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

* **פיצ'ר ראשון – מציאת רשימת 5 החברים הכי טובים שלך על פי נתונים מהפייסבוק:**

פיצר זה מנתח את כל התגובות והלייקים שחברים של אותו משתמש רשמו או עשו לו

המערכת תשלוף את כל האלבומים,תמונות,פוסטים,ואירועים של המשתמש שמופיעים בפייסבוק.

לאחר מכן מנתחת כל אחד מהתונים שהיא שלפה ובודקת מי מהחברים הביא לייק וכתב תגובה או היה נוכח באירוע (הן לפוסט,תמונות,אלבומים,אירועים)

לאחר שקיבלנו את כל האנשים וסכמנו על ידי אלגוריתם חישוב (כל תגובה או לייק מחושב שונה על פי הקטגוריה)

הרשימה עוברת למיון ומחזירה רק את ה5 החברים שדורגו הכי גבוה (בתור הרשימה כמובן יש את פרטי המשתמש + הדירוג שלו שהוא קיבל)

לאחר שהכל בוצע הפרטים שהוחזרו ברשימה מופיעים בטבלה שמציגה את שם + שם משפחה של החבר + תמונת פרופיל שלו + כמה הציון שהוא קיבל.

הפעולות שמומוש בפיצר הזה:

\* מציאת כל החברים והצגתם בטבלה

\* מציאת כל הלייקים שחברים שלך עשו על אלבומים,תמונות,פוסטים

\* מציאת כל התגובות שחברים שלך עשו על אלבומים,תמונות,פוסטים

\* מציאת כל האירועים שאתה הייתה בהם

\* מציאת כל האירועים שחברים שלך היו בהם

\* מציאת כל האירועים שאתה וחברים שלך הייתם נוכחים באותו מקום

\* חישוב וסכימה של הציון שכל חבר קיבל

מיקום בפרויקט: קובץ UI - נקרא בשם BestFiveFriends.cs הוא נמצא בתוך תקייה Panels

קובץ שדרכו המערכת עובדת(Façade) – קובץ הלוגי היחידי שהפיצר מכיר שאיתו הוא עובד הוא נקרא FacebookApplication.cs והוא נמצא בתוך תקייה בשם Façade.

קובצי לוגיקה שמאחורי הFaçade -

\* FetchInformations - קובץ הנמצא בתוך תקייה BestFriendsFolder אחראי על שליפת המידע מהשרת ושמירתו בתוך ServerInfomartion

\* UserRate - קובץ הנמצא בתוך תקייה BestFriendsFolder אחראי על יצירת אובייקט ששומר את המידע אודות המשתמש והניקוד של אותו משתמש

קבצי מאגר מידע -

\* ServerInformation - קובץ הנמצא בתוך תקייה ServerDetails אחראי על שמירה של כל המידע שנשלף מהשרת לשימוש חוזר (למנוע בקשות מיותרות לשרת ולשיפור ביצועי מערכת)

\* LoginInformation - קובץ הנמצא בתוך תקייה ServerDetails אחראי על שמירה של כל המידע אודות ההתחברות לשרת כמו הרשאות וקוד האפליקציה , ובנוסף שומר את היוזר שהתחבר אחרון.

* **פיצ'ר שני – ניתוח לפי תאריך שהמשתמש בוחר והצגת מידע מהפייסבוק בצורה טבלאית:**

פיצ'ר זה מאפשר למשתמש לבחור כל תאריך שהוא רוצה

והמערכת תעבור על כל חבריו של המשתמש ותחפש אם אחד מחבריו של המשתמש עשה  
פעולה מסוימת (העלה תמונה/פוסט/חוגג יומהולדת) בתאריך הנבחר לפי יום וחודש (לא לפי אותה שנה).  
  
