INF272

18. Campus Practical -Entity Framework 2.

Due: 2022/10/03 before 23:59 loaded on ClickUP.

IMPORTANT NOTE:

1. You should have logged into .\user (please note that there is no password for this local account) if you use the Informatorium PCs. If you log into your own account, then you will not be able to control SQL Server as your student account does not have administrative privileges. If not logged into .\user, please reboot, or log out, and log into .\user. Before rebooting or logging out, save your files on a safe location.

2. The activity instructions are in the sequence of the rubric. Some instructions will be longer or shorter based on what you need to do. Some instructions may be much longer for less marks as it is assumed that students can complete the task (less marks), but the description might be longer as a reminder and an additional learning opportunity.

3. If you try to test the pages linked to the Processing Logic Controllers (containing C#) and the file path is too long, you WILL get errors. There is a file execution limit (on the file folder path) in most Operating Systems (OS) and executable files cannot be accessed if the folder path mapping is too long. Make sure the file path is as short as possible by creating a folder in the root directory, copying your files into the root, and then running it from the root. In short, if you click on the (C:) drive, that would be considered the root directory. Programs would also run faster as the IDE would not have to traverse long file paths to call and raise (activate and use) processing capabilities from the OS.

4. Do not try to run the project from within the zipped file. Zipping "document-like" files, such as VS solution files, make use of lossless compression. Lossless compression looks for repeating values, which it then shortens and assigns to an internal zip dictionary. In other words, the face-value repetitive syntax application code in a zipped file is not the original code. For example, ConvertTo might be recoded to [1] in the compressed file, and then in the internal zip file dictionary, the value would be stored as [1] = ConvertTo. Simply stated, the standard format syntax is not present in a zipped file other than the zip dictionary as it has been replaced by indexed key values. You might have noticed opening files directly from within a zipped file is slower than expected. The reason for this is that the zipping program is reading from the internal zip dictionary and converting the index values temporarily back into the original values to "rebuild" the original file.

5. All three of the required NuGet packages (Entity Framework, PageList and PageList.MVC) have already been installed. There is no need to reinstall the required NuGet packages again.

6. **You can use the activity document to follow and understand the example pages in the activity solution as it contains steps, descriptions, and explanations that can also be followed in the example files. No internal documentation is provided in the project to explain sections of code.**

***Disclaimer**: There might be accidental and unintentional language errors and typos in this document.*

1. Database connection.

Start by searching for services (Windows Search function) and check if SQL Server is running. If not, then right click on it and select start. If it is running, then right click on it and then select restart.

Make sure that you have downloaded / opened the SQL script on the Background View (first page of the project when you open it. Unzip the file. Double click on the file. SQL Server will open. Connect to the server and then execute the script.

Care has been taken to ensure the script runs correctly.

If the script however throws errors, then identify the lines in the error log and correct the scripting errors. The probability of corrections and edits being required is very low, but it is always a possibility if the incorrect version of SQL Server has been installed. The correct version would be Version 15.0.2095.3 (as indicated on ClickUP) or newer. Newer versions should still execute the script. There are numerous reasons that this might sometimes happen, but in essence it will always be a possibility in a development environment.

As part of the activity you will be required to update the connection string.

Open the web-config file in the project and then update the connection string by changing the server name to your server name. The connection string may be found at the near bottom of the web.config file. This is the general location where Entity Framework [EF] always right the EF connection string.

Please note the example. You need to change SAMURAI_JACK to your server name. It is usually the PC / workstation name.

```
... data source=SAMURAI_JACK\SQLEXPRESS;...
```

After updating the connection string, then (1) Clean and then (2) Rebuild the solution (found on the Build menu-item). This is required to update the connection to SQL. If you do not do this, EF will not re-establish and map the new connection.

After updating the string and cleaning and rebuilding the project, then open the Server Explorer, and then right click on HumanResourceDB and select refresh. You might need to Clean and Rebuild again after refreshing the server in the Server Explorer. Under normal circumstances this would not be required, however if your PC is a little slow, then Visual Studio may not refresh the connection without completing another Clean and Rebuild.

The project should now be ready.

