# Lab: Team Builder

This document defines the **exercise assignments** for the "[Databases Advanced – Entity Framework" courses @ Software University](https://softuni.bg/courses/databases-advanced-entity-framework).

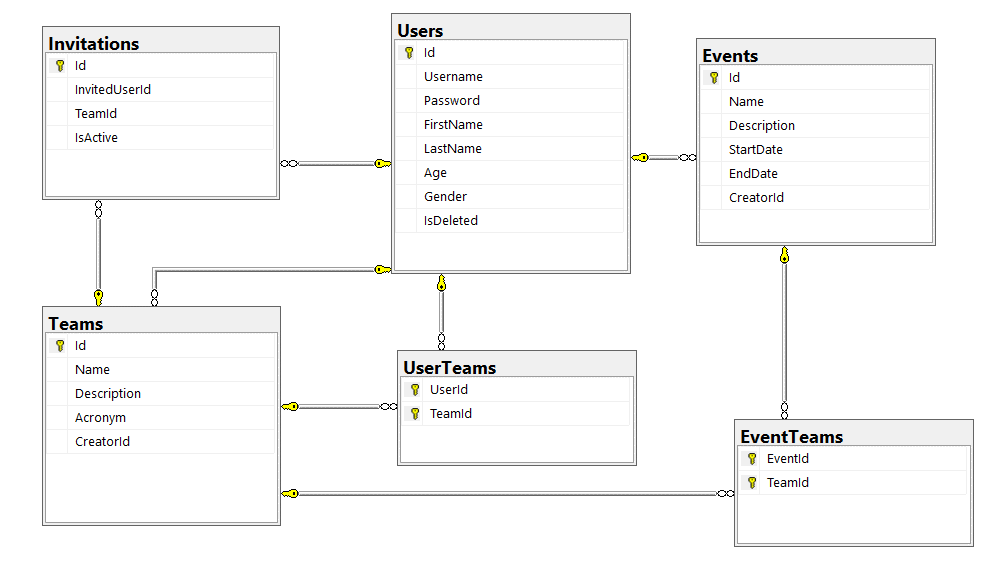
This Lab is continuation of previous Lab assignment. You can check previous part [here](https://softuni.bg/downloads/svn/DB-Fundamentals/DB-Advanced-EntityFramework/Feb-2017/12.%20DB-Advanced-EntityFramework-Workshop/12.%20DB-Advanced-EntityFramework-TeamBuilder-Lab-Basic-Logic.docx).

The first [section](#_Team_Builder) is duplicated in order to provide everything we need for our application in one file.

## Team Builder

Your task is to implement Team Builder console application. The application will consist of **users**, **teams** and **events**. Each event has several teams participating in it and each team has several users. **User** may **send** **join** **request** on **team** in order to be added to that team **or** **team** may **send** **invitation** **to user** so they can join.

The application consists of the following models:



**Here is information about each table:**

**Users**

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Constraints** |
| Id | Integer from 0 to 2,147,483,647 | Unique table identificator |
| Username | String from 3 to 25 symbols | Unique, Required |
| FirstName | String up to 25 symbols |  |
| LastName | String up to 25 symbols |  |
| Password | String from 6 to 30 symbols | Should contain one digit and one uppercase letter, Required |
| Gender | Enumeration | Could be: '*Male*' or '*Female*' |
| Age | Integer from 0 to 2,147,483,647 |  |
| IsDeleted | Bool |  |

**Teams**

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Constraints** |
| Id | Integer from 0 to 2,147,483,647 | Unique table identificator, Identity |
| Name | String up to 25 symbols | Unique, Required |
| Description | String up to 32 symbols |  |
| Acronym | String with exactly 3 symbols | Must be 3 symbols long, Required |

**Events**

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Constraints** |
| Id | Integer from 0 to 2,147,483,647 | Unique table identificator, Identity |
| Name | String up to 25 symbols, Unicode | Required |
| Description | String up to 250 symbols, Unicode |  |
| StartDate | DateTime in format {dd/MM/yyyy HH:mm} |  |
| EndDate | DateTime in format {dd/MM/yyyy HH:mm} | Must be after StartDate |
| CreatorId | Integer from 0 to 2,147,483,647 | Relationship with table Users |

**Invitations**

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Constraints** |
| Id | Integer from 0 to 2,147,483,647 | Unique table identificator, Identity |
| InvitedUserId | Integer from 0 to 2,147,483,647 | Relationship with table Users |
| TeamId | Integer from 0 to 2,147,483,647 | Relationship with table Teams |
| IsActive | Boolean |  |

**UserTeams**

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Constraints** |
| UserId | Integer from 0 to 2,147,483,647 | Relationship with table Users, Unique table identificator |
| TeamId | Integer from 0 to 2,147,483,647 | Relationship with table Teams, Unique table identificator |

**TeamEvents**

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Constraints** |
| TeamId | Integer from 0 to 2,147,483,647 | Relationship with table Teams, Unique table identificator |
| EventId | Integer from 0 to 2,147,483,647 | Relationship with table Events, Unique table identificator |

### Application Summary

**User** can **create** **event** or **team** – becoming their creator. **One** **event** may have **several teams** while **single team** can participate **in multiple events**. **Team** consists of **users** which also can be part of **other teams**.

