PERFORMANCE OPTIMIZATION TEST CASES

sales@vallettasoftware.com
www.vallettasoftware.com

CASE #1

A global leader in the field of minimally-invasive clinical solutions for the Aesthetic & Ophthalmology markets.

NO COMPRESSION OF STATIC CONTENT

Thanks to compression, it is usually possible to reduce the amount of loaded data by ~70%, which has a beneficial effect on the first appearance of content on the page

Cloudfront did not perform compression when serving content, we decided to enable this feature.

NO CACHE-CONTROL HEADERS FOR STATIC CONTENT

By default, the browser and all intermediate links to the main source can read the Cache-Control header to decide whether to cache content.

If caching settings are not set or set incorrectly, this may result in content being attempted to be retrieved from the original source (S3 bucket) too often, rather than from the Cloudfront cache, which is closer to the client.

In this case, such headers should have been given by the S3 bucket, which is what was configured.

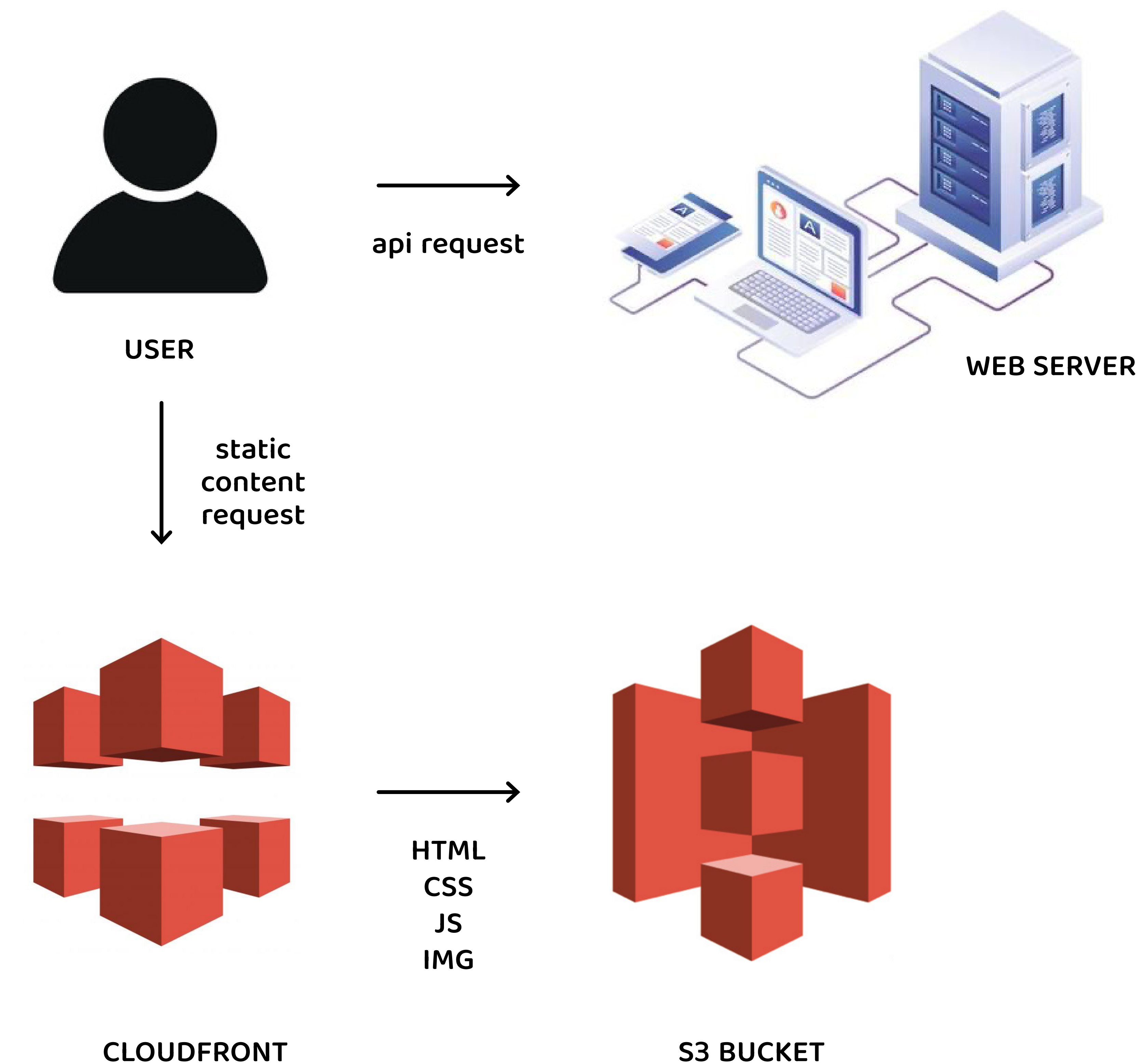
USELESS DEPENDENCIES AND BOILERPLATE ISSUE

The project contained a large amount of unused code and libraries. Therefore, we had a large bundle size and a long javascript execution time. We removed all unused dependencies from the code, and also used lazy loading to reduce the size of the loaded code when entering the page.

#Development #API #Backend #S3Bucket #Cloudfront #SPA

GENERAL INFRASTRUCTURE DIAGRAM

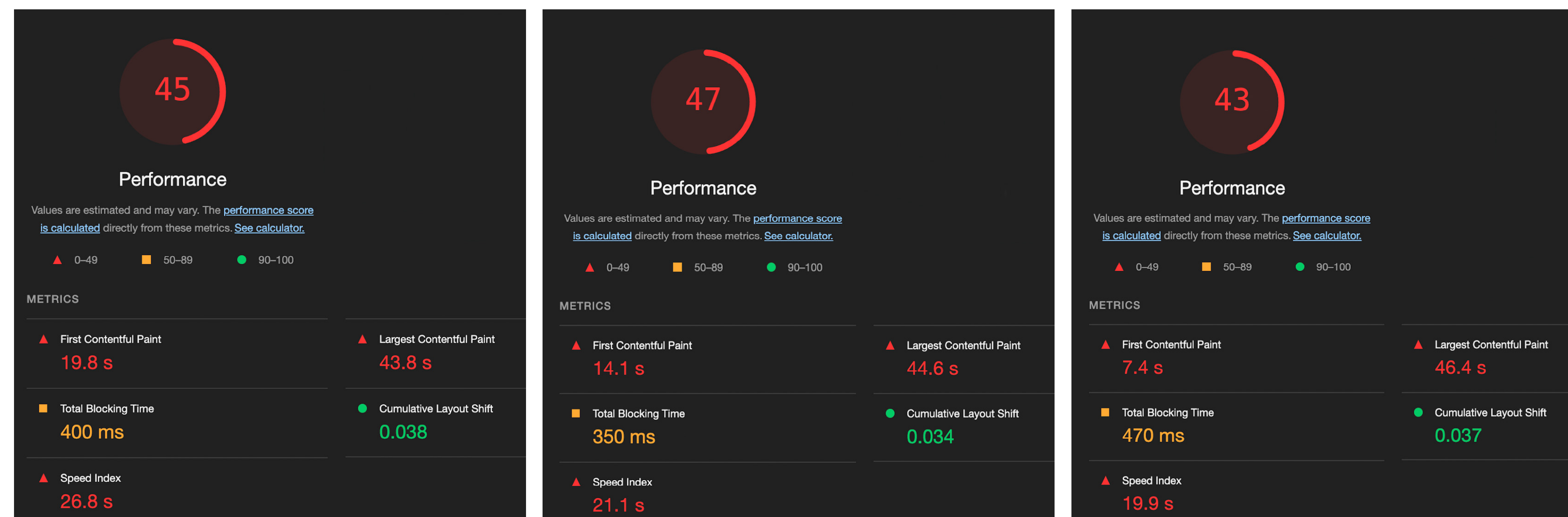
A classic SPA that receives static content from an S3 bucket via Cloudfront CDN and makes API requests to the backend.



