IBIMA Publishing
Communications of International Proceedings
https://ibimapublishing.com/p-articles/47ICT/2026/4725526/
Vol. 2026 (13), Article ID 4725526

A Local Telemetry Backend Prototype on Raspberry Pi 5 for Session-Based Data Synchronization*

Michal KIMBAR, Fryderyk GAJDZIK and Pawel KROWICKI

Wrocław University of Science and Technology, Faculty of Mechanical Engineering, Wroclaw, Poland,

Correspondence should be addressed to: Michal KIMBAR, michal.kimbar02@gmail.com

*Presented at the 47th IBIMA International Conference, 29-30 June 2026, Madrid, Spain

Abstract

In prototype telemetry systems, direct storage of data as CSV files, JSON files, or raw sample dumps is useful during early testing, but it complicates session lifecycle management, file integrity control, event history tracking, and separation between clients and the database layer. The aim of this work was to design and implement a local telemetry backend on Raspberry Pi 5 as an intermediary layer between API clients, a PostgreSQL database, and a local file repository. The prototype was implemented using Python, FastAPI, Uvicorn, and PostgreSQL. A hybrid persistence model was adopted, in which the relational database stores metadata, statuses, relationships, and events, while measurement files remain in the file system. The central element of the solution is a telemetry session model that organizes the process from session initialization, through file upload and SHA-256 integrity checking, to session finalization and preparation of data for further processing. Functional verification covered 30 CSV files divided into three size groups and five negative or boundary cases. All 30 valid uploads satisfied the adopted functional criteria, while the five negative and boundary cases resulted in the expected request rejection or in data not being marked as ready for further analysis.

Keywords: telemetry backend; edge computing; PostgreSQL; session model.

Introduction

Telemetry systems developed around microcontrollers, mobile devices, and local measurement stations often begin with direct sample logging to files. This model is sufficient for early-stage testing, but it becomes restrictive when data must be submitted by multiple clients, associated with a measurement session, and prepared for later analysis.
A measurement file alone does not contain complete information about the data source, session start time, transmission state, write integrity, or relationship with the device and acquisition parameters.

In practice, this leads to coupling between the communication layer and the persistence layer, dependency of clients on the storage structure, lack of a consistent session identifier, and difficulty in reconstructing the processing history. For telemetry data, data origin and the ability to audit information flow are also relevant. Literature on provenance in the Internet of Things (IoT) indicates that tracking the data source, processing history, and transmission events is important for credibility and later use of data (Hu et al., 2020).

Cite this Article as: Michal KIMBAR, Fryderyk GAJDZIK and Pawel KROWICKI Vol. 2026 (13) "A Local Telemetry Backend Prototype on Raspberry Pi 5 for Session-Based Data Synchronization " Communications of International Proceedings, Vol. 2026 (13), Article ID 4725526, https://doi.org/10.5171/2026.4725526

In edge and Industrial Internet of Things (IIoT) environments, local organization of data can reduce dependence on external infrastructure and prepare data for later synchronization or analysis (Shi et al., 2016; Satyanarayanan, 2017; Qiu et al., 2020). This work proposes a local data layer running on Raspberry Pi 5, including a backend service, API, PostgreSQL database, and repository of measurement files. ECG signals and components such as ESP32 or BLE are treated as example data sources; the core problem addressed in this paper concerns the organization of the data flow rather than a specific measurement path.

Technological Background and Related Work

Edge computing is commonly described as moving selected computing and storage functions closer to the place where data are generated, which reduces dependence on cloud infrastructure and may decrease latency (Shi et al., 2016; Satyanarayanan, 2017). In IIoT applications, this also includes local management of data state before further forwarding (Qiu et al., 2020). Classical descriptions of IoT emphasize that such architectures include not only sensors and connectivity, but also service layers, data management, and application integration (Atzori et al., 2010).

A second important context is provenance, understood as tracking the origin of data. In IoT systems, data are heterogeneous, distributed, and prone to integration errors; therefore, recording information about data sources, operations, and processing history is important (Hu et al., 2020). The presented work does not implement a full formal provenance system, but it applies practical mechanisms supporting data traceability: a session model, an event log, file metadata, and integrity checking.

