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

חישוב מסלול בעזרת Google Maps-

באפליקציה למשתמש ניתנת אופציה לחשב מסלול ב Google Maps המתחיל במיקום לבחירתו ומסתיים במיקום לבחירתו, כאשר ניתנת לו האפשרות לבחור מבין הצ'ק אינים הקודמים שלו ומהעיר שאותה סימן כHOME TOWN ,פיצ'ר ישמש את המשתמש לתכנון טיולים במיוחד למקומות שבהם היה בעבר וביצע צ'ק אין.

נעשה בצורה הבאה:

כנקודת התחלה ניתן להזין טקסט חופשי או לייבא דרך הסרבר של פייסבוק את ה”hometown” של אותו משתמש. השארת שדה ריק תחשב את המסלול מהמיקום הנוכחי. לאחר מכן לבחירת היעד יבחר המשתמש בין הזנת טקסט חופשי או בחירת ביקור חוזר באחד מה"check-ins" שביקר בהם בעבר בפייסבוק. לאחר בחירת מוצא ויעד יוכל לבחור את סוג המסלול: נהיגה, תחבורה, רכיבה או הליכה. לבסוף ילחץ על “Plan my trip” ויוכל לראות בחלון שיפתח את הצעות המסלול שבחר בגוגל מפות.

ניתן למצוא את הפיצ'ר בקוד במחלקת הממשק (MainForm) במתודה startTripButton\_Click ובהמשך במחלקת AppLogic במחלקה GetTripUrl().

עריכת תמונת רקע:

באפליקציה למשתמש ניתנת אופציה לערוך ולשמור את תמונת הרקע שלו בצורה הבאה:

* בחירת תמונה הרקע שלו כרקע לאפליקציה
* "הרקדת" התמונה ע"י הזזה.
* סיבוב תמונה (בכל פעם התמונה תסתובב 90 מעלות ימינה).
* הפיכת התמונה מצבעונית לשחור לבן (תרחיש הsequence diagram)

כאשר יוכל המשתמש לשמור את התמונה על מכשירו האישי. אם למשתמש אין תמונת רקע בפייסבוק תוצג תמונת הפרופיל שלו והפיצ'ר יתנהל כרגיל.

ניתן למצוא את הפיצ'ר בקוד במחלקת הממשק (MainForm) במתודה activatePhotoButton\_Click ובהמשך במחלקת AppLogic (ע"מ לערוך את התמונה) במחלקות RotateImage() ו - TurnImageBlackAndWhite().

### תבנית מס' 1 – Strategy

* סיבת הבחירה / שימוש בתבנית:

התבנית Strategy היא תבנית מסוג Behavioral. בקוד שלנו ראינו שימוש במספר פעולות שמתבצעות בצורה מאוד דומה, הבאת מידע משרתי הפייסבוק. שמנו לב שמדובר בשכפול קוד, שפוגע גם באפשרות לre-use וגם בmaintainability של הקוד, כאשר אם נרצה להוסיף עוד פעולות נצטרך להוסיף שורות קוד רבות במספר מקומות שונים בקוד. תבנית Strategy פותרת בעיה זו, כאשר התבנית מאפשרת לשנות התנהגות אלג' מבלי לשנות את הקוד שלו, כך שיהיה ניתן לשחזר פעולות דומות ו"להזריק" לוגיקה רצויה לפי צורך המערכת.

בחרנו להשתמש במתודה זו על פני Template Method (שבעצם עוזרת לפתור את אותה בעיה בעזרת הורשה ולא בעזרת קומפוזיציה) ראשית כי אין לנו צורך ביתרונות הירושה, ובעיקר כי ירושה פחות נכונה קונספטואלית, שכן במציאות ההקבלה אינה מסתדרת, ולכן עבודה עם קומפוזיציה מרגישה יותר "טבעית".

