AI Secretary — Runbook v9 (v6 + latest iOS code & weekly sum

AI Secretary — Runbook v6 (FULL CONTEXT PACK)

> **Immediate Next Step (do this first):**
> On **Render → Services → CGPTPROJECT-v2 → Environment**, set:
> `APP_BACKEND_BEARER`, `OPENAI_API_KEY`, `AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY` (optional `TTS_CACHE_S3_BUCKET`).
> Click **Save, rebuild, and deploy**. Then open `/diagnostics` to confirm flags are **true**. Finally, call `/ask` and `/speak` using the same bearer in the `Authorization` header.

---

0) Final Backend Files (drop-in)

`app.py` (production-ready Flask + Polly + OpenAI; JSON /health; bearer auth)

```
from __future__ import annotations
import hashlib
import hmac
import logging
import os
import sys
from functools import wraps
from typing import Callable, Optional, Tuple
import boto3
from botocore.exceptions import BotoCoreError, ClientError
from flask import Flask, Response, jsonify, make_response, request
# Optional OpenAI for /ask
try:
    from openai import OpenAI  # openai>=1.x
except Exception:  # pragma: no cover
    OpenAI = None  # type: ignore
# --- App setup ---
app = Flask(__name__)
app.logger.setLevel(logging.INFO)
app.logger.warning(
    "BOOT cwd=%s file=%s py=%s",
    os.getcwd(),
    __file__,
    sys.version.split()[0],
)
# --- Config ---
AWS_REGION = os.getenv("AWS_REGION", "us-east-1")
POLLY_VOICE_DEFAULT = os.getenv("POLLY_VOICE", "Joanna")
POLLY_FORMAT_DEFAULT = os.getenv("POLLY_FORMAT", "mp3")  # mp3 | ogg_vorbis
POLLY_ENGINE_DEFAULT = os.getenv("POLLY_ENGINE", "neural")
TTS_CACHE_S3_BUCKET = os.getenv("TTS_CACHE_S3_BUCKET")
# Security (why: prevent public abuse)
BACKEND_BEARER = os.getenv("APP_BACKEND_BEARER") or os.getenv("API_TOKEN")
# --- AWS clients ---
_session = boto3.session.Session(region_name=AWS_REGION)
polly = _session.client("polly")
s3 = _session.client("s3") if TTS_CACHE_S3_BUCKET else None
# --- Auth ---
def _extract_bearer(auth_header: Optional[str]) -> Optional[str]:
    if not auth_header:
        return None
    parts = auth_header.split()
```

```
    if len(parts) == 2 and parts[0].lower() == "bearer":
        return parts[1]
    return None
def require_bearer_token(fn: Callable):
    @wraps(fn)
    def wrapper(*args, **kwargs):
        if not BACKEND_BEARER:
            return jsonify({"error": "server_misconfigured_no_token"}), 500
        token = _extract_bearer(request.headers.get("Authorization"))
        if not token or not hmac.compare_digest(token, BACKEND_BEARER):
            return jsonify({"error": "unauthorized"}), 401
        return fn(*args, **kwargs)
    return wrapper
# --- TTS helpers ---
def _tts_cache_key(text: str, voice: str, fmt: str, engine: str) -> str:
    h = hashlib.sha256()
    h.update(voice.encode()); h.update(b"|")
    h.update(fmt.encode());   h.update(b"|")
    h.update(engine.encode());h.update(b"|")
    h.update(" ".join(text.split()).encode())  # normalize whitespace
    ext = "mp3" if fmt == "mp3" else "ogg"
    return f"tts/{voice}/{engine}/{h.hexdigest()}.{ext}"
def _s3_get(bucket: str, key: str) -> Optional[bytes]:
    if not s3:
        return None
    try:
        obj = s3.get_object(Bucket=bucket, Key=key)
        return obj["Body"].read()
    except (ClientError, BotoCoreError, KeyError):
        return None
def _s3_put(bucket: str, key: str, data: bytes, content_type: str) -> None:
    if not s3:
        return
    try:
        s3.put_object(
            Bucket=bucket,
            Key=key,
            Body=data,
            ContentType=content_type,
            CacheControl="public, max-age=31536000, immutable",
        )
    except (ClientError, BotoCoreError) as e:
        app.logger.warning("S3 cache put failed: %s", e)
def synthesize_speech(
    text: str,
    voice: str = POLLY_VOICE_DEFAULT,
    fmt: str = POLLY_FORMAT_DEFAULT,
    engine: str = POLLY_ENGINE_DEFAULT,
) -> Tuple[Optional[bytes], Optional[str]]:
    if not text:
        return None, None
    # Cache check
    cache_key = None
    if TTS_CACHE_S3_BUCKET:
        cache_key = _tts_cache_key(text, voice, fmt, engine)
        cached = _s3_get(TTS_CACHE_S3_BUCKET, cache_key)
