Data Visualization on Mobile Devices: A Case Study with Pluviometric Data

Wallyson Nunes Alves Lima[1], Milton Hirokazu Shimabukuro[2]

[1]Departamento de Matemática e Computação (DMC) – Faculdade de Ciências e Tecnologia da Unesp (FCT/Unesp)

wallyson.n.a.lima@gmail.com, milton.h.shimabukuro@unesp.br

Abstract.

Mobile devices have become popular over the years, earning more and more resources. One area that has been growing and being used on mobile devices is the visualization of data, with several scientific works being produced. In this context, it was proposed to analyze different visualization techniques to verify the suitability to mobile devices, to identify the inherent limitations of these devices and to compare with a web version.

It was verified that there was some difficulty in replicating some features of the web version in the mobile version, since there were differences in the use of the interface between the mobile device and the traditional computers (Interface touch x keyboard and mouse), and it was seen that the web version obtained a better performance than the mobile version.

Resumo.

Mobile devices have become increasingly popular over the years, gaining more and more features. One area that has grown significantly and is being widely used on mobile devices is data visualization, with numerous scientific studies being produced in this field. In this context, this study aimed to analyze different visualization techniques to assess their suitability for mobile devices, identify the inherent limitations of these devices, and compare them with a web version.

It was found that there were certain difficulties in replicating some features from the web version on the mobile version, due to differences in interface usage between mobile devices and traditional computers (touch interface vs. keyboard and

mouse). Additionally, the web version showed better performance compared to the mobile version.

1. Introduction

This article addresses data visualization techniques on mobile devices applied to rainfall index data, which has been collected over a long period from hundreds of stations in the state of São Paulo. The goal is to support the interpretation of this large volume of data. In this context, the study proposes creating visual representations using selected techniques to facilitate data analysis on mobile devices, taking into account their inherent limitations such as small screen size, limited processing power, low memory capacity, and battery life (Weng et al., 2012). Therefore, when designing visualizations for mobile devices, it is essential to carefully consider whether the visualization is suitable for such screens, whether a touch interface (using fingers) is appropriate instead of a traditional mouse and keyboard, and whether the visualizations are dynamic—users need to interact with the data rather than just view it (Ward; Grinstein; Keim; 2010).

The main library used to generate the visualizations was Data-Driven Documents (D3), a JavaScript library for manipulating documents based on data. It supports data visualization using Hypertext Markup Language (HTML), Scalable Vector Graphics (SVG), and Cascading Style Sheets (CSS), with an emphasis on web standards, and runs on major modern browsers using the Document Object Model (DOM). The DOM provides a structured tree representation of the document and allows manipulation of its objects. Another library used in this study was NVD3, a framework built on top of D3.js that reuses its visualizations and components. The initials “NV” relate to Novus Partners, the company that created the library.

Two applications were defined and implemented to support data analysis through visualizations. The first, called Mobile Visualization Tool (MobiViTool), is a web app that uses visualizations on Android through WebViews. The second was built solely using web technologies, maintaining similarity with the first, and is called

Web Visualization Tool (WebViTool). Both applications use D3 and NVD3 and are compared as potential solutions for the interpretation and analysis of rainfall data.

Throughout this work, Section 2 presents the definition of data visualization, its importance to society, its growth, and the process of creating a visualization. Section 3 explores the theoretical background on the Android platform and the implementation of a web app. Then, Section 4 details the architecture and implementation of this work, including the technologies used, the architecture of the applications, and their operational flow. Section 5 provides a performance analysis of the applications, highlighting their limitations, differences, development challenges, and the results obtained. Finally, Section 6 presents the conclusion of this study and potential future developments.

2. Data Visualization

Visualization is defined as the representation of information through graphical means, as an image can synthesize a large amount of information and is processed more efficiently by humans than merely observing textual data. As such, information visualization can enhance cognitive capacity by storing large quantities of information in quick and accessible formats using visual representations (Neugebauer et al., 2015).

Visual representations have the advantage of being language-independent—anyone who can interpret a chart is capable of extracting information from a visual representation. In our daily lives, information visualizations are commonly found to facilitate understanding, such as a table in a newspaper, a subway map, a city map, or a weather chart, among others. These examples clearly illustrate the power of data visualization, as it makes information interpretation more intuitive through visual representations.

The field of data visualization has grown rapidly since its inception, with a vast array of different techniques and tools becoming available. Most visualizations are built entirely using web technologies, primarily JavaScript, HTML, and CSS (Kerren et al., 2017).

Visual representations follow a process during their creation. Data may be stored in a simple or structured way and can originate from a variety of sources, such as a web server, database, or even textual data files. The purpose of visualization is to transform raw, hard-to-interpret data into visual representations that make it easier for users to extract information intuitively. The **Visualization Pipeline**, shown in Figure 1, illustrates the steps involved in creating or generating a visual representation (Ward; Grinstein; Keim; 2010):

- **Data Modeling**: At this stage, the data—whether stored in a file or database—must be structured, and it is necessary to understand the name, type, size, and attributes of the data;
- **Data Selection**: In this stage, the subset of data to be used in the visualization is identified. This can be done through algorithms;
- **Visual Mapping of Data**: This is a critical stage, often referred to as the heart of visualization, in which the data is mapped to graphical entities. Visual attributes such as size, color, and object position are controlled in this step;
- **Scene Parameter Configuration**: Visualizations may include attributes independent of the data itself, such as map selection colors, map sounds, and lighting specifications (for 3D visualizations).

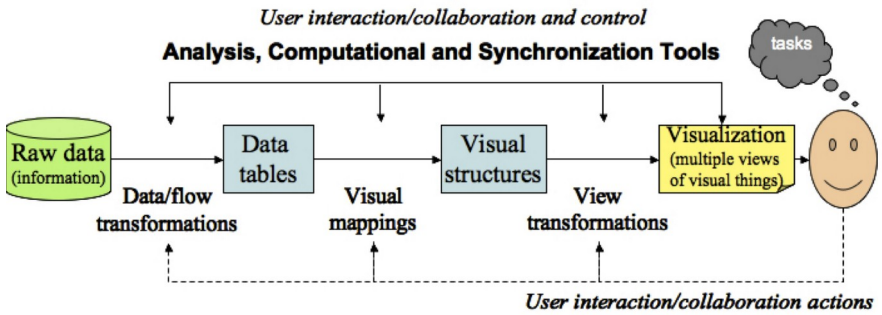


Figura 1: Processo de Visualization Pipeline

Fonte: Ward, Grinstein e Keim (2010).

