RAG Demo - Full-Stack AI-Powered Q&A System

A production-ready Retrieval-Augmented Generation (RAG) system with Angular frontend and .NET Core backend, featuring document upload, website scraping, and intelligent conversational AI.

🚀 Features Implemented

Backend Features (.NET 8.0)

✅ **Local Vector Database** - Qdrant for semantic search
✅ **Local Embeddings** - No API costs, runs offline
✅ **PDF Processing** - Extract and chunk documents with metadata
✅ **Website Scraping** - Ingest content from web pages with depth control
✅ **Semantic Search** - Find relevant context using vector similarity
✅ **GitHub Models Integration** - FREE AI-powered responses (GPT-4o-mini)
✅ **Smart Conversational Detection** - Filters casual greetings to avoid unnecessary searches
✅ **Formatted Prompts** - Structured AI responses with lists, headings, and code
✅ **Multi-page Crawling** - Optional link following (1-3 levels deep)
✅ **RESTful API** - Clean, documented endpoints with Swagger
✅ **Cost Effective** - 100% free for local deployment
✅ **Production Ready** - Scalable architecture with proper DI

Frontend Features (Angular 18)

✅ **Modern Chat Interface** - Real-time Q&A with loading states
✅ **Markdown Rendering** - Support for lists, headings, bold, code snippets
✅ **Admin Panel** - Separate interface for document management
✅ **Document Upload** - PDF file upload with progress indicators
✅ **Website Ingestion** - URL-based content scraping from admin panel
✅ **Message Formatting** - Beautiful list icons and structured responses
✅ **Conversational UX** - Smart handling of greetings and casual messages
✅ **Routing** - Separate routes for chat (/chat) and admin (/admin)
✅ **Error Handling** - Toast notifications for success/error states
✅ **Responsive Design** - Works on desktop, tablet, and mobile
✅ **PrimeNG UI** - Professional, modern UI components
✅ **Change Detection Optimization** - OnPush strategy for performance

---

📋 Prerequisites

Required

- .NET 8.0 SDK
- Docker Desktop (for Qdrant)
- Node.js 20+ and npm (for frontend)

Optional (for AI-powered responses)

- GitHub Models token (FREE) - Get from https://github.com/marketplace/models

---

🛠️ Quick Start

1. Start Qdrant Vector Database

```
# Pull and run Qdrant using Docker
docker run -d -p 6333:6333 -p 6334:6334 `
    -v qdrant_storage:/qdrant/storage `
    --name qdrant `
    qdrant/qdrant
```

Verify Qdrant is running:

```
curl http://localhost:6333/dashboard
```

2. Start Backend API

```
cd RAGDemoBackend
dotnet restore
dotnet run
```

The API will start at `https://localhost:5001` or `http://localhost:5000`

3. Start Frontend

```
cd RAGDemoFrontend
npm install
npm start
```

The app will open at `http://localhost:4200`

4. (Optional) Enable AI-Powered Responses

Get a FREE token from https://github.com/marketplace/models

```
# Set environment variable
$env:GH_TOKEN = "your-github-token"

# Or edit appsettings.json
# "GitHub": { "Token": "your-token" }
# "DemoSettings": { "UseGitHubModels": true }
```

5. Use the Application

1. Navigate to **Admin Panel** (`http://localhost:4200/admin`)
2. Upload a PDF document or ingest a website URL
3. Go to **Chat** (`http://localhost:4200/chat`)
4. Ask questions about your documents!

📊 Architecture

```
┌─────────────────────────────────────────┐
│      Angular 18 Frontend                │
│                                        │
│  ┌───────────────┐  ┌────────────────┐ │
│  │ Chat UI       │  │ Admin Panel    │ │
│  │ /chat         │  │ /admin         │ │
│  │               │  │                │ │
│  │- Q&A          │  │- Upload PDF    │ │
│  │- Markdown     │  │- Ingest URL    │ │
│  │- Formatted    │  │- Progress      │ │
│  └───────────────┘  └────────────────┘ │
└──────────────┬──────────────────────────┘
               │ HTTP/REST API
               ▼
┌─────────────────────────────────────────┐
│       .NET Core 8 API                   │
│                                        │
│  ┌─────────────────────────────────┐   │
│  │ Controllers                     │   │
│  │ - /api/chat/ask                 │   │
│  │ - /api/chat/upload              │   │
│  │ - /api/chat/ingest-url          │   │
│  │ - /api/chat/stats               │   │
│  └────────────┬────────────────────┘   │
│               │                         │
│  ┌────────────▼────────────────────┐   │
│  │ Services                        │   │
│  │ - Chat Service                  │   │
│  │ - Document Service              │   │
│  │ - Embedding Service             │   │
│  │ - Vector Store Service          │   │
│  └────┬────────────────┬───────────┘   │
│       │                │                │
│  ┌────▼──────────┐  ┌──▼──────────────┐ │
│  │ Local         │  │ Qdrant Vector   │ │
│  │ Embeddings    │  │ Store Service   │ │
│  └───────────────┘  └────────┬────────┘ │
└──────────────────────────────┬──────────┘
                               │
                     ┌─────────▼──────────┐  ┌───────────────────┐
```

```
                    │   Qdrant Database  │      │  GitHub Models   │
                    │   (Docker)         │      │  (GPT-4o-mini)   │
                    │   - Vector Storage │      │  - AI Responses  │
                    │   - Cosine Search  │      │  - Free Tier     │
                    └────────────────────┘      └──────────────────┘
```

