# CS4518 Mobile & Ubiquitous Computing – Term Project Proposal

**Project Title: Transcribe: Real-Time Speech-to-Text Mobile App**
**Course:** CS4518 Mobile & Ubiquitous Computing – D-Term 2025
**Team Members:** Utku Yakar, Karish Gupta
**Date:** April 29, 2025

**Idea:** Speech to text app that uses the android device's microphone and display verbal communication as text

**High Level Design:**

- **Main screen**
  - Enables users to transcribe speech to text. The main screen should have a large microphone button, that when pressed starts recording the user's audio. There should be some indication that audio is being recorded. Then the user speaks and the interpreted text pops up on the same screen.
- **History screen**
  - There should be an option to move to a second page that stores the history of past transcriptions. These transcriptions are displayed in order of most recent to earlier transcriptions as the user scrolls up.
- **Speech to text model**
  - Pre-trained model can be imported from Hugging Face and more fine-tuning can potentially be done to help improve the model by training on more audio to text data from large Hugging Face datasets.
- **Server**
  - A python server can be run locally to take in POST requests from the app and then pass these to the model. Then the response would be sent to the app with the model output transcribed text

## 1) Problem Statement

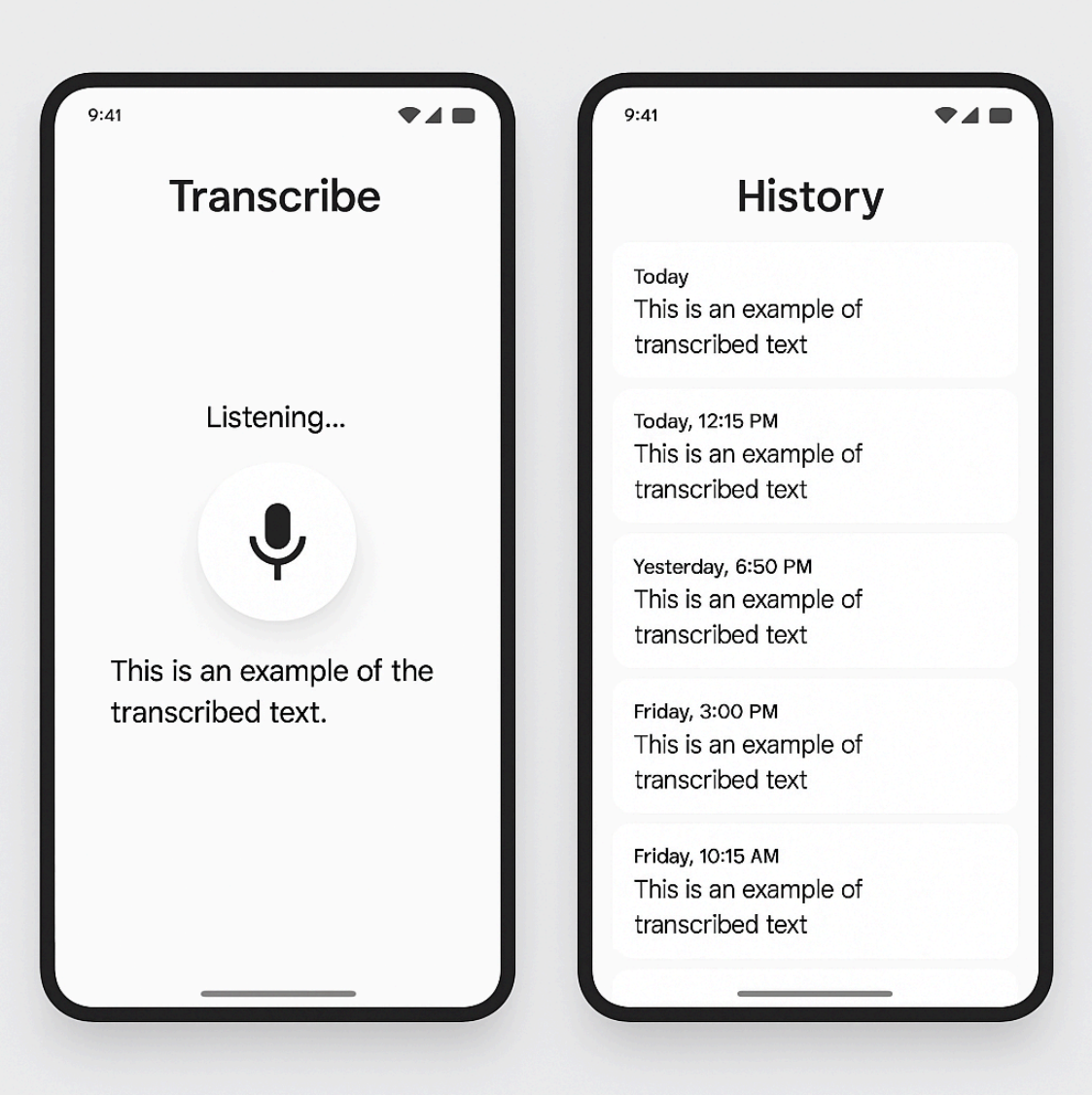
Many conversations, lectures, and videos still exclude people who are hard-of-hearing or operate in noisy environments. Existing captioning solutions often require an internet connection, add significant latency, or store user data on third-party servers. LiveCaptioner aims to provide on-device, low-latency transcription and a searchable history of all captured text, giving everyone instant visual access to the spoken word.