On the communication side, the project is based on the REST paradigm and HTTP semantics (Fielding, 2000; Fielding et al., 2022). The API contract can be described according to the OpenAPI standard (OpenAPI Initiative, 2025). The application layer was implemented using FastAPI and served by Uvicorn (FastAPI Project, 2026; Encode, 2026). File transfer was implemented using the multipart/form-data standard (Masinter, 2015).

For data persistence, PostgreSQL was selected as a relational database intended for storing metadata, relationships, and statuses. This choice follows from the need for referential integrity, transactional consistency, and extensibility of the data model (PostgreSQL Global Development Group, 2026). File integrity was based on SHA-256 checksums compliant with the Secure Hash Standard (National Institute of Standards and Technology, 2015). API security guidelines and zero-trust architecture recommendations also indicate that an intermediary interface should act as a controlled access point to data (OWASP Foundation, 2023; Rose et al., 2020; Fagan et al., 2020).

Against this background, the paper focuses on a local data layer for telemetry, rather than on signal acquisition itself or analytical algorithms. The contribution consists in combining an API, a relational database, a file repository, and a session model into a working prototype of a local edge node.

Aim and Contribution

The aim of this work was to develop and verify a prototype of a local telemetry backend on Raspberry Pi 5 that organizes the intake, persistence, and access to session-based data delivered by API clients. The problem was treated as an engineering task in edge computing, API design, and organization of a persistent data layer.

The scope of the work includes the design and implementation of an HTTP API service, a PostgreSQL relational database, a local file repository, a telemetry session model, file integrity checking, a synchronization event log, and deployment of the service as a process supervised by the systemd mechanism. The measurement layer, client application, and analytical module are treated as elements of the system environment; the main scope of the paper concerns the backend, data model, synchronization workflow, and functional verification.

The main contribution of the work is the design of a coherent model of a local data layer for session-based telemetry. The model combines controlled access through the HTTP API layer, hybrid data persistence, and separation of responsibilities between the client, backend logic, database, and file repository. The central element of the solution is

a telemetry session that organizes the data lifecycle from session creation, through file upload and integrity checking, to finalization and preparation of the material for further processing.

The contribution is architectural and implementation-oriented. It concerns the structuring of telemetry data flow in
a local edge node and the functional verification of the session, metadata, file, and status model.

Proposed Architecture

The architecture was designed as a layered system. At the input side, there are data sources and API clients: a mobile application, a measurement device, an integration script, or a test client. Communication with the backend is performed through an HTTP API. The application logic layer is a FastAPI service running locally on Raspberry Pi 5 (Raspberry Pi Ltd, 2026; FastAPI Project, 2026; Encode, 2026). The persistence layer consists of PostgreSQL and the local file system. The analysis module can use both metadata and source files. The resulting local telemetry backend architecture is shown in Fig 1.

