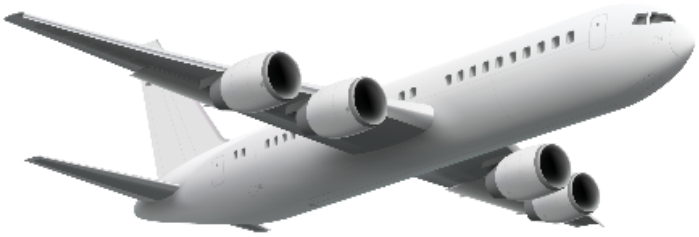
פרויקט מערכת טיסות

קורס פייתון - John Bryce

חלק א' - ליבת המערכת



JOHN BRYCE
תלמד ה'-טק. זה עובד!
a matrix company

תיאור המערכת:

מערכת ניהול טיסות מאפשרת לחברות תעופה (Airline Companies) לפרסם טיסות וללקוחות לבחור את הטיסה המתאימה להם ביותר במחיר אטרקטיבי.

המערכת תכלול בסיס נתונים (Database), שכבת Business Logics, ממשק REST API
וצד לקוח (Front End)

במסמך זה נתאר את השלב הראשון בבניית המערכת.

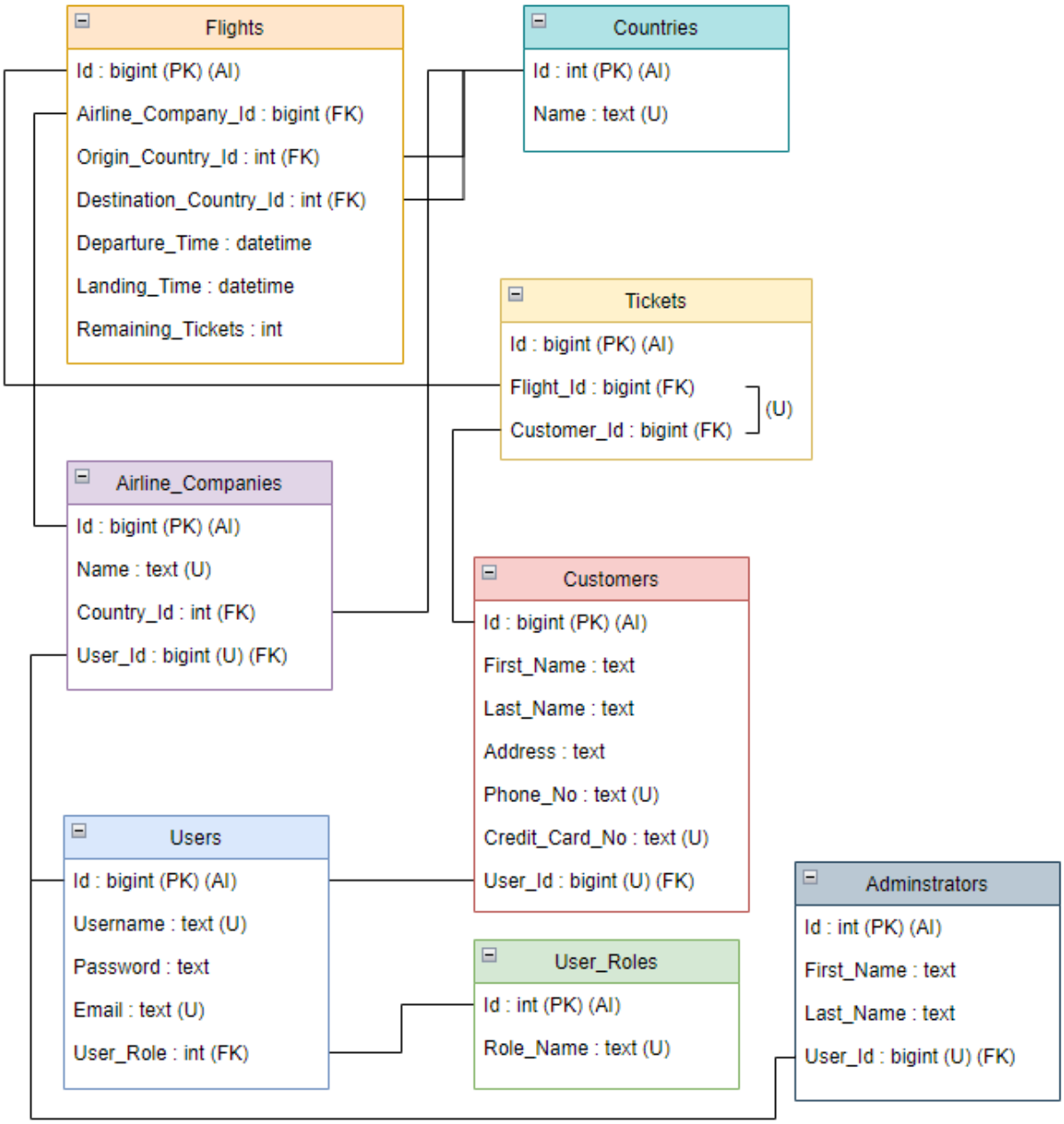
במערכת הטיסות ישנם 4 סוגי משתמשים:

