ChatGPT

Android PRD 文档与文件结构

本文件包含 Android 随身翻译 App 的产品需求文档以及建议的项目目录结构，方便阅览和归档。

PRD 文档

以下内容摘自《产品需求文档 – Android 原生随身翻译 App》：

1. 文档版本

版本	日期	作者	备注
1.0	2025-10-16	产品团队	初始版本

2. 背景与目标

本产品是一款随身语音翻译应用，基于原生 Android 开发。用户在旅行、商务或学习场景中，可通过应用快速实现中法语实时翻译：说中文时即时播报法语音频；听取法语时即刻展示中文文本。目标是实现 **< 600 ms** 的端到端翻译延迟，并在复杂网络及噪音环境下保持稳定。

2.1 业务背景

- 全球化需求促使随身翻译工具普及，但现有 App 通常在延时、隐私或多语言体验方面存在不足。
- OpenAI 的 Realtime API 和 GPT-4o/GPT-4.1 系列模型已经成熟，能够提供高精度 ASR、翻译和 TTS 能力。指出，原生开发在性能和用户体验方面仍是金标准，且可以直接访问底层硬件。

2.2 目标和成功标准

- 提供稳定、低延时的实时翻译体验：从用户发声到听到/看到译文的总时延控制在 600–900 ms；反馈流畅，不卡顿。
- 确保识别准确率：常用场景（日常对话、旅行、商务）的识别准确率 ⩾ 95%。
- 强调隐私与安全：默认关闭数据上传训练；使用加密连接和匿名会话。
- 兼顾开发迭代与测试：通过合理架构分层，支持未来新增语言、模型和功能。

3. 范围

3.1 包含

- Android 客户端（Kotlin）：语音采集、实时翻译、音频播放/文本显示、设置页面。
- 后端代理服务：提供 API 密钥管理、会话签名、访问控制。
- 接入 OpenAI Realtime API：使用模型如 GPT-4o-mini-transcribe、GPT-4.1/4o-mini 翻译、GPT-4o-mini TTS。
- 测试系统：用于性能、稳定性及兼容性测试。

3.2 不包含

- 订阅与支付系统（未来版本扩展）。

- 广告投放与商业化。
- 社交分享功能。

4. 用户故事与场景

编号	用户故事	受众
US1	作为一名旅行者，我希望对着手机说中文，立即听到法语播报，以便与当地人沟通。	旅行者
US2	作为一名留学生，我希望应用识别对方说的法语并显示中文文本，方便快速理解。	学生
US3	作为商务人士，我希望在会议期间使用翻译功能，同时能够选择不同翻译模型以追求速度或准确度。	商务人士
US4	作为重度用户，我需要离线兜底功能，当网络不稳定时仍可获得基本翻译文本。	各类用户

5. 功能需求

5.1 核心翻译流程

1. **麦克风采集**：
2. 使用 Android `AudioRecord` 采集麦克风输入，采样率 16 kHz 或 48 kHz 单声道，根据设备支持动态调整。
3. 启动前检查并请求录音权限；提示用户授权。
4. **实时流传输**：
5. 使用 WebRTC 原生库建立音频双向通道，将麦克风音频流发送到后端代理，再转发至 OpenAI Realtime API。
6. 为中文 → 法语链路启用语音合成返回；法语 → 中文链路仅接收文本。
7. **模型处理**：
8. ASR：默认使用 `gpt-4o-mini-transcribe`；当识别置信度低或口音复杂时自动切换 `gpt-4o-transcribe`。
9. 翻译：使用 `gpt-4o-mini`；用户可在设置中选择 `gpt-4.1` 以提高准确性。
10. TTS：使用 `gpt-4o-mini-tts` 生成功实时法语音频。
11. **输出呈现**：
12. A 链路（中文→法语）：播放翻译后语音并在 UI 中显示字幕；提供暂停/停止按钮。
13. B 链路（法语→中文）：在 UI 中即时显示翻译文本；如有连续句子，分页滚动显示。

