**Lab 3**

**Mustafa Zahid 20K-1045**

**Hassan Ali 20K-1052**

The lab guided through several key steps to access Azure blob storage from a .NET application using the Azure Storage SDK:

1. Create Azure resources

- Created a new resource group named StorageMedia

- Created a new storage account named mediastor[yourname] with LRS redundancy

- Enabled anonymous access on containers

- Recorded the blob endpoint URL, account name, and access key

2. Upload a blob

- Created two private containers named raster-graphics and compressed-audio

- Uploaded an image file graph.jpg into the raster-graphics container

3. Access containers using the SDK

- Created a .NET 6 console app and added the Azure.Storage.Blobs NuGet package

- Authenticated using StorageSharedKeyCredential with the access key

- Connected to the blob endpoint using BlobServiceClient

- Retrieved account metadata using GetAccountInfoAsync()

- Looped through containers using GetBlobContainersAsync()

4. Retrieve blob URIs

- Created EnumerateBlobsAsync to loop through blobs

- Called it on the raster-graphics container

- Created GetContainerAsync to create/get a container client

- Called it to create a vector-graphics container

- Uploaded graph.svg via the portal into vector-graphics

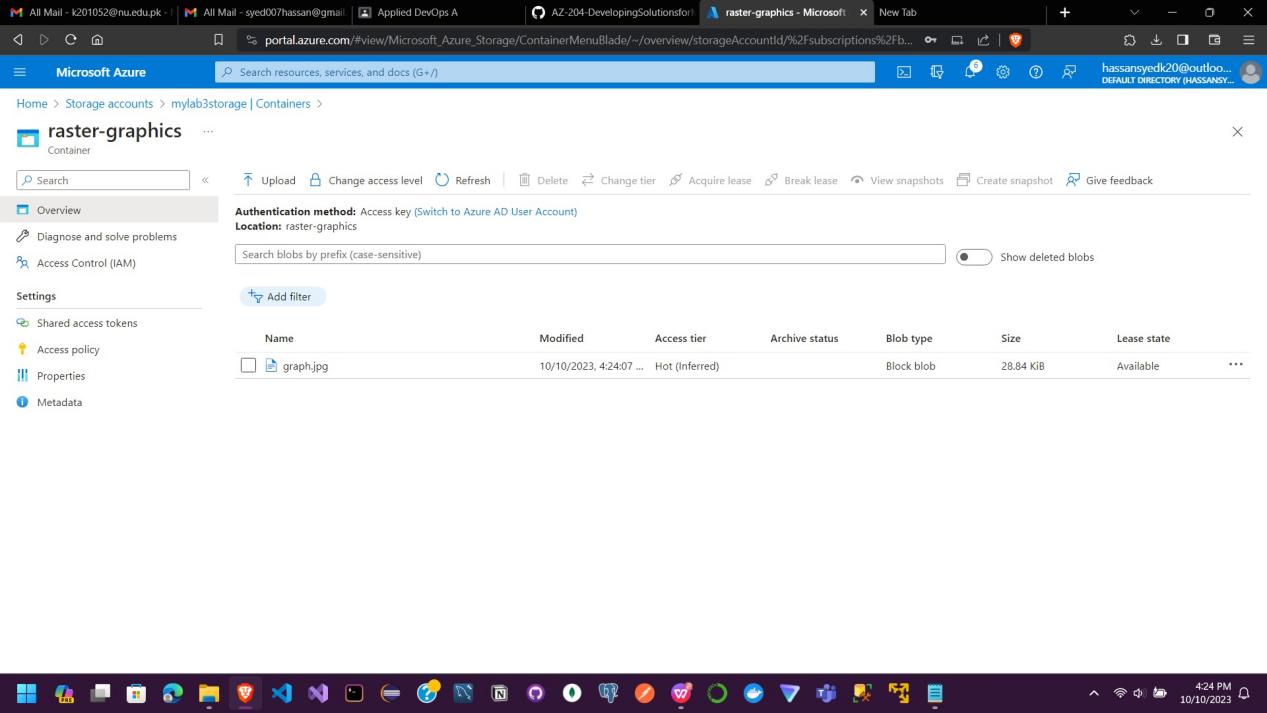
- Created GetBlobAsync to get a blob client and output the URI

