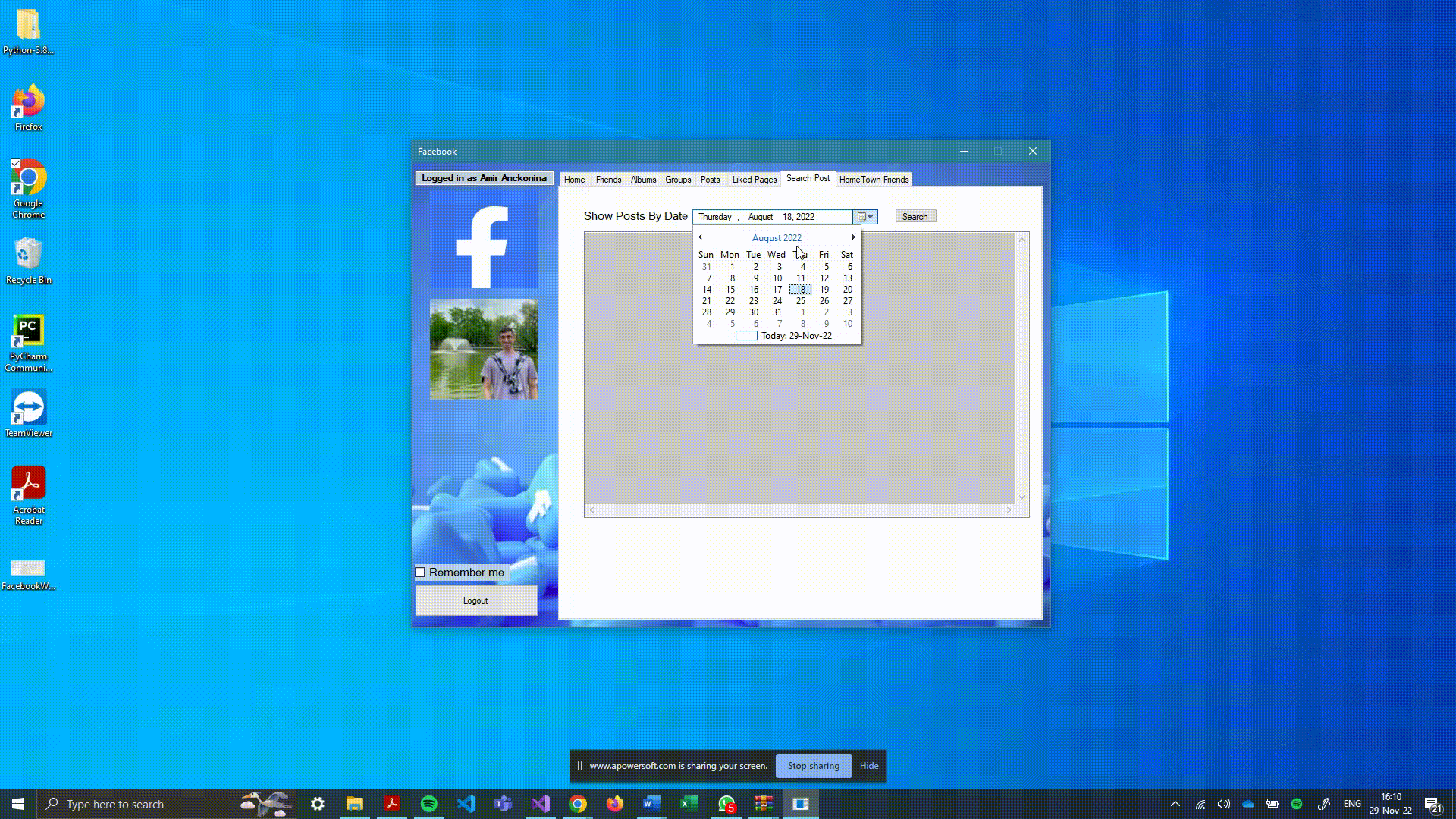
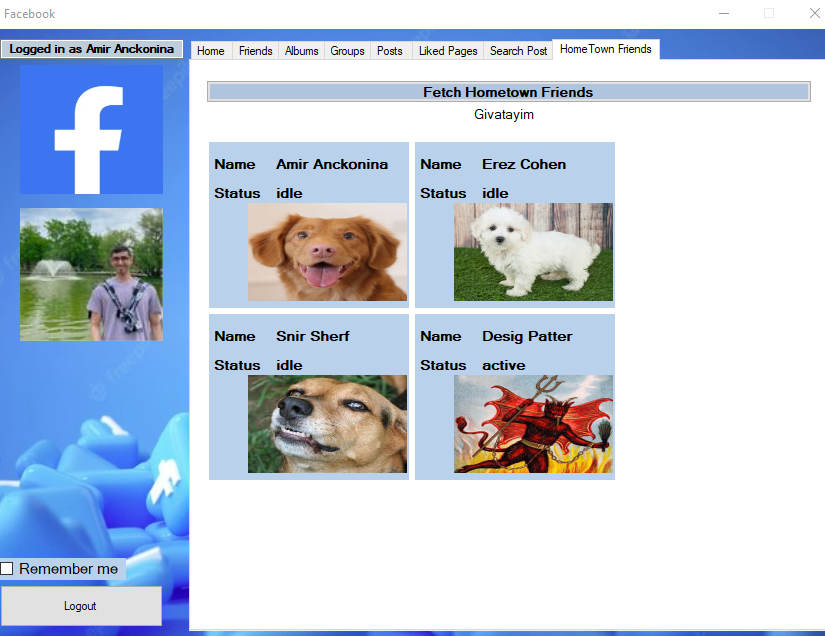
### תיאור קצר של הפיצ'רים שבחרנו לממש בתרגיל הקודם:

* **Search Post By Date**
  + הפיצ'ר הנ"ל נותן למשתמש את היכולת "לפלטר" פוסטים שפירסם בפייסבוק על בסיס תאריך.
  + המשתמש לוחץ על בחירת תאריך ונפתח עבורו תפריט נוח ואינטואיטיבי, בו ניתן לראות לוח שנה ולנוע בקלות בין שנים, חודשים וימים.
  + לאחר הבחירה, המשתמש לוחץ על Search וכל הפוסטים מהתאריך המצוין יופיעו.

****

* **Hometown Friends**
  + הפיצ'ר הנ"ל מאפשר למשתמש לקבל תצוגה נחמדה של חבריו בפייסבוקאשר מאותו עיר.
  + היות והגישה דרך הAPI חסומה, מימשנו Dummy Data בו רשימת חברים ולכל חבר יש רשומה של העיר בה הוא גר.
  + כמו כן, אנו מציגים למשתמש את סטטוס החבר כרגע באפליקציה.

****

### תבנית מס' 1 – Singleton

* **סיבת הבחירה / שימוש בתבנית:**

בחרנו לממש את המחלקה AppManager כמחלקה סינגלטונית כיוון שלאפליקציה שלנו נדרש אך ורק מנהל אחד ויחיד שידע לתפעל ולסנכרן בין הUI ללוגיקה.

כמו כן, האובייקט AppManager מכיל בו את כל האובייקטים הרצוים בינהם Forms, FbApiClient ועוד.

במידה והאפשרות ליצור יותר מ Instance אחד של המחלקה הייתה קיימת, זה עלול להוביל לפגיעה בפונקציונאליות של המערכת ולהוביל לבאגים לא רצוים.

זאת כי, בהרמת instance נוסף, יווצרו גם הData members בתוך AppManager, ובניהם בפרט FormAppSettings , FormLogin.

ניהול נוסף של Forms אלו עלול לגרום לבעיות ופגיעה בהגדרות הרצויות.

