

דוט נט תשפ"ה - פרויקט - שלב 5

1.....	דוט נט תשפ"ה - פרויקט - שלב 5
2.....	מטרות השלב
2.....	הנחיות לביצוע השלב והגשתו
2.....	ראשי פרקים - שלב 5
3.....	הנחיות פרטניות - שלב 5
3.....	פרק 1 - הוספת תבנית עיצוב observer לשכבת ה BL
9.....	פרק 2 - דרישות כלליות של שכבת התצוגה PL
10.....	פרק 3 - הוספת פרויקט PL מסוג WPF
11.....	פרק 4 - יצירת מסכים לפרויקט PL
13.....	פרק 5 - מסך תצוגת הרשימה - עיצוב
16.....	פרק 6 - מסך ניהול ראשי - עיצוב ומימוש
20.....	פרק 7 - מסך תצוגת הרשימה - הצגת פרטי הרשימה
21.....	פרק 8 - מסך תצוגת הרשימה - סינון הרשימה
24.....	פרק 9 - מסך תצוגת פריט בודד
28.....	פרק 10 - פתיחת מסך תצוגת פריט בודד מתוך מסך תצוגת הרשימה
29.....	פרק 11 - מסך תצוגת רשימה - מחיקת פריט בודד מהרשימה
29.....	פרק 12 - עיצוב המסכים בהתאמה אישית
30.....	פרק 13 - תמונת מצב
31.....	פרק 14 - יצירת תג והגשת שלב 5 במודל
32.....	נספחים - שלב 5
32.....	נספח 1 - דוגמאות למסכים
34.....	נספח 2 - דוגמא להגדרה ושימוש ב Converter

מטרות השלב

יצירת שכבת תצוגה (ממשק משתמש גרפי) בסיסית עבור ישות שירות לוגית אחת.
בשלים 1-4 בנינו את שכבת הנתונים (DAL) ואת השכבה הלוגית (BL) של הפרויקט וכתבנו תוכניות ראשיות לבדיקת כל שכבה.

בשלב זה, שלב 5:

נוסיף לשכבת ה BL את מנגנון ה observers:

- לכל ישות שירות לוגית נוסיף את האפשרות להיות "מושקפת", כלומר לדווח לשכבת התצוגה PL (המשקיפה) על שינויים ועדכונים במשתני תצורה או ברשימות שבבסיס נתונים.
- נוסיף שכבת תצוגה בסיסית עבור ישות נתונים לוגית ונשתמש בישות השירות הלוגי שלה.
- ניצור כמה מסכים:
 - מסך ניהול ראשי
 - מסך לתצוגת רשימה של פריטים מאותה הישות
 - מסך לתצוגה/הוספה/עדכון לניהול אותה ישות
- נתפעל את המסכים הנ"ל ואת הנתונים המוצגים/המנוהלים בעזרת פניות לשכבה הלוגית
- נוסיף למסכים את האפשרות ל"השקיף" על שינויים בבסיס הנתונים

הנחיות לביצוע השלב והגשתו

- שימו לב! ההנחיות במהלך המסמך מלווים בהסברים חשובים, להבנת כל מהלך ומהלך. הבנה זו תועיל לכם מאוד גם במהלך ההגנה בעל פה על הפרויקט וגם במהלך המבחן.
- חובה על 2 השותפים להגיש במודל קישור על פי הנחיות ההגשה
- שימו לב! במידה ולא יהיה איזון מסוים בין אחוז הפעילות בגיט של 2 השותפים - הציון יהיה שונה בין 2 השותפים. דרך רישום הפעילות (push) בגיט נוכל לוודא ש 2 השותפים תורמים לתרגיל.
- חובה לקרוא את מסמך התיאור הכללי של הפרויקט לפני תחילת העבודה על השלב
- ההנחיות הנחשבות **לבחירה** תסומנה **בכתום**
- חובה לבצע את השלב באותו מאגר git והמאגר בענן ובאותו ה-Solution כמו בשלב 0 (המקדים)
- חובה להמשיך להקפיד על צורת כתיבת הקוד (הזחות, שמות וכו')
- חובה לתעד את כל הסוגים, המתודות, השדות, והתכונות בעזרת תיעוד מפורמט (///)

ראשי פרקים - שלב 5

- הוספה לשכבת ה BL את מנגנון ה observers
- דרישות כלליות של שכבת התצוגה PL
- יצירת פרויקט חדש ו-3 מסכים חדשים בתוכו
- עיצוב המסכים - הוספת פקדים
- מימוש הקוד האחורי של המסכים

הנחיות פרטניות – שלב 5

פרק 1 - הוספת תבנית עיצוב observer לשכבת ה BL

רגע לפני שנתחיל לפתח את שכבת התצוגה PL, נוסיף כמה שיפורים לשכבת ה BL, כך שתממש את חלקה בתבנית העיצוב Observer. **פרק זה מכיל בעיקר הנחיות טכניות, ותקבלו מחלקות מוכנות מאתנו אבל עליכם להבין היטב את הנעשה.**

1א. תבנית העיצוב Observer ושילובה בפרויקט שלנו

תבנית העיצוב Observer היא פתרון תכנותי למצב שבו אובייקט (הנקרא Subject - "המושקף"), מנהל רשימה של אובייקטים (הנקראים Observers - "המשקיפים") שמאזינים לשינויים בו, ומעדכן אותם אוטומטית כאשר מתרחש שינוי במצבו.

תבנית זו מאפשרת "צימוד רפוי" (loose coupling) בין האובייקטים, כך שה Observers יכולים להגיב לשינויים בלי שה Subject תלוי ישירות במימוש שלהם.

בפרויקט שלנו:

- ישויות השירות הלוגי (הממשקים) בשכבת ה BL יהיו ה Subject - האובייקט "המושקף"
 - השינויים שעשויים להתרחש בשכבת ה BL הם הוספה/עדכון/מחיקה של אובייקט מהרשימה בבסיס נתונים, וכן עדכון של משתני תצורה או שעון המערכת
- המסכים בשכבת ה PL יהיו ה Observers - "המשקיפים"
 - המסכים ירצו לקבל הודעה על שינויים המתרחשים ב BL על מנת לעדכן את תצוגת המסכים באופן אוטומטי

נממש את תבנית העיצוב Observer באמצעות:

- בשכבת ה BL - הגדרת events בכל ישות שירות לוגית
- בשכבת ה PL - כל מסך יכיל מתודה שדרכה הוא מעוניין לדעת על שינויים שהתרחשו, וירשום את המתודה הזו לזרוע event המתאים ב BL.

1ב. הוספה לשכבת ה BL את המחלקה המוכנה ObserverManager

לצורך ניהול נכון של "המושקף" ו"המשקיפים" שירשמו לקבלת שינויים, הכנו עבורכם מחלקת עזר בשם `ObserverManager.cs`.

המחלקה מאפשרת לנהל את המשקיפים על ישויות השירות הלוגיות בשכבת ה BL ומציעה את התשתית הבאה:

- **event delegate** - עבור מתודות שרוצות השקפה על רשימות שלמות בבסיס נתונים. רוצות לדעת אם הרשימה השתנתה (נוסף/נמחק/התעדכן בה אובייקט) כדי לעדכן את תצוגת הרשימה כולה במסך תצוגת הרשימה.
- **hash table for event delegate** - עבור מתודות שרוצות השקפה על אובייקט ספציפי שהתעדכן כדי לעדכן את תצוגת האובייקט הבודד במסך תצוגת פריט בודד.
- מתודות שמאפשרות להוסיף מתודה משקיפה לכל אחד מה delegate הנ"ל
- מתודות שמאפשרות להסיר מתודה משקיפה מכל אחד מה delegate הנ"ל
- מתודה שתזמן את כל המתודות שנרשמו להשקפה על שינויים ברשימה כולה
- מתודה שתזמן את כל המתודות שנרשמו להשקפה על שינויים באובייקט ספציפי

ניתן למצוא את הקובץ בקישור הבא: [ObserverManager.cs](https://github.com/ObserverManager/ObserverManager.cs)

1. דרך סיייר הקבצים, העתיקו את הקובץ הפיזי `ObserverManager.cs` תחת תת-הספרייה בשם `BL\Helpers` ולאחר מכן, הוסיפו אותו לפרויקט תחת `BL\Helpers`.

1ג. הוספה לשכבת ה BL את הממשק המוכן IObservable

נצטרך להרחיב את ממשקי ישויות השירות הלוגיות (המושקפים) כך שיאפשרו למסכי התצוגה ב PL (המשקיפים) להירשם לשינויים בשכבה הלוגית.

לצורך כך, הכנו עבורכם ממשק בשם `IObservable` המאפשר:

- הוספת מתודה שתשקיף על עדכונים ברשימה שלמה:

```
void AddObserver(Action listObserver);
```

- הוספת מתודה שתשקיף על עדכונים באובייקט בודד:
`void AddObserver(int id, Action observer);`
- הסרת מתודה שהשקיפה על עדכונים ברשימה שלמה:
`void RemoveObserver(Action listObserver);`
- הסרת מתודה שהשקיפה על עדכונים באובייקט בודד:
`void RemoveObserver(int id, Action observer);`

ניתן למצוא את הקובץ בקישור הבא: [IObservable.cs](#)

1. דרך סייר הקבצים, העתיקו את הקובץ הפיזי `IObservable.cs` תחת תת-הספרייה בשם `BL\BIApi` ולאחר מכן, הוסיפו אותו לפרויקט תחת `BL\BIApi`.

1ד. עדכון כל ישות שירות לוגית להיות מושקפת

1. עבור כל אחת מישויות השירות הלוגיות ב `BL` (ממשקים), חוץ מאשר ממשק הניהול הלוגי `IAdmin`:
a. הוסיפו למחלקת העזר שלה (מחלקת ה `Manger`) שדה פנימי סטטי מטיפוס `ObserverManager` בשם `Observers` ואתחלו אותו.

למשל כך:

```
namespace Helpers;
using DalApi;
using System;

internal static class StudentManager
{
    internal static ObserverManager Observers = new(); //stage 5
    //...
}
```

b. הרחיבו את ממשק השירות בממשק `IObservable`.

למשל כך:

```
namespace BIApi;

public interface IStudent : IObservable //stage 5
{
    void Create(BO.Student boStudent);
    //...
}
```

c. במחלקת המימוש של אותו ממשק לוגי, ממשו את 4 המתודות החדשות שנוספו לממשק כך שיפנו ל `ObserverManager` של אותה מחלקת ה `Manager` המתאימה לו, כדי להוסיף/להסיר את המתודה המתקבלת למנגנון ההשקפה.

למשל כך:

```
using BIApi;
using Helpers;
namespace BImplementation;

internal class StudentImplementation : IStudent
{
    // ...
    #region Stage 5
    public void AddObserver(Action listObserver) =>
        StudentManager.Observers.AddListObserver(listObserver); //stage 5
    public void AddObserver(int id, Action observer) =>
        StudentManager.Observers.AddObserver(id, observer); //stage 5
    public void RemoveObserver(Action listObserver) =>
        StudentManager.Observers.RemoveListObserver(listObserver); //stage 5
    public void RemoveObserver(int id, Action observer) =>
        StudentManager.Observers.RemoveObserver(id, observer); //stage 5
    #endregion Stage 5
}
```

1ה. עדכון כל ישות שירות לוגית לדווח על שינויים דרך מגנן Observer

כעת, לאחר שהפכנו כל ישות שירות לוגית להיות מושקפת. נסיף דיווחים על שינויים בה בכל מקום שנצרך בעזרת המגנן שהוספנו.

1. עבור כל אחת מישויות השירות הלוגיות ב BL (ממשקים), **חוץ מאשר ממשק הניהול הלוגי Admin!**
a. הוסיפו בתוך מחלקת המימוש, בכל מקום שבו נעשה שינוי באובייקטים של הישות הלוגית הזו, קריאה למתודה המתאימה שתעורר את כל המתודות שמשיקות על השינוי הזה.

דוגמא 1 במחלקת StudentImplementation:

מתוך המתודה Create/Delete, לאחר שנוצר/נמחק Student, נרצה לדווח על עדכון הרשימה כולה בעקבות ההוספה/מחיקה של אותו סטודנט. בפועל, הדיווח יהיה למסך שמציג את הרשימה שנרשם לקבלת עדכונים על הרשימה כולה.

מתוך המתודה Update, לאחר שמתעדכן Student, נרצה לדווח על עדכון הרשימה כולה בעקבות כך וכן על עדכון בודד של אותו סטודנט. בפועל, הדיווח יהיה למסך שמציג את הרשימה שנרשם לקבלת עדכונים על הרשימה כולה. וכן למסך שמציג כרגע את אותו סטודנט בודד שפרטיו עודכנו ממקום אחר במקביל.

```
using BIApi;
using Helpers;
namespace BImplementation;

internal class StudentImplementation : IStudent
{
    private readonly static BIApi.IDal s_dal = BIApi.Factory.Get;

    public void Create(BO.Student boStudent)
    {
        //...
        s_dal.Student.Create(doStudent); //stage 4
        StudentManager.Observers.NotifyListUpdated(); //stage 5
        //...
    }

    public void Update(BO.Student boStudent)
    {
        //...
        s_dal.Student.Update(doStudent); //stage 4
        StudentManager.Observers.NotifyItemUpdated(doStudent.Id); //stage 5
        StudentManager.Observers.NotifyListUpdated(); //stage 5
        //...
    }

    public void Delete(int id)
    {
        //...
        s_dal.Student.Delete(id); //stage 4
        StudentManager.Observers.NotifyListUpdated(); //stage 5
        //...
    }

    //...
}
```

2. עבור כל אחת ממשיות השירות הלוגיות ב BL (ממשקים), חוץ מאשר ממשק הניהול הלוגי Admin:
a. הוסיפו בתוך מחלקת העזר, בכל מקום שבו נעשה שינוי באובייקטים של הישות הלוגית הזו, קריאה למתודה המתאימה שתעורר את כל המתודות שמשקיפות על השינוי הזה.

