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

* **חייה וירטואלית (טמגוצ'י)** – פיצ'ר זה מוסיף למשתמש חוויה נוספת לפייסבוק שהיא בחירת חייה וירטואלית.

ניתן לבחור מבין מספר חיות וירטואליות. יש לבחור לחיה שם ולהאכיל אותה כדי שמדד השמחה שלה יישאר גבוה. במידה ומדד השמחה של החיה יהיה נמוך, תצא הודעה למשתמש שהוא צריך להאכיל את החיה. ניתן לשנות את החיה בכל שלב.

כמו כן, לכל משפחת חיות יש אוכל שמתאים לה המתחלף בעת בחירת החיה.

פרטי החיה שהמשתמש בחר נשמרים ומוצגים לאחר סגירה ופתיחה של האפליקציה.

* **טיימר** – פיצ'ר זה מספק למשתמש התראה על זמן השימוש באפליקציה. המשתמש יכול בכל שלב ללחוץ על האייקון של השעון כדי לדעת כמה זמן הוא משתמש באפליקציה ובנוסף להגדיר טיימר שיתריע למשתמש שמומלץ לעשות הפסקה לפי הזמן שנבחר מבין האפשרויות בטופס. בחרנו לעשות פיצ'ר זה כי שימוש ברשתות חברתיות גורם להתמכרות ורצינו ליצור מנגנון שימחיש למשתמש את זמן השימוש באפליקציה וייתן למשתמש התראה לקחת הפסקה, הוספנו מנגון נוסף של שעון עצר.

### תבנית מס' 1 – [Static Factory Class]

* סיבת הבחירה / שימוש בתבנית:

בחרנו להשתמש בתבנית Factory Methodלצורך הגדרת מחלקה עבור כל סוג חיה קונקרטית ומחלקת בסיס אבסטרקטית לכל החיות.

בעת בחירת חיה כלשהי באמצעות המחלקה ChoosePetForm, נוצר אובייקט של אותה החיה באמצעות המחלקה הסטטית AnimalFactory שמכילה מתודה סטטית היוצרת ומחזירה את המופע הרלוונטי, כמו כן נשמרים תמונת החיה וסוג החיה (מבין מספר אפשרויות המוגדרות ב-Enum).

כאשר בוחרים חיה, למשל חתול, נוצר מופע מסוגCat הנשמר למשתנה מסוג Animal.

בשלב זה, מוגדרים הנתונים של כל חיה (תמונה וסוג החיה במקרה זה Domestic).

מימשנו תבנית זו על מנת שנוכל להרחיב את פיצ'ר החיה הוירטואלית בעתיד ונוכל להוסיף ולשנות פונקציונליות כדי להבדיל בין כל סוג חיה על ידי הוספת תכונות ייחודיות לכל טיפוס קונקרטי.

רצינו שפיצ'ר זה יהיה ניתן להרחבה בצורה פולימורפית ולמנוע שכפול קוד.

הבנו שכדאי לנו ליצור לכל חיה מחלקה משלה שתירש ממחלקה אבסטרקטית בשם Animal כך שאם בעתיד נרצה להוסיף חיות או להוסיף מידע שיישמר על כל חיה, נוכל לקבל ביורשה מתודות ותכונות אבסטרקטיות ממחלקת Animal.

בנוסף רצינו להראות שימוש פולימורפי לסוג חיה, הראנו זאת בכך שלכל סוג חיה יש אוכל אחר שמתאים לחיה כך שנוכל להוסיף סוגי חיות נוספים בעתיד ונוכל להוסיף אותם לסיווג שיצרנו.

* אופן המימוש:

ניתן למצוא את המימוש של תבנית העיצוב בתיקיית Factory Method.

מחלקת הבסיס שהיא אבסטרקטית – Animal ובה יש שדה מסוג eAnimalCategory.

הטיפוסים הקונקרטיים – Bear, Bird, Cat Chicken, Dog, Panda, Pig, Snail, Turtle, Wolf שיורשים מ-Animal.

