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

1. המשתמש יכול לסנן את רשימת החברים שלו לפי Birthday, Gender, Hometown, Location. לדוגמה, אם המשתמש בחר לסנן לפי תאריך לידה, אז לאחר הסינון יופיעו ברשימת החברים רק אלו שיש להם אותו תאריך לידה כמו שלו. לאחר מכן, המשתמש יכול לבחור אם ליצור קבוצת פייסבוק חדשה – לתת לה שם ולבחור privacy. באופן אוטומטי ישלחו מיילים להצטרפות לכל החברים שיצאו בסינון.

ניתן למצוא את ה- feature בלחיצה על כפתור Filter friends and create group"".

כמו כן, ניתן למצוא את ה-feature בקוד תחת המחלקה: FormFilteredFriends.

1. המשתמש יכול לבחור פוסט מרשימת הפוסטים שלו ולהציג עליו מידע סטטיסטי כמו מספר התגובות, מספר הלייקים, כמה לייקים הם של נשים וכמה של גברים. לאחר מכן המשתמש יכול לבחור לשמור את הסטטיסטיקה כתמונה.

ניתן למצוא את ה- feature בלחיצה על כפתור "Posts and statistics".

כמו כן, ניתן למצוא את ה-feature בקוד תחת המחלקה: FormPostsAndStatistic.

### שימוש ב- MultiThreading:

FormMain:

פעולת ה-Login של המשתמש לאפליקציה נעשית באמצעות thread נפרד.

FormFilterFriends:

כאשר המשתמש רוצה ליצור קבוצה חדשה, הוא לוחץ על כפתור "create group" ולאחר שיצירת הקבוצה הצליחה נשלחים מיילים לחברים של המשתמש שנבחרו לאחר הסינון עם הקישור לקבוצה החדשה.

פעולת שליחת האימיילים נעשית ב-thread נפרד (במחלקה EmailSender).

### שימוש ב- DataBinding:

FormPostsAndStatistic:

הבאת הפוסטים של המשתמש נעשתה באמצעות Data Binding. המשתמש יכול ללחוץ על fetch posts ולקבל רשימה של כל הפוסטים שלו (DisplayMember -> type). כשילחץ על פוסט ספציפי יוצג לו ה-message, name, created time של אותו הפוסט. בנוסף, המשתמש יכול לשנות את ה-Message של הפוסט (Two-Way).

FormMain:

הבאת האלבומים, קבוצות ואירועים של המשתמש נעשו באמצעות Data Binding.

כאשר המשתמש ילחץ על אלבום/קבוצה/אירוע ספציפי יוצג למשתמש תמונת האלבום/קבוצה/אירוע שלו וכן, יוצג גם זמן היצירה/פרטיות/מיקום בהתאמה.

### תבנית מס' 1 – [Singleton]

* **סיבת הבחירה / שימוש בתבנית:**

בחרנו להשתמש בתבנית Singleton עבור מחלקות הלוגיקה שלנו SingletonFormMainLogic, SingletonFilterFriendsLogic, SingletonPostsAndStatisticLogic מאחר ואלו מחלקות מהן נרצה מופע יחיד בלבד. כל מצב של יצירה נוספת של מופעים היא "בזבזנית" ואינה נדרשת באפליקציה שלנו.

בנוסף, בכל פעם כשניגש לאחת מהמחלקות לעיל, נרצה לקבל את אותו המופע ישירות כדי לקבל את אותם הנתונים על המשתמש שלנו ללא כל שינוי שנעשה בהם במהלך השהות באפליקציה. לא נרצה כל פעם ליצור קומפוזיציה של לוגיקה במחלקות אחרות ולבקש שיעבירו במתודות את המופע הנדרש של הלוגיקה כדי לעבוד עם המתודות. נרצה שכל המתודות ישירות יגשו למופע היחיד הקיים במערכת.

* **אופן המימוש:**

ניתן למצוא את מימוש ה-Singleton במחלקות: SingletonFormMainLogic, SingletonFilterFriendsLogic, SingletonPostsAndStatisticLogic ואת המחלקה Singleton<T> עצמה בקוד.

המימוש נעשה כפי שלמדנו בהרצאה- יצרנו מחלקה סטטית Singelton<T> וכמו כן, מחלקות שיהיו סינגלטוניות ולא יהיה ניתן ליצור מהן יותר ממופע יחיד.