דוגמא 2 במחלקת StudentManager:

מתוך המתודה לעדכון תקופתי של סטודנטים, בכל פעם שנזהה שסטודנט עודכן נרצה לדווח על כך ובעקבות כך על עדכון הרישמה כולה. בפועל, הדיווח יהיה למסך שמציג את הרישמה שנרשם לקבלת עדכונים על הרישמה כולה. וכן למסך שמציג כרגע את אותו סטודנט בודד שפרטיו עודכנו ממקום אחר במקביל.

```
namespace Helpers;
using DalApi;
using System;

internal static class StudentManager
{
    internal static ObserverManager Observers = new(); //stage 5

    //...

    internal static void PeriodicStudentsUpdates(DateTime oldClock, DateTime newClock) //stage 4
    {
        bool studentUpdated; //stage 5

        //if a specific student info is changed - then call NotifyItemUpdated(id, false)
        //and after all - call NotifyListUpdated();
        lock (AdminManager.blMutex) //stage 7
        {
            studentUpdated = false; //stage 5
            var list = s_dal.Student.ReadAll().ToList(); //stage 4
            foreach (var doStudent in list) //stage 4
            {
                //if student study for more than MaxRange years
                //then student should be automatically updated to 'not active'
                if (AdminManager.Now.Year - doStudent.RegistrationDate?.Year >=
                    s_dal.Config.MaxRange) //stage 4
                {
                    studentUpdated = true; //stage 5
                    s_dal.Student.Update(doStudent with { IsActive = false }); //stage 4
                    Observers.NotifyItemUpdated(doStudent.Id); //stage 5
                }
            }

            //if the current year was changed
            //it means that we need to announce that the whole list of student was updated
            bool yearChanged = oldClock.Year != newClock.Year; //stage 5
            if (yearChanged || studentUpdated) //stage 5
                Observers.NotifyListUpdated(); //stage 5
        }

        //...
    }
}
```

1. עדכון ישות הניהול הלוגית Admin להיות מושקפת

שיפרנו עבורכם את מחלקת ClockManager כך שתרכז ותנהל את ההשקפה והעדכון של - לא רק של שעון המערכת אלא גם של משתני התצורה. לכן המחלקה ClockManager תשנה כעת את שמה ל AdminManager.

המחלקה החדשה מכילה גם תוספות שיהיו רלוונטיות גם לשלב 7 אך לא מפריעות בשלב זה.

תוספות שהוספנו למחלקת AdminManager:

1. הוספנו למחלקת AdminManager עוד event בשם ConfigUpdatedObservers עבור מי שירצה להשקיף על עדכון משתני תצורה (מעבר ל event בשם ClockUpdatedObservers שהיה שם קודם לצורך שעון המערכת בלבד).
2. הוספנו למחלקת AdminManager דוגמא לתכונה עבור משתנה תצורה בשם MaxRange שמדווח על עדכונים בתכונת התצורה הזו.

```
internal static int MaxRange
{
    get => s_dal.Config.MaxRange;
    set {
        s_dal.Config.MaxRange = value;
        ConfigUpdatedObservers?.Invoke(); // stage 5
    }
}
```

3. הוספנו למחלקת AdminManager שתי מתודות בשם InitializeDB, ResetDB שיבצעו את אתחול/איפוס של בסיס הנתונים בצורה מסודרת: (1) אתחול/איפוס, (2) עדכון שעון לזמן הנוכחי, (3) עדכון משתני התצורה לערך הנוכחי (עדכונים שמעוררים את ה event לדווח למשקיפים) (מסכי התצורה) על השינוי.

כך נראות המתודה InitializeDB במחלקת AdminManager (חלקים ממנה רלוונטיים לשלב 7 אך לא מפריעים בשלב זה):

```
internal static void InitializeDB()
{
    lock (B1Mutex) //stage 7
    {
        DalTest.Initialization.Do();
        AdminManager.UpdateClock(AdminManager.Now); // stage 5 - needed for update the PL
        AdminManager.MaxRange = AdminManager.MaxRange; // stage 5 - needed for update the PL
    }
}

internal static void ResetDB()
{
    lock (B1Mutex) //stage 7
    {
        s_dal.ResetDB();
        AdminManager.UpdateClock(AdminManager.Now); //stage 5 - needed for update PL
        AdminManager.MaxRange = AdminManager.MaxRange; //stage 5 - needed for update PL
    }
}
```

ניתן למצוא את הקובץ בקישור הבא: AdminManager.cs

1. הסירו מהפרויקט את הקובץ הקודם ClockManager.cs ומחקו אותו גם פיזית דרך סייר הקבצים.
2. דרך סייר הקבצים, העתיקו את הקובץ הפיזי AdminManager.cs תחת תת-הספרייה בשם BL\Helpers ולאחר מכן, הוסיפו אותו לפרויקט במקום הקובץ הקודם תחת BL\Helpers.
3. בתוך המחלקה, הגדירו תכונות עבור משתני התצורה שלכם, על פי אותו פורמט של הדוגמא שסיפקנו. משתנה התצורה בשם MaxRange רלוונטי לדוגמת הסטודנטים ואין לכם אותו בפרויקט שלכם. בכל מקום שהוא מופיע יש להתאים למשתני הסביבה הנדרשים בפרויקט שלכם.
4. בכל המקומות בקוד שבו היתה פניה למחלקת ClockManager החליפו את הפניה לשם המחלקה החדש AdminManager.
5. בצעו את העדכונים הבאים בתוך המחלקה AdminImplementation:
a. בכל מקום שבו פניתם למשתנה התצורה בעזרת גישה ישירה דרך IConfig, למשל כך:

```
public int GetMaxRange() => s_dal.Config.MaxRange
public void SetMaxRange(int maxRange) => s_dal.Config.MaxRange
```

עדכנו את הגישה למשתנה התצורה דרך מחלקת AdminManager שגם מדווחת על שינויים בערך התכונה במידת הצורך. למשל כך:

```
public int GetMaxRange() => AdminManager.MaxRange;
public void SetMaxRange(int maxRange) => AdminManager.MaxRange = maxRange;
```

b. עדכנו את הקוד של מתודת האתחול InitializeDB ומתודת האיפוס ResetDB במחלקת AdminImplementation שתזמנה את המתודות המתאימות שנוספו כעת ב AdminManager:

```
public void InitializeDB()
{
    AdminManager.InitializeDB();
}

public void ResetDB()
{
    AdminManager.ResetDB();
}
```

c. הוסיפו לממשק הניהול הלוגי IAdmin, ארבע מתודות שיאפשרו הוספה/הסרה של מתודות משקיפות על שרון המערכת ועל משתני התצורה. למשל כך:

```
#region Stage 5
void AddConfigObserver(Action configObserver);
void RemoveConfigObserver(Action configObserver);
void AddClockObserver(Action clockObserver);
void RemoveClockObserver(Action clockObserver);
#endregion Stage 5
```

d. במחלקת המימוש AdminImplementation, ממשו את 4 המתודות החדשות שנוספו לממשק IAdmin, שיפנו ל AdminManager כדי להוסיף/להסיר את המתודה המתקבלת למנגנון ההשקפה על השרון ועל משתני התצורה. למשל כך:

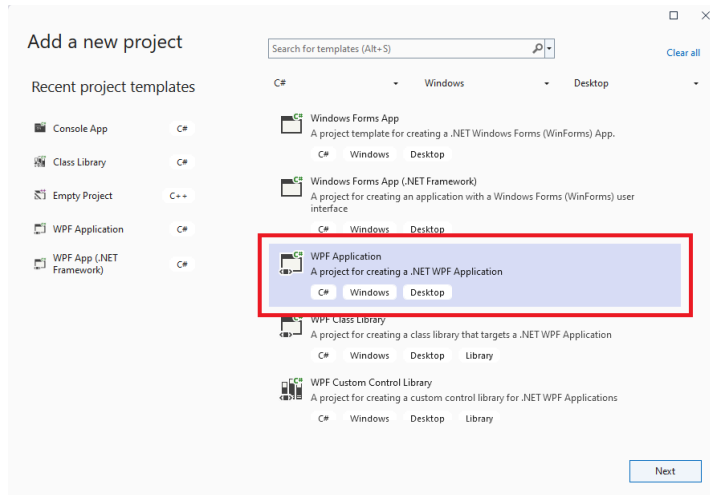
```
#region Stage 5
public void AddClockObserver(Action clockObserver) =>
    AdminManager.ClockUpdatedObservers += clockObserver;
public void RemoveClockObserver(Action clockObserver) =>
    AdminManager.ClockUpdatedObservers -= clockObserver;
public void AddConfigObserver(Action configObserver) =>
    AdminManager.ConfigUpdatedObservers += configObserver;
public void RemoveConfigObserver(Action configObserver) =>
    AdminManager.ConfigUpdatedObservers -= configObserver;
#endregion Stage 5
```


פרק 2 - דרישות כלליות של שכבת התצוגה PL

1. בשכבת התצוגה **חובה** לתפוס את כל החריגות ולהציג למשתמש הודעה מסודרת (MessageBox) על הבעיה
2. ההודעה על הבעיה חייבת להיות ברורה ומובנת למשתמש של המערכת (לבן אדם רגיל ולא רק למתכנת) - זאת אומרת המטרה היא **לא** להציג את הודעת החריגה ושמות המשתנים וכדומה, אלא להציג הודעה אינפורמטיבית ברמה של משתמש המערכת **עם כותרת משמעותית, תוכן מובן ואייקון מתאים**.
3. במסכים של **הוספה** של ישות - לאחר ביצוע מוצלח של ההוספה - מסך ההוספה צריך להיסגר אוטומטית ומסך הרשימה חייב להתעדכן אוטומטית ולכלול את הישות שנוספה.
4. במסכים **עדכון** של ישות - לאחר ביצוע מוצלח של העדכון - מסך העדכון צריך להיסגר אוטומטית ומסך הרשימה חייב להתעדכן אוטומטית ולכלול את הישות המעודכנת.
5. עדכון אוטומטי של התצוגה במסך הרשימה בעקבות הוספה/עדכון - יתבצע בעזרת **תבנית עיצוב observer** בשכבה הלוגית והרישום ל observers יעשה מתוך שכבת ה PL.
6. **לא יהיה שיתוף מידע ישיר בין המסכים** - העברת המידע תתבצע בעזרת העברת ארגומנטים עבור הפרמטרים של בנאי המסך.
7. על מנת לאפשר שמירה על עקרון ה-MVVM (Model-View-ViewModel), שמפריד בין הלוגיקה לעיצוב - **אסור לתת שם לפקדים דרך ה XAML כדי להשתמש בהם בקוד האחורי**. אלא יש לעשות שימוש במנגנון ה- **Data Binding** שמאפשר את ההפרדה בין הלוגיקה (קוד אחורי) לעיצוב (XAML).
8. יש **לצמצם את הקוד האחורי** למינימום הכרחי לטיפול בקלט מהמשתמש ובאירועים אחרים של WPF:
 - **חובה להשתמש ב- Data Binding באופן מלא:**
 - עבור נתונים המוצגים במסך, כגון: תוכן של ישות בודדת או רשימה של ישויות, **חובה** להגדיר תכונות מקבילות בקוד האחורי של **כל אחד מהמסכים**. התכונות המקבילות בקוד האחורי, מיועדות על מנת להחזיק את הנתונים המוצגים ו-או המתעדכנים. יש להגדיר את התכונות בקוד האחורי, כך **שעדכון תכונות אלו דרך הקוד האחורי יגרום לעדכון אוטומטי שלהן בתצוגה**. התכונות יוגדרו בקוד האחורי בעזרת אחת משתי האפשרויות:
 - **אפשרות 1:** במחלקה יוגדר אירוע של ממשק **INotifyPropertyChanged**, וב- **set** של התכונות האלה יופעל האירוע בהתאם
 - **אפשרות 2:** התכונות יוגדרו כתכונות תלות (Dependency Properties)
 - קישור האטריבוטים (ה- Binding) של הפקדים לנתונים יתבצע אך ורק דרך ה- **xaml בכל אחד מהמסכים** על מנת לסנכרן את התצוגה עם הנתונים (ולא דרך הקוד האחורי)
 - **אסור** לשנות דרך קוד האחורי את ה- **DataContext** או **ItemSource** – כי אלו הן גם **אטריבוטים של פקדים**, אלא יש לקשור אותן מתוך xaml לנתונים המתאימים
 - בזכות ה- **Binding** שיעשה דרך ה- xaml, אזי שינוי הנתונים בקוד האחורי ישפיע באופן אוטומטי על האטריבוטים של הפקדים בתצוגה.
 - במקרים שצריך לקשור בין אטריבוטים של שני פקדים - מותר לתת שם לפקד עם אטריבוט היעד
 - אם ברצונכם להוסיף תיאור לפקד (בתוך xaml) - על מנת שיהיה ברור למה הפקד נועד - כדאי להשתמש בהערות בפורמט: **<!--!> ... comment ...** →
 - דרך נוחה להוסיף הערה: כותבים את טקסט ההערה במקום שרוצים, עומדים על שורת ההערה ואז לוחצים על כפתור ההערה, בתפריט העליון.
 - **אסור** לגשת לתכונות של sender שמגיע כפרמטר במתודות טיפול באירועים (גם לא לאחר המרה לטיפוס המתאים) - אלא יש תמיד להתבסס על **DataBinding**
 - בשכבה זו **לא תתבצע לוגיקה** של הפרויקט למעט בדיקת תקינות קלט בסיסית (פורמט, תווים חוקיים וכו')
 - יש להשתמש ב- **IConverter** בשילוב עם **Data Binding** לצורך המרות או על מנת לחסום קלט בפקדים בצורה דינאמית או להסתיר פקדים בצורה דינאמית.
 - בכל פעולה המבוצעת מ-GUI יש לשלוח **לכל היותר בקשה אחת ל-BL, ולזכור לתפוס את החריגות האפשריות כתוצאה מפעולה זו**.
9. בשלב זה אפשר כבר להשתמש (אך עדיין לא חובה, אפשר לדחות את זה לשלב 6) במגוון רכיבים מתקדמים של **WPF מעבר לפקדים הבסיסיים, לדוגמא:**

- תסדירים (Layout) למיניהם (Grid, StackPanel, ועוד)
- שימוש במשאבים (Resources)
- שימוש בסגנון (Style)
- תבניות עיצוב (DataTemplate) בכללי ותבניות אלמנט בפקד מרובה נתונים בפרט

פרק 3 - הוספת פרויקט PL מסוג WPF



1. הוסיפו ל Solution פרויקט חדש בשם PL מסוג WPF Application.
2. לאחר שייפתח פרויקט חדש תחת הפתרון, לחצו לחיצה כפולה על שם הפרויקט ב-Solution Explorer
3. ייפתח לעריכה קובץ לניהול המאפיינים של הפרויקט, הוסף לקובץ את השורות הבאות (המודגשות בצהוב) בסוף האלמנט PropertyGroup:

```
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <OutputType>WinExe</OutputType>
    <TargetFramework>net8.0-windows</TargetFramework>
    <Nullable>enable</Nullable>
    <UseWPF>true</UseWPF>
    <BaseOutputPath>$(SolutionDir)\bin\</BaseOutputPath>
    <OutputPath>$(SolutionDir)\bin\</OutputPath>
    <AppendTargetFrameworkToOutputPath>>false</AppendTargetFrameworkToOutputPath>
    <AppendRuntimeIdentifierToOutputPath>>false</AppendRuntimeIdentifierToOutputPath>
    <UseCommonOutputDirectory>true</UseCommonOutputDirectory>
  </PropertyGroup>
</Project>
```

4. בצעו שמירה (Save) של הקובץ
5. הוסיפו references מכיוון פרויקט PL לעבר פרויקט BL
6. ודאו בקובץ המאפיינים של הפרויקט שהטקסט שלו התעדכן לאחר הוספת הרפרנס:

```
<ItemGroup>
  <ProjectReference Include="..\BL\BL.csproj"/>
</ItemGroup>
```

7. סמנו את הפרויקט PL כפרויקט ההתחלתי של ה-Solution, הפרויקט שירץ כשלוחצים על כפתור ה-PLAY הירוק:
 - a. עמדו עם עכבר ימני על הפרויקט PL
 - b. ואז לחצו על Set as Startup Project

פרק 4 - יצירת מסכים לפרויקט PL

בשלב זה, ניצור מסכי תצוגה בסיסיים עבור **ישות שירות לוגית נבחרת אחת**, שדרכה נציג את ישויות הנתונים הלוגיות הרלוונטיות עבורה. **במסמך התיאור הכללי של הפרויקט תציין אותה ישות לוגית שעליה תעבדו בשלב זה.** וכן המסכים הנדרשים עבורה בשלב 5.

