

**ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA**  
**SCUOLA DI INGEGNERIA E ARCHITETTURA (SEDE DI**  
**BOLOGNA)**

**Anno Accademico 2021/2022**

---

**END-OF-CURRICULAR INTERNSHIP REPORT /**  
**THESIS INTERNSHIP REPORT**

Conducted by the student:

*Jacopo Jop - Matriculation: 00873291*

Enrolled in the Degree Program in

*Computer Engineering L - 9254*

At:

*Mumble s.r.l.*

On the following topic:

***Frontend Development in React.js of an Application for Website  
Performance Analysis***

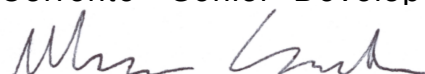
**Student:**

\_\_\_\_\_  
(firma)

**Academic Tutor:**  
Prof. (Paolo Bellavista)

\_\_\_\_\_  
(Signature for approval of the final report)

**Host Organization Representative:**  
Marco Corrente - Senior Developer



# 1. Table of Contents

|                                    |    |
|------------------------------------|----|
| 1. Table of Contents               | 2  |
| 2. Introduction                    | 3  |
| • Technologies Used and Mastered   | 3  |
| 3. Activities                      | 5  |
| • Training                         | 5  |
| • PageSpeed Dashboard              | 6  |
| • Backend - Frontend Communication | 7  |
| • Laravel Sanctum                  | 8  |
| • Graphics                         | 9  |
| • PageSpeed Insights API           | 10 |
| • Performance History              | 14 |
| • Terraform and Deploy on AWS      | 15 |
| • Performance                      | 16 |
| 4. Conclusions                     | 18 |
| • Issues and Improvements          | 18 |
| • Strengths                        | 19 |
| 5. Bibliography                    | 20 |

## 2. Introduction

The internship was carried out at Mumble S.R.L., a company based in Modena.

The company focuses on developing cloud-based web and mobile applications for clients from highly diverse sectors, with 40% of them being international. In addition, it develops internal solutions to improve workflow and productivity.

Mumble currently consists of nearly twenty young professionals and is continuously growing under the leadership of its three young founders, Mattia Farina, Giacomo Torricelli, and Francesco Vellani.

In this highly collaborative environment, I was able to move freely, being integrated into every aspect, from a company email and Slack account for sharing materials, to meetings with employees and group activities, and even being invited to official product presentations. Most importantly, whenever I had doubts, I could easily turn to various programmers and engineers who were willing to help and teach me.

The goal of the internship was always to develop a web application in React.js through to its release, but initially, the project's form was not clear. Only after the first 15 days of training was the activity to be performed defined. The technologies to be used were also decided progressively as development advanced.

### • Technologies Used and Mastered

During the initial period of the internship, which could be defined as training, the technologies used were limited to Visual Studio Code as the editor for writing small JavaScript applications based on the React.js library and Node.js. Some of these were maintained in repositories on Git to experiment with deployment on the Google Firebase platform or to implement a small database in MongoDB. During the training period, the Next.js framework and deployment on Vercel, the platform created by the developers of Next.js, were marginally addressed.

When I started the main project of this internship, PageSpeed Dashboard, I was paired with another intern with whom I divided the tasks. To facilitate collaboration, the company created corporate accounts for us on GitLab and, most importantly, provided a starter container on Docker to work in a shared environment and resolve

compatibility issues, as the other intern used a Windows PC while I used a Mac.

The project's frontend is built in native React, while the backend, developed by my colleague, is made with Laravel, a PHP framework. The entire application relies on a MySQL database. To make requests to the backend, I used the Axios library; for authentication and session management, we relied on Laravel's Sanctum API for SPAs (Single Page Applications); for the graphical aspect, I leveraged components from the Material UI library and Tailwind classes, optimizing stylesheets according to the BEM (Block Element Modifier) methodology and using the SASS preprocessor to make the code more readable.

The entire project revolves around the implementation of PageSpeed Insights, a Google API for measuring the performance of a web page.

