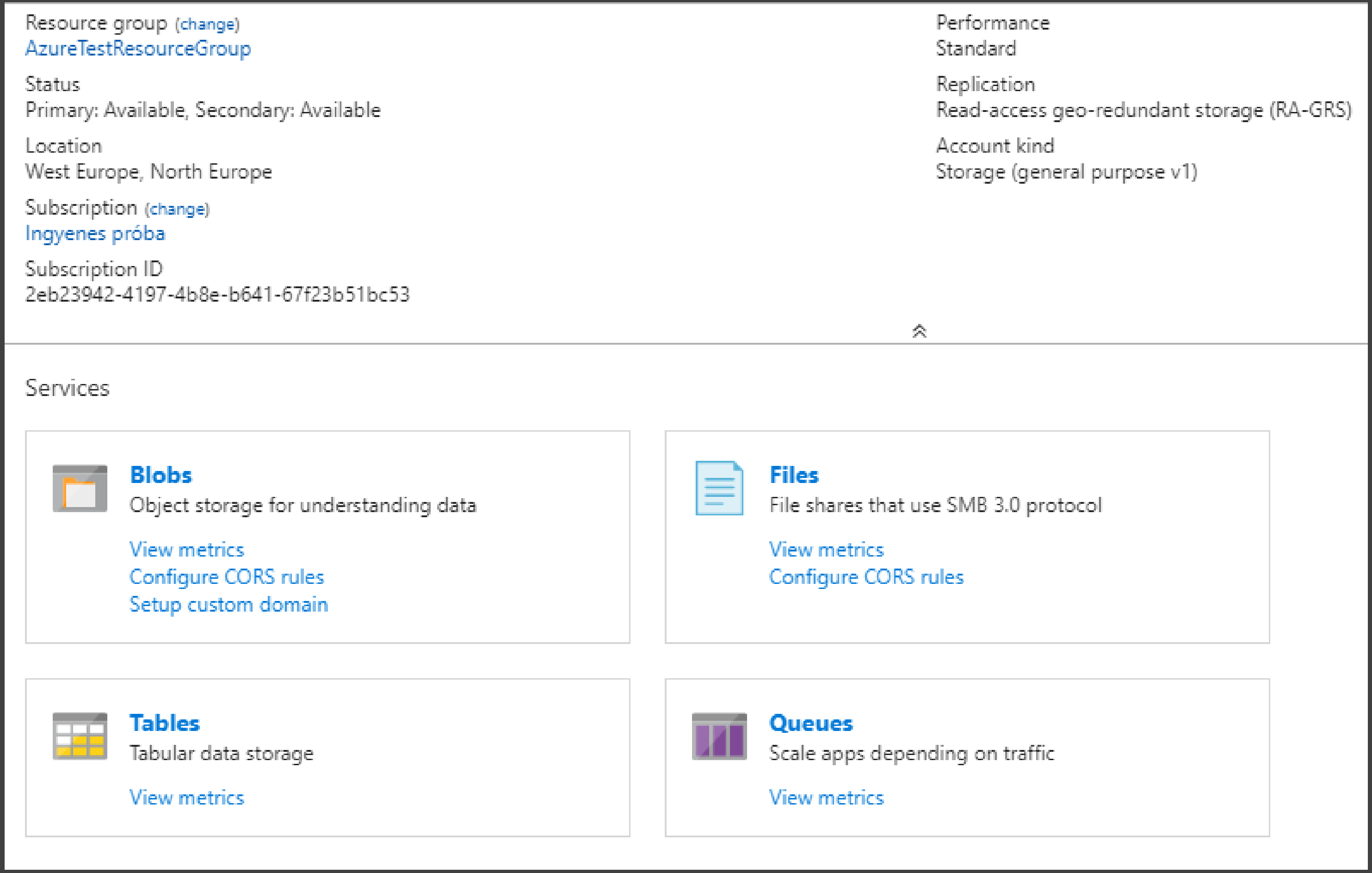
Microsoft Azure – BLOB Storage

Milan Pekardy

pekardy@dcs.uni-pannon.hu

Create storage account

- Storage accounts > Add



Resource group (change)
AzureTestResourceGroup

Status
Primary: Available, Secondary: Available

Location
West Europe, North Europe

Subscription (change)
Ingyenes próba

Subscription ID
2eb23942-4197-4b8e-b641-67f23b51bc53

Performance
Standard

Replication
Read-access geo-redundant storage (RA-GRS)

