Responsive Maretec Forecast Visualization

Tomás Jacob Martins, Prof. João Brisson Lopes
Instituto Superior Técnico, Universidade de Lisboa

***Abstract*—Maritime and environmental forecasts have been used for many years by scientists and the general population to plan and manage their lives and jobs. Forecast information can be conveyed through geospatial visualizations but such representations need to adapt to modern web technologies, especially small screen devices like smartphones.**

This document describes a solution that takes into account all these limitations, such as data rates and screen sizes, to create a streamlined user experience across devices. To make it possible, we convert the MOHID Models HDF output to JSON, through an algorithm that extracts only the necessary information. Files are stored on a web server that serves the forecasts as requested by client visualizations. To complete the system, we developed a Javascript-based map to be integrated in the Maretec Forecast website. With this map, users can visualize a variety of magnitudes such as water temperature and salinity among many others.

We assessed our implementation based on its files size and conversion process speed. We were able to convert these files efficiently and reduce file size significantly. The visualization developed works similarly across devices and can be used even with slow connections. To assess the usability of our website we performed tests with users who also answered a SUS questionnaire.

***Index Terms*—geographic/geospatial visualization, web development, maps, Maretec**

I. Introduction

Weather and maritime forecasts are the product of mathematical models, developed by universities or research centres, that use current variables state and calculations to produce the most likely to happen scenarios. Visualization tools allow users to visualize the outputs of these models in a way that is relevant and understandable to them. Governmental agencies might use complex tools to visualise areas impacted by storms while the common citizen just needs to know if it will rain. Also, accurate forecasts can be a powerful tool to prepare for extreme weather events. They can also be used in emergency situations and catastrophes, allowing better allocation of emergency personnel and resources.

The Maritime Environment and Technology Centre (Maretec)[1] is one of such groups responsible for research and development in this field. Located in Lisbon, Portugal, it uses its own predictive models and research expertise to provide weather and maritime forecasting. Its most relevant domains are the Portuguese offshore coast and riverbeds, as well the Portuguese continental shelf.

A. Problem

The majority of the population wants a "quick and easy" way to visualise and digest forecast information. With the internet, smartphones created the possibility to gather any information almost anywhere on the globe.

This paradigm of internet access created a series of challenges for web developers [1]. Most screens in mobile devices are smaller than computer screens and forced websites to adapt to these displays. Responsive websites adjust the content to screen size. In the case of maps, the ability to zoom in to more detail might be handy to users but is largely overlooked by some developers.

Maretec currently uses a website not prepared for the new reality of web access[2]. Maps displaying information are based on rasterized images that do not scale properly on mobile devices, driving away users that might access the website on this type of equipment.

B. Goals

The main goal of this work is to provide information access to both users with a mobile device and users with a traditional computer screen. Using modern Javascript libraries it is possible to design responsive interfaces without programming two completely different websites. Another constraint in mobile devices is mobile data usage. Current model outputs sizes are huge and inadequate to use with web pages and need to be adapted.

Our goals can be broken down in two main objectives: parse the models output into a web appropriate format and size and develop a Javascript-based visualization to display forecast data, suited for small screen sizes that allows true zooming.

II. State of the Art

Currently, a user has many ways to access weather forecasts. Besides news broadcast, a very common way are applications in computers and smartphones that display the temperature and general forecast for the day given the user location.

The biggest limitation to this kind of interfaces is the short amount of information they convey to the user. Most of the times the design is just an image with the state of the sky and temperature, either as a current or short term forecast. The Modelo Hidrodinâmico (MOHID) System is an integrated modelling system, developed by Maretec. The MOHID Water and MOHID Land models are used to forecast information of several environmental physical magnitudes on major riverbeds across the Iberian peninsula and to forecast weather and maritime conditions over the Atlantic Ocean and the Portuguese coast and its offshore.

[1] http://maretec.org/

[2] http://forecast.maretec.org

A. Data Formats

Maritime and weather forecasts are usually done by calculating the effect of multiple sea and atmosphere variables, requiring many different computations. The necessity for various scenarios and variables results in large amounts of information to be processed and stored.

The format used by the MOHID models is the Hierarchical Data Format (HDF) storage format. Its hierarchical structure allows accessing datasets in a way similar to file systems used by every major operating system and with a good performance, even for large files.

B. Current Visualization

Our work focus on the "Modelling Maps" section of the Maretec homepage. After selection a domain to visualize, the user can choose the time and magnitude he/she wishes see. At this point, the webpage requests the web server for the image corresponding to the selected options. Figure 1 is a screenshot of this webpage showing a map with a magnitude selected, in this case, "Temperature", for the selected time and domain.

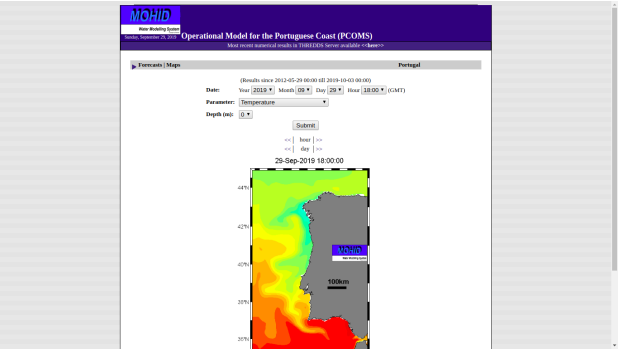


Fig. 1: Example Map showing Water Temperature for the Portuguese Coast Domain on a laptop screen (1366x768px)

Figure 1 displays the webpage on an average laptop screen, with a resolution of 1366 pixels (width) by 768 pixels (height). When opened on a mobile device, it becomes harder to examine and understand the results (Figure 2).

