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

* **רשימת משאלות-** המשתמש יכול להוסיף את "רשימת המשאלות" שלו, המחולקת לפי קטגוריות (אוכל,קניות,חוויות וחיות) ולפרסם אותה כפוסט לפייסבוק שלו. קיימת אפשרות להוסיף משאלה עם תמונה וללא תמונה. אפשר לסמן ברשימה כאשר השגת את המשאלה שלך, אפשר למחוק משאלה. כל הנתונים נשמרים עם המשתמש.
* **אימונים-** המשתמש יכול להוסיף את האימונים שהוא ביצע, בהוספת אימון יש להוסיף: סוג אימון,תאריך, משך זמן האימון וקלוריות שנשרפו. ישנה תצוגה בפרופיל של המשתמש בטבלה של כל האימונים שביצע.  
   המשתמש יכול לקבל סטטיסטיקות של האימונים שהוא ביצע- כמה אימונים הוא עשה כל חודש וכמה קלוריות הוא שרף כל חודש. כל הנתונים נשמרים עם המשתמש.  
  **הוספנו אפשרות לפרסם כפוסט בפייסבוק ולהזמין חברים להתאמן.**

### תבנית מס' 1 – Factory Method

* סיבת הבחירה / שימוש בתבנית:

תבנית ה-Factory Method נבחרה על מנת להפריד את הלוגיקה של יצירת אובייקטים מסוג WishListItem מיתר הקוד. המטרה היא להבטיח שהאובייקטים ייווצרו בצורה תקינה ועקבית, תוך שימוש בפונקציות יצירה ברורות שמספקות ולידציה על הנתונים הדרושים. בנוסף, התבנית מספקת הרחבה קלה ליצירת סוגים שונים של פריטים בעתיד, אם יידרש.

#### אופן המימוש:

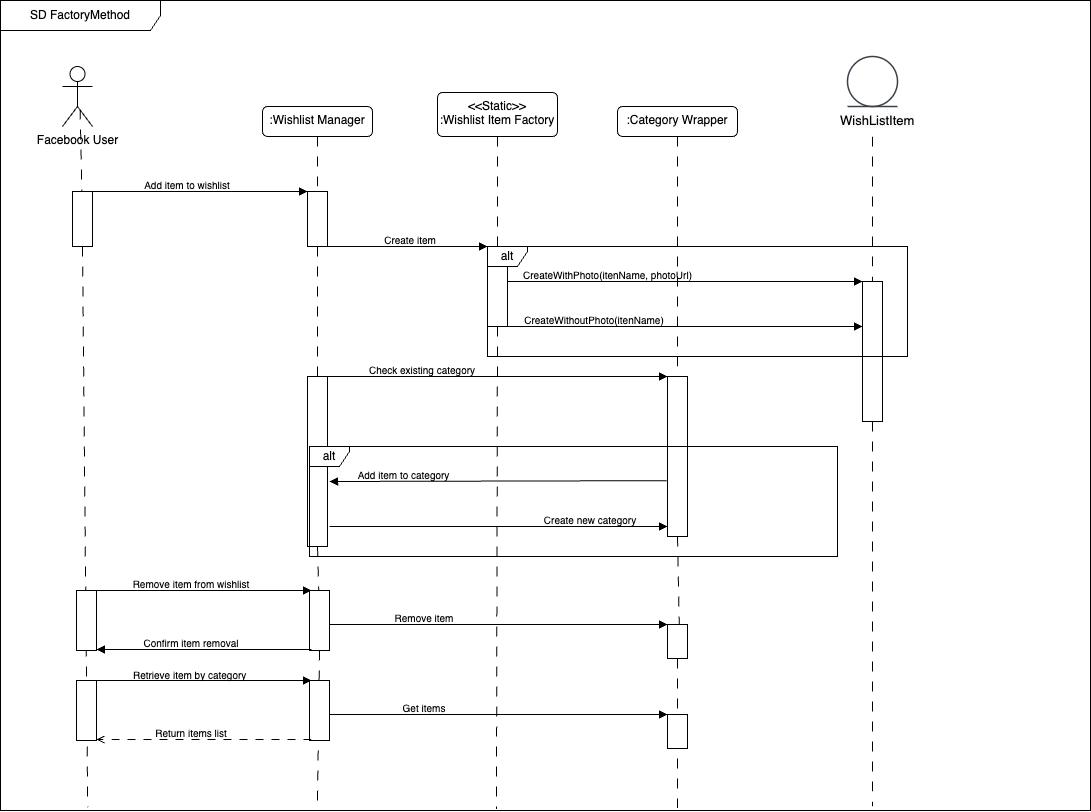
#### המימוש מתבצע באמצעות מחלקה סטטית בשם WishListItemFactory. במחלקה זו קיימות שתי פונקציות:

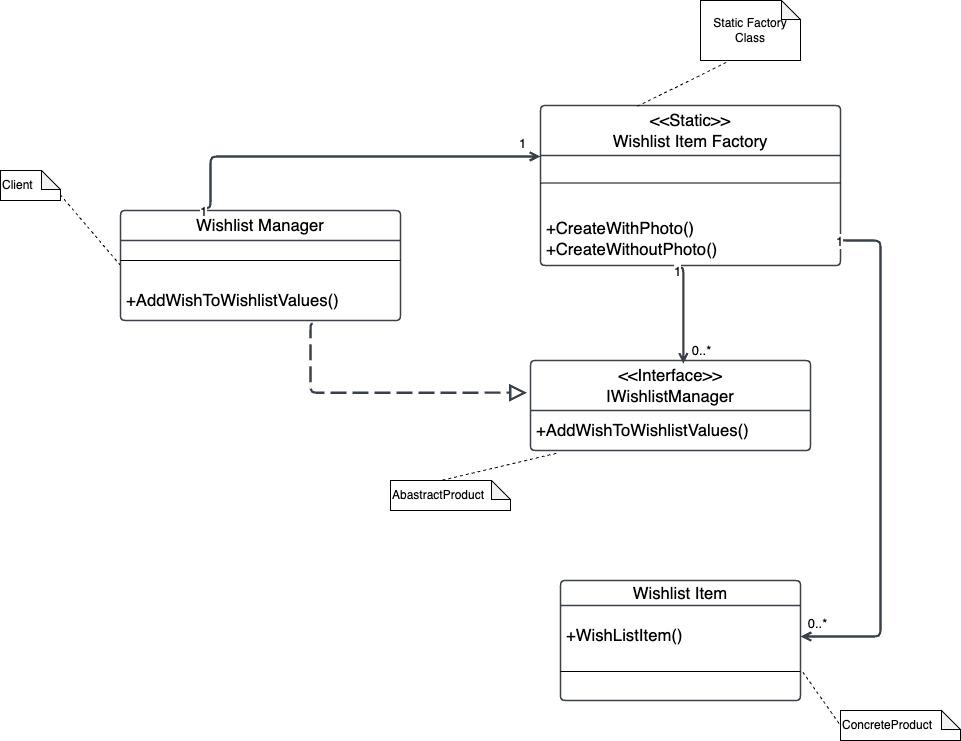
1. CreateWithPhoto - יוצרת אובייקט WishListItem הכולל תמונה. הפונקציה מבצעת בדיקה אם ה-URL של התמונה תקין (לא ריק או null), ואם לא, נזרקת חריגה.
2. CreateWithoutPhoto - יוצרת אובייקט WishListItem ללא תמונה.

המימוש בקוד:

1. מחלקת WishListItemFactory (קובץ: WishListItemFactory.cs)​.  
2. מחלקת WishlishManager - שורות 18-20  
3. מחלקת FormMain - שורות 462-497

* Sequence Diagram





* Class Diagram

### תבנית מס' 2 – Builder

* סיבת הבחירה / שימוש בתבנית:

תבנית ה-Builder נבחרה לצורך יצירה מסודרת ומובנית של אובייקטים מסוג Workout, הכוללים מאפיינים כמו קטגוריה, משך זמן, תאריך וקלוריות. השימוש בתבנית נועד להתמודד עם הצורך לוודא שכל הנתונים הדרושים מוגדרים בצורה תקינה, תוך ביצוע ולידציה לכל מאפיין במהלך תהליך הבנייה.

