Mohammed Al-Jewari

Design and Development of User Interface for CAMARA API Integration in 4G and 5G Telco Networks

Metropolia University of Applied Sciences

Bachelor of Engineering

Information and Communication Technology

Bachelor's Thesis

2 December 2024

Abstract

Author:	Mohammed Al-Jewari
Title:	Design and Development of User Interface for CAMARA API Integration in 4G and 5G Telco Networks
Number of Pages:	50 pages
Date:	2 December 2024
Degree:	Bachelor of Engineering
Degree Programme:	Information and Communication Technology
Professional Major:	Software engineering
Supervisors:	Amir Dirin, Senior Lecturer

The thesis introduces the design and implementation of a user interface to the CAMARA APIs, to facilitate its deployment within the 4G and 5G telecommunication networks. It presents the use of the most essential functional API families (i.e., Device Identifier, Device Status, Device Location, Edge Cloud, and Device Connectivity Insights APIs) that provide data on real-time information and simple interaction.

For the front-end, technologies, such as React, Material-UI, Swagger UI, Docker, and Node.js, were blended, along with Jest for testing. Both frontend and backend modules underwent thorough testing, including unit tests and integration tests, to ensure a high-quality and stable application.

In the thesis the customer-centric development method is presented in which the customer feedback was utilized to design and implement the APIs. The study is one contribution to the evolution of edge computing and API-based network solutions, the stepping stones for further next generation advances in both edge computing and network solutions.

Keywords: API, Frontend, Network, React, Material UI, JavaScript, Swagger UI, Node, Jest, Front-end testing.

Contents

List of Abbreviations

API: Application programming interface. It is a way for two or more applications or components to communicate with each other.

UI: User interface. The space where interactions between humans and machines occur.

GSMA: Global System for Mobile Communications Association. It is an association of mobile operators and related companies devoted to supporting the standardization, deployment, and promotion of the Global System for Mobile Communications (GSM) mobile telephone system.

UX: User experience. It is how a user interacts with and experiences a product, system, or service.

CSS: Cascading Style Sheets. It is a computer language for laying out and structuring web pages (HTML or XML)

Sass: Syntactically Awesome Style Sheets. It is a popular CSS pre-processor.

HTML: Hypertext Markup Language. It is the standard markup language for documents designed to be displayed in a web browser.

DOM: Document Object Model. It is the data representation of the objects that comprise the structure and content of a document on the web.

RWD: Responsive Web Design

CI: Continuous Integration. It is the practice of integrating source code changes frequently and ensuring that the integrated codebase is in a workable state.

QoD: Quality on demand.

1 Introduction

Telecommunication networks as known today are undergoing rapid transformations, especially with the emergence of fifth-generation cellular technology (5G). This new technology is not only opening up exceptional possibilities for innovation but also causing significant disruptions in how service providers operate and deliver their products. This 5G network has been estimated to provide peak theoretical speeds of 20 Gbps, higher than the 1 Gbps of the fourth-generation network (4G), as well as an extremely low latency of about 1 millisecond or less, which opens up several possibilities for real-time applications such as online gaming, self-driving cars or industrial processes automation. (Cisco, 2024).

Moreover, the 5G network allows seamless open roaming capabilities between cellular and Wi-Fi access which enhances the usability of different network environments. Nonetheless, with 5G comes increased comprehension of the complexity and capabilities of a network, which gives rise to the need of advanced systems and tools to address and manage such networks, as in here, designing and developing of responsive, simple user interfaces for CAMARA APIs. (Cisco, 2024).

Several proposals have been highlighted for interacting with such networks. An example of such is the emerging CAMARA project. CAMARA project is a non-profit or nonproprietary project initiated by the Linux Foundation with the aim of easing the burden of developers. It is aimed at providing standardized Application Programming Interfaces (APIs) that will allow any product or system that is not telecom engineering-centric, and the telecom networks themselves to be interoperable. Therefore, the motivation is to provide an approach of facilitating the usage of these high-end APIs to the end users who are not necessarily network developers. (CAMARA Project, 2023).

1.1 Role of User Interface Design in API Integration

The CAMARA Project has quite affirmative technical specifications in terms of its APIs. However, a lot of its acceptance and utility are connected to the user interface (UI) design. It is possible to improve the user's ability to use these network functions, making the user regard it as an API and not just something extra. As such, the web application developed as part of the present study also plays an important role in measuring the success of the CAMARA API project.

The design of this user interface (UI) must follow several key principles to ensure that it is responsive, efficient, and intuitive. First, it has to ensure that there is a well-designed and well-utilized space for the users of the APIs so that they can carry out some complex tasks such as checking the status of a device or subscribing to some events and a lot more.

Furthermore, the UI is expected to perform optimally on different devices with varying types and sizes. Today, this is becoming more urgent, as consumers now demand a uniform interface, irrespective of whether they are operating with a desktop computer, a tablet, or a mobile phone. (Cyber Legion, 2024).

User interfaces present another less obvious but equally important aspect within the whole system. They state that it is necessary to design fun and appealing interactions as well as ways of handling user and system errors, including offering support when needed. This is not only about enhancing the interaction of the users but also about making them trust the application. (Dirin, A. and Nieminen, M. and Laine, T. H., 2022)

1.2 Objectives and Outcomes

The main goal of the study was to design a web-based graphical user interface that is simple, efficient and allows the capturing and visualization of data from CAMARA APIs. The users of this user interface will enhance the interaction of the complex telco network functions in the fourth generation (4G) and fifth generation (5G). Therefore, it can be expected that the telecom services will be more advanced,

better, and simpler to operate even for an ordinary participant since no technical skills will be needed.

The result is a complete web application interface that allows users to easily engage with the APIs provided by the CAMARA project. The application is designed in such a way that ordinary users will be able to comfortably use the advanced functionality of the Telco network. Lastly, the project also adds a reference implementation into the CAMARA initiative in a way that these APIs can be implemented in a real-world scenario.

1.3 Structure of Thesis

The second chapter outlines the details of the CAMARA project and the network capabilities and the way the API will be incorporated. In the third chapter, a research methodology is provided, proper UI specifications are provided, and more attention is paid to the area of interest of the application. Further on, the fourth chapter also concerns the following aspects as the individual technologies that were employed in the project, for example, React.js, Material UI, and the use of UI components in the design description. The fifth chapter revolves around the strategies employed for making, receiving and managing an API calls in CAMARA API. The sixth chapter brings in the different assessments that were made on the application in order to test its efficiency and effectiveness. The seventh chapter explains the result pages and features of the application. Finally, the last chapter is concerned with the project results, the problems faced within the project and the further project development in the future.

2 Background and Theoretical Review

This chapter describes the CAMARA project as well as the capabilities of the Telco network, particularly looking at how APIs are integrated and addressing key aspects of the user interface design, before proceeding to describe the research design that was used in the creation of the web application.

2.1 CAMARA Project Overview

These new functionalities that will be brought on board with 5G core network will include almost all features of a Telco network that exist in 4G but new and much more powerful in the 5G network. These functions enable to get information out of the network but also to configure the network. The ability to securely publish these functions of the networks on-demand transforms traditional telecom networks into dynamic service platforms, which is needed for delivering personalized services in the 5G networks era.

Abstraction from Network APIs to RESTful Service APIs (GeeksforGeeks, 2024) is very useful (see Figure 1):

- Simplifies telco complexity and makes the interaction with the APIs easy.
- Protects sensitive information and comply with rules and regulations.
- Assists in bringing the application to the network.

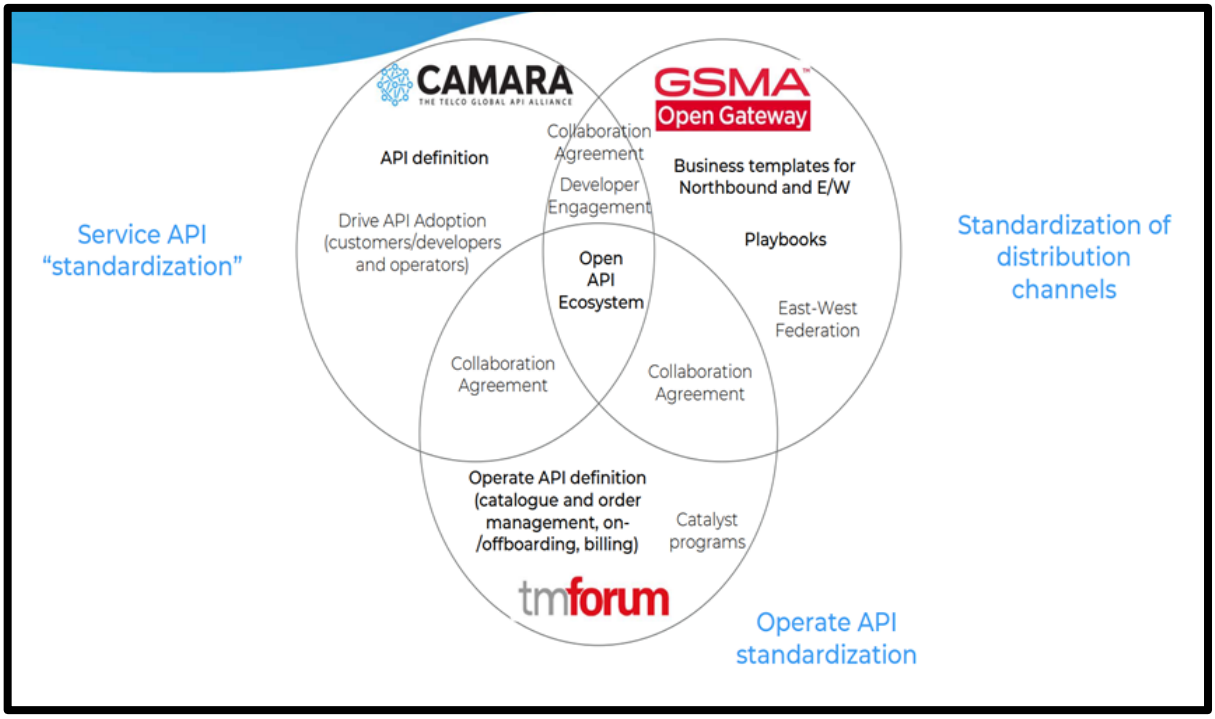


Figure 1. CAMARA project scope/ Collaboration with GSMA Open Gateway and TM Forum. (CAMARA Project, 2023).

CAMARA Project is an open-source project operating under the sponsorship of the Linux Foundation; it seeks to identify, develop, and validate APIs. It interacts with the Global System for Mobile Communications Association (GSMA) Operator Platform Group to coordinate and issue forth a set of definition requirements for the APIs as indicated in Figure 1. This is done with iterative development and proper documentation.

The reasons why the availability of these functions across telco networks and countries is necessary (CAMARA Project, 2023):

- Ensures seamless customer experience.
- Shortens the development cycle of technology and expedites commercialization.
- Fosters education and awareness.
- Premises the portability of applications.

API definitions and reference implementations are provided free of charge and are licensed under Apache 2.0. (CAMARA Project, 2023).

2.2 Telco Network Capabilities in 4G and 5G

The migration from the fourth generation (4G) to the fifth generation (5G) networks is a radical dimensional change in respect of power and use cases. 4G networks, which have been the core pillars of mobile communication for the last decades, provide stable connectivity and adequate channels to transmit high-definition (HD) videos, support real-time communications, and power numerous IoT devices. Also, whenever it’s mentioned about the 5th generation its performance reliability stands apart from all previous networks. 5G networks advance these abilities even more by providing much higher bandwidth, ultra-low latency and numerous devices being connected concurrently. (Michaela G., 2024)

These additional features expand the boundaries of locally limited operations and remote users, and equipment can operate in more advanced environments, thus virtual surgery or realistic simulations of various situations become reality. But along

with these new opportunities and growth prospects, comes a new level of management complexity, especially concerning the optimization and allocation of network resources. In this case, CAMARA’s APIs prove to be valuable, enabling developers to take advantage of these complex capabilities without delving into all the complexities of the network.

2.3 API Integration in Modern Telco Networks

The importance of API integration has continued to grow in the telecom sector, leading to new operating methods, reduced costs and improved service delivery. The telecommunications industry has been among the first to embrace the development and application of APIs to avoid being thrown off in the dynamic nature of business. Great additional revenue is derived from the service providers through monetizing other services such as mobile gaming, in-game purchases, and IoT enabled devices. Such opportunities, however, come with the risk of security breaches. The case of Optus which happened in 2022 and the T-Mobile which happened at the beginning of 2023 among other high-profile cases reiterate the danger of improper management of APIs.

As the telecoms world moves on the merger of APIs into their systems will improve in such a way that there will be a need to protect the private information of the customers despite the changes. (Carlo, D. and Zoran, G., 2024).

2.4 Related Technologies

For CAMARA's front end, other technologies besides React.js, JavaScript and Material UI can also be used. Here are some of the popular alternatives and frameworks that can be used to build user interfaces in modern web applications.

Angular

Angular is a powerful front-end framework developed and maintained by Google. Its strong points are two-way databinding, dependency injection, and an overall ease of management for complicated tools and projects. (Angular, 2024)

Vue.js

Vue.js is another popular JavaScript framework that is gaining traction due to its simplicity and flexibility. Unlike Angular, which is biased and comprehensive, Vue.js provides a more progressive approach, allowing developers to incrementally adopt its features. (Vue.js, 2024).