Account kind
Storage (general purpose v1)

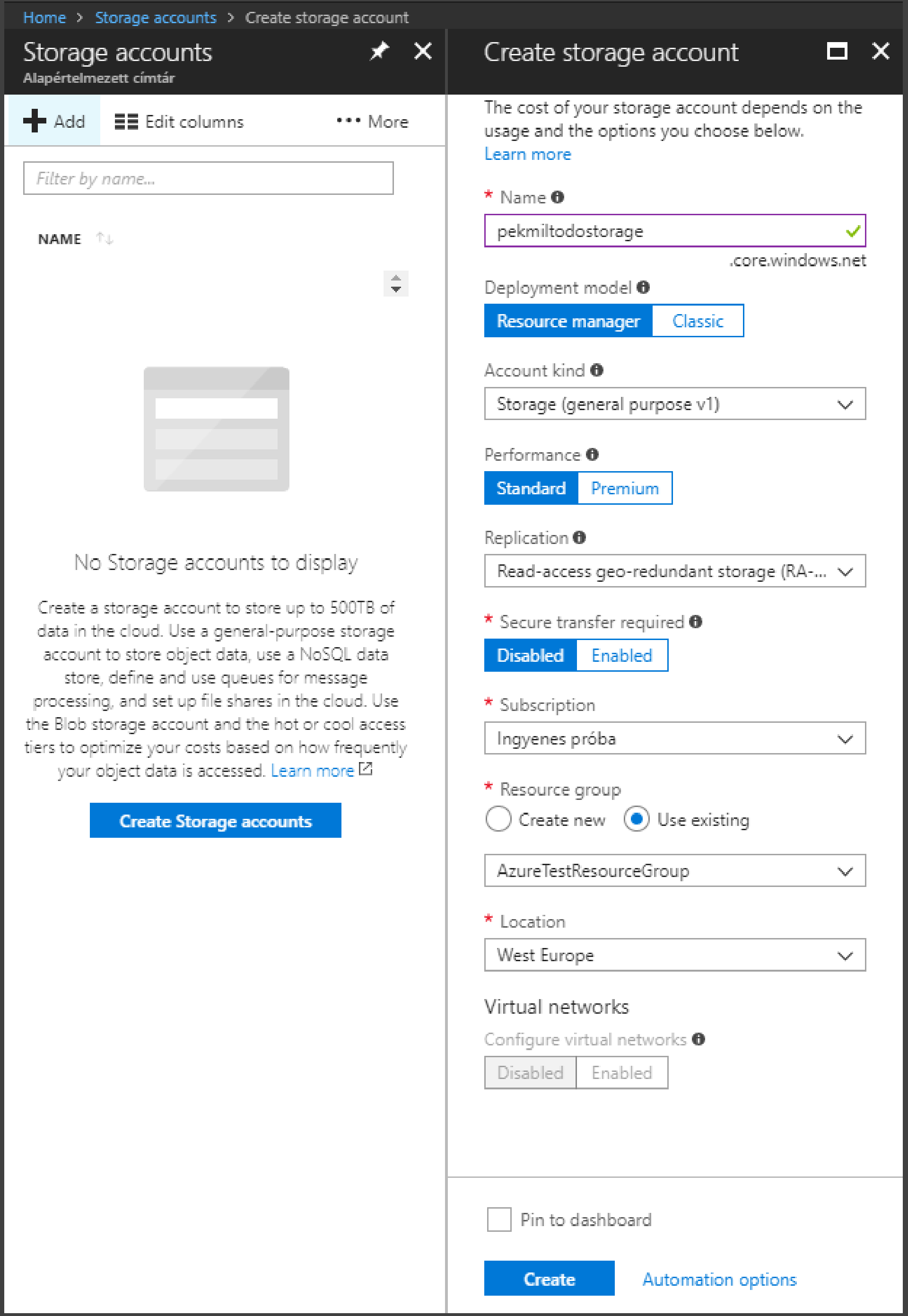
Services

Blobs
Object storage for understanding data
View metrics
Configure CORS rules
Setup custom domain

