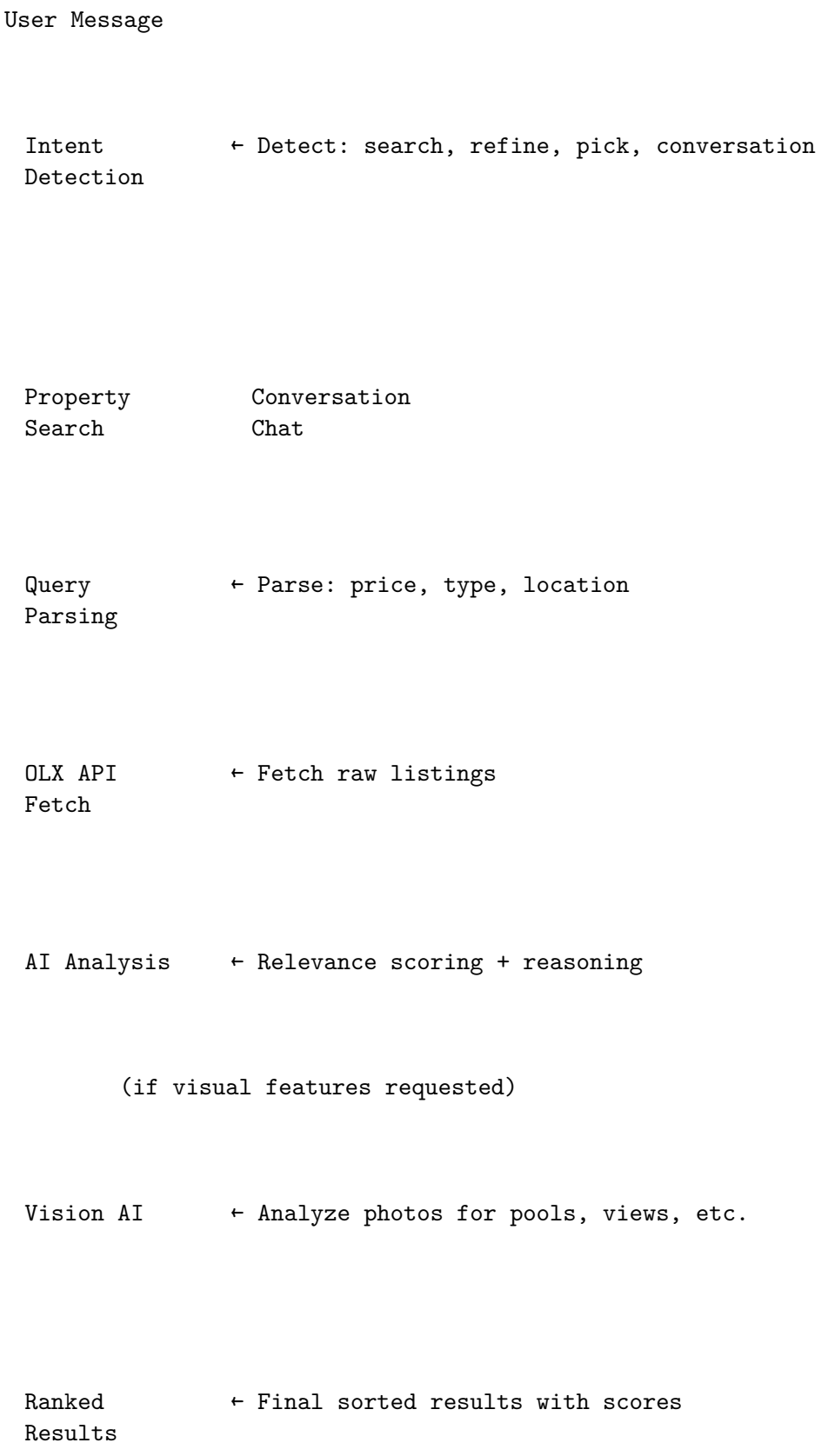
---

2. System Architecture

2.1 High-Level Architecture



2.2 Request Flow



3. Technology Stack

3.1 Backend

Technology	Version	Purpose
Node.js	18+	Runtime
TypeScript	5.x	Type safety
Express.js	4.x	HTTP server
Zod	3.x	Request validation
esbuild	-	Fast bundling
pdf-lib	-	PDF generation

3.2 Frontend

Technology	Version	Purpose
React	18.x	UI framework
TypeScript	5.x	Type safety
Vite	5.x	Build tool
CSS	-	Styling (no framework)

3.3 AI Services

Provider	Model	Purpose
Groq	`llama-3.3-70b-versatile`	Primary text AI
Groq	`meta-llama/llama-4-scout-17b-16e-instruct`	Vision AI
Ollama	`llama3.3-thinking-claude`	Local fallback
Anthropic	`claude-sonnet-4-20250514`	Final fallback

3.4 External Services

Service	Purpose
OLX Portugal	Property listings API
Render.com	Backend hosting
Hostinger	Frontend hosting

