

```

// ----- Phase 1: Offline AI + Document Ingestion + Dynamic TTS -----
package com.monster.study.ai

import android.content.Context
import android.speech.tts.TextToSpeech
import java.util.*
import org.apache.pdfbox.pdmodel.PDDocument
import org.apache.pdfbox.text.PDFTextStripper
import kotlin.random.Random

// ----- Document Model -----
data class Document(
    val id: String,
    val title: String,
    val text: String
)

// ----- Text Normalization -----
object TextProcessor {
    private val stopWords =
        setOf("the", "is", "a", "an", "and", "or", "to", "of", "in", "on", "for", "with", "as", "by", "at")

    fun normalize(text: String): List<String> =
        text.lowercase(Locale.US)
            .replace(Regex("[^\\p{L}\\p{N}\\s]"), " ")
            .split(Regex("\\s+"))
            .filter { it.length > 2 && it !in stopWords }
}

// ----- PDF Loader -----
object PdfLoader {
    fun load(context: Context, asset: String, id: String): Document {
        val pdf = PDDocument.load(context.assets.open(asset))
        val text = PDFTextStripper().getText(pdf)
        pdf.close()
        return Document(id, asset, text)
    }
}

// ----- Offline AI Engine -----
class OfflineAI(private val docs: List<Document>, private val tts: TextToSpeech? = null) {

    private val invertedIndex = mutableMapOf<String, MutableSet<String>>()
    private val docTokens = mutableMapOf<String, List<String>>()

```

```

private val docMap = docs.associateBy { it.id }

init {
    docs.forEach { doc ->
        val tokens = TextProcessor.normalize(doc.text)
        docTokens[doc.id] = tokens
        tokens.toSet().forEach { token ->
            invertedIndex.getOrPut(token) { mutableSetOf() }.add(doc.id)
        }
    }
}

// ----- Ask a query -----
fun ask(query: String): String {
    val queryTokens = TextProcessor.normalize(query)
    val scores = mutableMapOf<String, Int>()
    queryTokens.forEach { token ->
        invertedIndex[token]?.forEach { docId ->
            scores[docId] = (scores[docId] ?: 0) + 1
        }
    }
    val bestDocId = scores.maxByOrNull { it.value }?.key ?: return "No relevant information found."
    val responseText = docMap[bestDocId]?.text?.take(1000) ?: "No relevant information found."
    speak(responseText)
    return responseText
}

// ----- Speak with dynamic TTS -----
private fun speak(text: String) {
    tts ?: return
    val pitch = Random.nextDouble(0.9, 1.2).toFloat()
    val rate = Random.nextDouble(0.9, 1.1).toFloat()
    tts.setPitch(pitch)
    tts.setSpeechRate(rate)
    tts.speak(text, TextToSpeech.QUEUE_FLUSH, null, null)
}

// ----- Optional: Get document text directly -----
fun getDocumentText(id: String): String? = docMap[id]?.text
}

// ----- Document Repository -----

```

```

object DocumentRepository {
    private val docs = mutableListOf<Document>()
    private var aiInstance: OfflineAI? = null
    private var ttsInstance: TextToSpeech? = null

    // Initialize with documents and optional TTS
    fun initialize(context: Context, initialDocs: List<Document>, tts: TextToSpeech? = null) {
        docs.clear()
        docs.addAll(initialDocs)
        ttsInstance = tts
        aiInstance = OfflineAI(docs, ttsInstance)
    }

    // Add a new document and rebuild AI
    fun addDocument(doc: Document) {
        docs.add(doc)
        aiInstance = OfflineAI(docs, ttsInstance)
    }

    fun getAI(): OfflineAI? = aiInstance
    fun getAllDocs(): List<Document> = docs.toList()

    // Load a PDF asset and add to repository
    fun loadPdf(context: Context, asset: String, id: String) {
        val doc = PdfLoader.load(context, asset, id)
        addDocument(doc)
    }
}

// ----- Phase 2: Flashcards, Cloze, MCQ, Active Recall -----
package com.monster.study.revision

import com.monster.study.ai.DocumentRepository
import kotlin.random.Random
import android.speech.tts.TextToSpeech
import java.util.*

// ----- Flashcard Model -----
data class Flashcard(
    val id: String,
    val question: String,
    val answer: String,
    val category: String = "Uncategorized",
    var lastReviewed: Long = 0,
    var confidence: Int = 0 // 0 = unknown, higher = more confident

```

)

// ----- Flashcard Engine -----

object FlashcardEngine {

private val flashcards = mutableListOf<Flashcard>()

private const val CLOZE_BLANK = "____"

private var tts: TextToSpeech? = null

// Initialize TTS for reading flashcards

fun setTTS(ttsInstance: TextToSpeech) {

tts = ttsInstance

}

// ----- Generate flashcards from documents -----

fun generateFromDocuments() {

flashcards.clear()

DocumentRepository.getAllDocs().forEach { doc ->

val sentences = doc.text.split(Regex("\\.\\s+")).filter { it.length > 20 }

sentences.forEachIndexed { index, s ->

// Simple Cloze: remove a random word

val words = s.split(" ")

if (words.size < 5) return@forEachIndexed

val i = Random.nextInt(words.size)

val question = words.toMutableList().apply { this[i] = CLOZE_BLANK }.joinToString(" ")

flashcards.add(Flashcard("\${doc.id}_\$index", question, words[i], doc.title))

}

}

}

// ----- Get next flashcard -----

fun getNextFlashcard(): Flashcard? {

if (flashcards.isEmpty()) return null

// Prioritize low-confidence cards first

val card = flashcards.minByOrNull { it.confidence }

card?.lastReviewed = System.currentTimeMillis()

card?.let { speakQuestion(it) }

return card

}

// ----- Mark flashcard result -----

fun markFlashcard(id: String, correct: Boolean) {

flashcards.find { it.id == id }?.apply {

confidence = if (correct) confidence + 1 else maxOf(confidence - 1, 0)

```

        lastReviewed = System.currentTimeMillis()
    }
}

// ----- Generate MCQ -----
fun generateMCQ(flashcard: Flashcard, options: Int = 4): Pair<String, List<String>> {
    val correct = flashcard.answer
    val distractors = mutableSetOf<String>()
    val words = DocumentRepository.getAllDocs().flatMap { it.text.split(" ") }
        .filter { it.length > 2 && it != correct }
    while (distractors.size < options - 1 && words.isNotEmpty()) {
        distractors.add(words.random())
    }
    val allOptions = (distractors + correct).shuffled()
    speakQuestion(flashcard)
    return Pair(flashcard.question, allOptions)
}

// ----- Recap Mode -----
fun recap(limit: Int = 5) {
    val recapText = DocumentRepository.getAllDocs()
        .flatMap { it.text.split(". ") }
        .take(limit)
        .joinToString(". ")
    speakText(recapText)
}

// ----- Random Cloze Review -----
fun getRandomCloze(): Flashcard? = flashcards.randomOrNull()?.also { speakQuestion(it) }

// ----- TTS Helpers -----
private fun speakQuestion(card: Flashcard) {
    speakText(card.question)
}

private fun speakText(text: String) {
    tts ?: return
    val pitch = Random.nextDouble(0.9, 1.2).toFloat()
    val rate = Random.nextDouble(0.9, 1.1).toFloat()
    tts.setPitch(pitch)
    tts.setSpeechRate(rate)
    tts.speak(text, TextToSpeech.QUEUE_FLUSH, null, null)
}

