### \*\*\* הערה: אנו מציינים שפניות לשרת של פייסבוק עבור שליפה של אלבומי תמונות אינו נתמך יותר. בכל מקום בקוד שיש שימוש של שליפת אלבומים אנו מציגים הודעה למסך שמעדכנת אותך (הבודק) שהפניה אינה נתמכת יותר אך עבדה בהגשה הראשונה. לאחר שתלחץ על אישור, האפלקיציה תמשיך לרוץ בצורה תקינה לחלוטין. גיא רונן ביקש שנציין את הערה הזאת לבודק, תודה.

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

* **פיצ'ר 1 -** **Travel Buddy:** הפיצ'ר מאפשר למשתמש למצוא שותף לטיולים סביב העולם או למצוא אנשים אשר היו במקומות שהוא מעוניין להגיע אליהם ותוך כדי ליצור איתם קשר ולקבל המלצות. ניתן למצוא את הפיצ'ר תחת התיקיה Features.
* **פיצ'ר 2 – Volunteer:** הפיצ'ר מאפשר למצוא מקומות להתנדב בהם, להוסיף התנדבויות ולמחוק אותם, כל זאת במטרה לעזור לקהילת החברים. ניתן למצוא את הפיצ'ר תחת התיקיה Features.

### שימוש ב- MultiThreading:

* **שימוש 1:**

**היכן:** המקום בו עשינו שימוש בתכנות אסינכרוני הוא במחלקה Controllers.

**כיצד:** כל Controller שנוצר (Friend, Page, Photo, Post, Profile, Status) כולל אכלוס המידע שלו נוצר באמצעות Thread נפרד.

ב-Constructor של Controllers אנו קוראים למתודה LaunchThreads אשר רצה בלולאה על כל סוגי ה-Controllers הקיימים בתוך ה-Enum (שמחזיק את סוגי ה-Controllers) ומפעילה את ה-Threads. בנוסף, אנחנו תמיד שומרים בתוך המשתנה activeThreads את כמות ה-Threads הפעילים שלנו בכדי לדעת שברגע שהמידע הוטען במלואו נוכל להעלים את בר ההתקדמות (ProgressBar).

**מדוע:** הסיבה שהשתמשנו ב-MultiThreading במקום זה היא כדי שנוכל לטעון את המידע במקביליות ובמהירות גבוהה ובכך לאפשר חווית משתמש רציפה.

* **שימוש 2:**

**היכן:** המקום בו עשינו שימוש בתכנות אסינכרוני הוא במחלקה FormMain.

**כיצד:** כל כפתור שנלחץ מאתחל Thread נפרד.

**מדוע:** הסיבה שהשתמשנו ב-MultiThreading במקום זה היא כדי שנוכל ליצור חווית משתמש רציפה מבלי שהאפליקציה תיתקע במעבר בין הפאנלים השונים.

### שימוש ב- DataBinding:

* **היכן:** המקום בו עשינו השמה ל-DataBinding הוא ב-ProfileController על ידי הפונקציה populateProfileData. והשימוש ב- DataBindingנמצא במתודה onUserNameChanged, בנוסף ה-UserFullName שנמצא ב-FormMain מחובר ל-DataBinding ומקשיב לשינויים.

**כיצד:** האופן בו מימשנו את ה- DataBindingהוא על ידי שימוש ב-2 TextBoxes שכאשר ה-Events שלהם Validated מתבצעת השמה לשם המלא של המשתמש (LoggedInUser.Name) באמצעות השם הפרטי ושם המשפחה שעודכנו ב-TextBoxes בתוך ה-ProfileController באמצעות ה-Two Way DataBinding, זאת אומרת שאנו לא לוקחים את הטקסט מה-TextBox, אלא העדכון מתבצע בצורה אוטומטית. בנוסף, השינוי של השם המלא מתעדכן בצורה אוטומטית ב-FormMain ליד התמונת פרופיל.

**מדוע:** הסיבה שהשתמשנו במקום זה ב-Two Way DataBinding היא כדי שלמשתמש תהיה אופציה לעדכן את פרטיו האישיים.

### תבנית מס' 1 – Singleton

* סיבת הבחירה / שימוש בתבנית:

אנו בחרנו להשתמש בתבנית זו ב-Class בשם SingletonFileOperations – מחלקה זאת אחראית על ניהול שמירת המידע של הפיצ'רים ב-DataBase (קבצי XML).

