**תיאור קצר על הפיצ'רים שבחרנו לממש בתרגיל מספר 1 :**

1. הצגת סטטיסטיקה למשתמש לפי רמת פעילות .  
   בפיצ'ר זה נתנו למשתמש את האופציה לבחור את הפעילות המינימלית והמקסימלית שלו בפייסבוק . המשמש מקבל פירוט על איזה חודש הפעילות שלו הייתה מקסימלית \ איזה חודש הפעילות שלו הייתה הכי מינימלית ומה מספר הפרסומים שהעלה באותם חודשים .למשתמש נפתח באפליקציה טופס חדש עם שני כפתורים אשר לפי בחירת המשתמש מוצגת לו התוצאה המבוקשת.
2. הפיצ'ר השני הוא הצגה למשתמש את הפוסט שקיבל את המקסימום \ מינימום \ ממוצע של לייקים.   
   פיצ'ר זה מקנה למשתמש את הידיעה לגבי הפוסטים שלו , מי הכי פחות אהוב ומי הכי אהוב מבחינת הלייקים . למשתמש נפתח באפליקציה טופס חדש עם שלושה כפתורים אשר לפי בחירת המשתמש מוצגת לו התוצאה המבוקשת.
3. הפיצ'ר השלישי הינו פיצ'ר של תזמן פוסטים לעמוד פייסבוק של המשתמש .  
   הפיצ'ר הזה פותח טופס חדש למשתמש ובו שני מקומות למלא את הקלט המבוקש , הראשון הוא מספר השעות עד להעלאת הפוסט והשני הוא הסטטוס של הפוסט שאותו המשתמש רוצה לעלות . הפיצ'ר הזה מחזיר תגובה למשתמש אם אכן תוזמן בהצלחה והועלה הפוסט או שהפוסט לא הועלה .

**Design Pattern - Static Factory Method**

סיבת הבחירה \ מטרת השימוש–

ראשית , ניתן לראות כי כלל המחלקות בצד הלקוח , הינן תחת אותו אב משותף (Form), כלל המחלקות יורשות מForm ויוצרות משפחה פולימורפית .  
בעקבות העובדה שנוצרה לנו משפחה פולימורפית אנו יכולים להעביר את כל מלאכת היצירה של האובייקטים ממשפחה זו למחלקה נפרדת .  
בחרנו לממש את מחלקת FormFactory כך שתהיה מחלקה נפרדת לחלוטין ותכיל מתודת יצירה , שמתודה זו תיהיה סטטית .   
מבחינה קונספטואלית וreality driven יותר נכון שמחלקה שכלל תפקידה הוא לייצר טפסים תייצר לנו את הטפסים בפרויקט .

באמצעות העובדה שעברנו לשימוש בתבנית העיצוב Factory Method שהיה תחת מחלקת: creational pattern , נמנענו מהיצירה באמצעות שימוש ב: new בצד הלקוח , ובכך הפכנו את הקוד ליותר קל לתחזוקה ופתוח להרחבה ושינויים .

לסיכום , אם בעתיד נרצה להוסיף טופס נוסף לפרויקט או לחילופין להסיר טופס לא נצטרך לשנות את הקוד רק להחליף את ה FormFactory שיהיה רלוונטי למערכת הקיימת.

אופן המימוש –

את המימוש של תבנית העיצוב , ניתן לראות במחלקת FormFactory, במחלקה זו קיימת רק מתודה אחת סטטית (שכן לא נרצה לייצר את המחלקה בכל פעם שנרצה ליצור טופס ) CreateForm מתודה זו מקבלת שני פרמטרים : הפרמטר הראשון הוא הEnum ששמו : eFormType אשר בEnum זה ניתן למצוא את כלל הטפסים ושמותיהם שניתן לייצר בפרויקט , הפרמטר השני שהמתודה מקבלת הוא הFormMain .

תהליך יצירת הטפסים נעשה במחלקת Program ומחלקת FormMain .

