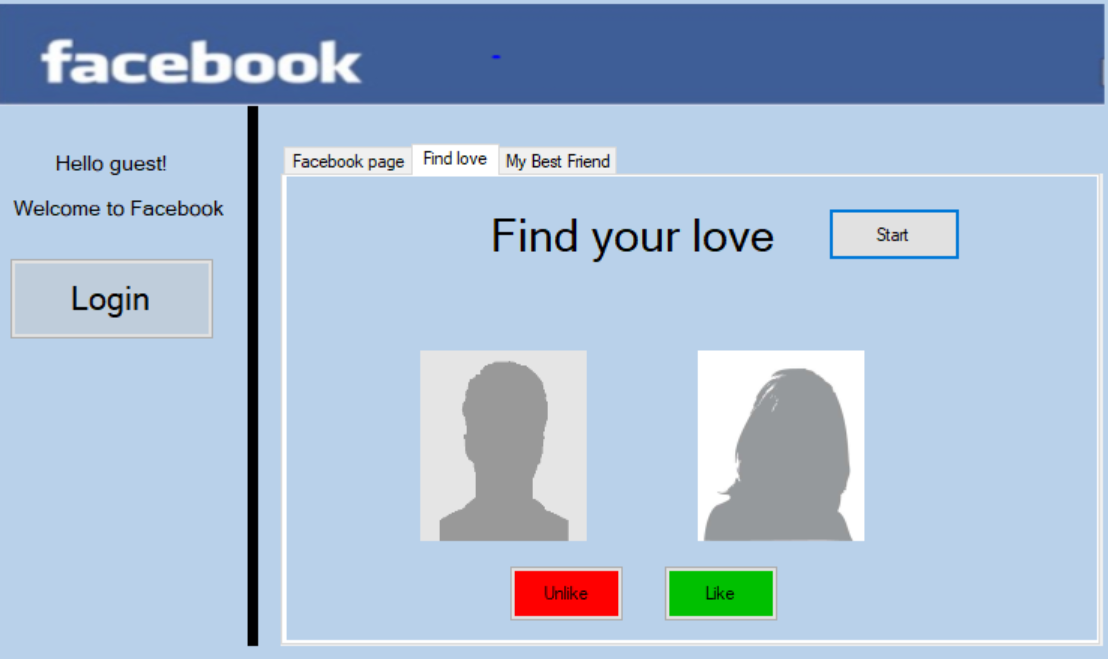
### תיאור קצר של הפיצ'רים שבחרנו לממש בתרגיל הקודם:

* **Find Love**-

פיצ'ר זה מטרתו מציאת בן זוג למשתמש הפייסבוק מתוך מאגר החברים של חברי הפייסבוק שלו. הפיצ'ר עובד כך שתמונה אקראית של חבר פייסבוקי קופצת למסך ולמשתמש ישנה אפשרות לסמן like אם אהב או לעבור לתמונה של מישהו אחר.

כאשר המשתמש אהב את אחת התמונות, ישנה אפשרות לשלוח הודעה על מנת להתחיל שיחה חדשה.



* **Top Ten FaceBook Friends-**

פיצ'ר זה מציג למשתמש רשימה של חבריי הפייסבוק הכי טובים שלו.

רשימת החברים מחושבת לפי פרמטרים שונים , כמו: מספר לייקים, מספר תגובות וכו'.



### תבנית מס' 1 – [Singelton]

* **סיבת הבחירה / שימוש בתבנית:**

בחרנו לממש את תבנית סינגלטון עבור המחלקה FacebookManager במטרה להגביל את יצירת המופעים של המחלקה למופע אחד בלבד.

מחלקה זו תפקידה לנהל את הלוגיקה של האפליקציה, השדה המרכזי שלה הוא LoggedInUser שכן לא ניתן להיות מחובר בו זמנית משני משתמשים כאשר האפליקציה רצה, לכן לא נרצה שתהיינה אפשרות לייצר מופע נוסף ממחלקה זאת.

בנוסף, לא נרצה יותר ממופע אחד של המחלקה FacebookManager מכיוון שכל מופע אחר של מחלקה זו צריך להיות בעל State זהה (בזבוז של מקום בזיכרון),

