PHASE 5: STURCTURAL HEALTH MONITORING

COLLEGE CODE : 1101

COLLEGE NAME : Aalim Muhammed Salegh College Of Engineering

DEPARTMENT : COMPUTER SCIENCE AND ENGINEERING

STUDENT NM-ID : 21665fc543a5c29a5fd4daaf67acc870

ROLL NO : 110123104109

DATE : 07-05-2025

Completed the project named as phase 5

TECHNOLOGY PROJECT NAME: STRUCTURAL HEALTH MONITORING

SUBMITTED BY

NAME : A SYED RIZWAN

MOBILE NO : 9342561101

MEMBERS	USER ID
A SYED RIZWAN	21665fc543a5c29a5fd4daaf67acc870
S MOHAMMED SALEEM	2f37afc5d42e933a6454d87f4ce9cbd2
B SYED SABIRUDEEN	267f206b32f8c92ba4147b7753193055
M SIVA	b41e46b8139d7c881cd42bbfd0d30801
A SUJATH ALI	880addfa3ab909fafb6490dac0cb09fd

Abstract

This project introduces a real-time, AI-powered Structural Health Monitoring (SHM) system aimed at enhancing infrastructure safety through intelligent analysis of sensor data. As infrastructure ages or faces varying environmental and operational stresses, the need for continuous health monitoring becomes critical. The system presented here addresses this need by utilizing strain and vibration data collected from sensors and stored in CSV format, which is then analyzed using Google's advanced Gemini AI model. The model classifies the structural condition into three key categories—**normal**, **moderate**, and **critical**—based on predefined thresholds and intelligent pattern recognition, helping stakeholders make informed decisions about maintenance and risk mitigation.

The application is developed in Python, incorporating a user-friendly Graphical User Interface (GUI) built with Tkinter to display real-time data updates, health status indicators, and alerts. The system is equipped with audio alerts that automatically notify users in case of critical stress levels, ensuring quick response to potential structural failures. To maintain efficiency and responsiveness, the application uses **threading**, allowing concurrent data analysis, visualization, and user interaction without compromising performance.

Moreover, the application supports external API integration, allowing for scalability and real-world deployment, such as interfacing with remote servers or cloud dashboards for broader infrastructure management. Despite being a compact prototype, this system is designed with scalability and modularity in mind, making it suitable for future enhancements like multi-sensor support, cloud-based analytics, and mobile notifications.

Overall, this project serves as a practical demonstration of how AI can be effectively integrated with structural monitoring systems to provide real-time, automated insights, ensuring safety, reducing downtime, and supporting proactive infrastructure maintenance.

Index

1. Project Demonstration

Overview:

The SHM system is demonstrated live, showing its key features and real-time decision-making capabilities.

Demonstration Details:

System Walkthrough: Displays a clean GUI showing current strain and vibration readings with

clear visual and audio alerts.

AI Analysis: Uses Gemini AI to assess safety status and categorize it.

Sound Alerts: Critical = MP3 alert; Moderate = System beep; Normal = No alert.

Real-Time Interaction: Sensor data updates and threading used for API calls.

Outcome:

Demonstrates the ability to read and interpret real sensor data, interact with AI for safety classification, and alert users visually and audibly.

2. Project Documentation

Overview:

Full documentation is included for easy understanding and future scaling.

Documentation Sections:

System Architecture:

Explains the sensor input, Gemini API interaction, and Tkinter-based UI flow.

Code Documentation

Python code comments detail the workflow for sensor loading, AI prompting, alert

User Guide:

Explains how to start/stop monitoring, interpret alerts, and update the CSV file with new sensor data.

Admin Guide:

Covers API key setup (.env), external package installation (pygame, requests, dotenv), and troubleshooting tips.

Testing Reports:

System was tested with varying sensor values to verify all three alert levels work reliably, and GUI remains responsive under load.

Outcome:

The project is fully documented for replication or extension.

3. Feedback and Final Adjustments

Overview:

Feedback was taken from peers and mentors.

Steps:

Collected via live testing:

UI clarity, responsiveness, and alert effectiveness were positively noted.

