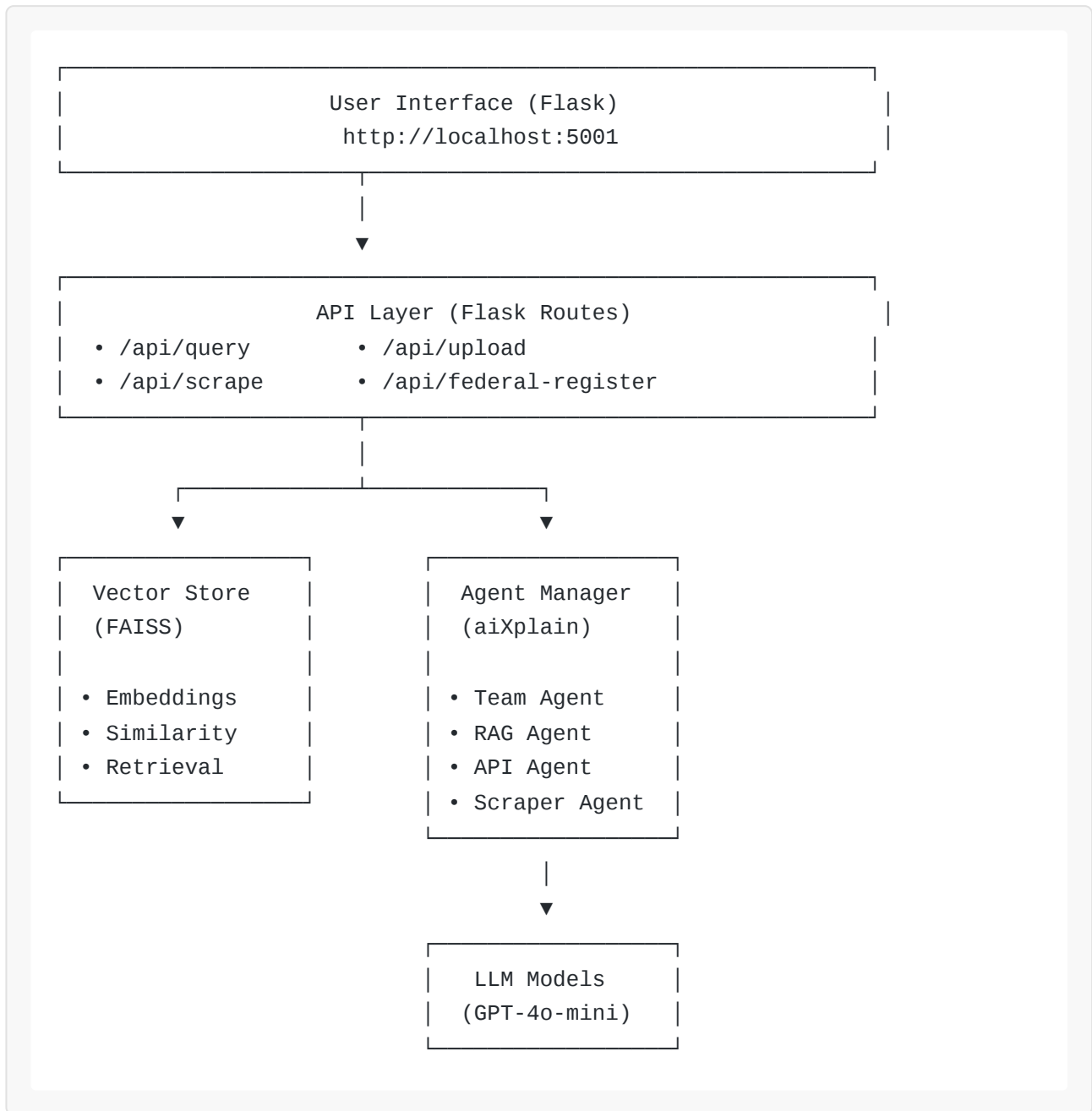


آلية عمل نظام Policy Navigator - دليل تقني شامل

نظرة عامة على النظام

Policy Navigator هو نظام RAG متعدد الوكلاء (Multi-Agent RAG System) يستخدم منصة aiXplain لمعالجة الاستفسارات حول السياسات والتنظيمات الحكومية الأمريكية.

