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

**פיצ'ר מס' 1 – "FaceRide!":**

פיצ'ר המאפשר למשתמש למצוא טרמפ מכל מקום לכל מקום בעזרת הרשת החברתית.

כדי להשתמש בפי'צר הנ"ל, על המתמש לבצע:

1. הזנה למערכת את נקודת האיסוף, או לחילופין שימוש במיקמו הנוכחי של המשתמש בעזרת שירותי מיקום.

2. הזנה למערכת את היעד, אותו יוכל להזין באופן ידני או "למשוך" מתוך רשימת האירועים אליו הוא רשום. לאחר מכן יתאפשר למשתמש לצפות ביעד בו בחר במפה בחלון נפרד.

3. בחירת רדיוס חיפוש רצוי ומגדר.

המערכת מאתרת מרשימת חבריו של המשתמש את כל אלו אשר נמצאים ברדיוס החיפוש בו בחר ומציגה אותם כרשימה המכילה מידע בסיסי אודותם. בעת בחירת המשתמש (לחיצה כפולה על החבר הרצוי), נפתח מסך ובו הודעה Auto Generated המוכנה לפרסום על הקיר או לשליחה דרך ה-Messenger, המציגה את בקשת המשתמש מאותו חבר לטרמפ אל היעד אליו הוא רוצה להגיע.

**מחלקות המעורבות בפיצ'ר**:

* AppForm
* FaceRideManager
* LocationServices
* AppFacade
* MapForm

**פיצ'ר מס' 2 – "Contacts":**

פיצ'ר זה נועד ליצור רשימת אנשי קשר מנתוני חברי הפייסבוק של המשמש ויצוא הרשימה לסמרטפון של המשתמש.

דרך הפעולה:

1. על המשתמש לעבר ללשונית הcontacts.

.2 לאחר מכן על המשתמש ללחוץ על כפתור הshow.

3 המערכת תיצור רשימת אנשי קשר ותזין אוטומטית את הנתונים לטבלה וויזואלית.

4. אופציונלי: המשתמש יכול לערוך בטבלה את נתוני החברים מהפייסבוק שהוזנו לפי טעמו.

5. אופציונלי: המשתמש יכול לערוך עמודות נוספות רלוונטיות לרשימת אנשי קשר שאינן חלק מנתוני הפייסבוק אך הן כן רלוונטית בשביל אנשי הקשר, חלק מעמודות אלו מותאמות בשמותיהן לתצוגה מיוחדת במכשרי הסמרטפון.

6. על המשתמש ללחוץ על כפתור הsave כדי לשמור את הנתונים כקובץ.csv

.7 בלשונית תפתח דפדפן פנימי אשר נמצא בדף הסבר מיוחד על איך לייצא את רשימת אנשי הקשר באופן מיידי לחשבון Gmail שיאפשר שימוש פרקטי ברשימת אנשי הקשר שנוצרה בסלולרי האישי של המשתמש.

**מחלקות המעורבות בפיצ'ר**:

* AppForm
* AppFacade
* GoogleContacts (המכיל תתי מחלקותEmail, Organization, Websites, Address, Phone )

### תבנית מס' 1 –Strategy

* **סיבת הבחירה / שימוש בתבנית:**

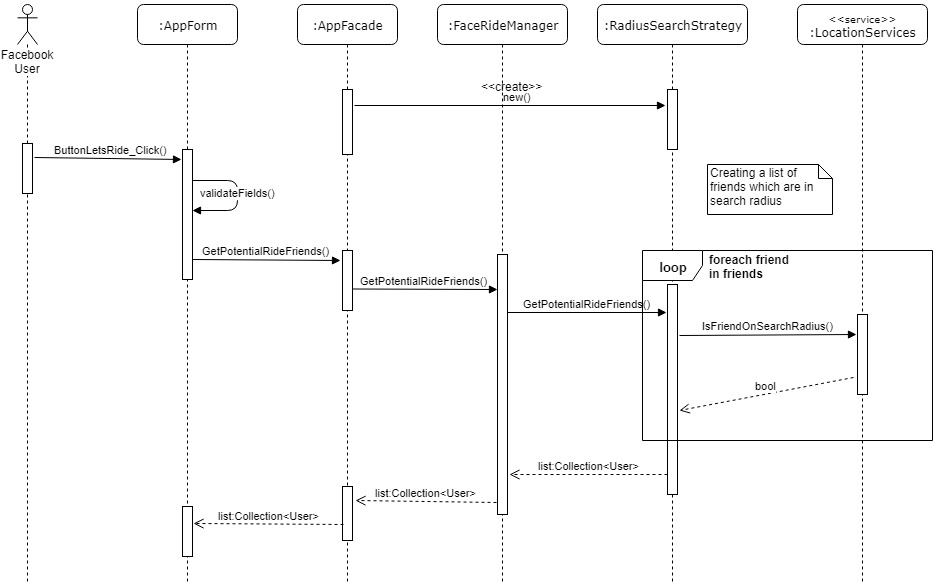
באפליקציה שלנו קיים פיצ'ר המאפשר איתור של חברים פוטנציאלים לנסיעה משותפת.