המחלקות בקוד שמייצגות את ה"שחקנים" בתבנית –

Client - FormMain .

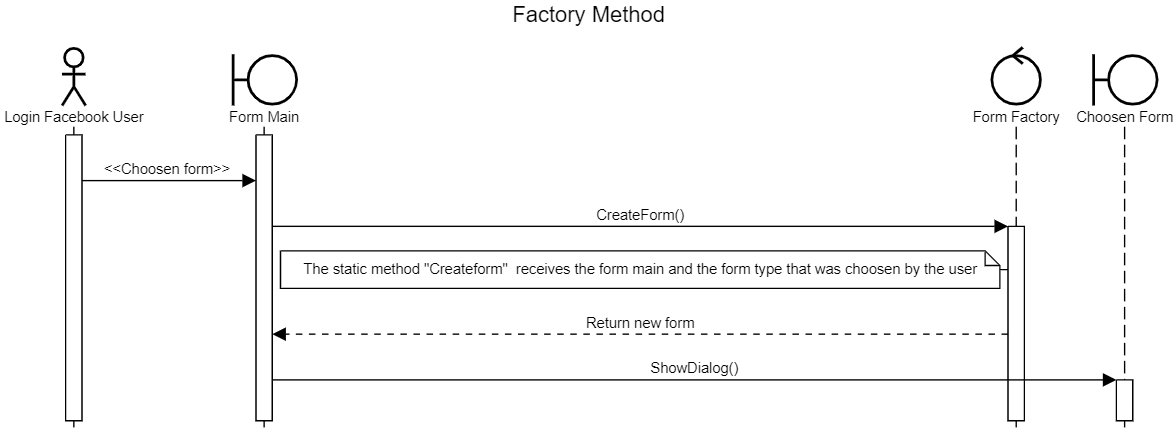
Static Factory Class - FormFactory .

Abstract Product – Form .

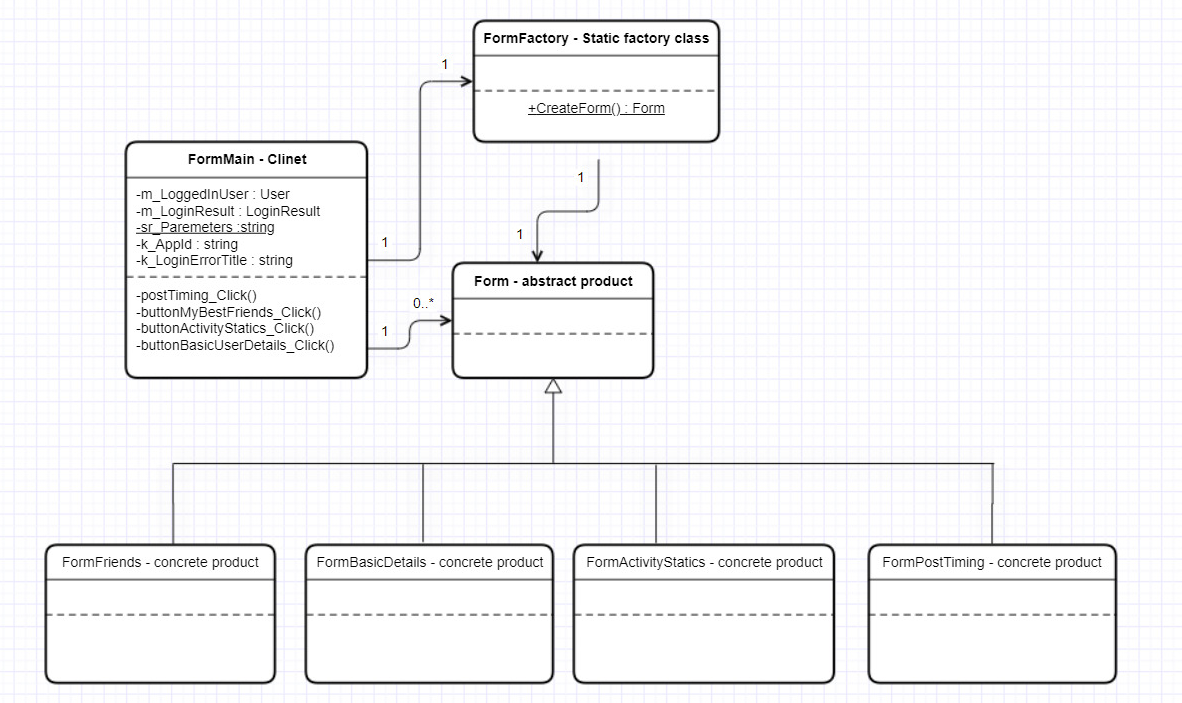
Concrete Products - FormActivityStatics,FormFriends, FormPostTiming

