### 

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

* **Friend Matcher**  
  מציאת חברים עם דברים משותפים לפי הגדרות של גיל, מין, עיר מגורים ועמודים משותפים שעשית להם לייק. החישוב מבוסס לפי ניקוד הניתן על עיר אותה המשתמש מבקש מתוך ערי המגורים של חבריו והעמודים האהובים המשותפים. ניתן לבחור כמה אנשים להציג בצורה ממוינת לפי סינון כמות האנשים אותם המשתמש ירצה לראות.
* **Checkins Finder**מציאת פוסטים של חברים המבוססים על מיקומים ותאריכים.  
  המשתמש מזין שמות של מיקומים או מילות מפתח אותו מעוניין לראות מי מהחברים יצר פוסט עם המיקום. בנוסף, ניתן לסנן את הפוסטים עם המיקומים על פי תאריך על ידי סימון V.  
  לאחר החיפוש ניתן יהיה לראות את כל הפוסטים הרלוונטים עם המיקום הנבחר ועל ידי לחיצה על אותו פוסט ניתן לראות אותו עם המפה מוצגת לצידו הימני של האפליקציה.

### שימוש ב- MultiThreading:

**•**  במחלקה FormMain במתודה OnShown בשורה 27 בקוד נעשה שימוש בעבודה אסינכרונית.

העבודה האסינכרונית מפעילה מתודה בשם populateUiFromFacebookData שדואגת לאתחל את כל המידע הנחוץ למשתמש ומידע הבסיסי שלו בחלון ה-FormMain. יש צורך לעבוד בצורה אסינכרונית מכיוון שתהליך זה לוקח זמן ועלול לתקוע את ממשק המשתמש עד שיסתיים.

**•**  במחלקה FormMain במתודה fetchUserPosts בשורה 121 בקוד נעשה שימוש בעבודה אסינכרונית.

העבודה האסינכרונית מפעילה את המתודה fetchUserPosts שמביאה את הפוסטים של המשתמש. לאחר הבאת הפוסטים, המתודה מחברת אותם ל-DataBinding. תהליך זה לוקח זמן עקב הפנייה לשרתי פייסבוק ולכן העדפנו להשתמש בעבודה אסינכרונית כדי שממשק המשתמש יהיה פנוי.

**•** כמו כן, השתמשנו במתודה Invoke ובתוכה ב-Lambda Expression שמפעילה Action עבור פעולות עבודה עם ממשק המשתמש ופקדי הטפסים כדי שנוכל לפנות ולשלוט בהם בכל תהליכון:

במחלקה FormMain בשורה 66 על מנת לעדכן את רשימת המידע הבסיסי ב-ListBox.

במחלקה FormMain בשורה 77 על מנת להציג את דפי הלייקים של המשתמש ב-ListBox.

במחלקה FormMain בשורה 88 על מנת להציג את רשימת החברים ב-ListBox.

במחלקה FormMain בשורה 99 על מנת להציג את הקבוצות האהובות ב-ListBox.

במחלקה FormMain בשורה 110 על מנת להציג את אלבומי המשתמש ב-ListBox.

### שימוש ב- DataBinding:

במחלקה FormMain השתמשנו ב 2WayDataBinding על מנת לאפשר למשתמש לערוך את הפוסטים אותם כתב דרך תיבת ה description.  
בשורה 121 ניתן לראות את הפונקציה המאפשרת זאת.  
מכיוון שרכיב הPost המקורי שנמצא בdll שקיבלנו אינו מאפשר עריכה של פוסטים קיימים ראינו לנכון לייצר רכיב אחר אשר יעטוף אותו ויספק את שירות העריכה, רכיב זה מחזיק בתוכו אובייקט מסוג Post .  
יצרנו DataSource מהרכיב החדש postAdaperBindingSource וכך איפשרנו יכולת עריכת לפוסטים הקיימים.

### תבנית מס' 1 – [Strategy]

* סיבת הבחירה / שימוש בתבנית:  
  תבנית האסטרטגיה מאפשרת להגדיר משפחת אלגוריתמים (במקרה זה, דרכים שונות לחישוב ניקוד חברים) ולהפוך אותם להחלפה ביניהם. זה מאפשר לך להוסיף אסטרטגיות ניקוד חדשות מבלי לשנות את מחלקת FriendMatcherLogic. בזכות השימוש בתבנית נזכה גמישות בשינויים בקוד הקשורים בפיצר זה ופתיחות להרחבה.

