FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

AsyncAPI-First Design for Event-Driven Architectures: Improve Developer Experience

José Silva

Mestrado em Engenharia Informática e Computação

Supervisor: João Pedro Dias

July 25, 2024

AsyncAPI-First Design for Event-Driven Architectures: Improve Developer Experience

José Silva

Mestrado em Engenharia Informática e Computação

July 25, 2024

Resumo

Desde analistas desportivos em tempo real até casas inteligentes, sistemas modernos têm contribuído para o aumento da necessidade de arquiteturas escaláveis e responsivas. Atualmente, arquiteturas baseadas em eventos (EDA) são a solução mais usada para combater esta necessidade. Ao contrário de modelos tradicionais de pedido e resposta como REST APIs, estas arquiteturas baseiam-se na propagação, deteção e resposta de eventos, desde dados de um sensor até interações de um dado utilizador. De todos os paradigmas desta área, o publish/subscribe é o paradigma mais utilizado e consiste na troca assíncrona de mensagens com base numa categoria ou tópico. Esta troca de mensagens é feita usando um event bus ou um broker. Apesar deste tipo de arquiteturas satisfazerem a maioria das necessidades de sistemas modernos, existem algumas dificuldades associadas com a sua utilização. As mais severas estão relacionadas com a complexidade de gerir as mensagens e garantir a sua consistência. Na maioria dos sistemas, uma mensagem é composta pelo event header numa especificação como CloudEvents e pela event data que contêm a informação do evento e é normalmente serializada num formato como Avro ou JSON. O facto de ser uma área recente faz também com que o número de ferramentas para interagir com estas arquiteturas seja reduzido, dificultando o seu desenvolvimento e design. Em parceria com a Kuehne+Nagel, esta tese, explora diferentes métodos de criar documentação para aplicações deste tipo de arquiteturas assíncronas e distribuídas de maneira a melhorar a performance e a experiência dos seus desenvolvedores. Em primeiro lugar, foi feita uma breve comparação das diferentes ferramentas existentes com este prepósito. De seguida, o desenvolvimento de uma ferramenta com este exato prepósito é descrito. As suas funcionalidades fundamentais incluem a geração de ficheiros de especificação com base na especificação de AsyncAPI e utilizando ficheiros de schema como referência, a partir de uma interface gráfica (GUI) intuitiva e fácil de usar. Para avaliar estas ferramentas, para além da comparação da performance da ferramenta desenvolvida com ferramentas já existentes, foram feitas experiências com sujeitos com diferentes níveis de conhecimentos na área. Com o aumento da adesão a este tipo de arquiteturas, este tipo de ferramentas não contribui apenas para o aumento de ferramentas disponíveis nesta área de conhecimentos mas também oferece a arquitetos e desenvolvedores uma solução mais direta e eficiente para criar sistemas responsivos e robustos.

Keywords: AsyncAPI, OpenAPI, Event-Driven Architecture, Avro, CloudEvents, Kafka.

Abstract

Modern systems, from real-time Sports analyses to Smart houses, have contributed to the growth of the demand for responsive and scalable architectures. The adoption of Event-Driven architectures (EDA) is currently the most used solution to fulfill these demands. Unlike traditional request-response models like REST APIs, these architectures revolve around various events' propagation, detection, and response, from sensor inputs to user interactions. Among all the paradigms in this scenario, the publish/subscribe paradigm is one of the most used and consists of an asynchronous exchange of messages based on a category or topic. This exchange is done using an event bus or a broker. Although these kinds of architectures fulfill most of the needs of modern systems, there are a lot of challenges associated with them. The more severe are related to the complexity of handling the messages and ensuring their overall consistency. In most systems, a message is composed by the event header in a specification like CloudEvents and the event data and is serialized using Avro or JSON format. The field's novelty also makes the number of tools to interact with these architectures low, making the overall design and development harder. In pair with Kuehne+Nagel, this thesis explores methods for creating documentation for the applications of these asynchronous and distributed architectures in order to increase performance and contribute to a better experience for their developers. Firstly, a comparison of the available tools that can do it was made. Then, it is described the development of a developer-friendly helper tool with this exact purpose. Its key functionalities include bootstrapping a specification file compliant with the AsyncAPI specification and using schema files as a reference from an easy and intuitive Graphical User Interface (GUI). Different scenarios were used to compare the performance of this tool to that of other existing tools. Also, experiments were conducted with subjects with different backgrounds and expertise levels. As organizations increasingly embrace event-driven paradigms, this tool not only contributes to the growing body of knowledge in this field but also offers a more straightforward solution for architects and developers to create responsive and robust systems with more efficiency while having a better experience.

Keywords: AsyncAPI, OpenAPI, Event-Driven Architecture, Avro, CloudEvents, Kafka.

Acknoledgements

Firstly, I would like to thank my supervisor, Professor João Pedro Dias, for his invaluable guidance and availability throughout the course of this thesis. Your insights and expertise were crucial to the making of this thesis.

Secondly, thank you to António Sernadas and Nuno Alves from Kuehne+Nagel for your insights and support and for always making me feel welcome in the company.

I am also immensely grateful to my friends, who have provided unwavering support and motivation.

I would like to extend my heartfelt thanks to my family, especially my mother and sister. Your constant understanding, encouragement, and love have been my greatest source of strength.

Finally, thank you to my father, who I know would be really proud of me.

Thank you all for your significant contributions to the completion of this thesis.

José Silva

"The beautiful thing about learning is nobody can take it away from you."

B.B. King

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this chapter, the topics of this thesis are introduced. It is also presented the problem and the motivation to create a solution to it. Lastly, the main goals of this document are described, as well as its structure.

Section 1.1, p. 1 presents the context of this thesis. Section 1.2, p. 2 contains the motivation for the work. Section 1.3, p. 3 presents the goals to be achieved. Finally, Section 1.4, p. 3 describes the structure of this document.

1.1 Context

Event-Driven Architectures represent a paradigm shift in how software is developed, designed, and operated, offering robustness, scalability, and flexibility. That is why these types of architectures have been attracting the attention of architects, researchers, and organizations to fulfill the demands of modern systems. Unlike traditional request-response models like REST APIs, Event-Driven Architectures revolve around the propagation, detection, and response of events between different types of applications. An example of the usage could be the Internet of Things (IoT) field [14]. In this field, every object becomes a complex cyber-physical system (CPS) [20], where its software and physical characteristics are linked. These objects usually use sensors and interfaces (APIs) to exchange data. In order to improve the behavior and reliability of these systems, these architectures are used for asynchronous communication. One of the most used paradigms today is the publish/subscribe [2] following a protocol like the Message Queuing Telemetry Transport (MQTT) protocol, for example. In this paradigm, the messages are published and consumed by a CPS according to a certain category or topic and are not directly exchanged by the agents. Although these kinds of architectures fulfill most of the needs of modern systems, there are a lot of challenges associated with them. The more severe ones are related to the complexity of handling

the messages and ensuring their overall consistency because there must be an agreement on the expected message categories and the internal message structure and format. Usually, a message is composed by the event header in a specification like CloudEvents and the event data and is serialized using Avro [3] or JSON [26] format. The current industry standard for defining asynchronous communication is the AsyncAPI specification [6]. It defines a set of fields that can be used in AsyncAPI Documents to describe an application's API. However, although there are already industry standards to define an Asynchronous API and to define the message's structure and events specification, the field's novelty makes the number of tools to interact with these architectures low, making the overall design and development harder, especially for people who don't have any background in this field.

Regarding the problem to solve in this thesis, its definition was made in pair with Kuehne+Nagel[1].

The search for possible problems to solve started in July 2023. After some research, an obvious problem in how the operations of an app were defined in an AsyncAPI Document was found. The problem was defining whether the operation was related to a publish from the app or a subscription to the channel. In the Document, if an operation had, for instance, a subscribe field, it didn't mean that the application subscribed to a channel but that the application allowed other applications to subscribe to that channel to receive messages. This could easily be misunderstood by non-experienced developers, so the first idea for the problem of this thesis was to find a way to deal with this easily misunderstood field. However, in October of 2023, AsyncAPI announced a new version of the AsyncAPI Specification that has a different way of defining the operations. In this new version, the operations are defined with an action field that can have the values "send" and "receive". This solved the problem that was expected to be solved in this thesis. Nevertheless, the research on how to create and understand an AsyncAPI Document continued, and it was found that there only existed a few number of tools to create this type of documents, and most of them were not able to create them with the most recent version of the AsyncAPI Specification. So, the available solutions weren't optimal for creating an AsyncAPI Document, so this became the problem that this thesis tried to solve.

1.2 Motivation

Since the development and management of Event-Driven Architectures (EDAs) is a recent field with a low number of existing tools to design and interact with them, resulting in a harder and more time-consuming experience for architects and developers, a solution to this problem would bring advantages, such as:

Improve the design experience of Event-Driven Architectures Having an easy-to-use tool to design Event-Driven Architectures would reduce the time spent in these processes as well as the learning curve to do it.

[1] https://home.kuehne-nagel.com/en/

Reduce the time spent on the creation of an AsyncAPI Document A more easy-to-use tool that provides some pre-defined fields and structure could reduce the time spent in the creation of an AsyncAPI Document.

Reduce the errors made by developers An easy-to-use tool with features such as validation of schemas and application's AsyncAPI Documents could reduce the errors developers make while creating an AsyncAPI Document.

To summarize, the main motivation is to improve the developer experience in the design of Event-Driven Architectures.

1.3 Goals

The first goal of this thesis is to understand the current issues developers face while creating AsyncAPI Documents, researching the current industry standards and available tools in the field, and comparing their main features.

The second one is to conceptualize one tool that would combine the existing tools' best features to try to improve the overall experience of developers and architects in creating these Documents, reducing their time spent on it and their errors in the process.

The third goal is to validate the created tool's feasibility by comparing it with other existing tools in different scenarios and through a user experiment with subjects with different levels of expertise.

To accomplish these goals, the following hypothesis was claimed:

H: A tool capable of creating an AsyncAPI Document with an intuitive Graphical User Interface reduces the time spent on creating an AsyncAPI Document and the error proneness.

Based on the hypothesis, we will try to address the following research questions (RQs):

RQ1 *Does the solution help developers by reducing the time spent creating an AsyncAPI Document?*

To answer this question, the time spent creating an AsyncAPI document using the developed prototype and the tools provided by AsyncAPI will be compared in a feasibility user study.

RQ2 *Does the solution help developers by reducing the number of errors made while creating an AsyncAPI Document?*

To answer this question, a feasibility user study will compare the creation of a valid AsyncAPI document using the developed prototype and the tools provided by AsyncAPI.

1.4 Document Structure

This document is structured into six chapters that can be described as follows:

- Chapter 1 (p.1) **Introduction**, introduces the context, motivation, and goals of this thesis.
- Chapter 2 (p.5) **State of the Art**, describes the solutions that exist relevant to this dissertation, doing a brief introduction to AsyncAPI and comparing different tools to generate API Documentation.
- Chapter 3 (p.16) **Problem Statment**, presents the problem that this thesis intends to address, what is the solution, and how it is going to be solved and evaluated.
- Chapter 4 (p.19) **Solution Prototype**, Explore and describes the development of the solution prototype and its features.
- Chapter 5 (p.30) **Validation**, Describes and documents the process used to validate the designed prototype.
- Chapter 6 (p.49) **Conclusions**, contains a recap of the work done and some ideas for future work.

Chapter 2

State of the Art

In this chapter, will be discussed the current state of the specification of asynchronous APIs and Event-Driven Architectures, as well as some tools to design them. Since this is a relatively new area and its overall body of knowledge is still not that big, will also be discussed some tools for designing HTTP APIs based in openAPI [25] to try to find ideas that could be used in the designing of asynchronous APIs.

2.1 AsyncAPI

AsyncAPI [6] is an open-source initiative that seeks to improve the build and maintenance of event-driven architectures. The AyncAPI specification is currently the industry standard for defining asynchronous APIs.

2.1.1 Main Concepts

To understand how the specification and the tools work, it is first necessary to understand the main concepts of AsyncAPI. In the following subsections, all the main concepts are presented.

Producer A producer is an application that detects state changes, known as events, and broadcasts these changes as messages. Producers are called publishers in publish/subscribe models of event-driven architectures, and their goal is to distribute messages to the broker so subscribers can receive them.

Consumer A consumer is an application that listens for a specific event from a broker and reacts accordingly when this event occurs. In publish/subscriber models, the consumer is called subscriber, and when it receives an event from a broker, it is completely unaware of the producer or other consumers.

Server A server represents a messaging broker system whose role is to maintain the connections and enable communication between producers and consumers.

Channel Intending to organize and transmit messages, a channel is a mechanism created by the server. In communication between a producer and a consumer, the producer sends a message through the channel, which queues it to the specific consumer. Depending on the protocol used, the channel is defined as a topic, queue, subject, routing key, or path.

Application In Event-Driven Architecture (EDA), an application is a producer, a consumer, or both. In these types of architectures, an application could be a micro-service, an IoT device (sensor), a process, etc. The applications can be written in different programming languages supporting the same protocol.

Protocol A protocol is a defined set of rules for exchanging information between applications and/or servers. Some examples of protocols used in Event-Driven Architecture are:

- WebSockets
- Kafka
- MQTT
- AMQP

Message A message is an asset used to exchange information between a producer and one or more consumers through a channel. It can be defined as an event or a command. Messages must be processed and serialized into a certain format, e.g., binary, XML, JSON, Avro, etc. It can also include information that describes the message itself. This type of information is often called properties or headers.

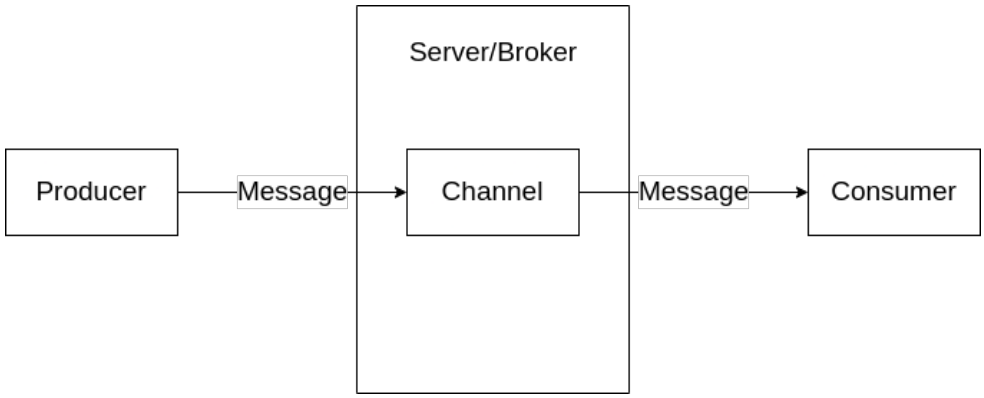


Figure 2.1: Simple EDA diagram example

2.1.2 AsyncAPI Document

An AsyncAPI document is composed of a set of fields defined by AsyncAPI Specification to describe an application's API. It is typically a single document that encapsulates the API description but can reference other files for additional details. In an event-driven system, it acts as a mutual agreement between the senders and the receivers since it specifies the required payload content that a producer sends and that a consumer receives.

2.1.2.1 Structure

An AsyncAPI document has a specific and well-defined structure format that follows the AsyncAPI specification. It is composed of five main components, although it is not mandatory to use all of them, that provide information about the API's characteristics and behavior. The main components are:

- **Info field** - defines the API's metadata, e.g., title, version, description, contact, license, etc.
- **Servers field** - includes crucial information about the connection, e.g., host, port, protocol, etc. It facilitates connection in different environments like production, staging, and development.
- **Channels field** - provides a map of different channels used by the application to communicate. Each channel can be defined here by its address, expected message format of communication, the server where it is available, etc.
- **Operations field** describes the different operations the application performs. Its main fields, such as action, channel, and message, offer a clear description of the purpose of each operation and whether the application receives or sends messages.
- **Components field** - is used for defining reusable structures or definitions. All the items defined in this field only become part of the API if referenced in other fields.

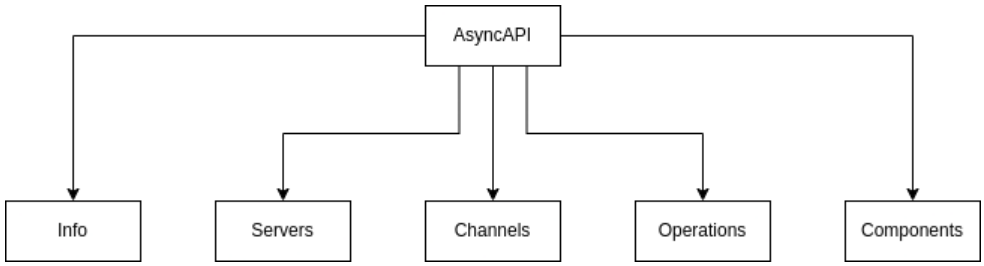


Figure 2.2: AsyncAPI Document Structure

2.1.2.2 Example

Simulating a simple smart LED's API that sends an event with the information that it was turned on or off, this section will present the creation of a simple AsyncAPI document. First of all, the user has to fill out the info field with the title of the API, its version, and its license.

```
---
asyncapi: 3.0.0
info:
  title: Smart LED API
  version: 1.0.0
  description: >
    This is an example of an API of a smart LED
    that sends an event when is turned on or off.
  license:
    name: Apache 2.0
    url: https://www.apache.org/licenses/LICENSE-2.0
---
```

Listing 2.1: AsyncAPI Document Info Component

After that, the servers field has to be filled out. This example will use a Kafka cluster as the host and Kafka as the protocol. Since this is only a demonstrative example, only one server will be used. However, we could use more.

```
---
servers:
  my-server:
    host: test.mykafkacluster.org:18092
    protocol: kafka
    description: Test broker
---
```

Listing 2.2: AsyncAPI Document Servers Component

The next step is to describe the channels needed in the channel field. It will only need two channels, one for turning the LED on and another for turning it off. In each channel, it must be specified the address and the format of the message that it will be sent, and in this case, it will be also needed to add a parameter with the ID of our LED. Since the message format will be used in other fields, it will be created in the Components field and used as a reference.

```
---
channels:
  turnOn:
    address: smart.led.1.0.action.{ledId}.turn.on
```

```
5       messages:
6         turnOn:
7           $ref: '#/components/messages/turnOnOff'
8       parameters:
9         ledId:
10          $ref: '#/components/parameters/ledId'
11    turnOff:
12      address: smart.led.1.0.action.{ledId}.turn.off
13      messages:
14        turnOff:
15          $ref: '#/components/messages/turnOnOff'
16      parameters:
17        ledId:
18          $ref: '#/components/parameters/ledId'
19  ---
20  \
```

Listing 2.3: AsyncAPI Document Channels Component

The two operations that the LED will send (turn on and off) will be added in the operations field. Both operations have the action "send", and they will use the channel specified in the Channels field and the message format written in the Components field.

```
1   ---
2   operations:
3     turnOn:
4       action: send
5       channel:
6         $ref: '#/channels/turnOn'
7       traits:
8         - $ref: '#/components/operationTraits/kafka'
9       messages:
10        - $ref: '#/channels/turnOn/messages/turnOn'
11    turnOff:
12      action: send
13      channel:
14        $ref: '#/channels/turnOff'
15      traits:
16        - $ref: '#/components/operationTraits/kafka'
17      messages:
18        - $ref: '#/channels/turnOff/messages/turnOff'
```

```
19 ---
```

Listing 2.4: AsyncAPI Document Operations Component

2.2 AsyncAPI Studio, CLI, and AsyncAPI Preview for VS Code

The AsyncAPI Studio [11] is a tool made by AsyncAPI that enables its users to develop, validate, and preview an AsyncAPI document. It can also convert any document to the latest AsyncAPI version. It is composed by two main components. The first is a text editor where the user can write a document. If the document has any errors, they will be presented to him so he can fix them. The second component is an HTML preview window where all the components of the document are displayed in a more tidy and easy-to-read way. In general, it is a simple tool to use. However, suppose the user is not comfortable with the AsyncAPI specification. In that case, it will be hard for him to use it since the main feature is the validation and preview of an AsyncAPI document.

