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

* **פיצ'ר 1: הורדת האלבומים.**

פיצ'ר הורדת האלבומים מאפשר למשתמש לבחור מתוך רשימת האלבומים המוצגת לפניו, על ידי יכולת בחירה מרובה, להוריד את כל התמונות מהאלבומים שהוא בחר לתיקיה ייעודית במחשב.

* **פיצ'ר 2: מציאת החבר "הטוב" ביותר.**

האפליקציה מציגה את רשימת הדפים שהמשתמש עשה להם לייק בחשבון הפייסבוק שלו. על ידי לחיצת כפתור האפליקציה תמצא את החבר שיש לו הכי הרבה דפים משותפים שהוא והחבר עשו להם לייק. האפליקציה תציג בחלון ייעודי את רשימת הדפים המשותפים לו ולחבר ותציג את התמונה והשם של החבר "הטוב" ביותר.

### תבנית מס' 1 – [Class Proxy (Virtual Proxy )]

* סיבת הבחירה / שימוש בתבנית:

אנו רוצים שיהיה לנו PictureBox שניתן בלחיצה על התמונה להגדיל ולהקטין את גודל התמונה ולכן השתמשנו בתבנית Class Proxy שמתחזה ל pictureBox המקורי והרחבנו אותה ללוקיגה הרצויה וכעת יש לנו ResizePictureBoxProxy שאנחנו יכולים לעשות REUSE בכל המערכת שלנו מתי שנרצה תמונה כזאת.

* אופן המימוש:

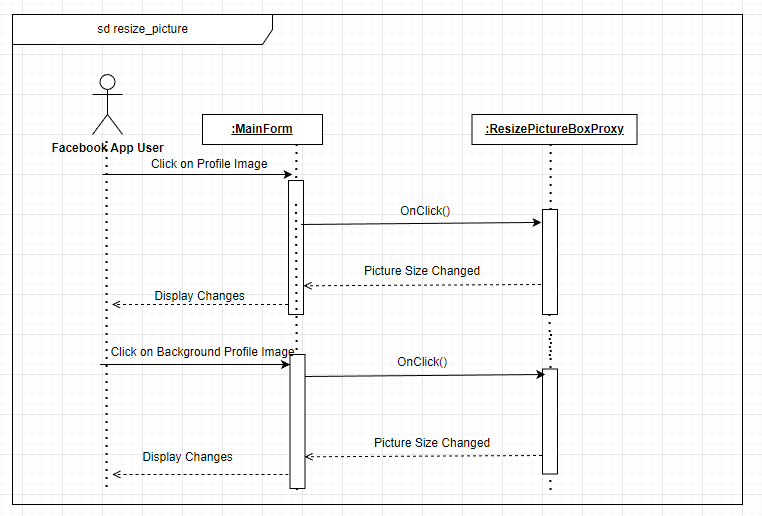
המחלקה ResizePictureBoxProxy יורשת PictureBox היא בעצם הPROXY שלנו ובעצם עושים OVERRIDE למתודה OnClick ובכך אנו עושים RESIZE בלחיצה על התמונה ובכך אנו יכולים להוסיף לטופס שלנו פקדים שהם יהיו נתנים לשינוי בלחיצה.

עשינו שימוש בתמונת פרופיל ובתמונת נושא בדף הראשי של הטופס.

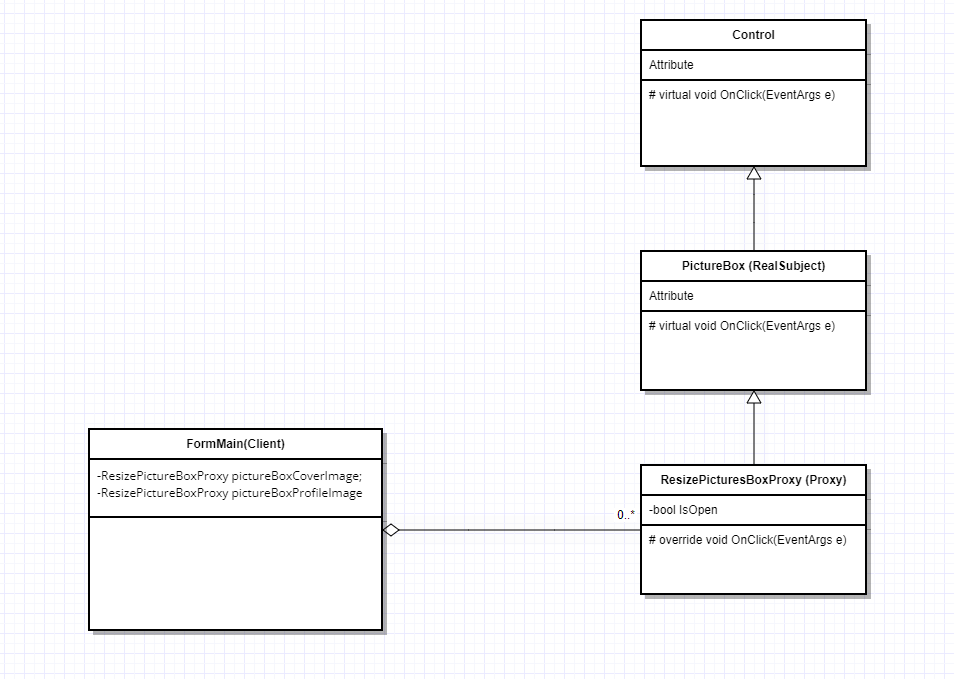
ניתן למצוא בקובץ FromMain.designer.cs בשורות 45-47, 527-528

