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

* - **Best Times to Post**מציאת הזמן הכי טוב לפרסם פוסט בחתך של תמונות, סטטוסים ווידאו ולפי קבוע זמן. בנוסף לכך, הצגה של מפת חום אשר מציגה את הזמן הטוב ביותר על גבי המפה לפי חתך כולל של כל הפרמטרים המוזכרים לעיל.
* **Biggest Fans** - מציאת החברים אשר עושים הכי הרבה לייקים לפוסטים של המשתמש, אותם אנו מגדירים כ״מעריצים הגדולים ביותר״ לפי חתך כולל של פוסטים.

### תבנית מס' 1 – Façade

* סיבת הבחירה / שימוש בתבנית:

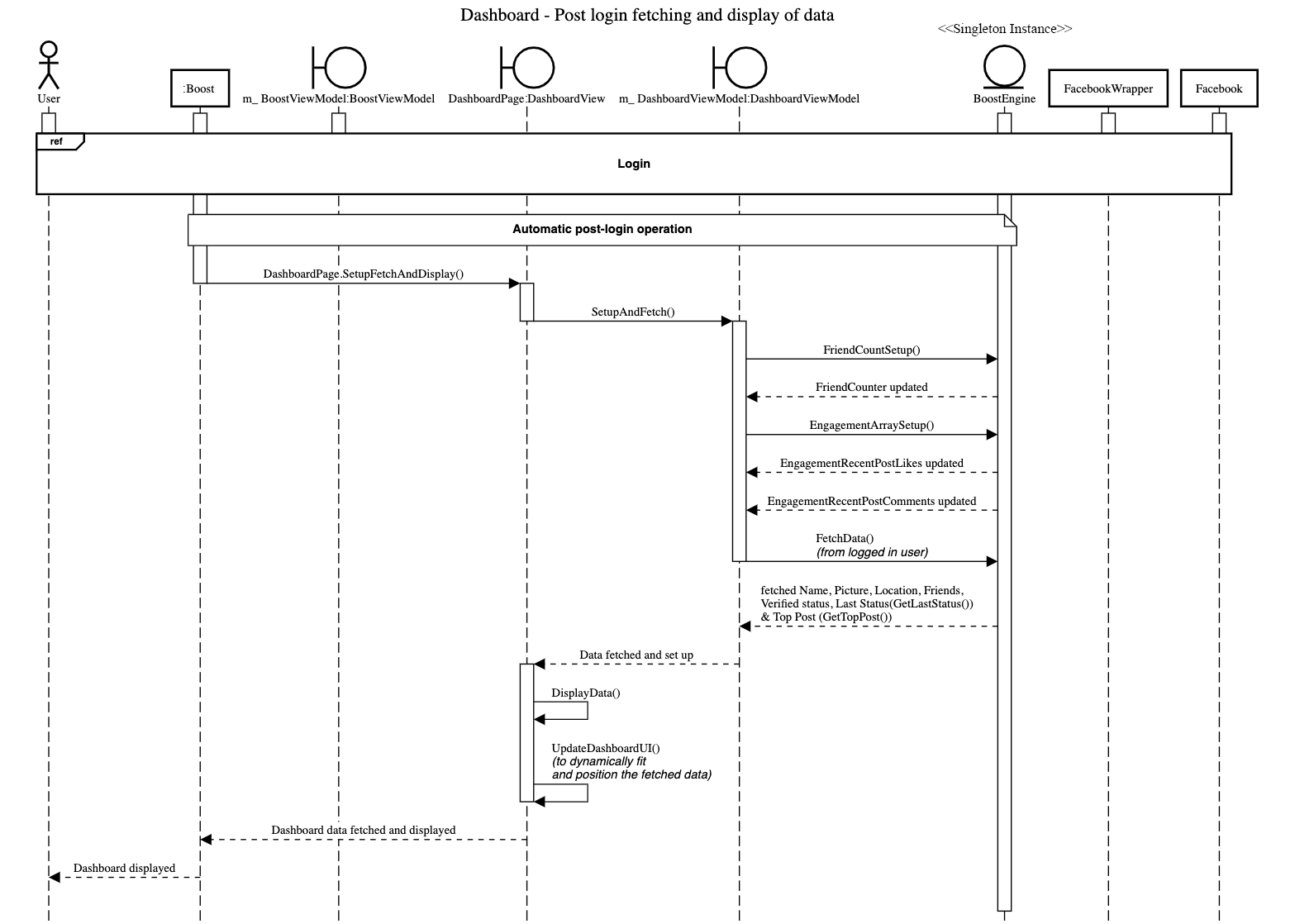
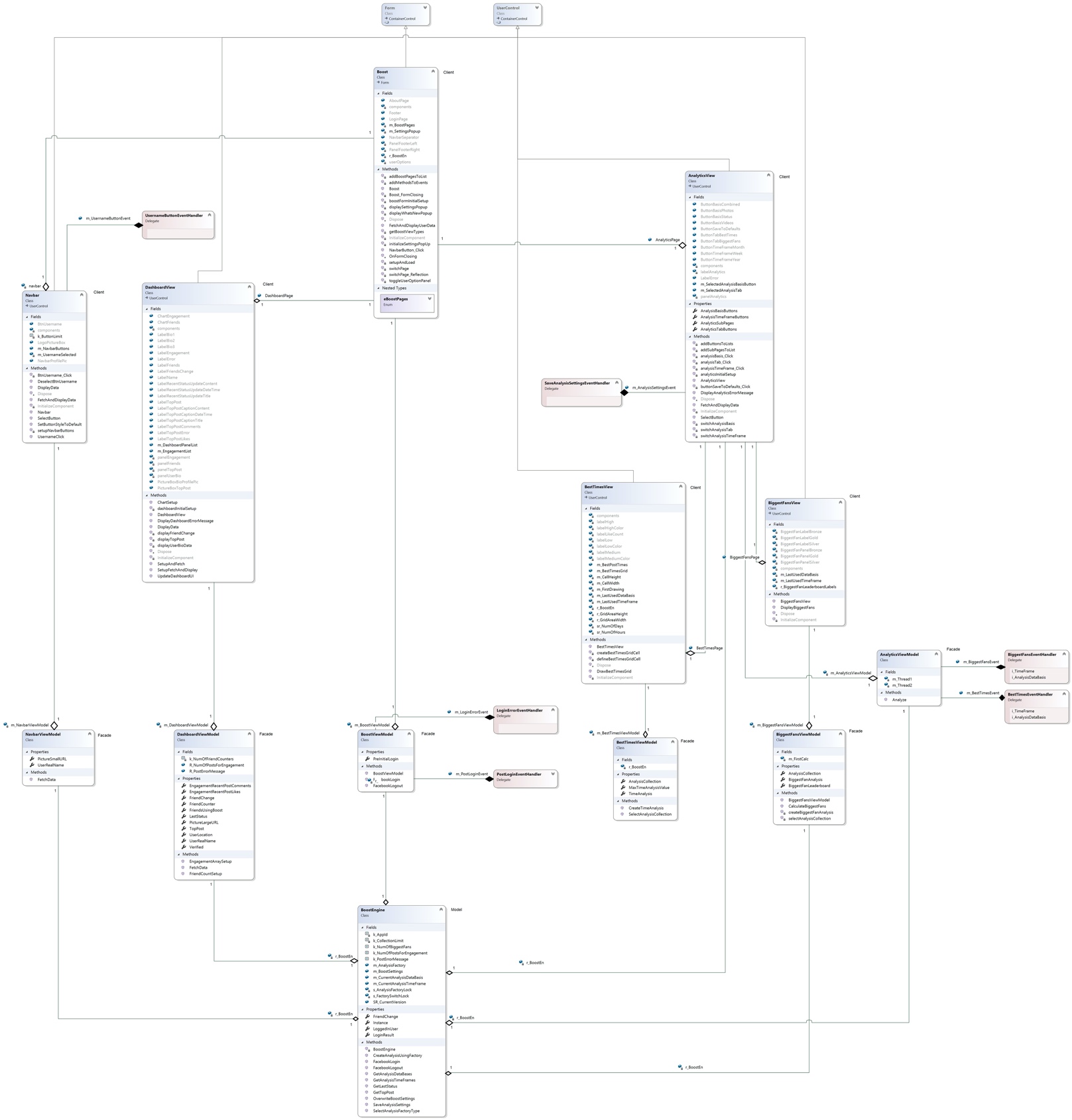
המערכת שלנו מכילה ה-UI מורכב בעל אלמנטים רבים: היא בנויה מ-Form ראשי (Boost) ו-User Controls כ-Views הנמצאים בתוכו המשרתים את תתי-האלמנטים שלו (עמודים, תפריט וכו׳). מאחר ובגרסה הקודמת, היו תהליכים שהתבצעו בלוגיקה וב-View, החלטנו להשתמש בתבנית העיצוב Façadeלצורך בניית View Models אשר יעשו את התהליכים (כגון טעינה וסידור) ויישאבו את הנתונים הנדרשים כך שה-View הרלוונטי רק יציג אותם.

