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

* **Smart post:** פיצ'ר זה מאפשר למשתמש לפרסם הודעה לחתך מסוים של החברים שלו. החתך נקבע לפי מספר פילטרים שהמערכת מאפשרת כגון: בחירה לפי מין, בחירה לפי מצב משפחתי (רווק, נשוי וגרוש), בחירה לפי גיל (מעל/מתחת לגיל 18), בחירה לפי עיר מגורים. על המשתמש לבחור באיזה מבין הפילטרים הוא רוצה להשתמש (אחד או יותר). לאחר שהמערכת תבצע את הפילטור היא תציג למשתמש את החברים שנבחרו. ברגע שהמשתמש יבחר לפרסם את הפוסט, המערכת תתייג את אותם חברים בפוסט.   
  את הפיצ'ר ניתן למצוא ב DesktopFacebook module תחת התיקייה CustomFeatures תחת התיקייה SmartFilter
* **Friendship match scale:** פיצ'ר זה מאפשר למשתמש לבחור את אחד מחבריו בפייסבוק המופיעים ברשימה ולחשב את מידת התאמתם ממספר בחינות: פרטים אישיים(עיר מגורים,סטטוס זוגי, תפקיד מקצועי אשר בו הם עוסקים בעבודתם הנוכחית), סגנון מוזיקה אהוב ומקומות בילוי מועדפים. המערכת מחשבת את אחוז ההתאמה של חברו עם המשתמש המחובר לאפליקציה ומציגה אותו.  
  את הפיצ'ר ניתן למצוא ב DesktopFacebook module תחת התיקייה CustomFeatures תחת התיקייה FriendshipMatchScale.

### תבנית מס' 1 –Static factory class

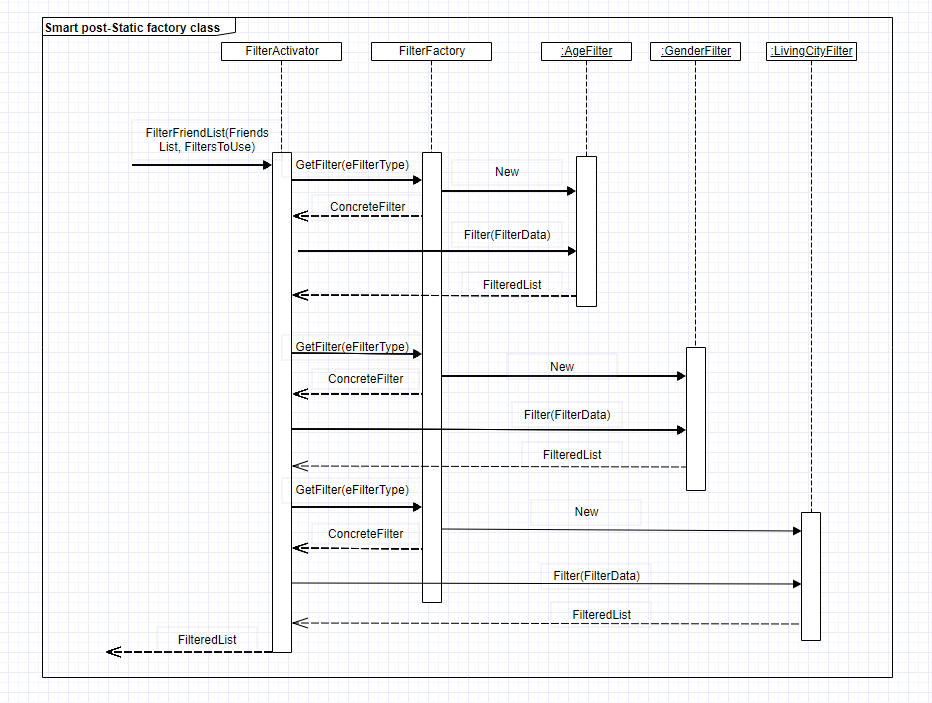
* סיבת הבחירה / שימוש בתבנית:

הסיבה שבחרנו להשתמש בתבנית זו היא שבמערכת ישנם כמה סוגי Filters שמאפשרים לנו לסנן חברים מרשימת החברים של המשתמש לפי פרמטרים מסוימים. כל filter ממש את הממשק IFriendsFilter. לוגיקת היצירה של כל filter נקבעת ע"י enum בשם eFilterType. השימוש כאן ב pattern של Static factory class בעצם מאפשר לנו להעביר את לוגיקת היצירה למחלקה נפרדת שאחראית ליצור את ה filter המתאים ולספק אותו למשתמש. בעצם הפרדנו את ההחלטה של איזה filter ליצור ואת היצירה עצמה מהקוד של ה client.