Files
File shares that use SMB 3.0 protocol
View metrics
Configure CORS rules

Tables
Tabular data storage
View metrics

Queues
Scale apps depending on traffic
View metrics



Home > Storage accounts > Create storage account

Storage accounts
Alapértelmezett címtár

No Storage accounts to display

Create a storage account to store up to 500TB of data in the cloud. Use a general-purpose storage account to store object data, use a NoSQL data store, define and use queues for message processing, and set up file shares in the cloud. Use the Blob storage account and the hot or cool access tiers to optimize your costs based on how frequently your object data is accessed. Learn more

Create storage account

The cost of your storage account depends on the usage and the options you choose below.
Learn more

* Name: pekmiltodostorage .core.windows.net

Deployment model: [✓ Resource manager] Classic

Account kind: Storage (general purpose v1)

Performance: [✓ Standard] Premium

Replication: Read-access geo-redundant storage (RA-...

* Secure transfer required: [✓ Disabled] Enabled

* Subscription: Ingyenes próba

* Resource group: Create new [✓ Use existing]
AzureTestResourceGroup

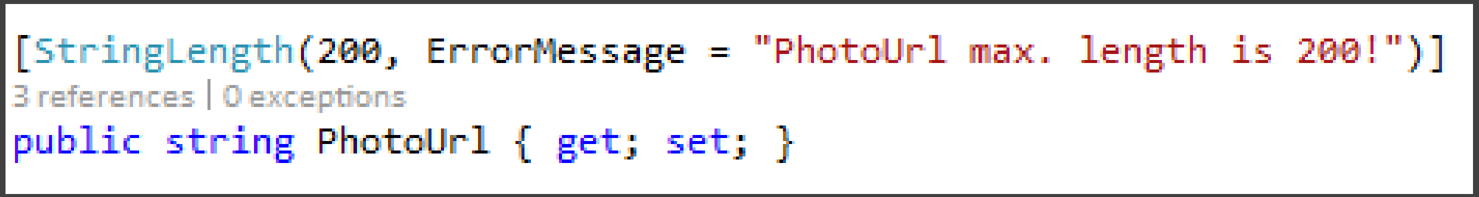
* Location: West Europe

Virtual networks
Configure virtual networks: Disabled Enabled

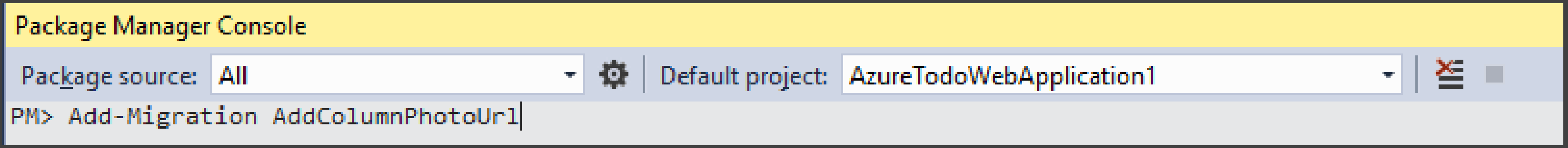
Pin to dashboard

Exercise 1

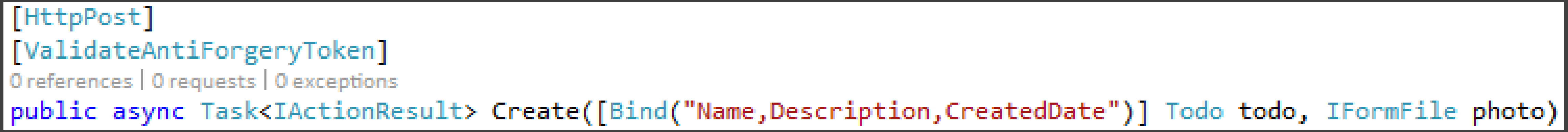
- Store a photo to a Todo entity
  - Modify the entity object (PhotoUrl)
  - Run Data Migrations
  - Modify the TodosController
  - Modify the Views



```
[StringLength(200, ErrorMessage = "PhotoUrl max. length is 200!")]
3 references | 0 exceptions
public string PhotoUrl { get; set; }
```



