"FbWindowAppWithDesignPatterns" היא אפליקציה חלונאית שכתובה ב- #C, משתמשת ב- API של Facebook ומדגימה שימוש ב- Design Patterns שונים

תיאור קצר של הפיצ'רים שבחרנו לממש באפליקציה:

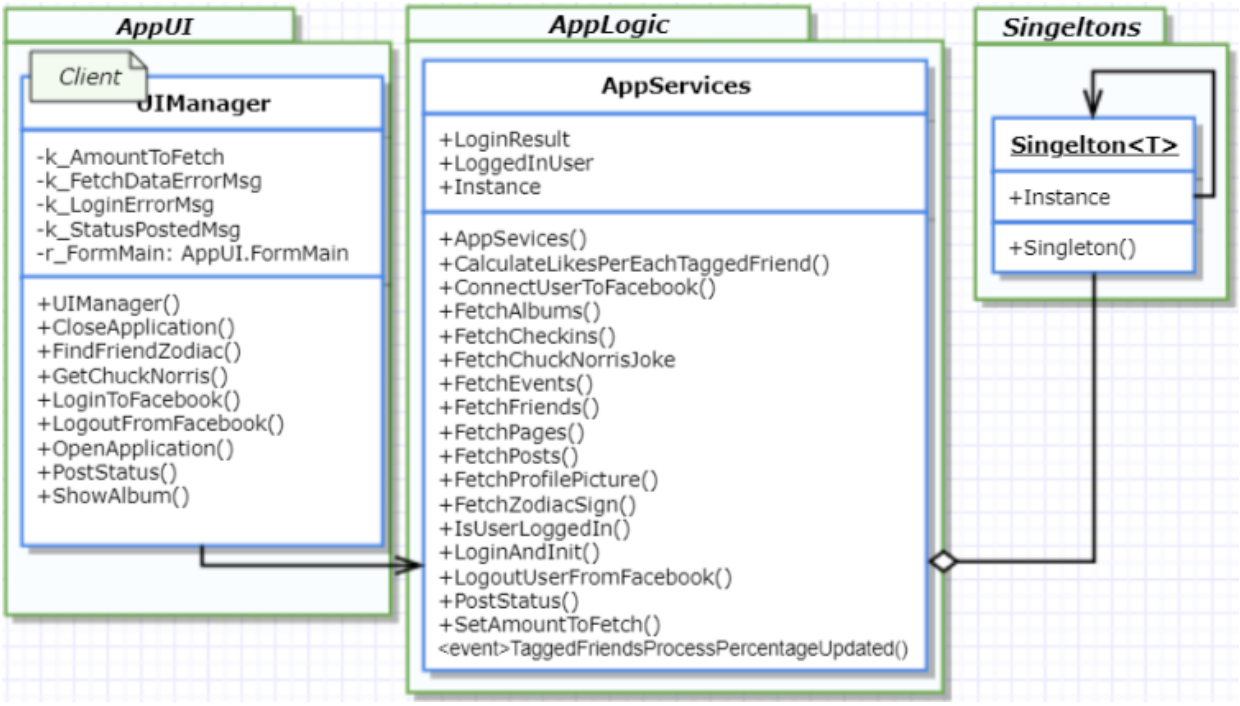
- <u>מציאת המזל (מתוך גלגל המזלות=זודיאק) של אחד החברים –</u> על המשתמש לבחור את אחד מחבריו ברשימת החברים וללחוץ על הכפתור המיועד עבור מציאת ה-*Zodiac*. האפליקציה תקפיץ חלון עם התוצאה של המזל, וכמו כן תציג תמונה מייצגת של מזל זה.
- <u>פרסום בדיחת צ'אק נוריס לאחד החברים –</u> על המשתמש לבחור את אחד מחבריו ברשימת החברים וללחוץ על הכפתור המיועד עבור שליפת בדיחת צ'אק נוריס מתוך מאגר הנמצא ברשת וניתן לשליפה ב-JSON. האפליקציה תקפיץ חלון עם הבדיחה, ותאפשר למשתמש לבחור האם לפרסם את הבדיחה כפוסט חדש עם תיוג של החבר המסומן.
- <u>צפייה באלבם תמונות –</u> על המשתמש לבחור את אחד מבין אלבומיו ברשימת האלבומים וללחוץ על הכפתור המיועד עבור פתיחת טופס מותאם המציג את כל התמונות שבאלבום בעזרת ממשק ידידותי.
- <u>מציאת החבר שהכי כדאי להעלות תמונה איתו –</u> רמת ה"כדאיות" נמדדת על פי לייקים. לדוגמא אם העלתי תמונה עם חבר א' שזכתה ל-100 לייקים ותמונה עם חבר ב' שזכתה ל-50 לייקים, אזי יותר "כדאי" לי להעלות תמונות עם חבר א' כדי לזכות בלייקים. הפיצ'ר מציג את רשימת החברים המתוייגים בתמונות איתנו וכמה לייקים גרף כל אחד מהם בסה"כ.

