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

* **תיאור הפי'צר הראשון- Find My Match:**

הפיצ׳ר מאפשר למשתמש לפלטר לפי מין וטווח גילאים ולראות את חבריו הרווקים שעונים על הקריטריונים שהוזנו.( SelectPreferencesForm)

לאחר שהמשתמש ימלא את הטופס, יפתח חלון חדש (FindMyMatchForm)ובו תוצג רשימת התוצאות מסודרת ב-listBox .המשתמש יכול לבחור את אחד מן ההתאמות ויפתח מסך(MyMatchForm) שיכיל את פרטי האדם שיכול להיות הזיווג עבורו.

במסך זה ניתן לצפות בתמונות, לראות רשימת עמודים אהובים, תמונת פרופיל ואף לפרסם פוסט ולתייג את ההתאמה שנבחרה.

בנוסף המשתמש יכול לצפות בתמונת המשתמש שאהב אותו הכי הרבה(על פי כמות הלייקים) .

וכך לנסות למצוא התאמה.

היכן הפיצר ממומש?

ממשק המשתמש נמצא: **SelectPreferencesForm.**

הלוגיקה נמצאת: **FindMyMatchLogic**

**\*הערה:** המשתמש לא יכול לצפות בתמונת המשתמש שאהב אותו הכי הרבה **ולא ניתן לפרסם פוסטים בגלל הרשאות ה  API.**

* **תיאור הפי'צר השני- Post Statistics:**

הפיצ׳ר מאפשר למשתמש לקבל מידע אודות הפוסטים שלו.( PostRankForm)

המשתמש יכול לצפות בפוסט שקיבל הכי הרבה תגובות כמו כן המשתמש יכול לצפות בפוסטים שלו ממוינים בסדר עולה\יורד לבחירתו לפי כמות התגובות.

בעת לחיצה על הפוסט יפתח מסך(PostInformationForm) שמציג פרטים על הפוסט:

תוכן הפוסט, תאריך כתיבת הפוסט, כמות התגובות ואפשרות לשתף את הפוסט.

כמו כן המשתמש יכול לבחור לצפות ב2 גרפים. גרף אחד מייצג את כמות הפוסטים שהמשתמש העלה ביחס לשנה, גרף שני מייצג את כמות הפוסטים ביחס לחודש.

פיצ'ר זה נותן ערך מוסף למשתמש מפני שבשימוש בפיצ'ר זה המשתמש יכול לעקוב אחרי איזה תוכן גרר יותר תגובות ובכך להגדיל את קהל היעד על ידי פרסום פוסטים דומים.

היכן הפיצר ממומש?

ממשק המשתמש נמצא: **PostRankForm.cs.**

הלוגיקה נמצאת: **PostRankFormLogic.cs**

**\*הערה:** המשתמש לא יכול **לפרסם פוסטים בגלל הרשאות ה  API.  
כמו כן, ישנם פוסטים שלא ניתן לצפות בתוכן שלהם עקב הרשאות הAPI.**

### תבנית מס' 1 – Singleton

* **סיבת הבחירה / שימוש בתבנית:**

סינגלטון נועד למקרים בהם מעוניינים להגביל את יצירת המופעים של [מחלקה](https://he.wikipedia.org/wiki/%D7%9E%D7%97%D7%9C%D7%A7%D7%94_(%D7%AA%D7%9B%D7%A0%D7%95%D7%AA)) מסוימת למופע יחיד.התבנית מאפשרת לרכיבי המערכת השונים לקבל גישה לרכיב שמנוהל על ידי הסינגלטון מבלי ליצור אובייקט נוסף של המחלקה. כמו כן משתמשים בתבנית זו בשני תרחישים מרכזיים:

* אם במערכת יש יותר ממופע יחיד של רכיב ויהיו פערים בתוכנם ייתכן ויווצר מצב שגוי.
* אם במערכת יש יותר ממופע יחיד של רכיב,והמופעים זהים זהו מצב בזבזני.

**אנו משתמשים בתבנית זו** על מנת לפתור מצב בזבזני של יצירת יותר ממופע יחיד של facebookAppManager ובכך לייעל את המערכת. לפני שימוש בתבנית זו היו מספר מופעים זהים של המחלקה facebookAppManager (אשר תפקיד מחלקה זו הוא לשלוף נתונים מהשרת של פייסבוק בעזרת (FacebookApiGraphWrapper. השתמשנו במחלקה זו במחלקות רבות במערכת ולכן יצרנו מצב בזבזני עקב קריאות מרובות לשרת של פייסבוק.

