МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ**

**ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА»**

**(БГТУ им. В. Г. Шухова)**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

**Лабораторная работа № 13**

по дисциплине: Объектно-ориентированное программирование

тема: «Знакомство с библиотеками Python. PyQT»

Выполнил: ст. группы ПВ-223

Игнатьев Артур Олегович

Проверил:

асс. Черников Сергей Викторович

Белгород 2024г.

**Лабораторная работа №13**

**«Знакомство с библиотеками Python. PyQT»**

**Цель работы:** приобретение практических навыков создания приложений на языке Python. QT приложений.

**Вариант 3**

Изображение выглядит как текст, Шрифт, линия, снимок экрана

Автоматически созданное описание

Код программы:

Main\_window.py

import sys

from PyQt6.QtWidgets import QMainWindow, QVBoxLayout, QWidget, QApplication, QMessageBox

from PyQt6.QtGui import QIcon

from database import init\_database

from stopwatch\_widget import StopwatchWidget

from task\_table\_widget import TaskTableWidget

class MainWindow(QMainWindow):

    def \_\_init\_\_(self):

        super().\_\_init\_\_()

        self.setWindowTitle("Task Manager")

        self.setWindowIcon(QIcon("images/logo.png"))

        self.stopwatch\_widget = StopwatchWidget()

        self.task\_table\_widget = TaskTableWidget()

        self.stopwatch\_widget.task\_started.connect(self.task\_table\_widget.start\_task)

        self.stopwatch\_widget.task\_stopped.connect(self.task\_table\_widget.load\_tasks)

        self.stopwatch\_widget.task\_paused.connect(self.task\_table\_widget.edit\_task)

        self.task\_table\_widget.task\_added.connect(self.stopwatch\_widget.load\_tasks)

        self.task\_table\_widget.task\_updated.connect(self.stopwatch\_widget.load\_tasks)

        self.task\_table\_widget.task\_deleted.connect(self.stopwatch\_widget.load\_tasks)

        self.task\_table\_widget.load\_tasks()

        main\_layout = QVBoxLayout()

        main\_layout.addWidget(self.stopwatch\_widget)

        main\_layout.addWidget(self.task\_table\_widget)

        main\_widget = QWidget()

        main\_widget.setLayout(main\_layout)

        self.setCentralWidget(main\_widget)

    def closeEvent(self, event):

        if self.stopwatch\_widget.timer.isActive():

            msg\_box = QMessageBox()

            msg\_box.setIcon(QMessageBox.Icon.Warning)

            msg\_box.setWindowTitle("Timer is running")

            msg\_box.setText("The timer is currently running. Are you sure you want to close the application?")

            msg\_box.setStandardButtons(QMessageBox.StandardButton.Yes | QMessageBox.StandardButton.No)

            result = msg\_box.exec()

            if result == QMessageBox.StandardButton.Yes:

                self.stopwatch\_widget.stop\_stopwatch()

                event.accept()

            else:

                event.ignore()

        else:

            event.accept()

if \_\_name\_\_ == "\_\_main\_\_":

    init\_database()

    app = QApplication([])

    window = MainWindow()

    window.resize(1000, 700)

    screen = window.screen().availableSize()

    position = window.screen().availableSize()

    x = (screen.width() - window.frameSize().width()) - 20

    y = 80

    window.move(x, y)

    window.show()

    sys.exit(app.exec())

database.py

import sqlite3

from sqlite3 import Error

def create\_connection():

    conn = None

    try:

        conn = sqlite3.connect("tasks.db")

    except Error as e:

        print(e)

    if conn:

        return conn

    else:

        raise Exception("Error connecting to the database")

def create\_tables(conn):

    cursor = conn.cursor()

    # Create the 'tasks' table

    cursor.execute("""

    CREATE TABLE IF NOT EXISTS tasks (

        id INTEGER PRIMARY KEY,

        name TEXT NOT NULL,

        daily\_target INTEGER,

        weekly\_target INTEGER

    )

    """)

    # Create the 'logs' table

    cursor.execute("""

    CREATE TABLE IF NOT EXISTS logs (

        id INTEGER PRIMARY KEY,

        task\_id INTEGER NOT NULL,

        name TEXT NOT NULL,

        start\_timestamp INTEGER NOT NULL,

        stop\_timestamp INTEGER NOT NULL,

        FOREIGN KEY (task\_id) REFERENCES tasks (id)

    )