1. Administrator – מנהל מערכת
2. Airline Company – חברת תעופה המעוניינת לעדכן טיסות השייכות לה
3. Customer – לקוח המעוניין לרכוש טיסה
4. Anonymous - גולש אנונימי שטרם נרשם לאתר

שלב 1 - יצירת ה-Database

לצורך הפרויקט, ניצור את הסכמה הבאה:

(הערה: הוסף not null constraint עבור השדות הרלוונטיים)



מקרא:

- PK - Primary Key
- FK - Foreign Key
- AI - Auto Increment
- U - Unique

פירוט הטבלאות:

- טבלת **Airline_Companies** מכילה את רשימת חברות התעופה.
לכל חברת תעופה יש מפתח מזהה (Id), שם (Name), קוד המדינה של החברה (Country_Id) ומס' משתמש (User_Id).
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 הפריטים הבאים: חברת התעופה אליה היא שייכת (Airline_Company_Id), קוד מדינת המקור (Origin_Country_Id), קוד מדינת היעד (Destination_Country_Id), זמן המראה (Departure_Time), זמן נחיתה (Landing_Time) ומספר כרטיסים שנשארו (Remaining_Tickets).

- טבלת **Customers** מכילה את רשימת הלקוחות הקיימים במאגר. לכל לקוח יש מפתח מזהה (Id), שם פרטי (First_Name), שם משפחה (Last_Name), כתובת (Address), מספר טלפון (Phone_No) מספר כרטיס אשראי (Credit_Card_No) ומס' משתמש (User_Id).

- טבלת **Tickets** מכילה את הכרטיסים שנרכשו עבור הטיסות. לכל כרטיס יש מפתח מזהה (Id), מספר טיסה (Flight_Id) ומספר לקוח (Customer_Id).
שים לב שאותו הלקוח אינו יכול לרכוש פעמיים כרטיס לאותה הטיסה, לכן השילוב של מספר הטיסה ומספר הלקוח הוא Unique.

- טבלת **Countries** מייצגת את כל המדינות הקיימות במערכת. לכל מדינה יש מפתח מזהה (Id) ושם (Name). *אתגר: הוסף שדה של תמונה (דגל) לכל מדינה

- טבלת **Administrators** מכילה את כל המנהלים במערכת. לכל מנהל יש מפתח מזהה (Id), שם פרטי (First_Name), שם משפחה (Last_Name) ומס' משתמש (User_Id).