* **אופן המימוש:**

יצרנו מחלקה – AppManager שמטרתה לנהל את הForms השונים במערכת ולהגביל את מספר מנהלי האפליקציה ל1.

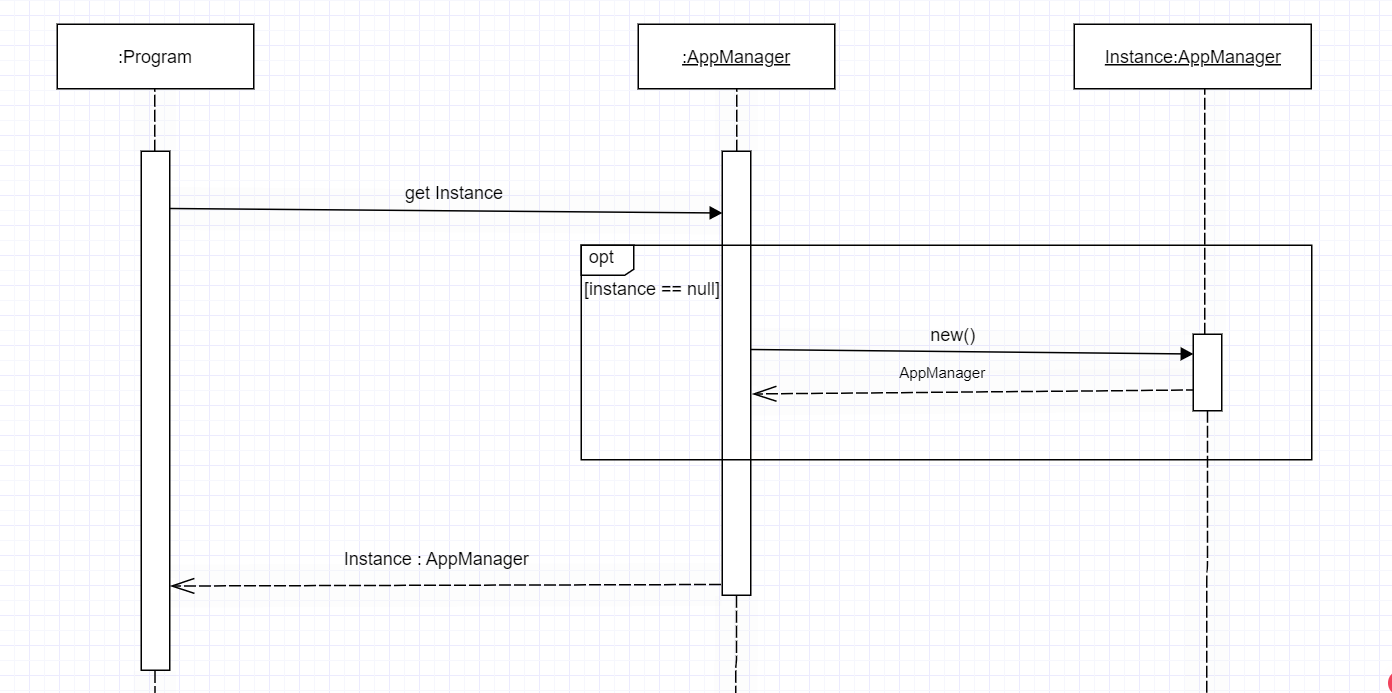
Access modifier של המחלקה שונה ל private. המחלקה הינה Sealed כי לא נרצה שתהיה אפשרות לרשת מהמחלקה (כי במידה וכן אז כשניצור אובייקט מהמחלקה היורשת אז נפעיל את הבנאי של הנורש והוא private).

