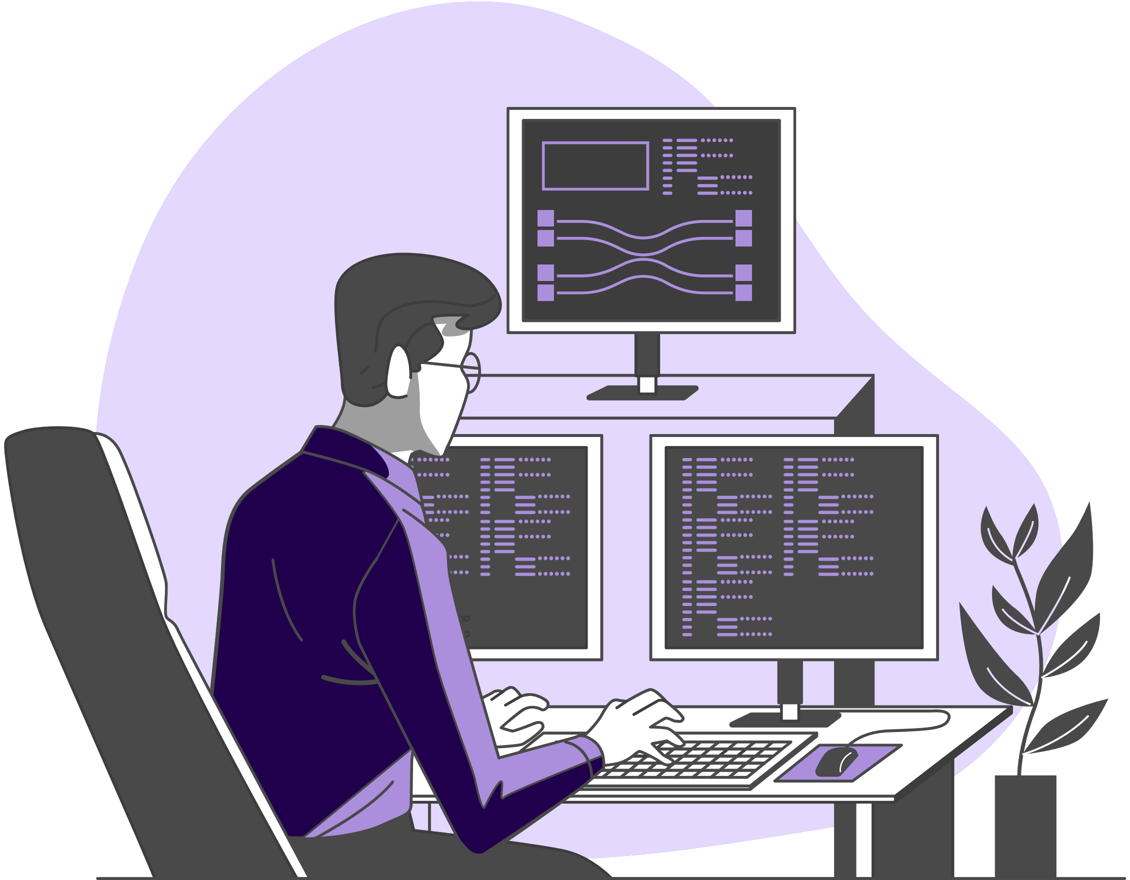


API_ASSET #TYPEDB

Matteucci Giacomo
Monti Yuri
Sorritelli Greta

Our Team

Giacomo Matteucci

giacomo.matteucci@studenti.unicam.it

Greta Sorritelli

greta.sorritelli@studenti.unicam.it

Yuri Monti

yuri.monti@studenti.unicam.it

Project Description

EXPECTED RESULTS	REALIZATION
Analysis of the graph database features (e.g. schema validation, forms, etc.)	✓
Comparison with respect to ACID properties	✓
Schema checking and type checking over properties	✓
Concurrency	?
Role and permissions	✗
Implementation of the prototype (e.g. node.js)	✓

