Queue & Hi



**מסמך תכנון ועיצוב**

גרסה 1.1

שמות הסטודנטים:

נדב נאור

תומר שובל

# תוכן העניינים

[תוכן העניינים 2](#_Toc420886256)

[תאור כללי של המערכת 7](#_Toc420886257)

[הנחות עבודה בכתיבת הפרוייקט 7](#_Toc420886258)

[מוסכמות רישום 8](#_Toc420886259)

[ספריות Common 9](#_Toc420886260)

[ממשק Ipost 9](#_Toc420886261)

[מחלקת Question : IPost 9](#_Toc420886262)

[מחלקת Answer : IPost 10](#_Toc420886263)

[מחלקת DiscussionThread 10](#_Toc420886264)

[מחלקת UserInfo 10](#_Toc420886265)

[מחלקת UserCredentials 11](#_Toc420886266)

[מחלקת AuthenticationToken 11](#_Toc420886267)

[מחלקות OperationResult 11](#_Toc420886268)

[מחלקת OperationResult הבסיסית 11](#_Toc420886269)

[מחלקת OperationResult המורחבת 11](#_Toc420886270)

[מחלקת AuthenticatedOperation<TPayload> 12](#_Toc420886271)

[מרחב Services 12](#_Toc420886272)

[ממשק IUserServices 12](#_Toc420886273)

[הוספת משתמש חדש 12](#_Toc420886274)

[ביצוע Log In 12](#_Toc420886275)

[קבלת פרטי חשבון משתמש 12](#_Toc420886276)

[שמירת פרטי משתמשים 13](#_Toc420886277)

[ממשק IPostServices 13](#_Toc420886278)

[הוספת שאלה 13](#_Toc420886279)

[מחיקת שאלה 13](#_Toc420886280)

[הוספת תשובה 13](#_Toc420886281)

[מחיקת תשובה 13](#_Toc420886282)

[העלאת דרוג לשאלה 13](#_Toc420886283)

[העלאת דרוג לתשובה 13](#_Toc420886284)

[הורדת דרוג לשאלה 13](#_Toc420886285)

[הורדת דרוג לתשובה 13](#_Toc420886286)

[המלצה על שאלה 13](#_Toc420886287)

[ממשק IPostQueries 14](#_Toc420886288)

[חיפוש חופשי 14](#_Toc420886289)

[חיפוש לפי תגיות 14](#_Toc420886290)

[שליפת השאלות שלי 14](#_Toc420886291)

[שליפת DiscussionThread ספציפי 14](#_Toc420886292)

[מרחב Notifications 14](#_Toc420886293)

[ממשק INotificationService 14](#_Toc420886294)

[סוגי התרעות 14](#_Toc420886295)

[מחלקת Notification 15](#_Toc420886296)

[מרחב Logic 15](#_Toc420886297)

[מרחב Validators 15](#_Toc420886298)

[ממשק IValidator 15](#_Toc420886299)

[המתודה IsValid 16](#_Toc420886300)

[מחלקה אבסרקטית ValidatorDecorator 16](#_Toc420886301)

[מרחב Validations.User 16](#_Toc420886302)

[מחלקת EmailAddressValidator : ValidatorDecorator<UserInfo> 16](#_Toc420886303)

[מחלקת NameValidator : ValidatorDecorator<UserInfo> 16](#_Toc420886304)

[מחלקת PasswordValidator : ValidatorDecorator<string> 16](#_Toc420886305)

[מחלקת RankDownValidator : ValidatorDecorator<UserInfo> 16](#_Toc420886306)

[מחלקת RecommendQuestionValidator : ValidatorDecorator<UserInfo> 16](#_Toc420886307)

[מרחב Validations.Question 17](#_Toc420886308)

[מחלקת ContentValidator : ValidatorDecorator<IPost> 17](#_Toc420886309)

[מחלקת TitleValidator : ValidatorDecorator<Common.Question> 17](#_Toc420886310)

[מחלקת MarkRightAnswerValidator : ValidatorDecorator<Answer> 17](#_Toc420886311)

[שכבת הגישה לבסיס הנתונים – שכבת ה-DAL 18](#_Toc420886312)

[עקרונות מנחים 18](#_Toc420886313)

[מחלקות כלליות 18](#_Toc420886314)

[Converter 18](#_Toc420886315)

[Exceptions 18](#_Toc420886316)

[Exceptions.DALConnectionError 19](#_Toc420886317)

[Exceptions. DALDuplicateKeyError 19](#_Toc420886318)

[Exceptions. DALZeroRowsAffectedError 19](#_Toc420886319)

[Exceptions. DALConstraintError 19](#_Toc420886320)

[מחלקות לגישה לבסיס הנתונים 19](#_Toc420886321)

[שירותי משתמש 19](#_Toc420886322)

[שירותי POST 20](#_Toc420886323)

[שירותי התרעות Notifications 21](#_Toc420886324)

[טבלאות מאגר הנתונים 22](#_Toc420886325)

[טבלת ההתראות notifications 22](#_Toc420886326)

[טבלת דירוגי התשובות answer\_rankings 22](#_Toc420886327)

[טבלת התשובות answers 23](#_Toc420886328)

[טבלת דירוגי השאלות question\_rankings 23](#_Toc420886329)

[טבלת השאלות questions 23](#_Toc420886330)

[טבלת התגיות tags 24](#_Toc420886331)

[טבלת המשתמשים users 24](#_Toc420886332)

[שכבת הלוגיקה 25](#_Toc420886333)

[עקרונות מנחים בשכבת הלוגיקה: 25](#_Toc420886334)

[מחלקות כלליות 25](#_Toc420886335)

[AuthTokenCache 25](#_Toc420886336)

[IAuthTokenSerializer 26](#_Toc420886337)

[AuthTokenSerializer 26](#_Toc420886338)

[INotificationAggregator 27](#_Toc420886339)

[NotificationAggregator 27](#_Toc420886340)

[NotificationService 27](#_Toc420886341)

[PostQueries 27](#_Toc420886342)

[PostServices 28](#_Toc420886343)

[UserServices 29](#_Toc420886344)

[שכבת התצוגה 30](#_Toc420886345)

[עקרונות מנחים בשכבת ההצגה: 30](#_Toc420886346)

[מודלים 30](#_Toc420886347)

[AbstractPost 30](#_Toc420886348)

[DelegateCommand 30](#_Toc420886349)

[AnswerModel 31](#_Toc420886350)

[QuestionInfo 31](#_Toc420886351)

[QuestionModel 31](#_Toc420886352)

[UserAccountModel 32](#_Toc420886353)

[NotificationsObserver 32](#_Toc420886354)

[NewNotificationsEventArgs 32](#_Toc420886355)

[AuthenticationTokenSingleton 32](#_Toc420886356)

[DiscussionThreadObserver 33](#_Toc420886357)

[NewDiscussionThreadVersionEventArgs 33](#_Toc420886358)

[NavigationManager 33](#_Toc420886359)

[NavigationRequestedEventArgs 34](#_Toc420886360)

[App.xaml 34](#_Toc420886361)

[MainWindow 34](#_Toc420886362)

[MainMenu.xaml 34](#_Toc420886363)

[MainMenuViewModel 35](#_Toc420886364)

[MainToolbar.xaml 35](#_Toc420886365)

[MainToolbarViewModel 35](#_Toc420886366)

[NewAnswer.xaml 35](#_Toc420886367)

[NewAnswerViewModel 35](#_Toc420886368)

[NewQuestion.xaml 36](#_Toc420886369)

[NewQuestionViewModel 36](#_Toc420886370)

[QuestionList.xaml 36](#_Toc420886371)

[QuestionListViewModel 37](#_Toc420886372)

[QuestionView.xaml 37](#_Toc420886373)

[QuestionViewModel 37](#_Toc420886374)

[AnswerView.xaml 38](#_Toc420886375)

[AnswerViewModel 38](#_Toc420886376)

[RegistrationControl.xaml 38](#_Toc420886377)

[RegistrationViewModel 38](#_Toc420886378)

[UserManagementView.xaml 39](#_Toc420886379)

[UserManagementViewModel 39](#_Toc420886380)

[NotificationsView.xaml 39](#_Toc420886381)

[NotificationsViewModel 39](#_Toc420886382)

[Styles.xaml 39](#_Toc420886383)

[דפוסי עיצוב - Design Patterns 40](#_Toc420886384)

[MVVM – Model-View-ViewModel 40](#_Toc420886385)

[Singleton 40](#_Toc420886386)

[Observer 40](#_Toc420886387)

[DiscussionThreadObserver 40](#_Toc420886388)

[NotificationsObserver 41](#_Toc420886389)

[Command 41](#_Toc420886390)

[Decorator 41](#_Toc420886391)

[שינויים והרחבות עתידיים 42](#_Toc420886392)

[נספח: דיאגרמות 44](#_Toc420886393)

[דיאגרמת מסד הנתונים (Data Base Diagram) 44](#_Toc420886394)

[דיאגרמת מחלקות 45](#_Toc420886395)

[מחלקות כלליות בשכבת ה-DAL 45](#_Toc420886396)

[מחלקות הגישה לבסיס הנתונים בשכבת ה-DAL 45](#_Toc420886397)

[מחלקות הפעילות בשכבת ה-DAL 46](#_Toc420886398)

[מחלקות ספריית Common 47](#_Toc420886399)

[מחלקות מרחב Common.Logic 48](#_Toc420886400)

[מחלקות שכבת ה-BL 49](#_Toc420886401)

[מחלקות בשכבת התצוגה 50](#_Toc420886402)

[דיאגרמות רצף - Sequence Diagram 51](#_Toc420886403)

[דיאגרמת רצף עבור תהליך Login 51](#_Toc420886404)

[דיאגרמת רצף תהליך ולידציה של משתמש (האם המשתמש הנוכחי ביצע login מסודר למערכת) 52](#_Toc420886405)

[דיאגרמת רצף כתיבת תשובה 53](#_Toc420886406)

[דיאגרמת רצף מנגנון עדכון התצוגה 54](#_Toc420886407)

# תאור כללי של המערכת

המערכת בנויה במתודולוגיית מודל N-Tier, כך שקיימת הפקרדה בין השכבות השונות במערכת.

* שכבת הגישה לבסיס הנתונים – ה-Data Access Layer, או בקיצור DAL
* שכבת הלוגיקה העסקית – ה- Business Layer או בקיצור BL
* שכבת התצוגה – ה- Presentation Layer או בקיצור PL
* ספריות Common – מכילות את האובייקטים והממשקים המשותפים לשלוש השכבות

כל שכבה מתקשרת עם השכבות שמעליה או מתחתיה באמצעות ממשקים מוגדרים מראש. הפרדה זו והשימוש בממשקים מאפשר לנו לעשות שינויים בתוך אחת השכבות בצורה שקופה לשכבות האחרות. זה מקל על שינויים עתידיים בקוד, שיכולים לנבוע מהחלפת טכנולוגיות, שינויי לוגיקה, התרחבות וכד'.

ספריות Common מכילות את האובייקטים והממשקים ששלוש השכבות צריכות להכיר. לדוגמא אובייקט User – הוא מוצג בשכבת התצוגה, מטופל בשכבת הלוגיקה, ומאוחסן בשכבת הגישה לנתונים. כך כל שכבה כוללת reference אליו, ולשכבות איתן היא עובדת.

# הנחות עבודה בכתיבת הפרוייקט

1. ממשק התצוגה מאפשר למשתמש אחד להיות מחובר.
2. התקשורת בין ממשק התצוגה לשכבה הבאה – ה-BL – מבוצע באמצעות WCF. באופן זה ניתן לפתוח מופעים רבים של המערכת, על מחשבים שונים, ללא הגבלה (לוגית).
3. המערכת משמשת סטודנטים בקורס באוניברסיטה. ככזו ביצוע Post-ים למערכת אינו עובר תהליך של moderation, תחת ההנחה כי לאוניברסיטה מנגנונים חיצוניים להתמודד עם משמעת הסטודנטים.
4. לכל קורס יש חשבון מרכז הוראה יחיד.
5. סטודנט במערכת לא יכול להיות גם מרצה או מרכז באותו הקורס.
6. הגדרת משתמש כמרצה תבוצע ע"י מרכז ההוראה בלבד.
7. במסכי החיפוש, כל עוד לא הוכנסו ערכים בפילטרים השונים – יוצגו כל התוצאות כאילו בוצעה שליפה ללא פילטרים.
8. בכל מקום בו המשתמש יכול להזין נתונים, המערכת מוודאה כי שדות חובה מולאו ומבצעת Input validation בשדות בהן נדרש לעשות כן.
9. הטקסט המוזן בשדות תוכן (content) הוא טקסט פשוט, ללא פירמוט/צבעים וכד'.

# מוסכמות רישום

1. כל שם מחלקה, שיטה, Property, קבוע, ו Enum נכתבים בPascalCasing כלומר עם אות גדולה בתחילת כל מילה.
2. לממשקים נוסיף את האות "I" לפני שם המחלקה.
3. על שדה פרטי (private member) יהיה בcamelCasing כלומר, אות ראשונה קטנה וכל אות ראשונה של מילה נוספת תהיה גדולה.
4. כל קריאה לprivate member ממחלקה תיקרא על ידי שימוש במילת המפתח this. לדוגמה:

this.privateMemberName

1. פרמטרים לשיטה או משתנים לוקאלים בתוך שיטה יתחילו באות קטנה וכל מילה נוספת מתחילה באות גדולה.

# ספריות Common

ספריית Common היא הבסיס למודלים באפליקציה. אין לה reference לאף ספרייה אחרת, אך כל הספריות, המייצגות את כל השכבות באפליקציה, מכירות אותה. אובייקטים נשלפים ממאגר הנתונים בעזרת ה-DAL ומוסבים לאוייבקטים שמועברים לשכבת ה-BL ומשם לשכבת התצוגה. שלושת השכבות צריכות לדבר באותה שפה, כלומר להכיר את אותם האובייקטים.

אובייקטים אלו מוגדרים בספריית Common.

מסמך זה יתאר את המחלקות המרכיבות אותם ואת הקשרים והממשקים בינהם.

## ממשק Ipost

ממשק זה משמש כממשק המשותף לשאלות ולתשובות במערכת. מטרתו לוודא כי המחלקות המתארות שאלות ותשובות חושפות את ה-member-ים החיוניים לפעילות המערכת.

string Content { get; }

תוכן ה-post (שאלה או תשובה) – התוכן לקריאה בלבד, שכן תוכן ה-post נקבע עם יצירתו ולא משתנה יותר.

RankingHistory Ranking { get; set; }

היסטוריית דירוג ה-post אשר מושפע ע"י המשתמשים במערכת

int AuthorID { get; }

מזהה כותב ה-post. נקבע עם יצירת ה-post ואינו משתנה.

int ID { get; }

מזהה ה-post אשר נקבע עם יצירת ה-post. זה המפתח הייחודי של השאלות/התשובות במערכת, אשר ישמש לכל אינטראקציה עם ה-post.

## מחלקת Question : IPost

מחלקה זו מממשת את ממשק Ipost. המחלקה מייצגת את אובייקט ה"שאלה".

בנוסף ל-properties הנדרשים מעצם מימוש הממשק כוללת את התכונות הנוספות הבאות

public string Title { get; set; }

כותרת השאלה. מחרוזת המכילה את נושא השאלה. משפט או שניים שמתארים את השאלה. כאשר השאלה מוצגת ברשימת שאלות, למשל, רק כותרת השאלה תוצג (ללא גוף השאלה).

public IEnumerable<string> Tags { get; set; }

התגיות המזהות את הקטגוריות אליהן השאלה משתייכת. כל שאלה יכולה להשתייך למספר קטגוריות. הסיווג לקטגוריות מאפשר לבצע חיפושים מותאים כדי לאתר שאלות הקשורות לנושא ספציפי, למשל.

ממומש כ-Ienumerable כדי לאפשר למתכנת לבחור איך לממש את הרשימה.

public bool IsRecommended { get; set; }

האם השאלה מומלצת. משתמשים במערכת יכולים להצביע על שאלה כשאלה מומלצת, כדי לבדלה מהשאלות האחרות שפורסמו ע"י משתתפי הקורס.

public int? RightAnswerId { get; set; }

מזהה התשובה הנכונה. לכל שאלה תשובה אחת שתוגדר כתשובה הנכונה, וזאת כדי להקל על משתתפי הקורס המשתמשים במערכת ולמנוע מצב בו סטודנט השתמש בתשובה לא נכונה שניתנה ע"י משתתף אחר בקורס.

## מחלקת Answer : IPost

מחלקה זו מייצגת את אובייקט ה"תשובה" ומממשת את Ipost. בנוסף למאפיינים הנדרשים ממימוש הממשק, ממשת המחלקה את מאפיין נוסף:

public int RelatedQuestionId { get; set; }

הקישור בין התשובה לשאלה עליה היא מנסה לענות.

