פרוייקט מלווה

מערכת ניהול טיסות

קורס דוט נט

כללי

<u>תיאור המערכת:</u>

מערכת ניהול טיסות מאפשרת לחברות תעופה (**AirlineCompanies**) לפרסם טיסות וללקוחות ליבחור את הטיסה המתאימה להם ביותר במחיר אטרקטיבי.

<u>שלבי בניית המערכת:</u>

1. בניית בסיס הנתונים וממשקי DAO
2. בניית ממשק WPF לעבודה מול המערכת
3. בניית Rest Services
4. בניית ממשק Web
5. רשות: הטמעת הפרוייקט בענן
6. רשות: בניית ממשק אנדרואיד

<u>גישה למערכת מתחלקת ל- 3 סוגי משתמשים:</u>

1. Administrator – מנהל מערכת
2. AirlineCompany – חברת תעופה המעוניינת לעדכן טיסות השייכות לה
3. Customer – לקוח המעוניין לרכוש טיסה

שלב 1

בניית ליבת המערכת

בניית ליבת המערכת

<u>תיאור:</u>

בשלב זה יוגדר מסד הנתונים לאחסון ושליפת מידע אודות לקוחות, חברות-תעופה וטיסות

יוגדרו 3 Entry Points למערכת עבור כל אחד מסוגי לקוחותיה (אדמיניסטרטור, חברת תעופה או לקוח) אשר יתחברו בביצוע Login.

הפרוייקט יהיה כתוב כ- **Class Library (.NET Framework)** , ולא כ- **Console App** !

הגדרת בסיס הנתונים:

1. **AirlineCompanies** – טבלת חברות תעופה

- ID – LONG, **PK**
- AIRLINE_NAME – STRING
- USER_NAME – STRING
- PASSWORD – STRING
- COUNTRY CODE – LONG, **FK**

2. **Flights** – טבלת טיסות

- ID – LONG, **PK**
- AIRLINECOMPANY_ID – LONG, **FK**
- ORIGIN_COUNTRY_CODE – LONG, **FK**
- DESTINATION_COUNTRY_CODE – LONG, **FK**
- DEPARTURE_TIME – DATE_TIME
- LANDING_TIME – DATE_TIME
- REMAINING_TICKETS – INT

3. **Customers** – טבלת לקוחות

- ID – LONG, **PK**
- FIRST_NAME – STRING
- LAST_NAME – STRING
- USER_NAME – STRING
- PASSWORD – STRING
- ADDRESS – STRING
- PHONE_NO - STRING
- CREDIT_CARD_NUMBER – STRING

4. **Tickets** – טבלת כרטיסים

- ID – LONG, **PK**
- FLIGHT_ID - LONG, **FK**
- CUSTOMER_ID – LONG, **FK**

5. **Countries** – טבלת מדינות

- ID – LONG, PK
- COUNTRY_NAME - STRING

פירוט הטבלאות:

- טבלת **AirlineCompany** מכילה את רשימת חברות התעופה. לכל חברת תעופה יש מפתח מזהה, שם חברה, שם משתמש וסיסמא (לצורך לוגין), וקוד המדינה של החברה. לכל חברת תעופה יש רשימה של טיסות אשר היא מפרסמת בכדי שלקוחות יקנו כרטיסים עבור הטיסה (ראה טבלת **Flights**, וטבלת **Tickets**)

- טבלת **Flights** מכילה את רשימת הטיסות. בפרטי הטיסה קיימים הפרטים הבאים: חברת התעופה אליה היא שייכת, קוד מדינת המקור, קוד מדינת היעד, זמן המראה, זמן נחיתה, מספר כרטיסים שנשארו, סטטוס הטיסה (טרם-המריאה/ באוויר/ נחתה. ראה טבלת **FlightStatus**)

- טבלת **Customer** מכילה את רישמת הלקוחות הקיימים במאגר. לכל לקוח יש מפתח מזהה, שם פרטי, שם משפחה, שם משתמש וסיסמא (לצורך לוגין), כתובת, מספר טלפון ומספר כרטיס אשראי

