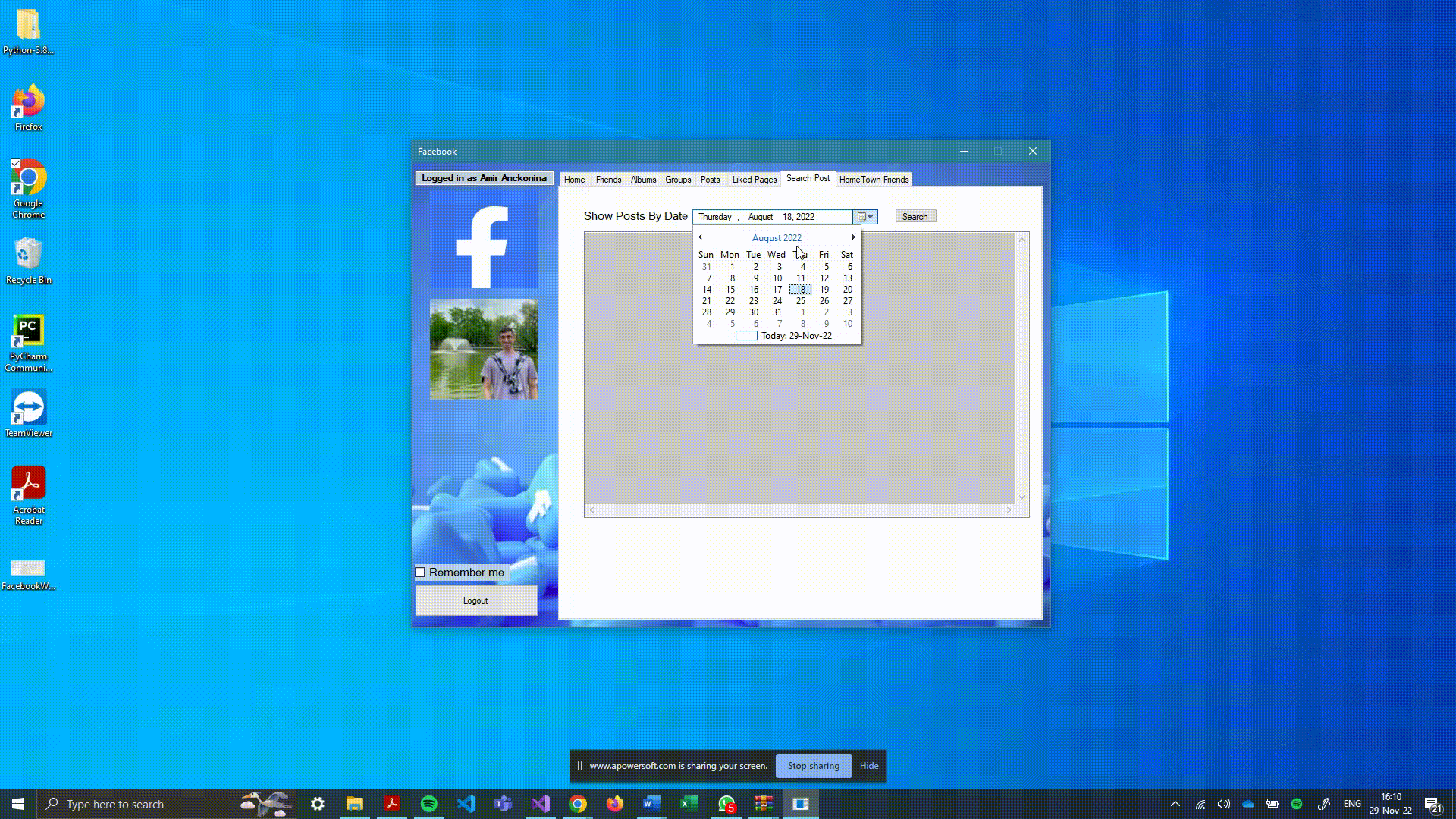
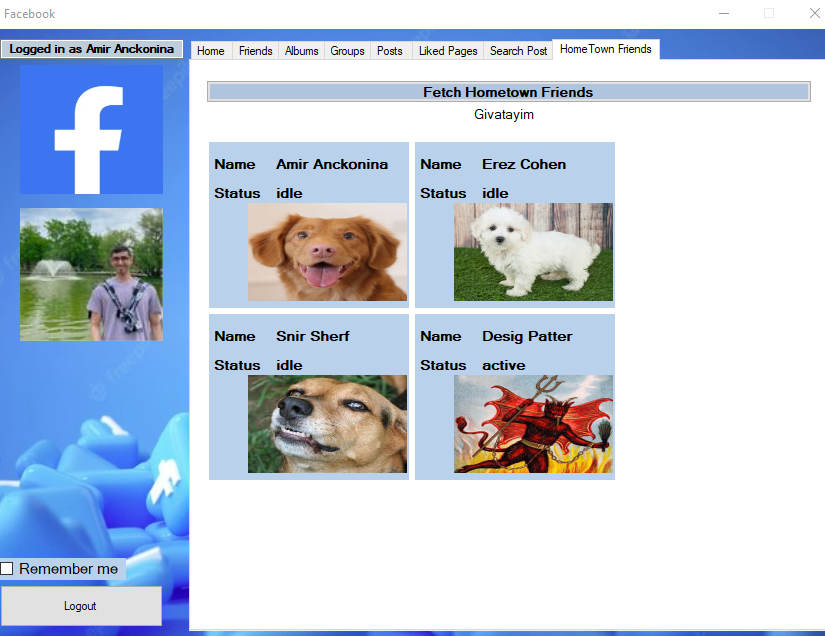
### תיאור קצר של הפיצ'רים שבחרנו לממש בתרגיל הקודם:

* **Search Post By Date**
  + הפיצ'ר הנ"ל נותן למשתמש את היכולת "לפלטר" פוסטים שפירסם בפייסבוק על בסיס תאריך.
  + המשתמש לוחץ על בחירת תאריך ונפתח עבורו תפריט נוח ואינטואיטיבי, בו ניתן לראות לוח שנה ולנוע בקלות בין שנים, חודשים וימים.
  + לאחר הבחירה, המשתמש לוחץ על Search וכל הפוסטים מהתאריך המצוין יופיעו.

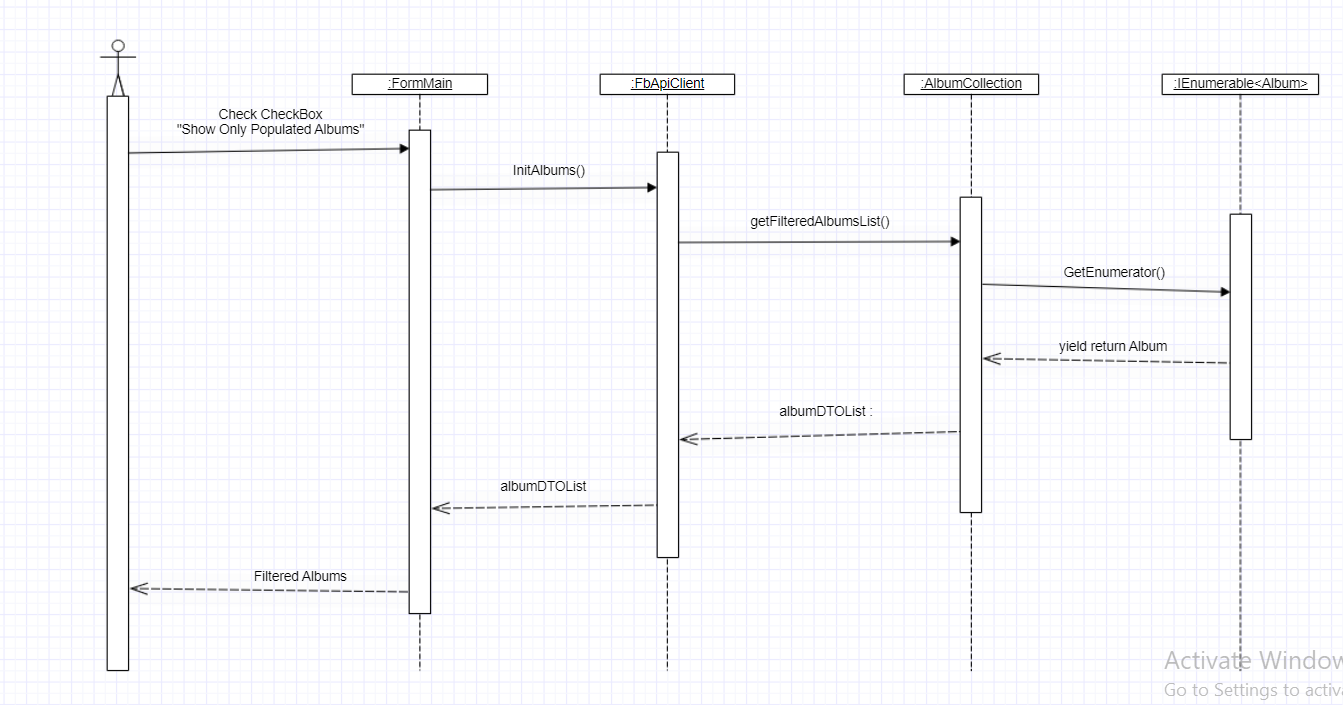
