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

**פיצ'ר ראשון – מציאת התאמה מבין חברי הפייסבוק**

פיצ'ר דייטינג המאפשר למשתמש למצוא חברים התואמים את תחומי העניין שלו.

המערכת מאפשרת למשתמש לבחור העדפות כגון מין וגיל. המערכת בודקת אצל חברי המשתמש אם החבר נמצא כפוטנציאלי מבחינת מין וטווח גילאים, ואם הוא / היא זמינים למערכת יחסים. (מצב מערכת היחסים יכול להיות כ: קשר פתוח, פרודים, גרושים, אלמנים, רווקים).

בנוסף, למשתמש יש את האפשרות להביט בתמונותיו של החבר מתוך האלבום שלו, ובכך להכיר את חברו בצורה טובה יותר.

המתודה FindAMatch נמצאת במחלקה ApplicationSystemManagerFacade וכן במחלקה MatchAnalyzer, והיא אחראית על מציאת החברים המתאימים (המתודה במחלקה ApplicationSystemManagerFacade קוראת למתודה במחלקה MatchAnalyzer).

**פיצ'ר שני – הזמן האופטימאלי לפרסום פוסט על מנת לקבל את מירב הלייקים**

פיצ'ר זה מתמקד בפופולריות של כלל הפוסטים שהמשתמש פירסם על מנת ליידע את המשתמש מתי הזמן האופטימאלי עבורו לפרסם את תוכן ענייניו.

המערכת מנתחת חישובים עבור כל פוסט כפונקציה של מספר הלייקים והתגובות ממכריו של המשתמש, וזאת על מנת להגיע לתוצאות מדויקות מבחינת התחשבות בזמנים לפרסום הפוסט - יום ושעה ביום (בוקר, צהריים, אחר-הצהריים, ערב, לילה, לקראת בוקר).

המערכת מחשבת את הציון עבור כל פוסט בהתאם לפרמטרים שציינו לעיל, ובנוסף מתחשבת בפקטורים חיצוניים כגון: תגובה = 3 לייקים.

המתודה BestTimeToPost נמצאת במחלקה ApplicationSystemManagerFacade וכן במחלקה BestTimeToShare, והיא אחראית על מציאת הזמן האופטימאלי לפרסום הפוסט (המתודה במחלקה ApplicationSystemManagerFacade קוראת למתודה במחלקה BestTimeToShare).

### תבנית מס' 1 – Facade

* סיבת הבחירה / שימוש בתבנית:

על מנת שממשק המשתמש לא יכיר באופן ישיר את העולם הלוגי של המערכת, החלטנו לייצר מחלקה מיוחדת, נפרדת, שנמצאת בתווך שבין השכבה של ממשק המשתמש לבין העולם הלוגי של המערכת. מדובר במחלקה .ApplicationSystemManagerFacade

תפקידה של מחלקה זו היא "לשטח" את העולם הלוגי, כלומר להתמודד עם כל המורכבות של העולם הלוגי, לעבוד מולו, ומספקת כלפי "מעלה" למי שמשתמש בה בצורה פשוטה, שטוחה ונוחה את הצרכים הנדרשים עבורה.

היא עושה זאת בכך שמסתירה מאחוריה אוסף של תתי-מערכות כדי להקל על מי שמשתמש בה לבצע פעולה או פעולות מסויימות בעזרתה, במקום לעבוד ישירות מול תתי-המערכות.

השתמשנו בתבנית זו בצורתה ה"האטומה" (שכבת ממשק המשתמש אינה מכירה את המערכות הלוגיות). בנוסף, מספקת לצרכן הקלה (כפי שציינו, לא יצטרך להתעסק עם המורכבות הלוגית של המערכת), בטיחות שימוש (לא יוכל לגרום נזק על ידי הפעלה לא תקינה של רכיב לוגי בשוגג), נוחות תחזוקה (באמצעות עדכון ה-Facade במידת הצורך).

* אופן המימוש:

המחלקה ApplicationSystemManagerFacade שמקבילה לשכבת ה-View Model מהווה הפרדה בין שכבת ה-View לבין שכבת הModel.

ה- Client מייצר מופע של מחלקה זו, ובאמצעותה נחשפים לו השירותים הבסיסים שהוא צריך, המתודות וה-Properties הרלוונטים לו.

