SUM-2017

New Web Technologies for the LHCb Online Monitoring Displays

by Charalampia Lagou| lagou.chara@gmail.com
supervised by Dr. Markus Frank

Geneva, September 2017

Aknowledgements

The author is more than grateful to her supervisor, Dr. M. Frank for sharing his experience with her, for the guidance and the support, as well as for the friendly atmosphere. She is also thankful to the CERN for offering her a place to the Summer Student Programme, which was for her a unique opportunity to work and study in a top-level, international research organisation. Moreover, she owes a debt of gratitude to her university professors, who supported her application to this programme with their reference letters. Her warmer thanks go to her beloved family and T. Tsiampokalos, for always being there for her. Finally, the author wishes to thank her friends for their unconditional support.

Summary

The *LHCb Online Monitoring Displays* is a web application, that gives access to real-time measurements and status information about the LHCb detector and its components, without the need to login. It is hosted at CERN on the computer lbcomet.cern.ch. The system is architecturally complex, based on the Comet technology for the data-transfer and the STOMP protocol for the communication between the clients and the message broker. The application is functional, however concerns are expressed over the future maintenance of the system's architecture as is. The cause of these concerns are firstly the fact that the STOMP JavaScript client package is outdated and flagged by the original author flagged as non-maintained and secondly that todays modern browsers support real-time bi-directional communication which, at the time of development was not compatible even with some of the major browsers. Therefore, the objective of this project is to investigate modern data-push mechanisms, which could complement or replace the already applied technologies. Real-time data-transfer techniques are discussed and an open-source solution that could co-exist with the existing Apache ActiveMQ sever is presented. More specifically, the suggested solution includes the simplification of the system's structure by subtracting the Orbited server, the replacement of XHR long-polling with WebSockets and to replace the STOMP protocol with the MQTT protocol. As far as the data interpretation on the client side is concerned, the Paho JavaScript library is recommended. The changes are implemented, with the student's work focused on the front-end, and the result is functional. In addition, considering the increased use of mobile devices, recommendations for a modern and responsive view of the web application are made. The new design is partially implemented, founded on the latest version of the Bootstrap framework. Even though it is not fully developed, the result is satisfactory but of course, there is room for improvement.

Key Words: LHCb; online displays; data-push; real-time data-transfer; Apache ActiveMQ; WebSockets; MQTT; Eclipse Paho JavaScript client; responsive web design; Bootstrap;

Contents

List of Figures

Abbreviations

AJAX	Asynchronous JavaScript and XML
CERN	European Organisation for Nuclear Research
CMS	C++ Messaging Service
CSS	Cascade Style Sheet
DIM	Distributed Information Management
DOM	Document Object Model
ECS	Experiment Control System
HTML	HyperText Mark-Up Language
HTTP	HyperText Transfer Protocol
JSON	JavaScript Object Notation
JSONP	JavaScript Object Notation with Padding
MQTT	Message Queueing Telemetry Transport
PVSS	Process Visualisation and Control System
RWD	Responsive Web Design
TCP	Transmission Control Protocol
URL	Unified Resource Locator
XHR	XMLHTTPRequest
XHTML	eXtensible Hypertext Markup Language
XML	eXtensible Markup Language
XSLT	extensible Stylesheet Language Transformations

1 Introduction

More than a decade ago the *LHCb Online Monitoring Displays* were designed and since then they are hosted at the lbcomet.cern.ch/Online. The web application gives access to real-time measurements and status information about the LHCb detector and its components. The implementation is based on the Comet technology for the dynamic data transport and uses the STOMP protocol for the communication between the clients and the message broker. The selection of these techniques was mostly based on browser capabilities at the time of development. Despite the fact that the application is functional, concerns are expressed over the future maintenance of the initial architecture. The causes of these concerns are mainly the doubtful upkeep of the already outdated STOMP JavaScript client package and the evolution of web browsers, which nowadays support fully duplex technologies. Thus, the objective of this project is to investigate modern data-push mechanisms, which could complement or replace the already applied technologies. The suggested solution complies with the limitation of working with open-source technologies that could co-exist with the existing Apache ActiveMQ sever. More specifically, it includes the simplification of the system structure, the replacement of XHR long-polling with the WebSockets technology, that allows for real-time data-transfer and the use of MQTT as wire protocol. The student's contribution is focused on the front-end, where the Paho JavaScript browser-based library is added, to achieve the connection with the MQTT broker over WebSockets. In addition, considering the increased use of mobile devices upgrading to a responsive web design is reasonable. The suggested design treats the LHCb collaboration as a brand and therefore, the addition of distinct features, reminding of the collaboration's logo and the consistency between the proposed design and that of other web pages of the collaboration are necessary. The new design is partially implemented and the front-end framework used to add responsiveness is Bootstrap 4. The final product is functional but of course there is room for improvement.

The rest of the report is structured in four chapters deepening into the real-time data-transfer on the web (*Chapter 2*), the responsive web design (*Chapter 3*) and the proposed solutions for the particular case of the *LHCb Online Displays* (*Chapter 4*). The report concludes with a few words about potential future improvements (*Chapter 5*).

2 Real-Time Data-Transfer on the Web

The greatest obstacle of the real-time data-transfer on the web is the application layer protocol, the HyperText Transfer Protocol (HTTP), which is not designed for bi-directional communication, letting only the client to start the communication through a channel that closes after the server's response [1]. Thus, the server is not able to inform the client about any data updates without the client's request and even then, the server may respond only once. This mechanism cannot offer real-time data-transfer and therefore, different tricks and technologies have been used in the past to achieve this. This chapter is focused on these techniques.