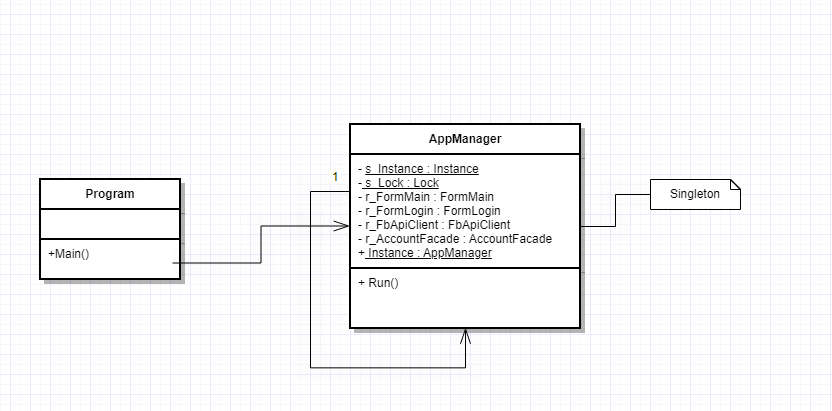
המחלקה מחזיקה שדה סטטי –

private static AppManager s\_Instance = null;

שדה זה מאותחל באופן Lazy רק במידה ונוצרת בקשה לקבל את הinstance.  
היווצרות האובייקט נוצרת בתוך getter של הפרופרטי Instance באמצעות שיטת double-check locking מה שהופך את היצירה ל thread-safe.