Refinements Made:

- Delay added to alert sound to avoid continuous looping.
- Added threading to prevent UI freeze during API calls.

Final Testing:

Confirmed proper display of all status levels, responsive alerts, and Gemini API integration without crashes.

Outcome:

Refinements improved usability and reliability. The app is ready for educational or prototyping use in real-world SHM systems.

4. Final Project Report Submission

Overview:

Final project report outlines:

Report Sections:

Executive Summary

- A compact and AI-integrated SHM prototype using real sensor data. **Phase Breakdown**

- Data handling, GUI creation, API usage, alert mechanism, multithreading.

Challenges & Solutions

- **Challenge:** UI freezing during API wait time.
- **Solution:** Added `threading.Thread` for analysis.
- **Challenge:** Proper audio alert playback.
- **Solution:** Integrated `pygame` with MP3 support.

Outcomes:

A modular, real-time safety assessment tool for civil structures.

Outcome:

The report encapsulates the end-to-end development and testing of the system, demonstrating both technical depth and practical application.

5. Project Handover and Future Works

Overview:

For future scaling and improvement.

Next Steps:

- Real-time integration with IoT sensors instead of CSV.
- Cloud-based alert system with SMS/email notifications.
- Historical data analysis for predictive maintenance.
- Mobile app version of the GUI for on-site engineers.
- Support for more complex AI analysis with structural diagrams.

Outcome:

Ready for handover with roadmap for future development.

PROGRAM:

Welcome To Colab Cannot save changes

File Edit View Insert Runtime Tools Help

nands + Code + Text Copy to Drive

```python
import tkinter as tk
import winsound
from tkinter import messagebox
import csv
import requests
import os
import platform
import threading
from dotenv import load_dotenv
import pygame


load_dotenv()

# Load sensor data from CSV
def load_sensor_dataset(file_path):
    try:
        with open(file_path, mode='r') as file:
            reader = csv.DictReader(file)
            return [{"strain": float(row["strain"]), "vibration": float(row["vibration"])} for row in reader]
    except FileNotFoundError:
        messagebox.showerror("File Error", "sensor_data.csv not found.")
        return []

# Analyze using Gemini
def analyze_with_gemini(data):
    rules = (
        "If strain > 0.8 and vibration > 0.2, then alert: 'Critical stress'. "
        "If strain > 0.5 and vibration > 0.05, then alert: 'Moderate stress'. "
        "Otherwise, alert: 'Normal operation'."
```

0s completed at 10:35 AM

```python
    )

    prompt = f"{rules} Analyze the following sensor data and tell me the status: strain = {data['strain']}, vibration = {data['vibration']}."

    url = "https://generativelanguage.googleapis.com/v1beta/models/gemini-1.5-flash-latest:generateContent"
    headers = {
        "Content-Type": "application/json",
        "x-goog-api-key": os.getenv('API_KEY')
    }

    messages = [{"role": "user", "parts": [{"text": prompt}]}]

    try:
        response = requests.post(url, headers=headers, json={"contents": messages})
        result = response.json()
        reply = result["candidates"][0]["content"]["parts"][0]["text"]
        return reply
    except Exception as e:
        return f"Error: {e}"

# Alert sound (using sound file)
def play_alert(level):
    alert_file = "emergency-alarm-69780.mp3"  # Path to your MP3 file

    pygame.mixer.init()  # Initialize the mixer module
    pygame.mixer.music.load(alert_file)  # Load the MP3 file

    if level == "Critical":
        pygame.mixer.music.play(-1, 0.0)  # Play the alert sound in a loop (if needed)
        pygame.time.delay(5000)  # Wait for 5 seconds
        pygame.mixer.music.stop()  # Stop the sound
```

