Sprawozdanie

Programowanie w Językach Funkcyjnych

Projekt: Aplikacja wspomagająca prace zespołu SCRUM

Student: Krystian Zborowski

Grupa: WCY21IJ1N1

Prowadząca: mgr inż. Weronika Gramacka

1. **Treść zadania**

Aplikacja desktopowa - PySide6, Aplikacja wspomagająca prace zespołu SCRUM

1. Tworzenie wymagań dla aplikacji w odpowiednim formacie  
2. Tablica zadań (proszę wzorować się na Jira)  
3. Możliwość przeprowadzenia gry planistycznej (planist poker)

4. Możliwość generowania raportów czasu pracy dla pracowników

1. **Realizacja zadania**

Program posiada 3 zakładki które odpowiadają kolejno za:

1 – Dodawanie projektów oraz przypisywanie do nich zadań. Każdemu zadaniu można przypisać status (do zrobienia, w trakcie, wykonane), na tej podstawie zadania umieszczane są w odpowiednich kategoriach. W każdym momencie można dodać oraz usunąć jakieś zadanie. Zadanie tworzy się domyślnie z ilością punktów poker planning ‘0’. Użytkownik może przypisać inną ilość punktów za pomocą przycisków znajdujących się na dole interfejsu.

Obraz zawierający tekst, zrzut ekranu, numer, oprogramowanie

Opis wygenerowany automatycznie

2 – Druga zakładka służy głównie do modyfikacji listy pracowników. Uzytkownik może dodać pracownika, bądź go usunąć z listy. Na samym dole znajduje się możliwość generowania raportów czasu pracy danego pracownika. Raport generuje się na podstawie wartości przypisanych w zakładce nr 3.

Obraz zawierający tekst, zrzut ekranu, wyświetlacz, oprogramowanie

Opis wygenerowany automatycznie

3 – Trzecia zakładka oferuje przypisanie danemu pracownikowi godzin pracy na danym projekcie. Użytkownik może wybrać danego pracownika oraz projekt z listy. Następnie przypisać mu daną ilość godzin pracy w danym dniu kalendarzowym.

Obraz zawierający tekst, zrzut ekranu, numer, Równolegle

Opis wygenerowany automatycznie

Tak wygląda wygenerowany raport:

Obraz zawierający tekst, zrzut ekranu, wyświetlacz, oprogramowanie

Opis wygenerowany automatycznie

Aplikacja opiera swoją pracę na trzech bazach danych: tasks.db, time\_reports.db, employees.db

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, wyświetlacz

Opis wygenerowany automatycznie

Obraz zawierający tekst, zrzut ekranu, Czcionka, numer

Opis wygenerowany automatycznie

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Oprogramowanie multimedialne

Opis wygenerowany automatycznie

Mój kod zawiera wiele funkcji, które odpowiadają za zapisywanie wpisanych danych do bazy danych, do usuwania danych oraz wczytywania tych danych na przykład do list ComboBox.

Odpowiednio:

def wczytaj\_pracownikow\_do\_projektu(self) – odpowiada za wczytywanie wszystkich pracowników z bazy danych employees.db do listy ComboBox w trzeciej zakładce

def wczytaj\_projekty\_do\_projektu(self) – odpowiada za podobne wczytywanie co poparzenia funkcja, tyle, że wczytuje projekty

def dodaj\_pracownika\_do\_projektu(self) – umożliwia zapisanie wiersza w bazie danych time\_reports.db

def dodaj\_projekt(self) – umożliwia dodanie nowego projektu

def dodaj\_zadanie(self) – umożliwia dodanie nowego zadania do bazy danych tasks.db, zadanie automatycznie tworzy się z wartością plannist poker równą 0

def usun\_zadanie(self) – umożliwia usunięcie zaznaczonego obecnie zadania z bazy danych tasks.db

def zmien\_status(self) – umożliwia zmianę statusu (progresu) obecnie zaznaczonego zadania