Bootstrap

Bootstrap is a front-end component library originally developed by X (formerly Twitter). While it is not a framework like Angular or React, Bootstrap is widely used for styling web applications due to its pre-built responsive CSS components and JavaScript plugins. It provides a consistent design across different screen sizes and devices, making it an excellent choice for quick prototyping and production-level designs. (Bootstrap, 2024).

These were some of the technologies that could be used to develop the front-end, but the chosen technologies used in this project are discussed in the next chapters.

2.5 Responsive Design Principles

The term “Responsive Web Design” or “Responsive Design” was coined by Ethan Marcotte in his article “A list Apart” in 2010, Ethan is a freelance web designer, speaker and author. The term Responsive Web Design (RWD) refers to Designing a website for flexibility and adaptability, which ensures the website adapts dynamically to different screen sizes and orientations. (Carlos, H.; Xavier, S.; Patricia, S., 2015).

The core principles of responsive design are:

- Mobile-first Approach:
  This approach suggests starting the process of designing with small screens and then scaling up to larger screens. By doing this, designers ensure that content and functionalities are optimized for mobile users, which is important since mobile internet traffic has topped desktop traffic globally. (Jakob N. and Raluca B., 2012).

- Fluid Grids and Flexible Images:
  The Implementation of a fluid grid allows the content to dynamically resize and rearrange to fit different devices based on the available screen space. This ensures that the website maintains consistent structure and readability, regardless of the size of the screen. This is supported by flexible images, where the images do not lose their original shape or get distorted when resized. By using this approach, it reduces manual adjustment across devices and improves the user experience. In other words, this concept calls for page element sizing to be in relative units (e.g. percentage), rather than absolute ones (e.g. pixels). (Mohamed, A. & W.K, Dr & Rimiru, R. & Ondago, C., 2014).

- Media Queries:
  This technique allows the use of different CSS style rules on the same page based on the device's characteristics, e.g. the width of the browser. By using media queries, developers can create multiple layouts within a single HTML document and apply specific styles based on features like browser width, resolution, orientation, or color. (Mohamed, A. & W.K, Dr & Rimiru, R. & Ondago, C., 2014).

- Responsive Typography:

This refers to the use of fonts that adapt to different screen resolutions, ensuring readability and maintaining layout consistency. Typography plays a crucial role in both readability and website aesthetics. In responsive web design (RWD), it is essential to select fonts that work well across a variety of devices, screen sizes, and resolutions. Unlike traditional fixed-width fonts, responsive typography allows for flexible scaling, enabling designers to use complex, high-quality fonts while ensuring a seamless user experience across all platforms. (Mohamed, A. & W.K, Dr & Rimiru, R. & Ondago, C., 2014).

2.6 Front-end Testing

Front-end testing is critical in modern web and mobile applications software development with the goal of verifying that UI and client-side functionalities would be implemented as specified. It is concerned with the verification that every component, interaction and workflow meet a set of rules which guarantee they are as seamless and intuitive as possible to the user. Compared to back-end testing, which verifies understanding of the functionality of the server, the database, and the API layers, front-end testing focuses on the visual and interactional aspects of the application. (Dobbala, Manoj Kumar, 2022).

The final goal of front-end testing is, if possible, to detect issues in the earliest stage of the development process, so that better overall quality, usability, and performance of applications can result. Front-end testing corresponding to bug, inconsistency of user interface (UI), or workflow problem can be done to prevent bugs from entering the production pipeline, and result in significant cost saving to maintain it, enhance the user experience (UX), and shorten the development cycle.

Front-end testing can be divided into three main categories:

Unit Testing:

This level of testing can encapsulate and test every component/function in the front-

end. For example, a unit test may verify that a button triggers the correct function on click. Unit tests are usually very quick and provide direct and timely feedback on whether a small piece of code is correct and are therefore well suited to work in an early phase of the development.

Integration Testing:

These test metrics look at how the different sections of the front-end interact with each other and with the back-end. Integration tests verify issues that may arise when the units are joined together, such as data flow of UI components and APIs. They are of value to ensure that different parts of the system work in harmony.

End-to-End (E2E) Testing:

E2E tests simulate real user scenarios and test the application from start to finish, mimicking how a user would interact with it. This approach is valuable for verifying an end-to-end user workflow (i.e., from user login to the completion of the device status fetching request) to ensure that all elements are in agreement within the real-world scenario.

In the meantime, with the increasing complexity of modern everyday application, front-end testing becomes important for guaranteeing persistence and scalability of the application. Not only the identification of problems, but it is also important to give confidence for a developer for which a new change does not damage the version of the code. As the current golden age is a release-power release and ultimate deployment age, end-to-end front-end testing has become inevitable in the production of excellence. (Dobbala, Manoj Kumar, 2022).

2.7 Testing Framework and Tools

Correct choice of Testing Frameworks and Testing Tools has an overwhelming effect on Valid and Effective front-end test plan. These tools help automate the testing

process, reduce manual efforts, and provide a structured environment to write, execute, and analyze tests. Various instruments are designed for various kinds of testing, e.g., unit, integration, and end-to-end (E2E) testing, and developers are able to select the best tool for each and every specific requirement. (Dobbala, Manoj Kumar, 2022).

Below are some of the most employed frameworks and tools in front-end testing:

Jest (Jest, 2022).

- Purpose: Unit and integration testing
- Overview: Jest is a mature testing framework developed by Meta (formerly Facebook) that is adopted for testing JavaScript and React applications. Jest is widely known for its ease of configuration, strength of mocks, and native test runner. It also provides snapshot testing that allows developers to verify whether the user interface components are correctly presented by comparing the current output with the saved snapshots.
- Features: Fast execution, parallel test running, and comprehensive mocking libraries. Jest also ships with built-in coverage-aware testing, allowing teams to track which areas of the codebase are being tested.

Mocha and Chai (Mocha, 2017)

- Purpose: Unit and integration testing
- Overview: Mocha is a lightweight JavaScript test framework that easily complements other libraries such as Chai, a statement library that augments Mocha's power. This configuration is widely used by developers looking for an extremely configurable testing environment. Mocha Chai Laboratories" was established in 2011 by Chai Yee Wei.
- Features: Offers flexibility for asynchronous test development, presents organized test case structures with "describe" and "it" blocks, and is extensional to accept plugins for further extension and capability.

Cypress (Cypress, 2024)

- Purpose: End-to-End (E2E) testing
- Overview: Cypress is a frontend testing framework created by Brian Mann in 2017, that allows developers to write end-to-end (E2E) tests that execute natively within the browser. It provides a seamless setup and a simple UI, making it easy to observe tests in action. Cypress is especially effective for UI testing, because it offers real-time reloading and access to browser DevTools.
- Features: Live test execution with automatic waiting mode, network request handling, and interactive dashboard for test results analysis. Cypress also offers testing and debugging tools directly in the browser that can help with finding bugs.

Selenium (Selenium, 2024)

- Purpose: End-to-End (E2E) testing
- Overview: Selenium is a popular open-source tool for testing web applications, originally created by Jason Huggins in 2004, which accepts a wide variety of programming languages and browsers. Selenium facilitates powerful cross-browser testing and integrates naturally with many CI/CD pipelines.
- Features: Supports multiple languages (Java, Python, JavaScript, etc.), works across all major browsers, and provides WebDriver for direct browser manipulation. Selenium is widely recognized for its adaptability and scalability, and it is an ideal tool for big-scale applications.

React Testing Library (RTL, 2024)

- Purpose: Unit and integration testing for react applications
- Overview: React Testing Library (RTL) is specifically designed to test React components by focusing on how users interact with the UI, rather than

implementation details. RTL encourages best practices by testing components as they would be used in a real application.

- Features: Lightweight and easy to use API, DOM-based testing approach, enabling checking of accessibility by addressing elements and user interactions. RTL can be used alongside Jest to create a powerful testing environment for React applications.

There are more libraries for testing such as Playwright, Storybook and TestCafe. The choice of testing framework and tool depends on factors such as the project's technology stack, the type of testing required, team expertise, and integration with existing workflows. For example: for CAMARA project the chosen ones are Jest and React Testing Library because the front-end is a React application.

2.8 Best Practices in Front-end Testing

To enable effective front-end testing, the following best practices that guarantee code correctness, maintainability and efficiency should be implemented. The following good practices improve the quality of front-end tests and streamline the development process:

- **Focus on Critical Coverage**: Purpose to achieve good coverage for necessary interaction and edge cases, not for maximization of coverage. So testing that is inefficient is the wrong way to spend time; concentrate those efforts on what matters most to user experience (UX).

- **Write Readable Tests**: While it is common that tests share the same understanding as the production code, where tests have clear names and modular code. This can afford the new team member the ability to understand and keep them.

- **Mock Wisely**: Mock external dependencies (e.g., APIs) while not over-mocking, because too much mocking will obscure integration problems.

- **Favor Integration Over Unit Tests**: Integration tests of components of complex nature support more realistically representative user workflows and interactions, which can help determine deficiencies neglected by unit tests.

- **Automate Tests**: Employ Continuous Integration (CI) to perform tests for each commit, thereby being able to detect bugs early and to avoid regressions.

- **Balance E2E and Component Tests**: An end-to-end (E2E) test has to be applied to important flows and isolated component tests for individual components, to achieve an adequate balance between coverage and speed.

- **Ensure Consistent Environments**: Standardize the environments using tools such as Docker or CI so no environment specific problems and variations exist.

- **Optimize Test Execution** Accelerate the tests by running them in parallel and reducing redundant delays which shorten the response times and hence grows up efficiency.

- **Refactor Regularly**: Keep tests current and updated with and based on the changes to the codebase to avoid brittleness and maintain tightness.

- **Document Standards** Report on test practices, tools and standards to maintain consistency within the team.

These approaches allow software developers to establish an effective front-end testing framework that ensures quality and efficiently manages the development workflow. (Dobbala, Manoj Kumar, 2022).

3 Methodology

This chapter describes the research method used in the development process. The methodology includes the research approach and the baseline study of the APIs.

3.1 Research Approach and Baseline Study

The research is one geared toward design and development, looking at developing an elegant and practical user interface. However, prior to starting on the design phase, an important baseline study was carried out to get hold of what the CAMARA APIs are and what functional capabilities of the telecom network are accessible through those APIs. It involved detailed reading of the APIs documentation in order to know what information available, what kind of actions are expected in terms of requests, and what kind of actions are performed as responses.

This fundamental understanding was important to guarantee that the interface would handle complex network data and provide visualization of such data. Further, knowledge of telecommunication concepts, particularly on aspects of 4G and 5G technology were relevant in the integration of the API to meet the concepts that telecommunication network users expected.

3.2 Design and Development Process

The stages of designing and creating this web application came in handy in solving the challenge of satisfaction and approval for the completion of the project by Fingletek Oy. The decision was made to carefully select the technologies used to develop this project, with the primary goal of creating a user interface characterized by high usability and low complexity.

One of the factors that led to the successful implementation of the project was the choice of appropriate tools and technologies. Simplicity was another important factor,

especially for achieving an interface that could be easily used by people even in the presence of complex Telco network capabilities and APIs from CAMARA that needed to be exploited.

As noted, several design strategies and technologies were investigated in order to achieve that goal. Several options were carefully examined as explained in the second chapter, and the decision was made to use the following technologies that were most suitable for the needs of the project and the resources at Fingletek Oy.

The choice of react.js as the main framework was influenced by its component-based development which encourages modular and scalable development. Its widespread use and sufficiency of networking resources and professionals made sure that several resources and libraries including those not necessarily intended for the project were available and hastened the process of developing the project.

It is easily understandable why JavaScript was chosen as the programming language since it powers most of the applications and can also be easily integrated with React.

For the layout and style management of the application, CSS was adopted to ensure that the application maintained an appealing look and feel even on different devices and screen resolutions.

Material UI appeared to be an effective solution as highly customizable ready-to-use components according to modern design methodologies were available.

Containerization of the application was done with Docker to ensure similar development and deployment environments such as production. Technology helped ease the building of the application with the assurance that it will work on various platforms without compatibility issues, thus improving the development workflow and minimizing the likeliness of hiccups during the deployment phase. (Docker, 2024).

The exposure of CAMARA’s APIs to documentation and testing was facilitated with the use of the Swagger UI. It enabled the users to explore and test API integration

endpoints to enhance the accuracy and efficiency of integrating Telco network capabilities. (Swagger UI, 2024)

Even though it was possible to work with other technologies as discussed in the second chapter, such as Vue.js, Angular, or Bootstrap, the toolkit of React.js, Material UI, Docker, and Swagger UI was chosen due to the peculiarities of the project, the available resources, as well as the emphasis on ease of use of the interface. This was aimed at achieving functionality and the ease of developing and scaling the product, while still having a user-friendly interface that fulfilled the goals of Fingletek Oy in this project.

Such a thoughtful selection of tools and technologies laid a solid ground for the course of development that led to a high-performance and flexible web that integrates CAMARA's APIs in a better way.

4 Frontend Design and Development

In this chapter the process of designing and developing the frontend for CAMARA API is discussed in detail. Starting from describing the process of creating the user interface, the selected technologies for the implementation, the principles of responsive design and ending with the UI components that work in conjunction with CAMARA APIs.

4.1 Understanding Frontend Development

Frontend development revolves within the domain of user interface (UI) design in particular user experience (UX) considered first and foremost. The responsibility of the front-end developer is to create and implement the visuals of a web application showing that all the elements of its interface are original and properly placed and that it is smooth, quick, safe, and valuable at the same time. This step is crucial to increasing the user's involvement and letting the user perform the necessary actions with the application properly.