המערכת כרגע מקבלת רדיוס חיפוש רצוי מהמשתמש ועל פיו מאתרת את החברים האפשריים לנסיעה. אם בעתיד, נרצה להרחיב את הפיצ'ר ולאפשר לבנות את רשימת החברים הפוטנציאלים לנסיעה על פי חיפוש אחר שאינו קשור במרחק ו/או בשימוש ברדיוס חיפוש, נוכל לעשות זאת ללא שיכפול קוד או תחזוקה מיותרת של קוד קיים ע"י החלפת הרכיב SearchStrategy, שהוא Property אסטרטגי של המחלקה FaceRideManager.

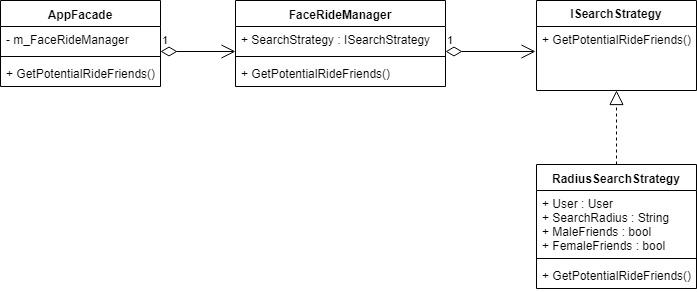
* **אופן המימוש:**

יצרנו Interface שנקרא ISearchStrategy הממוקם בתיקיית Interfaces תחת הפרוייקט Model. כאמור, המחלקה FaceRideManager מחזיקה בתור Property אובייקט אסטרטגיה מהסוג ISearchStrategy וכאשר נדרשת מה-Client לקבל את רשימת החברים האפשריים לנסיעה בטרמפ, היא מפעילה את אסטרטגית החיפוש בה בחר ה-Client מבעוד מועד, באופן המאפשר שינוי אסטרטגיה בזמן ריצה.

* **:Sequence Diagram**



* **Class Diagram:**



### תבנית מס' 2 – Iterator

* **סיבת הבחירה / שימוש בתבנית:**

מחלקת AlbumsManager אשר עוזרת לנו להתנהל מול אלבומי התמונות של המשתמש, מחזיקה כל הזמן רפרנס לאלבום הנוכחי עליו אנו רוצים לעבוד. כרגע אנו מחזיקים את הקישורים לתמונות באלבום ברשימה מקושרת, אך יכול להיות שבעתיד נרצה להוסיף פיצרים אשר קשורים במיון/סינון של תמונות או איחודם עם אלבומים אחרים. הגיוני לחשוב שבמידה ונרצה להוסיף פיצרים אלו, יהיה עלינו להחליף את מבנה הנתונים ואולי אפילו לצור מבנה נתונים במיוחד לצרכינו. לכן אנו רוצים לתת אפשרות לקליינט (הUI) לעבור באיטרציות על מבנה הנתונים בצורה שקופה שתאפשר לנו בעתיד לשנות את מבנה הנתונים מבלי לשנות את הקוד בUI. בנוסף הוספנו מיני פיצר של הצגת אלבום תמונות כמצגת רצה, כלומר יש חלון שבו מוצגת תמונה מהאלבום, וכל כמה שניות מתחלפת התמונה. לצורך כך עברנו על כל האלבום תוך שימוש בפטרן. //in View.SlideForm.cs

* **אופן המימוש:**

לצורך המימוש נעזרנו בפיצרים והכלים שC# נותנת לנו לטובת פטרן זה

internal class AlbumsManager : IEnumerable //in Model/AlbumsManager.cs

המחלקה AlbumsManager מממשת את האינטרפייס שקיבלנו מC#

public IEnumerator GetEnumerator() {foreach(string url in CurrentAlbumPhotosURL) yield return url; }

וכך המחלקה מממשת את הממשק – בעזרת yiled return שקיבלנו מC# וחוסך מאיתנו את הצורך בלעשות עוד מחלקות מקוננות.