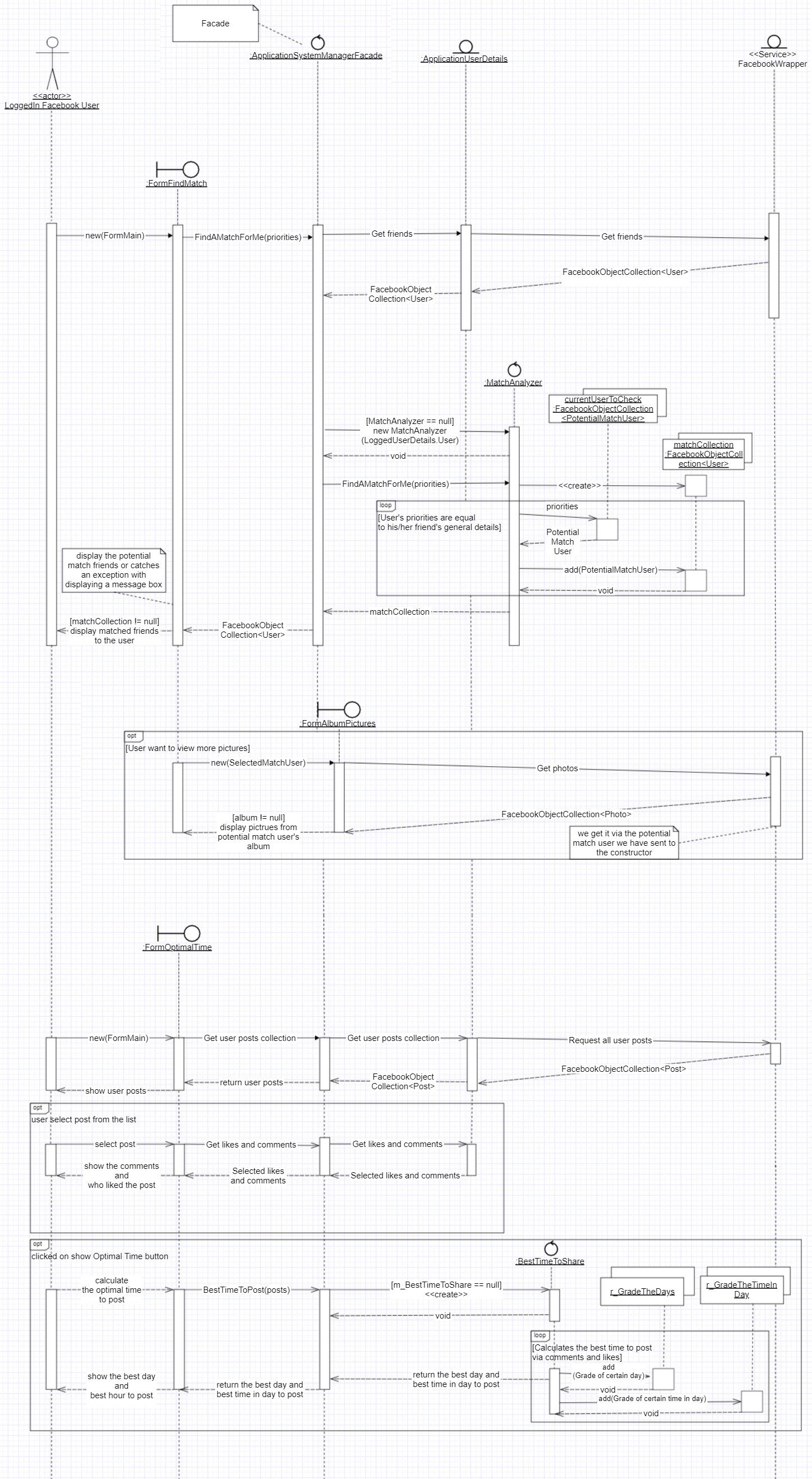
כחלק מהשירותים ש- ApplicationSystemManagerFacade מספקת, מדובר על המתודות FindAMatchForMe ו- BestTimeToPost.

בעקבות הפעלת מתודות אלו ב- ApplicationSystemManagerFacade, תתבצענה קישור וקריאה ללוגיקה הממומשת בשכבת ה-Model, והמידע המוחזר לצרכן יהיה שמיש עבורו באמצעות ה-Properties המתאימים.

