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

* **Calendar:**  
  מטרת הפיצ'ר: ריכוז נתונים של תאריך בלוח שנה והצגה מסודרת למשתמש, עם יכולת ניהול נתונים.   
  יומן המציג:  
  > ימי הולדת  
  > אירועים  
  > רשימת משימות  
  לפי תאריך שנבחר על ידי המשתמש.  
  המשתמש בוחר תאריך מתוך לוח שנה שמוצג לו. כאשר הוא לוחץ על התאריך המבוקש מוצגים בפניו כל הנתונים שציינו. המשתמש יכול לנהל את רשימת המשימות שלו ע"י:  
  א. הוספת משימות חדשות  
  ב. סימון וי על משימות שבוצעו, סימון זה יוביל למחיקת המשימה מן הרשימה.    
  כל המשימות שהמשתמש מוסיף ליומן נשמרות באפליקציה. סגירת האפליקציה וכניסה חוזרת תציג את כל המשימות של המשתמש.  
  הקוד של הפיצ'ר נמצא במחלקות: Calendar.cs ו- CalendarForm.cs .
* **Style your Facebook page:**  
  מטרת הפיצ'ר: כיום חשיבות רבה לנראות של עמוד חברתי. מספר העוקבים של עמוד מושפע רבות מהנראות והעיצוב שלו. ככל שעמוד יהיה יותר מושך לעין, מעניין וייחודי כך הסיכוי שמספר העוקבים שלו יגדל.  
  הפיצ'ר כולל מספר הגדרות שמטרתן שיפור נראות העמוד ומתן פן ייחודי לבעל העמוד.  
  ההגדרות הן:  
  > שינוי צבע העמוד  
  > שינוי צבע הכפתורים בעמוד  
  > שינוי צבע שם המשתמש    
  > הוספת מוטו אישי   
  > בחירת תמונות מועדפות של המשתמש שמייצגות אותו באופן הטוב ביותר, התמונות יוצגו בעמוד של המשתמש כאשר כל כמה שניות תמונה מתחלפת. המשתמש יכול לשנות בכל עת את התמונות, להוסיף תמונות או להסיר תמונות.  
  כל ההגדרות הללו נשמרות באפליקציה. סגירת האפליקציה וכניסה חוזרת תציג את כל ההגדרות האחרונות שבוצעו.

הקוד של הפיצ'ר נמצא במחלקות: SwappingPhotosBox.cs , MainForm ו- ImageGalleryAlbum

**עבודה א-סינכרונית:**  
בחרנו להראות את השימוש בעבודה א-סינכרונית במחלקה הראשית שלנו- MainForm, בה סוג הפניות לפייסבוק הינו המגוון ביותר. המגוון הרחב מאפשר להציג בצורה הטובה ביותר את היתרונות של ריבוי threads. פניות לפייסבוק לוקחות זמן, ובזמן זה ה-Form אינו אינטראקטיבי ופוגע באיכות ממשק המשתמש. ביצענו את העבודה הא-סינכרונית ע"י הפעלת המתודות שכוללות פנייה לפייסבוק ב-thread נפרד וכך אפשרנו חווית משתמש טובה יותר, שמאפשרת אינטראקציה מלאה עם ה-form מרגע הופעתו על המסך. (המשתמש אינו ממתין עד לטעינת המידע). הפעלת כל מתודה שלוקחת זמן ב-thread נפרד מאפשר ל-thread הראשי שהפעיל את ה-MainForm, להיות זמין לאינטראקציה אפשרית מצד המשתמש, ה- MainFormשמוצג למשתמש מגיב, ניתן להזזה ומאפשר לחיצה על פקדי ה-MainForm.  
   