    """)

    conn.commit()

def init\_database():

    conn = create\_connection()

    create\_tables(conn)

    conn.close()

if \_\_name\_\_ == "\_\_main\_\_":

    init\_database()

stopwatch\_widget.py

from PyQt6.QtCore import QTime, QTimer, Qt, QElapsedTimer, pyqtSignal

from PyQt6.QtGui import QFontDatabase, QFont, QIcon

from PyQt6.QtWidgets import QWidget, QVBoxLayout, QComboBox, QLabel, QHBoxLayout, QPushButton

from database import create\_connection

class StopwatchWidget(QWidget):

    task\_started = pyqtSignal(int)

    task\_paused = pyqtSignal(int)

    task\_stopped = pyqtSignal(int)

    def \_\_init\_\_(self, parent=None):

        super().\_\_init\_\_(parent)

        layout = QVBoxLayout(self)

        self.task\_combo = QComboBox()

        layout.addWidget(self.task\_combo)

        self.load\_tasks()

        self.time\_label = QLabel("00:00:00")

        layout.addWidget(self.time\_label)

        font\_id = QFontDatabase.addApplicationFont("fonts/TriodPostnaja.ttf")

        families = QFontDatabase.applicationFontFamilies(font\_id)

        self.time\_label.setFont(QFont(families[0], 20))

        self.time\_label.setAlignment(Qt.AlignmentFlag.AlignCenter)

        self.timer = QTimer()

        self.elapsed\_timer = QElapsedTimer()

        self.timer.timeout.connect(self.update\_stopwatch)

        self.start\_time = None

        button\_layout = QHBoxLayout()

        play\_button = QPushButton('&Start')

        play\_button.setIcon(QIcon("images/play.png"))

        play\_button.clicked.connect(self.start\_stopwatch)

        button\_layout.addWidget(play\_button)

        pause\_button = QPushButton('&Pause')

        pause\_button.setIcon(QIcon("images/pause.png"))

        pause\_button.clicked.connect(self.pause\_stopwatch)

        button\_layout.addWidget(pause\_button)

        stop\_button = QPushButton('S&top')

        stop\_button.setIcon(QIcon("images/stop.png"))

        stop\_button.clicked.connect(self.stop\_stopwatch)

        button\_layout.addWidget(stop\_button)

        layout.addLayout(button\_layout)

        self.time = QTime(0, 0)

    def load\_tasks(self):

        self.task\_combo.clear()

        conn = create\_connection()

        cursor = conn.cursor()

        cursor.execute("SELECT \* FROM tasks")

        tasks = cursor.fetchall()

        for task in tasks:

            self.task\_combo.addItem(task[1], task[0])

        conn.close()

    def start\_stopwatch(self):

        if not self.timer.isActive():

            self.timer.start(1000)

            self.elapsed\_timer.start()

            self.start\_time = QTime.currentTime()

            task\_id = self.task\_combo.currentData()

            if task\_id is not None:

                self.task\_started.emit(task\_id)

    def pause\_stopwatch(self):

        if self.timer.isActive():

            self.timer.stop()

            task\_id = self.task\_combo.currentData()

            if task\_id is not None:

                self.task\_paused.emit(task\_id)

    def stop\_stopwatch(self):

        if not self.timer.isActive():

            return

        self.timer.stop()

        elapsed\_time = self.elapsed\_timer.elapsed() // 1000

        task\_id = self.task\_combo.currentData()

        if task\_id is None:

            return

        conn = create\_connection()

        cursor = conn.cursor()

        cursor.execute("SELECT name FROM tasks WHERE id=?", (task\_id,))

        task\_name = cursor.fetchone()[0]

        cursor.execute("""

            INSERT INTO logs (task\_id, name, start\_timestamp, stop\_timestamp)

            VALUES (?, ?, datetime('now', 'localtime', '-' || ? || ' seconds'), datetime('now', 'localtime'))

        """, (task\_id, task\_name, elapsed\_time))

        conn.commit()

        conn.close()

        self.reset\_stopwatch()

        self.task\_stopped.emit(task\_id)

    def update\_stopwatch(self):

        self.time = self.time.addSecs(1)

        self.update\_stopwatch\_display()

    def update\_stopwatch\_display(self):

        self.time\_label.setText(self.time.toString('hh:mm:ss'))

    def reset\_stopwatch(self):

        self.time = QTime(0, 0)

        self.update\_stopwatch\_display()

task\_table\_widget.py

from PyQt6.QtCore import QTime, pyqtSignal

from PyQt6.QtWidgets import QTableWidgetItem, QInputDialog, QAbstractItemView

from PyQt6.QtWidgets import QWidget, QVBoxLayout, QTableWidget, QPushButton, QHBoxLayout

from database import create\_connection

class TaskTableWidget(QWidget):

    task\_added = pyqtSignal()

    task\_updated = pyqtSignal()

    task\_deleted = pyqtSignal()

    def \_\_init\_\_(self, parent=None):

        super().\_\_init\_\_(parent)

        self.table = QTableWidget(self)

        self.table.setColumnCount(5)

        self.table.setHorizontalHeaderLabels(["ID", "Task", "Day", "Week", "Spent"])

        self.table.hideColumn(0)

        self.table.setColumnWidth(1, 160)

        self.table.setColumnWidth(2, 20)

