Fullstack web-based 5G metrics dashboard

Ettore Carlo Marangon
Technische Universität Berlin
ettore.carlo.marangon@campus.tu-berlin.de

Moritz Schelten
Technische Universität Berlin
m.schelten@campus.tu-berlin.de

Daniel Schrenk
Technische Universität Berlin
d.schrenk@campus.tu-berlin.de

***Abstract*—This report presents the design and development of a full-stack, web-based 5G metrics dashboard that captures, stores, and visualizes both live and historical data from a 5G network. The project was initiated in response to the limitations of current 5G testing tools, especially to the lack of persistent data storage capabilities in platforms like OAIBOX. Our solution overcomes these issues by designing a modular system that includes a React-based frontend, a Node.js backend, a SQL database, and a Python-based scraper for 5G metric data collection. Our 5G metrics dashboard solution provides real-time monitoring and interactive analysis through a user-friendly interface, offering comprehensive insights into the performance of 5G networks. The system architecture is containerized with Docker, ensuring easy deployment and scalability. This report covers the project's objectives, the system requirements, the development approach, and the performance evaluation of our solution.**

***Index Terms*—5G Dashboard, 5G metrics, Scraper, React, Web Application**

I. Introduction

5G technology represents the next generation of mobile networks, promising to revolutionize connectivity with faster speeds, lower latency, and the capacity to support a vast number of connected devices [1]. This technological advancement is essential for the development of smart cities, the evolution of IoT (Internet of Things), and the seamless integration of artificial intelligence into everyday life [2]. Research into 5G is crucial not only to fully harness these benefits but also to address challenges such as infrastructure demands, technology standardization, and equitable access [3]. Understanding and advancing 5G technology is key to fostering innovation, economic growth, and societal progress in an increasingly digital world [1].

In this context, practical experimentation with cellular networks, traditionally dominated by network vendors and telecommunications operators due to the high costs and licensing restrictions, is becoming more accessible [4]. The rise of open-source 3GPP protocol stacks, supported by affordable Software-Defined Radio (SDR) systems, is broadening access to the 5G technology. A notable example is the *OpenAirInterface*™ (OAI) project, which leads in providing an open-source, 3GPP-compliant implementation of a 5G system [4]. Its main product, the OAIBOX[1], is a plug-and-play testing solution that includes a 5G core network, base stations, and user equipment, all of which can operate on general-purpose computing platforms [4]. The OAIBOX offers a web dashboard that allows users to visualize and configure their 5G testbed, as well as monitor real-time metrics of the core 5G network and radio access network. While the 5G data is displayed in a user-friendly manner on the dashboard, the generated data from the test sessions is not stored in any database and is therefore lost at the end of each session. This can be problematic for several reasons: without persistent data, it's impossible to track the performance of the network over time, identify trends, or analyze the long-term impact of different configurations. Moreover, losing data after each session hinders the ability to conduct thorough comparisons between test runs, making it difficult to diagnose issues, optimize system performance, or validate the results of ongoing experimentation. In response to these challenges, we initiated our project to address the need for persistent historical 5G data from the OAIBOX experimentations. Our goal is to design and develop a 5G metric dashboard that captures and displays data from previous OAIBOX test sessions as well as from tests performed in real time.

This report will detail the project's structure, outline the system's requirements, and provide a comprehensive overview of the development approach of our 5G metric dashboard solution. We therefore structure it as follows:

(i) We give a concise outline of the project's organization and objectives (Section II)
(ii) We present a detailed definition of the requirements for the 5G metrics dashboard (Section III)
(iii) We describe the approach used in developing the application (Section IV)
(iv) We provide an assessment of the completed 5G metrics dashboard (Section V)

II. Project overview

Project goal: The objective is to develop a web-based dashboard application that displays stored and live metric data from a 5G mobile network, including base stations and user devices. The system should also be able to capture and store the live 5G test metrics in a database, allowing users to retrieve, view, and compare this data from various test sessions. To achieve this, we will reverse engineer and capture websocket messages from an existing OAIBOX dashboard that will provide us the 5G data.