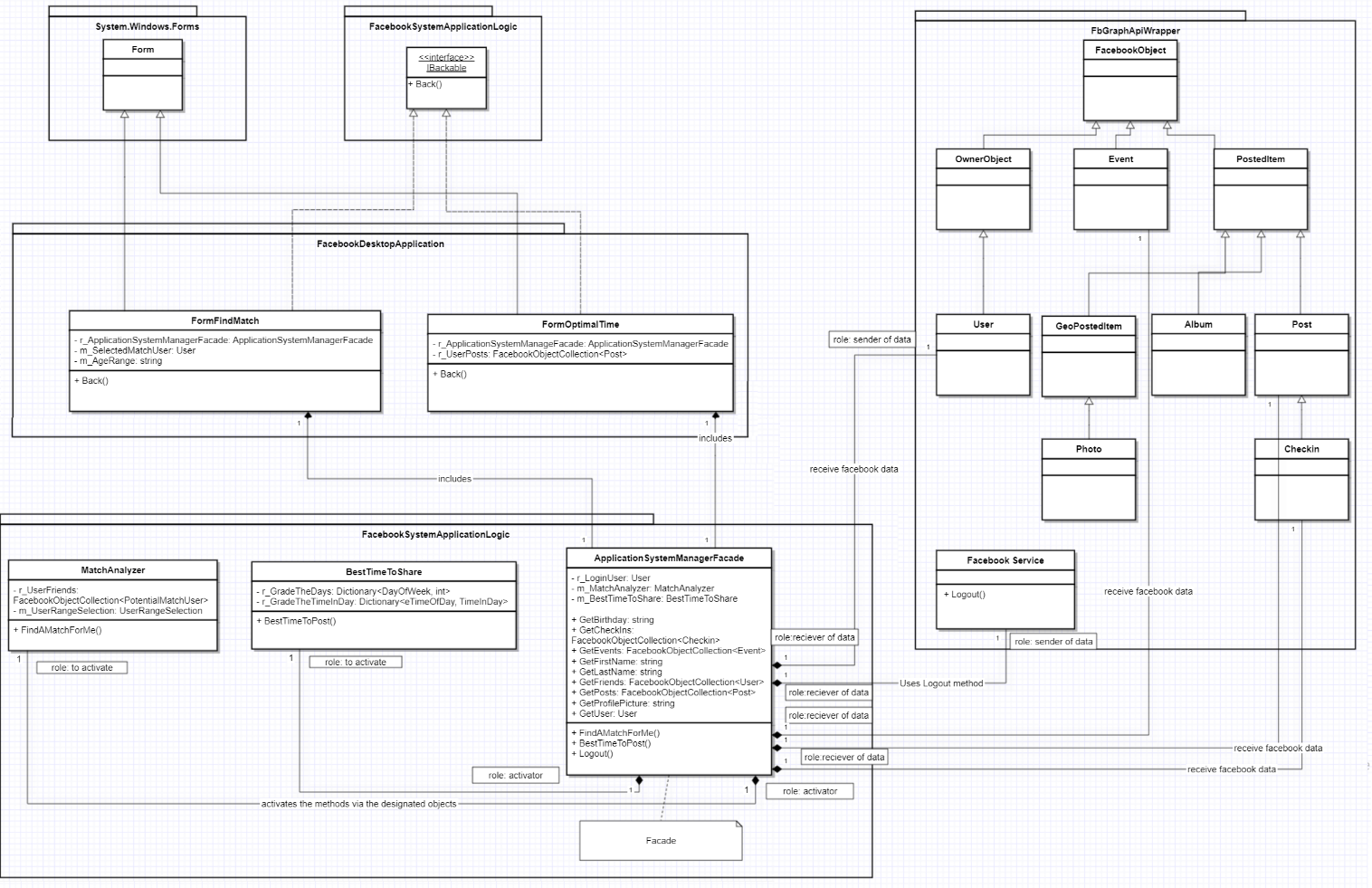
המחלקה ApplicationSystemManagerFacade מחזיקה מופעים של אובייקטים מטיפוס MatchAnalyzer, BestTimeToShare, על מנת לספק נוחות ובטיחות שימוש לצרכן, כך שאחריות ניהול אובייקטים אלה יתבצע באופן מוסתר ומרוכז בשכבת ה- View Model.

כלל הקריאות המפעילות את הפיצ'רים באפליקצייה עוברות דרך המחלקה ApplicationSystemManagerFacade שהיא ה-Facade, ומוחזרות דרכו אל ה- Client.

* Sequence Diagram



* Class Diagram



**Facade:**

ApplicationSystemManagerFacade

**Sub-system Classes**:

User

Post

Event

Checkin

Logout

MatchAnalyzer

BestTimeToShare

**Client:**

FormFindMatch

FormOptimalTime

### תבנית מס' 2 – Proxy

* סיבת הבחירה / שימוש בתבנית:

אנו מעוניינים להציג ברשימת ההתאמות בנוסף לשם המשתמש, גם את גילו המדוייק של החבר הנמצא מתאים וגם את מרחקו מה-LoginUser הנוכחי בהתאם למיקומו הגיאוגרפי של הנייד של ה-LoginUser.

במילים אחרות, אנחנו מעוניינים במחלקה שתתחזה לרכיב המקורי, כלומר ממשק זהה לשני המחלקות, וכן תספק שיפורים, שכלולים שלא היו קיימים ברכיב המקורי.

בעקבות כך שיש לנו שני Legacy Components אחד מהם הוא User והשני הוא listBox, התעניינו בלהציג את הפרטים של ה- User מעבר ל-DisplayMember Property שלו.

