



UNIVERSITETET I AGDER

*Development and Testing of Web GUI Application for the LHCb
VELO Data Quality Monitoring System*

by

Pavlo Prykhodko

Master Thesis in
Information and Communication Technology

Final Version

University of Agder

Meyrin, December 4, 2013

Abstract

A great interest of IT engineers at CERN is to simplify the access to the Data Quality Monitoring (DQM) applications that usually lay behind several layers of security firewalls. In order to make it simple and thus help to save time for the scientist who rely on this data, additional application for the Web had to be developed and tested. The goal of this thesis work was to develop such a Web DQM application for CERN. First, a Web Graphical User Interface (GUI) was developed. In parallel, an Apache server was installed and configured for testing. Moreover, software program called ROOTJS that processes and displays CERN data files on the Web was presented. Through this thesis project, new functionalities were developed to meet the requirements. Furthermore, the ROOTJS program was merged with the Web GUI application and series of tests were performed to showcase the capabilities of the application which was developed through this thesis work.

Preface

This thesis is the result of the Master's Thesis IKT-590 course that fulfils the requirements of fourth semester content at the Faculty of Engineering and Science, University of Agder (UiA), Grimstad, Norway. It was written externally at the European Organization for Nuclear Research (CERN) facilities in Meyrin, Switzerland. The project was carried out in a period from March 1, 2013 to December 4, 2013 and its workload equals to 30 ECTS. The author has completed the main goal of the project Development and Testing of Web GUI Application for the LHCb VELO Data Quality Monitoring System.

I would like to express my gratitude to the thesis supervisor Professor Frank Y. Li from UiA for his guidelines and assistances in giving feedback on both technical aspects of the thesis project and content of the thesis. Under his supervision a lot has been learned about the project and the thesis writing. I enjoyed many Skype conversations we had, even though we were present at different locations.

I would like to thank the external thesis project supervisor Paula Collins from CERN for her support, motivation and overall help during my stay at CERN. Under her supervision a lot has been learned about the LHCb VELO detector. This knowledge helped to get a better understanding of this thesis work.

I am also grateful to the team members of the main project Marco Gersabeck, Manuel Tobias Schiller, Michal Adam Wysokinski and Suvayu Ali for their constructive suggestions, motivation and feedback. A special thanks goes to Bertrand Bellenot for providing the ROOTJS program' source code and his support, and to Oscar Augusto de Aguiar Francisco for the important technical suggestion that helped to turn this thesis project into the right direction.

Last but not least, I am thankful to my mother Tanja Aaring and to my fiancée Olya Serediak for their patience, understanding and support throughout this difficult year.

Meyrin, December 4, 2013

Contents

Contents	2
List of Figures	5
List of Tables	8
1 Introduction	9
1.1 Background and motivation	9
1.1.1 CERN, LHC and experiments	10
1.1.2 LHCb experiment	11
1.2 Problem statement	11
1.3 Problem solution	12
1.4 Thesis organization	13
2 System and Technology Background	15
2.1 System and configuration	15
2.1.1 LHCb VELO	15
2.1.2 Setup overview	16
2.2 The VELO Web GUI application fundamentals	17
2.2.1 ROOTJS	17
2.2.2 GUI prototype	19
3 Framework for the VELO Web GUI Application Design	21
3.1 ROOT	22
3.2 Tools for VELO Web GUI design	23
3.2.1 Python	23

CONTENTS

3.2.2	Django Web framework	25
3.2.3	JavaScript	26
3.2.4	Apache HTTP (Web) server	26
3.2.5	mod_wsgi	26
3.2.6	X-Win32	27
3.2.7	PyCharm	28
3.3	VeloMoniGUI	29
4	Requirements and Design	31
4.1	Requirements	31
4.2	Design	32
4.2.1	Primary features	33
4.2.2	Auxiliary features	36
5	Solutions and Implementations	37
5.1	Server configuration	38
5.1.1	Preparing the VM	39
5.1.2	Configuring the VM	40
5.2	GUI development	41
5.2.1	Overview	41
5.2.2	Layouts	42
5.3	Development of solutions based on the ROOTJS program	47
5.3.1	Developed solutions for index.htm	47
5.3.2	Developed solutions for JSRootInterface.js	49
5.4	ROOTJS into GUI integration	50
6	Tests and Demonstrations	52
6.1	Test scenarios	52
6.2	Performed tests	52
6.2.1	Verification of implemented functionality	53
6.2.2	Browsers tests	55
6.2.3	Basic ROOT functionality tests	57
6.2.4	Auxiliary features tests	58
6.3	Numerical results	60

CONTENTS

6.3.1	Preparation	60
6.3.2	Results	61
6.4	Alternative solutions	62
7	Discussions	64
7.1	ROOTJS alternatives	64
7.2	Apache alternatives	65
8	Conclusion and Future Work	67
8.1	Summary of the thesis work	67
8.2	Contributions	68
8.3	Future work	68
	Bibliography	70
	Appendix A Configuration of the httpd.conf File	72
	Appendix B Configuration of the WSGI File	74
	Appendix C Configuration of the setting.py File	75
	Appendix D Adjusting Correct Paths	76
	Appendix E Developed Solutions for the CSS files	78

List of Figures

1.1	LHC map [3].	10
1.2	LHCb and VELO.	11
1.3	DQM applications and frameworks.	12
2.1	42 VELO modules [6].	16
2.2	Final setup diagram.	16
2.3	Zoom functionality in ROOTJS graphs.	17
2.4	The GUI of the ROOTJS program.	18
2.5	Core files of the ROOTJS program.	19
2.6	Prototype GUI [10].	20
3.1	Examples of ROOT graphs.	23
3.2	Default Python interpreter.	24
3.3	Typical default Django admin page.	25
3.4	X-Win 2012 configuration panel.	27
3.5	Default PyCharm IDE.	28
3.6	Overview of the VeloMoniGUI [18].	29
4.1	The VELO Web GUI application interface.	32
4.2	The VELO Web GUI application areas.	33
4.3	Tabs highlighting.	34
4.4	Navigation bars.	34
4.5	Sensor View tab.	35
4.6	ROOT file path window.	36
4.7	Collapsible and delete buttons.	36

LIST OF FIGURES

5.1	Implementation of the VELO Web GUI application.	37
5.2	Server Location.	38
5.3	VM preparation flow chart.	39
5.4	VM preparation flow chart.	40
5.5	Django Web GUI files.	41
5.6	The VELO Web GUI application structure diagram.	42
5.7	bbase.html inheritance diagram.	43
5.8	A piece of bbase.html code.	43
5.9	Sensor View template.	44
5.10	Inheritance blocks of <i>specialanalyses.html</i> file.	45
5.11	Django views diagram.	45
5.12	Views.	46
5.13	Part of the tabs code.	46
5.14	Django implementation.	47
5.15	Developing functions based on the <i>index.htm</i> file.	48
5.16	Part of the sensorview.html code.	48
5.17	if-statements.	49
5.18	Delay.	50
5.19	Graphs display.	50
5.20	Integration overview.	51
6.1	Active tab highlighting and Main tab title functions.	54
6.2	ROOT file path window.	54
6.3	Zoom in, zoom out of ROOT histograms.	55
6.4	Tabs comparison.	56
6.5	Zoom functionality in the VELO Web GUI application.	58
6.6	The collapsible feature.	59
6.7	The delete feature.	59
6.8	Execution start time.	60
6.9	Execution end time and the result difference.	60
6.10	Alert message in Chrome.	61
6.11	Tab navigations.	62
A.1	<i>httpd.conf</i> file.	73

LIST OF FIGURES

B.1	<i>youprojectname.wsgi</i> file.	74
C.1	Databases field in the <i>setting.py</i> file.	75
C.2	ROOT static path in the <i>setting.py</i> file.	75
D.1	<i>JSRootInterface.js</i> file.	76
D.2	Template file configuration.	77
E.1	CSS file.	78

List of Tables

5.1	Program versions.	40
6.1	Virtual Machines.	53
6.2	Browsers compatibility with the ROOTJS program.	55
6.3	Tested browsers.	56
6.4	Performance results.	61

Chapter 1

Introduction

In this thesis project, we present the development and testing of the Web GUI application for Data Quality Monitoring (DQM) of the Large Hadron Collider beauty Vertex Locator (LHCb VELO) detector. We refer to a program that was designed for reading specific data files called ROOT¹ files, and displaying their content in a Web browser. Then, we use it as a platform for the development of new functionalities to meet our requirements for the final application. This project is also part of a bigger project with the title "DQM Application Extension and Upgrade for the LHCb VELO detector". It is important to mention that we use the name "Web GUI" to refer to our application in the development stage. The final application is called "VELO Web GUI application".

