Solution Architecture Document for Azure Marketplace

1. Executive Summary

SmartLib is a powerful Retrieval-Augmented Generation (RAG) web application. It integrates with different vector stores for fast similarity searches and provides an intuitive interface for secure user management, document uploads, and interactive knowledge retrieval with generation capabilities. This document outlines the solution architecture for deployment on the Azure Marketplace.

2. Solution Overview

2.1 Purpose and Scope

The SmartLib application provides an intuitive interface for users to leverage the power of Retrieval-Augmented Generation. It allows for: - Secure user authentication and management - Document upload and vector processing - Intelligent document retrieval and generation - Integration with various AI models and vector stores

2.2 Target Users

- Knowledge management professionals
- Enterprise content managers
- Research organizations
- Educational institutions

3. Architecture Components

3.1 High-Level Architecture

The SmartLib solution consists of the following key components:

```
User Interface Layer
    ↓
Application Layer (Web Application)
    ↓
Service Layer (Authentication, Vector Store, OCR, Generation)
    ↓
Data Layer (Document Storage, Vector Indices, User Database)
```

3.2 Component Details

3.2.1 User Interface Layer

- Web-based interface for user interactions
- Responsive design for multiple device types

- Modern UI/UX with intuitive navigation

3.2.2 Application Layer

- Web framework
- Route handlers for request processing
- Session management
- Error handling and logging

3.2.3 Service Layer

- Authentication service
- Vector store service (ChromaDB by default, PGVector optional)
- Document processing service with OCR capabilities
- Generation service

3.2.4 Data Layer

- Document storage
- Vector indices (stored in ChromaDB file-based storage by default)
- User database (SQLite by default, stored on Azure Files)
- Session data

4. Azure Resource Requirements

4.1 Compute Resources

- Existing (user deployed)
- Azure Open Ai with model deployment
- Azure Keyvault
- Azure Storage Account (file share to hold data)
- Azure Redis cache
- Azure Document Intelligence (optional) Smartlib deployment via ARM Template:
- Azure App Service Plan: Minimum B1 (1 core, 1.75GB memory), recommended P1v2 (2 cores, 7GB memory)

4.2 Storage Resources

- Azure Files for shared data volume (/home/data)
- SQLite database (default, stored on Azure Files)

4.3 Authentication Resources

- Azure Active Directory B2C (optional for enterprise deployments)
- Key Vault for secret management

4.4 AI and Cognitive Resources

- Azure OpenAI Service for language model integration
- Azure Document Intelligence (optional) for advanced OCR capabilities

4.5 Networking Resources

- Virtual Network (optional)
- Application Gateway (optional for high-security deployments)
- Private Link (for secure database connections)

5. Deployment Architecture

5.1 Standard Deployment (Default with ChromaDB and SQLite)

The standard deployment uses Azure App Service with SQLite as the database and ChromaDB for vector storage:

```
Azure App Service (Web App) ← Azure Application Insights
        ↓
Azure App Service (Worker) ← Azure Cache for Redis (Task Queue)
        ↓
Azure Files (SQLite DB + ChromaDB Vectors + Shared Data) ← Azure Backup Service (User config
        ↓
Azure Storage Account ← Azure File share ← Azure CDN (for static assets) - User configuratio
```

6. Integration Points

6.1 External API Integrations

- Azure OpenAI Service for language model integration, (Customer-provided LLM endpoints) on Provision ArmTemplate
- Azure Document Intelligence for advanced OCR capabilities, (Customer-provided LLM endpoints) on Provision ArmTemplate
- Azure Keyvault (Customer-provided LLM endpoints) on Provision ArmTemplate

6.2 Authentication Integrations

- Microsoft Account authentication
- Custom authentication providers (local)

6.3 Data Source Integrations

- Not applicable in this version.

7. OCR Integration

7.1 OCR Capabilities

SmartLib offers dual OCR processing options to accommodate different deployment scenarios:

1. **Local OCR Processing**
   - Built into the worker container
   - No additional Azure services required
   - Good for basic OCR needs and air-gapped environments
   - Uses CPU and memory resources on the worker container
   - Need more memory to produce fast result since it use CPU.
   - If enable virtual grounding it will add more task to save the image
2. **Azure Document Intelligence**
   - Cloud-based advanced document processing
   - Superior accuracy for complex documents and layouts
   - Extensive language support and table extraction
   - Offloads processing from worker container

7.2 OCR Architecture

Local OCR Processing Flow:

```
Document Upload → Worker Processing → Local OCR
    → Text Extraction → Vector Store
```

Azure Document Intelligence Flow:

```
Document Upload → Worker Pre-processing → Azure Document Intelligence API
    → Worker Post-processing → Text Extraction → Vector Store
```

7.3 OCR Configuration

- Admin interface for OCR method selection
- Document Intelligence API key stored securely in Key Vault
- Environment variables set during deployment for Document Intelligence endpoint and key reference
- Dynamic switching between OCR methods without redeployment (need restart web & worker service to apply)

8. Security Architecture

8.1 Data Security

- All data encrypted at rest and in transit
- Azure Storage encryption (AES-256)
- TLS 1.2+ for all communications

8.2 Authentication Security

- Role-based access control (RBAC)

8.3 Network Security (user can add or customize if needed)

- Azure DDoS Protection
- Web Application Firewall (WAF)
- Private endpoints for internal resources

9. Scalability and Performance

9.1 Scaling Mechanisms

- Auto-scaling based on CPU, memory, and request metrics

9.2 Performance Optimizations

- Azure CDN for static assets
- Azure Cache for Redis for task queue and optional session storage
- Query optimization for vector search operations
- OCR processing offloading to Azure Document Intelligence

10. Availability and Disaster Recovery

Since it works on local each azure web apps, user can copy each database since it save in azure storage files.

10.1 High Availability

- No applied yet

10.2 Disaster Recovery (Adjust in Azure Storage Account)

- Regular automated backups or copy azure storage file
- Point-in-time recovery for databases
- Geographic replication for critical data

11. Monitoring and Management

11.1 Monitoring

- Azure Application Insights integration
- Custom metrics and dashboards
- Alert rules for critical conditions

11.2 Management

- Azure Portal integration
- Azure CLI support
- Infrastructure as Code (ARM templates)

12. Conclusion

This solution architecture provides a robust, scalable, and secure foundation for deploying the SmartLib application to the Azure Marketplace. The architecture leverages Azure's native services to ensure optimal performance, security, and reliability while providing customers with flexibility in deployment options based on their specific requirements.

By default, SmartLib uses SQLite for the application database and ChromaDB for vector storage, both stored on Azure Files. This provides a simple, cost-effective solution that works well for most deployments. For users with specific requirements, the architecture supports optional PGVector deployment using Azure Database for PostgreSQL (Not applicable in this version) and Azure Document Intelligence for advanced document processing capabilities.