### 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 – Strategy

* סיבת הבחירה / שימוש בתבנית:

במקום שהחלקים שאותם אנו רוצים להחליף בעתיד ימומשו או יוכרזו במחלקה של המנגנון (מדובר במחלקה BestTimeToShare) בצורה אבסטרקטית או וירטואלית, נגדיר בנפרד במקום זה בסיס פולימורפי (מדובר במחלקה IFactorStrategy), ובו נגדיר מתודה כאשר היא בעצם public abstract המשמשת כ-injection point שיהיה בעצם abstract method ב- abstract creature נפרד.

במילים אחרות "חור" שניתן "להזרקה" בעתיד, להחלפה בעתיד, קרי בעתיד יהיה אפשר לשנות את התוצאה ואת ההתנהגות של המנגנון מבלי לשנות את הקוד שלו ובלי לממש אותו מחדש, כלומר בלי לשכפל את הקוד שלו.

המתודה אחראית על קבלת שני פרמטרים כפאקטורים חיצוניים ובעזרתם לבצע חישוב השוואתי מסויים שינתח את ציונו של כל פוסט, ותמומש ע"י מי שמעוניין לספק שירות זה.

נציין בנוסף כי תתקיים קומפוזיציה בין המנגנון לבין הבסיס הפולימורפי, שאותו אם נרצה נוכל לשנות בעתיד.

* אופן המימוש:

ה- Mechanism שלנו היא המחלקה BestTimeToShare שבה ממומשת המתודה analyzeTheBestTime.

במתודה זו מוגדרים האובייקטים שמשמשים כפאקטורים חיצוניים האחראיים על ציונו של כל פוסט בהתאם להחלטתו של המשתמש. הציון מתבצע במתודה CurrentFactors שמספקת שירות שאחראי על אסטרטגיית ניתוח ציונו של הפוסט בבסיס הפולימורפי, שהוא המחלקה IFactorStrategy.

