# Entity Framework Core Regular Exam - 3 August 2024

Exam problems for the [Databases Advanced - Entity Framework course @ SoftUni](https://softuni.bg/trainings/4842/entity-framework-core-february-2025).  
Submit your solutions in the **SoftUni Judge** system (delete all **bin**/**obj** and **packages** folders) [here](https://judge.softuni.org/Contests/5484/Entity-Framework-Core-Regular-Exam-30-March-2025).

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 skip the attachments to the **ImportResults**, **ExportsResults,** and **Datasets** folders.

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.

# Social Network



## Project Skeleton Overview

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

1. Data – contains the SocialNetworkDbContext class and Models folder, which contains the **entity classes**
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:

### User

* Id– **integer**, **Primary Key**
* Username– **text** with length **[4, 20] (required)**
* Email– **text** with length **[8, 60]** **(required)**
* Password – **text** with a **minimum** **length** of **6 (required)**
* Posts – acollection of type Post
* Messages – acollection of type Message
* UsersConversations – acollection of type UserConversation

### Conversation

* Id– **integer**, **Primary Key**
* Title **– text** with length **[2, 30] (required)**
* StartedAt– **DateTime (required)**
* Messages– acollection of type Message
* UsersConversations – acollection of type UserConversation

### UserConversation

* UserId– integer, Primary Key, Foreign Key (required)
* User– User
* ConversationId– integer, Primary Key, Foreign Key (required)
* Conversation – Conversation

### Post

* Id– **integer**, **Primary Key**
* Content **– text** with length **[5, 300]** (**required**)
* CreatedAt– **DateTime (required)**
* CreatorId– **integer**, Foreign Key **(required)**
* Creator– **User**

### Friendship

* UserOneId– **integer**, Foreign Key (required)
* UserOne - User
* UserTwoId – **integer**, Foreign Key (required)
* UserTwo - User

### Message

* Id– integer, **Primary Key**
* Content– **text** with length **1, 200]** (**required**)
* SentAt– **DateTime (required)**
* Status – enum MessageStatus(Sent = 0, Delivered, Seen, Failed)(**required**)
* ConversationId - **integer**, Foreign Key **(required)**
* Conversation - Conversation
* SenderId - **integer**, Foreign Key **(required)**
* Sender – User

## Data Import (20pts)

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**.

**Clarification on Initial Data Seeding:**

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 **in the** OnModelCreating **method**.

If the **models are created correctly**, the **database is built successfully**, and the **seeding code is uncommented**, the following data should be automatically populated:

* 9 Users
* 20 Friendships
* 6 Conversations
* 26 UsersConversations

**Serialize & Deserialize files:**

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:**

|  |
| --- |
| **Duplication message** |
| Duplicated data. |

***Error message and Duplication message will be provided as constants in the skeleton.***

### XML Import

#### Import Messages

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

Each imported **message 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 Message Entities** - Each message entity must be validated against the following criteria:
  + **Content** – Must have a length between **1 and 200 characters** (inclusive)
  + **SentAt** – Must be a **valid DateTime format**

The **DateTime** **data** in the document will be in the following format: "yyyy-MM-ddТHH:mm:ss"  
Make sure you use CultureInfo.InvariantCulture

* + **Status** - Must be a **valid enum value**: **[ Sent, Delivered, Seen, Failed ]**
  + **ConversationId** – Must reference an **existing Conversation** in the database
  + **SenderId** – Must reference an **existing User** in the database
* **Duplication Check** - Before adding a message to the database**, ensure there are no existing records  
   with the same**:
  + **Content, SentAt, Status, and SenderId** within the same **ConversationId**
* If **any validation error occurs** for a message entity **or if any field references a non-existent Conversation/User**, the **message should not be imported**. The appropriate **error message** or **duplication message should be appended** to the method's output
* **Success Messages**
  + For **each successfully imported message**, append a **success message** to the output, formatted as **Successfully imported message (Sent at: {"yyyy-MM-ddTHH:mm:ss"}, Status: {messageStatus})**
* **Data Persistence**
  + After processing all messages from the XML file:
    - **Add the valid message entities** to the proper collection
    - **Save the changes** to the database

|  |
| --- |
| **Success message** |
| Successfully imported message – (Sent at: {"yyyy-MM-ddТHH:mm:ss"}, Status: {messageStatus}) |

