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

* מציאת שידוך פוטנציאלי: (Find Match Feature)  
  הפיצ׳ר מאפשר למצוא התאמות עבור דייטים פוטנציאליים בשביל המשתמש על בסיס העדפותיו, כגון: מגדר וגיל.

בנוסף, הדייטים הפוטנציאליים ממויינים בסדר יורד על בסיס כמות דפים אהובים משותפים שיש בין המשתמש והדייט הפוטנציאלי.

* מבט כללי על חבר נבחר: (Friend OverView Feature)  
  הפיצ׳ר מאפשר לבחור חבר מרשימת החברים ולקבל את כמות האינטרקציות שלו עם הפוסטים של המשתמש, כגון: מספר לייקים והתגובות על פוסטים.  
  בנוסף, הפיצ'ר מראה את המשותף בין החבר למשתמש, כגון: רשימת החברים המשותפים עם המשתמש, דפים שהמשתמש והחבר אהבו, ספורט אהוב משותף ושפות משותפות.

הערה: כל פיצ'ר ממומש בטופס נפרד, אל הטפסים הללו ניתן להגיע דרך כפתורים ייעודיים בטופס הראשי באפליקציה.

### תבנית מס' 1 – Singleton

* סיבת הבחירה / שימוש בתבנית:

באפליקציה שלנו מימשנו מחלקה בשם UserManager, תפקידה הוא לנהל את המשתמש במערכת שלנו אל מול שרתי פייסבוק. המחלקה הזו מאפשרת פעולות התחברות והתנתקות מפייסבוק ובנוסף מחזיקה מידע מפייסבוק אודות המשתמש (ה-User) המחובר כעת למערכת.

מחלקות שונות במערכת עושות שימוש במחלקה UserManager, כדי לבצע התחברות או התנתקות, אך בעיקר כדי לשלוף נתונים על המשתמש המחובר למערכת.

כחלק מאפיון האפליקציה קיים משתמש אחד בלבד המחובר למערכת, לכן לא רצינו לאפשר יצירה של יותר ממופע יחיד של המחלקה (אם יהיה יותר ממופע 1 זה לא תקין). בנוסף רצינו לאפשר לרכיבים שונים במערכת כולל רכיבים עתידיים, לעשות שימוש במופע היחיד של המחלקה ביוזמתם.

התבנית Singleton Pattern עונה לצרכינו מהאפיון, לכן בחרנו לממש את המחלקה UserManager כמחלקה סינגלטונית.

* אופן המימוש:

בחרנו לממש באופן המלא של סינגלטון, על ידי המימוש הבא:

הסתרה של בנאי המחלקה, כלומר: private constructor כדי שלא יהיה ניתן לייצר מופעים מחוץ למחלקה.

הגישה למופע של המחלקה נעשה באמצעות private static רפרנס למופע היחיד של המחלקה, שנקרא s\_Instance.

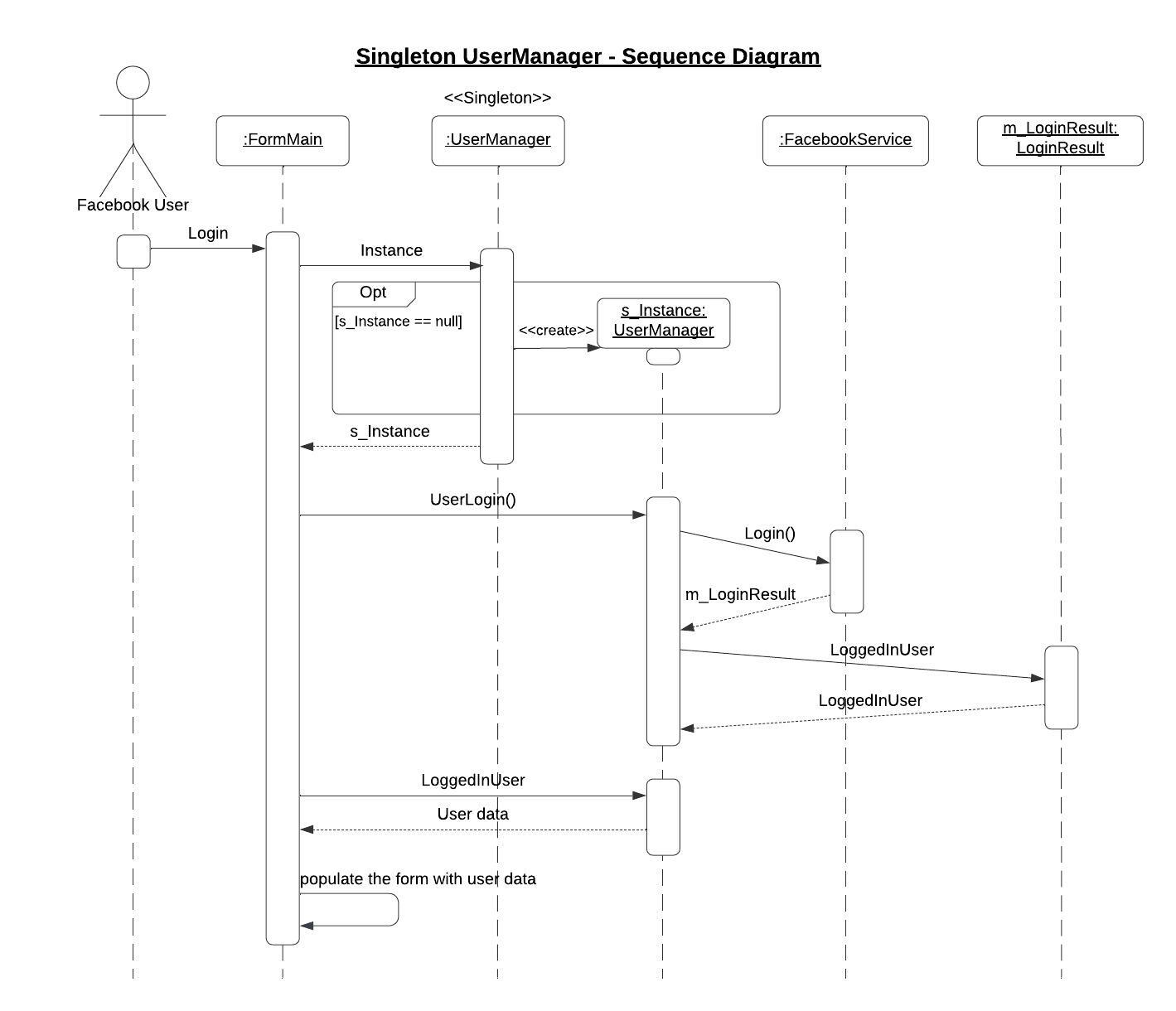
הגדרת המחלקה כ- sealed כדי לעזור בהתלבטות שלא יהיה ניתן לרשת מהמחלקה.

