ABCRetailers Web Application - Services Code Snapshots

Services

IAzureStorageService.cs

```
IAzureStorageService.cs   AzureStorageService.cs
ABCRetailers   ABCRetailers.Services.IAzureStorageService   GetAllEntitiesAsync<T>()
// Services/IAzureStorageService.cs
using ABCRetailers.Models;

namespace ABCRetailers.Services
{
    12 references
    public interface IAzureStorageService
    {
        // Table operations
        12 references
        Task<List<T>> GetAllEntitiesAsync<T>() where T : class, Azure.Data.Tables.ITableEntity, new();
        10 references
        Task<T?> GetEntityAsync<T>(string partitionKey, string rowKey) where T : class, Azure.Data.Tables.ITableEntity, new();
        4 references
        Task<T> AddEntityAsync<T>(T entity) where T : class, Azure.Data.Tables.ITableEntity;
        6 references
        Task<T> UpdateEntityAsync<T>(T entity) where T : class, Azure.Data.Tables.ITableEntity;
        4 references
        Task DeleteEntityAsync<T>(string partitionKey, string rowKey) where T : class, Azure.Data.Tables.ITableEntity, new();

        // Blob operations
        3 references
        Task<string> UploadImageAsync(IFormFile file, string containerName);
        2 references
        Task<string> UploadFileAsync(IFormFile file, string containerName);
        1 reference
        Task DeleteBlobAsync(string blobName, string containerName);

        // Queue operations
        4 references
        Task SendMessageAsync(string queueName, string message);
        1 reference
        Task<string?> ReceiveMessageAsync(string queueName);

        // File Share operations
        2 references
        Task<string> UploadToFileShareAsync(IFormFile file, string shareName, string directoryName = "");
        1 reference
        Task<byte[]> DownloadFromFileShareAsync(string shareName, string fileName, string directoryName = "");
    }
}
```

AzureStorageService.cs

Difficult to take Full Code Snapshot – Watch Video Recording for Start and End of Code – The Start and End of Code is also captured in the First and Last Snapshot Respectively)

AzureStorageService.cs (AzureStorageServices Action - The Preamble of the code is captured here)

AzureStorageService.cs

ABCRetailers | ABCRetailers.Services.AzureStorageService

```
// Services/AzureStorageService.cs
using Azure.Data.Tables;
using Azure.Storage.Blobs;
using Azure.Storage.Queues;
using Azure.Storage.Files.Shares;
using ABCRetailers.Models;
using System.Text.Json;

namespace ABCRetailers.Services
{
    4 references
    public class AzureStorageService : IAzureStorageService
    {
        private readonly TableServiceClient _tableServiceClient;
        private readonly BlobServiceClient _blobServiceClient;
        private readonly QueueServiceClient _queueServiceClient;
        private readonly ShareServiceClient _shareServiceClient;
        private readonly ILogger<AzureStorageService> _logger;

        0 references
        public AzureStorageService(
            IConfiguration configuration,
            ILogger<AzureStorageService> logger)
        {
            string connectionString = configuration.GetConnectionString("AzureStorage")
                ?? throw new InvalidOperationException("Azure Storage connection string not found");

            _tableServiceClient = new TableServiceClient(connectionString);
            _blobServiceClient = new BlobServiceClient(connectionString);
            _queueServiceClient = new QueueServiceClient(connectionString);
            _shareServiceClient = new ShareServiceClient(connectionString);
            _logger = logger;

            InitializeStorageAsync().Wait();
        }
```