4. Directory Structure

```
aipa/
  server/                      # Backend application
    src/
      index.ts              # Express entry point
      config.ts             # App configuration

      adapters/             # Data source adapters
        base.ts            # Adapter interface
        olx.ts             # OLX Portugal adapter
        registry.ts        # Adapter registry
        mock.ts            # Mock data for testing
        idealista.ts       # (disabled)
        kyero.ts           # (disabled)

      config/
        sitePolicies.ts # Site-specific configs

      domain/
        listing.ts         # Listing types
        query.ts           # Query types

      routes/               # API endpoints
        chat.ts            # Main chat endpoint
        search.ts          # Direct search
        rag.ts             # RAG management
        agent.ts           # AI agent
        threads.ts         # Conversation threads
        indexer.ts         # Scheduler control
        report.ts          # PDF reports
        diagnostics.ts     # Health checks

      services/             # Business logic
        aiService.ts       #   Core AI (2000+ lines)
        searchService.ts # Search orchestration
        visionService.ts # Image analysis
        agentService.ts    # ReAct agent
        threadService.ts # Thread management
        scheduledIndexer.ts # Auto-indexing
        queryParser.ts     # Query parsing
        reportService.ts # PDF generation
        geoService.ts      # Distance calc
        currencyService.ts # Currency conversion
        rag/               # RAG subsystem
            index.ts
            ragService.ts
            vectorStore.ts
```

```
            embeddingService.ts
            knowledgeBase.ts

        storage/
            searchStore.ts

        types/
            api.ts

        utils/
            priceRange.ts
            slug.ts

    data/rag/               # Persisted RAG data
    browser-data/           # Browser cache
    package.json
    tsconfig.json

web/                        # Frontend application
    src/
        App.tsx            # Main component
        main.tsx           # Entry point
        types.ts           # Type definitions
        styles.css         # Styling
    index.html
    package.json
    tsconfig.json
    vite.config.ts

deploy/                     # Deployment configs
    DEPLOYMENT-GUIDE.md
    hostinger-env.txt
    nodejs-app/
    public_html/

scripts/                    # Dev scripts
    start-servers.sh
    stop.sh
    dev.sh
    restart.sh

docs/                       # Documentation
    DEVELOPER_MANUAL.md

render.yaml                 # Render.com config
README.md
```

5. Backend Services

5.1 AI Service (`server/src/services/aiService.ts`)

The AI Service is the brain of Property Witch, handling all AI-related operations.

5.1.1 Backend Management

```typescript
// Available backends with priority order
type AIBackend = "groq" | "ollama" | "claude" | "local";

// Detect best available backend
async function detectBackend(): Promise<AIBackend>

// Check if AI is available
async function checkAIHealth(): Promise<{
  available: boolean;
  backend: AIBackend;
}>

// Switch to a different backend
async function setActiveBackend(
  backend: AIBackend,
  model?: string
): Promise<{ success: boolean; message: string }>

// Get all available backends
async function getAvailableBackends(): Promise<AvailableBackend[]>
```

5.1.2 Intent Detection

```typescript
type IntentResult = {
  intent: "search" | "refine_search" | "pick_from_results" |
          "conversation" | "follow_up" | "show_listings";
  isPropertySearch: boolean;
  confidence: number;
  reason: string;
  extractedFilters?: {
    location?: string;
    propertyType?: string;
    priceMin?: number;
    priceMax?: number;
    areaMin?: number;
    areaMax?: number;
    bedrooms?: number;
    keywords?: string[];
  };
  selectionCriteria?: string;
};
```

```
async function detectIntent(
  message: string,
  conversationHistory: AIMessage[],
  hasRecentResults: boolean
): Promise<IntentResult>
```

5.1.3 Listing Analysis

```
type ListingRelevanceResult = {
  id: string;
  isRelevant: boolean;
  relevanceScore: number;  // 0-100
  reasoning: string;       // AI explanation
};

// Analyze listings for relevance
async function filterListingsByRelevance(
  userQuery: string,
  listings: ListingForAnalysis[],
  options?: {
    skipAI?: boolean;
    timeout?: number;
    forceDetailed?: boolean;
    hasVisualFeatures?: boolean;
  }
): Promise<ListingRelevanceResult[]>

// Get only relevant listings with analysis
async function getRelevantListings<T>(
  userQuery: string,
  listings: T[]
): Promise<{ listing: T; relevance: ListingRelevanceResult }[]>

// Pick best listings based on criteria
async function pickBestListings(
  userQuery: string,
  listings: any[],
  count?: number
): Promise<{
  selectedListings: any[];
  explanation: string;
}>
```

5.1.4 Configuration

```
const AI_ANALYSIS_CONFIG = {
  enableAIAnalysis: true,
```

```
  maxListingsForAI: 100,
  batchSize: 8,
  analysisTimeoutMs: 45000,
  detailedAnalysisThreshold: 20,
  forceAIForVisualFeatures: true,
};
```

5.2 Search Service (`server/src/services/searchService.ts`)

Orchestrates the search flow combining adapters, AI, and vision.

5.2.1 Main Search Function

```
type SearchRequest = {
  query: string;
  userLocation: UserLocation;
};

type SearchResponse = {
  searchId: string;
  matchType: "exact" | "near-miss";
  listings: ListingCard[];
  totalCount: number;
  appliedPriceRange: PriceRange;
  blockedSites: BlockedSite[];
};

async function runSearch(request: SearchRequest): Promise<SearchResponse>
```

5.2.2 Search Flow

1. **Parse Query** - Extract price range, property type, location
2. **Run Adapters** - Fetch from OLX API
3. **Filter Results** - Apply price and geo filters
4. **AI Analysis** - Score relevance and generate reasoning
5. **Vision Analysis** - Analyze photos if visual features requested
6. **Combine Scores** - Merge text and vision scores
7. **Return Ranked** - Sort by combined score

5.3 Vision Service (`server/src/services/visionService.ts`)

Analyzes property photos using AI vision.

