# Entity Framework Core Regular Exam - 9 August 2024

Exam problems for the [Databases Advanced - Entity Framework course @ SoftUni](https://softuni.bg/trainings/4540/entity-framework-core-june-2024).  
Submit your solutions in the **SoftUni Judge** system (delete all **bin**/**obj** and **packages** folders) [here](https://judge.softuni.org/Contests/4822/Entity-Framework-Core-Retake-Exam-9-August-2024).

Before submitting your solutions in the **SoftUni Judge** system, delete all **bin**/**obj** and **packages** folders. If the **zip** file is still too large, you can delete the **ImportResults**, **ExportsResults** and **Datasets** folders too.

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

# NetPay



## Project Skeleton Overview

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

1. Data – contains the NetPayContext class, Models folder, which contains the **entity classes** and the **Configuration** class with the **connection string**
2. DataProcessor – contains the Serializer and Deserializerclasses, which are used for **importing** and **exporting** data
3. Datasets – contains the .json and .xml files for the import part
4. ImportResults – contains the **import** results you make in the Deserializer class
5. ExportResults – contains the **export** results you make in the Serializer class

## Model Definition (60 pts)

The application needs to store the following data:

### Household

* Id– integer, **Primary Key**
* ContactPerson **- text** with length **[5, 50] (required)**
* Email– **text** with length **[6, 80] (not required)**
* PhoneNumber – **text** with **length** **15. (required)**
  + The phone number must **start with a plus sign**, followed by **exactly three digits** for the country code, a **slash**, **exactly three digits** for the area or service provider code, a **dash**, and **exactly six digits** for the subscriber number:
    - Example -> **+144/123-123456**
    - **Use** the following string **for** correct **validation**: @**"**^\+\d{3}/\d{3}-\d{6}$**"**
* Expenses - a collection of type Expense

### Expense

* Id– integer, **Primary Key**
* ExpenseName **– text** with length **[5, 50] (required)**
* Amount **-** a decimal value in the range [0.01, 100 000](required)
* DueDate **-** DateTime **(required)**
* PaymentStatus **– PaymentStatus enum (Paid = 1, Unpaid, Overdue, Expired) (required)**
* HouseholdId - integer, foreign key **(required)**
* Household – Household
* ServiceId - integer, foreign key **(required)**
* Service - Service

### Service

* Id– integer, **Primary Key**
* ServiceName **– text** with length **[5, 30] (required)**
* Expenses **-** acollection of type Expense
* SuppliersServices **-** collection of type SupplierService

### Supplier

* Id– integer, **Primary Key**
* SupplierName– **text** with length **[3, 60]** (**required**)
* SuppliersServices **-** collection of type SupplierService

### SupplierService

* SupplierId– integer, Primary Key, foreign key (required)
* Supplier– Supplier
* ServiceId– integer, Primary Key, foreign key (required)
* Service – Service

## Data Import (20pts)

To ensure the application's functionality, it is essential to **populate the database with initial data**. Inside the **DbContext class**, you will find a **commented-out section** specifically designed for seeding data.   
**Before applying migrations** and updating the database, please **uncomment this section**.

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. Usage of DataTransferObjects or **AutoMapper** is **optional**.

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

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

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

|  |
| --- |
| **Error message** |
| Invalid data format! |

**If some data appears to be duplicated, do not import the entity, print a duplication data message:**

|  |
| --- |
| **Error message** |
| Error! Data duplicated. |

### XML Import

#### Import Households

Using the file "**households.xml"**, **import the data from the file** into the database.

Each imported **household should be validated** and **added to the database if it meets the specified criteria**. The method should **return a string containing information about each import attempt**, formatted as described.

