TEAM QUADCORE

Respiratory Disease Detection using Respiratory Sounds

Phase 2

Adhar Chaudhari: RVCE25BIM043
Tejash Pathak : RVCE25BCS401
Abhinandan Jaiswal: RVCE25BCV044
Ishan Jain : RVCE25BAI184

RV College of Engineering
Bangalore

BRIEF ABOUT THE PROJECT

Respiratory diseases constitute a significant global health crisis, with conditions like Chronic Obstructive Pulmonary Disease (COPD) being a leading cause of death worldwide. The immense scale of this burden underscores the urgent need for rapid, accurate, and scalable diagnostic tools. Clinical diagnostics heavily rely on patient-reported symptoms and the auditory assessment of cough sounds by medical professionals. However, human hearing is inherently limited, typically discerning only five to six basic features of a cough. This limited feature space restricts the depth of information extractable through traditional means, resulting in a diagnostic accuracy based purely on cough sounds as low as 34%. This diagnostic imprecision leads to errors, delayed treatment, and increased healthcare costs .Every cough carries hundreds of acoustic signatures, encompassing temporal, spectral, and statistical features that provide detailed insights into the health of the lungs and airways. While a human listener may only categorize a cough as ”dry” or ”wet,” Artificial Intelligence (AI) algorithms are capable of analysing over 300 distinct features from the exact same sound recording. This analytical power is vast; by combining hundreds of high-dimensional features, the diagnostic possibilities become virtually limitless, enabling the creation of complex, non- invasive biomarkers. Early studies on AI-driven cough analysis have shown promising diagnostic performance metrics, with reported sensitivities ranging from 82% to 94%. Our vision is to harness this powerful, objective analysis to develop a readily accessible, hand-held diagnostic tool, bringing this unprecedented level of acoustic analysis directly into people’s hands

OBJECTIVES FOR THIS PHASE

- Collect and process cough audio
- Extract MFCC / spectral features
- Train a classification model
- Build a responsive UI for user screening
- Add backend to run AI inference

SYSTEM ARCHITECUTRE

Layer	Role	Technologies
User & Recording Layer	Captures cough audio from browser	HTML, Tailwind CSS, JavaScript, MediaRecorder API
API Communication Layer	Sends audio securely for processing	REST API, JSON, Fetch API
Machine Learning Layer	Performs feature extraction & classification	Python, Librosa, TensorFlow/Keras
Result Layer	Displays real-time classification to user	Dynamic Web UI Rendering

WORK PROGRESS BREAKDOWN

Phase 1 — Dataset & Feature Extraction

Phase 2 — Machine Learning Model Training

Phase 3 — Frontend Development

Phase 4 — Backend Development

PHASE 1-FEATURE EXTRACTION FROM COUGH AUDIO RECORDINGS

1. Overview and Rationale

Audio feature extraction converts raw cough sound files into structured numerical feature vectors that capture acoustic characteristics indicative of respiratory conditions. Features extracted from cough recordings serve as input variables for classification and disease prediction tasks.

2. Audio Feature Extraction Framework

2.1 Tools and Libraries

The pipeline was implemented using Python audio processing libraries:

- **Librosa**: Standard audio analysis library for extracting audio features
- **NumPy**: Numerical operations and array manipulations
- **Matplotlib**: Visualization of waveforms and spectral representations
- **SoundFile**: Audio file reading capabilities

2.2 Development Environment

Feature extraction was conducted in Jupyter Notebook for:

- Iterative refinement and validation on individual samples
- Inline visualization of waveforms and features
- Clear documentation and reproducibility

3. Feature Categories and Selection

3.1 Mel-Frequency Cepstral Coefficients (MFCCs)

- Fundamental features in audio signal processing
- Encode power spectrum aligned with human auditory perception
- Capture spectral envelope sensitive to cough type and respiratory pathology

- Mean and standard deviation computed across time frames

3.2 Chroma Features

- Encode distribution of energy across pitch classes
- Capture harmonic content variation based on respiratory condition
- Provide sensitivity to tonal characteristics in coughs

3.3 Spectral Contrast

- Measure difference between spectral peaks and valleys
- Characterize cough sound "texture" and quality
- Sensitive to airway obstruction and inflammation