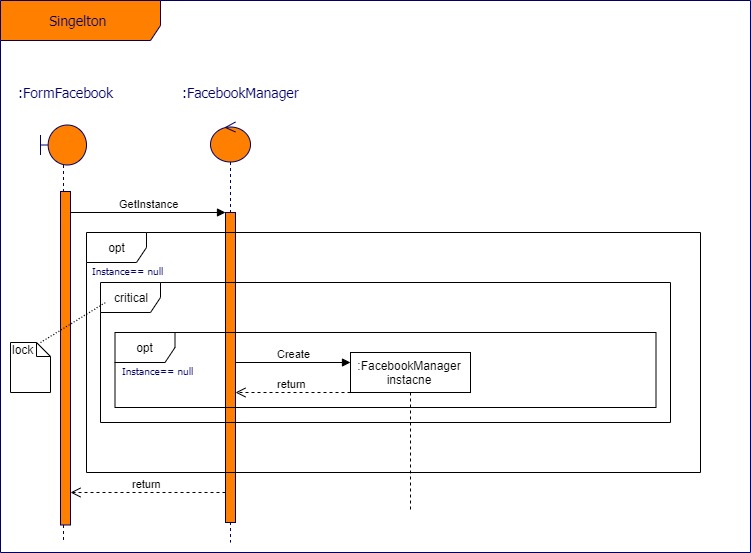
וכן 2 מופעים של FacebookManager עם State שונה הוא מצב לא תקין במערכת מהסיבות שתוארו מעלה.

בנוסף, מחלקה זו מחזיקה את כל הלוגיקה של המערכת, היא משתמשת ברכיבים רבים, לכן נרצה שאם בעתיד מישהו ירצה להשתמש במחלקה, הוא לא יצטרך להעביר רפרנס של המחלקה.

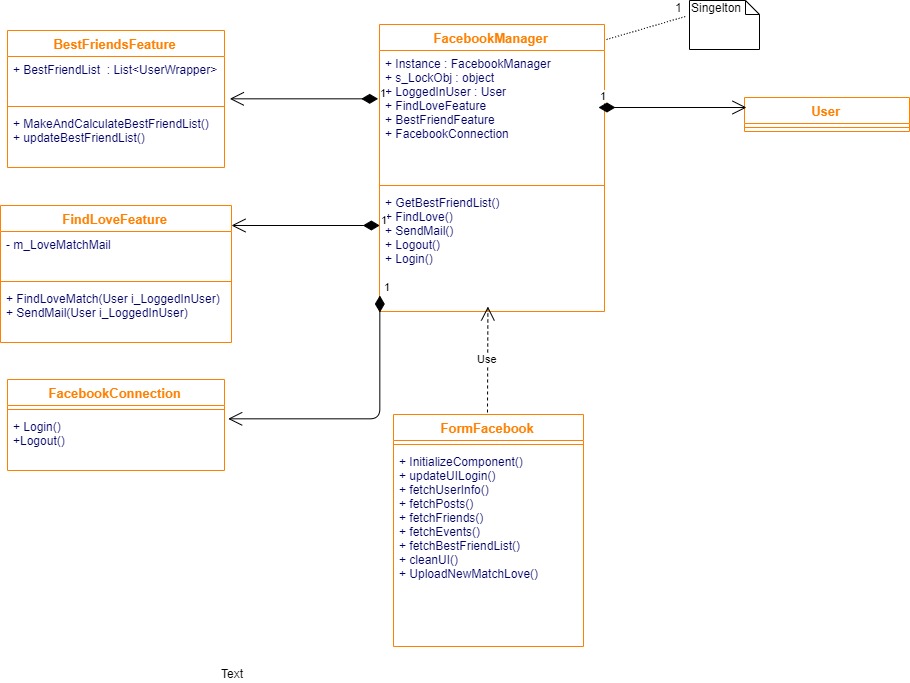
* **אופן המימוש:**

ישנם מספר דרכים לממש סינגלטון, אנחנו בחרנו לממש אותו בעזרת שיטתdouble check lock . בשיטה זו אנו מחזיקים שדה סטטי מטיפוס המחלקה, בנאי שמוגדר כprivate (על מנת שלא נוכל לייצר מופעים נוספים), ופרופרטי סטטי אשר בקריאתו הראשונה , מייצר את המופע פעם אחת בלבד . בdouble check lock אנו מצמצמים את תקורת פעולת הנעילה בעת יצירת האובייקט על ידי שתי בדיקות. הראשונה מתבצעת לפני פעולת הנעילה ותפקידה למנוע נעילה מיותרת בעתיד, שכן פעולת lock היא פעולה יקרה. השניה מתבצעת על מנת לוודא כי לא השתנה מצב המופע בפרק הזמן מהבדיקה הראשונה ועד הנעילה.

* **Sequence Diagram**



* **Class Diagram**



### תבנית מס' 2 – [Facade]

* **סיבת הבחירה / שימוש בתבנית:**

תבנית זו נבחרה במטרה לפשט ל-Client את ביצוע הפעולות השונות,ולהוות חציצה בין ממשק המשתמש (FormFacebook במקרה שלנו) למחלקות הלוגיות.

לדוגמא, בעזרת תבנית זו, כעת הקליינט לא צריך לדעת אם יש לו access token בעת התחברות עם פייסבוק,כלומר ההתחברות שקופה לגמרי עבורו.

