Code Documentation  
Document Search Bot API

Contents

[Code Documentation: Document Search Bot API 3](#_Toc161927381)

[**Introduction** 3](#_Toc161927382)

[**Components** 3](#_Toc161927383)

[**1. Login Controller** 3](#_Toc161927384)

[**2. Document Processing Controller** 4](#_Toc161927385)

[**3. Chat Controller** 6](#_Toc161927386)

[**4. Third-Party Libraries** 12](#_Toc161927387)

[**5. Azure Blob Storage Configuration with Azurite** 13](#_Toc161927388)

[**Integration** 13](#_Toc161927389)

[**Deployment** 13](#_Toc161927390)

[**Conclusion** 13](#_Toc161927391)

# Code Documentation: Document Search Bot API

**Introduction**

The Document Search Bot is designed to provide functionality for handling user authentication, document processing, and chat interactions. This document outlines the architecture, endpoints, third-party libraries, and Azure Blob Storage configuration using Azurite.

**Components**

**1. Login Controller**

* **Purpose:** Handles user authentication by validating login credentials against hardcoded values.
* **Endpoints:**
  + **POST /api/login:** Validates user credentials and returns the user's role and whether they are an admin.
* **Code:**

using Microsoft.AspNetCore.Mvc;

using Query\_Quasar\_Bot\_API.Models;

using System;

namespace Query\_Quasar\_Bot\_API.Controllers

{

[Route("api/[controller]")]

[ApiController]

public class LoginController : ControllerBase

{

[HttpPost]

public IActionResult Login(LoginRequest request)

{

try

{

// Hardcoded credentials

const string adminEmail = "admin@example.com";

const string adminPassword = "adminpassword";

const string userEmail = "user@example.com";

const string userPassword = "userpassword";

// Validate request

if (string.IsNullOrEmpty(request.UserMail) || string.IsNullOrEmpty(request.Password))

{

// Return BadRequest if user email or password is empty

return BadRequest(new { message = "UserMail and Password are required." });

}

// Check if the provided credentials match the hardcoded values

bool isAdmin = false;

if (request.UserMail == adminEmail && request.Password == adminPassword)

{

// Set isAdmin to true if admin credentials are provided

isAdmin = true;

}

else if (request.UserMail == userEmail && request.Password == userPassword)

{

// No need to set isAdmin = false here as it's already initialized to false

}

else

{

// Return Unauthorized if credentials do not match any predefined values

return Unauthorized(new { message = "Invalid credentials." });

}

// Return the user's role and whether they are admin

return Ok(new { message = "Login successful.", isAdmin });

}

catch (Exception ex)

{

// Return internal server error if an exception occurs

return StatusCode(500, new { message = $"An error occurred: {ex.Message}" });

}

}

}

}

**2. Document Processing Controller**

* **Purpose:** Manages document upload, deletion, and retrieval functionalities using Azure Blob Storage.
* **Endpoints:**
  + **POST /api/documents/upload:** Uploads a document to Azure Blob Storage.
  + **DELETE /api/documents/delete/{fileName}:** Deletes a document from Azure Blob Storage.
  + **GET /api/documents/list:** Retrieves a list of documents stored in Azure Blob Storage.
* **Code:**

using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Mvc;

using Azure.Storage.Blobs;

using System.IO;

using System.Threading.Tasks;

using System.Text;

using Aspose.Pdf.Text;

// BlobStorageController handles file upload, deletion, and retrieval from Azure Blob Storage

[ApiController]

[Route("api/documents")]

public class BlobStorageController : ControllerBase

{

private const string ConnectionString = "DefaultEndpointsProtocol=https;AccountName=genaipocvj;AccountKey=2cVv6ef9sJwEs/wrf9gHARm1jvPLdOTQiJkmP5wzr30SWYHj9duvE5o7CRM9N9As1nBgVBENqWoa+AStkueI5A==;EndpointSuffix=core.windows.net";

private const string ContainerName = "genaipocvj";

// Handles file upload to Azure Blob Storage

[HttpPost("upload")]

public async Task<IActionResult> UploadFile(IFormFile file)

{

try

{

var blobServiceClient = new BlobServiceClient(ConnectionString);

var blobContainerClient = blobServiceClient.GetBlobContainerClient(ContainerName);

await blobContainerClient.CreateIfNotExistsAsync();

var blobClient = blobContainerClient.GetBlobClient(file.FileName);

using (var stream = file.OpenReadStream())

{

await blobClient.UploadAsync(stream, true);

}

return Ok(new { Message = "File uploaded successfully" });

}

catch (Exception ex)

{

// Return 500 Internal Server Error if an exception occurs during upload

return StatusCode(StatusCodes.Status500InternalServerError, new { Error = ex.Message });

