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

* **Covid-19 Post :**

FormCorona– בעקבות וירוס הקורונה, הוספנו אפשרות של העלאת פוסט בו התוכן קבוע עם האשטאג #CoronaPost, פיצ'ר זה מאפשר למשתמש אשר חלה בקורונה לעדכן את כל החברים שהוא היה בקרבתם בתקופת השבועיים האחרונים מהיום בו הוא השתמש בפיצ'ר.

הפיצ'ר בודק את כל האנשים שהיו בקרבת היוזר בשבועיים האחרונים על פי תיוגים בפוסטים, תמונות וצ'ק אין שהתבצעו בתקופת השבועיים הללו.

**היכן למצוא** :בחלון הראשי שנפתח (FormFacebook) לאחר שמבצעים התחברות, ניתן לראות בשורה השלישית אייקון שרשום עליו 'Covid-19 Post ', לאחר לחיצה על אייקון זה, יפתח חלון המכיל את התוכן של הפוסט ואפשרות לפרסום.

- פיצ'ר זה יושב תחת המחלקה FormCorona בפרויקט ועושה שימוש במחלקות של FacebookWrraper כגון Post, Checkin, Friends וכו'.

* **Matching By Zodiac:**

FormMatch – פיצ'ר זה מציג למשתמש את כל החברים שלו שמתאימים לו על פי המזל שלו.

הפיצ'ר מסנן את החברים המתאימים לפי מזל ולפי העדפות המשתמש: גברים, נשים או שניהם. פיצ'ר זה ממומש במחלקה FormMatch ועושה שימוש במחלקות : PanelToUser, PanelWomen, PanelMen, PanelBsexual כאשר הפנלים מממשים תצוגת מידע עבור יוזר בודד בחברים של המשתמש, השוני בין הפנלים הוא בעיצוב שלהם כל אחד מעוצב בצבעים שונים כך שכל בחירה של המשתמש תקבל תצוגה ייחודית.

תוספים שהפיצ'ר משתמש בהם:

* הפיצ'ר משתמש במחלקה ProxyFriends על מנת לייעל את העבודה מול השרת של פייסבוק ולהשתמש במאגר זמני של רשימת חברים, שנוצר במערכת שלנו.
* הפיצ'ר עושה שימוש במחלקה FacadeMatch שאחראי על הבאת רשימת החברים שמתאימים ע"פ מזל ומין למשתמש מרשימת החברים שלו.

**היכן למצוא** :בחלון הראשי שנפתח (FormFacebook) לאחר שמבצעים התחברות, ניתן לראות בשורה השלישית אייקון שרשום עליו 'Find Match ', לאחר לחיצה על אייקון זה, יפתח חלון השואל את המשתמש מה ההעדפה המינית שלו.

- פיצ'ר זה יושב תחת המחלקה FormMatch בפרויקט ועושה שימוש במחלקה של FacebookWrraper :User.Friends

### תבנית מס' 1 – [Strategy]

**סיבת הבחירה / שימוש בתבנית:**

באפליקציה קיים פיצ'ר המאפשר ללקוח לראות את רשימת הפוסטים שלו, הטופס נקרא- FormPosts . אנו רוצים לאפשר למשתמש סינון של הפוסטים על פי קטגוריות משתנות, על כן צירפנו אפשרויות לחיפוש פוסטים בצורה שונה על פי בחירת המשתמש (פוסטים בשלוש שנים האחרונות , פוסטים עם מלל ארוך , כמות לייקים על פוסטים..),

אלגוריתם הסינון לקטגוריות השונות דומה אחד לשני, השוני הוא בסוג השאילתה המבצעת את הסינון של הרשימה המיוצרת, כלומר באסטרטגיה הנבחרה לסינון.

על כן יישמנו את pattern strategy, המאפשר בחירה של אסטרטגייה בזמן ריצה. כלומר כל פעם שהלקוח ירצה לסנן , הוא יבחר באסטרטגיה הרצוייה לו, ולפי בחירותו יתבצע הסינון.

**השימוש בפטרן זה** מאפשר גישה נוחה לאלגוריתם בכך שהוא מיישם ממשק משותף (משפחת האלגוריתמים) הנקרא IPostFilter , לכל האסטרטגיות כך שכל אסטרטגייה מממשת את הממשק במחלקה נפרדת .

ולכן אם בעתיד, נרצה להרחיב את הפיצ'ר ולאפשר לבנות עוד אפשרויות לחיפוש וסינון פוסטים בצורה שונה שלא מצויינת כבר, אסטרטגיות סינון ומיון שונות, נוכל לעשות זאת ללא שכפול קוד או תחזוקה מיותרת של קוד קיים, אלא ע"י הוספת מחלקה בודדת של פילטר המממשת את הממשק IPostFilter , מבלי לשנות את המחלקות הקיימות בחלק הלוגי.

**אופן המימוש:**

בפרוייקט הלוגי –

בנינו ממשק הנקרא " IPostFilter" המכיל את המתודה:

bool FilterPost(Post i\_PostToCheck)

הוספנו מחלקה חדשה הנקראת MyFilteredPosts שמממשת את הממשק IEnumerable<Post> ,

מחלקה זו מכילה אובייקט מסוג IPostFilter, אותו יוצר טופס Posts ע"פ בחירת המשתמש, המחלקה FormPosts מעבירה את המופע שהיא יצרה למחלקה 'הפוסטים המסוננים שלי ', על מנת לעדכן את השדה – m\_PostFilter שמצביע על הסינון שבחר המשתמש.