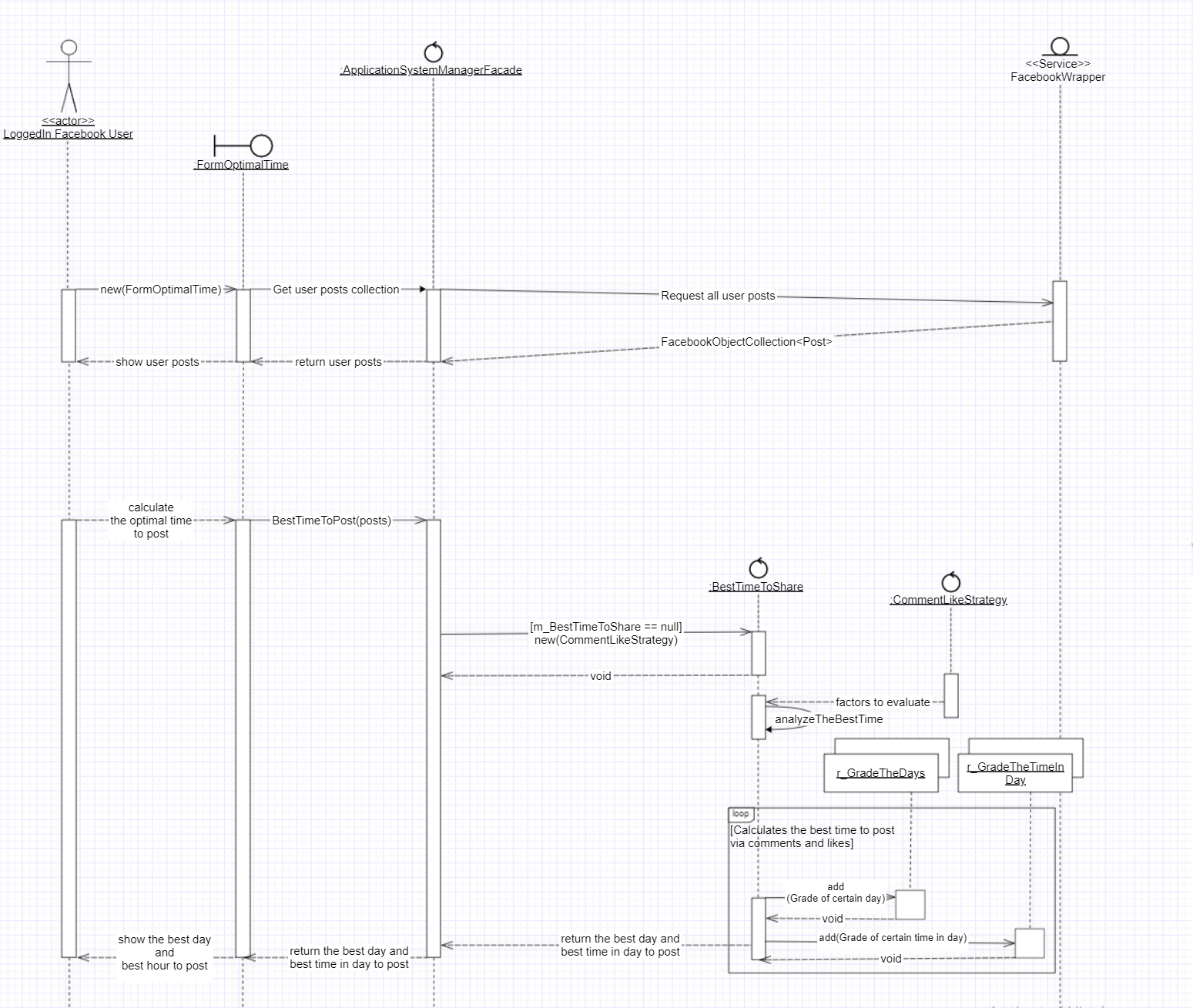
את אופן מימוש האסטרטגיה הקונקרטית מבצעת המחלקה CommentLikeStrategy, אשר מיישמת את המתודה CurrentFactors שמספקת את השירות הנ"ל באופן בו המשתמש ראה לנכון להגדיר חישוב ציון אופטימאלי לפרסום פוסטים בזמן נתון, כלומר החישוב הקונקרטי מתבצע במתודה הזו במחלקה האסטרטגית הקונקרטית המיועדת.

במקרה הנ"ל מדובר שתגובה = 3 לייקים.

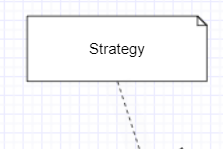
בעתיד, אם ירצו, יוכלו להגדיר אופן מימוש אסטרטגיה קונקרטית אחרת הרלוונטית למשתמש, כך שההחלפה תתבצע רק ברמת השיטה האסטרטגית ותו לא - באמצעות המחלקה MaleLikeStrategy (זכר = 2 לייקים) והמחלקה DivorcedLikeStrategy (גרוש\ה = 5 לייקים).

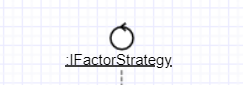
המחלקה ApplicationSystemManagerFacade מחזיקה מופע של אובייקט מטיפוס BestTimeToShare, על מנת לספק נוחות ובטיחות שימוש לצרכן, ובעת השימוש באובייקט נוכל לשנות את השיטה האסטרטגית לשיטה אחרת כרצוננו מכיוון שהבסיס הפולימורפי מוחזק בקומפוזיציה באובייקט.

