A20 EX01 Alex Girshfeld Tal Goldstein

# Introduction

“Facebook Influencers Toolkit" is a Facebook application implemented by us as part of Design Pattern course.

The application supplies a basic Facebook experience alongside supplying dashboards to allow the user to understand and analyze the influence of his "posts".

# Basic Facebook features:

1. Watch your posts - you can also preview their content in the post preview box. (the post you select to preview is the one that the influence preservation\expansion grades will be calculated to)

2.Watch your albums.

3.Watch your friends.

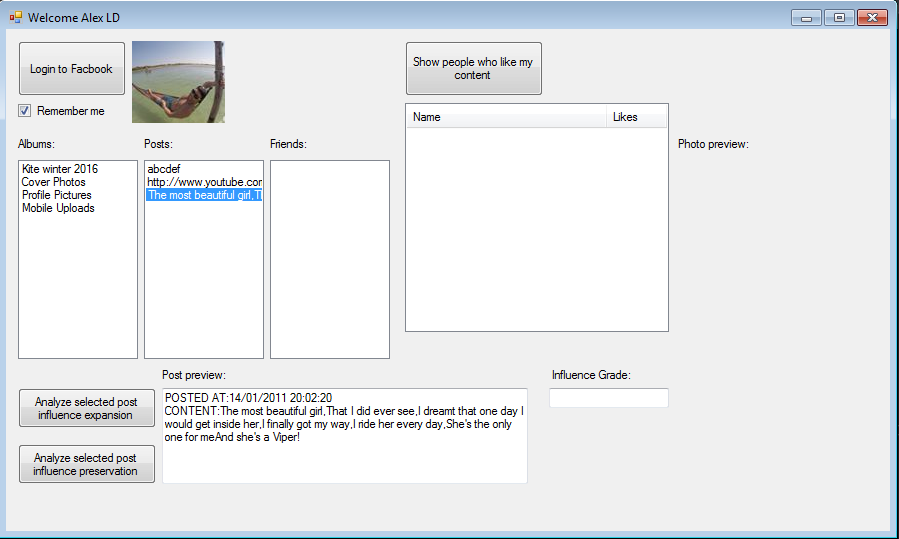
4.You can ask the application to remember your settings (window size) and to keep you logged in. It'll save your settings in your "C:\Users\Public" directory.

# Influence features:

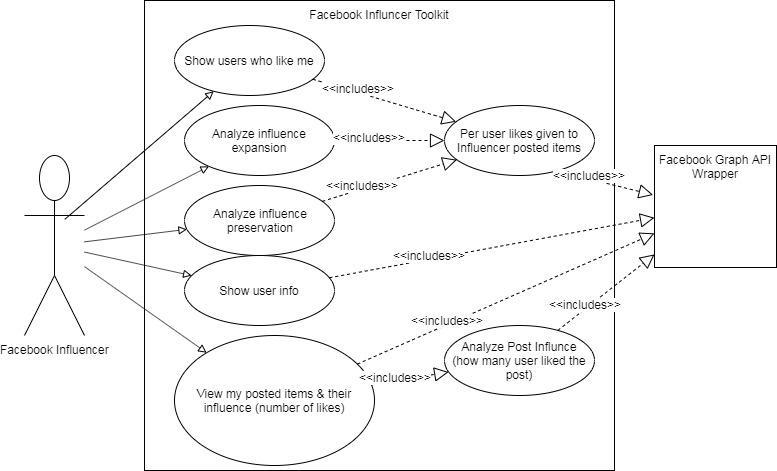
1. "Who likes me most?" - A sorted list of you friends who like you the most by the number of total likes given to your post
2. "influence preservation grade per post" - 0-100 grade of you post in order to preserve your current "fan base" of "likers" that means that if the people who always like you post didn't like this one it's "preservation grade" will be significantly lower.
3. "Influence expansion grade per post" - 0-100 grade of your post in order to expand your influence. meaning that if you want that people who before gave you less likes will now give you more likes.

\*In the Influence preservation\expansion feature there are two factors: A quantitative & a qualitative factor. The first factors the number of likes compared to the average amount of likes you post receives and the latter takes into account **who** liked this. Example: If he never liked any of your posts he'll improve the influence expansion grade.

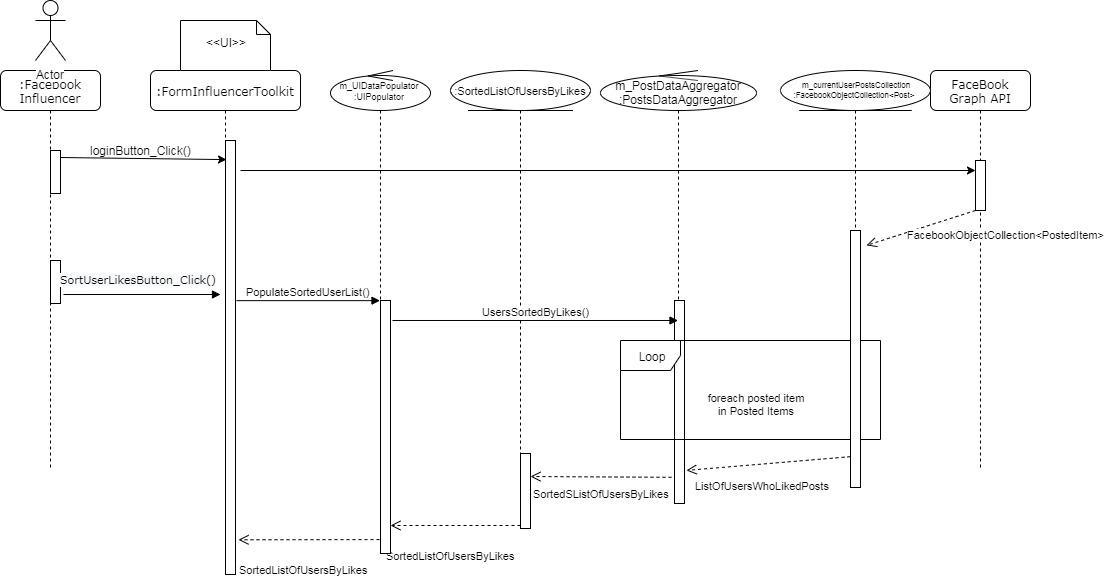
\*For features 2+3 you have to select the post you want to analyze from the list of posts and click on the corresponding button in the bottom left corner of the window. The selected post was is highlighted in blue:



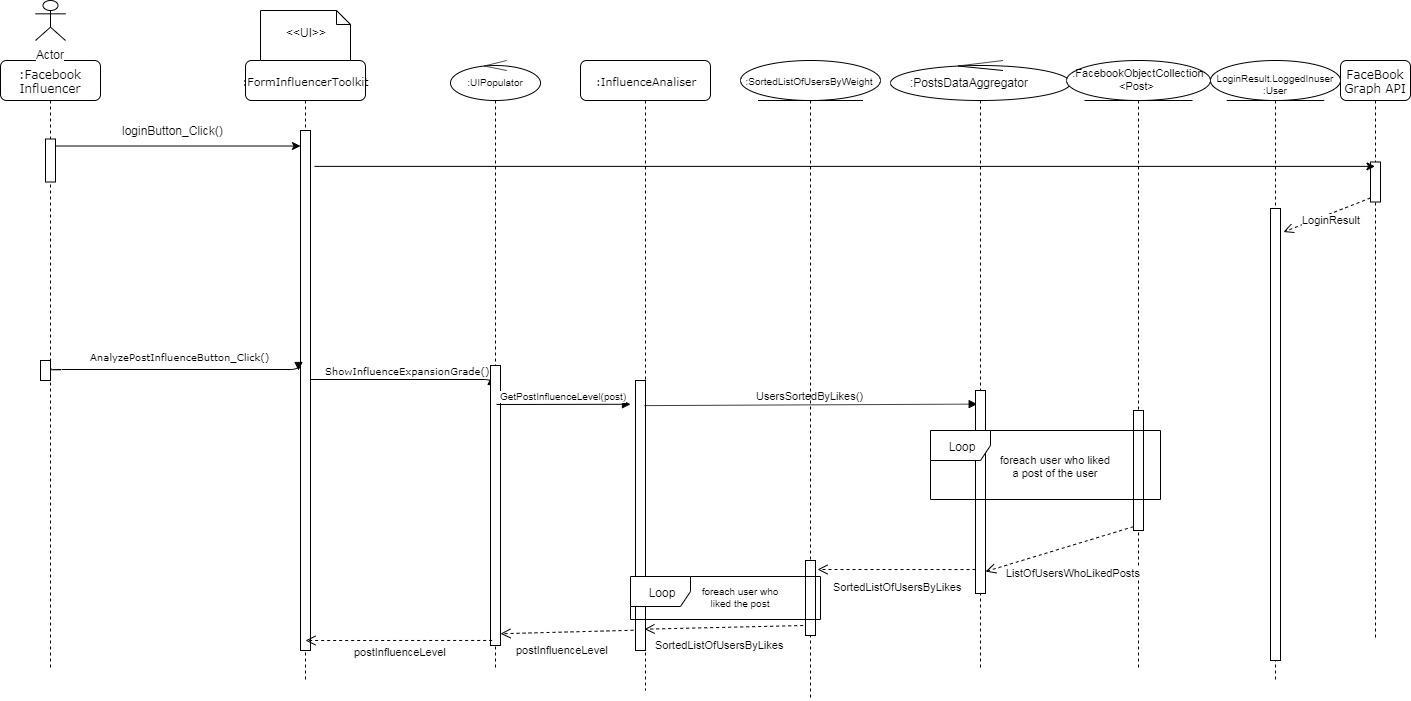
# Sequence Diagram of "Who likes me most?"



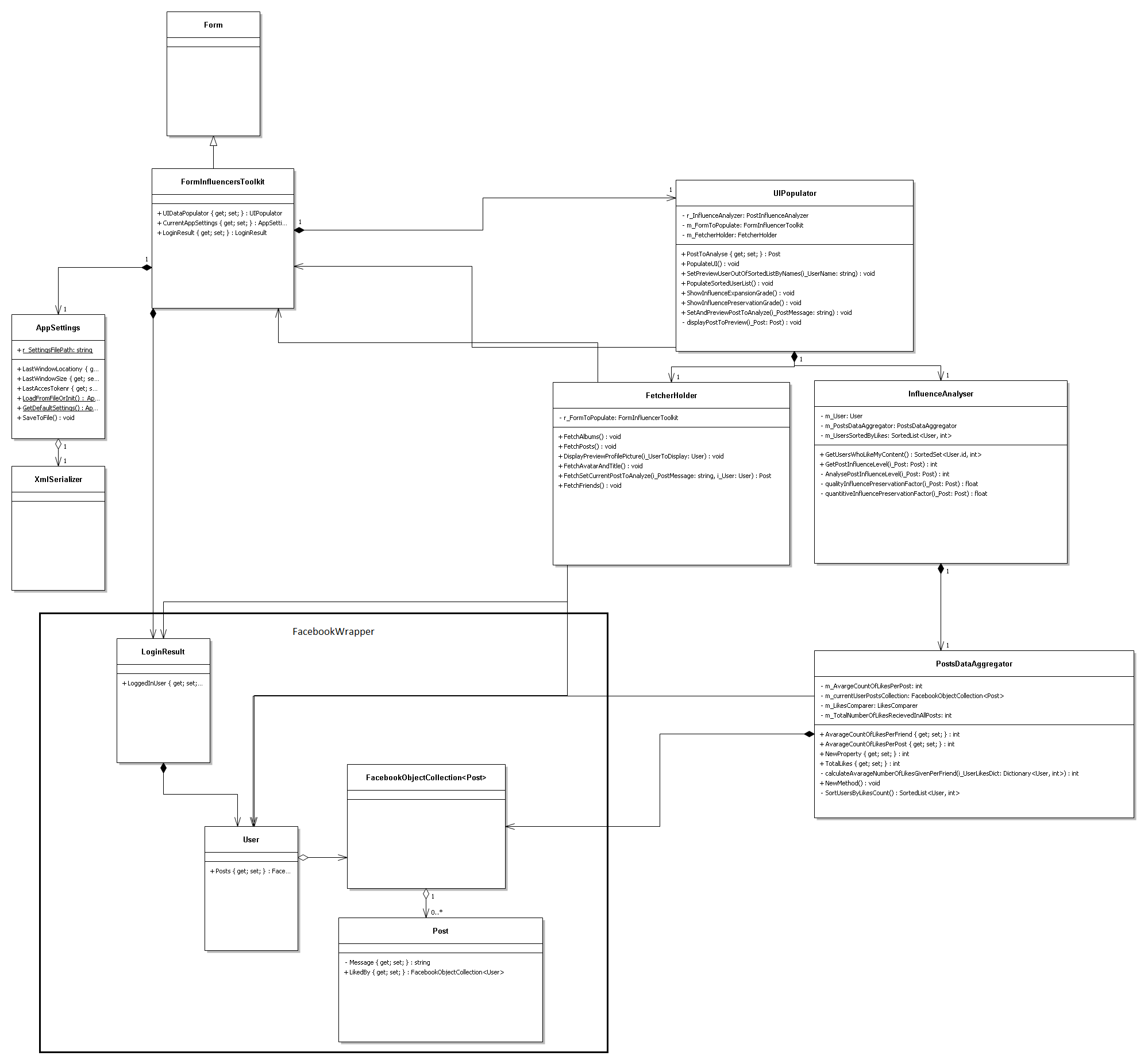
# Sequence Diagram of "Who likes me most?"