2.1 Before AJAX

Before the Asynchronous JavaScript and XML (AJAX), the update of the web page was possible through a new client-request, usually triggered by a button that caused the browser to refresh. The user had then to wait for the web page to reload [2]. The need for dynamic web applications lead to the development of other techniques, mentioned in the next paragraphs.

2.2 AJAX

The interactivity of the web was considerably improved with the appearance of AJAX. AJAX is a browser feature accessible through JavaScript, that allows a script to make an HTTP request to a website behind the scenes (asynchronously), without the need for a page reload. Apart from eXtensible Markup Language data (XML), other types e.g. JavaScript Object Notation (JSON) data may be transferred with this method [3]. To be more precise, AJAX is not a single technology but rather a set of technologies. It incorporates the eXtensible Hypertext Markup Language (XHTML) and the Cascade Style Sheets (CSS) for standards-based presentation, the Document Object Model (DOM) for dynamic display and interaction, the XML and the eXtensible Stylesheet Language Transformations (XSLT) for data interchange and manipulation, the XMLHttpRequest (XHR) for asynchronous data retrieval and the JavaScript language to bind everything together [3]. The concept of AJAX takes the data-update to another level, however the fact that this is still user-triggered is its major drawback [2].

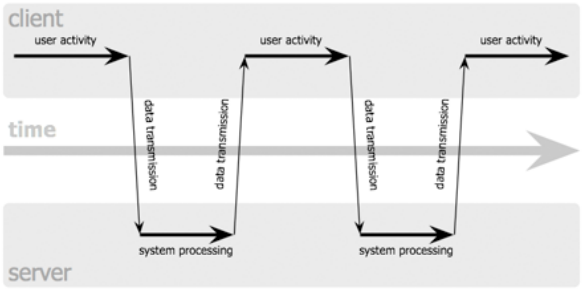

Figure 2.1 – Synchronous interaction pattern of traditional web application [4]

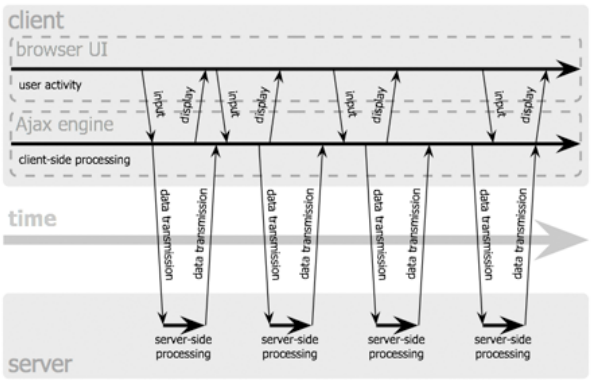

Figure 2.2 – Asynchronous pattern of an Ajax application [4]

2.3 Reverse AJAX

Reverse AJAX is the entire concept of issuing AJAX requests in a way that makes the server send events as response to the client, the soonest possible (low-latency communication) [3]. Reverse AJAX is actually a set of tricks to push content from the server to the client [2]. Such techniques are polling, piggyback, long-polling and streaming.

2.3.1 Polling

Polling is a process of repetitively querying (polling) data. The client sends a request to the server at regular intervals in order to get the server events as soon as possible. This is a mere AJAX request with a polling interval. It could be either a regular HTTP or a JavaScript Object Notation with Padding (JSONP) AJAX request [2], [3]. The latter is used for cross-domain requests [3].

2.3.2 Piggyback

Piggyback polling is a smarter method than polling, since it tends to remove the requests that return no data. There is no interval, the client sends requests to the server when they need to [3]. The main difference is that the server sends back both the answer to the current request of the client and the update about the server events, which the server kept while waiting for the next client request [3], [5].

2.3.3 Comet

Comet[1] is an umbrella term, encompassing multiple techniques to achieve a long-held HTTP request, allowing the server to push data without an explicit request by the client [6]. The comet technology achieves a low-latency communication, meaning that the events arrive to the browser as soon as they are spotted by the server. The communication channel between the client and the server is kept open until a time-out or a server event occurs. When the request is completed another long-lived request is sent [3]. Comet consists of two modes, long-polling and streaming

[1] According to A. Russell the word Comet is not an acronym [7].

[3]. It is underlined though that none of the comet techniques offers a true real-time bi-directional communication between the server and the client [8].

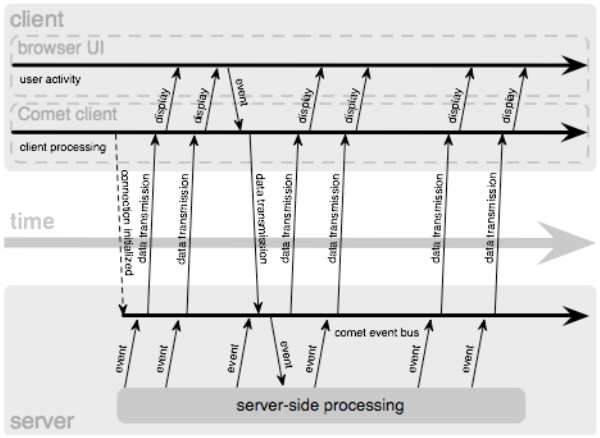

Figure 2.3 – Comet web application model [7]

A - Long Polling

In general, long-polling is a more efficient way of polling. The client submits the same request to the server but the connection is kept open for a longer period of time. As soon as an event occurs, a single response is committed and the connection closes. Immediately a new request is sent to the server and consequently the long-polling connection is re-opened. The client is waiting again for new events to arrive. Long-polling is implemented with either mere XHR (XHR) or multi-part XHR[2] [3].