5.3.1 Image Analysis

```
type ImageAnalysisResult = {
  features: ImageFeature[];
  confidence: Record<string, number>;
```

```
  description: string;
  propertyCondition?: "excellent" | "good" | "fair" | "needs_work" | "ruins";
  architecturalStyle?: string;
  surroundings?: string;
  rawAnalysis: string;
};

async function analyzeImage(imageUrl: string): Promise<ImageAnalysisResult | null>
```

5.3.2 Detectable Features

```
type ImageFeature =
  // Water features
  | "swimming_pool" | "sea_view" | "ocean_view" | "waterfront" | "river_view"
  // Natural surroundings
  | "forest" | "trees" | "garden" | "mountain_view" | "vineyard" | "olive_grove"
  // Terrain
  | "bare_land" | "flat_terrain" | "sloped_terrain" | "rocky_terrain"
  // Building condition
  | "ruins" | "old_building" | "needs_renovation" | "modern_architecture"
  | "traditional_architecture" | "rustic_style" | "luxury_finish"
  // Amenities
  | "parking" | "garage" | "terrace" | "balcony" | "rooftop" | "solar_panels"
  // Location type
  | "urban_area" | "suburban_area" | "rural_area" | "remote_location";
```

5.3.3 Feature Matching

```
// Extract visual features from user query
function extractImageFeatureQuery(userQuery: string): string[]
// "house with pool and sea view" → ["pool", "sea"]

// Check if listing features match requested
function matchesFeatureQuery(
  listingFeatures: ImageFeature[],
  requestedFeatures: string[]
): {
  matches: boolean;
  matchedFeatures: string[];
  score: number;
}
```

5.4 Thread Service (`server/src/services/threadService.ts`)

Manages persistent conversation threads.

```
type Thread = {
  id: string;
```

```
  title: string;
  createdAt: string;
  updatedAt: string;
  messages: ThreadMessage[];
  searchContext?: string;
  lastSearchResults?: ListingCard[];
};

// Thread operations
function createThread(title?: string): Thread
function getThread(threadId: string): Thread | undefined
function addMessage(threadId: string, role: string, content: string): void
function getConversationHistory(threadId: string, limit?: number): AIMessage[]
function storeSearchResults(threadId: string, listings: ListingCard[]): void
function getLastSearchResults(threadId: string): ListingCard[] | undefined
```

5.5 Scheduled Indexer (`server/src/services/scheduledIndexer.ts`)

Automatically indexes listings every 4 hours.

```
const CONFIG = {
  indexInterval: 4 * 60 * 60 * 1000,  // 4 hours
  searchDelay: 5000,                   // Between API calls
  maxListingsPerRun: 500,
  maxListingAgeDays: 90,
  enableVisionAnalysis: false,
  maxVisionAnalysisPerRun: 50,
};

// Locations indexed
const LOCATIONS = [
  "Lisbon", "Porto", "Algarve", "Cascais", "Sintra",
  "Braga", "Coimbra", "Évora", "Faro", "Vila Nova de Gaia"
];

// Query patterns
const QUERIES = [
  "apartments for sale",
  "houses for sale",
  "land for construction",
  "terreno urbano",
  "villa with pool",
  // ... more
];

function startScheduledIndexer(): void
function stopScheduledIndexer(): void
function runIndexerNow(): Promise<IndexerResult>
```

```
function getIndexerStatus(): IndexerStatus
```

---

6. AI Integration

6.1 Multi-Backend Architecture

Property Witch supports multiple AI backends with automatic fallback:

```
Primary: Groq Cloud (llama-3.3-70b-versatile)
    ↓ (if unavailable)
Fallback 1: Ollama Local (llama3.3-thinking-claude)
    ↓ (if unavailable)
Fallback 2: Claude API (claude-sonnet-4-20250514)
    ↓ (if unavailable)
Fallback 3: Local Analysis (no AI)
```

6.2 AI Call Flow

```typescript
async function callAIWithFallback(
  prompt: string,
  systemPrompt?: string
): Promise<string> {
  // 1. Try Groq
  if (groqAvailable) {
    const result = await callGroq(prompt, systemPrompt);
    if (result) return result;
  }

  // 2. Try Ollama
  if (ollamaAvailable) {
    const result = await callOllama(prompt, systemPrompt);
    if (result) return result;
  }

  // 3. Try Claude
  if (claudeAvailable) {
    const result = await callClaude(prompt, systemPrompt);
    if (result) return result;
  }

  throw new Error("No AI backend available");
}
```

6.3 Prompt Engineering

6.3.1 Intent Detection Prompt

```
You are analyzing a user message in a Portuguese real estate search app.
Determine the user's intent from these options:
- "search": Looking for properties (new search)
- "refine_search": Modifying current search
- "pick_from_results": Selecting from shown listings
- "conversation": General questions, not property search
- "follow_up": Questions about specific listings
- "show_listings": Wants to see results again

User message: "${message}"
Previous context: "${context}"

Respond with JSON only:
{
  "intent": "search|refine_search|pick_from_results|conversation|follow_up|show_listings",
  "isPropertySearch": true/false,
  "confidence": 0.0-1.0,
  "reason": "brief explanation"
}
```