3. Android Platform and Web App Implementation

Android is an operating system for mobile devices based on a modified version of the Linux kernel and other open-source projects. The system was originally developed by Android Inc. for digital cameras; however, it was soon realized that this was a very limited market, and the focus shifted to smartphones. Android Inc. was later acquired by Google, and in 2007, the first version—Android 1.5, known as Cupcake—was released.

Android was developed based on Linux and was designed as a software stack. At the base is the Linux kernel, responsible for managing the device's hardware. Above it, the Hardware Abstraction Layer (HAL) provides abstractions so that hardware components such as the camera, Bluetooth, and accelerometer can be accessed through the Java Application Programming Interface (API). Next, the Android Runtime (ART) layer offers features for Ahead-of-Time (AOT) and Just-in-Time (JIT) compilation, which allow previously compiled applications to run more efficiently. Native C/C++ libraries enable programs to be written using these languages. At the top is the Java API framework, which simplifies and reuses modular system components and services. Finally, the highest layer consists of applications created for the Android platform (Platform Architecture, 2018).

To create the visualizations on Android, web technologies were used, and to run them on Android, WebViews were employed. A WebView is a user interface component that displays web pages. It is used to import and run web libraries and code by using WebKit to render the web page and communicates directly with the Java API layer. In this way, an application built using WebView is considered a web app, as it relies on web technologies and native Android resources, with an architecture divided into communicating components. One component contains the HTML, CSS, JavaScript, and related files, which interact with WebKit to render the HTML, and in turn, communicate with native Android resources, as illustrated in Figure 2.

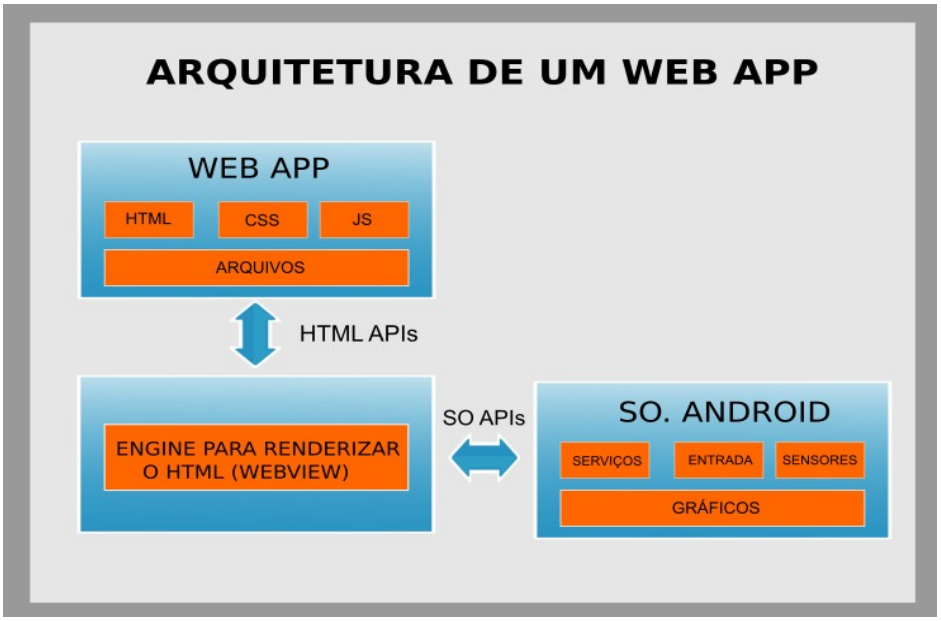
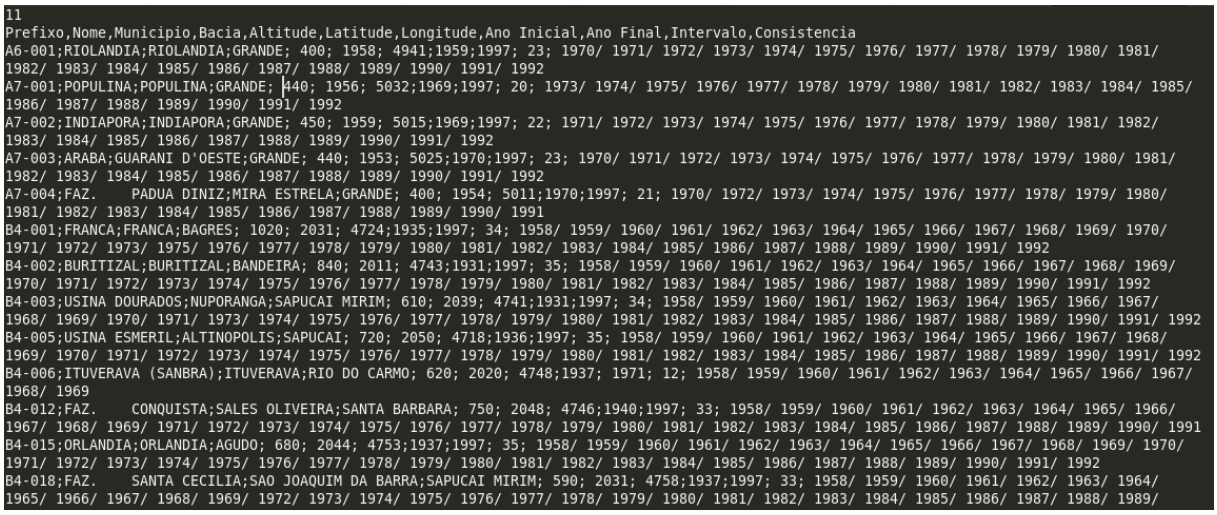


Figura 2: Arquitetura de um Web App.

