```
import os
import sys
import threading
import datetime
import time
import requests
import pyttsx3
import speech_recognition as sr
import openai
import cv2
import face_recognition
import numpy as np
from dotenv import load_dotenv
from PyQt5.QtWidgets import QMainWindow, QApplication, QLabel,
QVBoxLayout, QWidget, QFrame
from PyQt5.QtCore import Qt, QTimer, pyqtSignal, pyqtSlot
from PyQt5.QtGui import QFont, QColor

# --- LEVEL 10 SECURITY CONFIGURATION ---
load_dotenv('credentials.env')
openai.api_key = os.getenv("OPENAI_API_KEY")

STRICT_DISTANCE_LIMIT = 0.15  # Absolute Zero Threshold
VOICE_PASSCODE = "alpha delta nine"
LOCATION = "Vevay, Indiana"

class JarvisMarkIIIv10(QMainWindow):
    # Signals for thread-safe UI updates
    update_status_signal = pyqtSignal(str, str) # text, color
    update_distance_signal = pyqtSignal(float)

    def __init__(self):
        super().__init__()
        self.engine = pyttsx3.init()
        self.setup_voice()
        self.is_authenticated = False
        self.last_face_time = time.time()

        # UI Setup
        self.init_ui()

        # Load Master Reference
        if not os.path.exists("me.jpg"):
            print("CRITICAL: me.jpg not found.")
            sys.exit(1)
        self.master_img = face_recognition.load_image_file("me.jpg")
        self.master_enc =
face_recognition.face_encodings(self.master_img)[0]
```

```
        # Connect Signals
        self.update_status_signal.connect(self.set_ui_status)
        self.update_distance_signal.connect(self.set_ui_distance)

        # Start Security Thread
        threading.Thread(target=self.security_protocol,
daemon=True).start()

    def init_ui(self):
        self.setWindowFlags(Qt.FramelessWindowHint |
Qt.WindowStaysOnTopHint)
        self.setAttribute(Qt.WA_TranslucentBackground)
        self.setGeometry(50, 50, 400, 250)

        self.central_widget = QWidget()
        self.layout = QVBoxLayout()

        self.frame = QFrame()
        self.frame.setStyleSheet("background-color: rgba(5, 5, 5,
230); border: 1px solid #ff0000; border-radius: 5px;")
        self.frame_layout = QVBoxLayout()

        self.title_lbl = QLabel("J.A.R.V.I.S. MARK III [v.10]")
        self.title_lbl.setStyleSheet("color: #ff0000; font-family:
'Consolas'; font-size: 14px; font-weight: bold;")

        self.status_lbl = QLabel("SYSTEM: LOCKED")
        self.status_lbl.setStyleSheet("color: #ff3333; font-family:
'Consolas'; font-size: 18px;")

        self.distance_lbl = QLabel("DISTANCE: 0.0000")
        self.distance_lbl.setStyleSheet("color: #666; font-family:
'Consolas'; font-size: 12px;")

        self.directive_lbl = QLabel(f"DIRECTIVE: HONEST / NON-BIASED /
{LOCATION}")
        self.directive_lbl.setStyleSheet("color: #00f2ff; font-family:
'Consolas'; font-size: 10px;")

        self.frame_layout.addWidget(self.title_lbl)
        self.frame_layout.addWidget(self.status_lbl)
        self.frame_layout.addWidget(self.distance_lbl)
        self.frame_layout.addWidget(self.directive_lbl)
        self.frame.setLayout(self.frame_layout)

        self.layout.addWidget(self.frame)
        self.central_widget.setLayout(self.layout)
```

```
        self.setCentralWidget(self.central_widget)

    @pyqtSlot(str, str)
    def set_ui_status(self, text, color):
        self.status_lbl.setText(f"SYSTEM: {text}")
        self.status_lbl.setStyleSheet(f"color: {color}; font-family:
'Consolas'; font-size: 18px;")
        self.frame.setStyleSheet(f"background-color: rgba(5, 5, 5,
230); border: 1px solid {color}; border-radius: 5px;")

    @pyqtSlot(float)
    def set_ui_distance(self, dist):
        self.distance_lbl.setText(f"DISTANCE: {dist:.4f}")
        color = "#00ff00" if dist <= STRICT_DISTANCE_LIMIT else
"#ff3333"
        self.distance_lbl.setStyleSheet(f"color: {color}; font-family:
'Consolas'; font-size: 12px;")

    def setup_voice(self):
        voices = self.engine.getProperty('voices')
        for v in voices:
            if "UK" in v.name or "Hazel" in v.name:
                self.engine.setProperty('voice', v.id)
                break
        self.engine.setProperty('rate', 195)

    def speak(self, text):
        print(f"JARVIS: {text}")
        self.engine.say(text)
        self.engine.runAndWait()

    def listen(self):
        r = sr.Recognizer()
        with sr.Microphone() as source:
            r.adjust_for_ambient_noise(source, duration=0.8)
            audio = r.listen(source)
        try:
            return r.recognize_google(audio).lower()
        except:
            return ""

    def security_protocol(self):
        """Level 10: Absolute Zero Biometrics"""
        self.update_status_signal.emit("AUTHENTICATING", "#ffcc00")
        self.speak("Initiating Level Ten Security. Absolute Zero
Protocol engaged.")