def wczytaj\_projekty(self) – funkcja wczytująca wszystkie projekty znajdujące się w bazie danych tasks.db, dzięki niej możliwe jest na przykład wybranie projektu do przypisania godzin pracy danemu pracownikowi

def wczytaj\_zadania(self) – funkcja wczytująca zadania, przypisuje ona między innymi zadania do konkretnego statusu i wyświetla w poprawny sposób, dzięki tej funkcji widoczne są wszystkie zadania z bazy danych tasks.db

def handle\_poker\_points(self, button) – funkcja przypisująca wybranemu zadaniu daną ilość punktów w bazie danych tasks.db po kliknięciu danego przycisku w zakładce nr 1

def wczytaj\_pracownikow(self) – funkcja działająca podobnie co wczytywanie zadań i projektów, dzięki niej możliwe jest zobaczenie wszystkich użytkowników dostępnych w bazie danych employees.db

def dodaj\_pracownika(self) – funkcja, która umożliwia dodanie pracownika do bazy danych employees.db

def usun\_pracownika(self) – funkcja, która umożliwia usunięcie pracownika z bazy danych employees.db

def generuj\_raport(self) – funkcja umożliwiająca wygenerowanie raportu na podstawie zaznaczonego przedziału czasowego dla wybranego pracownika, funkcja odczytuje wartości z bazy danych time\_reports.db i wyświetla pożądane dane w wyskakującym okienku MessageBox

def wczytaj\_pracownikow\_raport(self) – funkcja umożliwiająca wyświetlenie wszystkich pracowników z bazy danych employees.db w ComboBox pracownicy w trzeciej zakładce

Kod źródłowy aplikacji:

import sys  
import sqlite3  
  
from PySide6 import QtGui, QtCore  
from PySide6.QtWidgets import (QMainWindow, QWidget, QApplication, QVBoxLayout, QLineEdit, QPushButton, QListWidget, QMessageBox,  
 QHBoxLayout, QLabel, QComboBox, QListWidgetItem, QRadioButton, QButtonGroup, QDateEdit, QTabWidget, QTableWidgetItem, QTableWidget)  
from collections import defaultdict  
  
class MyWidget(QMainWindow):  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 self.setWindowTitle("Pomoc SCRUM")  
  
 self.conn\_tasks = sqlite3.connect("tasks.db")  
 self.cur\_tasks = self.conn\_tasks.cursor()  
  
 self.conn\_employees = sqlite3.connect("employees.db")  
 self.cur\_employees = self.conn\_employees.cursor()  
  
 self.cur\_tasks.execute('''CREATE TABLE IF NOT EXISTS tasks (id INTEGER PRIMARY KEY, name TEXT, project TEXT, status TEXT, poker\_points INTEGER)''')  
 self.conn\_tasks.commit()  
  
 self.cur\_employees.execute('''CREATE TABLE IF NOT EXISTS employees (id INTEGER PRIMARY KEY, imie TEXT, nazwisko TEXT)''')  
 self.conn\_employees.commit()  
  
 self.central\_widget = QTabWidget()  
 self.setCentralWidget(self.central\_widget)  
  
 self.tab1 = QWidget()  
 self.tab2 = QWidget()  
  
 self.central\_widget.addTab(self.tab1, "Zadania")  
 self.central\_widget.addTab(self.tab2, "Zarządzanie pracownikami")  
  
 self.layout1 = QVBoxLayout()  
 self.tab1.setLayout(self.layout1)  
  
 self.layout2 = QVBoxLayout()  
 self.tab2.setLayout(self.layout2)  
  
 self.dodawanie\_projektu = QLineEdit(placeholderText="Dodaj nowy projekt...")  
 self.dodaj\_projekt\_przycisk = QPushButton("Dodaj projekt")  
 self.dodaj\_projekt\_przycisk.clicked.connect(self.dodaj\_projekt)  
  
