Vežbe 10 - Zadaci

Instalirati virtuelno okruženje:
sudo apt install python3-venv
Napraviti virtuelno okruženje:
python3 -m venv ime_okruzenja
Aktivirati virtuelno okruženje:
source ime_okruzenja/bin/activate
Instalirati PySide6 komandom:
pip3 install PySide6
Otvoriti folder u kom je napravljeno virtuelno okruzenje u VScode. Prilikom realizacije .py fajla, VScode bi trebao prepoznati interpreter u donjem desnom uglu (Python) a pored se nalazi verzija pajtona gde u zagradi pored broja verzije mora pisati ***(‘ime_okruzenja’:venv)***. Ako to nije slučaj, kliknuti na verziju i iz padajućeg menija odabrati interpreter virtuelnog okruženja. Ako to nije moguće aktivirati ga manuelno kao što je objašnjeno u koracima iznad.

Dizajner bi trebao biti dostupan sa instalacijom pyside6. Pokreće se iz terminala komandom: ***pyside6-designer***
Za prevođenje ui dizajna u py fajl koristiti komandu: ***pyside6-uic ime_fajla.ui > ime_fajla.py***
Ako se pojavi greška:
qt.qpa.plugin: Could not load the Qt platform plugin ’xcb’ in ” even though it was found. This application failed to start because no Qt platform plugin could be initialized. Reinstalling the application may fix this problem.
Available platform plugins are: eglfs, linuxfb, minimal, minimalegl, offscreen, vnc, xcb.
Aborted (core dumped)
Rešenje:
sudo apt install ‘∧libxcb.*-dev’

Zadatak 1. Realizovati aplikaciju koja osluškuje da li je korisnik pritisnuo strelicu gore ili dore na tastaturi. Po pritisku tastera neophodno je emitovati signal, gde će se u funkciji rukovaoca izvršiti promena boje labele.

```python
import sys
from PySide6.QtWidgets import (QApplication, QMainWindow,
QWidget, QLabel, QVBoxLayout)
from PySide6.QtCore import Qt, Signal, QObject

class SendSignal(QObject):
    change_style = Signal()

class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.initializeUI()
    def initializeUI(self):
        self.setGeometry(100, 100, 300, 200)
        self.setWindowTitle("Emitovanje signala")
        self.setUpMainWindow()
        self.show()

    def setUpMainWindow(self):
        self.index = 0
        self.direction = ""
        self.sig = SendSignal()
        self.sig.change_style.connect(self.changeBackground)
        header_label = QLabel(
                        """<p align='center'>Pritisni strelicu
                        <b>gore</b> ili <b>dole</b>
                        na tastaturi.</p>""")
        self.colors_list = ["red", "orange", "yellow",
                            "green", "blue", "purple"]
        self.label = QLabel()
        self.label.setStyleSheet(f"""background-color:
                                {self.colors_list[self.index]}""")
        main_v_box = QVBoxLayout()
        main_v_box.addWidget(header_label)
        main_v_box.addWidget(self.label)
        container = QWidget()
        container.setLayout(main_v_box)
        self.setCentralWidget(container)

    def keyPressEvent(self, event):
        if event.key() == Qt.Key.Key_Up:
            self.direction = "up"
            self.sig.change_style.emit()
        elif event.key() == Qt.Key.Key_Down:
            self.direction = "down"
            self.sig.change_style.emit()

    def changeBackground(self):
        if self.direction == "up" and \
            self.index < len(self.colors_list) - 1:
            self.index = self.index + 1
            self.label.setStyleSheet(f"""background-color:
                    {self.colors_list[self.index]}""")

        elif self.direction == "down" and self.index > 0:
            self.index = self.index - 1
            self.label.setStyleSheet(f"""background-color:
                    {self.colors_list[self.index]}""")

if __name__ == '__main__':
    app = QApplication(sys.argv)
    window = MainWindow()
    sys.exit(app.exec())
```