לצורך השימוש בכך הוספנו עוד מחלקה קטנה לטובת הAppFacade

//in Model/AppFacade.cs

public class AlbumFacade : IEnumerable

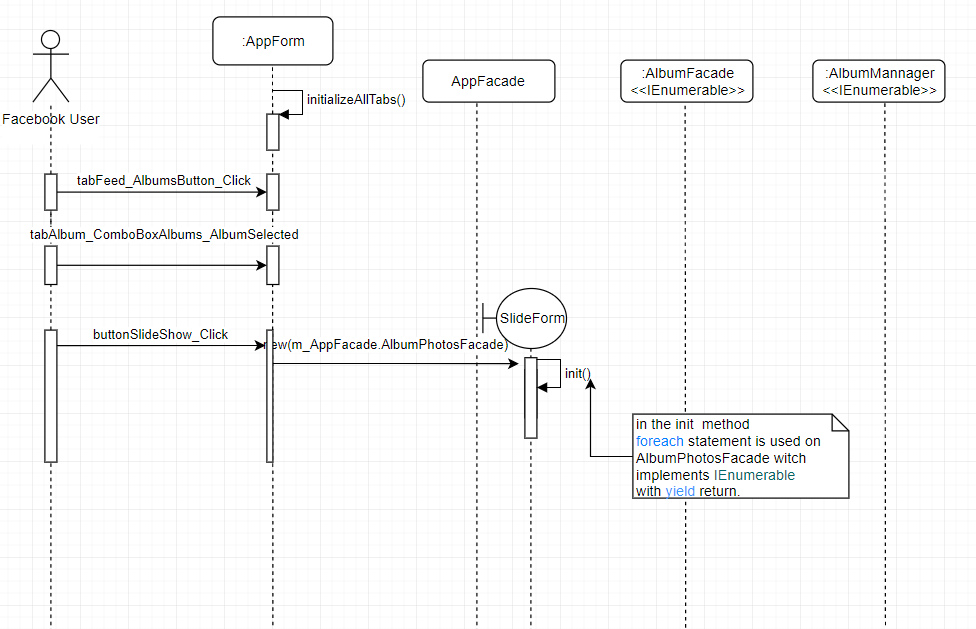
עשינו זאת כדי לחשוף למשתמש את האפשרות לעשות איטרציה על האלבום מבלי לחשוף לו דברים אחרים, בשביל בטיחות שימוש. מופע של מחלקה זאת נמצא ב AppFacade

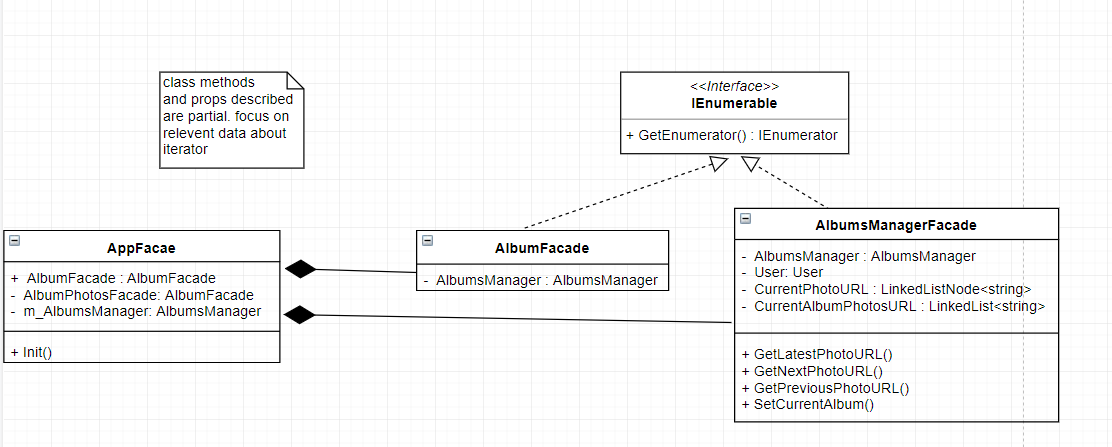
public AlbumFacade AlbumPhotosFacade { get; }

והשימוש בפטרן בסופו של דבר יהיה על ידי

foreach (string url in m\_AlbumFacade)

* Sequence Diagram



* Class Diagram 

### תבנית מס' 3 – Observer

* **סיבת הבחירה / שימוש בתבנית:**

הצורך לידע חלקים שונים בתוכנית על אירועים שהתרחשו. אלמנט חשוב היום בעולם האפליקציות זה מערך Notifications, היינו רוצים להתריע על שינויים ולידע חלקים בתוכנית על אירועים שהתרחשו. לדוגמא – כאשר מתווסף פוסט חדש, אנו רוצים לידע את המשתמש שהתווסף פוסט חדש – כך יוכל היוזר לדעת בזמן אמת על שינויים במידע ולפעול בהתאם.

