**האוניברסיטה הפתוחה**

**המחלקה למתמטיקה ולמדעי המחשב**

סדנה בתכנות מונחה עצמים – מערכת EasySchool לניהול בתי ספר

20375

מסמך עיצוב

**העבודה הוכנה על-ידי:** ולדיסלב אוסיפקוב 323363788

**בהדרכתו של** דני כלפון

# תוכן עניינים

[תוכן עניינים 2](#_Toc151149324)

[תיאור כללי 4](#_Toc151149325)

[הנחיות והנחות עבודה 4](#_Toc151149326)

[מוסכמות כתיבה 4](#_Toc151149327)

[שכבת בסיס הנתונים (EasySchool.Model) 5](#_Toc151149328)

[Users.cs 6](#_Toc151149329)

[Person.cs 6](#_Toc151149330)

[Room.cs 7](#_Toc151149331)

[Class.cs 7](#_Toc151149332)

[Student.cs 8](#_Toc151149333)

[Course.cs 8](#_Toc151149334)

[Teacher.cs 9](#_Toc151149335)

[Lesson.cs 10](#_Toc151149336)

[Grades.cs 11](#_Toc151149337)

[Events.cs 12](#_Toc151149338)

[Messages.cs 12](#_Toc151149339)

[SchoolInfo.cs 13](#_Toc151149340)

[שכבת הלוגיקה (EasySchoolViewModel) 14](#_Toc151149341)

[מחלקות כלליות 15](#_Toc151149342)

[מחלקות לוגיות 16](#_Toc151149343)

[שכבת התצוגה (EasySchool.View) 17](#_Toc151149344)

[דפוסי עיצוב -MVVM 18](#_Toc151149345)

[הצעות שיפור 19](#_Toc151149346)

[שיפור בהזנה וצפייה בנתונים 19](#_Toc151149347)

[שיפור בניהול המשתמשים 20](#_Toc151149348)

[דיאגרמות 21](#_Toc151149349)

[דיאגרמת מסד נתונים 21](#_Toc151149350)

[דיאגרמות עבור שכבת ה-Business logic 21](#_Toc151149351)

[לוגיקת התחברות 21](#_Toc151149352)

[דיאגרמת מחלקות עבור שכבת הלוגיקה 22](#_Toc151149353)

[הסבר מילולי לדיאגרמות 24](#_Toc151149354)

[דיאגרמה עבור שכבת הpresentation 24](#_Toc151149355)

[דיאגרמות רצף 25](#_Toc151149356)

[דיאגרמת התחברות 25](#_Toc151149357)

[דיאגרמת יצירת משתמש 26](#_Toc151149358)

[דיאגרמת עדכון משתמש 26](#_Toc151149359)

[דיאגרמת מחיקת משתמש 26](#_Toc151149360)

[דיאגרמת אירועים 27](#_Toc151149361)

[דיאגרמת הוספה/עדכון של ציון 27](#_Toc151149362)

# תיאור כללי

המערכת בנויה לפי מודל 3-Tier שבו המערכת מופרדת לשלוש שכבות:

* שכבת בסיס הנתונים (Database Layer) – שכבת המידע של הפרויקט
* שכבת הלוגיקה (Business Logic Layer) – שכבת ה-"היגיון"/צד שרת
* שכבת התצוגה (Presentation Layer) – השכבה שדרכה המשתמש מתקשר עם המערכת

כל שכבה מתקשרת עם כל שכבה אחרת באמצעות מערכת קבועה ומוגדרת מראש, מה שמאפשר החלפה של כל אחת מהשכבות בקטע קוד אחר לחלוטין ע"י שינוי פשוט באמצעיי התקשורת בין השכבות. בכך למעשה אנו יכולים לחסוך שכתוב של קטעים נרחבים מהקוד ולהוסיף דברים חדשים ללא קושי. המערכת נוצרה באמצעות WPF אבל בגלל המבנה הזה היא אינה חייבת להישאר תחתיו.

## הנחיות והנחות עבודה

1. המערכת מגיעה מאותחלת עם המידע המינימלי הנדרש להרצתה, כגון משתמש המנהל וטבלת schoolInfo מוגדרת מראש.
2. עבור כל מסד נתונים שהמערכת משתמשת בו, מוגדר בית ספר יחיד. המערכת לא מסוגלת לנהל מספר מסדי נתונים בו זמנית.
3. במערכת אין פונקציות חיפוש או חלוקת מסד הנתונים לדפים אלא מציגה את כל המידע למשתמש באופן ישיר, דבר אשר מגביל את היכולת הפרקטית שלה להכיל כמויות גבוהות במיוחד של ערכים. ההגבלה הפרקטית
4. לאחר שמשתמש התחבר לראשונה למערכת, לא תיתכן אפשרות עבורו לשנות סיסמה
5. מערכת השעות מוגבלת ל10 שעות ביום ושישה ימים בשבוע.
6. המערכת מארחת את שרת המידע באופן חיצוני ולכן נדרש חיבור אינטרנטי עקבי אצל כל משתמשי הקצה
7. שמות (פרטיים, משפחה, שמות משתמש וכדומה) מורכבים ממילים יחידות כל אחד וללא תווים
8. המערכת משתמשת בלוח שנה חיצוני שעובדה על בסיס תאריכים בפורמט אמריקאי (mm/dd/yyyy) לעומת הפורמט המקובל בישראל (dd/mm/yyyy)

## מוסכמות כתיבה

1. כל מילה שונה בשם של מחלקה תיכתב באות גדולה
2. שמו של כל ממשק יתחיל עם אות I גדולה ולאחריה מילה באות גדולה כנהוג במחלקות
3. שמות של משתנים המורכבים ממספר מילים יתחילו עם אות קטנה עבור המילה הראשונה ואותיות גדולות עבור שאר המילים
4. כל שדה מסוג private ייכתב עם הסימון "\_" בתחילתו
5. קבועים ייכתבו עם אותיות גדולות בלבד וסימונים "\_" בין המילים שלהם במידה וקיימות מספר מילים
6. הקוד יחולק באמצעות region# ניתן למזעור לשם נוחות קריאה

# שכבת בסיס הנתונים (EasySchool.Model)

שכבה זו היא האחראית להגדרת התקשורת של המערכת מול מסד הנתונים שלנו. בפרט, היא זו שמגדירה וממקבלת אלמנטים ממסד הנתונים אל אלו של המערכת ומגדירה את התקשורת ביניהם באמצעות מחלקות הDbSet ו-DbContext.

מסד הנתונים שבו המערכת משתמשת הוא Microsoft SQL Server והוא מוחזק על שרת בענן בתפעול ובאבטחה של Azure שתוחם את גודל המסד. החיבור אליו נעשה דרך connectionString ייעודי המוגדר בקובץ App.config. Datamodel מגדיר לנו את פעולות ה-get וה-set שלנו עבור הטבלאות וכמו כן הוא מה שמאפשר לגשת לתכונות (עמודות) השונות בטבלאות.

