ManageYourTickets

מסמך תכנון ועיצוב

מגיש: יובל לוי

תוכן עניינים

תיאור כללי של המערכת

המערכת מורכבת משלוש שכבות:

- שכבת בסיס הנתונים
- שכבת הלוגיקה
- שכבת ממשק המשתמש

לכל שכבה יש את התפקיד הייחודי שלה והיא מורכבת ממחלקות אשר מבצעות תפקיד זה כפי שיתואר בהמשך המסמך.

כל שכבה יכולה לתקשר עם השכבה שמתחתיה בלבד. כלומר שכבת הממשק יכולה להשתמש במחלקות של שכבת הלוגיקה, שכבת הלוגיקה יכולה להשתמש במחלקות של שכבת בסיס הנתונים ושכבת בסיס הנתונים לא מתקשרת עם אף שכבה.

הנחות עבודה בכתיבת הפרוייקט

- אל המערכת יהיה ניתן להתחבר רק ממשתמש אחד במקביל גם אם היא תתוקן על מספר עמדות קצה.
- כדי למנוע מצב של חוסר עקביות בנתונים, האובייקטים של משתמש, אולם, אירוע והתרחשות- עבר לא יהיו ניתנים להסרה מבסיס הנתונים. שדה מיוחד יוקצה כדי לסמן אובייקטים מסוג אלו ש"נמחקו" ואינם זמינים לשימוש.
- בכל שדה בו נדרש לכתוב שם עם אותיות (בעבור שם פרטי, שם אירוע, וכו), האותיות היחידות שיתקבלו הן אותיות באנגלית, אין בדיקה של אות גדולה/קטנה וניתן לרשום שמות המורכבים ממספר מילים (כלומר מותר להשתמש בתו רווח).
- לכל אובייקט שישמר על בסיס הנתונים יהיה מספר זהות ייחודי חיובי.
- האובייקט שמייצג את המשתמש של האדמין הוא האובייקט היחיד בבסיס הנתונים שלו יהיה מספר זהות 0. קיומו בבסיס הנתונים יובטח ע"י המערכת.

מוסכמות רישום

- כל שם שנכתב בקוד (למחלקה/ממשק/מתודה וכו) המורכב מיותר ממילה אחת, ייכתב עם אות גדולה בתחילת כל מילה נוספת.
- שם מחלקה/מתודה/פרופטי גם מתחילים עם אות גדולה.
- שם של ממשק מתחיל עם אות I (איי גדולה).
- כל שם של שדה פרטי מתחיל ב- '_'.
- שמות קבועים מורכבים מאותיות גדולות בלבד עם '_' בין כל מילה המרכיבה את השם.
- שמות של enum וערכיהם נכתבים עם אות גדולה בתחילת כל מילה כולל הראשונה.

שכבת בסיס הנתונים

שכבה זו נמצאת בניימספייס "DataBaseTier". השכבה אחראית על ביצוע כל הפעולות מול בסיס הנתונים. דרכה יתבצעו הוספה, עדכון, שליפה או הסרה של אובייקט יחיד מבסיס הנתונים וכן שליפה של כל האובייקטים מסוג מסוים מבסיס הנתונים.

השכבה אינה מתקשרת באופן יזום עם אף שכבה אחרת. התקשורת מתבצעת רק דרך השכבה הלוגית כאשר היא יכולה להשתמש במתודות אשר בשכבה.

המערכת שיצרתי עושה שימוש בבסיס נתונים על שרת אינטרנטי של MongoDB, בסיס הנתונים יש "אוספים" כשבכל אוסף שמורים "מסמכים" המייצגים אובייקטים מסוג מסוים.

הקשר מתבצע לא בחיבור מלא, כלומר חיבור למאגר הנתונים מתרחש רק כאשר מתבקשים לבצע פעולה. מכאן שאם נשלוף אובייקט (אחד או יותר) ממאגר הנתונים ובמהלך העבודה משתמש אחר יעדכן את הפרטים שלו, אז אנחנו לא נראה את העדכון הזה עד אשר נשלוף את האובייקט מחדש. מסיבה זאת המערכת אינה תואמת לעבודה מקבילה של יותר ממשתמש אחד אשר יכולה ליצור אי-עקביות בבסיס הנתונים.

יחד עם זאת, בעקבות החלוקה לשלוש שכבות, בעתיד יהיה ניתן להחליף את המימוש של השכבה ולהשתמש בבסיס נתונים אחר ובלבד שיהיה ניתן לשמור בו את האובייקטים הנדרשים וימומשו בו המתודות שמשומשות ע"י שכבת הלוגיקה.

המחלקה MangeDB

השכבה מכילה מחלקה סטטית יחידה בשם "MangeDB" ובה מספר קבועים ומתודות סטטיות.

- קבועים:
  - MONGODB_CONNECTION_STRING – מחרוזת שמשמשת לחיבור לשרת MongoDB.
  - DATA_BASE_NAME – שמו של בסיס הנתונים על השרת.
  - USERS_COLLECTION – שמו של האוסף בו נשמרים האובייקטים מסוג User.
  - EVENTS_COLLECTION - שמו של האוסף בו נשמרים האובייקטים מסוג Event.
  - HALLS_COLLECTION - שמו של האוסף בו נשמרים האובייקטים מסוג Hall.
  - ORDERS_COLLECTION - שמו של האוסף בו נשמרים האובייקטים מסוג Order.
  - OCCURRENCES_COLLECTION - שמו של האוסף בו נשמרים האובייקטים מסוג Occurrence.
  - PASTOCCURRENCES_COLLECTION - שמו של האוסף בו נשמרים האובייקטים מסוג PastOccurrence.

- מתודות:
  - `private static IMongoCollection<T> GetDBCollection<T>()` –

    מחזיר מבסיס הנתונים את האוסף של האובייקטים מטייפ T.

  - `private static string GetCollectionName<T>()` –

    מחזיר את השם של האוסף בבסיס הנתונים של האובייקטים מטייפ T.

  - `public static void InsertObject<T>(T anObject)` –

    מוסיף לבסיס הנתונים את האובייקט anObject מסוג T.

  - `public static T LoadObjectById<T>(int id)` –

    שולף (אבל לא מסיר) מבסיס הנתונים את האובייקט מסוג T עם ה-id הנתון.

  - `public static List<T> LoadObjects<T>()` –

    שולף מבסיס הנתונים את כל האובייקטים מסוג T.

  - `public static List<T> LoadObjectsByIntProperty<T>(string propertyName, int propertyValue)` –

    שולף מבסיס הנתונים את כל האובייקטים מסוג T אשר יש להם בפרופטי propertyName את הערך propertyValue .

  - `public static void UpdateObject<T>(T anObject, int id)` –

    מעדכנת בבסיס הנתונים את האובייקט מסוג T עם מספר הזיהוי id להיות anObject.

  - `public static bool IsObjectExists<T>(int id)` –

    מחזיר 'true' אם אובייקט מסוג T עם מספר זיהוי id שמור בבסיס הנתונים.

  - `public static void DeleteObject<T>(int id)` –

    מסיר מבסיס הנתונים את האובייקט מסוג T עם מספר הזיהוי id.

טבלאות אובייקטים

הטבלאות הבאות מייצגות את האובייקטים ששכבת בסיס הנתונים צריכה לדעת לאחסן ולשלוף מבסיס הנתונים.

חשוב להדגיש שהטבלאות לא מייצגות בהכרח את האופן בו יישמרו האובייקטים (למשל ניתן יהיה לשנות את הטייפ של השדות), דבר זה נתון לשיקולי המימוש של שכבת הבסיס.

משתמש

שדה	סוג	הסבר
Id	int	מספר הזהות של המשתמש.
FirstName	string	
LastName	string	
City	string	
StreetName	string	
StreetNumber	int	
PhoneNumber	string	
Password	string	סיסמת התחברות למערכת
Permissions	Dictionary<string, bool>	מאגד את ההרשאות של המשתמש. המפתח זה שם ההרשאה והערך הבוליאני זה האם למשתמש יש את ההרשאה הזאת.
IsDeleted	bool	האם המשתמש מחוק.

אולם

שדה	סוג	הסבר
Id	int	מספר הזהות של האולם. מפתח ראשי.
Name	string	
Type	HallType (enum)	סוג האולם
Lines	int	כמות השורות באולם
SeatsInLine	int	כמות הכיסאות בכל שורה באולם
City	string	
StreetName	string	
StreetNumber	int	
OpeningHour	TimeSpan	
ClosingHour	TimeSpan	
Facilities	Dictionary<string, bool>	מאגד את המתקנים שבאולם. המפתח זה שם מתקן והערך הבוליאני זה האם באולם יש אותו או לא.
IsDeleted	bool	האם האולם מחוק.

אירוע

שדה	סוג	הסבר
Id	int	מספר הזהות של האירוע. מפתח ראשי.
Name	string	
EventDuration	TimeSpan	
TicketPrice	int	מחיר הבסיס של כרטיס
PricingMethod	TicketPricingMethod (enum)	שיטת התמחור של כרטיס
Demand	int	מספר ההתרחשויות שנדרשות בעבור האירוע
Merchandise	Dictionary<string, bool>	מאגד את המוצרים הנלווים הקיימים לאירוע. המפתח זה שם מוצר והערך הבוליאני זה האם הוא קיים לאירוע או לא.
IsDeleted	bool	האם האירוע מחוק.

התרחשות

שדה	סוג	הסבר
Id	int	מספר הזהות של ההתרחשות. מפתח ראשי.
EventId	int	מספר הזהות של האירוע שההתרחשות שייכת אליו.
EventName	string	השם של האירוע שההתרחשות שייכת אליו.
HallId	int	מספר הזהות של האולם בו מתקיימת ההתרחשות.
HallName	string	שם האולם בו מתקיימת ההתרחשות.
Date	DateOnly	
Hour	TimeSpan	
CreateDate	DateOnly	תאריך יצירת אובייקט ההתרחשות.
HallApproval	bool	האם התקבלה בעברו התרחשות זאת אישור לשימוש באולם
IsDeleted	bool	האם ההתרחשות מחוקה.

הזמנה

שדה	סוג	הסבר
Id	int	מספר הזהות של ההזמנה. מפתח ראשי.
OccurrenceId	int	מספר הזהות של ההתרחשות שההזמנה שייכת אליה.
Seats	List<(int, int)>	רשימת המושבים שנרכשו בהזמנה. כל איבר ברשימה הוא צמד מספרים כאשר הראשון מס' שורה והשני מס' מושב.
Price	int	העלות הכוללת של ההזמנה.
CustomerFirstName	string	
CustomerLastName	string	
PhoneNumber	string	טלפון של הלקוח
CreditCard	string	כרטיס אשראי של הלקוח
OrderDate	DateOnly	התאריך בו בוצעה ההזמנה.
IsDeleted	bool	האם ההזמנה מחוקה.

התרחשות-עבר

שדה	סוג	הסבר
Id	int	מספר הזהות של ההתרחשות-עבר. מפתח ראשי.
EventId	int	מספר הזהות של האירוע שההתרחשות הייתה שייכת אליו.
HallId	int	מספר הזהות של האולם בו התקיימה ההתרחשות.
Date	DateOnly	התאריך בו הייתה ההתרחשות.
TicketsSold	int	כמות הכרטיסים שנמכרו להתרחשות.
Revenue	int	סהכ ההכנסות מההזמנות להתרחשות.
IsDeleted	bool	האם ההתרחשות-עבר מחוקה.

שכבת הלוגיקה

שכבת הלוגיקה נמצאת בניימספייס "LogicalTier" והיא מכילה את רוב המתודות והפעולות שמתבצעות בקוד שאינן קשורות לגישה אל בסיס הנתונים או אל ניהול של הממשק הגראפי של המערכת.

בשכבת הלוגיקה נמצאים המחלקות של כל האובייקטים שנשמרים על בסיס הנתונים וכן מרבית האובייקטים האחרים והמחלקות הסטטיות שנעשים בהם שימוש בתוכנית.

שכבת הלוגיקה יכולה לקרוא למתודות משכבת בסיס הנתונים אך לא למתודות משכבת ממשק-המשתמש.