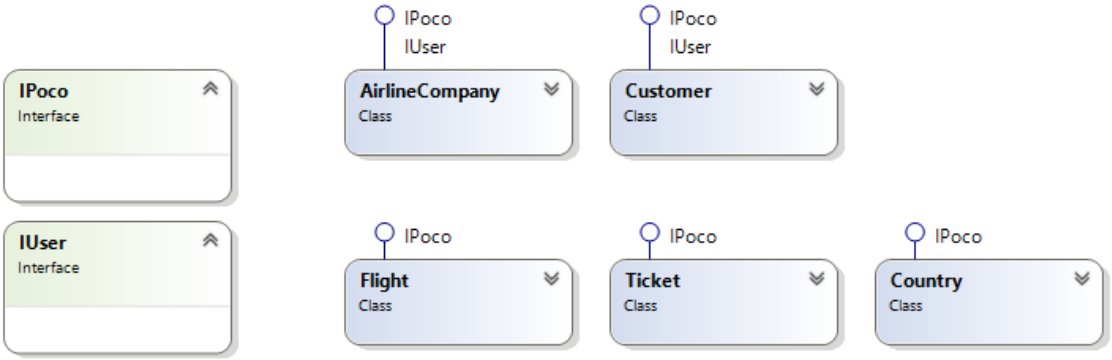
- טבלת **Tickets** מכילה את הכרטיסים שנרכשו עבור הטיסות. לכל כרטיס יש מפתח מזהה, מספר טיסה, מספר לקוח. שים לב שאותו הלקוח אינו יכול לרכוש פעמיים כרטיס לאותה הטיסה

- טבלת **Country** מייצגת את כל המדינות הקיימות במערכת. לכל מדינה יש קוד ושם

עיצוב המערכת

מידע

- להלן דיאגרמת המחלקות:

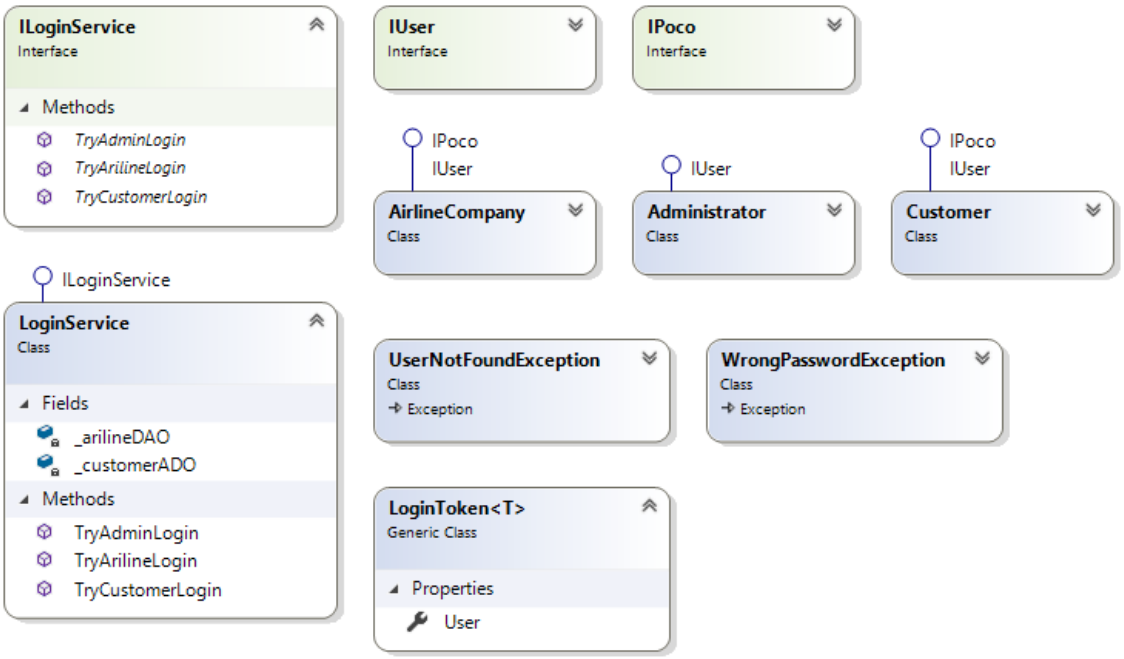


- יש לייצר אינטרפייס (ריק) בשם **IPoco**
  יש לייצר אינטרפייס (ריק) בשם **IUser**

- לכל **טבלה** יש ליצור מחלקת **POCO** :
  - המחלקה תכיל את כל שדות הטבלה
  - המחלקה תכיל בנאי ללא פרמטרים
  - *המלצה: הוסף בנאי המקבל כפרמטרים את כל שדות הטבלה (כולל Id?)
  - המחלקה תממש Equals, ==, !=, GetHashCode אשר ישתמש ב- Id של המחלקה

לוגין

- להלן דיאגרמת המערכת:



- כיצד מתבצע הלוגין?

לקוח אשר מעוניין לעשות לוגין: מקליד שם משתמש + סיסמא. לאחר מכן מופעלות שלושת הפונקציות TryAdminLogin, TryArilineLogin, TryCustomerLogin באופן סידרתי. אם הלוגין מצליח אז הפונקציה מחזירה אמת וה- LoginToken המתאים (המוגדר ב- out) מקבל ערך. אם הלוגין נכשל אז הפונקציה תחזיר שקר וה- LoginToken שיוחזר יהיה null.

