**Создание оконного приложения для редактирования расписания**

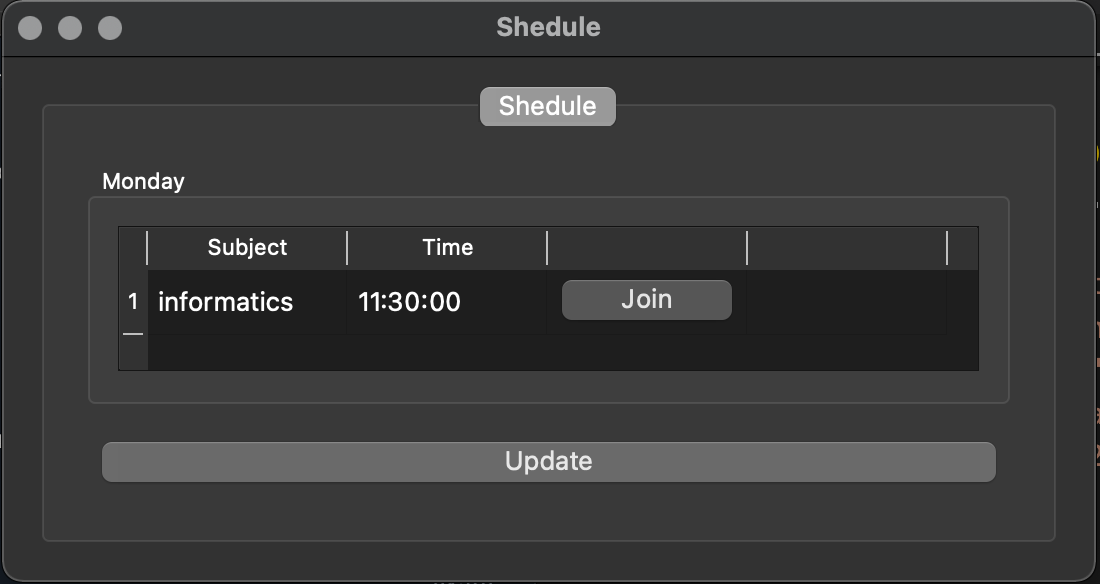
**Техническое задание:**

* Создать оконное приложение позволяющее редактировать базу данных с расписанием Вашей группы.

**Минимальные требования к разрабатываемой системе:**

1. Использование библиотеки PyQt5.
2. Использование адаптера psycopg2.
3. Приложение должно иметь при себе функционал позволяющий: просматривать базу данных в удобном для пользователя формате, удалять, добавлять и изменять записи в этой же базе данных.
4. Визуальная часть должна иметь при себе:
   1. Минимум 3 вкладки, в каждой из которых содержится информация из отдельной таблицы в базе данных.
   2. Внутри каждой вкладки информация должна отображаться в виде таблиц.
   3. Внутри каждой вкладки должна отображаться кнопка с обновлением информации.
   4. Внутри каждой таблицы должны отображаться все поля из таблицы в базе данных в виде колонок
   5. Внутри каждой таблицы после каждой строки записи должны быть отображены кнопки изменения и удаления записи
   6. В конце каждой таблицы должна находиться пустая строка с кнопкой для добавления новой записи.

**Пример частичной реализации:**

****

* **Импортируем необходимые библиотеки и адаптеры**

import psycopg2

import sys

from PyQt5.QtWidgets import (QApplication, QWidget,

QTabWidget, QAbstractScrollArea,

QVBoxLayout, QHBoxLayout,

QTableWidget, QGroupBox,

QTableWidgetItem, QPushButton, QMessageBox)

* **Создаем класс MainWindow с конструктором**

class MainWindow(QWidget):

def \_\_init\_\_(self):

super(MainWindow, self).\_\_init\_\_()

self.\_connect\_to\_db()

self.setWindowTitle("Shedule")

self.vbox = QVBoxLayout(self)

self.tabs = QTabWidget(self)

self.vbox.addWidget(self.tabs)

self.\_create\_shedule\_tab()

Класс QTabWidget создает структуру, которую можно заполнять вкладками.

Вкладки это подстраницы в окне приложения. Аналогом вкладок в оконных приложениях являются вкладки в веб-браузере.

