

Prototype Build & Wireframes

1. ARCHITECTURE DIAGRAM

The diagram shows what talks to what and how a bill photo becomes a fair per-person split. The PhonePe app sends the bill image to an orchestration layer (Make), which calls the AI layer (OCR + an NLU model), runs the allocation engine to distribute GST and service charge proportionally, persists to the database, and returns the split to the app — which fires requests through PhonePe's existing payment rails.

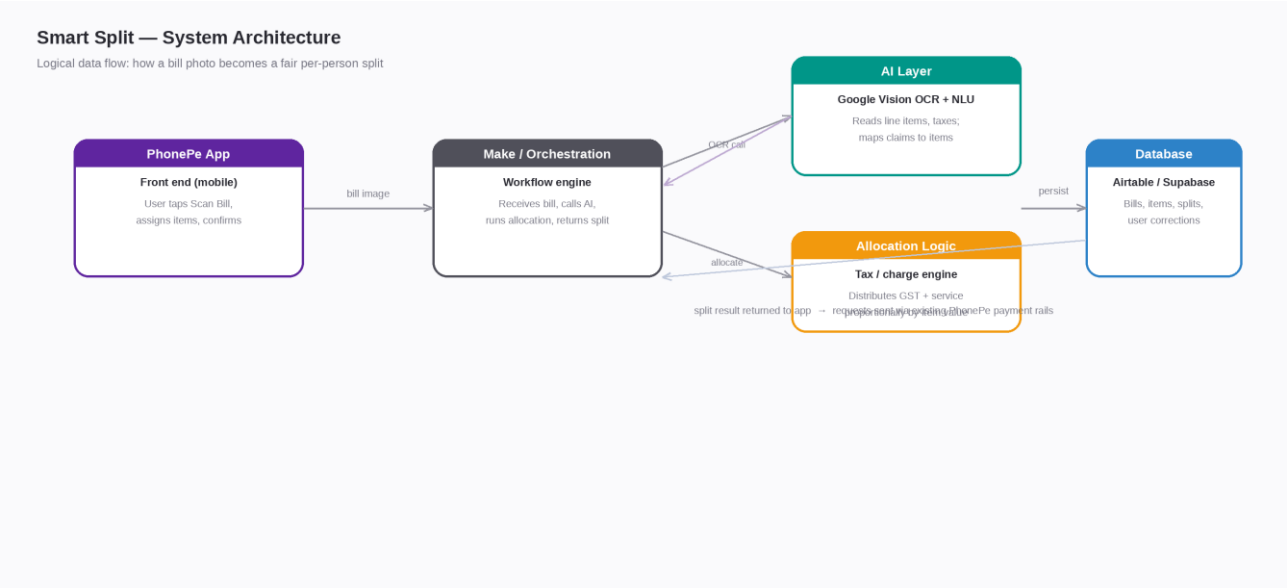


Figure 4.1 — Smart Split logical architecture: App → Orchestration (Make) → AI (OCR + NLU) + Allocation logic → Database, split returned to the app.

Why Make for orchestration Make was chosen because it handles multi-step, branching workflows (OCR → parse → iterate items → allocate → write) at lower cost per operation and has stronger support for HTTP/JSON and image-handling steps. It was chosen hand-rolled backend code for the prototype because it makes the data flow visible and fast to change while validating the concept — production would later harden the hot path into a service, but Make proves it cheaply now.

