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

* **Nearest Friends**- בפיצ'ר זה המשתמש בוחר רדיוס בו הוא מעוניין לחפש חברים מהרשימת חברים שלו.
* **Friend Analyzer**- פיצ'ר זה מחזיר את ממוצע הנתונים של הפעילות של המשתמש ושל חבר רנדומלי מהרשימת חברים שלו.
* **Match My Friends**- כאן מוגרלים שני חברים מרשימת החברים של המשתמש(גבר ואישה שנמצאים או מעל גיל 18 או ששניהם מתחת לגיל 18). המשתמש יכול לדרג אותם לפי איכות ההתאמה שהוא מרגיש שיש ביניהם ושומר את פרטי ההתאמה בlistbox.

### תבנית מס' 1 –Factory Method

**סיבת הבחירה / שימוש בתבנית:**

בתרגיל זה הרחבנו את אפליקציית הפיייסבוק שלנו כך שהיא תומכת במספר רב יותר של פיצ'רים, אשר פולימורפיים אחד לשני ומממשים interface משותף.

בשל כך, על מנת לספק ממשק פשוט ליצירת אובייקט קונקרטי שלא ימומש בclient, ראינו לנכון להשתמש בתבנית FactoryMethod.

שכן, תבנית זו מקלה על תחזוקת המערכת בעתיד, ומכמיסה את לוגיקת יצירת האובייקט הקונקרטי כך שהclient משתמש בצורה זהה בכלל האובייקטים במשפחה הפולימורפית.