Organization: The project was executed by a team of three members, each contributing to distinct aspects of the development process. Weekly meetings were conducted to review progress, discuss improvements, and identify the features necessary to achieve the system's objectives. An agile

[1]https://oaibox.com

development methodology was adopted, wherein the project was decomposed into individual stories, each representing a specific feature or task. These stories were then assigned to the team members for implementation. To ensure rigorous quality control, each new feature was submitted through a pull request within our GitHub repository. These pull requests underwent peer review by the other team members before integration into the main project. Task management and progress tracking were facilitated through GitHub's project management tools, where tickets and stories were systematically defined and organized. Additionally, a Continuous Integration and Continuous Deployment (CI/CD) pipeline was established within the GitHub repository using GitHub Actions[2]. This pipeline was triggered on pushes and pull requests to the main, DEV, and backend branches. It primarily focuses on the backend, performing essential tasks such as code checkout, setting up the Node.js environment, installing backend dependencies, and executing backend tests. This automated workflow ensured that the backend code was consistently tested, maintaining the integrity of the codebase and facilitating smoother integration of new features. Moreover, the pipeline deplxoyed a OpenAPI specification of the backends API using GitHub Pages [3].

III. System requirements

In this section, we outline the key requirements that guided the development of our 5G metrics dashboard. These requirements are categorized into three main areas: general system requirements, frontend requirements, and backend requirements.

A. General requirements

1) The system should capture all 5G websocket metric messages transmitted to the OAIBOX dashboard, ensuring no loss of data during the collection process.
2) The captured 5G metric data should be systematically sorted and classified into relevant categories and topics, enabling efficient retrieval and analysis based on specific parameters.
3) The categorized 5G metric data should be stored in a relational database, ensuring data persistence and long-term accessibility for subsequent analysis and reporting.
4) The system should be capable of displaying live 5G metric data, ensuring users to monitor ongoing network activities and performance metrics with minimal latency. Moreover, the live data should be directly scraped from the OAIBOX dashboard.

B. Frontend requirements

The following requirements address key aspects such as the performance, scalability and maintainability, user experience, as well as cross-browser compatibility of the dashboard's frontend.

1) The frontend must render live and stored data from the application's backend with a refresh rate of at least once per second, ensuring minimal delay in displaying critical metrics.
2) The frontend must be developed using a component-based architecture, allowing for easy reuse of UI elements and scalability as the project grows.
3) All UI components should be modular, with each component handling a specific piece of functionality, ensuring that new features can be added without impacting existing code.
4) The 5G metric data should be displayed using visual elements such as charts and graphs that are clear, intuitive, and easy to interpret by users.
5) The dashboard must provide interactive elements such as filters, dropdowns, and buttons to enhance the user's ability to explore and analyze the data effectively.
6) The frontend must be fully functional and visually consistent across all major web browsers, including Chrome, Firefox, Safari, and Edge, with no significant differences in user experience.

C. Backend requirements

The following requirements focus on the functional aspects necessary to ensure the backend of the dashboard effectively manages, processes, and serves 5G network metrics.

1) The backend must support the storage and retrieval of 5G network metrics with varying structures, ensuring flexibility to accommodate diverse types of metric data without requiring significant changes to the existing system.
2) The system must provide functionality to filter data based on a specified time range, allowing users to retrieve and analyze metrics from any desired time period efficiently.
3) The backend must implement data reduction techniques to minimize the load during data retrieval by selectively choosing and returning a representative subset of data points, especially when handling large volumes of time-series data.
4) The architecture must allow for easy extension to support new types of metrics. Adding support for additional metric types should be straightforward, without requiring extensive refactoring of the existing code.
5) The backend must implement a RESTful API that follows standard REST principles, providing a consistent and predictable interface for frontend communication and external integrations.
6) The backend must follow a layered architecture with clear separation of concerns, dividing responsibilities among different modules, such as those handling database operations, data models, and controllers. This modular approach will enhance maintainability, scalability, and the ability to manage complexity as the system evolves.
7) The system must ensure data integrity and consistency, particularly when handling concurrent requests for data storage and retrieval, by implementing appropriate transaction management and locking mechanisms.
8) The backend must support robust error handling and logging mechanisms to capture, diagnose, and respond

