Overview:

Create a Note Keeper REST Interface that supports, creation and management of notes using static in memory storage.

You will create .NET 9 based ASP.NET Core Web API project that will use a static in memory data structure to store the data needed for the notes. Your application will utilize Azure AI Foundry with the OpenAI model **gpt-4o-mini** to generate a set of tags related to the detail of the note. You will deploy your Web API application to a Microsoft Azure App Service hosted on a Linux Azure App Service Plan that uses the **Basic B1 Tier**.

Be sure to read the entire homework assignment instructions and the **Implementation Notes** section as it has important guidelines and restrictions before starting the design and implementation of your assignment.

Goals

- Setup and configure
  - Your development environment
    - Visual Studio 2022, VS Code or JetBrains Rider.
- Setup and configure
  - Your Azure account and subscription
- Familiarize yourself with
  - Your development environment
  - Azure Dashboard
  - Azure App Services
  - Azure App Service Plans
  - Azure AI Foundry
- Gain experience with:
  - .NET 9 ASP.NET Core Web API Projects
  - Building REST interfaces that run in Azure App Services
  - Integrating AI into your application
  - Debugging
    - Web API App Services
    - Web API App Services from the Azure Dashboard Logging

Required Nuget Packages for this assignment: (Versions can be newer)

```
  <ItemGroup>
    <PackageReference Include="Microsoft.AspNetCore.OpenApi" Version="9.0.1" />
    <PackageReference Include="Swashbuckle.AspNetCore" Version="7.2.0" />
    <PackageReference Include="Azure.AI.OpenAI" Version="2.1.0" />
    <PackageReference Include="Microsoft.Extensions.AI" Version="9.1.0-preview.1.25064.3" />
    <PackageReference Include="Microsoft.Extensions.AI.Ollama" Version="9.1.0-preview.1.25064.3" />
    <PackageReference Include="Microsoft.Extensions.AI.OpenAI" Version="9.1.0-preview.1.25064.3" />
    <PackageReference Include="Microsoft.Extensions.Hosting" Version="9.0.1" />
    <PackageReference Include="NJsonSchema" Version="11.1.0" />
  </ItemGroup>
```

Table of contents

Solution, project naming, and submission information

You shall create one Solution containing one .NET 9 Project. The solution shall be called **HW1NoteKeeperSolution**

Note:

- Double check your **appsettings json** files to be sure you have cleansed them of credentials prior to submission.
- Do not include files under the **bin, obj,** and **Properties\PublishProfiles** directory, or locally published files used for deployment.

The solution shall contain one project. This project shall be called **HW1NoteKeeper**

As stated in the syllabus: The completed assignment shall be placed in a single zip file that preserves the directory structure. and must be submitted to the assignment's associated drop box on the course canvas website. Submission by any other means **will not be accepted**. The zip file shall contain:

a) All assets such as configuration files, images, initialization scripts and resources that are part of the application. The TA must be able to build and deploy your homework assignment to Azure and experience the full functionality of your application.
b) A text file named **ProjectNotes.txt** that includes:
    1) The homework assignment title and number
    2) Your name and email address
    3) **Attribution regarding the use of AI in the development of your assignment**
    4) Any notes for the TA that may be needed for the TA to install, set up, login to and operate your homework assignment.
    5) Your **Azure AI Foundry Hub and Project name**
    6) Any custom Azure resource abbreviations you used
    7) The URL to Azure App Service Website
    8) The target **Uri Azure OpenAI Service**
        1. See 5. Implement Tag generation using the gpt the 4o mini model
    9) Do not include any credentials to your Azure Resources, or any compiled output.

If you have any questions regarding these guidelines, please ask your grading TA or the instructor via Microsoft Teams in the associated assignment channel.

1. Implement notes resource

1.1 API Operation – Create

Creates a new note entry

HTTP
POST https://[appservicename].azurewebsites.net/notes

URI Parameters

None

Request Body

```
{
  "summary": "<a summary of the note>",
  "details": "<the details of the note>"
}
```

Name	JSON Type	.NET Type	Min Length	Max Length	Description & Notes
summary	string	string	1	60	A field where the user can enter the summary of their note **Constraints**: Required, Not Null, Not Empty, Not All Spaces/Whitespace
details	string	string	1	1024	A field where the user can enter the details of their note **Constraints**: Required, Not Null, Not Empty, Not All Spaces/Whitespace

Notes:

- See the full data structure that will be returned in the sections Response Body: below and the full description of the response data structure in the section 1.4 API Operation – Retrieve by Id
- The **CreatedDateUtc** shall be updated with the current Utc DateTime.
- The **ModifiedDateUtc** shall be null.
- The **Tags** array shall contain the **Tags** generated by the **gpt-4o-mini** model you deployed in Azure AI Foundry for the **details** input string
    - **See 5. Implement Tag generation using the gpt the 4o mini model**

Responses

201 (Created)

Returned if the note was created successfully

For Example:

Given a note with a summary field containing "**Azure tips and tricks**", and a details field containing "**You can create custom dashboards to organize your resources**" was created with a noteId of **837df4ee-74d3-447f-9c0c-61491e887fa5** the key elements of the response shall look like this: (assuming the app service name is **mynotekeeper**).

Headers:

Standard Headers + Location header:

```
content-type: application/json; charset=utf-8
date: Fri, 02 Feb 2024 22:01:53 GMT
Location: http://mynotekeeper.azurewebsites.net/notes/837df4ee-74d3-447f-9c0c-61491e887fa5
server: Kestrel
```

Response Body:

400 (Bad Request)

Returned if the note has one or more invalid parameters.

```
{
  "noteId":"837df4ee-74d3-447f-9c0c-61491e887fa5",
  "summary":"Azure tips and tricks",
  "details":"You can create custom dashboards to organize your resources",
  "createdDateUtc":"2024-01-25T03:42:29.3402048Z",
  "modifiedDateUtc":null,
  "tags": [
    "Dashboards",
    "Customization",
    "Organization",
    "Resources"
  ]
 }
```

Headers:

None, beyond standard

Response Body:

No custom response body is required, you can use the default response body returned by ASP.NET Core.

1.2 API Operation – Update

Updates a note with the payload provided

HTTP
PATCH https://[appservicename].azurewebsites.net/notes/**<noteId>**

URI Parameters

Name	JSON Type	.NET Type	Min Length	Max Length	Description & Notes
noteId	string	string	Not Enforced	Not Enforced	The note id of the note **Constraints:** Required, Must Exist

Request Body

```
{
  "summary": "<a summary of the note>",
  "details": "<the details of the note>"
}
```

Name	JSON Type	.NET Type	Min Length	Max Length	Description & Notes
summary	string	string	1 Null	60	A field where the user can update the summary of their note. **Constraints**: Optional, however if provided must not be null or empty or all white space/spaces to cause an update to the summary field. If the summary field is updated the **ModifiedDateUtc** shall be updated with the current Utc date time.
details	string	string	1 Null	1024	A field where the user can update the details of their note. **Constraints**: Optional, however if provided must not be null or empty or all white space/spaces to cause an update to the details field. If the details field is updated the **ModifiedDateUtc** shall be updated with the current Utc date time.

Notes:

- The **CreatedDateUtc** shall not be altered.
- If the Summary field is empty or null, make no changes to the Summary field and do not update the **ModifiedDateUtc** unless the Detail field is updated.
- If the Detail field is empty or null, make no changes to the Detail field and do not update the **ModifiedDateUtc** unless the Summary field is updated.
- Empty means a string that has nothing in it or has 1 or more spaces.
- The **Summary** Field and or **Detail** Field being empty, or null does not result in a 400 bad request.
- The **Tags** array shall contain the Tags generated by the gpt-4o-mini model for updates made to **details** input string. If no change was made to the **details** string, then the tags shall remain the same.
    - **See 5. Implement Tag generation using the gpt the 4o mini model**
        - Hint AI may not generate the same tags each time it's called.

Responses

204 (No Content)

Returned if the note was updated successfully

Headers:

None, beyond standard

Response Body:

None

400 (Bad Request)

Returned if the note has one or more invalid parameters.

Headers:

None, beyond standard

Response Body:

No custom response body is required, you can use the default response body returned by ASP.NET Core.

404 (Not Found)

Returned if the note could not be found.

Headers:

None, beyond standard

Response Body:

No custom response body is required, you can use the default response body returned by ASP.NET Core.

1.3 API Operation – Delete

Deletes a note having the specified note id

HTTP
DELETE https://[appservicename].azurewebsites.net/notes/**<noteId>**

URI Parameters

Name	JSON Type	.NET Type	Min Length	Max Length	Description & Notes
noteId	string	string	Not enforced	Not enforced	The noteId of the note **Constraints**: Required, Must Exist.

