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

**פיצ'ר 1 – מציאת חברים שביקרו במקום מסוים:**

הפיצ'ר מציג רשימת ממויינת Chekins לפי שם המקום של כל החברים של המשתמש.   
המשתמש יכול לבחור Checkin מסויים באמצעות שתי דרכים שונות:

1) חיפוש ידני ע"פ שם של מקום מסויים.

2) בחירה מרשימת הCheckins הנתונה.

לאחר מכן תופיע רשימת החברים אשר שהו במקום זה.

כמו כן, המשתמש יוכל לבחור חבר מסויים ולקבל פרטים נוספים לגבי אותו Checkin כגון:  
תמונה של החבר מהמקום המתויג, שם המקום ותאריך התיוג.

\*ניתן למצוא פיצ'ר זה במחלקת ChekIn.

**פיצ'ר 2 – שלוש התמונות אשר קיבלו את מספר הלייקים הגבוה ביותר:**  
הפיצ'ר מאפשר למשתמש לצפות בשלוש או ב שש (לפי בחירה) התמונות שלו אשר קיבלו את כמות הלייקים הגבוהה ביותר, מבין כל תמונות המשתמש.   
בנוסף המשתמש יכול לבחור תמונה מבין תמונות אלה ולעדכן אותה כתמונות פרופיל בתוך האפליקציה.

כמו כן, קיימת אופציה להחליף כל אחת מהתמונות לתמונה בעלת מס' לייקים נמוך יותר, כך ניתן לראות שלשה של תמונות האהובות אחרי השלשה הראשונה (שלושת התמונות עם הכי הרבה לייקים) לפי הסדר ,בנוסף ניתן להסתיר את התמונות באמצעות לחיצה על כפתור "Hide Pictures" ולהחזירן באמצעות הכפתור "Restore", לאחר לחיצה על כפתור זה יופיעו שוב התמונות האחרונות שנבחרו לאחר השינויים.

\*ניתן למצוא פיצ'ר זה במחלקות LikedPhotos , , LikedPhotosActionesFirst LikedPhotosActionesChanged , LikedPhotosActionesBase

### תבנית מס' 1 – Iterator

**סיבת הבחירה / שימוש בתבנית:**

מטרת התבנית לשמש כלי לסריקה של אוספים שונים בצורה דומה וכללית ללא תלות באופן מימוש האוספים כלומר, במבני הנתונים בהם הם מוחזקים.

כמו כן, במידה ובעתיד המימוש של אותו אוסף ישתנה מסיבה כלשהי יהיה עלינו לבצע שינויים אך רק בIterator ולא בכל המקומות בקוד בהם אנו מבצעים סריקה של האוסף.

**אופן המימוש:**

תפקידים:ה- Client הוא: FacebookAppForm

ה- Aggregaate הוא: IAggreate  
ה- ConcreteAggregate הוא: FriendsAggregate

ה- Iterator הוא: IIterator.

ה- ConcreteIterator הוא: FriendsIterator.

מימוש:  
תחילה בנינו FriendsAggregate העוטפת את User.Friends לאחר מכן יצרנו מחלקה המספקת Iterator בשם IIterator כמו כן יצרנו איטרטור קונקרטי FriendsIterator עבור אוסף החברים של המשתמש.

האיטרטור מאפשר לנו סריקה של אוסף החברים באופן כללי ללא תלות במימוש שלו.

בקוד הדגמנו את פעולת האיטרטור בפיצ'ר השני במחלקה CheckIn בעזרתו אנו סורקים את אוסף החברים של המשתמש כדי לייבא את נתוני הCheckin של כל החברים.

אך באותה מידה יכלנו להשתמש בו בכל קטע קוד המצריך סריקה של אוסף החברים.

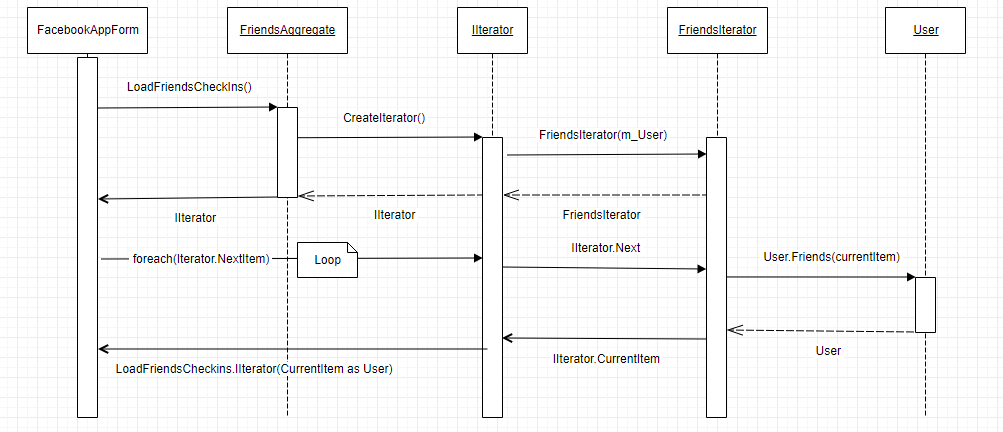
המימוש נמצא בקבצים:

A18\_Ex01\_Rahel\_206082703\_Avihay\_203766035\DesignPatterns\IIterator.cs

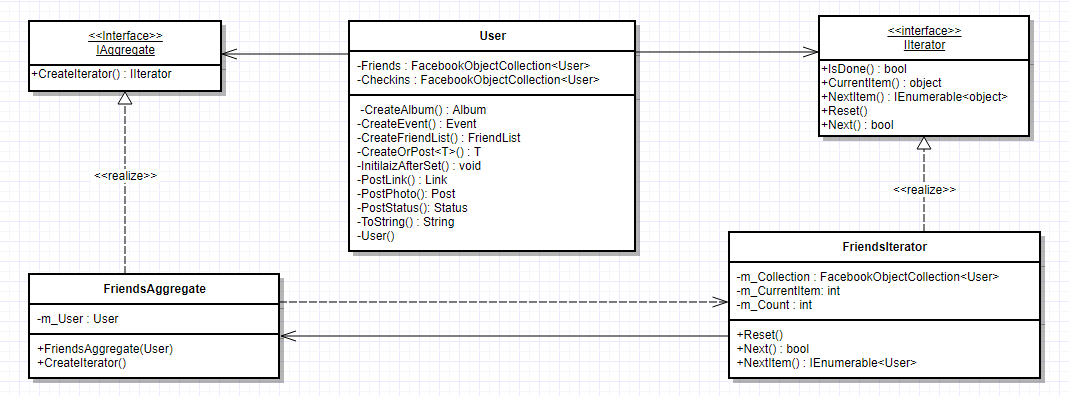
A18\_Ex01\_Rahel\_206082703\_Avihay\_203766035\DesignPatterns\FriendsAggregate.cs

A18\_Ex01\_Rahel\_206082703\_Avihay\_203766035 \Logic\CheckIn.cs

**Iterator - Sequence Diagram:**



**Iterator - Class Diagram:**



### תבנית מס' 2 – Strategy

**סיבת הבחירה / שימוש בתבנית:**

באפליקצייה שלנו יצרנו רשימות של אובייקטים שונים (רשימת חברים, אירועים, צ'אק אינים וכו'). אלגוריתם ליצירת רשימה של אובייקטים כזו מאוד דומה, השוני הוא בסוג האובייקט של הרשימה שמיוצר. לכן יישמנו את Strategy Pattern המאפשר לנו בחירה של אסטרטגיה (סוג אובייקט) בזמן ריצה. מאפשר גישה נוחה לאלגוריתם בכך שמסיר אותו מהמחלקה "המארחת" שלו ושם אותו במחלקה נפרדת, כך אפשר לבחור את האסטרטגיה המתאימה מכל מקום בקוד.

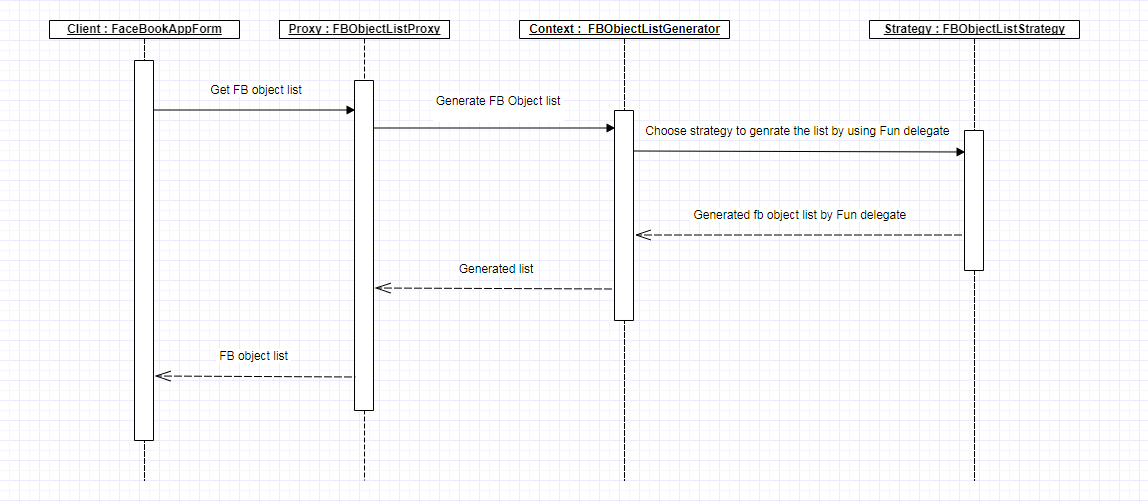
**אופן המימוש:**\* נציין כי לא השתמשנו בצורה הקלאסית של הפאטרן, אלא ביכולות של c# 3 .   
  