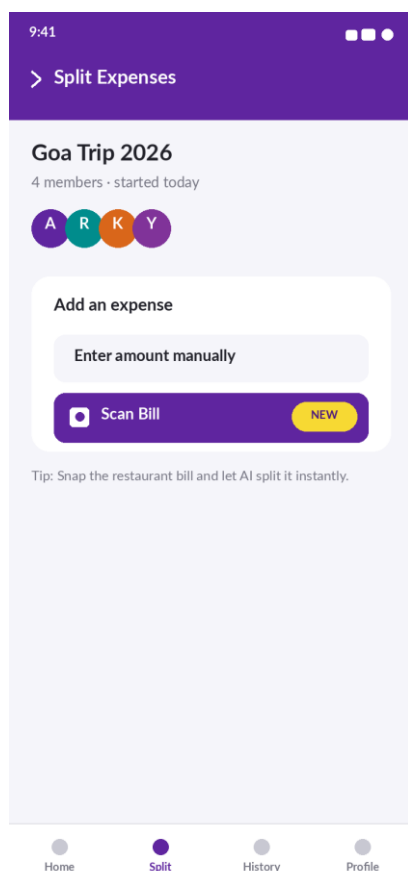
2. TOOL STACK

Tool	Purpose	Free / Paid Tier Used
Figma	Wireframes and high-fidelity prototype screens	Free
Make (Integromat)	Orchestration of the OCR → allocation → split pipeline	Free tier
Google Cloud Vision API	OCR — line-item, tax and total extraction	Paid (per call)
LLM / NLU (e.g., Gemini / GPT)	Natural-language item assignment	Paid (per call)
Airtable / Supabase	Database — bills, items, splits, corrections	Free tier

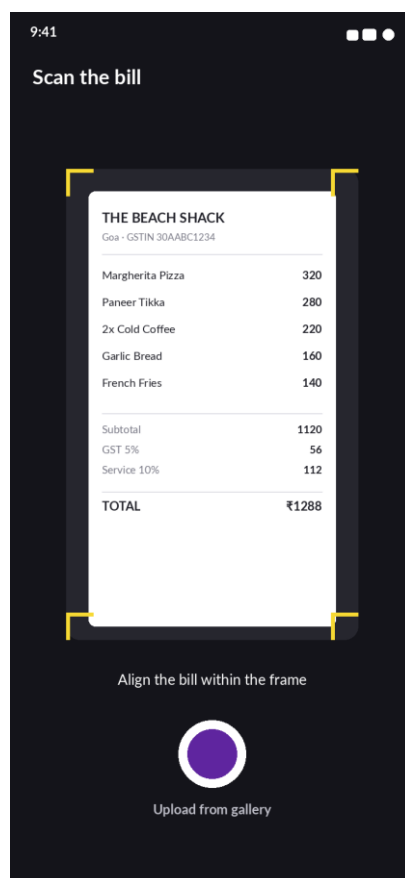
PhonePe payment rails (existing)	Sends split requests through the current flow	Internal
----------------------------------	---	----------

3. BUILD EVIDENCE (SCREENSHOTS)

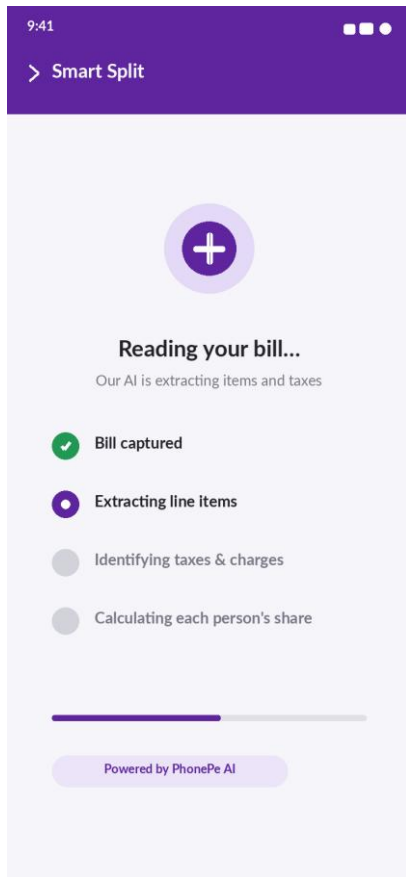
A working, clickable prototype of the Smart Split flow was built in Figma (file: figma.com/design/6mbd1dPg9JKjMEIXKFrDas). Every required touchpoint is captured and captioned below, including the OCR-uncertainty state where the AI is honest about what it could not read.



