Extraction and Orchestration of Data in Azure Applied to the Business (NMBS)

Master's Thesis

Master in Data Science, Big Data & Business Analytics (2024–2025)

Judith Castillo Martínez

Universidad Complutense de Madrid

Contents

1 Executive Summary

This work develops a data platform in the Microsoft Azure cloud, oriented to the ingestion, processing and exploitation of operational railway information from NMBS (De Nationale Maatschappij der Belgische Spoorwegen (NMBS) — in French Société Nationale des Chemins de fer Belges (SNCB) — the National Railway Company of Belgium). The main motivation arises from the need to improve service punctuality, optimise operational resources and reinforce transparency in communication with passengers and with the different internal stakeholders of the organisation.

To achieve these objectives, a governed data pipeline has been designed that leverages the distributed computing capacity of Azure Databricks (Apache Spark), the scalable and secure storage of Azure Data Lake Storage Gen2, and orchestration through Azure Data Factory. The final exploitation of the information is supported by Azure Cosmos DB with MongoDB API as a low-latency NoSQL document database, which facilitates fast querying of the processed data and opens the door to its integration into internal applications and executive dashboards.

2 Context and Objectives

2.1 Project Justification

In the railway domain, punctuality is one of the most critical indicators of service quality. Operating companies need to monitor, in near real time, both incidents and the use of network capacity. Having accurate and up-to-date information is essential not only to improve planning and support strategic decision-making, but also to ensure a more reliable and safer service for passengers.

In this context, having a modern cloud data platform makes it possible to centralise, process and serve information in an agile manner, reducing response times to incidents and improving communication with both the end user and internal teams.

2.2 Specific Objectives

The general objective of this project is to design and implement a governed data pipeline in Azure, capable of transforming raw and heterogeneous data into accessible, high-quality information. To this end, the following specific objectives are defined:

- Build an ingestion and processing flow based on Azure Databricks (Spark) that guarantees data quality and consistency.
- Implement a layered architecture (RAW, CLEAN, Serving) that ensures data traceability, scalability and reusability.
- Expose data through an Azure Cosmos DB service (API for MongoDB), replacing local solutions and eliminating dependencies on tunnels or manual configurations.
- Facilitate information exploitation through fast queries, internal applications and future integrations with visualisation tools (Power BI, Tableau).
- Automate the pipeline to allow periodic loads, ensuring that the information is always up to date.

These objectives are supported by the skills acquired in the master's programme, in particular:

- Distributed processing with Spark (Databricks).
- Modelling and querying of NoSQL databases.
- Development in Python for modular implementation and automation of notebooks.
- Use of Big Data technologies in the cloud for resource provisioning and process orchestration.

2.3 Stakeholders and Impacted Areas

The main stakeholder groups that would benefit from this project are:

- Operations Management, which requires reliable indicators for managing punctuality and optimising services.
- Planning and Railway Safety teams, which use data to detect bottlenecks and prevent risks.
- Customer Service area, which improves communication with passengers through validated and accessible information in near real time.
- Technology and Business Intelligence units, which find in the pipeline a consistent data base on which to build reports, dashboards and analytical models.

2.4 Note on the Data and NDA with NMBS

The data used in this work correspond to the GTFS (General Transit Feed Specification) standard, provided by NMBS/SNCB (National Railway Company of Belgium). Access to these datasets has been carried out under a Non-Disclosure Agreement (NDA), which restricts their use exclusively to the academic development of this project.

The purpose of using these data is strictly didactic and experimental, aimed at demonstrating the feasibility of a modern cloud data pipeline. No commercial exploitation nor public dissemination is envisaged beyond the academic framework of this Master's Thesis.

In a potential production scenario, data would be obtained from open portals, public APIs or specific information supply agreements, always ensuring regulatory, contractual and data security compliance.

3 Scope and Deliverables

The scope of the project has covered the complete implementation of the life cycle of a modern data pipeline, following the best practices of the medallion architecture (Bronze, Silver, Gold).

In the RAW (Bronze) layer, the ingestion of the GTFS files provided by NMBS/SNCB has been defined. These files, in text format (.txt), are automatically downloaded from the data portal and stored in Azure Data Lake Storage Gen2, converted into Delta Lake tables. This approach ensures data persistence, versioning and the ability to replay processes if necessary.