1. معمارية النظام (System Architecture)



2. تدفق معالجة الاستفسار (Query Processing Flow)

المرحلة 1: استقبال الاستفسار

User **Input**



→ "What are EPA air quality regulations?"



Flask **Route**: /api/**query** (POST)

1. Validate **input**
2. Extract **query** text
3. Initialize processing



المرحلة 2: البحث في قاعدة البيانات الشعاعية

FAISS Vector Search

Input: User query text

Process:

1. Generate query embedding
 - ↳ SentenceTransformer
| (all-MiniLM-L6-v2)
↳ 384-dimensional vector
2. Search FAISS index
 - ↳ L2 distance calculation
 - ↳ Find top K similar documents
3. Retrieve documents
 - ↳ Document content
 - ↳ Metadata (title, source, type)
 - ↳ Similarity score

Output: Top 3 relevant documents



مثال على النتائج:

```
[
  {
    "content": "The EPA regulates air quality under Clean Air Act...",
    "metadata": {
      "title": "40 CFR § 50.4",
      "source": "CFR Title 40",
      "type": "regulation"
    },
    "score": 0.87
  },
  {
    "content": "National Ambient Air Quality Standards (NAAQS)...",
    "metadata": {
      "title": "40 CFR § 50.6",
      "source": "CFR Title 40",
      "type": "regulation"
    },
    "score": 0.82
  }
]
```

المرحلة 3: تحضير السياق للوكيل

Context Preparation

Input: Retrieved documents + User query

Process:

1. Format documents

- ↳ Extract top 800 chars per doc
- ↳ Add document metadata
- ↳ Number documents sequentially

2. Build context object

```
{  
  "documents": [...],  
  "query": "user query text"  
}
```

Output: Structured context for agent



مثال على السياق المُعد:

User Question: What are EPA air quality regulations?

Retrieved Policy Documents:

Document 1 (from 40 CFR § 50.4):

The EPA regulates air quality under the Clean Air Act. National Ambient Air Quality Standards (NAAQS) are established for six principal pollutants: carbon monoxide, lead, nitrogen dioxide, ozone, particulate matter, and sulfur dioxide...

Document 2 (from 40 CFR § 50.6):

The national primary and secondary ambient air quality standards for particulate matter are set forth in this section. The primary standards are designed to protect public health...

المرحلة 4: معالجة الوكيل (Agent Processing)

Agent Manager

Component: AgentManager (agent_manager.py)

Agent ID: 6905048fa1a609715ed913cc (Team Agent)

Step 1: Load Team Agent

- ↳ AgentFactory.get(Team_AGENT_ID)
- ↳ Verify agent availability

Step 2: Prepare enhanced query

- ↳ Combine user query + context
- ↳ Add instructions for agent
- ↳ Format as structured prompt

Step 3: Execute Team Agent

- ↳ team_agent.run(enhanced_query)

Team Agent Decision

Analyzes query and context
Decides which sub-agent to use:

- RAG Agent → Document Q&A
- API Agent → Federal Register
- Scraper Agent → Web content
- Direct LLM → General queries



Selected Sub-Agent Execution

For this query: RAG Agent

- Processes documents
- Extracts relevant info
- Generates answer



- ↳ LLM Processing (GPT-4o-mini)

- Reads context

- Understands query
- Synthesizes answer
- Formats response

Output: Agent response **object**



مثال على استجابة الوكيل:

```
{
  "success": true,
  "answer": "The EPA regulates air quality through the Clean Air Act, which
establishes National Ambient Air Quality Standards (NAAQS) for six
principal pollutants: carbon monoxide, lead, nitrogen dioxide, ozone,
particulate matter, and sulfur dioxide. These standards are designed to
protect public health and the environment. The regulations are codified in
40 CFR Part 50.",
  "agent": "Team Agent",
  "agent_id": "6905048fa1a609715ed913cc"
}
```


المرحلة 5: تنسيق الاستجابة النهائية

Response Formatting

Input: Agent response + metadata

Process:

1. Extract answer text
2. Add source information
3. Include confidence metrics
4. Add metadata

Output: JSON response



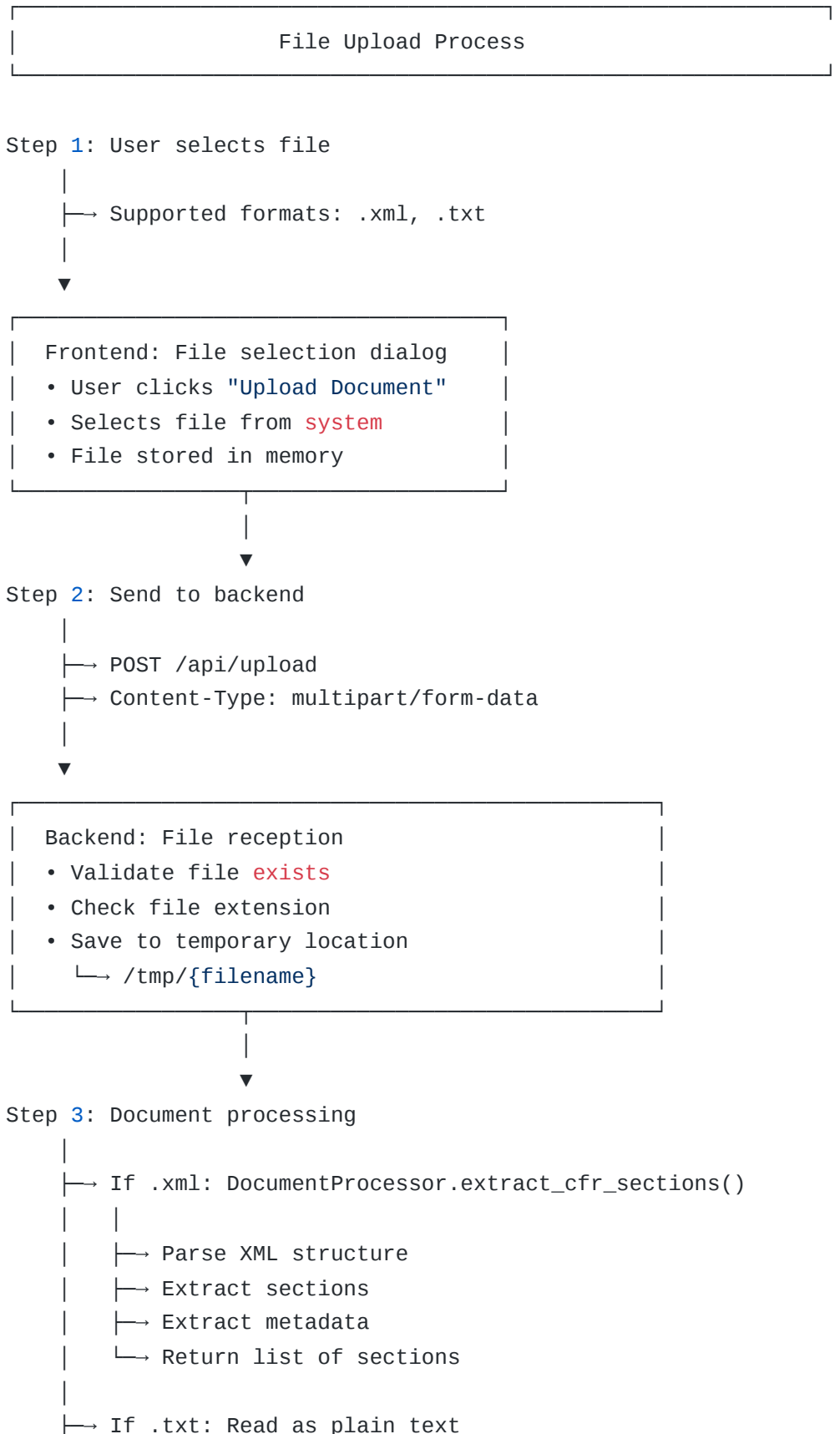
Final JSON Response

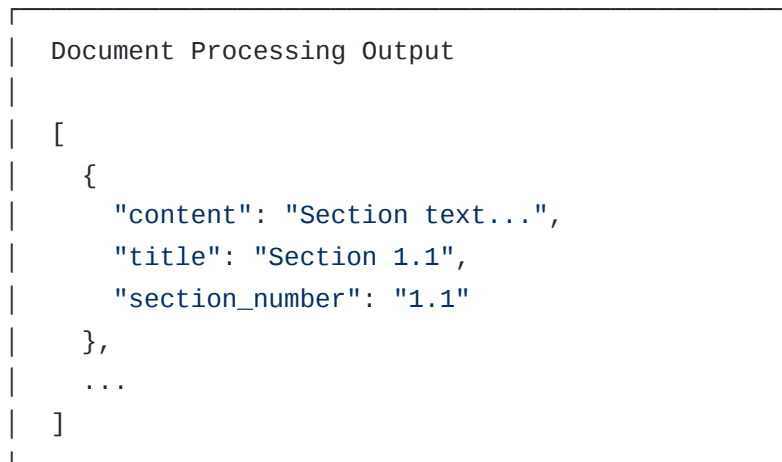
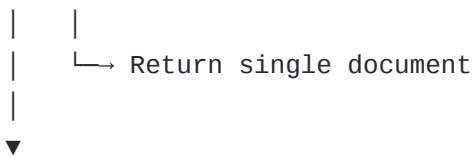
```
{  
  "answer": "The EPA regulates air quality...",  
  "source": "Multi-Agent RAG System (3 docs)",  
  "query": "What are EPA air quality...",  
  "num_results": 3,  
  "top_match": "40 CFR § 50.4",  
  "confidence": "0.87",  
  "mode": "multi_agent",  
  "agent": "Team Agent"  
}
```



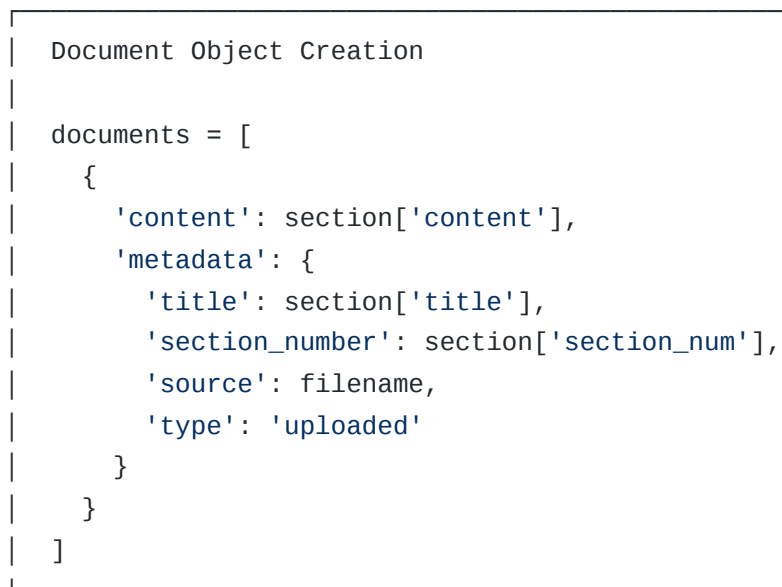
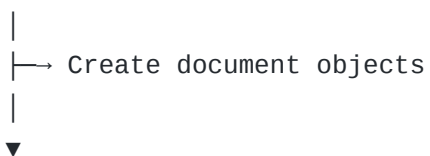
User sees formatted answer

3. تدفق رفع الملفات (File Upload Flow)

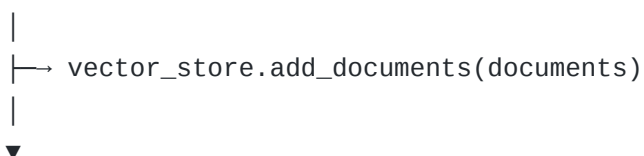




Step 4: Prepare for indexing



Step 5: Generate embeddings & index



FAISS Indexing Process

For **each** document:

1. Generate embedding

- ↳ `SentenceTransformer.encode(content)`
- ↳ 384-dim vector

2. Add to FAISS **index**

- ↳ `index.add(embedding)`
- ↳ Store metadata

3. Persist to disk

- ↳ `faiss.write_index()`
- ↳ `pickle.dump(metadata)`

Result: Documents now searchable



Step 6: Return success response



Response to User

```
{
  "message": "Successfully processed and
              indexed 42 sections from
              policy_doc.xml",
  "sections": 42
}
```

