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

* WellBeing הוא פיצ׳ר שנועד לתת למשתמש מידע אודות זמן השימוש שלו באפליקציה, ובכך לייצר מודעות לשימוש יתר במסכים בכלל ובאפליקציה שלנו בפרט.

כאשר יוצג למשתמש גרף אשר מתאר באופן וויזואלי את זמן השימוש שלו, ואת היסטוריית ההתחברות שלו בשבוע האחרון, יוכל המשתמש לנסות ולאתגר עצמו ולהשתמש באפליקציה באופן מידתי.

* MatchQuiz הינו פיצ'ר שמאפשר למשתמש לבחור חבר מרשימת החברים שלו ולקבל שאלון על החבר על סמך מספר מדדים. המערכת תחשב עבור המשתמש את אחוזי ההתאמה הזוגית עם החבר שבחר.

המערכת נותנת אינדקציה למשתמש האם הוא והחבר שבחר מתאימים, כך שיוכל לחשוב על הצעדים הבאים שלו במערכת היחסים הזאת.

הפיצ׳ר מתבסס על הנתונים שהמשתמשים מספקים לנו ושואל את השאלות שאנו חשבנו שהן המשמעותיות ביותר בקביעת התאמה, לדוגמה מקום מגורים, סוג לימודים וכו'.

### תבנית מס' 1 –Factory Methods - static Factory Class

* סיבת הבחירה / שימוש בתבנית:
* בחרנו לממש את התבנית הזו כיוון שזיהינו שימוש במשפחה פולימופרית של מחלקות אשר "חולקות" אב משותף והוא Form.

ע"י שימוש בתבנית זו העברנו את מלאכת היצירה של המשפחה לרכיב נפרד, ששייך לחלק הלוגי של המערכת, וכל client יכול לפנות אליו ביוזמתו ולקבל מופע של אחד מבני המשפחה\* של Form כנדרש.

(\*בני המשפחה הכוונה יורשים).

השימוש בתבנית הנ"ל תורמת לנו לתחזוקה של הקוד שכן אם בעתיד נרצה להרחיב את האפשרויות ולהוסיף עוד סוגי טפסים, נוכל לשנות (להוסיף) את הסוגים החדשים בFactory בלבד, שיידע ליצור אותם באופן ולהתנהג איתם באופן פולימורפי.

כמו כן, שימוש בתבנית זו נותנת לנו אפשרות לעשות שימוש חוזר במחלקת היצירה.

* אופן המימוש:

יצרנו מחלקת בנייה סטטית **FormFactory** static

המחלקה מחזיקה פונקיית create סטטית שמקבלת User וeFormType ,

היא יוצרת את הForm בהתאם לדרישות השונות ומחזירה אותו לclient.  
השתמשנו במחלקה זו לצורך יצירת הForms הבאים:

LoginForm, mainForm, QuizForm

* Sequence Diagram

Diagram

Description automatically generated

* Class Diagram

Diagram

Description automatically generated

1..\*xy

Createsxy

### תבנית מס' 2 – Proxy

* סיבת הבחירה / שימוש בתבנית:

הסיבה לבחירה בתבנית זו, היא ההבנה שבפיצ׳ר WellBeing אנו פונים לקובץ טקסט המשמש כ-DataBase. עד כה פנינו ישירות מהלוגיקה לקובץ ובעזרת הוספת המחלקה TimeUseProxy אני פונים רק בעת הצורך לDateBase שלנו.

בנוסף המחלקה מהווה חוצץ בין ממשק המשתמש לבין הרכיב הלוגי.

