1. Executive Summary & Purpose

Project Name: Kamleon PHT (Predict Health Toilet) Data Integration

Task involved: **Fase 1. Rediseño de prototipos sensóricos, arquitectura de Big Data e infraestructura modular.**

Resultado: APIs funcionales y listas para la integración).

Document Version: 1.0

Target Audience: Grants Justification Team, Technical Auditors, Project Stakeholders

1.1 Purpose

The primary objective of this integration is the automated synchronization of physiological hydration data from Kamleon's smart monitoring units to the **Lighthouse** clinical platform. This ensures that the data collected during the PHT project is centralized, verifiable, and available for validation—key requirements for the successful justification of project grants.

1.2 Systems Involved

System	Role	Technology Stack
Kamleon Cloud	Data Source	Google Firebase (Firestore)
PHT Sync Service	Middleware / Logic	Python 3.x with OAuth2
Lighthouse API	Data Consumer	RESTful API (OpenAPI 3.1.0)

1.3 Implementation Requirements & Constraints

- **Initial Requirement:** The PHT project requires clinical validation on a third-party platform (Lighthouse) to justify the physiological accuracy of Kamleon units for grant funding.
- **Key Dependencies:** The service utilizes the `firebase-admin` SDK for secure data extraction and the `requests` library for TLS-encrypted API communication.
- **Operational Constraints:** The service is configured with a **Batch Limit (200 records)**. This constraint was implemented to ensure that large data syncs do not exceed the Lighthouse API's timeout limits or memory buffers.
- **Assumptions:** It is assumed that the `analysisDate` field in the Kamleon database is the primary source of truth for the clinical timeline.

---

2. High-Level Architecture & Data Flow

The integration follows a **"Stateful Middleware"** pattern. Unlike a simple trigger, this system maintains a "memory" of what has been sent to prevent data duplication or gaps.

2.1 The Data Lifecycle

1. **Generation:** A Kamleon unit performs a measurement and uploads raw data to **Google Firebase**.
2. **Detection:** The **Sync Service** (running on a secure Kamleon server) queries Firebase for any records created *after* the last successful sync.
3. **Transformation (The "Cleaning" Phase):** The service re-formats the data (e.g., converting dates to Unix timestamps and adjusting casing) to meet the strict Lighthouse API specifications.
4. **Transmission:** The service requests an **OAuth2 Token** and, once authorized, pushes the data to the Lighthouse production endpoint.
5. **Confirmation:** Only after the API returns an `HTTP 201 Created` status does the service update its internal "Clock" (State File) to mark the process as complete. This specific status code ensures that Lighthouse has not only received the data but has successfully committed it to their clinical database.

---

3. Operational Modes

To guarantee 100% data availability for grant justification, the Sync Service operates in two distinct logical modes. This ensures that whether data is being sent in real-time or recovered from a past technical outage, the process remains consistent.

3.1 Mode A: Recurrent Synchronization (The "Continuous Janitor")

This is the standard operating mode for the PHT project. It acts as a continuous background process to ensure Lighthouse always has the most recent data.

- **Trigger:** The service runs as a **Persistent Background Daemon** using an internal execution loop.
- **Logic:** A `while` loop architecture that initiates a synchronization cycle, then pauses for a defined interval (e.g., 10 minutes) before automatically re-triggering.
- **Resilience:** Unlike a scheduled task, this resident service maintains a constant connection to the environment, allowing for immediate error recovery and state persistence in memory.
- **Purpose:** To provide seamless, automated data mirroring between Kamleon and Lighthouse without requiring external scheduling triggers.

3.2 Mode B: On-Demand Recovery (The "Time Machine")

This mode is designed specifically for technical audits and grant justifications. It allows an administrator to bypass the "Global Clock" to fix specific data gaps.

- **Trigger:** Manual execution by a Kamleon technician using command-line arguments (e.g., `--mode ondemand`).
- **Data Selection:** Targeted by **Device ID** and a **Start Timestamp** (e.g., "Pull all data for Unit 005 starting from March 1st").
- **Logic:** It uses a `>=` (Greater than or equal to) operator to "sweep" a specific period of history for the selected units.
- **Purpose:** To recover data in the event of a localized internet outage at a clinical center, ensuring the PHT project documentation is 100% complete for the final grant report.