מחלקה זו מכילה מתודה שמבצעת את המיון - FilterPostList() , המתודה מפעילה את אובייקט הסינון על כל פוסט ברשימת הפוסטים של המשתמש , במידה ואובייקט הסינון החזיר כי הפוסט כמתאים (TRUE), המתודה מוסיפה את הפוסט לרשימת הפוסטים שנשמרים במחלקה – MyFilteredPosts .

מימשנו בחלק הלוגי 3 מחלקות שמממשת את הממשק IPostFiltrer :

כלומר יצרנו משפחה של אלגוריתמי מיון שכל אחד במחלקה משלו -

* PostFilterByDate
* PostFilterByLength
* PostFilterByLikesNumber

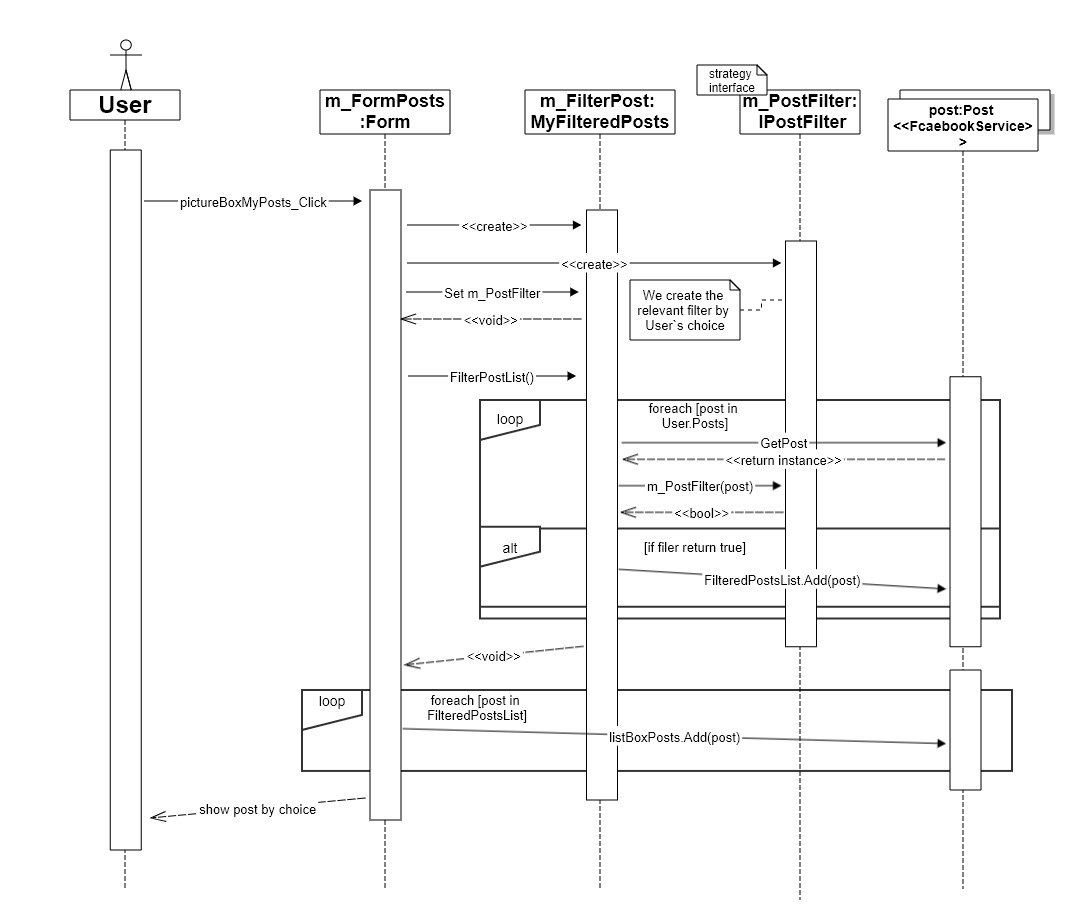
בפרוייקט הUI במחלקה FormPosts –

למשתמש מופיעים צ'ק בוקסים כשמות המחלקות לסינון שמימשנו , וכאשר הלקוח בוחר צ'ק בוקס, כלומר איזה פוסטים הוא רוצה לראות לאחר סינון, המחלקה FormPosts יוצרת מופע של האסטרטגייה הרלוונטית לסינון שנבחר ושולחת אותו למחלקה MyFilteredPosts, על מנת שמחלקה זו תיצור את הרשימה המסוננת.

הרשימה המסוננת שנוצרה תיווסף ותופיע ברשימת הפוסטים בטופס.

בביטול אסטרטגיית החיפוש בה בחר הלקוח , מתאפשר שינוי אסטרטגיה בזמן ריצה.

