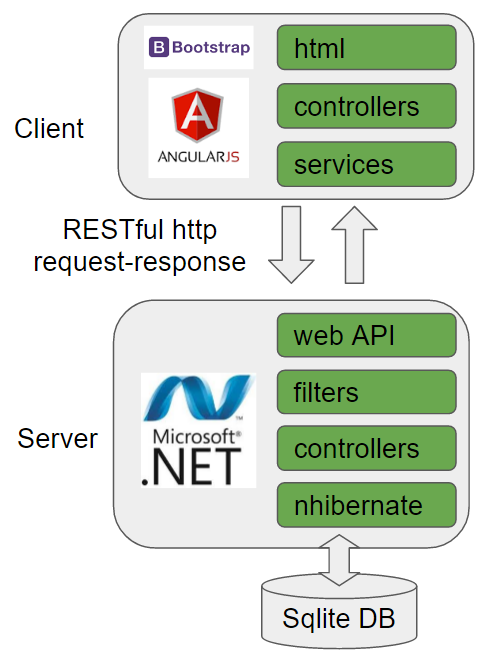
|  |  |
| --- | --- |
| EZVET  מערכת לניהול העולם הוטרינרי | מסמך עיצוב  מסמך זה מתאר את הדרישות ממערכת EZVet, מערכת לניהול העולם הוטרינרי.  רון גל  סדנה בתכנות מונחה עצמים |

תוכן עניינים

* תיאור כללי של המערכת 2
  + הנחות עבודה 2
  + מוסכמות רישום 2
* שכבת בסיס הנתונים 3
* טבלאות המערכת 4
  + טבלאות משתמשים 4
  + טבלאות פיענוח 6
  + טבלאות עיסקיות 8
* שכבת הלוגיקה 10
* שכבת השירותים 11
* שכבת התצוגה 12
* פרוייקט היישויות - Domain 13
* פרוייקט מיפוי הישיות - Maps 19
* פרוייקט השירותים - PlaySimple 21
  + נינג'קט - Ninject IoC Container 21
  + Query Processors 22
  + קונטרולרים - Controllers 26
  + ולידטורים - Validators 29
  + פילטרים - Filters 30
  + מחלקות מעבר - Dtos 31
  + מחלקות תשתית - Common 36
  + אורך חיי בקשה 38
* פרוייקט האתר - PlaySimpleSite 39
  + מסכים 39
  + קבצי עיצוב 42
  + קבצי javascript 42
* דפוסי עיצוב 44
  + Singleton 44
  + Factory Method 44
  + Decorator 44
* שינויים עתידיים 45
* דיאגרמות 46
  + מודל ישויות קשרים - ERD 46
  + דיאגרמות מחלקה - Class Diagrams 47
  + דיאגרמות רצף - Sequence Diagram 50

תיאור כללי של המערכת

המערכת תבנה באריכטקטורת שרת-לקוח כך שלשרת יהיה גישה למסד נתונים.

שכבת השרת והלקוח יתקשרו ע"י פרוטוקול http באופן שבו כל פניה עומדת בפני עצמה (stateless).

השרת יפנה למסד הנתונים ע"י O/R Mapping במימוש של NHibernate.

מוסכמות רישום

1. שמות הטבלאות ושדותיהן במסד הנתונים בפורמט PascalCase, כלומר אותיות קטנות פרט לאות הראשונה בכל מילה.
2. בצד השרת, שמות התיקיות, המחלקות והפונקציות יכתבו בפורמט PascalCase. בנוסף, פרמטרים של פונקציות ומשתנים יכתבו בפורמט של camelCase, כלומר מילה ראשונה באותיות קטנות וכל אות במילה נוספת תכתב באות גדולה.
3. קבצי צד הלקוח יכתבו בפורמט camelCase, כלומר מילה ראשונה באותיות קטנות וכל אות ראשונה במילה נוספת תכתב באות גדולה.
4. משתנים פרטיים ברמת המחלקה, יתחילו בקו תחתון ומילה ראשונה באותיות קטנות וכל אות ראשונה במילה נוספת תכתב באות גדולה (מוסכמה של C#).
5. שמות מחלקות ה-Domain (שהן ייצוג הישויות במסד הנותנים בקוד השרת) יהיו זהים לשמות הטבלאות.
6. כל מחלקה השייכת לקבוצה מסויימת תסתיים בסיומת הקבוצה להסברת תפקידה למשל ה- controller הוטרינרים יקרא VetsController.

שכבת בסיסי הנתונים

שכבה זו אחראית על תקשורת מול מסד הנתונים וביצוע פעולות שמירה, עדכון, מחיקה ושליפה של נתונים. המערכת משתמשת במסד נתונים מסוג Sqlite, בשם db.sqlite, הנמצא תחת תיקיית App\_Data.

שכבה זו אחראית על מספר תהליכים שונים במהלך חיי המערכת:

* כאשר המערכת עולה מאותחל אובייקט SessionFactory, שהינו Singleton, ויודע לייצר עצמים מסוג Session. עצמים אלו מייצגים חיבור למסד הנתונים, ובעזרתם ניתן לבצע פעולות במסד הנתונים.
* עבור כל בקשה המגיעה לשרת מאותחל ונשמר Session אשר יהיה זמין לבקשה לכל אורך חייה. תחת Session, עבור כל בקשה, מאותחל אובייקט מסוג Transaction (טרנזאקציה).
* כאשר נגמרת בקשה, אם התבצעו שינויים ב-Session, הם נשמרים במסד הנתונים או לחלופין מתבטלים במידה ואירעה שגיאה.
* כאשר המערכת נסגרת, מושמד אובייקט ה-Factory לצורך שחרור משאבים.

שכבת בסיס הנתונים מבוססת על Nhibernate שהינו פרויקט קוד פתוח המהווה Object Relation Mapper. פרוייקטים מסוג זה ממפים מחלקות בקוד לטבלאות במסד נתונים. Nhibernate מאפשר תקשורת מול בסיס נתונים באופן בלתי תלוי בסוג מסד הנתונים (בהתאם למסדי הנתונים הנתמכים). בעזרת Nhibernate ניתן לבצע שליפות מורכבות בשפת Linq. למשל, אגריגציות והצלבת מידע בין טבלאות. בנוסף, ניתן לבצע פעולות CRUD בעזרת אובייקט ללא כתיבה של שאילתות באופן ישיר.

לצורך אתחול Nhibernate יש להגדיר עבורו אילו מחלקות ממופות לטבלאות במסד הנתונים. הגדרות אלו נמצאות תחת הפרויקט בשם Maps. בעת עליית המערכת, אם מאפייני המחלקה השתנו, שינויים אלו יחלחלו לבסיס הנתונים באופן אוטומטי. כלומר, אם נוסף שדה / מחלקה חדשה והיא מופתה ב-Nhibernate - תתווסף עמודה / טבלה חדשה בהתאם במסד הנתונים. יש לציין שסוג העדכון הוא Non-Destructive Update, ולכן לעולם לא ימחק מידע מבסיס הנתונים, אלא רק יתווסף, מחיקת עמודות וטבלאות יש לבצע באופן ידני.

עקרונות מנחים לגבי שכבת הנתונים:

* המידע בטבלאות הנתונים נשמר בצורה מנורמלת לצורך מניעת כפילות מידע ואמינות.
* בחרנו לממש מדיניות של Session Per Call, כלומר עבור כל בקשה לאורך כל חייה, יהיה זמין עבורה Session בודד. מדיניות זו מפחיתה את כמות החיבורים הפתוחים למסד הנתונים בכל רגע נתון ובכך משפרת את הביצועים.
* ה-Session ישמר ב-IoC Container כך שיהיה זמין לשכבה זו.
* השם לכל המפתחות הראשיים הוא Id, השם של כל מפתח זר מורכב משם הישות, קו תחתון ו-Id, למשל Status\_Id.
* מחרוזות קבועות נשמרות במסד הנתונים בטבלאות פענוח כדאי לאפשר גמישות במחרוזות המוצגות למשתמש.

טבלאות המערכת

טבלאות משתמשים

**ADMIN**

טבלת המשתמשים המוגדרים כמנהלים במערכת

|  |  |  |  |
| --- | --- | --- | --- |
| **עמודה** | **טיפוס** | **אורך** | **הסבר** |
| ID | מספרי | 10 | מפתח ראשי לטבלה |
| USER\_NAME | מחרוזתי | 20 | שם המשתמש בעת כניסה למערכת |
| PASSWORD | מחרוזתי | 20 | סיסמת המשתמש בעת כניסה למערכת |
|  |  |  |  |
|  |  |  |  |

**EMPLOYEE**

טבלת המשתמשים המוגדרים כעובדים במערכת

|  |  |  |  |
| --- | --- | --- | --- |
| **עמודה** | **טיפוס** | **אורך** | **הסבר** |
| ID | מספרי | 10 | מפתח ראשי לטבלה |
| USER\_NAME | מחרוזתי | 20 | שם המשתמש של העובד בעת כניסה למערכת |
| PASSWORD | מחרוזתי | 20 | סיסמת המשתמש של העובד בעת כניסה למערכת |
| FIRST\_NAME | מחרוזתי | 20 | שמו הפרטי של העובד |
| LAST\_NAME | מחרוזתי | 20 | שם משפחתו של העובד |
| SALARY | מספרי | 10 | משכורתו של העובד |
| EMAIL | מחרוזתי | 50 | כתובת הדואר האלקטרוני של העובד |

**CUSTOMER**

טבלת המשתמשים המוגדרים כלקוחות במערכת

|  |  |  |  |
| --- | --- | --- | --- |
| **עמודה** | **טיפוס** | **אורך** | **הסבר** |
| ID | מספרי | 10 | מפתח ראשי לטבלה |
| USER\_NAME | מחרוזתי | 20 | שם המשתמש של הלקוח בעת כניסה למערכת |
| PASSWORD | מחרוזתי | 20 | סיסמת המשתמש של הלקוח בעת כניסה למערכת |
| FIRST\_NAME | מחרוזתי | 20 | שמו הפרטי של הלקוח |
| LAST\_NAME | מחרוזתי | 20 | שם משפחתו של הלקוח |
| BIRTH\_DATE | תאריכי |  | תאריך יום הולדתו של הלקוח, לחישוב הגיל |
| FREEZE\_DATE | מספרי | 10 | תאריך הקפאת חשבון המשתמש |
| REGION\_DECODE\_ID | מספרי | 10 | הפנייה לאיזור מגורי הלקוח |

טבלאות פיענוח

**REGION\_DECODE**

טבלת איזורי המגורים הלקוחות

|  |  |  |  |
| --- | --- | --- | --- |
| **עמודה** | **טיפוס** | **אורך** | **הסבר** |
| ID | מספרי | 10 | קוד איזור |
| NAME | מחרוזתי | 20 | פיענוח איזור |

**FIELD\_SIZE\_DECODE**

טבלת גדלי המגרשים

|  |  |  |  |
| --- | --- | --- | --- |
| **עמודה** | **טיפוס** | **אורך** | **הסבר** |
| ID | מספרי | 10 | קוד גודל מגרש |
| NAME | מחרוזתי | 20 | פיענוח גודל מגרש |

**FIELD\_TYPE\_DECODE**

טבלת סוגי המגרשים

|  |  |  |  |
| --- | --- | --- | --- |
| **עמודה** | **טיפוס** | **אורך** | **הסבר** |
| ID | מספרי | 10 | קוד סוג מגרש |
| NAME | מחרוזתי | 20 | פיענוח סוג מגרש |

**INVITATION\_STATUS\_DECODE**

טבלת סטטוסי בקשות להצטרפות של לקוחות להזמנות של לקוחות אחרים

|  |  |  |  |
| --- | --- | --- | --- |
| **עמודה** | **טיפוס** | **אורך** | **הסבר** |
| ID | מספרי | 10 | קוד סטטוס בקשה |
| NAME | מספרי | 10 | פיענוח סטטוס בקשה |

**ORDER\_STATUS\_DECODE**

טבלת סטטוסי הזמנת מגרש של לקוחות

|  |  |  |  |
| --- | --- | --- | --- |
| **עמודה** | **טיפוס** | **אורך** | **הסבר** |
| ID | מספרי | 10 | קוד סטטוס הזמנה |
| NAME | מחרוזתי | 20 | פיענוח סטטוס הזמנה |

**COMPLAINT\_TYPE\_DECODE**

טבלת סטטוסי תלונות של לקוחות על לקוחות אחרים

|  |  |  |  |
| --- | --- | --- | --- |
| **עמודה** | **טיפוס** | **אורך** | **הסבר** |
| ID | מספרי | 10 | קוד סטטוס תלונה |
| NAME | מחרוזתי | 20 | פיענוח סטטוס תלונה |

טבלאות עיסקיות

**COMPLAINT**

טבלת התלונות של לקוחות על לקוחות אחרים

|  |  |  |  |
| --- | --- | --- | --- |
| **עמודה** | **טיפוס** | **אורך** | **הסבר** |
| ID | מספרי | 10 | מפתח ראשי לטבלה |
| DESCRIPTION | מחרוזתי | 20 | הסבר התלונה |
| START | תאריכי |  | תאריך הזנת התלונה |
| COMPLAINT\_TYPE\_ID | מספרי | 10 | סוג התלונה |
| OFFENDED\_CUSTOMER\_ID | מספרי | 10 | הפנייה ללקוח המתלונן |
| OFFENDING\_CUSTOMER\_ID | מספרי | 10 | הפנייה ללקוח עליו התלוננו |

**FIELD**

טבלת המגרשים הניתנים להשכרה במערכת

|  |  |  |  |
| --- | --- | --- | --- |
| **עמודה** | **טיפוס** | **אורך** | **הסבר** |
| ID | מספרי | 10 | קוד מגרש |
| NAME | מחרוזתי | 20 | שם המגרש |
| FIELD\_TYPE\_ID | מספרי | 10 | הפנייה לסוג המגרש |
| FIELD\_SIZE\_ID | מספרי | 10 | הפנייה לגודל המגרש |

**ORDER**

טבלת הזמנות המגרשים של לקוחות

|  |  |  |  |
| --- | --- | --- | --- |
| **עמודה** | **טיפוס** | **אורך** | **הסבר** |
| ID | מספרי | 10 | קוד הזמנה |
| START\_TIME | תאריכי |  | מועד התחלת ההשכרה |
| PLAYERS\_NUMBER | מספרי | 2 | מספר שחקנים מקסימאלי להצטרפות |
| FIELD\_ID | מספרי | 10 | הפנייה למגרש המושכר |
| CUSTOMER\_ID | מספרי | 10 | הפנייה ללקוח המזמין |
| ORDER\_STATUS\_ID | מספרי | 10 | הפנייה לסטטוס ההזמנה |

**PARTICIPANT**

טבלת בקשות לקוחות להצטרפות להזמנות של לקוחות אחרים

|  |  |  |  |
| --- | --- | --- | --- |
| **עמודה** | **טיפוס** | **אורך** | **הסבר** |
| ID | מספרי | 10 | קוד בקשת הצטרפות, מפתח ראשי |
| DATE | תאריכי |  | תאריך בקשת ההזמנה |
| CUSTOMER\_ID | מספרי | 10 | הפנייה ללקוח המעוניין להצטרף |
| ORDER\_ID | מספרי | 10 | הפנייה להזמנה אליה הלקוח מעניין להצטרף |
| INVITATION\_STATUS\_ID | מספרי | 10 | הפנייה לסטטוס הבקשה |

**REVIEW**

טבלת חוות הדעת של לקוחות על לקוחות אחרים

|  |  |  |  |
| --- | --- | --- | --- |
| **עמודה** | **טיפוס** | **אורך** | **הסבר** |
| ID | מספרי | 10 | קוד חוות דעת, מפתח ראשי |
| TITLE | מחרוזתי | 70 | כותרת חוות הדעת |
| DESCRIPTION | מחרוזתי | 400 | תיאור חוות הדעת |
| DATE | תאריכי |  | תאריך כתיבת חוות הדעת |
| REVIWER\_CUSTOMER\_ID | מספרי | 10 | הפנייה ללקוח שכתב את חוות הדעת |
| REVIWED\_CUSTOMER\_ID | מספרי | 10 | הפנייה ללקוח עליו נכתבה חוות הדעת |

שכבת הלוגיקה

שכבה זו מהווה את הגשר בין שכבת הנתונים לשכבת השירותים המוצעים באתר. עבור כל ישות במערכת תמומש מחלקה האחראית על ביצוע הלוגיקה הרלוונטית אליה.