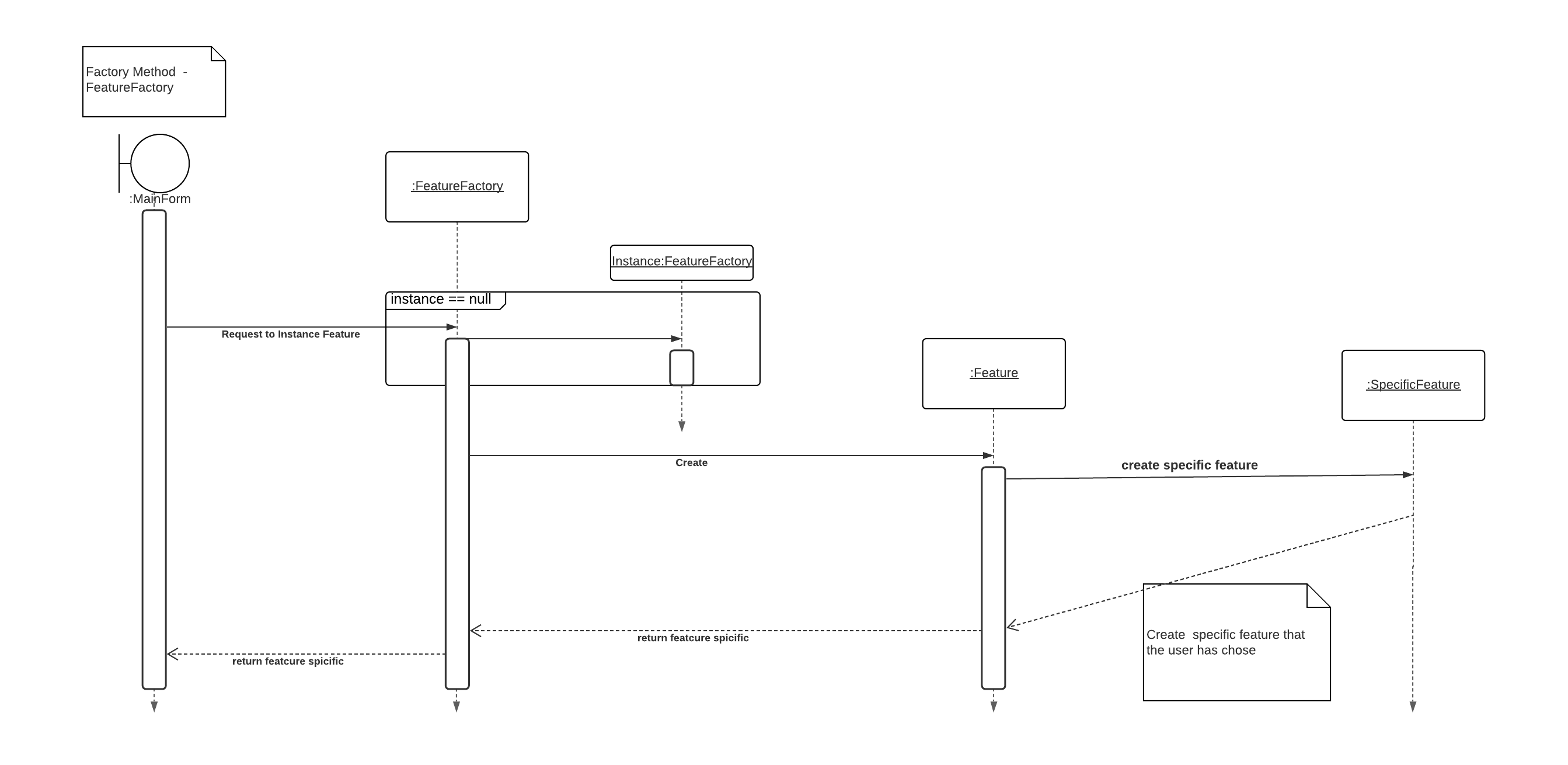
**אופן המימוש:**

יצרנו אב פולימורפי FeatureFactory למחלקה Feature שמכיל בתוכו את המתודה Create() שמחזירה טיפוס מסוג Feature. בכך מנענו את אפשרות יצירת הפיצ'רים בצורה ישירה בMainForm, זאת בכדי למנוע גישה לכך ממקומות שהם לא חלק מהלוגיקה של האפליקציה. מימשנו את FeatureFactory כסינגלטון מתוך רצון להשתמש רק instance שלו ולא לייצר אפשרות ליצור אובייקטים חדשים שלו מאחר ואין צורך בכך.

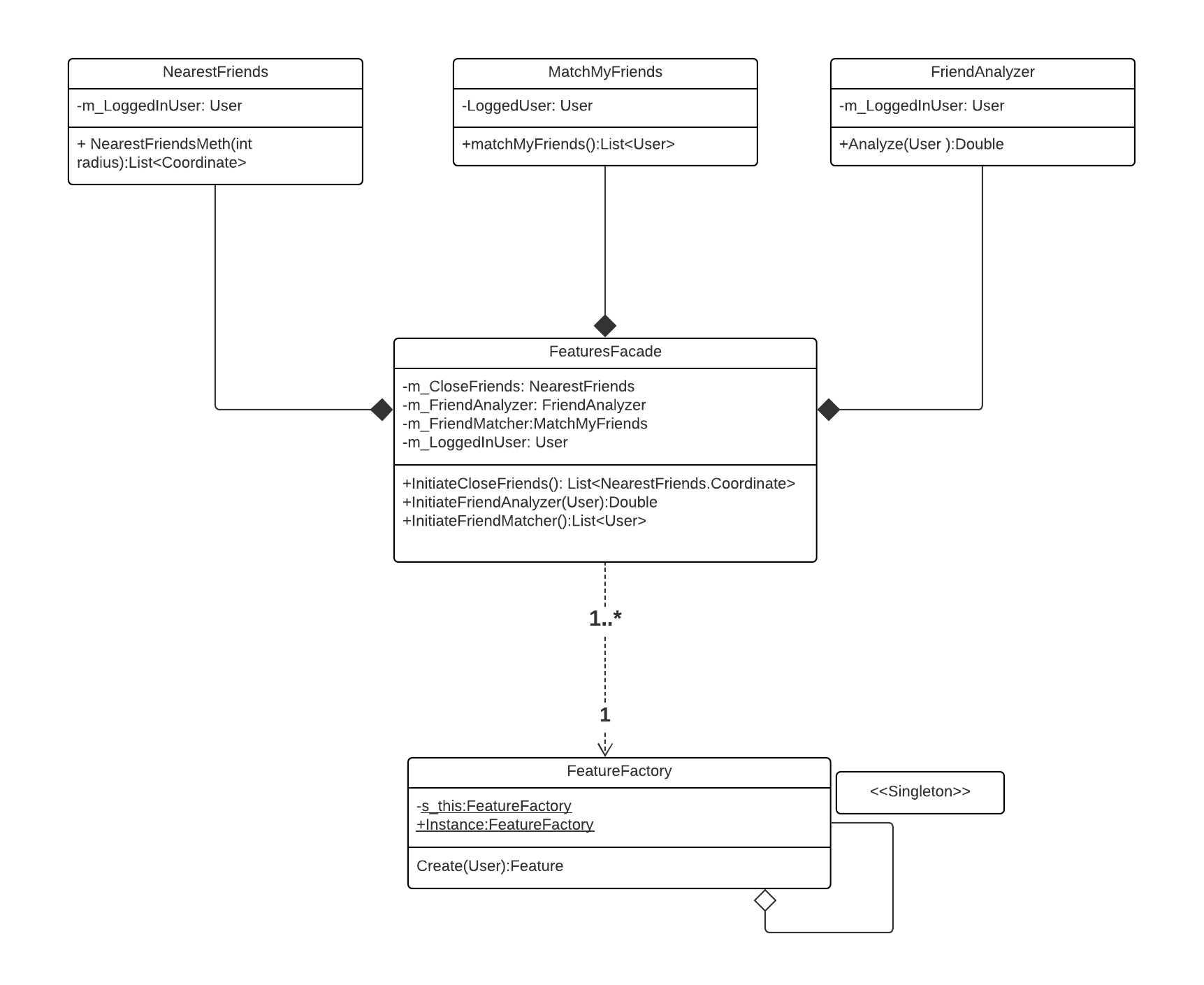
את מחלקות ה FactoryMethod ניתן למצוא בתיקיית FactoryMethod.