המימוש הוא Thread safe על ידי שימוש במנעול שעוטף את שלב יצירת האובייקט.

שימוש ב- Double check lock שהוא פטרנס יישומי כדי לחסוך פעולה יקרה של נעילה.

ניתן למצוא את המימוש בקוד תחת התיקייה Singleton.

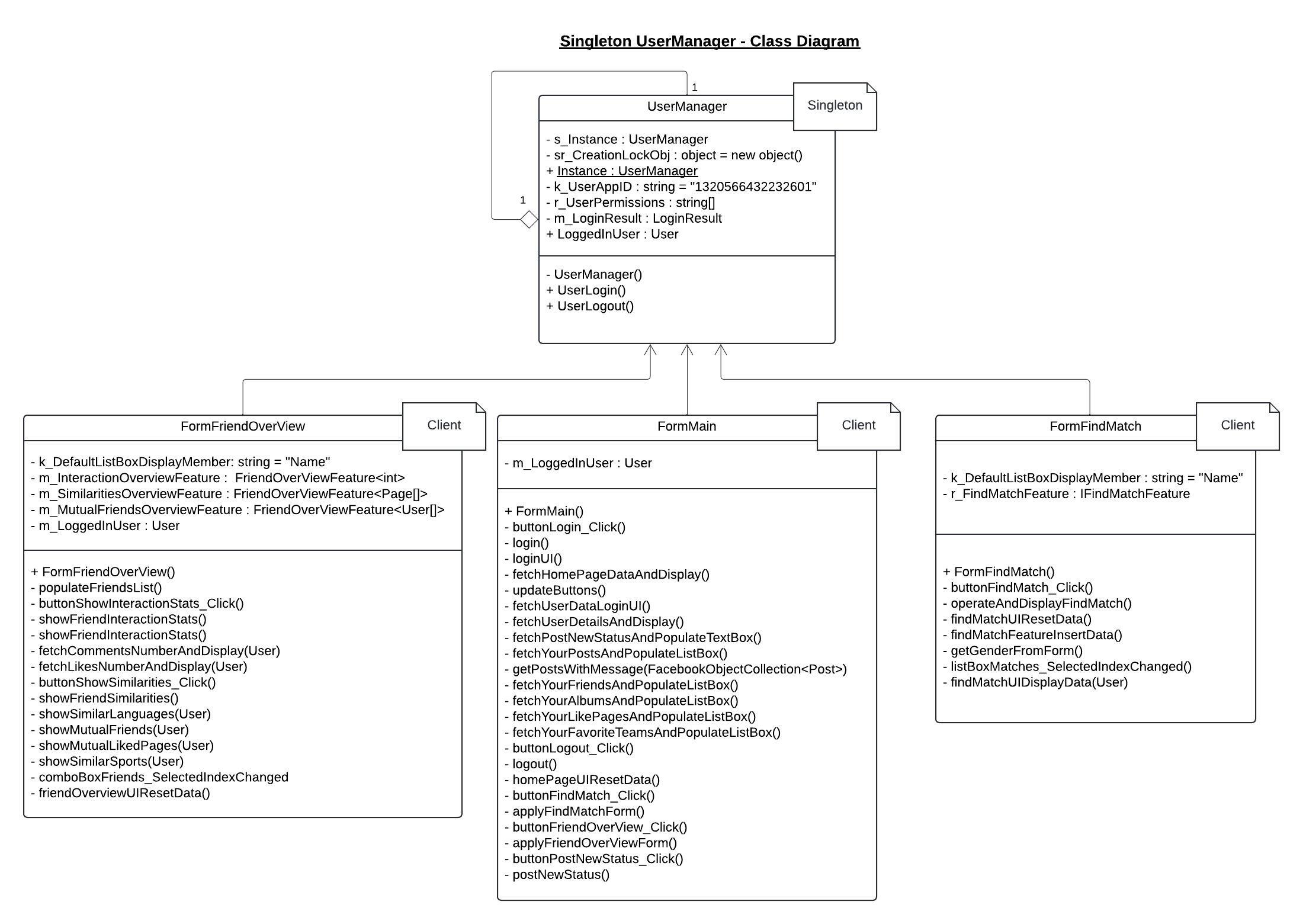
* Sequence Diagram

של האינטרקציה המורכבת ביותר.

* Class Diagram

**Singleton:** UserManager

**Clients:** FormMain, FormFindMatch, FormFriendOverView



### תבנית מס' 2 – Factory Method

* סיבת הבחירה / שימוש בתבנית:

באפליקציה שלנו מימשנו 3 טפסים שונים:

1. FormMain – הטופס הראשי של האפליקציה, המספק למשתמש חווית פייסבוק בסיסית.

2. FormFindMatch – טופס שבו המשתמש יכול למצוא דייט פוטנציאלי על בסיס העדפות מתוך רשימת החברים שלו, באמצעות הפיצ'ר למציאת שידוך פוטנציאלי.

3. FormFriendOverView – טופס שבו המשתמש יכול לבחור חבר מרשימת החברים שלו ולראות אינטרקציות של החבר על הפוסטים שלו ומאפיינים משותפים ביניהם, באמצעות הפיצ'ר למבט כללי על חבר נבחר.

המחלקות המממשות את שלושת הטפסים יורשות ממחלקת Form ולכן מהוות משפחה פולימורפית, בנוסף במהלך השימוש באפליקציה נרצה לבחור וליצור אובייקטים מהמשפחה. לכן מהאפיון נובע שנרצה להשתמש בדיזיין פטרן Factory Method כדי לבצע את מלאכת הבחירה והיצירה של אובייקטים מהמשפחה במודל ולא ב-Client.  
באופן זה, הוצאנו את מלאכת הבחירה והיצירה של הטפסים למקום אחד בקוד במודל, במקום שיהיה מפוזר ברחבי הקלייט. כך נמנענו משכפול קוד, על ידי מטודה ריוזבילית וכך ניתן לתחזוקה בקלות במקום יחיד במודל.

אם בעתיד נרצה להוסיף טופס חדש לאפליקציה או לשנות טופס קיים לטופס אחר, נצטרך רק לבצע שינויים קלים במחלקת המפעל לייצור טפסים ה- Factory שאחראית על בחירת וייצור הטפסים.

* אופן המימוש:

בחרנו לממש בתצורת Static Factory Class שהיא תצורה reality driven, אקסטנסבילית ותחזוקתית, לכן תצורה טובה מבחינה הנדסית.

יצרנו מחלקה סטטית בשם FormsFactory, שאחראית על מלאכת הבחירה והיצירה של הטפסים השונים.

במחלקה מימשנו מטודה סטטית CreateForm, המקבלת כפרמטר enum מסוג eFormType שהגדרנו בתוך המחלקה ומכיל את סוגי הטפסים השונים אותם ניתן לייצר.

