Goal

An Android (Flutter) app that ingests McDonald's "Points history" via screenshots, screen recordings, or live on-device capture, then outputs:

1. points expiring per month for the next 3 calendar months,
2. YTD totals for Earned / Used / Expired,
3. a correct current balance, and
4. a monthly chart + FIFO lot ledger.

Platforms & constraints

- Android only, min Android 14 (API 34).
- Single McD account.
- English locale first (MYT +08:00).
- On-device OCR (Google ML Kit).
- No cloud storage. SQLite for data.
- Target: a full-history import in ~1 minute (optimize aggressively).

Data ingestion

Supported inputs at launch

1. **Screenshots** (gallery picker).
2. **Screen recordings** (gallery video; user scrolls through full history).
3. **Live capture (accessibility)**: optional live screen grab while user scrolls; no auto-scroll needed.

Frame extraction (for video/live)

- Sample frames at 2–5 fps (adaptive: increase only when UI changes).
- Debounce identical frames using perceptual hash (pHash).
- Crop detection: detect the list rows region once (first frame), reuse crop across frames.

OCR

- ML Kit Text Recognition v2.
- Confidence threshold: if any field (date/type/points) < 0.99, mark row as **Needs review**.

- Keep raw media only for **7 days**, then auto-delete.

Parsing rules

- Each visible row -> one transaction.
- Supported types: **Earned**, **Used**, **Expired**.
- Treat multiple entries on the same date as distinct rows if **points differ**.
- Dedup key: (date, type, points); if collision occurs, keep the first seen.
- Expected text patterns per row (examples):
  - Earned / Used / Expired
  - Date: YYYY-MM-DD (e.g., 2025-08-10)
  - Points: integer (e.g., 377)
- Regex:
  - type: \b(Earned|Used|Expired)\b
  - date: \b(20\d{2})-(0[1-9]|1[0-2])-(0[1-9]|[12]\d|3[01])\b
  - points: \b-?\d{1,6}\b (normalize sign: Used negative, others positive)

Manual review screen

- Table of parsed rows with inline edit (date picker, type dropdown, number input).
- Save only after user confirms.

Domain logic

Expiry rule

- Each **Earned** lot expires **exactly 12 months** after earned_date (same day, MYT).
- No lot splitting required beyond FIFO accounting.

Consumption rule (FIFO)

- When **Used**, subtract from **oldest unexpired** Earned lots first.
- When **Expired**, remove remaining units in aged lots on their expiry date.
- Keep an internal **lot ledger** to replay history deterministically.

Calculations

- **Current Balance** = sum(all Earned) − sum(Used) − sum(Expired).
- **YTD aggregates** (Jan 1–today): totals for Earned / Used / Expired.

- **Upcoming 3 months** (calendar months): compute expiring amounts by summing lots whose expiry_month ∈ {M+1, M+2, M+3} with remaining units as of today.
- Provide a **monthly bar chart** of Earned/Used/Expired and a **lot ledger** view.

Data model (SQLite)

```
-- transactions: raw, human-confirmed rows

CREATE TABLE transactions (

 id INTEGER PRIMARY KEY AUTOINCREMENT,

 source TEXT NOT NULL CHECK (source IN ('screenshot','video','live')),

 type TEXT NOT NULL CHECK (type IN ('Earned','Used','Expired')),

 points INTEGER NOT NULL,

 date TEXT NOT NULL,            -- ISO 8601 YYYY-MM-DD (MYT)

 created_at TEXT NOT NULL DEFAULT (datetime('now')),

 unique_hash TEXT NOT NULL UNIQUE  -- SHA-1 of "date|type|points"

);


-- earned_lots: normalized per Earned row (one row per Earned transaction)

CREATE TABLE earned_lots (

 lot_id INTEGER PRIMARY KEY,

 transaction_id INTEGER NOT NULL UNIQUE REFERENCES transactions(id) ON DELETE
CASCADE,

 earned_date TEXT NOT NULL,

 original_points INTEGER NOT NULL,

 consumed_points INTEGER NOT NULL DEFAULT 0,

 expired_points INTEGER NOT NULL DEFAULT 0,

 expiry_date TEXT NOT NULL,       -- earned_date + 12 months (same day)

 CHECK(original_points >= 0),

 CHECK(consumed_points >= 0),

 CHECK(expired_points >= 0)
);


-- ledger_events: deterministic replay (for transparency & debug)
CREATE TABLE ledger_events (
 id INTEGER PRIMARY KEY AUTOINCREMENT,
 event_date TEXT NOT NULL,
 event_type TEXT NOT NULL CHECK (event_type IN ('EARN','USE','EXPIRE')),
 ref_transaction_id INTEGER REFERENCES transactions(id),
 lot_id INTEGER REFERENCES earned_lots(lot_id),
 delta INTEGER NOT NULL          -- +EARN or -USE/-EXPIRE units applied to this lot
);
CREATE INDEX idx_events_date ON ledger_events(event_date);
```