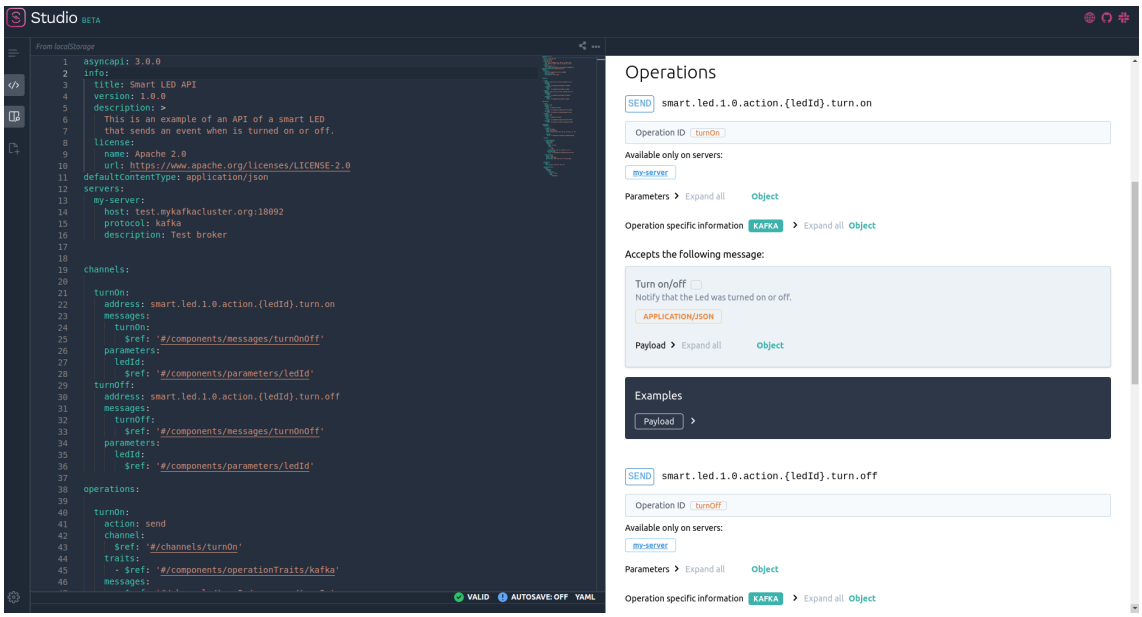


Figure 2.3: AsyncAPI studio with the example document 2.1.2.2

AsyncAPI also provides a command line interface (CLI) [8] that enables its users to create, validate, and convert the version of Documents. This tool also offers features like a command to find the differences between two documents or a command to optimize the structure of a document.

Finally, for Visual Studio Code [23] Users, there's there is the AsyncAPI preview extension that can be used to documents inside the text editor. In contrast with the AsyncAPI Studio, this extension provides some snippets to create a document faster and can manage references to other files.

2.3 Solace PubSub+ Event Portal

PubSub+ Event Portal [29] is a commercial solution provided by Solace that enables its users to design, discover, visualize, share, and manage various aspects of event-driven architectures (EDA). In the context of this thesis, the two most important tools to acknowledge are the Designer tool and the RunTime Event Manager tool. All the tools are provided in a Web User Interface (UI), making the overall user experience better.

2.3.1 PubSub+ Event Portal Designer

As the name says, the Designer tool is a tool to view and design Event-Driven Architectures. While the AsyncAPI Studio has a more application-driven approach, this tool focuses more on the overall environment of the architecture's applications. However, instead of using a text-based approach to define applications and events, it has an easy-to-use interface to do it. To create a simple smart LED API as the example(2.1.2.2), the user first has to create an application domain, then create the application (smart LED) and the events (turnOn and turnOff) and associate a payload schema to these events. It only supports Solace and Kafka Brokers, which can be a downside for many users.

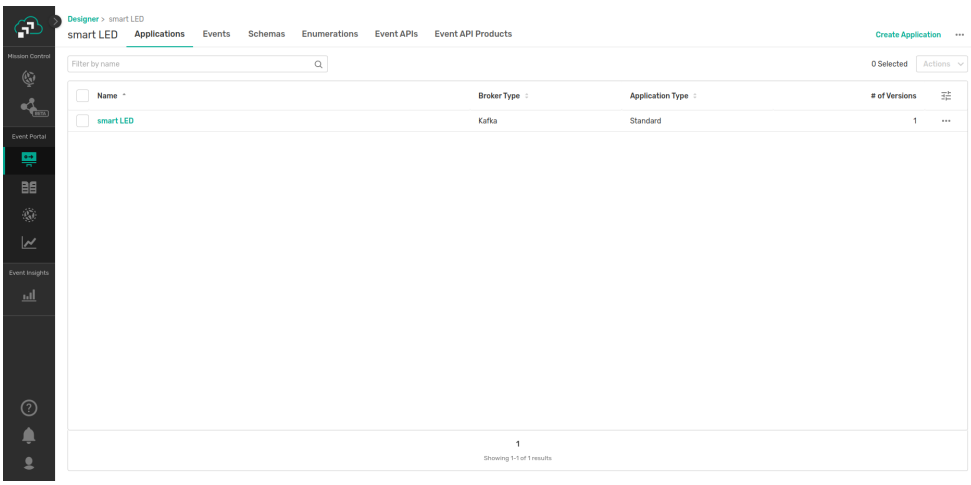


Figure 2.4: PubSub+ Designer with the example application 2.1.2.2

2.3.2 PubSub+ RunTime Event Manager

The Runtime Event Manager enables the creation of EDA models using the objects that were created in the Designer and the data from the user's brokers. The most relevant feature for the purpose of this thesis is the Architecture window, where the user can see a diagram model of our EDA.

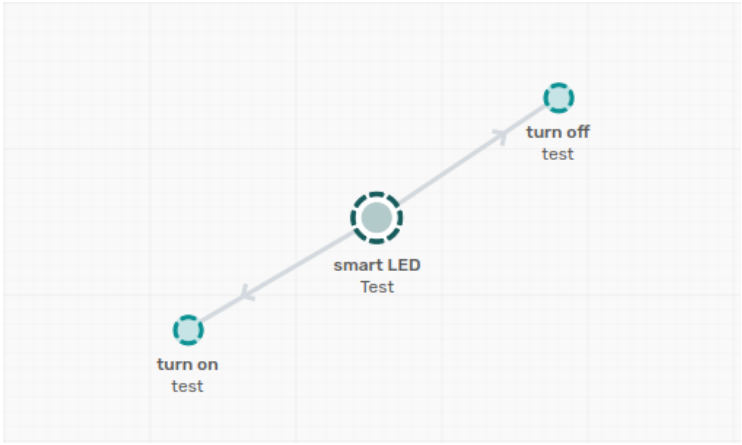


Figure 2.5: PubSub+ RunTime Event Manager with the example application 2.1.2.2

2.4 AsyncAPI Toolkit

The AsyncAPI toolkit [16] offers two ways to design Event-Driven Architectures. The first one is a text-editor-like approach that complies with the AsyncAPI Specification with the purpose of writing and validating AsyncAPI Documents. The second one is a graphical approach where the user can model and annotate the messages that are sent among all the applications. The user can further generate an AsyncAPI document based on the model.

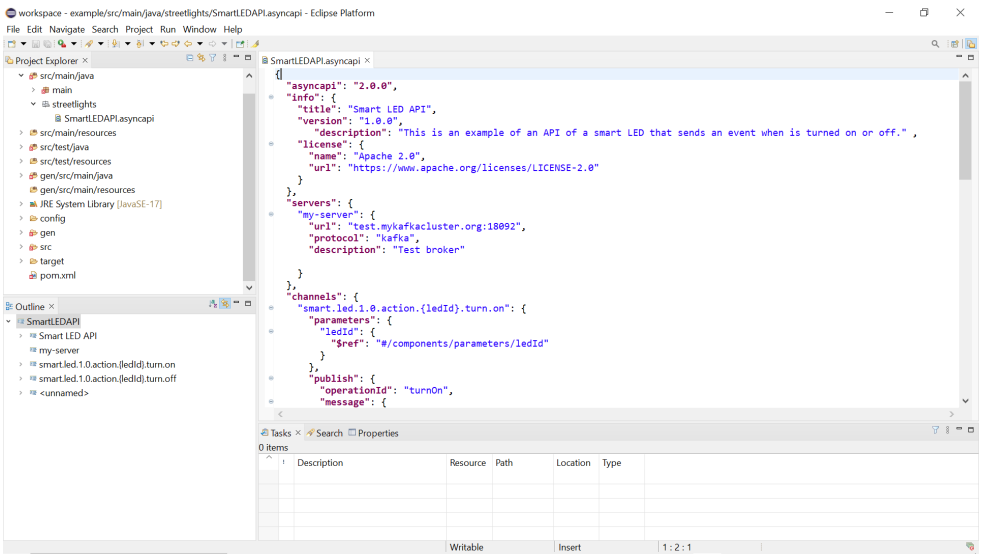


Figure 2.6: AsyncAPI Toolkit with the example application 2.1.2.2

2.5 Apicurio Studio

Apicurio Studio [5] is a web-based tool for designing APIs. Although it focuses more on OpenAPI, it also provides an option to design AsyncAPIs. The main difference between the development

of both approaches is the ability to have a user interface (UI) preview of the API for OpenAPI generated with Redoc[1] while it doesn't exist for the AsyncAPI. Another downside of this tool is that the latest AsyncAPI Specification version a user can use is 2.0. On the other hand, it provides a no-code way to generate an AsyncAPI document with an easy-to-use interface with forms to fill each document component.

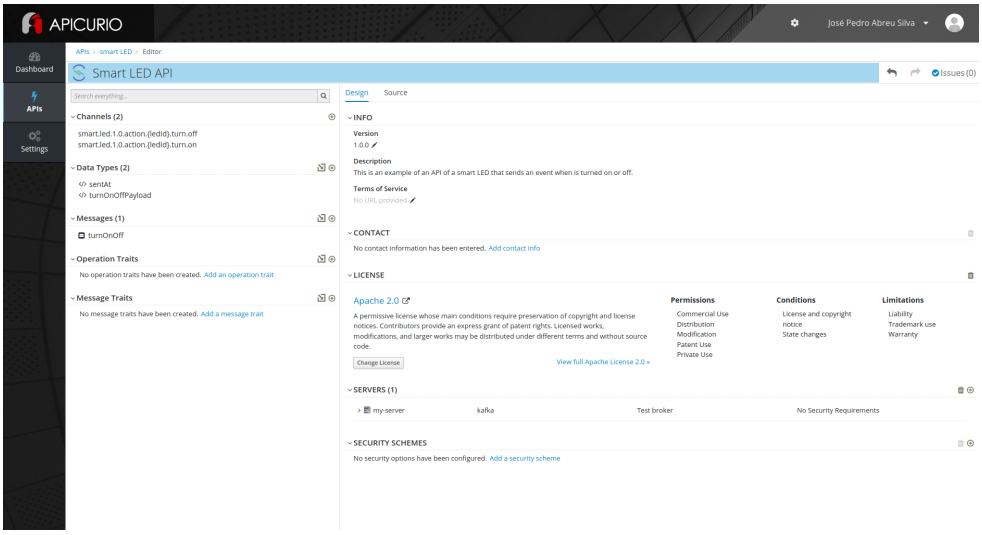


Figure 2.7: Apicurio Studio AsyncAPI's Interface

2.6 OpenAPItoUML

The OpenAPItoUML [13] is a tool very similar to AsyncAPI Toolkit 2.4. Its only feature is to generate an UML based in a OpenAPI Document. The whole processes to do this consists in three steps. The first one is to extract the model from the OpenAPI Document. The second step is to do a model-to-model to transform the extracted OpenAPI Model into a UML Model. The last step is to serialize the UML model as an XMI file, the standard format for UMLs.

2.7 Comparative Analysis

This section presents a comparative analysis of all the tools presented in the past sections. All the evaluation criteria are presented in Table 2.1. In Table 2.2 there is a summary of all the criteria that will be discussed below:

- **Generate an AsyncAPI Document** - OpenAPItoUML is not able to generate AsyncAPI Documents. AsyncAPI Studio and AsyncAPI CLI are the only ones that can generate AsyncAPI Documents with the latest version of the AsyncAPI Specification. However, these tools only provide a text-based approach to generate these documents. PubSub+ Event Portal and Apicurio Studio provide a friendly user interface to generate the Document and a

[1] https://redocly.github.io/redoc/

text-based approach. Lastly, AsyncAPI Toolkit only supports the generation of an AsyncAPI Document if the user provides a UML of EDA.

- **Generate an OpenAPI Document** - The Apicurio Studio, the OpenAPItoUML, and the PubSub+ Event Portal are the tools capable of generating OpenAPI Documents. The first one was made for this purpose, although it can also generate AsyncAPI documents. PubSub+ Event Portal, on the other hand, is more oriented to the design of AsyncAPI Documents.

- **Generate UI documentation** - PubSub+ Event Portal is the only tool that can't generate UI documentation. AsyncAPI Studio and CLI can generate HTML and Markdown Documentation. The AsyncAPI Toolkit and OpenAPItoUML can only generate UML documentation. Lastly, Apicurio Studio can generate HTML documentation but only for OpenAPI. AsyncAPI preview for Visual Studio Code can only generate documentation in the form of HTML.

- **Generate graph documentation** - The PubSub+ Event Portal is the only tool capable of generating a graph of the EDA. This feature is really handy to visualize the whole architecture.

	Description
1	Ability to generate an AsyncAPI Document
2	Ability to generate an OpenAPI Document
3	Ability to generate UI documentation
4	Ability to generate graph documentation

Table 2.1: Tools comparison criteria

	Criterion 1		Criterion 2	Criterion 3	Criterion 4
Tool	Generate an AsyncAPI Document	AsyncAPI version	Generate an OpenAPI Document	Generate UI documentation	Generate graph documentation
AsyncAPI Studio	✓	3.0		✓	
AsyncAPI CLI	✓	3.0		✓	
AsyncAPI Preview for VS Code	✓	3.0		✓	
PubSub+ Event Portal	✓	2.5	✓		✓
AsyncAPI Toolkit	✓	2.0		*UML*	
Apicurio Studio	✓	2.0	✓	OpenAPI only	
OpenAPItoUML		-	✓	*UML*	

Table 2.2: Tools Analysis Summary

2.8 Summary

The AsyncAPI Documents are the most used and well-structured way to define Event-Driven architecture. However, the lack of easy-to-use tools can make the whole process hard. AsyncAPI provides AsyncAPI Studio and AsyncAPI CLI and AsyncAPI Preview for Visual Studio Code which are tools that can generate and validate Documents from text. These tools use the latest AsyncAPI Specification version as well as older ones, but being text-based only can make it difficult for users who are not comfortable with the AsyncAPI Specification's components to generate valid Documents. PubSub+ Event Portal and Apicurio Studio provide forms to fill the components of a Document, making the creation of a Document easier. A third way to create an AsyncAPI Document can be from a UML using the AsyncAPI Toolkit. Regarding documentation, a part from AsyncAPI Preview for Visual Studio Code that can only generate HTML documentation,the tools provided by AsyncAPI can generate both HTML and Markdown documentation. The Apicurio Studio can also generate HTML documentation, but only for OpenAPI. On the other hand AsyncAPI Toolkit and OpenAPItoUML provide UML documentation. The first provides this type of documentation for AsyncAPI, and the second for OpenAPI. Regarding visualization of the Event-Driven Architecture, PubSub+ Event Portal can generate a graph of the whole architecture, which is a handy feature for understanding the connections between applications. To conclude, no tool provides both UI and graph documentation and uses the latest version of the AsyncAPI Specification, which is a gap in the overall body of knowledge of Event-Driven Architectures.

Chapter 3

Problem Statement

In this chapter, the problem intended to be addressed in this thesis will be described in detail as well as the strategy used to create a solution. Firstly, Section 3.1 p.16 presents the current issues of the existing tools to design Event-Driven Architectures. Section 3.2 p.17 outlines the research questions that will be worked on. The scope of the work is described in Section 3.3 p.18. Section 3.4 p.18 presents how the solution will be validated. Lastly, Section 3.5 p.18 summarizes all the chapters.

