Практическая работа №4: Создание графического интерфейса системы

Описание задания

В этой работе необходимо разработать графический интерфейс системы учёта посещений с указанными дополнениями.

Требуется реализовать следующие функции:

- **Запись статистики посещений в базу данных:** при фиксации прихода и ухода пользователя сохранять время и дату в локальную базу данных (SQLite). Для этого в Python используется модуль sqlite3 (входит в стандартную библиотеку, отдельная установка не требуется). Метод connect() создаёт файл базы при отсутствии и возвращает объект соединения. С помощью курсора (cursor) выполняются SQL-запросы INSERT/SELECT для сохранения и выборки данных.

- **Построение графика по дням недели:** обеспечить возможность отображения графика двух параметров (время прихода и время ухода) по оси Y в зависимости от дня недели (ось X). Для отрисовки графиков целесообразно использовать библиотеку PyQtGraph, оптимизированную для приложений на PyQt/PySide.

В PyQtGraph существует виджет PlotWidget, основанный на QGraphicsScene (благодаря чему он обеспечивает высокую скорость отрисовки). Пример:

достаточно создать PlotWidget, задать для него time = [1,2,…] и arrival = [..], а затем вызвать plot(time, arrival), чтобы отобразить кривую. Аналогично строится график ухода. На графике по оси X можно пронумеровать дни недели, а по оси Y – соответствующие значения времени (в часах или минутах от начала дня).

Пример кода обработчика

```
# pip install pyqt5 pyqtgraph
import sys
import numpy as np
from PyQt5.QtWidgets import QApplication, QMainWindow
import pyqtgraph as pg
from pyqtgraph import PlotWidget

class AttendancePlot(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("График посещаемости")
        self.resize(800, 500)

        self.plot = PlotWidget()
        self.setCentralWidget(self.plot)

        self._setup_plot()
```

```
    def _setup_plot(self):
        # Примерные данные: рабочие дни 0..4 (Пн..Пт)
        days = np.arange(5)  # [0, 1, 2, 3, 4]
        arrival_hours = np.array([9.2, 9.0, 9.4, 9.1, 9.3])   # время прихода (в часах)
        leave_hours   = np.array([18.0, 18.2, 18.3, 18.1, 17.9])  # время ухода

        # Настройка осей, сетки и легенды
        self.plot.setLabel('left', 'Время, часы')
        self.plot.setLabel('bottom', 'День недели')
        self.plot.showGrid(x=True, y=True, alpha=0.3)
        self.plot.addLegend()

        # Подписи дней недели
        ticks = [(0, 'Пн'), (1, 'Вт'), (2, 'Ср'), (3, 'Чт'), (4, 'Пт')]
        self.plot.getAxis('bottom').setTicks([ticks])

        # Диапазоны
        self.plot.setXRange(-0.5, 4.5)
        self.plot.setYRange(8.0, 20.0)

        # Отрисовка двух серий
        self.plot.plot(days, arrival_hours, name="Приход", symbol='o', symbolSize=8)
        self.plot.plot(days, leave_hours,   name="Уход",   symbol='t', symbolSize=8)

if __name__ == "__main__":
    app = QApplication(sys.argv)
    win = AttendancePlot()
    win.show()
    sys.exit(app.exec_())
```

- **Загрузка фотографии пользователя:** пользовательский интерфейс должен позволять выбрать и загрузить фотографию профиля. Обычно это реализуется кнопкой или пунктом меню «Загрузить фото», который вызывает стандартный диалог выбора файла (QFileDialog.getOpenFileName).

После выбора файла картинка загружается классом QPixmap и отображается, например, в виджете QLabel (у QLabel есть свойство pixmap, позволяющее показать изображение вместо текста)[6]. Таким образом можно представить фотографию пользователя в окне интерфейса.