* Sequence Diagram

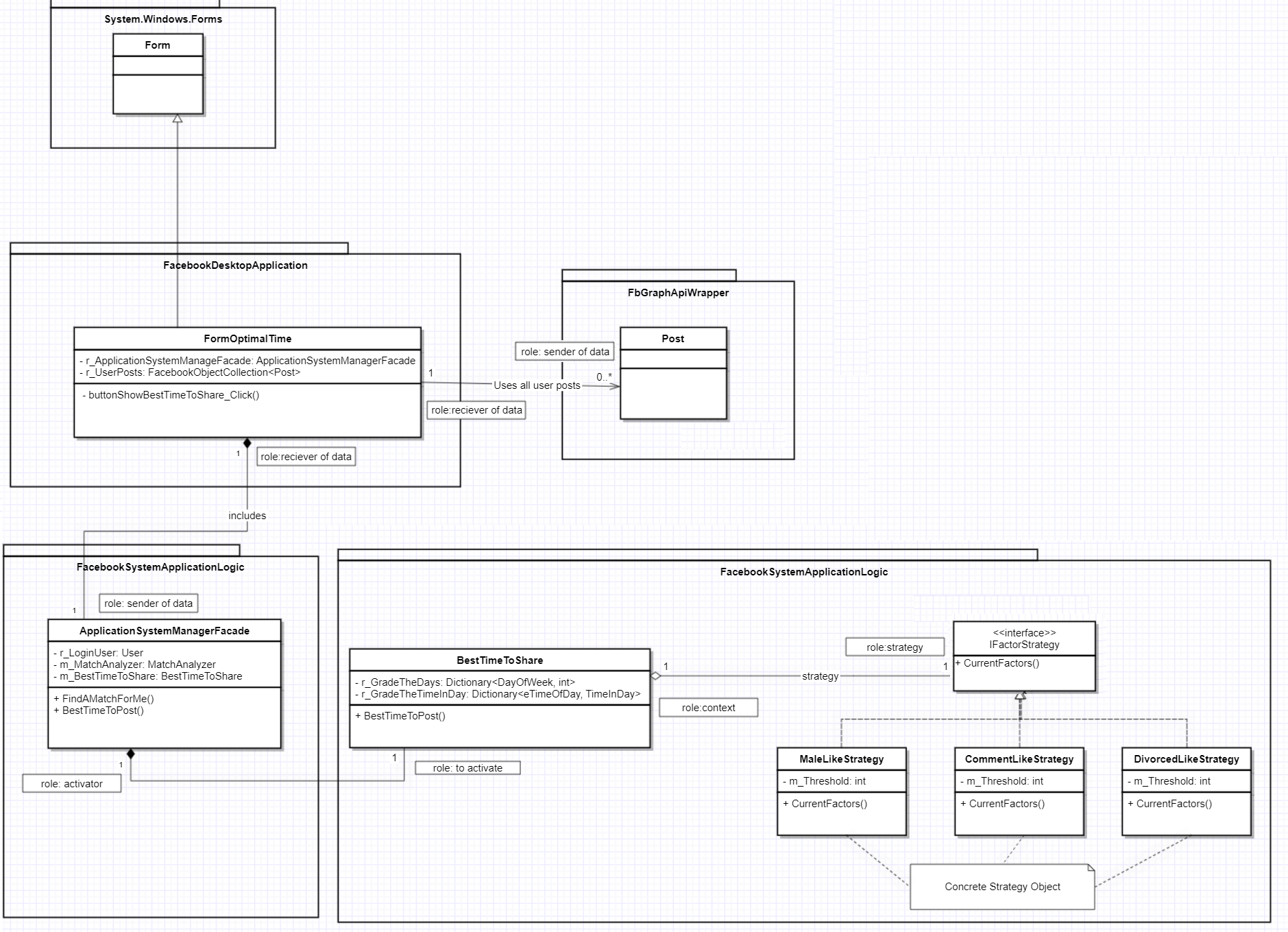








* Class Diagram



**Strategy:**

IFactorStrategy

**Concrete Strategy:**

CommentLikeStrategy

MaleLikeStrategy

DivorcedLikeStrategy

**Context:**

BestTimeToShare

### תבנית מס' 2 – Visitor

* סיבת הבחירה / שימוש בתבנית:

שני יצורים או יותר (מדובר במחלקה FormFindMatch ובמחלקה FormOptimalTime) רוצים לממש שירות של צביעה וסגירה, אבל לא רוצים לשכפל קוד.