Fonte: Autor

Additionally, the files containing pluviometric index data from the state of São Paulo were initially stored in several textual files. The data was separated by commas or semicolons, but it was found to be inefficient to use these textual files directly on mobile devices, as the large volume of data combined with the storage and processing limitations of mobile devices (Weng et al., 2012) would make it computationally expensive to handle the files locally.

With this in mind, a more efficient solution was to store the data on a database server (MariaDB), allowing the mobile device to request the data remotely as needed. This approach enables the device to retrieve only the data required during use, eliminating the need to download all the data to the device and thus improving the application's performance. An example of the data stored in a textual file is shown in Figure 3.



```
11
Prefixo,Nome,Municipio,Bacia,Altitude,Latitude,Longitude,Ano Inicial,Ano Final,Intervalo,Consistencia
A6-001;RIOLANDIA;RIOLANDIA;GRANDE; 400; 1958; 4941;1959;1997; 23; 1970/ 1971/ 1972/ 1973/ 1974/ 1975/ 1976/ 1977/ 1978/ 1979/ 1980/ 1981/
1982/ 1983/ 1984/ 1985/ 1986/ 1987/ 1988/ 1989/ 1990/ 1991/ 1992
A7-001;POPULINA;POPULINA;GRANDE; 440; 1956; 5032;1969;1997; 20; 1973/ 1974/ 1975/ 1976/ 1977/ 1978/ 1979/ 1980/ 1981/ 1982/ 1983/ 1984/ 1985/
1986/ 1987/ 1988/ 1989/ 1990/ 1991/ 1992
A7-002;INDIAPORA;INDIAPORA;GRANDE; 450; 1959; 5015;1969;1997; 22; 1971/ 1972/ 1973/ 1974/ 1975/ 1976/ 1977/ 1978/ 1979/ 1980/ 1981/ 1982/
1983/ 1984/ 1985/ 1986/ 1987/ 1988/ 1989/ 1990/ 1991/ 1992
A7-003;ARABA;GUARANI D'OESTE;GRANDE; 440; 1953; 5025;1970;1997; 23; 1970/ 1971/ 1972/ 1973/ 1974/ 1975/ 1976/ 1977/ 1978/ 1979/ 1980/ 1981/
1982/ 1983/ 1984/ 1985/ 1986/ 1987/ 1988/ 1989/ 1990/ 1991/ 1992
A7-004;FAZ.    PADUA DINIZ;MIRA ESTRELA;GRANDE; 400; 1954; 5011;1970;1997; 21; 1970/ 1972/ 1973/ 1974/ 1975/ 1976/ 1977/ 1978/ 1979/ 1980/
1981/ 1982/ 1983/ 1984/ 1985/ 1986/ 1987/ 1988/ 1989/ 1990/ 1991
B4-001;FRANCA;FRANCA;BAGRES; 1020; 2031; 4724;1935;1997; 34; 1958/ 1959/ 1960/ 1961/ 1962/ 1963/ 1964/ 1965/ 1966/ 1967/ 1968/ 1969/ 1970/
1971/ 1972/ 1973/ 1975/ 1976/ 1977/ 1978/ 1979/ 1980/ 1981/ 1982/ 1983/ 1984/ 1985/ 1986/ 1987/ 1988/ 1989/ 1990/ 1991/ 1992
B4-002;BURITIZAL;BURITIZAL;BANDEIRA; 840; 2011; 4743;1931;1997; 35; 1958/ 1959/ 1960/ 1961/ 1962/ 1963/ 1964/ 1965/ 1966/ 1967/ 1968/ 1969/
1970/ 1971/ 1972/ 1973/ 1974/ 1975/ 1976/ 1977/ 1978/ 1979/ 1980/ 1981/ 1982/ 1983/ 1984/ 1985/ 1986/ 1987/ 1988/ 1989/ 1990/ 1991/ 1992
B4-003;USINA DOURADOS;NUPORANGA;SAPUCAI MIRIM; 610; 2039; 4741;1931;1997; 34; 1958/ 1959/ 1960/ 1961/ 1962/ 1963/ 1964/ 1965/ 1966/ 1967/
1968/ 1969/ 1970/ 1971/ 1973/ 1974/ 1975/ 1976/ 1977/ 1978/ 1979/ 1980/ 1981/ 1982/ 1983/ 1984/ 1985/ 1986/ 1987/ 1988/ 1989/ 1990/ 1991/ 1992
B4-005;USINA ESMERIL;ALTINOPOLIS;SAPUCAI; 720; 2050; 4718;1936;1997; 35; 1958/ 1959/ 1960/ 1961/ 1962/ 1963/ 1964/ 1965/ 1966/ 1967/ 1968/
1969/ 1970/ 1971/ 1972/ 1973/ 1974/ 1975/ 1976/ 1977/ 1978/ 1979/ 1980/ 1981/ 1982/ 1983/ 1984/ 1985/ 1986/ 1987/ 1988/ 1989/ 1990/ 1991/ 1992
B4-006;ITUVERAVA (SANBRA);ITUVERAVA;RIO DO CARMO; 620; 2020; 4748;1937; 1971; 12; 1958/ 1959/ 1960/ 1961/ 1962/ 1963/ 1964/ 1965/ 1966/ 1967/
1968/ 1969
B4-012;FAZ.    CONQUISTA;SALES OLIVEIRA;SANTA BARBARA; 750; 2048; 4746;1940;1997; 33; 1958/ 1959/ 1960/ 1961/ 1962/ 1963/ 1964/ 1965/ 1966/
1967/ 1968/ 1969/ 1971/ 1972/ 1973/ 1974/ 1975/ 1976/ 1977/ 1978/ 1979/ 1980/ 1981/ 1982/ 1983/ 1984/ 1985/ 1986/ 1987/ 1988/ 1989/ 1990/ 1991
B4-015;ORLANDIA;ORLANDIA;AGUDO; 680; 2044; 4753;1937;1997; 35; 1958/ 1959/ 1960/ 1961/ 1962/ 1963/ 1964/ 1965/ 1966/ 1967/ 1968/ 1969/ 1970/
1971/ 1972/ 1973/ 1974/ 1975/ 1976/ 1977/ 1978/ 1979/ 1980/ 1981/ 1982/ 1983/ 1984/ 1985/ 1986/ 1987/ 1988/ 1989/ 1990/ 1991/ 1992
B4-018;FAZ.    SANTA CECILIA;SAO JOAQUIM DA BARRA;SAPUCAI MIRIM; 590; 2031; 4758;1937;1997; 33; 1958/ 1959/ 1960/ 1961/ 1962/ 1963/ 1964/
1965/ 1966/ 1967/ 1968/ 1969/ 1972/ 1973/ 1974/ 1975/ 1976/ 1977/ 1978/ 1979/ 1980/ 1981/ 1982/ 1983/ 1984/ 1985/ 1986/ 1987/ 1988/ 1989/
```