Design Patterns שבהם השתמשנו באפליקציה:

תבנית מס' 1 – [Singleton]

- <u>סיבת בחירה/ שימוש בתבנית:</u>
השתמשנו בתבנית זו פעמיים. פעם אחת עבור *"AppSettings"*, ופעם נוספת עבור *"AppServices"* (המחלקה שמנהלת את הלוגיקה). היה צורך בלוגיקה סינגלטונית ביוון שאין ברצוננו לאפשר לממשק להשתמש ביותר מלוגיקה אחת לאפליקציה, וכמו כן בזכות הסינגלטוניות היה ניתן לגשת אליה (לאותו מופע שלה) דרך כל המחלקות והטפסים השונים בממשק המשתמש, וזאת מבלי להעביר את הלוגיקה כפרמטר מאחד לשני. אם בעתיד נרצה להוסיף למערכת טפסים חדשים אז גם הם יוכלו לגשת ישירות וביוזמתם ללוגיקה.
- <u>אופן המימוש:</u>
לטובת מימוש הסינגלטון של המחלקה *"AppServices"*, השתמשנו במחלקה טמפלטית *"Singleton<T>"*.
במחלקה *"AppServices"*, תחת ה-*property "Instance"* יש החזרה של מופע היחיד של הלוגיקה.
השחקן היחיד במימוש תבנית זו היא המחלקה *"AppServices"* הסינגלטונית.

- <u>UML diagram:</u>



תבנית מס' 2 – [Adapter]

- **סיבת הבחירה / שימוש בתבנית:**

בחרנו בתבנית זו כיוון שהיה צורך לבצע תיאום בין האובייקטים הפייסבוקיים של הלוגיקה לבין האופן שבו הממשק משתמש מעוניין להציג אותם. האובייקט הפייסבוקי *"Post"* בפרט הוא זה שהעלה את הסוגיה על הפרק. למרבית האובייקטים הפייסבוקיים ניתן לגשת לשדה *"Name"* שלהם ולקבל את השם שמתאר את האובייקט, אך לעיתים (למשל באובייקט ה-*"Post"*), השדה הנ"ל לא מציין את התיאור של האובייקט ולכן עלינו לבצע מניפולציות מסוימות כדי להפיק תיאור קוהרנטי של האובייקט. בשלב הבא של ניתוח הבעיה התמקדנו בשאלה האם לממש *"Adapter"* או *"Proxy"*. האינסטינקט הראשוני היה ללכת על *"Proxy"* עבור אובייקט פייסבוקי ולממש בו את המתודה *"ToString"* כך שתבצע את המניפולציות שהסברנו קודם לכן. אך תוך כדי עבודה נחשפנו לבעיות שזה מעורר בממשק המשתמש: ה-*Data Binding* שביצענו לא יכל לנצל את מלוא הפוטנציאל כיוון שהיה צורך בכל פעם לשדך באופן ידני בין הפקד שמייצג את השם של האובייקט הפייסבוקי לבין המתודה *"ToString"* שלו, זאת כמובן בשונה לשאר ה-*properties* של האובייקט שמתממשקים עם ה-*Data Source* באופן אוטומטי. ניתן היה גם לשדך באופן ידני רק את האובייקטים הפייסבוקיים "הבעייתיים" ולתת לשאר האובייקטים הפייסבוקים להתממשק באופן מלא עם ה-*Data Source* שלהם. כך או כך, המימוש שובר את האחידות שבין האובייקטים הפייסבוקיים ומקשה בהתאמה על הקריאות. לפיכך גמלה בליבנו ההחלטה לא "לשכלל" דבר קיים באובייקט הפייסבוקי, אלא להוסיף לו שדה *property "DisplayName"* שתפקידו הבלעדי הוא להיות שם האובייקט שיוצג בממשק המשתמש. כלומר המעטפת שיצרנו לאובייקט הפייסבוקי היא זו שבזכותה ניתן לתאם בין האובייקט לבין הדרישה של הממשק להצגתו. כמו כן, בזכות בחירה של תבנית עיצוב זו, הצלחנו לממשק את כל ה-*Data Sources* עם הפקדים שמתאימים להם בצורה אחידה. אם בעתיד נרצה לצפות באובייקט פייסבוקי נוסף בממשק משתמש, הוא יצטרך רק לממש את אותו ממשק של *"AdapterFacebookObject"* וכך הממשק משתמש ידע איך להציג אותו.

