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

* [הפיצ'ר הראשון שבחרנו לממש : **BEST FRIENDS** כאשר המשתמש בוחר בפיצ'ר זה הוא מקבל את רשימת החברים שלו ממויינים לפי מספר הלייקים ומספר ה comments שכל אחד מחבריו עשה לו (כמובן במקרה בו כן הייתה גישה לרשימת החברים ול "LikeBY"), כאשר לכל חבר יש ניקוד המתקבל באופן הבא: עבור like יקבל 2 נקודות ועבור comment 2.5 נקודות. לבסוף נמיין את רשימת החברים לפי שדה הניקוד כאשר הניקוד הגבוהה ביותר ידורג ראשון וכן הלאה.

בנוסף לאחר הצגת רשימת ה best friends ל list box יוכל המשתמש לבחור בכל אחד מהחברים הופיעים שם ולראות עבור כל אחד את כמות ה likes ה comments ואת הניקוד שקיבל (כמובן הכל בהנחה שפייסבוק API יאפשר גישה לנתונים אלה)]

* [הפיצ'ר השני שבחרנו לממש: **CATEGORY PHOTOS** בפיצ'ר הזה המשתמש יכול לבחור מבין 2 הקטגוריות המופיעות לו ב combo box (countries ו- date) עבור הבחירה ב country נחלץ מתוך שרת הפייסבוק את כל אחת מהמדינות בהן הועלתה כל אחת מהתמונות, ועבור הבחירה date נחלץ משרת הפייסבוק את כל התאריכים בהן הועלו כל אחת מהתמונות. יוצג למשתמש

ב list box המדינות או התאריכים (תלוי בבחירתו). בנוסף המשתמש יוכל לבחור כל אחד מבין רשימת המדינות/תאריכים ולפי בחירתו תוצג ב list box נוסף רשימת התמונות השייכות לאותה בחירה. וגם כאן יוכל המשתמש לבחור בתמונה מסויימת להצגת התמונה עצמה.

לדוגמה: המשתמש בוחר countries ---> מוצגות לו כל המדינת בהן הוא העלה תמונה ----> המשתמש בוחר ב Thailand -----> מוצגת לו רשימת התמונות שהועלו בתאילנד ----> המשתמש בוחר בתמונה מסויימת להצגתה ---> מוצגת לו התמונה ]

### תבנית מס' 1 – [Singlton]

* סיבת הבחירה / שימוש בתבנית:

[אם בעתיד נצטרך להוסיף רכיב חדש למערכת המשתמש ב UiControler נרצה לעשות זאת ללא שינוי במערכת, כלומר בקוד הקיים של המחלקה UiControler – נרצה שהרכיב החדש יעשה שימוש באובייקט שנוצר ב UiControler ביוזמתו. כמו כן נרצה שיהיה שימוש באובייקט יחיד בכדי למנוע חוסר תאימות מידע, שכן האובייקט שומר את המידע המוצג למשתמש וכך נמנע מבזבזנות או באג לוגי]

* אופן המימוש:

[את המימוש ניתן למצוא בקוד במחלקה UiControler שזוהי בעצם המחלקה הסינגלטונית

תיאור המימוש:

א. הוספנו private constractor

ב. יצרנו property Instance שמממשת מתודת get סטטית שתפקידה לייצר מופע אם עדיין לא קיים (כמובן השתמשנו ב lock{ } כדי למנוע באגים בשימוש ב multi threading) אם המופע קיים אז היא תחזיר אותו.

ג. הגדרנו שדה שהוא מופע של המחלקה במחלקה עצמה.

ד. במחלקה FormMain שעושה באובייקט שימוש אנו פונים לפרופרטי ובה המתודה get לקבלת האובייקט. כיוון שזוהי הפנייה הראשונה, תתבצעה יצירה (new()) של האובייקט. כך בעצם כל רכיב חדש שנוסף למערכת פונה למתודה זו ביוזמתו לאותה מחלקה בה נמצא האובייקט המשותף.]