}

}

// Handles file deletion from Azure Blob Storage

[HttpDelete("delete/{fileName}")]

public async Task<IActionResult> DeleteFile(string fileName)

{

try

{

var blobServiceClient = new BlobServiceClient(ConnectionString);

var blobContainerClient = blobServiceClient.GetBlobContainerClient(ContainerName);

var blobClient = blobContainerClient.GetBlobClient(fileName);

if (await blobClient.ExistsAsync())

{

await blobClient.DeleteIfExistsAsync();

return Ok(new { Message = "File deleted successfully" });

}

else

{

// Return 404 Not Found if the file does not exist

return NotFound(new { Error = "File not found" });

}

}

catch (Exception ex)

{

// Return 500 Internal Server Error if an exception occurs during deletion

return StatusCode(StatusCodes.Status500InternalServerError, new { Error = ex.Message });

}

}

// Handles retrieval of blob list from Azure Blob Storage

[HttpGet("list")]

public IActionResult GetBlobList()

{

try

{

var blobServiceClient = new BlobServiceClient(ConnectionString);

var blobContainerClient = blobServiceClient.GetBlobContainerClient(ContainerName);

var blobs = blobContainerClient.GetBlobs();

var blobDetails = blobs.Select(blobItem => new

{

Name = blobItem.Name,

Size = blobItem.Properties.ContentLength, // Get the size of the blob

}).ToList();

return Ok(blobDetails);

}

catch (Exception ex)

{

// Return 500 Internal Server Error if an exception occurs during retrieval

return StatusCode(StatusCodes.Status500InternalServerError, new { Error = ex.Message });

}

}

}

**3. Chat Controller**

* **Purpose:** Facilitates user interactions by extracting content from documents stored in Azure Blob Storage and generating responses using OpenAI.
* **Endpoints:**
  + **POST /api/chat/ask:** Accepts user queries, searches for relevant content, and generates responses using OpenAI.
* **Code:**

using Microsoft.AspNetCore.Mvc;

using Microsoft.Azure.Storage;

using Microsoft.Azure.Storage.Blob;

using Newtonsoft.Json;

using System.Text;

using Xceed.Words.NET;

using DocumentFormat.OpenXml.Packaging;

using DocumentFormat.OpenXml.Spreadsheet;

using ExcelCell = DocumentFormat.OpenXml.Spreadsheet.Cell;

using DocumentFormat.OpenXml;

using DocumentFormat.OpenXml.Wordprocessing;

using Doc\_Search\_Bot;

using Python.Runtime;

using System.Diagnostics;

[Route("api/[controller]")]

[ApiController]

public class ChatController : ControllerBase

{

private readonly CloudBlobContainer \_blobContainer;

private readonly string \_openAiApiKey;

private readonly ILogger<ChatController> \_logger;

private readonly ExtractTextFromPdf \_pdfExtractor;

public ChatController(ILogger<ChatController> logger, ExtractTextFromPdf pdfExtractor)

{

\_logger = logger;

\_pdfExtractor = pdfExtractor;

// Azure Storage Account Connection String

var connectionString = "DefaultEndpointsProtocol=https;AccountName=genaipocvj;AccountKey=2cVv6ef9sJwEs/wrf9gHARm1jvPLdOTQiJkmP5wzr30SWYHj9duvE5o7CRM9N9As1nBgVBENqWoa+AStkueI5A==;EndpointSuffix=core.windows.net";

// OpenAI API Key

\_openAiApiKey = "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"; // Replace with your actual OpenAI API key

var storageAccount = CloudStorageAccount.Parse(connectionString);

var blobClient = storageAccount.CreateCloudBlobClient();

\_blobContainer = blobClient.GetContainerReference("genaipocvj");

}

[HttpPost("ask")]

public async Task<ActionResult<string>> AskQuestion([FromBody] string userQuery)

{

try

{

\_logger.LogInformation($"Received user query: {userQuery}");

string content = await SearchBlobStorage();

if (!string.IsNullOrEmpty(content))

{

string response = await GenerateOpenAIResponse(userQuery, content);

\_logger.LogInformation($"OpenAI response: {response}");

// Extracting only the assistant's content

string assistantContent = ExtractContentFromResponse(response);

\_logger.LogInformation($"Assistant content: {assistantContent}");

Console.WriteLine( assistantContent );

// Return the content in the response

return Ok(new { Content = assistantContent });

}

\_logger.LogWarning("Content not found");

return Ok("Content not found");

}

catch (Exception ex)

{

Console.WriteLine($"Internal server error: {ex.Message}");

Console.WriteLine($"Error processing PDF file: {ex.Message}");

Console.WriteLine($"Stack Trace: {ex.StackTrace}");

return StatusCode(500, $"Internal server error: {ex.Message}");