Algorithms (pseudocode)

Import pipeline

```
for each asset in import_batch:
 frames = extract_frames(asset, fps=2..5, dedupe=true)
 for frame in frames:
  text = ocr(frame)
  rows = parse_rows(text)  // detect blocks, then regex
  for row in rows:
   key = sha1(f"{row.date}|{row.type}|{row.points}")
   if !exists(transactions.key):
    insert row as 'Needs review' if low confidence
// After user verifies:
normalize_to_lots()
replay_ledger()
```

Normalization & ledger replay

```
normalize_to_lots():
 for t in transactions where type='Earned':
   upsert earned_lots(lot_id=t.id, original_points=t.points,
     earned_date=t.date, expiry_date=add_months(t.date,12))

replay_ledger():
 reset consumed/expired and ledger_events
 // 1) EARN events
 for lot in earned_lots ordered by earned_date:
   add_event(lot.earned_date, 'EARN', lot_id, +lot.original_points)

 // 2) USE events (FIFO)
 for t in transactions where type='Used' ordered by date:
   remaining = abs(t.points)
   for lot in earned_lots where lot.expiry_date >= t.date and lot.remaining()>0 ordered by earned_date:
     take = min(remaining, lot.remaining())
     lot.consumed_points += take
     remaining -= take
     add_event(t.date, 'USE', lot_id, -take)
     if remaining == 0: break

 // 3) EXPIRE events
 for lot in earned_lots ordered by expiry_date:
   rem = lot.remaining() // original - consumed - expired
   if rem > 0 and lot.expiry_date <= today:
     lot.expired_points += rem
     add_event(lot.expiry_date, 'EXPIRE', lot_id, -rem)
```

Projections (next 3 calendar months)

```
months = next_three_calendar_months(today)
for m in months:
  expiring_in_m = sum(lot.remaining() where month(lot.expiry_date)==m)
return [m -> expiring_in_m]
```

UI/UX

Home

- Cards:
    1. **Expiring Soon (next 3 months)** — month chips with totals.
    2. **YTD Earned / Used / Expired** — stacked bars or 3 small stat tiles.
    3. **Current Balance** — big number; tap to open lot ledger.

History

- Paginated list of confirmed transactions.
- “Needs review” banner if any rows pending confirmation.

Ledger

- FIFO lot table: earned date, original, consumed, expired, remaining, expiry date.

Charts

- Monthly bars showing Earned/Used/Expired for the last 12 months.

Notifications

- User-selectable schedule:
    - **14 days before month-end or 1st of month**, 09:00 MYT.
- Threshold alert: configurable (default 1,000) for “Next month’s expiring > threshold”.
- Use flutter_local_notifications + Android 14 exact alarms permissions if needed.

Permissions

- Photos & videos (gallery).
- Notifications.
- Accessibility service (optional for live capture; explain clearly).
- Foreground service for live capture if implemented.

Performance targets

- Video processing: adaptive fps, frame dedupe, batch OCR, isolate for parsing.
- Aim < 1 min for 500–800 rows on a mid-range device.
- Provide progress meter (frames parsed, rows detected, confidence).

Testing

- Include **10+ sample screenshots** and **3 sample videos** (portrait scrolling).
- Golden tests for:
    - OCR parsing (date/type/points extraction).
    - Deduplication logic.
    - FIFO consumption across edge cases (partial consumption, same-day events).
    - Expiry projection (month boundaries, leap years).
    - YTD totals (Jan 1 cutoff).
- Fuzz tests for minor OCR errors (e.g., 2025-08-10 vs 2025-08-1O).

Edge cases to handle

- Same day Earned and Used.
- Month-end and year-end boundaries.
- Negative values only for Used; ensure Expired derived from lots, not raw input.
- Device time vs MYT (force MYT for all logic).
- Duplicated frames/rows across imports.

Single Cursor prompt (copy-paste)

You are building an Android-only Flutter app (min Android 14) that parses McDonald's "Points history" and reports points Earned/Used/Expired with correct FIFO and expiries.

Scope:

- Inputs at launch: screenshots (gallery), screen recordings (gallery), and optional live capture (accessibility). User scrolls; no auto-scroll required.