Пример кода обработчика

```
# pip install pyqt5              # (подсказка) установка PyQt5
import sys                       # модуль для работы с аргументами и выходом из приложения
import sqlite3                   # стандартный модуль Python для SQLite
import base64                    # модуль для кодирования/декодирования base64

from PyQt5.QtCore import QBuffer, QByteArray, QIODevice   # буфер/байтовый
массив/режимы ввода-вывода Qt
```

```
from PyQt5.QtGui import QPixmap                         # класс для работы с растровыми изображениями
from PyQt5.QtWidgets import (                           # виджеты Qt
    QApplication, QWidget, QVBoxLayout, QLabel, QPushButton,
    QFileDialog, QMessageBox
)

DB_PATH = "photos.db"               # путь к файлу базы данных SQLite

def ensure_db():
    conn = sqlite3.connect(DB_PATH) # открыть (или создать) файл БД
    cur = conn.cursor()             # создать курсор для выполнения SQL
    cur.execute("""                 # создать таблицу, если её нет
        CREATE TABLE IF NOT EXISTS photos (
            id INTEGER PRIMARY KEY AUTOINCREMENT,  # идентификатор записи
            data TEXT NOT NULL                     # поле для строки base64
        )
    """)
    conn.commit()                   # зафиксировать изменения
    conn.close()                    # закрыть соединение

def pixmap_to_base64(pixmap: QPixmap, fmt: str = "PNG") -> str:
    """
    Кодирует QPixmap в строку base64.
    fmt: формат сериализации (например, 'PNG' или 'JPEG')
    """
    ba = QByteArray()                           # создаём байтовый контейнер Qt
    buff = QBuffer(ba)                          # оборачиваем контейнер в буфер
    buff.open(QIODevice.WriteOnly)              # открываем буфер на запись
    ok = pixmap.save(buff, fmt)                 # сохраняем QPixmap в буфер в заданном формате
    buff.close()                                # закрываем буфер
    if not ok:                                  # проверяем успешность сериализации
        raise RuntimeError("Не удалось сериализовать QPixmap в буфер")
    return base64.b64encode(bytes(ba)).decode("utf-8")  # кодируем байты в base64-строку

def base64_to_pixmap(b64: str) -> QPixmap:
    """
    Декодирует строку base64 обратно в QPixmap.
    """
    raw = base64.b64decode(b64.encode("utf-8"))     # декодируем base64-строку в «сырые» байты
    pixmap = QPixmap()                          # создаём пустой QPixmap
    if not pixmap.loadFromData(raw):            # пробуем загрузить изображение из байтов
        raise RuntimeError("Не удалось загрузить QPixmap из байтов")
    return pixmap                               # возвращаем восстановленный QPixmap

class PhotoDemo(QWidget):           # главное окно-демо наследуем от QWidget
    def __init__(self):
        super().__init__()          # инициализация базового класса QWidget
```

```
        self.setWindowTitle("Демо: фото ↔ base64 ↔ SQLite")  # заголовок окна
        self.resize(500, 400)                    # начальный размер окна

        self.label = QLabel("Фото не загружено")    # метка для отображения изображения/текста
        self.label.setStyleSheet(        # немного стилей для фона/рамки
            "QLabel { background: #f0f0f0; border: 1px solid #ccc; }"
        )
        self.label.setMinimumSize(300, 250) # минимальный размер области под фото
        self.label.setScaledContents(True) # масштабировать картинку под размер label

        self.btn_load_file = QPushButton("Загрузить фото (из файла)")      # кнопка выбора файла
        self.btn_save_db  = QPushButton("Сохранить в БД (как base64 → TEXT)") # кнопка
сохранения в БД
        self.btn_load_db  = QPushButton("Загрузить из БД (id=1)")          # кнопка загрузки из БД

        self.btn_load_file.clicked.connect(self.on_load_file)  # обработчик клика: выбрать файл
        self.btn_save_db.clicked.connect(self.on_save_db)      # обработчик клика: сохранить в БД
        self.btn_load_db.clicked.connect(self.on_load_db)      # обработчик клика: прочитать из БД

        layout = QVBoxLayout(self) # вертикальный лэйаут для размещения виджетов
        layout.addWidget(self.label)             # 1) область показа изображения
        layout.addWidget(self.btn_load_file)      # 2) кнопка «загрузить из файла»
        layout.addWidget(self.btn_save_db)        # 3) кнопка «сохранить в БД»
        layout.addWidget(self.btn_load_db)        # 4) кнопка «загрузить из БД»

        self.current_pixmap = QPixmap()          # текущий QPixmap в памяти (последнее
изображение)

    def on_load_file(self):
        path, _ = QFileDialog.getOpenFileName(    # открываем диалог выбора файла
            self, "Выбрать изображение",
            "", "Изображения (*.png *.jpg *.jpeg *.bmp *.gif)"  # фильтр форматов
        )
        if not path:    # если файл не выбран — выходим
            return
        pm = QPixmap(path)     # пробуем загрузить картинку в QPixmap
        if pm.isNull():                          # проверяем успешность
            QMessageBox.warning(self, "Ошибка", "Не удалось загрузить изображение.")
            return
        self.current_pixmap = pm       # сохраняем в «текущее» изображение
        self.label.setPixmap(pm)                 # показываем его в QLabel

    def on_save_db(self):
        if self.current_pixmap.isNull():   # если ещё ничего не загружено
            QMessageBox.information(self, "Внимание", "Сначала загрузите изображение.")
            return
        try:
            b64 = pixmap_to_base64(self.current_pixmap, fmt="PNG")  # кодируем QPixmap в base64
            conn = sqlite3.connect(DB_PATH)        # открываем БД
```

```
            cur = conn.cursor()                  # берём курсор
            cur.execute("DELETE FROM photos WHERE id=1")          # для простоты чистим
запись id=1
            cur.execute("INSERT INTO photos (id, data) VALUES (?, ?)", (1, b64))  # вставляем
новую
            conn.commit()                        # фиксируем транзакцию
            conn.close()                         # закрываем соединение
            QMessageBox.information(self, "Готово", "Изображение сохранено в БД (id=1).")
        except Exception as e:                   # ловим возможные ошибки
            QMessageBox.critical(self, "Ошибка", str(e))

    def on_load_db(self):
        try:
            conn = sqlite3.connect(DB_PATH)      # открываем соединение с БД
            cur = conn.cursor()                  # создаём курсор
            cur.execute("SELECT data FROM photos WHERE id=1")  # читаем строку base64 по id=1
            row = cur.fetchone()                 # забираем одну запись
            conn.close()                         # закрываем БД
            if not row:                          # если записи нет — сообщаем
                QMessageBox.information(self, "Нет данных", "В БД нет записи с id=1.")
                return
            b64 = row[0]                         # берём поле data (строка base64)
            pm = base64_to_pixmap(b64)           # декодируем обратно в QPixmap
            self.current_pixmap = pm             # обновляем текущий pixmap
            self.label.setPixmap(pm)             # отображаем на метке
        except Exception as e:                   # показываем ошибку при неудаче
            QMessageBox.critical(self, "Ошибка", str(e))

if __name__ == "__main__":                       # точка входа приложения
    ensure_db()                                  # гарантируем, что таблица в БД существует
    app = QApplication(sys.argv)                 # создаём объект приложения Qt
    w = PhotoDemo()                              # создаём наше окно-демо
    w.show()                                     # показываем окно
    sys.exit(app.exec_())                        # запускаем цикл обработки событий Qt
```