תפקידים:  
  
ה- Context :FBObjectListGenerator<T>.  
ה- Strategy : FBObjectListStrategy  
ה- Concrete Srtategy – שימוש ב- Delegate : Func<IEnumerable<T>>

מימוש:

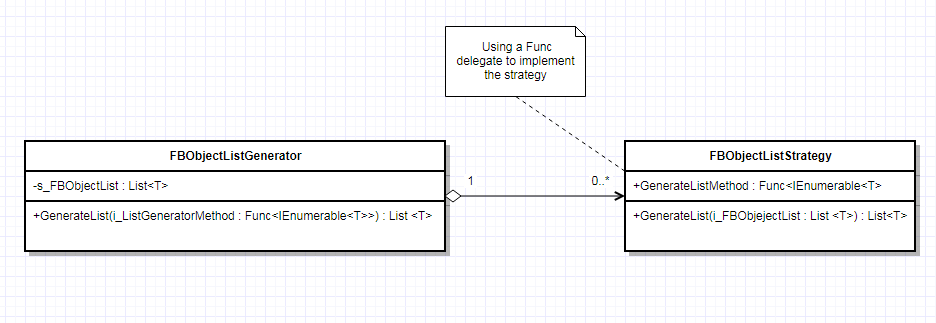
המחלקה FBObjectListGenerator יוצרת את רשימות האובייקטיים תוך שימוש ב FBObjectListStrategy . היא מעבירה לה את סוג האובייקט אותו צריך לייצר, ובאמצעות delegate מקבלת מצביע לפונקציה שמחזירה IEnumerable<T> (קולקצייה מסויימת). המחלקה הזאת יוצרת את הרשימות לפי סוג האסטרטגיה (אובייקט) ש- FBObjectListProxy שולחת לה בזמן ריצה.   
השימוש במצביע לפונקציה בעצם חוסך לנו בניית מחלקה לכל פתרון ונותן לנו אפשרות לפתור את הבעיה במחלקה אחת.  
  
המימוש נמצא בקבצים:

A18\_Ex01\_Rahel\_206082703\_Avihay\_203766035\DesignPatterns\FBObjectListGenerator.cs

A18\_Ex01\_Rahel\_206082703\_Avihay\_203766035\DesignPatterns\FBObjectListStrategy.c

**Strategy - Sequence Diagram:**

**Strategy - Class Diagram:**



### תבנית מס' 3 – Template Method

**סיבת הבחירה / שימוש בתבנית:**

בפיצ'ר השני שמימשנו קיימות 2 מחלקות שונות המממשות ממשק דומה ובעלות פונקציונאליות דומה למעט כמה שינויים, תבנית זו מאפשרת לנו במצב כזה ליצור מחלקה אבסטרקטית שבה תמומש הפונקציונאליות הזהה ו2 מחלקות שונות שירשו ממנה ובהן מימושים של החלקים הקונקרטים השונים.

כלומר, כל הפונקציונאליות הזהה תיעשה ממחלקת הבסיס האבסטרקטית ושאר החלקים השונים ימומשו במחלקות הקונקרטיות היורשים ממנה.

**אופן המימוש:**

כמו שאמרנו קיימות לנו 2 מחלקות שזהות ברוב הפונקציונאליות LikedPhotosActionesFirst ו LikedPhotosActionesChanged לכן יצרנו מחלקת בסיס אבסטרקטית LikedPhotosActionesBase המממשת את הפונציונאליות הזהה כך שהפונקציונאליות השונה נעשית בכל אחת מהמחלקות הקונקרטיות באמצעות ירושה.  
  
