Air India Voice Bot Platform

Portfolio Project Case Study

FreeSWITCH · mod_audio_fork · ESL · NLU · Air India Workflows · API Orchestration · TTS Playback

Prepared as a professional portfolio description of the solution I designed and implemented.

Air India Voice Bot Platform

FreeSWITCH · mod_audio_fork · ESL · NLU · Air India Workflow Automation

This document describes the voice automation platform I designed and implemented for airline support workflows, including telephony integration, audio streaming, NLU-driven intent operations, business API orchestration, and controlled response playback into a live FreeSWITCH call session.

Portfolio framing: this project demonstrates telephony integration, conversational backend design, session control, workflow orchestration, and practical problem-solving across real-time media systems.

Contents

Project positioning

This portfolio case study focuses on the architecture and implementation patterns I handled personally: FreeSWITCH call handling, media forking, ESL-based call control, bot-side NLU and dialogue operations, API-driven workflow execution, response generation, and the practical return path that injected generated audio back into the active call.

Project type	Real-time airline voice bot platform
Core result	Built a production-style architecture where caller audio was forked from FreeSWITCH to an external bot stack, interpreted through NLU and workflow logic, and returned to the live call by generating response audio files and injecting them through ESL-driven playback.

1. Project Summary

I built a voice automation solution around FreeSWITCH for Air India workflows. The system was designed to receive live caller audio, send that audio out for bot-side processing, understand what the user was asking for, execute the relevant business action, and then speak the result back into the same live call session.

The solution was not a simple IVR. It combined a telephony layer, a media-forking layer, an NLU layer, an API workflow layer, and a controlled playback-return path. This separation allowed the voice bot to behave like an operational support agent rather than a menu-driven audio tree.

A key engineering insight in the project was that outbound media and inbound playback did not have to be handled by the same mechanism. I used mod_audio_fork to move live call audio out of FreeSWITCH, and I used ESL plus a separate playback script to bring bot-generated audio back into the call. That pattern made the overall system practical and controllable.

2. My Role and Ownership

- Designed the overall architecture for the call-processing flow, including separation of telephony, streaming, bot operations, and playback control.
- Integrated FreeSWITCH with mod_audio_fork to send live audio to a remote processing stack.
- Used ESL as the control channel to interact with active FreeSWITCH sessions from an external service.
- Built or defined the bot-side NLU and intent-operation layer so caller speech could be converted into structured actions.
- Created Air India workflow handling through API calls based on the detected user intent and current dialogue state.
- Implemented the practical return path in which response text was converted into an audio file, detected by a separate service, and played back into the live call through ESL.
- Worked through multiple architectural approaches before reaching the final pattern that balanced feasibility, control, and conversational behavior.

3. Business Problem and Objectives

The objective was to build a voice bot that could go beyond fixed prompts and support real airline-style workflows. The caller should be able to speak naturally, have the system interpret the request, and receive a meaningful spoken response based on real backend operations.

To achieve that, the solution needed to solve several technical requirements at once: real-time call handling, extraction of live speech from the call, processing of the speech through bot logic, execution of business APIs, and re-injection of the response into the same call without losing session control.

The end goal was a system capable of acting on user intent instead of merely collecting DTMF input. That required a stateful architecture and a reliable orchestration path between telephony, NLU, business operations, and response playback.

4. Architecture Overview

At a high level, the architecture was divided into three major paths: outbound media, intelligence and workflow processing, and return-path playback control.