לכל מחלקה סינגלטונית יצרנו private constructor ו- instance יחיד ממנו ניתן לגשת לשאר השדות והמתודות במחלקה.

* **Sequence Diagram**

תמונה שמכילה טקסט, תרשים, מקביל, מסמך

התיאור נוצר באופן אוטומטי

* Class Diagram

|  |  |
| --- | --- |
| **Singleton** | **Client** |
| SingletonFormMainLogic | FacadeFormMain |
| SingletonFilterFriendsLogic | FacadeFilterFriends |
| SingletonPostsAndStatisticLogic | FacadePostsAndStatistic |

תמונה שמכילה טקסט, תרשים, מקביל, תוכנית

התיאור נוצר באופן אוטומטי

### תבנית מס' 2 – [Strategy]

* **סיבת הבחירה / שימוש בתבנית:**

ב-feature של Filter Friends אנו מסננות את רשימת החברים לפי קרטריונים שהמשתמש בוחר ובודקות האם הנתונים עבור קריטריונים אלו זהים עבור המשתמש וחבריו. במידה ויש התאמה, החבר יתווסף לרשימת החברים המפולטרת.

חשבנו כי בעתיד יהיה צורך ארכיטקטוני לשנות את שיטת הסינון (למשל: נרצה לסנן לפי החברים שאינם עומדים בקריטריונים שבחר המשתמש). לכן, בחרנו להשתמש ב- Pattern Strategy ולממש שני strategy שונים לסינון החברים.

* **אופן המימוש:**

ניתן למצוא את המימוש ואת המחלקות הקשורות במחלקות הבאות: IFilterStrategy, EqualFeatureFilterStrategy, NotEqualFeatureFilterStrategy, SingletonFilterFriendsLogic, FormFilterFriends

מימשנו Interface שבו מוגדרות ארבע המתודות לפיהן מתבצע הסינון. כמו כן, יצרנו שני classes שיהוו את שני סוגי ה-strategy השונים ויממשו את הממשק. במחלקה SingletonFilterFriendsLogic יצרנו משתנה בשם FilterStrategy מסוג הממשק שממנו נגשנו ל-4 המתודות שהגדרנו. בנוסף, במחלקה זו יש מתודה בשם FetchFriendsByUserFilterFeaturesSelection() שם נעשה סינון החברים באמצעות ה- strategy שיצרנו.

את ההגדרה של ה- Concrete strategy הספציפי אותו נרצה באפליקציה שלנו הגדרנו במחלקה FormFilterFriends.

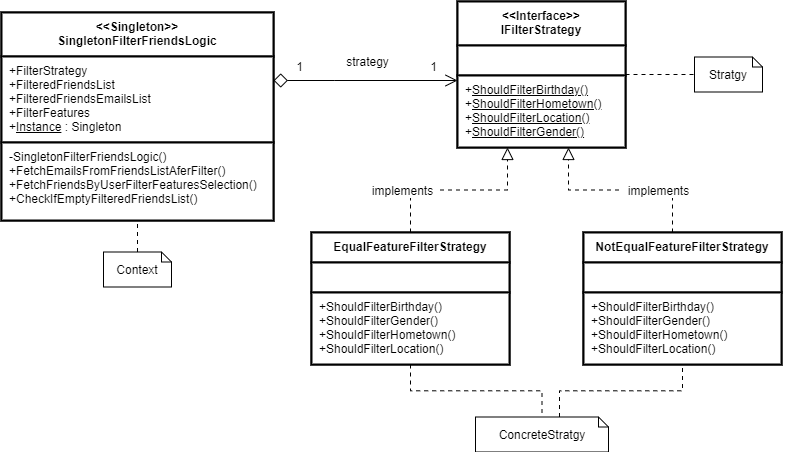
* **Sequence Diagram**

תמונה שמכילה טקסט, צילום מסך, גופן, מקביל

התיאור נוצר באופן אוטומטי

* **Class Diagram**

|  |  |
| --- | --- |
| **Strategy** | IFilterStrategy |
| **Context** | SingletonFilterFriendsLogic |
| **Concrete Strategy** | EqualFeatureFilterStrategy, NotEqualFeatureFilterStrategy |



### תבנית מס' 3 – [Proxy]

* **סיבת הבחירה / שימוש בתבנית:**

יש לנו שני רכיבים כאשר אחד מהם זורק exceptions. רכיב B משתמש ברכיב A אשר זורק exceptions ולכן, החלטנו לממש את שיטת ההתגוננות ברכיב נפרד כדי שהוא יהיה reuseble ולא נצטרך לממש try ו-catch בכל פעם כשנרצה להשתמש ברכיב A.