ה- Singelton מאפשר לנו ליצור אובייקט יחיד מסוג facebookAppManager כך שכל רכיבי המערכת השונים יוכלו לגשת אליו ישירות ביוזמתם (במערכת שלנו רק משתמש 1 יכול להיות מחובר ולכן אין צורך במופעים שונים) וכך אנו מונעים את המצב הבזבזני.

* אופן המימוש:

\*אנו ממשים את הקוד **במחלקה: facebookAppManager.**

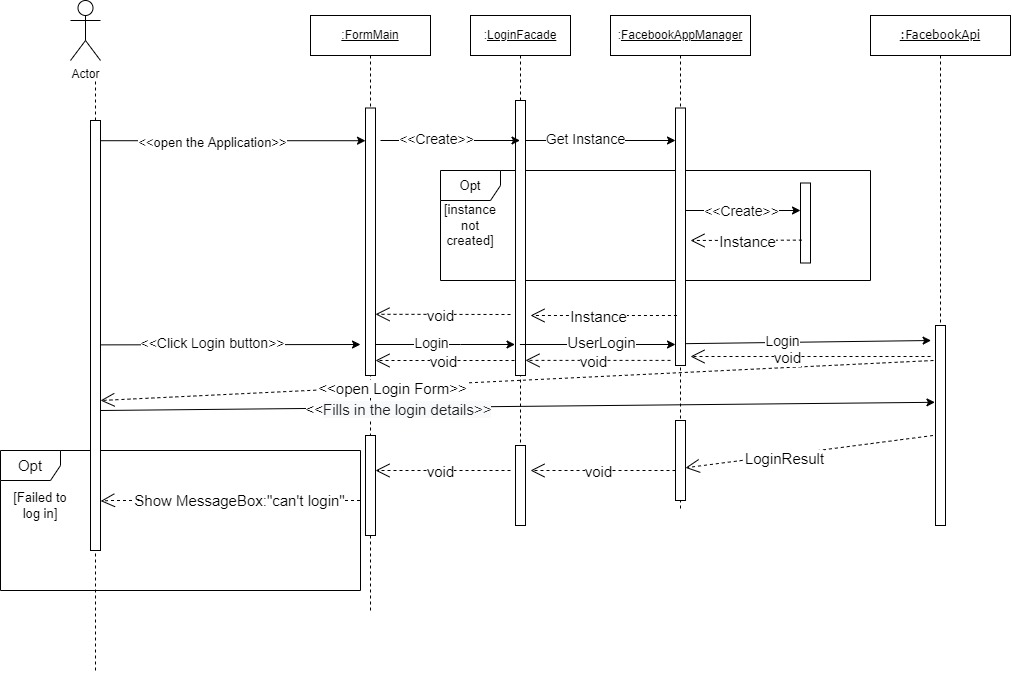
את המחלקה הנ"ל אשר מתקשרת עם FacebookApiGraphWrapper מימשנו באופן הבא:

-הגדרנו את המחלקה כ-Sealed עם Constructor Private על מנת למנוע ירושה.

* המחלקה מחזיקה מופע סטטי של עצמה בשם s\_Instanceאשר מאותחל לnull.

במחלקת קיימת מתודה Get של הInstance אשר כוללת Double Check-Lock שמונעת יצירת מופעים שונים של המחלקה(Thread-Safe). כך שרק בשימוש הראשון במחלקה בתוכנית נוצר מופע זה ובכל שימוש נוסף במחלקה, לא נוצר Instance חדש,אלא מוחזר ה-Instance הקיים.

* כאשר מתרחשת גישה לשרת וישנה בקשה לשלוף מידע ממנו, המידע המבוקש נשמר כPublic Property של הInstance. וכך בכל שימוש חוזר, במקום לפנות שוב לשרת, אנו מחזירים את המידע, אשר נשמר קודם לכן במחלקה של המופע.
* Sequence Diagram

חשוב לציין כי מופע המחלקה הסינגלטונית נוצר בעת בקשת המופע הראשון (במחלקה – (FormMainומסתיים כאשר המשתמש סוגר את האפליקציה.