עקרונות מנחים בשכבת הלוגיקה:

* שכבת הלוגיקה תפנה לבסיס הנתונים אך ורק בעזרת שכבת בסיס הנתונים, תוך שימוש בממשק הגנרי IDatabaseAccess.
* כל מחלקה בשכבת הלוגיקה תממש ממשק מוגדר מראש וזאת בשביל להפריד התנהגות ממימוש. אסטרטגיה זו תפחית את הצימוד בין מחלקות.
* כל מחלקה תהיה אחראית על ישות אחת וכך יבוא לידי מימוש עקרון

ה-Single Responsibility.

* הקוד יכתב בצורה מודולארית כך שיתמוך בהרחבות ללא צורך בשינוי קוד קיים וכך יבוא לידי מימוש עקרון ה-Open/Closed.
* המחלקות בשכבה זו ישמרו ב-IoC Container ויוזרקו לשכבת השירותים ובכך יבוא לידי מימוש עקרון ה-Dependency Inversion.
* שכבת הלוגיקה תחשוף לשכבת השירותים את המינימום הנדרש לצורך הפעלתה ובכך יבוא לידי ביטוי עקרון ה-Encapsulation.

שכבת השירותים

שכבה זו חושפת את נתוני ויכולות המערכת בעזרת ממשק המבוסס על בקשות Http בארכיטקטורת Rest וטכנולוגיית Web API של מיקרוסופט. תשובות לבקשות ישלחו בפורמט JSON. בעת פנייה לשכבה זו תבדק תקינות נתוני הבקשה. במידה ונתוני הבקשה ימצאו לא תקינים או לחלופין אירעה שגיאה כלשהי תוחזר הודעה מתאימה ללקוח.

עקרונות מנחים בשכבת השירותים:

* שכבת השירותים תפנה אך ורק לשכבת הלוגיקה לצורך הבאת נתונים, תוך שימוש בממשקים אשר הוגדרו בשכבה זו.
* לכל ישות במערכת ימומש שירות בהתאם לשם הישות בתופסת המילה Controller, למשל עבור המגרשים יוגדר השירות FieldsController. שירות זה יהיה זמין באתר בכתובת היחסית api/fields.
* כל בקשה תנותב לפונקציה הרלוונטית בכל שירות תופעל בהתאם לפרמטרים שנשלחו בבקשה ובהתאם ל-HttpVerb שנשלח.
  + הפועל Get ישמש לצורך הבאת מידע.
  + הפועל Post ישמש לצורך שמירת מידע חדש.
  + הפועל Put ישמש לצורך עדכון מידע קיים.
  + הפועל Delete ישמש לצורך מחיקת מידע קיים.
* תשובות לבקשות יכילו סטאטוס Http מקובל, למשל 200 (OK) עבור תשובה תקינה ו-500 (Internal Server Error) עבור שגיאת שרת.

שכבת התצוגה

שכבה זו מציגה ממשק משתמש (UI), על בסיס טכנולוגיית html, css ו-javascript על גבי הדפדפן ומנגישה למשתמש את יכולות שכבת השירותים. בחלק העליון של כל דף במערכת יוצג תפריט עם קישורים לכל הדפים הזמינים למשתמש בהתאם לרמת הרשאותיו.

עקרונות מנחים של בשכבת התצוגה:

* ממשק המשתמש יהיה אינטואיטיבי ואחיד בכל דפי המערכת.
* שכבת התצוגה תתנהג באופן זהה בכל הדפדפנים עבור הגרסאות האחרונות שלהם.
* שכבת התצוגה תתמוך בכל הרזולוציות עבור צגי מחשב. יש להדגיש כי לא תינתן תמיכה ברזולוציות נמוכות יותר (למשל, פלאפונים) .
* ממשק המשתמש יבנה בעזרת Grid כך שיהיה גמיש לשינויים.

פרויקט הישויות (Domain)

בפרויקט זה קיימות המחלקות אשר ייצגו בעזרת Nhibernate רשומות במסד הנתונים. מחלקות אלו אינן מכילות לוגיקה אלא מאפיינים בלבד כדאי לא לערבב בין שני חלקים בלתי תלויים במערכת. כל ישות ממופה לטבלה אחת במסד הנתונים. שמות המאפיינים במחלקות זהים לשמות העמודות במסד הנתונים. קשרים בין טבלאות מסוג one-to-many ו- many-to-one יבוטאו כמאפיינים מסוג רשימה מקושרת והצבעות למחלקות אחרות בהתאם. קשרים בין טבלאות יקבלו סיומת המורכבת מקו תחתון ולאחריו המילה id. חשוב לציין כי כל המאפיינים במחלקות הללו הינם מסוג virtual משיקולים טכנים הנובעים מאופן המימוש של Nhibernate.

**מחלקת Entity**

מחלקה מופשטת זו מהווה את הבסיס לכל המחלקות אשר מייצגות ישויות הנמצאות במסד הנתונים. היא מכילה מאפיין בודד, שהינו המפתח הראשי בכל הישויות במסד הנתונים. המחלקה בעלת התכונות הבאות:

public virtual int Id { get; set; }

**מחלקת Decode**

מחלקה מופשטת זו מהווה את הבסיס לכל המחלקות אשר מייצגות טבלאות פענוח במסד הנתונים. הצורך במחלקות פענוח מתעורר כאשר יש לשמור בשדה כלשהו מחרוזת מתוך אוסף קבוע של מחרוזות. במקום לשמור את המחרוזת באופן ישיר כחלק מהרשומה, ישמר בשדה מצביע לרשומה בטבלה אחרת אשר יחזיק את המחרוזת המבוקשת. כך ניתן לשנות בקלות את ערך המחרוזות בלי לעבור סדרתית על כל הרשומות ואפילו לתמוך במספר שפות שונות (בעזרת מספר שינויים נוספים). המחלקה בעלת התכונות הבאות:

public virtual int Id { get; set; }

public virtual string Name { get; set; }

**מחלקת Admin**

מחלקה זו יורשת ממחלקת Entity ומייצגת מנהל במערכת. מחלקה מכילה תכונות את התכונות הבאות:

public virtual string Username { get; set; }

public virtual string Password { get; set; }

**מחלקת UserBase**

מחלקה מופשטת זו יורשת ממחלקת Entity ומהווה את הבסיס לכלל מחלקות המשתמשים במערכת (לקוחות, עובדים). המחלקה בעלת התכונות הבאות:

public virtual string Username { get; set; }

public virtual string Password { get; set; }

public virtual string FirstName { get; set; }

public virtual string LastName { get; set; }

**מחלקת Employee**

מחלקה זו יורשת ממחלקת UserBase ומייצגת עובד במערכת. היא מכילה את המשכורת והאימייל של העובד. המחלקה בעלת התכונות הבאות:

public virtual int Salary { get; set; }

public virtual string Email { get; set; }

**מחלקת Customer**

מחלקה זו יורשת ממחלקת UserBase ומייצגת משתמש במערכת. היא מכילה כתובת הדואר האלקטרוני של הלקוח, תאריך יום הולדתו, תאריך הקפאת החשבון (במידה והוקפא). המחלקה מכילה הפנייה לאזור מגורי הלקוח ואת הרשימות הבאות: ההזמנות אשר הלקוח יזם, ההזמנות אליהן הלקוח ביקש להצטרף והתלונות והביקורות שנכתבו עליו. המחלקה בעלת התכונות הבאות:

public virtual DateTime BirthDate { get; set; }

public virtual string Email { get; set; }

public virtual DateTime FreezeDate { get; set; }

public virtual RegionDecode Region { get; set; }

public virtual IList<Order> Orders { get; set; }

public virtual IList<Participant> ParticipationRequests {get; set;}

public virtual IList<Complaint> Complaints { get; set; }

public virtual IList<Review> Reviews { get; set; }

**מחלקת RegionDecode**

מחלקה זו יורשת ממחלקת Decode ומייצגת אזור מגורי לקוח. הערכים התקינים הם גוש דן, נגב, ירושלים וחיפה. מחלקה זו לא מכילה תכונות נוספות מעבר לתכונות במחלקת הבסיס.

**מחלקת FieldSizeDecode**

מחלקה זו יורשת ממחלקת Decode ומייצגת גדלי המגרש הקיימים. הערכים התקינים הם: קטן בינוני וגדול. מחלקה זו לא מכילה תכונות נוספות מעבר לתכונות במחלקת הבסיס.

**מחלקת FieldTypeDecode**

מחלקה זו יורשת ממחלקת Decode ומייצגת את סוגי המגרשים הקיימים. הערכים התקינים הם כדורגל, כדורסל וטניס. מחלקה זו לא מכילה תכונות נוספות מעבר לתכונות במחלקת הבסיס.

**מחלקת InvitationStatusDecode**

מחלקה זו יורשת ממחלקת Decode ומייצגת סטטוס בקשת הצטרפות. הערכים התקינים הם אושר ונדחה (ע"י יוזם הזמנת המגרש). מחלקה זו לא מכילה תכונות נוספות מעבר לתכונות במחלקת הבסיס.

**מחלקת OrderStatusDecode**

מחלקה זו יורשת ממחלקת Decode ומייצגת סטטוס הזמנה של מגרש. הערכים התקינים הם: ממתין לאישור, אושר, בוטל (ע"י הלקוח) ונדחה (ע"י עובד). מחלקה זו לא מכילה תכונות נוספות מעבר לתכונות במחלקת הבסיס.

**מחלקת ComplaintTypeDecode**

מחלקה זו יורשת ממחלקת Decode ומייצגת את סוגי התלונה השונים שניתן להגיש על משתמש. הערכים התקינים הם אי תשלום, אי הגעה למשחק והתנהגות לא ספורטיבית. מחלקה זו לא מכילה תכונות נוספות מעבר לתכונות במחלקת הבסיס.

**מחלקת Complaint**

מחלקה זו יורשת ממחלקת Entity ומייצגת תלונה על משתמש במערכת. היא מכילה תיאור של התלונה, הפנייה לסוג התלונה ושתי הפניות למשתמשים - המשתמש אשר התלונן והמשתמש אשר עליו התלוננו. המחלקה בעלת התכונות הבאות:

public virtual string Description { get; set; }

public virtual ComplaintTypeDecode Type { get; set; }

public virtual DateTime Date { get; set; }

public virtual Customer OffendingCustomer { get; set; }

public virtual Customer OffendedCustomer { get; set; }

**מחלקת Field**

מחלקה זו יורשת ממחלקת Entity ומייצגת מגרש במערכת. היא מכילה את שם המגרש, הפנייה לסוג המגרש והפנייה לגודל המגרש. היא בעלת התכונות הבאות:

public virtual string Name { get; set; }

public virtual FieldTypeDecode Type { get; set; }

public virtual FieldSizeDecode Size { get; set; }

**מחלקת Order**

מחלקה זו יורשת ממחלקת Entity ומייצגת הזמנה של משתמש למגרש במערכת. היא מכילה את מספר השחקנים בהזמנה, שעות ההתחלה והסיום של ההזמנה. המחלקה מכילה הפנייה למשתמש אשר יזם את הבקשה, הפנייה לסטטוס הבקשה והפנייה למגרש עצמו. בנוסף, המחלקה מכילה את רשימת המשתתפים (פרט ליוזם הבקשה) הנוספים בהזמנה. המחלקה בעלת התכונות הבאות:

public virtual DateTime StartDate { get; set; }

public virtual User Owner { get; set; }

public virtual int PlayersNumber { get; set; }

public virtual OrderStatusDecode Status { get; set; }

public virtual Field Field { get; set; }

public virtual IList<Participant> Participants { get; set; }

**מחלקת Participant**

מחלקה זו יורשת ממחלקת Entity ומייצגת בקשת הצטרפות של לקוח להזמנה קיימת במערכת. היא מכילה את תאריך הבקשה, הפנייה לסטטוס הבקשה, הפנייה למשתמש אשר יזם את בקשת ההצטרפות והפנייה להזמנה עצמה. המחלקה מכילה את התכונות הבאות:

public virtual Customer Customer { get; set; }

public virtual Order Order { get; set; }

public virtual DateTime Date { get; set; }

public virtual InvitationStatusDecode Status { get; set; }

**מחלקת Review**

מחלקה זו יורשת ממחלקת Entity ומייצגת ביקורת של לקוח על לקוח אחר במערכת. היא מכילה כותרת ותיאור הביקורת, תאריך הביקורת ושתי הפניות למשתמשים - המשתמש אשר כתב הביקורת והמשתמש עליו כתבו את הביקורת. המחלקה בעלת התכונות הבאות:

public virtual string Title { get; set; }

public virtual string Description { get; set; }

public virtual DateTime Date { get; set; }

public virtual Customer Reviewer { get; set; }

public virtual Customer ReviewedCustomer { get; set; }

פרויקט מיפויי הישויות (Maps)

כדי ש-Nhibernate ידע את הקשר בין המאפיינים במחלקות היישויות השונות ובין הטבלאות במסד הנתונים, יש להגדיר עבור כל ישות קובץ XML המפרט את הקשרים הללו. קונפיגורציה בעזרת קבצי XML איננה גמישה לשינויים (כמו Auto-Refactoring למשל), מסורבלת ומאטה את קצב הפיתוח. לצורך כך יצרו פרוייקט קוד פתוח בשם Fluent Nhibernate המאפשר להגדיר את הקשרים הללו בעזרת קוד.