1.1 Background and motivation

In this section, the background and motivation of our thesis work is presented. We mention the connection of the thesis project to the biggest laboratory in the world. We also introduce the reader to the Large Hadron Collider and the main experiments that are located on it. However, the main focus will be on the experiment that is directly related to our thesis work. At the end of this section, the problem statement and the problem solution are covered.

¹ROOT is a software program that is widely used at CERN for data storing, analysing and more. It is presented in greater detail in Section 3.1

1.1.1 CERN, LHC and experiments

The European Organization for Nuclear Research (CERN) was established in 1954 to operate the world's biggest particle physics laboratory. The name CERN comes from French acronym "Conseil Européen pour la Recherche Nucleaire" that in English means the European Council for Nuclear Research. In 1952, this provisional body was concentrated mainly on research of the atom's inner part, that is why the name "nuclear" was used. CERN headquarters are based north-west for Geneva, Switzerland. CERN has 20 member states and 7 observer states, as of Autumn 2013.

At CERN both physicists and engineers are searching for answers on how our Universe was built and what was its fundamental structure. To study the fundamental particles, they built the world's largest and very complex tools. Purpose-built particle accelerators and detectors are used to make particle collisions at close to the speed of light. The collisions data is observed, recorded and stored for further analyses using state of the art computer facilities. Such interaction between the particles can give the insights to the fundamental laws of nature [1].

The Large Hadron Collider (LHC) is located on a border between Switzerland and France 175m under the ground and lies in a tunnel 27km in circumference, as shown in Fig. 1.1.

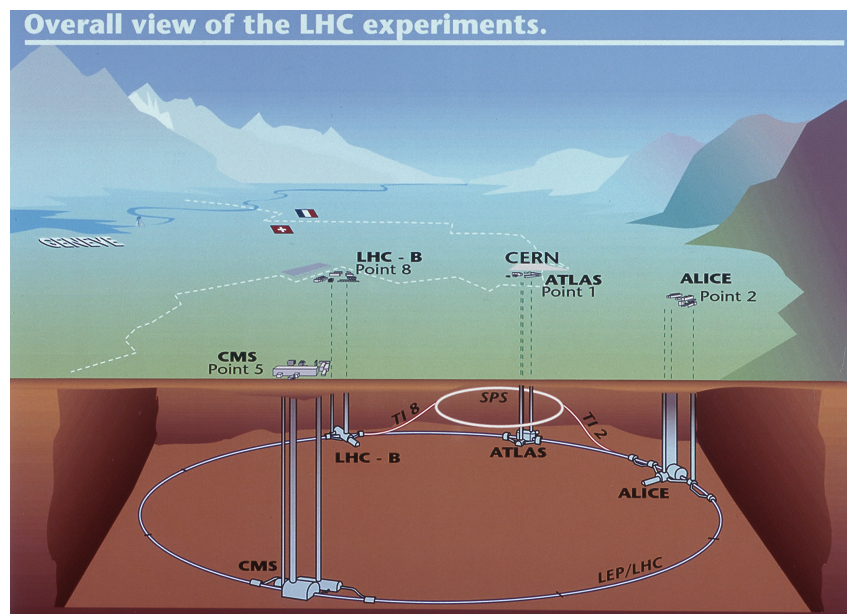


Figure 1.1: LHC map [3].

It is the world's largest and most powerful particle accelerator. ALICE, ATLAS, CMS and LHCb are 4 biggest experiments at CERN which are located on the LHC ring [2].

1.1.2 LHCb experiment

The LHCb experiment is located 100 m below the surface. Every layer of LHCb is designed to identify and measure a different aspect of the particles flying out from the collision, as shown in Fig. 1.2 a) and b). It was built to study the slight asymmetries between matter and antimatter with the help of particles called beauty quarks. These particles were common in the aftermath of the Big Bang, but they disappeared from the Universe today. LHCb generates billions of them together with their antimatter counterparts, the anti-beauty quarks. By studying the slight differences in decay between the beauty quarks and their antiparticles to unprecedented precision, the LHCb experiment is shedding light on one of the Universe's most fundamental mysteries [4].

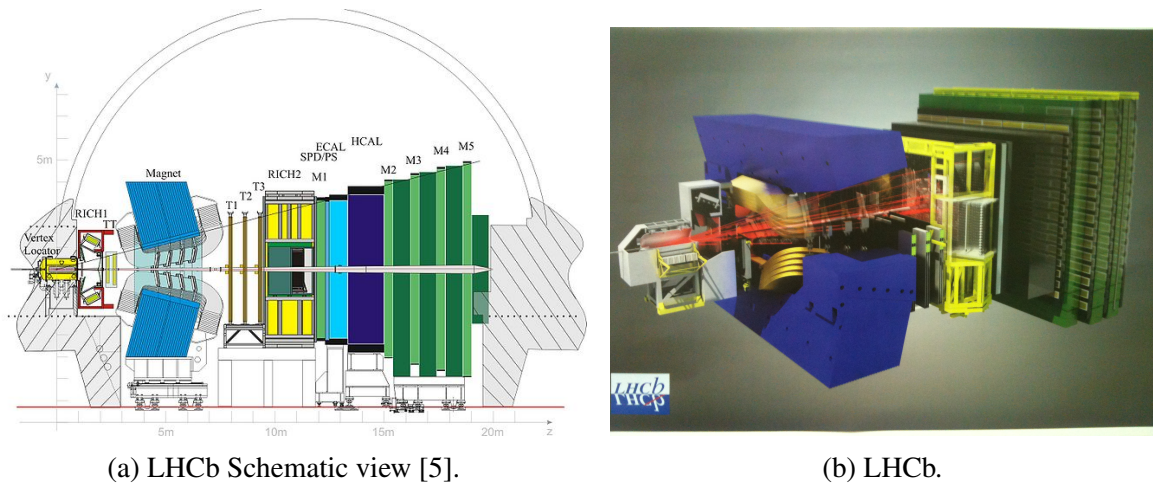


Figure 1.2: LHCb and VELO.

1.2 Problem statement

This section covers the goal of our project thesis and also our challenges and tasks.

The main goal of this thesis project is to develop and test a Web GUI application that can read, process and display ROOT graphs. This raises up a number of challenges and tasks. We have to find a Web framework program to build our Web GUI. We also have to develop the Web GUI that should contain as many familiar features of the previous standalone DQM application as possible. A ROOT program that was designed for the Web use is required. It should be possible to merge it seamlessly into the Web GUI program. Moreover, the application has to be tested on the performance and stability. At the same time, we are interested to construct this application that later can be easily integrated into the bigger project.

1.3 Problem solution

In this part of the chapter, the solutions to the challenges are presented along with the tasks that were mentioned in the previous section.

The Web version of the DQM application saves time for the scientists who rely on the monitored data. Our thesis project is a part of the bigger project that is divided into the number of individual parts, as shown in Fig. 1.3. The blocks that are marked with red color represent our thesis work. We develop a Web GUI application using Django Web Framework. We also develop new functionality for the ROOTJS program that handles ROOT files for the Web.