* Class Diagram

Singleton: FacebookAppManager

Client:  
UserProfileFacade  
PostRankFormLogic  
FindMyMatchLogic  
LoginFacade

### תבנית מס' 2 –Adapter

* **סיבת הבחירה / שימוש בתבנית:**

בחלון FindMyMatchForm לאחר שהמשתמש מזין את הפרמטרים הדרושים למציאת התאמה רצינו להציג לו ברשימת ההתאמות את הפרטים נוספים: שם, גיל, ועיר מגורים, כך שכל נתון יוצג בשורה נפרדת בתוך ה- listBox. רצינו ליצור ממשק שמסוגל לספק ולהביא מידע על הגיל הנוכחי אך הממשק User אפשר לנו לגשת רק לשם,תאריך לידה והמיקום של החבר. כמו כן, אין אפשרות להציג יותר מנתון בודד אודות ה- user ב-listbox שנמצא ב- FindMyMatchForm.

כלומר קבלנו אי-התאמה בין שני Legacy Components (בין User ל-ListBox).

לכן בחרנו להשתמש ב- Adapter המאפשר התאמה בין שני ה- Legacy Components ויציג את הנתונים הנוספים ששייכים לאובייקט user.

בעזרת תבנית זו שילבנו מספר נתונים מתוך האובייקט user והוספנו נתון חדש (גיל).   
כדי שנוכל להציג את הנתונים הנ"ל בListBox המרנו אותם לlist<String> והצגנו אותם ב-ListBox שנמצא ב- FindMyMatchForm.

* אופן המימוש:

יצרנו מחלקה חדשה – MyMatchFormatAdapterכך שבעת יצירת המופע ה-Constractor מקבל רפרנס ל- User.

לאחר שהמשתמש בוחר את הפילטורים שהוא רוצה להתאמה אנו יוצרים מופע של ה-adapter עבור כל התאמה.

MyMatchFormatAdapter מממש את הממשק IMyMatchFormat ,אשר בו נמצאת המתודה CreateFormattedMatchesList. התבנית מממשת את המתודה הזו כך, שמבוצע איחוד של הנתונים שציינו עבור כל התאמה ותכניס אותם לאובייקט מסוג -List<string>, שישלח לMyMatchFacade - ומשם ל-Listbox של FindMyMatchForm.

**הקוד ממומש במחלקות:**

המימוש מתחיל במחלקה - FindMyMatchForm, בה מופעלת המתודה FetchMatches ב-onShown.

לאחר מכן מתודה זו קוראת למתודה GetMyMatchesInfo שנמצאת במחלקה MyMatchFacade , מתודה זו יוצרת מופע של MyMatchFormatAdapter עבור כל אחד מההתאמות של המשתמש, שם היא משתמשת במתודה CreateFormattedMatchesList שנמצאת ב- MyMatchFormatAdapter וב- IMyMatchFormat.

* Sequence Diagram

[

* Class Diagram

[class diagram שמתארת את המחלקות שלכם שמעורבות בתבנית (תיאור מלא שכולל Properties ו- Methods) והיחסים ביניהם (תאור מלא עבור כל יחס כפי שלמדנו בכיתה). עבור כל מחלקה שלכם, כיתבו מי המקבילה שלה (אם יש כזו) בתבנית (Pattern) שבחרתם

### תבנית מס' 3 – Facade

מחלקות ממשק המשתמש בתוכניתנו, תקשרו עם רכיבים לוגים רבים. אם בעתיד נרצה להוסיף רכיבים לוגיים חדשים או לשנות רכיבים לוגיים קיימים, היה נדרש מאיתנו לבצע התאמות ולשנות את ממשק המשתמש, על מנת שיעבוד כנדרש.

התבנית Facade, יצרה הפרדה ברורה בין ממשק המשתמש לבין הלוגיקה אשר מנוהלת על ידיו.

וכך ממשק המשתמש יוכל לגשת רק למידע הנחוץ לו שעליו להציג למשתמש, ואילו הFacade יתקשר עם השכבה הלוגית, כך אנו מבטיחים שהשכבה הלוגית לא מערבת את ממשק המשתמש בתהליכים אשר אין צורך שהוא יכיר בהם. במידה ובעתיד נרצה להוסיף רכיב לוגי חדש, אנו נצטרך לערוך קלות

את ה– Facadeהרלוונטי.

