import sys # Import the sys module to access system-specific parameters and functions

from PyQt5.QtWidgets import QApplication, QMainWindow, QWidget, QLabel, QPushButton, QGridLayout, QLineEdit, QMessageBox, QDialog, QFormLayout, QLineEdit, QSpinBox, QComboBox, QDialogButtonBox, QVBoxLayout, QGroupBox # Import various PyQt5 widgets and layouts

from PyQt5.QtGui import QPixmap, QPalette, QBrush, QFont # Import PyQt5 GUI classes for handling images, palettes, brushes, and fonts

import json # Import the json module for parsing JSON data

from PyQt5.QtCore import pyqtSignal # Import pyqtSignal for creating custom signals

from PyQt5.QtWidgets import QApplication, QMainWindow, QTreeWidget, QTreeWidgetItem, QVBoxLayout, QWidget, QLabel, QPushButton, QGridLayout, QLineEdit, QMessageBox # Duplicate import of QApplication, QMainWindow, and other widgets; these should be removed

from PyQt5.QtGui import QPixmap, QPalette, QBrush, QFont # Duplicate import of QPixmap, QPalette, QBrush, and QFont; these should be removed

from PyQt5.QtWidgets import QApplication, QWidget, QVBoxLayout, QPushButton, QFileDialog # Duplicate import of QApplication, QWidget, QVBoxLayout, QPushButton, and QFileDialog; these should be removed

from PyQt5.QtMultimedia import QMediaPlayer, QMediaContent # Import PyQt5 multimedia classes for playing media content

from PyQt5.QtMultimediaWidgets import QVideoWidget # Import QVideoWidget for displaying video content

from PyQt5.QtCore import Qt, QUrl # Import Qt and QUrl for handling URLs and other core functionalities

import os # Import the os module for interacting with the operating system

import re # Import the re module for regular expression operations

from imageai.Detection import VideoObjectDetection

class VideoPlayer(*QWidget*): # Define a custom QWidget for playing videos

    def \_\_init\_\_(*self*):

*super*().\_\_init\_\_()

*self*.initUI() # Initialize the user interface

    def initUI(*self*):

*self*.setWindowTitle('Video Player') # Set the window title

*self*.setGeometry(300, 300, 300, 200) # Set the window geometry

*self*.layout = QVBoxLayout() # Create a vertical layout

*self*.setLayout(*self*.layout) # Set the layout for the widget

*self*.videoWidget = QVideoWidget() # Create a QVideoWidget for displaying video content

*self*.layout.addWidget(*self*.videoWidget) # Add the QVideoWidget to the layout

*self*.playButton = QPushButton('Choose Video', *self*) # Create a QPushButton for selecting a video

*self*.playButton.clicked.connect(*self*.playVideo) # Connect the button's clicked signal to the playVideo method

*self*.layout.addWidget(*self*.playButton) # Add the QPushButton to the layout

*self*.pauseButton = QPushButton('Play/Pause Video', *self*) # Create a QPushButton for playing/pausing the video

*self*.pauseButton.clicked.connect(*self*.pauseVideo) # Connect the button's clicked signal to the pauseVideo method

*self*.layout.addWidget(*self*.pauseButton) # Add the QPushButton to the layout

*self*.playSpecificVideoButton = QPushButton('Convert Video', *self*) # Create a QPushButton for converting a video

*self*.playSpecificVideoButton.clicked.connect(*self*.playSpecificVideo) # Connect the button's clicked signal to the playSpecificVideo method

*self*.layout.addWidget(*self*.playSpecificVideoButton) # Add the QPushButton to the layout

*self*.player = QMediaPlayer(None, QMediaPlayer.VideoSurface) # Create a QMediaPlayer for playing media content