כלומר, יש לנסות את שלושת צורות הלוגין עד אשר אחת מהפונקציות תחזיר אמת, או ששלושתם יחזירו שקר. אם שלושת הפונקציות החזירו שקר יש לזרוק אקספשיין UserNotFoundException.

שים לב: אם נמצא המשתמש באחת מהטבלאות (Customer או AirlineCompany) אך הסיסמא **לא נכונה**, יזרק אקספשיין WrongPasswordException.

שים לב שאין טבלת Administrators בשלב זה ולכן יש מנהל מערכת אחד אשר שם המשתמש שלו הוא **admin** וסיסמתו היא **9999**.

שים לב שאין לחזור על שם משתמש- כלומר לא ייתכן שיהיה את אותו שם המשתמש גם בטבלת Customer וגם בטבלת AirlineCompany. כמו כן לא יתכן לייצר משתמש חדש בשם admin. מכיוון ששם משתמש זה תפוס.

- אתגר: צור טבלת Administrators אשר תכיל את מנהלי המערכת

- יש לייצר אינטרפייס בשם ILoginService אשר יגדיר את המתודות של הלוגין וימומש ע"י מחלקת LoginService.

```
bool TryAdminLogin(string userName, string password, out LoginToken<Administrator> token);

bool TryCustomerLogin(string userName, string password, out LoginToken<Customer> token);

bool TryArilineLogin(string userName, string password, out LoginToken<AirlineCompany> token);
```

- מחלקת LoginService תכיל את חיבורי ה- DAO לטבלת לקוח ולטבלת החברות תעופה

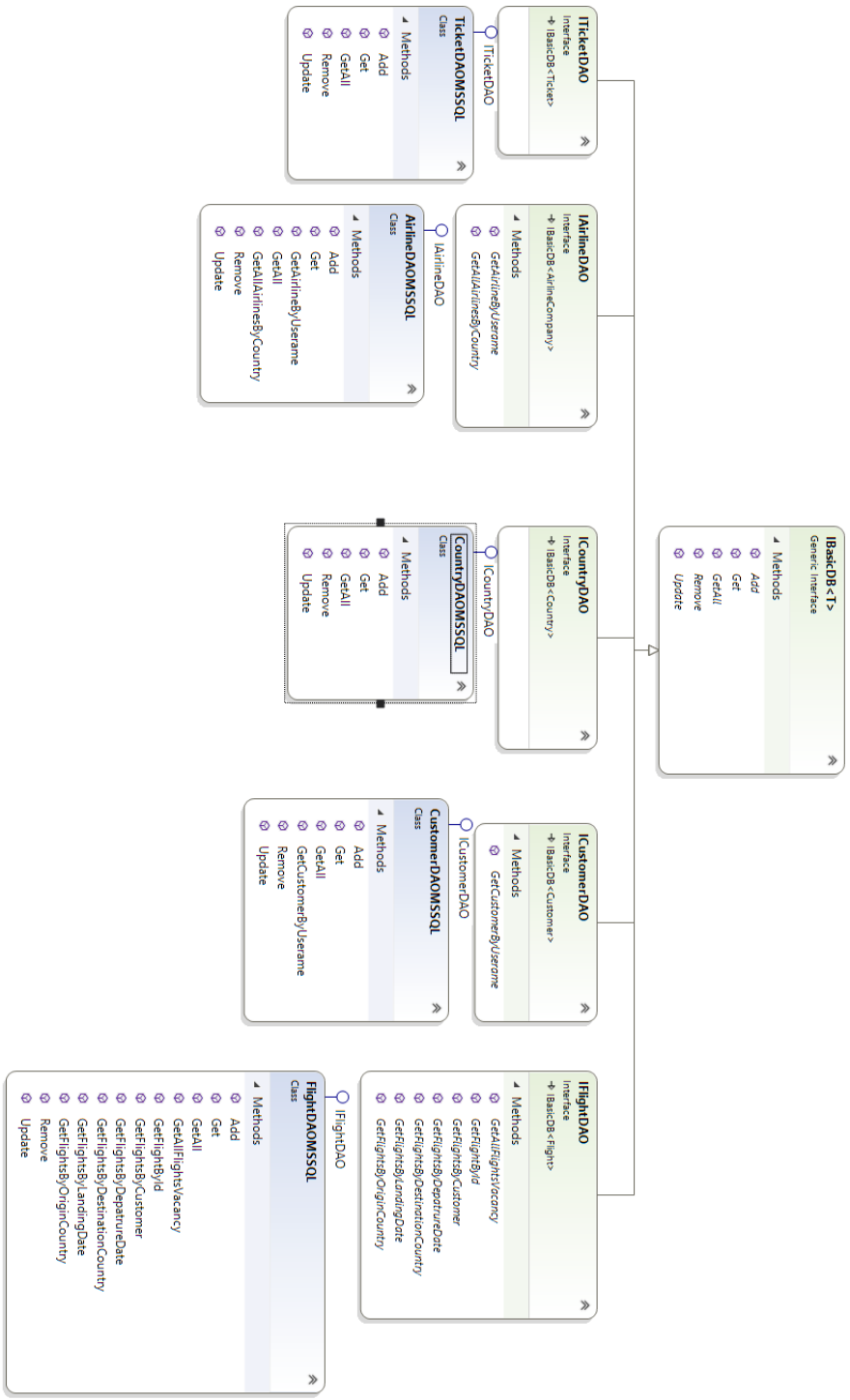
```
private IAirlineDAO _arilineDAO;

private ICustomerDAO _customerADO;
```