כל מחלקה בפרויקט זה, בזמן עליית המערכת, מתורגמת לקובץ XML אשר נטען ב-Nhibernate בצורה (כמעט) שקופה למתכנת. Fluent Nhibernate מגדיר מחלקת בסיס גנרית בשם ClassMap מסוג T, כאשר T הוא סוג הישות אותה יש למפות.

לכל ישות אשר הוגדרה בפרויקט ה-Domain מוגדרת מחלקה בעלת שם זהה עם הסופית Map. בכל מחלקה כזו אנו נעזרים במספר פונקציות של Fluent Nhibernate:

* הפונקציה Id - לצורך הגדרת המאפיין הממפה מפתח הראשי בטבלה. על המאפיין במחלקה להיות מספר (int).
* הפונקציה Map - לצורך הגדרת המאפיין הממפה שדה כלשהו בטבלה אשר מכיל מידע על הישות. על המאפיין במחלקה להיות טיפוס פרימיטיבי כלשהו (int, bool DateTime וכו').
* הפונקציה References - לצורך הגדרת מאפיין הממפה קשר מסוג many-to-one. על המאפיין להיות מצביע לישות אחרת אשר גם ממופה בעזרת Nhibernate.
* הפונקציה HasMany - לצורך הגדרת מאפיין הממפה קשר מסוג one-to-many. על המאפיין להיות מצביע לרשימת ישויות מסוג אשר גם ממופה בעזרת Nhibernate.

**מחלקת AdminMap**

מחלקה זו יורשת ממחלקת ClassMap של Fluent Nhibernate ומגדירה את הקשר בין מחלקת Admin לטבלת ADMIN במסד הנתונים.

**מחלקת EmployeeMap**

מחלקה זו יורשת ממחלקת ClassMap של Fluent Nhibernate ומגדירה את הקשר בין מחלקת Employee לטבלת EMPLOYEE מסד הנתונים.

**מחלקת CustomerMap**

מחלקה זו יורשת ממחלקת ClassMap של Fluent Nhibernate ומגדירה את הקשר בין מחלקת Customer לטבלת CUSTOMER מסד הנתונים.

**מחלקת RegionDecodeMap**

מחלקה זו יורשת ממחלקת ClassMap של Fluent Nhibernate ומגדירה את הקשר בין מחלקת RegionDecode לטבלת REGION\_DECODE מסד הנתונים.

**מחלקת FieldSizeDecodeMap**

מחלקה זו יורשת ממחלקת ClassMap של Fluent Nhibernate ומגדירה את הקשר בין מחלקת FieldSizeDecode לטבלת FIELD\_SIZE\_DECODE מסד הנתונים.

**מחלקת FieldTypeDecodeMap**

מחלקה זו יורשת ממחלקת ClassMap של Fluent Nhibernate ומגדירה את הקשר בין מחלקת FieldTypeDecode לטבלת FIELD\_TYPE\_DECODE מסד הנתונים.

**מחלקת InvitationStatusDecodeMap**

מחלקה זו יורשת ממחלקת ClassMap של Fluent Nhibernate ומגדירה את הקשר בין מחלקת InvitationStatusDecode לטבלת INVITATION\_STATUS\_DECODE מסד הנתונים.

**מחלקת OrderStatusDecodeMap**

מחלקה זו יורשת ממחלקת ClassMap של Fluent Nhibernate ומגדירה את הקשר בין מחלקת OrderStatusDecode לטבלת ORDER\_STATUS\_DECODE מסד הנתונים.

**מחלקת ComplaintTypeDecodeMap**

מחלקה זו יורשת ממחלקת ClassMap של Fluent Nhibernate ומגדירה את הקשר בין מחלקת ComplaintTypeDecode לטבלת COMPLAINT\_TYPE\_DECODE מסד הנתונים.

**מחלקת ComplaintMap**

מחלקה זו יורשת ממחלקת ClassMap של Fluent Nhibernate ומגדירה את הקשר בין מחלקת Complaint לטבלת COMPLAINT במסד הנתונים.

**מחלקת FieldMap**

מחלקה זו יורשת ממחלקת ClassMap של Fluent Nhibernate ומגדירה את הקשר בין מחלקת Field לטבלת FIELD מסד הנתונים.

**מחלקת OrderMap**

מחלקה זו יורשת ממחלקת ClassMap של Fluent Nhibernate ומגדירה את הקשר בין מחלקת Order לטבלת ORDER מסד הנתונים.

**מחלקת ParticipantMap**

מחלקה זו יורשת ממחלקת ClassMap של Fluent Nhibernate ומגדירה את הקשר בין מחלקת Participant לטבלת PARTICIPANT מסד הנתונים.

**מחלקת ReviewMap**

מחלקה זו יורשת ממחלקת ClassMap של Fluent Nhibernate ומגדירה את הקשר בין מחלקת Review לטבלת REVIEW מסד הנתונים.

פרויקט השירותים (PlaySimple)

פרויקט זה מהווה מימוש של שכבת השירותים. כל המחלקות בפרויקט זה נועדו, בצורה ישירה או עקיפה, כדי להנגיש את הפונקציונאליות הקיימת בשכבת הלוגיקה ובשכבת הנתונים לשכבת התצוגה.

נינג'קט (Ninject IoC Container)

נינג'קט הוא פרויקט open source שנועד להזרקת אובייקטים (והתלויות שלהם). אנו נשתמש בנינג'קט בכדי לממש את עקרון היפוך התלויות (Dependency Inversion). בניגוד לגישה בה כל קונטרולר יוצר את כל האובייקטים בהם הוא תלוי, בעת יצירת קונטרולר יוזרקו אליו כל התלויות שלו באופן אוטומטי דרך Ninject בעזרת התשתית של Web API. בנוסף, בעזרת נינג'קט ניתן גם לנהל את אורך חיי הבקשה. כברירת מחדל כל אובייקט אשר מבוקש מנינג'קט נוצר מחדש עבור כל בקשה, אלא אם הוגדר אחרת. ניתן להגדיר עבור אובייקט מסויים שהוא יהיה Singleton וכך בקשות נשנות לאותו אובייקט יחזירו את אותו עצם. ניתן גם להגדיר שאובייקט מסויים יהיה Singleton ברמת הבקשה (נשתמש ביכולת זו לצורך ניהול ה-Session של Nhibernate).

**מחלקת NinjectWebCommon (תחת App\_Start)**

מחלקה זו נוצרת באופן אוטומטית עם התקנת חבילת ה-Nuget של Ninject עבור Web API. היא מכילה את הפונקציונאליות הכרוכה באתחול של Ninject ובניקוי הזכרון בעת עצירת הפרויקט. לא נפרט כאן על דרך פעולת מחלקה זו, יש לעיין באתר של Ninject לפרטים נוספים.

נעיר רק כי במחלקה זו תחת הפונקציה RegisterServices נתאחל את מחלקת NinjectConfigurator אשר תוסיף את כל האובייקטים הרלוונטים ל-IoC. בנוסף, תחת הפונקציה Start נגדיר ל-Web API להשתמש ב-Ninject IoC לצורך הזרקת אובייקטים בעזרת קביעת ה-DependencyResoler של Web API להיות אובייקט מסוג NinjectDependencyResoler.

**מחלקת NinjectDependencyResolver**

מחלקה זו מממשת את הממשק IDependencyResolver של Net. ויודעת לתשאל את ה-IoC לצורך שליפת אובייקט או רשימה של אובייקטים מממנו לפי סוג המחלקה מבוקש. המחלקה מכילה את הפונקציות הבאות:

public object GetService(Type serviceType)

public IEnumerable<object> GetServices(Type serviceType)

public IDependencyScope BeginScope()

public void Dispose()

**מחלקת NinjectConfigurator**

מחלקה זו אחראית על רישום כל האובייקטים ב-Ninject IoC בעזרת קריאה למתודה Configure. מתודה זו קוראת ל-AddBindings אשר רושמת בפועל את כל סוגי המחלקות. בין המחלקות אשר יירשמו ב-IoC יהיו ה-SessionFactory של Nhibernate - אובייקט אשר יודע לייצר עצמים מסוג Session של Nhibernate ומתודה אשר תייצר Session ייחודי עבור כל בקשה. המחלקה מכילה את הפונקציות הבאות:

public void Configure(IKernel container)

public void AddBindings(IKernel container)

Query Processors

קבוצת מחלקות זו אחראית על מימוש שכבת הלוגיקה באתר. בהתאם לעקרונות המנחים בשכבת הלוגיקה, עבור כל ישות יוגדר query processor בודד אשר יכיל את הלוגיקה הקשורה באותה ישות. עבור כל מחלקה בקבוצה זו יוגדר ממשק מתאים. ממשק זה ישומש בעת שליפת ה-query processor מה-IoC. שם כל ממשק מסוג זה יהיה כשם ה-query processor עם קידומת I (לציון Interface) ויכיל חתימות זהות למתודות ב-query processor.

**ממשק <IDatabaseAccess<T**

ממשק זה מגדיר מתודות המאפשרות גישה למסד הנתונים עבור ישות מסוג T כאשר T יורשת מ-Entity. באמצעות ממשק זה ניתן להכניס, לעדכן ולשלוף נתונים (בעזרת Linq) מבסיס הנתונים. ממשק זה מגדיר את הפעולות הבאות:

void Update(int id, T entity);

T Save(T entity);

T Get(int id);

IQueryable<T> Query();

**מחלקת <DatabaseAccessBase<T**

מחלקה מופשטת זו מממשת את <IDatabaseAccess<T בעזרת Nhibernate. כל query processor יירש ממימוש מסויים של מחלקה זו. המחלקה תממש את המתודות שלIDatabaseAccess

**מחלקת CustomersQueryProcessor**

מחלקה זו אחראית על ישות הלקוח. מחלקה זו יורשת מ<DBAccessBase<Customer ומממשת את הממשק ICustomersQueryProcessor. תבצע הכנס ועדכון של לקוחות במסד הנתונים. כמו כן, מחלקה זו גם אחראית על החזרת הלקוחות ממסד הנתונים בהתאם לפרמטרים האפשרים ממסכי המערכת השונים. המחלקה תכיל את הפעולות הבאות:

IEnumerable<DTOs.Customer> Search(string firstName, string lastName, int? minAge, int? maxAge, int? region, int? customerId);

DTOs.Customer GetCustomer(int id);

DTOs.Customer Save(DTOs.Customer customer);

DTOs.Customer Update(int id, DTOs.Customer customer);

bool Exists(string username);

**מחלקת EmployeesQueryProcessor**

מחלקה זו אחראית על ישות העובד. מחלקה זו יורשת מ<DBAccessBase<Employee ומממשת את הממשק IEmployeesQueryProcessor. תבצע הכנסה, עדכון של עובדים במסד הנתונים. כמו כן מחלקה זו גם אחראית על החזרת העובדים ממסד הנתונים בהתאם לפרמטרים האפשרים ממסך חיפוש העובדים של מנהל. המחלקה תכיל את הפעולות הבאות:

IEnumerable<DTOs.Employee> Search(string firstName, string lastName, string eMail, int? id);

DTOs.Employee GetEmployee(int id);

DTOs.Employee Save(DTOs.Employee employee);

DTOs.Employee Update(int id, DTOs.Employee employee);

void Delete(int id);

**מחלקת FieldsQueryProcessor**

מחלקה זו אחראית על ישות המגרש. מחלקה זו יורשת מ<DBAccessBase<Field ומממשת את הממשק IFieldsQueryProcessor. תבצע הכנסה, עדכון של מגרש במסד הנתונים. כמו כן, מחלקה זו גם אחראית על החזרת המגרשים ממסד הנתונים בהתאם לפרמטרים האפשרים ממסכי המערכת השונים. המחלקה תכיל את הפעולות הבאות:

IEnumerable<DTOs.Field> Search(int? pageNum, int? fieldId, string fieldName, int? typeId);

DTOs.Field GetField(int id);

DTOs.Field Save(DTOs.Field field);

DTOs.Field Update(int id, DTOs.Field field);

void Delete(int id);

**מחלקת OrdersQueryProcessor**

מחלקה זו אחראית על יישות ההזמנה. מחלקה זו יורשת מ<DBAccessBase<Order ומממשת את הממשק IOrdersQueryProcessor. תבצע הכנסה, עדכון של הזמנה במסד הנתונים. כמו כן, מחלקה זו אחראית על החזרת ההזמנות בהתאם לפרמטרים האפשרים ממסכי המערכת השונים. המחלקה תכיל את הפעולות הבאות:

List<DTOs.Order> Search(int? orderId, int? ownerId, string ownerName, int?[] orderStatusIds, int? fieldId, string fieldName, DateTime? startDate, DateTime? endDate);

DTOs.Order GetOrder(int id);

DTOs.Order Save(DTOs.Order order);

DTOs.Order Update(int id, DTOs.Order order);

List<DTOs.Order> SearchOptionalOrders(int? fieldId, string fieldName, int? fieldTypeId, DateTime date);

IEnumerable<DTOs.Order> SearchAvailableOrdersToJoin(int? ownerId, string ownerName, int? orderId, int? orderStatusId, int? fieldId, string fieldName, DateTime? startDate, DateTime? endDate);

**מחלקת ComplaintsQueryProcessor**

מחלקה זו אחראית על ישות התלונה. מחלקה זו יורשת מ<DBAccessBase<Complaint ומממשת את הממשק IComplaintsQueryProcessor. תבצע הכנסה, עדכון של תלונה במסד הנתונים. כמו כן, מחלקה זו אחראית על החזרת התלונות בהתאם לפרמטרים האפשרים ממסכי המערכת השונים. המחלקה תכיל את הפעולות הבאות:

IEnumerable<DTOs.Complaint> Search(int? customerId, DateTime? fromDate, DateTime? untilDate, int? complaintType);

DTOs.Complaint GetComplaint(int id);

DTOs.Complaint Save(DTOs.Complaint complaint);

**מחלקת ReviewsQueryProcessor**

המחלקה אחראית על הישות חוות דעת של לקוחות על לקוחות אחרים. מחלקה זו יורשת

מ <DBAccessBase<Review ומממשת את הממשק IReviewsQueryProcessor. תבצע הכנסה, עדכון של חוות דעת במסד הנתונים. כמו כן, המחלקה תבצע שליפה של הישויות ממסד הנתונים בהתאם לפרמטרים האפשרים ממסכי המערכת השונים.