*self*.player.setVideoOutput(*self*.videoWidget) # Set the video output for the QMediaPlayer

    def playVideo(*self*):

        options = QFileDialog.Options()

        options |= QFileDialog.ReadOnly

        fileName, \_ = QFileDialog.getOpenFileName(*self*, "Choose Video", "", "Videos (\*.mp4 \*.avi \*.mkv);;All Files (\*)", *options*=options) # Open a file dialog to select a video file

        if fileName:

*self*.player.setMedia(QMediaContent(QUrl.fromLocalFile(fileName))) # Set the media content for the QMediaPlayer

*self*.player.play() # Play the video

    def pauseVideo(*self*):

        if *self*.player.state() == QMediaPlayer.PlayingState:

*self*.player.pause() # Pause the video if it's currently playing

        else:

*self*.player.play() # Play the video if it's currently paused

    # Assuming you have a VideoObjectDetection class defined elsewhere

    detector = VideoObjectDetection() # Create an instance of the VideoObjectDetection class

    detector.setModelTypeAsYOLOv3() # Set the model type to YOLOv3

    detector.setModelPath(os.path.join(execution\_path, "yolov3.pt")) # Set the path to the YOLOv3 model file

    detector.loadModel() # Load the model

    # Specify the input and output file paths for video detection

    input\_file\_path = os.path.join(execution\_path, fileName) # Path to the input video file

    output\_file\_path = os.path.join(execution\_path, "video\_detected.avi") # Path to the output video file

    # Detect objects in the video and save the result

    detector.detectObjectsFromVideo(*input\_file\_path*=input\_file\_path,

*output\_file\_path*=output\_file\_path,

*frames\_per\_second*=20, *log\_progress*=True) # Detect objects in the video and save the result

    def playSpecificVideo(*self*):

*self*.player.setMedia(QMediaContent(QUrl.fromLocalFile(output\_file\_path))) # Set the media content for the QMediaPlayer

*self*.player.play() # Play the video

class SignUpWindow(*QWidget*): # Define a custom QWidget for the sign-up process

    windowClosed = pyqtSignal() # Define a custom signal for when the window is closed

    def \_\_init\_\_(*self*):

*super*().\_\_init\_\_()

*self*.setWindowTitle('Sign Up') # Set the window title

*self*.resize(350, 200) # Set the window size

        layout = QGridLayout() # Create a grid layout

        label\_email = QLabel('Email') # Create a QLabel for the email input

*self*.email\_input = QLineEdit() # Create a QLineEdit for the email input

        layout.addWidget(label\_email, 0, 0) # Add the QLabel to the layout

        layout.addWidget(*self*.email\_input, 0, 1) # Add the QLineEdit to the layout

        label\_password = QLabel('Password') # Create a QLabel for the password input

*self*.password\_input = QLineEdit() # Create a QLineEdit for the password input

        layout.addWidget(label\_password, 1, 0) # Add the QLabel to the layout

        layout.addWidget(*self*.password\_input, 1, 1) # Add the QLineEdit to the layout

        button\_signup = QPushButton('Sign Up') # Create a QPushButton for the sign-up action

        layout.addWidget(button\_signup, 2, 0, 2, 2) # Add the QPushButton to the layout

*self*.setStyleSheet("background-color: #f0f0f0;") # Set the window's background color

        label\_email.setStyleSheet("font-size: 14px;") # Set the font size for the email QLabel

        label\_password.setStyleSheet("font-size: 14px;") # Set the font size for the password QLabel

*self*.setLayout(layout) # Set the layout for the widget

        button\_signup.clicked.connect(*self*.handleSignUp) # Connect the button's clicked signal to the handleSignUp method

    def handleSignUp(*self*):

        email = *self*.email\_input.text() # Get the text from the email QLineEdit

        password = *self*.password\_input.text() # Get the text from the password QLineEdit

        email\_regex = r'*\b*[A-Za-z0-9.\_%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,}*\b*' # Define a regular expression for validating email addresses

        if not re.match(email\_regex, email):

            QMessageBox.warning(*self*, "Invalid Email", "Please enter a valid email address.") # Show a warning message if the email is invalid

            return

        if not password:

            QMessageBox.warning(*self*, "Empty Password", "Please enter a password.") # Show a warning message if the password is empty

            return

*self*.user\_info = {'email': email, 'password': password} # Store the user's email and password

        QMessageBox.information(*self*, "Sign Up Successful", "Your account has been created successfully.") # Show an information message indicating successful sign-up

*self*.close() # Close the sign-up window

    def closeEvent(*self*, *event*):

*self*.windowClosed.emit() # Emit the windowClosed signal when the window is closed

*super*().closeEvent(event) # Call the superclass's closeEvent method

# Define a custom QDialog for the user profile

class ProfileWindow(*QDialog*):

    # Initialize the ProfileWindow with optional user\_info

    def \_\_init\_\_(*self*, *user\_info*=None):

*super*().\_\_init\_\_()

        # Set the window title to "User Profile"

*self*.setWindowTitle("User Profile")

        # Set the window geometry (position and size)

*self*.setGeometry(100, 100, 300, 400)

        # Create a QGroupBox for user information

*self*.formGroupBox = QGroupBox("User Information")

        # Create a QLineEdit for the user's name

*self*.nameLineEdit = QLineEdit()

        # Create a QSpinBox for the user's age

*self*.ageSpinBar = QSpinBox()

        # Create a QComboBox for the user's degree

*self*.degreeComboBox = QComboBox()

        # Add degree options to the QComboBox

*self*.degreeComboBox.addItems(["BTech", "MTech", "PhD"])

        # Create a QLineEdit for the user's email

*self*.emailLineEdit = QLineEdit()

        # Create a QLineEdit for the user's phone number

*self*.phoneLineEdit = QLineEdit()

        # Create a QLineEdit for the user's address

*self*.addressLineEdit = QLineEdit()

        # Call the createForm method to set up the form layout

*self*.createForm()

        # Create a QDialogButtonBox with OK and Cancel buttons

*self*.buttonBox = QDialogButtonBox(QDialogButtonBox.Ok | QDialogButtonBox.Cancel)

        # Connect the OK button to the updateProfile method

*self*.buttonBox.accepted.connect(*self*.updateProfile)

        # Connect the Cancel button to the reject method

*self*.buttonBox.rejected.connect(*self*.reject)

        # Create a QVBoxLayout for the main layout

        mainLayout = QVBoxLayout()

        # Add the form group box to the main layout

        mainLayout.addWidget(*self*.formGroupBox)

        # Add the button box to the main layout

        mainLayout.addWidget(*self*.buttonBox)

        # Set the main layout for the window

*self*.setLayout(mainLayout)

        # If user\_info is provided, load it; otherwise, load from file

        if user\_info:

*self*.loadUserInfo(user\_info)

        else:

*self*.loadUserInfoFromFile()

    # Method to create the form layout

    def createForm(*self*):

        layout = QFormLayout()

        # Add form fields to the layout

        layout.addRow(QLabel("Name"), *self*.nameLineEdit)

        layout.addRow(QLabel("Email"), *self*.emailLineEdit)

        layout.addRow(QLabel("Phone"), *self*.phoneLineEdit)

        layout.addRow(QLabel("Address"), *self*.addressLineEdit)

        layout.addRow(QLabel("Degree"), *self*.degreeComboBox)

        layout.addRow(QLabel("Age"), *self*.ageSpinBar)

        # Set the layout for the form group box

*self*.formGroupBox.setLayout(layout)

    # Method to update the user profile

    def updateProfile(*self*):

        # Collect user information from the form

*self*.user\_info = {

            'name': *self*.nameLineEdit.text(),

            'email': *self*.emailLineEdit.text(),

            'phone': *self*.phoneLineEdit.text(),

            'address': *self*.addressLineEdit.text(),

            'degree': *self*.degreeComboBox.currentText(),

            'age': *self*.ageSpinBar.value()