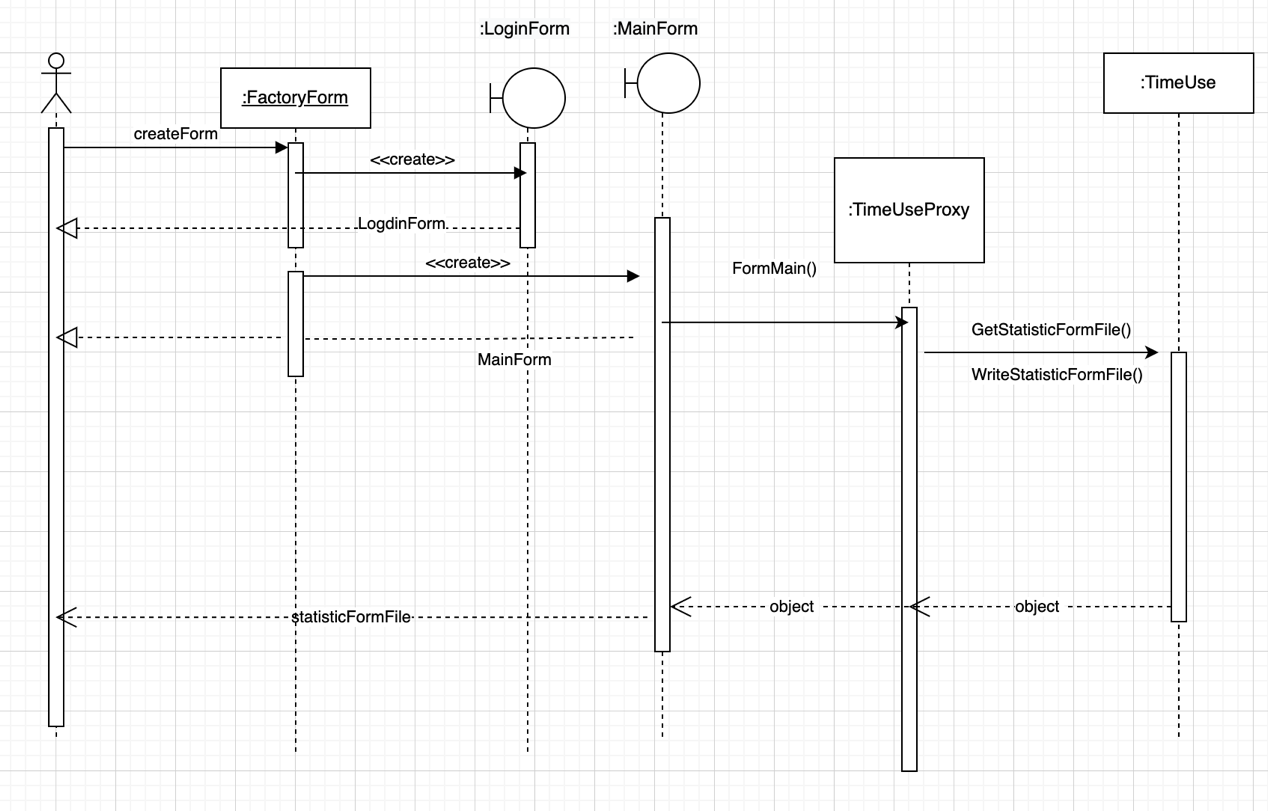
באופן זה אנחנו חושפים לclient רק את שתי המתודות הרלוונטיות של כתיבה וקריאה מה database ובכך אנו משפרים גם את רמת החשיפה של החלק הלוגי של הקוד ומסתירים מהclient רכיבים אחרים שקשורים לdatabase אך לא רלוונטיים אליו.