##### Example

|  |
| --- |
| **messages.xml** |
| <?xml version='1.0' encoding='UTF-8'?>  <Messages>  <Message SentAt="2025-02-24T14:30:00">  <Content>Hey everyone, let's finalize the project scope.</Content>  <Status>Sent</Status>  <ConversationId>1</ConversationId>  <SenderId>1</SenderId>  </Message>  <Message SentAt="2025-02-24T14:36:00">  <Content></Content>  <Status>Sent</Status>  <ConversationId>1</ConversationId>  <SenderId>1</SenderId>  </Message>  <Message SentAt="2025-02-24T14:40:00">  <Content>I think we should focus on the core functionalities first.</Content>  <Status>Delivered</Status>  <ConversationId>1</ConversationId>  <SenderId>2</SenderId>  </Message>  <Message SentAt="2025-02-24T14:45:00">  <Content>Agreed! We can add extra features later.</Content>  <Status>Seen</Status>  <ConversationId>1</ConversationId>  <SenderId>1</SenderId>  </Message>  …  <Messages> |
| **Output** |
| Successfully imported message (Sent at: 2025-02-24T14:30:00, Status: Sent)  Invalid data format.  Successfully imported message (Sent at: 2025-02-24T14:40:00, Status: Delivered)  Successfully imported message (Sent at: 2025-02-24T14:45:00, Status: Seen)  Successfully imported message (Sent at: 2025-02-24T14:50:00, Status: Sent)  Successfully imported message (Sent at: 2025-02-24T14:55:00, Status: Delivered)  Successfully imported message (Sent at: 2025-02-24T15:00:00, Status: Seen)  Successfully imported message (Sent at: 2025-02-24T15:05:00, Status: Sent)  Successfully imported message (Sent at: 2025-02-24T15:10:00, Status: Delivered)  Duplicated data.  ... |

Upon **correct import logic**, you should have imported **56 messages**

### JSON Import

#### Import Posts

Using the file **"**posts.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 error occurs** for the post entity (**invalid date format, missing required fields, non-existent creator ID**), **do not** import the entity and append an **error message** to the method output.
* **Duplication Check** - Before adding a post to the database**, ensure there are no existing records  
   with the same:** **Content, SentAt, and CreatorId**

The **CreatedAt** data in the document will be in the following format: "yyyy-MM-ddTHH:mm:ss"  
Make sure you use CultureInfo.InvariantCulture

* **The users (Creators) associated with every post** are represented by an **integer ID**, which **must match an existing record** in the database.

|  |
| --- |
| **Success message** |
| Successfully imported post (Creator {creatorUsername}, Created at: 2025-03-01T10:00:00) |

##### Example

|  |
| --- |
| **posts.json** |
| [  {  "Content": "Just finished a fantastic hiking trip in the mountains! Nature is truly refreshing.",  "CreatedAt": "2025-02-24T08:30:00",  "CreatorId": 1  },  {  "Content": "Spent the weekend painting landscapes. Art is my therapy!",  "CreatedAt": "2025-02-23T14:15:00",  "CreatorId": 2  },  {  "Content": "Spent the weekend painting landscapes. Art is my therapy!",  "CreatedAt": "2025-02-23T14:15:00",  "CreatorId": 2  },  {  "Content": "Had an intense chess match today. Strategy and patience are key!",  "CreatedAt": "2025-02-22T19:45:00",  "CreatorId": 3  },  …  ] |
| **Output** |
| Successfully imported post (Creator john\_doe, Created at: 2025-02-24T08:30:00)  Successfully imported post (Creator jane\_doe, Created at: 2025-02-23T14:15:00)  Duplicated data.  Successfully imported post (Creator alex\_smith, Created at: 2025-02-22T19:45:00)  Successfully imported post (Creator sara\_miller, Created at: 2025-02-21T11:20:00)  Successfully imported post (Creator michael\_brown, Created at: 2025-02-20T17:10:00)  Successfully imported post (Creator emily\_white, Created at: 2025-02-19T09:00:00)  Successfully imported post (Creator david\_jackson, Created at: 2025-02-18T16:40:00)  Successfully imported post (Creator olivia\_taylor, Created at: 2025-02-17T18:30:00)  Successfully imported post (Creator william\_clark, Created at: 2025-02-16T07:45:00)  Invalid data format.  **...** |

Upon **correct import logic**, you should have imported **27** **posts**

## Data Export (20 pts)

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

### XML Export

#### Export All Users With Their Friendships Count and Their Posts

Export **all users** along with **their associated posts**. The exported data should be in **XML format**. Order the **users alphabetically by their username**.