AzureStorageService.cs (InitializeStorageAsync Action)

```
1 reference
private async Task InitializeStorageAsync()
{
    try
    {
        _logger.LogInformation("Starting Azure Storage initialization...");

        // Create tables
        await _tableServiceClient.CreateTableIfNotExistsAsync("Customers");
        await _tableServiceClient.CreateTableIfNotExistsAsync("Products");
        await _tableServiceClient.CreateTableIfNotExistsAsync("Orders");
        _logger.LogInformation("Tables created successfully");

        // Create blob containers with retry logic
        var productImagesContainer = _blobServiceClient.GetBlobContainerClient("product-images");
        await productImagesContainer.CreateIfNotExistsAsync(Azure.Storage.Blobs.Models.PublicAccessType.Blob);

        var paymentProofsContainer = _blobServiceClient.GetBlobContainerClient("payment-proofs");
        await paymentProofsContainer.CreateIfNotExistsAsync(Azure.Storage.Blobs.Models.PublicAccessType.None);

        _logger.LogInformation("Blob containers created successfully");

        // Create queues
        var orderQueue = _queueServiceClient.GetQueueClient("order-notifications");
        await orderQueue.CreateIfNotExistsAsync();

        var stockQueue = _queueServiceClient.GetQueueClient("stock-updates");
        await stockQueue.CreateIfNotExistsAsync();

        _logger.LogInformation("Queues created successfully");

        // Create file share
        var contractsShare = _shareServiceClient.GetShareClient("contracts");
        await contractsShare.CreateIfNotExistsAsync();

        // Create payments directory in contracts share
        var contractsDirectory = contractsShare.GetDirectoryClient("payments");
        await contractsDirectory.CreateIfNotExistsAsync();

        _logger.LogInformation("File shares created successfully");

        _logger.LogInformation("Azure Storage initialization completed successfully");
    }
    catch (Exception ex)
    {
        _logger.LogError(ex, "Failed to initialize Azure Storage: {Message}", ex.Message);
        throw; // Re-throw to make the error visible
    }
}
```

AzureStorageService.cs (GetAllEntitiesAsync Action)

```
// Table Operations
12 references
public async Task<List<T>> GetAllEntitiesAsync<T>() where T : class, ITableEntity, new()
{
    var tableName = GetTableName<T>();
    var tableClient = _tableServiceClient.GetTableClient(tableName);
    var entities = new List<T>();

    await foreach (var entity in tableClient.QueryAsync<T>())
    {
        entities.Add(entity);
    }

    return entities;
}
```

AzureStorageService.cs (GetEntityAsync Action)

```
10 references
public async Task<T?> GetEntityAsync<T>(string partitionKey, string rowKey) where T : class, ITableEntity, new()
{
    var tableName = GetTableName<T>();
    var tableClient = _tableServiceClient.GetTableClient(tableName);

    try
    {
        var response = await tableClient.GetEntityAsync<T>(partitionKey, rowKey);
        return response.Value;
    }
    catch (Azure.RequestFailedException ex) when (ex.Status == 404)
    {
        return null;
    }
}
```

AzureStorageService.cs (AddEntityAsync Action)

```
4 references
public async Task<T> AddEntityAsync<T>(T entity) where T : class, ITableEntity
{
    var tableName = GetTableName<T>();
    var tableClient = _tableServiceClient.GetTableClient(tableName);

    await tableClient.AddEntityAsync(entity);
    return entity;
}
```

AzureStorageService.cs (UpdateEntityAsync Action)

```
6 references
public async Task<T> UpdateEntityAsync<T>(T entity) where T : class, ITableEntity
{
    var tableName = GetTableName<T>();
    var tableClient = _tableServiceClient.GetTableClient(tableName);

    try
    {
        // Use IfMatch condition for optimistic concurrency
        await tableClient.UpdateEntityAsync(entity, entity.ETag, TableUpdateMode.Replace);
        return entity;
    }
    catch (Azure.RequestFailedException ex) when (ex.Status == 412)
    {
        // Precondition failed - entity was modified by another process
        _logger.LogWarning("Entity update failed due to ETag mismatch for {EntityType} with RowKey {RowKey}",
            typeof(T).Name, entity.RowKey);
        throw new InvalidOperationException("The entity was modified by another process. Please refresh and try again.");
    }
    catch (Exception ex)
    {
        _logger.LogError(ex, "Error updating entity {EntityType} with RowKey {RowKey}: {Message}",
            typeof(T).Name, entity.RowKey, ex.Message);
        throw;
    }
}
```

AzureStorageService.cs (DeleteEntityAsync Action)

```
4 references
public async Task DeleteEntityAsync<T>(string partitionKey, string rowKey) where T : class, ITableEntity, new()
{
    var tableName = GetTableName<T>();
    var tableClient = _tableServiceClient.GetTableClient(tableName);

    await tableClient.DeleteEntityAsync(partitionKey, rowKey);
}
```