[2]https://docs.github.com/en/actions
[3]https://pages.github.com/

to issues in a way that minimizes downtime and ensures reliable operation of the system.

9) The backend should include automated testing frameworks to ensure that all modules, including API endpoints and data processing logic, are thoroughly tested and validated before deployment.

IV. Approach

A. System's architecture

The architecture of our 5G metric dashboard web application is a modular, containerized system designed to efficiently manage and display both historical and live 5G metric data. The system comprises five key components:

Frontend: Serves as the user interface, allowing users to interact with and visualize the 5G data.

Backend: Acts as the central hub for data processing, managing requests between the frontend, scraper and the database.

Database: Stores both historical and live telemetry data, providing a persistent data repository.

Nginx: Functions as a reverse proxy, directing traffic efficiently and ensuring secure access to the application.

Scraper: Collects the 5G telemetry data contained in the WebSocket messages generated from an external OAIBOX instance and feeds it into the database.

Each component, except for the Backend and Database which are combined into a single container, operates within its own Docker[4] container, ensuring isolation and consistency across different environments. Docker offers several advantages, including flexibility, which allows each component to be developed in the most suitable programming language without constraints from the other system's modules. Additionally, Docker simplifies local deployment and speeds up integration, making it easier to set up or modify the entire application stack. The containerized nature of the dashboard application also facilitates future integration with advanced container orchestration tools like Kubernetes[5] or OpenShift[6], laying the groundwork for enhanced security and scalability. Docker Compose is used to orchestrate the four component containers, streamlining deployment and management while ensuring the system remains scalable, maintainable, and capable of secure data handling.

Fig. 1 shows the overall architecture of our system.

B. Frontend

The frontend of the 5G metric dashboard is a dynamic and modular interface developed using React[7] and Vite[8], to facilitate fast development and efficient builds. The project folder is organized into a maintainable structure, with key directories including `src`, `public`, `styles`, and `utils`. Within the `src` directory, the `components` folder is the core

[4]https://www.docker.com
[5]https://kubernetes.io
[6]https://openshift.com
[7]https://react.dev
[8]https://vitejs.dev

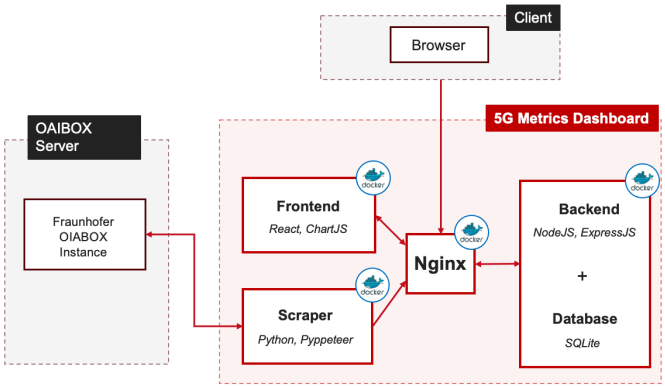

Fig. 1. System architecture of the 5G metric dashboard

of the frontend, housing 27 reusable components that follow React's modular design principles (Req. B2). In addition, the `public` folder contains static assets, and the `styles` directory houses global and shared styling resources, including `graph-colors.js` for defining color schemes used in charts and `variables.css` for common CSS variables and design tokens. The `utils` folder includes utility modules like `constants.js`, `fetching.js` for managing data retrieval, and `transform-data.js` for processing and transforming data.