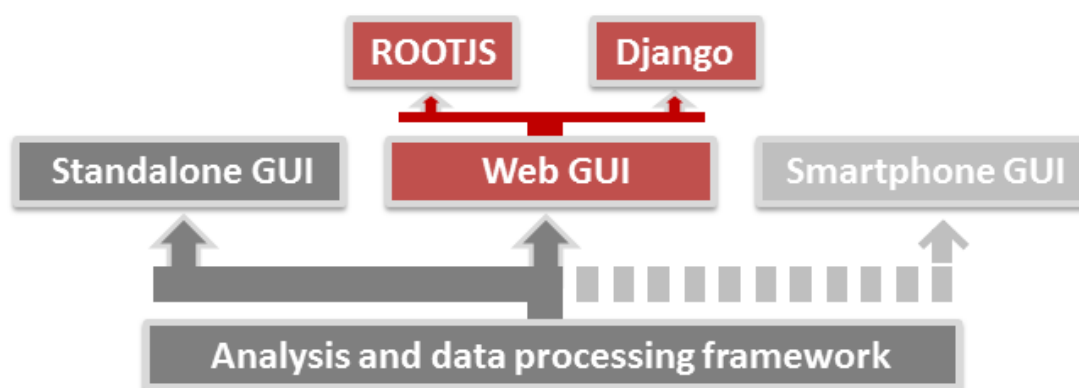


Figure 1.3: DQM applications and frameworks.

In the itemized list below, we address our solutions to the challenges and tasks which were presented in the previous section of this chapter.

- Create a Web GUI application using Django Web Framework
- Develop new functionality for the ROOTJS program that handles ROOT files for the Web
- Merge new ROOTJS program into Web GUI application
- Install and configure a server to test the VELO Web GUI application

Additionally to our Web GUI application, a standalone GUI application is also being developed for the Linux OS, as it was shown in the figure above. Moreover, there are plans to create another GUI application for smartphones and tablets later. The core framework of the project is the analysis and data processing framework which will be used by both standalone and Web GUI applications. However, all these parts are not in the scope of our thesis work and they are not included in the thesis.

At the end of January 2013, the LCH was stopped for 2 years for maintenance and upgrade along with all the experiments. That was the perfect time to begin the development of the VELO Web GUI application.

1.4 Thesis organization

In Chapter 2, the system and the technology background is presented. The reader will be introduced to the Web GUI application fundamentals. We also reveal the key part of the LHCb experiment to which our application is connected.

In Chapter 3, we give an overview on the frameworks and the developing tools that are used for the VELO Web GUI application design.

In Chapter 4, we cover our work in the thesis project. The requirements and the design of the VELO Web GUI application are the main topics of this chapter.

In Chapter 5, Solutions and implementations are presented. The chapter is divided into 4 sections to give a better overview and provide more details on the development progress.

Chapter 6 covers various tests that were performed on the VELO Web GUI application. The achieved results are introduced and commented.

CHAPTER 1. INTRODUCTION

In Chapter 7, we present the discussions part of the thesis project. We discuss the alternatives for the ROOTJS program and the Apache server.

Chapter 8 contains the conclusion of this thesis work. The chapter also covers contributions and future work.

Chapter 2

System and Technology Background

This chapter gives an overview of the thesis project's technological background. The reader will be introduced to the VELO Web GUI application fundamentals. Furthermore, the development tools are also presented.

2.1 System and configuration

Current section covers the system and configuration of the thesis project. We start this section with the presentation of the VELO detector which is a part of the LHCb experiment.

2.1.1 LHCb VELO

The LHCb VELO is the silicon micro strip Vertex Detector, responsible for identifying and triggering on vertices from the interaction region. It consists of 88 silicon sensors and 200k analogue readout channels. The control system must take track the voltages and currents of about 1000 high and low voltage supplies and their evolution, together with a host of other parameters, including readout streams with various levels of filtered information, the monitoring systems of the LHC beams, the VELO position, and the relationship between the monitored values. In Fig. 2.1 we can see 42 VELO modules.



Figure 2.1: 42 VELO modules [6].

The raw data that the LCHb VELO produces is processed by several core framework programs developed by CERN. It is saved in a database as data files with `.root` extension.

2.1.2 Setup overview

The final setup of the system that hosts the VELO Web GUI application after it is deployed is shown in Fig. 2.2.

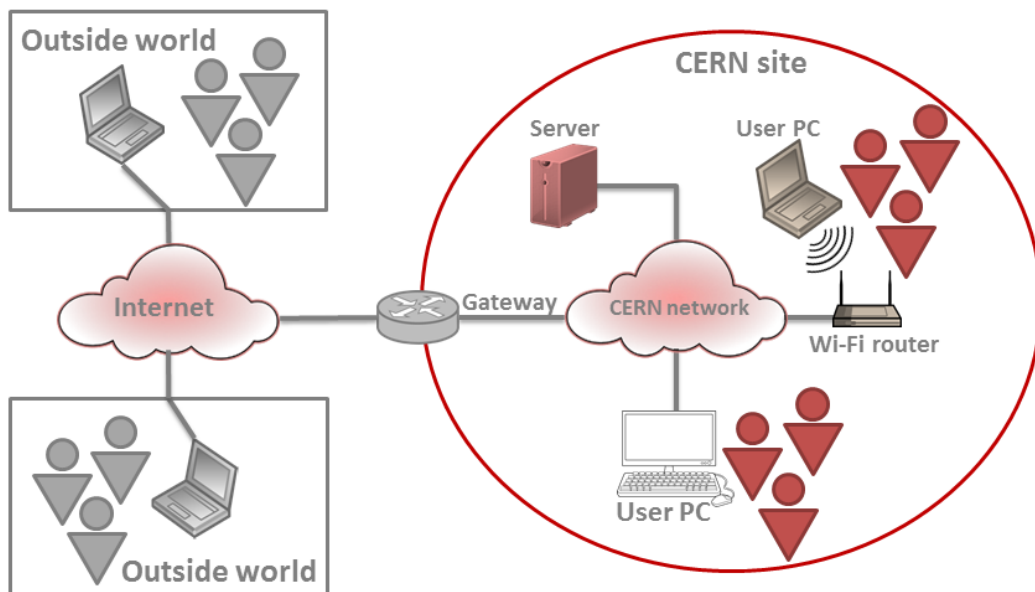


Figure 2.2: Final setup diagram.

The circle on the right side of the figure represents the CERN network. Inside it we can see how users are connected to the Apache server that hosts the VELO Web GUI application. On the top-left and bottom-left parts of the figure the users from various parts of the world are accessing the server.

2.2 The VELO Web GUI application fundamentals

In this section, the prototype of the Web GUI and the prototyping program are presented. We also present the ROOTJS program which contributes to our basics of work.

2.2.1 ROOTJS

The ROOTJS program was developed by one of the ROOT software programmers, Bertrand Bellenot at CERN. The main purpose of the program was to show how the online monitoring of detectors can be done through an Internet browser such as Mozilla Firefox, Google Chrome and Internet Explorer. ROOTJS is limited and does not provide all the functionalities of the ROOT program. Users can zoom-in and zoom-out the graphs while the y and x axis are scaled accordingly, as shown in Fig. 2.3.