2. Searching (re-use and update pattern). Use the street address to search.

In this instance you will have to make use of the searching pattern that may be found in the x3 provided example Controllers and the associated views. You will be referencing the Activity.cs Model in the Models folder. The Activity.cs Model has been provided and you will not need to update or change it. If you refer to the Activity.cs Model, you will notice the following:

```
public location LocationDetails { get; set; }
public department DepartmentDetails { get; set; }
public employee EmployeeDetails { get; set; }
public job JobDetails { get; set; }
```

As may be seen in the Activity.cs Model, you will need to make use of x4 tables in the provided database. The Language-Integrated Query (LINQ) to join 4 tables is similar to the query used to access one table, the query used to join 2 tables as well as the query to join 3 tables.

In the query to join 2 and 3 tables, the query is extremely sequential in term of accessing the tables. It follows a direct and sequential [Pk] Parent-Child [FK] relationship such as for example:

[PK_ID] Employees >> Location [FK_ID].

As such you will find the LINQ query extremely sequential. Please note how the temporary variables are used to access the data in terms of the [PK] and [FK] relationship. You can also make use of lambda statements. Please refer to the linked example project (Background View) for examples on how to use lambda to access and manipulate data from a database.

In this instance you will make use of the Street Address to search for users.

Please note that the following parameter details have been provided. You will need to work between the ActionResult( ) and the Return( ) and the Index.cshtml and Details.cshtml Views.

```
Index(string sortOrder, string currentFilterTextbox, string textboxSearchString, int? page)
```

You will need to complete the following tasks which may require that you move between the Index View and he Activity's Controller. This is not necessarily required; however you might find that it might help a little with the coding process.

1. Add the associated connection to the database as you will be required to use the database as linked to the project by means of the Entity Framework Models.

2. Add the null exception If Statement that would check the content of the textbox on the View.

3. Add the necessary textbox on the View. You can do this later, but it is a little easier to ensure relevant naming conventions. In this case "Spelling" of variables such as ViewBags and the HTML Form Helper matter as you should access the correct details by means of the Form Helper.

4. If you wish to do so, you can add the relevant ViewBags. Please refer to point (2) referenced in the next section regarding this option.

5. Associate the filter ViewBag with the search string. The ViewBags will cast (send) the data from and to the Index View. A ViewBag is a way to pass data from the Controller to the View and from the View to the Controller. It is a type of object and is a dynamic property under the Controller global base class.

A general note on ViewBags: *Compared to ViewData (a dictionary or variable of sorts, that contains key-value pairs such as data types or lists), a ViewBag (generic dictionary) casts (sends) data, and the value being cast must be typecast (equal to, ConvertTo) to an appropriate type (int, string, double, decimal etc) before using it . This means that on the View and in the Controller, the ViewBag sends and receive data like an object type. It is not exactly the same as an object type, but like an object type, it can send and receive any type of data. However, in the Controller, parameters and / or variables are required for the ViewBag data. The ViewBag casts (sends) data, however you need to use an appropriate variable of the required data type that would receive the cast. For example, you might need to cast a ViewBag.studentrecord to List<xxxx.Models.Enrollment> if you are sending data that would eventually be used in a specific sequence, or you need to cast your ViewBag to a temporary variable with a distinct data type. If you do not do so, you will receive a ConvertTo ( ) data type or a null exception error. A ViewBag internally inserts data into a ViewData dictionary (data type variable) in the Controller.*

6. Before using a search lambda statement, you will be required to store the data from the relevant tables to temporary internal variables with a local scope. Essentially it would be private within the ActionResult and only applicable to the ActionResult in the Controller where it is being used. The tables that would be used is **(1)** locations, **(2)** departments, **(3)** employees and **(4)** jobs. You will search with a string that should not be empty. If the string is not null or empty, then the street_address is identified withing the local variable where the location data is being stored.

7. The street_address is identified so that the number of keys being used on the join on search is reduced to only the necessary keys that will be used for the join. This reduces the processing cost and overhead (time the CPU uses to process the join for the search). You will notice that on first load, the page loads somewhat slowly as it is returning all the data to the View. However, the joining on pre-selected keys would speed up the search process.

8. Create a LINQ statement that will use the key associated with street_address to join the x4 tables stipulated previously in point 6. The join will be sequenced, as always, in a [PK_ID] Parent-Child [FK_ID] relationships. You will need x3 temporary variables that is converted into a list as you are receiving a list of data from the relevant entities in the database. A database entity is essentially nothing more than a well formulated, structured, and sequential list.