4א. פירוט המסכים שירכיבו את הפרויקט בשלב זה

בשלב 5 לא ניצור את **מסך הכניסה למערכת**, אלא נאפשר כניסה ישירה **למסך הניהול הראשי**. בשלב 6 תוסיפו את מסך הכניסה למערכת אשר יוביל מסך הניהול הראשי.

3 המסכים שניצור בשלב זה:

1. **מסך ניהול ראשי זמני** - דרכו ניתן יהיה להגיע לשאר המסכים. לכל המסכים שניצור בעתיד ובפרט למסך מ"ס 2.
2. **מסך תצוגת רשימת הפריטים** - מסך להצגת רשימה של כל הפריטים מסוג **ישות נתונים לוגית**. הפריטים שיוצגו ברשימה יכילו **מידע מצומצם** אודות **ישות נתונים לוגית רחבה אחרת**. בלחיצה על על פריט ברשימה יוצג המידע המורחב יותר במסך מ"ס 3.
3. **מסך לתצוגה/הוספה/עדכון של פריט בודד** - מסך לניהול פריט בודד מסוג **ישות הנתונים הלוגית**.

לדוגמא:

- נניח שישות הנתונים הלוגית הרחבה **BO.Course** היא זו שנבחרה במסמך התיאור הכללי של הפרויקט.
 - אזי, במסך מספר 2 תופיע רשימה של פריטים מסוג **ישות הנתונים הלוגית BO.CourseInList**, זו ישות המכילה מידע מצומצם על קורס.
 - בלחיצה כפולה על פריט ברשימה, נפתח מסך מספר 3 שמציג מידע מלא ומפורט על אותו פריט מסוג **ישות BO.Course**. ובמסך זה, ניתן לצפות במידע המלא ו/או לעדכן אותו.
 - בנוסף, אם רוצים להוסיף קורס חדש, יפתח **אותו מסך מספר 3**, אך עם שדות ריקים. ולאחר שהם ימולאו בערכים יתווסף אותו קורס לרשימה במסך מספר 2.
- ניתן לראות שעם יצירת הפרויקט, **נוצר לנו כבר באופן אוטומטי מסך מספר 1**, מסך הניהול הראשי, בשם **MainWindow**, נשתמש בו ונעדכן אותו בהמשך המסמך.
- כעת, ניצור 2 מסכים נוספים, **מסך מספר 2** - להצגת רשימת הפריטים ו**מסך מספר 3** - להצגת/עדכון/הוספה של פריט בודד ברשימה.
- בשלב זה, אנו נציע לכם הצעה לעיצוב ונדרוך אתכם צעד צעד. לאחר מכן, כל זוג יפתח וישכלל את העיצוב כרצונו.

מומלץ, לפני שממשיכים לראות דוגמא של 3 המסכים בסוף מסמך זה: נספח 1 - דוגמאות למסכים

4ב. יצירת תת-ספרייה עבור מסכים של אותה ישות לוגית

נרצה ליצור תת-ספרייה עבור כל המסכים הקשורים לאותה ישות שירות לוגית (כך גם תעשו בעתיד, כשתוסיפו מסכים לישויות שירות לוגיות נוספות). בתפריט ההקשר של פרויקט **PL** (עמדו עם עכבר ימני על פרויקט **PL**)

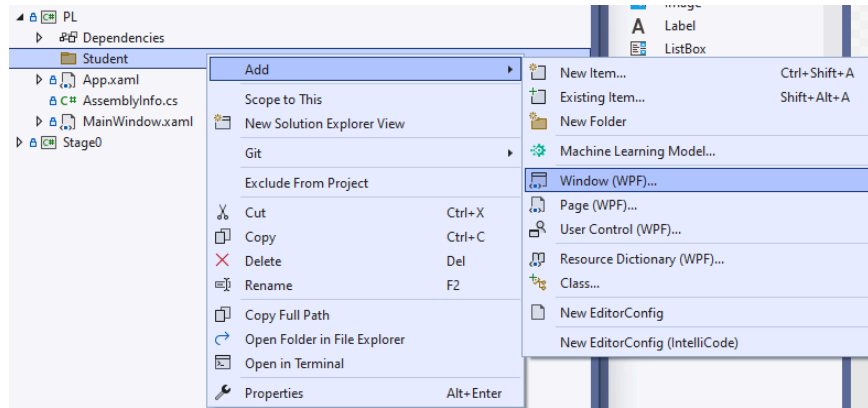
1. לחצו על **Add → New Folder** וקראו לספרייה בשם **ישות השירות הלוגית**.

a. למשל: Course

ג4. יצירת מסך להצגת רשימת הפריטים

להוספת מסך חדש עבור תצוגת רשימת הפריטים מסוג ישות השירות הלוגית:

1. בתפריט ההקשר של תת-הספרייה (עמדו עליה עם עכבר ימני), לחצו על: **Add→ Window(WPF)**



2. קראו למסך בשם מתאים

b. למשל: CourseListWindow

3. נתבונן למשל בקוד ה-aml שנוצר עבור מסך הצגת רשימה:

```
<Window x:Class="PL.Course.CourseListWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:local="clr-namespace:PL.Course"
    mc:Ignorable="d"
    Title="CourseListWindow" Height="450" Width="800">
    <Grid>

    </Grid>
</Window>
```

4. אפשר לראות שהמסך נוצר עם רכיב של תסדיר Grid בתוכו, בשלב זה הוא ריק.

ד4. יצירת מסך להצגת פריט בודד

להוספת מסך חדש עבור תצוגת פריט בודד מסוג ישות הנתונים הלוגית הרחבה הנבחרת:

1. בצעו את אותם צעדים 1-4, מהסעיף הקודם כדי ליצור מסך חדש נוסף, בשם מתאים, עבור תצוגת פריט בודד

a. למשל: CourseWindow

פרק 5 - מסך תצוגת הרשימה - עיצוב

בפרק זה נעסוק רק בעיצוב פקדים במסך תצוגת הרשימה. ללא מימוש הפונקציונאליות שלהם.

5א. מבנה מסך תצוגת הרשימה

כאמור, אסור לתת שם לפקדים ולכן המלצה חשובה: הוסיפו הערה ב XAML מעל פקדים משמעותיים על מנת שקוד ה XAML יהיה ברור ונדע מה תפקיד הפקד. לאחר שנסיים עם העיצוב, בפרקים הבאים נסביר כיצד לממש את הפונקציונאליות שלהם תוך שימוש מלא במנגנון ה- Data Binding.

הצעה לעיצוב מסך תצוגת הרשימה - יכיל פקד מסוג Grid (ה- Grid החיצוני), אשר מורכב מ 3 שורות:

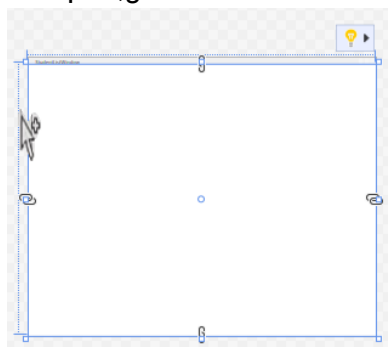
- שורה עליונה, פקד מסוג Grid (ה- Grid הפנימי), אשר תורכב מ 2 עמודות:
 - עמודה שמאלית, פקד מסוג Label - שיכיל את שם הקטגוריה שעל פיה נרצה לסנן את רשימת הפריטים שמוצגים במסך
 - עמודה ימנית, פקד מסוג ComboBox - שיכיל את האפשרויות לסינון הרשימה, ודרכו ניתן לבחור ערך לסינון הרשימה
- שורה אמצעית, רשימה מסוג ListView או DataGridView - שתציג את רשימת הפריטים בצורה מתקדמת - לחיצה על פריט ברשימה תאפשר לצפות/לעדכן פריט ברשימה.
- שורה תחתונה, פקד מסוג Button - לחיצה עליו תפתח מסך חדש שדרכו נוכל להוסיף פריט לרשימה

5ב. עיצוב ה Grid החיצוני במסך תצוגת הרשימה

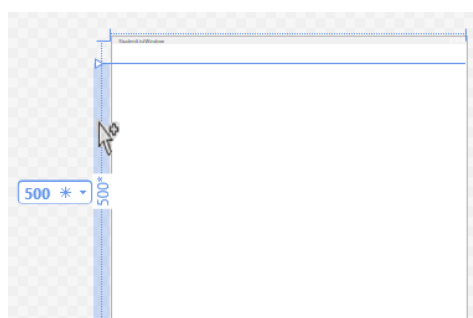
1. דרך ה-XAML שנו את גובהו של המסך בתוך התג <Window> ל- "Height=640".
2. זה יהיה ה Grid החיצוני, ויכיל בהמשך גריד פנימי. מומלץ להוסיף מעליו הערה שתסמן שהוא החיצוני
3. חלקו את ה Grid ל 3 שורות ביחס של 40:*500. תוכלו לעשות זאת באחת מ 2 הדרכים הבאות:
 - דרך 1: כתבו את השינויים ידנית ל-XAML בתוך התג <Grid>:

```
<!--External Grid-->
<Grid>
    <Grid.RowDefinitions>
        <RowDefinition Height="40*" />
        <RowDefinition Height="500*" />
        <RowDefinition Height="Auto" />
    </Grid.RowDefinitions>
</Grid>
```

- דרך 2: במסך הגרפי עברו עם העכבר בשוליו של ה grid, סמן העכבר ישתנה לחץ עם פלוס קטן



במידה ונלחץ מקש שמאלי על העכבר, קו זה יישאר ויהווה חלוקה של ה-grid:

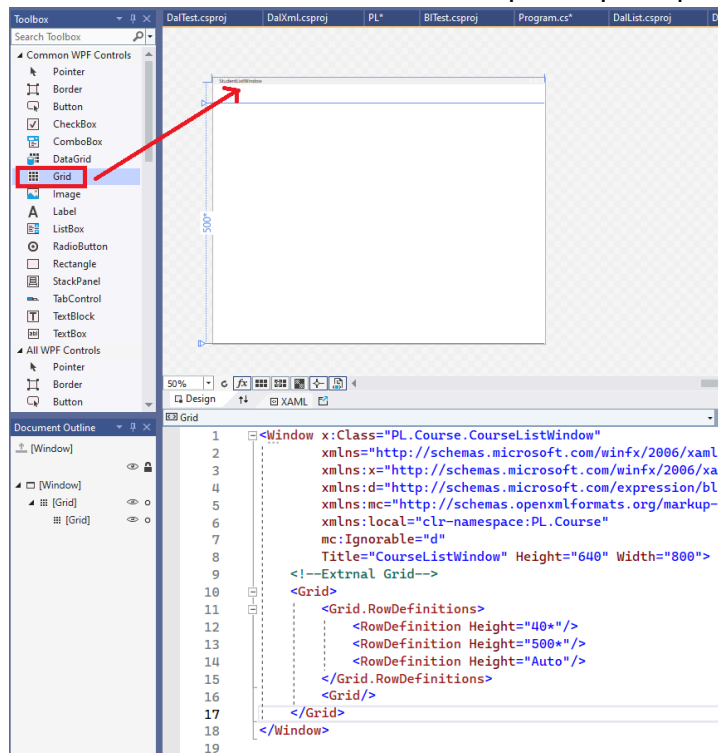


4. עיינו בקוד שהתקבל בחלונת ה XAML: קיבלתם 3 שורות של RowDefinition. מה מסמל הערך שנתנו לאטריבוט Height?

- מספרים בלבד - הערכים ישארו קבועים בשינוי גודל המסך
- מספרים עם כוכבית - ישמרו היחסים בשינוי גודל המסך
- כוכבית בלבד - ישמרו יחסים זהים בשינוי גודל המסך
- פירוש ערך Auto באטריבוטים של גובה השורה או רוחב העמודה של גריד - הערך ייקבע באופן אוטומטי בהתאם לגדלים של הפקדים שהשורה או העמודה יכולו בתוך תאים שלה (אם אין עדיין פקדים, השורה או העמודה לא תוצג, אך היא עדיין קיימת)

5. יצירת ה Grid הפנימי העליון במסך תצוגת הרשימה

1. צרו Grid פנימי שיופיע בתוך השורה העליונה של ה- Grid החיצוני הראשי, באופן הבא: גררו עם העכבר מחלונת (Toolbox) את הרכיב הגרפי Grid. גררו אותו לשורה העליונה של ה Grid החיצוני הראשי. זה ישרש להכלת החלק העליון של מסך תצוגת הרשימה.



2. כתוצאה מהגרירה, נוצר תג xml חדש ריק מסוג Grid שנראה כך: <Grid/>

3. החליפו תג זה כולו, בהגדרה הבאה:

```
<!--Internal, Upper Grid-->
<Grid HorizontalAlignment="Stretch" Height="auto" Grid.Row="0" VerticalAlignment="Stretch"
Width="auto">
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="*" />
        <ColumnDefinition Width="*" />
    </Grid.ColumnDefinitions>
</Grid>
```

הסבר: ה-grid הפנימי החדש ממוקם בשורה 0 של ה-grid החיצוני שבו הוא מוכל (הספירה מתחילה ב-0), הוא מתוח לרוחב ולגובה ומקבל את הגובה והרוחב שלו באופן אוטומטי. הוא מחולק ל 2 עמודות שוות רוחב (*).