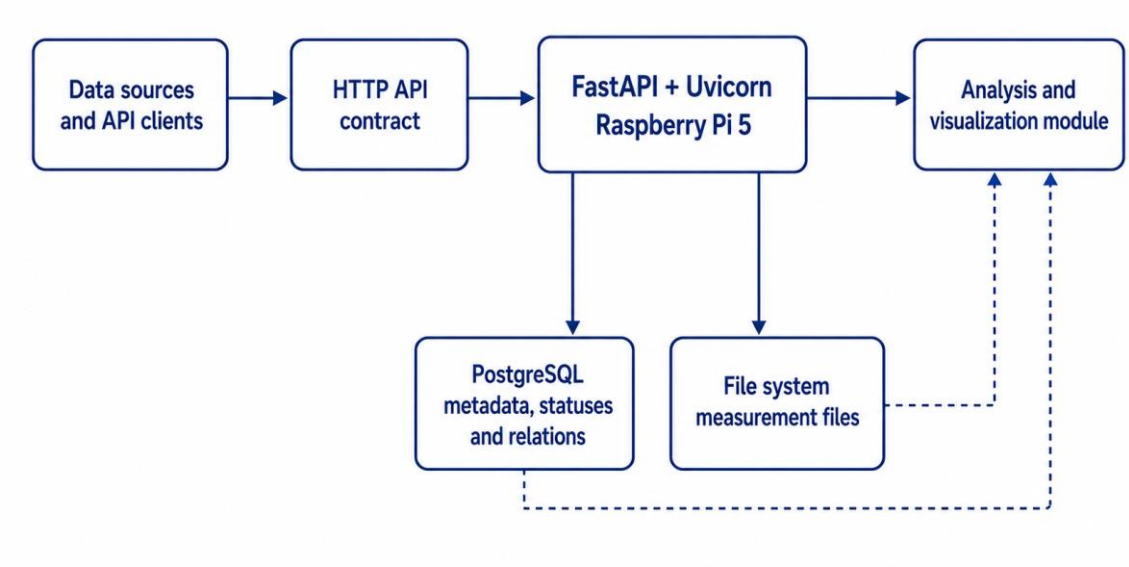

Fig 1. Local telemetry backend architecture with separated API, application logic, and data persistence layers

The API client does not need to know internal relationships, data model versioning, or file locations on the storage medium. It receives an API contract, while request validation, metadata storage, event registration, and status updates remain the responsibility of the backend. This reduces coupling between clients and the internal database structure.

The second architectural assumption was a hybrid data persistence model. Measurement files can be large and are usually analyzed as complete artifacts; therefore, they are stored in the file system. PostgreSQL stores the file description, session association, processing state, and technical metadata. This separation preserves the relational model for information requiring consistency, while avoiding the need to store large signal records as database objects.

PostgreSQL remains a local service and is not exposed directly to clients. Write operations are handled by the backend and protected by an API key mechanism. The access boundary is therefore explicit: the client communicates with the application service, not with the database engine.

Session Model and Synchronization Workflow

The basic unit of data organization is the telemetry session, understood as a single episode of data acquisition and synchronization. A session links the data source, acquisition time, transfer state, file metadata, integrity

control, and later processing stage. As a result, the measurement file is not treated as an isolated artifact, but as an element of a controlled data lifecycle.

The model separates two types of persistence. PostgreSQL stores information requiring consistency, filtering, and relationships: session metadata, statuses, technical events, and results of further processing. The file system stores measurement files because they are larger input artifacts usually analyzed as complete files. This division limits the storage of large signal records in the database while preserving the association between the file and the session context.

The synchronization workflow includes session initialization, file upload, storage in the local repository, SHA-256 checksum calculation, metadata recording, session finalization, and exposure of the data for further processing. File transfer is performed using multipart/form-data (Masinter, 2015), while the SHA-256 checksum is used to verify consistency between the physical file and its description in the database (National Institute of Standards and Technology, 2015).

Separating the stages of the data lifecycle makes it possible to distinguish a started session, a session with a correctly stored file, and a session ready for further analysis. This is relevant in the case of interrupted or repeated transfers, because responsibility for recognizing the process state remains on the backend side.

Prototype Implementation

The prototype was deployed on Raspberry Pi 5 running an operating system from the Raspberry Pi OS family (Raspberry Pi Ltd, 2026). The application layer was developed in Python, using FastAPI as the framework for implementing the HTTP API and Uvicorn as the ASGI server (FastAPI Project, 2026; Encode, 2026). Communication with PostgreSQL was implemented using the psycopg library (The Psycopg Team, 2026). Database connection settings, the data directory, and the API key were separated into environment variables.

The implementation was organized around the session lifecycle rather than exposing the database structure to the client. The API contract includes operations for session creation, file upload, upload finalization, status retrieval, and access to diagnostic information. This makes it possible to maintain a stable client interface even if the internal data model is extended.

File handling is performed in several stages. The backend writes the input stream to a temporary location, checks basic correctness conditions, and then moves the file to the directory assigned to the session. Metadata are persisted in PostgreSQL after the file has been successfully written. This sequence reduces the risk of inconsistency between the database description and the physical presence of the file on disk. Technical synchronization events are recorded in parallel.