- טבלת **Users** מכילה את כל <u>המשתמשים</u> במערכת.
משתמשים אלו משוייכים ללקוחות, חברות טיסה ומנהלים.
לכל משתמש יש מפתח מזהה (Id), שם משתמש (Username), סיסמא (Password), כתובת אימייל (Email) וסיווג משתמש (User_Role).
השדה User_Role ישמש כדי להבדיל בין לקוחות, חברות טיסה ומנהלים (ראו טבלת User_Roles). *אתגר: הוסף שדה של תמונה (thumbnail) לכל משתמש

- טבלת **User_Roles** מכילה את סיווגי המשתמשים במערכת. לכל סיווג יש מפתח מזהה (Id) ושם סיווג (Role_Name).
<u>לתוך טבלה זו יש להוסיף שלושה סיווגים: לקוח, חברת טיסה ומנהל.</u>

לאחר יצירת כל הטבלאות, נוסיף מספר stored procedures ב-DB (נשתמש בהם בהמשך):

get_airline_by_username(_username text)

פונקציה זו תשמש אותנו בעת ביצוע הלוגין, והיא תחזיר חברת תעופה (airline) לפי שם המשתמש שלה. (רמז - השתמשו ב-join עם טבלת ה-users)

get_customer_by_username(_username text)

פונקציה זו (בדומה לקודמת) תחזיר לקוח (customer) באמצעות שם המשתמש שלו.

get_user_by_username(_username text)

פונקציה זו תאפשר לנו לשלוף יוזרים מן ה-DB בצורה נוחה יותר.

get_flights_by_parameters(_origin_counry_id int, _detination_country_id int, _date date)

פונקציה זו תחזיר את כל הטיסות העונות על הפרמטרים הבאים: מס' מדינת מקור, מס' מדינת יעד ותאריך.

get_flights_by_airline_id(_airline_id bigint)

פונקציה זו תחזיר את כל הטיסות השייכות לחברת התעופה

get_arrival_flights(_country_id int)

הפונקציה תחזיר את כל הטיסות הנוחתות ב-12 שעות הקרובות במדינה שניתנה

get_departure_flights(_country_id int)

הפונקציה תחזיר את כל הטיסות הממריאות ב-12 שעות הקרובות מן במדינה שניתנה

get_tickets_by_customer(_customer_id bigint)

הפונקציה תחזיר את כל כרטיסי הטיסה השייכים ללקוח הנתון

שלב 2 - יצירת הפרויקט וה-Repository

כעת יהיה עלינו ליצור את הפרויקט ב-Pycharm.

בפרויקט יהיה קובץ config ובו נשמור את פרטי הגישה (Connection string) ל-DB שלנו.

בשלב הבא נייצר מחלקת Repository שתטפל בהתממשקות מול ה-DB. לצורך כך, המחלקה תשתמש ב-**Sql-alchemy**, כפי שראינו בכיתה.

לפני כתיבת ה-Repository עצמו, יש ליצור את כל מחלקות ה-Models שבהן נעשה שימוש בפרויקט:

- Flight
- AirlineCompany
- Customer
- Administrator
- User
- Country
- Ticket

במחלקות הנ"ל נייצר שדות המייצגים Columns כפי שלמדנו.
כמו כן, על כל מחלקה לרשת מה-declarative_base של Sql-alchemy.

לאחר שכל המחלקות הללו מוכנות, נתחיל לכתוב את מחלקת ה-Repository.

כיצד יעבוד ה-Repository?
בפרויקט יהיה רק Repo אחד, והוא יהיה אוניברסלי לכל הטבלאות. יהיו בו הפעולות CRUD הבאות:
Get by id, Get All, Add, Update, Add All, Remove.

בנוסף לפעולות ה- CRUD, יש להוסיף את הפונקציות הבאות (תוך שימוש ב- alchemy):

- getAirlinesByCountry(country_id)
- getFlightsByOriginCountryId(country_id)
- getFlightsByDestinationCountryId(country_id)
- getFlightsByDepartureDate(date)
- getFlightsByLandingDate(date)
- getFlightsByCustomer(customer)

לבסוף, ב-Repo נוסיף פונקציות שיקראו ל- stored procedure שיצרנו ב-DB קודם לכן.
- כל פונקציה שתופעל ב- Repository תיכתוב לקובץ לוג תיעוד לפעולה
- יש לדאוג להשתמש ב- try except finally ו/או with בהתאם לצורך

שלב 3 - הוספת ה-Facades

כעת נייצר שכבה בשם Business Logics שתתממשק עם ה-Repository.