מעבר למסדי הנתונים הנ"ל יש לנו מספר פונקציות שעוסקות ישירות בעדכון מסד הנתונים ואינן חלק מהלוגיקה השוטפת של התוכנה. נציג אותן כאן:

* Globals.cs – מחלקה שמהותה היא להגדיר את כל הקבועים (constants) ולספק פעולות שנוגעות בהם, למשל המרה בין מספר יום לבין שם יום (למשל, 2 ל-"Monday")
* MessageHandler.cs – מתודה שמטרתה היא ליצור הודעה ולשמור אותה במסד הנתונים באמצעות הפונקציה CreateMessage
* TeacherCoursesHandler.cs – מתודה שמטרתה ליצור מילון (dictionary) שמכיל את כל המקצועות שמורה מסוים יכול ללמד, מה שמאפשר גישה נוחה יותר אליהם שכן במסד הנתונים עצמו הם רשומים בשדות שונים בתור firstCourseID, secondCourseID וכן הלאה, ולכן הגישה אליהם מורכבת יותר ואי אפשר לגשת אליהם באופן סדרתי כבמערך וגם אז זה לא בהכרח יעבוד (למשל, ייתכן והמורה מלמד שני קורסים אבל הם רשומים בתור המקצוע הראשון והמקצוע השלישי שלו, בזמן שsecondCourseID נותר ריק)

## Users.cs

טבלה המייצגת את פרטי חשבונות המשתמשים במערכת. ההפרדה בין משתמש לאדם נעשית שכן הפונקציונליות של המשתמש (כגון שינוי סיסמה עבור המשתמש, התחברות לחשבון המשתמש, מחיקת המשתמש או יצירת משתמש חדש) שונה מהפונקציונליות שלו כבן אדם

|  |  |  |
| --- | --- | --- |
| משתנה | טיפוס | הסבר |
| UserID | Int | מספר מזהה למשתמש, מוגדר בincrement |
| Username | String | שם המשתמש. מוגבל ל16 תווים ע"י המסד |
| Password | String | סיסמה. מוגבלת ל16 תווים ע"י המסד |
| isDisabled | Bool | האם המשתמש כבוי (למשל בפתיחת שנה חדשה) |
| hasToChangePassword | Bool | האם על המשתמש לשנות סיסמה (שימושי עבור התחברות ראשונה) |

## Person.cs

טבלה המייצגת את פרטי האנשים שמחזיקים בחשבונות השונים במערכת

|  |  |  |
| --- | --- | --- |
| משתנה | טיפוס | הסבר |
| personID | Int | מספר מזהה לבן אדם, מוגדר בincrement |
| userID | Int | מספר מזהה למשתמש, נגזר מ-user.cs |
| firstName | String | שם פרטי, מוגבל ל20 תווים |
| lastName | String | שם משפחה, מוגבל ל20 תווים |
| Email | String | כתובת אימייל, מוגבלת ל50 תווים |
| phoneNumber | String | מספר טלפון. מוגבל ל12 תווים |
| Birthdate | System.DateTime | תאריך לידה |
| isStudent | bool | האם המשתמש תלמיד |
| isTeacher | bool | האם המשתמש מורה |
| isSecretary | bool | האם המשתמש מהמזכירות |
| isPrinciple | bool | האם המשתמש מנהל |
| isParent | bool | האם המשתמש הורה |

## Room.cs

טבלה המייצגת את החדרים והמקומות הפיזיים המוגדרים במוסד. "חדר" בהקשר הזה יכול להיות גם מגרש הכדורגל.

|  |  |  |
| --- | --- | --- |
| משתנה | טיפוס | הסבר |
| roomID | Int | מספר מזהה לחדר, מוגדר בincrement |
| roomName | String | השם שניתן לחדר, למשל חדר מחשבים או מעבדה |

## Class.cs

טבלה המייצגת את כיתות הלימוד השונות בבית הספר

|  |  |  |
| --- | --- | --- |
| משתנה | טיפוס | הסבר |
| classID | Int | מספר מזהה לכיתה, מוגדר בincrement |
| roomID | Int | מספר מזהה לחדר, נגזר מRoom.cs |
| className | String | השם שניתן לכיתה, למשל כיתת מב"ר או כיתה ח' |

## Student.cs

טבלה המייצגת את התלמידים הלומדים בבית הספר

|  |  |  |
| --- | --- | --- |
| משתנה | טיפוס | הסבר |
| studentID | Int | מספר מזהה לתלמיד, נגזר מתוך Person.cs – המספר המזהה של האדם שמוגדר בתור isStudent |
| classID | Int | מספר מזהה לכיתה, נגזר מתוך Class.cs |
| absencesCounter | Int | מספר החיסורים הרשומים עבור התלמיד |
| parentID | Int | מספר מזהה להורה התלמיד, נגזר מתוך Person.cs – המספר המזהה של האדם שמוגדר בתור isParent. אין חובה להגדרת הורה לתלמיד |

## Course.cs

טבלה המייצגת את המקצועות הנלמדים בבית הספר

|  |  |  |
| --- | --- | --- |
| משתנה | טיפוס | הסבר |
| courseID | Int | מספר מזהה למקצוע, מוגדר בincrement |
| courseName | String | שם המקצוע, למשל אנגלית או ספרות |
| isHomeroomTeacherOnly | Bool | האם זה מקצוע מחנך |

## Teacher.cs

טבלה המתארת את המורים ואת המקצועות אותם הם מלמדים

|  |  |  |
| --- | --- | --- |
| משתנה | טיפוס | הסבר |
| teacherID | Int | מספר מזהה למורה, נגזר מתוך Person.cs – המספר המזהה של האדם שמוגדר בתור isTeacher. |
| classID | Int | מספר מזהה של הכיתה שהמורה מחנך בה, נגזר מתוך Class.cs. אם השדה הזה ריק זה אומר שהמורה לא מחנך של אף כיתה |
| firstCourseID | bool | מספר מזהה של המקצוע הראשון שהמורה מלמד, נגזר מ.Course.cs. שדה חובה |
| secondCourseID | bool | מספר מזהה של המקצוע השני שהמורה מלמד, נגזר מ.Course.cs. אינו שדה חובה |
| thirdCourseID | bool | מספר מזהה של המקצוע השלישי שהמורה מלמד, נגזר מ.Course.cs. אינו שדה חובה |
| fourthCourseID | bool | מספר מזהה של המקצוע ההרביעי שהמורה מלמד, נגזר מ.Course.cs. אינו שדה חובה |

## Lesson.cs

טבלה המייצגת את השיעורים השונים בשעות השונות שהתלמידים ייקחו. כל שורה בטבלה מסוגלת להכיל פרטים על עד ארבעה שיעורים בלבד ולו בתנאי שהם באותו מקצוע, וכל הוספה נוספת מאלצת יצירת שורה חדשה בטבלה

|  |  |  |
| --- | --- | --- |
| משתנה | טיפוס | הסבר |
| lessonID | Int | מספר מזהה של השיעור, מוגדר בincrement |
| teacherID | String | מזהה המורה המלמד. בדומה לשאר מזהיי המורים |
| courseID | Int | מזהה המקצוע הנלמד. נגזר מתוך Course.cs |
| classID | Int | מזהה הכיתה המזוהה. נגזר מתוך Class.cs |
| roomID | Int | מזהה החדר המזוהה. נגדר מתוך Room.cs. לא חובה להזין |
| firstLessonDay | byte | היום שבו השיעור הראשון מהמקצוע המוגדר נמצא. |
| firstLessonHour | byte | השעה שבה השיעור הראשון מהמקצוע המוגדר נמצא |
| secondLessonDay | byte | היום שבו השיעור השני מהמקצוע המוגדר נמצא. |
| secondLessonHour | byte | השעה שבה השיעור השני מהמקצוע המוגדר נמצא |
| thirdLessonDay | byte | היום שבו השיעור השלישי מהמקצוע המוגדר נמצא. |
| thirdLessonHour | byte | השעה שבה השיעור השלישי מהמקצוע המוגדר נמצא |
| fourthLessonDay | byte | היום שבו השיעור הרביעי מהמקצוע המוגדר נמצא. |
| fourthLessonHour | byte | השעה שבה השיעור הרביעי מהמקצוע המוגדר נמצא |

## Grades.cs

הטבלה הזאת מייצגת את הציונים שתלמיד ספציפי קיבל בקורס ספציפי (שני השדות הנ"ל מהווים מפתח לטבלה)

|  |  |  |
| --- | --- | --- |
| משתנה | טיפוס | הסבר |
| studentID | int | מזהה התלמיד, בדומה למקודם |
| courseID | int | מזהה הקורס, בדומה למקודם |
| teacherID | int | מזהה המורה, בדומה למקודם |
| Score | Byte | הציון שהוזן לתלמיד |
| Notes | String | שדה בן עד 150 תווים המיועד לשם הערות |