* אופן המימוש:

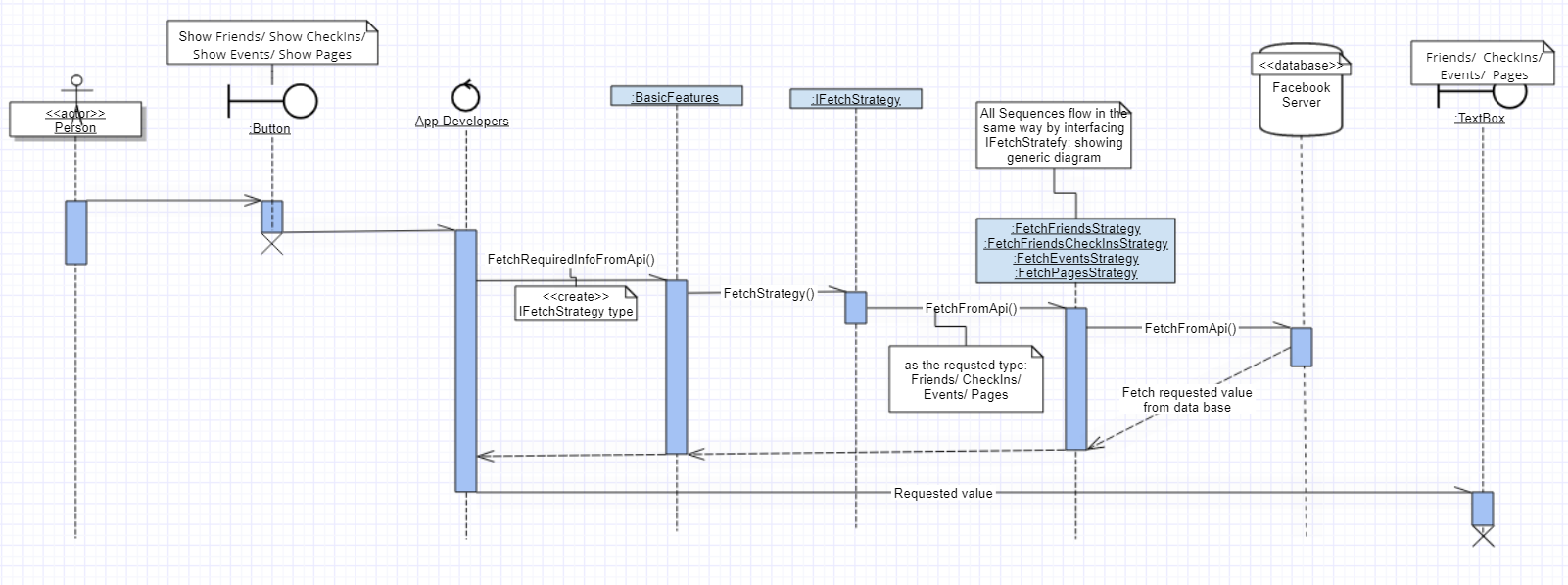
מימשנו Interface שמאלץ מימוש של מתודה האחראית להבאת מידע משרתי פייסבוק, מימשנו ארבע מחלקות שכל אחת ממשת באופן שונה את הממשק IFetchStrategy:

FetchEventsStrategy, FetchCheckInsStrategy ,FetchPagesStrategy ,FetchFriendsStrategy