 self.projekt\_label = QLabel("Projekt:")  
 self.projekt\_combobox = QComboBox()  
 self.projekt\_combobox.currentIndexChanged.connect(self.wczytaj\_zadania)  
 self.projekt\_layout = QHBoxLayout()  
 self.projekt\_layout.addWidget(self.projekt\_label)  
 self.projekt\_layout.addWidget(self.projekt\_combobox)  
  
 self.dodawanie\_zadania = QLineEdit(placeholderText="Dodaj nowe zadanie...")  
 self.dodaj\_zadanie\_przycisk = QPushButton("Dodaj zadanie")  
 self.dodaj\_zadanie\_przycisk.clicked.connect(self.dodaj\_zadanie)  
  
 self.lista\_zadan = QListWidget()  
  
 self.usun\_zadanie\_przycisk = QPushButton('Usuń zadanie')  
 self.usun\_zadanie\_przycisk.clicked.connect(self.usun\_zadanie)  
  
 self.wczytaj\_projekty()  
  
 self.status\_label = QLabel("Status:")  
 self.status\_combobox = QComboBox()  
 self.status\_combobox.addItems(["Do zrobienia", "W trakcie", "Wykonane"])  
 self.status\_combobox.currentIndexChanged.connect(self.zmien\_status)  
 self.status\_layout = QHBoxLayout()  
 self.status\_layout.addWidget(self.status\_label)  
 self.status\_layout.addWidget(self.status\_combobox)  
  
 self.layout1.addWidget(self.dodawanie\_projektu)  
 self.layout1.addWidget(self.dodaj\_projekt\_przycisk)  
 self.layout1.addLayout(self.projekt\_layout)  
 self.layout1.addWidget(self.dodawanie\_zadania)  
 self.layout1.addWidget(self.dodaj\_zadanie\_przycisk)  
 self.layout1.addWidget(self.lista\_zadan)  
 self.layout1.addWidget(self.usun\_zadanie\_przycisk)  
 self.layout1.addLayout(self.status\_layout)  
  
 # Przyciski do pokera  
 self.poker\_planning\_label = QLabel("Poker Planning:")  
 self.poker\_planning\_layout = QHBoxLayout()  
 self.poker\_planning\_layout.addWidget(self.poker\_planning\_label)  
  
 self.poker\_points\_group = QButtonGroup()  
  
 points\_buttons = ["0", "0.5", "1", "2", "3", "5", "8", "13", "20", "40", "100"]  
 for points in points\_buttons:  
 button = QRadioButton(points)  
 self.poker\_points\_group.addButton(button)  
 self.poker\_planning\_layout.addWidget(button)  
  
 self.layout1.addLayout(self.poker\_planning\_layout)  
 self.poker\_points\_group.buttonClicked.connect(self.handle\_poker\_points)  
  
 # Zarządzanie pracownikami  
 self.lista\_pracownikow = QTableWidget()  
 self.lista\_pracownikow.setColumnCount(3)  
 self.lista\_pracownikow.setHorizontalHeaderLabels(["ID", "Imię", "Nazwisko"])  
 self.wczytaj\_pracownikow()  
  
 self.layout2.addWidget(self.lista\_pracownikow)  
  
 self.imie\_input = QLineEdit(placeholderText="Imie pracownika...")  
 self.nazwisko\_input = QLineEdit(placeholderText="Nazwisko pracownika...")  
 dodaj\_pracownika\_przycisk = QPushButton("Dodaj pracownika")  
 dodaj\_pracownika\_przycisk.clicked.connect(self.dodaj\_pracownika)  
  
 self.usun\_pracownika\_przycisk = QPushButton("Usuń pracownika")  
 self.usun\_pracownika\_przycisk.clicked.connect(self.usun\_pracownika)  
  
 self.layout2.addWidget(self.imie\_input)  
 self.layout2.addWidget(self.nazwisko\_input)  
 self.layout2.addWidget(dodaj\_pracownika\_przycisk)  
 self.layout2.addWidget(self.usun\_pracownika\_przycisk)  
  
 # Generowanie raportu  
 self.generuj\_raport\_button = QPushButton("Generuj raport")  
 self.generuj\_raport\_button.clicked.connect(self.generuj\_raport)  
  