אחד היתרונות בתבנית, הוא יכולת ההכמסה של רכיבים לוגים שונים,וכך מתאפשרת תחזוקה עתידית נוחה וקלה יותר.

בנוסף, אם בעתיד, נרצה לבנות מערכת שמצריכה שימוש בשירותים שהוא מציע,נוכל לעשות בו שימוש חוזר בקלות.

* **אופן המימוש:**

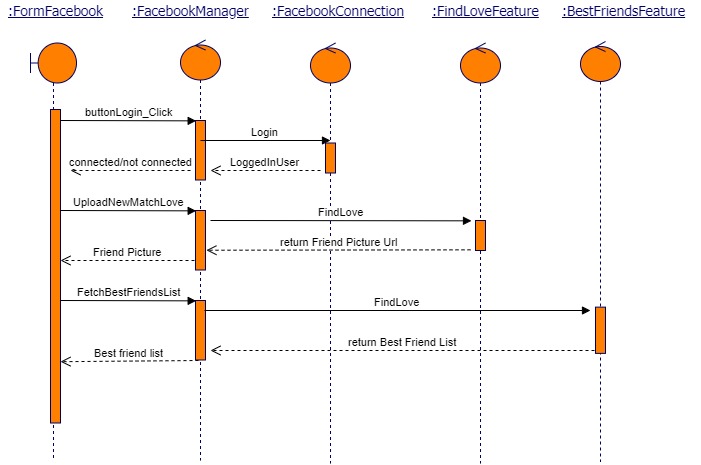
המימוש של התבנית נמצא במחלקה FacebookManager (הפאסאד עצמו).

מחלקה זו מכילה מתודות לוגיות שונות הנוגעות לפעילות האפליקציה, וכן מתודות האחראיות לרכז מספר קריאות לוגיות לשם נוחות שימוש עבור ה-Client (נציין כי מדובר ב-Façade שקוף, שכן באפשרות ה-Client לגשת למחלקות בהם עורך שימוש ה-Façade).

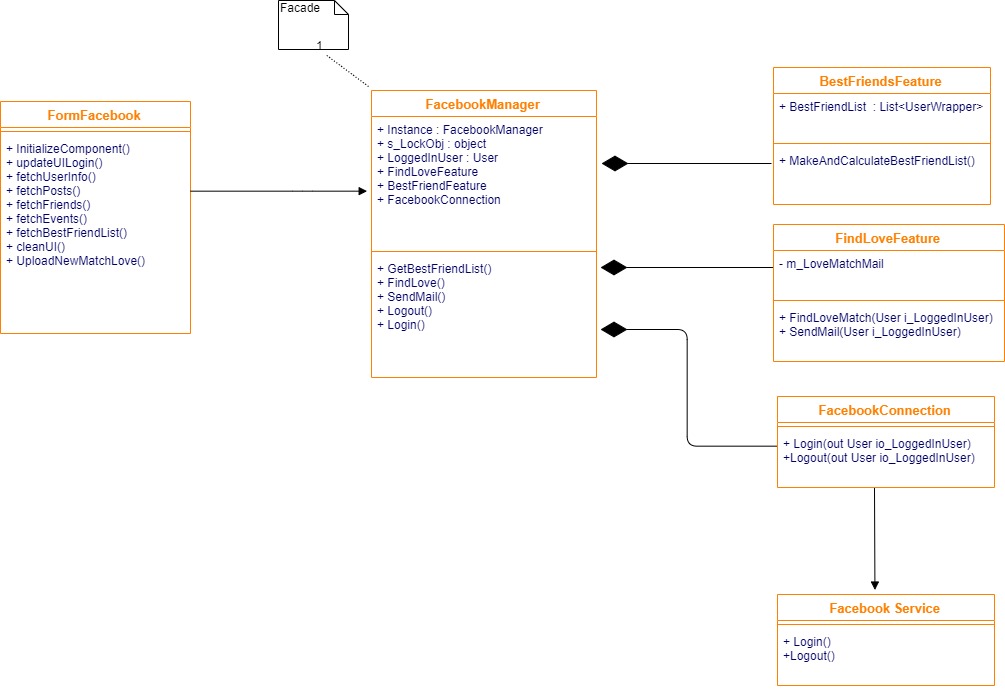
כמו כן הפאסאד "מסתיר" את החיבור עם פייסבוק ושני הפיצ'רים העיקריים במערכת (FindLoveFeature,BestFriendsFeature) וחושף ממשק נוח לשימוש עבור הקליינט,

כלומר חושף עבורו רק מה שהוא צריך ובכך מאפשר שימוש בטוח ונקי.

* **Sequence Diagram**



* **Class Diagram**



### תבנית מס' 3 – [proxy]

* **סיבת הבחירה / שימוש בתבנית:**

הבחירה בתבנית זו נועדה לפתור צורך ברכיב PictureBox משוכלל, שצורתו עגולה.