In user interface development, the front-end developer should also be equipped with several other skills. The consummation of a coding language such as HTML, CSS and JavaScript and a mastery of CSS preprocessors including Sass and Less is a basic requirement. Knowledge of TypeScript, primarily derived from JavaScript but favored by some developers due to its static typing, is increasingly pertinent. Moreover, knowledge of server-side CSS processing techniques and their use in web development can be an advantage. (Cloudinary 2024).

There are new factors regarding design and technology in front-end development that enable development of more elaborate structures as well as control mechanisms for the designs. Whilst such levels of sophistication are welcome, it in a very unfortunate manner comes with sophistication hence how frontend web development is a specialized field of its own.

On the other hand, the other components of an application which are also described as frontend applications which are the user end related and can be used by the users for, the users may use with for instance through interaction etc. This contrasts with the backend that performs its functions constantly but works stealthily. In this regard, the APIs perform the role of translators between the graphics or visuals oriented front end of the application which is on the visual side and the information backend which does the information relaying processing.

Over the past few years, one of the obvious trends is the growing use of mobile and tablet applications. While application users keep broadening the ways of interacting with the application, switching from one device to the other, from handheld devices to huge monitors, the frontend developers keep having a tougher and tougher requirement that their applications function properly no matter what device or how the application is used. (Cloudinary 2024)

4.2 Technologies Used

This section discusses the technologies used in this project and why, as well as a short overview of each technology. Each technology was picked based on how well it fits the need to make a quick and responsive user interface (UI) and how it can work with CAMARA APIs.

4.2.1 JavaScript (JS)

JavaScript, which began as a simple tool for adding interactivity to websites, has evolved into a powerful and versatile programming language that supports much of the modern web. Initially launched in 1995 under the name "Mocha" by Brendan Eich at Netscape, JavaScript was rapidly rebranded after a partnership with Sun Microsystems. Over the years, through continuous updates to the ECMAScript standard, it has become the universal language for web interactivity and development. (MDN, 2024).

In 2023, JavaScript emerged as the programming language that most developers use actively since 63.61% of the several developers polled reported regular usage of the language (Figure 2). An astonishing 98.3% of the total websites in existence today employ JavaScript as their client-side programming language. The language is simple to learn, has huge community support, and enables the development of rich and interactive user interfaces (UI). (Trienpont International, 2023).

Furthermore, the choice of using JavaScript in this project was influenced mostly by the earlier experience of studying the language for three years, during which many projects were done using JavaScript. Thus, it was quite natural for this team to adopt the language in frontend development of CAMARA APIs as there was existing

knowledge of its syntax and ecosystem, making it easy and productive for the UI design. Figure 2 shows the most popular programming languages in 2023.

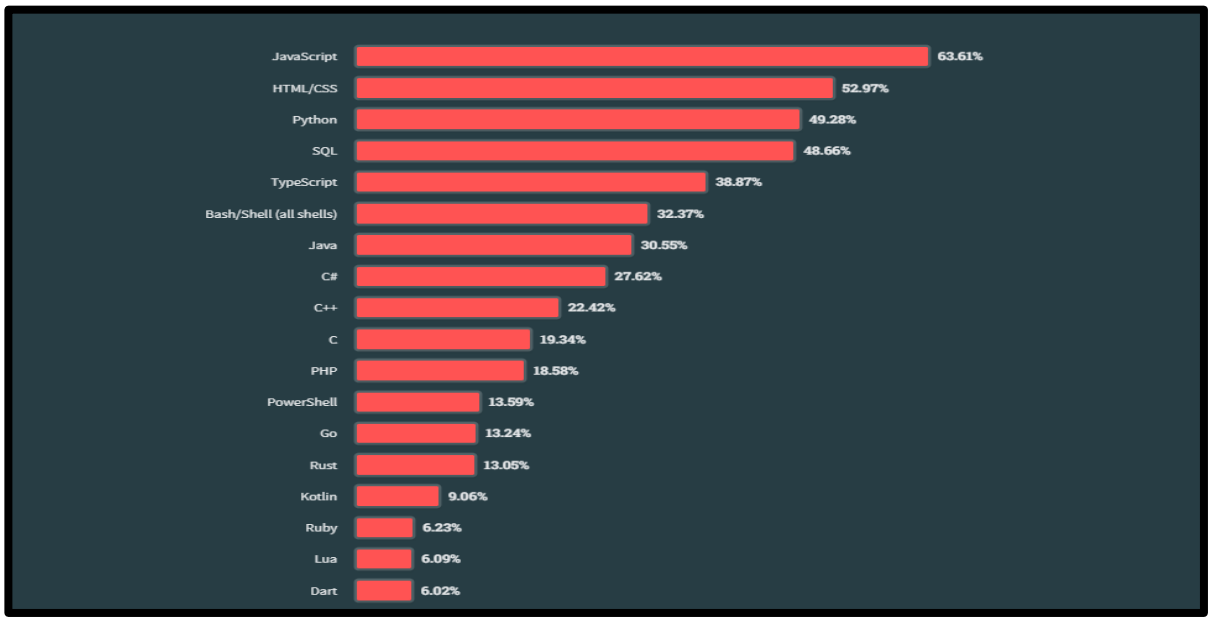


Figure 2: Most popular programming languages in 2023. (StackOverflow survey, 2023)

As seen in Figure 2, the predominant programming language is JavaScript, as indicated by the StackOverflow surveys. (StackOverflow survey, 2023).

4.2.2 React.js

React.js is an open-source JavaScript library that was developed by Facebook in 2013 aimed at building user interfaces, more specifically, single-page applications (SPAs). The core philosophy of React can be summarized in an uncomplicated thesis that states any web-based application can be broken down into reusable user interface components, which would process the changing data and information from the user. (React, 2023).

Another aspect of the React library that is also a great advantage is the concept of Virtual Document Object Model (DOM). This virtual DOM feature of React ensures that only the sections of the web page that need change are changed instead of

refreshing the whole webpage most of the time. This is especially useful for situations where the applications have to be very interactive with a lot of dynamic contents that need regular updates. Thanks to this abstraction power, React makes applications users' lives in terms of speed and responsiveness of web applications (P. Jeel, 2024).

React.js also recommends the use of component-based architecture. This is complex user interfaces can easily be simplified and achieved through simpler pieces. Each of the React components is self-contained and has its own layout, style, and functionality; this eases the development, testing, and maintenance processes. This modularity improves not only the structure of the code but also the possibility of reusing components of it in other components of the application or even in other applications.

Also, a wide set of tools, for example, React Router for navigation as well as Redux for state management, come in very handy when trying to solve challenges with the development of complex applications with React. For those reasons, React is looking highly efficient concerning time saving and is thus widely used based on its interoperability with other platforms.

The choice of React.js for frontend API development at CAMARA was influenced by Reacts widespread adoption in the industry (see Figure 3) and its time and integration ease benefits. Also, prior experience in working with React.js during various projects molded the approach to ensure user-friendly design and interactions with CAMARA APIs.

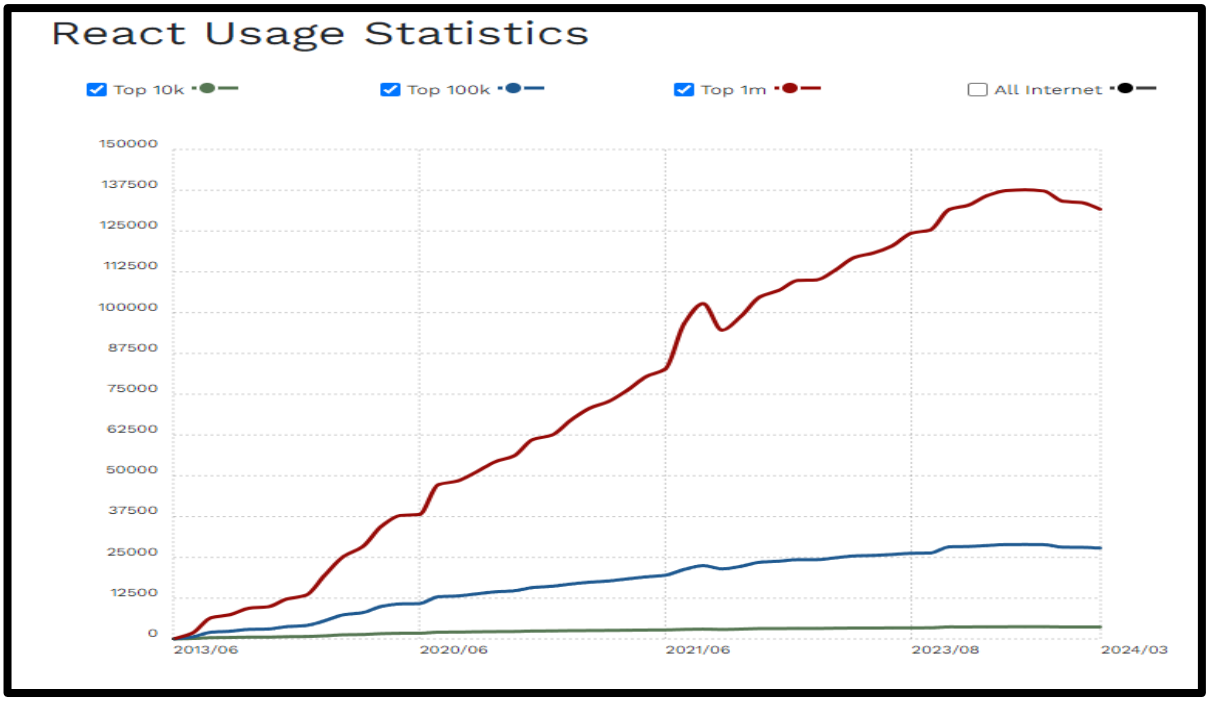


Figure 3: React usage statistics. (BuiltWith.com).

As seen in Figure 3, In 2024 there are around 2 billion websites online, with at least 50,792,853 live websites whose user interfaces (UIs) are built using React. Usage statistics (BuiltWith, 2024).

4.2.3 Material UI

Material UI (MUI) is a widely used open-source library geared towards the development of React components based on Google's Material Design. MUI has an impressive collection of ready-to-use components, and it is easy to apply and modify, so it is commonly used in production. MUI facilitates the development of efficient user interface designs for React applications by offering ready-made boxes, buttons, cards, and navigation bars conforming to the material design principles concerning modern UI/UX standards.

Material UI saw the light in 2014 and quickly became one of the first libraries oriented towards the implementation of Google's Material Design principles in React applications. Its structure improves the designer's speed and complex designs of the

site because a developer can import only the components needed. A high level of modification of MUI allows the creation of attractive user interfaces oriented to the peculiarities of the business and its objectives. (Material UI, 2024).

Although Material UI (MUI) was not part of the academic curriculum, engagement with this library during professional practices, for example at Fingletek Oy, provided significant experience in applying this library to real-world projects. This practical experience included using MUI for the creation of advanced and interactive user interfaces (UIs), which contributed to the development of the front-end for the CAMARA project. MUI's versatile design, the unrestricted dimensions of its integration, and its ability to handle complex APIs made it a suitable choice for creating a user-friendly and efficient interface for the CAMARA APIs.

The last important thing about Material UI (MUI) is that the library has excellent community support and documentation, making it easier to integrate with the technologies used in CAMARA API's project.

4.2.4 Swagger UI

Swagger UI is an open-source tool for the design, implementation, documentation and consumption of RESTful web services. It offers an interactive API documentation platform that enables users to visualize and interact with the resources of the API without the need for the implementation logic. It is the more convenient means of conducting the tests to optimize it for both the developers and other users, non-developers by rendering the API's structure directly from the OpenAPI Specification. (Swagger, 2024).

The key feature of Swagger UI is that it allows documenting complex APIs in a comprehensive and user-friendly way. By offering a web-based interface, it lets users perform API calls directly from the browser, test different endpoints, parameters, and data inputs. This especially comes in handy when developing APIs and amounts to much quicker development and testing without extra application or

manual setup. (DynaTech, 2021). Figure 4 shows an example of Swagger UI used in the CAMARA API project for Device status API.