IEnumerable<DTOs.Review> Search(int reviewedCustomerId);

DTOs.Review GetReview(int id);

DTOs.Review Save(DTOs.Review review);

DTOs.Review Update(int id, DTOs.Review review);

**מחלקת ParticipantsQueryProcessor**

מחלקה זו אחראית על ישות ההצטרפות להזמנה. מחלקה זו יורשת מ   
<DBAccessBase<Participants ומממשת את הממשק IParticipantsQueryProcessor. תבצע הכנסה, עדכון ושליפה של הישות במסד הנתונים בהתאם לפרמטרים האפשרים ממסכי המערכת השונים.

IEnumerable<DTOs.Participant> Search(int? customerId, int? ownerId, int?[] statusIds, string ownerName, int? orderId, int? fieldId, string fieldName, DateTime? fromDate, DateTime? untilDate);

DTOs.Participant GetParticipant(int id);

DTOs.Participant Save(DTOs.Participant participant);

DTOs.Participant Update(int id, DTOs.Participant participant);

void Delete(int id);

קונטרולרים (Controllers)

קבוצת מחלקות זו אחראית על חשיפת השירותים המוצעים באתר בממשק REST. כאשר נשלחת בקשה לשרת נבחרת הפונקציה בקונטרולר הרלוונטי בהתאם ל-HTTP Verb בהתאם לשמות וערכי הפרמטרים של הבקשה. מגנון זה מובנה בתוך Web API. כתובת כל קונטרולר היא שמו ללא הסופית Controller, כלומר הכתובת היחסית של OrdersController תהיה api/orders/, אם נשלח בקשה ל-api/orders/3 עם פועל HTTP מסוג Get הפונקציה שתופעל בשרת תהיה (Get(int id כאשר id יקבל את הערך 3. חשוב להדגיש כי במחלקות אלו לא תתבצע לוגיקה, אלא הן יכילו query processors מתאימים אשר יבצעו את הלוגיקה עבורן.

**מחלקת CustomersController**

הקונטרולר יבצע פעולות חיפוש ושינוי פרטים, חיפוש והקפאת לקוחות. המחלקה תכיל את הפעולות הבאות:

IEnumerable<DTOs.Customer> Search(string firstName, string lastName, int? minAge, int? maxAge, int? regionId, int? customerId);

DTOs.Customer Get(int id);

DTOs.Customer Save([FromBody]DTOs.Customer customer);

DTOs.Customer Update(int id, [FromBody]DTOs.Customer customer);

**מחלקת LoginController**

הקונטרולר יבצע הרשמה והתחברות למערכת, תחזיר ללקוח את נתוני הלקוח המתחבר. המחלקה תזרוק שגיאות מתאימות של הרשאות במקרה של אי הצלחה. תכיל את הפועלות הבאות:

LoginResponse Login(UserCredentials credentials)

RegistrationReponse Registration(DTOs.Customer customer)

**מחלקת EmployeesController**

הקונטרולר יבצע את הפעולות הקשורות לטיפול במשתמשים מסוג עובדים: שליפה, הוספה, שינוי ומחיקה. המחלקה תכיל את הפעולות הבאות:

public IEnumerable<DTOs.Employee> Search(string firstName, string lastName, int eMail, int id);

public DTOs.Employee Get(int id);

public DTOs.Employee Save(DTOs.Employees employees);

public DTOs.Employee Update(int id, DTOs.Employees employees);

public void Delete(int id);

**מחלקת FieldsController**

הקונטרולר יבצע את הפעולות הקשורות לטיפול במגרשים: שליפה, הוספה, שינוי ומחיקה. המחלקה תכיל את הפעולות הבאות:

IEnumerable<DTOs.Employee> Search(string firstName, string lastName, string eMail, int? id)

DTOs.Employee Get(int id)

DTOs.Employee Save([FromBody]DTOs.Employee employee)

DTOs.Employee Update([FromUri]int id, [FromBody]DTOs.Employee employee)

void Delete([FromUri]int id)

**מחלקת OrdersController**

הקונטרולר יבצע את כלל הפועלות הקשורות להזמנה ובניהן הפעולות קשורות להצטרפות שחקנים אחרים להזמנה קיימת.

List<DTOs.Order> SearchMyOrders(int? orderId, int? orderStatusId, int? fieldId, string fieldName, DateTime? fromDate, DateTime? untilDate);

List<DTOs.Order> Search(int? orderId, int? ownerId, string ownerName, int? orderStatusId, int? fieldId, string fieldName, DateTime? fromDate, DateTime? untilDate);

List<DTOs.Order> SearchAvailableOrdersToJoin(int? ownerId, string ownerName, int? orderId, int? orderStatusId, int? fieldId, string fieldName, DateTime? fromDate, DateTime? untilDate);

DTOs.Order Get(int id);

DTOs.Order Save([FromBody]DTOs.Order order);

DTOs.Order Update([FromUri]int id, [FromBody]DTOs.Order order);

IEnumerable<DTOs.Participant> SearchParticipants(int orderId, int? customerId);

List<DTOs.Order> SearchAvailableOrders(int? fieldId, string fieldName, int? fieldType, DateTime? date);

List<DTOs.Order> SearchOptionalsOrders(int? orderId, int? fieldId, int? fieldType, DateTime? date);

DTOs.Order UpdatePraticipant([FromUri]int id, [FromBody]DTOs.Participant participant);

DTOs.Participant JoinToOrder(int? orderId);

**מחלקת ReportsController**

הקונטרולר יבצע את הפקת כלל הדוחות, יחזיר לעובד או למנהל את הדוחות ע"פ הפרמטרים המתקבלים לכל דו"ח. המחלקה תכיל את הפעולות הבאות:

List<OffendingCustomersReport> GetOffendingCustomersReport(DateTime? fromDate, DateTime? untilDate, int? complaintType)

List<CustomersActivityReport> GetCustomersActivityReport(string firstName, string lastName, int? minAge, int? maxAge, DateTime? fromDate, DateTime? untilDate)

List<UsingFieldsReport> GetUsingFieldsReport(string fieldName, int? fieldId, DateTime? fromDate, DateTime? untilDate)

מחלקת ReviewsController

הקונטרולר יבצע את כלל הפועלות הקשורות להזנת חוות דעת של לקוחות על לקחות אחרים. המחלקה תכיל את הפעולות הבאות:

List<DTOs.Review> Search(int? customerId = null)

DTOs.Review Get(int id)

DTOs.Review Save([FromBody]DTOs.Review Review)

DTOs.Review Update([FromUri]int id, [FromBody]DTOs.Review Review)

מחלקת ComplaintsController

הקונטרולר יבצע את כלל הפועלות הקשורות להזנת תלונות לקוחות על לקחות אחרים. המחלקה תכיל את הפעולות הבאות:

ComplaintsController(IComplaintsQueryProcessor ComplaintsQueryProcessor);

List<DTOs.Complaint> Search(int? customerId = null);

DTOs.Complaint Get(int id);

DTOs.Complaint Save([FromBody]DTOs.Complaint complaint);

**sd**מחלקת ParticipantsController

הקונטרולר יבצע פועלות הקשורות להצטרפות של לקוחות להזמנות של לקחות אחרים. המחלקה תכיל את הפעולות הבאות:

List<DTOs.Participant> SearchParticipants(int? ownerId, string ownerName, int? orderId, int? invitationStatusId, int? fieldIdl, string fieldName, DateTime? fromDate, DateTime? untilDate);

DTOs.Participant Get(int id);

DTOs.Participant Save([FromBody]DTOs.Participant Participant);

DTOs.Participant Update([FromUri]int id, [FromBody]DTOs.Participant Participant);

void Delete([FromUri]int id);

ולידטורים (Validators)

קבוצת מחלקות זו מייצגות את האילוצים על שדות במידע המגיע מהלקוח אל השרת. בעת הגעת בקשה לשרת מופעלים הולידטורים המתאימים בהתאם לסוג הבקשה. אם נשלח לשרת מידע אשר נמצא לא תקין ע"י הולידטורים שהופעלו, הלקוח יקבל תשובת שגיאה עם Http Status 403 - המייצג בקשה לא תקינה. כל ולידטור יורש מהמחלקה ValidationAttribute של Net. ומממש את ה-Template Method בשם IsValid - מתודה זו מקבלת את ערך השדה ומחזירה אינדיקציה האם הערך תקין או לא. אינדיקציה זו נשמרת על אובייקט בקשת הלקוח לצורך שימוש בזמן מאוחר יותר.

**מחלקת AboveAttribute**

מחלקה זו יורשת מהמחלקה ValidationAttribute של Net. ודורסת את המתודה IsValid. ולידטור זה משמש לאכיפה שערך תכונה נומרית גדול מערך מינימאלי מסויים.

**מחלקת ExistsInDBAttribute**

מחלקה זו יורשת מהמחלקה ValidationAttribute של Net. ודורסת את המתודה IsValid. ולידטור זה משמש לאכיפה שערך תכונה מייצג ישות שקיימת במסד הנתונים. למשל הולידטור משמש לוידוא שמשתמש אכן קיים בעת הגשת תלונה עליו.

**מחלקת IsEnumOfTypeAttribute**

מחלקה זו יורשת מהמחלקה ValidationAttribute של Net. ודורסת את המתודה IsValid. ולידטור זה משמש לאכיפה שערך תכונה המייצג ערך פענוח שקיים במסד הנתונים. למשל הולידטור משמש לוידוא שהערך 'חיפה' אכן נמצא ברשימת פיענוח האיזורים - RegionDecode.

**מחלקת ListExistsInDbAttribute**

מחלקה זו יורשת מהמחלקה ValidationAttribute של Net. ודורסת את המתודה IsValid. ולידטור זה משמש לאכיפה שערך תכונה מייצג רשימה של ישויות שקיימות במסד הנתונים.

**מחלקת NotInFutureAttribute**

מחלקה זו יורשת מהמחלקה ValidationAttribute של Net. ודורסת את המתודה IsValid. ולידטור זה משמש לאכיפה שערך תכונה מסוג תאריך אינו בזמן העתיד.

**מחלקת NotInPastAttribute**

מחלקה זו יורשת מהמחלקה ValidationAttribute של Net. ודורסת את המתודה IsValid. ולידטור זה משמש לאכיפה שערך תכונה מסוג תאריך אינו בזמן העבר.

פילטרים (Filters)

קבוצת מחלקות זו אחראית על הלוגיקה אשר צריכה להתבצע לפני עיבוד כל בקשה בצד השרת. תפקיד כל פילטר שונה בתכלית משאר הפילטרים. פילטר יכול להיות מוגדר כך שיפעל לפני כל בקשה לשרת באופן גלובלי, עבור קונטרולר ספציפי (ובכך יופעל לפני כל מתודה השייכת לקונטרולר) או עבור פונקציה ספציפית. כל פילטר יורש מהמחלקה ActionFilterAttribute של Net. ויכול לממש שתי Template Methods:

* המתודה OnActionExecuting - קוד אשר יתבצע לפני הפעולה בקונטרולר הרלוונטי.
* המתודה OnActionExecuted - קוד אשר יתבצע לאחר הפעולה בקונטורלר הרלוונטי.

**מחלקת GlobalAuthorizationFilter**

פילטר זה מופעל לפני כל בקשה לשרת ומקבל את פרטי המשתמש (שם וסיסמה), ומאמתת אותם מול מסד הנתונים. אם הנתונים אומתו בהצלחה ישמרו הרשאות המשתמש (משתמש, עובד, מנהל) על הבקשה לצורך וידוא ההרשאות בקונטרולר. אם אימות הנתונים נכשל, תוחזר תשובת שגיאה עם Http Status 401 - המייצג שגיאת הרשאות.

**מחלקת ValidateModelAttribute**

פילטר זה מופעל לפני כל בקשה לשרת ומוודא אם לפחות אחד מהולידטורים מצא נתונים לא תקין בבקשה. אם נמצאו נתונים לא תקינים תוחזר תשובה שגיאה עם Http Status 403 - המייצג בקשה לא תקינה.

**מחלקת TransactionFilterAttribute**

פילטר זה מופעל לפני כל בקשה לשרת ואחראי על יצירת Session של Nhibernate. בנוסף, פילטר זה מופעל לאחר סיום כל בקשה בשרת ואחראי על שמירת הנתונים במסד הנתונים במידה והבקשה הסתיימה בהצלחה (Commit) או גלגול הנתונים לאחור במידה ואירעה שגיאה (Rollback).

מחלקות מעבר (Dtos)

קבוצת מחלקות זו מהווה ייצוג של הישויות אשר נשלחות מהלקוח לשכבת השירותים ובחזרה. אובייקטי המעבר נועדו לצורך יצירת הבדל ברור בין ישויות מערכת ויצוגן עבור הלקוח. להפרדה זו מספר יתרונות:

* מניעת שליחת נתונים זדוניים לשרת (Over posting). בעיה זו מתעוררת למשל, כאשר יש צורך לחשוף שירות המאפשר עדכון ישות מסויימת. אם חתימת הפונקציה תכיל את הישות עצמה הממופה למסד הנתונים, משתמש זדוני יוכל לשלוח לעדכן מאפיין שאינו אמור לעדכן ובכך לפגום במידע אחר במסד הנתונים. כאשר מוגדר Dto שדותיו הם שדות אשר אמורים להישלח באופן מפורש מהלקוח אל השרת ולהיפך.
* הקטנת תעבורת הרשת. המידע אשר מוחזר ללקוח ינורמל וייצג באופן מדוייק את השדות הרלוונטים לתצוגה בלבד.

כל מחלקה בקבוצה זו אחראית על הגדרת תקינות כל מאפייניה בעזרת ולידטורים. מעל כל מאפיין יופיעו הולידטורים המגדירים את תקינות המידע, כך למשל מחרוזת המייצגת סיסמה חייבת להיות בעלת שמונה תווים לפחות כאשר כל תו הינו מספר או אות באנגלית. יש לציין שהולידטורים הרלוונטים מופעלים באופן אוטומטי בעת הגעת כל בקשה לשרת כחלק מהתשתית של Web API.

