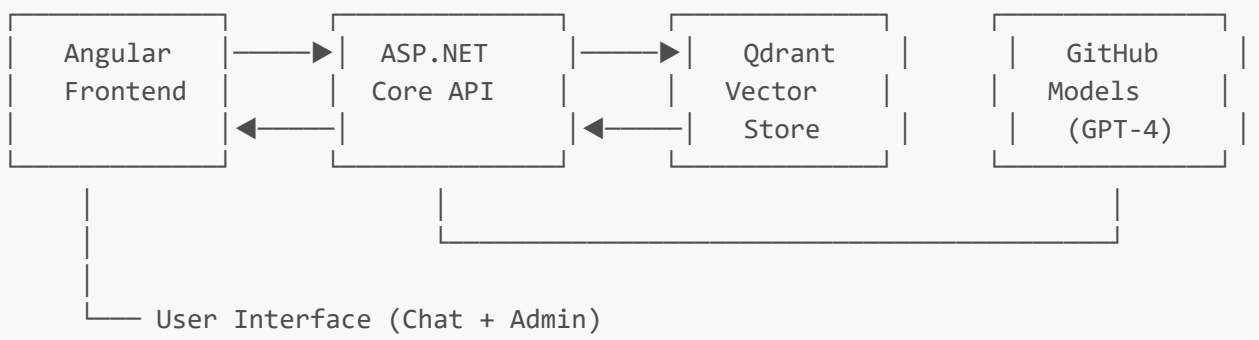
RAG Demo - Architecture & Implementation Guide

System Architecture

High-Level Overview



Components

1. Frontend (Angular 18)

- **Chat Interface**: User-facing Q&A interface
- **Admin Panel**: Document upload and website scraping
- **Services**: HTTP communication with backend API
- **Features**:
  - Real-time chat with loading states
  - Markdown formatting (lists, headings, bold, code)
  - Conversational message handling
  - Document upload (PDF)
  - Website content ingestion

2. Backend (ASP.NET Core 8)

- **Controllers**: REST API endpoints
- **Services**:
  - Chat Service: AI response generation
  - Document Service: PDF and web content processing
  - Embedding Service: Text vectorization
  - Vector Store Service: Qdrant integration
- **Features**:
  - RAG (Retrieval-Augmented Generation)
  - Vector similarity search
  - PDF text extraction
  - Website scraping with depth control
  - Conversational context detection

3. Vector Database (Qdrant)

- Stores document embeddings (768-dimensional vectors)
- Semantic similarity search
- Metadata storage for source tracking

4. AI Model (GitHub Models / Azure AI)

- Model: GPT-4o-mini
- Used for intelligent response generation
- Fallback to mock responses if unavailable

Implementation Steps

Phase 1: Core RAG Pipeline

1. **Document Ingestion**
    - PDF text extraction using iTextSharp
    - Text chunking (configurable size with overlap)
    - Generate embeddings using local model
2. **Vector Storage**
    - Store chunks in Qdrant with metadata
    - Index by document source and content
3. **Retrieval & Generation**
    - User asks a question
    - Generate query embedding
    - Search top-K similar chunks
    - Pass context to LLM
    - Generate answer with sources

Phase 2: Frontend Development

1. **Chat Interface**
    - Message display with user/bot distinction
    - Markdown rendering with icons
    - Loading states and error handling
    - Change detection optimization (OnPush strategy)
2. **Admin Panel**
    - PDF document upload
    - Website URL ingestion
    - Progress indicators
    - Success/error notifications

3. **Routing**
    - Separate routes for chat (/chat) and admin (/admin)
    - Navigation bar

Phase 3: Enhancements

1. **Smart Conversational Detection**
    - Filter casual greetings ("thanks", "hi", "bye")
    - Avoid unnecessary document searches
2. **Response Formatting**
    - Structured prompts for better AI output
    - Headings, lists, bold text support
    - Code snippet formatting
3. **Website Scraping**
    - Extract text from web pages
    - Optional link crawling (1-3 levels)
    - Multiple page processing