- מחלקת LoginToken תשמור את המשתמש ב property

```
public T User { get; set; }
where T : IUser
```

עבודה מול בסיס הנתונים

- להלן דיאגרמת המערכת:



- כל האינטרפייסים של ה- DAO יורשים מאינטרפייס בשם IBasicDB<T>

```
T Get(int id);

IList<T> GetAll();

void Add(T t);

void Remove(T t);

void Update(T t);

where T : IPoco
```

הסיבה לך היא שכל DAO לטבלה בבסיס נתונים חייבים להכיל את האפשרויות הבסיסיות של הבאת מידע, עדכון ומחיקה. מעבר לכך, ייתכנו אינטרפייסים המכילים עוד אפשרויות, לדוגמא ב IAirlineDAO יש אפשרות לקבל חברת תעופה לפי שם-משתמש וכו'. לכל אינטרפייס של DAO קיימת מחלקה יורשת (MSSQL) אשר מממשת את האינטרפייס מול בסיס הנתונים ה- MSSQL

- האינטרפייס ITicketDAO יורש מאינטרפייס IBasicDB<Ticket> והוא אינטרפייס ריק (כרגע) מתוך מחשבה שאם יהיה צורך בעוד פעולות ב DAO אז נגדיר אותם בתוכו (קח בחשבון שניתן להוסיף עוד מתודות ומחלקות בהתאם לצורך...)
המחלקה TicketDAOMSSQL מממשת את אינטרפייס ITicketDAO

- האינטרפייס IAirlineDAO יורש מאינטרפייס IBasicDAO<AirlineCompany> ומכיל את שתי הפונקציות הבאות:

```
AirlineCompany GetAirlineByUserame(string name);

IList<AirlineCompany> GetAllAirlinesByCountry(int countryId);
```

- האינטרפייס ICustomerDAO יורש מאינטרפייס IBasicDAO<Customer> ומכיל את שתי הפונקציה הבאה:

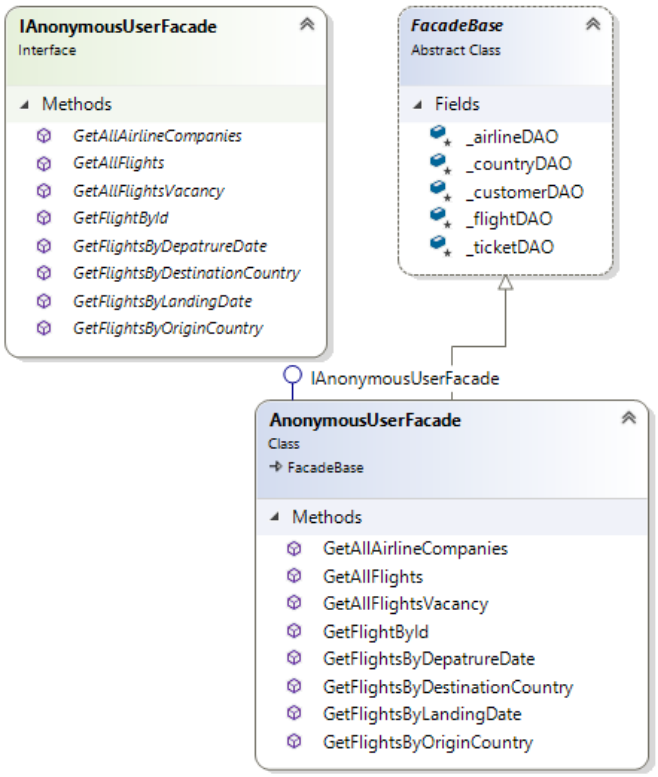
```
Customer GetCustomerByUserame(string name);
```

- האינטרפייס IFlightDAO יורש מאינטרפייס <IBasicDAO<Flight ומכיל את הפונקציות הבאות:

```
Dictionary<Flight, int> GetAllFlightsVacancy();

Flight GetFlightById(int id);

IList<Flight> GetFlightsByOriginCountry(int countryCode);

IList<Flight> GetFlightsByDestinationCountry(int countryCode);

IList<Flight> GetFlightsByDepatrureDate(DateTime departureDate);

IList<Flight> GetFlightsByLandingDate(DateTime landingDate);

IList<Flight> GetFlightsByCustomer(Customer customer);
```