מחלקות

IDBObject.cs

ממשק למחלקות של אובייקטים שמיועדים להישמר על גבי בסיס הנתונים.

- שדות:

```
int Id
bool IsDeleted
```

- מתודות:

```
public static void Restore<T>(T anObject)
public void Delete()
public string IsChangeable()
```

User.cs

מחלקה שמייצגת משתמש במערכת. מממשת את הממשק IDBObject.

- שדות:

```
public int Id
public string FirstName
public string LastName
public string City
public string StreetName
public int StreetNumber
public string PhoneNumber
public string Password
public bool IsDeleted
public Dictionary<string, bool> Permissions
```

- מתודות:

```
public static string Create(string idNumber, string firstName, string lastName, string city, string streetName, string streetNumber, string phoneNumber, string password, bool usersPermission, bool hallsPermission, bool eventsPermission, bool occurrencesPermission, bool ordersPermission, bool revenuesPermission)

public string Edit(string firstName, string lastName, string city, string streetName, string streetNumber, string phoneNumber, string password, bool usersPermission, bool hallsPermission, bool eventsPermission, bool occurrencesPermission, bool ordersPermission, bool revenuesPermission)

private static string DataCheck(string idNumber, string firstName, string lastName, string city, string streetName, string streetNumber, string phoneNumber, string password)

public void Delete()

public string IsChangeable()

public string ChangePassword(string newPassword)

public override string ToString()
```

Event.cs

מחלקה שמייצגת אירוע. מממשת את הממשק IDBObject.

במחלקה מוגדר ה-enum "TicketPricingMethod" שמציין את השיטה על פיה יקבע המחיר של הכרטיסים לאירוע. ערכיו הם: Fixed, EarlyBird, LastMinute, Availability, ByDemand.

- שדות:

```
public int Id

public string Name

public TimeSpan EventDuration

public int TicketPrice

public TicketPricingMethod PricingMethod

public Dictionary<string, bool> Merchandise

public int Demand

public bool IsDeleted
```

- מתודות:

```
public static string Create(string idNumber, string eventName, TimeSpan eventDuration, string ticketPrice, TicketPricingMethod pricingMethod, bool hasBrochure, bool hasPoster, bool hasShirt)

public string Edit(string eventName, TimeSpan eventDuration, string ticketPrice, TicketPricingMethod pricingMethod, bool hasBrochure, bool hasPoster, bool hasShirt)

private static string DataCheck(string idNumber, string eventName, TimeSpan eventDuration, string ticketPrice)
```

```
public void Delete()
public void UpdateDemand(int demand)
public override string ToString()
```

Hall.cs

מחלקה שמייצגת אולם. מממשת את הממשק IDBObject.

במחלקה מוגדר ה-enum "`HallType` " שמציין את השיטה על פיה יקבע המחיר של הכרטיסים לאירוע. ערכיו הם: Theatre, Traverse, Thrust.

- שדות:

```
public int Id
public string Name
public HallType Type
public int Lines
public int SeatsInLine
public string City
public string StreetName
public int StreetNumber
public TimeSpan OpeningHour
public TimeSpan ClosingHour
public Dictionary<string, bool> Facilities
public bool IsDeleted
```

- מתודות:

```
public static string Create(string idNumber, string hallName, HallType
type, string lines, string seatsInLine, string city, string streetName,
string streetNumber, TimeSpan openingHour, TimeSpan closingHour, bool
hasParking, bool hasCafeteria, bool hasRestroom)
public string Edit(string hallName, HallType type, string lines, string
seatsInLine, string city, string streetName, string streetNumber, TimeSpan
openingHour, TimeSpan closingHour, bool hasParking, bool hasCafeteria,
bool hasRestroom)
private static string DataCheck(string idNumber, string hallName, HallType
type, string lines, string seatsInLine, string city, streetName, string
streetNumber, TimeSpan openingHour, TimeSpan closingHour)
public void Delete()
public string IsChangeable()
public override string ToString()
```

Occurrence.cs

מחלקה שמייצגת אובייקט של התרחשות של אירוע. מממשת את הממשק IDBObject.

- שדות:

```
public int Id

public int EventId

public string EventName

public int HallId

public string HallName

public DateOnly Date

public TimeSpan Hour

public DateOnly CreateDate

public bool HallApproval

public bool IsDeleted
```

- מתודות:

```
public static string Create(string idNumber, Event anEvent, Hall hall,
DateOnly date, TimeSpan hour)

public string Edit(Event anEvent, Hall hall, DateOnly date, TimeSpan hour)

private static string DataCheck(string idNumber, Event anEvent, Hall hall,
DateOnly date, TimeSpan hour, bool existingObject)

public void Delete()

public string IsChangeable()

public bool[,] GetSeatsMetrix()

public void UpdateHallApproval(bool isApproval)

public override string ToString()
```

Order.cs

מחלקה שמייצג אובייקט של הזמנה של כרטיסים. מממשת את הממשק IDBObject.

- שדות:

```
public int Id

public int OccurrenceId

public List<(int, int)> Seats

public int Price

public string CustomerFirstName

public string CustomerLastName

public string PhoneNumber
```

```
public string CreditCard

public DateOnly OrderDate

public bool IsDeleted
```

- מתודות:

```
public static string Create(string idNumber, int occurrenceId, List<(int, int)> seats, int price, string firstName, string lastName, string phoneNumber, string creditCardNumber)

public string Edit(string firstName, string lastName, string phoneNumber, string creditCard)

private static string DataCheck(string idNumber, string firstName, string lastName, string phoneNumber, string creditCardNumber)

public void Delete()

public string IsChangeable()

public override string ToString()
```

PastOccurrence.cs

מחלקה שמייצגת התרחשות-עבר, כלומר התרחשות שהייתה במערכת והיה ניתן לקנות עבורה כרטיסים. מטרת האובייקט היא לתעד את המכירות וההכנסות של אותה התרחשות לצורך הפקת דוחות. מממשת את הממשק IDBObject.

- שדות:

```
public int Id

public int EventId

public int HallId

public DateOnly Date

public int TicketsSold

public int Revenue

public bool IsDeleted
```

- מתודות:

```
public static void Create(Occurrence occurrence)

public void Delete()

public string IsChangeable()
```

ListMemento.cs

המחלקה מייצגת "snapshot" של רשימת אובייקטים שעברה סינון ושל פרטי המסנן שבוצע עליה. חלק מדפוס העיצוב "Memento".

- שדות:

```
public List<int> IdList
```

```
public string FilterProperty
public string FilterType
public string FilterValue
```

- מתודות:

```
public ListMemento(List<int> idList, string property, string Type, string value)
```

ITicketPricing.cs

ממשק למחלקות שמבצעות את פעולת החישוב של מחיר כרטיס בעבור אירוע מסוים. חלק מדפוס העיצוב "Strategy".
מכיל מתודה יחידה:

```
public int CalculatePrice(Event anEvent, Occurrence occurrence)
```

AvailabilityPricing.cs

מחלקה שמחשבת מחיר כרטיס לאירוע ששיטת התמחור שלו היא "Availability" ובה המחיר מתייקר ככל שנותרו פחות מושבים פנויים באולם. מממשת את הממשק "ITicketPricing". חלק מדפוס העיצוב "Strategy".
מכיל מתודה יחידה:

```
public int CalculatePrice(Event anEvent, Occurrence occurrence)
```

ByDemandPricing.cs

מחלקה שמחשבת מחיר כרטיס לאירוע ששיטת התמחור שלו היא "ByDemand" ובה המחיר נהיה זול יותר ככל שחולפים יותר ימים מהרכישה האחרונה של כרטיס. מממשת את הממשק "ITicketPricing". חלק מדפוס העיצוב "Strategy".
מכיל מתודה יחידה:

```
public int CalculatePrice(Event anEvent, Occurrence occurrence)
```

EarlyBirdPricing.cs

מחלקה שמחשבת מחיר כרטיס לאירוע ששיטת התמחור שלו היא "EarlyBird" ובה המחיר זול יותר ככל שקונים את הכרטיס מוקדם יותר. מממשת את הממשק "ITicketPricing". חלק מדפוס העיצוב "Strategy".
מכיל מתודה יחידה:

```
public int CalculatePrice(Event anEvent, Occurrence occurrence)
```

FixedPricing.cs

מחלקה שמחשבת מחיר כרטיס לאירוע ששיטת התמחור שלו היא "Fixed" ובה המחיר אינו משתנה ונשאר כפי שהוא. מממשת את הממשק "ITicketPricing". חלק מדפוס העיצוב "Strategy".

מכיל מתודה יחידה:

```
public int CalculatePrice(Event anEvent, Occurrence occurrence)
```

LastMinutePricing.cs

מחלקה שמחשבת מחיר כרטיס לאירוע ששיטת התמחור שלו היא "LastMinute" ובה המחיר זול יותר ככל שקונים את הכרטיס מאוחר יותר. מממשת את הממשק "ITicketPricing". חלק מדפוס העיצוב "Strategy".

מכיל מתודה יחידה:

```
public int CalculatePrice(Event anEvent, Occurrence occurrence)
```

State.cs

מחלקה אבסטרקטית שמייצגת את מצב ההתחברות של המשתמש הנוכחי במערכת. חלק מדפוס העיצוב "State".

- שדות:

```
protected CurrentUser currentUser
```

- מתודות:

```
public abstract void HandleTimeout()
public abstract void HandleActivity()
```

ActiveState.cs

מחלקה שמייצגת את מצבו של המשתמש הנוכחי במערכת כאשר הוא מחובר ופעיל. יורשת מהמחלקה "State". חלק מדפוס העיצוב "State".

- שדות:

```
protected CurrentUser currentUser
```

- מתודות:

```
public abstract void HandleTimeout()
public abstract void HandleActivity()
public override string ToString()
```

InactiveState.cs

מחלקה שמייצגת את מצבו של המשתמש הנוכחי במערכת כאשר הוא מחובר אך לא פעיל. יורשת מהמחלקה "State". חלק מדפוס העיצוב "State".

- שדות:

```
protected CurrentUser currentUser
```

- מתודות:

```
public abstract void HandleTimeout()
public abstract void HandleActivity()
```

```
public override string ToString()
```

LoggedOutState.cs

מחלקה שמייצגת את מצבו של המשתמש הנוכחי במערכת כאשר הוא אינו-מחובר. יורשת מהמחלקה "State". חלק מדפוס העיצוב "State".

- שדות:

```
protected CurrentUser currentUser
```

- מתודות:

```
public abstract void HandleTimeout()
public abstract void HandleActivity()
public override string ToString()
```

Constants.cs

מחלקה סטטית המכילה מספר קבועים שמשמשים את השכבה הלוגית ושכבת הממשק.

- שדות:

```
public const int ACTIVE_STATE_DURATION = 300000
public const int INACTIVE_STATE_DURATION = 30000
public const int ADMIN_ID = 0
public const string ADMIN_PASSWORD = "Admin"
public const int ID_MAX_LENGTH = 6
public const int PASSWORD_MIN_LENGTH = 4
public const int PASSWORD_MAX_LENGTH = 10
public const int EVENT_OCCURRENCE_INTERVAL = 30
public const int HALL_OCCURRENCE_INTERVAL = 30
public const int MIN_OPENING_HOUR = 6
public const int MAX_SEATS_IN_HALL = 500
public const int MAX_LINES_IN_HALL = 25
public const int MAX_SEATS_IN_LINE = 25
public const int MAX_PRICE_DISCOUNT = 25
public const int MAX_PRICE_INCREASEMENT = 25
```

CurrentUser.cs

מחלקה שמייצגת את המשתמש הנוכחי שמחובר למערכת, מממשת את הממשק " INotifyPropertyChanged". מממשת את דפוס העיצוב "Singleton ".

