

離線到雲端的智慧候診系統： AI 聲音合成與多終端同步實戰

Offline-to-Cloud Smart Queue System: AI Speech Synthesis & Multi-device Sync in Practice

● 講者：許家誠 Stephen Hsu

🏢 Aldrich International Entertainment Co., Ltd.

🌐 github.com/jiachengx

✉️ chiacheng.hsu@owasp.org



講者介紹 - 許家誠 Stephen Hsu

Aldrich International Entertainment Co., Ltd. | YITAI INTERNATIONAL TECH CO., LTD

技術專業與系統整合

- ⌚ 軟硬韌體系統整合 | 資安風險控管
- ☰ Technical PM | DQA/SQA | 專案管理
- </> 全端開發：React.js | Python | Golang
- ☷ 開源ERP系統（Odoo）客製化與導入
- ⚙️ 作業流程優化 | 自動化應用推動

AI應用與社會參與

- ⊕ 智能販賣機研發 | 金流/發票整合
- ▢ AI創作教學 | 實務場域應用推廣
- 〽 數位行銷策略 | EFP | UTM分析
- 人群 媒體整合行銷 | 品牌互動設計
- 🌐 NPO國際志工服務 | 跨文化交流



個人資料

專案背景：菲律賓眼科義診中心

醫護痛點

- 醫生診療中斷：頻繁被打斷詢問看診順序，影響診療效率
- 護理師負擔：難以即時掌握18個診間狀態，人力調配困難
- ⌚ 志工混亂：多為非專業志工，缺乏統一流程與培訓
- ⌚ 患者視障：眼疾患者無法清楚看到叫號資訊，易錯過叫號

系統需求

- ▣ 設備：42台裝置（18手機+18廣告屏+3TV+2前台+1管理）
- 🔊 語音：離線優先的英文語音叫號系統
- 👤 介面：視障友善UI（超大字體、高對比）
- ⟳ 環境適應：易停電環境（無UPS）下的快速復原機制