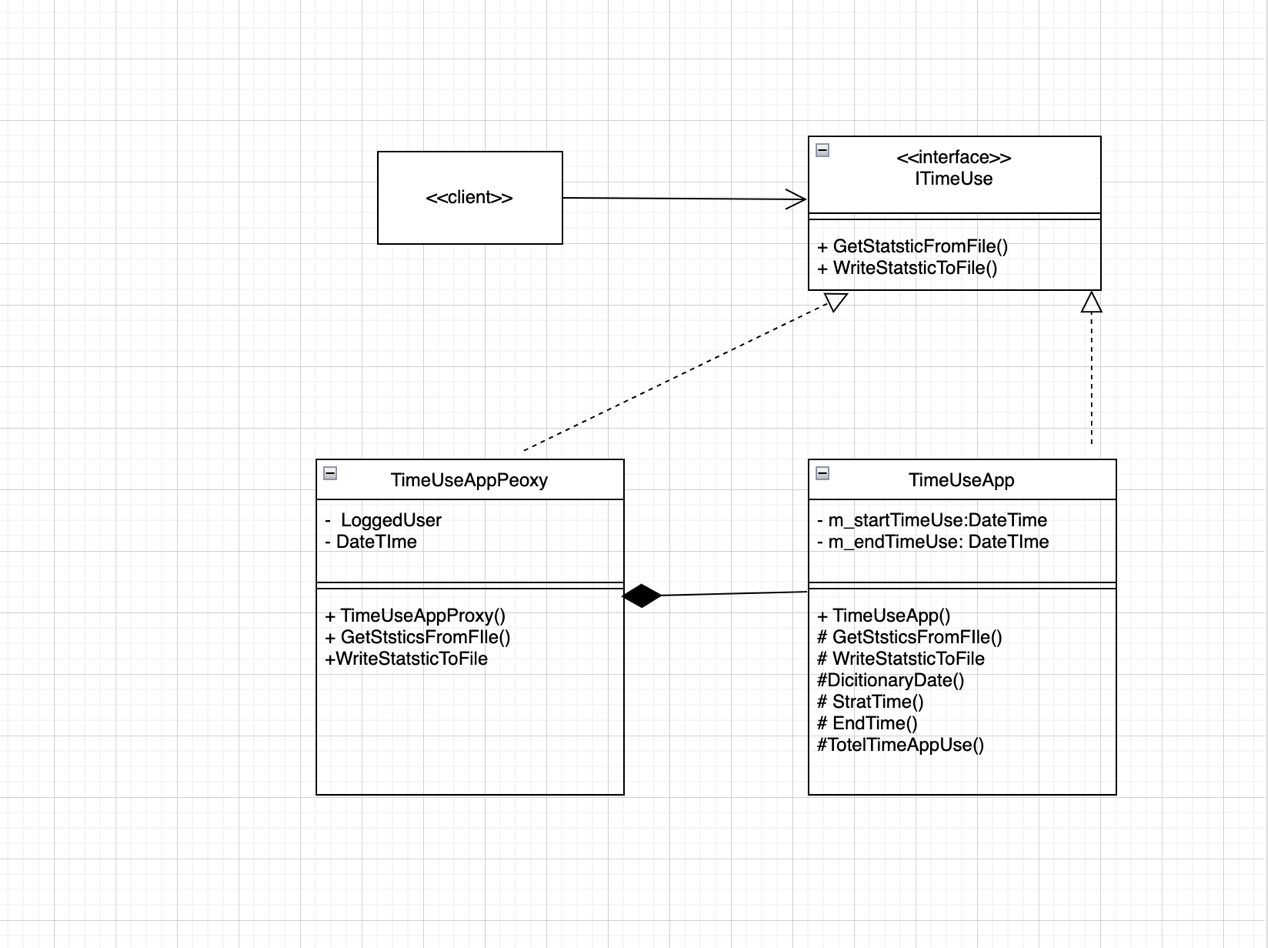
* אופן המימוש:
* Real subject – TimeUseApp
* Proxy – timeUseApp

יצרנו interface שנקרא ITimeUse שמחזיק את מתודות הקריאה והכתיבה. המחלקה המקורית של הTimeUse ממשת את הinterface, יצרנו מחלקה חדשה שנקראת TimeUseProxy כך שגם היא מממשת את הInterFace, למחלקה זו יש data member של המחלקה המקורית הTimeUse.

