🚐Wheels & Wins – Pam 2.0 Build Playbook (Gemini-First Edition)

This playbook is a **step-by-step execution guide** for building Pam 2.0. Follow each phase in order. Each phase contains prompts ready to paste into your coding AI. **Do not skip steps.**

Phase 0 – Rules of the Build

1. Always work in the `pam-2.0` branch.
2. Keep existing PAM frontend; only rebuild backend.
3. Build in **staging first**, never overwrite production.
4. After each phase → run tests.
5. Each module must be **<300 lines**, simple, modular.
6. Supabase schema: add **only what's needed**.

Phase 1 – Setup & Scaffolding

Prompt 1.1 – Repo Setup

```
Create a new branch `pam-2.0`. Keep the existing PAM frontend code. Wipe the
old PAM backend. Scaffold a new FastAPI app with:
- `/chat` WebSocket + REST endpoint
- Supabase client setup (env vars for keys)
- Basic health check route
- CI/CD config for staging (Render backend, Netlify frontend)
```

Phase 2 – Conversational Engine

Prompt 2.1 – Gemini-First Engine

```
Build a FastAPI service with:
- `/chat` endpoint (WebSocket + REST)
- Request format: { user_id, message, context }
- Send to Gemini API (primary)
- Return: { response, ui_action?, metadata? }
- Log into Supabase (pam_messages)
- Keep <300 lines, fully async, error-handled
```

Phase 3 – Context Manager

Prompt 3.1 – Context Middleware

```
Add a ContextManager class that:
- Loads user profile from Supabase (profiles table)
- Merges context: vehicle, budget, preferences
- Passes this into Gemini requests
- Caches context per session to reduce DB hits
```

Phase 4 – Passive Trip Logger (Wheels)

Prompt 4.1 – Trip Logging Service

```
Create a TripLogger module that:
- Listens to location pings (periodic updates)
- Detects overnight stops (12+ hrs in one place)
- Saves to Supabase `trips` table (user_id, start, end, route, stops)
- Runs as a background task, no user input required
```

Phase 5 – Savings Tracker (Wins)

Prompt 5.1 – Savings Guarantee

```
Add a SavingsTracker module that:
- Reads user expenses from `expenses` table
- Calculates: total_saved = (discounts + optimized choices)
- Compares to subscription price ($14.99)
- If savings < sub price → mark free month in pam_savings table
```

Phase 6 – Safety Layer

Prompt 6.1 – PamGuardian

```
Create a filter middleware PamGuardian:
- Intercepts AI responses before sending to user
- If medical/emergency → respond with: ⚠️ Call 000 immediately
- Else → pass AI response through
- Log all flagged events in Supabase safety_events
```

Phase 7 – Testing

Prompt 7.1 – Unit Tests

```
Generate pytest tests for each module:
- Conversational engine → mock Gemini response
- Context manager → mock Supabase profile
- Trip logger → simulate GPS pings
- Savings tracker → mock expenses
- Safety layer → test emergency queries
- Ensure <5s runtime for test suite
```

Phase 8 – Deployment

Prompt 8.1 – Deploy Staging

```
Deploy backend (FastAPI) to Render under pam-2.0-staging.
Deploy frontend (Netlify) pointing to staging backend.
Verify health check, WebSocket connection, Supabase writes.
Do NOT replace production — staging only.
```

Supabase Schema Changes

Apply only once, after Phase 4–6.

```sql
-- Trips Table
CREATE TABLE trips (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  user_id UUID REFERENCES auth.users(id) ON DELETE CASCADE,
  start TIMESTAMPTZ,
  end TIMESTAMPTZ,
  route JSONB,
  stops JSONB,
  created_at TIMESTAMPTZ DEFAULT NOW()
);

-- Savings Tracker
CREATE TABLE pam_savings (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  user_id UUID REFERENCES auth.users(id) ON DELETE CASCADE,
  month DATE,
  total_saved NUMERIC,
  free_month BOOLEAN DEFAULT FALSE,
  created_at TIMESTAMPTZ DEFAULT NOW()
```

```
);

-- Safety Events
CREATE TABLE safety_events (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  user_id UUID REFERENCES auth.users(id) ON DELETE CASCADE,
  event_type TEXT,
  details JSONB,
  created_at TIMESTAMPTZ DEFAULT NOW()
);
```

End State

- Pam 2.0 runs on staging (`pam-2.0-staging`).
- Gemini is the default AI brain.
- Frontend PAM stays intact.
- Backend is modular, clean, and future-proof.
- New modules can be added/tested without breaking production.