המתודות בהן נעשה שימוש א-סינכרוני:  
> fetchUserInfo : טעינת אודות המשתמש (יום הולדת, סטטוס מערכת יחסים, עיר מגורים וכו'..)  
> fetchUserPosts : טעינת הפוסטים של המשתמש   
> fetchEvents : טעינת האירועים של המשתמש  
> fetchCheckIn : טעינת הצ'ק אין של המשתמש  
> fetchGroups : טעינת הקבוצות אליהן המשתמש שייך  
> fetchFriends : טעינת רשימת החברים של המשתמש  
> loadFavoritePhotosToSwappingPhotosBox  
  
 נשים לב, שהמתודה- loadFavoritePhotosToSwappingPhotosBox אינה כוללת פניה לפייסבוק, אך היות ובזמן הצגת ה-MainForm מתבצעת טעינה של התמונות מועדפות על המשתמש מהקובץ אל ה- SwappingPhotosBox, והטעינה יכולה לקחת זמן, מימשנו גם את המתודה הזו בעזרת עבודה א-סינכרונית ובכך שיפרנו משמעותית את חווית המשתמש.

**עבודה עם Data Binding:**  
עשינו שימוש ב-Data Binding במקומות הבאים:  
> במחלקה MainForm במתודה fetchUserPosts.  
 יצרנו panel שמציג את ה-message של כל post ב-listBoxPost, שינוי ה-message ב-panel יעדכן  
 באופן אוטומטי את המידע ב-post הנוכחי ב-listBoxPost. לחיצה על post ב- listBoxPostתחשוף   
 את ה-panel ותציג ב-message את המידע מפייסבוק או במידה ובוצע עדכון במהלך התוכנית, תציג את   
 העדכון האחרון.  
> במחלקה MainForm במתודה mainForm\_Load, קישרנו בין ה-appSettingsBindingSource שיצרנו  
 למחלקה AppSettings עבור שתי מטרות:  
 1. קישרנו בין RememberUser ב-AppSettings ל-property: CheckState של  
 rememberUserCheckBox.  
 2. קישרנו בין LastMoto ב-AppSettings ל-property: Text של labelMoto.  
שני השימושים הללו מאפשרים עדכון אוטומטי של שדות ב- AppSettings. כך, אין צורך לעדכן את השדות באופן ידני לאחר כל שינוי (שיפור תחזוקתיות). ובסגירת האפליקציה וטעינתה מחדש, כל המידע האחרון ששונה באפליקציה יישמר בקובץ ולאחר מכן ייטען ל-MainForm ויוצג בפקדים הרלוונטיים.

### **תבנית מס' 1 – [Facade]**

* סיבת הבחירה / שימוש בתבנית:

בחרנו לממש מחלקת facade מרכזית אשר תסתיר מאחוריה את הממשק הלוגי של   
המערכת ותפשט את השימוש בו עבור ממשק המשתמש. ה-facade מרכז את כל הנדרש לניהול הצד הלוגי של המערכת וכתוצאה מכך:  
1. ה-client-ים השונים לא נדרשים להכיר את כל תתי המערכות בצד הלוגי ולרוב פשוט  
 פונים ל-facade באמצעות פעולה פשוטה, והוא מבצע את הפונקציונליות הנדרש.  
2. הקוד ב-client-ים הרבה יותר קריא ומסודר.  
3. תחזוקתיות- אם בעתיד נרצה לשנות משהו בממשק הלוגי, השינוי יהיה נקודתית ב-  
 facade ולא במספר מקומות ב-UI, דבר אשר יקל על התחזוקה. בנוסף, אם בעתיד נרצה  
 להחליף את ממשק המשתמש, לא נצטרך לממש מחדש את התקשורת מול הצד הלוגי   
 ונוכל להשתמש ב-facade.

* אופן המימוש:

המחלקה AppManager הינה ה-facade והיא נמצאת ב-AppLogic. יש מופע אחד של AppManager וכל form במערכת מחזיק instance שלו, על מנת לבצע פונקציונליות בצד הלוגי של המערכת. ה-AppManger מחזיק מופעים של מחלקות לוגיות הנדרשות לניהול המערכת (למשל- מופע של המחלקה AppSettings). כך, כל טופס במערכת שזקוק למשהו מהצד הלוגי, פונה ל-AppManger והוא מנתב אל הפעולה הלוגית הנדרשת ומחזיר תשובה לממשק המשתמש במידת הצורך.

