WinZo AI Companion Handover Guide

PlayPal AI | FastAPI + Vite prototype

November 15, 2025

This guide summarizes the state of the WinZo AI Companion MVP located at `/home/vas40/Competitions/GAIM/sahil2`. It covers setup, execution, known gaps, and the key AI prompting strategies so future maintainers can extend the build confidently.

1 Setup Instructions

1.1 Prerequisites

- Python 3.10+ with virtual environment support, Node.js 18+, npm.
- `uvicorn` and `fastapi` are pulled in through `backend/requirements.txt`. Frontend uses Vite + React.
- Optional: Google Gemini access token for higher-fidelity responses.

1.2 Backend (FastAPI)

1. `cd backend`
2. `python -m venv .venv`
   `source .venv/bin/activate`
3. `pip install -r requirements.txt`
4. Export environment variables:
   - `GEMINI_API_KEY=` your key (optional; rule templates handle fallback).
   - `GEMINI_MODEL=` override (defaults to `gemini-1.5-flash`).
5. Run `uvicorn app:app --reload --port 8000`.

1.3 Frontend (Vite + React)

1. `cd frontend`
2. `npm install`
3. Optional `.env`: `VITE_API_BASE_URL=http://localhost:8000`
4. Start dev server: `npm run dev` (served at `http://localhost:3000`).

2 Working Code Repository

Root layout	`backend/` (FastAPI service), `frontend/` (Vite app), `README.md` (quick start).
Backend entry	`backend/app.py` exposes `/chat` POST endpoint that dispatches to `generate_response`.
AI logic	`backend/companion_logic.py` handles prompt templating, Gemini integration, and deterministic fallbacks using `data.py`.
Frontend shell	`frontend/src/App.jsx` wires up surfaces (Lounge, Squad, Matchmaking, Rewards). Each page embeds `ChatBox`.
Reusable UI	`frontend/src/components/ChatBox.jsx` manages persona selectors, fetches the backend, renders Markdown, and surfaces smart nudges.

3 How to Execute & Test

1. Launch backend and frontend using the commands above in two terminals.
2. Navigate to `http://localhost:3000`. Switch between navigation tabs to confirm that the same chat component adapts to each context.
3. Validate backend manually: send a POST to `/chat` with JSON payload `{ "companion": "Veer", "mood": "Pumped", "game": "Arcade", "performance": "WinStreak", "context": "matchmaking" }` using curl or Thunder Client. Expect a JSON reply with `response`, `suggestions`, and `context` keys.
4. Smoke-test suggestions: in the UI select “LoseStreak” + “Shooter” and run “Ask Companion”. Smart nudges should include “Shift to a relaxed lobby” and “Warm-up aim mode” cards.
5. Optional build checks: `npm run build` for the Vite bundle. No automated Python tests exist yet; manual regression revolves around API contract validation above.

4 Known Issues & Troubleshooting

- **Missing Gemini SDK or API key**: the backend gracefully falls back to deterministic templates. If Gemini access is required, install `google-generativeai` (already listed) and ensure the key is exported before starting uvicorn.
- **CORS or network mismatch**: Vite runs on port 3000 with API defaulting to `http://localhost:8000`. Set `VITE_API_BASE_URL` when hosting the backend elsewhere.
- **Stale virtualenv**: deleting `backend/.venv` without deactivating may leave pip pointing at the wrong interpreter. Recreate the env and reinstall requirements.
- **Node dependency drift**: if npm install fails in CI, remove `frontend/node_modules` and re-run with the documented Node 18 baseline.
- **Large language model latency**: Gemini calls are synchronous; long pauses block the API. Consider reducing prompt size or keeping the fallback template enabled for demos.

5 Future Student / Instructor Workflows

- **Extending personas**: add metadata in `backend/data.py` (persona, tone, style, sample lines) and update `ChatBox` select options to expose the new persona in the UI.
- **New surfaces**: replicate `frontend/src/pages/MatchmakingLab.jsx` to create another tab. Pass a new `context` string so backend-specific copy can be tuned via `CONTEXT_HINTS` and suggestion rules.
- **LLM experimentation**: adjust `_build_gemini_prompt` or route to other models by changing `GEMINI_MODEL`. Keep prompts under 120 words as enforced by the current spec.
- **Automated tests**: add FastAPI route tests (e.g., pytest + httpx) to verify fallback responses and suggestion combinations; add React component tests (Vitest) for ChatBox interactions.
- **Demo collateral**: capture screen recordings across the four navigation tabs showing how suggestions adapt, and host them alongside this PDF when submitting future milestones.

6 Prompt Library (Key Prompts)

LLM prompt (Gemini):

```
You are <companion>, the <persona> companion inside WinZo's PlayPal AI.
Stay <tone> with <style> cadence.
Craft a concise coaching message (max 120 words) ...
**<companion> checking in!**
- Mode: ...
- Energy Boost: ...
- Smart Nudge: ...
- Game/Reward Insight: ...
Keep emojis tasteful (max 2).
```

This lives in `backend/companion_logic.py::_build_gemini_prompt` and is fed into Google Gemini when API access is configured.

Rule-based template: When Gemini is absent, the system stitches together persona examples, mood/performance/game mappings, and context-specific headers inside `_template_response`. This ensures deterministic replies for demos.

Suggestion rules: `_build_suggestions` links user state to UI nudges (e.g., LoseStreak triggers “Shift to a relaxed lobby” + drill suggestions, “Shooter” mood adds warm-up guidance, “matchmaking” adds auto-match). Reuse these patterns when designing new automations.

Prepared for project handover – PDF generated via XeLaTeX.