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

* הצגת "הצעדים הראשונים" של המשתמש בפייסבוק: הפוסט הראשון שהוא העלה, תמונת הפרופיל הראשונה, תמונת הCover- הראשונה, התמונה הראשונה שהוא העלה בקיר, התמונה הראשונה באלבום הראשון, הצ'ק-אין הראשון.
* הצגת המקומות האהובים שנבחרו על ידי המשתמש.

המידע מחושב על ידי כמות הפעמים שהמשתמש ביקר במקום מסוים, וממוין בסדר יורד, כך שהמקום המופיע ראשון ברשימה הינו "המקום הפוטנציאלי האהוב ביותר" של המשתמש.

### תבנית מס' 1 – Iterator

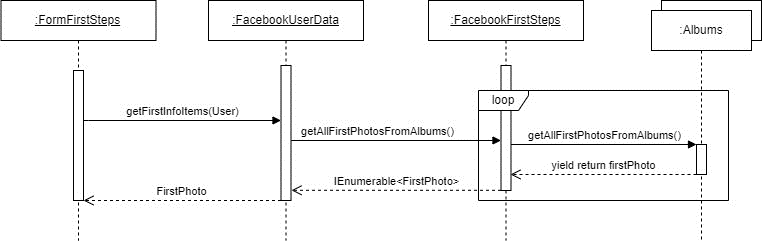
* סיבת הבחירה / שימוש בתבנית:

באפליקציה שלנו יש את המחלקה FacebookFirstSteps, המהווה Base class עבור אוסף כלשהו של Photos וPosts. אנו רוצים לאפשר מעבר (Traversal) על אוסף התמונות והפוסטים של מופע מסוים במחלקה זו מבלי שהצרכן יכיר את מבנה הנתונים או המימוש הפנימי של האוספים, בכדי שאם בעתיד נרצה לשנות את מבנה הנתונים שבו שמור האוסף, נוכל לעשות זאת מבלי שה- Client יצטרך לשנות את הקוד שלו. בנוסף, מובן שנרצה לאפשר את המעבר בזמן ריצה O(n) כאשר n הוא מספר האלמנטים במבנה הנתונים. לכן, בחרנו להשתמש בתבנית Iterator, העונה על צרכים אלו.

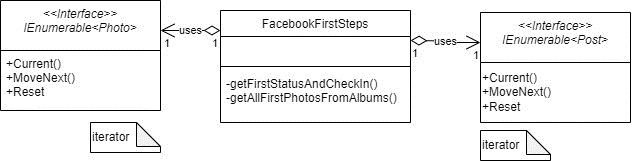
* אופן המימוש:

שינינו את המחלקה FacebookFirstSteps כך שהוספנו לה מתודות שמחזירות אובייקט של הממשק IEnumerable<Post> וגם IEnumerable<FirstPhoto> ובכך תהיה “*foreach-able*”.

* Sequence Diagram



* Class Diagram



### תבנית מס' 2 – Strategy

* סיבת הבחירה / שימוש בתבנית:

האפליקציה שלנו מציגה את המקומות האהובים על היוזר.

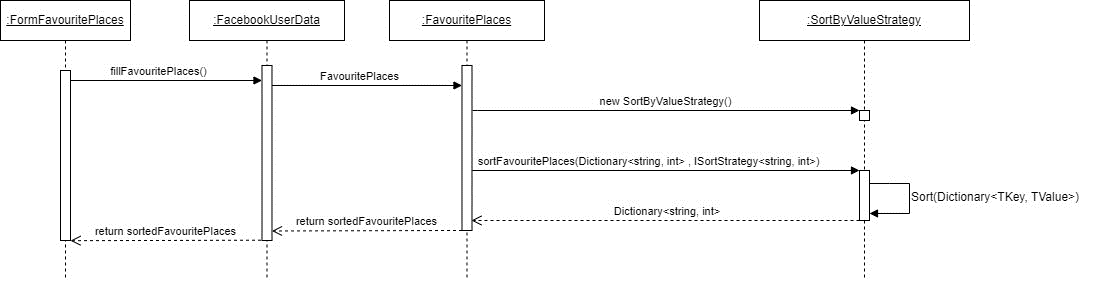
מכיוון שרשימה זו עלולה להיות ארוכה מאוד, ואנחנו רוצים להציג את המקומות האהובים ביותר בראש הרשימה, אנו ממיניים את המקומות לפי כמות הפעמים שהיוזר ביקר בהם. מיון המקומות והצגתם למשתמש היא לוגיקה אחידה העושה שימוש באותו אלגוריתם מיון.   
לכן, למקרה שבעתיד נרצה להוסיף אלגוריתם מיון אחר אשר ממיין באופן שונה את המקומות שהיוזר ביקר בהם, אנו נשתמש בתבנית Strategy המאפשרת לקבוע ולשנות בזמן ריצה את קריטריון המיון. בנוסף, תבנית זו מאפשרת לנו לתמוך בוריאציות מיון חדשות בעתיד מבלי לשנות את הלוגיקה; ה- Client רק צריך לבחור קריטריון מיון חדש ולהעביר אותו. כלומר, אנו מפרידים בין מה שמעניין את המשתמש (קריטריון המיון) לבין פרטי המימוש הטכניים של הלוגיקה (אופן המיון עצמו) שאינו רלוונטי ל- Client. כמו כן, השימוש בתבנית Strategy מונע מאיתנו יצירת מחלקות רבות המכילות קוד משותף, משמע מונע שכפול קוד.