 # Pola wyboru pracownika i zakresu dat  
 self.pracownik\_label\_raport = QLabel("Pracownik:")  
 self.pracownik\_combobox\_raport = QComboBox()  
 self.data\_poczatkowa\_label = QLabel("Data początkowa:")  
 self.data\_poczatkowa = QDateEdit()  
 self.data\_koncowa\_label = QLabel("Data końcowa:")  
 self.data\_koncowa = QDateEdit()  
  
 # Układ dla pól wyboru pracownika i zakresu dat oraz przycisku generowania raportu  
 self.raport\_layout = QHBoxLayout()  
 self.raport\_layout.addWidget(self.pracownik\_label\_raport)  
 self.raport\_layout.addWidget(self.pracownik\_combobox\_raport)  
 self.raport\_layout.addWidget(self.data\_poczatkowa\_label)  
 self.raport\_layout.addWidget(self.data\_poczatkowa)  
 self.raport\_layout.addWidget(self.data\_koncowa\_label)  
 self.raport\_layout.addWidget(self.data\_koncowa)  
 self.raport\_layout.addWidget(self.generuj\_raport\_button)  
  
 self.layout2.addLayout(self.raport\_layout)  
  
 self.tab3 = QWidget() # Nowa karta  
 self.central\_widget.addTab(self.tab3, "Przypisywanie pracowników do projektów")  
  
 self.layout3 = QVBoxLayout()  
 self.tab3.setLayout(self.layout3)  
  
 self.pracownik\_label = QLabel("Pracownik:")  
 self.pracownik\_combobox = QComboBox()  
 self.projekt\_label = QLabel("Projekt:")  
 self.projekt\_combobox = QComboBox()  
 self.godziny\_label = QLabel("Godziny pracy:")  
 self.godziny\_input = QLineEdit()  
 self.data\_label = QLabel("Data pracy:")  
 self.data\_edit = QDateEdit()  
  
 self.dodaj\_pracownika\_do\_projektu\_przycisk = QPushButton("Dodaj godziny pracy")  
 self.dodaj\_pracownika\_do\_projektu\_przycisk.clicked.connect(self.dodaj\_pracownika\_do\_projektu)  
  
 self.layout3.addWidget(self.pracownik\_label)  
 self.layout3.addWidget(self.pracownik\_combobox)  
 self.layout3.addWidget(self.projekt\_label)  
 self.layout3.addWidget(self.projekt\_combobox)  
 self.layout3.addWidget(self.godziny\_label)  
 self.layout3.addWidget(self.godziny\_input)  
 self.layout3.addWidget(self.data\_label)  
 self.layout3.addWidget(self.data\_edit)  
 self.layout3.addWidget(self.dodaj\_pracownika\_do\_projektu\_przycisk)  
  
 self.wczytaj\_pracownikow\_do\_projektu()  
 self.wczytaj\_projekty\_do\_projektu()  
  
 self.conn\_time\_reports = sqlite3.connect("time\_reports.db")  
 self.cur\_time\_reports = self.conn\_time\_reports.cursor()  
  
 self.cur\_time\_reports.execute('''CREATE TABLE IF NOT EXISTS time\_reports(id INTEGER PRIMARY KEY, employee TEXT, project TEXT, hours INTEGER, date TEXT)''')  
 self.conn\_time\_reports.commit()  
  