4. تدفق استخراج المحتوى من URL (URL Scraping Flow)

URL Scraping Process

Step 1: User inputs URL

↳ Example: `https://home.treasury.gov/`

Frontend: URL input

- User enters URL
- Clicks "Extract Content"

Step 2: Send to backend

↳ `POST /api/scrape`

↳ Body: `{"url": "https://..."}`

Backend: URL validation

- Check URL format
- Verify not `empty`

Step 3: Web scraping

↳ `url_scraper.scrape_url(url)`

URLScraperTool Process

1. HTTP Request
 - ↳ `requests.get(url, timeout=30)`
 - ↳ User-Agent: `PolicyNavigatorAgent/1.0`
 - ↳ Get HTML response

2. HTML Parsing

- ↳ `BeautifulSoup(html, 'html.parser')`
- ↳ Parse DOM structure

3. Content Extraction

- ↳ Extract title
 - ↳ Try `<title>` tag
 - ↳ Fallback to `<h1>`
- ↳ Extract main content
 - ↳ Find `<main>`, `<article>`, or `<div>`
 - ↳ Remove `<script>`, `<style>`, `<nav>`
 - ↳ Extract clean text
- ↳ Extract links
 - ↳ Find all `<a>` tags
 - ↳ Convert to absolute URLs

4. Metadata

- ↳ Check if government site (.gov domain)
- ↳ Count words
- ↳ Set status

Output: Scraped content object



Scraping Result

```
{
  "url": "https://home.treasury.gov/",
  "title": "U.S. Department of the Treasury",
  "content": "The Treasury Department is...",
  "links": [...],
  "is_government": true,
  "status": "success",
  "word_count": 1247
}
```



Step 4: Index scraped content

- ↳ Same as file upload indexing process



```
Add to Vector Store
• Generate embedding
• Add to FAISS index
• Persist to disk
```

