SOMIOD

Service Oriented Middleware for Interoperability and Open Data

David Coimbra   
Engenharia Informática  
Instituto Politécnico de Leiria ESTGLeiria, Portugal  
2200507@my.ipleiria.pt

Luís Duarte  
Engenharia Informática

Instituto Politécnico de Leiria ESTGLeiria, Portugal  
2202148@my.ipleiria.pt

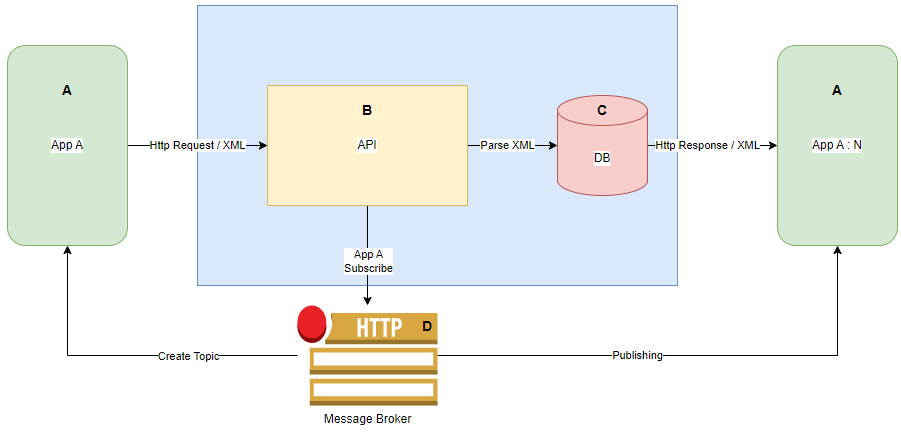
Francisco Lopes  
Engenharia Informática  
Instituto Politécnico de Leiria ESTGLeiria, Portugal  
2191202@my.ipleiria.pt

Nuno Costa   
Engenharia Informática  
Instituto Politécnico de Leiria ESTGLeiria, Portugal

Humberto Ferreira  
Engenharia Informática

Instituto Politécnico de Leiria ESTGLeiria, Portugal

*Abstract*

The SOMIOD project is a decentralized data collection and dissemination platform that enables the integration of data from various sources, such as sensors and databases, in order to facilitate the development of monitoring applications. The platform is based on a microservices architecture, which allows for the integration of multiple services that can be independently developed, deployed and scaled. The platform also uses a publish-subscribe messaging system based on the MQTT protocol, which enables the real-time dissemination of data to subscribed clients. The SOMIOD project has been evaluated in terms of performance, scalability and reliability, demonstrating its ability to handle large amounts of data and support a high number of concurrent connections. The SOMIOD project has the potential to revolutionize the way data is collected and disseminated, enabling the development of a wide range of monitoring applications in various sectors, including agriculture, health, transportation and environmental monitoring.

Keywords - IoT, Middleware, Systems Integration, Data, Service Oriented

# Introduction

The SOMIOD project is a software platform that aims to facilitate the development of distributed Internet of Things (IoT) applications. It provides a set of tools and services that allow developers to easily connect IoT devices, exchange data, and implement real-time data processing and analysis.

Fig1 - System Architecture 1

SOMIOD was designed to be highly modular and scalable, making it suitable for a wide range of IoT applications. It uses a publish-subscribe messaging model and supports multiple communication protocols, including MQTT and HTTP. Additionally, SOMIOD provides a RESTful API and a web-based dashboard for monitoring and managing IoT deployments.

In this paper, we present the architecture and features of SOMIOD, and demonstrate how it can be used to build a simple IoT application. We also discuss some of the challenges and opportunities that arise when using SOMIOD for IoT development and outline some directions for future work.

# System Architecture

In this project, we focused on developing a system architecture for a distributed application that leverages the SOMIOD platform. Our system architecture aims to provide a scalable, flexible, and reliable framework for building and deploying applications that can interact with a variety of devices and data sources. To achieve these goals, we utilized a combination of technologies and approaches, including RESTful APIs, MQTT messaging, and microservices architecture. Our system architecture also includes features such as notification capabilities and data storage and management. Overall, our system architecture serves as a foundation for building and deploying applications that can effectively leverage the SOMIOD platform to solve a range of real-world problems.

## Component A

The component A is the embodiment application that as access to the full CRUD of our API. In that same application we have an interface that was built in Windows Form App. The dashboard as access to other menus like Data, Subscription, Module and Application forms. Throw them we can make HTTP operations so we can request the data from our API.

## Component B

The component B is our API. It handles HTTP methods such as GET, POST, PUT and DELETE. Our API receives data in XML format that parses data with the column type variables in our database and applies it to each request received.

The API then sends XML as well with a code status.

Extra: We have a special controller that receives the HTTP request and our “res\_type” variable that help us choose the controller that we want to use for this HTTP Request.

## Component C

The component C is our Database. Since we are using a type of MVC structure, we applied a package so we could use a ORM so that our queries to our database became safer, simpler, more readable and cleaner.

