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

* פיצ'ר 1: קולאז' תמונות

המשתמש יכול להשתמש בפיצ'ר על ידי לחיצה על אופציית ה-Collage שנמצאת בתפריט הצד של האפליקציה.

ברגע שהמשתמש לוחץ על הכפתור Fetch all photos, מוצגות לו כל התמונות המשויכות למשתמש הפייסבוק שלו ב-Listbox, שמתוכו הוא יכול לסמן עד 4 תמונות (על ידי לחיצה על ctrl + left mouse click כדי לסמן או לבטל סימון תמונה), ובסיום הבחירה ללחוץ על הכפתור Make collage, שלאחר לחיצה עליו מוצג לו קולאז' מהתמונות אותן הוא בחר.

הפיצ'ר ממומש במחלקה FormCollage.

* פיצ'ר 2: תצוגה ממויינת של אלבומי תמונות לפי פרמטרים

המשתמש יכול להשתמש בפיצ'ר על ידי לחיצה על אופציית ה- Albums בתפריט הצד של האפליקציה.

ברגע שהמשתמש לוחץ על Fetch all albums, מוצגים לו כל האלבומים המשויכים למשתמש הפייסבוק שלו ב- Listbox.

לאחר מכן המשתמש יכול לבחור את אחת מן האופציות של המיון (לפי תאריך יצירת האלבום או לפי כמות התמונות באלבום) וללחוץ על הכפתור Sort albums כדי לראות את האלבומים בצורה ממוינת לפי בחירתו.

הפיצ'ר ממומש במחלקה FormAlbum.

### תבנית מס' 1 – Singleton

* **סיבת הבחירה / שימוש בתבנית:**

השתמשנו בסינגלטון עבור מחלקה בשם FacebookUserManager. מחלקה זו מחזיקה את לוגיקת ההתחברות למערכת וכן את המשתמש המחובר.

בחרנו לממש מחלקה זו כסינגלטון מהסיבות הבאות:

* דרוש מופע אחד בלבד למחלקה זו, מכיוון שהיא אחראית לניהול הלוגיקה ויש צורך במנהל אחד כזה בדיוק. אם היו כמה מופעים של מחלקה זו, אז היה מדובר בכשל לוגי שהרי זה לא נכון לייצר מס' מנהלים שמייצגים למעשה את אותה הישות במערכת. ובנוסף, אנחנו לא רוצים כפילות עבור הלוגיקה.
* אם בעתיד נרצה להוסיף פיצ'רים נוספים, נוסיף אותם תחת המחלקה FacebookUserManager ומכיוון שמחלקה זו היא סינגלטון, אז יובטח לנו שגם פיצ'רים אלו יופיעו ללא כפילות.
* מכיוון שכל פעולה שקשורה בהבאת המידע מתחילה בגישה אל ה FacebookUserManager, אז יש אליו נקודות גישה רבות מרכיבים שונים ביוזמתם – ולכן פטרן זה בא לעזרתנו גם עבור סיבה זו.

* אופן המימוש:

מחלקת FacebookUserManager ממומשת כסינגלטון באופן הבא:

* המחלקה היא sealed.
* הבנאי הוא בהרשאת גישה .Private
* המחלקה מכילה שדה סטטי מטיפוס המחלקה – הנוצר רק בעת הבקשה הראשונה לקבלת המופע היחיד (lazy initialize), ותהליך יצירתו מוגן באמצעות מנגנון double check lock. השדה מאותחל לnull.
* מתודה סטטית Instance() אשר בפנייה ראשונה מייצרת את המופע הסטטי, והחל מהפנייה הבאה רק מחזירה אותו.