🔧 Configuration

Edit `appsettings.json`:

```json
{
  "Qdrant": {
    "Host": "localhost",
    "Port": "6334",
    "UseHttps": false,
    "CollectionName": "documents"
  },
  "DemoSettings": {
    "ChunkSize": 500,
    "MaxSearchResults": 5
  }
}
```

📡 API Endpoints

Health Check

```
GET /api/chat/health
Response: {
  "status": "RAG Demo API is running",
  "timestamp": "2026-01-15T10:30:00Z"
}
```

Get Statistics

```
GET /api/chat/stats
Response: {
  "documentChunks": 42,
  "vectorStore": "Qdrant",
  "embeddingModel": "Local (all-MiniLM-L6-v2 compatible)",
  "timestamp": "2026-01-15T10:30:00Z"
}
```

Upload Document

```
POST /api/chat/upload
Content-Type: multipart/form-data

file: [PDF file]

Response: {
  "message": "Processed document.pdf"
}
```

Ingest Website Content

```
POST /api/chat/ingest-url
Content-Type: application/json

{
  "url": "https://example.com",
  "includeLinks": true,
  "maxDepth": 2
}

Response: {
  "url": "https://example.com",
  "chunksCreated": 25,
  "status": "Success",
  "processedUrls": ["https://example.com", "https://example.com/about"]
}
```

Ask Question

```
POST /api/chat/ask
Content-Type: application/json

{
  "question": "What is the product pricing?",
  "sessionId": "optional-session-id"
}

Response: {
  "answer": "Based on the documents, here are the key pricing details:\n\n1. Basic Plan - $29/month\n2. Pro Plan - $79/month\n3. Enterprise - Custom pricing",
  "sources": ["product-guide.pdf", "pricing.pdf"],
  "sessionId": "abc-123"
}
```

Delete Document

```
DELETE /api/chat/document/{documentName}

Response: {
  "message": "Deleted document.pdf"
}
```

Test with Swagger

Navigate to: `https://localhost:5001/swagger`

🎯 How It Works

1. Document Ingestion

PDF Upload:

- PDF uploaded via frontend admin panel
- Text extracted using iText7
- Text split into configurable chunks (default 500 chars)
- Each chunk generates a 768-dimensional embedding vector
- Chunks + embeddings + metadata stored in Qdrant

Website Scraping:

- URL provided via admin panel
- HTML content fetched and parsed
- Text extracted from main content areas
- Optional: Follow links up to 3 levels deep
- Each page chunked and embedded separately
- Source URL tracked in metadata

2. Semantic Search

- User asks a question in chat interface
- Question converted to embedding vector (same model)
- Qdrant performs cosine similarity search
- Top 5 most relevant chunks retrieved
- Source documents tracked for citation

3. Intelligent Response Generation

Conversational Detection:

- System filters casual messages ("thanks", "hi", "bye")
- Avoids unnecessary vector searches for greetings
- Provides friendly conversational responses

RAG Pipeline:

- Retrieved chunks provide context to AI
- GitHub Models (GPT-4o-mini) generates response
- Response formatted with markdown (lists, headings, code)
- Sources included for transparency
- Fallback to mock response if AI unavailable

4. Frontend Rendering

- Markdown automatically parsed and styled
- Lists display with icons (✓ for numbered, • for bullets)
- Headings properly sized and weighted
- Code snippets highlighted
- Real-time loading indicators
- Toast notifications for actions

💰 Cost Analysis

Component	Cost	Notes
Qdrant (Docker)	**$0**	Runs locally
Embeddings	**$0**	Local CPU inference
PDF Processing	**$0**	iText7 open source
.NET Hosting	**$0**	Local dev
Total	**$0/month**	🎉 Free!