```

```

// ----- Get all flashcards -----
fun getAllFlashcards(): List<Flashcard> = flashcards.toList()
}

// ----- Phase 3: Incremental Learning & Dynamic Updates -----
package com.monster.study.learning

import com.monster.study.ai.Document
import com.monster.study.ai.DocumentRepository
import com.monster.study.revision.FlashcardEngine
import com.monster.study.revision.Flashcard
import kotlin.random.Random

// ----- Incremental Learning Manager -----
object LearningManager {

    // ----- Add a single new document -----
    fun addDocument(doc: Document) {
        DocumentRepository.addDocument(doc)    // Update offline AI
        FlashcardEngine.generateFromDocuments() // Refresh flashcards to include new content
    }

    // ----- Add multiple documents at once -----
    fun addDocuments(docs: List<Document>) {
        docs.forEach { DocumentRepository.addDocument(it) }
        FlashcardEngine.generateFromDocuments()
    }

    // ----- Track user queries for adaptive learning -----
    data class QueryLog(val query: String, val response: String, val timestamp: Long =
System.currentTimeMillis())
    private val queryHistory = mutableListOf<QueryLog>()
    private const val MAX_HISTORY = 100

    fun logQuery(query: String, response: String) {
        queryHistory.add(QueryLog(query, response))
        if (queryHistory.size > MAX_HISTORY) queryHistory.removeAt(0)
    }

    // ----- Suggest documents relevant to past queries -----
    fun suggestRelevantDocuments(): List<Document> {
        if(queryHistory.isEmpty()) return emptyList()
        val recentTokens = queryHistory.takeLast(10).flatMap { it.query.split(" ") }
        return DocumentRepository.getAllDocs().filter { doc ->
            val docTokens = doc.text.split(" ")

```

```

        recentTokens.any { it in docTokens }
    }
}

// ----- Update flashcard confidence based on query history -----
fun adaptFlashcardsBasedOnQueries() {
    val recentQueries = queryHistory.takeLast(20)
    recentQueries.forEach { q ->
        val flash = FlashcardEngine.getNextFlashcard()
        flash?.let {
            // If the user's query contains the answer, increase confidence
            if (q.query.contains(it.answer, ignoreCase = true)) {
                FlashcardEngine.markFlashcard(it.id, true)
            }
        }
    }
}

// ----- Optional: Random review prompt -----
fun promptRandomReview() {
    val card = FlashcardEngine.getRandomCloze()
    card?.let {
        // can integrate TTS reading or pop-up in UI
        println("Review Card: ${it.question} -> ${it.answer}")
    }
}

// ----- Clear query history -----
fun clearHistory() {
    queryHistory.clear()
}

// ----- Phase 4: Memory & Personalized Storage -----
package com.monster.study.memory

import android.content.Context
import com.monster.study.ai.DocumentRepository
import java.io.File
import org.json.JSONArray
import org.json.JSONObject

// ----- Memory Entry -----
data class MemoryEntry(
    val key: String,

```

```

    val value: String,
    val timestamp: Long = System.currentTimeMillis()
)

// ----- Memory Manager -----
object MemoryManager {

    private val memory = mutableListOf<MemoryEntry>()
    private const val FILE_NAME = "monster_memory.json"

    // ----- Load memory from local storage -----
    fun load(context: Context) {
        val file = File(context.filesDir, FILE_NAME)
        if (!file.exists()) return
        val array = JSONArray(file.readText())
        memory.clear()
        for (i in 0 until array.length()) {
            val obj = array.getJSONObject(i)
            memory.add(MemoryEntry(obj.getString("key"), obj.getString("value"),
obj.getLong("timestamp")))
        }
    }

    // ----- Save memory to local storage -----
    fun save(context: Context) {
        val array = JSONArray()
        memory.forEach {
            val obj = JSONObject()
            obj.put("key", it.key)
            obj.put("value", it.value)
            obj.put("timestamp", it.timestamp)
            array.put(obj)
        }
        File(context.filesDir, FILE_NAME).writeText(array.toString())
    }

    // ----- Remember a new fact -----
    fun remember(context: Context, key: String, value: String) {
        memory.removeAll { it.key.equals(key, ignoreCase = true) } // overwrite if exists
        memory.add(MemoryEntry(key, value))
        save(context)
    }

    // ----- Recall a fact -----

```

```

fun recall(key: String): String? {
    return memory.find { it.key.equals(key, ignoreCase = true) }?.value
}

// ----- Forget a fact -----
fun forget(context: Context, key: String) {
    memory.removeAll { it.key.equals(key, ignoreCase = true) }
    save(context)
}

// ----- List all memory entries -----
fun listAll(): List<MemoryEntry> = memory.toList()

// ----- Optional: Search memory by keyword -----
fun search(keyword: String): List<MemoryEntry> {
    return memory.filter { it.key.contains(keyword, ignoreCase = true) ||
it.value.contains(keyword, ignoreCase = true) }
}
}

// ----- Phase 5: Online Wikipedia & Voice Search -----
package com.monster.study.online

import android.app.Activity
import android.content.Intent
import android.speech.RecognizerIntent
import android.speech.tts.TextToSpeech
import android.webkit.WebView
import android.webkit.WebViewClient
import android.widget.Toast
import kotlinx.coroutines.*
import org.json.JSONObject
import java.net.URL
import java.util.*

class OnlineAssistant(
    private val activity: Activity,
    private val tts: TextToSpeech,
    private val webView: WebView
) {

    companion object {
        const val VOICE_REQUEST_CODE = 1001
    }
}