3.1 Current Issues

Even though a large number of organizations already use Event-Driven Architectures (EDAs) due to their ability to fulfill the demand for consistency, scalability, and resilience in modern systems, it is still a recent field with frequent updates in the industry standard technologies and reduced number of tools to design and interact with them.

This makes the task of designing these architectures harder than it could be because architects and developers have to learn about new versions of technologies and specifications rules with a high frequency and still remember older versions since, as is presented in Section 2.7, not all available tools work with the most recent version of the AsyncAPI Specification, the industry standard to define asynchronous APIs.

Overall, after the comparative analyses in Section 2.7, there are some problems with the existing tools that can be listed. As said before, the first one is that not all of them are compatible with the most recent version of AsyncAPI (version 3.0). This can be a down point because, for instance, if an organization has to create an architecture and its developers choose to use different

tools to define its applications, the end result could be the existence of Documents with different versions of the AsyncAPI Specification.

Another problem is that not all the tools can generate UI documentation based on an AsyncAPI Document. This could be an issue because, especially for new users of the AsyncAPI, reading and understanding how an application works by only reading the AsyncAPI Document can be a challenging task. A third problem is that most of the tools rely on a text-based approach to generate AsyncAPI Documents, which can be time-consuming if the user is not completely comfortable with the current version of AsyncAPI being used.

Finally, the last problem is the nonexistence of a feature to visualize the relationship between the applications of an Architecture in most of the tools. This is a handy feature because if we change how an application works in this type of Architectures, for instance, a producer, it will cause some effect in applications that have any communication to it, so it is really useful for architects and developers to have an easy way to understand how the whole Architecture works.

3.2 Research Statement

The following hypothesis was claimed in the research of this thesis:

H: A tool capable of creating an AsyncAPI Document with an intuitive Graphical User Interface reduces the time spent on creating an AsyncAPI Document and the error proneness.

By *creating an AsyncAPI Document*, it is assumed that the application is capable of creating valid AsyncAPI Documents.

By *intuitive Graphical User Interface*, it's assumed that the application has an interface that is easy to understand and navigate.

By *time spent*, it is assumed that creating an AsyncAPI Document using the solution is faster than using the tools that already exist.

By *error proneness*, it's assumed that a user doesn't commit so many errors while creating an AsyncAPI Document using the solution as he does with other tools.

Based on the hypothesis, we will try to address the following research questions (RQs):

RQ1 *Does the solution help developers by reducing the time spent creating an AsyncAPI Document?*

Having an easy-to-use Graphical User Interface (GUI) can make the task of creating an AsyncAPI Document easier.

RQ2 *Does the solution help developers by reducing the number of errors made while creating an AsyncAPI Document?*

Having a structured form to fill out with information of the AsyncAPI Document can reduce the number of errors made while creating an AsyncAPI Document.

3.3 Scope

After discussing the current issues in Section 3.1 , and analyzing and comparing all the available tools in Section 2.7, it was decided to create a solution with some of the best features from the available tools and some other that fulfill the current issues that those have.

The first step was to create an easy-to-use graphical user interface (GUI) that enables the user to fill the AsyncAPI Document's components using forms like Apicurio (Section 2.5).

The second feature is validating the AsyncAPI Documents. To do this, the open-source tools provided by AsyncAPI will be used since they are the ones that are more reliable and up-to-date. These tools will also be used to generate more types of Documentation from the AsyncAPI Documents, such as an HTML preview of the document's details.

The third feature to implement is a way to create and validate schemas and use them as a reference in the Document. This would ensure the AsyncAPI Document's overall readability, making it easier to understand.

3.4 Validation

After the development of the solution defined in Section 3.3, the solution was validated to confirm if it fulfills the needs listed in the research questions listed in Section 3.2. The validation was done in two main parts. The first one was the evaluation of the Documentation generated by the tool. The second one was the evaluation of the feasibility of the tool.

On the one hand, to validate the generation of the Documentation by the tool, real use cases were used to ensure that everything generated was correct and without errors.

On the other hand, to validate the tool's feasibility, a user study was conducted with subjects with different levels of expertise to determine whether the interface improves the user's experience or not.

3.5 Summary

In this chapter, it is described and evaluated the current issues with the available tools. The more relevant ones are concerned with using the most recent version of the AsyncAPI Specification, its usability, and the ability to generate other types of documentation than AsyncAPI Documents. After that, we described the solution and the strategy to develop and validate it.

Chapter 4

Solution Prototype

In this chapter, the solution prototype is explored, and its features and architecture are described. It also discussed the reasons behind the major decisions faced throughout the development process.

4.1 Overview

The developed prototype is a web application designed to facilitate the creation of AsyncAPI Documents through a user-friendly interface. Users fill out forms corresponding to different components of an AsyncAPI Document, and the application generates a complete Document based on the input. The application also enables users to upload and validate different schemas and examples to be used in the document as a reference and allows them to download not only the AsyncAPI Document but also a zip file containing the AsyncAPI Document and all the schemas' files used as a reference in a well-structured way.

4.2 Architecture

The architecture of the web application is designed to be scalable and maintainable. It comprises two main components: the frontend, built with React [21], and the backend, built with Node.js [15] and Express.js [30]. The decision to use a web frontend instead of a native one was made because of the benefits in terms of platform compatibility, which ensures that the application having cross-platform accessibility can reach the widest possible audience, and in terms of flexibility since a

web application can leverage from the vast number of libraries and tools available in the web development ecosystem such as UI components libraries or validation libraries. The frontend communicates with the backend via RESTful API [28] calls using Axios [1], and all the validation and processing of the form's data are done in the backend. On top of all that, to ensure that the apps run consistently across different environments, both apps are containerized using Docker [19]. Figure 4.1 displays the prototype's deployment diagram. It may seem that the user interacts with both frontend and backend through its device, but it interacts only with the frontend app, and only the frontend running in the user's device interacts with the backend.

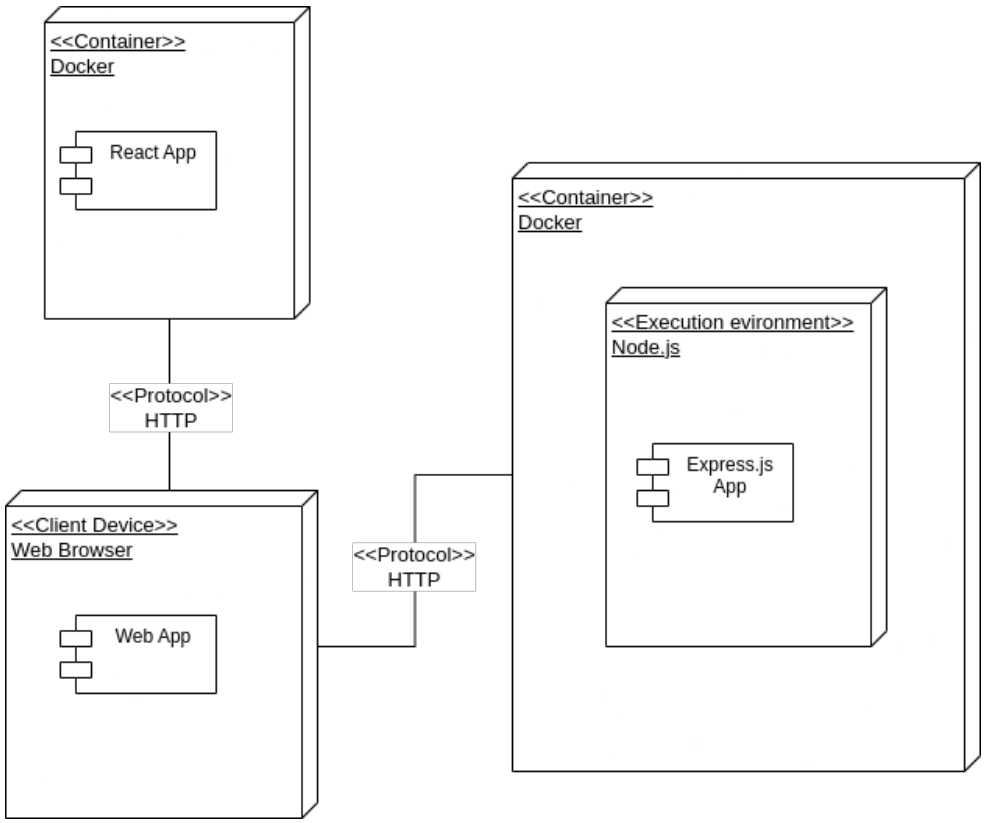


Figure 4.1: Prototype's deployment diagram

4.3 Technological Decisions

The first decision that was discussed was what technologies should be used for both the frontend and backend. React [21], a component-based Javascript library for building user interfaces, was chosen for the frontend. For the backend, the decision was to make a Node.js[15] application using the Express.js Framework[30]. The fact that AsyncAPI provides Node.js libraries with tools to use with AsyncAPI Documents, such as the AsyncAPI bundler and the AsyncAPI parser, made this an easy decision. On the other hand, having a unified JavaScript environment across the frontend and backend would make the development of the whole app more straightforward, making Node.js[15] and Express.js[30] a great fit for the backend.

One of the most important decisions to make was how to implement the forms section to fill the data of the AsyncAPI Document. Since an AsyncAPI Document can have different fields depending on the application usage, and different companies have their own mandatory fields for their application's Documents, having only one form structure that fits all the needs would be a hard task to do.

The first idea to solve this problem was to create a form that would fit most of the AsyncAPI Documents with the mandatory field of the AsyncAPI specification and the most used optional field as arbitrary to fill up or not. This idea was doable but wasn't optimal since many AsyncAPI Documents would still need to be done by hand.

The second idea for this problem and the final solution was to use JSON Forms [12], which is a library that supports React [21], Vue [32], and Angular [17] designed to generate forms dynamically based on a JSON Schema [26], making it easy to update the form structure by just changing the schema. It also provides validation of the input based on the schema rules. With this solution, the application can provide a default form structure that fits the needs of most cases but can also provide a way to change the form and make it fit a specific AsyncAPI Document structure.

Another decision made was regarding the components field of an AsyncAPI Document. Since it was decided to use schema files as references in the generated Document, the components field would present a certain level of redundancy since this field's purpose is to be used as references to other fields. So the final decision was to have the other four AsyncAPI Components (Info field, Servers field, Channels field, and Operations field) and one form to add other arbitrary fields to the document.

Regarding creating schemas and using them as a reference, the first idea was to upload the schemas files into the application. However, this would make it necessary to use an external tool like a code editor to create them. The first solution to this was to implement a text input so the user could create the schemas in there. After discussing possible problems regarding the format of the schemas, implementing a small code editor inside the application was the final solution.

4.4 Feature Design

In this section, the main features implemented in the prototype will be described, and its design considerations will be discussed.

The application is divided into three user interfaces where the user can manage the applications, the schemas and examples used as references, and the settings where the user defines the form's structure.

Figure 4.2 displays a user interface sample for the prototype's main view: the applications dashboard. This is where the user can create an application's AsyncAPI Document, and it is divided into six main features:

- **Main Navbar**. Located at the top of the screen, the main navbar enables the user to navigate through the three dashboards: Applications, Schemas, and Settings.

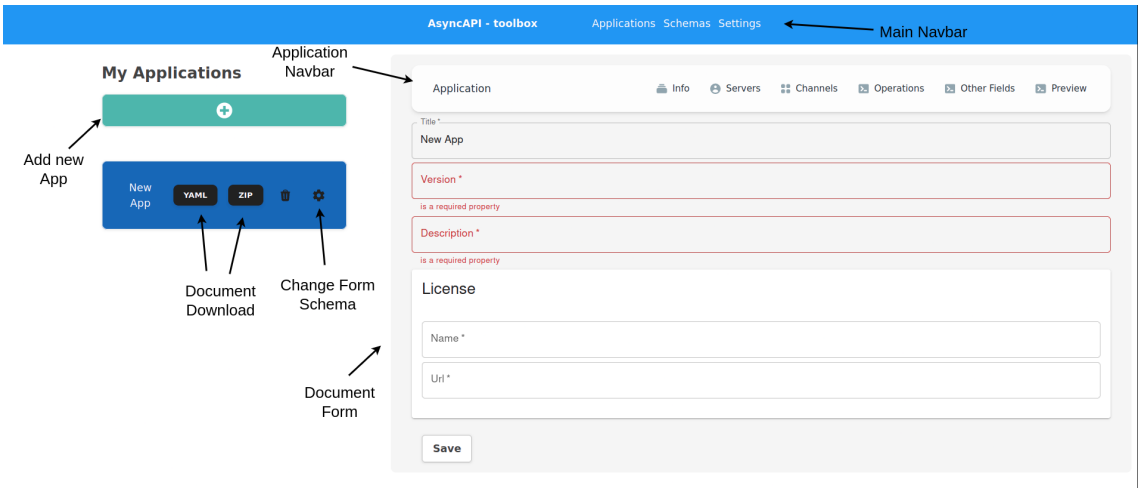


Figure 4.2: Layout of the prototype's applications Dashboard

- **Application Navbar**. On top of the right side panel, the Application navbar enables the user to navigate through the forms of each AsyncAPI Component and to the preview of the Document.
- **Document Forms**. The Document forms are where the user can fill the data of each Document field. There are five forms : Info, Servers, Channels, Operations, and Other Fields.
- **Document Download**. This is where the user can download the YAML file of the document or a ZIP file containing the YAML file and all the schema files used as reference.
- **Add new App**. This is where the user can add a new application to the applications list. Firstly, the user will have to choose one setting for the forms and then can start creating the application's AsyncAPI Document.
- **Change Form Schema**. This is where the user can change the form structure used to fill out the application's information.

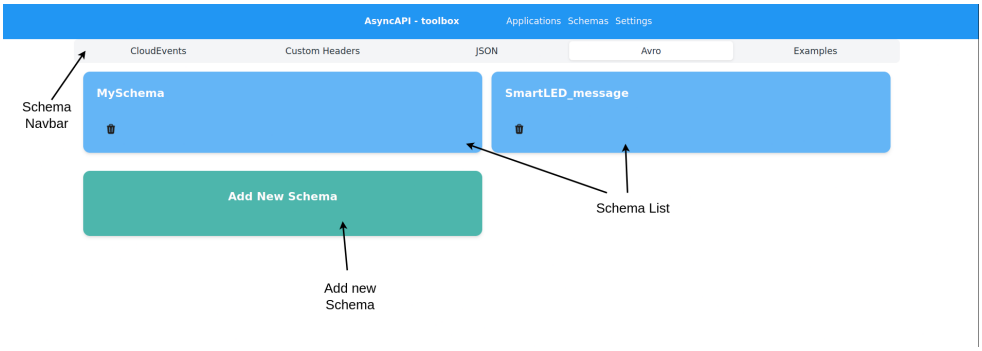


Figure 4.3: Layout of the prototype's schemas Dashboard

Figure 4.3 displays a user interface sample for the schemas dashboard. This is where the user can create schemas and examples to use as reference in the AsyncAPI Document, and it is divided into three main features:

- **Schemas Navbar**. Located at the top of the panel, the schemas navbar enables the user to navigate through the five dashboards corresponding to four schema formats and examples: CloudEvents[31], Custom Headers, JSON[26], avro[3] , and Examples.
- **Schema List**. A list of added schemas where the user can access schemas that were previously added.
- **Add Schema**. The add schema button, when pressed, displays a small code editor where the user can create new schemas and add them to the schema list.

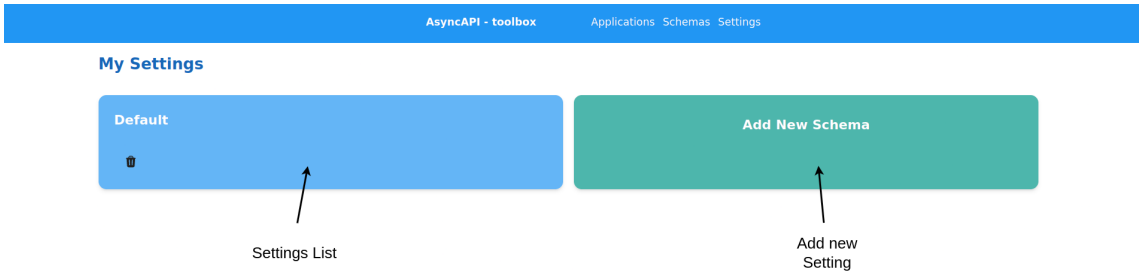


Figure 4.4: Layout of the prototype's settings Dashboard

Figure 4.4 displays a user interface sample for the settings dashboard. This is where the user can create schemas to define the structure of the forms used to generate the AsyncAPI Document. It is similar to the schemas dashboard interface and is divided into two main features:

- **Settings List**. A list of added settings schemas where the user can access previously added schemas.
- **Add new setting**. When pressed, the add new schema button displays a small code editor where the user can create and add new schemas to the schema list.

4.4.1 Dynamic Forms and Settings

As previously described in Section 4.3, the JSON Forms library [12] was used to dynamically create the forms to fill out the AsyncAPI Document's fields based on a JSON Schema [26]. This simplified the data validation and solved the problem of the need for different arbitrary fields to exist. However, the way the user changes the JSON schema [26] of the forms, how the forms data

are converted to the AsyncAPI Document fields, and how the schema files are used as a reference in the Document still have to be handled.

Since there are five forms to fill out to create a document (Info, servers, channels, operations, other fields), five JSON schemas are needed to create a complete form structure. To define a complete structure, it was decided to join all the form's JSON schemas in one unique object with five different objects inside, one for each single form. Each inner object has the JSON schema for each form.

```
{
 "info": {/*JSON Schema*/},
 "servers": {/*JSON Schema*/},
 "channels": {/*JSON Schema*/},
 "operations": {/*JSON Schema*/},
 "otherFields": {/*JSON Schema*/}
}
```

Listing 4.1: Sample Settings JSON file to define the forms structure.

Going more in-depth into the form's schema definition for an asyncAPI Document generation, there were some problems that needed to be handled.