# Sequence Diagram of "Influence preservation grade per post?"



# Class diagram



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

* ״Who likes me most?״ הינו פיצ׳ר המאפשר למשתמש לקבל רשימה המכילה את כל המשתמשים אשר אי פעם סימנו ״אהבתי״ (Like) לאחד או יותר מהפוסטים שפורסמו על ידו. בנוסף, הרשימה מסודרת לפי מספר הסימונים אותם עשה כל משתמש, ובכך מאפשרת למשתמש המבקש לראות מי הם המשתמשים אשר עוקבים אחר התוכן אותו הוא מפרסם ואוהבים אותו יותר מבין עוקביו.
* "Influence preservation grade per post" הינו פיצ׳ר המאפשר למשתמש לבחור פוסט ספציפי מבין כל הפוסטים אותם פרסם,ולקבל עליו ציון בין 0 ל-100 לאחר ניתוח השפעת אותו פוסט נבחר על קהל עוקביו של המשתמש. ככל שיותר משתמשים אשר עוקבים אחריו באופן קבוע סימנו Like לאותו פוסט, כך ציונו יהיה גבוה יותר בכך שעזר לשמר את השפעתו האינטרנטית של המשתמש על קהל עוקביו בכדי שימשיכו לעקוב אחר התוכן אותו הוא מפרסם.
* ״Influence expansion grade per post״ הינו פיצ׳ר משלים לנ״ל, המאפשר למשתמש לבחור פוסט ספציפי מבין כל הפוסטים אותם פרסם,ולקבל עליו ציון בין 0 ל-100 לאחר ניתוח השפעת אותו פוסט נבחר על קהל עוקביו של המשתמש. ככל שיותר משתמשים אשר אינם מסמנים Like באופן קבוע לפרסומים של המשתמש אכן סימנו Like לפוסט, ובפרט משתמשים אשר מעולם לא סימנו Like לאף פוסט של המשתמש, כך ציונו של הפוסט גבוה יותר, בכך שעזר להרחיב את השפעתו האינטרנטית של אותו משתמש ועזר לו להגיע לקהלים חדשים נוספים ובכך לאסוף משתמשים חדשים אשר יתווספו לקהל עוקביו.

### **תבנית מס' 1 – Adapter**

* **סיבת הבחירה / שימוש בתבנית:**

החלטנו לממש Caching proxy עבור הLoginResult דרך Serialization. מכיוון שDLL שמתממשק עם Facebook הינו legacy component שייצר אובייקטים שהם לא ניתנים לSerialization (גם לא ע"י עבודה עם [XmlIgnoreAttribute](https://docs.microsoft.com/en-us/dotnet/api/system.xml.serialization.xmlignoreattribute?view=netframework-4.8) ) אז החלטנו לייצר ADAPTER שיאפשר לנו לקחת אובייקטים שמגיעים Legacy component ולא מתאימים לlegacy component אחר ולהתאים ביניהם ע"י שימוש בadapter pattern עבור הרכיבים הבאים:

* LoginResult:
* User
* Post
* Album

* **אופן מימוש:**

מומשו 5 מחלקות

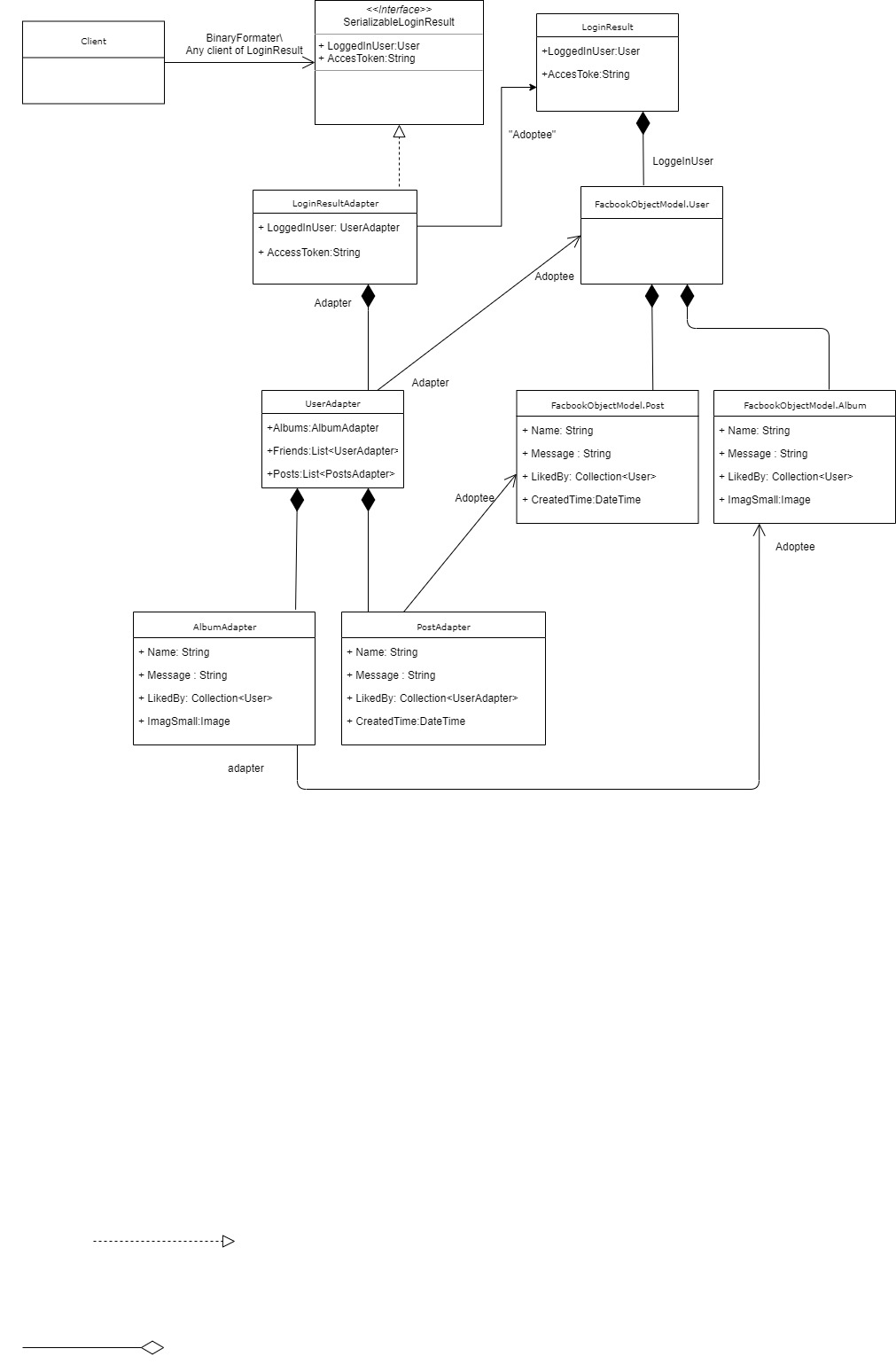
* LoginResultAdapter
* UserAdapter
* PostAdapter
* AlbumAdapter
* CollectionAdapter