השתמשנו ב-pattern proxy אשר מציע פתרון לבעיה. השתמשנו ב-proxy מסוג protective proxy.

* **אופן המימוש:**

יצרנו מחלקה בשם ProxyFormMainFacade שתחקה את המחלקה FacadeFormMain מאחר וה-facade שלנו לא מממש שום ממשק השתמשנו ב-Class Proxy כלומר, ה-Proxy יורש ממחלקת ה-facade וה-client (FormMain) משתמש ב-proxy כאילו הוא ה-façade ללא שום שינוי.

בתוך ה- facade יש 3 מתודות אשר זורקות :exceptions (GetLoggedInUserAlbums, GetLoggedInUserGroups, GetLoggedInUserEvents). המתודות ניגשות לאלבומים, לקבוצות ולאירועים של המשתמש. מאחר וההרשאה הזו אינה בתוקף עוד יש סכנה של זריקת שגיאות (כמו שקרה לנו). לכן, ב-Proxy שיצרנו עטפנו את המתודות האלו ב-try ו-catch וכך ה- client (FormMain) יוכל לגשת באותו אופן כמו שהוא ניגש בfacade אך באופן בטוח.

* **Sequence Diagram**

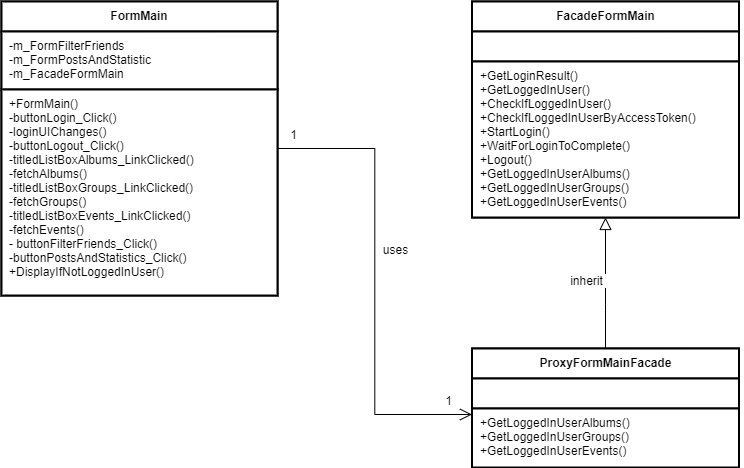
תמונה שמכילה טקסט, תרשים, מקביל, תוכנית

התיאור נוצר באופן אוטומטי

* **Class Diagram**

[class diagram שמתארת את המחלקות שלכם שמעורבות בתבנית (תיאור מלא שכולל Properties ו- Methods) והיחסים ביניהם (תאור מלא עבור כל יחס כפי שלמדנו בכיתה). עבור כל מחלקה שלכם, כיתבו מי המקבילה שלה (אם יש כזו) בתבנית (Pattern) שבחרתם

|  |  |
| --- | --- |
| **Proxy** | ProxyFormMainFacade |
| **Real Subject** | FacadeFormMain |
| **Client** | FormMain |



### תבנית מס' 4 – [Facade]

* **סיבת הבחירה / שימוש בתבנית:**

יש לנו 3 Forms באפליקציה שלנו. נרצה עבור כל מסך ליצור facade אשר יגש למחלקת הלוגיקה של ה-Form ויהווה "צינור" עבור המתודות והשדות הנדרשים בשביל מחלקות ה-UI.

שימוש ב-facade מעודד בטיחות שימוש- הסתרה מה-Client את מתודות הלוגיקה וחשיפה רק של המתודות הרלוונטיות לו. דבר זה, מגן והופך את השימוש לבטיחותי יותר. בנוסף, זה מעלה את נוחות השימוש.