בנוסף השימוש בתבנית נותן לנו הפרדה, התבנית מפרידה בין האלגוריתמים לחישוב הניקוד לבין הלוגיקה של התאמת החברים.

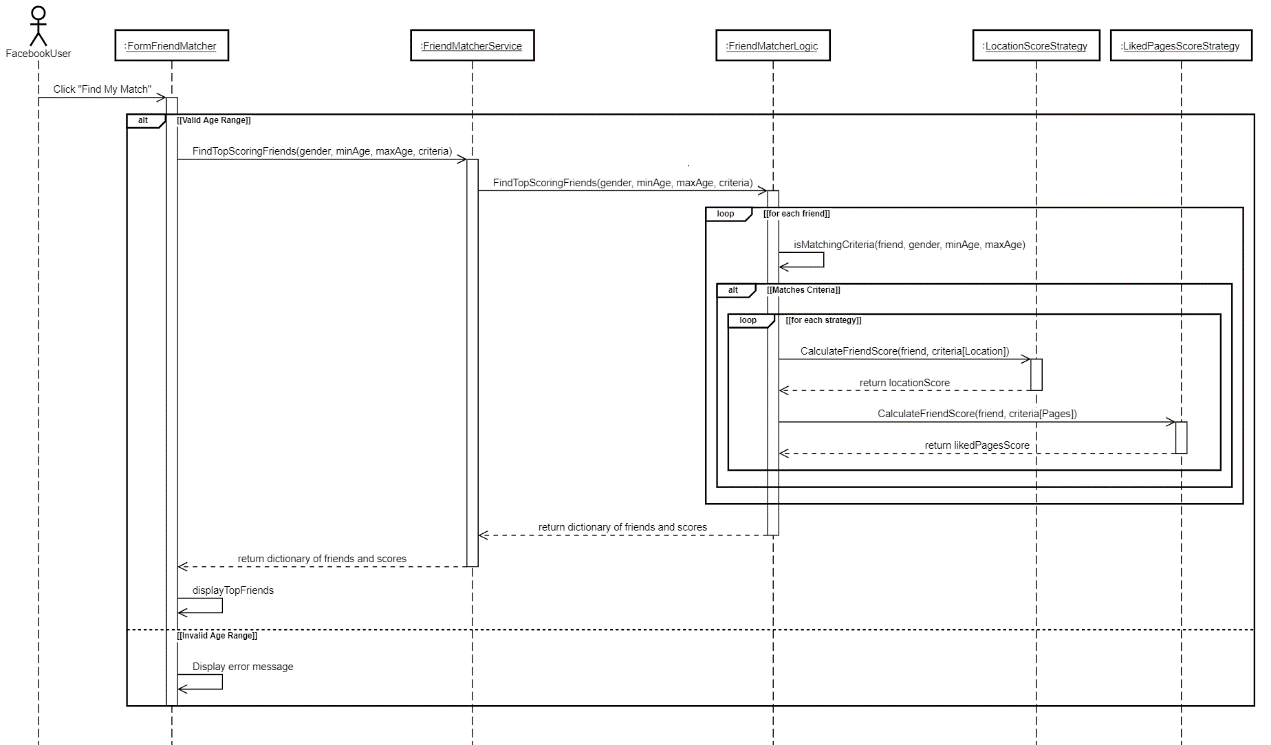
* אופן המימוש:

התבנית מיושמת באמצעות ממשק שמגדיר את השיטה המשותפת, מחלקות קונקרטיות שמיישמות את הלוגיקה הספציפית, ומחלקת הקשר שמשתמשת באסטרטגיות הללו לחישוב התוצאות לפי קלט המשתמש.