גישה מהקוד היא דרך הMainForm

* **Sequence Diagram**



* **Class Diagram**



### תבנית מס' 2 –Façade

### סיבת הבחירה / שימוש בתבנית:

בחרנו בתבנית זו מכיוון שהרחבנו את אפליקציית הפייסבוקית שלנו ורצינו למנוע גישה מה Client ללוגיקה באופן ישיר, זאת כדי לשמור על הפרדה בין הלוגיקה לממשק המשתמש וליצור ביניהם מתווך.

בחרנו בשימוש בתבנית הפאסד בצורה "שקופה" כדי לעזור לנו בתחזוק עתידי של המערכת והסתרה מן המשתמש.

**אופן המימוש:**

הגדרנו מחלקת MainFacade .

ב MainFacade יישמנו את לוגיקת ההתחברות ושליפת נתוני המשתמש מפייסבוק. מחלקת MainFacade מחזיקה מופעים מסוג: LoginResult ,UserProxy.

הclient ניגש למחלקה זו באמצעות המופע היחיד(instance) . בנוסף בחרנו לממש את MainFacade כמחלקה סינגלטונית (singleton) וזאת מכיוון שרצינו שהיא תהווה אובייקט יחיד ושלא יהיו אפשרויות ליצור עוד מופעים שלה.

מימשנו עוד תבנית facade ששמה FeatureFacade גם היא מומשה בצורה שקופה.