- שדות:

```
private static CurrentUser? _instance
```

```
public User User

public State State

public TimeSpan StateCountDown

public System.Timers.Timer InnerTimer

public event PropertyChangedEventHandler? PropertyChanged
```

- מתודות:

```
public static CurrentUser? GetInstance(User? user = null)

public void HandleTimeout()

public void HandleActivity()

public void ResetStateCountDown()

protected void OnPropertyChanged([CallerMemberName] string? propertyName =
null)
```

DatabaseMethods.cs

מחלקה סטטית שאחראית על הקישור עם שכבת בסיס הנתונים. זו המחלקה היחידה שקוראת למתודות משכבת בסיס הנתונים ולכן כל מתודה (משכבת הלוגיקה או הממשק) שתרצה לפעול על בסיס הנתונים, תשתמש במתודות ממחלקה זו.

- מתודות:

```
internal static void AddObject<T>(T anObject)

internal static void DeleteObject<T>(T anObject)

internal static void UpdateObject<T>(T anObject)

internal static bool IsObjectExists<T>(int id)

public static T? GetObject<T>(int id)

public static List<T> GetList<T>(bool isDeleted = false)

public static List<T> GetListByIntProperty<T>(string propertyName, int
propertyValue, bool isDeleted = false)
```

IdGenerator.cs

מחלקה סטטית שאחראית על חיפוש של מספר זיהוי תקין ופנוי לאובייקט מסוג T.
במחלקה מתודה יחידה:

```
public static int GenerateAvailableIdNumber<T>()
```

RevenuesReportProducer.cs

מחלקה שמייצרת אובייקט שמפיק דו"ח הכנסות מהזמנות להתרחשויות שהיו במערכת.

- שדות:

```
public List<Event> Events
public List<Hall> Halls
public DateTime? StartDate
public DateTime? EndDate
public bool ShowByEvents
public bool ShowByHalls
public bool ShowByDates
```

- מתודות:

```
public Tuple<bool, string> ProduceReport()
private string CanProduce()
private List<PastOccurrence> GetPastOccurrencesList()
private string GetRevenuesString(List<PastOccurrence> list)
```

SetUpMethods.cs

מחלקה סטטית עם מתודות שיש לקרוא להן לפני העלאת המערכת בכדי לדאוג להפעלה תקינה שלה.

- מתודות:

```
public static void SetUp()
private static void AdminVerification()
private static void DeletePastOccurrencesAndOrders()
```

StringChecks.cs

מחלקה סטטית עם מתודות שמבצעות בדיקות על מחרוזות.

- מתודות:

```
public static bool IsDigitsOnly(string aString)
public static bool IsLettersOrSpaces(string aString)
public static string IsPhoneNumber(string phoneNumber)
public static string IsCreditCardNumber(string creditNumber)
public static string IsVallidPassword(string password)
public static string IsVallidIdNumber(string idString)
```

שכבת ממשק המשתמש

שכבת ממשק-המשתמש נמצאת בניימספייס "UITier".

שכבת הממשק אחראית על הצגה וניהול של הרכיבים הגרפים במערכת והאינטראקציה עם המשתמש. לכן מרבית המחלקות בשכבה זו מהוות DataContext לאובייקט גראפי כלשהו שאיתו המשתמש מתקשר.

שכבת הממשק יכולה לגשת למחלקות משכבת הלוגיקה אך לא יכול לגשת את שכבת בסיס הנתונים.

מחלקות

Converters

קיימות מספר מחלקות הממשות את הממשק "IValueConverter", המשומשות ע"י קבצי xaml שונים בשכבה. מטרת כל אחת מהן היא להמיר ערך כלשהו לערך אחר, לא בהכרח מאותו טייפ. כל המחלקות מבצעות זאת באמצעות המתודה:

```
public object Convert(object value, Type targetType, object parameter, CultureInfo culture)
```

בנוסף למתודה זאת, בכל המחלקות יש את המתודה שאינה ממומשת:

```
public object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)
```

המחלקות הן:

- BooleanToVisibilityConverter.cs

ממירה 'true' ל- Visibility.Visible וממירה 'false' ל- Visibility.Collapsed.

- InverseBooleanToVisibilityConverter.cs

ממירה 'false' ל- Visibility.Visible וממירה 'true' ל- Visibility.Collapsed.

- ObjectToBooleanConverter.cs

ממירה כל אובייקט שאינו null ל-'true' וממירה אובייקט null ל-'false'.

- TimeSpanWithoutHoursConverter.cs

ממירה ערך TimeSpan למחרוזת מייצגת ללא שעות.

- TimeSpanWithoutSecondsConverter.cs

ממירה ערך TimeSpan למחרוזת מייצגת ללא השניות.

HallMap

HallMapGrid.cs

מחלקה אבסטרקטית ליצירת אובייקט Grid שייצג מפה של אולם כחלק מתהליך יצירת הזמנה חדשה. המפה תכלול את הבמה, מושבי האולם, מספר שורות ומושבים, סימון מושבים תפוסים וכן אפשרות לבחירת מושבים להזמנה. חלק מדפוס העיצוב "Template Method".

- שדות:

```
public Grid HallMap

public int Lines

public int SeatsInLine

public int GridRows

public int GridColumns

public bool[,] TakenSeats

public RelayCommand ClickSeatCommand
```

- מתודות:

```
public Grid CreateGrid(int lines, int seatsInLine, bool[,] takenSeats,
RelayCommand clickSeatCommand)

protected void PropertiesInitialize(int lines, int seatsInLine, bool[,]
takenSeats, RelayCommand clickSeatCommand)

protected virtual void SetGridRowsAndGridColumns()

protected void AddRowsAndColumns()

protected abstract void AddLinesNumbers()

protected abstract void AddSeatsButtons()

protected abstract void AddStage()

protected Button CreateSeatButton(int seatRow, int seatColumn)

protected static TextBlock CreateLineNumberTextBlock(string text)

protected static Border CreateStageBorder()

protected void AddElementToGrid(int row, int column, UIElement element,
int rowSpan = 1, int columnSpan = 1)
```

TheatreMapGrid.cs

מחלקה שיורשת את "HallMapGrid" ומייצרת אובייקט Grid שמייצג אולם מסוג "Theatre" שבו יציע אחד של מושבים ממול הבמה. חלק מדפוס העיצוב "Template Method".

- מתודות:

```
protected override void SetGridRowsAndGridColumns()
```

```
protected override void AddLinesNumbers()
protected override void AddSeatsButtons()
protected override void AddStage()
```

ThrustMapGrid.cs

מחלקה שיורשת את "HallMapGrid" ומייצרת אובייקט Grid שמייצג אולם מסוג "Thrust" שבו שלושה יציעים: מלפני משמאלה ומימינה לבמה. חלק מדפוס העיצוב "Template Method".

- מתודות:

```
protected override void SetGridRowsAndGridColumns()
protected override void AddLinesNumbers()
protected override void AddSeatsButtons()
protected override void AddStage()
```

TraverseMapGrid.cs

מחלקה שיורשת את "HallMapGrid" ומייצרת אובייקט Grid שמייצג אולם מסוג "Traverse" שבו שני יציעים משני צדדים מנוגדים לבמה. חלק מדפוס העיצוב "Template Method".

- מתודות:

```
protected override void SetGridRowsAndGridColumns()
protected override void AddLinesNumbers()
protected override void AddSeatsButtons()
protected override void AddStage()
```

RelayCommand.cs

מחלקה שמממשת את "ICommand ". מייצגת אובייקט שנטען להצמיד לכפתורים בממשק המשתמש. לאובייקט זה תוגדר פעולה שתתבצע בלחיצת הכפתור ותנאי מתי יהיה ניתן ללחוץ על הכפתור.

- שדות:

```
private Action<object?> _execute
private Func<object?, bool>? _canExecute
public event EventHandler? CanExecuteChanged
```

- מתודות:

```
public bool CanExecute(object? parameter)
public void Execute(object? parameter)
```

```
public void RaiseCanExecuteChanged()
```

מחלקות DataContext אבסטראקטיות

ViewModelBase.cs

מחלקה אבסטרקטית שנועדה לשמש כאב-טיפוס לכל מחלקה שהיא DataContext לאובייקט כלשהו בשכבת ממשק-המשתמש. היא מממשת את הממשק "INotifyPropertyChanged" שמעניק יכולת לעדכן את הרכיבים הגראפיים כאשר נתונים משתנים. בנוסף היא מכילה מתודה לטיפול במקרה של שגיאות.

- שדות:

```
public event PropertyChangedEventHandler? PropertyChanged
```

- מתודות:

```
protected void OnPropertyChanged([CallerMemberName] string? propertyName =
null)
```

```
protected static void ErrorHandler(Exception ex)
```

UserControlViewModelBase.cs

מחלקה אבסטרקטית שיורשת את "ViewModelBase". המחלקה מייצגת אב-טיפוס בעבור DataContext לאובייקט כלשהו מסוג UserControl. אין למחלקה מתודות או שדות והיא נועדה (ביחד עם WindowViewModelBase.cs) ליצור הבדל במחלקות שהן DataContext לאובייקט מסוג UserControl לבין אלו שהן DataContext לאובייקט מסוג Window.

WindowViewModelBase.cs

מחלקה אבסטרקטית שיורשת את "ViewModelBase". המחלקה מייצגת אב-טיפוס בעבור DataContext לאובייקט כלשהו מסוג Window. אין למחלקה מתודות או שדות והיא נועדה (ביחד עם UserControlViewModelBase.cs) ליצור הבדל במחלקות שהן DataContext לאובייקט מסוג Window לבין אלו שהן DataContext לאובייקט מסוג UserControl.

AddViewModelBase.cs

מחלקה אבסטרקטית שיורשת את "UserControlViewModelBase". המחלקה מייצגת אב-טיפוס בעבור DataContext לעמוד שמטרתו היא הוספה של אובייקט מסוג T אל בסיס הנתונים.

- שדות:

```
public string IdNumber
```

```
public string ErrorMessage
```

```
public RelayCommand ClearCommand
```

```
public RelayCommand AddCommand
public RelayCommand RandIdCommand
```

- מתודות:

```
protected abstract void Add(object? parameter)
protected abstract bool IsAllDataFilled(object? parameter)
protected abstract void Clear(object? parameter)
private void GetRandId(object? parameter)
```

EditViewModelBase.cs

מחלקה אבסטרקטית שיורשת את "UserControlViewModelBase". המחלקה מייצגת אב-טיפוס בעבור DataContext לעמוד שמטרתו היא עריכה של אובייקט מסוג T הקיים בבסיס הנתונים.

- שדות:

```
public IDBObject? SelectedObject
public string ErrorMessage
public RelayCommand EnterIdCommand
public RelayCommand EditCommand
public RelayCommand ResetCommand
```

- מתודות:

```
protected abstract void PropertiesInitialize()
protected abstract void Edit(object? parameter)
protected abstract bool IsAllDataFilled(object? parameter)
protected void Reset(object? parameter)
private void EnterId(object? parameter)
```

ListViewModelBase.cs

מחלקה אבסטרקטית שיורשת את "UserControlViewModelBase". המחלקה מייצגת אב-טיפוס בעבור DataContext לעמוד שמטרתו היא הצגת רשימה של כל האובייקטים מסוג T הקיימים בבסיס הנתונים. בעמוד יהיה ניתן למיין, לסנן ולהדפיס את הרשימה וכמו כן למחוק/לשחזר אובייקטים וכן לעבור לדף עריכה בעבור אובייקט נבחר מהרשימה.

- שדות:

```
public List<T> ObjectsList
public string ListHeader
public T? SelectedObject
public bool ShowDeleted
```

```
public List<string> PropertiesList
public string SelectedProperty
public List<string> FilterTypesList
public string SelectedFilterType
public string FilterValue
public bool IsFiltered
private Stack<ListMemento> _listFilterHistory
public RelayCommand DeleteCommand
public RelayCommand RestoreCommand
public RelayCommand EditCommand
public RelayCommand PrintCommand
public RelayCommand AddFilterCommand
public RelayCommand UndoFilterCommand
public RelayCommand ResetListCommand
public RelayCommand SortListCommand
```

- מתודות:

```
private List<T> GetObjectsList()
protected virtual void Delete(object? parameter)
private void Restore(object? parameter)
private void GoToEditPage(object? parameter)
private void OpenPrintWindow(object? parameter)
private void SortList(object? parameter)
private void AddFilter(object? parameter)
private void UndoFilter(object? parameter)
private void ResetList(object? parameter)
private static List<string> GetPropertyList()
private void SetFilterTypesList()
```

מחלקות DataContext קונקרטיות

ApproveOccurrencesViewModel.cs

מחלקה שיורשת את "WindowViewModelBase". משמשת כ- DataContext לחלון ובו המשתמש יכול לאשר או לדחות קיום של התרחשות באולם.

