# Entity Framework Core: Retake Exam

Exam problems for the [Databases Advanced - Entity Framework course @ SoftUni](https://softuni.bg/courses/databases-advanced-entity-framework).

Download the provided **skeleton** and **use it** in your solutions.

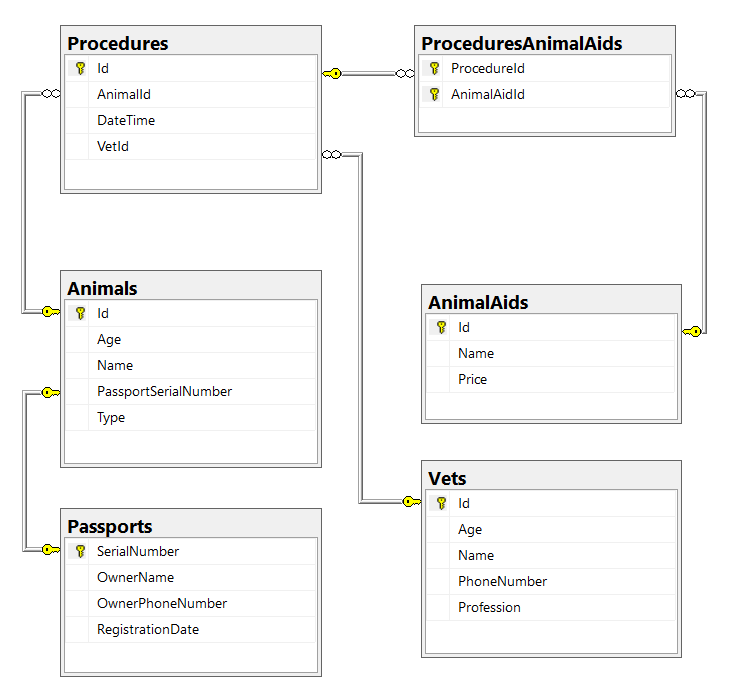
You are **not allowed** to **change the structure** or the names of the provided **classes** and **methods** or **delete them**. **It is very important to use only the pre-installed packages and not change their versions.** Any usage of other packages is at your **own** **risk**.

Submit your project solutions in the SoftUni Judge system, as a **zip** archive file – excluding the **bin** and **obj** folders.

Your task is to create a database console application using **Entity Framework Core** using the **Code First** approach. Design the **models** and **methods** for manipulating the data, as described below.

# Pet Clinic

Doctor John Dolittle’s clinic is starting to drastically grow. He now has more clients, he has employed doctors and organizing his clinic’s records on paper is becoming difficult. He has contacted you to help him with the changes he wants to make and wants you to design a database to keep track of the processes in the clinic.



# Project Skeleton Overview

You are given a **project skeleton**, which includes the following folders:

* App – contains the Startup class, which is the **entry point of the application**. Also contains an AutoMapperprofile, which you can configure if you choose to use AutoMapper in your app.
* Data – contains the PetClinicContext class and the **connection string**
* Models – contains the **entity classes**
* DataProcessor – contains the Serializer and Deserializerclasses, which are used for **importing** and **exporting** data

## Problem 1. Model Definition (50 pts)

In his clinic, Dr. Dolittle has employed **different kinds of** **doctors**, who serve patients – **animals**. Different **procedures** are performed on an animal, which have a pre-defined **cost**.

Design the following database models:

### Animal

* Id – integer, **Primary Key**
* Name – text with **min length 3** and **max length 20** (**required**)
* Type – text with **min length 3** and **max length 20** (**required**)
* Age – integer, **cannot be negative or 0 (required)**
* PassportSerialNumber ­– string, **foreign key**
* Passport – the **passport** of the animal **(required)**
* Procedures – the **procedures**, performed on the animal

### Passport

* SerialNumber – a string consisting of **exactly** **10 characters, starting with 7 letters** and **ending with 3 digits, Primary Key**
* Animal – the **animal** to which the **passport** is registered **(required)**
* OwnerPhoneNumber – the phone number of the animal’s owner, **required**, make sure it matches **one** of the following requirements:
  + either starts with **+359** and is **followed by** **9 digits**
  + or consists of **exactly** **10** digits, starting with **0**
* OwnerName – the name of the animal’s owner; text with **min length 3** and **max length 30** (**required**)
* RegistrationDate – the **date and time** on which the passport was registered **(required)**