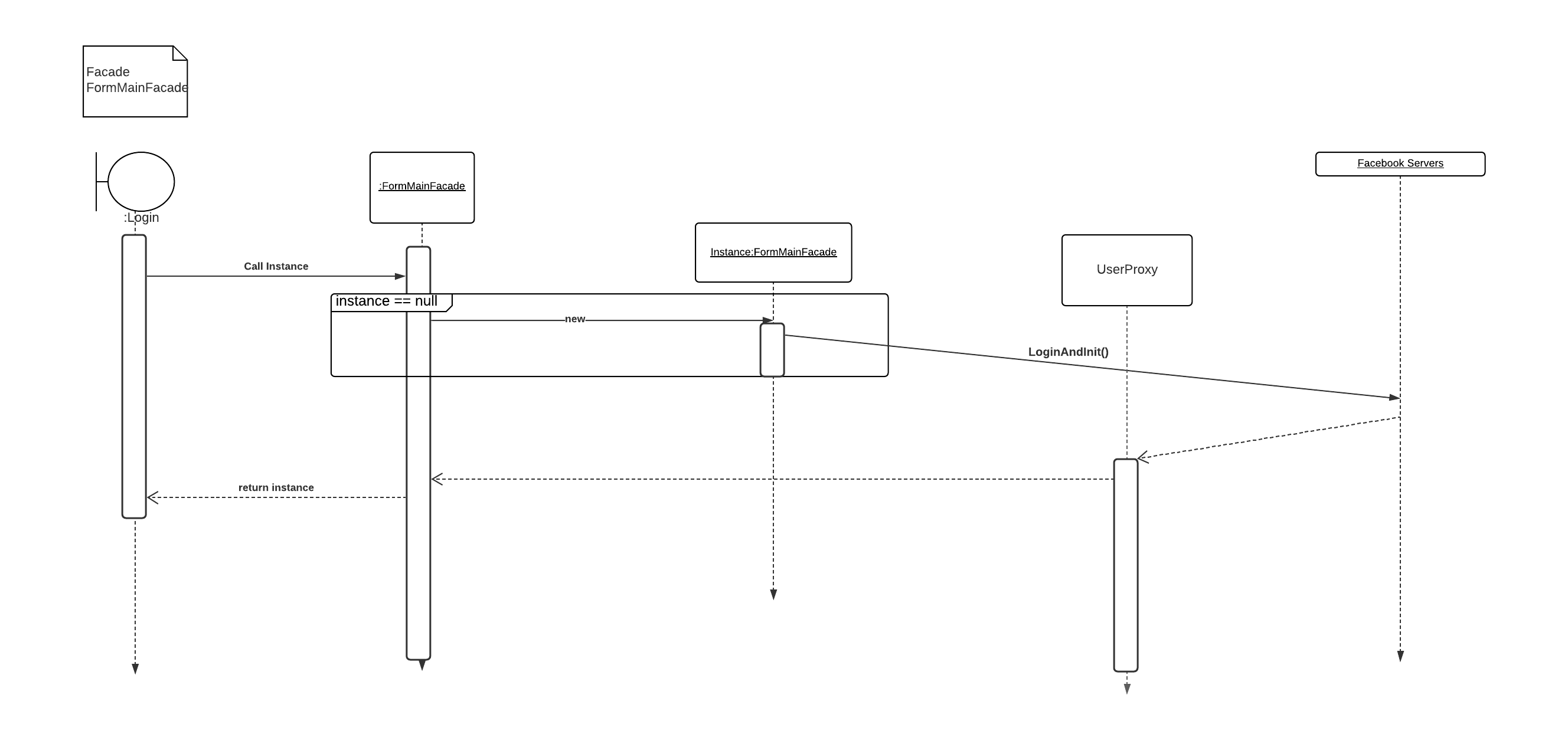
מחלקת ה FeatureFacade מחזיקה מופעים מסוג: NearestFriends, MatchMyFriends ו-FriendAnalyzer. ובנוסף גם אובייקט מסוג User.

מטרת המחלקה הזו היא לאפשר גישה מבוקרת לפיצ'רים הקיימים במערכת דרך סדר פעולות מוגדר.