4 המחלקות הראשונות הן Adapters "מן המניין" והמחלקה החמישית היא Static אשר משמשת ל"תרגם" collections<T> ל Lists<adaptedT>

* Sequence Diagram

[sequence diagram שמציגה את התהליך שקשור לתבנית והאינטראקציה בין המחלקות]

* Class Diagram



אבל הדרישה עבור כלל Clients של LoginResult היא לשמור על הproperties המקוריים של LoginResult עבור האפקליציה ולאפשר סיריאליזציה ושמירה על דיסק (מומחש ע"י הInterface). ה Interfaceבדיאגרמה הזו ממחיש את התכונות האלה. אנחנו ממשים Adapter מאותן סיבות גם לאובייקטים Album, User, וPost ומשתמשים באובייקטים החדשים להרכיב את הLoginResultAdapter.

### **תבנית מס' 2 – Proxy**

* **סיבת הבחירה / שימוש בתבנית:**

החלטנו להרחיב את הפונקציונאליות של האפלקציה ע"י מתן אפשרות לעבוד OFFLINE. אם המשתמש סימן את RemmemberMe האפליקציה תשמור את האובייקטים הרלוונטים בCACHE וכך בפעם הבאה שהמשתמש יפעיל את האפליקציה – הצגת הנתונים לא יהיה תלוי בחיבור שלו לאינטרנט.

בשביל לתמוך בפונקציונאליות כזו הפתרון המתבקש הוא כמובן Caching Proxy לאובייקט הרלוונטי לאפליקציה:

LoginResultAdapter – שהוא בעצם גרסה מותאמת של הLoginResult המקורי שמגיע הDLL הlegacy. הגרסה המותאמת הזו מאפשר Serialization .

כלל האובייקטים באפליקציה ימשיכיו לבקש מהPROXY את אותם דברים שהם קיבלו מהLoginResultAdapter המקורי. רק שאם לא בוצע תהליך חיבור – הProxy יתן להם את הנתונים ששמורים לוקאלית במנגנון CACHING.

* **אופן המימוש:**

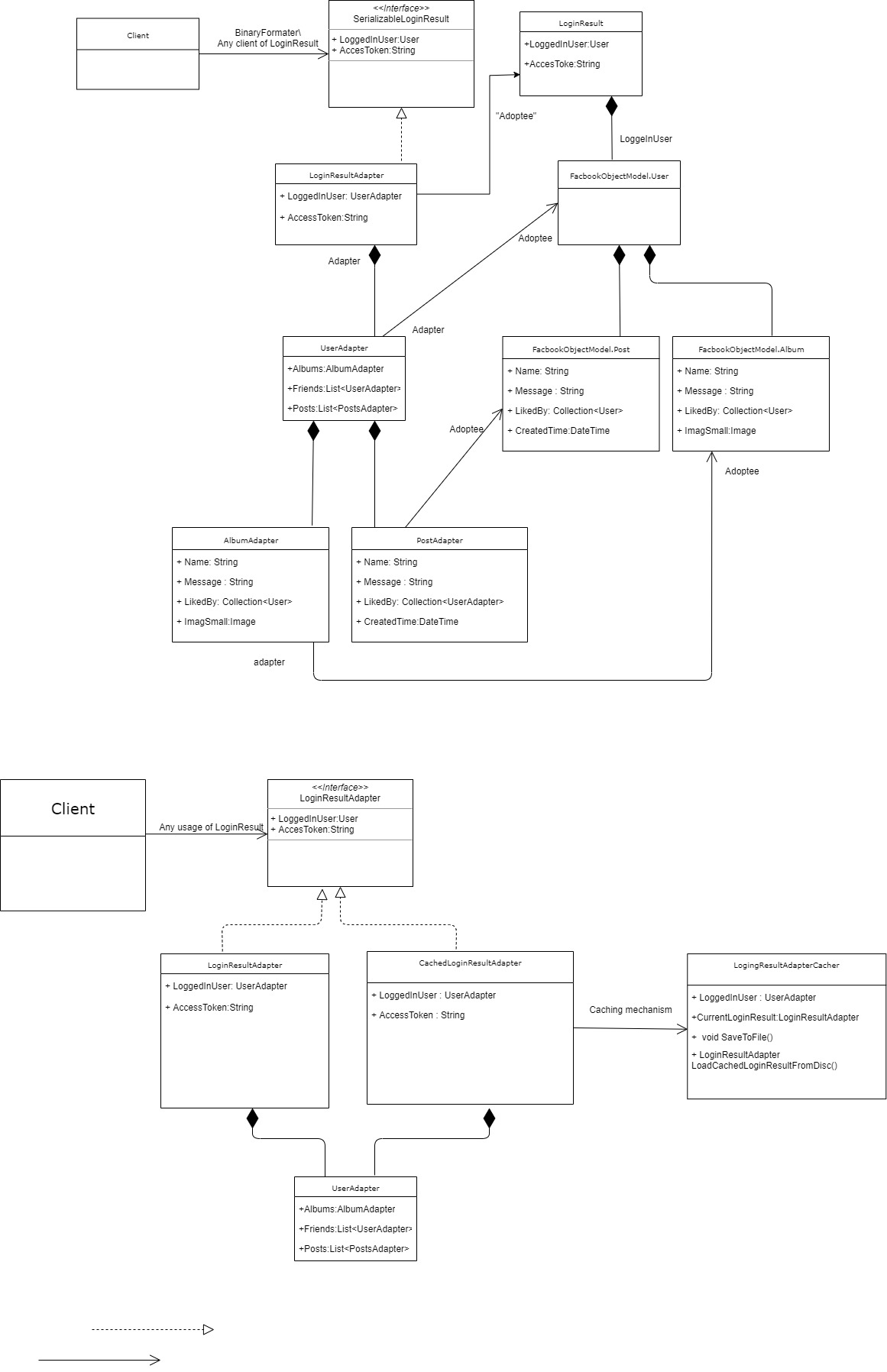
המחלקה: CachedLoginResultAdapter (הProxy עצמו) נותנת לכל המערכת שירות כמו loginResultAdapter רגיל ומפעילה כשנדרש את המחלקה LoginResultAdapterCacher.