Figura 3: Arquivo textual com os dados

Fonte: Autor

4. Architecture and Implementation

The main purpose of this study was to develop a web app for the Android platform to explore visualization techniques using pluviometric index data from the State of São Paulo. Additionally, a web-only version of the same web app was developed to compare its limitations and differences with the mobile version.

The visualizations were built using web technologies and the D3 and NVD3 visualization libraries. Thus, the mobile version, called **MobiViTool**, is an Android application developed in Java and XML, and its backend uses a MariaDB database that connects remotely to the application. Since mobile applications differ significantly from desktop applications in terms of limitations and user interface (Cobas; Iglesias; Seoane; 2015), a web version—called **WebViTool**—was also created. This version was developed using the PHP programming language, the open-source Apache web server, the jQuery JavaScript library, and for generating visualizations, the D3 and NVD3 libraries, along with HTML, CSS, and JavaScript.

Table 1 below presents the development environment used in this study for both MobiViTool and WebViTool.

Tabela 1. Estrutura do ambiente de desenvolvimento

	Ambiente de Desenvolvimento	**Plataforma**
Lado Servidor	Sistema Operacional	Fedora 27
	Servidor de Banco de Dados	MariaDB (10.1.29)
	Web Server	Apache (2.4.29)
Lado Cliente	Biblioteca para o ambiente	jQuery (1.9.1)
	Bibliotecas para os gráficos	D3 (v3 e v4), NVD3 (1.1.15)
	Sistema Operacional	Android 6.0 Marshmallow

4.1. Arcthitecture process

MobiViTool is a web app that uses a database server to store pluviometric data. First, the user selects the station locations and the time period to be analyzed through the application. The app then establishes a remote connection with the database, which returns the averages of the selected data. Next, the app generates a text file in CSV or TSV format, structured specifically for use within the application. A WebView is then rendered, which reads the data from the text files and, using visualization libraries (D3, NVD3) and JavaScript code, generates the visualizations.

In the case of **WebViTool**, an Apache server and a database are used. The user accesses this server via a web browser, and the data is processed on the server side. The processing is handled using the PHP programming language, and after processing, the visualization is displayed on the mobile device.

Unlike MobiViTool, WebViTool performs all processing on the server side, and only the final result is rendered on the client side. As shown in the diagram in **Figure 4**, WebViTool has one less architectural component compared to MobiViTool, as illustrated in **Figure 5**.