הפעולות שמומשו והפיצ'ר מזהה אותם והם מוצגות בצורה טבלאית הם:

* מציג את כל החברים של המשתמש אשר חוגגים יום הולדת בתאריך שהמשתמש בחר
* מציג את כל החברים של המשתמש אשר בתוך אחד מהאלבומים שלהם נמצאת תמונה שהועלתה בתאריך שהמשתמש בחר
* מציג את כל החברים של המשתמש אשר העלו פוסט בתאריך שהמשתמש בחר

מיקום בקוד: קובץ SmartCalendar.cs שנמצא בתוך התיקייה Panels

החלק הלוגי נמצא בעיקר תחת תיקיית Façade בקובץ FacebookApplication.cs)

### תבנית מס' 1 – Command

* סיבת הבחירה / שימוש בתבנית:

הסיבה בגללה בחרנו לממש Pattern Command במערכת שלנו נבעה   
מכך שנתקלנו בבעיה הנדסית שהיא אומרת שאם בעתיד נרצה לבצע פעולות נוספות שיופיעו אצלנו בCombobox אנחנו נצטרך לשנות בכמה מקומות שונים בקוד דבר שיקשה עלינו מבחינת תחזוקתיות, קריאות, ופוטנציאל לבאגים, כלומר אנחנו מחפשים פיתרון הנדסי אשר תפתור לנו את הבעיה, לכן בחרנו ב command שזה מה שהוא בדיוק פותר , הוא מציע אפשרות לייצירת תפריט פעולות שאם בעתיד נרצה להוסיף או להסיר פעולה כל מה שנצטרך זה לשנות במקום אחד בלבד והוא יעשה זאת לבד אוטומטית.

בPanel SmartCalendar יש לנו תפריט בחירה המוצג למשתמש ב ComboBox בשם comboBoxCalendarSelection רצינו לשפר את התפריט.

הבנו שיש לנו פה בעיה הנדסית שבמידה ובעתיד נרצה לעדכן את התפריט נצטרך לבצע כמה שינויים בקוד ורצינו לייצר מצב שכדי לעדכן את התפריט נצטרך להוסיף שורת קוד אחת ופונקציה שעושה את הפעולה של ה MenuItem החדש שהוספנו.  
כך כשנרצה לשנות את התפריט שלנו נוכל למנוע באגים שיכולים לקרות עקב כך כי שכחנו להוסיף שורה בעוד מקום שהיינו צריכים לשנות אותה כדי שהפונקציה שרצינו שתפעל תעבוד.  
התבנית מצמצמת משמעותית את הזמן שייקח למתכנת עתידי לעדכן את התפריט, האופציה לבאג שנובעת מעדכון התפריט בכמה מקומות, תחזוקתיות של המערכת,   
הקוד יותר קריא ומסודר.   
\* עשינו שימוש ב Databinding כדי לקשר בין התפריט שייצרנו לבין ה comboBoxCalendarSelection שנמצא בתיקיית Panel ב SmartCalendar