תפקידים:

ה- AbstractClass- LikedPhotosActionesBase  
ה- ConcretassA/B- LikedPhotosActionesFirst, LikedPhotosActionesChanged.

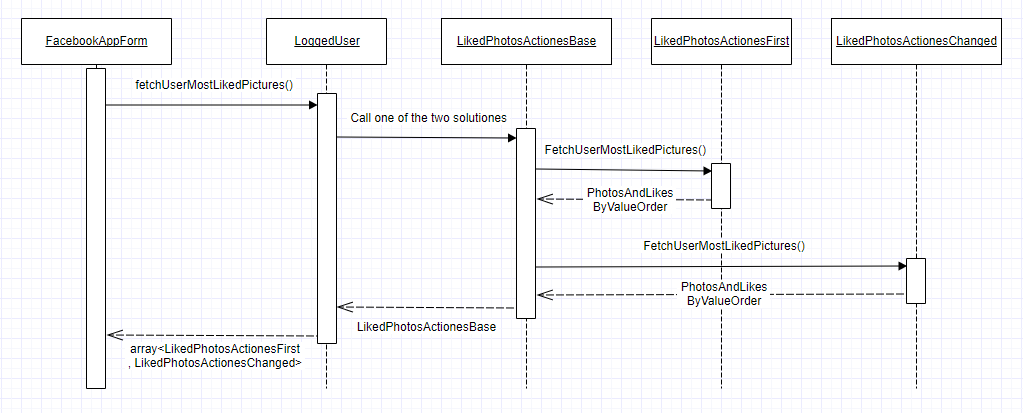
מימוש:  
  
המימוש נמצא בקבצים:

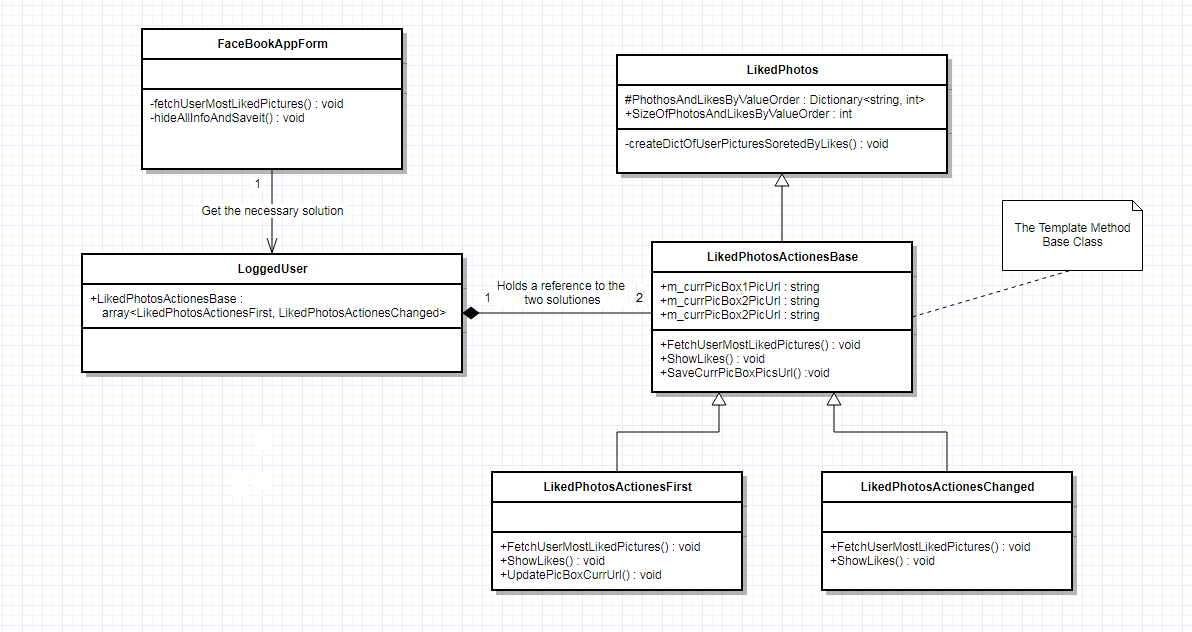
A18\_Ex01\_Rahel\_206082703\_Avihay\_203766035\Logic\ LikedPhotosActionesBase.cs

A18\_Ex01\_Rahel\_206082703\_Avihay\_203766035\Logic\ LikedPhotosActionesFirst.cs

A18\_Ex01\_Rahel\_206082703\_Avihay\_203766035\Logic\LikedPhotosActionesChanged.cs

**Template Method - Sequence Diagram:**



**Template Method - Class Diagram:**