Property Witch (AIPA) - Developer Manual

Version: 1.0
Last Updated: February 4, 2026
Project Type: Full-Stack Web Application

Table of Contents

---

1. Project Overview

Property Witch (internally called AIPA - AI Property Assistant) is an AI-powered real estate search application focused on the Portuguese property market. It allows users to search for properties using natural language queries and get AI-assisted responses.

Key Features

- Natural language property search
- Real-time property listings from OLX Portugal API
- AI-powered chat assistant (using Groq API with Llama 3.3)
- Intent detection (search vs. conversation)
- Property filtering by location, price, and type
- Responsive web interface

Live URLs

- **Frontend:** https://propertywitchtest.com (Hostinger)
- **Backend API:** https://propertywitch.onrender.com (Render.com)

---

2. Technology Stack

Backend

Technology	Purpose
Node.js 18+	Runtime environment
Express.js 4.19	Web framework

Groq API	AI/LLM provider (Llama 3.3 70B)
OLX Portugal API	Property listings source
CORS	Cross-origin resource sharing
dotenv	Environment variables

Frontend

Technology	Purpose
React 18	UI framework
TypeScript	Type safety
Vite 5	Build tool & dev server
CSS3	Styling

Deployment

Service	Purpose
Render.com	Backend hosting (free tier)
Hostinger	Frontend static hosting
GitHub	Source control

---

3. Project Structure

```
aipa/
├── server/                    # Backend application
│   ├── index.js              # Main server file (production)
│   ├── package.json          # Dependencies & scripts
│   ├── .env                  # Environment variables (local)
│   ├── .env.production       # Production env vars template
│   ├── data/
│   │   └── rag/
│   │       └── property-assistant.json  # RAG knowledge base
│   └── src/                  # TypeScript source (for development)
│       ├── index.ts          # Entry point
│       ├── config.ts         # Configuration
│       ├── adapters/         # Property source adapters
│       │   ├── base.ts       # Adapter interface
│       │   ├── olx.ts        # OLX Portugal adapter
│       │   ├── kyero.ts      # Kyero adapter (stub)
│       │   └── registry.ts   # Adapter registry
│       ├── routes/           # API route handlers
│       │   ├── chat.ts       # Chat endpoint
│       │   ├── search.ts     # Search endpoint
│       │   ├── agent.ts      # Agent endpoint
│       │   └── rag.ts        # RAG endpoints
│       ├── services/         # Business logic
│       │   ├── aiService.ts  # AI integration
│       │   ├── searchService.ts
│       │   ├── agentService.ts
│       │   └── rag/          # RAG system
│       ├── types/            # TypeScript types
│       └── utils/            # Utility functions
│
├── web/                      # Frontend application
│   ├── index.html           # HTML entry point
│   ├── package.json         # Dependencies & scripts
│   ├── vite.config.ts       # Vite configuration
│   ├── tsconfig.json        # TypeScript config
│   ├── .env.production      # Production API URL
│   └── src/
│       ├── main.tsx         # React entry point
│       ├── App.tsx          # Main application component
│       ├── styles.css       # Global styles
│       └── types.ts         # TypeScript types
│
├── scripts/                  # Utility scripts
│   ├── start.sh             # Start local servers
│   ├── stop.sh              # Stop servers
│   └── dev.sh               # Development mode
│
├── render.yaml              # Render.com deployment config
├── README.md                # Project readme
└── .gitignore               # Git ignore rules
```

---

4. Backend Architecture

4.1 Main Server (`server/index.js`)

The production server is a simple Express.js application that:

1. Serves API endpoints
2. Connects to OLX Portugal API for listings
3. Uses Groq API for AI responses

Key Components:

```
// Health endpoints
GET /               → Server status
GET /api/health     → Health check with Groq status
GET /api/ai/health → AI availability check

// Main chat endpoint
POST /api/chat      → Handles all user interactions
```

4.2 Chat Flow

```
User Message → Intent Detection → Search Intent?
                                      ↓ Yes
                                  OLX API Search → Filter Results → AI Summary →
Response
                                      ↓ No
                                  AI Conversation → Response
```

4.3 OLX Integration

The `searchOLX()` function:

1. Parses query for location (region IDs)
2. Extracts price constraints
3. Calls OLX Portugal public API
4. Filters results client-side (price, property type)
5. Returns formatted listings

OLX API Endpoint:

```
https://www.olx.pt/api/v1/offers/?
offset=0&limit=40&category_id=16&sort_by=created_at:desc
```

Region IDs:

```
{
  'lisboa': 11, 'lisbon': 11,
  'porto': 13,
  'faro': 8, 'algarve': 8,
  'braga': 3,
  'coimbra': 6,
  // ... more regions
}
```

