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

**מיפוי חלוקת ערים לפי רשימת החברים:** פיצ'ר זה מציג את כל הערים בהם גרים החברים של המשתמש ואת כמות החברים שגרים בכל עיר.

למשתמש מוצג דיאגרמת פאי, כך שכל נתח מהדיגארמה מייצג עיר אחרת - דבר המאפשר למשתמש לנתח את פילוח הערים של החברים שלו בפייסבוק.

**הצגת כל האירועים של המשתמש בהתאם לתאריך שנבחר בלוח שנה:** פיצ'ר זה מציג למשתמש את כל האירועים אליו הוא רשום בהתאם לתאריך ספציפי שהוא בוחר.

למשתמש מוצג חלון בו הוא יכול לבחור תאריך עבורו ירצה לראות את כל האירועים ולאחר לחיצה על תאריך מסוים - יוצג לו ברשימה שמות כל האירועים שמתרחשים בתאריך שנבחר.

### תבנית מס' 1 – Singelton

* סיבת הבחירה / שימוש בתבנית:

במהלך פיתוח האפליקציה, הבחנו בכך שישנם מספר רכיבים שנדרשים לגשת לאותו הרכיב הלוגי היחיד שקיים במערכת שהינו - LogicManager.

ועל מנת לתת גישה לאותם הרכיבים הנ"ל ממקומות שונים במערכת למחלקה LogicManager, בחרנו להשתמש בסינגלטון.

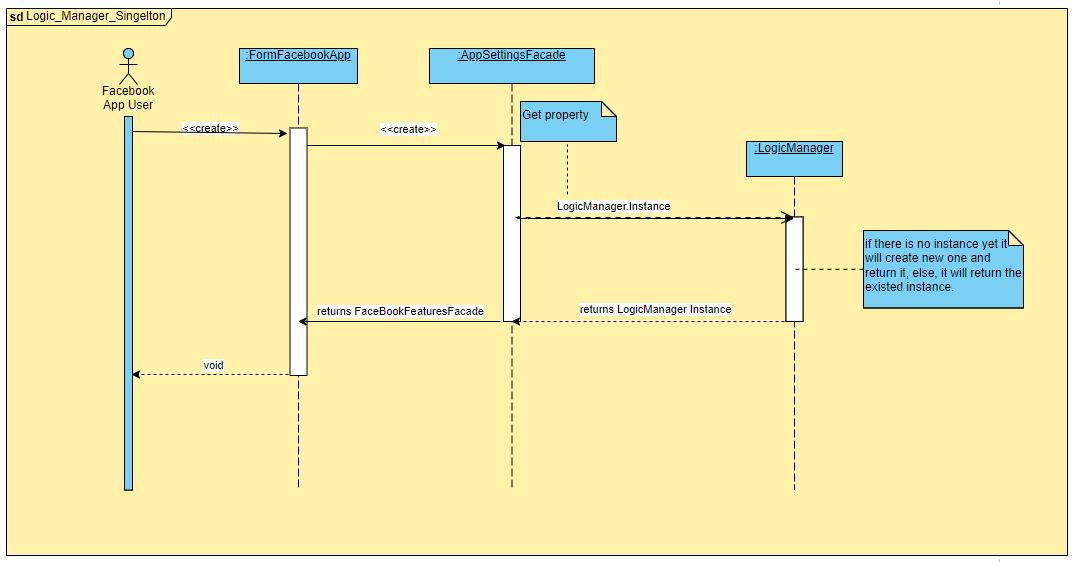
הדרך בה עשינו זאת היא הגבלת כמות המופעים של המחלקה למופע בודד. וגישה אליה באמצעות בקשה למופע שלה.

* אופן המימוש:

המחלקה LogicManager מומשה **כ- singelton**. המימוש נעשה כפי שהוצג בהרצאה, באמצעות הגדרת המחלקה כsealed על מנת למנוע אפשרות הורשה ממנה.