CASE #1: LIGHTHOUSE MEASUREMENTS

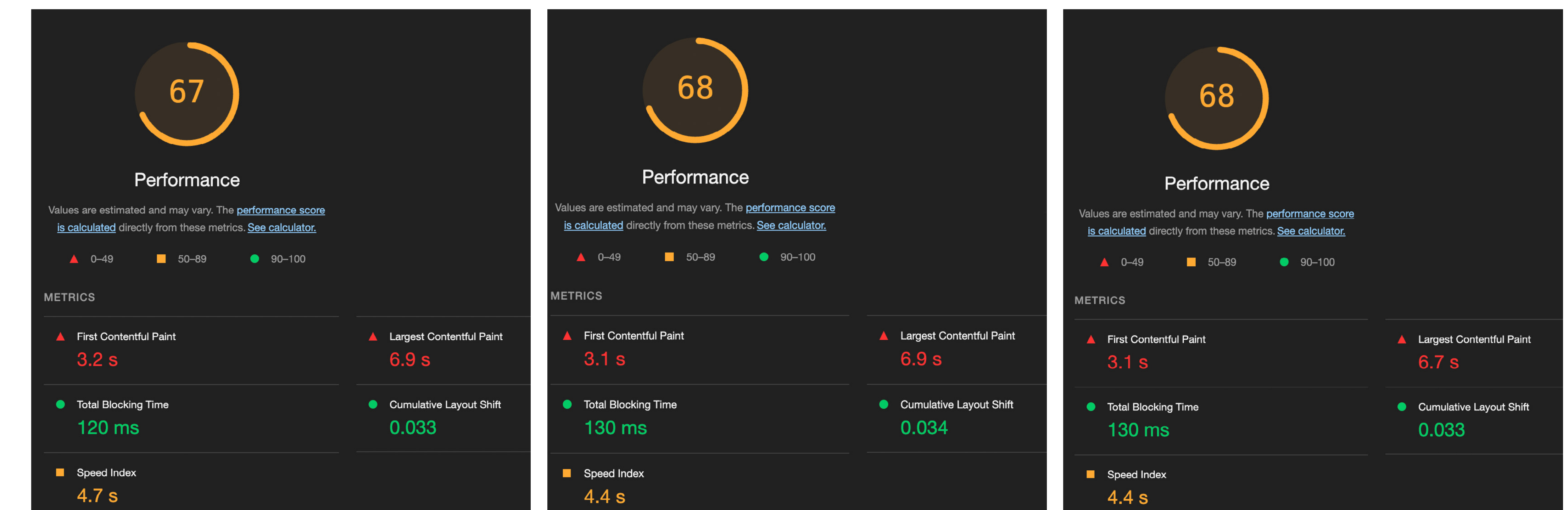
Valletta Software Development

BEFORE



	Performance	FCP	LCP	TBT	CLS	Speed Index
1	45	19.8	43.8	400	0.038	26.8
2	47	14.1	44.6	350	0.034	21.1
3	43	7.4	46.4	470	0.037	19.9
Average	45.00	13.77	44.93	406.67	0.04	22.60

AFTER



	Performance	FCP	LCP	TBT	CLS	Speed Index
1	67	3.2	6.9	120	0.033	4.7
2	68	3.1	6.9	130	0.034	4.4
3	68	3.1	6.7	130	0.033	4.4
Average	67.7	3.13	6.83	126.66	0.033	4.5

CASE #2

PROBLEM FORMULATION

The customer came to us with the idea of creating a social network for traders, where more experienced traders will be able to maintain public portfolios, and less experienced ones will be able to study their decisions and behavior, learning and repeating all actions after them (including in automatic mode).

The project was supposed to be built on the basis of mobile devices. One of the Customer's requirements was the mandatory use of the NativeScript platform and the Angular framework. We had to carry out the development, and the product support should have been carried out by the Customer's own team, which had only these competencies.

CHALLENGE

During the development of the application, **we created 180+ design layouts** and began implementation. Almost immediately we were faced with the shortcomings of the NativeScript platform, namely, a suboptimal rendering mechanism. The platform is designed for classic small mobile applications; attempts to use it for rich, dynamically changing layouts **caused freezes and lags**. Navigation and swiping also caused difficulties, especially on technically weak mobile devices.