**Sequence Diagram – Strategy**

****

**"השחקנים" בתבניות עיצוב:**

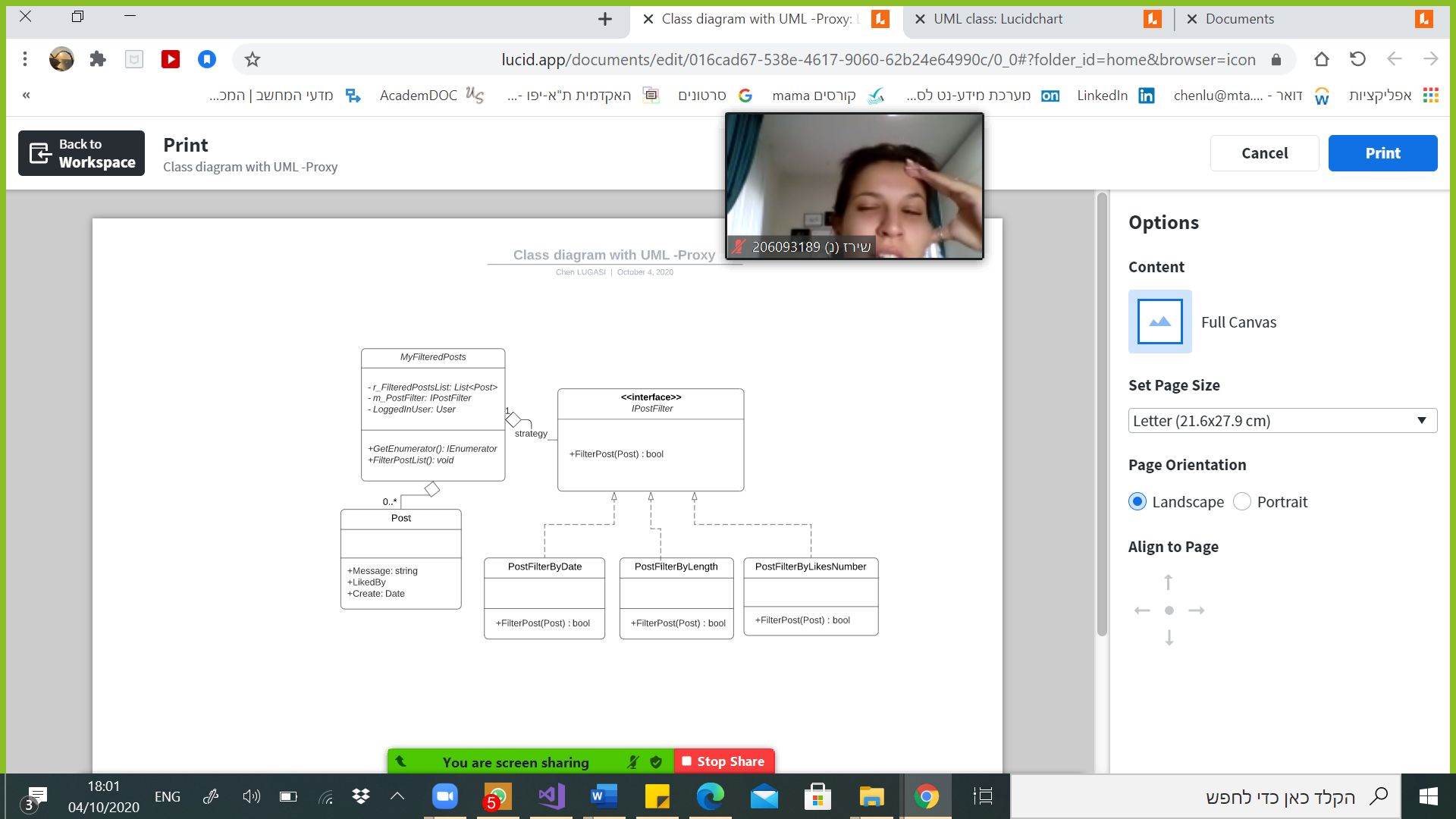
**Context** : MyFilteredPosts

**Strategy** : IPostFilter

**ConcreteStrategy :**

* PostFilterByDate
* PostFilterByLength
* PostFilterByLikesNumber

**Class Diagram - Strategy**



### תבנית מס' 2 – [Iterator]

**סיבת הבחירה / שימוש בתבנית:**

מטרת התבנית לשמש כלי לסריקה של אוספים שונים בצורה דומה וכללית ללא תלות באופן מימוש האוספים כלומר, במבני הנתונים בהם הם מוחזקים.

באפליקציה שלנו יש עבודה עם רכיבי aggregate, כלומר רכיבים שמחזיקים אוסף כלשהו.

בפרט המחלקות הבאות:

1.**FacadeMatch** שנמצאת בחלק הלוגי, קיים אוסף של חברים (Friends ) המתאימים לזיווג למשתמש ע"פ בחירת מגדר שהמשתמש בחר.

אנו רוצים לתת גישה בUI ללקוח לסריקת קולקציה זו מבלי לחשוף את מבנה הנתונים של אחד מהאובייקטים וללא תלות במימוש מבנה הנתונים שלהם, על מנת שבעתיד תהיה לנו האפשרות לבצע שינויים במחלקה FacadeMatch ללא שינוי הקוד אצל המשתמש והתוכנית עדיין תהיה תקינה , ולכן תבנית זו עוזרת לפתרון הבעיה בכך שמהמחלקה FacadeMatch מממשת את המתודה GetEnumarator בעזרת פיצ'ר yield return – פיצ'ר זה יוצר "מאוחרי הקלעים" מחלקה חדשה מסוג IEnumarator עם חמשת המתודות הדרושות : T GetCurrent, object GetCurrent, Dispose(), IsDone, MoveNext .