5.2 UI 结构

- **首页/翻译界面**：主屏幕包含麦克风按钮、语言方向指示、音量/音频路由控制。发起翻译后显示实时字幕和音频播放进度。
- **设置页面**：

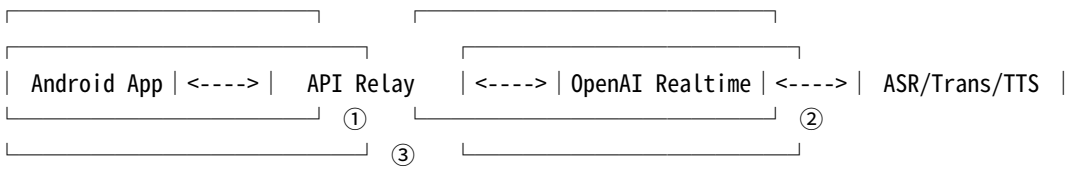
- 选择说话语言与目标语言（默认为中文↔法语）。
- 模型选择：ASR、翻译和 TTS 各自可选择默认和高精度版本 [1]。
- 网络模式：启用低延时模式或离线兜底（本地 Whisper v3）。
- 隐私选项：开关“数据用于训练”标签；显示隐私政策链接。
- **历史记录（可选）**：记录最近的对话文本和时间戳，便于回顾。

5.3 非功能需求

- **性能**：端到端延迟 < 900 ms（目标 < 600 ms）。
- **稳定性**：连续使用 60 分钟不出现内存泄漏或线程阻塞。
- **安全与隐私**：所有会话经 HTTPS/WebRTC 加密；后端代理控制 API Key；用户数据默认不用于模型训练。
- **扩展性**：支持添加新语言对；模块化设计支持插入新的模型。

6. 技术架构

系统架构图：



① 端侧 WebRTC：AudioRecord 捕获音频数据并通过 WebRTC 发送；接收流式法语音频和中文文本。
② API Relay：签发临时凭证、转发流并实施速率限制；可与 TURN/STUN 结合解决 NAT 问题。
③ OpenAI：执行实时语音识别、翻译和语音合成。

6.2 Android 客户端模块划分

模块	描述
UI 层	使用 Jetpack Compose 构建界面，提供麦克风按钮、状态栏、字幕显示、设置界面。
业务逻辑层	管理对话状态、控制翻译流程、维护当前语言和模型配置。
音频与通讯层	集成 WebRTC Native 库，管理音频采集(`AudioRecord`)、音频播放(`AudioTrack`)、网络流(`RTCPeerConnection`) 与编码参数。
后端通信层	与 API Relay 交互，获取实时会话凭证；使用 Retrofit + WebSocket 实现控制信息的传输；管理错误重试、心跳包。
本地模型兜底	集成 Whisper v3 (可选) 用于离线识别。
数据存储层	使用 Room 或 DataStore 保存历史记录和用户设置。
安全模块	集成 Android Keystore 存储本地凭证；确保日志不记录敏感音频或文本内容。

6.3 原生模块调优

- **音频采集**：设置采样率和 buffer size；使用低延时模式；避免在主线程进行音频 IO。

- **网络优化**：使用 `NetworkCallback` 监听网络变动；动态调整 WebRTC 音频比特率；在弱网环境下提高缓冲。
- **后台服务**：当用户切换应用时保持翻译会话在前台服务中运行。
- **资源管理**：翻译会话结束后释放 `RTCPeerConnection`、模型实例等资源，防止内存泄漏。

6.4 后端代理接口示例

接口	方法	说明
`/session/start`	POST	发起新会话，返回临时会话 ID、WebRTC 凭证和模型默认配置
`/session/update`	PATCH	更新模型选择、语言对等配置
`/session/stop`	POST	结束会话，释放资源
`/session/metrics`	POST	上传客户端性能指标（延迟、错误码）用于分析