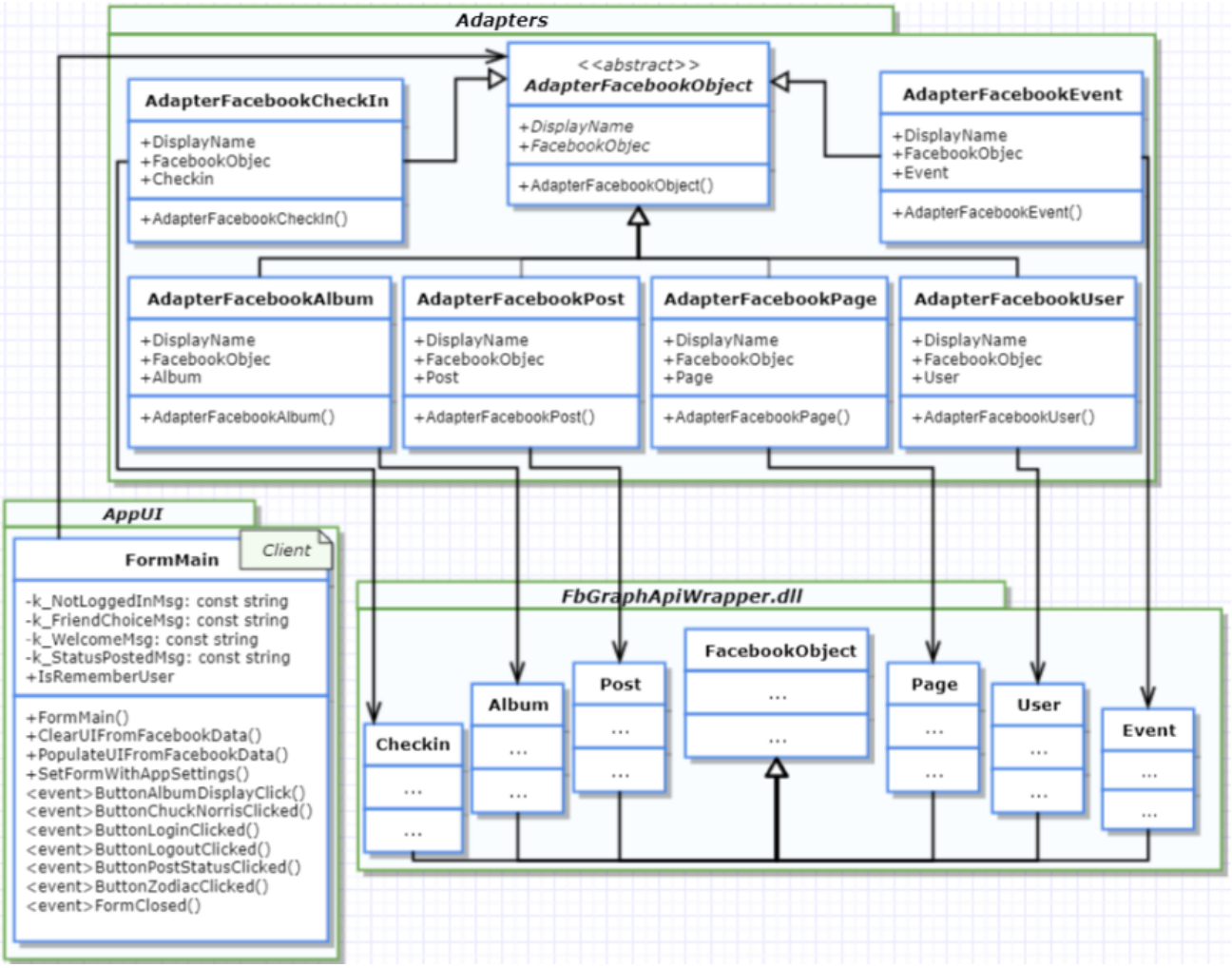
השחקנים במימוש תבנית הם:

Client => Class "FormMain"
Target Class => Class "AdapterFacebookObject"
Adaptee => Class "FacebookObject"
Adapter => Classes: "AdapterPost", "AdapterPage", "AdapterUser", "AdapterEvent", "AdapterAlbum", "AdapterCheckin"

- **אופן המימוש:**

יצרנו פרוייקט חדש שתחתיו מימשנו את כל מחלקות ה-Adapter למיניהן, כל אובייקט פייסבוקי קיבל מחלקה משלו וירש מן המחלקה האבסטרקטית *"AdapterFacebookObject"*. הלוגיקה *"AppServices"* יודעת כיצד לשאוב נתונים מן המשתמש המחובר למערכת, ולכן בכל הפעלה של מתודת *Fetch* כלשהי (למשל *"FetchAlbums"*, *"FetchPosts"*) תדע הלוגיקה לשאוב את הרשימה המתאימה וליצור מכל אובייקט פייסבוקי את האדפטר המותאם לו. בסופו של דבר תחזיר הלוגיקה רשימה של אדפטרים לממשק המשתמש (ספציפית לטופס *"FormMain"* שקורא למתודות ה-*Fetch*) והוא כבר ידע בעזרת שימוש ב-Data Binding כיצד לשדך בין האדפטר לבין הפקדים השייכים לו.

- UML diagram:


Adapters
<<abstract>>
AdapterFacebookObject
+DisplayName
+FacebookObjec
+AdapterFacebookObject()
AdapterFacebookCheckIn
+DisplayName
+FacebookObjec
+Checkin
+AdapterFacebookCheckIn()
AdapterFacebookEvent
+DisplayName
+FacebookObjec
+Event
+AdapterFacebookEvent()
AdapterFacebookAlbum
+DisplayName
+FacebookObjec
+Album
+AdapterFacebookAlbum()
AdapterFacebookPost
+DisplayName
+FacebookObjec
+Post
+AdapterFacebookPost()
AdapterFacebookPage
+DisplayName
+FacebookObjec
+Page
+AdapterFacebookPage()
AdapterFacebookUser
+DisplayName
+FacebookObjec
+User
+AdapterFacebookUser()
AppUI
Client
FormMain
-k_NotLoggedInMsg: const string
-k_FriendChoiceMsg: const string
-k_WelcomeMsg: const string
-k_StatusPostedMsg: const string
+IsRememberUser
+FormMain()
+ClearUIFromFacebookData()
+PopulateUIFromFacebookData()
+SetFormWithAppSettings()
<event>ButtonAlbumDisplayClick()
<event>ButtonChuckNorrisClicked()
<event>ButtonLoginClicked()
<event>ButtonLogoutClicked()
<event>ButtonPostStatusClicked()
<event>ButtonZodiacClicked()
<event>FormClosed()
FbGraphApiWrapper.dll
Checkin
Album
Post
FacebookObject
Page
User
Event

תבנית מס' 3 – [Factory Method]