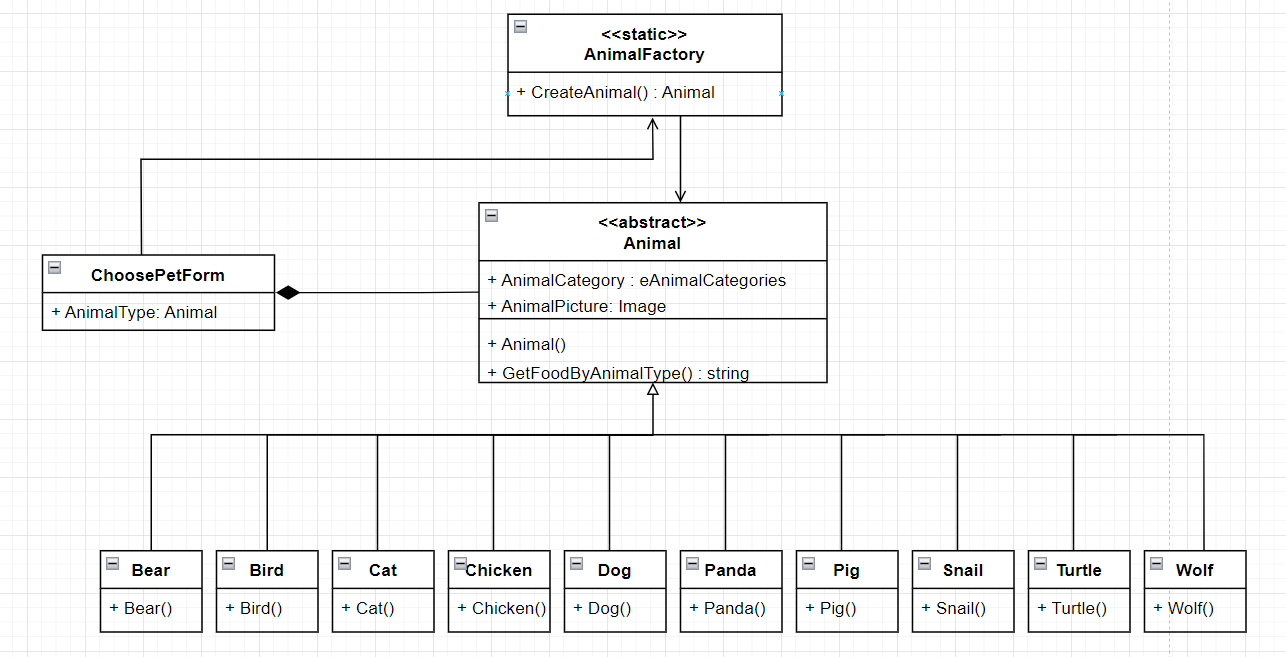
ומחלקה לייצור החייה הקונקרטית – AnimalFactory היוצרת ומחזירה את הטיפוס הקונקרטי של החיה שנבחרה.

המחלקות והקוד נמצא בתקיית Factory Method.

* Sequence Diagram

תמונה שמכילה טקסט, מקורה, מפה

התיאור נוצר באופן אוטומטי

* Class Diagram

### תבנית מס' 2 – [Adapter]

* סיבת הבחירה / שימוש בתבנית:

בחרנו להשתמש בAdapter בפרוייקט שלנו כדי לבצע התאמה בין שתי רכיבי לגאסי,הראשון הוא FromMain שמימשנו בחלק הראשון של הפרוייט, והחלק השני הוא POST שנמצא בתוך FacebookWrapper.ObjectModel (אקרא לה בקצרה POST בהמשך ההסבר).

כדי לעשות זאת החלטנו להשתמש בשתי Adapter's, כדי לענות על שתי בעיות שונות שפגשנו בדרך:

ראשית למחלקת FacebookWrapper יש תכונות(properties ( שכבר אינם פעילים, ולכן יצרנו Adapter שלוקח רק את התכונות שרלוונטיות כרגע בקוד שלנו, אבל נותן לנו אפשרות להוסיף לממשק תכונות חדשות שלנו/תכונות שקיימות אך נממש אותם בצורה שונה.

לדוגמא : LikeCount עבור POST אינו פעיל אז בADAPTER שיצרנו אנו מוסיפים LikeCount משלנו עבור כל פוסט.

ובאותו אופן במחשבה לעתיד אם דברים אחרים יפסיקו לפעול או שנרצה להוסיף עוד דברים לממשק של POST.

רכיב לגאסי ראשון -קליינט – הוא הAdapter השני שאנו ממשים.

ADAPTER – PostAdapter.

רכיב לגאסי שני - POST.

שנית אנו החלטנו להשתמש בDataBinding two way עם הפוסטים שלנו, וכשעשינו זאת אז נתקלנו בכמה בעיות : התמונה של POST.PictureURL לא התקבלה טוב כPictureBox , הDescription של הפוסט היה ארוך מידי לTextBox או label וריצנו להפוך אותו לlistBox.

לכן יצרנו Adapter נוסף שמותאם עבור WinForms כלומר עבור ממשק אחר לא יהיה בו צורך, שבו אנו פותרים את הבעיות שלנו ומוסיפים לממשק את הרכיבים הדרושים.