**מחלקת Entity**

מחלקה מופשטת זו מהווה בסיס לכל מחלקות המעבר האחרות, פרט למחלקות המייצגות דוחות אשר אינן מבוססות על ישות ממסד הנתונים. למחלקה זו שני משתני טיפוס גנריים המגדירים את סוג מחלקת המעבר וסוג הישות ה-Domain שמחלקת המעבר מקושרת אליה. במחלקה זו מוגדר המזהה אשר יוכל בכל Dto המייצג ישות במערכת. בנוסף, מוגדרת חתימה מופשטת של פונקציה מסוג Factory Method לצורך אתחול כל שדות האובייקט לפי שדות הישות מסד הנתונים המתאימה. המחלקה בעלת התכונות הבאות:

public int Id { get; set; }

public abstract TDTO Initialize(TDomain domain);

**מחלקת Complaint**

מחלקה זו יורשת ממחלקת Entity ומייצגת אובייקט מעבר של תלונה. המחלקה מממשת את המתודה המופשטת Initialize של Entity והיא בעלת התכונות הבאות:

public virtual string Description { get; set; }

public virtual int Type { get; set; }

public virtual Customer OffendingCustomer { get; set; }

public virtual Customer OffendedCustomer { get; set; }

**מחלקת Employee**

מחלקה זו יורשת ממחלקת Entity ומייצגת אובייקט מעבר של עובד. המחלקה מממשת את המתודה המופשטת Initialize של Entity והיא בעלת התכונות הבאות:

public virtual string Username { get; set; }

public virtual string Password { get; set; }

public virtual string FirstName { get; set; }

public virtual string LastName { get; set; }

public virtual int Salary { get; set; }

public virtual string Email { get; set; }

**מחלקת Field**

מחלקה זו יורשת ממחלקת Entity ומייצגת אובייקט מעבר של מגרש. המחלקה מממשת את המתודה המופשטת Initialize של Entity והיא בעלת התכונות הבאות:

public virtual int Name { get; set; }

public virtual int Type { get; set; }

public virtual int Size { get; set; }

**מחלקת Order**

מחלקה זו יורשת ממחלקת Entity ומייצגת אובייקט מעבר של הזמנה. המחלקה מממשת את המתודה המופשטת Initialize של Entity והיא בעלת התכונות הבאות:

public virtual long StartDate { get; set; }

public virtual Customer Owner { get; set; }

public virtual int PlayersNumber { get; set; }

public virtual int Status { get; set; }

public virtual Field Field { get; set; }

public virtual IList<int> Participants { get; set; }

**מחלקת Participant**

מחלקה זו יורשת ממחלקת Entity ומייצגת אובייקט מעבר של בקשת הצטרפות להזמנה קיימת. המחלקה מממשת את המתודה המופשטת Initialize של Entity והיא בעלת התכונות הבאות:

public virtual Customer Customer { get; set; }

public virtual Order Order { get; set; }

public virtual DateTime Date { get; set; }

public virtual int Status { get; set; }

**מחלקת Review**

מחלקה זו יורשת ממחלקת Entity ומייצגת אובייקט מעבר של ביקורת על משתמש. המחלקה מממשת את המתודה המופשטת Initialize של Entity והיא בעלת התכונות הבאות:

public virtual string Title { get; set; }

public virtual string Description { get; set; }

public virtual DateTime Date { get; set; }

public virtual Customer Reviewer { get; set; }

public virtual Customer ReviewedCustomer { get; set; }

**מחלקת Customer**

מחלקה זו יורשת ממחלקת Entity ומייצגת אובייקט מעבר של לקוח. המחלקה מממשת את המתודה המופשטת Initialize של Entity והיא בעלת התכונות הבאות:

public virtual string Username { get; set; }

public virtual string Password { get; set; }

public virtual string FirstName { get; set; }

public virtual string LastName { get; set; }

public virtual DateTime BirthDate { get; set; }

public virtual string Email { get; set; }

public virtual int Region { get; set; }

**מחלקת UsingFieldsReport**

מחלקה מייצגת אובייקט מעבר של דוח המופק על מידת השימושיות של מרגשים ע"פ שעות וע"פ ימים. האובייקט אינו מבוסס ישות ממסד הנתונים, ולכן אינו יורש מ- Entity. המחלקה בעלת התכונות הבאות:

public virtual int FieldId { get; set; }

public virtual string FieldName { get; set; }

public virtual int WeekDayOrders { get; set; }

public virtual int WeekEndOrders { get; set; }

public virtual int hours16\_18Orders { get; set; }

public virtual int hours18\_20Orders { get; set; }

public virtual int hours20\_22Orders { get; set; }

**מחלקת CustomersActivityReport**

מחלקה מייצגת אובייקט מעבר של דוח המופק על פעילות משתמשים. האובייקט אינו מבוסס ישות ממסד הנתונים, ולכן אינו יורש מ- Entity. המחלקה בעלת התכונות הבאות:

public int CoustomerId { get; set; }

public virtual string FirstName { get; set; }

public virtual string LastName { get; set; }

public virtual int Age { get; set; }

public virtual long LastGameDate { get; set; }

public virtual int NumberOfOrders { get; set; }

public virtual int NumberOfCanceledOrders { get; set; }

public virtual int NumberOfJoiningAsGuest { get; set; }

**מחלקת OffendingCustomersReport**

מחלקה מייצגת אובייקט מעבר של דוח המופק על כמות התלונות על לקוח שמסויים. האובייקט אינו מבוסס ישות ממסד הנתונים, ולכן אינו יורש מ- Entity. המחלקה בעלת התכונות הבאות:

public int CoustomerId { get; set; }

public virtual string FirstName { get; set; }

public virtual string LastName { get; set; }

public virtual int NumberOfComplaints{ get; set; }

**מחלקת LoginResponse**

מחלקה מייצגת אובייקט מעבר של היוזר המחובר למערכת והרשאותיו המתאימות. האובייקט אינו מבוסס ישות ממסד הנתונים, ולכן אינו יורש מ- Entity. המחלקה בעלת התכונות הבאות:

public int UserId { get; set; }

public bool IsUserFrozen { get; set; }

public string Role { get; set; }

public string AuthorizationKey { get; set; }

**מחלקת RegistrationReponse**

מחלקה מייצגת אובייקט מעבר של בדיקה בעת הרשמה האם ה- username אותו הנרשם רוצה קיים כבר במערכת. האובייקט אינו מבוסס ישות ממסד הנתונים, ולכן אינו יורש מ- Entity. המחלקה בעלת התכונות הבאות:

public bool AlreadyExists { get; set; }

**מחלקת UserCredentials**

מחלקה מייצגת אובייקט מעבר של שם וסיסמת היוזר המתחבר למערכת. האובייקט אינו מבוסס ישות ממסד הנתונים, ולכן אינו יורש מ- Entity. המחלקה בעלת התכונות הבאות:

public string Username { get; set; }

public string Password { get; set; }

מחלקות תשתית (Common)

**מחלקת Consts**

מחלקה סטאטית זו מכילה קבועים שונים אשר רלוונטים לכל המערכת. המחלקה בעלת התכונות ותת המחלקות הבאות:

public const string DB\_PATH = "~/App\_Data/db.sqlite";

public static class Roles

{

public const string Customer = "Customer";

public const string Employee = "Employee";

public const string Admin = "Admin";

}

public static class Decodes

{

public enum OrderStatus

{

Sent=1,

Accepted,

Rejected,

Canceled

}

public enum ComplaintType

{

Payment=1,

Attendance,

Sportsmanship

}

public enum FieldSize

{

Small = 1,

Medium,

Large

}

public enum FieldType

{

Football=1,

Basketball,

Tennis

}

public enum InvitationStatus

{

Sent=1,

Accepted,

Rejected

}

public enum RegionDecode

{

Dan=1,

Negev,

Haifa,

Jerusalem

}

}

**מחלקת GlobalExceptionHandler**

מחלקה זו אחראית על החזרת הודעות שגיאה ללקוח בעקבות שגיאות אשר אירעו בשרת. המחלקה יורשת מהמחלקה ExceptionHandler של Net. ודורסת את המתודה Handle. כאשר תזרק שגיאה אשר לא תתפס בשרת, מתודה זו תקרא באופן אוטומטי ע"י התשתית של Web API.

**מחלקת GlobalExceptionLogger**

מחלקה זו אחראית על תיעוד שגיאות אשר אירעו בשרת. המחלקה יורשת מהמחלקה ExceptionLogger של Net. ודורסת את המתודה Log. כאשר תזרק שגיאה אשר לא תתפס בשרת, מתודה זו תקרא באופן אוטומטי ע"י התשתית של Web API. תיעוד השגיאות יתבצע בעזרת רישום לקובץ.

**מחלקת WebApiConfig**

מחלקה סטאטית זו מכילה פונקציה בודדת בשם Register אשר אחראית על הגדרת המערכת עם עלייתה. בפונקציה זו מוגדרים:

* הפילטרים אשר יופעלו באופן גלובלי עם הגעת כל בקשה לשרת.
* המחלקות אשר יטפלו בשגיאות, לצורך תיעוד ולצורך החזרת הודעה מתאימה ללקוח.
* המוסכמה הממפה בין URL-ים שונים לקונטרולרים.

**מחלקת global.asax**

מחלקה זו מכילה מתודה אחת בשם Application\_Start ומהווה נקודת התחלה לאתר (בדומה לפונקצית Main). הפונקציה זו קוראת (באופן עקיף) למתודה הסטאטית WebApiConfig.Register.

**Playsinmpledatecpnverter**

משמש לסריאליזציה של אובייקטים על ידי json.net עבור אובייקטים הנשלחים על גוף בקשות http ובשביל לשלוח ללקוח בתשובת http תאריכים בפורמט dd mm yyyy.

**Datetineparameters binding**

משמש להמרת מחרוזות בפורמט dd mm yyyy הנשלחות מהלקוח על הquerystring לאובייקט מסוג datetime בצד השרת בפונקציות בקונטרולר השונות.

אורך חיי בקשה

בסעיף זה נתאר את השלבים שבקשה עוברת מהרגע שהיא מתקבלת בשרת ועד שחוזרת תשובה ללקוח:

1. הבקשה מתקבלת בשרת.
2. נבחר הקונטרולר והמתודה שיש להפעיל בהתאם ל-URL של הבקשה והפרמטרים שלה. בנוסף, Ninject מזריק את התלויות של הקונטרולר.
3. הבקשה עוברת דרך הפילטרים הבאים:
   1. הרשאות - נקבעת רמת ההרשאה של הבקשה (משתמש / עובד / מנהל). במידה ולא נמצאה הרשאה מתאימה, מוחזרת שגיאת הרשאות ללקוח.
   2. ולידציה - נבדקת תקינות פרמטרי הבקשה. במידה והפרמטרים נמצאו לא תקינים, מוחזרת שגיאה ללקוח.
   3. חיבור למסד נתונים - נפתח חיבור למסד נתונים.
4. המתודה שנבחרה בקונטרולר מופעלת ומתבצעת הלוגיקה העסקית הרלוונטית (בעזרת query processor). במידה וקרתה שגיאה מוחזרת הודעה ללקוח.
5. לאחר סיום הבקשה, במידה והשתנו נתונים, שינויים אלו מועברים לבסיס הנתונים (בעזרת הפילטר של מסד הנתונים). במידה והייתה שגיאה מתבצע ביטול השינויים.

פרוייקט האתר (PlaySimpleSite)

פרוייקט זה מהווה את שכבת התצוגה במערכת. בעזרתו, ניתן לשלוח בקשות REST לשכבת השירותים לצורך עדכון וצפייה בנתונים. העיצוב ממשק המשתמש יעשה בעזרת Bootstrap, שהינה ספריה פופולארית לעיצוב אתרי אינטרנט.

בנוסף, בפרויקט יעשה שימוש ב-AngularJS, שהינו framework לצורך פיתוח Single Page Applications. כל פעולה אינטראקטיבית באתר אשר תכתב ב-javascript תעשה שימוש ב-AngularJS. כך למשל מנגון המעבר בין דפים (Routing), מנגון שליחת בקשות http והעברת נתונים דו-כיוונית בין המודל של כל דף והתצוגה שלו ימומשו בעזרת framework זה. AngularJS גם תומך ב-Dependency Injection באופן מובנה. כך ניתן להזריק שירותים מסויימים לחלקי המערכת השונים.

מסכים

**index.html - המסך הראשי**

זהו למעשה המסך היחיד המווה דף html שלם ומייצג את הדף הראשי במערכת. מסך זה יכיל את הגדרות שמשותפות לכל שאר הדפים (קבצי עיצוב של css, כותרת האתר וכו'). דף זה יכיל איזור אשר אליו יוזרקו התבניות של כל שאר הדפים במערכת ואת תפריט הקישורים בין דפי המערכת. הקישורים אשר יוצגו יהיו בהתאם להרשאות המשתמש.

עיצוב זה מאפשר גמישות מירבית עבור שינויים עתידיים. כל שינוי אשר יעשה במסך הראשי יבוצע אוטומטית בשאר דפי המערכת. יתר על כך, עיצוב זה מאפשר חוויית משתמש נעימה יותר, מפני שבמעבר בין דפים לא מתבצעת טעינה מחדש של כל הדף, אלא רק חלק ממנו.

**loginForm.html**

תבנית זו מייצגת את מסך ההתחברות למערכת. זהו המסך הראשון המוצג בעת הכניסה לאתר. באמצעותו ניתן לגלוש לשאר חלקי האתר באמצעות שם משתמש וסיסמה. לאחר הזנת פרטי המשתמש נקבעות ההרשאותיו והדפים אליהם הוא יכול לגלוש בהתאם. במידה וחשבון הלקוח אשר מנסה להתחבר הוקפא, הוא לא יוכל להתחבר ויקבל הודעה מתאימה.

**registrationForm.html**

תבנית זו מייצגת את מסך הרשמה. מסך זה מיועד ללקוחות חדשים אשר לא נרשמו לאתר. באמצעותו ניתן למלא את כל הפרטים הקשורים לנתוני המשתמש ולבצע הרשמה לאתר. לאחר ההרשמה המשתמש יועבר לדף הראשי.

**searchAvailableOrders.html**

תבנית זו מייצגת את מסך חיפוש מגרשים פנויים. מסך זה מיועד ללקוחות המעוניינים להזמין מגרש לשעות מסויימות. בחלק העליון במסך זה ניתן לחפש מגרשים פנויים לפי שם ומספר מגרש, לפי טווח תאריכים וסוג המשחק המבוקש. תוצאות החיפוש מוצגות כטבלה בחלק התחתון של המסך. בכל שורה בטבלה ניתן ללחוץ על הקישור 'הזמן' לצורך יצירת הזמנה חדשה במגרש זה.