HINT: *Take note of the relationship between the tables. The tables are being joined to the employees child entity that contain x3 foreign keys. You will be using x2 of the x3 foreign keys. In other words, the join will work in the following sequence:*

[PK] locations >> departments [FK]
[PK] departments >> employee [FK].
employees [FK] << [Pk] jobs.

3. Sorting and Filtering (re-use and update pattern).

To enable the sorting and filtering you will need multiple ViewBags. The ViewBags will keep the data persistent when the View undergoes a page change when sorting and searching is triggered by a user event. Make sure the ViewBag sort parameter name is exactly the same as the ViewBag sort parameter on the Index View. On the Index View, the ViewBag sort parameter is used in the table headings.

In the ActionLink, the first string is the heading display name, the second string is the View name as stipulated in the Controller, and then the ActionLink sort order is one of the ActionLink overloads. An overload is a set of sequential statements that calls functions from the IDE and whatever Software Development Kit (SDK) such as .NET or .NET Core that the IDE uses.

It does not matter if the ViewBags on the View and the ViewBag sort parameters are in the same sequence as the ViewBags cast based on the ViewBag name. It does however help as the casting would be sequential point-to-point and "random" casting would not occur. This reduces processing cost and overhead.

The ViewBag sort parameters used at the beginning of the ActionResult receives sorting details from the ViewBags on the View, and then stores it temporarily so that it can be used in the Case Select sort.

The ViewBag sort parameters is at the beginning of the Controller so that is receives data on load to reduce processing cost and overhear. Remember, a ViewBag works similar to an object data type. As such, it is slow in transferring data as it needs to cast based on potentially multiple data type options.

ViewBags casts on potential data type options so it in essence loops through most common to least common data types based on priority. Placing the ViewBag sort parameters as close as possible to the beginning of the ActionResult of the Controller therefore loads the data while RAM is still swapping relatively quickly with regards to the CPU, resulting in the load occurring faster as more RAM and CPU resources is still available. The Switch Case Select will eventually use the data in the ViewBag sort parameters.

The Switch Case Select is after the search and the join as the search and join uses Model predefined data types. It contains a lot of data, but as the data being transferred is narrowly defined, the processing overhead is less than the processing overhead for the ViewBags. However, as the processing overhead is still high, but lower than the ViewBags, it would be after the ViewBags.

The Switch Case Select uses the least processing overhead and would therefore be near the end of the ActionResult. It would then use the remainder RAM and CPU swapping overhead.

A Switch Case Select is a standard selection statement that can be populated with any type of process application code. A Switch Case Select is activated by means of a string pattern. Make sure that the ViewBag parameters and the string pattern in the Case is the same.

Spelling and capitalisation matter here as the Case Select only looks for a pattern. One small typo or spelling variation and the pattern will not be the same and would therefore not work. The Case Select would then exit to the default (at the end of the Switch) and then your search and sort will reset to the default with all the data that can be loaded from the LINQ join.

The break in the Switch is an exit option. So if a lambda statement is executed, then the Switch breaks (such as in "stop") and then exits from the rest of the Select. Sequence the Switch Case Select in the most common to least common options as these matter in terms of processing overhead.

Carefully consider what a user might do from left to right on the View as this is culturally sensitive (form example where people read from left to right). If a culture reads from right to left, then arrange your Switch from what would be displayed on the right side to the left side of the View.
Complete the following tasks as required for sort and filtering:

1. Update the ViewBag sort parameter to sort and filter based on **(1)** Name, **(2)** Surname, **(3)** Address, **(4)** Email, **(5)** Department and **(6)** JobTitle. Make sure that the spelling associated with the parameters are the same as the spelling to be used on the Switch. Make sure that (eventually) the necessary details on the Index View are the same as the ViewBag sort parameters.

2. Update the Switch based on the sequence of the ViewBag sort parameters. As this is fairly straightforward, refer to the examples as to how one would organise your code. Refer to point 1 one paragraph above to note the sequence regarding the OrderByDecending and the OrderBy in the Switch.

3. Update the ActionLinks on the Activity Controller's associated Index View so as to display the data indicated in point 1 to paragraphs above.

4. If you so wish, update the DisplayFor by referencing the appropriate Model Items. You will eventually be required to do so to enable functionality.

