



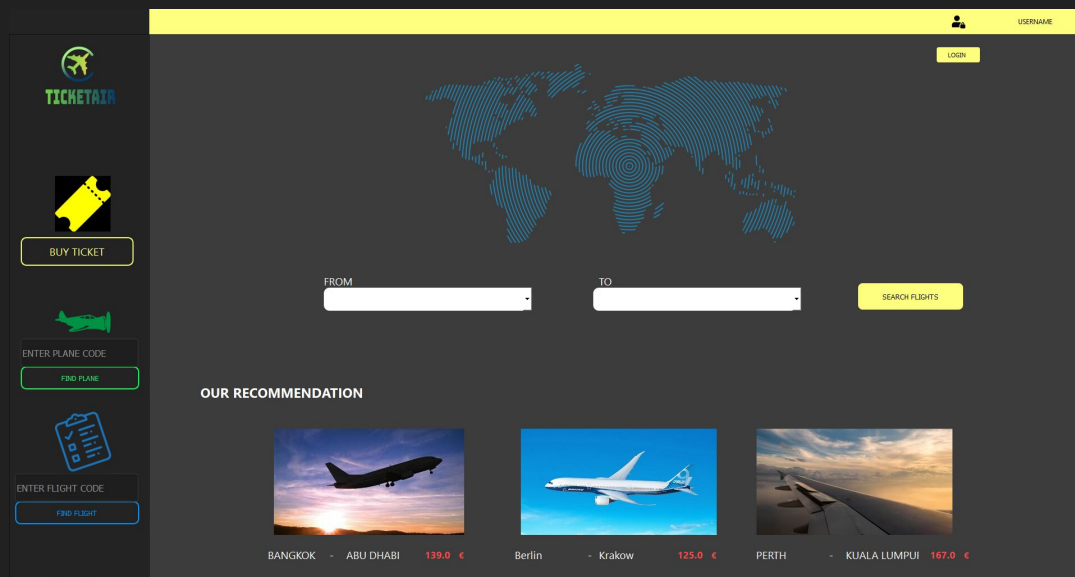
TicketAIR

Anna Janowska, Piotr del Fidali

PLAN PREZENTACJI

- ❑ Co robi nasza aplikacja? – prezentacja działania TicketAIR'a
- ❑ Jak robić GUI za pomocą Qt Designer?
- ❑ Jak przechowywać dane? – czyli kilka słów o bazach danych i języku SQL

JAK DZIAŁA TICKETAIR?



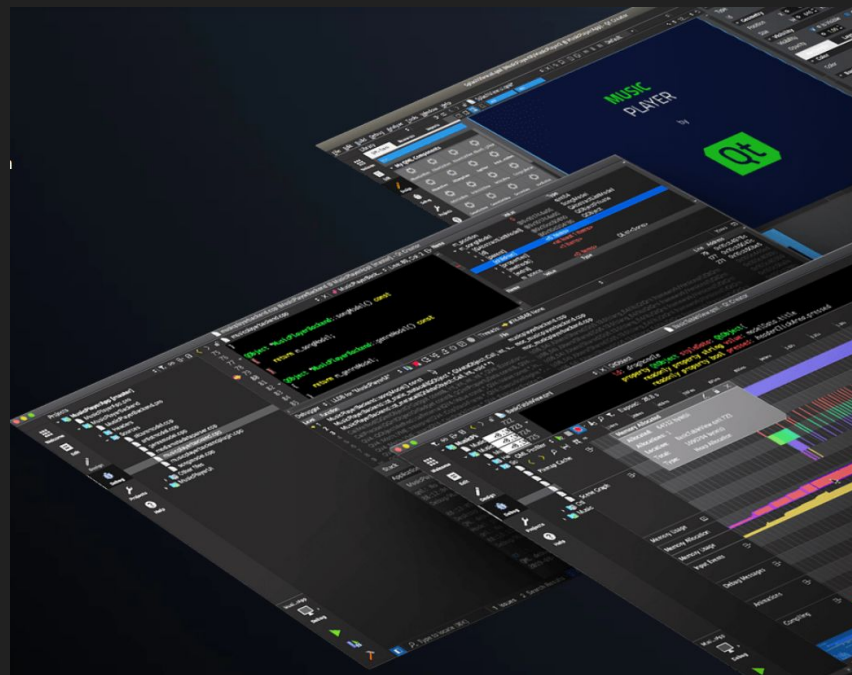
To aplikacja obsługująca rezerwację biletów linii lotniczych. System udostępnia użytkownikowi możliwość przeglądania dostępnych miejsc, zakup biletów oraz uzyskanie informacji dotyczących szczegółów lotu.