- **סיבת הבחירה / שימוש בתבנית:**

מאחר והיו לנו אדפטרים שכולם ירשו מן המחלקה האבסטרקטית *"AdapterFacebookObject"*, אז יצרנו מחלקה סטטית *"FactoryOfFacebookObjectAdapters"* שכל תפקידה הוא להיות אחראית על יצירת האדפטר המתאים ובכך לנצל את הפולימורפיזם בין האדפטרים השונים. בעזרת שימוש בתבנית עיצוב זו אנו חוסכים מן הלוגיקה ביצוע של פעולות new ליצירת אדפטר, ומונעים תיקונים שעלולים להתבצע בעתיד בקוד במידה ודרך יצירת האובייקטים השתנתה (למשל- נוספו פרמטרים חדשים בקונסטרקטור לאדפטר). אם בעתיד נרצה להוסיף אדפטר לאובייקט פייסבוקי נוסף אז נוכל פשוט לעדכן את תהליך היצירה שלו ב- *Factory* ולהשתמש בזה כאוות נפשנו בלוגיקה.

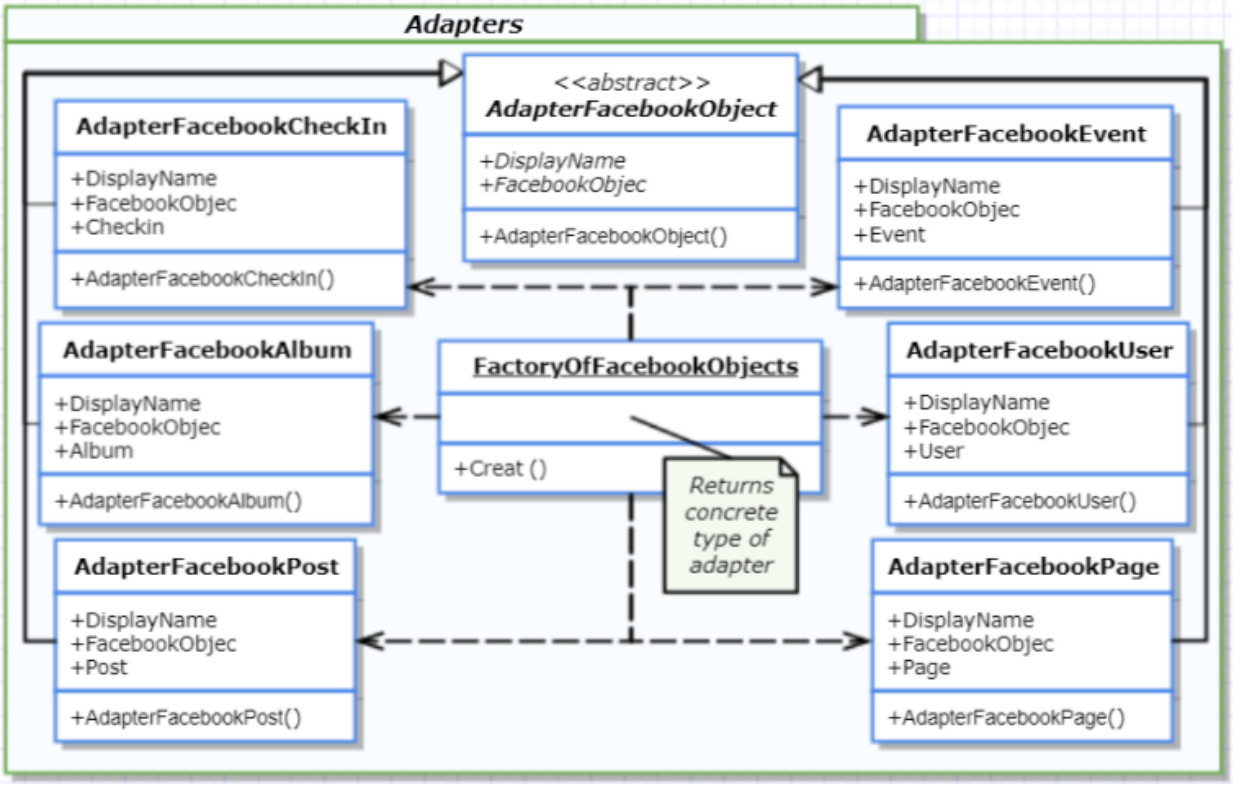
השחקנים במימוש הם:

Product => Class "AdapterFacebookObject"
Creator => Class "FactoryFacebookObjectAdapter"
ConcreteProduct => Classes: "AdapterPost", "AdapterPage", "AdapterUser", "AdapterEvent", "AdapterAlbum", "AdapterCheckin"

- **אופן המימוש:**

תחת הפרוייקט שמאגד בתוכו את האדפטרים יצרנו מחלקה סטטית שנקראת *"FactoryOfFacebookObjectAdapters"* ובה מתודה סטטית אחת *"Create"* לטובת יצירה של האדפטר. המתודה מקבלת אובייקט פייסבוקי כלשהו ולפי הסוג שלו יודעת לאבחן איזה אדפטר צריך ליצור ולהחזיר.

- **UML diagram:**



תבנית מס' 4 – [Proxy]

- **סיבת הבחירה / שימוש בתבנית:**

לטובת מימוש הפיצ'ר של צפייה באלבום תמונות רצינו לטעון תמונות שיטענו לטופס בצורה "עצלנית" (כלומר שיטענו רק כשיש צורך בהצגתן), ושכללנו אותן עוד יותר על ידי מעבר של סמן העכבר על גבי התמונה. ברגע שסמן העכבר עובר על תמונה היא גדלה מעט וממוסגרת בצבע כחול כדי שיהיה ברור למשתמש על איזו תמונה הוא מסתכל, כשסמן העכבר לא עליה היא חוזרת לצורתה המקורית. כפי שניתן לראות מדובר בשכלול של האובייקט *"PictureBox"* על בסיס היכולות הקיימות אצלו, כך שממשק המשתמש מתייחס לפרוקסי ול-*"PictureBox"* בצורה זהה.