The first one was the definition of **$ref** fields used to define references in the document. The problem with this is that the library uses the **$ref** keyword to find references to other schemas and can't be handled as the name of a field in the form. To solve this, the keyword **toolbox$ref** can be used to create **$ref** field. The workflow of this arrangement is that the form's field will be named **toolbox$ref**, the data generated by the input will be an object with the key **"toolbox$ref"**, and then this key will be converted to **$ref** as shown in figure 4.5.

Figure 4.5: Example on how the "toolbox$ref" is processed.

The second problem to solve is handling nested objects with arbitrary key names in the AsyncAPI Document. Examples of this can be seen in the definitions of servers, channels, and operations. As shown in the sample example below, the servers component is composed of objects with the server's name in its key.

```
"servers": {
    "myServer": {
        /* Fields */
    },
    "myOtherServer": {
        /* Fields */
```

```
        }
}
```

Listing 4.2: Example on how the server's name is defined in the Servers Component.

To solve this problem, these types of fields are defined in the forms' schemas as an object with an array of objects with the name field, and when the input data is generated from the form, the name field of each object in the array is used as a key to defining each object. Figure 4.6 displays the whole workflow of this process.

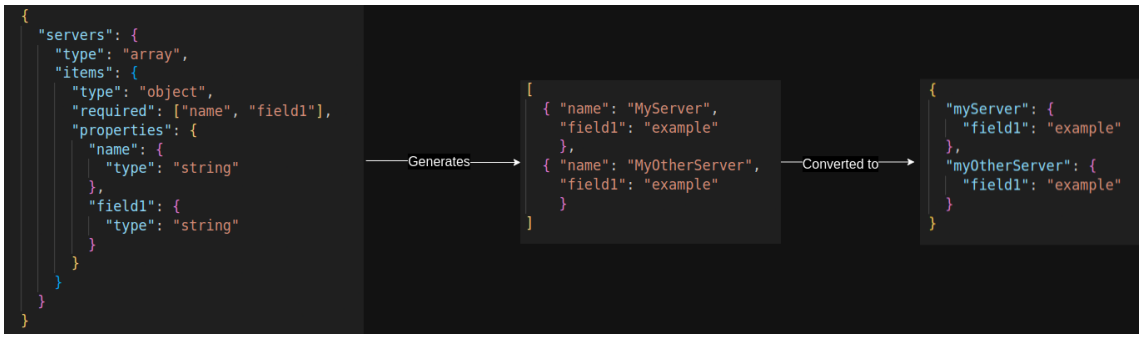


Figure 4.6: Sample on how the "name" field is handled by the application.

This process is done in the Servers, Channels, and Operations Components. It is also done in the Messages component inside the Channels component.

Lastly, having a list of the paths to the created schema files in the form to use them as a reference would improve the user experience in creating an AsyncAPI Document. This could be reached by having a field with an "enum" type and an array of the schemas' paths. However, updating the form's schema every time a schema is created is not a good option because it would require more time than just writing the path. To solve this problem, a feature was implemented that, before rendering the form, checks if the schema has a list with the keyword **"$ToolboxSchemas"** and converts it to a list of the paths to all the schemas. Figure 4.7 displays the whole workflow of this process.

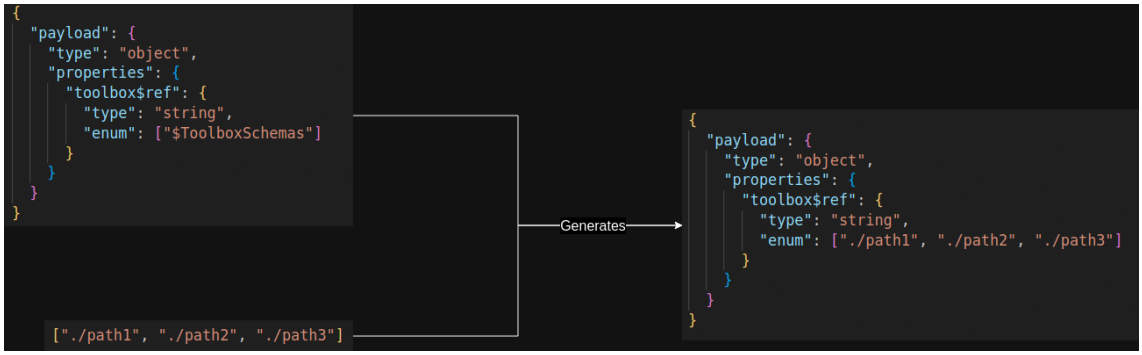


Figure 4.7: Sample on how "$ToolboxSchemas" is handled by the application.

4.4.2 Schema Creation, Visualization and Validation

As described before in Section 4.3, the first idea to have schemas in the application was to upload the schemas files directly into it. However, this would make it necessary to use an external tool like a code editor to create them. After some discussion, implementing a small code editor inside the application was the final solution. To do this, the monaco-editor [22], the editor that powers Visual Studio Code [23], was implemented in the schemas dashboard. Every time the user presses the button to add a new schema in any of the schemas tabs, the editor pops up, and the user can create a schema there and save it in the application as shown in Figure 4.8.

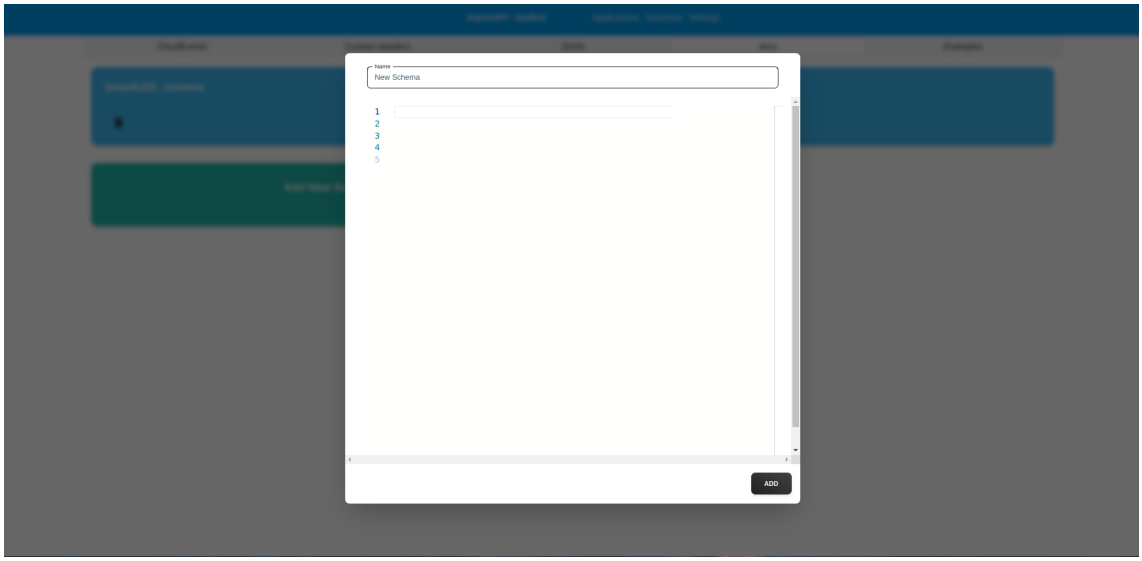


Figure 4.8: Monaco-editor implemented in Schemas Dashboard

To visualize the schemas, react-code-block [24], which provides React components for rendering code snippets with syntax highlighting, was used. The monaco-editor [22] could be used again to perform this, but this more lightweight solution was chosen because most of the editor's features weren't needed for this proposal. Figure 4.4 how the added schemas are displayed to the user.

Although the code editor already displays syntax errors when creating a schema, it is validated in the backend after the schema is added to the application. To do this, different libraries are used according to the type of schema added. For avro schemas, avro-js [4] is used to parse, validate, and generate examples based on the schema. For cloudEvents and customHeaders schemas, it is used the yaml library to parse and validate. For JSON schemas and examples, the javascript's JSON methods are used to parse and then validate the schemas. If an invalid schema is added to the application, that schema will show up as red in the schemas list as shown in Figure 4.10 and won't be available to use in the Application's forms.

Figure 4.9: Schema Visualization in Schemas Dashboard.

4.4.3 AsyncAPI Preview and Validation

To preview the created AsyncAPI Document, the asyncapi-react [10] component is implemented in the application. However, this is not enough to have an easy-to-read HTML application documentation. Since we are using references to the schemas' files in the document and the component does not support references, there is a need to create a unified document first. For this matter, the application uses the asyncapi-bundler [7] library that merges all files into one AsyncAPI Document. Before the files are merged together, all documents are firstly validated using the asyncapi-parser [9], which is a package that enables the parsing and the validation of an AsyncAPI Document. If the files are invalid, a diagnostic list will be shown in the Application Preview tab as shown in Figure 4.11.

If everything is valid, all the files are merged together and passed to the asyncapi-react [10] component that renders the HTML Documentation as shown in Figure 4.12

4.4.4 YAML and ZIP files

The application provides two possible formats for downloading the AsyncAPI Document. The first one is the AsyncAPI Document in YAML format, the most used format for this type of Document. The second one is a ZIP file containing the asyncAPI Document and all the files used as a reference in the document. In the ZIP file, the files are separated into different folders corresponding to the file type, as shown in Figure 4.13.

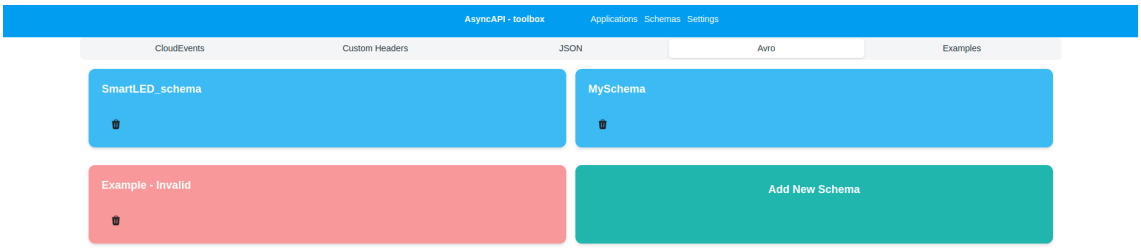


Figure 4.10: Schema List with invalid schema.

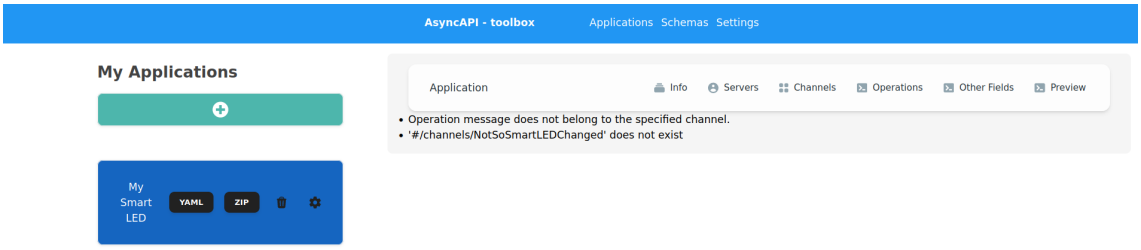


Figure 4.11: Preview of an invalid AsyncAPI Document.

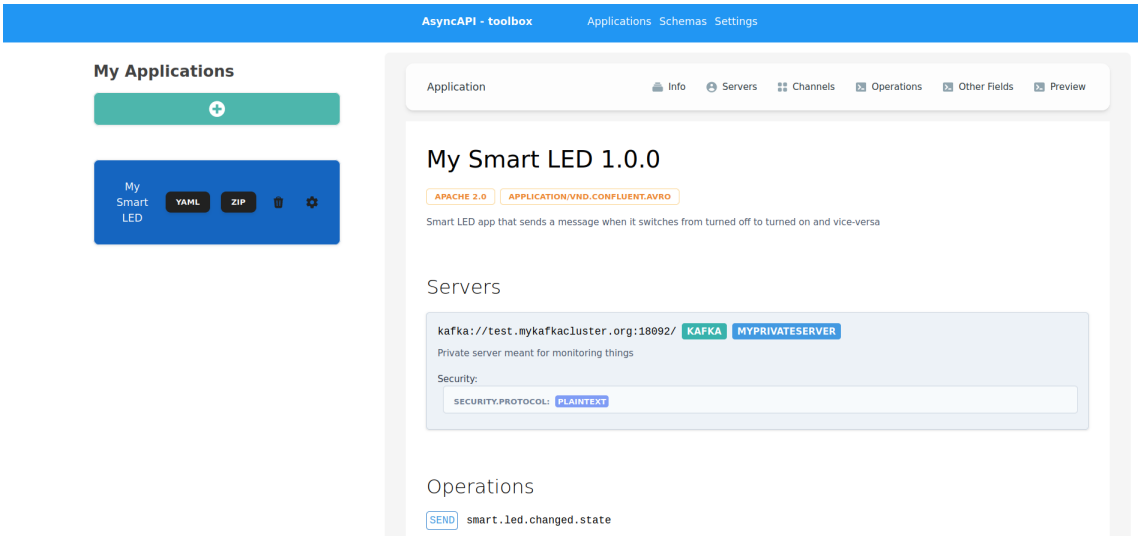


Figure 4.12: Preview of a valid AsyncAPI Document.

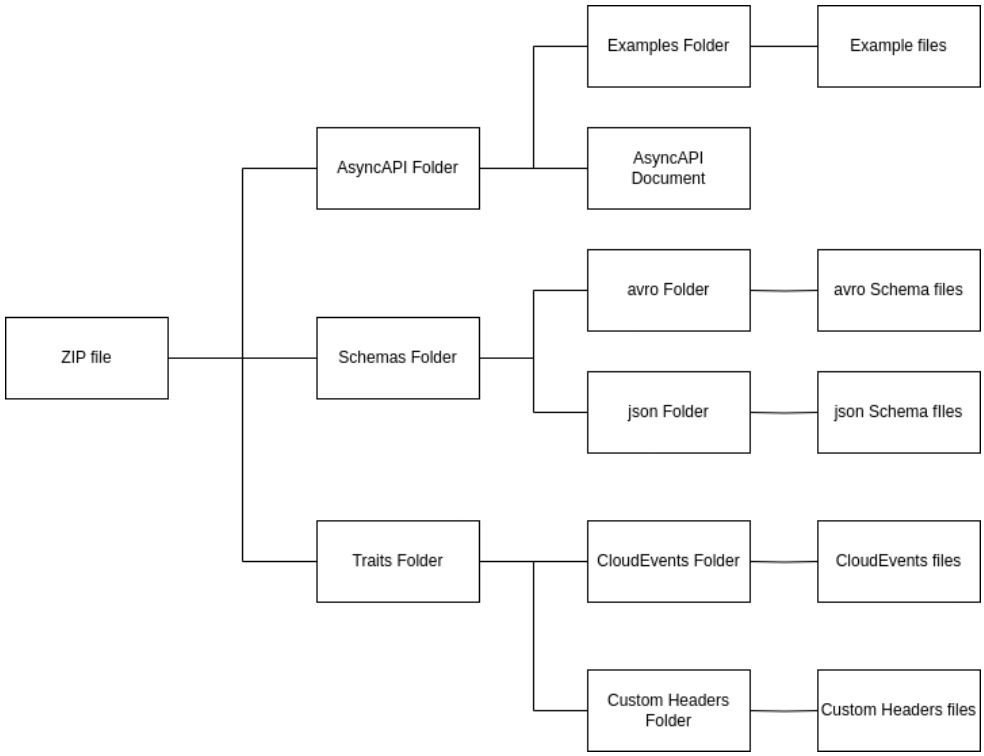


Figure 4.13: ZIP file structure.

4.5 Installation

Since the prototype is containerized with Docker [19], users only need to have Docker installed on their machines. A docker-compose file is presented in the prototype's root folder, so users only have to run the command "docker-compose up" to run it.

4.6 Discussion

This chapter focused on describing the developed prototype, detailing its architecture and implemented features, and explaining the ideas behind each decision. The developed prototype's features differ from the features of the tools analyzed in the State of the Art Chapter 2. Comparing it to the analyzed tools, it is not wrong to say that it fits between the tools provided by AsyncaAPI and apicurio since it provides a way to see the app's documentation in an easy-to-read HTML template and has validation features like the first and an easy-to-use interface with forms to fill out the document's component like the second. However, the prototype is still very different from all the tools. The creation of an asyncAPI Document using references to schema files can be only done using the AsyncAPI CLI and the AsyncAPI Preview extension for Visual Studio Code but only in a text-based way in which the user writes the whole Document and the tools are only used to validate. Due to time limitations, it was not possible to implement a way to generate graph documentation that would show the relationship between apps, channels, and brokers.

Chapter 5

Validation

This chapter describes and documents the process used to validate the designed prototype by comparing the prototype's usage to other existing tools in different scenarios and then documenting the empirical study made.

5.1 Methodology

In order to validate the prototype's viability, the study went through two main phases to try to answer the **Research Questions** (Section 3.2):

- **Scenario Driven Experiments:** Several scenarios were defined in order to compare possible pathways to the process of completing different tasks using the prototype and the tools provided by AsyncAPI.
- **Feasibility User Study:** The study considered two treatments – **experimental** in which the participant created an AsyncAPI Document by following a set of tasks using the developed prototype and **control** in which the participants created solved the same set of tasks but using the AsyncAPI Preview extension for Visual Studio Code.

These phases will be described in depth in the following sections.

5.2 Scenario Driven Experiments

All the scenarios defined in this section will explore a step-by-step description of possible pathways for completing different tasks using the developed prototype and using the tools provided by

AsyncAPI. These scenarios will test different aspects of the tools: validation, creation of different Documents with the same structure, and change of structure of a previously created Document. All the scenarios will be described in four sections:

Script The script section defines the task using a hypothetical real scenario. Three fictional characters are used for this. Alice is the character performing the task in the developed prototype, Bob uses the AsyncAPI tools, and Carol is the person dictating their tasks.

Alice's perspective This section presents all the steps that Alice did to perform the tasks using the developed prototype.

Bob's perspective This section presents all the steps that Bob did to perform the tasks using the AsyncAPI tools.

Comparison Conclusion This section compares both perspectives and presents a brief conclusion about it.

5.2.1 Scenario 1 - Validation

Script

Carol has a Smart LED that sends a message to her private monitoring server when turned on or off, but it still doesn't have an AsyncAPI Document, so she asked her friends Alice and Bob to make one. She provided them with all the information needed to create all the AsyncAPI components, a file with a schema of the message that the app sends, and a file with the traits of the message in CloudEvents format. However, the schema has some error, so Alice and Bob have to figure that out and tell her there is a problem with the schema.

