(Demo prep)

Implementing the Catalog repository

1. Before implementing our repository, we will need to decide which his going to be the class that represents the objects that will be managed by the repository and that will make their way to our database. The class we use for this should not be confused with our DTOs because we want to have the freedom of updating how we store the items in the database at any given point, regardless of the contract that we need to honor with our service clients.

   We will then give the name of "Entities" to the classes that our repository use.

2. Add **Entities** directory

3. Add Item.cs:

```
public class Item
{
    public Guid Id { get; set; }

    public string Name { get; set; }

    public string Description { get; set; }

    public decimal Price { get; set; }

    public DateTimeOffset CreatedDate { get; set; }
}
```

4. dotnet add package MongoDB.Driver

5. Add Repositories folder

6. Add ItemsRepository.cs to Repositories folder

7. Close navigation pane

8. Implement ItemsRepository

```
public class ItemsRepository
{
    private const string collectionName = "items";
    private readonly IMongoCollection<Item> dbCollection;
    private readonly FilterDefinitionBuilder<Item> filterBuilder = Builders<Item>.Filter;

    public ItemsRepository()
    {
        var mongoClient = new MongoClient($"mongodb://localhost:27017");
        var database = mongoClient.GetDatabase("Catalog");
        dbCollection = database.GetCollection<Item>(collectionName);
    }

    // Now, for our repository public methods we will be using the Asynchronous programming
    model. This will avoid performance bottlenecks and enhance the overall responsiveness of our
    service.

    // To take advantage of this model all of our methods will become asynchronous by returning
    async Task and by using the await keyword any time they interact with the database. We will
    also use the Async suffix on all the methods to surface the fact that the methods are
    asynchronous.

    public async Task<IReadOnlyCollection<Item>> GetAllAsync()
    {
        return await dbCollection.Find(filterBuilder.Empty).ToListAsync();
    }

    public async Task<Item> GetAsync(Guid id)
    {
        FilterDefinition<Item> filter = filterBuilder.Eq(entity => entity.Id, id);
        return await dbCollection.Find(filter).FirstOrDefaultAsync();
    }

    public async Task CreateAsync(Item entity)
    {
        if (entity == null)
        {
            throw new ArgumentNullException(nameof(entity));
        }

        await dbCollection.InsertOneAsync(entity);
    }

    public async Task UpdateAsync(Item entity)
```

```
    {
        if (entity == null)
        {
            throw new ArgumentNullException(nameof(entity));
        }

        FilterDefinition<Item> filter = filterBuilder.Eq(existingEntity => existingEntity.Id, entity.Id);
        await dbCollection.ReplaceOneAsync(filter, entity);
    }

    public async Task RemoveAsync(Guid id)
    {
        FilterDefinition<Item> filter = filterBuilder.Eq(entity => entity.Id, id);
        await dbCollection.DeleteOneAsync(filter);
    }
}
```

Our ItemsRepository is ready. In the next lesson we will integrate it with our existing ItemsController so that it can start querying items from MongoDB.