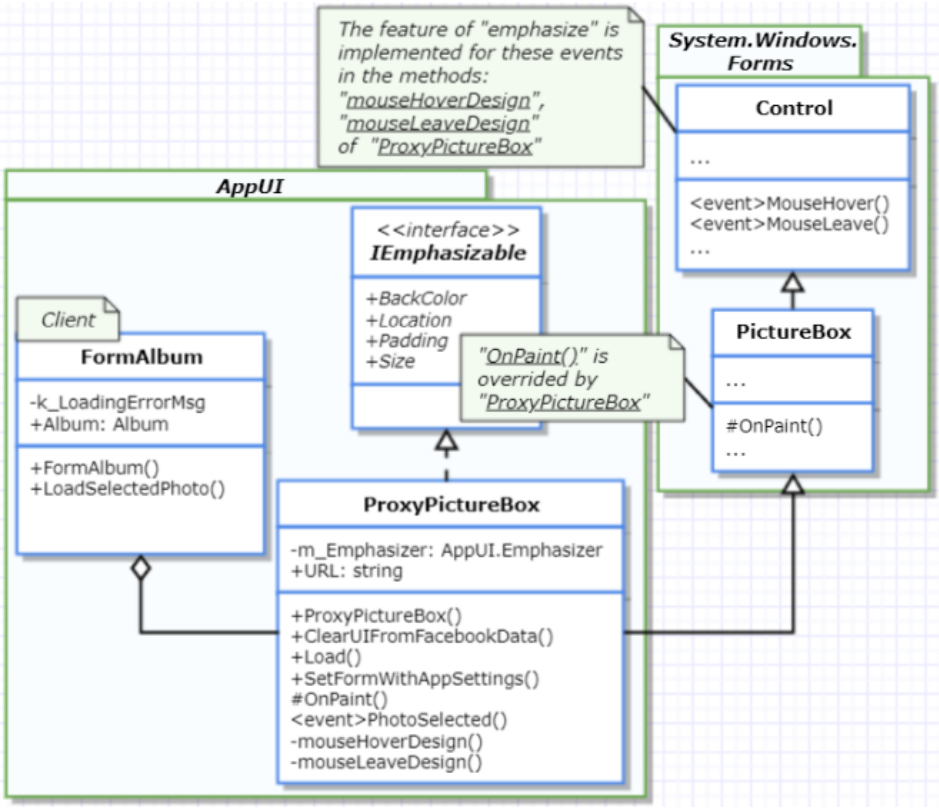
השחקנים במימוש תבנית הם:

Client => WinForm "FormAlbumDisplay"
Proxy => Class "ProxyPictureBox"
Subject => Class "PictureBox"

- **אופן המימוש:**

יצרנו מחלקה *"ProxyPictureBox"* שיורשת מהמחלקה *"PictureBox"* ובכך יורשת את השירותים הבסיסיים שלה. בתוך מחלקה זו דרסנו את המימוש עבור *"OnPaint"* כדי לאפשר את תכונת העצלנות של התמונה. כמו כן שידכנו בין אירועים שנוגעים למעבר העכבר על גבי התמונה עם המתודות החדשות *"mouseHoverDesign"*, *"mouseLeaveDesign"* שאחראיות לעיצוב התמונה. לבסוף, השתמשנו בפרוקסי במחלקה *"FormAlbumDisplay"* כדי להציג את התמונות שבתוך האלבום, יחד עם התכונות החדשות שמימשנו.

- **UML diagram:**



תבנית מס' 5 – [Observer]

- **סיבת הבחירה / שימוש בתבנית:**

השתמשנו בתבנית זו על מנת לתאר את הקשר שבין הטופס הראשי *"FormMain"* לבין המחלקה *"UIManager"* שתפקידה לנהל הקשרים עם הלוגיקה *"AppServices"*, *"AppSettings"* וטפסים נוספים. כאשר מצב כלשהו משתנה בטופס הראשי, אז ה-*"UIManager"* צריך לדעת על כך ולפעול בהתאם.