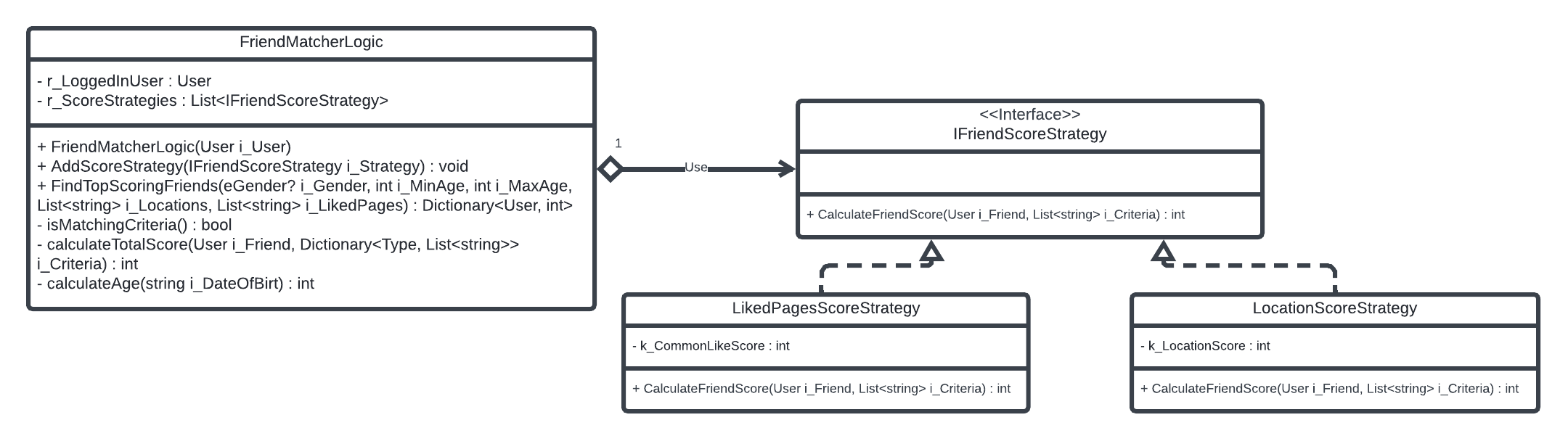
גישה זו מבטיחה שהלוגיקה של חישוב הניקוד יכולה להתרחב או להשתנות בקלות מבלי להשפיע על הפונקציות הבסיסיות של התאמת החברים.

ממשק IFriendStrategy מגדיר מעין חוזה עבור אסטרטגיות ניקוד שונות.  
האסטרטגיות לניקוד LikedPageScoreStrategiy, LocationScoreStrategy מיישמות את הלוגיקה הספציפית לניקוד עבור כל אסטרטגיה.  
במחלקה FriendMatcherLogic ישנו שימוש באסטרטגיות לחישוב הניקוד הכולל לחברים.

* Sequence Diagram



* Class Diagram



### תבנית מס' 2 – [Façade]

* סיבת הבחירה / שימוש בתבנית:

בחרנו להשתמש ב Facade עקב צורך ורצון לפשט את שכבת הUI ככל האפשר.  
הלוגיקה שכל רכיב UI כלל בתוכו סיבכה ללא צורך את שכבת רכיבי ה - UI.אין צורך שברכיבים שאחרים על תצוגת המשתמש גם נבצע פעולות לוגיות כמו מיון, בניית אובייקטים מורכבים וחישובים, רצינו לבצע הפרדה בין הלוגיקה לבין רכיבי חווית המשתמש, ע"י סיפוק ממשק פשוט בין רכיבי ה - UI ללוגיקה.  
השימוש בתבנית זו תורם לנוחות ובטיחות שימוש בשכבה הלוגית.  
בנוסף לכך, תבנית זאת כמובן תורמת לנוחות, בטיחות שימוש ונוחות תחזוקה.

בשל שימוש ב- Data Binding נוצר צורך לעבד את הנתונים מהלוגיקה. כתוצאה מכך, המחלקה שמייצגת את ה- Façade "משטחת את הלוגיקה" כדי שממשק המשתמש יוכל להשתמש בנתונים בצורה נוחה יותר.