        if cached:
            ct = "audio/mpeg" if fmt == "mp3" else "audio/ogg"
```

```
            return cached, ct
    try:
        resp = polly.synthesize_speech(
            Text=text,
            OutputFormat=fmt,
            VoiceId=voice,
            Engine=engine,
        )
        audio = resp["AudioStream"].read()
        ct = "audio/mpeg" if fmt == "mp3" else "audio/ogg"
        if TTS_CACHE_S3_BUCKET and cache_key and audio:
            _s3_put(TTS_CACHE_S3_BUCKET, cache_key, audio, ct)
        return audio, ct
    except Exception as e:  # boundary
        app.logger.exception("Polly error: %s", e)
        return None, None
# --- Utilities ---
def _openai_client():
    if OpenAI is None:
        raise RuntimeError("openai package not available")
    key = os.getenv("OPENAI_API_KEY")
    if not key:
        raise RuntimeError("OPENAI_API_KEY not set")
    return OpenAI(api_key=key)
def generate_ai_reply(prompt: str) -> str:
    model = os.getenv("OPENAI_MODEL", "gpt-4o-mini")
    try:
        client = _openai_client()
        res = client.chat.completions.create(
            model=model,
            messages=[
                {"role": "system", "content": "You are a concise AI
Secretary."},
                {"role": "user", "content": prompt},
            ],
            temperature=0.3,
        )
        return (res.choices[0].message.content or "").strip()
    except Exception as e:
        app.logger.warning("OpenAI error: %s", e)
        return "I'm here."
# --- Routes (public JSON) ---
@app.get("/health")
def health() -> Response:
    return jsonify({"ok": True, "service": "ai-secretary", "polly_region":
AWS_REGION})
@app.get("/diagnostics")
def diagnostics() -> Response:
    import flask
    import boto3 as boto3_pkg
    return jsonify(
        {
            "python_version": sys.version.split()[0],
            "flask_version": flask.__version__,
            "boto3_version": boto3_pkg.__version__,
            "aws_region": AWS_REGION,
            "s3_cache_enabled": bool(TTS_CACHE_S3_BUCKET),
            "env_flags": {
```

```
                "APP_BACKEND_BEARER": bool(os.getenv("APP_BACKEND_BEARER")),
                "OPENAI_API_KEY": bool(os.getenv("OPENAI_API_KEY")),
                "AWS_ACCESS_KEY_ID": bool(os.getenv("AWS_ACCESS_KEY_ID")),
                "AWS_SECRET_ACCESS_KEY":
bool(os.getenv("AWS_SECRET_ACCESS_KEY")),
                "TTS_CACHE_S3_BUCKET": bool(os.getenv("TTS_CACHE_S3_BUCKET")),
            },
        }
    )
@app.get("/__stamp")
def __stamp() -> Response:
    return jsonify(
        {"commit": os.getenv("RENDER_GIT_COMMIT") or os.getenv("GIT_SHA") or
"unknown"}
    )
@app.get("/__where")
def __where() -> Response:
    return jsonify({"cwd": os.getcwd(), "__file__": __file__})
@app.get("/")
def index():
    return (
        "<h1>AI Secretary</h1>"
        '<p>Service is running. Try <a href="/health">/health</a> or <a
href="/diagnostics">/diagnostics</a>.</p>',
        200,
        {"Content-Type": "text/html; charset=utf-8"},
    )
# --- Routes (auth required) ---
@app.post("/ask")
@require_bearer_token
def ask() -> Response:
    data = request.get_json(silent=True) or {}
    prompt = str(data.get("prompt", "")).strip()
    if not prompt:
        return jsonify({"error": "missing_prompt"}), 400
    reply = generate_ai_reply(prompt)
    return jsonify({"reply": reply})
@app.post("/speak")
@require_bearer_token
def speak() -> Response:
    data = request.get_json(silent=True) or {}
    text = str(data.get("text", "")).strip()
    voice = str(data.get("voice", POLLY_VOICE_DEFAULT))
    fmt = str(data.get("format", POLLY_FORMAT_DEFAULT))
    engine = str(data.get("engine", POLLY_ENGINE_DEFAULT))
    audio, content_type = synthesize_speech(text, voice=voice, fmt=fmt,
engine=engine)
    if not audio:
        return jsonify({"error": "tts_failed"}), 500
    resp = make_response(audio)
    resp.headers["Content-Type"] = content_type or "application/octet-stream"
    return resp
if __name__ == "__main__":
    port = int(os.getenv("PORT", "5000"))
    app.run(host="0.0.0.0", port=port, debug=bool(os.getenv("FLASK_DEBUG")))
```