7. 测试计划

概括单元测试、集成测试、用户测试，涵盖低延时测试、噪音鲁棒性、异常恢复和并发压力测试。

8. 里程碑计划

阶段	时间	目标
需求分析与架构设计	2025-10-16 – 2025-10-20	完成 PRD、系统设计、技术选型
原型开发	2025-10-21 – 2025-11-15	实现核心翻译链路、基本 UI、设置页面
内测与调优	2025-11-16 – 2025-12-15	性能优化、模型选择策略完善、离线兜底集成
公测与发布	2026-01-01 前	上架 Google Play，收集用户反馈

9. 风险与假设

列出了模型调用成本、网络依赖、硬件差异和隐私法规等风险及假设。

Android 项目文件结构

以下为建议的 Android 原生项目目录结构，用于随身翻译 App。该结构遵循模块化和分层原则，便于维护、扩展和测试。

```
translator-android-app/
├── app/                          # 主应用模块
│   ├── build.gradle              # 模块配置文件
│   └── src/
│       └── main/
│           ├── AndroidManifest.xml
│           ├── java/com/example/translatorapp/
│           │   ├── presentation/         # 界面层 (Jetpack Compose)
│           │   │   ├── home/             # 首页，含麦克风按钮、字幕展示
│           │   │   ├── settings/         # 设置页面，模型选择、语言配置
```

```
│            │    │    ├── history/          # 历史记录页面（可选）
│            │    │    └── components/       # 复用的 UI 组件（按钮、卡片等）
│            │    ├── domain/               # 领域层
│            │    │    ├── model/            # 数据模型，如 TranslationState、UserSettings
│            │    │    ├── usecase/          # 用例层，封装业务逻辑
│            │    │    └── repository/       # 抽象数据访问接口
│            │    ├── data/                 # 数据层
│            │    │    ├── repository/       # 实现 Repository，调度网络和本地数据
│            │    │    ├── datasource/       # 远程（APIRelay）与本地（Room）数据源
│            │    │    └── model/            # 数据层模型映射
│            │    ├── network/              # 网络通信，Retrofit/WebSocket 封装
│            │    ├── audio/                # 音频采集与播放模块（AudioRecord、
AudioTrack、WebRTC）
│            │    ├── webrtc/               # WebRTC 库封装，管理 RTCPeerConnection
│            │    ├── localmodel/           # 本地 Whisper 模型集成（可选）
│            │    ├── util/                 # 工具类（网络监测、权限管理等）
│            │    └── di/                   # 依赖注入（Hilt/Koin）配置
│            └── res/
│                ├── values/               # 字符串、主题、样式
│                ├── drawable/             # 图片资源
│                └── raw/                  # 离线模型或其他原始文件
├── build.gradle
├── settings.gradle
└── gradle/                      # Gradle 脚本和配置
```

文件结构说明

- **presentation** 层负责 UI 和用户交互，采用 Jetpack Compose，按照页面划分子目录。
- **domain** 层放置纯业务逻辑，不依赖安卓框架，便于测试和复用。
- **data** 层处理所有数据来源，包括远程 API、WebRTC 流、本地数据库等。
- **audio** 与 **webrtc** 模块封装音频采集、播放和 WebRTC 连接的细节，实现低延时传输。
- **localmodel** 用于集成本地离线模型，如 Whisper v3，用作网络不可用时的兜底方案。
- 使用 **di** 目录集中管理依赖注入配置，方便测试和模块替换。

本文件整合了 Android PRD 文档及项目文件结构示例，旨在为开发团队提供完整的需求与结构参考。如需查看更详细的 iOS 版本或其他资料，请参阅相关文档。

---

1 Flutter vs React Native: Complete 2025 Framework Comparison Guide | Blog
https://www.thedroidsonroids.com/blog/flutter-vs-react-native-comparison