- שדות:

```
public List<Occurrence>? OccurrencesList
```

```
public Occurrence? SelectedOccurrence
public RelayCommand SetApprovalCommand
```

- מתודות:

```
private void SetApproval(object? parameter)
```

EnterIdViewModel.cs

מחלקה שיורשת את "WindowViewModelBase". משמשת כ- DataContext לחלון ובו המשתמש יכול להזין מספר זהות ולהעביר אותו לשימוש העמוד הנוכחי. מיועד לשימוש ע"י עמודי העריכה בשביל לקלוט מספר זהות של אובייקט אותו המשתמש רוצה לערוך.

- שדות:

```
public string InputId
public RelayCommand EnterCommand
```

- מתודות:

```
private static void CloseWindowWithTrueResult(object? parameter)
```

ErrorPageViewModel.cs

מחלקה שיורשת את "UserControlViewModelBase". המחלקה מייצגת DataContext לעמוד שמטרתו היא הצגת שגיאה ופרטיה. עמוד זה מיועד לשימוש כעמוד שגיאה שהמערכת תופנה אליו בכל פעם ששגיאה מתרחשת.

- שדות:

```
public string ErrorMessage
public string ExceptionType
public string StackTrace
```

EventAddViewModel.cs

מחלקה שיורשת את "AddViewModelBase" בעבור אובייקט Event. המחלקה מייצגת DataContext לעמוד שמטרתו היא הוספה של אובייקט מסוג Event אל בסיס הנתונים.

- שדות:

```
public string EventName
public DateTime EventDuration
public string TicketPrice
public TicketPricingMethod[] PricingMethodList
public TicketPricingMethod SelectedMethod
```

```
public bool HasBrochure
public bool HasPoster
public bool HasShirt
```

- מתודות:

```
protected override void Add(object? parameter)
protected override bool IsAllDataFilled(object? parameter)
protected override void Clear(object? parameter)
```

EventEditViewModel.cs

מחלקה שיורשת את "EditViewModelBase" בעבור אובייקט Event. המחלקה מייצגת DataContext לעמוד שמטרתו היא עריכה של אובייקט מסוג Event הקיים בבסיס הנתונים.

- שדות:

```
public string EventName
public DateTime EventDuration
public string TicketPrice
public TicketPricingMethod[] PricingMethodList
public TicketPricingMethod SelectedMethod
public bool HasBrochure
public bool HasPoster
public bool HasShirt
```

- מתודות:

```
protected override void PropertiesInitialize()
protected override void Edit(object? parameter)
protected override bool IsAllDataFilled(object? parameter)
```

EventListViewModel.cs

מחלקה שיורשת את "ListViewModelBase" בעבור אובייקט Event. המחלקה מייצגת DataContext לעמוד שמטרתו היא הצגת רשימה עם כל האובייקטים מסוג Event הקיימים בבסיס הנתונים. על הרשימה יהיה ניתן לבצע את הפעולות המתוארות במחלקת האב. למחלקה אין שדות או מתודות.

HallAddViewModel.cs

מחלקה שיורשת את "AddViewModelBase" בעבור אובייקט Hall. המחלקה מייצגת DataContext לעמוד שמטרתו היא הוספה של אובייקט מסוג Hall אל בסיס הנתונים.

- שדות:

```
public string HallName
public HallType[] HallTypesList
public HallType SelectedType
public string LinesNumber
public string SeatsInLine
public string City
public string StreetName
public string StreetNumber
public DateTime OpeningHour
public DateTime ClosingHour
public bool HasParking
public bool HasCafeteria
public bool HasRestroom
```

- מתודות:

```
protected override void Add(object? parameter)
protected override bool IsAllDataFilled(object? parameter)
protected override void Clear(object? parameter)
```

HallEditViewModel.cs

מחלקה שיורשת את "EditViewModelBase" בעבור אובייקט Hall. המחלקה מייצגת DataContext לעמוד שמטרתו היא עריכה של אובייקט מסוג Hall הקיים בבסיס הנתונים.

- שדות:

```
public string HallName
public HallType[] HallTypesList
public HallType SelectedType
public string LinesNumber
public string SeatsInLine
public string City
public string StreetName
public string StreetNumber
public DateTime OpeningHour
public DateTime ClosingHour
public bool HasParking
```

```
public bool HasCafeteria
public bool HasRestroom
```

- מתודות:

```
protected override void PropertiesInitialize()
protected override void Edit(object? parameter)
protected override bool IsAllDataFilled(object? parameter)
```

HallListViewModel.cs

מחלקה שיורשת את "ListViewModelBase" בעבור אובייקט Event. המחלקה מייצגת DataContext לעמוד שמטרתו היא הצגת רשימה עם כל האובייקטים מסוג Event הקיימים בבסיס הנתונים. על הרשימה יהיה ניתן לבצע את הפעולות המתוארות במחלקת האב. למחלקה אין שדות או מתודות.

HomePageViewModel.cs

מחלקה שיורשת את "UserControlViewModelBase". המחלקה מייצגת DataContext לעמוד שמטרתו היא להיות עמוד בית/דיפולט של המערכת. למחלקה אין שדות או מתודות.

MainWindowViewModel.cs

מחלקה שיורשת את "WindowViewModelBase". משמשת כ- DataContext לחלון הראשי של המערכת. המחלקה אחראית על התחברות/התנתקות של משתמש למערכת, הצגת תפריט הניווט בין העמודים השונים של המערכת וכן הצגת העמוד הנוכחי למשתמש.

- שדות:

```
public bool IsLogged
public CurrentUser? TheCurrentUser
public string IdNumber
public string Password
public bool[] OpenExpanders
public UserControlViewModelBase CurrentPageVM
public RelayCommand UserActivityCommand
public RelayCommand LogInCommand
public RelayCommand LogOutCommand
public RelayCommand NavBarButtonCommand
public RelayCommand OpenWindowCommand
```

- מתודות:

```
private void UserActivity(object? parameter)
```

```
private void LogIn(object? parameter)
private void LogOut(object? parameter)
private void InactivityLogOut()
private void NavBarButton(object? buttonContent)
private void OpenWindow(object? buttonContent)
public void ShowErrorPage(Exception ex)
```

OccurrenceAddViewModel.cs

מחלקה שיורשת את "AddViewModelBase" בעבור אובייקט Occurrence. המחלקה מייצגת DataContext לעמוד שמטרתו היא הוספה של אובייקט מסוג Occurrence אל בסיס הנתונים.

- שדות:

```
public  List<Event>? EventsList
public Event? SelectedEvent
public List<Hall>? HallsList
public Hall? SelectedHall
public DateTime? SelectedDate
public DateTime SelectedHour
```

- מתודות:

```
protected override void Add(object? parameter)
protected override bool IsAllDataFilled(object? parameter)
protected override void Clear(object? parameter)
```

OccurrenceEditViewModel.cs

מחלקה שיורשת את "EditViewModelBase" בעבור אובייקט Occurrence. המחלקה מייצגת DataContext לעמוד שמטרתו היא עריכה של אובייקט מסוג Occurrence הקיים בבסיס הנתונים.

- שדות:

```
public List<Hall>? HallsList
public Hall? SelectedHall
public DateTime? SelectedDate
public DateTime SelectedHour
```

- מתודות:

```
protected override void PropertiesInitialize()
protected override void Edit(object? parameter)
```

```
protected override bool IsAllDataFilled(object? parameter)
```

OccurrenceListViewModel.cs

מחלקה שיורשת את "ListViewModelBase" בעבור אובייקט Occurrence. המחלקה מייצגת DataContext לעמוד שמטרתו היא הצגת רשימה עם כל האובייקטים מסוג Occurrence הקיימים בבסיס הנתונים. על הרשימה יהיה ניתן לבצע את הפעולות המתוארות במחלקת האב. למחלקה אין שדות או מתודות.

OrderAddViewModel.cs

מחלקה שיורשת את "AddViewModelBase" בעבור אובייקט Order. המחלקה מייצגת DataContext לעמוד שמטרתו היא הוספה של אובייקט מסוג Order אל בסיס הנתונים.

- שדות:

```
public List<Event>? EventsList
public Event? SelectedEvent
public List<Occurrence>? OccurrencesList
public Occurrence? SelectedOccurrence
public bool IsOccurrenceSelected
public Grid? HallMap
public Button? SelectedSeatButton
public bool IsSeatSelected
public List<(int, int)> SelectedSeats
public int TicketPrice
public int SeatsCount
public int OrderPrice
public bool IsOrderReady
public string CustomerFirstName
public string CustomerLastName
public string PhoneNumber
public string CreditCard
public RelayCommand BackCommand
public RelayCommand SetSeatsSelectionCommand
public RelayCommand OrderReadyCommand
```

- מתודות:

```
protected override void Add(object? parameter)
```

```
protected override bool IsAllDataFilled(object? parameter)
protected override void Clear(object? parameter)
private void Back(object? parameter)
private void SetSeatsSelection(object? parameter)
private void SetTicketPrice()
private void CreateHallMapGrid()
private void ClickSeat(object? parameter)
```

OrderEditViewModel.cs

מחלקה שיורשת את "EditViewModelBase" בעבור אובייקט Order. המחלקה מייצגת DataContext לעמוד שמטרתו היא עריכה של אובייקט מסוג Order הקיים בבסיס הנתונים.

- שדות:

```
public Occurrence? OrderOccurrence
public int OrderPrice
public List<(int, int)> OrderSeats
public string CustomerFirstName
public string CustomerLastName
public string PhoneNumber
public string CreditCard
```

- מתודות:

```
protected override void PropertiesInitialize()
protected override void Edit(object? parameter)
protected override bool IsAllDataFilled(object? parameter)
```

OrderListViewModel.cs

מחלקה שיורשת את "ListViewModelBase" בעבור אובייקט Order. המחלקה מייצגת DataContext לעמוד שמטרתו היא הצגת רשימה עם כל האובייקטים מסוג Order הקיימים בבסיס הנתונים. על הרשימה יהיה ניתן לבצע את הפעולות המתוארות במחלקת האב. למחלקה אין שדות או מתודות.

PrintedObjectListViewModel.cs

מחלקה שיורשת את "WindowViewModelBase". משמשת כ- DataContext לחלון ובו תודפס רשימה של אובייקטים מסוג T. מיועד לשימוש ע"י עמודי הרשימה כשהמשתמש רוצה להדפיס את הרשימה המוצגת בעמוד.

- שדות:

```
public string ObjectListString
```

- מתודות:

```
private static string CreateObjectListString(List<T> objectsList)
```

RevenuesReportViewModel.cs

מחלקה שיורשת את "UserControlViewModelBase ". משמשת כ- DataContext לעמוד שמטרתו היא הדפסת דוח הכנסות מהרכישות שבוצעו במערכת להתרחשויות עבר. דוח ההכנסות יופק ע"פ נתונים שיוכנסו ע"י המשתמש (אירועים, אולמות, תאריכים) ויוצגו בו גם קטגוריות אופציונאליות שיבחרו ע"י המשתמש (אירועים, אולמות, תאריכים).

- שדות:

```
public List<Event> EventsList
public List<Event> SelectedEvents
public List<Hall> HallsList
public List<Hall> SelectedHalls
public DateTime? SelectedStartDate
public DateTime? SelectedEndDate
public bool ShowByEvents
public bool ShowByHalls
public bool ShowByDates
public string ErrorMessage
public String RevenuesReport
public RelayCommand ProduceCommand
public RelayCommand ClearCommand
```

- מתודות:

```
private void Produce(object? parameter)
private void Clear(object? parameter)
```

SetEventDemandViewModel.cs

מחלקה שיורשת את "WindowViewModelBase". משמשת כ- DataContext לחלון ובו המשתמש יכול לקבוע את כמות הדרישה להתרחשויות בעבור כל אחד מהאירועים שבבסיס הנתונים.