```

```

init {
    webView.settings.javaScriptEnabled = true
    webView.webViewClient = WebViewClient()
}

// ----- Launch voice recognition -----
fun listenVoice() {
    val intent = Intent(RecognizerIntent.ACTION_RECOGNIZE_SPEECH).apply {
        putExtra(RecognizerIntent.EXTRA_LANGUAGE_MODEL,
RecognizerIntent.LANGUAGE_MODEL_FREE_FORM)
        putExtra(RecognizerIntent.EXTRA_LANGUAGE, Locale.getDefault())
        putExtra(RecognizerIntent.EXTRA_PROMPT, "Speak your query to Monster")
    }
    activity.startActivityForResult(intent, VOICE_REQUEST_CODE)
}

// ----- Handle voice result -----
fun handleVoiceResult(requestCode: Int, resultCode: Int, data: Intent?) {
    if (requestCode != VOICE_REQUEST_CODE || resultCode != Activity.RESULT_OK || data
== null) return
    val results = data.getStringArrayListExtra(RecognizerIntent.EXTRA_RESULTS)
    val query = results?.firstOrNull() ?: return
    searchWikipedia(query)
}

// ----- Search Wikipedia online -----
fun searchWikipedia(query: String) {
    CoroutineScope(Dispatchers.IO).launch {
        try {
            val apiUrl = "https://en.wikipedia.org/api/rest_v1/page/summary/${query.replace(" ",
" _")}"
            val jsonStr = URL(apiUrl).readText()
            val json = JSONObject(jsonStr)
            val title = json.optString("title", "")
            val extract = json.optString("extract", "No summary found.")

            withContext(Dispatchers.Main) {
                // Load summary into WebView
                webView.loadData("<h3>$title</h3><p>$extract</p>", "text/html", "UTF-8")
                // Speak summary
                tts.speak(extract, TextToSpeech.QUEUE_FLUSH, null, null)
            }
        } catch (e: Exception) {
            withContext(Dispatchers.Main) {

```

```

        Toast.makeText(activity, "Wikipedia search failed.", Toast.LENGTH_SHORT).show()
    }
}
}
}
}
// ----- Phase 6: Library & Classification System -----
package com.monster.study.library

import com.monster.study.ai.Document
import com.monster.study.ai.DocumentRepository

// ----- Library Document Model -----
data class LibraryDocument(
    val document: Document,
    val category: String,
    val tags: List<String> = emptyList()
)

// ----- Document Library -----
object DocumentLibrary {

    private val library = mutableListOf<LibraryDocument>()

    // ----- Initialize library from existing documents -----
    fun initialize() {
        library.clear()
        DocumentRepository.getAllDocs().forEach { doc ->
            library.add(LibraryDocument(doc, "Uncategorized"))
        }
    }

    // ----- Add a document with category and optional tags -----
    fun addDocument(doc: Document, category: String, tags: List<String> = emptyList()) {
        library.add(LibraryDocument(doc, category, tags))
        DocumentRepository.addDocument(doc) // keep offline AI updated
    }

    // ----- Get documents by category -----
    fun getByCategory(category: String): List<LibraryDocument> {
        return library.filter { it.category.equals(category, ignoreCase = true) }
    }

    // ----- Get documents by tag -----

```

```

fun getByTag(tag: String): List<LibraryDocument> {
    return library.filter { it.tags.any { t -> t.equals(tag, ignoreCase = true) } }
}

// ----- Search documents in the library by keyword -----
fun search(keyword: String): List<LibraryDocument> {
    return library.filter {
        it.document.text.contains(keyword, ignoreCase = true) ||
        it.document.title.contains(keyword, ignoreCase = true)
    }
}

// ----- List all categories -----
fun listCategories(): List<String> {
    return library.map { it.category }.distinct()
}

// ----- List all tags -----
fun listTags(): List<String> {
    return library.flatMap { it.tags }.distinct()
}

// ----- Remove a document -----
fun removeDocument(doc: Document) {
    library.removeAll { it.document.id == doc.id }
    // Optionally, remove from offline AI as well (rebuild)
    DocumentRepository.addDocument(doc) // or adjust AI accordingly
}
}

// ----- Phase 7: Unified WebView Dashboard -----
package com.monster.study.ui

import android.app.*
import android.content.*
import android.os.*
import android.speech.tts.TextToSpeech
import android.webkit.*
import android.widget.Toast
import androidx.appcompat.app.AppCompatActivity
import com.monster.study.ai.DocumentRepository
import com.monster.study.memory.MemoryManager
import com.monster.study.online.OnlineAssistant
import com.monster.study.revision.FlashcardEngine

```

```

class MonsterActivity : AppCompatActivity(), TextToSpeech.OnInitListener {

    private lateinit var webView: WebView
    private lateinit var tts: TextToSpeech
    private lateinit var onlineAssistant: OnlineAssistant

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        // ----- Initialize TTS -----
        tts = TextToSpeech(this, this)

        // ----- Load memory -----
        MemoryManager.load(this)

        // ----- Setup WebView -----
        webView = WebView(this).apply {
            settings.javaScriptEnabled = true
            webViewClient = WebViewClient()
            addJavascriptInterface(MonsterBridge(), "Monster")
            loadUrl("file:///android_asset/index.html")
        }

        setContentView(webView)

        // ----- Initialize Online Assistant -----
        onlineAssistant = OnlineAssistant(this, tts, webView)
    }

    override fun onInit(status: Int) {
        if (status == TextToSpeech.SUCCESS) tts.language = java.util.Locale.US
    }

    // ----- JavaScript Interface -----
    inner class MonsterBridge {

        @JavascriptInterface
        fun speak(text: String) {
            tts.speak(text, TextToSpeech.QUEUE_FLUSH, null, null)
        }

        @JavascriptInterface
        fun askAI(input: String): String {
            val ai = DocumentRepository.getAI()

```

```

        val response = ai?.ask(input) ?: "No offline AI available."
        speak(response)
        return response
    }

```

```

@JavascriptInterface
fun askAdaptiveAI(input: String): String {
    // Placeholder for adaptive responses; can integrate Phase 8 later
    val response = FlashcardEngine.recap()
    speak(response)
    return response
}

```

```

@JavascriptInterface
fun remember(key: String, value: String) {
    MemoryManager.remember(this@MonsterActivity, key, value)
    Toast.makeText(this@MonsterActivity, "Remembered: $key",
        Toast.LENGTH_SHORT).show()
}

```

```

@JavascriptInterface
fun recall(key: String): String {
    return MemoryManager.recall(key) ?: "No memory found for $key."
}

```

```

@JavascriptInterface
fun forget(key: String) {
    MemoryManager.forget(this@MonsterActivity, key)
    Toast.makeText(this@MonsterActivity, "Forgot: $key", Toast.LENGTH_SHORT).show()
}

```

```

@JavascriptInterface
fun flashcardNext(): String {
    val card = FlashcardEngine.getNextFlashcard()
    return card?.question ?: "No flashcards available."
}