למחלקה mainForm יש data member של המחלקה החדשה TimeUseProxy

כאשר יש בקשה לכתיבה או קריאה ממסד הנתונים הוא קודם כול בודק האם האינסטנס קיים אם לא יוצר אותו ואם כן קורא למתודה המתאימה. בדרך זו ניצור אובייקטים "יקרים" רק על פי דרישה.

* Sequence Diagram
* Class Diagram



Proxy

Realsubject

1

1

### תבנית מס' 3 – Facade

* סיבת הבחירה / שימוש בתבנית:

הסיבה לבחירה נבעה מכך שהClient של FacebookService שהוא הFormLogin

פונה ישירות לFaceBookService שהוא רכיב מהחלק הלוגי של המערכת.

FacebookService הינו Legacy Component והשימוש בו דורש קונפיגורציות והתאמות שונות כמו בחירת Permissions, appID וכו'.

לכן השימוש בFacade Pattern מקל על השימוש ברכיב הלוגי FacebookService ומפשט אותו. בגישה זו הClient לא יפנה ישירות לFacebookService אלא יחזיק מופע של הFacade שלנו וישתמש בו. כעת אם ירצה הclient לעשות פעולה כמו התחברות לפייסבוק, הוא יוכל לעשות זאת "בלחיצת כפתור אחת" ע"י הפעלת המתודה Login של הfacade, שהיא בעצמה מאחורי הקלעים עושה את הקונפיגורציות הדרושות. השימוש בFacade מפשט לclient את השימוש ברכיב הלוגי, כמו כן תורם לתחזוקתיות ובטיחות שימוש, כיוון שהclient משתמש ברכיב ביניים אשר מפשט ומשטח את המידע.

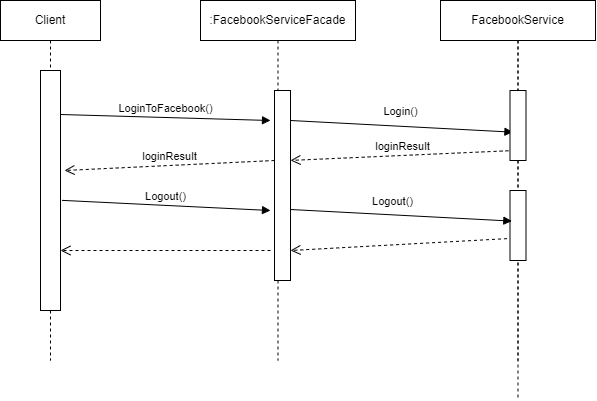
* אופן המימוש:

יצרנו מחלקה שנקראת FacebookFacade שמחזיקה שדות רלוונטיים להפעלת מתודות של FacebookService כמו Login שדורשת Permissions וappID.

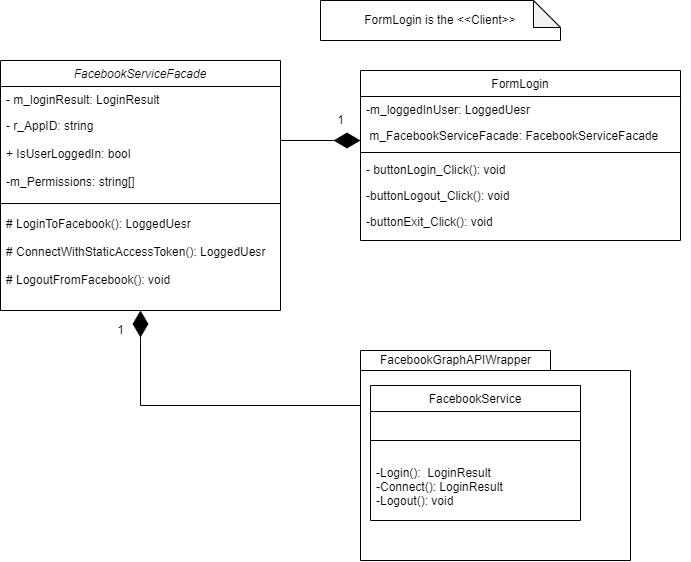
את שדות אלו נאתחל בFacade לפי צרכי המערכת.