3.3 Design Decisions & Alternatives

Alternative 1: Firebase Cloud Functions (Real-time Triggers)

- **Decision:** Rejected.
- **Reasoning:** Cloud Functions are "stateless" and ephemeral. If the Lighthouse API is temporarily down or experiencing high latency, a Cloud Function might timeout or fail silently, leading to permanent data gaps.

Alternative 2: Scheduled Cron Jobs (Task-based Execution)

- **Decision:** Rejected.
- **Reasoning:** Cron jobs trigger a new instance of a script at fixed intervals. If a previous sync is still running (due to a large data batch or slow network), a second Cron instance could start, leading to "Race Conditions" or duplicate data processing. Furthermore, Cron does not easily allow for resident memory persistence.

Chosen Approach: Standalone Python Middleware (Persistent Daemon)

- **Reasoning:** * **Stateful Checkpointing:** By using a continuous `while` loop within a single persistent process, the service maintains a strict "Linear History." It ensures that Batch B never starts until Batch A is confirmed with an **HTTP 201 Created**.
    - **Resource Efficiency:** A resident service avoids the overhead of "cold-starting" the Python interpreter and re-authenticating with Firebase every few minutes.
    - **Audit-Proof Reliability:** This architecture provides a 100% reliable audit trail via the `last_sync_state.json` file. If the service is interrupted, it resumes exactly where it stopped, ensuring no record is ever skipped—a critical requirement for grant justification.
    - **Development Velocity:** This decoupled approach allowed for a faster development timeline and easier local testing ("local_dump" mode) without interfering with the production cloud infrastructure.

---

4. Technical Architecture & Reliability

This section explains the "Safety Net" built into the code to ensure the data is handled professionally.

4.1 Stateful Tracking (The "Checkpoint" System)

The service does not "guess" what to send. It uses a **Strict Checkpoint Logic**:

1. **Download:** Data is pulled from Firebase.
2. **Transmit:** Data is sent to the Lighthouse API.
3. **Confirm:** The API must return an `HTTP 201 Created` (Success).
4. **Save:** Only *after* success is confirmed, the service updates the `last_sync_state.json` file.

- **Benefit:** If the internet fails halfway through, the "Checkpoint" isn't moved. When the internet returns, the service simply starts over from the last successful point. **No data is ever lost.**

4.2 Security & Authentication

The integration utilizes **OAuth2 (Password Grant Flow)**.

- **Handshake:** The service sends encrypted credentials to the Lighthouse Identity Provider.
- **Bearer Token:** It receives a temporary "Access Token" valid for a specific duration.
- **Encryption:** Every single byte of data is wrapped in **TLS 1.2/1.3 encryption (HTTPS)**, ensuring medical data remains private during transit.

4.3 Data Validation & Testing (Local Dump)

To ensure the highest level of data integrity before production deployment, the Sync Service includes a specialized **Internal Folder Target**.

- **Purpose:** This mode allows technicians to "intercept" the data batches locally instead of sending them to the API. It is used to verify that the **Data Normalization** (Unix timestamps, lowercase mapping, and field renaming) is functioning correctly.
- **Trigger:** Activated by switching the `active_target` in the `config.yaml` to `local_dump`.
- **Output:** The service generates timestamped JSON files (e.g., `20260327_143005_unit_022.json`) within a secure local directory (`output-json`).

- **Audit Value:** This provides a "Pre-Flight" check capability. Auditors can compare the raw Firestore record against the local JSON dump to confirm that no data was corrupted or lost during the transformation process.

---

5. Data Model & Field Mapping (PHT Production)

To ensure the PHT project meets the strict medical data standards of the Lighthouse platform, the Sync Service extracts a comprehensive set of 30+ parameters. These are categorized into Identifiers, Clinical Data, and High-Resolution Sensor Metadata.

