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

* פיצ'ר ראשון – GetRandomPhotoFromRandomAlbum:

בפיצ'ר זה אנחנו מספקים למשתמש לשחק במשחק שבו הוא צריך לנחש באיזה תאריך צולמה התמונה.  
המערכת מגרילה אלבום ותמונה מתוך האלבום.  
מערכת נותנת למשתמש לבחור את התאריך שהוא חושב שבו צולמה התמונה.  
המשתמש מקבל הודעה מתאימה בהתאם לבחירה.

* פיצ'ר 2 – DownloadAlbum:

הפיצ'ר נותן אופציה לבחירת אלבום מתוך אלבומי התמונות של המשתמש.   
ברגע שהמשתמש יבחר אלבום הוא יוכל לצפות בתמונות שנמצאות באלבום זה ולוודא שזה האלבום הרצוי.  בלחיצה על כפתור ההורדה תיפתח חלונית לבחירת נתיב שבו תיווצר תיקייה ולשם יורדו תמונות האלבום הנבחר.

### תבנית מס' 1 – [Proxy]

* **סיבת הבחירה / שימוש בתבנית:**

תבנית ה-Proxy נבחרה לשימוש במערכת כדי לשפר את הביצועים ולמנוע גישה ישירה ושאילתות מרובות לאובייקט המשתמש של Facebook (שהוא User מה-SDK). באמצעות שימוש בתבנית זו, ניתן לאחסן את המידע הנדרש באופן מקומי בזיכרון (Cache) ולבצע קריאות ל-API של Facebook רק בעת הצורך, מה שמקטין את התלות ברשת, מייעל את התקשורת, ומשפר את זמן התגובה של המערכת.

* **אופן המימוש:**

תבנית ה-Proxy ממומשת באמצעות המחלקה UserCashingProxy.

רצינו לייצר מחלקת UserCashingProxy שתשמור את נתוני אובייקט ה-User של .Facebook

מכיוון שאובייקט ה-User של Facebook הואlegacy קומפוננט אז לא ניתן לשנות אותו ומכיוון שלא מממש interface וכל המתודות שלו הם אינם virtual אז לא ניתן שהמחלקה תירש מUser ולכן לא היה ניתן שרכיב ה-Proxy שלנו התחזה לרכיב הUser

לכן בנינו מחלקה שנקראת UserAdapter שהיא מהווה מתאם לUser על מנת שרכיב ה-Proxy יוכל להתחזות ממש אליו. ה- UserAdapter מחזיק בקומפוזיציה של User.

בשביל זה מיממשנו ממשק IUserAdapter שיממשו UserAdapter ו UserCashingProxy .

המידע המבוקש (כמו קבוצות, פוסטים, תמונות וכו') מאוחסן במשתני Cache בתוך המחלקה.

כל משתנה Cache מתעדכן רק אם האובייקט המקורי (User) עבר שינוי מאז הפעם האחרונה שבה המידע נטען.

Legacy A – UserAdapter, Legacy B – SystemManager, IProxy – IuserAdapter , Proxy – UserCashingProxy.

מיקום בקוד:

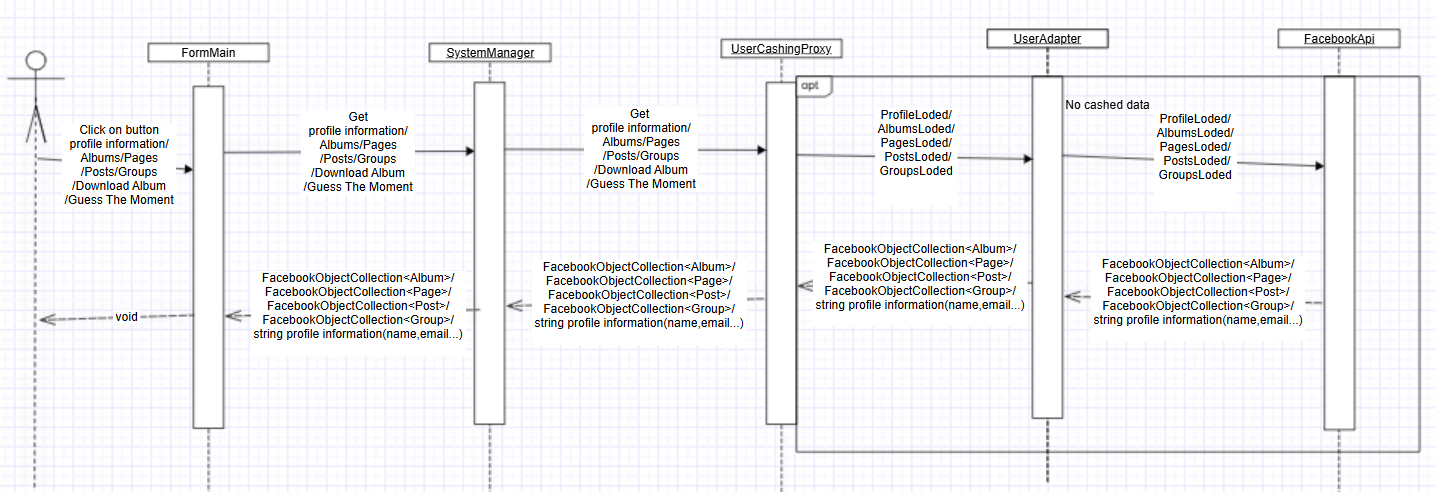
מחלקת UserCashingProxy: אחראית לנהל את ה-Caching של נתוני המשתמש.