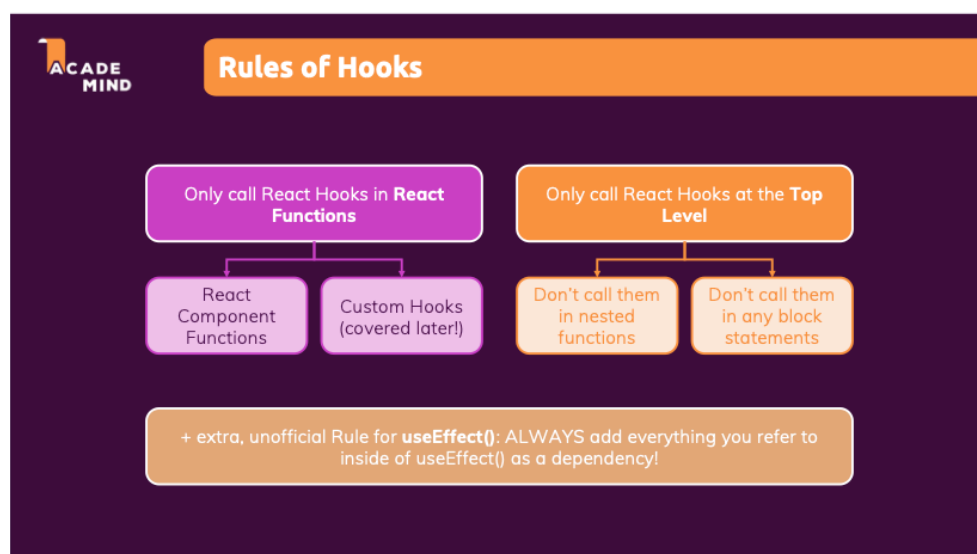
The release was automated using a Terraform script to deploy on AWS EC2 instances.

### 3. Activities

- **Training**

The first two weeks of the internship were dedicated to training, particularly to learning development in React.js. For this purpose, I was assigned a nearly 50-hour video course enriched with various challenges and exercises, conducted on the Udemy platform. Thanks to the course, I significantly improved my frontend development skills and was introduced to various Hooks and React routing, which I utilized in the project and which helped me better understand the potential of this library. Hooks were introduced in React starting from version 16.8 to add new functionalities to the new “Functional components,” which have thus replaced “Class components,” as classes cannot use Hooks.

The Hooks I used most extensively were: “useState” to track the state of components; “useRef” to access and store variable values of text fields that do not trigger a re-render when modified; “useEffect” to perform side effects, such as data fetching, when the event specified as a dependency occurs; “useContext” to maintain the state of certain values globally, often for user authentication information used in routing, allowing the hiding of accessible pages based on the login status. For routing, from the “react-router-dom” library, I used the Hooks “useNavigate” to redirect the user and “useLocation” to retrieve information about the site’s position using the URL.



Schema riassuntivo delle regole degli Hook presentato nel corso.

The course also covered cross-cutting topics such as deployment on the Google Firebase development platform, integration with MongoDB for database creation, and data fetching.

Finally, an introduction to Next.js was provided, which I found very interesting. Next.js is a framework developed by Vercel that is based on React but introduces a significant difference. In the case of a native React application, the rendering of static websites occurs directly on the client, so performance is influenced by the computing power of the user's machine. Next.js, on the other hand, enables server-side rendering. This topic will be explored in depth in the thesis.

## • PageSpeed Dashboard

Once I gained some familiarity with React, the tutor Marco Corrente and the company's CTO paired me with the intern Matteo Saetti, with whom I collaborated from that point onward. With them, the project specifications were defined, and roles were assigned. We were also taught the company's convention for formatting git commits to make them easily interpretable by other programmers.

According to the specifications, we were required to develop an application to evaluate the performance of any web page by simply entering its URL on the platform. Specifically, registered users can create projects, with each project consisting of tests. These are simply the results returned by Google's PageSpeed Insights (PSI) API, cataloged and stored in the database. The tests must be accessible through a table displayed on the project screen and are characterized by a title, the analyzed URL, the generation date, and the ability to edit the title or delete them. When viewing a test, the stored results are presented intuitively.

To make the application more interesting and useful, I proposed during development to add the following functionality: if a project contains multiple tests performed on the same URL, it is possible to visit a dedicated page to view a line graph, allowing visualization of performance evolution over time. In this version, the application is potentially useful to the company's developers to assess whether changes made to a project have improved or worsened its performance, providing an overall view of the trend.

## • Backend - Frontend Communication

Once the development environment was set up, we began working on Login and Registration.

Registration obviously involves saving the new user in the MySQL database. In this case, the user is characterized by a username, an email that serves as the unique field in the database associated with projects and tests, and an alphanumeric password with a minimum of 8 characters.

To interface with the database, my colleague created APIs with the necessary fields for each request or response. He handled the backend, including database queries and checks, such as user uniqueness or reporting error messages. My role was to invoke the correct API and provide properly formatted parameters to request processed data from the backend or send data to be saved in the database. In the case of registering a new user, the frontend is responsible for creating a User object containing the correct attributes to send. If the values are inadequate, Laravel's checks communicate the error messages to be displayed on the screen.