        self.table.setColumnWidth(3, 20)

        self.table.setColumnWidth(4, 40)

        self.table.horizontalHeader().setStretchLastSection(True)

        self.table.setEditTriggers(QAbstractItemView.EditTrigger.NoEditTriggers)

        self.load\_tasks()

        add\_button = QPushButton("Add")

        add\_button.clicked.connect(self.add\_task)

        edit\_button = QPushButton("Edit")

        edit\_button.clicked.connect(self.edit\_task)

        delete\_button = QPushButton("Delete")

        delete\_button.clicked.connect(self.delete\_task)

        button\_layout = QHBoxLayout()

        button\_layout.addWidget(add\_button)

        button\_layout.addWidget(edit\_button)

        button\_layout.addWidget(delete\_button)

        layout = QVBoxLayout(self)

        layout.addWidget(self.table)

        layout.addLayout(button\_layout)

    def load\_tasks(self):

        conn = create\_connection()

        cursor = conn.cursor()

        cursor.execute("""

            SELECT tasks.id, tasks.name, tasks.daily\_target, tasks.weekly\_target, COALESCE(SUM(strftime('%s', logs.stop\_timestamp) - strftime('%s', logs.start\_timestamp)), 0) as time\_spent

            FROM tasks

            LEFT JOIN logs ON tasks.id = logs.task\_id

            GROUP BY tasks.id

        """)

        tasks = cursor.fetchall()

        self.table.setRowCount(0)

        for i, task in enumerate(tasks):

            self.table.insertRow(i)

            for j, value in enumerate(task):

                if j == 4:

                    value = QTime(0, 0).addSecs(value).toString('hh:mm:ss')

                item = QTableWidgetItem(str(value))

                self.table.setItem(i, j, item)

        conn.close()

    def start\_task(self, task\_id):

        pass    # there were some thoughts about what to do when timer starts, but finally i decide to skip it

    def add\_task(self):

        task\_name, ok = QInputDialog.getText(self, "Add Task", "Task name:")

        if ok and task\_name:

            daily\_target, ok = QInputDialog.getInt(self, "Add Task", "Daily target (minutes):", value=0, min=0)

            if not ok:

                return

            weekly\_target, ok = QInputDialog.getInt(self, "Add Task", "Weekly target (minutes):", value=0, min=0)

            if not ok:

                return

            conn = create\_connection()

            cursor = conn.cursor()

            cursor.execute("INSERT INTO tasks (name, daily\_target, weekly\_target) VALUES (?, ?, ?)",

                           (task\_name, daily\_target, weekly\_target))

            conn.commit()

            conn.close()

            self.load\_tasks()

            self.task\_added.emit()

    def edit\_task(self):

        selected\_items = self.table.selectedItems()

        if not selected\_items:

            return

        selected\_item = selected\_items[0]

        selected\_row = self.table.row(selected\_item)

        task\_id = int(self.table.item(selected\_row, 0).text())

        task\_name = self.table.item(selected\_row, 1).text()

        daily\_target = int(self.table.item(selected\_row, 2).text())

        weekly\_target = int(self.table.item(selected\_row, 3).text())

        new\_task\_name, ok = QInputDialog.getText(self, "Edit Task", "Task name:", text=task\_name)

        new\_daily\_target, ok = QInputDialog.getInt(self, "Edit Task", "Daily target (minutes):", value=daily\_target, min=0)

        if not ok:

            return

        new\_weekly\_target, ok = QInputDialog.getInt(self, "Edit Task", "Weekly target (minutes):", value=weekly\_target, min=0)

        if not ok:

            return

        if ok and new\_task\_name:

            conn = create\_connection()

            cursor = conn.cursor()

            cursor.execute("UPDATE tasks SET name=?, daily\_target=?, weekly\_target=? WHERE id=?",

                           (new\_task\_name, new\_daily\_target, new\_weekly\_target, task\_id))

            conn.commit()

            conn.close()

            self.load\_tasks()

            self.task\_updated.emit()

    def delete\_task(self):

        selected\_items = self.table.selectedItems()

        if not selected\_items:

            return

        selected\_item = selected\_items[0]

        selected\_row = self.table.row(selected\_item)

        task\_id = int(self.table.item(selected\_row, 0).text())

        conn = create\_connection()

        cursor = conn.cursor()

        cursor.execute("DELETE FROM tasks WHERE id=?", (task\_id,))

        conn.commit()

        conn.close()

        self.load\_tasks()

        self.task\_deleted.emit()

Результат работы:

Изображение выглядит как текст, снимок экрана, мультимедиа, программное обеспечение

Автоматически созданное описание Изображение выглядит как текст, снимок экрана, мультимедиа, программное обеспечение

Автоматически созданное описание

**Вывод:** на этой лабораторной работе были приобретены практические навыки создания приложений на языке Python. QT приложений.