`requirements.txt`

```
flask>=3.0.0
gunicorn>=21.2.0
```

```
boto3>=1.35.0
botocore>=1.35.0
openai>=1.35.0
```

`render.yaml`

```
services:
  - type: web
    name: CGPTPROJECT-v2
    runtime: python
    buildCommand: "pip install -r requirements.txt"
    startCommand: "gunicorn -b 0.0.0.0:$PORT app:app"
    healthCheckPath: "/health"
    envVars:
      - key: PYTHON_VERSION
        value: 3.12.5
      - key: APP_BACKEND_BEARER
        sync: false
      - key: OPENAI_API_KEY
        sync: false
      - key: AWS_ACCESS_KEY_ID
        sync: false
      - key: AWS_SECRET_ACCESS_KEY
        sync: false
      - key: AWS_REGION
        value: us-east-1
```

`.env.example`

```
APP_BACKEND_BEARER=changeme-dev-token
OPENAI_API_KEY=sk-...
AWS_ACCESS_KEY_ID=AKIA...
AWS_SECRET_ACCESS_KEY=...
AWS_REGION=us-east-1
# Optional voice/cache
POLLY_VOICE=Joanna
POLLY_ENGINE=neural
POLLY_FORMAT=mp3
# If you want S3 caching of TTS results:
# TTS_CACHE_S3_BUCKET=your-bucket-name
---
```

1) iOS Client Files

`ios/App/Networking/JarvisClient.swift`

```
import Foundation
struct JarvisClient {
    let baseURL: URL
    let bearer: String
    private func request(path: String, json body: [String: Any]) throws ->
URLRequest {
        guard let url = URL(string: path, relativeTo: baseURL) else { throw
URLError(.badURL) }
        var req = URLRequest(url: url)
        req.httpMethod = "POST"
        req.setValue("application/json", forHTTPHeaderField: "Content-Type")
        req.setValue("Bearer \(bearer)", forHTTPHeaderField: "Authorization")
        req.httpBody = try JSONSerialization.data(withJSONObject: body,
options: [])
        req.timeoutInterval = 30
        return req;
    }
```

```
    func ask(prompt: String) async throws -> String {
        let req = try request(path: "/ask", json: ["prompt": prompt])
        let (data, resp) = try await URLSession.shared.data(for: req)
        guard let http = resp as? HTTPURLResponse,
(200..<300).contains(http.statusCode) else {
            throw URLError(.badServerResponse)
        }
        let obj = try JSONSerialization.jsonObject(with: data) as? [String:
Any]
        guard let reply = obj?["reply"] as? String else { throw
URLError(.cannotDecodeRawData) }
        return reply
    }
    func speak(text: String, voice: String? = nil) async throws -> Data {
        var body: [String: Any] = ["text": text]
        if let voice { body["voice"] = voice }
        let req = try request(path: "/speak", json: body)
        let (data, resp) = try await URLSession.shared.data(for: req)
        guard let http = resp as? HTTPURLResponse,
(200..<300).contains(http.statusCode) else {
            throw URLError(.badServerResponse)
        }
        return data
    }
}
```