- **Интерактивные поля ввода:** каждый описанный элемент интерфейса (ввод времени, выбор даты, загрузка фото, фильтр по дате и т.д.) должен иметь соответствующие поля и кнопки для взаимодействия. Qt предлагает множество виджетов для ввода: однострочное текстовое поле (QLineEdit) для ввода текста, флажки (QCheckBox), выпадающие списки (QComboBox), поля ввода даты/времени (QDateTimeEdit) и др. Эти виджеты добавляются в окна и реагируют на сигналы (нажатие кнопки, изменение текста и т.д.). Например, QLineEdit предназначен для однострочного ввода текста, Интерфейс должен позволять пользователю вводить необходимые данные (имя, время, комментарии и т.п.) в интерактивных полях. **(Пример использования см. презентацию)**

- **Отдельное окно для каждого пункта меню:** каждый пункт главного меню или кнопка должна открывать своё окно (форму). В Qt любое окно – это виджет без родителя: достаточно создать экземпляр QWidget или QMainWindow (без параметра-родителя), и он

откроется как самостоятельное окно. Не существует ограничения на число окон, однако обычно для разных разделов программы создают отдельные классы-окна. Например, при нажатии на пункт меню «Статистика» можно создать и показать окно StatsWindow, а при выборе «Профиль» – окно ProfileWindow. Главное – сохранить в программе ссылку на открытое окно, иначе оно сразу закроется при выходе из функции.