5.1 Field Inventory & Definitions

The following fields are synchronized in every successful batch:

Category	Field Name	Description	Transformation
Identifiers	measureID, userID, unitID, centerID, organizationID	Unique keys linking the test to a specific user, hardware unit, and clinical site.	None
Core Clinical	analysisDate	The exact moment the physiological sample was analyzed.	**Converted to Unix Milliseconds (13-digit integer)**
Core Clinical	measType	The scenario of the test (e.g., urine, waterflush).	**Forced to lowercase**
Biomarkers	usg, pH, condValue, score, hematuria_score	Calculated indicators for hydration (Specific Gravity), acidity, and health scores.	None

Sensor Data	F0 through F7, clear	Raw spectral data from the 8-band optical sensor array.	None
Sensor Data	conductivity, real, img	The complete conductivity curve and baseline optical reference data.	None
Calibration & Normalization	postAssemblyCalibration, water_reference_8b	Hardware-specific calibration offsets to ensure inter-device consistency.	None

5.2 Specific Normalization Rules

To maintain a "Zero-Error" rejection rate from the Lighthouse API, the following logic is applied during the mapping phase:

1. **Temporal Integrity:** Firestore "Timestamp" objects are deconstructed and rebuilt as **Unix Epoch Integers**. This ensures that time-zone offsets do not interfere with the clinical timeline.
2. **Schema Enforcement:** String-based categories (like measType) are normalized to lowercase. For example, WaterFlush or Urine are transmitted as waterflush and urine to match the API's strict validation schema.
3. **Nested Object Preservation:** Complex structures like devices and model metadata are preserved to allow Lighthouse researchers to filter results by hardware version.

---

6. System Reliability & Audit Trail

For grant justification, the "PHT Sync Service" provides a transparent audit trail of every data movement.

6.1 Logging & Monitoring

The service maintains a rotating log file (`sync_service.log`) with a 5MB limit and 3-generation backup. It records:

- **Auth Events:** Success or failure of OAuth2 handshakes and the specific duration of token validity (defaulting to 14 days if not specified by the server).
- **Job Status:** Distinction between **Sync** (automatic) and **On-Demand** (manual) modes, including records found per unit.
- **API Telemetry:** Detailed server responses. In the event of a failure, the service logs the **exact JSON error message** returned by the Lighthouse API (e.g., specific validation errors for a 422 status).
- **Memory Usage:** The service utilizes `tracemalloc` and `psutil` to log **Peak RAM consumption** as a percentage of total system memory after every cycle, ensuring long-term server stability.

6.2 Error Handling & Retries

- **Network Resilience & The "Global Clock":** The system follows a "Confirm-before-Save" logic. The local `last_sync_state.json` is **only** updated after the API returns an **`HTTP 201 Created`** response. If the network is unreachable, the "Global Clock" freezes, and the service retries the same batch in the next cycle. No data is ever skipped.
- **401 Unauthorized (Expired Tokens):** The service implements an **Automatic Retry Mechanism**. If a 401 error is detected, the service clears the local token cache, requests a new Bearer Token, and immediately re-attempts the transmission.
- **Data Validation (422 Errors):** If the API rejects data due to formatting, the service logs the rejection details but does not advance the clock for that specific batch, allowing for manual technical intervention without data loss.

7. Future Enhancements

As the PHT project transitions from the pilot phase to large-scale clinical deployment, the following technical enhancements are planned to ensure maximum system performance:

7.1 Multi-Threaded Unit Synchronization (Parallelism)

- **Concept:** Currently, the "Janitor" service processes units sequentially. To decrease latency as more devices are deployed, the service will be upgraded to a multi-threaded architecture.
- **Benefit:** This allows the system to synchronize data from multiple clinical units (e.g., Unit 005 and Unit 025) simultaneously. This ensures that a network delay at one clinical center does not stall the data flow for the entire project.

7.2 Centralized Sync Management Dashboard (PHT Console)

- **Concept:** Building a web-based "Control Plane" or Frontend that allows the Aftersales team to monitor all active "while-loop" daemons in real-time.
- **Benefit:** This removes the need for a technician to use a terminal/CLI. An operator could trigger "On-Demand" recoveries, adjust sync intervals, or view memory/success logs for specific units through a visual interface, significantly reducing human error and operational costs.

7.3 Permanent Clinical Audit Vault

- **Concept:** Implementation of a long-term local archive of all JSON payloads sent to the Lighthouse API.
- **Benefit:** Since cloud databases (Firebase) often have retention limits or high storage costs for raw sensor arrays, this "Vault" ensures that 100% of the project's data remains available for legal and medical auditing for the full duration required by the grant, independent of cloud provider policies.