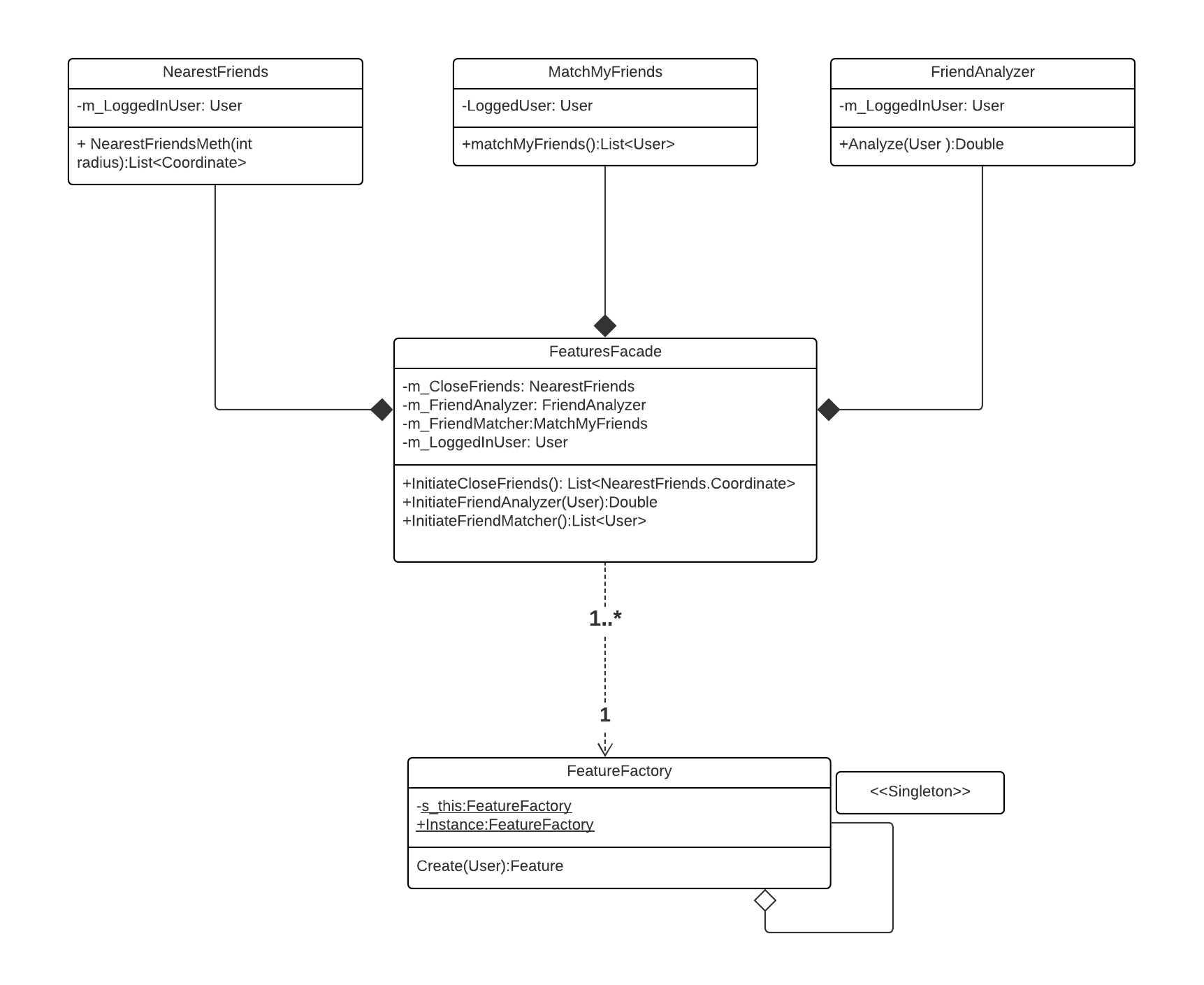
את מחלקות ה Facade ניתן למצוא בתיקיית Facade.

גישה מהקוד היא דרך FeatureFactory.

