GiftCare – Architecture, UX, and PO Handoff (v1)

Owner: Orchestrator
Sequence per request: 1) Architect → 2) UX → 3) PO
Context: Complements the PRD already drafted

---

1) Architect – System Design & API Contracts

1.1 Reference Architecture (MVP-first)

- **Clients**: Web app (SPA) and mobile (later), using OAuth 2.1 / OIDC for auth.
- **API Gateway**: AuthN/Z, rate-limits, request signing, idempotency.
- **Core Services**:
- **User Service**: profiles, timezone, contact verification.
- **Recipient Service**: recipients, addresses, delivery channels.
- **Calendar Service**: Google OAuth, import, dedupe, in-app events.
- **Catalog Service**: gift levels → vendor SKUs, availability, pricing bands.
- **Policy Engine**: mandate caps vs. expected amount; decide proceed / rail switch / SKU swap / skip.
- **Payments Service**: mandate onboarding, pre-debit notifications, charge execution, reconciliation hooks.
- **Fulfilment Service**: digital e-code issuance, delivery via channels; physical adapter (P1).
- **Notification Service**: templates, channels (Email/SMS/WhatsApp), pre-debit and delivery messages.
- **Scheduler & Orchestrator**: daily job, lead-time evaluation, retries, DLQ.
- **Audit & Events**: immutable logs, event bus (Kafka/NATS) for decoupling.
- **External Integrations**: Google Calendar, Payment Gateway (UPI Autopay, card e-mandate, NACH), Vendor API(s), Channel providers (email, SMS, WhatsApp).
- **Data Stores**:
- Postgres for OLTP;
- Redis for cache, idempotency keys, locks;
- Object store for invoices/templates;
- Vault for secrets/keys.
- **Observability**: Metrics, tracing, logs; dashboards for payments, fulfilment, notifications.
- **Security**: Tokenization for cards via PG; PII encryption at rest; fine-grained IAM; least privilege; WAF.

```
[Client] → [API Gateway] → [User/Recipient/Calendar/Catalog/Policy/Payments/
Fulfilment/Notification]
↘↙
                                    [Event Bus]   [Audit Log]
External: Google Calendar ↔ Calendar Service
External: Payment Gateway ↔ Payments Service ↔ UPI Autopay/Card/NACH
External: Vendor APIs ↔ Fulfilment Service
External: Email/SMS/WhatsApp ↔ Notification Service
```

1.2 Key Sequence Diagrams (text form)

A) Onboarding with UPI Autopay Mandate 1. Client → API: start mandate (user, cap, frequency, rail=UPI).
2. Payments Service → PG: create mandate; PG returns deeplink/QR.
3. User approves in UPI app (one-time AFA).
4. PG → webhook: mandate_activated(mandate_id, cap, frequency).
5. Payments → User Service: update instrument status=Active.
6. Notification → user: setup success.

B) Scheduled Digital Gift Execution 1. Scheduler (T-10d): pulls upcoming events; calls Policy Engine with (level, expected price, caps).
2. Policy decides: proceed / swap SKU / alternate rail / skip; write decision.
3. Scheduler (T-24h): Notification sends pre-debit.
4. Scheduler (T-0 10:00): Payments charges via PG (idempotency key).
5. On success: Fulfilment requests e-code from vendor; receives code.
6. Notification delivers code to recipient; Audit logs; status visible to user.

C) Google Calendar Import 1. Client → Calendar: OAuth consent; refresh token stored.
2. Calendar poll/webhook pulls events; parser maps birthday/anniversary; dedupe; suggestions to user for mapping if ambiguous.

1.3 API Contracts (Representative JSON)

Auth - `POST /v1/auth/login`
Request: { email, otp }
Response: { access_token, refresh_token, expires_in }

Mandates & Payments - `POST /v1/mandates` Request: { rail: "upi"|"card"|"nach", cap_amount, currency, frequency } Response: { mandate_id, status, auth_link?, instructions }

- `GET /v1/mandates/{mandate_id}` Response: { mandate_id, rail, status, cap_amount, currency, frequency, created_at }
- `POST /v1/payments/charge` Request: { order_id, instrument_id, amount, currency, idempotency_key } Response: { charge_id, status, pg_txn_id, auth_state, message }
- **Webhooks (PG → GiftCare)**
- `/webhooks/payments`: events = mandate_activated, payment_succeeded, payment_failed, chargeback, refund_processed Payload base: { event, data: {...}, signature }

Recipients & Events - `POST /v1/recipients` Request: { name, relationship, contacts: { email?, phone?, whatsapp? }, address? } Response: { id, ... }

- `POST /v1/events` Request: { recipient_id, type: "birthday"|"anniversary"|"custom", date, recurrence?, timezone } Response: { id, ... }
- `POST /v1/calendar/google/connect` Request: { code }
  Response: { status: "connected" }

Catalog & Policy - `POST /v1/catalog/select` Request: { user_id, level_code, constraints? } Response: { sku_id, vendor_id, expected_amount, currency }

- `POST /v1/policy/decide` Request: { user_id, event_id, expected_amount, instruments: [...], policies: {...} } Response: { decision: "proceed"|"swap_sku"|"alternate_rail"|"skip", details }

Fulfilment & Notifications - `POST /v1/fulfil/digital` Request: { order_id, vendor_id, sku_id } Response: { code, expiry, redemption_url }

- `POST /v1/notify/send` Request: { channels: ["email"|"sms"|"whatsapp"], template_id, to, vars } Response: { message_ids: [...] }

Idempotency & Errors - All POSTs accept `Idempotency-Key` header.
- Standard error: { error: { code, message, field?, retryable? } }

1.4 Data & Infra Notes

- DB schema per PRD data model; foreign keys; soft deletes; created_at/updated_at.
- Partition large tables (events, charges, notifications) by month.
- Use outbox pattern for PG and Vendor calls; DLQ with automatic replay; backoff with jitter.
- PII encryption and field-level access policies; audit log is append-only.