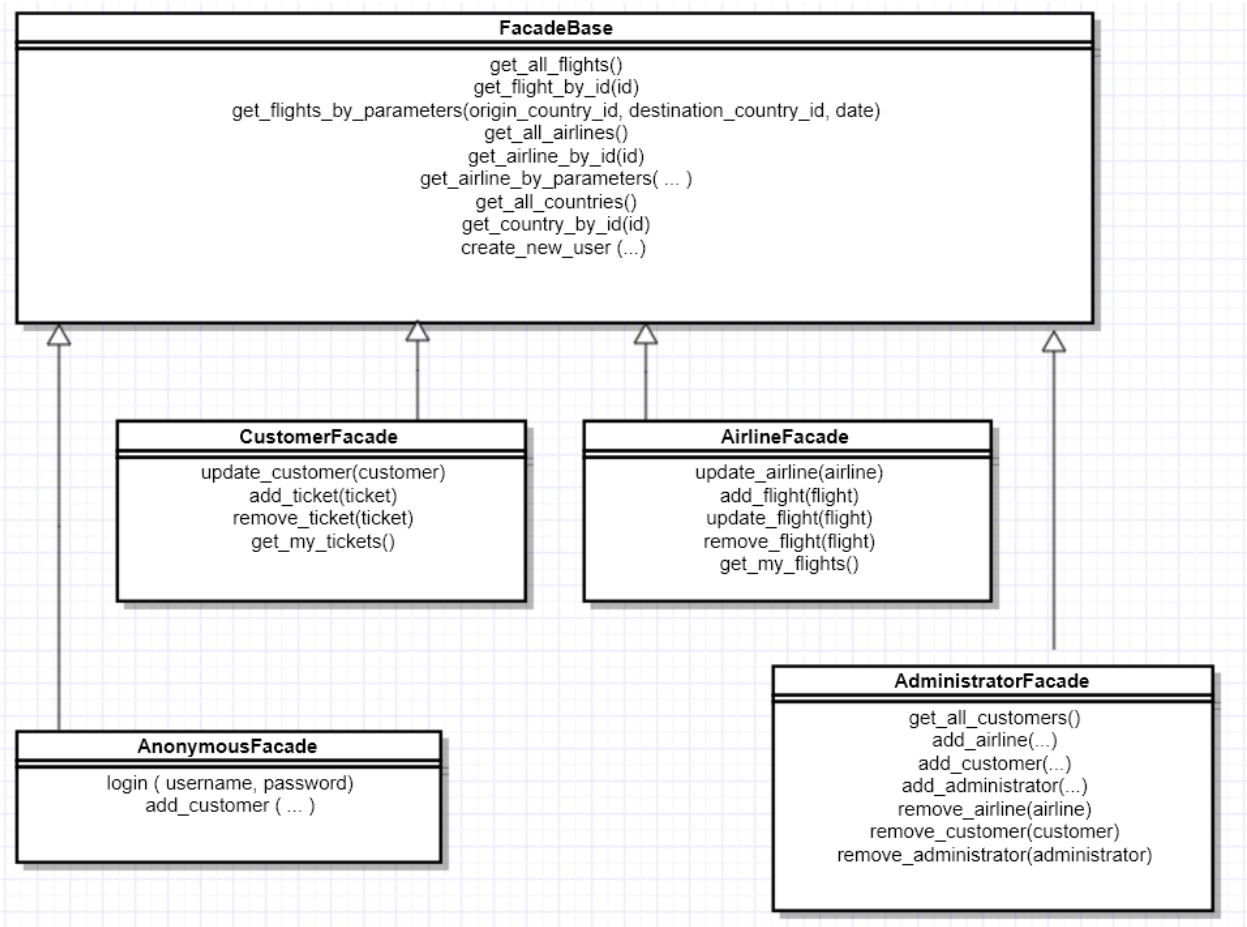
כמחלקת בסיס תהיה לנו מחלקת FacadeBase << **אבסטרקטית >>**. אשר תכיל את הפונקציות הבאות: (המשותפות לכל ה- Facades)