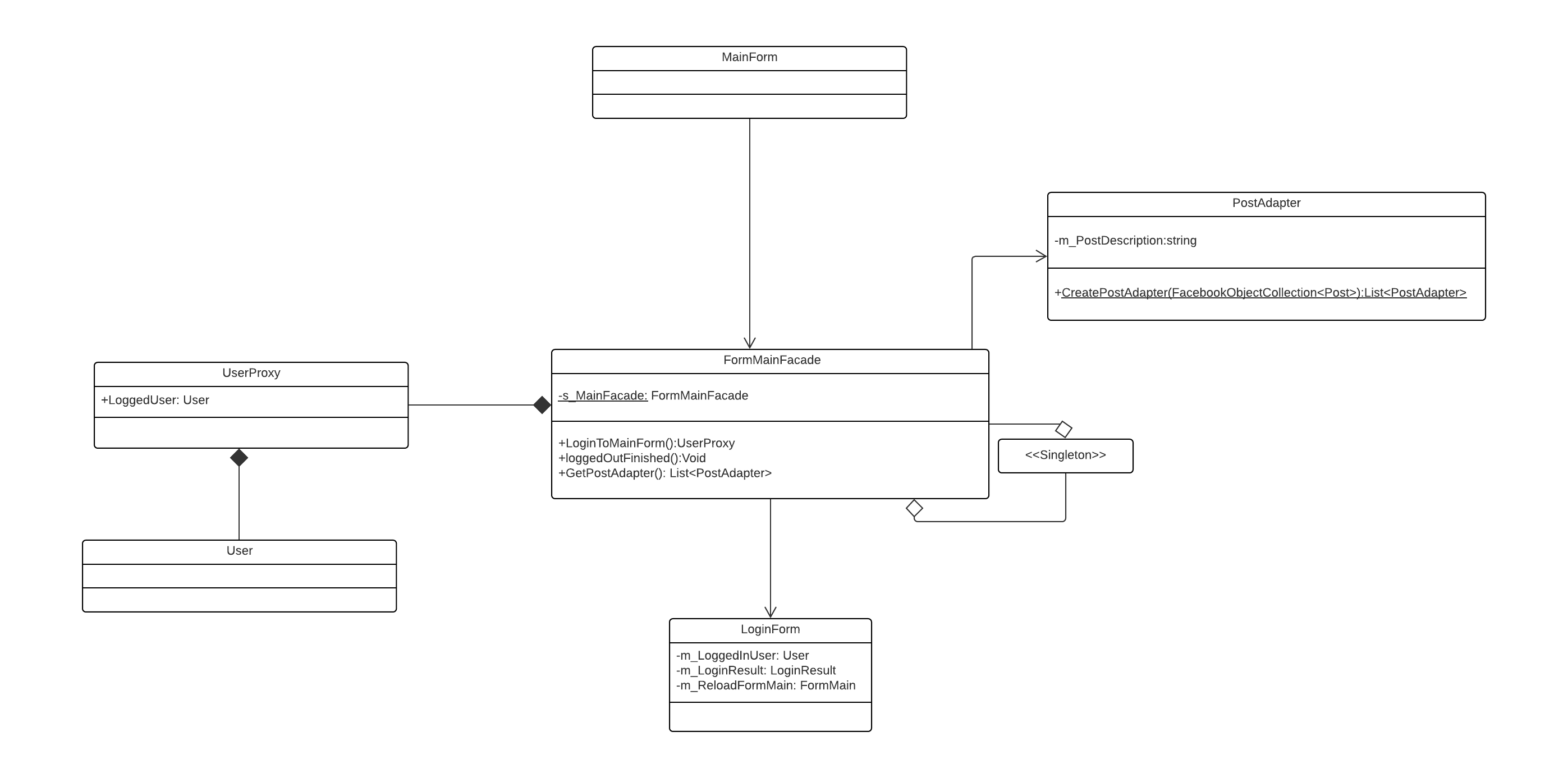
* **Sequence Diagram**



* **FeatureFacade**



* **Class Diagram MainFacade**



### תבנית מס' 3 – Adapter

**סיבת הבחירה / שימוש בתבנית:**

בחרנו בתבנית זו מאחר ורצינו להעניק למשתמש יכולת לערוך רכיב שלא ניתן לעריכה ועשינו זאת באמצעות מחלקת PostAdapter.

**אופן המימוש:**

כפי שציינו בסיבת הבחירה בתבנית, הרכיב post לא מספק יכולת שינוי ועריכה עבור תוכן הפוסט ולכן הגדרנו מחלקה PostAdapter שתפקידה להתחזות לרכיב המקורי, היא מחזיקה post בקומפוזיציה ובנוסף מספקת שירות של set במידה והמשתמש מחליט לערוך פוסט קיים.