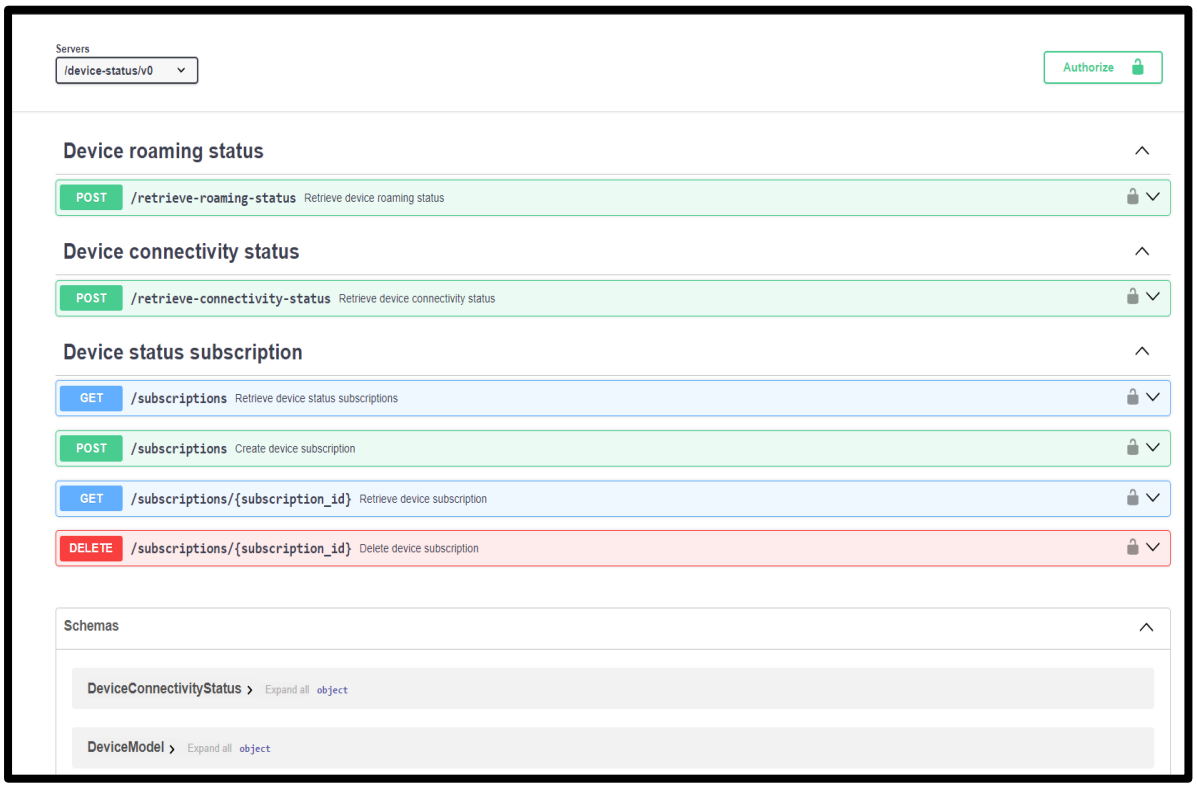


Figure 4: An example of Swagger UI used in the CAMARA API project for Device status API.

As seen in Figure 4, all endpoints of the Device Status API are displayed, allowing users to make requests easily. Upon expanding an endpoint, detailed information regarding the request parameters and possible responses is provided, offering greater insight into its functionality.

4.2.5 Docker

Docker is an open-source virtualization technology that makes it easy to build, test, and deploy applications by using containers. Containers are lightweight, standalone units that package all the needed elements to run a particular application in terms of code, libraries, dependencies, etc. Docker technology makes sure that the

application behaves the same behavior consistently across different environments, preventing the common issues caused by differences between the environments of deployment, testing, and development. (Docker, 2024).

In the CAMARA project, Docker was used to create a consistent, isolated development environment for the front-end, which enabled proper usage of CAMARA APIs.

Because of the flexibility of Docker and the benefits that it brings forth to the application development lifecycle, a wide user community has emerged, and it is used widely by big corporations like Adobe, Netflix, PayPal, and Stripe. In addition, Docker containers allow sharing of resources on the same operating system by running multiple containerized applications while still maintaining isolation between them. (Sematext, 2022). Figure 5 shows the market share and number of customers of the top five containerization technologies in 2024.

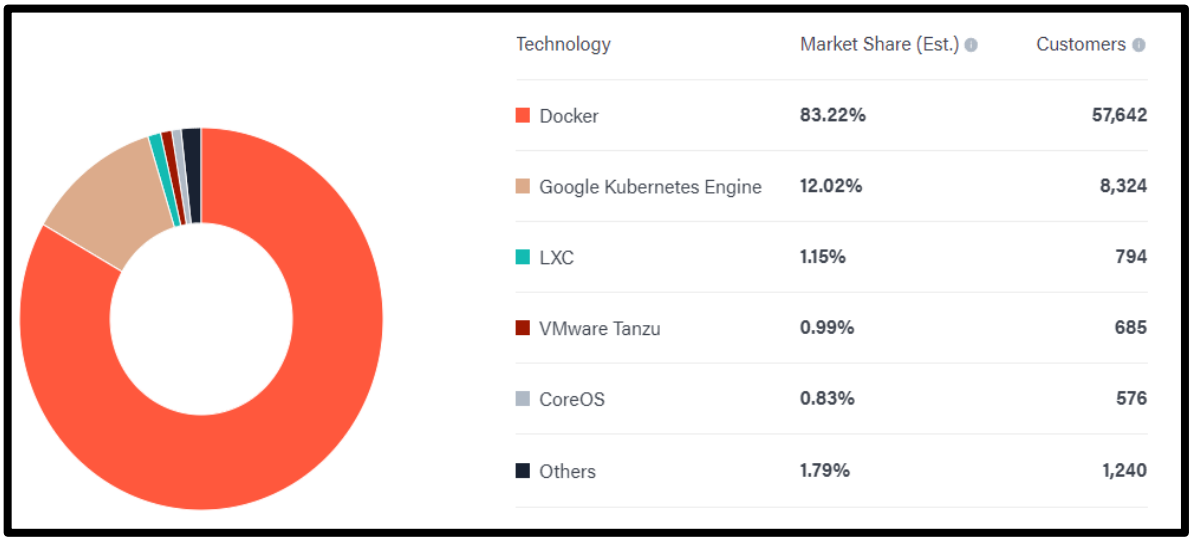


Figure 5: Top 5 Containerization technologies in 2024. (6sense, 2024).

As seen in Figure 5, Docker is the leading technology used for containerization worldwide, accounting for 83.22% of the market. (6sense, 2024).

5 API Integration

In this chapter, the method and practices of the integration of APIs in the front-end of the CAMARA project are described. It goes into a discussion on how to manage API requests and responses, deal with errors, and deliver a reliable execution of API calls. Special emphasis is on the application of strong methods to guarantee continuous data flow between the front-end application and the API/back-end services.

5.1 Integrating CAMARA's APIs with Axios

Axios is a promise-typed HTTP client that makes programming API integrations easy in the browser and Node.js environments. Due to its isomorphic nature, it allows the same code to be executed smoothly in both environments. For the client side Axios makes API calls using "XMLHttpRequests" while on the server side Axios uses the native Node.js HTTP module.

Axios offers a collection of capabilities that make it one of the best candidates for API interactions, including:

- Support for Promises: It makes use of JavaScript promises, which simplifies the handling of asynchronous requests, and also makes the code more readable and easier to maintain.

- Request and Response Interception: Interception/modification of requests or responses can be achieved in Axios, where additional authorization headers or logging can be added, for example.

- Automatic Data Transformation: It automatically serializes JavaScript objects into JSON and parses JSON responses, which is more efficient in terms of request and response handling. (JSON, N.D).

- Timeouts and error Handling: It supports setting timeouts for requests, and decent error handling features, which are crucial for managing API calls that may fail due to network issues or server errors.

Axios gracefully performs the job of data format, not just JSON, “FormData”, and “application/x-www-form-urlencoded”, with ease, providing a versatile solution to many API needs. (Axios, 2024)

In the scope of the present study, Axios is employed to handle API calls, to maintain fluent communication between the services of the backend. It integrates several request types (e.g., “GET”, “POST”, “PUT”, “DELETE” to access (retrieve) or modify data from the CAMARA APIs with better control over data transfer (e.g., data input, data output)). Such ease and freedom in the manipulation of complex API requests makes Axios a natural fit for this type of integration task. (Axios, 2024)

5.2 Handling API Requests and Responses

Efficient handling of API (Application programming interface) requests is one of the key factors for both good user experience and error handling. Figure 6 provides an illustration of how an authenticated request can be made to obtain a device identifier.

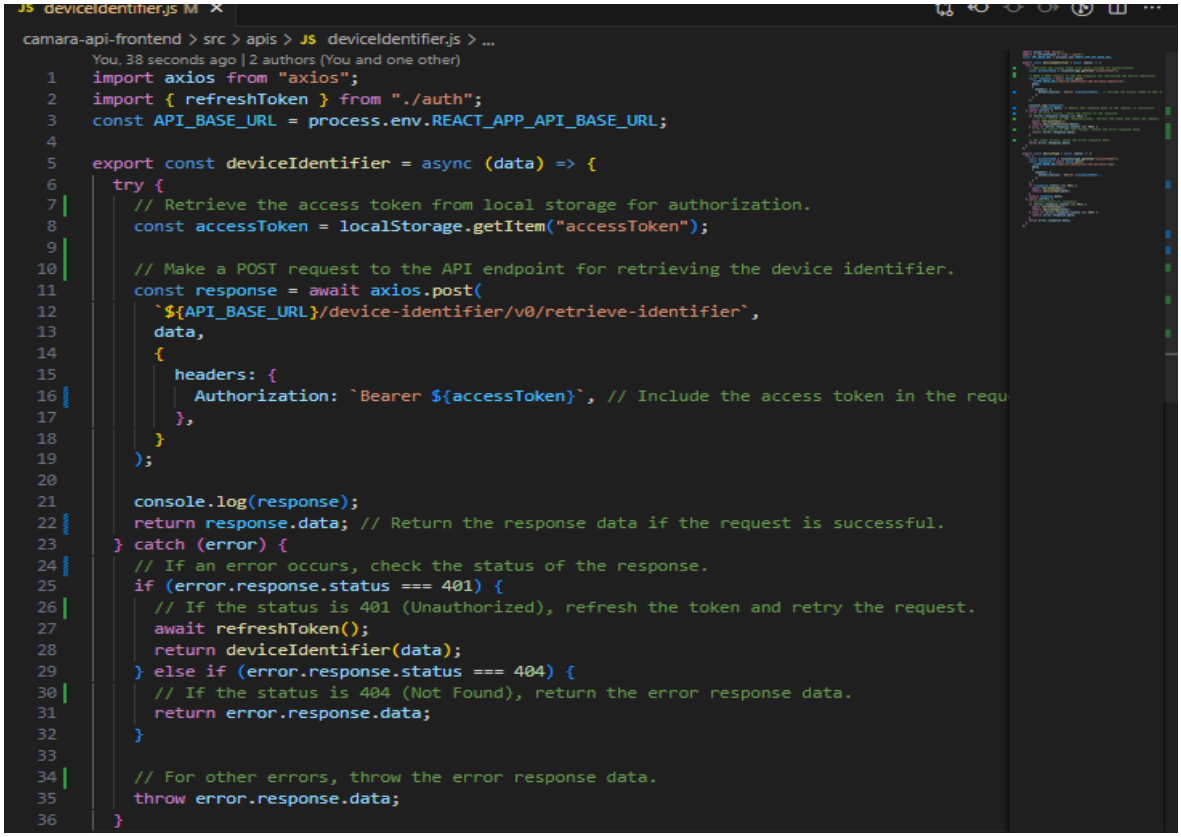


Figure 6. Device Identifier function from the project.

As seen in Figure 6 above, the code gets an "accessToken" from the local storage of the browser and an authorization request to the API is made. When a 401 Unauthorized error is caught (meaning token has expired), the token is refreshed, and the request is retried. The "accessToken" is stored in the browser's local storage only after a successful login request. This methodology influences the treatment of authentication on the client-side for device Identifier API request.

The function is sufficiently general to be applied for any number of API requests by changing the “API_BASE_URL” and the endpoint, and thus applicable across different API interactions, such as device identification. This represents the reusability in the code. If the request is successful, “response.data” which is the data to be displayed to the user, is returned, containing the necessary information.

Different error scenarios are handled specifically:

- For a “401 Unauthorized” error, the “refreshToken” function is invoked to refresh the access token, and the request is retried.
- For “404 Not Found” error, the error information is also provided to show that the requested data could not be retrieved.
- All other errors are thrown to be handled by the calling function or component in a structured manner.

Error handling is described in detail in the following section.

5.3 Error handling

Effective error handling ensures that users receive clear feedback about issues and that the application can recover from minor errors without a significant impact on the user experience.

This project’s approach to error handling involves:

- Token Expiry Management: When a “401 Unauthorized” error is encountered due to an expired token, the function initiates a process to refresh the access token using the “refreshToken” function, then retries the original request with the new token.

- Handling Not Found Errors: When an error response of type “404 Not Found” is received, the error response data is directly returned. This enables the user interface (UI) to notify the user that state requested data or resource is absent.

- General Error Handling: For other unexpected errors, raising the error guarantees that higher-level functions or components can deal with them as they are needed, maintaining the modularity of error handling.

This organized way of error handling guarantees that API interactions are robust and that the user experience is not compromised by problems like token expiry or

resource unavailability, it also promotes the maintainability of its code by making errors more predictable and therefore controllable.

There is an error that tends to happen in users' inputting wrong data. Using client-side validation helps to identify such errors at an early stage and to always send the valid data to the API.

For instance, by using a function such as "ValidateMSISDN" for validation of MSISDN format a phone number helps prevent wrong requests. When an input error is detected, a notification is shown to the user via "enqueueSnackbar", providing an informational message such as "Invalid phone number format" or "Please select an API"

Figure 7 illustrates error handling for incorrect user input.