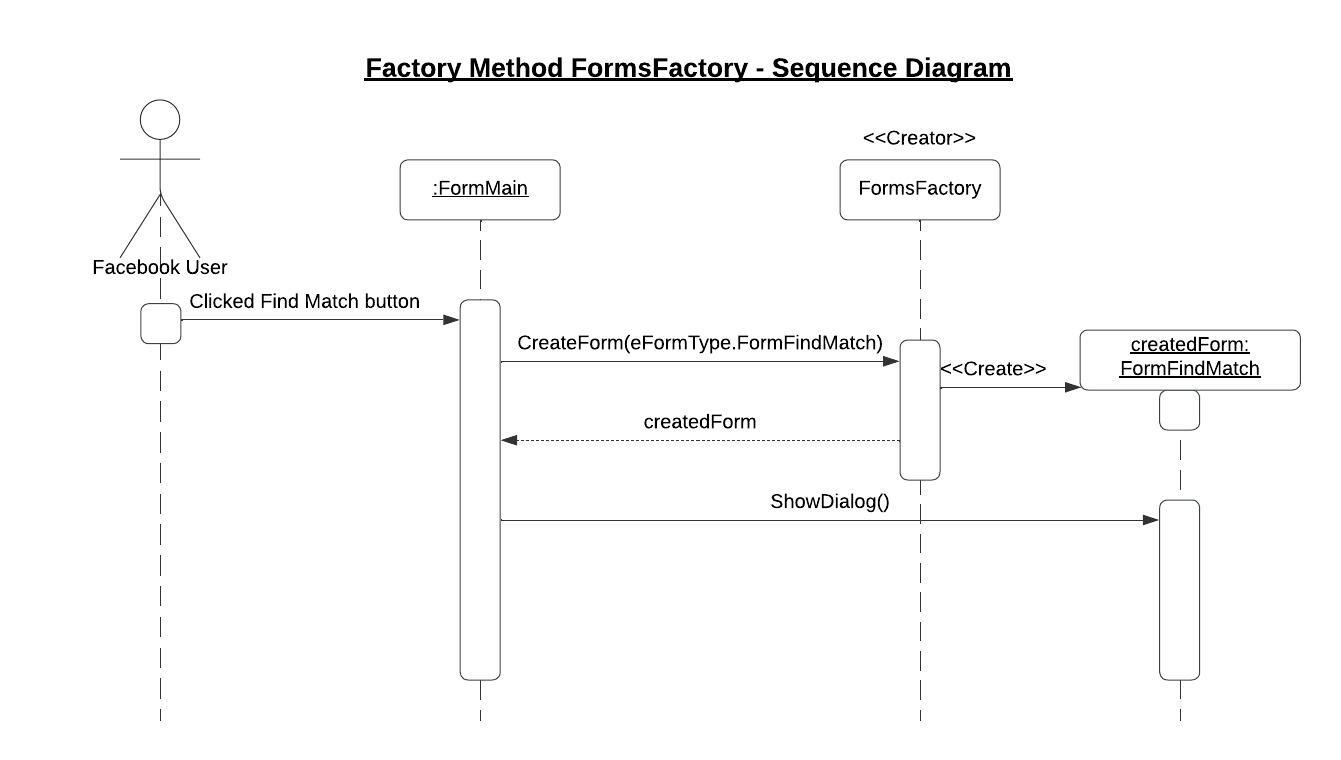
המטודה יוצרת את הטופס המתאים לפי ה-enum ומחזירה רפרנס אליו כמחלקת האב Form.

ה- Client משתמש במחלקה FormsFactory וספציפית במטודה CreateForm בכדי לייצר את הטופס הרצוי בעזרת ה-enum.

ניתן למצוא את המימוש בקוד תחת התיקייה Factory.