הclient (במקרה שלנו LoginForm) יחזיק מופע של המחלקה Façade ויפנה אליה בכל פעולה רלוונטית, כלומר במקרה של התחברות לפייסבוק, נפעיל את LoginToFacebook של הfacade (והfacade בעצמו יפעיל את המתודה של FacebookService תוך כדי שימוש בשדות הרלוונטיים, ויעשה את ההתאמות והקונפיגורציות הדרושות).

* Sequence Diagram



* Class Diagram1



1

Connects to facebook

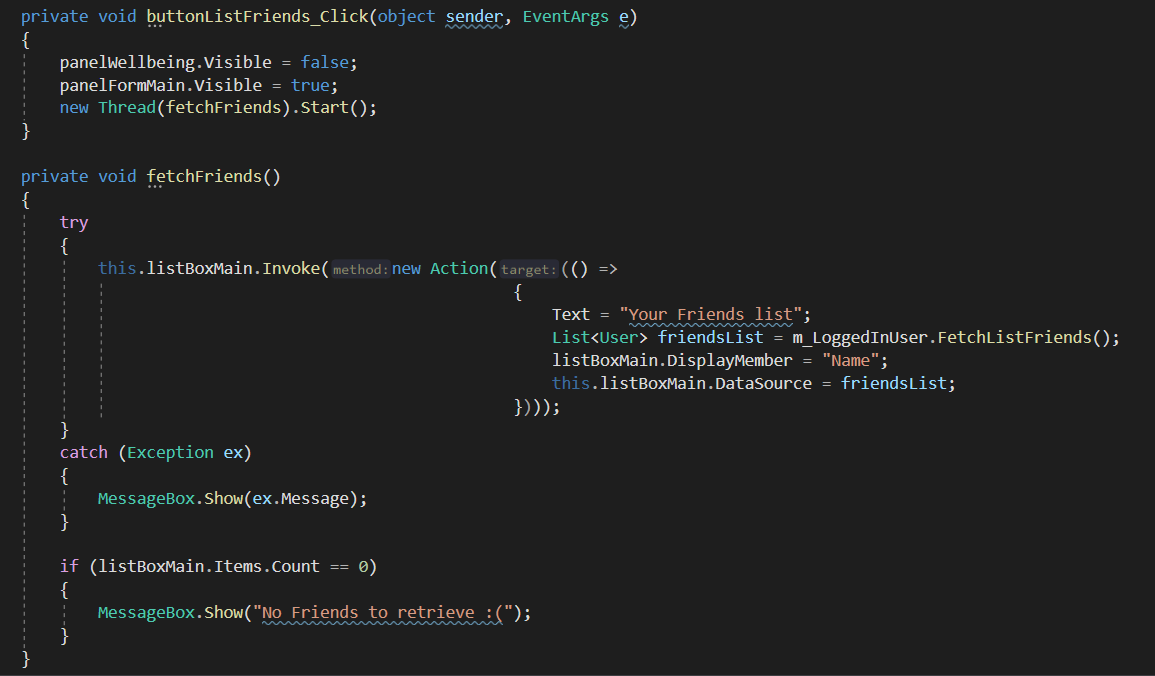
**מימוש עבודה אסינכרונית:**

בגרסה הראשונה של האפליקציה שלנו, כאשר המשתמש השתמש באחד ממגוון הפיצ'רים שהאפליקציה מאפשרת (קבלת רשימת החברים שלו , רשימת דפים שהוא אוהב,רשימת האלבומים שלו , רשימת קבוצות שהוא חבר בהן, רשימת פוסטים שהוא פרסם ,רשימת אירועים שלו), הרכיב הלוגי של המערכת פנה לשרתים של פייסבוק על מנת לקבל את הנתונים הדרושים. כידוע במערכת עם thread אחד, כאשר הthread הזה "עסוק" במשימה של תקשורת מול פייסבוק וקבלת הנתונים, שאר המערכת נהיית סטטית ונכנסת למצב של "קיפאון" עד שתסתיים המשימה - קבלת הנתונים מפייסבוק.

