MSD Project Plan - Captain’s Log

Partners: Brandon Mountan & Thanmay Lakkireddy

Pitch:

Hey, I just need a coder to build this super cool idea. I tried to vibe code it but that didn't work because I guess my prompt wasn't good enough. Oh, I can't pay you for the work, but the experience and exposure will be invaluable!

I want you to build a "captain's log" app like Star Trek (basically a voice recorder). Users should be able to record captains log notes. They need to be able to play back the audio, but I also want them to be able to read transcriptions too. I want this to be a social app, so users should be able to have "friends" in the app and be able to share log notes with them, too. Oh, and I want a search feature that lets users find log entries, like, "what did I think about the gamma quadrant?" or "what were Spock and Bones arguing about?"

Before I agree to choose you for this unpaid project, I need you to give me a detailed design that I'll run by my cousin who's a software engineer, but also I want a video and prorotype app. Thanks! Oh yeah, huh, yeah, copyright, let's not worry about that. CBS doesn't seem litigous

Requirements:

1. Login page and prompt with signup button below.
2. After signup, go back to login page.
3. Home page has list of logs with each log having an expandable feature that shows the transcription.
4. Each log also has a play button? Or is it easier to have one play button and the desired log needs to be selected?
5. Each log also has a share button to share with friends.
6. Search bar is included in the log screen as well.
7. The Log screen has a record button that takes user to another page.
8. Recording page is only for recording. Start, stop, save, delete.
9. Has home button and friends button so user can navigate.
10. Friends page displays list of friends with add, delete options.

Plan: Part One

The Android App consists of proposed signatures for the relevant Activity, Composables, ViewModels, Repositories, and other classes you think you'll need to implement

Activities

- Main Activity - Single activity hosting all Compose screens.
    - class MainActivity : ComponentActivity()
        - onCreate(): Initialize app and set Compose content
        - requestPermissions(): Request audio recording and storage permissions

Composables

- LogScreen
    - Purpose: Display list of your log entries, log button, friends button. Each entry has a play button and can expand to display transcription. Simple UI
    - Parameters: viewModel: LogsViewModel
- RecordScreen
    - Purpose: Record new audio log. Start, stop, delete, save buttons.
    - Parameters: viewModel: RecordViewModel
- FriendsScreen
    - Purpose: Manage friends list. Send/recieve requests
    - Parameters: viewModel: FriendsViewModel

ViewModels

ViewModels manage UI state and business logic. Uses StateFlow to update

- RecordViewModel
    - class RecordViewModel(private val logRepo: LogRepository, private val audioRecorder: AudioRecorder)
        - State:
            - isRecording: StateFlow<Boolean>
            - recordingDuration: StateFlow<Int>
            - uploadStatus: StateFlow<UploadState>
        - Methods:
            - fun startRecording()
            - fun stopRecording()
            - Suspend fun saveLog(title: String)

- LogsViewModel
    - class LogsViewModel(private val logRepo: LogRepository, private val audioPlayer: AudioPlayer)
        - State:
            - logs: StateFlow<List<LogEntry>>
            - searchQuery: StateFlow<String>
            - expandedLogId: StateFlow<String?>

            - currentlyPlaying: StateFlow<String?>
        - Methods:
            - suspend fun loadLogs()
            - fun updateSearchQuery(query: String)
            - suspend fun search()
            - fun toggleExpanded(logId: String)
            - fun playLog(logId: String)
            - fun pauseLog()
            - suspend fun shareLog(logId: String, friendsIds: List<String>)

- FriendsViewModel
    - class FriendsViewModel(private val friendsRepo: FriendsRepository)
        - State:
            - Friends: StateFlow<List<Friend>>
            - pendingRequests: StateFlow<List<FriendRequest>>
        - Methods:
            - suspend fun loadFriends()
            - suspend fun sendFriendRequest(username: String)
            - suspend fun acceptFriendRequest(requestId: String)

Repositories

Repo's deal with the data and communicate with API's

- LogRepository
    - class LogRepository(private val apiService: ApiService, private val localDb: LogDatabase)
        - Methods:
            - suspend fun createLog(audioFile: File, title: String): LogEntry
            - suspend fun getLogs(searchQuery: String? = null): List<LogEntry>
            - suspend fun shareLog(logId: String, friendsIds: List<String>)
- FriendsRepository
    - class FriendsRepository(private val apiService: ApiService)
        - Methods:
            - suspend fun getFriends(): List<Friend>
            - suspend fun sendFriendRequest(username: String)
            - suspend fun acceptFriendRequest(requestId: String)
            - suspend fun getPendingRequests(): List<FriendRequests>
- AuthRepository
    - class AuthRepository(private val apiService: ApiService)
        - Methods:
            - suspend fun login(username: String, password: String): String
            - suspend fun logout()