ה-facade בחלקו אטום ובחלקו שקוף בכדי לבצע פשרה בין היתרונות של סוגי ה-facade השונים. כלומר, המחלקה Calendar אינה חשופה כלל, המחלקה AppSettings חשופה באופן חלקי אך ה-facade משטח ומפשט את העבודה מולה. ה-User חשוף במלואו   
(Legacy component).

ה-client:

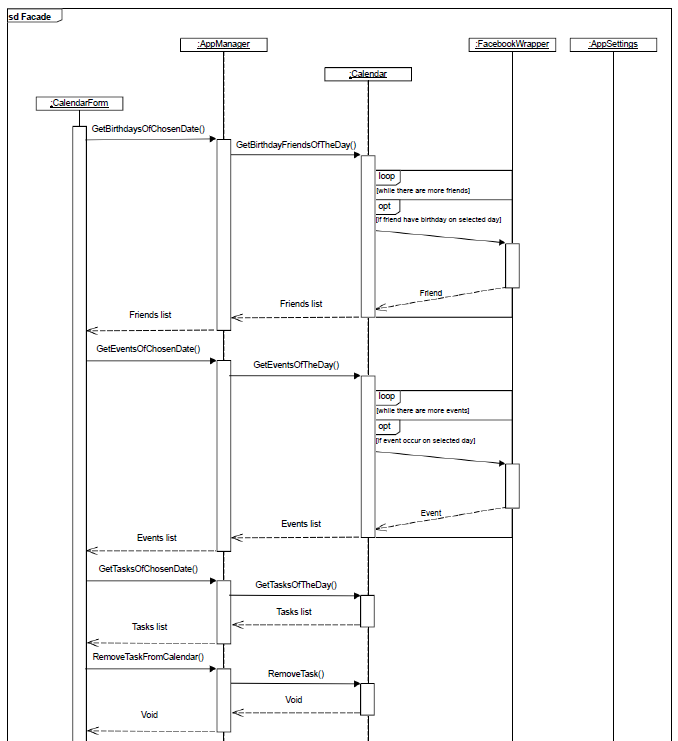
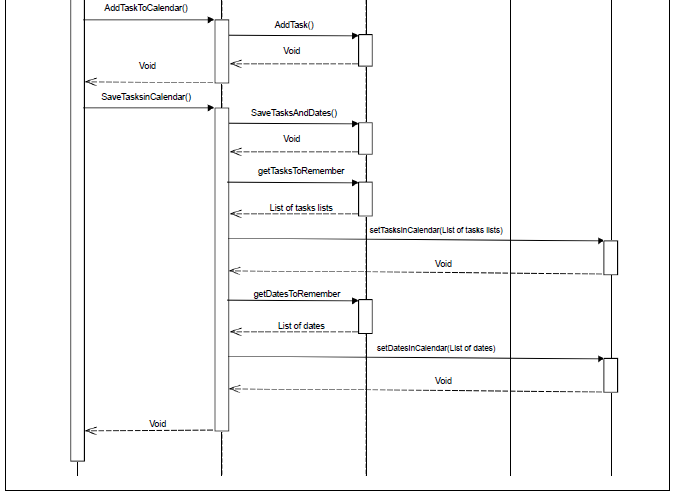
* LoginForm
* MainForm
* CalendarForm
* ImageGallery
* ImageGalleryAlbum

ה-facade:

* AppManger

תתי המערכות הינן:

* AppSettings
* Calendar
* User השייכת ל- FacebookWrapper
* Sequence Diagram

### Class Diagram

### 

### **תבנית מס' 2 – [Singelton]**

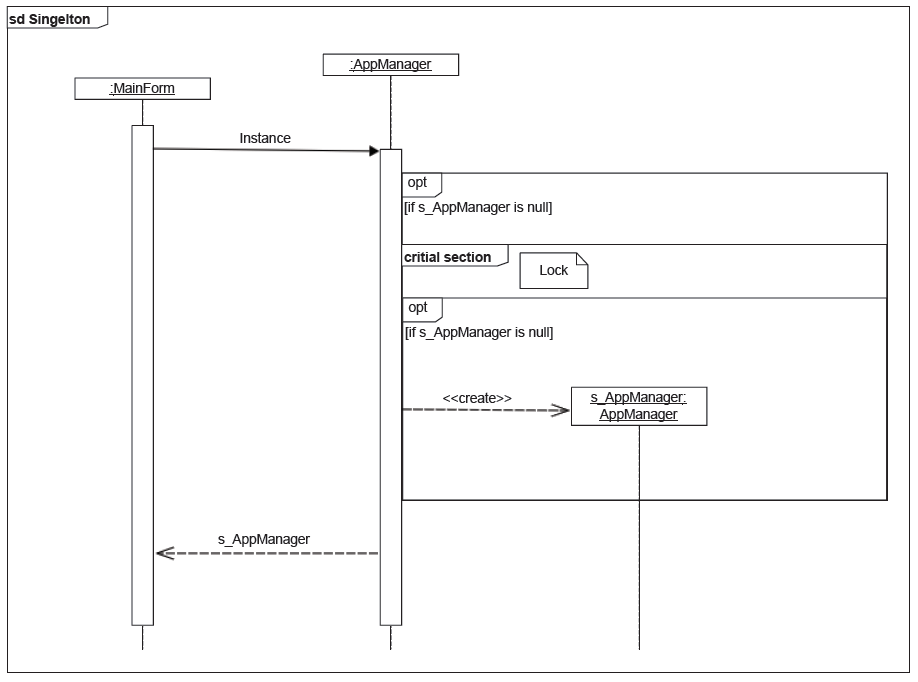
* סיבת הבחירה / שימוש בתבנית:

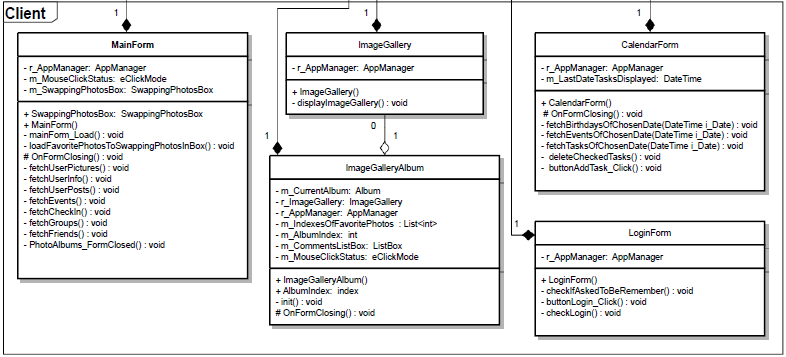
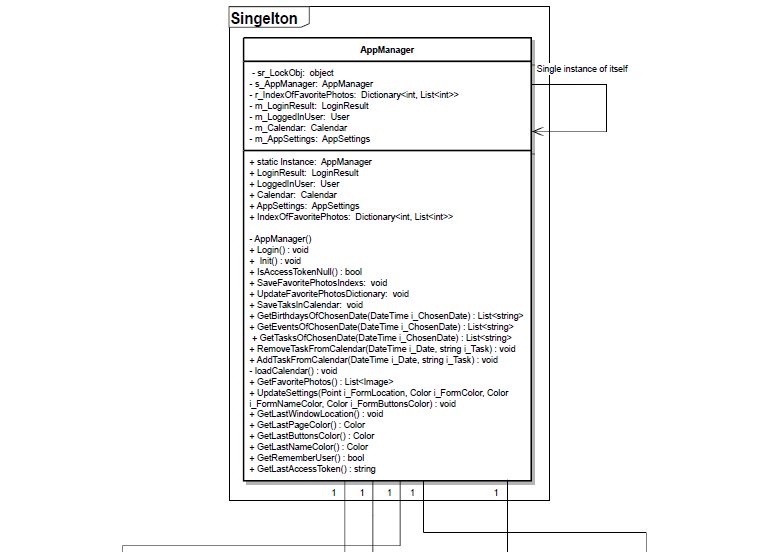
בחרנו לממש את מחלקת AppManager כ-singelton. מחלקת AppManager מחזיקה את כל המידע הנדרש לתפעול הצד הלוגי של המערכת עבור משתמש יחיד. לכן, נרצה שיהיה מופע אחד של מחלקת AppManager במערכת, וכן שכל מחלקה ב-UI תוכל להשתמש בו ביוזמתה.  
במידה ויהיו כמה מופעים ייתכן ושני טפסים שונים יחזיקו במופע שונה של AppManager וכתוצאה מכך יוצג מידע שגוי (למשל- פרופיל של משתמש x ולחיצה על אלבומי התמונות תוביל לצפייה באלבומים של משתמש y). לכן, יהיה מופע יחיד של AppManager, וכל אחד מהטפסים במערכת יחזיק instance שלו.

