Overview:

Update your Note Keeper REST API application to support creation and management of notes using an Azure SQL Database.

You will create .NET 9 based ASP.NET Core Web API project with Entity Framework Core 9 that will use an Azure SQL Database to persist data needed for the notes. You will deploy your application to a **NEW Linux Azure App Service** hosted on your **EXISTING 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.

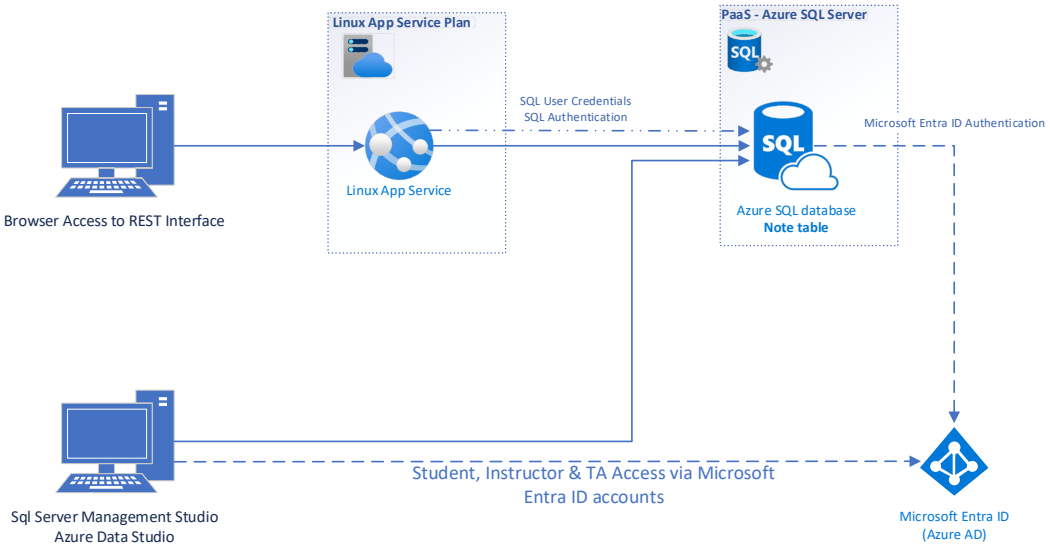
Goals

- Setup and configure.
  - Azure SQL Server and Database
  - Custom Settings
- Familiarize yourself with
  - Azure SQL Server & Database
  - Entity Framework Core
  - Application Insights (Extra Credit)
  - Managed Identities (Extra Credit)
- Gain experience with:
  - Development environment
  - .NET 9 ASP.NET Core Web API Projects
  - Building REST interfaces that run in Azure App Services
  - Azure App Services
  - Azure App Service Plans
  - Debugging
    - Web API App Services from Visual Studio / Visual Studio Code
    - Web API App Services from the Azure Dashboard Logging

Table of contents

Core Requirements - Infrastructure Architecture

Assignment 02
Infrastructure Architecture



CSCI E94 Fundamentals of Cloud Computing and Open AI with Microsoft Azure
Copyright 2024-2025

Instructor: Joseph Ficara
2025-02-13 v5.00

Solution, project naming, and submission information

You shall create one Solution containing one Visual Studio Project. The solution shall be called **HW2NoteKeeperSolution.** The solution shall contain the core project that shall be called **HW2NoteKeeper.** If you implement one or more extra credit options those projects shall be added to the **HW2NoteKeeperSolution**.

Note:

- Do not include any Azure credential information in your submission unless specifically requested to do so. 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** directory and the **Properties\PublishProfiles** directory.
- All settings shall be configured in Azure under the Environment Application Settings area
    - Remember linux requires a double underscore _ between the section and field name
        - Ex. MySetting__Url, where MySettings is the section name and Url is the field name.
- If you implement extra credit options add the associated projects to the **HW2NoteKeeperSolution**

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) The name of the Azure SQL **Server** you created.
    7) The name of the Azure SQL **Database** you created.
    8) The **URL** to the **Azure SQL Server** with the **Database** name included
    9) Any custom Azure resource abbreviations you used
        1. You can the abbreviation recommendations for Azure resources here: Abbreviation recommendations for Azure resources - Cloud Adoption Framework | Microsoft Learn
    10) The modified **AddUsersToYourAzureSQLServerInstance.sql** file with the teaching assistants and instructor's user principal names updated accordingly.
    11) The **URL** to **Azure App Service Website**.
    12) The target **Uri Azure OpenAI Service**
    13) Don't include the compiled output
        1. **Do not include the publish, bin or obj directory**