במקרה זה :

רכיב לגאסי ראשון -קליינט – הוא מחלקת FormMain שנעזר במחלקה שלנו FacebookPosts.

ADAPTER – PostAdapterWinForms.

רכיב לגאסי שני - PostAdapter.

אופן המימוש:

**PostAdapter**

ממומש כObject Adapter בתוך תיקיית Adapters ואליו קשורים הקבצים :

והמחלקות הרלוונטיות לפי הclass diagram.

**PostAdapterWinForms**

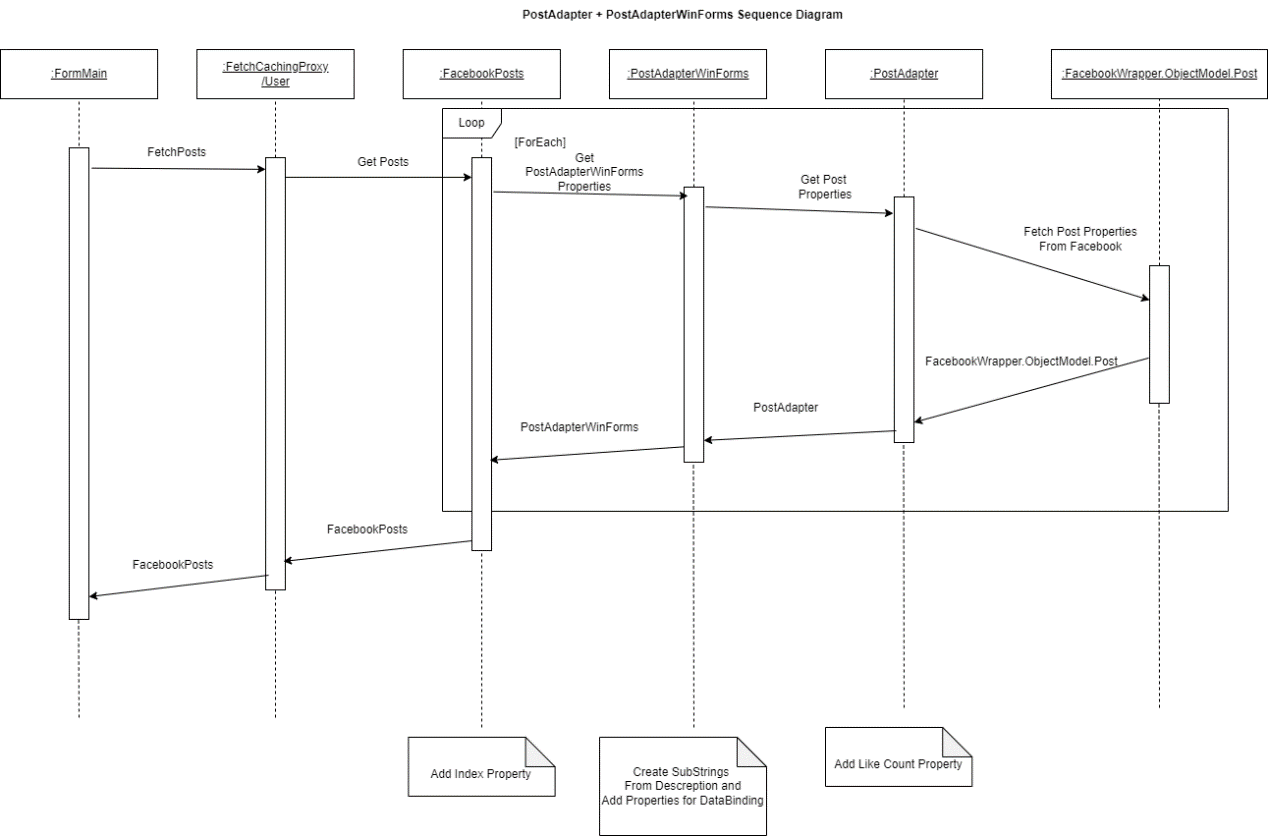
ממומש כClassAdapter כי הוא מסתמך על הבסיס הכללי יותר שעשינו עם הPostAdapter .

לשניהם יש ממשק במחשבה על תוספות לעתיד, במידה ונרצה להוסיף את נתונים או להוסיף פעולות שלא קיימות כרגע.

בחרנו לבצע את הSequence Diagram שמשלב את שני האדפטרים יחדיו, בפעולה הראשונית של טעינת הפוסטים למערכת, מצב זה קורה במתחיל במתודה FetchCachingProxy(LoggedInUser).