Other Classes

- AudioRecorder
    - class AudioRecorder(private val context: Context)
        - Methods:
            - fun startRecording(outputFile: File)
            - fun stopRecording(): File
            - fun getRecordingDuration(): Int
- AudioPlayer
    - class AudioPlayer()
        - Methods:
            - fun play(audioUrl: String)
            - fun pause()
            - fun seekTo(position: Int)
            - fun getCurrentPosition(): Int

Plan: Part Two

For backend systems, API routes that will be exposed to clients, description/API of services, description of what containers will be running, and how they will communicate with each other.

Room and LDAP

API routes

Method	**Endpoint**	**Request**	**Response**
POST	/api/auth/login	{username, password}	{sessionId, userId}
POST	/api/logs	audioFile, title	LogEntry object
GET	/api/logs	query parameter	Array of LogEntry
POST	/api/logs/:id/share	{friendsIds: [string]}	{success: true}
GET	/api/friends	None	Array of Friend
POST	/api/friends/request	{username: string}	FriendRequest object
POST	/api/friends/accept/:id	None	{success: true}

Description/API of services

- AuthService
    - Handle user authentication with LDAP
        - Methods:
            - authenticateLDAP(username, password): Promise<User>
            - createSession(userId): Promise<String>
            - validateSession(sessionId): Promise<User>
- LogService
    - Manage log entries, storing audio, transcription, and search
        - Methods:
            - createLog(userId, audioFile, title): Promise<LogEntry>
            - getUserLogs(userId, searchQuery?): Promise<LogEntry[]>
            - shareLogWithFriends(logId, friendsIds): Promise<void>
- TranscriptionService
    - Converts audio logs to text. OpenAI Whisper AI
        - Methods:
            - transcribeAudio(audioFile): Promise<string>
- FriendsService
    - Manage friends, friends list, and friend requests
        - Methods:
            - getFriends(userId): Promise<Friend[]>
            - sendFriendRequest(fromUserId, to Username): Promise<FriendRequest>
            - acceptFriendRequest(requestId): Promise<void>
            - getPendingRequests(userId): Promise<FriendRequest[]>

Containers and Communication

- Backend will run on Docker. Containers communicate through a Docker network managed by Docker Desktop. The Ktor server connects to LDAP via internal DNS and makes outbound HTTPS calls to OpenAI API for transcription and embeddings. The audio files are stored locally on the Android device and uploaded temporarily to the backend only for transcription processing
- Containers:
    - Ktor API Server Container
    - LDAP Server Container: user authentication

Plan: Part Three

For any AI models that are being proposed, propose a pre-trained model that can be used, and describe any fine-tuning that might need to be performed to adapt it. Be specific about what data you think would be required to perform any necessary fine-tuning

<u>API's and fine-tuning</u>

- Speech-to-Text Transcription
- Model: OpenAI Whisper API
    - Whisper is a pre-trained model that works well for basic speech recognition.
- Fine-tuning: no fine-tuning initially. In the future, we could fine-tune on domain-specific vocabulary
- Data required: Hundreds of hours of audio with accurate transcriptions
- Semantic Search
- Model: OpenAI text-embedding 3-small or sentence transformers/all-MiniLM-L6-v2
    - Both of these convert text into vector embeddings
- Fine-tuning: no fine-tuning initially. If there are similarities in logs, then we could fine-tune using pairs of log entries that are similar.
- Data required: Thousands of pairs of similar log entries.

Plan: Part Four

Any risks in terms of technologies you'll need to use that you're unfamiliar with

1. OpenAI API Integration
2. Implementing semantic search
3. File storage

Plan: Part Five

A <u>Prototype</u> of the frontend system (in a git repo)

1. *This can rely on hardcoded data, and can use some placeholder elements*
2. *It should be enough that you can "demo" the app and explain what it does and*
3. *what it "would do"*
4. *As an example, you can replace an API call with {delay(500); use(hardcodedValue)}*

Plan: Part Six

An 8-10 minute video presentation that you might show to the client who asked you for a bid on this project. It should describe the proposed app organization and components, and show the functional features of your demo to explain how the app "would work"

Deadlines:

- Team formation: tell me by Monday, Oct 27
- The detailed plan is due by Thursday, Oct 30
- The demo + video are due by Thursday, Nov 6
- Classmate plans to review on Oct 31
- My review due Nov 6