`ios/App/Audio/AudioPlayer.swift`

```
import AVFoundation
final class AudioPlayer: NSObject, AVAudioPlayerDelegate {
    private var player: AVAudioPlayer?
    private let session = AVAudioSession.sharedInstance()
    func play(data: Data) {
        do {
            try session.setCategory(.playback, mode: .default, options:
[.duckOthers])
            try session.setActive(true, options: [])
            let p = try AVAudioPlayer(data: data)
            p.prepareToPlay(); p.play()
            self.player = p
        } catch { print("AudioPlayer error:", error) }
    }
    func stop() {
        player?.stop()
        player = nil
        try? session.setActive(false, options: [.notifyOthersOnDeactivation])
    }
}
```

`ios/App/Views/VoiceTestView.swift`

```
import SwiftUI
import AVFoundation
struct VoiceTestView: View {
    private let client = JarvisClient(
        baseURL: URL(string: "https://cgptproject-v2.onrender.com")!,
        bearer: "<APP_BACKEND_BEARER>"
    )
    @State private var prompt = "Say hello in three words."
    @State private var reply = ""
    @State private var status = "Idle"
```

```
    @State private var lastAudio: Data?
    private let player = AudioPlayer()
    var body: some View {
        VStack(spacing: 16) {
            Text("Backend: cgptproject-v2").font(.headline)
            TextField("Prompt", text:
$prompt).textFieldStyle(.roundedBorder).padding(.horizontal)
            HStack {
                Button("Ask") { Task { await ask() } }
                Button("Speak") { Task { await speak() } }
                Button("Play Last") { if let a = lastAudio { player.play(data:
a) } }
                Button("Stop") { player.stop() }
            }.buttonStyle(.borderedProminent)
            Text("Reply: \(reply)").padding(.horizontal)
            Text("Status:
\(status)").font(.footnote).foregroundColor(.secondary)
        }.padding()
    }
    private func ask() async {
        await MainActor.run { status = "Asking…" }
        do { let r = try await client.ask(prompt: prompt)
             await MainActor.run { reply = r; status = "Ask ✓" } }
        catch { await MainActor.run { status = "Ask failed:
\(error.localizedDescription)" } }
    }
    private func speak() async {
        await MainActor.run { status = "Speaking…" }
        do { let audio = try await client.speak(text: prompt)
             await MainActor.run { lastAudio = audio; status = "Speak ✓
(playing)"; player.play(data: audio) } }
        catch { await MainActor.run { status = "Speak failed:
\(error.localizedDescription)" } }
    }
}
@main
struct JarvisClientApp: App {
    var body: some Scene { WindowGroup { VoiceTestView() } }
}
```

---

2) Render Settings (final)

- **Start Command:** `gunicorn -b 0.0.0.0:$PORT app:app`
- **Build Command:** `pip install -r requirements.txt`
- **Health Check Path:** `/health`
- **Service URL (v2):** `https://cgptproject-v2.onrender.com`
- **Environment:** set the secrets; `/diagnostics` must show `true` for them.

---

3) Key Logs & Evidence (from this session)

- Repeated 200s for `/health`; transient 502 during rollout; diagnostics before/after secrets.

---

4) Conversation Snapshot (condensed)

- Old HTML `/health` on prior services → created **v2**, ensured `app:app`, added root `/` route, cleared cache, validated JSON health, set env, delivered iOS client code.

---
5) Hand-off TL;DR to start a new chat
> Backend is at `https://cgptproject-v2.onrender.com` with JSON `/health`;
bearer auth via `APP_BACKEND_BEARER`; AWS/OpenAI keys present; need help
refining iOS client and optional S3 TTS cache.