מכך ש- listBox יודע להפעיל את ה-ToString של User, החלטנו לממש אובייקט נפרד שגם הוא מספק את הממשק ש-listBox רוצה.

במקרה הזה הוא מספק את הממשק של Object והוא מממש ToString, מחזיק RealSubject – User, ומציע את ה- ToStringהמשוכלל יותר.

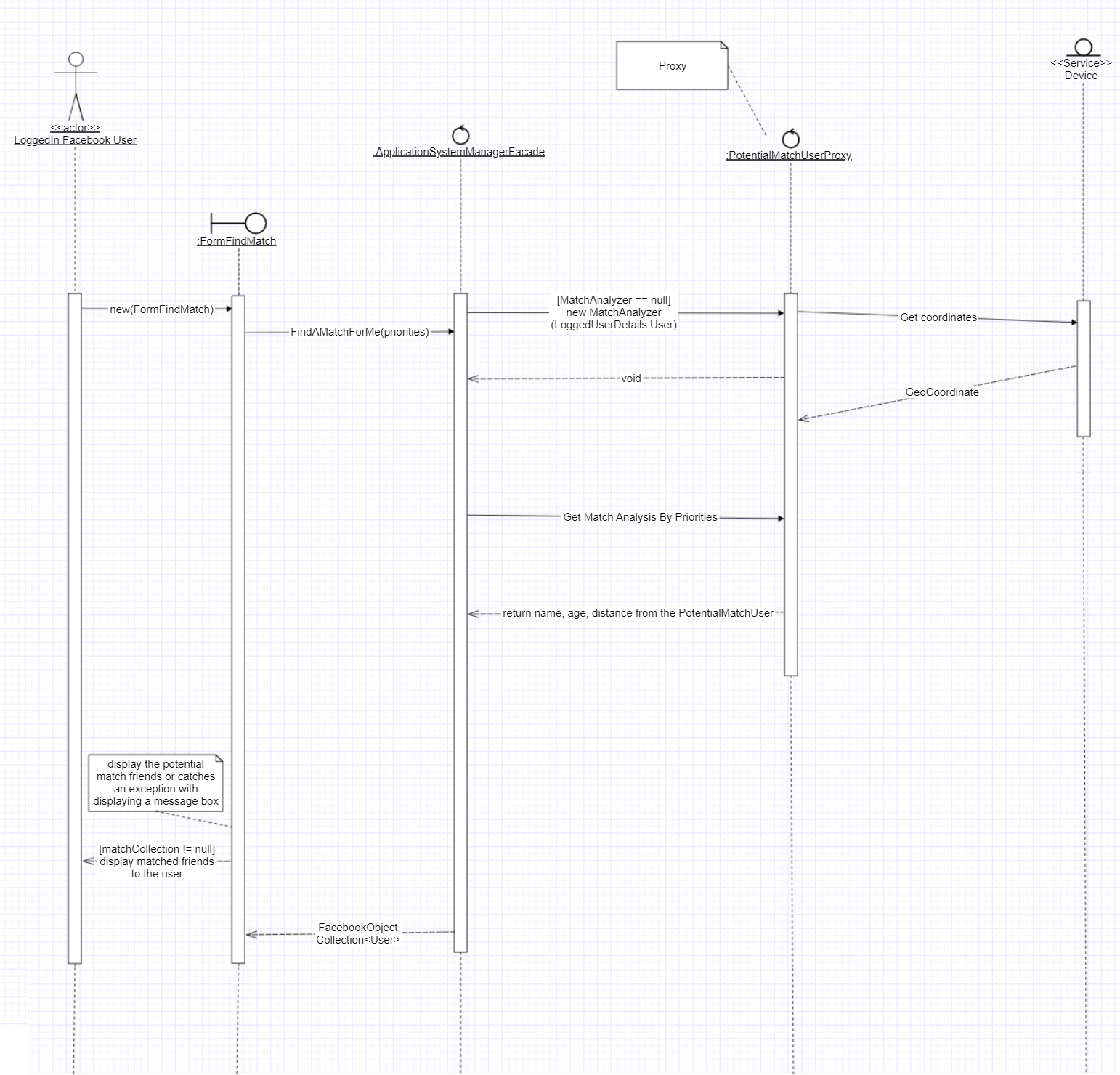
* אופן המימוש:

המחלקה PotentialMatchUserProxy אחראית על התחזות לרכיב המקורי, ומשכללת אותו לטובת ה-Client. לכן, בעת הוספת Itemים ל-listBoxFriends, נוסיף את ה-Itemים מסוג PotentialMatchUserProxy אשר מהווה את המחלקה המשוכללת.