4.4 AI Integration (Groq)

Uses Groq API with Llama 3.3 70B model:

- **API URL:** `https://api.groq.com/openai/v1/chat/completions`
- **Model:** `llama-3.3-70b-versatile`
- **Max Tokens:** 512 (summaries) / 1024 (chat)

---

5. Frontend Architecture

5.1 Main Component (`web/src/App.tsx`)

The single-page application with:

- Chat interface
- Property listings display
- Search results management

- Approved listings panel

5.2 State Management

Key state variables:

```
messages: ChatMessage[]        // Chat history
searchResponse: SearchResponse // Current search results
approvedListings: ListingCard[] // User-approved properties
isLoading: boolean             // Loading state
aiAvailable: boolean           // AI backend status
```

5.3 API Communication

```
const API_BASE_URL = import.meta.env.VITE_API_URL || "";

// Fetch with retry logic
const fetchWithRetry = async (url, options, maxRetries, delayMs) => {
  // Handles network errors during server cold starts
};
```

5.4 Types (`web/src/types.ts`)

```
type ListingCard = {
  id: string;
  title: string;
  priceEur: number;
  displayPrice: string;
  locationLabel: string;
  beds?: number;
  baths?: number;
  areaSqm?: number;
  image?: string;
  sourceSite: string;
  sourceUrl: string;
  matchScore?: number;
};

type SearchResponse = {
  searchId: string;
  matchType: "exact" | "near-miss";
  note: string;
  listings: ListingCard[];
  // ...
};
```

---

6. API Documentation

6.1 POST /api/chat

Main endpoint for all user interactions.

Request:

```json
{
  "message": "find apartments in lisbon under 200000",
  "userLocation": {
    "label": "Lisbon, Portugal",
    "lat": 38.7223,
    "lng": -9.1393,
    "currency": "EUR"
  }
}
```

Response (Search):

```json
{
  "type": "search",
  "intentDetected": "search",
  "message": "Found 15 properties in Lisbon...",
  "searchResult": {
    "searchId": "search-123456",
    "matchType": "exact",
    "note": "Found 15 properties",
    "listings": [...],
    "blockedSites": []
  },
  "aiAvailable": true,
  "aiBackend": "groq"
}
```

Response (Chat):

```json
{
  "type": "chat",
  "message": "The Portuguese property market...",
  "aiAvailable": true,
  "aiBackend": "groq"
}
```

6.2 GET /api/ai/health

Check AI backend availability.

Response:

```json
{
  "available": true,
  "backend": "groq",
  "model": "llama-3.3-70b-versatile"
}
```

---

7. Database & Storage

7.1 Current Implementation

The production server is **stateless** - no database required. All data comes from:

- OLX API (real-time listings)
- Groq API (AI responses)

7.2 RAG Knowledge Base (Development)

Located at `server/data/rag/property-assistant.json` :

- Contains Portuguese real estate knowledge
- Tax information, buying process, regions, etc.
- Pre-computed embeddings for similarity search

Structure:

```json
{
  "knowledge": [
    {
      "id": "buying-process-overview",
      "content": "The process of buying property in Portugal...",
      "metadata": {
        "title": "Buying Property in Portugal",
        "category": "buying-process",
        "tags": ["buying", "process", "steps"]
      },
      "embedding": [0.93, 0, 0, ...]
    }
  ]
}
```

---

8. AI Integration

8.1 Groq API

Configuration:

```
GROQ_API_KEY=<YOUR_GROQ_API_KEY>
```

Usage:

```javascript
const response = await fetch("https://api.groq.com/openai/v1/chat/completions", {
  method: "POST",
  headers: {
    "Authorization": `Bearer ${GROQ_API_KEY}`,
    "Content-Type": "application/json"
  },
  body: JSON.stringify({
    model: "llama-3.3-70b-versatile",
    messages: [
      { role: "system", content: systemPrompt },
      { role: "user", content: userMessage }
    ],
    temperature: 0.7,
    max_tokens: 512
  })
});
```

8.2 Intent Detection

Simple keyword-based detection:

```
const searchKeywords = [
  'find', 'search', 'show', 'looking for',
  'apartment', 'house', 'land', 'property',
  'buy', 'rent', 'under', 'below', 'near',
  'lisbon', 'porto', 'algarve', // locations
  'bedroom', 'bed', 'sqm', '€', 'euro'
];

// 2+ keyword matches = search intent
```

---

9. Deployment

9.1 Backend (Render.com)

Service Configuration:

- **Type:** Web Service
- **Runtime:** Node
- **Root Directory:** `server`
- **Build Command:** `npm install`
- **Start Command:** `node index.js`

Environment Variables:

```
GROQ_API_KEY=<YOUR_GROQ_API_KEY>...
NODE_VERSION=20
```

render.yaml:

```
services:
  - type: web
    name: propertywitch
    runtime: node
    rootDir: server
```

```yaml
    buildCommand: npm install
    startCommand: node index.js
    envVars:
      - key: GROQ_API_KEY
        sync: false
```

Note: Render free tier spins down after 15 min inactivity (50+ sec cold start).