הבחירה בתבנית זו מתאימה במיוחד למערכת שלנו, שבה יצירת אובייקט Workout מתבצעת על בסיס נתונים שמתקבלים מממשק המשתמש (AddWorkoutForm). תבנית ה-Builder מאפשרת להפריד בין הלוגיקה של קבלת הנתונים לבין לוגיקת הבנייה עצמה, הנעשית באופן הדרגתי ומדויק באמצעות WorkoutBuilder.

בנוסף, השימוש בקומפוזר מאפשר לנהל את סדר הבנייה ולהבטיח שהאובייקט ייבנה בצורה נכונה, מבלי שהממשק יידרש לדעת את כל הפרטים הטכניים של תהליך הבנייה. הבחירה בתבנית זו מספקת גם גמישות להוספת מאפיינים חדשים בעתיד, תוך שמירה על קוד מסודר ונגיש.

* אופן המימוש:

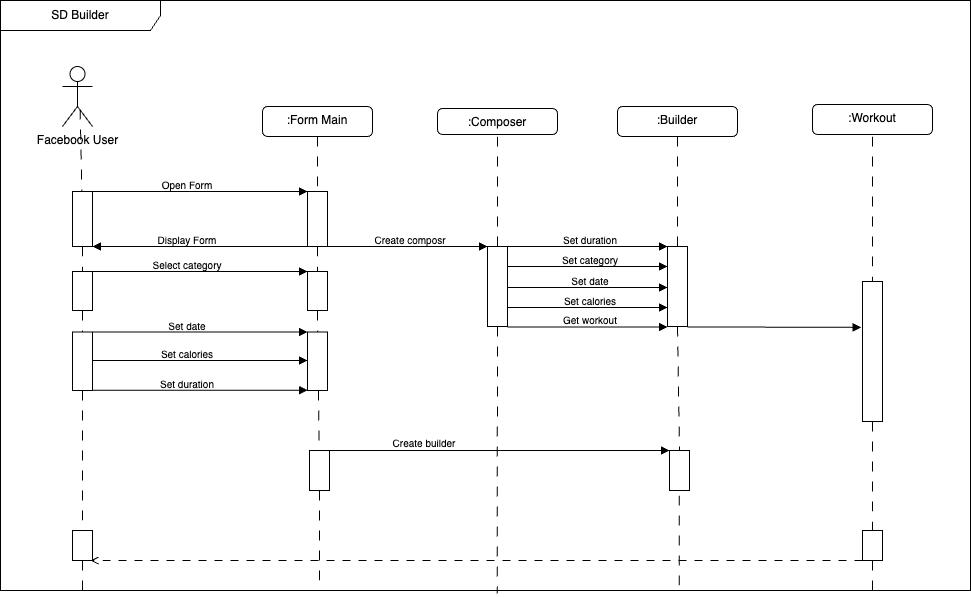
מחלקת WorkoutComposer מתפקדת כמובילה של תהליך הבנייה בתבנית ה-Builder. היא אחראית על ניהול תהליך הבנייה של האובייקט Workout ומכתיבה את הסדר שבו יש להפעיל את מחלקת הבילדר.

המחלקה מקבלת נתונים חיוניים (כגון משך זמן, קטגוריה, תאריך וקלוריות), ומשתמשת בממשק הבילדר כדי ליצור את האובייקט בהתאם להגדרות שהועברו.