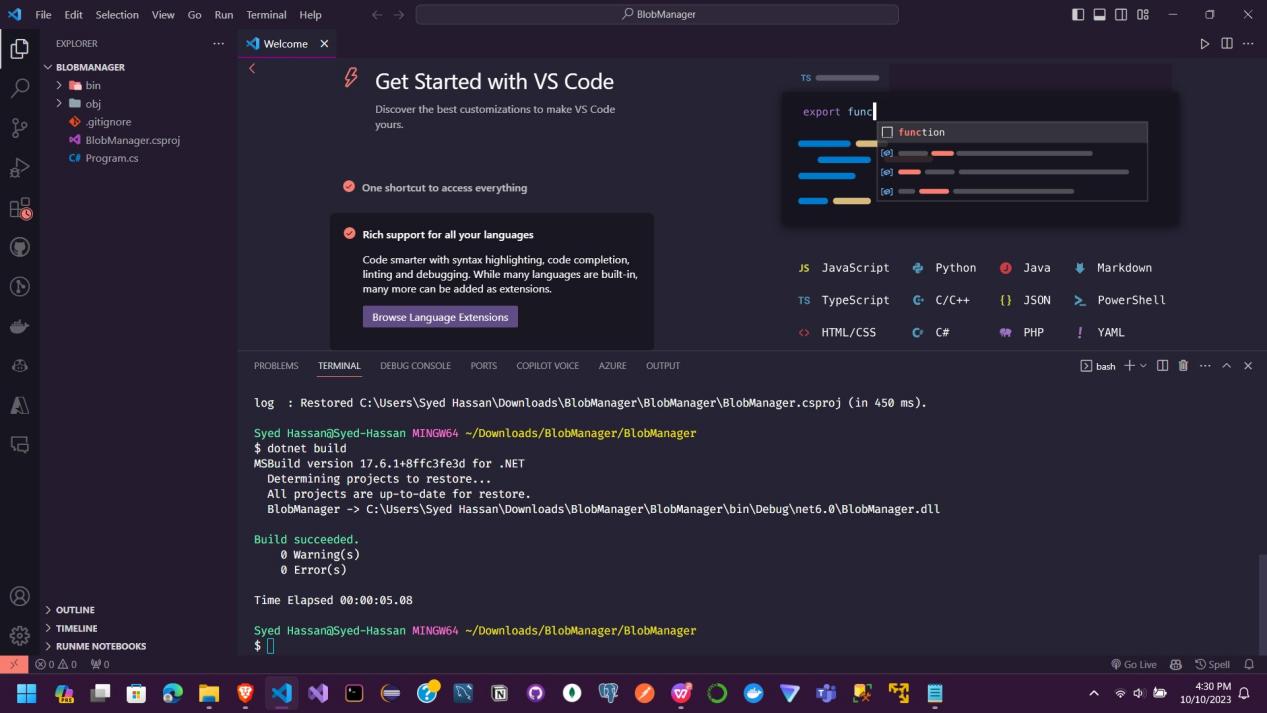
- Called it on graph.svg and output the blob URI