SOLUTION

To solve these problems, we first, as far as possible, **reduced the amount of data transferred** from endpoints to the mobile application, and also optimized queries so that data was retrieved as quickly as possible. This only partially helped, since the problem was precisely in the NativeScript rendering engine, and not in the work with the data itself. After this, we conducted a series of tests and selected the necessary parameters for optimizing the rendering processes, which completely solved all the problems that arose. The use of these parameters, as well as potential rendering problems, have not been described in the NativeScript documentation, and are entirely our know how.

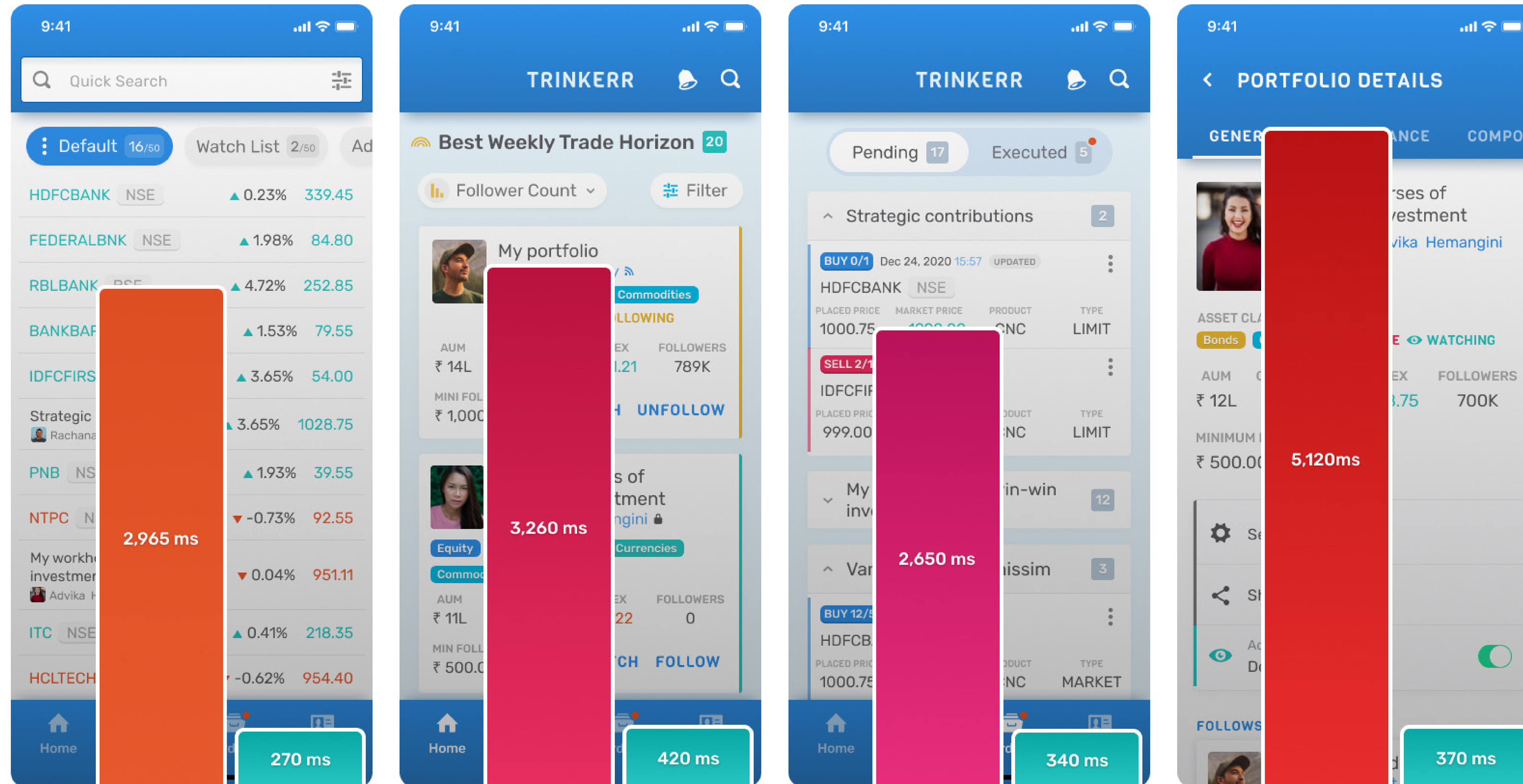
RESULTS

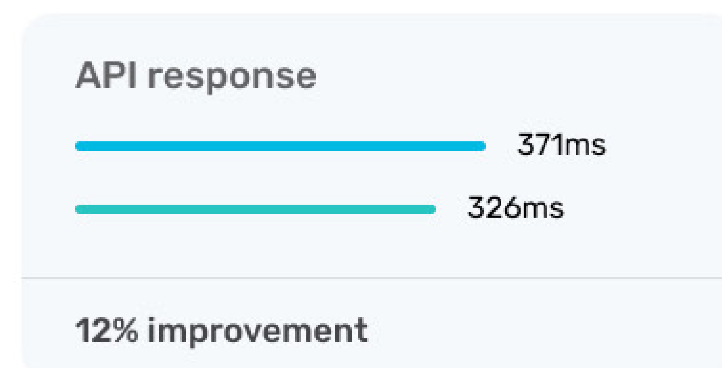
- Having applied this solution, the problem of lags and freezes was solved.