המחלקה: LoginResultAdapterCacher הינה מחלקה אשר אחראית על ניהול וביצוע תהליך הכתיבה והשחזור של הנתונים (מנגנון הCACHING).

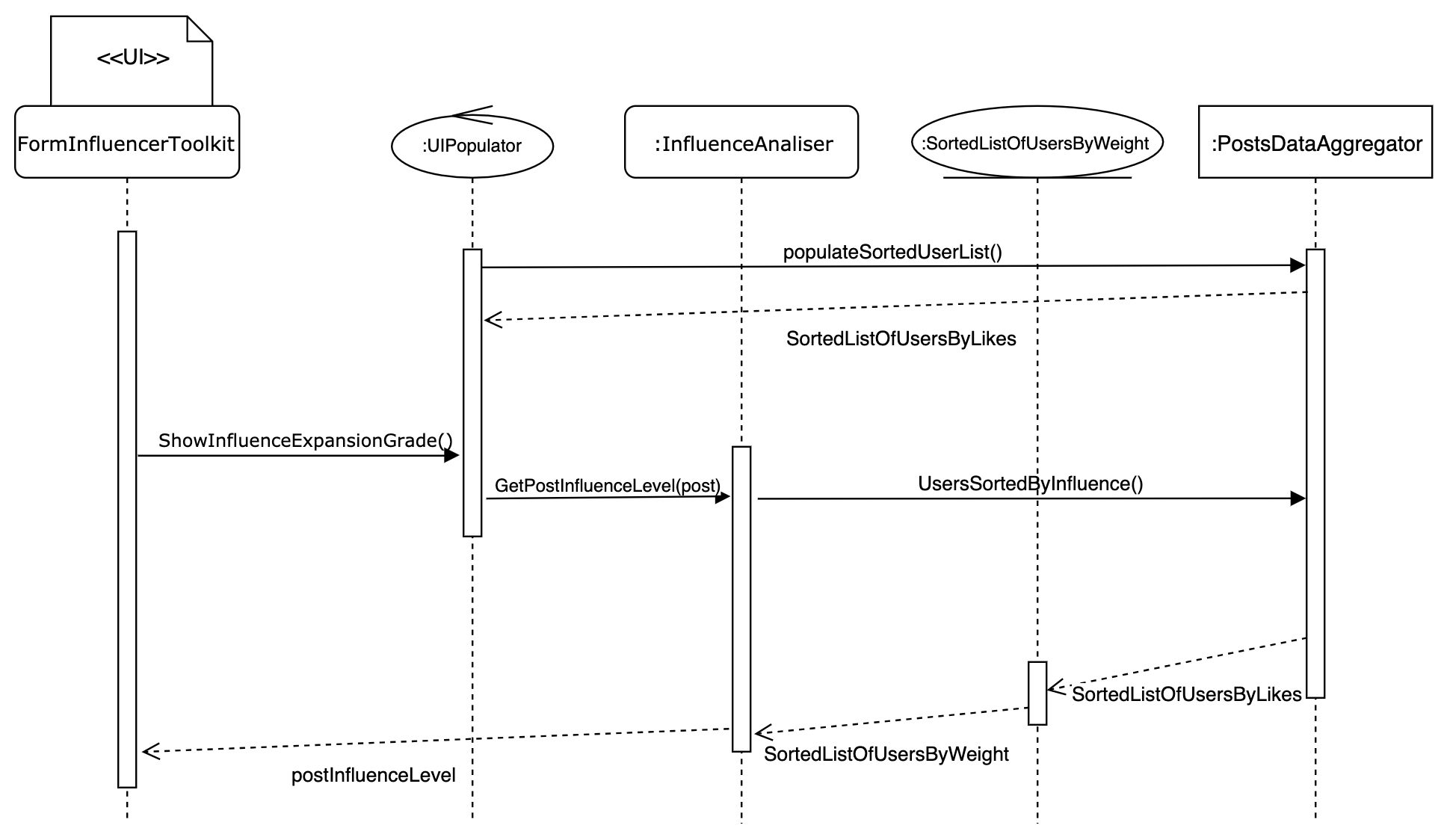
* Sequence Diagram

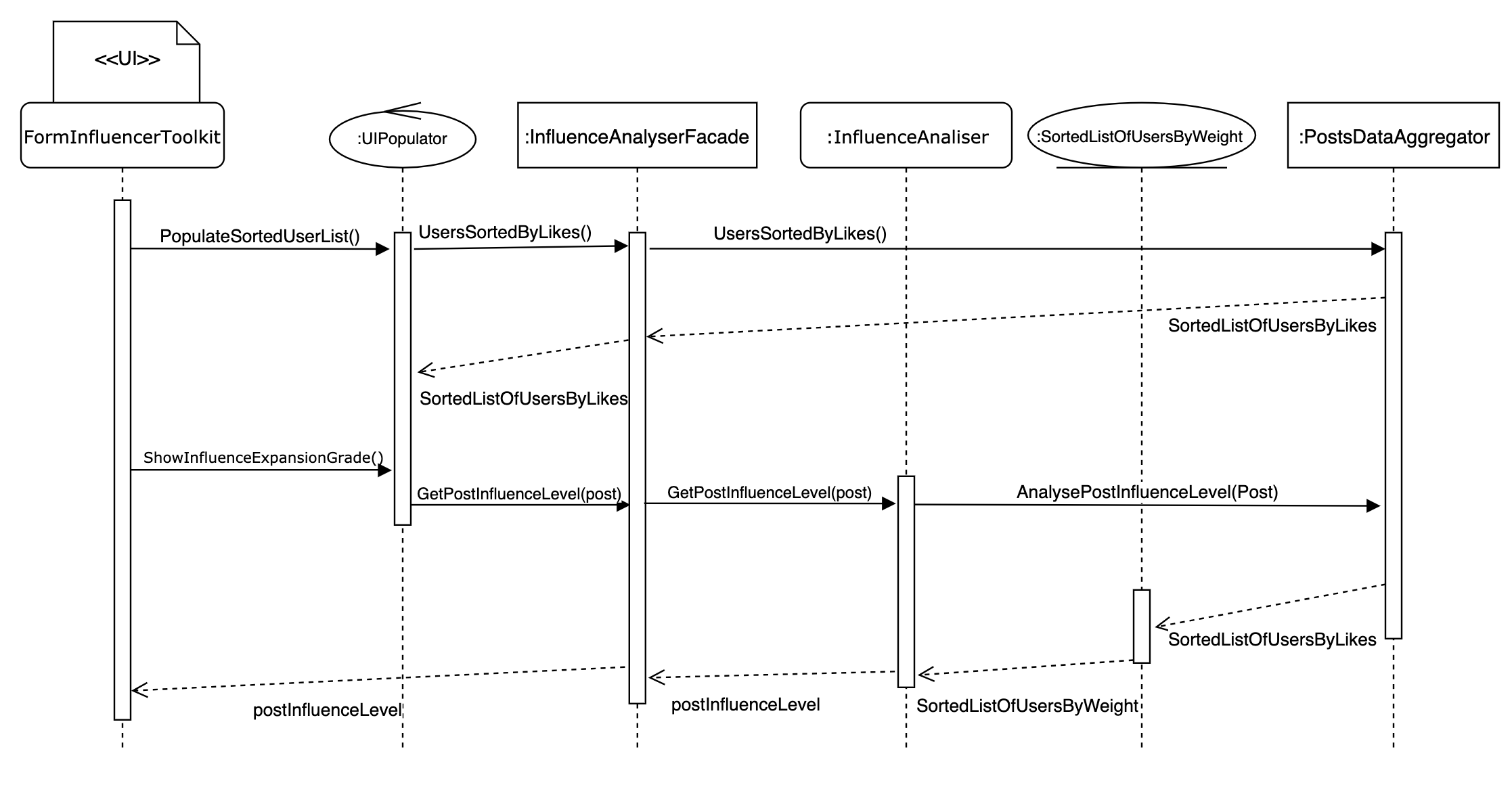
[sequence diagram שמציגה את התהליך שקשור לתבנית והאינטראקציה בין המחלקות]

* Class Diagram – הInterface מתאר את הציפיה של משתמשי האובייקט. הפרוקסי שלנו יודע לתת מענה לכל דרישה של האובייקט המקורי ומשתמש ברכיב שמבצע עבורו כתיבה לדיסק ושחזור מדיסק.

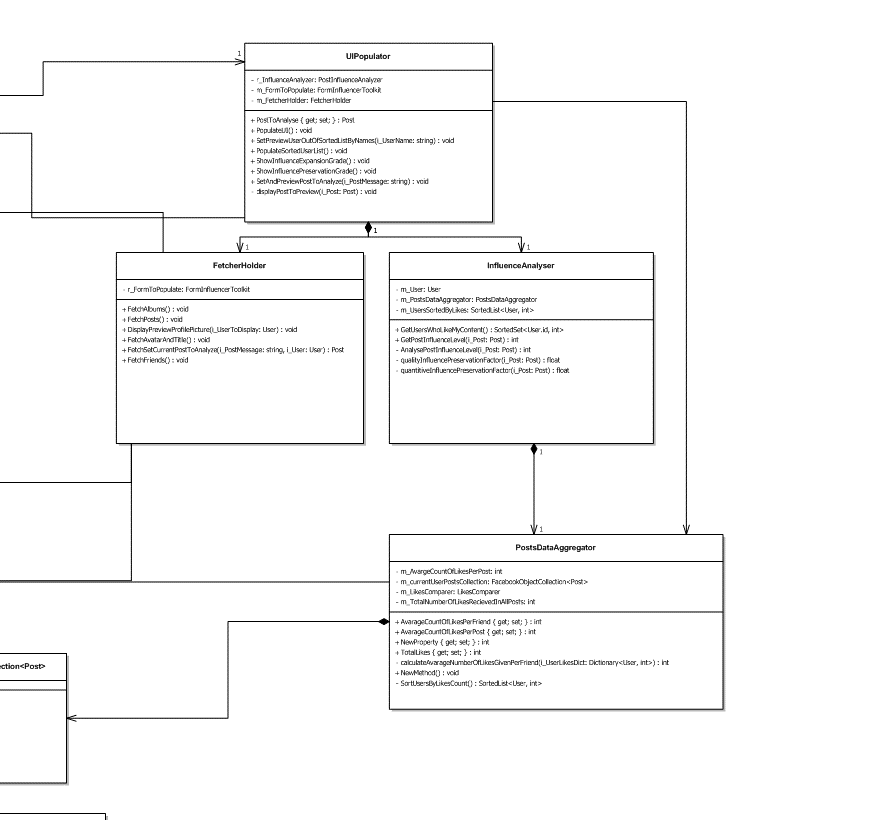


### **תבנית מס' 3 – Facade**

* סיבת הבחירה / שימוש בתבנית: מאפשר לבודד את החלק הלוגי המורכב של חישוב "רמת ההשפעה" של פוסטים. הרכיב העונה לשאלות מתממשק עם רכיבים שרוצים לקבל את "רמת ההשפעה" של post מסויים. במצב שלפני ה-refactor הנ"ל יש לנו שני רכיבים לוגיים   
  שה-UIpopulator משתמש בהם ויש תלות ביניהם: PostsDataAggregator ו-InfluenceAnalyser
  + השני משתמש בראשון (COMPOSITION)
