Overview:

Update your Note Keeper REST API application to support creation and management of **attachments** using Azure Storage to store the attachments in Blob Storage.

You will create .NET 9 based ASP.NET Core Web API project with Entity Framework Core 9 for the storage of notes and **Azure Blob Storage for the storage of attachments**. You can use the previous assignment as a foundation for this work. All the previous note resources, add, delete, update etc. must still operate as they did in assignment 2. Note there is a required enhancement to the note delete functionality, see 2.5 API Operation – Note Delete Enhancement

The association of attachments to notes is accomplished using the Azure Blob Storage **container name**. **Notes are associated with attachments by setting the container's name to the Note Id**. All blobs in the associated container shall be attached to the corresponding note.

Example: If a note has an Id of 2b2d6477-639e-4631-9a35-36a18c7f8f94, the Azure Blob Storage container name shall be 2b2d6477-639e-4631-9a35-36a18c7f8f94. All blobs in the container named 2b2d6477-639e-4631-9a35-36a18c7f8f94 are considered attached to the note with a note id of: 2b2d6477-639e-4631-9a35-36a18c7f8f94.

You will use Visual Studio to deploy your application to a 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**. Once assignment 1 has been graded either delete that app service or stop it to free CPU and memory resources in you app service plan.

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 Storage
    - Managed Identities for Azure Storage
- Familiarize yourself with
    - Azure Storage
- Gain experience with:
    - Azure SQL Server & Database
    - Visual Studio 2022 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
        - Web API App Services from the Azure Dashboard Logging

Assignment 03
Infrastructure Architecture

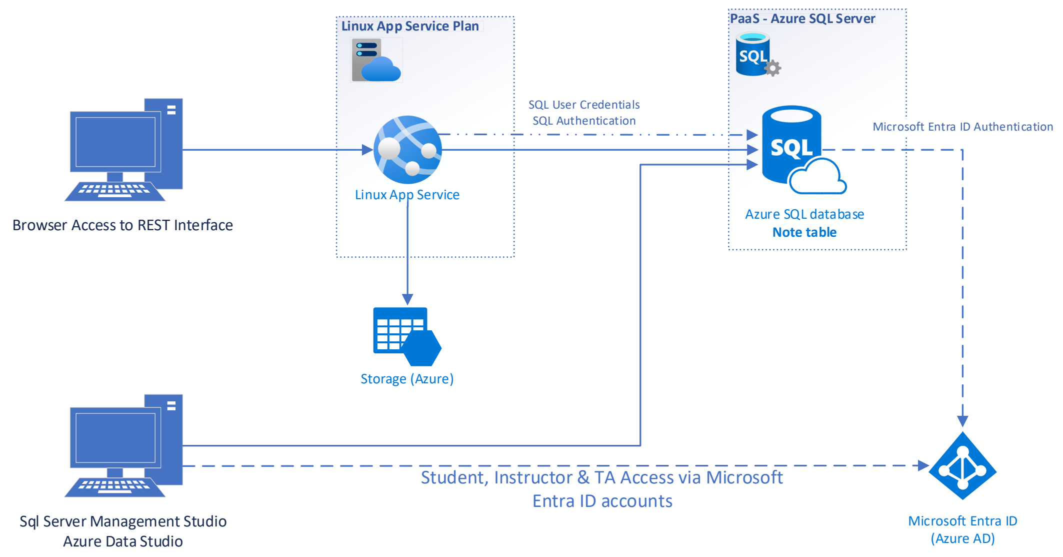


Table of contents

Solution, project naming, and submission information

You shall create one Solution containing one Visual Studio Project. The solution shall be called **HW3NoteKeeperSolution.** The solution shall contain the core project that shall be called **HW3NoteKeeper.** If you implement one or more extra credit options, those projects shall be added to the **HW3NoteKeeperSolution**.

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 **HW3NoteKeeperSolution**

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) **The name of your Azure Storage account**
   10) 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
   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**

1. Create an Azure Storage Account

1.1. You shall create a V2 azure storage account to store attachments.

2. Implement attachments resource

2.1 API Operation – Create and Update

Creates a new attachment entry and **stores the attachment content in Azure Blob Storage** under a **container** with the same name as the associated Note Id. If the attachment already exists it updates the blob with the new attachment provided.

Support a default maximum attachments per note of **3** using the configurable **MaxAttachments** setting, which can be overridden in the Azure Portal's application settings area.

HTTP
PUT `https://[appservicename].azurewebsites.net/notes/{`**`noteId`**`}/attachments/{`**`attachmentId`**`}`

URI Parameters

Name	Data Type	.NET Type	Min Length	Max Length	Description
noteId	string	string	Not Enforced	Not Enforced	The note id of the note **Constraints:** Required, Must Exist
attachmentId	string	string	Not Enforced	Not Enforced	The Id of the attachment which is the name of the file uploaded. **Constraints:** Required, must not be null, empty or whitespace. See Blob Names. You can use the SDK to validate blob names.