A.1 - XHR

Regarding the XHR long-polling, the client initiates the connection to the server using an XHR object. The connection is suspended and if an event occurs before the time-out, the server sends back to the client a response. The reception of the response terminates the connection and triggers a new long-lived request to the server [3]. It is emphasised that in this method it is important to check whether the user is still online before sending the response to them. This check is necessary in order to avoid wasting resources on a connection that does not exist any more [9].

A.2 - Multi-Part XHR

In this method the client sends a request to the server and when an update is available the server responds by appending a script tag into the web page. The returned script is executed and another script tag is generated dynamically. This trick suspends the connection [3], [9]. In parallel, there is no way to interrupt it, which is the main disadvantage of this method [3].

B - Streaming

In the streaming mode, one persistent connection is opened. A single request is made and all events are transferred through the same connection. There are two common streaming techniques, the hidden iframe and the multi-part XHR [3].

[2]Multi-part XHR is also known as script tags.

B.1 - Hidden iframe

A hidden iframe[3] is embedded in the web page. The iframe is filled with a chunked HyperText Mark-Up Language (HTML) page, meaning that it has infinite length, and the server exploits the infinity of the page to make a persistent connection and asynchronously send updates to the client, whenever an event has occurred. Specifically, in each event occurrence the server responds with a new script tag, which when executed updates the display [9], [10].

B.2 - Multi-Part XHR

As for the multi-part XHR, the server response contains the "Content-Type: multipart/x-mixed-replace" header of the HTTP protocol, which browsers interpret as a changing document. This means that they are prepared to receive new versions of this document, a fact that keeps the connection alive. In the occurrence of an event the server sends an updated document with which the browser replaces the current one [9].

2.4 SSE

In the context of the Server Sent Events (SSE) the sever may send updates to a web page at any time, without the need to first receive a client request before sending each update, by pushing messages to the web page. The SSE offer only one-way communication, always from the server to the client [11]. The SSE is the technology behind the Facebook notifications and the Twitter feeds [12].

2.5 WebSockets

WebSockets is a technology that enables a bi-directional, full-duplex communication[4] between the client and the server. Such a connection is reliable, remaining open until the window is closed [13] or the connection is lost/stopped by any side. The connection opens with a single HTTP request, called WebSockets handshake, containing special headers [14], asking the server to make an upgraded connection, from the HTTP to the WebSockets protocol. If this upgrade is successful, a persistent connection between the client and the server is made on the Transmission Control Protocol (TCP) level [1]. In this way, the client and the server can exchange data at any time, mutually. Specifically, the server may pass data to the client without prior request by the client, allowing for dynamic content update and the user may also send data to the server [15].

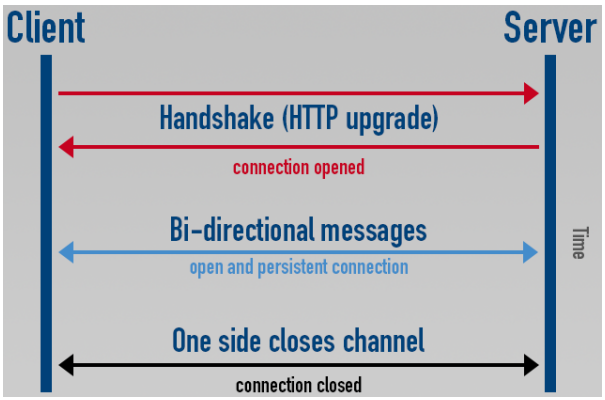

Figure 2.4 – Visual representation of web sockets [16]

[3] The hidden iframe is also known as forever frame [10]

[4] A full-duplex connection allows data flow in either direction. The client can send to the server and the server can send to the client as well [13].

3 Responsive Web Design

In 2016 mobile and tablet internet traffic outpaced the desktop internet traffic for the first time (*Figure 3.1*). In spite of the fact that mobile internet usage continues to grow (*Figure 3.2*), more time on site is still spent on desktop [17]. Therefore, web design should not move its focus to offering only mobile friendly experiences. Modern web design should serve users' needs, supporting each device and browser selection. In other words, web design should be responsive.
The most popular search engine, *Google Search*, takes seriously the importance of responsive design, including device targeting in the ranking criteria of search results [18]. More specifically, *mobilegeddon* was introduced in 2015 with the aim of favouring mobile friendly sites on mobile google search [19].
Thus, responsiveness of the web design is now necessary for a usable and high-ranked web site.

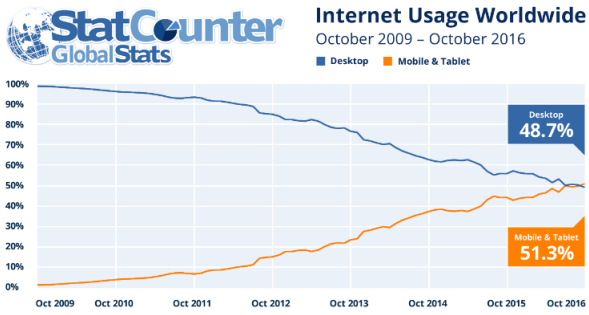

Figure 3.1 – Since 2016 mobile and tablet devices drive more traffic than desktop computers [20].