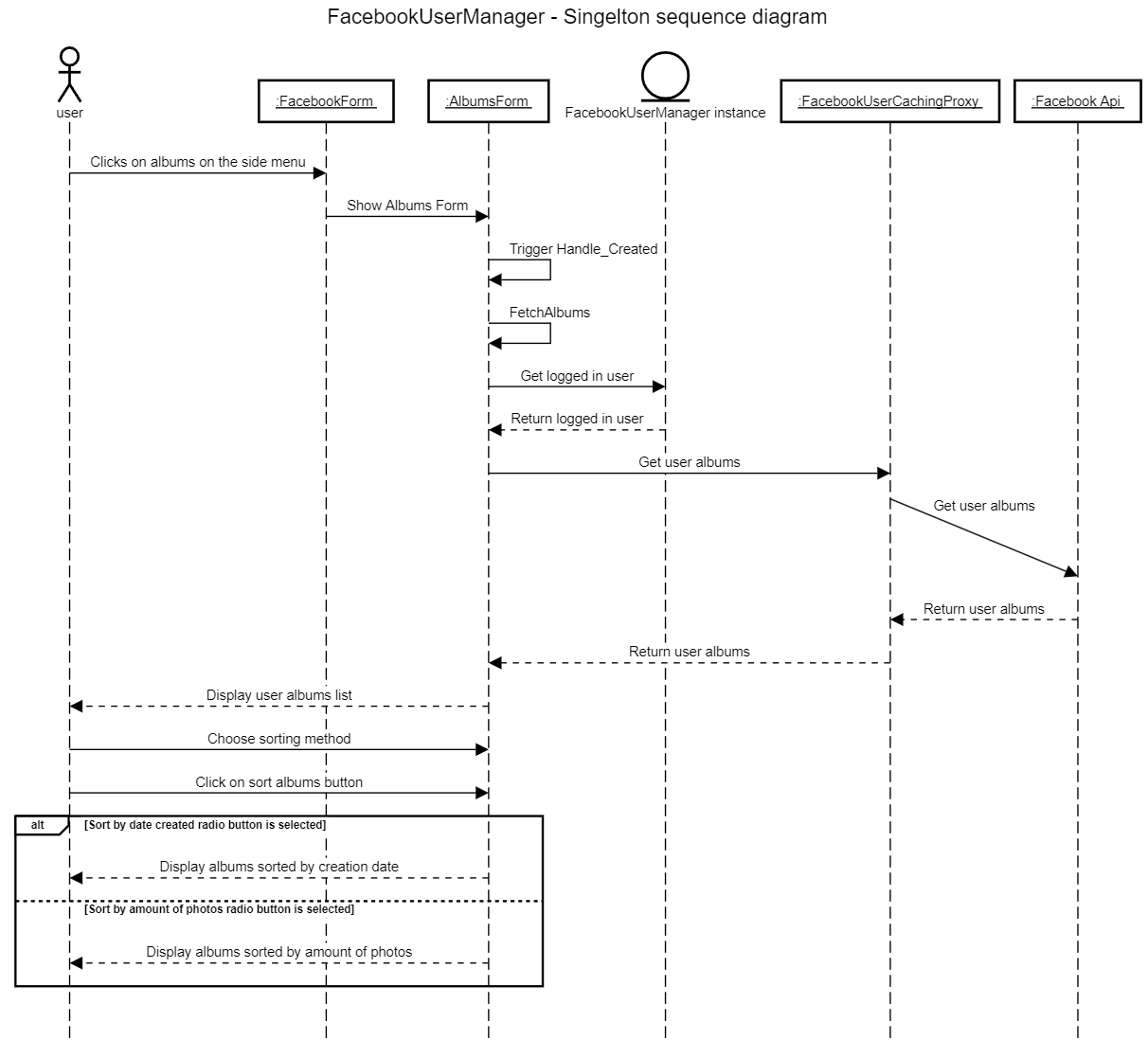
**FacebookUserManager הוא הסינגלטון.**

* Class diagram

Diagram

Description automatically generated

* Sequence diagram



### תבנית מס' 2 – Factory Method

* **סיבת הבחירה / שימוש בתבנית:**

השתמשנו ב design pattern factory method על מנת שאם בעתיד נרצה להרחיב את האפליקציה שלנו ולהוסיף טפסים נוספים, נוכל לעשות זאת בצורה קלה ובמינימום שינוי בקוד הלוגי שאינו יושב במחלקת ה-

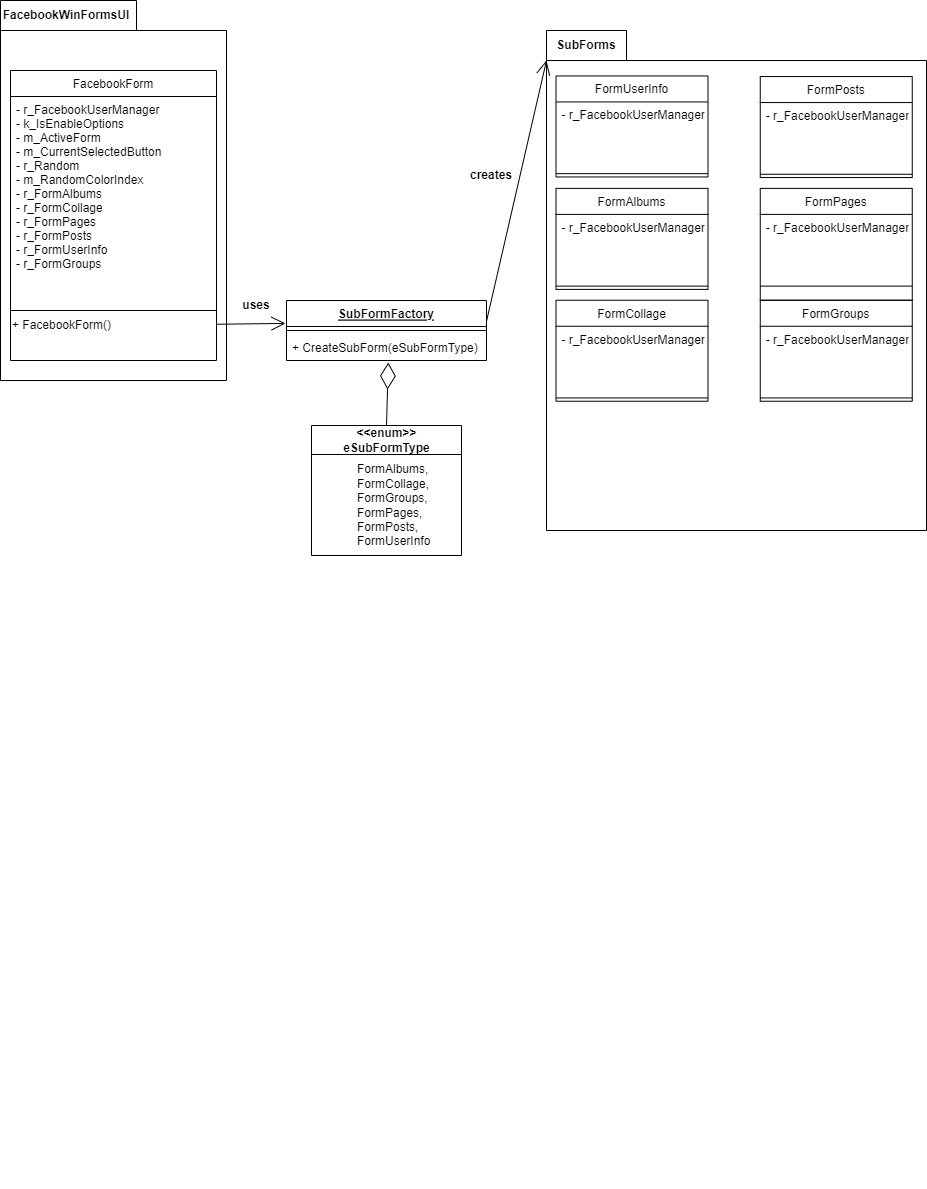
Factory. בנוסף רצינו שהטופס הראשי של האפליקציה לא יכיר את תתי הטפסים של האפליקציה ובכך מנענו תלות מעגלית בין המחלקות.

* **אופן המימוש:**

יצרנו את המחלקה SubFormFactory אשר יושבת תחת הפרוייקט FacebookWinFormsUI.