Request Body

Name	Required	.NET Type	Constraints	Description & Notes
fileData		IFormFile	1. Required 2. Not Null 3. Not Empty	The file upload data using the IFormFile interface data type

Operation

2.1.1 The content type shall be set using the content type provided by **IFormFile fileData** parameter.
2.1.2 The **attachmentId** shall be used as the BlobId.
2.1.3 The container name shall be the same value as the associated Note Id
2.1.4 The container access shall be private.
2.1.5 Ensure a custom **Metadata** property called **NoteId** is set for the blob with the note Id of the associated note.
2.1.6 If the attachment identified by the **attachmentId** parameter on the URI **exists** then a 204 (No Content) response shall be returned, and the attachment blob shall be updated.
2.1.7 If the attachment identified by the **attachmentId** parameter on the URI **does not exist** then a 201 (Created) response shall be returned, and the attachment blob shall be created. See responses section below
2.2.6 If the **note is NOT** present return a 404 (Not Found) and log the operation using a Log Warning

Responses

204 (No Content)

Returned if the attachment blob was successfully updated.

Headers:

None beyond standard

Response Body:

None

201 (Created)

Returned if the attachment blob was successfully created.

Assuming the *appservicename* is **noteswithattachments** see the following example:

Example:

If the Note Id is **a6066fd2-aa95-42dc-9921-e09975091123** and the **attachmentId** parameter contains "**mypic.png**" and the blob with the id "**mypic.png**" **DOES NOT** exist in a container named **a6066fd2-aa95-42dc-9921-e09975091123**, then a blob will be created with an id of **mypic.png** in a container named **a6066fd2-aa95-42dc-9921-e09975091123**

Headers:

Location:

```
Location: https://noteswithattachments.azurewebsites.net/notes/a6066fd2-aa95-
42dc-9921-e09975091123/attachments/mypic.png
```

Response Body:

None

204 (No Content)

Returned if the attachment blob was successfully updated.

Assuming the *appservicename* is **noteswithattachments** see the following example:

Example:

If the Note Id is **a6066fd2-aa95-42dc-9921-e09975091123** and the **attachmentId** parameter contains "**mypic.png**" and the blob with the id "**mypic.png**" **DOES** exist in a container named **a6066fd2-aa95-42dc-9921-e09975091123**, then the blob with an id of **mypic.png** in the container named **a6066fd2-aa95-42dc-9921-e09975091123** will be updated with the content provided.

Headers:

None beyond standard

Response Body:

None

400 (Bad Request)

Returned if there are 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 attachment will cause the number of attachments stored to exceed the **MaxAttachments** value that has been set.

Headers:

None beyond standard

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

Property	Value
Status	403
Title	Attachment limit reached
Detail	Attachment limit reached MaxAttachments [<the current setting for MaxAttachments]

Example:

```
{
  "title":"Attachment limit reached",
  "status":403,
  "detail":"Attachment limit reached MaxAttachments: [3]
}
```

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

404 (Not Found)
Returned if the note or attachment 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.

2.2 API Operation – Delete

Deletes an attachment having the specified attachment id from Azure Blob Storage.

HTTP
DELETE `https://[appservicename].azurewebsites.net/notes/{`**`noteId`**`}/attachments/{`**`attachmentId`**`}`

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
attachmentId	string	string	Not enforced	Not enforced	The attachmentId of the attachment This is the filename associated with the attachment that was created by the PUT operation. See 2.1 API Operation – Create and Update **Constraints**: Required, Must Exist.

Operation

2.2.1 If the attachment blob is present but there is a failure to delete the attachment blob, return a 500 (Internal Server Error) and log the issue using a **Log Error**

2.2.2 If the attachment blob is present and deleted with success return a 204 (No Content) and log the operation using a **Log Information**

2.2.3 If the attachment blob is **NOT** present return a 204 (No Content) and log the operation using a Log Warning.

There is a lot of controversy around this scenario, however to make things simpler for the client we are opting with the success scenario as the end state is the desired state, the attachment is not present.

2.2.4 If the **note** is **NOT** present return a 404 (Not Found) and log the operation using a Log Warning

Request Body

None

Responses

204 (No Content)

Returned if the attachment was deleted successfully or if the attachment does not exist when attempting to delete it.

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.

500 (Internal Server Error)

Returned if the attachment is present but could not be deleted.

Headers:

None beyond standard

Response Body:

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

2.3 API Operation – Retrieve by Id

Retrieves an attachment having the specified attachment id from blob storage

HTTP
GET `https://[appservicename].azurewebsites.net/notes/{`**`noteId`**`}/attachments/{`**`attachmentId`**`}`

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
attachmentId	string	string	Not enforced	Not enforced	The attachmentId of the attachment This is the filename associated with the attachment that was created by the PUT operation. See 2.1 API Operation – Create and Update **Constraints**: Required, Must Exist.