---

8. Future Enhancements

8.1 API & Output Technical Description

The PHT Sync Service is designed with a **Pluggable Output Architecture**, allowing the system to switch between a live clinical production environment and a local validation environment for technical auditing.

Target A: Production (Lighthouse / We-ShareCare)

- **Type:** RESTful API (OpenAPI 3.1.0 Compliant).
- **Authentication:** OAuth2 (Password Grant Flow) via TLS-encrypted HTTPS.
- **Endpoints:**
  - `POST /token`: Identity Handshake (returns a Bearer Access Token).
  - `POST /measurements`: Data ingestion point for batch processing.
- **Primary Goal:** Live synchronization of clinical data for project monitoring.

Target B: Validation (Local Data Dump)

- **Type:** Internal Filesystem Storage (JSON).
- **Mechanism:** Direct serialization of the normalized data to the server's local disk.
- **Path:** `output-json/` (Configurable in `config.yaml`).
- **Primary Goal:** Technical "Pre-Flight" testing and manual data verification. This ensures the **Normalization Rules** (milliseconds, lowercase mapping) are 100% accurate before transmitting to the production API.
  -

8.2 Sample API Request (Payload)

JSON

```
pht_production_fields:

  # Identifiers

  - "measureID"/ "urineID" [str]: The primary unique identifier for the measurement event

  - "cartridgeID" [str]: Identifier for the physical sensor cartridge installed in the unit.
```

- "userID" [str]: Identifier for the participant. Defaults to "guest" for non-logged-in sessions to maintain privacy while preserving the data point.

- "unitID"/deviceID" [str]: The unique serial number of the Kamleon urinal hardware.

- "centerID" [str]: Hierarchical identifiers used to group data by the specific clinical center

- "organizationID" [str]: Hierarchical identifiers used to group data by the specific clinical organization

- "screenID" [str]: Identifier for the digital interface version present on the hardware during the test.

- "wReferenceID" [str]: Identifier for the baseline calibration reference (Water Reference) used to normalize the sensor for that specific measurement.

- "lastUrine" [str]: cross-reference id of the last urine detected before this measure used for historic connection

- "lastMeasurement"/"lastMeasurementID" [str]: cross-reference id of the last measure detected before this measure used for historical connection

- "ID"/"measType" [str]: *Values:* urine (human sample), waterflush (cleaning cycle), init (system startup), stuck (issue in detection for long measure)

Core Data

- "analysisDate" [timestamp]: The precise (up to milliseconds) UTC timestamp of the measurement

- "condValue" [int]: The "High Water Mark" conductivity value extracted from the raw sensor array; used as the primary metric for sample concentration.

- "usg" [float]: Numerical index (e.g., 1024.8) representing the Urine Specific Gravity predicted by the Kamleon model.

- "pH"[int] Measurement of the sample's acidity/alkalinity, used as a secondary health indicator.

- "score"[int]: A simplified, user-facing hydration metric (e.g., 1–10 scale) calculated from the usg.

- "hematuria_score"[int]: A specialized index used to detect the potential presence of blood in the sample (clinical safety monitoring)

- "colorChart" [int]: A digital representation of the sample's color, traditionally used in clinical "WUT" (Weight, Urine, Thirst) hydration assessments.

- "Prediction_Precision" [str]: reliability of the prediction ("good" or "bad")

```
  # Sensor Arrays

  - "F0","F1","F2","F3","F4","F5","F6","F7","clear" [arrays]: arrays of the values read from the optical sensors.

  - "conductivity", "real", "img" [arrays]: Data arrays representing the Resultant Magnitude, Real (Resistive), and Imaginary (Reactive) components of the sample's complex electrical impedance.



  # Nested Objects

  - "postAssemblyCalibration": metrics used for implementing the calibration of conductivity performed by backend of kamleon

  - "water_reference_8b": reference used as baselines for normalization of the sensor reading.

  - "devices": list of ides linked to the unit (screen and sensor)

  - "model": list of the informationused for feeding the usg models
```

8.3 Expected Responses

- **HTTP 201 Created:** Data accepted. The Sync Service moves the "Checkpoint" forward.
- **HTTP 400 Bad Request:** Invalid input data structure.
- **HTTP 401 Unauthorized:** Invalid or expired OAuth2 token.