Specifically, communication between frontend and backend occurs through Axios. Axios is a promise-based HTTP client for Node.js and browsers, with its main functionalities being:

- Making XMLHttpRequests from the browser
- Making HTTP requests from Node.js
- Intercepting requests and responses
- Automatically transforming data into JSON

The tutor recommended this client as it is commonly used by the company.

```
axios.get(`/api/project`, { params: { id: projectId } },
  {
    headers: {
      // Overwrite Axios's automatically set Content-Type
      "Content-Type": "application/json",
    },
  })
  .then((res) => {
    if (res.status === 200) {
      setProject(res.data.project[0]);
      if (res.data.project.length === 0) {
        navigate("/not-exist");
      }
      return res;
    }
  })
  .catch((err) => {
    alert(err.message);
  });
```

In this example of a GET request, the “project” API is invoked, passing the ID of the project to be retrieved as a parameter. If a positive response is received, the frontend logic is applied; otherwise, in case of an error, it is communicated to the user.

This logic is the same for all GET or POST requests sent to the database.

However, it should be clarified that React does not communicate directly with the database; rather, the React frontend application communicates with the Laravel backend application, which in turn manages the database.

Axios is also used to control the validity of requests through its interceptor functionality. Specifically, a function was created, wrapping the entire site, to intercept error states or messages to communicate them to the user or redirect them to specific pages. An example is the case of a user with an expired session attempting to view a page they no longer have access to, resulting in a 401 or 419 error; Axios intercepts and recognizes the error, performing a logout to reset the browser’s local memory and stored values in the context, then redirecting the user to the login page.

## • **Laravel Sanctum**

For user authentication and session management, we chose to use Laravel Sanctum for SPAs, which uses Laravel’s internal cookies based on its web authentication guard, ensuring protection against CSRF (Cross-Site Request Forgery). Sanctum attempts to authenticate the requesting user using cookies only if the request originates from the frontend of its SPA.

Specifically, upon login, the “laravel\_session” and “XSRF-TOKEN” cookies are generated and sent to the middleware where the API access Routes are protected. This way, only a valid session allows the exchange of sensitive data with the database. For those attempting to access pages directly without logging in, Sanctum returns the “401 (Unauthorized)” error, which is intercepted by Axios.

With this system, it is possible to set the automatic session expiration time, which returns the differentiated “419” error, allowing the user to be informed that their session has expired before forcing a logout.



## • Graphics

A fundamental part of the frontend is the graphical aspect. The design of this project was inspired by a project created by other interns at the company, for which they shared the Figma model (a vector graphics editor specific for web design) to better understand the components and obtain the company logos in SVG format.

In this case, the technologies used to modify the style of React components are primarily four. First, components from the Material UI library, such as tables, buttons, text fields, or date pickers, as they already have a modern and nearly final style for our case. I was asked by the company to use this library to test its usability, as it is not their usual library, and they typically use components from an internally developed library. In my experience, the MUI library is very restrictive in terms of design, as modifying a component's style requires altering the already applied classes (which must first be identified), only to realize that only some modifications are possible. For example, to change a button's color, it is necessary to create a Theme with a color palette to import on every page and apply to each button, resulting in a lot of unoptimized code when only one element needs modification. I did not encounter this complexity with other libraries or frameworks used before this experience.

For the rest, additional styling is managed by classes from the Tailwind framework to produce less CSS code. Thanks to Tailwind's integration, changes are almost immediate to write, and it allowed me to give the project a responsive structure to automatically adapt to smartphones, tablets, or desktops.

Initially, each React component had its own stylesheet to allow testing the appearance of a page without requiring the rest of the site to be functional. However, this led to unnecessary repetitions of classes, often very similar across different text fields, tables, or buttons.

The stylesheets produced for individual components were then consolidated into a single global SCSS stylesheet, using the SASS preprocessor to simplify the code and make it less redundant. Thanks to SASS, unique classes could be written for similar components while maintaining their small differences, eliminating all repetitions.

Finally, to improve the readability of the HTML code and the accessibility of classes, the global stylesheet was adapted to the BEM nomenclature to align with the company's practices. Following this nomenclature, each class is characterized by the Block it belongs to (e.g., .result), the child Elements of the block (e.g., .result\_\_score), and Modifiers (e.g., .result\_\_score—info).