5d. יצירת פקדים בתוך ה Grid הפנימי העליון

נרצה לאפשר סיוון בתצוגת הרשימה. הסיוון יבוצע ע"פ התכונה המוגדרת במסמך תיאור הכללי. לשם כך נשתמש בפקד ComboBox שילווה בפקד Label לצידו.

- ComboBox הוא רכיב גרפי שיאפשר למשתמש בחירה מתוך רשימה נפתחת.
 - למשל: רשימה של סמסטרים שלפיה נרצה לסנן את רשימת הקורסים שיוצגו במסך.
- ו- Label הוא רכיב גרפי שמשמש בתור תווית טקסט, בד"כ מתלווה לפקד נוסף כדי לתאר אותו.
 - למשל: תווית שבה יהיה כתוב: "Select Semester"

1. גררו עם העכבר, מחלונית ה- Toolbox, פקד מסוג Label, לתוך העמודה הראשונה (השמאלית, עמודה מספר 0) של הגריד הפנימי העליון. לחילופין, אפשר להוסיף פקד מסוג Label ע"י עריכת קוד ה-XAML. כתוצאה מהגרירה/עריכה ייוצר בתוך תג ה <Grid> הפנימי, תג xaml חדש מסוג <Label>.

2. גררו עם העכבר, מחלונית ה- Toolbox, פקד מסוג ComboBox לתוך העמודה הראשונה (השמאלית, עמודה מספר 0) של הגריד הפנימי העליון. לחילופין, אפשר להוסיף פקד מסוג ComboBox ע"י עריכת קוד ה-XAML. כתוצאה מהגרירה/עריכה ייוצר בתוך תג ה <Grid> הפנימי, תג xaml חדש מסוג <ComboBox>.

3. דרך ה XAML, ודאו שפקד ה Label נמצא תחת ה Grid הפנימי העליון וכן שהאטריבוט של העמודה שלו היא 0 (כלומר נמצא בעמודה השמאלית של הגריד הפנימי העליון).

```
Grid.Column="0"
```

4. דרך ה XAML, ודאו שפקד ה ComboBox נמצא תחת ה Grid הפנימי העליון וכן שהאטריבוט של העמודה שלו היא 1 (כלומר נמצא בעמודה הימנית של הגריד הפנימי העליון).

```
Grid.Column="1"
```

5. עבור פקד ה ComboBox:

a. שנו את האטריבוטים האלו באופן הבא:

```
HorizontalAlignment="Stretch" VerticalAlignment="Stretch"
```

b. הסירו את שאר האטריבוטים שמופיעים ב XAML עבור פקד ה ComboBox

5ה. הוספת פקד לתצוגת הרשימה, בחלק האמצעי של ה-Grid החיצוני

נרצה להוסיף, בחלק האמצעי של ה-Grid החיצוני, פקד שיכיל את רשימת הפריטים מסוג ישות העזר הלוגית. ויצג אותה בצורה גרפית, כטבלה מעוצבת עם עמודות ושורות בתבנית מתקדמת. לצורך כך ישנם פקדים שונים להצגת רשימה טבלאית בצורה מתקדמת כגון: ListView, DataGrid ועוד. פקדים אלו מבוססים על הפקד הפשוט יותר להצגת רשימה (ListBox), אבל מאפשרים להוסיף עיצוב מתוחכם יותר לפריטים.

1. גררו עם העכבר, מחלונית ה- Toolbox, את פקד הרשימה המתקדם שבחרתם לעבוד איתו. גררו אותו לתוך ה-Grid החיצוני, מתחת ל- Grid הפנימי העליון.

2. דרך ה XAML, שימו לב שיש להגדיר את השורה שבה הפקד יופיע: "Grid.Row=1", שורה זו מתייחסת לשורה בתוך ה- Grid החיצוני. כלומר פקד הרשימה יוצב בשורה השניה של הגריד החיצוני (פקד ה- Grid הפנימי העליון מוצב כבר בשורה הראשונה (מספר 0)).

3. אם בחרתם לעבוד עם ListView, אז דרך ה XAML, הסירו את האטריבוט שנוסף אוטומטית ל ListView (במידה והוא נוסף):

```
d:ItemsSource="{d:SampleData ItemCount=5}"
```

4. בין אם בחרתם לעבוד עם פקד DataGrid או ListView - עליכם לשלב בתוכו הגדרה של Template לחלק מהעמודות (כל שורה מייצגת אובייקט של ישות וכל עמודה מייצגת תכונה של ישות). למשל תכונה בוליאנית יש להציג עם פקד מסוג CheckBox, צבע טקסט שונה לתוכן של עמודה מסוימת שברצונכם להדגיש וכדומה.

a. ייתכן ותעדיפו לטפל בעניין ה Template בהמשך לאחר שכבר תחברו את הפקד לרשימה הלוגית ותטפלו באפשרות ש סיוון הרשימה. כאן: [8d. מסך תצוגת הרשימה - שכלול תצוגת הרשימה](#).

5ו. הוספת פקד Button לפתיחת מסך להוספת פריט לרשימה, בחלק התחתון של ה-Grid החיצוני

נרצה להוסיף, בחלק התחתון של ה-Grid החיצוני, פקד מסוג Button כך שלחיצה עליו תפתח מסך חדש שדרכו נוכל להוסיף פריט לרשימה.

1. יש ליצור את הכפתור בשורה השלישית "Grid.Row=2" של הגריד החיצוני. מכיוון שהשורה השלישית של הגריד החיצוני הוגדרה עם גובה Auto, אז קשה לגרור כפתור לתוכה דרך ה- Toolbox, לכן הוסיפו ישירות ב-xaml של מסך תצוגת הרשימה, מתחת לפקד הרשימה, את השורה הבאה:

```
<Button Content="Add" Grid.Row="2" HorizontalAlignment="Center" Height="50" Width="100"/>
```

פרק 6 - מסך ניהול ראשי - עיצוב ומימוש

כאמור, עם יצירת הפרויקט, נוצר לנו באופן אוטומטי מסך מספר 1, המסך הראשי בשם **MainWindow**. נשתמש במסך זה כמסך הניהול הראשי ונעצב אותו באופן זמני. בשלב הבא של הפרויקט נשכלל אותו.

הערה: בשלב זה (שלב 5), **מסך הניהול הראשי** ישמש כנקודת כניסה זמנית למערכת עבור משתמש בתפקיד "מנהל". אך בשלב הבא (שלב 6) תוסיפו **מסך כניסה למערכת** הכולל ת.ז ומכניס משתמש בתפקיד "מנהל" למסך הניהול הראשי ומשתמש בתפקיד "מתנדב" למסך מתנדב ראשי.

6א. מסך ניהול ראשי - עיצוב פקדים

1. הגדירו שורות ועמודות כרצונכם בתוך ה- Grid על מנת למקם את כל רכיבי התצוגה הנדרשים במסך זה. או אפילו כמה Grids פנימיים בתוך ה Grid הראשי. לשיקולכם.

2. כאמור, **אסור** לתת שם לפקדים ולכן המלצה חשובה: הוסיפו הערה ב XAML מעל פקדים משמעותיים על מנת שקוד ה XAML יהיה ברור ונדע מה תפקיד הפקד.

3. מקמו בתוך ה Grid שהגדרתם את הרכיבים הבאים:

- אזור "שעון המערכת":
 - Label - שיציג את ערך שעון המערכת הנוכחי
 - 5 כפתורים (Buttons) לקידום שעון המערכת: קידום בדקה, קידום בשעה, קידום ביום וקידום בשנה.
- אזור "משתני תצורה":
 - TextBox - שיציג ערך של משתנה תצורה, וגם יאפשר לערוך/לשנות אותו
 - Button - יציג כפתור שלחיצה עליו תעדכן את ערך משתנה התצורה בבסיס הנתונים
 - טיפול נפרד לכל משתנה תצורה, (רק כאלה שנקבעים בתצורה כגון: מספר שנים מקסימלי ללימוד סטודנט, ולא כאלו שנקבעים פנימית כגון: מספר רץ)
- Button - כפתור שלחיצה עליו תוביל לפתיחת מסך **תצוגת הרשימה**
- Button - כפתור שלחיצה עליו תוביל ל**אתחול (Initialize)** בסיס הנתונים (יצירת נתונים התחלתיים)
- Button - כפתור שלחיצה עליו תוביל ל**איפוס (ResetDB)** בסיס נתונים (ניקוי בסיס הנתונים)

6ב. מסך ניהול ראשי - יצירת אובייקט גישה לשכבת ה BL

1. מתוך מסך הניהול הראשי נרצה לגשת לגשת לישויות השירות הלוגיות "ניהול", לצורך כך נצטרך גישה ל BL. לכן, בקוד האחורי (code-behind) של המסך (class MainWindow), הוסיפו שדה פרטי סטטי בשם `s_bl` מטיפוס IBL שיאותחל באובייקט של מחלקה BL המממשת את הממשק IBL. בדיוק כמו ב `BITest`.

```
static readonly BIApi.IBl s_bl = BIApi.Factory.Get();
```

6ג. מסך ניהול ראשי - הגדרת הקשר נתונים (DataContext)

כאמור, אנו רוצים להימנע מלתת שם לפקדים בכל המסכים (ובפרט במסך ניהול ראשי). אם כך, כיצד ניגש לאטריבוטים של הפקדים דרך הקוד האחורי (code-behind)?

דרך ה-XAML נשתמש במנגנון ה- Data Binding, ע"מ לקשר את האטריבוטים של הפקדים לתכונות תלות (Dependency Property) שנגדיר בקוד האחורי של מחלקת המסך עצמו.

1. דרך קובץ ה- XAML של המסך, הגדירו את הקשר נתונים של המסך כולו להיות מקושר לתכונות שלו עצמו. הוסיפו לאלמנט **Window** אטריבוט בשם **DataContext** עם ערך **יוני** שמקשר אותו לעצמו.

```
<Window x:Class="PL.Course.CourseListWindow"
...
DataContext="{Binding RelativeSource={RelativeSource Mode=Self}}">
```

2. מעכשיו, בתוך ה XAML שך מסך הניהול, נשתמש במנגנון ה- Data Binding על מנת לקשר אטריבוטים של פקדים אל תכונות תלות מקבילה שנגדיר בקוד האחורי. נדגים זאת מיד.

ד6. מסך ניהול ראשי - קישור תצוגת שעון המערכת לתכונת תלות בקוד האחורי

1. בקוד האחורי של מסך ניהול הראשי, הגדירו תכונת תלות (Dependency Property), בשם שתבחרו, מסוג **DateTime** שתייצג את ערכו של התאריך המוצג על המסך. מומלץ להשתמש ב **propdp** בשם snippet.

למשל:

```
public DateTime CurrentTime
{
    get { return (DateTime)GetValue(CurrentTimeProperty); }
    set { SetValue(CurrentTimeProperty, value); }
}

public static readonly DependencyProperty CurrentTimeProperty =
    DependencyProperty.Register("CurrentTime", typeof(DateTime), typeof(MainWindow));
```

2. דרך ה-XAML, גשו לפקד מסוג Label שהגדרתם עבור הצגת ערכו הנוכחי של שעון המערכת. קבעו את האטריבוט **Content** שלו בעזרת מנגנון ה- Binding, וקשרו אותו לתכונת התלות מסוג DateTime שזה עתה הגדרתם בקוד האחורי.

למשל:

```
<Label Content="{Binding CurrentTime}" FontWeight="Bold" FontSize="20"/>
```

3. כעת, שינויים בקוד האחורי בתכונת שעון המערכת (CurrentTime), ישפיעו על תצוגת התאריך ב Label שעל המסך.

ה6. מסך ניהול ראשי - יצירת כפתורים לקידום שעון המערכת

1. דרך ה-XAML, גשו ל 5 הפקדים מסוג Button שהגדרתם עבור קידום שעון המערכת: קידום בדקה, קידום בשעה, קידום ביום וקידום בשנה.

2. עבור כל כפתור:

- a. ערכו את אטריבוט ה Content שלו בהתאם לייעוד שלו
- b. בקוד האחורי, הוסיפו מתודה שתירשם **לאירוע של לחיצה על הכפתור (Click)** לצורך הצגת מסך תצוגת הרשימה
- c. וודאו **שהמתודה רשומה כראוי לאירוע** הלחיצה (אפשר לראות את זה גם דרך ה XAML של כל אחד מהאלמנטים מסוג Button).
- d. תנו למתודת האירוע **שם משמעותי** (שינוי של שם של אירוע מחייב שינוי השם גם בנקודת הרישום לאירוע. טיפ: הדרך הפשוטה לבצע זאת היא לשנות את השם של המתודה בעזרת CTRL+R+R ואז שם המתודה מתעדכן בכל המקומות שמשתמשים בה באופן אוטומטי).
- e. ממשו את מתודת האירוע כך שתבצע קריאה למתודת קידום השעון של ממשק הניהול עם יחידת הזמן לקידום המתאימה

למשל ב XAML:

```
<Button Content="Add One Minute" Click="btnAddOneMinute_Click"/>
```

ובקוד האחורי:

```
private void btnAddOneMinute_Click(object sender, RoutedEventArgs e)
{
    s_b1.Admin.ForwardClock(BO.TimeUnit.MINUTE);
}
```

16. מסך ניהול ראשי - אזור משתני התצורה

עבור כל משתנה תצורה שנקבע בתצורה ואשר הכנתם עבורו זוג של TextBox ו-Button, בצעו את הפעולות הבאות:

1. קשרו את תצוגת ערכו של משתנה התצורה (TextBox) לתכונת תלות בקוד האחורי:

a. בקוד האחורי של מסך ניהול הראשי, הגדירו תכונת תלות (Dependency Property), בשם שתבחרו, שתייצג את ערכו המוצג על המסך.

i. טיפוס התכונה יהיה מאותו טיפוס שבו הוגדר משתנה התצורה במחלקת Config (למשל: טיפוס int עבור משתנה תצורה שמגדיר טווח מקסימלי בשנים)

ii. מומלץ להשתמש ב snippet בשם propdp.

b. דרך ה-XAML, גשו לפקד מסוג TextBox שהגדרתם עבור הצגת התוכן של אותו משתנה תצורה. קבעו את האטריבוט Text שלו בעזרת מנגנון ה-Binding, וקשרו אותו לתכונת התלות מהסוג שזה עתה הגדרתם בקוד האחורי.

c. כעת, שינויים בקוד האחורי בערכו של משתנה התצורה, ישפיעו על התצוגה שלו במסך. וכן להיפך.

