PROJEKT NA ZALICZENIA WYKŁADU I LABORATORIUM Z PRZEDMIOTU

„PROJEKTOWANIE SYSTEMÓW BAZ DANYCH”

„Wizyty”

Autor: Jakub Hawro, Paweł Żelazek

Państwowa Wyższa Szkoła Zawodowa w Legnicy
Wydział Nauk Technicznych i Ekonomicznych
Kierunek Informatyka, Rok II, n2PAM1, studia niestacjonarne

OPIS FUNKCJONALNY PROJEKTU

Cel projektu:

Celem projektu jest stworzenie aplikacji bazodanowej, której zadaniem będzie odzwierciedlenie systemu rezerwacji dla gabinetu weterynaryjnego.

Proponowana nazwa:

Wizyty

Opis projektu:

Program ten ma za zadanie rezerwować usługi weterynaryjne dla poszczególnych klientów. Aplikacja umożliwia dodawanie pracowników i klientów, edytowanie ich danych, ustawianie im odpowiednich godzin pracy (dla pracowników) oraz ich usuwanie(wszystkie typowe operację na bazach danych). Poprzez zakładkę usługi mamy możliwość stworzenia odpowiednich usług wykonywanych w przychodni weterynaryjnej, które można połączyć z wybranymi pracownikami. Program umożliwia pracownikom dodawać klientów i rezerwować im usługi wybierając odpowiedni dzień oraz godzinę. Użytkownik ma także możliwość zmiany hasła.

Technologie przewidziane do wykorzystania w projekcie:

Instrukcje, tablice, pętle, funkcje, moduły, wyjątki, obsługa plików, obsługa Python 3.7.5 +, obsługa MySQL,

Zewnętrzne biblioteki

Biblioteka mysql-connector-python,

Biblioteki PyQT5 (QPushButton, QVerticalLayout, QHorizontalLayout, QSQLTable, QLabel, QWidget, QLineEdit, QSortFilterProxyModel, QCalendarWidget, QDateTimeEdit, , QVBoxLayout, QCheckBox, QGroupBox, QHBoxLayout,QSqlTableModel, QSqlRelationalTableModel, QTextEdit, QVBoxLayout, QDialogButtonBox, QIcon, QSqlDatabase, QApplication, QMainWIndow), sys, datetime.

Założenia i realizowane funkcje:

- stworzenie gui operującego na bazie danych MySQL,
- obsługa gui przez użytkownika,
- dodawanie, edytowanie, usuwanie danych o pracownikach, usługach, godzinach pracy, klientach oraz rezerwacjach.

Inne cechy projektu (aplikacji)

W przypadku chęci uruchomienia ze źródeł, należy zainstalować Python co najmniej w wersji 3.7.5 oraz git.

W programie zainicjowałem kilka własnych metod. Oprócz głównej metody main znajdują się poniższe metody:

- query_to_db - funkcja wykonująca jedno zapytanie na bazie danych,
- transaction_to_db - funkcja wykonująca transakcje na bazie danych (czyli kilka query naraz),
- change - metoda edytująca zaznaczone wiersze - Wstawia wartości z wierszy w odpowiednie pola,
- table_init - inicjuje wygląd tabeli,
- if_checked - sprawdza poprawność wprowadzonych danych,
- add - dodaje nowego klienta do bazy danych i odświeża widok,
- modify - modyfikuje bazę danych,
- remove - usuwa klientów z bazy danych,
- searching - wyszukuje po wszystkich kolumnach tabeli,
- initUI - inicjuje UI,
- cancel - metoda wyłącza aplikację,
- login - sprawdza, czy dany użytkownik jest w bazie, po czy zwraca jego ID,
- closeEvent - zapisuje rozmiar i położenie okna podczas zamknięcia,
- hour - Wypełnia godzinami odpowiednie pola,
- changeGroupBox - sprawdza, czy zaznaczono checkbox pracownik,
- employee_type - Sprawdza, czy dana osoba jest pracownikiem, czy nie,