הסיבות שבחרנו להשתמש בתבנית זו הן:

1. נקודת שליטה אחת על ניהול שמירת הנתונים בקבצים. כל פעולות הקבצים מטופלות על ידי מופע אחד, מה שמקל על התחזוקה של המחלקה, זאת אומרת שאם יהיו תקלות במערכת שקשורות לשמירת הנתונים, נדע לפנות אליה ולטפל בתקלה. בנוסף, מפשטת את הניהול של גישה מקבילה לקבצים על ידי הסדרת גישה דרך מופע הסינגלטון.
2. נקודת גישה גלובלית - מספקת נקודת גישה גלובלית לפעולות הקבצים, מה שהופך אותו פשוט לקריאה/כתיבה מכל מקום באפליקציה מבלי להעביר הפניות. בנוסף, מונע יצירת מופעים מרובים, מה שעלול להוביל לפתיחת נקודות אחיזה מיותרות בקבצים. כמו כן, זה מבטיח לנו ששימוש שקשור לפעולות על קבצים (כמו נתיבי קבצים, פורמטים) יהיו עקביים בכל נקודות השימוש.
3. יישום פשוט - מפחית את המורכבות של הקוד, מכיוון שאין צורך לטפל בלוגיקה של פעילויות על קבצים במספר מקומות שונים. בנוסף, מאפשר מעקב אחר הנתונים שנכתבו או נקראו.

* אופן המימוש:

מחלקה Singleton<T>

תיאור:

מחלקת Singleton<T> היא יישום גנרי של תבנית העיצוב של Singleton. זה מבטיח שרק מופע אחד של המחלקה שצוינה (T) נוצר ומספק נקודת גישה גלובלית אליו. המחלקה משתמשת בנעילה עם בדיקה כפולה כדי להבטיח בטיחות של ה-Thread במהלך יצירת מופע – כלומר שלא ייווצר מצב ששני תהליכים ייכנסו לקטע הקוד שבו נוצר המופע ועקב כך ייווצר יותר ממופע אחד של המחלקה.

ניתן למצוא את המימוש של Singleton<T> תחת התיקייה של הפיצ'ר Volunteering.

מחלקה SingletonFileOperations

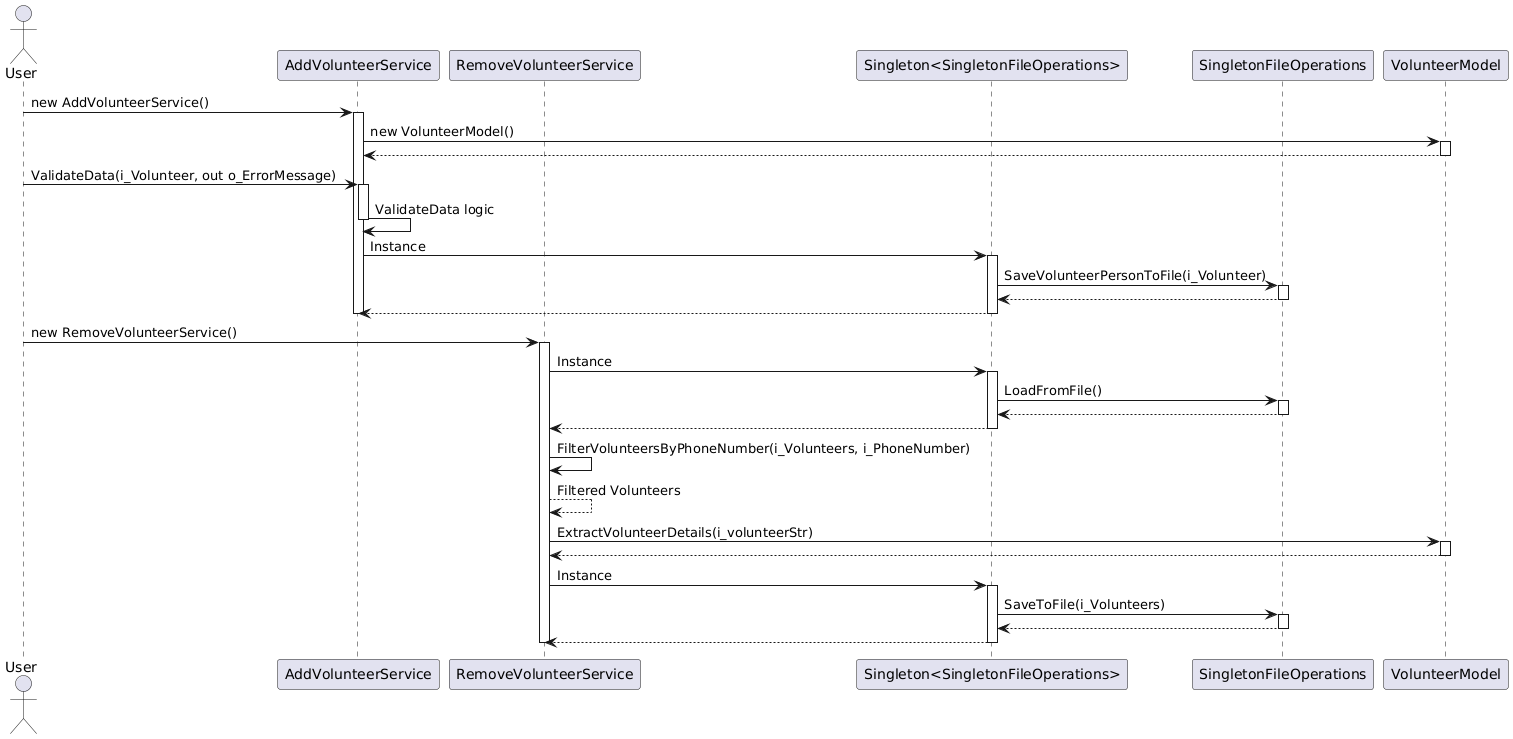
תיאור:

המחלקה SingletonFileOperations מטפלת בפעולות על קבצים הקשורות לנתוני מתנדבים, כגון שמירה וטעינת מידע על מתנדבים לקובץ XML וממנו. מחלקה זו עוקבת אחר דפוס Singleton כדי להבטיח שרק מופע אחד נוצר. הוא מספק שיטות לשמירת מתנדב בודד לקובץ, שמירת רשימת מתנדבים וטעינת מתנדבים מהקובץ.

ניתן למצוא את המימוש של SingletonFileOperations תחת התיקייה של הפיצ'ר Volunteering.

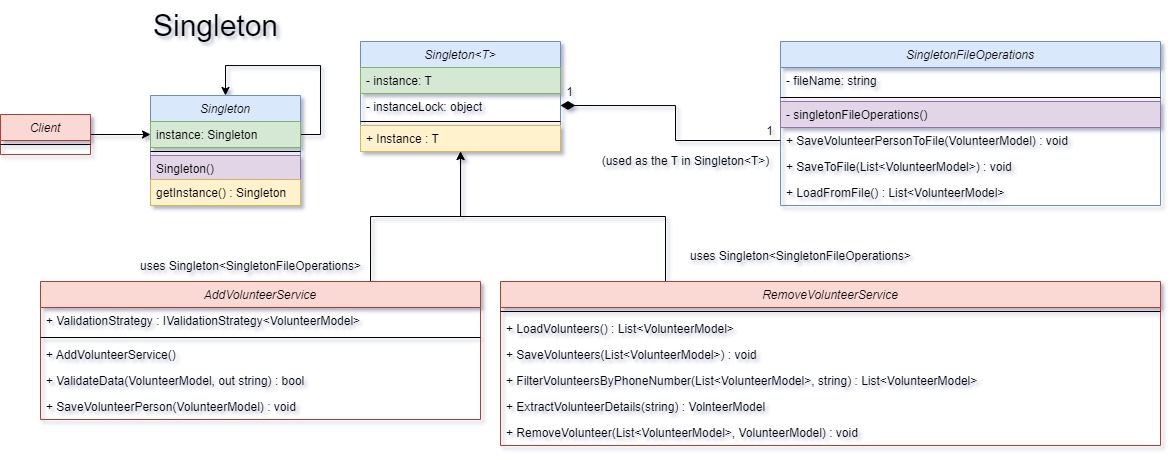
השימוש במחלקות אילו מתבצע בקוד במחלקות AddVolunteerService ו RemoveVolunteerService שנמצאות תחת התיקייה של הפיצ'ר Volunteering.