Thanks to the integration of BEM and SASS, when reading the classes in HTML pages, it is immediately clear what changes they entail.

```
// Example of a class and its children using Tailwind classes
// Formatted with BEM
.result {
  &__score {
    @apply flex flex-col md:flex-row -mx-3 overflow-hidden;
    &-graph {@apply my-3 px-3 overflow-hidden md:w-2/5;}
    &-info {@apply my-3 px-3 w-full overflow-hidden md:w-3/5;}
  }

  &__title {
    @apply text-gray-900 mb-2;
    &-audit {@apply text-5xl;}
    &-performance {@apply text-3xl md:text-5xl;}
  }

  &__subtitle {
    @apply text-gray-500 w-64 sm:w-10/12 truncate;
    &-audit {@apply text-3xl mb-16;}
    &-performance {@apply text-2xl md:text-3xl mb-8;}
  }

  &__performance {
    @apply text-3xl text-gray-900 mb-8 mt-16;
  }
}

// Usage of the BEM-formatted class in HTML code
<div className="result__title result__title-audit">{info.title}</div>
<div className="result__subtitle result__subtitle-audit">{info.url}</div>
<div className="result__performance">Performance Score</div>
<div className="result__score">
  <div className="result__score-graph">...</div>
</div>
```

## • PageSpeed Insights API

The core of this project is the integration of Google's PageSpeed Insights API. Its operation involves analyzing the page at the URL passed as a parameter and, after a few seconds, responding with a JSON object containing the results of the dozens of tests it performs. Our task was to identify the most relevant entries according to the company, filter them, and save them in the database. The integration with PSI is entirely handled in the backend, where the URL sent via Axios is received, its formatting is verified, and then a GET request is made to PSI.

The response is a highly articulated JSON object, and we limited ourselves to analyzing the audits in the "lighthouseResult" branch, as Lighthouse is Google Chrome's official extension for website evaluation. Out of the 59 audits, we selected 6 in addition to the overall performance score, namely:

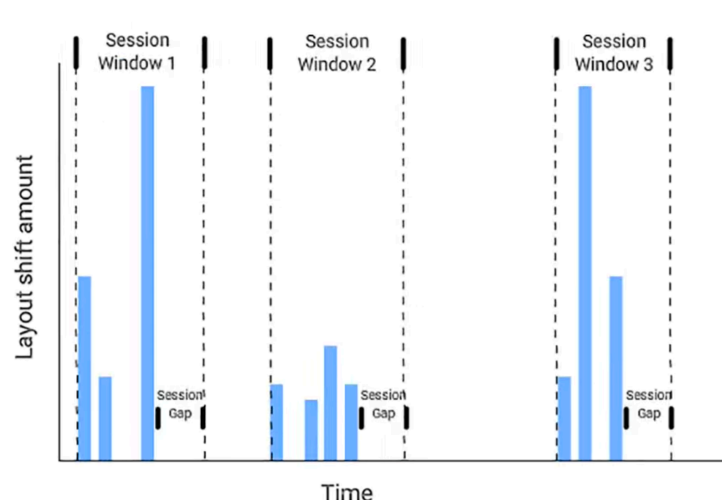
- **First Contentful Paint (FCP):** Indicates how long it takes to render the first DOM element from when the page is accessed. A good score is under 1.8 seconds, a poor score is over 3 seconds.

- **Speed Index (SI):** Measures how quickly the page's content becomes visible during loading, literally recording a video analyzed frame by frame. A good score is under 3.4 seconds, a poor score is over 5.8 seconds.

- **Largest Contentful Paint (LCP):** Indicates when the heaviest page element becomes visible to the user. A good score is under 2.5 seconds, a poor score is over 4 seconds.

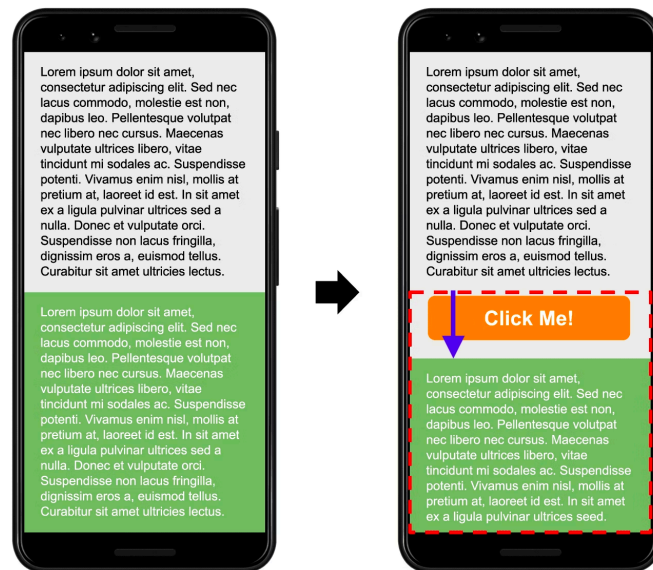
- **Time to Interactive (TTI):** Indicates the time required for the page to become fully interactive. By interactive, it means: content has been rendered, most event handlers have been registered, and the page responds to user interaction in less than 50 ms. A good score is under 3.8 seconds, a poor score is over 7.3 seconds.