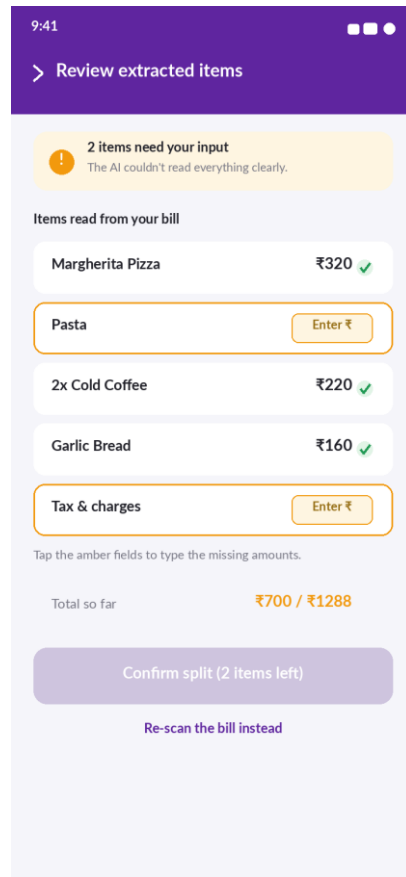
Screen 1 — Entry point. The new 'Scan Bill (AI)' option appears at the expense-entry step, beside manual entry.



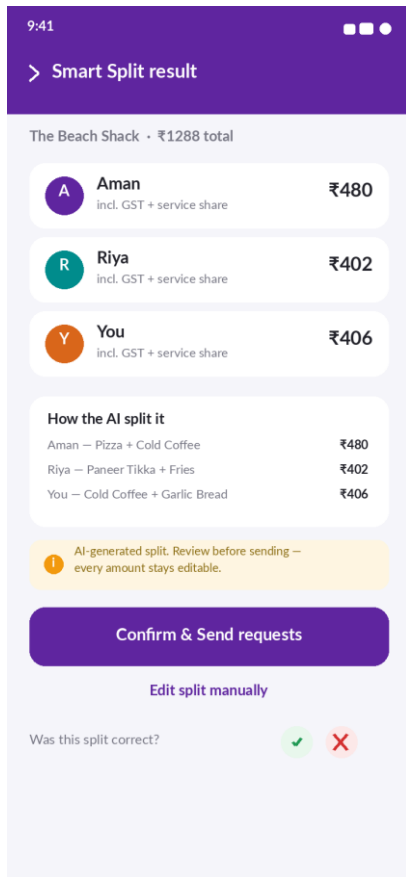
Screen 2 — Input. The user photographs the bill; corner guides help align the full receipt in frame.



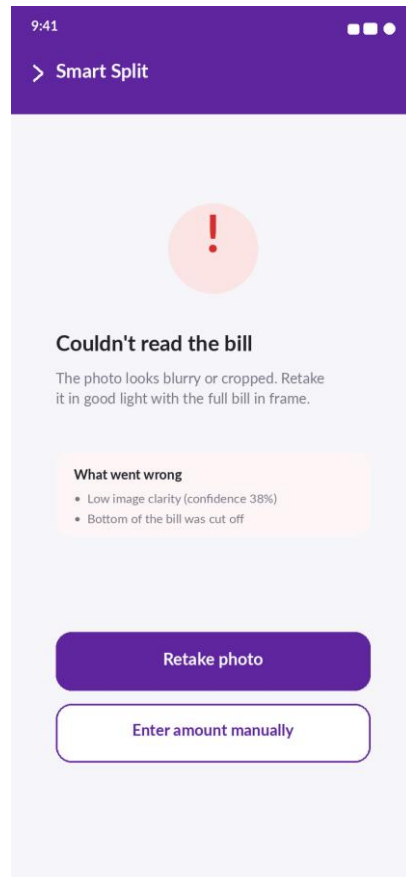
Screen 3 — AI processing. OCR extracts items while step indicators show progress.



