Today we will be creating an animal shelter “Dotnetimals”, that sells lost and abandoned cats!

Exercise 1, Setting up The Project

Create a new web api project (`dotnet new webapi`).

Exercise 2, Creating Entities

Inside your newly created project, create a *Data* folder. Inside the Data folder, create an *Entities* folder that holds the entities of our animal shelter. Since the cats should be sellable (at a cheap price) for anyone interested, and that people can order multiple cats, the entities should be as follows:

`Cat`
int Id, string Name, string Color, decimal Price, DateTime Birthdate, string FavoriteDish

`Order`
int Id, DateTime OrderDate, string OrderNumber, ICollection<Cat> Cats

Exercise 3, Applying Data Annotations

Apply data annotiations to the entities above.

- The name of a cat should be no longer than 20 characters long, and at minimum 2 characters long.
- The price of a cat must be a range from 0-100 and of datatype “currency”.
- The color of a cat must only contain letters (i.e. no white-space, numbers or special characters).
- The name and price of a cat are required.

Consider: Even though the price of the cat is required, do you have to specify it explicitely using a data annotation? Why/why not? In relation to this, how do you make a value type optional? (It has to do with nullable types)

Hint: For the color of the cat to only contain letters, use the Regular expression `@"^[A-Z]+[a-zA-Z""'\s-]*$"`.

Exercise 4, Creating a DbContext

Having created your entities, it is now time to create the `DbContext` that represents our data!

Inside the Data folder, create a `CatContext` class that is derived from the `DbContext` class. This class should contain `DbSets` of both type `Cat` and `Order`. Name the `DbSets` according to the plural of the entities they represent (i.e. `Cats` and `Orders`).

Create a constructor for the `CatContext` that takes one options argument of type `DbContextOptions<CatContext>`. Remember to call the base class with the options argument.

Exercise 5, Deciding on a Database Provider

We need to decide on what database provider to use. SQLite is fairly straight forward. You could also try SQL Server Express.

Install the package for the database provider that you want to target.

Next, we need to install an application with which we can manage our database.

This could be JetBrains' Datagrip, or you could install SQL Server Management Studio (or DB Browser for SQLite if you are using SQLite).

Exercise 6, Configuring the DbContext

Before we can use our `DbContext`, we need to configure it.

In the `ConfigureServices` method inside *Startup.cs*, add the `DbContext` as a service so that it can be used anywhere in our application using dependency injection. Use the `UseSqlite` or `UseSqlServer` method to configure the connection. (the connection string should be stored in *appsettings.json*)

Exercise 7, EF Tooling and Migrations

Add an initial migration using `dotnet ef migrations add [migration name]`. This adds migration files to your project according to the current database schema and the entities in your code.

Next, call `dotnet ef database update` from the command line. This updates your database according to the migration files.

Have a look at your database, through on of the previously mentioned tools. Does your table data look as expected?

Consider: Have a look at the generated migration files. Can you make out where the files apply and revert migrations?

Exercise 8, Seeding the Database

In the Data folder, create a `DbInitializer` class that seeds the database with some cats.

Exercise 9, Using the DbContext

Create a `CatsController` for your API that handles requests to the /cats endpoint. In the controller, query your database using the `DbContext` and LINQ syntax.

To be able to handle orders, create an `OrdersController` for your API that handles requests to the /orders endpoint.

Hint: Remember you can use dependency injection to retrieve an instance of the `DbContext` in your controllers.

Exercise 10, Using The Repository Pattern (optional)

Refactor your code to take advantage of the repository pattern.

Exercise 11, Creating a Client for your API

Now that our Web API is in place, we can create a client that uses the API so that the homeless cats can finally get a new home!

Create a client that uses the http-endpoints of your dotnetimals api project. The client should display info on the cats that are currently for sale, and have customers be able to create orders on the cats.

Hint: This can be done as a simple console application that takes three commands:

- A command to list all available cats.
- A command to buy a specific cat.
- A command to see all cats that you have bought.

Exercise 12, Working With LINQ

To get some more experience with LINQ, go through the official LINQ tutorial on docs.microsoft.com.

Exercise 13, Storing data as documents (very optional)

If you want to try out a non-relational solution for storing and accessing data, try to implement the "Dotnetimals" application using the MongoDB document database.

Here are some pointers to get you started:

- https://docs.mongodb.com/manual/tutorial/getting-started/
- https://docs.mongodb.com/ecosystem/drivers/csharp/