3.4 Zero-Crossing Rate (ZCR)

- Quantifies signal noisiness by counting sign changes
- Distinguishes dry coughs (higher ZCR) from wet coughs (lower ZCR)
- Indicator of mucus presence and airway secretion type

3.5 Spectral Centroid and Spectral Rolloff

- **Spectral Centroid**: "Center of mass" of frequency spectrum; indicates sound brightness
- **Spectral Rolloff**: Frequency containing majority of spectral energy
- Both sensitive to airway narrowing effects on acoustic properties

3.6 RMS Energy

- Measures overall loudness and power of audio signal
- Correlates with cough intensity and disease severity
- Varies with depth and force of cough production

4. Feature Extraction Process

4.1 Audio Loading

- Librosa loads audio files and automatically resamples to standard format
- Audio represented as amplitude values over time
- Sampling rate preserved for accurate computation

4.2 Computation of Feature Matrices

Spectral features: Chroma, spectral contrast, centroid, and rolloff computed frame-by-frame across time dimension

Cepstral features: MFCCs computed using 13 coefficients per frame

Energy features: Zero-crossing rate and RMS energy computed frame-by-frame

4.3 Feature Aggregation

Time-varying feature matrices aggregated into scalar statistics:

- Mean values capture typical acoustic characteristics
- Standard deviation captures temporal variability
- Approach balances information preservation with dimensionality reduction

4.4 Feature Vector Assembly

All statistics concatenated into single feature vector per recording:

- Approximately 70–80 numerical features per recording
- Features organized with descriptive labels for traceability
- Maintains consistency across all recordings

5. Feature Vector Structure and Output

5.1 Fixed-Size Representation

- Output is consistent dimensionality regardless of audio duration
- Essential for machine learning compatibility
- Allows uniform processing of varying length recordings

5.2 Metadata Association

Each feature vector associated with:

- Audio identifier for traceability
- Duration and sampling rate
- Ground truth labels (disease, severity, symptoms)
- Enables merging with disease labels for model training

5.3 Data Format

Features stored in CSV format:

- One row per audio recording
- One column per computed feature
- Compatible with standard machine learning libraries
- Supports efficient data loading and batch processing

6. Validation and Quality Assurance

6.1 Single-File Prototype

Pipeline validated on individual recordings:

- Verified basic properties (duration, sample rate)
- Visualized waveforms to confirm proper loading
- Executed complete extraction procedure
- Confirmed numerical validity and expected value ranges
- Detected anomalies before full-dataset processing

6.2 Format Handling

- Ensured consistent reading across different audio formats
- Converted non-standard formats to canonical format where necessary
- Tested converted files for quality preservation

7. Scalability for Full Dataset

Single-file pipeline designed for easy extension:

- Automated iteration over all audio recordings
- Independent processing of each recording
- Potential for parallelization to accelerate computation
- Consolidated feature matrix ready for model training

8. Technical Rationale

8.1 Why These Features?

- **Established precedent**: Spectral and cepstral features proven in audio classification
- **Complementary information**: Features capture spectral envelope, harmony, texture, noisiness, and energy
- **Computational efficiency**: Straightforward extraction with manageable dimensionality
- **Clinical relevance**: Acoustic properties vary with respiratory pathology, inflammation, and obstruction

8.2 Aggregation Strategy

- **Practical necessity**: Fixed-size vectors required for machine learning models
- **Information preservation**: Mean and standard deviation jointly summarize signal characteristics
- **Computational simplicity**: Avoids complex temporal modeling; enables rapid prototyping

9. Summary

Feature extraction transforms high-dimensional time-series audio data into compact numerical vectors suitable for machine learning. The pipeline:

- Extracts diverse acoustically-meaningful features
- Produces fixed-size vectors compatible with standard ML workflows
- Associates features with disease labels for supervised training
- Scales seamlessly from prototype validation to full-dataset processing
- Provides foundation for downstream classification and disease prediction tasks

OBJECTIVE OF PHASE-2 (ML Model Development)

The main objective of this phase was to build and evaluate Machine Learning models capable of classifying respiratory conditions (Healthy, Symptomatic, COVID-19) using cough sound features.