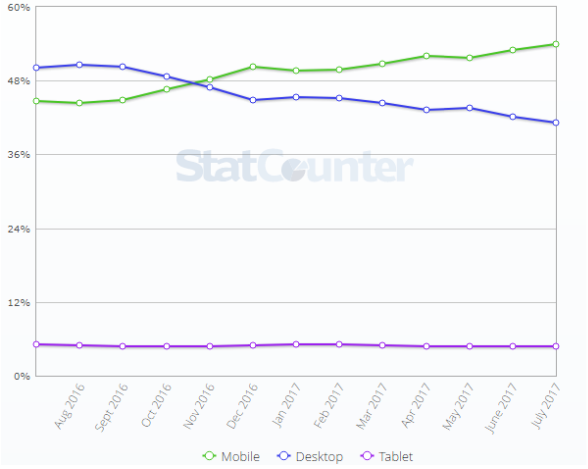

Figure 3.2 – Desktop vs Mobile vs Tablet Market Share Worldwide [21].

3.1 A bit of History

The Responsive Web Design (RWD) was born just a few years ago, in 2010, when the web designer E. Marcotte wrote on *A List Apart* about a combination of three existing techniques that allows for the production of single web pages that adapt to different viewing contexts, sizes and browsers [22], [23], [24].
Before the RWD some developers used JavaScript to perform layout changes to improve page legibility. A primitive example of such a design is the Audi website of 2001, which its designers claim to be the very first flexible web page. The *Razorfish* team used JavaScript to serve up dynamically the small, medium or large layout depending on the user's viewport size [25]. Similar is the work of C. Adams, who created a flexible web page back in 2004. The JavaScript code was capable to transform the single-column web page to a multiple-column one on screens with a width larger than 1024 pixels [26]. The idea of a grid system was introduced in 2007, when the *960 Grid System* was released [22]. By 2008 fluid layouts appeared. In late 2008 media types were

already extended by media queries [22]. Regarding flexible images, they are a result of R. Rutter's experimentation that perfectly complemented the idea of E. Marcotte for the RWD [24]. Around a year after the publication of E. Marcotte's book about the RWD, L. Wroblewski presented a catalogue of multi-device layout patterns [22], [27]. Since 2011 more and more websites become responsive and frameworks are written to ease responsive design. It is apparent that E. Marcotte's description of the RWD had a massive impact on contemporary web design. It is indicative that *Mashable* called the year of 2013 *The Year of the RWD* [22].

3.2 Definition

RWD is a web development concept that makes a single, fluid web page look and behave optimally on all personal computing devices [22], [28], [29].
Until the appearance of this approach separate device-optimised pages were designed for the same purpose. E. Marcotte noted that having disconnected designs, different for each device, platform or browser is unsustainable, as such a solution puts designers under the obligation to produce new versions each time a new browser or platform comes out. He suggested to use a combination of techniques that make the page adapt to users' needs and consequently to the medium that renders them, treating each viewing context as a facet of the same experience [24]. This idea reminds of what another web developer described as adaptable pages back in 2000, while arguing that the web pages should be accessible, legible and pleasant regardless of the browser, platform or screen of the reader and should offer equivalent user experiences independent of the size and the resolution of the screen or the number of colours. Pages should be served in diverse viewing modes apart from the conventional ones such as print, braille browsers [30].
The three core ingredients of RWD are the fluid grid, the flexible images and the media queries. The first component, allows for proportional design by replacing absolute units with relative ones. The second element allows for image sizing in relative units and it prevents overflowing its container. The third ingredient identifies the media type as well as the physical characteristics of the devices and browsers and it conditionally loads style rules, that target the respective viewing context [24]. In practice RWD supports more than just ratio maintenance and layout changes, such as increase of the target area on links for small screens, selectively show or hide elements that might enhance the page navigation and gradually alter the size and leading of the text [23].

3.3 Pros and Cons

RWD is highly advantageous. Most important for social sharing is the use of a single URL for both mobile and desktop versions of the same web page. Moreover, the maintenance of only one web site is both time and money efficient. The fact that this approach does not rely on user-agent detection makes the design more resilient and future-proof. However, this concept is not appropriate for web sites that require great changes in content, layout and functionality for different devices. Also already designed web sites that might take more effort to be transformed into an adaptable page than just creating a dedicated web page for each device cannot benefit from the RWD [31].

3.4 Frameworks

A plethora of responsive frameworks has been developed, among which the leading ones are Bootstrap and Foundation. Bootstrap is actually the most popular one [32]. It was created as internal tool from Twitter developers and in 2011 it was released as open-source. It is by far the most popular responsive framework, bearing almost 115,000 stars on GitHub [33], [34]. Foundation was developed by ZURB and it was released under the MIT license in 2011 as well. It is also quite popular having more than 16,000 stars on GitHub and it is used for big websites such as Facebook, Mozilla, Ebay, Yahoo! and National Geographic [33], [34]. Concerning browser support, both frameworks offer excellent support for the latest versions of the major browsers. Regarding the layout, they have a 12-column grid system and they follow the mobile-first approach [35]. Other less popular responsive frameworks are Skeleton, SemanticUI, Pure etc.

4 Online LHCb Monitoring Displays

The *Online LHCb Monitoring Displays* is a web application for the presentation of real-time measurements and status information about the LHCb detector and its components. In this chapter, the architecture of this system and the web design of its pages are discussed. After the presentation of the current status, suggestions for improvement are made.