### Vet

* Id – integer, **Primary Key**
* Name – text with **min length 3** and **max length 40** (**required**)
* Profession – text with **min length 3** and **max length 50** (**required**)
* Age – integer number, minimum value of **22** years and maximum **65 (required)**
* PhoneNumber – **required, unique**, make sure it matches **one** of the following requirements:
  + either starts with **+359** and is **followed by** **9 digits**
  + or consists of **exactly** **10** digits, starting with **0**
* Procedures – the **procedures**, performed by the vet

### Procedure

* Id – integer, **Primary Key**
* AnimalId ­– integer, **foreign key**
* Animal – the **animal** on which the procedure is performed **(required)**
* VetId ­– integer, **foreign key**
* Vet – the clinic’s employed doctor servicing the patient **(required)**
* ProcedureAnimalAids – **collection of** type **ProcedureAnimalAid**
* Cost – the **cost of the procedure**, **calculated by summing the price of the different services performed;** does not need to be inserted in the database
* DateTime – the date and time on which the given procedure is performed **(required)**

### AnimalAid

* Id – integer, **Primary Key**
* Name – text with **min length 3** and **max length 30** (**required, unique**)
* Price – decimal (**non-negative, minimum value: 0.01**, **required**)
* **AnimalAidProcedures** – **collection of** type **ProcedureAnimalAid**

### ProcedureAnimalAid

* ProcedureId – integer, **Primary Key**
* Procedure – the animal aid’s **procedure** **(required)**
* AnimalAidId – integer, **Primary Key**
* **AnimalAid** – the procedure’s **animal aid** **(required)**

## Problem 2. Data Import (30 pts)

For the functionality of the application, you need to create several methods that manipulate the database. The **project skeleton** already provides you with these methods, inside the **Deserializer** **class**. Use **Data Transfer Objects** as needed.

Use the provided **JSON** and **XML** files to populate the database with data. Import all the information from those files into the database.

**You are not allowed to modify the provided JSON and XML files.**

### JSON Import

#### Import Animal Aids

Start by importing the least dependent entity – the **different types of animal aid** that the clinic provides. Using the file **animalAids.json**, import the data from that file into the database. Print information about each imported object in the format described below.

##### Constraints

* If any validation errors occur (such as if an animal aid **name** **is too long/short**), **do not import the entity**
* If an animal aid **already** **exists, do not import it**

**If a record does not meet the requirements from the first section, print an error message:**

|  |  |
| --- | --- |
| **Success message** | **Error message** |
| Record {animal aid name} successfully imported. | Error: Invalid data. |

##### Example

|  |
| --- |
| **animalAids.json** |
| [  {  "Name": "Internal Deworming",  "Price": 8.00  },  {  "Name": "Fecal Test",  "Price": 7.50  },  {  "Name": "H3N8",  "Price": 30.00  },  {  "Name": "Nasal Bordetella",  "Price": 5.60  },  {  "Name": "External Deworming",  "Price": -35.00  },  {  "Name": "Bordetella",  "Price": 7.50  **},**  **…**  **]** |
| **Output** |
| Record Internal Deworming successfully imported.  Record Fecal Test successfully imported.  Record H3N8 successfully imported.  Record Nasal Bordetella successfully imported.  Error: Invalid data.  … |

#### Import Animals

Using the file **animals.json**, import the data from that file into the database. Print information about each imported object in the format described below.

##### Constraints

* If any validation errors occur (such as if a name, type or serial number is invalid) do not import the entity
* If a passport with the same serial number **exists, do not import the entity**

**If a record does not meet the requirements from the first section, print an error message:**

|  |  |
| --- | --- |
| **Success message** | **Error message** |
| Record {animal name} Passport №: {passport serial number} successfully imported. | Error: Invalid data. |