משם נלקח המידע של היוזר וFacebookPosts עושה העתקה עמוקה של POST לPostAdpapteWinForms מחלקה זה משתמשת בPostAdapter והיא לוקחת את המידע מPOST מהשרת פייסבוק – (לוקח זמן) ומקבלת ממנה גם את מס' הלייקים המתאים לכל פוסט.

לאחר מכן PostAdpapteWinForms מוסיפה את התכונות שלה לכל פוסט ומחזירה אותם לFacebookPosts שמחזיר לעמוד הראשי.

* Sequence Diagram
* Diagram

  Description automatically generatedClass Diagram

### תבנית מס' 3 – [CachingProxy]

* סיבת הבחירה / שימוש בתבנית:

בפרוייקט שלנו הדבר שלוקח הכי הרבה זמן הוא לקיחת המידע מהאינטרנט ורצינו למצוא פתרון כדי לשפר את הפרוייקט מבחינת ביצועים.

רצינו לשפר את הביצועים של המערכת ולהפחית את הפעמים שצריך לגשת לשרת הפייסבוק, חלק מהמערכת שלנו הוא קבלת הנתונים ע"י לחיצה של כפתורים אחד מהכפתורים (Posts,Friends,Photos,LikedPages,All ) שנמצאים על המסך הראשי.

וכל פעם שרצינו לקבל את המידע לחצנו על הכפתור וחיכינו לקבל מידע מהשרת, כך שכל פעולה כזו לקחה זמן.

לכן הפתרון שלנו הוא להתשמש בCachingProxy ויצרנו מחלקה בשם FetchCachingProxy מחלקה זו תשמור את הנתונים של UserAdapter (יצרנו אדפטר כי USER הוא רכיב לגאסי שלא נוכל לשנות ולהוסיף לו interface ) בטעינה הראשונית של האפליקציה, וכשהמשתמש ינסה לקרוא לאחד מהפעולות ע"י לחיצת הכפתור, במקום להשתמש בUser דרך השרת, מחלקת הProxy שלנו תתחזה אליו ותחזיר את הנתונים שהם כבר שמורים במערכת אצלה וכך תחסוך את כל זמן הטעינה והמערכת תשפר את הביצועים.

מעבר לכך אם נרצה בעתיד נוכל להוסיף אפשרות של שמירת הנתונים לקובץ וטעינה מהם, כך שאם הייתה התחברות, ופעם לאחר מכן ננסה להתחבר ולא יהיה חיבור אינטרנט אז נוכל לטעון מקובץ את המידע הרלוונטי ( יצרנו הכנה לכך במחלקת FetchCachingProxy לשימוש עתידי – אנו בודקים האם יש אינטרנט לפני שטוענים את הקובץ ניתן לממש בעתיד שאם אין אינטרנט אז נטען מקובץ ).

הערה:

נתון שלא להשתמש בUserProxy במטלה אבל אנו השתמשנו בצורתCachingProxy כדי לשפר ביצועים, וזו סיבה שונה ממה שהוצג בכיתה,בנוסף מימשנו בצורה שונה ממש שלמדנו בכיתה.

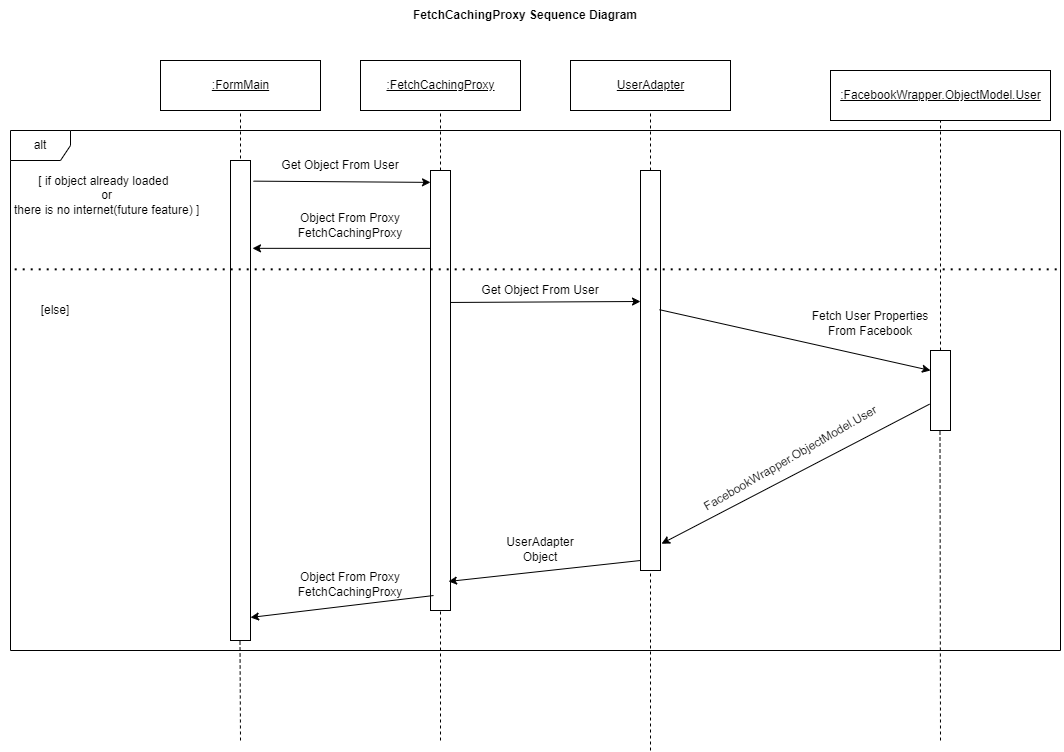
* אופן המימוש:

מיקום : בתוך תיקיית ObjectProxy מופיעות המחלקות, השימוש מתבצע בFormMain במתודת fetchCachingProxy.

מימשנו את הCachingProxy כObjectProxy , יש לנו ממשק בשם IFetch שיש לו את כל התכונות שאנו יכולים למשוך מהיוזר.

כאשר אנו טוענים בפעם הראשונה FetchCachingProxy ימשוך את המידע דרך השרת, בכל פעם אחרת אנו נחזיר את המידע הקיים.

מכיוון שבמערכת שלנו המידע נטען על פי דרישה של ליחצה מהמשתמש, גם בפעם הראשונה שנלחץ על הכפתור המידע כבר יהיה אצל המחלקת Proxy שלנו מכיוון שטענו אותה ב OnShown לאחר שהתחברנו למערכת.

* Sequence Diagram
* Diagram

  Description automatically generatedClass Diagram

**DataBinding**

בחרנו לעשות שימוש בדאטה ביינדינג עם feature הפוסט שעשינו בפרוייקט במטלה 1.  
תזכורת :  
אנו מציגים פוסט אחד כל פעם עם כפתורי חצים ימינה ושמאלה כדי לעבור לפוסט הבא או הקודם.   
במטלה הקודמת היה לנו כמה עשרות שורות קוד כדי למלא את כל הפרטים על הפוסט (תמונה, תיאור ,שם ,תאריך ועוד).

בעזרת הPostAdapterWinForms + PostAdapter הוספנו LikeCount שלא מתקבל משרת הפייסבוק, החלטנו שמספרי הלייקים כרגע יהיו מספרים רנדומליים ולא מקובץ/DB כי זה לא היה העיקר.  
בנוסף יצרנו קטעי מחרוזת מDescription בתוך מחלקת הPostAdapterWinForms כדי שנוכל להכניס את זה לListBox בצורה יפה.

יצרנוbindingSource עבור הפוסטים, וגם עבור הdescreption כדי להתאים את זה לצרכנו.  
חיברנו את postAdapterWinFormsBindingSource למידע שלנו ע"י :

שורה אחת בלבד :

כך שעבור כל פוסט נטען את המידע שלו, כאשר נזוז עם החץ נטען את הפוסט באינדקס אחריו בתור.

את הlistbox של הdescription אנו טוענים ע"י חיבור DataSource עם הGUI של WINFORMS :

Graphical user interface, text, application

Description automatically generated

**עבודה אסינכרונית**

השתמשנו בעבודה אסינכרונית עם Threads ב:

* כשאנו טוענים את המידע לFetchCachingProxy , כלומר טעינת (פוסטים,תמונות,רשימת חברים, דפים שעשינו להם לייק), מכייון שאנו טוענים אסינכרונית , אנחנו עושים לכפתורים של הפוסטים,תמונות וכו' Disable כברירת מחדל, ומאפשר את הכפתורים לאחר שטענו את הנתונים, כדי שלא יהיה Thread נפרד שיפעיל את הכפתורים לפני שטענו את הנתונים – נקבל שגיאה.
* טעינת הלוגויים והתמונות שאנו משתמשים בהם במערכת.
* טעינת תמונות החיות בפיצ'ר החיות במערכת.
* טעינת השם של המשתמש, תמונת פרופיל.
* טעינת הIntro של המשתמש.

אנו הוספנו לפריטים שצריכים להשאר קשורים Thread המקורי שלהם כפי שלמדנו מהסרטון את Invoke(new Action(()=>…) למשל כל הפריטים שקשור לUI שפועלים בתוך Thread נפרד.

Text

Description automatically generatedText

Description automatically generatedדוגמאות מהקוד מ FormMain :