דבר זה הופך את הקוד ליותר Maintainable ו-Resuable במקרה של החלפה/שינוי של ה-UI.

* אופן המימוש:

לכל View בו ישנה הצגה של חישובים/אנליזות/טעינת נתונים נבנה View Model רלוונטי הנמצא בפרוייקט תחת תיקייה בשם ViewModels.

כל View Model מכיל כפרופרטיז את המידע אשר ה-View הרלוונטי רק צריך להציג ובכך עושה בשבילו את כל התהליכים הנדרשים.

* Sequence Diagram
*  Class Diagram

### תבנית מס' 2 – Singleton

* סיבת הבחירה / שימוש בתבנית:

1. במערכת שלנו היינו צריכים גישה ל-BoostEngine אשר מכיל את כל החלק הלוגי של המערכת באופן כזה שהוא יוכל להיות נגיש מכל מחלקות ה-View וה-ViewModel. בצורה כזו, לא היינו צריכים להעביר את המופע בין המחלקות השונות שהשתמשו ב-BoostEngine. דבר זה עוזר במקרה של שימוש עתידי במחלקות נוספות שהיה עלול לגרור שינוי בקוד הקיים. השימוש בסינגלטון מונע זאת.
2. השימוש במופע היה צריך להיות באופן כזה שיהיה רק BoostEngine אחד במערכת. אם היה שימוש בכמה מופעים של המחלקה אז היה יכול להגרם מצב שהנתונים שהיו מופיעים בו לא היו מסונכרנים וכתוצאה מכך לא היו מוצגים לפי רצון המשתמש.

* אופן המימוש:

מימוש הקוד היה באמצעות ריפלקשן וגנריות (משתנה T), עפ״י הדרך שהודגמה בכיתה כאשר נעשה שימוש ב-Double Check Lock. בריפלקשן, מתבצע חיפוש של ה-Private Constructor של המחלקה שאיתה אנו מייצרים את אובייקט הסינגלטון.

מיקום מימוש הקוד הוא:

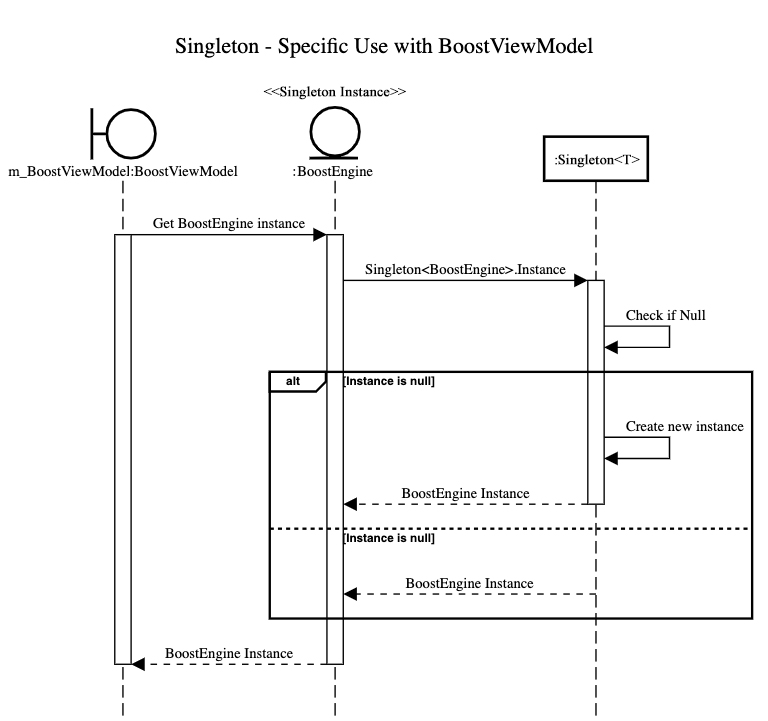
Project🡺Model🡺DesignPatterns🡺Singleton.cs

מיקום זה נבחר לכל התבניות עיצוב לשימוש עתידי אפשרי של תבניות אלה.