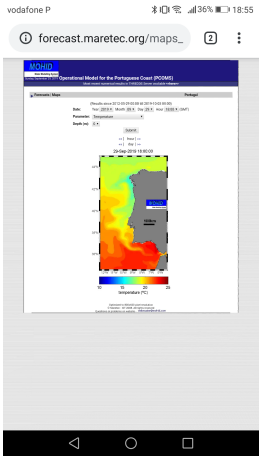


Fig. 2: Example Map showing Water Temperature for the Portuguese Coast Domain on a mobile device screen (1080x1920px)

III. Related Websites

Across the Internet, there are websites, software and services offering weather forecasts and maritime information. The target audience is not always the same: while some websites target the general public, others can have more detailed information for experts or even for scientific purposes. An analysis of their merits and drawbacks, allows us to develop a better visualization.

A. Aguamod Platform

The Aguamod Platform[3] integrates results from high accuracy models with field data. Upon landing, the webpage does not display any information, only a map, centered on Portugal. We can choose which variables to show from the set of available magnitudes, such as Water Temperature and Air Temperature and Direction (Figure 3).

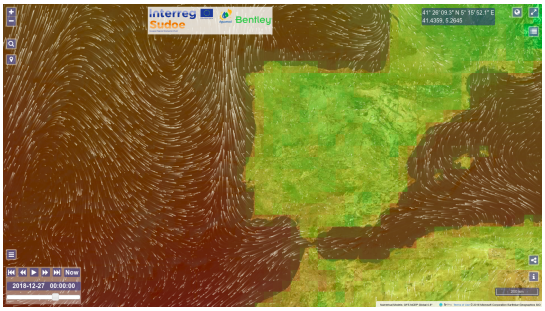


Fig. 3: Aguamod Platform homepage with Air Temperature and Direction selected

Despite the selection being only for air temperature, wind direction is also shown as animated streaklines. These are small animated lines simulating the trajectory of particles carried by the wind (Figure 3). Across all the variables possible to visualize, the platform uses a discrete rainbow colour scheme to represent values, similar to what is used by Maretec Forecast.

B. Portuguese Institute for Sea and Atmosphere (IPMA)

From the website homepage on Figure 4, we can immediately see some of the crucial information on a weather website: the forecast for up to three days.

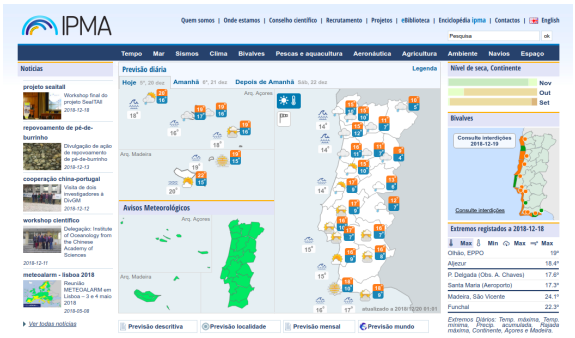


Fig. 4: IPMA Homepage

[3]http://aguamod.maretec.org/

Exploration of the website shows some similarities with Maretec Forecast. The scales, when used, also use a simple colour scheme ranging from blue to red with discrete steps. Both websites use rasterized image formats, such as Portable Network Graphics (PNG) to display information. The colour scheme is slightly different but both use an overlay of arrow glyphs to show wave direction. Also, IPMA does not offer the possibility for users to zoom in an area or interact with the visualization.

C. Ocean Data Laboratory

This visualization is mainly directed at users with scientific background and extended knowledge of the scientific domain. There are many more options and buttons than in other visualizations and a more complex environment, as shown on Figure 5. A discrete rainbow colour scale is used to represent temperature values, similar to other visualizations.

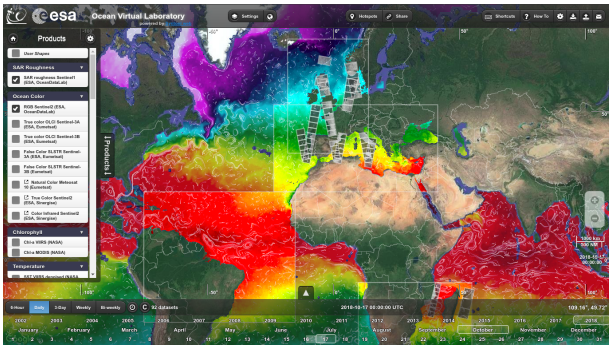


Fig. 5: Ocean Data Laboratory Homepage

D. Wind Guru

Wind Guru[4] is a weather and maritime information forecast website, focused on wind conditions. It is widely regarded as one the best websites amongst surfers and other aquatic sports practitioners to check on sea conditions. In this case, the goal is to inform users if a certain location is favourable to practice a water sport.

This visualization makes use of glyphs and colours simultaneously. Glyphs are the main visual element and colours are a complement to the information provided.

E. Summary

Most websites visited use a rainbow colour scheme to represent magnitude ranges and explain its usage with labels and/or colour scales. The user experiences difficulties understanding a rainbow colour scheme because he/she has yet to develop a particular mental association between colour and value. The colours blue and red are a good example, as they are usually associated with cold and hot respectively [2]. Most of the analysed websites do not make use of other associations, such as ”dark-is-more” and ”opaque-is-more” [3] , that could increase their usability.

Glyph usage is common and standard in most websites. A small coloured glyph is used to display two magnitudes simultaneously. For this work, we use glyphs in the same fashion, with extra caution because of screen sizes.

The visualization we developed attempts to minimize some of the observed issues, such as inconsistent colour scales and unresponsive designs. Scales are kept constant across time and different colours are used across variables. The visualization has few buttons to maintain a simple and easy interaction.

