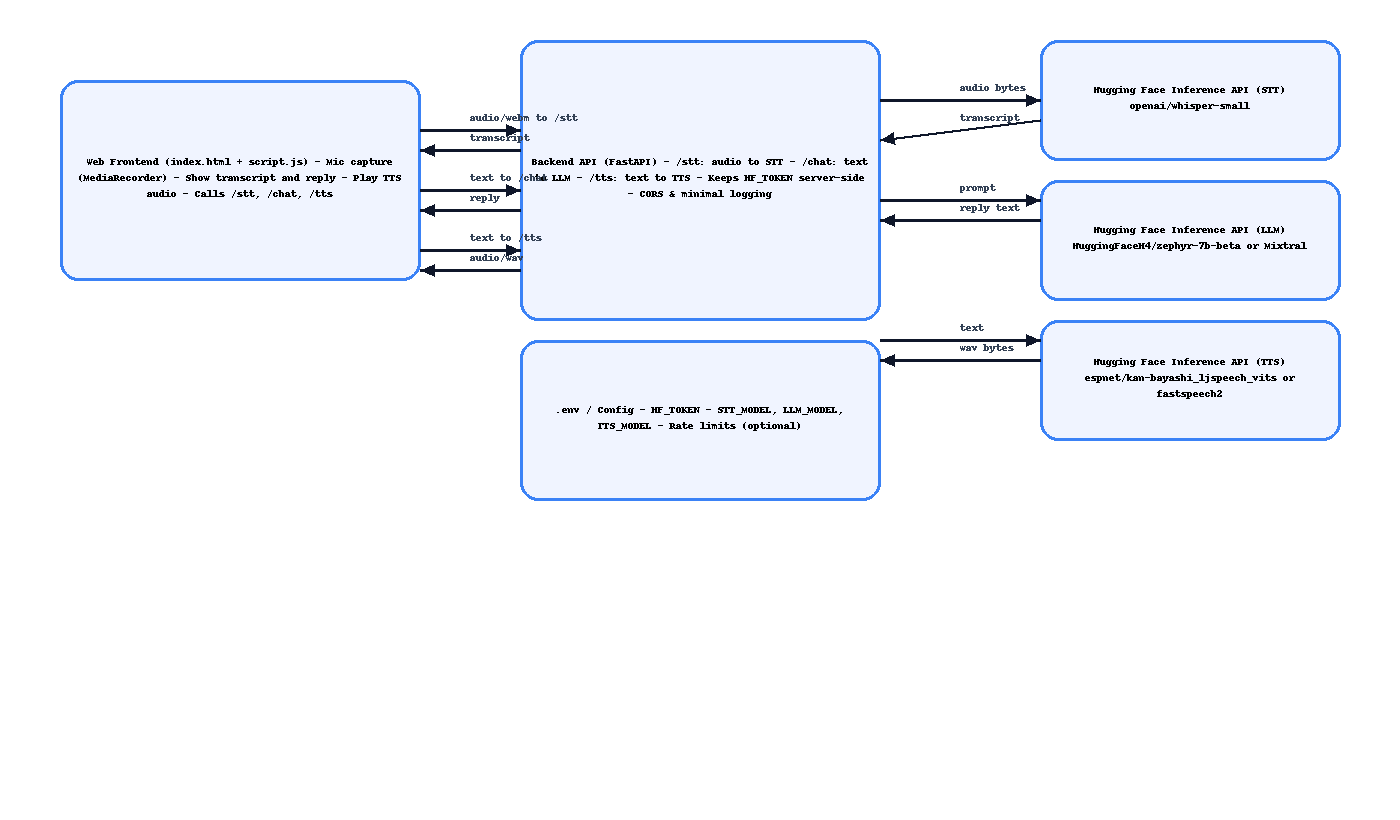
Hosted Hugging Face Voice Assistant — Architecture & Implementation

# Overview

A browser-based Voice Assistant that uses Hugging Face hosted models for Speech-to-Text (STT), a Large Language Model (LLM) for text generation, and Text-to-Speech (TTS). The FastAPI backend calls the Hugging Face Inference API, keeping the HF access token on the server.

# High-level Architecture Diagram



# Components

Web Frontend (index.html + script.js):  
- Mic capture with MediaRecorder  
- Sends audio to /stt  
- Sends transcript to /chat  
- Sends reply to /tts, plays audio

Backend API (FastAPI):  
- /stt: receives audio, forwards to STT model  
- /chat: forwards text to LLM model  
- /tts: forwards text to TTS model, streams WAV  
- Keeps HF\_TOKEN safe on server; CORS for dev

Hugging Face Hosted Models:  
- STT: openai/whisper-small (or whisper-medium)  
- LLM: HuggingFaceH4/zephyr-7b-beta (or Mixtral-8x7B-Instruct)  
- TTS: espnet/kan-bayashi\_ljspeech\_vits (or fastspeech2)

# End-to-End Data Flow

1) Browser records audio and posts to /stt  
2) Backend sends audio to HF STT and returns transcript  
3) Browser posts transcript to /chat; backend returns reply text from LLM  
4) Browser posts reply text to /tts; backend returns audio bytes to play

# API Endpoints (Backend)

/health — GET → service status + model names

/stt — POST multipart/form-data (audio=file) → { text }

/chat — POST application/json { prompt } → { reply }

/tts — POST application/json { text } → audio/wav stream

# Configuration & Environment

Environment variables:  
- HF\_TOKEN (required)  
- STT\_MODEL, LLM\_MODEL, TTS\_MODEL (optional)  
Run locally: uvicorn backend.app:app --reload --host 127.0.0.1 --port 8000

# Security Considerations

- Never expose HF\_TOKEN to the client  
- Restrict CORS to your domain in production; use HTTPS  
- Validate upload size/type; set timeouts and rate limits  
- Minimal logging; avoid storing raw audio unless necessary

# Deployment & Scalability

- Deploy FastAPI on a VM/container; set env vars there  
- Serve frontend as static files (or from same backend)  
- Horizontal scale behind a load balancer  
- Optional caching of frequent TTS outputs

# Future Enhancements

- Streaming STT and TTS  
- Conversation memory (vector DB)  
- Integrations (ServiceNow/Power Automate)  
- Multi-language support and domain guardrails

# How This Project Combines Uvicorn, MCP, and Streamlit

This Voice Assistant project demonstrates concepts from three different ecosystems:  
  
1. Uvicorn (with FastAPI):  
- Acts as the ASGI server that runs the backend.  
- Routes HTTP requests (e.g., /stt, /chat, /tts) to FastAPI functions.  
- Enables the frontend (HTML/JS) to communicate with the backend.  
  
2. MCP-style Role:  
- The backend acts conceptually like an MCP server.  
- It receives user intent (speech converted to text).  
- It routes that intent to the correct tool or model (STT, LLM, TTS) using Hugging Face APIs.  
- Even though it is not a formal MCP server, it plays the same routing role between an agent and external tools.  
  
3. Streamlit (optional alternative):  
- Instead of a custom frontend, Streamlit could provide a simple Python-based interface.  
- This allows quick prototyping of speech input, transcript display, and TTS playback in one script.  
- Streamlit is especially useful for demos and research, while HTML+JS frontend is closer to a production-ready web UI.  
  
Together, the project illustrates:  
- Uvicorn as the router for web requests.  
- MCP concepts as the router for AI/agent tool calls.  
- Streamlit as an optional UI builder for humans.  
  
This combination highlights an understanding of both traditional web backend design and modern AI agent tool orchestration.