Important! Azure SQL Server User Notes

1. Add the user, which you log into Visual Studio with, to your Azure SQL Server database. See the script **AddUsersToYourAzureSQLServerInstance.sql** for an example on how to add users.
2. Be sure to perform an **az login** with the user used in step 1.
3. Remember to verify your local development's access to the database using the user used in step 1.
4. Modify the file **AddUsersToYourAzureSQLServerInstance.sql** to include all the teaching assistants and instructor to your Azure SQL Server. You do not need to include your user id in this file.
5. These users should already be Global Admin's and have the subscription owner role in your tenant. You will still need to add them to your Azure SQL Server, see **AddUsersToYourAzureSQLServerInstance.sql** for instructions and the necessary script to do so.
6. You need to use the User Principal from the over page for each user in Entra Id

    - Jonathan Franck
        - jon001sox@gmail.com#EXT#@<your domain>.onmicrosoft.com
    - Max Eringros
        - meringros@gmail.com#EXT#@<your domain>.onmicrosoft.com
    - Joe Ficara
        - jficarainstructor2025@outlook.com#EXT#@<your domain>.onmicrosoft.com
    - Jessica Pratt
        - jphvdta@outlook.com#EXT#@<your domain>.onmicrosoft.com
    - Nancy Forero
        - contactnaneccles@gmail.com#EXT#@<your domain>.onmicrosoft.com
    - Gustavo Varo
        - gustavo.varo@gmail.com#EXT#@<your domain>.onmicrosoft.com

If you have any questions regarding these guidelines, please ask a TA or the instructor.

1. Create a Microsoft Entra ID User to be the admin of your database

1.1. You shall create a Microsoft Entra ID user called dbadmin

1.2. You shall login into the Azure Portal using the Microsoft Entra ID user you created in step 1.1

1.3. You shall reset your password and setup MFA support

1.4. The Microsoft Entra ID user is required to run the script AddUsersToYourAzureSQLServerInstance.sql

2. Create your Azure SQL Server and Database

2.1. Your database server shall be created using the Azure portal workflow that provides the creation of database and server and use the azure standard naming prefix.

2.2. Your database shall be created using the workflow that provides the creation of database and server and use the azure standard naming prefix.

2.3. Your database and server shall be put in a new resource group called **rg-sqldata**

2.4 The database server shall **use both SQL Authentication** and **Microsoft Entra ID Authentication**

2.5 You shall assign the **dbadmin** user as the **Microsoft Entra ID Admin** for your database server.

2.6 You shall create a **SQL Server admin login and password for your database server**. This is **SQL Server authentication** not **Microsoft Entra ID** performed in step 1

2.7 The database shall NOT use the SQL elastic pool.

2.8 The database shall use **Locally redundant backup storage.**

2.9 The database shall use **DTU-based purchasing model.**

2.10 The database shall use **the Basic Tier**

2.11. The database shall have **a max size of 2 GB.**

2.12. The database shall have **5 DTUs.**

2.13. The database **ESTIMATED COST / MONTH shall be 4.90 USD**

2.14. The rest of the settings shall be the default provided by the portal wizard.

2.15. You shall include a screen capture the "Review + Create" screen including all the details you used in the creation of your database and server. The screen capture file shall be called **DBAndServerCreationDetails.png.**

2.16 You shall include **DBAndServerCreationDetails.png** it in your **assignment zip file**.

2.17 Ensure you update the firewall rules using "**Add current client IP address**."

2.18 Ensure you update the firewall rules to **Allow Azure services** and resources to access this server.

3. Define the notekeeper database table using code first

3.1 The database table shall be called note (the table name is note, it's not notes on purpose)

3.2 The Note database table shall have the following columns.

Table Name: **Note**

Column Name	SQL Datatype	.NET Type	Min Length	Max Length	Description & Notes
id	Uniqueidentifier	Guid	NA	NA	This shall not be a SQL identity column and shall be the primary key of the table
summary	nvarchar(60)	string	1	60	A field where the user can enter the summary of their note **Constraints**: Required, Not Null, Not Empty
details	nvarchar(1024)	string	1	1024	A field where the user can enter the details of their note. **Constraints**: Required, Not Null, Not Empty
CreatedDateUtc	datetimeoffset(7)	DateTimeOffset	N/A	N/A	The time the note was created. Note: Use UtcDateTime to convert back to DateTime .NET Type
ModifiedDateUtc	datetimeoffset(7)	DateTimeOffset	N/A	N/A	Note: Use UtcDateTime to convert back to DateTime .NET Type