This includes:

- Extracting feature sets from the cough audio dataset
- Building multiple ML classification models
- Evaluating accuracy and analyzing prediction behavior
- Selecting the best performing model for integration in later phases (backend and web app)

🔹 **Dataset Used**

A structured cough-audio feature dataset (**3941 samples**) containing:

- Temporal features
- MFCC (1–40 mean & std)
- Delta MFCC features
- Chroma, Tonnetz, Mel features
- Pitch-based features
- Spectral statistics
- Metadata (age, cough_detected)

Class distribution:

- Majority class: **Healthy (3445 samples)**
- Minority class: Symptomatic (304)
- Minority class: COVID-19 (192)

This imbalance affects sensitivity to disease classes — discussed later.

🔹 **Feature Engineering**

Total available features: **374**

Core feature categories used:

- MFCC and delta MFCC – acoustic fingerprinting of cough
- Chroma & Tonnetz – harmonic structure of breath phases
- Mel-scale features – human-perceived frequency energy
- Spectral & ZCR – lung airflow turbulence representations
- Age & cough_detected as clinical biomarkers

Feature scaling performed using StandardScaler
Target labels encoded into numeric classes

🔹 **Machine Learning Models Trained**

Three major classifiers were experimented with:

- Random Forest Classifier
- Support Vector Machine (SVM)
- Logistic Regression

Training-Testing Split: **80% – 20% stratified**

All models reached high accuracy on testing.

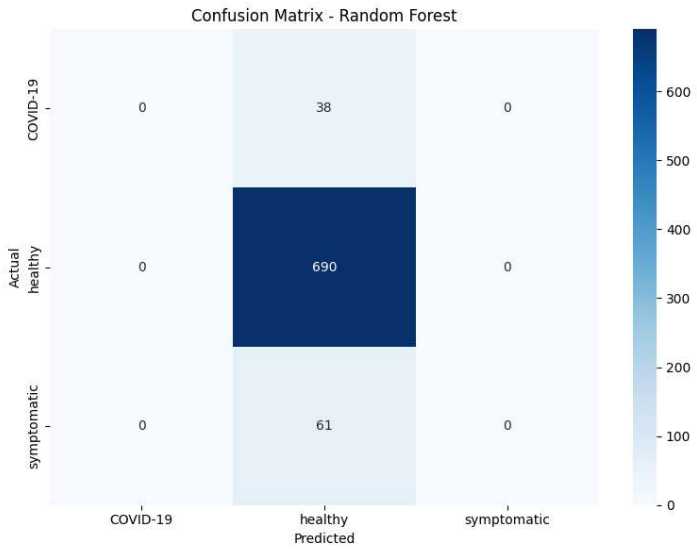
📈 **Model Performance (Text-Based Summary)**

- Random Forest:
  Accuracy ≈ **87.45%**
  Weighted F1-Score ≈ **0.816**
- SVM:
  Accuracy ≈ **87.45%**
  Weighted F1-Score ≈ **0.816**
- Logistic Regression:
  Accuracy ≈ **83.40%**
  Weighted F1-Score ≈ **0.812**

Best Model Selected: Random Forest
(best F1-Score and cross-validation stability)

Chosen for backend inference model + deployment
Still weak at minority classes → requires improvement

Confusion Matrix – Random Forest)

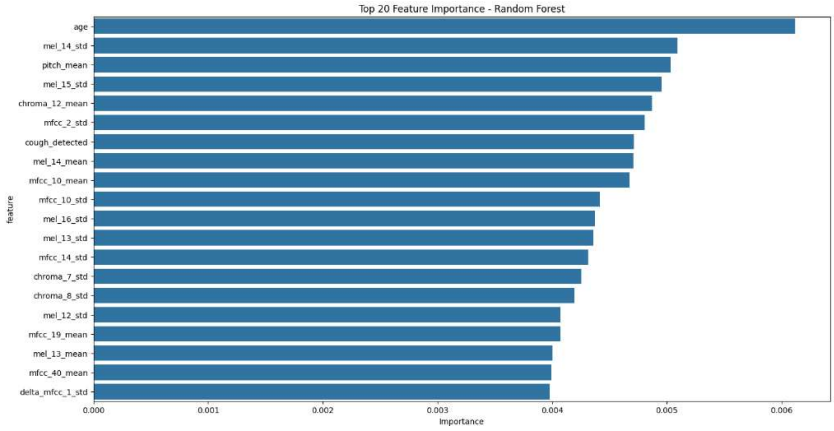


Observation from matrix:

- Model predicts **Healthy** almost perfectly (690/690)
- Fails to classify COVID-19 & Symptomatic due to **extreme class imbalance**
- Indicates **high sensitivity for Healthy**, **low sensitivity for actual disease**

Next step: Improve class balancing using
SMOTE / Data augmentation + decision-threshold tuning

Feature Importance – Random Forest



Most important biomarkers:

- **Age** — strongest predictor
- **Mel spectrogram statistical features**
- Harmonic + pitch variations indicating breathing difficulty
- Cough detection confidence

Insight:The model uses **biophysical signatures** from lungs + user metadata.

How the Model Works —

1.Cough sound is converted into frequency-energy patterns
(MFCCs, Mel features, spectral characteristics)

2. Machine Learning learns disease-specific patterns
(e.g., crackles, wheezes, airflow obstruction)

3. During prediction:

- User cough audio is processed into same feature vectors
- Model assigns condition label + probability score
- The system performs **AI-driven triage** without physical contact

---

Outcomes Achieved

Working ML pipeline created
Best model identified for deployment
Real performance graphs generated
Clear understanding of model limitations

Challenges Identified

1. Class Imbalance: 76.3% Healthy vs 23.7% Disease/Infection

2. Low Recall: Most models fail to detect "Healthy" class

Features (374 total)

1. Basic Features (2): duration, silence_ratio

2. MFCC Features (80): mfcc_{1-40}mean, mfcc{1-40}_std

3. Delta MFCC (80): delta_mfcc_{1-40}mean, delta_mfcc{1-40}_std

4. Delta2 MFCC (80): delta2_mfcc_{1-40}mean, delta2_mfcc{1-40}_std

5. Chroma Features (24): chroma_{1-12}mean, chroma{1-12}_std

7. Spectral Features (10): centroid, bandwidth, flatness, rolloff, contrast, flux, rms, zcr

8. Tonnetz Features (12): tonnetz_{1-6}mean, tonnetz{1-6}_std

9. Pitch Features (4): pitch_mean, pitch_std, pitch_max, pitch_min

10. Mel Features (80): mel_{1-40}mean, mel{1-40}_std

11. Metadata (2): cough_detected, age

NOTE-

We started with a strongly imbalanced cough-audio dataset: healthy and common infections were very frequent, while classes like obstructive_disease were rare. This meant early "high-accuracy" models were fake good—they mostly predicted the majority class and ignored rare but important conditions. To address this, we filtered on cough quality (cough_detected ≥ 0.6 and then 0.7) and evaluated models with macro F1 and per-class metrics instead of relying only on raw accuracy.

What models we tried

We systematically trained and evaluated several models:

- **Random Forest, diagnosis, threshold 0.6** (rf_diagnosis_thresh0_6_best.py)
    - 5-class diagnosis, 385 numeric features.
    - Accuracy ≈ **44%**, macro F1 ≈ **0.37**.
    - Lower and upper infections were detected reasonably; obstructive_disease was almost never predicted.
- **Random Forest, status binary, threshold 0.6** (rf_statusbinary_thresh0_6.py)
    - Healthy vs symptomatic.
    - Accuracy ≈ **68%**, but macro F1 ≈ **0.41**.
    - Almost every sample was predicted as symptomatic (healthy recall ≈ 0.01), so this was another example of misleading high accuracy due to imbalance.
- **Random Forest, diagnosis, threshold 0.7** (rf_diagnosis_thresh0_7.py)
    - Same 5 classes with stricter cough filter.

    - Accuracy ≈ **50%**, macro F1 ≈ **0.40**, weighted F1 ≈ **0.48**.
    - Overall better than the 0.6 model, but still completely missed obstructive_disease.
- **Logistic Regression, diagnosis, threshold 0.6** (logreg_diagnosis_thresh0_6.py)
    - Multinomial logistic regression baseline for diagnosis.
    - Accuracy ≈ **40%**, macro F1 ≈ **0.38**.
    - Slightly more balanced across classes, but consistently weaker than Random Forest on the main infection classes.