Request Body

None

Responses

204 (No Content)

Returned if the note was deleted successfully.

Headers:

None, beyond standard

Response Body:

None

404 (Not Found)

Returned if the note could not be found.

Headers:

None, beyond standard

Response Body:

No custom response body is required, you can use the default response body returned by ASP.NET Core.

1.4 API Operation – Retrieve by Id

Retrieves a note having the specified noteId.

HTTP
GET https://[appservicename].azurewebsites.net/notes/**<noteId>**

URI Parameters

Name	JSON Type	.NET Type	Min Length	Max Length	Description & Notes
noteId	string	string	Not Enforced	Not Enforced	The noteId of the note **Constraints:** Required, Must Exist

Request Body

None

Responses

200 (OK)

Returned if the note was retrieved successfully

Headers:

None, beyond standard

Response Body:

The details of the entity retrieved

```
{
  "noteId":"<the id of the note>",
  "summary":"<the note summary entered by the user>",
  "details":"<the note details entered by the user>",
  "createdDateUtc":"<the date & time in UTC the note was created>",
  "modifiedDateUtc":"<the date & time in UTC the note was modified or null if never
                      modified>",
"tags": [
    "tag 1 generated by AI",
    "tag 2 generated by AI ",
    "tag 3 generated by AI ",
    "etc.."
  ]
}
```

Name	JSON Type	.NET Type	Description & Notes
noteId	string	string	The note id of the note
summary	string	string	The note summary entered by the user
details	string	string	The note details entered by the user
createdDateUtc	string	DateTime	The date and time the note was created Use standard ISO 8601 format with the time component: "2024-01-25T03:44:08.077Z" See: https://en.wikipedia.org/wiki/ISO_8601
modifiedDateUtc	String	DateTime	The date and time the note was modified Use standard ISO 8601 format with the time component: "2024-01-26T04:42:41.012Z" See: https://en.wikipedia.org/wiki/ISO_8601
tags	string[]	List<string>	The tags generated by the **gpt-4o-mini** model using the content of the **details** entered by the user

404 (Not Found)

Returned if the note could not be found.

Headers:

None, beyond standard.

Response Body:

No custom response body is required, you can use the default response body returned by ASP.NET Core.

1.5 API Operation – Retrieve all

Retrieves all notes

HTTP
GET https://[appservicename].azurewebsites.net/notes

URI Parameters

None

Request Body

None

Responses

200 (OK)

Returned regardless of whether there are any notes to return.

Headers:

None, beyond standard.

Response Body:

```
[
  {
    "noteId": "837df4ee-74d3-447f-9c0c-61491e887fa5",
    "summary": "Azure tips and tricks",
    "details": "You can create custom dashboards to organize your resources",
    "createdDateUtc": "2024-01-25T03:42:29.3402048Z",
    "modifiedDateUtc": null,
    "tags": [
       "Dashboards",
       "Customization",
       "Organization",
      "Resources"
    ]
   },
  {
    "noteId": "c5b08f40-4fec-4ccd-bc17-bd40e8ba112a",
    "summary": "Visual Studio Tips and Tricks",
    "details": "Past special converts Json to C# classes",
    "createdDateUtc": "2024-01-25T04:18:15.6092849Z",
    "modifiedDateUtc": "2024-01-26T03:22:13.4024167Z",
  "tags": [
    "Json",
    "C#",
    "Conversion",
    "Programming",
    "Code",
    "Data"
  ]
  }
]
```

The details of all the notes available to be returned, see 1.4 API Operation – Retrieve by Id for field details.

2. Add swagger UI with documentation to your REST interface

a) Ensure the swagger UI is presented when the user navigates to your applications base URL.
   a. When running in Azure the base url is:
      `https://[appservicename].azurewebsites.net/index.html`
      Where [appservicename] is your app services name
   b. When running locally from Visual Studio the base url is:
      `https://localhost:[portnumber]/index.html`
      Where [portnumber] is the port number visual studio assigned to you .NET Core Web API app
   c. Do not require or redirect to the /swagger sub-resource to display the UI
b) Ensure that all REST operations are present and can be exercised from the swagger user interface
c) Ensure that all REST operations have a description
d) Ensure that all input and output payload fields have a description.
   a. Except for the "**`ProblemDetails`**" payload that is created automatically by ASP.NET Core you don't have to add documentation to it.
e) Ensure that all Http Status codes your application can return for each REST operation are present and described with the corresponding http status code and text. 200 Success, 201 Created, 204 Success, 400 Bad Request, 404 Not Found and so forth.
f) Ensure that the response is of type application/JSON