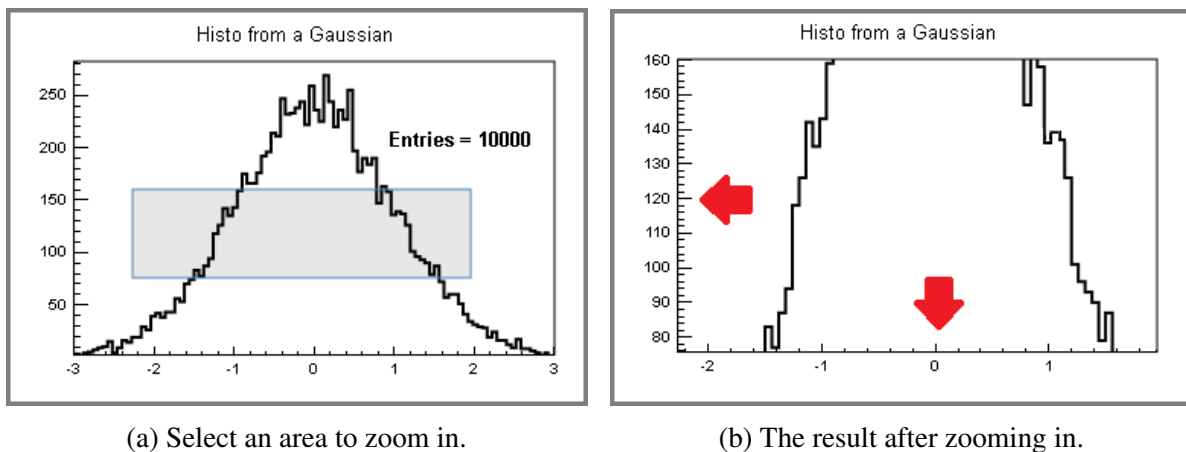


Figure 2.3: Zoom functionality in ROOTJS graphs.

In Fig. 2.4, the ROOTJS program is shown. Left part of the image is the navigation panel.

CHAPTER 2. SYSTEM AND TECHNOLOGY BACKGROUND

It contains 4 main interaction parts:

- Text field
- Drop-down list
- Load button
- Reset button

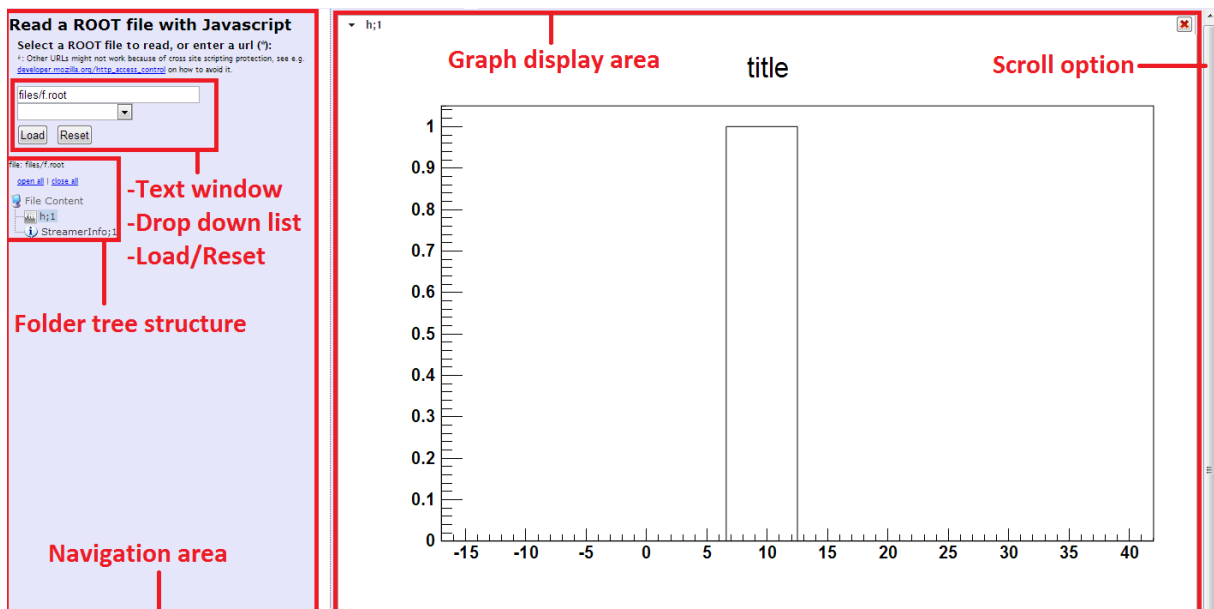


Figure 2.4: The GUI of the ROOTJS program.

In the drop-down list, users can select available ROOT files. The text field on top of this list indicates the selected file. After the desired file is selected it can be loaded using the Load button. If the wrong file is selected one can press the Reset button to start over. When a ROOT file is loaded, its content appears below the Load and Reset buttons in a folder tree structure. Users can double click on any of the content files and they will be displayed.

The right part of Fig. 2.4 is reserved to display the content of ROOT files. Users can display multiple graphs from a single ROOT file. The program adds them in separate collapsible blocks under already existing graphs. To find all previously displayed graphs users can utilize the scroll option and scroll up.

ROOTJS program consists mostly of JavaScript but also HTML and CSS files. A diagram that represents the core JavaScript files of the program is presented in Fig. 2.5.

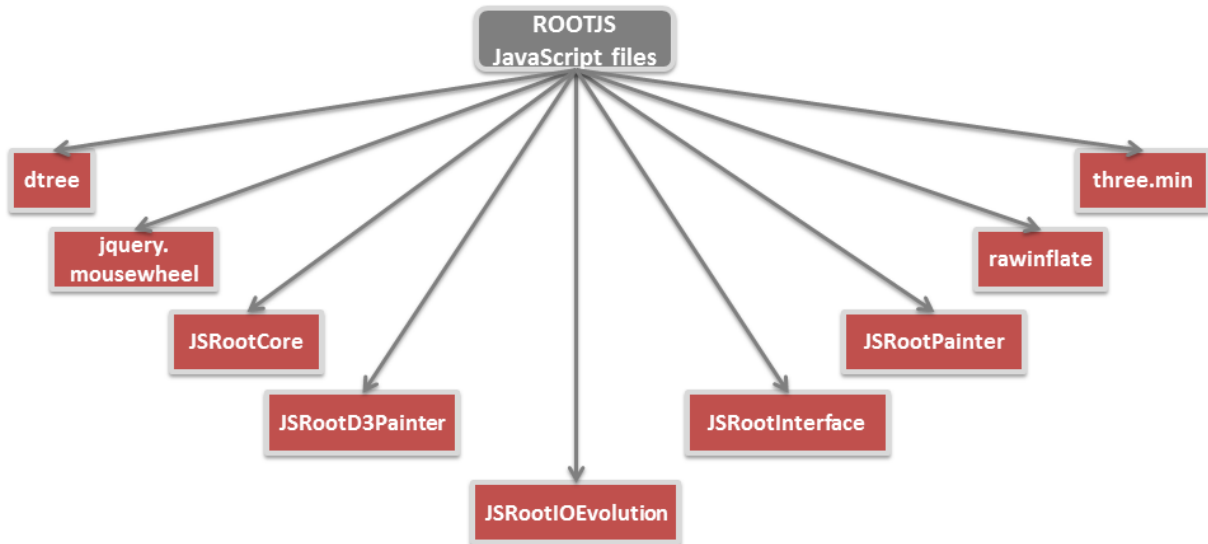


Figure 2.5: Core files of the ROOTJS program.

Their tasks are to read, process and serve the content of ROOT files to the display. In this thesis project, we focus mainly on the *JSRootInterface.js* file which connects all other core files together.