Table of Contents

01 **TypeDB**
Features and ACID properties

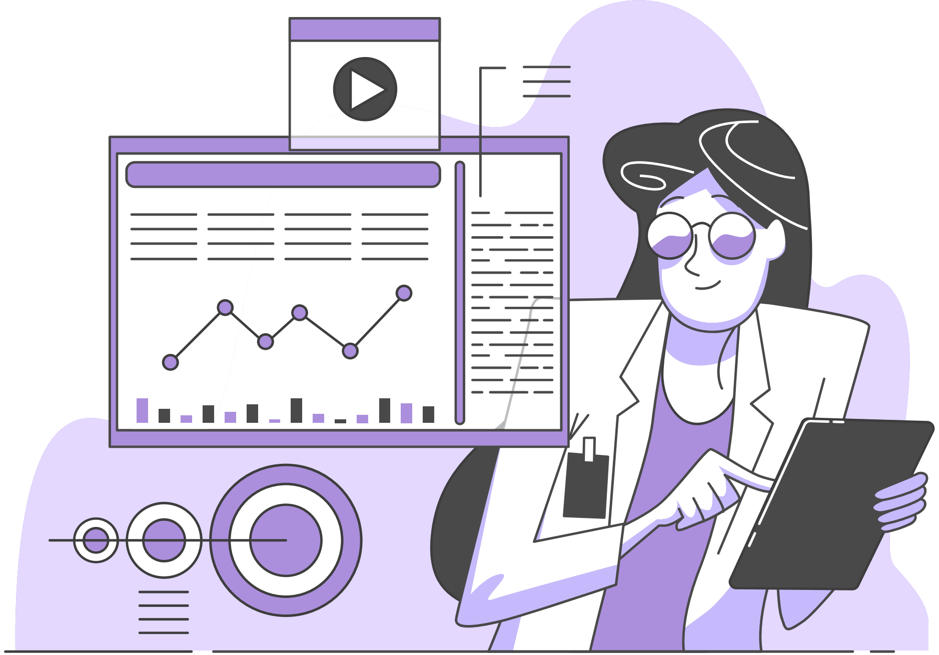
02 **TypeDB vs other DB**
Comparison with other graph databases and relational model

03 **Digital Twins and Ditto CRUD APIs**
What is a Digital Twin and how it works, Ditto APIs and our APIs

04 **Other Technologies used**
TypeDB Studio, Node.js,
Docker and Postman

05 **Technical Implementation**
Technical details about our implementation and architecture

06 **Conclusions and Further Work**
What we have done and what could be implemented in future



01

TypeDB

A strongly-typed database,
based on a logical type system

TypeDB in a Nutshell

TypeDB is a strongly-typed database, based on a logical type system.
It guarantees data integrity and safety, giving a higher level of expressivity that simplifies the work and tackles domains that seemed too complex before.

Features

Types

- Entity
- Relation
- Attribute

Schema

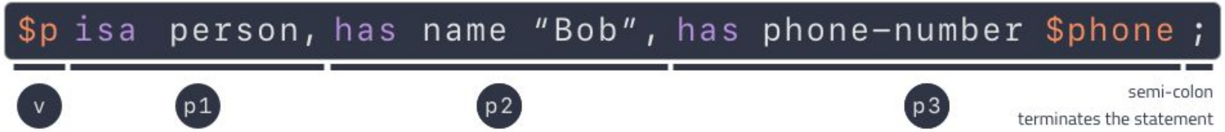
- type-checking
- logic validation of queries
- type-inference
- rule-inference

Query Pattern Anatomy

- Statement structure
- Pattern structure

Query Pattern Anatomy

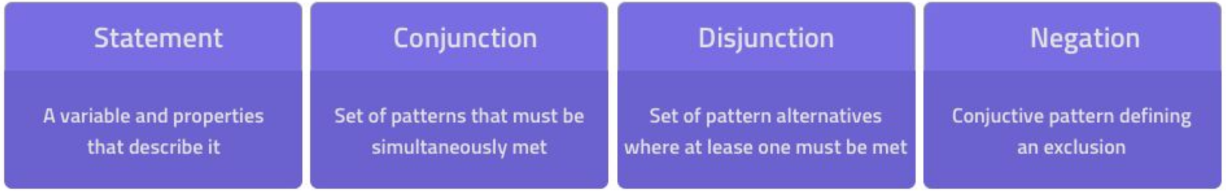
Statement structure



```
$p isa  person, has name “Bob”, has phone-number $phone ;
```

semi-colon
terminates the statement

Pattern structure



Statement	Conjunction	Disjunction	Negation
A variable and properties that describe it	Set of patterns that must be simultaneously met	Set of pattern alternatives where at lease one must be met	Conjuctive pattern defining an exclusion

TypeDB and ACID Properties

A

Atomicity

All operations of a transaction are successful, or none are persisted.
TypeDB transactions operate under a snapshot model.

C

Consistency

DB only moves to a correct state only when a transaction is committed.
Two primary types of data-level conflicts: modify-delete, key.