## 2) Motivation

- **Accessibility** – empower deaf and hard-of-hearing users with live captions anywhere (classroom, cafe, public transport).
- **Productivity** – turn spontaneous meetings or voice memos into searchable notes without manual typing.
- **Privacy-first** – perform speech recognition locally or on a user-controlled LAN server, avoiding cloud storage of raw audio.

## 3) High-Level Feature Set

| Core Feature | Description | Category |
|---|---|---|
| **Live Transcription Screen** | Tap a microphone to start/stop capture; live text scrolls in real time. | UI + Sensing |
| **History Screen** | Reverse-chronological list of prior transcripts with timestamp & length; tap to view full text. | UI + Persistence |
| **On-device Speech API or Local Whisper Server** | Choice between Android SpeechRecognizer (offline) or Retrofit calls to a LAN Python service running Whisper-tiny from Hugging Face. | Sensing + Processing |
| **Keyword Search & Share** | Full-text Room query; share any transcript via Android Sharesheet. | UI + Persistence |
| **Robust Rotation Handling** | Transcription continues across configuration changes. | Persistence |




Design moc for transcribe app. This shows the main screen when listening and displaying the transcribed text and the history screen displaying past transcriptions and when they occurred.

## 4) Technical Approach & Requirement Mapping

### 4.1 UI Complexities (≥ 2 screens, ≥ 2 ViewGroups, Intent extras)

| Requirement | Planned Implementation |
|---|---|
| ≥ 2 screens | MainFragment (live capture) and HistoryFragment (RecyclerView list). |
| ≥ 2 ViewGroups | ConstraintLayout for Main, RecyclerView (with LinearLayoutManager) inside CoordinatorLayout for History. |
| Intent extras | On list-item click, send transcript id & snippet via Intent to TranscriptDetailsActivity. |

### 4.2 Mobile Sensing & Processing

- **Primary Sensor:** Android microphone accessed through the SpeechRecognizer API for fully offline automatic speech recognition (ASR).

- **Optional External Processing:** Retrofit 2 client sends compressed WAV or FLAC to a FastAPI endpoint running the Whisper-tiny model on a local-area-network (LAN) server. The user can switch between on-device and LAN modes in Settings.

- **Data Path Overview:**

    - Audio captured in a foreground service.

    - In on-device mode, SpeechRegocnizer returns partial results that stream into a MutableStateFlow.

    - In LAN mode, the same audio buffer is forwarded, and the JSON response is parsed into the flow.

    - UI observes the flow for real-time caption updates.

### 4.3 Data Persistence

- **Configuration-safe Transcription:** The recording service is bound to the activity; it survives rotations. UI state (current transcript text) is exposed via ViewModel + Flow, so reconstruction is automatic.

- **Long-Term Storage:** Room database with a Transcript entity (id, text, startTime, duration). Full-text search (FTS5) is enabled for keyword queries.

- **User Preferences:** Jetpack datastore holds settings such as preferred ASR backend, language, and retention policy.

- **Retention Policy:** Old transcripts (default 30 days) are pruned by a WorkManager task that runs once every 24 hours.

## 5) System Architecture

**App Architecture:**

| Layer | Technology | Responsibility |
|---|---|---|
| Presentation | Jetpack Compose (Material 3) | Reactive UI, navigation, theming |
| Domain | Kotlin Coroutines + Flow | Business logic, error handling |
| Data | Room 2.7, DataStore | Local persistence, preferences |
| Platform / Network | SpeechRecognizer, Retrofit, FastAPI (Whisper) | Audio capture and optional LAN inference |
| Dependency | Hilt DI, Gradle KTS | Modular build, test injection |

**“Cloud” Architecture:**

| Layer | Technology | Responsibility |
|---|---|---|
| Server (Run locally) | Python, Flask | Accept POST requests and respond with model output |
| Cloud Model | Whisper-tiny model, PyTorch, Transformers | Takes in audio data received in POST request, processes data, and outputs transcription text |
| On-device | TFLite | Takes in audio data directly from device, process data, and output transcription text |