CZYM JEST QT DESIGNER?



Wieloplatformowe zintegrowane środowisko do projektowania i budowania interfejsu programów za pomocą widżetów z biblioteki Qt.

Umożliwia tworzenie aplikacji na platformach stacjonarnych, mobilnych i wbudowanych.



JAK ZAINSTALOWAĆ QT DESIGNER?

Windows:

- pobrać ze strony <https://www.qt.io/download>
- użyć komend(dla Pythona 3.6 lub nowszego): `pip install pyqt5`

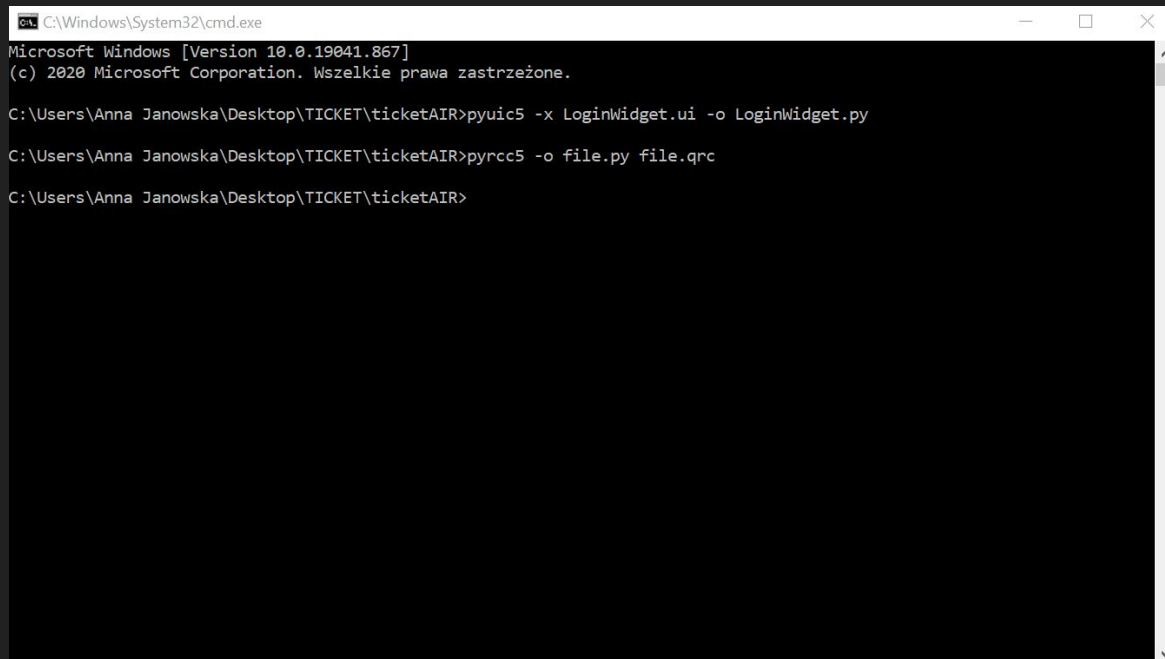
Mac OS X:

- zainstalować https://wiki.qt.io/PySide_Binaries_MacOSX oraz moduł do Python'a przy pomocy komendy: `brew install pyqt`

Linux (Ubuntu/Debian):

- za pomocą komendy `sudo apt-get install python3-pyqt5`

KOMPILACJA DO PYTHONA



```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19041.867]
(c) 2020 Microsoft Corporation. Wszelkie prawa zastrzeżone.

C:\Users\Anna Janowska\Desktop\TICKET\ticketAIR>pyuic5 -x LoginWidget.ui -o LoginWidget.py

C:\Users\Anna Janowska\Desktop\TICKET\ticketAIR>pyrcc5 -o file.py file.qrc

C:\Users\Anna Janowska\Desktop\TICKET\ticketAIR>
```

CZYM SĄ BAZY DANYCH?

- ❑ To uporządkowany zbiór informacji, który posiada własną strukturę
- ❑ Pogrupowanie i podzielenie danych ma za zadanie ułatwić późniejsze pozyskiwanie, przetwarzanie i wprowadzanie informacji
- ❑ Dane są powiązane poprzez relacje
- ❑ Dostęp do danych umożliwia język SQL (Structured Query Language)



STRUCTURED QUERY LANGUAGE (SQL)

SQL - język zapytań wykorzystywany w relacyjnych bazach danych.