- Redrawing of the main screens, which took about 800-1400 milliseconds, was completed in 4-6 ms. **The acceleration was more than 200 times.**
- By the way, we usually do not use NativeScript to develop mobile applications, we use ReactNative where the above problems are absent.
- The application was successfully completed and transferred to the Customer's team.

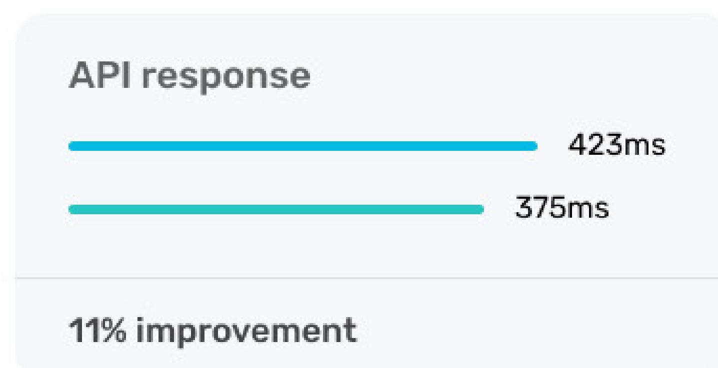
#Development #MobileDevelopment #NativeScript #ReactNative #SocialNetwork #Trading #Education

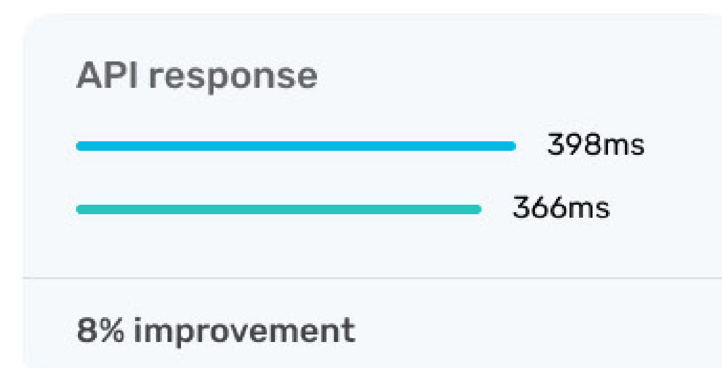
CASE #2: AVERAGE SCREEN RENDERING TIME

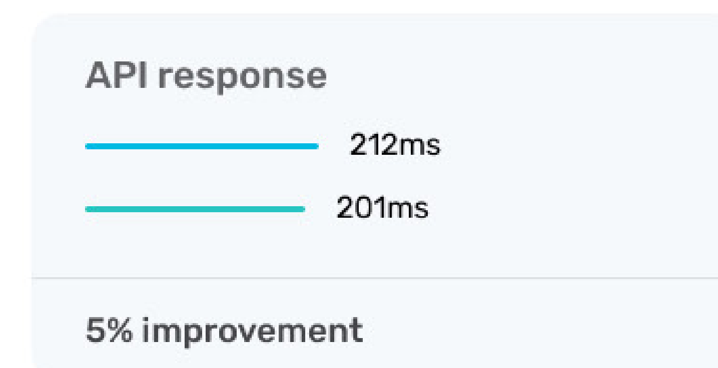
Valletta Software Development











CASE #3

Valletta Software Development

№1 service of activities for kids and families in Norway. The client had an existing platform created with a huge amount of data on different cultural and sport events all over Norway. The most important issues were to meet the requirements of the search engines, and to implement advanced filtering.

