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

* הפיצ'ר מאפשר למשתמש לחפש מקומות בילוי משותפים לחברים שייבחר.

המשתמש יוכל לבחור מתוך רשימת החברים שלו קבוצת חברים. המתשמש יוכל לבחור על ידי חיפוש מתקדם לפי שם המשתמש של החבר. כמו כן, המשתמש יוכל לטעון לאפליקציה את הרשימה האחרונה אשר הוא יצר. לאחר בחירת כל החברים אשר בהם הוא מעוניין, המשתמש יוכל לבחור למצוא את כל המקומות המשותפים להם(המיוצגים על ידי עמודים). האפליקציה תציג רשימה ממוינת של כל המקומות שכל החברים הנבחרים אהבו וגם תציג כמות מספרית של כמה חברים אהבו את המקום.

ניתן למצוא את הקוד תחת קובץ FindPlaceForm.

* הפיצ'ר מאפשר למשתמש במערכת לקחת חלק בקבוצות נסיעה משותפות (carpool) בין ערים שונות על סמך רשימת החברים שלו. כל קבוצת נסיעה מכילה בתוכה אוסף של נסיעות משותפות בין הערים.

המשתמש יכול ליצור קבוצה חדשה או להצטרף לקבוצה קיימת על פי בחירתו. כמו כן, המשתמש יכול להצטרף לאחת מן הנסיעות הקיימות בקבוצות אליהן הוא שייך, ובמידה ולא קיימת נסיעה כזו הוא יוכל ליצור בקשת נסיעה חדשה. בעת יצירת הנסיעה, המשתמש יכול לבחור אם לקחת חלק בנסיעה זו כנהג או כנוסע מן המניין וגם למלא את פרטי הנסיעה בשביל שאר חברי הקבוצה (מאיפה לאיפה הנסיעה, זמנים וכו').

ניתן למצוא את הקוד תחת קובץ PoolMyRideForm.

### תבנית מס' 1 –singleton

* סיבת הבחירה / שימוש בתבנית:

נרצה שבמערכת שלנו יהיה מנהל אחד ויחיד של המידע על היוזר. המידע של היוזר צריך להיות יחיד כי אם יהיו לנו 2 מנהלים כאלו אחד מהשניים יכול לקרות:

1. יכול להווצר מצב שבו מנהל אחד מחזיק מידע על משתמש פייסבוק מסויים, ומנהל שני יחזיק מידע על משתמש פייסבוק אחר. זה מצב לא תקין במערכת שלנו, שכן היא פועלת מול משתמש יחיד ברגע נתון. מצב כזה יגרום לבאגים במידע (המשתמש יבקש את רשימת החברים ויקבל רשימה של משתמש האחר).
2. מידע כפול נשמר במערכת (לדוגמא מנהל אחד שומר את רשימת החברים ואז במקום אחר נוצר עוד מנהל אשר יחזיק גם ברשימת החברים. לכן, נרצה רק מנהל אחד וכל המערכת תפנה אליו – שיתוף מידע.

* אופן המימוש:

החלטנו לממש את הsingleton בצורה גנרית כפי שלמדנו בשיעור. יצרנו מחלקה גנרית אשר T הוא תמיד מסוג class. הקונסטרקטור הוא STATIC ולכן פרטי. יצרנו property שנקרא INSTANCE אשר יוצר את הsingleton עצמו. המימוש נעשה כthread safe למקרה שנקרא לinstance ב2 threadים שונים. הmember הסטטי שמחזיק את הinstance הוא volatile בשביל מימוש הthread safe.

את הקוד הגנרי ניתן למצוא בתיקיית UTILS תחת הקוד Singleton.

המחלקה אשר ממומשת כסינגלטון נמצאת בקוד של UserDataManager שמשתמש בסינגלטון הגנרי.

* Sequence Diagram

A close up of a map

Description automatically generated

* Class Diagram

[class diagram שמתארת את המחלקות שלכם שמעורבות בתבנית (תיאור מלא שכולל Properties ו- Methods) והיחסים ביניהם (תאור מלא עבור כל יחס כפי שלמדנו בכיתה). עבור כל מחלקה שלכם, כיתבו מי המקבילה שלה (אם יש כזו) בתבנית (Pattern) שבחרתם

A close up of a map

Description automatically generated

המקבילה בתבנית:

Singleton<T> - Signelton<UserDataManager>

### תבנית מס' 2 – Adapter

* סיבת הבחירה / שימוש בתבנית:

המערכת שלנו עובדת מול DB אשר יודע להחזיר את המידע שביקשנו כאובייקט של XML וגם לקבל אובייקט XML ולשמור אותו. הUI שלנו עובד עם אובייקטים מסוג List. לכן, מימשנו ADAPTER אשר יתאם בין הDB לUI. כלומר הוא זה שימיר את המידע הXMLי לLIST ולהפך.

מבחינתו הDB והUI הם legacy components ולא נרצה לשנות אותם. כמו כן, אם בעתיד נרצה להחליף את מימוש הDB שלנו, לא נרצה לשנות קוד בUI, אלא נרצה רק להחליף את הADAPTER.

* אופן המימוש:

יצרנו interface בשם IDBHandler אשר מגדיר אלו מתודות ניתן להפעיל על הadapter. הadapter ממש את הממשק הנ"ל וגם מכיל מתודות פרטיות לטובת המימוש של הממשק וגם מחזיק בקומפוזיציה את הBD המקורי. כמו כן, בכל מקום אשר נצטרך קריאה לDB, נשמור member מסוג IDBHandler בעזרת קומפוזיציה. הmember הנ"ל יחזיק את הadapter.  
בממימוש זה, אם בעתיד הDB יתחלף וכך גם הadapter איתו, לא נצטרך לשנות את הקוד שלנו.

את הקוד ניתן למצוא ב תיקיית DBHandler בקבצים הבאים: DBHandlerAdapter , IDBHandler.

* A close up of a map

  Description automatically generatedSequence Diagram (דגשים על הadapter מסומנים בכחול)
* A close up of text on a white background

  Description automatically generatedClass Diagram (דגשים על הadapter מסומנים בכחול)

המקבילה בתבנית:

Client – FindPlaceForm

IService – IDBHandler

Adapter – DBHandlerAdapter

Adoptee - DBHandler

### תבנית מס' 3 – Caching Proxy

* סיבת הבחירה / שימוש בתבנית:

במערכת שלנו אנו זקוקים למידע מהשרת של פייסבוק המכיל נתונים של הuser. לעיתים, אנו נדרשים בזמן קצר להשתמש במידע אשר כבר קראנו לו בעבר. על מנת לחסוך קריאות יקרות לרשת, אנו מבצעים caching ברמת הזיכרון עבור כל מידע אותו אנו מבקשים בהשרת של פייסבוק. כלומר, בפעם הראשונה שנבקש מידע מסויים, נפנה לProxy אשר יודע שאין לו את המידע ולכן הוא יפנה לשרת. אך בפעם השנייה והלאה, כשנפנה ל Proxy הוא יודע שהמידע המבוקש כבר אצלנו שמור ולכן יחזיק אותו בלי לפנות לשרת. כתוצאה מכך, על כל מידע שנצטרך, תיהיה לנו פנייה אחת ויחידה לשרת. כך נחסוך משאבים רבים.

* אופן המימוש:

ה Proxy מחזיק propety של היוזר המחובר למערכת (האובייקט האמיתי). כשמעוניינים לבצע בקשה הדורשת מידע מפייסבוק, אנו משתמשים ב Proxy אשר במידה וזו הפעם הראשונה שביקשנו מידע זה, ישתמש ביוזר המחובר בשביל לפנות לשרתי פייסבוק ולקבלת המידע. בכל פעם אחרת, הוא ישתמש במידע ששמר על הproprty המבוקש. כמו כן, במערכת שלנו קיימת האפשרות להתנתקות והתחברות מחדש מכל יוזר שנרצה. לכן, כאשר היוזר מתנתק, אנחנו מבצעים ריסטרט לכל הproprties ששומרים את המידע בעזרת ריפלקשן. אם בעתיד נרצה להוסיף עוד cache למידע נוסף, לא נצטרך לשנות את המתודת הריסטרט. מימשנו את ה Proxy כthread safe מכיוון שבקשות לשרתים בדרך כלל מתבצעות כבקשות אסינכרוניות ונרצה שגם ה Proxy יתמוך בזה.

את הקוד ניתן למצוא ב-UserDataManager וב- IUserDataManager.

* Sequence Diagram (דגשים בסגול)

A close up of text on a white background

Description automatically generated

* A picture containing text, map

  Description automatically generatedClass Diagram (דגשים בכחול)

המקבילה בתבנית:

Client – MainForm, PoolMyRideForm, FindPlacesForm

ISubject – IUserDataManager

RealSubject – User

Proxy – UserDataManger

### עבודה אסינכרונית

החלטנו להוסיף בMainForm שימוש בפעולות אסינכורוניות. כאשר מתחברים לאפליקציה שלנו, היא פונה לשרתי פייסבוק בשביל לקבל פרטים מסויימים המוצגים במסך הראשי של המשתמש.

כל פעולה כזאת היא יקרה (פניה לשרת) ולכן נרצה לבצע את כל הפניות כפניות אסינכרוניות. במתודה fetchUserDetails השתמשנו במתודה Invoke של הListBox המכיל את נתונים אלו. מתודה זו מקבלת מתודה המבצעת פניה לשרת ומריצה את הפניה בThread נפרד. כתוצאה מכך, טעינת המסך הראשי יותר מהירה עבור המשתמש. כל Thread מטפל בפניה מסויימת ולא תלוי באחר.

נשים לב כי הקריאה לInvoke משתמשת בסינגלטון r\_UserDataManager ולכן מימשנו אותו כthread safe וגם את הפניות לשרת.

### עבודה עם Data Binding

החלטנו להוסיף Data Binding ב FindPlaceFormאשר מחבר בין ה Items של הListBox בUI לבין האובייקט List<User> ששומר את החברים של היוזר. במסך זה קיימים 2 ListBoxים ולכן ביצענו פעמיים Data Binding.

רשימה אחת מייצגת את החברים שנבחרו על ידי המשתמש ורשימה שנייה מייצגת את החברים שלא נבחרו על ידי המשתמש. על ידי ה Data Binding הדו כיווני, כל פעם שהמשתמש מעביר חבר מרשימה לרשימה, החבר עובר גם בתצוגה וגם ברשימה.

לאחר שהמשתמש מסיים לבחור את החברים ועובר למציאת מקום משותף, אפליקציה מחשבת את המקומות המעודפים על החברים שנבחרו לפי הרשימה של החברים הנבחרים.