IV. Architecture

In our problem, the models' output is an HDF file that needs to be converted to a web-friendly format, so that it can be used on a Javascript-based interactive map. To process and convert the data we can use a variety of programming languages to create efficient and accurate scripts. Data is then served through a web-server to the visualization, where users can visualize magnitudes over a time period, with a simple interface and consistent colours scales.

A. Data Formats

As one of this work goals was the transmission of data over the internet we needed to use a serializable data format. With usability and standardization in mind, we focused on some of the most common formats used on the internet.

1) XML: Extensible Markup Language (XML)[5] is a markup language, which means it consists of tags describing content. This type of file structure allows any machine or human to understand the contents of a file without any given context since the tags should be self explicative. The use of this format allows communication between disparate systems without the need for each system to parse contents. While human-readable, XML is still a verbose language, full of long and descriptive tags. In the context of this work, where the information is often similar and repetitive, this becomes a problem. It can increase file size and result into large amounts of bytes with repeated information

2) CSV: The concept behind Comma Separated Values (CSV) [4] files is simple: each record is split into fields by commas. So each entry is separated and every computer program could parse this format. While useful for numbers and computations, the use of commas leads to problems with more complex strings and sentences, or even numbers if we use commas as the decimal separator.

For this work, CSV is not appropriate because it lacks metadata. While we might create an application that knows exactly what each field means, this would go against our principle of usability. HDF files contain a description of each field that is not easily translated into the CSV format.

B. JSON

JavaScript Object Notation (JSON) [5] is a file format that serializes information in a human-readable form, without adding unnecessary information. This is achieved with the use of key-value pairs. The use of the Javascript[6] syntax made this format very popular to transfer information over the internet

[4]https://windguru.cz

[5]https://www.w3.org/XML/

[6]http://json.org/

and is now commonly used to gather data from databases and display it on client web browsers. Like XML, this format is self-describing. Each key can be a string explaining the associated value, making it easy to understand and parse.

This format is a good fit for the information contained in HDF files because each dataset can be assigned as an "object" inside the file and have its values associated with an array or dictionary. Its compatibility with most internet browsers helps make this format the best option for this work.

C. Considerations for geospatial data

XML and JSON are self-describing formats, that seem fit for the necessities of the system. But we deal with geospatial data, consisting of an area, defined by its coordinates, and corresponding information. Both formats offer specifications for this specific type of data.

The Geography Markup Language (GML) allows the use of geographic coordinates as an XML extension. It can store geographic entities that are defined by a geometry and the necessary number of properties.

The JSON equivalent of GML is GeoJSON. This format is a regular JSON file, that enforces specific rules to define geographic points or areas, similar to GML. It keeps the same Javascript syntax and definitions and was specified in RFC 7974 [6].

Either of these formats can hold the information retrieved from the HDF file. The main difference between them is the overhead. The tags necessary to define a GML file take up space and make the file verbose. GeoJSON is in common use in similar geographic visualizations and works well with Javascript, allowing a faster and easier development. For these reasons, we selected GeoJSON as the target format for the HDF data conversion.

D. Data Conversion

Once the target data format is chosen, it was necessary to understand how conversion from HDF to GeoJSON had to be done. As defined previously, this process needs to be efficient, as it will be executed every day, multiple times per day. It is also intended to produce a small file with only the necessary information, stripping away irrelevant parts of the HDF file.

1) Matlab: Matlab is a commercial program with a wide range of applications going from data analysis and graphing, to complex artificial intelligence algorithms. The Maretec Forecast currently uses Matlab to create the rasterized images of the forecasts available on the website. Matlab could be used to create and write JSON files since it is a solid and tested program.

Its main problem is the fact that Matlab is a proprietary software, with costly licenses. The use case for this program, in the context of this work, might even be underwhelming when compared with the full extent of functionalities available.

2) Python Script: Python is an object-oriented programming language in development since the early 1990's. It is a language commonly used for introductory courses on programming because of its simple syntax but yet powerful features. The base version of Python does not include many of the mathematical features that Matlab does, but through the open-source community efforts, several libraries complement Python and make it remarkably powerful [7].

Amongst these libraries, it is possible to find H5Py[7], designed to open, read and manipulate HDF5 files. With this reliable and efficient addition, Python became a logical choice for our work.

3) Summary: When comparing these options, we have a clear downside in the usage of Matlab. Its proprietary software and licenses are hard to justify when there is a free option. The capacities of Matlab far exceed this work requirements and it becomes hard to justify such a financial investment. Python Script can accomplish our goals with the same efficiency and stability as Matlab. For all these reasons, we chose to carry out data conversion with Python.

E. Solution Architecture

This work details a data processing and visualization solution to be included in the already existing full solution. Therefore, our options were limited by the current solution constraints and implementation.

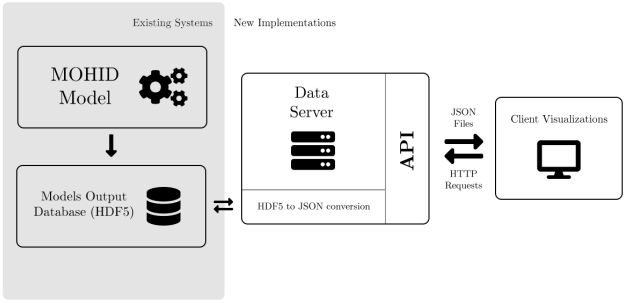


Fig. 6: Solution Architecture

Figure 6 illustrates the solution in place and the changes necessary for the implementation of this work. On the left side of the scheme we can see the MOHID Models and their output database that use to access the output files.