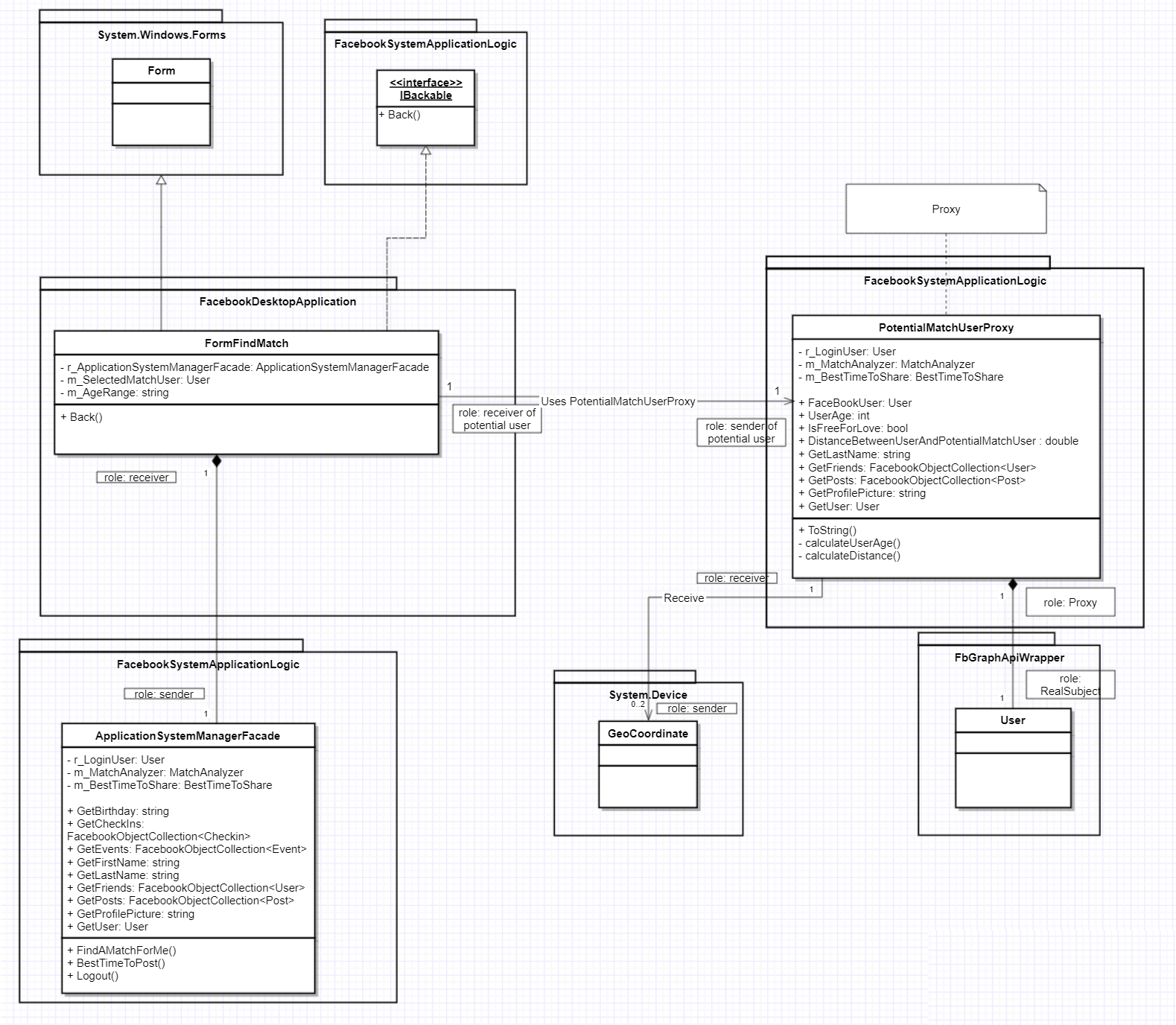
"השקיפות" בין המחלקות בצד ה-Client באה לידי ביטוי על ידי סיפוק השירותים הללו באמצעות המתודה ToString.

בעקבות הפעלת מתודה זו, מה שנראה על גבי ה-listBox יהיו פרטים של שהחבר שנמצא מתאים על ידי האפליקציה (שם פרטי, שם משפחה, גיל, מרחק מיקום גיאוגרפי מהמשתמש המחובר עד אליו בקילומטרים).