Each component folder contains a `.jsx` file for logic and a corresponding `.css` file for styling. This approach ensures a clear separation of concerns and promotes reusability (Req. B3). Furthermore, the user interface is built around four main components:

Tabs Component: Allows users to choose between three different types of data to display: (i) cn5g module health data, (ii) gNB telemetry data, and (iii) gNB log data.

Forms Component: Provides tools for users to apply filters, including dropdowns, input fields, calendars, and switch toggles, allowing for tailored data selection (Req. B5).

Information Display Component: Displays the selected filters and handles various messages such as errors, informational updates, and warnings, ensuring users are well-informed.

Content Component: Utilizes the Chart.js[9] library to present the queried data, with line charts and other graphical representations to implement the data visualization (Req. B4).

To enhance user interaction, the application features a comprehensive message-rendering system that categorizes messages based on their purpose and urgency. These messages are visually distinguished by icons and color schemes that correspond to the message type. Warning messages, denoted by an exclamation icon, inform users of potential issues that require attention but are not critical. Error messages, marked by a red alert icon, signify more serious problems that need immediate resolution (e.g., data fetch errors). Informational messages provide users with general updates or notifications and are represented with an information icon. Additionally,

[9]https://www.chartjs.org

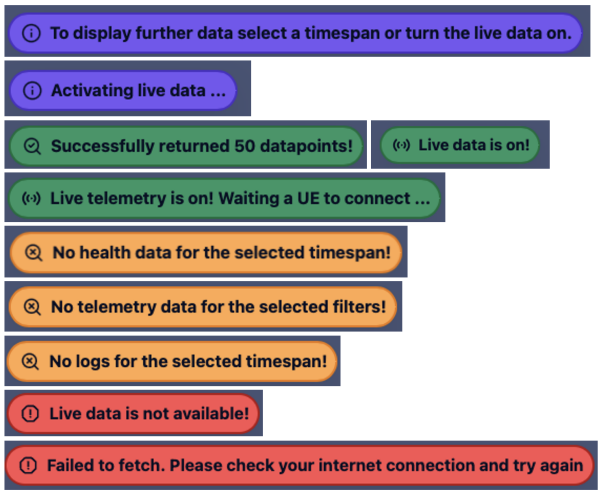

Fig. 2. Application UI messages

success messages are further divided into three types: ”live data” success, indicated by an online status icon, confirming that real-time data is being displayed; ”queried data found” success, represented by a checkmark icon, confirming that the requested data was retrieved; and ”queried data not found” success, marked by a cross icon, notifying users that the query was executed but no matching data was found. Fig. 2 given an overview of the UI messages.

The frontend can render both stored 5G metric data and live data fetched in real-time from the OIABOX dashboard, ensuring that users have access to up-to-date and historical telemetry information (Req. B1). This dual capability is crucial for providing comprehensive insights and a responsive user experience. The integration of React's component-based architecture and Vite's rapid build capabilities contributes to a high-performance and interactive interface, making the frontend a powerful tool for monitoring and analyzing the scraped 5G telemetry data. Lastly, the frontend is designed to be compatible with all major web browsers, including Chrome, Firefox, Safari, and Edge (Req. B6).

An overview of the application's frontend, along with demo videos showcasing its use, can be found in our GitHub repository.

C. Scraper

To meet the requirement A.1 defined in section III, we implemented a scraper service. This service ensures continuous and reliable data collection from the OAIBOX instance. The scraper component is a Docker container designed to automate data extraction from the OAIBOX dashboard. Utilizing a headless browser, the scraper simulates user interactions by logging into the dashboard, capturing incoming WebSocket messages, processing the data, and forwarding it to the backend service. The scraper is implemented using Python, leveraging the Pyppeteer[10] package; an unofficial variant of