השירות שאותו לא נרצה לשכפל הוא שירות "שמשפץ", משנה את ה-state במחלקות האקטיביות, הקונקרטיות, משהו חיצוני.

כדי לעשות זאת ולהימנע מהיכרות ציקלית עם "המשפץ" (מדובר במחלקה PainterCloserVisitor), "המשפץ" מכיר את המחלקות האקטיביות בצורתן האבסטרקטית (מדובר במחלקה Form), ולא מכיר אותן בצורתן הקונקרטית.

אם בעתיד נרצה להוסיף מישהו שזקוק ל"שיפוץ" (כלומר טופס נוסף שיורש מ-Form), אז ניתן לעשות שימוש חוזר ע"י קריאה לאותו "המשפץ" שהזכרנו.

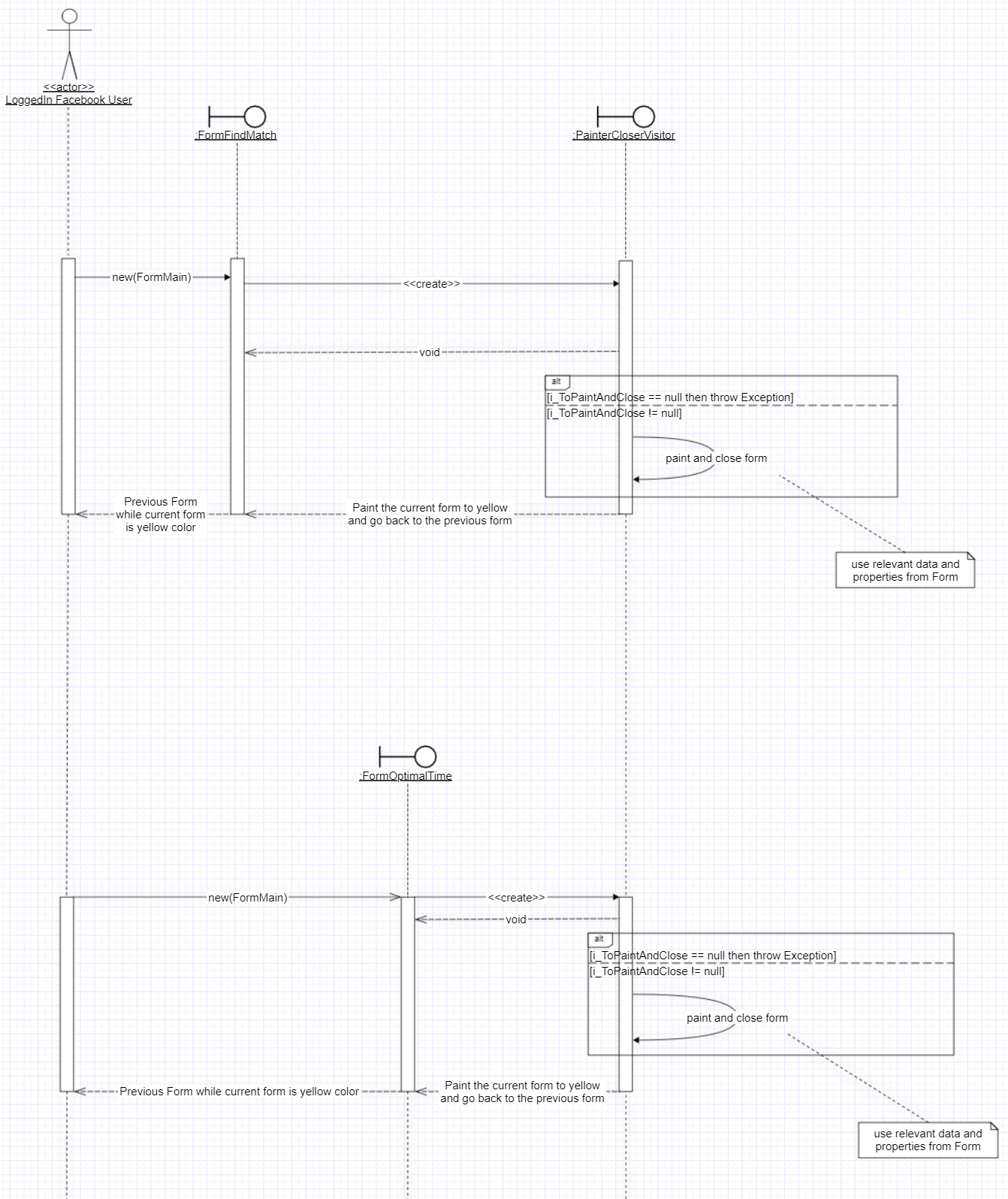
* אופן המימוש:

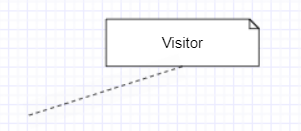
המחלקה Form מספקת את שירות ה-Properties והמתודות המתאימים. במקרה שלנו מדובר ב-Property שנקרא BackColor שאחראי על צביעת הרקע של הטופס באפליקציה, ובמתודה Close שאחראית על סגירתו של הטופס באפליקציה.

המחלקה FormFindMatch והמחלקה FormOptimalTime יורשות מ-Form ובעצם מקבלות בירושה את ה-Properties והמתודות המסופקים כחלק משירותיו של Form.

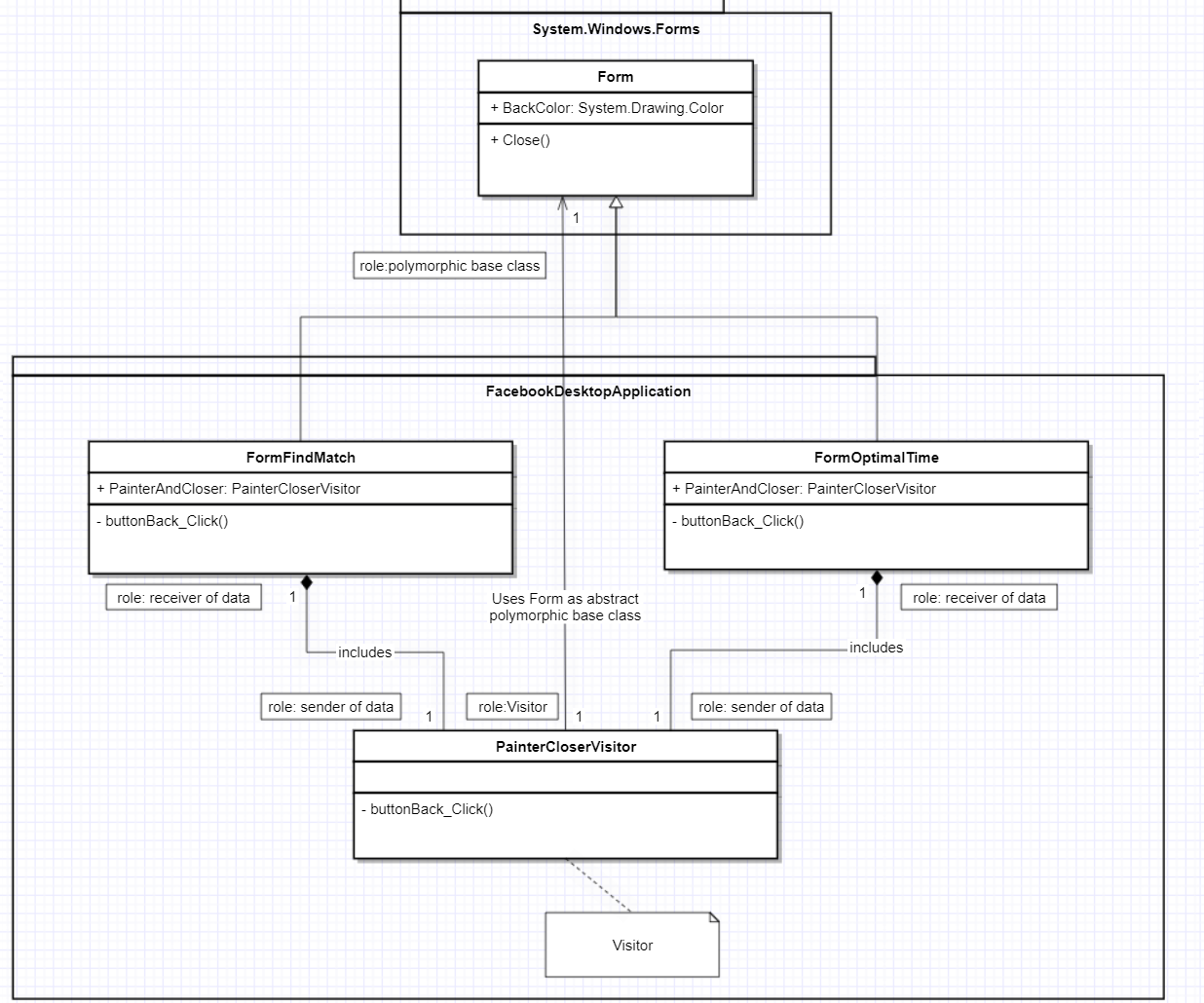
המחלקה PainterCloserVisitor אחראית על עדכון שירות הצביעה וסגירת הטפסים הקונקרטים באפליקציה דרך היכרות עם עם האב האבסטרקטי באופן ישיר, כלומר היכרות ישירה עם Form, ולא ע"י היכרות ישירה עם המחלקה FormFindMatch והמחלקה FormOptimalTime.