## מחלקת DiscussionThread

מחלקה זו מאגדת את השאלה וכל התשובות שניתנו לה לכדי אובייקט אחד. כך ניתן להתייחס לאוסף זה כאובייקט לוגי אחד ולהקל בתפעולו (למשל כדי למחוק אותו).

המחלקה כוללת את השאלה, IEnumerable של התשובות, ומספר גרסה.

מספר הגרסה משמש לצרכי עדכון תצוגה וכדי שהמערכת תדע להבדיל בין גרסה עדכנית לגרסה לא עדכנית של ה-Discussion.

public Question Question { get; set; }

השאלה אשר פותחת כל Discussion.

public List<Answer> Answers { get; set; }

התשובות הקיימות עבור השאלה הנוכחית

public int Version { get; set; }

מזהה גרסת ה-Discussion, אשר גדל באחד עבור כל שינוי שנעשה בתשובות או בשאלה.

## מחלקת RankingHistory

מחלקה זו מתארת את היסטוריית הדירוגים של פוסט מסוים. המחלקה יורשת מList של RankingEntry.

public int OverallRanking { get; }

מחשב את סה"כ הדירוג של הפוסט הנוכחי

## מחלקת RankingEntry

מחלקה זו מתארת מופע בודד של דירוג משתמש כלשהו של שאלה\תשובה.

public int UserID { get; }

המזהה של המשתמש שביצע את הדירוג

public RankingType RankingType { get; }

סוג הדירוג שביצע המשתמש על הפוסט

## מחלקת RankingType

זהו enum שמסמל את סוגי הדירוג האפשריים.

יש לenum זה שני ערכים: Up, Down

## מחלקת UserInfo

פרטי המשתמש - מחלקה זו מכילה את כל המאפיינים המתארים את חשבון המשתמש במערכת, למעט הסיסמא, אשר נמצאת במחלקת UserCredentials המתוארת בהמשך.

public int Ranking { get; set; }

דירוג המשתמש. מאפיין מספרי הבא לציין את "דרגתו" של המשתמש במערכת. פעילות המשתמש במערכת מזכה אותו בדירוג מסויים, אשר מאפשר לו לבצע פעולות מסויימות, כפי שנקבע בשכבת ה-Business Logic.

public string Username { get; set; }

שם המשתמש במערכת

public string EmailAddress { get; set; }

כתובת המייל של המשתמש במערכת. הכתובת משמשת גם לזיהוי חד חד ערכי של המשתמש, וגם להרחבות עתידיות – כדי לשלוח למשתמש התראות בזמן אמת, או בדו"חות תקופתיים לגבי פעילותו או פעילות אחרים במערכת הקשורה לשאלות ששאל או לתשובות שענה לשאלות אחרים.

public bool IsAdmin { get; set; }

האם המשתמש הוא מנהל בקורס, כלומר מרצה/רכז אשר לו הרשאות נרחבות יותר מאשר של משתמש סטודנט (כגון מחיקת שאלות).

public bool IsMuted { get; set; }

האם חשבון המשתמש מושתק? למרצה בקורס מותר להשתיק סטודנט אשר עבר על כללי ההתנהגות במערכת, שאל שאלות או ענה תשובות בצורה שאינה מקובלת על צוות הקורס. פעולותיו של סטודנט שהושתק יהיו מוגבלות ע"י המערכת, עד אותו מועד בו יבחר המרצה לבטל את ההשתקה.

public int ID { get; set; }

מזהה חד חד ערכי של המשתמש במערכת.

## מחלקת UserCredentials

מחלקה זו כוללת את שני המאפיינים שם משתמש וסיסמא. במחלקה זו נעשה שימוש כדי לשמור את פרטי המשתמש שכבר הזדהה מול המערכת, כדי לוודא שלכל אורך פעילותו במערכת ניתנו לו הרשאות המתאימות לסוג המשתמש (סטודנט/מרצה/רכז) ולדירוגו.

שם המשתמש והסיסמא נקבעים בבונה של המחלקה, ולא ניתנים לשינוי לאחר מכן (מבחוץ), אלא לקריאה בלבד

public string Username { get; private set; }

שם המשתמש

public string Password { get; private set; }

סיסמת המשתמש

שם המשתמש והסיסמא משמשים ליצירת ה-AuthenticationToken – המזהה שישמר בצד המערכת כדי לוודא מי הוא המשתמש עבור כל פעולה ב-BL.

## מחלקת AuthenticationToken

זהו ה-token שנוצר לאחר ביצוע login. המערכת שומרת אותו ומוודאה בכל פעילות של המשתמש כי קיים token וכי הוא זהה לזה השמור במערכת. כך נמנע מצב שמשתמש לא מורשה יבצע פעולות במערכת, או שמשתמש מורשה יבצע פעולות אשר אינו מורשה לבצע.

public string TokenString { get; private set; }

המחלקה כוללת מאפיין אחד – מחרוזת ה-token הנקבעת ע"י בונה המחלקה ולאחר מכן זמינה לקריאה בלבד ע"י משתמש המחלקה

## מחלקת AuthenticatedIdentity

מחלקה שמייצגת זיהוי כלשהו של משתמש עבור client מסויים.

אובייקט זה יוחזק על ידי הPL והוא ישמור את הAuthenticationToken הנוכחי ואת הUserId שלו כדי שיוכל לבצע שאילתות על פרטי המשתמש.

public int UserID { get; set; }

public AuthenticationToken Token { get; set; }

## מחלקות OperationResult

מחלקה המשמשת כ-return type של פעולות במערכת.

המטרה היא לאפשר קבלת תשובה לגבי ביצוע פעולה. האם הביצוע הצליח, ואם לא למה, או במידה והביצוע הצליח, לכלול מידע שחוזר עם הביצוע.

לוגיקה זו אינה מאפשרת להחזיר ערך בוליאני, או להחזיר את המידע שפעולה מוצלחת מחזירה ובמקרה כישלון להחזיר null ולכן יש צורך בהחזרת complex data type כזה.

### מחלקת OperationResult הבסיסית

public bool IsSuccessful { get; private set; }

מאפיין בוליאני המתאר אם הפעולה הצליחה או נכשלה

public ICollection<string> ErrorMessages { get; private set; }

רשימת הודעות השגיאה שהתקבלו. יכולה להיות יותר משגיאה אחת. למשל, בעת יצירת משתמש, יכול להיות ששם המשתמש תפוס, וכן שכתובת ה-mail שסופקה אינה חוקית. במקום להצריך כמה הפעלות חוזרות, ולשלוח את המידע שוב ושוב אל צד השרת, ניתן לבצע ולידציה של כלל הרכיבים ולהחזיר הודעות שגיאה מרובות מתאימות.

### מחלקת OperationResult המורחבת

public class OperationResult<TPayload> : OperationResult

מחלקה זו, היורשת מ-OperationResult הבסיסית, כוללת גם Payload – מחזיק מקום גנרי שאמור לכלול את התוצר של הפעלת ה-operation הרלוונטי. למשל, במקרה של ביצוע Login מוצלח, ה-OperationResult שיוחזר יכלול גם את IsSuccessful=true, גם אפס הודעות שגיאה, וגם Payload שיכלול את ה-AuthneticationToken הרלוונטי לפעולה.

## מחלקת AuthenticatedOperation<TPayload>

כאשר משתמש במערכת מבצע פעולה, יש לוודא בכל פעם כי מדובר במשתמש מורשה אשר ביצע לוגין מסודר למערכת ואשר יש לו הרשאות לביצוע הפעולה.

במקום להעביר לכל מתודה את הפרמטרים שלה, יש להעביר אובייקט AuthenticatedOperation אשר יכלול את ה-AuthenticationToken ואת ה-payload כאשר ה-payload הוא generic placeholder שיכלול את הפרמטרים שיש להעביר.

הבנאי של המחלקה מקבל את ה-payload ואת ה-AuthenticationToken אשר זמינים לאחר מכן לקריאה בלבד לטובת ביצוע הפעולה

public AuthenticationToken Token { get; private set; }

public TPayload Payload { get; private set; }

# מרחב Services

מרחב זה מכיל את ממשקי השירותים הזמינים במערכת.

כל ממשק מכיל את המתודות שיש לממש במערכת, בחלוקה לשלוש מחלקות

## ממשק IUserServices

ממשק זה מכיל את כל הפעולות שניתן לבצע על משתמש במערכת.

ממשק זה משמש את השירותים במוגדרים במערכת ב-wcf כדי לאפשר למערכת לבצע פעולות מול מאגר הנתונים.

כל המתודות מוגדרות כ-OperationContract כדי להגדיר כי מתודות אלו מגדירות פעולות אשר הינן חלק מ"חוזה" המוגדר ב-wcf

### הוספת משתמש חדש

OperationResult AddNewUser(UserInfo newUser, UserCredentials userCredentials);

מתודה זו מאפשרת להוסיף משתמש חדש למערכת

### ביצוע Log In

OperationResult<AuthenticationToken> LogIn(UserCredentials userCredentials);

מתודה זו מאפשרת לבצע Log in למערכת

### קבלת פרטי חשבון משתמש

OperationResult<UserInfo> GetUserInfo(AuthenticationToken authToken);

מתודה זו מאפשרת למשוך את פרטי חשבון המשתמש ממאגר הנתונים

קבלת פרטי כל המשתמשים במערכת

IEnumerable<UserInfo> GetAllUsersData(AuthenticationToken authToken);

מתודה זו משמשמת לשליפת פרטי כל המשתמשים במערכת. מתודה זו משמשת את מסך ניהול המשתמשים

### שמירת פרטי משתמשים

OperationResult SaveUsersData(AuthenticatedOperation<IEnumerable<UserInfo>> usersData);

מתודה זו משמשת לשמירה של אובייקטי משתמש אל מאגר הנתונים

## ממשק IPostServices

ממשק זה משמש לביצוע פעולות על Post, כלומר על שאלות או תשובות במערכת.

כל הפעולות מבוצעות לאחר בדיקת ההרשאות המתאימות במערכת, ולכן מקבלות כפרמטר אובייקט מסוג AuthenticatedOperation אשר כולל את ה-AuthenticationToken של המשתמש שמבקש לבצע את הפעולה ואת ה-payload – הפרמטרים לביצוע הפעולה.

### הוספת שאלה

void AddQuestion(AuthenticatedOperation<Question> question);

### מחיקת שאלה

void DeleteQuestion(AuthenticatedOperation<int> questionId);

### הוספת תשובה

void AddAnswer(AuthenticatedOperation<Answer> answer);

### מחיקת תשובה

void DeleteAnswer(AuthenticatedOperation<int> answerId);

### העלאת דרוג לשאלה

void VoteUpQuestion(AuthenticatedOperation<int> questionId);

### העלאת דרוג לתשובה

void VoteUpAnswer(AuthenticatedOperation<int> answerId);

### הורדת דרוג לשאלה

void VoteDownQuestion(AuthenticatedOperation<int> questionId);

### הורדת דרוג לתשובה

void VoteDownAnswer(AuthenticatedOperation<int> answerId);

## ביטול דירוג לשאלה

void CancelQuestionRanking(AuthenticatedOperation<int> questionId);

## ביטול דירוג לתשובה

void CancelAnswerVote(AuthenticatedOperation<int> answerId);

## המלצה על שאלה

void RecommendQuestion(AuthenticatedOperation<int> questionId);

## ביטול המלצה על שאלה

void UnrecommendQuestion(AuthenticatedOperation<int> questionId);

## סימון תשובה כנכונה

void MarkAsRightAnswer(AuthenticatedOperation<int> answerId);

## ביטול סימון תשובה כנכונה

void UnmarkAsRightAnswer(AuthenticatedOperation<int> answerId);

## ממשק IPostQueries

ממשק זה משמש להגדיר את הפעולות המוגדרות בחוזה ב-wcf ומשמשות לשליפות נתונים לטובת הצגה במסכים השונים במערכת

גם בממשק זה האובייקט המועבר כפרמטר לכל המתודות הוא אובייקט AuthenticatedOperation הכולל את ה-AuthenticationToken של המשתמש המבקש לבצע את הפעולה ואת ה-Payload המכיל את הפרמטרים לביצוע המתודה, וזאת כדי שב-BL נוכל לבדוק ולוודא שהמשתמש אכן מאושר לביצוע הפעולה המבוקשת.

### חיפוש חופשי

IEnumerable<Question> FreeSearch(AuthenticatedOperation<string> searchString);

מתודה זו מחזירה רשימת שאלות התואמת את פרטי החיפוש שהועברו

### חיפוש לפי תגיות

IEnumerable<Question> TagsSearch(AuthenticatedOperation<IEnumerable<string>> tags);

שימוש במתודה זו מחזיר את כל השאלות במערכת שתויגו לפי רשימת התגיות שהועברה כפרמטר בשליפה

### שליפת השאלות שלי

IEnumerable<Question> GetMyQuestions(AuthenticationToken authToken);

מתודה זו משמשת לשליפת כל השאלות שהמשתמש שאל בעבר במערכת

### שליפת השאלות האחרונות

IEnumerable<Question> GetLatestQuestions();

### שליפת DiscussionThread ספציפי

DiscussionThread GetDiscussionThreadById(AuthenticatedOperation<int> id);

מתודה זו משמשת לשליפה Thread ספציפי לפי המזהה המספרי שלו

# מרחב Notifications

מרחב זה כולל את הממשקים והמחלקות המשמשים להעברת התראות למשתמש. בעוד המשמש מנווט במערכת וצופה בשאלות של חבריו לקורס, או לאחר ביצוע כניסה מחודשת למערכת, על המערכת להציג למשתמש פעולות שבוצעו בהיעדרו, כגון תשובות שנוספו לשאלות ששאל, שינויים בדירוג שלו, סימון תשובות שענה כנכונות וכד'.

## ממשק INotificationService

ממשק זה מכיל מתודה אחת המשמשת לשליפת התרעות במערכת עבור המשתמש הנוכחי

IEnumerable<Notification> GetNotifications(AuthenticationToken authToken);

## סוגי התרעות

המערכת כוללת סוגים שונים של התרעות. תשובה חדשה לשאלה, שינוי בדירוג שאלה/תשובה, תשובה סומנה כנכונה ועוד'.

כל סוגי ההתרעות הקיימים במערכת מוגדרים ב-Enum של סוגי ההתרעות

public enum NotificationType

{

NewAnswer,

QuestionRankedUp,

QuestionRankedDown,

AnswerRankedUp,

AnswerRankedDown,

AnswerMarkedAsRight,

QuestionMarkedAsRecommended,

QuestionMarkedAsUnrecommended

PostWasDeleted,

YouWereMuted,

YouWereUnmuted,

MarkedAsLecturer,

UnmarkedAsLecturer

}

בעת יצירת התרעה חדשה, מגדיר היוצר את סוג ההתרעה. כך ניתן לייצר רק התרעות מסוגים קיימים ומוגדרים במערכת, אשר עבורן מוגדרת וקיימת התנהגות מתאימה.

## מחלקת Notification

מחלקה זו מגדירה את אובייקט ההתרעה במערכת.

האובייקט כולל כמה מאפיינים:

public string Message { get; set; }  
ההודעה עצמה

public DateTime TimeStamp { get; set; }

מועד יצירת ההתרעה

public bool Seen { get; set; }

האם ההתרעה נצפתה ע"י המשתמש אליו יועדה

public UserInfo Recipient { get; set; }  
אובייקט המשתמש אליו ההתרעה מיועדת

public NotificationType Type { get; set; }  
סוג ההתרעה

# מרחב Logic

מרחב זה, הכלול בספרייה נפרדת (Common.Logic) כולל את חוקי הולידציה במערכת. חוקים אלו משמשים את הן את ה-BL והן את שכבת התצוגה כדי לבצע ולידציה לפעולות שהמשתמש מבקש לבצע.

הצורך בולידציה בשני המקומות ברור. פעם אחת בממשק המשתמש – גם כדי להציג למשתמש רק פעולות שמותר לו לבצע, וגם כדי להציג למשתמש הודעות שגיאה על ערכי Input שגויים עוד לפני ששולחים אותם לעיבוד בצד השרת. פעם שניה – בשכבת ה-business logic כדי לוודא שהפעולה שהגיעה לצד השרת אכן מותרת לביצוע למשתמש שביקש לבצעה.

# מרחב Validators

מרחב זה כולל את כל הולידציות שמוגדרות במערכת.

## ממשק IValidator

ממשק זה מגדיר את המתודה שכל אובייקט שיממש את ממשק ה-validator חייב לממש.