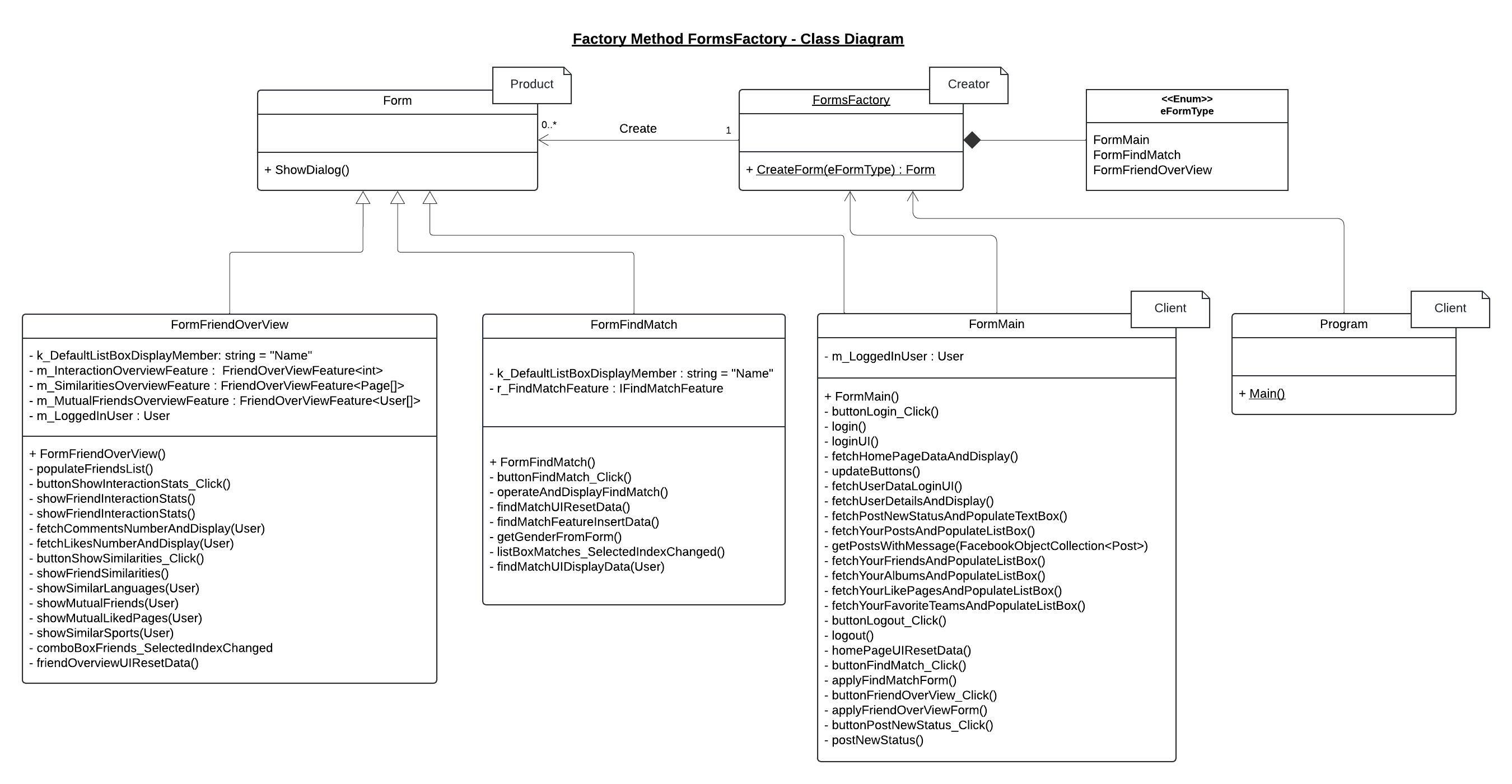
* Sequence Diagram

של האינטרקציה המורכבת ביותר.



* Class Diagram

**Static Factory Class – Creator:** FormsFactory  
**AbstractProduct:** Form  
**ConcreteProducts:** FormMain, FormFindMatch, FormFriendOverView  
**Clients:** Program, FormMain



### תבנית מס' 3 – Proxy

* סיבת הבחירה / שימוש בתבנית:

בפיצ'ר למציאת זיווג פוטנציאלי, המחלקהFormFindMatch עושה שימוש במחלקהFindMatchFeature בצורה שכבר עובדת ומתאימה לצרכיינו ולכן לא היינו רוצים לשנות אותה, אבל כן היינו רוצים לשכלל אותה, לכן ההתייחסות למחלקה היא כמחלקת לגאסי.

השכלול שהיינו רוצים למממש במערכת, הוא ניהול של רשימת החברים של המשתמש באופן יעיל על מנת לשפר את הביצועים, על ידי מניעת פניות חוזרות ונשנות לשרת של פייסבוק.