---

2) UX – Screens, Flows, Content

2.1 Information Architecture

- **Onboarding**
- Mandate setup → Calendar connect → Recipients → Policies → Review
- **Home**
- Upcoming events, status chips, quick actions (add recipient/event)
- **Recipients**
- List → Detail (level, channels, next event, last gift)
- **Calendar**
- Month view with click-to-add
- **Automation Log**
- Timeline of pre-debits, charges, fulfilments, notifications
- **Settings**
- Payment rails, notification channels, export invoices (P1)

2.2 Key Flows

A) First-Run Onboarding 1. Welcome → value prop: “Set it once, never miss again.”
2. Mandate setup: pick rail (UPI recommended) → approve → success.
3. Connect Google Calendar (skip allowed).
4. Add first recipient (smart defaults by relationship → level).
5. Choose policies (swap SKU vs alternate rail vs skip).
6. Review summary → Finish.

B) Add Recipient - Form: name, relationship, birthday (day/month), delivery channel, level.
- Validation: at least one contact method; birthday can be yearless.

C) Create Event In-App - Calendar tap → modal: type, recipient, date, recurrence; timezone auto-filled.

D) FYI Notifications - Pre-debit: amount, date, rail, cancel window info (non-blocking).
- Delivery: “Sent to Rahul via WhatsApp at 10:02.”

2.3 Wireframe Notes (low-fi spec)

- **Home**: hero banner with next events; card list with chips: Ready / Needs policy / Skipped.
- **Recipient Detail**: header with name + relationship; level selector; contact chips; next event; history list.
- **Calendar**: month grid; dots for events; add button; import status toast.
- **Automation Log**: filter by status; each item shows time, action, result, deep link to detail.

2.4 Content Guidelines

- Tone: calm, reliable, concise.
- Avoid approval-seeking language on event day; reinforce automation.
- Error copy focuses on self-healing: “We’ve selected a similar gift within your level due to price changes.”

2.5 States

- Empty, loading, success, warning (policy needed), error (vendor/rail down), skipped (by policy).
- Accessibility: keyboard navigation, labels, high-contrast mode.

---

3) PO – Backlog Breakdown with Acceptance Criteria

3.1 Epics

- E1: Mandate Onboarding & Payments
- E2: Calendar & Events
- E3: Recipients & Levels
- E4: Catalog & Policy Engine
- E5: Scheduler & Orchestration
- E6: Fulfilment & Notifications
- E7: Observability & Admin
- E8: Security & Compliance

3.2 Stories (top P0)

E1-S1: Create UPI Autopay Mandate - Given I’m authenticated, when I start UPI mandate with cap and frequency, then I receive a deeplink/QR and, after approval, the mandate shows Active.
AC: webhook processed, status Active, cap stored, pre-debit channel verified.

E1-S2: Tokenized Card e-Mandate - Given I choose card rail, when I complete one-time AFA, then mandate is Active with token_ref stored.
AC: token_ref non-null, policy blocks charges beyond cap without AFA.

E2-S1: Google Calendar Connect & Import - Given I connect Google, when sync runs, then birthdays/anniversaries appear with mapping suggestions and no duplicates.
AC: at least 90 percent correct classification in test set.

E2-S2: In-App Event Creation - Given a recipient, when I add an event with yearly recurrence, then it appears in calendar and scheduler queue at correct timezone.

E3-S1: Add Recipient - Given I enter name, relationship, birthday, and contact, when I save, then the recipient is created with level and delivery channel.
AC: validation if no contact; birthday can omit year.

E4-S1: Policy Decision - Given an upcoming event, when expected price exceeds primary cap, then the policy engine either swaps SKU, alternates rail, or skips per user policy.
AC: decision persisted; no transaction splitting.

E5-S1: Daily Scheduler Run - Given events in next 14 days, when scheduler runs, then it forecasts, sends pre-debits at T-24h, and enqueues T-0 execution.
AC: idempotent; re-runs do not duplicate.

E6-S1: Digital Gift Fulfilment - Given a successful charge, when fulfilment requests e-code, then code is delivered to recipient within 60 seconds.
AC: delivery proof logged; retries on transient failures.

E6-S2: Notifications - Given an upcoming charge, when pre-debit is due, then a message is sent with required info; after fulfilment, a delivery confirmation is sent.
AC: deliverability $\geq$ 99 percent in staging tests.

E7-S1: Audit & Metrics - Given any state change, when it occurs, then an immutable audit record is written and metrics emitted.

E8-S1: Compliance Guards - Given a planned charge, when amount exceeds mandate parameters, then the system does not attempt auto-charge and applies policy.

3.3 Definition of Ready (DoR)

- Story has clear user value, acceptance criteria, dependencies listed, test data ready, UI copy outlined, non-functional constraints noted.

3.4 Definition of Done (DoD)

- Code merged with tests; feature flags in place; telemetry added; documentation updated; security review passed; e2e scenario green in staging.

3.5 Dependencies & Milestones

- Phase A: E1-S1, E2-S1, E3-S1, E5-S1, E6-S2
- Phase B: E1-S2, E4-S1, E6-S1, E7-S1
- Phase C: Physical fulfilment adapters, invoices, admin dashboard

---

4) Handoff Notes

- Architects: confirm PG capabilities for mandate webhooks and cap change events; finalize idempotency strategy across services.
- UX: validate copy for non-blocking FYIs; ensure policy selection is crystal clear.
- PO: order P0 stories for sprint 1 and create test data fixtures for mandates, vendors, and calendar imports.