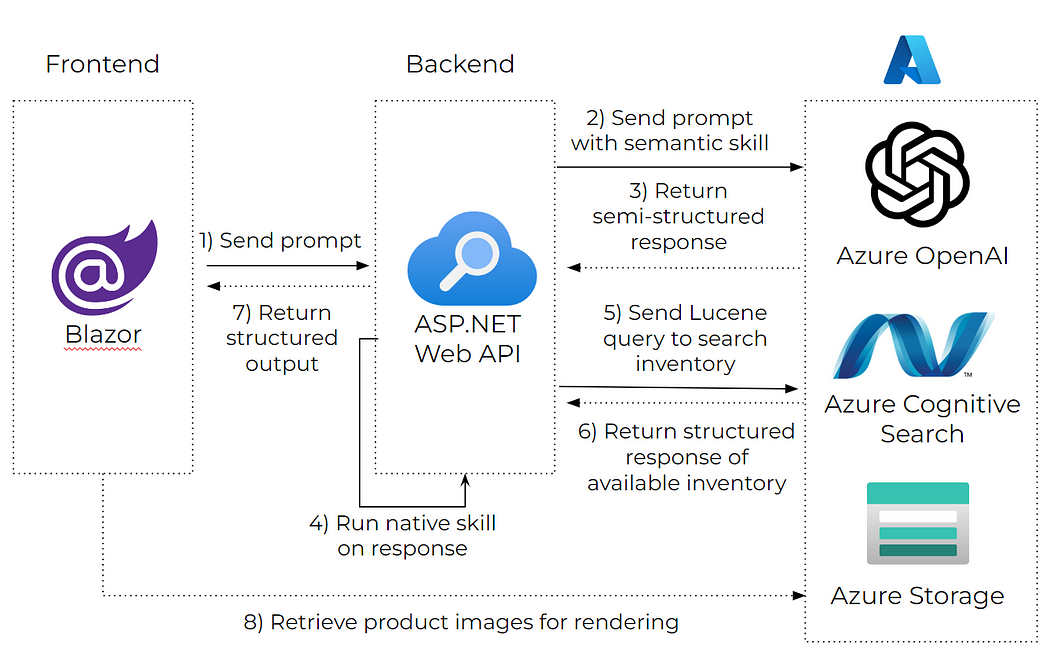
**ChatGPT Meets Enterprise data with Azure OpenAI and Cognitive Search**

**E-Commerce solution with Generative AI that integrates Angular, .NET Core, Azure API Management (APIM), and Azure Cognitive Search**

A diagram of a software process

Description automatically generated

**Example:**

**Angular app to fetch data from Azure Cognitive Search via APIs exposed through APIM, integrating seamlessly with Azure OpenAI for intelligent responses.**

 **Frontend**: Angular (UI/UX for the e-commerce platform).

 **Backend**: .NET Core APIs for business logic (Product, Inventory, Orders, User Info).

 **Search**: Azure Cognitive Search for intelligent search capabilities.

 **AI**: Azure OpenAI for generative AI prompts and user interactions.

 **Integration Layer**: Azure API Management for centralized API management and security.

Yes, you can integrate all your service-related APIs (like Inventory Service, Product Service, User Info, Order Service, etc.) with Azure AI Studio, but it requires a structured approach. Azure AI Studio allows you to create AI models and integrate them with your applications.

**Architecture Overview**

1. **Frontend**: Angular (UI/UX for the e-commerce platform).
2. **Backend**: .NET Core APIs for business logic (Product, Inventory, Orders, User Info).
3. **Search**: Azure Cognitive Search for intelligent search capabilities.
4. **AI**: Azure OpenAI for generative AI prompts and user interactions.
5. **Integration Layer**: Azure API Management for centralized API management and security.

Here's how you can approach this:

**1. Centralize Your APIs**

* If your services are already hosted as APIs (e.g., REST or gRPC), ensure they are well-documented and accessible.
* You can use API Management tools like **Azure API Management** to centralize, secure, and manage all your APIs. This provides a single gateway for your services.