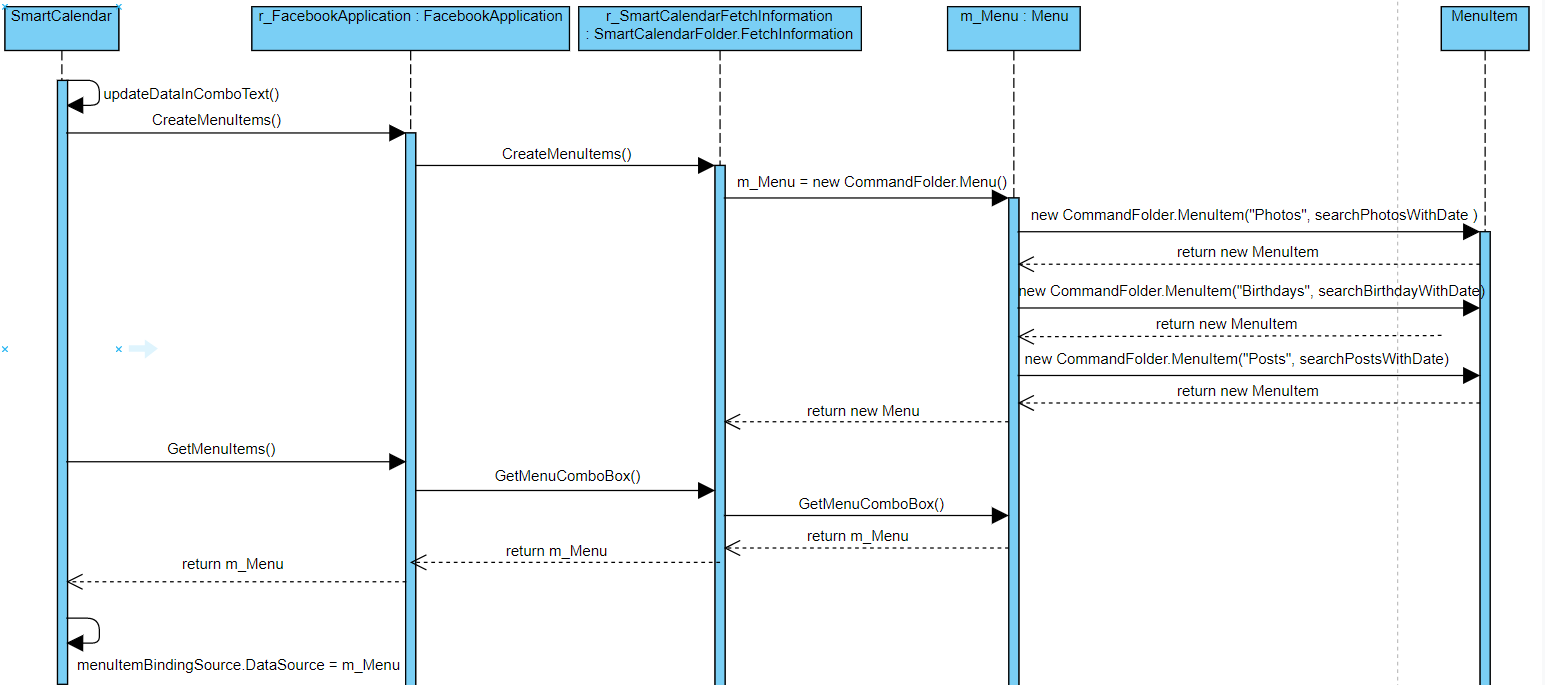
* אופן המימוש:

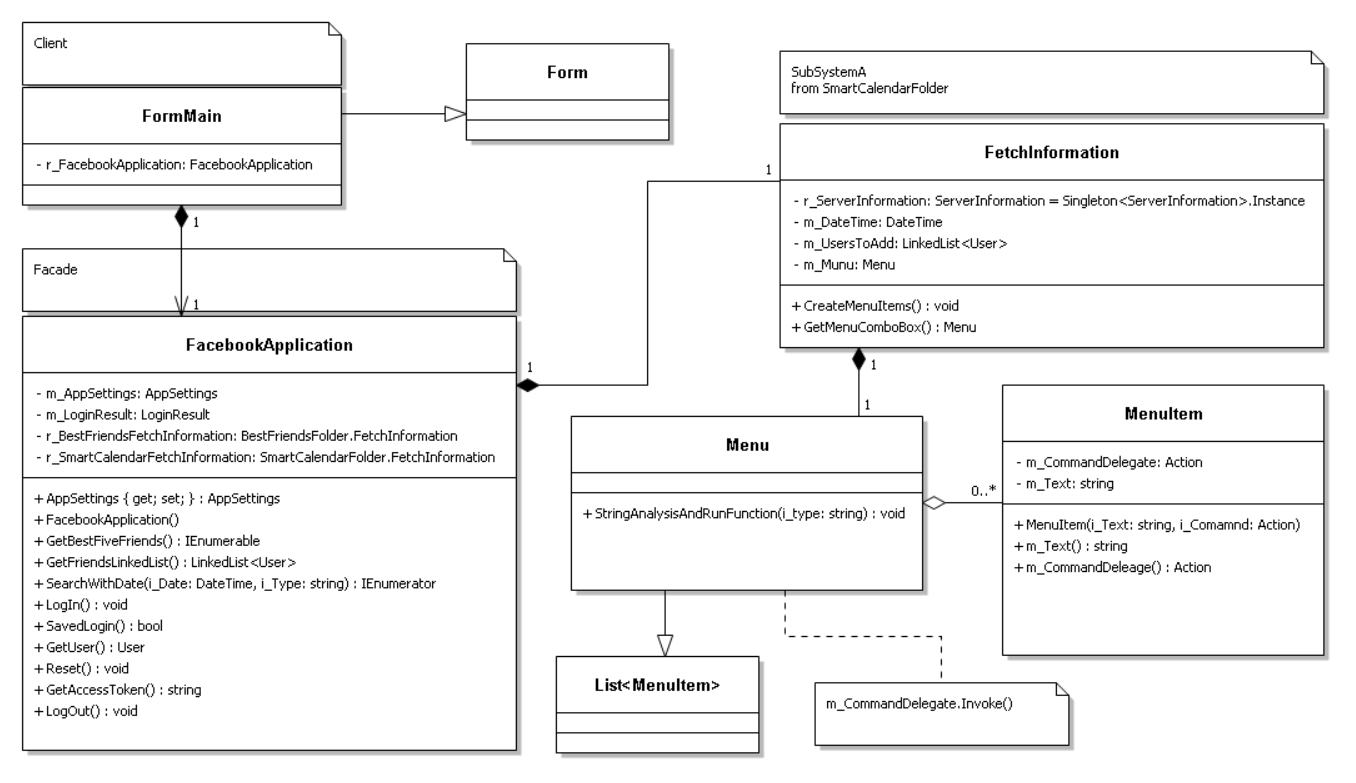
הוספנו תיקייה ששמה CommandFolder ובתוכה שני מחלקות :  
מחלקה ראשונה "MenuItem" אחראית על המבנה של האובייקטים שנרצה להוסיף לMenu,   
שיש בכל אחד מהם string - השם שאנחנו נרצה להציג ב comboBoxCalendarSelection למשתמש.

ו reference - של מתודה שתופעל ברגע שהמשתמש ייבחר בה.

מחלקה שנייה "Menu" מחלקה שיורשת מ List<MenuItem> אשר כל האובייקטים של המחלקה MenuItem נמצאים בה ברשימה/תפריט.  
ב Menu ישנה מתודה בשם StringAnalysisAndRunFunction(string i\_Type)   
אשר עוברת על כל הMenuItem שקיימים בתפריט ומפעילה את המתודה המתאימה לstring שהיא קיבלה כשאר היא מצאה את המתודה שהיא צריכה להפעיל היא בודקת אם המתודה לא null ובמידה והיא לא null היא מפעילה את המתודה המתאימה.  
בפעם הראשונה שמוצג החלון של SmartCalendar נקראת המתודה updateDataInComboText() אשר נמצאת ב SmartCalendar בתוך תיקיית Panel   
ועושה שתי פעולות :

יוצרת את הMenu על ידי שימוש בFacade שלנו.  
ומעדכנת את הDataSource של menuItemBindingSource אשר מציגה למשתמש את הפעולות שבהם הוא יכול להשתמש במערכת שלנו שמופיעות בתפריט מסודר שמי שאחראי להציג אותו הוא : comboBoxCalendarSelection שנמצא בתיקיית Panel ב SmartCalendar

* Sequence Diagram
* Class Diagram



### תבנית מס' 2 – [Iterator]

* סיבת הבחירה / שימוש בתבנית:

סיבות לבחירת Iterator :  
גישה למבנה נתונים של אובייקטים מבלי לחשוף את סוג מבנה הנתונים.

חשיפה חלקית של properties של אובייקט.