Пример кода организации такой структуры

```
# pip install pyqt5
import sys
from PyQt5.QtWidgets import (
    QApplication, QMainWindow, QWidget, QPushButton,
    QVBoxLayout, QLabel, QAction, QMessageBox
)
from PyQt5.QtCore import Qt


# ---------------------------
# ОКНО "Статистика"
# ---------------------------
class StatsWindow(QWidget):
    """Окно статистики. Наследуемся от QWidget, чтобы это было отдельное окно."""
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Статистика")
        self.resize(400, 250)

        # Простейшее содержимое окна — надпись
        label = QLabel("Здесь будет статистика посещаемости.\n(Например, графики и
таблицы)")
        label.setAlignment(Qt.AlignCenter)

        # Размещаем элементы в вертикальном лэйауте
        layout = QVBoxLayout(self)
        layout.addWidget(label)


# ---------------------------
# ОКНО "Профиль"
# ---------------------------
class ProfileWindow(QWidget):
    """Окно профиля пользователя."""
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Профиль сотрудника")
        self.resize(400, 250)

        # Простейшее содержимое — надпись
        label = QLabel("Карточка сотрудника.\n(Фото, ФИО, должность и т.п.)")
```

```
        label.setAlignment(Qt.AlignCenter)

        # Размещаем элементы
        layout = QVBoxLayout(self)
        layout.addWidget(label)


# ---------------------------
# ГЛАВНОЕ ОКНО ПРИЛОЖЕНИЯ
# ---------------------------
class MainWindow(QMainWindow):
    """
    Главное окно приложения с меню и кнопками, открывающими отдельные окна.
    Важно: мы храним ссылки на дочерние окна в self._windows, чтобы они не были
    уничтожены после выхода из обработчика (иначе окно мгновенно закроется).
    """
    def __init__(self):
        super().__init__()
        self.setWindowTitle("HR Manager GUI — главное окно")
        self.resize(600, 400)

        # Хранилище ссылок на открытые окна.
        # Можно использовать список, словарь или набор, по ситуации.
        self._windows = []  # сюда будем класть объекты окон

        # Настроим меню
        self._init_menu()

        # Настроим центральную область с кнопками
        self._init_central()

    def _init_menu(self):
        """Создаём меню приложения с пунктами, открывающими окна."""
        menubar = self.menuBar()                # стандартная строка меню
        menu_windows = menubar.addMenu("Окна")   # добавим меню "Окна"

        # Пункт меню "Статистика"
        act_stats = QAction("Открыть «Статистика»", self)
        act_stats.triggered.connect(self.open_stats_window)
        menu_windows.addAction(act_stats)

        # Пункт меню "Профиль"
        act_profile = QAction("Открыть «Профиль»", self)
        act_profile.triggered.connect(self.open_profile_window)
        menu_windows.addAction(act_profile)

        # Пункт меню "О программе" — просто показать диалог
        act_about = QAction("О программе", self)
        act_about.triggered.connect(self.show_about)
```

```
        menubar.addAction(act_about)

    def _init_central(self):
        """Создаём центральный виджет с кнопками (альтернатива меню)."""
        central = QWidget()
        layout = QVBoxLayout(central)

        lbl = QLabel("Выберите действие:")
        lbl.setAlignment(Qt.AlignCenter)

        btn_stats = QPushButton("Открыть окно «Статистика»")
        btn_stats.clicked.connect(self.open_stats_window)

        btn_profile = QPushButton("Открыть окно «Профиль»")
        btn_profile.clicked.connect(self.open_profile_window)

        layout.addWidget(lbl)
        layout.addWidget(btn_stats)
        layout.addWidget(btn_profile)
        layout.addStretch(1)  # визуальный отступ

        self.setCentralWidget(central)

    # ---------------------------
    # ОБРАБОТЧИКИ ОТКРЫТИЯ ОКОН
    # ---------------------------
    def open_stats_window(self):
        """
        Создаём окно статистики и показываем его.
        КРИТИЧНО: сохраняем ссылку на объект окна в self._windows,
        иначе локальная переменная бы уничтожилась после выхода из метода,
        и окно закрылось бы сразу же.
        """
        win = StatsWindow()     # создаём экземпляр отдельного окна
        win.show()              # показываем как независимое окно
        self._windows.append(win)  # сохраняем ссылку

    def open_profile_window(self):
        """Аналогично создаём и показываем окно профиля."""
        win = ProfileWindow()
        win.show()
        self._windows.append(win)

    # ---------------------------
    # ПРОЧЕЕ
    # ---------------------------
    def show_about(self):
        """Простейший диалог «О программе»."""
        QMessageBox.information(
```

```
        self, "О программе",
        "Демо: главное окно + отдельные окна для пунктов меню.\n"
        "Важно хранить ссылки на окна, чтобы они не закрывались сразу."
    )


# ---------------------------
# ТОЧКА ВХОДА
# ---------------------------
if __name__ == "__main__":
    app = QApplication(sys.argv)  # создаём приложение Qt
    mw = MainWindow()             # создаём главное окно
    mw.show()                     # показываем
    sys.exit(app.exec_())
```

