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

* מאפשר למשתמש להודיע לחברי הפייסבוק שלו בעזרת פוסט בכל פעם שהוא מתחיל לשדר באתר Twitch.tv
* מאפשר למשתמש להגיב בצורה גורפת להזמנות אירועים שקיבל בפייסבוק.

### תבנית מס' 1 – Proxy

* סיבת הבחירה / שימוש בתבנית:

המחלקה TwitchAPIWrapper המשמשת לגישה לפונקציונאליות לAPI של Twitch, אמורה לספק רק מידע אודות הנתונים שנאספו, וכן הטופס FormMain אמור רק לבקש אותו ולא להכיר את ההיגיון הרצוי בעקבות המידע שהתקבל.  
המחלקה TwitchForFacebookProxy משמשת כנקודת אמצע המאפשרת למשתמש בטופס לבקש את אותן הבקשות מהAPI, אך מעבירה את ביצוע ההיגיון למחלקה הייעודית לכך.

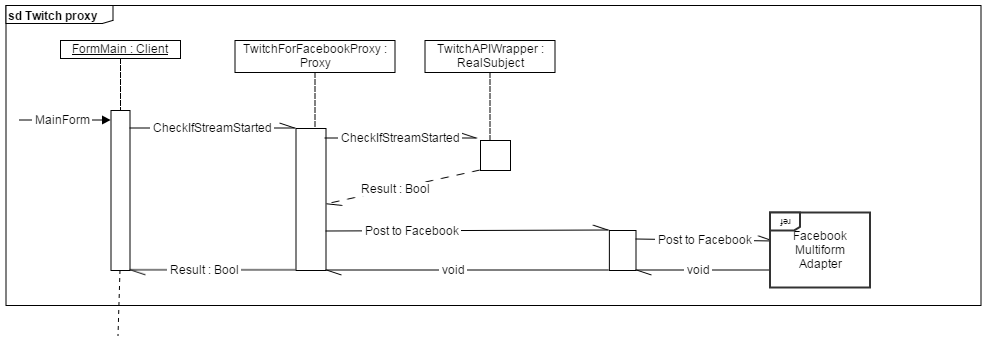
בכך אנחנו תומכים במיינטיינאביליות התכנית.

* אופן המימוש:

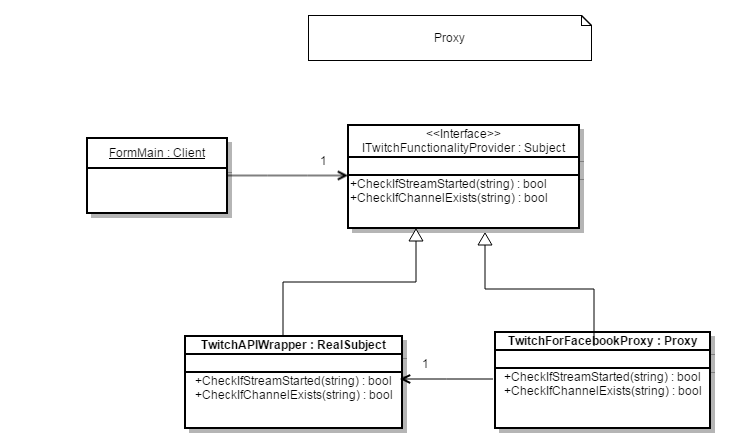
בFormMain בשורה קיימת המתודה AutoPostTwitchUpdate() הנקראת כל זמן קצוב ובודקת אם המשתמש הרצוי התחיל כרגע לשדר.  
שם אנו קוראים למתודה CheckIfStreamStarted(string) של TwitchForFacebookProxy.

המתודה מבצעת הבדיקה דרך הAPI ואז שולחת עדכון לפייסבוק והודעה למשתמש על כך שההודעה נשלחה.

* Sequence Diagram



* Class Diagram



### תבנית מס' 2 – Adapter

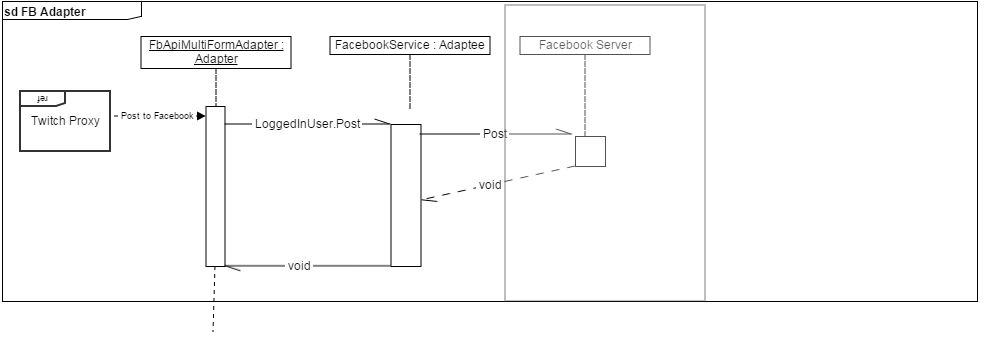
* סיבת הבחירה / שימוש בתבנית:

עלה לנו הצורך לגשת למשתמש המחובר ביותר מטופס אחד(ספציפית טופס שאינו הטופס בו התחברנו למערכת), אך מכיוון שהAPI של הפייסבוק שסופק לנו לא מכיל אפשרות גישה חוזרת למשתמש המחובר, וכדי למנוע מצב שבו צריך להעביר את הreference לאובייקט המשתמש בין הטפסים, היינו צריכים לשנות את דרך הגישה והשימוש במערכת החיבור.  
בכך שבחרנו בגישה זו אנחנו תומכים במיינטיינאביליות התכנית בכך ששומרים דרך גישה יחידה וקונסיסנטית במקום להעביר הפניות וליצור מקומות נוספים לתחזוקה עתידית.

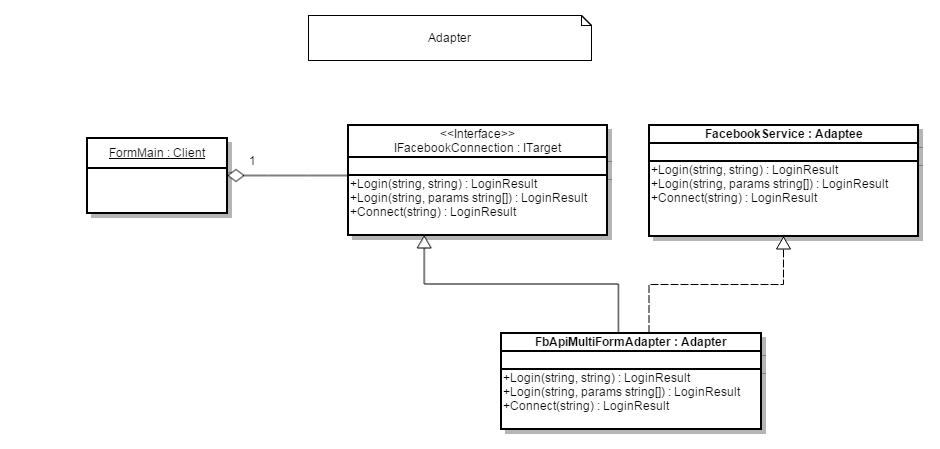
* אופן המימוש:

בנקודות החיבור בקוד(כאשר כפתור הlogin נלחץ והמשתמש מתחבר לפייסבוק), אנו קוראים לפקודות הLogin והConnect של מחלקת הFbApiMultiFormAdapter המממשת את האינטרפייס IFacebookConnection, כל מה שהמחלקה עושה היא לשמור רפרנס של המשתמש המחובר בעקבות חיבור מוצלח, אליו לאחר מכן ניתן לגשת מכל מקום מפני שהמחלקה היא סינגלטונית.

* Sequence Diagram



* Class Diagram



+LoggedInUser : User

### תבנית מס' 3 – Concrete Factory

* סיבת הבחירה / שימוש בתבנית:

כדי לתמוך בסקיילביליות ולאפשר שינוי של אופן השמירה והקריאה(מבחינת מיקום השמירה(כונן, שרת, ענן, כו') וסוג סיריאליזציה(XML, JSON, כו').

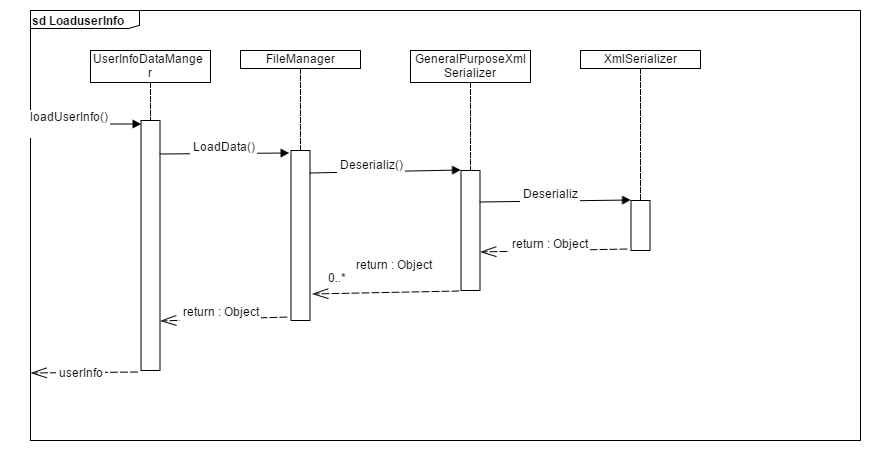
* אופן המימוש:

המחלקה UserInfo מחזיקה מופע של מחלקה UserInfoDataManager שמנהלת את שמירת והעלאת נתונים הרלונטים לUserInfo

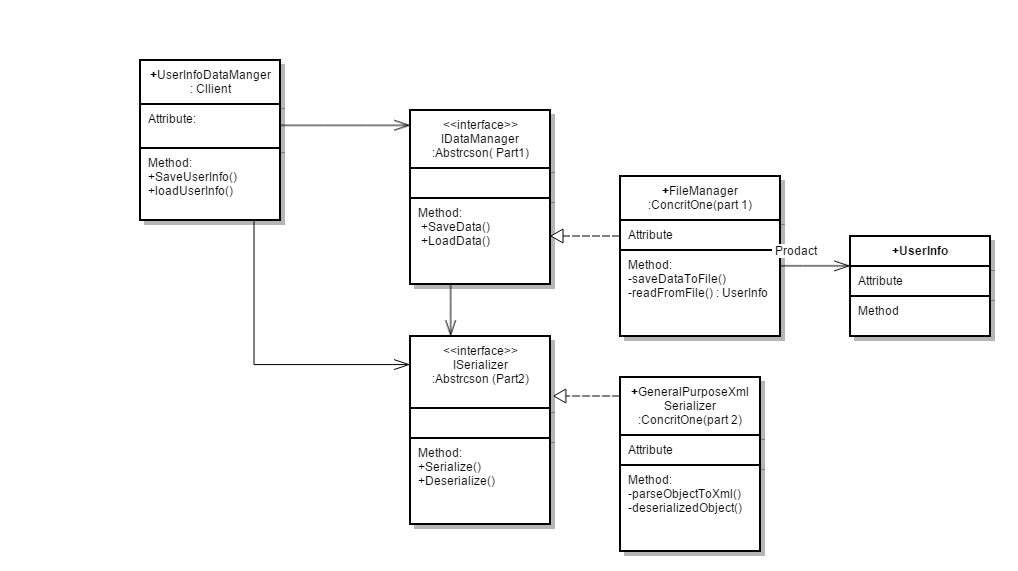
UserInfoDataManager מחזיקה מופעים של המחלקה GeneralPurposeXmlSerializer שמממש את ISerializer

וFileManager שיורש מ DataManager ובכך שומרת קבצי XML .

* Sequence Diagram



* Class Diagram



### תבנית מס' 4 – Singleton

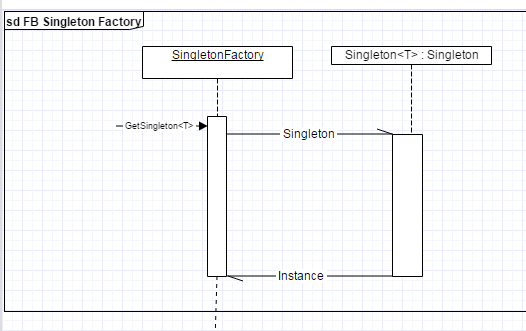
* סיבת הבחירה / שימוש בתבנית:

היה צורך בייצוג יחיד של מספר מחלקות תחזוקה(FbApiMultiFormAdapter, UserInfo, TwitchForFacebookProxy, TwitchAPIWrapper), כדי ליצור גישה גלובאלית לנתונים מסוימים הרלוונטיים לכל התכנית.

* אופן המימוש:

יצרנו מפעל סינגלטונים(המחלקה SingletonFactory) המאפשרת לנו לבקש לקבל גישה לעותק היחיד עבור סוג מסוים בו אנו רוצים להשתמש.  
המפעל מחזיר לנו את העותק היחיד, ואם אינו קיים, יוצר אותו בLazy Creation.

* Sequence Diagram



* Class Diagram

### C:\Users\Mor\Downloads\Singleton_Class_Diagram.png

### עבודה אסינכרונית

* היכן:

בעת בקשת מידע מהפייסבוק בטופס הראשי(במתודה initUserInfo).

ובעת בקשות מידע מTwitch בבדיקת תחילת השידור(במחלקה TwitchForFacebookProxy).

* כיצד:

הפעלנו את בקשות הקריאה והגישה לרשת בצורה אסינכרונית ובמקרה של עדכון רכיבי טופס השתמשנו בInvoke במקרה הצורך.

* מדוע:

בקשות הרשת לוקחות זמן רב עד לקבלת הנתונים הסופית ובזמן הזה מקפיאות את התכנית.  
העברתן לתהליכונים נפרדים אפשרה לנו לגרום לחוויית המשתמש להיות חלקה.

### Data Binding

* היכן:

קישרנו את רשימות החברים, הפוסטים והאירועים בFormMain לרכיבים מהAPI שסופק לנו.