6.3.2 Listing Analysis Prompt (Detailed Mode)

```
You are a Portuguese real estate expert. Analyze these listings for the query.

User Query: "${query}"

Listings:
${JSON.stringify(listings, null, 2)}

For EACH listing provide a DETAILED analysis (4-6 sentences):
- Explain why this listing matches or doesn't match the criteria
- Analyze the price (€/m² if applicable)
- Evaluate location benefits
- Note key features and potential concerns
- Give your recommendation

Respond with JSON array:
[{
  "id": "listing_id",
  "isRelevant": true/false,
  "relevanceScore": 0-100,
  "reasoning": "Your detailed 4-6 sentence analysis..."
}]
```

6.4 Token Configuration

```
// Groq API
{
```

```
  model: "llama-3.3-70b-versatile",
  max_tokens: 4096,
  temperature: 0.3,
}

// Claude API
{
  model: "claude-sonnet-4-20250514",
  max_tokens: 4096,
  temperature: 0.3,
}

// Ollama
{
  model: "llama3.3-thinking-claude",
  options: { num_predict: 4096, temperature: 0.3 }
}

// Vision (Groq)
{
  model: "meta-llama/llama-4-scout-17b-16e-instruct",
  max_tokens: 1000,
  temperature: 0.3,
}
```

---

7. Vision Service

7.1 Architecture

```
User Query: "house with pool near Lisbon"


  Extract Visual     → ["pool"]
  Features


  AI Text            → Score each listing
  Analysis              Note: "pool not confirmed in text"


  Filter for         → Listings needing photo check
```

```
  Vision Analysis       (score 25-85, has photos)


  Vision AI          → Analyze up to 15 photos
  (Groq Vision)         Detect: swimming_pool


  Combine Scores     → Text score + Vision bonus
                        [  Photo confirmed: pool]
```

7.2 When Vision Triggers

Vision analysis is triggered when:

1. User requests visual features (pool, sea view, garden, etc.)
2. AI text analysis says feature “not confirmed in text”
3. Listing has moderate relevance score (25-85)
4. Listing has photos available

7.3 Feature Synonym Mapping

```typescript
const featureSynonyms = {
  pool: ["swimming_pool"],
  sea: ["sea_view", "ocean_view", "waterfront"],
  ocean: ["ocean_view", "sea_view", "waterfront"],
  beach: ["sea_view", "ocean_view", "waterfront"],
  forest: ["forest", "trees"],
  garden: ["garden", "trees"],
  mountain: ["mountain_view"],
  ruins: ["ruins", "old_building", "needs_renovation"],
  modern: ["modern_architecture", "luxury_finish"],
  parking: ["parking", "garage"],
  terrace: ["terrace", "balcony", "rooftop"],
  rural: ["rural_area", "remote_location"],
};
```

7.4 Caching

Vision results are cached in memory to avoid redundant API calls:

```typescript
const analysisCache = new Map<string, ImageAnalysisResult>();
const CACHE_TTL = 24 * 60 * 60 * 1000; // 24 hours
```

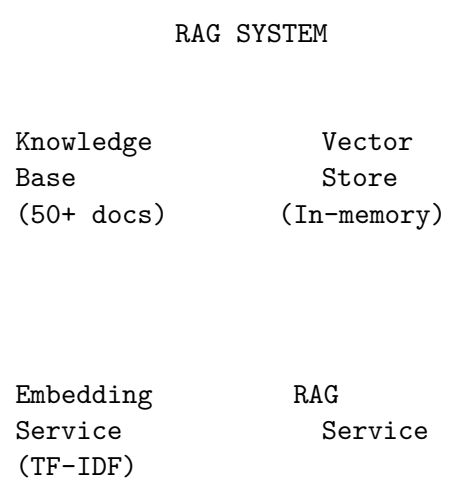
---

8. RAG System

8.1 Overview

The RAG (Retrieval-Augmented Generation) system provides Property Witch with domain knowledge about Portuguese real estate.

8.2 Components



8.3 Knowledge Base Categories

Category	Topics
Buying Process	NIF, lawyers, notaries, contracts, surveys
Taxes	IMT, Stamp Duty, IMI, capital gains, NHR
Regions	Algarve, Lisbon, Porto, Alentejo, Silver Coast, Madeira, Azores
Property Types	Urban land, rural land, ruins, quintas, apartments, villas
Visas	Golden Visa, D7 visa, NHR tax regime
Financing	Mortgages, bank requirements, costs
Legal	Property registration, habitação license, building permits

8.4 Vector Store

```typescript
// Collections
type Collection = "knowledge" | "listings" | "conversations";

// Document structure
```

```typescript
type Document = {
  id: string;
  content: string;
  embedding: number[];
  metadata: Record<string, any>;
};

// Operations
class VectorStore {
  addDocuments(collection: string, docs: Document[]): void
  search(collection: string, query: number[], topK: number): SearchResult[]
  searchByKeywords(collection: string, query: string, topK: number): SearchResult[]
  removeDocuments(collection: string, ids: string[]): void
  getStats(): Record<string, number>
  persist(): void  // Save to disk
  load(): void     // Load from disk
}
```

