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

* הפיצ'ר מאפשר למשתמש ליצור רשימת השמעה מתוך השירים ששיתף בפוסטים שלו בעבר ולנגן אותם בפורמט וידאו או אודיו.
* הפיצר מאפשר חיפוש מתקדם לפי מילת מפתח, לפי מין, גיל ועוד, בפוסטים של חברים או/ו איוונטים או/ו בקבוצות .

עבודה אסינכרונית:

ממומשת במחלקת ה UI – FacbookPlayerForm בפרוייקט FacebookApp. סיבת השימוש: רצינו לאפשר לחלון להיות רספונסיבי בזמן שהשירים של המשתמש שנתמכים במערכת מיובאים ונשאבים ל listbox. הפעולה נעשתה בעזת lambda expression inline שלא מקבלת פרמטרים ומפעילה את המטודה הפרטית: loadSongs(User i\_LoggedInUser)

פעולת הטעיתה ל listbox מתבעצת ע"י allSongsListBox.Invoke

שימוש ב Data Binding:

הצורך: מימוש טעינת חברים והפוסטים ל listbox באופן אוטומטי ע"י שורת קוד אחת באופן קל לתחזוק עם פוטנציאל נמוך לשגיאות.

ממומשת ב FormMain כאשר ListboxFriends ו ListboxPosts מוגדרים שתי Data Sources: userBindingSource ו postBindingSource בהתאמה. הקישור בין שתי ה BindingSources לקולקציית החברים והפוסטים מתבצעת במטודה

private void fetchUserInfo().

### תבנית מס' 1 – Factory Method

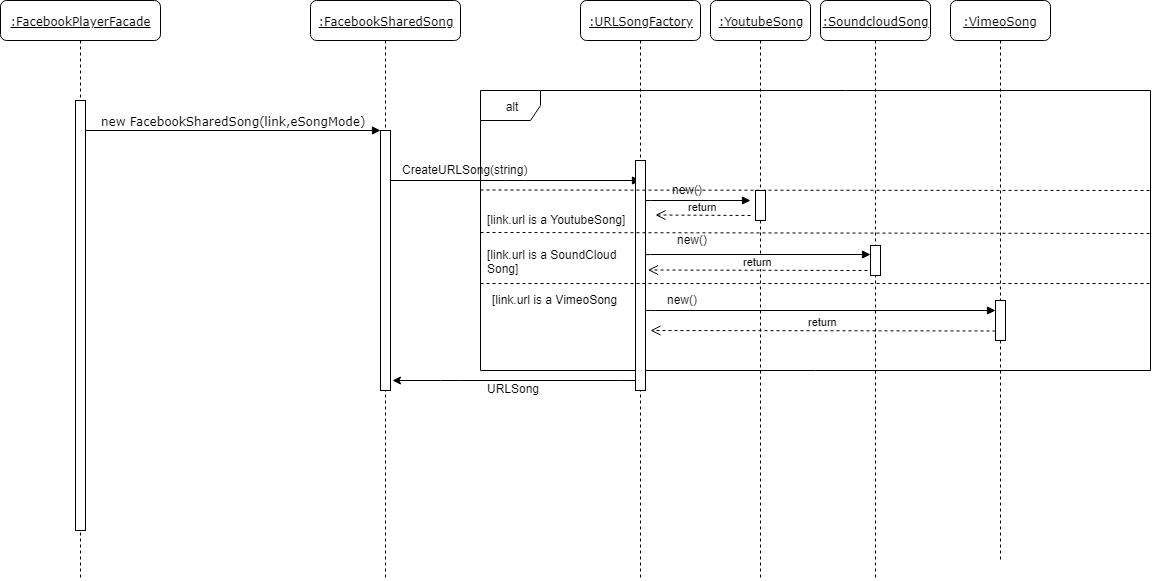
* סיבת הבחירה / שימוש בתבנית:

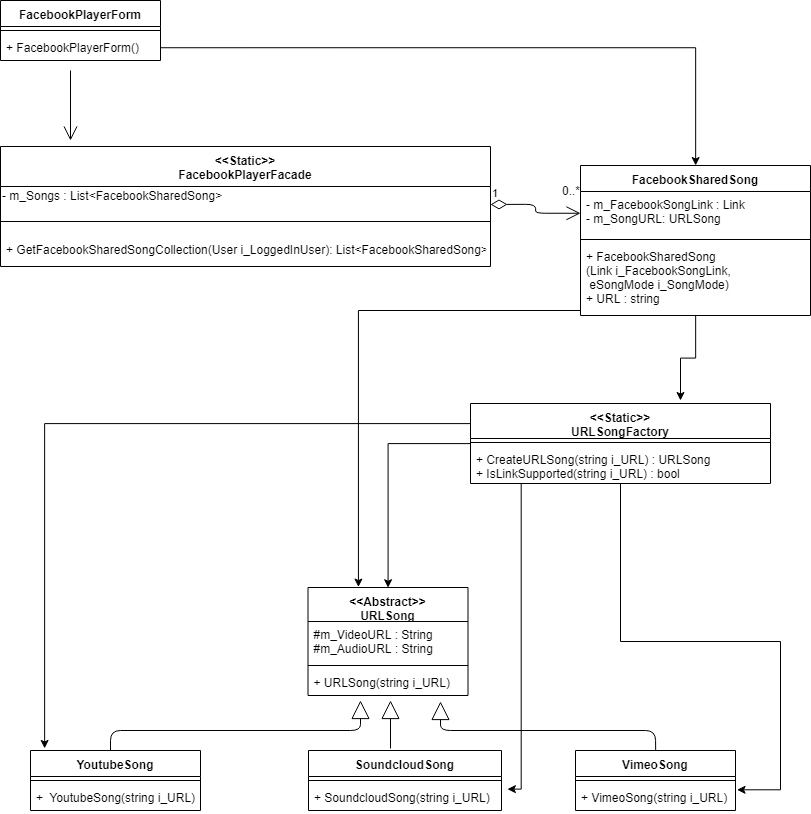
במערכת קיימת משפחה פולימורית URLSong abstract ממנה יורשים YoutubeSong, VimeoSong, SoundcloudSong לוגיקת הבחירת המחלקה ליצירה היתה ממומשת ב client באופן קשה לתחזוק להרחבת המשפחה הפולימורפית בעתיד.

* אופן המימוש:

Static URLSongFactory הנמצאת ב FacebookAppLogic מכילה כעת את לוגיקת הבחירה והיצירה של אובייקטים. הפונקציה CreateURLSong(string i\_URL) מקבלת מחרוזת ולפי התוכן שלה מיצרת אובייקט URL קונקרטי מהמשפחה הפולימורפית ומחזירה אותו בתור URLSong כך שהקליינט מכיר רק את האב הפולימורפי URLSong.

* Sequence Diagram





URLSongFactory => StaticFactoryMethod

URLSong => Abstract Base class

YoutubeSong, SoundcloudSong, VimeoSong => Concret classes

FacebookSharedSong => Client

### תבנית מס' 2 – Facade

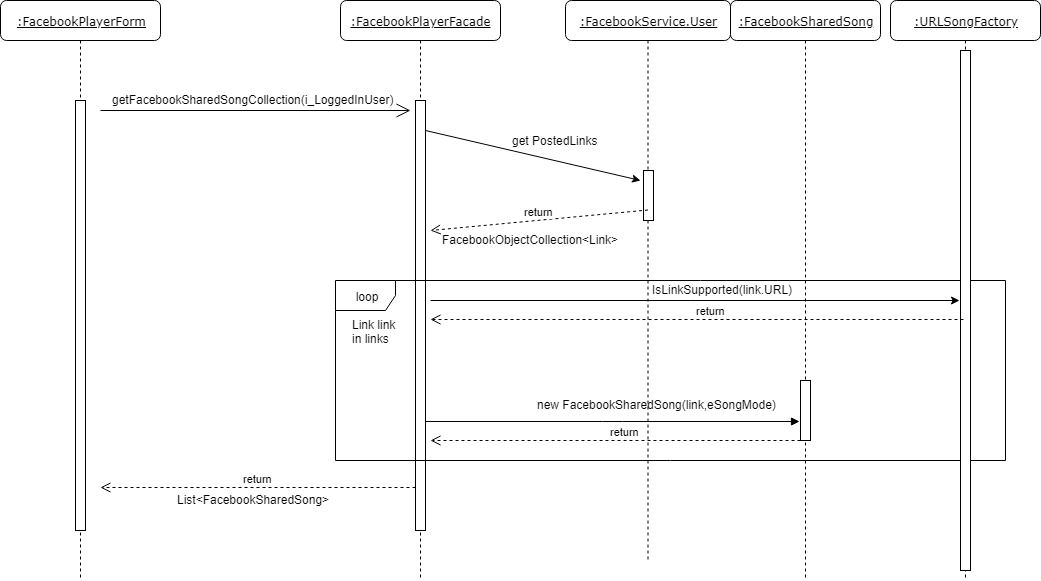
* סיבת הבחירה / שימוש בתבנית:

FacebookPlayerForm היא מחלקת UI שהכירה מספר מחלקות לוגיות (FacebookSharedSong, URLSongFactory) והשתמשה בהן באופן מסובך וכן ביצעה עוד פעולות על FacebookService. עלה הצורך לשטח את הלוגיקה ולספק ממשק פשוט יותר לרכיב ה UI.