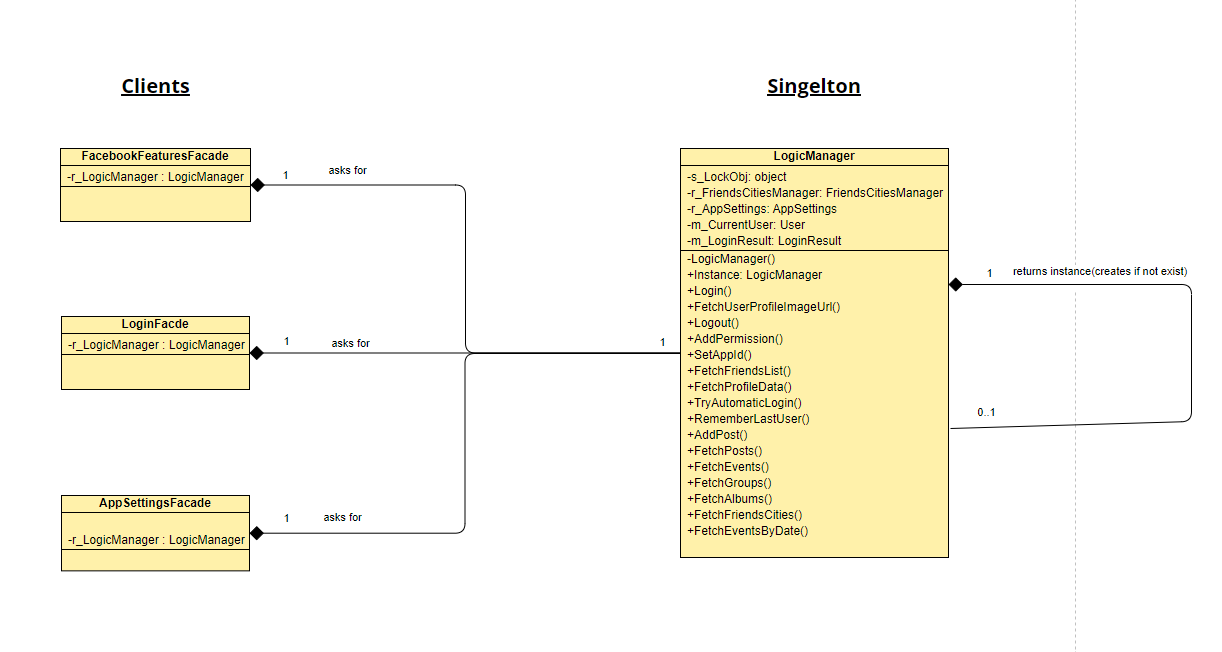
כמו כן הבנאי(constructor) מומש כprivate וזאת על מנת שלא ניתן יהיה ליצור יותר ממופע אחד של המחלקה במהלך ריצת התכנית. ולמעשה יצירת המחלקה תתבצע רק כאשר אחד הרכיבים יבקש Instance של המחלקה, בבקשה הראשונה למופע של LogicManager יופעל הבנאי (בקריאה ל Instance) וייוצר המופע היחיד של המחלקה, ולאחר מכן כל רכיב נוסף שיפנה למחלקה לצורך קבלת מופע, יקבל את אותו המופע שנוצר בפעם הראשונה.

* Sequence Diagrams



הערה: בדיאגרמה הנ"ל אנו מראים מקרה אחד של שימוש בסינגלטון במערכת שלנו אך ישנם 3 רכיבים שמשתמשים בסינגלטון הנ"ל והם מוצגים ב – class diagram.