  + גם ה-UIPOPULATOR – משתמש בראשון וזה מייצר משולש בהיררכיות שונות עם תלויות ש"לא קל ללמוד ולנתח" – ייתכן והמצב שנוצר הוא לא מיינטיניבילי.
  + ע"י שימוש ב-FACADE כל התלויות ירוכזו במקום אחד.
* אופן המימוש: את ה-Facade הכנסנו כרמה נוספת בין ה-UIPOPULATOR לבין שתי המחלקות איתן הוא עובד, PostsDataAggregator ו-InfluenceAnalyser. כ-Facade שקוף.  
  למעשה, ה-UIPOPULATOR אינו מכיר כעת את המחלקות הנוספות, ועובד עימן דרך ה-Facade.  
  בתוך ה-Facade ישנן כל המתודות אשר היו בשימוש על ידי ה-UIPOPULATOR.   
  בעזרת שימוש בקומפוזיציה, ה-Facade מנגיש למשתמש בו את הפונקציונליות והלוגיקה אשר בשכבה הלוגית באמצעות שיטוח למתודות פשוטות, ללא צורך להשתמש בשכבות הלוגיות השונות וללא תלות כלשהי בעומקן, או במשתנים מסוג מסויים, אלא רק ב-Facade עצמו המרכז את היכולות הקיימות ומפעיל את המתודות הרלוונטיות בהתאמה.  
  את ה-Facade מימשנו בצורה פשוטה, כך שאינו מכיל הרבה קוד ואינו משנה או מוסיף יכולות כלשהן למערכת, אלא רק משטח את המערכת על מנת להקל על ה-Client.  
  דוגמת קוד מה-Facade, ניתן לראות שימוש בקומפוזיציה ב-Constructor:  
  