****

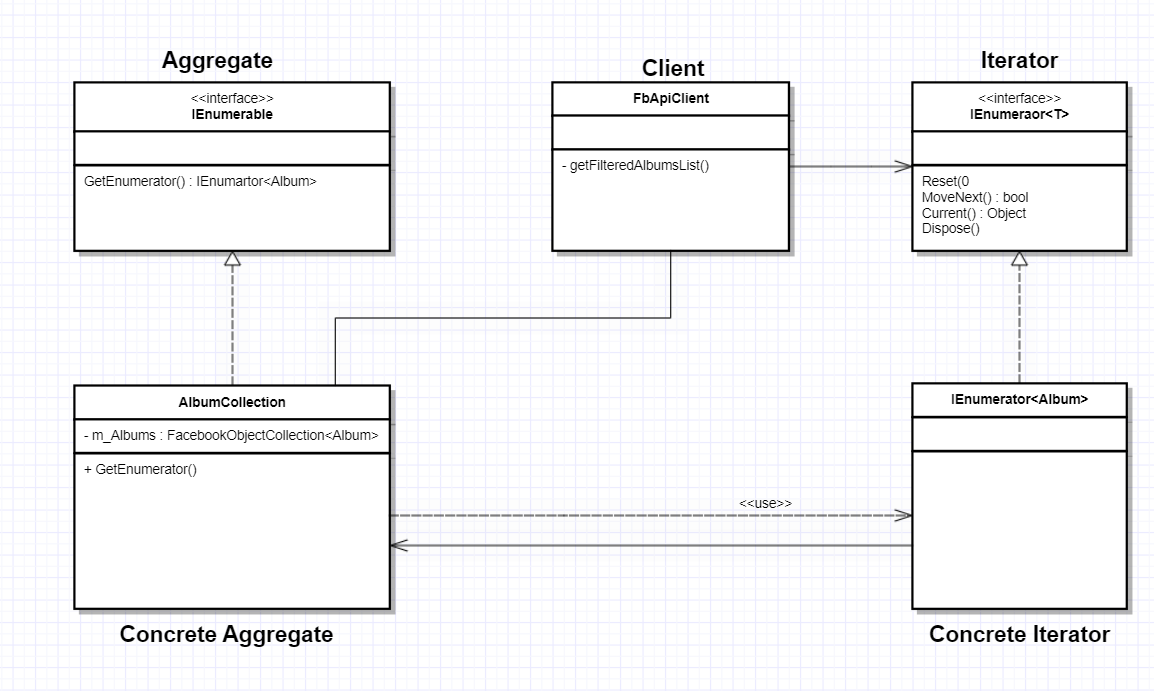
* **Hometown Friends**
  + הפיצ'ר הנ"ל מאפשר למשתמש לקבל תצוגה נחמדה של חבריו בפייסבוקאשר מאותו עיר.
  + היות והגישה דרך הAPI חסומה, מימשנו Dummy Data בו רשימת חברים ולכל חבר יש רשומה של העיר בה הוא גר.
  + כמו כן, אנו מציגים למשתמש את סטטוס החבר כרגע באפליקציה.