The center of this system is the data server. Here we process the HDF files and convert them into JSON files. The conversion speed is dependent on the resources allocated to this server.

Besides conversion, this server also works as a Web Server, providing an entry point for client visualizations to request information through a Representational State Transfer (REST) Application Programming Interface (API). At this point, the web server processes the client request and serves the desired files. Server responses are JSON files, following the GeoJSON specification.

F. Variable Representation

The most common way to illustrate information is the use of graphs and charts. For this work, information is tied to a

[7]https://www.h5py.org

geographic zone, either a location or an area, so we use geo visualizations, i.e. maps. The MOHID models forecast a series of different magnitudes. For our visualization, there are two types of variables to show: Area and Vectors.

Area variables are scalar magnitudes that can be represented through a squared area on a map, such as temperature. If an adequate colour scale is used, a large amount of small areas can convey information quickly and accurately.

Vector variables cannot be fully represented by areas (e.g. wind direction). One option is to use glyphs to represent direction.

1) Colour: Colour is used on maps as a way to facilitate the visualization and comparison of information. The use of sequential colour scales allows users to understand the increase or decrease of a magnitude in areas.

Defining effective defaults for weather visualizations was the goal set by P. Samuel Quinan and Miriah Meyer [8]. Their investigation resulted in some recommendations for default colour maps and ways to layer independent fields. They do not include a list of colours to use in each case, but instead, explain the reasoning behind the colours they pick. These were based on principles of effective colour usage [9].

Darker colours are associated to higher values of whatever magnitude, a phenomenon called "dark-is-more" bias [3], and clearer colours correspond to lower values. Notable exceptions are some variables such as temperature, as mentioned before, because of the users preconceived ideas [2].

2) Glyphs: Another challenge to overcome when working with weather or maritime forecast visualization is the simultaneous display of multiple variables. Colours can not overlap without meaning being lost, even with transparency. Most visualizations rely on glyphs to solve this problem. Glyphs can take many forms and shapes [10] but in this context they consist of a symbol, usually an arrow, to display the direction of wind or sea current through glyph angle. Glyphs don't have to be always the same size. Intensity of the current, for example, can be shown by glyph size, or its concentration in an area. Figure 7 shows an example of glyph usage with a rainbow scale in the background showing wind intensity.

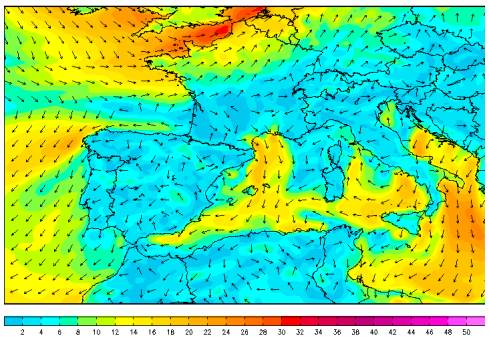


Fig. 7: Wind intensity and direction over South-west Europe, source: freemeteo.com[8]

[8]https://freemeteo.com.pt/tempo/lisbon/mapas/vento/?gid=2267057&language=portuguese&country=portugal

G. Visualization Libraries

Javascript is a very powerful language that allows creating pretty much any type of interactive elements on the web. Its libraries are a set of methods and classes that extend the Javascript core and provide extra functionalities, such as interactive and responsive maps. The user can zoom in and zoom out to inspect certain areas or select options to see more or less information.

1) Leaflet: Leaflet[9] is one of the most used open-source libraries to display and manage maps. Its main strength is the small size of its files and the responsive design it incorporates, making it a strong tool when designing web pages for mobile devices.

The documentation is simple but lacks concrete implementation examples and has an overall steep learning curve. Its "mobile first" approach, makes this library very suited for our work.

2) OpenLayers: OpenLayers[10] base package contains almost every feature usually necessary for this type of visualizations and approaches more faithfully the demands of a scientific oriented system, instead of just providing a base to display maps on a browser. The consequence is that the base package is reasonably large in size, at nearly 1 Mb for version 5.3.0.

This is one of the most complete and mature packages available. It is compatible across every modern browser and mobile devices. Its multiple versions provide some guarantee of its usefulness, but the significant changes in each version somehow complicates development.

3) Google Maps: This API[11] allows developers to incorporate into their website part of Google's platform for maps, navigation and geolocation. The most common usage is a widget with locations and/or pinpoints.

The problems starts when requiring anything beyond such simple tasks. More complex tasks usually rely on a third-party libraries or systems, that are tied to Google's limitations and terms of service. Unlike the previous open-source libraries, the developer is not in control. Ads can show up unexpectedly and the service is no longer free above a set amount of traffic.

For the scientific purpose of this work, commercial API's are not the most viable option precisely because of the clash of interests with commercial tech companies.

4) Map Providers: Another important component of a map library are map providers. These usually deliver satellite images, road maps or a mix of the two.

Map providers play an important part in the subjective beauty of maps. Some designs might be more appealing to users while others may be more effective. Some options for map providers are: Google Maps, OpenStreetMap (community based)[12], Mapbox Studio[13], Stamen Maps[14] and Transas Web

[9]https://leafletjs.com/
[10]https://openlayers.org/
[11]https://cloud.google.com/maps-platform/
[12]https://www.openstreetmap.org/
[13]https://www.mapbox.com/gallery/
[14]https://stamen.com/

Map Services[15].

5) Summary: There are plenty options of Javascript libraries to build a visualization with. Leaflet stands out from the crowd due to its mobile-first approach, allowing us to build a strong and complex visualization with a lightweight library. The tasks visualization has to accomplish are rather simple, because we shifted the weight of the data preparation to the server-side, so the core features of Leaflet (e.g., Zooming, Tile Layers and Drag Panning) are right to accomplish our goals.