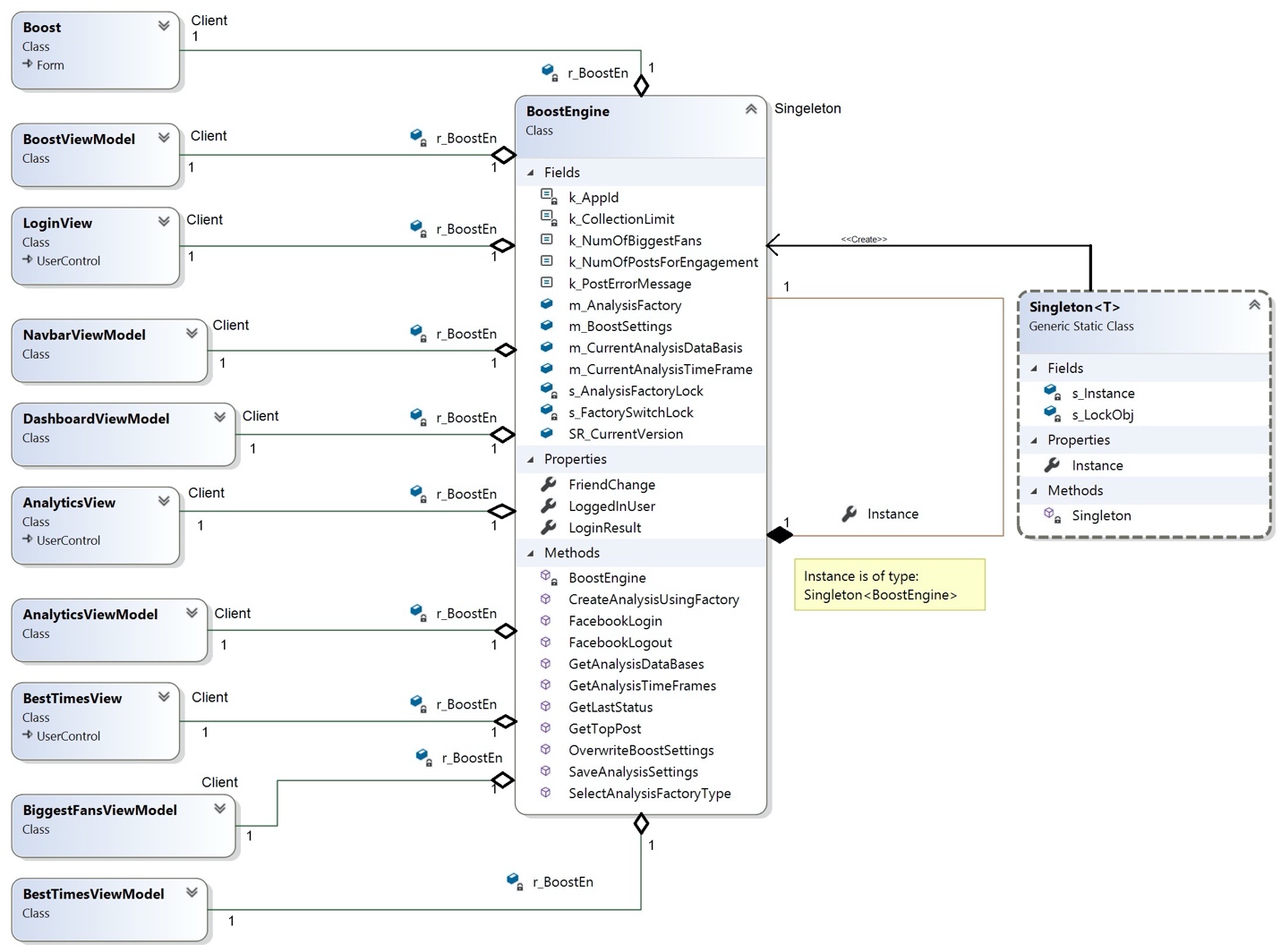
מיקום השימוש בתבנית נמצא ב:

Project🡺Model🡺BoostEngine.cs

* Sequence Diagram



* Class Diagram



### תבנית מס' 3 – Abstract Factory

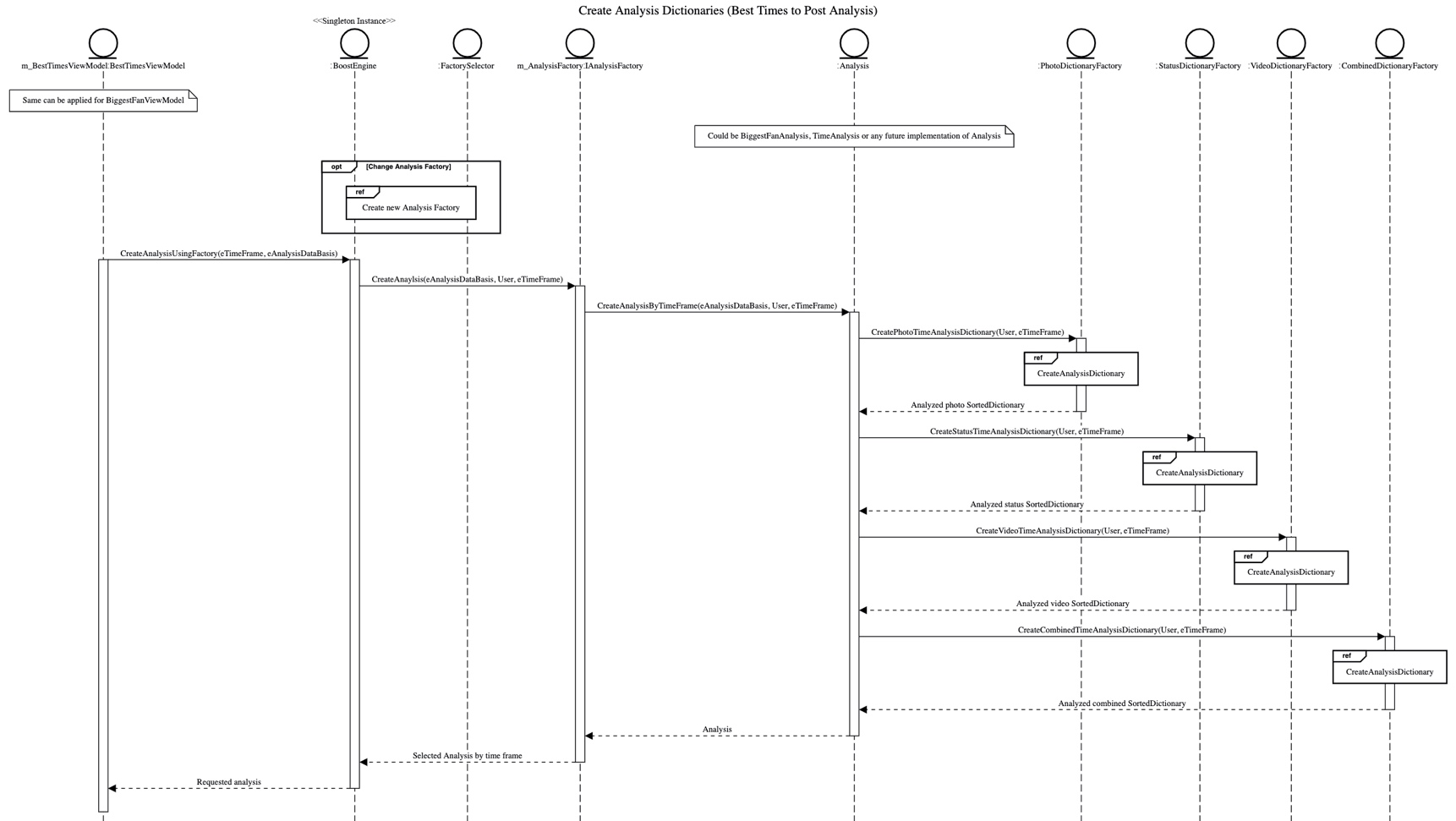
* סיבת הבחירה / שימוש בתבנית:

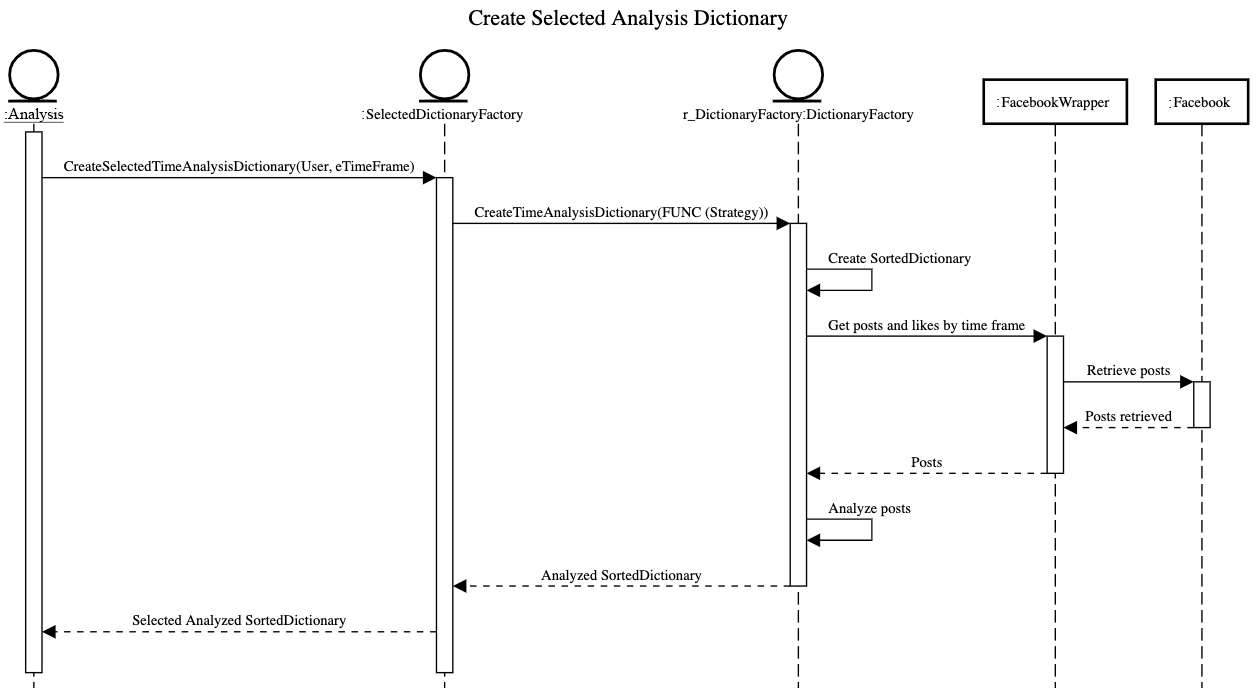
במערכת שלנו יש צורך בבנייה של כמה אנליזות שונות (משפחה פולימורפית) ובתוכן קיימת משפחה פולימורפית שבה מתבצע פילוח לפי סוגים שונים של בסיס נתונים לאנליזה (תמונות, סטטוסים, סרטונים וכולם יחדיו). שימוש בתבנית עיצוב זו דואגת לתחזוקה העתידית של הקוד בכך שאם בעתיד נוסיף סוג אנליזה חדש לא נצטרך לשנות דבר בקוד הקיים ולא יהיו בעיות תחזוקה בנושא.