[10]https://github.com/pyppeteer

Method	Path	Query Parameters	Description
GET	/api/messages	topic, timeStart?, timeEnd?, limit?	Query all data for a given topic and optional timespan
POST	/api/messages	-	Add a metrics datapoint
GET	/api/messages/telemetry	timeStart?, timeEnd?, uelds?, limit?	Query telemetry data for given timespan and given list UE devices
GET	/api/messages/telemetry/ues	timeStart?, timeEnd?	Query list of available Ues for a given time-span
GET	/api/messages/latest	topic	Query latest datapoint for a specified topic
GET	/api/docs	-	Swagger UI

Fig. 3. Backend API

the JavaScript library Puppeteer[11], which facilitates headless Chromium browser automation. For security, credentials required for accessing the OIABOX dashboard are stored in a local `.env.local` file within the project folder, allowing the codebase to be shared on GitHub without exposing sensitive information. The Python script is asynchronous, designed to handle multiple I/O-bound operations concurrently, including launching the browser, navigating web pages, intercepting WebSocket frames, and transmitting data to the backend. This approach enhances resource utilization and execution speed, ensuring efficient and uninterrupted data capture.

D. Nginx

To add another layer of security and handle CORS Errors more easily the traffic in our application is handled by an Nginx[12] container. Each request is sent to Nginx first and then forwarded to the correct service. CORS errors occur when a web application tries to access resources from a different origin without the correct permissions, resulting in blocked requests. Nginx helps by adding the necessary CORS headers (like Access-Control-Allow-Origin and Access-Control-Allow-Methods) to responses, ensuring that cross-origin requests are allowed and handled properly. Moreover, adding Nginx to the architecture enhances security by providing a single public entry point for your services, reducing the attack surface. Additionally the internal structure of the application is hidden, making it harder for attackers to target the application directly. Finally, Nginx implements access control headers to restrict unauthorized access. This centralized point of control makes it easier to manage security policies and monitor traffic effectively.

E. Backend

The backend of the 5G metric dashboard is a robust and modular system responsible for managing the flow of data between the frontend, database, and external data sources. It is designed to handle the complex requirements of persisting,

[11]https://pptr.dev
[12]https://nginx.org

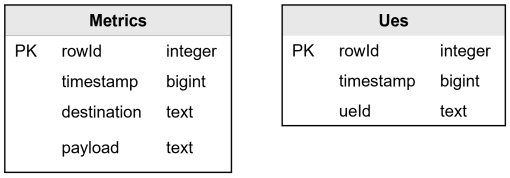

Fig. 4. Data Tables

processing, and serving large volumes of 5G telemetry data efficiently.

The backend is implemented using Node.js[13], leveraging Express.js[14] for its web framework to build a RESTful API that adheres to standard REST principles (Req. C5). This API serves as the primary interface for communication between the frontend and the database, as well as external systems such as the OAIBOX instances from which telemetry data is collected. Key functional aspects of the backend include:

Data Persistence and Querying: The backend is designed to handle 5G metrics of varying structures, ensuring flexibility in data storage (Req. C1). Metrics are stored in a SQL database, with support for filtering data based on time ranges to enable efficient querying and analysis (Req. C2). The backend employs optimized queries to minimize the data load during retrieval, selectively fetching only the necessary data points to reduce bandwidth and processing time (Req. C3).

Modular Architecture and Extensibility: Following the principle of separation of concerns, the backend is organized into distinct modules, each responsible for specific functionalities such as data models, database operations, and API controllers (Req. C6). This modular structure ensures that the system is maintainable and scalable, allowing for the straightforward addition of new metric types or features without disrupting existing functionality (Req. C4).

Database Management: The backend interacts with a SQLite3 database, chosen for its simplicity and efficiency in managing telemetry data. The use of SQL allows for complex querying capabilities, ensuring that data retrieval is both flexible and performant.

To optimize the data management process, we have refactored the data model and now maintain only two primary tables, as shown in fig. 4. Metric records, which follow a uniform structure, are stored in the **Metrics** table. This table consists of three key fields: a `timestamp`, `destination`, and `payload`. The `destination` field implies the structure of the associated `payloads`, allowing for efficient querying of data over a given time span.