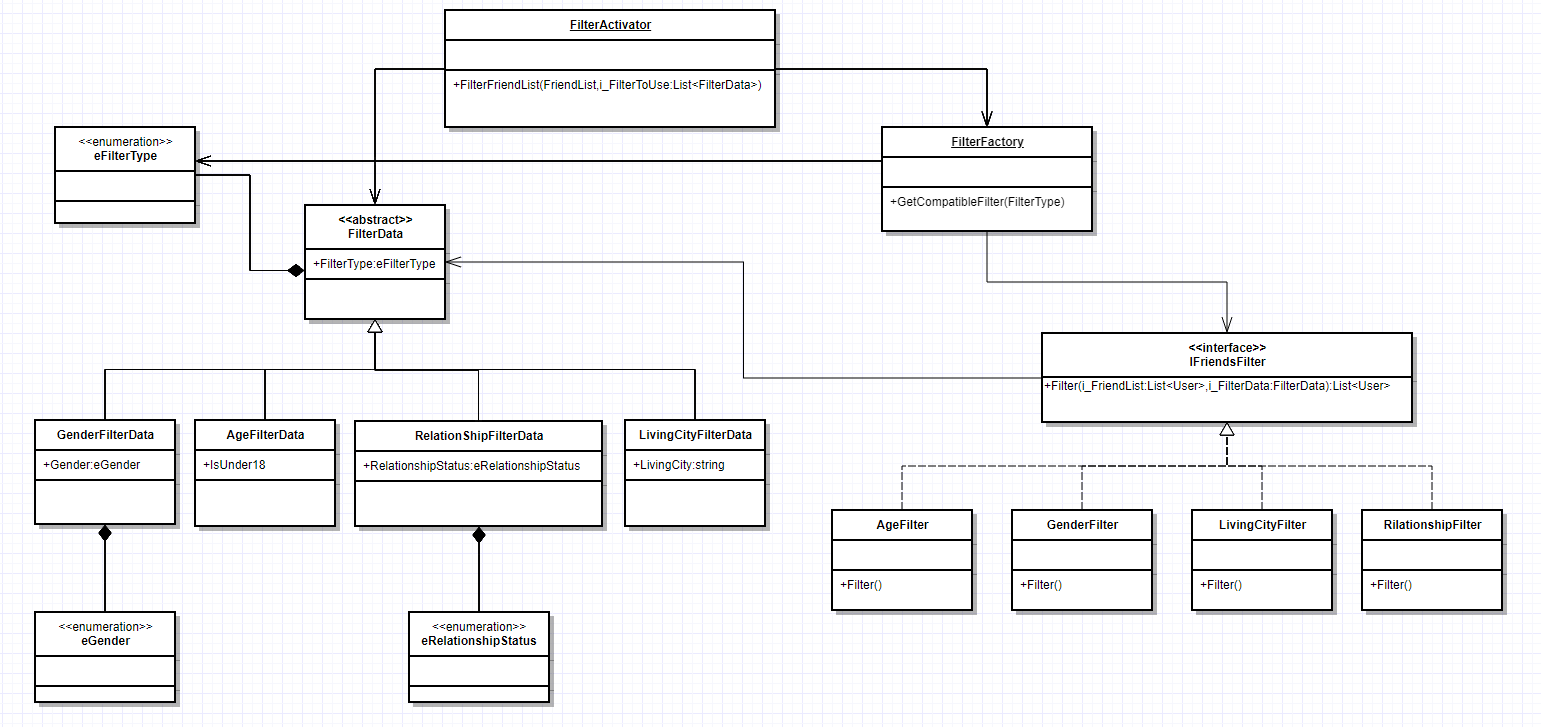
* אופן המימוש:

בחרנו לממש את pattern זה תחת ה smart post feature. יצרנו מחלקה סטטית בשם FilterFactory. המחלקה מכילה פעולה בשם GetCompatibleFilter שמקבלת פרמטר מסוג eFilterType ועל פיו מבצעת את ההחלטה איזה filter ליצור. המחלקה תחזיר אובייקט מסוג IFriendsFilter שתהווה מימוש קונקרטי של אחד מה- filters שבמערכת.

* Sequence Diagram



* Class Diagram



* המחלקה FilterFactory מייצגת את ה static factory class
* המחלקה IFriendsFilter מייצגת את ה abstract product
* המחלקות AgeFilter, GenderFilter, LivingCityFilter, RelationshipFilter מייצגות את ה Concrete product

### תבנית מס' 2 – Adapter

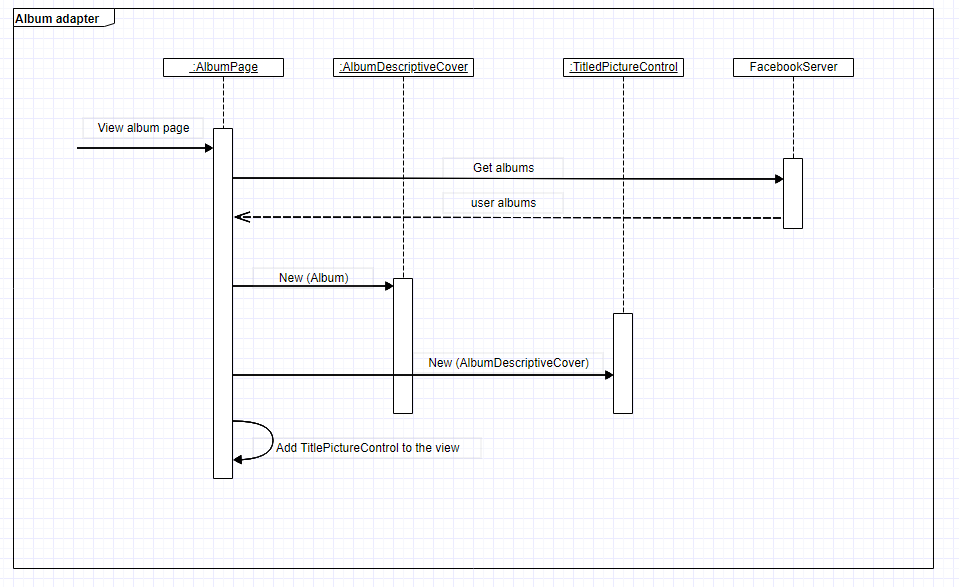
* סיבת הבחירה / שימוש בתבנית:

במערכת שלנו יש רכיב UI שאחראי על הצגת תמונה עם תיאור שלה. הרכיב נקרא TitledPictureBox. הרכיב מצפה לקבל אובייקט המממש את הממשק IDescriptivePicture. במערכת אנו רוצים להשתמש ברכיב זה על מנת להציג את האלבומים של המשתמש. במקרה שלנו Album הינו רכיב legacy ולכן היה עלינו להשתמש ב adapter pattern לצורך ההתאמה של Album ל IDescriptivePicture שלה מצפה הרכיב UI.