**Diagrams**

Sequence Diagram :



Class Diagram :



**Design Pattern Singleton**

סיבת הבחירה \ מטרת השימוש –

מחלקת LogicManager הינה מחלקה שמשמשת לנו כ Singleton וכ Façade.

בחרנו לעשות מחלקה זו כ Singleton מכיוון שזוהי מחלקה שאנו לא רוצים ממנה יותר ממופע אחד ורוצים לאפשר לכלל הרכיבים במערכת לגשת אליה , תפקיד המחלקה הוא לשמש כממשק המחבר בין הUI לבין הצד הלוגי של הפרויקט .  
במידה ולא היינו יוצרים את מחלקת LogicManager כ Singleton ומאפשרים ליצור מחלקה זו יותר מפעם אחת היה יכול לקרות 2 בעיות שאנו רוצים לחסוך :

1. אם בכל המופעים של המחלקה תמיד יש את אותם הנתונים , בזבוז של מקום בזיכרון .
2. אם בכל המופעים של המחלקה לא יהיה את אותם הנתונים – מצב לא תקין , טעות לוגית שאנו לא יודעים היכן מוחזק הנתונים הנכונים.

אופן המימוש –

1. הגדרנו את המחלקה כ sealed כלומר לא ניתן לרשת ממחלקה זו , אם יהיה ניסיון לרשת ממחלקה זו יתקבל בעיית קומפילציה. מכיוון שאם בעתיד מחלקה שיורשת מ LogicManager תרצה לייצר את אח אחד הבנים שיורשים ממנה , יהיו חייבים לייצר את האב : LogicManager .
2. ביצירת Singleton -> הסתרת הבנאי .  
   אם לא היינו עושים c'tor פרטי , כל אחד יכול ליצור אובייקט של מחלקה זו ולכן מראש מנענו את האופציה ליצירה .
3. מתודה שמחזירה את ה Instance של המופע היחיד שייווצר למחלקה זו, מתודה זו מומשה עם Double check lock (רלוונטי להשתמש ב Double check lock שיש קטע קריטי) .

ה Singletonשיצרנו הינו Thread safe .

(מימוש נוסף של תבנית העיצוב Singleton ממומש במחלקת AppSetting )