השימוש מתבצע על ידי קריאה למחלקה Singleton שהיא מחלקה סטטית ושימוש ב-T בתור SingletonFileOperations, על ידי כך מקבלים מופע יחיד של המחלקה SingletonFileOperations.

* Sequence Diagram
* Class Diagram

סימנו את ההקשר בין תבנית העיצוב שקשורה למחלקה שלנו לתבנית העיצוב הבסיסית בעזרת הצבעים השונים. הדיאגרמה הימנית היא הדיאגרמה שקשורה לקוד שלנו והדיאגרמה השמאלית היא הדיאגרמה הכללית של תבנית העיצוב Singleton.

זאת אומרת המחלקות המקבילות הן באותו הצבע והמתודות המקבילות גם הן באותו הצבע בין התבנית הכללית לבין הקוד שלנו.



### תבנית מס' 2 – Strategy

* סיבת הבחירה / שימוש בתבנית:

בחרנו להשתמש בתבנית העיצוב Strategy מכיוון שקיים אצלנו בקוד שימוש בולדיציה של המידע במחלקות שונות. המחלקות של ה-Service כוללות ולידציות שונות לנתונים שונים, אך עם זאת כל ולידציה תואמת לכמה מחלקות. כדי להמנע משכפול קוד תבנית Strategy נותנת מענה מיטבי בכך שמאפשרת להשתמש בצורה דומה בסוגי הולידציות השונות. כל אסטרטגיית אימות מתמקדת בסוג מסוים של אימות, מה שמקל על ניהול והרחבת כללי אימות באופן עצמאי. על ידי ניתוק כללי אימות מהשירותים המשתמשים בהם, הקוד הופך מודולרי יותר וקל יותר לתחזוקה. ניתן לעדכן או להחליף כל אסטרטגיה מבלי להשפיע על חלקים אחרים של האפליקציה.

מבנה תבנית העיצוב מאפשר שימוש חוזר בלוגיקת האימות בחלקים שונים של האפליקציה.

ניתן ליצור בקלות אסטרטגיות אימות שונות עבור סוגים שונים של נתונים או הקשרים, תוך הבטחה שכללי האימות מיושמים כראוי.

ככל שצצות דרישות אימות חדשות, ניתן להוסיף אסטרטגיות חדשות מבלי לשנות את הקוד הקיים. זה הופך את המערכת לניתנת להרחבה ולהתאמה לדרישות המתפתחות.

במילים פשוטות, בעזרת תבנית עיצוב זו מנענו שכפול קוד מיותר של ולידציה של נתונים זהים במחלקות שונות.

* אופן המימוש:

ממשק IValidation<T>:

הממשק ממומש באופן גנרי, כולל בתוכו מתודה של ולידציה שמקבלת מחלקה (T) ובעזרת ממשק משותף אנו יכולים לעבוד עם כל סוגי ה-Strategies בצורה אחידה.

ניתן למצוא את הממשק בקוד תחת תיקיית ValidationStrategy שנמצאת תחת תיקיית Features.

מחלקות VolunteerCategoryValidation, VolunteerDateValidation, VolunteerLocationValidation, VolunteerPhoneNumberValidation, נמצאות תחת התיקייה VolunteerValidations שנמצאת תחת התיקייה ValidationStrategy שהיא תחת תיקייה Features.

מחלקות TravelAgeRangeValidation, TravelCountryValidation, TravelDateValidation, TravelGenderValidation, נמצאות תחת התיקייה TravelBuddyValidations שנמצאת תחת התיקייה ValidationStrategy שהיא תחת תיקייה Features.

ממומשות בצורות שונות כך שכל מחלקה תתאים לתנאי הבדיקה הרלוונטיים אליה. אך עם זאת כולן מממשות את המתודה Validate של הממשק IValidation<T> וזה מה שמאפשר עבודה אחידה עם כל המחלקות ללא צורך בשכפול קוד.

מחלקת AddVolunteerService משתמשת בולידציות הרלוונטיות אליה - VolunteerCategoryValidation, VolunteerDateValidation, VolunteerLocationValidation, VolunteerPhoneNumberValidation. במידה והנתונים תקינים והבדיקה עוברת, המידע נשמר בקובץ, במידה ולא תזרק הודעת שגיאה עם הנתונים החסרים או לא תקינים.