Służy do:

- tworzenia baz,
- wstawiania pobierania
- modyfikowania danych.

SQL jest wykorzystywany jedynie do komunikacji z bazą danych, nie można tworzyć w nim kompletnych programów.



SKŁADNIA SQL

Użycie SQL, zgodnie z jego nazwą, polega na zadawaniu zapytań do bazy danych. Zapytania można zaliczyć do jednego z czterech głównych podzbiorów:

- **SQL DML** (ang. Data Manipulation Language – „język manipulacji danymi”)
– INSERT, UPDATE, DELETE
- **SQL DDL** (ang. Data Definition Language – „język definicji danych”)
– CREATE, DROP, ALTER
- **SQL DCL** (ang. Data Control Language – „język kontroli nad danymi”)
– GRANT
- **SQL DQL** (ang. Data Query Language – „język definiowania zapytań”)
– SELECT

NAJCZĘŚCIEJ STOSOWANE BAZY DANYCH

MySQL – baza relacyjna, stosowana w prostych aplikacjach, dobrze skalowalna

PostgreSQL – baza relacyjno-obiektowa, zaawansowana, realizuje skomplikowane procesy analityczne

SQLite – bezserwerowa baza relacyjna, która ma postać biblioteki wbudowanej w aplikację, znajduje się w jednym pliku, ale sprawdza się jedynie przy stronach, które nie generują dużego ruchu

Oracle – relacyjno-obiektowa baza stworzona przez Oracle, przystosowana do obsługi dużych zasobów

MongoDB – zapewnia wysoką wydajność przy bardzo rozbudowanych zasobach danych

SQL LITE W PYTHONIE

PySQLite jest częścią standardowej biblioteki Pythona.

Żeby użyć tego modułu w swoim programie wystarczy zaimportować bibliotekę.

```
1 import sqlite3
```

CONNECTION

- Connection jest obiektem, który reprezentuje bazę danych
- Jako argument podajemy nazwę pliku, w którym będą przechowywane dane
- Jeżeli plik o takiej nazwie nie istnieje to automatycznie zostanie on utworzony
- Kolejnym krokiem jest stworzenie obiektu Cursor, na którym będziemy wykonywać metodę execute() wykonującą komendy SQL
- Jak już skończymy używać połączenia, to należy je zamknąć

```
def addFlight(self, flight):  
    con = sqlite3.connect('ticketair.db')  
    cur = con.cursor()  
    cur.execute('INSERT INTO flights VALUES ' + flight.values())  
    con.commit()  
    con.close()
```

CREATE TABLE – tworzenie tabel

CREATE TABLE tworzy nową tabelę w bazie danych. Pozwala na nadanie nazwy tabeli oraz każdej z jej kolumn

```
def create_tables(self):
    con = sqlite3.connect('ticketair.db')
    cur = con.cursor()
    try:
        cur.execute('''CREATE TABLE flights
                        (origin text, destination text, flightNumber text, planeTailNumber text, price real, time real, capacity integer, sold integer)
                    ''')
    except sqlite3.OperationalError:
        pass
```

INSERT – dodawanie danych

INSERT INTO dodaje do tabeli nowy wiersz. Możemy na 2 sposoby używać tego polecenia. 1. Podajemy nazwy kolumn i wartości. 2. Jeżeli wprowadzamy dane do wszystkich kolumn to wystarczy wypisać wartości.

```
def addFlight(self, flight):
    con = sqlite3.connect('ticketair.db')
    cur = con.cursor()
    cur.execute('INSERT INTO flights VALUES ' + flight.values())
    con.commit()
    con.close()
```

```
class Flight:
    def __init__(self):...

    def __init__(self, origin, destination, flightNumber, plane, date, price, time, sold=0):...

    def values(self):
        return f'("{self.origin}", "{self.destination}", "{self.flightNumber}", "{self.plane.tailNumber}", ' \
            f' "{self.date}", {self.price}, {self.time}, {self.plane.capacity}, {self.sold})'
```