### המתודה IsValid

OperationResult IsValid(T data);

משמשת לבדיקת הפעולה אותה מבקשים לבצע. מקבלת את ה-payload הגנרי עליו מתבצעת הבדיקה, ומחזירה OperationResult (ראה הסבר למעלה).

## מחלקה אבסרקטית ValidatorDecorator

מחלקה זו מגדירה את ה-Decorator עבור ממשק ה-IValidator. השימוש ב-design pattern זה מאפשר לממשק את המתודה IsValid בצורה דינאמית בכל מימוש של הממשק. כך אין צורך לשמור את הקוד במקום מרכזי וניתן לבצע הרחבות בצורה דינאמית ולפי הצורך.

בנוסף מחלקה זו מממשת את ה-template method design pattern כך שהיא מגדירה את האלגוריתם הנכון לביצוע הולידציה (כלומר, לבדוק האם הביצוע עובר ולידציה, ואם לא להחזיר את הודעות השגיאה הרלוונטיות) אך משאירה את המימוש הספציפי למחלקות היורשות את המחלקה האבסטרקטית.

המחלקות היורשות יממשו את InternalValidator. כך ישמר האלגוריתם כולו, בלי לפגוע ביכולת של המחלקות היורשות לבצע את הבדיקה בהתאם לצרכיהן.

# מרחב Validations.User

מרחב זה מכיל את הולידציות שיש לבצע על מאפייני משתמש (בעת יצירתו או עדכון של פרטיו) וכן האם מותר לו לבצע פעולות מסויימות.

## מחלקת EmailAddressValidator : ValidatorDecorator<UserInfo>

מבצעת בדיקה של כתובת המייל של המשתמש

OperationResult IsValidInternal(UserInfo userInfo)

## מחלקת NameValidator : ValidatorDecorator<UserInfo>

מבצעת בדיקה של שם המשתמש

OperationResult IsValidInternal(UserInfo userInfo)

## מחלקת PasswordValidator : ValidatorDecorator<string>

מבצעת בדיקה של הסיסמא שנבחרה עבור המשתמש

OperationResult IsValidInternal(string userInfo)

## מחלקת RankDownValidator : ValidatorDecorator<UserInfo>

בודקת האם לשמתמש מותר להצביע "למטה" עבור שאלה/תשובה, כלומר לתת דרוג שלילי לשאלות/תשובות

IsValidInternal(UserInfo userInfo)

## מחלקת RecommendQuestionValidator : ValidatorDecorator<UserInfo>

מבצעת בדיקה האם למשתמש מותר להמליץ על שאלה

OperationResult IsValidInternal(UserInfo userInfo)

## מחלקת UserValidator : ValidatorDecorator<UserInfo>

מבצעת בדיקה האם פרטי המשתמש שהוזנו מתאימים לרישום משתמש חדש.

OperationResult IsValidInternal(UserInfo userInfo)

# מרחב Validations.Question

מרחב זה מכיל את הולידציות שיש לבצע על מאפייני שאלה (בעת יצירת שאלה או עדכון פרטי שאלה) וכן האם מותר לבצע פעולות מסויימות על השאלה

## מחלקת ContentValidator : ValidatorDecorator<IPost>

בודקת האם תוכן ה-post (שאלה או תשובה) תקין.

OperationResult IsValidInternal(IPost userInfo)

## מחלקת TitleValidator : ValidatorDecorator<Common.Question>

בודקת האם ה-Title, הנושא שניתן לשאלה תקין

OperationResult IsValidInternal(Common.Question userInfo)

# שכבת הגישה לבסיס הנתונים – שכבת ה-DAL

השכבה המקשרת בין שכבת הלוגיקה (ה-BL) ובסיס הנתונים עצמו.

השכבה מאפשרת לקוד הפרוייקט לגשת לבסיס הנתונים בצורה שקופה, וללא צורך בהכרת סוג בסיס הנתונים בו נעשה שימוש. השכבה מסתירה את הפרטים הטכניים הקטנים מהמשתמש, ומאפשרת למתכנת לעבוד עם אובייקטים במקום עם טבלאות ורשומות.

השכבה משתמשת ב-Entity Framework לביצוע ההמרה בין האובייקטים הבסיסיים איתם עובדת ADO.NET – DataTables המכילות רשומות, ושאילתות ה-SQL המשמשות לתקשורת עם בסיס הנתונים ושליפת/עדכון אותן טבלאות לאובייקטים אותם היא חושפת למשתמש.

בפרוייקט זה, שכבה זו ממירה את האובייקטים שנוצרים ע"י ה-Entity Framework לאובייקטים המוכרים בשאר הפרוייקט, בספריית Common. אבחנה זו בין Internal Objects – האובייקטים הפנימיים הקיימים ב-EntityFramework ואובייקטים חיצוניים, אליהם פונים שאר רכיבי הפרוייקט, ואותם ניתן לחשוף אף החוצה, לטובת הרחבות עתידיות, מאפשרים להתמודד בקלות עם המקרים בהם האובייקט שנוצר אוטומטית ע"י ה-EntityFramework לא בדיוק מתאים ל-Design שלנו, או מוסיף תכונות ושיטות שאנו לא מעוניינים לחשוף החוצה.

המרה זו מאפשרת להחליף את שכבת ה-DAL בקלות יתרה, כיוון שלא קיים צימוד בין בסיס הנתונים והאובייקטים אותם מייצרת ה-Entity Framework לבין האובייקטים מולם פועל יתר הפרוייקט.

## עקרונות מנחים

השכבה נותנת שירותי שליפה, עדכון והכנסה של נתונים מבסיס הנתונים, באופן שקוף, וללא ביצוע ולידציה, אותנתיקציה או כל שירות אחר הממומש בשכבת הלוגיקה.

שכבה זו, הממומשת כספריה, תימצא על השרת עליו מותקנים רכיבי שכבת הלוגיקה

מאפייני ההתקשרות עם בסיס הנתונים יאוחסנו בקובץ ההגדרות של השכבה, כך שהספרייה לא תכלול את פרטי השרת, בסיס הנתונים וה-credentials המשמשים להזדהות מולו.

כך נוכל לבצע שינויים במיקום בסיס הנתונים (למשל, לעבור לשרת חזק יותר) ללא צורך בהוצאת גרסה חדשה של המערכת

במקרה כישלון כל השיטות בכל המחלקות מחזירות Exception – החריגות יהיו Custom Exceptions כך שגם בהחלפת בסיס הנתונים, מבחינת הקוד בשכבות מעל יוחזרו אותם ה-Exceptions ולא יהיה צורך בשינוי קוד, וההתנהגות תישמר.

## מחלקות כלליות

### Converter

מחלקה המכילה שיטות סטאטיות המשמשות להמרה של אובייקט מ-common לאובייקט של ה-EntityFramework ובחזרה.

השיטות במחלקה כולן מוגדרות Internal ויהיו נגישות רק בתוך ה-DAL

### Exceptions

מחלקה זו מכילה את כל ה-custom exceptions שנועדו לפעפע חריגות בעבודה מול בסיס הנתונים כלפי מעלה. בינהן ניתן למנות את חריגות בשל מידע לא קיים (ניסיון לעדכן רשומה לא קיימת), חריגות בשל כפילויות (ניסיון להכניס רשומה נוספת עם מפתח ראשי זהה לרשומה קיימת) וכן חריגות כלליות שיזרקו בשל אי זמינות של בסיס הנתונים.

כל החריגות הן מחלקות היורשות ממחלקת הבסיס Exception ומממשות את שלושת הקונסטרקטורים הנפוצים:

Exception(), Exception(String message), Exception(String message, Exception inner)

### Exceptions.DALConnectionError

חריגה זו נזרקת כאשר יש תקלה/שגיאה בהתחברות לבסיס הנתונים

Message עשוי לכלול פרטים לגבי השגיאה (שרת לא זמין, משתמש וסיסמא לא נכונים וכולי)

Inner יכלול את החריגה המקורית שנזרקה מבסיס הנתונים

### Exceptions. DALDuplicateKeyError

חריגה זו נזרקת כאשר נעשה ניסיון לבצע insert לרשומה עם מפתח ראשי שכבר בשימוש ע"י רשומה קיימת.

Message עשוי לכלול פרטים לגבי השגיאה (איזה ערכים הועברו, וכד')

Inner יכלול את החריגה המקורית שנזרקה מבסיס הנתונים

### Exceptions. DALZeroRowsAffectedError

חריגה זו נזרקת כאשר עדכון (update) שהורץ מול בסיס הנתונים לא עדכן אף רשומה. מצב זה עשוי לקרות כאשר רשומה שעבדנו על השינוי שלה כבר נמחקה ע"י מישהו אחר – מצב תקין לגמרי. לדוגמא – כאשר נרצה להוסיף תשובה לשאלה ועד שנגמור לערוך את התשובה רכז הקורס כבר מחק את השאלה.

### Exceptions. DALConstraintError

חריגה זו נזרקת כאשר שינוי כלשהו בנתונים (בין אם בהכנסה ובין אם בעדכון) גורם לסטייה מהגבולות המוגדרים בבסיס הנתונים. למשל – אם מנסים להכניס רשומה חדשה בלי לציין ערכים לשדות שלא מאפשרים ערכי null.

## מחלקות לגישה לבסיס הנתונים

### שירותי משתמש

המחלקה מאפשרת פעולות הקשורות לחשבון משתמש, כגון יצירת משתמשים, עדכון פרטי משתמשים, ושליפת פרטי התרעות למשתמשים

#### הוספת משתמש חדש

מקבלת אובייקט משתמש חדש ופרטי הזדהות, ומוסיפה אותו לבסיס הנתונים. לאחר ההוספה מחזירה אובייקט משתמש, עם שדה ID לפי המספור האוטומטי בבסיס הנתונים

public UserInfo CreateNewUser(UserInfo userInfo, UserCredentials userCredentials)

#### כניסה למערכת

מקבלת פרטי הזדהות ומחזירה ערך Boolean

public bool Login(UserCredentials userCredentials)

#### שליפת כל משתמשי המערכת

מחזירה את כל המשתמשים הרשומים בטבלת המשתמשים

public IEnumerable<UserInfo> GetAllUsers()

#### שמירת שינויים בחשבון משתמש

public void SaveUserData(UserInfo userinfo)

#### שמירת שינויים ברשימת חשבונות משתמשים

משתמשת בשמירת שינויים בחשבון משתמש עבור כל החשבונות שהועברו

public void SaveUsersData(IEnumerable<UserInfo> usersInfo)

#### שליפת פרטי משתמש

מחזירה את פרטי המשתמש המבוקש

public UserInfo GetUserInfo(int userId)

### שירותי POST

מחלקה זו משמשת לפעולות על posts – כלומר שאלות ותשובות במערכת. שליפת שאלות ותשובות, הוספת שאלות ותשובות ועדכון נתונים לגבי שאלות ותשובות.

כל הפעולות שמשנות אובייקט מבצעות את הפעולה או זורקות Exception במקרה שגיאה.

כל הפעולות שמשנות אובייקט שאלה או תשובה מגדילות את ה-version של השאלה ב-1.

כל שינוי בדירוג של post נשמר גם בטבלאות הדירוגים. כך ניתן לדעת האם משתמש כבר דירג שאלה זו בעבר ולמנוע ממנו לבצע דירוג חוזר לאותו כיוון.

לדוגמא, משתמש שנתן דירוג חיובי לשאלה לא יכול לתת דירוג חיובי נוסף. הוא כן יוכל לשנות את דעתו ולהוריד את הדירוג חזרה לניטראלי, או להורידו שוב לדירוג שלילי. וחוזר חלילה.

במצב זה משתמש לא יוכל לתת דירוג חיובי/שלילי אין סופי לשאלה (להקליק שוב ושוב על "העלה/הורד דירוג").

#### הוספת שאלה

public Question AddQuestion(Question question)

#### מחיקת שאלה

public void DeleteQuestion(int questionId)

#### הוספת תשובה

public Answer AddAnswer(Answer answer)

#### מחיקת תשובה

public void DeleteAnswer(int answerId)

#### סימון שאלה כמועדפת

public void RecommendQuestion(int questionId)

#### ביטול סימון שאלה כמועדפת

public void UnrecommendQuestion(int questionId)

#### הצבעה חיובית לשאלה

public void RankUpQuestion(int questionId)

#### הצבעה שלילית לשאלה

public void RankDownQuestion(int questionId)

#### ביטול דירוג השאלה

public void CancelQuestionRank(int questionId, int userId)

#### הצבעה חיובית לתשובה

public void RankUpAnswer(int answerId)

#### הצבעה שלילית לשאלה

public void RankDownAnswer(int answerId)

#### ביטול דירוג התשובה

public void CancelAnswerRank(int answerId, int userId)

#### סימון תשובה כנכונה

public void MarkAsRightAnswer(int answerId)

#### ביטול סימון תשובה כנכונה

public void UnmarkAsRightAnswer(int answerId)

#### העלאת מספר גרסה לשאלה

public void IncrementVersion(int questionId)

#### חיפוש חופשי

מחזירה את כל השאלות שעונות על פרטי החיפוש, אשר מתבצע בבנושא השאלות ובתוכן השאלות.

public IEnumerable<Question> FreeSearch(string searchString)

#### חיפוש לפי תגיות

מחזירה את כל השאלות שאחת מהתגיות שסופקו מקושרת אליה

public IEnumerable<Question> TagsSearch(IEnumerable<string> tags)

#### שליפת כל השאלות של משתמש מסויים

public IEnumerable<Question> GetQuestionsByUser(int userId)

#### שליפת השאלות האחרונות

public IEnumerable<Question> GetLatestQuestions (int userId)

#### שליפת שאלה וכל התשובות שלה

public DiscussionThread GetDiscussionThreadById(int id)

#### שליפת שאלה על פי מזהה

public Question GetQuestionById(int id)

#### שליפת תשובה על פי מזהה

public Question GetAnswerById(int id)

#### שליפת תשובות לשאלה

מתודה פרטית המשמשת את המתודה הציבורית המאחזרת את כל ה-thread

private List<Answer> GetAnswers(Question question)

מתודות המשמשות לשמירת היסטוריית השימושים במערכת, כך שמשתמש לא יוכל להעלות דירוג של שאלה יותר מפעם אחת. המתודות נקראות ע"י ה-BL, ולכן ציבוריות.

public void RankQuestion(int questionId, int userId, int rank)

public void RankAnswer(int answerId, int userId, int rank)

מתודות המשמשות לשליפת היסטוריית הדירוג של המשתמש לשאלות / תשובות

public int GetQuestionRankingHistory(int questionId, int userId)

public int GetAnswerRankingHistory(int answerId, int userId)

### שירותי התרעות Notifications

מחלקה זו משמשת לטיפול בנושא ההתרעות. היא מכילה מתודות לשמירה של התרעות לבסיס הנתונים ולשליפת התרעות מבסיס הנתונים.

המחלקה עושה שימוש ב-NotificationTyps enum המכיל את סוגי ההתרעות השונים, אשר מאוחסנים בבסיס הנתונים ע"פ המזהה המספרי שלהם. המתודות הסטאטיות במחלקה Converter משמשות להמרה בין אובייקט ההתרעה החיצוני (מ-common) והפנימי (מה-EntityFramework)

#### שמור התרעה חדשה

מתודה הנקראת ע"י ה-BL עבור כל פעולה שנרצה להתריע בפני המשתמש כי ארעה.

public void SaveNotification(int userId, string message, NotificationType notificationType)

#### מתודה לאחזור התרעות

מתודה לשליפת כל ההתרעות עבור משתמש מאז תאריך/זמן מסויים

public IEnumerable<Notification> GetNotifications(int userId, DateTime since)

## טבלאות מאגר הנתונים

### טבלת ההתראות notifications

משמשת לאחסון הודעות המערכת המיועדות למשתמשים.

|  |  |  |
| --- | --- | --- |
| שם שדה | סוג | הערות |
| Id | int | מזהה הודעה, identity, מפתח ראשי |
| notification\_type | int | סוג ההודעה |
| Message | nvarchar | תוכן ההודעה |
| Timestamp | datetime | מועד יצירת ההודעה |
| recipient | int | הנמען |
| Seen | tinyint | האם הנמען ראה את ההודעה |

כל הודעה מיועדת למשתמש אחד. המנהלים במערכת (מרצה/רכז) מקבלים את כלל ההתרעות הרלוונטיות לשאלות ותשובות.