**searchOrders.html**

תבנית זו מייצגת את מסך צפייה ואישור בהזמנות. מסך זה מיועד לעובדים לצורך חיפוש הזמנות של לקוחות. בחלק העליון במסך זה ניתן לחפש הזמנות לפי מספר וסטטוס הזמנה, לפי מספר ושם מגרש, לפי טווח תאריכים ולפי מזהה לקוח. תוצאות החיפוש מוצגות כטבלה בחלק התחתון של המסך. בכל שורה בטבלה ניתן ללחוץ על הקישור 'צפה' לצורך מעבר לדף פרטי ההזמנה', או 'אשר' או 'דחה' באופן ישיר. המסך יעלה באופן דיפולטי עם ההזמנות המחכות לאישור.

**editOrder.html**

תבנית זו מייצגת את מסך פרטי הזמנה. מסך זה מיועד ללקוחות לצורך יצירה, עריכה וצפייה בהזמנה. פרטי ההזמנה מופעים בחלק העליון של המסך. בחלק התחתון של המסך, בצד ימין, מופיעים המשתתפים הנוכחים בהזמנה, ובצד שמאל מופיעות כל הבקשות של לקוחות שעוד לא אושרו או נדחו.

**ownedOrders.html**

תבנית זו מייצגת את מסך ההזמנות של לקוח. מסך זה מיועד לעקב אחר פרטי ההזמנות. ניתן לסנן את ההזמנות בעזרת השדות בחלק העליון של המסך. ההזמנות מוצגות כטבלה בחלק התחתון של המסך. בכל שורה בטבלה ניתן ללחוץ על הקישור 'צפה' לצורך מעבר למסך פרטי ההזמנה.

**searchAvailableOrdersToJoin.html**

תבנית זו מייצגת את מסך הצטרפות למגרש. מסך זה מיועד ללקוחות לצורך חיפוש הזמנות אליהן ניתן להצטרף. בחלק העליון של המסך ניתן לחפש הזמנות לפי מזהה ושם הלקוח המזמין, לפי מספר וסטאטוס ההזמנה, לפי מספר ושם המגרש וטווח תאריכים. בחלק התחתון של המסך ניתן לראות הזמנות שאליהן ניתן להצטרף כטבלה. בכל שורה בטבלה ניתן ללחוץ על הקישור 'הצטרף' לצורך שליחת בקשת הצטרפות להזמנה.

**pendingOrdersToJoin.html**

תבנית זו מייצגת את מסך בקשות הצטרפות. מסך זה מיועד ללקוח לצורך מעקב אחר כל בקשות ההצטרפות להזמנות אשר הלקוח שלח. בחלק העליון ניתן לסנן את הבקשות לפי מספר ושם מזמין, לפי מספר וסטטוס הזמנה, לפי שם ומספר מגרש ולפי טווח תאריכים. בחלק התחתון של המסך ניתן לראות את כל בקשות ההצטרפות כטבלה. בכל שורה בטבלה ניתן ללחוץ על הקישור 'בטל' לצורך ביטול הבקשה.

**searchCustomers.html**

תבנית זו מייצגת את מסך חיפוש לקוחות. מסך זה מיועד לכלל המשתמשים באתר והוא מאפשר חיפוש של לקוחות באתר. בחלק הלעיון ניתן לחפש לקוחות לפי שם פרטי ומשפחה, לפי טווח גילאים, איזור מגורים ומזהה לקוח. בחלק התחתון תוצאות החיפוש מוצגות בטבלה. בכל שורה בטבלה ניתן ללחוץ על הקישור 'עבור לדף הלקוח' לצורך מעבר לדף פרטי הלקוח.

**editCustomer.html**

תבנית זו מייצגת את מסך פרטי לקוח. מסך זה מיועד ללקוחות ומנהלים. לקוח יכול לערוך את פרטיו. עבור לקוחות אחרים, יוצגו פרטי הלקוח ללא אפשרות עריכה, והם יכלו לכתוב ולקרוא חוות דעת על הלקוח או להתלונן על הלקוח עקב סיבה מסויימת. התלונות וחוות הדעת יפתחו בחלון מודאלי. מנהל יראה את פרטי הלקוח ויוכל להקפיא את החשבון המשתמש לפרק זמן קבוע מראש.

**reportFields.html**

תבנית זו מייצגת את מסך דוח מגרשים. מסך זה מיועד לעובדים ומנהלים. בחלק העליון של מסך זה ניתן לחפש מגרשים לפי מזהה ושם מגרש וטווח תאריכים. בחלק התחתון של המסך יוצג הדוח כטבלה ובה כמות ההזמנות במגרשים לפי שעות הפעילות.

**reportCustomer.html**

תבנית זו מייצגת את מסך דוח פעילות לקוח. מסך זה מיועד לעובדים ומנהלים. בחלק העליון של המסך ניתן לחפש לקוחות לפי שם פרטי ושם משפחה, לפי טווח גילאים וטווח תאריכים. בחלק התחתון של המסך יוצג הדוח כטבלה ובה כמות ההזמנטת שמשתמש יזם, ביטל ואליהן הוא הצטרף.

**reportComplaints.html**

תבנית זו מייצגת את מסך דוח תלונות. מסך זה מיועד למנהלים בלבד לצורך צפייה בתלונות. בחלק העליון של המסך ניתן לחפש תלונות לפי סוג וטווח תאריכים. בחלק התחתון של המסך יוצגו דוח הלקוחות אשר עליהם התלוננו כטבלה. בכל שורה ניתן ללחוץ על הקישור 'צפה בתלונות' בשביל לפתוח חלון מודאלי עם כל התלונות אשר הוגשו על הלקוח.

**searchEmployee.html**

תבנית זו מייצגת את מסך חיפוש עובדים. מסך זה מיועד למנהלים. בחלק העליון ניתן לחפש עובדים לפי שם פרטי ושם משפחה, כתובת דואר אלקטרוני ומזהה. בחלק התחתון מוצגות תוצאות החיפוש כטבלה. בכל שורה ניתן ללחוץ על הקישור 'עבור לדף העובד' לצורך מעבר לדף פרטי העובד.

**editEmployee.html**

תבנית זו מייצגת את מסך פרטי עובד. מסך זה מיועד למנהלים לצורך צפייה, יצירה ועדכון פרטי עובדים.

**editField.html**

תבנית זו מייצגת את מסך פרטי מגרש. מסך זה מיועד לעובדים לצורך צפייה, יצירה ועדכון פרטי מגרש.

קבצי עיצוב

**bootstrap.css**

קובץ עיצוב זה מכיל את העיצוב המינימאלי הדרוש עבור bootstrap.

**bootstrap-theme.css**

קובץ עיצוב זה מכיל הגדרות עיצוב מתקדמות של bootstrap. בקובץ זה ניתן למצוא למשל אנימציות וגרדיאנטים.

**app.css**

קובץ עיצוב זה מכיל את כל הגדרות העיצוב שרלוונטיות אך ורק למערכת. בקובץ זה יתאפשר לדרוס חוקי עיצוב של bootstrap במידה והם לא יתאימו לעיצוב של המערכת.

קבצי javascript

**app.js**

קובץ זה אחראי על אתחול Angular וקישור כל נתיב באתר (URL) לתבנית html וקונטרולר של angular בצורה חד-חד ערכית.

**services.js**

שירות זה יכיל אובייקט עם הגדרות רלוונטית לכלל המערכת. שירות זה יחשוף בין היתר את הפרטים המזהים של הלקוח ורמת ההרשאות שלו.

**filters.js**

קובץ זה מכיל אפשרויות תצוגה מתאימות לצורכי המערכת של תאריכים ומועדי הזמנות המבוססות על פטפורמת Angular.

**directives.js**

קובץ זה מכיל מאפייני תגי html יעודיים למערכת על מנת לאפשר התנהגויות רצויות של ה- UI.

**loginControllers.js**

קובץ זה יחשוף מתודות אשר יישלחו לקונטרולר של הרשמה והתחברות בשכבת השירותים את הבקשות הבאות בהתאם לחתימת המתודות בשכבת השירותים:

* פרטי משתמש אם קיימים ומידע לגבי הרשאותיו.
* הנכסת לקוח חדש למערכת.

לשירות זה יוזרק ע"י Angular שירות בקשות ה-http.

**employeeControllers.js**

קובץ זה יחשוף מתודות אשר יישלחו לקונטרולר של העובדים בשכבת השירותים את הבקשות הבאות בהתאם לחתימת המתודות בשכבת השירותים:

* חיפוש עובדים לפי שם פרטי, שם משפחה, כתובת אימייל ומזהה עובד.
* שליפת עובד לפי מזהה עובד.
* שמירה של עובד חדש לפי פרטיו המזהים.
* עדכון פרטי עובד קיים לפי מזהה עובד ופרטיו.
* מחיקת עובד לפי מזהה עובד.

לשירות זה יוזרק ע"י Angular שירות בקשות ה-http.

**fieldControllers.js**

קובץ זה יחשוף מתודות אשר יישלחו לקונטרולר של המגרשים בשכבת השירותים את הבקשות הבאות בהתאם לחתימת המתודות בשכבת השירותים:

* חיפוש מגרשים לפי מזהה מגרש, שם מגרש, טווח תאריכים וסוג מגרש.
* שליפת מגרש לפי מזהה מגרש.
* שמירה של מגרש חדש לפי פרטיו המזהים.
* עדכון פרטי מגרש קיים לפי מזהה מגרש ופרטיו.
* מחיקת מגרש לפי מזהה מגרש.

לשירות זה יוזרק ע"י Angular שירות בקשות ה-http.

**ordersControllers.js**

קובץ זה יחשוף מתודות אשר יישלחו לקונטרולר של ההזמנות בשכבת השירותים את הבקשות הבאות בהתאם לחתימת המתודות בשכבת השירותים:

* חיפוש הזמנה לפי מזהה, סטטוס, שם וטווח תאריכים.
* שליפת הזמנה לפי מזהה הזמנה.
* שמירה של הזמנה חדשה לפי פרטיה המזהים.
* עדכון פרטי הזמנה קיימת לפי מזהה הזמנה ופרטיה.

לשירות זה יוזרק ע"י Angular שירות בקשות ה-http.

**reportControllers.js**

קובץ זה יחשוף מתודות אשר יישלחו לקונטרולר של הדוחות בשכבת השירותים את הבקשות הבאות בהתאם לחתימת המתודות בשכבת השירותים:

* הפקת דוח תלונות על משתמשים לפי טווח תאריכים וסוג תלונה
* הפקת דוח פעילות משתמשים לפי מזהה מגרש, שם מגרש וטווח תאריכים.
* הפקת דוח שימושיות מגרש לפי סוג תלונה וטווח תאריכים.

לשירות זה יוזרק ע"י Angular שירות בקשות ה-http.

**customerControllers.js**

קובץ זה יחשוף מתודות אשר יישלחו לקונטרולר של הלקוחות בשכבת השירותים את הבקשות הבאות בהתאם לחתימת המתודות בשכבת השירותים:

* חיפוש לקוחות לפי שם פרטי, שם משפחה, טווח גילאים, איזור מגורים ומזהה לקוח.
* שליפת לקוח לפי מזהה לקוח.
* שמירה של לקוח חדש לפי פרטיו המזהים.
* עדכון פרטי לקוח קיים לפי מזהה לקוח ופרטיו.
* הוספת תלונה על לקוח לפי פרטי התלונה ומזהי הלקוחות.
* הקפאת חשבון לקוח לפי מזהה.

לשירות זה יוזרק ע"י Angular שירות בקשות ה-http.

דפוסי עיצוב

אלו הם דפוסי העיצוב בהם ייעשה שימוש בפרויקט:

Singleton

דפוס עיצוב זה מגדיר שעבור מחלקה מסויימת יווצר לכל היותר מופע אחד. דפוס עיצוב זה שכיח מאוד ותפקידו למנוע יצירה של מספר אובייקטים עבור ממחלקה שעבורה נדרש מופע אחד לכל היותר.

בפרויקט יצירה של SessionFactory של Nhibernate היא פעולה יחסית יקרה. לכן, יש ליצור אובייקט זה פעם אחת בלבד עם עליית המערכת ולשמור הפנייה אליו לאורך כל חיי המערכת. אובייקט זה נשמר ב-IoC עם אורך חיים מסוג Constant, כלומר בכל פעם שיבקשו אובייקט מסוג זה מה-IoC יוחזר אותו מופע ובכך ימומש באופן עקיף דפוס העיצוב Singleton.

Factory Method

דפוס עיצוב זה מגדיר דרך אחידה ליצירת מופעים של מחלקה מסויימת ונותן למחלקות היורשות להחליט איזה סוג של עצם ליצור. דפוס עיצוב זה שכיח מאוד ותפקידו לייצר אובייקטים בצורה עקיפה ללא תלות במימוש ספציפי.

בפרויקט לחלק ניכר של המחלקות קיימת מחלקה מקבילה מסוג Dto המייצגת את אותו אובייקט, אך עם שדות מעט שונים המתאימים לתצוגה בלקוח. במחלקת הבסיס של כל מחלקות ה-Dto קיימת מתודה מופשטת בשם Initialize המקבלת אובייקט Domain. בכל המחלקות היורשות מתודה זו מאתחלת את כל השדות של ה-Dto בהתאם לאובייקט ה-Domain שנשלח אליה. לפרטים נוספים ניתן לעיין בדיאגרמת המחלקות של מחלקות ה-Dto.

Decorator

מימוש דפוס עיצוב זה מגדיר דרך להוספת פונקציונאליות לעצמים ללא צורך ביצירת מחלקות ייעודיות. דפוס עיצוב זה מעלה את המודולאריות של המערכת בכך שהוא מחלק אחראיות בין מספר אובייקטים.

