cognizant

CogniSense

AI Personal Assistant - Architecture

Version 1

Table of Contents

Revision History

Version	Effective Date	Change Description	Affected sections	Prepared by
1	01/06/2025	Initial creation	All	Truc Duong

Introduction

Background and Project Objective

Design and Build a GenAI-based Personal Assistant that enables the following:

• Users can upload or link the content like blogs, official tool documentation, group discussion, meeting notes to the system.

• Clients should be able to ask questions related to their own data and get answers.

• The system can generate flashcards and quizzes based on a given topic and chat history.

• The system utilizes Azure services.

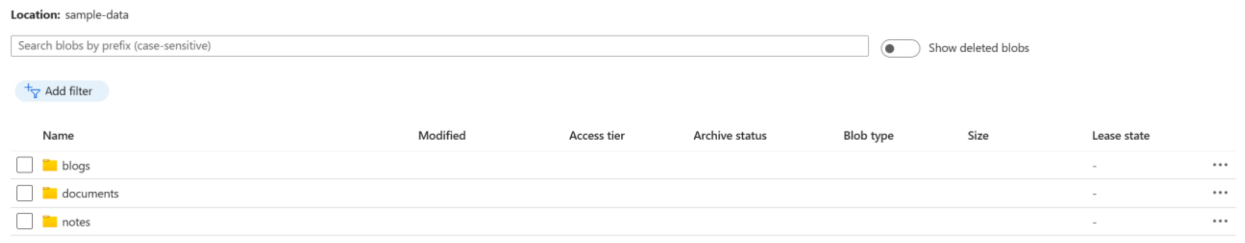
Technology Components

This section defines the technologies and services used in each component of the data platform.

Components List

Data Sources

Sources are stored in Azure Blob Storage and categorized into 3 types corresponding to 3 different containers as follows.



Location: sample-data

Search blobs by prefix (case-sensitive)

Show deleted blobs

Add filter

Name	Modified	Access tier	Archive status	Blob type	Size	Lease state
blogs						-
documents						-
notes						-

Category 1 – Blogs: These are the website links clients paste to add new documents to knowledge base.

Category 2 - Documents: Any text file that are in 3 formats, including PDF, DOCX, TXT.

Category 3 - Notes: Any kind of notes and chat logs which must be saved as TXT file first.

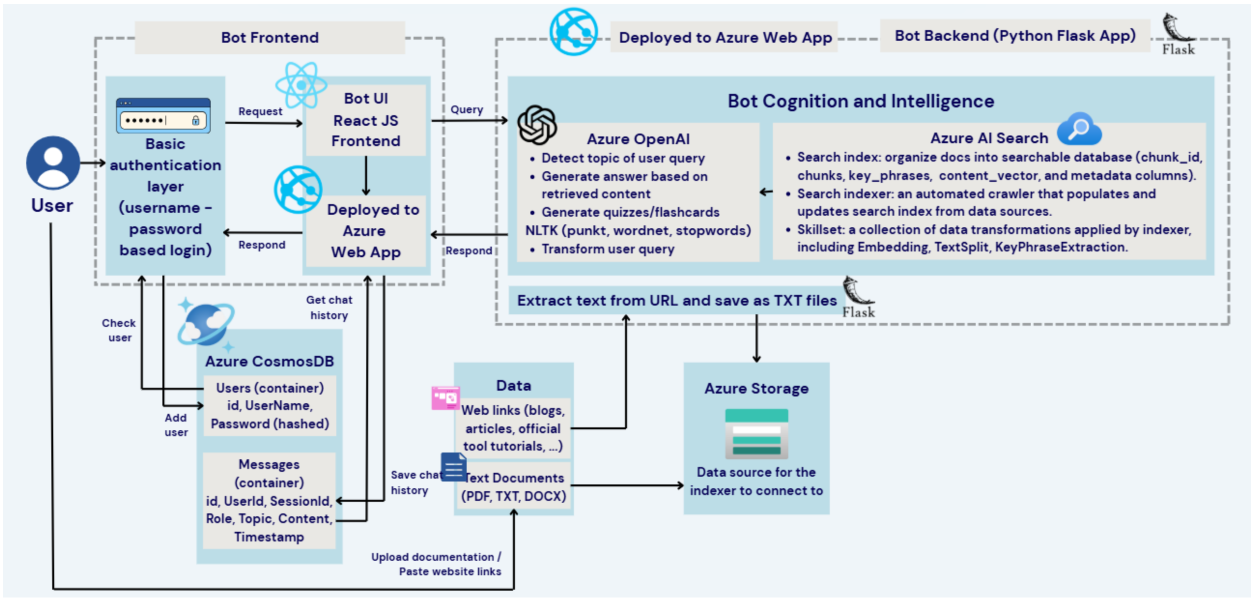
Solution Components

Components List and Description

Component	Resource Name	Description	Functionality
Azure Web App	bot-backend	Host the backend logic layer powering the AI personal assistant's conversational and content-processing features	Manages tasks like scraping content from URL, querying knowledge, generating responses, flashcards, or quizzes
Azure Web App	cognisense	Host the frontend interface for users to add content to the knowledge base, interact with chatbot, and access learning tools (quizzes, flashcards)	Send data to backend and Azure services then return the result once processed successfully, ensuring users have a seamless experience
App Service Plan	ASP-raxsharing-be21	Allocates the computing resources (CPU, memory, and storage) needed to run the web applications.	Enable hosting, scaling, and managing the performance of 2 web apps within a defined pricing tier
Azure AI Foundry	truc-may2898u-eastus2	Simplifies the development and deployment of GenAI applications with prebuilt components	Streamlines building, evaluating, and deploying GenAI workflows with minimal setup
Azure Cosmos DB account	cognisense-cdb	A fully managed NoSQL and relational database for modern app development, offering single-digit millisecond response time, instant scalability, along with guaranteed speed at any scale.	Stores user credentials with username and password (hashed form), and chat conversation with user ID, session ID, role, topic, content, timestamp
Azure OpenAI	cognisense-model	Creates embedding model and chat completion model	Takes prompts, documents, generates relevant responses