- **Cumulative Layout Shift (CLS):** A measure derived from the sequence of unexpected layout shifts that occur during the entire lifecycle of a page. A layout shift occurs each time a visible element changes its position from one rendered frame to the next. A sequence of layout shifts, called a session window, occurs when one or more individual layout shifts happen in rapid succession with less than 1 second between each shift and a maximum of 5 seconds for the total window duration. The largest sequence is the session window with the highest cumulative score of all layout shifts within that window.



Esempio di finestre di sessione. Le barre blu rappresentano i punteggi di ogni singolo turno di layout.

The layout shift score is the product of the “impact fraction” and “distance fraction,” i.e., the fraction of the screen area filled after a shift (from 0% to 100%) and how far the elements have moved (from 0% to 100%). In essence, Cumulative Layout Shift is an index of the annoying movement of on-screen elements, very common in sites with many advertisements. A good score is under 0.1, a poor score is over 0.25.



Esempio di uno spostamento del layout

- **Total Blocking Time (TBT):** Measures the total amount of time a page is blocked and unresponsive to user inputs, such as mouse clicks, screen taps, or keyboard presses. The value is calculated by summing the blocking portion of all long tasks between First Contentful Paint and Time to Interactive. Any task that executes for more than 50 ms is a long task. The time after 50 ms is the blocking portion. For example, if a long task of 70 ms is detected, the blocking portion will be 20 ms. A good score is under 200 ms, a poor score is over 600 ms.

- **Performance Score:** Simply the weighted average of the six previous scores, with percentages: FCP 10%, SI 10%, LCP 25%, CLS 25%, TBT 30%. A good score is above 90, a poor score is below 49.

# Amazon

<https://amazon.com>

## Performance Score



**i** More details about performances.

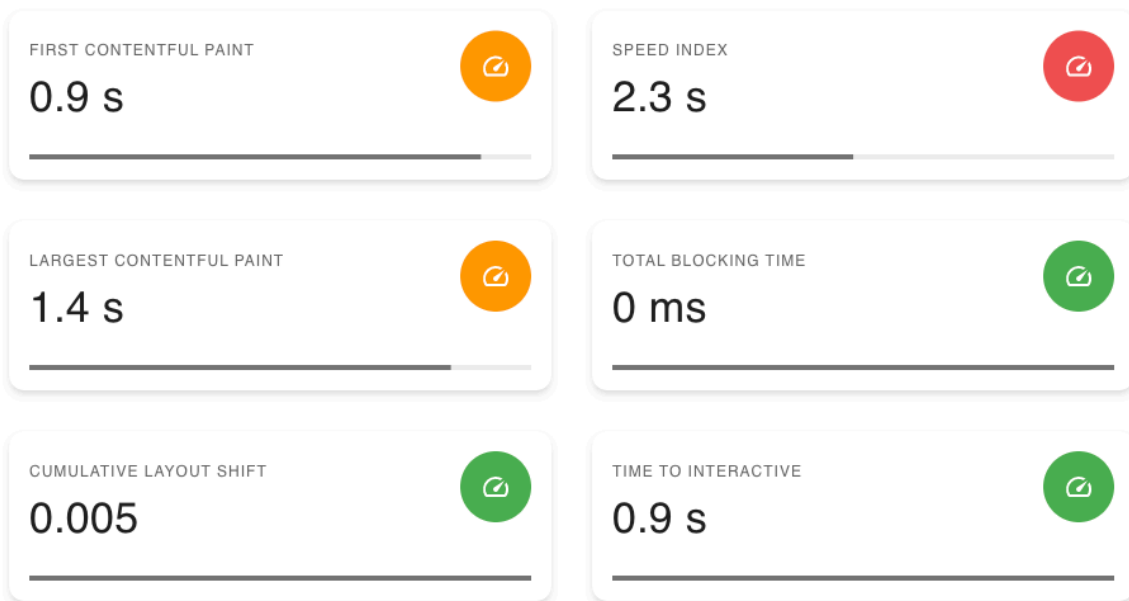


The Performance score is a weighted average of the metric scores.