* אופן המימוש:

לכל form יש façade משלו על מנת לשמש כבסיס ארכיקטקטוני למודל MVVM:

CheckinsFinderService - אחראי על חיפוש צ'ק-אינים. משמש כ Façade ל FormCheckinFinder

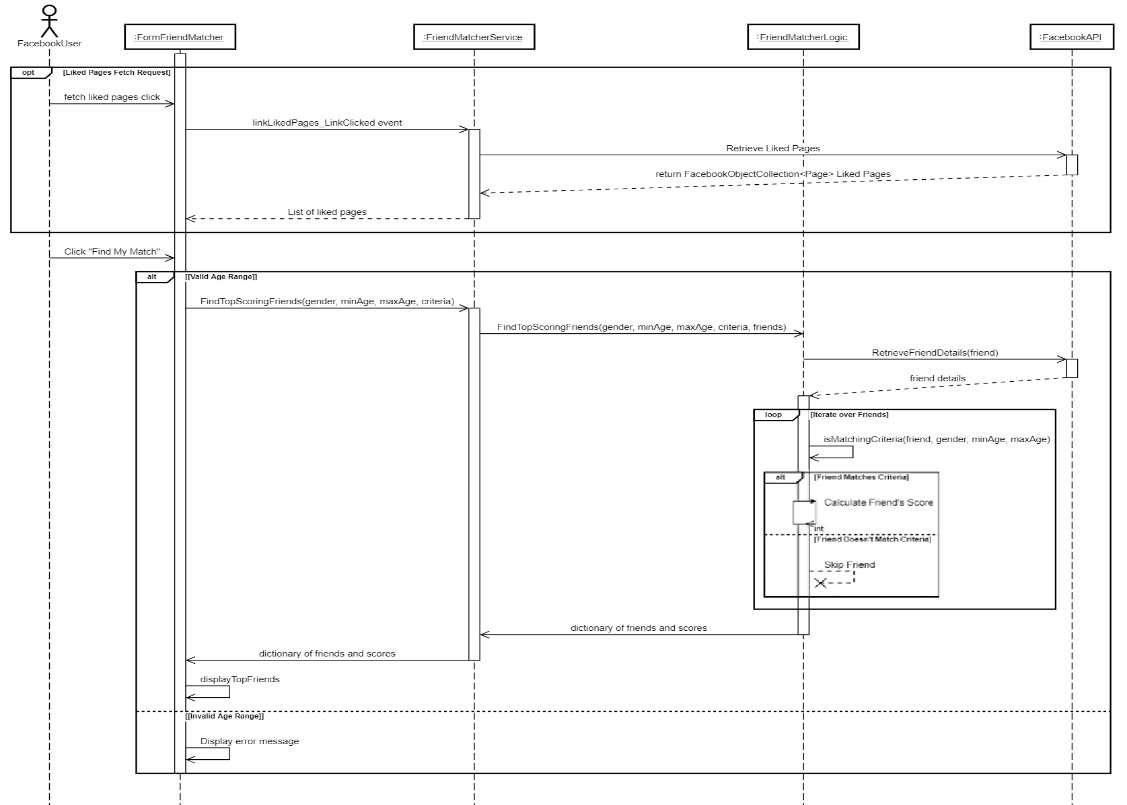
FriendMatcherService - אחראי על התאמת חברים לפי קריטריונים. משמש כ Façade ל FormFriendMatcher