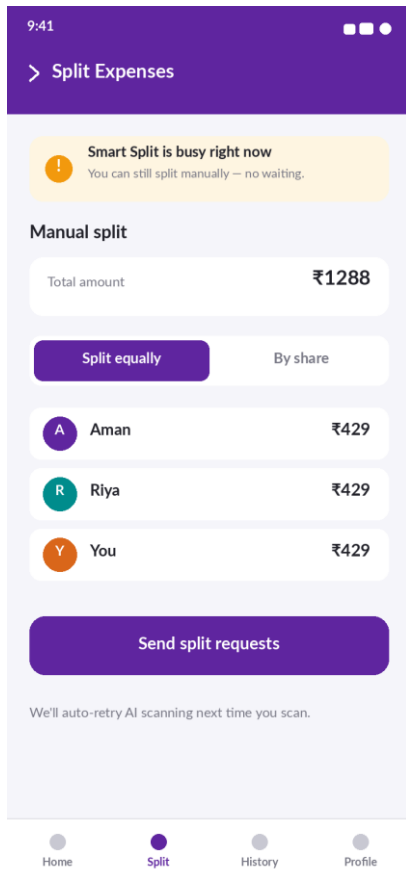
Screen 4 — OCR uncertainty. When the AI can't read a line, it flags it ('Pasta ???') and asks the user to enter it, rather than guessing. Confirm stays disabled until totals reconcile.



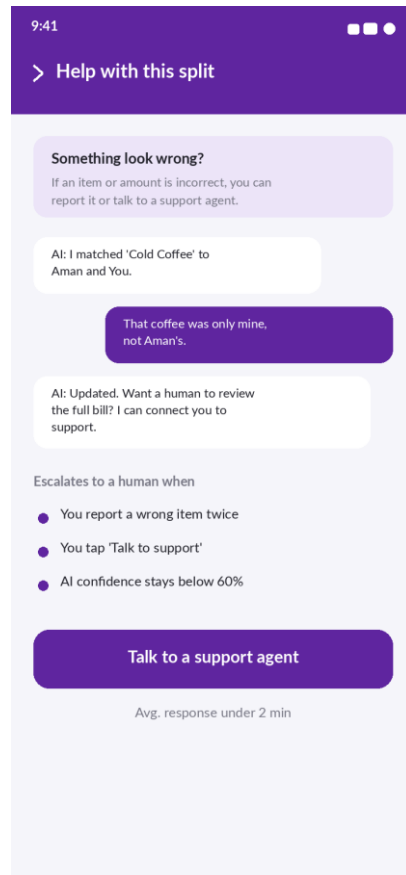
Screen 5 — AI output. Each member's exact share with GST + service distributed; itemised breakdown, AI disclaimer and thumbs feedback. Amounts stay editable.



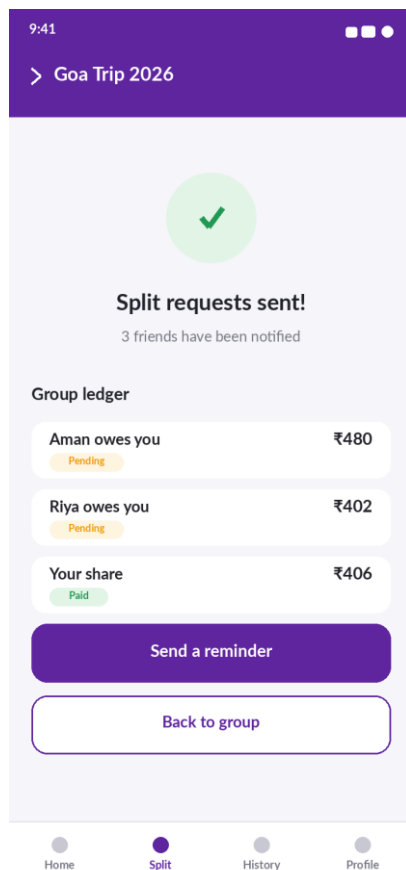
Screen 6 — Error scenario. A blurry/cropped photo is rejected with the reason (confidence 38%) and clear recovery actions.