 # Wywołanie funkcji wczytaj\_pracownikow\_do\_projektu w \_\_init\_\_  
  
 self.wczytaj\_pracownikow()  
 self.wczytaj\_pracownikow\_raport()  
 self.wczytaj\_pracownikow\_do\_projektu()  
 self.wczytaj\_projekty\_do\_projektu()  
  
 def wczytaj\_pracownikow\_do\_projektu(self):  
 self.pracownik\_combobox.clear()  
 self.cur\_employees.execute('''SELECT imie, nazwisko FROM employees''')  
 pracownicy = self.cur\_employees.fetchall()  
 for pracownik in pracownicy:  
 imie\_nazwisko = f"{pracownik[0]} {pracownik[1]}"  
 self.pracownik\_combobox.addItem(imie\_nazwisko)  
  
 def wczytaj\_projekty\_do\_projektu(self):  
 self.projekt\_combobox.clear()  
 self.cur\_tasks.execute('''SELECT DISTINCT project FROM tasks''')  
 projekty = self.cur\_tasks.fetchall()  
 for projekt in projekty:  
 self.projekt\_combobox.addItem(projekt[0])  
  
 def dodaj\_pracownika\_do\_projektu(self):  
 pracownik = self.pracownik\_combobox.currentText()  
 projekt = self.projekt\_combobox.currentText()  
 godziny = self.godziny\_input.text()  
 data = self.data\_edit.date().toString("yyyy-MM-dd")  
  
 if pracownik and projekt and godziny and data:  
 self.cur\_time\_reports.execute(  
 '''INSERT INTO time\_reports (employee, project, hours, date) VALUES (?, ?, ?, ?)''',  
 (pracownik, projekt, godziny, data))  
 self.conn\_time\_reports.commit()  
 QMessageBox.information(self, "Sukces", "Pracownik został pomyślnie przypisany do projektu.")  
 self.godziny\_input.clear()  
 self.data\_edit.setDate(QtCore.QDate.currentDate()) # Ustawienie daty na dzisiejszą  
 else:  
 QMessageBox.warning(self, "Błąd", "Wypełnij wszystkie pola.")  
  
 def dodaj\_projekt(self):  
 projekt\_nazwa = self.dodawanie\_projektu.text()  
 if projekt\_nazwa:  
 self.projekt\_combobox.addItem(projekt\_nazwa)  
 self.dodawanie\_projektu.clear()  
 self.wczytaj\_projekty()  
 else:  
 QMessageBox.warning(self, "Błąd", "Nazwa projektu nie może być pusta")  
  
 def dodaj\_zadanie(self):  
 zadanie\_nazwa = self.dodawanie\_zadania.text()  
 projekt = self.projekt\_combobox.currentText()  
 if zadanie\_nazwa:  
 zadanie\_do\_dodania = f"{zadanie\_nazwa} - {projekt}"  
 self.cur\_tasks.execute('''INSERT INTO tasks (name, project, status, poker\_points) VALUES (?, ?, ?, 0)''',  
 (zadanie\_do\_dodania, projekt, "Do zrobienia"))  
 self.conn\_tasks.commit()  
 self.wczytaj\_zadania()  
 self.dodawanie\_zadania.clear()  
 else:  
 QMessageBox.warning(self, "Błąd", "Nazwa zadania nie może być pusta")  
  
 def usun\_zadanie(self):  
 zaznaczone\_zadanie = self.lista\_zadan.currentItem()  
 if zaznaczone\_zadanie is not None:  
 zadanie\_text = zaznaczone\_zadanie.text()  
 zadanie\_parts = zadanie\_text.split(" - ")  
 name = f"{zadanie\_parts[0]} - {zadanie\_parts[1]}"  
 projekt = zadanie\_parts[1]  
 if len(zadanie\_parts) == 3:  
 self.cur\_tasks.execute('''DELETE FROM tasks WHERE name = ? AND project = ?''', (name, projekt))  
 elif len(zadanie\_parts) == 2:  
 self.cur\_tasks.execute('''DELETE FROM tasks WHERE name = ? AND project IS NULL''', (name,))  
 self.conn\_tasks.commit()  
 self.wczytaj\_zadania()  
 self.wczytaj\_projekty()  
 else:  
 QMessageBox.warning(self, "Błąd", "Wybierz zadanie do usunięcia")  
  