- **Диалоговые окна:** для взаимодействия с пользователем (подтверждение операций, показ ошибок, запрос дополнительных данных, выбор файлов) используются диалоги. В Qt диалоговые окна создаются через класс QDialog или через готовые диалоги (QMessageBox, QInputDialog, QFileDialog и т.д.). Это модальные окна, блокирующие основное окно до закрытия диалога[9]. При необходимости, в коде по нажатию кнопки может создаваться QDialog(self) (где self – главное окно), задаваться заголовок и содержимое, и запускаться методом exec()[10]. Например, для подтверждения удаления записи можно показать QMessageBox.question, а для выбора даты – QInputDialog.getText. Использование диалогов обеспечивает более удобный и интуитивный интерфейс.

Теоретический материал

PyQtGraph

PyQtGraph – это специализированная библиотека на Python для построения интерактивной графики и визуализации данных, основанная на Qt (PyQt/PySide)[3]. В отличие от Matplotlib, PyQtGraph глубоко интегрирован в Qt и использует мощь QGraphicsScene, поэтому обеспечивает очень быстрый рендеринг графиков и поддерживает интерактивность (масштабирование, перемещение, аннотации) «из коробки»[3]. Типичный сценарий: создаётся виджет PlotWidget, добавляемый в окно приложения, и с помощью метода plot(x, y) на нём рисуется линия данных[5]. Например, чтобы показать изменение времени прихода за неделю, можно вычислить числовые значения времени (в часах) для каждого дня и передать их в plot(). PyQtGraph также позволяет задавать подписи осей, легенды, маркеры и другие параметры оформления[3]. Благодаря высокой производительности PyQtGraph подходит даже для реального времени или частого обновления графиков в GUI-приложениях.

```
# pip install pyqt5 pyqtgraph
import sys
import numpy as np
from PyQt5.QtWidgets import QApplication, QMainWindow
import pyqtgraph as pg
```

```python
from pyqtgraph import PlotWidget


class AttendancePlot(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("График посещаемости")
        self.resize(800, 500)

        self.plot = PlotWidget()
        self.setCentralWidget(self.plot)

        self._setup_plot()

    def _setup_plot(self):
        # Примерные данные: рабочие дни 0..4 (Пн..Пт)
        days = np.arange(5)  # [0, 1, 2, 3, 4]
        arrival_hours = np.array([9.2, 9.0, 9.4, 9.1, 9.3])  # время прихода (в часах)
        leave_hours = np.array([18.0, 18.2, 18.3, 18.1, 17.9])  # время ухода

        # Настройка осей, сетки и легенды
        self.plot.setLabel('left', 'Время, часы')
        self.plot.setLabel('bottom', 'День недели')
        self.plot.showGrid(x=True, y=True, alpha=0.3)
        self.plot.addLegend()

        # Подписи дней недели
        ticks = [(0, 'Пн'), (1, 'Вт'), (2, 'Ср'), (3, 'Чт'), (4, 'Пт')]
        self.plot.getAxis('bottom').setTicks([ticks])

        # Диапазоны
        self.plot.setXRange(-0.5, 4.5)
        self.plot.setYRange(8.0, 20.0)

        # Отрисовка двух серий
        self.plot.plot(days, arrival_hours, name="Приход", symbol='o', symbolSize=8)
        self.plot.plot(days, leave_hours, name="Уход", symbol='t', symbolSize=8)


if __name__ == "__main__":
    app = QApplication(sys.argv)
    win = AttendancePlot()
    win.show()
    sys.exit(app.exec_())
```