0s completed at 10:35 AM

```
    elif level == "Moderate":
        if platform.system()=="Windows":
            winsound.Beep(800,1500)
    else:
        pass

class SensorApp:
    def __init__(self, root):
        self.root = root
        self.root.title("🧠 Structural Health Monitoring")
        self.root.geometry("700x500")
        self.root.configure(bg="#1e1e2f")
        self.root.resizable(False, False)  # Fixed size

        self.index = 0
        self.running = False
        self.data = load_sensor_dataset("sensor_data.csv")

        # Styling
        self.font_title = ("Helvetica", 20, "bold")
        self.font_text = ("Helvetica", 13)
        self.bg_color = "#1e1e2f"
        self.fg_color = "#ffffff"
        self.accent = "#00ffc8"

        # Main Center Frame
        self.main_frame = tk.Frame(self.root, bg=self.bg_color, width=700, height=500)
        self.main_frame.place(relx=0.5, rely=0.5, anchor="center")

        # Title
        self.title_label = tk.Label(
```

0s completed at 10:35 AM

```
            self.main_frame, text="🔧 Structural Health Monitor", font=self.font_title,
            fg=self.accent, bg=self.bg_color
        )
        self.title_label.pack(pady=20)

        # Sensor Data Display
        self.sensor_display = tk.Label(
            self.main_frame, text="Sensor Data: ", font=self.font_text,
            fg=self.fg_color, bg=self.bg_color
        )
        self.sensor_display.pack(pady=10)

        # Response Display
        self.response_display = tk.Label(
            self.main_frame, text="Waiting for input...", font=self.font_text,
            fg=self.fg_color, bg=self.bg_color, wraplength=600, justify="center",
            width=60, height=4  # Fixed dimensions
        )
        self.response_display.pack(pady=10)

        # Status Indicator
        self.status_label = tk.Label(
            self.main_frame, text="", font=("Helvetica", 14, "bold"),
            bg=self.bg_color
        )
        self.status_label.pack(pady=10)

        # Buttons
        self.button_frame = tk.Frame(self.main_frame, bg=self.bg_color)
        self.button_frame.pack(pady=20)
```

0s completed at 10:35 AM

```
        self.start_btn = tk.Button(
            self.button_frame, text="▶ Start Monitoring", width=18, bg="#2aefaa",
            font=self.font_text, command=self.start_monitoring
        )
        self.start_btn.grid(row=0, column=0, padx=10)

        self.stop_btn = tk.Button(
            self.button_frame, text="■ Stop Monitoring", width=18, bg="#ff6f61",
            font=self.font_text, command=self.stop_monitoring
        )
        self.stop_btn.grid(row=0, column=1, padx=10)

    # Step 1: Show data
    def update_ui(self):
        if self.running and self.index < len(self.data):
            current_data = self.data[self.index]

            self.sensor_display.config(
                text=f"Sensor Data: Strain = {current_data['strain']} | Vibration = {current_data['vibration']}"
            )
            self.response_display.config(text="🔄 Processing...")


            self.root.after(300, lambda: self.process_analysis(current_data))
        elif self.index >= len(self.data):
            self.status_label.config(text="✅ Monitoring complete.", fg="#00ffc8")
            self.running = False

    # Step 2: Analyze after processing
    def process_analysis(self, current_data):
        def run_analysis():
```