Screen 7 — Fallback. When the AI is down, the user splits manually with no waiting; AI auto-retries next time.



Screen 8 — Human-in-the-loop. The user corrects a misread item; defined triggers escalate to a support agent.



Screen 9 — Confirmation & back navigation. Requests are sent; the ledger tracks pending/paid and 'Back to group' returns the user.

Design decisions — why each screen exists

- **Entry point (1):** Scan Bill is placed exactly where the friction lives (the division step) so AI replaces manual math at the moment of pain, while manual entry stays one tap away for users who prefer it.
- **OCR uncertainty (4):** This is the most important screen. Users need to trust the AI, so when it is unsure it says so and asks rather than guessing — visible honesty is what earns trust in a money feature.
- **AI output (5):** Item-level transparency is shown before any payment request is sent, because receivers pay faster when they can see what they are paying for.
- **Fallback & human-in-loop (7, 8):** The flow must never trap the user: an outage drops to manual split with no wait, and clear triggers route disputes to a human instead of a dead end.

Backend wiring

The orchestration is a live Make scenario (#6109702) in the team workspace. An instant webhook fires on bill upload; the flow calls OCR over HTTP, parses the JSON, iterates each line item, and creates a split request per member.



Figure 4.3 — Backend: Make scenario #6109702. Webhook → HTTP OCR → JSON parse → Iterator (per item) → HTTP create split request.

4. USER FLOW DIAGRAM

The updated journey after Smart Split is introduced. The happy path runs from opening the group to sending requests; exception paths (unreadable bill, AI unavailable, low-confidence / disputed item) branch to retake, manual fallback and human support.