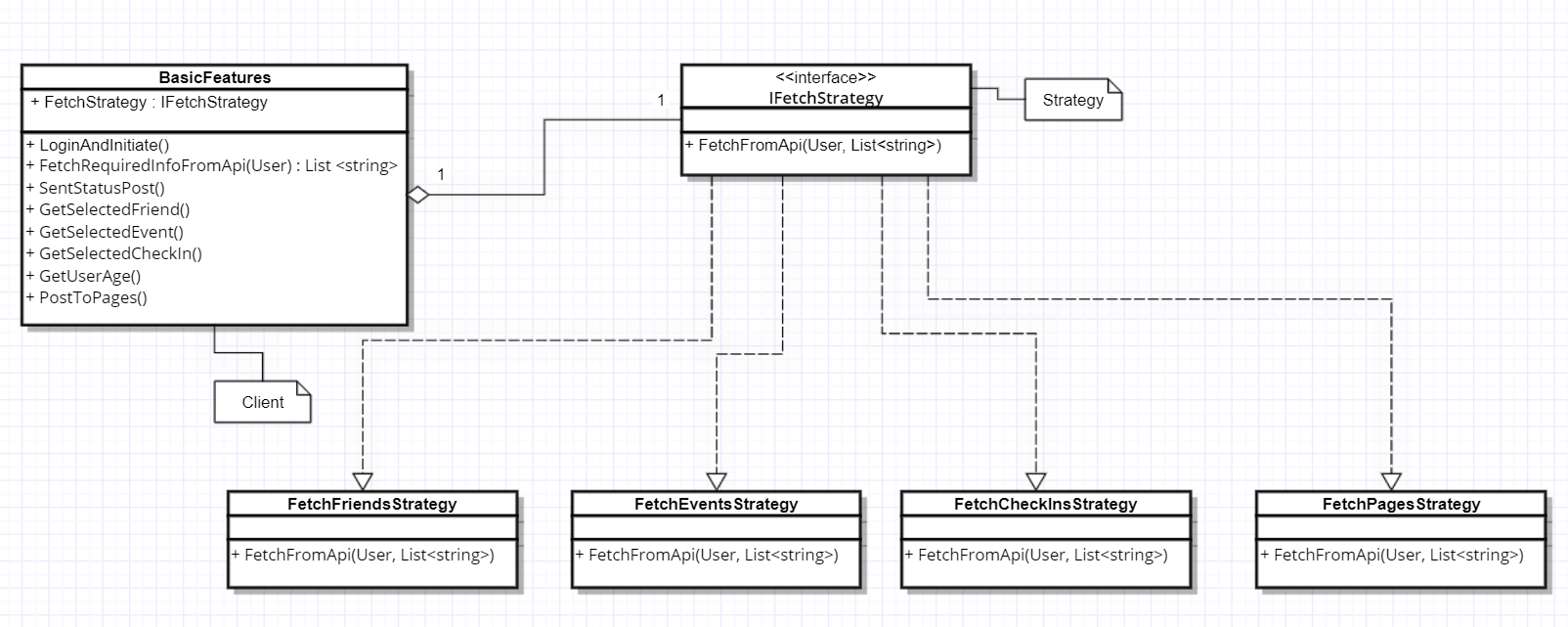
המחלקה AppLogic מקבלת אינדיקציה מממשק המשתמש על איזה סוג של מידע נדרש, לאחר מכן מייצרת את הממשק שמוחזק בקומפוזיציה אצל המחלקה BasicFeatures (שמופע שלה מוחזק בAppLogic בקומפוזיציה) בשורות (AppLogic.cs)34-56 ונותנת לו את הערך של ה-Strategy הרצוי.

לאחר מכן בתטודה FetchRequiredInfoFromApi בBasicFeatures בשורות (BasicFeatures.cs)34-41 מופעלת המתודה של ה Interface (כל מתודה במחלקה שאחראית לה) שכעת מאותחלת לפי דרישות ממשק המשתמש.

* Sequence Diagram



* Class Diagram



### תבנית מס' 2 – Observer

* סיבת הבחירה / שימוש בתבנית:

במערכת שלנו הפקדים אינם פעילים עד אשר מתבצעת התחברות לאפליקציה, ולכן יש פה תלות של רכיבים רבים (הפקדים) לבין טופס המערכת הראשי.

