**TiDiT**

**Application Programming Interface**

**v1.0**

**28.06.2023**

**Contributer**

|  |  |  |
| --- | --- | --- |
| **Name** | **Beneficiary** | **Email** |
| **O. Tufan DOĞAN** | INNOVA | [osdogan@innova.com.tr](mailto:osdogan@innova.com.tr) |
| **Büşra ÖNER** | BITES | Busra.oner@bites.com.tr |
| **Öndel DAL**  **Taşkın KIZIL**  **Selçuk AKSOY**  **Alper GIRGIN** | BITES  BEWELL  BEWELL  BEWELL | [Ondel.dal@bites.com.tr](mailto:Ondel.dal@bites.com.tr)  [taskin@bewelltech.com.tr](mailto:taskin@bewelltech.com.tr)  [selcuk.aksoy@bewelltech.com.tr](mailto:selcuk.aksoy@bewelltech.com.tr)  [alper.girgin@bewelltech.com.tr](mailto:alper.girgin@bewelltech.com.tr) |

**Table of** **Contents**

[**Table of** **Contents** 3](#_Toc170991984)

[**1** **Overview of API** 3](#_Toc170991985)

[**2** **API Structure** 4](#_Toc170991986)

[**2.1** **Frontend Application Architecture** 4](#_Toc170991987)

[**2.2** **Backend Application Architecture** 6](#_Toc170991988)

[**2.3** **Unity Application Architecture** 8](#_Toc170991989)

[**3** **API Capabilities** 10](#_Toc170991990)

[**4** **Documentation** 10](#_Toc170991991)

[**5** **Conclusion** 10](#_Toc170991992)

# **Overview of API**

The TiDiT architecture adopts an interoperability approach aimed at establishing a standardized method for discovering and accessing digital twins implemented by various vendors. The methodology draws inspiration from the Web of Things (WoT), which focuses on using Semantic Web technologies to define, expose, and consume web-based entities. Within the W3C WoT framework, the Thing Description (TD) serves as a foundational element, striving to become a standardized structure for semantically describing web entities, thereby enhancing their interoperability.

The TiDiT Gateway API will provide Digital Twins registry service. In addition  APIs will offer an interface operating at the syntactic level, utilizing a common format to define schema-based search criteria. It's the responsibility of the Gateway API to transform such requests into their corresponding SPARQL queries. When the Gateway API services component receives a search criteria in the form of a SPARQL query, it forwards it to the Discovery Services. These services are responsible for applying filtering to Thing Descriptions (TDs) stored in the TD Repository.

# **API Structure**

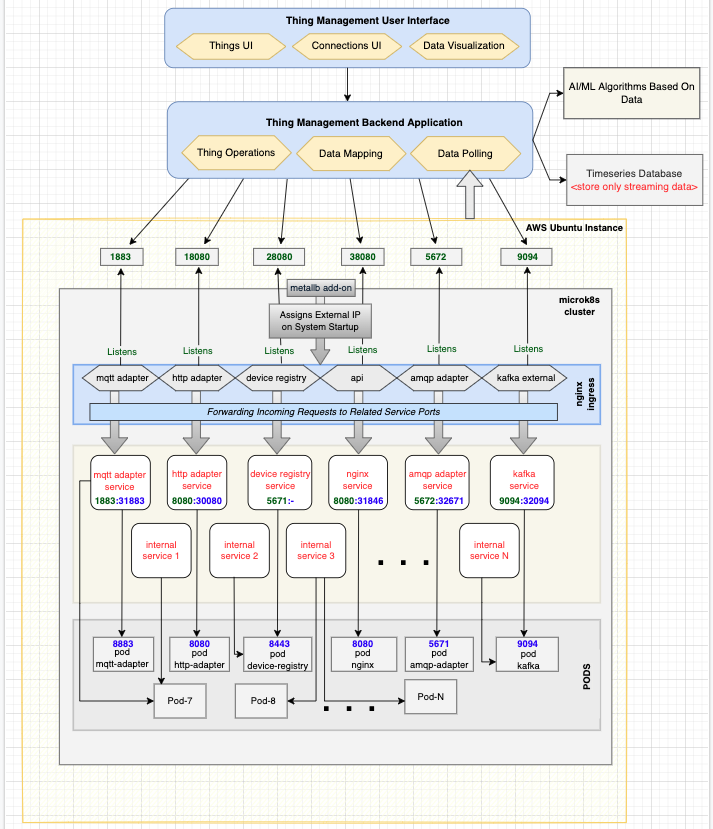
## **Frontend Application Architecture**

The Frontend API is the bridge between our functional Backend part and user. It will send the user input to the Backend Application.