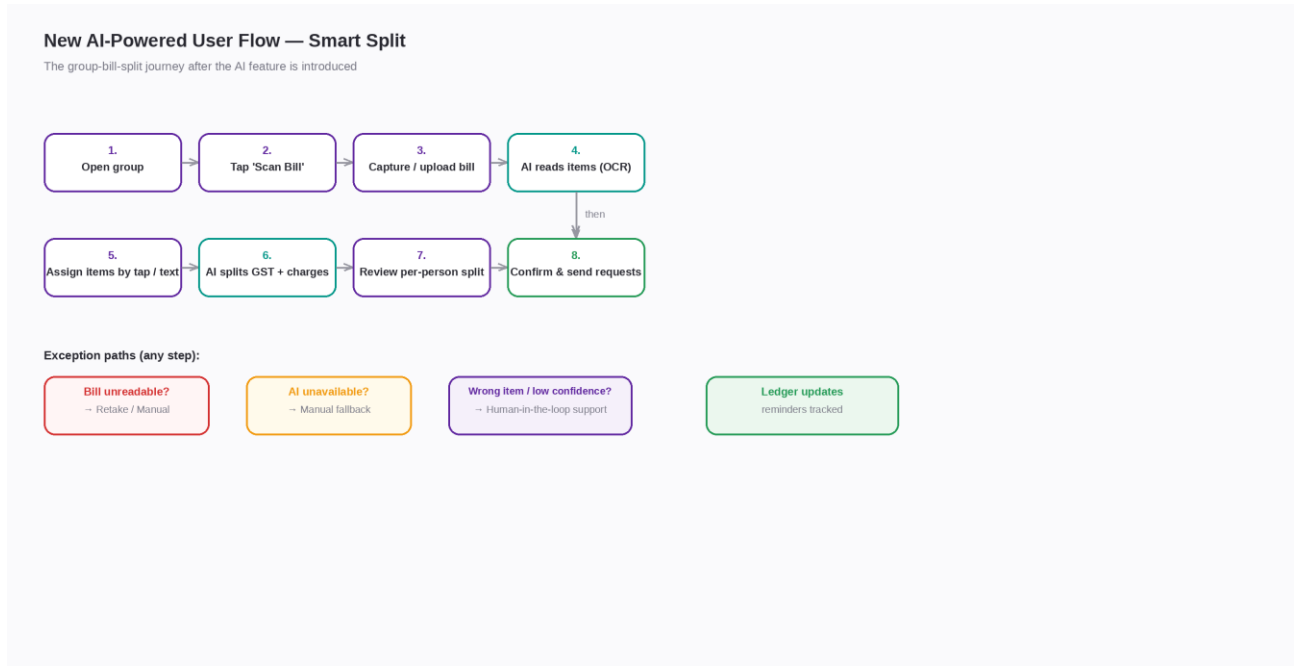


Figure 4.4 — New Smart Split user flow with happy path and exception branches.

5. WIREFRAMES

Low-fidelity wireframes for the three most important screens — scan entry, AI split result, and the error/fallback state — capturing structure and layout intent before visual design.

Low-Fidelity Wireframes — Key Screens

Structure & layout intent before visual design

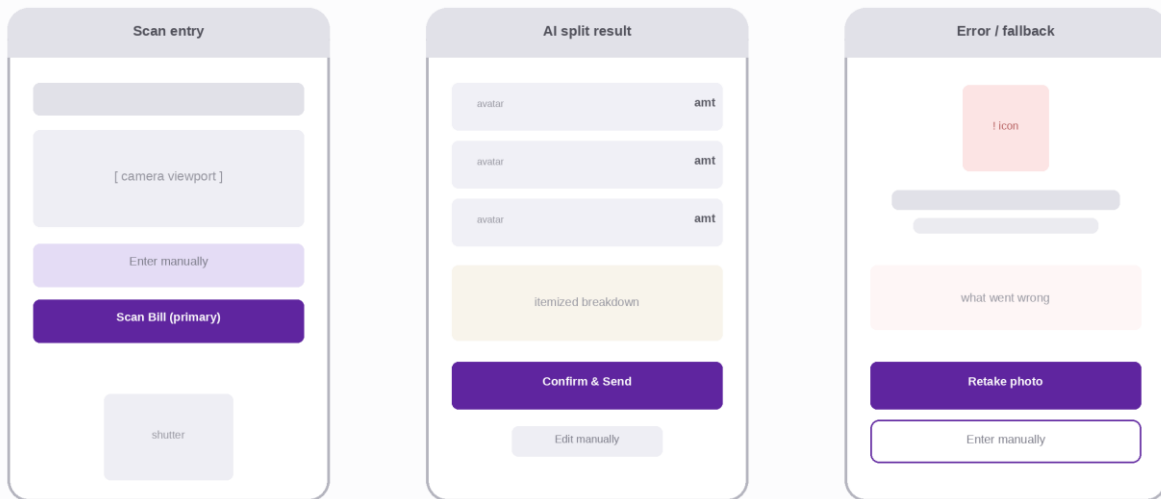


Figure 4.5 — Low-fidelity wireframes: scan entry, AI split result, error/fallback.

6. UX RATIONALE

- **Why the flow is designed this way:** Scan Bill is placed at the division step so the AI replaces manual math at the exact moment of pain, while manual entry stays available for those who want it.
- **Handling errors and fallback:** The design never traps the user: a blurry image gives a specific reason and two recovery actions; an AI outage drops to manual split with no waiting; an unreadable line is surfaced honestly rather than guessed.
- **Human-in-the-loop in the UX:** Correction is built into the result screen (every amount editable, thumbs feedback), and named triggers — two reported errors, a support button, or sub-60% confidence — make escalation predictable, not a dead end.
- **Helping users trust the AI:** Trust is earned through visible itemisation, an explicit AI disclaimer, confidence flags on uncertain items, and a hard guardrail that blocks any split whose parts don't add up to the total.