For performance reasons, we introduced a separate **UEs** table, which is particularly relevant for data associated with the `gnb.telemetry` topic. This table stores all `ueIds` along with their corresponding `timestamps`, for which data has been collected. The **UEs** table enables efficient population of dropdown menus in the frontend without the need to iterate over all metric records, thus enhancing the overall performance and user experience of the dashboard.

Data Reduction Strategy: To handle large volumes of telemetry data efficiently, the backend employs a data reduction strategy based on a modulo procedure. A `limit` parameter can be specified to determine the maximum number of records to be returned. During selection, a step size is determined by the modulo divisor, allowing every n-th data record to be selected. This divisor is calculated by dividing the total number of available records by the specified limit.

While this reduction strategy is straightforward, more sophisticated approaches, such as partitioning data into time intervals and calculating averages for metric data, were considered. However, due to time constraints within the project, these were not implemented. The chosen method serves its purpose effectively and is largely independent of the underlying data structure. Depending on the size of the original dataset, the actual number of records returned by this method will always be between 50% and 100% of the specified limit, ensuring a balanced trade-off between data volume and detail.

API Implementation and Security: The RESTful API is designed to be stateless and idempotent, following best practices to ensure that the API is easy to use and integrates well with the frontend. Security is a key concern, with measures such as input validation, error handling, and logging implemented to safeguard the system against common vulnerabilities.

Error Handling and Logging: The backend includes comprehensive error handling mechanisms to manage and log errors effectively, ensuring that issues can be diagnosed and resolved quickly (Req. C8). This approach minimizes downtime and ensures that the system remains reliable, even under load.

Testing and Continuous Integration: To ensure the reliability of the backend, automated testing frameworks are integrated into the development process. Unit tests and integration tests are employed to validate the functionality of each module and API endpoint before deployment (Req. C9).

The backend's design and implementation reflect a careful balance between flexibility, performance, and maintainability, ensuring that it can effectively support the real-time and historical data processing needs of the 5G metric dashboard application. An overview of the API endpoints and their description can be found in fig. 3.

V. Evaluation

A. Automated Testing

To ensure the robustness, reliability, and maintainability of the backend module, an automated testing suite was implemented using Jest[15], a popular testing framework for Node.js. This suite comprehensively covers the core functionalities of the module, including data insertion, retrieval, error handling, and edge cases. The key aspects evaluated through the testing suite include:

[13]https://nodejs.org

[14]https://expressjs.com

[15]https://jestjs.io

Data Integrity: Tests validate that data is correctly inserted into and retrieved from the database, ensuring that the module handles both typical and boundary conditions (e.g., empty datasets, large datasets) effectively.

Error Handling: The suite tests various error scenarios, such as invalid input parameters, database connection issues, and malformed data. This ensures that the module not only catches errors but also responds with informative messages, which is crucial for debugging and maintenance.

B. Continuous Integration Workflow

To streamline development and maintain code quality, a CI/CD pipeline was established using GitHub Actions. The pipeline is triggered automatically upon any push to the `dev`, `backend`, or `main` branches, ensuring that the latest changes are thoroughly tested before they are merged or deployed. The CI/CD workflow includes the following steps:

Code Checkout: The pipeline begins by checking out the latest code from the repository to ensure that all dependencies and configurations are up to date.

Dependency Installation: All necessary dependencies are installed in a clean environment to prevent issues related to missing or outdated packages.

Test Execution: The Jest test suite is executed in its entirety, with the results being recorded. If any test fails, the pipeline is halted, and the issue must be resolved before the code can be merged.

Reporting: Test results and any errors or warnings are logged, providing detailed feedback that can be used to quickly identify and fix issues.

An overview of exemplary CI/CD workflow runs can be found in fig. 5.