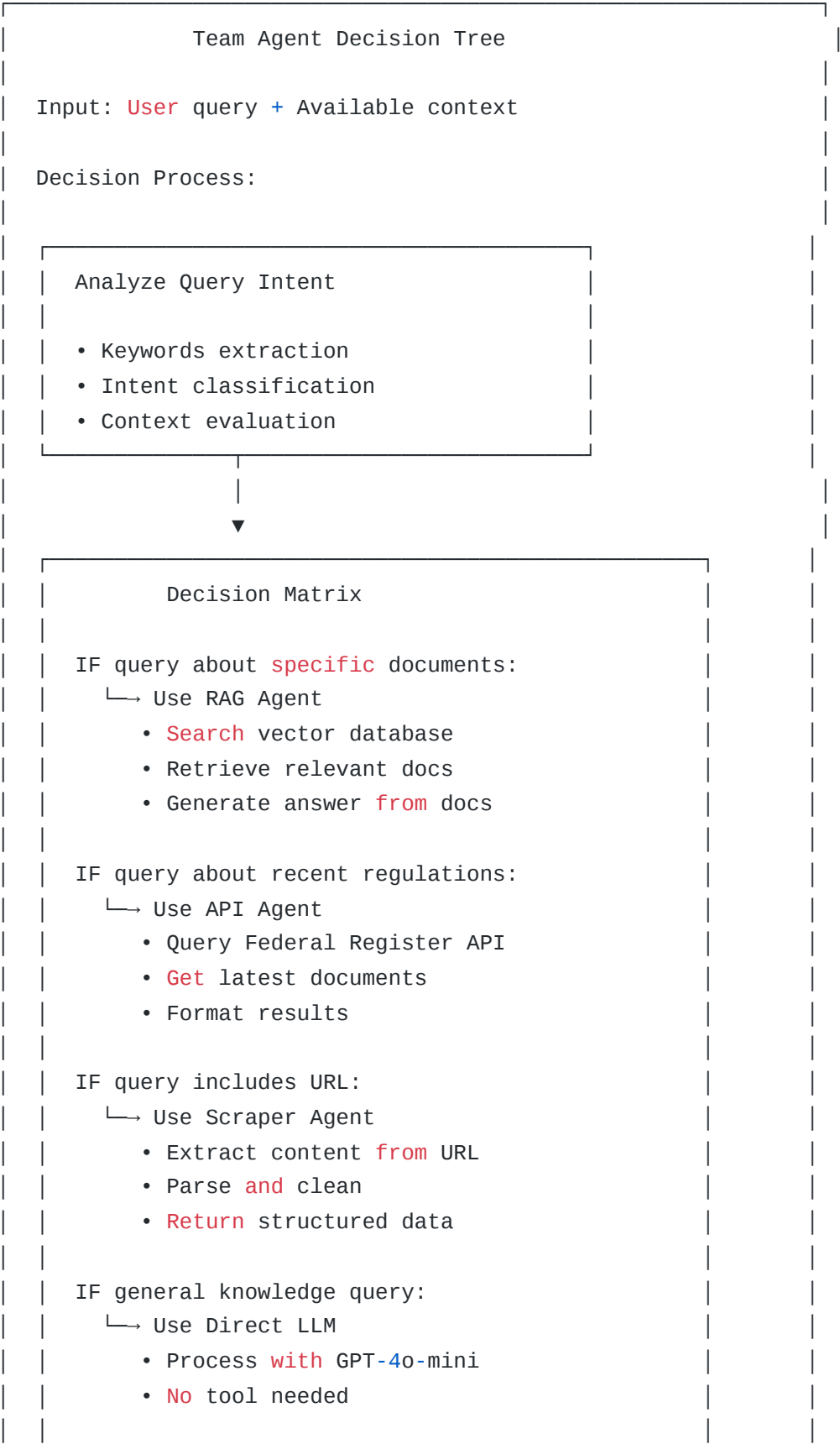


Step 5: Return success



```
Response to User
{
  "message": "Successfully scraped and indexed
              content from treasury.gov",
  "sections": 1,
  "title": "U.S. Department of the Treasury"
}
```

5. اختيار الأداة المناسبة (Tool Selection Logic)




```
IF complex multi-step query:
  ↳ Use Multiple Agents
    • Coordinate sub-agents
    • Combine results
    • Synthesize final answer
```

أمثلة على اختيار الأداة:

Reason	Selected Tool	Query Example
Document-based question	RAG Agent	"What are EPA air quality standards"
Recent/time-sensitive	API Agent	"Latest EPA rules published this week"
URL provided	Scraper Agent	"Extract content from epa.gov"
General knowledge	Direct LLM	"What is the Clean Air Act"
Multi-step task	Multiple Agents	"Find recent rules about air quality and summarize"

6. التقنيات المستخدمة (Technologies Used)

:Backend Stack

Python 3.9+

- └─ Flask 3.0.0 (Web framework)
- └─ Flask-CORS (Cross-origin)
- └─ python-dotenv (Environment vars)

Vector Database:

- └─ FAISS (Facebook AI Similarity)
- └─ sentence-transformers
 - └─ all-MiniLM-L6-v2 model

AI/ML:

- └─ aiXplain SDK 0.2.36
 - └─ AgentFactory
 - └─ ModelFactory
- └─ GPT-4o-mini (via aiXplain)

Web Scraping:

- └─ BeautifulSoup4
- └─ requests
- └─ lxml

Data Processing:

- └─ numpy
- └─ pickle

HTML5 + CSS3 + JavaScript

└─ Responsive design

└─ Dark/Light mode toggle

└─ Modern UI components

Features:

└─ Drag & drop file upload

└─ Real-time query processing

└─ Animated gradients

└─ Interactive elements

7. تدفق البيانات الكامل (Complete Data Flow)

Complete System Flow

1. User Interaction

|
|→ Query: "What are EPA regulations?"
|



2. Frontend Processing

|
|→ Validate input
|→ Show loading state
|→ Send AJAX request
|



3. Backend Reception

|
|→ Flask route: /api/query
|→ Extract query text
|



4. Vector Search

|
|→ Generate query embedding (384-dim)
|→ Search FAISS index (L2 distance)
|→ Retrieve top 3 documents
| ↳ [Doc1: score=0.87, Doc2: score=0.82, Doc3: score=0.78]
|