## Events.cs

הטבלה הזאת מייצגת את האירועים השונים של התלמידים והמורים בבית הספר

|  |  |  |
| --- | --- | --- |
| משתנה | טיפוס | הסבר |
| eventID | Int | מזהה של האירוע, מוגדר בincrement |
| submitterID | Int | המזהה של מי שרשם את האירוע |
| recipientID | Int | המזהה של האדם שמקבל את האירוע. שדה זה יכול להיות ריק אם האירוע מוגדר לכיתה למשל |
| recipientClassID | Int | המזהה של הכיתה שמקבלת את האירוע. שדה זה יכול להיות ריק אם האירוע מוגדר לאדם מסוים למשל |
| Name | String | שם האירוע. מוגבל ל50 תווים |
| Description | String | תיאור של פרטי האירוע. מוגבל ל250 תווים |
| eventDate | System.DateTime | התאריך והשעה של האירוע |
| Location | String | המיקום של האירוע. מוגבל ל50 תווים. אינו שדה חובה |

## Messages.cs

הטבלה הזאת מייצגת את ההודעות השונות הנשלחות בין משתמש לבין עצמם ובין המערכת לבין המשתמשים

|  |  |  |
| --- | --- | --- |
| משתנה | טיפוס | הסבר |
| messageID | int | מזהה ההודעה. מוגדר באמצעות increment |
| senderID | int | מזהה השולח כפי שהוגדר מקודם. אם ריק, אז זאת הודעת מערכת |
| recipientID | int | מזהה המקבל כפי שהוגדר מקודם. אם ריק, ההודעה לא מופנית לאף אדם ספציפי |
| recipientClassID | int | מזהה הכיתה המקבלת כפי שהוגדר מקודם. אם ריק, ההודעה לא מופנית לאף כיתה ספציפית |
| forAllManagement | bit | האם ההודעה מופנית להנהלה |
| forAllTeachers | bit | האם ההודעה מופנית למורים |
| forAllStudents | bit | האם ההודעה מופנית לתלמידים |
| forEveryone | bit | האם זו הודעה כללית |
| data | string | תוכן ההודעה, מוגבל ל350 תווים |
| date | System.DateTime | תאריך ושעת שליחת ההודעה |
| title | String | כותרת ההודעה |

## SchoolInfo.cs

מזהה המידע על בית הספר. בניגוד לשאר המזהים, מוגדר באיחסון שורות ולא באיחסון עמודות – העמודה הראשונה מהווה מזהה לערך הרלוונטי שנמצא בעמודה השנייה ובהרחבה לכך הגישה נעשית ע"י חיפוש הערך לפי שמו ואז שליפת הערך המתאים לו. זה נעשה כך שכן קיים רק בית ספר אחד במערכת בכל פעם וזוהי הדרך הנוחה ביותר לאכוף את ההגבלה הזאת

|  |  |  |
| --- | --- | --- |
| משתנה | טיפוס | הסבר |
| Key | String | משמש בפועל כמזהה של הערכים המבוקשים, אנחנו לא "שולפים" אותו אלא מחפשים באמצעותו  ערכי החיפוש הם:   * schoolName – שם בית הספר * schoolDescription – תיאור על בית הספר * schoolImage – הכתובת במחשב של הלוגו של בית הספר |
| Value | String | הערך אותו אנחנו מבקשים למצוא בחיפוש שב-key לפי ערך החיפוש הרלוונטי |

שכבת הלוגיקה (EasySchoolViewModel)

שכבת קוד זו מכילה בתוכה את כל האובייקטים ומתודות העזר שמהותן לקשר בין שכבת התצוגה לשכבת מסד הנתונים. העקרונות המנחים עבורנו בעת הפיתוח של השכבה הנ"ל הם כלדהלן:

* שכבת הלוגיקה היא זו המספקת את הליכי אימות הנתונים ובדיקת החריגות סביב הגישה אל מסד הנתונים
* שכבת הלוגיקה שומרת על עיקרון הכימוס בכך שהיא מאפשרת תקשורת בין מחלקות שונות אך ורק באמצעות מתודות ציבוריות או תכונות (properties) ושואפת למזער את המידע שהמחלקה מנגישה למחלקות אחרות לכדי מידע רלוונטי עבורן בלבד
* אנו נשאף למקסם גנריות בכתיבת הקוד ככל הניתן וזו במטרה להפוך אותו ליותר גמיש לשינויים, החלפות והתאמות
* שכבת הלוגיקה בנויה לפי עיקרון ה-open-close, כלומר כל מחלקה ופעולה שמתבצעת היא מערכת סגורה בפני עצמה ואינה גמישה לשינויים אבל כן גמישה להרחבות של המודולריות שלה
* שכבת הלוגיקה בנויה באופן שמאפשר לנו למקסם מחזור של קוד ומונע כתיבה מיותרת
* כל מחלקה בשכבת הלוגיקה ממונה על תפקיד אחד בלבד, מה שעוזר לנו להחליף אותה במידה ונצטרך מבלי לגרום לכל הקוד לקרוס
* נשתמש בדפוסי עיצוב כאשר הם רלוונטיים, ובפרט בדפוס העיצוב מסוג MVVM שמטרתו לאפשר לנו לתחום את הלוגיקה מהתצוגה ומסד הנתונים ביותר קלות
* נשאיר מחלקות המשמשות למטרות דיבאגינג או אימות/בדיקת איכות הקוד שכן הפרויקט במהותו בנוי להרחבה ואינו מהווה מוצר מוגמר

## מחלקות כלליות

קבוצה זו היא קבוצה של מחלקות שמהותן היא להוות תמיכה ופונקציונליות כללית עבור התוכנה שאנו כותבים, חלקן משויכות למודל עצמו (למשל, בעזרה בעבודה במסד הנתונים) וחלקן משויכות לתיקיית ה-utilities ומטרתם לאפשר לנו למחזר קוד בהתאם לעקרונות המנחים שלפיהם אנו עובדים

* BaseViewModel.cs – מחלקה מופשטת שמטרתה לנהל את אירועי הINotifyPropertyChange
* IMessageBoxService – מחלקה שמגדירה את סוגי ההודעות האפשריים באמצעות enum-ים וכמו כן ממשק בוליאני ImessageBoxService שאמור להגדיר את המבנה של ההודעות הנ"ל כאשר הפונקציונליות שלהן מוגדרת ע"י המחלקות שיורשות אותו
* IScreenViewModel.cs – מחלקה המגדירה את הממשק שמטרתו להגדיר את המשתמש המנסה להתחבר למערכת ואת ההרשאות שלו והפונקציונליות של המחלקות היורשות אותה תהיה לבדוק אותו ולאפשר גישה לסוגי חלונות מסוימים
* ObservableDictionary.cs – קטע קוד חיצוני אשר מגדיר את הפונקציונליות של היומן (dictionary) איתו נעבוד
* RelayCommand.cs – מחלקה שכל מהותה היא להעניק פונקציונליות לממשק ICommand לממש אותה באמצעות delegates ובכך לקיים תקשורת בין המחלקות הלוגיקה לתצוגה עם Binding
* ValidityResult.cs – מתודה המכילה מבנה שמהותו לבצע בדיקות אימות ולהחזיק את המידע הנ"ל כאובייקט ובהרחבה לכך להעניק לו פונקציונליות של אובייקט

## מחלקות לוגיות

מחלוקת שהפונקציונליות שלהן היא לוגית ומהותן היא לנהל את מסד הנתונים בפועל ולקשר בינו לבין שכבת התצוגה.