I

Isolation

Concurrent transactions operate as if they were run sequentially.
Full isolation is guaranteed by snapshot isolation.

D

Durability

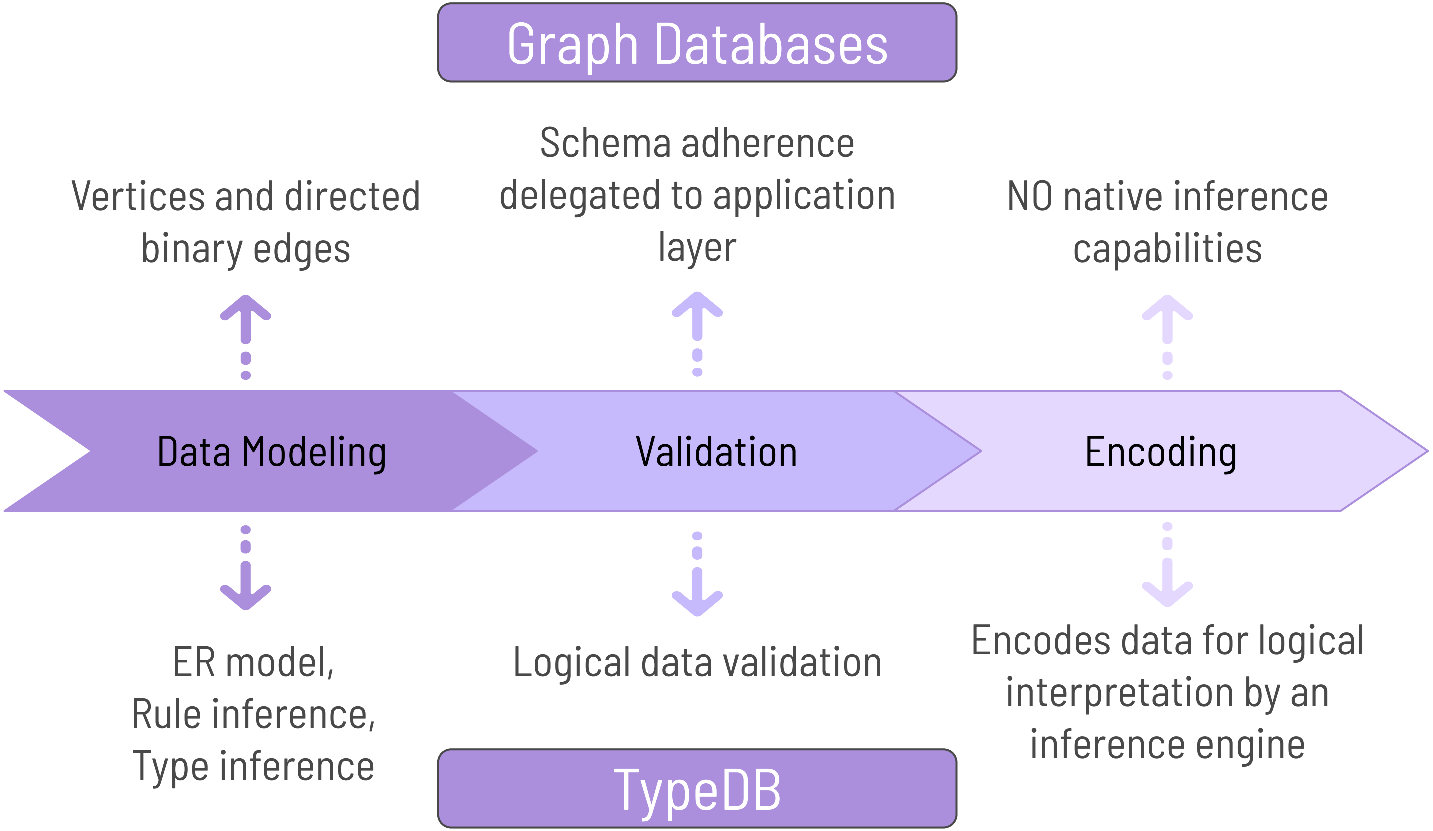
No data lost or corruption in the event of hw or power failure (data that finished committing will be available on reboot).