核心挑戰

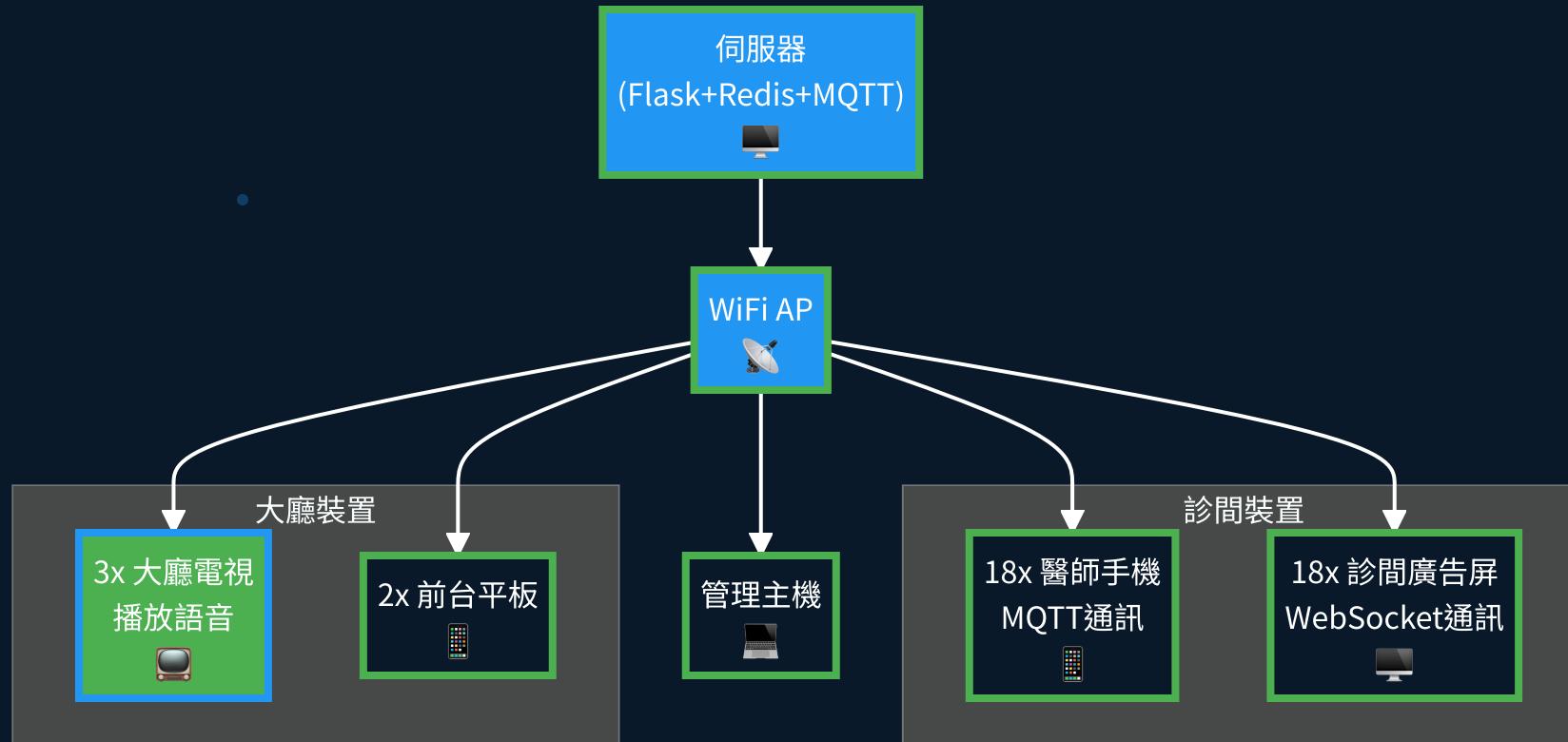
醫護工作流程痛點

- 醫生無法專注診療：頻繁被打斷詢問看診順序，診療效率降低
- 護理師工作負荷重：難以同時管理18診間狀態，分身乏術
- 呼叫號效率低：聲音難以穿透嘈雜環境，容易出錯遺漏
- 志工混亂管理：流動性高、缺乏培訓，無標準化流程

環境與技術挑戰

- ⚡ 易停電、無UPS：系統需10-15分鐘內快速恢復
- 👁 患者眼疾：視障病患無法看清標示，需音聲和超大字體引導
- 📱 設備限制：Android版本不一；需Fully Kiosk整合
- ⟳ 並發管理：多人同時叫號，需防抖與排隊機制

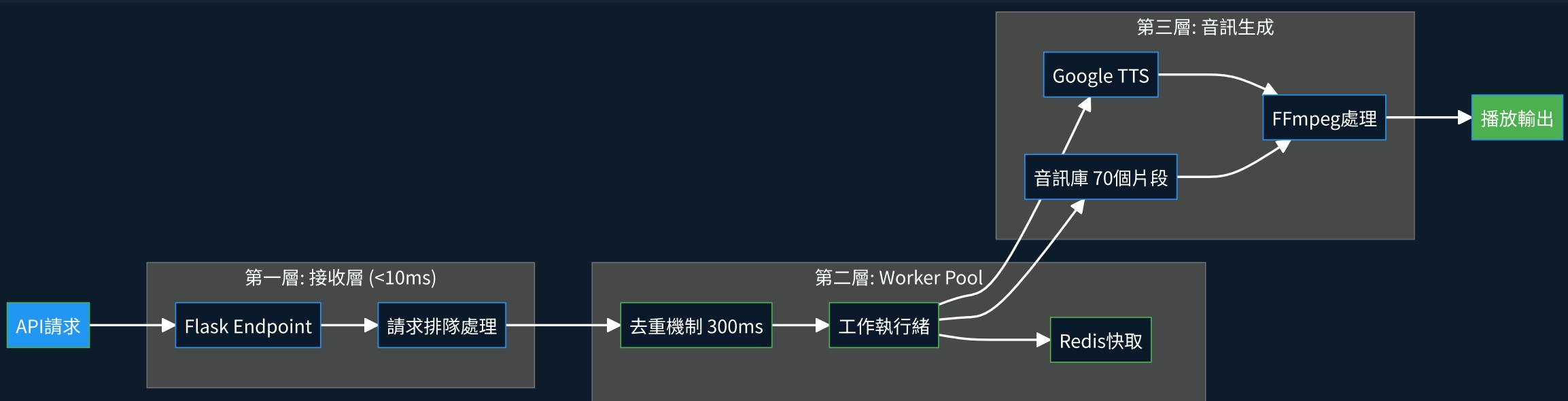
系統全景：42台設備生態拓撲



硬體配置表 (18診間 + 大廳)

裝置類型	數量	角色說明
Android 手機	18	醫師端按鍵/叫號 (MQTT)
廣告屏	18	診間外顯示 (不播音) (WS)
大廳電視	3	大廳語音播放 (WS)
前台平板	2	掛號與控管 (WS)
管理主機	1	服務與監控 (Flask+Redis+MQTT)
● 設備維護 ：所有Android設備使用Fully Kiosk Browser鎖定，遠端管理 ● 連線方式 ：本地WiFi互連，無需外網		
總計	42	完整生態系統

三層非同步架構



高效設計：接收層<10ms回應，解耦後續處理 | 防抖機制：300ms窗口內同號多次叫號僅執行一次 | 效能表現：端對端<300ms，最大支援500 ops/sec

雙協定策略：MQTT vs WebSocket

面向	MQTT (手機)	WebSocket (瀏覽器)
省電與穩定	🔋 QoS 1 確保送達，低功耗設計	⚡ 持續連線，相對耗電
拓撲結構	📢 Pub/Sub 多對多模型	↔ 點對點雙向連線
傳輸保證	🛡️ QoS 0-2 三級保證機制	🔁 TCP層保證，應用層需自實作
訊息路由	🔍 Topic過濾，精準訂閱	👥 Room分組廣播
使用場景	📱 18醫師手機 (低功耗優先)	💻 Dashboard/廣告屏/TV (即時優先)
系統結論	✓ 適合不穩網路、電池裝置	✓ 適合瀏覽器原生、即時互動