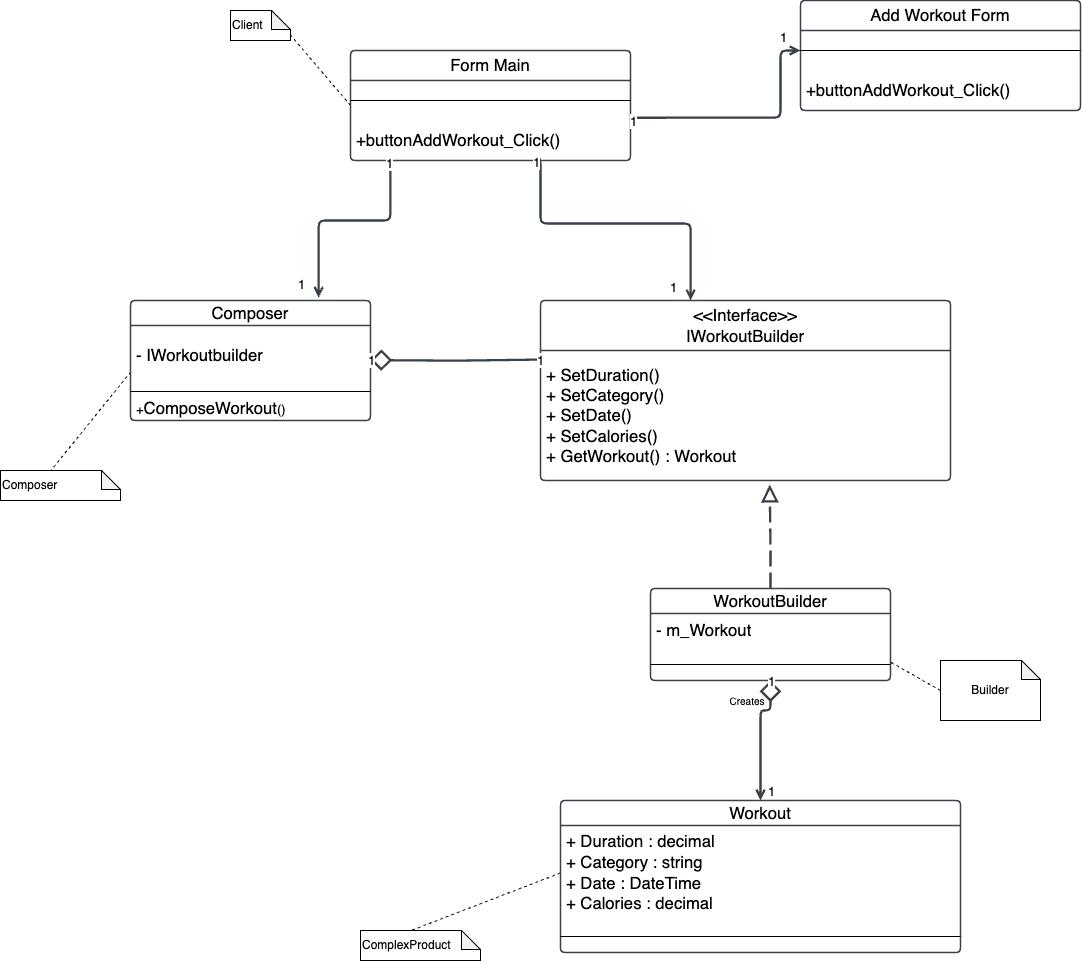
מחלקת WorkoutBuilder אחראית על הבנייה הפיזית של האובייקט Workout.

היא כוללת מתודות ייעודיות עבור כל מאפיין באובייקט.  
  
מיקום הקוד: 1. קובץ WorkoutComposer.cs​  
2. קובץ WorkoutBuilder.cs  
3. בתהליך הוספת אימון חדש מתוך ממשק המשתמש (AddWorkoutForm), נעשה שימוש בקומפוזר כדי ליצור אובייקט Workout תוך הסתמכות על מחלקת הבילדר: שורות 52-53

* Sequence Diagram



* Class Diagram



### תבנית מס' 3 – Facade

* סיבת הבחירה / שימוש בתבנית:  
  תבנית ה-Facade נבחרה במערכת כדי לפשט את הגישה לפונקציונליות מורכבת הנמצאת ברקע, לדוגמה: אינטראקציה עם ממשקי API של Facebook, ניהול רשימות משאלות ואימונים. במקום שממשק המשתמש יתמודד עם כל שכבות הלוגיקה ויידרש להבין את כל פרטי המימוש, תבנית ה-Facade מציעה נקודת גישה אחידה וברורה לממשקים מורכבים אלו.  
  התבנית מקטינה את התלות הישירה בין רכיבי המערכת ומקלה על תחזוקה והרחבה עתידית. כמו כן, היא מאפשרת עבודה גמישה ונקייה עם מרכיבי הלוגיקה על ידי בידוד שלהם מאינטראקציות ה-UI.
* אופן המימוש:  
  מחלקת FacebookFacade  
  תפקיד: מספקת נקודת גישה אחידה לפונקציונליות של Facebook, כולל ניהול פוסטים, אלבומים, קבוצות, חברים ועוד.  
  מימוש: המחלקה עושה שימוש במחלקות FacebookManager ו-FacebookManagerUI לניהול הלוגיקה והאינטראקציה עם המשתמש.