Naturally, more heavily weighted metrics have a bigger effect on your overall Performance score.

The weightings are chosen to provide a balanced representation of the user's perception of performance. The weightings have changed over time

## Metrics



Risultati di un test al sito [amazon.com](https://amazon.com) dove vengono riportati i valori sopra descritti.

## • Performance History

Up to this point, we essentially created a version of Google Lighthouse with persistence, which is already more useful to a programmer who does not need to remember or note down results. However, what is missing is the overall perspective, as it is necessary to open multiple tests to understand the evolution of the project being analyzed. For this reason, I decided to add the Performance History section.

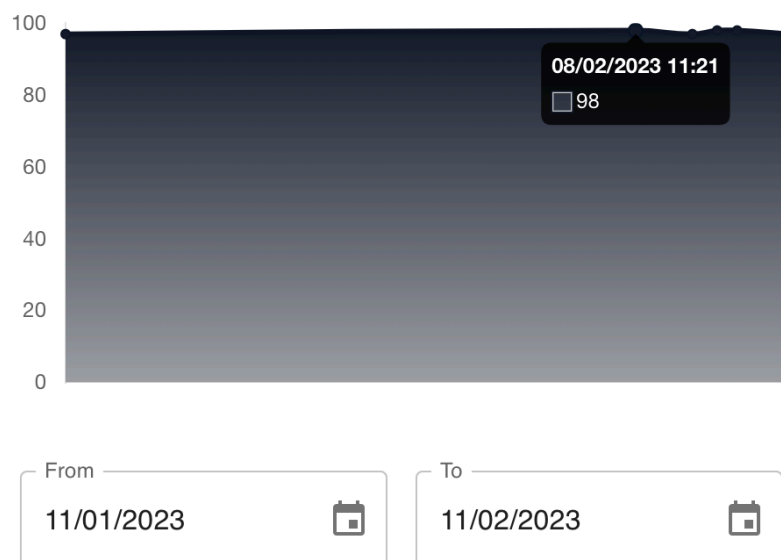
On this page, the values of each Performance Score are plotted on a line and point graph, accompanied by the creation date and time. Based on the date, it is possible to adjust the time interval of interest using two Date Pickers.

Since the test results in the database are linked to their respective URL, the frontend only needs to specify the URL and time interval to the backend, where the tests are filtered, and it is verified that the owner user and the authenticated user are the same, as multiple users could perform tests on the same URL.

The graph is based on the Chart.js and react-chartjs-2 libraries.

### Performance Scores

<https://tirocini.unibo.it>



Esempio di storico dei test al sito [tirocini.unibo.it](https://tirocini.unibo.it)  
Il punteggio è quasi costante poiché non sono state apportate modifiche alla piattaforma nel periodo di testing.

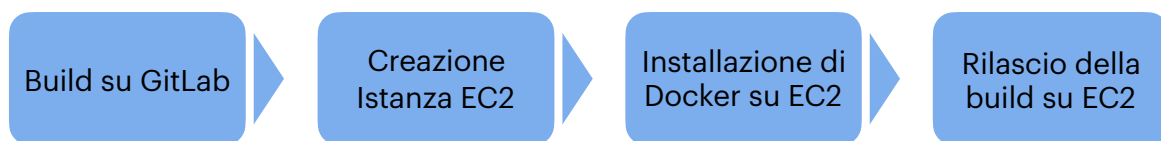
## • Terraform and Deploy on AWS

Once all functionalities were implemented and the graphics refined, we focused on releasing the application on a server. To allow us to experiment with a semi-professional and advanced release, the company instructed us to use Terraform for its automation and AWS as the web server.

In this case as well, we were provided with a two-hour video course to at least introduce Terraform. After that, using specially created company accounts, we began experimenting with AWS and EC2 (Elastic Compute Cloud) instances. Here, we identified the free “t2.micro” instance as the suitable option for our use. An instance is a virtual server in the cloud, and this one, in particular, features a single vCPU and 1GB of RAM running Linux.

Starting from a sample Terraform script, the necessary changes were made to configure the desired instance, including Frankfurt, Germany, servers as the region and zone since they are the closest to us, public and private subnets for network configuration, the SSH access key pair for the instance remotely, and the instance type as t2.micro. Finally, we included a “docker\_registry” variable indicating the link to the repository where the project build is compiled.

Once the instance is launched with this configuration, a bash script is executed to install Docker, log in, pull the build from GitLab, and run it. This way, whenever a change is made to the project, the “terraform destroy” command is sufficient to destroy the existing instance, and “terraform apply” executes our script to generate the new build, create an instance, and perform the release.