Process process = Process.GetCurrentProcess();

Console.WriteLine($"Memory Usage: {process.WorkingSet64} bytes");

}

}

private string ExtractContentFromResponse(string jsonResponse)

{

dynamic responseObject = JsonConvert.DeserializeObject(jsonResponse);

string assistantContent = responseObject["choices"][0]["message"]["content"];

return assistantContent;

}

private async Task<string> SearchBlobStorage()

{

var blobItems = await ListBlobItemsAsync();

StringBuilder combinedContent = new StringBuilder();

foreach (var blobItem in blobItems)

{

if (blobItem is CloudBlockBlob blockBlob)

{

var content = await DownloadBlobContentAsync(blockBlob);

// Log the content for debugging

\_logger.LogInformation($"Content for {blockBlob.Name}: {content}");

// Append the content to the combinedContent

combinedContent.Append(content);

// Add two lines of hyphens as separation

combinedContent.AppendLine();

combinedContent.AppendLine("----------------------------------------");

combinedContent.AppendLine();

}

}

string mergedContent = combinedContent.ToString();

if (!string.IsNullOrEmpty(mergedContent))

{

\_logger.LogInformation("Merged content found in storage");

\_logger.LogInformation(mergedContent);

return mergedContent;

}

\_logger.LogInformation("No content found in storage");

return null; // Content not found

}

private async Task<IEnumerable<IListBlobItem>> ListBlobItemsAsync()

{

var blobItems = new List<IListBlobItem>();

BlobContinuationToken continuationToken = null;

do

{

var results = await \_blobContainer.ListBlobsSegmentedAsync(continuationToken);

continuationToken = results.ContinuationToken;

blobItems.AddRange(results.Results);

} while (continuationToken != null);

return blobItems;

}

private async Task<string> DownloadBlobContentAsync(CloudBlockBlob blob)

{

using (var memoryStream = new MemoryStream())

{

await blob.DownloadToStreamAsync(memoryStream);

//\*\*\*\*

string extension = Path.GetExtension(blob.Name).ToLower();

System.Console.WriteLine(blob.Name + " " + extension);

string text;

if (extension == ".docx" || extension == ".doc")

{

text = ExtractTextFromDocx(memoryStream);

}

else if (extension == ".pptx" || extension == ".ppt")

{

text = ConvertPptxToDocxAndExtractText(memoryStream);

}

else if (extension == ".xls" || extension == ".xlsx")

{

text = ExtractTextFromXlsx(memoryStream);

}

else if (extension == ".pdf")

{

text=\_pdfExtractor.Extract\_Text\_FromPdf(memoryStream);

}

else

{

text = "Unsupported file type";

}

memoryStream.Seek(0, SeekOrigin.Begin);

return text;

}

}

private string ExtractTextFromDocx(Stream stream)

{

// Logic to extract text from .docx file using external library (e.g., Aspose.Words)

using (var document = DocX.Load(stream))

{

return document.Text;

}

}

private string ConvertPptxToDocxAndExtractText(Stream pptxStream)

{

using (MemoryStream docxStream = new MemoryStream())

{

// Create a Word document

using (WordprocessingDocument wordDocument = WordprocessingDocument.Create(docxStream, WordprocessingDocumentType.Document))

{

MainDocumentPart mainPart = wordDocument.AddMainDocumentPart();

mainPart.Document = new DocumentFormat.OpenXml.Wordprocessing.Document();

mainPart.Document.Body = new Body(); // Initialize the Body element

// Iterate through slides in the PowerPoint presentation

using (PresentationDocument presentationDocument = PresentationDocument.Open(pptxStream, false))

{

foreach (SlidePart slidePart in presentationDocument.PresentationPart.SlideParts)

{

// Create a new paragraph for each slide

DocumentFormat.OpenXml.Wordprocessing.Paragraph slideParagraph = new DocumentFormat.OpenXml.Wordprocessing.Paragraph();

// Extract text from each text element in the slide

foreach (var textElement in slidePart.Slide.Descendants<DocumentFormat.OpenXml.Drawing.Paragraph>())

{

// Append the text to the slide's paragraph

slideParagraph.Append(textElement.CloneNode(true));

}

// Check if mainPart.Document.Body is null

if (mainPart.Document.Body != null)

{

// Add the slide's paragraph to the Word document

mainPart.Document.Body.Append(slideParagraph.CloneNode(true));

}

else

{

// Handle the case where Body is null (optional based on your application logic)

Console.WriteLine("Error: Body is null");

}

}

}

}

// Extract text from the Word document

return ExtractTextFromWord(docxStream);

}

}

private string ExtractTextFromWord(Stream wordStream)