* UserCreationViewModel.cs – מאפשרת את יצירת המשתמש עצמה
* UserUpdateViewModel.cs – מאפשרת עדכון או מחיקה של המשתמש. נפרדת מיצירת המשתמש שכן קיים פער לוגי גדול מספיק בין הפעולות בשביל להצדיק הפרדה זו
* LoginViewModel.cs – אחראית על הרצף הלוגי שמאמת את המשתמש ומחבר אותו לחשבון שלו
* NewPasswordViewModel.cs – מחלקה המגדירה את יצירת הסיסמא החדשה עבור משתמש חדש המתחבר לראשונה
* ApplicationViewModel.cs – מנהלת את הלוגיקה של הדף הראשי ואחראית על ניווט בין החלונות השונים לאחר שהמשתמש התחבר
* SchoolInfoViewModel.cs – מציגה את מסך ה-"אודות" של בית הספר שאליו המשתמש מגיע לאחר התחברותו
* ContactsInfoViewModel.cs – מנהלת את דף הקשר של בית הספר
* RoomManagementViewModel.cs – מנהלת את החדרים המוגדרים בבית הספר
* ClassManagementViewModel.cs – מנהלת את הכיתות השונות בבית הספר
* CourseManagementViewModel.cs – אחראית על מקצועות הלימוד בבית הספר
* LessonManagementViewModel.cs – מנהלת את לוגיקת השיעורים ובוחרת מה להציג (למשל, מגבילה את תצוגת המקצועות לבחירה רק לאלו שהמורה יכול ללמד)
* WeeklyScheduleViewModel.cs – מעדכנת את הלו"ז השבועי של השיעורים לפי סוג המשתמש
* CreateMessageViewModel.cs – אחראית על יצירה ושליחה של הודעות בין משתמשים ושליחה של הודעות אוטומטיות שונות
* MessagesDisplayViewModel.cs – מנהל את מערכת קבלת ההודעות של המשתמש
* EventManagementViewModel.cs – מערכת לניהול ולשליחה של אירועים, הגדרת המוען והנמען, וניהול הצגתם למשתמש
* CalenderViewModel.cs – אחראית על לוגיקת הצגת לוח השנה למשתמש ובפרט ביצוע פעולות כגון עדכון, מחיקה, הוספה והצגת אירועים
* StudentGradesViewModel.cs – מנהלת את הצגת הציונים של התלמיד
* GradesReportViewModel.cs – מאפשרת למורה לעדכן את ציוני התלמידים השונים בכל המקצועות שהוא מלמד
* SchoolManagementViewModel.cs – מנהלת את מסך "ניהול בית הספר" אשר המנהל שולט בו ומאפשרת שינוי שם בית הספר, הוספת תיאור עבור בית הספר, שינוי לוגו לבית הספר, וכמו כן גם החלת שנת לימודים חדשה

# שכבת התצוגה (EasySchool.View)

כזכור, התקשורת של שכבת התצוגה עם שכבת הלוגיקה מתבצעת באמצעות קריאה לממשקים שמיישמים את הפונקציות השונות של הלוגיקה באמצעות Data Binding, הדבר מאפשר לנו לחסוך בכתיבה של קוד ובהרחבה לכך במחזור רחב של אותו הקוד.

להלן כמה הנחיות עבודה כלליות שנעזרתי בהן:

* ממשק המשתמש צריך להיות כמה שיותר אינטואיטיבי ומופשט, גם במחיר של עומק תפעולי (הפרויקט הנ"ל אמור להוות "הוכחת מודל")
* כל הקשר עם הלוגיקה ייעשה דרך Data binding לממשקים בלבד, בעיקר בשביל ליצור אפיק תקשורת נפרד בין הלוגיקה לתצוגה מאשר בין הלוגיקה לעצמה (מחלקות שמשתמשות בשיטות של מחלקות אחרות, למשל)
* לעיתים נעשה שימוש בBinding שהוא two-way וזאת במטרה "להצביע" ללוגיקה על הבחירות של המשתמש על מנת שהיא תבצע עדכונים לנקודת ההסתכלות שלה בזמן אמת
* נשאף למקסם גמישות של המסך כמה שניתן וננסה למזער שימוש בגדלים ספציפיים עבור תאים ומקומות

במהלך העבודה על הפרויקט התאמתי כל ViewModel ל-View כלשהו ביחס של אחד לאחד ולכן המחלקות חוזרות על עצמן ותפקיד מחלקות התצוגה הוא לשקף את פעילות מחלקות הלוגיקה המקבילות להן למשתמש.

באופן דומה, בחלק זה גם מספר מחלקות שכל מהותן היא עיצובית, דהיינו לעצב פקדים מסוימים עבור המשתמש (למשל מילונים או מסכי הודעות) ולא נציין אותם מעבר לנקודה זו שכן הם לא מהווים חשיבות פונקציונלית

כן נדגיש חלק ממחלקות התצוגה שיש בהן עניין או מורכבות נוספת מעבר לייעוד הבסיסי הנ"ל. נציין בנפרד את המחלקה הבאה:

* IHavePassword.cs – מחלקה של ממשק שכל מהותו להוות שכבת אבטחה לסיסמה באמצעות ניהול זיכרון ומצביעים. משמשת אותנו גם בcodebehind של קבצי הזאמל וגם בהגדרת כמה מחלקות שמהן נגזרים ממשקי התקשורת עבור הData binding

ועתה נחזור לקבצי הXAML וה-codebehind שלהם שראויים להתייחסות נוספת:

* LoginWindow.xaml – מיישם אבטחת מידע של SecureString בcodebehind
* NewPasswordWindow.xaml – אותו כבר כמו חלון ההתחברות SchoolInfoView.xaml
* ApplicationMainWindow.xaml – מכיל data template שאמורים לייצג את כל הכפתורים מהמסמך הראשי שנטענים בהתאם להרשאות הhasPermition הרלוונטיות למשתמש.
* CalenderView.xaml – מנוהל באמצעות ממשק חיצוני
* ClassManagementView.xaml – מהווה דוגמה לטבלאות המתמלאות באופן אוטומטי מתוך המידע המגיע מהממשקים השונים. אופן התצוגה שלהן הוא של טבלה יחידנית. הפורמט הזה חוזר על עצמו מספר פעמים ולכן נדגיש אותו רק כאן
* GradesReportView.xaml – בודק את זה ששדה הציונים מקבל מספרים בלבד ונמצא באורך התקין בcodebehind
* SchoolInfoView.xaml – מהווה מסך ריק שכל העיצוב בו נטען ישירות לתוכו ואינו נגיש בdesigner
* UserCreationView.xaml – מבצע בדיקת שדות בcodebehind
* UserUpdateView.xaml – מבצע בדיקת שדות בcodebehind

# דפוסי עיצוב -MVVM

במהלך העבודה על הפרויקט הזה נעשה שימוש נרחב ומהותי בארכיטקטורת MVVM, שהיא כשלעצמה תבנית עיצוב מהותית ומשמשת כווריאציה של 3-tier. חילקנו את הקוד לשלושה Namespaces נפרדים בכדי להגדיר את השכבות השונות:

* Model – מייצג את שכבת המידע המודולרית שלנו, תפקידה לייצג את מסד הנתונים ולעזור "לסדר" את המידע ולהנגיש אותו לשאר השכבות (בין אם באמצעות פעולות Get, set) ובין אם ע"י משיכתו וסידורו באופן שמונגש ללוגיקה.
* ViewModel – משמשת אותנו בתור שכבת הלוגיקה שמבצעת את הפעולות כנגד מסד הנתונים מצד אחד ובסיוע בהצגת המידע מנגד. מתקשרת עם התצוגה (View) בזמן אמצעות באמצעות הקפצת התראות דרך BaseViewModel שמממשת את INotifyPropery ומאותת לו כיצד לפעול בהתאם לשינויים שהוא עשה באמצעות אירועים (events)
* View – מתפקדת כשכבת התצוגה. מתקשרת עם הלוגיקה דרך שימוש בממשקים של ICommand שמתפקידם להוות רכיב מפריד בין הלוגיקה לתצוגה ובכך להפחית את התלות של המימוש הספציפי של הלוגיקה מהתצוגה ולהפך. בהינתן ולמשל תחליף רכיבי תצוגה לא נצטרך להחליף את כלל הממשק הלוגי שלנו שכן הוא אינו מותאם עבור התצוגה הספציפית הנ"ל