9.2 Frontend (Hostinger)

Deployment Steps:

1. Build: `cd web && npm run build`
2. Upload `dist/` contents to Hostinger via File Manager or FTP
3. Ensure `.htaccess` handles SPA routing

Build Command:

```bash
npm run build
# Outputs to web/dist/
```

Environment:

```
# web/.env.production
VITE_API_URL=https://propertywitch.onrender.com
```

9.3 GitHub Repository

```
https://github.com/eduartgeorgia/propertywitch.git
```

Branches:

- `main` - Production branch (auto-deploys to Render)

---

10. Development Setup

10.1 Prerequisites

- Node.js 18+
- npm or yarn
- Git

10.2 Clone & Install

```bash
git clone https://github.com/eduartgeorgia/propertywitch.git
cd propertywitch

# Install backend dependencies
cd server
npm install
```

```
# Install frontend dependencies
cd ../web
npm install
```

10.3 Environment Setup

Backend (`server/.env`):

```
PORT=3000
GROQ_API_KEY=<YOUR_GROQ_API_KEY>
```

Frontend (`web/.env.development`):

```
VITE_API_URL=http://localhost:3000
```

10.4 Running Locally

Terminal 1 - Backend:

```
cd server
node index.js
# Server runs on http://localhost:3000
```

Terminal 2 - Frontend:

```
cd web
npm run dev
# Frontend runs on http://localhost:5173
```

10.5 Testing API

```
# Health check
curl http://localhost:3000/api/ai/health

# Search test
curl -X POST http://localhost:3000/api/chat \
  -H "Content-Type: application/json" \
  -d '{"message":"apartments in lisbon under 200000"}'
```

---

11. Configuration

11.1 Backend Config

Variable	Description	Default
`PORT`	Server port	3000
`GROQ_API_KEY`	Groq API key	Required

11.2 Frontend Config

Variable	Description	Default
`VITE_API_URL`	Backend API URL	"" (relative)

11.3 OLX API Categories

Category ID	Description
16	All Real Estate
1723	Apartments for Sale
1724	Houses for Sale
4795	Land for Sale

11.4 OLX Region IDs

Region	ID
Lisboa	11
Porto	13
Faro/Algarve	8
Braga	3
Coimbra	6
Setúbal	15
Aveiro	1
Leiria	10
Santarém	14

---

12. Troubleshooting

12.1 Common Issues

Issue: "Cannot GET /api/ai/health" on Render

- **Cause:** Old server version deployed
- **Fix:** Manually trigger deploy in Render dashboard

Issue: No listings returned

- **Cause:** OLX API may be rate limiting or region not found
- **Fix:** Check server logs, verify region ID exists

Issue: AI responses slow/timeout

- **Cause:** Groq rate limits or Render cold start
- **Fix:** Implement request retry logic, upgrade Render plan

Issue: Frontend can't connect to backend

- **Cause:** CORS or wrong API URL
- **Fix:** Check `VITE_API_URL` in `.env.production`

12.2 Debug Commands

```
# Check Render server status
curl https://propertywitch.onrender.com/

# Test OLX API directly
curl "https://www.olx.pt/api/v1/offers/?limit=5&category_id=16"

# View server logs (local)
node index.js 2>&1 | tee server.log
```

12.3 Render Dashboard

- URL: https://dashboard.render.com
- Service: propertywitch
- Logs: Available in "Logs" tab
- Deploy: Manual deploy from "Deploys" tab

Appendix A: Full TypeScript Server (Development)

The `server/src/` directory contains the full TypeScript implementation with:

- RAG (Retrieval-Augmented Generation) system
- Multi-step AI agents
- Multiple property source adapters
- Advanced intent detection

This is **disabled in production** due to:

- Complexity
- Cold start time
- Browser scraping dependencies (Puppeteer/Playwright)

To use the TypeScript version locally:

```
cd server
npm run dev  # Uses tsx to run TypeScript directly
```

Appendix B: Future Improvements

1. **Database Integration** - Store user preferences, saved searches
2. **More Property Sources** - Idealista, Kyero, Imovirtual
3. **Map Integration** - Show properties on map
4. **User Authentication** - Save favorites, compare properties

5. **Price Alerts** - Notify users of new matching listings
6. **Paid Render Plan** - Eliminate cold start delays

Document End

For questions or issues, contact the development team or check the GitHub repository.