 def zmien\_status(self):  
 zaznaczone\_zadanie = self.lista\_zadan.currentItem()  
 if zaznaczone\_zadanie is not None:  
 nowy\_status = self.status\_combobox.currentText()  
 zadanie\_text = zaznaczone\_zadanie.text()  
 zadanie\_parts = zadanie\_text.split(" - ")  
 name = f"{zadanie\_parts[0]} - {zadanie\_parts[1]}"  
 projekt = zadanie\_parts[1]  
 if len(zadanie\_parts) == 3:  
 self.cur\_tasks.execute('''UPDATE tasks SET status = ? WHERE name = ? AND project = ?''', (nowy\_status, name, projekt))  
 elif len(zadanie\_parts) == 2:  
 zadanie\_nazwa = zadanie\_parts[0]  
 self.cur\_tasks.execute('''UPDATE tasks SET status = ? WHERE name = ? AND project IS NULL''', (nowy\_status, name))  
 self.conn\_tasks.commit()  
 self.wczytaj\_zadania()  
  
 def wczytaj\_projekty(self):  
 self.projekt\_combobox.clear()  
 self.cur\_tasks.execute('''SELECT DISTINCT project FROM tasks''')  
 projekty = self.cur\_tasks.fetchall()  
 for projekt in projekty:  
 self.projekt\_combobox.addItem(projekt[0])  
  
 def wczytaj\_zadania(self):  
 self.lista\_zadan.clear()  
 projekt = self.projekt\_combobox.currentText()  
  
 statuses = ["Do zrobienia", "W trakcie", "Wykonane"]  
 for status in statuses:  
 self.cur\_tasks.execute('''SELECT name, poker\_points FROM tasks WHERE project = ? AND status = ?''',  
 (projekt, status))  
 zadania = self.cur\_tasks.fetchall()  
 if zadania:  
 column\_header = QListWidgetItem(status)  
 column\_header.setFlags(column\_header.flags() & ~QtCore.Qt.ItemIsSelectable)  
 column\_header.setBackground(QtGui.QColor("lightgray"))  
 self.lista\_zadan.addItem(column\_header)  
  
 for zadanie in zadania:  
 item = QListWidgetItem(f"{zadanie[0]} - {zadanie[1]}")  
 self.lista\_zadan.addItem(item)  
  
 def handle\_poker\_points(self, button):  
 selected\_item = self.lista\_zadan.currentItem()  
 if selected\_item is not None:  
 selected\_task = selected\_item.text()  
 selected\_task\_parts = selected\_task.split(" - ")  
 name = f"{selected\_task\_parts[0]} - {selected\_task\_parts[1]}"  
 projekt = selected\_task\_parts[1]  
 poker\_points = button.text() if button.isChecked() else "0"  
 self.cur\_tasks.execute('''UPDATE tasks SET poker\_points = ? WHERE name = ? AND project = ?''',  
 (poker\_points, name, projekt))  
 self.conn\_tasks.commit()  
 QMessageBox.information(self, "Poker Planning",  
 f"Zadanie: {selected\_task}\n\nPunkty Poker Planning: {poker\_points}")  
 self.wczytaj\_zadania()  
 else:  
 QMessageBox.warning(self, "Błąd", "Wybierz zadanie")  
  
 def wczytaj\_pracownikow(self):  
 self.lista\_pracownikow.setRowCount(0)  
 self.cur\_employees.execute('''SELECT \* FROM employees''')  
 pracownicy = self.cur\_employees.fetchall()  
 for row, pracownik in enumerate(pracownicy):  
 self.lista\_pracownikow.insertRow(row)  
 for col, dane in enumerate(pracownik):  
 item = QTableWidgetItem(str(dane))  
 self.lista\_pracownikow.setItem(row, col, item)  
  
 def dodaj\_pracownika(self):  
 imie = self.imie\_input.text()  
 nazwisko = self.nazwisko\_input.text()  
 if imie and nazwisko:  
 self.cur\_employees.execute('''INSERT INTO employees (imie, nazwisko) VALUES (?, ?)''', (imie, nazwisko))  
 self.conn\_employees.commit()  
 self.wczytaj\_pracownikow()  
 self.wczytaj\_pracownikow\_raport()  
 else:  
 QMessageBox.warning(self, "Błąd", "Wypełnij wszystkie pola")  
  
