

アーキテクチャ図「Qnect」

[ユーザー]

| (Web ブラウザ経由)



[Streamlit UI (Qnect) / アプリケーションロジック (Python)]

| (メインアプリ, 履歴検索ページ, DB スキーマ管理ページ)

|

|— (SQL 実行/スキーマ取得) —————→ [SQL Server] [2]

|

| (DB データ/スキーマ情報)

|

|— (取得結果)

|

|— (スキーマ.md 読書き) —————→ [ローカルストレージ (src/prompts_nltosql/)] [1][2]

|— (マスキング設定.json 読込み) —————→ [ローカルストレージ (config/data_masking/)]

|— (チャット履歴.json 読書き) —————→ [ローカルストレージ (data/chat_history.json)]

|

| (1) 質問/チャット入力/要約対象データ/ベクトル化対象テキスト

| (2) SQL 生成/応答生成/要約生成/Embedding 指示 (+プロンプト(スキーマ.md[1][2], 履歴等))

|—————→ [Azure OpenAI]

|— Embedding API (テキストベクトル化)

|— Chat Completion API (Chat モデル) [(SQL/応答/要約生成)]

|

| (3) 生成 SQL/応答/要約/テキストベクトル



[Streamlit UI (Qnect) / アプリケーションロジック (Python)]

| (4) チャット履歴(ベクトル含)アップロード

| (6) 類似チャット履歴検索指示(ベクトル+キーワード)

|—————→ [Azure AI Search] ◀— (チャット履歴ベクトル DB(HNSW)+キーワード)

| (ID, 質問, SQL, 要約, 要約ベクトル, タイムスタンプ)

| (5) アップロード結果 / (7) 検索結果(関連チャット)



[Streamlit UI (Qnect) / アプリケーションロジック (Python)]

|

| (8) チャット履歴保存指示 / (10) チャット履歴検索指示(キーワード)

|—————→ [Azure Cosmos DB] ◀— (チャット履歴 DB)

| (ID, 質問, SQL, 結果サンプル, 要約, タイムスタンプ)

| (9) 保存結果 / (11) 検索結果(チャット)



[Streamlit UI (Qnect) / アプリケーションロジック (Python)]

| (各種処理結果を UI に表示)



[ユーザー]

[1] ローカルストレージ (src/prompts_nltosql/)

[2] SQL Server

フロー説明:

1.ユーザーが Streamlit UI を通じて操作:

ユーザーは Web ブラウザで表示された Qnect の画面から、自然言語での質問入力、チャットメッセージの送信、履歴検索のためのキーワード入力、DB スキーマ取得対象テーブルの選択などを行います。

2.アプリケーションロジックが Azure OpenAI へ指示:

Streamlit アプリケーション内の Python ロジックが、ユーザー入力や保存されている情報（例: src/prompts_nltosql/ 内のスキーマ Markdown、過去のチャット履歴）を基に、Azure OpenAI に対して処理を依頼します。

- **SQL 生成:** ユーザーの自然言語の質問と関連スキーマ情報を組み合わせて、SQL クエリの生成を依頼します。
- **応答生成:** チャット形式での対話において、過去のやり取りや検索結果を考慮して、ユーザーへの返答を生成するよう依頼します。
- **要約生成:** SQL 実行結果のサンプルデータを基に、その内容を分かりやすく要約するよう依頼します。
- **Embedding 指示:** テキスト（質問や要約文など）をベクトル化するよう依頼します。

3.Azure OpenAI が結果を生成:

指示を受けた Azure OpenAI は、GPT モデルや Embedding モデルを使って、SQL クエリ、チャットの返答、要約文、またはテキストのベクトルデータを生成し、アプリケーションロジックに返します。

4.チャット履歴（ベクトル含む）を Azure AI Search へアップロード:

メインアプリケーションで SQL 生成から要約までの一連のやり取りが完了した際や、チャットでの対話が記録される際に、その内容（質問、生成 SQL、AI による要約など）と、要約文を Azure OpenAI Embedding API でベクトル化したデータを Azure AI Search にアップロードします。これにより、後で類似の質問や内容を効率的に検索できるようになります。

5.Azure AI Search からのアップロード結果受信:

Azure AI Search は、データのアップロードが成功したかどうかの結果をアプリケーションロジックに返します。

6.Azure AI Search へ類似チャット履歴の検索を指示:

ユーザーが履歴検索ページでキーワードを入力した際、またはメインアプリのチャット機能で AI が過去の情報を参照する必要がある場合、アプリケーションロジックは入力されたキーワード（およびキーワードをベクトル化したもの）を **Azure AI Search** に送り、類似する過去のチャット履歴の検索を指示します。

7.Azure AI Search から検索結果を受信:

Azure AI Search は、保存されているチャット履歴の中から、指示されたベクトルやキーワードに類似するものを探し出し、その結果（関連する過去の質問、**SQL**、要約など）をアプリケーションロジックに返します。

8.チャット履歴を Azure Cosmos DB へ保存指示:

メインアプリケーションで **SQL** 生成から要約までの一連のやり取りが完了した際や、チャットでの対話が記録される際に、その内容（質問、生成 **SQL**、**SQL** 実行結果のサンプル、AI による要約、タイムスタンプなど）を **Azure Cosmos DB** に保存するよう指示します。**Azure Cosmos DB** は、リレーショナルな表形式のデータ（構造化データ）はもちろん、**JSON** などの柔軟な形式を持つ半構造化データも、スキーマを事前に定義せずに自在に保存できるクラウドデータベースサービスです。

9.Azure Cosmos DB からの保存結果受信:

Azure Cosmos DB は、データの保存が成功したかどうかの結果をアプリケーションロジックに返します。

10.Azure Cosmos DB へチャット履歴の検索を指示:

ユーザーが履歴検索ページでキーワードを入力した際、アプリケーションロジックは **Azure Cosmos DB** に対しても、そのキーワードを含むチャット履歴の検索を指示します。これは **Azure AI Search** によるベクトル検索とは別の、キーワードベースの検索です。

11.Azure Cosmos DB から検索結果を受信:

Azure Cosmos DB は、キーワードに合致するチャット履歴のデータを検索し、その結果をアプリケーションロジックに返します。