In the development of the Frontend Application, JavaScript and React will be used. The Frontend Application will include the following modules:

* Thing Management
* Connection Management
* Data Visualization
* User Management

In the Thing Management section, users can add, edit, and delete objects referred to as "Things" within the system.

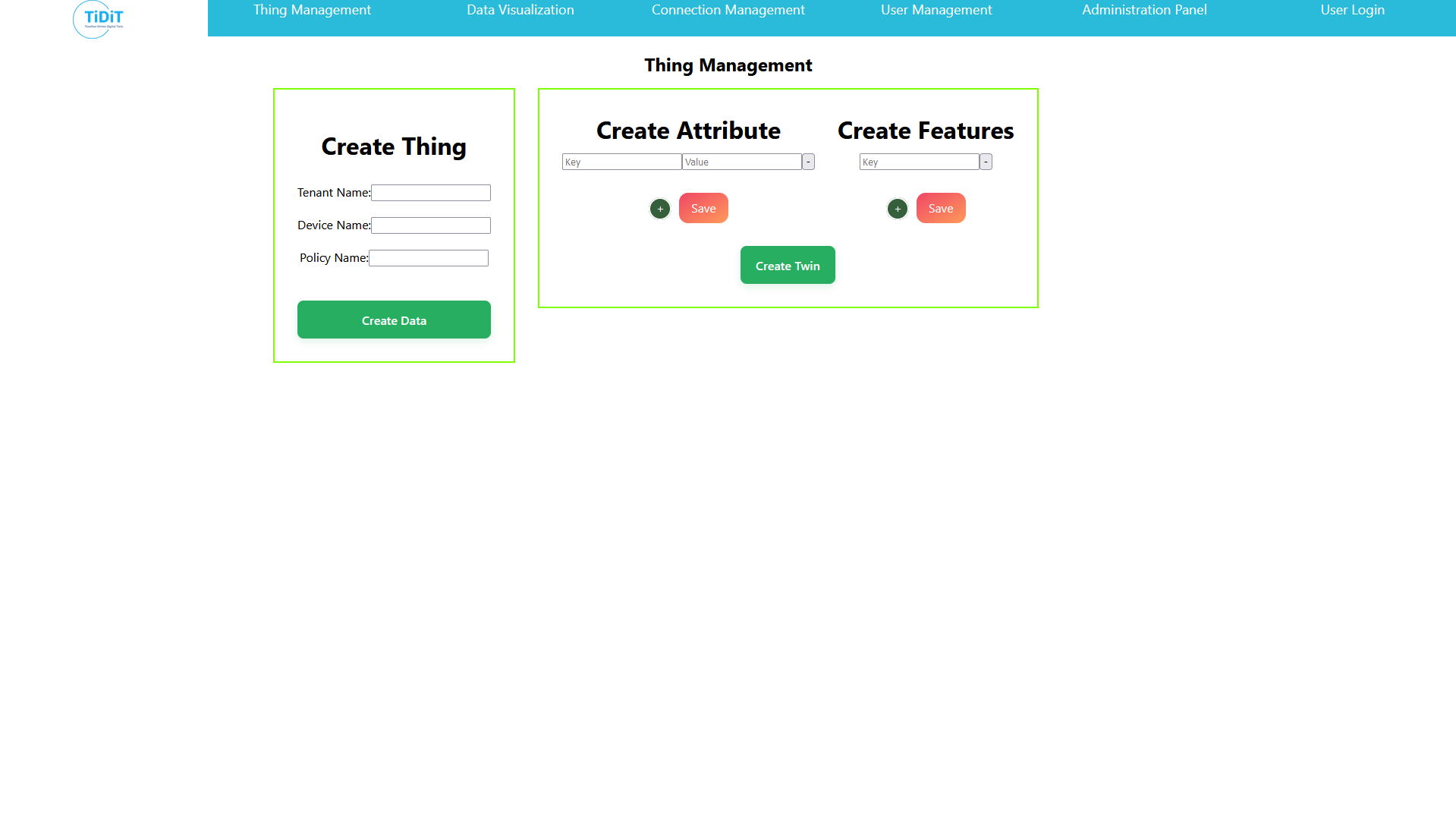


Stated Frontend Architecture

The Connection Management section allows users to define connections between Things and the system. These connections can be defined using various protocols such as MQTT, AMQP, HTTP, etc.

The Data Visualization section enables users to visualize data in a graphical format, such as charts or graphs.

In the User Management section, users within the TiDiT system can be defined, organized into user groups, and assigned permissions. These users, user groups, and permissions can be created, edited, and inactivated as needed.