3. Create an Azure App Service Plan

Create a Linux Azure App Service plan using the **Basic B1 Tier** to host your app service on. Remember to follow recommended abbreviations for Azure resource types. See Implementation notes:

4. Create an Azure App Service

Create an Azure App Service and deploy your note keeper Web API implementation to it. Be sure to provide the URL to your REST interface endpoint in your Project Notes.TXT file. Remember to follow recommended abbreviations for Azure resource types. See Implementation notes:

5. Implement Tag generation using the gpt the 4o mini model

5.a Setup Azure AI Foundry Hub & Project

5.a.1 Set up the Azure AI Foundry Hub and Project

- Create an Azure AI Foundry Hub and a new project. Ensure the project meets the requirements for deploying AI models.

5.a.2 Create the Required Resources

- Provision the necessary **Azure Storage Account** following the recommended resource naming conventions and abbreviations.

5.a.3 Deploy the GPT-4o-mini Model

- Use the Azure AI Foundry interface to deploy the **gpt-4o-mini** model to your project.

5.a.4 Document Key Information

- In your project notes.txt file and include the following:
  - The **EndpointUri** for your **Azure AI Foundry OpenAI Service Uri.**
  - The **ApiKey** for accessing the project.

5.a.5 Secure Configuration settings in your App Service

- Navigate to the Azure Portal and access your **App Service** settings. Add the following application settings:
  - **EndpointUri**: Specify the **Azure AI Foundry OpenAI Service Uri**
  - **ApiKey**: Set the API key for accessing the project.
- **Important**: Do not hard code the API key into your code files. Instead:
  - Store the API key securely in the **secrets.json** file.

5.a.6 Follow Azure Resource Naming Best Practices

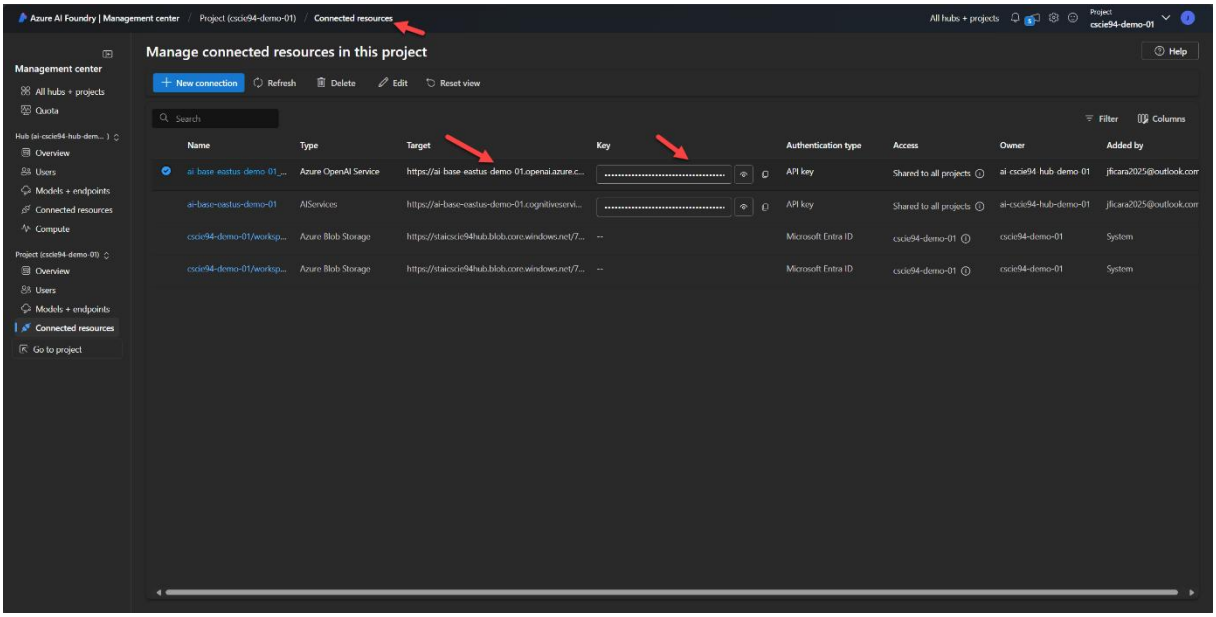
- See Implementation notes:

5.b Generate the Tags from the details string provided for the note

5.b.1 When adding a note always generate the Tags using the details string

5.b.2 When updating a note only generate the Tags if the details string has changed

5.b.3 You shall use either the system prompt approach or the prompt enhancement approach.