LoginService - אחראי על התחברות ושמירת הגדרות. משמש כ Façade ל FormLogin

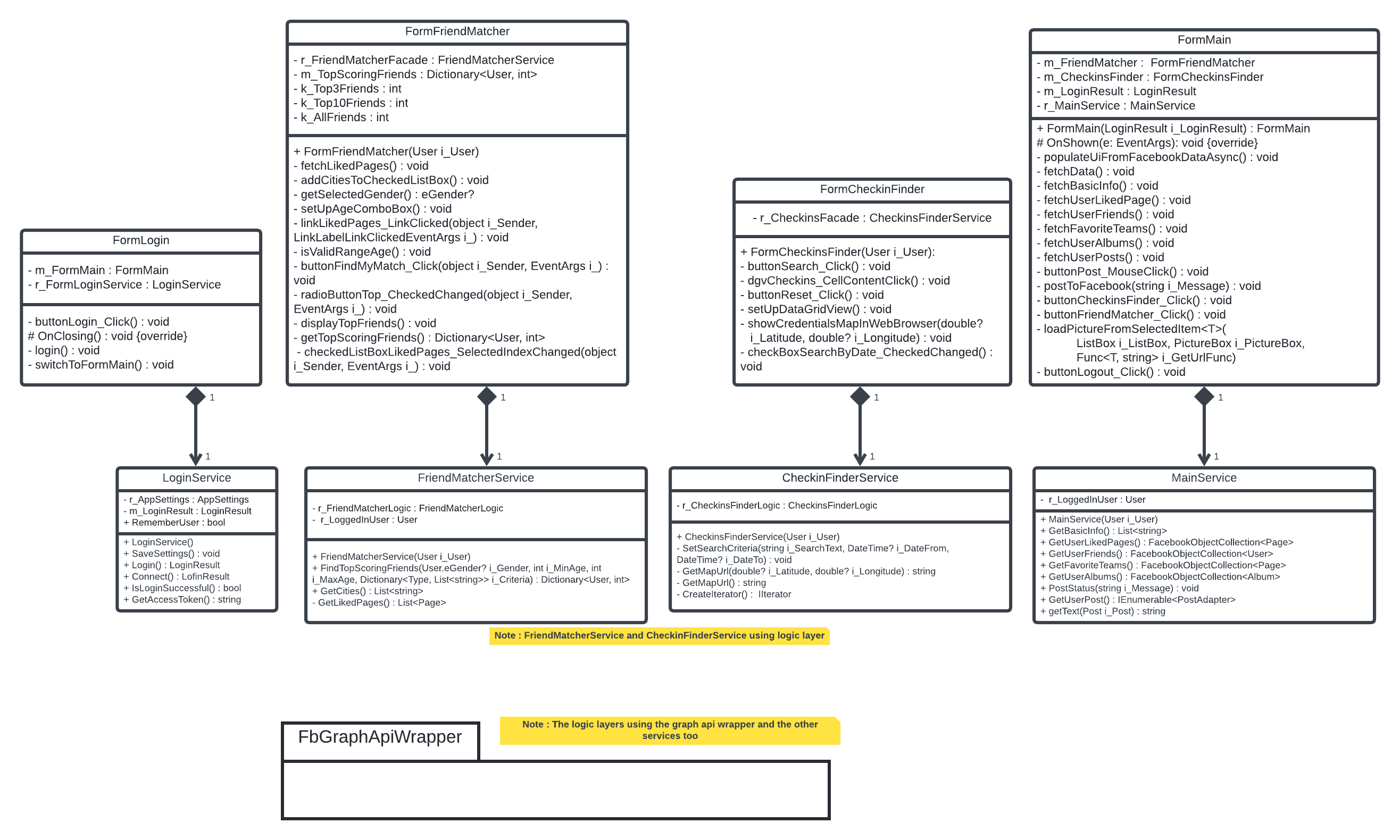
MainService - אחראי על שליפה והצגה של מידע בסיסי על המשתמש. משמש כ Façade ל FormMain

Clients: FormMain, FormFriendMatcher, FormLogin, FormCheckinsFinder  
Facades: MainService, FriendMatcherService, LoginService, CheckinsFinderService

* Sequence Diagram



* Class Diagram



### תבנית מס' 3 – [Iterator]

* סיבת הבחירה / שימוש בתבנית:

בתבנית האיטרטור בקוד שלנו, האיטרטור (CheckinInfoIterator) מאפשר גישה לאלמנטים של CheckinInfo מבלי לחשוף את מבנה הנתונים הפנימי של האוסף.

התבנית מספקת שיטה אחידה ונקיה לעבור על אוסף של CheckinInfo באמצעות הממשק IIterator. זה מאפשר לקוד שלנו לעבור על האוסף בצורה לולאתית ולבצע פעולות שונות על כל אלמנט מבלי להזדקק לשינויים בכל מקום בקוד שמבצע את האיטרציה.

התבנית מאפשרת תמיכה בעבודה עם סוגים שונים של אוספים מבלי לשנות את הקוד המשתמש באיטרטור. זה מבטיח שהקוד שלנו גמיש ומוכן להתרחבות בעתיד, כך שניתן יהיה להוסיף אוספים חדשים מבלי לשנות את הלוגיקה של האיטרטור.