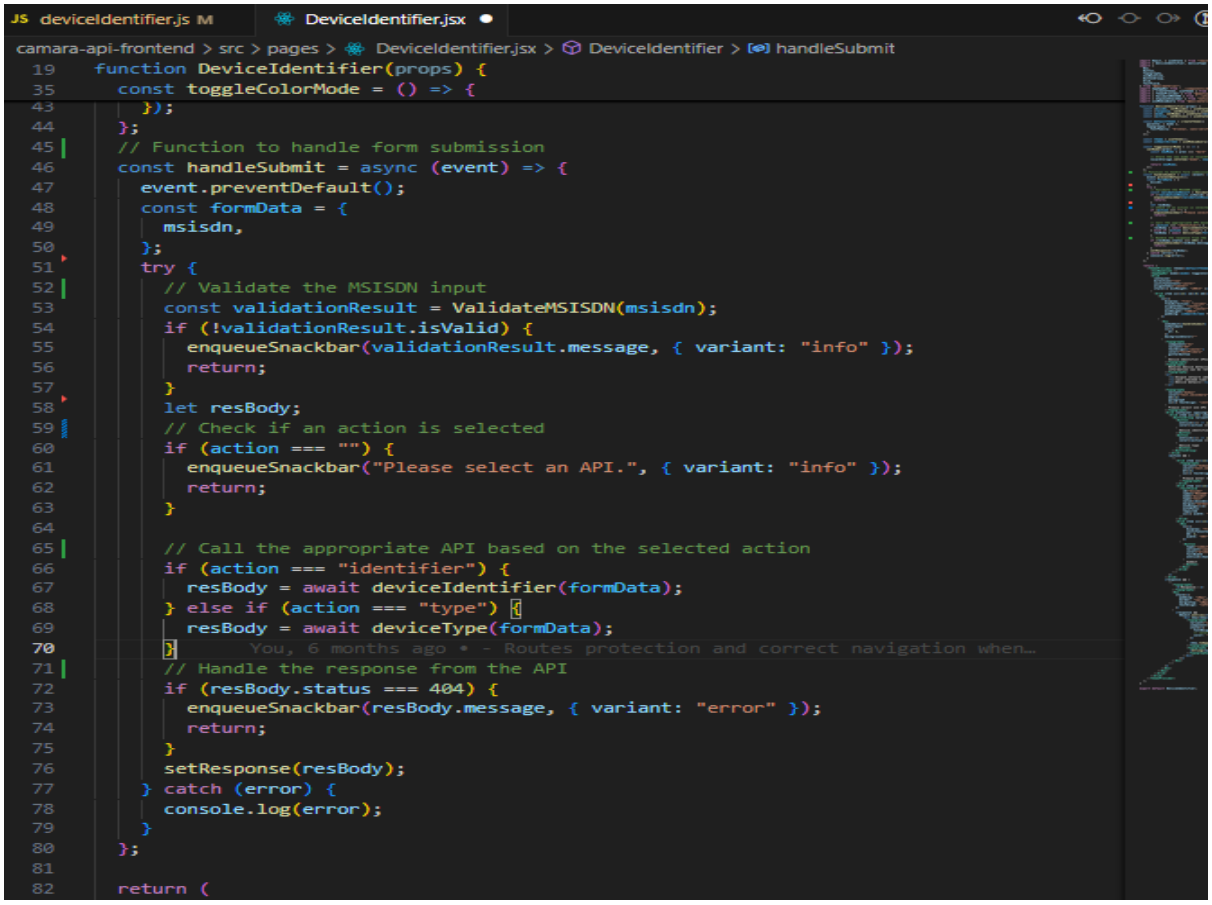


Figure 7. Device Identifier user input error's handling.

As seen in Figure 7 above, the code has a data validation phase prior to sending a background request to an API serving as “validateMSISDN”, the function checks the validity of the user input, thereby mitigating the risk of incorrect data. With “enqueueSnackbar”, users are immediately informed of incorrect inputs or missing data, allowing them to fix the inputs before resubmitting.

The CAMARA project is designed to provide a responsive and user-friendly experience by following API integration guidelines, which will keep the data access and display smooth by utilizing the backend API calls. Implementation of libraries (such as “Axios”) for optimized request management and for a standardized error handling provides strong communication between the front-end and the API services.

6 Testing

This chapter describes the testing process of the front-end of the CAMARA project, with the aim of verifying the usability and robustness of the system by carrying out extensive unit and integration testing. Testing was an intrinsic part of the development flow since its earliest stages and unit tests were also an integral part of the flow when small components are verified as soon as the component implements the functionality.

Because of the complexity of the system, namely, having a front-end module (CAMARA API front-end) and two backend modules (API service backend and NEF-NBI-Event-Monitoring backend that handles the subscriptions) unit tests were especially important. These tests ensured perfect data exchange between the frontend and backend services and validated that data exchanged by the API is processed as intended from frontend to backend, through the whole system. The integration tests further validated the compatibility and correct behaviour of these linked modules and confirmed the robustness of the whole architecture.

The next sections present a detailed description of certain performed tests, their use and findings as well as a text in each table of the significance of the tests conducted after that shows the outcome of the tests.

Example tests:

a) Component Rendering

Table 1. Component rendering test confirms the basic rendering functionality of the component, ensuring it is displayed correctly to users.

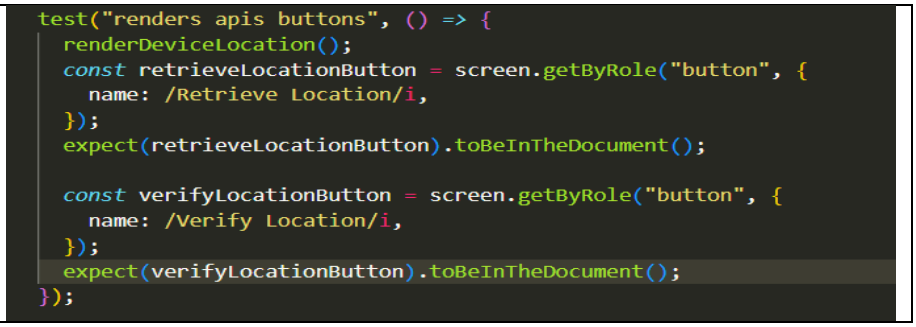
Aspect	Details
Test Case	Ensure that the device location component renders correctly by checking for the presence of the main title.
Explanation	In this test, it renders the component and check for the title “Device Location APIs” to confirm the component’s successful rendering.

Code Illustration	`test("renders device location page", () => {`
	`  renderDeviceLocation();`
	`  const title = screen.getByText(/Device Location APIs/i);`
	`  expect(title).toBeInTheDocument();`
	`});`

b) API Buttons Render Properly

Table 2. The API Buttons rendering test verifies the UI's completeness by confirming that all essential buttons are visible.

Aspect	Details
Test Case	Verify the presence of API interaction buttons, like Retrieve Location, Verify Location, etc.
Explanation	This test ensures that each button, which corresponds to different API actions, is rendered. This setup allows users to interact with the various functionalities provided by the component.

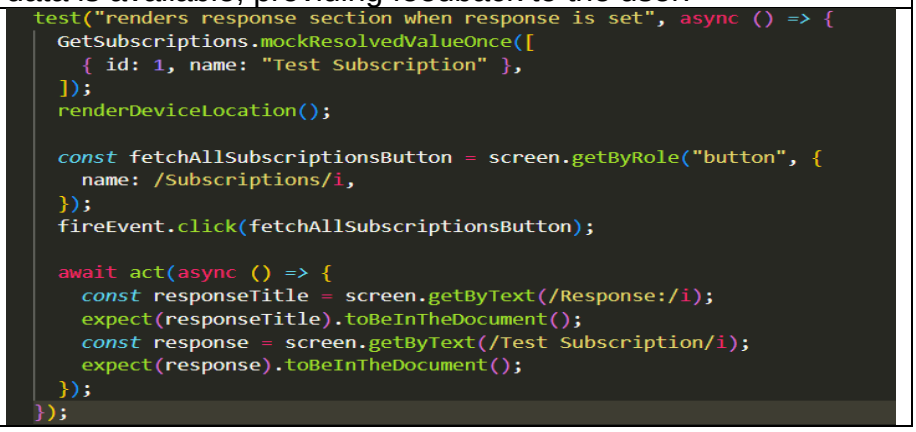
Code Illustration	

```
test("renders apis buttons", () => {
  renderDeviceLocation();
  const retrieveLocationButton = screen.getByRole("button", {
    name: /Retrieve Location/i,
  });
  expect(retrieveLocationButton).toBeInTheDocument();

  const verifyLocationButton = screen.getByRole("button", {
    name: /Verify Location/i,
  });
  expect(verifyLocationButton).toBeInTheDocument();
});
```

c) Response Section Rendering

Table 3. Test focusing on the dynamic rendering of the response section based on the availability of API data.

Aspect	Details
Test Case	Ensure the response section displays when the API call returns data.
Explanation	By mocking the “GetSubscriptions” API call to return sample data, this test ensures that the response section renders when data is available, providing feedback to the user.
Code Illustration	

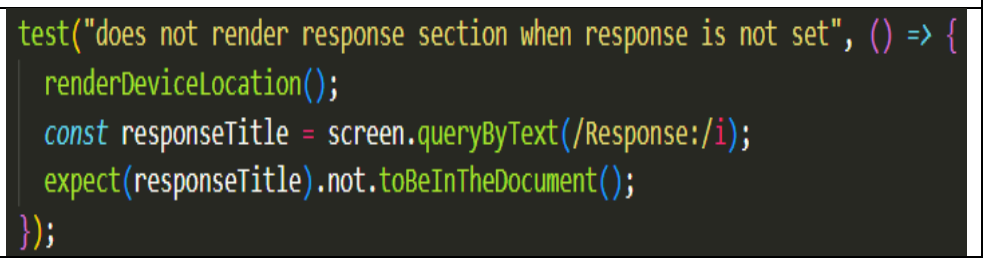
```
test("renders response section when response is set", async () => {
  GetSubscriptions.mockResolvedValueOnce([
    { id: 1, name: "Test Subscription" },
  ]);
  renderDeviceLocation();

  const fetchAllSubscriptionsButton = screen.getByRole("button", {
    name: /Subscriptions/i,
  });
  fireEvent.click(fetchAllSubscriptionsButton);

  await act(async () => {
    const responseTitle = screen.getByText(/Response:/i);
    expect(responseTitle).toBeInTheDocument();
    const response = screen.getByText(/Test Subscription/i);
    expect(response).toBeInTheDocument();
  });
});
```

d) Conditional Response Section

Table 4. Test ensuring that the user interface remains uncluttered by hiding unnecessary sections.

Aspect	Details
Test Case	Check that the response section is not rendered if there is no data.
Explanation	Here, the absence of data results in the response section remaining hidden. This ensures a clean UI with no unnecessary elements.
Code Illustration	

```
test("does not render response section when response is not set", () => {
  renderDeviceLocation();
  const responseTitle = screen.queryByText(/Response:/i);
  expect(responseTitle).not.toBeInTheDocument();
});
```

e) Retrieve Location Form

Table 5. Test validating the presence of form elements needed for the location retrieval process.

Aspect	Details
Test Case	Verify that the Retrieve Location form fields and submit button render.
Explanation	This test confirms that the Retrieve Location form displays all necessary input fields and the “Retrieve!” button, making it ready for user interaction.
Code Illustration	

```
test("renders Retrieve Location form.", () => {
  renderDeviceLocation();
  fireEvent.click(screen.getByRole("button", { name: /Retrieve Location/i }));

  expect(screen.getByLabelText(/Phone Number/i)).toBeInTheDocument();
  expect(screen.getByLabelText(/Network Access Identifier/i)).toBeInTheDocument();
  expect(screen.getByRole("button", { name: /Retrieve!/i })).toBeInTheDocument();
});
```

f) Get a Subscription By ID

Table 6. Test ensuring that users can retrieve and view subscription details accurately.

Aspect	Details
Test Case	Validate that the component fetches and displays subscription details using its ID.
Explanation	This test verifies that entering a valid subscription ID retrieves and displays the subscription details.
Code Illustration	

```
test("renders Get Subscription By Id form.", async () => {

  GetSubscriptionById.mockResolvedValue({ subscriptionId: "7", ... });

  renderDeviceLocation();
  fireEvent.click(screen.getByRole("button", { name: /Get Subscription By Id/i }));

  await act(async () => {
    fireEvent.change(screen.getByLabelText(/Subscription Id/i), { target: { value: "7" } });
    fireEvent.click(screen.getByRole("button", { name: /Submit/i }));
  });
  expect(screen.getByText(/phoneNumber: "12345678901"/i)).toBeInTheDocument();
});
```

g) Delete Subscription Form

Table 7. Test validating the deletion workflow by ensuring proper feedback is given to the user after an operation.

Aspect	Details
Test Case	Check that the deletion of a subscription works correctly.
Explanation	This test ensures that after entering a subscription ID and submitting the form, the component shows a success message upon deletion.
Code Illustration	

```
test("renders Delete Subscription form.", async () => {
  DeleteSubscription.mockResolvedValue({ status: 204, data: {} });
  renderDeviceLocation();
  fireEvent.click(
    screen.getByRole("button", { name: /Delete Subscription/i })
  );
  await act(async () => {
    fireEvent.change(screen.getByLabelText(/Subscription Id/i), {
      target: { value: "1" },});
    fireEvent.click(screen.getByRole("button", { name: /Submit/i }));
  });
  expect(screen.getByText(/Subscription deleted successfully/i)
  ).toBeInTheDocument();
});
```

Above (Tables 1-7) there are examples of tests for the Device Location component. In this project, thorough testing was conducted for all system parts, including the frontend and backend endpoints where applicable. The results of these tests are summarized as "passed" as shown in Figure 8, which highlights the number of test suites and individual tests successfully executed.