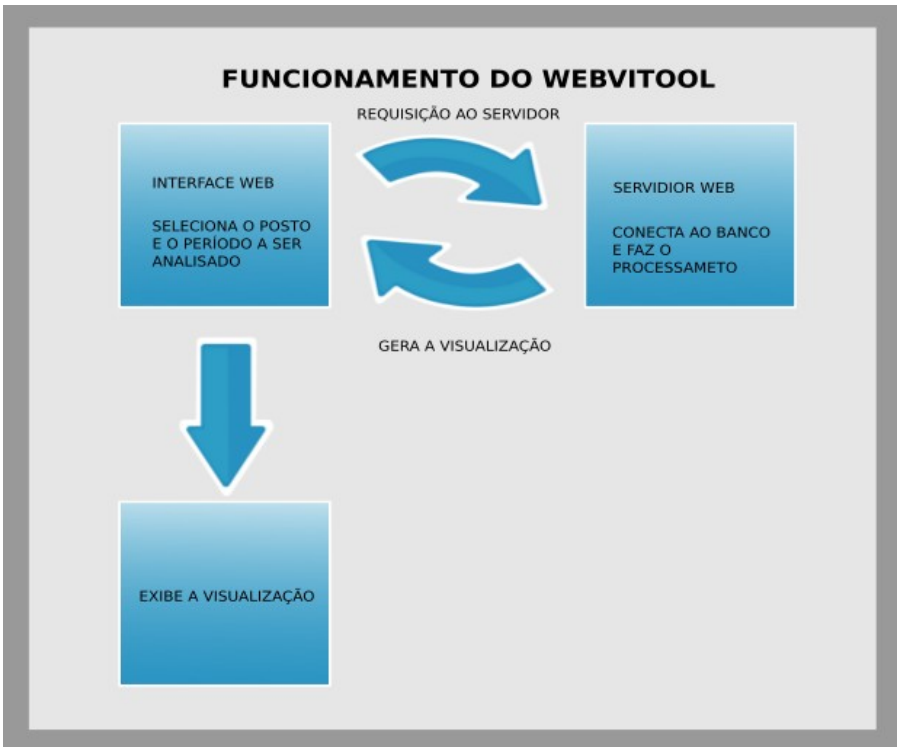


Figura 4: Arquitetura de Funcionamento do Webvitool

Fonte: Autor

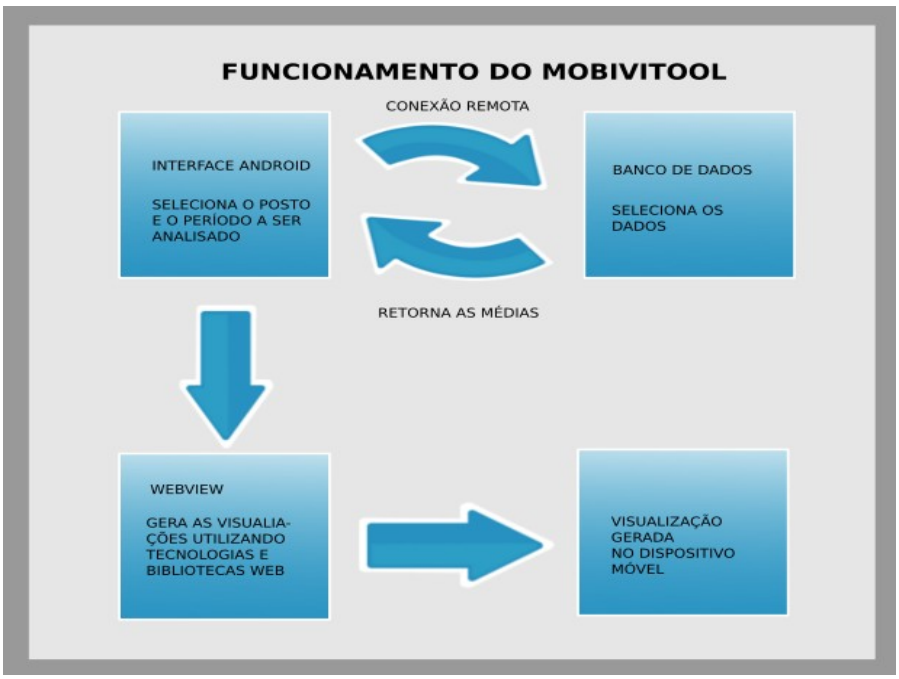


Figura 5: Arquitetura de uma Webview em uma aplicação Android

Fonte: Autor

5. Discussion and Results

The study of different visualization techniques for mobile devices involved evaluating both design and implementation. A discussion is presented on the selection of techniques and the limitations encountered when working with visualizations on mobile devices, as well as a comparison between the mobile version (**MobiViTool**) and its web counterpart (**WebViTool**), both accessed via smartphone.

Initially, to compare the two versions, the idea was to create a hybrid application for the web version using frameworks such as Apache Cordova or Xamarin. However, it was found that there would be little difference between the two applications, as both would use the same architecture and web libraries to generate the visualizations. The only distinction would be in the interface design: one using web technologies (as in Cordova or Xamarin) and the other using Java and XML. With this in mind, it was decided to build the web version entirely as a server-side application, as this comparison would be more meaningful, given that the execution occurs on the server and the smartphone merely acts as a display device.

One of the main guidelines in choosing the visualization techniques was to consider the limitations of smartphones, while also selecting techniques that would effectively represent the pluviometric data from the state of São Paulo. Based on this, the first technique chosen was the **Bar Chart**, due to its simplicity and ease of understanding. For example, the data could be sorted by monthly rainfall averages, making interpretation more intuitive and allowing for a highlighted value—displayed in a different color and with a tooltip showing the numeric value upon clicking a specific month. See **Figure 6**.

Expanding on this visualization, another bar chart-based visualization was created to analyze more than one average per station simultaneously. See **Figure 7**.

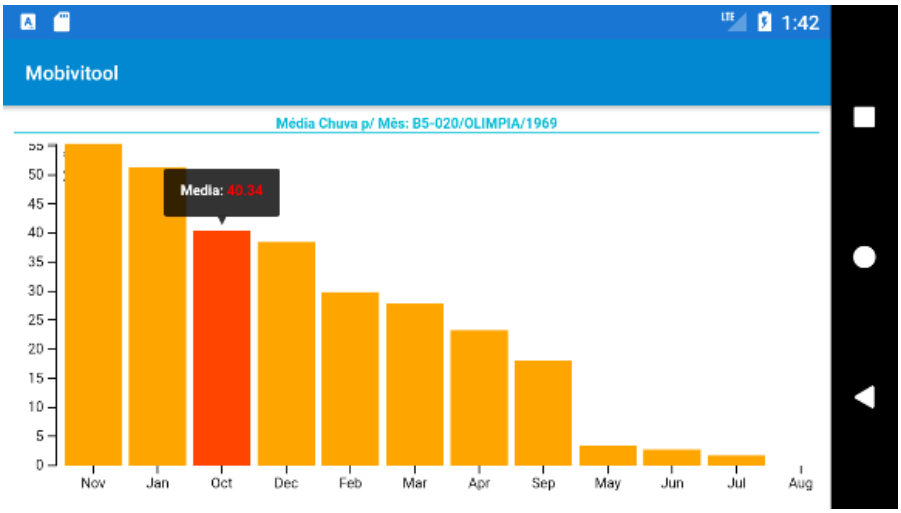


Figura 6: Visualização no Mobivitool Simple chart utilizando a técnica “Bar Chart” organizada descrescente pela média do volume pluviométrico de cada mês

Fonte: Autor

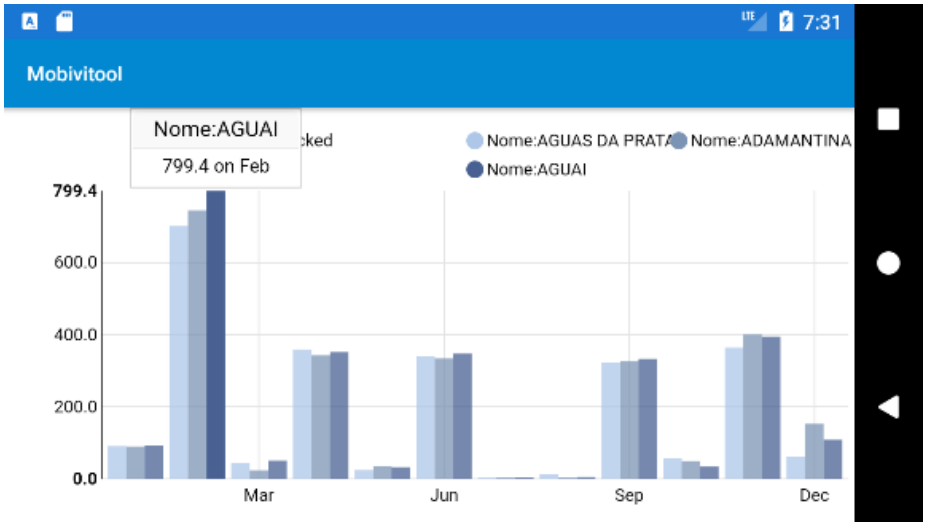


Figura 7: Visualização no Mobivitool baseada na técnica bar chart, mútiplas barras

Fonte: Autor

Using the **“Brush and Zoom”** technique, which allows the user to select a region of interest to zoom in on—focusing on a specific area—proved ideal for the small screens of mobile devices. This way, a visualization that is initially reduced in size can, through user interaction, represent a large amount of data in a limited space.