Key Technologies

Component	Technology	Purpose
Frontend	Angular 18 (Standalone)	Modern SPA framework
UI Library	PrimeNG	Professional UI components
Backend	ASP.NET Core 8	RESTful API
Vector DB	Qdrant	Semantic search
Embeddings	all-MiniLM-L6-v2	Local embedding model
AI Model	GPT-4o-mini (GitHub)	Response generation
PDF Parser	iTextSharp	Text extraction
Web Scraper	HtmlAgilityPack	HTML parsing

Production Readiness Checklist

1. Security

- [ ] Add authentication/authorization (JWT tokens)
- [ ] Implement rate limiting on API endpoints
- [ ] Add CORS configuration for specific domains
- [ ] Secure API keys in Azure Key Vault or similar

- [ ] Input validation and sanitization
- [ ] HTTPS enforcement
- [ ] SQL injection protection (use parameterized queries)

2. Performance

- [ ] Enable response caching for repeated queries
- [ ] Implement connection pooling for Qdrant
- [ ] Add CDN for frontend static assets
- [ ] Optimize embedding batch processing
- [ ] Implement request throttling
- [ ] Database indexing optimization
- [ ] Lazy loading for frontend modules

3. Monitoring & Logging

- [ ] Application Insights / ELK Stack integration
- [ ] API endpoint monitoring
- [ ] Error tracking (Sentry or similar)
- [ ] Performance metrics (response times, throughput)
- [ ] Vector store health checks
- [ ] User activity analytics
- [ ] Alert system for failures

4. Scalability

- [ ] Containerize with Docker
- [ ] Kubernetes orchestration
- [ ] Horizontal scaling for API
- [ ] Load balancer configuration
- [ ] Qdrant cluster setup for high availability
- [ ] Message queue for async processing (RabbitMQ)
- [ ] Redis for session management

5. Data Management

- [ ] Backup strategy for vector database
- [ ] Document versioning system
- [ ] Data retention policies
- [ ] GDPR compliance (data deletion)
- [ ] Audit logs for document operations

6. Quality Assurance

- [ ] Unit tests (>80% coverage)
- [ ] Integration tests for RAG pipeline
- [ ] E2E tests for critical user flows
- [ ] Load testing (Apache JMeter / k6)
- [ ] Security testing (OWASP)

- [ ] API documentation (Swagger/OpenAPI)

7. Configuration

- [ ] Environment-specific configs (Dev/Staging/Prod)
- [ ] Feature flags for gradual rollouts
- [ ] Configurable chunk sizes and overlap
- [ ] Adjustable similarity thresholds
- [ ] Model selection configuration

Deployment Guide

Option 1: Azure Deployment (Recommended)

Backend (.NET API)

```
# 1. Create Azure App Service
az webapp create --name rag-demo-api --resource-group rg-rag-demo --plan asp-rag-demo

# 2. Configure app settings
az webapp config appsettings set --name rag-demo-api --resource-group rg-rag-demo --settings \
  "GitHub__Token=<your-token>" \
  "Qdrant__Url=<qdrant-url>" \
  "Qdrant__ApiKey=<qdrant-key>"

# 3. Deploy from GitHub Actions
# (See deployment pipeline below)
```

Frontend (Angular)

```
# 1. Build production bundle
npm run build -- --configuration production

# 2. Deploy to Azure Static Web Apps
az staticwebapp create --name rag-demo-ui --resource-group rg-rag-demo

# 3. Configure API URL
# Update environment.production.ts with API endpoint
```

Qdrant Vector Database

```
# Option A: Qdrant Cloud (Managed)
# Sign up at cloud.qdrant.io and create cluster
```

```
# Option B: Self-hosted on Azure Container Instances
az container create --name qdrant --resource-group rg-rag-demo \
  --image qdrant/qdrant:latest --ports 6333 6334 \
  --cpu 2 --memory 4
```

Option 2: Docker Deployment

```
# docker-compose.yml
version: '3.8'
services:
  frontend:
    build: ./RAGDemoFrontend
    ports:
      - "80:80"
    environment:
      - API_URL=http://backend:5000

  backend:
    build: ./RAGDemoBackend
    ports:
      - "5000:8080"
    environment:
      - Qdrant__Url=http://qdrant:6333
      - GitHub__Token=${GITHUB_TOKEN}
    depends_on:
      - qdrant

  qdrant:
    image: qdrant/qdrant:latest
    ports:
      - "6333:6333"
    volumes:
      - qdrant-data:/qdrant/storage

volumes:
  qdrant-data:
```

```
# Deploy
docker-compose up -d

# Scale backend
docker-compose up -d --scale backend=3
```