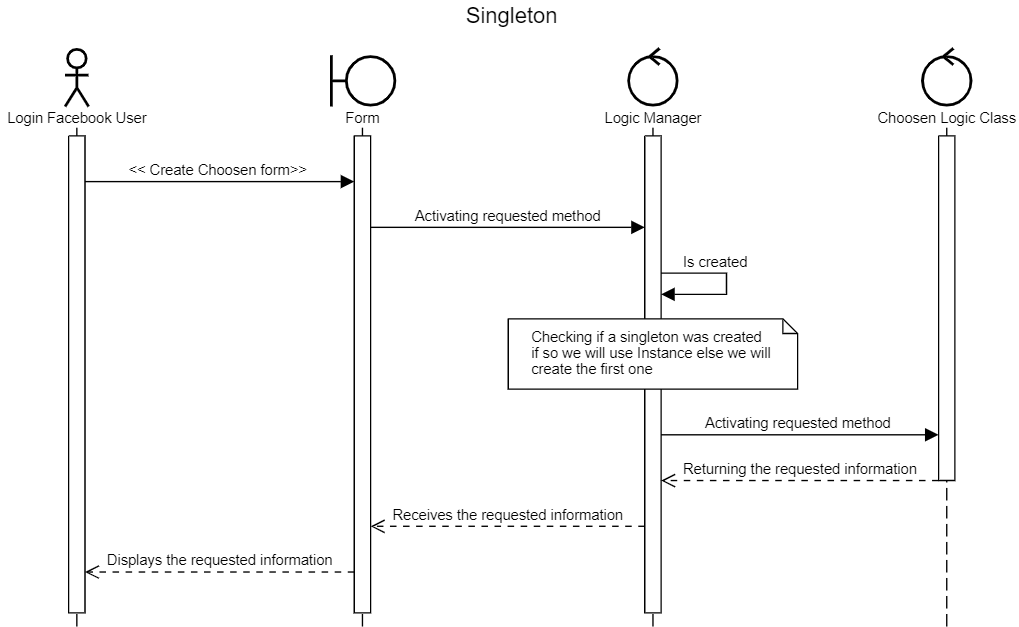
המחלקות בקוד שמייצגות את ה"שחקנים" בתבנית –

Singleton – Logic Manager .

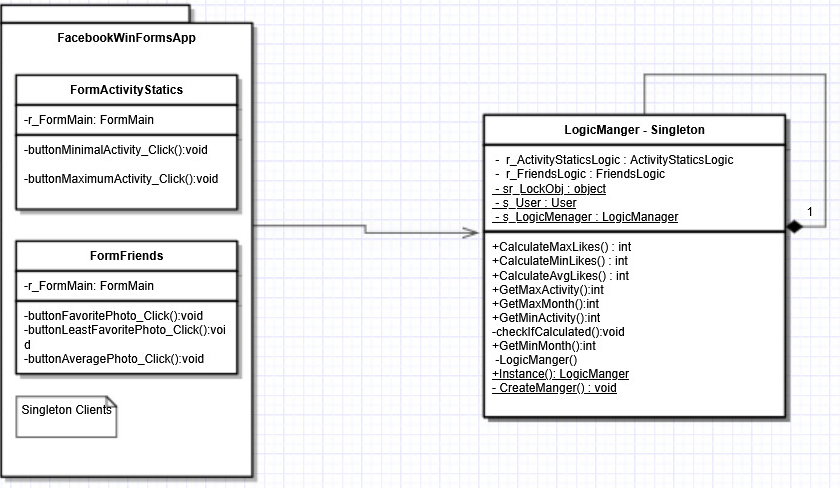
המחלקןת שמשתמשות בסינגלטון – מחלקות הלוגיות כגון : FrindesLogic , ActivityStaticsLogic .

**Diagrams**

Sequence Diagram :



Class Diagram :



**Design Pattern Façade**

סיבת הבחירה \ מטרת השימוש–

Façade הוא רכיב שמשטח ומקל על מי שמשתמש בו לבצע פעולות שבפועל מבוצעות ע"י תתי מערכות אשר נמצאות מאחורי הרכיב הזה, כלומר מסתיר את תתי המערכות מהמשתמש.

הסיבה שבחרנו להשתמש ב Façade היא הצורך בהפרדה בין ממשק המשתמש ללוגיקה שמתבצעת מאחורי הקלעים. ה Façade ממומש במחלקת ה LogicManager שהיא מחלקה מיוחדת בפרויקט אשר נמצאת בטווח בין השכבה של ממשק המשתמש לבין העולם הלוגי של הפרויקט . ה Façade שבחרנו לממש הינו Façade שקוף , היתרון של שימוש ב Façade זה שהUI לא צריך להכיר את כלל מחלקות הלוגיקה אלא להכיר את מחלקת ה Façade והוא מכיר את מחלקות הלוגיקה .

