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

* פיצ'ר ראשון – GetRandomPhotoFromRandomAlbum:

בפיצ'ר זה אנחנו מספקים למשתמש לשחק במשחק שבו הוא צריך לנחש באיזה תאריך צולמה התמונה.  
המערכת מגרילה אלבום ותמונה מתוך האלבום.  
מערכת נותנת למשתמש לבחור את התאריך שהוא חושב שבו צולמה התמונה.  
המשתמש מקבל הודעה מתאימה בהתאם לבחירה.

* פיצ'ר 2 – DownloadAlbum:

הפיצ'ר נותן אופציה לבחירת אלבום מתוך אלבומי התמונות של המשתמש.   
ברגע שהמשתמש יבחר אלבום הוא יוכל לצפות בתמונות שנמצאות באלבום זה ולוודא שזה האלבום הרצוי.  בלחיצה על כפתור ההורדה תיפתח חלונית לבחירת נתיב שבו תיווצר תיקייה ולשם יורדו תמונות האלבום הנבחר.

### תבנית מס' 1 – [Strategy]

* **סיבת הבחירה / שימוש בתבנית:**

השתמשנו ב Strategy בפיצ׳ר ב- DownloadAlbumManager אשר ממיין את התמונות לפי בחירה שנבחרה ע"י המשתמש בממשק המשתמש.

מה זה מאפשר:

* + Extensability- כרגע המערכת מציעה 4 דרכים שונות למיין אלבום. אם בעתיד ירצו לממש או לבחור בדרכים אחרות למיין את האלבום, ניתן לעשות זאת בצורה יחסית נוחה, ללא צורך ש- DownloadAlbumManager יכיר את המחלקה החדשה ומבלי לשנות אותו.
  + גנריות- המערכת מממשת פעם אחת בלבד את מתודת המיון, ומי שמשתמש בה בוחר את אסטרטגיית המיון.
  + Maintainability- תחזוקת הקוד הממיין פשוטה בעלת פוטנציאל נמוך מאוד לבאגים.
* **אופן המימוש:**

1. כאשר המשתמש בוחר את אסטרטגיית המיון ובוחר אלבום להורדה ה- ISortPhotosStrategy שלנו מאותחל לאסטרטגיה המתאימה ב- FormMain.
2. בשלב הבא כאשר אנחנו רוצים להוריד את האלבום בהפעלת m\_SortPhotosStrategy.Sort(m\_AlbumToDownload) מופעלת האסטרטגיית מיוו הרצויה.

מיקום בקוד:

בתוך תיקיית SortStrategy קיימים:

* + ממשק (interface) ISortPhotosStrategy אשר מגדיר מתודת Sort אשר כל אסטרטגיה שנבחר למיון, תממש את ממשק זה.
  + מחלקות:

\* SortByCreationTimeAscending.

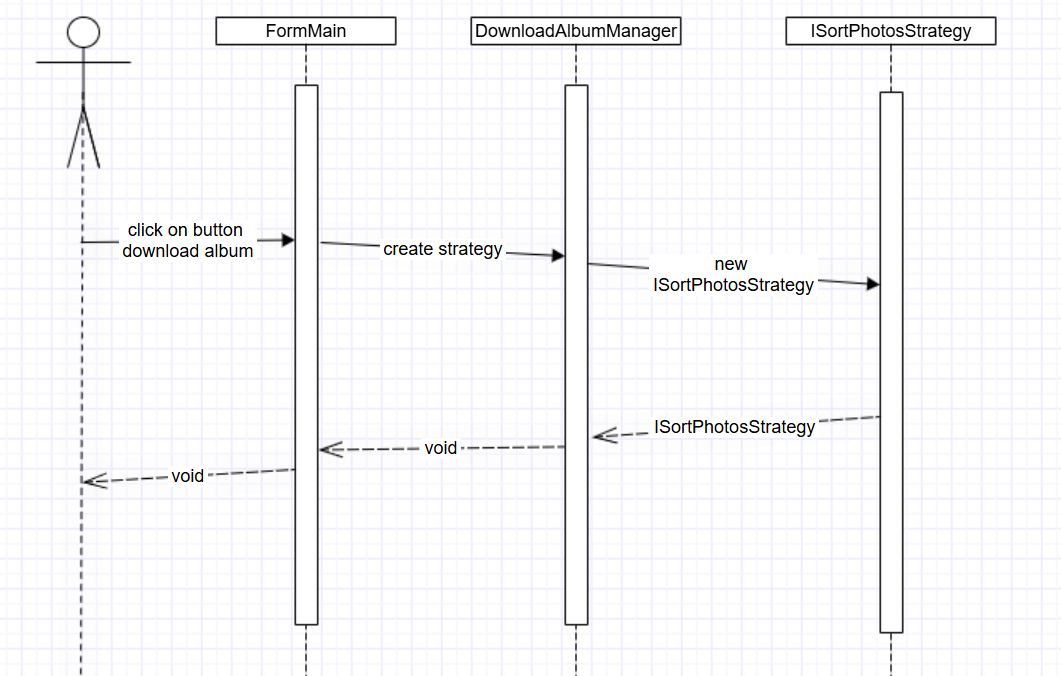
\* SortByCreationTimeDescending.