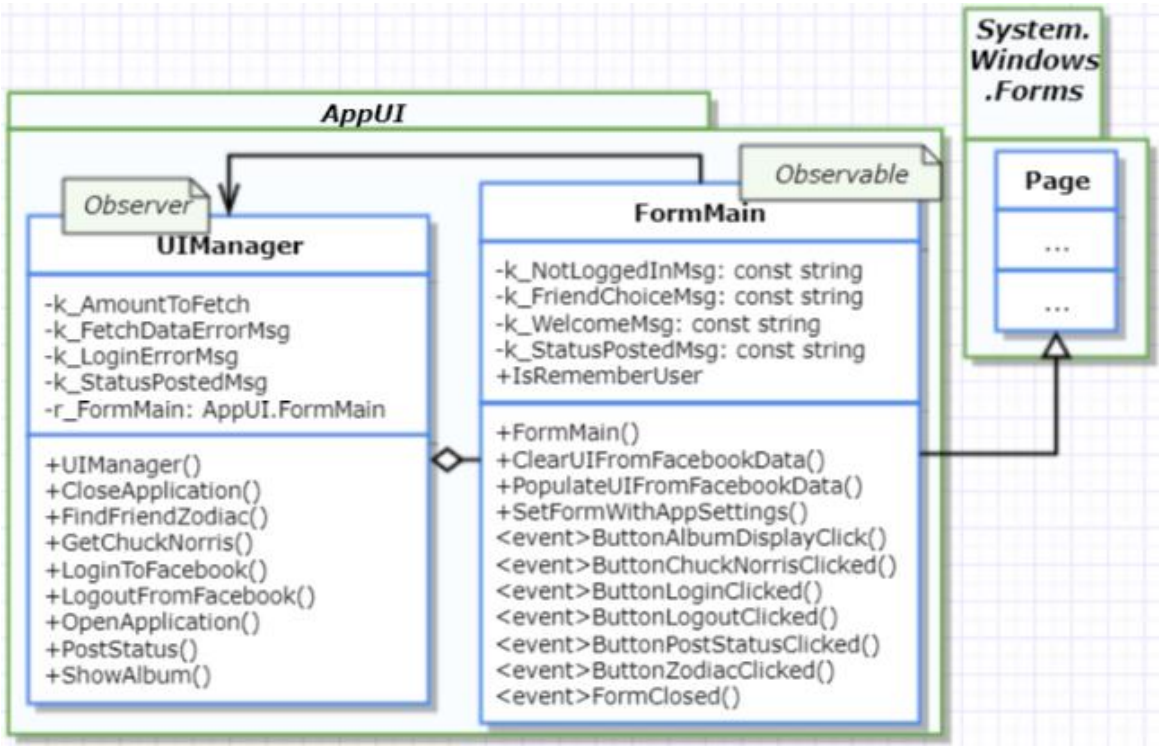
השחקנים במימוש תבנית הם:

Observable => WinForm "FormMain"
Observer => Class "UIManager"

- **אופן המימוש:**

ב-*"FormMain"* הגדרנו אירוע עבור כל תרחיש שסביר שיהיו אליו מאזינים נוספים (במקרה שלנו המאזין היחיד – *"UIManager"*). המאזין בעת יצירתו מגדיר שהוא מאזין לתרחיש מסוים אצל *"FormMain"*, וברגע ש- *"FormMain"* מעורר תרחיש זה אז המאזין יודע לפעול בהתאם. למשל, ע"י לחיצה של המשתמש על כפתור ה-*"Logout"* יש להתנתק מהפייסבוק ולשנות את מראית הטופס. כל מה שקשור למראית הטופס אז הטופס יודע לנהל את עצמו ולעצב את עצמו, אך עבור פעולת הניתוק הוא יעורר תרחיש שנקרא *"ButtonLogoutClick"*. המאזין היחיד שלנו *"UIManager"* מאזין לתרחיש זה, וברגע שיתעורר אז יבצע את פעולת הניתוק מהפייסבוק.
שחקנים:

- **UML diagram:**



תבנית מס' 6 – [Iterator]

- **סיבת הבחירה / שימוש בתבנית:**

הלוגיקה שלנו מממשת את הפיצ'ר עבור מציאת החבר שהכי כדאי להעלות תמונה איתו, והיא עושה זאת על ידי שימוש במבנה נתונים מסוג *Dictionary* לטובת יעילות. אין אנו רוצים שממשק המשתמש ידע שאנו משתמשים במבנה נתונים זה ויסתמך על עובדה זו, כיוון שאולי בעתיד נרצה להחליף את מבנה הנתונים למבנה אחר, מבלי שמשק המשתמש יהיה תלוי בכך. לכן היה עלינו להגדיר מבנה נתונים מיוחד שיחזיק את החברים המתוייגים, ולממש לו איטרטור, כך שממשק המשתמש יוכל לסרוק על פני מבנה הנתונים מבלי לדעת באיזה אופן האובייקטים בו מוחזקים.