2. ממשו את פעולת הלחיצה על הכפתור (Button). כך שתעדכן את ערכו של משתנה התצורה ב DAL:

a. ערכו את אטריבוט ה Content שלו בהתאם לייעוד שלו

b. בקוד האחורי, הוסיפו מתודה שתירשם לאירוע של לחיצה על הכפתור (Click) לצורך עדכון משתנה התצורה

c. ודאו שהמתודה רשומה כראוי לאירוע הלחיצה ותנו למתודת האירוע שם משמעותי

d. ממשו את מתודת האירוע כך שתבצע קריאה למתודת עדכון משתנה התצורה. למשל:

```
s_b1.Admin.SetMaxRange(MaxYearRange);
```

17. מסך ניהול ראשי - הגדרת משקיפים על שעון המערכת ומשתני התצורה

1. הגדירו 2 מתודות פרטיות למחלקת MainWindow ש"שקיפו" על שינויים שיעשו בערך השעון ובשאר משתני התצורה בבסיס הנתונים.

a. clockObserver - "מתודת השקפה על השעון" - שתזמן על ידי מחלקת BL.ObserverManager בכל פעם שערך שעון המערכת מתעדכן

i. בגוף המתודה בצעו השמה מחדש של הערך הנוכחי של שעון המערכת (תוך פניה לממשק Admin של ה BL) לתוך תכונת התלות מסוג DateTime בקוד האחורי.

```
CurrentTime = s_b1.Admin.GetClock();
```

b. configObserver - "מתודת ההשקפה על משתני התצורה" - שתזמן על ידי מחלקת BL.ObserverManager בכל פעם שערך של אחד ממשתני התצורה מתעדכן

i. בגוף המתודה בצעו השמה מחדש של ערכי משתני התצורה (תוך פניה לממשק Admin של ה BL) לתוך תכונות התלות המתאימות

```
MaxYearRange = s_b1.Admin.GetMaxRange();
```

18. מסך ניהול ראשי - אירוע טעינת המסך

1. בקוד האחורי של מסך ניהול הראשי, הוסיפו מתודה שתירשם לאירוע של טעינת המסך כולו (Loaded)

2. ודאו שהמתודה רשומה כראוי לאירוע הטעינה ותנו למתודה שם משמעותי

3. בקוד האחורי, ממשו את מתודת האירוע כך שבטעינת המסך תבצעו בה הפעולות הבאות: (שימו לב לבצע את כולן דרך ממשק Admin של ה BL)

a. השמה של הערך הנוכחי של שעון המערכת (תוך פניה לממשק Admin של ה BL) לתוך תכונת התלות מסוג DateTime בקוד האחורי.

```
CurrentTime = s_b1.Admin.GetClock();
```

b. עבור כל משתנה תצורה: השמה של ערך המשתנה (תוך פניה לממשק Admin של ה BL) לתוך תכונת התלות המתאימה לו בקוד האחורי.

c. דרך מתודת הממשק Admin.AddClockObserver הוסיפו את "מתודת ההשקפה על השעון" שהגדרתם בסעיף הקודם כמשקיפה.

```
s_b1.Admin.AddClockObserver(clockObserver);
```

d. דרך מתודת הממשק Admin.AddConfigObserver הוסיפו את "מתודת ההשקפה על משתני התצורה" שהגדרתם בסעיף הקודם כמשקיפה.

```
s_bl.Admin.AddConfigObserver(configObserver);
```

ט. מסך ניהול ראשי - אירוע סגירת המסך

1. בקוד האחורי של מסך הניהול הראשי, הוסיפו מתודה שתירשם לאירוע של סגירת המסך כולו (Window_Closed)
2. ודאו שהמתודה רשומה כראוי לאירוע הסגירה ותנו למתודה שם משמעותי (אפשר לראות את זה גם דרך ה XAML באלמנט Window).
3. בקוד האחורי, ממשו את מתודת האירוע כך שבסגירת המסך תבצעו בה הפעולות הבאות : (שימו לב לבצע את כולן תוך פניה לממשק Admin של ה BL)
 - a. דרך מתודת הממשק Admin.RemoveClockObserver הסירו את "מתודת ההשקפה על השעון" שהגדרתם בסעיף הקודם כך שלא תשקיף יותר.

```
s_bl.Admin.RemoveClockObserver(clockObserver);
```
 - b. דרך מתודת הממשק Admin.RemoveConfigObserver הסירו את "מתודת ההשקפה על התצורה" שהגדרתם בסעיף הקודם כך שלא תשקיף יותר.

```
s_bl.Admin.RemoveConfigObserver(configObserver);
```
4. הריצו את הפרויקט ובדקו את התוצאות המתקבלות.

י6. מסך ניהול ראשי - יצירת כפתור לצורך פתיחת מסך תצוגת הרשימה

1. דרך ה-XAML, גשו לפקד מסוג Button שהגדרתם עבור פתיחת מסך תצוגת הרשימה. ערכו את אטריבוט ה Content שלו בהתאם לייעוד שלו למשל:

```
Content="Handle Courses"
```
2. כאמור, אסור לתת שם לפקדים ולכן המלצה חשובה: הוסיפו הערה ב XAML מעל הפקד על מנת שיקוד ה XAML יהיה ברור ונדע מה תפקיד הפקד
3. בקוד האחורי, הוסיפו מתודה שתירשם לאירוע של לחיצה על הכפתור (Click) לצורך הצגת מסך תצוגת הרשימה.
4. ודאו שהמתודה רשומה כראוי לאירוע הלחיצה ותנו למתודת האירוע שם משמעותי
5. ממשו את מתודת האירוע כך שתבצע פתיחה של מסך חדש מסוג מסך תצוגת הרשימה. לאחר פתיחת מסך תצוגת הרשימה יש להמשיך אפשר גישה למסך הראשי (שימוש במתודה Show ולא במתודה ShowDialog).
- למשל:

```
private void btnCourses_Click(object sender, RoutedEventArgs e)
{
    new CourseListWindow().Show();
}
```
6. הריצו את הפרויקט ובדקו את התוצאות המתקבלות.

יא. מסך ניהול ראשי - יצירת כפתורים לאתחול ואיפוס בסיס הנתונים

- במסך הניהול הראשי , יצרתם כפתור אחד שלחיצה עליו תוביל לאתחול בסיס נתונים וכפתור שני שלחיצה עליו תוביל לאיפוס בסיס הנתונים.
- עבור כל אחד מהכפתורים שיצרתם:
1. דרך ה-XAML, גשו לפקד מסוג Button המתאים עבורו
 2. בקוד האחורי, הוסיפו מתודה שתירשם לאירוע של לחיצה על הכפתור (Click) לצורך אתחול או איפוס בהתאמה
 3. ודאו שהמתודה רשומה כראוי לאירוע הלחיצה ותנו למתודת האירוע שם משמעותי
 4. מימוש מתודת אירוע הלחיצה יכלול:
 - a. הודעה למשתמש (MessageBox) שתודא שאכן הוא מעוניין לבצע את האתחול/איפוס, ורק במידה והוא עונה "כן" יש להמשיך בביצוע המתודה. אחרת, אין לבצע דבר.
 - b. סגירת כל המסכים הפתוחים, חוץ מהמסך הראשי שישאר פתוח.
 - c. קריאה למתודה המתאימה בממשק הניהול של ה BL:

```
s_bl.Admin.InitializeDB();
s_bl.Admin.ResetDB();
```
 - d. במהלך הזמן שייקח אתחול/איפוס בסיס הנתונים יש לדאוג שהאייקון של העכבר ישתנה לשעון חול.

5. הריצו את הפרויקט ובדקו את התוצאות המתקבלות.

פרק 7 - מסך תצוגת הרשימה - הצגת פרטי הרשימה

כעת נחזור לטיפול בפונקציונאליות של מסך תצוגת הרשימה שכבר עיצבנו ב [פרק 5 - מסך תצוגת הרשימה - עיצוב](#).

מסך תצוגת הרשימה - מסך צריך להציג רשימה של כל הפריטים מסוג **ישות נתונים לוגית**. הפריטים שיוצגו ברשימה יכילו **מידע מצומצם** אודות **ישות נתונים לוגית רחבה אחרת**. בלחיצה על פריט ברשימה יוצג המידע המורחב יותר במסך **תצוגת פריט בודד**.

הרשימה תתמלא ע"י הפעלת שאילתא ב BL. כלומר, נרצה לבצע השמה של הרשימה לתוך האטריבוט **ItemsSource** של פקד ה **ListView** או פקד ה **DataGrid** (תלוי באיזה פקד מתקדם להצגת רשימה בחרתם ב- [4 - הוספת פקד לתצוגת הרשימה, בחלק האמצעי של ה-Grid החיצוני](#)).

7א. מסך תצוגת הרשימה - יצירת אובייקט גישה לשכבת ה BL

1. בקוד האחורי של המסך נצטרך גישה ל BL, על מנת לבקש ממנו את רשימת הישויות. לכן, הוסיפו בקוד האחורי שדה סטטי פרטי בשם **s_bl** מטיפוס **IBL** שיאוחל באובייקט של מחלקה **BL** המממשת את הממשק **IBL**. בדיוק כמו ב **BITest**.

```
static readonly BIApi.IBl s_bl = BIApi.Factory.Get();
```

7ב. מסך תצוגת הרשימה - הגדרת הקשר נתונים למסך והגדרת תלות עבור הרשימה

כאמור, אנו רוצים להימנע מלתת שם לפקדים (ובפרט לפקדי תצוגת הרשימה המתקדמים **ListView\DataGrid**), אזי לא נוכל לגשת לאטריבוט **ItemsSource** של פקד התצוגה דרך הקוד האחורי. בנוסף, נרצה שבכל פעם שהאוסף ישתנה אזי הרשימה תתעדכן גם בתצוגה. לכן:

2. דרך קובץ ה- XAML של המסך, הגדירו את הקשר נתונים של המסך כולו להיות מקושר לתכונות שלו עצמו. הוסיפו לאלמנט **Window** אטריבוט בשם **DataContext** עם ערך **on** שמקשר אותו לעצמו.

```
<Window x:Class="PL.Course.CourseListWindow"
    ...
    DataContext="{Binding RelativeSource={RelativeSource Mode=Self}}">
```

3. בקוד האחורי של המסך, הגדירו תכונת תלות (**Dependency Property**), בשם שתבחרו, מסוג **IEnumerable** של הישיות הלוגית המצומצמת. כדי שנוכל לטעון אליה את רשימת הפריטים הלוגית ולקשר אותה ל **ItemsSource** של ה- **ListView\DataGrid**. מומלץ להשתמש ב snippet בשם **propdp**.

למשל:

```
public IEnumerable<BO.CourseInList> CourseList
{
    get { return (IEnumerable<BO.CourseInList>)GetValue(CourseListProperty); }
    set { SetValue(CourseListProperty, value); }
}

public static readonly DependencyProperty CourseListProperty =
    DependencyProperty.Register("CourseList", typeof(IEnumerable<BO.CourseInList>),
    typeof(CourseListWindow), new PropertyMetadata(null));
```

4. דרך קובץ ה- XAML של המסך, הוסיפו לאלמנט **ListView\DataGrid** אטריבוט **ItemsSource** וקשרו אותו דרך מנגנון ה **Data Binding** לתכונת התלות כפי שהגדרתם זה עתה בקוד האחורי.

```
<ListView Margin="5" Grid.Row="1" ItemsSource="{Binding Path=CourseList}">
```

או:

```
<DataGrid Margin="5" Grid.Row="1" ItemsSource="{Binding Path=CourseList}">
```

מכיוון שהאטריבוט **Path** הוא ברירת המחדל של מנגנון ה- **Binding** אזי שורה זו מקבילה לשורה הבאה:

```
<ListView Margin="5" Grid.Row="1" ItemsSource="{Binding CourseList}">
```

5. אם תריצו את התוכנית, תגלו שעדיין הרשימה ריקה בתצוגה, כיוון שעדיין לא הרצנו בקוד האחורי את השאילתא שטוענת את הרשימה לתוך תכונת התלות **CourseList**. בינתיים המשיכו לפרק הבא.

פרק 8 - מסך תצוגת הרשימה - סינון הרשימה

נמשיך בטיפול בפונקציונליות של מסך תצוגת הרשימה וכעת נרצה להשתמש בערכי רשימת הפריטים שמופיעים ב `ComboBox` לצורך סינון הרשימה המוצגת במסך.

ראשית נמלא את ה `ComboBox` בערכים אפשריים על פי הקריטריון הנדרש לסינון, המוגדר במסמך התיאור הכללי של הפרויקט. ולאחר מכן, נגיב לאירוע בחירת הערך ב `ComboBox` ונציג ברשימת הפריטים שבמסך רק את הפריטים השייכים לקטגוריה הנבחרת ב `ComboBox`.

הערכים שבהם נרצה למלא את ה- `ComboBox`, מוגדרים כ `enum` כלשהו ב `BL`.

למשל, אם זה ה- `enum` שהוגדר ב `BO`:

```
public enum SemesterNames
{
    WinterA,
    SpringB,
    Year,
    Summer,
    Elul,
    None
}
```

אז נרצה להציג ב `ComboBox` את רשימת הסמסטרים האפשריים, ובכל פעם שנבחר ערך אחר ב `ComboBox` נסנן את רשימת הקורסים המוצגת בהתאם לסמסטר הנבחר.

8א. מסך תצוגת הרשימה - מילוי ה- `ItemsSource` של ה- `ComboBox` בערכים

מכיוון שאנו רוצים להימנע מלתת שם לפקדים (ובפרט ל `ComboBox`), אזי לא נוכל לגשת לאטריבוט `ItemsSource` של ה `ComboBox` דרך הקוד האחורי.

לכן, דרך ה-`XAML` נשתמש באטריבוט `Source` של מנגנון ה- `Data Binding`, ע"מ לקשר את האטריבוט `ItemsSource` של הפקד `ComboBox` למשאב סטטי שיקושר לאוסף הלוגי של ערכי ה `Enums`. את המשאב הסטטי נגדיר ב `XAML` ואת האוסף הלוגי נגדיר כטיפוס חדש בקוד האחורי.

1. בפרויקט `PL`, הוסיפו קובץ קוד חדש בשם `Enums.cs`
2. הגדירו בתוכו טיפוס חדש מסוג אוסף שמחזיר את כל רשימת הפריטים ב `Enum` שהגדרנו (הגדירו מחלקה חדשה מסוג `IEnumerable` שמחזירה `IEnumerator`)

למשל עבור `enum` מסוג `BO.SemesterNames`, נגדיר ב `PL` בתוך הקובץ `Enum.cs` את האוסף הלוגי

כך:

```
namespace PL;
internal class SemestersCollection : IEnumerable
{
    static readonly IEnumerable<BO.SemesterNames> s_enums =
        (Enum.GetValues(typeof(BO.SemesterNames)) as IEnumerable<BO.SemesterNames>)!;

    public IEnumerator GetEnumerator() => s_enums.GetEnumerator();
}
```

3. בקובץ המשאבים של הפרויקט `App.xaml` (אפשר גם בקובץ ה `XAML` של המסך עצמו), הגדירו משאב סטטי עם מפתח `x:Key` בשם שתבחרו, וקשרו אותו לאובייקט האוסף הלוגי המקומי (`local`) שהגדרתם בסעיף הקודם.