```

```

@JavascriptInterface
fun markFlashcard(id: String, correct: Boolean) {
    FlashcardEngine.markFlashcard(id, correct)
}

```

```

@JavascriptInterface
fun searchWiki(query: String) {

```

```

        onlineAssistant.searchWikipedia(query)
    }

    @JavascriptInterface
    fun listenVoice() {
        onlineAssistant.listenVoice()
    }
}

// ----- Forward voice results -----
override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
    super.onActivityResult(requestCode, resultCode, data)
    onlineAssistant.handleVoiceResult(requestCode, resultCode, data)
}

override fun onDestroy() {
    super.onDestroy()
    tts.shutdown()
}
}
// ----- Phase 8: Adaptive Language Engine -----
package com.monster.study.ai

import java.util.*

// ----- Conversation Memory -----
data class ConversationEntry(
    val input: String,
    val response: String,
    val timestamp: Long = System.currentTimeMillis()
)

object ConversationMemory {
    private val history = mutableListOf<ConversationEntry>()
    private const val MAX_HISTORY = 50

    // Add conversation entry
    fun add(input: String, response: String) {
        history.add(ConversationEntry(input, response))
        if (history.size > MAX_HISTORY) history.removeAt(0)
    }

    // Retrieve recent context as a single string
    fun recentContext(): String {

```

```

        return history.joinToString(" ") { it.input + " " + it.response }
    }

    fun clear() {
        history.clear()
    }
}

// ----- Adaptive Responder -----
class AdaptiveResponder(private val documents: List<Document>) {

    // Respond to input considering conversation context
    fun respond(input: String): String {
        val context = ConversationMemory.recentContext()
        val queryTokens = TextProcessor.normalize(input)

        // Find the most relevant document
        val bestDoc = documents.maxByOrNull { doc ->
            val docTokens = TextProcessor.normalize(doc.text)
            queryTokens.count { it in docTokens }
        }

        var response = bestDoc?.text?.take(500) ?: "I don't have information on that yet."

        // If conversation is long, provide concise response
        response = if (context.length > 100) response.split(".").first() else response

        // Log conversation for adaptive learning
        ConversationMemory.add(input, response)

        return response
    }

    // Optional: prioritize keywords seen frequently in past queries
    fun keywordPrioritizedResponse(input: String): String {
        val keywords = ConversationMemory.recentContext()
            .split(" ")
            .groupBy { it.lowercase(Locale.US) }
            .eachCount()

        val docScores = documents.map { doc ->
            val tokens = TextProcessor.normalize(doc.text)
            tokens.count { it.lowercase(Locale.US) in keywords.keys } to doc
        }
    }
}

```

```

        val best = docScores.maxByOrNull { it.first }?.second
        val answer = best?.text?.take(500) ?: "No relevant information found."
        ConversationMemory.add(input, answer)
        return answer
    }

    // Clear memory if user wants to reset adaptive responses
    fun reset() {
        ConversationMemory.clear()
    }
}

// ----- Phase 9: Revision Analytics & Feedback -----
package com.monster.study.analytics

import com.monster.study.revision.FlashcardEngine
import com.monster.study.revision.Flashcard
import com.monster.study.memory.MemoryManager
import kotlin.math.roundToInt

// ----- Analytics Manager -----
object RevisionAnalytics {

    // ----- Flashcard statistics -----
    data class FlashcardStats(
        val total: Int,
        val reviewed: Int,
        val correct: Int,
        val accuracy: Int // percent
    )

    // Compute overall flashcard statistics
    fun computeFlashcardStats(): FlashcardStats {
        val flashcards = FlashcardEngine.getAllFlashcards()
        val total = flashcards.size
        val reviewed = flashcards.count { it.lastReviewed > 0 }
        val correct = flashcards.sumOf { it.confidence }
        val accuracy = if (reviewed > 0) ((correct.toDouble() / reviewed) * 100).roundToInt() else 0
        return FlashcardStats(total, reviewed, correct, accuracy)
    }

    // Identify weakest flashcards
    fun weakFlashcards(limit: Int = 10): List<Flashcard> {
        return FlashcardEngine.getAllFlashcards()
    }
}

```

```

        .sortedBy { it.confidence }
        .take(limit)
    }

    // Suggest next revision items (questions only)
    fun suggestNextRevision(limit: Int = 5): List<String> {
        val weak = weakFlashcards(limit)
        return weak.map { it.question }
    }

    // Memory usage summary
    fun memorySummary(): String {
        val memCount = MemoryManager.listAll().size
        return "You have $memCount memory entries stored."
    }

    // Generate simple performance feedback
    fun performanceFeedback(): String {
        val stats = computeFlashcardStats()
        val weakTopics = weakFlashcards(3).map { it.question }
        return "Revision Summary: ${stats.total} flashcards, ${stats.reviewed} reviewed, " +
            "accuracy: ${stats.accuracy}%. \nWeak topics: $weakTopics"
    }

    // Optional: full detailed report
    fun fullReport(): String {
        val stats = computeFlashcardStats()
        val weak = weakFlashcards()
        val weakDetails = weak.joinToString("\n") { "${it.question} (Confidence: ${it.confidence})" }
        return """
            === Monster Revision Report ===
            Total Flashcards: ${stats.total}
            Reviewed: ${stats.reviewed}
            Correctness (sum of confidence): ${stats.correct}
            Accuracy: ${stats.accuracy}%

            Weak Topics:
            $weakDetails

            Memory Summary: ${memorySummary()}
            """.trimIndent()
    }
}

# ----- Phase 10: Full Monster v2.4 GitHub-Ready Build -----

```

```

import os

# Root project folder
root = "MonsterStudy_v2_4"

# ----- Folder Structure -----
folders = [
    ".github/workflows",
    "app/src/main/java/com/monster/study/ai",
    "app/src/main/java/com/monster/study/revision",
    "app/src/main/java/com/monster/study/learning",
    "app/src/main/java/com/monster/study/memory",
    "app/src/main/java/com/monster/study/online",
    "app/src/main/java/com/monster/study/library",
    "app/src/main/java/com/monster/study/analytics",
    "app/src/main/java/com/monster/study/ui",
    "app/src/main/assets",
]

# Create folders
for f in folders:
    os.makedirs(os.path.join(root, f), exist_ok=True)

# ----- Kotlin Code Files -----
files = {
    # AI & Learning
    "app/src/main/java/com/monster/study/ai/Phase1_AI.kt": "<insert Phase1 code here>",
    "app/src/main/java/com/monster/study/revision/Phase2_Flashcards.kt": "<insert Phase2 code here>",
    "app/src/main/java/com/monster/study/learning/Phase3_LearningManager.kt": "<insert Phase3 code here>",
    "app/src/main/java/com/monster/study/memory/Phase4_MemoryManager.kt": "<insert Phase4 code here>",
    "app/src/main/java/com/monster/study/online/Phase5_OnlineAssistant.kt": "<insert Phase5 code here>",
    "app/src/main/java/com/monster/study/library/Phase6_Library.kt": "<insert Phase6 code here>",
    "app/src/main/java/com/monster/study/ui/Phase7_MonsterActivity.kt": "<insert Phase7 code here>",
    "app/src/main/java/com/monster/study/ai/Phase8_AdaptiveResponder.kt": "<insert Phase8 code here>",
    "app/src/main/java/com/monster/study/analytics/Phase9_RevisionAnalytics.kt": "<insert Phase9 code here>",
}