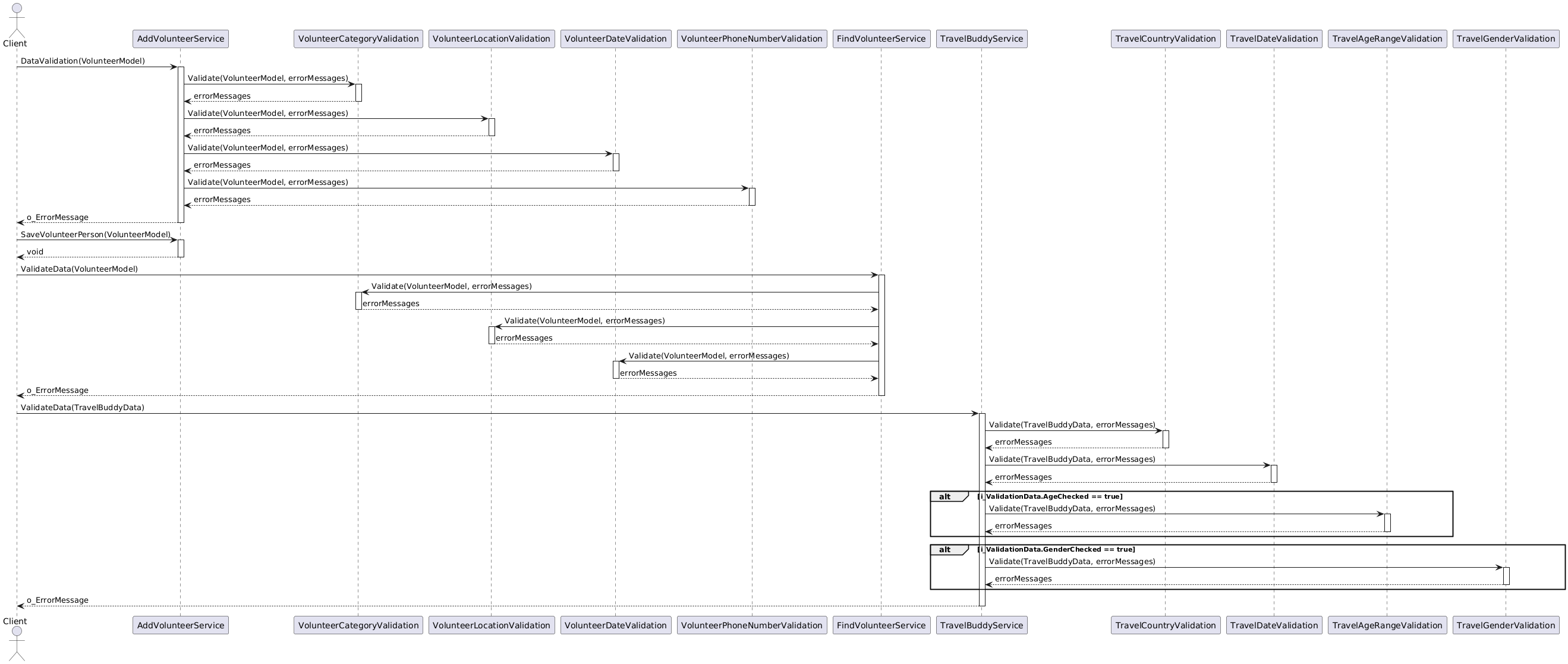
ניתן למצוא את המחלקה תחת התיקייה Services שנמצאת תחת התיקייה Volunteering שנמצאת תחת התיקייה Features.

מחלקת FindVolunteerService משתמשת בולידציות הרלוונטיות אליה - VolunteerCategoryValidation, VolunteerDateValidation, VolunteerLocationValidation. במידה והנתונים תקינים והבדיקה עוברת, המידע מוצג למשתמש, במידה ולא תזרק הודעת שגיאה עם הנתונים החסרים או לא תקינים.

ניתן למצוא את המחלקה תחת התיקייה Services שנמצאת תחת התיקייה Volunteering שנמצאת תחת התיקייה Features.

מחלקת TravelBuddyService משתמשת בולידציות הרלוונטיות אליה - TravelAgeRangeValidation, TravelCountryValidation, TravelDateValidation, TravelGenderValidation. במידה והנתונים תקינים והבדיקה עוברת, המידע מוצג למשתמש, במידה ולא תזרק הודעת שגיאה עם הנתונים החסרים או לא תקינים.