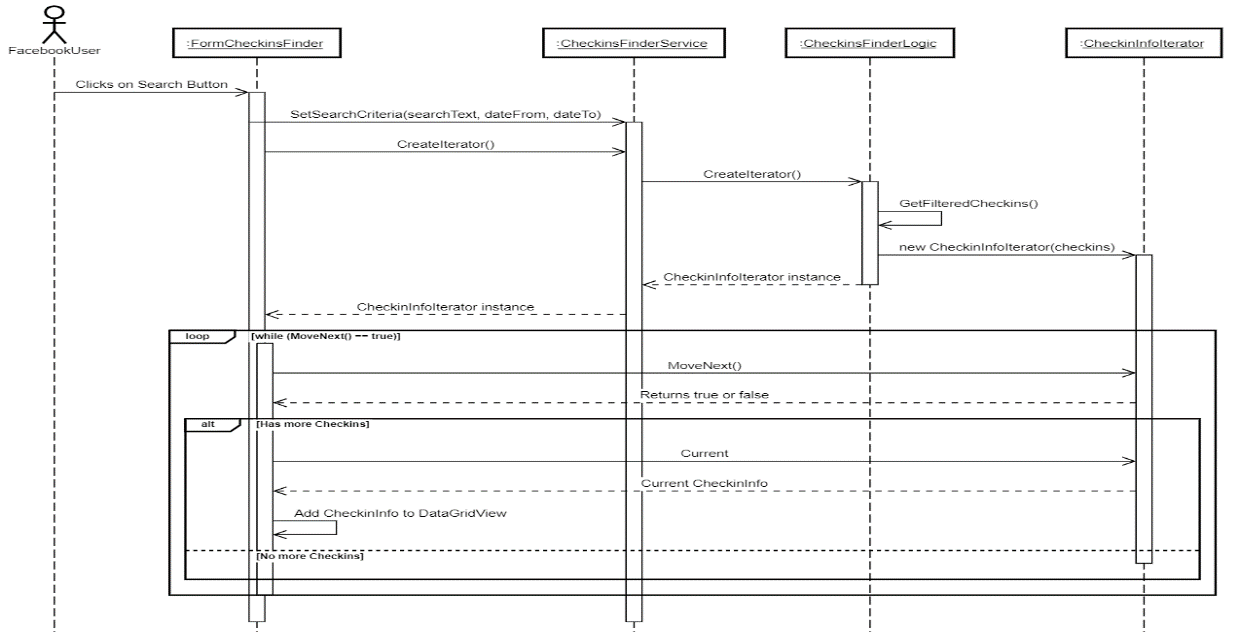
באופן כללי, נרצה לספק ל-Client בתוכנית שלנו (או לכל מחלקה אחרת), אוסף מיקומים זה מבלי לתת לו את האפשרות להוריד או להוסיף מיקומים לאוסף זה תוך שמירת חופש הבחירה למחלקה לבחור איזה מבנה נתונים ברצונה לממש את אוסף המיקומים שלה בעתיד, ובכך לא נצטרך לשנות את הקוד אצל ה Client.

* אופן המימוש:

ממשק (IIterator): מגדיר את החוזה לאיטרטור, כולל שיטות כמו MoveNext, Current, ו- Reset.

מימוש האיטרטור (CheckinInfoIterator): מיישם את הממשק ומספק את הלוגיקה לעבור על רשימת CheckinInfo.  
ממשק לאוסף (IAggregate): מגדיר חוזה למימוש של אוספים שתומכים באיטרטור.  
מימוש האוסף (CheckinsFinderLogic): מיישם את הממשק IAggregate, מספק אוסף של CheckinInfo ומאפשר יצירת איטרטור עבור האוסף.  
**שימוש (**CheckinsFinderService**):** מספק שירות לעבודה עם האוסף, כולל הגדרת קריטריונים, קבלת אוספים מסוננים ויצירת איטרטור.  
FormCheckinsFinder משתמש בשיירות לבצע חיפושים, להציג נתונים ולנווט על המפה.

* Sequence Diagram



* Class Diagram