Note:

- DateTimeOffset stores the date and time along with the offset, you can read more about it here: DateTimeOffset Struct (System) | Microsoft Docs
- You can read more about datetimeoffset(7) here: datetimeoffset (Transact-SQL) - SQL Server | Microsoft Docs
- Entity Framework Core 9 will automatically create the correct SQL Datatypes for you when using the code first approach required for this assignment. You can read more about Entity Framework Core here: Overview of Entity Framework Core - EF Core | Microsoft Docs

3.3 The Tag database table shall have the following columns.

Table Name: **Tag**

Column Name	SQL Datatype	.NET Type	Min Length	Max Length	Description & Notes
id	Uniqueidentifier	Guid	NA	NA	This shall not be a SQL identity column and shall be the primary key of the table
NoteId	Uniqueidentifier	Guid	NA	NA	The ID of the note the tag is associated with
Name	The name of the tag	string	1	30	The AI generated Tag Name. If the name generated by AI is larger than 30 characters truncate it to 30 characters

To keep things simple we have a one to many relationship between the Note and the Tag, we are not normalizing Tags and not creating a many to many relatinship. Meaning if two notes have the same set of tags those tags will be duplicated in the database but have their own unique Id.

4. Implement notes resource

4.1 API Operation – Create

Creates a new note entry, and the associated **Tags** using AI, and store it in the Azure SQL Database **note** table. Support a default of **10** notes using the configurable **MaxNotes** setting, which can be overridden in the Azure Portal's application settings area. See: 6. Create a custom setting to limit the number of notes that can be added

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
details	string	string	1	1024	A field where the user can enter the details of their note. **Constraints**: Required, Not Null, Not Empty

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 4.4 API Operation – Retrieve by Id
- The **CreatedDateUtc** shall be updated with the current Utc DateTime.
- The **ModifiedDateUtc** shall be null.

Responses

201 (Created)

Returned if the note was created successfully.

For Example:

If the 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, 04 Feb 2022 22:01:53 GMT
Location: http://mynotekeeper.azurewebsites.net/notes/837df4ee-74d3-447f-9c0c-61491e887fa5
server: Kestrel
```

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"
  ]
 }
```

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.

403 (Forbidden)

Returned if the addition of the note will cause the number of notes stored to exceed the **MaxNotes** value that has been set.

There is no limit to the number of Tags that can be associated with a note.

Headers:

None, beyond standard

Response Body:

Return a **ProblemDetails** object that contains the following information:

Property	Value
Status	403
Title	Note limit reached
Detail	Note limit reached MaxNotes [<the current setting for MaxNotes]

Example:

```
{
  "title":"Note limit reached",
  "status":403,
  "detail":"Note limit reached MaxNotes: [10]
}
```

Note:

- The **ProblemDetails** class is defined by the Microsoft.AspNetCore.Mvc framework. You can read more about it here: ProblemDetails Class (Microsoft.AspNetCore.Mvc) | Microsoft Docs

4.2 API Operation – Update

Updates a note with the payload provided and stores it in the Azure SQL Database **note** table.

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 Empty 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 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 Empty 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 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 **Summary** Field and or **Detail** Field being empty, or null does not result in a 400 bad request.
- The **Tags** stored to the database shall contain only the new 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.

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.

404 (Not Found)

Returned if the note could not be found.

405 (Method not allowed)

Returned if the note Id is not provided

404 (Not Found)

Returned if the note could not be found.

For all error responses:

Headers:

None, beyond standard

Response Body:

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

4.3 API Operation – Delete

Deletes a note having the specified note id from the Azure SQL Database **note** table. Shall also delete all of the associated tags from the **tag** table.

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

400 (Bad Request)

Returned if the note has one or more invalid parameters.

404 (Not Found)

Returned if the note could not be found.

405 (Method not allowed)

Returned if the note Id is not provided

404 (Not Found)

Returned if the note could not be found.

For all error responses:

Headers:

None, beyond standard

Response Body:

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

4.4 API Operation – Retrieve by Id

Retrieves a note having the specified noteId from the Azure SQL Database **note** table along with all of the associated tags.