ה Façade משתמשת בעולם הלוגי ובעצם מכירה את העולם הלוגי , מבצעת את הפעולות שהמשתמש צריך בצורה נוחה , פשוטה ומקלה על המשתמש . ה Façade מכיל את המתודות מהעולם הלוגי וכך בכל קריאה של המשתמש לאחת ממתודות אלה יקרא ה Façade ויבצע את הפעולות באמצעות קריאה למתודות בממשק הלוגי, כלומר ממשק המשתמש לא פונה יותר לכל מתודה לוגית ומכיר מחלקות מרובות , אלא מכיר מחלקה אחת שמהווה לו החוצץ ומאפשרת לו את נוחות השימוש , בטיחות ועוד.

אופן המימוש –

הפנייה של המשתמש תיהיה בלעדית למחלקת LogicManager המשמשת כ Façade .

כלל המחלקות הלוגיות יהיו מוסתרות והמחלקות המוחצנות יהיו רק אלה שנמצאות ב Façade ובכך נהנה מכלל היתרונות שיש ל Façade בדומה ליתרונות אינקפסולציה.

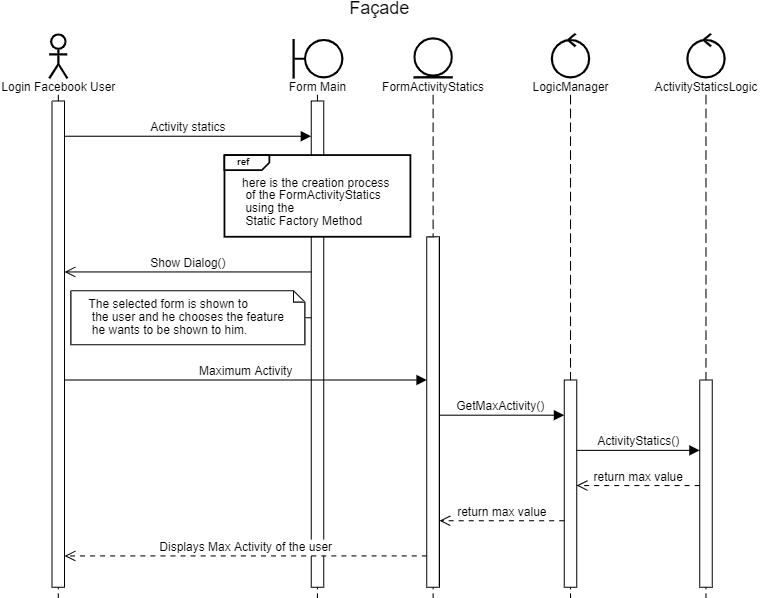
הUI לא מכיר את כלל המחלקות הלוגיות בפרויקט בניגוד לקודם לכן דבר אשר מקל על העבודה מול מחלקת הלוגיקה ובנוסף אם בעתיד נשנה את הפרויקט או נרצה להוסיף מחלקות לוגיות נוספות לא נצטרך לשנות המון מקומות בקוד אלא רק את העובדה שהמחלקה הלוגית הנוספת תכיר את ה Façade.