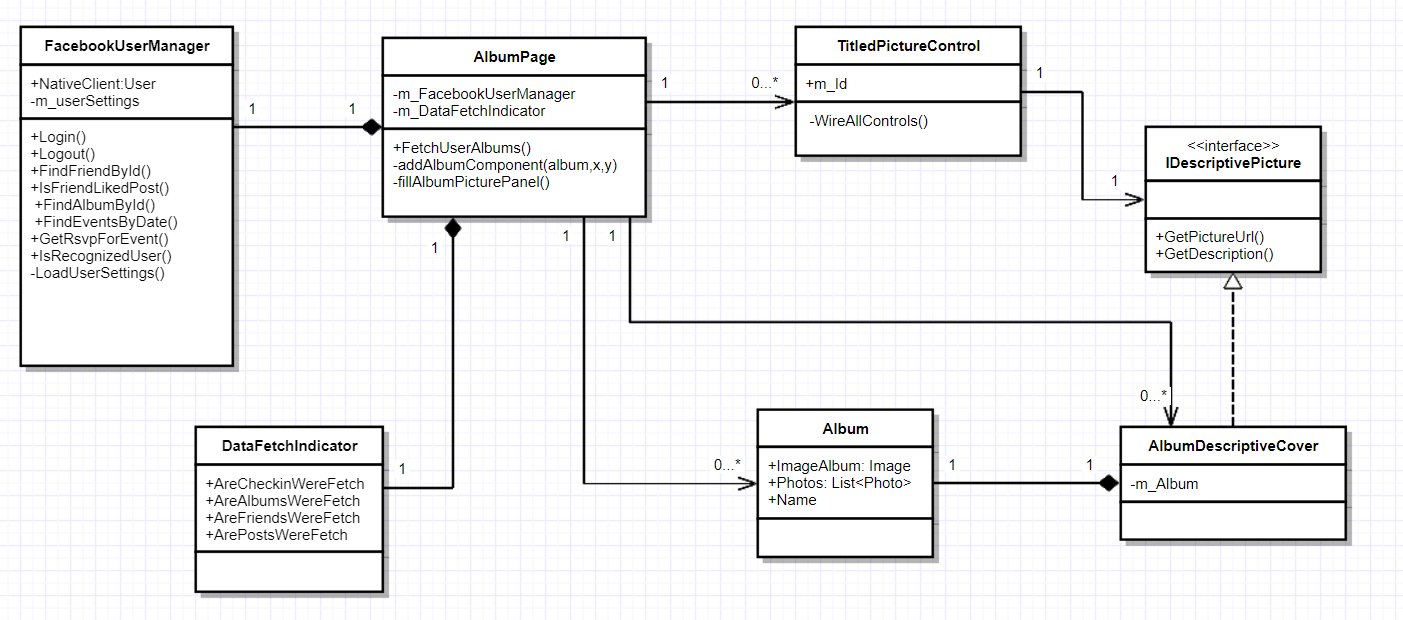
* אופן המימוש:

לצורך המימוש היה עלינו ליצור מחלקה חדשה בשם AlbumDescriptiveCover שמממשת את ממשק IDescriptivePicture. המחלקה מקבלת ב ctor שלה אוביקט מסוג Album על מנת שתוכל להחזיר את פרטי האלבום (תמונה ושם). בעת יצירת דף האלבומים במערכת שלנו, המערכת תעבור על אלבומי המשתמש ותיצור AlbumDescriptiveCover עבור כל אלבום. לבסוף המערכת תעביר את אובייקט ה AlbumDescriptiveCover ל TitledPictureBox על מנת להציג אותו ב UI.

* Sequence Diagram



* Class Diagram



* המחלקה TitledPictureControl מייצגת את ה client
* המחלקה IdescriptivePicture מייצגת את ה target
* המחלקה AlbumDescriptiveCover מייצגת את ה Adapter
* המחלקה Album מייצגת את ה Adaptee

### תבנית מס' 3 – Proxy

* סיבת הבחירה / שימוש בתבנית:

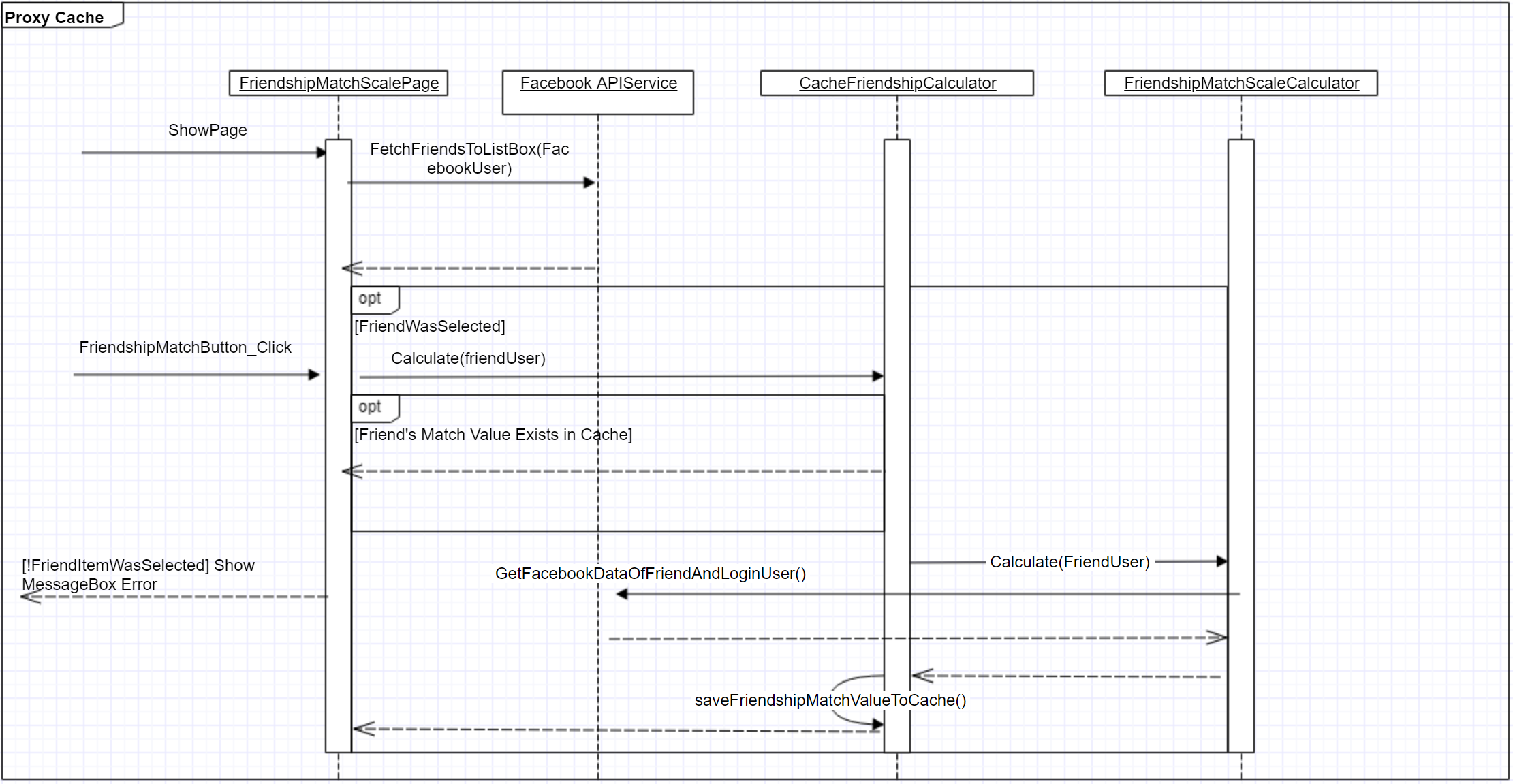
במערכת שלנו ישנו רכיב כFriendshipMatchScaleCalculator אשר אחראי לבצע חישוב של אחוז התאימות על פי פרמטרים קבועים מראש(פרטים אישיים, מקומות בילוי, מוסיקה) בין המשתמש שביצע LOGIN לבין החבר שאותו הוא בחר בטופס. הבעיה איתה היינו צריכים להתמודד היא העובדה שעבור כל לחיצה בחבר של אותו המשתמש שכבר נבחר בפעמים הקודמות אנו מבצעים חישוב מחדש של כל הפרמטרים כאשר החישוב הנ"ל התבצע בעבר אבל לא נשמר במערכת. נשים לב שעבורינו FriendshipMatchScaleCalculator הינו רכיב Legacy ואין באפשרותינו לשנותו. בנוסף, היינו צריכים לשמר את אופן ההפעלה של FriendshipMatchScaleCalculator בדף FriendshipMatchScalePage אך עם זאת להוסיף יכולת Caching. לכן בחרנו בproxy design pattern אשר כל תפקידו הוא לשמר את אופן העבודה מול הclass המקורי תוך הוספת יכולות נוספות למערכת. (במקרה שלנו, הוספת יכולת caching)