V. Implementation

In its core, the process only needs to open an HDF file, iterate its content and save it into a JSON file. While simple, this method results in a JSON file just as big as the original HDF, rendering our whole work pointless. The key to reduce the size of the file is removing redundant information, so we designed an algorithm to remove areas with negligible data, merge others with similar values and implement it with a python script.

A. Models Output

Each MOHID HDF dataset contains a table of cells, where each cell represents a squared area of the domain. Each cell value can be interpreted according to the metadata of each dataset. Information is distributed across datasets and only the analysis of the whole file can reveal the complete set of information. In the following, when mentioning a cell, we refer to a single value within a dataset, but also its corresponding geographical squared area. At the root of each MOHID model output, there are three folders: Time, Grid and Results.

In the ”Time” folder, we can find 25 datasets with the hour, day, month and year of the information.

The ”Grid” folder contains the information about the geography of the domain, such as bathymetry and if whether a cell is underwater or not and two important datasets: Latitude and Longitude. The combination of these datasets allows us to know the coordinates of each map cell.

In the ”Results” folder, there are other folders, where each one contains the information for a single magnitude. Within it, there are usually 25 datasets (one for each hour of day, from 0h to 24h). Each cell contains the value of the magnitude of its corresponding geographical area.

In these files, the information is packed and structured to match the hierarchical nature of the HDF file. To convert it to JSON, it is necessary to pack this information differently. Each JSON object will represent a single cell and pack all the information, from different datasets, in that cell (Figure 8).

[15]https://wms.transas.com/

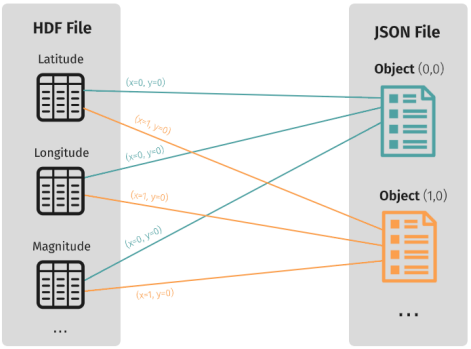


Fig. 8: Simple Schematic of the HDF to JSON transformation

B. GeoJSON Format

GeoJSON structure follows a simple logic: single points are represented by a coordinate pair: latitude and longitude. Areas, such as squares, or more complex regions can be represented by an array of geographic coordinate pairs. Each feature can contain properties, represented by a key-value pair, using the Javascript syntax. This way we can enclose several properties in a single region or point.

C. Data Processing

The approach described, where each cell has a corresponding JSON object results in large files. As an example, a table with 125 by 177 cells creates 22125 objects in a single file, where each object contains the 5 coordinate pairs needed to define it plus the magnitudes.

At this point, an example JSON file converted from a concrete HDF file (Portuguese Coast domain, with 35 Mb) takes up to 32 Mb, for a single forecast hour. This file is inadequate to use on mobile devices or computers with a slower internet connection.

To achieve the efficiency and performance goals we set, we added a processing stage to eliminate redundant data. The first elimination is to clean unused values, such as values of sea-related magnitudes in land. If the script detects such a case, it does not create a JSON object for that cell.

Reducing the number of objects per file allows us to send less data with the same relevant information to the user. To achieve this, we used a simple form of thresholding. As each magnitude might have a different tolerance to this difference, we defined a threshold for each magnitude. This considerably reduces the size of each JSON file, but this is a computationally expensive and time-consuming process.

At the start of each grid line, the algorithm registers the value of the first cell. Then, moves onto its adjacent cell. The values of each cell are compared and, if its difference is lower than the threshold, they are considered as a single region and the algorithm moves onto the next adjacent cell.

This process repeats itself until the difference is over the threshold, in that case, it records the coordinates of the first and last cell belonging to this region and stores it as a single object. To complete the execution, all the lines are iterated. With this

implementation, adjacent similar values will be concatenated into a single JSON object.

To get the coordinates for each concatenated object, we take the coordinates of the first and last cell. In the following line, if the last cell does not end exactly in the same row as the previous the result has a block effect. Each region is exactly a square and the edges between regions are sharp, resulting in a pixelation effect, especially noticeable on the shore (Figure 9 (a)).

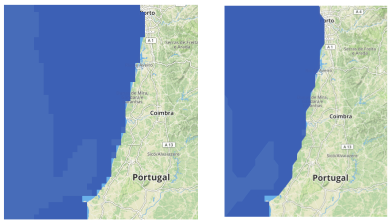


Fig. 9: Illustration of the Square Coastline problem (a) and result of the smoothing algorithm (b)

To get around this problem, we implemented a small fix, based on the rationale of the Marching Squares algorithm [11]. Using a table consisting of binary values, the algorithm defines a region by comparing the four corner values of each cell.

Our implementation takes the value of the last cell in a region and compares it with the cells above and below. If the cell above has a different value than the shape should not be a square but a triangle instead. With this logic, there are three possible outcomes, as shown on Figure 10 (a, b). This solution allows us to have a correct outline of the shore and a much smoother visualization (Figure 9 (b)).

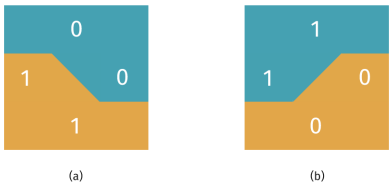


Fig. 10: An edge square can take two shapes (a, b) depending on the values above and below

To compress our data even more, we can make a direct proportion between the cell value and the scale to fit the value in a 0 to 10 range, using a file defining the settings for each magnitude (magnitudes.json). This way, we send to the visualization a small integer value and the client, using the same magnitude setting, can deconstruct the value.