המשתמש עשוי לבצע חיפושים מרובים עם העדפות שונות, כאשר כל שינוי בפרמטרים יכול להשפיע על התוצאות. כדי למקסם את זמן התגובה ולהפחית את עומס הפניות לפייסבוק לקבלת רשימת החברים של המשתמש, השתמשנו בתבנית של Proxy מסוג Cache Proxy.

* אופן המימוש:

מימשנו באפליקציה ממשק משותף שנקרא IFindMatchFeature, הפרוקסי מתחזה ל- FindMatchFeature ולכן מבחינת המחלקה FormFindMatch ההחלפה הינה שקופה, על ידי פולימורפיזם בין המחלקת פרוקסי לבין המחלקה המקורית.

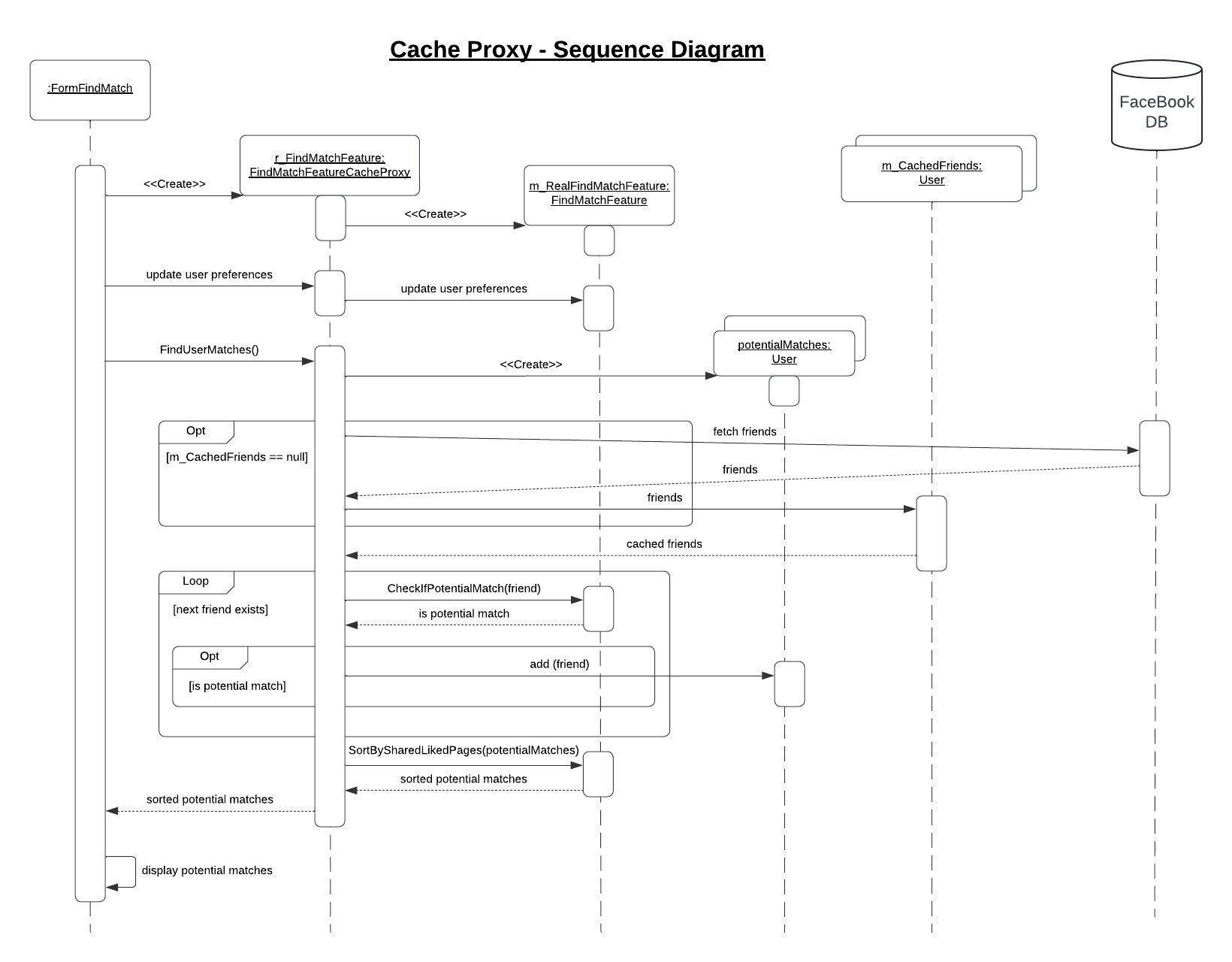
הפרוקסי ממומש באופן Object Proxy, בכך שהוא מחזיק אובייקט מסוג FindMatchFeature בקומפוזיציה.