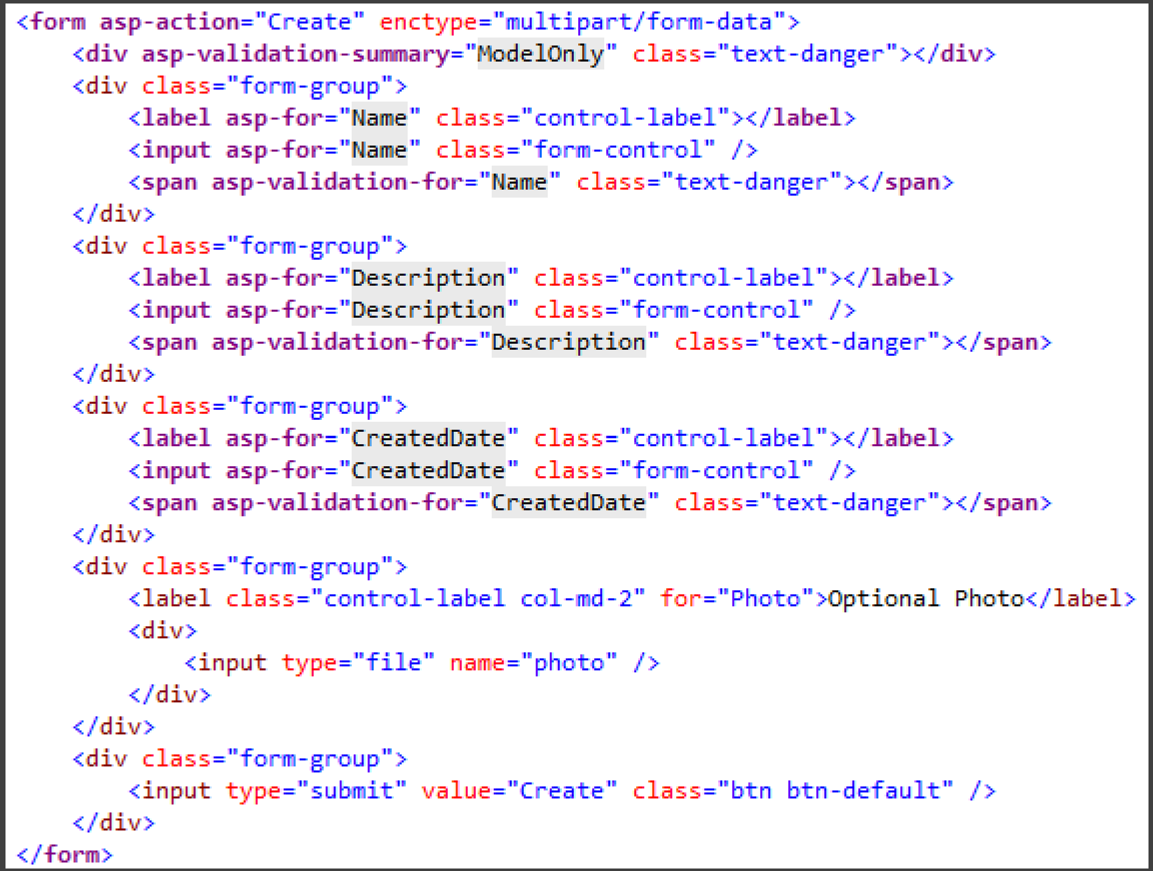
```
PM> Add-Migration AddColumnPhotoUrl
```



```
[HttpPost]
[ValidateAntiForgeryToken]
0 references | 0 requests | 0 exceptions
public async Task<IActionResult> Create([Bind("Name,Description,CreatedDate")] Todo todo, IFormFile photo)
```