SELECT – wybieranie danych

SELECT zwraca nam dane z tabeli. W celu zawężenia zwracanych danych korzystamy ze słowa kluczowego **WHERE**. Po zastosowaniu funkcji `execute('SELECT ...')` na `Cursorze`, musimy jeszcze wydobyć dane przy pomocy jednej z komend: `fetchone()`, `fetchmany(size=cursor.arraysize)`, `fetchall()`, funkcje te zwracają krotki z danymi.

```
def getPlane(self, tailNumber):  
    con = sqlite3.connect('ticketair.db')  
    cur = con.cursor()  
    cur.execute(f'SELECT * FROM planes WHERE tailNumber = "{tailNumber}"')  
    tailNumber, year, flightMiles, capacity = cur.fetchone()  
    con.close()  
    return Plane(year, flightMiles, capacity, tailNumber)
```

SELECT DISTINCT – wybieranie danych

SELECT DISTINCT pozwala na wybranie unikalnych wartości z danej kolumny.

```
def getDestinations(self, origin=None):
    con = sqlite3.connect('ticketair.db')
    cur = con.cursor()
    if origin == None or origin == '':
        cur.execute(f'SELECT DISTINCT destination FROM flights')
    else:
        cur.execute(f'SELECT DISTINCT destination FROM flights WHERE origin="{origin}"')
    destinations = []
    for dest, in cur.fetchall():
        destinations.append(str(dest))
    return destinations
```


DELETE – usuwanie danych

DELETE to instrukcja, która umożliwia usuwanie wierszy z tabeli. Istnieje możliwość usunięcia wszystkich wierszy bez usuwania tabeli (DELETE FROM table_name).

```
def cancelFlight(self, flightNumber):
    con = sqlite3.connect('ticketair.db')
    cur = con.cursor()
    cur.execute(f"DELETE FROM flights WHERE flightNumber='{flightNumber}'")
    deleted = cur.rowcount
    con.commit()
    con.close()
    return deleted >= 1
```

UPDATE – aktualizacja danych

UPDATE służy do modyfikowania już istniejących rekordów w tabeli.

```
def addTicket(self, user, ticket):
    con = sqlite3.connect('ticketair.db')
    cur = con.cursor()
    cur.execute(
        f"INSERT INTO tickets VALUES ('{ticket.flight.flightNumber}', {ticket.price}, {ticket.places}, {ticket.luggageID})")
    con.commit()
    cur.execute(f"UPDATE users SET ticketsIDs = ticketsIDs || '{cur.lastrowid}' || '|' WHERE login = '{user}'")
    con.commit()
    cur.execute(
        f"UPDATE flights SET sold = sold + {ticket.places} WHERE flightNumber = '{ticket.flight.flightNumber}'")
    con.commit()
    con.close()
```

ZADANIE 1

W pliku TicketAir.py w klasie Database została stworzona funkcja createTables(). Tworzymy w niej tabele lotów, użytkowników i biletów. Zmodyfikuj funkcję w ten sposób, aby została również stworzona tabela samolotów. Zwróć uwagę, żeby wartości miały odpowiedni typ tailNumber (text), year (integer), flightMiles (real), capacity (integer).

ZADANIE 2

W pliku TicketAir.py w klasie Database uzupełnij funkcję addPlane() tak, aby dodawała do tabeli nowy wiersz z odpowiednimi wartościami samolotu. Musisz dokonać połączenia z bazą danych, stworzyć obiekt cursor, wykonać na nim komendę SQL i na koniec zamknąć połączenie. Dodatkowo: Upewnij się, że klasa Plane zwraca wartości: tailNumber, yearOfProduct, flightMiles, capacity oraz że jest zaimportowany moduł sqlite3.

Wskazówka: Sprawdź strukturę funkcji addFlight().

ZADANIE 3

Na razie cena biletu jest zależna jedynie od ilości osób i lotu, jednak w okienku z kupowaniem zostały stworzone dodatkowo radioboxy z rozmiarem bagażu.

Dopisz w pliku `BuyTicketWidget.py` funkcję `updateLuggagePrice()` tak, żeby rodzaj bagażu miał wpływ na cenę końcową biletu.

Uwaga: Bagaż poniżej 15kg powinien zwiększać cenę biletu o 5% a cięższy o 15%.

ŹRÓDŁA:

<https://docs.python.org/3/library/sqlite3.html>

<https://doc.qt.io/qtforpython-5/>

<https://pl.wikipedia.org/wiki/SQLite>

<https://pl.wikipedia.org/wiki/SQL>

<https://www.codecademy.com/articles/sql-commands>