בנוסף, הוא מחזיק אוסף מסוג FaceBookObjectCollection<User>שתפקידו להיות ה- Cache של רשימת החברים של המשתמש.

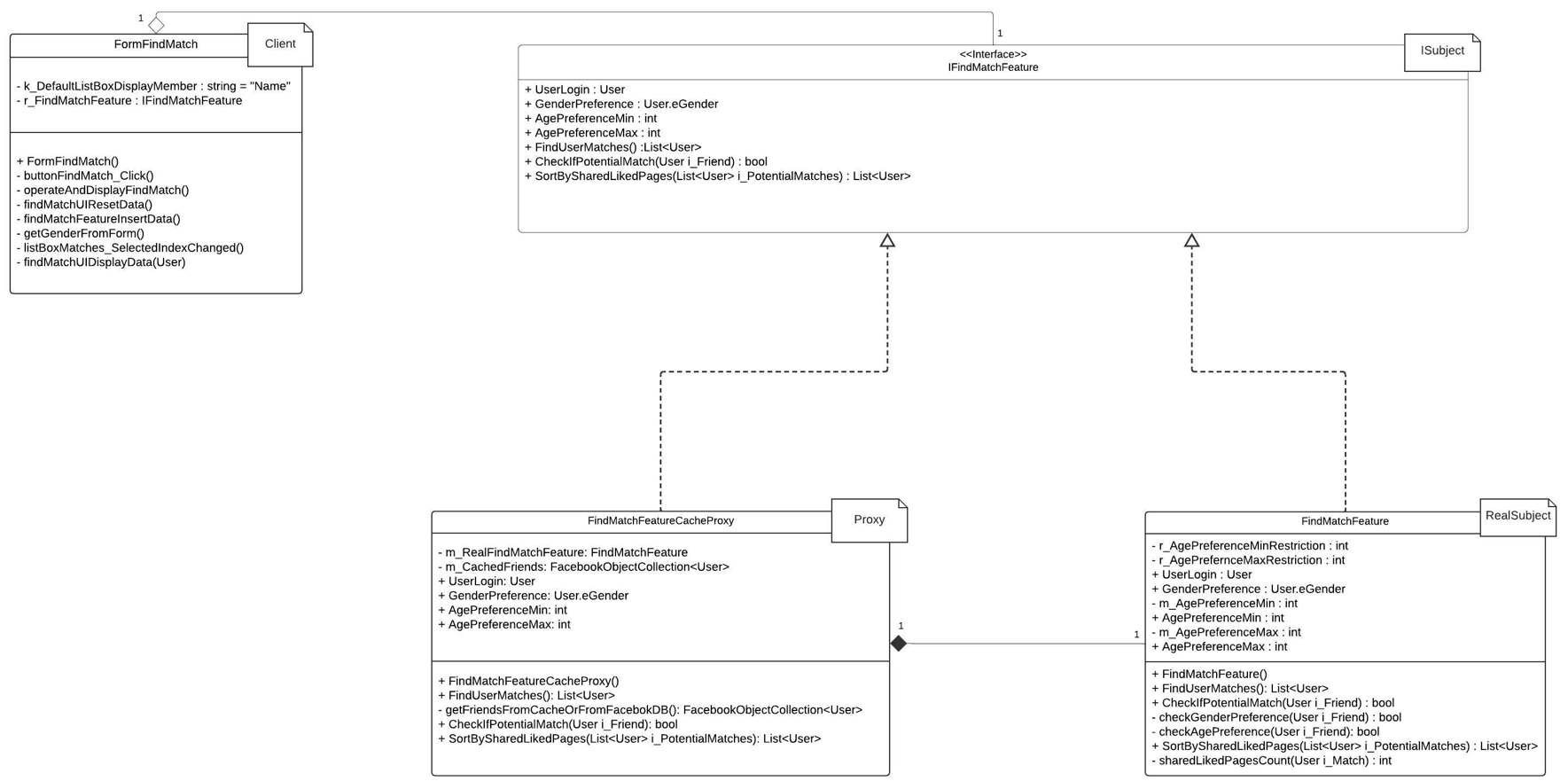
כחלק מהממשק אותו הפרוקסי מממש, יש מימוש אלטרנטיבי למטודה FindUserMatches, כאשר במימוש מתבצעת בדיקה האם רשימת החברים נמצאת בCache על מנת למנוע פניה חוזרת לשרת של פייסבוק.