In the CLEAN (Silver) layer, several notebooks have been developed in Azure Databricks, each oriented to a specific GTFS table. At this stage, essential quality rules have been applied: removal of nulls in key fields (stop_id, trip_id, route_id), removal of duplicates, validation of time formats in HH:MM:SS notation, normalisation of geographic coordinates, standardisation of text in descriptive fields and type conversions. As a result, the data are cleansed, coherent and ready for reliable analysis.

Finally, in the GOLD (Serving) layer, a loading process has been implemented towards a cloud-managed document database, specifically Azure Cosmos DB with API for MongoDB. At this stage, the clean data are exported from Databricks to Cosmos DB collections, ensuring idempotency through the generation of deterministic identifiers (_id) from natural keys. In addition, column names have been adapted to ensure compatibility with MongoDB (avoiding characters such as "." or "$"). This change eliminates the dependency on external tunnels (ngrok) and local Docker deployments, and provides greater robustness, security and scalability to the pipeline.

The main deliverables of this work include:

- An auto-ingestion notebook responsible for daily download from the NMBS/SNCB web source and storage in RAW.
- A general ingestion notebook, responsible for converting the files to Delta Lake and organising them in the RAW zone.
- The specific cleaning notebooks per table (12 in total), oriented to the cleansing and transformation of each GTFS dataset.

- A notebook for loading into Cosmos DB (03_load_to_cosmos), which exposes the clean tables as document collections ready for querying.
- A Databricks Job, configured to orchestrate all phases in a sequential and traceable manner (Ingestion $\rightarrow$ Cleaning $\rightarrow$ Load).
- The technical documentation, which details the transformation rules applied, the architecture design and the configuration required to execute the pipeline, including aspects of parameterisation and security.

4 Expected Results

The developed platform provides an orchestrated and reproducible ETL pipeline in Azure Databricks, capable of transforming raw data into cleansed information served in near real time.

The expected results can be summarised in three dimensions:

- **Data quality**: by applying cleaning and normalisation processes in the Silver layer, inconsistencies, nulls and duplicates are removed, and compliance with formats and ranges is validated. This ensures that the data used in analyses and applications are reliable and traceable.
- **Flexibility and scalability**: the use of widgets and parameterisable configurations in the notebooks allows adjusting storage paths, databases and target collections, facilitating the reuse and extension of the pipeline to new GTFS, XML or API sources. Moreover, the integration with Cosmos DB (MongoDB API) eliminates the dependency on local solutions such as ngrok, enabling a native cloud connection with managed scalability.
- **Information exploitation**: publishing in Cosmos DB enables consumption by internal applications, APIs (FastAPI) and executive dashboards in tools such as Tableau or Power BI. As a result, the organisation gains visibility over service status, improves planning processes and offers more transparent communication to passengers.

In the long term, the proposed architecture offers the possibility of integrating more real-time sources, automating more advanced quality controls and enabling an analytics ecosystem in which operational queries, executive reports and predictive models of punctuality coexist.

5 Reference Architecture in Azure

The architecture designed in this project is based on a modern cloud data platform approach, following Microsoft's recommendations for analytical architectures in Azure and adapting them to the specific use case of rail transport.

The design responds to the need for a governed, scalable and secure ETL flow that allows moving from raw data to information useful for operations and decision-making, while complying with good security practices and the ISO/IEC 27001 and 27017 standards for cloud environments.

5.1 Data Sources

The main data sources integrated correspond to the GTFS (General Transit Feed Specification) standard, which describes information related to timetables, routes, stops and trips in flat files (.txt). These datasets come from open portals (e.g. NMBS/SNCB) and constitute the operational basis of the pipeline.

In addition to GTFS, the architecture is prepared to admit additional sources such as:

- XML files with hierarchical structures, useful for enriching information with incidents or service conditions.
- REST/JSON APIs, which allow capturing near real-time data such as weather conditions or the state of rail traffic.

All these sources are initially incorporated in raw form, without transformation, in the cloud storage repository.

5.2 Data Zones (Bronze, Silver, Gold)

The medallion architecture pattern has been adopted, organising data into three layers and facilitating governance:

- **Bronze (RAW)**: contains the data as they arrive from the sources, stored in Delta format in Azure Data Lake Storage Gen2. This layer ensures traceability and versioning.
- **Silver (CLEAN)**: stores the data after cleaning and normalisation processes, removing duplicates, validating formats and applying quality rules.
- **Gold (Serving/Applications)**: corresponds to the exploitation layer. In this project, the Silver data are loaded into document collections in Azure Cosmos DB (MongoDB API), which allows feeding internal applications, APIs (FastAPI) and dashboards in tools such as Power BI or Tableau.