* Class Diagram



### תבנית מס' 2 – Factory Method

* **סיבת הבחירה / שימוש בתבנית:**

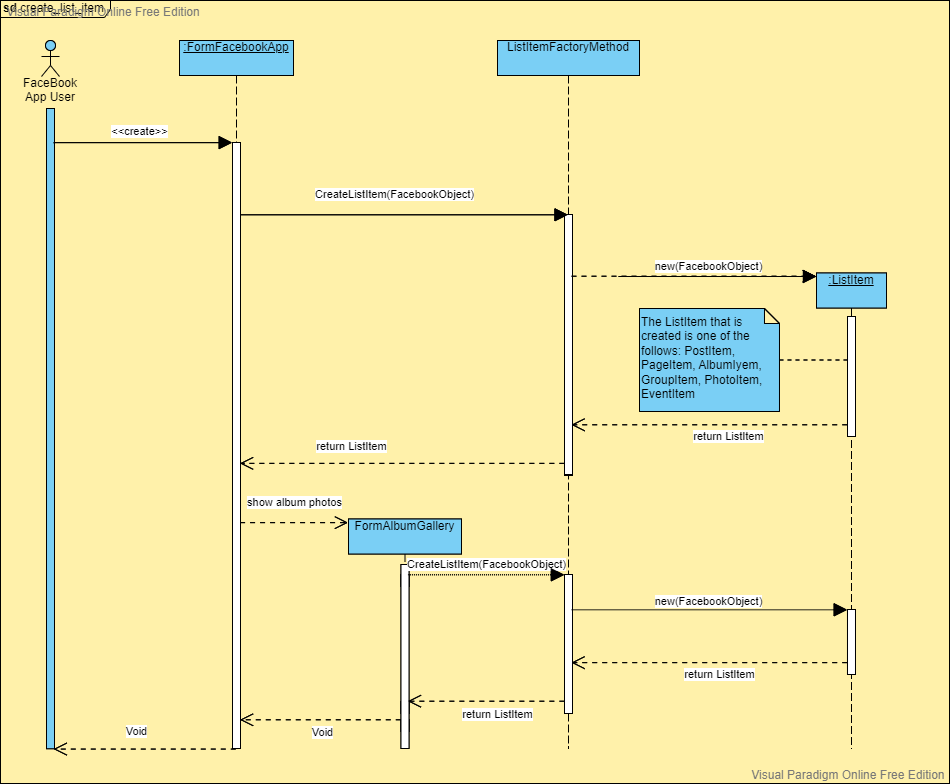
הסיבה לשמה בחרנו להשתמש בתבנית זו היא מכיוון שבאפליקצייה שלנו יש משפחה פולימורפית של רכיבי UI מסוג ListItem (PageItem, AlbumItem, PostItem ועוד.. ), ובמקום שרכיב ה UI יהיה זה שאחראי לייצר את את ה ListItem בהתאם לשימוש שלו, מי שיהיה אחראי לייצר אותו הינו המחלקה הסטטית ListItemFactoryMethod. היתרון לכך שהעברנו את האחריות ליצירת ה – ListItem למחלקה סטטית הינו בכך שבמידה ויהיו בעתיד רכיבי UI נוספים שרוצים גם ליצור רכיבי ListItem אז חסכנו שכפול קוד, מנענו מה UI לממש את היצירה בעצמו (בטיחות המשתמש), ובעתיד במידה ונרצה להכניס רכיב נוסף שהינו מסוג ListItem אזי התחזוקה של הקוד תיהיה רק במחלקה הסטטית במקום שינוי הקוד בכל רכיבי ה UI שמממשים בעצמם את היצירה של ListItem.

* **אופן המימוש:**

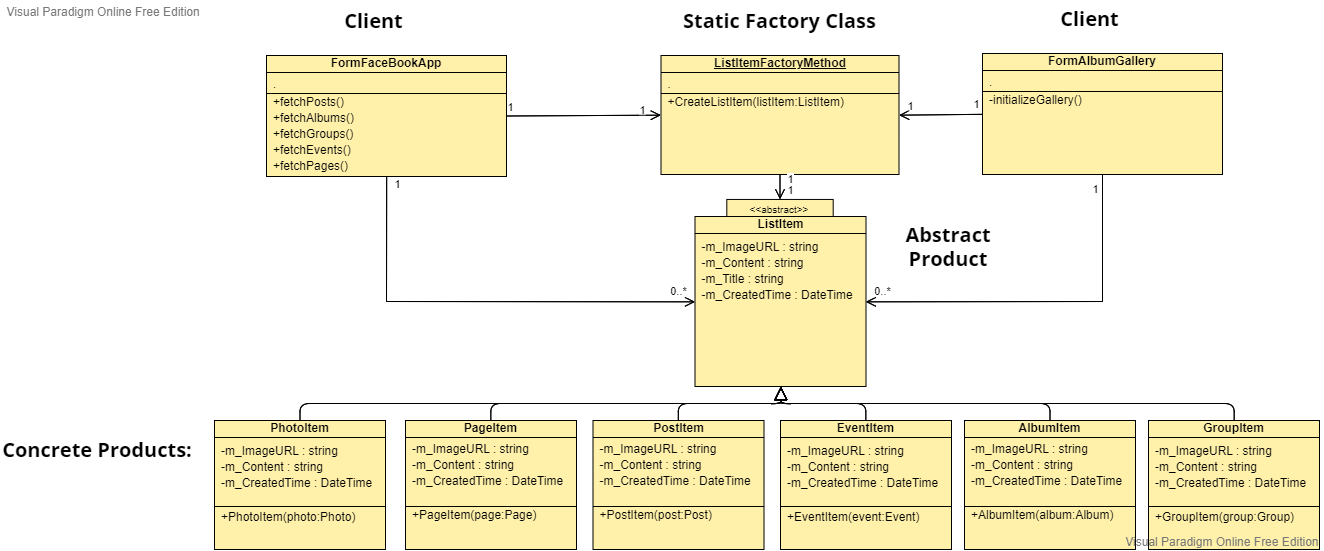
יצרנו את המחלקה הסטטית ListItemFactoryMethod שהיא משמשת בתפקיד ה- **static factory method** ובתוכה מתודה סטטית CreateListItem. על מתודה זו האחריות לייצר רכיב UI מסוג ListItem שמשמש תפקיד כ **Abstract product** בהתאם לפרטמר שנשלח אליה. הפרמטר שנשלח אליה הינו מסוג faceBookObject ובהתאם אליו היא מייצרת אובייקט מסוג של ListItem ומחזירה אותו למשתמש. ובצד הUI במחלקה FormFacebookApp בכל מקום בו נדרשנו לייצר רכיב מסוג ListItem, ישנה קריאה למתודה CreateListItem (למשל במתודת fetchPosts במקום לייצר PostItem נעזרנו במחלקת ListItemFactoryMethod ושלחנו כפרמטר את ה – post שהינו רכיב מסוג facebookObject ). שימוש נוסף בצד ה UI הינו בטופס FormAlbumGallery בו נדרשנו לייצר רכיב מסוג PhotoItem.