2.2.2 GUI prototype

In the starting phase of the project, a prototype of the GUI was developed. This prototype was the same for both standalone GUI and the Web GUI. It was created using an Internet based service called Pidoco that provides the prototyping framework [10]. In Fig 2.6, we demonstrate the Pidoco prototype of our GUI. It has five main horizontal navigation tabs:

- VELO view
- Run view
- Special analysis
- Sensor view
- TELL1 view

Some of these main tabs contain sub-tabs with additional properties. We present them and the entire Web GUI later in this thesis. Moreover, this prototype allows us to leave comments

CHAPTER 2. SYSTEM AND TECHNOLOGY BACKGROUND

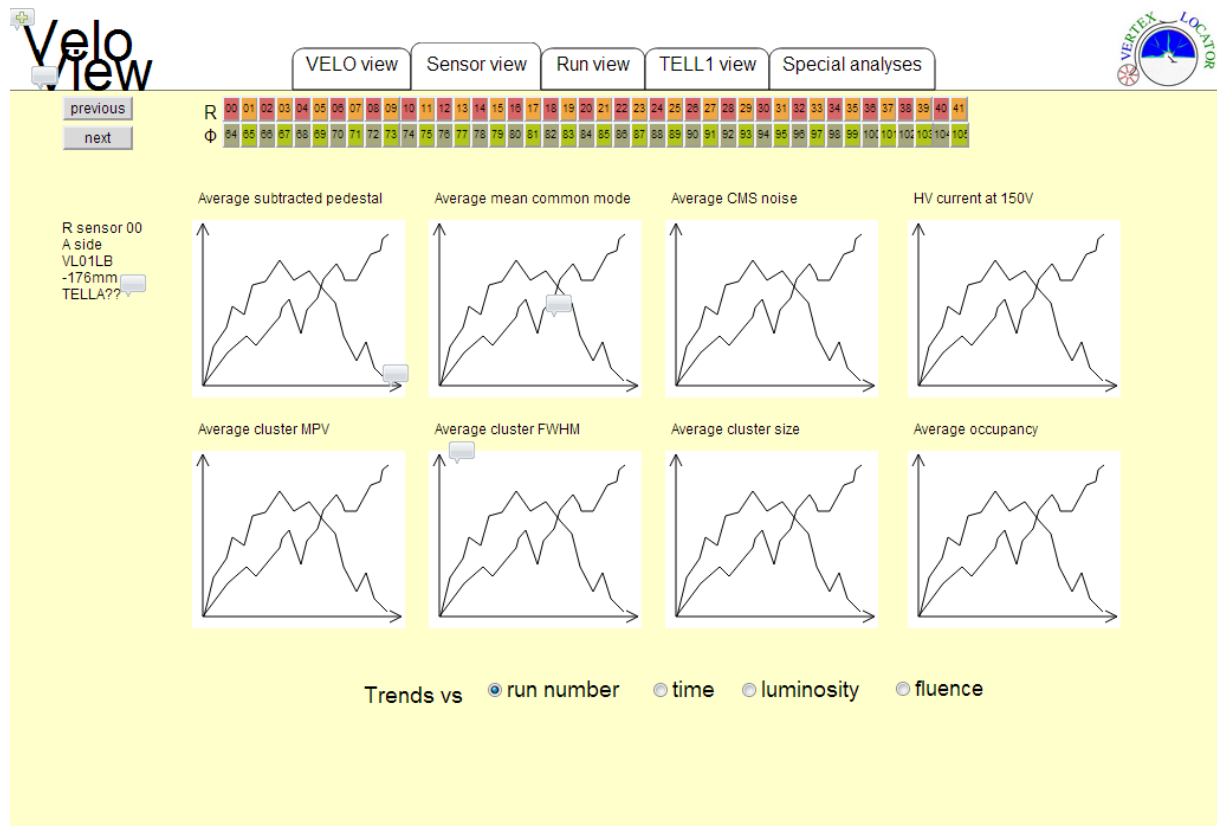


Figure 2.6: Prototype GUI [10].

for the future improvements and fixes. It provides the possibility for creating discussion threads, for example about certain functionality of the future program. Each of the project participants can log into this Pidoco prototype through the Internet and initiate a discussion by creating a comment.

Chapter 3

Framework for the VELO Web GUI Application Design

This chapter focuses on the frameworks and the development tools that were used for the VELO Web GUI application design. They had different purposes in the development process of the application as presented below.

- ROOT Framework was used to create ROOT files that contained various graphs (e.g. histograms). These were later used by our application.
- Django is built on Python. It was the key tool in the project. We used Django to build our application.
- JavaScript builds up the ROOTJS program that we used as the platform to develop new functionalities.
- The Apache HTTP Server was used to test the application and verify its functionalities.
- mod_wsgi module was used together with the Apache server for our application.
- We used X-Win 32 to establish a remote connection to our server where the VELO Web GUI application was located. In this way, we applied fixes and changes to our application.
- PyCharm is an IDE that we used to develop our Django application.

In this chapter, the previous version of the DQM application is presented. Additionally, we describe some of its important features and write about why the Web version and the upgrade was needed.

3.1 ROOT

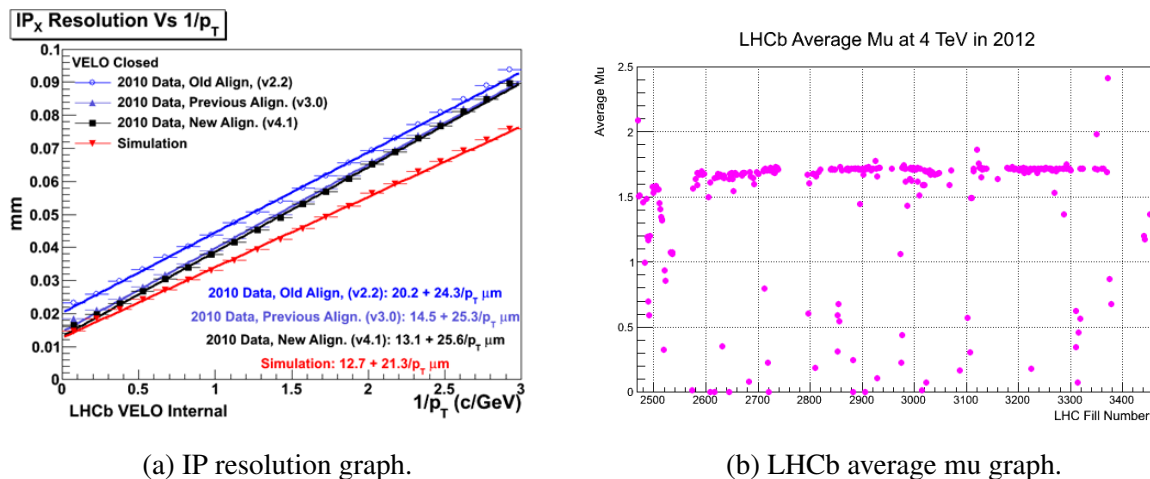
Scientists and engineers work with the large amount of data. This requires a system that is capable of analysing the data in an efficient way. ROOT program is a set of OO frameworks which provides that functionality. It is important to mention that ROOT is a software program and it has nothing to do with root privileges, rights or root access on computers. The data is defined as a set of objects and in this way the specialized storage methods are used to get direct access to the separate attributes of the selected objects, avoiding to touch unnecessarily the bulk of data. The following methods are included in arbitrary number of:

- dimensions
- function evaluation
- minimization
- graphics
- visualization classes

The analysis system can be easily set up to query and process the data interactively or in a batch mode. The possibility of using a general parallel processing framework called the Parallel ROOT Facility (PROOF), to speed up analysis is also included.

ROOT contains the build-in CINT¹ C++ interpreter and the command language, the scripting, or macro and the programming language are all C++. Moreover, ROOT is an open system and that makes it a premier platform to make simulation, data analysis and data acquisition. It also can be extended by linking the external libraries [7]. In the Fig. 3.1 a) and b) we can see the examples of ROOT graphs.

¹CINT is the C and C++ interpreter



(a) IP resolution graph.

(b) LHCb average mu graph.

Figure 3.1: Examples of ROOT graphs.

There are two other versions of ROOT such as PyROOT and ROOTpy. PyROOT is simply the Python extension of ROOT. It is a module that allows the user to interact with any ROOT class using Python interpreter. Python wrapped code is not required to include new ROOT classes, because the process is done generically using the ROOT dictionary [8]. ROOTpy can be called more pythonic PyROOT. It gives a better, feature-rich, "pythonic" interface with the ROOT libraries on top of the existing PyROOT bindings [9].

3.2 Tools for VELO Web GUI design

In this part, we write about the tools that we used in this project. We begin this section with the presentation of the Python programming language.