בתוך המחלקה החזקנו Enum eSubFormTypeאשר מייצג את שמות תתי הטפסים השונים אשר המחלקה מסוגלת לייצר. במחלקה קיימת מתודה סטטית CreateSubForm אשר מקבלת כפרמטר eSubFormType ולפיו מייצרת תת טופס מתאים ומחזירה אותו כפלט.

* Class diagram



* Sequence Diagram

A picture containing table

Description automatically generated

### תבנית מס' 3 – Caching Proxy

* סיבת הבחירה / שימוש בתבנית:

השתמשנו בתבנית זו עבור המחלקה FacebookWrapper.ObjectModel. User,

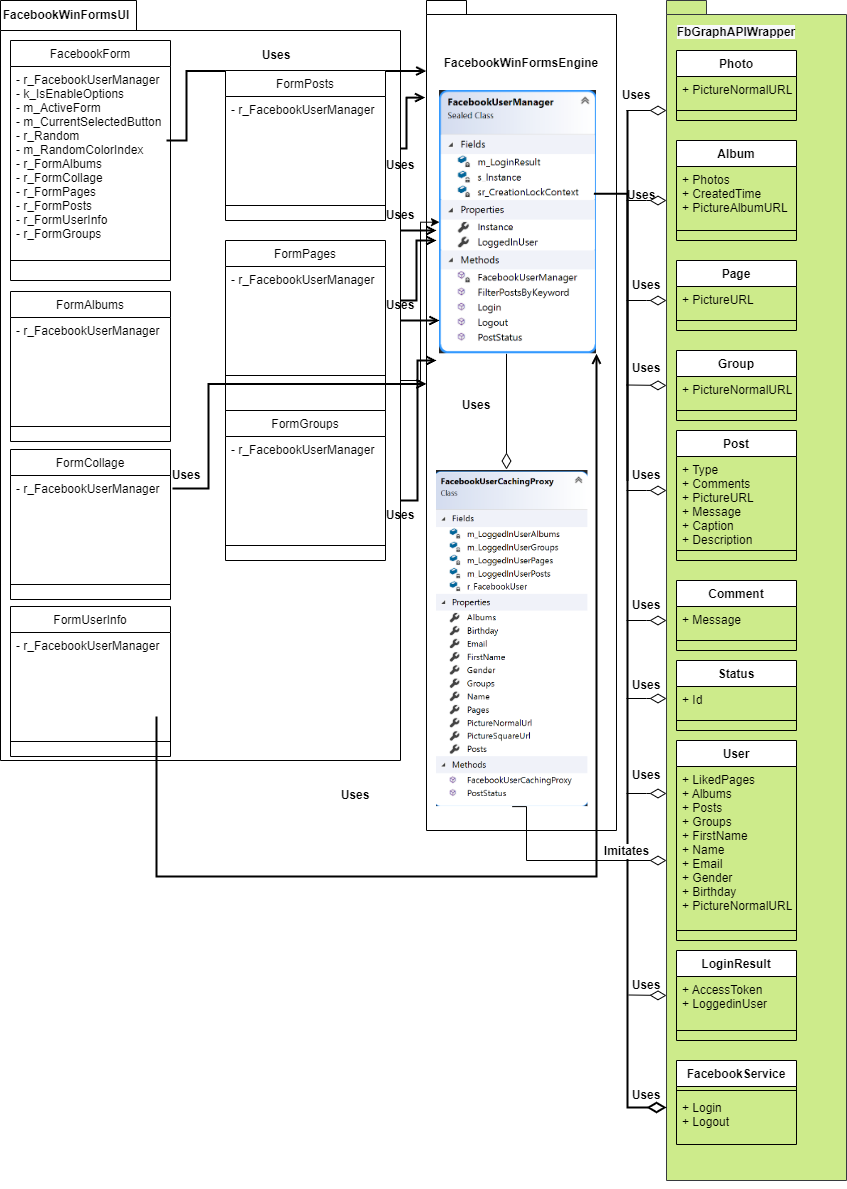
על מנת לשפר את ביצועי המערכת.