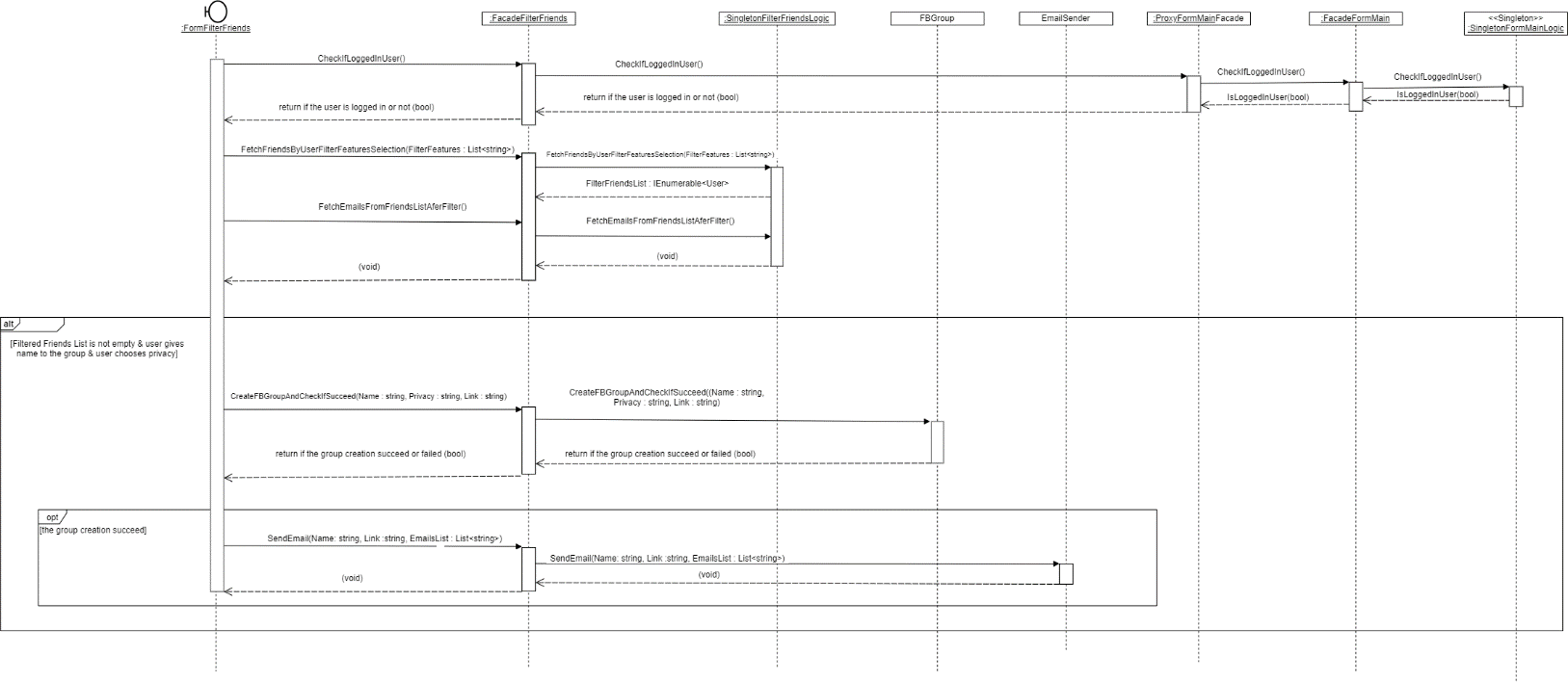
רצינו "לשטח" את העולם הלוגי ולספק לממשק המשתמש את מה שהוא צריך מהלוגיקה בצורה פשוטה ובטוחה.

* **אופן המימוש:**

עבור כל form יצרנו מחלקת façade שניגשת למחלקת הלוגיקה האחראית על אותו מסך ודולה ממנו את המתודות הרלוונטיות.

ה-Forms: SingletonFilterFriendsLogic, SingletonFormMainLogic, SingletonPostAndStatisticLogic.

ה- facades : FacadeFilterFriends, FacadeFormMain, FacadePostAndStatistic.

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

|  |  |  |
| --- | --- | --- |
| **SubSystem** | **Facade** | **Client** |
| SingletonFilterFriendsLogic, EmailSender, FBGroup, ProxyFormMainFacade | FacadeFilterFriends | FormFilterFriends |
| SingletonFormMainLogic | ProxyFormMainFacade  מתחזה ל-facade:  FacadeFormMain | FormMain |
| SingletonPostAndStatisticLogic, ProxyFormMainFacade | FacadePostAndStatistic | FormPostsAndStatistic |

תמונה שמכילה טקסט, תרשים, תוכנית, מקביל

התיאור נוצר באופן אוטומטי