Anyone from a team can **invite** people to join. Only the **creator** may **remove** **users** or to **disband** the whole **team**.

In order for a team to successfully participate in event – team’s creator must apply for it and later on to be approved by the creator of the event.

### Application Functionality

Team Builder contains the following functionality:

* **RegisterUser <username> <password> <repeat-password> <firstName> <lastName> <age> <gender>**Registers a new user.

|  |  |  |
| --- | --- | --- |
| **Case** | **Message** | **Exception** |
| Success | User [username] was registered successfully! | None |
| Username is not valid | Username [username] not valid! | ArgumentException |
| Password is not valid | Password [password] is not valid! | ArgumentException |
| Age is not in valid format or is non-positive number | Age not valid! | ArgumentException |
| Gender is not valid | Gender should be either “Male” or “Female”! | ArgumentException |
| Passwords do not match | Passwords do not match! | InvalidOperationException |
| Username is taken | Username [username] is already taken! | InvalidOperationException |
| There is currently logged in user | You should logout first! | InvalidOperationException |

\*Validation on first/last name is removed for the sake of simplicity – you are not obligated to perform any validation checks.

* **Login <username> <password>**Logs a user into the system and keep a reference to it until the “**Logout**” command is called.

|  |  |  |
| --- | --- | --- |
| **Case** | **Message** | **Exception** |
| Success | User [username] successfully logged in! | None |
| Either user does not exist or password does not match or user is deleted | Invalid username or password! | ArgumentException |
| There is currently logged in user | You should logout first! | InvalidOperationException |

* **Logout**Logs out a user from the application.

|  |  |  |
| --- | --- | --- |
| **Case** | **Message** | **Exception** |
| Success | User [username] successfully logged out! | None |
| There is no user logged in. | You should login first! | InvalidOperationException |

* **DeleteUser**Deletes currently logged in user and then logs out.

|  |  |  |
| --- | --- | --- |
| **Case** | **Message** | **Exception** |
| Success | User [username] was deleted successfully! | None |
| There is no user logged in. | You should login first! | InvalidOperationException |

|  |  |  |
| --- | --- | --- |
| **Case** | **Message** | **Exception** |
| Success | Event [eventName] was created successfully! | None |
| Either start date or end date is in invalid format | Please insert the dates in format:  [dd/MM/yyyy HH:mm]! | ArgumentException |
| Start date is after end date | Start date should be before end date. | ArgumentException |
| There is no logged in user | You should login first! | InvalidOperationException |

* **CreateEvent <name> <description> <startDate> <endDate>**Creates an event (currently logged user is it’s creator). Keep in mind when parsing dates that there should be additional spaces between them.

**\***There might be several events with the same name. **Always pick the one with the latest start date!**

* **CreateTeam <name> <acronym> <description>**Creates a team (currently logged user is it’s creator). Description is optional.

|  |  |  |
| --- | --- | --- |
| **Case** | **Message** | **Exception** |
| Success | Team [team] successfully created! | None |
| Team does exist | Team [team] exists! | ArgumentException |
| Acronym is not valid | Acronym [acronym] not valid! | ArgumentException |
| There is no logged in user | You should login first! | InvalidOperationException |

* **InviteToTeam <teamName> <username>**Sends an invite to the specified user to join given team. If the user is actually the creator of the team – add him/her directly!

|  |  |  |
| --- | --- | --- |
| **Case** | **Message** | **Exception** |
| Success | Team [teamName] invited [username]! | None |
| If the user is not creator of the team nor part of it | Not allowed! | InvalidOperationException |
| Either user or team does not exist | Team or user does not exist! | ArgumentException |
| There is an already active invite | Invite is already sent! | InvalidOperationException |
| There is no logged in user | You should login first! | InvalidOperationException |

* **AcceptInvite <teamName>**Checks current user’s active invites and **accepts** the one from the team specified.

|  |  |  |
| --- | --- | --- |
| **Case** | **Message** | **Exception** |
| Success | User [username] joined team [teamName]! | None |
| Team does not exist | Team [teamName] not found! | ArgumentException |
| There is no invite from that team | Invite from [teamName] is not found! | ArgumentException |
| There is no logged in user | You should login first! | InvalidOperationException |

* **DeclineInvite <teamName>**Checks current user’s active invites and **declines** the one from the team specified.

|  |  |  |
| --- | --- | --- |
| **Case** | **Message** | **Exception** |
| Success | Invite from [teamName] declined. | None |
| *\*Look in above command to see other cases.* | | |