למשל:

```
<Application x:Class="PL.App"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="clr-namespace:PL"
    StartupUri="MainWindow.xaml">
    <Application.Resources>
        <local:SemestersCollection x:Key="SemestersCollectionKey" />
    </Application.Resources>
</Application>
```

4. דרך קובץ ה- `XAML` של המסך, הוסיפו לפקד ה `ComboBox`, אטריבוט בשם `ItemsSource` שמגדיר את מקור נתונים שלו. קשרו אותו דרך האטריבוט `Source` של מנגנון ה `Data Binding` למשאב הסטטי עם המפתח שהגדרתם עבורו.

למשל:

```
<ComboBox ItemsSource="{Binding Source={StaticResource SemestersCollectionKey}}"
    Grid.Column="1" HorizontalAlignment="Stretch" VerticalAlignment="Stretch"/>
```

5. הריצו את הפרויקט ובדקו את התוצאות המתקבלות. בדקו שה ComboBox התמלא בערכי ה ENUM.

8.ב. מסך תצוגת הרשימה - סינון רשימת הפריטים בזמן אירוע SelectionChanged של ה ComboBox

כדי לסנן את הרשימה המוצגת במסך ע"פ הערך הנבחר ב- ComboBox, נקשר את הערך הנבחר בתצוגה לתכונה רגילה בקוד המסך האחורי. ונגיב לאירוע השינוי ע"י ביצוע השאילתא מחדש ע"פ הקריטריון החדש לסינון.

מכיוון שאנו רוצים להימנע מלתת שם לפקדים (ובפרט ל ComboBox), אזי לא נוכל לגשת לאטריבוט SelectedValue של ה ComboBox דרך הקוד האחורי. בנוסף, נרצה שבכל פעם שהבחירה ב- ComboBox תשתנה אזי נבצע את השאילתא מחדש ע"פ הקריטריון החדש לסינון.

לכן, דרך ה-XAML נשתמש במנגנון ה- Data Binding, ע"מ לקשר את האטריבוט SelectedValue של הפקד ComboBox לתכונה רגילה שנגדיר בקוד האחורי של המסך. ערכה של תכונה זו יתעדכן בכל פעם שהערך הנבחר ב- ComboBox ישתנה.

1. בסעיפים הקודמים, הגדרתם דרך קובץ ה- XAML של המסך, את הקשר הנתונים של המסך כולו להיות מקושר לתכונות שלו עצמו. הוספתם לאלמנט Window אטריבוט DataContext עם ערך יחסי (RelativeSource) שמקשר אותו לעצמו (Self).

2. בקוד האחורי של המסך, הגדירו תכונה רגילה (כלומר לא Dependency Property), בשם שתבחרו, מסוג enum שמוגדר ב BO, שעל פיו יש לסנן את התכונה הזו נקשר לפריט הנבחר ב ComboBox. התכונה הזו תתעדכן רק מכיוון התצוגה לכיוון הקוד האחורי ולא להיפך, ולכן היא יכולה להיות תכונה רגילה ולא תכונת תלות.

a. קבעו ערך התחלתי לתכונה, ע"פ הערך שבו תרצו שהרשימה תהיה מסוננת על פיו בעליית המסך.

i. לרוב נרצה שבעליית המסך הרשימה תוצג במלואה וללא סינון, לכן כדאי להוסיף ל enum, ערך נוסף שמייצג "קטגוריה כללית" כגון None או All.

למשל:

```
public BO.SemesterNames Semester { get; set; } = BO.SemesterNames.None;
```

3. דרך קובץ ה- XAML של המסך, הוסיפו לפקד ה ComboBox, אטריבוט בשם SelectedValue שמייצג את הערך הנבחר בו. קשרו אותו דרך אטריבוט Path של מנגנון ה Data Binding לתכונה מסוג ה- enum שהגדרתם בסעיף הקודם. עם Mode=TwoWay.

הערה: בגלל שהגדרנו בקוד האחורי את התכונה המקושרת כתכונה רגילה ולא כתכונת תלות, אזי היא לא תדווח על שינויים לתצוגה. כיוון השינוי יהיה מכיוון התצוגה, בעזרת מנגנון ה Binding שיעדכן את התכונה בקוד בכל פעם שהבחירה בתצוגה תשתנה. הפעם היחידה שהתכונה תקבל ערך מכיוון הקוד לכיוון התצוגה תהיה באתחול המסך ולכן הגדרנו בכל זאת Mode=TwoWay. אם נשנה את ערך התכונה דרך הקוד זה לא ישפיע על התצוגה, על מנת שזה ישפיע נצטרך להגדיר את התכונה כתכונת תלות, ולא מספיק ה Mode=TwoWay.

למשל:

```
<ComboBox Grid.Column="1" SelectedValue="{Binding Path=Semester, Mode=TwoWay}"
ItemsSource="{Binding Source={StaticResource SemestersCollectionKey}}"/>
```

ושוב, אפשר גם בלי ה Path:

```
<ComboBox Grid.Column="1" SelectedValue="{Binding Semester, Mode=TwoWay}"
ItemsSource="{Binding Source={StaticResource SemestersCollectionKey}}"/>
```

4. לאחר שקשרנו את התכונה הזו לערך הנבחר ב ComboBox, נרצה להגיב לאירוע שינוי הערך ולגרום לרשימת הפריטים להתעדכן בתצוגה. השתמשו באירוע ברירת מחדל של לחיצה על ה ComboBox האירוע נקרא SelectionChanged.

a. תנו למתודת האירוע שם משמעותי

b. במימוש מתודת האירוע ה:

i. ערכו של הקריטריון לסינון מתעדכן אוטומטית לתוך תכונת ה- enum שהגדרתם בקוד האחורי, ע"פ הבחירה ב ComboBox בתצוגה. ואין צורך להוסיף דבר מעבר.

ii. ע"פ ערך הקריטריון לסינון שיתעדכן אוטומטית, יש להשתמש בקריאה למתודה של שכבת ה-BL המחזירה רשימה מסוננת ולעדכן את תכונת התלות מסוג

IEnumerable שהגדרתם בקוד האחורי של המסך. עדכון זה, יגרום אוטומטית לעדכון הרשימה בתצוגה.

למשל:

```
CourseList = (Semester == BO.SemesterNames.None) ?  
s_b1?.Course.ReadAll()! : s_b1?.Course.ReadAll(null, BO.CourseFieldFilter.SemesterName, Semester)!;
```

8g. מסך תצוגת הרשימה - הגדרת משקיפים על רשימת הפריטים בבסיס הנתונים

1. בקוד האחורי של מסך תצוגת הרשימה, הגדירו "מתודת השקפה על הרשימה" - מתודה פרטית ש"תשקיף" על שינויים ברשימת הפריטים בבסיס הנתונים.

a. המתודה לא תזומן ב PL אלא תזומן על ידי מחלקת המימוש ב BL של אותה ישות בכל פעם שהרשימה תתעדכן בבסיס נתונים (הוספה/עדכון/מחיקה של פריט ברשימה). כפי שמימשנו בפרק 1 של שלב זה.

b. במימוש המתודה הפעילו את אותה שאלתא מהסעיף הקודם שמחזירה מחדש את הרשימה המסוננת. (קוד שחוזר על עצמו? אולי כדאי לעטוף במתודת עזר פרטית?)

2. בקוד האחורי של מסך תצוגת הרשימה, הרשמו לאירוע טעינת המסך תצוגת הרשימה (Loaded), ובמימוש מתודת האירוע הוסיפו את מתודת ההשקפה מהסעיף הקודם כמשקיפה על שינויים ב BL.

3. בקוד האחורי של מסך תצוגת הרשימה, הרשמו לאירוע סגירת המסך תצוגת הרשימה (Closed), ובמימוש מתודת האירוע הוסיפו את מתודת ההשקפה מהסעיף הקודם מהיותה משקיפה על שינויים ב BL.

למשל:

```
private void queryCourseList()  
=> CourseList = (Semester == BO.SemesterNames.None) ?  
s_b1?.Course.ReadAll()! : s_b1?.Course.ReadAll(null, BO.CourseFieldFilter.SemesterName,  
Semester)!;
```

```
private void courseListObserver()  
=> queryCourseList();
```

```
private void Window_Loaded(object sender, RoutedEventArgs e)  
=> s_b1.Course.AddObserver(courseListObserver);
```

```
private void Window_Closed(object sender, EventArgs e)  
=> s_b1.Course.RemoveObserver(courseListObserver);
```

5. הריצו את התוכנית ופתחו את מסך תצוגת הרשימה. שנו את הבחירה ב- ComboBox ובדקו שהרשימה מתעדכנת על פי הסינון.

6. אם עדיין השתמשותם ב ListView\DataGrid בצורה פשוטה אז התצוגה המתקבלת בתוך ה-ListView\DataGrid היא בעצם ה ToString של הישות. בפרק הבא נדבר על שכלול תצוגת הרשימה.

8d. מסך תצוגת הרשימה - שכלול תצוגת הרשימה

בין אם בחרתם לעבוד עם פקד DataGrid או ListView - עליכם לשלב בתוכו הגדרה של Template לחלק מהעמודות (כל שורה מייצגת אובייקט של ישות וכל עמודה מייצגת תכונה של ישות).

למשל תכונה בוליאנית יש להציג עם פקד מסוג CheckBox, צבע טקסט שונה לתוכן של עמודה מסוימת שברצונכם להדגיש וכדומה.

במצב כזה יש לקשר כל עמודה בנפרד לתכונה המקבילה לה בישות הנתונים הלוגית המוצגת ברשימה דרך מנגנון ה- Binding.

למשל, אם בחרתם בפקד ה- DataGrid להצגת הרשימה:

```
<DataGrid ItemsSource="{Binding Path=StudentList}" IsReadOnly="True" AutoGenerateColumns="False"  
EnableRowVirtualization="True" RowDetailsVisibilityMode="VisibleWhenSelected"  
MouseDoubleClick="dgStudentList_MouseDoubleClick" SelectedItem="{Binding SelectedStudent}">  
<DataGrid.Columns>  
    <DataGridTextColumn Binding="{Binding Id}" Header="Id" Width="Auto"/>  
    <DataGridTextColumn Binding="{Binding Name}" Header="Name" Width="Auto"/>  
    ...  
    <DataGridTemplateColumn Header="Is Active" Width="Auto">  
        <DataGridTemplateColumn.CellTemplate>  
            <DataTemplate>
```



```

<CheckBox IsEnabled="False" IsChecked="{Binding Path=IsActive}" />
</DataTemplate>
</DataGridTemplateColumn.CellTemplate>
</DataGridTemplateColumn>
...
</DataGrid.Columns>
</DataGrid>

```

פרק 9 - מסך תצוגת פריט בודד

כעת, ניצור מסך שמייצג פריט בודד על כל תכונותיו, מסוג ישות הנתונים הלוגית הרחבה הנבחרת. למשל: BO.Course.

עבור כל תכונה של הישות הלוגית יוצגו זוג פקדים במסך - תיאור וערך. ערכי הפקדים יהיו מקושרים ב-XAML, דרך מנגנון ה Binding, לשדות של אובייקט מסוג ישות הנתונים הלוגית שיאותחל בקוד האחורי.

המסך יפתח ב 2 מצבים:

- **מצב הוספה** - כל השדות של האובייקט יהיו ללא ערך או עם ערך ברירת מחדל. המשתמש ימלא דרך התצוגה את הערכים ולאחר מכן ילחץ על כפתור Add שיוסיף את האובייקט החדש לבסיס הנתונים בעזרת שכבת ה-BL.
- **מצב עדכון** - כל השדות של האובייקט יהיו מלאים בערכים של אובייקט קיים. המשתמש יעדכן דרך התצוגה את הערכים שהוא מעוניין לעדכן ולאחר מכן ילחץ על כפתור Update שיעדכן את האובייקט הקיים בבסיס הנתונים בעזרת שכבת ה-BL.

תזכורת: אין לתת שם לפקדי תצוגה, אלא לעבוד עם Data Binding מלא.

9א. מסך תצוגת פריט בודד - עיצוב פקדים

המסך אמור לייצג אובייקט בודד מסוג ישות הנתונים הלוגית הרחבה הנבחרת.