המחלקות בקוד שמייצגות את ה"שחקנים" בתבנית –

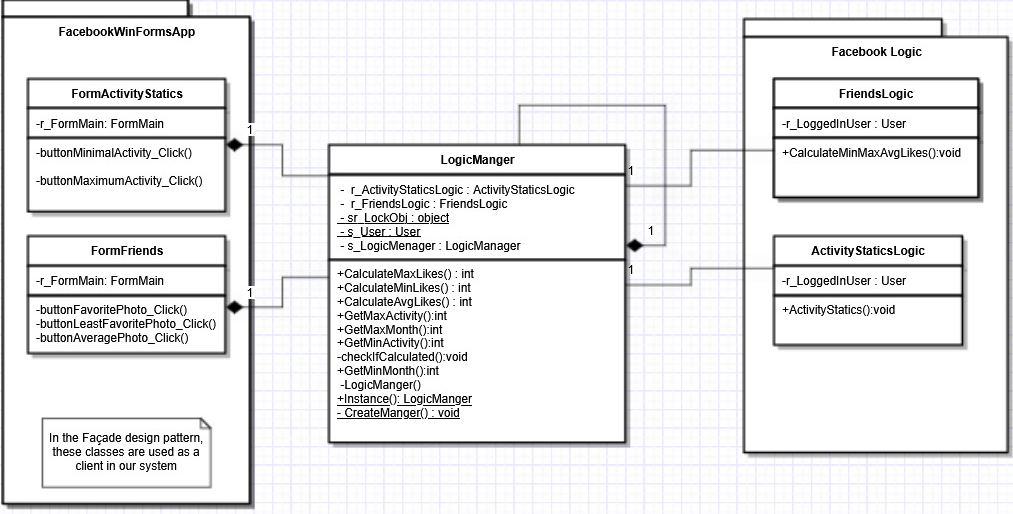
Client - FormMain ,FormFriends,FormActivityStatics

Façade - LogicManger

**Diagrams**

Sequence Diagram : 

Class Diagram :



**עבודה א סינכרונית עם ממשק משתמש**

בפרויקט שלנו השתמשנו בכמה מקומות בעבודה א סינכרונית .

המקום ה1 שבו חל השימוש הוא בFormMain , לפני השימוש בעבודה א סינכרונית , כל פעם שהמשתמש פתח את העמוד והראשי ולאחר מכן לחץ על פיצ'ר , אשר בעקבות לחיצה זו נפתח לו טופס חדש , המשתמש לא יכל להיות אינטראקטיבי עם המסך הראשי עד שלא סגר את הטופס החדש שנפתח לו במקביל .

ולכן בכל יצירת טופס חדש במערכת , הטופס נוצר ב Thread אחר , דבר המקנה לנו אינטראקטיביות , וכך נוכחנו לראות שאכן כאשר המשתמש לוחץ על אחד מן הפיצ'רים המוצגים לו , גם כאשר הטופס החדש נפתח הוא עדיין יכול להיות אינטראקטיבי עם הטופס השני דבר שלא יכל לקרות קודם לכן .

דבר זה מוצג בקוד במחלקת FormMain , במתודות : postTiming\_Click , buttonMyBestFriends\_Click , buttonActivitiyStatics\_Click, buttonBasicUserDetails\_Click .

נוסף על כך בטופס : FormBasicDetails כלל פעולות הfetch בוצעו בThread נפרד, עשינו זאת מכיוון שכאשר אנו מריצים את התוכנית שלנו , לוקח זמן עד שהנתונים יעלו ובזמן הזה המשתמש לא יכול לעשות שום דבר באפליקציה, כלומר האפליקציה נתקעת עד הבאת הנתונים ולכן כדי שבזמן הבאת הנתונים של המשתמש , המשתמש יוכל להיות אינטראקטיבי עשינו את כלל פעולות אלו בThread נפרד.

במתודה button\_Albums\_Click מופעלת המתודה fetchAlbums ב Thread נפרד.

במתודה button\_Events\_Click מופעלת המתודה fetchEvents ב Thread נפרד.

במתודה likedPages\_Click מופעלת המתודה fetchLikedPage ב Thread נפרד.

במתודה buttonViewPosts\_Click מופעלת המתודה fetchPosts ב Thread נפרד.

**Two-Way Data Binding with WinForms**

השתמשנו ב Data Bindingבמחלקת FormBasicDetails במתודה : fetchPosts , במתודה זו נמצא הקוד המקשר בין הDataSource והפוסטים אותם אנן מציגים למשתמש.

הDataBinding מראה למשתמש פרטים על הפוסט אותו הוא בחר כמו : תאריך יצירת הפוסט , השם של הפוסט וכו’.