ולכן בכל ריצה של foreach על המחלקה FacadeMatch, כאשר נגיע לשורה yield return , נקבל את המופע של האיבר הבא.

בכך שאנחנו חושפים את השירות GetEnumarator אנחנו מאפשרים למשתמש לסרוק את הנתונים שהוחזרו לו בלי הצורך ובלי תלות לדעת את מבנה הנתונים.

2. **MyFilteredPost** – שנמצאת בחלק הלוגי, ומכילה אוסף פוסטים ע"פ הסינון שהוגדר בדיזיין strategy שדיברנו עליו לעיל.

הסיבה שבגינה השתמשנו במחלקה זו באיטרטור, היא מכיוון שאנו רוצים להגן על החלק הלוגי מממשקי משתמש שונים שעושים בו שימוש וצריכים לסרוק את הקולקציה , כך שממשקים אלו לא יוכלו לפגוע באוסף עצמו ( לדוגמא – clear ) , לכן נרצה לאפשר למשתמשים בUI לעבור על הקולקציה בעזרת איטרטור .

**אופן המימוש:**

נראה את אופן המימוש בשתי המחלקות השונות :

1. **FacadeMatch** : המחלקה יושבת בחלק הלוגי, ומממשת את הממשק :IEnumerable<User>

מתודת הממשק- GetEnumerator(), עושה שימוש ב- yield return (כפי שהסברנו בסיבת הבחירה) .

מימשנו במחלקה זו filter Iterator, על מנת שהאיטרטור יתקדם רק לחברים שיש להם התאמה למשתמש. הסינון מתבצע ע"פ גלגל המזלות שמוגדר במחלקה זו.

תחילה מקבלים רשימה של חברים שממויינת על פי מגדר ממחלקה שנקראת ProxyFriends , (זה לאחר שהלקוח בחר מגדר למיון ) .

המתודה עוברת בלולאה foreach על כל חבר ברשימה המסוננת ובודקת האם הוא מתאים מבחינת גלגל המזלות ללקוח שלנו , במידה וכן נגיע לyeild return שיחזיר את האיטרטור לחבר הנוכחי הנבדק.

איך הClient סורק את הקולקציה:

בחלק הUI במחלקה FormMatch – אנו רוצים להבין את מי מהחברים להוסיף לפאנל – "המתאימים לזיווג" , ולכן עוברים בלולאת foreach על המחלקה FacadeMatch, שמחזירה איטרטור לאיבר הנוכחי בכל התקדמות של הלולאה:

foreach(User fb\_Friend in r\_FacadeToFormMatch)

החזרת האיבר הנוכחי מתבצעת כפי שציינו ב- yield return.

ולכן יהיה מופע של אובייקט מסוג FacadeMatch בטופס FormMatch, שבעזרת אובייקט זה נוכל לסרוק את החברים המוחזרים מבלי לדעת את סוג מבנה הנתונים שהם מאוחסנים בו.

1. **MyFilteredPost –** מחלקה זו נמצאת בחלק הלוגי, ומחזיקה אוסף של פוסטים לאחר שעברו סינון כפי שהסברנו בפטרן Strategy.

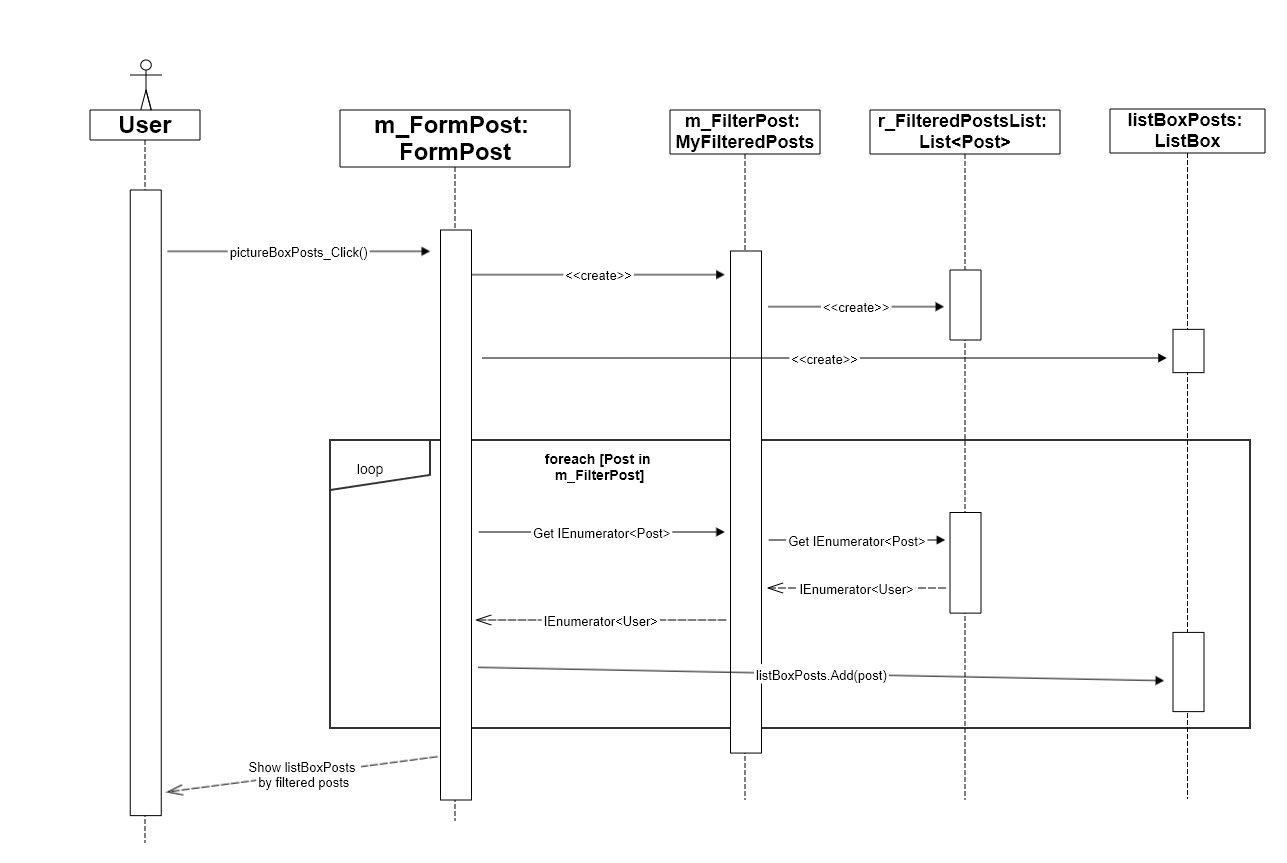
מחלקה זו מממשת את הממשק IEnumarable<Post> **,** כך שבמתודה GetEnumarator אנו מחזירים את האיטרטור של הקולקציה המסוננת . כפי שנלמד בכיתה , כל קולקציה שנמצאת בsystem.Collection מממשת את הממשק IEnumarable .