## Component D

The component D is our Message Broker. Each time there is a POST in our subscription controller, our broker/Msqtt creates a “topic” based on his endpoint and type of operation that our data controller is going to receive (Subscription).

Each time there is a POST/DELETE in our data controller, the message broker publishes to the “topic” that was created previously depending on the type of the subscription.

| Models Table | | | |
| --- | --- | --- | --- |
| # | Variables | Type | Access |
| Application | Id | Int | Get/Set |
| Application | Name | string | Get/Set |
| Application | Creation\_dt | DateTime | Get/Set |
| Module | Id | Int | Get/Set |
| Module | Name | String | Get/Set |
| Module | Creation\_dt | DateTime | Get/Set |
| Module | Parent | Int | Get/Set |
| Subscription | Id | Int | Get/Set |
| Subscription | Name | String | Get/Set |
| Subscription | Creation\_dt | DateTime | Get/Set |
| Subscription | Parent | Int | Get/Set |
| Subscription | Event\_ | String | Get/Set |
| Subscription | Endpoint | String | Get/Set |
| Data | Id | Int | Get/Set |
| Data | Content | String | Get/Set |
| Data | Creation\_dt | DateTime | Get/Set |
| Data | Parent | Int | Get/Set |

1. Models Table
2. Models Tables

| Controllers Table | | | |
| --- | --- | --- | --- |
| # | Function | Method | Type |
| Application | Get | GET | HttpResponseMessage |
| Application | Post | POST | HttpResponseMessage |
| Application | Put | PUT | HttpResponseMessage |
| Application | Delete | DELETE | HttpResponseMessage |
| Application | GetAll | GET | HttpResponseMessage |
| Application | ParseXml | - | Application |
| Application | UpdateModel | - | Application |
| Application | CreateXml | - | XElement |
| Application | CreateXmlExtended | - | XElement |
| Module | Get | GET | HttpResponseMessage |
| Module | Post | POST | HttpResponseMessage |
| Module | Put | PUT | HttpResponseMessage |
| Module | Delete | DELETE | HttpResponseMessage |
| Module | GetAll | GET | HttpResponseMessage |
| Module | ParseXml | - | Module |
| Module | UpdateModel | - | Module |
| Module | CreateXml | - | XElement |
| Module | CreateXmlExtended | - | XElement |
| Subscription | Post | POST | HttpResponseMessage |
| Subscription | Delete | DELETE | HttpResponseMessage |
| Subscription | ParseXml | - | Subscription |
| Subscription | CreateXml | - | XElement |
| Data | Post | POST | HttpResponseMessage |
| Data | Delete | DELETE | HttpResponseMessage |
| Data | ParseXml | - | Data |
| Data | CreateXml | - | XElement |
| CustomerHttp | SelectController | - | HttpControllerDescriptor |

1. Controller Table
2. Controller Table

| Forms Table |
| --- |
| Names |
| Form1 |
| Application |
| Modules |
| Subscription |
| Data |

1. Form Table
2. Forms Table

# Integration/App Development

The Integration/App Development section of this project focuses on the process of integrating the various components developed in the System Architecture section and creating applications that utilize these components. This involves creating APIs that allow the components to communicate with each other and with external applications, as well as developing user-facing apps that leverage the functionality of the system. The goal of this process is to create a cohesive and user-friendly system that can be easily integrated into various business processes and operations. In order to achieve this, careful planning and attention to detail is required in order to ensure that all components work seamlessly together and provide the desired functionality. This includes testing and debugging the system to ensure that it operates correctly and meets the needs of the users. Ultimately, the successful integration and app development of this project will enable the creation of innovative and efficient solutions for various businesses and organizations.

# Conclusions and Future Work

# Appendix

*Appendix A*

| Controllers Table | | |
| --- | --- | --- |
| # | Routes | Method |
| Application | api/somiod/applications/{applicationName} | GET |
| Application | api/somiod/applications/ | POST |
| Application | api/somiod/applications/{applicatioName} | PUT |
| Application | api/somiod/applications/{applicatioName} | DELETE |
| Application | api/somiod/applications/ | GET (All) |
| Module | api/somiod/applications/{applicationName}/modules/{moduleName} | GET |
| Module | api/somiod/applications/{applicationName}/modules | POST |
| Module | api/somiod/applications/{applicationName}/modules/{moduleName} | PUT |
| Module | api/somiod/applications/{applicationName}/modules/{moduleName} | DELETE |
| Module | api/somiod/applications/{applicationName}/modules/ | GET (All) |
| Subscription | api/somiod/applications/{applicationName}/modules/{moduleName}/subscriptions/ | POST |
| Subscription | api/somiod/applications/{applicationName}/modules/{moduleName}/subscriptions/{subscriptionName} | DELETE |
| Data | api/somiod/applications/{applicationName}/modules/{moduleName}/data/ | POST |
| Data | api/somiod/applications/{applicationName}/modules/{moduleName}/data/{dataId} | DELETE |

*Appendix B*

Each individual of the group did their tasks since we have a happy group, we had a happy project but unfortunately, we didn’t had time for more.

Everyone worked in every part of the project.