- change - metoda edytująca zaznaczone wiersze - Wstawia wartości z wierszy w odpowiednie pola,
- change_p - metoda ogólna odpowiadająca za dodawanie i usuwanie usług wybranemu pracownikowi,
- center - centruje okno,
- closeEvent - zapisuje rozmiar i położenie okna podczas zamknięcia,
- refresh - odświeża widok tabeli rezerwacji,
- change -metoda edytująca zaznaczone wiersze - Wstawia wartości z wierszy w odpowiednie pola,
- empty - w momencie, gdy wszystkie pola tekstowe są wpisane, włącza przycisk. Sprawdza wypełnienie pól formularza zmiany hasła,
- accept - metoda odpowiedzialna za zmianę hasła,

Wszelkie diagramy będą dostępne pod adresem https://github.com/Lioheart/Weterynarz/tree/master/documentations

Poniżej opis przypadków użycia:

Aby przypisać daną usługę (np. strzyżenie) do pracownika, należy postępować zgodnie z kolejnością poniżej:

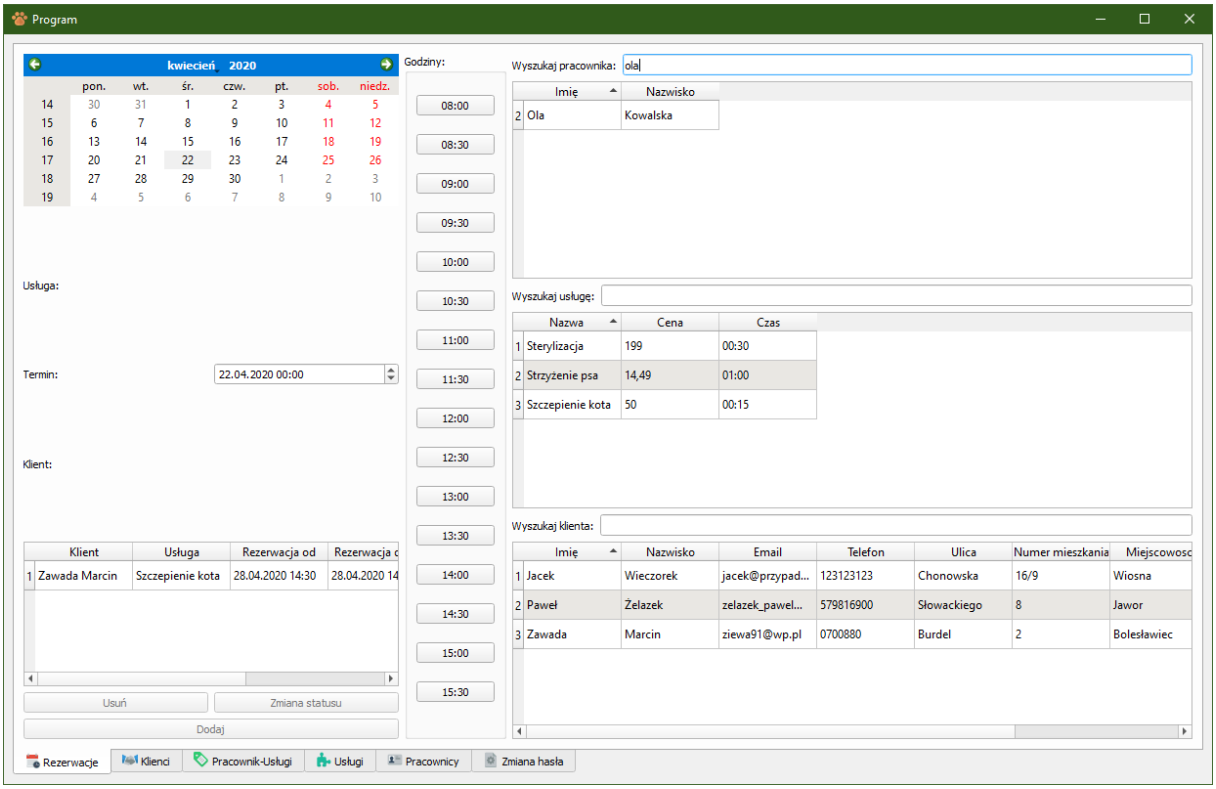
1. Dodaj nowego pracownika. Zaznacz, że dana osoba jest pracownikiem (czyli weterynarzem). Mogą też być to osoby, które tylko sprzątają, albo rejestrują wizyty, jak sekretarki i wtedy nie ma potrzeby tego zaznaczać. Przydziel godziny pracy dla nowego pracownika.
2. Następnie przejdź na zakładkę Usługi i dodaj nowe usługi. Czas podawany jest w formacie hh:mm.
3. Gdy już będziesz miał pracowników i usługi, możesz przejść na zakładkę Pracownik-Usługi i z prawej strony wybrać pracownika. Powinno się wyświetlić jego imię i nazwisko. Następnie w tabeli poniżej kliknij dwukrotnie na usługę. Dana usługa została przydzielona do pracownika.