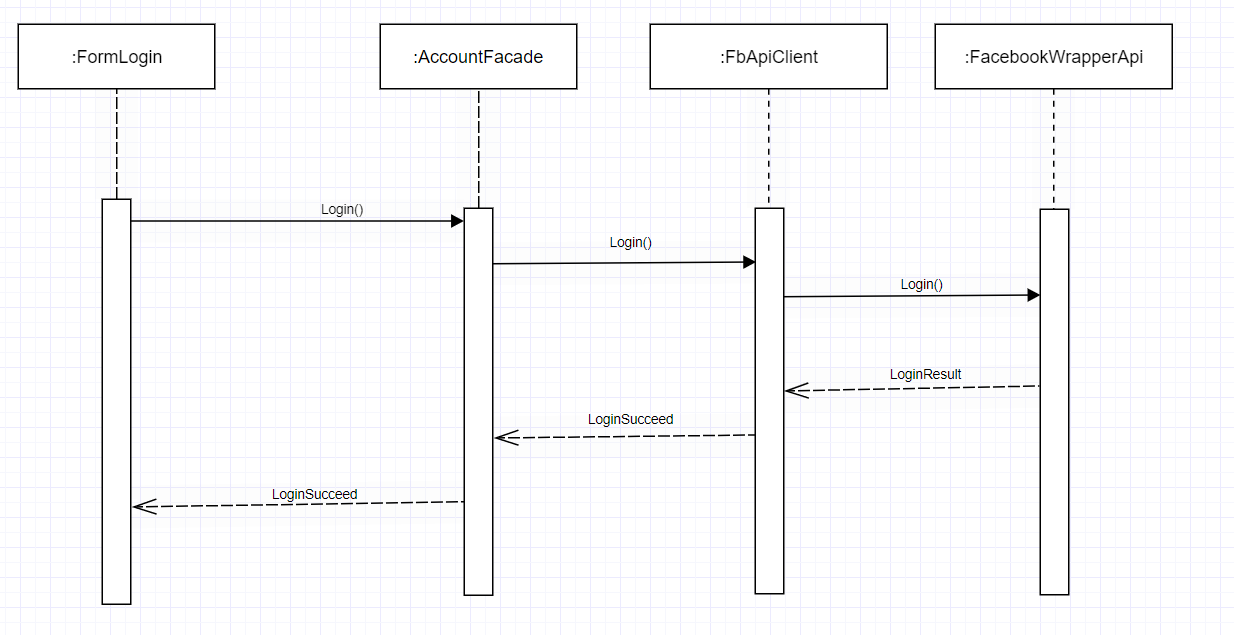
* **Sequence Diagram**



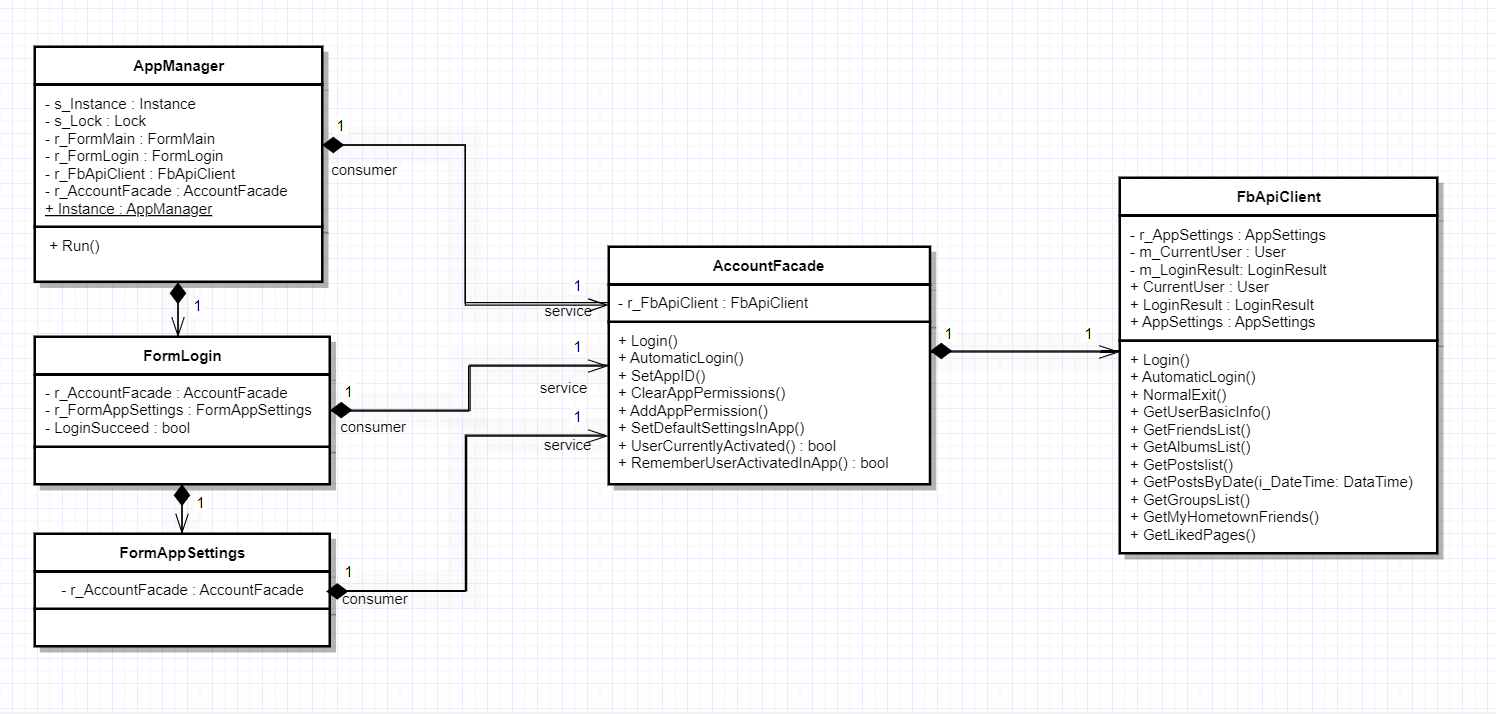
* **Class Diagram**

### תבנית מס' 2 – Facade

* **סיבת הבחירה / שימוש בתבנית:**
  + בחרנו לממש באפליקציה שלנו AccountFacade.
  + למעשה, באפליקציה יש לנו אובייקט FbApiClient ובו ממומשות מתודות רבות, ושירותים שונים.
  + לאותו FbApiClient, יש מס' צרכנים שונים באפליקציה. הראשי מבינהם FormMain שמקבל מהקליינט הנ"ל שירותי תכנים רבים. (הקליינט = FbApiClient)
  + אך, הצרכנים האחרים FormLogin, FormAppSettings, AppManager - זקוקים לשרותיו של הקליינט רק לטובת פעולות טכניות למשתמש. כגון: התחברות, שינוי הגדרות וכדומה.
  + עבור צרכנים אלו, נרצה לפשט את השימוש בקליינט, להעניק להם רק סט שירותים מצומצם, כי אינם זקוקים לרוב הפעולות המורכבות שהקליינט מחזיק.
  + ומכאן, שהשימוש ב Facade אכן ייקל על הצרכנים הללו, ולכן ראינו לנכון לממש זאת.
* **אופן המימוש:**
  + בעת יצירת האפליקציה ב AppManager אנו יוצרים קליינט FbApiClient .
  + מיד לאחר יצירת FbApiClient אנו יוצרים את AccountFacade ומעבירים את הקליינט ב DependencyInjection לאובייקטי ה Forms - FormLogin, FormAppSettings
  + מכאן, למעשה, ה Forms הנ"ל לא מחזיקים את ה FbApiClient אלא רק AccountFacade, וכל קריאה למתודה שמצריכה את שירותיו של הקליינט נעשית אך ורק דרך AccountFacade.
* **Sequence Diagram**
* בחרנו להראות כאן sequence ספיציפי של פעולת Login , מצרכן ספיציפי של ה Facade.