* Sequence Diagram





* Class Diagram



**Visitor:**

PainterCloserVisitor

**Polymorphic Abstract Class:**

Form

**Concrete Classes**:

FormFindMatch

FormOptimalTime

### תבנית מס' 3 – Observer

* סיבת הבחירה / שימוש בתבנית:

על מנת לממש את מערכת היחסים של מודיע-מאזין מבלי שהמודיע, ה-Observable (מדובר על המחלקה LogoutManager) יכיר את המאזין שלו, ה-Observer (מדובר על המחלקה FormMain) באופן ישיר ובכל זאת ירצה להודיע לו דברים, נגדיר ממשק, ה-IObserver (מדובר במחלקה ILogoutObserver) ובתוכו מתודה שאחראית על עדכון המאזין\ים לפי תוכן מסויים (המתודה Update).

המודיע יחזיק אוסף של ממשקים כאלה ויאפשר אם נרצה בעתיד - הצטרפות של מאזין לאוסף, הסרה של מאזין מהאוסף והודעה לאוסף המאזינים כשירצה להודיע תוכן קונקרטי.

\*\*\*ב-C# נממש את זה בתור Delegate כי כך הוא נותן מימוש טוב יותר ל-Pattern הזה, הופך אותו ליותר קריא, יותר reusable, יותר תחזוקתי ויותר encapsulated (בזכות ה-public event).

* אופן המימוש:

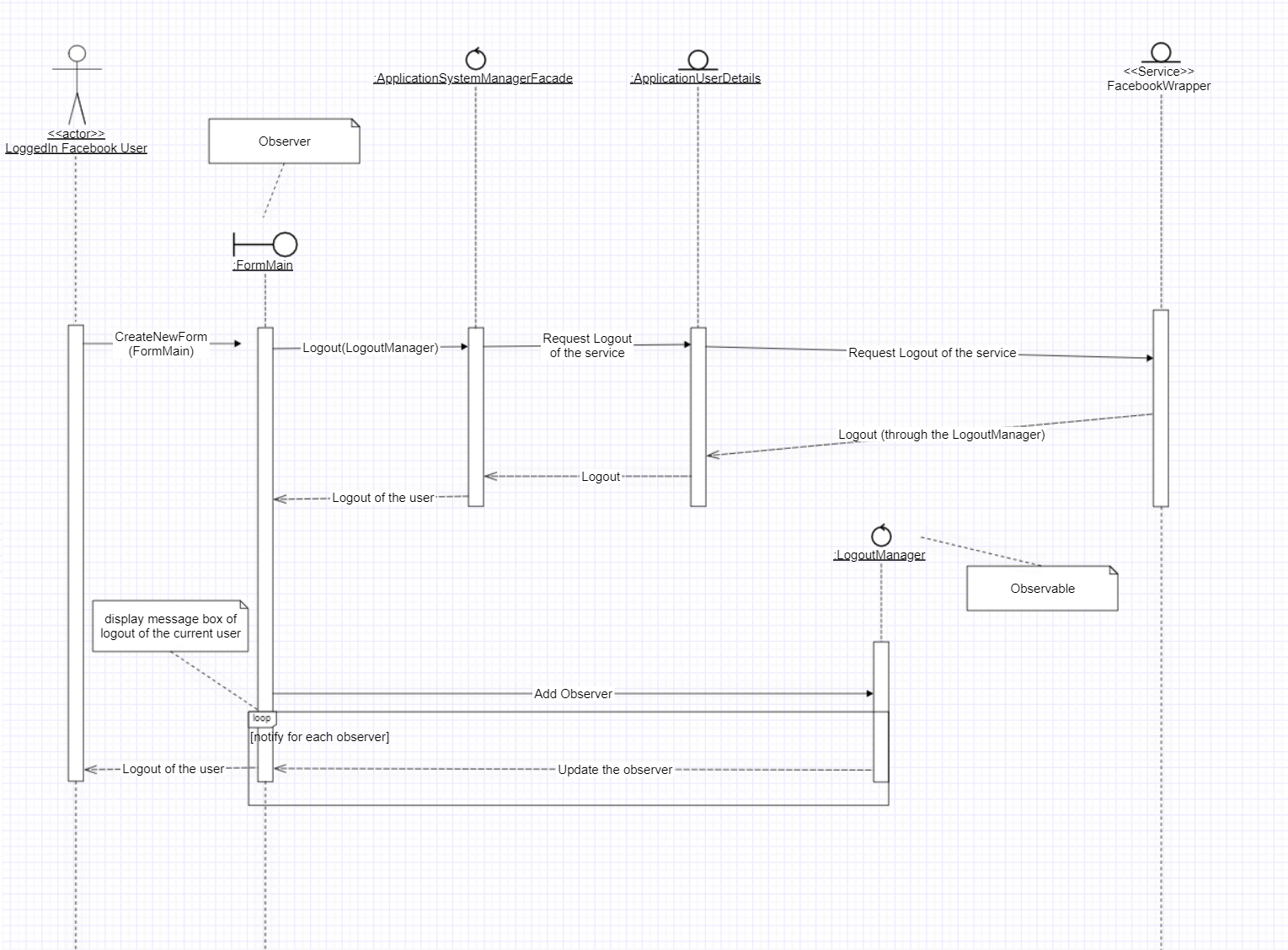
במחלקה ApplicationUserDetails ישנה מתודה Logout שמפעילה את מתודה ה-Logout של FacebookService, והיא המתודה המקשרת את תחילת הליך ההתנתקות בין המודיע למאזינים באפליקציה.

המחלקה ILogoutObserver מכילה את המתודה Update שמספקת שירות עדכון לכל המאזינים בכך ששולחת את שמו של ה-LoginUser הנוכחי שהתנתק מהאפליקציה.