### טבלת דירוגי התשובות answer\_rankings

טבלה שומרת את היסטוריית הדירוגים של משתמש לתשובה. הדירוג יכול להיות חיובי (+1) או שלילי (-1)

|  |  |  |
| --- | --- | --- |
| שם שדה | סוג | הערות |
| author\_id | Int | המשתמש שדירג, מפתח ראשי |
| answer\_id | int | התשובה שדורגה, מפתח ראשי |
| Rank | tinyint | הדירוג |

הטבלה מקשרת בין משתמשים ובין התשובות שהם דרגו, כדי למנוע דירוג יתר

### טבלת התשובות answers

בטבלה זו נשמרות התשובות לשאלות.

|  |  |  |
| --- | --- | --- |
| שם שדה | סוג | הערות |
| id | int | מזהה השאלה, מפתח ראשי |
| question\_id | int | מזהה השאלה עבורה התשובה מיועדת |
| contents | nvarchar | תוכן התשובה |
| ranking | int | דירוג התשובה |
| author\_id | int | מזהה המשתמש שפרסם את התשובה |
| seen | tinyint | האם התשובה נראתה |
| created | datetime | מועד יצירת התשובה |

הטבלה מקושרת לטבלת השאלות – כל תשובה קשורה לשאלה אחת בדיוק

### טבלת דירוגי השאלות question\_rankings

טבלה שומרת את היסטוריית הדירוגים של משתמש לשאלה. הדירוג יכול להיות חיובי (+1) או שלילי (-1)

|  |  |  |
| --- | --- | --- |
| שם שדה | סוג | הערות |
| author\_id | int | מזהה המשתמש המדרג, מפתח ראשי |
| question\_id | int | מזהה השאלה המדורגת, מפתח ראשי |
| rank | tinyint | הדירוג |

הטבלה מקשרת בין המשתמשים לשאלות שהם דרגו כדי למנוע דרוג יתר

### טבלת השאלות questions

בטבלה זו נשמרות השאלות

|  |  |  |
| --- | --- | --- |
| שם שדה | סוג | הערות |
| id | int | מזהה השאלה, identity, מפתח ראשי |
| title | nvarchar | כותרת השאלה |
| contents | nvarchar | תוכן השאלה |
| ranking | int | דירוג השאלה |
| author\_id | int | מזהה המשתמש השואל |
| recommended | tinyint | האם השאלה מומלצת |
| created | datetime | מועד יצירת השאלה |
| right\_answer\_id | int | מזהה התשובה הנכונה |
| version | int | גרסת השאלה |

### טבלת התגיות tags

בטבלה זו נשמרות התגיות שצורפו לכל שאלה

|  |  |  |
| --- | --- | --- |
| שם שדה | סוג | הערות |
| question\_id | int | מזהה השאלה, מפתח ראשי |
| tag | nvarchar | התגית, מפתח ראשי |

כל תגית קשורה לשאלה אחת. תגיות זהות יכולות להיות קשורות לשאלות רבות, אך כיוון שתגית היא טקסט חופשי, אין צורך בטבלת תגיות ראשית.

### טבלת המשתמשים users

טבלת המשתמשים במערכת. מכילה את השדות הבאים

|  |  |  |
| --- | --- | --- |
| שם שדה | סוג | הערות |
| Id | Int | מזהה המשתמש, identity, מפתח ראשי |
| name | Nvarchar | שם המשתמש |
| email | varchar | כתובת המייל של המשתמש |
| isadmin | tinyint | האם מרצה בקורס |
| isowner | tinyint | האם רכז הקורס |
| ismuted | tinyint | האם המשתמש מושתק |
| ranking | int | דירוג המשתמש |
| Created | datetime | מועד יצירת המשתמש |
| Pwd | Varchar | סיסמת המשתמש |

# שכבת הלוגיקה

שכבה המקשרת בין שכבת ההצגה למשתמש לשכבת הגישה לנתונים. שכבת ההצגה פונה לשכבה זו בבקשה לביצוע פעולות במערכת. שכבה זו פונה לשכבת הגישה לבסיס הנתונים על מנת לשמור או לשלוף נתונים מהמאגר.

בשכבת הלוגיקה יתבצעו פעולות שונות שיבטיחו שמצד אחד המידע שנשמר בבסיס הנתונים הוא תקין ובפורמט המתאים, ומצד שני יבטיח שהמידע שמגיע מבסיס הנתונים לשכבת ההצגה למשתמש יגיע לאחר סינונים שונים ובהתאם למה שרלוונטי לשכבת ההצגה.

למעשה שכבה זו היא השכבה האוכפת את התנהגות המערכת. כך ניתן להוסיף ממשק תצוגה נוסף, כאשר הקוד שבו יטפל רק בתצוגה, אך הלוגיקה העסקית של המערכת תישמר במקום אחד.

שכבת הלוגיקה שלנו חושפת את השירותים שלה לשכבת התצוגה למשתמש על ידי טכנולוגיית WCF.

## עקרונות מנחים בשכבת הלוגיקה:

* אימות נתונים - שכבת הלוגיקה תספק אימות (VALIDATIONS) עבור הנתונים שמתקבלים משכבת ההצגה למשתמש בביצוע פעולות.
* אימות משתמש – שכבת הלוגיקה תזהה את המשתמש בכל כניסה למערכת, ותנפיק למשתמש Token אותו הוא יצטרך להציג בביצוע כל פעולה כדי להזדהות.
* ניהול הרשאות – שכבת הלוגיקה תמנע ממשתמש לבצע פעולה שהוא לא מורשה אליה, למשל – משתמש שאיננו מרצה או מרכז, שמנסה להשתיק משתמש אחר. המטרה היא למנוע אפשרות של עקיפת שכבת ההצגה למשתמש על ידי כלים חיצוניים.
* מימוש לוגיקות שונות בהתאם לאפיון המערכת. למשל – דירוג של שאלה\תשובה משנה גם את דירוג הכותב שלה.
* עקרון הכימוס בא לידי ביטוי בשכבת הלוגיקה בכמה אופנים. קודם כל, כמעט כל שתי מחלקות ייגשו אחד לשני רק באמצעות interfaceים, כך שניתן בקלות להחליף את השימוש של מחלקה מסוימת למחלקה אחרת ללא שינויי קוד רבים וללא תלות במימוש הפנימי של המחלקה.
* Dependency Injection – כל מחלקה תקבל את הממשקים עליהן יש תלות בConstructor. כך בקלות ניתן להחליף מימושים של הממשקים בלי שינוי של קוד באותה מחלקה, וכמעט ללא שינוי קוד. כמו כן, יהיה קל להוסיף Unit Tests בשלב יותר מאוחר של הפרויקט – כמובן שלא נוסיף אותם במסגרת פרויקט זה.

## מחלקות כלליות

### AuthTokenCache

מחלקה זו היא Singleton ומשמשת כCache של Authentication tokens כדי לזרז את אימות פעולת משתמש.

המחלקה תאפשר לנו לא לפנות לDAL בכל פעם שמשתמש מנסה לבצע פעולה ומזדהה עם הauthentication token שלו.

אם הAuthentication token שנבדק לא נמצא בCache, נוכל להפיק מהAuthentication Token את הCredentials של המשתמש ולאמת את זהותו בפניה לDAL. אם זהותו אומתה, נוסיף את הToken לCache.

הCache משתמש בConcurrentBag על מנת שיהיה קל לשלוף ולהכניס אליו Tokens באופן אסינכרוני באופן שהוא Thread-Safe.

static AuthTokenCache Instance

ConcurrentBag<string> TokenCache

### IAuthTokenSerializer

ממשק שמאפשר להמיר Authentication Token לזהות של יוזר ולהפך.

bool ValidateAndSerialize(UserCredentials userCredentials, out AuthenticatedIdentity serializedIdentity)

מחזירה "אמת" אם אימתנו בהצלחה את הCredentials של המתמש ושמרנו בCache את הtoken הנוצר.

int Deserialize(AuthenticationToken authToken)

מקבל AuthenticationToken ומחזיר את הUserId שאליו שייך הtoken. אם הToken לא תקין ולא אמיתי, תעוף שגיאה.

### AuthTokenSerializer

מחלקה זו מייצרת Authentication Token מUserCredentials ולהפך. מחלקה זו יורשת מIAuthTokenSerializer. כאשר מתקבלים UserCredentials של משתמש, אנחנו רוצים להחזיר למשתמש AuthenticatedIdentity שמתאים ספציפית לUserCredentials שהוא הזין (ואומתו בהצלחה) כך שבכל פעולה שהוא יבצע הוא יעביר את הAuthenticationToken וכך הBL יוכל לזהות את המשתמש בהצלחה ללא צורך בבדיקת הCredentials מול הDB. מה שיכול להאיץ את תהליך אימות המשתמש בכל פעולה. כמו כן ניתן להרחיב את הAuthenticationToken כן שהוא יהיה תקף רק לפרק זמן מוגבל, וכך במידה והAuthenticationToken נגנב על ידי פורץ, הToken יוכל להיות בשימוש רק לפרק זמן מוגבל. במימוש אמיתי של המערכת שמעבר לגבולות הפרויקט, היה ניתן גם לייצר את הAuthenticationToken מהIP Address של המשתמש וגם מהUserCredentials שלו, ובכל שלב שמתקבל Token, לאמת אותו מול הClient IP Address, כך שגם אם הToken נגנב, לא ניתן להשתמש בו ממחשב אחר.

bool ValidateAndSerialize(UserCredentials userCredentials, out AuthenticatedIdentity serializedIdentity)

int Deserialize(AuthenticationToken authToken)

### INotificationAggregator

ממשק שמשמש לאיחוד התראות דומות לכדי רשימה מצומצמת של התראות. כדי למנוע תצוגה של כמות גדולה מדי של התרעות. ההתרעות מקובצות לפי נושאים.

IEnumerable<Notification> AggregateNotifications(IEnumerable<Notification> notifications)

### NotificationAggregator

מחלקה זו משתמשת לאיחוד התראות דומות לכדי רשימה מצומצמת של התראות. המחלקה יורשת מINotificationAggregator.

הפעולה הראשית של המחלקה מקבלת רשימה של התראות שונות – כך שיש התראה אחת לכל סוג. לדוגמא, אם משתמש שאל שאלה, וכשחזר למערכת התווספו לשאלה שלו ארבע תשובות, במקום לקבל ארבע התרעות על תשובה חדשה, הוא יקבל התראה אחת על ארבע תשובות חדשות.

public IEnumerable<Notification>

AggregateNotifications(IEnumerable<Notification> notifications)

### NotificationService

יורשת מINotificationService. מחלקה זו מספקת שירות לשכבת התצוגה שנותן לה את כל ההתראות על פעולות שהתרחשו שהמשתמש עדיין לא מודע אליהם.

הפעולה הראשית מקבל Authentication Token שבאמצעותו ניתן לזהות את המשתמש שמבקש לקבל את ההתראות החדשות שלו. המחלקה תבדוק מול שכבת הDAL איזה התראות עדיין לא "נמשכו", ואז תשתמש בNotificationsAggregator כדי לאחד התראות דומות, ותחזיר את התוצאה למשתמש.

public IEnumerable<Notification> GetNotifications(AuthenticationToken authToken)

### PostQueries

יורשת מIPostQueries. מחלקה זו מספקת שירותי שאילתות על שאלות ותשובות שהוזנו במערכת.

המחלקה יורשת מממשק מתאים שחושף אותה כשירות WCF לשכבת ההצגה למשתמש.

כל הפעולות במחלקה זו מקבלות AuthenticationToken או AuthenticatedOperation כפרמטר על מנת שיהיה אפשר לזהות את המשתמש שמקבל לבצע את הפעולה.

הפעולות שניתן לבצע על ידי מחלקה זו הם חיפוש חופשי לפי טקסט, חיפוש לפי תגיות, חיפוש לפי שאלות שהמשתמש שקורא לפעולה הזין למערכת, חיפוש שאלות אחרונות, וקבלת התוכן של שאלה וכל התשובות שהוזנו עבורה לפי id של השאלה.

public IEnumerable<Question> FreeSearch(AuthenticatedOperation<string> searchString)

public IEnumerable<Question> TagsSearch(AuthenticatedOperation<IEnumerable<string>> tags)

public IEnumerable<Question> GetMyQuestions(AuthenticationToken authToken)

public DiscussionThread GetDiscussionThreadById(AuthenticatedOperation<int> id)

public IEnumerable<Question> GetLatestQuestions()

### PostServices

יורשת מIPostServices. מחלקה זו מספקת שירות לשכבת תצוגה של פעולות הזנה ומחיקה על שאלות ותשובות. כל הפעולות במחלקה זו מקבלות AuthenticationToken או AuthenticatedOperation על מנת לזהות את המשתמש שביצע את הפעולה ולאמת אותו ואת ההרשאות שלו.

המחלקה יורשת מממשק מתאים שחושף אותה כשירות WCF לשכבת ההצגה למשתמש.

עבור כל פעולה שדורשת הרשאות לא בסיסיות, המחלקה תבדוק האם למשתמש יש את ההרשאות הנחוצות. עבור פעולות הזנה ומחיקה, המחלקה תוודא שהמשתמש איננו מושתק.

בין הפעולות שניתן לבצע במחלקה זו הם הזנת שאלה חדשה, מחיקת שאלה קיימת לפי id, הזנת תשובה לשאלה, מחיקת תשובה לפי id, העלאת דרגה של שאלה\תשובה לפי id, הורדת דרגה של שאלה\תשובה לפי id, וסימון שאלה כממולצת לפי id.

לכל פעולה יתבצעו Validations מתאימים על המידע המוזן, ועל ההרשאות של המשתמש עצמו בהתאם לפעולה המבוקשת.

העלאת דירוג של שאלה או תשובה יגרום להעלאת דירוג המשתמש שכתב אותה. כמו גם הורדת דירוג של שאלה או תשובה יוריד את דירוג המשתמש שכתב אותה.

public void AddQuestion(AuthenticatedOperation<Question> question)

public void DeleteQuestion(AuthenticatedOperation<int> questionId)

public void AddAnswer(AuthenticatedOperation<Answer> answer)

public void DeleteAnswer(AuthenticatedOperation<int> answerId)

public void VoteUpQuestion(AuthenticatedOperation<int> questionId)

public void VoteDownQuestion(AuthenticatedOperation<int> questionId)

public void VoteUpAnswer(AuthenticatedOperation<int> answerId)

public void VoteDownAnswer(AuthenticatedOperation<int> answerId)

public void RecommendQuestion(AuthenticatedOperation<int> questionId)

public void UnrecommendQuestion(AuthenticatedOperation<int> questionId)

public void CancelQuestionRanking(AuthenticatedOperation<int> questionId)

public void CancelAnswerVote(AuthenticatedOperation<int> answerId)

public void MarkAsRightAnswer(AuthenticatedOperation<int> answerId)

public void UnmarkAsRightAnswer(AuthenticatedOperation<int> answerId)

### UserServices

יורשת מIUserServices. מחלקה זו משתמש שירות לשכבת התצוגה לביצוע פעולות על משתמשים.

המחלקה יורשת מממשק מתאים שחושף אותה כשירות WCF לשכבת ההצגה למשתמש.

על פעולה מקבלת Authentication Token או AuthenticatedOperation על מנת שיהיה אפשר לזהות את המשתמש שמבצע את הפעולה ולוודא שיש לו את ההרשאות הנדרשות לביצוען. באופן זה לכל פעולה יבוצעו Validations שיוודאו את תקינות המידע שהתקבל, וההרשאות של המשתמש.

בין הפעולות שניתן לבצע במחלקה הם הוספה של משתמש חדש למערכת (זה מתבצע בתהליך הרישום למערכת), כניסה למערכת וקבלת הAuthenticationToken המתאים, קבלת המידע על כל המשתמשים במערכת (רק מרצים ורכזים יכולים לבצע זאת), שמירת מידע חדש על המשתמשים (זה יתבצע בשמירה במסך הניהול של הרכז והמרצים לאחר עדכונו), וקבלת מידע על משתמש ספציפי.

public OperationResult AddNewUser(UserInfo newUser, UserCredentials userCredentials)

public OperationResult<AuthenticationToken> LogIn(UserCredentials userCredentials)

public IEnumerable<UserInfo> GetAllUsersData(AuthenticationToken authToken)

public OperationResult SaveUsersData(AuthenticatedOperation<IEnumerable<UserInfo>> usersData)

public OperationResult<UserInfo> GetUserInfo(AuthenticationToken authToken)

# שכבת התצוגה

שכבה זו מכילה את הממשק למשתמש שמכיל את המסכים שבהם המשתמש יכול לצפות, ולבצע אינטראקציה. השכבה גם מכילה validations בצד לקוח, פניות לשכבת הBL לקבלת שירותים, שמירה על מצב המשתמש המחובר כעת, קבלת התראות משכבת הBL והצגתן, ועוד...