Option 3: Kubernetes Deployment

```yaml
# k8s-deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: rag-demo-api
spec:
  replicas: 3
  selector:
    matchLabels:
      app: rag-demo-api
  template:
    metadata:
      labels:
        app: rag-demo-api
    spec:
      containers:
      - name: api
        image: your-registry/rag-demo-api:latest
        ports:
        - containerPort: 8080
        env:
        - name: Qdrant__Url
          value: "http://qdrant-service:6333"
        resources:
          requests:
            memory: "512Mi"
            cpu: "500m"
          limits:
            memory: "1Gi"
            cpu: "1000m"
---
apiVersion: v1
kind: Service
metadata:
  name: rag-demo-api-service
spec:
  type: LoadBalancer
  ports:
  - port: 80
    targetPort: 8080
  selector:
    app: rag-demo-api
```

CI/CD Pipeline (GitHub Actions)

```yaml
# .github/workflows/deploy.yml
name: Deploy RAG Demo

on:
```

```yaml
  push:
    branches: [ main ]

jobs:
  build-and-deploy-backend:
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v3

    - name: Setup .NET
      uses: actions/setup-dotnet@v3
      with:
        dotnet-version: '8.0.x'

    - name: Build
      run: dotnet build --configuration Release
      working-directory: ./RAGDemoBackend

    - name: Publish
      run: dotnet publish -c Release -o ./publish
      working-directory: ./RAGDemoBackend

    - name: Deploy to Azure
      uses: azure/webapps-deploy@v2
      with:
        app-name: 'rag-demo-api'
        publish-profile: ${{ secrets.AZURE_WEBAPP_PUBLISH_PROFILE }}
        package: './RAGDemoBackend/publish'

  build-and-deploy-frontend:
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v3

    - name: Setup Node
      uses: actions/setup-node@v3
      with:
        node-version: '20'

    - name: Install dependencies
      run: npm ci
      working-directory: ./RAGDemoFrontend

    - name: Build
      run: npm run build -- --configuration production
      working-directory: ./RAGDemoFrontend

    - name: Deploy to Azure Static Web Apps
      uses: Azure/static-web-apps-deploy@v1
      with:
        azure_static_web_apps_api_token: ${{ secrets.AZURE_STATIC_WEB_APPS_TOKEN
}}
        repo_token: ${{ secrets.GITHUB_TOKEN }}
```

```
          action: "upload"
          app_location: "./RAGDemoFrontend/dist"
```

Cost Estimation (Azure - Monthly)

Service	Tier	Estimated Cost
App Service (API)	B1 Basic	$13
Static Web App (Frontend)	Free	$0
Container Instance (Qdrant)	2 vCPU, 4GB	$50
GitHub Models API	Pay-per-use	$5-20
Application Insights	Basic	$2
Total		**$70-85/month**

Cost Optimization Tips

- Use Azure Reserved Instances (save 30-40%)
- Implement caching to reduce AI API calls
- Use Qdrant Cloud free tier for development
- Auto-scaling based on traffic patterns

Success Metrics

Technical KPIs

- Response time < 2 seconds (p95)
- API availability > 99.9%
- Search accuracy > 85%
- Vector search latency < 100ms

Business KPIs

- User engagement rate
- Query success rate
- Document processing throughput
- Cost per query

Next Steps / Roadmap

1. **Short Term (1-2 weeks)**
    - Add user authentication
    - Implement request logging

    - Create comprehensive API documentation

2. **Medium Term (1-2 months)**
    - Multi-language support
    - Advanced search filters
    - Document management dashboard
    - Chat history persistence

3. **Long Term (3-6 months)**
    - Multi-tenant support
    - Custom embedding models
    - Real-time collaboration
    - Mobile app (React Native)
    - Advanced analytics dashboard

Support & Resources

- **Documentation**: See README.md for setup instructions
- **API Docs**: `https://your-api.com/swagger`
- **Qdrant Docs**: https://qdrant.tech/documentation/
- **GitHub Models**: https://github.com/marketplace/models
- **PrimeNG**: https://primeng.org/

Last Updated: January 2026
Version: 1.0
Maintainer: Mohd Waseem