Flusso dello script Terraform

However, with each release, the instance’s URL and IP address change because a domain was not purchased to associate with the script, which is quite inconvenient since accessing the site requires logging into the AWS console to obtain the new URL each time. Due to resource and time constraints, further improvements to the release process were not pursued.

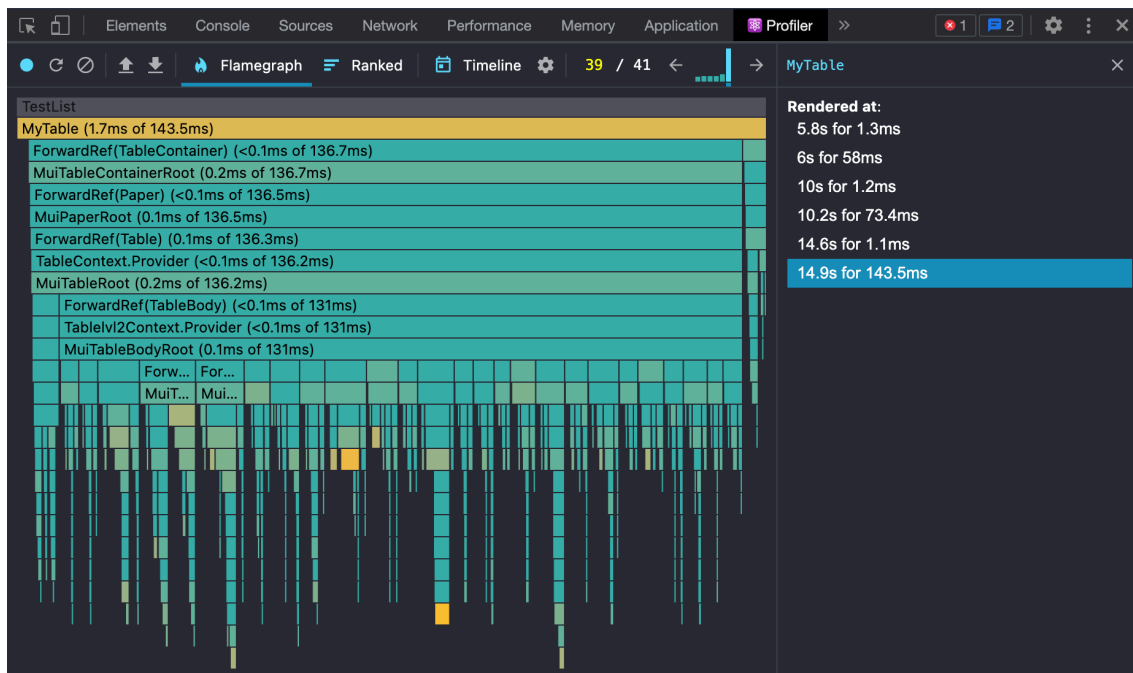
Unfortunately, after the internship ended, our AWS accounts were deleted, and company emails were closed, so I no longer have the ability to run the project except locally in the immediate term.

## • Performance

Regarding the project's performance, some measurements of the rendering times of the site's various pages were conducted using the Chrome DevTools extension found in the React.js documentation, simply called React Developer Tools. This extension integrates into the browser's developer tools, adding the Components tab, which shows the component hierarchy tree, allowing editing of their properties, and the Profiler tab, useful for recording performance information. The measurements were performed locally since, as mentioned earlier, our AWS access was revoked, and our instance was shut down.

The graphs obtained show React's characteristic behavior, where only new DOM elements are rendered, preserving everything that remains unchanged, thus saving resources during navigation on relatively complex and impactful elements such as the menu, footer, or site layout. The login page takes approximately 65 ms to render and includes almost all recurring site elements.

Individual test pages are influenced by the server response, but since the amount of data received is always the same, with minor fluctuations, loading the container with all test information takes 4.5-5.1 ms to render locally.



Scheda Profiler di React Developer Tools che mostra i tempi di caricamento di ogni componente e dei suoi figli. La parte bassa e frastagliata rappresenta il caricamento delle singole righe della tabella dei test.



The only pages where loading times may vary are those containing tables, as they could potentially require a large number of rows from the database. To avoid exchanging unnecessarily large packets with the server, we decided to limit the number of table rows to sizes of 8, 15, and 25 at the user's discretion, while managing pagination in the backend. This way, the user receives only the desired fraction and not the entire test list, reducing loading times and data traffic. However, a difference in rendering times for the three chosen sizes can be observed, confirming that requesting the entire list would be significantly slower. Running the three tests in quick succession, we see in this example that pagination at 8 takes 58 ms, at 15 takes 73.4 ms, and at 25 takes 143.5 ms, with nearly linear growth. The intermediate intervals of about 1.2 ms are the rendering of the empty table with the loading icon.