Next to the testing jobs, the pipeline also deploys an OpenAPI[16] specification using GitHub Pages. The OpenAPI specification is a standard for documenting APIs. It shows the different endpoints and documents them thoroughly. Moreover, when running the Docker containers, it is possible to execute from the Swagger UI requests against the documented endpoints. The deployed specification can be found at this link: https://mrettore.github.io/awt-pj-ss24-fullstack-web-based-5G-metrics-dashboard-2/.

C. Results

The integration of automated testing with a continuous integration pipeline has significantly improved the development workflow. It provides early detection of bugs, enforces code quality, and ensures that new changes do not introduce regressions. As a result, the backend module is more reliable and easier to maintain, with a strong assurance of its correctness and performance across different environments.

This approach has also facilitated collaboration among team members, as it ensures that only well-tested code is merged into key branches, reducing the likelihood of issues in production and improving overall project quality.

[16]https://www.openapis.org/

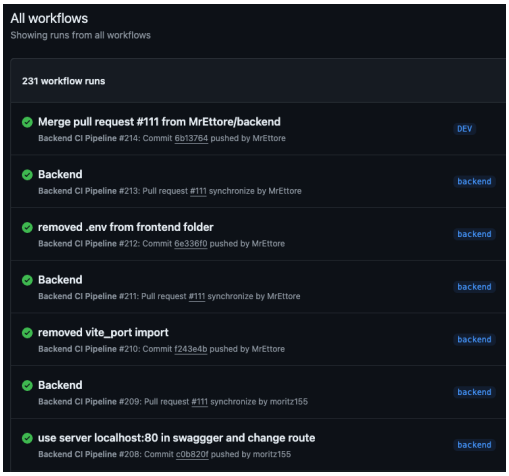

Fig. 5. CI/CD Workflow Runs

VI. Conclusion

The development of the 5G metrics dashboard marks a significant achievement in enhancing the ability to monitor both live and historical data from 5G networks. The project set out to address the shortcomings of existing tools by creating a comprehensive, modular, and containerized web application that offers robust data visualization and storage solutions.

By integrating a React-based frontend, a Node.js backend, a SQL database, and a Python-based scraper, the system successfully provides real-time monitoring while ensuring that historical data is stored for future analysis. The decision to use Docker containers for each system component has been crucial in maintaining consistency across different environments, simplifying deployment, and preparing the system for future scalability with advanced orchestration tools. The evaluation process, which included automated testing and the implementation of a CI/CD pipeline, has significantly contributed to the system's reliability and maintainability. The pipeline has streamlined the development process, allowing for early detection of issues and ensuring that only thoroughly tested code is integrated into the system.

Overall, the 5G metrics dashboard meets its goals by providing a user-friendly, scalable, and reliable solution for analyzing 5G network performance. It serves as a powerful tool for network engineers, enabling deeper insights into the behavior and efficiency of 5G networks. In conclusion, this project demonstrates not only the feasibility of capturing and visualizing 5G metrics in real-time but also establishes a solid foundation for continued innovation in network performance monitoring and testing.

Repository

The source code for our 5G metric dashboard application as well as its documentation can be found at: https://github.com/MrEttore/awt-pj-ss24-fullstack-web-based-5G-metrics-dashboard-2.

REFERENCES

[1] N. T. Le, M. A. Hossain, A. Islam, D.-y. Kim, Y.-J. Choi, and Y. M. Jang, "Survey of Promising Technologies for 5G Networks," *Mobile Information Systems*, vol. 2016, 2016.

[2] H. Rahimi, A. Zibaeenejad, and A. A. Safavi, "A Novel IoT Architecture based on 5G-IoT and Next Generation Technologies," in *2018 IEEE 9th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, Nov. 2018.

[3] E. NetWorld2020 *et al.*, "5g: Challenges, research priorities, and recommendations," *Joint White Paper September*, 2014.

[4] OAIBOX, "Oaibox 5g lab manual," 2023, accessed: 2024-08-15. [Online]. Available: https://api.oaibox.com/downloads/OAIBOX5GLabManual.pdf