Repository Inventory (Backend) — as of your screenshot
Root files
- `app.py` — active Flask backend (v2), JSON `/health`, `/diagnostics`,
`/ask`, `/speak`.
- `app_working.py` — older/alternate app module (not used by Gunicorn).
- `chat_gpt_voice_assistant.py` — legacy script.
- `constraints.txt` — Python constraints (pin set).
- `Makefile` — helper scripts (local/dev).
- `Procfile` — Heroku-style start command (currently unused if Service Start
Command is set in Render).
- `README.md` — project readme.
- `render.yaml` — Render Blueprint (defines `CGPTPROJECT-v2`).
- `requirements.txt` — Python deps (Flask, Gunicorn, Boto3, OpenAI).
- `response.mp3` — sample audio artifact.
- `runtime.txt` — Python runtime hint (legacy; Render uses PYTHON_VERSION env
instead).
Folder: `scripts/`
- `ask_json.sh` — curl helper for `/ask` returning JSON.
- `prove_runtime.sh` — prints runtime info.
- `smoke.sh` — quick health/auth checks.
- `ws_test.py` — websocket/aux testing script (legacy).
Folder: `venv/` (local virtualenv)
- `bin/`, `include/`, `lib/`, `pyvenv.cfg` — local-only; **do not** rely on
these on Render.
> Note: Gunicorn entrypoint for production is **`app:app`** (file `app.py` at
repo root). Keep `CGPTPROJECT-v2` Start Command as:
> `gunicorn -b 0.0.0.0:$PORT app:app`
---
iOS Client Project Inventory (from your screenshot)
Project root
- `JarvisClient.xcodeproj` — Xcode project file
Target: JarvisClient/
- `Assets.xcassets/`
  - `AccentColor.colorset/`
  - `AppIcon.appiconset/`
  - `Contents.json`
- `ContentView.swift` — your existing SwiftUI view
- `Info.plist` — app permissions (microphone if needed), bundle settings
- `Item.swift` — template model from starter app
- `JarvisClientApp.swift` — SwiftUI app entry

- `secrets.swift` — your local constants (URL/token)
Targets
- `JarvisClientTests/` — unit tests
- `JarvisClientUITests/` — UI tests
  - `JarvisClientUITests.swift`
  - `JarvisClientUITestsLaunchTests.swift`
New client files we supplied (add to project + Target Membership)
- `ios/App/Networking/JarvisClient.swift` — HTTP client for `/ask` and `/speak`
- `ios/App/Audio/AudioPlayer.swift` — MP3 playback (no mic required)
- `ios/App/Views/VoiceTestView.swift` — buttons to Ask/Speak/Play/Stop
> Set in `VoiceTestView.swift`:
> - `baseURL = https://cgptproject-v2.onrender.com`
> - `bearer = <APP_BACKEND_BEARER>` (exactly matches the server)

1.1 iOS Client (Latest Architecture — HTTP only)