1. עבור כל תכונה של הישות הלוגית גררו מה Toolbox אל תוך ה Grid, זוג פקדים במסך:
 - מימין - תיאור התכונה (Label)
 - משמאל - ערך התכונה (בחרו פקד מסוג TextBox, ComboBox, CheckBox וכו' לפי סוג הערך)
 - לכל פקד כזה יש אטריבוט שמייצג את הערך - את האטריבוטים הללו נקשר דרך מנגנון ה Binding לשדות של אובייקט מטיפוס הישות הלוגית שיאותחל בקוד האחורי - נרחיב מיד כיצד לעשות זאת.
2. עליכם לדאוג שכל טור של פקדים יהיו מיושרים לאותה נקודה ובאותו הגודל. חשבו איך לעשות זאת
 - ניתן לקבוע את האטריבוטים האלו עבור כל פקד, אך מומלץ להשתמש לחילופין בפקודות XAML מתקדמות כמו Style, Resources כדי לקבוע את העיצובים בצורה גורפת
 - עליכם לקחת בחשבון את העיקרון "אל תחזרו על עצמך" (DRY - Don't Repeat Yourself) על מנת להקטין עד כמה שאפשר את code smell של "חזרות לא הכרחיות" (Needless Repetition)
 - ייתכן ותעדיפו לעשות את זה בשלב 6 ולא כעת
 - בנוסף, ייתכן ושימוש ב Grid פנימי יכול לסייע, אך לא חובה
3. שימו לב להתאים בין סוג הפקד לסוג התכונה, למשל:
 - לתכונה מסוג מחרוזת יתאים פקד מסוג TextBox
 - לתכונה מסוג Enum יתאים פקד מסוג ComboBox
 - לתכונה מסוג בוליאני יתאים פקד מסוג CheckBox
 - לתכונה מסוג DateTime יתאים פקד מסוג DatePicker
4. צרו פקד מסוג Button בתחתית המסך, בתוך ה Grid. כפתור יחיד זה ישמש גם לביצוע פעולת ההוספה וגם לביצוע פעולת העדכון. לאחר שהמשתמש ימלא/יעדכן את כל הערכים של הפקדים/האובייקט, הוא ילחץ על הכפתור להוספה של ישות חדשה או עדכון של ישות קיימת.
5. עליכם לדאוג לכך שהטקסט שיופיע על כפתור הוספה/עדכון ייקבע מתוך הבנאי של המסך בהתאם למצב שבו המסך נפתח. הוספה/עדכון:
 - דרך ה XAML, תצטרכו לקבוע את ערכו של אטריבוט ה Content של הכפתור בעזרת מנגנון ה Binding ולקשר אותו לתכונת תלות (נניח בשם ButtonText) מטיפוס מחרוזת שתגדירו בקוד האחורי של המסך.

```

<Button Content="{Binding ButtonText}" Click="btnAddUpdate_Click"/>

```

- מתוך הבנאי של המסך (עוד לפני הקריאה ל `InitializeComponent`), קבעו את הטקסט של תכונת התלות על פי המצב שבו נפתח המסך הוספה/עדכון.
`ButtonText = id == 0 ? "Add" : "Update";`

9. מסך תצוגת פריט בודד - קישור המסך לאובייקט

המסך מייצג פריט בודד מסוג ישות הנתונים הלוגית הרחבה הנבחרת (למשל: BO.Course). נרצה שבעליית המסך כל הפקדים ב-XAML המייצגים תכונות של הישות, יהיו מקושרים לאובייקט קיים מסוג אותה ישות:

- אם המסך במצב **הוספה** אז נרצה שהם יהיו מקושרים לאובייקט **חדש שניצר** בעל ערכים ריקים/ברירת מחדל
 - אם המסך במצב **עדכון** אז נרצה שהם יהיה מקושרים לאובייקט קיים **שנאחזר מה BL**, עם ערכים מלאים של אותו פריט לעדכון.
1. בקוד האחורי, צרו שדה פרטי שיאפשר גישה ל BL, כמו שעשיתם במסך הקודם. על מנת שתוכלו לבקש ממנו את האובייקט הרצוי.
 2. דרך קובץ ה-XAML של המסך, הגדירו את הקשר נתונים של המסך כולו להיות מקושר לתכונות שלו עצמו. הוסיפו לאלמנט **Window** אטריבוט בשם **DataContext** עם ערך **יוני** שמקשר אותו לעצמו.

```
<Window x:Class="PL.Course.CourseListWindow"
        ...
        DataContext="{Binding RelativeSource={RelativeSource Mode=Self}}">
```

3. בקוד האחורי של המסך, הגדירו תכונת תלות (Dependency Property), בשם שתבחרו, מסוג ישות הנתונים הלוגית. כדי שנוכל לטעון אליה את האובייקט המתאים מה BL. מומלץ להשתמש ב snippet בשם propdp.

```
public BO.Course? CurrentCourse
{
    get { return (BO.Course?)GetValue(CurrentCourseProperty); }
    set { SetValue(CurrentCourseProperty, value); }
}

public static readonly DependencyProperty CurrentCourseProperty =
    DependencyProperty.Register("CurrentCourse", typeof(BO.Course), typeof(CourseWindow), new
    PropertyMetadata(null));
```

4. בקוד האחורי של המסך, הוסיפו לבנאי פרמטר מסוג **int** עם ערך ברירת מחדל 0, שמייצג את ה **Id של הישות שהמסך יציג את תכונותיה**. בפרק הבא, נממש את הקוד שיוצר את המסך מתוך מסך תצוגת הרשימה, ודרך הפרמטר הזה נעביר מידע למסך הנוכחי, האם מדובר במצב של עדכון ישות קיימת או יצירת ישות חדשה.

5. בקוד האחורי של המסך, הוסיפו בגוף הבנאי, לאחר **InitializeComponent**, פקודות שיבצעו השמה של אובייקט מסוג הישות הלוגית לתוך תכונת התלות, בשם שבחרתם. על פי הפרמטר **Id** שהתקבל בבנאי של המסך:

- a. אם מגיע **Id** עם ערך ברירת מחדל - אז יש לייצר אובייקט חדש (new), עם ערכים ריקים או ערכי ברירת מחדל, לכל תכונה של האובייקט.
- b. אחרת - יש לקרוא למתודת ה **BL** המתאימה, על פי ה **Id** כדי לקבל את האובייקט הקיים מה **Dal** (זכרו לתפוס חריגות).

למשל:

```
CurrentStudent = (id != 0) ? s_bl.Student.Read(id)! : new BO.Student() { Id = 0, CurrentYear = BO.Year.None, RegistrationDate = s_bl.Admin.GetClock() };
```

6. נרצה לקשר את האטריבוטים המתאימים של כל פקד ופקד במסך לתכונה המתאימה לו באובייקט שבקוד האחורי, וכל זאת מבלי לתת שמות לפקדים. בנוסף, נרצה שבכל פעם שהערך של הפקד משתנה בתצוגה אזי התכונה המקושרת אליו באובייקט תשתנה גם. דרך קובץ ה-XAML של המסך, עבור כל פקד שמייצג ערך, קשרו דרך מנגנון ה- **Binding** את האטריבוט הרלוונטי לשדה המתאים של אובייקט התלות שהגדרתם בקוד האחורי.

למשל, עבור שדה מסוג מחרוזת המיוצג במסך עם **TextBox**:

```
<TextBox Grid.Row="2" Grid.Column="1"
          Text="{Binding CurrentCourse.CourseName, Mode=TwoWay, NotifyOnValidationError=true,
          ValidatesOnExceptions=true}" HorizontalAlignment="Left" Height="NaN" Margin="3"
          VerticalAlignment="Center" Width="120"/>
```

ולמשל, עבור שדה מסוג **Enum** המיוצג במסך עם **ComboBox**:

```
<ComboBox Grid.Row="3" Grid.Column="1" HorizontalAlignment="Left" Height="NaN" Margin="3"
            ItemsSource="{Binding Source={StaticResource YearsCollectionKey}}"
            SelectedValue="{Binding CurrentCourse.InYear, Mode=TwoWay, NotifyOnValidationError=true,
            ValidatesOnExceptions=true}" VerticalAlignment="Center" Width="120"/>
```

שימו לב שגם במסך זה, עליכם למלא את רשימת האיברים ב- ComboBox בעזרת מנגנון ה-Binding, כפי שלמדתם לעשות במסך הקודם וללא ניתנת שם לפקדים.

9g. מסך תצוגת פריט בודד - אטריבוט IsReadOnly ואטריבוט Visibility ב 2 מצבי המסך

הערה: סעיף זה דורש שימוש ב Converter, נושא טיפה מתקדם. מי שזה מכביד עליו יכול לדחות אותו לשלב 6.

הסבר להגדרה ושימוש ב Converter ניתן לראות במסמך זה ב: [נספח 2 - דוגמא להגדרה ושימוש ב Converter](#).

מכיוון שהמסך מיועד לטיפול ב 2 מצבים שלא מקיימים במקביל, אלא פעם המסך יפתח במצב הוספה ופעם במצב עדכון - יש לטפל בהבדלים בין 2 המצבים מבחינת תצוגת הפקדים. התכונה Id של האובייקט שהגדרנו בתכונת התלות של המסך, היא זו שקובעת את המצב, אם היא עם ערך ברירת מחדל אז אנחנו במצב הוספה, אחרת - מצב עדכון.

1. אם יש תכונות של הישות שאסור לעדכן אותן במצב "עדכון", אך בכל זאת אנו רוצים שיופיעו על המסך, אזי יש לדאוג שכבר בעליית המסך הפקדים המתאימים שלהן יהיו "נראים" אך "לא מאופשרים" `IsReadOnly="True"`. לדוגמא תכונת ה `Id`:

a. במידה והתכונה Id של הישות היא "מספר רץ" אז הפקד שמקושר לתכונה Id של האובייקט - צריך להיות "נראה" אך "לא מאופשר" ב 2 המצבים, מכיוון ש:

i. במצב עדכון - האובייקט מגיע עם Id קיים ואסור לשנותו

ii. במצב הוספה - ה Id מגיע עם ערך ברירת מחדל, ומקבל ערך באופן אוטומטי רק לאחר הוספת האובייקט החדש בפועל ל BL. ולא מקבל ערך מהמשתמש.

2. מכיוון שאין לתת שם פקדים אזי דרך ה-XAML, יש לקבוע את האטריבוט `IsReadOnly`, פעם אחת בעליית המסך, בעזרת מנגנון ה-Binding, ולקשר אותו לתכונה Id של האובייקט. ומכיוון שהן אינן מאותו סוג יש להגדיר גם Converter מתאים שממיר ערכים בהתאמה.

3. כנ"ל לגבי תכונות שצריכות לא להיראות בכלל במצב מסוים (הוספה או עדכון).

למשל:

```
<TextBox Grid.Row="0" Grid.Column="1" HorizontalAlignment="Left" Height="NaN" Margin="3"
Text="{Binding CurrentStudent.Id, Mode=TwoWay, NotifyOnValidationError=true,
ValidatesOnExceptions=true}"
IsReadOnly="{Binding ButtonText, Converter={StaticResource ConvertUpdateToTrueKey}}"
VerticalAlignment="Center" Width="120"/>

<Label Grid.Row="5" Grid.Column="0" Content="Registration Date:"
Visibility="{Binding ButtonText, Converter={StaticResource ConvertUpdateToVisibleKey}}"
HorizontalAlignment="Left" Margin="3" VerticalAlignment="Center"/>

<DatePicker IsEnabled="false" Grid.Row="5" Grid.Column="1"
Visibility="{Binding ButtonText, Converter={StaticResource ConvertUpdateToVisibleKey}}"
HorizontalAlignment="Left" Height="NaN" Margin="3" SelectedDate="{Binding
CurrentStudent.RegistrationDate, Mode=TwoWay, NotifyOnValidationError=true,
ValidatesOnExceptions=true}" VerticalAlignment="Center" Width="120"/>
```

9d. מסך תצוגת פריט בודד - לחיצה על כפתור הוספה/עדכון לצורך מימוש פעולות הוספה/עדכון

לאחר שהמשתמש ימלא/יעדכן את כל הערכים של הפקדים/האובייקט, הוא ילחץ על הכפתור להוספה של ישות חדשה או עדכון של ישות קיימת.

1. הוסיפו לכפתור אירוע ברירת מחדל של לחיצה על הכפתור (Click) וממשו את מתודת האירוע כך:

a. אפשר להבדיל בין המצבים לפי ערכה של תכונת התלות שמייצגת את הטקסט שעל כפתור ההוספה/עדכון.

```
if (ButtonText == "Add") ...
```

b. בהתאם למצב המסך(הוספה/עדכון), בצעו קריאה למתודה המתאימה ב BL שמקבלת כפרמטר את האובייקט מסוג הישות (תכונת תלות).

```
s_b1.Student.Create(CurrentStudent!);
IX
s_b1.Student.Update(CurrentStudent!);
```

c. לאחר ביצוע פעולת הוספה/עדכון, יש לשלוח הודעת הצלחה מתאימה למשתמש ולסגור את מסך תצוגת פריט בודד.

d. במידה ונזרקה חריגת BL, יש לתפוס אותה ולשלוח בעקבותיה MessageBox מתאים.

e. הריצו את התוכנית ובדקו שאכן לאחר כל פעולה, הרשימה במסך תצוגת הרשימה התעדכנה באופן אוטומטי.

9ה. מסך תצוגת פריט בודד - הגדרת משקיפים על הפריט הבודד

כמובן שכל מסך שמעתה תוסיפו לפרויקט, יש להגדיר אותו כמשקיף על התוכן שהוא מציג. באותו אופן שלימדנו אתכם להגדיר משקיפים במסכים הקודמים. זה ברור, אך בכל זאת נציין.

1. יש להגדיר מתודת השקפה שתמלא את הפריט מחדש

```
int id = CurrentStudent!.Id
CurrentStudent = null;
CurrentStudent = s_bl.Student.Read(id);
```

2. יש להירשם לאירוע טעינת המסך ולהוסיף את מתודת ההשקפה כמשקיפה על פריט בודד

```
if (CurrentStudent!.Id != 0)
    s_bl.Student.AddObserver(CurrentStudent!.Id, studentObserver);
```

3. יש להירשם לאירוע סגירת המסך ולהסיר את מתודת ההשקפה כמשקיפה על פריט בודד

4. תוכלו לוודא שהגדרתם את ההשקפה נכון, אם תפתחו את מסך פריט בודד פעמיים במקביל על אותו פריט. וברגע שתעדכנו את תוכן הפריט ותלחצו על עדכון, אז באופן אוטומטי יתעדכן תוכן המסך השני (וגם כמובן תוכן מסת תצוגת רשימה)

פרק 10 - פתיחת מסך תצוגת פריט בודד מתוך מסך תצוגת הרשימה

כעת, נחזור למסך תצוגת הרשימה על מנת לפתוח דרכו את מסך תצוגת פריט בודד:

- לחיצה כפולה על פריט ברשימה תפתח את מסך תצוגת פריט בודד במצב עדכון של הפריט הנבחר
- לחיצה על כפתור Add בתחתית הרשימה תפתח את מסך תצוגת פריט בודד במצב הוספה

10א. מסך תצוגת הרשימה - קישור הפריט הנבחר ברשימה לתכונה בקוד האחורי

נרצה להגדיר בקוד האחורי תכונה מטיפוס ישות הנתונים הלוגית המצומצמת המיוצגת כפריט בודד ברשימה, כדי שנוכל לקשר אותה באופן אוטומטי לפריט הנבחר מהרשימה.

1. נגדיר בקוד האחורי של המסך, תכונה רגילה (כלומר לא **Dependency Property**), בשם שתבחרו, מטיפוס ישות הנתונים הלוגית. את התכונה הזו נקשר לפריט הנבחר ב **ListView\DataGrid**. התכונה הזו תתעדכן רק מכיוון התצוגה לכיוון הקוד האחורי ולא להיפך, ולכן היא יכולה להיות תכונה רגילה ולא תכונת תלות.

למשל:

```
public BO.CourseInList? SelectedCourse { get; set; }
```

2. דרך קובץ ה-XAML של המסך, הוסיפו לפקד ה **ListView\DataGrid**, אטריבוט בשם **SelectedItem** שמייצג את הערך הנבחר בו. קשרו אותו דרך מנגנון ה **Data Binding** לתכונה שהגדרתם בסעיף הקודם.