D. Visualization

One of the major problems of the weather forecast websites we assessed is the lack of a default choice for colour and variable representation.

To represent two variables simultaneously we use two options: colour and glyphs. Figure 11 represents the colour scheme that will be used for the “stand-out” variables such as Temperature (a) and Wind Speed (b). Other variables will be represented using different colours, but always with brightness appropriate to their range c) .

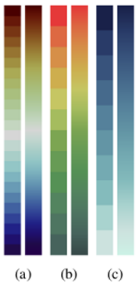


Fig. 11: Colour scheme for Temperature (a), Wind Speed (b). Example scheme for other variables (c). Adapted from Quinan and Meyer [8]

Our choice for colour use is a suggestion based on the available literature [8] [3] [2] [9], but if deemed necessary, it can be changed on the ”magnitudes.json” file mentioned before. This file holds the color scale for each magnitude and can be easily changed or adapted.

Figure 12 displays the developed visualization of Water Temperature for the Portuguese coast, on a laptop screen. We can see the shore outline, as well as part of the colour scale on display.

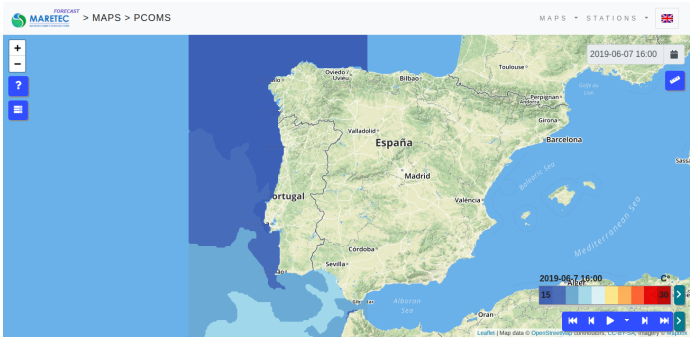


Fig. 12: Implemented Visualization showing Water Temperature for the Portuguese coast

The glyph used is a black arrow, as it can be seen with most colour schemes as background. Sizing is an import aspect for glyphs, because it affects the users’ ability to understand what is being shown. In this visualization, glyphs resize as the user zooms in or out of the map, keeping an appropriate size across devices. Figure 13 displays the visualization as seen on a mobile device.

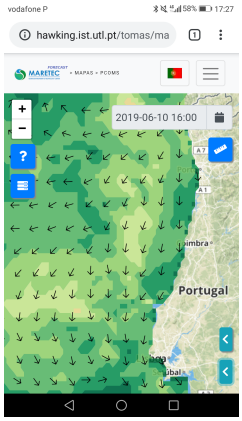


Fig. 13: Implemented Visualization showing Water Current Velocity and Direction for the Portuguese coast, visualized on a mobile device

VI. Testing

Our goal for this work was to create a visualization set for mobile devices. But to achieve it, our development focused a lot on the ability to convert data into a mobile suited format. Therefore, we cannot ignore the need to test and verify the viability of this process. So we split the tests into two different parts, success of the conversion program and the usability of the visualization.

A. Performance

One of the requirement for the data conversion was script efficiency and speed. The script should be able to run on different computers, but the capacity of each influences the execution speed.

To compare the execution speed of the script, we ran the conversion program on two largely different computers: a 2013 laptop and a recent desktop computer. The laptop is a mid-tier computer from 2013, while the desktop is a high end computer suited for scientific computing. Table I compares the two systems, its components and software:

TABLE I: Comparison of specifications for each system where performance was tested

Components	Asus K56b Laptop	Desktop Computer
Processor Model	Intel Core i7-3537U	Intel Core i9-9900K
Processor Speed	2.0 GHz	3.6 GHz
CPU Threads	4	16
RAM	8 Gb	16 Gb
Hard Drive Memory	80 Gb	3 Tb
Operating System	Ubuntu 18.04	Windows 10

For this test, two HDF files, from the same domain and day are used to create the JSON files. These files contain a total of 23 magnitudes, but only 7 will be computed for the current domain, as specified in the domains configuration file. The domain was “PCOMS” and depicts the Portuguese Coast.

In Test 1, we converted temperature, for a single hour, for the first available timeframe. A single JSON file was generated. In Test 2, we used one of the available configurations and convert only a single magnitude: Temperature. A total of 25 JSON files were generated, corresponding to 25 hours of a day (0 to 24 hours).

In Test 3, we computed the 7 aforementioned magnitudes. After conversion, a total of 168 JSON files were created, 24 for each magnitude, each for a single hour of the day. In this test, threading was used to split the computation across the available processor cores available.

Each test was conducted 5 times, at different moments throughout the day. Table II shows the average time, in seconds, for the execution on both computers.

TABLE II: Test 1, Test 2 and Test 3 Results, script execution time on two diferent systems

	Asus K56b Laptop	Desktop Computer
Test 1	1.99 s	0.67 s
Test 2	37.68 s	12.13 s
Test 3*	115.76 s (3 Threads)	10.98 s (15 Threads)

* Used Threading

The first takeaway from these results is the importance of the machine where the script is executed. The difference between systems is significative and so are the results. For test 3, the one closest to how the script will be executed in production, we can see the script runs ten times faster on the test server. This increase can be justified with the number of Central Processing Unit (CPU) threads running simultaneously, as well as the increased processor clock.

For the desktop computer, the full domain (test 3) needs 11 seconds to execute. Extrapolating these results, we can say with confidence that the system will be able to convert all the domains currently forecasted by Maretec every day without limitations.

B. Data Sizes

Each JSON file created contains a single magnitude for a single hour for a unique domain. This approach allows the user to only load the requested information, instead of forcing the download of the full dataset.