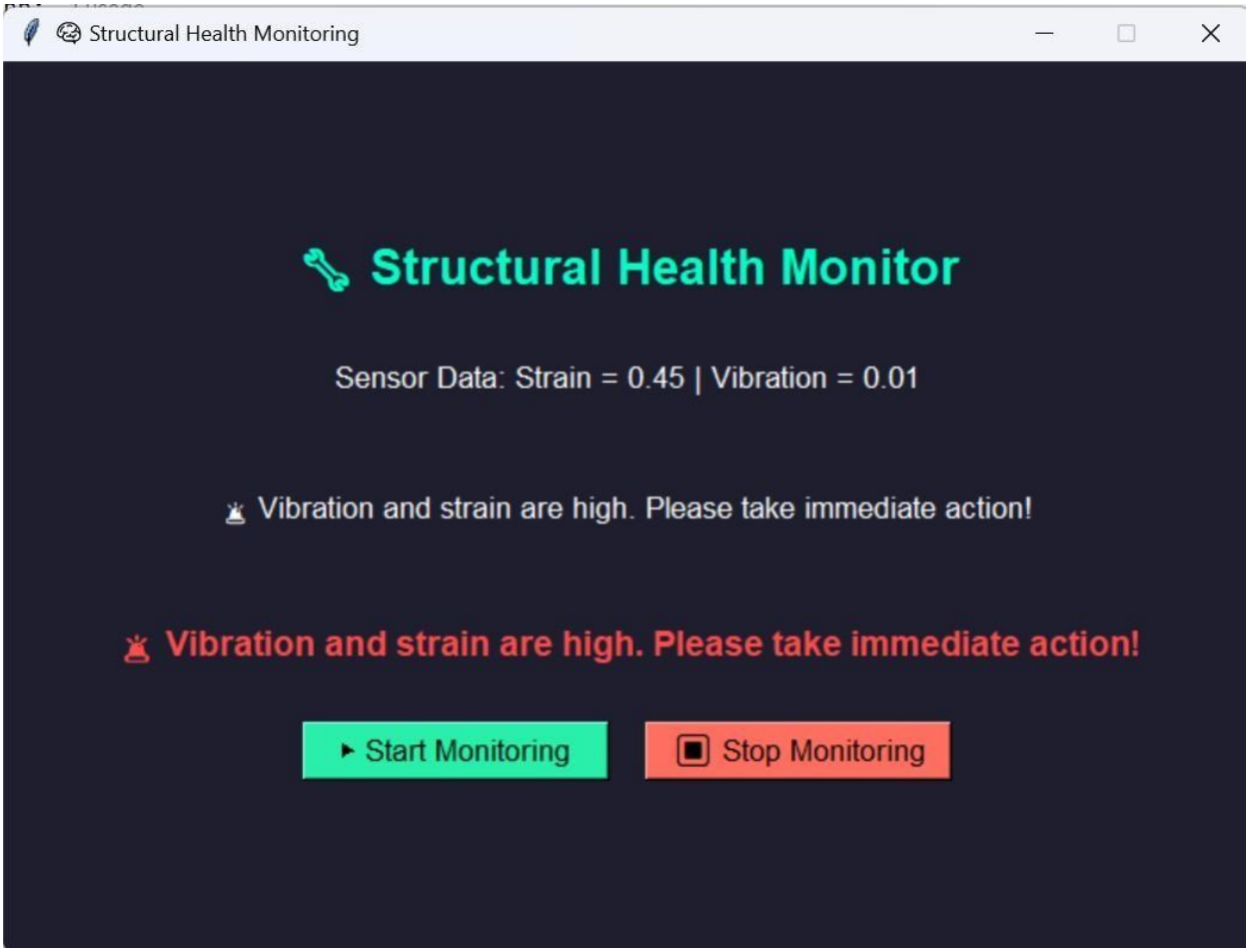
0s completed at 10:35 AM

```
            response = analyze_with_gemini(current_data)

            if "Critical stress" in response:
                message = "🚨 Vibration and strain are high. Please take immediate action!"
                self.status_label.config(text=message, fg="#ff4c4c")
                self.response_display.config(text=message)
                play_alert("Critical")
            elif "Moderate stress" in response:
                message = "⚠️ Moderate Stress Detected. Monitor closely."
                self.status_label.config(text=message, fg="#ffaa00")
                self.response_display.config(text=message)
                play_alert("Moderate")
            else:
                message = "✅ Vibration and strain levels are normal. All systems nominal."
                self.status_label.config(text=message, fg="#00ff7f")
                self.response_display.config(text=message)

            self.index += 1
            self.root.after(2500, self.update_ui)

        threading.Thread(target=run_analysis).start()

    def start_monitoring(self):
        if not self.running:
            self.running = True
            self.index = 0
            self.update_ui()

    def stop_monitoring(self):
        self.running = False
        self.status_label.config(text="⏸ Monitoring paused.", fg="#cccccc")
```

0s completed at 10:35 AM

```
# Run App
if __name__ == "__main__":
    root = tk.Tk()
    app = SensorApp(root)
    root.mainloop()
```

Output:



GITHUB LINK

https://github.com/syed2006-hub/NMFINALPROJECT.git