* אופן המימוש:

מחלקת AppManager מכילה מופע סטטי פרטי של עצמה המאותחל ל-null. המופע חשוף לכל המבקש באמצעות מתודה public static כאשר במידה ומישהו מעוניין במופע של המחלקה- לפני שהיא מחזירה את המופע, היא בודקת אם המופע בעודו null ובמידה וכן, היא יוצרת אותו. בדיקה זו ויצירת המופע רק במידת הצורך מבוצעת באמצעות מנגנון ה-double-check-lock. בזכות הבדיקה והעובדה שה-constructor של המחלקה הינו private, המחלקה יכולה להבטיח שאכן יהיה לה מופע יחיד והוא המופע בו היא מחזיקה.   
בנוסף, על מנת להבהיר שהמחלקה לא ניתנת להורשה (ה-constructor הוא private) כתבנו בחתימת המחלקה sealed.

* Sequence Diagram



* Class Diagram  
  

### **תבנית מס' 3 – [Proxy]**

* סיבת הבחירה / שימוש בתבנית:

בחרנו לממש את pattern proxy בכדי לשפר את הממשק של pictureBox בכך שיציג תמונה שצורתה עגולה. מאחר ואנו מקבלים מפייסבוק את תמונת הפרופיל כמחרוזת URL, אנו יכולים להעביר מחרוזת זו לאחת המתודות המוצעות במחלקת pictureBox (load, loadAsync, imageLocation) והתמונה תוצג אך, ה-pictureBox יציג את התמונה כמרובעת ואילו אנו נרצה שהתמונה תהיה עגולה. כלומר, הממשק של שני הרכיבים מתאים אך התוצאה שאנחנו מקבלים לא מספיק טובה לצרכינו ולכן, נשדרג את הממשק. למקרה ובעתיד המערכת תצטרך להציג תמונות עגולות נוספות (למעט תמונת הפרופיל)- נממש את הלוגיקה שמשנה את צורתה של התמונה לעיגול ברכיב נפרד במערכת – RoundPictureBoxProxy. מחלקה זו מתחזה ל-pictureBox וכך ה-client עובד בצורה זהה מול pictureBox ומול roundPictureBox.