## עקרונות מנחים בשכבת ההצגה:

* שכבת ההצגה ממומשת באמצעות MVVM (Model-View-ViewModel) כך שיש הפרדה מלאה בין המסכים (views), הלוגיקה של המסכים (view models) והמחלקות מידע (models). כך במידת הצורך בעתיד, ניתן בקלות לשנות את המסכים מבלי לשנות את הלוגיקה של המסכים, ולהפך.
* קישור הView לViewModels יעשה דרך DataTemplates, והצגת המסכים יעשו באמצעות ContentControl שיכיל את הViewModel. באופן זה, בקוד אין צורך להתעסק כלל בViews, והView המתאים יוצג באופן אוטומטי כאשר נבחר בקוד להציג את הViewModel שלו.
* בכל פעולה על המסך יהיה שימוש בICommand. במידת הצורך אם יותר ממסך אחד צריכים את אותו ICommand יהיה ניתן לשתף בין המסכים את אותו Command.
* שימוש בDesign Patterns במקומות רלוונטיים, שישמרו על עיצוב השכבה גמיש לשינויים.

## מודלים

### AbstractPost

מחלקה אבסטרקטית המתארת פוסט כלשהו במערכת. כמו כן המחלקה מממשת את INotifyPropertyChanged כך שכל שינוי בתכונות המחלקה יגרמו לשינוי מתאים במסך המוצג למשתמש.

public DateTime DatePosted

public int VotesCount

public string Author

public string Content

public event PropertyChangedEventHandler PropertyChanged;

### DelegateCommand

מממשת את ICommand. זהו מימוש של ICommand שמקבל שני delegates אחד עבור CanExecute ואחד עבור Execute (שהם חלק מהInterface של ICommand). כאשר wpf משתמש בCanExecute כדי להחליט אם ניתן להפעיל את הCommand או לא.

public event EventHandler CanExecuteChanged;

public bool CanExecute(object parameter)

public void Execute(object parameter)

public void RaiseCanExecuteChanged()

### AnswerModel

יורשת מAbstractPost. מחלקה המייצגת תשובה לשאלה במערכת, היודעת לעדכן את המסך במקרה ויש שינוי של אחד הערכים בה. מחלקה זו יורשת מAbstractPost. ניתן לצפות במודלים מסוג Answer במסכים שמראים Discussion Thread שלם, כלומר, שאלה, והתשובות שכתבו עבורה או בהזנה של תשובה חדשה לשאלה. כמו כן, ניתן להמיר כל מופע של מחלקה זו במופע של המחלקה ששאר המערכת מכירה (מחלקה בCommon).

public bool Answered

public Answer ToExternal()

public int RelatedQuestionId

### QuestionInfo

מחלקה המייצגת מידע תצוגתי של מידע על שאלה במערכת. יש לשים לב שלמחלקה זו אין את כל המידע על השאלה (למשל אין את תוכן השאלה). זאת מאחר שנעשה שימוש במודל זה רק עבור המסך תוצאות חיפוש – שבו מוצג רק חלק מהמידע על השאלה בכל שורה מבין התוצאות חיפוש. כמו כן המחלקה מממשת את INotifyPropertyChanged כדי להודיע למסך על עדכונים.

public ObservableCollection<string> Tags

public bool IsRecommended

public string Author

public string Title

public int Ranking

public int AnswersCount

public Question ToExternal()

public event PropertyChangedEventHandler PropertyChanged;

### QuestionModel

יורשת מAbstractPost. מחלקה המייצגת מידע תצוגתי של שאלה במערכת. במחלקה זו משתמשים כדי להזין שאלה חדשה למערכת, וכדי לצפות בDiscussion Thread שלם – בו יש שאלה ותשובות שהוזנו עבורה. כמו כן, מחלקה זו מממשת את INotifyPropertyChanged כדי ששינויים בכל מופע של המחלקה ייראו על המסך. כמו כן ניתן להמיר אותה למחלקה שכל השכבות מכירות DiscussionThread.

public int AnswerCount

public ObservableCollection<AnswerModel> Answers

public ObservableCollection<string> Tags

public bool Recommended

public string Title

public DiscussionThread ToExternal()

### RankingHistoryModel

מחלקה משמשת למידע תצוגתי על היסטוריית דירוג פוסטים.

public RankingHistory ToExternal()

public int OverallRanking

### UserAccountModel

מחלקה זו משמשת למידע תצוגתי של משתמש במערכת. השימוש במחלקה זו הוא ביצירת משתמש חדש, ובמסך ניהול משתמשים אליו רכז ההוראה והמרצים יכולים לגשת. גם מחלקה זו מממשת את INotifyPropertyChanged וניתן להמיר אותה למחלקה שכל השכבות מכירות UserInfo.

public string Username

public int ID

public int Ranking

public bool IsAdmin

public bool IsMuted

public UserInfo ToExternal()

public event PropertyChangedEventHandler PropertyChanged;

### NotificationsObserver

מחלקה זו פונה באופן פריודי לBL, ומתשאלת על התראות חדשות שמותאמות למשתמש שכרגע מחובר. במידה ויש התראות חדשות, יוקפץ event מתאים שיודיע על כך, עם ההתראות החדשות.

public event EventHandler<NewNotificationsEventArgs> NewNotifications;

### NewNotificationsEventArgs

מחלקת המידע שבה משתמשים באירוע של קבלת התראות חדשות.  
הNotificationsObserver יוצר מופע של מחלקה זו בכל פעם שזיהה שיש התראות חדשות.

public IEnumerable<Notification> NewNotifications { get; private set; }

public void Dispose()

### AuthenticationTokenSingleton

מחלקה זו שומרת על הAuthentication Token שחזר לאחר תהליך הLog-in של המשתמש (אם בוצע), כך שכל מחלקה שרוצה לפנות לשכבת הBL וצריכה את הAuthentication Token על מנת להזדהות, תוכל לקחת אותו ממקום אחד בקלות.

המחלקה הזו מממשת את הDesign Pattern של Singleton מאחר ויכול בכל רגע נתון להיות רק משתמש אחד מחובר בכל עמדת קצה. לכן לא יכולה להיות במערכת שני Authentication Tokens שונים בו זמנית.

באמצעות מחלקה זו ניתן ליצור AuthenticatedOperation שמכיל כאמור גם את הAuthenticationToken הנוכחי של המשתמש המחובר.

בכל log in וlog out של משתמש מהמערכת, המחלקה תקפיץ אירוע מתאים.

public static AuthenticationTokenSingleton Instance

public AuthenticatedIdentity AuthenticatedIdentity { get; }

public UserInfo AuthenticatedUser { get; }

public void LogIn(AuthenticatedIdentity authenticatedIdentity, UserInfo authenticatedUser)

public void LogOut()

public bool IsLoggedIn { get; }

public AuthenticatedOperation<T> CreateAuthenticatedOperation<T>(T input)

public event EventHandler<EventArgs> UserLoggedOut;

public event EventHandler<EventArgs> UserLoggedIn;

### DiscussionThreadObserver

מחלקה זו מקשיבה לשינויים שקורים בDiscussion-Thread מסויים (שינוי בשאלה הנישאלת\דירוג שלה\סימון שלה כמומלצת או כל שינוי אחר באחד מהתשובות שהוזנו לשאלה) ומודיעה על כך שיש גרסה חדשה של הDiscussion Thread, וכך בעצם המסך יכול להתעדכן אוטומטית כשקורה איזהשהו שינוי בDiscussion Thread שעכשיו צופה בו המשתמש.

public event EventHandler<NewDiscussionThreadVersionEventArgs> NewDiscussionThreadVersion;

public DiscussionThread StartObservingDiscussionThread(int questionId)

public void Dispose()

### NewDiscussionThreadVersionEventArgs

מחלקת המידע שבה משתמשים באירוע שבו יש גרסה חדשה של הDiscussion Thread שנצפה כרגע.

הDiscussionThreadObserver יוצר מופע חדש של מחלקה זו בכל פעם שיש שינוי בDiscussionThread.

public DiscussionThread NewDiscussionThread { get; set; }

### NavigationManager

מחלקה זו מודיעה על בקשה של מעבר למסך אחר.

כל הViewModels שמהם ניתן לנווט למסך אחר, יכירו את מחלקה זו, ובכל פעם שהViewModel רוצה להחליף מסך, הוא פונה לNavigationManager עם הViewModel שהוא רוצה לעבור אליו. לאחר מכן הNavigationManager מעלה event עם בקשת השינוי, המסך הראשי מקשיב לאירוע זה, ומחליף את המסך הראשי בהתאם.

public void RequestNavigation(object viewModel);

event EventHandler<NavigationRequestedEventArgs> NavigationRequested;

### NavigationRequestedEventArgs

מחלקת המידע שמכילה את הViewModel אליו רוצים לבצע navigation.

בכל פעם שהNavigationManager רוצה להודיע על navigation הוא יוצר מופע של מחלקה זו.

public object ViewModelToNavigate { get; set; }

### App.xaml

הxaml הראשי של שכבת ההצגה למשתמש.

קובץ זה מציין מי החלון הראשון שנפתח בעליית התכנית, וכמו כן מגדיר DataTemplates שמקשרים בין כל הViews לViewModels שלהם, באופן כזה שבכל פעם שנרצה להציג ViewModel מסויים על המסך, באופן אוטומטי אנחנו נראה את הView שלו, והDataContext של הView יהיה הViewModel המתאים לו שרצינו "להציג".

### MainWindow

החלון הראשי של המערכת.

החלון מחלק את המסך למספר חלקים שבכל אחד מהם יוצג View אחר.

הMainMenu וMainToolbar הם כפי שניתן לראות במסמך אפיון ומדריך למשתמש. הNotificationsViewModel זה הViewModel שיציג התראות למשתמש על פעולות שהתבצעו שיעניינו את אותו משתמש, והMainVM זהו ViewModel שמתחלף כל הזמן בהתאם לנוויגציה של המשתמש במערכת – כלומר זה החלק היחיד שמשתנה במסך הראשי, כאשר המשתמש רוצה להירשם למערכת החלק הזה יכיל את הViewModel של הרישום, כאשר המשתמש צופה בשאלה, חלק זה יכיל את הDiscussion Thread שהוא צופה בו וכו...

public object MainVM

public NotificationsViewModel NotificationsVM

public MainMenuViewModel MainMenuVM

public MainToolbarViewModel MainToolbarVM

### MainMenu.xaml

View זה יכיל את רכיבהכפתורי הניווט למסכים הראשיים. ניתן לראות את View זה במדריך למשתמש.

### MainMenuViewModel

זהו הViewModel שמתאים לMainMenu. הViewModel יכיל את כל הCommands שיתבצעו בלחיצה על הכפתור הרלוונטי בMainMenu.

public bool IsLoggedIn { get; }

public bool IsMuted { get; }

public bool IsUserAdmin { get; set; }

public ICommand NavigateHome { get; set; }

public ICommand NavigateMyQuestions { get; set; }

public ICommand NavigateNewQuestion { get; set; }

public ICommand NavigateUserManagement { get; set; }

public event PropertyChangedEventHandler PropertyChanged;

public void Dispose()

### MainToolbar.xaml

זהו הView של הToolbar העליון שמכיל את כפתור הLogin, Registration ואת החיפוש המהיר.

### MainToolbarViewModel

זהו הViewModel של הMainToolbar. הוא מכיל את כל הCommands של הפעולות האפשריות מהToolbar, ואת הטקסט של החיפוש החופשי.

public bool IsLoggedIn { get; }

public string LoggedInUser { get; }

public string SearchText { get; set; }

public ICommand DoSearch { get; set; }

public ICommand DoLogin { get; set; }

public ICommand DoRegister { get; set; }

public ICommand DoLogout { get; set; }

public event PropertyChangedEventHandler PropertyChanged;

public void Dispose()

### NewAnswer.xaml

זהו המסך המוצג בעת כתיבה של תשובה חדשה לשאלה.

ניתן לראות את מסך זה במדריך למשתמש.

### NewAnswerViewModel

זהו הViewModel של מסך כתיבת התשובה החדשה.

יש בו את הCommand של פרסום התשובה החדשה, ויש בו את הModel שמחזיק את הפרטים של התשובה החדשה שהמשתמש מזין.

public AnswerModel Answer

public string ErrorMessages { get; set; }

public ICommand SubmitAnswer { get; set; }

public event PropertyChangedEventHandler PropertyChanged;

### NewQuestion.xaml

זהו המסך המוצג בעת כתיבה של שאלה חדשה לשאלה.

ניתן לראות את מסך זה במדריך למשתמש.

### NewQuestionViewModel

זהו הViewModel של מסך כתיבת השאלה החדשה.

יש בו את הCommand של פרסום השאלה החדשה, ויש בו את הModel שמחזיק את הפרטים של השאלה החדשה שהמשתמש מזין.

public string TagsText { get; set; }

public string ErrorMessages { get; set; }

public ICommand AddNewTag { get; set; }

public ICommand RemoveTag { get; set; }

public ICommand PublishQuestion { get; set; }

public QuestionModel Question { get; set; }

public void ValidateAndSubmitQuestion()

public event PropertyChangedEventHandler PropertyChanged;

### QuestionList.xaml

זהו הView שמכיל רשימה של שאלות (ללא תשובותיהם).

View זה מוצג כאשר רוצים להציג תוצאות של חיפוש, או כאשר מסתכלים על "השאלות שלי" (השאלות שהמשתמש הזין) על ידי לחיצה בMainMenu.

במסך יש ItemsControl שמגדיר ItemTemplate לכל אחד מהשאלות שהוא מציג, כך שבאופן אוטומטי, כל שאלה שמצויינת בViewModel תוצג על ידי ItemTemplate זה.

### QuestionListViewModel

זהו הViewModel של QuestionList. ViewModel זה מחזיק את רשימת השאלה שברצוננו להציג למשתמש (כתוצאות של חיפוש או כאשר המשתמש צופה ב"שאלות שלי"). כמו כן הViewModel מכיל Command של צפייה בDiscussionThread ספציפי על ידי לחיצה על אחד השאלות במסך QuestionList – השאלה שנלחצה תעבור כפרמטר לCommand של NavigateToQuestion.

public ICommand NavigateToQuestion { get; set; }

public ICommand DoTagSearch { get; set; }

public ObservableCollection<QuestionInfo> Questions

public event PropertyChangedEventHandler PropertyChanged;

### PostViewModel<T>

מחלקה גנרית אבסטרקטית ממנה שבה יש properties וmethods שישמשו דברים משותפים בViewModels של שאלות ותשובות. ביניהם, דירוג לחיוב ושלילה, מחיקה וכו..

public T Post { get; }

public ICommand RankUp { get; }

public ICommand CancelRankUp { get; }

public ICommand RankDown { get; set; }

public ICommand CancelRankDown { get; }

public ICommand Delete { get; set; }

public bool IsAuthor { get; }

public bool IsRankUpEnabled { get; }

public bool IsRankDownEnabled { get; }

public bool IsRankedUp { get; }

public bool IsRankedDown { get; }

public bool CanDeletePost { get; }

public bool CanAddAnswer { get; }

public virtual void Dispose()

public event PropertyChangedEventHandler PropertyChanged;

### QuestionView.xaml

החלק התצוגתי של שאלה – בתוך מסך צפייה בשאלה תשובותיה (Discussion Thread). הView הזה יופיע לשאלה הראשית שמוצגת כרגע במסך.

### QuestionViewModel

מחלקה היורשת מPostViewModel

זהו הViewModel של השאלה (היחידה) שנצפית כרגע במסך.

לViewModel זה יש properties שיהיה ניתן לעשות אליהם Binding מהView.

זה כולל את הModel (QuestionModel) וCommands שניתן להפעיל בלחיצה על כפתור בView כמו דירוג השאלה לחיוב\שלילה.

כמו כן הViewModel מממש את INotifyPropertyChanged כך שבכל שינוי שלו, הView יתעדכן אוטומטית בהתאם.

public QuestionModel Question

public ObservableCollection<AnswerViewModel> Answers

public ICommand RankUp { get; set; }

public ICommand RankDown { get; set; }

public ICommand Delete { get; set; }

public ICommand AddAnswer { get; set; }

public bool IsRecommendationEnabled { get; set; }

public ICommand RecommendQuestion { get; set; }

public ICommand UnrecommendQuestion { get; set; }

public ICommand DoTagSearch { get; set; }

public override void Dispose()

public event PropertyChangedEventHandler PropertyChanged;

### AnswerView.xaml

החלק התצוגתי של תשובה – בתוך מסך צפייה בשאלה ותשובותיה (Discussion thread). לכל תשובה שמוצגת על המסך יהיה מופע של הView הזה.

### AnswerViewModel

מחלקה היורשת מPostViewModel

הViewModel של תשובה שנצפית על המסך.

לViewModel יש properties שיהיה ניתן לעשות אליהם Binding מהView.