Alice's Prespective

- **Step 1 -** Open the prototype.
- **Step 2 -** Add a new app in the Applications dashboard.
- **Step 3 -** Fill up the Info form with the information provided by Carol.
- **Step 4 -** Fill up the Servers form with the information provided by Carol.
- **Step 5 -** Go to the Schemas dashboard and add a new CloudEvents schema with the schema from the file Carol sent her.
- **Step 6 -** Go to the avro tab and add a new avro schema with the schema from the file Carol sent her.
- **Step 7 -** A comment pops up saying the schema is invalid, so Alice informs Carol about it.

Bob's Prespective

- **Step 1 -** Create an empty folder and open it with Visual Studio Code.
- **Step 2 -** Create three folders called "asyncapi", "schemas", and "traits".
- **Step 3 -** Create a file in the "asyncapi" folder called "Smart-LED-app.yml".
- **Step 4 -** In the created file, create the Info component with the information provided by Carol.
- **Step 5 -** Create the Servers component with the information provided by Carol.
- **Step 6 -** Copy the traits file that Carol sent him into the "traits" folder.
- **Step 7 -** Copy the schema file that Carol sent him into the "schemas" folder.
- **Step 8 -** In the "Smart-LED-app.yml" file, create the Channels component with the information provided by Carol.
- **Step 9 -** Create the Operations component with the information provided by Carol.
- **Step 10 -** Open the command palette (Ctrl + Shift + P) and click on the "Preview AsyncAPI" command.
- **Step 11 -** The preview tab shows a blank page, which means there is an error in the Document, so Bob rechecks the whole document and can't find any error.
- **Step 12 -** Open a terminal to diagnose the document using the AsyncAPI Cli validation command.
- **Step 13 -** Run the command "asyncapi validate ./asyncapi/Smart-LED-app.yml".
- **Step 14 -** The output says the schema is invalid, so Alice informs Carol about it.

Comparison Conclusion

Since the AsyncAPI preview extension for Visual Studio Code doesn't provide any diagnostic about the document's errors, Bob only discovered that the schema Carol sent him wasn't valid after completing the whole Document and using the AsyncAPI Cli to figure out what was wrong in it. On the other hand, Alice only had to do half of the steps of Bob because the developer prototype validates the schema when it is added to it.

5.2.2 Scenario 2 - Template-based generation

Script

Carol has a Smart LED that sends a message to her private monitoring server when turned on or off, and she already has a well-structured folder with documentation about its functionality with the same structure presented in Figure 4.13. However, she bought five more Smart LEDs and asked her friends Alice and Bob to make her the documentation of her new LEDs. The only difference between each document is a custom field in the AsyncAPI Document with the name application-ID. Alice and Bob had already made the documentation for her first Smart LED, so they already have the base template. The id of each new LED will be 2,3,4,5,6.

Alice's Prespective

- **Step 1 -** Open the prototype.
- **Step 2 -** Select the previously created document.
- **Step 3 -** Go to the Other Fields form and change the "application-ID" field.
- **Step 4 -** Download the zip.
- **Step 5 -** Repeat **step 3** and **step 4** four more times.

Bob's Prespective

- **Step 1 -** Create five copies of the previously created document.
- **Step 2 -** Open the folder where the five copies were created.
- **Step 3 -** Open the the "Smart-LED-app.yml" file of one of the copies.
- **Step 4 -** Change the "application-ID" field.
- **Step 5 -** Repeat **step 3** and **step 4** four more times.

Comparison Conclusion

Bob's work was a little bit more tiring because he had to copy the previously created folder five times and navigate in each one of the copies to change the "application-ID" field. On the other hand, Alice only had to change the field and click the download zip button five times and didn't have to make any copies or navigate through the files.

5.2.3 Scenario 3 - Adding a new field to a Document

Script

Carol has a Smart LED that sends a message to her private monitoring server when turned on or off, and her friends Alice and Bob have already created a well-structured folder with documentation about its functionality with the same structure presented in Figure 4.13. However, the info component doesn't have the contact field with her name and email, so she asked her friends to add that to the Document.

Alice's Prespective

- **Step 1 -** Open the prototype.
- **Step 2 -** Go to the Settings Dashboard and open the Default settings schema.
- **Step 3 -** Copy the Default settings schema and add the contact field as shown in Figure X.
- **Step 4 -** Go to the Application Dashboard and add the Contact information in the Info form.
- **Step 5 -** Download the ZIP file.

Bob's Prespective

- **Step 1 -** Open the application's folder in Visual Studio Code.
- **Step 2 -** Open the the "Smart-LED-app.yml" file.
- **Step 3 -** Add the contact field and its correspondent information.

Comparison Conclusion

Alice had a harder task adding a new field to the Document since she had to change the form's schema to be able to add it. On the other hand, Bob only had to write the information for the new field directly in the document.

5.2.4 Summary

In summary, the developed prototype is feasible for performing all the tasks described in the three scenarios. It could perform better when used in the first scenario, where the user needed to validate an invalid schema, and in the second scenario, where the user had to create Documentation using a previously created Document as a base. However, when the user had to add a new field to the Document (Scenario 3), the prototype could underperform in comparison with the tools provided by AsyncAPI since it was easier to just add it to the Document instead of having to change the whole form just to be able to do it.

5.3 Feasibility User Study

This study considered two main treatments – **control** in which participants solved a set of tasks in order to create an AsyncAPI Document for a Smart LED that sends a message to a monitoring server when it is turned on and off using the tools provided by AsyncAPI and **experimental** in which participants solved the same set of tasks but using the developed prototype described in Chapter 4. This study was inspired by the one performed by Bruno Piedade in his research in Visual Programming Language for Orchestration with Docker [27].

The goal of this feasibility user study was to provide answers to the research questions defined in Section 3.2. To answer the **Research Question 1**, the time of completion of each task was considered. To answer the **Research Question 2**, the validity of the Document was checked after the tasks' conclusion. Additionally, a set of perception-based questions (PBQ) was considered to evaluate the environment on completing the tasks, the complexity of the task's description, and the experience using the tools. This was useful because even if the data may suggest that the usage of the developed prototype improves the performance of the creation of an AsyncAPI Document, it would be meaningless if the participants disliked the experience of using it.

5.3.1 Design

The evaluation focused on the completion of different tasks in order to create an AsyncAPI Document. Each task corresponds to the completion of one component of the Async API Document:

- **Task 1** The participants had to create the "Info" Component.
- **Task 2 -** The participants had to create the "Servers" Component.
- **Task 3 -** The participants had to create the "Channels" Component using an avro schema and a ClodEvents schema as references.
- **Task 4 -** The participants had to create the "Operations" Component using the information created in the "Channels" Component as a reference.

After completing all the tasks, the participants were asked to validate the created Document to check whether it was valid.

5.3.2 Participants

Recruitment was done among students and working software developers. The whole recruitment process was conducted through direct communication with the participants. The goal was to recruit subjects who had some prior experience with OpenAPI, AsyncAPI, Request-Response Models, and Event-Driven Architectures. In the end, a total of 12 participants were recruited, of which 75% were students (9 participants) and 25% were working software developers of different companies. The participants were randomly distributed in two groups (6-6) corresponding to the treatments: **control** (CG) and **experimental** (EG). The participants in the control group had access to a zip file

containing an example of an AsyncAPI Document and to Visual Studio Code with the AsyncAPI Preview extension. On the other hand, the participants in the experimental group were given access to the developed prototype. All the participants had access to any resource on the internet for reference. All the participants from both groups were also asked to fill out a background questionnaire that inquired about their experience with the technologies of interest to see if this had any impact on the results. The included experience with OpenAPI, AsyncAPI, avro, JSON, CloudEvents, some communication protocols and tools to manage AsyncAPI Documents.

5.3.3 Data Sources

The data sources used in this evaluation were the following:

- The answers to the background questionnaire.
- The individual task times and the global time to complete all the tasks.
- The answers to the assessment questionnaire.

5.3.4 Environment

All the experimental sessions of the **experimental** group were made in person, while the sessions of the **control** group were made both remotely or in person. The decision to make the sessions of the **experimental** group in person was made because most participants didn't want to run the developed prototype in their machines. All the experiments were done in a quiet place with almost no distractions or noise.

5.3.5 Task Definition

The goal of the task is for the participants to have a valid AsyncAPI Document after completing all the tasks. To do this, a simple fictional application of a Smart LED that only sends a message when it is turned on or turned off was described to them. The idea was to make the simplest possible AsyncAPI Document so as not to overwhelm the participants with information about the application behavior so that they could focus on the tool's functionalities. The task's difficulty increases in the completion order where the first two tasks where the participants only had to complete the "Info" and "Servers" components with the provided information, are quite straightforward, while the third task, the most challenging one, asked them to create schemas and use them as reference in the Document and the last one also needed some attention with the references to the channel and message created on the third task.

5.3.6 Procedure

A full session took between 20 to 50 minutes per participant. All the sessions were conducted individually in the presence of the researcher to clarify any misunderstanding or provide help with any technical problem. To follow their logic, the participants were encouraged to think out loud.

At the beginning of each session, the participants were provided with a Google Form with the instructions for the procedure. The procedure was organized in the following sections:

- **Background questionnaire -** A set of questions to understand the degree of experience with related technologies and tools.
- **Tutorial -** The participants had to follow a simple tutorial reviewing the basis of an AsyncAPI Document and its Structure in order to maintain consistency between participants with a low level of experience with AsyncAPI Documents and participants with a lot of experience. The tutorial provided a set of instructions to explore the tools used in the experiment.
- **Tasks -** A set of four tasks was provided to the participant in order to complete different components of an AsyncAPI Document with a final goal of creating a whole valid and well-structured Document.
- **Assessment questionnaire -** After completing all the tasks, the participants had to fill out a questionnaire to assess their experience and evaluate their experience working with the tools.

All the materials used can be found in Appendix A.

5.3.7 Data Collection

To collect the results of the background questionnaire, the answers to linear numeric scales, multiple-choice questions, and a 5-point Likert scale were used from the result of the form response sheet. To measure the time of each task, the start time and end time that each participant registered were used. The sum of all tasks' timers was the total time spent creating the AsyncAPI Document. The tasks' time results addressed the **Research Question 1**.

5.3.8 Data Analysis

Initially, all the data from the different sources were manually compiled into a single spreadsheet and then uploaded to the SPSS (Statistical Package for the Social Sciences) platform [18]. The platform was used to perform a set of statistical tests, including the Mann-Whitney U (MW-U) and McNemar tests, depending on the characteristics of the variables. It was also used to generate graphs. To determine the acceptance of the alternative hypothesis, a 5% significance level ($\rho < 0.05$) was used, indicating that there was a 95% confidence level for rejecting the null hypothesis.

5.3.9 Pilot Experiments

One pilot experiment was conducted with a person with some knowledge of the project to validate the cohesion of the experimental procedure and all the materials involved. It showed that the materials had minor inconsistencies, such as typos, and some minor problems concerning the order of the tasks' exercises. The results of this pilot experiment helped understand the difficulty of the tasks and solving these minor problems.

5.4 Data Analysis

Since the collected data was all quantitative, all of it was used as target of the statistical tests performed. Most of the hypotheses required a significance test so, **Mann-Withney U** (**MW-U**) and **McNemar** tests were run against the interest variables. To do this, it was adopted a notation during the analysis that denotes $\bar{x}$ as the mean, σ as the standard deviation, u as the U statistic of Mann-Whitney U tests, H_0 as the null hypothesis, H_1 as the alternative hypothesis and ρ as the probability of rejecting H_0.

5.4.1 Background

Considering the answears the the background section of the questionnaire, the goal of this analysis was to determine whether the participants of both groups had the same skill and knowledge level to ensure that the differences of time spent in the completion of the tasks weren't related to the experience difference across both groups.

A summary of the obtained results for the Likert and numeric scale questions is displayed on Table 5.1. Considering the alternative hypothesis H_1 that states that the groups are different (CG $\neq$ EG) for every background question, it is concluded based on the results that there is **not a significant difference** of skill and knowledge level between the two groups. The only question that presents a noticeable disparity between the groups is BQ14 which asked the participants in how many projects have they created/updated an AsyncAPI Document. However, The probability of rejecting the is still greater than the significance level determined ($\rho < 0.05$) so it is not enough to reject the null hypothesis.

Furthermore, the participants were asked to choose from a list of protocols the ones they were experienced with. Figure 5.1 and Table 5.2 display the data and the results of their experience. Considering the alternative hypothesis H_1 that states that the groups are different (CG $\neq$ EG) for the number of protocols that the participants are experienced with, it is concluded based on the results that there is **not a significant difference** of experience level between the two groups. It is interesting to note that as shown in Figure 5.1 all the participants were experienced with HTTP, both groups had the same number of experienced people with WebSockets and Kafka, and the number of experienced participants with MQTT of both groups only differ in one participant.

Besides that, the participants were asked to choose from a list of tools listed in Chapter 2 the ones they were experienced with. Figure 5.2 and Table 5.3 display the data and the results of their experience. Considering the alternative hypothesis H_1 that states that the groups are different (CG $\neq$ EG) for the number of tools that the participants are experienced with, it is concluded based on the results that there is **not a significant difference** of experience level between the two groups. The fact that the **control** group doesn't have any participants that have used any of the tools while the experimental group does have should be noted.

To summarize the background analysis, considering all the data collected and the subsequent analysis, it is valid to assert confidently that the subjects were evenly distributed between both groups.

	CG		EG		MW-U		
	$\bar{x}$	σ	$\bar{x}$	σ	H_1	u	ρ
BQ1	2.33	1.03	2.50	1.05	$\neq$	16.0	0.737
BQ2	4.00	1.55	4.00	1.55	$\neq$	18.0	1.000
BQ3	1.67	1.21	2.17	1.32	$\neq$	14.5	0.530
BQ4	1.67	1.21	2.17	1.32	$\neq$	14.5	0.530
BQ5	4.17	0.75	4.33	0.82	$\neq$	15.5	0.665
BQ6	3.17	1.17	3.00	0.90	$\neq$	17.0	0.867
BQ7	4.33	1.63	4.17	1.71	$\neq$	13.5	0.390
BQ8	2.00	0.89	2.83	1.48	$\neq$	11.0	0.246
BQ9	3.00	1.55	3.83	1.48	$\neq$	11.0	0.206
BQ10	1.33	0.81	1.83	1.17	$\neq$	12.5	0.293
BQ11	4.50	1.22	4.50	1.22	$\neq$	18.0	1.00
BQ12	2.00	0.90	2.83	1.17	$\neq$	10.0	0.185
BQ13	1.17	0.40	1.33	0.52	$\neq$	15.0	0.523
BQ14	1.00	0.00	1.83	1.17	$\neq$	9.0	0.058
BQ15	1.33	0.51	2.00	1.10	$\neq$	12.0	0.279

BQ1. How many years of experience do you have as a software developer?
BQ2. I consider myself experienced with openAPI.
BQ3. I consider myself experienced with AsyncAPI.
BQ4. I consider myself experienced with AsyncAPI Specification.
BQ5. I consider myself experienced with Request-Response Models.
BQ6. I consider myself experienced with Event-Driven Architectures.
BQ7. I consider myself experienced with JSON.
BQ8. I consider myself experienced with Avro.
BQ9. I consider myself experienced with YAML.
BQ10. I consider myself experienced with CloudEvents.
BQ11. Until now, approximately, in how many projects have you worked on which have used JSON?
BQ12. Until now, approximately, in how many projects have you worked on which have used Avro?
BQ13. Until now, approximately, in how many projects have you worked on which have used CloudEvents?
BQ14. Until now, approximately, in how many projects have you created/updated an AsyncAPI Document?
BQ15. Until now, approximately, in how many projects have you used an AsyncAPI Document created by others to understand the functionality of an application?

Table 5.1: Summary of the answers to the Likert and numeric scale questions in the background questionnaire. Contains the mean and standard deviation for each group and the results of the MW-U test.

	CG		EG		MW-U		
	$\bar{x}$	σ	$\bar{x}$	σ	H_1	u	ρ
KP	2.50	1.05	2.67	1.03	$\neq$	16.0	0.737

KP. Number of known protocols.

Table 5.2: Summary of the number of known protocols by each participant checked in the background questionnaire. Contains the mean and standard deviation for each group and the results of the MW-U test.

	CG		EG		MW-U		
	$\bar{x}$	σ	$\bar{x}$	σ	H_1	u	ρ
UT	0.00	0.00	0.83	0.99	$\neq$	9.0	0.058

UT. Number of previously used tools.

Table 5.3: Summary of the number of previously used tools by each participant checked in the background questionnaire. Contains the mean and standard deviation for each group and the results of the MW-U test.

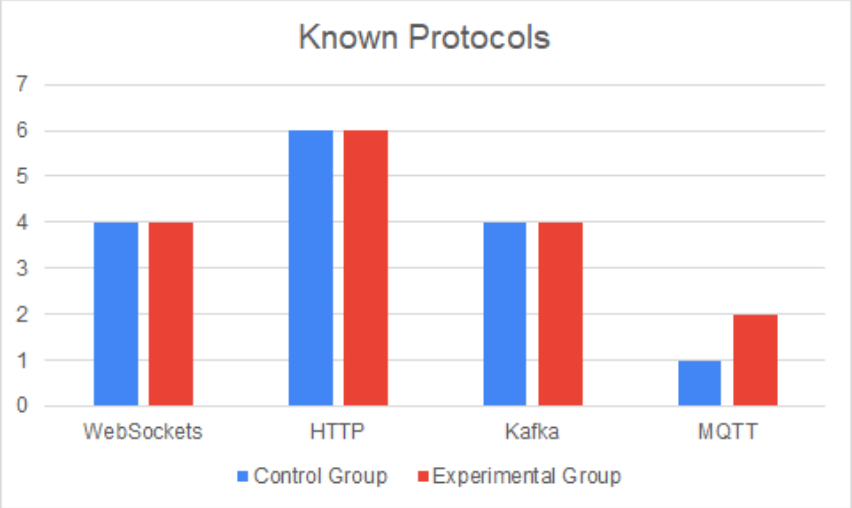


Figure 5.1: Known protocols of each group's participants.

