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

3	QT-TaskManager

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 | QMessage-
Box.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

```python
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

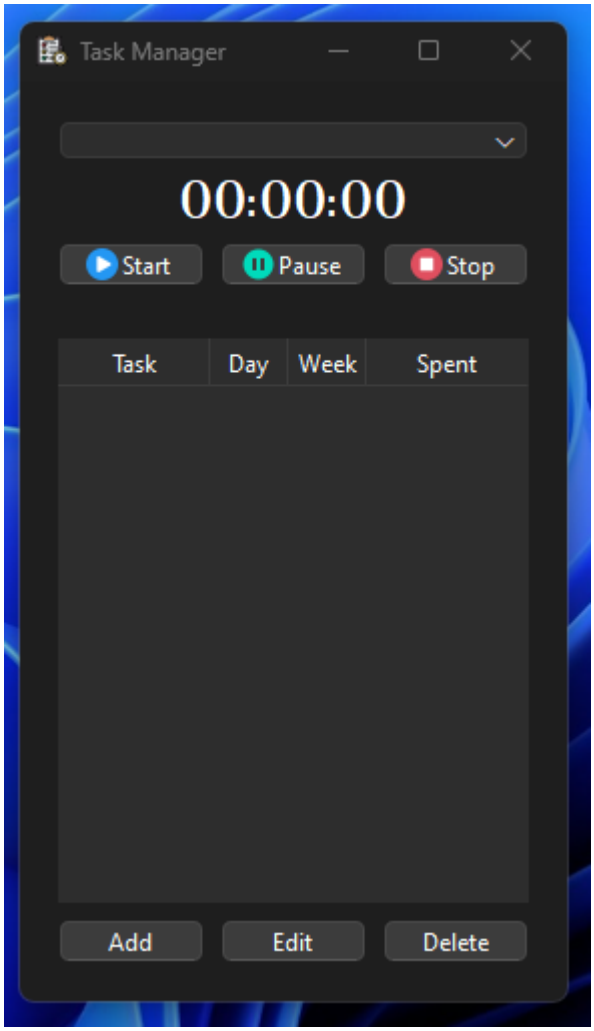
```
from PyQt6.QtCore import QTime, pyqtSignal
from PyQt6.QtWidgets import QTableWidgetItem, QInputDialog, QAbstractItemView

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)
```

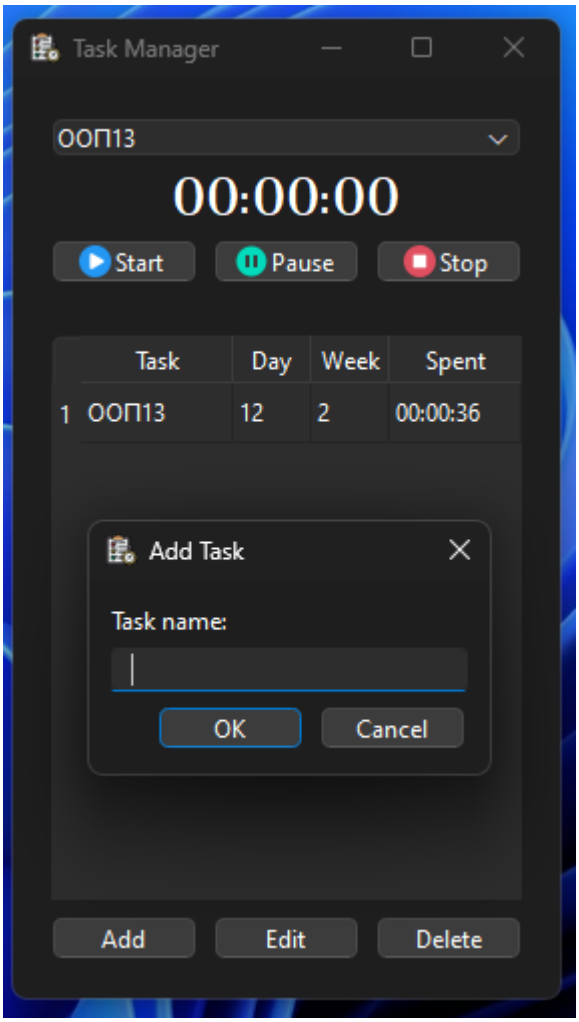
```
    def load_tasks(self):
        conn = create_connection()
        cursor = conn.cursor()
        cursor.execute("""
            SELECT tasks.id, tasks.name, tasks.daily_target, tasks.weekly_target,
COALESCE(SUM(strftime('%s', logs.stop_timestamp) - strftime('%s',
logs.start_timestamp)), 0) as 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 tar-
get (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 tar-
get (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=?, week-
ly_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:





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