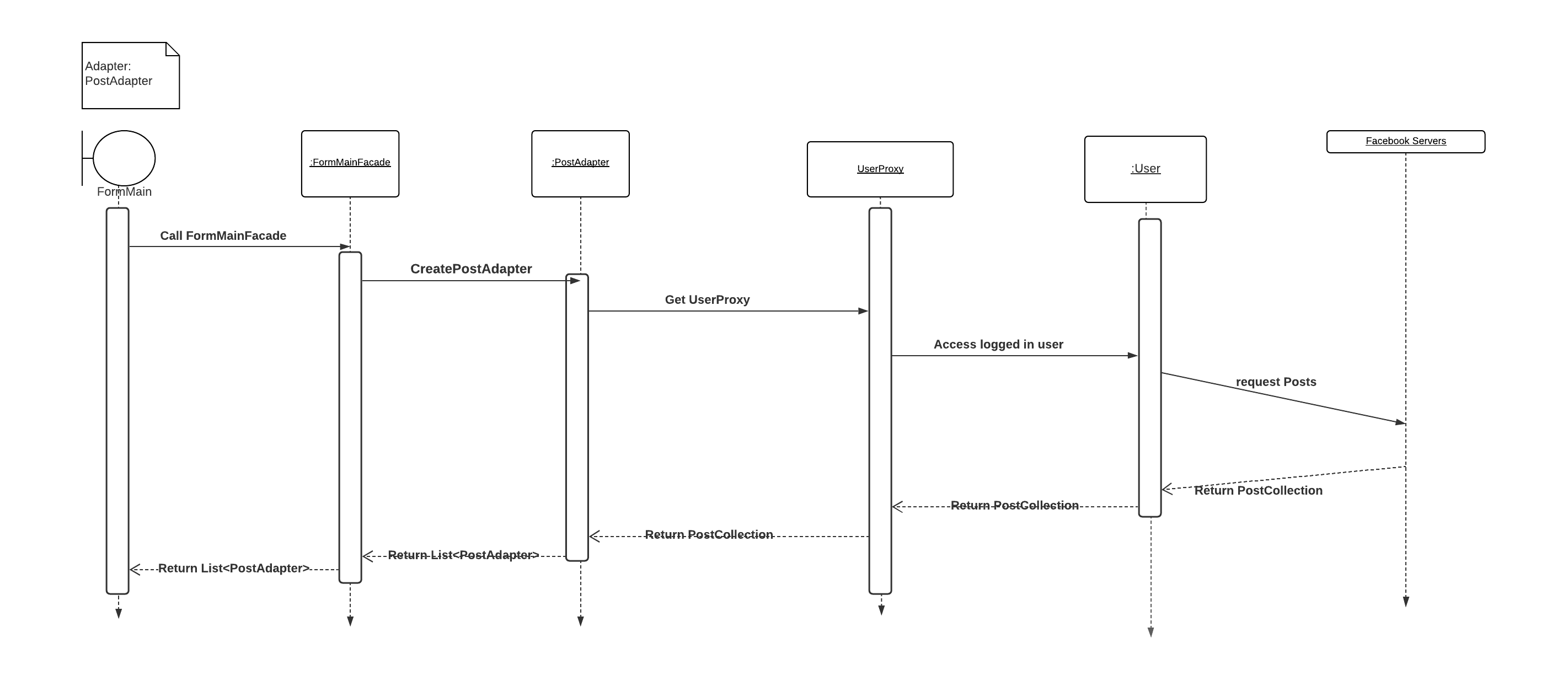
המחלקה PostAdapter מממשת מתודתCreatePostsAdapter() המקבלת רשימה של posts ודואגת להחזיק את הרשימה כרשימה של PostAdapter.

את מתודה זו מפעיל הclient MainForm)) כאשר הוא מעוניין לגשת לרשימת הפוסטים של המשתמש.