```

```

# Assets: HTML Dashboard
"app/src/main/assets/index.html": ""
<!DOCTYPE html>
<html>
<head><meta charset="UTF-8"><title>Monster v2.4 Dashboard</title></head>
<body>
<h2>Monster v2.4 Dashboard</h2>
<input id="query" placeholder="Ask Monster">
<button onclick="askAI()">Ask AI</button>
<button onclick="listen()">Voice</button>
<div id="result"></div>

<script>
function askAI(){
    const q=document.getElementById('query').value;
    const r=Monster.askAI(q);
    document.getElementById('result').innerText=r;
}
function listen(){ Monster.listenVoice(); }
</script>
</body>
</html>
""
,

```

```

# Android Manifest
"app/src/main/AndroidManifest.xml": ""
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
package="com.monster.study">
    <uses-permission android:name="android.permission.RECORD_AUDIO"/>
    <uses-permission android:name="android.permission.INTERNET"/>
    <application
        android:allowBackup="true"
        android:label="Monster"
        android:supportsRtl="true"
        android:theme="@style/Theme.AppCompat.Light.NoActionBar">
        <activity android:name=".ui.MonsterActivity" android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
    </application>
</manifest>
""
,

```

```

# Gradle
"build.gradle": """
buildscript {
    repositories { google(); mavenCentral() }
    dependencies { classpath "com.android.tools.build:gradle:8.2.0" }
}
allprojects { repositories { google(); mavenCentral() } }
""",

```

```

"settings.gradle": """
rootProject.name = "MonsterStudy_v2_4"
include ':app'
""",

```

```

"app/build.gradle": """
plugins { id 'com.android.application' }
android {
    namespace 'com.monster.study'
    compileSdk 34
    defaultConfig {
        applicationId "com.monster.study"
        minSdk 26
        targetSdk 34
        versionCode 4
        versionName "2.4"
    }
    buildTypes { release { minifyEnabled false } }
}
dependencies {
    implementation "androidx.appcompat:appcompat:1.7.0"
    implementation "androidx.recyclerview:recyclerview:1.3.0"
}
""",

```

```

# GitHub Workflow
".github/workflows/android-build.yml": """
name: Build Monster APK
on:
    push:
        branches: ["main"]
    workflow_dispatch:
jobs:
    build:

```

```

runs-on: ubuntu-latest
steps:
  - name: Checkout Repository
    uses: actions/checkout@v4
  - name: Set up JDK
    uses: actions/setup-java@v4
    with:
      distribution: temurin
      java-version: 17
  - name: Set up Android SDK
    uses: android-actions/setup-android@v3
  - name: Grant execute permission for Gradle
    run: chmod +x ./gradlew
  - name: Build Debug APK
    run: ./gradlew assembleDebug
  - name: Upload APK
    uses: actions/upload-artifact@v4
    with:
      name: Monster-v2.4-APK
      path: app/build/outputs/apk/debug/app-debug.apk
,,,,,
}

# ----- Write Files -----
for path, content in files.items():
    full_path = os.path.join(root, path)
    os.makedirs(os.path.dirname(full_path), exist_ok=True)
    with open(full_path, "w", encoding="utf-8") as f:
        f.write(content)

print(f"MonsterStudy v2.4 fully generated in folder '{root}'. Push to GitHub to build APK!")
// ----- Phase 11: Misleading Information Detector -----
package com.monster.study.analytics

import com.monster.study.library.DocumentLibrary
import com.monster.study.ai.Document
import com.monster.study.revision.FlashcardEngine
import kotlin.math.abs
import android.speech.tts.TextToSpeech

// ----- Misleading Detector -----
object MisleadingDetector {

    /**

```

```

* Analyze a document for misleading or conflicting information.
* - Compares against existing library documents
* - Checks flashcards for consistency
* @param document Document to check
* @param threshold similarity threshold (0..1) below which info may be misleading
* @return List of issues found
*/
fun analyze(document: Document, threshold: Double = 0.6): List<String> {
    val issues = mutableListOf<String>()
    val docTokens = TextProcessor.normalize(document.text).toSet()

    // Compare with library documents in the same category
    DocumentLibrary.getByCategory(document.category).forEach { libDoc ->
        val libTokens = TextProcessor.normalize(libDoc.document.text).toSet()
        val overlap = docTokens.intersect(libTokens).size.toDouble()
        val similarity = if (docTokens.isNotEmpty()) overlap / docTokens.size else 0.0

        if (similarity < threshold) {
            issues.add("Possible conflicting info in '${libDoc.document.title}' vs
'${document.title}'")
        }
    }

    // Optional: check flashcards
    FlashcardEngine.getAllFlashcards().forEach { card ->
        if (!document.text.contains(card.answer, ignoreCase = true)) {
            issues.add("Flashcard answer '${card.answer}' not found in document
'${document.title}'")
        }
    }

    return issues
}

/**
* Alert issues via console or optional TTS
* @return true if issues found
*/
fun alertIssues(document: Document, tts: TextToSpeech? = null): Boolean {
    val issues = analyze(document)
    if (issues.isNotEmpty()) {
        issues.forEach { issue ->
            tts?.speak(issue, TextToSpeech.QUEUE_ADD, null, null)
            println("Warning: $issue")
        }
    }
    return issues.isNotEmpty()
}