ISSUES IDENTIFIED

The client came with a ready project that had the **following pain points**:

- Mobile devices were not supported.
- Poor UX, weak UI.
- Overall instability of the code base, multiple errors in the web application's operation.

During the initial communication, it became clear that a simple preparation of the mobile version of the web application did not satisfy the client, who required a complete redesign of the application and a radical improvement of the UX.

CHALLENGES

While working on the project, we encountered the problem of the long loading time of the main page, as its server-side generation consisted of many operations: setting the client's geographical location, fetching open/available activities at the moment based on their location sorted by distance, forming a section of currently prioritized activities, etc. The **solution** was the automatic daily generation of a cache for all cities in Norway, which led to a page load speed increase **from 8 seconds to 100 milliseconds, i.e., 80 times faster**.

APPROACH

We started working and, in tandem with the client, **prepared 160+ layouts** of the new application, including about 90 for mobile platforms. After the approval of the layouts, we proceeded to develop the specification. Upon completion of these stages, the client realized that to implement all ideas, it was necessary to completely update the technology stack used. Thus, the application transformed from a simple ASP.NET MVC into Angular Universal/NgRx on the frontend and .NET Core/MS SQL on the backend.

RESULTS

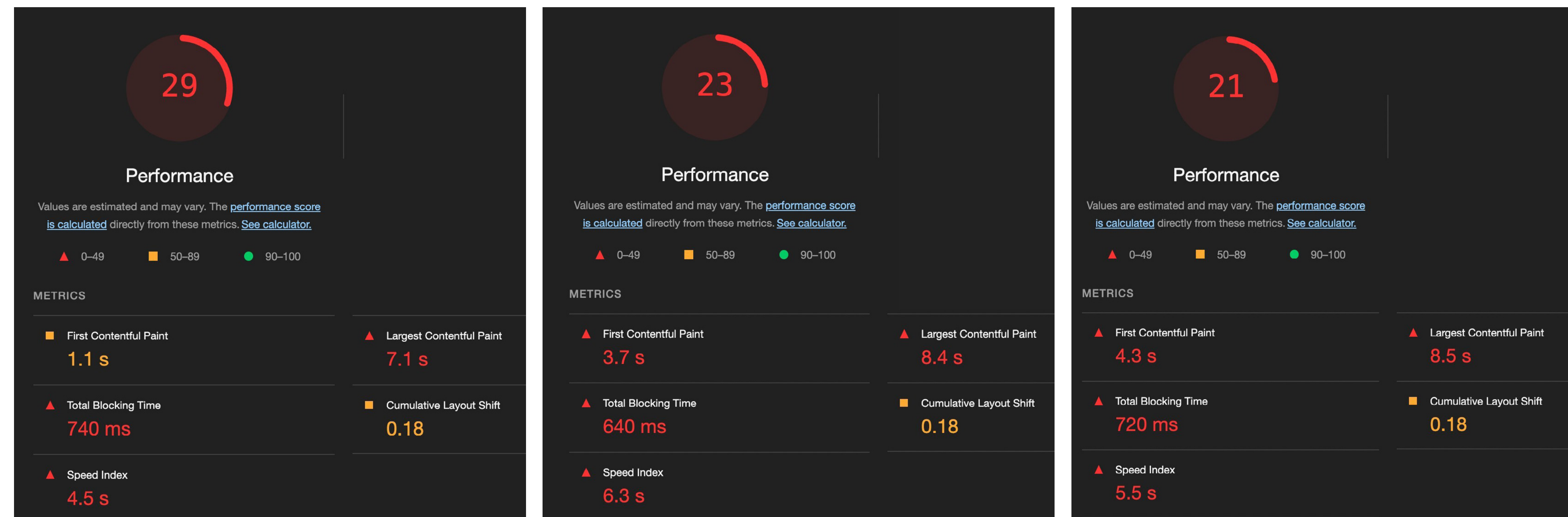
- The implementation of all ideas **took about 18 months.**
- Using SPA and SSR technologies, we have implemented deep integration with Google maps.
- Updated project **attracted over $700,000 in investments.**
- Monetizing the project through cooperation with activity owners (running advertising campaigns on the project and more).