* **KickMember <teamName> <username>**Removes specified user member from given team. Only the creator of the team can kick other members.

|  |  |  |
| --- | --- | --- |
| **Case** | **Message** | **Exception** |
| Success | User [username] was kicked from [teamName]! | None |
| Team does not exist | Team [teamName] not found! | ArgumentException |
| User does not exist | User [username] not found! | ArgumentException |
| User is not a member in team | User [username] is not a member in [teamName]! | ArgumentException |
| Current user is not creator of the team | Not allowed! | InvalidOperationException |
| User to be kicked is the creator of the team | Command not allowed. Use DisbandTeam instead. | InvalidOperationException |
| There is no logged in user | You should login first! | InvalidOperationException |

* **Disband <teamName>**Deletes given team. Allowed for the team’s creator only.

|  |  |  |
| --- | --- | --- |
| **Case** | **Message** | **Exception** |
| Success | [teamName] has disbanded! | None |
| Team does not exist | Team [teamName] not found! | ArgumentException |
| Current user is not creator of the team | Not allowed! | InvalidOperationException |
| There is no logged in user | You should login first! | InvalidOperationException |

* **AddTeamTo <eventName> <teamName>**Adds given team for event specified. If there are more than one events with same name pick the latest start date.

|  |  |  |
| --- | --- | --- |
| **Case** | **Message** | **Exception** |
| Success | Team [teamName] added for [eventName]! | None |
| Event does not exist | Event [eventName] not found! | ArgumentException |
| Team does not exist | Team [teamName] not found! | ArgumentException |
| Current user is not creator of the event | Not allowed! | InvalidOperationException |
| Team is already added to event | Cannot add same team twice! | InvalidOperationException |
| There is no logged in user | You should login first! | InvalidOperationException |

* **ShowEvent <eventName>**Shows details for given event.

|  |  |  |
| --- | --- | --- |
| **Case** | **Message** | **Exception** |
| Success | [eventName] [eventStartDate] [eventEndDate]  [description]  Teams:  -[teamName]  … | None |
| Event does not exist | Event [eventName] not found! | ArgumentException |

* **ShowTeam <teamName>**Show details about given team.

|  |  |  |
| --- | --- | --- |
| **Case** | **Message** | **Exception** |
| Success | [teamName] [teamAcronym]  Members:  --[username1]  …  --[usernameN] | None |
| Team does not exist | Team [teamName] not found! | ArgumentException |

* **Exit**Exits application.

|  |  |  |
| --- | --- | --- |
| **Case** | **Message** | **Exception** |
| Success | *None* | None |

If command’s name is different from any of the commands above, throw NotSupportedException with message: “Command [commandName] not valid!”.

If format of the command is not valid (invalid number or arguments) throw FormatException with message: “Invalid arguments count!”

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| RegisterUser johny j0hny j0hny John Smith 22 Male  RegisterUser johny Inval1d Inval1d John Smith 22 Male  Login johny Invalid  Login johny Inval1d  Logout  Logout j0hny  Login johny Inval1d  DeleteUser  Logout  Login johny Inval1d  Exit | Password j0hny not valid!  User johny was registered successfully!  Invalid username or password!  User johny successfully logged in!  User johny successfully logged out!  Invalid arguments count!  User johny successfully logged in!  User johny was deleted successfully!  You should login first!  Invalid username or password! |
| RegisterUser daniel Dan123 Dan123 Daniel Trevor 22 MMale  RegisterUser daniel Dan123 Dan123 Daniel Trevor 22 Male  Login daniel Dan123  CreateEvent TEDexSofia Inovation 01-01-2012 12:00 02-01-2012 22:00  CreateEvent TEDexSofia Inovation 01/01/2012 12:00 02/01/2012 22:00  CreateTeam Band BND  CreateTeam BitColns BCS  AddTeamTo TEDexSofia Band  AddTeamTo TEDexSofia BCS  AddTeamTo TEDexSofia Band  AddTeamTo TEDexSofia BitColns  ShowEvent TEDexSofia  Exit | Gender should be either “Male” or “Female”!  User daniel was registered successfully!  User daniel successfully logged in!  Please insert the dates in format: [dd/MM/yyyy HH:mm]!  Event TEDexSofia was created successfully!  Team Band successfully created!  Team BitColns successfully created!  Team Band added for TEDexSofia!  Team BCS not found!  Cannot add same team twice!  Team BitColns added for TEDexSofia!  TEDexSofia 01/01/2012 12:00 02/01/2012 22:00  Inovation  Teams:  -Band  -BitColns |
| RegisterUser gordon Ham123 Ham123 Gordon Hamilton -2 Male  RegisterUser gordon Ham123 Ham123 Gordon Hamilton 32 Male  RegisterUser terrydom Terry123 Terry123 Terry Molina 32 Female  Login gordon Ham123  CreateEvent CrackIT ITHardware 22/10/2013 12:00 22/10/2013 22:00  CreateEvent CrackIT ITHard 13/08/2015 12:00 15/08/2015 22:00  CreateTeam Crackers CKS  CreateTeam Balder BLD  InviteToTeam Crackers terry-dom  InviteToTeam Crackers terrydom  InviteToTeam Balder terrydom  Logout  Login terrydom Terry123  AcceptInvite CrackIT  AcceptInvite Crackers  DeclineInvite Balder  Disband Balder  Logout  Login gordon Ham123  ShowTeam Balder  Disband Balder  ShowTeam Crackers  KickMember Crackers terry-dom  KickMember Crackers terrydom  AddTeamTo CrackIT Crackers  ShowEvent CrackIT  Exit | Age not valid!  User gordon was registered successfully!  User terrydom was registered successfully!  User gordon successfully logged in!  Event CrackIT was created successfully!  Event CrackIT was created successfully!  Team Crackers successfully created!  Team Balder successfully created!  Team or user does not exist!  Team Crackers invited terrydom!  Team Balder invited terrydom!  User gordon successfully logged out!  User terrydom successfully logged in!  Team CrackIT not found!  User terrydom joined team Crackers!  Invite from Balder declined.  Not allowed!  User terrydom successfully logged out!  User gordon successfully logged in!  Balder BLD  Members:  Balder has disbanded!  Crackers CKS  Members:  --terrydom  User terry-dom not found!  User terrydom was kicked from Crackers!  Team Crackers added for CrackIT!  CrackIT 13/08/2015 12:00 15/08/2015 22:00  ITHard  Teams:  -Crackers |