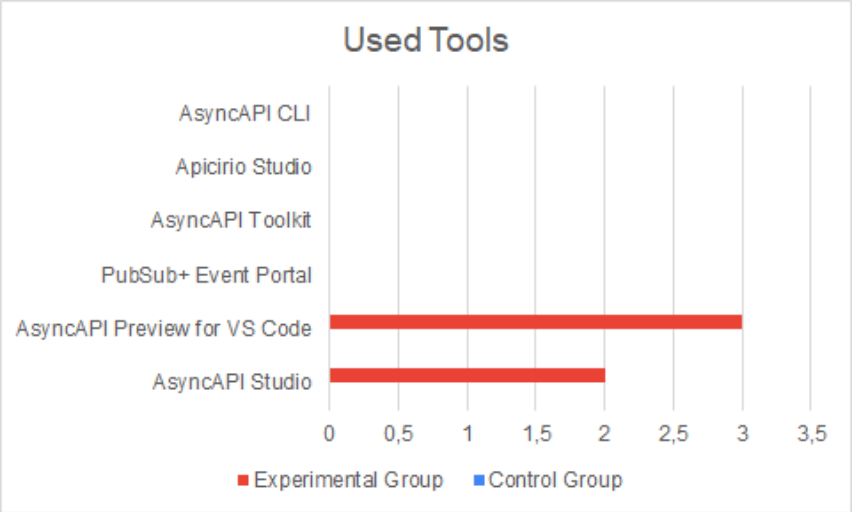


Figure 5.2: Used Tools of each group's participants.

5.4.2 Tasks

Considering the time marked by each participant at the beginning and end of each task, the goal of this analysis was to determine whether the **control** group would spend a bigger amount of time completing the tasks than the **experimental** group in order to answer the **Research Question 1**.

Figure 5.3 displays the distribution of time spent by each participant to complete each task, by group. It can be easily seen, that task 3 is the task that participants from both groups spent more time completing while other tasks seem more balanced.

Table 5.4 summarizes the results obtained for the time spent by each participant of each group along with the results of the MW-U test, the time is displayed in the format mm:ss. Considering the alternative hypothesis H_1 that states that the amount of time spent in the completion of each task by the participants of the **control** group was bigger than the amount of time spent by the **experimental** group (i.e. CG > EG), it is concluded based on the results that indeed, **completing Task 1 and Task3 was significantly faster** for the participants in the **experimental** group. Task 1 consisted of creating the "Info" component of the Document, however, the participants of the **experimental** group had to create folders with the structure presented in Figure 4.13 while the **control** group's participants did have to do it because the prototype already manages that. In Task 3 the participants had to create the "Channels" component and the schemas to use as a reference in it. These results support that the way the schemas were created and used as a reference in the prototype was sufficiently intuitive for participants to learn how to use it. The other tasks were similar to both groups.

Furthermore, Figure 5.4 and Table 5.5 display the results of the total time spent completing all the tasks, creating a complete AsyncAPI Document. Considering the alternative hypothesis H_1 that states that the amount of time spent in the creation of an AsyncAPI Document by the participants of the **control** group was bigger than the amount of time spent by the **experimental** group (i.e. CG > EG), it is concluded based on the results that indeed, **the creation of an AsyncAPI Document**

was significantly faster for the participants in the **experimental** group.

Besides that, the results of the validation of the AsyncAPI Document were analyzed. Figure 5.5 the distribution of the valid AsyncAPI Document created by each group and Table 5.6 displays the results of the McNemar tests and the percentage of participants that have created an invalid AsyncAPI Document. Considering the alternative hypothesis H_1 that states that the participants of the **control** group would create a larger number of invalid AsyncAPI Documents than the **experimental** group (i.e. CG > EG), it is concluded based on the results that **both groups created a similar number of invalid AsyncAPI Documents**. Taking account these results, we can conclude that the prototype wasn't successful in reducing the number of errors made while creating an AsyncAPI Document which answers to **Research Question 2**.

In summary, these results provide information to answer **Research Question 1** and **Research Question 2**. The prototype reduced the overall time spent in creating an AsyncAPI Document even though there wasn't a meaningful improvement for Task 2 and Task 4. On the other hand, results show that the developed prototype was able to reduce the number of errors made while creating an AsyncAPI Document.

	CG		EG		MW-U		
	$\bar{x}$	σ	$\bar{x}$	σ	H_1	u	ρ
T1	04:19	01:12	01:49	00:58	>	2.0	0.008
T2	02:30	01:31	01:30	00:32	>	10.5	0.203
T3	09:30	03:27	04:00	01:15	>	2.0	0.009
T4	06:00	04:38	03:00	00:53	>	8.0	0.103

T1. Time spent on the completion of Task 1.
T2. Time spent on the completion of Task 2.
T3. Time spent on the completion of Task 3.
T4. Time spent on the completion of Task 4.

Table 5.4: Summary of time spent by each participant to complete each task, by group. Contains the mean and standard deviation for each group and the results of the MW-U test.

	CG		EG		MW-U		
	$\bar{x}$	σ	$\bar{x}$	σ	H_1	u	ρ
TT	22:19	09:49	10:10	03:07	>	3.0	0.016

TT. Time spent on the completion of all the tasks.

Table 5.5: Summary of the total time spent completing all the tasks, creating a complete AsyncAPI Document, by group. Contains the mean and standard deviation for each group and the results of the MW-U test.

	CG	EG	McNemar	
	%	%	H_1	ρ
DE	16.7	0.0	>	0.063

DE. AsyncAPI Documents with errors.

Table 5.6: Summary of the results of the McNemar test applied to the validation of the final AsyncAPI Document, by group. Contains the percentage of subjects who have created an invalid AsyncAPI Document.

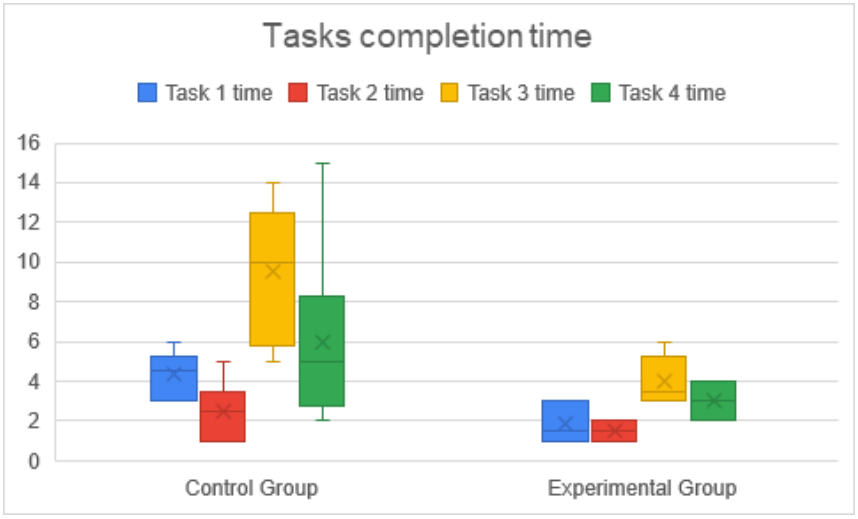


Figure 5.3: Distribution of time spent by each participant to complete each task, by group.

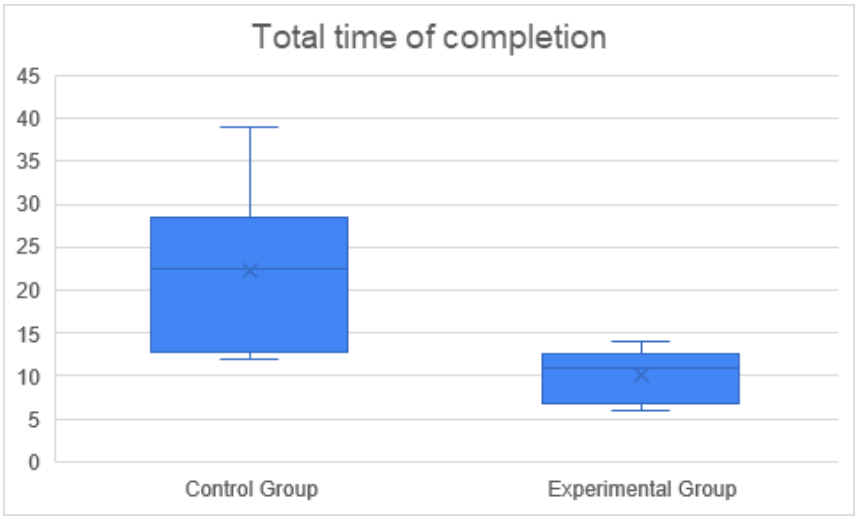


Figure 5.4: Distribution of the total time spent completing all the tasks, creating a complete AsyncAPI Document, by group.

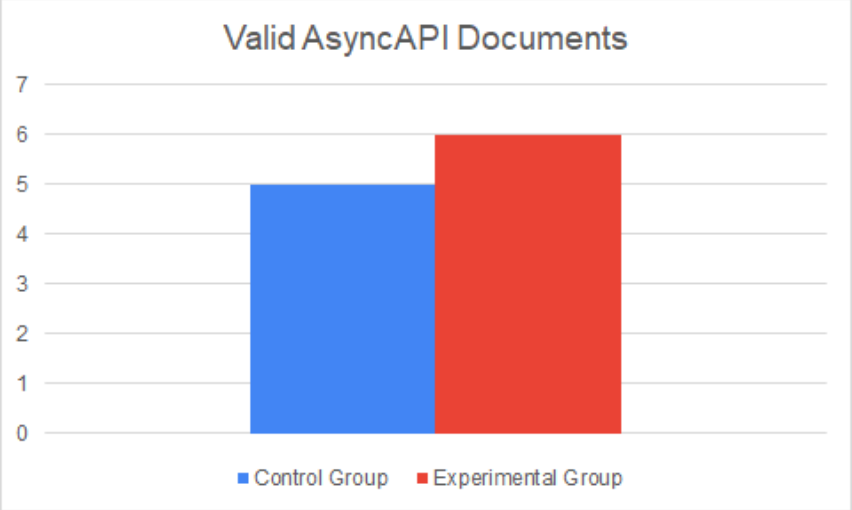


Figure 5.5: Distribution of the valid AsyncAPI Document created by each group.

5.4.3 Assessment Questionnaire

The assessment questionnaire featured a set of identical questions to evaluate possible environmental differences and task understandability along with questions to access PEoU (Perceived Ease of Use). MW-U tests were used to consider potential differences between both groups.

Figure 5.6 shows the distribution of answers to the Environmental question by each group and the first line of Table 5.7 displays its summary. Considering the alternative hypothesis H_1 that states that the participants of the **control** group had a different perception of the environment than the **experimental** group (i.e. CG $\neq$ EG), it is concluded based on the results that **there isn't a significant difference between both groups**.

A question was made to assess possible differences between the understandability of the provided tasks. Figure 5.7 shows the distribution of answers to the Task understandability question by each group and the second line of Table 5.7 displays its summary. Considering the results. it is concluded that **there is not a significant difference** regarding the tasks descriptions across both groups.

Lastly, two questions were made regarding the PEoU (Perceived Ease of Use). The last two lines of Table 5.7 display its summary of answers to the PEoU questions by each group, and Figure 5.8 and Figure 5.9 show the distribution of answers to PEoU questions by each group. Considering the alternative hypothesis H_1 that states that the participants of the **control** group would find the toolchain and the creation of an AsyncAPI Document more difficult than the **experimental** group (i.e. CG > EG), it is concluded based on the results that **there isn't a significant difference between both groups**.

Overall, the participants found their experience with both tools very similar.

	CG		EG		MW-U		
	$\bar{x}$	σ	$\bar{x}$	σ	H_1	u	ρ
ENV	1.33	0.52	1.17	0.41	$\neq$	15.0	0.523
TU	1.17	0.41	1.00	0.00	$\neq$	15.0	0.317
PEoU1	2.33	1.21	1.33	0.52	$>$	9.0	0.120
PEoU2	1.33	0.52	1.33	0.52	$>$	18.0	1.000

ENV. The environment was distracting.
TU. I found the task descriptions complex and difficult to follow.
PEoU1. Overall, I found the toolchain difficult to use.
PEoU2. I found it difficult to understand how an AsyncAPI Document is created.

Table 5.7: Summary of the answers to the assessment questionnaire, by each group. Contains the mean and standard deviation for each group and the results of the MW-U test.

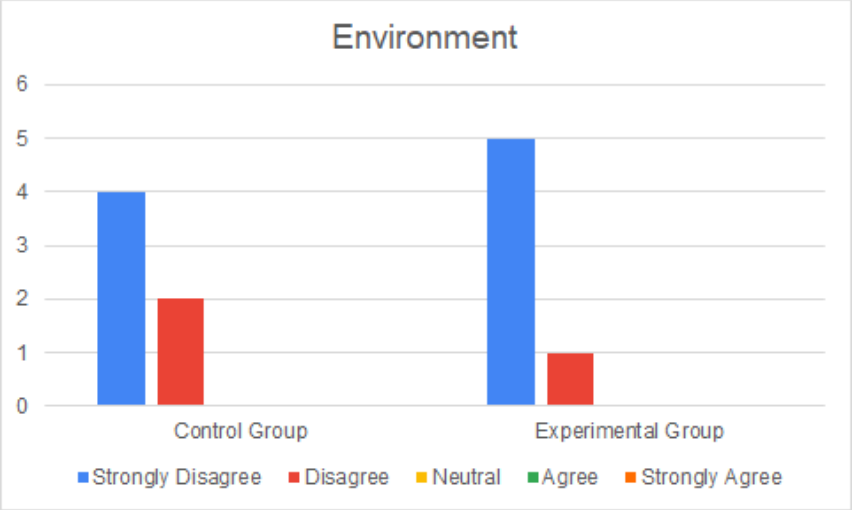


Figure 5.6: Distribution of answers to the Environment question by each group.

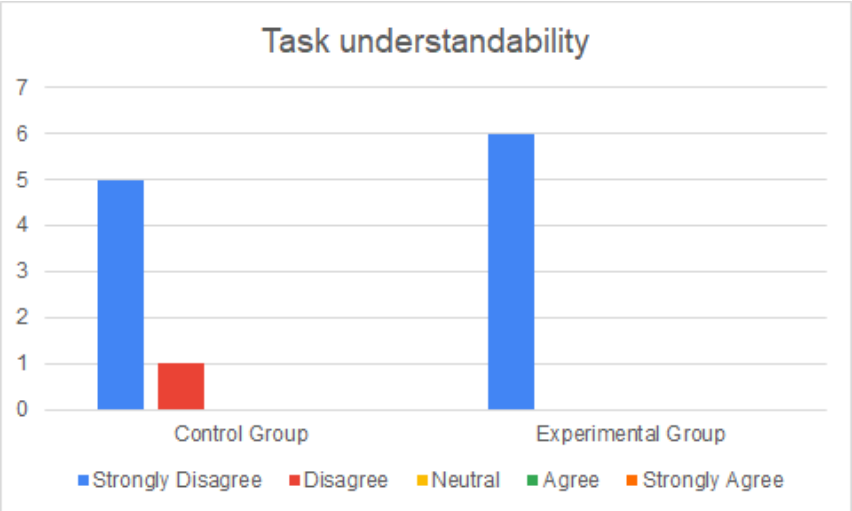


Figure 5.7: Distribution of answers to the Task Understandability question by each group.

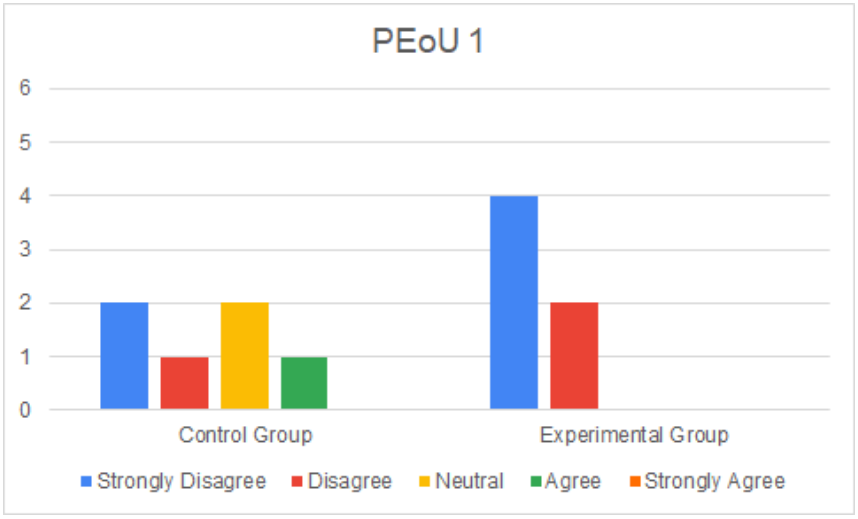


Figure 5.8: Distribution of answers to the first PEoU question by each group.

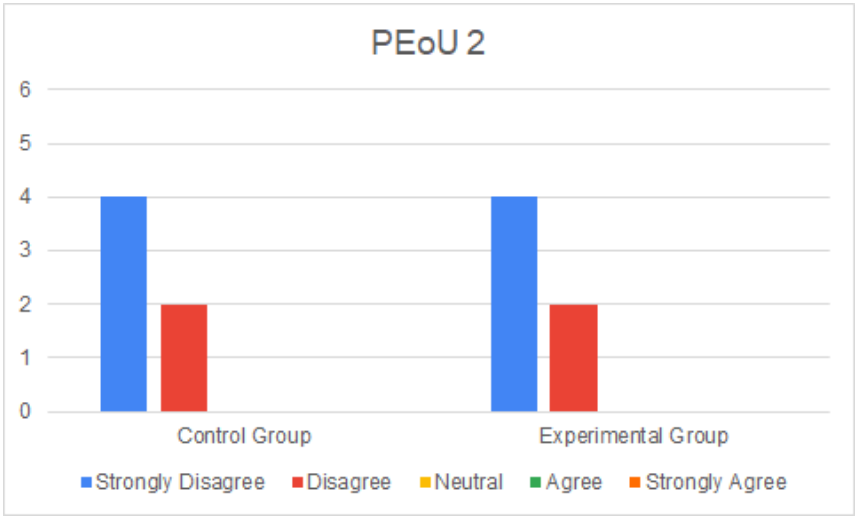


Figure 5.9: Distribution of answers to the second PEoU question by each group.