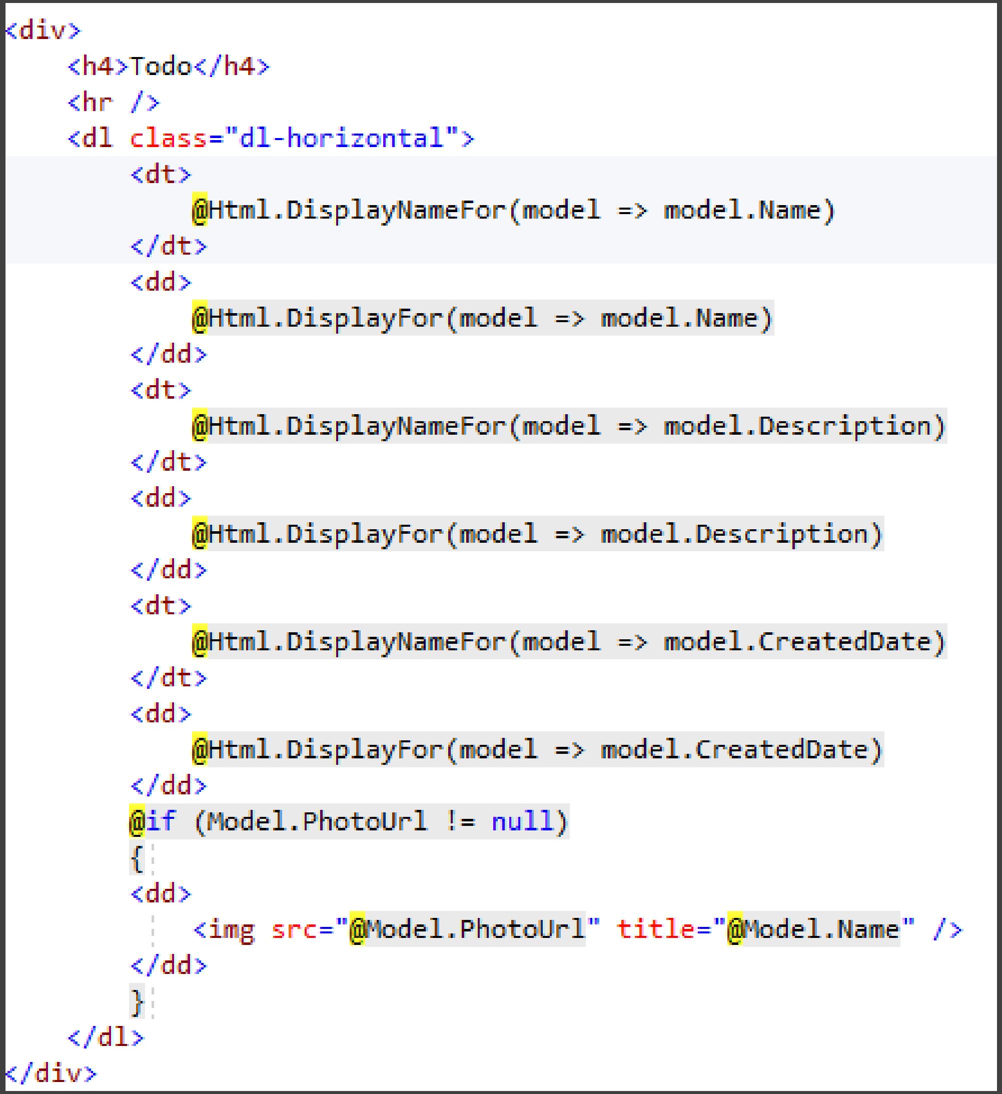
Exercise 1

- Create



```
<form asp-action="Create" enctype="multipart/form-data">
    <div asp-validation-summary="ModelOnly" class="text-danger"></div>
    <div class="form-group">
        <label asp-for="Name" class="control-label"></label>
        <input asp-for="Name" class="form-control" />
        <span asp-validation-for="Name" class="text-danger"></span>
    </div>
    <div class="form-group">
        <label asp-for="Description" class="control-label"></label>
        <input asp-for="Description" class="form-control" />
        <span asp-validation-for="Description" class="text-danger"></span>
    </div>
    <div class="form-group">
        <label asp-for="CreatedDate" class="control-label"></label>
        <input asp-for="CreatedDate" class="form-control" />
        <span asp-validation-for="CreatedDate" class="text-danger"></span>
    </div>
    <div class="form-group">
        <label class="control-label col-md-2" for="Photo">Optional Photo</label>
        <div>
            <input type="file" name="photo" />
        </div>
    </div>
    <div class="form-group">
        <input type="submit" value="Create" class="btn btn-default" />
    </div>
</form>
```

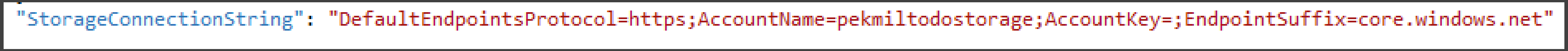
- Details



```
<div>
    <h4>Todo</h4>
    <hr />
    <dl class="dl-horizontal">
        <dt>
            @Html.DisplayNameFor(model => model.Name)
        </dt>
        <dd>
            @Html.DisplayFor(model => model.Name)
        </dd>
        <dt>
            @Html.DisplayNameFor(model => model.Description)
        </dt>
        <dd>
            @Html.DisplayFor(model => model.Description)
        </dd>
        <dt>
            @Html.DisplayNameFor(model => model.CreatedDate)
        </dt>
        <dd>
            @Html.DisplayFor(model => model.CreatedDate)
        </dd>
        @if (Model.PhotoUrl != null)
        {
        <dd>
            <img src="@Model.PhotoUrl" title="@Model.Name" />
        </dd>
        }
    </dl>
</div>
```