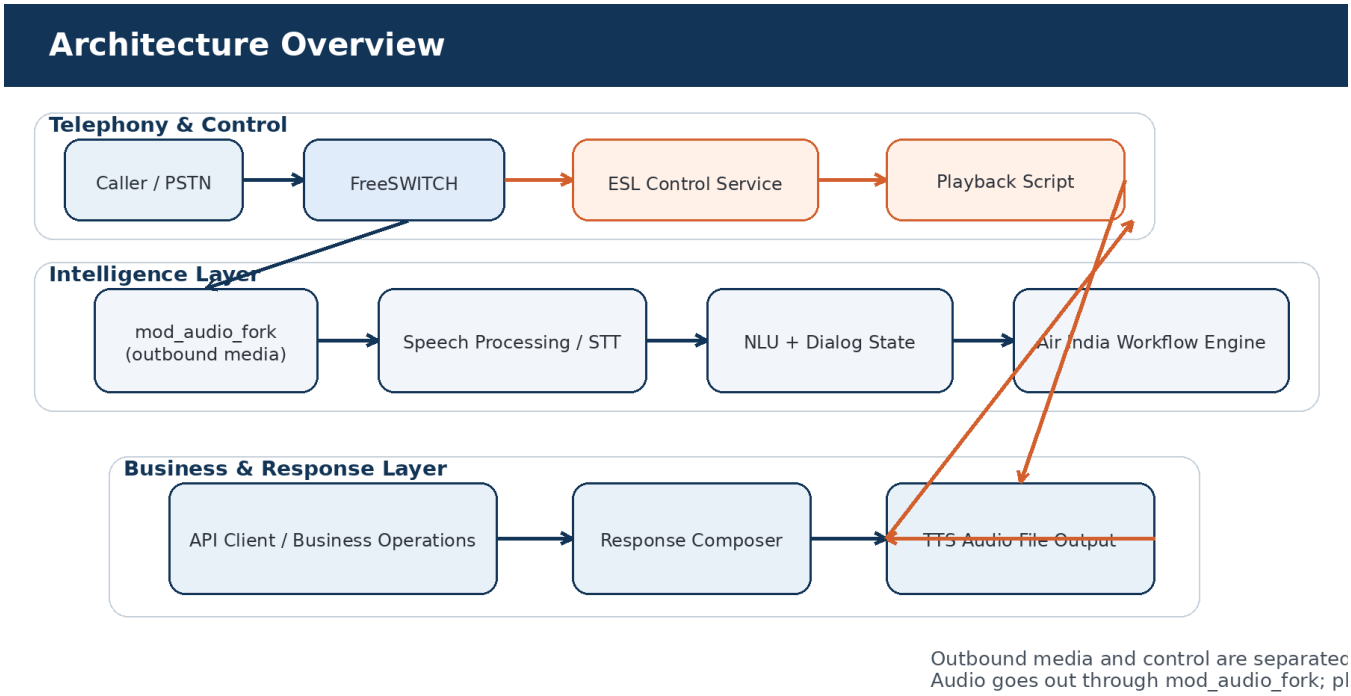

Figure 1. High-level architecture separating audio forking, bot intelligence, and ESL-driven playback return.

Architecture interpretation

- FreeSWITCH handled the live call leg and session lifecycle.
- mod_audio_fork was used only for sending caller audio out to the bot side; it was not treated as the mechanism for directly playing generated response audio back into the call.
- The bot side performed the intelligence work: speech understanding, NLU, state management, workflow selection, API operations, response composition, and TTS generation.
- ESL acted as the control bridge back into FreeSWITCH so an external service could instruct the active channel to play the generated response file.

5. Component Breakdown

Component	Role in the project
FreeSWITCH	Handled incoming call sessions, media state, and playback target behavior. It was the telephony anchor for the whole system.

mod_audio_fork	Forked live call audio to another server for processing. This provided the outbound media path from the telephony environment to the bot-processing layer.
Bot server	Received the audio stream and converted it into processable bot input. This server hosted the intelligence part of the application.
NLU and dialog layer	Mapped natural caller speech into a structured intent, extracted entities, tracked state, and determined the next operation or question.
Workflow and API layer	Executed Air India business flows by calling the appropriate APIs according to the detected intent and the information already collected.
Response composer and TTS	Turned workflow outcomes into natural spoken responses and generated response audio as a file.
Playback watcher / handler	Detected or located the generated audio file and coordinated the return path.
ESL service	Connected externally to FreeSWITCH and sent playback commands so the file could be injected into the active call stream.

6. End-to-End Call Flow

The following flow shows how audio moved from the caller to the bot and then returned to the same active call through the separate control path.

