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

# Top 4 Popular Photos

This feature shows the user his “Top popular” photos.

We scan all the user photos, in all of the users’ albums, and give each photo a score. The score is calculated 60% from the likes on that photo, and 40% others from the number of comments on that photo.

After the scan is complete, we show the user his top 4 photos.

Later, the user can select a photo, from the photos shown under the feature panel, and like it and/or comment the photo.

This ranking feature does not exist on Facebook, getting the “Top liked” or “Top commented” photos is done manually.

# News Feed Text Filtering

The feature gives you the ability to filter posts in your news feed.

The user has the option to create different filter groups.

For each group the user can add text items which will be removed from the news feed.

Furthermore, the user has the option to remove unnecessary item and groups and can decide whether to enable or disable the feature on the news feed.

For Example, if the user doesn’t want to see any series spoilers, the user will create a “spoilers” group and will add item such as: “Game of Thrones”, “Marvel”, etc.

### תבנית מס' 1 – Singelton

* סיבת הבחירה / שימוש בתבנית:

המחלקה שממשת את תבנית ה- Singelton הינה PopularPanelMgt. המחלקה אחראית על הצגת התמונות הנבחרות על המסך.

הסיבות לבחירת התבנית:

1. לאפשר גישה של מחלקות אחרות (במקרה שלנו SpecialPictureBox שמציגה תמונה) לצורך עדכון התמונה הקיימת שנבחרה, בלי ליצור תלות דו-כיוונית בינהן.
2. לאפשר יצירה רק של מופע אחת של המחלקה (אין צורך ליצירת שני פנלים שמציגים אותו דבר במערכת).

* אופן המימוש:

המימוש הינו Thread Safe כדי לתמוך בטעינה אסינכרונית, LazyCreation כדי לייצר את המחלקה רק אחרי שהנתונים והקיטלוג של התמונות הסתיים.

ניתן למצוא את המימוש במחלקה PopularPanelMgt (PopularPanelMgt.cs).

המימוש הינו ב Property Instance שבודק אם הכבר נוצרה מחלקה כזו, במידה ולא אז נועלים את המנעול (למנוע יצירת מספר מחלקות בכמה תהליכונים נפרדים) ומחזירים את המופע היחיד של המחלקה.