איך הClient סורק את הקולקציה:

בUI תחת טופס פוסט (FormPosts) , תחת המתודה fetchAllPosts בבחירת הסינון הרלוונטי המשתמש יעבור בלולאת foreach על המופע של המחלקה myFilterPosts ויוסיף לרשימת הפוסטים בטופס את הקולקציה אותה הוא סורק.

חילקנו את הדיאגרמות לשני הטפסים הרלווטים לטובת נוחות קריאה.

* במידה ואין בכך צורך , אפשר להתייחס רק לאחד מהם . תודה.
* **Sequence Diagram - Iterator**

**1.FormPosts**

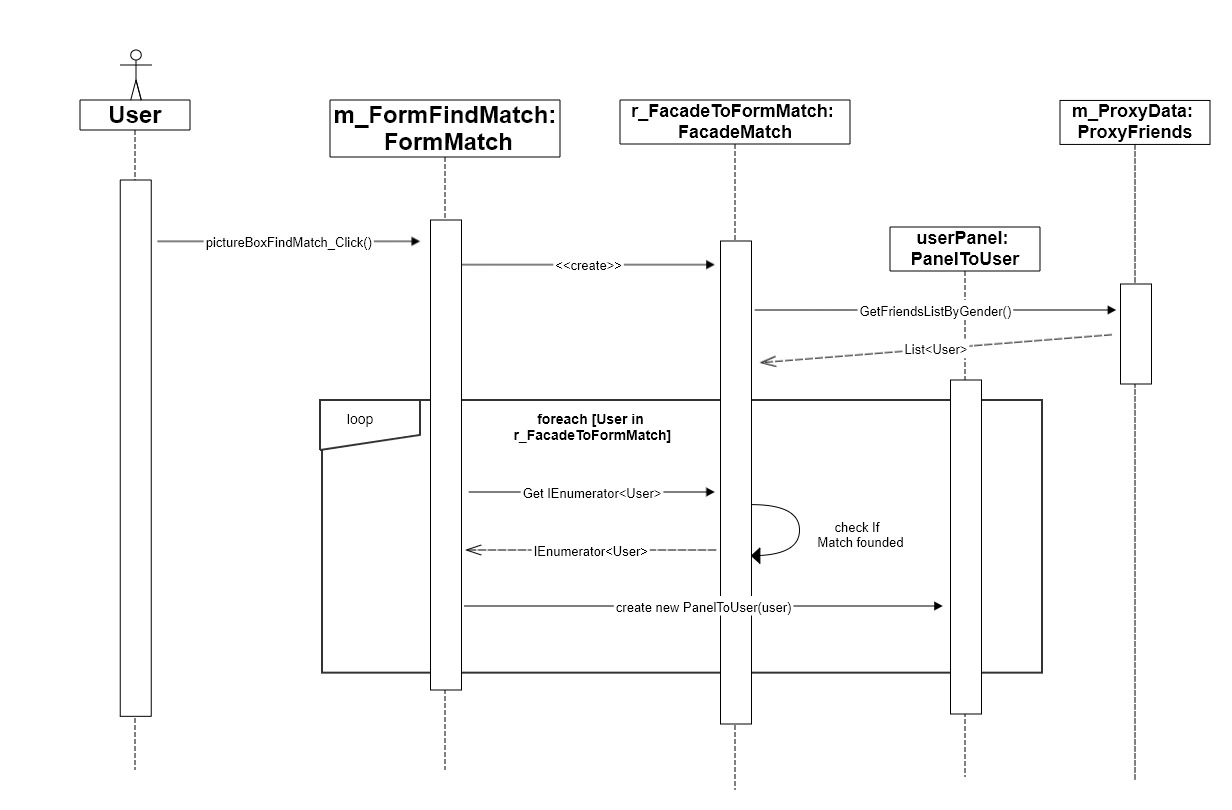
השחקנים בתבנית עיצוב הם :

