(Demo prep)

Implementing dependency injection and configuration

1. In order to start using dependency injection in our microservice we will need to start using interfaces as opposed to concrete types when constructing dependencies. Let’s start by updating ItemsRepository so that it implements an interface that we can use in ItemsController.

2. Extract interface from ItemsRepository

3. Inject IItemsRepository to ItemsController:

```
private readonly IItemsRepository itemsRepository;

public ItemsController(IItemsRepository itemsRepository)
{
    this.itemsRepository = itemsRepository;
}
```

4. Update ItemsRepository constructor:

```
public ItemsRepository(IMongoDatabase database)
{
    dbCollection = database.GetCollection<Item>(collectionName);
}
```

5. Update appsettings.json:

```
"ServiceSettings": {
  "ServiceName": "Catalog"
},
"MongoDbSettings": {
  "Host": "localhost",
  "Port": "27017"
},
```

6. Add the Settings directory

7. Add MongoDBSettings.cs (uses Expression body definition):

```
public class MongoDbSettings
{
    public string Host { get; init; }
```

```
        public int Port { get; init; }
        public string ConnectionString => $"mongodb://{Host}:{Port}";
    }
```

8. Add ServiceSettings.cs:

```
    public class ServiceSettings
    {
        public string ServiceName { get; init; }
    }
```

9. Update Startup.cs (**collapse navigation pane**):

```
private ServiceSettings serviceSettings;

public Startup(IConfiguration configuration)
{
    Configuration = configuration;
}

public IConfiguration Configuration { get; }

// This method gets called by the runtime. Use this method to add services to the container.
public void ConfigureServices(IServiceCollection services)
{
    BsonSerializer.RegisterSerializer(new GuidSerializer(BsonType.String));
    BsonSerializer.RegisterSerializer(new DateTimeOffsetSerializer(BsonType.String));

    serviceSettings = Configuration.GetSection(nameof(ServiceSettings)).Get<ServiceSettings>();

    services.AddSingleton(serviceProvider =>
    {
        var mongoDbSettings =
Configuration.GetSection(nameof(MongoDbSettings)).Get<MongoDbSettings>();
        var mongoClient = new MongoClient(mongoDbSettings.ConnectionString);
        return mongoClient.GetDatabase(serviceSettings.ServiceName);
    });

    services.AddSingleton<IItemsRepository, ItemsRepository>();
```

10. Place breakpoints at:
    - ItemsController constructor
    - ItemsRepository constructor
    - Startup → AddSingleton for IMongoDatabase

11. Run the service

12. Query for items and show dependency injection and configuration in action

So, we have reached the end of this module with a fully working, decoupled and configurable microservice that is able to store and query for our catalog items in a MongoDB database. In the next module we will introduce our second microservice and will learn about the multiple challenges of having microservices talk to each other.