## 4. Conclusions

### • Issues and Improvements

There are many aspects of the project that can be improved and functionalities to be added to make the application effectively usable in a professional context.

The first functionality to implement is password recovery and reset. All the necessary components and views have already been created to allow the user to recover their credentials. The user must enter their email and receives a link to reset the password in their inbox. However, it was not possible to implement the “Mailgun” email delivery service, which, for reasons we could not identify, once integrated into our project, failed to authenticate with the mail server, forcing us to abandon it.

A convenient functionality for the user that was not implemented is the ability to access test results directly from the tooltips of the points in the performance history graph. Currently, the user must access tests directly from the table in the project screen, where there is also no search functionality, only the option to change the table’s sorting.

The major issues are directly related to the implementation of PageSpeed, particularly a bug that sometimes prevents performing two tests in quick succession, triggering a cycle of unsuccessful tests. Since performing a test takes several seconds and is also influenced by the internet connection, and the API does not serve more than 400 requests in 100 seconds, it could be that an error in the request triggers multiple attempts that reach the critical threshold, rendering the platform unusable for a certain period. What is unclear is what triggers this cycle of requests, as the user must wait for the previous test to complete before starting a new one, each test lasts a few tens of seconds, and the issue does not always occur. The only solution not tried is the use of an “API key” generated by PSI, which should ensure the ability to perform more queries per second.

The main limitation of our application is the lack of integration with the mobile side of Google’s API, as we limited ourselves to obtaining site values only in their desktop version. It would, however, be possible to adapt everything to perform tests on the mobile version of sites, where elements on the screen are typically rearranged and resized to ensure easy usability for the user, inevitably leading to different results.

Additionally, in this version of the project, there is no possibility for the user to modify the types of values returned by the tests, which are

only the six predefined ones. It could be useful to allow users to choose which types are relevant, selecting from all those provided by the API.

- **Strengths**

I consider the final result satisfactory nonetheless; all the initial functionalities were developed, and the specifications were met, even adding several parts that inevitably extended the timeline but, at the same time, enhanced the project's value. The format of the tests is exactly as desired by the company, which could decide to use the application initially for internal purposes.

Since almost all the main algorithms and checks are performed on the server side, the finished site is highly responsive for the user and should ensure a high level of security.

The design is simple but functional and adapts to any screen, ensuring a good user experience, an aspect also supported by components that indicate data loading to avoid leaving the user waiting and reduce layout shift.

Although not 100% complete, this project allowed me to discover and use many current and marketable technologies in the job market, as well as to experience the profession of a Web Developer.

## 5. Bibliography

- React.js: <https://it.reactjs.org/docs/getting-started.html>
- Udemy course: <https://www.udemy.com/course/react-the-complete-guide-incl-redux/learn>
- React Router: <https://reactrouter.com/en/main>
- Next.js: <https://nextjs.org/docs/getting-started>
- Google Firebase: [https://firebase.google.com/docs?gclid=ds&gclid=CLj1jdTlj\\_0CFUNvGwod0A4Lrw&hl=it](https://firebase.google.com/docs?gclid=ds&gclid=CLj1jdTlj_0CFUNvGwod0A4Lrw&hl=it)
- MongoDB: <https://www.mongodb.com/docs/>
- GitLab: <https://www.gitlab.com>
- Laravel: <https://laravel.com/docs/9.x>
- Laravel Sanctum: <https://laravel.com/docs/9.x/sanctum>
- Axios: <https://axios-http.com/docs/intro>
- Google PageSpeed Insight: <https://developers.google.com/speed/docs/insights/v5/about?hl=it>
- Material UI: <https://mui.com/material-ui/getting-started/overview/>
- Tailwind: <https://tailwindcss.com/docs/installation>
- SASS: <https://sass-lang.com/documentation/>
- BEM: <https://scalablecss.com/bem-quickstart-guide/#1391467>
- Chart.js e react-chartjs-2: <https://react-chartjs-2.js.org/>
- Terraform: <https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/instance>
- AWS EC2: <https://aws.amazon.com/it/ec2/instance-types/>
- React Developer Tools: <https://chrome.google.com/webstore/detail/react-developer-tools/fmkadmapgofadopljbjfkapdkoienihi>
- Docker: <https://docs.docker.com/>