ניתן למצוא את המחלקה תחת התיקייה Services שנמצאת תחת התיקייה TravelBuddy שנמצאת תחת התיקייה Features.

* Sequence Diagram
* Class Diagram

תמונה שמכילה טקסט, צילום מסך, גופן, מספר

התיאור נוצר באופן אוטומטיסימנו את ההקשר בין תבנית העיצוב שקשורה למחלקה שלנו לתבנית העיצוב הבסיסית בעזרת הצבעים השונים. הדיאגרמה התחתונה היא הדיאגרמה שקשורה לקוד שלנו והדיאגרמה העליונה היא הדיאגרמה הכללית של תבנית העיצוב Strategy.  
זאת אומרת המחלקות המקבילות הן באותו הצבע והמתודות המקבילות גם הן באותו הצבע בין התבנית הכללית לבין הקוד שלנו.

### תבנית מס' 3 – Facade

* סיבת הבחירה / שימוש בתבנית:

בחרנו בתבנית עיצוב Facade מכיוון שיצרנו מספר Controllers אשר חולקים ביניהם את אותם המאפיינים ובמקום שנצטרך לתפעל ב-FormMain כל Controller בנפרד יצרנו מעטפת שדרכה אנו יכולים ליצור ולנהל את כל ה-Controllers שלנו מבלי שה-FormMain יצטרך להכיר אותם. בעקבות זאת, פישטנו את השימוש של היצירה ושל העבודה עם ה-Controllers השונים על ידי שימוש ב-Interface שאותו מממשים כל ה-Controllers השונים. כמו כן, תהליך היצירה של Controller חדש מתבצע בצורה פשוטה מכיוון שכתבנו את המחלקה Controllers בצורה גנרית. במילים אחרות, אנו לא חושפים למשתמש את כל הפאנלים השונים. אנו מביאים לו נקודת גישה אחת שדרכה הוא יכול לנהל את השימוש בפאנלים השונים.

* אופן המימוש:

מחלקה FormMain מייצגת את ה-Client ומשתמשת במחלקה Controllers (שהיא מממשת את הנקודת גישה של המשתמש לפאנלים השונים).

ניתן למצוא את המחלקה בקוד בקובץ הראשי וניתן למצוא את השימוש של המחלקה במחלקת ה-Controllers בתוך המתודות השונות אשר משמשות את ה-Client בהצגת הפאנלים השונים (buttonXXXX\_Click).

מחלקת Controllers מייצגת את הממשק שדרכו אנו מנהלים את כל הרכיבים (פאנלים) השונים. אנו מחזיקים במחלקה Controllers מבנה נתונים מסוג מילון אשר מחזיק בתוכו את כל סוגי הפאנלים השונים של המערכת ומייצר אותם ב-Threads שונים כדי ליצור חווית משתמש רציפה. הקוד נכתב בצורה גנרית, כאשר ה-Constructor של המחלקה מזהה באיזה פאנל מדובר ומפעיל את ה-Constructor של כל פאנל.