וגם מימוש התבנית בקובץ ResizePictureBoxProxy.cs

* Sequence Diagram



* Class Diagram



### תבנית מס' 2 – [Singleton]

* סיבת הבחירה / שימוש בתבנית:

יש לנו את האוביקט LoggedInUser שאנו מקבלים מ loginResult אחרי התחברות משרתי פייסבוק אנו רוצים שהאובייקט זה יחיד במערכת ושלא יהיה ניתן להתחבר דרך משתמש אחר או לשנות את האובייקט מכייון שהמערכת שלנו תומכת רק במשתמש אחד והתחברות עם משתמש נוסף יהיה שגוי ובנוסף אנו רוצים שאובייקט זה יהיה חשוף לכול חלקי המערכת.

לכן יצרנו מחלקה LoggedInUserSingleton שתהיה סיגלטונית במערכת ורק דרכה נבצעה התחברות לפייסבוק ונחשוף את LoggedInUserשיהיה זמין לכול מי שמבקש במערכת.

* אופן המימוש:

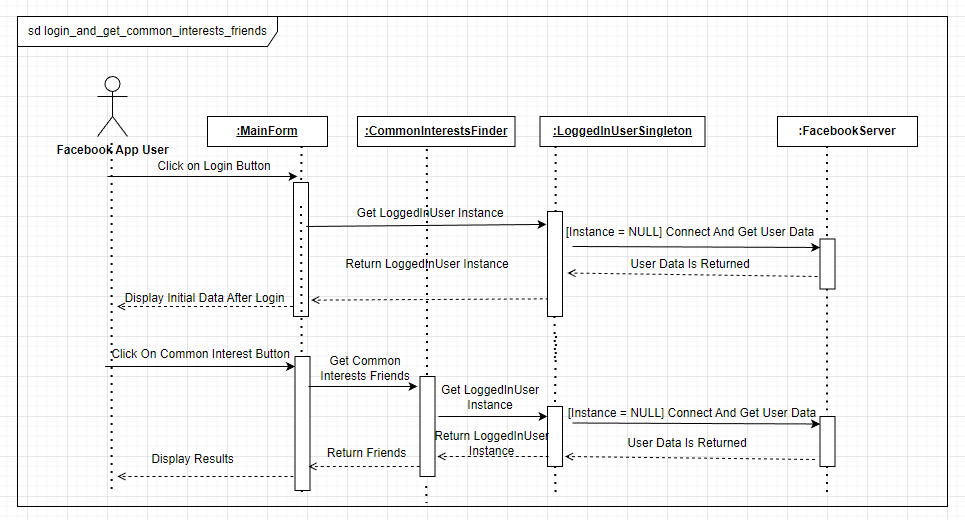
יצרנו מחלקה LoggedInUserSingleton שהיא תהיה sealed וגם עם בנאי פרטי ובעצם היא המחלקה הסיגלטונית שלנו. בתוך המחלקה מימשנו pattern זה ע"י שדה סטטי מטיפוס המחלקה (LoggedInUserSingleton) ובנוסף על מנת שהמחלקה תהיה גם thread safe השתמשנו באובייקט lock מטיפוס object. הוספנו מנגנון לטיפול בשגיאות. ובנוסף אנו טוענים את האוביקט רק שמבקשים אותו fully-lazy.