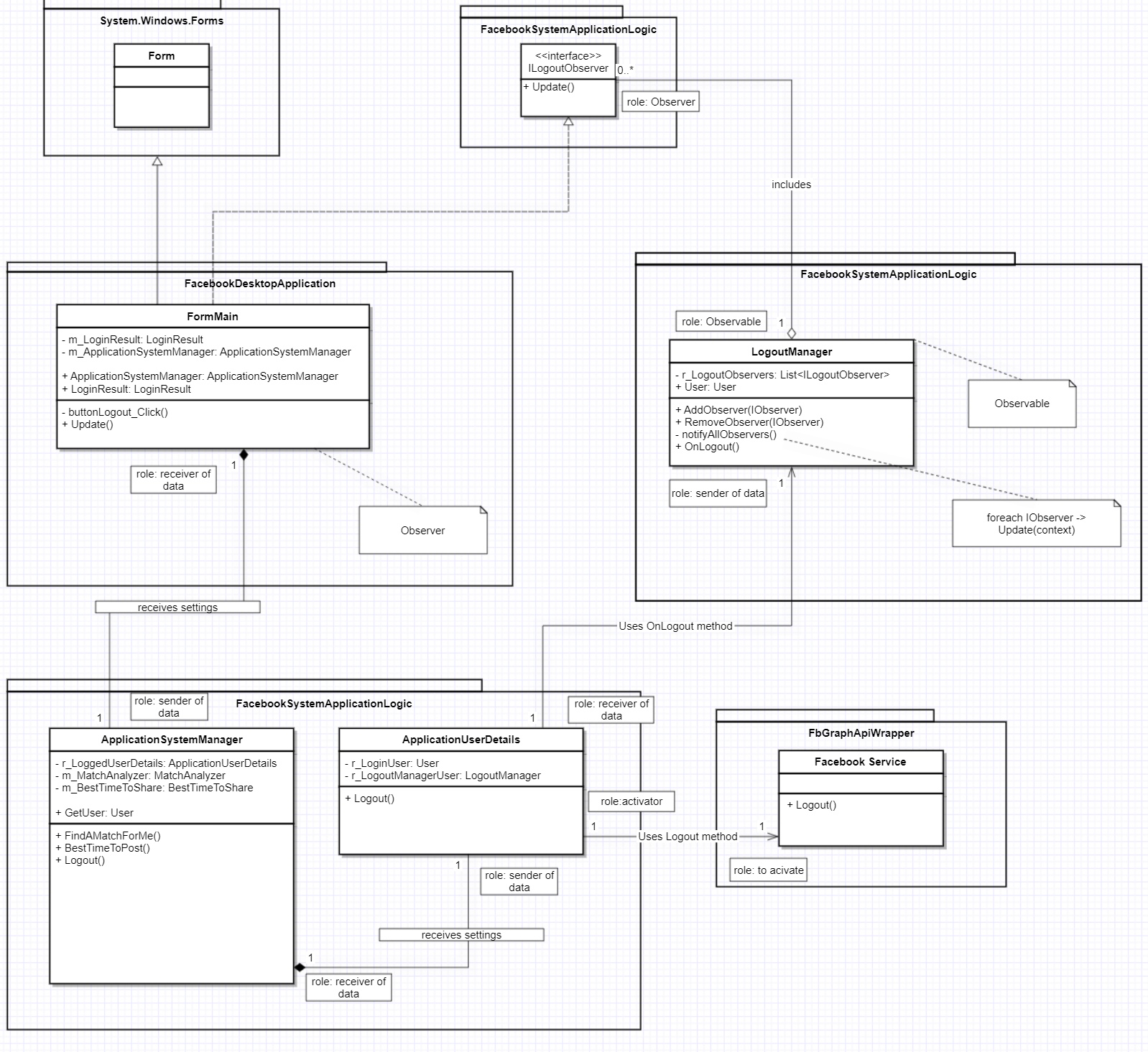
המחלקה LogoutManager אחראית על הודעת התנתקות של ה-LoginUser מהאפליקציה לכל המאזינים הרלוונטים. בנוסף מתעדכנת בעת הוספת מאזינים לאוסף, הסרת מאזינים מהאוסף.

המחלקה FormMain אחראית על האזנת ההתנתקות של ה-LoginUser מהאפליקציה ווידוא שההתנתקות התבצעה בהצלחה באמצעות MessageBox שמוצג בעת ההתנתקות של המשתמש.

* Sequence Diagram



* Class Diagram





**Observer:**

FormMain

**Polymorphic Interface:**

ILogoutObserver

**Observable:**

LogoutManager