* אופן המימוש:

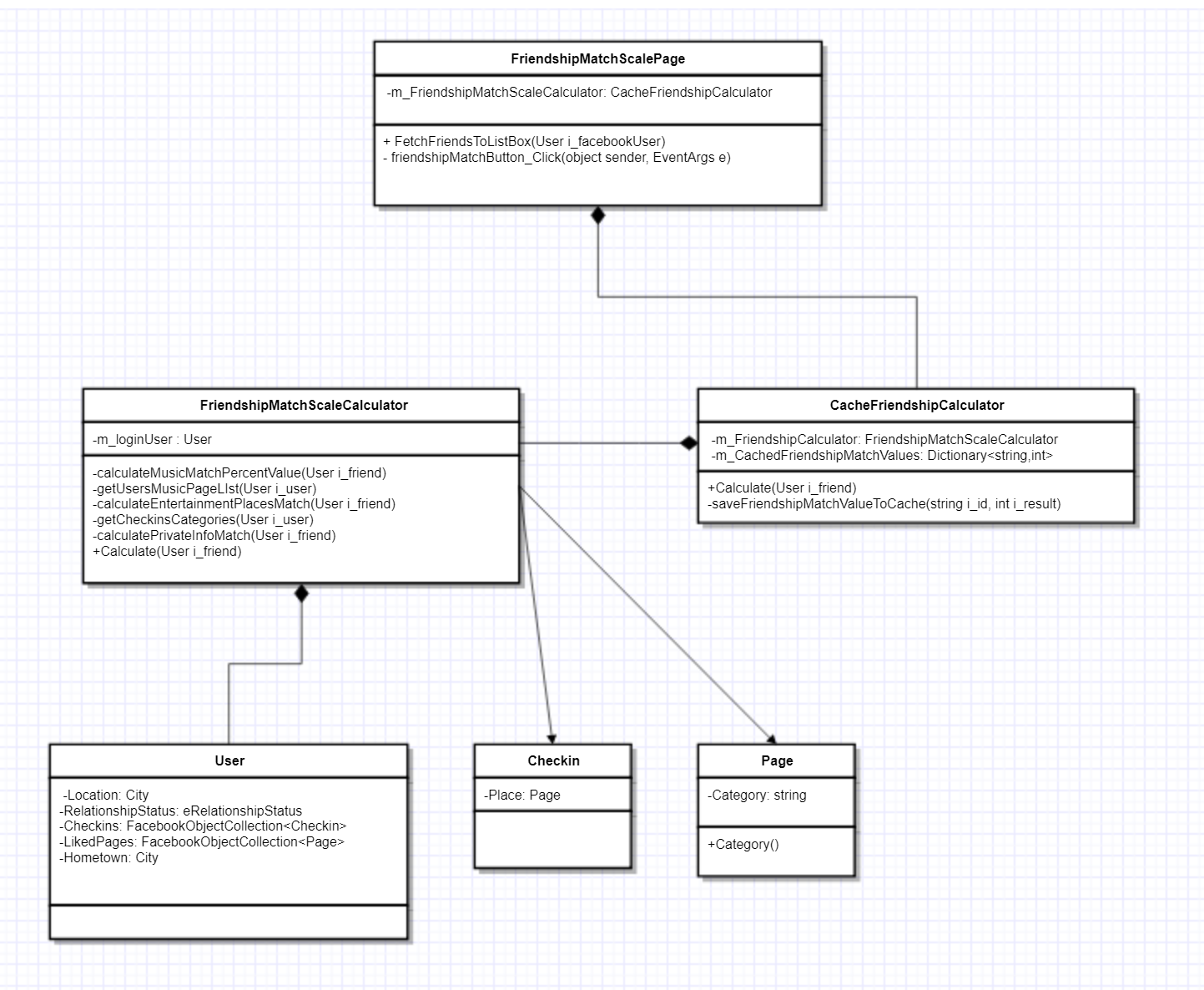
בחרנו להתשמש בdesign pattern הנ"ל באופן הבא:

יצרנו רכיב **Cache**FriendshipCalculator אשר 'יעטוף' את הרכיב FriendshipMatchScaleCalculator (באמצעות קומפוזיציה) ובנוסף יכיל dictionary של friendID,matchValue אשר בו יישמרו כל ערכי החישוב שבוצעו. לCLASS הנ"ל תיהיה את הפונקציה calculate כמו שיש לclass המקורי (FriendshipMatchScaleCalculator) וכך הדף ימשיך לעבוד באותו אופן מול הרכיב רק שהפעם בפונקציה calculate של הProxy אנו נבדוק האם החבר שנבחר קיים בdictionary. אם כן, נחזיר את הערך שנמצא בdictionary, אחרת, נקרא לפונקציה calculate של הרכיב המקורי ונשמור את הערך המוחזר בdictionary.

* Sequence Diagram



* Class Diagram



* המחלקה FriendshipMatchScalePage מייצג את הclient
* המחלקה FriendshipMatchScaleCalculator מייצגת את הLegacy component שאותו נרצה "לשדרג" ולהוסיף את יכולת הCaching.
* המחלקה CacheFriendshipCalculator מייצגת את הProxy ובו יש את היכולת הנוספת – Caching.

### עבודה אסינכרונית:

בחרנו להשתמש בעבודה עם multithreading בעת טעינת ה posts של המשתמש. את השימוש ב multithreading ניתן למצוא במחלקה UserProfileForm תחת הפונקציה fetchUserPostAsync. מטרת הפונקציה היא לטעון את ה posts האחרונים של המשתמש ולהציג אותם. הצגת ה post כוללת יצירת userControl חדש עבור כל post שמכיל בתוכו מכלול של פקדים לצורך ביצוע פעולות על ה post. השימוש בעבודה אסינכרונית ממומש באופן הבא:  
הפעולה fetchUserPostAsync מייצרת thread חדש האחראי לקבל את ה posts מפייסבוק , לייצר עבור כל אחד את ה userControl שלו ולהוסיף אותו ל panel המתאים ב UI. על מנת להוסיף את ה userControl של כל post ל panel הראשי של ה posts היה עלינו להשתמש בפונקציה invoke של ה panel על מנת שנוכל לשנות את פקד ה UI מ thread שהוא לא ה thread שיצר אותו.  
בחרנו להשתמש בעבודה אסינכרונית במקרה זה על מנת לאפשר למשתמש להמשיך להשתמש באפליקציה מבלי לחכות עד שכל ה posts שלו ייטענו. בעזרת multithreading שיפרנו את חווית השימוש של המשתמש.

### שימוש בDataBindings:

בחרנו להשתמש בעבודה עם DataBinding עבור פרטיו האישיים של המשתמש שביצע LOGIN (Name, Gender, Birth Date, HomeTown, Current City). השימוש ב data Binding בא לידי ביטוי בשלבים הבאים:

1. הוספת Data source לפרויקט מסוג User.
2. הוספת Labels עבור הצגת פרטיו האישיים של המשתמש + קישורם לDataBinding שיצרנו בסעיף1
3. הוספת אובייקט ל binding source, הdata member שנוצר באופן אוטומטי (כתוצאה מקישור הdatabinding לLabels בסעיף 2.) – התבצע בCTOR של UserProfileForm.

בחרנו להשתמש כאן בDataBinding מכיוון שכלל הפרטים האישיים של המשתמש נלקחים כולם מאובייקט יחיד – User , ולכן הדבר מאפשר בשורת קוד אחת לבצע קישור אל מול כל פרטיו האישיים של המשתמש. בנוסף, במידה והם משתנים במהלך ריצת המערכת, השינוי יתבצע באופן אוטומטי כתוצאה מהDataBinding ואילו ללא השימוש בו, המידע שיוצג יהיה המידע שנלקח בתחילת ריצת האפליקציה אשר מהווה מידע לא עדכני.