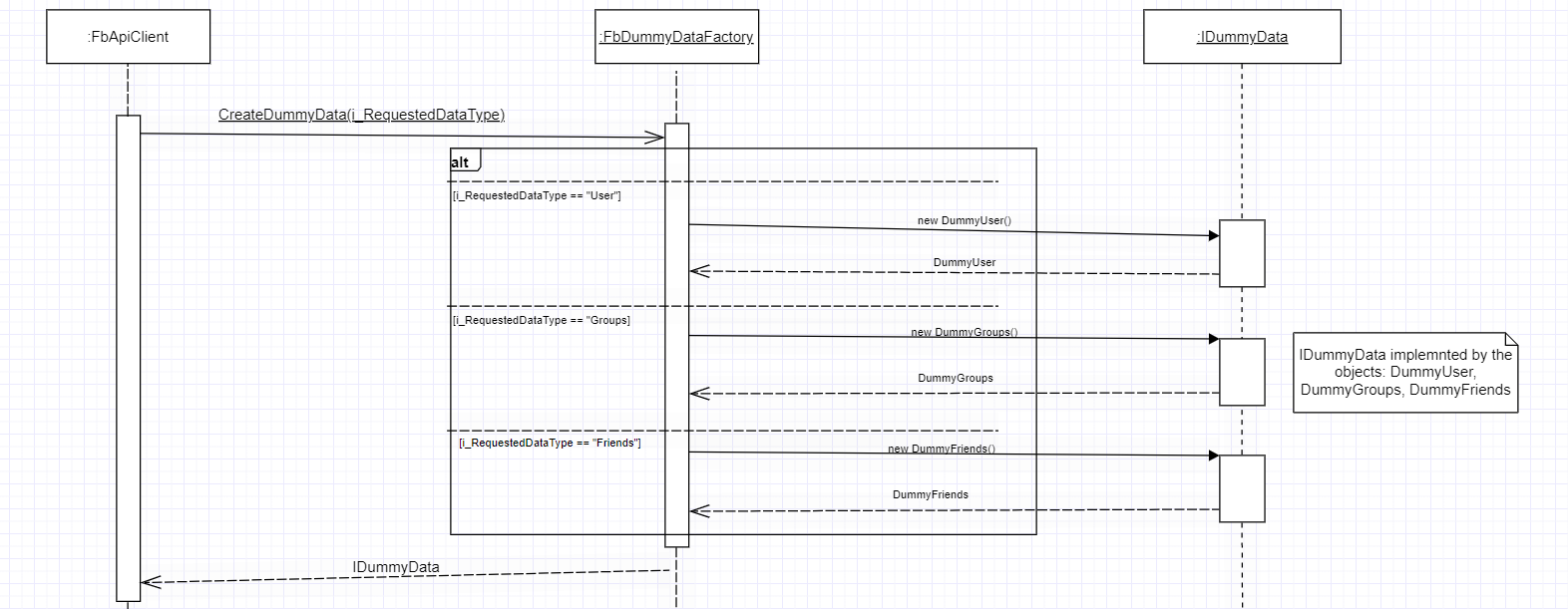


* **Class Diagram**

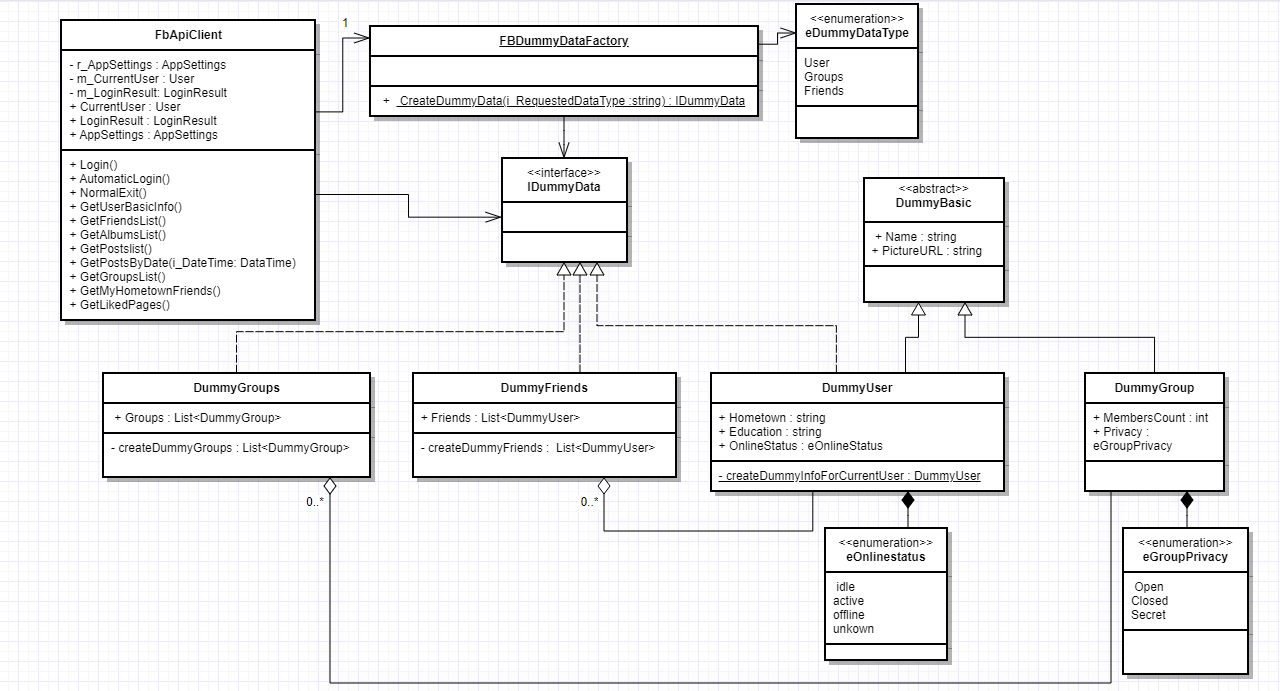


### תבנית מס' 3 – FactoryMethod

* **סיבת הבחירה / שימוש בתבנית:**
  + באפליקציה שלנו, בחרנו לממש FactoryMethod עבור יצירת ה DummyData.
  + על מנת להעשיר את האפליקציה שלנו במידע – יצרנו כבר בתרגיל 1 קבוצות, חברים ופרטי משתמש, כ"הכנה למזגן" לטובת הנתונים שנצליח או לא נצליח להביא דרך פייסבוק.
  + באפליקציה אנו יוצרים דאטה מסוגים שונים ועל כן ראינו לנכון לממש FactoryMethod, בו עבור כל דרישה נקבל את הדאטה המתאים.
* **אופן המימוש:**
  + הגדרנו מחלקה סטטית FbDummyDataFactory ובתוכה בדיוק מתודה סטטית אחת CreateDummyData.
  + המתודה הנ"ל מקבלת string פשוט ובו סוג הדאטה המבוקש.
  + המתודה בודקת את הערך ועל פי enum ייעודי שהוגדר, המייצג את תמיכת ה"פאקטורי" ביצירת דאטה, אנו בודקים האם סוג הדאטה הנ"ל נתמך, ובמידה וכן מחזירים אובייקט חדש מסוג IDummyData
  + IDummyData הינו interface הממומש על ידי DummyFriends, DummyGroups, DummyUser.
  + במידה וסוג הדאטה המבוקש אינו נתמך,אז נזרוק Exception מתאים המודיע שסוג הדאטה אינו נתמך על ידי המערכת. הגדלנו לעשות ואף יצרנו DummyDataException ייעודי לכך.
* Sequence Diagram



* Class Diagram



### עבודה אסינכרונית -

כאשר מתבצעת התחברות של המשתמש לאפליקציה, המערכת מבצעת קריאות לשרתי פייסבוק על מנת להביא את כל המידע הדרוש עבור אותו משתמש (פוסטים, תמונות, רשימת חברים, דפים אהובים ואלבומים).  