מחלקת SystemManager: משמשת כמתווך לניהול המערכת.

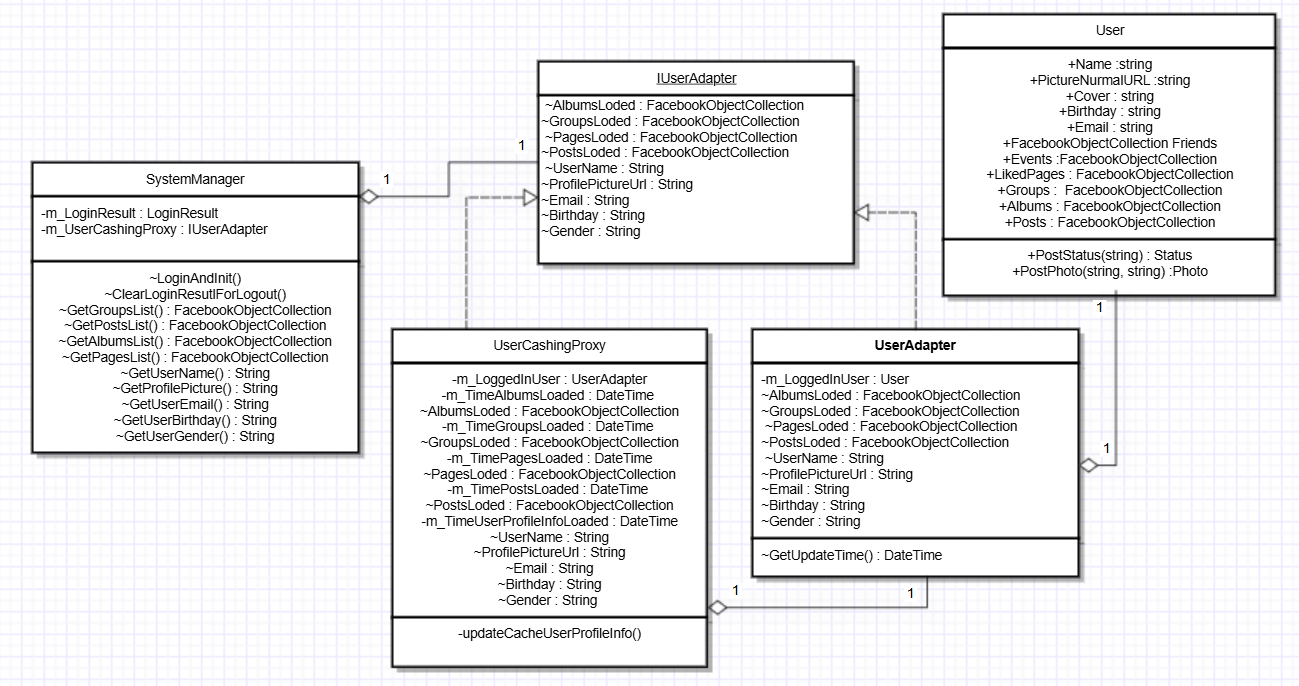
מחלקת UserAdapter: מתאם לUser

מחלקת : IUserAdapter ממשק שמממשים UserAdapter , UserCashingProxy

* **Sequence Diagram**



* **Class Diagram**



### תבנית מס' 2 – [Factory method]

* **סיבת הבחירה / שימוש בתבנית:**

בחרנו להשתמש בתבנית [Factory Method] מכיוון שכל הForms במערכת   
(כגון FormLogin ,FormMain) יורשים מהמחלקה הבסיסית Form ומשתייכים למשפחה פולימורפית. הבחירה בתבנית מאפשרת לנו לרכז את הלוגיקה של יצירת הForms במחלקה נפרדת (FormFactory) ובכך להפריד את אחריות יצירת האובייקטים מהClient  
גישה זו מספקת כמה יתרונות משמעותיים:

* **קונסיסטנטיות ואחידות בקוד:** כל הForms נוצרים דרך מנגנון סטנדרטי ומרוכז.
* **שימוש חוזר (Reusability):** הקוד ליצירת Forms מסוימים נכתב פעם אחת בלבד.
* **תחזוקה והרחבה קלה:** במידה ונדרש להוסיף Form חדש למשפחה, נוכל לעדכן את המחלקה FormFactory בלבד, מבלי לשנות את הקוד בClient
* **הפחתת מורכבות ב-Client:** הClient אינו צריך להבין כיצד ליצור את

הForms הוא רק מבקש את הForm הרצוי ומקבל אותו.

* **אופן המימוש:**

מימוש התבנית נעשה במחלקה FormFactory שהיא מחלקה סטטית. במחלקה זו הוגדרה מתודה סטטית בשם createFormBasedOnType אשר מקבלת פרמטר מסוג eFormType (Enumerator) ומחזירה אובייקט של Form בהתאם לסוג שהוזן.

מיקום בקוד:

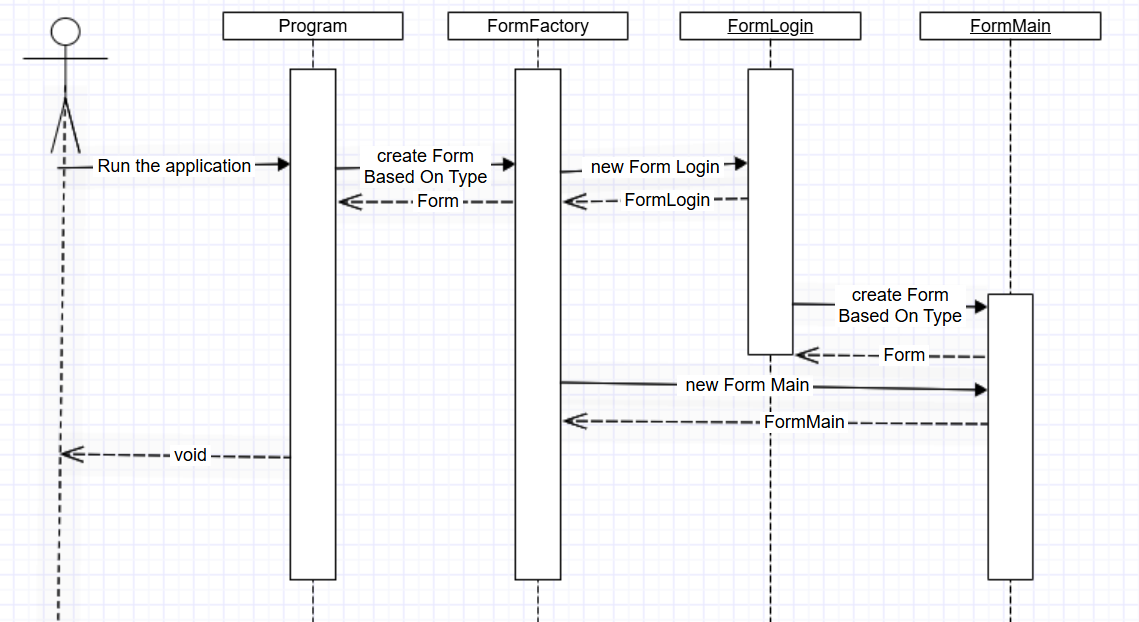
**במחלקת Program:**  
הFactory Method משמש כאן לניהול נקודת הכניסה הראשית לאפליקציה:

Application.Run(FormFactory.createFormBasedOnType(FormFactory.eFormType.FormLogin));