3.2.1 Python

Python is a popular high-level dynamic programming language. It is powerful and fast. Python standard libraries cover everything from asynchronous processing to zip files. It has many features that make it more attractive than programming languages such as Java, Ruby, Perl and others. These features are:

CHAPTER 3. FRAMEWORK FOR THE VELO WEB GUI APPLICATION DESIGN

- intuitive object orientation
- strong introspection capabilities
- readable and very clean syntax
- full modularity, supporting hierarchical packages
- extensive standard libraries and third party modules for every possible task
- very high level dynamic data types
- extensions and modules easily written in C, C++(or Java for Jython, or .NET languages for IronPython)
- embeddable within applications as as scripting interface

Python is also used in variety of application domains such as Web and Internet development, Desktop GUIs, Education and more. Moreover, it cooperates well with other objects such as COM, .NET, and COBRA. Python runs on most operating systems like Windows, Linux/Unix, OS/2, Amiga and Mac. In Fig. 3.2, we demonstrate an example of the default Python interpreter.

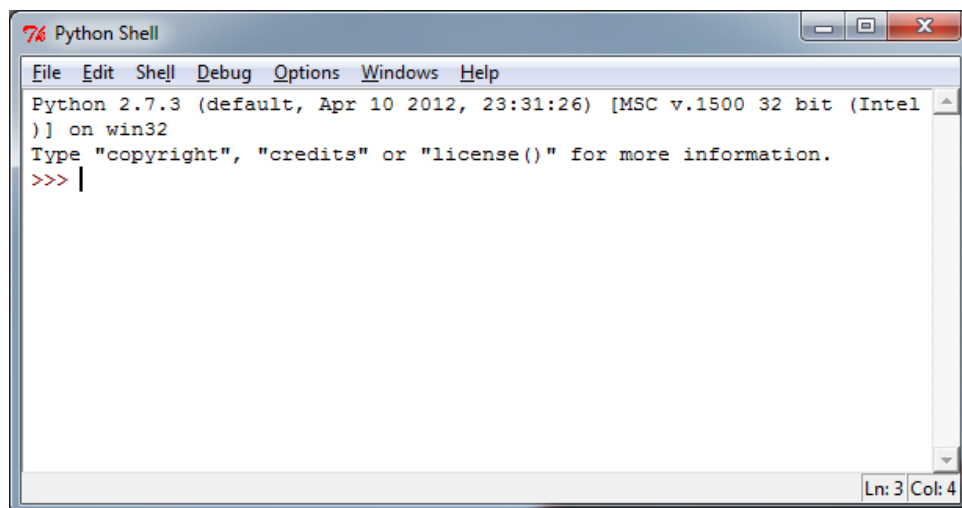


Figure 3.2: Default Python interpreter.

Additionally, it is easy to learn Python because of the well written documentation, availability of the books, wiki page, conferences to mention few. Finally, Python is under open source license that means that it is freely distributable and usable even for commercial use [11].

3.2.2 Django Web framework

Django is a Web Development Framework based on Python. It was originally developed to avoid writing new Web applications entirely from scratch. Django was born back in 2003 at Lawrence Kansas, USA in Lawrence Journal-World newspaper environment where quick deadlines are common. The World Online team was in charge of several local news web-sites maintenance and production. And in 2005 when Django was efficient enough it became an open source software. Nowadays tens of thousands users are using this well-established open source project.

There are two key features of Django. First, it is an open source program. It is acutely focused on dealing with day to day problems in the Web environments that Django's developers are facing or have faced. Second, it contains many useful features such as an admin page, as shown Fig. 3.3.

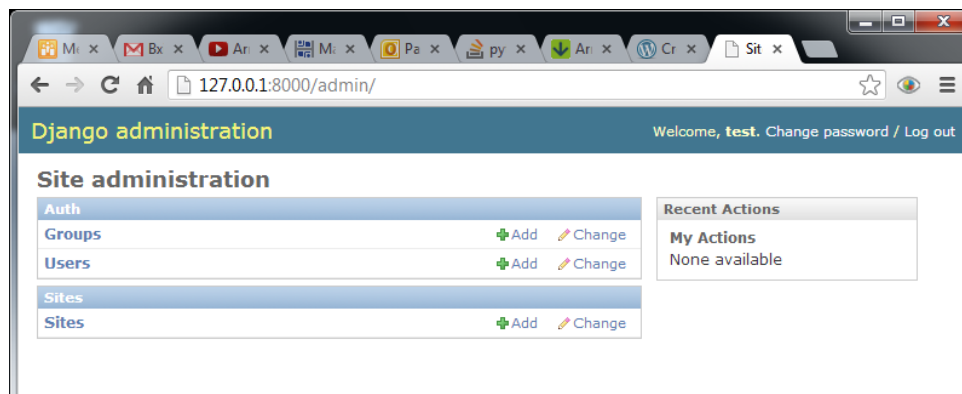


Figure 3.3: Typical default Django admin page.

These are particularly suitable for sites as Amazon.com, craigslist.org, and washington-post.com that offer dynamic, database-driven information. Even though Django is better for this type of sites, it is also an effective tool in building all sort of dynamic Web sites.

In conclusion, there are plenty of educational contents available out there such as both free and paid books, tutorials, forums and more.[12].

3.2.3 JavaScript

JavaScript (JS) is a scripting language that was originally developed by Netscape Communication Corporation for the Web use in 1995. JS should not be confused with Oracles' Java programming language. These languages are totally unrelated and have different schematics. JS has syntax which is influenced by C programming language and it also copies many names and naming conventions from Java. JavaScript is a lightweight object-oriented, prototype-based, multi-paradigm scripting language that is type safe and dynamic. JS uses an interpreter and supports also object-oriented, imperative and functional programming styles. JavaScript has a standard called ECMAScript. The modern Internet browsers are updated and fully support the ECMAScript v5.1, as of 2012 [13].

3.2.4 Apache HTTP (Web) server

The Apache HTTP (Web) Server is an open source code implementation of an HTTP (Web) server that is freely available. Additionally, this server is robust, includes many features and is on commercial-grade scale. To develop such a server a group of volunteers all over the globe has joined their forces. Through the Internet communication they have developed and planned the server and its documentation. The whole project is a part of Apache Software Foundation. Moreover, the huge international user community has contributed ideas, thoughts, code and documentation to this project [14].

3.2.5 mod_wsgi

mod_wsgi is a simple Apache module that is used to host applications written in Python which supports Python Web Server Gateway Interface (WSGI). mod_wsgi module is well suitable for your own personal Web hosting services and mainly for high performance production web sites [15].

3.2.6 X-Win32

X-Win 32 is a software program that provides necessary tools to establish a connection from a Microsoft Windows based machine to a UNIX based machine such as Linux. During this connection a Windows OS can remotely display UNIX windows or the Terminal side by side with its other applications. The program is developed by StarNet Communications and based on X Windows System for MS Windows. Last version of X-Win 32 program is 2012 [16]. It offers different types of standard connection protocols such as telnet and ssh. Additionally, users can benefit from proprietary LIVE Connection Protocol that extends the standard connection protocols with more features. The Configuration panel and Secure Shell (SSH) Terminal of X-Win 32 2012 program is presented in Fig. 3.4.