ניתן למצוא את המימוש בקוד תחת התיקייה Proxy והתקייה Features.

* Sequence Diagram



* Class Diagram



### תבנית מס' 4 – Strategy

* סיבת הבחירה / שימוש בתבנית:

בפיצ'ר FriendsOverView נדרשת היכולת להביא נתונים שונים שקשורים לאינטרקציות עם חברים. כגון ספירת לייקים ותגובות שחבר מסוים ביצע לפוסטים של המשתמש, חיפוש חברים משותפים, מציאת דפים ותחומי עניין משותפים.

בהתחשב באופי הדינמי והמתפתח של דרישות המערכת ושל הפיצ'ר, חשוב לשמר אריכטקטורה גמישה וניתנת להרחבה.

בעזרת התבנית של ה-Strategy , הוספת פונקציונליות חדשה לא דורשת שינוי בקוד הקיים, אלא רק הוספת מחלקה חדשה שמממשת את הממשק המתאים. זה מאפשר למערכת להישאר מודולרית וקלה לתחזוקה עתידית.

* אופן המימוש:

**Context**: FriendOverViewFeature

**Strategy Interface:**IFriendOverViewStrategy

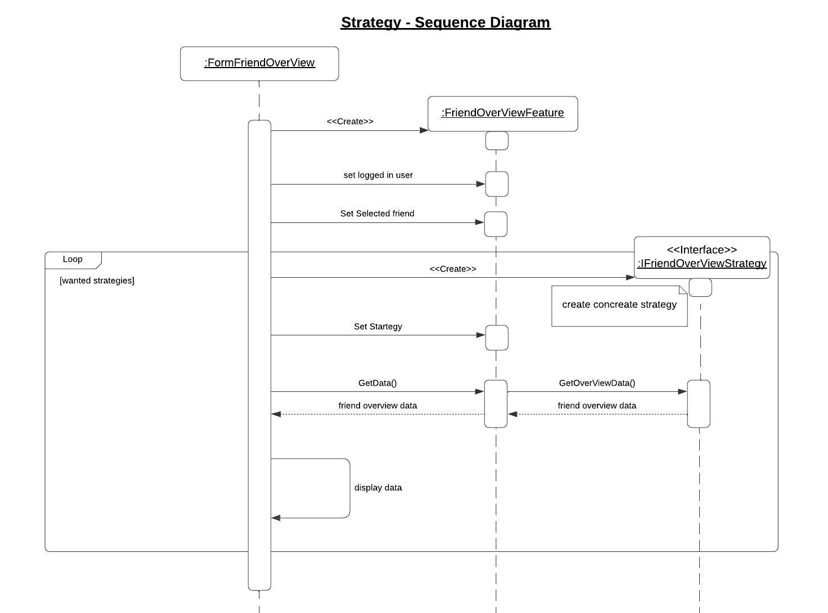
**Concrete Strategies:**CommentsFromFriendStrategy, LikesFromFriendStrategy, MutualFriendsStrategy, MutualLikedPagesStrategy, SimiliarLanguagesStrategy, SimiliarSportsStrategy

במחלקה FriendOverViewFeature ממומשת המתודה GetData שתפקידה לקרוא למתודה GetOverViewData מהאסטרטגיה הנוכחית אותה המחלקה מחזיקה בקומפוזיציה בזמן הריצה.