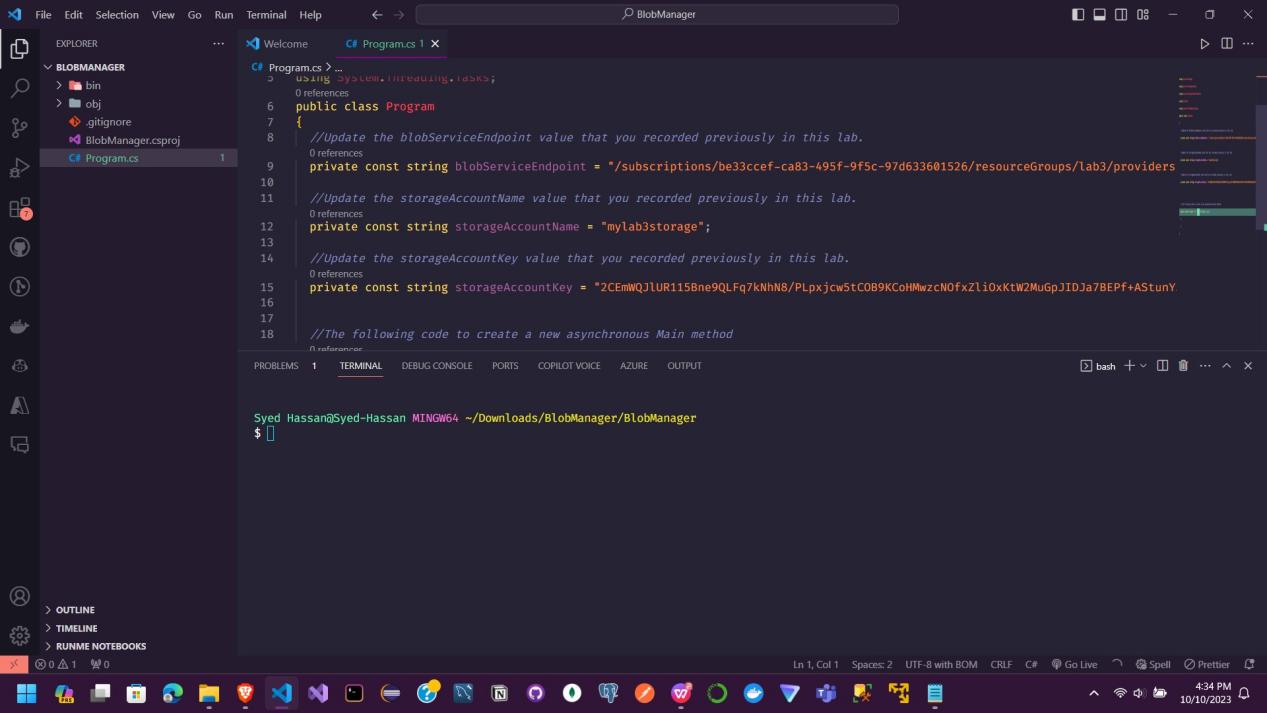
5. Test URI

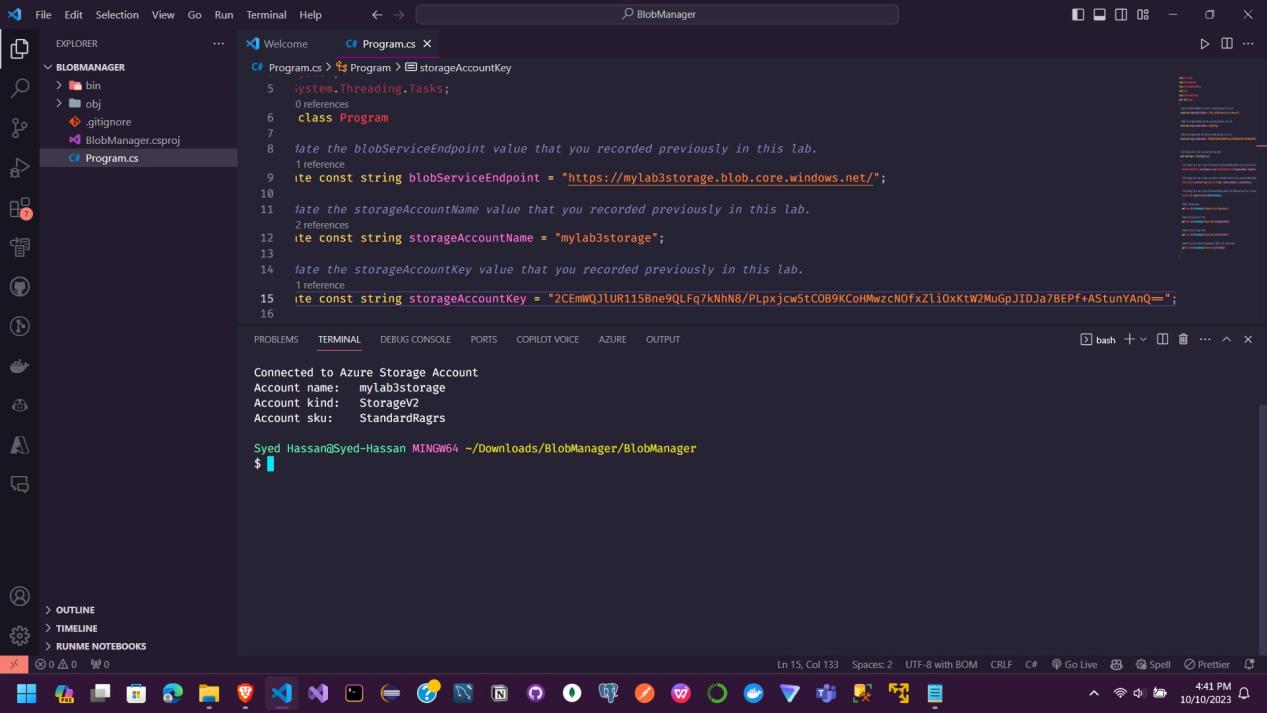
- Opened the blob URI in the browser to confirm access

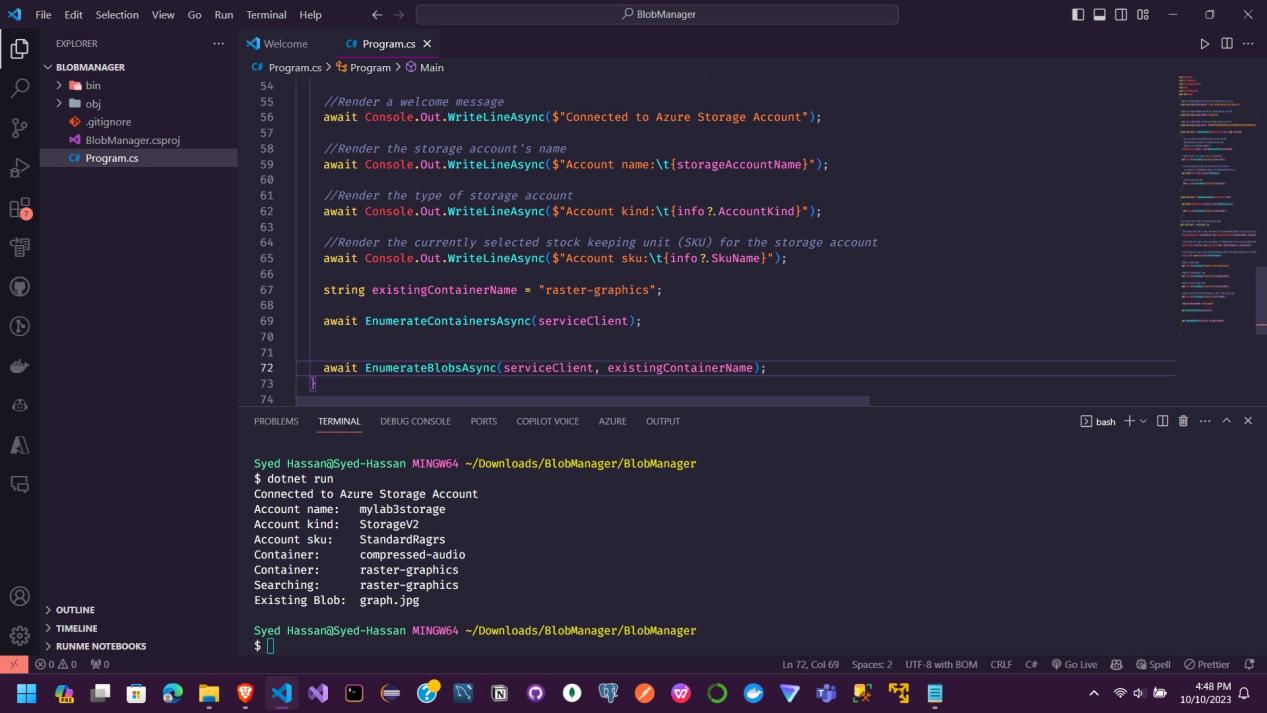
In summary, the lab provided end-to-end experience creating Azure resources, using the .NET SDK to manage blobs, generating access URIs, and testing blob access in the browser. The key takeaway is how to leverage the Azure Storage SDK in a .NET app to provide programmatic access to Azure blob storage.