Database integrity and indexing mechanisms were used to support the session model, prevent ambiguous associations, and provide consistent access to information about the data state. The database therefore does not act merely as a storage component, but as part of the layer controlling the telemetry lifecycle.

The implementation also includes diagnostic mechanisms used to assess the consistency of the local node. The system checks consistency between metadata and the file repository, correctness of basic states, and the presence of data required for further processing. The service was deployed as a systemd unit, which provides automatic startup after device restart and process supervision (freedesktop.org, 2026).

Functional Verification

The prototype verification was functional in character. Its aim was not to measure performance or perform operational validation, but to check whether the session model and hybrid persistence model support the basic synchronization workflow: receiving data from an API client, creating a session, storing a file, persisting metadata, updating state, and retrieving information about the synchronization process. The verification was performed on 30 CSV files representing test measurement sessions. The set included three size groups: 10 small files of approximately 0.5-1 MB, 10 medium files of approximately 2-5 MB, and 10 larger files of approximately 8-15 MB. Data were submitted from an application client, an integration script, and HTTP tools used to generate controlled requests.

For each valid workflow, three aspects were checked: the session lifecycle, data persistence consistency, and data availability for the analytical module. This included session creation, file acceptance and finalization, status retrieval, file presence in the repository, file-size consistency, metadata storage, SHA-256 checksum consistency, and availability of the event history.

In addition, five negative and boundary tests were performed: missing API key, invalid API key, missing file in the request, invalid data format, and repeated transfer for an existing session. These tests checked whether the backend distinguishes correct synchronization from basic integration errors and whether incomplete data are prevented from being marked as ready for further processing.

All 30 valid uploads satisfied the adopted functional criteria. Consistency was achieved between files stored in the file system and metadata persisted in PostgreSQL, including file-size consistency and SHA-256 checksum consistency. The five negative and boundary tests resulted in the expected request rejection or in data not being marked as ready for analysis. This result confirms the correctness of the main functional scenario, but it does not constitute a measurement of throughput, scalability, or resilience under multi-client load.

Discussion and Limitations

The main outcome of the work is the transition from simple file-based storage to a model in which the telemetry session serves as the central organizational object. This makes it possible to associate measurement data with the source, synchronization state, metadata, integrity control, and later processing.

Another relevant aspect is the separation of API clients from PostgreSQL. The client uses the API contract, while validation logic, metadata storage, status updates, and event registration remain on the backend side. This facilitates further extension of the data model and reduces coupling between client applications and the internal database structure.

The proposed model can be adapted to other types of telemetry data because the basic synchronization workflow is not dependent on a specific input signal. It requires at least the association of a file or data packet with a session, metadata, transfer state, and an integrity control mechanism.

The verification confirmed the correctness of the main functional scenario, but it did not include throughput, scalability, or resilience testing under multi-client load. Another limitation of the work is the lack of a quantitative comparison between the adopted hybrid model and alternative persistence variants, such as storing files as large database objects or using a purely file-based archival model. The API key mechanism should be treated as a basic access-control layer in a test environment. Further work should include stronger authentication, scope-based authorization, load testing, formalization of disaster recovery procedures, and integration with the target data analysis module.

Conclusions

This paper presented a prototype of a local telemetry backend on Raspberry Pi 5 for session-based data synchronization. The solution combines an HTTP API, a PostgreSQL relational database, a local file system, and a telemetry session model.

The main result of the work is the separation of responsibilities between the API client, backend, database, and file repository. In the proposed model, the client does not communicate directly with PostgreSQL, measurement files are stored in the file system, and the database persists metadata, statuses, relationships, and events. This makes it possible to associate a raw file with the session, data source, integrity checksum, and synchronization history.

Functional verification confirmed the correctness of the scenario including session creation, upload of 30 CSV files in three size groups, metadata persistence, SHA-256 calculation, finalization, and status and event retrieval. The solution therefore structures telemetry data flow around sessions, metadata, integrity control, and controlled API access.