Request Body

None

Responses

200 (OK)

Returned if the attachment was successfully retrieved.

Headers:

None beyond standard

Response Body:

The attachment retrieved from Block Blob storage with content type set to the content type of the stored attachment. See FileStreamResult

Ensure the downloaded file has the name as the blob id.

404 (Not Found)

Returned if the note or the attachment 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.

2.4 API Operation – Retrieve all

Retrieves all attachment blob Ids with the blob's summary information for the associated note.

HTTP
GET `https://[appservicename].azurewebsites.net/notes{`**`noteId`**`}/attachments`

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

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

```
[
  {
    "attachmentId": "mypic1.jpg",
    "contentType": "image/jpeg",
    "created":"{2/14/2022 10:53:09 PM +00:00}",
    "lastModified":"{2/14/2022 10:53:09 PM +00:00}",
    "length":2185980
  },
  {
    "attachmentId": "mydoc.pdf",
    "contentType": "application/pdf",
    "created":"{2/15/2022 1:22:01 PM +00:00}",
    "lastModified":"{2/16/2022 12:33:21 PM +00:00}",
    "length":180152
  }
```

Name	JSON Type	.NET Type	Description & Notes
attachmentId	string	string	The id of the blob
contentType	string	string	The content type of the blob
createdDate	string	DateTimeOffset	The date/time created as recorded by the blob SDK
lastModifiedDate	string	DateTimeOffset	The last modified date as recorded by the blob SDK
length	string	long	The length of the blob as recorded by the blob SDK

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.

2.5 API Operation – Note Delete Enhancement

Enhance the previous implementation of note deletion and add support for the deletion of the note's associated attachments. You shall continue to delete the note from the Azure SQL Database **note** table and all of the associated tags from the **tag** table. Add support for deleting the note's attachments from blob storage. Be sure to delete all attachments and the container the attachments reside in.

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.

Operation

2.5.1 Delete the note from the database, if the note does not exist return a 404

2.5.2 Delete the tags associated with the note, if there is a failure in deletion of the tags you shall not delete the note or the attachments and log an error with the details of the failure, and return with a 500.

2.5.3 Delete all attachments from blob storage, if there are no attachments no error is raised.

2.5.4 Delete the attachment container from blob storage, if there is no associated container no error is raised.

2.5.5 If there is a failure to delete an existing attachment or a failure to delete the container, log an error with the details of the failure(s), but do not report a failure to the caller. A separate log entry shall be logged for each storage related failure that occurs;

For example, if there are three blobs and two blobs fail to be deleted then log two errors, one for each blob that failed to be deleted. Also, in this scenario it's likely that the container failed to be deleted so a third log entry indicating the details of the container deletion failure shall also be logged.

Alternatively, you can delete the container and all associated blobs within the container at the same time. If the container deletion fails log an error indicating the container deletion failure but do not return an error to the caller. If you take this approach note it in your project notes .txt file.

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.

3. Add the following attachments when seeding

Your API app shall automatically seed Azure Storage with the attachments associated with the following notes. These files are in the **SampleAttachments** directory of the **03-Assignment.zip** file.

Be sure that that the **NoteId** metadata entry is set to the associated note Id that is auto assigned.

You can start with an empty set of tables so that you seed the azure table and blobs in azure storage using the same conditional logic you used to seed the table in Azure Storage.

It is not necessary to handle cases where one or more of the notes or attachments are missing, your logic can seed only when there are no note entries in the Azure SQL Database table.

When submitted the table must have all the notes in the database and all attachments stored as blobs in blob storage under a container with the name of the associated note id. Additionally, you shall also have all the tags generated via the o4 mini model as described in assignment 2.

The relationship between the note in the database and attachment in blob storage is established by the container name the blob resides in. The container name has the same value as the associated note id.

There are no changes to the database schema, which consists of the table definition, from assignment 2.

You may perform the **seeding by running locally and connecting to azure**, which means you don't have to deploy the seeding attachment files to your website if you don't want to.

Table Schema Definition				
Id	**Name**	**Details**	**CreatedDateUtc**	**Modified DateUtc**
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

Not included in table schema
associated attachments stored in blob storage
MilkAndEggs.png Oranges.png
WrappingPaper.png Tape.png
Chocolate.png Diamonds.png NewCar.png
AzureLogo.png AzureTipsAndTricks.pdf

4. Enhance the custom setting to include the number of attachments that can be added

4.1. The custom setting property shall be called **MaxAttachments** and shall be type **int**

4.2. The default value shall be **3**

4.3 Add the setting to the custom setting section **NoteSettings**

5. Ensure your swagger UI in this assignment still operates as it did for assignment 2 with the inclusion of the Attachments resource operations

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. 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

6. 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.