- We removed the WebSocket `/voice` path (server doesn't implement it) and use **only HTTP**:
  - `POST /ask` with JSON `{ "prompt": string }` → `{ "reply": string }`
  - `POST /speak` with JSON `{ "text": string, "voice"?: string }` → raw MP3 bytes
- New files added to target **JarvisClient**:
  - `Secrets.swift` — base URL + bearer (must match Render)
  - `JarvisClient.swift` — minimal HTTP client
  - `AudioPlayer.swift` — MP3 playback helper
  - `ContentView.swift` — updated UI (Ask/Speak/Play/Stop + New Button hook)
- Ensure **Target Membership** is checked for these files.

```
// file: Secrets.swift
import Foundation
enum Secrets {
    static let baseURL = URL(string: "https://cgptproject-v2.onrender.com")!
    static let backendBearer = "<APP_BACKEND_BEARER>" // EXACT value used on
Render
}
// file: JarvisClient.swift (HTTP; no WebSocket)
import Foundation
struct JarvisClient {
    let baseURL: URL
    let bearer: String
    private func request(path: String, json body: [String: Any]) throws ->
URLRequest {
        guard let url = URL(string: path, relativeTo: baseURL) else { throw
URLError(.badURL) }
        var req = URLRequest(url: url)
        req.httpMethod = "POST"
        req.setValue("application/json", forHTTPHeaderField: "Content-Type")
        req.setValue("Bearer \(bearer)", forHTTPHeaderField: "Authorization")
        req.httpBody = try JSONSerialization.data(withJSONObject: body,
options: [])
        req.timeoutInterval = 30
        return req
    }
```

```
    struct APIError: LocalizedError {
        let code: Int; let body: String
        var errorDescription: String? { "HTTP \(code): \(body)" }
    }
    func ask(prompt: String) async throws -> String {
        let req = try request(path: "/ask", json: ["prompt": prompt])
        let (data, resp) = try await URLSession.shared.data(for: req)
        guard let http = resp as? HTTPURLResponse,
(200..<300).contains(http.statusCode) else {
            let body = String(data: data, encoding: .utf8) ?? ""
            throw APIError(code: (resp as? HTTPURLResponse)?.statusCode ?? -1,
body: body)
        }
        let obj = try JSONSerialization.jsonObject(with: data) as? [String:
Any]
        guard let reply = obj?["reply"] as? String else { throw
URLError(.cannotDecodeRawData) }
        return reply
    }
    func speak(text: String, voice: String? = nil) async throws -> Data {
        var body: [String: Any] = ["text": text]
        if let voice { body["voice"] = voice }
        let req = try request(path: "/speak", json: body)
        let (data, resp) = try await URLSession.shared.data(for: req)
        guard let http = resp as? HTTPURLResponse,
(200..<300).contains(http.statusCode) else {
            let body = String(data: data, encoding: .utf8) ?? ""
            throw APIError(code: (resp as? HTTPURLResponse)?.statusCode ?? -1,
body: body)
        }
        return data
    }
}
// file: AudioPlayer.swift
import AVFoundation
final class AudioPlayer: NSObject, AVAudioPlayerDelegate {
    private var player: AVAudioPlayer?
    private let session = AVAudioSession.sharedInstance()
    func play(data: Data) {
        do {
            try session.setCategory(.playback, mode: .default, options:
[.duckOthers])
            try session.setActive(true, options: [])
            let p = try AVAudioPlayer(data: data)
            p.prepareToPlay(); p.play()
            self.player = p
        } catch { print("AudioPlayer error:", error) }
    }
    func stop() {
        player?.stop()
        player = nil
        try? session.setActive(false, options: [.notifyOthersOnDeactivation])
    }
}
// file: ContentView.swift (HTTP-only + New Button)
import SwiftUI
import AVFoundation
enum ResponseMode: String, CaseIterable, Identifiable, Codable {
```

```
    case friendly, concise
    var id: String { rawValue }
    var label: String { rawValue.capitalized }
}
struct ContentView: View {
    private let client = JarvisClient(baseURL: Secrets.baseURL, bearer:
Secrets.backendBearer)
    private let player = AudioPlayer()
    @State private var mode: ResponseMode = .concise
    @State private var prompt: String = "Say hello in three words."
    @State private var reply: String = ""
    @State private var status: String = "Ready"
    @State private var lastAudio: Data?
    @State private var isBusy = false
    var body: some View {
        NavigationView {
            VStack(spacing: 16) {
                Picker("Tone", selection: $mode) {
                    ForEach(ResponseMode.allCases) { Text($0.label).tag($0) }
                }
                .pickerStyle(.segmented)
                .padding(.horizontal)
                TextField("Type a message…", text: $prompt, axis: .vertical)
                    .textInputAutocapitalization(.sentences)
                    .lineLimit(1...6)
                    .padding(12)
                    .background(Color(.secondarySystemBackground))
                    .clipShape(RoundedRectangle(cornerRadius: 12))
                    .padding(.horizontal)
                HStack {
                    Button { Task { await doAsk() } } label: {
                        HStack { if isBusy { ProgressView() } ; Text("Ask") }
                    }
                    .buttonStyle(.borderedProminent)
                    .disabled(isBusy || prompt.trimmingCharacters(in:
.whitespacesAndNewlines).isEmpty)
                    Button { Task { await doSpeak() } } label: { Text("Speak")
}
                        .buttonStyle(.bordered)
                        .disabled(isBusy || prompt.isEmpty)
                    Button("Play Last") { if let data = lastAudio {
player.play(data: data) } }
                        .disabled(lastAudio == nil)
                    Button("Stop") { player.stop() }
                }
                .padding(.horizontal)
                Button { Task { await newButtonAction() } } label: {
                    Text("New Button").frame(maxWidth: .infinity)
                }
                .buttonStyle(.bordered)
                .padding(.horizontal)
                if !reply.isEmpty {
                    Divider()
                    Text(reply).frame(maxWidth: .infinity, alignment:
.leading).padding(.horizontal)
                }
                Spacer()
                Text("Status:
\(status)").font(.footnote).foregroundStyle(.secondary).padding(.bottom, 8)
```

```
            }
            .navigationTitle("AI Secretary • v2")
        }
        .task { await checkHealth() }
    }
    private func checkHealth() async {
        var req = URLRequest(url:
Secrets.baseURL.appendingPathComponent("/health"))
        req.httpMethod = "GET"
        do {
            let (_, resp) = try await URLSession.shared.data(for: req)
            status = (resp as? HTTPURLResponse)?.statusCode == 200 ? "Server:
OK" : "Server: unreachable"
        } catch { status = "Health failed: \(error.localizedDescription)" }
    }
    private func doAsk() async {
        await MainActor.run { isBusy = true; status = "Asking…" }
        defer { Task { await MainActor.run { isBusy = false } } }
        do {
            let r = try await client.ask(prompt: prompt)
            await MainActor.run { reply = r; status = "Ask ✓" }
        } catch {
            await MainActor.run { status = "Ask failed:
\(error.localizedDescription)" }
        }
    }
    private func doSpeak() async {
        await MainActor.run { isBusy = true; status = "Speaking…" }
        defer { Task { await MainActor.run { isBusy = false } } }
        do {
            let mp3 = try await client.speak(text: prompt)
            await MainActor.run { lastAudio = mp3; status = "Speak ✓
(playing)"; player.play(data: mp3) }
        } catch {
            await MainActor.run { status = "Speak failed:
\(error.localizedDescription)" }
        }
    }
    private func newButtonAction() async {
        do {
            var req = URLRequest(url:
Secrets.baseURL.appendingPathComponent("/diagnostics"))
            req.httpMethod = "GET"
            let (data, _) = try await URLSession.shared.data(for: req)
            await MainActor.run { status = "Diagnostics bytes: \(data.count)"
}
        } catch {
            await MainActor.run { status = "New button failed:
\(error.localizedDescription)" }
        }
    }
}
```