תיאור ה- Business Logic

- להלן דיאגרמת ה- Anonymous Facade:



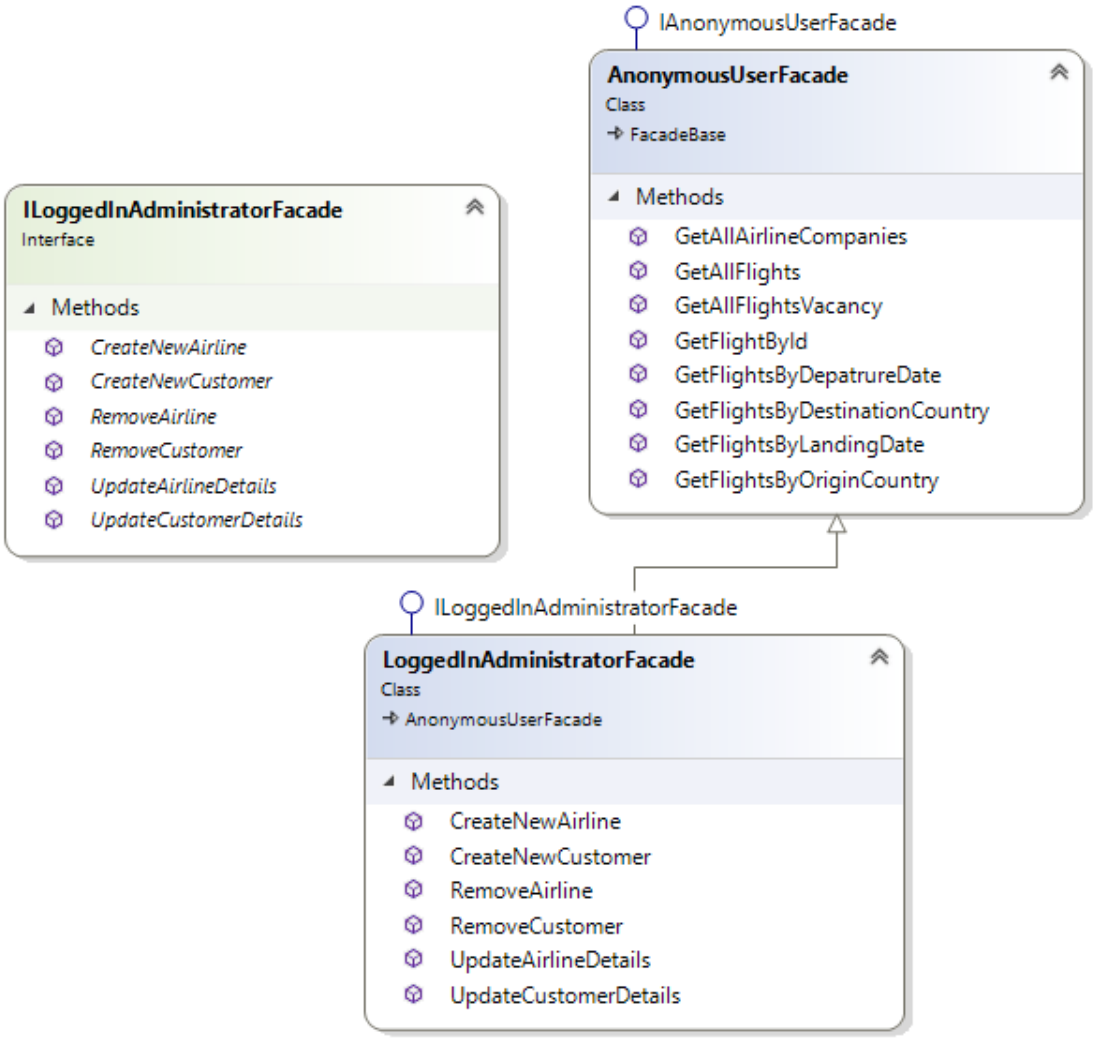
- באינטרפייס IAnonymousUserFacade רשימת האפשרויות החשופות בפני משתמש **"אנונימי"**(כלומר משתמש שלא עשה לוגין). להלן הפונקציות:

```
IList<Flight> GetAllFlights;()

IList<AirlineCompany> GetAllAirlineCompanies;()

Dictionary<Flight, int> GetAllFlightsVacancy;()

Flight GetFlightById(int id);

IList<Flight> GetFlightsByOriginCountry(int countryCode);

IList<Flight> GetFlightsByDestinationCountry(int countryCode);

IList<Flight> GetFlightsByDepatrureDate(DateTime departureDate);

IList<Flight> GetFlightsByLandingDate(DateTime landingDate);
```