4.1 Architecture

The LHCb detector is permanently monitored to ensure its proper functioning. Even though the built-in control system automatically detects malfunctions of its components, manual checks are necessary in order to spot unclassified failures. The *LHCb Online Monitoring Displays* serves as an online platform offering access to information related to the performance of the detector without the necessity to login. In this way collaborators from all over the world may have access to this data [36] and not only researchers who are physically present in the control room of the experiment.
This section is dedicated to the architecture of this system, with an emphasis on the front-end. The weaknesses of the present architecture are discussed and suggestions for a future-proof solution are made.

4.1.1 Current Status

The present implementation of the system starts with the Experiment Control System (ECS). The Process Visualisation and Control System of Siemens (WINCC)[5] was selected for this role. In WINCC the data is organized in unique-named data-points. The *data-point exporter* marks new data as "for publication" and the *publisher* subscribed to the data-points, publishes this data through the Distributed Information Management (DIM). The *feeder* subscribes to updates of the requested data-points when provided by the *publisher*, on behalf of the clients and it forwards all the updates, so that they reach the *web engine*. The C++ Messaging Service (CMS) is used as forwarding mechanism to the Apache ActiveMQ message broker, which is the first component of the *web-engine*. An Orbited server completes the structure of the web-engine. The inter-component communication is made through the Simple Text Oriented Messaging Protocol (STOMP) over TCP. The job of the *web-engine* is to listen to the client requests for HTML pages and status data as well as to the updates of the *feeder* and send them to the subscribed clients. Transferring the updates to the clients works over the STOMP through a JavaScript library within the concept of the Comet technology. More specifically, the

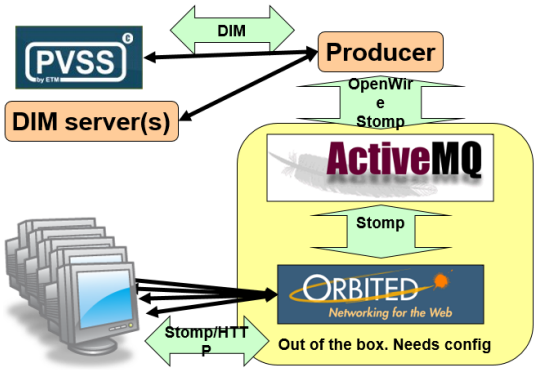

Figure 4.1 – *LHCb Online Displays* | System Architecture [37]

[5] WINCC was formerly called PVSS before the company was integrated in Siemens.

applied Comet-like data-transfer technique is the XHR long-polling [36], [37].
Despite the fact that this particular implementation is still fully functional, structural and technical changes should be made. In more detail, the *web-engine* could be re-structured to reduce complexity, as the same functionality could be supported without the Orbited server. Furthermore, the last version of the STOMP JavaScript library is already outdated and its original author flagged the project as not-maintained [38]. In parallel, there is no coordinated community support present and only individual developers have made contributions [39]. Even though the project was donated to Apache [40] it does not seem to be active. Moreover, the data-transfer technology does not offer real-time data-push, since the client needs to ask regularly for data updates. The reason is that back when this system was initially implemented such technologies were not supported by browsers. Today such data-push mechanisms are compatible with all major browsers.
In the following section, the suggested solution is presented.

4.1.2 Suggested Solution

As mentioned above, the first change to be made is the simplification of the system structure by removing the Orbited server, which will allow the direct communication between the Apache ActiveMQ server and the client. Regarding the data-transfer, apart from the Comet technology, the Apache ActiveMQ engine natively supports WebSockets. The replacement of XHR long-polling with WebSockets is beneficial for the *LHCb Online Displays*, since WebSockets are now supported by all major browsers[6] [41]. WebSockets offer real-time data-transfer. Additionally, the full-duplex communication capabilities of this technique may also be useful for the development of new features in the future. To facilitate client access, Apache ActiveMQ supports both the STOMP and the Message Queueing Telemetry Transport (MQTT) protocol [42] as a higher level protocol based on WebSockets. However, it is already known that at the moment STOMP is outdated due to missing coordinated community support. The alternative is to use the MQTT protocol as mentioned. MQTT is also used for Facebook's Messenger [43], a fact that is reassuring regarding its potential and community support.
Concerning the implementation, the student's contribution is focused on the front-end changes. Changes in the back-end e.g. simplification of system structure by removing the Orbited server, configuring WebSockets in the Apache ActiveMQ server were made by the supervisor.
Front-end client components use the Paho Javascript Client, produced by IBM to implement the real-time data-transfer. This is a browser-based JavaScript library that uses WebSockets to connect to an MQTT broker [44], [45]. The JavaScript implementation that deals with the connection and the subscription to the topics that the server offers, entitled *lhcb.display.data.js* was changed to accommodate the Paho client. The changes were tested on a development server and the final product is functional.

4.2 Web Design

Web design is of paramount importance for a web site like the *LHCb Online Displays*, though the page has no competitors and is targeted to an audience of specialists, interested only in the content

[6] The WebSockets are nowadays supported by Internet Explorer, Chrome, Firefox, Edge, Safari, Opera, even by Android and iOS browsers. The only exception is the Opera Mini browser [41]

and being indifferent to the "cover".
It is essential to invest in usability, select the right color palette, fonts, graphics and layout, since the design is a communication channel and the appearance of each component should reflect the message to convey. The page should be legible and its design intuitive, making navigation fast and effortless. This will create a pleasant user experience.
In the present section the current design of this web page is studied and suggestions for improvements are made. A new design is partially implemented and the resulting page is presented.