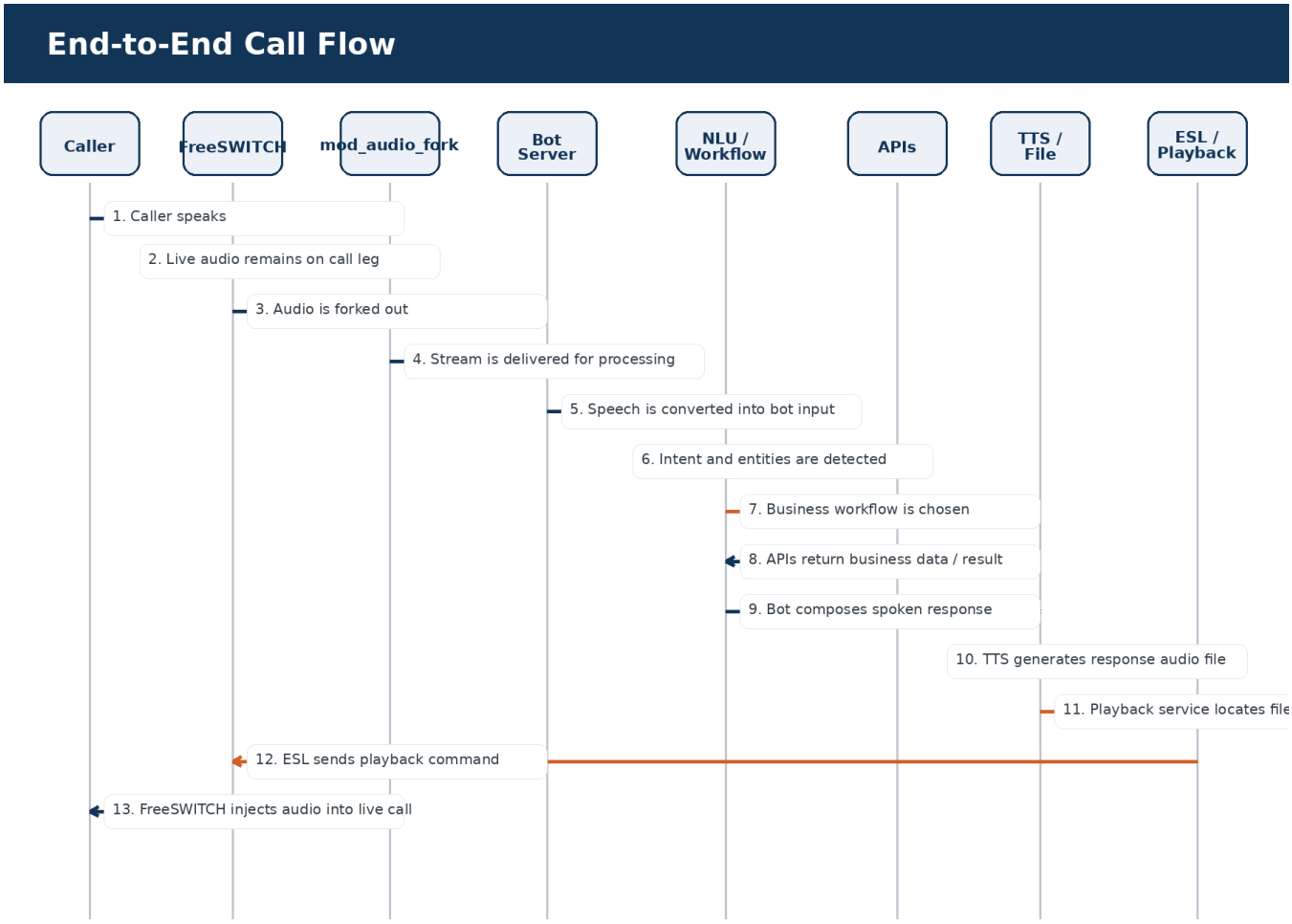

Figure 2. End-to-end execution flow from live caller speech to ESL-based playback return.