This design ensures that transformation processes do not alter the original data and enables different consumption levels according to needs.

5.3 Azure Services and Roles

The architecture is based on several key Azure ecosystem services:

- **Azure Data Lake Storage Gen2 (ADLS)**: central data repository, organised into RAW and CLEAN zones.
- **Azure Databricks**: distributed processing engine, where the ingestion, cleaning and loading notebooks are developed, as well as orchestration through Jobs.
- **Azure Data Factory (ADF)**: in an extended deployment, it can manage orchestration at pipeline level, integrating Databricks with other sources or systems.
- **Azure Cosmos DB (MongoDB API)**: managed cloud service that replaces the dependency on local Mongo and ngrok, providing a robust, scalable and secure Serving layer.
- **Azure Key Vault**: credentials and secrets manager, essential in production environments to protect connection URIs and passwords.

The roles of each service are clearly differentiated: ADLS as storage, Databricks as processing, ADF as orchestration and Cosmos DB as managed document database.

5.4 Security and Compliance Design

Security is a cross-cutting aspect throughout the architecture:

- Access to ADLS is regulated through Azure RBAC (Role-Based Access Control) and container permissions.
- Connections to Cosmos DB are encrypted in transit (SSL/TLS) and protected with user authentication and keys managed in Key Vault.
- Databricks uses Service Principals for authentication instead of personal credentials.

This design follows practices compatible with ISO/IEC 27001 (information security) and ISO/IEC 27017 (security controls for cloud services): encryption in transit and at rest, separation of environments, access auditing and use of managed secrets.

Regarding compliance, although GTFS data are public, the architecture is prepared to handle sensitive data if new sources are incorporated, applying anonymisation, masking and access auditing with Azure Purview.

5.5 Observability and Costs

Pipeline observability is achieved with native Databricks mechanisms (execution logs, process metrics, errors), which can be integrated into Azure Monitor and Log Analytics to consolidate performance and quality indicators.

In terms of costs, the main resources consumed are:

- **ADLS Gen2**: low storage cost.
- **Databricks**: the most expensive resource due to Spark cluster usage, mitigated with auto-termination and autoscaling policies.
- **Cosmos DB (MongoDB API)**: cost associated with the chosen plan (serverless or provisioned throughput). For this academic project, the serverless mode ensures reduced costs and pay-per-use.
- Possible deployment costs of the API in the future as the solution grows, if applicable.

Several measures have been applied to optimise and control service maintenance costs without incurring disproportionate expenses for the scope of this project:

- Use of clusters sized to the current needs of the project.
- Consolidation of data contained in the files using `COALESCE(1)`.
- Avoiding indefinite use of resources once the corresponding execution process has finished, in order to prevent high costs and inefficient resource usage.

6 Pipeline Design (ETL)

In this project, an explicit ETL (Extract, Transform, Load) approach has been chosen instead of ELT. The main reason is that transformations and quality validations are carried out in the Databricks (Apache Spark) distributed processing layer, before loading the data into the document database. In this way, it is possible to ensure and guarantee that the data contained in the previously processed files, which will be used for exploitation in later phases, are cleansed and consistent after the cleaning process.

This avoids delegating certain functionalities or responsibilities to the NoSQL database (Cosmos DB), whose function in the architecture is limited to acting as a low-latency query and consumption repository.

This approach makes it possible to:

- Ensure end-to-end traceability and quality.
- Leverage Spark's power to efficiently apply complex transformations and business rules.
- Keep the document database lightweight and exclusively oriented to consumption, avoiding overloading it with processing tasks that are not part of its purpose.

Thus, the pipeline follows the logic of extracting data from the sources (Extract), transforming them in the CLEAN zone (Transform) and, finally, loading them into MongoDB/Cosmos DB (Load) for exploitation in applications and dashboards.

6.1 Ingestion (RAW Zone)

The first stage of the pipeline corresponds to the RAW layer, whose main function is the ingestion and persistence of data in a format as close as possible to the original. In this case, the input files belong to the GTFS (General Transit Feed Specification) standard provided by NMBS/SNCB, in text format (.txt). Among the files included are (see Annex 1):

- `agency.txt`, `calendar.txt`, `calendar_dates.txt`, `feed_info.txt`, `routes.txt`, `stop_times.txt`, `stop_time_overrides.txt`, `stops.txt`, `trips.txt`, `transfers.txt` and `translations.txt`.