Implementation notes:

1. **Include Attribution regarding the use of AI in the development of your assignment in your project notes.txt file**
    a. **You may not use AI to fully implement your assignment or extra credit**
        i. **Example: You can't use it to generate all of your unit tests**
    b. **You may use it as an aid in the development of the assignment.**
        i. **Implementation of small snippets of code or explaining how code works**
2. You may not use / employ
    a. The "any samples" from the Azure Dashboard to create this solution
3. You can use the Visual Studio ASP.NET Core Web API Application Template
    a. You must delete all non-relevant items as described in the lecture
4. All exceptions within your code must be handled, the service must not return any exception information to the caller raised inside of your code. Exceptions raised before your code is reached don't need to be captured in this assignment.
5. Use built in exceptions to handle error exceptional cases wherever possible.
6. When creating azure resources, including the resource group your Azure resources will reside in, be sure to use the recommended abbreviations for Azure resource types. See: Recommended abbreviations for Azure resource types - Cloud Adoption Framework | Microsoft Docs
    a. Note not all services have abbreviations, where none are specified choose your own but be consistent.
    b. **Document** any **custom abbreviations** you used in your **project notes.txt** file
7. Map App Service REST errors to standard HTTP Error codes as defined in the requirements
    a. If a situation is not specified in the requirements pick a standard HTTP status code and document under what conditions, it is returned in your projectnotes.txt file

Other Notes:

- Up to 15 Points shall be deducted for lack of comments in code.
- All methods, classes etc. must be documented
    a. C# with XML comments.
- Your project must build with no errors or warnings

Extra Credit:

To qualify for extra credit, you must first implement all core requirements of the assignment. Only the extra credit options specified are eligible for additional points, aside from points awarded for early submission. To earn extra credit, you must completely implement an extra credit option. You are free to choose either a single extra credit option or multiple options from those listed.

Students registered for graduate credit are required to complete one extra credit option. The selected option will be included in your base grade calculation but will not raise your overall score above 100%. Failure to complete an extra credit option will result in a deduction equivalent to the points assigned to the lowest-value extra credit option. For specific point values and details, refer to the grading rubric for this assignment.

Extra Credit Option 1: Build a client-side application.

That allows management of the note and associated note items. Management means allowing the create, update, and delete operations for notes.

The details of the user interface are up to you so have fun with it. Just be sure it's intuitive and does not require the TA to download or install any additional software such as frameworks or other tools to build and run your user interface. Your client application must be run from the TA's computer and not deployed to any remote / cloud location.

You may build:

1. Console app
2. UWP app (Windows Store App)
3. MAUI app
4. WPF app
5. JavaScript app
6. ASP.NET MVC App
7. ASP.NET WebForms App
8. Windows Forms App
9. Blazor App (Client or Server)

Extra Credit Option 2: Create xUnit Tests to validate ALL the REST operations provided in your API

Ensure that the xUnit project is included in your solution with your ASP.NET Core Web API project.

Ensure that the GETs, PATCH, POST and DELETE operations are all fully covered in your tests.

Generate the client SDK for your REST interface and include it as part of your solution

Use Visual Studio 2022's built in support via the Manage Connected Services, Service Reference, Open API functionality. Utilize this SDK to exercise the REST interface using functional tests, both Positive and Negative tests.

Configure your tests to execute against the instance you have deployed in Azure.

Positive tests:

Test that verifies the success conditions. In positive tests, be sure to validate the proper http status code and response payload.

Validation includes ensuring that the response payload, as mentioned above, contain the values that are expected.

Example:

Creation of a test that successfully retrieves a note by id and validates the http status code and the note summary and details match the summary and details expected as well as the expected **CreatedDateUtc** and **ModifiedDateUtc**. When validating the Created and Modified date's the exact time is not necessary to validate just that the Created Date and Modified Date are present when they should be and have a correct relationship to each other. The Modified Date, if the note has been modified, must always be greater in time than the Created Date example.

Negative tests:

Tests that verify the failure conditions. In negative tests, be sure to validate the proper http status code.

Example:

A test that fails because the note summary for specified for a note was null when trying to create a note. Here the test would validate that the http status code is a 400 (Bad Request)