כתוצאה מכך, עלולה להיפגע חווית המשתמש, שכן היינו רוצים מצב בו כאשר המערכת "עסוקה" בדבר אחד, היא עדיין תהיה רספונסיבית עם המשתמש, אפילו ברמה הפשוטה ביותר של האפשרות להזיז החלון של האפליקציה.

על מנת לפתור את בעיה זו, הוספנו שימוש בtherads נוספים במספר מתודות שעלולות להוות "נקודות תורפה" בקוד, כלומר מתודות שעלולות לעלות הרבה זמן (באופן יחסי) ולפגוע בחווית המשתמש. למעשה, את כל המתודות שמביאות רשימות כלשהן (קבלת רשימת החברים שלו , רשימת דפים שהוא אוהב,רשימת האלבומים שלו , רשימת קבוצות שהוא חבר בהן, רשימת פוסטים שהוא פרסם ,רשימת אירועים שלו), כתבנו כך שיופעלו בthread נפרד, כך שכשאר המשתמש ילחץ על אחד הכפתורים האלה, לדוגמה Friend list , הפעולה תתבצע בthread נפרד ובכך שאר האפליקציה תהיה רספונסיבית עם המשתמש.

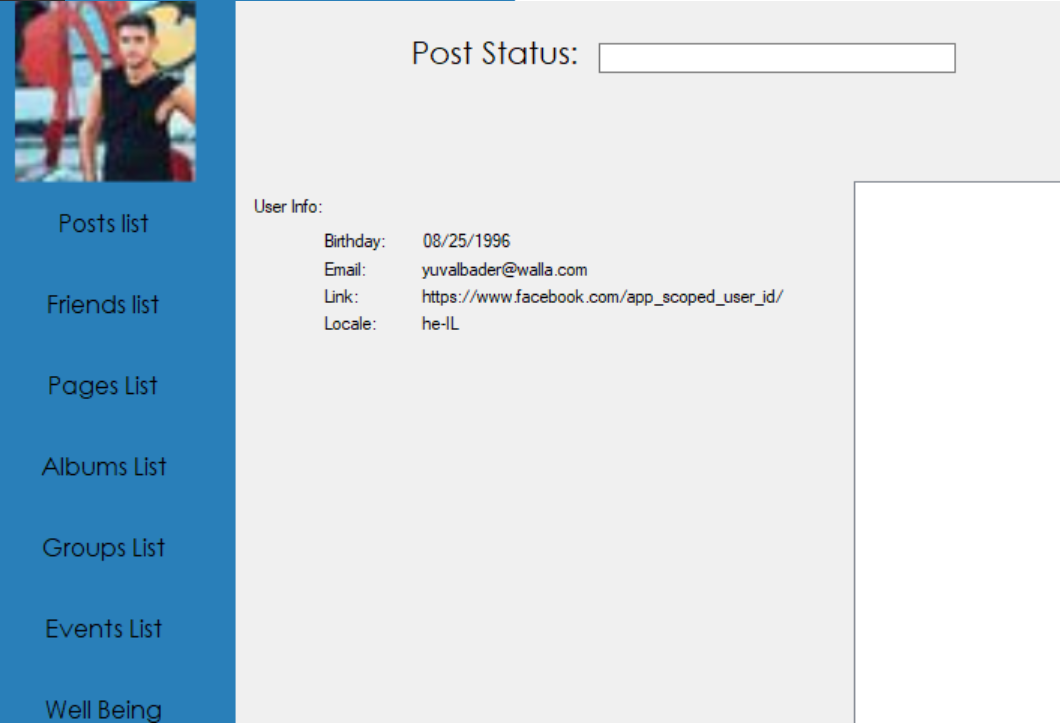
דוגמה בקוד: כפי שניתן לראות לחיצה על כפתור ListFriends יוצרת thread חדש ומפעילה את fetchFriends() .