* Sequence Diagram

**:UiControler**

**FormMain:**

getInstance

opt

s\_Instance **:UiControler**

[s\_Instance == null]

new()

return the Instance

* Class Diagram -

A diagram of a user interface

Description automatically generated

### תבנית מס' 2 – [Factory Method]

* סיבת הבחירה / שימוש בתבנית:

[נרצה שה client יהיה קריא יותר ע"י כך שננקה אותו מ new() וif else ובכך התכנית תהיה מודולרית יותר, כלומר נרצה להפריד בין משתמש האובייקט לבין יוצר האובייקט. עבור הפיצ'ר category photos יש לכל קטגוריה מחלקה משלה בה יש לוגיקה הממשת אותה. מחלקות אלה הן משפחה פולימורפית, ועבור כל מחלקה כזו אנו מקצים אובייקט משלו ע"י new() לכן נרצה להפריד בין השימוש באובייקט לבין היצירה. כמו כן אם בעתיד נרצה להוסיף קטגוריה נוספת יש יתרון בתחזוקתיות של הקוד שכן יהיה ברור יותר שיצירת האובייקט תתבצע במחלקה factory. ובנוסף כך זה קורה גם במציאות- נרצה שהתכנית תשקף כמה שיותר את המציאות. ]

* אופן המימוש:

[הוספנו מחלקה CategoryPhotoManagerFactory שתפקידה לייצר אובייקט המייצג את המחלקה של הקטגוריה לפיה ימויינו התמונות (בפיצ'ר category photos) על פי בחירת המשתמש בקטגוריות המופיעות ב combo box של אותו פיצ'ר. כאשר המשתמש בוחר קטגוריה באירוע אנו מחלצים את שם הקטגוריה שביקש ולפי מחרוזת זו אנו יודעים במחלקה המייצרת איזה אובייקט לייצר. כמו כן אותן מחלקות המייצגות את הקטגוריה שנבחרה ממשות את האינטרפייס ICategoryPhotoManager וכך גם כל קטגוריה שנרצה להוסיף בעתיד. לאחר יצירת האובייקט ניתן לעשות בו שימוש להפעלת לוגיקת המיון לפי הקטגוריה שנבחרה. ]

* Sequence Diagram

[ בעמוד הבא: ]

A screenshot of a computer

Description automatically generated

* Class Diagram

A diagram of a company

Description automatically generatedבדיאגרמה הבאה המחלקה UiController היא המקבילה של המחלקה Client,  
המחלקה CategoryPhotoManagerFactory היא המקבילה של המחלקה StaticFactoryClass,  
האינטרפייס ICategoryPhotoManager הוא המקביל של AbstractProduct,  
המחלקה CountriesPhotoManager היא המקבילה של המחלקה ConcreteProduct1,  
המחלקה CreatedDatePhotoManager היא המקבילה של המחלקה ConcreteProduct2

### תבנית מס' 3 – [Class Proxy]

* סיבת הבחירה / שימוש בתבנית:

[נרצה שהclient יוכל לבצע פעולות בכמה חוטים מבלי לשנות את הקוד.

כלומר, שנוכל להוסיף חוטים שיעדכנו את המסך של המשתמש מבלי לשנות את הקוד בכל מקום שבו מעדכנים את המידע או מבצעים כל אינטרקציה עם האובייקטים של Win Forms.

כלומר, אנו דורשים שירות נוסף שלא קיים לאובייקטים של Win Forms.  
לכן בחרנו לממש עבור אותם אובייקטים את design pattern Proxy

בנוסף, אם בעתיד נרצה להוסיף שירות נוסף לאובייקטים של Win Forms נוכל לבצע זאת מבלי לשנות את ה client ובכך הקוד שלנו יהיה יותר קל לתחזוקה.]

* אופן המימוש:

[הוספנו פרוייקט חדש בשם ProxyComponents ובו קיימים 3 מחלקות חדשות: ButtonProxy, ListBoxProxy, PictureBoxProxy

ב client נחליף את השימוש (היצירה) בButton, ListBox, PictureBox בהתאמה במחלקות החדשות.

מחלקות אלו יורשות בהתאמה מהמחלקות Button, ListBox, PictureBox  
ובכך "מחקות" את המחלקות המקוריות וכן נותנות שירות נוסף אשר מאפשר לבצע פעולה כלשהי באופן בטוח למערכת מרובה חוטים (ThreadSafe).  
כלומר, ניתן יהיה לבצע פעולה כלשהי שהגדרנו מכל חוט באמצעות שימוש במטודה Invoke אשר תבצע את הפעולה בחוט המקורי שיצר את אותה מחלקה.  
לדוגמה, עבור המחלקה ButtonProxy נוכל לבצע את הפעולה myButton.Text = "Login" מכל חוט כיוון שמימשנו מחדש את הProperty Text כך שיתמוך במערכת מרובה חוטים באמצעות מטודה Invoke ובאמצעות Action]

* Sequence Diagram

[

A diagram of a button

Description automatically generated]

* Class Diagram

בדיאגרמה הבאה המחלקה Button היא המקבילה של המחלקה RealSubject,  
המחלקה ButtonProxy היא המקבילה של Proxy  
והמחלקה FormMain היא המקבילה של המחלקה Client

A diagram of a software application

Description automatically generated with medium confidence