* Sequence Diagram



* Class Diagram



**Proxy:**

PotentialMatchUserProxy

**Real Subject:**

User

**Client:**

FormFindMatch

### תבנית מס' 3 – Factory Method

* סיבת הבחירה / שימוש בתבנית:

תחילה נשים לב כי לכל Form יש מאפיינים משותפים בעקבות ירושתם מהמחלקה Form ועל כן מהווים משפחה פולימורפית כאשר אב המשפחה הפולימורפי הוא Form. על מנת לא לממש את לוגיקת הבחירה והיצירה שוב ושוב, כלומר במקום לממש את מלאכת הבחירה והיצירה ב-Client, מימשנו את הלוגיקה במחלקה נפרדת באמצעות Static Factory Class, ע"י מתודה סטטית, ובכך ה-Client ימסור את ה-Context ויקבל את המופע באופן אבסטרקטי של אחד ה-Formים. בנוסף, מספקת לצרכן קוד יותר "נקי" (פחות שורות קוד עמוסות, מקוננות וכו' בממשק המשתמש), קונסיסטנטיות, reuse, וכן פחות דרישה לתחזוקת קוד כאשר בעתיד ה-Client ירצה להרחיב את המשפחה הפולימורפית.

* אופן המימוש:

המחלקה FormFactory שאחראית על מימוש מלאכת הבחירה ולוגיקת היצירה ב-Client, מספקת את השירותים הללו באמצעות המתודה CreateNewForm.

בעקבות הפעלת מתודה זו, תתבצע מלאכת הבחירה בעזרת eFormType שמכיל סט סופי של סוגי Formים שהוגדרו עבור ה-Formים הנדרשים.

בחירתו של ה-Client מובאת למתודה זו ולאחר מכן נוצר מופע של סוג ה-Form שנבחר בהתאמה, ומ-CreateNewForm בעצם חוזר טיפוס מסוג אובייקט האב הפולימורפי – Form.

סוגי ה-Formים השונים:

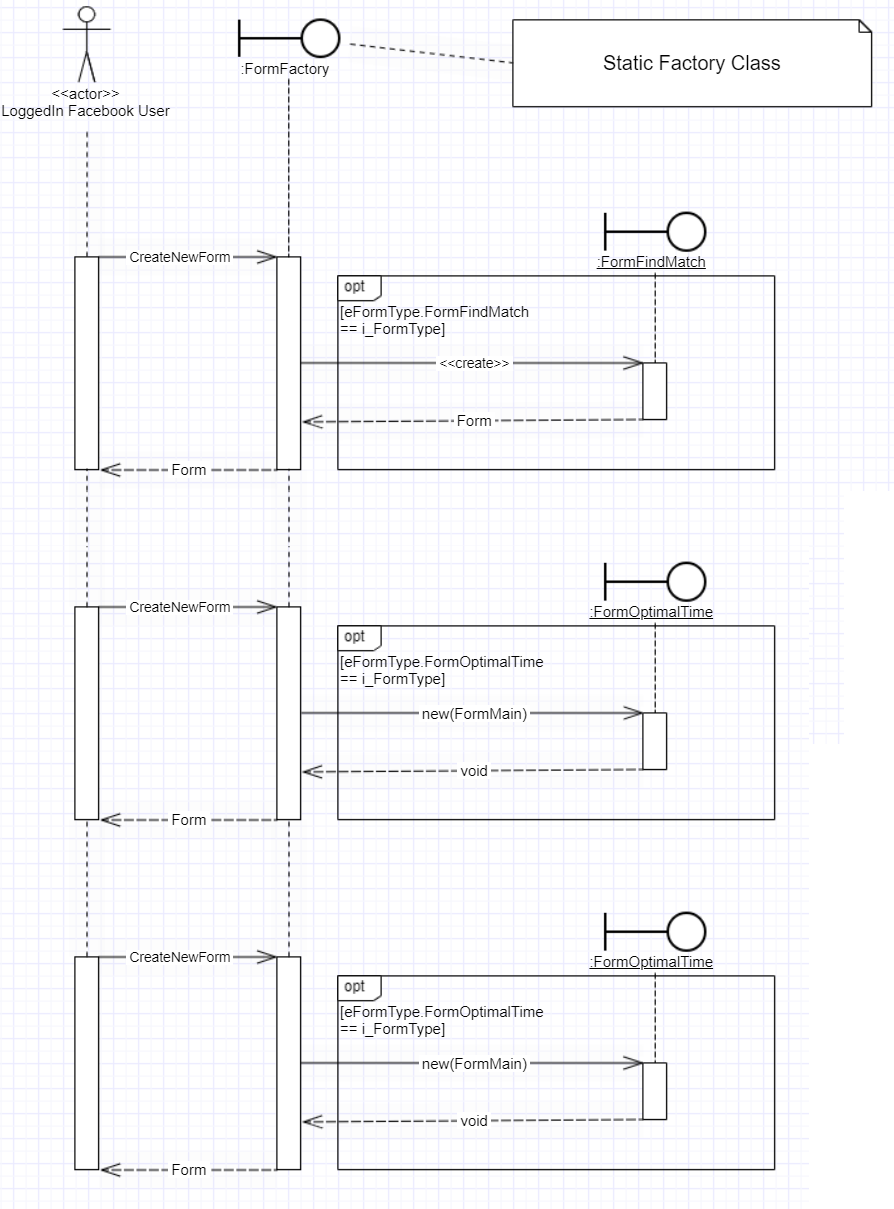
FormFindMatch – מחלקה שמתפקדת כמסך של הפיצ'ר: מציאת התאמה מבין חברי הפייסבוק, ויורשת מ-Form.