5. Context Preparation

|
|→ Format documents
|→ Build context object
|→ Prepare agent prompt
|



6. Agent Processing

|
|→ Load Team Agent (ID: 6905048fa1a609715ed913cc)
|→ Send query + context
|



7. Team Agent Decision



- └─> Analyze query intent
- └─> Evaluate available context
- └─> **Select** appropriate sub-agent
 - └─> Decision: Use RAG Agent



8. RAG Agent Execution



- └─> Read retrieved documents
- └─> Process **with** LLM (GPT-4o-mini)
 - └─> Understand query
 - └─> Extract relevant info
 - └─> Generate coherent answer



9. Response Synthesis



- └─> Format answer
- └─> **Add** metadata
- └─> Include source info



10. **Return to** Frontend



- └─> JSON response
- └─> Display formatted answer
- └─> **Show** source documents
- └─> Display confidence score

8. معالجة الأخطاء والاستراتيجيات الاحتياطية (Error Handling) (& Fallback)

```

Error Handling Strategy

Level 1: Agent Failure
├─ IF Team Agent fails:
│   └─ Try direct LLM call
│
Level 2: LLM Failure
├─ IF LLM fails:
│   └─ Return raw document excerpts
│
Level 3: Vector Search Failure
├─ IF FAISS fails:
│   └─ Return error message
│       "Please try again or upload documents"
│
Level 4: Complete System Failure
├─ IF all fails:
│   └─ Return user-friendly error
│       "System temporarily unavailable"
│
Logging:
├─ All errors logged to console
├─ Stack traces preserved
└─ User sees friendly message
```

9. الأداء والتحسينات (Performance & Optimization)

:Vector Search Performance

FAISS Index Statistics

- Index type: IndexFlatL2
- Embedding dimension: 384
- Total documents: 3,136
- Average search time: ~50ms
- Memory usage: ~5MB

Optimization:

- └─ Batch embedding generation
- └─ Index persistence to disk
- └─ Lazy loading of embeddings

:Agent Response Time

Response Time Breakdown

1. Vector search: ~50ms
2. Context preparation: ~10ms
3. Agent processing: ~2-5s
 - └─ Agent decision: ~500ms
 - └─ LLM generation: ~1.5-4.5s
4. Response formatting: ~5ms

Total: ~2-5 seconds

10. الأمان وأفضل الممارسات (Security & Best Practices)

Security Measures

API Key Management:

- ↳ Stored in .env file (not in code)
- ↳ Never committed to git
- ↳ Loaded via python-dotenv

CORS Protection:

- ↳ Flask-CORS configured
- ↳ Allows cross-origin requests

Input Validation:

- ↳ Query length limits
- ↳ File type restrictions (.xml, .txt)
- ↳ URL format validation
- ↳ Sanitization of user inputs

Rate Limiting:

- ↳ Timeout on HTTP requests (30s)
- ↳ Agent execution timeout

Data Privacy:

- ↳ Temporary files cleaned after processing
- ↳ No logging of sensitive data
- ↳ Vector embeddings are anonymized

الخلاصة (Summary)

نظام Policy Navigator هو نظام RAG متقدم يجمع بين:

1. البحث الشعاعي (FAISS) - للعثور على المستندات ذات الصلة
2. الوكلاء المتعددين (Multi-Agent) - لاتخاذ القرارات الذكية
3. نماذج اللغة الكبيرة (LLM) - لتوليد إجابات طبيعية
4. أدوات مخصصة (Custom Tools) - لمعالجة البيانات المتخصصة

التدفق الكامل:

User Query → Vector Search → Context Prep → Team Agent →
Sub-Agent Selection → LLM Processing → Response Generation →
User Interface

النظام يوفر:

- إجابات دقيقة مبنية على المستندات ✓
- اختيار ذكي للأدوات المناسبة ✓
- معالجة أخطاء قوية ✓
- أداء محسن ✓
- واجهة مستخدم حديثة ✓

تم إنشاء هذا المستند بواسطة: Policy Navigator Technical Team

التاريخ: November 2025

الإصدار: 2.0 (Multi-Agent System)