****

### תבנית מס' 1 – Iterator

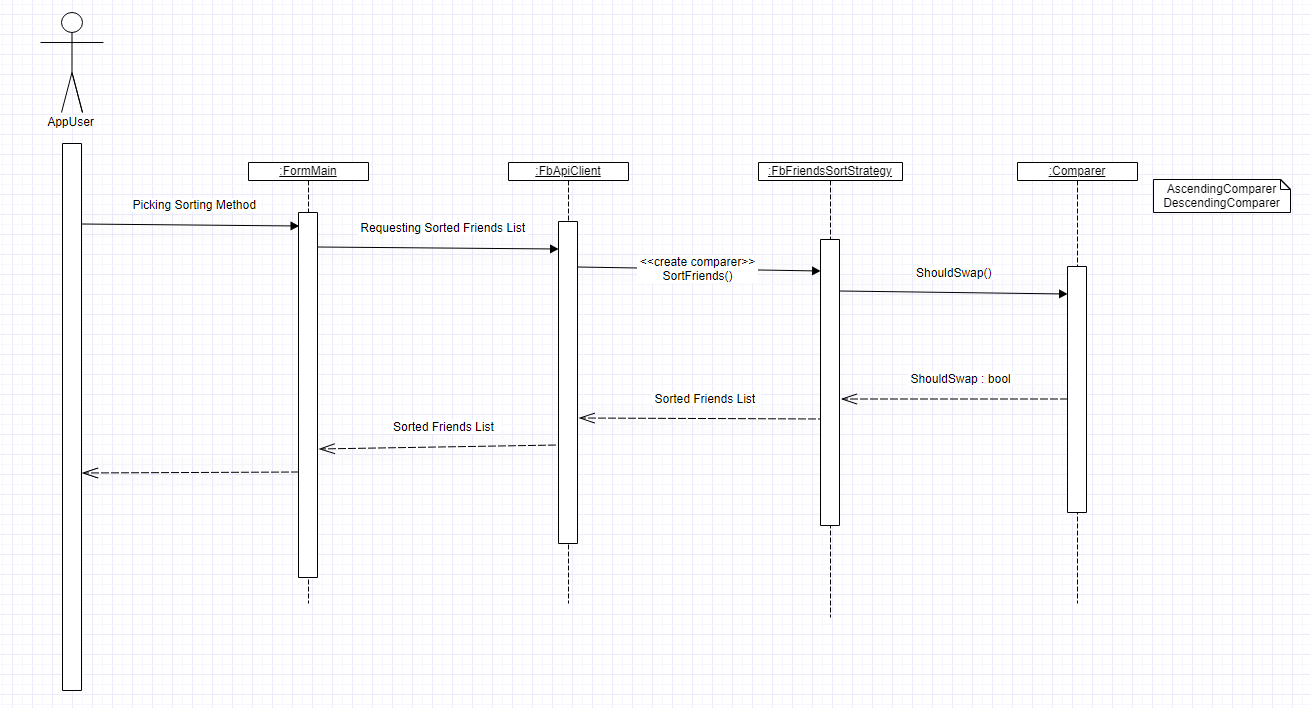
* **סיבת הבחירה / שימוש בתבנית:**
  + באפליקציה שלנו אנחנו מקבלים את נתוני האלבומים של המשתמש משרת פייסבוק.
  + לרוב, ישנם למשתמש אלבומים ריקים או אלבומים ישנים עם מעט תמונות.
  + הוספנו CheckBox ב Albums Tab בו אנו "מפלטרים" את האלבומים ומציגים למשתמש רק את אלה שהם "Populated", כלומר מלאים בתוכן, ורלוונטים באמת עבורו.
  + הגדרנו כמות של 7 תמונות באלבום כ Filter, במטרה להציג רק אותם.
  + ובהתאם, מימשנו Filter Iterator, שמאפשר לנו לרוץ רק על האלבומים העומדים בתנאי.
  + האיטרטור מאפשר לנו את הסתרת מבנה הנתונים וחשיפת ממשק אחיד המאפשר מעבר על הקולקציה.