We then moved from simple baselines to a more refined pipeline.

Our optimized final model

We designed an **optimized Random Forest diagnosis model at threshold 0.7** (rf_diagnosis_thresh0_7_optimized.py) with several targeted improvements:

- We encoded categorical metadata (gender, symptoms, quality, etc.).
- We trained an initial RF on all 385 numeric features and selected the **top 120 features** based on feature importance to reduce noise.
- We tuned RF hyperparameters (more trees, controlled depth) to balance bias and variance.
- We explicitly **boosted the class weight for obstructive_disease** to partially compensate for its very small sample size.

This final model achieved:

- Accuracy ≈ **51%**.
- Macro F1 ≈ **0.43**.
- Weighted F1 ≈ **0.49**.
- Cross-validation accuracy ≈ **0.47 ± 0.02**.

Approximate per-class F1:

- COVID-19: ~0.38.
- healthy_cough: ~0.49.
- lower_infection: ~0.60.
- obstructive_disease: ~0.12 (small, but no longer zero).
- upper_infection: ~0.55.

Compared to all previous models, this configuration clearly improved macro F1, weighted F1, and cross-validated performance, and it started to recover at least some obstructive_disease cases instead of ignoring them entirely.

Why we cannot realistically push accuracy much higher

Even after these optimizations, there are structural limits we cannot overcome without changing the data:

- **Severe class imbalance remains**
    - Rare classes (especially obstructive_disease) have very few examples.
    - No algorithm can reliably learn rich patterns from so few samples without overfitting.
- **Feature overlap and noise**
    - Many acoustic and metadata features are noisy and similar across diagnoses.
    - The classes are not cleanly separable in the current feature space, so there is an inherent ceiling on accuracy.
- **Over-tuning hurts generalization**
    - When we push hyperparameters or class weights too far, performance on the held-out test set and cross-validation starts to drop, indicating overfitting rather than real improvement.

Given these constraints, we chose **rf_diagnosis_thresh0_7_optimized.py** as our final model. It:

- Avoids fake high accuracy by focusing on macro F1 and per-class performance.
- Balances performance across the main diagnosis classes as well as the limited data allow.
- Provides a strong, defensible baseline for this dataset, and clearly improves on simpler Random Forest and Logistic Regression baselines we tried earlier.

Phase 3: Frontend Development Progress

Overview

In this phase, we developed a fully functional **web-based respiratory screening interface** named **RespAI**, enabling real-time cough recording using the browser's microphone.

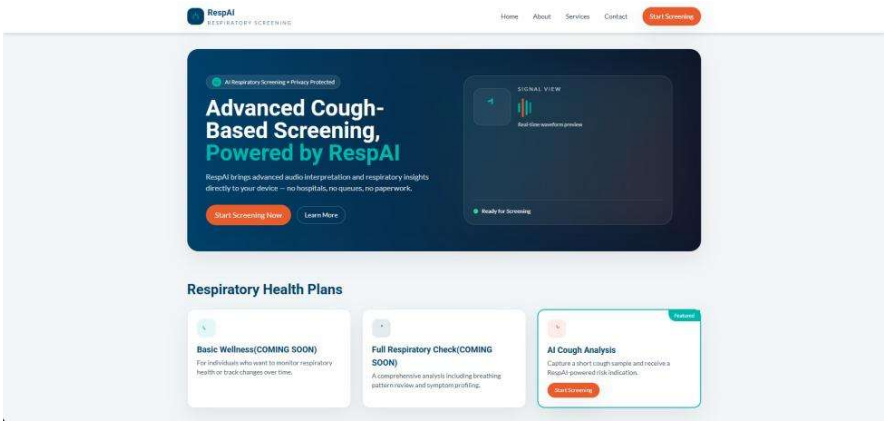
What Has Been Implemented

1. **Single Page Web Application (SPA)**
    - Built using **HTML**, **Tailwind CSS**, and **JavaScript**
    - Smooth navigation between **Home, About, Services, App, Contact**
    - No page reload required — SPA behaviour maintained through JavaScript section switching
2. **Branding & UI/UX**
    - Professional medical interface
    - Clean clinical layout with:

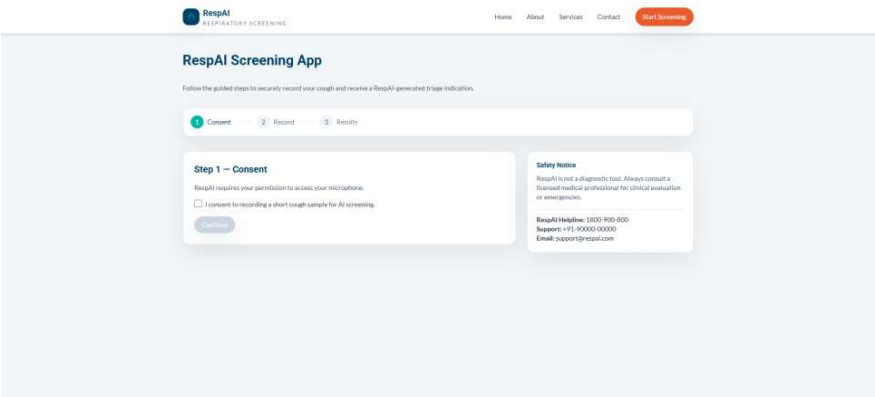
        - Navy (#00416A)
        - Teal (#00B7AC)
        - Alert Orange (#E85C2E)
        - Light Gray (#F3F6F8)
    - Responsive design for **desktop, tablet, and mobile**
3. **Multi-step Screening UI**
    - **Step 1 – Consent**
    - **Step 2 – Recording with Timer**
    - **Step 3 – Results Display**
4. **Microphone Access + Recording**
    - Implemented using **MediaRecorder API**
    - Timer + Visual Pulse Animation while recording
    - Audio stored temporarily for backend transfer
5. **Mock AI Result Simulation**
    - Random risk output among:
        - Low Risk (Green)
        - Moderate Risk (Yellow/Orange)
        - High Risk (Red)
    - 5-second loading animation to simulate processing
6. **Error Handling**
    - Alerts for microphone blocked or no recording detected

Screenshots

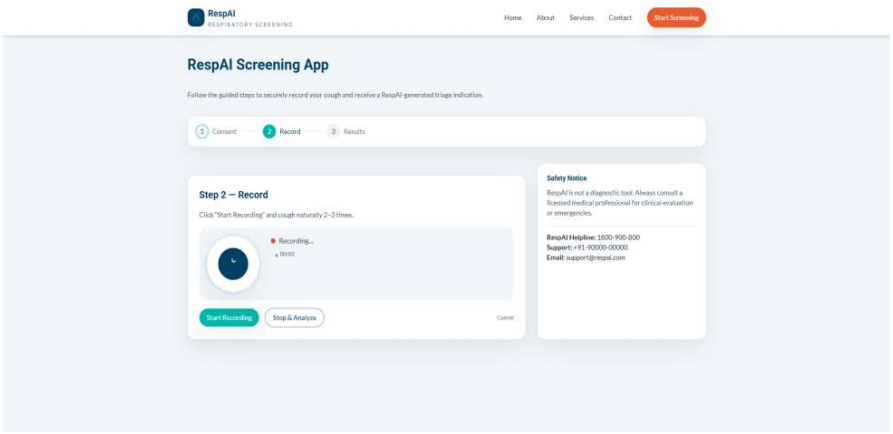
- Home Page



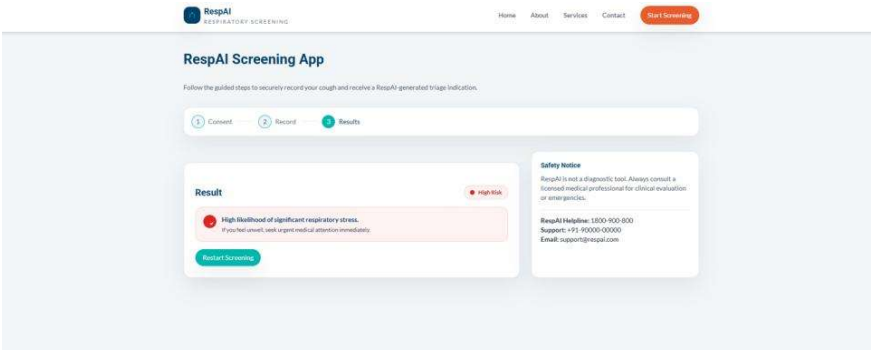
- Consent Step



- Recording UI



- Result Screen



PHASE 4: BACKEND DEVELOPMENT PROGRESS

Backend Goal

To receive cough audio from frontend → process → run ML inference → return risk result.