# הצעות שיפור

## שיפור בהזנה וצפייה בנתונים

למרות שהמערכת בעלת גודל אינסופי בתיאוריה מבחינה תכנותית (כביכול, ללא תלות בגודל האחסון האפשרי שמשתנה בין מימוש למימוש), הבעיה המהותית ביותר במערכת, ומה שמגביל אותה מבחינת אפשרויות הרחבה היא היכולת המוגבלת שלה להזין מידע פר תהליך וכמו כן להציג את המידע הזה למשתמש. הנה מספר הצעות שיפור לדוגמה:

* כאשר אנו מציגים מידע היקפי גדול למשתמש, למשל על כל התלמידים בבית הספר לפי מקצוע, האפשרות לדפדף ולמצוא פרטים ספציפיים מוגבלת ע"י קיבוץ אלפבתי והיכולת להציג את המסך בבת אחת שכן המימוש של הצגת הנתונים הוא פשוט הצגה של עמודות ישירות מתוך מסד הנתונים ומהווה חסם מגביל משמעותי ביכולת של המשתמש למצוא את המידע המבוקש. ע"י הוספת שורת חיפוש שמקושרת לפונקציית find כלשהי או לחילופין לבדיקת contains כלשהי על המסד. כמו כן, ההצגה מבוססת השורות יכולה להיות מוחלפת להצגה הנגזרת מתוך מילון, דבר אשר ייקל משמעותית ניסוח של פונקציות שכאלו שכן מילון ייאפשר איחוד של שדות מתוך מסד הנתונים.
* כאשר אנחנו רושמים שיעורים חדשים לתלמידים אנו לא רואים את הלו"ז השבועי מלפנינו, דבר אשר מוביל למצב שסביר ואף אפשרי שנדרוס בטעות שיעורים שכבר רשומים בלו"ז הספציפי. ע"י הוספת מסך הלו"ז למסך הוספת השיעורים נוכל לזהות את השיעורים האלה, וניתן להוסיף פונקציונליות של הקלקה על השעות ספציפיות בלו"ז השבועי כדרך רישום שיעורים לעומת הממשק הנוכחי שמבצע עדכון ללא שום אמצעי חזותי.
* השורות בטבלאות עצמן מוגבלות בגודלן ולא ניתנות להרחבה "כלפי מטה" כתוצאה מהאופן שבו תיבות טקסט עובדות. רואים זאת במיוחד בדף "צור קשר" ששם ברשימת המקצועות ייתכן מצב וכל המקצועות שמורה מלמד לא ייכנסו בתיבת הטקסט. פתרון פשוט לזה הוא שינוי הגדלים של תיבות הטקסט האחרות לגדלים קבועים או לחילופין יצירת תיבות טקסט שיכולות להתרחב כלפי מטה בכדי לספק יותר מקום כנדרש
* במקומות שבהן יש ComboBox עשוי להיות קשה לדפדף בין הערכים השונים שכן יש חלקים, למשל בחלון הלו"ז, שבהם יש כיתות, חדרים, ותלמידים באותה קבוצה יהיה קשה לדפדף בינם ולמצוא את מה שמבוקש לנו בהנחה ויהיו הרבה חדרים שונים, מורים שונים וכיתות שונות. פתרון אפשרי לבעיה הזאת יכול להיות להגדיר ComboBox-ים נפרדים לאפשרויות נפרדות – למשל ליצור ComboBox עבור הקטגוריה הספציפית שאותה אנחנו מבקשים למצוא ואז ליצור עוד אחד שבוחר מתוך אותה הקטגוריה. הדבר ידרוש מאיתנו לקשור את שתי הקופסאות אלו באלו. פתרון נוסף שיהיה יותר קל ליישום תכנותי אבל לא ניתן להרחבה (תחת הנחת היסוד שאין לנו ולא תהיה לנו קטגוריה נוספת מלבד לקטגוריות הקיימות) היא ליצור מספר ComboBox-ים, אחד לכל קטגוריה, ולבחור מתוכן.

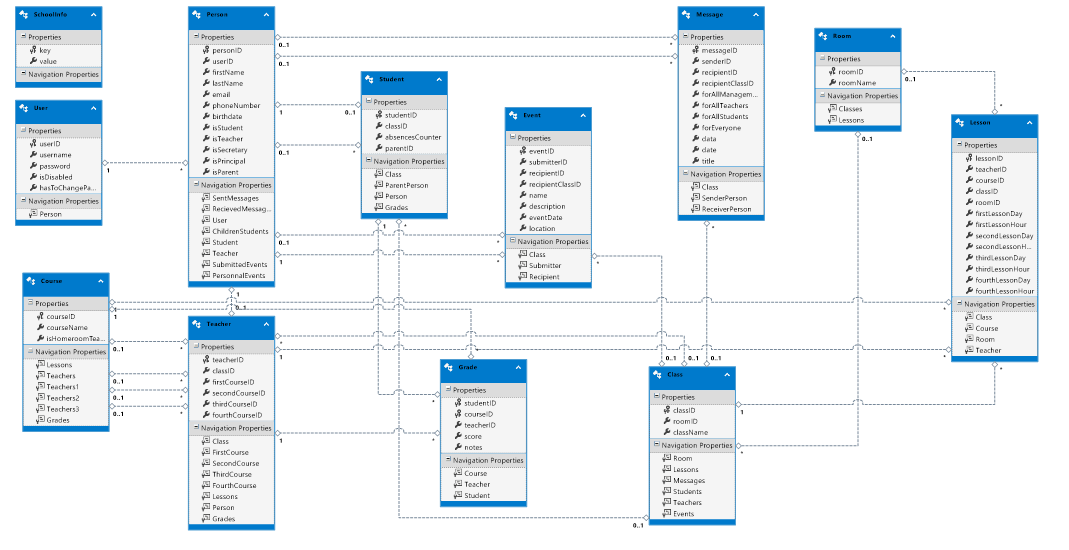
## שיפור בניהול המשתמשים

מערכת ניהול המשתמשים, בין אם בהרשמה, בשמירת הנתונים שלהם, או במחיקתם, מוגבלת לפי העקרונות במסמך האפיון והמדריך למשתמש ובתיאוריה המשתמש, אם יירצה, מסוגל "לשבור" אותה בהנחה והוא לא יתנהל מולה באופן זהיר. נציג כאן חלק מהסוגיות הללו וכיצד ניתן להתגבר עליהן