כך שכל רביכי ה – item למיניהם (PhotoItem, PostItem, AlbumItem, PageItem, EventItem, GroupItem ) משמשים תפקיד כ **Concrete Product** .

* **Sequence Diagram**

****

* **Class Diagram**



### תבנית מס' 3 – Facade

* סיבת הבחירה / שימוש בתבנית:

במהלך פיתוח האפליקציה הבחנו בכך שהשכבה הלוגית היא מאוד גדולה ועמוסה, ומכאן, מכיוון שלרכיבי הUI הייתה גישה לכלל המתודות של השכבה הלוגית, זה גרם למצב שבו לחלק מהרכיבים הייתה גישה להרבה מאוד מתודות שאינן רלוונטיות אליו.

לכן, הבחירה בFacade התבצעה על מנת שכל רכיב UI במערכת יעבוד אך ורק מול מתודות בשכבה הלוגית שרלוונטיות אליו, ולא יהיה חשוף למתודות שאין לו צורך בהן, כלומר מסתיר אותן מהרכיב, דבר שתורם להכמסה.

כמו כן גם מגביר את נוחות השימוש.

בנוסף כעת רכיבי הUI אינם מכירים את השכבה הלוגית, שכן יש את הFacade שמקשר ביניהם ואם בעתיד נרצה להוסיף רכיב UI חדש, כל שנצטרך לעשות הוא ליצור facade חדש, ויצריך שינוי מינימלי בלבד בקוד הקיים, ויתרה מכך אם בעתיד נרצה להחליף את השכבה הלוגית, הדבר לא יצריך שינוי בקוד של רכיבי הUI אלא רק בשכבה המקשרת, הלא היא הFacade .

* אופן המימוש:

כל רכיב UI שבעבר פנה לאובייקט LogicManager נדרש כעת לעבור דרך **Facade** כלשהו. על כן יצרנו 3 רכיבי Facade והם: LoginFacade, FacebookFeaturesFacade, AppSettingsFacade כך שכעת כל פניה של רכיב UI עבור הלוגיקה עובר דרכם בלבד.

