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

* **Random friend liked pages:** From a list of friends, who are also using this application, pick randomly one friend and see 10 pages that this friend likes. The pages are shown as a grid of pictures 2 \* 5.
* **Find friends by city:** This feature helps the user to see in his list of a friends, who are also using this application, only the friends in a specific location (provided by the user).

### תבנית מס' 1 – Caching Proxy

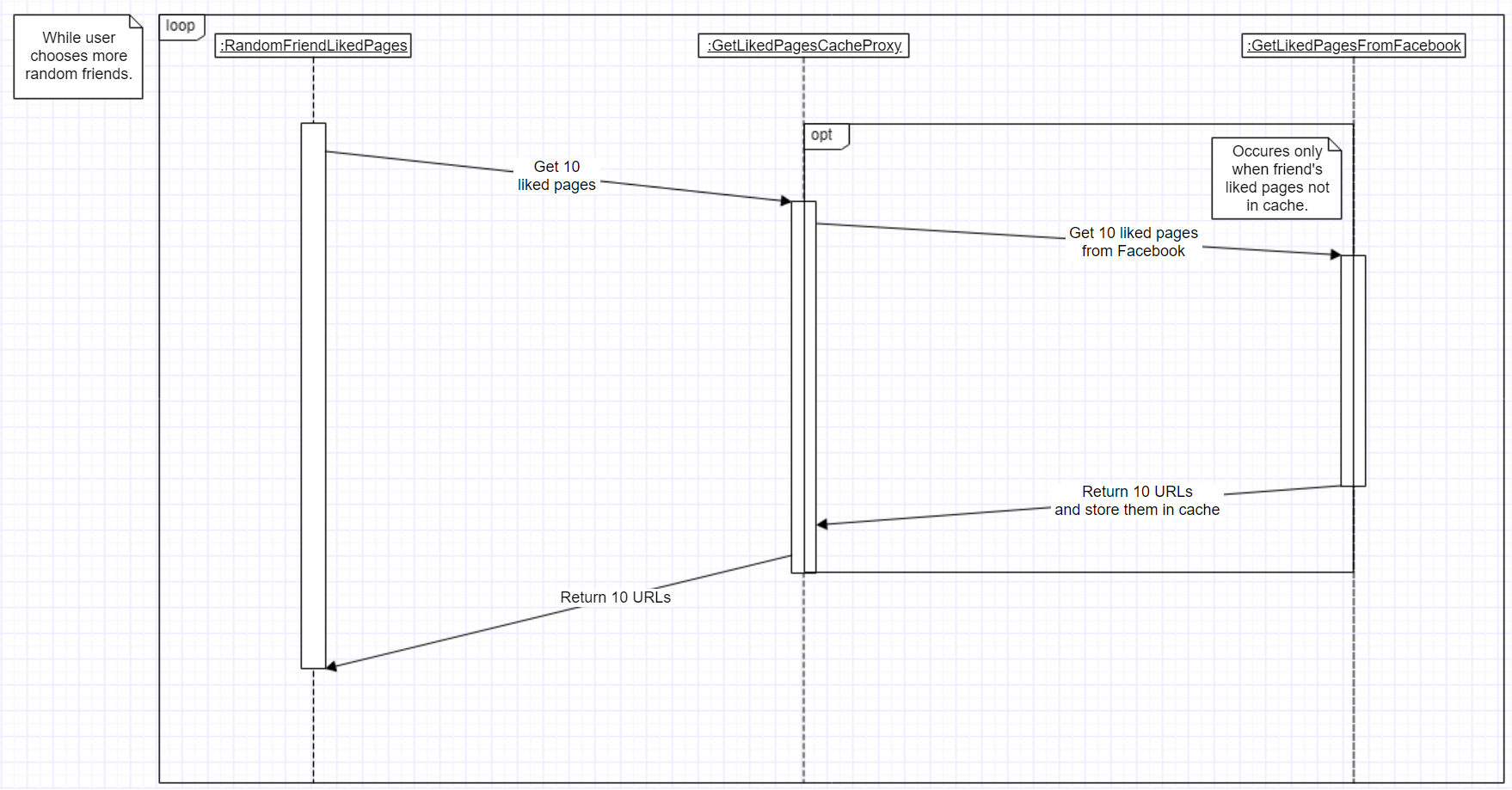
* סיבת הבחירה / שימוש בתבנית:

We’ve decided to implement Caching Proxy pattern for RandomFriendLikedPages feature.  
When we get our randomized friend info about his liked pages (picture of each page), we would like to store that data for possible next time (if user gets same random friend in the same session). That gives us the option to show same pictures even if user had lost connection to server (only the pictures that are stored in cache) and it gives faster results (time efficiency) to user which improve user experience.

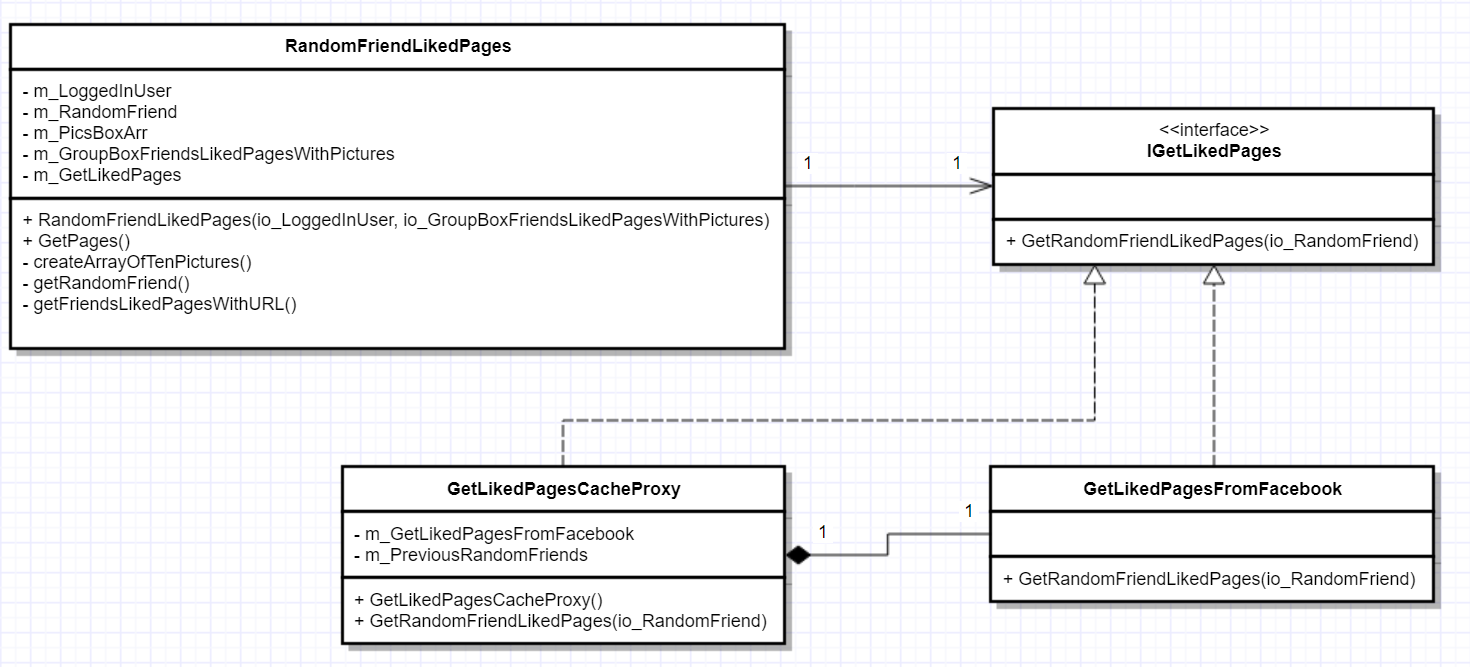
* אופן המימוש:

**Relevant classes:** IGetLikedPages, GetLikedPagesFromFacebook, GetLikedPagesCacheProxy, RandomFriendLikedPages.

* Sequence Diagram



* Class Diagram

**  
Client:** RandomFriendLikedPages.  
**Subject:** IGetLikedPages.  
**Real Subject:** GetLikedPagesFromFacebook.  
**Proxy:** GetLikedPagesCacheProxy.