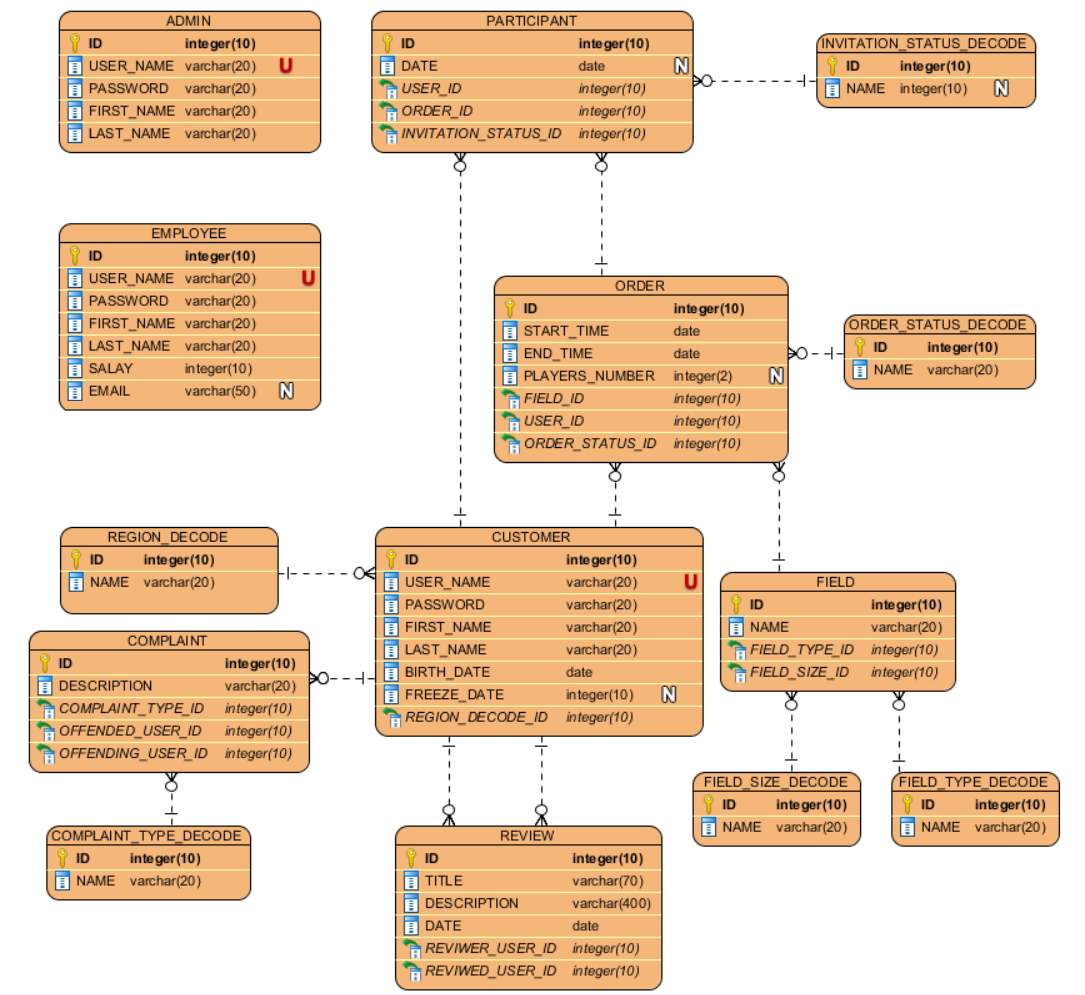
בפרויקט הוגדרו מספר ולידטורים אשר מוודאים תקינות של שדות Dto שנשלח לשרת. לכל שדה ניתן להוסיף כל מספר של ולידטורים והתשתית של Web API תדע להפעיל את כולם אחד אחר השני. כך למשל מחרוזת המייצגת סיסמה יכולה להיות בעלת שני ולידטורים, אחד המוודא שהמחרוזת בעלת 8 תווים לפחות וולידטור המוודא שהסיסמה מכילה רק אותיות לועזיות ומספרים. נעיר כי באופן דומה גם הפילטרים שמומשו במערכת ומופעלים ע"י התשתית של Web API מהווים סוג של Decorator.

שינויים עתידיים

1. שיפור אבטחת המידע באתר. במימוש הנוכחי זהות המשתמש נשמרת ב-Cookie שאינו מוצפן. בצורה זו ניתן בקלות יחסית להתחזות למשתמשים אחרים. בעתיד יש לייצר פרוטוקל אבטחה בטוח יותר, למשל בעזרת שימוש ב-Cookies המוצפנים של Net.
2. אפשרות לבטל הצטרפות של לקוח להזמנה של לקוח אחר, בתלות בפרק זמן מסויים לפני תחילת המשחק או בכלל. המצב כיום הוא כאשר לקוח מאשר הצטרפות של לקוח אחר להזמנה שלו אין לשני הצדדים אפשרות לבטל את ההצטרפות.
3. אפשרות לתשלום באמצעות כרטיסי אשראי, המצב כיום הוא שלמערכת אין תמיכה בביצוע תשלום מכוון. במצב הנוכחי, התשלום צריך להתבצע טלפונית או במעמד ההזמנה עצמה. השאיפה היא לאפשר תשלום באמצעות סליקה דרך האתר.
4. הצעת מיזוג קבוצות משחק כאשר אין מספיק משתתפים. המערכת תזהה הזמנות חופפות או קרובות אחת לשניה אשר מספר המשתתפים הנוכחי בה קטן ממספר השחקנים המבוקש, ותציע למזמינים אפשרות לאחד את ההזמנות.
5. יכולת ביטול ההזמנה ע"י העובד לאחר אישורה, המצב כיום הוא שלאחר שהזמנת מגרש אושרה, לעובד אין אפשרות לבטל אותה אלא רק ללקוח עצמו מסוגל לעשות זאת.

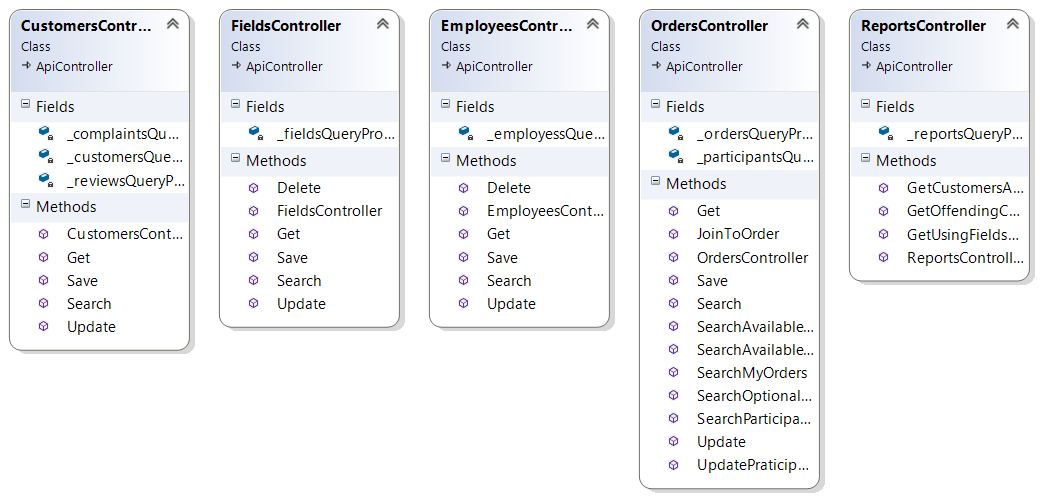
דיאגרמות

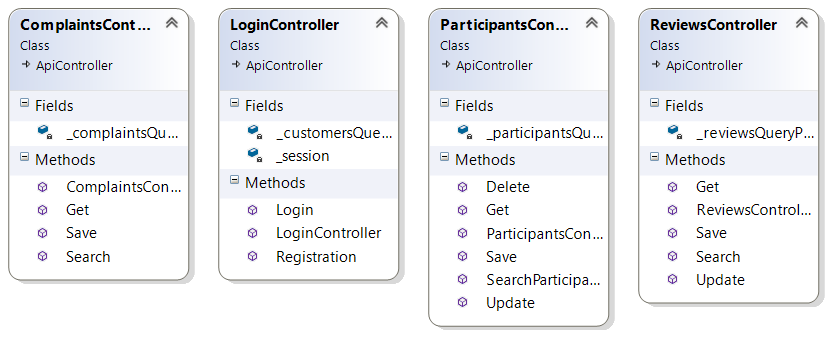
מודל ישויות קשרים (ERD)



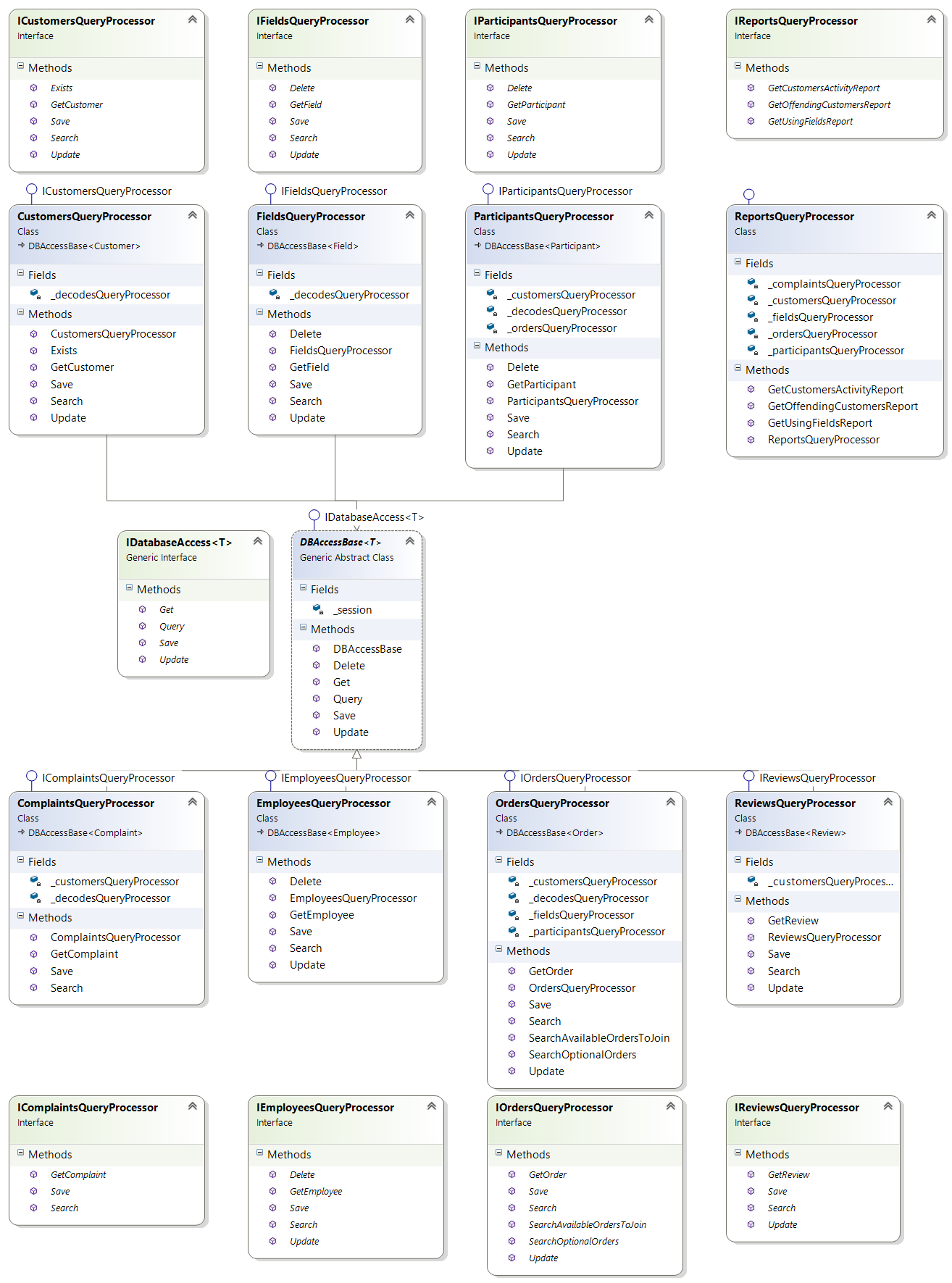
דיאגרמות מחלקות (Class Diagram)