Ingestion is implemented in Databricks using PySpark, leveraging its schema inference and automatic header validation capabilities. The files are converted and stored in Delta Lake format within Azure Data Lake Storage Gen2, which allows for historical, versioned storage optimised for subsequent queries.

From an organisational point of view, ingestion has been developed in a single notebook (`01_ingesta`) that centralises data loading and writing in the RAW layer. This decision adheres to the divide and conquer principle: it avoids mixing later transformations in this phase and keeps the flow clearer and easier to debug. Once stored in RAW, the data are available for use in the subsequent cleaning and transformation stages.

6.2 Cleaning and Transformations (CLEAN Zone)

The second stage of the pipeline corresponds to the CLEAN layer, whose purpose is to cleanse the data stored in RAW, normalise them and ensure their consistency. This phase is fundamental, as it guarantees that the data exposed to subsequent applications and analyses comply with basic quality and coherence rules.

Transformations are performed in PySpark (DataFrames) and have been split into different notebooks, one for each GTFS table, which facilitates modularity and error detection (see Annex 2).

The main cleaning rules applied include:

- Removal of null values in fields acting as primary keys (stop_id, trip_id, route_id).
- Removal of duplicate records according to unique identifiers.
- Normalisation of text strings, for example standardising stop names.
- Data type conversions, such as casting coordinates (stop_lat, stop_lon) to `double` or categorical fields (direction_id, transfer_type) to `int`.
- Validation of time formats in `arrival_time` and `departure_time` fields using the HH:MM:SS pattern.
- Validation of geographic ranges for latitude and longitude coordinates.

As output of this process, clean Delta tables are generated and stored in the CLEAN path of the Data Lake, organised by table. Each notebook reads directly from the RAW zone and writes to the CLEAN zone, which ensures a clear separation of responsibilities and facilitates pipeline maintenance. This modular structure (one notebook per GTFS file) allows for more efficient and precise orchestration within the global Job.

6.3 Loading into Document Database (Azure Cosmos DB – MongoDB API)

Once the data have been cleansed, the loading phase towards the service or Serving Layer is performed, implemented in this project through Azure Cosmos DB with API for MongoDB.

The objective of this stage is to make the clean data available in an accessible, low-latency format, suitable for operational queries, APIs and consumption applications.

To this end, the official `mongo-spark-connector` is used, which allows writing directly from Spark to Mongo collections. Each Delta table in the CLEAN layer is transformed into an independent collection within the Cosmos DB database.

During the loading process:

- Deterministic identifiers (_id) are generated by applying hash functions (`sha2`) over combinations of key columns, ensuring idempotency in reloads (see Annex 3).
- Column names are adapted to comply with MongoDB restrictions (avoiding characters such as "." or "$") (see Annex 4).
- The level of parallelism is controlled (`coalesce(1)`) to optimise writes in Cosmos, since this service penalises excessive concurrent connections.

This entire flow has been centralised in a single notebook (`03_load_to_cosmos`), dependent on the correct completion of the cleaning stage.

The use of managed Cosmos DB replaces the previous need to deploy MongoDB locally and open tunnels with ngrok, providing security, scalability and a cost model adjusted to real consumption.

7 Orchestration and Operation

The orchestration of the complete pipeline has been implemented through a Job in Databricks, designed to guarantee sequential and traceable execution of all stages. The flow contemplates the following phases:

- Zero stage (automatic download): daily retrieval of GTFS files from the NMBS/SNCB website and direct loading into the RAW zone of the Data Lake.
- Ingestion (RAW): execution of the `01_ingesta` notebook, which stores the data in Delta format in the RAW layer.
- Cleaning (CLEAN): execution, either in parallel or sequentially, of the `02_clean_**` notebooks, each dedicated to one of the GTFS tables.
- Loading in Cosmos DB (Serving): execution of the `03_load_to_cosmos` notebook, dependent on the correct completion of the cleaning processes.
- Exploitation through FastAPI: deployment of a lightweight service that queries the data in Cosmos DB and exposes them to internal applications or dashboards.

In this way, it is ensured that the data follow a logical flow from initial ingestion to final exposure, with explicit dependencies that prevent inconsistencies or incomplete executions.

7.1 Sequential Execution in Databricks

The Job chains: Zero stage → Ingestion → Cleaning → Publication → Exploitation (FastAPI). The dependencies ensure order and controlled recovery in case of failure.