\* SortByCommentsAmountAscending.

\* SortByCommentsAmountDescending.

אשר מממשות את הממשק לפי האסטרטגיה הספציפית שנבחרה.

* + המחלקה DownloadAlbumManager שמחזיקה data member מסוג ISortPhotosStrategy בשם m\_SortPhotosStrategy שמאותחלת באסטרטגיה המתאימה בהתאם לבחירה של המשתמש בממשק המשתמש (מאותחלת בFormMain –).
  + לאחר בחירת האלבום שנרצה להוריד מפעילים את הפונקציה של המיון המתאימה בעזרת: m\_SortPhotosStrategy.Sort(m\_AlbumToDownload).
* **Sequence Diagram:**



* **Class Diagram:**

תמונה שמכילה טקסט, צילום מסך, תרשים, גופן

תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי.

### תבנית מס' 2 – [Observer]

* **סיבת הבחירה / שימוש בתבנית:**

תבנית Observer נותנת מענה לצורך בניהול תלות בין אובייקטים בתוך מערכת. כאשר אובייקטים מסוימים צריכים להודיע לאחרים על שינויים במצבם, תבנית זו מציעה מנגנון יעיל להודעות ועדכונים אוטומטיים.

בהקשר של אפליקציית הפייסבוק, על ידי שימוש ב- Observer, אנו מאפשרים ל-SystemManager ליידע את ה- FormLogin ו- FormMain באופן מיידי על כל אירוע התחברות שצלח או לא צלח. בייחוד בסביבה מרובת תהליכים, כשיש צורך בכמה פעולות ואירועים במקביל, הObserver מקנה לנו תקשורת יעילה ותקינה בין האובייקטים גם כאשר יש תלות בין האירועים.

בנוסף, תבנית זו מקנה לנו קריאות בקוד כאשר ניתן בקלות לראות כיצד רכיבי המערכת מתנהלים וכיצד הם מתקשרים זה עם זה. כמו כן, אנחנו מייעלים את ניהול הקוד בעת שילוב שינויים או הכנסת פונקציונליות חדשה למערכת.

* **אופן המימוש:**

1. כאשר המשתמש לוחץ על כפתור הכניסה (buttonLogin\_Click), נקרא ל- LoginAndInit דרך SystemManager.

2. אם הכניסה הצליחה, מתודת notifyObservers נקראת עם הערך `true`, מה שמציין שהמשתמש מחובר והיא בתורה עוברת על כל האובזרברים ומפעילה את UpdateLoginStatus עם הערך `true` .

3. ה- FormMainו- FormLogin מתווספים כאובזרברים ל- SystemManager בקונסטרקטור שלהם.

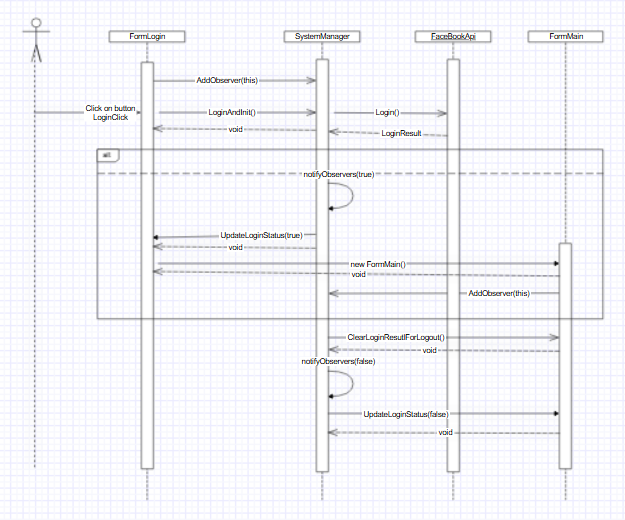
4. כאשר סטטוס ההתחברות משתנה, SystemManager מודיע ל-FormLogin ול- FormMain על ידי הפעלת המתודה UpdateLoginStatus עם סטטוס ההתחברות המתאים כלומר התחברות או התנתקות מהמערכת.

5. ה- MainForm מעדכן את רכיבי ממשק המשתמש שלו בהתבסס על מצב ההתחברות, כמו שינוי טקסט לחצן ההתחברות.