- get_all_flights()
- get_flight_by_id(id)
- get_flights_by_parameters(origin_country_id, destination_country_id, date)
- get_all_airlines()
- get_airline_by_id(id)
- get_airline_by_parameters(…)
- get_all_countries()
- get_country_by_id(id)
- create_new_user (user) - for internal usage

עבור כל סוג משתמש עלינו ליצור מחלקת Facade. אלו יהיו ה-Facades שלנו:

- AnonymousFacade
- CustomerFacade
- AirlineFacade
- AdministratorFacade

ראה דיאגרמה בעמוד הבא ...



<u>הסבר:</u>

מחלקת AnonymousFacade:

(תירש ממחלקת FacadeBase , ובנוסף יהיו בה הפונקציות הבאות)

- login (username, password)
    - After a successful login the function will return the ***<u>correct facade</u>*** [customer, airline or admin] and a ***<u>login-token</u>*** member inside (@getter)
    - Upon failure will return None or raise an error (i.e. WrongPasswordError)
- add_customer (…)

מחלקת CustomerFacade:
(תירש מ-AnonymousFacade, ובנוסף יהיו בה הפונקציות הבאות)

- update_customer (customer)
- add_ticket (ticket)
- remove_ticket (ticket)
- get_my_tickets ()

מחלקת AirlineFacade:
(תירש מ-AnonymousFacade, ובנוסף יהיו בה הפונקציות הבאות)

- get_my_flights ()
- update_airline (airline)
- add_flight (flight)
- update_flight (flight)
- remove_flight (flight)

מחלקת AdministratorFacade:
(תירש מ-AnonymousFacade, ובנוסף יהיו בה הפונקציות הבאות)

- get_all_customers()
- add_airline (...)
- add_customer (...)
- add_administrator (...)
- remove_airline (airline)
- remove_customer (customer)
- remove_administrator (administrator)

כל אחת מן הפונקציות הנ"ל תשתמש ב-Repository כדי לבצע את הפעולות שלה מול ה-DB.

יש לאכוף את המידע המגיע לפונקציות ולדאוג שהערכים יהיו הגיוניים, לפני ביצוע הפעולה.

לדוגמא:

- לא ניתן ליצור טיסה (flight) עם שדה remaining_tickets שלילי (למשל 1-)
- לא ניתן ליצור טיסה שבה זמן הנחיתה מתרחש לפני זמן ההמראה
- לא ניתן ליצור טיסה שבה מדינת המקור ומדינת היעד הן אותה המדינה
- לא ניתן ליצור משתמש (user) עם סיסמא קצרה מ-6 תווים (*רשות)
- איסור על חברת תעופה לערוך טיסה של חברה אחרת
- ועוד...

שלב 4 - Login token

בשלב זה ה-business logic שלנו כמעט מוכן, אך חסר בו רכיב קריטי ביותר.

ה-Login token הינו אובייקט אשר נשתמש בו ב-facades השונים לצורך ביצוע פעולות מורשות (כלומר פעולות שרק משתמשים רשומים יכולים לעשות).
ה- LoginToken יוחזק כשדה בתוך ה Facade (ויועבר ב- init)

שלב א':
ניצור בפרויקט את מחלקת LoginToken ובה יהיו השדות:

- id
- name
- role (customer/airline/admin)

כעת ניגש לפונקציה login שכבר יצרנו ב-AnonymousFacade

הפונקציה תקבל שם משתמש וסיסמא (כפרמטרים) ותבדוק האם הם תואמים למשתמש מסויים. אם כן- הפונקציה תחזיר את ה Facade המתאים עם האובייקט LoginToken (המכיל את פרטי המשתמש) בתוכו

שלב ב':
כעת ניגש לכל הפונקציות ב-facades הבאים:

- CustomerFacade
- AirlineFacade
- AdminFacade

ונוסיף להן שימוש ב- token.