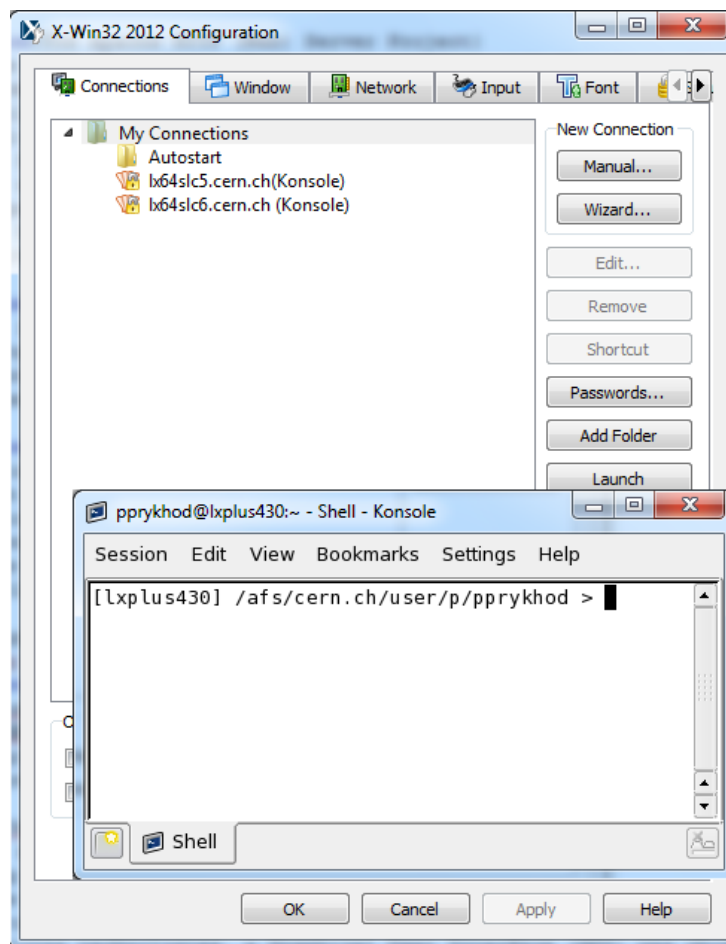


Figure 3.4: X-Win 2012 configuration panel.

3.2.7 PyCharm

PyCharm is an Integrated Development Environment (IDE) that was developed by JetBrains software company. It is powerful and tools reach program which makes Python programming quick and efficient. PyCharm is also a very powerful tool for developing Django applications [17]. It provides many features such as:

- Project Navigation
- Coding Assistance
- Coding Analysis
- Graphical Debugger
- Python Refactoring
- Version Control Integration

The default IDE of PyCharm is shown in Fig. 3.5. On the left part of the figure we can see a navigation tree and file content on the right part.

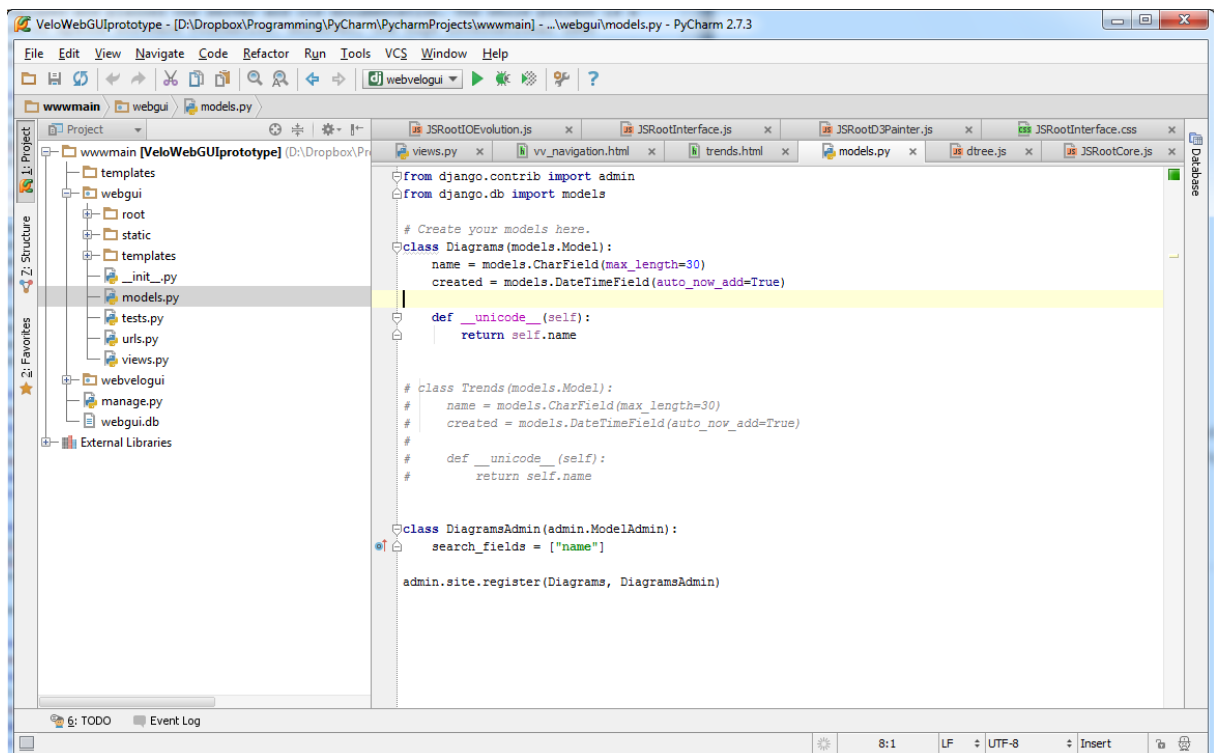


Figure 3.5: Default PyCharm IDE.

3.3 VeloMoniGUI

VeloMoniGUI was the previous DQM application of the LCHb VELO detector. Since the start of the first LHC synchronization tests in August 2008, the VELO monitoring became an important part in the development process. Such an application could make it easier to centralize all developed tools for better maintenance. In order to establish a common and user friendly communication between users and the underlying technologies, a GUI was chosen. *VeloMoniGUI* was first proposed in February 2009 and later accepted as the framework for the VELO offline monitoring [18]. The main GUI of the application is shown in Fig. 3.6.

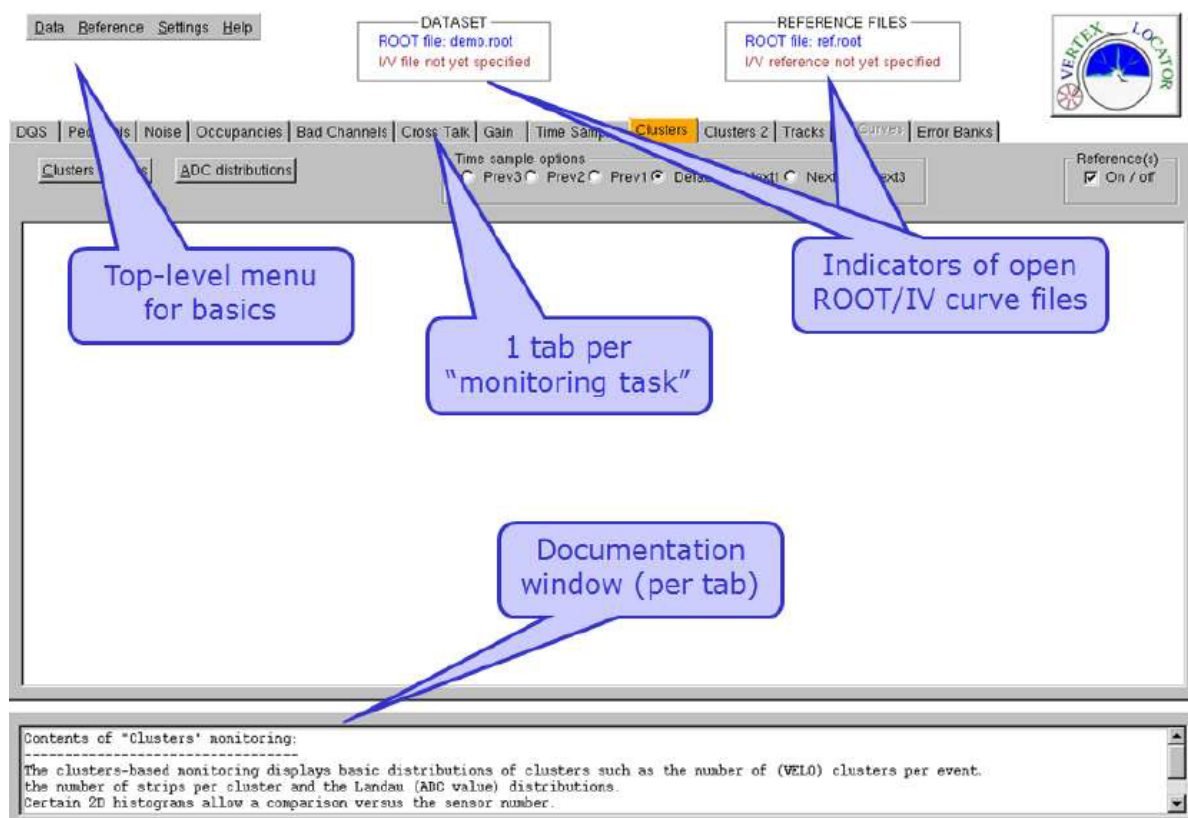


Figure 3.6: Overview of the VeloMoniGUI [18].