**2. Connect APIs to Azure AI Studio**

Azure AI Studio can interact with your APIs in the following ways:

* **Custom AI Models:** If you are building custom AI models (e.g., using Azure OpenAI or Machine Learning), these models can call your APIs to fetch data or trigger actions.
* **Azure Functions:** Create Azure Functions that act as connectors between your AI models and your APIs. These functions can call your services and process the data as needed.
* **Logic Apps:** Use Azure Logic Apps to orchestrate API calls and integrate them with Azure AI Studio workflows.

**3. Use Azure Cognitive Services**

If you are using Azure Cognitive Services or Azure OpenAI, you can:

* Call your APIs from within custom AI models to retrieve real-time data.
* Use APIs to augment AI model responses (e.g., fetching real-time inventory details or user-specific data).

**4. Set Up Security**

* Use **Azure Active Directory (AAD)** for authentication and authorization.
* Ensure APIs are secured using tokens (OAuth2, JWT) or other mechanisms.
* Restrict access to sensitive services and data.

**5. Build a Unified Data Layer**

* If the data from your APIs is frequently needed for AI workflows, consider creating a **data lake** or a **data warehouse** (e.g., Azure Data Lake or Azure Synapse Analytics).
* Use Azure Data Factory to extract, transform, and load (ETL) data from your APIs into a centralized data store.

**6. Train and Fine-Tune Models**

* Use the data from your APIs to train custom AI models in Azure Machine Learning or Azure OpenAI.
* Fine-tune models to incorporate business-specific logic (e.g., prioritizing certain products, user behaviors, etc.).

**7. Monitor and Optimize**

* Use **Azure Monitor** to track API performance and ensure smooth integration.
* Continuously optimize AI models and API endpoints based on usage patterns and feedback.

**Example Workflow**

1. **User Request:** A user requests product details.
2. **AI Model Interaction:** The Azure OpenAI model interprets the request.
3. **API Call:** The model calls the Product Service API to fetch product details.
4. **Response:** The AI model processes and formats the data, then sends it back to the user.

|  |
| --- |
| **Query Azure Cognitive Search**  [HttpGet("search")]  public async Task<IActionResult> SearchProducts(string query)  {  var searchClient = new SearchClient(  new Uri("**https://<your-search-service>.search.windows.net"),**  "<index-name>",  new AzureKeyCredential("<your-search-key>")  );  var options = new **SearchOptions**  {  IncludeTotalCount = true,  Filter = "inventoryStatus eq 'InStock'"  };  var results = await **searchClient.SearchAsync<Product>(query, options);**  return Ok(results.Value);  } |
| **Integrate Azure OpenAI**  [HttpPost("generate-query")]  public async Task<IActionResult> GenerateQuery([FromBody] string userInput)  {  var openAiClient = new **OpenAIClient**(new Uri("https://<your-openai-service>.openai.azure.com"),  new AzureKeyCredential("<your-openai-key>"));  var completion = await **openAiClient**.GetCompletionsAsync("<deployment-name>", new CompletionsOptions  {  Prompts = { $"Enhance this search query for e-commerce: {userInput}" },  MaxTokens = 50  });  var enhancedQuery = completion.Value.Choices.First().Text;  return Ok(enhancedQuery.Trim());  } |
| **Call Search API**  searchProducts(query: string) {  this.http.get<Product[]>(`/api/search?query=${query}`).subscribe(results => {  this.products = results;  });  } |