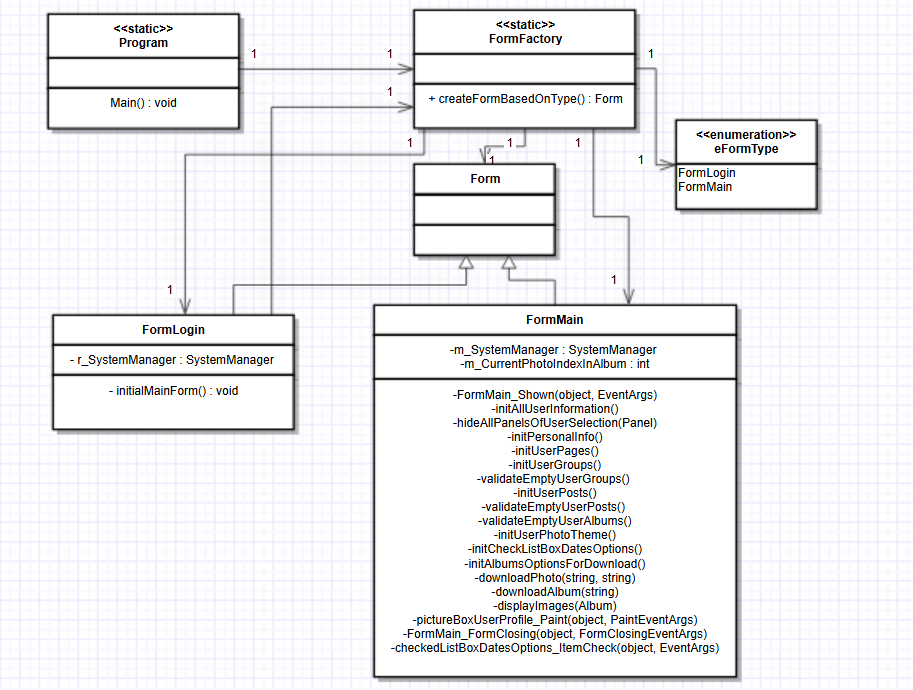
**במחלקת FormLogin:**  
לאחר ביצוע תהליך ההתחברות (Login), מחלקה זו משתמשת בFactory Method לצורך יצירת הForm הבא (FormMain) :

Form formMain = FormFactory.createFormBasedOnType(FormFactory.eFormType.FormMain);

* **Sequence Diagram**



* **Class Diagram**



### תבנית מס' 3 – [Singleton]

* **סיבת הבחירה / שימוש בתבנית:**

התבנית נבחרה עבור המחלקה GuessTheMomentManager כדי להבטיח שקיים מופע יחיד בלבד של המחלקה לאורך חיי האפליקציה.

מטרת השימוש בתבנית היא:

ייעול ניהול הזיכרון על ידי שליטה מרכזית במופע יחיד של המחלקה.

מניעת בעיות תחרות (race conditions) בזמן עבודה עם נתונים רגישים.

מפשט את תהליך השמירה והטעינה של משחק.

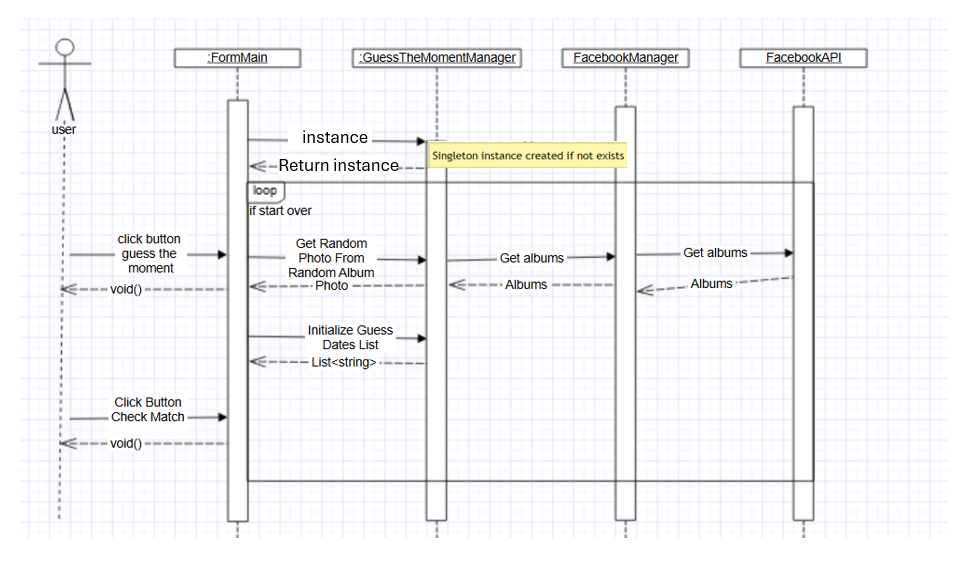
מבטיח גישה עקבית למשחק מכל חלקי האפליקציה.