### תבנית מס' 2 – Prototype

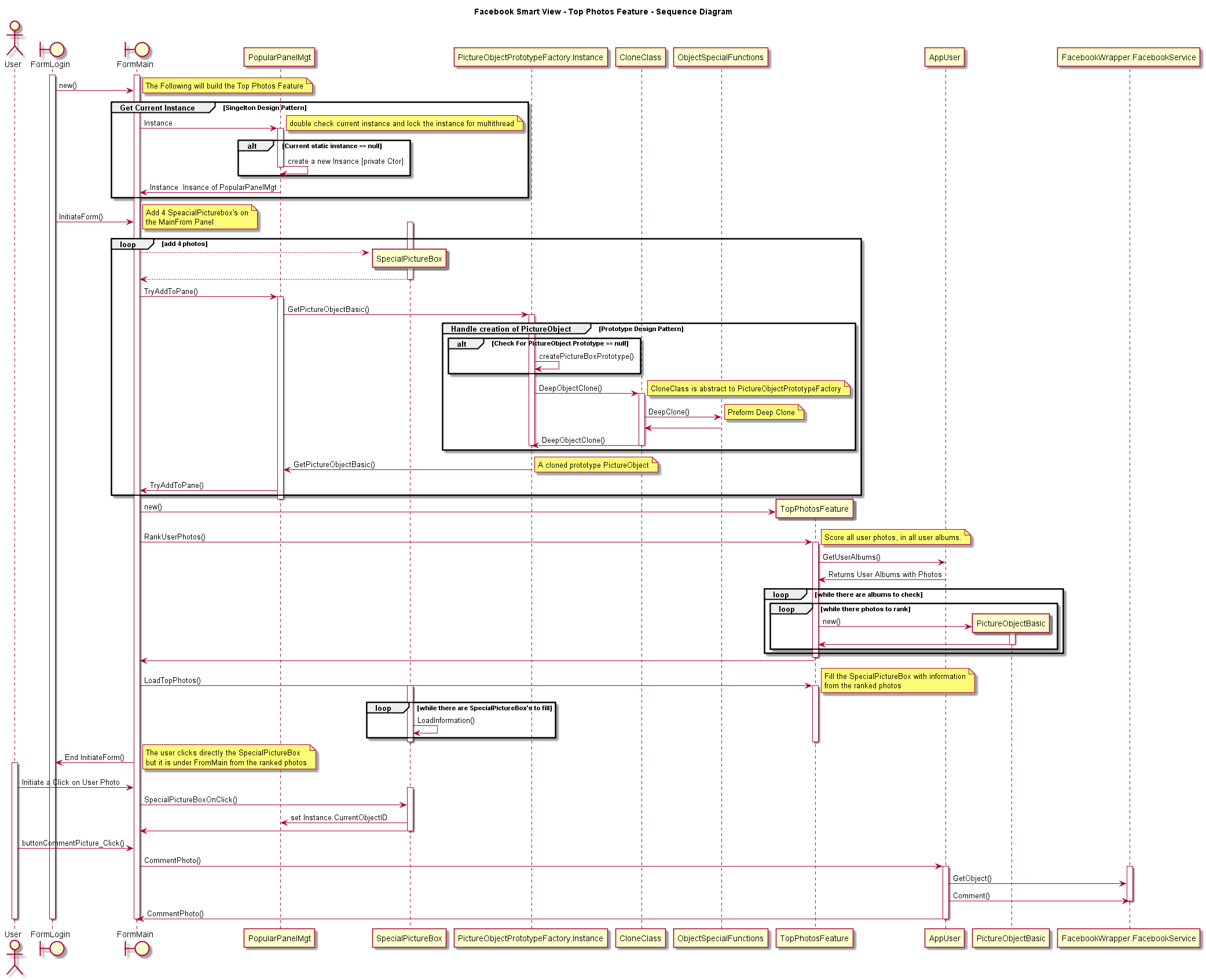
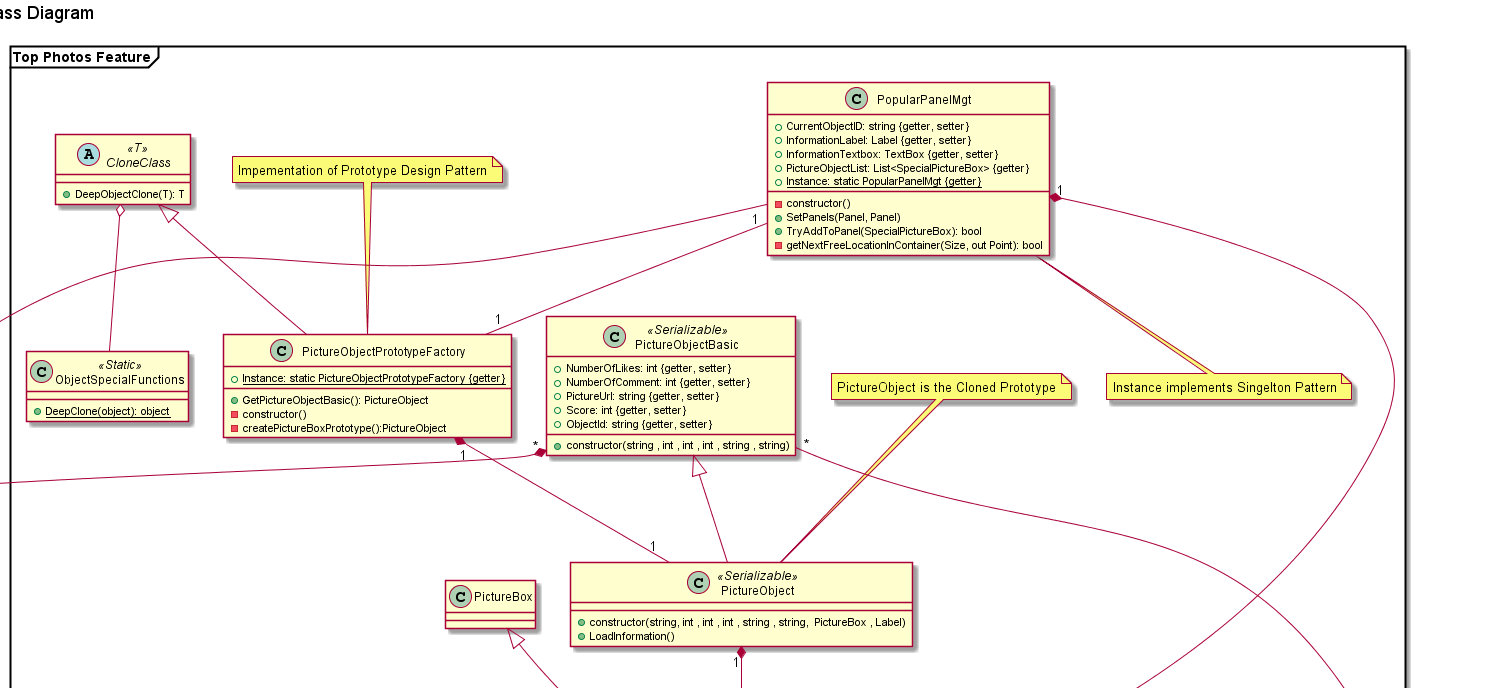
* סיבת הבחירה / שימוש בתבנית:

בזמן טעינת הנתונים של התמונות, נוצרים הרבה אובייקטים דומים, שמכילים נתונים שונים במעט אחד מהשני, במקום ליצור מחדש את אובייקטי התמונה (שמיכילים Meta Data על התמונות), אנחנו יוצרים אובייקט בסיס ומעתיקים אותו כל פעם שיש בקשה ליצירת תמונה חדשה, אחר כך רק משנים את הפרטים של האובייקט.

אנחנו לא משתמשים במתודת Clone הסטנדרטית (וממשים Clone), אלא פונים למחלקה חיצונית שאחרית לייצר אובייקטי תמונה (ואחראית על העתקה שלהם).

המחלקה האבסטרקטית CloneClass מחזירה אובייקט משוכפל מסוג T. המחלקה ObjectSpecialFunctions מממשת שכפול של אוביקט מסוג T, האוביקט חייב להיות Serializable.

המחלקה PictureObjectPrototypeFactory מייצרת אובייקטי PictureObject ע"י שכפול של PictureObject האחד והיחיד (הפרוטוטיפ הראשון שנוצר).

* אופן המימוש:
  + PictureObjectPrototypeFactory מחלקה שמממשת את CloneClass<PictureObject>, היא אחראית להחזיר העתק של הפרוטיטיפ של PictureObject. היא מחזירה מופע חדש של PictureObject על ידי העתקת המופע היחיד שלו. (לא ממומש באמצעות Hashmap בגלל שיש רק סוג אחד של אובייקט).
  + CloneClass (ראה מימוש ב ClonePrototype.cs) הינה מחלקה אבסטרקטית שנותנת מימוש גנרי להעתקת אובייקטים. משתמשת במימוש DeepClone של ObjectSpecialFunctions
  + ObjectSpecialFunctions ממשת את פונקצית הDeepClone באמצעות Serialize. (ראה מימוש ב ClonePrototype.cs)
  + הוגדר על המחלקה PictureObject, PictureObjectBasic להיות [Serializable]
* Sequence Diagram (עבור שני הDesign Patterns )
* Class Diagram (עבור שני הDesign Patterns )