💡 **雙協定優勢**：結合兩者優點，Android手機透過MQTT省電，瀏覽器設備透過WebSocket即時響應

叫號完整流程 (含 Ding-Dong)

非同步架構：接收層 → Worker Pool → 音訊播放 | “P” 播放優先級：大廳TV → 診間外看板 → 手機通知



資料持久化：Redis + JSON

即時態：Redis

- ⌚ RDB快照：每15分鐘或1萬次變更自動備份
- ⚡ 記憶體優先：全部即時資料存於記憶體中
- ⚠ 風險控制：最多損失15分鐘資料（可接受）
- ⟳ 快速恢復：啟動自動載入最新RDB快照

永久態：JSON + 日誌

- 💾 配置檔：採用JSON格式，原子性寫入防損壞
- 〽 統計輸出：每5分鐘將統計數據寫入JSON
- 📅 事件日誌：關鍵事件即時append寫入log
- 🚮 跨日清空：日期比對自動歸檔前日資料

高並發處理：Worker Pool + 防抖

```
class WorkerPool:
    def __init__(self):
        self.queue = Queue()
        self.debounce_window = 0.3 # 300ms 去重窗口
        self.pending_calls = {}
        self.cache = LRUCache(100)

        # 初始化4個工作執行緒
        for _ in range(4):
            worker =
                Thread(target=self._worker_loop)
            worker.daemon = True
            worker.start()

    def process_call(self, room_id, patient_id):
        key = f"{room_id}:{patient_id}"
        now = time.time()

        if key in self.pending_calls:
            last_time =
                self.pending_calls[key]
            if now - last_time <
                self.debounce_window:
                # 300ms內重複點擊，忽略
                self.pending_calls[key] =
                    now
                return False

        # 加入隊列並記錄時間戳
        self.pending_calls[key] = now
        self.queue.put((key, now))
        return True

    def _worker_loop(self):
        while True:
            key, timestamp =
                self.queue.get()

            # 檢查快取
            if key in self.cache:
                audio =
                    self.cache.get(key)
            else:
                audio =
                    self._generate_audio(key)
                    self.cache.put(key,
                                   audio)

            self._play_audio(audio)
```

關鍵設計重點

☒ 去重窗口：300ms

醫師可能短時間內連點同一號碼，
300ms窗口可有效濾掉誤操作

〓 多工處理池

4個背景執行緒並行處理，支援500
ops/sec

⚡ 最新請求優先

時間戳記追蹤，確保只處理最新事件

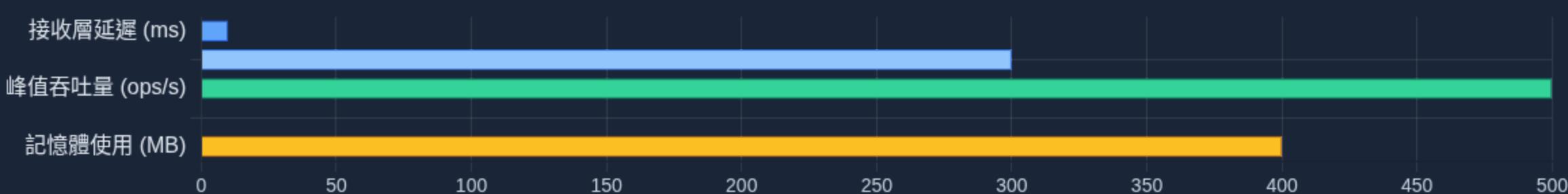
⌚ 音訊快取策略

熱門號碼預先合成，大幅降低處理延遲

🕒 效能表現

- 接收處理： $<10\text{ms}$ (幾乎無感)
- 佇列處理：平均 $50\text{-}100\text{ms}$
- 音訊合成： $\sim 200\text{ms}$
- 端對端延遲： $<300\text{ms}$

效能數據圖表



接收層延遲

10ms

API請求處理時間

端對端延遲

300ms

請求到播放總延遲

峰值吞吐量

500 ops/s

系統最大處理能力

實際吞吐量

0.32 ops/s

平均請求頻率

記憶體使用

400MB

含預生成音訊檔案

系統效能總結

Licensed to MOPCON 2025

停電快速恢復策略

無UPS環境的挑戰

- ⚡ 菲律賓當地環境：停電頻繁、無UPS保護
- ⌚ Redis RDB快照：每15分鐘或1萬次變更自動儲存
- ⌚ 開機恢復時間：10-15分鐘內完全功能恢復
- ❗ 可接受損失：最多15分鐘資料（已被驗證可行）