##### Example

|  |
| --- |
| **animals.json** |
| [  {  "Name":"Bella",  "Type":"cat",  "Age": 2,  "Passport": {  "SerialNumber": "etyhGgH678",  "OwnerName": "Sheldon Cooper",  "OwnerPhoneNumber": "0897556446",  "RegistrationDate": "12-03-2014"  }  },  {  "Name":"Charlie",  "Type":"cat",  "Age": 3,  "Passport": {  "SerialNumber": "anothev650",  "OwnerName": "Magda Bjork",  "OwnerPhoneNumber": "+35989776512",  "RegistrationDate": "15-04-2015"  }  },  {  "Name":"Chester",  "Type":"dog",  "Age": 11,  "Passport": {  "SerialNumber": "adoggoo451",  "OwnerName": "Adriana Lima",  "OwnerPhoneNumber": "35989776512",  "RegistrationDate": "31-12-2016"  }  },  {  "Name":"Lucy",  "Type":"cat",  "Age": 6,  "Passport": {  "SerialNumber": "acattee321",  "OwnerName": "Ivan Ivanov",  "OwnerPhoneNumber": "0887446123",  "RegistrationDate": "10-06-2015"  }  **},**  **…**  **]** |
| **Output** |
| Record Bella Passport №: etyhGgH678 successfully imported.  Error: Invalid data.  Error: Invalid data.  Record Lucy Passport №: acattee321 successfully imported.  … |

### XML Import

#### Import Vets

The next key figure in our app are the vets who take care of the patients. The info about them is given in the **vets.xml** file

##### Constraints

* Validate each row of information according to the first section. If the validation fails, **do not import the vet**
* If a vet with the same **phone** **number** already exists, **do not import the vet**

**If a record does not meet the requirements from the first section, print an error message:**

|  |  |
| --- | --- |
| **Success message** | **Error message** |
| Record {vet name} successfully imported. | Error: Invalid data. |

**Example:**

|  |
| --- |
| **vets.xml** |
| <?xml version="1.0" encoding="UTF-8"?>  <Vets>  <Vet>  <Name>Michael Jordan</Name>  <Profession>Emergency and Critical Care</Profession>  <Age>45</Age>  <PhoneNumber>0897665544</PhoneNumber>  </Vet>  <Vet>  <Name>Melanie Bennington</Name>  <Profession>Surgery</Profession>  <Age>21</Age>  <PhoneNumber>+359284566778</PhoneNumber>  </Vet>  <Vet>  <Name>Edmond Halley</Name>  <Profession>Veterinary Nursing</Profession>  <Age>24</Age>  <PhoneNumber>+359284566778</PhoneNumber>  </Vet>  <Vet>  <Name>Niels Bohr</Name>  <Profession>Internal Medicine</Profession>  <Age>32</Age>  <PhoneNumber>0879557712</PhoneNumber>  </Vet>  <Vet>  <Name>Werner Heisenberg</Name>  <Profession>Pediatrics, Genetics and Reproduction</Profession>  <Age>55</Age>  <PhoneNumber>0879535712</PhoneNumber>  </Vet>…  </Vets> |
| **Output** |
| Record Michael Jordan successfully imported.  Error: Invalid data.  Record Edmond Halley successfully imported.  Record Niels Bohr successfully imported.  Record Werner Heisenberg successfully imported.  … |

#### Import Procedures

Now it's time to import the records of the procedures done on the animals. Parse the information from the **procedures.xml** file.

##### Constraints

* Do not import a procedure if:
  + A **vet** with such **name** **does** **not** **exist**
  + An **animal** with given **serial** **number** **does** **not** **exist**
  + An **animal** **aid** with given **name** **does** **not** **exist**
  + The **same** **animal** **aid** is given **more** **than** **once** in the **same** **procedure**

**If a record does not meet the requirements and constraits, print an error message:**

|  |  |
| --- | --- |
| **Success message** | **Error message** |
| Record successfully imported. | Error: Invalid data. |

**Example:**

|  |
| --- |
| **procedures.xml** |
| <?xml version="1.0" encoding="UTF-8"?>  <Procedures>  <Procedure>  <Vet>Niels Bohr</Vet>  <Animal>acattee321</Animal>  <DateTime>14-01-2016</DateTime>  <AnimalAids>  <AnimalAid>  <Name>Nasal Bordetella</Name>  </AnimalAid>  <AnimalAid>  <Name>Internal Deworming</Name>  </AnimalAid>  <AnimalAid>  <Name>Fecal Test</Name>  </AnimalAid>  </AnimalAids>  </Procedure>  <Procedure>  <Vet>Jennifer Evans</Vet>  <Animal>bernied355</Animal>  <DateTime>15-04-2016</DateTime>  <AnimalAids>  <AnimalAid>  <Name>Lyme Test</Name>  </AnimalAid>  <AnimalAid>  <Name>Fecal Test</Name>  </AnimalAid>  </AnimalAids>  </Procedure>  <Procedure>  <Vet>Michael Jordan</Vet>  <Animal>barkeer355</Animal>  <DateTime>03-02-2016</DateTime>  <AnimalAids>  <AnimalAid>  <Name>Injectable Bordetella</Name>  </AnimalAid>  <AnimalAid>  <Name>Canine Heartworm Test</Name>  </AnimalAid>  </AnimalAids>  **</Procedure>**  **…**  </Procedures> |
| **Output** |
| Record successfully imported.  Record successfully imported.  Record successfully imported.  Record successfully imported.  … |