This technique is particularly suitable for separating data into time series. In this study, it was successfully used to display **10 years of pluviometric index data in a single visualization** without compromising usability or readability, as shown in **Figure 8**.

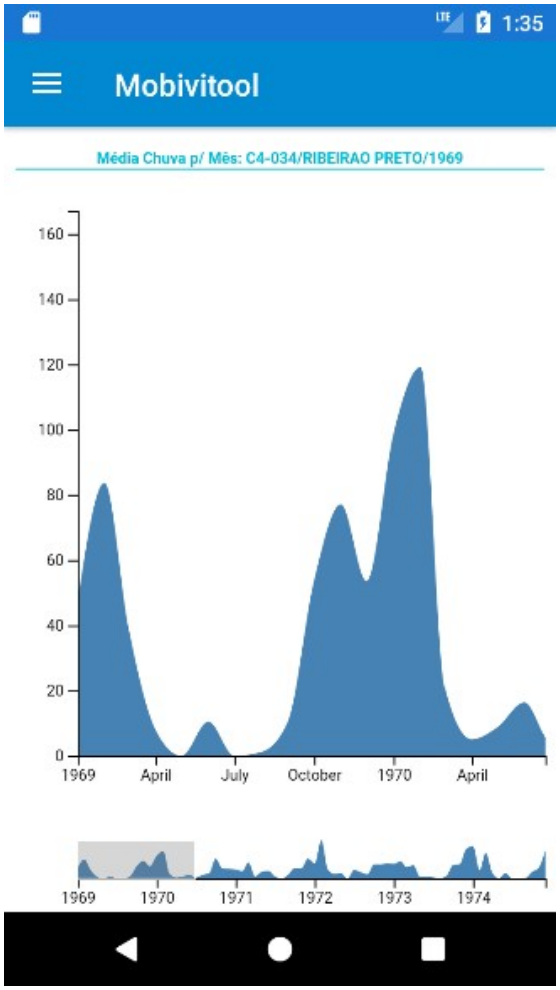


Figura 8: Visualização no Mobivitool baseada na técnica “Brush and Zoom”

Fonte: Autor

Another technique selected was based on **geospatial location**, used to divide the data from each station according to its position on the map of the state of São Paulo. The visualization of each station's information was enabled through zooming to focus on the region of interest, along with a tooltip to display the station's data.

Figures **9** and **10** show the visualizations in the **MobiViTool** application and a corresponding visualization in **WebViTool**, respectively.

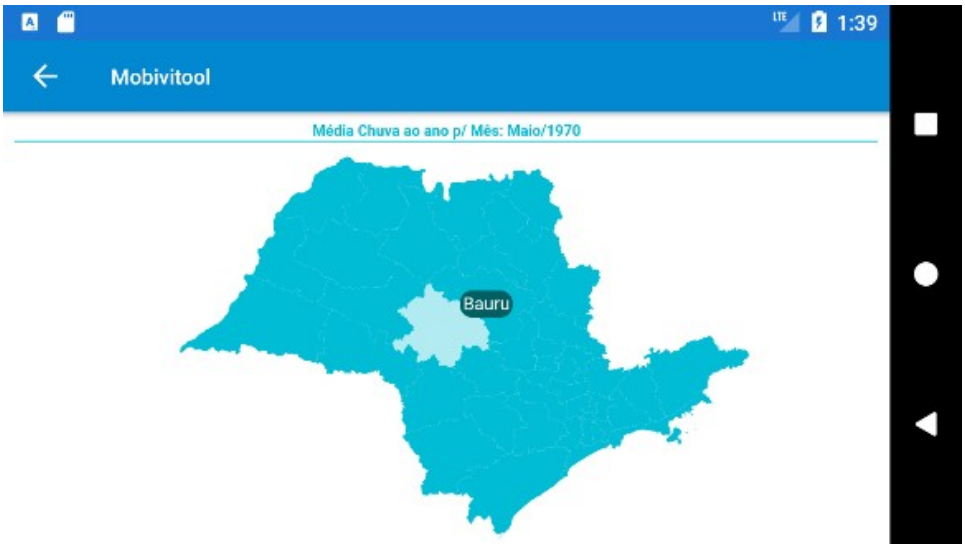


Figura 9: Visualização do Mobivitool baseada na técnica geoespacial no mapa do Estado de São Paulo