2) Render & Environment (confirmed working)

- **Service**: CGPTPROJECT-v2 → `https://cgptproject-v2.onrender.com`
- **Start Command**: `gunicorn -b 0.0.0.0:$PORT app:app`
- **Build Command**: `pip install -r requirements.txt`
- **Health Check**: `/health` (returns JSON)

- **Environment Variables** (required): `APP_BACKEND_BEARER`, `OPENAI_API_KEY`, `AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY`, `AWS_REGION=us-east-1`
- Optional: `TTS_CACHE_S3_BUCKET` for S3-based Polly cache
- **Diagnostics**: `/diagnostics` confirms flags; use it before testing iOS.

3) Weekly Summary (high signal)

- Fixed 502s by standardizing Gunicorn start and pointing to the v2 service URL.
- Implemented `/diagnostics` and validated env flags.
- Backend now returns JSON `/health`; root `/` added for quick visual check.
- iOS app: removed WebSocket client for `/voice` (nonexistent) → switched to **HTTP** `/ask` + `/speak`.
- Added `Secrets.swift`, `JarvisClient.swift`, `AudioPlayer.swift`, and replaced `ContentView.swift`. New Button hook included.
- Strategy: You're open to **OpenAI GPTs/GPT Store** for orchestration (private GPT with Actions against your backend), keeping proprietary email/telephony server-side.

AI Secretary — Directory & File Trees (Addendum)

Backend repo (chatgpt_project/):
- app.py, app_working.py, chat_gpt_voice_assistant.py, constraints.txt, Makefile, Procfile, README.md,
- render.yaml, requirements.txt, response.mp3, runtime.txt
- scripts/: ask_json.sh, prove_runtime.sh, smoke.sh, ws_test.py
- venv/: (local virtualenv)

iOS App (JarvisClient/ target):
- APIClient.swift, AudioMonitor.swift, AudioPlayer.swift, ChatSession.swift, ChatStore.swift,
- ChatViewModel.swift, ChatVM.swift (if present), ContentView.swift, Haptics.swift,
- HistoryServerView.swift, HistoryView.swift, InputBar.swift, ListeningIndicator.swift, MailComposer.swift,
- Message.swift, TypingBubble.swift, secrets.swift, JarvisClient.swift, JarvisClientApp.swift, Info.plist,
- Assets.xcassets (AppIcon included), JarvisClient.xcodeproj

Developer Appendix — Auth + Login

Drop-in files and integration notes.