- שדות:

```
public List<Event>? EventsList
public Event? SelectedEvent
```

```
public int Demand
public RelayCommand SetDemandCommand
```

- מתודות:

```
private void SetDemand(object? parameter)
```

UserAddViewModel.cs

מחלקה שיורשת את "AddViewModelBase" בעבור אובייקט User. המחלקה מייצגת DataContext לעמוד שמטרתו היא הוספה של אובייקט מסוג User אל בסיס הנתונים.

- שדות:

```
public string FirstName
public string LastName
public string City
public string StreetName
public string StreetNumber
public string PhoneNumber
public string Password
public bool UsersPermission
public bool HallsPermission
public bool EventsPermission
public bool OccurrencesPermission
public bool OrdersPermission
public bool RevenuesPermission
```

- מתודות:

```
protected override void Add (object? parameter)
protected override bool IsAllDataFilled(object? parameter)
protected override void Clear(object? parameter)
```

UserDetailsViewModel.cs

מחלקה שיורשת את "UserControlViewModelBase". המחלקה מייצגת DataContext לעמוד שמטרתו היא להציג למשתמש שמחובר כעת למערכת את פרטיו ולתת לו אפשרות לשנות את סיסמתו.

- שדות:

```
public User? TheUser
public string Password
```

```
public bool PasswordEnabled
public RelayCommand ChangePasswordCommand
```

- מתודות:

```
public void ChangePassword (object? parameter)
```

UserEditViewModel.cs

מחלקה שיורשת את "EditViewModelBase" בעבור אובייקט User. המחלקה מייצגת DataContext לעמוד שמטרתו היא עריכה של אובייקט מסוג User הקיים בבסיס הנתונים.

- שדות:

```
public string FirstName
public string LastName
public string City
public string StreetName
public string StreetNumber
public string PhoneNumber
public string Password
public bool UsersPermission
public bool HallsPermission
public bool EventsPermission
public bool OccurrencesPermission
public bool OrdersPermission
public bool RevenuesPermission
```

- מתודות:

```
protected override void PropertiesInitialize()
protected override void Edit(object? parameter)
protected override bool IsAllDataFilled(object? parameter)
```

UserListViewModel.cs

מחלקה שיורשת את "ListViewModelBase" בעבור אובייקט User. המחלקה מייצגת DataContext לעמוד שמטרתו היא הצגת רשימה עם כל האובייקטים מסוג User הקיימים בבסיס הנתונים. על הרשימה יהיה ניתן לבצע את הפעולות המתוארות במחלקת האב. למחלקה אין שדות ומתודות מלבד הגדרה מחדש של Delete בכדי להתאימה למחיקת משתמשים.

- מתודות:

```
protected override void Delete(object? parameter)
```

דפוסי עיצוב

Memento

Memento הוא דפוס עיצוב המאפשר לשמור ולשחזר את המצב הקודם של אובייקט מבלי לחשוף את פרטי היישום שלו.

המערכת עושה שימוש בדפוס עיצוב זה במחלקה ListMemento, מחלקה זו נועדה לשמור את המצב של רשימת אובייקטים לפני שבוצע לה סינון. המטרה היא שיהיה אפשר לבצע "ביטול סינון" ולשחזר אחורה את המצב של הרשימה לפני שהופעל הסינון. באובייקט של מחלקה זו יישמרו רשימת האובייקטים וכן המאפיינים של המסנן שהופעל על הרשימה. כך בעת השחזור יהיה ניתן לקבל מידע על המסנן שבוצע על הרשימה וכעת בוטל.

המחלקה שתהיה אחראית על הצגה למשתמש וניהול של הרשימות תכיל מחסנית של ListMemento. בעבור כל סינון שיבוצע על הרשימה יווצר אובייקט ListMemento ויישמר במחסנית. בכל פעם שהמשתמש ירצה לבצע "ביטול סינון" ישלף האובייקט ListMemento העליון במחסנית והאובייקטים שהיו ברשימה ישוחזרו ממנו וכמו כן גם הפרטים של המסנן שהופעל על הרשימה.

Method template

Template Method הוא דפוס עיצוב המגדיר שלד של אלגוריתם במחלקת-על ומאפשר למחלקות-בת היורשות ממנה לשנות שלבים ספציפיים של האלגוריתם מבלי לשנות את המבנה שלו.

המערכת עושה שימוש בדפוס עיצוב זה בכדי לייצר מפות אולם לסוגים שונים של אולמות. מפת האולם תהיה אובייקט שיכלול את הבמה, מושבי האולם, מספור שורות ומושבים, סימון מושבים תפוסים וכן אפשרות לבחירת מושבים להזמנה.

המחלקה HallMapGrid היא מחלקת העל. זו מחלקה אבסטרקטית ובה מתודה ראשית אחת שמכילה את אלגוריתם ליצירת אובייקט Grid שמייצג את מפת האולם. האלגוריתם מורכב מקריאה למתודות (אבסטרקטיות או לא) שמחלקות הבת יכולות לכתוב מחדש ובכך להתאים את מפת האולם לסוג האולם שהן מייצגות מבלי לשנות את מתודת האלגוריתם הראשית.

מחלקות הבת הן: TheatreMapGrid, ThrustMapGrid ו- TraverseMapGrid שמייצרות מפות אולם עם: יציע מושבים אחד, 3 יציעים, 2 יציעים (בהתאמה).

בחלק הקוד שיציג למשתמש את מפת האולם אין צורך לכתוב קוד נפרד בעבור כל אחת ממחלקות-הבת. יש אובייקט מסוג HallMapGrid שבזמן ריצה יוזן בו אובייקט של מחלקת-הבת המתאימה לסוג האולם שנרצה להציג, ותקרא מתודת האלגוריתם שמשותפת לכולן.

Singleton

Singleton הוא דפוס עיצוב המבטיח שלמחלקה יש רק מופע אחד, תוך מתן מצביע גלובלי למופע זה.

המערכת עושה שימוש בדפוס עיצוב זה במחלקה CurrentUser משכבת הלוגיקה. מחלקה זו מייצגת את המשתמש שכרגע מחובר למערכת, ומכיוון שלמערכת ניתן להתחבר רק ממשתמש אחד במקביל, השימוש ב- Singleton יבטיח זאת.

למחלקה CurrentUser תהיה תכונה סטטית פרטית בשם _instance שתשמור את האובייקט היחיד (אם קיים) של המחלקה. הבנאי של המחלקה יהיה פרייבט כדי למנוע יצירה בלתי מבוקרת של אובייקטים. המתודה הסטטית GetInstance של המחלקה תהיה אחראית על יצירת אובייקט CurrentUser , שמירה שלו ב- _instance (גם אם זה אומר למחוק את האובייקט ששמור שם כבר) והחזרה של _instance למשתמש. זאת תהיה הדרך היחידה ליצור אובייקט CurrentUser וגם זאת תהיה הדרך היחידה לקבל גישה ל- _instance. כך ייאכף כי יצירה של אובייקט חדש בהכרח תמחוק את האובייקט הקיים (במידה ויש).

State

State הוא דפוס עיצוב המאפשר לאובייקט לשנות את התנהגותו כאשר מצבו הפנימי משתנה.

המערכת עושה שימוש בדפוס עיצוב זה בעבור ייצוג מצב החיבור של המשתמש הנוכחי במערכת (נקרא מצב-משתמש). לאחר שמשתמש מתחבר למערכת, לחיבור שלו יש מצבים שונים: פעיל, לא-פעיל ומנותק. מצב-המשתמש רלוונטי בעיקר בעבור שני מקרים: כאשר המשתמש מבצע פעולה כלשהי עם העכבר או המקלדת וכאשר נגמר הזמן שהוקצה למצב-המשתמש (הזמן מוגדר בנפרד). כאשר שני מקרים אלו מתרחשים אז בכל מצב-משתמש התגובה של המערכת היא אחרת כאשר לפעמים התגובה היא מעבר למצב-משתמש אחר. דפוס העיצוב State מאפשר למערכת לעבור בין המצבים בצורה נוחה וגם להתייחס לכל המצבים כאחד מבלי לכתוב קוד נפרד לכל מצב.

בשכבת הלוגיקה קיימת המחלקה האבסטרקטית State ולה שלוש מחלקות- בת: ActiveState, InactiveState ו- LoggedOutState. במערכת יהיה משתנה מסוג State שייצג את המצב הנוכחי של המשתמש הנוכחי במערכת, כשבפועל הערך המוזן בו יהיה אחד משלוש מחלקות-הבת.כאשר יש צורך לקרוא למתודה של משתנה זה אז בכל מצב-משתמש תקרא אותה מתודה כשבפועל המימוש שלה ע"י כל מחלקת-בת יכול להיות שונה. בנוסף גם מעבר בין מצבי-משתמש (שינוי ערכו של המשתנה) מטופל בתוך מחלקות הבת עצמן כחלק ממימוש זה. בכך נחסך קוד גם בקריאה למתודות השונות וגם במעבר בין המצבים.

Strategy

Strategy הוא דפוס עיצוב המאפשר להגדיר משפחה של אלגוריתמים, להכניס כל אחד מהם למחלקה נפרדת ולהפוך את האובייקטים שלהם לניתנים להחלפה.

המערכת עושה שימוש בדפוס עיצוב זה כאשר מחשבים את עלות כרטיס בעבור התרחשות של אירוע. לכל אירוע יש שיטת תמחור שונה לכרטיסים וכאשר יוצרים במערכת הזמנה חדשה להתרחשות אלגוריתם החישוב של עלות הכרטיסים נקבע לפי שיטת התמחור של האירוע שההתרחשות שייכת לו.

בשכבת הלוגיקה נמצא הממשק ITicketPricing ובו מתודה אחת שהיא למעשה האלגוריתם לחישוב מחיר כרטיס. בנוסף קיימות 5 מחלקות סטטיות שונות שמממשות את הממשק ITicketPricing. מחלקות אלו הן: AvailabilityPricing, ByDemandPricing, EarlyBirdPricing, FixedPricing ו- LastMinutePricing. כל אחת מהן מממשת את מתודת אלגוריתם החישוב בעבור סוג אחר של שיטת תמחור.

בחלק הקוד שאחראי על חישוב מחיר הכרטיס אין צורך לכתוב קוד נפרד בעבור כל אחת מהמחלקות הממשות את ITicketPricing. יהיה משתנה מסוג ITicketPricing שבזמן ריצה יוזן בו אובייקט של מחלקה-מממשת המתאימה לשיטת החישוב של האירוע שאליו מתבצעת ההזמנה, וממנו תקרא מתודת האלגוריתם שמשותפת לכולן אך תופעל בהתאם לערך בזמן ריצה של המשתנה.

שינויים עתידיים אפשריים במערכת

- ניתן להוסיף בדיקה שכל המידע שנשמר באובייקטים בבסיס הנתונים עקבי. כלומר לוודא שלא קיימים שני אובייקטים שהמידע שלהם מתנגש זה עם זה. למשל, מושבים להתרחשות מסוימת שמופיעים ככאלו שנרכשו בשתי הזמנות שונות או התרחשות שהאירוע שלה לא קיים בבסיס הנתונים.
נוכל להוסיף את בדיקות אלו ע"י הוספת מתודות מתאימות למחלקה SetUpMethods. המתודות ירוצו בהפעלת המערכת ויעברו על בסיס הנתונים ויבדקו את עקביות המידע בו. בדומה למתודות הנוכחיות ב- SetUpMethods, המתודות יוכלו לטפל באי-עקביות ע"י ניסיון לתקן את הבעיה (למשל ע"י מחיקה או שינוי נתונים) רק להודיע למשתמש שנמצאה בעיה בנתונים.