- Locale: English, MYT (+08:00). On-device OCR only (Google ML Kit).

- Storage: SQLite (sqflite). No cloud or exports. No biometrics.

- Performance goal: ingest a full history in ~1 minute on a mid-range device.

Implement:

1) Ingestion

  - For videos/live: extract frames at 2–5 fps with adaptive sampling and perceptual dedupe (pHash).

  - Use ML Kit Text Recognition v2 on each frame. Assemble rows by detecting list-row blocks and parsing:

    type: (Earned|Used|Expired)

    date: YYYY-MM-DD

    points: integer; normalize sign (Used negative).

  - Confidence policy: if any field < 0.99 flag row as "Needs review".

  - Deduplicate rows by (date,type,points) using SHA-1 "date|type|points" as unique_hash.

  - Keep raw media at most 7 days; then auto-delete.

2) Review UI

  - A table editing screen listing parsed rows with low confidence.

  - Allow fixing date/type/points and confirming save.

3) Domain model & DB (SQLite)

  - Tables: transactions, earned_lots, ledger_events (see schema below).

  - Normalize Earned into earned_lots with expiry_date = earned_date + 12 months (same day).

  - FIFO rule: Used consumes from oldest unexpired lots first.

  - Expiry: on expiry_date, remaining units in a lot expire (no lot-splitting beyond FIFO).

  - Replay engine rebuilds ledger_events deterministically on every change.

SQL schema:

```
CREATE TABLE transactions (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  source TEXT NOT NULL CHECK (source IN ('screenshot','video','live')),
  type TEXT NOT NULL CHECK (type IN ('Earned','Used','Expired')),
  points INTEGER NOT NULL,
  date TEXT NOT NULL,
  created_at TEXT NOT NULL DEFAULT (datetime('now')),
  unique_hash TEXT NOT NULL UNIQUE
);

CREATE TABLE earned_lots (
  lot_id INTEGER PRIMARY KEY,
  transaction_id INTEGER NOT NULL UNIQUE REFERENCES transactions(id) ON DELETE
CASCADE,
  earned_date TEXT NOT NULL,
  original_points INTEGER NOT NULL,
  consumed_points INTEGER NOT NULL DEFAULT 0,
  expired_points INTEGER NOT NULL DEFAULT 0,
  expiry_date TEXT NOT NULL
);

CREATE TABLE ledger_events (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  event_date TEXT NOT NULL,
  event_type TEXT NOT NULL CHECK (event_type IN ('EARN','USE','EXPIRE')),
  ref_transaction_id INTEGER REFERENCES transactions(id),
  lot_id INTEGER REFERENCES earned_lots(lot_id),
  delta INTEGER NOT NULL
);
CREATE INDEX idx_events_date ON ledger_events(event_date);
```

4) Calculations & UI

  - Home cards:

    a) Expiring soon (next 3 calendar months) – computed from remaining units in earned_lots by expiry month.

    b) YTD totals (Jan 1–today): Earned / Used / Expired from transactions + replayed ledger.

    c) Current Balance.

  - Charts: 12-month bar chart of Earned/Used/Expired.

  - Ledger screen: per-lot view (earned date, original, consumed, expired, remaining, expiry).

5) Notifications

  - Settings: choose either “14 days before month-end” or “1st of each month” at 09:00 MYT.

  - Threshold alert (default 1000): notify if next month’s expiring total > threshold.

  - Use flutter_local_notifications.

6) Tests

  - Provide 10+ sample screenshots and 3 videos.

  - Golden tests for parsing, dedupe, FIFO, expiries, YTD, and month-boundary cases.

  - Fuzz replacement tests for OCR confusion (0/O, 1/I).

7) Libraries (suggested)

  - google_mlkit_text_recognition

  - image_picker / file_picker

  - video_player or ffmpeg_kit_flutter for frame extraction + hashing

  - sqflite + path_provider

  - crypto (SHA-1)

  - flutter_local_notifications

  - riverpod for state management

Deliverables:

- Flutter app with modular layers: ingestion, parsing, domain (FIFO/expiry), persistence, presentation.

- Well-documented Replay engine (deterministic).

- Seed data & golden tests passing.

Non-functional:

- All date math in MYT, ISO 8601 (YYYY-MM-DD).

- Aim for import UX with progress meter and cancel button.

If you want, I can also generate:

- a ready-made **Riverpod** project structure,
- a **frame extractor** helper (with pHash),
- and a set of **golden test fixtures** from your screenshots.