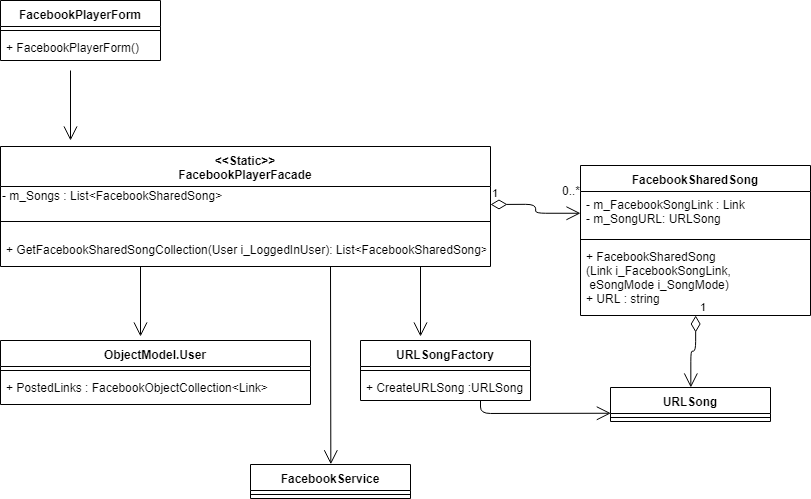
* אופן המימוש:

FacebookPlayerForm משתמש ב FacebookPlayerFacade שבין היתר מחזיק ואחראי על היצירה של הנתונים הלוגיים (FacebookSharedSong) שה Form משתמש בהם על מנת להציג שירים למשתמש ולאפשר לו ליצור רשימות השמעה.

* Sequence Diagram



* Class Diagram



FacebookPlayerForm => Client

FacbookPlayerFacade => Façade

ObjectModel.User, FacbookService, UrlSongFactory, URLSong => Logic classes hidden from the Client

### תבנית מס' 3 –Adapter

* סיבת הבחירה / שימוש בתבנית:

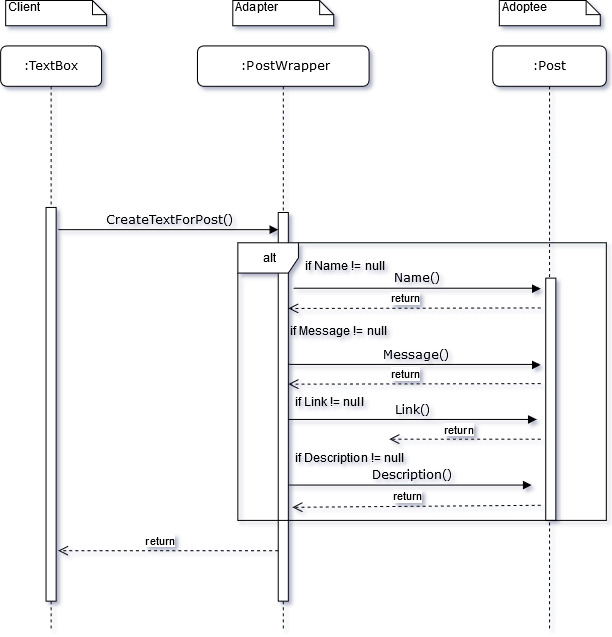
המחלקה TextBox במקרה שלנו היא הקליינט שמצפה לאינטרפייס (כלומר מטודה) שמביאה לו סטרינג מוכן של הטקסט של כל הפוסט. הסטרינג אשר כולל בתוכו מגוון שדות של המחלקה Post , בסדר מסויים, עם לוגיקה מסויימת כאשר יכול להיות שמספר שדות אפילו לא נמצאים בפוסט מסויים (null). למחלקה Post אין ממשק שכזה ולכן מתבקש לממש ADAPTER. במקרה שלנו המחלקה PostWrapper, שמגישה לקליינט אינטרפייס כמו שהוא ציפה.

מכאן נובעים עוד יתרונות כמו קוד ריוזבילי וקריא יותר. פחות לוגיקה בUI.

* אופן המימוש:

[המחלקה PostWrapper היא האדפטר. הקליינט במקרה שלנו הוא הTextBox שרוצה לקבל סטרינג מוכן של כל הטקסט של הפוסט הקיים, בסדר מסויים.   
אז לקחנו אובייקט Post ועטפנו אותו במחלקה PostWrapper ושם מימשנו מטודה createTextForPost() המממשת את מה שהקליינט דורש ומחזירה string. ]

* Sequence Diagram



* Class Diagram