7.2 Connection with Cosmos DB (Mongo API)

In this version of the pipeline, the initial provisional solution based on local MongoDB + ngrok has been replaced by a deployment in Azure Cosmos DB with API for MongoDB (see Annex 7).

This brings significant advantages:

- Elimination of external tunnels and exposed ports.
- Enhanced security and regulatory compliance, thanks to native integration with Azure.
- Scalability and centralised monitoring, facilitating a future move to production.

The `03_load_to_cosmos` notebook uses the official `mongo-spark-connector`, configured with the URI provided by Cosmos, PLAIN authentication and optimisation parameters for distributed loads.

7.3 Periodic Scheduling of the Pipeline

The Databricks Job has been scheduled to run daily at 2:00 a.m. using the native scheduler (see Annex 10). With this configuration:

- Data in Cosmos DB are automatically updated.
- Dependence on human intervention is reduced.
- Consistency and traceability in daily updates are ensured.

7.4 FastAPI: Data Exploitation

With the aim of enabling simple and standardised access to the data processed in the GOLD layer, a REST API has been developed using the FastAPI framework in Python. This API exposes, in a controlled manner, the information stored in the document database (Azure Cosmos DB with MongoDB API), which allows internal applications and dashboards to consume the data without needing direct access to the storage system.

API Architecture (see Annex 11)

- **Connection to the database**: managed through environment variables (`MONGO_URI`, `DB_NAME`), ensuring good security practices and facilitating deployments in different environments.
- **Data models (Pydantic)**: define the structure of API responses, automatically validating documents obtained from MongoDB.
- **REST endpoints**: allow querying stops, routes, trips, timetables and transfers in a simple way.

Main Models

- **Route**: route information (id, agency, names, colours).
- **Trip**: definition of a trip associated with a route and a service.
- **Stop**: stop data with name, coordinates and location type.
- **StopTime**: arrival and departure times at each stop for a trip.
- **Transfer**: information on transfers between stops.

Examples of Implemented Endpoints

- `/routes` → returns the list of routes.
- `/stops` → list of stops with coordinates.
- `/trips/{route_id}` → trips associated with a route.
- `/trips/{trip_id}/stops` → ordered stops of a trip.
- `/search/stops?name=X` → search for stops by name.
- `/transfers/{stop_id}` → transfers from a stop.

Benefits

- Facilitates external consumption of data (mobile applications, dashboards, integrations).
- Provides automatic OpenAPI/Swagger documentation, useful for development teams.
- Enables scaling towards an API-first architecture, in which different applications rely on a common data backend.

8 Conclusions

The development of this project has made it possible to build a modern data pipeline in Azure, with a modular architecture based on RAW, CLEAN and Serving zones. Throughout the work, both the technical feasibility of the solution and its alignment with the needs of the railway sector in terms of punctuality, data quality and exploitation capacity have been validated.

One of the main achievements of the project has been the construction of an orchestrated and reproducible flow that guarantees end-to-end traceability: from raw data ingestion to their exposure in a document database and exploitation through an API. This approach not only ensures consistency, but also facilitates scalability, since each stage is autonomous and parameterisable.

During the technical discussion, strengths and limitations have been identified. Among the strengths are:

- The use of Delta Lake as storage format, providing versioning and time-travel capabilities, which are very useful in historical data environments.
- The modular division of notebooks, which facilitates debugging and orchestration in Databricks Jobs.
- The implementation of explicit quality rules (removal of nulls, validation of coordinates and times), which increase the reliability of the processed data.
- The integration of data with Azure Cosmos DB (MongoDB API), which eliminates the need for external tunnels and guarantees greater security and scalability.
- The deployment of a FastAPI service to expose data as REST services, which enables direct consumption in applications, integrations or dashboards.

However, improvement areas have also been detected. Although Cosmos DB solves the connection issues of the first version, it is advisable to strengthen the automation of quality checks. A natural evolution would be to integrate a more advanced data quality framework (such as Deequ or rules in Azure Data Factory), with alerts and validations before publishing data in the service layer.

From a business point of view, the developed pipeline lays the foundations for improving the monitoring of punctuality, incident detection and resource planning. Exposing the data through FastAPI increases accessibility and allows building internal applications or near real-time dashboards, which improves the organisation's transparency and responsiveness.

In overall terms, the project has shown that it is possible to build a scalable and governed cloud data pipeline with relatively accessible resources. The proposed architecture is flexible and easily extendable to new sources, and can serve as a basis for future advanced analytics initiatives, including delay prediction models or demand analysis.