4.2.1 Current Status

The initial design of the *LHCb Online Displays* is not based on any kind of framework. CSS rules are written for basic styling, affecting the fonts, the colors and the widths of the elements. The design is not responsive, however, font-sizes and widths as percentages provide with a sense of adaptability. The dynamically produced page adapts pretty well on desktops. However, on mobile devices the web page is squashed into the dimensions of the mobile phone. There was anyway no need to support mobile devices back when the page was created and still many of the interested researchers prefer the desktop view. Nevertheless, optimizing the appearance to match any device would have a positive impact on the page accessibility, as researchers would be able to view the page any time from anywhere, which may be very useful in emergency situations.

Figure 4.2 – Viewing the *LHCb Online Displays - Home* on different devices

Figure 4.3 – Viewing the *LHCb Online Displays - Page 1* on different devices

4.2.2 Suggested Design

The primary goal is to add responsiveness to the *LHCb Online Displays*. Furthermore, considering the LHCb collaboration as a brand, adding some distinct features in the design is necessary. Therefore, a new design is proposed in order to add a sense of consistency between the *LHCb Online Displays* and other pages of the collaboration through the addition of characteristic features from the latter.
For the transition to a responsive design it is suggested to use the latest stable version of a front-end framework. As mentioned above, the most popular ones are Bootstrap and Foundation. Between these two, Bootstrap is selected as the best fit for the particular case of the *LHCb Online Displays*. The two frameworks are real competitors, offering similar features. Therefore, the criterion for the selection is the massive, unparalleled community support of Bootstrap, as well as the fact that it is

actually a Twitter product. The latest version of Bootstrap is used for the production of the new responsive design. Currently Bootstrap 4 is in a beta version, which is more or less stable.
As far as the distinctive design is concerned, effort is put into adding characteristic features e.g. colours, icons, that remind of the LHCb collaboration. In order to be consistent with other official pages of the experiment, related web pages were studied. The suggested design is inspired by that of the collaboration homepage and its colour palette consists of colours that match the LHCb logo. The figures below are screenshots of the suggested design of the *LHCb Online Displays* home page.

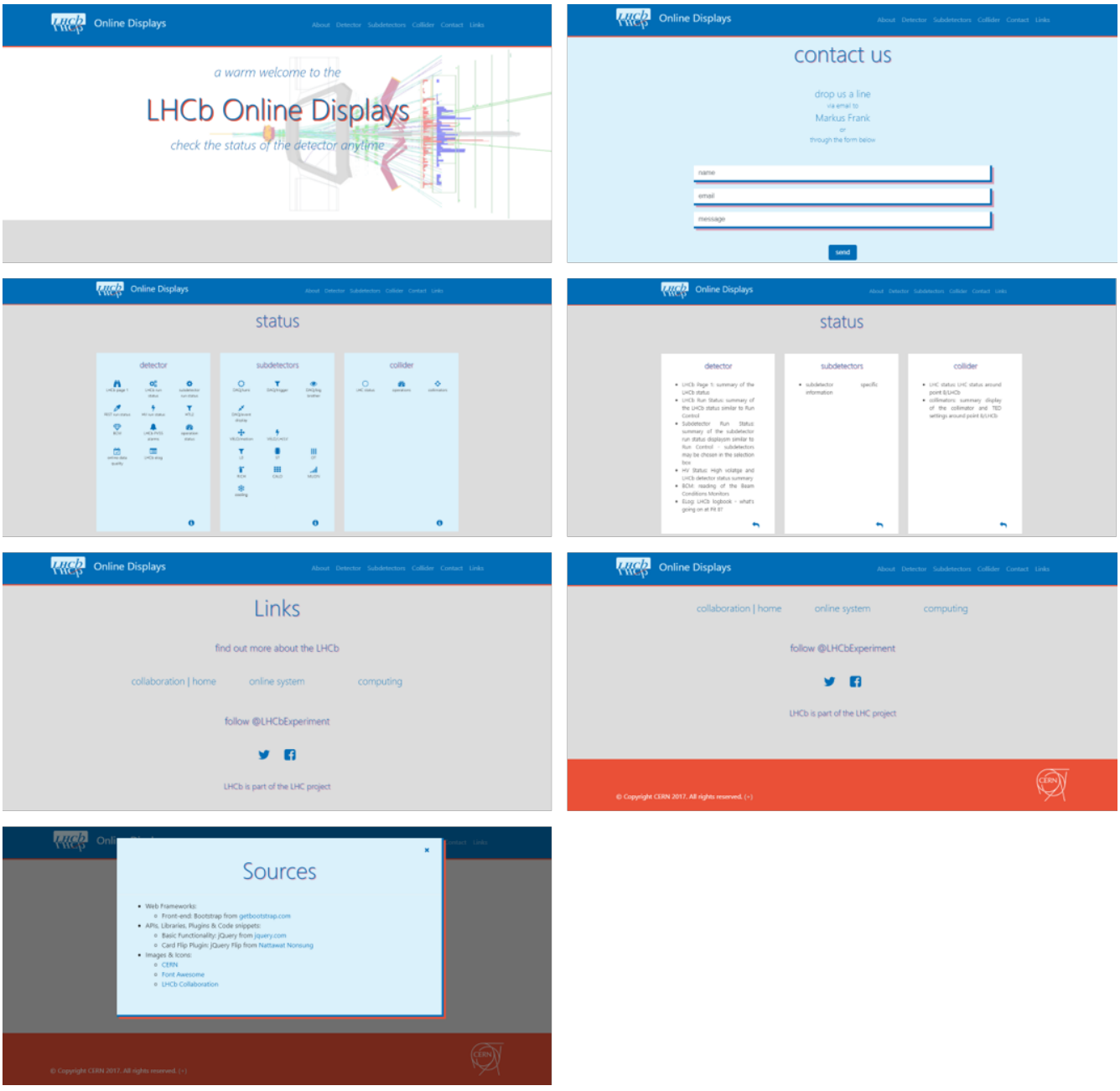

Figure 4.4 – Suggested design | Section of the *LHCb Online Displays - Home*

Despite the fact that the home page design is responsive and modern (*Figure 4.5*), the presentation