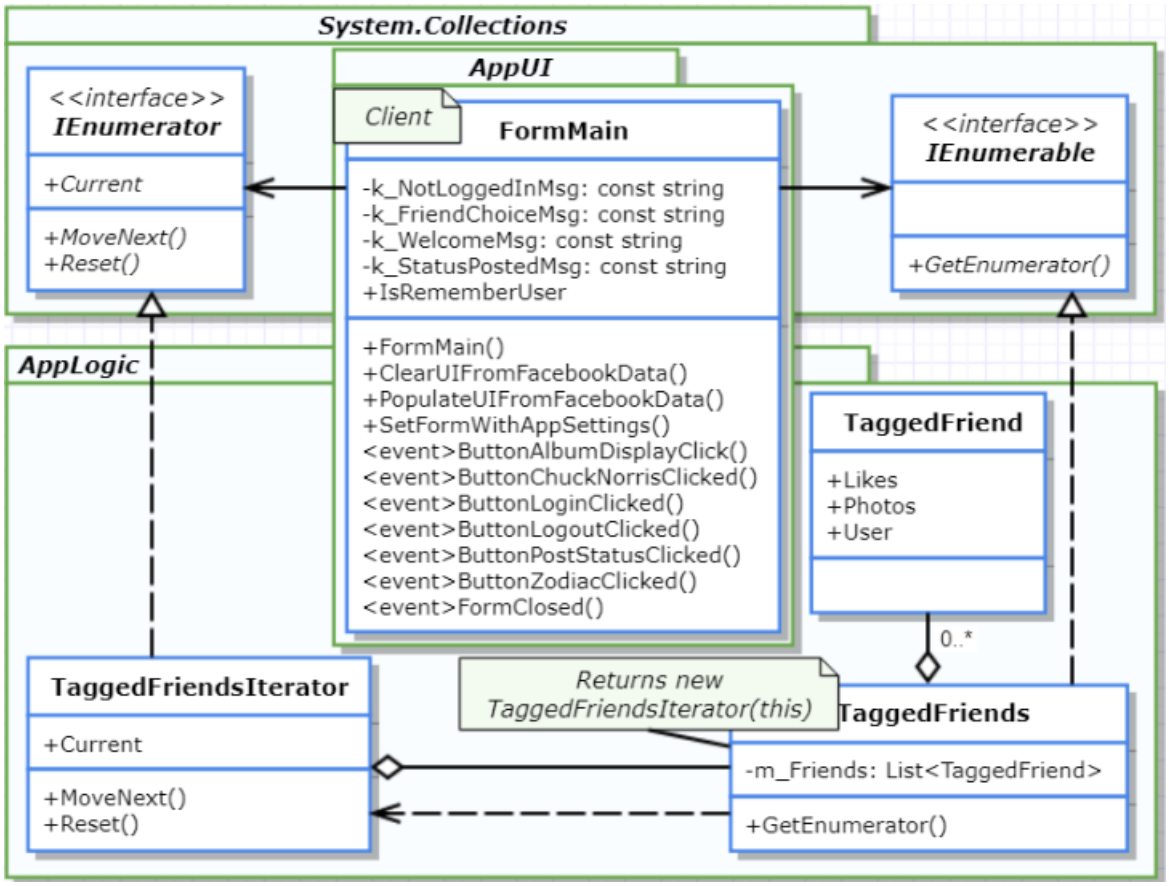
השחקנים במימוש תבנית הם:

Aggregate => IEnumerable
ConcreteAggregate => Class "TaggedFriends"
Iterator type => IEnumerator
ConcreteIterator => Inner Class "TaggedFriendsIterator"

- **אופן המימוש:**

יצרנו מחלקה חדשה בלוגיקה בשם *"TaggedFriends"* שהיא האובייקט המייצג קולקציה של אובייקטים מסוג *"TaggedFriend"*. במחלקה פרטית בתוכה יצרנו איטרטור בשם *"TaggedFriendsIterator"* בעזרתו יהיה ניתן לסרוק את האיברים בקולקציה. ממשק המשתמש (בלומר *"FormMain"*) מקבל מהלוגיקה כאובייקט את *"TaggedFriends"*, ולאחר מכן עושה בזה שימוש בעזרת ספריית *System.LINQ* כדי למיין את האובייקטים בקולקציה, כל זאת מבלי לדעת משהו מבנה הנתונים שבו נשמרים האובייקטים.

- **UML diagram:**



תבנית מס' 7 – [Visitor]

- **סיבת הבחירה / שימוש בתבנית:**

השתמשנו בתבנית זאת על מנת למנוע שכפול קוד של תכונת "הדגשה" (Emphasized). רצינו שכמה אובייקטים (תמונה וכפתור) יהיו בעלי יכולת זו אבל אם היינו מממשים את היכולת בכל אחת ממחלקות האובייקטים (למעשה במחלקות הפרוקסי שלהם) אז היה לנו שכפול קוד. כך הפכנו את הקוד ליותר reusable ו-maintainable.

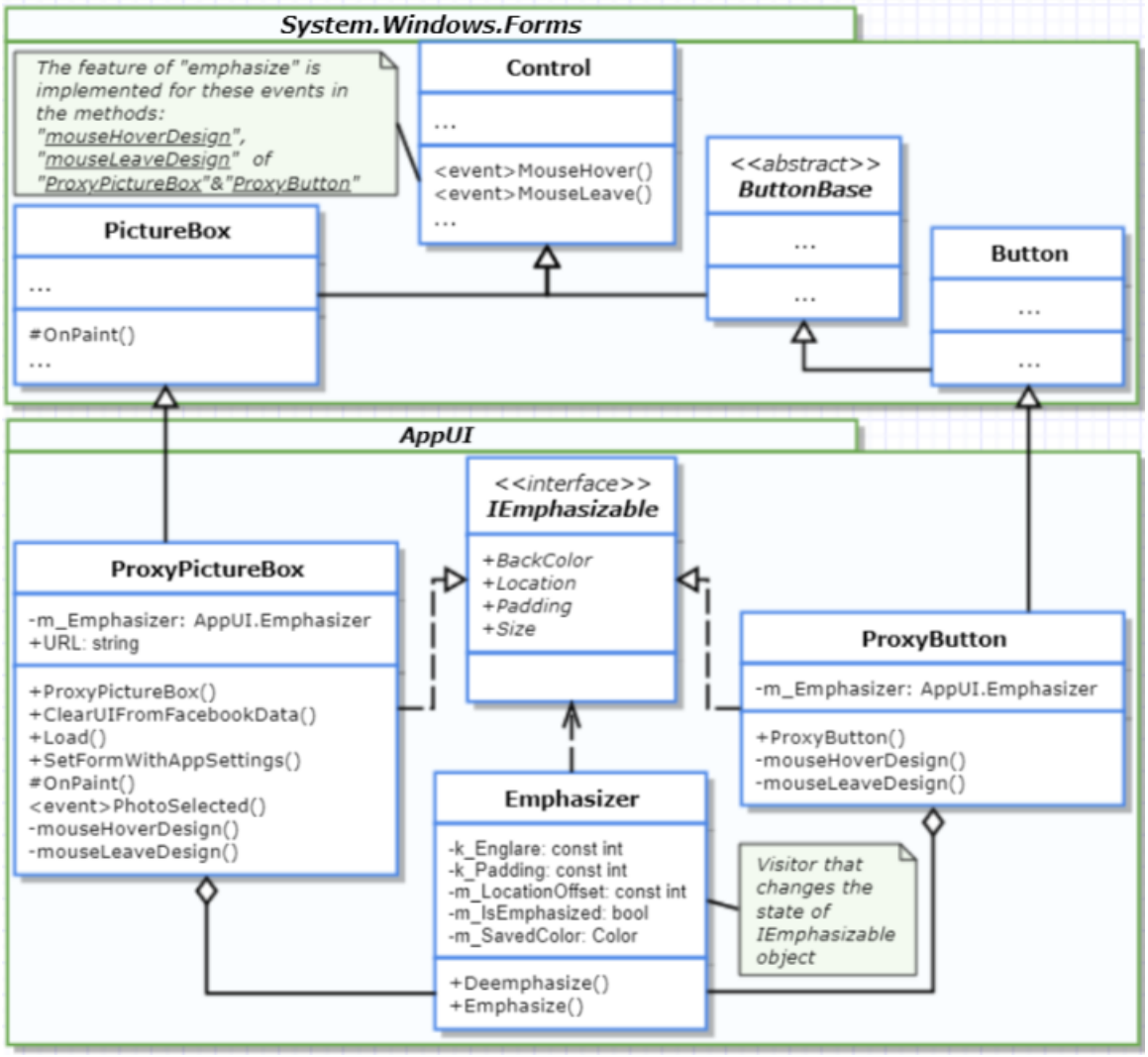
השחקנים במימוש תבנית הם:

Client => WinForm "FormMain"
Element => Classes: "ProxyPictureBox", "ProxyButton"
Visitor => Class "Emphasizer"
IVisitor => Interface "IEmphasizable"

- **אופן המימוש:**

היה עלינו לייצא את תכונת ההדגשה למחלקה נפרדת, וכמו כן יצרנו ממשק חדש *"IEmphasized"* אשר מי שמעוניין להיות בעל יכולת זו יהיה עלינו לממש את הממשק. בהמשך, הבטחנו שכל אחת ממחלקות הפרוקסי של תמונה וכפתור (*"ProxyPictureBox"*, *"ProxyButton"*) יחזיקו באגרגציה את הממשק של התכונה שהוא *"Emphasiser"*, דרכו ניתן להפעיל את התכונה לפי הצורך.

- **UML diagram:**



עבודה אסיכרונית ע"י שימוש ב- Thread-ים:

<u>שימוש בתכנות אסיכרוני נעשה בשני מקומות:</u>

1) במחלקה "*FormMain*"במתודה "*PopulateUIFromFacebookData*" יצרנו טעינה אסיכרונית של כל אחד מסוגי האובייקטים הנטענים לאפליקציה. כלומר מתבצעת טעינה אסיכרונית של פוסטים, חברים, אלבומים, עמודים ואירועים. היתרון הוא שבעת טעינת האפליקציה אין צורך לחכות עד לטעינת כל הנתונים על מנת להשתמש בה, כך המשתמש גם לא חווה תחושה כי האפליקציה נתקעה מפאת הטעינה הארוכה, וגם הוא כבר יכול להשתמש בנתונים שנטענו כאשר במקביל נטענים נתונים נוספים לאפליקציה.
2) במחלקה "*FormMain*"במתודה "*buttonFindBestTaggedFriend_Click*" הפונקציה "*showTaggedFriendsResults*" תתבצע באופן אסיכרוני. גם הפעם היה ידוע לנו כי החישוב של פיצ'ר זה עלול לקחת זמן רב, ולכן במקום לתקוע את האפליקציה היה עדיף לתת לו לפעול ב-thread נפרד, כך שהמשתמש בזמן הזה יכול להמשיך להשתמש באפליקציה עד שהנתונים יטענו.

עבודה עם Data Binding:

עשינו שימוש ב-*Data Binding* בכל מקום בממשק משתמש של האפליקציה שבו יש לנו רשימת פריטים. לדוגמא, "*listBoxPosts*" זה הרשימה שמציגה את הפוסטים (למען הדיוק, את האדפטרים של הפוסטים) של המשתמש. כאשר המשתמש יבחר פוסט מתוך הרשימה, הפקדים שרשומים ל-*Data Source* שהוא "*AdapterPost*" של הפוסט, ישאבו ממנו את הנתונים הרלוונטים ויציגו אותם. כמו כן, מסיבה טכנית לא ברורה לא היה ניתן לשאוב ישירות מתוך ה-"*AdapterPost*" את הנתונים אודות ה-*Post*, אלא רק את הנתון של ה-"*DisplayName*", ולכן היה עלינו לדאוג שבעת בחירת הפוסט מתוך הרשימה יתבצע גם שידוך ספציפי בין האובייקט *Post* לבין ה-*DataSource* שלו "*postDataSource*", על מנת שהפקדים יוכלו להציג את הנתונים מתוך ה-*Post* (כלומר כל הנתונים שאינם "*DisplayName*"). כל התיאור הנ"ל ביצענו עבור כל אחת מהרשימות.