* Sequence Diagrams: באופן פעולת המערכת לפני מימוש ה-Facade, ניתן לראות כי קיים קשר ישיר ואף ישנה אינטראקציה בין שכבות ה-UI לבין שכבות הלוגיקה השונות:  
  

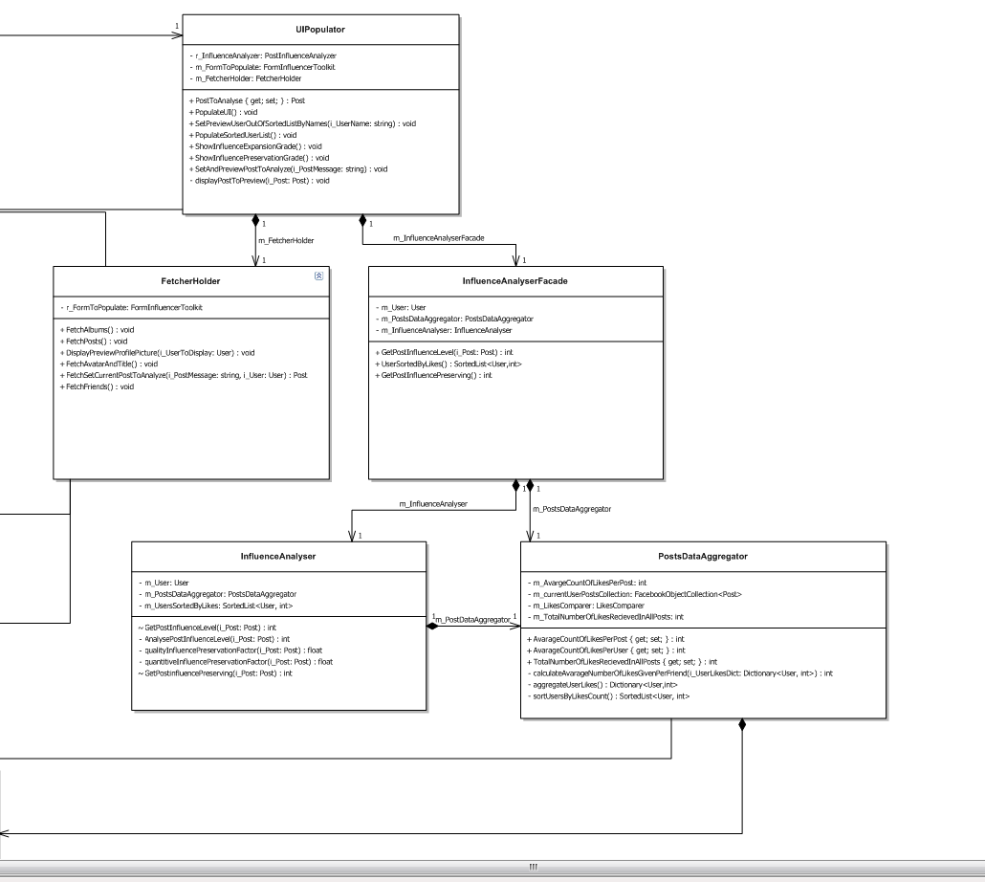
לאחר מימוש ה-Facade, ניתן לראות כי ישנה הפרדה ברורה בין שכבות הלוגיקה השונות לבין שכבת ה-UI. שכבת ה-UI פונה לקבלת מידע מהשכבה הלוגית דרך נקודה אחת בלבד, ובכך אינה מכירה כלל את השכבה הלוגית או את עומקה ולכן משתמשת   
ב-Facade כממשק המקל על השימוש בשכבה הלוגית המורכבת:

* Class Diagram
* לפני השינוי:



**לאחר השינוי:**

השטחה של הגישה של UIpopulator לשני הרכיבים: PostDataAggregator ו InfluenceAnalyser ההפעלה והגישה לשני הרכיבים עכשיו ממחלקה שנקראת InfluenceAnalyserFacade

**Façade:** InfluenceAnalyserFacade

**Class 1:** PostDataAggregator

**Class 2:** InfluenceAnalyzer

### **תבנית מס' 4 – Singleton**

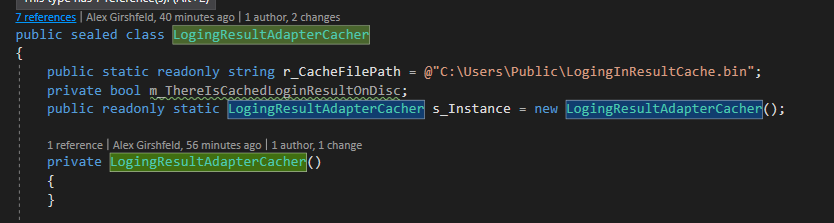
* **סיבת הבחירה / שימוש בתבנית:**

במהלך מימוש של מנגנון הCACHING של הProxy זוהה צורך לנהל את התהליך כתיבה וקריאה מהזכרון בצורה ריכוזית ע"י רכיב אחד שייתן שירות למערכת.

ניהול הנתיב, התזמון, ההרשאות ומצב הזכרון צריכים מן הסתם להיות מבצועים בצורה מסונכרת ע"י רכיב שכל הזמן "יודע" מה יש לו בזכרון וכדי להבטיח את הנ"ל הוחלט לממש את הרכיב הזה כSingleton

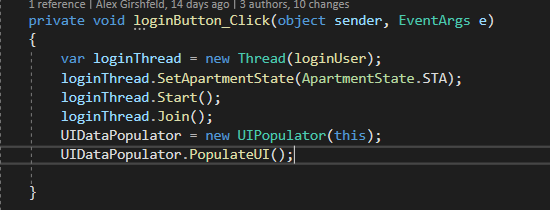
* **אופן המימוש:**

המחלקה: LoginResultAdapterCacher הינה מחלקה אשר אחראית על ניהול וביצוע תהליך הכתיבה והשחזור של הנתונים (מנגנון הCACHING).



