# DAY -1

**🔧 What is uv?**

uv is a next-gen Python package manager from Astral. It’s:

* ⚡️ **Ultra fast** (written in Rust)
* ✅ Compatible with requirements.txt or pyproject.toml
* 🧼 Replaces pip, venv, pip-tools, virtualenv, poetry, etc.
* 💻 Used internally at OpenAI

**📦 What is pyproject.toml?**

pyproject.toml is the **modern standard configuration file** for Python projects.  
Think of it like the package.json of Python.

It defines:

* Your **project metadata** (name, version, description)
* Your **dependencies**
* Your **build system**
* Your **tooling config** (e.g., for uv, black, ruff, pytest, etc.)

Here’s a clean summary of everything we’ve done so far — you can paste this directly into your notes:

**✅ Step 2: Production-Grade Python Environment Setup (with uv and pyproject.toml)**

**1. 📦 Installed uv**

Used uv as our modern, fast dependency and virtualenv manager.

Install command:

curl -Ls https://astral.sh/uv/install.sh | sh

**2. 🧪 Created .venv using uv**

Created a virtual environment **inside the project folder** (production-grade approach):

uv venv

source .venv/bin/activate

**3. 🛠️ Created pyproject.toml**

Defined all runtime and dev dependencies in one clean config file.

**Runtime dependencies:**

* fastapi
* uvicorn[standard]
* sounddevice
* numpy
* pydantic

**Dev tools:**

* black → autoformatter
* ruff → linter/static checker
* pytest → testing framework

[project]

name = "scribe-ai"

version = "0.1.0"

description = "Real-time voice-to-note AI scribe"

readme = "README.md"

requires-python = ">=3.9"

dependencies = [

"fastapi",

"uvicorn[standard]",

"sounddevice",

"numpy",

"pydantic"

]

[project.optional-dependencies]

dev = [

"black",

"ruff",

"pytest"

]

[tool.uv]

virtualenvs.in-project = true

[tool.black]

line-length = 88

[tool.ruff]

line-length = 88

target-version = "py39"

select = ["E", "F", "I"]

[tool.pytest.ini\_options]

addopts = "-ra -q"

testpaths = ["tests"]

**4. ⚙️ tool.\* in pyproject.toml**

These are **config sections** for dev tools:

* tool.black → sets line length for autoformatting
* tool.ruff → sets linter rules for unused imports, formatting, etc.
* tool.pytest → controls pytest behavior
* tool.uv → makes sure .venv/ is local to the project

**5. 📥 Installed all dependencies (main + dev tools)**

uv pip install .[dev]

This installs everything:

* Runtime dependencies from [project.dependencies]
* Dev tools from [project.optional-dependencies].dev

A **WebSocket** is a special communication protocol that allows **real-time, two-way interaction** between a **client** (like a web browser or mobile app) and a **server**.

Unlike normal HTTP requests (which are one-way: client asks, server replies), WebSockets keep the connection **open**, so **both** sides can send data anytime — perfect for **live updates** like speech apps, chat apps, games, or notifications.

🧠 Think of it like a **phone call**, not sending letters:

* HTTP = letter-based communication (slow, one-way)
* WebSocket = live phone call (fast, two-way)

In a **speech app**, you need real-time interaction. Here’s why HTTP isn't enough:

* **HTTP**: Every time you want to send data (like a chunk of audio), you must open a new connection, send it, and close it. That creates **delays** and **overhead**.
* **WebSocket**: You open **one connection**, and then you can stream data **continuously** in both directions — **perfect for sending live audio and getting live transcripts** back without waiting.

💡 For example:

* Your app sends audio chunks as the user speaks (WebSocket → Server).
* Server replies with transcribed text instantly (Server → WebSocket).

This **back-and-forth flow** is what makes speech feel live

Your **FastAPI app will act as a WebSocket server**. That means:

* You define a **special route** like /ws/audio-stream
* But unlike REST routes (@app.get, @app.post), you use @app.websocket
* This route does **not** expect a request → response flow — it stays open

📌 Inside this WebSocket route:

* You’ll **wait for audio chunks** from the client (await websocket.receive\_bytes() or .receive\_text()).
* You **process** each chunk (e.g., speech-to-text using Whisper/OpenAI).
* You **send back transcription** using await websocket.send\_text().

🔹 Step 6: What is WebRTC? (And how is it different from WebSocket?)

WebRTC (Web Real-Time Communication) is a protocol for direct peer-to-peer communication between browsers or devices. It's used for:

🔊 Voice calls (e.g., Google Meet, Zoom)

🎥 Video streaming

📁 File sharing (without server)

🧠 WebRTC is optimized for media (audio/video) and includes:

Echo cancellation

Latency control

NAT traversal (to go through firewalls)

Encryption (by default)

**At the simplest level, a mic is just a hardware device that listens to sound and converts it into a digital signal your computer or microcontroller can understand.**

So we need to understand the hardware setup better and also start coding to use to laptop mike as a starting point