4. Joining 4 tables (re-use and update pattern).

The joining of the x4 tables have already been described in section 1. What is of importance in this section is as follows:

1. Remember the [Pk] Parent-Child [FK] relationship. If the [Pk] Parent-Child [FK] relationship sequence is incorrect, then the View will display one column of data. If the [Pk] Parent-Child [FK] relationship is used in the appropriate sequence, then all the relevant data as associated with **(1)** Name, **(2)** Surname, (3) Address, **(4)** Email, **(5)** Department and **(6)** JobTitle will display correctly.

2. Make sure that the variable receiving the data from the join is the same as the variable that is returned by the PageList result. The sequence of the PageList return is a standard sequence as defined by the NuGet package hosted on GitHub.

3. The general sequence for the join in the required LINQ statement is as may be seen below. Please note the number of "Join" "From" options. It is directly influenced by the number of tables. There will be one "Join" "From" for each two tables being traversed by means of a two-by-two table relationship.

```
var ……. = ( from
            join …….. on …….equals ……. into …….
            from …….
            join …….  in ……. on …….. equals ……. into …….
            from ……. in …….
            join …… in ……. equals ……. into …….
            from ……. in …….
            select new Model (in this case Activity which contains the Detail Properties)
              {
                  Detail Property from Model = …… ,
                  Detail Property from Model = …… ,
                  Detail Property from Model = …… ,
                  Detail Property from Model = ……
              });
```

It is important to remember that the LINQ statement is not the only approach that can be followed. You may make use of an include lambda statement to achieve the same result. No support regarding this option will be provided as there is sufficient examples in the linked project as found on the Background View.

An example for such a statement would be as follows:

```
var ……. = db. ……. Include(xx => xx. ……. ).Include( xx => xx. …….)
            .Select ( yy => new
             {
                ……. = xx. …….,
                ……. = xx. …….
             });
```

5. Updated and finalize functionality. Update the Views for ActivityController.cs.

For this section it will be required to finalise the pattern to ensure functionality. The Controller needs to be finalised based on the pattern identified in the sample controllers.

As part of the activity, add a Details ActionResult. The Details ActionResult is the same standard Details code used in an EF Scaffolded set of application code. The only difference would be the main table being accessed. In this case it would be the employees table as found in HumanResourcesDB. Add an empty Details View by right clicking on the word "Details" next to the ActionResult and then adding the empty View.

Finalise the Details View by making sure that the following attributes from the **(1)** Name, **(2)** Surname, **(3)** Address, **(4)** Email, **(5)** Department and (6) JobTitle tables are displayed.

- **first_name** >> From the Employees Table.
- **last_name** >> From the Employees Table
- **street_address** >> From the Locations Table.
- **city** >> From the Locations Table.
- **state_province** >> From the Locations Table.
- **country_name** >> From the Countries Table (indirectly access without the controller).
- **region_id** >> From the Regions Table (indirectly access without the controller).
- **phone_number** >> From the Employees Table.
- **hire_date** >> From the Employees Table.
- **salary** >> From the Employees Table.
- **email** >> From the Employees Table.
- **department_name** >> From the Departments Table.
- **job_title** >> From the Jobs Table

For a few pointers in terms of some of the details mentioned above in all the previous sections, please refer to the rubric listed below as it indicates some of the relationships to be applied in all the task indicated above.

ONE FINAL NOTE:

Please take note how PageLists are implemented in all the Views. This is a standard approach. The Model that is being referenced regarding the PageList is the core Model containing all the details onto which all the joins will take place.

INF272

18. Campus Practical -Entity Framework 2.

Due: 2022/10/03 before 23:59 loaded on ClickUP.

(____ / 20 marks)

REQUIREMENT	MARK
1. Database connection.	**1**
2. Searching (re-use and update pattern). Use the street address to search.	**4**
3. Sorting and Filtering (re-use and update pattern). Use the following details: **(a)** EmployeeDetails.first_name, **(b)** EmployeeDetails.last_name, **(c)** LocationDetails.street_address, **(d)** EmployeeDetails.email, **(e)** DepartmentDetails.department_name and **(f)** JobDetails.job_title.	**4**
4. Joining 4 tables (re-use and update pattern). Use the following details: **(a)** locations, **(b)** departments, **(c)** employees, and **(d)** jobs.	**6**
5. Updated and finalize functionality. Update the following Views (ActivityController.cs): **(a)** Index.cshtml, **(b)** Details.cshtml.	**5**
TOTAL	**20**