For each user:

* **Include all their posts**. **Posts should be listed in ascending order by their PostId**
* Include the **Friendships count**, which represents the **total number of friendships** a user has based on the Friendships mapping table. A user is considered a **friend** if their ID appears **either as UserOneId or UserTwoId**

**Data Fields**:

* **User:** Export the **username,** their **friendships count,** and their **posts**.
* **Post:** Export the **post content** and **creation date**

**Expected XML Output**:

* The root element should be <Users>
* Each user should be represented by a <User> element with an **attribute** 'Friendships=X' indicating their **number of friendships**
* All posts should be presented as an **array** **of** <Post> **elements**, within their associated user
* Each post should be represented by a <Post> **element**

##### Example

|  |
| --- |
| ExportUsersWithFriendShipsCountAndTheirPosts(dbContext) |
| <?xml version="1.0" encoding="utf-16"?>  <Users>  <User Friendships="6">  <Username>alex\_smith</Username>  <Posts>  <Post>  <Content>Had an intense chess match today. Strategy and patience are key!</Content>  <CreatedAt>2025-02-22T19:45:00</CreatedAt>  </Post>  <Post>  <Content>Finally published my first book! A dream come true after years of dedication.</Content>  <CreatedAt>2025-03-03T09:45:00</CreatedAt>  </Post>  <Post>  <Content>Access to clean drinking water should be a fundamental right, not a privilege.</Content>  <CreatedAt>2025-04-02T14:30:00</CreatedAt>  </Post>  </Posts>  </User>  <User Friendships="3">  <Username>david\_jackson</Username>  <Posts>  <Post>  <Content>Practicing guitar has been so rewarding. Music is the soul’s language!</Content>  <CreatedAt>2025-02-18T16:40:00</CreatedAt>  </Post>  <Post>  <Content>We bought our first house! Finally moving into our dream home.</Content>  <CreatedAt>2025-03-07T14:45:00</CreatedAt>  </Post>  <Post>  <Content>The rising cost of healthcare is leaving many families behind. We need reforms!</Content>  <CreatedAt>2025-04-03T09:15:00</CreatedAt>  </Post>  </Posts>  </User>  <User Friendships="3">  <Username>emily\_white</Username>  <Posts>  …  </Posts>  …  </User>  …  <Users> |

### JSON Export

#### All Conversations With Messages Chronologically

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

* **Selection Criteria**:
  + Select **all conversations** in the database
  + For each conversation, export its **title** and **start time** (StartedAt)
  + Include **all messages** within each conversation
* **Data Fields**:
  + Conversation – **Id, Title, StartedAt**
  + Message – **Content, SentAt, Status, SenderUsername**
* **Ordering:**
  + **Order** conversations **by their start time** (StartedAt **ascending**)
  + **Order** messages within each conversation **by sending time, ascending**
* **DateTime Format:**
  + **All date and time fields (**StartedAtandSentAt**)** must be formatted as **"**yyyy-MM-ddTHH:mm:ss**"**
  + **Example: "**2025-02-24T14:30:00**"**
  + Ensure thatCultureInfo.InvariantCultureis used for correct serialization

##### Example

|  |
| --- |
| **ExportConversationsWithMessagesChronologically(dbContext)** |
| [  {  "Id": 6,  "Title": "Study Group",  "StartedAt": "2024-08-10T14:00:00",  "Messages": [  {  "Content": "Hey ladies, let's focus on the upcoming exam topics.",  "SentAt": "2024-08-10T14:05:00",  "Status": 0,  "SenderUsername": "jane\_doe"  },  {  "Content": "Yes! I was reviewing the lecture on algorithms, it’s tricky.",  "SentAt": "2024-08-10T14:10:00",  "Status": 1,  "SenderUsername": "sara\_miller"  },  {  "Content": "Let’s break it down together. What part is confusing?",  "SentAt": "2024-08-10T14:15:00",  "Status": 2,  "SenderUsername": "emily\_white"  },  {  "Content": "I struggle with recursion. The call stack gets confusing.",  "SentAt": "2024-08-10T14:20:00",  "Status": 0,  "SenderUsername": "olivia\_taylor"  },  {  "Content": "I can explain! Think of it like a nested to-do list.",  "SentAt": "2024-08-10T14:25:00",  "Status": 1,  "SenderUsername": "jane\_doe"  },  {  "Content": "That makes sense! Let’s go over an example step by step.",  "SentAt": "2024-08-10T14:30:00",  "Status": 2,  "SenderUsername": "sara\_miller"  },  {  "Content": "Also, let’s make flashcards for key concepts.",  "SentAt": "2024-08-10T14:35:00",  "Status": 0,  "SenderUsername": "emily\_white"  },  {  "Content": "Good idea! I’ll prepare some on data structures.",  "SentAt": "2024-08-10T14:40:00",  "Status": 1,  "SenderUsername": "olivia\_taylor"  },  {  "Content": "Awesome! Let’s quiz each other in an hour.",  "SentAt": "2024-08-10T14:45:00",  "Status": 2,  "SenderUsername": "jane\_doe"  }  ]  },  {  "Id": 5,  "Title": "BackUp Group",  "StartedAt": "2024-08-10T20:11:00",  "Messages": […  ] |

## Prepare the Solution File Archive

A screenshot of a computer

AI-generated content may be incorrect.