זה כולל את הModel (AnswerModel) וCommands שניתן להפעיל בלחיצה על כפתור בView כמו דירוג התשובה לחיוב\שלילה, או מחיקת התשובה.

כמו כן הViewModel מממש את INotifyPropertyChanged כך שבכל שינוי שלו, הView יתעדכן אוטומטית בהתאם.

public AnswerModel Answer

public bool IsQuestionAuthorOrAdmin { get; }

public ICommand MarkAsRight { get; set; }

public ICommand UnmarkAsRight { get; set; }

public ICommand RankUp { get; set; }

public ICommand RankDown { get; set; }

public ICommand Delete { get; set; }

public event PropertyChangedEventHandler PropertyChanged;

### RegistrationControl.xaml

מסך זה הוא מסך הרישום למערכת.

במסך זה מזינים את כל האינפורמציה הדרושה כדי להירשם למערכת.

ניתן לראות את מסך זה במדריך למשתמש.

### RegistrationViewModel

הViewModel של מסך הרישום למערכת.

מסך זה מכיל את הCommand של רישום המשתמש לאחר הזנת הפרטים – שיוודא שכל הפרטים שהוזנו הם תקינים על ידי שימוש בValidations, ולאחר מכן ישמור את המשתמש החדש במערכת.

הUser והUserCredentials יתמלאו על ידי המשתמש כאשר ימלא את השדות הרלוונטים במסך.

public UserCredentials Credentials { get; set; }

public ICommand RegisterUser { get; set; }

public string UserName { get; set; }

public string Password { get; set; }

public string Confirmation { get; set; }

public string Email { get; set; }

public string RegistrationResult { get; }

public event PropertyChangedEventHandler PropertyChanged;

### UserManagementView.xaml

זהו מסך ניהול המשתמשים שאליו רק מרצים ורכז ההוראה יכולים לגשת.

במסך זה ניתן להשתיק משתמשים, להפוך משתמש למרצה, ועוד...

ניתן לראות את מסך זה במדריך למשתמש.

### UserManagementViewModel

זהו הViewModel של מסך הניהול למשתמש. הViewModel מכיל בObservableCollection את כל המשתמשים הקיימים במערכת, ואת הCommand של שמירת השינויים שהתבצעו במסך.

public ObservableCollection<UserAccountModel> Users

public ICommand ApplyChanges { get; set; }

public event PropertyChangedEventHandler PropertyChanged;

### NotificationsView.xaml

הView של הצגת התראות למשתמש. הView יכיל את כל ההתראות האחרונות שרלוונטיות למשתמש המחובר.

### NotificationsViewModel

הViewModel של המצגת התראות למשתמש.

ViewModel זה יכיל בObservableCollection את כל ההתראות האחרונות שיוצגו על המסך. כמו כן הViewModel ייצור מופע של הNotificationsObserver כך שיוכל לדעת על התראות חדשות משכבת הBL ולהציג אותם למשתמש על ידי עדכון הproperty: Notifications

public ObservableCollection<Notification> Notifications;

### Styles.xaml

מכיל Styleים שונים לControlים בכל שכבת הPL.

# דפוסי עיצוב - Design Patterns

דפוסי עיצוב בהם ייעשה שימוש בפרויקט:

## MVVM – Model-View-ViewModel

תבנית זו תפקידה לייצר הפרדה בין המסכים (הViews) לבין אובייקטי המידע (Models) והלוגיקה של המסכים (הViewModels). כך ניתן להחליף כל אחד מרכיבים אלה עם מעט מאוד שינויים בשכבות האחרות (בדרך כלל אם אין שבירת ממשקים בין השכבות אז גם אין שינויים בכלל).

בתבנית זו נעשה שימוש בPresentation Layer כך שלכל מסך יש את הView, Model והViewModel שלו. ניתן לראות את ה"חיבור" בין הViews לViewModels בקובץ App.xaml על ידי DataTemplates, מה שגורם לכך שבכל פעם שנרצה להציג על המסך ViewModel מסוים, יוצג באופן אוטומטי הView הרלוונטי שלו.

## Singleton

תבנית שתפקידה לוודא שקיים למחלקה מופע אחד בדיוק, בנוסף התבנית מספקת מצביע גלובלי שיאפשר לגשת למופע זה. שימוש בתבנית זו ימנע מהיווצרותן של שגיאות לוגיות הנגרמות כתוצאה מיצירה של כמה מופעים עבור מחלקות שנדרש עבורן מופע אחד לכל היותר.

בתבנית זו נעשה שימוש במחלקה AuthenticationTokenSingleton.

מחלקה זו מייצגת את הAuthenticationToken של המשתמש הנוכחי שמחובר למערכת. מאחר שבכל רגע נתון בכל עמדת קצה רק משתמש אחד יכול להיות מחובר אל המערכת, נעשה שימוש בתבנית Singleton במחלקה זו. שימוש בתבנית יעזור לנו לוודא שלא קיים יותר ממשתמש אחד שמחובר למערכת – כלומר שיש רק AuthenticationToken אחד (שהשימוש שלו זה הזדהות מול שכבת הBL). הAuthenticationTokenSingleton נגיש לכל מחלקה בPresentation Layer, וניתן באמצעותו גם להשיג מידע על המשתמש המחובר מהBL (למשל לבדיקת הרשאות).

## Observer

תבנית המגדירה תלות אחד-לרבים בין האובייקטים, כאשר מצב של אובייקט אחד משתנה, כל שאר האובייקטים שתלויים בו מקבלים התראה על השינוי ומתעדכנים באופן אוטומטי.

בתבנית זו נעשה שימוש במחלקות הבאות:

### DiscussionThreadObserver

מחלקה זו מייצגת ומודיעה על הגרסה העדכנית ביותר של הDiscussion Thread שנצפה כעת (במידה והמסך הנוכחי הוא מסך צפייה בשאלה והתשובות שלה. מחלקה זו בעצם תודיע לכל מחלקה שמבקשת, בכל פעם שנעשה שינוי והתעדכן משהו בDiscussion Thread הנוכחי. היא תעשה זאת על ידי כך שכל כמה שניות היא פונה לשכבת הBL ובודקת האם שדה הVersion של אותו Discussion Thread בו אנחנו צופים כעת, גבוהה יותר מהVersion הנוכחי שנצפה כעת. כאשר הDiscussionThreadObserver מזהה שהגרסה גדלה, הוא מודיע בEvent לSubscriber שלו – שבמקרה זה זו המחלקה QuestionViewModel – שעליו לעדכן את הDiscussion Thread אותו הוא מציג לאחד חדש יותר, ודרך הEventArgs מעביר לו את הDiscussionThread החדש אותו עליו להציג.

NotificationsObserver

מחלקה זו מודיעה לכל דורש על כל ההתרעות החדשות שרלוונטיות למשתמש המחובר.

בכל כמה שניות הNotificationsObserver פונה לשכבת הBL ומבקש התרעות חדשות עבור המשתמש המחובר. בתשובה הוא יקבל רק התראות חדשות שלא קיבל כבר בעבר מהBL, ויודיע על ההתרעות החדשות על ידי Event. הSubscriber היחיד כרגע של הNotificationsObserver הוא הNotificationsViewModel שמציג את כל ההתראות שרלוונטיות למשתמש. ההתרעות החדשות יועברו על בEventArgs של הEvent.

## Command

תבנית שעושה אנקפסולציה לבקשה בתור אובייקט, כך שניתן להעביר את אובייקט זה בין מחלקות כאובייקט, וכל מחלקה יכולה להפעיל את הבקשה עם פרמטרים משלה.

תבנית זו ממומשת על ידי המחלקה DelegateCommand שבה משתמשים בכל שכבת הPL. השימוש בתבנית יעשה כך, שהViewModel יוכל להגדיר Commandים שונים לפעולות שונות, ולחשוף אותם לViews דרך Binding כך שהView יוכל להפעיל את הCommandים עם פרמטרים שלו (שקיבל למשל דרך User Input). כמו כן DelegateCommand מממש גם CanExecute, כך שהView יכול להציג למשתמש האם הוא יכול לבצע את הבקשה או שלא (למשל על ידי העברת כפתור למצב Disabled).

## Decorator

תבנית שמאפשרת לחבר מספר אחראויות לאובייקט דינמי אחד, כך שהפעלה של אובייקט זה יפעיל את כל האחראויות שמחוברות לאובייקט. בצורה זו, השימוש באובייקט יכול להיות באין יודעין על כל המחלקות ש"עוטפות" את האובייקט הנתון.

השימוש בתבנית זו נעשה במחלקה ValidatorDecoratorובכל המחלקות היורשות ממנה:

ContentValidator, MarkRightAnswerValidator, TitleValidator, EmailAddressValidator, NameValidator, PasswordValidator, RankDownValidator, RecommendQuestionValidator.

כל מחלקה כזאת יורשת מValidatorDecorator וכך ניתן לכל פעולה (למשל רישום של משתמש חדש) ליצור "שרשרת" של בדיקות שונות כך שכל Validator עוטף Validator אחר, ולבסוף מתקבל אובייקט Validator יחיד שמבצע את כל הבדיקות הדרושות על מנת לבדוק האם הרישום של המשתמש למערכת תקין. בדרך זו ניתן לעשות Reuse לוולידציה כך שניתן יהיה להשתמש בה במקומות שונים במערכת (למשל גם בPL וגם בBL) .שרשרת הValidations נבנית פעם אחת בכל מחלקה, ואז כל השיטות של המחלקה משתמשות בוולידטור בלי ידיעה מאיזה וולידטורים הוא בנוי. אם ברצוננו להוסיף וולידציה חדשה לפעולה מסוימת (למשל לרישום משתמש חדש) עלינו רק להוסיף בבניית השרשרת את המחלקה החדשה היורשת גם היא מValidatorDecorator, ולא לשנות את השיטות עצמן שקוראות לValidatorDecorator ומבקשות ממנו לבצע וולידציה.

# שינויים והרחבות עתידיים

המערכת תוכננה ועוצבה כך ששינויים עתידיים יהיו קלים ליישום.

נראה כעת כיצד הגמישות לשינויים באה לידי ביטוי ואילו שינויים ניתן יהיה לבצע בעתיד.

המערכת בנויה על סמך מודל שלושת השכבות 3-Tier. הפרדה זו יוצרת שכבות אבסטרקציה, כך שבעתיד ניתן יהיה להחליף כל אחת מהשכבות בלי להחליף את שאר השכבות. או להחליף ממשק בין שתי שכבות, בלי להחליף את הממשק בין שתי שכבות אחרות. לדוגמה, אם נחליט שברצוננו לעבוד עם מסד נתונים שונה, או עם framework שונה כמו NHibernate נוכל לעשות זאת ללא כל שינוי בשכבות הBL ובPL. דוגמה נוספת לשינוי היא למשל החלפת שכבת הPL במימוש WEBי (כמעט) ללא שום שינוי בשכבת הBL\DAL (השינוי היחיד שיש לבצע הוא בהגדרת הServices של הBL כך שניתן יהיה לפנות אליהם באמצעות Rest API ושינוי זה הוא מאוד מאוד קטן).

הוספת מסכים חדשים בשכבת התצוגה תתבצע בקלות רבה. חוץ מכתיבת המסך החדש, כל שעלינו לעשות הוא להוסיף את הקוד שמבצע מעבר למסך החדש – כלומר למשל אם נוסיף כפתור חדר לMainMenuViewModel, כל שנצטרך לעשות הוא להוסיף לו Command שיוצר מופע חדש של המסך החדש וקורא לNavigationManager כדי שיעבור למסך החדש.

שינוי מסך קיים – מאחר שמימשנו את שכבת הPL בעזרת MVVM, ניתן יהיה בקלות רבה לשנות את איך שכל מסך נראה על ידי שינוי הView בלבד, ללא שום שינוי בViewModel. גם ההפך נכון, ניתן לשנות למשל מה כפתור מסוים מבצע או פעולת משתמש מסוימת מבצעת רק על ידי שינוי בViewModel ללא שום שינוי בView. למשל אם נרצה למשל לשנות את הכפתורים בMainMenu לHyperlink, נוכל לעשות זאת בקלות בView מבלי לשנות את הקוד שמבצע את הNavigation בViewModel.

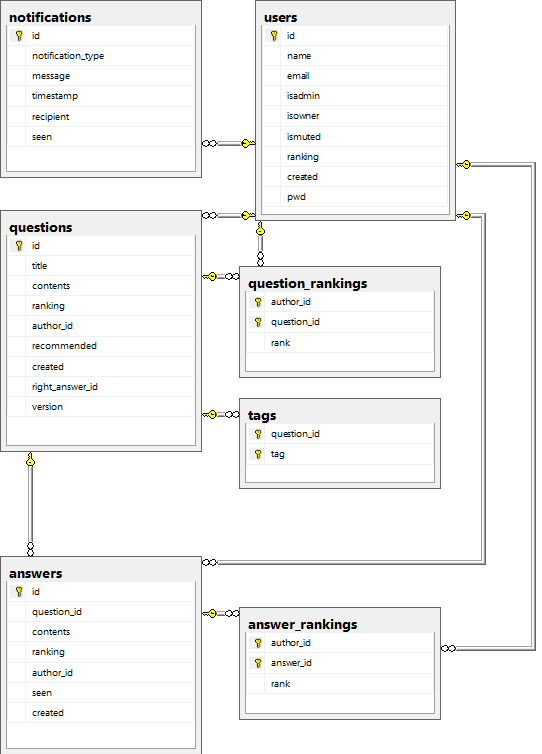
הוספת התרעות חדשות – הצורה שבה בנויה תשתית ההתרעות מאפשרת בעתיד להוסיף התרעות חדשות בקלות. בDB נשמרות ההתראות בטבלה בה רשומה ההתראה, והמשתמש שמיועד לקבל את ההתרעה. בDAL תהיה פעולה שמאפשרת לBL לפנות אליה ולהוסיף את ההתרעה החדשה לDB. בדרך זו, בכל פעולה ניתן להוסיף לBL פניה לDAL שישמור התרעה חדשה עבור מבצע הפעולה (ואולי עוד משתמשים – תלוי בפעולה). למשל, אם נחליט שהזנת שאלה למערכת תודיע לרכז, אז בPostServices בפועלה של הוספת שאלה חדשה, נוסיף קריאה לDAL של שמירת ההתרעה עבור הרכז.

הוספת וולידציות חדשות או שינוי קיימות – מאחר שמימשנו את מנגנון הוולידציות במערכת באמצעות Decorator, כל וולידציה יושבת במחלקה נפרדת. כך, אם נרצה לשנות את הלוגיקה בבדיקה מסוימת, נוכל לשנות אך ורק את המחלקה הספציפית שמבצעת את הבדיקה שברצוננו לשנות, ללא כל שינוי של מחלקות אחרות. בנוסף, אם נרצה להוסיף וולידציה חדשה לפעולה כלשהי במערכת, יהיה ניתן או לעשות Re-Use לוולידציה קיימת, או ליצור מחלקה חדשה של וולידציה, ולהוסיף אותה כחלק מהבניה של הDecorator עבור אותה פעולה. הכוונה בבנייה של Decorator היא החלק בו אנחנו בונים decorator על decorator עד אשר נשאר מופע בודד של validator.

שינוי תוכן הAuthenticationToken – העיצוב הנוכחי של הפרויקט, כתוב כך שהPL אינו מודע אף פעם לתוכן של הAuthenticationToken שהוא מקבל מהBL לאחר חיבור מוצלח למערכת. הדבר היחיד שהPL עושה הוא לשמור את הAuthenticationToken אצלו ולהעביר אותו לBL על מנת לזהות את המשתמש שמבקש לבצע פעולה. בדרך זו הBL הוא היחיד שיודע לכתוב ולפענח את הAuthenticationToken. בדרך זו ניתן למשל בעתיד להחליט למשל להצפין את התוכן של הAuthenticationToken בכל פעם שמשתמש מנסה להתחבר למערכת. ההצפנה יכולה להיות תלויה למשל בתאריך והשעה הנוכחיים כך שבמידה ותהיה מתקפה כלשהי על המערכת על ידי פורצים, הם לא יוכלו לדעת להפיק AuthenticationToken משלהם כדי לבצע פעולות בשם משתמשים קיימים במערכת.

# נספח: דיאגרמות

## דיאגרמת מסד הנתונים (Data Base Diagram)



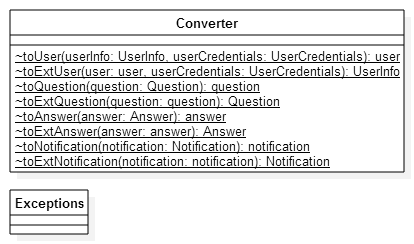
## דיאגרמת מחלקות

הערות כלליות עבור דיאגרמות המחלקות

הדיאגרמות יוצרו בעזרת StarUML2.