הגבלה של כמות האובייקטים ממבנה הנתונים.

ובמערכת שלנו בשני הפיצרים רצינו לתת גישה חלקית שנותנת מידע על המשתמש (לדוגמה שם פרטי ומשפחה , תמונה ובפיצר הראשון גם ניקוד), בנוסף רצינו אפשרות עתידית שאם נרצה להגדיל את כמות החברים הכי טובים נוכל להגדיל זאת על ידי שינוי במקום אחד בקוד, בלי לשנות את כל הקוד, רצינו שאם בעתיד נירצה להחליף מבנה נתונים שבו נשמר המידע אודות האובייקטים שלנו לא נצטרך לשנות בצד הקליינט בכלל, כלומר אנחנו רצינו תבנית עיצוב שתאפשר לנו לרוץ על מבנה נתונים בלי לדעת את מבנה הנתונים, ובנוסף לרוץ על כמות ספציפית שאנחנו נרצה להגדיר לו בצד הקליינט, שהקליינט לא יהיה גלוי לכל המידע שקיים באובקייט ששמור אלא מידע חלקי ספציפי, ואם בעתיד יבוצעו שינויים בצד הלוגי כמו שינוי מבנה נתונים או הדרך שהוא ממומש זה לא יפגע ויצטרך מאיתנו לשנות בצד הקליינט שום דבר.

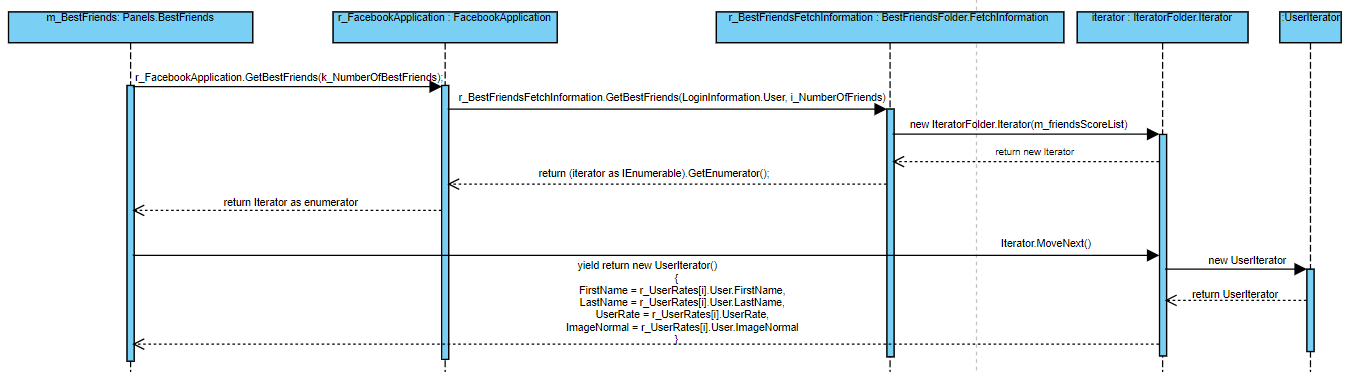
לכן תבנית עיצוב Iterator פותר לנו את הבעיות שבהם נתקלנו בכך שהקליינט אינו מכיר את מבנה הנתונים שהלוגיקה עובדת איתם , הקליינט יכול להגדיר כמה להגביל את כמות האוביקייטים כלומר כמה אובייקטים מתוך האוסף אנחנו רוצים , ובנוסף המידע שמגיע מוגבל כלומר אנחנו לא מקבלים את User עצמו שבתוכו יש מלא אלמנטים אלא מקבלים UserIterator שהוא מביא לנו בדיוק את המידע שאנחנו צריכים וככה מונע מאיתנו גישה לדברים שאיננו צריכים ומונע פוטנציאל לבאגים.