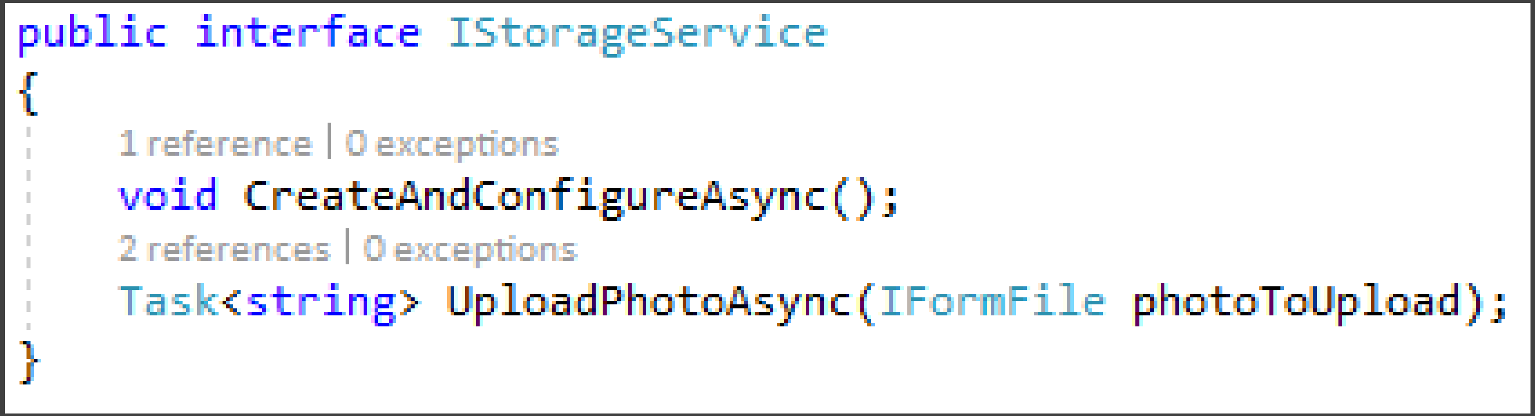
Exercise 2

- Use the storage to store the uploaded images:
  - Get the storage connections string: Storage accounts > Settings > Access keys
  - Save it in the appsettings.json file



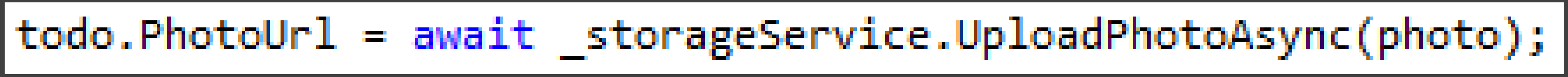
```
"StorageConnectionString": "DefaultEndpointsProtocol=https;AccountName=pekmiltodostorage;AccountKey=;EndpointSuffix=core.windows.net"
```