7. Create an Azure App Service

Create a **new Azure App Service** on your existing **Azure Linux Azure App Service Plan** and deploy your note keeper Web API implementation your new Azure App Service. 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:

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.

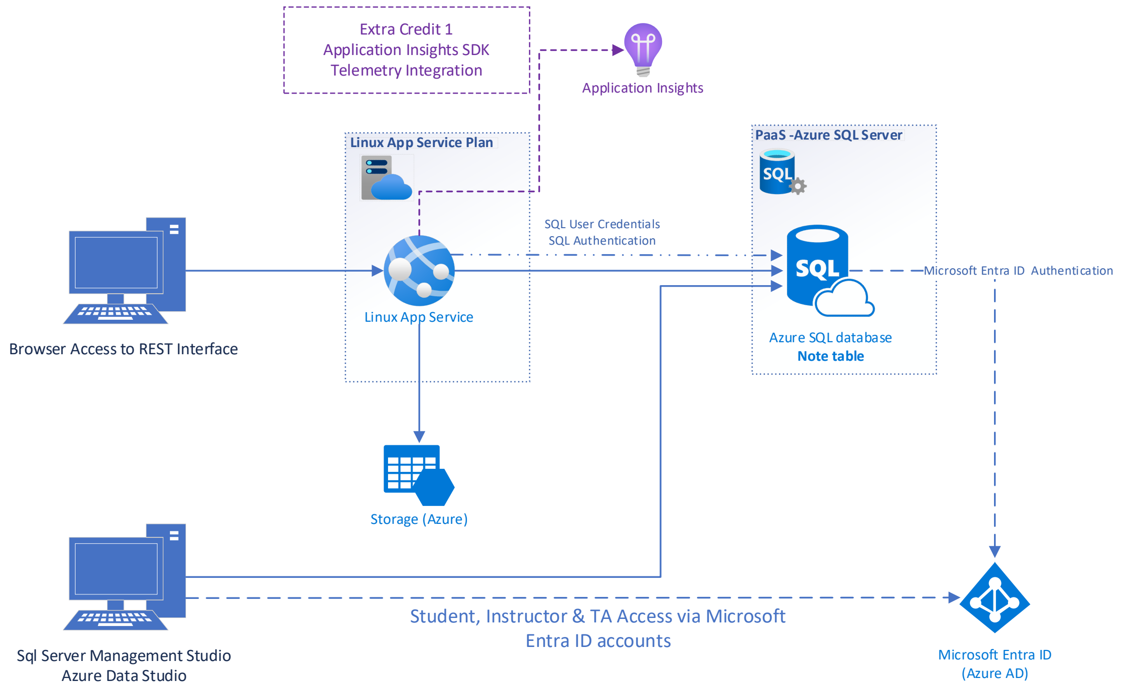
1. When an attachment is **created** the event shall have an event name of "**AttachmentCreated**" and shall contain
1.1. A property, **named attachmentid**, that contains the attachment blobid information.
1.2. A single metric**, AttachmentSize** that contains the length of the attachment.

2. When an attachment is **updated** the event shall have an event name of "**AttachmentUpdated**" and shall contain
2.1. A property, **named attachmentid**, that contains the attachment blobid information.
2.2. A single metric**, AttachmentSize** that contains the length of the attachment.

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**".

Extra Credit 1 - Assignment 03 Infrastructure Architecture



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

Instructor: Joseph Ficara
2025-02-27 v5.00

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

1. Create a new App Service that utilizes its managed identity to access your Azure Storage account
1.2. Add a user assigned managed identity to your new Azure App Service. You could use the same user assigned managed identity from assignment 2 if you implemented the associated extra credit.
1.3. Grant your App Service the **Storage Blob Data Owner** role for access to the **blob storage account**
1.4. Update the connection string to Azure Storage in your new Azure App Service to utilize the Azure AD Based connection string that has no credentials in it.

2. Ensure your development environment also uses managed identities to access the azure storage
2.1. Assign the user for the Microsoft Account you use to log into the Azure portal the **Storage Blob Data Owner role on the Azure Storage Account**
2.3. **Repeat step 2.1 for all the TAs and Instructor's Azure Portal Accounts**
2.4. Create a new **appsettings.managedidentities.json** file that contains the connection string that uses the Azure AD Based connection string to your azure storage account 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 **managedidentities** when testing locally.) When deploying to Azure you may use portal overrides for your storage account connection settings or the **appsettings.managedidentities.json** file. If using the **appsettings.managedidentities.json** you will need to `ASPNETCORE_ENVIRONMENT` variable to **managedidentities** in the Azure Portal Application Settings.