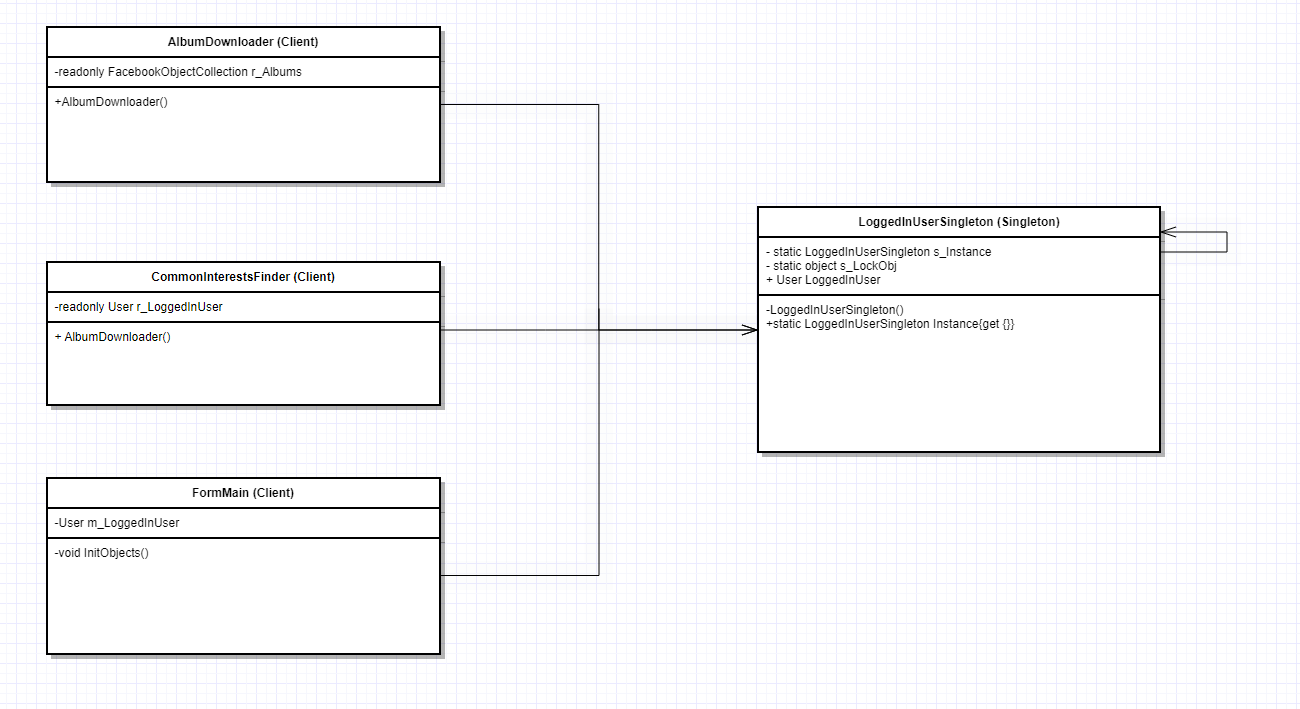
בתוך המחלקה יש לנו פרופרטי LoggedInUser שרק כאשר אנו מבצעים התחברות דרך שרתי פייסבוק רק אז אנחנו טוענים אותו.

(הפרופרטי LoggedInUser ציבורי עם GET ציבורי וSET פרטי כך שאפשר לגשת אליו רק דרך ה Instance ולא ניתן לשנות אותו)

ובכך תהיה גישה מכול מקום במערכת דרך LoggedInUserSingleton.Instance.LoggedInUser

ניתן למצוא בקובץ LoggedInUserSingleton.cs

* Sequence Diagram
* Class Diagram



### תבנית מס' 3 – [Adapter]

* סיבת הבחירה / שימוש בתבנית:

על מנת שנוכל לשמור תמונות מאפליקציית הפייסבוק שלנו למחשב בפורמט אחיד, ובלי לממש את הלוגיקה והמניפולציות על הנתונים בכל מקום שנרצה לשמור תמונה, יצרנו את מחלקת PhotoAdapter.  
בעזרת המחלקה הזו בכל מקום שנרצה לבצע שמירה של תמונה כמו השימוש במחלקה AlbumDownloader , נוכל להשתמש במתאם על מנת לשמור בצורה אחידה את התמונות שנרצה להוריד.

* אופן המימוש:

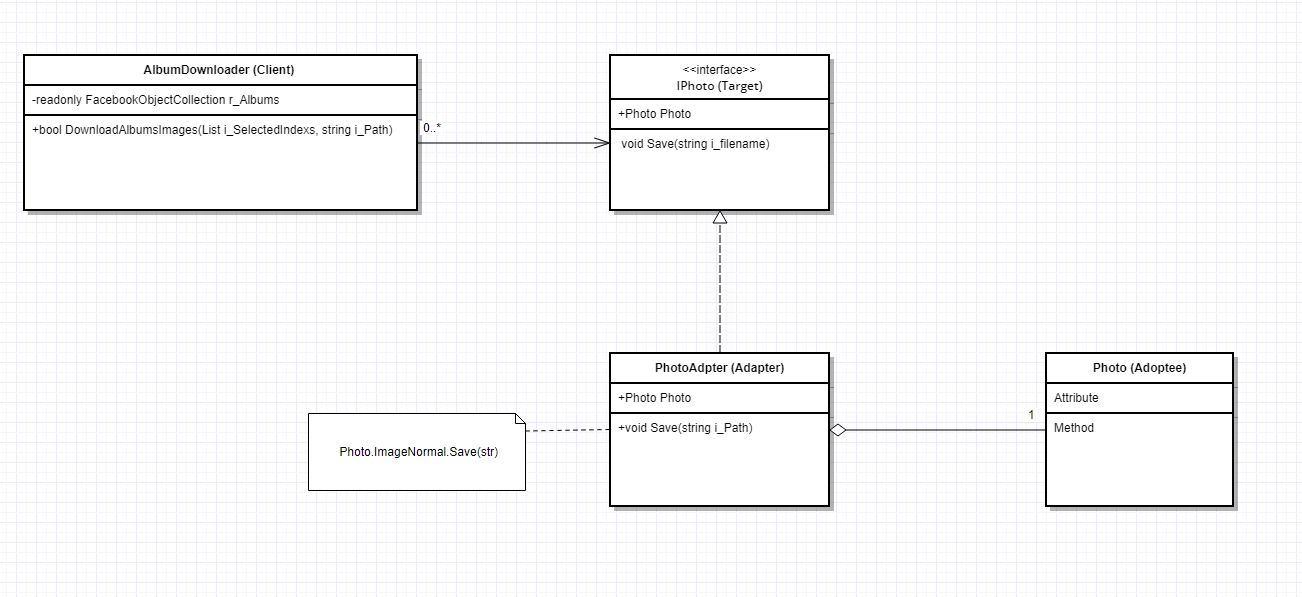
יצרנו מחלקה בשם PhotoAdapter אשר היא המתאם בין הלקוח שלנו AlbumDownloader לבין רכיב הLegacy Photo .  
מחלקת PhotoAdapter מממשת את הInterface IPhoto שמכיל את המתודה האבסטרקטית Save().

מחלקת הלקוח שלנו מעוניינת לשמור תמונה ורוצה לשמור על ממשק אחיד שבו מעבירים את נתיב התמונה ומחלקת המתאם שלנו מבצעת את הלוגיקה הפנימית שנדרשת על מנת לשמור תמונה בפורמט אחיד.

ניתן למצוא בקובץ PhotoAdapter.cs   
ובקובץ AlbumDownloader.cs בשורות 25-29

* A picture containing text, diagram, screenshot, parallel

  Description automatically generatedSequence Diagram
* Class Diagram



**עבודה אסינכרונית עם ממשק משתמש**

לאחר התחברות המשתמש לאפליקציית הפייסבוק שלנו אנחנו רוצים לטעון נתונים רבים שמגיעים משרתי הפייסבוק לאפליקציה.  
תהליך זה לוקח לא מעט זמן ואנחנו רוצים לבצע אותו בצורה אסינכרונית על מנת לאפשר למשתמש לאחר שהתחבר לבצע פעולות נוספות מבלי שהאפליקציה "תתקע" ותחסום לו את היכולת לבצע פעולות נוספות במקביל ובכך תספק חוויה נעימה יותר למשתמש.

לכן המתודה initData בשורה 33 בקובץ FormMain.cs תופעל בThread נפרד שבו אנו מייצרים Thread-ים נוספים עבור טעינה של רכיבים שונים (שורות 52-55 בקובץ FormMain.cs)

המטרה בשימוש בתיכנות אסינכרוני הוא לאפשר למשתמש לאחר שהתחבר לבצע פעולות נוספות מבלי שהאפליקציה "תתקע" ותחסום לו את היכולת לבצע פעולות נוספות במקביל, ובכך, גם תורמת לשיפור חווית המשתמש.

**עבודה עם Binding Data**

באפליקציה שלנו השתמשנו ב-Data Binding לחיבור בין אוסף הPages שמתקבל בLikedPages לבין רשימת הדפים שהמשתמש אהב בטאב Common Interests וגם לרכיב המידע נוסף שמציג את שם העמוד והתיאור שלו. (שורות 71-81 בקובץ FormMain.cs)

ע"י ה-Data binding אנו יכולים לקשר בין העמודים שהמשתמש אהב, אותם אנחנו מקבלים מיד לאחר ההתחברות, לבין שני הרכיבים שמציגים מידע על העמודים שהמשתמש אהב בטאב Common Interets.