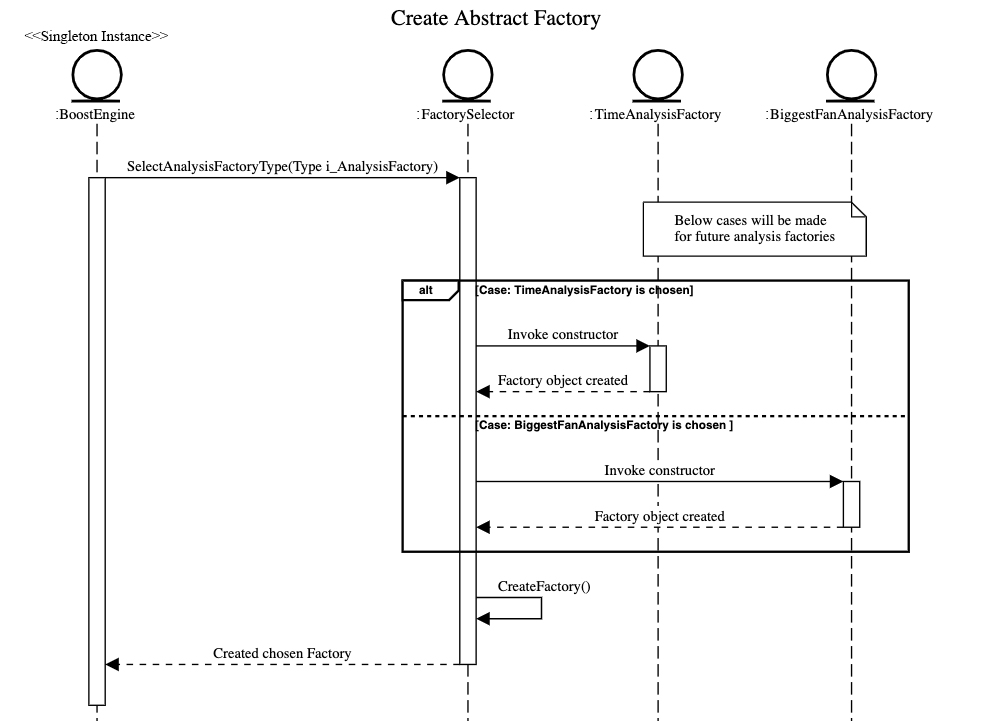
* אופן המימוש:

במערכת נוצר Interface אשר מוחזק ב-BoostEngine ממנו יורשות 2 מחלקות TimeAnalysisFactory ו- BiggestFanAnalysisFactoryאשר מחזירות אנליזה בנוייה לפי החתכים המבוקשים. בנוסף, השתמשנו ב-Adapter ו-Strategy על מנת ליצור את ה-FactoryMethod לכל אחת מהאנליזות.

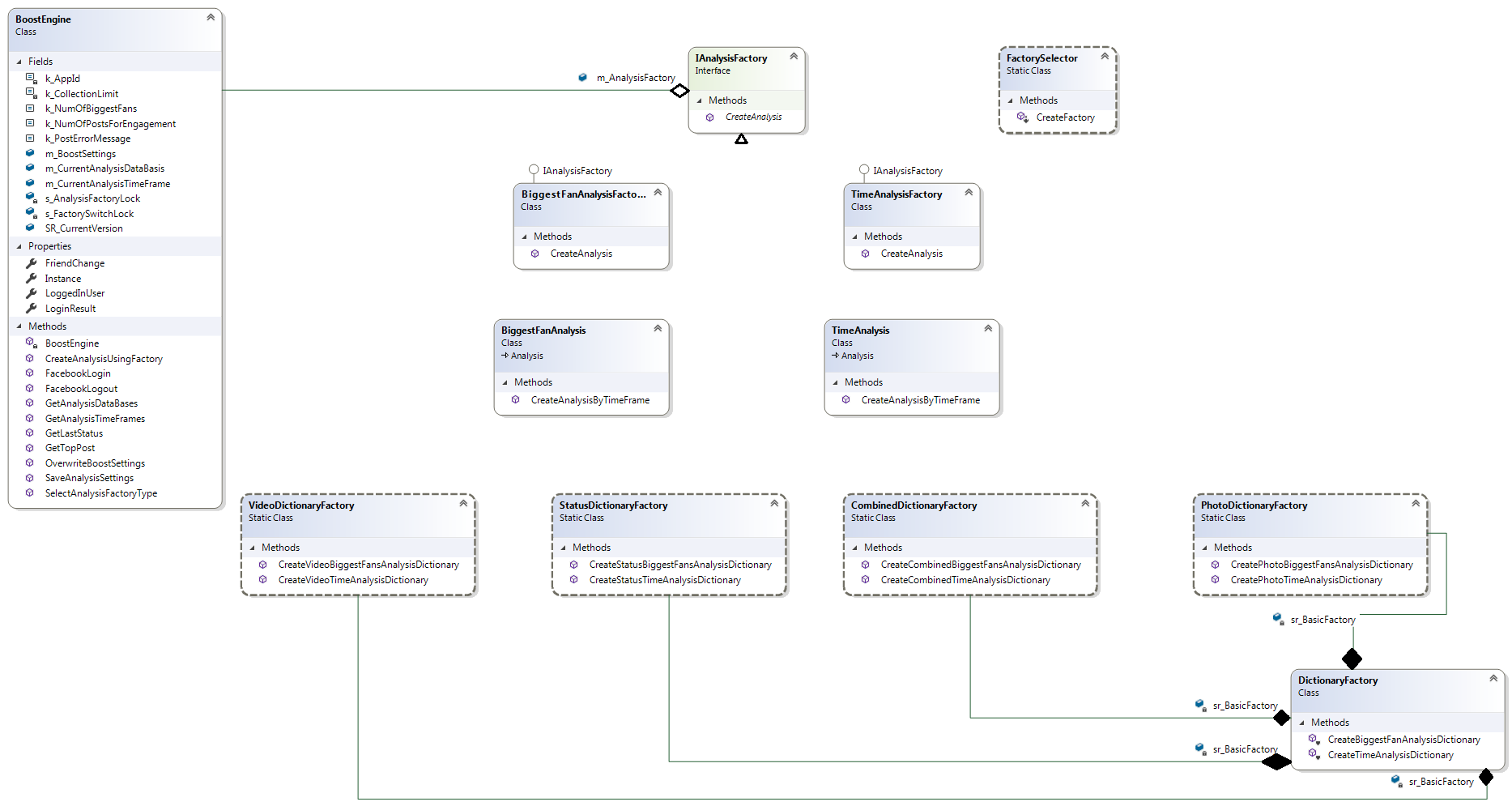
השתמשנו ב-Strategy בשלב יצירת האנליזה תוך שאלה: ״איזה סוג של פרמטר לאנליזה אנחנו רוצים לעשות?״. בנוסף, השתמשנו ב-Adapter לצורך עטיפת סוגי האנליזות הרצויות ובכך לאפשר ריפלקשן נוח וקוד קל לתחזוקה.

