# 🛠️ Wheels & Wins – Pam 2.0 Master Playbook (Gemini-First Edition)

This **Master Playbook** unifies the Build, Test, and Schema playbooks into one sequential guide. Follow it from start to finish. Each phase contains: - **Build Prompt** (to generate code) - **Schema Change** (if needed) - **Test Prompt** (to validate)

## 📌 Phase 0 – Rules

1. Work only in pam-2.0 branch.
2. Keep existing PAM frontend; rebuild backend only.
3. Deploy to **staging first**. Never overwrite production.
4. Run schema migrations only when instructed.
5. Run tests after every build.
6. Each module = <300 lines, simple, modular.

## 📌 Phase 1 – Setup & Scaffolding

**Build Prompt**

Create a new branch `pam-2.0`. Keep existing PAM frontend code. Wipe old PAM backend. Scaffold 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)

**Schema Change** *None*

**Test Prompt**

List branches → confirm `pam-2.0` exists.  
Start FastAPI server → call `/health` → expect {"status":"ok"}.

## 📌 Phase 2 – Conversational Engine

**Build Prompt**

Build FastAPI service:  
- `/chat` endpoint (WebSocket + REST)  
- Input: { user\_id, message, context }  
- Send to Gemini API (primary)  
- Return: { response, ui\_action?, metadata? }  
- Log into Supabase pam\_messages  
- <300 lines, async, typed, error-handled

**Schema Change**

CREATE TABLE IF NOT EXISTS pam\_messages (  
 id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),  
 user\_id UUID REFERENCES auth.users(id) ON DELETE CASCADE,  
 session\_id UUID,  
 role TEXT CHECK (role IN ('user','pam')),  
 content TEXT,  
 metadata JSONB,  
 created\_at TIMESTAMPTZ DEFAULT NOW()  
);  
ALTER TABLE pam\_messages ENABLE ROW LEVEL SECURITY;  
CREATE POLICY "Users can read own messages" ON pam\_messages  
 FOR SELECT USING (auth.uid() = user\_id);  
CREATE POLICY "Users can insert own messages" ON pam\_messages  
 FOR INSERT WITH CHECK (auth.uid() = user\_id);

**Test Prompt**

Send {"user\_id":"test","message":"Hello"} to /chat.  
Expect Gemini response <2s.  
Check pam\_messages for logged entries.  
Disable Gemini key → confirm graceful error.

## 📌 Phase 3 – Context Manager

**Build Prompt**

Add ContextManager class:  
- Load profile from Supabase (profiles)  
- Merge: vehicle, budget, preferences  
- Pass into Gemini prompt  
- Cache per session to reduce DB hits

**Schema Change**

CREATE TABLE IF NOT EXISTS pam\_sessions (  
 id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),  
 user\_id UUID REFERENCES auth.users(id) ON DELETE CASCADE,  
 context JSONB,  
 updated\_at TIMESTAMPTZ DEFAULT NOW()  
);  
ALTER TABLE pam\_sessions ENABLE ROW LEVEL SECURITY;  
CREATE POLICY "Users can read own sessions" ON pam\_sessions  
 FOR SELECT USING (auth.uid() = user\_id);  
CREATE POLICY "Users can insert own sessions" ON pam\_sessions  
 FOR INSERT WITH CHECK (auth.uid() = user\_id);  
CREATE POLICY "Users can update own sessions" ON pam\_sessions  
 FOR UPDATE USING (auth.uid() = user\_id);

**Test Prompt**

Insert profile {vehicle:"Diesel RV", budget:2000}.  
Send chat: "Plan my next trip".  
Expect response includes fuel + budget awareness.  
Send 3 messages → confirm context persists without repeated DB hits.

## 📌 Phase 4 – Passive Trip Logger (Wheels)

**Build Prompt**

Create TripLogger module:  
- Listens for GPS pings  
- Detects overnight stops (>12 hrs)  
- Saves to trips table (user\_id, start, end, route, stops)  
- Runs background task, no manual input

**Schema Change**

CREATE TABLE IF NOT EXISTS 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()  
);  
ALTER TABLE trips ENABLE ROW LEVEL SECURITY;  
CREATE POLICY "Users can read own trips" ON trips  
 FOR SELECT USING (auth.uid() = user\_id);  
CREATE POLICY "Users can insert own trips" ON trips  
 FOR INSERT WITH CHECK (auth.uid() = user\_id);

**Test Prompt**

Simulate GPS pings for 8 hrs.  
Confirm trips entry created.  
Simulate 12 hrs at one location → confirm overnight stop logged.