```

```

        }
        return true
    }
    return false
}
}
// ----- Phase 12: Confidence Scoring & Prioritized Revision -----
package com.monster.study.analytics

import com.monster.study.library.DocumentLibrary
import com.monster.study.revision.Flashcard
import com.monster.study.revision.FlashcardEngine
import com.monster.study.memory.MemoryManager

// ----- Confidence Manager -----
object ConfidenceManager {

    // Represents a scored document
    data class ScoredDocument(val title: String, val category: String, val confidence: Double)

    /**
     * Update flashcard confidence based on user performance and memory consistency
     */
    fun updateFlashcardConfidence() {
        FlashcardEngine.getAllFlashcards().forEach { card ->
            val correctFactor = card.confidence.toDouble() / 100
            val memoryFactor = if (MemoryManager.recall(card.question) != null) 0.1 else 0.0
            val newConfidence = (correctFactor + memoryFactor).coerceIn(0.0, 1.0)
            card.confidence = (newConfidence * 100).toInt()
        }
    }

    /**
     * Compute document confidence based on:
     * - Flashcard coverage (how many flashcards appear in document)
     * - MisleadingDetector checks
     */
    fun computeDocumentConfidence(): List<ScoredDocument> {
        val scoredDocs = mutableListOf<ScoredDocument>()
        DocumentLibrary.listCategories().forEach { category ->
            val docs = DocumentLibrary.getByCategory(category)
            docs.forEach { libDoc ->
                // Count flashcards covered in this document
                val flashcardsCovered = FlashcardEngine.getAllFlashcards().count { card ->

```

```

        libDoc.document.text.contains(card.answer, ignoreCase = true)
    }
    val coverageScore = if (FlashcardEngine.getAllFlashcards().isEmpty()) {
        flashcardsCovered.toDouble() / FlashcardEngine.getAllFlashcards().size
    } else 0.0

    // Penalize for misleading content
    val misleadingPenalty = if
(MisleadingDetector.analyze(libDoc.document).isEmpty()) 0.2 else 0.0
    val confidence = (coverageScore - misleadingPenalty).coerceIn(0.0, 1.0)

    scoredDocs.add(ScoredDocument(libDoc.document.title, category, confidence))
    }
    }
    return scoredDocs.sortedByDescending { it.confidence }
}

/**
 * Suggest top N documents for revision based on confidence
 */
fun topRevisionDocuments(n: Int = 5): List<ScoredDocument> {
    updateFlashcardConfidence()
    return computeDocumentConfidence().take(n)
}

/**
 * Suggest flashcards for revision, prioritized by lowest confidence
 */
fun prioritizedFlashcards(n: Int = 10): List<Flashcard> {
    updateFlashcardConfidence()
    return FlashcardEngine.getAllFlashcards()
        .sortedBy { it.confidence }
        .take(n)
}
}

// ----- Phase 13: Final Polish + Mind Maps -----
package com.monster.study.ui

```

```

import android.app.Activity
import android.os.Bundle
import android.webkit.WebView
import android.webkit.WebViewClient
import android.speech.tts.TextToSpeech
import android.widget.Toast

```

```

import androidx.appcompat.app.AppCompatActivity
import com.monster.study.library.DocumentLibrary
import com.monster.study.revision.FlashcardEngine
import com.monster.study.memory.MemoryManager
import com.monster.study.analytics.ConfidenceManager
import java.util.*

// ----- MonsterActivity with Mind Map -----
class MonsterActivity : AppCompatActivity(), TextToSpeech.OnInitListener {

    private lateinit var webView: WebView
    private lateinit var tts: TextToSpeech

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        // Initialize TTS
        tts = TextToSpeech(this, this)

        // Load Memory
        MemoryManager.load(this)

        // Setup WebView
        webView = WebView(this).apply {
            settings.javaScriptEnabled = true
            webViewClient = WebViewClient()
            addJavascriptInterface(MonsterBridge(), "Monster")
            loadUrl("file:///android_asset/dashboard.html")
        }
        setContentView(webView)
    }

    override fun onInit(status: Int) {
        if (status == TextToSpeech.SUCCESS) tts.language = Locale.US
    }

    inner class MonsterBridge {

        @android.webkit.JavascriptInterface
        fun speak(text: String) {
            tts.speak(text, TextToSpeech.QUEUE_FLUSH, null, null)
        }

        @android.webkit.JavascriptInterface

```

```

fun getMindMapData(): String {
    // Build a JSON representing topics, flashcards, and documents
    val nodes = mutableListOf<Map<String, Any>>()
    DocumentLibrary.listCategories().forEach { category ->
        nodes.add(mapOf("id" to category, "label" to category, "type" to "category"))
        val docs = DocumentLibrary.getByCategory(category)
        docs.forEach { doc ->
            nodes.add(
                mapOf(
                    "id" to doc.document.title,
                    "label" to doc.document.title,
                    "type" to "document",
                    "parent" to category
                )
            )
        }
    }
    FlashcardEngine.getAllFlashcards().forEach { card ->
        // Assuming each flashcard has a category property
        nodes.add(
            mapOf(
                "id" to card.question,
                "label" to card.question,
                "type" to "flashcard",
                "parent" to (card.category ?: "Uncategorized")
            )
        )
    }
    return nodes.toString()
}

```

```

@android.webkit.JavascriptInterface
fun remember(key: String, value: String) {
    MemoryManager.remember(this@MonsterActivity, key, value)
    Toast.makeText(this@MonsterActivity, "Remembered: $key",
        Toast.LENGTH_SHORT).show()
}

```

```

@android.webkit.JavascriptInterface
fun recall(key: String): String {
    return MemoryManager.recall(key) ?: "No memory found for $key."
}

```

```

@android.webkit.JavascriptInterface

```

```

        fun forget(key: String) {
            MemoryManager.forget(this@MonsterActivity, key)
            Toast.makeText(this@MonsterActivity, "Forgot: $key", Toast.LENGTH_SHORT).show()
        }
    }

    override fun onDestroy() {
        super.onDestroy()
        tts.shutdown()
    }
}

<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Monster v2.4 Dashboard</title>
<script src="https://cdnjs.cloudflare.com/ajax/libs/vis/4.21.0/vis.min.js"></script>
<link href="https://cdnjs.cloudflare.com/ajax/libs/vis/4.21.0/vis-network.min.css"
rel="stylesheet"/>
<style>
    #mindmap { width: 100%; height: 600px; border: 1px solid #ccc; }
</style>
</head>
<body>
<h2>Monster v2.4 Dashboard</h2>

<input id="query" placeholder="Ask Monster">
<button onclick="askAI()">Ask AI</button>
<button onclick="listen()">Voice Command</button>
<div id="result"></div>

<h3>Mind Map of Topics</h3>
<div id="mindmap"></div>

<script>
function askAI() {
    const q=document.getElementById('query').value;
    const r=Monster.askAI(q);
    document.getElementById('result').innerText=r;
}

function listen() { Monster.listenVoice(); }

// Load Mind Map