- המחלקה האבסטרקטית FacadeBase מכילה כשדות את רשימת כל ה- DAO הקיימים במערכת. כל ה- Facade יורשים ממחלקה זו (בכדי שיקבלו את רשימת ה- DAO).

```
protected IAirlineDAO _airlineDAO;

protected ICountryDAO _countryDAO;

protected ICustomerDAO _customerDAO;

protected IFlightDAO _flightDAO;

protected ITicketDAO _ticketDAO;
```

להלן דיאגרמת ה- LoggedInAdministratorFacade:



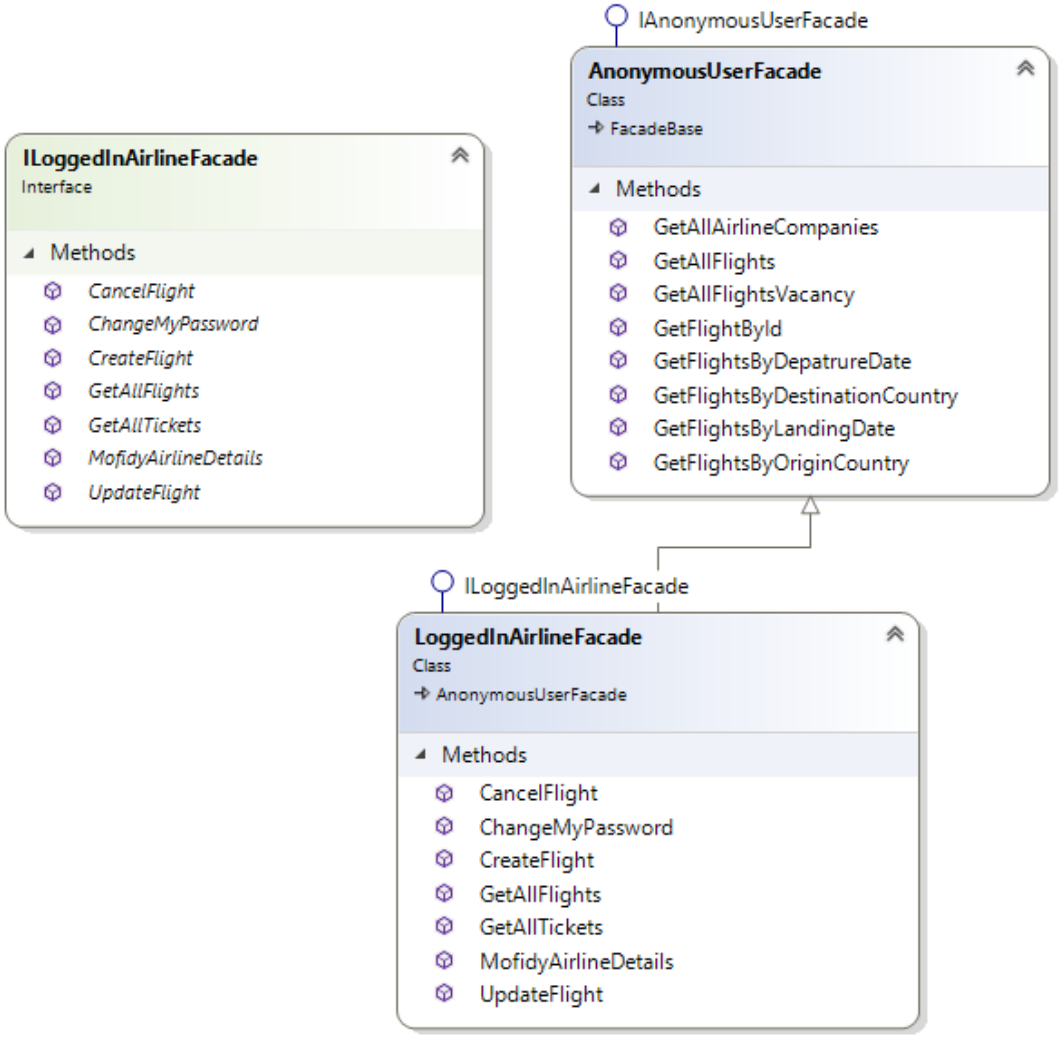
- ה- LoggedInAdministrator Facade מטרתו להכיל את רשימת האפשרויות אשר קיימות עבור משתמש מסוג **מנהל** (לאחר שעשה לוגין בהצלחה). המחלקה יורשת מ- AnonymousUserFacade מכיוון שעבור המנהל קיימות גם כל האפשרויות הקיימות עבור משתמש "אנונימי" (שעדיין לא עשה לוגין). כמו כן קיימות ב- LoggedInAdministrator Facade אפשרויות נוספות אשר רק מנהל שעשה לוגין יכול לקבל. האפשרויות הקיימות למנהל מתוארות באינטרפייס ILoggedInAdministratorFacade בפונקציות הבאות:

```
void CreateNewAirline(LoginToken<Administrator> token, AirlineCompany airline);

void UpdateAirlineDetails(LoginToken<Administrator> token, AirlineCompany customer);

void RemoveAirline(LoginToken<Administrator> token, AirlineCompany airline);
```

```
void CreateNewCustomer(LoginToken<Administrator> token, Customer customer);

void UpdateCustomerDetails(LoginToken<Administrator> token, Customer customer);

void RemoveCustomer(LoginToken<Administrator> token, Customer customer);
```