מחלקת WishlistFacade  
תפקיד: ניהול רשימת המשאלות של המשתמש, כולל הוספה, מחיקה, והצגת נתונים בממשק המשתמש.  
מימוש: עושה שימוש במחלקות WishlistManager ו-WishlistManagerUI כדי לנהל את הלוגיקה והאינטראקציה הוויזואלית.  
מחלקת WorkoutFacade

תפקיד: ניהול אימוני המשתמש, כולל הצגת סיכום האימונים, עדכון נתונים, והזמנת חברים לאימונים משותפים.  
מימוש: משלבת את WorkoutManager ו-WorkoutManagerUI לניהול הנתונים והאינטראקציה הוויזואלית.

שימוש ב-Facade ב-FormMain

מחלקת FormMain עושה שימוש ב-Facade כדי לבצע פעולות שונות בממשק המשתמש, תוך הפחתת התלות בלוגיקה הפנימית.

מיקום הקוד: 1. קובץ WorkoutFacade.cs​  
2. קובץ WishlistFacade.cs

3. קובץ FacebookFacade

4. במחלקה **FormMain** נעשה שימוש בתבנית **Facade** כדי לפשט אינטראקציה עם תתי-מערכות מורכבות. להלן סיכום השימושים:

### FacebookFacade:

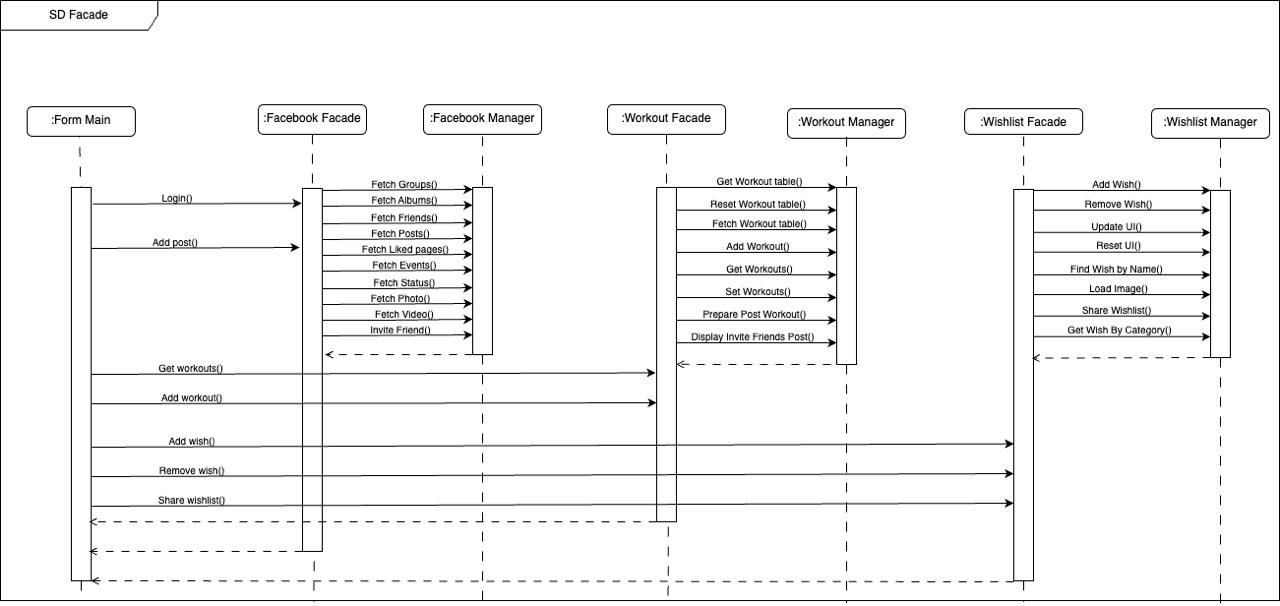
* **אתחול**:
  + אתחול המופע מתבצע לאחר התחברות המשתמש (שורות 104, 227).
* **שימושים מרכזיים**:
  + משיכת נתונים:
    - קבוצות (FetchGroups, שורה 188).
    - חברים (FetchFriends, שורה 192).
    - פוסטים (FetchPosts, שורה 196).
    - אלבומים (FetchAlbums, שורה 201).
    - דפים אהובים (FetchLikedPages, שורה 205).
    - אירועים (FetchEvents, שורה 209).
  + פרסום מידע:
    - פרסום סטטוס (PostStatus, שורה 298).
    - פרסום תמונה (PostPhoto, שורה 311).
    - פרסום סרטון (PostVideo, שורה 330).
  + יצירת ממשק:
    - יצירת פאנלים עבור פוסטים, אלבומים, קבוצות ועוד (שורה 255 ואילך).