8.5 Embedding Service

Uses TF-IDF for fast local embeddings:

```typescript
function computeEmbedding(text: string): number[] {
  // 1. Tokenize and normalize
  const tokens = tokenize(text.toLowerCase());

  // 2. Compute TF (Term Frequency)
  const tf = computeTermFrequency(tokens);

  // 3. Compute IDF (Inverse Document Frequency)
  const idf = computeIDF(tokens, corpus);

  // 4. TF-IDF vector
  return computeTFIDF(tf, idf);
}
```

8.6 RAG Retrieval Flow

```typescript
async function buildRAGContext(
  query: string,
  options?: { maxDocs?: number; includeListings?: boolean }
): Promise<string> {
  // 1. Retrieve relevant knowledge
  const knowledge = await retrieveKnowledge(query, 5);

  // 2. Optionally retrieve similar listings
  let listings = [];
  if (options?.includeListings) {
```

```
    listings = await retrieveSimilarListings(query, 5);
  }

  // 3. Format context
  return formatContext(knowledge, listings);
}
```

---

9. Search Engine

9.1 Query Parsing

```
// Price intent patterns
"under 50k"           → { type: "under", max: 50000 }
"between 100k-200k"   → { type: "between", min: 100000, max: 200000 }
"around €75,000"      → { type: "around", target: 75000 }
"above 1 million"     → { type: "over", min: 1000000 }

// Currency detection
"$50k", "50k USD"     → USD
"€50k", "50k EUR"     → EUR
"£50k", "50k GBP"     → GBP

// Property type detection
"apartment", "flat"   → Apartment
"house", "villa"      → House
"land", "plot"        → Land
"room", "quarto"      → Room

// Location extraction
"in Lisbon"           → { city: "Lisbon" }
"near Porto"          → { city: "Porto", nearBy: true }
"Algarve area"        → { region: "Algarve" }
```

9.2 Listing Type Detection

```
function detectListingType(listing: Listing): "sale" | "rent" | undefined {
  const text = `${listing.title} ${listing.description}`.toLowerCase();

  // Rent indicators
  const rentKeywords = /arrendar|alugar|aluguer|rent|mês|month|mensal/;
  const lowPrice = listing.priceEur < 5000;

  // Sale indicators
  const saleKeywords = /venda|vender|à venda|for sale|compra/;
  const highPrice = listing.priceEur >= 30000;

  if (rentKeywords.test(text) || (lowPrice && !saleKeywords.test(text))) {
```

```
    return 'rent';
  }
  if (saleKeywords.test(text) || highPrice) {
    return 'sale';
  }
  return undefined;
}
```

9.3 Match Types

```
// Exact match: within ±2% of target price
// Near-miss: within ±10% of target price

const MATCH_RULES = {
  exactTolerancePercent: 0.02,    // 2%
  nearMissTolerancePercent: 0.1,  // 10%
  strictRadiusKm: 50,
  nearMissRadiusKm: 50,
};
```

9.4 Relevance Scoring

```
// Score components (0-100 total)
const scoring = {
  baseScore: 50,
  propertyTypeMatch: +35,        // Correct type
  propertyTypeMismatch: -20,     // Wrong type
  locationMatch: +15,            // In requested area
  priceInRange: +20,             // Within budget
  visualFeatureMatch: +25,       // Pool, sea view, etc.
  visualFeatureMissing: -10,     // Requested but not found
  isRoomNotApartment: -25,       // Room when apartment requested
  isCommercial: -30,             // Commercial property
};
```

---

10. Data Adapters

10.1 Adapter Interface

```
interface SiteAdapter {
  siteId: string;
  siteName: string;
  searchListings(context: SearchContext): Promise<Listing[]>;
}

interface SearchContext {
  query: string;
```

```
  priceRange: PriceRange;
  userLocation: UserLocation;
  propertyType?: string;
}

interface Listing {
  id: string;
  sourceSite: string;
  sourceUrl: string;
  title: string;
  priceEur: number;
  currency: string;
  beds?: number;
  baths?: number;
  areaSqm?: number;
  address?: string;
  city?: string;
  lat?: number;
  lng?: number;
  propertyType?: string;
  listingType?: 'sale' | 'rent';
  description?: string;
  photos: string[];
  lastSeenAt: string;
}
```

10.2 OLX Adapter

Uses OLX Portugal's public JSON API (not scraping).

```
// API endpoint
const OLX_API = "https://www.olx.pt/api/v1/offers";

// Categories
const OLX_CATEGORIES = {
  IMOVEIS: 16,           // All real estate
  VENDA: 3,              // Sale
  ARRENDAMENTO: 4,       // Rent
  TERRENOS_VENDA: 4795,  // Land
};

// Regions (district IDs)
const OLX_REGIONS = {
  lisboa: 11,
  porto: 13,
  faro: 8,
  setubal: 15,
  braga: 3,
```

```
  aveiro: 1,
  coimbra: 6,
  // ... all 18 Portuguese districts
};

// Request example
async function fetchListings(query: string, region: number): Promise<OLXResponse> {
  const url = `${OLX_API}?offset=0&limit=50&category_id=16&region_id=${region}&query=${query}`
  const response = await fetch(url, {
    headers: { "Accept": "application/json" }
  });
  return response.json();
}
```

10.3 Mock Adapter

For development and testing:

```
// Generates realistic mock listings
function generateMockListing(query: string): Listing {
  return {
    id: `mock-${uuid()}`,
    title: generateTitle(query),
    priceEur: randomPrice(50000, 500000),
    photos: ["/placeholder.jpg"],
    // ... other fields
  };
}
```

---

11. API Reference

11.1 Chat Endpoints

POST /api/chat Main chat endpoint for all user interactions.