##### Constraints

* **Validation of Households Entities** - Each household entity must be validated against the following criteria:
  + **ContactPerson** – Must meet the constraints for the property, described above
  + **Email** – Must meet the constraints for the property, described above
  + **PhoneNumber** - Must meet the constraints for the property, described above
* If **any validation error occurs** for a household entity, the **entity should not be imported**, and an   
  **error message should be appended** to the method's output.
* **Duplication Check** - Before adding an entity to the database**,   
  ensure there are no existing records with the same**:
  + **ContactPerson** OR **Email** OR **PhoneNumber**
* If **any of these fields match an existing record**, the **household entity should not be imported**, and a **duplication error message should be appended** to the method's output
* **Success Messages**
  + For **each successfully imported household**, append a **success message** to the output, formatted as **Successfully imported household. Contact person: {contactPerson}**
* **Data Persistence**
  + After processing all households from the XML file,   
    **add the valid household entities** to the proper collection
  + **Save the changes** to the database

|  |
| --- |
| **Success message** |
| Successfully imported household. Contact person: {**contactPerson**} |

##### Example

|  |
| --- |
| **households.xml** |
| <?xml version="1.0" encoding="utf-8" ?>  <Households>  <Household phone="+144/123-123456">  <ContactPerson>Alexander Ivanov</ContactPerson>  <Email>alexander.ivanov@example.com</Email>  </Household>  <Household phone="+144/124-123457">  <ContactPerson>Vasil Dimitrov</ContactPerson>  <Email>vasil.dimitrov@example.com</Email>  </Household>  <Household phone="+166/124-166457">  <ContactPerson>Dimi</ContactPerson>  <Email>tooShortName@example.com</Email>  </Household>  <Household phone="+199/124-166457">  <ContactPerson>TooLongName ThatWillNotBeInsertedDueToLenghtDatabaseConstarints</ContactPerson>  <Email>v.dimitrova@example.com</Email>  </Household>  …  </Households> |
| **Output** |
| Successfully imported household. Contact person: Alexander Ivanov  Successfully imported household. Contact person: Vasil Dimitrov  Invalid data format!  Invalid data format!  Successfully imported household. Contact person: Georgi Nikolov  ... |

Upon **correct import logic**, you should have imported **44 records**

### JSON Import

#### Import Expenses

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

##### Constraints

* If any of the required properties is missing, **do not** import any part of the entity and **append an error message** to the **method output**.
* **If any foreign key leads to an inexisting record valid record, do not** import any part of the entity and **append an error message** to the **method output**.
* If **any validation error occurs** for the **expense** entity (**invalid name, amount, date** OR **payment status**), **do not** import any part of the entity and **append an error message** to the **method output**.
  + The **DateTime** **data** in the document will be in the following format: "yyyy-MM-dd"
  + Make sure you use CultureInfo.InvariantCulture
* All records in **"expenses.json"** are guaranteed to be **unique**
* To receive the **correct Success message**, remember to **format the Amount value** to the   
  **second decimal** **place**.

|  |
| --- |
| **Success message** |
| Successfully imported expense – {**expenseName**}, Amount: {**amount:F2**} |

##### Example

|  |
| --- |
| **expenses.json** |
| [  {  "ExpenseName": "Electricity Home",  "Amount": 120.50,  "DueDate": "2024-08-25T00:00:00",  "PaymentStatus": "Unpaid",  "HouseholdId": 1,  "ServiceId": 1  },  {  "ExpenseName": "Water Home",  "Amount": 50.50,  "DueDate": "2024-08-20T00:00:00",  "PaymentStatus": "Unpaid",  "HouseholdId": 1,  "ServiceId": 2  },  {  "ExpenseName": "Internet Home",  "Amount": 40.00,  "DueDate": "2024-08-25T00:00:00",  "PaymentStatus": "Not Paid",  "HouseholdId": 1,  "ServiceId": 3  },  {  "ExpenseName": "Internet Office",  "Amount": 70.00,  "DueDate": "2024-08-15T00:00:00",  "PaymentStatus": "Paid",  "HouseholdId": 2,  "ServiceId": 3  },  …  ] |
| **Output** |
| Successfully imported expense. Electricity Home, Amount: 120.50  Successfully imported expense. Water Home, Amount: 50.50  Invalid data format!  Successfully imported expense. Internet Office, Amount: 70.00  Successfully imported expense. Water Summer House, Amount: 10.50  Successfully imported expense. Security Home, Amount: 50.00  Invalid data format!  **...** |

Upon **correct import logic**, you should have imported **105** **records**

## Data Export (20 pts)

**Use the provided methods in the** Serializer class**.** Usage of **Data Transfer Objects and AutoMapper** is **optional**.