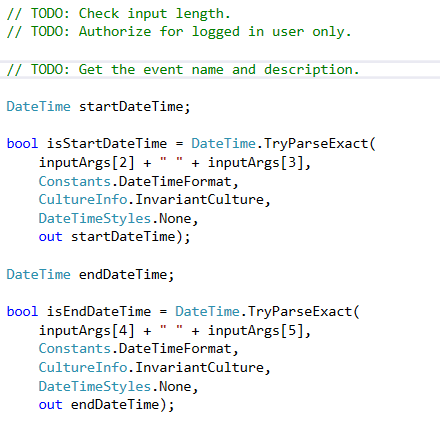
## Implement Advanced Logic

In this section we will continue with adding commands. First command to implement is CreateEventCommand. Again we should the requirements for this one in [Application Functionality](#_Application_Functionality) section.

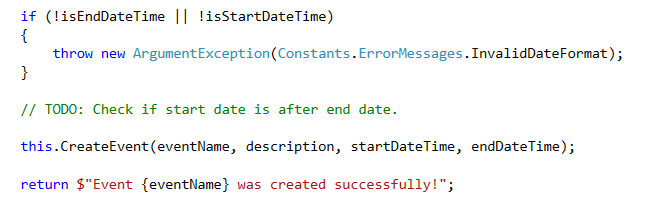
Create another command in the Commands folder with Execute method in it.

This command is special though because of the format of the input. We have to parse **two dates each** having **additional** **whitespace** in it: “**dd/MM/yyyy HH:mm**”.

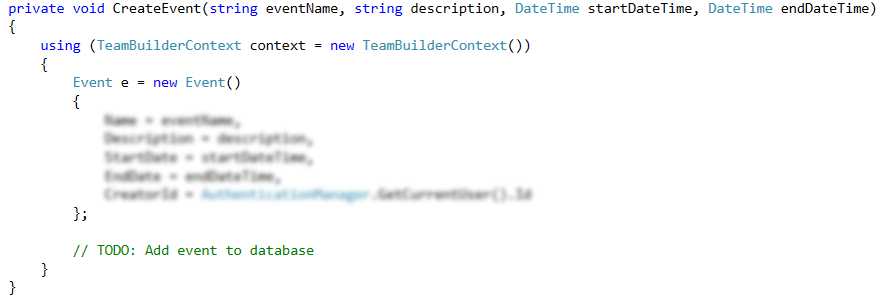
As starter we should do the following:



Now let’s validate the dates and create the event:



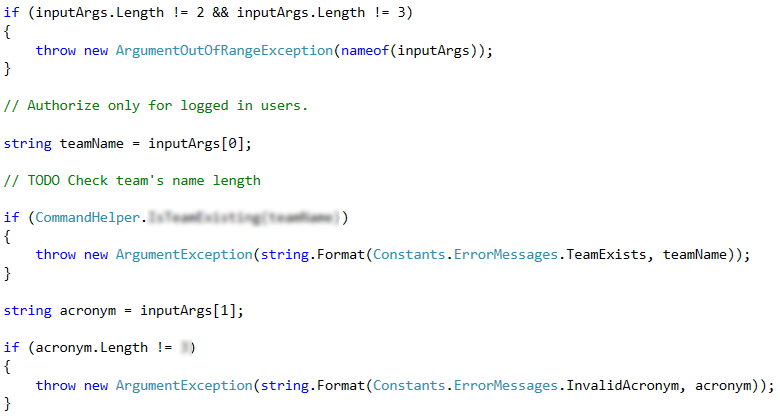
Let’s peak inside the CreateEvent() method:



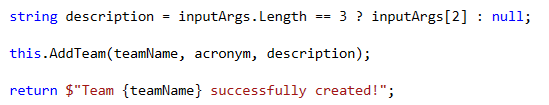
Go back to the CommandDispatcher class and another case for this command as well.