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: "2022-01-06T22:22:55.6348220"
modifiedDateUtc	String	DateTime	The date and time the note was modified Use standard ISO 8601 format with the time component: "2022-01-06T22:22:55.6348220"
tags	string[]	List<string>	The tags generated by the **gpt-4o-mini** model using the content of the **details** entered by the user

- Notes regarding createdDateUtc and modifiedDateUtc:
  - As long as the DateTime is used the built-in serialization should produce the correct date time format in the response. The round-trip ("O", "o") format specifier produces the correct output if needed
  - See also: https://en.wikipedia.org/wiki/ISO_8601

Headers:

None, beyond standard.

400 (Bad Request)

Returned if the note has one or more invalid parameters.

404 (Not Found)

Returned if the note could not be found.

For all error responses:

Headers:

None, beyond standard

Response Body:

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

4.5 API Operation – Retrieve all.

Retrieves all notes from the Azure SQL Database **note** table, with the ability to filter by the tag name the note is associated with if the tag name is provided

HTTP
GET https://[appservicename].azurewebsites.net/notes?tagName=<tag name>

URI Parameters

Name	JSON Type	.NET Type	Description & Notes
tagName	string	string	The name of the tag to filter on Constraints: Optional, if provided must be non-white space and the results are filtered using the tag name associated with the note and the value provided.

Request Body

None

Responses

200 (OK)

Returned regardless of whether there are any notes to return.

Headers:

None, beyond standard

Response Body:

The details of all the notes available to be returned, see 4.4 API Operation – Retrieve by Id for field details.


```
[
  {
    "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"
  ]
  }
]
```

4.6 API Operation – Retrieve all Tags.

Retrieves a **unique** list of all tags from the Azure SQL Database **tag** table.

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

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:

```
[
  {
    "name": "dairy"
  },
  {
    "name": "decorations"
  },
  {
    "name": "essentials"
  },
  {
    "name": "food"
  }
]
```

Name	JSON Type	.NET Type	Description & Notes
Name	string	string	The name of the tag **Note**: The list shall contain a unique list of tag names; no duplicate entries are permitted in the result.

5. Automatically seed the database tables with default data using your API App

Your API app shall automatically seed the note table with **4** note entries as defined below:

Id	Summary	Details	CreatedDateUtc	ModifiedDateUtc
Auto Assigned Value	Running grocery list	Milk Eggs Oranges	Initialized with current utc time at run	Null
Auto Assigned Value	Gift supplies notes	Tape & Wrapping Paper	Initialized with current utc time at run	Null
Auto Assigned Value	Valentine's Day gift ideas	Chocolate, Diamonds, New Car	Initialized with current utc time at run	Null
Auto Assigned Value	Azure tips	portal.azure.com is a quick way to get to the portal Remember double underscore for linux and colon for windows	Initialized with current utc time at run	Null

Additionally, the seeing code will use AI to generate the tags in the Tag table for each of the 4 notes added during the seeding process.

6. Create a custom setting to limit the number of notes that can be added

6.1 The custom setting shall be called NoteSettings.

6.2. The custom setting property shall be called MaxNotes and shall be type int.

6.3. The default value shall be 10.

6.4. You shall add the custom setting to your app service's configuration blade using the application settings area.

6.5. There is no limit to the number of tags that can be generated per note.

7. Ensure your swagger UI created in assignment 2 still operates as it did for assignment 1 with the new additions specified in this assignment

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 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. Exception the "**`ProblemDetails`**" payload 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

8. Use your existing Azure App Service Plan

Use the **EXISTING** **Linux Azure App Service plan** you created for Assignment 1 that utilizes the Basic B1 tier to host your app service on.

9. Create an Azure App Service

9.1 Create a NEW Linux 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.

9.2 Your App Service's application configuration shall be configured to use SQL authentication to connect to the database using the credentials you setup in step 2.6. You will configure the connection settings in the configuration area of your app service in the application settings area.