1. The caller spoke into an active FreeSWITCH session.
2. FreeSWITCH maintained the call leg while mod_audio_fork copied or forked the media stream to an external processing service.
3. The bot-side service received the audio and transformed it into a representation suitable for language understanding.
4. The NLU layer identified the caller's intent, extracted relevant entities, and used the current dialogue state to determine the next business operation.
5. The workflow layer selected the correct Air India operation and called the required APIs.
6. The result was turned into response text and converted into an audio file through TTS.
7. A dedicated watcher or handler process located that response file and used ESL to instruct FreeSWITCH to play it on the active channel.
8. The caller heard the bot's answer inside the same call, and the next utterance repeated the cycle.

7. NLU, Intent Processing, and Dialogue Logic

One of the core parts of the project was the bot-side NLU layer. This is where raw spoken input stopped being only audio and became an actionable user request. I designed the solution so that intent identification and operation execution were clearly connected.

After the speech input was processed, the NLU layer identified what the caller wanted to do. This included determining the active intent, extracting useful entities from the utterance, and checking whether enough information was already available to proceed with a backend operation.

The dialogue layer also needed state awareness. Airline workflows are usually multi-step. A caller may provide a partial request, respond to a follow-up question, or change the subject mid-conversation. Because of that, the bot could not be stateless; it needed to remember the current stage, previously captured values, and what was still missing before an API call could be made.

NLU responsibilities I implemented or defined

- Intent recognition: classify the user's spoken request into the correct operational route.
- Entity extraction: pull key values from the utterance so the workflow layer could construct valid API requests.
- Slot filling / missing-data detection: determine what critical information was still absent and ask the next question accordingly.
- Dialogue continuity: preserve context across turns instead of treating every utterance as a brand-new request.
- Fallback behavior: handle unclear or low-confidence utterances in a controlled way instead of making incorrect business calls.

This design allowed the bot to operate on intent rather than on isolated keywords. In practical terms, that meant the caller could speak more naturally and the system could still decide what backend operation to perform.

8. Air India Workflow and API Orchestration

Once the bot understood the user's intent, the next layer handled the actual Air India business flow. This was the operational heart of the solution. I mapped detected intents to specific workflow handlers and connected those handlers to the required APIs.

The workflow layer validated whether the required fields were available, selected the correct API or sequence of APIs, formatted the payload, interpreted the response, and turned that response into something usable by the conversation layer.

This separation of NLU and workflow logic was important. NLU answered the question, "What is the caller asking?" The workflow layer answered, "What operation should now be performed, and what should the bot say next?"

Operational pattern

- User speech entered the bot as a real utterance, not as a DTMF menu choice.
- The NLU layer detected the current intent and extracted candidate entities.
- The workflow layer checked which parameters were already present and whether another clarification step was needed.
- When enough data was available, the bot made the correct API call for the Air India workflow.
- The API result was normalized into a conversation-friendly outcome such as confirmation, clarification, status output, or a retry / fallback path.

Why this matters in a portfolio context

This project demonstrates not only telephony integration but also applied backend orchestration. The voice bot was able to perform real operations on user intent, which is materially different from a static IVR. From a portfolio standpoint, this shows capability in conversational systems, state management, business integration, and real-time system design.

9. Response Audio Generation and Playback Return Path

A major practical detail in the implementation was how the response came back into the call. mod_audio_fork was used to send audio out, but it was not the mechanism through which bot responses were directly played back into the call in my design.

Instead, once the bot formed a response, that response was converted into an audio file. A separate process then detected or located the generated file, associated it with the correct call or session, and used ESL to tell FreeSWITCH to play that file on the live channel.

This meant the return path was file-based and control-driven. Outbound audio moved over the media fork; inbound response playback moved over an ESL command path. That design choice made the system easier to coordinate and gave precise control over when and how the bot spoke.

Path	Main mechanism	Purpose
Outbound media	mod_audio_fork	Move live caller audio from FreeSWITCH to the external bot server.
Bot processing	NLU + workflow + APIs	Interpret caller intent and execute the correct airline operation.
Return playback	TTS file + watcher + ESL	Inject the generated response audio back into the active call.