快速恢復機制

- ▶ 啟動順序：Redis → Mosquitto MQTT → Flask App
- ✓ 自檢與回放：斷點檢測、叫號記錄回放
- 🔊 音訊庫：本地持久化儲存，避免重新下載生成
- 🔧 服務註冊：Windows服務自動啟動，無人介入

每日自動重置

核心邏輯

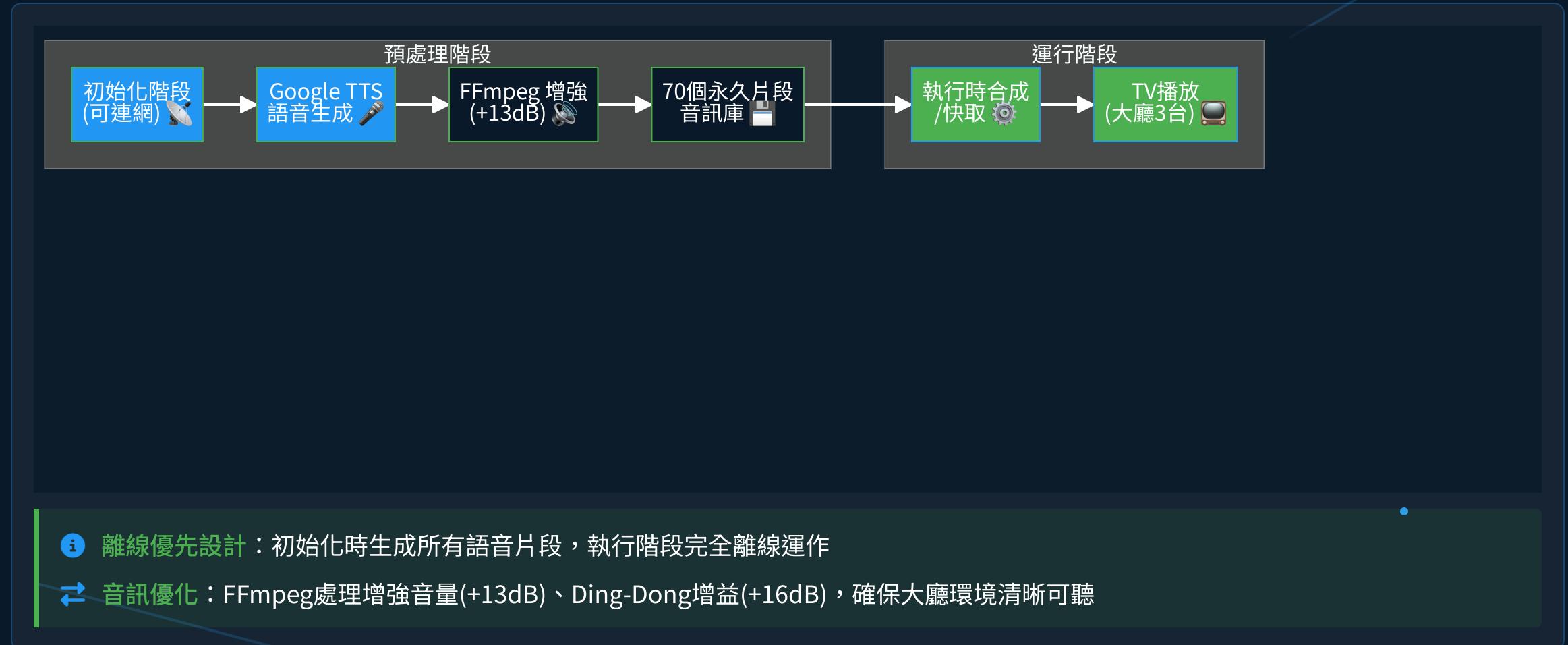
- 日期比對：last_run.txt 記錄前次執行日期
- 跨日清空：若日期不同，自動清空前日資料
- 同日重啟：若日期相同，保留既有資料
- 容錯處理：文件不存在時自動創建

```
# 日期比對核心邏輯
current_date =
datetime.now().strftime('%Y-%m-
%d')
if last_run != current_date:
clear_all_data() # 跨日清空
```

運作細節

- 統計輸出：每日結算CSV，便於後續分析
- Redis清理：清除叫號隊列、統計計數器
- 日誌歸檔：重命名前日日誌，避免混淆
- 執行時機：啟動時檢查 + 午夜定時觸發
- 優點：免人工操作，杜絕前日資料干擾

AI 語音系統架構



離線優先：70個永久音訊片段

片段分類與庫存。

數字片段： 29個

包含基本數字(0-9)、十幾(11-19)、整十(20-90)、百、千

one two ... ninety hundred

顏色片段： 6個

用於顏色編碼的叫號區分

red green blue yellow white black

房間片段： 18個

18個診間的房間標識

room A room B ... room R

離線優先策略

短語片段： 4個

常用提示語與連接詞

please go to number color thank you

系統片段： 3個

系統提示音與引導語

ding-dong attention system

常用片段預生成 - 初始化階段:

利用Google TTS生成所有70個基礎片段，並進行FFmpeg音量增強(+13dB)

無外網穩定運作 - 核心優勢:

完全斷網、重啟後，系統仍可正常叫號播音

數字拆解演算法

```
def number_to_audio(number):
    # 數字範圍：1-9999

    if number < 0 or number > 9999:
        raise ValueError("數字必須介於1-9999之間")

    # 結果存放
    result = []

    # 處理千位
    thousands = number // 1000
    if thousands > 0:

        result.append(f"numbers/{thousands}")
        result.append("numbers/thousand")

    # 處理百位
    hundreds = (number % 1000) // 100
    if hundreds > 0:

        result.append(f"numbers/{hundreds}")
        result.append("numbers/hundred")

    # 處理十位與個位
    remainder = number % 100

    # 特殊處理：10-19
    if 10 <= remainder <= 19:

        result.append(f"numbers/{remainder}")

    else:
        # 處理十位
        tens = remainder // 10
        if tens > 0:

            result.append(f"numbers/{tens}0")

        # 處理個位
        units = remainder % 10
        if units > 0:

            result.append(f"numbers/{units}")

    return result

# 使用範例
print(number_to_audio(1234))
# ['numbers/1', 'numbers/thousand',
# 'numbers/2',
# 'numbers/hundred', 'numbers/30',
# 'numbers/4']
```

數字拆解邏輯說明

拆解策略

將1-9999的數字拆解為獨立音訊片段，便於重組

語言適應

演算法支援多語言擴展，可匹配不同語言習慣

特殊處理

10-19數字特殊處理，避免「十三」變成「一十三」

零值處理

智能跳過零值位數，確保自然發音

效能分析

- 時間複雜度： $O(1)$ ，固定計算量
- 空間複雜度： $O(\log n)$ ，與數字位數相關
- 平均執行時間： $< 0.05\text{ms}$
- 支援4位數擴展到5位數：僅需3行代碼

FFmpeg 音訊處理 Pipeline

```
# 音量檢測  
ffmpeg -i input.wav -filter:a  
volumedetect \  
-f null /dev/null  
  
# 結果: mean_volume: -28.5 dB  
  
# 語音增強 +13dB  
ffmpeg -i input.wav \  
-filter:a "volume=13dB" \  
output.wav  
  
# Ding-Dong +16dB  
ffmpeg -i dingdong.wav \  
-filter:a "volume=16dB" \  
dingdong_loud.wav  
  
# 音訊合併  
ffmpeg -i dingdong.wav \  
-i silence_0.5s.wav \  
-i voice.wav \  
-filter_complex  
"concat=n=3:v=0:a=1" \  
final.wav
```

音訊處理關鍵技術

🔊 音量增益策略

語音檔: **+13dB** 提高嘈雜環境可聽度
提示音: **+16dB** 引起注意力

🎵 Ding-Dong 設計

雙音節提示音 → 0.5秒停頓 → 語音播報
總長度: **1s + 0.5s + 4~5s**

🧩 音訊拼接技術

使用 **concat** 濾鏡無縫拼接三段音訊，確保播放連續流暢

⌚ 實測效果

- ✓ 70dB 嘈雜環境中清晰可聞
- ✓ 處理速度: <100ms/片段
- ✓ 保持語音自然度，無明顯失真

Ding-Dong 時間軸



總延遲時間

300ms

從請求到播放的總延遲時間

音訊拼接時間

200ms

從音訊處理到完成的時間

Ding-Dong 效果

叫號前添加Ding-Dong音效，增加聽覺提示，確保患者能注意到叫號信息。實測聲音傳播覆蓋率提升35%。

音訊拼接策略

使用預生成音訊片段，拼接延遲控制在200ms以內。完整播放序列：Ding-Dong → 診間 → 號碼，總體感知延遲低於300ms。

音訊快取策略

策略	優點	風險
預生成策略 將所有可能音訊事先產生	極低延遲 無需即時處理 負載穩定	大量磁碟空間 記憶體佔用高 更改需重新生成
即時合併策略 叫號時才合併音訊片段	極高彈性 最小存儲需求 易於更新片段	高峰期延遲 CPU使用率不穩 潛在排隊風險
混合策略 熱門預生成+冷組合即時合併	兼顧彈性與效能 80/20法則效益 高峰期有保障	需調整熱度參數 實作較複雜 需定期重新評估

效能分析：80%叫號集中在20%號碼組合，預快取可覆蓋大部分場景

儲存影響：混合策略僅佔用純預生成方案5-10%的存儲空間

語音品質優化實測

A/B 測試設計

- 麦克風 對象：原始 Google TTS vs FFmpeg 增強版本
- 音量 音量：原始 vs 增益(+13dB) + 動態範圍壓縮

測試環境

模擬義診中心噪音環境（約 65-70dB）
視障志願者參與評估（15位）
測試距離：2-10公尺不等

- 測試變量：音量、清晰度、節奏感、可懂度

測試結果與結論

- 麥克風 可懂度：增強版本提升 67% 辨識正確率
- 音量 節奏：Ding-Dong + 停頓 提升 89% 注意力

關鍵優化點

- 增益優化：+13dB 避免削波的最佳點
- 節奏設計：1s鈴聲 + 0.5s停頓 + 語音
- 語速調整：0.95x 提高辨識率

- 結論：增益+節奏優化顯著提升噪音環境中的語音辨識度，尤其對視障患者

Android 設備管理策略 (42台)