התבנית רלוונטית לפתרון הבעיה מכיוון שרכיב זה הוא legacy, כלומר לא ניתן לערוך שינויים בקוד שלו, ולכן אנו חייבים לרשת ממנו כדי לייצר בסיס פולימורפי.

פרוקסי זה הואclass proxy - פרוקסי שיורש מהreal subject(PictureBox), ובכך "מתחזה" אליו,מממש את הלוגיקה הנוספת הנדרשת,ואז יוכל לשמש כתחליף,(הפולימורפיזם נובע מההורשה).

מימוש התבנית באופן זה מחייב,מהסיבה שהרכיב בלתי ניתן לשינוי,כאמור.

באופן זה, אם נרצה בעתיד, נוכל להשתמש ברכיב המשודרג באותו אופן שהשתמשנו ברכיב הקודם ללא צורך בשינויים בקליינט.

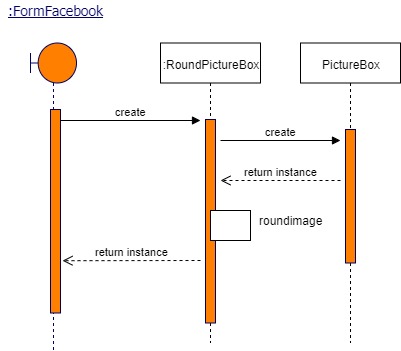
כמו כן, רכיבים עתידיים יוכלו להשתמש בפרוקסי ולבצע בו שימוש חוזר בקלות.

* **אופן המימוש:**

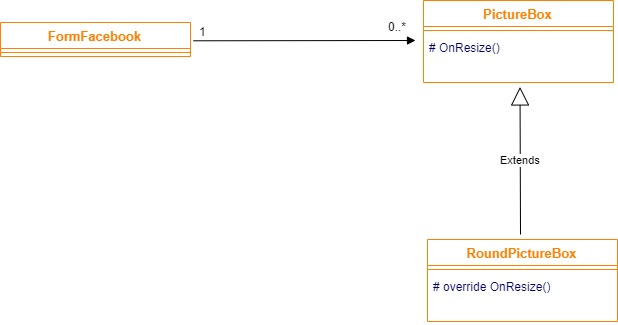
מחלקת RoundPictureBoxProxy היא מחלקת הפרוקסי שלנו, שמתחזה לpictureBox,

ע"י הורשה. הרכיב החדש RoundPictureBoxProxyמוסיף את לוגיקת שינוי הצורה לעגולה,ע"י מימוש מחדש של המתודה OnResize) . כעת, בכל מקום בו נרצה תמונה עגולה באפליקציה נוכל להשתמש בפרוקסי באותו אופן שהשתמשנו ברכיב המקורי ולקבל תמונה עגולה.

* **Sequence Diagram**



* **Class Diagram**



### עבודה אסינכרונית-

***מקום בקוד*:**

FormFacebook.UpdateUILogin()

***תיאור המימוש:***

ישנן מספר שיטות מרכזיות שאחראיות להביא את נתוני המשתמש לאחר התחברותו למערכת,

בהן מימשנו עבודה אסינכרונית:

fetchPosts()

fetchFriends()

fetchEvents()

fetchUserInfo()

***מוטיבציה*:**

בכל קריאת מידע מהיוזר המחובר בפייסבוק, יש גישה לשרתים של פייסבוק, ופעולה זו לוקחת זמן. בכדי שהקליינט יהיה תגובתי בזמן זה, ובכדי ש"עיכוב" זה לא יורגש ע"י המשתמש, נחלק את העבודה ה"מסיבית" למספר תהליכונים נפרדים ,שיבצעו אותה במקביל.

### עבודה עם Data Binding-

***מקום בקוד*:**

fetchPosts()

fetchFriends()

fetchEvents()

***תיאור המימוש:***

כדי לבצע קישור בין פקדי הUI לבין הנתונים הלוגיים, נשתמש באובייקט BindingSource . נגדיר את מקור הנתונים שלו להיות הטיפוס הקונקרטי, וכן נגדיר את מקור הנתונים של הפקד להיות אותו אובייקטBindingSource. כך, למעשה, מתבצע קישור בין אובייקטים אלו.

***מוטיבציה*:**

אם לא נשתמש בData binding, ונרצה לייצג נתון לוגי כלשהוא באמצעות פקד , נצטרך לדאוג (בעצמנו)לעדכן את רכיבי הUI הרלוונטים, אם וכאשר יש שינוי בנתון הלוגי. עקרון הdata binding חוסך טרחה זו, שכן מקשר בין אובייקטים אלו באופן אוטומטי.