10. Approaches Tried and Why the Design Evolved

This project did not emerge as a single-step design. I explored and worked through multiple patterns before settling on the final architecture. That evolution itself is a meaningful part of the project because it shows practical system design under real constraints.

Approach	Description	Why it helped	Why it was not enough alone
Live audio fork only	Use mod_audio_fork to push caller audio to a remote server.	Proved the media extraction path.	Did not itself provide a complete response playback strategy.
Backend-generated response audio	Generate bot output as an audio file after intent processing.	Made the response deterministic and easy to control.	Still needed a way to inject that file into the active call.
ESL control path	Use ESL to command FreeSWITCH externally.	Enabled precise playback control and session targeting.	Required a separate service to coordinate file readiness and channel context.
Stateful NLU + workflow layer	Operate on user intent rather than simple audio events.	Allowed real Air India actions and follow-up questions.	Increased orchestration complexity and required better session management.
Separated outbound and return paths	Keep audio export and audio return as two different mechanisms.	Created a practical and controllable architecture.	Required careful mapping between files, sessions, and playback timing.

11. Key Engineering Challenges and How I Solved Them

Challenge: separating media transport from conversational control

Solution: I treated audio export and response playback as different channels. mod_audio_fork solved outbound media; ESL solved controlled playback into the same live call.

Challenge: turning caller speech into real backend operations

Solution: I added an NLU and workflow layer so the bot could identify intent, collect required values, and trigger the correct Air India API sequence instead of behaving like a static IVR.

Challenge: managing multi-step conversation state

Solution: the design required context tracking so the bot could ask follow-up questions, preserve collected parameters, and continue the same task over multiple turns.

Challenge: response timing and session mapping

Solution: I used a separate playback handler to find the correct audio file, associate it with the active call, and send the right ESL playback command at the right time.

Challenge: making the solution operationally realistic

Solution: I moved toward a modular architecture so telephony, bot logic, and workflow execution could be reasoned about and improved independently.

12. Technical Stack

Layer	Technologies / responsibilities
Telephony	FreeSWITCH for live call handling and media session control.
Audio export	mod_audio_fork for sending live call audio to the external processing environment.
Call control	ESL for external control over FreeSWITCH sessions and playback operations.
Bot layer	A custom bot-processing service that accepted speech input and drove intent-based operations.
Language understanding	NLU layer for intent recognition, entity extraction, and dialogue-state-aware decisioning.
Business integration	Air India workflow handlers connected to the relevant APIs based on user intent.
Response generation	Response composer plus TTS to create playable audio files.
Return path	A watcher / playback script that located the generated audio file and injected it back into the call via ESL.

13. Outcome and Portfolio Value

This project is a strong portfolio case because it sits at the intersection of multiple difficult areas: telephony, real-time media handling, conversational AI, stateful workflow orchestration, and practical system integration.

It shows that I can move beyond isolated coding tasks and design an end-to-end solution where every layer has a clear responsibility: FreeSWITCH for call control, mod_audio_fork for outbound media, NLU and workflow logic for intent operations, and ESL for controlled response playback.

It also highlights a design mindset grounded in practical engineering. Instead of forcing one module to do everything, I separated concerns and built a working architecture around the real behavior of the tools involved.

14. Key Learnings

- Real-time voice systems work best when media transport and call control are treated as separate concerns.
- A conversational bot becomes much more valuable when it operates on intent and business workflows instead of fixed prompts alone.
- ESL provides a powerful bridge for controlling FreeSWITCH externally when the media-return path must be orchestrated outside the core stream.
- State management is essential for airline-style workflows because real callers provide information over multiple turns, not in one perfect utterance.
- Good architecture often comes from iterating through imperfect approaches and keeping the parts that proved reliable in practice.

Closing note

From a portfolio perspective, this project represents a complete voice-automation system: telephony integration, audio transport, NLU and intent operations, business API orchestration, TTS generation, and response injection into a live call. It demonstrates both breadth across multiple technologies and depth in handling the operational details that make a real-time voice bot actually work.