5.5 Validation Threats

This section identifies and describes the possible threats that might affect the experiment results. The validation threats are:

- **Small Sample Size -** Due to the lack of availability of some pre-recruited people, the number of participants recruited was low, which takes some confidence away from the results. To better confirm the findings it would be important to experiment with a larger sample size.
- **Psychological bias -** The group in which the participants are should be unknown to them to prevent any biased result. However, in some cases, this was impossible to achieve since some participants made the experiment in the same place and time. Nevertheless, efforts like avoiding verbal exchanges with themselves were made.
- **Environment influence -** Since some participants did the experiment remotely while others did it in person, the differences in the environment could cause deviations due to external factors. However, there were no problems in any of the environments, and the result of the assessment questionnaire supports that there was not any significant difference, so this threat can be discarded.
- **Time Registry Metrics -** The fact that the time was registered in the "hh:mm" format could have caused errors in some measurements. To prevent this, the time could have been measured in seconds.
- **Experience differences -** The results must be independent of skills and experience between the two groups. Because of this, the background questionnaire was an important part of the process, and its data analysis supports that the groups were balanced, so this threat can be discarded.
- **Sample bias -** The fact that most of the participants were students and didn't have much experience related to AsyncAPI may have caused possible bias in the results. Because of this, a more heterogeneous sample should be used in a further study.

5.5.1 Summary

In this chapter, the validity of the developed prototype was evaluated.

Firstly, three different scenarios were defined with three different tasks. Based on this, the process of completing those tasks was compared step-by-step using the developed prototype and using the tools provided by AsyncAPI. With this comparison, it was concluded that the prototype performed better when used in a task where the user needed to validate an invalid schema, and in the task, where the user had to create Documentation using a previously created Document as a base. However, in a task where the user had to add a new field to the Document, the prototype underperformed in comparison with the tools provided by AsyncAPI.

Secondly, the results and design of the conducted feasibility user study were described and discussed. To summarize, the findings support the hypothesis that the use of the developed prototype reduced the time spent in creating an AsyncAPI Document. However, the improvements were not reflected in all the steps of its creation. The creation of the "Channels" components and the creation of schemas to use as a reference was the step that showed the best improvement. While the time spent creating the "Servers" and "Operations" was very similar in both tools.

The findings related to percepetion-based metrics, didn't show any significant difference between the two tools since the participants of both groups felt that in general both tools were easy to use.

Even though the results seem promising, specially in terms of the prototypes performance in creating an AsyncAPI Document, additional research would be useful to confirm the findings. Testing the approach with a bigger and more diverse sample, with different levels of expertise is recommended to do it.

Chapter 6

Conclusions

Event-Driven Architectures changed the way how software is designed, developed and operated, offering robustness, scalability, and flexibility. Although there are already standards in the industry to define these types of architectures' applications as well as the structure of its messages and events, the field's novelty makes the tools to interact with these architectures low and with few features to help developers to define the application's functionalities.

The state of the art showed that the existent tools to create an AsyncAPI Document are not that much and some of them don't even work with the most recent version of the AsyncAPI Specification making the decision of using the tools provided by AsyncAPI almost mandatory. Even though these tools provided by AsyncAPI can generate complete valid Documents, they still lack of features that can improve the developer experience and performance.

In this thesis aimed to contribute to the overall book of knowledge of Event-Driven Architectures and to contribute to the improvement of workflow of Kuehne+Nagel developers. This lead to the development of an web application has as main features bootstrapping a specification file compliant with the AsyncAPI specification, using schema files as a reference when needed, from an easy and intuitive Graphical User Interface (GUI).

6.1 Hypothesis and Research Questions Revisited

This thesis has started with the following hypothesis:

H: A tool capable of creating an AsyncAPI Document with an intuitive Graphical User Interface reduces the time spent on creating an AsyncAPI Document and the error proneness.

It was then developed a prototype capable of creating AsyncAPI Documents based on the information that a user filled out in a form, validating schemas and using them as a reference in the Documents. The prototype was then used to validate the approach by comparing its usage to the usage of tools provided by AsyncAPI and by conducting a feasibility user study. It was concluded that the results generally support the hypothesis in terms of performance (reducing the

overall time spent creating an AsyncAPI Document), usability (the participants found the tool easy to use) and error proneness (none of the participants created an invalid AsyncAPI Document with the tool).
All the Research Questions can also be answered as follows:

RQ1 *Does the solution help developers by reducing the time spent creating an AsyncAPI Document?*

Has shown in Chapter 5, the solution does in did help developers reduce the total time spent in creating an AsyncAPI by improving the performance related to the use of schemas as reference in the "Channels" and the creation of a folder structure for all the files needed in the AsyncAPI Document before starting the creation of the Document.

RQ2 *Does the solution help developers by reducing the number of errors made while creating an AsyncAPI Document?*

The results in Chapter 5 shown that there isn't a significant different in the number of invalid AsyncAPI Documents created with the prototype and the tools provided by AsyncAPI. However, the number the percentage of errors made using both cases is very low it was not possible to prove that the app reduced the number of errors or not.

To summarize, the contributions of this thesis are as follows:

- **Comparison of the main features of the available tools used to create AsyncAPI Documents.** In chapter 2 its described and compared the main features of the available tools to create AsyncAPI Documents. This was used as a base to define the main features of the developed prototype.
- **An web application for creating AsyncAPI Documents.** A prototype to create AsyncAPI Documents was created in order to help developers reduce their time spent creating them and the number of errors in it as described in Chapter 4.
- **Validation by comparing the prototype and the tools provided by AsyncAPI and through a feasibility user study.** Firstly, three scenarios were defined to compare the performance of using the developed prototype and the tools provided by AsyncAPI in the completion of three tasks. Then, a feasibility user study was designed and conducted to empirically validate the developed prototype. All the validation process is described and documented in Chapter 5.

6.2 Future Work

This Section is dedicated to describing ideas for improvements which go beyond the immediate objective of this thesis and propose them as future work to further consolidate and complete the

work done. For this, two topics will be address: the developed prototype and the feasibility user study.

Prototype

Although the implemented features in the prototype, were enough to reach the previously defined goals to the application, some features could further improve the performance of the creation of an AsyncAPI Document and the overall user experience of the app. The feature ideas are as follows:

- **Text Editor.** As shown in Scenario 3 presented in Chapter 5, in some tasks having a way to write directly into the Document makes the developer work easier. In this context, having a text editor to directly edit the AsyncAPI Document would be a good feature to implement.
- **Operations reference List.** Like the reference list used to define the choose the path to the schema files, a list of channels and messages created in "Channels" component would be useful to have in the "Operations" component form.
- **Import Document.** Having the ability to import the data from an AsyncAPI Document into the app would be a useful feature to have in order to manage previously created Documents.
- **Version Conversion.** Having the feature to import a document and a feature to convert AsyncAPI Documents with older versions of the AsyncAPI Specification hand-to-hand would facilitate the managing of previously created Documents using the application.
- **Graph generation.** The generation of a graph showing the relationship between application would facilitate the understanding of the whole event-driven Architecture.

Feasibility User Study

As stated in the summary of Chapter 5, although the findings obtained in the conducted feasibility user study are promising, further research is needed to consolidate the results. For this purpose, is recommended to perform the study with a larger number of participants with a wider range of experience related with AsyncAPI Documents and tools. This would avoid the possible validation threats related with this results listed in Section 5.5.

References

[1] Axios, June 2024. Available at https://axios-http.com/.

[2] Ala Al-Fuqaha, Mohsen Guizani, Mehdi Mohammadi, Mohammed Aledhari, and Moussa Ayyash. Internet of things: A survey on enabling technologies, protocols, and applications. *IEEE Communications Surveys & Tutorials*, 17(4):2347–2376, 2015.

[3] Apache. Apache avro, January 2024. Available at https://avro.apache.org/.

[4] Apache. avro-js, June 2024. Available at https://www.npmjs.com/package/avro-js.

[5] Apicurio. Apicurio studio, January 2024. Available at https://www.apicur.io/studio/.

[6] AsyncAPI. Asyncapi, January 2024. Available at https://www.asyncapi.com/.

[7] AsyncAPI. Asyncapi bundler, June 2024. Available at https://github.com/asyncapi/bundler.

[8] AsyncAPI. Asyncapi cli, January 2024. Available at https://www.asyncapi.com/docs/tools/cli.

[9] AsyncAPI. asyncapi-parser, June 2024. Available at https://github.com/asyncapi/parser-js.

[10] AsyncAPI. asyncapi-react, June 2024. Available at https://github.com/asyncapi/asyncapi-react.

[11] AsyncAPI. Asyncapi studio, January 2024. Available at https://studio.asyncapi.com/.

[12] EclipseSource. Json forms, June 2024. Available at https://jsonforms.io/.

[13] Hamza Ed-douibi, Javier Luis Cánovas Izquierdo, and Jordi Cabot. Openapitouml: A tool to generate uml models from openapi definitions. In Tommi Mikkonen, Ralf Klamma, and Juan Hernández, editors, *Web Engineering*, pages 487–491, Cham, 2018. Springer International Publishing.

[14] D. Evans. The internet of things: How the next evolution of the internet is changing everything. pages 1–11. CISCO, 2011.

[15] OpenJS Foundation. Node.js, June 2024. Available at https://nodejs.org/en/about.

[16] Belategi L. et al. Gómez A., Iglesias-Urkia M. Model-driven development of asynchronous message-driven architectures with asyncapi. Springer International Publishing, 2021.

[17] Google. Angular, June 2024. Available at https://angular.dev/.

[18] IBM. Statistical package for the social sciences, June 2024. Available at https://www.ibm.com/spss.

[19] Docker Inc. Docker, June 2024. Available at https://www.docker.com/.

[20] N. Jazdi. Cyber physical systems in the context of industry 4.0. In *2014 IEEE International Conference on Automation, Quality and Testing, Robotics*, pages 1–4, 2014.

[21] Meta. React, June 2024. Available at https://react.dev/reference/react.

[22] Microsoft. monaco-editor, June 2024. Available at https://microsoft.github.io/monaco-editor/.

[23] Microsoft. Visual studio code, June 2024. Available at https://code.visualstudio.com/docs.

[24] Thomas Constantine Moore. react-code-block, June 2024. Available at https://react-code-blocks-rajinwonderland.vercel.app/.

[25] OpenAPI. Openapi, January 2024. Available at https://www.openapis.org/.

[26] Felipe Pezoa, Juan L Reutter, Fernando Suarez, Martín Ugarte, and Domagoj Vrgoč. Foundations of json schema. In *Proceedings of the 25th International Conference on World Wide Web*, pages 263–273. International World Wide Web Conferences Steering Committee, 2016.

[27] Bruno Manuel Nascimento Costa Galvinas Piedade. Visual programming language for orchestration with docker, 2020.

[28] Leonard Richardson and Sam Ruby. *Restful web services*. O'Reilly, first edition, 2007.

[29] Solace. Solace pubsub+ event portal, January 2024. Available at https://docs.solace.com/Cloud/Event-Portal/event-portal-lp.htm.

[30] StrongLoop, IBM, and other contributors. Express.js, June 2024. Available at https://expressjs.com/.

[31] CloudEvents Authors The Linux Foundation. Cloudevents, June 2024. Available at https://cloudevents.io/.

[32] Evan You. Vue.js, June 2024. Available at https://vuejs.org/.

Appendix A

Feasibility User Study Materials

This appendix contains the materials used in the feasibility user study, namely, the Google Forms provided to the control and experimental groups.

A.1 Control

Empirical Study in AsyncAPI Documents creation

Welcome and thank you for you availability to be part of this study. You will be asked to perform a set of task in order to create an AsyncAPI Document using Visual Studio Code.

1. How many years of experience do you have as a software developer?

   Marcar apenas uma oval.

   - ◯ < 1 year.
   - ◯ 1 to 3 years.
   - ◯ 3 to 5 years.
   - ◯ 5 to 8 years
   - ◯ 9+ years.

2. I consider myself experienced with **openAPI**.

   Marcar apenas uma oval.

   - ◯ Not Applicable
   - ◯ Strongly Disagree
   - ◯ Disagree
   - ◯ Neutral
   - ◯ Agree
   - ◯ Strongly Agree

3. I consider myself experienced with **AsyncAPI**.

   Marcar apenas uma oval.

   - ◯ Not Applicable
   - ◯ Strongly Disagree
   - ◯ Disagree
   - ◯ Neutral
   - ◯ Agree
   - ◯ Strongly Agree

4. I consider myself experienced with **AsyncAPI Specification**.

   Marcar apenas uma oval.

   - ◯ Not Applicable
   - ◯ Strongly Disagree
   - ◯ Disagree
   - ◯ Neutral
   - ◯ Agree
   - ◯ Strongly Agree

5. I consider myself experienced with **Request-Response models.**

   Marcar apenas uma oval.

   - ◯ Not Applicable
   - ◯ Strongly Disagree
   - ◯ Disagree
   - ◯ Neutral
   - ◯ Agree
   - ◯ Strongly Agree

6. I consider myself experienced with **Event-Driven Architectures**.

   Marcar apenas uma oval.

   - ◯ Not Applicable
   - ◯ Strongly Disagree
   - ◯ Disagree
   - ◯ Neutral
   - ◯ Agree
   - ◯ Strongly Agree

7. I consider myself experienced with...

Marcar apenas uma oval por linha.

	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree
...JSON.	◯	◯	◯	◯	◯
...Avro.	◯	◯	◯	◯	◯
...YAML.	◯	◯	◯	◯	◯
...CloudEvents.	◯	◯	◯	◯	◯

8. I consider myself experienced with...

Marque todas que se aplicam.

- ☐ WebSockets
- ☐ HTTP
- ☐ Kafka
- ☐ MQTT
- ☐ None

Until now, approximately in how many projects have you ...

9. ...worked on which have used JSON?

Marcar apenas uma oval.

- ◯ 0
- ◯ 1 to 3
- ◯ 3 to 5
- ◯ 5 to 8
- ◯ 9+

10. ...worked on which have used Avro?

Marcar apenas uma oval.

- 0
- 1 to 3
- 3 to 5
- 5 to 8
- 9+

11. ...worked on which have used CloudEvents?

Marcar apenas uma oval.

- 0
- 1 to 3
- 3 to 5
- 5 to 8
- 9+

12. ...created/updated an AsyncAPI Document?

Marcar apenas uma oval.

- 0
- 1 to 3
- 3 to 5
- 5 to 8
- 9+

13. ...used an AsyncAPI Document created by others (colleagues or third parties) to understand the functionality of an application?

Marcar apenas uma oval.

- ◯ 0
- ◯ 1 to 3
- ◯ 3 to 5
- ◯ 5 to 8
- ◯ 9+

14. Which tool(s) have you used to create/update AsyncAPI Documents?

Marque todas que se aplicam.

- ☐ AsyncAPI Studio
- ☐ AsyncAPI CLI
- ☐ PubSub+ Event Portal
- ☐ AsyncAPI Toolkit
- ☐ Apicurio Studio
- ☐ AsyncAPI Preview for VS Code
- ☐ Outro: _____

Tutorial - Explore an AsyncAPI Document

General Instructions

Read the following instructions very carefully.

To start the following tutorial you must have access to the following tools:

- Visual Studio Code .
- asyncapi-preview extension for Visual Studio Code .
- AsyncAPI_Example.zip .

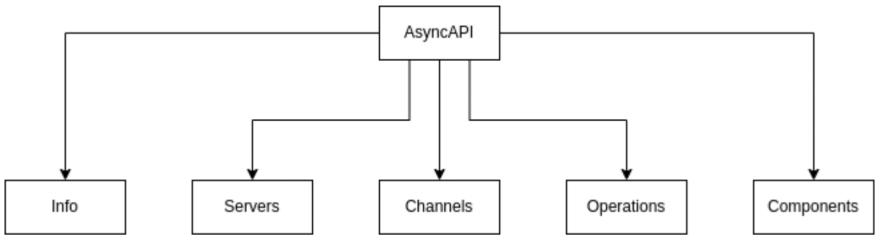
Once you're ready, advance to the next section.

Tutorial - What is an AsyncAPI Document?

An AsyncAPI document is composed of a set of fields defined by AsyncAPI Specification to describe an application's API. It is typically a single document that encapsulates the API description but can reference other files for additional details. In an event-driven system, it acts as a mutual agreement between the senders and the receivers since it specifies the required payload content that a producer sends and that a consumer receives.

This type of document has a specific and well-defined structure format that follows the AsyncAPI specification. It is composed of five main components, although it is not mandatory to use all of them, that provide information about the API's characteristics and behavior. The main components are:

- **Info field** - defines the API's metadata, e.g., title, version, description, contact, license, etc.
- **Servers field** - includes crucial information about the connection, e.g., host, port, protocol, etc. It facilitates connection in different environments like production, staging, and development.
- **Channels field** - provides a map of different channels used by the application to communicate. Each channel can be defined here by its address, expected message format of communication, the server where it is available, etc.
- **Operations field** - describes the different operations the application performs. Its main fields like action, channel, and message, offer a clear description of the purpose of each operation and whether the application receives or sends messages.
- **Components field** - is used for defining reusable structures or definitions. All the items defined in this field only become part of the API if referenced in other fields.



Tutorial - Explore an AsyncAPI Document

15. Follow along these next steps. Tick each step as you complete it.

Marque todas que se aplicam.

- [ ] Unzip the AsyncAPI_Example.zip.
- [ ] Open the folder with VS Code.
- [ ] Explore the schemas folder.
- [ ] Explore the traits folder.
- [ ] Explore the asyncapi folder.
- [ ] Open the AsyncAPIDocument.yml .
- [ ] Explore all the components of the document.
- [ ] Find where you can add traits.
- [ ] Find where you can add schemas..
- [ ] Find where you can add Examples.

If you've ticked all the boxes, advance to the next section.

Tasks

Task - 1

Read the following instructions very carefully.