* **Создаем метод для подключения к базе данных**

def \_connect\_to\_db(self):

self.conn = psycopg2.connect(database="<название вашей базы данных>",

user="postgres",

password="1234",

host="localhost",

port="5432")

self.cursor = self.conn.cursor()

* **Создаем метод для отображения вкладки с расписанием**

def \_create\_shedule\_tab(self):

self.shedule\_tab = QWidget()

self.tabs.addTab(self.shedule\_tab, "Shedule")

self.monday\_gbox = QGroupBox("Monday")

self.svbox = QVBoxLayout()

self.shbox1 = QHBoxLayout()

self.shbox2 = QHBoxLayout()

self.svbox.addLayout(self.shbox1)

self.svbox.addLayout(self.shbox2)

self.shbox1.addWidget(self.monday\_gbox)

self.\_create\_monday\_table()

self.update\_shedule\_button = QPushButton("Update")

self.shbox2.addWidget(self.update\_shedule\_button)

self.update\_shedule\_button.clicked.connect(self.\_update\_shedule)

self.shedule\_tab.setLayout(self.svbox)

Класс QWidget() создает виджет, который будет являться вкладкой в нашем приложении. Данный класс может также использоваться для создания отдельных окон, но в нашем случае будет вкладкой.

self.tabs.addTab(self.shedule\_tab, "Shedule") добавляет в структуру с вкладками новую вкладку с названием "Shedule".

Класс QGroupBox() может группировать виджеты, он предоставляет

рамку, заголовок вверху и может отображать несколько виджетов

внутри. В нашем случае он служит исключительно в декоративных

целях

* **Создаем метод для отображения таблицы с расписанием на понедельник**

def \_create\_monday\_table(self):

self.monday\_table = QTableWidget()

self.monday\_table.setSizeAdjustPolicy(QAbstractScrollArea.AdjustToContents)

self.monday\_table.setColumnCount(4)

self.monday\_table.setHorizontalHeaderLabels(["Subject", "Time", "", ""])

self.\_update\_monday\_table()

self.mvbox = QVBoxLayout()

self.mvbox.addWidget(self.monday\_table)

self.monday\_gbox.setLayout(self.mvbox)

Класс QTableWidget() создает пустую пользовательскую таблицу аналогичную таблицам Excel.

setSizeAdjustPolicy(QAbstractScrollArea.AdjustToContents)

устанавливает возможность изменения размера под размер данных в ячейке.

Метод setColumnCount() задает таблице количество колонок.

Метод setHorizontalHeaderLabels(["Название", "Название"])

задает колонкам подписи.

* **Создаем метод для обновления таблицы с расписанием на понедельник**

def \_update\_monday\_table(self):

self.cursor.execute("SELECT SQL запрос на вывод расписания на день 'понедельник'")

records = list(self.cursor.fetchall())

self.monday\_table.setRowCount(len(records))

for i, r in enumerate(records):

r = list(r)

joinButton = QPushButton("Join")

self.monday\_table.setItem(i, 0,

QTableWidgetItem(str(r[2])))

self.monday\_table.setItem(i, 1,

QTableWidgetItem(str(r[4])))

self.monday\_table.setCellWidget(i, 2, joinButton)

joinButton.clicked.connect(

lambda:self.\_change\_day\_from\_table(i, "monday"))

self.monday\_table.resizeRowsToContents()

Заполнение таблицы происходит в цикле for для того, чтобы динамически обрабатывать изменения в количестве записей.

Метод setRowCount() задает таблице количество строк.

Кнопка joinButton не является отдельным свойством класса MainWindow, так как нам не нужно ее "запоминать". Далее интерпретатор запоминает ее с помощью функции-обработчика clicked.connect().

Метод setItem(<Номер строки>, <Номер колонки>, <Строка с данными>) записывает в ячейку с определенным адресом строковые данные.

Метод setCellWidget(<Номер строки>, <Номер колонки>, <Виджет>)

помещает в ячейку с определенным адресом виджет.

В нашем случае это кнопка "Join".

Метод resizeRowsToContents() автоматически адаптирует размеры ячеек таблицы под размер данных внутри этой ячейки. Это необходимо использовать для экономии визуального пространства.

* **Создаем метод изменяющий запись в базе данных по нажатию на кнопку "Join"**

def \_change\_day\_from\_table(self, rowNum, day):

row = list()

for i in range(self.monday\_table.columnCount()):

try:

row.append(self.monday\_table.item(rowNum, i).text())

except:

row.append(None)

try:

self.cursor.execute("UPDATE SQL запрос на изменение одной строки в базе данных", (row[0],))

self.conn.commit()

except:

QMessageBox.about(self, "Error", "Enter all fields")

Метод columnCount() возвращает количество колонок таблицы.

Конструкция item(<Номер строки>, <Номер столбца>).text() возвращает текст, записанный в определенной ячейке.

* **Создаем метод обновляющий все таблицы на вкладке**

def \_update\_shedule(self):

self.\_update\_monday\_table()

…

Ваши методы обновления таблиц

* **"Запускаем" наше приложение**

app = QApplication(sys.argv)

win = MainWindow()

win.show()

sys.exit(app.exec\_())

**Система может быть дополнена Вашим функционалом, но должна соответствовать минимальным требованиям.**