AzureStorageService.cs (UploadImageAsync Action)

```
// Blob Operations
3 references
public async Task<string> UploadImageAsync(IFormFile file, string containerName)
{
    try
    {
        var containerClient = _blobServiceClient.GetBlobContainerClient(containerName);

        // Ensure container exists
        await containerClient.CreateIfNotExistsAsync(Azure.Storage.Blobs.Models.PublicAccessType.Blob);

        var fileName = $"{Guid.NewGuid()}{Path.GetExtension(file.FileName)}";
        var blobClient = containerClient.GetBlobClient(fileName);

        using var stream = file.OpenReadStream();
        await blobClient.UploadAsync(stream, overwrite: true);

        return blobClient.Uri.ToString();
    }
    catch (Exception ex)
    {
        _logger.LogError(ex, "Error uploading image to container {ContainerName}: {Message}", containerName, ex.Message);
        throw;
    }
}
```

AzureStorageService.cs (UploadFileAsync Action)

```
2 references
public async Task<string> UploadFileAsync(IFormFile file, string containerName)
{
    try
    {
        var containerClient = _blobServiceClient.GetBlobContainerClient(containerName);

        // Ensure container exists
        await containerClient.CreateIfNotExistsAsync(Azure.Storage.Blobs.Models.PublicAccessType.None);

        var fileName = $"{DateTime.Now:yyyyMMdd_HHmmss}_{file.FileName}";
        var blobClient = containerClient.GetBlobClient(fileName);

        using var stream = file.OpenReadStream();
        await blobClient.UploadAsync(stream, overwrite: true);

        return fileName;
    }
    catch (Exception ex)
    {
        _logger.LogError(ex, "Error uploading file to container {ContainerName}: {Message}", containerName, ex.Message);
        throw;
    }
}
```

AzureStorageService.cs (DeleteBlobAsync Action)

```
1 reference
public async Task DeleteBlobAsync(string blobName, string containerName)
{
    var containerClient = _blobServiceClient.GetBlobContainerClient(containerName);
    var blobClient = containerClient.GetBlobClient(blobName);

    await blobClient.DeleteIfExistsAsync();
}
```

AzureStorageService.cs (SendMessageAsync Action)

```
// Queue Operations
4 references
public async Task SendMessageAsync(string queueName, string message)
{
    var queueClient = _queueServiceClient.GetQueueClient(queueName);
    await queueClient.SendMessageAsync(message);
}
```

AzureStorageService.cs (ReceiveMessageAsync Action)

```
1 reference
public async Task<string?> ReceiveMessageAsync(string queueName)
{
    var queueClient = _queueServiceClient.GetQueueClient(queueName);
    var response = await queueClient.ReceiveMessageAsync();

    if (response.Value != null)
    {
        await queueClient.DeleteMessageAsync(response.Value.MessageId, response.Value.PopReceipt);
        return response.Value.MessageText;
    }

    return null;
}
```

AzureStorageService.cs (UploadToFileShareAsync Action)

```
// File Share Operations
2 references
public async Task<string> UploadToFileShareAsync(IFormFile file, string shareName, string directoryName = "")
{
    var shareClient = _shareServiceClient.GetShareClient(shareName);
    var directoryClient = string.IsNullOrEmpty(directoryName)
        ? shareClient.GetRootDirectoryClient()
        : shareClient.GetDirectoryClient(directoryName);

    await directoryClient.CreateIfNotExistsAsync();

    var fileName = $"{DateTime.Now:yyyyMMdd_HHmmss}_{file.FileName}";
    var fileClient = directoryClient.GetFileClient(fileName);

    using var stream = file.OpenReadStream();
    await fileClient.CreateAsync(stream.Length);
    await fileClient.UploadAsync(stream);

    return fileName;
}
```

AzureStorageService.cs (DownloadFromFileShareAsync Action)

```
1 reference
public async Task<byte[]> DownloadFromFileShareAsync(string shareName, string fileName, string directoryName = "")
{
    var shareClient = _shareServiceClient.GetShareClient(shareName);
    var directoryClient = string.IsNullOrEmpty(directoryName)
        ? shareClient.GetRootDirectoryClient()
        : shareClient.GetDirectoryClient(directoryName);

    var fileClient = directoryClient.GetFileClient(fileName);
    var response = await fileClient.DownloadAsync();

    using var memoryStream = new MemoryStream();
    await response.Value.Content.CopyToAsync(memoryStream);

    return memoryStream.ToArray();
}
```

AzureStorageService.cs (GetTableName Action – This part contains the two closing braces)

```
        }

        5 references
        private static string GetTableName<T>()
        {
            return typeof(T).Name switch
            {
                nameof(Customer) => "Customers",
                nameof(Product) => "Products",
                nameof(Order) => "Orders",
                _ => typeof(T).Name + "s"
            };
        }
    }
}
```