PROJECT SCOPE DOCUMENT

Prosopo: AI Persona Manager

BUILT FOR	**RMIT iOS App Development coursework, built individually as a solo project**
STATUS	Core app shipped, widget in progress
STACK	Swift, SwiftUI, MVVM, SwiftData, Firebase Auth, Firestore, OpenAI API, WidgetKit, App Groups
SOURCE	Personal project, available to walk through on request

1. Overview

Prosopo lets users define reusable AI personas, a name, role, tone, and guardrails, and turns them into clean, consistent system prompts they can drop into ChatGPT, Claude, Gemini, or any other AI assistant. Rather than rewriting the same instructions every time, users build a small library of personas and reuse them across conversations.

The app is built entirely solo in SwiftUI using MVVM architecture, with two tier local and cloud persistence, Firebase Auth for sign in, and a WidgetKit home screen widget for quick access to favourite personas. It was built as an individual project for RMIT iOS App Development coursework.

2. Problem Statement

Anyone who uses AI assistants regularly runs into the same friction: rewriting the same system prompt from scratch every time they want a consistent persona, whether that is a blunt editor, a patient tutor, or a terse code reviewer. Without a reusable library, that phrasing either gets retyped, half remembered, or lost between sessions.

The result is inconsistent AI behaviour across sessions. The same intended persona comes out slightly differently each time it is described from memory, and there is no single place to refine a persona once and reuse it everywhere. Prosopo exists to give that persona definition a permanent, structured home.

3. In Scope

- Persona CRUD: create, edit, and delete personas with name, role, tone, and guardrails
- Tone and consistency controls, mapped to an underlying LLM temperature
- AI assisted persona suggestion from free text, using structured JSON output from OpenAI
- Firebase Auth, with Sign in with Apple and Google Sign In
- Two tier persistence: SwiftData for local drafts, Firestore for synced, finished personas

- WidgetKit home screen extension for quick access to favourite personas
- Unit tests covering the add persona, edit persona, and home list view models

4. Out of Scope

- Android version
- Web app
- Direct LLM chat within the app: Prosopo generates prompts for use elsewhere rather than hosting conversations itself
- Team or shared persona libraries
- App Store distribution, pending Apple Developer Program enrolment

5. Key Architecture Decisions

Two tier persistence: SwiftData local and Firestore cloud

Drafts live locally in SwiftData so an in progress persona survives an app restart before the user commits to it. Finished personas sync to Firestore, scoped per authenticated user. This separation means the app is fully usable offline for persona management, and cloud sync only happens on deliberate save actions.

Computing temperature from tone and a consistency slider

Each persona has a tone, precise, supportive, constructive, concise, assertive, or friendly, each mapped to a base LLM temperature from 0.1 to 0.6. Rather than exposing raw temperature to the user, the UI exposes a single consistency slider from 0 to 1. The actual temperature sent to OpenAI is computed as the base value plus the consistency offset from 0.5, scaled by 0.4, then clamped to the 0 to 1 range. This keeps the control intuitive for non technical users while still giving fine grained influence over response determinism.

Structured JSON output for persona suggestion

One feature lets a user describe a persona in plain language and have the app fill in the form fields automatically. Rather than treating this as a simple chat completion, the request instructs OpenAI to return strict JSON matching the persona schema, including a closed set of valid tone values, which is decoded directly into a Persona model. Malformed or out of range values are caught and clamped rather than allowed to corrupt the model state.

App Group UserDefaults for WidgetKit favourites

Favourites are stored in shared App Group UserDefaults specifically so the WidgetKit extension can read them without a network round trip. A widget that required a network call on every timeline reload would be slow and battery inefficient. App Group storage gives the widget instant read access to the same data the main app writes.

Gesture disambiguation on persona tiles

Persona tiles support four interactions on the same tap target: single tap to view, double tap to edit, triple tap to delete with confirmation, and long press for a quick action menu. The implementation debounces the single tap behind a cancellable DispatchWorkItem and uses simultaneousGesture for the long press, so a double or triple tap cancels the pending single tap action before it fires instead of running both handlers.

6. Outcomes

- Core app shipped and fully functional: persona creation, editing, deletion, tone and consistency controls, structured AI suggestion, Firebase Auth with Sign in with Apple and Google Sign In
- Two tier persistence working: SwiftData local drafts and Firestore cloud sync per user
- WidgetKit extension built and working in local testing: reads favourite persona from App Group, copies prompt to clipboard via a prosopo:// deep link, timeline reloads on save
- Full App Group entitlements blocked on Apple Developer Program enrolment, not available during coursework, so widget distribution is not yet complete
- Unit tests covering add persona, edit persona, and home list view models

7. Integration Footprint

Service	Role	Auth method
Firebase Auth	Sign in with Apple and Google Sign In	Apple nonce + SHA256, Google Sign In SDK
Firestore	Cloud persistence for finished personas per user	Firebase Auth UID scoping
OpenAI API	Persona suggestion (structured JSON) and prompt generation	API key, server side in production
SwiftData	Local persistence for drafts	No external auth, on device only
WidgetKit	Home screen widget for quick persona access	App Group entitlement
App Groups	Shared UserDefaults between main app and widget	Provisioning profile entitlement

8. My Role

Sole developer on this project, covering:

- Overall architecture and MVVM structure
- SwiftUI implementation for every screen and interaction
- Firebase integration, including Sign in with Apple, Google Sign In, and Firestore
- OpenAI integration for persona suggestion and prompt generation
- SwiftData and Firestore persistence design
- WidgetKit extension
- Unit tests

9. What is Next

Apple Developer Program enrolment is the next concrete step. It unlocks the full App Group entitlements the WidgetKit extension needs for distribution, and is a prerequisite for App Store submission. Once that is in place, the remaining widget work can be completed and the app can move toward a public release. Longer term, an Android or web version is a possible expansion, once the core experience is proven on iOS.

Personal project, available to walk through on request. Akash Ramesh is available to walk through implementation in detail during any interview or technical review.