Vežbe 9 - 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 u PySide6 kao instancu klase MainWindow. Dodati padajući meni *File* u kom se nalazi opcija *Quit*. Dodati prečicu na opciju *Quit*.

```python
import sys
from PySide6.QtWidgets import (QApplication, QMainWindow)
from PySide6.QtGui import QAction

class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.initializeUI()

    def initializeUI(self):
        self.setMinimumSize(450, 350)
        self.setWindowTitle("Main Window Template")
        self.setUpMainWindow()
        self.createActions()
        self.createMenu()
        self.show()

    def setUpMainWindow(self):
        pass

    def createActions(self):
        self.quit_act = QAction("&Quit")
        self.quit_act.setShortcut("Ctrl+Q")
        self.quit_act.triggered.connect(self.close)

    def createMenu(self):
        file_menu = self.menuBar().addMenu("File")
        file_menu.addAction(self.quit_act)

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

Zadatak 2. Realizovati repliku aplikacije Notepad.

```python
import sys
from PySide6.QtWidgets import (QApplication, QMainWindow,
                            QMessageBox, QTextEdit, QFileDialog, QInputDialog,
                            QFontDialog, QColorDialog)
from PySide6.QtCore import Qt
from PySide6.QtGui import QIcon, QTextCursor, QColor, QAction

class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.initializeUI()

    def initializeUI(self):
        self.setMinimumSize(400, 500)
        self.setWindowTitle("Notepad")
        self.setUpMainWindow()
        self.createActions()
        self.createMenu()
        self.show()

    def setUpMainWindow(self):
        self.text_edit = QTextEdit()
        self.setCentralWidget(self.text_edit)

    def createActions(self):
        self.new_act = QAction(QIcon("slike/new_file.png"), "New")
        self.new_act.setShortcut("Ctrl+N")
        self.new_act.triggered.connect(self.clearText)

        self.open_act = QAction(QIcon("slike/open_file.png"), "Open")
        self.open_act.setShortcut("Ctrl+O")
        self.open_act.triggered.connect(self.openFile)

        self.save_act = QAction(QIcon("slike/save_file.png"), "Save")
        self.save_act.setShortcut("Ctrl+S")
        self.save_act.triggered.connect(self.saveToFile)

        self.quit_act = QAction(QIcon("slike/exit.png"), "Quit")
        self.quit_act.setShortcut("Ctrl+Q")
        self.quit_act.triggered.connect(self.close)

        self.undo_act = QAction(QIcon("slike/undo.png"), "Undo")
        self.undo_act.setShortcut("Ctrl+Z")
        self.undo_act.triggered.connect(self.text_edit.undo)

        self.redo_act = QAction(QIcon("slike/redo.png"), "Redo")
        self.redo_act.setShortcut("Ctrl+Shift+Z")
        self.redo_act.triggered.connect(self.text_edit.redo)

        self.cut_act = QAction(QIcon("slike/cut.png"), "Cut")
        self.cut_act.setShortcut("Ctrl+X")
        self.cut_act.triggered.connect(self.text_edit.cut)

        self.copy_act = QAction(QIcon("slike/copy.png"), "Copy")
        self.copy_act.setShortcut("Ctrl+C")
        self.copy_act.triggered.connect(self.text_edit.copy)

        self.paste_act = QAction(QIcon("slike/paste.png"), "Paste")
        self.paste_act.setShortcut("Ctrl+V")
        self.paste_act.triggered.connect(self.text_edit.paste)

        self.find_act = QAction(QIcon("slike/find.png"), "Find All")
        self.find_act.setShortcut("Ctrl+F")
        self.find_act.triggered.connect(self.searchText)

        self.font_act = QAction(QIcon("slike/font.png"), "Font")
        self.font_act.setShortcut("Ctrl+T")
        self.font_act.triggered.connect(self.chooseFont)

        self.color_act = QAction(QIcon("slike/color.png"), "Color")
        self.color_act.setShortcut("Ctrl+Shift+C")
        self.color_act.triggered.connect(self.chooseFontColor)

        self.highlight_act = QAction(QIcon("slike/highlight.png")
                                , "Highlight")
        self.highlight_act.setShortcut("Ctrl+Shift+H")
        self.highlight_act.triggered.connect(self.chooseFontBackgroundColor)

        self.about_act = QAction("About")
        self.about_act.triggered.connect(self.aboutDialog)

    def createMenu(self):
        file_menu = self.menuBar().addMenu("File")
        file_menu.addAction(self.new_act)
        file_menu.addSeparator()
        file_menu.addAction(self.open_act)
        file_menu.addAction(self.save_act)
        file_menu.addSeparator()
        file_menu.addAction(self.quit_act)

        edit_menu = self.menuBar().addMenu("Edit")
        edit_menu.addAction(self.undo_act)
        edit_menu.addAction(self.redo_act)
        edit_menu.addSeparator()
        edit_menu.addAction(self.cut_act)
        edit_menu.addAction(self.copy_act)
        edit_menu.addAction(self.paste_act)
        edit_menu.addSeparator()
        edit_menu.addAction(self.find_act)

        tool_menu = self.menuBar().addMenu("Tools")
        tool_menu.addAction(self.font_act)
        tool_menu.addAction(self.color_act)
        tool_menu.addAction(self.highlight_act)

        help_menu = self.menuBar().addMenu("Help")
        help_menu.addAction(self.about_act)

    def clearText(self):