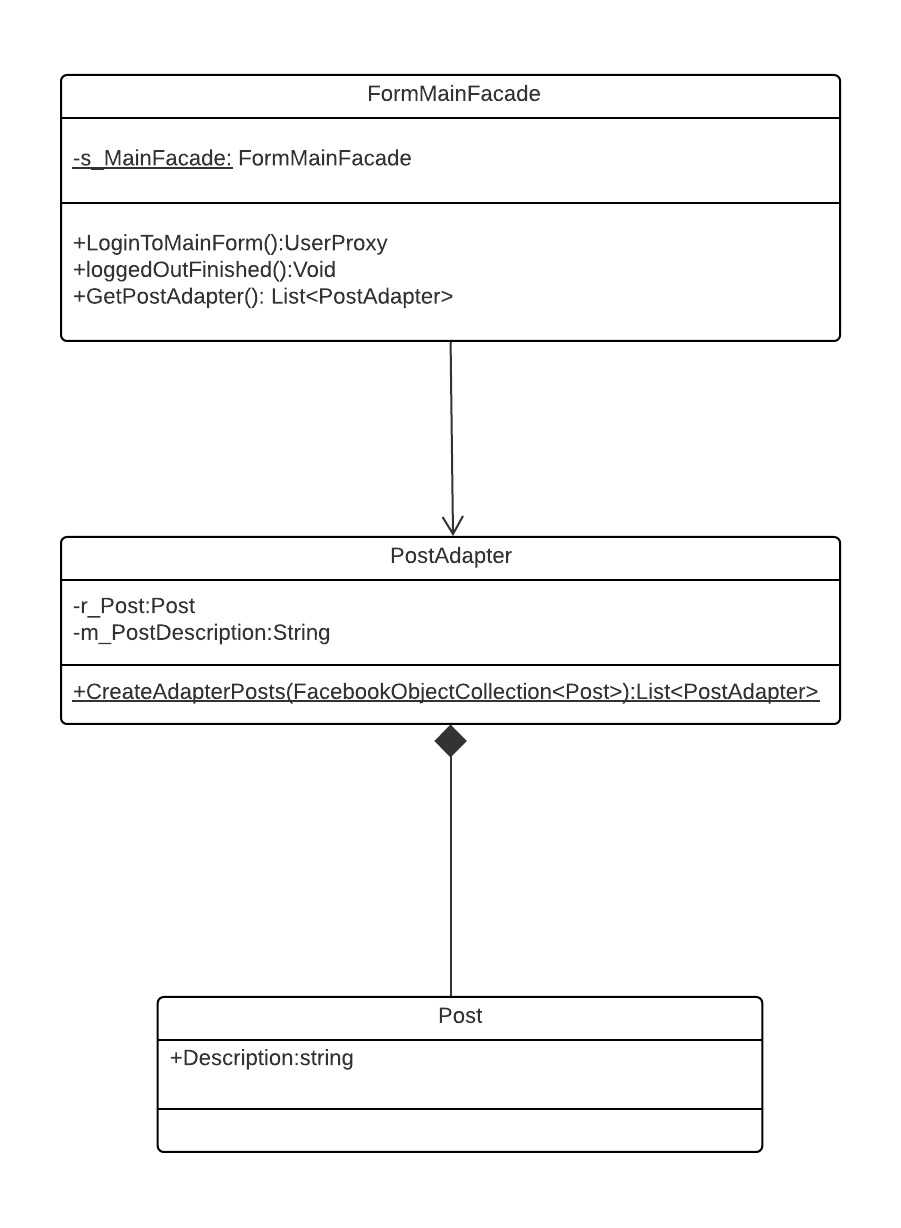
ניתן למצוא בתיקיית הAdapter.

הגישה בקוד היא דרך FormMainFacade

* **Sequence Diagram**



* **Class Diagram**



### שימוש בData Binding

* השתמשנו ב2 Way data binding בפרטי המשתמש.

One way בטעינת פרטי המשתמש דרך UserBindingSource, וtwo way בשינוי פרטי המשתמש דרך כפתור Edit.

שינוי פרטי המשתמש נעשה דרך מעבר בUserProxy המאפשר שירותי עריכה ומחזיק בתוכו אובייקט מסוג User.

* גם בטעינת הפוסטים ובשינוי שלהם השתמשנו בtwo way data binding, לטובת שינוי הפוסטים עבדנו עם PostAdapter כפי שתואר למעלה (postAdapterBindingSource).

### עבודה אסינכרונית

* מימשנו את העבודה האסינכרונית באמצעות כך שחילקנו את טעינת פרטי המשתמש userBindingSource לthread אחד שנקרא userDetailsThread. ואת טעינת הפעילות של הloggedInUser לthread שני שנקרא fetchersThread. אלו שני הthreads העיקריים.
* FetchersThread נתנו priority גבוה יותר משל userDetailsThread בשל המשימה שעליו לעשות והמשקל שלה.
* בתוך fetchersThread חילקנו את טעינת הפוסטים,אלבומים,חברים וצ'קאינס לתתי threads.
* הוספנו עוד thread בכניסה לtab של הפיצ'רים בכדי שיטען פעולה כבדה של אחד הפיצ'רים מבעוד מועד.
* שאר האפליקציה רצה על הmainThread.