Aby wykonać rezerwację wizyty danego klienta na określony dzień, należy postępować zgodnie z kolejnością poniżej:

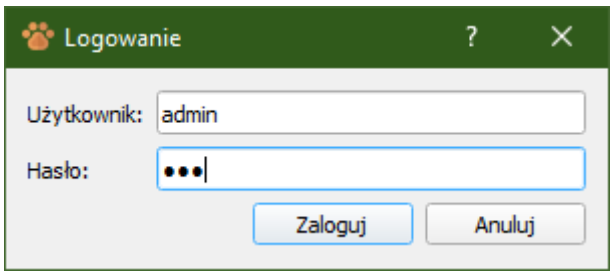
1. Wykonaj poprzednie trzy kroki.
2. Dodaj nowego klienta (jeśli nie ma go w bazie danych).
3. Przejdź na zakładkę rezerwację. Wybierz dzień klikając w niego. Termin powinien się zmienić.
4. Wybierz pracownika. Lista ta będzie zawierała jedynie pracowników, którym przydzieliłeś godziny robocze.
5. Wybierz usługę przypisaną dla danego pracownika.
6. Wybierz klienta.
7. Wybierz godzinę, na którą ma się pojawić klient. Godziny są podzielone na interwały półgodzinne.
8. Poniżej znajduje się lista wszystkich rezerwacji wizyt dla danego pracownika.

Poniżej znajdują się przykładowe zrzuty modelu interfejsu użytkownika:

Zakładka rezerwacji:



Okno logowania:



Fragmenty kodów źródłowych:

Aby uzyskać efekt jak powyżej, tj. ukrytych liter w polu Hasło, należało zastosować bardzo proste ustawienie pola QLineEdit:

```
self.txthaslo.setEchoMode(QLineEdit.Password)
```

Powyższa linia ustawia sposób wyświetlania na typ Password, co powoduje ukrycie wszelkich znaków wpisywanych.

Użytkownik, który próbuje się zalogować, musi znajdować się w bazie. Podczas tworzenia nowego użytkownika login i hasło jest takie same. Należy je zmienić po pierwszym zalogowaniu się użytkownika. Tylko aktualnie zalogowany użytkownik może zmienić swoje hasło, i tylko swoje.

Poniżej przedstawiam sposób, w jaki następuje logowanie:

```
login = self.txtlogin.text()
haslo = self.txthaslo.text()
query = "SELECT * FROM uzytkownik WHERE uzytkownik_nazwa = '{}' AND haslo =
sha('{}');".format(login, haslo)
odczytanie = query_to_db(query)
if odczytanie:
    self.accept()
    return odczytanie[0]
else:
    msg = QMessageBox()
    msg.setIcon(QMessageBox.Warning)
    msg.setText("Błędna nazwa użytkownika lub hasło.")
    msg.setWindowTitle("Błąd logowania")
    msg.exec_()
```

Przypisuję do dwóch zmiennych, login i hasło, dane pobrane z pól o tych samych nazwach. Następnie tworzę zapytanie query, w które wstrzykuję te wartości. Należy zauważyć, że hasło nie jest podawane bezpośrednio, a poprzez funkcję MySQL sha(), hashującą je na zakodowany ciąg znaków. Następnie zapytanie jest wysyłane do bazy danych i jeśli nastąpi odczytanie (czyli znajdzie danego użytkownika), nastąpi akceptacja i zostanie zwrócony pierwszy wynik tego zapytania (czyli numer ID z bazy), w przeciwnym wypadku wyskoczy komunikat o błędnym logowaniu.