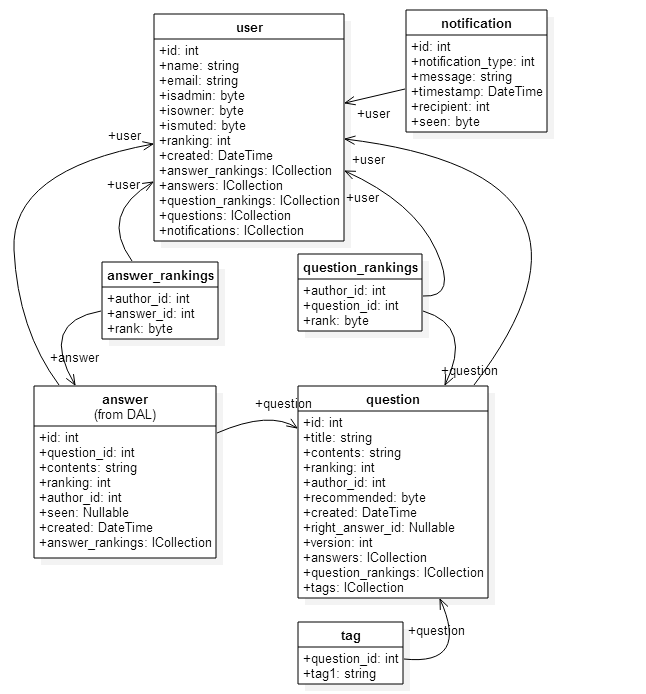
1. הסימן + מסמן גישה ציבורית – public
2. הסימן – מסמן גישה פרטית – private
3. הסימן ~ מסמן גישה פנימית – internal

### מחלקות כלליות בשכבת ה-DAL



### מחלקות הגישה לבסיס הנתונים בשכבת ה-DAL

מחלקות אלו נוצרו אוטומטית ע"י ה-EntityFramework. הדיאגרמה מציגה את הקשרים בינהן. המחלקות שנוצרו ע"י ה-EntityFramework מכילות שיטות רבות לגישה למאגר הנתונים, אשר אינן מופיעות בדיאגרמה זו.



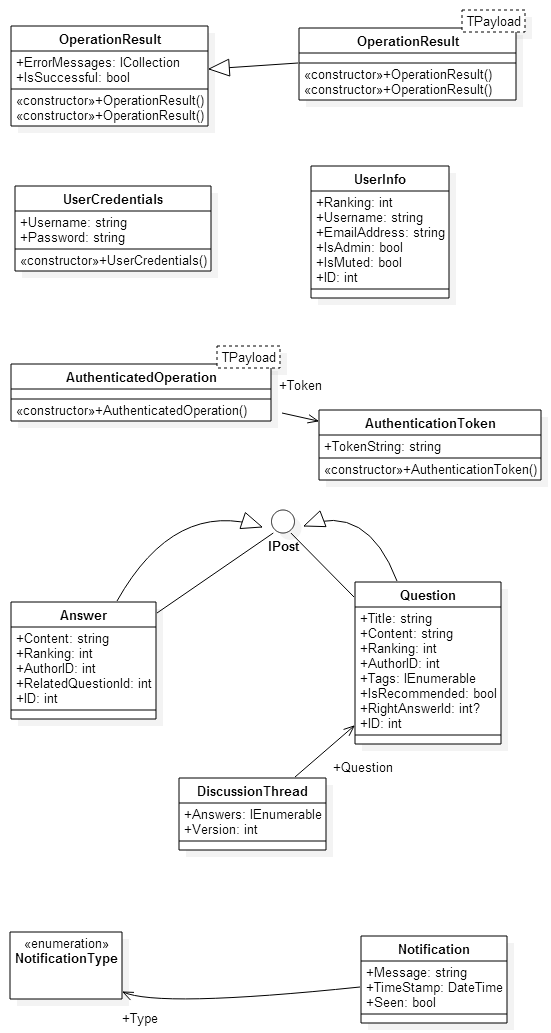
### מחלקות הפעילות בשכבת ה-DAL

מחלקות אלו הן המחלקות שחשופות לשכבת ה-BL לביצוע הפעילות מול מאגר הנתונים.



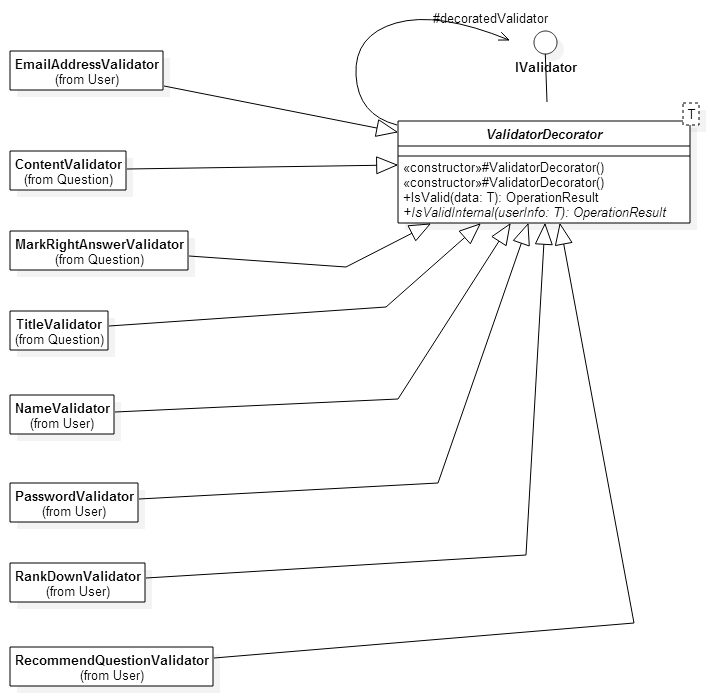
### מחלקות ספריית Common

ספריות אלו כוללות את האובייקטים הבסיסיים במערכת. כל השכבות האחרות משתמשות באובייקטים אלו.



### מחלקות מרחב Common.Logic

במרחב זה קיימים מרחבי User ו-Question המכילים את המימושים של ממשק IValidator בעזרת ירושה מ-ValidatorDecorator המאפשר מימוש של ולידציות לתצוגה ול-BL עבור כלל הפעולות שניתן לבצע במערכת.  
כל המחלקות מממשות את IsValidInternal כפי שמוגדר, ולכן המתודה לא מוצגת בדיאגרמה

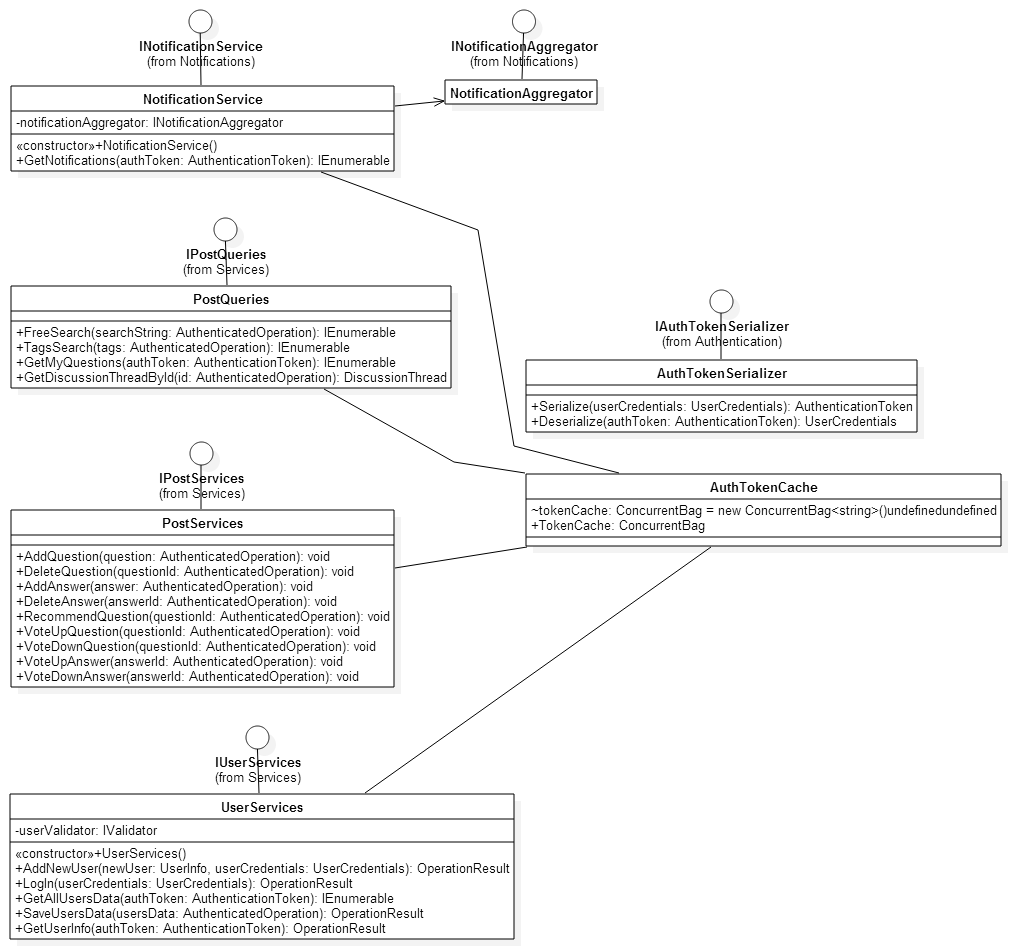


### מחלקות שכבת ה-BL

בשכבה זו נעשה שימוש ב-Services כדי לממש את הפעולות המבוצעות במערכת.

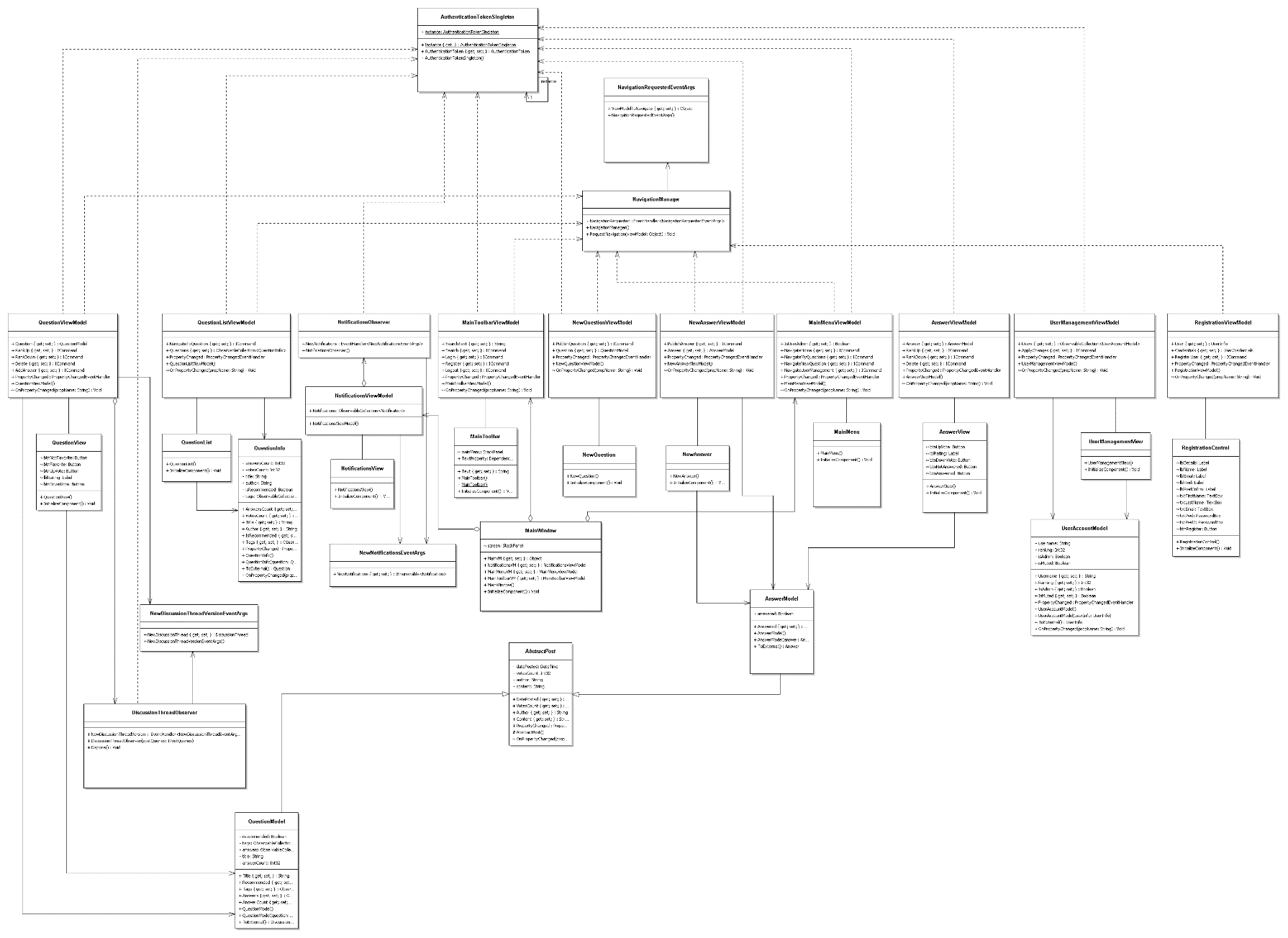
כלל האובייקטים בשכבה ממשים את הממשקים שהוגדרו בשכבת Common, אך מוצגים כאן

כל המחלקות בשכבה זו משתמשות במתודות לאחר שקיבלו הרשאה – כלומר ברשותן AuthenticationToken מתאים



### מחלקות בשכבת התצוגה

בשכבה זו נמצאים כל מסכי התצוגה, המחלקות שפונות לBL, והData Models שנמצאים בPL שנעשה בהם שימוש בשכבה זו (נעשה בשכבה זו גם שימוש בData Models שנמצאים בCommon אבל זה לא מסומן כאן).

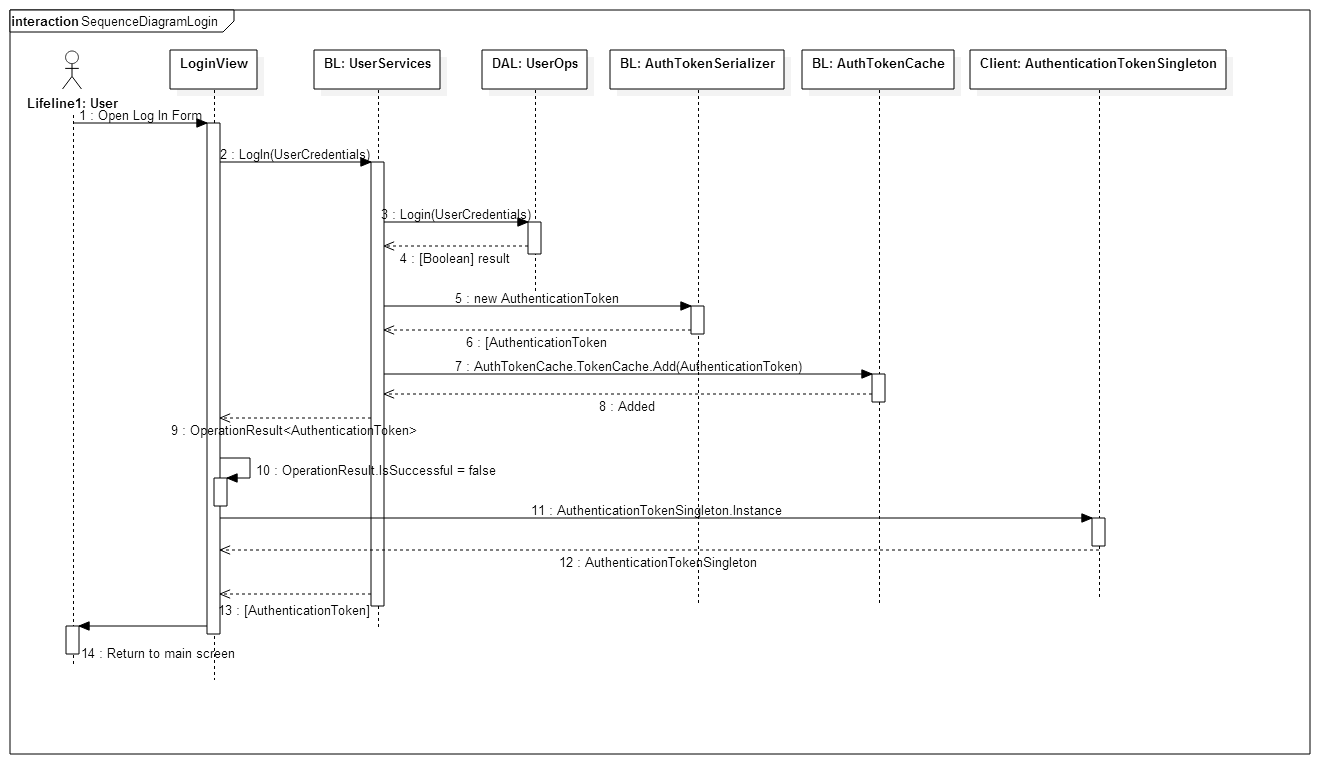


## דיאגרמות רצף - Sequence Diagram

דיאגרמות הרצף המפורטות מטה יתארו חלק מהתהליכים המרכזיים במערכת. התיאור הוא כללי וממצה ומטרתו להסביר את הלוגיקה בפעולות במערכת. בפועל יהיו הבדלים ושינויים בין הדיאגרמות והמימוש בפועל, שכן המימוש, בעיקר בשכבת התצוגה, הינו מורכב ביותר מחד, אבל עקבי וזהה ומתבסס על עקרונות ה-MVVM מאידך.  
כדי שהדיאגרמות יהיו בהירות ויסבירו את תהליכי העבודה, הן יכילו את תהליך העבודה המשמעותי, ולא יצמדו לדקויות בהעברת הודעות לרכיבי .NET WPF ודומיהם.