* נכון לכרגע המערכת לא מסוגלת להבדיל בין משתמשים בעלי שמות משתמש זהים וסיסמאות זהות ומאפשרת ליצור משתמשים בעלי שמות משתמש זהים, דבר אשר בהינתן סיסמה זהה יכול להוביל לכשל של המערכת. דרך אחת לפתור את זה היא באמצעות מניעת היכולת ליצור שמות משתמשים זהים באמצעות השוואה כנגד מסד הנתונים בבדיקת הValidityCheck.
* ניתן להוסיף את האפשרות לשנות סיסמא בכל עת ע"י מימוש החלון של NewPasswordViewModel בתור הדפים שהמשתמש יכול לגשת אליהם מתוך המשתמש בכל רגע נתון. דבר זה ייחזק את אבטחת המידע במערכת אף יותר
* במקרה והמשתמש שכח את סיסמתו ניתן להוסיף מערכת שמאפשרת לו לחדש אותה באמצעות שליחת מייל (פרט מידע זה רשום עבור כל משתמש) שבו תוגדר סיסמה אקראית זמנית שלאחר ההתחברות הראשונה איתה המשתמש יידרש לשנות את סיסמתו. יצירת מערכת שכזאת תדרוש יצירה של מחלקה חדשה במערכת אך היא לא תהיה אחת שתהיה תלויה ב-Viewmodels אחרים.
* הוספת תנאים להרכב הסיסמה, כגון חובת סימנים או אותיות בגדלים שונים בתוך הסיסמה כתנאי להגדת סיסמה במערכת. ניתן ליישם תנאי זה בבדיקת הValidityCheck ולהציג למשתמש חזותית האם הסיסמה שלו עומדת בתנאים בטרם הוא מנסה להכניס אותה באמצעות יצירת Checkbox-ים שמציגים "וי" כאשר תנאי כלשהו בסיסמה הוגשם באמצעות binding ותקשורת מבוססת Notifications.
* לצורכי פשטות, לא השוונו במימוש שלנו את הסיסמאות בעת הכניסה כאשר הן hashed ובאופן דומה הסיסמה במסד הנתונים רשומה בטקסט נקי ולא מסווג. במימוש מאובטח יותר נרצה להקשיח את האבטחה הזאת ולהעלות אותה למימוש מאובטח יותר
* ניתן למזג את חלון עדכון המשתמשים עם חלון יצירת המשתמשים ע"י הוספת האפשרות ליצור מנהל באופן ישיר במערכת ולא להצטרך ליצור אותו תחת הגדרה "אחרת" כלשהי ולאחריה לעדכן אותו. הפתרון הפשוט ביותר יהיה הסרה של הפונקציונליות הזאת והוספת ההנחה שלבית ספר יכול להיות רק מנהל יחיד שממונה ומוחלף בעת יצירת המערכת, או לחילופין שאם נוצר משתמש מסוג מנהל אז המשתמש שיצר אותו נחסם מכניסה

# דיאגרמות

## דיאגרמת מסד נתונים

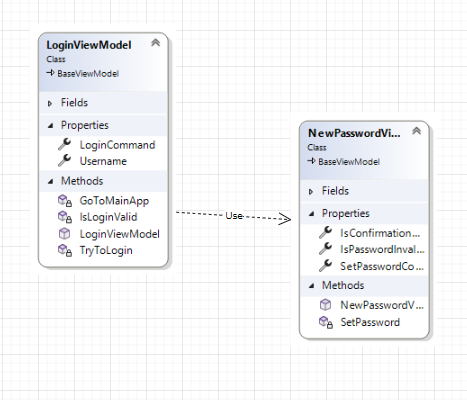


## דיאגרמות עבור שכבת ה-Business logic

הערות:

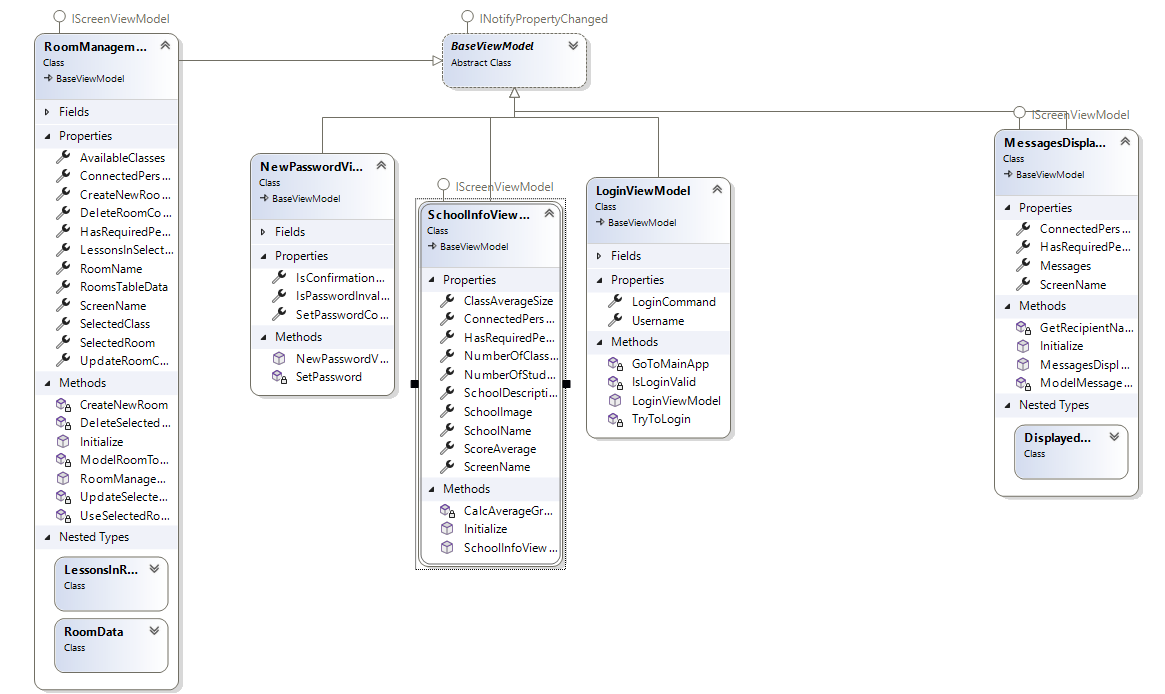
1. ApplicationViewModel משתמשת בכל המחלקות למעט המחלקות הקשורות להתחברות (שיייוצגו בדיאגרמה נפרדת) שכן החלון הראשי יכול פוטנציאלית לקשר לכולן ולכן לא נציג את הקשר הזה בנפרד
2. מטעמי נוחות נחלק את הלוגיקה למספר דיאגרמות שונות של שימוש שכן לא נוכל להכניס את הכל לדיאגרמה מרכזית אחת
3. לא נכליל את RelayCommand שכן מהותה לתקשר בין שכבת הלוגיקה לשכבת התצוגה והיא אינה חיונית לתפקודה של הלוגיקה (היא משמשת לשם המרת פונקציות לממשקים עבור קריאות ב-Binding)