Each magnitude creates files with similar sizes, but between different magnitudes, sizes can vary depending on the scale used. Table III lists the average size for each magnitude resulting from the execution of test 3 described in the previous section.

TABLE III: List of data sizes for the generated JSON Files for each magnitude

Magnitude	Single Hour	Full Day
Nitrate	36.9 Kb	886 Kb
Oxygen	42.3 Kb	1000 Kb
Salinity	39.5 Kb	947 Kb
Temperature	86.0 Kb	2241 Kb
Velocity*	480.3 Kb	12341 Kb
Zooplankton	0.05 Kb	1.1 Kb

* Includes Direction and Intensity

To put these results in context, we measured the size of the rasterized image used in the current Maretec Forecast website for the same domain and timeframe as in the previous tests.

The temperature images displayed to the user are 24,1 Kb in size, and with this work, they will be replaced by JSON files with around 86 Kb in size. Despite the increase, we must note that this new file will allow visualization and interaction across all devices. These results prove that the change to the JSON files and interactive map come at the cost of only a couple hundred Kb, a value that can be accommodated by most devices.

C. User Testing

To access the usability for this visualization we used the System Usability Scale (SUS) [12] [13] [14]. SUS consists of a questionnaire with 10 questions each with 5 possible answers, ranging from "Strongly Disagree" to "Strongly Agree". The questionnaire was submitted to twelve participants [15].

Before the questionnaire, each user was asked to complete three different, albeit similar, tasks. "Read the Water Temperature off the coast of Lisbon for today", "Select Salinity Magnitude for the 6th of June of 2019", "Visualize the evolution of the Sea Current magnitude from the 6th of June of 2019 until the 10th of June of 2019". These tasks cover most of the visualization features and were all performed on a mobile phone (Huawei P9 Lite, with a 5" screen).

The SUS average on the 12 subjects was **82**. This score according to the scale proposed by Bangor, Kortum and Miller [16], **classifies as B**. The results also show that 3 users classified the system as barely acceptable. When asked why, two users suggested that the finger motion to zoom was not very natural and presented some problems and had to resort to the buttons on the top left of the screen. The most overall satisfied users claimed the system worked very well and had a more fluid use than other mobile applications.

D. Summary

We can safely say we reached our goals in terms of performance and data sizes. The visualization can be used across devices without limitations and is suited for the large majority of the users of the website. Current Maretec systems will be able to make the transition for the new process without many issues and should convert the data with ease.

Our tests with users proved that our solution is easy to use. Overall, users found the proposed tasks fairly easy to complete and most said they would use the visualization frequently.

VII. Conclusion

This work aimed to create a visualization of MOHID data suited for every type of device, screen aspect ratio and orientation, while keeping in mind the limitations of each device. For smartphones and other mobile devices, this was a challenge not only because of the smaller screen sizes but other factors such as network data rates. Therefore, this work focused a lot on a solution that ensured users only have to download small amounts of information to interact and visualize information.

Using the already existing systems at Maretec, we created a way to convert the MOHID models' output into JSON files that can be used on Javascript-based visualizations. To achieve this, we used Python script to convert HDF files and developed an efficient algorithm, capable of aggregating similar data and discard irrelevant information, while making use of the available hardware.

These files are downloaded by the client visualization which in turn, using Leaflet, displays an interactive map that allows zooming without losing quality. This visualization was integrated into a redesigned Maretec Forecast website, creating a fully responsive experience. Testing allowed us to confirm that files were small and suitable for mobile devices and different networks speeds.

A. Comparison

Over this document, we introduced the system and website currently in place at Maretec. The landing page for both pages are completely different, besides the navigation bar, our visualization now takes up the whole screen and no space is left unused (Figure 14).

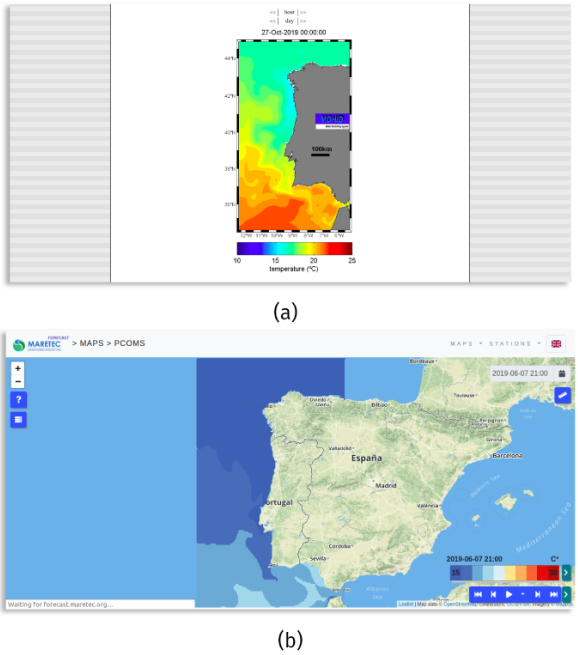
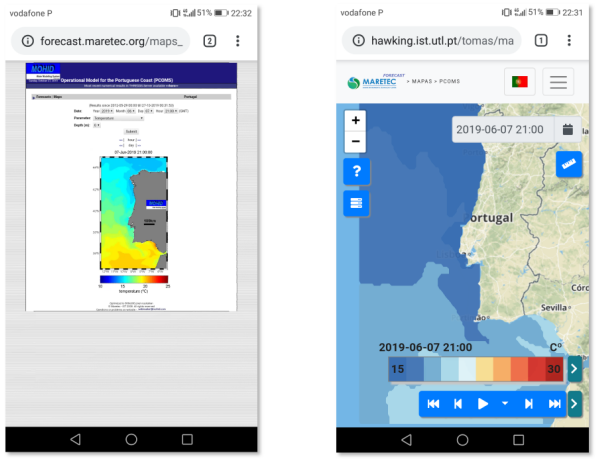