References

- Atzori, L., Iera, A. and Morabito, G. (2010), 'The Internet of Things: A Survey', Computer Networks, vol. 54, no. 15, pp. 2787-2805, doi: 10.1016/j.comnet.2010.05.010.
- Encode (2026), Uvicorn Documentation. [Online]. Encode. [Accessed 2026-05-13]. Available at: https://uvicorn.dev/.
- Fagan, M., Megas, K. N., Scarfone, K. and Smith, M. (2020), Foundational Cybersecurity Activities for IoT Device Manufacturers, NIST Interagency/Internal Report 8259, National Institute of Standards and Technology, doi: 10.6028/NIST.IR.8259.
- FastAPI Project (2026), FastAPI Documentation. [Online]. FastAPI. [Accessed 2026-05-13]. Available at: https://fastapi.tiangolo.com/.
- Fielding, R. T. (2000), Architectural Styles and the Design of Network-Based Software Architectures, PhD thesis, University of California, Irvine, Available at: https://ics.uci.edu/~fielding/pubs/dissertation/top.htm.
- Fielding, R. T., Nottingham, M. and Reschke, J. (2022), HTTP Semantics, RFC 9110, Internet Engineering Task Force, doi: 10.17487/RFC9110.
- freedesktop.org (2026), systemd.service Manual Page. [Online]. freedesktop.org. [Accessed 2026-05-13]. Available at: https://www.freedesktop.org/software/systemd/man/systemd.service.html.
- Hu, R., Yan, Z., Ding, W. and Yang, L. T. (2020), 'A Survey on Data Provenance in IoT', World Wide Web, vol. 23, no. 3, pp. 1441-1463, doi: 10.1007/s11280-019-00746-1.
- OpenAPI Initiative (2025), OpenAPI Specification Version 3.2.0. [Online]. OpenAPI Initiative. [Accessed 2026-05-13]. Available at: https://spec.openapis.org/oas/v3.2.0.html.
- Masinter, L. (2015), Returning Values from Forms: multipart/form-data, RFC 7578, Internet Engineering Task Force, doi: 10.17487/RFC7578.
- OWASP Foundation (2023), OWASP API Security Top 10 - 2023. [Online]. OWASP Foundation. [Accessed 2026-05-13]. Available at: https://owasp.org/www-project-api-security/.
- PostgreSQL Global Development Group (2026), PostgreSQL Documentation. [Online]. PostgreSQL Global Development Group. [Accessed 2026-05-13]. Available at: https://www.postgresql.org/docs/.
- Qiu, T., Chi, J., Zhou, X., Ning, Z., Atiquzzaman, M. and Wu, D. O. (2020), 'Edge Computing in Industrial Internet of Things: Architecture, Advances and Challenges', IEEE Communications Surveys & Tutorials, vol. 22, no. 4, pp. 2462-2488, doi: 10.1109/COMST.2020.3009103.
- Raspberry Pi Ltd (2026), Raspberry Pi Documentation. [Online]. Raspberry Pi Ltd. [Accessed 2026-05-13]. Available at: https://www.raspberrypi.com/documentation/.
- Rose, S., Borchert, O., Mitchell, S. and Connelly, S. (2020), Zero Trust Architecture, Special Publication 800-207, National Institute of Standards and Technology, doi: 10.6028/NIST.SP.800-207.
- Satyanarayanan, M. (2017), 'The Emergence of Edge Computing', Computer, vol. 50, no. 1, pp. 30-39, doi: 10.1109/MC.2017.9.
- Shi, W., Cao, J., Zhang, Q., Li, Y. and Xu, L. (2016), 'Edge Computing: Vision and Challenges', IEEE Internet of Things Journal, vol. 3, no. 5, pp. 637-646, doi: 10.1109/JIOT.2016.2579198.
- National Institute of Standards and Technology (2015), Secure Hash Standard (SHS), FIPS 180-4, National Institute of Standards and Technology, doi: 10.6028/NIST.FIPS.180-4.
- The Psycopg Team (2026), Psycopg 3 Documentation. [Online]. The Psycopg Team. [Accessed 2026-05-13]. Available at: https://www.psycopg.org/psycopg3/docs/.