BATTERY HEALTH REMAINDER SYSTEM

TECH STACK, ARCHITECTURE & TOOLS

Project Title:

Automated Battery Health Reminder System Using Local Python Scheduler & Mock Databases

1. Objective

The objective of this project is to automatically remind users to check their smart lock's battery level if it has not been checked in the last 30 days.
Additionally, the project measures the effectiveness of notifications by tracking how many users click on the notification (CTR – Click Through Rate).

This system fulfills three major goals:
✔ Identify inactive locks
✔ Notify their users
✔ Track user engagement for weekly analysis

2. Technologies Used

Programming Language

- Python 3.x

Tools & Libraries

- Standard Python Libraries:
    - json → mock data storage
    - datetime → stale lock calculation
    - uuid → campaign ID generation
- No external databases used (AWS skipped as requested)

3. Local Mock Database Setup

Instead of AWS DynamoDB & PostgreSQL, we used:

Purpose	File	Description
Lock data	locks.json	Simulates DynamoDB table for lock status
Notification log	sent_notifications.json	Stores sent notification records
Click log	click_logs.json	Records user clicks

This makes the project lightweight and locally runnable.

4. System Architecture

Flow Overview

1. Load Lock Data
   Read locks.json.
2. Identify Stale Locks
   Any lock whose last_checked is older than 30 days is considered stale.
3. Send Notification (Simulated)
   A mock notification is generated for each stale lock.
4. Log Notifications
   Every sent notification is added to sent_notifications.json.
5. Track Clicks
   Via CLI commands, users simulate clicking notifications.
6. Generate Summary Report
   CTR is calculated based on clicks vs. notifications sent.

5. How to Run the System

Run the weekly reminder script

python main.py

Simulate a notification click

python -m analytics click <lock_id> <campaign_id>

Generate campaign CTR summary

python -m analytics summary <campaign_id>

FINDINGS & RECOMMENDATIONS

1. Findings

A. Users often forget to check lock battery levels

By analyzing stale lock data, multiple locks exceeded the 30-day limit, indicating users do not regularly monitor battery health.

B. Notifications improve user engagement

Tracking notification clicks provides measurable metrics.
Example:

Notifications Sent: 10

Clicks: 4

CTR: 40%

This data helps understand how responsive users are to reminders.

C. Click-Through Rate (CTR) is a strong indicator

CTR helps measure campaign success:

- High CTR → Users appreciate reminders
- Low CTR → Notification content/timing may need optimization

D. Separate notification and click logs help in campaign analytics

Maintaining two logs (sent_notifications.json, click_logs.json) provides clarity in reporting and eliminates data mixing.

2. Recommendations

1. Migrate to Cloud for Production

For real-world scaling:

- Use AWS DynamoDB for locks
- AWS RDS PostgreSQL for user mapping
- AWS Lambda for automated weekly execution
- FCM for real push notifications

Would remove manual triggers and improve performance.

2. Improve Notification Content

Richer notifications could improve CTR:

- Show “Estimated Battery Life Remaining”
- Highlight how long it has been since last check
- Provide actionable CTA (e.g., “Tap to open app and check battery”)

3. Add User Incentives

Introducing reward points or badges for regular lock maintenance could boost engagement.

4. Add In-App Battery Health Dashboard

Charts showing:

- Battery check history
- Estimate of next expected check
- Battery performance trends

This builds user trust and helps maintenance planning.

5. Predict Battery Failure using Simple Algorithms

Future enhancement:

- Analyze historical timestamps
- Predict potential battery failure
- Send pre-emptive alerts before failure occurs

This could reduce sudden lock-out issues**.**

GitHub Link: https://github.com/Bharath-B1805/Atomberg_Battery_Remainder