Developed Frontend Application

## **Backend Application Architecture**

The Backend Application will be developed using Java and the Apache Streampipes technologies. Structurally, the Backend Application will facilitate data flow through different ports and various protocols. Python, TensorFlow, Keras, and similar technologies will be used for developing AI/ML algorithms. InfluxDB will be utilized as the Time Series Database, while PostgreSQL will serve as the Relational Database. The Backend Application will include the following modules:

* User Management
* Device Management (Thing Management)
* Relational Database Management
* Time Series Database Management
* Connection Management
* Service Management
  + Monitoring
    - Tracing & Metrics
    - Statistical Models
    - Fault Detection
    - Dashboard
  + Simulation
    - Executable Models
    - Visualization
    - Timeshift
  + Diagnosis
    - Condition Monitoring
    - Discover Event
    - Route Cause Analysis
  + Decision & Prediction
    - Semantic Models
    - Relational Models
    - Behavior Modeling
    - Performance Modeling
  + Control
    - Conflict Detection
    - Conflict Prevention
    - Conflict Resolution
    - Actuation

User Management

Backend functionality will be implemented to handle user management operations such as adding, editing, and deleting users, as well as creating user and permission groups.

Device Management

IoT devices and sensors within the system will be defined as "Things" and can be edited or deleted. The necessary functionalities will be implemented in the backend for managing Things.

Relational Database Management

This module will include operations for defining users, user groups, and roles in the relational database. PostgreSQL will be used as the Relational Database.

Time Series Database Management

A time-series database will be used to store real-time sensor data. InfluxDB will be utilized as the time-series database. The backend application will include functions to transfer sensor data from the devices to InfluxDB.

Connection Management

In the connection management module, the backend application will receive sensor data from different protocols using various adapter structures available in the cloud2edge package. Modules will be developed to receive data from different ports and communication protocols and transfer it to InfluxDB and the Frontend application.

Service Management

The service management module will be implemented in the Backend application, incorporating AI/ML algorithms.

Please refer to the attached for details on **backend API** integration and usage. (Please double-click to access the document.)

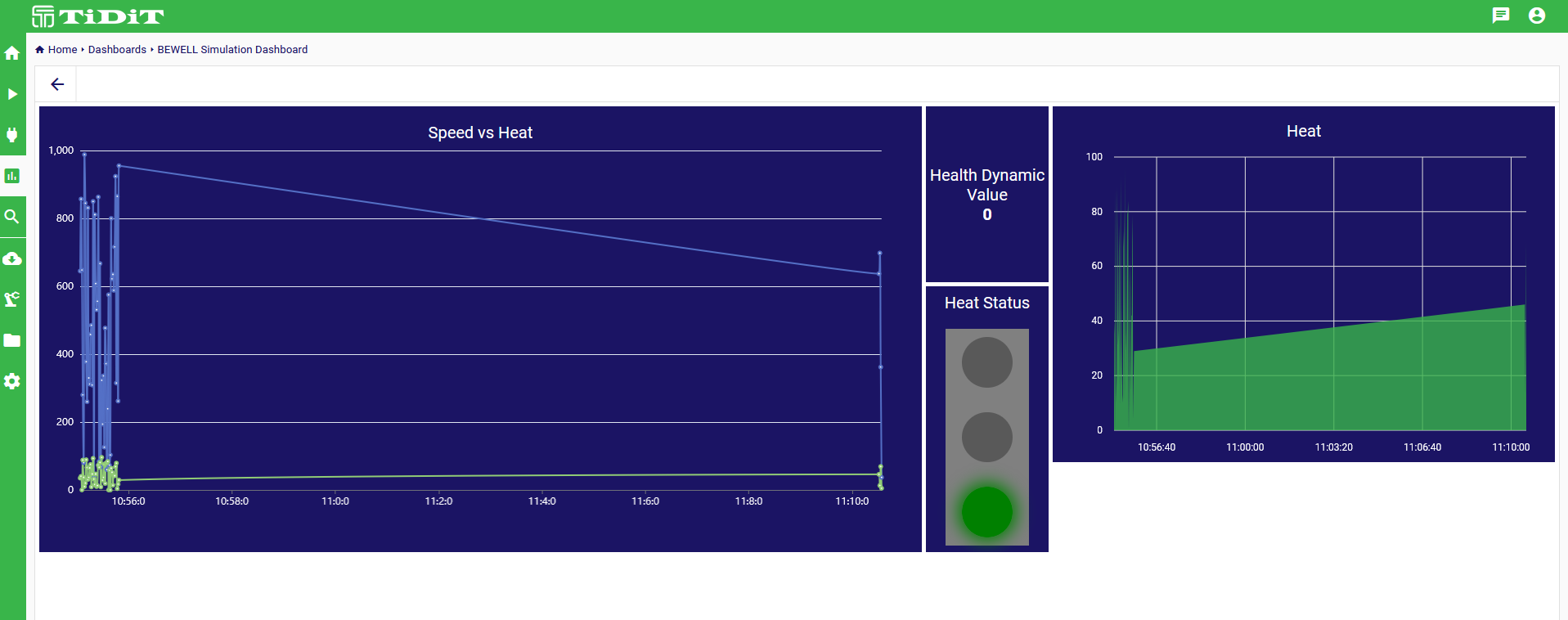


Please refer to the attached for details on **Interoperability** **API** integration and usage. (Please double-click to access the document.)