FormOptimalTime – מחלקה שמתפקדת כמסך של הפיצ'ר: הזמן האופטימאלי לפרסום פוסט

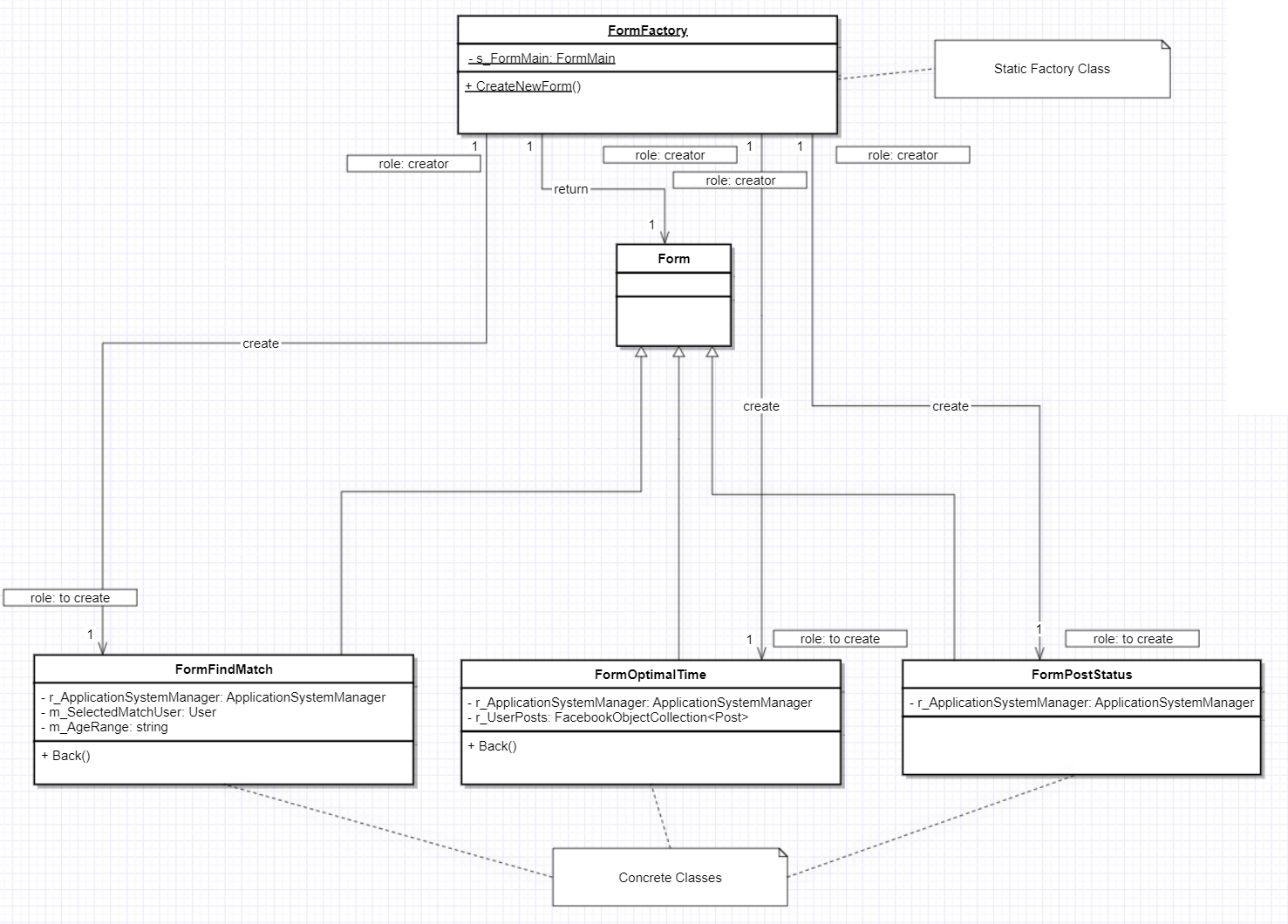
על מנת לקבל את מירב הלייקים, ויורשת מ- .Form

FormPostStatus – מחלקה שמתפקדת כמסך של פרסום פוסט בפייסבוק, ויורשת מ-Form.