### XML Export

#### Export Households Which Have Expenses To Pay

Export **all households** which have **at least one expense** with a **payment status different from "Paid"**. The households should be **exported with all their expenses that are NOT "Paid"**.   
The exported **data should be in XML format**.   
**Order** the **households alphabetically** by their **contact person**. Within each household, **order the expenses** by **payment date** **in ascending order** and by **amount in ascending order** if dates are the same.

#### Data Fields

* **Household**:
  + **ContactPerson**: Export the contact person of the household
  + **Email**: Export the email of the household
  + **PhoneNumber**: Export the phone number of the household
  + A collection of **Expenses**
* **Expense:**
  + **ExpenseName**: Export the name of the expense
  + **Amount**: Export the amount of the expense, **formatted** to the second decimal place
  + **PaymentDate**: Export the due date of the expense
  + **ServiceName**: Export the name of the service

**Expected XML Output**:

* The root element should be **<Households>**
* Each household should be represented by a **<Household>** element
* Each expense should be represented by an **<Expense>** element within its associated household

##### Example

|  |
| --- |
| ExportHouseholdsWhichHaveExpensesToPay(context) |
| <?xml version="1.0" encoding="utf-16"?>  <Households>  <Household>  <ContactPerson>Alexander Ivanov</ContactPerson>  <Email>alexander.ivanov@example.com</Email>  <PhoneNumber>+144/123-123456</PhoneNumber>  <Expenses>  <Expense>  <ExpenseName>Water Home</ExpenseName>  <Amount>50.50</Amount>  <PaymentDate>2024-08-20</PaymentDate>  <ServiceName>Water</ServiceName>  </Expense>  <Expense>  <ExpenseName>Electricity Home</ExpenseName>  <Amount>120.50</Amount>  <PaymentDate>2024-08-25</PaymentDate>  <ServiceName>Electricity</ServiceName>  </Expense>  </Expenses>  </Household>  <Household>  <ContactPerson>Anton Kolev</ContactPerson>  <Email>anton.kolev@example.com</Email>  <PhoneNumber>+144/123-126786</PhoneNumber>  <Expenses>  <Expense>  <ExpenseName>Water Summer House</ExpenseName>  <Amount>10.50</Amount>  <PaymentDate>2024-07-20</PaymentDate>  <ServiceName>Water</ServiceName>  </Expense>  </Expenses>  </Household>  …  <Households> |

### JSON Export

#### All Services With Suppliers

Export **all services** along **with their associated suppliers**. The exported **data** should be in **JSON format** and adhere to the following specifications:

* **Selection Criteria**:
  + Select all services.
  + For each service, export its name.
  + For each service, include all suppliers that provide the service
* **Data Fields**:
  + **Service**:
    - **ServiceName**: The name of the service.
  + **Supplier**:
    - **SupplierName**: The name of the supplier.
* **Ordering:**
  + Order **services alphabetically** by ServiceName.
  + For each service, order the **suppliers alphabetically** by SupplierName**)**

##### Example

|  |
| --- |
| **ExportServicesWithSuppliers(context)** |
| [  {  "ServiceName": "Electricity",  "Suppliers": [  {  "SupplierName": "E-Service"  },  {  "SupplierName": "Energy-PRO"  },  {  "SupplierName": "Light"  },  {  "SupplierName": "ZEC"  }  ]  },  {  "ServiceName": "Gas",  "Suppliers": [  {  "SupplierName": "BlueHome"  },  {  "SupplierName": "GasGas"  }  ]  },  …  ] |