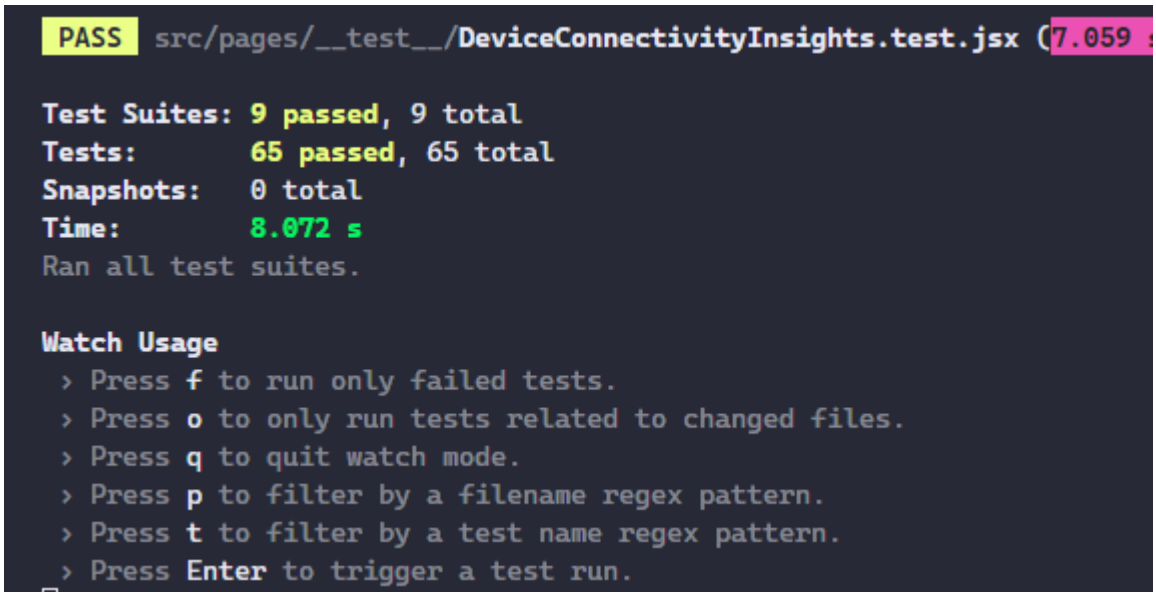


Figure 8. Successful execution of test suites and individual tests for the project.

Figure 8 shows that all 9 test suites and 65 individual tests passed.

7 Results

This chapter elaborates the test result of the CAMARA API project, providing screenshots of their respective pages and the purpose of the API. After completing the development of the initial four API pages (Device Identifier, Device Location, Device Status, and Edge Cloud) the application underwent acceptance testing by the customer. The test evaluated the usability, functionalities, and total performance of the created pages. The customer has also been pleased with the result and has requested that the same style and methodology be applied to the remaining API families.

As development progressed for the rest of the API pages, the same testing process was followed to maintain consistency and ensure customer satisfaction. A functional list was created specifying all functionalities that are to be tested for each API. This list served as a guide to systematically verify that each feature met the specified requirements and functioned as intended. Initial testing was carried out to test the functionalities against the specified subset of functionalities, conducted by the developer. Then a second round of tests was conducted by a member of the team for verification of findings and to improve quality assurance. This ensured that all API pages adhered to the same high standards of usability and functionality, meeting the customer's expectations at each stage.

All purposes and requirements as planned in the project were executed successfully and the set objectives were achieved. Specifically, it should be noted that although different API families are also present elsewhere in the system, they are not yet within the scope of this project. The following set of APIs is explained and discussed in the next section to give an overview of existing work for reference in future work.

7.1 Device Identifier API

This API family allows the consumer to retrieve the current identity (IMEI) of the mobile device being used by a given mobile subscriber and the type (manufacturer and model) of the mobile device used by a given mobile subscriber. (CAMARA Project Atlassian, 2024). Figure 9 shows the interface, highlighting the two APIs available for user interaction, which can be accessed by selecting one from the options.

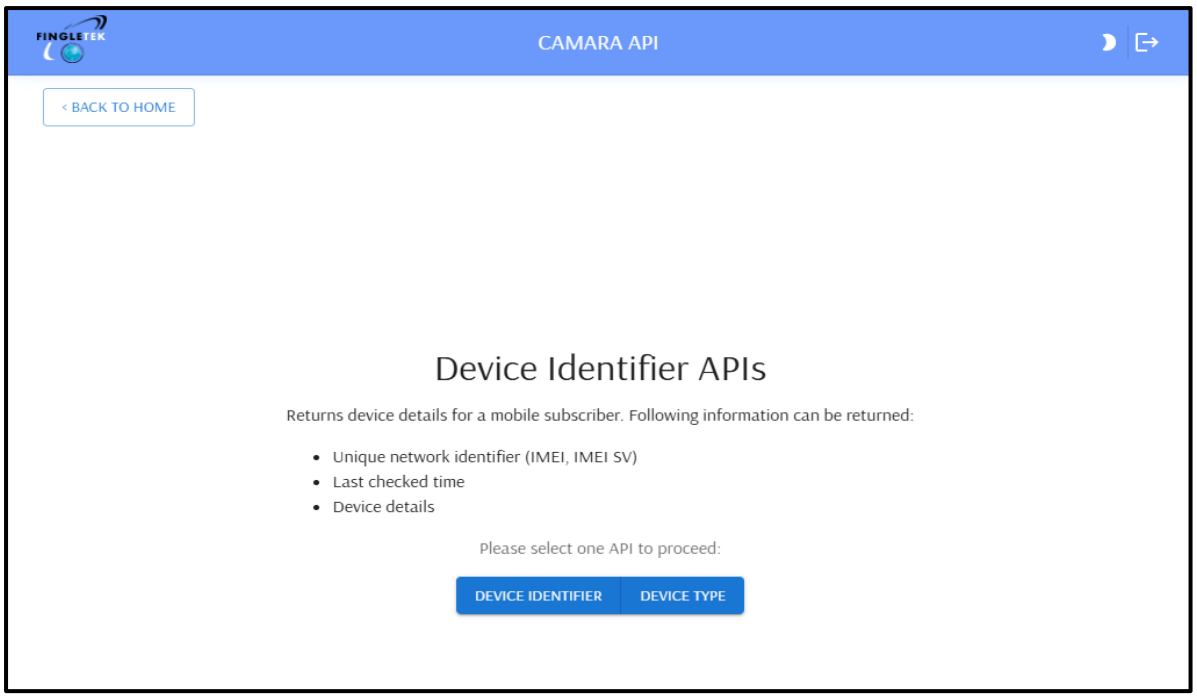


Figure 9. Device identifier API page from CAMARA API project.

After choosing the desired one the corresponding form will show asking for the required inputs.
Then the user can submit the request form, if the inputs are valid then the user can submit it to get the data from the API service as shown in Figure 10.

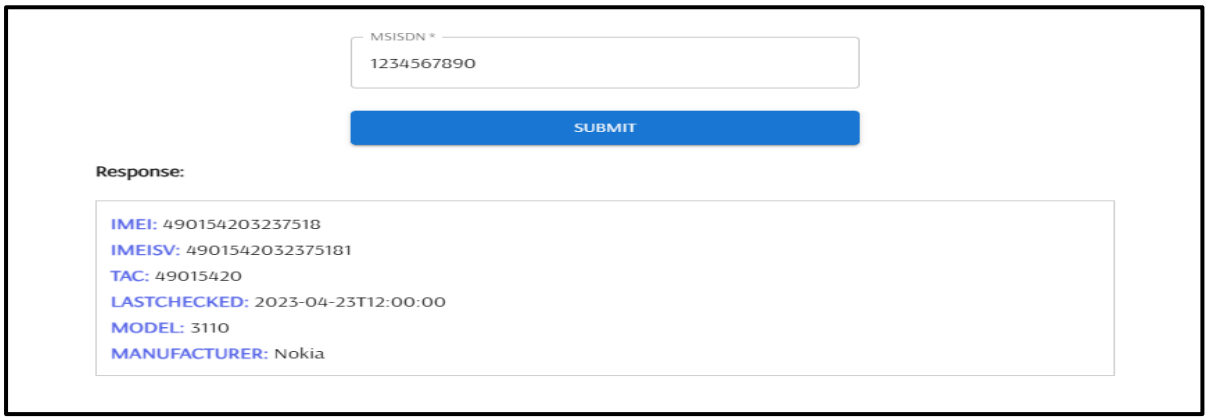


Figure 10. Device identifier example response.

As shown in Figure 10, the user can see the response after giving “MSISDN” of the device and submitting the request.

7.2 Device Status API

This API family allows the consumer to check if a device is reachable or is not connected to the network, check if a device is roaming, and in which country, and allows them also to receive notifications if the connectivity or roaming status of the device changes. (CAMARA Project Atlassian, 2024). Figure 11 shows the interface of the device status API, highlighting the available APIs for user interaction, which can be accessed by selecting one from the options.

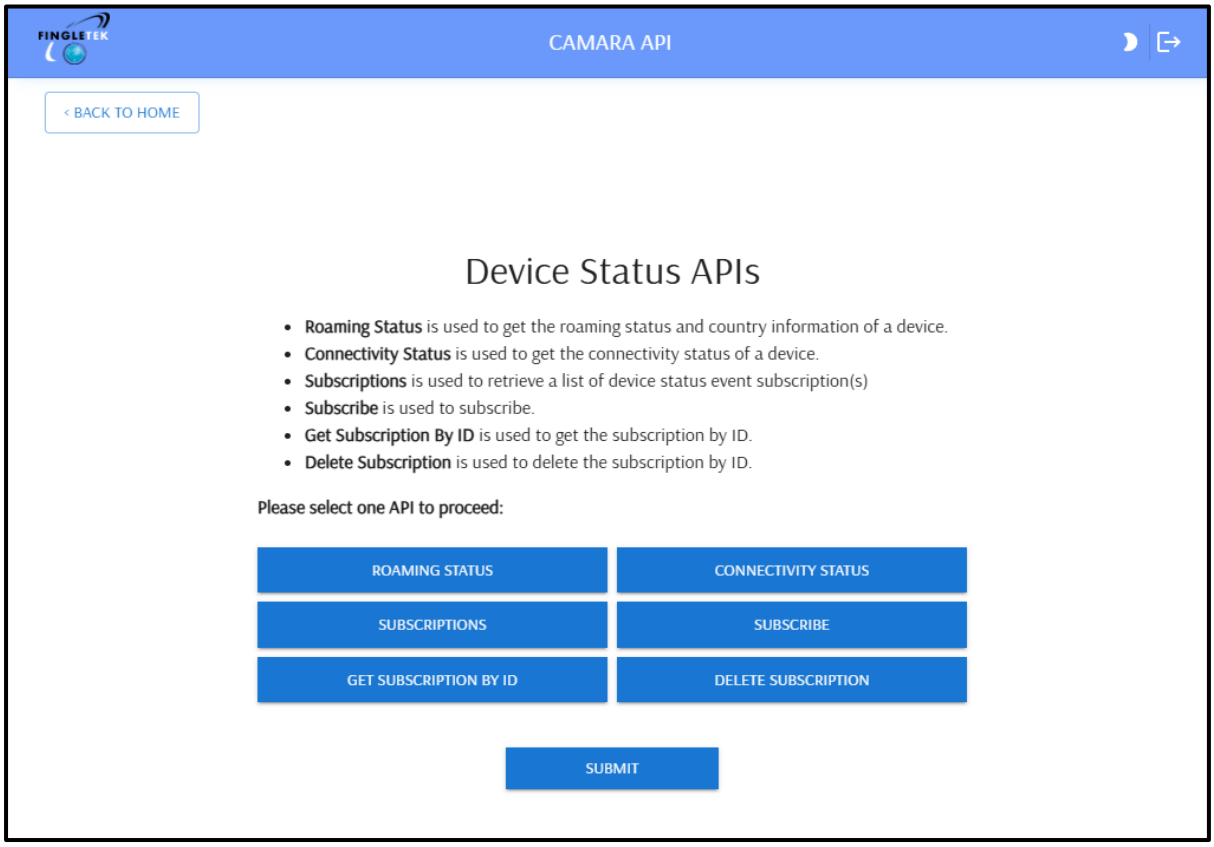


Figure 11. Device status API page of CAMARA API project.

After choosing the desired one a corresponding form will show asking for the required inputs. Then the user can submit the form and get the desired data from the API service if it is available. The response body is shown in Figure 12.

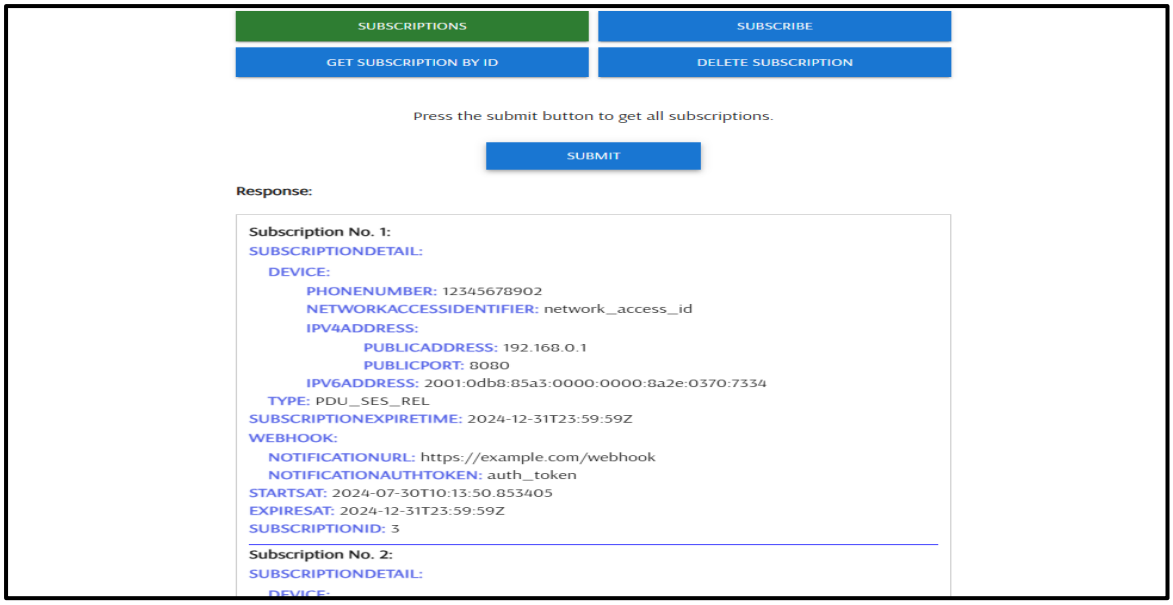


Figure 12. Device status API example response

As shown in Figure 12, the user can see the response to all subscriptions request after pressing the submit button.