W dalszej części możemy spotkać się z sytuacją, gdzie trzeba usunąć danego pracownika. Ale co jeśli dany pracownik ma przypisane usługi, albo jakieś rezerwacje? MySQL nie usunie nam rekordu z danej tabeli, jeśli w innej tabeli istnieje relacja do danego rekordu. I z tym można sobie poradzić:

```
tekst = 'Błąd! Nie można usunąć danego użytkownika!'
query1 = 'DELETE FROM uzytkownik WHERE uzytkownik_id = %s'
val = (self.id_modify,)
query2 = 'DELETE FROM godziny WHERE uzytkownik_id = %s'
query3 = 'DELETE FROM uzytkownik_usluga WHERE uzytkownik_id = %s'
query4 = 'DELETE FROM wizyty WHERE uzytkownik_id = %s'
```

```
if self.id_modify == 1:
    msg = QMessageBox()
    msg.setIcon(QMessageBox.Critical)
    msg.setText('Błąd! Nie można usunąć domyślnego użytkownika')
    msg.setWindowTitle("Popraw dane")
    msg.exec_()
else:
    ret = QMessageBox.question(self, 'Usuwanie użytkownika', "Czy na pewno
chcesz usunąć danego użytkownika?",
                               QMessageBox.Yes | QMessageBox.No,
QMessageBox.No)
    if ret == QMessageBox.Yes:
        if self.if_checked(tekst, [(query4, val), (query3, val), (query2,
val), (query1, val)]):
            msg = QMessageBox()
            msg.setIcon(QMessageBox.Information)
            msg.setText('Użytkownik został usunięty')
            msg.setWindowTitle("Usunięto")
            msg.exec_()
```

Najpierw przypisujemy odpowiednie zapytania usuwające do zmiennych query. Następnie sprawdzamy, czy chcemy usunąć użytkownika o ID 1. Jeśli tak, to wyskoczy nam komunikat o błędzie, gdyż nie można usunąć użytkownika Admin. W przeciwnym wypadku pojawi się komunikat, czy na pewno chcemy usunąć danego użytkownika. Jeśli odpowiedzią jest tak, następuje przesłanie powyższych zapytań wraz z ID danego użytkownika, którego chcemy usunąć oraz tekstem ostrzegawczym, w razie niepowodzenia. Zapytanie to jest realizowane w sposób transakcyjny (czyli wykonywane są po kolei każde z zapytań, by po ostatnim wystąpiła metoda commit). Zapytania te najpierw usuwają nam użytkownika z tabel z relacjami, by w ostateczności usunąć pracownika z głównej tabeli. Po tym wszystkim wyświetlany jest komunikat.

Modyfikacja oraz dodawanie są bardzo podobne. Na przykładzie usług pokażę jak zmodyfikować istniejącą już usługę:

```
tekst = 'Dane zostały błędnie zmodyfikowane.'
query = 'UPDATE uslugi SET nazwa = %s, cena = %s, czas = %s, Opis = %s
WHERE uslugi_id = %s;'
val = (
    self.txt_nazwa.text(),
    self.txt_cena.text().replace(',', '.'),
    self.txt_czas.text(),
    self.txt_opis.toPlainText(),
    self.id_modify
)

if self.if_checked(tekst, query, val):
    msg = QMessageBox(self)
    msg.setIcon(QMessageBox.Information)
    msg.setText('Informacje o usłudze zostały pomyślnie zmodyfikowane')
```

```
    msg.setWindowTitle("Zmodyfikowano usługi")
    msg.exec_()
```

Jak widać, zapytanie te ma wartości, które domyślnie w MySQL nie znaczą nic (albo znaczą coś innego). Chodzi dokładnie o %s. Dzięki temu zapytanie te jest odporne na ataki SQL Injection. Wartości natomiast przechowuję w tupli o nazwie val. Następnie zapytanie wraz z danymi jest przekazywane do metody odpowiedzialnej za komunikację z bazą danych. Gdy zostanie zwrócona prawda, zapytanie wykona się poprawnie i użytkownik otrzyma powiadomienie o udanej modyfikacji.