Fonte: Autor

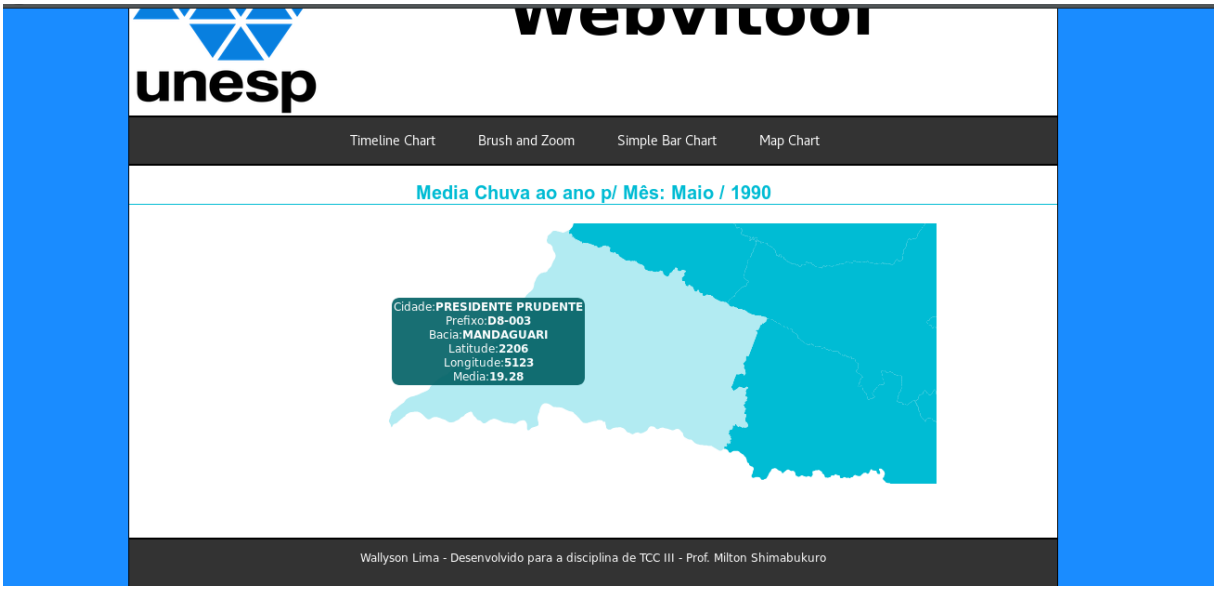


Figura 10: Visualização do Webvitool baseada na técnica geoespacial com o zoom na região de interesse

Fonte: Autor

The following section describes the development process and highlights the difficulties encountered. The first system developed was **MobiViTool**, and several challenges arose when attempting to run the visualization libraries and web technologies through a WebView on Android. The **D3** and **NVD3** libraries did not run locally and had to be accessed via external links, requiring the application to download them from the internet. Visualizations were initially created and tested in a web browser, and once completed, they were adapted for Android. This strategy was adopted due to the ease of testing and debugging using browser developer tools—a task that is not trivial on Android.

Once MobiViTool was completed, development began on the version built solely with web technologies, primarily using the **PHP** programming language. This development process was much faster, since the visualization technologies were originally designed for web environments. The only significant challenge was porting the Java code used to build the interfaces into PHP.

The interface difference between an Android application and a web/desktop application—**touch interface vs. mouse/keyboard**, respectively—is a key factor that impacted development. In the web version, certain features such as displaying tooltips when hovering over an area of interest (e.g., a bar in a bar chart or a city in a geospatial visualization) could not be replicated in Android due to the absence of mouse interactions.

The performance of both applications on mobile devices was also analyzed. Two devices were used for testing: a **virtual smartphone** within the Android Studio IDE, configured with 2 CPU cores, 1 GB of RAM, and a 720 x 1280 screen, representing low to mid-range smartphones at the time of testing; and a **physical smartphone**, the **Motorola Moto E 2nd Generation**, with 4 cores, 1 GB of RAM, and a 960 x 540 screen, representing an entry-level smartphone with low processing power.

Performance tests were carried out using the profiling tools available in Android Studio, analyzing **memory usage**, **CPU utilization**, and the **number of threads used**. The tests measured the time from selecting a visualization until it was fully loaded in the WebView.

The first test involved the **multi-bar bar chart visualization** shown in Figure 7. On the **virtual smartphone**, loading the visualization required between **45% and 60% average CPU usage**, peaking at **68%**; memory usage ranged from **60 MB to 90 MB on average**, with a peak at **92 MB**, and it used **52 threads**.

On the **real smartphone**, the same visualization required **10% to 30% average CPU usage**, peaking at **41%**; memory usage ranged from **20 MB to 60 MB on average**, with a peak at **80 MB**, and it used **51 threads**.

Another variable analyzed was the **load time** of the visualization. The **virtual smartphone** took approximately **58 seconds**, while the **physical smartphone** took **53 seconds**, as shown in **Table 2**. The consistency between the results from the virtual and physical devices indicates that a properly configured virtual smartphone can provide reliable performance metrics for evaluation.

Tabela 2. Teste visualização bar chart com múltiplas barras no MobiViTool

	CPU %	Pico CPU %	Memória (MB)	Pico Memória (MB)	Qtde Threads	Tempo (segundos)
Smartphone Virtual	45-60	68	60-90	92	52	58
Smartphone Físico	10-30	41	20-60	80	51	53

The **"Brush and Zoom"** technique was used to display data spanning up to 10 years. On the physical smartphone, CPU usage ranged from **5% to 35%**, peaking at **40%**. Memory usage ranged from **35 MB to 80 MB**, with a peak of **87 MB**, and the application used **51 threads**. In terms of load time, the physical smartphone took approximately **10.68 seconds**.

This visualization was implemented using the **D3** library, whereas the previous one used **NVD3**. Although the "Brush and Zoom" technique required more memory due to the larger volume of data, it was slightly more efficient in CPU usage. This is because **NVD3 is a library built on top of D3**, and therefore places a heavier load on the CPU. The higher memory usage in the "Brush and Zoom" visualization was expected, as it processes a larger dataset, as shown in **Table 3**.

Tabela 3. Teste visualização “Brush and Zoom” no MobiViTool

	CPU %	Pico CPU %	Memória (MB)	Pico Memória (MB)	Qtde Threads	Tempo (segundos)
Smartphone Físico	5-35	40	35-80	87	51	10,68

Another technique analyzed was the **geospatial technique**, which demands high processing power due to the need to load data from all stations, combined with the requirement to render the map of the state of São Paulo—an operation that heavily taxes the CPU.

On the **physical smartphone**, CPU usage ranged from **20% to 70%**, peaking at **78%**, and memory usage ranged from **60 MB to 100 MB**, with a peak of **100 MB**. The application used **53 threads**. In terms of loading time, it took approximately **72 seconds** to load on the physical smartphone.

This loading time was noticeably longer than the other visualizations, largely due to the physical smartphone’s limitations in processing the map of São Paulo, which contains many Cartesian coordinates and therefore requires significant computational power.