### תבנית מס' 3–Composite

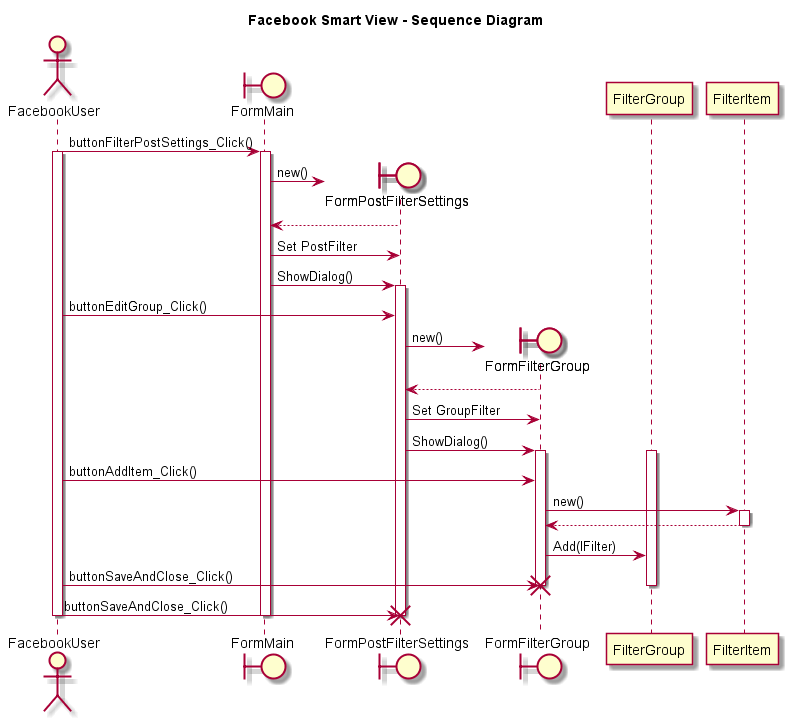
* סיבת הבחירה / שימוש בתבנית:

בפיצ'ר מספר 2 אנחנו מבצעים סינון של הnews feed עבור מילים שנבחרו על ידי המשתמש. הוספנו את היכולת להוסיף תתי קבוצות. לדוגמא, ניצור קבוצת חגים. תחת חגים ניצור item רגיל שמכיל את המילה "שבועות" וניצור item קבוצה בשם "חגים אהובים" שמכילה את המילה "חנוכה". הפיצ'ר יסנן את המילה "חנוכה" וגם את המילה "שבועות". כלומר, עלינו להתייחס לitem ולgroup בצורה דומה ובכך שאנחנו מפעילים את המתודה שמבצעת את המיון אנחנו לא צריכים לבדוק את הtype שנמצא ברשימת המילים לסינון של המשתמש.

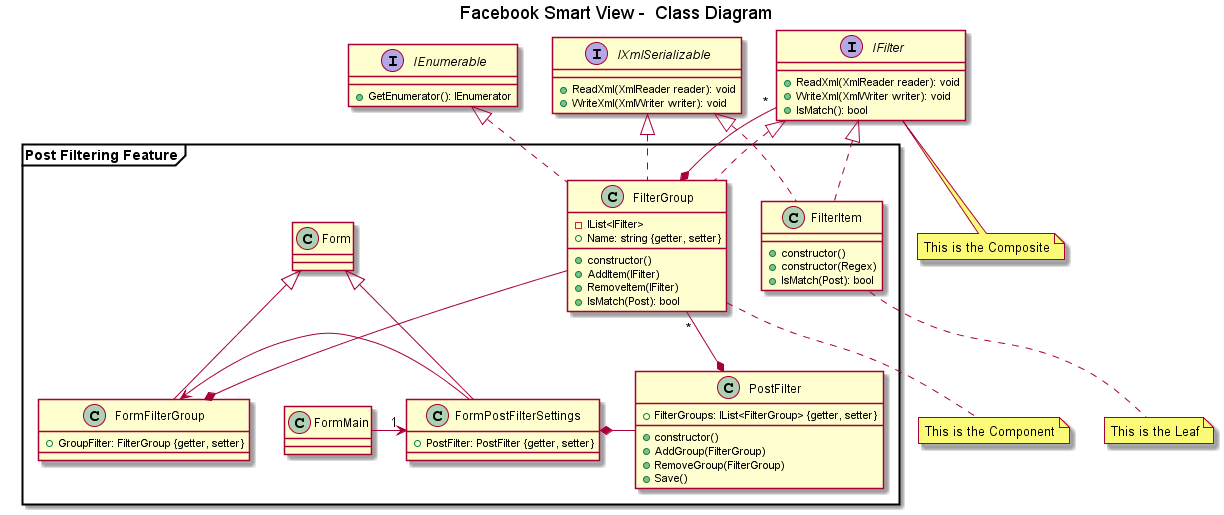
* אופן המימוש:

1. יצרנו interface IFilter שמחייב מימוש של IXmlSerializable ושל IsMatch Method(ראה IFilter.cs). משמש בתור הComponent
2. FilterGroup class מממש את IFilter ומכיל IList<IFilter>. בנוסף, מימוש הIsMatch שלו מבצע IsMatch על כל item ברשימה. (ראה FilterGroup.cs). משמש בתור הComposite
3. FilterItem class מממש את IFilter ומכיל את המימוש שמכיל את ביצוע הסינון המילים בnews feed(ראה FilterItem.cs). משמש בתור הLeaf.

* Sequence Diagram



* Class Diagram



### עבודה עם DataBinding

* יצרנו Two Way Data Binding עבור רשימת הPosts המופיעים בnews feed.
* הData Source מקושר לאובייקט FacebookWrapper.ObjectModel.Post.
* השתמשנו במספר שדות מתוך האובייקט(CreatedTime,PictureURL, From.Name) והם נמצאים בFormMain בpanelPostDetails.
* בFormMain.cs ניתן לראות שעבור הlistBoxNewsFeed בחרנו את הDataSource שיצרנו.
* בFromMain.cs בfetchNewsFeed() method ניתן לראות שאנחנו מבצעים את החיבור בין הpostBindingSource לPost Collection.(ראה שורה 95)

### עבודה אסינכרונית

* כאשר האפליקציה נפתחת, מוצגת ליוזר LoginForm על מנת להכניס את הסיסמא שלו. במידה והlogin הצליח אז תטען האפליקציה הפייסבוקית עם נתוניו האישיים.
* טעינת המידע כוללת מידע אישי, Newsfeed, טעינת תמונות וחישוב תמונות חשובות(פיצ'ר שפותח). אך טעינה זו לוקחת זמן ואנחנו מעוניינים שזמן ההמתנה של היוזר יהיה מובן, כלומר שהיוזר ידע מה קורה כרגע באפליקציה.
* לטובת זה, יצרנו FormLoader שמציג את סטטוס הטעינה ברגע נתון. הFormLoader רץ באותו הthread של הLoginForm ומפעיל thread חדש שבו נטען הFormMain, שם מתבצעות לקיחת הנתונים משרת הפייסבוק וחישוב הנתונים) – (ראה FormLogin.cs שורה 76)
* בנוסף, על מנת למקבל תהליכי טעינה אז קיימים תתי תהליכים נוספים בזמן טעינת המידע האישי(ראה FormMain שורה 81).
* בנוסף, ידוע שלא ניתן לשנות property של רכיבי UI מתוך thread שונה מהthread בו הוא נוצר ולכן על מנת שנוכל לשנות את תצוגת הFormLoader מתוך הthread של הFormMain היינו צריכים להשתמש במתודת הInvoke של FormLoader (ראה FormLogin.cs שורה 85, 90)