02

TypeDB vs other DB

Comparison with other Graph Databases and Relational Model

Graph Databases: Main Differences



Graph Databases

Vertices and directed binary edges

Schema adherence delegated to application layer

NO native inference capabilities

Data Modeling

Validation

Encoding

ER model,
Rule inference,
Type inference

Logical data validation

Encodes data for logical interpretation by an inference engine

TypeDB

Relational Model: Main Differences

Relational Databases	TypeDB
Data Modeling	
Normalisation is necessary (physical independence of data)	ER model, no normalisation needed (logical independence of data)
Schema	
● Primary key ● Foreign key (depending on model) ● Null values	● Primary key ● Foreign key (as a related relation) ● Attributes as first-class citizens

03

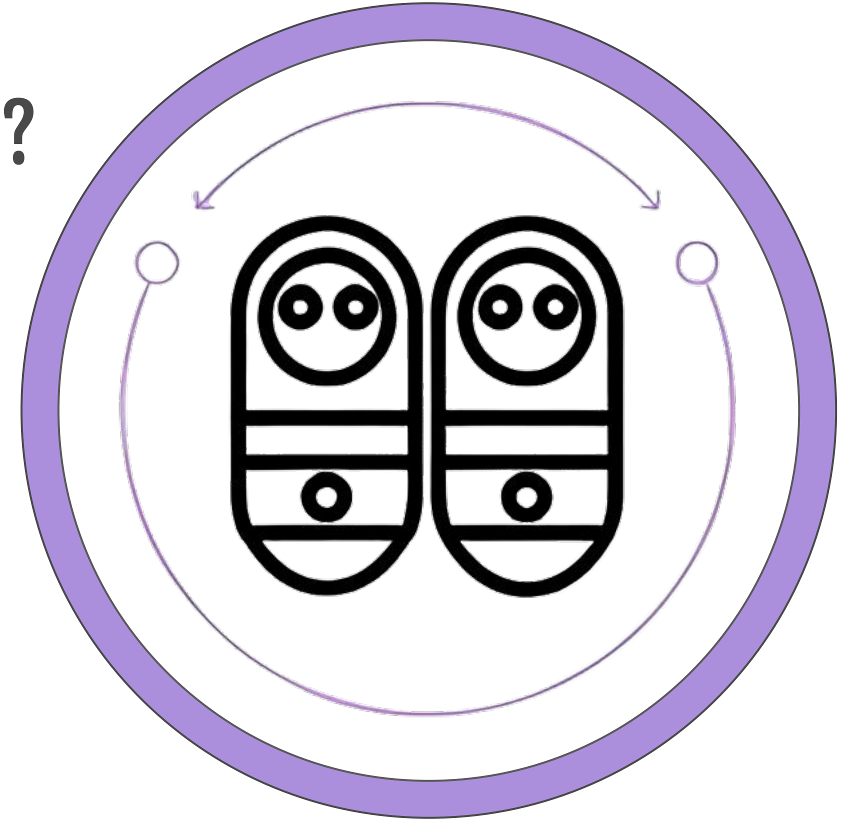
Digital Twins and Ditto CRUD APIs

Digital Twins and how they work, Ditto APIs and our APIs

What is a Digital Twin?

A Digital Twin is the abstract representation of anything.
It can be realised representing:

- main characteristics of the object
- attached data
- attached behaviour



Ditto and CRUD APIs:
a way to manipulate Digital Twins

Eclipse Ditto is an open source framework that helps to build digital twins of devices connected to the internet. Ditto acts as IoT middleware, providing an abstraction layer for IoT solutions interacting with physical devices via the digital twin pattern.

C Create

http://POST

R Read

http://GET

U Update

http://PUT

D Delete

http://DELETE

CRUD APIs mustn't be confused with REST APIs.
Our project is based on REST APIs, inspired by the HTTP APIs of Ditto's technology.

04

Other Technologies used

TypeDB Studio, Node.js,
Docker and Postman

Technologies Used

TypeDB Studio

GUI to interact with a TypeDB database.

Docker

Sw platform to deploy an application (without configuring anything).

Postman

Platform to design, build, test and iterate APIs.

Node.js

Runtime environment that executes JavaScript applications.

GitHub

Collaborative hosting service, used to store the source code of a software.

VS Code

Code editor for different programming languages.

Webstorm

IDE for Javascript code.