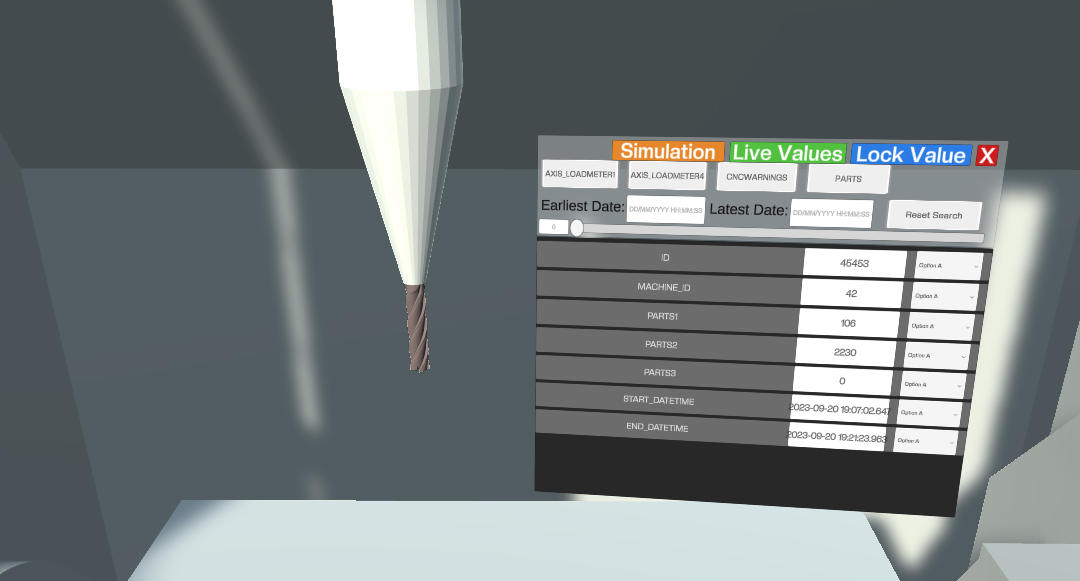


## **Unity Application Architecture**

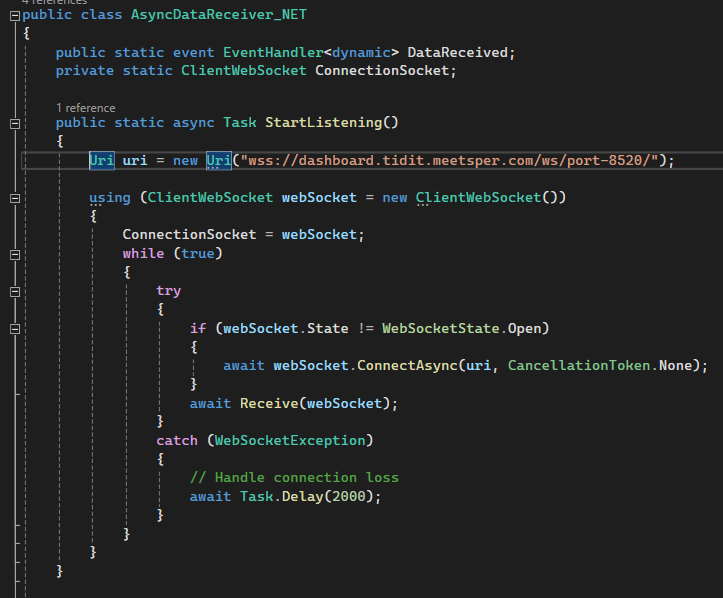
The Unity Application will be developed using C# technology. It will used for simulation and data visualization purposes. The API written for Unity will asynchronously collect data from the Apache Streampipes. The collected data will be shown to users on dashboards inside Unity.



Apachce Streampipes Data Pipeline for Unity



Dashboard Created Inside Unity for Monitoring



Unity Connection Handler

# **API Capabilities**

[Every partner that are responsible from API development can implement their part]

# **Documentation**

[Every partner that sent API documents can implement their part with screenshots]

For documenting Backend API we used Swagger documentation tool. The documentations of Backend API will be additionally added to the file.

# **Conclusion**

This API documentation has covered everything TiDiT aim to achieve. We've outlined the core functionalities and architecture. From onwards Frontend and Backend sides will merged into as one single Fullstack Application. The development and merging will go through Apache Streampipes.