קריאות אלו דורשות זמן ומשאבים משום שמדובר בפניות לשרת חיצוני, ועד שהן לא מסתיימות האפליקציה נראת כ"תקועה", היא אינה רספונסיבית ולא ניתן להשתמש בממשק המשתמש.  
על מנת לקצר את זמני הטעינה של האפליקציה ולשפר את חוויית המשתמש, העברנו את הטיפול בהבאת המידע לטרדים נוספים.  
דבר זה מאפשר לאפליקציה להיות תגובתית (responsive) גם כשהמערכת עדיין לא סיימה להביא את כל המידע ובנוסף מקצרת את זמני הטעינה של האפליקציה. עבור כל שינוי ברכיבי ה UI מהטרדים שיצרנו, השתמשנו במטודה Invoke הנמצאת על כל אחד מהפקדים, ובכך עדכון המידע מתבצע דרך הטרד שאחראי על ה UI.  
  
השינויים הללו נמצאים במחלקה FormMain בבמטודות הבאות:

initBasicUserInfo(),initForm(),initLikedPages(),initAlbums(),initFriends(),

initGroups()

### עבודה עם Data Binding –

בקוד שלנו עשינו שימוש ב Data Bindingבtab אשר מציג את ה posts של המשתמש באופן הבא:   
המערכת מפעילה את הטופס הראשיFormMain , אשר שולח בקשה לשרת פייסבוק ובין היתר מחזיר collection של post. אנו עוברים על רשימת הפוסטים ויוצרים רשימה חדשה של אובייקט שיצרנו PostDTO. יצרנו postDTOBindingSource. במתודה initPosts() הגדרנו את DataSource להיות רשימת ה postDTO אשר קיבלנו. בנוסף, הגדרנו את הproperty Data Source של postListBox להיות postDTOBindingSource. וכך יצרנו data binding.

התהליך:

1. נוצרת בקשה לשרת לקבל רשימת פוסטים
2. postDTOBindingSource מתמלא בנתונים
3. postListBox מתמלא בנתונים מתוך postDTOBinding Souce
4. postPanel (חלונית בצבע תכלת) שואבת נתונים משתנים כאשר index של listbox משתנה. (הפקדים שנמצאים בתוך postPanel מקושרים ל postDTOBindingSource.