        }

        # Save the user information

*self*.saveUserInfo()

        # Close the dialog

*self*.close()

    def saveUserInfo(*self*):

    # Attempt to save user information to a JSON file

        try:

            with open('user\_info.json', 'w') as file:

                json.dump(*self*.user\_info, file)

            # Display a success message if the save is successful

            QMessageBox.information(*self*, "Success", "Profile information saved successfully.")

        except *IOError* as e:

            # Display an error message if the save fails

            QMessageBox.critical(*self*, "Error", f"Failed to save profile information: {e}")

    def loadUserInfo(*self*, *user\_info*):

        # Populate the form fields with the provided user information

*self*.nameLineEdit.setText(user\_info.get('name', ''))

*self*.emailLineEdit.setText(user\_info.get('email', ''))

*self*.phoneLineEdit.setText(user\_info.get('phone', ''))

*self*.addressLineEdit.setText(user\_info.get('address', ''))

*self*.degreeComboBox.setCurrentText(user\_info.get('degree', ''))

*self*.ageSpinBar.setValue(user\_info.get('age', 0))

    def loadUserInfoFromFile(*self*):

        # Attempt to load user information from a JSON file

        try:

            with open('user\_info.json', 'r') as file:

                user\_info = json.load(file)

            # Load the user information into the form

*self*.loadUserInfo(user\_info)

        except (*IOError*, json.JSONDecodeError):

            # Display a warning if the file does not exist or is not valid JSON

            QMessageBox.warning(*self*, "Warning", "No saved profile information found.")

    def \_\_init\_\_(*self*, *user\_info*=None):

        # Initialize the ProfileWindow with optional user information

*super*().\_\_init\_\_()

*self*.setWindowTitle("User Profile")

*self*.setGeometry(100, 100, 300, 400)

*self*.formGroupBox = QGroupBox("User Information")

*self*.nameLineEdit = QLineEdit()

*self*.ageSpinBar = QSpinBox()

*self*.degreeComboBox = QComboBox()

*self*.degreeComboBox.addItems(["BTech", "MTech", "PhD"])

*self*.emailLineEdit = QLineEdit()

*self*.phoneLineEdit = QLineEdit()

*self*.addressLineEdit = QLineEdit()

*self*.createForm()

*self*.buttonBox = QDialogButtonBox(QDialogButtonBox.Ok | QDialogButtonBox.Cancel)

*self*.buttonBox.accepted.connect(*self*.updateProfile)

*self*.buttonBox.rejected.connect(*self*.reject)

        mainLayout = QVBoxLayout()

        mainLayout.addWidget(*self*.formGroupBox)

        mainLayout.addWidget(*self*.buttonBox)

*self*.setLayout(mainLayout)

        if user\_info:

*self*.loadUserInfo(user\_info)

    def createForm(*self*):

        # Create the form layout with labels and input fields

        layout = QFormLayout()

        layout.addRow(QLabel("Name"), *self*.nameLineEdit)

        layout.addRow(QLabel("Email"), *self*.emailLineEdit)

        layout.addRow(QLabel("Phone"), *self*.phoneLineEdit)

        layout.addRow(QLabel("Address"), *self*.addressLineEdit)

        layout.addRow(QLabel("Degree"), *self*.degreeComboBox)

        layout.addRow(QLabel("Age"), *self*.ageSpinBar)

*self*.formGroupBox.setLayout(layout)

    def updateProfile(*self*):

        # Update the user information based on the form inputs

*self*.user\_info = {

            'name': *self*.nameLineEdit.text(),

            'email': *self*.emailLineEdit.text(),

            'phone': *self*.phoneLineEdit.text(),

            'address': *self*.addressLineEdit.text(),

            'degree': *self*.degreeComboBox.currentText(),

            'age': *self*.ageSpinBar.value()

        }

*self*.saveUserInfo()

*self*.close()

    def saveUserInfo(*self*):