Request:

```
{
  message: string;                    // User message
  threadId?: string;                  // Thread ID for persistence
  userLocation: {
    label: string;                    // "Lisbon, Portugal"
    lat: number;                      // 38.7223
    lng: number;                      // -9.1393
    currency: string;                 // "EUR"
  };
  mode: "search" | "chat" | "auto";   // Interaction mode
  conversationHistory?: AIMessage[];  // Previous messages
```

```
  lastSearchContext?: string;          // Search context
  conversationId?: string;             // Conversation ID
}
```

Response:

```
{
  type: "search" | "chat";
  intentDetected: string;
  message: string;                     // AI response
  parsedIntent?: ParsedSearchIntent;
  searchResult?: SearchResponse;
  suggestions?: string[];
  searchContext?: string;
  threadId?: string;
  aiAvailable: boolean;
  aiBackend: string;
  _timings?: Record<string, number>;
}
```

GET /api/chat/ai/health Check AI service availability.

Response:

```
{
  available: boolean;
  backend: "groq" | "ollama" | "claude" | "local";
  model: string;
}
```

GET /api/chat/ai/backends List all available AI backends.

Response:

```
{
  backends: [{
    id: string;
    name: string;
    available: boolean;
    models: string[];
    currentModel?: string;
  }];
  current: {
    backend: string;
    model: string;
  };
}
```

POST /api/ai/switch Switch to a different AI backend.

Request:

```
{
  backend: "groq" | "ollama" | "claude" | "local";
  model?: string;
}
```

11.2 Search Endpoints

POST /api/search Direct search without chat context.

Request:

```
{
  query: string;
  userLocation: UserLocation;
}
```

Response:

```
{
  searchId: string;
  matchType: "exact" | "near-miss";
  listings: ListingCard[];
  totalCount: number;
  appliedPriceRange: PriceRange;
  blockedSites: BlockedSite[];
}
```

11.3 RAG Endpoints

GET /api/rag/stats Get RAG system statistics.

Response:

```
{
  collections: {
    knowledge: number;
    listings: number;
    conversations: number;
  };
  totalDocuments: number;
}
```

POST /api/rag/initialize Reinitialize RAG system.

POST /api/rag/query Test RAG retrieval.

Request:

```
{
  query: string;
  topK?: number;
}
```

GET /api/rag/knowledge List all knowledge entries.

DELETE /api/rag/clear Clear all RAG data.

11.4 Thread Endpoints

GET /api/threads List all threads.

Response:

```
{
  threads: [{
    id: string;
    title: string;
    createdAt: string;
    updatedAt: string;
    messageCount: number;
  }];
}
```

POST /api/threads Create new thread.

Request:

```
{
  title?: string;
}
```

GET /api/threads/:id Get thread with history.

DELETE /api/threads/:id Delete thread.

PATCH /api/threads/:id Update thread title.

11.5 Indexer Endpoints

GET /api/indexer/status Get indexer status.

Response:

```
{
  isRunning: boolean;
  lastRunAt?: string;
  nextRunAt?: string;
```

```
  listingsIndexed: number;
  errors?: string[];
}
```

POST /api/indexer/start Start scheduled indexer.

POST /api/indexer/stop Stop scheduled indexer.

POST /api/indexer/run Trigger manual indexer run.

11.6 Agent Endpoints

POST /api/agent Run ReAct agent for complex queries.

Request:

```
{
  query: string;
  userLocation: UserLocation;
  maxSteps?: number;
}
```

POST /api/agent/check Check if query needs agent.

---

12. Frontend Architecture

12.1 Component Structure

```
// Main App Component (web/src/App.tsx)
function App() {
  // State
  const [messages, setMessages] = useState<ChatMessage[]>([]);
  const [searchResponse, setSearchResponse] = useState<SearchResponse | null>(null);
  const [currentThreadId, setCurrentThreadId] = useState<string | null>(null);
  const [isLoading, setIsLoading] = useState(false);
  const [aiBackend, setAiBackend] = useState<string | null>(null);
  const [approvedListings, setApprovedListings] = useState<ListingCard[]>([]);

  // ... 1200+ lines of component logic
}
```

12.2 Key Features

12.2.1 Retry with Backoff

```
async function fetchWithRetry(url: string, options: RequestInit, retries = 3) {
  for (let i = 0; i < retries; i++) {
    try {
      const response = await fetch(url, options);
      if (response.ok) return response;
      if (response.status >= 500) {
        // Server error - retry with exponential backoff
        await new Promise(r => setTimeout(r, 1000 * Math.pow(2, i)));
        continue;
      }
      return response; // Client error - don't retry
    } catch (error) {
      if (i === retries - 1) throw error;
      await new Promise(r => setTimeout(r, 1000 * Math.pow(2, i)));
    }
  }
}
```