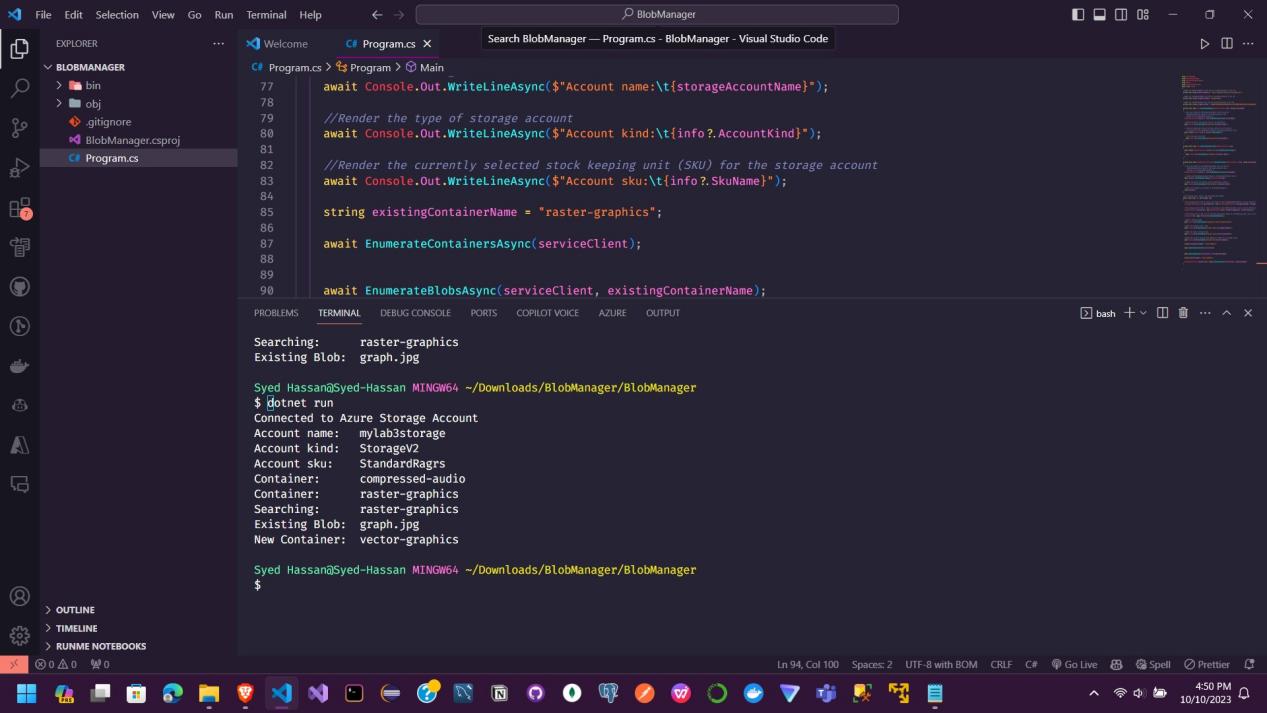


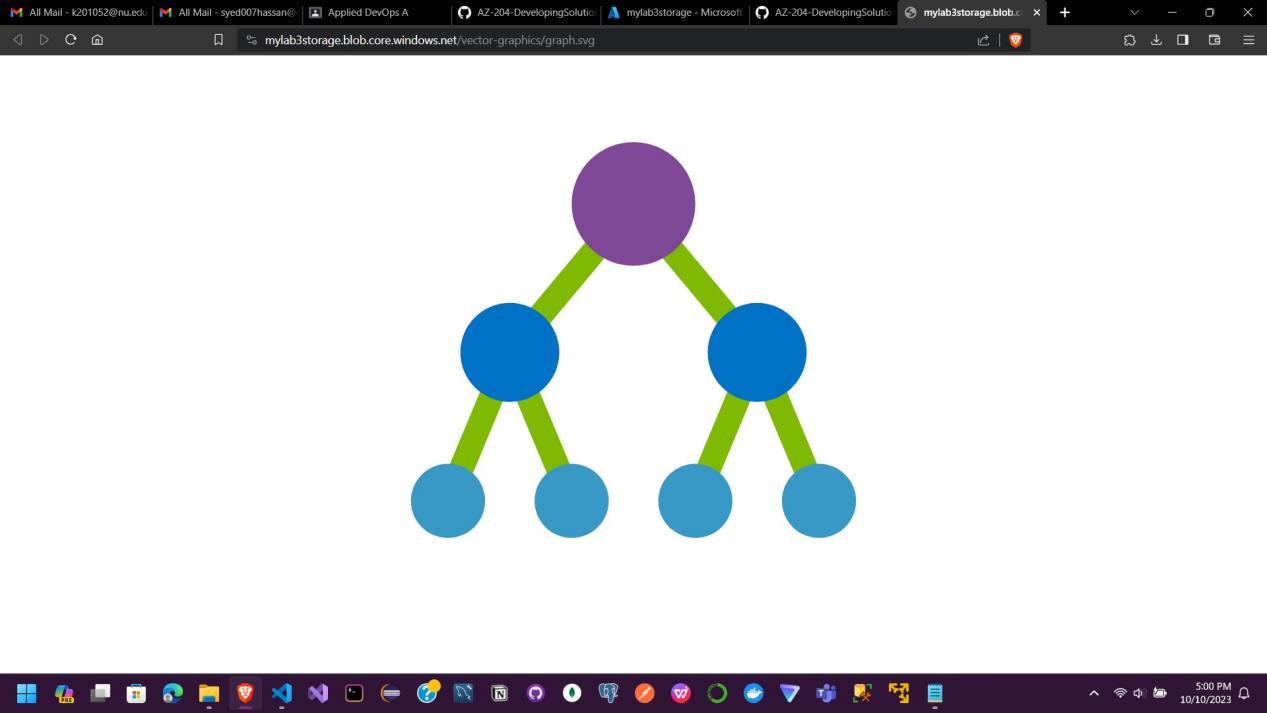












Real Life Application

Here is a real-world example of how the concepts in this lab could be applied:

**Photo Sharing Application**

A photo sharing app that allows users to upload and share images could leverage Azure blob storage and the .NET SDK like this:

- Create an Azure storage account to store images

- Use the SDK to create a "photos" container for image uploads

- When a user uploads a photo, use the SDK to upload the image as a blob

- The app would call GetBlobClient to get a reference to the new blob

- Generate a SAS token for temporary read access to the blob

- Return the blob URI with the SAS token to the app

- The app can display the image in the browser using the URI

This allows the app to securely upload images to Azure blob storage. The SDK is used behind the scenes to manage access and upload files. Shared access signatures provide temporary access to read the images.