Feel free to test the command.

Moving on to the next one – it is CreateTeamCommand which is relatively the same. If we take closer to the command however we will see that we have optional argument (team’s description) which means that we have to check for the length of the input parameters exclusively in our command:

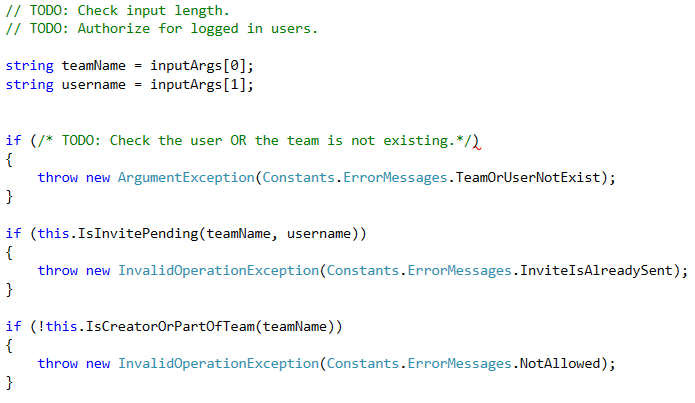


What is left after the validation is to actually insert the newly created team:

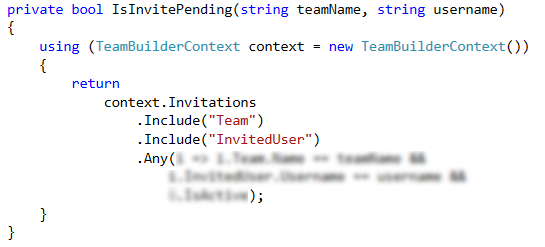


The implementation of AddTeam() is left to you. 😊

Next one will be InviteToTeamCommand. Here we don’t have any specifics we have just to follow the cases which are given:

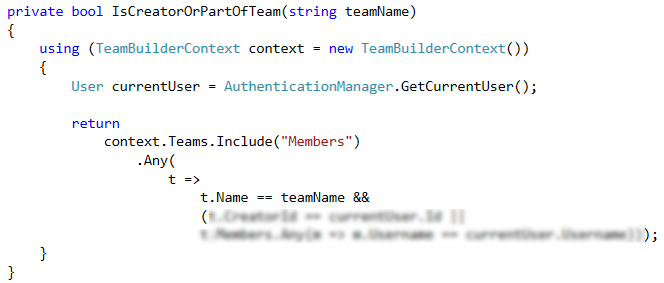


We have to make a couple of queries in order to validate e of any active invites or to validate if the current user may invite other people.

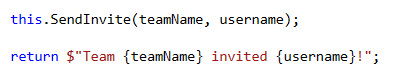


Here you should check if there is at least one **active** **invite** from given **team** for given **username**.

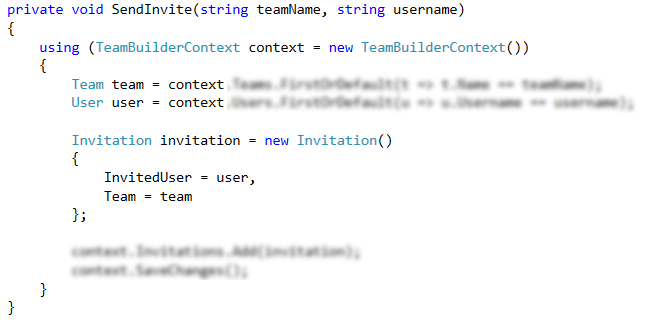
In order the user to invite some users he must be either **creator** of the **team** **or** a **member** of it. Let’s implement that query:



After we performed every validation it is time to send the invitation to the user:

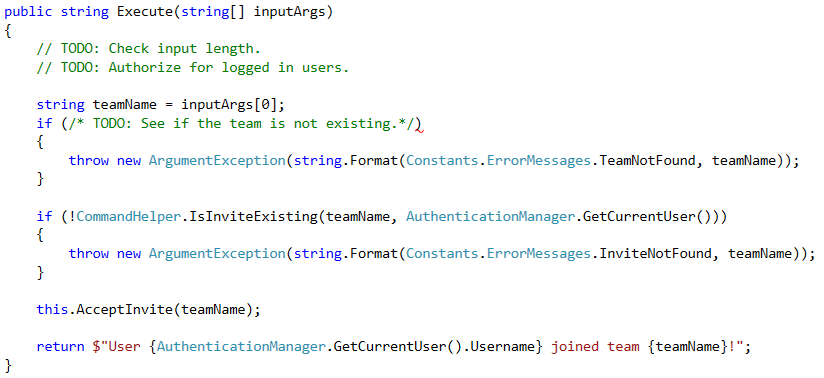


Remember when **creating** new **invitation** to load the user and the team from the context:

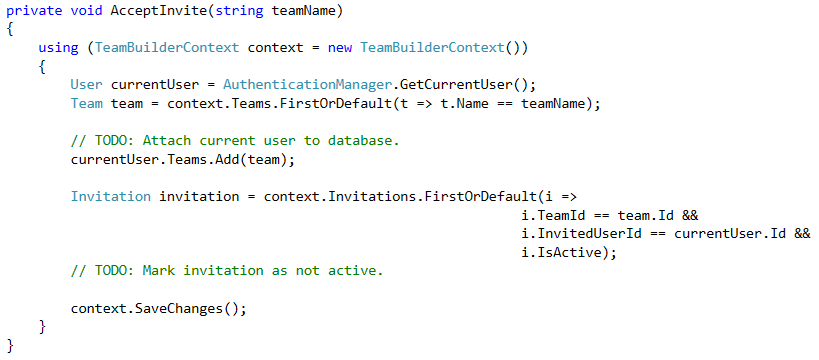


Remember to go back to the **command** **dispatcher** and insert cases for the last two commands!

Now we jump on AcceptInviteCommand. Again we may need to see the [Functionality](#_Application_Functionality) section in order to understand what is expected from the command. This command will be implemented just like others – with one Execute method containing the logic of the command:



After all validation is passed it is time to see how the invite should be accepted:

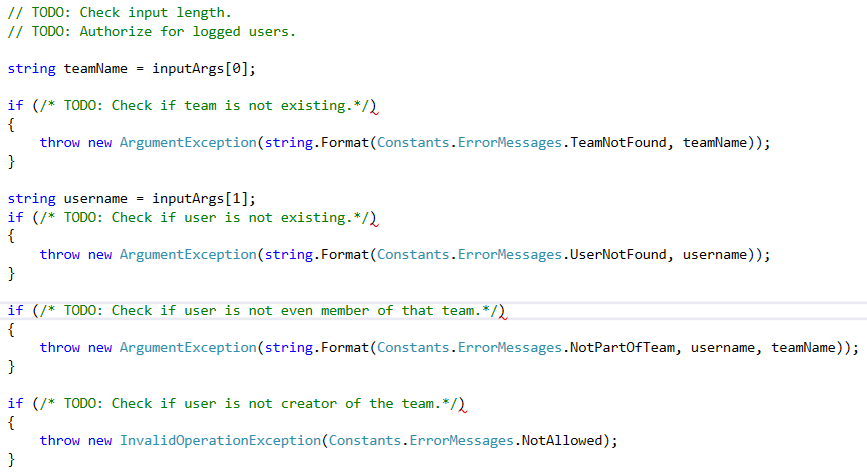


Well we have DeclineInviteCommand too. It contains the same logic (well, without current user joining a team) – we should make the invitation (if it exists) not active. This command is **left to you** to implement. 😊

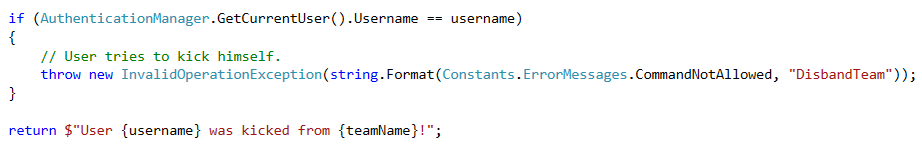
Remember to go to CommandDispatcher and add new case for the new commands … [again](http://reactiongifs.me/wp-content/uploads/2014/05/MRW-I-miss-McDonalds-breakfast-by-just-a-few-minutes-james-franco-crying-pineapple-express.gif).

It is time to kick some … members. Let’s implement KickMemberCommand. Again it is good idea to see the [Functionality](#_Application_Functionality) section.