        # Save the user information to a JSON file

        with open('user\_info.json', 'w') as file:

            json.dump(*self*.user\_info, file)

    def loadUserInfo(*self*, *user\_info*):

        # Populate the form fields with the provided user information

*self*.nameLineEdit.setText(user\_info.get('name', ''))

*self*.emailLineEdit.setText(user\_info.get('email', ''))

*self*.phoneLineEdit.setText(user\_info.get('phone', ''))

*self*.addressLineEdit.setText(user\_info.get('address', ''))

*self*.degreeComboBox.setCurrentText(user\_info.get('degree', ''))

*self*.ageSpinBar.setValue(user\_info.get('age', 0))

class HomePage(*QMainWindow*):

    def \_\_init\_\_(*self*):

        # Initialize the HomePage window

*super*().\_\_init\_\_()

*self*.setWindowTitle("Home Page")

*self*.setGeometry(100, 100, 800, 600)

*self*.background = QPixmap(r'C:\Users\user*\D*esktop\11')

*self*.setBackgroundImage()

*self*.welcomeLabel = QLabel("Welcome to Where is Ai", *self*)

*self*.welcomeLabel.setGeometry(300, 200, 200, 50)

        font = QFont()

        font.setBold(True)

        font.setPointSize(20)

*self*.welcomeLabel.setFont(font)

*self*.signUpButton = QPushButton("Sign Up", *self*)

*self*.signUpButton.move(400, 500)

*self*.signUpButton.clicked.connect(*self*.openSignUpWindow)

*self*.quitButton = QPushButton("Quit", *self*)

*self*.quitButton.move(500, 500)

*self*.quitButton.clicked.connect(*self*.close)

*self*.videoButton = QPushButton("video", *self*)

*self*.videoButton.move(300, 500)

*self*.videoButton.clicked.connect(*self*.openvideoWindow)

*self*.profileButton = QPushButton("Profile", *self*)

*self*.profileButton.move(200, 500)

*self*.profileButton.clicked.connect(*self*.openProfileWindow)

        # Initialize signUpWindow attribute

*self*.signUpWindow = None

    def openProfileWindow(*self*):

        # Check if user\_info.json exists and is not empty

        try:

            with open('user\_info.json', 'r') as file:

                user\_info = json.load(file)

*self*.profileWindow = ProfileWindow(user\_info)

*self*.profileWindow.show()

        except (*IOError*, json.JSONDecodeError):

            # If user\_info.json does not exist or is empty, prompt to sign up

            QMessageBox.warning(*self*, "Warning", "Please sign up first.")

*self*.openSignUpWindow()

        # Initialize signUpWindow attribute

*self*.signUpWindow = None

    def setBackgroundImage(*self*):

        # Set the background image for the HomePage window

        palette = QPalette()

        palette.setBrush(QPalette.Background, QBrush(*self*.background))

*self*.setPalette(palette)

    def openSignUpWindow(*self*):

        # Open the SignUpWindow

*self*.signUpWindow = SignUpWindow()

*self*.signUpWindow.show()

*self*.signUpWindow.windowClosed.connect(*self*.openProfileWindow)

    def openvideoWindow(*self*):

        # Open the VideoPlayer window

*self*.openvideoWindow = VideoPlayer()

*self*.openvideoWindow.show()

    def openProfileWindow(*self*):

        # Check if signUpWindow is not None and has user\_info

        if *self*.signUpWindow and hasattr(*self*.signUpWindow, 'user\_info'):

*self*.profileWindow = ProfileWindow(*self*.signUpWindow.user\_info)

*self*.profileWindow.show()

        else:

            # Handle case where signUpWindow is None or user\_info is not set

            QMessageBox.warning(*self*, "Error", "User information not available.")

if \_\_name\_\_ == '\_\_main\_\_':

    # Initialize the QApplication and the HomePage window

    app = QApplication(sys.argv)

    homePage = HomePage()

    homePage.show()

    sys.exit(app.exec\_())