of the status-pages is a bit problematic (*Figure 4.6*). In more detail, they appear as replacement of the respective section of the home page when their view is selected. As in the original design of the page, the status-pages are added through iframes. The status-pages are originally not responsive and therefore they do not comply well with the rest of the design. The layout could definitely improve exploiting the framework's responsive grid and flexible breakpoints to replace the table structure of the status-pages. Responsiveness of the iframe content design would certainly remove the x-axis scrollbar but it would not affect the y-axis scrollbar. The reason is that the content page is produced dynamically and the returned height corresponds to part of it. Probably, using a plugin, such as the *iframe-resizer* plugin [46], could be a solution. Otherwise, subtracting the iframe and re-writing the code that generates the status-pages in order to just replace the HTML of the respective page section may be efficient.

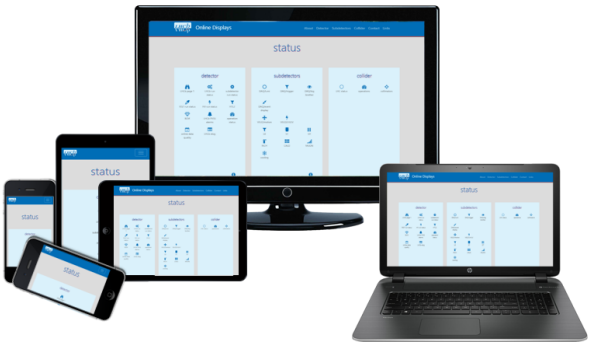

Figure 4.5 – Suggested design of the *LHCb Online Displays - Home* on different devices

Figure 4.6 – Scrollbars appear on the x-axis and/or the y-axis of the embedded status-pages

5 Conclusions and Future Work

The *LHCb Online Displays* is an open-access, web system providing information about the status of the LHCb detector and its components. The system was designed more than a decade ago and since then, some of the used packages became outdated and at the same time web browsers evolved allowing for real-time bi-directional communication. Therefore, the objective of this project is to investigate new web technologies that could complement or replace the already applied ones. Data-push mechanisms for the web are discussed and an open-source solution, compatible with the existing Apache ActiveMQ server is presented. Precisely, the solution includes the simplification of the system's architecture by subtracting the Orbited server, the replacement of the almost real-time data-transfer through Comet (XHR long-polling) with WebSockets, the adoption of MQTT as wire protocol instead of the STOMP and the use of the Paho JavaScript client library. The implementation of these changes is successful, with the student's work focused on the front-end. Additionally, renewing the web page layout is suggested, with the aim of adding responsiveness to the design. The new design is partially implemented, founded on the Bootstrap framework. The final product is functional, however the appearance of the embedded, non-responsive status-pages does not comply well with the rest of the page. Hence, improvements should be made towards this direction in the future.

More specifically, future work should be concentrated over the replacement of the table structure of the status-pages, exploiting Bootstrap's responsive grid and flexible breakpoints. Moreover, the computation of the iframe's full height after its load is of major importance for the appearance of the integrated status-pages. This seems not to work properly, since the status-pages are constructed dynamically and the returned height represents just a part of them. This issue could be possibly solved either by using a plugin e.g. the *iframe-resizer* or by subtracting the iframe and in parallel, reconstructing the code that generates these pages, so that it replaces the HTML of the respective section in the home page instead of constructing a full HTML page. Furthermore, changes on the fonts and the colour-palette should be made on the status-pages to match those of the new home page. Finally, it is noted that user testing should follow and corrections should be made based on the given feedback before releasing the final version.

References

[1] Frank, M. *WebSocket versus Comet – Real-Time Web Applications*. Online, Published on pointsoftware.ch, 2013. Retrieved 4th of September 2017.

[2] Sprocketonline. *Exploring Reverse AJAX*. Online, Published on gmapsdotnetcontrol.blogspot.de, 2006. Retrieved 4th of September 2017.

[3] Carbou, M. *Introduction to Comet*. Online, Published on ibm.com/developerworks, 2011. Retrieved 4th of September 2017.

[4] Garrett, J. *Ajax: A New Approach to Web Applications*. Online, Published on web.archive.org, 2005. Retrieved 4th of September 2017.

[5] DirectWebRemoting. *Reverse AJAX*. Online, Published on directwebremoting.org, n.d. Retrieved 4th of September 2017.

[6] Wikipedia, the Free Encyclopedia. *Comet (programming)*. Online, Published on wikipedia.org, 2017. Retrieved 4th of September 2017.

[7] Russell, A. *Comet: Low Latency Data for the Browser*. Online, Published on infrequently.org, 2006. Retrieved 4th of September 2017.

[8] Pusher. *What are WebSockets?*. Online, Published on pusher.com, 2017. Retrieved 4th of September 2017.

[9] Prusty, N. *Push Technology In PHP*. Online, Published on qnimate.com, 2017. Retrieved 4th of September 2017.