השתמשנו ב Facade עבור מחלקות של ממשק המשתמש בעלי פונקציונאליות דומה, כדי שממשק המשתמש יכול להשתמש במידע הנחוץ לו בצורה נוחה ומותאמת (המידע שהתקבל מהשכבה הלוגית לאחר ביצוע הלוגיקה)

דבר זה יצר סדר ארגון ונוחות עבור כל מחלקה בממשק המשתמש, מפני שכל טופס עובד מול ה- Facade הרלוונטי אליו ולא מול מספר של רכיבים שונים מהשכבה הלוגית.

יתרון נוסף של השימוש ב- Facade הוא הכמסה של הרכיבים הלוגיים וקורא להפעלתם באופן מסודר תחת אותו רכיב.

* אופן המימוש:

יצרנו Facades עבור הטפסים בצורה הבאה:

FriendFacade- FriendProfileForm, MyMatchForm

LoginFacade-FormMain

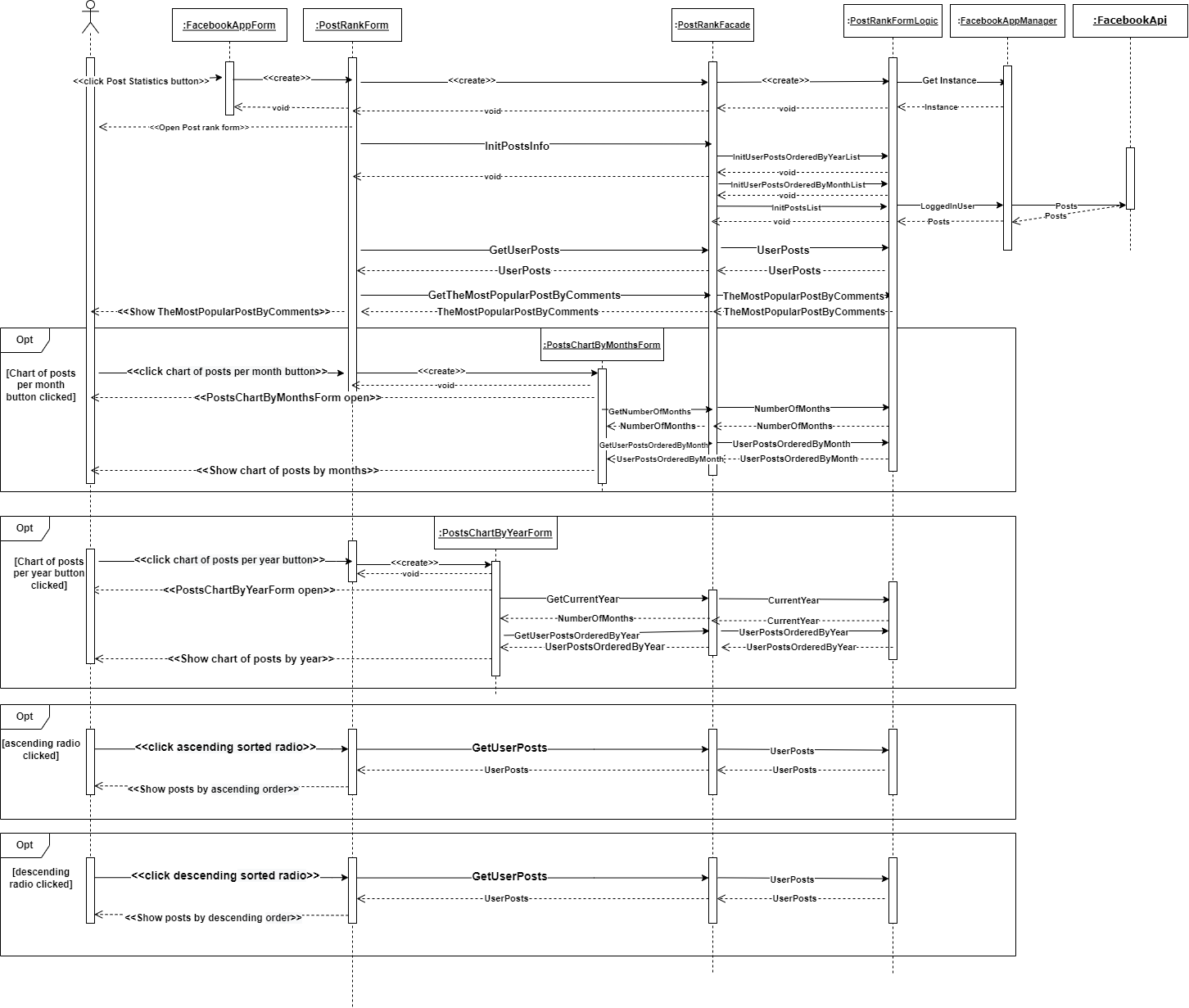
MyMatchFacade- FindMyMatchForm, SelectPreferencesForm