#Development #WebDevelopment #MobileDevelopment #Angular #.Net #MSSQL #SSR

CASE #3: LIGHTHOUSE MEASUREMENTS

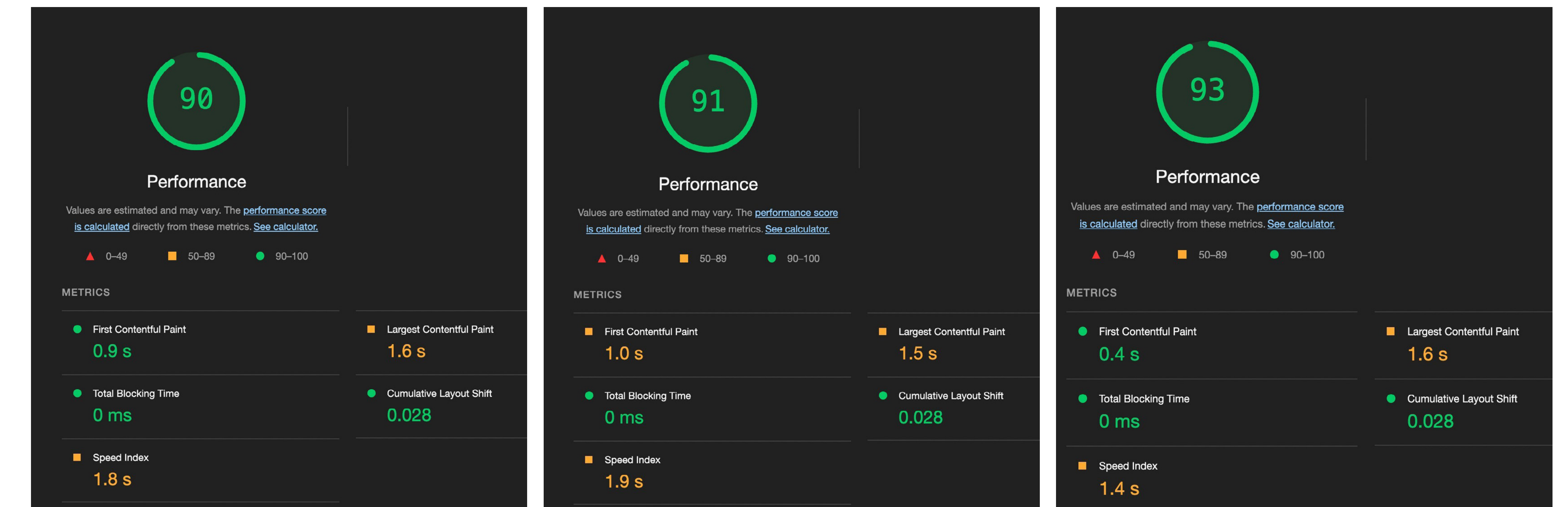
Valletta Software Development

BEFORE



	Performance	FCP	LCP	TBT	CLS	Speed Index
1	29	1.1	7.1	740	0.18	4.5
2	23	3.7	8.4	640	0.18	6.3
3	21	4.3	8.5	720	0.18	5.5
Average	**24.3**	**3.03**	**8**	**700**	**0.18**	**5.4**

AFTER



	Performance	FCP	LCP	TBT	CLS	Speed Index
1	90	0.9	1.6	0	0.028	1.8
2	91	1.0	1.5	0	0.028	1.9
3	93	0.4	1.6	0	0.028	1.4
Average	**91.3**	**0.76**	**1.56**	**0**	**0.028**	**1.7**