        cap = cv2.VideoCapture(0)
```

```
        # Step 1: Facial Match (0.15 Limit)
        authenticated = False
        start_time = time.time()

        while time.time() - start_time < 15:
            ret, frame = cap.read()
            if not ret: continue

            rgb_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
            encodings = face_recognition.face_encodings(rgb_frame)

            if encodings:
                dist =
face_recognition.face_distance([self.master_enc], encodings[0])[0]
                self.update_distance_signal.emit(dist)

                if dist <= STRICT_DISTANCE_LIMIT:
                    authenticated = True
                    break
            else:
                self.update_distance_signal.emit(1.0)

        if not authenticated:
            self.speak("Identity mismatch. Level Ten lockout
initiated.")
            os._exit(0)

        # Step 2: Voice Passcode
        self.update_status_signal.emit("PASSCODE REQ", "#ffcc00")
        self.speak("Visual match confirmed. State Level Ten passcode,
Sir.")

        if VOICE_PASSCODE in self.listen():
            self.is_authenticated = True
            self.update_status_signal.emit("AUTHORIZED", "#00f2ff")
            self.speak("Access granted. Neural bridge established.
Welcome home, Sir.")
            self.start_persistence_monitor(cap)
            self.main_brain_loop()
        else:
            self.speak("Invalid passcode. Terminal shutdown.")
            os._exit(0)

    def start_persistence_monitor(self, cap):
        """Locks system if face leaves the frame."""
        def monitor():
            while self.is_authenticated:
```

```
                ret, frame = cap.read()
                rgb_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
                encs = face_recognition.face_encodings(rgb_frame)

                if not encs:
                    if time.time() - self.last_face_time > 1.5:
                        self.update_status_signal.emit("PERSISTENCE
LOST", "#ff0000")
                        self.speak("User presence lost. Automatic core
lockout.")
                        os._exit(0)
                else:
                    self.last_face_time = time.time()
                time.sleep(0.5)

        threading.Thread(target=monitor, daemon=True).start()

    def main_brain_loop(self):
        while True:
            query = self.listen()
            if not query: continue

            if "shutdown" in query or "go to sleep" in query:
                self.speak("Disconnecting. Goodnight, Sir.")
                os._exit(0)

            try:
                # INTEGRATED HONESTY DIRECTIVE
                response = openai.ChatCompletion.create(
                    model="gpt-4o",
                    messages=[
                        {"role": "system", "content": f"""You are
J.A.R.V.I.S. Mark III.
                        - Location: {LOCATION}.
                        - Directive: Regarding government and
politics, you are strictly HONEST, UNBIASED, and NON-PARTISAN.
                        - Rule: Provide objective facts only. Do not
favor any party or ideology.
                        - Format: Be extremely short and concise.
Refer to the user as Sir."""},
                        {"role": "user", "content": query}
                    ]
                )
                self.speak(response.choices[0].message.content)
            except Exception as e:
                self.speak("Neural link failure, Sir.")

if __name__ == "__main__":
```

```
app = QApplication(sys.argv)
jarvis_app = JarvisMarkIIIv10()
jarvis_app.show()
sys.exit(app.exec_())
```