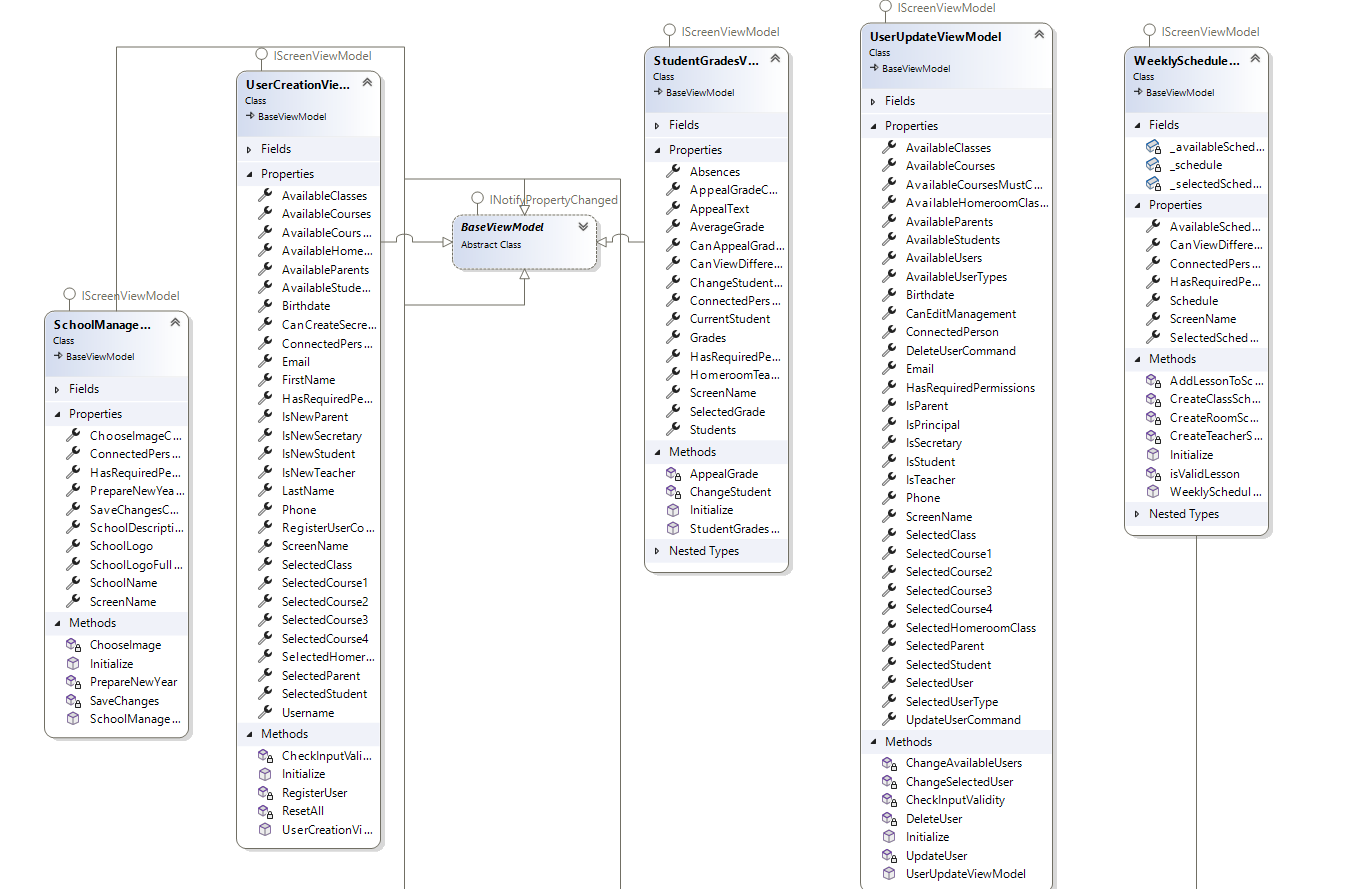
### לוגיקת התחברות



הליך ההתחברות מאמת את המשתמש כנגד מסד הנתונים ובמידה ויש צורך מפנה לחלון יצירת הסיסמה החדשה