Production Costs (Optional)

Upgrade	Monthly Cost	When Needed
OpenAI API (GPT-3.5)	~$5-20	Better answers
Azure App Service	~$55	Cloud hosting
Qdrant Cloud (1GB)	~$25	Managed vector DB

🚀 Key Features Breakdown

Chat Interface

- **Real-time messaging** with user/bot distinction
- **Markdown support** for formatted responses
- **List rendering** with visual icons (✓ •)
- **Code highlighting** for technical content
- **Loading states** with spinner animation
- **Error handling** with user-friendly messages

- **Keyboard shortcuts** (Enter to send, Shift+Enter for new line)
- **Auto-scroll** to latest message
- **Session persistence** across questions

Admin Panel

- **Dual upload methods**: PDF files or website URLs
- **Progress tracking** for uploads and ingestion
- **Depth control** for website crawling (1-3 levels)
- **Link following** option for comprehensive scraping
- **Success notifications** with chunk counts
- **Processed URL list** showing all scraped pages
- **Responsive design** for mobile admin access

Backend Intelligence

- **Smart routing** - Conversational vs. knowledge queries
- **Contextual prompts** - Structured AI instructions
- **Source tracking** - Maintains document provenance
- **Fallback handling** - Mock responses if AI unavailable
- **Chunk optimization** - Configurable size and overlap
- **Metadata enrichment** - URL, title, timestamp tracking

🚀 Next Steps & Roadmap

Current Capabilities

✅ Document Q&A with PDF uploads
✅ Website content ingestion
✅ AI-powered responses with formatting
✅ Separate admin and user interfaces
✅ Real-time chat experience
✅ Source attribution

Immediate Improvements

- [ ] Add user authentication (JWT)
- [ ] Implement chat history persistence
- [ ] Add document management dashboard
- [ ] Support more file formats (DOCX, TXT)
- [ ] Add batch upload capability
- [ ] Implement response streaming

Production Enhancements

- [ ] Database persistence (Entity Framework + PostgreSQL)
- [ ] Redis caching for frequent queries
- [ ] Rate limiting per user/IP

- [ ] Multi-tenant support
- [ ] Advanced analytics dashboard
- [ ] Multi-language support
- [ ] Mobile app (React Native)
- [ ] Document versioning
- [ ] Custom embedding models
- [ ] A/B testing for prompts

� Tips & Best Practices

For Best Results

1. **Chunk Size**: 500 chars works well for general content. Increase to 1000 for technical documentation.
2. **Search Results**: 5 chunks provide good context. Adjust in `appsettings.json` if needed.
3. **Question Format**: Be specific in questions for better retrieval accuracy.
4. **Document Organization**: Upload related documents together for coherent context.
5. **Website Scraping**: Start with `maxDepth: 1` to test before going deeper.
6. **Memory**: Qdrant caches chunks in memory for fast retrieval.

Performance Optimization

- Use `ChangeDetectionStrategy.OnPush` in Angular components
- Implement lazy loading for large document lists
- Cache frequent queries on backend
- Optimize chunk size based on content type
- Use indexes on Qdrant collections

Security Considerations

- Validate file uploads (size, type, content)
- Sanitize user inputs before querying
- Implement rate limiting on API endpoints
- Use HTTPS in production
- Secure API keys in environment variables
- Add authentication before production deployment

📖 Documentation

- **ARCHITECTURE.md** - Detailed system architecture and production deployment guide
- **API Documentation** - Available at `https://localhost:5001/swagger` when backend is running
- **Component Structure** - See `/src/app/components` for Angular component organization

---

🤝 Contributing

This is a demo/reference project showcasing RAG implementation. For production use:

- Add comprehensive unit and integration tests

- Implement proper error handling and logging
- Add monitoring and alerting
- Security hardening (authentication, authorization, input validation)
- Performance optimization and load testing
- Documentation for API consumers

📝 License

🎓 Learn More

- **RAG Concepts**: https://www.pinecone.io/learn/retrieval-augmented-generation/
- **Qdrant Docs**: https://qdrant.tech/documentation/
- **GitHub Models**: https://github.com/marketplace/models
- **Angular Best Practices**: https://angular.dev/best-practices
- **PrimeNG Components**: https://primeng.org/

Questions or Issues? Check the troubleshooting section or review the logs!

Production Ready? See ARCHITECTURE.md for deployment checklist.

Built with 💜 using .NET, Angular, and Qdrant