1) AuthViewModel.swift

```
// Path: JarvisClient/AuthViewModel.swift
import Foundation

@MainActor
final class AuthViewModel: ObservableObject {
    @Published var isAuthenticated = false
    @Published var email: String = ""
    @Published var password: String = ""
    @Published var error: String? = nil

    private let baseURL = Secrets.baseURL
    private let tokenKey = "auth.token"

    init() {
        // restore token
        if let token = UserDefaults.standard.string(forKey: tokenKey),
           !token.isEmpty {
            Secrets.authToken = token
            isAuthenticated = true
        }
    }

    func register() async {
        await auth(endpoint: "/auth/register")
    }

    func login() async {
        await auth(endpoint: "/auth/login")
    }

    func logout() {
        Secrets.authToken = nil
        UserDefaults.standard.removeObject(forKey: tokenKey)
        isAuthenticated = false
    }

    private func auth(endpoint: String) async {
        error = nil
        guard let url = URL(string: endpoint, relativeTo: baseURL) else {
            error = "Bad URL"; return
        }
        var req = URLRequest(url: url)
        req.httpMethod = "POST"
        req.setValue("application/json", forHTTPHeaderField: "Content-Type")
        let body: [String: String] = ["email": email, "password": password]
        req.httpBody = try? JSONSerialization.data(withJSONObject: body)

        do {
            let (data, resp) = try await URLSession.shared.data(for: req)
            guard let http = resp as? HTTPURLResponse else {
                error = "No response"; return
            }
            if (200..<300).contains(http.statusCode) {
                // expecting { "token": "..." }
                let obj = (try? JSONSerialization.jsonObject(with: data)) as? [String: Any]
                let token = obj?["token"] as? String
                if let token, !token.isEmpty {
                    Secrets.authToken = token
```

```
                    UserDefaults.standard.set(token, forKey: tokenKey)
                    isAuthenticated = true
                } else {
                    error = "Token missing"
                }
            } else {
                let msg = String(data: data, encoding: .utf8) ?? "HTTP \(http.statusCode)"
                error = msg
            }
        } catch {
            self.error = error.localizedDescription
        }
    }
}
```

2) LoginView.swift

```
// Path: JarvisClient/LoginView.swift
import SwiftUI

struct LoginView: View {
    @ObservedObject var auth: AuthViewModel
    var onDone: () -> Void

    var body: some View {
        NavigationStack {
            Form {
                Section("Account") {
                    TextField("Email", text: $auth.email)
                        .keyboardType(.emailAddress)
                        .textInputAutocapitalization(.never)
                        .autocorrectionDisabled()
                    SecureField("Password", text: $auth.password)
                }
                if let e = auth.error {
                    Text(e).foregroundStyle(.red).font(.footnote)
                }
                Section {
                    Button("Sign In") { Task { await auth.login(); if auth.isAuthenticated { onDone() } } }
                        .buttonStyle(.borderedProminent)
                    Button("Create Account") { Task { await auth.register(); if auth.isAuthenticated { onDon
                }
            }
            .navigationTitle("Sign In")
        }
    }
}
```

3) Secrets.swift additions

```
// Add these to your existing Secrets.swift

// mutable at runtime to store the login token
static var authToken: String? = nil

// helper for adding Bearer automatically
static func authorizedRequest(_ url: URL, method: String = "GET") -> URLRequest {
    var req = URLRequest(url: url)
    req.httpMethod = method
    if let t = authToken, !t.isEmpty {
        req.setValue("Bearer \(t)", forHTTPHeaderField: "Authorization")
    } else {
        // fall back to backendBearer if present
```

```
        if !backendBearer.isEmpty {
            req.setValue("Bearer \(backendBearer)", forHTTPHeaderField: "Authorization")
        }
    }
    return req
}
```

4) Presenting Login on launch

```
// In JarvisClientApp.swift
@main
struct JarvisClientApp: App {
    @StateObject private var auth = AuthViewModel()
    var body: some Scene {
        WindowGroup {
            RootView().environmentObject(auth)
        }
    }
}

// RootView.swift
import SwiftUI
struct RootView: View {
    @EnvironmentObject var auth: AuthViewModel
    @State private var showLogin = false

    var body: some View {
        ContentView()
            .sheet(isPresented: $showLogin) {
                LoginView(auth: auth) { showLogin = false }
            }
            .onAppear { showLogin = !auth.isAuthenticated }
    }
}
```

Notes

• The server must expose /auth/register and /auth/login returning a JSON token. • If you prefer only bearer via APP_BACKEND_BEARER, you can keep LoginView hidden.