**דיאגרמת מחלקות עבור הקונטרולרים:**

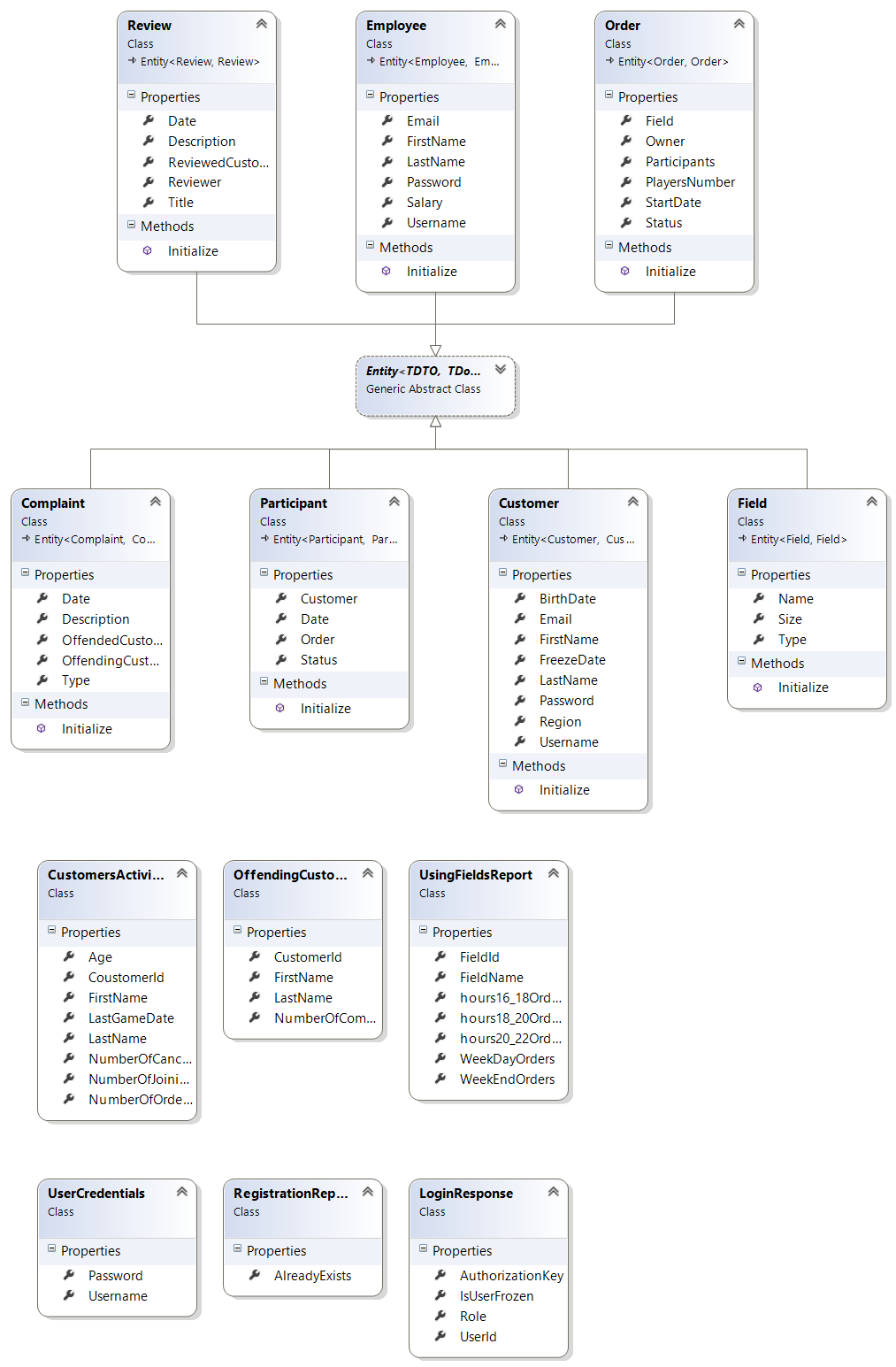




**דיאגרמת מחלקות עבור מחלקות ה-Query Processors:**



**דיאגרמת מחלקות עבור מחלקות ה-Dto:**



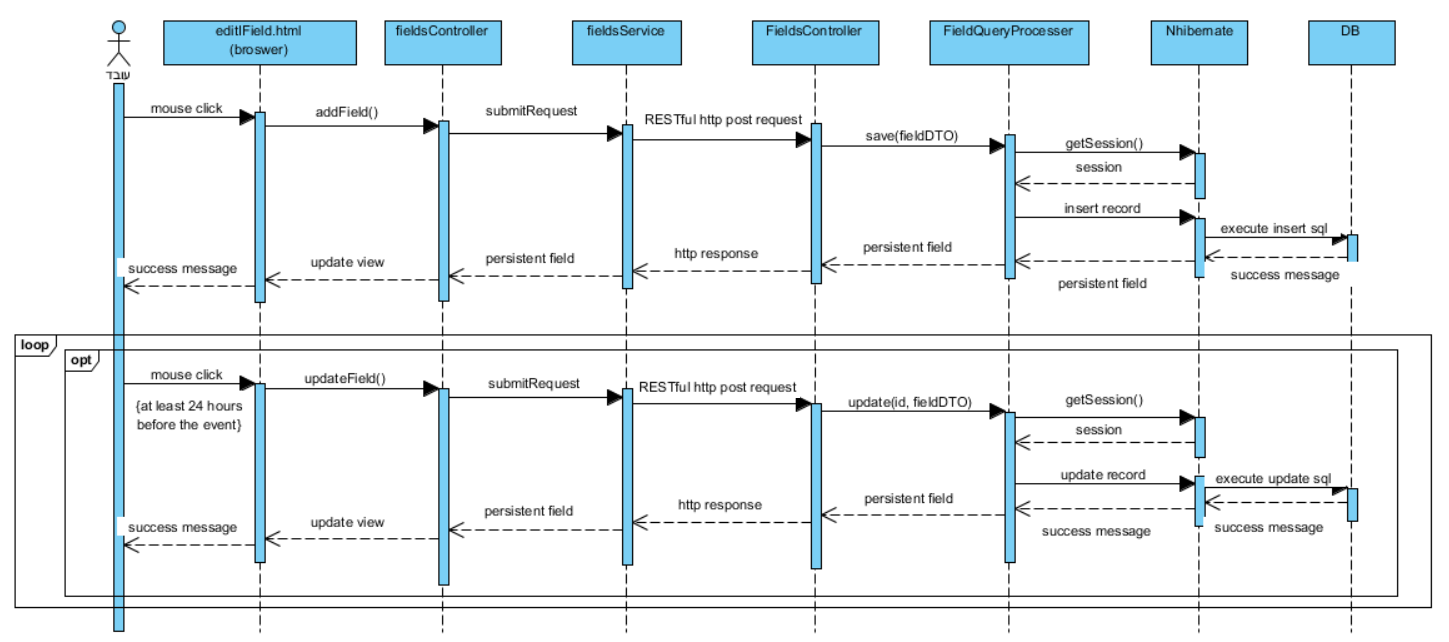
דיאגרמות רצף (Sequence Diagram)

בחלק זה נציג את התהליכים המרכזיים של המערכת בעזרת דיאגרמות רצף.

בדיאגרמות נתמקד בפעולות המשתמש ולא נראה בתרשימים פעולות כגון עליית הדף, סגירת הדף, הזנת הנתונים ובדיקת תקינות הנתונים. כמו כן לא נציג תרשמים עבור פעולות שאין בהן עדכון של מסד הנתונים כגון צפיה דוחות או צפיה בהזמנות קיימות.

**יצירת מגרש**

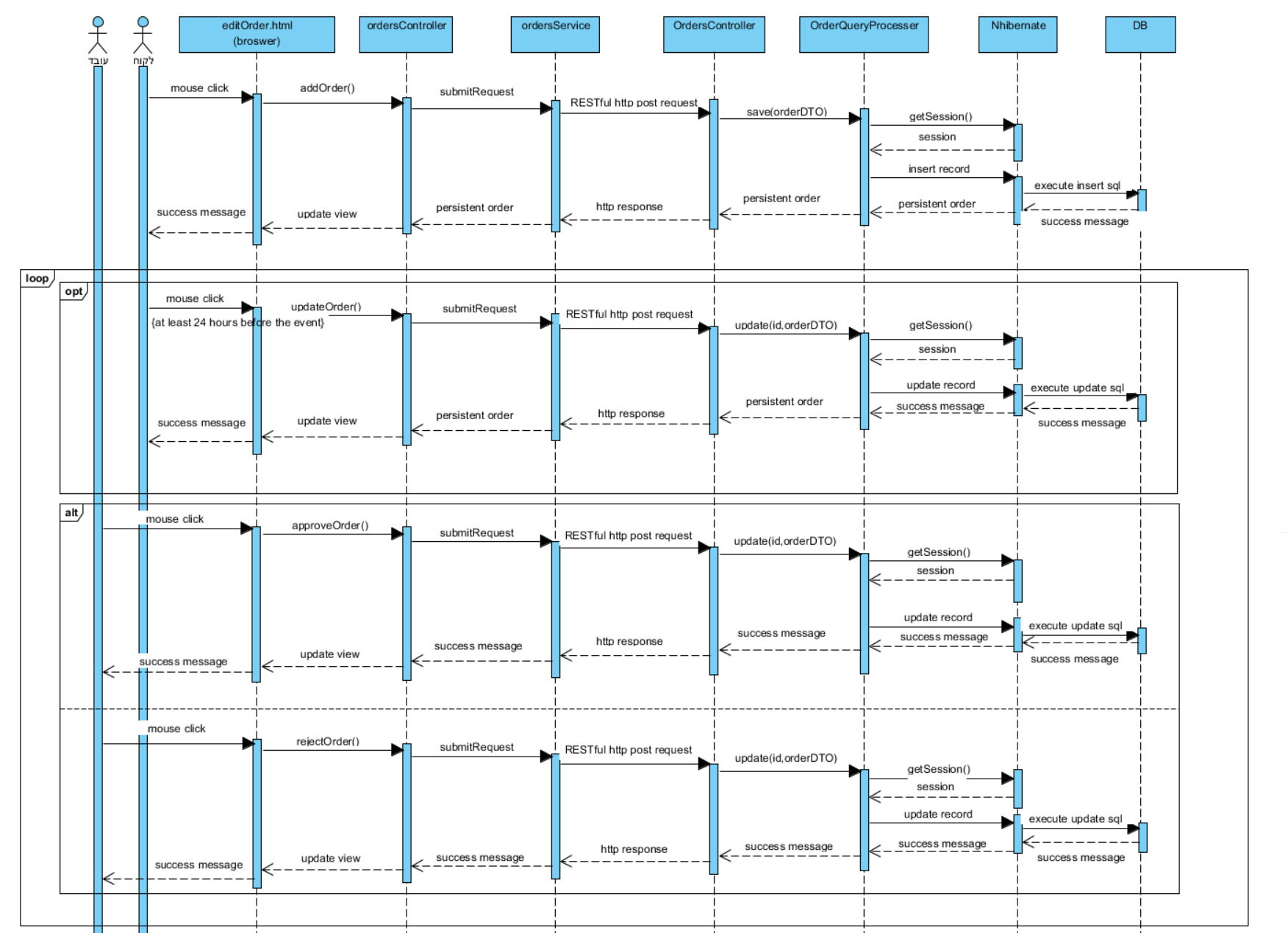
בהתליך זה העובד יוצר מגרש חדש. בלחיצה על הוספת מגרש בדף, השירות בצד הלקוח מבצע קריאת http עם נתוני המגרש החדש. הקונטרולר בצד השרת מבצע את פעולת ההוספה למסד הנתונים בעזרת מחולל השליפות. לאחר מכן הקונטרולר מחזיר ללקוח את הרשומה עם מזהה ממסד הנתונים. כעת העובד לעדכן המגרש בכל זמן נתון.



**הזמנת מגרש**

בהתליך זה הלקוח מבצע הזמנה חדשה. בלחיצה על הזמנת מגרש בדף, השירות בצד הלקוח מבצע קריאת http עם נתוני ההזמנה החדשה. הקונטרולר בצד השרת מבצע את פעולת ההוספה למסד הנתונים בעזרת מחולל השליפות. לאחר מכן הקונטרולר מחזיר ללקוח את הרשומה עם מזהה ממסד הנתונים מוצגת ללקוח הודעה מתאימה. הלקוח יכול לבצע עדכון של המגרש כל עוד לא נותרו פחות מ-24 שעות לתאריך ההזמנה.

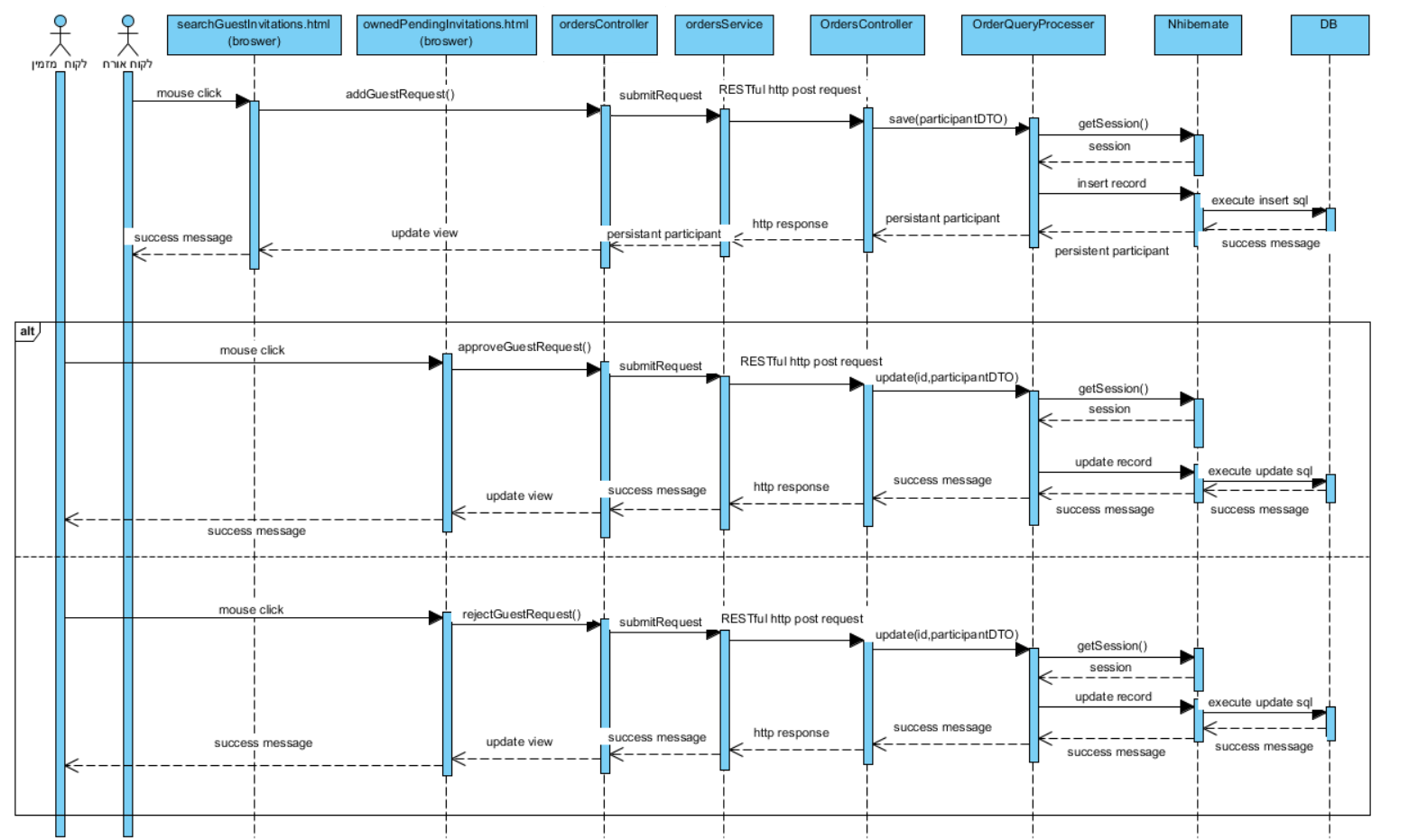
כעת העובד יכול לאשר או לדחות את ההזמנה ע"י שליחת הנתונים המתאימים לקונטרולר בצד השרת ובכך לעדכן את הנתונים. כמו כן הוא יקבל הודעת חיווי מתאימה.



**הצטרפות להזמנה קיימת**

בהתליך זה לקוח מבקש להצטרף להזמנה של לקוח אחר. בלחיצה על בקשה להצטרפות השירות בצד הלקוח מבצע קריאת http עם נתוני הבקשה החדשה. הקונטרולר בצד השרת מבצע את פעולת ההוספה למסד הנתונים בעזרת מחולל השליפות. לאחר מכן הקונטרולר מחזיר ללקוח את הרשומה עם מזהה ממסד הנתונים ומוצגת ללקוח הודעה מתאימה.

כעת הלקוח המזמין יכול לאשר או לדחות את ההזמנה ע"י שליחת הנתונים המתאימים לקונטרולר בצד השרת וכך לעדכן את רשומת בקשת ההצטרפות (participant). כמו כן הוא מקבל הודעת חיווי מתאימה.



**הקפאת לקוח**

בהתליך זה לקוח מזין תלונה על לקוח אחר והמנהל מחליט האם להשהות את הלקוח השני. הלקוח מזין את התלונה בלחיצה על כפתור שלח במסך פרטי הלקוח עליו הוא מתלונן, השירות בצד הלקוח מבצע קריאת http עם נתוני התלונה החדשה. הקונטרולר בצד השרת בצע את פעולת ההכנסה למסד הנתונים בעזרת מחולל השליפות, והוא מחזיר ללקוח הודעה שהפעולה בוצעה בהצלחה.

כעת המנהל רואה בדוח התלונות את התלונה ויכול להקפיא או להתעלם מהתלונה ע"י שליחת הנתונים המתאימים בעזרת השירות לקונטרולר בצד השרת וכך לעדכן את רשומת הלקוח (customer). כמו כן הוא מקבל הודעת חיבוי מתאימה.