### תבנית מס' 2 – Factory Method

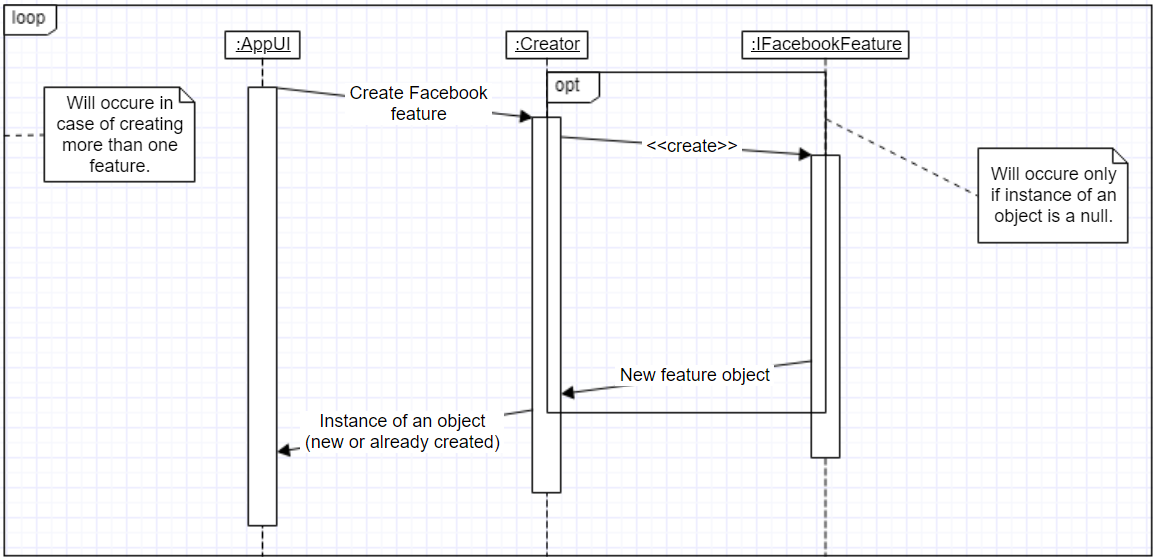
* סיבת הבחירה / שימוש בתבנית:

We’ve decided to implement Factory Method pattern in case of additional future features.  
Instead of writing the whole code over again in case of a new feature, we can just implement IFacebookFeature and add new functions. Every concrete creator contains a singleton pattern because we don’t want to create multiple instances of same objects. Also, FriendsByRequest feature is a form (hence it is inheriting from class Form), so we’ve created a wrapper (FriendsByRequestWrapper) that allows us to hold this feature as property and in the same time implement IFacebookFeature interface.