על כל פונקציה לבדוק שה-token תואם את המשתמש בטרם ביצוע הפעולה:

- כלומר במקרה של עדכון/מחיקה- הפונקציה תבדוק שאכן מדובר באותה יישות שהאייטם שייך לה: לדוגמא בפעולת ביטול כרטיס טיסה הפונקציה תבדוק שהכרטיס שייך לאותה לקוח המופיע ב- token
- האם מתאים לאותו FACADE (רשות)

כמו כן, באמצעות ה-token נאכוף את הבדיקה הבאה:

- חברת תעופה לא תוכל למחוק (או לערוך) טיסה שלא שייכת לה

שלב 5 - טסטים (Tests)

כעת עלינו לכתוב טסטים באמצעות Pytest.

הטסטים יבחנו את כל הפעולות המתבצעות על ידי ה-Facades. מטרתנו היא לבחון את התנהגות המערכת שלנו ולוודא שהיא פועלת כפי שתכננו אותה.

אנו נייצר database נפרד לצורך הטסטים, בשם "test_db".
טיפ: מכיוון שאנו עובדים בגישת code first כל שעלינו לעשות הוא לכוון את ה- connection string לדאטא בייס של הטסטים, וכל היתר יקרה אוטומטית (פרט ל stored procedure שאותם יש לכתוב לתוך הדאטא בייס של הטסטים)

בקובץ ה-config של הפרויקט נאחסן את פרטי ההתחברות (Connection string) ל-DB שייצרנו עבור הטסטים.

שימו לב שלדוגמא בטסטים של ה customer facade נצטרך לייצר anonymous facade ולבצע לוגין כדי לקבל את ה- customer facade

דוגמאות לטסטים:

- טסט שמייצר חברת תעופה, ובודק שהיא נוצרה כראוי
- טסט שמייצר טיסה, ומנסה להוסיף כרטיסי טיסה ואז בודק שהם נוספו
- וכו'...

טסטים שמנסים לבצע פעולות אסורות:
- יצירת טיסה עם תאריכים לא הגיוניים, מס' כרטיסים שלילי
- רכישה של כרטיס לטיסה שאזלו לה הכרטיסים
- וכו'...

שים לב: בתחילת ריצה של כל טסט, עליך לאתחל את הנתונים ב- DB.
טיפ: מומלץ לעשות זאת באמצעות stored procedure אשר מוחקת את הרשומות מהטבלאות ואז מוסיפה 2-3 רשומות **לכל טבלה**. כך יהיה תמיד מידע מוכן מראש לטובל כל טסט

<u>הערות:</u>

- יש לכתוב קוד נקי ומסודר ולהקפיד על כללים כגון: שדות המתחילים באות קטנה, מחלקות באות גדולה וכו'
- יש להפריד את המחלקות לקבצים נפרדים
- יש להוסיף שורת תיעוד סטנדרטי לכל פונקציה ומחלקה
- יש להקפיד שכל הטסטים יעברו בהצלחה
- יש לעלות את הקוד ל- GIT ולשלוח קישור
- יש להוסיף את כל שמות הספריות שהשתמשתם בהם לקובץ ה- ***requirements.txt***
- יש לדאוג לקובץ הגדרות config
    - שם תהיה ה- connection string
    - וגם ה LOGLEVEL
- יש לוודא שה- logger הוא Singleton
- יש לבצע מיגרציה של בסיס הנתונים ל- GIT

ב ה צ ל ח ה !