7.3 Device Location API

This API family allows the consumer to verify the location of the device (Location-verification), retrieve the location of a device (Location-retrieval), and allows the consumer to subscribe and receive notifications about a device entering or leaving a certain area (Geofencing-subscription). This API family is limited to 4G and 5G. (CAMARA Project Atlassian, 2024). Figure 13 shows the interface of the device location API, highlighting the available APIs for user interaction, which can be accessed by selecting one from the options.

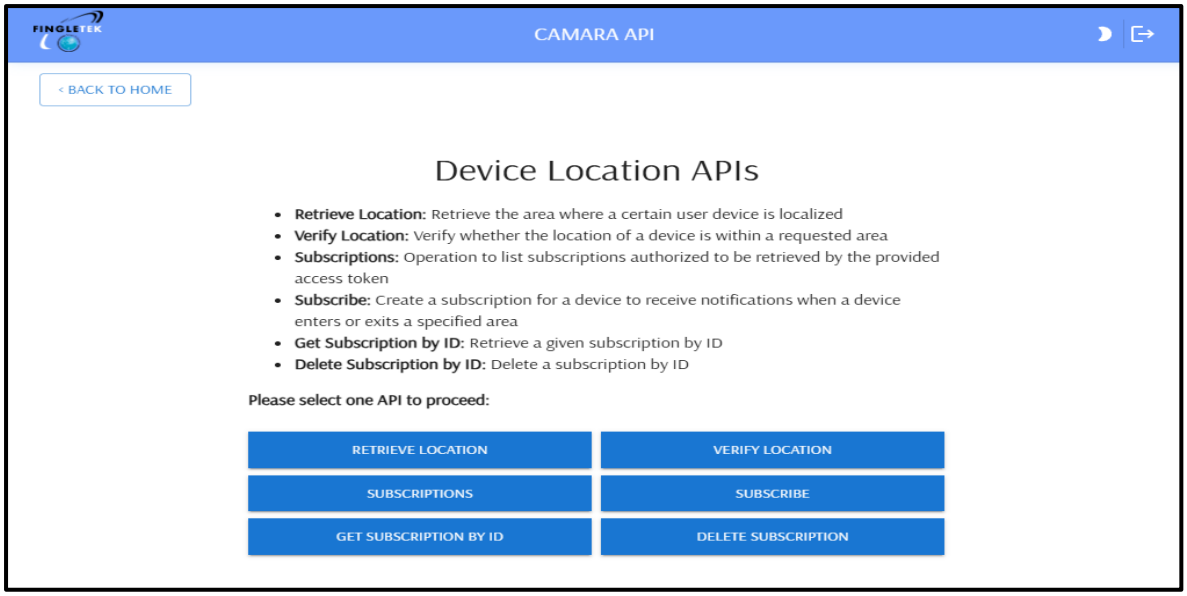


Figure 13. Device location API page of CAMARA API project.

After choosing the desired one, a corresponding form will show asking for the required inputs. Then the user can submit the form and get the data available by the API service. The example response is shown in Figure 14.

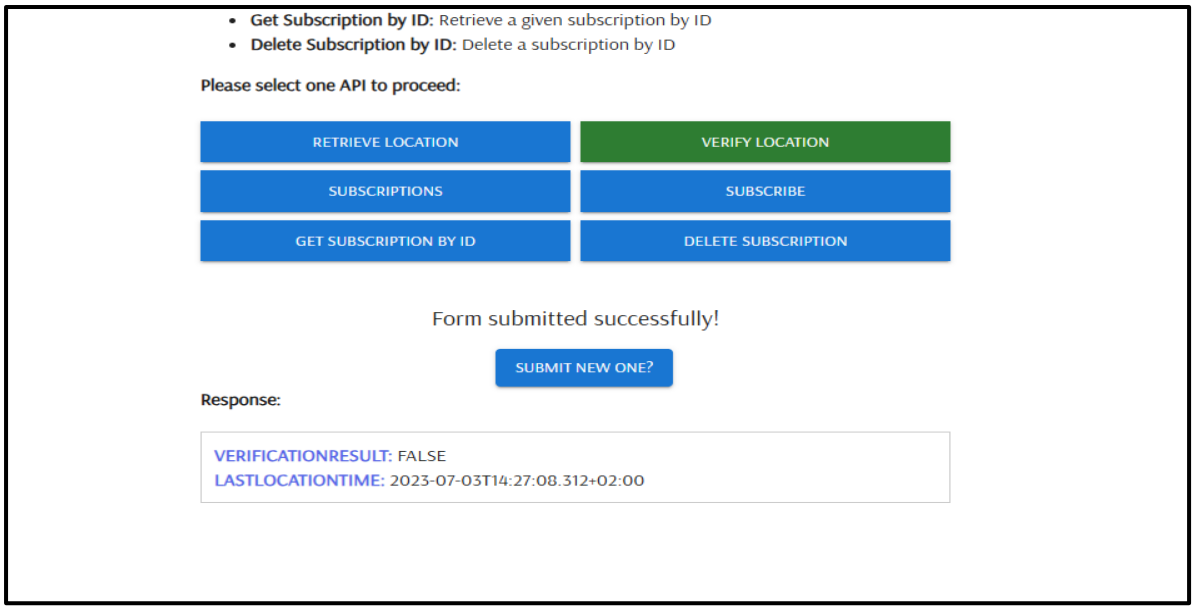


Figure 14. Device location example response.

As shown in Figure 14, the user can get the response after filling the form of Verify location API and pressing the submit button. The user can make a new request by pressing the “Submit new one?” button.

7.4 Edge Cloud API

This API family allows the consumer to discover the closest edge cloud zone to a given device, provide and manage application images to be deployed on resources within the operator network, use reserved compute resources within the operator network for the deployment of applications on VMs (Virtual machines) or countries and allows the consumer to influence the traffic routing from the user device toward the Edge instance of the application. This API is also limited to 4G and 5G. (CAMARA Project Atlassian, 2024). Figure 15 shows the interface of edge cloud API, highlighting the available APIs of this API family for user interaction, which can be accessed by choosing one from the options.

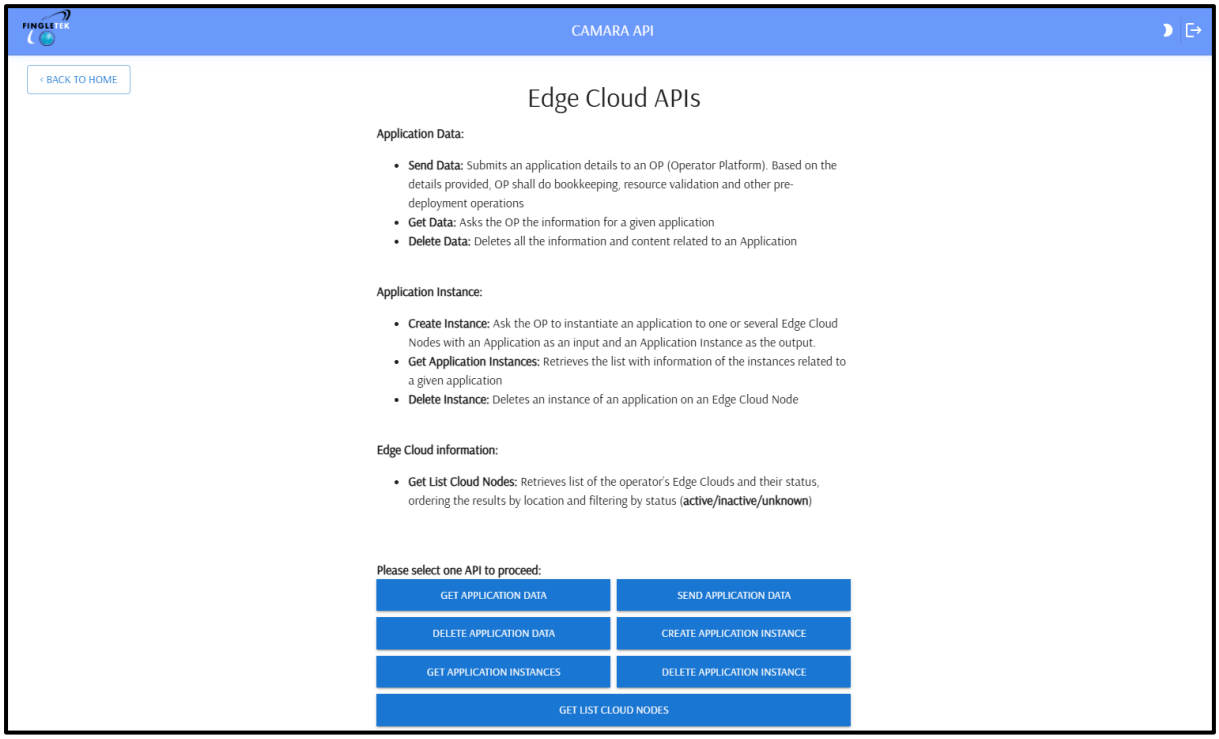


Figure 15. Edge cloud API page of CAMARA API project.

After choosing the desired one, a corresponding form will show asking for the required inputs for the chosen API. Then the user can submit the form and get the data available by the API service. The example response body is shown in Figure 16.

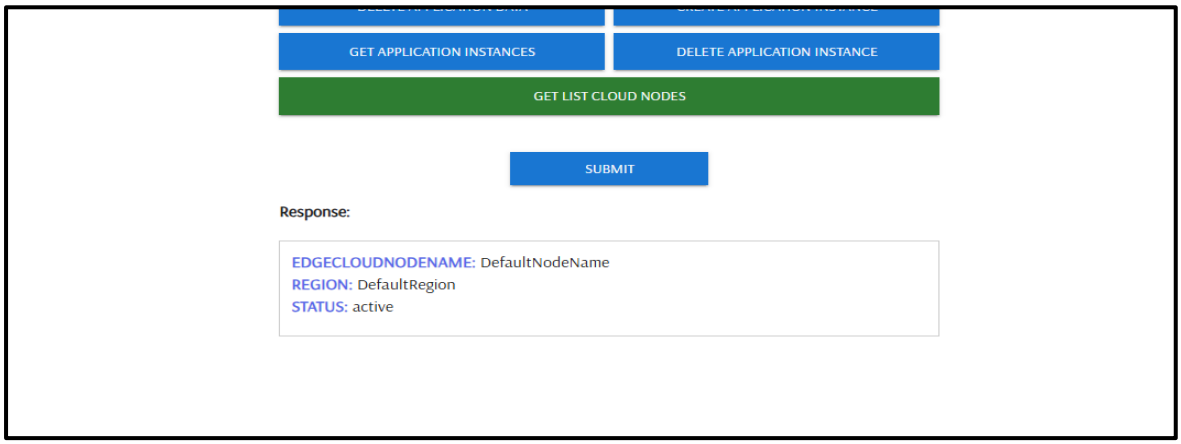


Figure 16. Edge Cloud API example response.

As shown in Figure 16, the user can see the response to “Get List Cloud Nodes” by pressing the “Submit” button.

7.5 Device Connectivity Insights API

This API family allows the consumer to define intents in the form of policy thresholds for QoS (Quality of Service) metrics against the device and the application service. The API service will alert the consumers when the policy has been breached. This API is also limited to 4G and 5G. (CAMARA Project Atlassian, 2024).

Figure 17 shows the interface of device connectivity insights API, highlighting the available APIs of this API family for user interaction, which can be accessed by choosing one from the options.