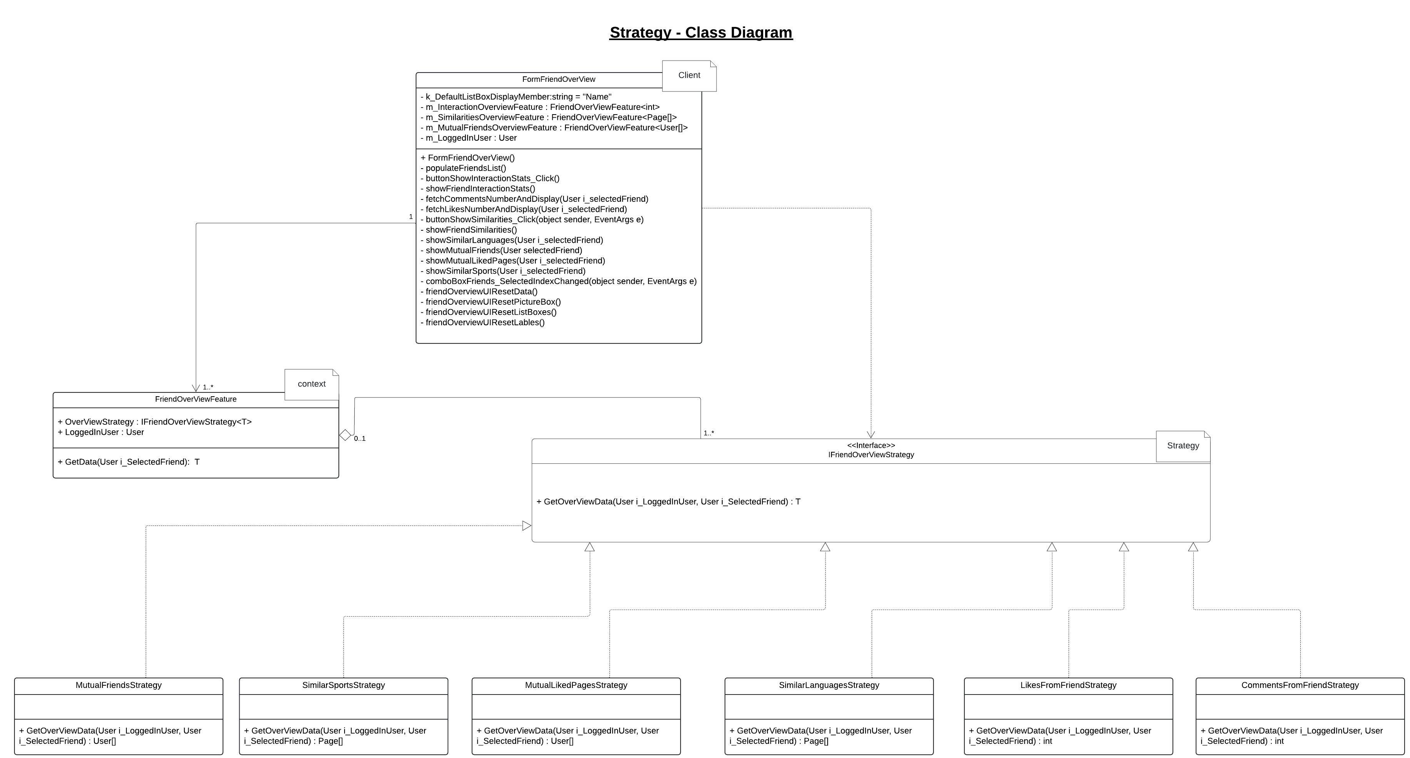
האסטרטגיות הקונקרטיות ממומשות כל אחת באופן שונה והן מממשות את הממשק IFriendOverViewStrategy לפי הטיפוס המבוקש. כך ניתן להוסיף בקלות פעולות נוספות לאפליקציה מבלי לשנות את הקוד הקיים, רק על ידי יצירת אסטרטגיה חדשה והגדרתה כאסטרטגיה הנבחרת במערכת בזמן ריצה.

ניתן למצוא את המימוש בקוד תחת התיקייה Strategy.

* Sequence Diagram



* Class Diagram



**עבודה אסינכרונית עם ממשק משתמש:**

מימשנו במחלקת FormMain של האפליקציה, השימוש בתכנות אסינכרוני ממחיש את הצורך לעבד נתונים רבים משרתי פייסבוק במקביל תוך שמירה על חוויית משתמש חלקה ומהירה. כל פעולה שדורשת פנייה לשרת מתבצעת ב-Thread נפרד, מה שמאפשר ביצוע מקבילי של מספר פעולות ללא חשש לתקיעות או השהיות בממשק המשתמש.

העבודה האסינכרונית ב FormMain -כוללת את הפעולות הבאות:

* שליפת נתוני המשתמש המחובר והצגתם בממשק.
* טעינת פרטים נוספים כמו פוסטים, אלבומים, עמודים שאהב, חברים וכו.'..
* פרסום פוסטים חדשים ועדכונם ברשימת הפוסטים של המשתמש.

הבחירה בשימוש בתכנות אסינכרוני נבעה מהצורך להבטיח שהממשק משתמש ישאר תגובתי ויעיל, בזמן שביצוע הפעולות השונות ובעיקר הפעולות הארוכות של משיכת מידע משרתי פייסבוק לא יהווה עומס על האפליקציה. זה מאפשר למשתמש להמשיך לנווט באפליקציה ולבצע פעולות נוספות בזמן שנתונים נטענים ברקע, מה שמשפר את חוויית השימוש ומזרז את התגובה של האפליקציה.

**עבודה עם Data Binding:**

השתמשנו ב-Data Binding  במחלקת ה-FormMain בצורה שבאה לידי ביטוי בהצגת רשימת החברים, הצגת הדפים שאהב, הצגת האלבומים, והצגת קבוצות הספורט האהובות.

בנוסף, השתמשנו בdata binding להצגת הפוסטים שהמשתמש פרסם.