Tabela 4. Teste visualização geoespacial no MobiViTool

	CPU %	Pico CPU %	Memória (MB)	Pico Memória (MB)	Qtde Threads	Tempo (segundos)
Smartphone Físico	20-70	78	60-100	100	53	72

When analyzing the three techniques, it was observed that the visualizations performed differently. The fastest visualization in terms of load time was **"Brush and Zoom"**, which loaded in just **10.68 seconds**, even though it consumed more CPU and memory than the bar chart technique. This is because the bar chart used the **NVD3** library, which is built on top of D3 but is more resource-efficient. The **geospatial** visualization consumed the most CPU and memory, due to the need to load Cartesian coordinates and render the map of São Paulo.

Next, the performance of **WebViTool** was tested. Since it is a web application that runs entirely on the server side, it was necessary to create a **WebView** on Android that pointed to the application's URL, thus rendering the application inside the WebView. Android Studio's performance analysis tools were used to monitor the application's behavior. The same visualizations were tested.

The first test was also the **bar chart** technique. On the physical smartphone, CPU usage ranged from **5% to 55%**, with a peak of **57%**. Memory usage ranged from **73 MB to 90 MB**, peaking at **94 MB**, and the application used **47 threads**. As for the **load time**, the visualization was generated in just **4 seconds**, as shown in **Table 5**.

Tabela 5. Teste visualização bar chart no WebViTool

	CPU %	Pico CPU %	Memória (MB)	Pico Memória (MB)	Qtde Threads	Tempo (segundos)
Smartphone Físico	5-55	57	73-90	94	47	4

The **“Brush and Zoom”** visualization had the following performance on the physical smartphone: CPU usage ranged from **5% to 40%**, peaking at **40%**; memory usage ranged from **50 MB to 80 MB**, with a peak of **81 MB**, and it used **49 threads**. In terms of load time, it took **2 seconds**, as shown in **Table 6**.

Tabela 6. Teste visualização “Brush and Zoom” no WebViTool

	CPU %	Pico CPU %	Memória (MB)	Pico Memória (MB)	Qtde Threads	Tempo (segundos)
Smartphone Físico	5-40	40	50-80	81	49	2

Finally, the **geospatial technique** was tested in **WebViTool**. On the physical smartphone, CPU usage ranged from **5% to 70%**, peaking at **79%**. Memory usage ranged from **75 MB to 100 MB**, with a peak of **101 MB**, and it used **49 threads**. As for the load time, the visualization took **14 seconds**, as shown in **Table 7**.

Tabela 7. Teste visualização geoespacial no WebViTool

	CPU %	Pico CPU %	Memória (MB)	Pico Memória (MB)	Qtde Threads	Tempo (segundos)
Smartphone Físico	5-70	79	75-100	101	49	14

When analyzing the tests of **WebViTool**, it followed the same processing pattern observed in **MobiViTool** for the visualizations executed on the physical smartphone. Now, when comparing the performance of the two applications (**MobiViTool** vs. **WebViTool**), we can see that **WebViTool** loaded the visualizations in less time, used a lower percentage of CPU, and showed stable memory usage with only minor variations.

The **WebViTool** application can be considered more suitable for implementation, especially given that both applications require internet access. It also became clear—particularly in the case of the **geospatial visualization**, which took a long time to load—that further optimization would be necessary.

6. Conclusion

Recalling some key points, there were certain challenges in creating visualizations for mobile devices, primarily because web technologies were used—technologies that do not run natively on Android and required the use of **WebView**. Additionally, once the visualizations were created and validated in the browser, they had to be adapted for mobile devices. It became evident that some features that worked in the web version could not be replicated in the mobile version due to differences in interface usage between mobile devices and traditional computers (**touch interface vs. keyboard and mouse**), as well as mobile limitations such as **small screen size**, **limited processing power**, and **restricted memory** (Weng et al., 2012).

Testing clearly showed that the visualizations performed better on a real smartphone than on a virtual one, although the virtual device can still serve as a good performance and resource usage indicator. When analyzing the performance of both applications, **MobiViTool** and **WebViTool**, it was observed that the application running entirely on the **server side**—WebViTool—offered **significant performance gains**, even considering the trade-off that data is processed on the server and only displayed on the smartphone. Despite this gap, **WebViTool proved to be the more advantageous solution**.

As future work, it is suggested to implement more **complex visualizations**, such as **3D charts**, develop a version using **native programming** to analyze performance more accurately, and explore **new forms of interaction** through the touch interface.

Referências

Ward, M., Grinstein, G., Keim, D.. **Interactive Data Visualization: Foundations, Techniques, and Applications.** 1º Edição. Boca Raton: CRC Press, 2010. 507pg.

Kerren, A., Kucher, K., Li, Y., Schreiber, F., **BioVis Explorer: A visual guide for biological data visualization techniques.** Plos One, 2017.

Cobas, C., Iglesias, I., Seonane, F., **NMR data visualization, processing, and analysis on mobile devices.** Wiley Online Library, 2015.

Neugebauer, T., Bordeleau, E., Burrus, V., Brzezinski, R.. **DNA Data Visualization (DDV): Software for Generating Web-Based Interfaces Supporting Navigation and Analysis of DNA Sequence Data of Entire Genomes.** Plos One, 2015.

Weng, Y., Sun, F., Grigsby, J., **GeoTools: An android phone application in geology.** Comput. Geosci, 2012.

[1]- WebView. Disponível em: <https://developer.android.com/reference/android/webkit/WebView.html>. Acesso em: 20 de maio. 2018.

[2]- D3 Data-Driven Documents. Disponível em: <https://d3js.org/>. Acesso em: 20 de maio. 2018.

[3]- D3 Brush. Disponível em: <https://github.com/d3/d3-brush/>. Acesso em: 20 de maio. 2018.

[4]- D3 Zoom. Disponível em: <https://github.com/d3/d3-zoom/>. Acesso em: 20 de maio. 2018.

[5]- Arquitetura da Plataforma. Disponível em: <https://developer.android.com/guide/platform/index.html?hl=pt-br#api-framework>. Acesso em: 01 de junho. 2018.

[6]- Building Web Apps in WebView. Disponível em: <https://developer.android.com/guide/webapps/webview.html>. Acesso em: 01 de junho. 2018.