Client – FormPosts

Aggregator - MyFilteredPosts

- Iterator – מסופק ע"י הרשימה המוחזקת במחלקה MyFilteredPosts

**2.FormMatch**



**השחקנים בתבנית עיצוב הם :**

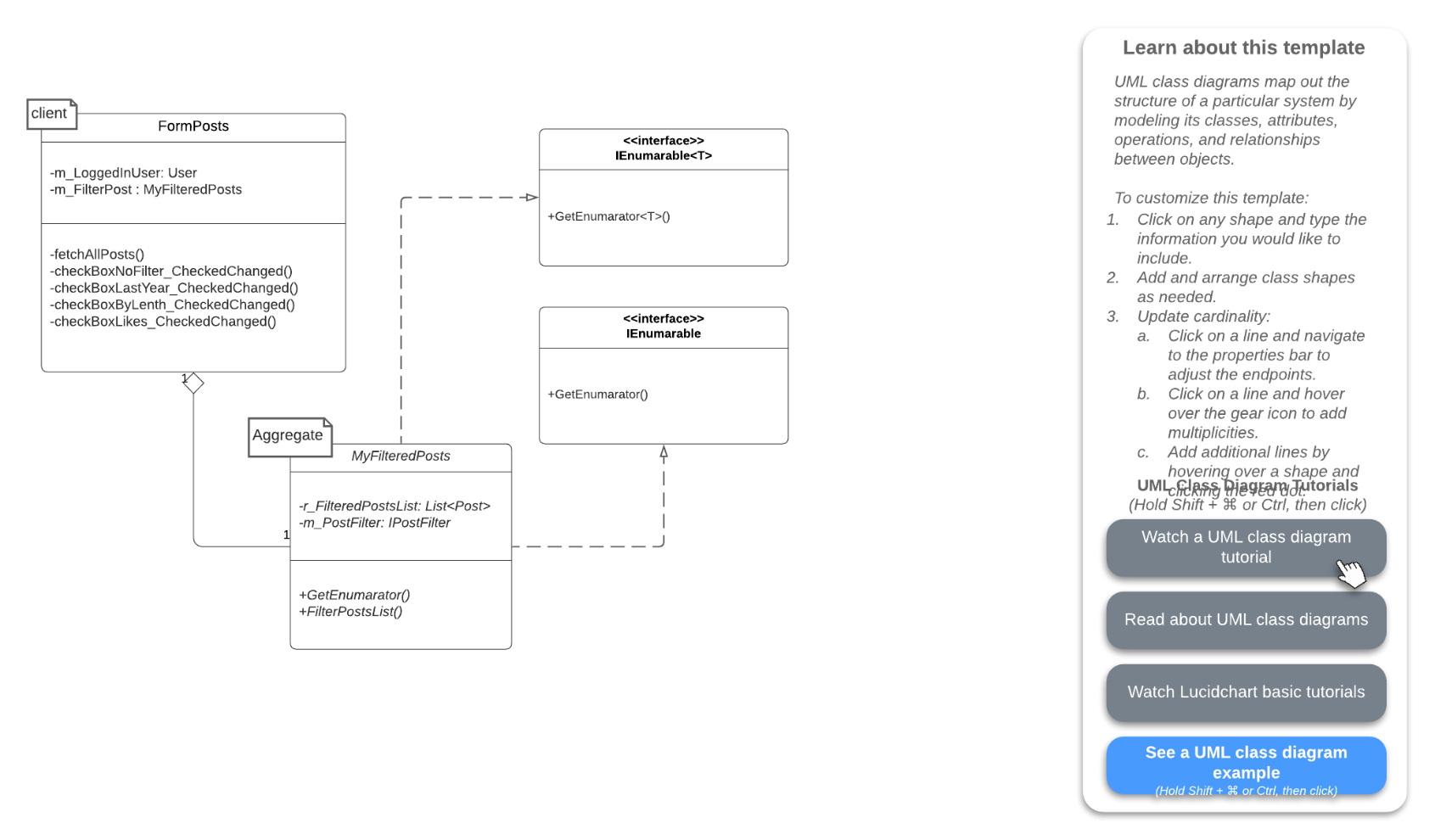
Client - FormMatch

Aggregator - FacadeMatch

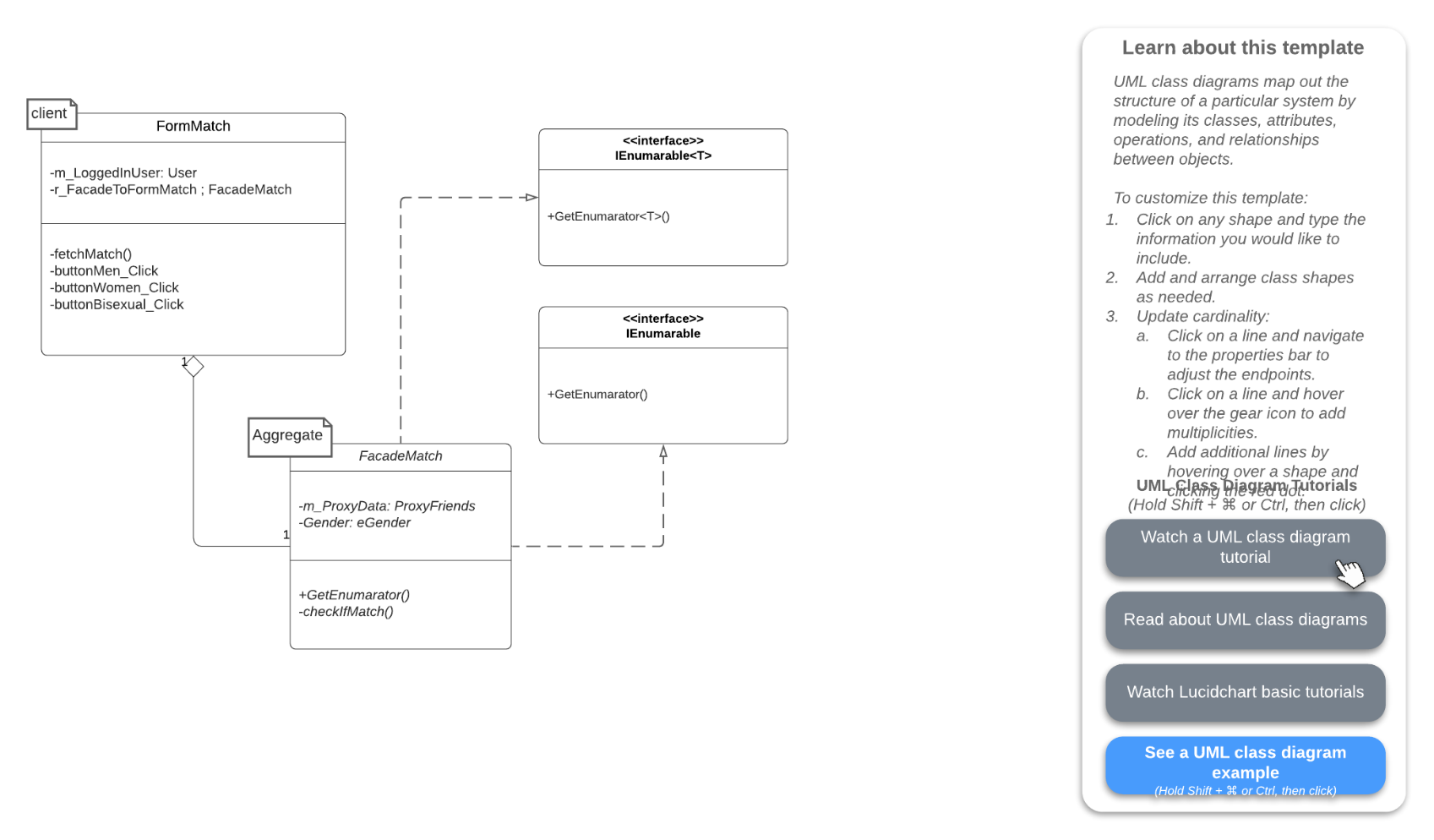
Iterator- מסופק ע"י yield return

* **Class Diagram - Iterator**

**1.FormPosts**

****

**2.FormMatch**



### תבנית מס' 3 – [Observer]

**סיבת הבחירה / שימוש בתבנית:**

הצורך ליידע חלקים שונים בתוכנית על אירועים שהתרחשו הוא אלמנט חשוב היום בעולם האפליקציות ( מערך Notifications ), גם באפליקציה שלנו היינו רוצים להתריע על שינויים ולידע חלקים בתוכנית על אירועים שהתרחשו.

באפליקציה שלנו ישנו פיצ'ר – FormCorona שבו המשתמש מודיע כי הוא חולה בקורונה בעזרת פוסט שמפרסם.