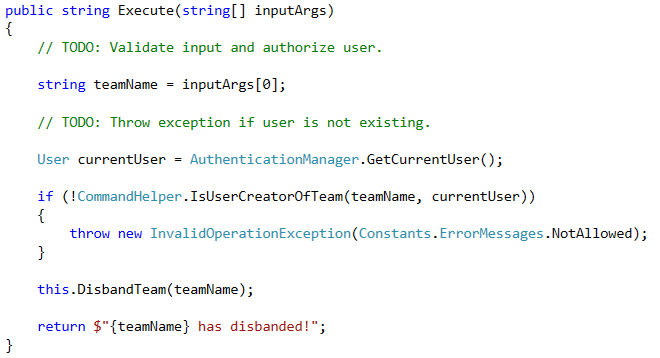
We will have something like this:



One last check before we finish up this command:

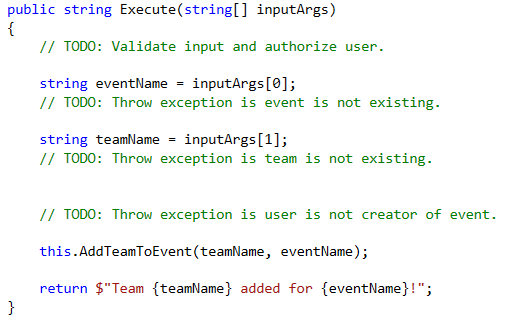


Let’s move on to the DisbandTeamCommand:

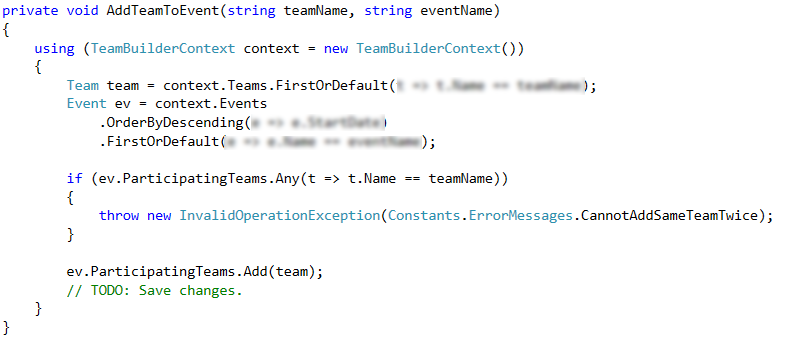


The DisbandTeam method just deletes the team with name specified.

The last command we will show in this lab is AddTeamToCommand:



Let’s peak into AddTeamToEvent():



Feel free to test the application (we already have all commands but two).

ShowTeam and ShowEvent are left to you to implement.

You can fully test your application with the examples given above or make your tests.

## \*Improve Application

We saw that after adding command each time we had to go back to the **command dispatcher** and a new switch **case** for it. It is ... **obnoxious**. There is an easy way to implement but it requires Reflection.

Go to the CommandDispatcher class and above the switch case insert the following code:

|  |
| --- |
| // Get command's type.  Type commandType = Type.GetType(  "TeamBuilder.App.Core.Commands." + commandName + "Command");  // If command's type is not found – it is not valid command.  if (commandType == null)  {  throw new NotSupportedException($"Command {commandName} not supported!");  }  // Create instance of command with the type that we already extracted.  object command = Activator.CreateInstance(commandType);  // Get the method called “Execute” of the command.  MethodInfo executeMethod = command.GetType().GetMethod("Execute");  // Invoke the method we found passing the instance of the command and  // array of all expected arguments that the method should take when it is invoked.  result = executeMethod.Invoke(command, new object[] { inputArgs }) as string; |

After that you can **remove** the **switch case** construction and the application will run just the same (note that if you mistyped even one command’s name or method signature – it is not likely to work).

## Import Data – XML

Let’s add additional commands ImportUsers and ImportTeams. Here is what they should do:

\*Follow the format given in the [resources](http://svn.softuni.org/admin/svn/DB-Fundamentals/DB-Advanced-EntityFramework/Feb-2017/13.%20DB-Advanced-EntityFramework-Big-Overall-Exercise/13.%20DB-Advanced-EntityFramework-TeamBuilder-Lab-Advanced-Logic-Resources.zip) file bundle (“users.xml” and “teams.xml”).

* **ImportUsers <filePathToXmlFile>**Import users from given xml file.

|  |  |  |
| --- | --- | --- |
| **Case** | **Message** | **Exception** |
| Success | You have successfully imported {usersCount} users! | None |
| If there is any error while parsing xml | Xml format not valid! | FormatException |
| File not found | Path {filePath} is not valid! | FileNotFoundException |

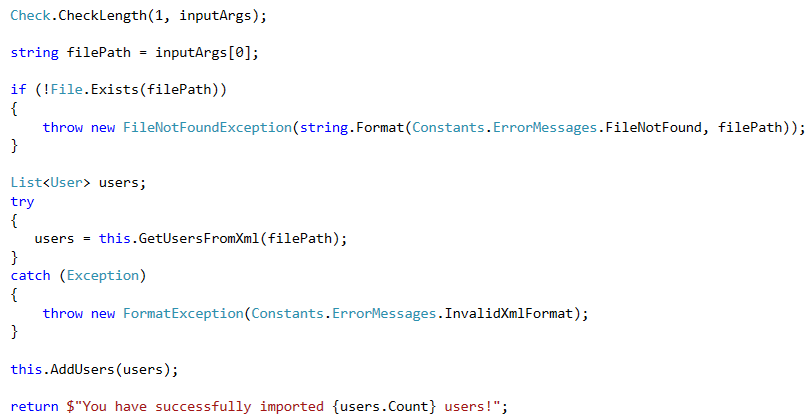
* **ImportTeams <filePathToXmlFile>**Import teams from given xml file.