9.3 Do not save these credentials in any of your applicationsettings.json files

9.4 Remember to follow recommended abbreviations for Azure resource types. See Implementation notes:

Implementation notes:

1. You may not use / employ.
    a. Any samples from the Azure Dashboard to create this solution.
2. 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.
3. 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.
4. Use built in exceptions to handle error exceptional cases wherever possible.
5. 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
6. 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:

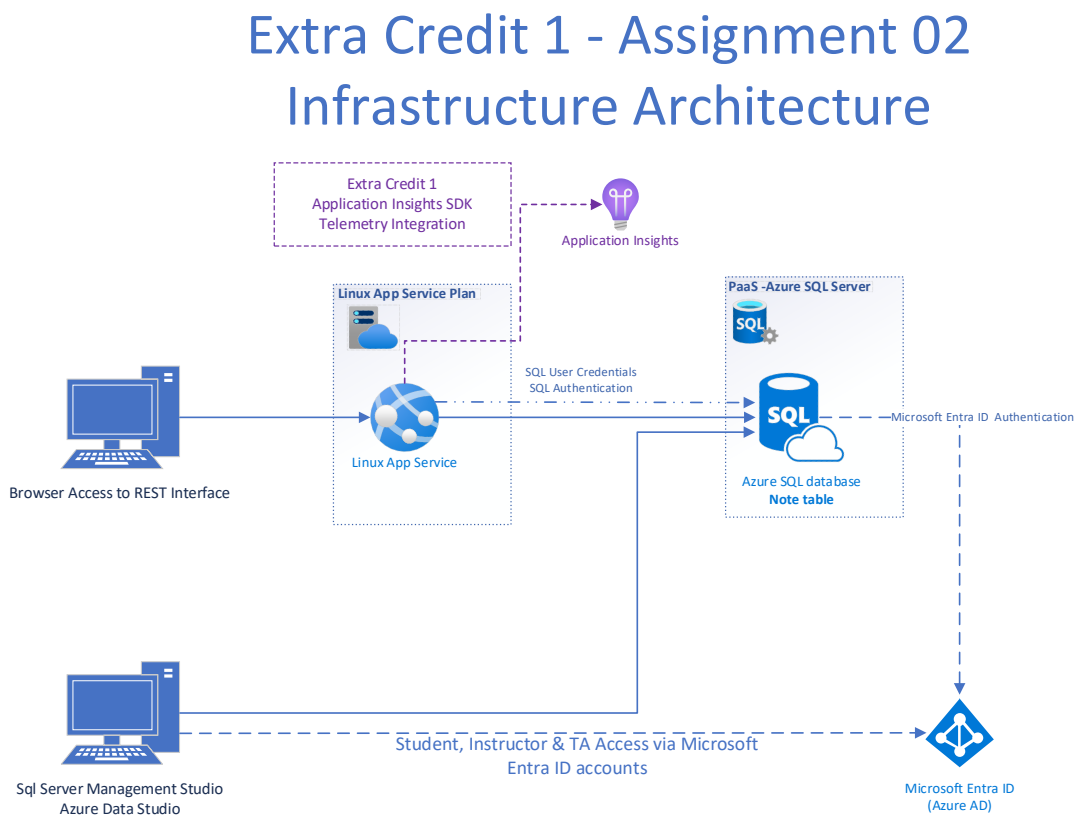
- Up to 15 Points shall be deducted for lack of comments in code.
- All methods, classes etc. must be documented.
    a. C# with XML comments.
- Your project must build with no errors or warnings.

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: Log custom telemetry into application insights.



1. When a note is **created** the event shall have an event name of "**NoteCreated**" and shall contain
1.1. A property, **named summary**, that contains the summary information.
1.2. Two metrics**, SummaryLength** that contains the length of the summary and **DetailsLength** that contains the length of the details field.
1.3. A property named **tagcount**, that contains the number of tags associated with a note.

2. When a note is **updated** the event shall have an event name of "**NoteUpdated**" and shall contain
2.1. A property, **named summary**, that contains the summary information updated. If the summary information was not included in the update the summary value shall be empty
2.2. A property, **named details**, that contains the detailed information updated. If the details information was not included in the update the details value shall be empty
2.3. Three metrics**, SummaryLength** that contains the length of the summary and **DetailsLength** that contains the length of the details field and T**agCount** that contains the number of tags. If either the summary or details were not provided, then the metric value for the corresponding metric **SummaryLength**, **DetailsLength** respectively shall be 0. If the details were not provided, then the metric value **tagcount** shall be 0.

3. All validation errors shall be logged using the **TrackTrace** method and shall include the details of the validation error that occurred and the input payload that caused the exception. The input payload property name shall be "**InputPayload**".