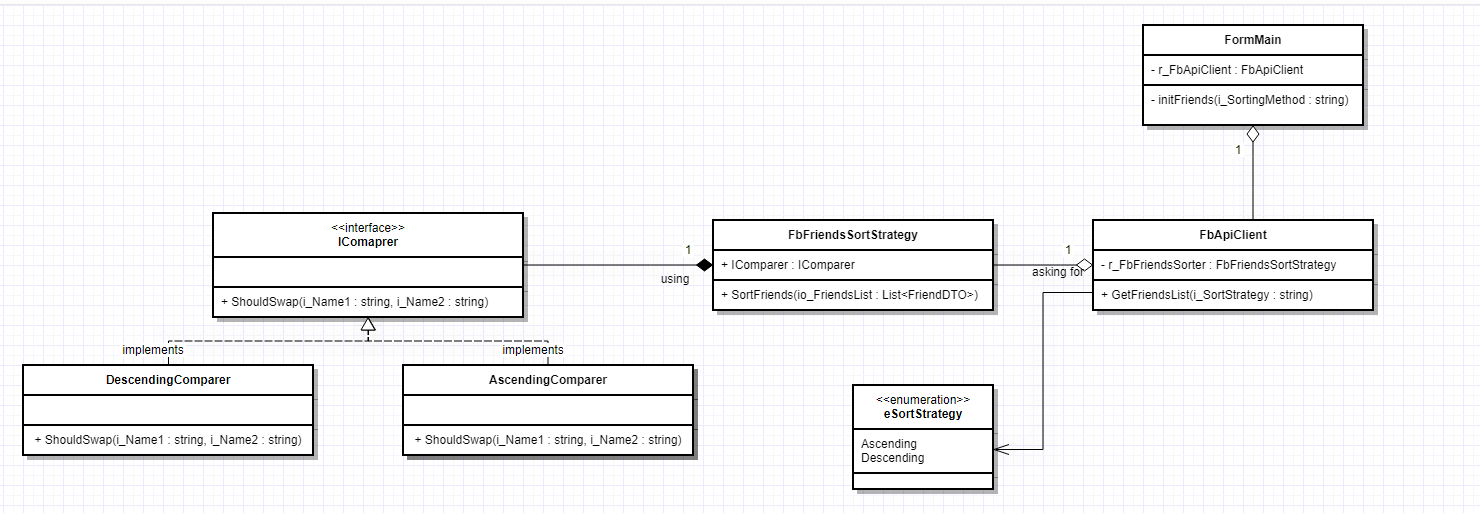
* **אופן המימוש:**
  + במחלקה FbApiClient, במתודה getFilteredAlbumsList(), אנחנו יוצרים aggregate שהוא המחלקה AlbumCollection.
  + זוהי מחלקה אשר מממשת את הממשק IEnumrable<Album>, ובפרט את המתודה GetEnumerator().
  + כאשר, ה Filter Iterator שלנו ממומש בצורה של תנאי – החזר רק את האלבומים בהם יש יותר מ 7 תמונות.   
    המתודה מחזירה IEnumerator<Album>, בעזרת yield return.
* **Sequence Diagram**
* **Class Diagram**