- שים לב שפונקציות ה- Façade אמורות לפנות למחלקות ה- DAO בכדי לגשת ל- DB

להלן דיאגרמת ה- LoggedInAirlineFacade:



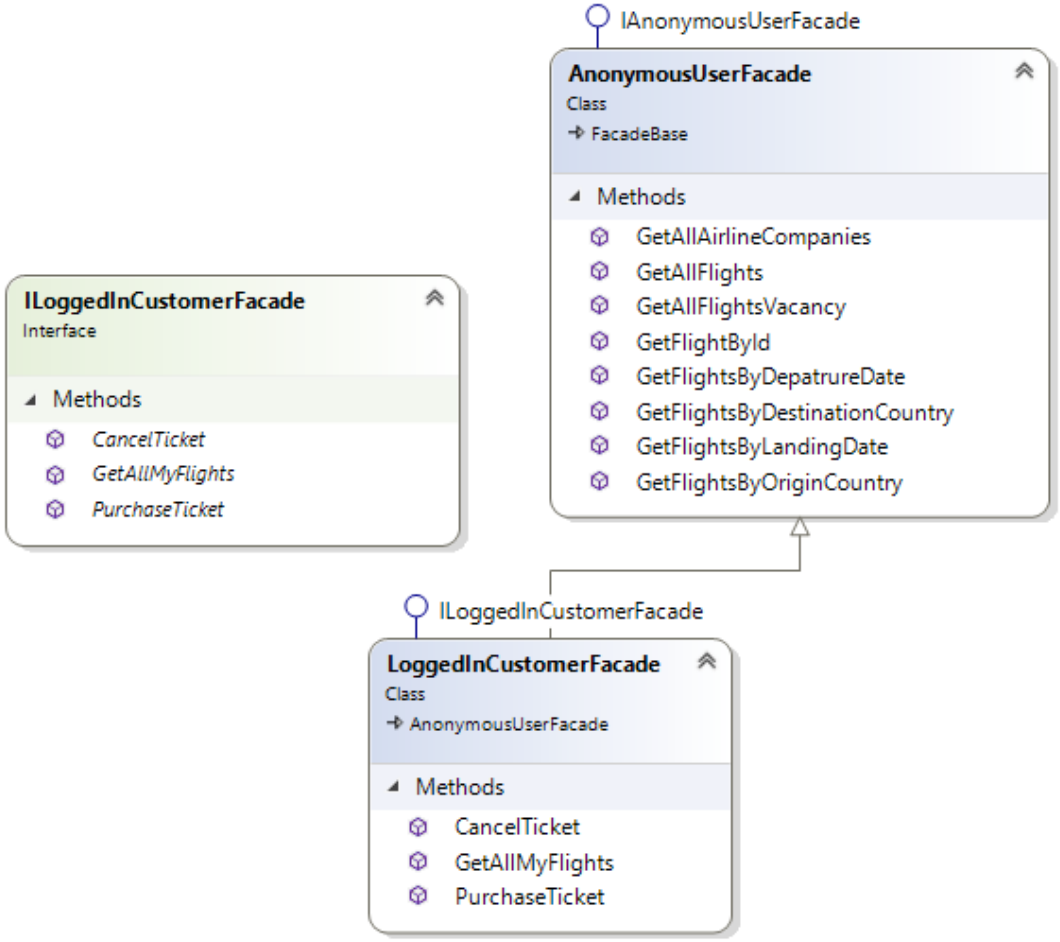
- ה- LoggedsInAirline Facade מטרתו להכיל את רשימת האפשרויות אשר קיימות עבור משתמש מסוג **חברת תעופה** (לאחר שנעשה לוגין בהצלחה). המחלקה יורשת מ- AnonymousUserFacade מכיוון שעבור חברת תעופה קיימות גם כל האפשרויות הקיימות עבור משתמש "אנונימי" (שעדיין לא עשה לוגין). כמו כן קיימות ב-