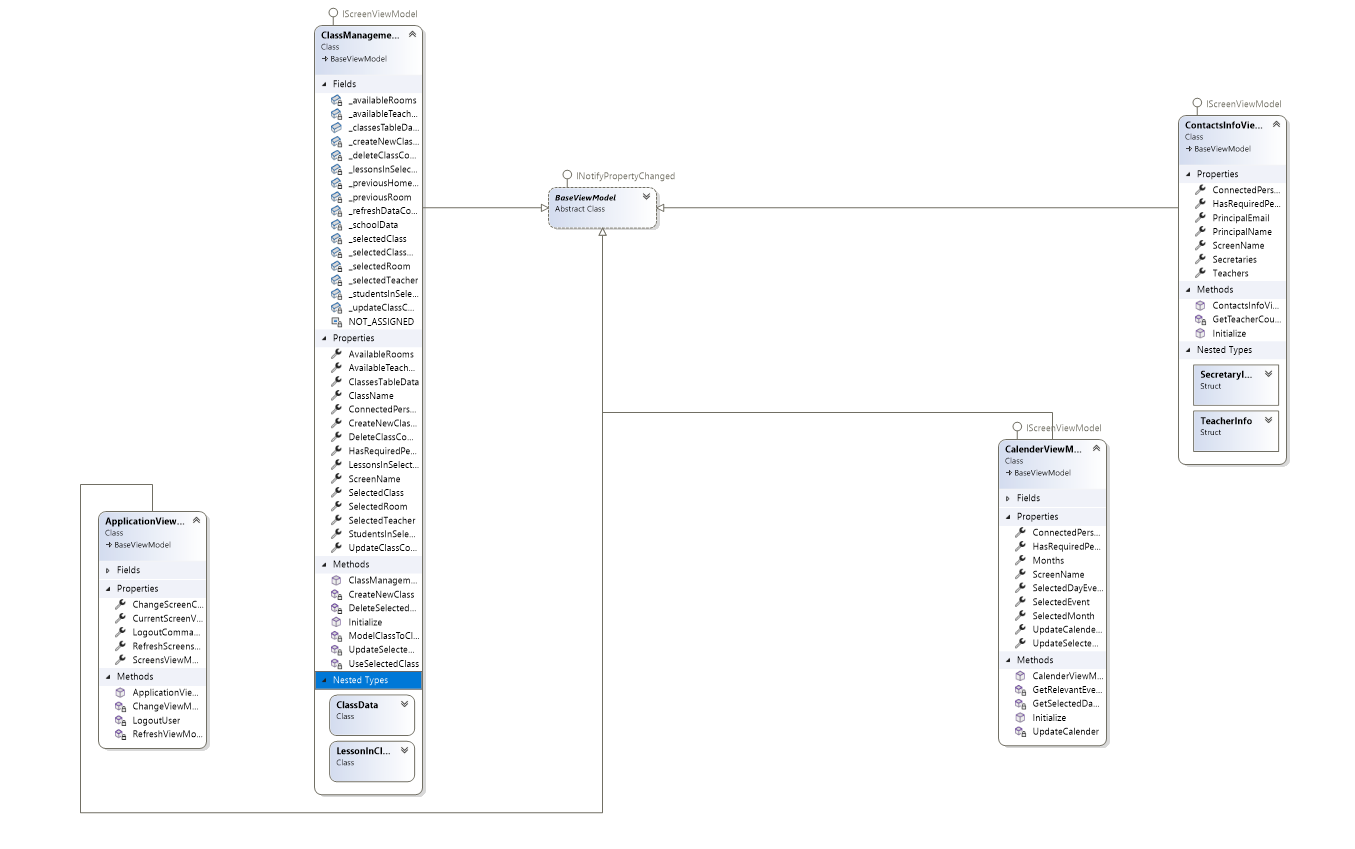
### דיאגרמת מחלקות עבור שכבת הלוגיקה





תמונה שמכילה טקסט, צילום מסך, מקביל, מספר

התיאור נוצר באופן אוטומטי



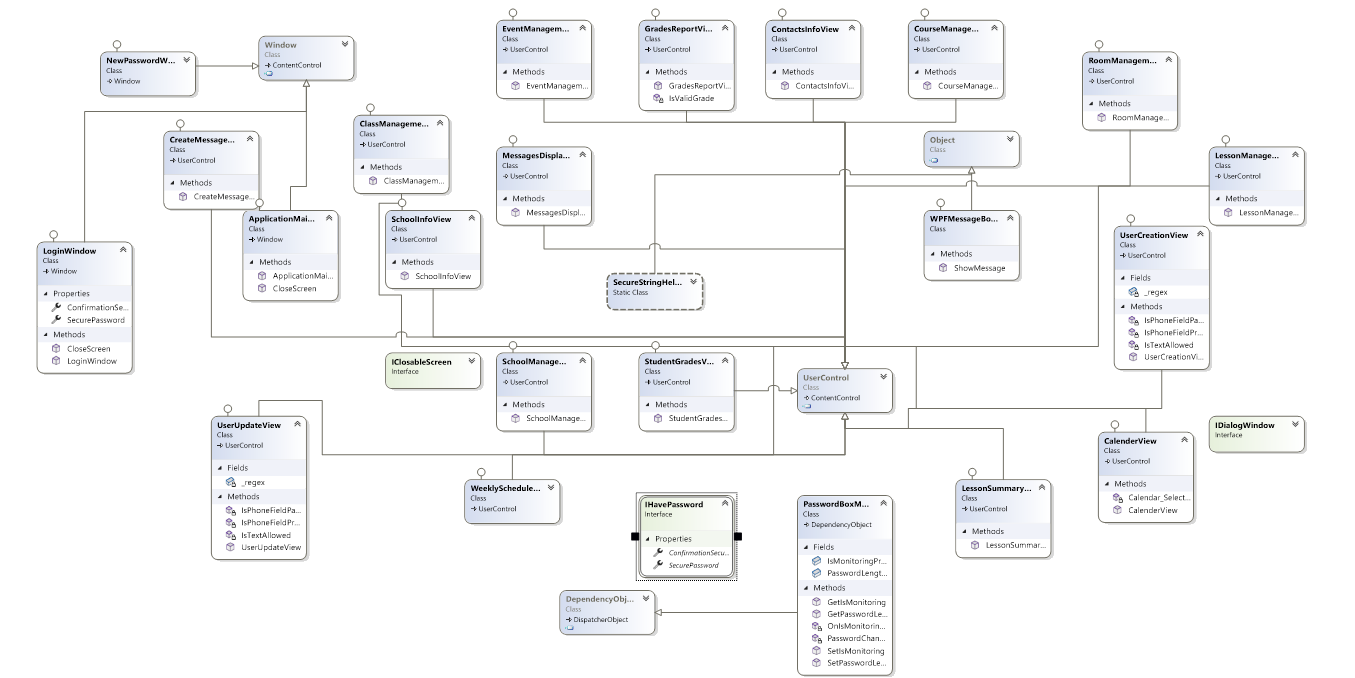
### הסבר מילולי לדיאגרמות

כל המחלקות הלוגיות יורשות מBaseViewModel, שמהותו לממש את הממשק InotifyPropertyChange ולהקפיץ אירועיי עדכון מידע, ואת הממשק IscreenViewModel שמהותו לממש את לוגיקת איתחול המשתמש במסך ולממש את הפונקציונליות של המשתמש במסך. מעבר לכל אלו, ולעובדה שApplicationViewModel מממשת את הלוגיקה של כל החלונות חוץ מהחלונות שקשורים להתחברות, דיאגרמת המחלקות שלנו שטוחה מאוד שכן כל מחלקה מסוגלת לפעול באופן עצמאי כמעט ולחלוטין ומתקשרת עם מסד הנתונים בדרכים שעוקפות את הפונקציונליות של המחלקות אחרות, כמו למשל מבני עזר (struct) או אובייקטים מיוחדים שאוספים מידע מtables שונים במסד הנתונים בכדי לשמור על האוטונומיות שלהם.

היתרון בשיטת העבודה הזאת היא שקל מאוד להחליף ולמחוק מחלקות שלמות (שכן התלות שלהן אלו באלו מינימלית מבחינה לוגית וכל מחלקה מתעדכנת באופן עצמאי במסד הנתונים, שלא קשור ישירות ללוגיקה התפעולית של הקוד) והחיסרון הוא שימוש נרחב יותר במתודות פרטיות ומבני עזר בכדי לשמר את הביזור הנ"ל

## דיאגרמה עבור שכבת הpresentation

שכבת התצוגה פועלת באופן דומה לשכבת הלוגיקה בכך שלא קיימות תלויות פנימיות בין המסכים למעט התלות שבין ApplicaitonView.xaml לשאר המסכים וכל מסך מתנהל בנפרד ועצמאית למלא קבצי תמיכה מסוג Utility



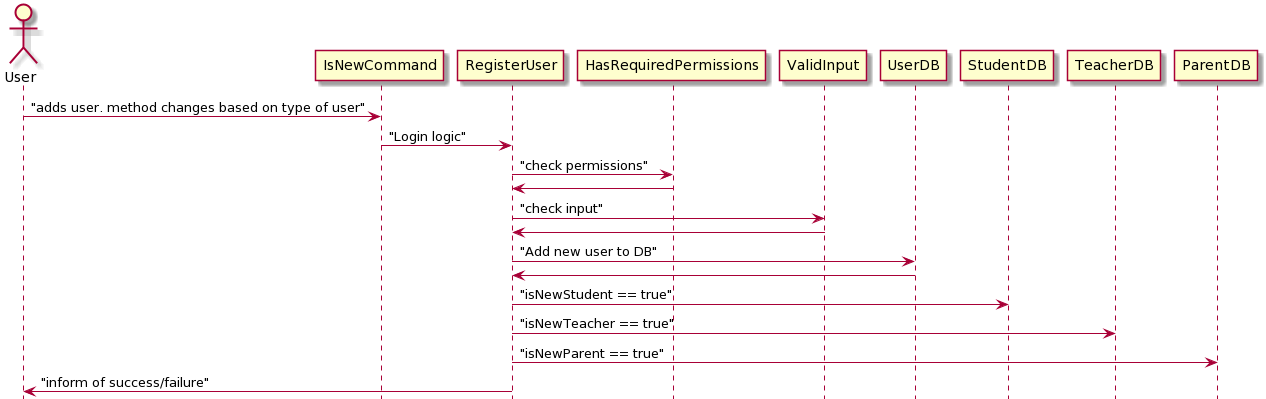
# דיאגרמות רצף

## דיאגרמת התחברות

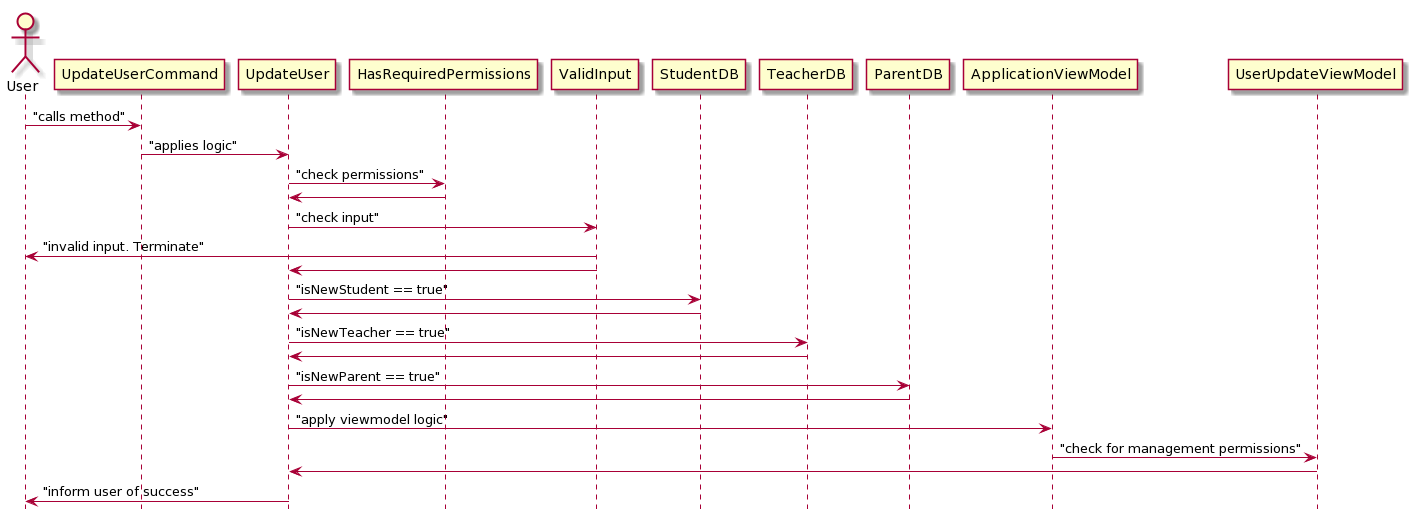
תמונה שמכילה טקסט, צילום מסך, קו, גופן

התיאור נוצר באופן אוטומטי

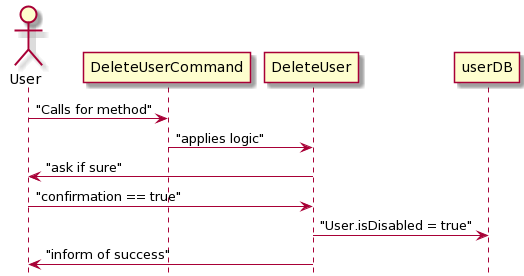
## דיאגרמת יצירת משתמש



## דיאגרמת עדכון משתמש



## דיאגרמת מחיקת משתמש



## דיאגרמת אירועים

תמונה שמכילה טקסט, צילום מסך, גופן, קו

התיאור נוצר באופן אוטומטי

## דיאגרמת הוספה/עדכון של ציון

