ARCHITECTURE DOCUMENT

Prosopo: AI Persona Manager

BUILT FOR	**RMIT iOS App Development coursework, built individually as a solo project**
STACK	Swift, SwiftUI, MVVM, SwiftData, Firebase Auth, Firestore, OpenAI API, WidgetKit, App Groups
AUTHOR	Akash Ramesh
STATUS	Core app shipped, widget in progress

1. System Overview

Prosopo is a SwiftUI app built with MVVM architecture, using a two tier persistence model: SwiftData for local drafts and Firestore for synced, finished personas, scoped per authenticated user. Firebase Auth handles sign in through Sign in with Apple and Google Sign In.

A WidgetKit extension reads favourite personas from shared App Group storage for home screen quick access. The OpenAI API powers structured persona suggestion and prompt generation, with responses decoded directly into the app's Persona model.

2. High Level Architecture

Component map

Layer	Component	Responsibility
UI	SwiftUI Views	Persona list, editor, suggestion flow, gesture handling
State	MVVM View Models	Business logic, temperature computation, persistence coordination
Local persistence	SwiftData	Draft personas, on device only
Cloud persistence	Firestore	Finished personas, scoped per Firebase Auth UID
Auth	Firebase Auth	Sign in with Apple, Google Sign In
AI	OpenAI API	Persona suggestion (structured JSON), prompt generation
Widget	WidgetKit, App Groups	Home screen favourite persona display and quick copy

Persona save and suggestion flows

PERSONA SAVE FLOW

PERSONA SUGGESTION FLOW

```
User
  |
  v
SwiftUI App
  |
  v
SwiftData
(local draft)
  |
  v
Firestore
(on commit,
 per user)
```

```
User
  |
  v
SwiftUI App
  |
  v
OpenAI API
(structured JSON)
  |
  v
Persona model
(decoded, clamped)
```

Widget flow

```
Main App
  |
  v
App Group UserDefaults
(shared, on device)
  |
  v
WidgetKit Extension
(home screen)
```

3. Persistence Architecture

Persona data moves through two persistence layers. SwiftData holds drafts locally, on device, so an in progress persona survives an app restart before the user commits to it. There is no network dependency at this stage, and the app remains fully usable offline for creating and editing personas.

Once a persona is finished and saved, it syncs to Firestore, scoped to the authenticated user's UID. This is the point where cloud sync happens, deliberately, rather than on every keystroke or field change. If the device is offline at save time, the write queues through Firestore's own offline persistence and syncs once connectivity returns.

4. Authentication Design

Sign in with Apple uses a nonce and SHA256 challenge: the app generates a random nonce, hashes it with SHA256, and sends the hash to Apple as part of the authorization request. Apple returns a signed credential that Firebase Auth verifies against the original nonce, which prevents replay of a captured credential.

Google Sign In runs through the Google Sign In SDK, which returns an ID token that Firebase Auth exchanges for a Firebase credential.

Both providers resolve to the same Firebase Auth user model. Firestore reads and writes are scoped to that user's UID, so personas are isolated per account regardless of which provider the user signed in with.

5. Persona Suggestion Pipeline

A user can describe a persona in plain language, for example a blunt but fair code reviewer, and have the app fill in the form fields automatically. That free text is sent to the OpenAI API with a system prompt that instructs the model to return strict JSON matching the Persona schema, including a closed set of valid tone values.

The response is decoded directly into a Persona model. Any field that falls outside the expected range or tone set is caught during decoding and clamped to the nearest valid value, rather than allowed to produce an invalid or corrupted Persona instance. This keeps the suggestion feature from ever putting the app into a bad state, even if the model returns an unexpected value.

6. Temperature Computation Model

Each persona has one of six tones: precise, supportive, constructive, concise, assertive, or friendly. Each tone maps to a base temperature between 0.1 and 0.6, with more deterministic tones sitting toward the lower end of that range and more expressive tones sitting toward the higher end.

Rather than exposing that raw temperature value to the user, the UI exposes a single consistency slider from 0 to 1. The final temperature sent to OpenAI is computed as:

```
temperature = clamp(base + (consistency - 0.5) * 0.4, 0, 1)
```

A raw temperature value means little to a non technical user. The consistency slider gives the same fine grained influence over response determinism through a single, intuitive control, while the tone selection still does most of the work of setting the persona's baseline behaviour.

7. WidgetKit Architecture

Favourite personas are written to shared UserDefaults in an App Group container, accessible to both the main app and the WidgetKit extension. When a favourite changes, the main app reloads the widget's timeline, so the home screen widget picks up the change without waiting for its next scheduled refresh.

Tapping the widget opens the app through a prosopo:// deep link, which copies the favourite persona's prompt to the clipboard as part of the same interaction.

Reading from Firestore instead would mean a network call on every timeline reload, which is slow and battery inefficient for something that should feel instant. App Group storage gives the widget the same instant, offline read access that SwiftData gives the main app for drafts.

8. Gesture Handling Design

Persona tiles support four interactions on the same tap target: a single tap opens the persona for viewing, a double tap opens it for editing, a triple tap deletes it after confirmation, and a long press opens a quick action menu.

> The risk with stacking multiple tap gesture counts on one view is that a double or triple tap also registers as a single tap on its way there. The implementation debounces the single tap behind a cancellable DispatchWorkItem: the single tap action is scheduled after a short delay, and a subsequent double or triple tap cancels that pending work item before it fires.
>
> The long press is attached with simultaneousGesture, so it can be recognized independently of the tap count gestures rather than competing with them.

9. Security Design

Concern	Approach	Where enforced
User authentication	Firebase Auth via Sign in with Apple and Google Sign In	Firebase Auth
Replay protection	SHA256 hashed nonce challenge for Sign in with Apple	Sign in with Apple flow
Data isolation	Firestore reads and writes scoped to the authenticated user's UID	Firestore
API key handling	OpenAI API key held server side in production, not embedded in the client	Backend, build configuration
Local data	Draft personas stored in on device SwiftData only, no network exposure	SwiftData
Widget data	Favourites shared through an App Group entitlement scoped to this app's group only	App Groups, WidgetKit

10. My Contributions

As the sole developer on this project, my ownership covered:

- Overall architecture and MVVM structure
- SwiftUI implementation for every screen and interaction
- Firebase integration: Sign in with Apple, Google Sign In, Firestore
- OpenAI integration for persona suggestion and prompt generation
- SwiftData and Firestore persistence design
- WidgetKit extension, including App Group configuration
- Unit tests for the add persona, edit persona, and home list view models

---

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