🛠️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

```sql
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

```sql
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.