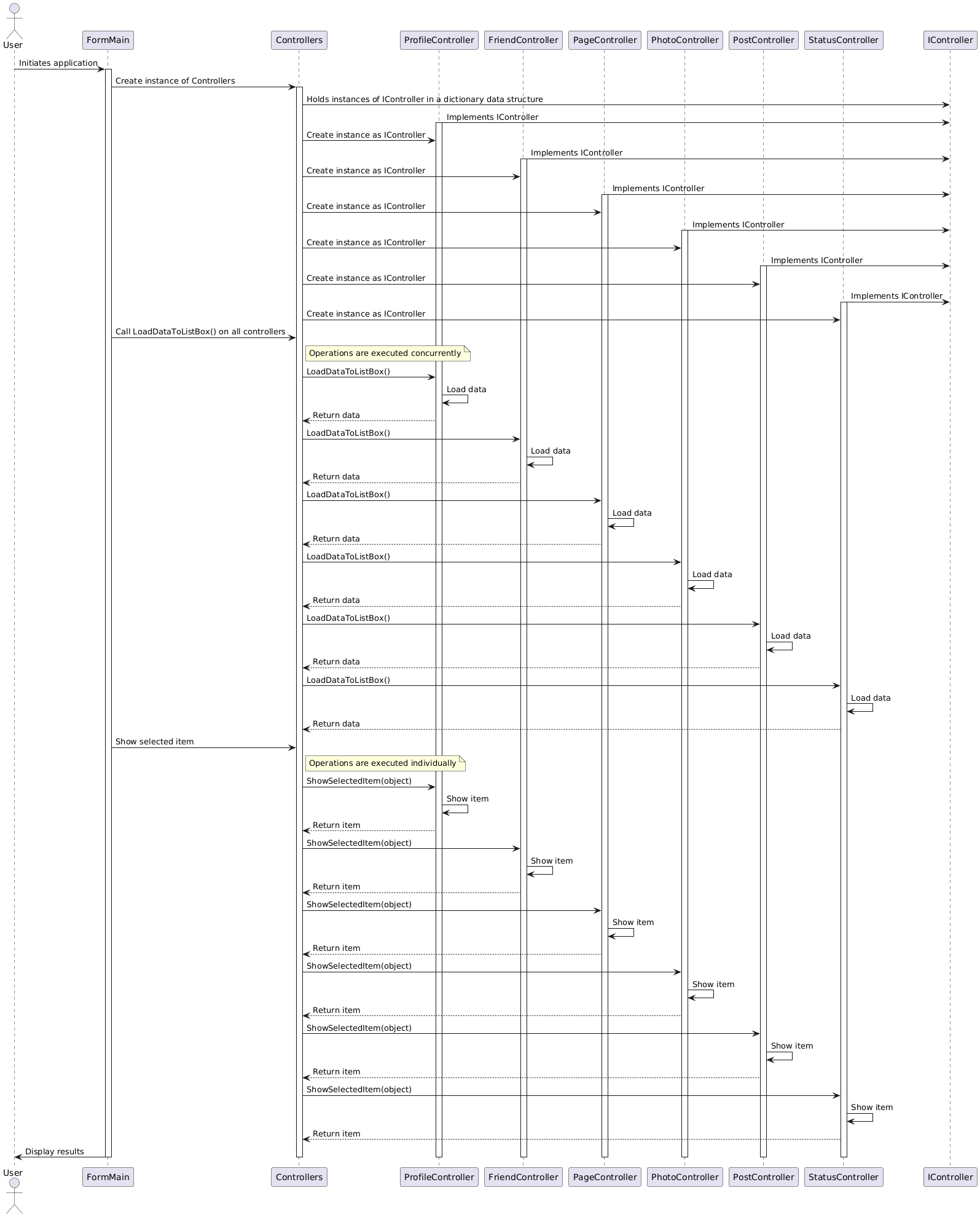
ניתן למצוא את המחלקה בקוד בתוך התיקייה ControllersFacade ובתוכה יש מחלקה בשם Controllers.

ממשק IController מאפשר לנו לנהל את כל הפאנלים השונים (Controller) דרך מבנה נתונים אחד מתוך המחלקה Controllers. זאת אומרת שכל Controller צריך לממש את הממשק IController וזה מאפשר לנו עבודה בצורה אחידה עם כל הפאנלים השונים (Controller).

ניתן למצוא את הממשק בקוד בתוך התיקייה ControllersFacade ובתוכה יש ממשק בשם IController.

מחלקה/Controller אילו הם הרכיבים הבודדים אשר מציגים כל פאנל באפליקציה. כל פאנל אחראי על הצגת הנתונים למשתמש בהתאם לתפקיד שלו. חילוץ המידע מה-FacebookAPI וההשמה שלו ברכיב המתאים. כל Controller צריך לממש את הממשק IController ובכך לאפשר עבודה נוחה אל מול המחלקה המנהלת Controllers.

ניתן למצוא את מחלקות ה-Controller השונים בתוך התיקייה Controller אשר נמצאת בתוך התיקייה ControllersFacade.

* Sequence Diagram
* Class Diagram

סימנו את ההקשר בין תבנית העיצוב שקשורה למחלקה שלנו לתבנית העיצוב הבסיסית בעזרת הצבעים השונים. הדיאגרמה הימנית היא הדיאגרמה שקשורה לקוד שלנו והדיאגרמה השמאלית היא הדיאגרמה הכללית של תבנית העיצוב Facade.