### דיאגרמת רצף עבור תהליך Login

עם הכניסה למערכת המשתמש יקליק על כפתור Login ב-MainToolBar. בתגובה המערכת תציג את מסך Login בו יקליד המשתמש את שמו ואת סיסמתו למערכת. השם והסיסמא יבדקו מול מאגר הנתונים בעזרת ה-DAL. במידה והשם והסיסמא נכונים, יווצר AuthenticationToken אשר ישמר ב-cache ב-BL כדי לחסוך בדיקות חוזרות מול מאגר הנתונים עבור כל פעולה. ה-Token יוחזר לשכבת הפרזנטציה, וישמר כ-singleton ב-client. מסך ה-login יסגר, והשליטה תוחזר למשתמש במסך הראשי.  
בכל שלב בדרך בו תהיה שגיאה, תוחזר הודעת השגיאה למשתמש במסך ה-login כדי לאפשר לו לתקן ולנסות שוב את התהליך.



### דיאגרמת רצף תהליך ולידציה של משתמש (האם המשתמש הנוכחי ביצע login מסודר למערכת)

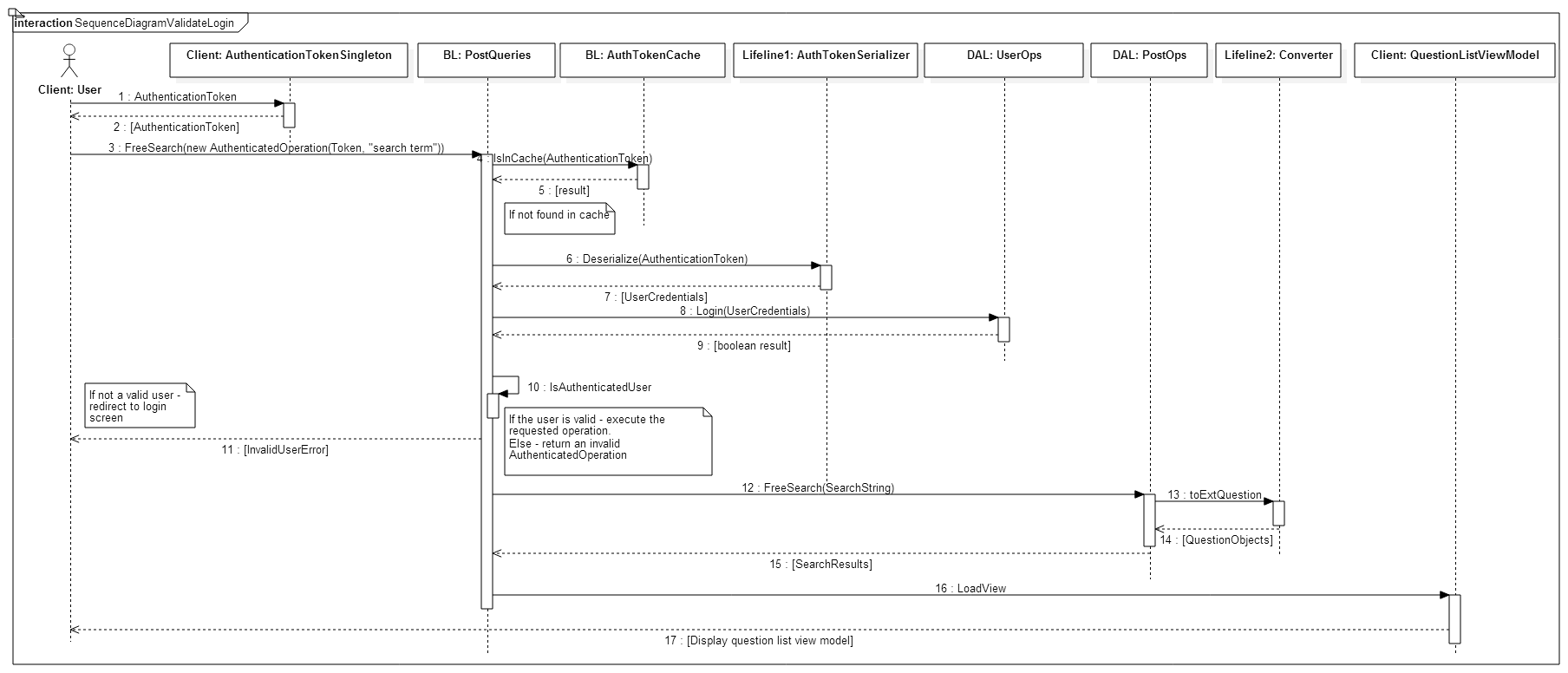
כל בקשה לפעולה של המשתמש במערכת מתחילה בבדיקה האם המשתמש מחובר למערכת.

שכבת התצוגה שומרת את ה-AuthneticationToken של המשתמש ב-singleton בצד האפליקציה.

בצד שכבת ה-BL, ה-Token-ים נשמרים ב-cache.

כל פעולה כוללת פרמטר של AuthneticationToken הנשלח מצד ה-Client עם שאר הפרמטרים של ה-operation. בצד ה-BL הם מושווים ל-Token שב-cache, אם קיים, או מושווים דרך ה-DAL לנתונים שבמאגר הנתונים.  
אם קיימת התאמה, המשתמש מזוהה, וניתן להמשיך ולבדוק האם הוא מורשה לביצוע הפעולה הספציפית, או פשוט לבצעה.

דיאגרמה זו מתארת תהליך ביצוע חיפוש טקסט חופשי במערכת, כיוון שזוהי פעולה פשוטה, אותה מורשים כל המשתמשים לבצע. רק לאחר סיום תהליך הולידציה יבוצע החיפוש בפועל, התוצאות ישלפו ממאגר הנתונים, יומרו לאובייקטים המוכרים (ב-common) וישלחו חזרה לשכבת התצוגה שתציג את ה-view המתאים



### דיאגרמת רצף כתיבת תשובה

דיאגרמה זו באה להדגיש את ההתנהגות בעת הוספת עדכון למערכת, אשר גורר שינויים רבים ב-BL ובעקבותיו במאגר הנתונים, אליהם המשתמש אינו מודע.

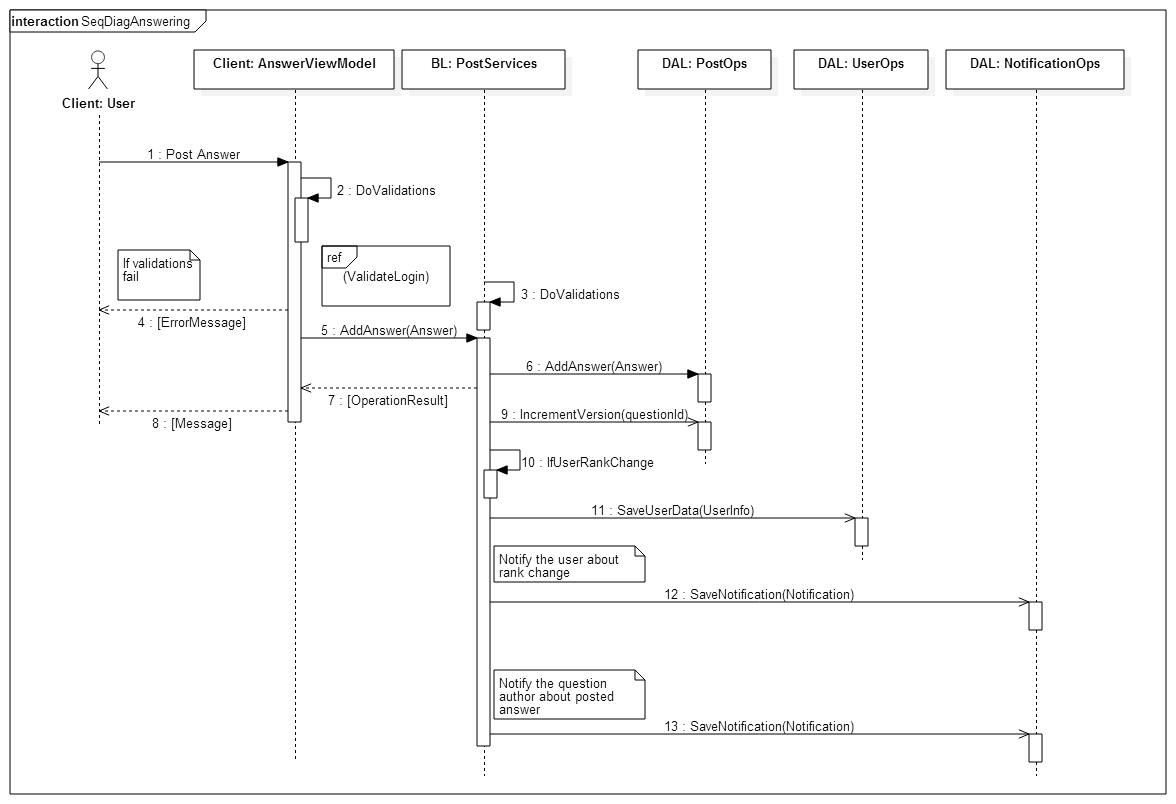
כאשר משתמש כותב תשובה לשאלה ושולח אותה למערכת, המערכת מוודאה שהמשתמש אכן מחובר למערכת (מוסבר בדיאגרמה הקודמת), ומעבירה את התשובה ל-BL שמירה ב-DAL.

בנוסף, מתבצעים התהליכים הבאים:

1. ולידציות לתוכן ההודעה ב-PL.
2. בדיקה האם הגיע הזמן להעלות את דירוג המשתמש במערכת, והעלאת הדירוג במידה וכן
3. יצירת התראה עבור כותב השאלה
4. העלאת ה-version של השאלה.

אחרי ולידציה מוצלחת, למעט שמירת השאלה, שאר הפעולות יכולות להתבצע בצורה אסינכרונית.

אותו תהליך בדיוק קורה בכל פעולה של המשתמש במערכת – בעת שינוי דירוג של שאלה, בעת המלצה על שאלה, כאשר תשובה מסומנת ע"י מרצה כנכונה. דיאגרמה זו מהווה template עבור כל אותן פעולות, שההבדל היחידי בינן הוא הנתונים שהמשתמש מעדכן.



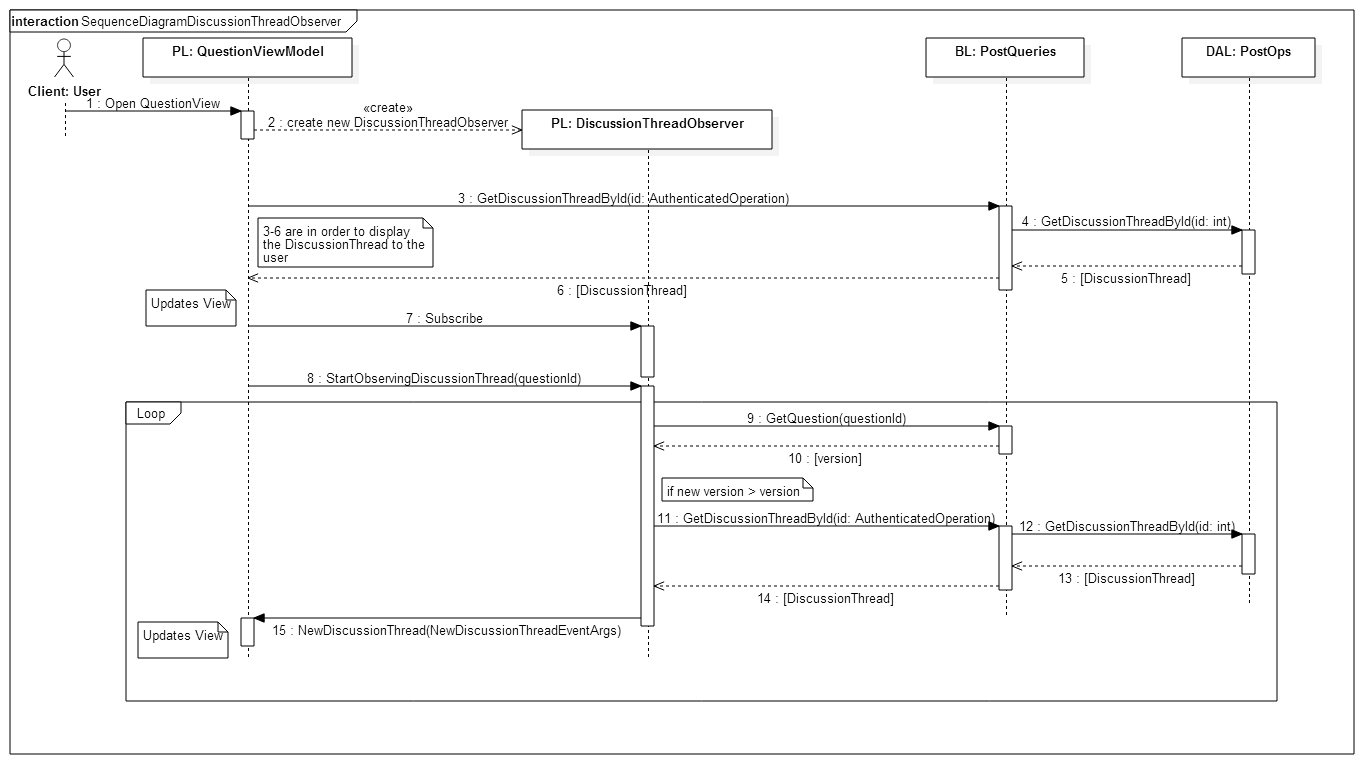
### דיאגרמת רצף מנגנון עדכון התצוגה

דיאגרמות אלו ממחישות את פעולתו של מנגנון ההתרעות במערכת. בעת צפיה במסך כלשהו, פועל ברקע מנגנון ההתרעות. במידה ו-version של שאלה (ולכן של כל ה-DiscussionThread) משתנה, מנגנון ההתרעות דוחף את השינויים למקומות הרלוונטיים במסך, כדי למנוע מצב בו משתמש שמשתהה במסך מסויים לא ידע שהנתונים שמוצגים בו כבר אינם עדכניים.

מנגנון ההתרעות פונה כל פרק זמן קבוע ל-DAL בבקשה לקבלת גרסת השאלה העדכנית. במידה והיא שונה מהגרסה הנוכחית, הוא שולף את פרטי ה-DiscussionThread העדכניים ומעדכן את התצוגה.

הרצף הוא כזה – ה-QuestionViewModel נרשם ל-Event של ה-DiscussionThreadObserver. ואז קורא ל- StartObservingDiscussionThread כשהוא מעביר לו את מזהה השאלה שצריך לנטר.

ה-DiscussionObserverThread מריץ Thread שבלולאה שולף מה-DAL את נתוני השאלה ובודק אם ה-Version השתנה, ואם כן, הוא שולף את נתוני ה-DiscussionThread ממאגר הנתונים ומקפיץ את ה-Event



בנוסף, ובמקביל, מנגנון ההתרעות שולף התרעות כלליות עבור המשתמש, במידה והן קיימות הוא מציג אותן בפאנל ההתרעות. כאן, הוא שולף את ההתרעות בהתאם לחשבון המשתמש – במידה ומדובר בסטודנט הוא שולף התרעות רק עבור המשתמש, במידה ומדובר במרצה הוא שולף התרעות עבור מרצה.