* אופן המימוש:

מימשנו את RoundPictureBoxProxy כ-class proxy. נשים לב, כי מבחינה טכנית – pictureBox לא מממש שום interface וכן, הוא legacy component כך שלא נוכל לשנות עובדה זו. לכן, אין שום Interface שיכול להיות בסיס פולימורפי משותף ל-pictureBox ול- RoundPictureBoxProxyולכן הדרך היחידה ש- RoundPictureBoxProxyיהיה פולימורפי ל-pictureBox היא באמצעות הורשה.   
ב- RoundPictureBoxProxyאנו דורסים את מתודת OnPaint (המקורית), כאשר הלוגיקה החדשה משתמשת במתודת OnPaint המקורית (base.OnPaint) אך, בנוסף לכך מממשת את לוגיקת עיגול התמונה. RoundPictureBoxProxy מפעילה את המתודה OnPaint בכל פעם שמתרחש שינוי כלשהוא ב-visual display.

ה- Client :

* MainForm

הממשק :

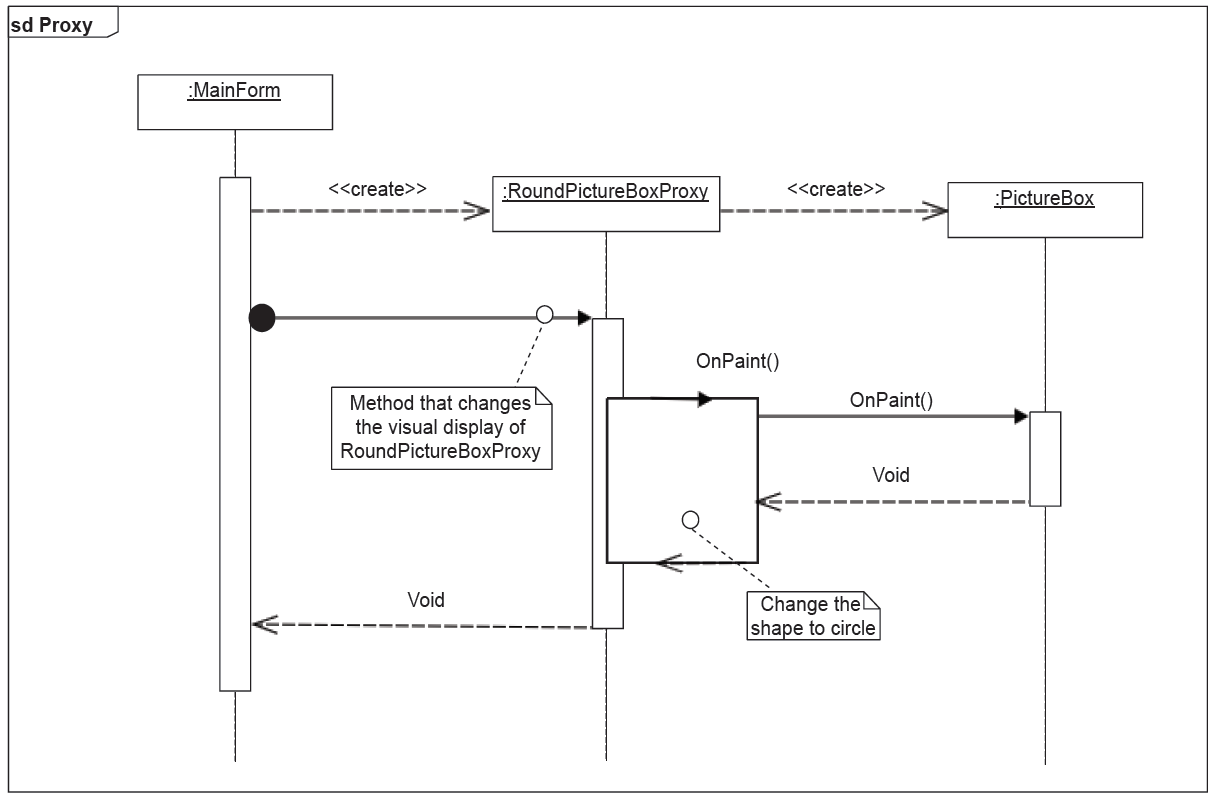
* מתודת OnPaint.

ה- Real Subject :

* PictureBox

ה- Class Proxy :

* RoundPictureBoxProxy.
* Sequence Diagram



* Class Diagram