### **עבודה אסינכרונית –**

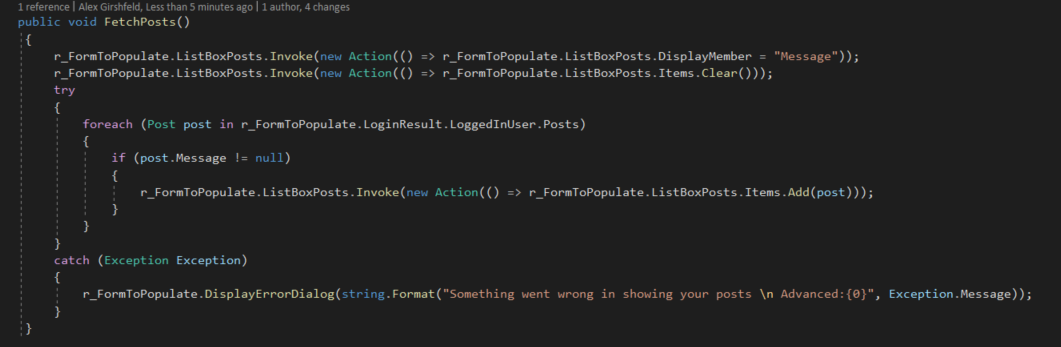
* אנחנו משתמשים THREADים חדשים במהלך:
  + הרצת מתודת הloginUser אשר מבצעת תהליכי תקשורת רשתיים
* הסיבה לביצוע שתי הפעולות הנ"ל מTHREADים נוספים היא חווית משתמש יותר "חיה" ותגובתית. כלומר משתמש שביצע פעולה שלוקחת זמן (במקרה הזה עקב ממשק רשת וטעינת נתונים) לא יחווה חלון "קפוא" עד סיום הפעולות.
* הפעלת הTHREADים:



* טעינה והפעלת הפקדים מTHREAD נפרד (UIPopulator):

אין לנו אפשרות לבצע את פעולות טעינת המידע לפקדים דרך הUIPopulatorThread (כפי שראינו בסרטון ההדרכה – פעולות כאלו נדרש לבצע מTHREAD הUI ש"מריץ" את הפקדים עצמם)

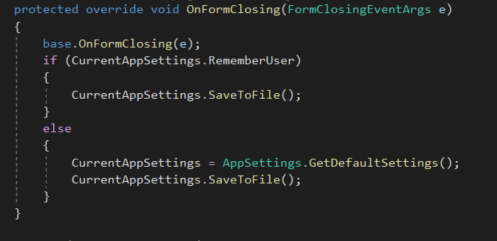
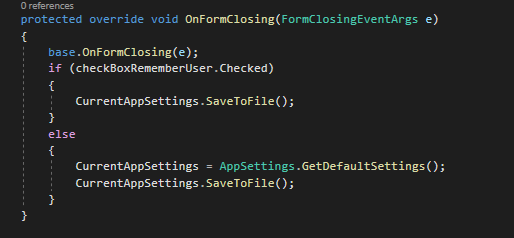
ולכן אנחנו נבצע פעולות אלו ע"י INVOKE מהפקדים עצמם ונבצע את הפעולות האלה בTHREAD הUI עם LAMBDA EXPRESSION)



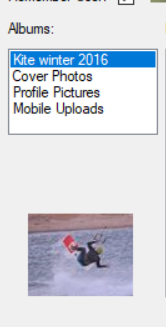
### **עבודה עם dataBinding**

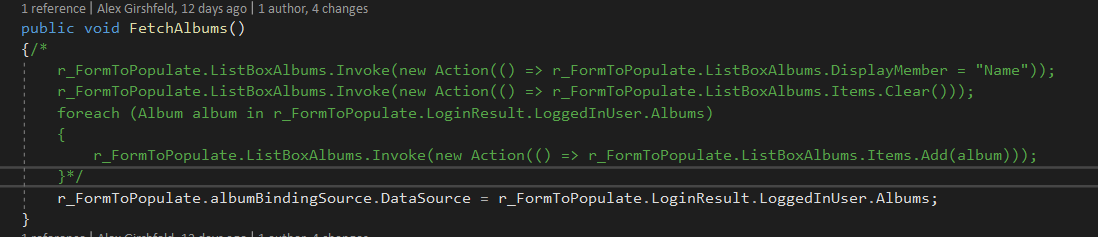
* שינינו את העבודה עם הפקד Remember user לעבודה עם two-way data binding ול CurrentAppSettings. ועכשיו אפשר שהCurrentAppSettings הופך להיות הרפרנס הבלעדי ללוגיקה בעת כיבוי החלון (הרבה יותר ברור, מיינטנבילי ונוח לשינוי)

לפני השינוי התייחסות לפקד: אחרי (התייחסות להגדרות הנוכחיות):



* שינינו את העבודה האלבומים של היוזר לtwo-way data binding. כמו שמצויין בסרטון ההדגמה הבאת ה"תוכן" (אלבומים) הוחלפה בחיבור של האלבומים שנמצאים לאובייקט albumBindingSource שבתורו דואג לאכלס pictureBox שמציג את תמונת האלבום לפי בחירת המשתמש.





TODO:

* General:
  + ~~Databinding – Alex possible binding via fetch methods in UIpopulator~~
  + ~~AsyncWorking with multithreading~~
  + Updating Class diagram - Alex
  + First Ex features description
* Façade:
  + ~~Code - TAL~~
    - ~~Extracting the two public methods from influence analyzer~~
    - ~~Moving the usage of post aggregator by UIpopulator (now uses postAgregator by himself)~~
    - ~~The new façade class holds the post aggregator, Influence analyzer and the interactions between them~~
  + ~~Sequence diagram - Tal~~
* Caching Proxy (UserCachedProxy)
  + Code
  + Features:
    - Display of the last updated time
    - Maximum update interval
    - Each time the data being accessed – check if update needed (max update interval)
    - Uses the FacebookObjectCacher to save the data
  + Sequence Diagram - Tal
* FacebookObjectCacher
  + Code
  + Sequence Diagram - Tal