12.2.2 Thread Management

```
// Create new thread
const createNewThread = async () => {
  const response = await fetch(`${API_BASE_URL}/api/threads`, {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({ title: 'New Chat' }),
  });
  const thread = await response.json();
  setCurrentThreadId(thread.id);
  setMessages([]);
};

// Load existing thread
const loadThread = async (threadId: string) => {
  const response = await fetch(`${API_BASE_URL}/api/threads/${threadId}`);
  const thread = await response.json();
  setCurrentThreadId(threadId);
  setMessages(thread.messages);
};
```

12.2.3 Listing Approval

```
const toggleApproval = (listing: ListingCard) => {
  setApprovedListings(prev => {
    const exists = prev.some(l => l.id === listing.id);
    if (exists) {
      return prev.filter(l => l.id !== listing.id);
    }
```

```
    return [...prev, listing];
  });
};
```

12.3 Type Definitions

```
// web/src/types.ts

interface ChatMessage {
  id: string;
  role: "user" | "assistant";
  content: string;
  timestamp: string;
  type?: "search" | "chat";
}

interface ListingCard {
  id: string;
  title: string;
  priceEur: number;
  displayPrice: string;
  locationLabel: string;
  beds?: number;
  baths?: number;
  areaSqm?: number;
  image?: string;
  sourceSite: string;
  sourceUrl: string;
  distanceKm?: number;
  matchScore?: number;
  aiReasoning?: string;
  listingType?: "sale" | "rent";
  propertyType?: string;
  visionAnalyzed?: boolean;
  visualFeatures?: string[];
}

interface SearchResponse {
  searchId: string;
  matchType: "exact" | "near-miss";
  listings: ListingCard[];
  totalCount: number;
  appliedPriceRange: PriceRange;
}
```

12.4 Environment Configuration

```
// vite.config.ts
export default defineConfig({
  server: {
    proxy: {
      '/api': {
        target: 'http://localhost:4000',
        changeOrigin: true,
      },
    },
  },
});

// In production, set VITE_API_URL
const API_BASE_URL = import.meta.env.VITE_API_URL || "";
```

---

13. Configuration

13.1 Environment Variables

Required

Variable	Description	Example
`GROQ_API_KEY`	Groq API key	`gsk_xxxxx...`

Optional - AI

Variable	Default	Description
`GROQ_MODEL`	`llama-3.3-70b-versatile`	Groq model
`ANTHROPIC_API_KEY`	-	Claude fallback
`CLAUDE_MODEL`	`claude-sonnet-4-20250514`	Claude model
`OLLAMA_URL`	`http://localhost:11434`	Local Ollama
`OLLAMA_MODEL`	`llama3.3-thinking-claude`	Ollama model
`OPENAI_API_KEY`	-	For embeddings

Optional - Server

Variable	Default	Description
`PORT`	`4000`	Server port
`MOCK_DATA`	`false`	Use mock data
`ENABLE_VISION_INDEXING`	`false`	Vision in indexer

Frontend

Variable	Description
`VITE_API_URL`	Backend URL (empty = same origin)

13.2 Application Config

```
// server/src/config.ts

export const APP_CONFIG = {
  port: Number(process.env.PORT ?? 4000),
  mockData: toBool(process.env.MOCK_DATA, false),
  reportsDir: path.resolve(projectRoot, "reports"),
};

export const MATCH_RULES = {
  exactTolerancePercent: 0.02,     // 2% for exact match
  nearMissTolerancePercent: 0.1,   // 10% for near-miss
  strictRadiusKm: 50,
  nearMissRadiusKm: 50,
};
```

13.3 AI Analysis Config

```
// server/src/services/aiService.ts

const AI_ANALYSIS_CONFIG = {
  enableAIAnalysis: true,
  maxListingsForAI: 100,           // Max listings to analyze
  batchSize: 8,                    // Listings per AI call
  analysisTimeoutMs: 45000,        // 45 second timeout
  detailedAnalysisThreshold: 20,   // Detailed if  20 results
  forceAIForVisualFeatures: true,  // Always use AI for visual queries
};
```

---

14. Deployment

14.1 Render.com (Backend)

render.yaml

```
services:
  - type: web
    name: propertywitch
    runtime: node
    region: frankfurt
```

```
    rootDir: server
    buildCommand: npm install && npm run build
    startCommand: node dist/server.js
    healthCheckPath: /health
    envVars:
      - key: GROQ_API_KEY
        sync: false
      - key: NODE_ENV
        value: production
```

Build Configuration

```
// server/package.json
{
  "scripts": {
    "build": "esbuild src/index.ts --bundle --platform=node --target=node18 --outfile=dist/serv
    "start": "node dist/server.js"
  }
}
```

14.2 Hostinger (Frontend)

1. Build frontend:

```
cd web
npm run build
```

2. Upload `dist/` contents to `public_html/`
3. Configure `.htaccess` for SPA routing:

```
RewriteEngine On
RewriteBase /
RewriteRule ^index\.html$ - [L]
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
RewriteRule . /index.html [L]
```

14.3 Local Development

```
# Terminal 1: Backend
cd server
npm install
npm run dev

# Terminal 2: Frontend
cd web
npm install
npm run dev
```

```
# Or use the script
./scripts/start-servers.sh
```

