Dokumentacja

Spis treści:

1. Spis użytych technologii
2. Lista plików i opis ich zawartości
3. Kolejność i sposób uruchamiania plików
4. Schemat projektu bazy danych
5. Zależności funkcyjne
6. EKNF
7. Najtrudniejsze podczas realizacji projektu
8. Spis użytych technologii
   1. Python w wersji 3.11.4.
   2. Biblioteki w pythonie, między innymi sqlalchemy. Bardziej szczegółowy spis użytych bibliotek w pliku requirements.txt
   3. Baza danych MariaDB, na serwerze zajęciowym
   4. ERD editor dla wizualizacji zależności między tabelami
9. Lista plików i opis ich zawartości

Pimp my wheels

|

| - data (folder z danymi dla projektu)

| \

| | - parameters (folder ze podstawowymi stałymi)

| | \

| | | - dates.json

| | |

| | | - employees.json

| | |

| | | - services\_parts.json

| |

| | - raw (folder z imionami i nazwiskami osób, przed obróbką)

| | \

| | | - female\_first\_names.csv (imiona żeńskie, przed obróbką)

| | |

| | | - female\_surnames.csv (nazwiska żeńskie, przed obróbką)

| | |

| | | - male\_first\_names.csv (imiona męskie, przed obróbką)

| | |

| | | - male\_surnames.csv (nazwiska męskie, przed obróbką)

| |

| | - brands.csv (marki, modele i ceny samochodów)

| |

| | - equipment.csv(nazwy, typy i ceny rynkowe części samochodów)

| |

| | - female\_surnames.csv(nazwiska żeńskie, po obróbce)

| |

| | - male\_surnames.csv(nazwiska męskie, po obróbce)

| |

| | - names.csv(imiona, po obróbcę)

|

| - src(folder dla podstawowej generacji bazy)

| \

| | - emulation (folder dla emulacji akcji podejmowanych przez osób)

| | \

| | | - \_\_init\_\_.py

| | |

| | | - customer\_decision\_maker.py (akcji podejmowane przez klientów)

| | |

| | | - emulation.py (emulacja jednego dnia pracy)

| | |

| | | - equipment\_generator.py (generator narzędzi)

| | |

| | | - workshop\_decision\_maker.py (akcji dotyczące napraw w warsztatu)

| | |

| | | - workshop\_emulator.py (akcji dotyczące sprzedaży/kupna warsztatu)

| |

| | - generators

| | \

| | | - \_\_init\_\_.py

| | |

| | | - brand\_geneerator.py (generator aut dla tabel)

| | |

| | | - equipment\_generator.py (generator narzędzi dla tabel)

| | |

| | | - personal\_data\_generator.py (generator danych personalnych)

| |

| | - models (folder architektur i zależności w bazie)

| | \

| | | - \_\_init\_\_.py

| | |

| | | - base.py (classa bazowa dla pozostałych tabel w sqlalchemhy)

| | |

| | | - customer.py (tabela klientów)

| | |

| | | - employee.py (tabela pracowników)

| | |

| | | - equipment.py (tabela katalogu narzędzi)

| | |

| | | - inventory.py (tabela urzytych narzędzi)

| | |

| | | - service.py (tabela napraw)

| | |

| | | - transaction.py (tabela transakcji bankowych)

| | |

| | | - vehicle.py (tabela kupionych i sprzedanych pojazdach)

| | |

| | | - workshop.py (tabela warsztatów)

| |

| | - person

| |

| | - utils

| | \

| | | - \_\_init\_\_.py

| | |

| | | - names.py (wczytywanie danych z data/raw)

| |

| | - \_\_init\_\_.py

|

| - venv (wirtualne środowisko pythona)

|

| - Dokumentacja.docx (dokumentacja)

|

| - main.py (plik generujący całą bazę)

|

| - PimpMyWheelsSchema.vuerd.json (wizualizacja zależności i tabel w bazie)

|

| - requirements.txt (spis użytych bibliotek)

1. Kolejność i sposób uruchamiania plików
   1. Na początek należy zainstalować python w wersji 3.11.4. Można to

zrobić na oficjalnej stronie:

<https://www.python.org/downloads/release/python-3114/>

Wraz z pythonem instaluje się pip (administrator bibliotek dla pythona). Można sprawdzić czy się zainstalował za pomocą wpisania polecenia:

pip list

w cmd. Jeżeli pojawia się lista zaistalowanych bibliotek, to wszystko działa, jeżeli nie, można zainstalować administrator anaconda:

<https://www.anaconda.com/download>

3.2 Należy zainstalować wszystkie biblioteki dla projektu. Można to łatwo zrobić za pomocą komendy:

pip install -r requirements.txt

Znajdując się w głównym folderze PimpMyWheels

3.3 Instalujemy Visual Studio Code. Można to zrobić z oficjalnej strony:

<https://code.visualstudio.com/download>

3.4 W Visual Studio Code instalujemy ERD Editor.

3.5 Dalej można zapoznać się z ogólną architekturą bazy w pliku PimpMyWheelsSchema.vuerd.json.

3.6 Ostatnim krokiem wchodzimy do pliku Raport.Rmd i generujemy pdf raporta za pomocą Knittr

3.7 Teraz można otworzyć Raport.pdf

1. Schemat projektu bazy danych i zależności funkcyjne

Wizualizacja schematu projektu bazy danych jest w pliku: PimpMyWheelsSchema.vuerd.json

Poniżej przedstawimy opis każdej tabeli:

* 1. Customers

id(Int, unsigned, not null, autoincrement , primary key)

name(String(25) , not null)

surname(String(25) , not null)

email(String(60) , not null)

phone\_number(String(12) , not null)

birth\_date(Date, not null)

address(String(200) , not null)

account\_creation\_date(Date, not null)

account\_deletion\_date(Date)

last\_active(Date)

* 1. Employees

id (Int, unsigned, not null, autoincrement, primary key)

name(String(25), not null)

surname(String(25), not null)

email(String(60), not null)

phone\_number(String(12), not null)

birth\_date(Date, not null)

address(String(200), not null)

workshop\_id(foreign key("workshops.id"))

position(String(100), not null)

hire\_date(Date, not null)

resignation\_date(Date)

salary(DECIMAL(8,2), unsigned, not null)

* 1. Equipment

id(Int, unsigned, not null, autoincrement, primary key)

name(String(255), not null)

type(String(50), not null)

cost(DECIMAL(8, 2), not null)

* 1. Inventory

id(Int, unsigned, not null, autoincrement, primary key)

equipment\_id(Int, unsigned, not null, foreign key(equipment.id))

service\_id(Int, unsigned, foreign key(services.id))

workshop\_id(Int, unsigned, not null, foreign key(workshops.id))

delivery\_date(Date, not null)

* 1. Services

id(Int, unsigned, not null, autoincrement, primary key)

employee\_id(Int, unsigned, foreign key(employee.id))

start\_date(Date, not null)

end\_date(Date)

work\_cost(DECIMAL(8, 2), not null)

transaction\_id(Int, foreign key(transactions.id))

description(String(100), not null)

vehicle\_id(Int, foreign key(vehicle.id))

* 1. Transactions

id(Int, unsigned, not null, autoincrement, primary key)

transaction\_method(Enum(“cash”, “card”), not null)

other\_party(Int, not null, foreign key(customers.id))

date(Date, not null)

transaction\_type(Enum(“income”, “cost”), not null)

value(DECIMAL(8, 2), not null)

* 1. Vehicles

id(Int, unsigned, not null, autoincrement, primary key)

purchase\_id(Int, unsigned, foreign key(transactions.id))

sale\_id(Int, unsigned, foreign key(transactions.id))

workshop\_id(Int, unsigned, not null , foreign key(workshops.id))

brand(String(15))

model(String(15))

* 1. Workshops

id(Int, unsigned, not null, autoincrement, primary key)

address(String(200), not null)

phone\_number(String(12), not null)

stations\_number(Int, unsigned, not null)

opening\_date(Date, not null)

1. EKNF

Zależności funkcyjne:

1. Customers

id → name, surname, email, phone\_number, birth\_date, address, account\_creation\_date, account\_deletion\_date

1. Employees

id → name, surname, email, phone\_number, birth\_date, address, workshop\_id, hire\_date, resignation\_date

1. Equipment

id → name, type, cost

name → cost

1. Inventory

id → equipment\_id, service\_id, workshop\_id, delivery\_date

1. Services

id → employee\_id, start\_date, end\_date, work\_cost, transaction\_id, description

1. Transactions

id → transaction\_method, other\_party, date, transaction\_type, value

1. Vehicles

id → purchase\_id, sale\_id, workshop\_id, brand, model

1. Workshops

id → address, phone\_number, stations\_number, opening\_date

W tabeli Equipment:

name → cost.

Normalizacja mówi, że należało by rozdzielić tabeli na dwie, aczkolwiek optymalizacja wskazuje, że można zostać przy jednej tabeli.

1. Najtrudniejsze podczas realizacji projektu

Najtrudniejsze podczas realizacji projektu było stworzyć architekturę bazy, oraz napisać kod, wypełniający ją.