```

```

function loadMindMap() {
    const rawData = Monster.getMindMapData(); // JSON string
    const nodesArr = JSON.parse(rawData.replace(/'/g,"")); // Convert single quotes to double
    quotes
    const nodes = nodesArr.map(n => ({id:n.id, label:n.label}));
    const edges = nodesArr.filter(n=>n.parent).map(n=>({from:n.parent, to:n.id}));
    const container = document.getElementById('mindmap');
    const data = {nodes: new vis.DataSet(nodes), edges: new vis.DataSet(edges)};
    const options = {nodes:{shape:'box', color:'#FFA500'}, edges:{arrows:'to'}};
    new vis.Network(container, data, options);
}

```

```

window.onload = loadMindMap;

```

```

</script>

```

```

</body>

```

```

</html>

```

```

// ----- Phase 14: Smart Study Scheduler & Adaptive Session Manager

```

```

-----

```

```

package com.monster.study.scheduler

```

```

import com.monster.study.revision.Flashcard

```

```

import com.monster.study.revision.FlashcardEngine

```

```

import com.monster.study.library.DocumentLibrary

```

```

import kotlin.random.Random

```

```

import java.util.*

```

```

// ----- Study Session Model -----

```

```

data class StudySession(

```

```
    val sessionId: String = UUID.randomUUID().toString(),

```

```
    val startTime: Long = System.currentTimeMillis(),

```

```
    var endTime: Long? = null,

```

```
    var active: Boolean = true,

```

```
    val reviewedDocuments: MutableSet<String> = mutableSetOf(),

```

```
    val reviewedFlashcards: MutableSet<String> = mutableSetOf()

```

```

)

```

```

// ----- Session Manager -----

```

```

object StudySessionManager {

```

```
    private val sessions = mutableListOf<StudySession>()

```

```
    private var currentSession: StudySession? = null

```

```

// Start a new study session

```

```

fun startSession(): StudySession {

```

```

        val session = StudySession()
        sessions.add(session)
        currentSession = session
        return session
    }

    // End current session
    fun endSession() {
        currentSession?.apply {
            endTime = System.currentTimeMillis()
            active = false
        }
        currentSession = null
    }

    fun getCurrentSession(): StudySession? = currentSession
    fun getAllSessions(): List<StudySession> = sessions.toList()

    // ----- Adaptive Study Scheduler -----

    // Select next document to review based on:
    // 1. Low confidence (from ConfidenceManager)
    // 2. Balanced course coverage
    fun nextDocument(): String? {
        val allDocs = DocumentLibrary.listCategories().flatMap { category ->
            DocumentLibrary.getByCategory(category).map { it.document.title to category }
        }.toMutableList()

        // Remove already reviewed in current session
        val reviewed = currentSession?.reviewedDocuments ?: emptySet()
        allDocs.removeAll { reviewed.contains(it.first) }

        if (allDocs.isEmpty()) return null

        // Pick document with lowest confidence first (simulate balanced selection)
        val shuffled = allDocs.shuffled(Random(System.currentTimeMillis()))
        val nextDoc = shuffled.first()
        currentSession?.reviewedDocuments?.add(nextDoc.first)
        return nextDoc.first
    }

    // Select next flashcard to review adaptively
    fun nextFlashcard(): Flashcard? {
        val allCards = FlashcardEngine.getAllFlashcards().toMutableList()

```

```

    val reviewed = currentSession?.reviewedFlashcards ?: emptySet()
    allCards.removeAll { reviewed.contains(it.id) }
    if (allCards.isEmpty()) return null

    // Prioritize low confidence
    val sorted = allCards.sortedBy { it.confidence }
    val nextCard = sorted.first()
    currentSession?.reviewedFlashcards?.add(nextCard.id)
    return nextCard
}

// Generate session summary
fun sessionSummary(): String {
    val session = currentSession ?: return "No active session."
    val docCount = session.reviewedDocuments.size
    val cardCount = session.reviewedFlashcards.size
    val duration = if (session.endTime != null) (session.endTime!! - session.startTime) / 1000
else 0
    return "Session ID: ${session.sessionId}\nDocuments reviewed: $docCount\nFlashcards
reviewed: $cardCount\nDuration (s): $duration"
}

// ----- Time-Bomb / Exam Mode -----
fun timeBombMode(durationSeconds: Int, onTick: (Int) -> Unit, onComplete: () -> Unit) {
    Thread {
        var remaining = durationSeconds
        while (remaining > 0 && currentSession?.active == true) {
            Thread.sleep(1000)
            remaining--
            onTick(remaining)
        }
        onComplete()
    }.start()
}

// ----- Controlled Voice Trigger -----
fun controlledVoiceCommand(input: String, callback: (String) -> Unit) {
    if (!input.lowercase().startsWith("monster")) {
        callback("Command ignored. Say 'Monster' first to activate voice control.")
    } else {
        val command = input.removePrefix("Monster").trim()
        // You can route commands here (e.g., ask AI, recall memory)
        callback("Command received: $command")
    }
}

```

```

    }
}
// ----- Phase 15: Multilingual Voice Engine & Speech-to-Speech -----
package com.monster.study.voice

import android.app.Activity
import android.content.Intent
import android.os.Build
import android.speech.RecognizerIntent
import android.speech.SpeechRecognizer
import android.speech.tts.TextToSpeech
import android.widget.Toast
import java.util.*
import kotlin.collections.ArrayList
import kotlinx.coroutines.*

// ----- Voice Session Modes -----
enum class VoiceMode { TEXT_TO_SPEECH, SPEECH_TO_TEXT, SPEECH_TO_SPEECH }

// ----- Voice Engine -----
class MonsterVoiceEngine(private val activity: Activity) : TextToSpeech.OnInitListener {

    private lateinit var tts: TextToSpeech
    private var recognizer: SpeechRecognizer? = null
    private var currentLanguage: Locale = Locale.US
    private var mode: VoiceMode = VoiceMode.TEXT_TO_SPEECH
    private var onResultCallback: ((String) -> Unit)? = null

    companion object {
        const val VOICE_REQUEST_CODE = 2001
    }

    init {
        tts = TextToSpeech(activity, this)
    }

    override fun onInit(status: Int) {
        if (status == TextToSpeech.SUCCESS) {
            tts.language = currentLanguage
            tts.setSpeechRate(1.0f)
            tts.setPitch(1.0f)
        } else {
            Toast.makeText(activity, "TTS Initialization failed.", Toast.LENGTH_SHORT).show()
        }
    }
}