## Problem 2. Data Export (20 pts)

### JSON Export

Your task is to write logic in the provided **ExportAnimalsByOwnerPhoneNumber** method in the **Serializer** **class**. Export all **animals** by their **owner's** **number** sorted by **age** **ascending**, then by **serial** **number** **alphabetically**.

**Export dates in the format "dd-MM-yyyy", using** CultureInfo.InvariantCulture**!**

#### Example:

|  |
| --- |
| ExportAnimalsByOwnerPhoneNumber(context, "0887446123") |
| **[**  **{**  **"OwnerName": "Ivan Ivanov",**  **"AnimalName": "Jessy",**  **"Age": 3,**  **"SerialNumber": "jessiii355",**  **"RegisteredOn": "05-11-2015"**  **},**  **{**  **"OwnerName": "Ivan Ivanov",**  **"AnimalName": "Lucy",**  **"Age": 6,**  **"SerialNumber": "acattee321",**  **"RegisteredOn": "10-06-2015"**  **}**  ] |

### XML Export

Implement the provided **ExportAllProcedures** method in the **Serializer**. Export all **procedures**: for each **procedure**, export the **animal’s** **serial** **number**, the **owner’s** **phone** **number**, the **date** of the **procedure** and the **names** and **prices** of the **animal** **aids** performed. In the end, export the **total** **price** of the **procedure**.

**Order the procedures by date ascending, then by passport serial number alphabetically.**

**Export dates in the format "dd-MM-yyyy", using** CultureInfo.InvariantCulture**!**

|  |
| --- |
| ExportProcedures(context) |
| <Procedures>  <Procedure>  <Passport>acattee321</Passport>  <OwnerNumber>0887446123</OwnerNumber>  <DateTime>14-01-2016</DateTime>  <AnimalAids>  <AnimalAid>  <Name>Internal Deworming</Name>  <Price>8.00</Price>  </AnimalAid>  <AnimalAid>  <Name>Fecal Test</Name>  <Price>7.50</Price>  </AnimalAid>  <AnimalAid>  <Name>Nasal Bordetella</Name>  <Price>5.60</Price>  </AnimalAid>  </AnimalAids>  <TotalPrice>21.10</TotalPrice>  </Procedure>  <Procedure>  <Passport>kljsdfk325</Passport>  <OwnerNumber>0899446676</OwnerNumber>  <DateTime>19-01-2016</DateTime>  <AnimalAids>  <AnimalAid>  <Name>H3N2</Name>  <Price>28.00</Price>  </AnimalAid>  <AnimalAid>  <Name>Lyme Test</Name>  <Price>15.00</Price>  </AnimalAid>  </AnimalAids>  <TotalPrice>43.00</TotalPrice>  </Procedure>  …  </Procedures> |

## Problem 4. Bonus Task (10 pts)

**Implement the bonus method in the** PetClinic.DataProcessor **project for an additional amount of points**.

### Update Vet Profession

Implement the method DataProcessor.Bonus.UpdateVetProfession, which receives a vet’s **phone number** and a **new** **profession**. Your task is to **find the vet** by that phone number and **update their profession**.

After the profession is updated, return the message “{vet.Name}'s profession updated from {oldProfession} to {newProfession}.”.

If the vet is not found, return “Vet with phone number {phoneNumber} not found!”

#### Examples

|  |
| --- |
| DataProcessor.Bonus.UpdateVetProfession(context, "+359284566778", "Primary Care") |
| Edmond Halley's profession updated from Veterinary Nursing to Primary Care. |

|  |
| --- |
| DataProcessor.Bonus.UpdateVetProfession(context, "+359887123456", "Primary Care") |
| Vet with phone number **+**359887123456 not found! |