In these tasks you will create an AsyncAPI Document for an Application of a smart LED that sends a message when it switches from turned off to turned on and vice-versa.

16. Start time

Exemplo: 08h30

17. Exercise - 1
    - Open an empty folder in VS Code.
    - Create a Folder called "asyncapi".
    - In this folder, create a folder called "examples".
    - In the root folder, create two more folders called "schemas" and "traits".

    Marque todas que se aplicam.

    ☐ Done.

18. Exercise - 2
    -In the "asyncapi" folder, create an YAML file with the name "SmartLEDapp.yml". This will be the application's AsyncAPI Document.
    - The application will use the version 3.0.0 of the AsyncAPI Specification.
    -Let's create the "Info" component. The tittle of the application is "My Smart LED".

    Marque todas que se aplicam.

    ☐ Done.

19. Exercise - 3
    - The version of our Document will be "1.0.0". And the description will be "Smart LED app that sends a message when it switches from turned off to turned on and vice-versa".

    Marque todas que se aplicam.

    ☐ Done.

20. Exercise - 4
    - The license of the application is the "Apache 2.0" licence with the url "https://www.apache.org/licenses/LICENSE-2.0.html".

    Marque todas que se aplicam.

    ☐ Done.

21. Finish time

Exemplo: 08h30

Task 2

The Servers Component.

22. Start time

Exemplo: 08h30

23. Exercise - 1
- Create the servers component.

Marque todas que se aplicam.

☐ Done.

24. Exercise - 2
- Let's name the server "myPrivateServer".

Marque todas que se aplicam.

☐ Done.

25. Exercise - 3
- In the host field put the address of the server "test.mykafkacluster.org:18092". Kafka will be the protocol.

Marque todas que se aplicam.

☐ Done.

26. Exercise - 4
- The description of the server is "Private server meant for monitoring things."

Marque todas que se aplicam.

☐ Done.

27. Finish time

_____

Exemplo: 08h30

Task 3

The channels component and the Schemas.

28. Start time

_____

Exemplo: 08h30

29. Exercise - 1
- In the channels component you will have to define the channels that the application will use and the messages that will be sent or received.
- Since we will need to define Schemas for the message payload and traits, create a new file in the "traits" folder called "SmartLED_traits.yml".

Marque todas que se aplicam.

☐ Done.

30. Exercise - 2
- In the file that you just created, define the traits of the app's message using a CloudEvents schema.
- This is the schema

Marque todas que se aplicam.

☐ Done.

31. Exercise - 3
- Go to the "schemas" folder and create a file called "SmartLED_message.avsc" to define the schema of the app's message payload with an avro schema.
- This is the schema

Marque todas que se aplicam.

☐ Done.

32. Exercise - 4
- Go to the "examples folder" and create a file called "SmartLED_example.json". This file will have an example of the payload of our message.
- Using the avro schema that you created in the "SmartLED_message.avsc" file as reference, create an example of the payload.

Marque todas que se aplicam.

☐ Done.

33. Exercise - 5
- Go to the AsyncAPI Document and create the channels component.
- The channel name is "SmartLEDChanged," and its address is "smart.led.changed.state". In the summary field, write "Channel used to monitor the change of state of my smart LED".

Marque todas que se aplicam.

☐ Done.

34. Exercise - 6
- Define the message sent by the app.
- The message name and title is "TurnOnOff".
-The summary of the message is "Message describing a change of the smart LED state".

Marque todas que se aplicam.

☐ Done.

35. Exercise - 7
- Since the payload Schema is in avro format, the schema Format is "application/vnd.apache.avro+json;version=1.9.0".
- In the Schema field, write the path to the schema you created in Exercise 3.

Marque todas que se aplicam.

☐ Done.

36. Exercise - 8
- In the traits field add a new trait with a reference to the path to the file you created in Exercise 2.

Marque todas que se aplicam.

☐ Done.

37. Exercise - 9
- In the examples, add a new example
- Use the example that
you created in Exercise 4 as a reference.
- The Summary is "Example message sent by the SmartLED".

Marque todas que se aplicam.

☐ Done.

38. Finish time

Exemplo: 08h30

Task 4

The Operations Component.

39. Start time

Exemplo: 08h30

40. Exercise - 1
- Create the operations component.
- Add an operation with the name "StateChanged".
- The action of the operation is "send".

Marque todas que se aplicam.

☐ Done

41. Exercise - 2
- Since you already defined the channel, you will use it as a reference in the channel field like "#/channels/SmartLEDChanged".

Marque todas que se aplicam.

☐ Done

42. Exercise - 3
- Since you already defined the message, you will use it as a reference in the channel field like "#/channels/SmartLEDChanged/messages/TurnOnOff".

Marque todas que se aplicam.

☐ Done

43. Exercise - 4
- The summary field is "This operation provides events about changes in the Smart LED state".
- The description field is "Every time the Smart LED has a change of state, an event is sent to the monitoring channel."

Marque todas que se aplicam.

☐ Done

44. Exercise - 5
- Define the default content type as "application/vnd.confluent.avro".

Marque todas que se aplicam.

- [ ] Done

45. Exercise - 6
- Let's Preview the Document. To do this, press **SHIFT+CTRL+P** to open command palette and select Preview AsyncAPI.

Marque todas que se aplicam.

- [ ] Done

46. Exercise - 7
- In the AsyncAPI Preview window you can see...

Marque todas que se aplicam.

- [ ] ... a blanc page.
- [ ] ... the details of the app's functionalities.

47. Finish time

_____

Exemplo: 08h30

Post-experience Questionnaire

You've completed all the tasks.
Please answer the following questions regarding your experience.

48. Mark the answers that best reflect your opinions.

Marcar apenas uma oval por linha.

	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree
The environment was distracting.					
I found the task descriptions complex and difficult to follow					
Overall, I found the toolchain difficult to use.					
I fount it difficult to understand how an AsyncAPI Document is created.					

49. Any comments about your experience?

A.2 Experimental

Empirical Study in AsyncAPI Documents creation

Welcome and thank you for you availability to be part of this study. You will be asked to perform a set of task in order to create an AsyncAPI Document using the AsyncAPI Toolbox.

1. How many years of experience do you have as a software developer?

   Marcar apenas uma oval.

   - ◯ < 1 year.
   - ◯ 1 to 3 years.
   - ◯ 3 to 5 years.
   - ◯ 5 to 8 years.
   - ◯ 9+ years.

2. I consider myself experienced with **openAPI**.

   Marcar apenas uma oval.

   - ◯ Not Applicable
   - ◯ Strongly Disagree
   - ◯ Disagree
   - ◯ Neutral
   - ◯ Agree
   - ◯ Strongly Agree

3. I consider myself experienced with **AsyncAPI**.

   Marcar apenas uma oval.

   - ◯ Not Applicable
   - ◯ Strongly Disagree
   - ◯ Disagree
   - ◯ Neutral
   - ◯ Agree
   - ◯ Strongly Agree

4. I consider myself experienced with **AsyncAPI Specification**.

   Marcar apenas uma oval.

   - ◯ Not Applicable
   - ◯ Strongly Disagree
   - ◯ Disagree
   - ◯ Neutral
   - ◯ Agree
   - ◯ Strongly Agree

5. I consider myself experienced with **Request-Response models.**

   Marcar apenas uma oval.

   - ◯ Not Applicable
   - ◯ Strongly Disagree
   - ◯ Disagree
   - ◯ Neutral
   - ◯ Agree
   - ◯ Strongly Agree

6. I consider myself experienced with **Event-Driven Architectures**.

   Marcar apenas uma oval.

   - ◯ Not Applicable
   - ◯ Strongly Disagree
   - ◯ Disagree
   - ◯ Neutral
   - ◯ Agree
   - ◯ Strongly Agree

7. I consider myself experienced with...

   Marcar apenas uma oval por linha.

	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree
...JSON.	◯	◯	◯	◯	◯
...Avro.	◯	◯	◯	◯	◯
...YAML.	◯	◯	◯	◯	◯
...CloudEvents.	◯	◯	◯	◯	◯

8. I consider myself experienced with...

   Marque todas que se aplicam.

   - ☐ WebSockets
   - ☐ HTTP
   - ☐ Kafka
   - ☐ MQTT
   - ☐ None

Until now, approximately in how many projects have you ...

9. ...worked on which have used JSON schemas?

Marcar apenas uma oval.

- 0
- 1 to 3
- 3 to 5
- 5 to 8
- 9+

10. ...worked on which have used Avro?

Marcar apenas uma oval.

- 1
- 1 to 3
- 3 to 5
- 5 to 8
- 9+

11. ...worked on which have used CloudEvents?

Marcar apenas uma oval.

- 0
- 1 to 3
- 3 to 5
- 5 to 8
- 9+

12. ...created/updated an AsyncAPI Document?

    Marcar apenas uma oval.

    - ◯ 0
    - ◯ 1 to 3
    - ◯ 3 to 5
    - ◯ 5 to 8
    - ◯ 9+

13. ...used an AsyncAPI Document created by others (colleagues or third parties) to understand the functionality of an application?

    Marcar apenas uma oval.

    - ◯ 0
    - ◯ 1 to 3
    - ◯ 3 to 5
    - ◯ 5 to 8
    - ◯ 9+

14. Which tool(s) have you used to create/update AsyncAPI Documents?

    Marque todas que se aplicam.

    - ☐ AsyncAPI Studio
    - ☐ AsyncAPI CLI
    - ☐ PubSub+ Event Portal
    - ☐ AsyncAPI Toolkit
    - ☐ Apicurio Studio
    - ☐ AsyncAPI Preview for VS Code
    - ☐ Outro: _____

Tutorial - Explore the AsyncAPI Toolbox

General Instructions
Read the following instructions very carefully.

To start the following tutorial you must have access to the following tools:

- A Web Browser of your choice.
- A connection to the AsyncAPI Toolbox.

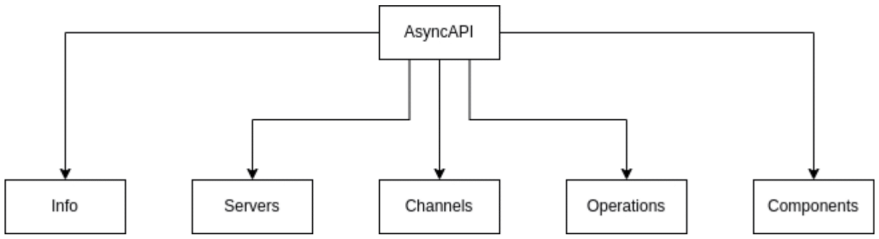
Once you're ready, advance to the next section.

Tutorial - What is an AsyncAPI Document?

An AsyncAPI document is composed of a set of fields defined by AsyncAPI Specification to describe an application's API. It is typically a single document that encapsulates the API description but can reference other files for additional details. In an event-driven system, it acts as a mutual agreement between the senders and the receivers since it specifies the required payload content that a producer sends and that a consumer receives.

This type of document has a specific and well-defined structure format that follows the AsyncAPI specification. It is composed of five main components, although it is not mandatory to use all of them, that provide information about the API's characteristics and behavior. The main components are:

- **Info field** - defines the API's metadata, e.g., title, version, description, contact, license, etc.
- **Servers field** - includes crucial information about the connection, e.g., host, port, protocol, etc. It facilitates connection in different environments like production, staging, and development.
- **Channels field** - provides a map of different channels used by the application to communicate. Each channel can be defined here by its address, expected message format of communication, the server where it is available, etc.
- **Operations field** - describes the different operations the application performs. Its main fields like action, channel, and message, offer a clear description of the purpose of each operation and whether the application receives or sends messages.
- **Components field** - is used for defining reusable structures or definitions. All the items defined in this field only become part of the API if referenced in other fields.



Tutorial - Explore the AsyncAPI Toolbox

15. Follow along these next steps. Tick each step as you complete it.

Marque todas que se aplicam.

- [ ] Enter the Application Dashboard.
- [ ] Find where you can add a new Application.
- [ ] Add a new Application with the Default Settings.
- [ ] Find where you can fill the fields of the "Info" component.
- [ ] Find where you can fill the fields of the "Servers" component.
- [ ] Find where you can fill the fields of the "Channels" component.
- [ ] Find where you can fill the fields of the "Operations" component.
- [ ] Find where you can fill other fields of the document.
- [ ] Enter the Schemas Dashboard.
- [ ] Find where you can add CloudEvents files.
- [ ] Find where you can add JSON files.
- [ ] Find where you can add Avro files.
- [ ] Find where you can add Examples.

If you've ticked all the boxes, advance to the next section.

Tasks

Task - 1
Read the following instructions very carefully.

In these tasks you will create an AsyncAPI Document for an Application of a smart LED that sends a message when it switches from turned off to turned on and vice-versa.

16. Start time

Exemplo: 08h30

17. Exercise - 1
Add a new app.

Marque todas que se aplicam.

- [ ] Done.

18. Exercise - 2
-Let's start for filling the "Info" component. First of all, let's name the application by filling up the title section with "My Smart LED".

Marque todas que se aplicam.

- [ ] Done.

19. Exercise - 2
- The version of our Document will be "1.0.0". And the description will be "Smart LED app that sends a message when it switches from turned off to turned on and vice-versa".

Marque todas que se aplicam.

- [ ] Done.

20. Exercise - 3
    - In the Licence section, use the "Apache 2.0" licence with the url "https://www.apache.org/licenses/LICENSE-2.0.html".

    Marque todas que se aplicam.

    - [ ] Done.

21. Finish time

    _____

    Exemplo: 08h30

Task 2

The Servers Component.

22. Start time

    _____

    Exemplo: 08h30

23. Exercise - 1
    - Open the Servers tab and add a new server.

    Marque todas que se aplicam.

    - [ ] Done.

24. Exercise - 2
    - Open the Servers tab.

    Marque todas que se aplicam.

    - [ ] Done.

25. Exercise - 3
    - Let's name the server "myPrivateServer".

    Marque todas que se aplicam.

    ☐ Done.

26. Exercise - 4
    - In the host field put the address of the server "test.mykafkacluster.org:18092". Kafka will be the protocol.

    Marque todas que se aplicam.

    ☐ Done.

27. Exercise - 5
    - The description of the server is "Private server meant for monitoring things."

    Marque todas que se aplicam.

    ☐ Done.

28. Finish time

    _____

    Exemplo: 08h30

Task 3

The channels component and the Schemas dashboard.

29. Start time

    _____

    Exemplo: 08h30

30. Exercise - 1
    - In the channels component you will have to define the channels that the application will use and the messages that will be sent or received.
    - Since we will need to define Schemas for the message payload and traits open the Schemas dashboard.

    Marque todas que se aplicam.

    ☐ Done.

31. Exercise - 2
    - Go to the CloudEvents tab and add a new schema with the name "SmartLED_traits" to define the traits of the app's message.
    - This is the schema

    Marque todas que se aplicam.

    ☐ Done.

32. Exercise - 3
    - Go to the Avro tab and add a new schema named "SmartLED_message" to define the schema of the app's message payload.
    - This is the schema

    Marque todas que se aplicam.

    ☐ Done.

33. Exercise - 4
- Open the schema that you just created and generate an example.
- With that example, go to the Examples tab and create a file named "SmartLED_example".

Marque todas que se aplicam.

☐ Done.

34. Exercise - 5
- Go to the Applications dashboard and to the Channels section of the app to add a new channel.
- The channel name is "SmartLEDChanged," and its address is "smart.led.changed.state". In the summary field, write "Channel used to monitor the change of state of my smart LED".

Marque todas que se aplicam.

☐ Done.

35. Exercise - 6
- Add a new message.
- The message name and title is "TurnOnOff".
-The summary of the message is "Message describing a change of the smart LED state".

Marque todas que se aplicam.

☐ Done.

36. Exercise - 7
    - Since the payload Schema is in avro format, the schema Format is "application/vnd.apache.avro+json;version=1.9.0".
    - In the Schema field, chose from the list the path to the schema you created in Exercise 3.

    Marque todas que se aplicam.

    ☐ Done.

37. Exercise - 8
    - Add a new trait.
    - Choose from the list the path to the schema you created in Exercise 2.

    Marque todas que se aplicam.

    ☐ Done.

38. Exercise - 9
    - Add a new example
    - Choose from the lists the example that
    you created in Exercise 4.
    - The Summary is "Example message sent by the SmartLED".

    Marque todas que se aplicam.

    ☐ Done.

39. Exercise - 8
- Save all.

Marque todas que se aplicam.

☐ Done.

40. Finish time

Exemplo: 08h30

Task 4

The Operations Component.

41. Start time

Exemplo: 08h30

42. Exercise - 1
- Go to the Operations tab.
- Add an operation with the name "StateChanged".
- The action of the operation is "send".

Marque todas que se aplicam.

☐ Done

43. Exercise - 2
- Since you already defined the channel, you will use it as a reference in the channel field like "#/channels/SmartLEDChanged".

Marque todas que se aplicam.

☐ Done

44. Exercise - 3
- Since you already defined the message, you will use it as a reference in the channel field like "#/channels/SmartLEDChanged/messages/TurnOnOff".

Marque todas que se aplicam.

☐ Done

45. Exercise - 4
- The summary field is "This operation provides events about changes in the Smart LED state".
- The description field is "Every time the Smart LED has a change of state, an event is sent to the monitoring channel."

Marque todas que se aplicam.

☐ Done

46. Exercise - 5
- Save all.
- Go to the Other Fields tab.
- Define the default content type as "application/vnd.confluent.avro".
- Save all.

Marque todas que se aplicam.

☐ Done

47. Exercise - 6
    - In the AsyncAPI Preview window you can see...

    Marque todas que se aplicam.

    - [ ] ... a blanc page.
    - [ ] ... the details of the app's functionalities.

48. Finish time

    _____

    Exemplo: 08h30

Post-experience Questionnaire

You've completed all the tasks.
Please answer the following questions regarding your experience.

49. Mark the answers that best reflect your opinions.

Marcar apenas uma oval por linha.

	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree
The environment was distracting.					
I found the task descriptions complex and difficult to follow					
Overall, I found the toolchain difficult to use.					
I fount it difficult to understand how an AsyncAPI Document is created.					

50. Any comments about your experience?