אנו רוצים שכאשר אדם מודיע שהוא חולה בקורונה, תמונת הפרופיל שלו תתעדכן בכך שיתווסף הכיתוב שהוא חולה (#CoronaSick ) , ובנוסף רשימת המקומות שהוא שהה בהם בחודש האחרון יסומנו כמקומות "אדומים" כלומר אלו הם המקומות ששהה בהם חולה קורונה.

פטרן האובסרבר עונה לנו על הצורך בכך שהוא מאפשר לאובייקטים שונים להירשם כ"מקשיבים" לאותו אירוע במערכת.

בנוסף לא רצינו שיהיה קשר ציקלי בין ישויות ותבנית זו מאפשרת זאת ע"י loose coupling בעזרת שימוש בdelegate כNotifier .

על כן , טופס הקורונה יכיל כפתור (=pictureBox ) "חכם" המורה על פרסום הפוסט, כפתור זה משמש כ"מודיע" שפורסם פוסט קורונה.

האובסרברים המממשים את הממשק IObserverCoronaSick הינם - הטופס הראשי , ירשם "כמקשיב" לכפתור זה על מנת לעדכן את תמונות בפרופיל ובנוסף טופס הצ'קאין ירשם כ"מקשיב" על מנת לעדכן בצבע אדום את רשימת המקומות בהם המשתמש שהה בחודש האחרון.