Zadatak 2. Realizovati aplikaciju koja broji klikove miša. Pored nje se pokreće dugačak zadatak koji pravi pauzu od 5 sekundi. Ukazati na problem blokiranja petlje događaja.

```python
import sys
from time import sleep

from PySide6.QtCore import Qt
from PySide6.QtWidgets import (QApplication, QLabel, QMainWindow,
                               QPushButton,QVBoxLayout,QWidget)

class Window(QMainWindow):
    def __init__(self, parent=None):
        super().__init__(parent)
        self.clicksCount = 0
        self.setupUi()

    def setupUi(self):
        self.setWindowTitle("Freezing GUI")
        self.resize(300, 150)
        self.centralWidget = QWidget()
        self.setCentralWidget(self.centralWidget)

        self.clicksLabel = QLabel("Counting: 0 clicks", self)
        self.clicksLabel.setAlignment(Qt.AlignHCenter | Qt.AlignVCenter)
        self.stepLabel = QLabel("Long-Running Step: 0")
        self.stepLabel.setAlignment(Qt.AlignHCenter | Qt.AlignVCenter)
        self.countBtn = QPushButton("Click me!", self)
        self.countBtn.clicked.connect(self.countClicks)
        self.longRunningBtn = QPushButton("Long-Running Task!", self)
        self.longRunningBtn.clicked.connect(self.runLongTask)

        layout = QVBoxLayout()
        layout.addWidget(self.clicksLabel)
        layout.addWidget(self.countBtn)
        layout.addStretch()
        layout.addWidget(self.stepLabel)
        layout.addWidget(self.longRunningBtn)
        self.centralWidget.setLayout(layout)

    def countClicks(self):
        self.clicksCount += 1
        self.clicksLabel.setText(f"Counting: {self.clicksCount} clicks")

    def reportProgress(self, n):
        self.stepLabel.setText(f"Long-Running Step: {n}")

    def runLongTask(self):
        for i in range(5):
            sleep(1)
            self.reportProgress(i + 1)

app = QApplication(sys.argv)
win = Window()
win.show()
sys.exit(app.exec())
```

Zadatak 3. Nadograditi prethodni primer koristeći klasu QThread rešiti ukazani problem.

```python
import sys
from PySide6.QtWidgets import (QMainWindow, QApplication,
                            QWidget, QLabel, QPushButton,
                            QVBoxLayout)
from PySide6.QtCore import QObject, QThread, Signal, Qt
from time import sleep

class Worker(QObject):
    finished = Signal()
    progress = Signal(int)

    def run(self):
        for i in range(5):
            sleep(1)
            self.progress.emit(i + 1)
        self.finished.emit()

class Window(QMainWindow):
    def __init__(self, parent=None):
        super().__init__(parent)
        self.clicksCount = 0
        self.setupUi()

    def setupUi(self):
        self.setWindowTitle("Freezing GUI")
        self.resize(300, 150)
        self.centralWidget = QWidget()
        self.setCentralWidget(self.centralWidget)

        self.clicksLabel = QLabel("Counting: 0 clicks", self)
        self.clicksLabel.setAlignment(Qt.AlignHCenter | Qt.AlignVCenter)
        self.stepLabel = QLabel("Long-Running Step: 0")
        self.stepLabel.setAlignment(Qt.AlignHCenter | Qt.AlignVCenter)
        self.countBtn = QPushButton("Click me!", self)
        self.countBtn.clicked.connect(self.countClicks)
        self.longRunningBtn = QPushButton("Long-Running Task!", self)
        self.longRunningBtn.clicked.connect(self.runLongTask)

        layout = QVBoxLayout()
        layout.addWidget(self.clicksLabel)
        layout.addWidget(self.countBtn)
        layout.addStretch()
        layout.addWidget(self.stepLabel)
        layout.addWidget(self.longRunningBtn)
        self.centralWidget.setLayout(layout)

    def countClicks(self):
        self.clicksCount += 1
        self.clicksLabel.setText(f"Counting: {self.clicksCount} clicks")

    def reportProgress(self, n):
        self.stepLabel.setText(f"Long-Running Step: {n}")

    def runLongTask(self):
        self.thread = QThread()
        self.worker = Worker()

        self.worker.moveToThread(self.thread)

        #signal za pokretanje niti
        self.thread.started.connect(self.worker.run)
        #signal za zavrsetak niti
        self.worker.finished.connect(self.thread.quit)
        #signalizira event loop-u da se objekti mogu obrisati
        self.worker.finished.connect(self.worker.deleteLater)
        self.thread.finished.connect(self.thread.deleteLater)
        #signal na postojecu metodu
        self.worker.progress.connect(self.reportProgress)

        #pokreni nit
        self.thread.start()

        self.longRunningBtn.setEnabled(False)
        self.thread.finished.connect(
            lambda: self.longRunningBtn.setEnabled(True)
        )
        self.thread.finished.connect(
            lambda: self.stepLabel.setText("Long-Running Step: 0")
        )

app = QApplication(sys.argv)
win = Window()
win.show()
sys.exit(app.exec())
```

Zadatak 4. Realizovati aplikaciju koja koristi QRunnable i QThreadPool kako bi se napravilo više niti koje se paralelno izvršavaju i koriste.

```python
import random
import sys
import time

from PySide6.QtCore import QRunnable, Qt, QThreadPool
from PySide6.QtWidgets import ( QApplication, QLabel,QMainWindow,
                               QPushButton, QVBoxLayout, QWidget)

class Runnable(QRunnable):
    def __init__(self, n):
        super().__init__()
        self.n = n
    def run(self):
        for i in range(5):
            print(f"Programska nit {self.n}, korak {i + 1}/5")
            time.sleep(random.randint(700, 2500) / 1000)


class Window(QMainWindow):
    def __init__(self, parent=None):
        super().__init__(parent)
        self.setupUi()

    def setupUi(self):
        self.setWindowTitle("QThreadPool + QRunnable")
        self.resize(250, 150)
        self.centralWidget = QWidget()
        self.setCentralWidget(self.centralWidget)

        self.label = QLabel("Hello, World!")
        self.label.setAlignment(Qt.AlignHCenter | Qt.AlignVCenter)
        countBtn = QPushButton("Click me!")
        countBtn.clicked.connect(self.runTasks)

        layout = QVBoxLayout()
        layout.addWidget(self.label)
        layout.addWidget(countBtn)
        self.centralWidget.setLayout(layout)

    def runTasks(self):
        threadCount = QThreadPool.globalInstance().maxThreadCount()
        self.label.setText(f"Pokrenuto {threadCount} niti")
        pool = QThreadPool()

        for i in range(threadCount):
            runnable = Runnable(i)
            pool.start(runnable)


app = QApplication(sys.argv)
window = Window()
window.show()
sys.exit(app.exec())
```

Zadatak 5. Pošto QRunnable ne podržavaju emitovanje signala, za razliku od QThread-a, realizovati metod prenosa podataka između QRunnable instance i glavnog programa. Više informacija na linku.

```
from PySide6.QtWidgets import (QVBoxLayout, QLabel, QPushButton,
                               QWidget, QMainWindow, QApplication)
from PySide6.QtCore import QTimer, QRunnable, Slot, Signal, QObject, QThreadPool

import sys
import time
import traceback

class WorkerSignals(QObject):
    finished = Signal()
    error = Signal(tuple)
    result = Signal(object)
    progress = Signal(int)

class Worker(QRunnable):

    def __init__(self, fn, *args, **kwargs):
        super(Worker, self).__init__()
        self.fn = fn
        self.args = args
        self.kwargs = kwargs
        self.signals = WorkerSignals()

        self.kwargs['progress_callback'] = self.signals.progress

    @Slot()
    def run(self):
        try:
            result = self.fn(*self.args, **self.kwargs)
        except:
            traceback.print_exc()
            exctype, value = sys.exc_info()[:2]
            self.signals.error.emit((exctype, value, traceback.format_exc()))
        else:
            self.signals.result.emit(result)
        finally:
            self.signals.finished.emit()



class MainWindow(QMainWindow):
    def __init__(self, *args, **kwargs):
        super(MainWindow, self).__init__(*args, **kwargs)

        self.counter = 0
        layout = QVBoxLayout()
        self.l = QLabel("Start")
        b = QPushButton("DANGER!")
        b.pressed.connect(self.oh_no)

        layout.addWidget(self.l)
        layout.addWidget(b)

        w = QWidget()
        w.setLayout(layout)
        self.setCentralWidget(w)
        self.show()

        self.threadpool = QThreadPool()
        print("Maximum %d threads" % self.threadpool.maxThreadCount())

        self.timer = QTimer()
        self.timer.setInterval(1000)
        self.timer.timeout.connect(self.recurring_timer)
        self.timer.start()

    def progress_fn(self, n):
        print("%d%% done" % n)

    def execute_this_fn(self, progress_callback):
        for n in range(0, 5):
            time.sleep(1)
            progress_callback.emit(n*100/4)

        return "Done."

    def print_output(self, s):
        print(s)

    def thread_complete(self):
        print("THREAD COMPLETE!")

    def oh_no(self):
        worker = Worker(self.execute_this_fn)
```

```python
        worker.signals.result.connect(self.print_output)
        worker.signals.finished.connect(self.thread_complete)
        worker.signals.progress.connect(self.progress_fn)

        self.threadpool.start(worker)

    def recurring_timer(self):
        self.counter +=1
        self.l.setText("Counter: %d" % self.counter)


app = QApplication(sys.argv)
window = MainWindow()
app.exec()
```