DrivePilot Live

Public Project Case Study - Resume Portfolio Version

A local-first market-signal dashboard that turns high-volume LINE group notifications into structured operational context, including notification intake, location parsing, dashboard views, health checks, and report generation.

GitHub Repository: https://github.com/vampirehollieee/drivepilot-live

Project Type	Independent personal project / portfolio case study
Product Positioning	LINE notification market-signal observation system
Core Constraint	No automated order acceptance, no LINE operation, no dispatch decision automation
Public Version	Sanitized demo screenshots and sample data only

Overview

DrivePilot Live is a local-first dashboard designed to reduce the cognitive load of reading high-volume LINE group notifications during ride-driving operations. It converts noisy notification streams into structured market signals such as active zones, parsed locations, system status, and report snapshots.

The public repository is a sanitized portfolio version. It demonstrates architecture, runtime scripts, UI direction, documentation, and demo data without exposing real LINE messages, real locations, private configuration, API keys, or webhook URLs.

Problem

LINE working groups can produce fragmented notifications at high speed. Important location signals, dispatcher messages, repeated zones, and operational context can be mixed with irrelevant or duplicated messages.

Manually reading every message is slow, stressful, and easy to miss. The project was created to transform raw notifications into a more readable market-observation cockpit instead of forcing the user to stare at chat streams all day, because apparently humanity decided chat apps should also be command centers.

Solution

DrivePilot Live receives forwarded Android notifications, parses useful fields, classifies market signals, stores structured local data, and displays the results through desktop and mobile dashboards. It also provides health checks, report generation, and optional notification routing through Discord.

The system deliberately avoids risky automation. It does not accept orders, does not operate LINE, does not make dispatch decisions, and does not replace human judgment.

System Architecture

1	**LINE group notifications**	Source notification stream from working groups.
2	**MacroDroid on Android**	Captures notification content and forwards it.
3	**Local HTTP receiver**	Receives POST requests on the local machine.
4	**Parser / resolver layer**	Extracts locations, aliases, zones, signal type, and confidence metadata.
5	**File-based data layer**	Stores structured outputs and generated summaries.
6	**Dashboard / mobile dashboard**	Displays market overview, map view, active zones, and system state.
7	**Report / health tools**	Builds market reports and checks runtime health.

Core Features

- Notification intake from Android-forwarded LINE notifications.
- Parser workflow for addresses, aliases, zones, signal categories, and confidence metadata.
- Desktop dashboard for command-center style market observation.
- Mobile dashboard for quick review during operation.
- Market report generation using structured local data.
- System health checks for local services and data freshness.
- Sanitized demo screenshots and demo JSON samples for public review.

Technology Used

Layer	Technologies / Files
Runtime	Windows, PowerShell, BAT scripts, local-first workflow
Data Processing	Python standard library, JSON, JSONL, CSV
Frontend	HTML, CSS, JavaScript, Leaflet / map UI
Notification Flow	MacroDroid, local HTTP receiver, optional Discord webhook integration
Documentation	README, live-system README, runtime notes, demo-data, sanitized screenshots

Delivered Outcomes

- Built a working end-to-end flow from notification intake to dashboard presentation.
- Created desktop and mobile dashboard directions for fast operational reading.
- Separated public portfolio materials from private runtime data.
- Added sanitized screenshots and demo JSON outputs to make the repository reviewable.
- Established a structured Codex-assisted workflow with task boundaries and changelog discipline.

Safety and Privacy Handling

The public repository is intentionally sanitized. It excludes real runtime data, original LINE messages, actual addresses, private configuration files, API keys, webhook URLs, generated logs, and live data outputs.

Demo screenshots use abstract grids, masked map visuals, fake groups such as Group A / Group B / Group C, and placeholder zones such as North Zone / Central Zone / South Zone. The purpose is to show product structure without exposing real operational data.

AI-Assisted Development Workflow

The project uses a controlled AI-assisted workflow rather than broad, uncontrolled code generation. Tasks are scoped into small Codex work orders, each with explicit boundaries, acceptance criteria, and no-go areas. The workflow prioritizes preserving existing working behavior, documenting changes, and avoiding unnecessary rewrites.

Project Highlights

- Defined a real-world operational problem and converted it into a focused software tool.
- Designed a local-first data pipeline instead of relying on cloud infrastructure prematurely.
- Built a dashboard-oriented interface for noisy notification streams.
- Applied privacy-aware public repository preparation, including demo data and sanitized screenshots.
- Managed AI-assisted development with clear task boundaries, review steps, and changelog discipline.

Roadmap

Near Term	Improve public documentation, keep demo screenshots updated, preserve privacy boundaries.
Product Stability	Continue parser quality checks, address resolver improvements, and system health reporting.
Portfolio Polish	Improve README layout, case-study presentation, and project storytelling for job applications.

Note: This public case study is designed for portfolio review. It summarizes the project without exposing private runtime data or sensitive implementation secrets.