Finally, this work opens several lines of future research:

- Automation and standardisation of data quality rules.
- Integration with corporate visualisation and BI systems (Power BI, Tableau).
- Extension towards real-time data sources (traffic APIs, railway IoT).
- Evolution towards an API-first approach, where FastAPI is the central data access layer.

In short, the developed pipeline represents an important step towards building a modern data platform in the railway sector, and lays the foundations both to improve daily operations and to enable a more ambitious analytics strategy in the future.

9 Bibliography

9.1 Official Documentation of the Technologies Used

- Databricks. (n.d.). Delta Lake Documentation. Retrieved from `https://docs.databricks.com/delta`
- Microsoft Azure. (n.d.). Azure Data Lake Storage Gen2 Documentation. Retrieved from `https://learn.microsoft.com/azure/storage/blobs/data-lake-storage-introducti`
- MongoDB. (n.d.). MongoDB Manual. Retrieved from `https://www.mongodb.com/docs/`
- Azure Cosmos DB. (n.d.). Cosmos DB API for MongoDB Documentation. Retrieved from `https://learn.microsoft.com/azure/cosmos-db/mongodb`
- Apache Spark. (n.d.). Spark SQL, DataFrames and Datasets Guide. Retrieved from `https://spark.apache.org/docs/latest/sql-programming-guide.html`
- FastAPI. (n.d.). FastAPI Documentation. Retrieved from `https://fastapi.tiangolo.com/`

9.2 Other Documentation Provided in the Master's Contents

- Universidad Complutense de Madrid (2024–2025). Teaching material for the subject "Spark". Master in Data Science, Big Data & Business Analytics 2024–2025 UCM.
- Universidad Complutense de Madrid (2024–2025). Teaching material for the subject "Big Data Technologies". Master in Data Science, Big Data & Business Analytics 2024–2025 UCM.
- Universidad Complutense de Madrid (2024–2025). Teaching material for the subject "Python, modules 1 and 2".

- Universidad Complutense de Madrid (2024–2025). Teaching material for the subject "NoSQL".

The principles applied in the pipeline design respond to the contents taught in the Official Master's in Big Data & Business Analytics, particularly in the subjects of Big Data Technologies for cloud data processing and Distributed Processing with Spark. The recommended methodologies and practices in the teaching material have been followed, which has ensured both technical validity and alignment with academic and professional standards.

10 Annexes

10.1 Annex 1. Architecture in Azure

Diagram of the general architecture (Data Lake, Databricks, Cosmos DB, FastAPI).

It represents the three layers of the pipeline (RAW, CLEAN, GOLD).

10.2 Annex 2. Ingestion and Storage

- Screenshot of the `00_autoloader_ingesta_gtfs` notebook (with the code that downloads the GTFS files).
- Example of the GTFS files in Azure Data Lake (RAW container).

10.3 Annex 3. Cleaning and Transformation

- Screenshot of a cleaning notebook (`stop_times`) as an example.
- Example of a Delta table in Databricks after cleaning.

10.4 Annex 4. Publication in Cosmos DB

- Screenshot of the code that writes from Spark to Cosmos DB.
- Image of the database in Azure Cosmos DB Explorer showing the created collections.

10.5 Annex 5. Orchestration

- Screenshot of the Databricks Job configuration that chains the notebooks.
- Example of the log of a successful execution.

10.6 Annex 6. FastAPI

- Screenshot of the API code (example: `/stops` endpoint).
- Test in Swagger UI (FastAPI interactive documentation) showing a working endpoint.

Documentation route:
`https://pruebasnmbs-hdd0debbdabsg8ga.westeurope-01.azurewebsites.net/docs`

10.7 Annex 7. Automation

- Image of the Databricks Job scheduler with the task configured at 2 a.m.

10.8 Annex 8. GitHub Repository

Image of the repository with the folder structure: `01_ingesta`, `02_clean`, `03_load_to_cosmos`, `fastapi/`.

Repository information:

- The repository is private and access has been granted to Santiago Mota and Carlos Ortega at the following email addresses: `smota.clases@gmail.com` and `cof@qualityexcellence.`
- Link: `https://github.com/Trolobubu/TFM-Pipeline-ETL-NMBS`

Video:
`https://drive.google.com/file/d/135acukN-sFDi6IlypIjdtcztF6ELjse-/view?usp=sharing`