* Sequence Diagram





* Class Diagram



<<Create>>

<<Create>>

<<Use>>

<<Use>>

<<Use>>

<<Use>>

<<Use>>

<<Use>>

<<Use>>

<<Use>>

<<Create>>

<<Create>>

**Abstract Factory**

**Concrete Factory A**

**Concrete Factory B**

**Client**

**Product A**

**Product B**

1

1

1

1

1

1

1

1

1

1



**עבודה אסינכרונית**

עבודה אסינכרונית באה לידי ביטוי בכמה מקומות:

1. ViewModels🡺 AnalyticsViewModel.cs

נוצרו שני ת׳רדים לפעולות שני הפיצ׳רים המרכזיים שלנו הכוללים אנליזה וציור. ת׳רד 1 המתועדף בעדיפות גבוהה הוא לציור ואנליזה של הפיצ׳ר שהחלון שלו כרגע מופיע ות׳רד 2, המתועדף בתיעדוף רגיל עובד על הפיצ׳ר שהחלון שלו כרגע מוסתר.

הת׳רדים מושמים ומופעלים כחלק ממתודה Analyze.

1. View🡺 LoginView.cs

שימוש בת׳רדים חדשים למתודת האיבנט המפעילה את Connect במקרה שזוכרים את המשתמש משימוש קודם: m\_ContinueAsEvent.Invoke()

1. ViewModels🡺 BoostViewModel.cs

שימוש בת׳רד חדש למתודת הפעולות שבאות לאחר ההתחברות (Setup, Fetch, Display etc…): m\_PostLoginEvent.Invoke()

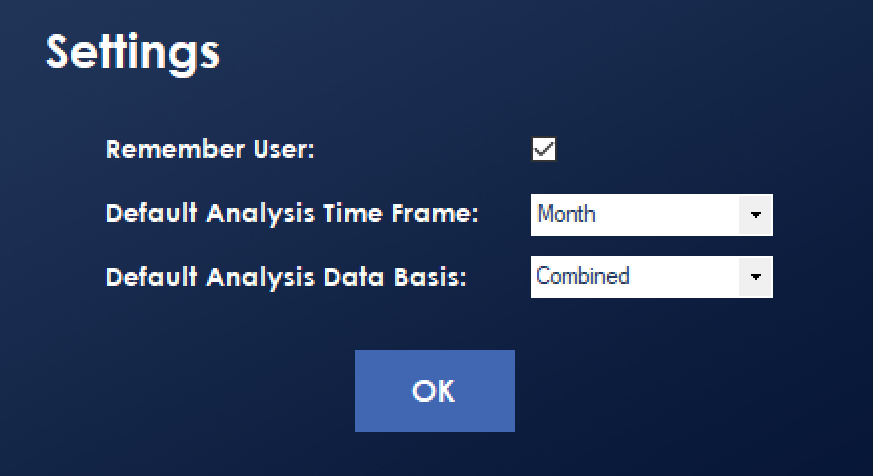
**עבודה עם Data Binding**

עשינו שימוש ב-Two-Way Data Binding בשני מקומות באותו הקשר: מחלקת הגדרות משתמש, BoostSettings.cs, בה יש לנו ערכי ברירת מחדל הנשמרים על המשתמש ולשימוש המשתמש.