### WishlistFacade:

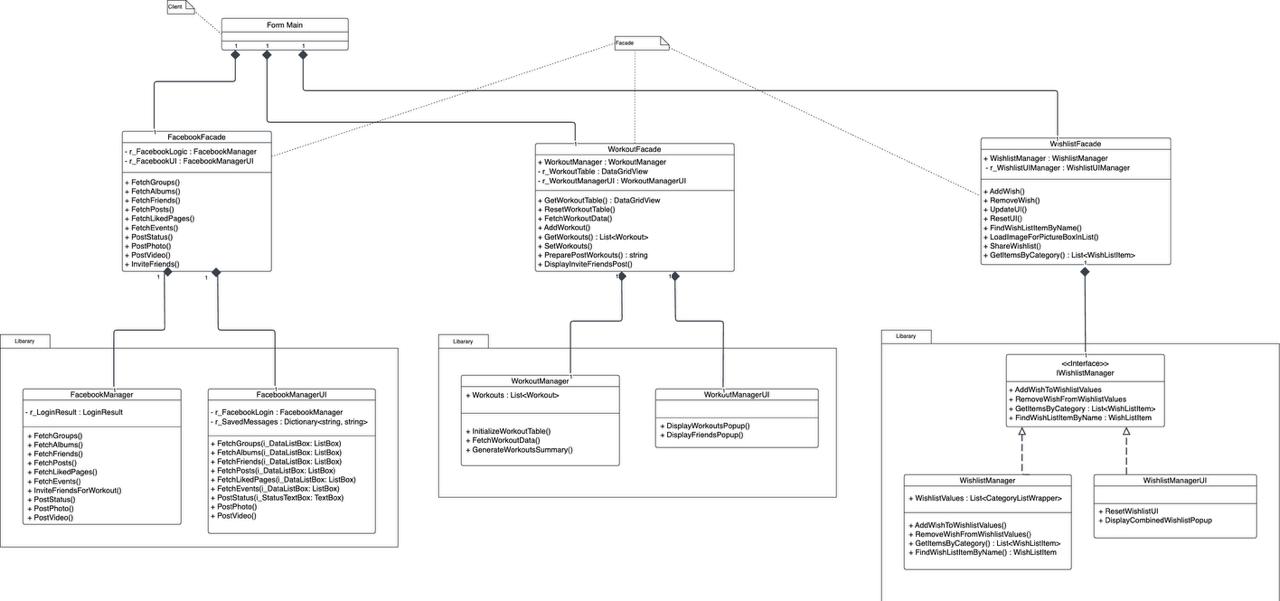
* **אתחול**:
  + אתחול המופע מתבצע מתוך הגדרות האפליקציה (r\_AppSettings.WishlistFacade, שורות 34-35, 44-45).
* **שימושים מרכזיים**:
  + ניהול נתונים:
    - הוספת פריט חדש לרשימת המשאלות (AddWish, שורה 477).
    - עדכון ממשק לאחר הוספה או שינוי (UpdateUI, שורה 490).
    - איפוס הממשק (ResetUI, שורה 183).
    - מחיקת פריט (RemoveWish, שורה 613)
  + חיפוש נתונים:
    - חיפוש פריט ברשימת המשאלות (FindWishListItemByName, שורה 533,543).
  + שיתוף:
    - יצירת פוסט רשימת משאלות (ShareWishlist, שורה 566).