* Sequence Diagram



* Class Diagram



**Factory Method:**

FormFactory

**Polymorphic Abstract Class:**

Form

**Concrete Classes**:

FormFindMatch

FormOptimalTime

FormPostStatus

### עבודה אסינכרונית:

המקומות בהם השתמשנו בתכנות אסינכרוני הם:

**FormMain.autoLogin -** מדובר במתודה שמופעלת במידה והמשתמש בחר לשמור את העדפותיו ולכן אנו טוענים את כל דף הבית ע"י מתודה זו. פעולה זו דורשת הבאת מספר רב של נתונים משרת הפייסבוק ולכן הקריאה ל- FormMain.autoLogin מופעלת בThread נפרד.

**FormMain.fetchFriends -** מדובר במתודה שמביאה משרת הפייסבוק את כלל חבריו של המשתמש, ומכיוון שפעולה זו דורשת הבאת מספר רב של נתונים החלטנו להפעיל עבורה Thread נפרד.

**FormMain.fetchCheckins -** מדובר במתודה שמביאה משרת הפייסבוק את כלל המקומות בהם המשתמש עשה צ'ק-אין, ומכיוון שפעולה זו דורשת הבאת מספר רב של נתונים החלטנו להפעיל עבורה Thread נפרד.

**FormMain.fetchEvents -** מדובר במתודה שמביאה משרת הפייסבוק את כלל האירועים בהם נחשף המשתמש, ומכיוון שפעולה זו דורשת הבאת מספר רב של נתונים החלטנו להפעיל עבורה Thread נפרד.

המתודות לעיל מופעלות ע"י Threadים שונים ובתורן צריכות לעדכן ו/או להוסיף פקדים לפאנלים ו/או לאובייקטים שנוצרו קודם לכן ע"י ה- Main-Thread.

המתודה Invoke() , היא מתודה שמשתמשת ב-Delegate ששמו Action על מנת להפעיל את אותן מתודות האחראיות על תחזוקה קונקרטית עבור ה-Thread הנוכחי.

מתודה זו מכילה את קטע הקוד הדרוש ע"י Lambda Expression.

כל זאת כיוון שרק ה-Thread שעידכן/הוסיף את האובייקטים רשאי לשנות/לעדכן אותם.

הסיבה לביצוע הפעולות הנ"ל מ- Threadים נוספים, היא חווית משתמש יותר "חיה" ותגובתית.

כלומר משתמש שביצע פעולה ממושכת לאורך זמן (במקרה זה עיכוב בקבלת המידע, הנתונים ע"י ממשק רשת וטעינתו), לא יחווה חלון "קפוא" עד סיום הפעולות.

פעולות רשתיות/שמירה לדיסק ואיכלוס UI, עלולות לייצר תחושה של איטיות כי הן דורשות זמן המתנה ארוך יחסית כל אחת מסיבותיה. בעקבות הפעלת Threadים שונים במקביל, נעשה שימוש יעיל יותר בפס הרוחב של הרשת (Connectionים מכמה כיוונים), מה שמאיץ ונותן תחושה למשתמש של "מהירות" בתהליך קבלת הנתונים ומונע זמן המתנה ארוך לפעילותו של ה-Main-Thread בשאר התוכנית.

### עבודה עם DataBinding:

המקומות בהם השתמשנו ב-DataBinding הם:

**FormMain –** listBoxFriends של המשתמש המחובר מקושר ל- friendsListBindingSource ומציג את רשימת החברים.

בנוסף, listBoxEvents של המשתמש המחובר מקושר ל- eventBindingSource ומציג את רשימת האירועים.

**FormOptimalTime –** listBoxComments של המשתמש המחובר מקושר ל-commentsBindingSource ומציג את רשימת הערות של הפוסטים.

בנוסף, listBoxLikeBy של המשתמש המחובר מקושר ל- likedByBindingSource ומציג את רשימת סימוני הלייקים.

ה- DataBindings לעיל נועדו על מנת להציג חיבור בין data לבין פקדים ב-UI באמצעות BindingSource רלוונטים. ה- BindingSources הללו, ברגע שמכילים reference לקולקציה של נתונים/לאוסף נתונים, בכל רגע נתון הם מחזיקים אחד כזה וקוראים לו Current.

ה-Current של אותו BindingSource הוא זה שהנתונים שלו יוצגו בפקדים ב-UI.

ברגע שנבחר איזשהו Item ב-listBox כלשהו ויהיה איזשהו אירוע של SelectedIndexChanged, אז ה-listBox ילך ל-DataSource ויודיע לו שה-Current שלו השתנה (של ה-DataSource), וברגע שה-Current השתנה הפקדים יאוכלסו בנתונים של Current.

מאחורי הקלעים הקוד ממומש ע"י המתודה InitializeComponent(), בעזרת Reflection.

בנוסף, מאחורי הקלעים ה- Visual Designer שלנו מאפשר לנו לשלוט על לא מעט קונפיגורציות של ה-DataBinding.