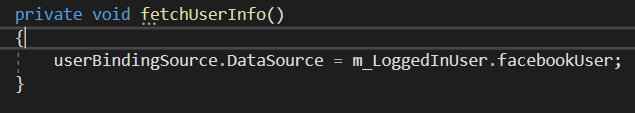
**מימוש** **:Data Binding**

את השימוש בData binding עשינו בשני מקומות:

**1.**



השתמשנו ב- Data Source מסוג User בMainForm כדי להציג פאנל שמרכז פרטים בסיסיים אודות המשתמש. בחרנו מתוך User את השדות הרלוונטיים עבורונו ואותם בחרנו להציג.

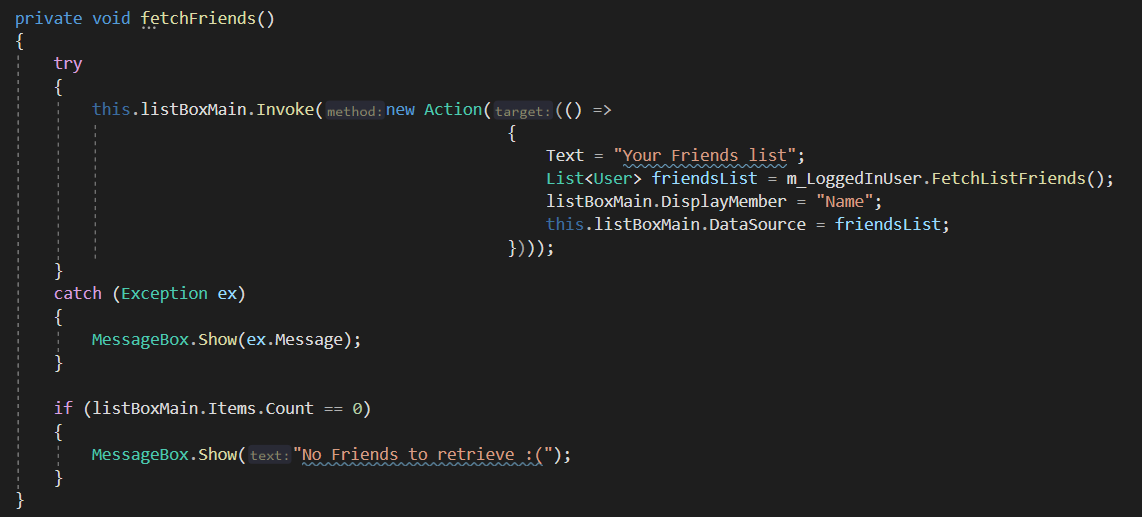
את הקישור בין הloggedUser ל dataSource עשינו במתודה 

את מתודה זו אנו מפעילים כאשר אנחנו יוצרים את הטופס FormMain.

האובייקט ש"קשרנו" (bind) הוא מסוג User והפקד שאליו קשרנו הוא פאנל של פקדים.

2. בכל המתודות שמביאות רשימות כלשהן (קבלת רשימת החברים שלו , רשימת דפים שהוא אוהב,רשימת האלבומים שלו , רשימת קבוצות שהוא חבר בהן, רשימת פוסטים שהוא פרסם ,רשימת אירועים שלו) עשינו שימוש בdataBinding באופן הבא:

האובייקט שקשרנו הוא רשימה של נתונים שקיבלנו מהרכיב הלוגי (לדוגמה רשימת חברים , רשימת דפים וכו'), והפקד שאליו קשרנו במקרה הזה הוא הLIstBoxMain.

כלומר במקום להכניס לListBoxMain את הנתונים אחד אחד בforeach, עשינו קישור בין מקור המידע (הרשימה) לפקד.