 def usun\_pracownika(self):  
 zaznaczony\_wiersz = self.lista\_pracownikow.currentRow()  
 if zaznaczony\_wiersz != -1:  
 id\_pracownika = self.lista\_pracownikow.item(zaznaczony\_wiersz, 0).text()  
 self.cur\_employees.execute('''DELETE FROM employees WHERE id = ?''', (id\_pracownika,))  
 self.conn\_employees.commit()  
 self.wczytaj\_pracownikow()  
 self.wczytaj\_pracownikow\_raport()  
 else:  
 QMessageBox.warning(self, "Błąd", "Wybierz pracownika do usunięcia")  
  
 def generuj\_raport(self):  
 pracownik = self.pracownik\_combobox\_raport.currentText()  
 data\_poczatkowa = self.data\_poczatkowa.date().toString("yyyy-MM-dd")  
 data\_koncowa = self.data\_koncowa.date().toString("yyyy-MM-dd")  
  
 self.cur\_time\_reports.execute(  
 '''SELECT project, hours FROM time\_reports WHERE employee = ? AND date BETWEEN ? AND ?''',  
 (pracownik, data\_poczatkowa, data\_koncowa))  
 raport = self.cur\_time\_reports.fetchall()  
  
 # Tworzenie słownika, w którym kluczem będzie nazwa projektu, a wartością suma godzin przepracowanych na projekcie  
 raport\_projektow = defaultdict(int)  
 for projekt, godziny in raport:  
 raport\_projektow[projekt] += godziny  
  
 # Wyświetlanie raportu  
 raport\_text = f"Raport dla pracownika: {pracownik}\n"  
 raport\_text += f"Data początkowa: {data\_poczatkowa}, Data końcowa: {data\_koncowa}\n\n"  
 raport\_text += "Projekt | Suma godzin\n"  
 raport\_text += "-" \* 30 + "\n"  
 for projekt, suma\_godzin in raport\_projektow.items():  
 raport\_text += f"{projekt} | {suma\_godzin}\n"  
  
 QMessageBox.information(self, "Raport", raport\_text)  
  
 def wczytaj\_pracownikow\_raport(self):  
 self.pracownik\_combobox\_raport.clear()  
 self.cur\_employees.execute('''SELECT imie, nazwisko FROM employees''')  
 pracownicy = self.cur\_employees.fetchall()  
 for pracownik in pracownicy:  
 imie\_nazwisko = f"{pracownik[0]} {pracownik[1]}"  
 self.pracownik\_combobox\_raport.addItem(imie\_nazwisko)  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 app = QApplication(sys.argv)  
 window = MyWidget()  
 window.resize(700, 500)  
 window.show()  
 sys.exit(app.exec())

1. **Podsumowanie**

Aplikacja desktopowa stworzona przy użyciu technologii PySide6 umożliwia tworzenie projektów oraz przypisywanie do nich zadań. Użytkownik aplikacji ma możliwość zmiany statusu dla każdego zadania, co automatycznie zmienia położenie zadania w tabeli. Dodatkowo każde zadanie może mieć przydzieloną daną ilość punktów Poker Planning, zaś po wykonaniu, można usunąć zadanie z bazy danych.

Aplikacja opiera swoje działanie na trzech bazach danych: zadania, czasy pracy oraz pracownicy. Dwie ostatnie bazy danych wykorzystywane są do ostatniego podpunktu zadania projektowego. Użytkownik w każdym momencie może dodać lub usunąć pracownika z bazy danych. Bazując na wartościach z baz danych zadania oraz pracownicy można utworzyć wiersz w trzeciej bazie danych, mianowicie każdemu pracownikowi można przypisać czas pracy na danym projekcie w danym dniu. Na podstawie wierszy z bazy danych time\_reports można stworzyć raport pracy dla pracownika w określonym przez użytkownika okresie.