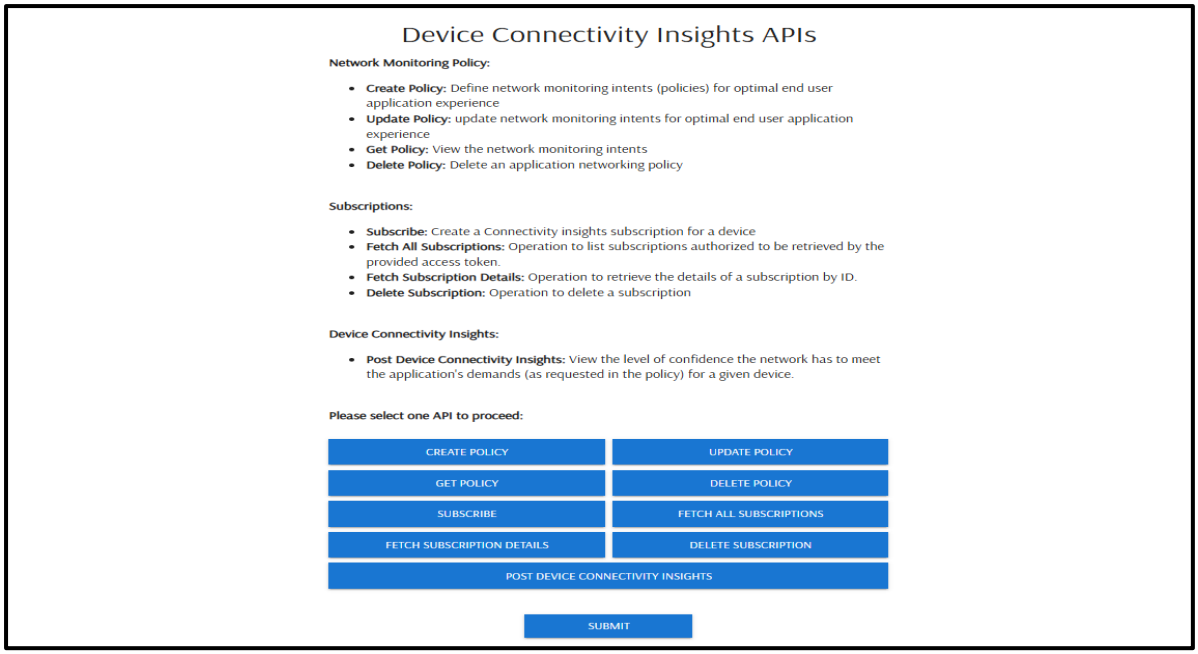


Figure 17. Device connectivity insights API interface of CAMARA API project.

After selecting the desired API, a corresponding form will show asking for the required inputs for the chosen API. Then the user can submit the form and get the data available by the API service. The example response body is shown in Figure 18.

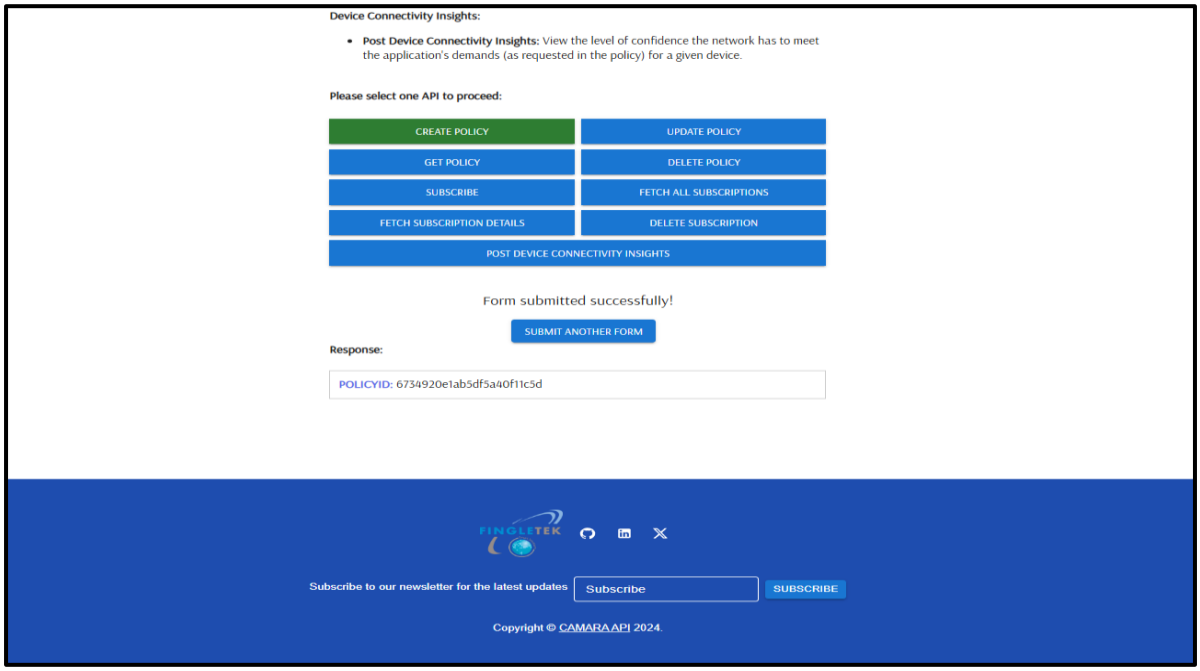


Figure 18. Device connectivity insights example response.

As shown in Figure 18, the response after a successful policy creation request, the API service sends back the ID of the created policy.

8 Conclusion

This chapter contains a summary, future work, and closing words. The chapter starts with a summary highlighting the most important aspects of each step, moves on to the important contributions after this, discusses the possible future work, and ends in closing words highlighting what it was like to work on this project and what was accomplished.

This thesis discusses implementing, testing, and evaluating a system integrating multiple API families, including the Device Identifier APIs, Device Status APIs, Device Location APIs, Edge Cloud APIs, and Device Connectivity Insights APIs. The goal was to create a robust platform that facilitates seamless interaction with these APIs while ensuring reliability, usability, and performance.

The thesis starts by describing the reasons behind the system, particularly the increasing need for fully integrated, efficient, and user-friendly API solutions. Having been carefully designed and iteratively developed, the system was designed to handle critical aspects of device management, data accessibility, and service scalability.

Key achievements include the successful implementation of API interfaces, comprehensive front-end and integration testing, and validation of the system’s capabilities through real-world use cases. Testing efforts focused on ensuring functionality, reliability, and adherence to best practices in front-end testing, as documented in the relevant chapters. These tests played an important role in discovering and resolving possible errors and in the end, it provided a finished and robust application.

In addition, the presentation of results for each of the available APIs shows the system's support for a variety of functionalities, making the system suitable for device identification, for monitoring connectivity dynamics in real-time, as well as for providing connectivity-related information. The visual representations and functionally descriptive information of the interface for each of the APIs demonstrate the practicality and usability of the developed solution.

This study makes several important contributions:

- Integrated System Design: A unified platform that simplifies API interactions for both developers and end-users.
- Comprehensive Testing Framework: Adoption and demonstration of best practices in front-end and integration testing for improved quality.
- Enhanced Usability: A user-friendly interface that ensures accessibility to essential IoT services and insights.

While the system achieves its stated objectives, further improvements can be made: Expanding the system to support additional APIs and functionalities such as Home Devices Qod (Quality on Demand) APIs and Device Swap API, implementing advanced analytics and machine learning capabilities for predictive insights, refining the user interface to enhance usability based on user feedback and usability testing, and strengthening system security to meet emerging challenges in IoT and edge computing environments.

The experience of working on this postgraduate study was challenging and rewarding. It is a significant step towards the development of efficient and integrated systems for the management of API services. It provided the opportunity to go deep into the difficulties of device data management, edge computing, and API development. After efficient research and practical exercises in problem-solving, a deep knowledge of the topic was attained, which then allowed the design of practical solutions and achieve significant results.

References

Camara Project. 2023 Camara Project. Available at: <https://camaraproject.org/>. Accessed 26 August 2024.

Cyber Legion. 2024. A comprehensive guide to network APIs: utilization and security practices. LinkedIn. Available at: <https://www.linkedin.com/pulse/comprehensive-guide-network-apis-utilization-security-practices-etwhe/>. Accessed 27 August 2024.

Goss, Michaela. 2024. 5G vs. 4G: Learn the key differences between them. TechTarget. Available at: <https://www.techtarget.com/searchnetworking/feature/A-deep-dive-into-the-differences-between-4G-and-5G-networks>. Accessed 30 August 2024.

De Micheli, Carlo; Gorgiev, Zoran. 2024. APIs in the Telecom Industry. Equixly. Available at: <https://equixly.com/blog/2024/04/09/apis-in-telecom/>. Accessed 30 August 2024.

Cloudinary. 2024. Front-End Development: The Complete Guide. Available at: <https://cloudinary.com/guides/front-end-development/front-end-development-the-complete-guide>. Accessed 02 September 2024.

Trienpont International. 2023. 5 Reasons Why JavaScript is the World's Most Popular Programming Language. Medium. Available at: <https://medium.com/@trienpont/5-reasons-why-javascript-is-the-worlds-most-popular-programming-language>. Accessed 04 September 2024.

StackOverflow survey. 2023. Most popular technologies. Available at: <https://survey.stackoverflow.co/2023/#section-most-popular-technologies-programming-scripting-and-markup-languages>. Accessed 06 September 2024.

Angular. 2024. What is Angular? Available at: < https://angular.dev/overview>. Accessed 26 September 2024.

Vue.js. 2024. Introduction what is Vue? Available at: < https://vuejs.org/guide/introduction.html>. Accessed 26 September 2024

Bootstrap. 2024. Get started with Bootstrap. Available at: < https://getbootstrap.com/docs/5.3/getting-started/introduction/>. Accessed 25 September 2024.

GeeksforGeeks. 2024. REST API Introduction. Available at: < https://www.geeksforgeeks.org/rest-api-introduction/>. Accessed 27 September 2024.

Docker. 2024. What is Docker? Available at: < https://docs.docker.com/get-started/docker-overview/>. Accessed 23 September 2024.

Swagger. 2024. About Swagger. Available at: < https://swagger.io/about/>. Accessed 24 September 2024.

Raluca, Budiu; Jakob, Nielsen. 2012. Mobile Usability. Pearson Education, 2012. Accessed 07 October 2024

Mohamed, A. & W.K, Dr & Rimiru, R. & Ondago, C. (2014). Responsive Web Design in Fluid Grid Concept Literature Survey. The International Journal of Engineering and Science (IJES). 3. 49-57. Available at <https://www.researchgate.net/publication/279985804_Responsive_Web_Design_in_Fluid_Grid_Concept_Literature_Survey>. Accessed 07 October 2024.

MDN. 2024. JavaScript technologies overview. Available at: < https://developer.mozilla.org/en-US/docs/Web/JavaScript/JavaScript_technologies_overview>. Accessed 26 September 2024.

React. 2023. Thinking in React. Available at: <https://react.dev/learn/thinking-in-react> Accessed 09 September 2024.

Patel, Jeel. 2024. Why Use React? – Top 8 Reasons Experts Use React in 2024. Available at: <https://www.monocubed.com/blog/why-use-react/>. Accessed 09 September 2024.

BuiltWith. 2024. React Usage Statistics. Available at: <https://trends.builtwith.com/javascript/React>. Accessed 10 September 2024.

Material UI. 2024. Material UI - Docs. Available at: <https://mui.com/material-ui/getting-started/>. Accessed 13 September 2024.

Cisco. 2024. What is 5G? Available at: <https://www.cisco.com/c/en/us/solutions/what-is-5g>. Accessed 23 September 2024.

Dirin, A., Nieminen, M., & Laine, T. H. 2022. Feelings of Being for Mobile User Experience Design. International Journal of Human-Computer Interaction, 39(20), 4059–4079. Available at: <https://doi.org/10.1080/10447318.2022.2108964>. Accessed 23 September 2024.

DynaTech. 2021. Swagger API Testing – Meaning, Working Methodology & More. Available at <https://dynatechconsultancy.com/blog/swagger-api-testing>. Accessed 30 September 2024.

Sematext. 2022. Docker. Available at <https://sematext.com/glossary/docker/>. Accessed 30 September 2024.

6sense. 2024. Containerization. Available at <https://6sense.com/tech/containerization>. Accessed 30 September 2024.

Designveloper. 2024. 8 Responsive Web Design Principles You Need To Know. Available at: <https://www.designveloper.com/blog/responsive-web-design-principles/>. Accessed 1 October 2024.

Carlos Hernández-Nieto, Xavier Sánchez, Patricia Salinas. 2015. A Mobile First Approach for the Development of a Sustainability Game, Procedia Computer Science, Volume 75, Pages 182-185, ISSN 1877-0509. Available at <https://www.sciencedirect.com/science/article/pii/S1877050915036972>. Accessed 04 October 2024.

Axios. 2024. What is Axios?. Available at < https://axios-http.com/docs/intro>. Accessed 28 October 2024.

JSON. N.D. Introducing JSON. Available at < https://www.json.org/json-en.html>. Accessed 28 October 2024.

Dobbala, Manoj Kumar. (2022). Validate Faster, Develop Smarter: A Review of Frontend Testing Best Practices and Frameworks. Journal of Mathematical & Computer Applications. 1-4. 10.47363/JMCA/2022(1)151. Available at <https://www.researchgate.net/publication/380809023_Validate_Faster_Develop_Smarter_A_Review_of_Frontend_Testing_Best_Practices_and_Frameworks>. Accessed 04 November 2024.

Jest. 2022. Getting started. Available at <https://archive.jestjs.io/docs/en/22.x/getting-started.html>. Accessed 05 November 2024.

Mocha. 2017. Mocha Documentation. Available at <https://docs2.w3cub.com/mocha/>. Accessed 05 November 2024.

Cypress. 2024. Why Cypress?. Available at <https://docs.cypress.io/app/get-started/why-cypress>. Accessed 05 November 2024.

Selenium. 2024. Documentation. Available at <https://www.selenium.dev/documentation/>. Accessed 06 November 2024.

RTL. 2024. React Testing Library documentation. Available at <https://testing-library.com/docs/react-testing-library/intro/>. Accessed 06 November 2024.

CAMARA Project Atlassian. 2024. CAMARA Project. Available at <https://lf-camaraproject.atlassian.net/wiki/spaces/CAM/overview?homepageId=14548994>. Accessed 12 November 2024.