למשל:

```
<ListView Grid.Row="1" ItemsSource="{Binding CourseList}" SelectedItem="{Binding SelectedCourse}"
MouseDoubleClick="lsvCoursesList_MouseDoubleClick"/>
```

3. כך הקישור נוצר באופן אוטומטי

10ב. פתיחת מסך תצוגת פריט בודד במצב עדכון מתוך מסך תצוגת רשימה

על מנת לפתוח את מסך תצוגת פריט בודד במצב עדכון, לאחר לחיצה כפולה על פריט ברשימה:

1. הוסיפו לפקד ה- **ListView\DataGrid** אירוע לחיצה כפולה (**MouseDoubleClick**) על פריט ברשימה

2. ממשו את מתודת האירוע כך שתבצע יצירה של מסך חדש מסוג **מסך תצוגת פריט בודד במצב עדכון** (כלומר צריך לשלוח **Id** של הפריט הנבחר ברשימה לבנאי של המסך שנוצר)

3. ודאו שהמסך נפתח במצב של **Show**, כלומר ברגע שהוא נפתח אפשר לחזור למסך הקודם, עד שהמסך הנוכחי לא נסגר.

למשל:

```
private void lsvCoursesList_MouseDoubleClick(object sender, MouseButtonEventArgs e)
{
    if (SelectedCourse != null)
        new CourseWindow(SelectedCourse.Id).Show();
}
```

10ג. פתיחת מסך תצוגת פריט בודד במצב הוספה מתוך מסך תצוגת רשימה

על מנת לפתוח את מסך תצוגת פריט בודד במצב הוספה, לאחר לחיצה כפתור Add:

1. בתחילת המסך, הוסיפו כפתור **Add** בתחתית מסך תצוגת הרשימה, הוסיפו לכפתור אירוע ברירת מחדל של לחיצה על הכפתור (**Click**)

2. ממשו את מתודת האירוע כך שתבצע יצירה של מסך חדש מסוג **מסך תצוגת פריט בודד במצב הוספה** (כלומר לא צריך לשלוח **Id** של אף פריט לבנאי של המסך שנוצר)

3. ודאו שהמסך נפתח במצב של **Show**, כלומר ברגע שהוא נפתח אפשר לחזור למסך הקודם, עד שהמסך הנוכחי לא נסגר.

10ד. רענון הרשימה לאחר הוספה/עדכון של פריט

1. הריצו את התוכנית, פתחו את מסך תצוגת הרשימה, הוסיפו/עדכנו פריטים ברשימה דרך מסך תצוגת פריט.

2. וודאו שלאחר כל הוספה/עדכון/מחיקה של פריט ברשימה - הרשימה מתעדכנת באופן אוטומטי בזכות מנגנון המשיקים שמימשתם.

פרק 11 - מסך תצוגת רשימה - מחיקת פריט בודד מהרשימה

1. במסך תצוגת הרשימה, בעזרת שימוש ב Template של ListView\DataGrid הוסיפו כפתור נפרד לכל שורה ברשימה. כך שלחיצה על הכפתור תבצע מחיקה של הפריט שהכפתור שייך אליו ברשימה:

a. לפני המחיקה וודאו בעזרת MessageBox שהמשתמש בטוח ברצונו למחוק את הפריט הנבחר.

b. קריאה למתודה Delete ב BL, שתנסה לבצע מחיקה של הפריט.

c. במידה והמחיקה הצליחה, אזי באופן אוטומטי הפריט יעלם מהרשימה בזכות מנגנון המשקיפים שמימשנו.

d. במידה והמחיקה נכשלה, (שכבת ה BL כאמור בודקת אם מותר למחוק וכו'), יש לתפוס את החריגה מה BL ולשלוח במקומה MessageBox מתאים.

2. אפשרות נוספת למימוש מחיקת פריט בודד - כפי שמצויין במסמך הכללי: לממש את המחיקה מתוך מסך "תצוגת פריט בודד" - לבחירתכם. העיקר שתהיה אפשרות למחוק פריט בודד. יש לשים לב שכפתור המחיקה "נראה" רק כשהמסך נפתח במצב של הוספה/עדכון. כמובן, שבכל אחת מהדרכים, יש לשמור על כללי המחיקה הנכונה.

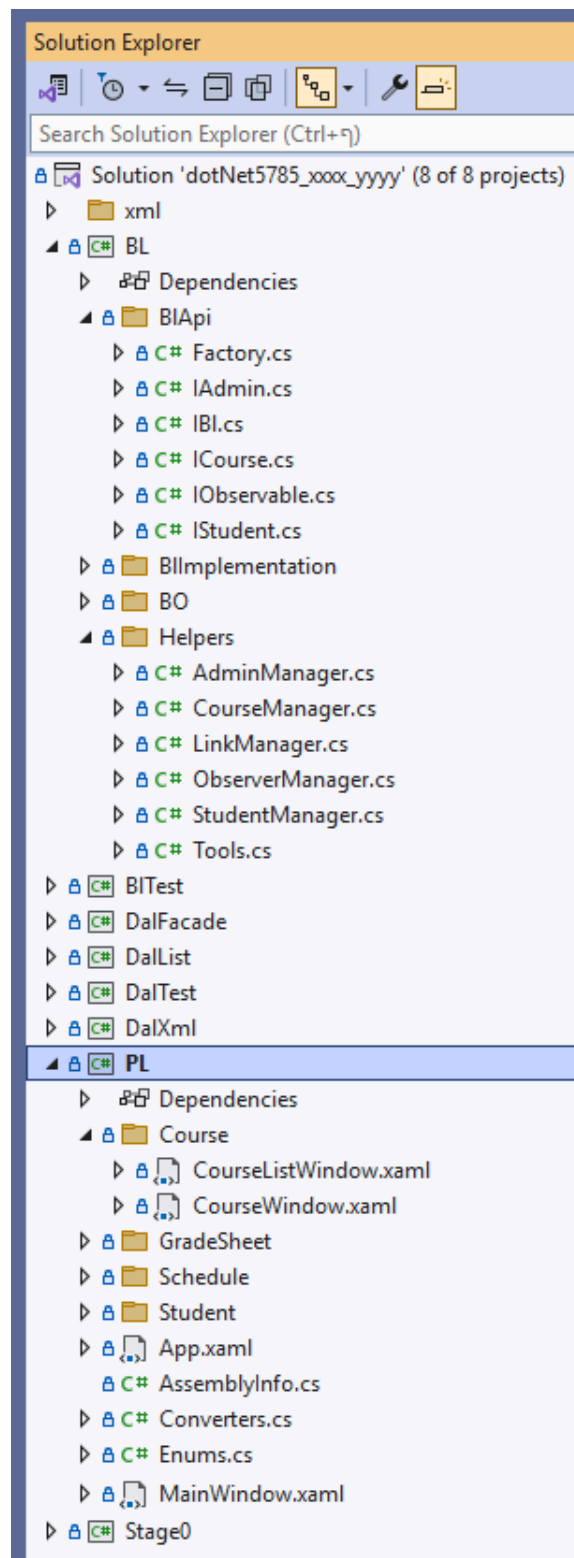
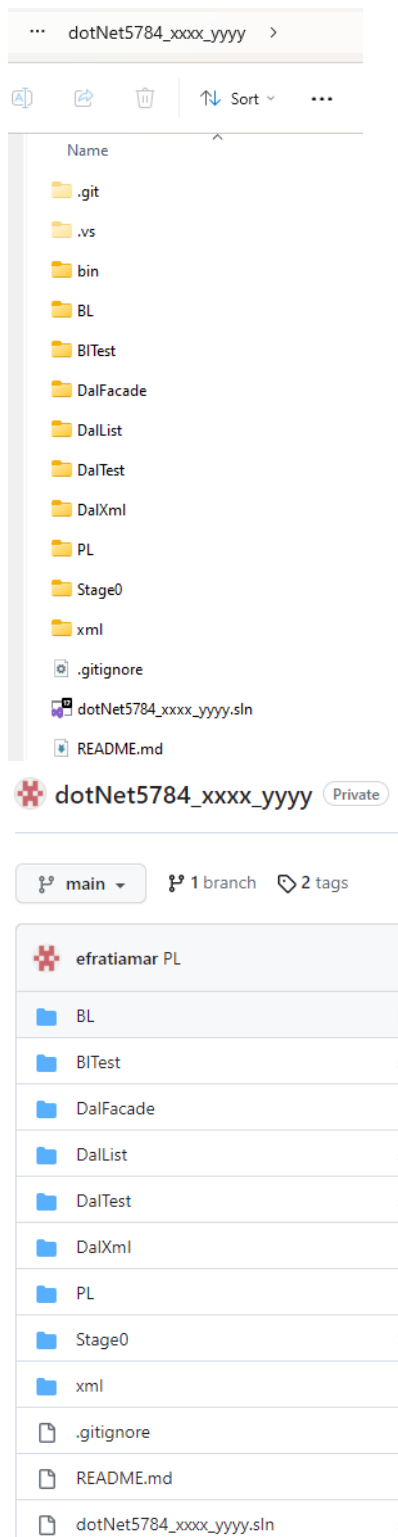
פרק 12 - עיצוב המסכים בהתאמה אישית

כאן יש לכם יד חופשית לעצב את המסכים כדי הדמיון הטובה עליכם, ניתן לבחור צבעים, גופנים, כותרות ועוד.

אפשר גם לעשות שימוש בנושאים מתקדמים של WPF כפי שמופיעים בתחילת מסמך זה. אך אפשר להמתין איתם לשלב 6.

פרק 13 - תמונות מצב


לאחר ביצוע כל הפרקים של שלב 5, המאגר ב-github.com, ה-Solution ב-Visual Studio, והתיקיות בסייר הקבצים אמורים להיראות כך:



פרק 14 - יצירת תג והגשת שלב 5 במודל

ע"פ ההנחיות שמוסברות בשלב 0:

- צרו תג בשם: Stage5 Final commit
- הגישו קישור לתג במודל של כל אחד מבני הזוג

 סוף שלב 5 - בהצלחה

נספחים - שלב 5
נספח 1 - דוגמאות למסכים

דוגמא למסך ניהול ראשי - מסך מספר 1

The screenshot shows the 'MainWindow' application. It features a title bar with standard window controls. The main content area includes a 'Max Year Range' label with a text input field containing the number '5' and an 'Update' button. Below this is a descriptive text: '[The maximum number of years for study, After that the student becomes InActive]'. To the right, a blue box highlights a date and time display '9.12.2024 12:36' above a vertical stack of five buttons: 'Add One Minute', 'Add One Day', 'Add One Hour', 'Add One Month', and 'Add One Year'. At the bottom of the window, there are four buttons arranged in a 2x2 grid: 'Reset DB', 'Init DB', 'Handle Students', and 'Handle Courses'.

דוגמא למסך תצוגת רשימה - מסך מספר 2

תצוגת הרשימה אינה מתקדמת - והדרישה בשלב זה היא לשכלל אותה ולהציגה בצורה משוכללת יותר
בעזרת Template

The screenshot shows the 'CourseListWindow' application. It has a title bar with standard window controls. Below the title bar is a 'Select Semester:' label followed by a dropdown menu currently set to 'None'. The main area is a list box containing five course entries, each with the following fields: Id, CourseNumber, CourseName, InYear, and InSemester. The entries are: Id: 1001, CourseNumber: 101-666-555, CourseName: CourseB 222, InYear: FirstYear, InSemester: SpringB; Id: 1003, CourseNumber: 103-666-767, CourseName: CourseD, InYear: FirstYear, InSemester: SpringB; Id: 1004, CourseNumber: 101-666-777, CourseName: CourseA 1, InYear: ExtraYear, InSemester: WinterA; Id: 1005, CourseNumber: 101-666-555, CourseName: CourseB 222. At the bottom right of the list box is an 'Add' button.

דוגמא למסך תצוגת פריט בודד במצב הוספה -
מסך מספר 3

CourseWindow

Id: 1004

CourseNumber: 101-666-777

CourseName: CourseA 1

InYear: ExtraYear

Semester: WinterA

Day In Week: Wednesday

Start Time: 18:00:00

End Time: 13:00:00

Credits: 1

Update

דוגמא למסך תצוגת פריט בודד במצב הוספה -
מסך מספר 3

CourseWindow

Id: 0

CourseNumber:

CourseName:

InYear: None

Semester: None

Day In Week: None

Start Time:

End Time:

Credits:

Add

נספח 2 - דוגמא להגדרה ושימוש ב Converter

נניח שנרצה לקשר דרך מנגנון ה Binding ב XAML בין 2 תכונות של רכיב גרפי ושל רכיב לוגי שאינן מאותו טיפוס. מכיוון ש 2 התכונות אינן מאותו טיפוס יש להגדיר ב XAML באזור ה Binding גם Converter מתאים שממיר מטיפוס אחד לשני.

לדוגמא, נניח שנרצה להציג ערך מסוג Enum, שמייצג שנת לימוד של קורס, בתוך פקד מסוג `TextBlock`. תכונת ה Text של הפקד מקושרת בעזרת מנגנון ה Binding כרגיל.

נרצה קשר בין ערך ה- Enum לבין צבע תכונת ה- Background של אותו פקד - והן אינן מאותו טיפוס! לצורך כך נגדיר Converter:

1. צרו בפרויקט PL קובץ חדש בשם `Converters.cs`, שיהיה ייעודי לכל ה Converters שתצטרכו בפרויקט PL.

2. הגדירו בתוך הקובץ החדש, מחלקה חדשה שיורשת מ- `IValueConverter`. למשל:

```
class ConvertYearToColor : IValueConverter
{
    public object Convert(object value, Type targetType, object parameter, CultureInfo culture)
    {
        BO.Year year = (BO.Year)value;
        switch (year)
        {
            case BO.Year.FirstYear:
                return Brushes.Yellow;
            case BO.Year.SecondYear:
                return Brushes.Orange;
            case BO.Year.ThirdYear:
                return Brushes.Green;
            case BO.Year.ExtraYear:
                return Brushes.PaleVioletRed;
            case BO.Year.None:
                return Brushes.White;
            default:
                return Brushes.White;
        }
    }

    public object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)
    {
        throw new NotImplementedException();
    }
}
```

a. בקובץ המשאבים של הפרויקט `App.xaml` הוסיפו הגדרה של משאב סטטי עם מפתח `x:Key` בשם שתבחרו, וקשרו אותו ל Converter שהגדרתם בסעיף הקודם.

למשל:

```
<Application.Resources>
    <local:ConvertYearToColor x:Key="ConvertYearToColorKey" />
</Application.Resources>
```

b. נציין את שם ה Converter בעזרת מנגנון ה Binding דרך ה XAML:

למשל:

```
<DataGridTemplateColumn Header="InYear" Width="Auto">
    <DataGridTemplateColumn.CellTemplate>
        <DataTemplate>
            <TextBlock Text="{Binding InYear,Mode=TwoWay}" FontSize="16" Background="{Binding Path=InYear,
Converter={StaticResource ConvertYearToColorKey}}"/>
        </DataTemplate>
    </DataGridTemplateColumn.CellTemplate>
</DataGridTemplateColumn>
```