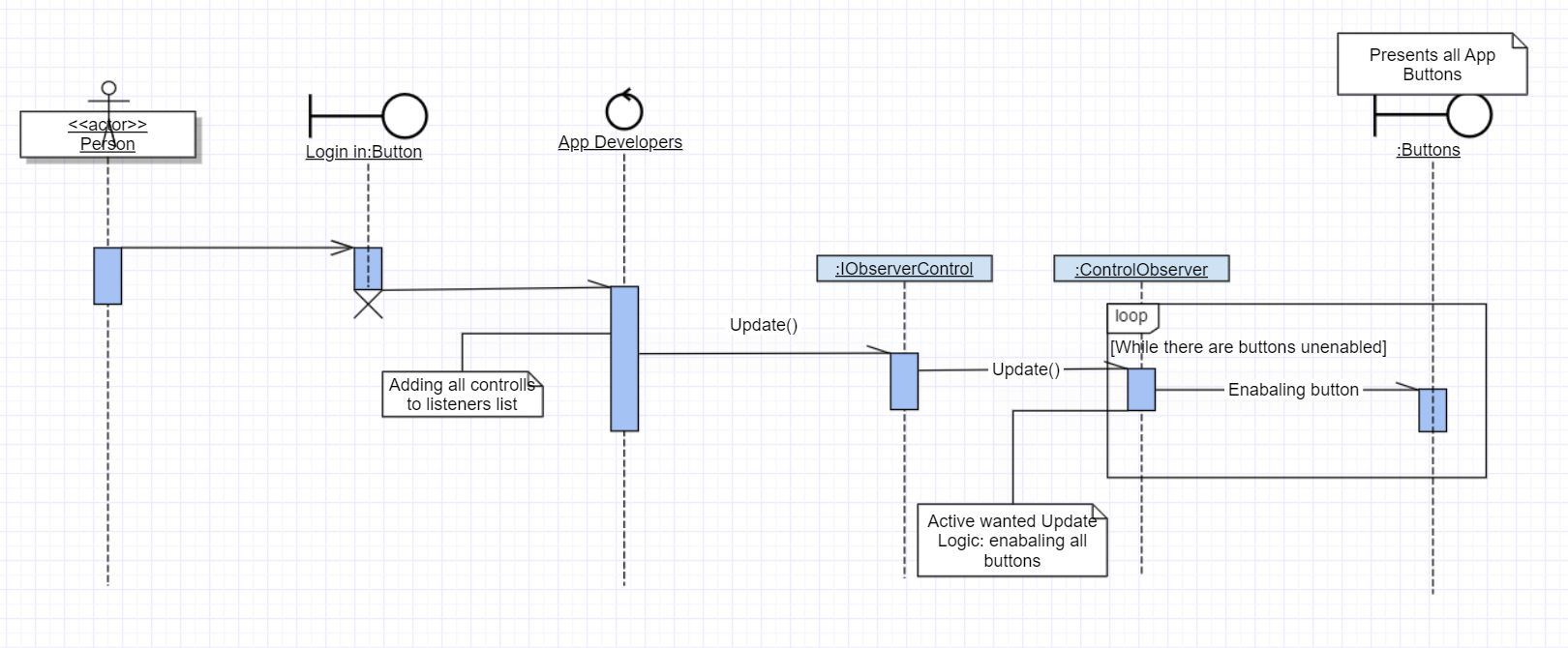
בנוסף, רצינו שכאשר יהיה שינוי בטופס אז כל הפקדים יתעדכנו בפקדים באופן אוטומטי, וגם לצורך הוספת פעולות עתידיות שרלוונטיות לכלל הפקדים.

תבנית Observer(תבנית מסוג behavioral) משמשת אותנו להגדרת הסובייקט (שהוא טופס האפליקציה) לבין המאזינים (שהם הפקדים) כך כאשר מתבצע שינוי בטופס הראשי שבעקבותו אנו רוצים לעדכן את הפקדים, ממש בצורת הפתרון שמציעה תבנית זו. בנוסף, Observer עוזר לנו בתחזוקה עתידית של כלל הפקדים, כאשר נוכל להגדיר פעולות בצורה תחזוקתית פשוטה בעתיד (כמו למשל להגדיל את כל הפקדים לשימוש לכבדי ראיה).

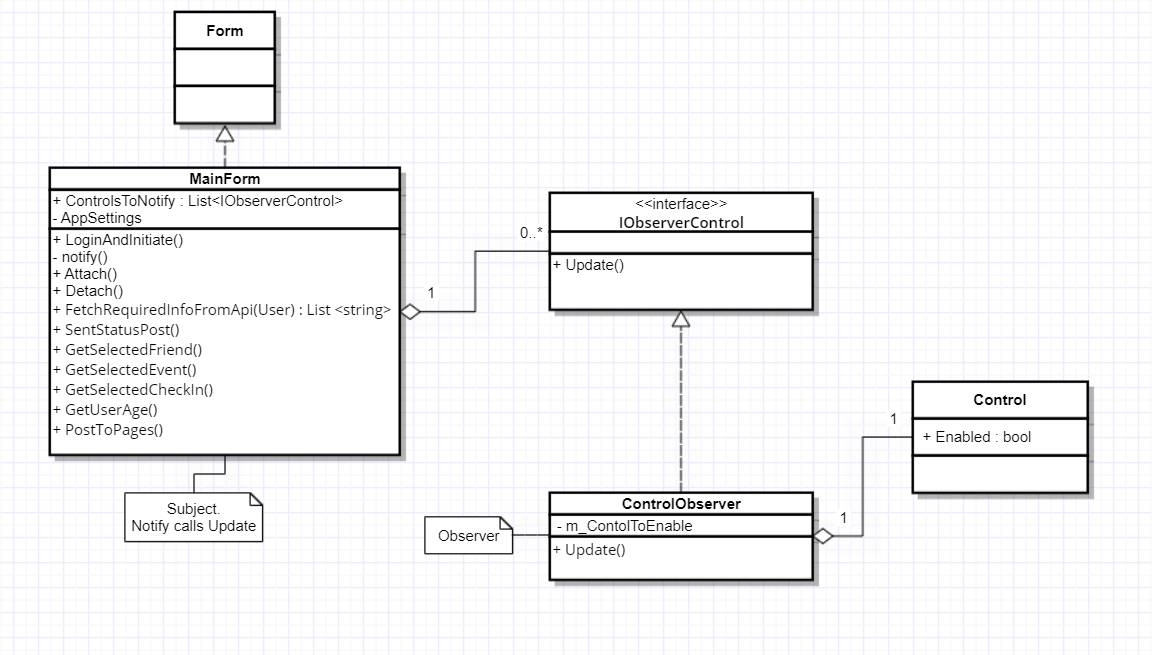
* אופן המימוש:

מימשנו interface (IObserverControl) שמחייב לממש את המתודה Update שבעצם אחראית על העדכון הרלוונטי לפקד (במקרה שלנו, לאפשר את הצגתו). המחלקה ControlObserver מחזיקה בקומפוזיציה פקד ומממשת את הממשק IObserverControl, וכמובן גם את המתודה Update (בשורות 14-17), בצורה שכאשר מפעילים אותה, היא מאפשרת את הצגתו של הפקד שמחזיקה. כאשר המשתמש מתחבר לאפליקציה, הפקדים מתווספים לרשימת המיודעים, וכאשר מתבצעת התחברות בהצלחה הטופס מודיע לכל המקשיבים (באמצעות המתודה notify) שיפעילו את המתודה Update של ControlObserver (ובעצם יאפשרו את הצגתם).

* Sequence Diagram



* Class Diagram



### תבנית מס' 3 – Template Method

* סיבת הבחירה / שימוש בתבנית:

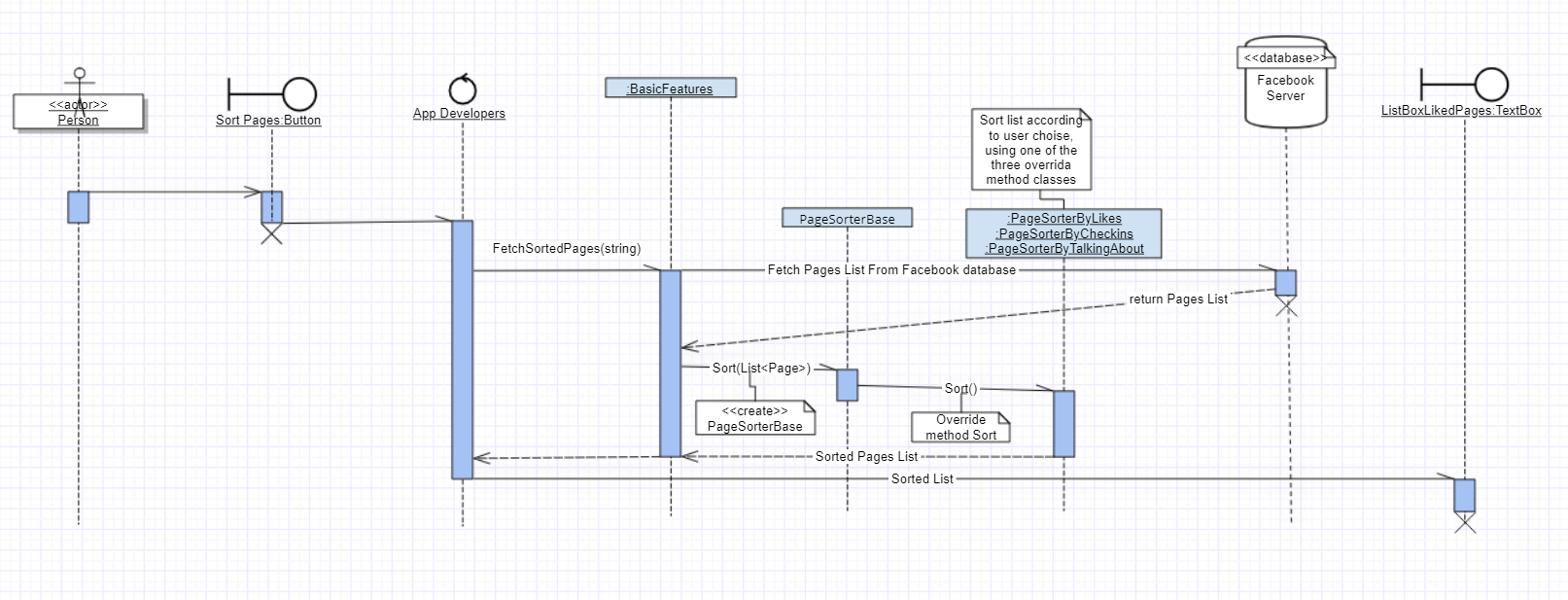
התבנית Template Method היא תבנית מסוג Behavioral. באפליקציה שלנו עלה הצורך למיין את רשימת הדפים האהובים לפי סדר מסוים ע"פ בחירת המשתמש: כמות לייקים, כמות צ'ק אינס ופופולריות הדף (Talking about count). שמנו לב שמדובר בשכפול קוד (כיוון שמלבד התנאי עצמו באלגוריתם המיון כל שאר הקוד זהה), וכמו כן ישנה גם פגיעה בmaintainability של הקוד, כאשר אם נרצה להוסיף עוד פעולות נצטרך להוסיף שורות קוד רבות במספר מקומות שונים בקוד. תבנית Template Method פותרת בעיה זו, כאשר מאפשרת לשנות התנהגות אלג' מבלי לשנות את הקוד שלו, כך שיהיה ניתן לשחזר פעולות דומות ו"להזריק" לוגיקה רצויה לפי צורך המערכת, ובמקרה שלנו – לשנות את תנאי המיון (שיהיו לי מספר לייקים או כמות צ'ק אינס למשל).

* אופן המימוש:

מימשנו מחלקה אבסטרקטית PageSorterBase עם מתודה אבסטרקית shouldSwap, שתהווה בעצם את נקודת "הזרקת" הלוגיקה. את המתודה הזו יממשו כל המחלקות האחראיות על מיון מסוים: (@) PageSorterByLikes, PageSorterByCheckins, PageSorterByTalkingAbout.

תהליך המיון יקרה לאחר שהמשתמש יבחר את סוג המיון – מיון דפים לפי כמות לייקים, כמות צ'ק אינים או לפי מספר המייצג כמות Talking About. לחיצה על Sort Pages, תוביל אותו למתודה sortPagesButton\_Click בשורות 565 – 598. המתודה תבדוק באיזה שיטת מיון בחר המשתמש ותשלח את הפרטים הרלוונטים למתודה FetchSortedPages(string) בחלק הלוגי של האפליקציה שתיצור את אחת משלוש המחלקות הנ"ל (@) לפי בחירת המשתמש, ותשלח את רשימת העמודים האהובים שיובאה משרתי הפייסבוק למחלקת Sort שבמחלקה שיצרה.

* Sequence Diagram



* Class Diagram