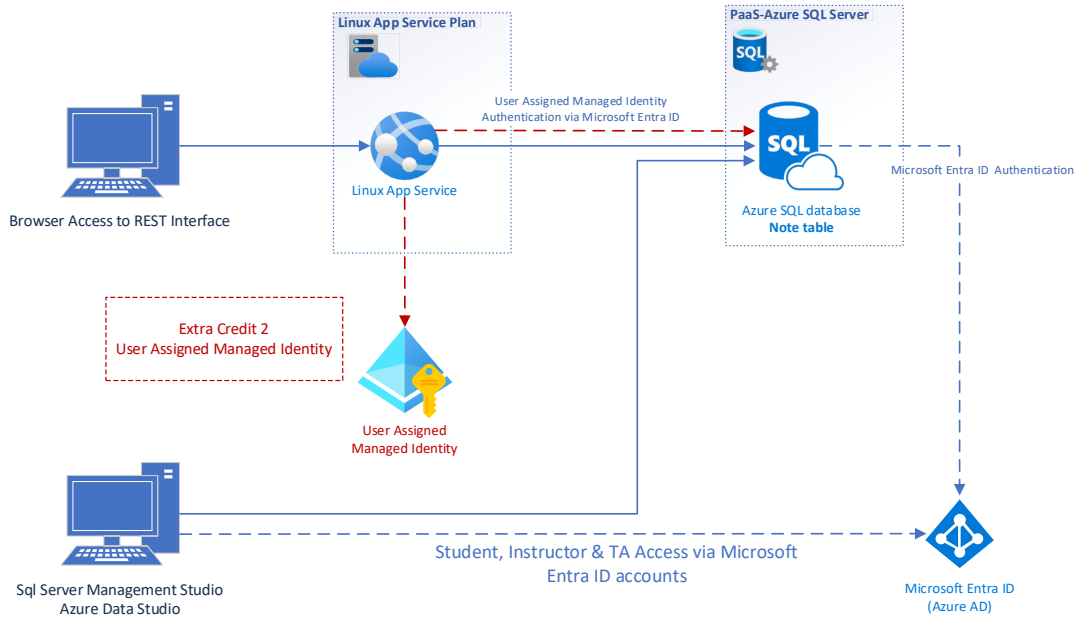
4. All exceptions shall be logged using the **TrackException** method and shall include the details of the exception that occurred and the input payload that caused the exception. The input payload property name shall be "**InputPayload**".

5. Be sure to include the name of your application insights instance in your project notes.txt file

6. Be sure to setup your SDK Control Channel authentication key using the API Access blade in application insights
6.1 Configure this key on your app service's configuration in the application settings area.
6.2 Ensure that your application properly registers this key in your program.cs file.
6.3 Rember this key can't be retrieved once you navigate away from the screen where it is created so you will need to store it in a safe place including your secrets.json file in addition to setting it up on the portal for your app service as described in 6.1
6.4 Provide your SDK Control Channel authentication key in your project notes.txt file

Extra Credit Option 2: Use managed identities for authentication to Azure SQL

Extra Credit 2 - Assignment 02
Infrastructure Architecture



CSCI E94 Fundamentals of Cloud Computing and Open AI with Microsoft Azure
Copyright 2024-2025

Instructor: Joseph Ficara
2025-02-13 v5.00

1. Create a new App Service that utilizes its managed identity to access your Azure SQL Database using your existing Linux app service plan running on the Basic B1 Tier
1.2. Create a managed identity called **id_dbadmin**.
1.3. Configure your app service to use a **user assigned managed identity** with the new managed identity you created in step 1.2.
1.4. Create a user for the **id_dbadmin** managed identity on your database containing the note table.
1.5. Grant db_datareader, db_datawriter and dd_ddmin access to your app service's manage identity account on your database containing the note table.
1.6. Update the connection string, which is in the application settings area, in your new Azure App Service to utilize the Microsoft Entra ID Based connection string that has no credentials in it.
1.7. Don't forget to setup the **AZURE_CLIENT_ID** in the Environment Settings / App Settings for your app service.

2. Ensure your development environment also uses managed identities to access the database. Steps 2.1 – 2.3 should have already been completed as part of the basic requirements.
2.1. Create a user for the Microsoft Account you use to log into the Azure portal on your database containing the note and tag table.
2.2 Grant db_datareader, db_datawriter and dd_ddmin access to the Microsoft Account you use to access the portal to your database containing the note table.
2.3. Repeat steps 2.1 and 2.2 for all the TAs and Instructor's Azure Portal Accounts
2.4. Create a new **appsettings.managedidenties.json** file that contains the connection string that uses the Microsoft Entra ID Based connection string that has no credentials in it. (Remember to remove any existing connection string settings that may be present in your secrets.json file and set your `ASPNETCORE_ENVIRONMENT` variable to managedidenties when testing locally)