PostRankFacade- PostRankForm, PostsChartByMonthsForm, PostsChartByYearForm

UserProfileFacade- FacebookAppForm, MyMatchForm, PostInformationForm, UserProfileForm

ה-Facades מכילים את הרכיבים הלוגים הבאים:

* FriendFacade מתקשר עם המחלקה הלוגית - FriendLogic
* LoginFacade מתקשר עם המחלקה הלוגית – FacebookAppManager
* UserProfileFacade מתקשר עם המחלקה הלוגית – FacebookAppManager
* MyMatchFacade מתקשר עם המחלקה הלוגית- FindMyMatchLogic
* PostRankFacade מתקשר עם המחלקה הלוגית- PostRankFormLogic
* Sequence Diagram



* Class Diagram

[class diagram שמתארת את המחלקות שלכם שמעורבות בתבנית (תיאור מלא שכולל Properties ו- Methods) והיחסים ביניהם (תאור מלא עבור כל יחס כפי שלמדנו בכיתה). עבור כל מחלקה שלכם, כיתבו מי המקבילה שלה (אם יש כזו) בתבנית (Pattern) שבחרתם

### עבודה אסינכרונית

השתמשנו בתכנות אסינכרוני עבור פעולות ששולפות נתונים מהשרת של Facebook בטפסים המצויינים למטה, זאת מאחר שפעולות אלו לוקחות זמן רב ופוגעות בחווית המשתמש אשר אין באפשרותו לסגור את החלון המוצג ,להזיז את החלון וכו'.  
לכן, גרמנו לכך שהאפליקציה תרוץ בmain thread ופעולות שניגשות לשרת יבוצעו בthread נפרד.בצורה זו האפליקציה עובדת בצורה רציפה יותר וחווית המשתמש שופרה.

היכן בקוד נעשה שימוש בעבודה אסיכרונית:

**1 ) במחלקה MyMatchForm**

כאשר הטופס נפתח אנו מבקשים מהשרת את הנתונים של ההתאמה : שם המשתמש, מין, תמונת פרופיל, עמודים אהובים,תמונות, מיקום וכו'.  
במתודה onShown שנקראת לאחר פתיחת החלון נוצר Thread נפרד עבור המתודה fetchMatchInfo אשר טוענת את נתוני ההתאמה. בתוך המתודה , עבור הפונקציות fetchLikedPages ו- fetchPictures אשר לוקחות זמן רב יצרנו תתי טריידים נוספים אשר קורים במקביל ולכן מזרזים את שליפת המידע מהשרת ונותנים תחושה של של שימוש רציף באפליקציה.

**2) במחלקה UserProfileForm**

במתודה onShown שנקראת לאחר פתיחת חלון פרופיל המשתמש יצרנו Thread נפרד עבור המתודה fetchInfo אשר טוענת את נתוני המשתמש. על מנת לייעל את המערכת יצרנו תתי טרידים עבור הפונקציות : fetchNewsFeed, fetchFriends fetchUpcomingBirthdays, fetchAlbums. שכן פעולות אלו לוקחות זמן ויצירת Thread עבור כל אחד מהן מזרז את שליפת הנתונים.

**3) במחלקה FindMyMatchForm**

במתודה onShown שנקראת לאחר פתיחת החלון יצרנו שני טרידים עבור כל אחת מהפונקציות הבאות :  
fetchFan, fetchMatches. פעולות אלו לוקחות זמן רב ויצירת thread עבור כל אחת מהן מאפשרת עבודה במקביל וכך מייעלת את שליפת הנתונים מהשרת ונותנת למשתמש תחושה של שימוש רציף באפליקציה.

עבודה עם Data Binding