14.4 Production Environment

```
   Hostinger              Render.com
   (Frontend)           (Backend)

 propertywitch          propertywitch
 .com                   .onrender.com



                        Groq Cloud
                        (AI API)
```

---

15. Development Guide

15.1 Adding a New Data Adapter

1. Create adapter file:

```
// server/src/adapters/newsite.ts
import { SiteAdapter, Listing, SearchContext } from "./base";

export const newSiteAdapter: SiteAdapter = {
  siteId: "newsite",
  siteName: "New Site",

  async searchListings(context: SearchContext): Promise<Listing[]> {
    // Implementation
  }
};
```

2. Register in registry:

```
// server/src/adapters/registry.ts
import { newSiteAdapter } from "./newsite";

export const ADAPTERS: SiteAdapter[] = [
  olxAdapter,
  newSiteAdapter,  // Add here
];
```

15.2 Adding New Knowledge

```
// server/src/services/rag/knowledgeBase.ts

export const PORTUGAL_REAL_ESTATE_KNOWLEDGE: KnowledgeDocument[] = [
  // Add new entry
  {
    id: "new-topic",
    title: "New Topic Title",
    content: "Detailed content about the topic...",
    category: "category-name",
    tags: ["tag1", "tag2"],
    lastUpdated: "2026-02-06",
  },
  // ... existing entries
];
```

15.3 Adding New Visual Features

1. Update detection patterns:

```
// server/src/services/visionService.ts
const featureSynonyms = {
  newfeature: ["new_feature_1", "new_feature_2"],
  // ...
};
```

2. Update vision prompt to detect feature
3. Add to ImageFeature type

15.4 Testing

```
# Run with mock data
MOCK_DATA=true npm run dev

# Test specific endpoint
curl -X POST http://localhost:4000/api/chat \
  -H "Content-Type: application/json" \
  -d '{"message":"find apartments in lisbon","mode":"search","userLocation":{"label":"Lisbon","
```

---

16. Troubleshooting

16.1 Common Issues

AI Not Responding

```
Error: No AI backend available
```

Solutions: 1. Check `GROQ_API_KEY` is set 2. Verify Groq API status 3. Check network connectivity 4. Fallback to mock mode: `MOCK_DATA=true`

Search Returning No Results **Check:** 1. OLX API accessibility 2. Query parsing (check logs for parsed intent) 3. Price range (might be too restrictive) 4. Location matching

Vision Not Working **Check:** 1. Groq API key has vision model access 2. Photo URLs are accessible 3. Check vision service logs

High Latency **Causes:** - Cold start (Render free tier) - Large result sets - Multiple AI calls

Solutions: - Use `fetchWithRetry` on frontend - Enable batching for AI analysis - Limit results with `maxListingsForAI`

16.2 Debug Logging

```
// Enable verbose logging
console.log(`[Search] Query: ${query}`);
console.log(`[AI Analysis] Mode: ${isDetailed ? 'DETAILED' : 'BRIEF'}`);
console.log(`[Vision] Detected features: ${features.join(', ')}`);
console.log(`[Intent AI] Detected: ${intent} (${confidence})`);
```

16.3 Health Checks

```
# Backend health
curl http://localhost:4000/health

# AI health
curl http://localhost:4000/api/chat/ai/health

# RAG stats
curl http://localhost:4000/api/rag/stats

# Indexer status
curl http://localhost:4000/api/indexer/status
```

16.4 Error Codes

Code	Meaning	Solution
400	Validation error	Check request format
429	Rate limited	Wait and retry
500	Server error	Check logs
503	Service unavailable	AI backend down

Appendix A: Glossary

Term	Definition
AIPA	AI Property Assistant (internal codename)
IMT	Imposto Municipal sobre Transmissões (property transfer tax)
IMI	Imposto Municipal sobre Imóveis (annual property tax)
NHR	Non-Habitual Resident (tax regime)
NIF	Número de Identificação Fiscal (tax ID)
Quinta	Farm estate with house and land
Moradia	House/villa
Terreno	Land/plot
Urbano	Urban (buildable) land
Rústico	Rural (non-buildable) land
RAG	Retrieval-Augmented Generation
ReAct	Reasoning + Acting (AI agent pattern)

Appendix B: Quick Reference

API Endpoints

Method	Endpoint	Purpose
POST	`/api/chat`	Main chat
GET	`/api/chat/ai/health`	AI status
POST	`/api/search`	Direct search
GET	`/api/threads`	List threads
POST	`/api/threads`	Create thread
GET	`/api/rag/stats`	RAG stats
POST	`/api/agent`	Run agent
GET	`/api/indexer/status`	Indexer status

Environment Variables

```
# Required
GROQ_API_KEY=gsk_xxxxx

# Optional
PORT=4000
GROQ_MODEL=llama-3.3-70b-versatile
MOCK_DATA=false
```

Project Commands

```
# Development
npm run dev          # Start with hot reload
```

```
npm run build        # Build for production
npm start            # Run production build

# Scripts
./scripts/start-servers.sh  # Start all servers
./scripts/stop.sh           # Stop all servers
```

End of Developer Manual

Property Witch / AIPA - Version 2.0
Last Updated: February 6, 2026