Managed Identity	debugsessionmanagedidentity	Provides an automatically managed identity for Azure services to access other Azure resources securely	Enables secure, password-less authentication to services like Azure Storage or Key Vault without managing credentials manually
Azure Search service	cognisense-vector-search	Gives developers infrastructure, APIs, and tools for building a rich search experience over private, heterogeneous content in web, mobile, and enterprise applications	Index document into searchable chunks, adds metadata, using skillset in Azure AI Search for chunking, embedding, and key phrase extraction
Azure Storage account	cognisensekbsa	Stores massive amount of unstructured dataset	Stores uploaded documents and extracted website content and serves as a source for indexing

System Architecture



Bot Frontend
Deployed to Azure Web App
Bot Backend (Python Flask App)
Flask
User
Basic authentication layer (username - password based login)
Request
Bot UI React JS Frontend
Query
Deployed to Azure Web App
Respond
Respond
Bot Cognition and Intelligence
Azure OpenAI
• Detect topic of user query
• Generate answer based on retrieved content
• Generate quizzes/flashcards
NLTK (punkt, wordnet, stopwords)
• Transform user query
Azure AI Search
• Search index: organize docs into searchable database (chunk_id, chunks, key_phrases, content_vector, and metadata columns).
• Search indexer: an automated crawler that populates and updates search index from data sources.
• Skillset: a collection of data transformations applied by indexer, including Embedding, TextSplit, KeyPhraseExtraction.
Extract text from URL and save as TXT files
Flask
Check user
Get chat history
Azure CosmosDB
Users (container) id, UserName, Password (hashed)
Add user
Messages (container) id, UserId, SessionId, Role, Topic, Content, Timestamp
Save chat history
Data
Web links (blogs, articles, official tool tutorials, ...)
Text Documents (PDF, TXT, DOCX)
Azure Storage
Data source for the indexer to connect to
Upload documentation / Paste website links

This diagram outlines the complete architecture of a GenAI-powered personal assistant web application designed to help users manage and interact with unstructured content such as documents, URLs, and notes. The system supports intelligent querying, summarization, and knowledge retention features like quizzes and flashcards.

User Interaction Layer

- **Authentication**
    - The user initiates interaction via a username-password-based login.
    - Login credentials are authenticated against the Azure CosmosDB Users container.
- **Bot Frontend (UI)**
    - A ReactJS-based UI is deployed as an Azure Web App.
    - Allows users to:
        - Upload documents (PDF, TXT, DOCX).
        - Paste external URLs.
        - Ask questions to the assistant.
        - Trigger flashcard/quiz generation.
    - Sends authenticated requests to the backend via HTTP.

Backend Services (Bot Cognition and Intelligence)

- **Bot Backend (Flask App)**
    - A Python-based Flask application also deployed as an Azure Web App, which handles:
        - Query processing
        - Text extraction
        - Communication with OpenAI, Azure AI Search, CosmosDB, and Azure Storage
- **Azure OpenAI**
    - Uses models to:
        - Detect the intent and topic of the user's question
        - Answer questions using relevant content from the personal knowledge base
        - Generate quizzes or flashcards for active recall
        - Utilize NLTK for basic NLP tasks like lemmatization, stopword removal
- **Azure AI Search**
    - Core search engine to index and retrieve user data using semantic and keyword-based search:

        - Search Index: Builds a searchable structure with fields like chunk_id, chunks, key_phrases, content_vector, etc.
        - Skillset: Predefined transformations used during indexing such as:
            - Text splitting
            - Key phrase extraction
            - Embedding generation for vector search
        - Search Indexer: Automatically pulls data from Azure Storage and updates the index

Data Ingestion and Storage

- **Content Types**
    - Input includes:
        - Web links (blogs, tool documentation, articles)
        - Text documents (PDF, TXT, DOCX)
    - Extracted using the backend, which:
        - Converts content into .TXT format
        - Saves to Azure Storage as the source for indexing
- **Azure Storage**
    - Acts as the primary content repository.
    - Serves as a data source for Azure AI Search indexer.
- **Azure CosmosDB**
    - Stores structured application metadata:
        - **Users container**: Authenticated users with fields like id, username, and hashed password
        - **Messages container**: Conversation history containing sessionId, topic, content, and timestamp

System Flow Summary

1. User logs in (auth verified via CosmosDB).
2. Uploads or pastes link via frontend.
3. Flask backend extracts and stores text into Azure Storage.
4. Azure AI Search indexer processes the new content and updates search index.
5. User interacts with chatbot via React UI:
    - Questions sent to Flask backend.

    - Queries processed by OpenAI and enriched with Azure AI Search results.

6. Generated responses or learning materials (quizzes/flashcards) are returned to the user.