        answer = QMessageBox.question(self, "Obrisi tekst",
        "Da li zelite da obrisete tekst?",
        QMessageBox.StandardButton.No,
        QMessageBox.StandardButton.Yes)

        if answer == QMessageBox.StandardButton.Yes:
            self.text_edit.clear()

    def openFile(self):
        file_name, _ = QFileDialog.getOpenFileName(self, "Otvori",
                                "","HTML Files (*.html);;Text Files (*.txt)")
        if file_name:
            with open(file_name, "r") as f:
                notepad_text = f.read()
            self.text_edit.setText(notepad_text)

    def saveToFile(self):
        file_name, _ = QFileDialog.getSaveFileName(self, "Sacuvaj",
                                "", "HTML Files (*.html);;Text Files (*.txt)")
        if file_name.endswith(".txt"):
            notepad_text = self.text_edit.toPlainText()
            with open(file_name, "w") as f:
                f.write(notepad_text)
        elif file_name.endswith(".html"):
            notepad_richtext = self.text_edit.toHtml()
            with open(file_name, "w") as f:
                f.write(notepad_richtext)
        else:
            QMessageBox.information(self, "Nije sacuvan",
                                    "Tekst nije sacuvan.",
                                    QMessageBox.StandardButton.Ok)

    def searchText(self):
        find_text, ok = QInputDialog.getText(self, "Pretrazi tekst",
                                             "Pronasao:")
        if ok:
            extra_selections = []
            self.text_edit.moveCursor(QTextCursor.MoveOperation.Start)
            color = QColor(Qt.GlobalColor.gray)
            while(self.text_edit.find(find_text)):
                selection = QTextEdit.ExtraSelection()
                selection.format.setBackground(color)

                selection.cursor = self.text_edit.textCursor()
                extra_selections.append(selection)

            self.text_edit.setExtraSelections(extra_selections)

    def chooseFont(self):
        current = self.text_edit.currentFont()
        opt = QFontDialog.FontDialogOption.DontUseNativeDialog
        font, ok = QFontDialog.getFont(current, self, options=opt)
        if ok:
            self.text_edit.setCurrentFont(font)

    def chooseFontColor(self):
        color = QColorDialog.getColor()
        if color.isValid():
            self.text_edit.setTextColor(color)

    def chooseFontBackgroundColor(self):
        color = QColorDialog.getColor()
        if color.isValid():
            self.text_edit.setTextBackgroundColor(color)

    def aboutDialog(self):
        QMessageBox.about(self, "About Notepad",
        """<p>Razvoj notpepad aplikacije u PySide6 biblioteci</p>
        <p>Kontakt bla bla</p>""")

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

Zadatak 3. Napraviti aplikaciju koja dodaje QDockWidget. Dodati mogućnosti rada u režimu punog ekrana i dodati traku sa dodatnim informacijama.

```
import sys
from PySide6.QtWidgets import (QApplication, QMainWindow,
                            QWidget, QCheckBox, QTextEdit, QDockWidget,
                            QToolBar, QStatusBar, QVBoxLayout)
from PySide6.QtCore import Qt, QSize
from PySide6.QtGui import QIcon, QAction

class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.initializeUI()

    def initializeUI(self):
        self.setMinimumSize(450, 350)
        self.setWindowTitle("Dodavanje Docking Widget-a")
        self.setUpMainWindow()
        self.createDockWidget()
        self.createActions()
        self.createMenu()
        self.createToolBar()
        self.show()

    def setUpMainWindow(self):
        self.text_edit = QTextEdit()
        self.setCentralWidget(self.text_edit)

        self.setStatusBar(QStatusBar())

    def createActions(self):
        self.quit_act = QAction(QIcon("slike/exit.png"), "Quit")
        self.quit_act.setShortcut("Ctrl+Q")
        self.quit_act.setStatusTip("Quit program")
        self.quit_act.triggered.connect(self.close)
```

```python

        self.full_screen_act = QAction("Full Screen", checkable=True)

        self.full_screen_act.setStatusTip("Prebaci se na mod punog ekrana")
        self.full_screen_act.triggered.connect(self.switchToFullScreen)

    def createMenu(self):
        self.menuBar().setNativeMenuBar(False)
        file_menu = self.menuBar().addMenu("File")
        file_menu.addAction(self.quit_act)
        view_menu = self.menuBar().addMenu("View")
        appearance_submenu = view_menu.addMenu("Appearance")
        appearance_submenu.addAction(self.full_screen_act)

    def switchToFullScreen(self, state):
        if state: self.showFullScreen()
        else: self.showNormal()

    def createToolBar(self):
        toolbar = QToolBar("Main Toolbar")
        toolbar.setIconSize(QSize(16, 16))
        self.addToolBar(toolbar)
        toolbar.addAction(self.quit_act)

    def createDockWidget(self):
        dock_widget = QDockWidget()
        dock_widget.setWindowTitle("Lista")
        auto_bullet_cb = QCheckBox("Check box")
        auto_bullet_cb.toggled.connect(self.changeTextEditSettings)

        dock_v_box = QVBoxLayout()
        dock_v_box.addWidget(auto_bullet_cb)
        dock_v_box.addStretch(1)

        dock_container = QWidget()
        dock_container.setLayout(dock_v_box)

        dock_widget.setWidget(dock_container)

        self.addDockWidget(Qt.DockWidgetArea.LeftDockWidgetArea, dock_widget)

    def changeTextEditSettings(self, checked):

        if checked:
            print('CheckBox je odabran')
        else:
            print('CheckBox nije odabran')

```

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