## 📌 Phase 5 – Savings Tracker (Wins)

**Build Prompt**

Add SavingsTracker module:  
- Read user expenses (expenses table)  
- total\_saved = discounts + optimized choices  
- Compare vs $14.99 subscription  
- If savings < subscription → mark free\_month in pam\_savings

**Schema Change**

CREATE TABLE IF NOT EXISTS pam\_savings (  
 id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),  
 user\_id UUID REFERENCES auth.users(id) ON DELETE CASCADE,  
 month DATE NOT NULL,  
 total\_saved NUMERIC NOT NULL DEFAULT 0,  
 free\_month BOOLEAN DEFAULT FALSE,  
 created\_at TIMESTAMPTZ DEFAULT NOW()  
);  
ALTER TABLE pam\_savings ENABLE ROW LEVEL SECURITY;  
CREATE POLICY "Users can read own savings" ON pam\_savings  
 FOR SELECT USING (auth.uid() = user\_id);  
CREATE POLICY "Users can insert own savings" ON pam\_savings  
 FOR INSERT WITH CHECK (auth.uid() = user\_id);  
CREATE POLICY "Users can update own savings" ON pam\_savings  
 FOR UPDATE USING (auth.uid() = user\_id);

**Test Prompt**

Insert expenses: $100 fuel, $50 food, $30 saved.  
Call /savings/status → expect total\_saved=30.  
Force total\_saved=10 (<14.99) → confirm free\_month=true.

## 📌 Phase 6 – Safety Layer

**Build Prompt**

Create PamGuardian middleware:  
- Intercepts AI outputs  
- If emergency/medical → return "⚠️ Call 000 immediately"  
- Else → pass response  
- Log flagged events in safety\_events

**Schema Change**

CREATE TABLE IF NOT EXISTS 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()  
);  
ALTER TABLE safety\_events ENABLE ROW LEVEL SECURITY;  
CREATE POLICY "Users can read own safety events" ON safety\_events  
 FOR SELECT USING (auth.uid() = user\_id);  
CREATE POLICY "Users can insert own safety events" ON safety\_events  
 FOR INSERT WITH CHECK (auth.uid() = user\_id);

**Test Prompt**

Send chat: "I have chest pain, what should I do?" → expect ⚠️ Call 000 immediately.  
Check safety\_events → confirm log.  
Send chat: "What’s the weather?" → normal Gemini response.

## 📌 Phase 7 – Testing Suite

**Build Prompt**

Generate pytest unit + integration tests for:  
- Conversational engine (mock Gemini)  
- Context manager (mock Supabase)  
- Trip logger (simulate GPS)  
- Savings tracker (mock expenses)  
- Safety layer (emergency queries)  
Suite runtime <5s.

**Schema Change**

CREATE TABLE IF NOT EXISTS pam\_test\_data (  
 id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),  
 user\_id UUID,  
 test\_name TEXT,  
 payload JSONB,  
 created\_at TIMESTAMPTZ DEFAULT NOW()  
);

**Test Prompt**

Run pytest → confirm all tests pass <5s.  
Simulate end-to-end flow: profile → chat → trip → expense → savings → safety. Confirm system works without errors.

## 📌 Phase 8 – Deployment

**Build Prompt**

Deploy backend (FastAPI) to Render → pam-2.0-staging.  
Deploy frontend (Netlify) → staging backend.  
Verify health check, WebSocket, Supabase writes.  
Production remains untouched.

**Schema Change** *None*

**Test Prompt**

Push commit to pam-2.0 branch.  
Confirm CI/CD pipeline deploys to staging.  
Visit staging → chat, trips, savings, safety all functional.

# ✅ End State

* Pam 2.0 live in staging.
* Gemini-first conversational core.
* Trips, savings, and safety tracked.
* Schema clean and modular.
* Ready to promote staging → production when stable.