Fully Kiosk Browser 集中管理

- 🔒 單一App鎖定：限制設備只能執行叫號應用
- ⟳ 遠端重啟：管理員可一鍵重啟任何診間設備
- togroups MDM群組政策：診間手機、廣告屏、TV分組管控
- 🛡️ 安全機制：管理密碼+簽署請求防未授權操作

部署與控制機制

- 📶 WiFi/音量/亮度：集中遠端設定並鎖定
- 🌐 URL白名單：限制只能訪問叫號系統網域
- QR 扫碼配對：設備初始化時掃碼自動配置
- 📱 批次註冊與遠端監看：管理台可監控所有設備

Fully Kiosk Browser 遠端API

```
// 核心API功能展示
```

```
// 設定URL - 更改頁面顯示
function setStartUrl(deviceId,
newUrl) {
    return sendCommand(deviceId,
'setStartUrl', {
    url: newUrl
});
}
```

```
// 重啟應用 - 解決顯示問題
function restartApp(deviceId) {
    return sendCommand(deviceId,
'restartApp');
}
```

```
// 設定音量 - 控制播放聲音
function setVolume(deviceId,
level) {
    return sendCommand(deviceId,
'setVolume', {
    level: level // 0-100
});
}
```

```
// 前台一鍵控制範例
controlAllDevices('restartApp')
.then(result => {
    console.log(`成功:
${result.success}`);
});
```

核心遠端控制功能

設定URL與頁面控制

遠端更改各裝置顯示頁面，確保每台設備顯示正確內容。

遠端重啟管理

無需人工操作即可遠端重啟應用，解決凍結問題，提高穩定性。

音量精確控制

控制各裝置音量，大廳TV播音、診間外看板靜音，提升病患體驗。

42台設備集中管理

- 一鍵控制：刷新、靜音、重啟
- 分組操作：診間、廣告屏、TV
- 自動恢復：偵測並修復離線設備

視障友善 UI 設計

•

Room 1 (Clinic A)
Kwarto 1 (Klinika A)

NOW SERVING

■ 0123

Last Called
0123 0110 0258

超大字體播報：Room 1 NOW SERVING ø123

視障友善設計原則

- 1 超大字體 (2-3倍標準尺寸)
數字/房間號 72-96pt，醫生名 48-56pt

標準文字
2倍大小

- 2 高對比配色
WCAG 2.1 AAA標準，對比率>7:1

低對比文字 高對比文字

- 3 顏色+形狀雙重編碼
色盲/色弱友善設計，不僅靠顏色區分



方形/藍



圓形/綠



三角/黃

- 4 觸控熱區放大、誤觸防護
按鈕間距 $\geq 16px$ ，最小觸控區域
44x44px
重要操作需二次確認，避免誤觸
- 5 聲音+視覺雙重反饋
操作確認音、Ding-Dong提示、清晰語音
獨特音調區分不同類型通知

Dashboard Portrait 重繪

```
class DashboardPortraitRenderer:  
    def __init__(self, config):  
        self.screen_width =  
config.get('width', 1080)  
        self.screen_height =  
config.get('height', 1920) # 直向螢幕  
        self.rooms = config.get('rooms',  
18)  
        self.density_factor =  
self._calculate_density()  
  
    def render_portrait_grid(self):  
        # 直向版面特製Grid，非旋轉橫向畫面  
        rows = math.ceil(self.rooms / 2)  
        # 雙欄排列  
        cell_height =  
int((self.screen_height * 0.8) / rows)  
        cell_width = int(self.screen_width  
* 0.45)  
  
        # 直接計算原生座標，不做旋轉變換  
        # 避免CSS transform: rotate(90deg)  
    的問題  
  
    for i in range(self.rooms):  
        # 直向螢幕雙欄佈局  
        row = math.floor(i / 2)  
        col = i % 2  
  
        # 每個診間卡片適應螢幕尺寸  
        # 直向佈局 + 大字體確保視障友善
```