The same approach could apply for videos, audio files, documents, or any other media the app needs to store in Azure. The SDK handles authentication and uploading/downloading while the app focuses on user experiences.

So in summary, the lab concepts allow building rich media applications with Azure as the backend for storage and content delivery. The SDK removes the complexity so developers can focus on core app experiences.

**Purpose**

The main purpose of this lab is to demonstrate how to programmatically manage and access Azure blob storage from a .NET application using the Azure Storage SDK.

Some of the key learning objectives include:

- Learning how to authenticate and interact with Azure Storage using a familiar SDK from within .NET code.

- Practicing creating Azure Storage accounts, containers, and uploading blobs through code.

- Seeing how to generate URIs with SAS tokens to provide limited access to private blobs.

- Getting first-hand experience with the storage account hierarchy of containers and blobs.

- Understanding how the SDK handles connecting to storage endpoints, container/blob operations, and security.

- Learning techniques to list, iterate through, and retrieve metadata about storage objects like accounts, containers, and blobs.

- Discovering how easy and productive it can be to integrate cloud storage into an application with the Azure Storage SDK.

At a high level, it aims to unlock Azure blob storage capabilities for .NET developers in a hands-on, practical way. The skills to programmatically manage storage with the SDK are foundational for most real-world applications that use Azure.

So in summary, its purpose is to teach core skills for access and managing Azure Storage from .NET apps, which opens up many possibilities for building robust cloud applications.

**Another real life example**

Here is another real-world example of how the concepts covered in this lab could be applied:

**Document Management System**

Many businesses need to store, manage, and share documents like reports, invoices, presentations, etc. A document management system could leverage Azure blob storage and the .NET SDK like this:

- Create a storage account to store the documents

- Use the SDK to create containers for each department (marketing, sales, HR, etc)

- Allow users to upload documents to their department container

- The app would call the SDK to upload each document as a blob

- Create an index blob in each container to track documents

- The app updates the index blob when new files are added

- To download a document, query the index blob for the file name

- Get the blob client using the SDK and generate a SAS URI

- Return the SAS URI to the user to download the file

This allows the app to securely store business documents in Azure blob storage. The SDK handles authentication, uploading, and downloading in the background. SAS tokens provide temporary access to download files.

The same approach could apply for any files users need to upload/download like videos, photos, music, etc. The SDK abstracts the storage complexities so developers can focus on app functionality and user experience.

In summary, this example demonstrates a secure document management use case powered by Azure blob storage and the .NET SDK for the backend. The concepts from the lab provide the foundation for building many real-world solutions.

**Similarity/differences with Firebase:**

Yes, there are some similarities between using Azure Storage with the .NET SDK and Firebase for uploading files:

- They both allow storing user-generated content like images, videos, etc. in the cloud.

- They handle authentication and securely uploading files behind the scenes.

- Developers can focus on building app experiences vs storage logic.

- Generated URLs provide access to uploaded content.

Some key differences:

- Firebase is a full backend-as-a-service with built-in SDKs, while Azure Storage is infrastructure you can access with any SDK.

- Firebase includes hosting for static content, Azure Storage blobs would need a compute service for hosting.

- Firebase has native SDKs for mobile apps, Azure Storage can be used from any platform.

- Azure Storage offers more flexibility and control over storage architecture.

- Pricing and scales are different. Azure charges for usage while Firebase has tiers.

So in summary, they both enable cloud storage for apps, but Firebase is higher-level while Azure Storage is more flexible and universal. The .NET SDK provides a familiar way for .NET developers to access Azure.

The lab concepts demonstrate foundational techniques like uploading, downloading, and generating URLs that apply to many cloud storage services including Firebase. The implementation and managing storage details differ across providers.