### WorkoutFacade:

* **אתחול**:
  + אתחול המופע מתבצע מתוך הגדרות האפליקציה (r\_AppSettings.WorkoutFacade, שורות 36-37,48-52).
  + הטבלה מוצגת בלוח האימונים (panelWorkouts, שורה 38).
* **שימושים מרכזיים**:
  + ניהול נתונים:
    - משיכת נתוני אימונים (FetchWorkoutData, שורה 294).
    - הוספת אימון חדש (AddWorkout, שורה 626).
    - איפוס טבלת אימונים (ResetWorkoutTable, שורה 182).
  + שיתוף:
    - יצירת סיכום אימונים לשיתוף (PreparePostWorkouts, שורה 640).
    - הזמנת חברים לפעילות (InviteFriends, שורה 661).  
      במחלקה InviteFriendsForm - הצגת פוסט הזמנת חברים (DisplayInviteFirendsPost, שורה 68)
* Sequence Diagram



* Class Diagram



#### עבודה אסינכרונית

העבודה האסינכרונית מתבצעת בעיקר במקומות שבהם יש צורך לשלוף נתונים מ-API של Facebook או לבצע פעולות כבדות מבלי לחסום את ממשק המשתמש (UI).

### מקומות מרכזיים בעבודה אסינכרונית בקובץ FacebookManagerUI.cs

#### 1. שליפת נתונים

שליפת נתונים מ-Facebook מתבצעת באמצעות Thread, תוך שימוש במנעול (lock) כדי להבטיח עבודה בטוחה עם משאבים משותפים. פונקציות אלו מבצעות קריאות ל-Facebook API ועדכונים דינמיים בממשק המשתמש.

* **פונקציות אסינכרוניות עיקריות**:
  + FetchGroups (שורות 27–61): שליפת קבוצות משתמש והצגתן ב-ListBox.
  + FetchAlbums (שורות 61–96): שליפת אלבומים והצגת שמות האלבומים בממשק.
  + FetchFriends (שורות 97–131): שליפת רשימת חברים ועדכונה ב-ListBox.
  + FetchPosts (שורות 132–169): שליפת פוסטים ותמונות והצגתן בממשק.
  + FetchLikedPages (שורות 170–204): שליפת עמודים שאהבת והצגתם.
  + FetchEvents (שורות 205–238): שליפת אירועים והצגתם בממשק.

#### 2. פרסום פוסט

פרסום פוסטים מתבצע בצורה אסינכרונית כדי למנוע חסימת ממשק המשתמש במהלך פעולת הפרסום.

* **פונקציות אסינכרוניות עיקריות**:
  + PostStatus (שורות 240–264): פרסום סטטוס ב-Facebook עם הודעה על הצלחה או שגיאה.

#### 3. פתיחת אלבום תמונות

פתיחת אלבום מאפשרת צפייה בתמונות מתוך אלבום שנבחר. פעולה זו מתבצעת גם היא בצורה אסינכרונית, על ידי יצירת ממשק דינמי.

* **פונקציות אסינכרוניות עיקריות**:
  + MakeAlbumPanel (שורות 743–749): לחיצה על כפתור צפייה באלבום מפעילה Thread לפתיחת אלבום והצגת תמונות בממשק.

#### Data Binding

##### **מקומות מרכזיים ב-Data Binding-**

FacebookManagerUI.cs:

בפונקציה: (**MakeAlbumPanel (669-774**

* **מטרה**: לאפשר למשתמש לערוך את שם האלבום בתיבת טקסט, ולעדכן את המידע המוצג ואת אובייקט הנתונים בפועל.
* **אופן הפעולה**:
  1. כאשר המשתמש משנה את הטקסט בתיבת הטקסט, השינוי מתעדכן באופן מיידי בתווית המציגה את שם האלבום.
  2. כשמשתמש עוזב את תיבת הטקסט, מתבצעת בדיקה אם השם השתנה.
  3. אם יש שינוי, המידע המעודכן נשמר באובייקט האלבום, והרשימה בממשק מתעדכנת בהתאם.

#### בפונקציה: (MakePostPanel (548-668

* **מטרה**: לאפשר למשתמש לערוך את תוכן הפוסט בתיבת טקסט, ולעדכן את המידע המוצג והנתון הקיים.
* **אופן הפעולה**:
  1. בעת שינוי הטקסט בתיבת הטקסט, השינוי מתעדכן באופן מיידי בתווית המציגה את תוכן הפוסט.
  2. כאשר המשתמש בוחר לשמור את התוכן החדש, המידע המעודכן נשמר באובייקט הפוסט ומוצג.