המקומות בהם השתמשנו ב-Two-Way Data Binding הם:

1. View🡺Settings.cs

מחלקת Settings היא מחלקה של Form פופאפ של הגדרות משתמש:



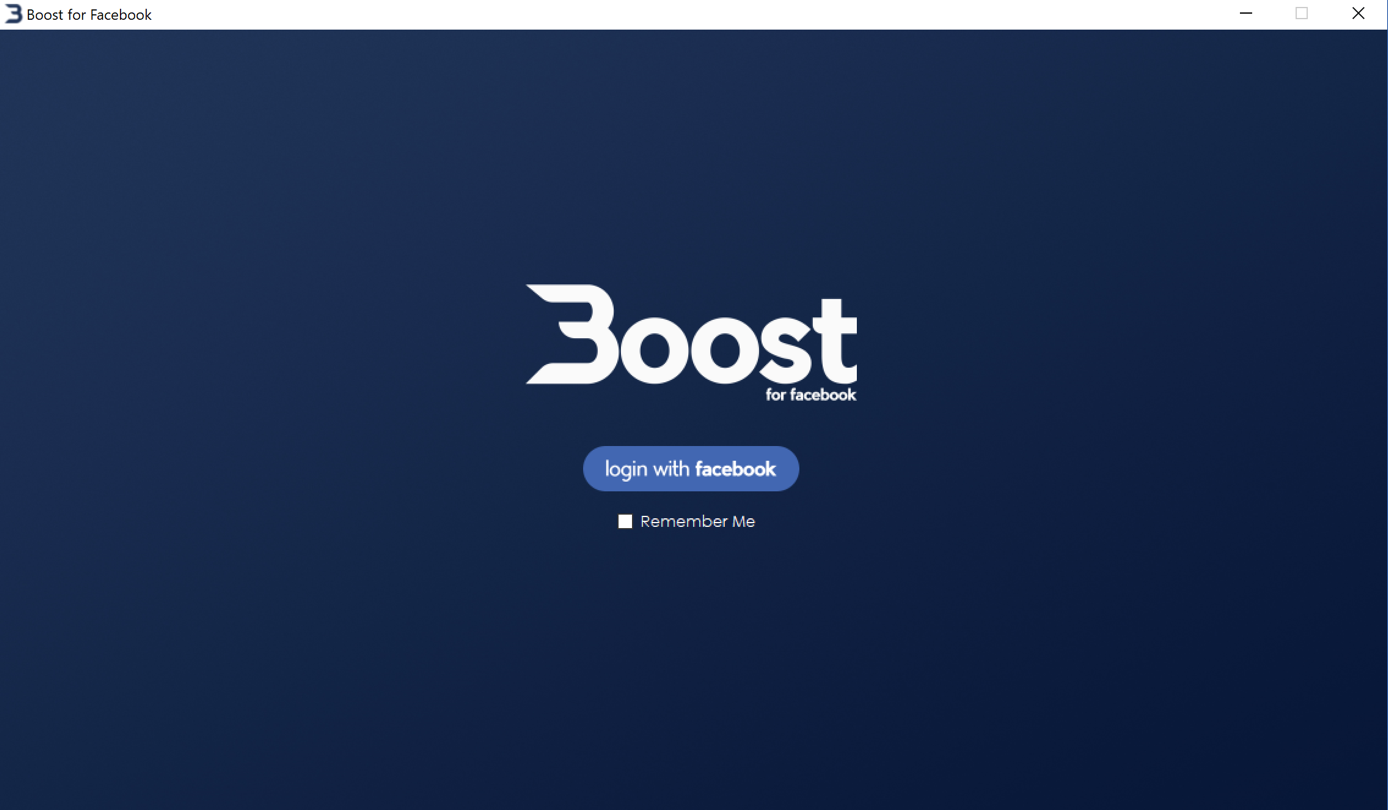
כל פעם שנעשה שינוי באחת מההגדרות (זכירת משתמש, קבוע זמן דיפולטי ובסיס דאטה דיפולטי) הגדרות מחלקת ההגדרות (שנשמרות לקובץ בסיגרת האפליקציה) משתנות בהתאם.

ניתן לראות את הקישור במחלקה עצמה ובמחלקת: Settings.Designer.cs

1. View🡺LoginView.cs

מחלקת LoginView מייצגת את עמוד ההתחברות לאפליקציה.

בדומה לחלון ההגדרות הנ״ל, יש Two-Way Data Binding לשמירת זכירת המשתמש:



כאשר נעשה שינוי בסימון ה-Checkbox, הממבר במחלקה משתנה אוטומטית ואת השינוי ניתן גם לראות בחלון ההגדרות הנ״ל.

ניתן לראות את הקישור במחלקה עצמה ובמחלקת: LoginView.Designer.cs

במידה ובהתחברות הבאה, ההגדרה של זכירת המשתמש היא חיובית, החלון יופיע בצורה הבאה:

