Brainstorming of the tasks

Tair:
Set up the basic database structure for users, notes, and recordings.
(Database implementation:DB waitlist table, DB notes table, DB users table, …)
Routing: Script for waitlist, script for auth, db <-> frontend communication
Design: Creating a consistent color scheme, Designing a casing for hardware, 3D Printing case for hardware
Hardware: Soldering components

(System scope & decisions

Define **MVP scope** explicitly (what is the *first demoable version*?)

Decide **single-user vs multi-user** flow (one device → one account or shared?) (i suggest single)

Decide **real-time** vs **batch transcription** (short chunking in MVP, for testing batch transcription)

Define **output format** (raw transcript vs summary vs action points) -> (summary prepared for Feedpulse format)

Decide **where “truth” lives** (device, backend, FeedPulse, app DB)

Error states & fallback behavior (no Wi-Fi, no API, low battery)

Decide whether the device works only with an internet connection or also supports offline recording. (i suggest only with an internet without storing audio in long term OR maybe storing certain audio on SD before summary compilation then clearing it.))

Backend / Infrastructure:
Audio file storage strategy (raw audio stored or discarded?)
API contract between ESP32 ↔ backend ↔ frontend
Rate limiting / API cost control (Gemini usage)
Logging & observability (basic logs, error tracking)
Deployment environment (local only vs cloud-hosted MVP) (first local, then cloud-hosted as could have)
Security basics (API keys storage, device auth token)

AI / Transcription Intelligence (Critical depth):
Audio preprocessing (noise reduction, silence trimming)
Chunking strategy for long audio
Speaker diarization (even basic “speaker A / B”) (No, cause we will use 1 microphone for mvp)
Prompt design for **educational context** (meetings vs lectures)
Evaluation criteria (how do we know transcription is “good enough”?)
Latency vs quality tradeoff
Cost benchmarking (Gemini Flash vs others)
Bias / hallucination mitigation (especially summaries)

Hardware & Firmware (Risk-heavy area):
Beyond “setup ESP32”:

- Power management & battery life estimation
- On-device buffering strategy
- Button / gesture interaction definition (start/stop recording) (in app action/physical trigger)
- LED / haptic feedback (recording status, errors) (eu regulations + Hardware UI basically)
- Firmware update strategy (OTA or wired only) - i think wired only
- Heat, enclosure airflow considerations
- Debug interface access after casing is closed
- Hardware failure modes (mic disconnects, SD corruption)

Frontend / UX (More than “coding UI”):

- User journey mapping (first-time user → first transcript)

Anita:
Design the web app UI/UX in Figma
Define the structure of the app
Implement the core pages
Connect the app to Supabase to load and display saved notes

Francisco:

Making the AI (Gemini API) transcribe/summarize phrases
AI transcription from audio into a phrase
Connecting the AI to the microphone
Coding the ESP32 on Arduino IDE
Topic Segmentation
Context Awareness
Adaptive Summarization
LED changing color from recording status (information)

Marcell:
Hardware

- Setting up ESP32 and connecting the necessary components
- Efficiently designing the circuit the orientation of components
- Setting up the firmware for ESP32
- Record start and stop button
- Coding and helping with the connection of the database, backend, frontend, and hardware

Ozan:

Front end for the home page the transcription page

Deciding what the case is going to look like

Designing the model for the case

Connecting the hardware to the app

Define how the app is going to look like

Display battery life on app

Display connection status in app

---

*From top to bottom priority level.

Must have:

Hardware Device
Store data from the device
Sending data to DB.
basic database structure for users, notes, and recordings.
Branding (presentation, posters, design)
Chunking strategy for long audio
Logging & observability (basic logs, error tracking)
Figma design
On-device buffering strategy
Web app UI to view and manage transcribed notes
Illustrating data (basic visual representation of notes/usage)
User journey mapping (first-time user → first transcript)
LED / haptic feedback (recording status, errors) (eu regulations + Hardware UI basically)
Button first start/stop recording trigger
Documentation
Error feedback clarity (AI failed, device offline, etc.) (first in console, then in app)
Security basics (API keys storage, device auth token)
API contract between ESP32 ↔ backend ↔ frontend

Should Have:

Transcription with AI cloud
Audio preprocessing (noise reduction, silence trimming)
Evaluation criteria (how do we know transcription is “good enough”?)
Latency vs quality tradeoff
Frontend:

Navigation
Accessibility (readability, font, contrast, color scheme)
Mobile implementation priority (2nd desktop)
Consistency between hardware + app feedback
Illustrating Data.
Notes management dashboard (list of notes, specific note page (summary + CRUD))
gesture interaction in app (start/stop recording)

Could have:

User Friendly Casing (3d modeled, printed)
Topic segmentation
Prompt design for **educational context** (meetings vs lectures (feedpulse oriented))
Clear UX feedback for system states
Heat, enclosure airflow considerations
Dark mode/light mode UI
Battery life estimation and power management
Audio file storage strategy (raw audio stored or discarded?)

Won’t have:

Offline usage without internet connection
Enterprise scalability