05

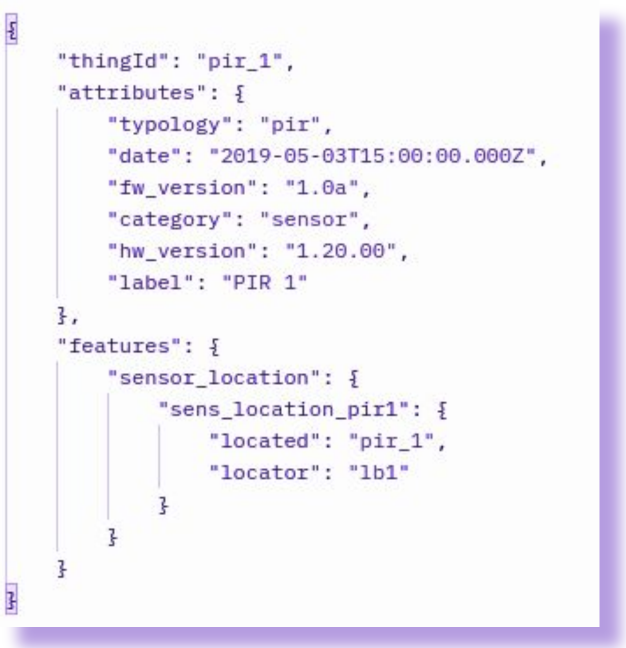
Technical Implementation

Implementation of the prototype
(NodeJS)

How is (our) Thing structured?

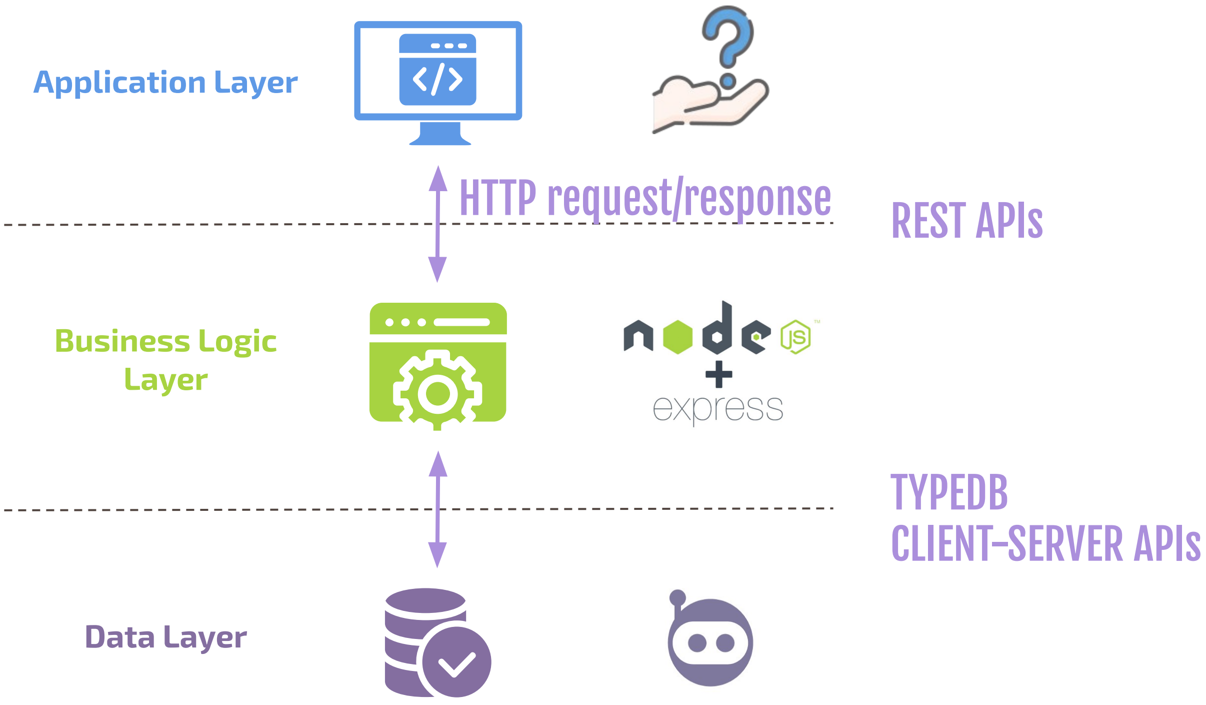
A Thing represents a Digital Twin with its characteristics, that can be described as follows:

- ThingId (thing Identifier)
- Attributes (thing Properties)
- Features (thing Relations)



```
{
    "thingId": "pir_1",
    "attributes": {
        "typology": "pir",
        "date": "2019-05-03T15:00:00.000Z",
        "fw_version": "1.0a",
        "category": "sensor",
        "hw_version": "1.20.00",
        "label": "PIR 1"
    },
    "features": {
        "sensor_location": {
            "sens_location_pir1": {
                "located": "pir_1",
                "locator": "lb1"
            }
        }
    }
}
```

Architecture



Application Layer

HTTP request/response

REST APIs

Business Logic Layer

TYPEDB CLIENT-SERVER APIs

Data Layer

Main code structure

Query Constructor

Creates query parts in order to execute an operation.