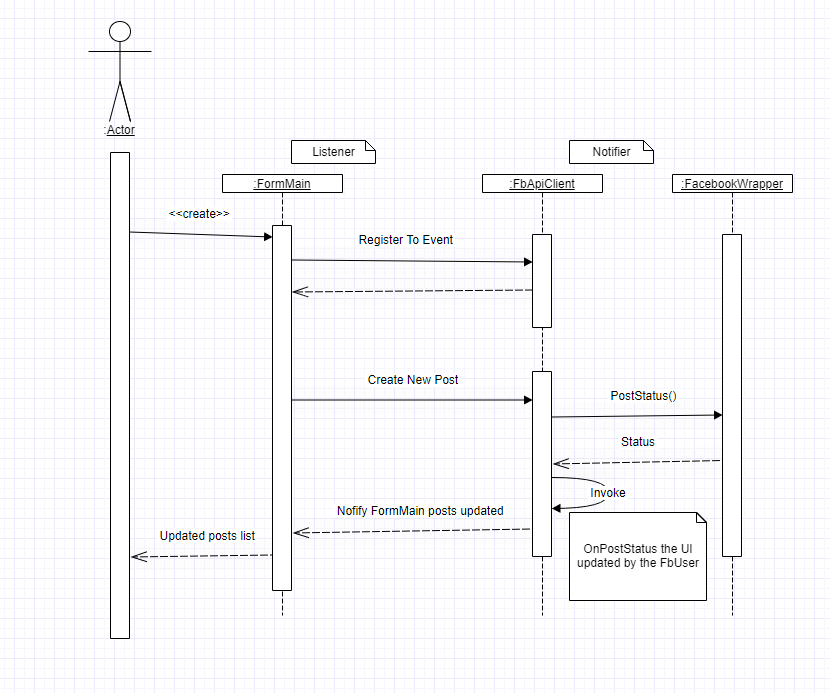
### תבנית מס' 2 – Strategy

* **סיבת הבחירה / שימוש בתבנית:**
  + למעשה, מימשנו פיצ'ר המאפשר למשתמש לקבל את רשימת חבריו באופן ממוין על פי שם. המשתמש יכול לבחור בסדר Ascending, דהיינו לפי סדר אלפבתי, או בסדר Descending, אלפבתי הפוך.
  + עבור כל בחירה נרצה לממש מיון, של רשימת החברים, אך בזמן ריצה נרצה להחליט על אופן המיון, כלומר האם יהיה בסדר אלפבתי רגיל או הפוך, או לחילופים בעתיד למיין על פי פרמטרים אחרים.
  + פעולת מיון רשימת החברים למעשה זהה, ריצה על List והחלפת איברים, אך תנאי ההשוואה, ה Comparer, הינו שונה.
  + תבנית ה Strategy נבחרה בכדי לממש מיון זהה עם תנאי השוואה שונה. מונעת שכפול קוד, ונותנת אפשרות פשוטה להרחיב את המערכת עם מימושים דומים של מיון תחת אותה תשתית.
* **אופן המימוש:**
  + יצרנו מחלקה חדשה בשם FbFriendsSortStrategy שלמעשה מחזיקה אצלה בבטן IComparer, שזהו Interface עם מתודה בודדת ShouldSwap.
  + לinterface הנ"ל שני מימושים, AscendingComparer ו DescendingComparer ששניהם ממשים את המתודה ShouldSwap כל אחד.
  + בתוך המחלקה FbFriendsSortStrategy קיימת המתודה SortFriends שעושה שימוש ב Comparer הנוכחי שמוגדר במחלקה
  + כאשר המשתמש מעוניין לקבל רשימה ממוינת, ה Client תחילה מייצר Comparer על פי הסוג הנדרש ואז קורא ל SortFriends.
* **Sequence Diagram**



* **Class Diagram**

### תבנית מס' 3 – Observer

* **סיבת הבחירה / שימוש בתבנית:**
  + בחרנו בתבנית Observer עבור עדכוני פוסטים המבוצעים בחלק הלוגי של המערכת.
  + למעשה, אפליקציית הפייסבוק של המשתמש יכולה להיות פתוחה במשך זמן רב, ובשלב מסוים מבוצע עדכון של פוסט, ייתכן אפילו ממכשיר אחר.
  + במצב כזה, נרצה שהקליינט, החלק הלוגי יעדכן את הUI שבוצע שינוי.
  + שיטת עבודה כזו, עוזרת להימנע מבדיקות תדירות של המערכת על השינויים (דגימות מצד הUI אל מול האפליקציה) ומאפשרת במקום ל UI לקבל התראה שיש תוכן חדש שעליו להציג ובכך הוא מעדכן את התצוגה ואת הפוסט החדש שהתווסף.
* **אופן המימוש:**
  + למעשה ה Notifier שלנו הוא ה FbApiClient שהוא הרכיב הלוגי**.**
  + וה Listener שלנו הוא ה FormMain, ה UI**.**
  + הגדרנו event Action<PostDTO> ב Notifier שמטרתו לעדכן על פוסט שפורסם זה עתה.
  + ב Listener שזהו ה FormMain יש מתודה updatePosts שמופעלת כאשר ה Notifier מבצע Invoke ושולח את תוכן הפוסט.
  + ה FormMain מקבל את העדכון ומעדכן בתצוגה בהתאם את הפוסט החדש.
* **Sequence Diagram**
* **Class Diagram**