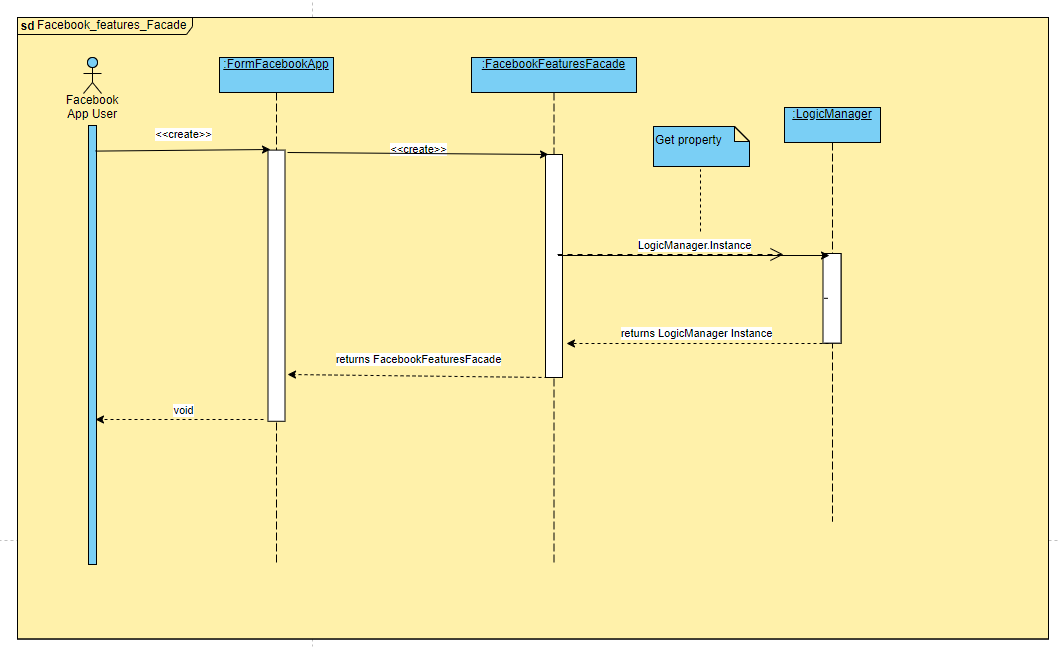
בצד ה – UI ישנם 4 רכיבים אשר משתמשים ב – Facdes באופן הבא: FormLogin משתמש ב LoginFacade עבור פעולות שנדרשות לרישום לאפליקצייה, UIManager משתמש גם הוא ב – LoginFacde עבור ניסיון להתחבר אוטומטית לאפליקצייה ( במידה ובוצע RememberMe בכניסה האחרונה למערכת).

בנוסף, רכיב האפליקצייה הראשית FormFacebookApp משתמש ב – FacebookFeaturesFacade עבור שימוש בפיצ'רים שונים שקיימים בלוגיקה,

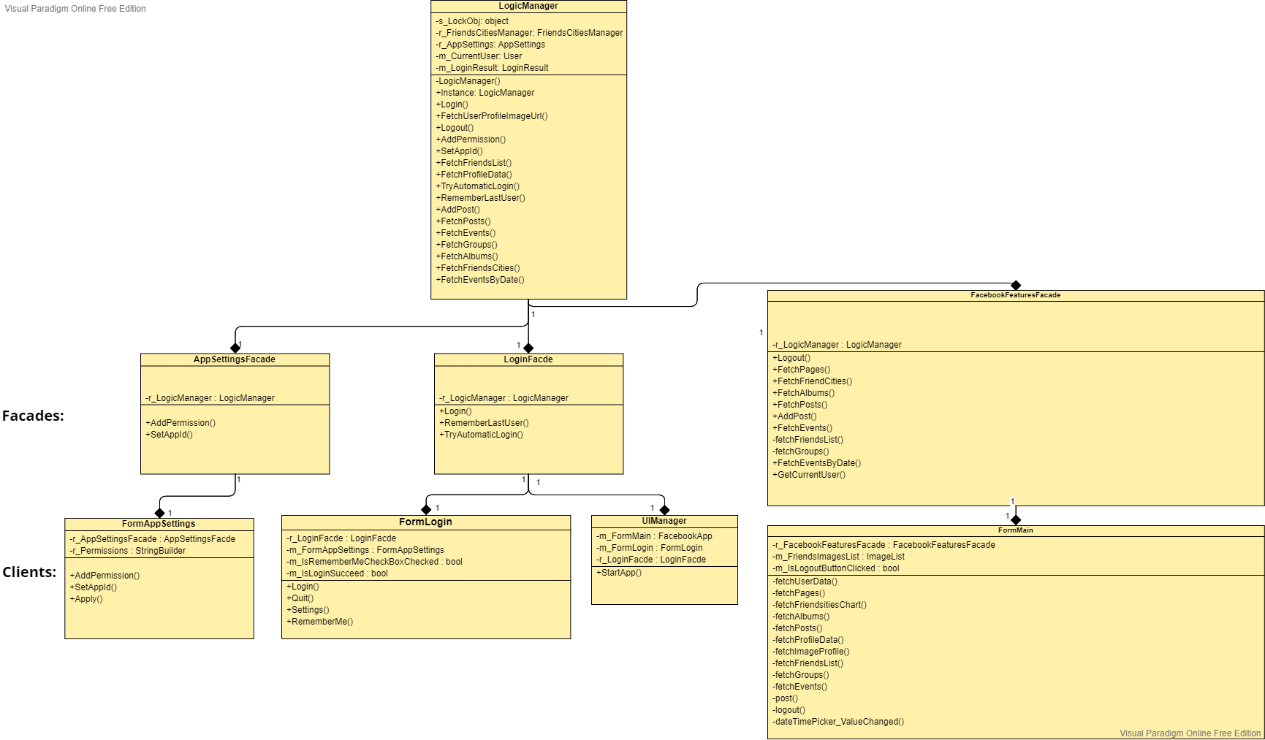
ורכיב ה- FormAppSettings משתמש ב AppSettingsFacade עבור מימוש פעולות בהגדרות האפליקצייה שקיימות בלוגיקה.