|  |  |  |
| --- | --- | --- |
| **Case** | **Message** | **Exception** |
| Success | You have successfully imported {teamsCount} teams! | None |
| If there is any error while parsing xml | Xml format not valid! | FormatException |
| File not found | Path {filePath} is not valid! | FileNotFoundException |

First let’s add 2 new error messages in our Constants.ErrorMessages class:

|  |
| --- |
| public const string FileNotFound = "Path {0} is not valid!";  public const string InvalidXmlFormat = "Invalid Xml format!"; |

Let’s begin with ImportUsers command:

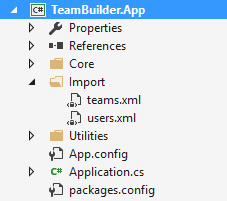


GetUsersFromXml() will simply read the xml file at given path, parse it and **return** a **collection** of type User. \*Make sure when parsing gender to ignore casing.

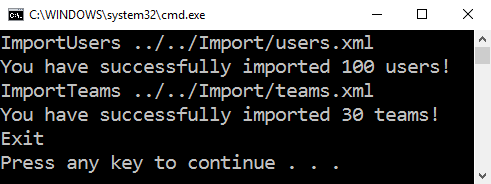
AddUsers() will add the collection with users to database.

ImportTeamsCommand is relatively the same so it’s left to you.

You could create **Import** folder and put your xml files there:



At last something like this should happen:



## Export Data – JSON

One last command to add: ExportTeam:

* **ExportTeam <teamName>**Exports team just like ShowTeam() but in json format and in file. The exported file should be where the “**.exe**” file is and should be named “**team.json**”.

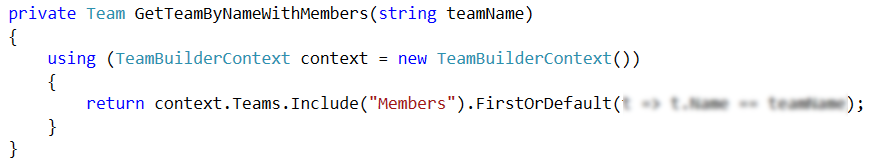
|  |  |  |
| --- | --- | --- |
| **Case** | **Message** | **Exception** |
| Success | Team {teamName} exported! | None |
| Team does not exist | Team [teamName] not found! | ArgumentException |

### Hints

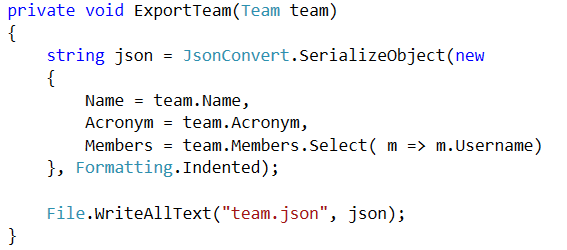
The screenshot below shows implementation of the upper method:



However with this structure when we get the team from database we want to **include** the **members**:



When serializing to **json** you can use **anonymous** object (there is **no** need of **DTO** class):



### Example

|  |
| --- |
| RegisterUser tonyy Tonyy1 Tonyy1 Tony Smith 23 Male  RegisterUser barber Barber1 Barber1 Paul Trevoc 12 Male  RegisterUser kaban Kaban1 Kaban1 Elena Terry 18 Female  RegisterUser salazar Salazar1 Salazar1 Salazar Tore 34 Male  RegisterUser johnya J0hnyy J0hnyy John Smith 22 Male  RegisterUser derprot Lama123 Lama123 Tedesse Melaku 22 Female  Login tonyy Tonyy1  CreateTeam Monsterr MNR  InviteToTeam Monsterr kaban  InviteToTeam Monsterr barber  InviteToTeam Monsterr salazar  InviteToTeam Monsterr johnya  InviteToTeam Monsterr derprot  Logout  Login barber Barber1  AcceptInvite Monsterr  Logout  Login kaban Kaban1  AcceptInvite Monsterr  Logout  Login salazar Salazar1  AcceptInvite Monsterr  Logout  Login johnya J0hnyy  AcceptInvite Monsterr  Logout  Login derprot Lama123  AcceptInvite Monsterr  Logout  ExportTeam Monsterr  Exit |

The exported file **team.json** should look like this:

|  |
| --- |
| {  "Name": "Monsterr",  "Acronym": "MNR",  "Members": [  "barber",  "kaban",  "salazar",  "johnya",  "derprot"  ]  } |