**setting up Azure Cognitive Search**, **configuring APIM**, and **building the Angular frontend**

|  |
| --- |
| **1. Setting Up Azure Cognitive Search**  **Step 1.1: Create a Search Service**   1. Go to the [Azure Portal](https://portal.azure.com/). 2. Search for **"Cognitive Search"** and click **Create**. 3. Provide the required details:    * Subscription    * Resource Group    * Search Service Name    * Pricing Tier (Start with "Free" or "Basic" for testing) 4. Once created, access the service and note the **API Key** and **Endpoint**.   **Step 1.2: Define a Search Index**  An index is the core structure for search queries. You define it based on your data structure.   1. Use the **Azure Portal** or **Azure SDKs**. 2. Example Index Schema (e.g., for a product catalog):   json  Copy code  {  "name": "product-index",  "fields": [  { "name": "productId", "type": "Edm.String", "key": true, "searchable": false },  { "name": "productName", "type": "Edm.String", "searchable": true },  { "name": "description", "type": "Edm.String", "searchable": true },  { "name": "price", "type": "Edm.Double", "filterable": true },  { "name": "inventoryStatus", "type": "Edm.String", "filterable": true }  ]  }   1. Populate the index using:    * Azure SDKs    * REST API    * Azure Data Factory   **Step 1.3: Query the Search Index**   * Example using Azure SDK in .NET:   csharp  Copy code  var client = new SearchClient(  new Uri("https://<your-search-service>.search.windows.net"),  "product-index",  new AzureKeyCredential("<your-search-key>")  );  var options = new SearchOptions  {  Filter = "price gt 100 and inventoryStatus eq 'InStock'",  OrderBy = { "price desc" }  };  var results = await client.SearchAsync<Product>("laptop", options);  foreach (var result in results.Value.GetResults())  {  Console.WriteLine(result.Document.ProductName);  }  **2. Configuring Azure API Management (APIM)**  **Step 2.1: Create an APIM Instance**   1. In the Azure Portal, search for **"API Management"** and click **Create**. 2. Provide details:    * Subscription    * Resource Group    * Instance Name    * Pricing Tier (Developer for testing)   **Step 2.2: Import APIs**   1. Go to the APIM instance. 2. Under **APIs**, click **Add API**. 3. Import:    * **OpenAPI Specification** (for .NET Core APIs)    * **Function App** (if using Azure Functions)    * **HTTP** (manual setup for REST APIs)   **Step 2.3: Apply Policies**  Policies allow you to handle caching, rate limiting, authentication, etc.   * **Example Policy: Rate Limiting**   xml  Copy code  <rate-limit-by-key calls="10" renewal-period="60" counter-key="@((context.Request.Headers.GetValueOrDefault("x-user-id", "anonymous")))" />   * **CORS Policy**:   xml  Copy code  <cors>  <allowed-origins>  <origin>https://your-angular-app.com</origin>  </allowed-origins>  <allowed-methods>  <method>\*</method>  </allowed-methods>  <allowed-headers>  <header>\*</header>  </allowed-headers>  </cors>  **3. Building the Angular Frontend**  **Step 3.1: Angular Setup**   1. Create a new Angular project:   bash  Copy code  ng new e-commerce-app  cd e-commerce-app   1. Install dependencies:   bash  Copy code  npm install @angular/material @angular/cdk @angular/flex-layout  npm install @azure/msal-browser  **Step 3.2: Add a Search Component**   1. Generate a component:   bash  Copy code  ng generate component search   1. Update the template (search.component.html):   html  Copy code  <mat-form-field appearance="fill">  <mat-label>Search Products</mat-label>  <input matInput (keyup.enter)="search()" [(ngModel)]="query" />  </mat-form-field>  <button mat-raised-button color="primary" (click)="search()">Search</button>  <div \*ngIf="products.length">  <h2>Search Results:</h2>  <mat-list>  <mat-list-item \*ngFor="let product of products">  {{ product.productName }} - ${{ product.price }}  </mat-list-item>  </mat-list>  </div>   1. Update the TypeScript (search.component.ts):   typescript  Copy code  import { HttpClient } from '@angular/common/http';  import { Component } from '@angular/core';  @Component({  selector: 'app-search',  templateUrl: './search.component.html',  styleUrls: ['./search.component.css']  })  export class SearchComponent {  query: string = '';  products: any[] = [];  constructor(private http: HttpClient) {}  search() {  this.http  .get<any[]>(`/api/search?query=${this.query}`)  .subscribe((data) => {  this.products = data;  });  }  }  **Step 3.3: API Integration**   1. Configure proxy for development (proxy.conf.json):   json  Copy code  {  "/api": {  "target": "https://<your-apim-instance>.azure-api.net",  "secure": true,  "changeOrigin": true  }  }   1. Add proxy configuration to angular.json:   json  Copy code  "architect": {  "serve": {  "options": {  "proxyConfig": "src/proxy.conf.json"  }  }  } |