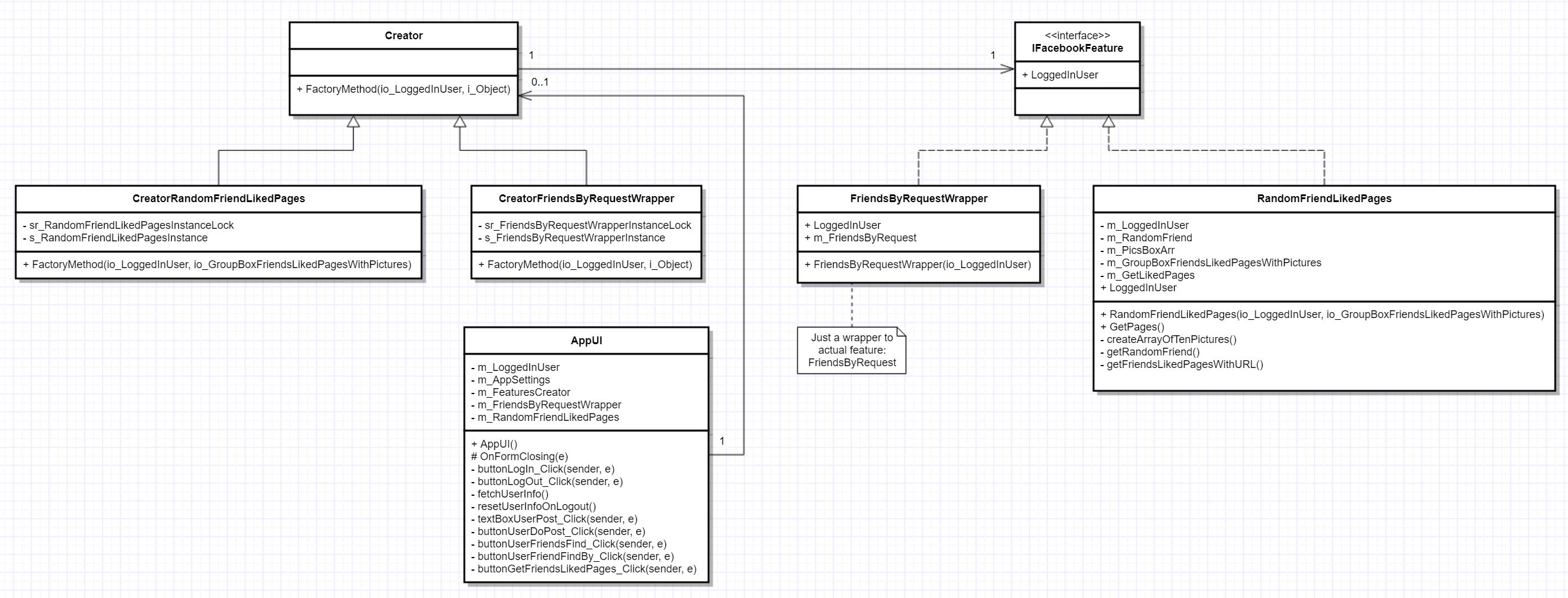
* אופן המימוש:

**Relevant classes:** IFacebookFeature, RandomFriendLikedPages, FriendsByRequestWrapper, Creator, CreatorRandomFriendLikedPages, CreatorFriendsByRequestWrapper, AppUI.

* Sequence Diagram



* Class Diagram

  
  
**Product:** IFacebookFeature.  
**Concrete Products:** RandomFriendLikedPages, FriendsByRequestWrapper.  
**Creator:** Creator.  
**Concrete Creators:** CreatorRandomFriendLikedPages, CreatorFriendsByRequestWrapper.  
**Client:** AppUI.

### תבנית מס' 3 – Facade

* סיבת הבחירה / שימוש בתבנית:

We’ve decided to implement the Facade Pattern for FriendsByRequest feature.

Hence the feature is divided in to two sub features: Friends in a given city & Single friends.

It would be more comfortable for use and maintainable to divide them in to separate subsystems,

where each of sub features will be given a base class implementation and an interface as the Facade to give simplification for the usage.

* אופן המימוש:

**Relevant Classes:** FriendsByRequest, IInformationGather, SingleFriendsFinder, FriendInACityFinder.

* Sequence Diagram



* Class Diagram



### Asynchronous Programming

We have used asynchronous programming in following classes: AppUI, RandomFriendLikedPages.  
In AppUI we’ve implemented it in buttonLoginClick method, which starts in a new thread fetchUserInfo method. Last mentioned method uses invoke on groupBoxUserInfo with lambda expression because fetchUserInfo uses textboxes (thus we can’t control them from the thread they have not been created on).  
In RandomFriendLikedPages we are using a new thread to run getFriendsLikedPagesWithURL from the GetPages method. We’ve decided to do this in a different thread, because for every random friend we are getting ten liked pages and for every page we ask the server for an image.  
All of the above has been done to improve user experience and give him an option to keep interacting with the application window even there are multiple threads working at the background.

### Data Binding

We have used data binding in AppUI form.  
The data source is connected to m\_LoggedInUser.Friends at buttonUserFriendsFind\_Click method and this data can be seen in User Friends section. We’ve decided to implement data binding here because it’s more efficient (less code to write and less possible mistakes and bugs) to use data binding instead of doing placement for every field by ourselves.