- הוספת מצב-משתמש חדש שייקרא "שמור" שיהיה המצב הראשון שהמשתמש הנוכחי מקבל לאחר התחברות והוא יימשך זמן קבוע שבו המשתמש לא יוכל להיות מנותק אוטומטית עקב חוסר פעילות. כלומר בניגוד למצב "פעיל" שבו הטיימר מתחיל מחדש בכל פעולה שהמשתמש מבצע, במצב "זמן שמור" הפעילות לא תשפיע על הטיימר. לאחר שהזמן של מצב זה יסתיים המשתמש יועבר למצב-משתמש "פעיל".
שינוי זה יתבצע ע"י יצירת מחלקה חדשה "SaveState" שתירש מ-State ותממש את המחלקות שלה בהאתם למה שתיארנו, בנוסף נצטרך להגדיר את המצב-משתמש ההתחלתי להיות המצב SaveState.

- המחלקה ListMemento מממשת את דפוס העיצוב Memento. אחד מתפקידיה של מחלקה זו הוא שמירת רשימת אובייקטים. במימוש הנוכחי זה מתבצע ע"י שמירה של מספרי הזהות של האובייקטים ברשימה. זאת מכיוון שאובייקט כולו ניתן למשוך מבסיס הנתונים בעזרת מספר הזהות ולכן ניתן להסתפק בשמירת מספרי זהות בלבד. דבר זה חוסך בעלות-מקום אבל מצריך יותר עלות-זמן כאשר אנו יוצרים את האובייקט ListMemento ובעיקר כאשר אנו משחזרים את הרשימה (כי אז צריך לגשת מחדש אל בסיס הנתונים ולמשוך את האובייקטים).
אם נרצה לתעדף עלות-זמן על פני עלות-מקום נוכל לשמור את הרשימה כולה. נשנה את המחלקה ListMemento כך שבמקום התכונה של רשימת מספרי זהות תהיה רשימה של אובייקטים מסוג מסוים. בנוסף נשנה את הקוד האחראי על שחזור הרשימה, הוא לא יצטרך לגשת אל בסיס הנתונים אלא רק לקחת ישירות את הרשימה מהאובייקט ListMemento.

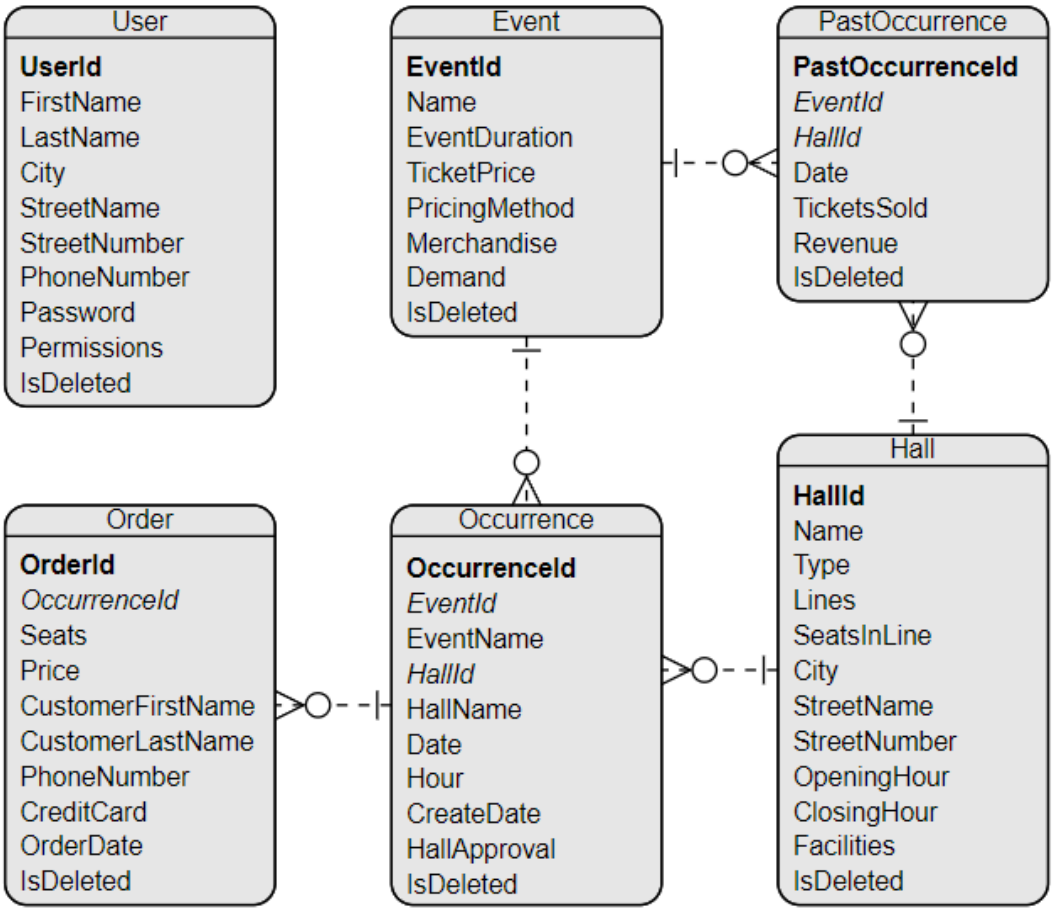
- נתונים רבים במערכת שמורים במחלקה Constants בשכבת הלוגיקה. שמירה זאת במקום מרוכז מאפשרת בקלות שינוי של נתונים במערכת ללא שינוי כלשהו בקוד אחר.
למשל במחלקת Constants מוגדרת ההגבלה לגודל האולם שניתן ליצור. אם נרצה לאפשר אולם גדול יותר, נניח מבחינת מספר השורות, כל שנצטרך לעשות הוא לשנות את הקבוע המתאים במחלקה. כל מקום במערכת שמשתמש בנתון זה יושפע גם הוא ללא צורך בהתערבות נוספת.

- החלוקה של המערכת ל-3 שכבות נפרדות מאפשרת לשנות את המימוש של אחד השכבות בלי שנצטרך לבצע שינויים בשכבות האחרות.
אם למשל נרצה לממש את שכבת בסיס הנתונים ע"י שמירה על שרת מסוג אחר, נוכל לבצע זאת. צריך רק שהשרת החדש יוכל לשמור את אותם האובייקטים הנדרשים ושיכתבו כל המתודות שנקראות ע"י שכבת הלוגיקה. אך הדרך בה ישמרו האובייקטים והקוד הפנימי של המתודות אינו ישפיע על שאר המערכת.

דיאגרמות

דיאגרמת בסיס הנתונים

- שדה מובלט הוא מפתח ראשי.
- שדה עם כתב נטוי הוא מפתח משני.
- למפתחות בכל ישות נקבע שם ייחודי אך בפועל יתאפשר ששמות המפתחות יהיו שונים.



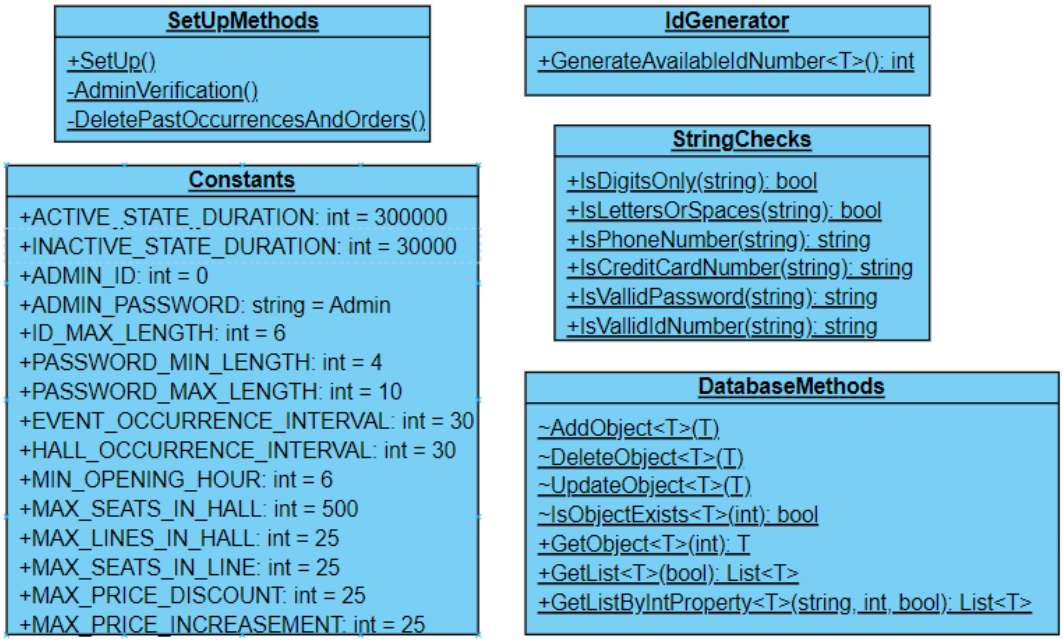
דיאגרמות מחלקות

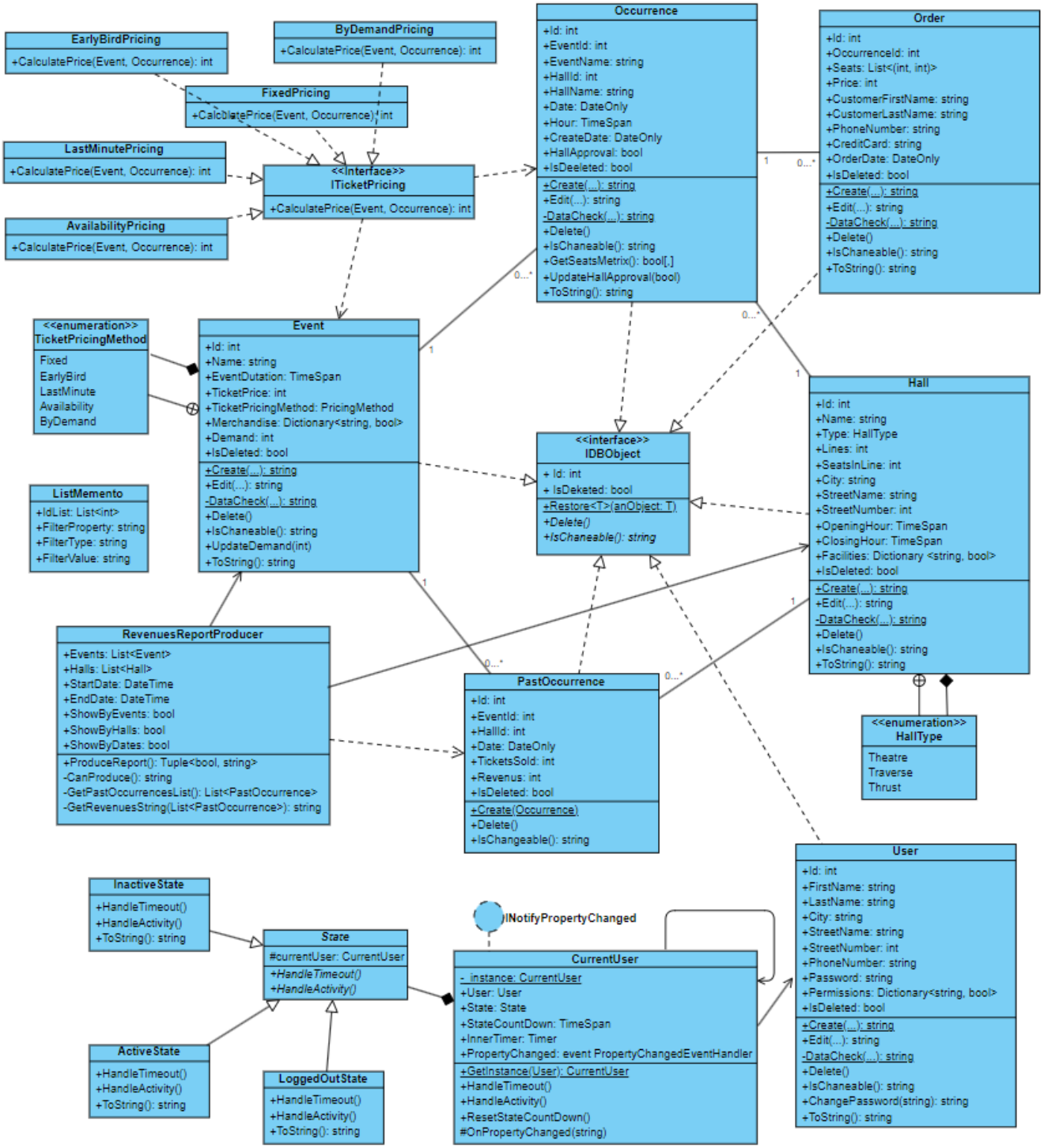
כדי ליצור דיאגרמות קריאות ככל הניתן:

- בעבור מתודות עם רשימת פרמטרים מאוד ארוכה רשמתי (....) במקום את הפרמטרים.
- אם למשתנה פרייבט קיים מאפיין (פרופרטי) פומבי, אז בדיאגרמה ייכתב רק המאפיין.
- אם מחלקה יורשת מחלקה או מממשת ממשק שלא מופיעים בשכבה שהדיאגרמה מייצגת, אז שם המחלקה/ממשק יצוין בבועה מעל המחלקה.
- משתנה שמחלקה יורשת ממחלקת-אב לא יופיע.
- מתודה שמחלקה יורשת מחלקת-אב תופיע רק אם היא עברה שכתוב (override).
- אם בין מחלקה A לבין מחלקה/ממשק B מתקיים קשר. וקיימת מחלקה C שיורשת/מממשת את B ומקיימת גם היא את אותו סוג קשר עם A. אז הדיאגרמה לא תציין את הקשר בין A לבין C.