בעתיד יכול להיות שיהיו חלקים בתוכנית שירצו לפעול בהתאם, לדוגמא אם יש כמה פוסטים חדשים, אז אולי נרצה לנתח מה הפוסט הכי רלוונטי מבין כל החדשים ואז להציגו ליוזר. אם יש פוסט חדש אז אולי נרצה לבדוק אם יש לו לייקים, ואם כן אז לשלוח הודעה לאנשים שעשו לייקים.

* **אופן המימוש:**

C# עוזרים לנו גם בפטרן זה בעזרת delegate וevent

מה שעשינו זה הוספנו פיצר אשר נמצא בתוך הAppFacade. תפקיד הפיצר הוא לבדוק אחת לדקה האם יש פוסט חדש – אם כן, אנחנו נודיע לכל מי שמתעניין בכך. במקרה שלנו הקליינט (הUI) מתעניין בכך – מה שהקליינט רוצה לעשות אצלנו זה להציג למשתמש MessageBox ובו כתוב שיש X פוסטים חדשים.

in Model/AppFacade.cs

public event Action<int> m\_ObserversForNewPost;

הצהרנו על אירוע שמודיע לאנשים מה קורה כאשר יש פוסט חדש

private Timer m\_TimerCheckUpdates;

לצורך הממימוש הוספנו טיימר שאחת לדקה מפעיל את המתודה הבאה:

private void OnTimedEvent(object source, ElapsedEventArgs e)

מתודה זאת בודקת אם יש פוסט חדש – במידה והתשובה כן אז היא עושה

m\_ObserversForNewPost.Invoke(dif);

אצל הקליינט ב View/AppForm.cs אנחנו בוחרים להאזין לאירוע הנ"ל

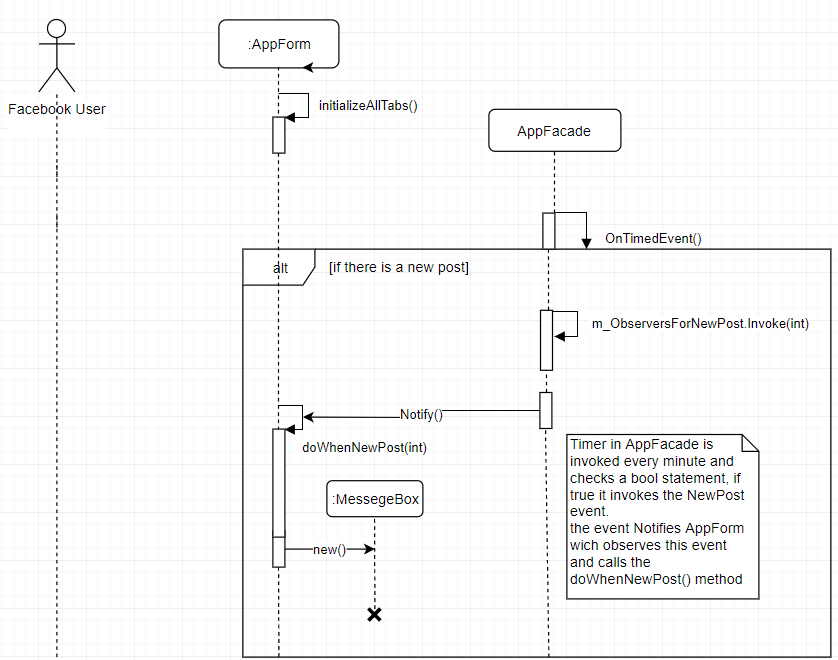
m\_AppFacade.m\_ObserversForNewPost += doWhenNewPost;

בעזרת המתודה

private void doWhenNewPost(int i\_NumOfNewPosts)

שמציגה את ההודעה שאנו רוצים למשתמש.

* Sequence Diagram



* Class Diagram