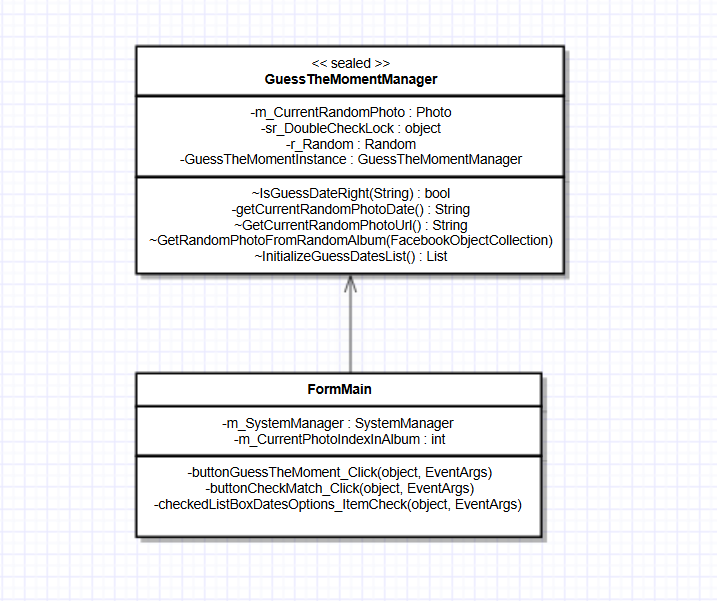
* **אופן המימוש:**
  + המחלקה מכילה משתנה סטטי פרטי מסוג GuessTheMomentManager בשם s\_GuessTheMomentInstance אשר מחזיק את המופע היחיד.
  + מנגנון נעילה כפולה (Double-Checked Locking) באמצעות אובייקט sr\_DoubleCheckLock מבטיח יצירה בטוחה של המופע בסביבת ריבוי-Threads.
  + הבנאי מוגדר כפרטי כדי למנוע יצירה חיצונית של מופעים חדשים.
  + גישה למופע היחיד מתבצעת באמצעות מאפיין סטטי ציבורי GuessTheMomentInstance.
  + המחלקה מסומנת כ-sealed כדי למנוע ירושה ממנה.

מיקום בקוד:

השימוש במחלקה מופיע בקובץ FormMain.cs, בתוך אירועים המופעלים על ידי לחיצה על כפתורים וממשק המשתמש.

(למשל buttonGuessTheMoment\_Click)

* **Sequence Diagram**
* 
* **Class Diagram**



**Data Binding**

בפרויקט עשינו שימוש ב- Data Binding לקישור בין נתוני העמודים מפייסבוק לרכיב ה ListBox והpictureBoxPagePhoto  בממשק המשתמש. המתודה initUserPages() שולפת את העמודים בצורה אסינכרונית, ומעדכנת את ה ListBox דרך אובייקט BindingSource . כך, כל שינוי ברשימת הנתונים מתעדכן אוטומטית בממשק ללא צורך בהתערבות ידנית. הגישה מאפשרת הפרדה בין לוגיקת הנתונים לתצוגה, משפרת ביצועים, ומקלה על תחזוקת הממשק**.**

במתודה נעשה שימוש בשורת הקוד הבאה: pageBindingSource.DataSource = pages;

**תכנות אסינכרוני**

**טעינה של נתונים כלליים על הUser:**

* **מיקום:** בפונקציות  FormMain\_Shown() ו-  buttonProfile\_Click()
* **סיבה:** טעינת נתונים כמו תמונת פרופיל ושם ונתונים נוספים עשויה להיות כבדה, ולכן מתבצעת בתהליך נפרד כדי לשמור על ממשק מהיר וחלק.
* **ביצוע:** שימוש ב Thread לטעינה ברקע ועדכון פקדי הממשק, labelUserName.Text pictureBoxUserProfile  עם Invoke.

**טעינת פוסטים, דפים, קבוצות ואלבומים:**

* **מיקום:** פונקציות buttonPosts\_Click(), buttonPages\_Click(), buttonGroups\_Click()  ו-buttonAlbums\_Click().
* **סיבה:** טעינת מידע רב עשויה לעכב את הממשק.
* **ביצוע:** כל פעולה מתבצעת בתהליך נפרד עם עדכון פקדים (כגון ListBox  דרך Invoke  לשמירה על תגובתיות הממשק.

**פיצ'ר : GuessTheMoment**

* **מיקום:** פונקציה  buttonGuessTheMoment\_Click().
* **סיבה:** טעינת תמונות כבדות מאלבומים עלולה לעכב את הממשק.
* **ביצוע:** שימוש בתהליך נפרד לטעינת התמונה ועדכון PictureBox  באמצעות Invoke.

**פיצ'ר DownloadAlbum** :

* **מיקום:** פונקציה buttonDownloadAlbum\_Click().
* **סיבה:** הורדת אלבומים עשויה להעמיס על הממשק.
* **ביצוע:** תהליך נפרד מבצע את ההורדה ועדכון הפקדים מתבצע דרך Invoke.