- Create a storage service interface and implementation



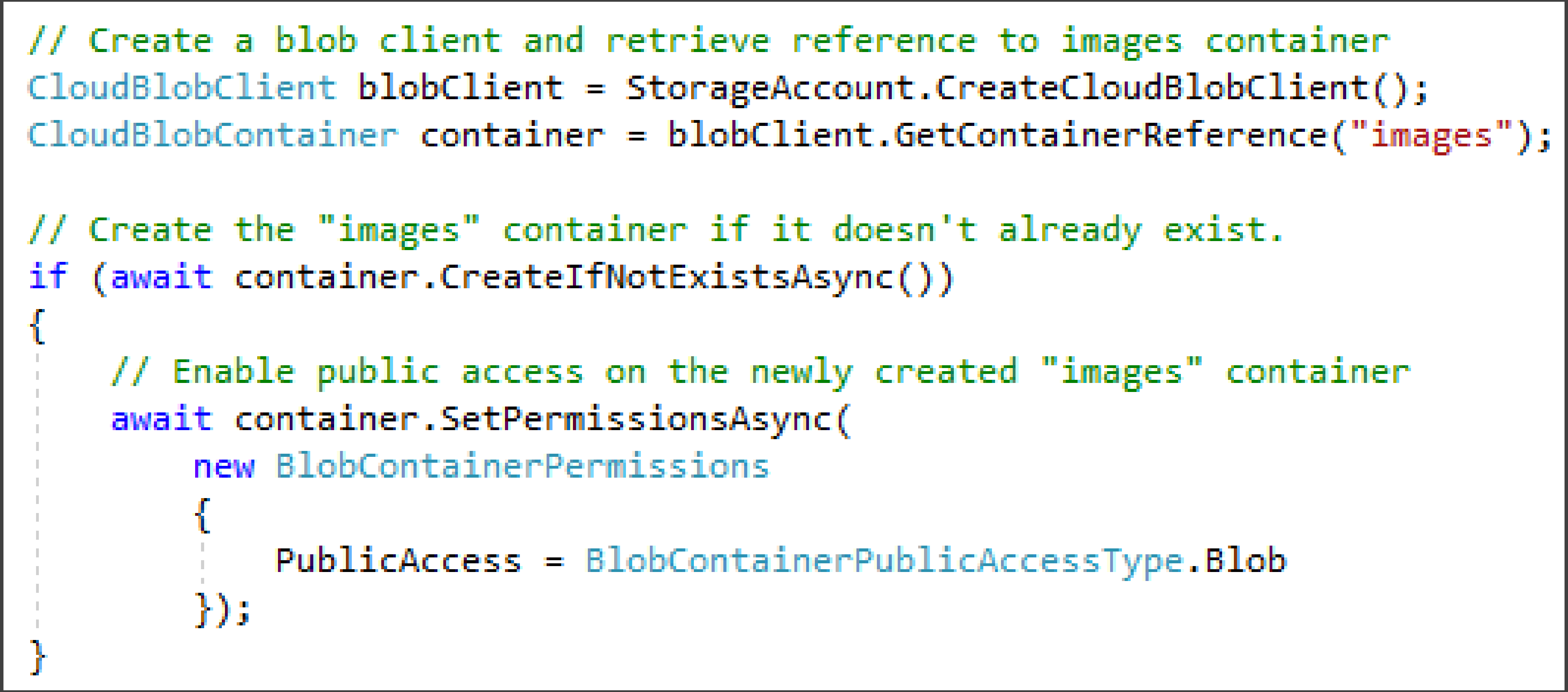
```
public interface IStorageService
{
    1 reference | 0 exceptions
    void CreateAndConfigureAsync();
    2 references | 0 exceptions
    Task<string> UploadPhotoAsync(IFormFile photoToUpload);
}
```

- Use the service in the Create method of the Controller



```
todo.PhotoUrl = await _storageService.UploadPhotoAsync(photo);
```