* אופן המימוש:

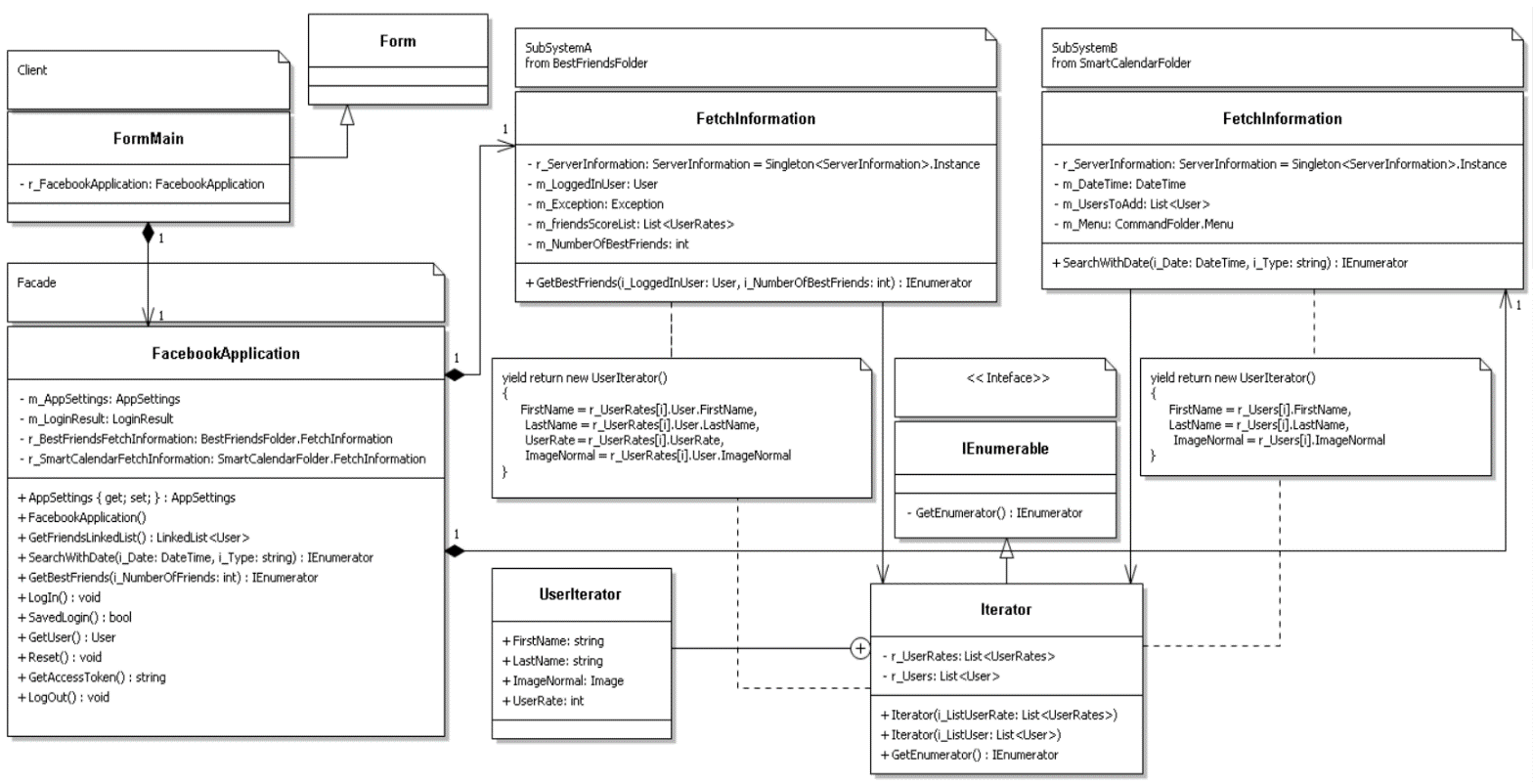
את המימוש שלנו ביצענו במחלקות -מחלקה בשם Iterator אשר נמצאת בתוך ,IteratorFolder FacebookApplication שהוא הFACADE שלנו, ובשני מחלקות הלוגיקה של שני הפיצרים BestFriendsFolder.FetchInformation