* אופן המימוש:

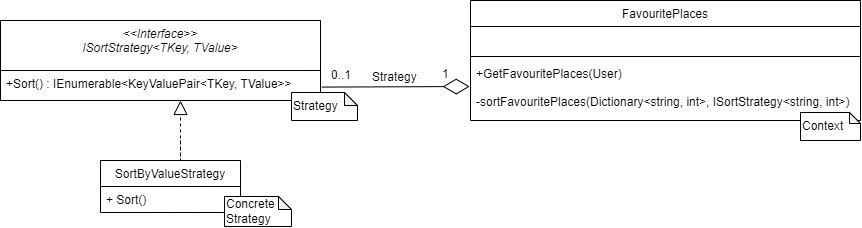
בתיקיית Logic בקובץ FavouritePlaces.cs   
יצרנו ממשק גנרי חדש בשםISortStrategy<TKey, TValue> (Strategy), המכריז על מתודה בוליאנית בשם Sort.  
לאחר מכן, עבור קריטריון מיון, יצרנו מחלקה שמממשת את ISortStrategy<TKey, TValue>

בנינו מחלקה שנקראת SortByValueStrategy היורשת מ- ISortStrategy<TKey, TValue> ובכך מאפשרת בחירת קרטריוני מיון נוספים במידה ובעתיד נרצה להוסיף פונקציות מיון.

* Sequence Diagram



* Class Diagram



### תבנית מס' 3 – Command

* סיבת הבחירה / שימוש בתבנית:

מהיות שקיימת בקוד אופציה להתחבר לממשק גם על ידי לחיצה על התחברות עם המשתמש השמור (במידה ויש), וגם על ידי לחיצה להתחברות עם משתמש אחר, נוצרת בעיה של שכפול קוד בצד של ה Client.

לכן השתמשנו בתבנית Command בין היתר על שינוי ה State של הכפתורים שיש לנו בתכנית.

באמצעות שימוש ב- pattern command, מנענו שכפול קוד מהיות שהפעלנו פעולה אחת, הרלוונטית לשתי אופציות ההתחברות ואפשרנו להשתמש באותו Command במקומות שונים במערכת, מה שכמובן הקל על התחזוקה העתידית שלנו בקוד.

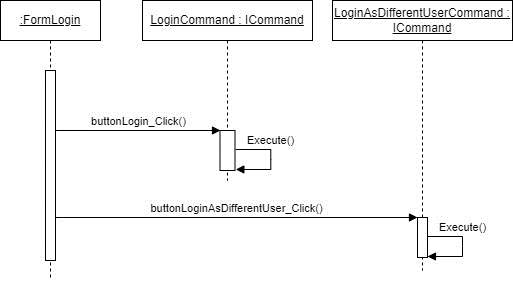
בנוסף תרמנו לקריאות הקוד שלנו, שכן ניתן לשנות מצבי כפתורים באמצעות פקודה אחת פשוטה – Execute().

* אופן המימוש:

יש בתיקיית הUI את הקובץ Icommand ואת המחלקות היורשות ממנו: LoginAsDifferentUserCommand, LoginCommand

המחלקות הללו ממשות את ICommand ובכך יוצרות לוגיקה מעט שונה עבור כל מחלקה אבל מימוש זהה באמצעות קריאה למתודת Execute() בשתי המחלקות באופן זהה.

* Sequence Diagram



* Class Diagram