כאשר מושכים מידע באמצעות ה- Properties של המחלקה User, מתבצעת משיכת מידע משרתי פייסבוק שעלולה לקחת זמן רב, לכן רצינו להגיע למצב שאם כבר משכנו מידע כלשהו משרתי פייסבוק, נעשה שימוש חוזר במידע שמשכנו ולא נצטרך להמתין למידע זהה שיגיע שוב משרתי פייסבוק, התבנית Caching Proxy עזרה לנו לפתור בעיה זו.

* אופן המימוש:

יצרנו מחלקה בשם FacebookUserCachingProxy, אשר יש לה ממשק זהה למחלקה User במובנים שבה רכיבי האפליקציה שלנו משתמשים בה, אך השוני בה הוא שעבור ה-Properties שמושכים רכיבים כבדים יחסית שלוקח זמן ארוך להביאם מהשרתים, בפעם הראשונה שבה אנחנו מושכים את המידע מהשרת עבור הProperty המסויים, אנחנו שומרים בממבר במחלקה את המידע שמשכנו, ובפעמים הבאות שימשך המידע המחלקה מחזירה את המידע הלוקאלי שיושב בממבר שלה ולא מושכת שוב מידע מהשרת ובכך חוסכת זמן הפעלה ארוך ומשפרת את ביצועי המערכת.

* Class diagram



* Sequence diagram

Diagram

Description automatically generated

### עבודה אסינכרונית

* **היכן מומש?**

בכל תת-טופס של האפליקציה תחת המטודה שמקשיבה לאירוע HandleCreated ותחת המטודה שמקשיבה לאירוע buttonFetch…\_Click.

* **איך מומש?**

בכל מחלקה של תת טופס, נרשמנו לאירוע HandleCreated על מנת לדעת מתי הטופס נוצר ואותחל.

לאחר שתת הטופס נוצר ואותחל, נוצר thread שתפקידו למשוך את המידע הרלוונטי לכל תת טופס ומופעל אוטומטית ברגע שהמשתמש בוחר לראשונה את תת הטופס בתפריט.

בנוסף, בכל טופס ישנה אפשרות ללחוץ על כפתור אשר מושך גם הוא את אותו מידע רלוונטי ב thread נפרד.

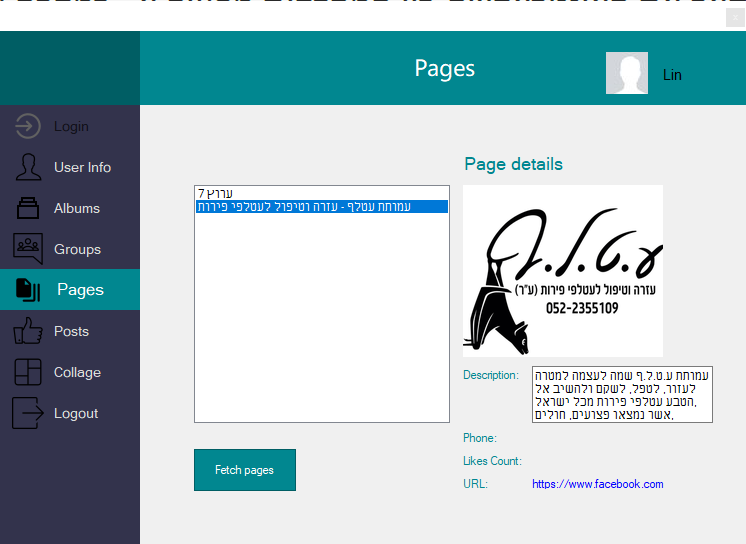
כל thread כזה דואג להציג את המידע הרלוונטי שאותו הוא משך בפקדים המתאימים שנמצאים על הטופס ועבור כל פקד כזה השתמשנו ב invoke אשר דואגת להפעיל את המתודה הרלוונטית ב thread אליו הוא שייך.

* **מדוע מומש?**

על מנת לאפשר למשתמש שימוש חלק ומקביל ברכיבי המערכת השונים ללא צורך להמתין לטעינת המידעים השונים.

### עבודה עם Data Binding

* **היכן מומש?**

במחלקת FormPages

Text

Description automatically generated