ו-SmartCalendarFolder.FetchInformation, דוגמת מימוש של הפיצר BestFriends אשר מחזיר את ה5 החברים הכי טובים, במחלקה FetchInformation של הפיצר הראשון (BestFriends) קיימת מטודה אשר עוברת בלולאה על הקולקציה הנ"ל (רשימת החברים הכי טובים) וממשת את ממשק הIEnumerable. במטודה GetEnumratorשנמצאת במחלקה Iterator אנו יוצרים אובייקט UserIterator אשר מממש את הממשק IEnumartor והוא האיטרטור המוחזר לFacebookApplication אשר הוא הFacade שמוחזר אל הClient.

* Sequence Diagram



* Class Diagram



### תבנית מס' 3 – [Strategy]

* סיבת הבחירה / שימוש בתבנית:

הסיבה שבחרנו להשתמש ב strategy היא מכיוון שהפיצר BestFriends בתוכו ממומשים אלגוריתמים שמבניהם דומה והשוני בהם הוא בתנאי הבדיקה ולכן אנחנו צריכים פתרון לבעיה הנדסית על מנת למנוע שיכפול קוד, תחזוקתיות, ופוטנציאל לבאגים, וכאן strategy בא לעזרתנו מכיוון שהוא מאפשר לנו לבנות אלגוריתם כללי שאפשר להבדיל אותו על ידי תנאי לדוגמה אצלנו בקוד אם אחנו רוצים לשלוף מידע אודות פוסטים ולייקים , פוסטים ותגובות, תמונות ולייקים, תמונות ותגובות, אלבומים ולייקים, אלבומים ותמונות, ובנוסף חישוב המידע אודות לייקים או תגובות, כלומר אלגוריתם שמובדל על ידי אסטרטגיה שונה על פי בחירת הקליינט.

לכן בחרנו בתבנית עיצוב Strategy וככה במקום שיהיה אלגוריתם לכל אחד מהדברים שתיארנו קודם, יש אלגוריתם אחד ויחיד אשר מה שאנו שולחים אליו זה את התנאים שמבדילים בין אלגוריתם לאלגוריתם כלומר אסטרטגיה בדיקה שונה, ככה מנענו שיכפול קוד, שיפרנו תחזוקתיות עתידים שאם בעתיד נרצה להוסיף גם אירועים ולייקים לא נצטרך לבנות אלגוריתם נוסף אלא להוסיף שורת קוד אחת שמשתמשת באלגוריתם הקיים ובנוסף אם נגלה באג בקוד לא נצטרך לשנות בכל האלגוריתמים אלא במקום אחד בלבד, ובנוסף מנענו פוטנצאל לבאגים בכך שאין צורך שוב פעם לבנות את הקוד מחדש.

* אופן המימוש:

את המימוש ביצענו בצורה הבאה:

יצרנו מחלקות בשמות UserRateBy ו-CalculatorUserRate.

המחלקה CalculatorUserRate היא מחלקה גנרית אשר מקבלת T אשר הוא יכול להיות סוג הטיפוס בהתאם למה שהקלייט מבקש.(לדוגמה Like).

המחלקה מחזיקה בתוכה member מסוג delegate Func אליו נשלח כל פעם האסטרטגיה הרצויה, כאשר התבנית היא <T, string, bool>.

בתוך המחלקה קיימת מטודה CalculateUserRate אשר מחשבת על פי האסטרטגיה שאותה קיבלנו ועל פי הסוג T שקיבלנו את הניקוד, אותה אסטרגטיה אנחנו מפעילים באמצעות פעולת Invoke, יחד עם שני פרמרטים T וstring אותם נשלח להשוואה.