Fig. 14: Comparison of old Platform (a) and new Visualization (b) for the same variable (temperature) and timeframe, as visualized on a 1366px by 768px screen

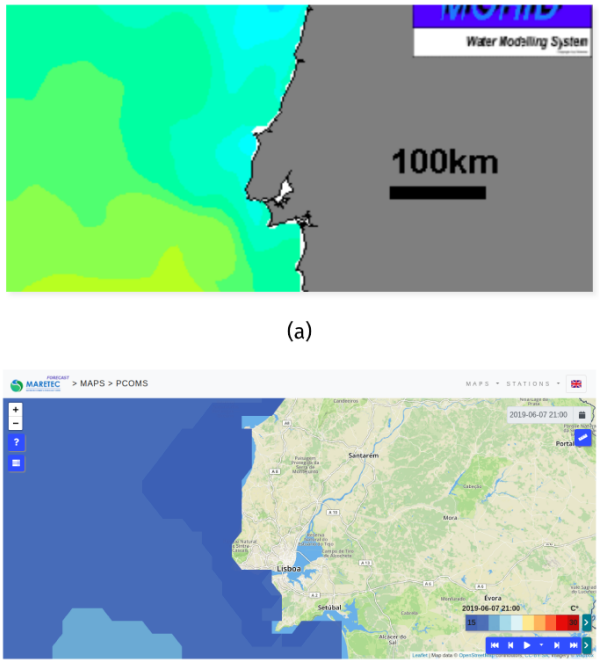
The visualization has the same behaviour on mobile devices, fitting the whole screen. In this case, the improvements are even more noticeable (Figure 15).



(a) (b)

Fig. 15: Comparison of old Platform (a) and new Visualization (b) for the same variable (temperature) and timeframe, as visualized on a 1080px by 1920px screen

One of our goals was to allow users to interact with the visualization without losing accuracy and quality. This was achieved, as users now can use the map controls to zoom on a region without any pixelation effect (Figure 16).



(a)

(b)

Fig. 16: Comparison of the zoom mechanic on the current website using browser embedded zoom (a), and the developed visualization (c) (1366px by 768px screen)

B. Future Work

The development algorithm, while functional, has room for improvement. Currently, the algorithm iterates all cells in a row, looking for values below a threshold. It is possible to change the algorithm to search across rows, reducing, even more, the number of polygons per file and decreasing overall file size. Although not described in this work, we explored this concept and found out that the decrease in file size might not compensate the tremendous increase in computing power necessary. A simple test for this concept took much longer to execute hinting that such improvement to file size might be counter productive. Further exploration needs to be done to confirm this hypothesis.

References

[1] N. C. Zakas, "The evolution of web development for mobile devices," *Communications of the ACM*, vol. 56, no. 4, 2013.

[2] B. Schneider and T. Nocke, "The feeling of red and blue—a constructive critique of color mapping in visual climate change communication," in *Handbook of Climate Change Communication: Vol. 2*. Springer, 2018, pp. 289–303.

[3] K. B. Schloss, C. C. Gramazio, A. T. Silverman, M. L. Parker, and A. S. Wang, "Mapping color to meaning in colormap data visualizations," *IEEE transactions on visualization and computer graphics*, vol. 25, no. 1, pp. 810–819, 2018.

[4] Y. Shafranovich, "Common format and mime type for comma-separated values (csv) files," 2005.

[5] T. Bray, "The javascript object notation (json) data interchange format," 2014.

[6] H. Butler, M. Daly, A. Doyle, S. Gillies, S. Hagen, T. Schaub *et al.*, "The geojson format," *RFC 7946; The Internet Engineering Task Force*, 2016.

[7] T. E. Oliphant, "Python for scientific computing," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 10–20, 2007.

[8] P. S. Quinan and M. Meyer, "Visually comparing weather features in forecasts," *IEEE transactions on visualization and computer graphics*, vol. 22, no. 1, pp. 389–398, 2015.

[9] K. Moreland, "Diverging color maps for scientific visualization," in *International Symposium on Visual Computing*. Springer, 2009, pp. 92–103.

[10] M. O. Ward, "A taxonomy of glyph placement strategies for multidimensional data visualization," *Information Visualization*, vol. 1, no. 3-4, pp. 194–210, 2002.

[11] C. Maple, "Geometric design and space planning using the marching squares and marching cube algorithms," in *2003 International Conference on Geometric Modeling and Graphics, 2003. Proceedings*. IEEE, 2003, pp. 90–95.

[12] J. Brooke *et al.*, "Sus-a quick and dirty usability scale," *Usability evaluation in industry*, vol. 189, no. 194, pp. 4–7, 1996.

[13] A. Bangor, P. T. Kortum, and J. T. Miller, "An empirical evaluation of the system usability scale," *Intl. Journal of Human–Computer Interaction*, vol. 24, no. 6, pp. 574–594, 2008.

[14] J. Brooke, "Sus: a retrospective," *Journal of usability studies*, vol. 8, no. 2, pp. 29–40, 2013.

[15] T. S. Tullis and J. N. Stetson, "A comparison of questionnaires for assessing website usability," in *Usability professional association conference*, vol. 1. Minneapolis, USA, 2004.

[16] A. Bangor, P. Kortum, and J. Miller, "Determining what individual sus scores mean: Adding an adjective rating scale," *Journal of usability studies*, vol. 4, no. 3, pp. 114–123, 2009.