{

StringBuilder text = new StringBuilder();

using (WordprocessingDocument wordDocument = WordprocessingDocument.Open(wordStream, false))

{

// Iterate through paragraphs in the Word document

foreach (var paragraph in wordDocument.MainDocumentPart.Document.Descendants<DocumentFormat.OpenXml.Wordprocessing.Paragraph>())

{

text.AppendLine(paragraph.InnerText);

}

}

return text.ToString();

}

private string ExtractTextFromXlsx(Stream stream)

{

var text = new StringBuilder();

using (var spreadsheetDocument = SpreadsheetDocument.Open(stream, false))

{

var workbookPart = spreadsheetDocument.WorkbookPart;

var sharedStringTablePart = workbookPart.SharedStringTablePart;

foreach (var sheet in workbookPart.Workbook.Descendants<Sheet>())

{

var worksheetPart = workbookPart.GetPartById(sheet.Id) as WorksheetPart;

if (worksheetPart != null)

{

var sharedStringTable = sharedStringTablePart.SharedStringTable;

foreach (var cell in worksheetPart.Worksheet.Descendants<ExcelCell>())

{

if (cell.DataType != null && cell.DataType == CellValues.SharedString)

{

int sharedStringIndex = int.Parse(cell.InnerText);

text.Append(sharedStringTable.ElementAt(sharedStringIndex).InnerText).Append('\t');

}

else

{

text.Append(cell.InnerText).Append('\t');

}

}

text.AppendLine();

}

}

}

return text.ToString();

}

private async Task<string> GenerateOpenAIResponse(string userQuery, string content1)

{

\_logger.LogInformation(content1);

using (HttpClient httpClient = new HttpClient())

{

httpClient.DefaultRequestHeaders.Add("Authorization", $"Bearer {\_openAiApiKey}");

var messages = new List<object>

{

new { role = "system", content = "You are a helpful assistant that provides information based on the given content ,Note1:First Understand the user content and read your content word by word then give the response,Note2:Some data in the content will not be in structured you want read that type of content also, Avoid blank space in the content. If the information is not found, respond with 'No Content Found'"},

new { role = "user", content = $"{userQuery}, Search in the Content and give the response" },

new { role = "assistant", content=content1 }

};

var requestBody = new { model = "gpt-3.5-turbo", messages, max\_tokens = 4096 }; // Adjust max\_tokens as needed

var response = await httpClient.PostAsJsonAsync("https://api.openai.com/v1/chat/completions", requestBody);

// Log response status code and content for troubleshooting

\_logger.LogInformation($"Response Status Code: {response.StatusCode}");

var result = await response.Content.ReadAsStringAsync();

\_logger.LogInformation($"Response Content: {result}");

if (!response.IsSuccessStatusCode)

{

// Log additional information for troubleshooting

\_logger.LogError($"Request URL: {httpClient.BaseAddress}{httpClient.DefaultRequestHeaders}");

\_logger.LogError($"Request Body: {requestBody}");

}

response.EnsureSuccessStatusCode();

return result.Trim();

}

}

}

**4. Third-Party Libraries**

* **Aspose.Words:** Used for extracting text from .docx files.
* **DocX:** Utilized for processing .docx files.
* **Newtonsoft.Json:** Employed for JSON serialization and deserialization.
* **Microsoft.Extensions.Logging:** Integrated for logging functionality.
* **Xceed.Words.NET:** Utilized for working with Word documents.
* **Microsoft.Azure.Storage:** Utilized for interacting with Azure Storage services.
* **DocumentFormat.OpenXml:** Utilized for processing XML-based document formats such as PowerPoint (.pptx) and Excel (.xlsx).

**5. Azure Blob Storage Configuration with Azurite**

* **Azurite Installation:**
  + Azurite is a lightweight Azure Storage emulator that allows you to develop and test Azure services locally. Follow the installation instructions for your operating system from the [Azurite GitHub repository](https://github.com/Azure/Azurite).
* **Configuration:**
  + Set the **ConnectionString and Container name**  in the **DocumentProcessingController** to point to your Azurite instance.
  + Ensure Azurite is running and accessible from your application.

**Integration**

The Document Search Bot API components work together to provide a seamless user experience. The Login Controller handles authentication, while the Document Processing Controller manages document-related tasks. The Chat Controller leverages extracted content from documents to generate responses to user queries.

**Deployment**

The Document Search Bot API can be deployed to a hosting service capable of running ASP.NET Core applications. Ensure proper configuration of Azure Blob Storage and third-party libraries for seamless functionality.

**Conclusion**

The Document Search Bot API offers robust functionality for user authentication, document processing, and chat interactions. With its modular architecture, integration capabilities, and support for third-party libraries, it provides a versatile solution for various applications.