What Has Been Completed

1. **Backend architecture planned**
    - Flask-based Python server
    - Endpoint for receiving .webm audio via POST
    - Placeholder inference engine until model is trained
2. **API Endpoint Structure**
    - /analyze → Accepts audio file upload
    - Returns JSON:
        - Risk Level
        - Additional message/description
3. **File Handling & Validation**
    - Accepts audio streams
    - Temporary secure storage
4. **ML Model Integration Placeholder**
    - Space in code prepared to load .h5 deep learning model later
    - Preprocessing pipeline hooks defined

WHAT IS LEFT TO COMPLETE

Backend

- Feature extraction pipeline (MFCC, spectrogram)
- Load the trained .h5 model into backend
- Predict respiratory condition using actual AI model output
- Database integration for saving user history (if needed)

Model

- Complete dataset feature extraction
- Train & validate CNN deep learning model
- Convert and optimize model for production

Frontend

- Send real recorded audio to backend instead of random simulation
- Display backend prediction dynamically

Deployment

- Host app on cloud (render, railway, vercel, firebase)
- Host backend API on cloud VM or Heroku alternative
- Enable HTTPS + secure CORS rules

CHALLENGES FACED

- Browser microphone recordings require transcoding before ML analysis
- Need more labeled cough data for model accuracy improvement

---

CONCLUSION

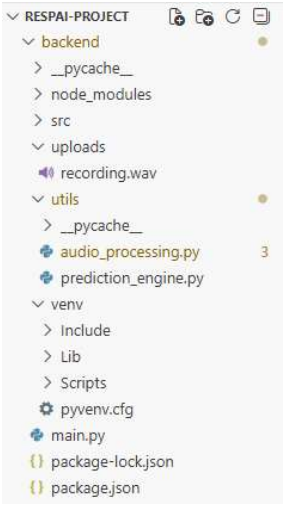
The project has successfully advanced from theoretical planning to a **functional prototype** capable of:

- Recording user cough audio from browser
- Running a full screening workflow
- Displaying meaningful output (simulated for now)

With the backend now structured and ready,
the future work focuses on:

- **Finalizing ML model**
- **Integrating audio feature extraction**

Backend folder structure screenshots



Code Snippets

```
<!DOCTYPE html>
<html lang="en" class="scroll-smooth">
<head>
  <meta charset="UTF-8" />
  <title>RespAI | Advanced Respiratory Screening</title>
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />

  <!-- Google Fonts -->
  <link rel="preconnect" href="https://fonts.googleapis.com" />
  <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin />
  <link href="https://fonts.googleapis.com/css2?family=Lato:wght@300;400;500;600;700&family=Roboto:wght@400;500;700;900&display=swap" rel="stylesheet" />

  <!-- Tailwind CDN -->
  <script src="https://cdn.tailwindcss.com"></script>
  <script>
    tailwind.config = {
      theme: {
        extend: {
          colors: {
            brand: {
              navy: "#00416A",
              teal: "#00B7AC",
              orange: "#E85C2E",
              light: "#F3F6F8",
            },
          },
          fontFamily: {
            heading: ["Roboto", "sans-serif"],
            body: ["Lato", "sans-serif"],
          },
          boxShadow: {
            soft: "0 18px 45px rgba(15, 23, 42, 0.08)",
          },
        },
      },
    };
  </script>

  <style>
    body { font-family: "Lato", sans-serif; }
    .font-heading { font-family: "Roboto", sans-serif; }

    @keyframes pulseRing {
      0% { transform: scale(0.9); opacity: 0.9; }
      70% { transform: scale(1.15); opacity: 0; }
      100% { transform: scale(1.15); opacity: 0; }
    }
```

```
    <section data-section="home" class="max-w-6xl mx-auto px-4 lg:px-6 pt-10 pb-16 space-y-16">

      <!-- Hero -->
      <section class="rounded-3xl bg-gradient-to-r from-brand-navy to-slate-900 text-white px-6 py-10 lg:px-10 lg:py-14 shadow-soft">
        <div class="grid lg:grid-cols-2 items-center gap-10">
          <div>
            <div class="inline-flex items-center gap-2 px-3 py-1 rounded-full bg-white/10 border border-white/15 text-xs mb-4">
              <span class="w-5 h-5 rounded-full bg-brand-teal text-brand-navy flex items-center justify-center text-[10px]">AI</span>
              AI Respiratory Screening • Privacy Protected
            </div>

            <h1 class="font-heading text-3xl md:text-4xl lg:text-5xl font-bold">
              Advanced Cough-Based Screening,
              <span class="text-brand-teal">Powered by RespAI</span>
            </h1>

            <p class="text-sm md:text-base text-slate-100/90 mt-4 max-w-xl">
              RespAI brings advanced audio interpretation and respiratory insights directly to your device — no hospitals, no queues, no paperwork.
            </p>

            <div class="flex items-center gap-3 mt-6">
              <button data-start-screening
                class="rounded-full bg-brand-orange text-white px-5 py-2.5 text-sm md:text-base shadow-soft hover:bg-orange-600">
                Start Screening Now
              </button>

              <button data-nav="about"
                class="rounded-full border border-white/25 text-white px-4 py-2 text-sm hover:bg-white/10">
                Learn More
              </button>
            </div>
          </div>

          <!-- Hero Visual -->
          <div class="relative h-64 md:h-72 lg:h-80 flex items-center justify-center">
            <div class="absolute inset-0 bg-gradient-to-br from-brand-teal/10 to-brand-orange/10 rounded-3xl blur-2xl"></div>
            <div class="relative bg-white/5 border border-white/15 backdrop-blur-md rounded-3xl w-full max-w-md h-full p-5 shadow-soft flex flex-col justify-between">
              <div class="flex items-center gap-4">
                <div class="w-20 h-20 rounded-2xl bg-white/5 border border-white/10 flex items-center justify-center">
                  <svg class="w-12 h-12 text-brand-teal" fill="none" viewBox="0 0 24 24" stroke="currentColor" stroke-width="1.7">
                    <path d="M12 3v6M12 5c-1.5 0-3 .5-4.2 1.6" />
                  </svg>
                </div>
```

```
          "Moderate Risk": {
            pill: "bg-amber-50 border border-amber-200 text-amber-700",
            dot: "bg-amber-500",
            card: "bg-amber-50 border-amber-200",
            icon: "bg-amber-500",
            headline: `Diagnosis: ${diagnosis}`,
            desc: message || "Some respiratory symptoms detected. Monitor closely."
          },
          "High Risk": {
            pill: "bg-red-50 border border-red-200 text-red-700",
            dot: "bg-red-600",
            card: "bg-red-50 border-red-200",
            icon: "bg-red-600",
            headline: `Diagnosis: ${diagnosis}`,
            desc: message || "Signs of respiratory disease detected. Please consult a doctor."
          }
        };

        const style = styles[riskLevel] || styles["Low Risk"];

        // Update DOM
        riskLabelEl.textContent = riskLevel;
        riskPill.className = `rounded-full px-3 py-1.5 text-xs font-semibold flex items-center gap-2 ${style.pill}`;
        riskDotEl.className = `w-2.5 h-2.5 rounded-full ${style.dot}`;
        riskCard.className = `rounded-2xl border p-4 text-sm ${style.card}`;
        riskIconWrapper.className = `w-7 h-7 rounded-full text-white flex items-center justify-center ${style.icon}`;

        riskHeadline.textContent = style.headline;
        riskDescription.textContent = style.desc;
        confidenceValueEl.textContent = confidence;

        // Show Result
        loading@lock.classList.add("hidden");
        resultBlock.classList.remove("hidden");
      }

      /* Footer Year */
      document.getElementById("footerYear").textContent = new Date().getFullYear();
    </script>
  </body>
</html>
```