[10] Egli, P. *Comet, HTML5 WebSockets: Overview of Web Based ServerPush Technologies*. Online, Published on peteregli.net, 2017. Retrieved 4th of September 2017.

[11] Mozilla Developer Network. *Server-sent Events*. Online, Published on developer.mozilla.org, 2017. Retrieved 4th of September 2017.

[12] w3schools.com. *HTML5 Server-Sent Events*. Online, Published on w3schools.com, 2017. Retrieved 4th of September 2017.

[13] Xiang, D. *What is a Web Socket?* [Video]. Online, Published on youtube.com, 2015. Retrieved 4th of September 2017.

[14] Carbou, M. *WebSockets*. Online, Published on ibm.com/developerworks, 2011. Retrieved 4th of September 2017.

[15] Mozilla Developer Network. *WebSockets*. Online, Published on developer.mozilla.org, 2015. Retrieved 4th of September 2017.

[16] Hanson, J. *What are WebSockets*. Online, Published on pubnub.com, 2013. Retrieved 4th of September 2017.

[17] Stone Temple. *Mobile vs Desktop Usage: Mobile Grows But Desktop Still a Big Player*. Online, Published on stonetemple.com, 2017. Retrieved 1st of September 2017.

[18] Google Webmaster Central Blog. *Continuing to make the web more mobile friendly*. Online, Published on webmasters.googleblog.com, 2016. Retrieved 1st of September 2017.

[19] Wikipedia, the Free Encyclopedia. *Mobilegeddon*. Online, Published on wikipedia.org, 2017. Retrieved 1st of September 2017.

[20] StatCounter. *Mobile and Tablet Internet Usage Exceeds Desktop for First Time Worldwide*. Online, Published on gs.statcounter.com, 2016. Retrieved 1st of September 2017.

[21] StatCounter. *Desktop vs Mobile vs Tablet Market Share Worldwide*. Online, Published on gs.statcounter.com, 2017. Retrieved 1st of September 2017.

[22] Wikipedia, the Free Encyclopedia. *Responsive Web Design*. Online, Published on wikipedia.org, 2017. Retrieved 1st of September 2017.

[23] A List Apart. *Responsive Web Design*. Online, Published on alistapart.com, 2010. Retrieved 1st of September 2017.

[24] Marcotte, E.. *Responsive Web Design*, 2011. A Book Apart. New York, New York.

[25] Jim Kalbach. *The First Responsive Design Website: Audi (circa 2002)*. Online, Published on experiencinginformation.com, 2012. Retrieved 1st of September 2017.

[26] Adams, C. *Varying Layout according to Browser Width*. Online, Published on themaninblue.com, 2004. Retrieved 1st of September 2017.

[27] Wroblewski, L. *Multi-Device Layout Patterns*. Online, Published on lukew.com, 2012. Retrieved 1st of September 2017.

[28] Mozilla Developer Network Web Docs. *Responsive Web Design*. Online, Published on developer.mozilla.org, 2015. Retrieved 1st of September 2017.

[29] Mozilla Developer Network Web Docs. *Responsive Design vs Adaptive Design*. Online, Published on developer.mozilla.org, 2016. Retrieved 1st of September 2017.

[30] A List Apart. *A Dao of Web Design*. Online, Published on alistapart.com, 2000. Retrieved 1st of September 2017.

[31] Mozilla Developer Network Web Docs. *Responsive Design*. Online, Published on developer.mozilla.org, 2013. Retrieved 1st of September 2017.

[32] Bootstrap. *Home*. Online, Published on getbootstrap.com, 2017. Retrieved 4th of September 2017.

[33] GitHub. *Most Starred Repositories*. Online, Published on github.com, 2017. Retrieved 4th of September 2017.

[34] SitePoint. *The 5 Most Popular Frontend Frameworks of 2017 Compared*. Online, Published on sitepoint.com, 2017. Retrieved 4th of September 2017.

[35] Blankenship, W. *Bootstrap vs. Foundation: Which Framework Is Right for You?*. Online, Published on upwork.com, 2017. Retrieved 4th of September 2016.

[36] Frank, M. and Gaspar, C. 2012. An Information System to Access Status Information of the LHCb Online. *Journal of Physics: Conference Series*, 396(6), p. 062007.

[37] Frank, M. 2011. *Online Displays for the World*. Presentation at the JCOP Meeting, Geneva, Switzerland.

[38] Mensil, J. *Stepping Out From Personal Open Source Projects*. Online, Published on jmesnil.net, 2015. Retrieved 4th of September 2017.

[39] Mensil, J. *Stomp.js*. Online, Published on github.com, 2015. Retrieved 4th of September 2017.

[40] Mesnil, J. *Code donation for stomp.js*. Online, Published on nabble.com, 2016. Retrieved 4th of September 2017.

[41] Can I use.... *Web Sockets*. Online, Published on caniuse.com, 2017. Retrieved 4th of September 2017.

[42] Apache ActiveMQ. *WebSockets*. Online, Published on activemq.apache.org, n.d. Retrieved 4th of September 2017.

[43] Zhang, L. *Building Facebook Messenger*. Online, Published on facebook.com, 2011. Retrieved 4th of September 2017.

[44] Eclipse. *Building Facebook Messenger*. Online, Published on eclipse.org, n.d. Retrieved 4th of September 2017.

[45] IBM. *Eclipse Paho JavaScript client*. Online, Published on github.com, 2017. Retrieved 4th of September 2017.

[46] Bradshaw, D. *iFrame-Resizer*. Online, Published on github.com, 2017. Retrieved 4th of September 2017.