בעתיד יכול להיות שיהיו חלקים נוספים בתוכנית שירצו לפעול בהתאם, כלומר להקשיב לכך שהמשתמש חלה בקורונה (כפתור הפרסום הופעל), כל מה שהם יצטרכו לעשות זה להירשם כמקשיבים לכפתור החכם ולממש את הממשק IObserverCoronaSick.

**אופן המימוש:**

בפטרן זה נעזר באפשרויות של C# בעזרת Action - public event

בחלק הUI ,יצרנו מחלקה של כפתור חכם : SmartPictureBox , אשר יורשת מPictureBox

במחלקה זו הצהרנו על אירוע **: public event Action CoronaSick**,

כך שבכל פעם שהמשתמש יפרסם שהוא חולה בקורונה , בעזרת לחיצה על כפתור זה , נבצע Invoke לרשימת המאזינים , כלומר נודיע לכל מי שנרשם לאירוע , שפורסם פוסט קורונה.

* במחלקת ה- FormCorona , ישנו data member מסוג SmartPictureBox , והוא מופעל ע"י המשתמש בעת לחיצת "פרסם פוסט".

יצרנו בנוסף ממשק – IobserverCoronaSick , המכיל מתודה אחת – UpdateCoronaSick().

האובייקטים שירצו להירשם כמקשיבים לכפתור פרסום הפוסט, יצטרכו לממש ממשק זה.

המחלקות המקשיבות שבאפליקציה שלנו הן :

* מחלקת FormFacebook , המממשת את מתודת הממשק הנ"ל, מתודה זו מופעלת כאשר המודיע מבצע Invoke , כיוון שהטופס הראשי , נרשם באמצעותה לאירוע :

m\_FormCorona.PostButton.CoronaSick += new Action(user\_CoronaSick)

מתודה זו אחראית לעדכן את ה- label שמוצמד לתמונת הפרופיל (#CoronaSick ) ,

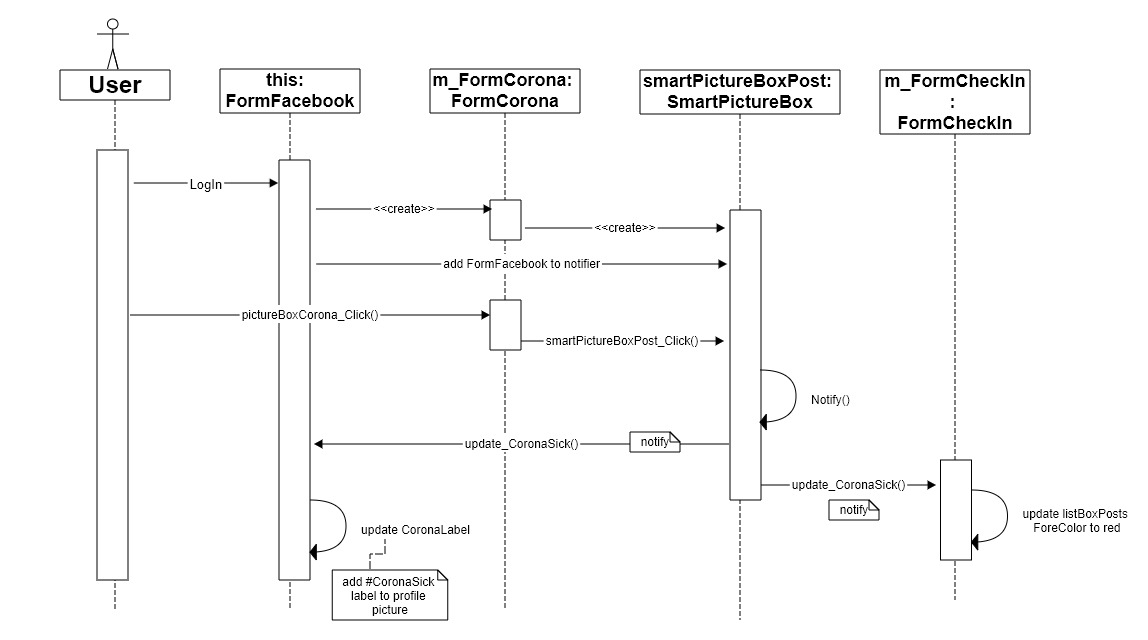
* מחלקת FormCheckin, המממשת את מתודת הממשק הנ"ל, מתודה זו מופעלת כאשר המודיע מבצע Invoke , כיוון שהטופס צ'ק אין נרשם כך:

m\_SmartPictureBoxPostNotifier.CoronaSick += new Action(UpdateCoronaSick);

אזי , הטופס מציג את רשימת המקומות בהם שהה המשתמש בחודש האחרון בצבע אדום, בתוספת כותרת – "קורונה בחודש האחרון "

(אחרת מציג את כל המקומות שבהם שהה בצבע שחור).

בנוסף, נתנו למשתמש אפשרות להוריד את הלייבל מתמונת הפרופיל, ולהחזיר את טופס הצ'ק אין לברירת המחדל, בכך שהוא לוחץ על כפתור " החלמתי " .

**Sequence Diagram- Observer**

"השחקנים" בתבנית העיצוב:

**Subject :** SmartPictureBox

**Observer:**

* FormCheckin
* FormFacebook

**Class Diagram - Observer**