כך שלבסוף כל רכיב UI משתמש ביכולות הלוגיות אליו הוא נדרש בלבד, מבלי להכיר את כל המימוש הלוגי, ובו יש יכולות שלא רלוונטיות לו.

* Sequence Diagram



הערה: בדיאגרמה הנ"ל אנו מראים מקרה אחד של שימוש ב - facade במערכת שלנו אך ישנם 3 רכיבים שממשים את ה- design pattern הנ"ל והם מוצגים ב – class diagram.

* Class Diagram 

### עבודה אסינכרונית עם ממש המשתמש:

באפליקצייה עבודנו בצורה אסינכרונית עם ממשק המשתמש בכל המקומות בהם האפליקציה מבצעת קריאות לשרתי פייסבוק אשר מטרתם להביא/לשלוח מידעים שונים עבור רכיב ה UI (למשל: פוסטים, תמונות, קבוצות, רשימת חברים ועוד..).  
קריאות אלו לוקחות זמן משום שמדובר בפניות לשרת חיצוני, ועד שהן לא מסתיימות האפליקציה תקועה.  
על מנת לשפר את חוויית המשתמש, העברנו את הטיפול בהבאת המידע לטרדים נפרדים. כך שבכל בקשה לשרת פייסבוק ישנו טרד (שאינו הטרד של ה UI ) שאחראי על ביצוע פעולה זו.  
בנוסף, עבור כל שינוי ברכיבי ה UI מהטרדים שיצרנו, השתמשנו במטודה Invoke הנמצאת על כל אחד מהפקדים, ובכך עדכון המידע מתבצע דרך הטרד שאחראי על ה UI.

**קוד:** במחלקה FormFacebookApp במתודה fetchUserData עבור כל שליפת נתונים משרת פייסבוק ישנו שימוש בעבודה אסינכרונית באמצעות טרד, למשל במתודה ניתן למצוא טרד עבור: fetchFriendsList, FetchPosts, fetchAlbums וכו..

שימוש נוסף הינו באותה מחלקה עבור פרסום פוסט שנמצא במתודה buttonPost\_click מכיוון שלאחר שהמתמש כותב פוסט נדרש לעדכן אותו בשרתי פייסבוק – פעולה שעלולה לקחת זמן רב.

בנוסף באותה מחלקה ניתן למצוא שימוש בטרדים עבור כל מתודת Refresh שמבצעת עדכון של הנתונים מהשרת, והם:

linkLabelRefreshPosts\_LinkClicked, linkLabelRefreshAlbums\_LinkClicked, linkLabelRefreshGroups\_LinkClicked, linkLabelRefreshEvents\_LinkClicked,

linkLabelRefreshChart\_LinkClicked, linkLabelPages\_LinkClicked, dateTimePicker\_ValueChanged

**עבודה עם Data Bindings**

באפליקצייה השתמשנו ב – data bindings עבור פרטי המשתמש אשר כוללים את הפרטים הבאים:

תמונת פרופיל, שם פרטי, שם משפחה, מייל, תאריך לידה, locale , אודות.

רכיבי ה UI הנל מקושרים ל data bindings שנקרא: userBindingSource, והקישור הנ"ל מתבצע במתודה fetchProfileData. וכך בכל שינוי שיהיה עבור פרטי המשתמש, רכיבי ה – UI הנ"ל יתעדכנו בהתאם.