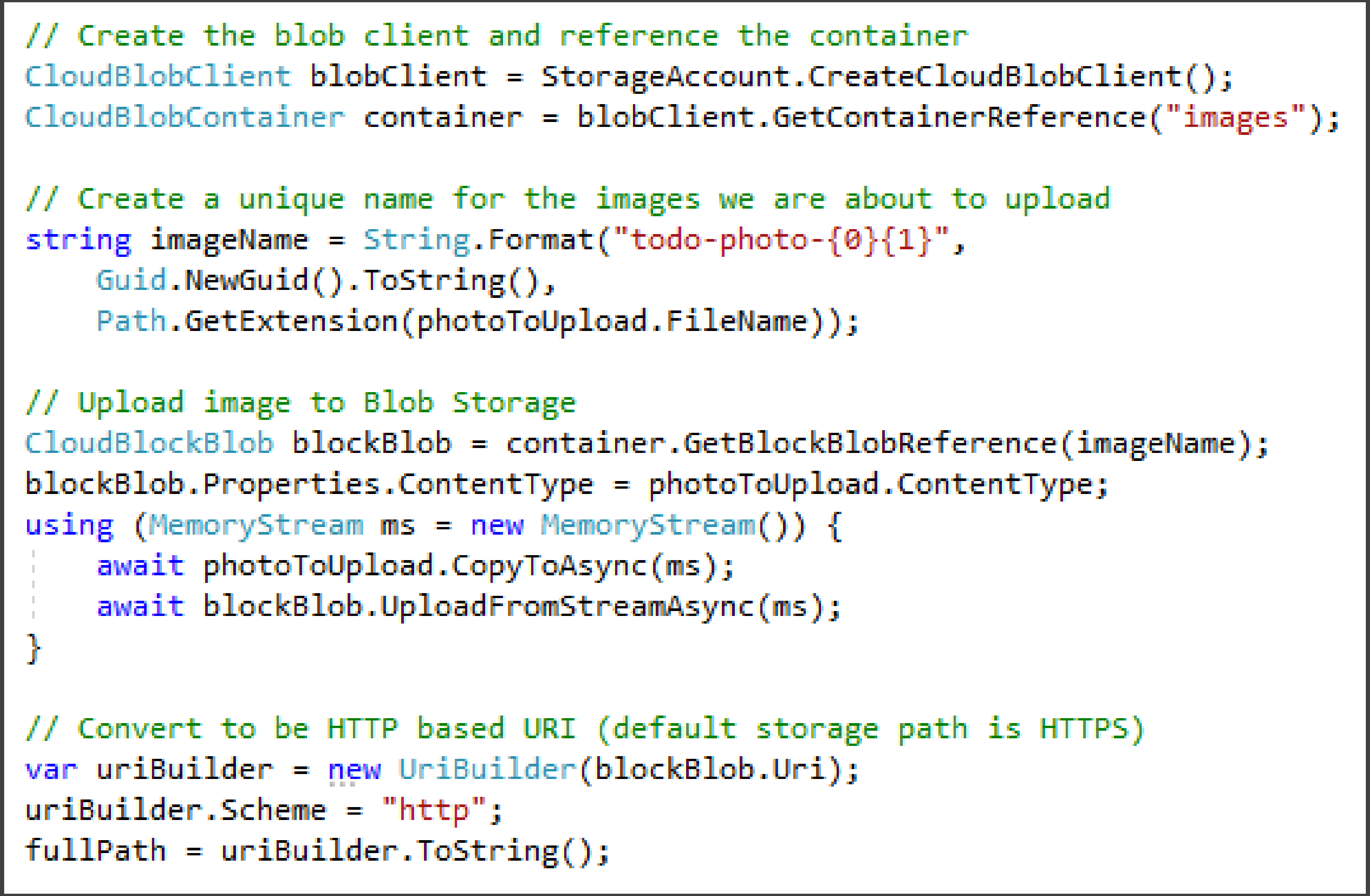
Exercise 2 - CreateAndConfigureAsync



```
// Create a blob client and retrieve reference to images container
CloudBlobClient blobClient = StorageAccount.CreateCloudBlobClient();
CloudBlobContainer container = blobClient.GetContainerReference("images");

// Create the "images" container if it doesn't already exist.
if (await container.CreateIfNotExistsAsync())
{
    // Enable public access on the newly created "images" container
    await container.SetPermissionsAsync(
        new BlobContainerPermissions
        {
            PublicAccess = BlobContainerPublicAccessType.Blob
        });
}
```

Exercise 2 - UploadPhotoAsync



```
// Create the blob client and reference the container
CloudBlobClient blobClient = StorageAccount.CreateCloudBlobClient();
CloudBlobContainer container = blobClient.GetContainerReference("images");

// Create a unique name for the images we are about to upload
string imageName = String.Format("todo-photo-{0}{1}",
    Guid.NewGuid().ToString(),
    Path.GetExtension(photoToUpload.FileName));

// Upload image to Blob Storage
CloudBlockBlob blockBlob = container.GetBlockBlobReference(imageName);
blockBlob.Properties.ContentType = photoToUpload.ContentType;
using (MemoryStream ms = new MemoryStream()) {
    await photoToUpload.CopyToAsync(ms);
    await blockBlob.UploadFromStreamAsync(ms);
}

// Convert to be HTTP based URI (default storage path is HTTPS)
var uriBuilder = new UriBuilder(blockBlob.Uri);
uriBuilder.Scheme = "http";
fullPath = uriBuilder.ToString();
```

Exercise 3 – Delete photos

- Modify the Edit page so that the user can delete the attached photo (if one exists)
  - Implement the necessary functions in the controller and storage service
- Modify the Delete functionality: if the Todo has a photo attached then delete the photo from the storage