Natomiast dodawanie wygląda następująco:

```
tekst = 'Nie wprowadzono wszystkich danych'
# Dodanie nowego użytkownika
query = 'INSERT INTO uslugi (nazwa, cena, czas, Opis) VALUES (%s, %s, %s,
%s)'
val = (
    self.txt_nazwa.text(),
    self.txt_cena.text().replace(',', '.'),
    self.txt_czas.text(),
    self.txt_opis.toPlainText()
)

if self.if_checked(tekst, query, val):
    msg = QMessageBox(self)
    msg.setIcon(QMessageBox.Information)
    msg.setText('Dodano nową usługę')
    msg.setWindowTitle("Dodano nową usługę")
    msg.exec_()
else:
    msg = QMessageBox(self)
    msg.setIcon(QMessageBox.Warning)
    msg.setText('Usługa znajduje się już w bazie')
    msg.setWindowTitle("Błąd!")
    msg.exec_()
```

Również tworzymy zapytanie, jednakże tym razem te zapytanie wygląda inaczej. Pojawiła się wartość VALUES, w której zawarte są %s. Dane te zostaną zastąpione przez dane znajdujące się w tupli val. Następnie, tak jak poprzednio, zapytanie wraz z danymi zostaną przesłane do metody komunikującej się z bazą danych i jeśli zostanie poprawnie wykonane, użytkownik otrzyma komunikat. W przeciwnym wypadku, gdy już dana usługa znajduje się w bazie, użytkownik otrzyma komunikat o błędzie a zapytanie nie wykona się.

W momencie, kiedy chcemy pokazać dane z tabeli, która zawiera referencje do kilku innych tabel, musimy uważać, aby te dane się nie zdublowały. Chodzi o to, że MySQL nie wie, że dane tabele mają referencje między sobą. Jeśli byśmy użyli zwykłego zapytania SELECT, To te wartości zostaną zdublowane o tyle razy, o ile występuje ono w danych tabelach, w

których referencje występują. Np. w tabeli wizyty mamy jedną wizytę dla pracownika Ola Kowalska oraz klienta Jacek Przypadek, to te dane zostaną pokazane dla każdego pracownika i każdego klienta. Aby temu zapobiec, należy w zapytanie SELECT zaznaczyć które kolumny zawierają referencję do innych kolumn tabel referencyjnych.

```
query = QSqlQuery(
    'SELECT wizyty.wizyty_id, CONCAT(klienci.imie, " ", klienci.nazwisko)
AS klient, uslugi.nazwa, '
    'wizyty.rezerwacja_od, wizyty.rezerwacja_do, wizyty.status FROM
wizyty,klienci,uslugi WHERE wizyty.klienci_id= klienci.klienci_id AND
wizyty.uslugi_id = uslugi.uslugi_id AND wizyty.rezerwacja_od >
CURRENT_TIMESTAMP AND wizyty.uzytkownik_id = ' + str(
        self.id_pracownik) + ';')
self.model.setQuery(query)
```

W tym przykładzie tworzymy skomplikowane zapytanie, w którym chcemy wyłuskać jakie rezerwacje ma dany pracownik. Musimy zaznaczyć które kolumny mają referencję:

```
WHERE wizyty.klienci_id= klienci.klienci_id AND wizyty.uslugi_id =
uslugi.uslugi_id
```

Wszystko to dzieje się po słowie WHERE. Zaznaczamy, że kolumna klienci_id z tabeli wizyty jest tym samym co kolumna klienci_id z tabeli klienci. Dzięki temu możemy uniknąć zdublowanych wartości pokazywanych w tabeli.

Podsumowanie:

Wszelkie dane, diagramy, powyższa dokumentacja, plik instalacyjny oraz kod źródłowy znajdują się w poniższym repozytorium:

https://github.com/Lioheart/Weterynarz/

Znajduje się tam również instrukcja jak zainstalować program (ze źródeł oraz poprzez pobranie pliku setup.exe).

Baza danych użyta w tym programie znajduje się na zewnętrznych, darmowych serwerach. Program został napisany przy pomocy edytora PyCharm Community Edition.

Aby zalogować się do programu, należy użyć poniższych danych logowania:

Login: Admin
Hasło: 123

Wszelkie grafiki użyte w programie są dostępne bez opłat.