מיקום בקוד:

1. הגדרת ממשק IFacebookObserver :
   * מכיל מתודה UpdateLoginStatus - שנממש במחלקות שמממשות את הממשק ונפעיל אותה כאשר יש שינוי במצב ההתחברות של המשתמש.
2. עדכון SystemManager :
   * הוספת רשימה של אובזרברים .m\_FacebookObservers -
   * מימוש מתודות AddObserverו RemoveObserver-לניהול האובזרברים.
   * הוספת מתודה פרטית notifyObservers שקוראת למתודות העדכון של כל האובזרברים.
3. עדכון FormLogin :
   * הוספה שלו לרשימת האובזרברים בקונטרולר.
   * מימוש הממשק - IFacebookObserver מימוש המתודה UpdateLoginStatus שמעדכנות את ממשק המשתמש בהתאם לשינויים.
4. עדכון FormMain :
   * הוספה שלו לרשימת האובזרברים בקונטרולר.
   * מימוש הממשק IFacebookObserver - מימוש המתודה UpdateLoginStatus שמעדכנות את ממשק המשתמש בהתאם לשינויים.
5. שילוב בתהליך ההתחברות \ התנתקות:
   * בעת התחברות או התנתקות, SystemManager קורא לnotifyObservers.
   * FormMain ו- FormLogin מקבלים עדכונים ומשנים את ממשק המשתמש בהתאם.

* **Sequence Diagram**



* **Class Diagram**

תמונה שמכילה טקסט, קו, גופן, תרשים

תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי.

### תבנית מס' 3 – [Iterator]

* **סיבת הבחירה / שימוש בתבנית:**

תבנית העיצובIterator נבחרה כדי לאפשר מעבר נוח ואחיד על אוסף הפוסטים של המשתמש .Facebook המימוש מאפשר לעבור על הפוסטים בצורה גמישה תוך שמירה על אינקפסולציה והימנעות מחשיפת המבנה הפנימי של האוסף.

* **אופן המימוש:**

1. הוגדרה מחלקה PostCollection המיישמת את IEnumerable<Post>, כך שהיא מאפשרת איטרציה על אוסף הפוסטים.
2. בתוך PostCollection יש מחלקה פנימית PostIterator המיישמת את IEnumerator<Post>,

אשר אחראית על ניהול האיטרציה:

* אינדקס m\_CurrentIdx שומר את המיקום הנוכחי באוסף.
* MoveNext()- מקדם את האינדקס ובודק שהאוסף לא השתנה במהלך האיטרציה.
* -Reset() מאפס את האינדקס.
* -Current מחזיר את הפריט הנוכחי, עם בדיקות חריגה מתאימות.
* GetEnumerator()- מחזיר מופע חדש של PostIterator, כך שכל לולאת foreach יכולה לפעול על המופע בנפרד.

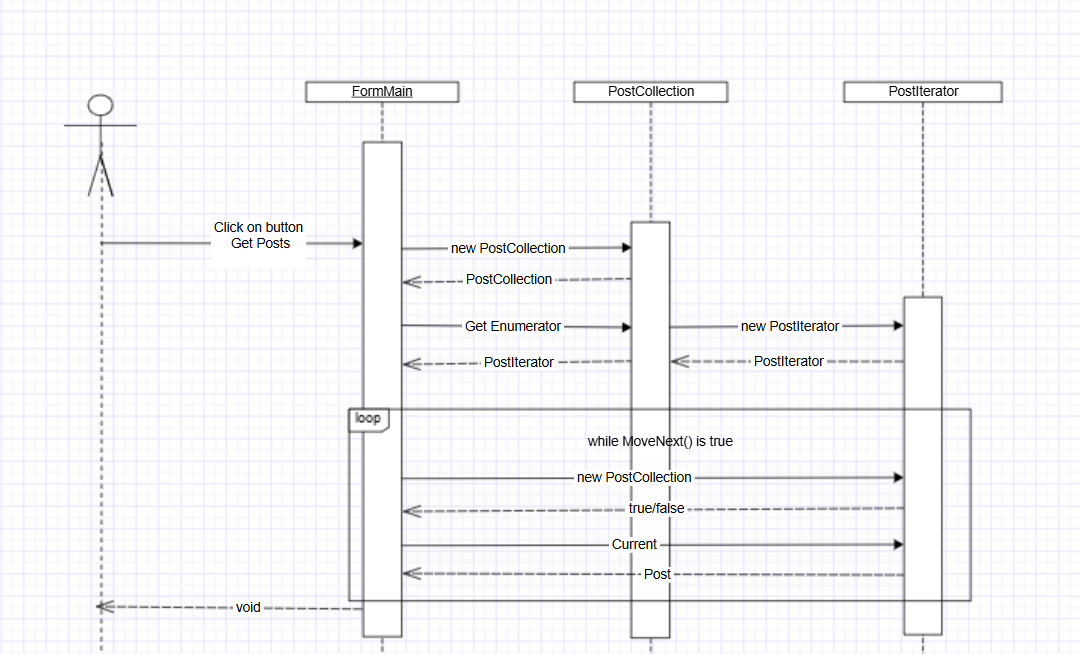
מיקום בקוד:

-המחלקה PostCollection מכילה את אוסף הפוסטים (r\_Posts).

-מתודת GetEnumerator() מחזירה את האיטרטור PostIterator.

PostIterator- - איטרטור לאובייקט Post

* **Sequence Diagram**



* **Class Diagram**