שכבת הלוגיקה

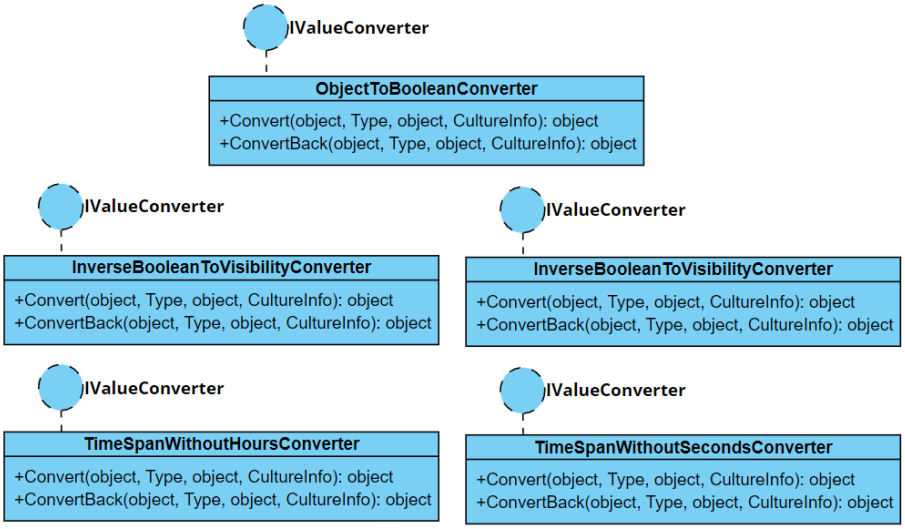
מחלקות סטטיות



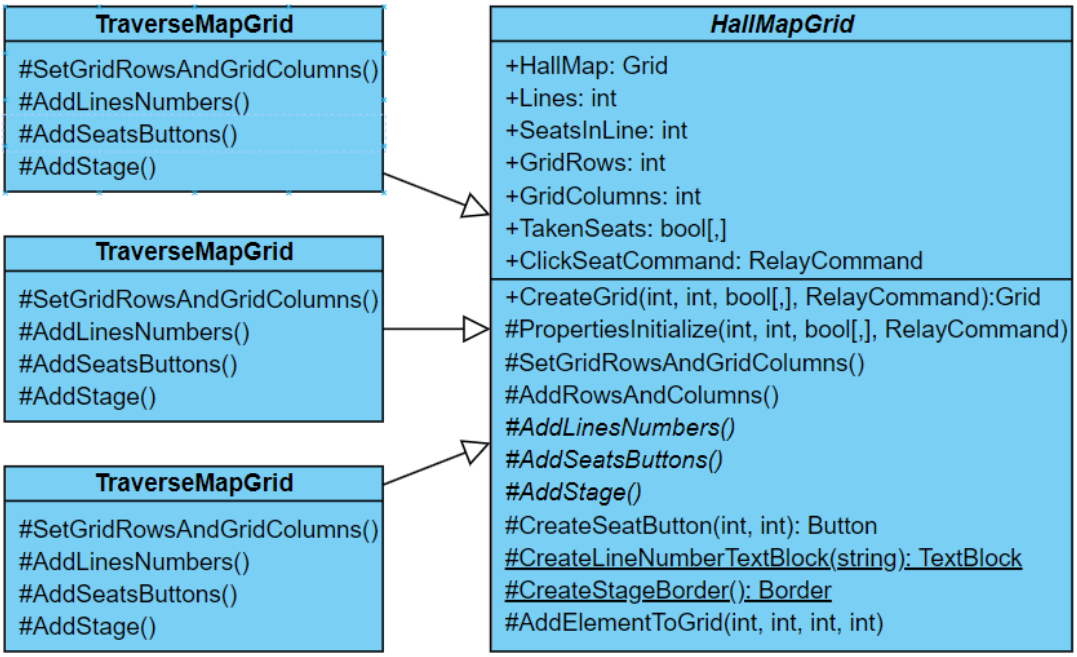

EarlyBirdPricing
+CalculatePrice(Event, Occurrence): int
ByDemandPricing
+CalculatePrice(Event, Occurrence): int
FixedPricing
+CalculatePrice(Event, Occurrence): int
LastMinutePricing
+CalculatePrice(Event, Occurrence): int
<<Interface>>
ITicketPricing
+CalculatePrice(Event, Occurrence): int
AvailabilityPricing
+CalculatePrice(Event, Occurrence): int
Occurrence
+Id: int
+EventId: int
+EventName: string
+HallId: int
+HallName: string
+Date: DateOnly
+Hour: TimeSpan
+CreateDate: DateOnly
+HallApproval: bool
+IsDeeleted: bool
+Create(...): string
+Edit(...): string
-DataCheck(...): string
+Delete()
+IsChaneable(): string
+GetSeatsMetrix(): bool[,]
+UpdateHallApproval(bool)
+ToString(): string
Order
+Id: int
+OccurrenceId: int
+Seats: List<(int, int)>
+Price: int
+CustomerFirstName: string
+CustomerLastName: string
+PhoneNumber: string
+CreditCard: string
+OrderDate: DateOnly
+IsDeleted: bool
+Create(...): string
+Edit(...): string
-DataCheck(...): string
+Delete()
+IsChaneable(): string
+ToString(): string
<<enumeration>>
TicketPricingMethod
Fixed
EarlyBird
LastMinute
Availability
ByDemand
Event
+Id: int
+Name: string
+EventDutation: TimeSpan
+TicketPrice: int
+TicketPricingMethod: PricingMethod
+Merchandise: Dictionary<string, bool>
+Demand: int
+IsDeleted: bool
+Create(...): string
+Edit(...): string
-DataCheck(...): string
+Delete()
+IsChaneable(): string
+UpdateDemand(int)
+ToString(): string
<<interface>>
IDBObject
+ Id: int
+ IsDeleted: bool
+Restore<T>(anObject: T)
+Delete()
+IsChaneable(): string
Hall
+Id: int
+Name: string
+Type: HallType
+Lines: int
+SeatsInLine: int
+City: string
+StreetName: string
+StreetNumber: int
+OpeningHour: TimeSpan
+ClosingHour: TimeSpan
+Facilities: Dictionary <string, bool>
+IsDeleted: bool
+Create(...): string
+Edit(...): string
-DataCheck(...): string
+Delete()
+IsChaneable(): string
+ToString(): string
ListMemento
+IdList: List<int>
+FilterProperty: string
+FilterType: string
+FilterValue: string
RevenuesReportProducer
+Events: List<Event>
+Halls: List<Hall>
+StartDate: DateTime
+EndDate: DateTime
+ShowByEvents: bool
+ShowByHalls: bool
+ShowByDates: bool
+ProduceReport(): Tuple<bool, string>
-CanProduce(): string
-GetPastOccurrencesList(): List<PastOccurrence>
-GetRevenuesString(List<PastOccurrence>): string
PastOccurrence
+Id: int
+EventId: int
+HallId: int
+Date: DateOnly
+TicketsSold: int
+Revenus: int
+IsDeleted: bool
+Create(Occurrence)
+Delete()
+IsChangeable(): string
<<enumeration>>
HallType
Theatre
Traverse
Thrust
User
+Id: int
+FirstName: string
+LastName: string
+City: string
+StreetName: string
+StreetNumber: int
+PhoneNumber: string
+Password: string
+Permissions: Dictionary<string, bool>
+IsDeleted: bool
+Create(...): string
+Edit(...): string
-DataCheck(...): string
+Delete()
+IsChaneable(): string
+ChangePassword(string): string
+ToString(): string
InactiveState
+HandleTimeout()
+HandleActivity()
+ToString(): string
State
#currentUser: CurrentUser
+HandleTimeout()
+HandleActivity()
INotifyPropertyChanged
CurrentUser
- instance: CurrentUser
+User: User
+State: State
+StateCountDown: TimeSpan
+InnerTimer: Timer
+PropertyChanged: event PropertyChangedEventHandler
+GetInstance(User): CurrentUser
+HandleTimeout()
+HandleActivity()
+ResetStateCountDown()
#OnPropertyChanged(string)
ActiveState
+HandleTimeout()
+HandleActivity()
+ToString(): string
LoggedOutState
+HandleTimeout()
+HandleActivity()
+ToString(): string