Query Runner

Calls Query Constructor to compose a real query and runs this in a specific transaction.

Query Manager

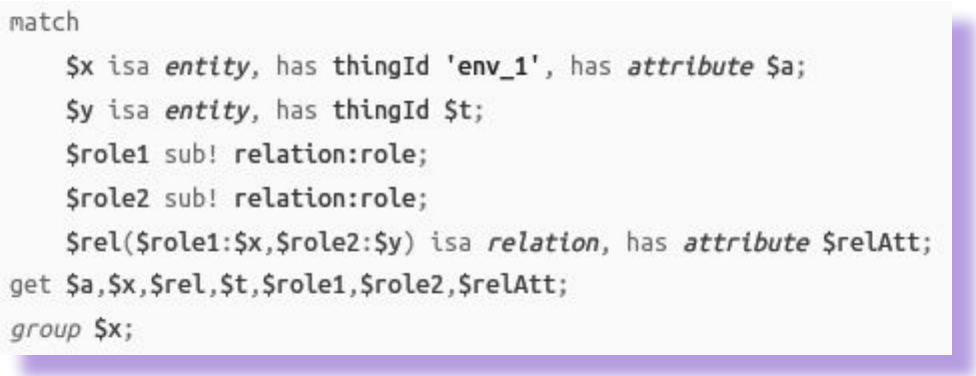
Instantiates a connection with DB creating a specific transaction (to run multiple queries through Query Runner).

Client Functions

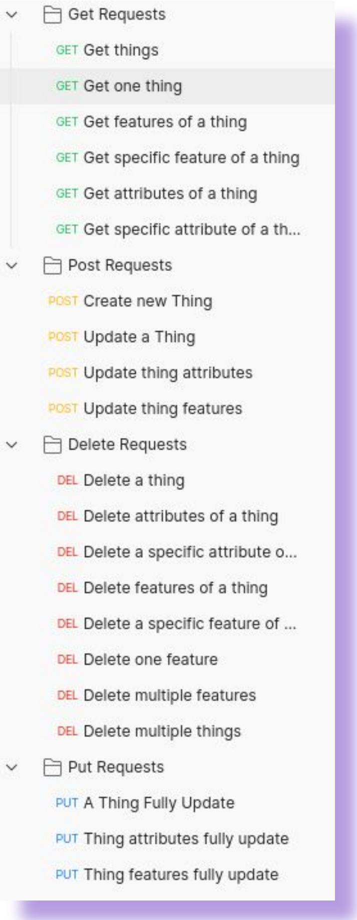
Manages sessions and transactions in DB.

Implemented APIs – Example

GET http://localhost:3030/things/env_1



```
match
    $x isa entity, has thingId 'env_1', has attribute $a;
    $y isa entity, has thingId $t;
    $role1 sub! relation:role;
    $role2 sub! relation:role;
    $rel($role1:$x,$role2:$y) isa relation, has attribute $relAtt;
get $a,$x,$rel,$t,$role1,$role2,$relAtt;
group $x;
```



- Get Requests
  - GET Get things
  - GET Get one thing
  - GET Get features of a thing
  - GET Get specific feature of a thing
  - GET Get attributes of a thing
  - GET Get specific attribute of a th...
- Post Requests
  - POST Create new Thing
  - POST Update a Thing
  - POST Update thing attributes
  - POST Update thing features
- Delete Requests
  - DEL Delete a thing
  - DEL Delete attributes of a thing
  - DEL Delete a specific attribute o...
  - DEL Delete features of a thing
  - DEL Delete a specific feature of ...
  - DEL Delete one feature
  - DEL Delete multiple features
  - DEL Delete multiple things
- Put Requests
  - PUT A Thing Fully Update
  - PUT Thing attributes fully update
  - PUT Thing features fully update

06

Conclusions and Further Work

What we have done and what could be implemented in future

Conclusions

The main activity of the work carried out was to process and transform digital twins data and properties through the implementation of some REST APIs.

The APIs implemented are able to connect to a TypeDB database and collect data from the database, performing the REST operations via HTTP requests.

Further Work

- Addition of some APIs to populate the database.

- Taken a .csv file and a program in NODE.JS, it calls the API to insert METADATA and FEATURES automatically.

Thanks!