### תיאור קצר של הפיצ'רים שבחרנו לממש בתרגיל הקודם:

* **Youtube Posts Feature** - A feature that allows the user picking a specific facebook friend and search all his posts containing youtube clips, play it, add clips to favourite list.
* **Laetest Pages Posts** – A feature that allows the user getting a quick view on the latest liked-pages posts of all of his friends. Only the latest posts are picked and sorted right after.

### שכלולים של הפיצ'רים והרחבת המערכת:

* **Youtube Posts Feature** - Adding the option to download the chosen youtube clip, adding the option to drag and drop it into the desktop. Adding the options to Serialize to Json format as well.
* **Laetest Pages Posts** – By using a remote server (rented from DigitalOcean and configured in linux ubuntu server) the users are now able to complain on a specific page post – so that when the number of users that complain on this post reach 10, an automatic mail is sended from the application email to the page owner mail. (information about user sessions, page posts and so on is saved on the database).
* **Application Configuration** – when there is not config.json file the config.json file is automatically created from default parameters. However, if there exsists a config.json file, the application configurations is fetched from this file (for example, number of maximum pages posts downloaded from facebook service). Implementation: ApplicationConfiguraion.cs,JsonConfiguraionApplication.cs
* **Basic Logger** – writes a log file mentioning db connections.

### תבנית מס' 1 – Caching Proxy

* סיבת הבחירה / שימוש בתבנית:

We implemented the caching proxy design pattern in our Latest-Pages-Posts feature. The process of fetching the posts from pages of a specific facebook user takes time. Usually 2-5 seconds. This delay hurts the UX of the application and also makes it less scalable. (In a scenario where the number of facebook friends and pages posts is enourmous, the application in its current position will not statisfy.) Therefore we save the required pages posts of user's liked pages in a special data structure inside a proxy class so when another request for the same data is being submitted, we will not need to reload the data from Facebook server; we will take them from our cache of pages posts. We chose to implement the pattern by an interface and a composition. The Interface ( IFetcherOfPagesPosts ) and two classes that implement it ( ProxyCacheOfPagesPosts class and FetcherOfPagesPosts class. ) where ProxyCacheOfPagesPosts "has a" FetcherOfPagesPosts instance.

**Advantages:**

Performance & Scalability: Using a cache improves loading-from-server time considerably (after the first time we download the data from Facebook, the next time we need them they are already available in our cache). This improves user experience and gives faster results. Moreover, the more data and users the application will handle have to handle, the more useful this cache will become.

Offline Working: We may be able to work offline (or at least see the information that we already got, because it is saved in our cache). It is also possible to turn the cache which is currently saved only in variables and data structures, into a local database for future use as a long term cache.

* אופן המימוש:

Relevant Classes:

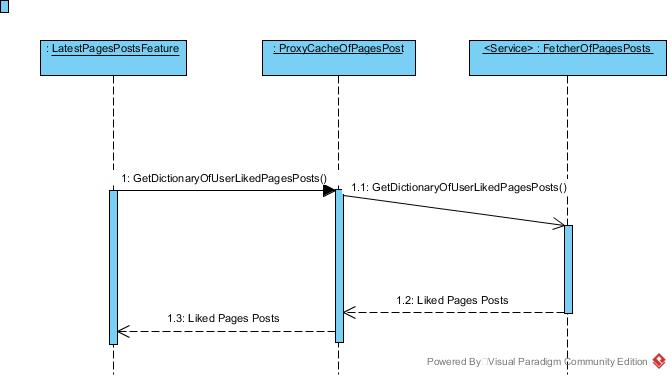
**IFetcherOfPagesPosts**

**FetcherOfPagesPosts**

**ProxyCacheOfPagesPosts**

**LatestPagesPostsFeature (the Client of the proxy).**

**Sequence Diagram**



**LatestPagesPostsFeature – Client**

**IFetcherOfPagesPosts – Subject**

**ProxyCacheOfPagesPosts– Proxy**

**FetcherOfPagesPosts – Real Subject**

### C:\Users\nadav\AppData\Local\Microsoft\Windows\INetCache\Content.Word\proxy.jpgClass Diagram

### תבנית מס' 2 – Factory Method (Static factory class)

* סיבת הבחירה / שימוש בתבנית:

**Description:** We implemented the static factory class design pattern (simple factory) using the class FacebookFeatureCreator. This class implements an interface (IFacebookFeature). When we want to add a new feature to the app, we may solve this future problem by simply calling the static CreateFacebookFeature() method in the static FacebookFeatureCreator class (we only need to specify a feature name which is represented as an enum in the class eKindsOfFacebookFeature)

**Advantages:**

Single Responsible Principle: The factory does seperate between the internal logic of the feature and the logic of the "creation process" of the feature. This way the new feature is added is by calling the CreateFacebookFeature() method from the logic, which returns an instance of the feature object, and therefore the create responsibily, logic responsibility are splitted.

Maintainability: When making changes to our system, this pattern allows us to only make changes in the factory method class (only in one place), instead of specifying changes in every place that defined a creation of a feature. This prevents some bugs caused by forgetting code changes and also prevents code duplication. The solution is to force the creation of objects to occur through a common factory rather than to allow the creational code to be spread throughout your system. If all the code in your program must go through this factory whenever it needs to create one of your objects, then all you must do when you add a new object is to modify the factory.

Usability: The creation is not the user’s responsibility, instead of knowing how to create the object itself and dealing with the "new" keyword in a specific manner, the only thing the user does is asking and receiving an instance of facebook feature. This kind of encapsulation helps the user to concentrate on the logic.

More advantages:

* Allows you to easily test certain parts of your application so you can build and test the other parts.
* Allows you to change the design of the application easily and readily.
* Less loose-coupling.

אופן המימוש:

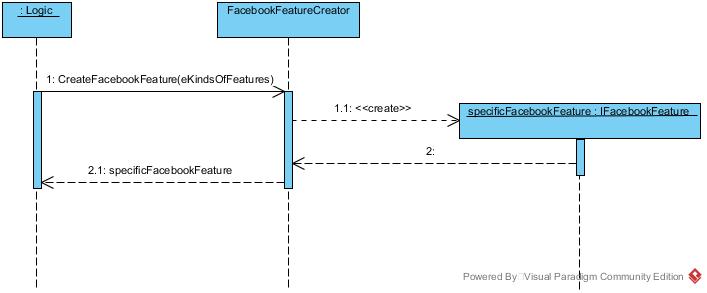
Relevant Classes:

**eKindsOfFeature (enum)**

**IFacebookFeature (interface)**

**FacebookFeatureCreator (class)**

**Sequence Diagram**



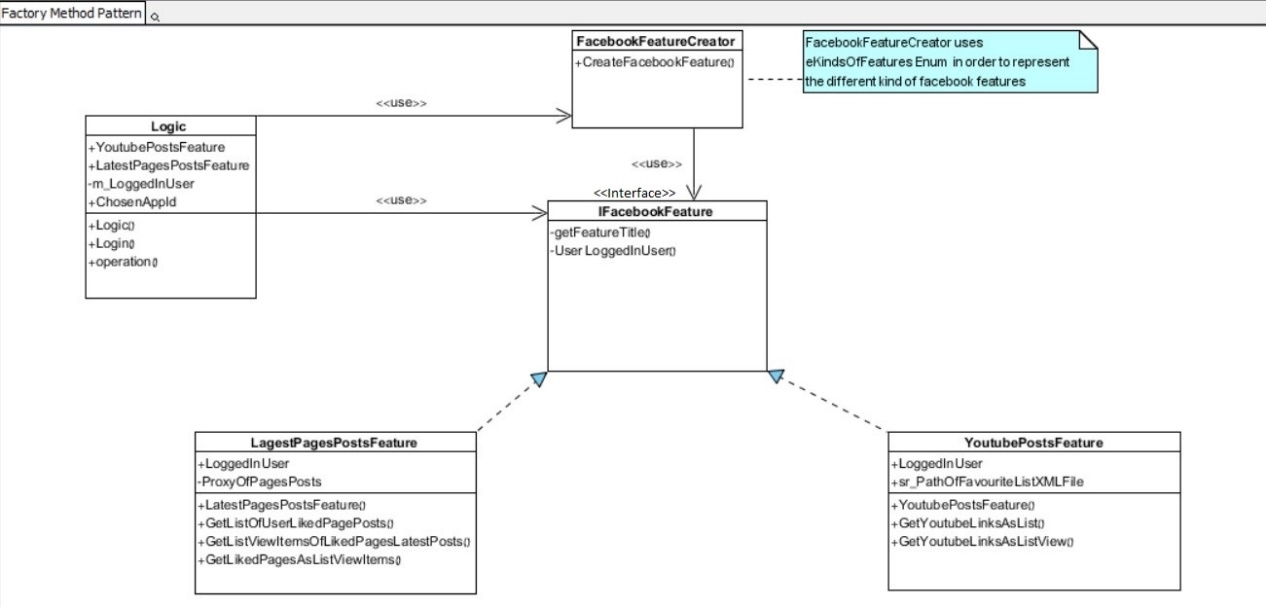
Class Diagram

**Logic – Client**

**FacebookFeatureFactory – Creator & Concrete Creator**

**IFacebookFeature – Product**

**YoutubePostsFeature – Concrete Product**

**LatestPagesPostsFeature - Concrete Product**

### תבנית מס' 3 – Transparent Facade

* סיבת הבחירה / שימוש בתבנית:

**Description:** In the latest development of the application, we added an option to view all user sessions of the applications. As part of dealing with the Mysql API which is made up of different classes and subsystems we wanted to make it easier for us the developers to deal with the database logic. As a result, there is an easier "language" to communicate with the database. Creating a layer to abstract and unify the related interfaces in the application. Use of a facade will define an entry point to each subsystem level and thus make them communicate only through their facades;

**Advantages:**

Maintainability: It reduces dependencies of the external code on the inner working of the libraries and thus providing flexibility. In the above structure for Façade pattern the Façade class insulates the subsystem from the client. The client only interacts with the Façade class without knowing about the subsystem classes so when ever the developer wishes to use the application database, there is less place for errors and more straight-forward ways to achieving different functionalities.

Security: The client is using the db without having knowledge of how the internal database connection API looks like. Once the request is being submitted the façade class layer calls the methods of the mysql subsystems for processing of the databse. After processing it returns the control to the client class with the confirmation about the order being processed. More over, if the json configuration file which is at the moment fetched from the user harddisk, will be fetched from the app DB, then the security will be even tighter thanks to this façade.

More Advantages:

* make the code more readable, and more adjustable.
* reduce dependencies of outside code on the inner workings of a library, since most code uses the facade, thus allowing more flexibility in developing the application,
* avoid dealing with the many my-sql classes and hard-to-maintain architecture.( with a single API.)

.

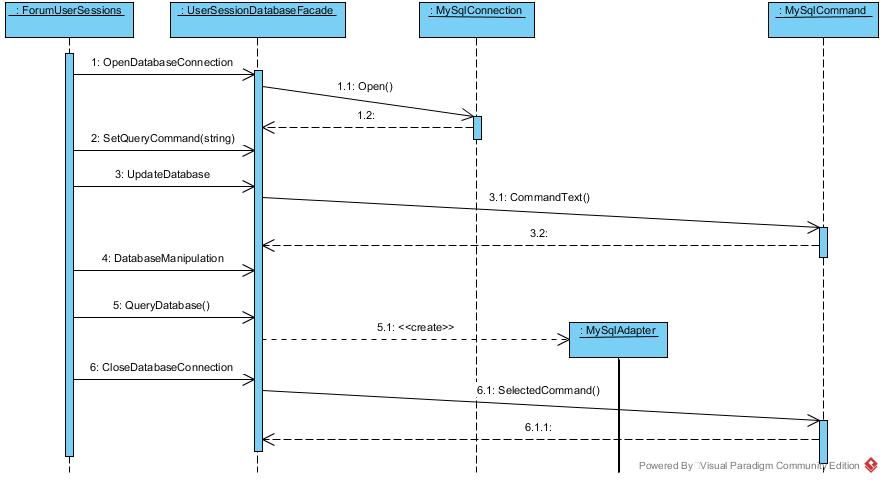
* אופן המימוש:

Relevant Classes:

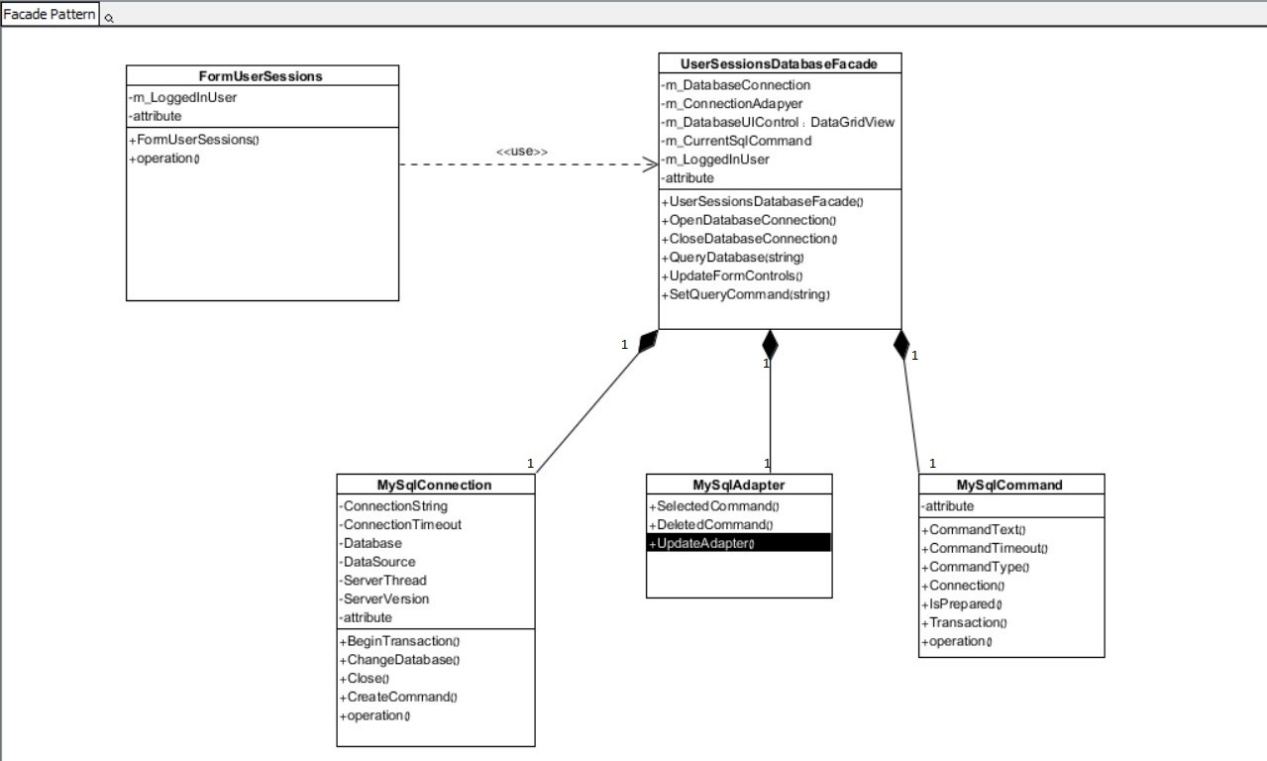
**UserSessionsDatabaseFacade**

**FormUserSessions**

Sequence Diagram



Class Diagram

****

**Asynchronous Programming**

In the pages posts form of the application ( FormPagesPosts ), we used the Thread instances for making sure any delay caused by ongoing fetching from the facebook server will not interrupt the UX and the user of the application will be free to do whatever he wants until the data is finished-loading. The problem of slow fetching of the liked-pages-posts (partly solved with the proxy pattern) is also being solved by using the proxy pattern but that does not solve the UI delay so multi-threading is a must in this case.

Moreover, each time the user chooses to complain on a specific page post, a db query operation is taking time and without opening anoher thread for that, the UI will get stuck as well.

Where In The Code: line 827 in FormPagesPosts

Line 781 in FormPagesPosts

Line 820 in FromPagesPosts

Line 56 in FormPostInfo

**Data Binding**

The connection to a database (added recently) which allow the user to view user sessions of the application is viewed to the user by a GridView control. This control has a data source initialized to be the databse we are using. As a result, the developer does not have to worry about the logic of transforming each one of the database rows and columns into a viewable object. Moreover, if in the future, the databse will be changed, not a single line of code would have to be re-written because the database will still be just a data-source of the same GridView control.

Where In The Code: Line 231 in UserSessionsDatabaseFacade