שכבת ממשק המשתמש

מחלקות Converters

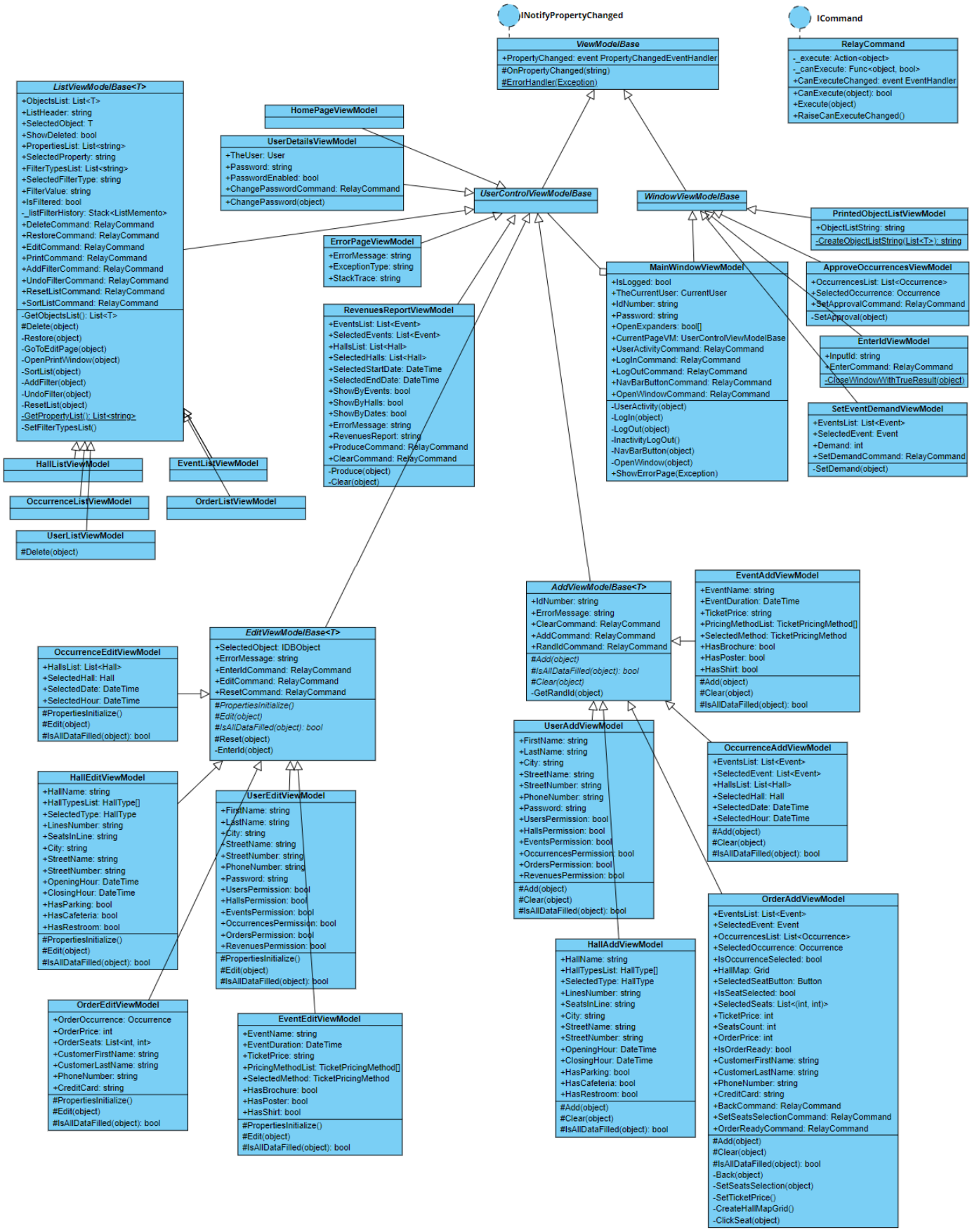


מחלקות HallMap



מחלקות DataContext

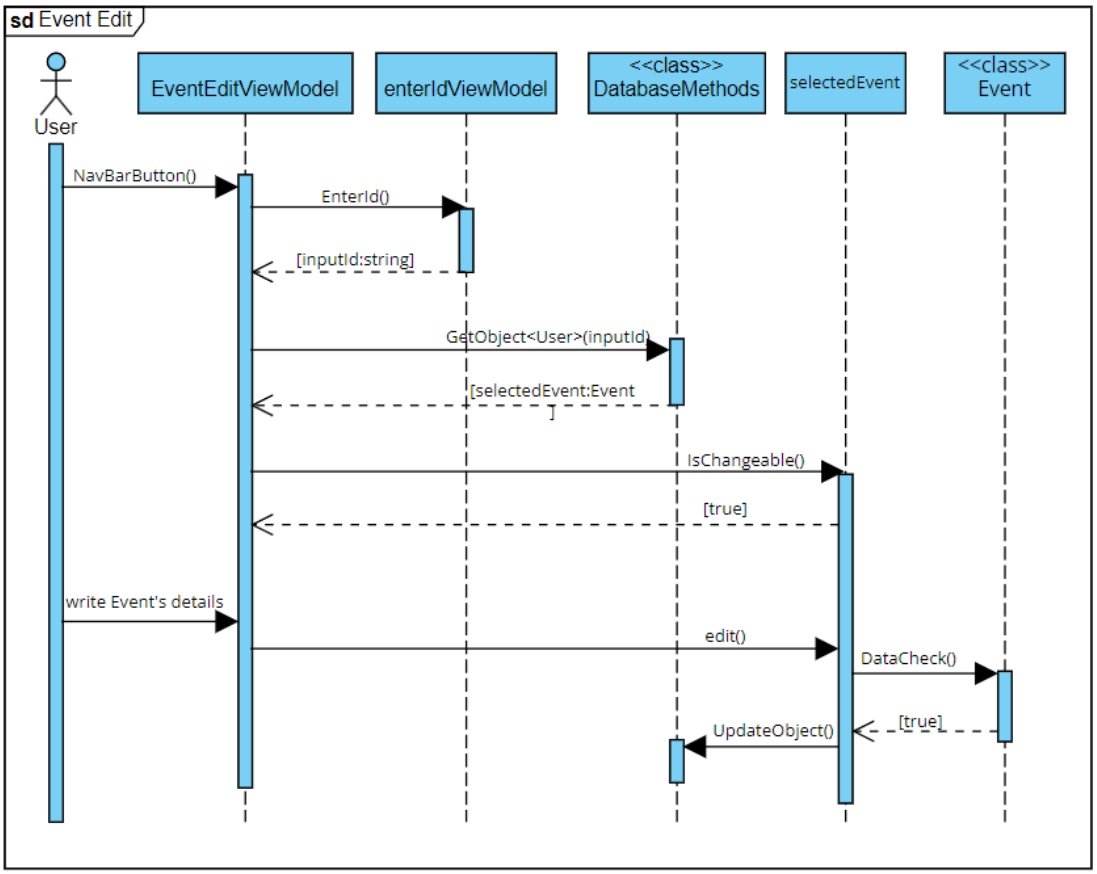
- לכל מחלקה עם תכונה מסוג RealyCommand יש קשר Composition עם המחלקה RealyCommand. לא כללתי קשרים אלו בדיאגרמה כדי לא להעמיס יתר על המידה.



דיאגרמות רצף

עריכת אירוע

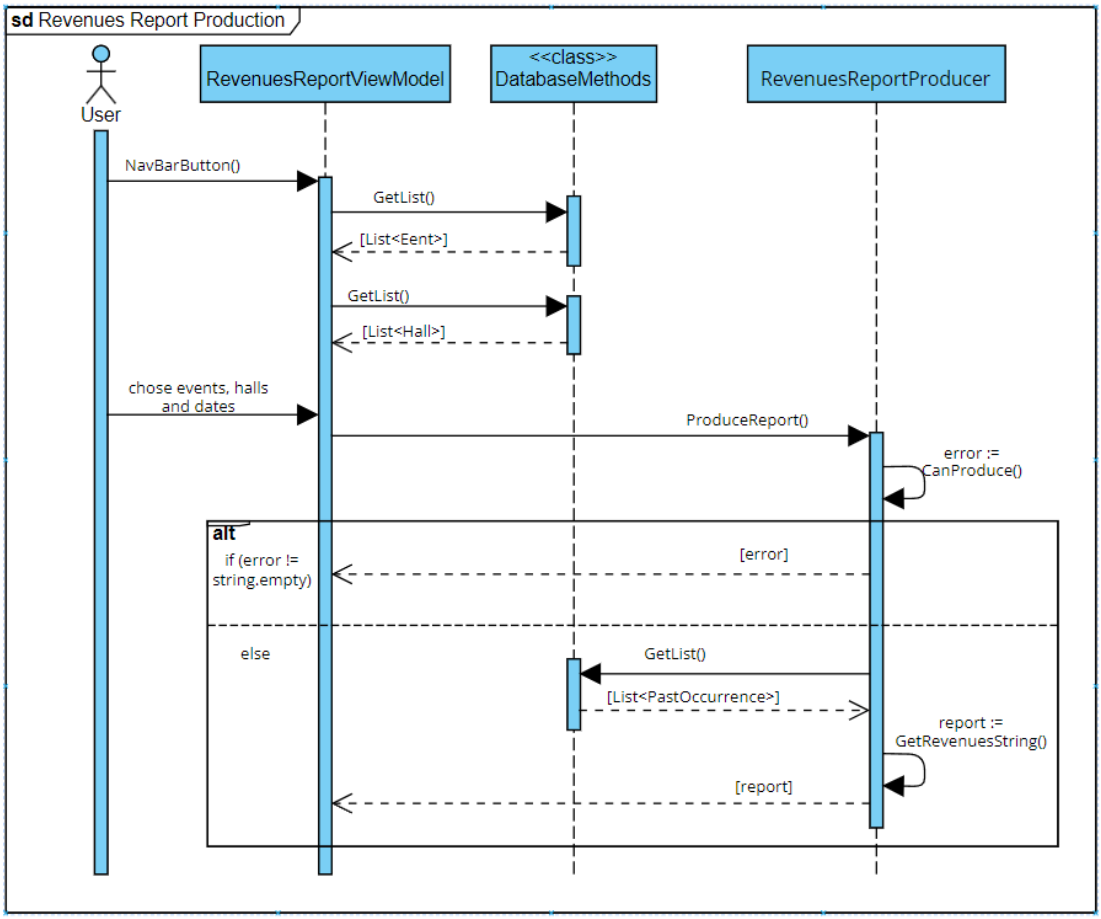
- המשתמש במערכת מעלה דרך תפריט הניווט את עמוד עריכת אירוע.
- בשביל לבחור אירוע לעריכה ייפתח חלון משני ובו המשתמש במערכת יכול להזין מספר זהות, המחרוזת שתוזן מוחזרת לעמוד העריכה.
- עמוד העריכה פונה למחלקה הסטטית המטפלת בגישה לבסיס הנתונים ומביא משם אובייקט-אירוע שמספר הזהות שלו היא המחרוזת שהתקבלה.
- עמוד העריכה בודק שאכן ניתן לשנות את פרטי האירוע שנבחר.
- המשתמש במערכת יוכל לכתוב את פרטיו החדשים של האירוע.
- עמוד העריכה יפעיל את פעולת העריכה על אובייקט-האירוע הנבחר עם הפרטים החדשים שהוכנסו.
- אובייקט-האירוע משתמש במחלקה הסטטית שלו לבדוק שהפרטים שהוכנסו תקינים.
- אם הפרטים תקינים אז אובייקט-האירוע פונה למחלקה הסטטית המטפלת בגישה לבסיס הנתונים ושולח את עצמו אל המתודה שמעדכנת אובייקט בבסיס הנתונים.



הפקת דו"ח הכנסות

- המשתמש במערכת מעלה דרך תפריט הניווט את עמוד הפקת הדו"ח.
- דרך המחלקה הסטטית המטפלת בגישה לבסיס הנתונים שולפים את רשימת האירועים ולאחר מכן את רשימת האולמות שבבסיס הנתונים.
- המשתמש במערכת בוחר את האירועים האולמות והתאריכים עליהם ירצה להפיק את הדו"ח.
- נוצר אובייקט שאחראי על הפקת דו"ח הכנסות עם הפרטים שמולאו ע"י המשתמש.
- האובייקט בודק שכל הפרטים שהוזנו תקינים.
  - אם לא, מוחזרת לעמוד ההפקה הודעת שגיאה
  - אם כן, דרך המחלקה הסטטית המטפלת בגישה לבסיס הנתונים נשלפים כל אובייקטי התרחשות-עבר שבבסיס הנתונים.

מכל האובייקטים האלו שתואמים לפרטים שהזין המשתמש מופק דו"ח הכנסות שמוחזר לעמוד ההפקה.



יצירת הזמנה חדשה

- המשתמש במערכת מעלה דרך תפריט הניווט את עמוד הוספת הזמנה חדשה.
- נעשית קריאה למחלקה סטטית שאחראית לספק מספר זהות פנוי ותקין. היא פונה למחלקה האחראית על גישה לבסיס הנתונים, בשביל לשלוף את רשימת כל ההזמנות שבסיס הנתונים.
- המשתמש בוחר אירוע אליו הוא ירצה ליצור הזמנה ובעקבות כך נשלפת רשימת כל ההתרחשויות של האירוע הזה מבסיס הנתונים.
- מתוך רשימת ההתרחשויות המשתמש בוחר בהתרחשות אליה הוא רוצה ליצור הזמנה.
- נוצר אובייקט לחישוב עלות כרטיס לאירוע. האובייקט מחשב ומחזיר את העלות.
- נוצר אובייקט של מייצר מפת אולם. האובייקט מייצר ומחזיר את מפת האולם.
- המשתמש לוחץ על מושבים במפת האולם, כל לחיצה תסיר או תוסיף את המושב לרשימת המושבים שנבחרו.
- המשתמש רושם את פרטי הלקוח שההזמנה שייכת לו.
- מתודת היצירה של הזמנה חדשה מתבצעת ע"י המחלקה Order שמקבלת את כל הפרטים שהוכנסו/נבחרו ובודקת האם הם כולם תקינים.
- אם הם תקינים נוצר אובייקט הזמנה חדש והוא נשמר בבסיס הנתונים ע"י המחלקה האחראית על גישה לבסיס הנתונים. מחרוזת עם תוצאות הבדיקה חוזרת אל עמוד יצירת ההזמנה.