```

```

}

// ----- Language Settings -----
fun setLanguage(locale: Locale) {
    currentLanguage = locale
    tts.language = locale
}

// ----- Text-to-Speech -----
fun speak(text: String, queue: Boolean = false) {
    val queueMode = if (queue) TextToSpeech.QUEUE_ADD else
TextToSpeech.QUEUE_FLUSH
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.LOLLIPOP) {
        tts.speak(text, queueMode, null, UUID.randomUUID().toString())
    } else {
        @Suppress("DEPRECATION")
        tts.speak(text, queueMode, null)
    }
}

// ----- Speech-to-Text -----
fun listenVoice(onResult: (String) -> Unit) {
    onResultCallback = onResult
    val intent = Intent(RecognizerIntent.ACTION_RECOGNIZE_SPEECH).apply {
        putExtra(RecognizerIntent.EXTRA_LANGUAGE_MODEL,
RecognizerIntent.LANGUAGE_MODEL_FREE_FORM)
        putExtra(RecognizerIntent.EXTRA_LANGUAGE, currentLanguage)
        putExtra(RecognizerIntent.EXTRA_PROMPT, "Speak to Monster")
    }
    activity.startActivityForResult(intent, VOICE_REQUEST_CODE)
}

fun handleVoiceResult(requestCode: Int, resultCode: Int, data: Intent?) {
    if (requestCode != VOICE_REQUEST_CODE || resultCode != Activity.RESULT_OK || data
== null) return
    val results: ArrayList<String>? =
data.getStringArrayListExtra(RecognizerIntent.EXTRA_RESULTS)
    val spokenText = results?.firstOrNull() ?: ""
    onResultCallback?.invoke(spokenText)
}

// ----- Speech-to-Speech -----
fun speakBack(spokenInput: String, callback: (String) -> String) {
    // Process input via callback (AI, flashcards, or session manager)

```

```

        val response = callback(spokenInput)
        speak(response)
    }

    // ----- Optional Captions -----
    fun speakWithCaptions(spokenInput: String, callback: (String) -> String, onCaption: (String) ->
Unit) {
        val response = callback(spokenInput)
        onCaption(response)
        speak(response)
    }

    // ----- Background Study Mode -----
    fun backgroundStudy(documents: List<String>, delayPerDoc: Long = 30000L) {
        CoroutineScope(Dispatchers.Default).launch {
            for (doc in documents) {
                speak(doc)
                delay(delayPerDoc)
            }
        }
    }

    // ----- Voice Modulation / Variety -----
    fun modulateVoice(pitch: Float = 1.0f, rate: Float = 1.0f) {
        tts.pitch = pitch
        tts.setSpeechRate(rate)
    }

    fun shutdown() {
        tts.stop()
        tts.shutdown()
        recognizer?.destroy()
    }
}

// ----- Phase 16: Unified Session + Voice Adaptive Engine -----
package com.monster.study.session

import android.app.Activity
import android.content.Intent
import android.os.Bundle
import android.speech.tts.TextToSpeech
import android.webkit.WebView
import android.webkit.WebViewClient
import android.widget.Toast

```

```

import androidx.appcompat.app.AppCompatActivity
import com.monster.study.ai.*
import com.monster.study.memory.MemoryManager
import com.monster.study.revision.FlashcardEngine
import com.monster.study.voice.MonsterVoiceEngine
import kotlinx.coroutines.*
import java.util.*

class MonsterSessionActivity : AppCompatActivity(), TextToSpeech.OnInitListener {

    private lateinit var webView: WebView
    private lateinit var voiceEngine: MonsterVoiceEngine
    private lateinit var tts: TextToSpeech

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        // Initialize TTS
        tts = TextToSpeech(this, this)

        // Load memory
        MemoryManager.load(this)

        // Initialize voice engine
        voiceEngine = MonsterVoiceEngine(this)

        // Setup WebView Dashboard
        webView = WebView(this).apply {
            settings.javaScriptEnabled = true
            webViewClient = WebViewClient()
            addJavascriptInterface(SessionBridge(), "Monster")
            loadUrl("file:///android_asset/session_dashboard.html")
        }
        setContentView(webView)
    }

    override fun onInit(status: Int) {
        if (status == TextToSpeech.SUCCESS) tts.language = Locale.US
    }

    // ----- JavaScript Interface for UI -----
    inner class SessionBridge {

        @android.webkit.JavascriptInterface

```

```
fun speak(text: String) { voiceEngine.speak(text) }
```

```
@android.webkit.JavascriptInterface
fun askAI(input: String): String {
    val ai = DocumentRepository.getAI()
    val response = ai?.ask(input) ?: "No offline AI available."
    voiceEngine.speak(response)
    return response
}
```

```
@android.webkit.JavascriptInterface
fun askAdaptiveAI(input: String): String {
    val responder = AdaptiveResponder(DocumentRepository.getAllDocs())
    val response = responder.respond(input)
    voiceEngine.speak(response)
    return response
}
```

```
@android.webkit.JavascriptInterface
fun flashcardNext(): String {
    val card = FlashcardEngine.getNextFlashcard()
    return card?.question ?: "No flashcards available."
}
```

```
@android.webkit.JavascriptInterface
fun markFlashcard(id: String, correct: Boolean) {
    FlashcardEngine.markFlashcard(id, correct)
}
```

```
@android.webkit.JavascriptInterface
fun startVoiceSession() {
    voiceEngine.listenVoice { spokenText ->
        // Adaptive response
        val response =
AdaptiveResponder(DocumentRepository.getAllDocs()).respond(spokenText)
        voiceEngine.speak(response)
    }
}
```

```
@android.webkit.JavascriptInterface
fun startBackgroundStudy(delayMs: Long) {
    val docsText = DocumentRepository.getAllDocs().map { it.text.take(500) }
    voiceEngine.backgroundStudy(docsText, delayMs)
}
```

```

    @android.webkit.JavascriptInterface
    fun remember(key: String, value: String) {
        MemoryManager.remember(this@MonsterSessionActivity, key, value)
        Toast.makeText(this@MonsterSessionActivity, "Remembered: $key",
            Toast.LENGTH_SHORT).show()
    }

```

```

    @android.webkit.JavascriptInterface
    fun recall(key: String): String {
        return MemoryManager.recall(key) ?: "No memory found for $key."
    }

```

```

    @android.webkit.JavascriptInterface
    fun forget(key: String) {
        MemoryManager.forget(this@MonsterSessionActivity, key)
        Toast.makeText(this@MonsterSessionActivity, "Forgot: $key",
            Toast.LENGTH_SHORT).show()
    }

```

```

    @android.webkit.JavascriptInterface
    fun modulateVoice(pitch: Float, rate: Float) {
        voiceEngine.modulateVoice(pitch, rate)
    }
}

```

```

override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
    super.onActivityResult(requestCode, resultCode, data)
    voiceEngine.handleVoiceResult(requestCode, resultCode, data)
}

```

```

override fun onDestroy() {
    super.onDestroy()
    tts.shutdown()
    voiceEngine.shutdown()
}
}

```