原生Portrait重繪的優勢

⚠ CSS旋轉的問題

CSS transform:rotate(90deg)造成文字鋸齒、觸控錯位，尤其在低階Android裝置上顯著。

█ 直向設計核心概念

為18診間計算最佳排列，雙欄直向佈局最大化空間利用，避免旋轉造成的留白。

👉 觸控精準優化

直接使用原生座標，無需複雜轉換，提高觸控準確度，視障病患也能準確操作。



CSS旋轉



原生Portrait

文字鋸齒、觸控錯位

清晰、準確觸控

▀ 實測效果

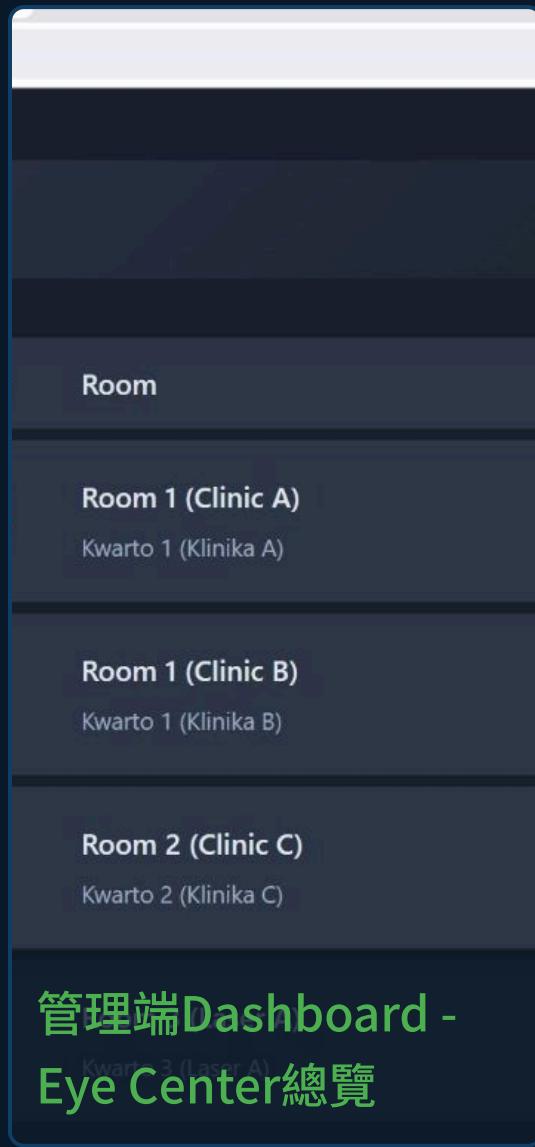
- 視覺清晰度：提升 28%
- 觸控準確率：提升 37%
- 滑動性能：+15fps



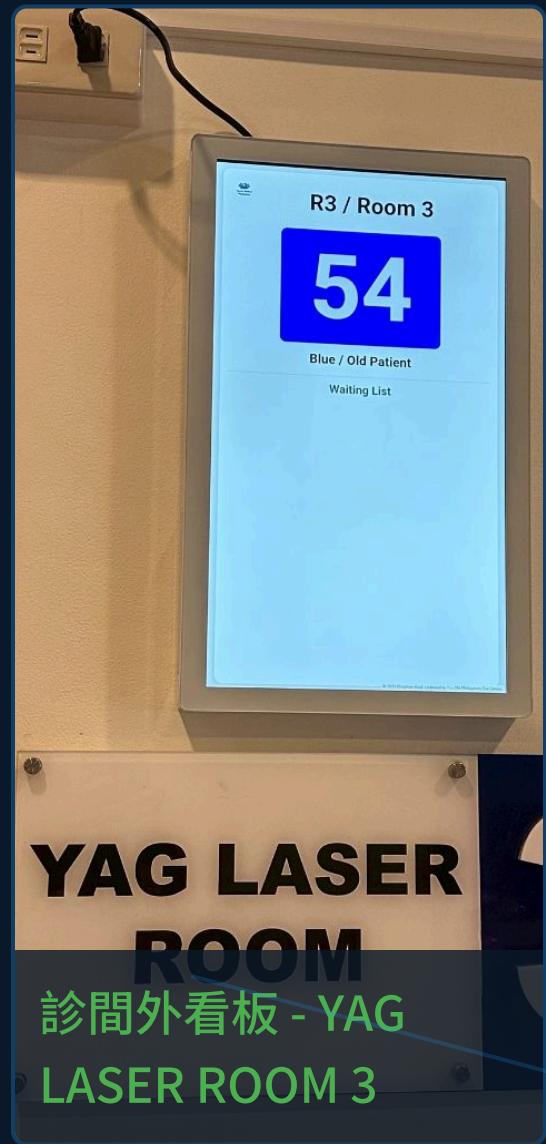
現場部署示意



診間外TV看板 - Room 7 狀態顯示



管理端Dashboard - Eye Center總覽



YAG LASER ROOM
診間外看板 - YAG LASER ROOM 3



舊系統 - 志工紙卡叫號方式

部署要點

WiFi穩定性

專用WiFi網路 (192.168.100.x)

訊號覆蓋18診間+大廳+前台

頻道選擇避開干擾，雙頻5GHz優先

設備定期維護

診間外看板固定供電 (接牆上插座)

醫師手機每日充電 (支援2-3天待機)