זאת אומרת המחלקות המקבילות הן באותו הצבע והמתודות המקבילות גם הן באותו הצבע בין התבנית הכללית לבין הקוד שלנו.

תמונה שמכילה טקסט, צילום מסך, גופן, אתר

התיאור נוצר באופן אוטומטי

### תבנית מס' 4 – Observer

* סיבת הבחירה / שימוש בתבנית:

סיבת הבחירה בתבנית Observer היא שכאשר המשתמש נכנס לפרופיל האישי שלו ומשנה את שמו הפרטי או את שם המשפחה שלו אנו רוצים שהשם המלא של המשתמש יתעדכן בטופס הראשי בצורה אוטומטית. מסיבה זו שמנו את המאזין בתוך המחלקה/Controller שברגע שהמשתמש משנה את שמו והשם אכן חוקי הוא מודיע שמישהו שינה אותו וקורא לפונקציה שאחראית על העדכון של השם המלא במחלקה הראשית שקוראים לה FormMain שמחזיקה במתודה שאחראית לטפל בשינוי.

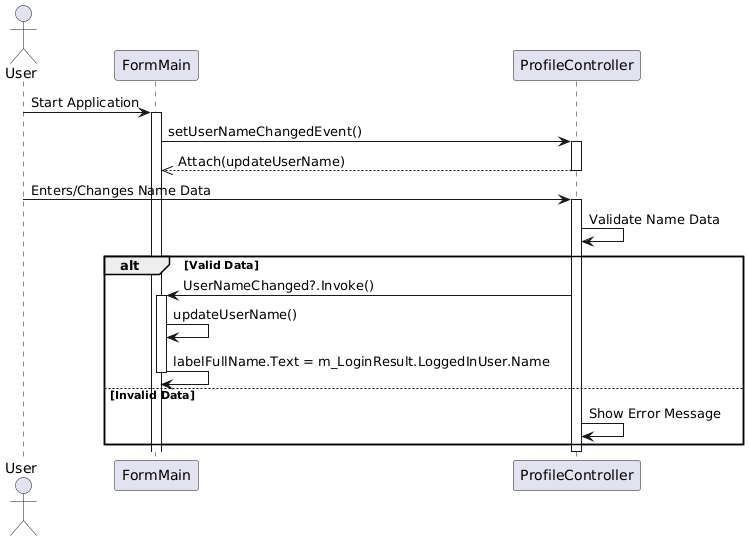
* אופן המימוש:

מחלקה FormMain (שהיא משמשת בתור Listener) מחזיקה במתודה אשר אחראית על עדכון ה-labelFullName. המתודה ניגשת למאפיין Name מתוך ה-FacebookAPI ושולפת משם את השם המעודכן של המשתמש, וזאת מבלי שעדכנו בפועל את המאפיין Name מכוון שבעזרת Two Way DataBinding ה-Name יתעדכן אוטומטית ברגע ששינינו את המאפיינים FullName ו-LastName בתוך ה-ProfileController בעזרת האובייקט של ה-LoggedInUser.

ניתן למצוא את המתודה (updateUserName) בקוד בתוך המחלקה FormMain שנמצאת בתוכנית הראשית. בנוסף, ניתן גם למצוא את המתודה בשם setUserNameChangedEvent אשר אחראית לשלוח ל-ProfileController את ה-Event (שורות 194 ו 202).

מחלקה ProfileController היא ה-Observer. ניתן למצוא במחלקה Event Action אשר מחזיק מצביע למתודה שאחראית לעדכן את ה-FormMain שהוא ה-Listener כאשר בוצע שינוי בשם הפרטי או בשם המשפחה וזאת לאחר שהשינוי עבר ולידציה.

ניתן למצוא את המחלקה בקוד בתוך התיקייה Controller שהיא נמצאת בתוך התיקייה ControllersFacade.

* Sequence Diagram
* Class Diagram

סימנו את ההקשר בין תבנית העיצוב שקשורה לקלאס שלנו לתבנית העיצוב הבסיסית בעזרת הצבעים השונים. הדיאגרמה השמאלית היא הדיאגרמה שקשורה לקוד שלנו והדיאגרמה הימנית היא הדיאגרמה הכללית של תבנית העיצוב Observer.

זאת אומרת המחלקות המקבילות הן באותו הצבע והמתודות המקבילות גם הן באותו הצבע בין התבנית הכללית לבין הקוד שלנו.