LoggedsInAirline Facade אפשרויות נוספות אשר רק חברת תעופה שעשתה לוגין יכולה לקבל. האפשרויות הקיימות לחברת התעופה מתוארות באינטרפייס ILoggedInAirlineFacade בפונקציות הבאות:

```
IList<Ticket> GetAllTickets(LoginToken<AirlineCompany> token);

IList<Ticket> GetAllFlights(LoginToken<AirlineCompany> token);

void CancelFlight(LoginToken<AirlineCompany> token, Flight flight);

void CreateFlight(LoginToken<AirlineCompany> token, Flight flight);

void UpdateFlight(LoginToken<AirlineCompany> token, Flight flight);

void ChangeMyPassword(LoginToken<AirlineCompany> token, string oldPassword, string newPassword);

void MofidyAirlineDetails(LoginToken<AirlineCompany> token, AirlineCompany airline);
```

- שים לב שפונקציות ה- Façade אמורות לפנות למחלקות ה- DAO בכדי לגשת ל- DB

להלן דיאגרמת ה- LoggedInCustomerFacade:



- ה- LoggedInCustomer Facade מטרתו להכיל את רשימת האפשרויות אשר קיימות עבור משתמש מסוג **לקוח** (לאחר שנעשה לוגין בהצלחה). המחלקה יורשת מ- AnonymousUserFacade מכיוון שעבור לקוח קיימות גם כל האפשרויות הקיימות עבור משתמש "אנונימי" (שעדיין לא עשה לוגין). כמו כן קיימות ב- LoggedInCustomer Facade אפשרויות נוספות אשר רק חברת תעופה שעשתה לוגין יכולה לקבל. האפשרויות הקיימות ללקוח מתוארות באינטרפייס ILoggedInCustomerFacade בפונקציות הבאות:

```
IList<Flight> GetAllMyFlights(LoginToken<Customer> token);

Ticket PurchaseTicket(LoginToken<Customer> token, Flight flight);

void CancelTicket(LoginToken<Customer> token, Ticket ticket);
```

- שים לב שפונקציות ה- Façade אמורות לפנות למחלקות ה- DAO בכדי לגשת ל- DB

מחלקת FlyingCenterSystem

- יש לכתוב מחלקה זו **כסינגלטון**

- פניות ללוגין יבוצעו אך ורק דרך מחלקה זו

- אין לייצר FAÇADE "לבד", אלא לייצר FACADE רק דרך מחלקה זו, ולכן יש להוסיף פונקציה בשם GetFacade (גנרית?) אשר מחזירה FACADE מוכן. וזאת כדי לשלוט במספר החיבורים הקיימים במערכת בו-זמנית (ניהול החיבורים יתואר בשלב הבא)

- בבנאי של המחלקה יש לייצר Thread אשר "מתעורר" פעם ביום (יש לקנפג את זמן ההתעוררת ב appconig). כאשר ה Thread מתעורר, הוא מעביר את כל הכרטיסים של טיסות שכבר נחתו לפני יותר מ- 3 שעות, לטבלה שנקראת TicketHistory. כמו כן הוא מעביר את כל הטיסות שכבר נחתו לפני יותר מ- 3 שעות, לטבלה שנקראת FlightsHistory

בדיקת המערכת

- יש לכתוב פרוייקט TEST אשר מטרתו לבדוק את תקינות המערכת

- הפרוייקט בתחילתו **ימחק את כל המידע** בבסיס הנתונים (פרט לטבלת Administrator עבור מי שביצע את האתגר)

- ה- TEST יבצע לוגין עבור שלושת סוגי המשתמשים: לקוח, חברה ומנהל. וזאת דרך מחלקת **FlyingCenterSystem** (כמובן ששימוש בפונקציות המשתמש האנונימי אינן מצריכות לוגין)

- ה- TEST יבקש את ארבעת ה- FACADE ממחלקת **FlyingCenterSystem**

- ה- TEST יקרא לכל פעולות ארבעת ה- FACADE (לאחר הלוגין המתאים) ויבדוק את תקינות המערכת. לדוגמא משתמש מנהל יצור חברה חדשה. ולאחר מכן יבדוק שהחברה הוכנסה לבסיס הנתונים בצורה נכונה ע"י כך שיקרא את נתוני החברה וישווה את הנתונים שהוכנסו אל מול הנתונים שהוצאו. כך עבור כל פונקציה-ופונקציה בכל אחד מה FACADE

- כמובן שכל ה"עבודה" תיעשה מול ה- FACADE **ולא באופן ישיר** מול ה- DAO

- פרוייקט גמור הוא פרוייקט שכל ה- TEST כתובים ועוברים בהצלחה

הערות כלליות

- ניתן להוסיף מתודות למחלקות ולאניטרפייסים לפי הצורך
- אם יש צורך בשינוי ה- design הקיים, נא להתייעץ קודם
- יש להוסיף מחלקות של Exception לצורך אירועים חריגים במערכת (לפחות עוד 4 מחלקות)
- יש לכתוב קוד ברור, נקי, מסודר עם הערות
- יש לשמור את ה- ConnectionString בקובץ AppConfig
- יש להעלות את הפרוייקט ל- GIT לצורך בדיקה

בהצלחה