באופן דומה יצרנו מחלקה UserRateBy אשר גם היא מחלקה גנרית אשר מקבלת T1, T2

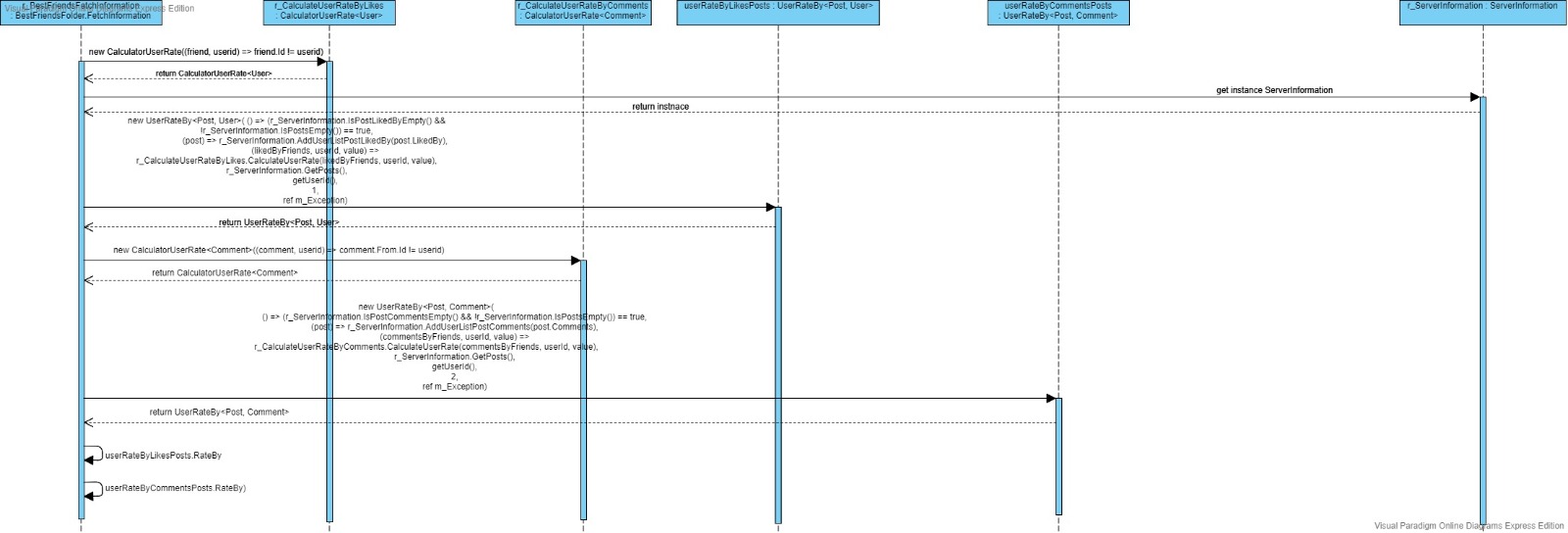
ובתוכה יש כמה אסטרטגיות שנצטרך לבדוק ולכן קיימים members שנקראים isEmpty, MethodOne וMethodTwo, אשר IsEmpty הוא delegate Func בעל תבנית <bool> וMethodOne הוא delegate Func בעל תבנית <T1, FacebookObjectCollection<T2>>

וMethodTwo הוא delegate Action בעל תבנית <FacebookObjectCollection<T2>

בתוך המחלקה קיימת מטודה RateBy אשר מריצה את האסטרטגיה הראשונה IsEmpty ובודקת אם היא מתקיימת אם כן היא רצה בלולאה על items מסוג T1 ועליהם את האסרטגיה בשם MethodOne אשר מחזירה איבר מסוג T2 שעליו מריצים את אסטרטגיה בשם MethodTwo.

מיקום שני המחלקות הנ"ל נמצא במחלקה BestFriends.FetchInformation מכיוון ששני המחלקות הנ"ל קשורות בקשר ישיר אחד לשני ואינם יכולות להתקיים מחוץ למחלקה ולכן גם שני המחלקות הנ"ל Private.

* Sequence Diagram



* Class Diagram