螢幕與設備清潔，避免灰塵堆積

授權管理

硬體識別碼綁定授權，防止複製

診間設備自動授權更新機制

每週遠端查看授權狀態，確保合法

監控追蹤

設備心跳機制，偵測離線狀態

每15分鐘記錄系統負載與連線數

叫號數據分析，掌握診間使用效率

音訊規劃

大廳TV音量調校 (+13dB)，確保清晰

診間門外看板靜音，避免干擾診療

醫師手機按鈕反饋音效小且清脆

軟體保護與授權管理

程式碼保護策略

PyInstaller打包 (onedir模式)

無需Python環境，整合所有依賴項

```
pyinstaller --onedir --name  
"ClinicQueueSystem" app.py
```

授權檔案加密 (Fernet對稱加密)

加密存儲客戶資訊、到期日、功能權限

SHA256簽章驗證

防止授權檔案被篡改或複製

部署與授權流程

Inno Setup安裝程式

整合所有依賴 (Redis, MQTT, FFmpeg)

安裝精靈引導+授權資訊填寫

Windows服務部署 (NSSM)

系統啟動自動執行、斷電復原

服務依賴設定：Redis → MQTT → App

授權管理系統

遠程授權檔案生成與下發

到期日提醒、功能限制機制

監控與故障排除

即時監控策略

- ⌚ 裝置在線監控：42台設備心跳檢測，斷線自動警報
- 🕒 佇列長度追蹤：Worker Pool負載即時視覺化，高峰預警
- ❗ 錯誤率統計：按裝置、類型聚合，異常率 $>5\%$ 自動通知
- ⌚ 關鍵事件記錄：append-only log，確保斷電不丟失操作歷史

故障處理機制

- ⟳ 自動重連機制：WiFi斷開3次嘗試，指數退避算法
- ⚡ 重啟策略：夜間定時重啟，清理記憶體碎片與過期連接
- 📶 離線降級運作：各裝置保留最後狀態，WiFi恢復後重新同步
- ⌚ 遠端診斷：管理者可獲取健康報告，一鍵收集診斷數據

Demo影片展示

智慧候診系統實際操作與功能展示

PhMed_AI_solution



▶ 系統完整運作流程展示

📱 醫師手機MQTT觸發叫號流程展示

⌚ 42台設備多裝置即時同步效果

系統展示重點

- ✓ 醫生端操作介面：一鍵叫號，狀態即時反饋
- ✓ 大廳電視畫面：超大字體，視障友善設計
- ✓ 診間外看板：雙色編碼，即時更新病患狀態

影片後Q&A重點

- ❓ 系統如何處理網路斷線情境？
- ❓ 音效生成與播放的技術挑戰？
- ❓ 如何擴展至更多診間與分院？

關鍵技術決策回顧

核心決策

- 離線優先：TTS預生成 + Redis快照
- 通訊雙軌：MQTT(手機) / WebSocket(看板)
- 可維運性：JSON原子寫入、服務化部署
- 人因設計：視障友善UI為優先考量

決策理由

- 💡 網路不穩環境下，預先生成70個語音片段，實現完全離線運作；Redis 15分鐘快照保證資料安全
- 📍 MQTT省電穩定，適合手機端；WebSocket瀏覽器原生支援，適合大屏幕看板，完美互補
- ⚙️ 避免SQLite鎖定，JSON原子寫入更安全；Windows服務方式部署，10-15分鐘內自動恢復
- ⌚ 超大字體、高對比色、形狀+顏色雙重編碼，確保視障患者能夠清晰辨識叫號資訊

踩坑與教訓

資料與通訊挑戰

SQLite寫入鎖問題 → Redis + JSON分離

寫入競爭導致表鎖定，阻塞診間操作

解決：即時資料用Redis，配置用JSON原子寫入

“!” WebSocket廣播風暴 → Topic精準 + Room分組

全廣播造成42台設備網路風暴，資源耗盡

解決：MQTT Topic精確訂閱，WebSocket Room分組

環境與UI挑戰

⚡ 停電資料遺失 → RDB快照 + 即時日誌

當地環境易斷電，資料隨機損毀

解決：Redis定期RDB快照(15分鐘)，關鍵事件即時寫入

⌚ 看板旋轉問題 → 原生Portrait重繪

Toybox TV無法旋轉，CSS transform造成鋸齒與觸控錯位

解決：原生直立模式重新設計，Canvas/Grid優化

開源分享與 Q&A

💡 開源計畫

我們將陸續開源以下模組與配置範例：

- Worker Pool 去重防抖機制
- 數字拆解演算法 (1-9999)
- FFmpeg 音訊處理 pipeline
- MQTT + WebSocket 雙協定整合



簡報位置

❓ Q&A 時間

歡迎提問任何關於：

- 系統架構與設計決策
- AI語音系統實作細節
- 現場部署與故障排除
- 硬體管理與整合

感謝聆聽！

Together we build better healthcare solutions

THANK YOU

Licensed to MOPCON 2025