

Project Dossier: EchoGuard

1. Project Overview

- **Project Name:** EchoGuard
 - **Mission Statement:** To build a smart room monitoring system that actively listens to ambient audio through a microphone, classifies specific emergency sounds in real-time, and displays alerts on a user-friendly web dashboard.
 - **Core Problem Solved:** Provides an automated, always-on monitoring solution for critical audio events (like a smoke alarm or breaking glass) that might otherwise go unnoticed, enhancing safety and security.
 - **Key Differentiator:** An end-to-end system that leverages the unique and challenging domain of Audio AI, directly fusing ECE concepts (signal processing) with advanced Machine Learning, data management, and an interactive UI.
-

2. System Architecture & Technology Stack

This section outlines the components and how they communicate.

- **Technology Stack:**
 - **Language:** Python 3.9+
 - **AI / Audio Processing:** Librosa, Scikit-learn (for initial model), TensorFlow/Keras (for advanced model), numpy
 - **Real-time Audio Capture:** PyAudio
 - **Backend & Data:** Pandas, Flask
 - **Frontend:** Streamlit
- **Architectural Diagram:**
 1. **User Interface (Frontend - Streamlit):** The user starts the system and views alerts.
 2. **API Layer (Backend - Flask):** The Frontend requests alert data from this layer.
 3. **Core Logic (AI Engine - Python/Librosa/PyAudio):** The Frontend triggers this engine to start listening.
 4. **Data Management (Backend - Pandas):** The AI Engine sends alert data to this layer for storage.

-
5. **Database (File System):** The actual alert data is stored in a CSV file.

3. Team Roles & Responsibilities

- **You (Lead Architect & Presenter):**
 - **Responsibilities:** Overall system design, data collection/processing, development of the core AI audio classification model, integration of all components, and final project presentation.
 - **Key Skills to Master:** Librosa (Audio Loading, Mel-Spectrograms), Scikit-learn/TensorFlow (Model Training), PyAudio (Real-time Audio), System Integration.
- **"The Builder" (Backend & API Engineer):**
 - **Responsibilities:** Development of the data storage system for alerts, creation of the data management logic, and building the Flask API to serve alert data to the frontend.
 - **Key Skills to Master:** Pandas (DataFrame manipulation, CSV I/O), Flask (creating routes and returning JSON data).
- **"The UI/UX Specialist" (Frontend & Product Designer):**
 - **Responsibilities:** Design and development of the Streamlit web dashboard, ensuring a clean and intuitive user experience, creating presentation materials, and documenting the project.
 - **Key Skills to Master:** Streamlit (widgets, layout, data display), UI/UX design principles, Technical Communication (for README and slides).

4. Detailed Project Execution Plan

This is the step-by-step guide from start to finish.

1. **Environment Setup:** Create a requirements.txt file listing all necessary libraries (librosa, scikit-learn, tensorflow, pyaudio, pandas, streamlit, flask).
2. **Data Curation (Task for: You):**
 - **Goal:** To assemble a clean, labeled dataset of audio clips.
 - **Source:** Urbansound8K dataset, freesound.org, or other free sound effect libraries.
 - **Action:** Create a root folder named dataset/. Inside it, create subfolders for each sound class: alarm/, glass/, and background/. Place at least 50-100 short audio clips (.wav format is best) into each

respective folder. This structured folder layout is crucial for the next step.

3. Audio Pre-processing Script (preprocessor.py) (Task for: You):

- **Functionality:**

- The script will iterate through each subfolder in dataset/.
- For each audio file, it will use librosa.load() to load the audio waveform.
- It will then use librosa.feature.melspectrogram() to convert the waveform into a Mel-Spectrogram (the "image" of the sound).
- It will save these spectrograms as numerical numpy arrays (.npy files) or as images (.png files) into a new directory, processed_data/, while maintaining the subfolder structure.

1. AI Model Training Script (trainer.py) (Task for: You):

- **Functionality:**

- The script loads all the processed spectrogram data from processed_data/.
- It uses the folder names (alarm, glass, background) as the labels.
- It splits the data into a training set (80%) and a testing set (20%).
- **Model Architecture:** You will build a **Convolutional Neural Network (CNN)** using TensorFlow/Keras. A simple CNN with a few convolutional layers, pooling layers, and a final dense layer is a perfect architecture for classifying these spectrogram "images."
- The script compiles and trains the model.
- It evaluates the model's accuracy on the test set.
- It saves the final, trained model into a single file named **audio_model.h5**.

2. Data Management Logic (data_manager.py) (Task for: The Builder):

- **Functionality:**

- Create the file alert_log.csv.
Columns: Timestamp, AlertType (e.g., "Smoke Alarm Detected"), ConfidenceScore.
- Create a function log_alert(alert_type, confidence) that takes the alert details, gets the current timestamp, and appends a new row to alert_log.csv.
- Create a function get_alerts_df() that reads alert_log.csv and returns its content as a Pandas DataFrame.

1. AI Core Engine (audio_engine.py) (Task for: You):

- **Functionality:**

- Loads the trained audio_model.h5.
- Uses the **PyAudio** library to open a stream from the microphone.
- Enters a while loop to continuously listen to the audio stream in short chunks (e.g., 2-3 seconds).
- Inside the loop:
 - For each chunk of audio data, use Librosa to create a Mel-Spectrogram on the fly.
 - Pre-process this spectrogram to match the format the model was trained on.
 - Use model.predict() to classify the spectrogram.
 - The model will output probabilities for each class.
 - **Decision Logic:** If the probability for "alarm" or "glass" is above a set threshold (e.g., 90%), it's a positive detection.
 - Calls the log_alert() function from data_manager.py, passing the detected class and the confidence score.

2. Streamlit Dashboard (dashboard.py) (Task for: The UI/UX Specialist):

- **Functionality:**

- Sets up the title and layout (st.title("EchoGuard - Smart Audio Monitor")).
- Creates a "Start/Stop Monitoring" button to trigger your audio_engine.py.
- Creates a main status display: "Status: Monitoring" or "Status: Inactive."

- Creates an "Alert Log" section that calls the Flask API to fetch and display the contents of alert_log.csv in a clean, auto-updating table. It should highlight critical alerts in red.

1. Flask API (api.py) (Task for: The Builder):

- **Functionality:**
 - A simple Flask server.
 - One endpoint: @app.route('/get-alerts').
 - The function for this route will call get_alerts_df() from data_manager.py and return the DataFrame as a JSON response.

2. Integration (Lead: You):

- You will guide the UI/UX specialist on how to use Python's subprocess or threading to run your audio_engine.py in the background when his "Start" button is pressed.
 - You will ensure the communication between the Streamlit app and the Flask API is working flawlessly.
-

5. Final Deliverables & Presentation

- **GitHub Repository:**
 - A clean project structure.
 - A professional README.md file (written by the UI/UX Specialist) explaining the project's unique Audio AI approach.
 - The requirements.txt file.
- **Presentation (Delivered by: You):**
 - Slides created by the UI/UX Specialist.
 - **Structure:**
 - The Problem:** The limitation of traditional safety systems (they can't "hear").
 - Our Solution:** Introducing EchoGuard, a smart listening system.
 - The Core Technology:** Explain the concept of a Spectrogram—"How we turn sound into an image."

4. **Live Demo:** The most crucial part. Start the system, play a smoke alarm sound effect on your phone, and show the alert pop up on the dashboard in real-time.
 5. **Challenges & Future Work:** Discuss the challenge of real-world noise and how the system could be expanded.
- **Video Demo:** A short, well-edited screen recording of the final product in action, created by the UI/UX Specialist.