VeloMoniGUI consists of Top-level menu for basics, 1 tab per "monitoring task", Documentation window (per tab) and Indicators of open ROOT/IV curve files. There are different monitoring tasks that represent all relevant aspects of the detector and they are plotted into histograms for further analyses. The monitoring task tabs are:

CHAPTER 3. FRAMEWORK FOR THE VELO WEB GUI APPLICATION DESIGN

- Overview
- Cross Talk
- Tracks
- IV Curves
- DQS
- Pedestals
- Error Banks
- IV Trending
- Trends
- Clusters
- Bad Channels
- Detailed Trends
- Clusters 2
- Time Samples
- Noise
- Occupancies
- Gain

There were good reasons why the program had to be upgraded. First, the program was badly planned. Second, VeloMoniGUI was written partly in Python and partly in C++ programming languages. These two reasons already were causing difficulties for maintenance and debugging. Furthermore, the program had some issues in the GUI interface and also lack of additional features such as:

- Complicated procedure to find ROOT files and load them (too many mouse clicks)
- No automatic Data Quality Summary (DQS) status (only manual)
- No Web based application alternatives

Finally, the access to this application from outside the CERN network was complicated and time consuming. Taking into account all these reasons for the upgrade we began to plan and develop the Web GUI application.

Chapter 4

Requirements and Design

This chapter covers our work in this thesis project. The chapter is divided into two sections - the requirements and the design of the VELO Web GUI application.

4.1 Requirements

The VELO Web GUI application was the main result of the thesis project. It was developed with certain requirements in mind. The requirements were divided into primary requirements and auxiliary requirements. To fulfil the primary requirements the VELO Web GUI application should:

- read and process ROOT files
- display 2D histograms
- have basic ROOT functionality such as zooming in and out of graphs
- have a simple to use GUI
- contain familiar parts from the previous version of the application (e.g tabs)
- be user friendly

CHAPTER 4. REQUIREMENTS AND DESIGN

The auxiliary requirements were optional. To fulfil them the application should:

- have an option to collapse individual histograms
- have an option to delete individual histograms

These options were already present in the original ROOTJS program. Our task was to make them work in the modified version of the ROOTJS program. All these requirements were the guidelines for the development of the final application.

4.2 Design

In this part of the chapter, we present the design of the VELO Web GUI application and its features. The main layout of the application is shown in Fig.4.1.



Figure 4.1: The VELO Web GUI application interface.

The corresponding structure diagram to the layout in Fig. 4.1 can be found in Subsection 5.2.2. The Design section is split into 2 subsections, where the primary and the auxiliary features of the application are presented.

4.2.1 Primary features

In this subsection, we present the primary features of the VELO Web GUI application. The main GUI of the application is divided into 3 areas, as shown in Fig. 4.2.

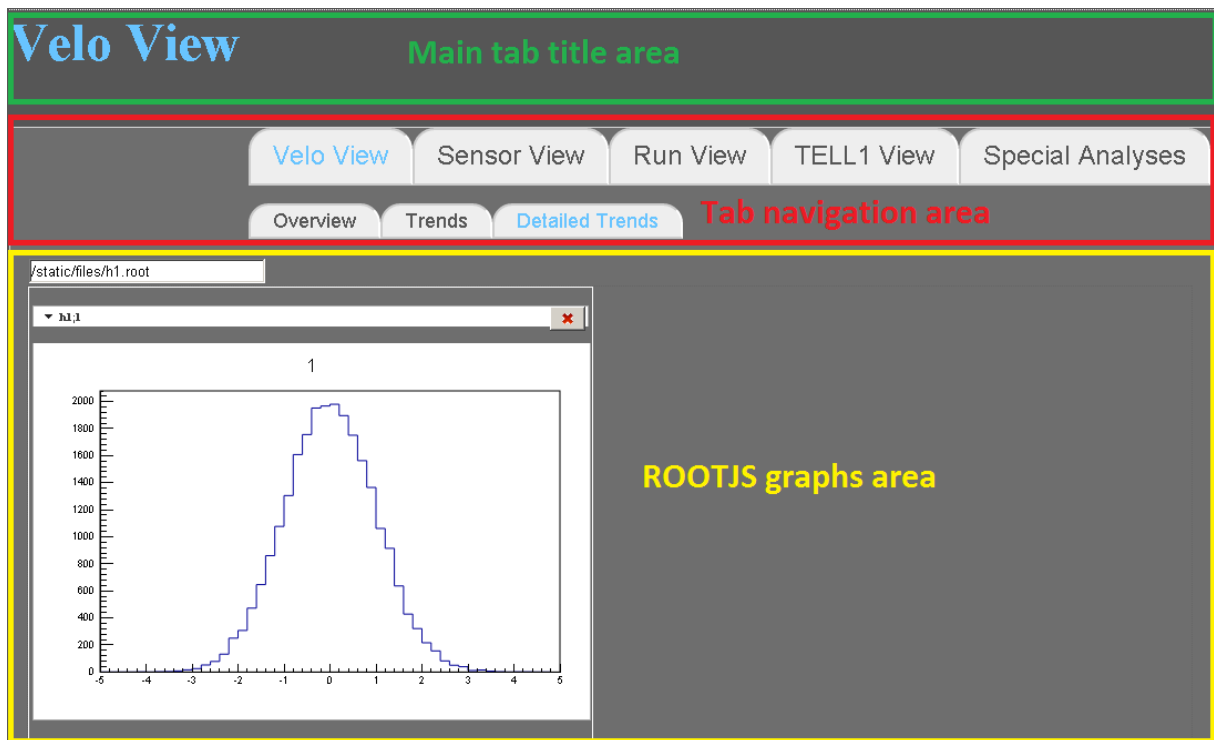


Figure 4.2: The VELO Web GUI application areas.

These areas are also summarized in a list below. Further in this subsection, we introduce all the areas in greater detail.

- Main tab title area
- Tab navigation area
- ROOTJS graphs area

Main tab title area

The main tab title area design was developed in the following way. When users selected one of the main tabs, its menu title name was displayed in the top-left corner of the GUI, marked with the red line in Fig. 4.3. The menu title had bigger font size than the tab title. This was useful in some situations for example, when people who wanted to follow the demonstration seated far away from the screen. Moreover, the selected tab was highlighted with the blue color, as shown in Fig 4.3. In this way, users could keep even better track of their current location.

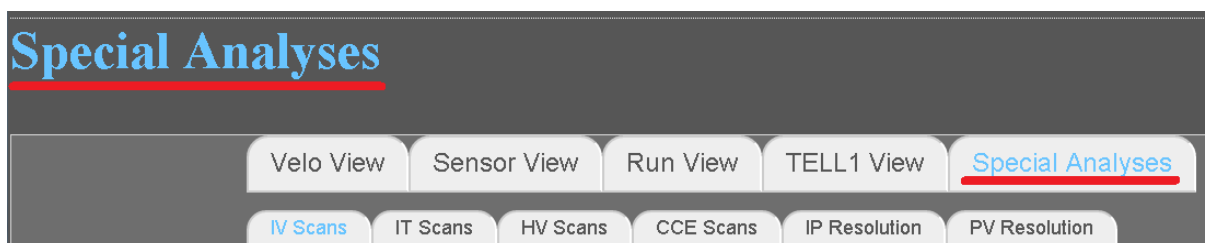


Figure 4.3: Tabs highlighting.

Tab navigation area

The tab navigation area of the application contained 5 main horizontal tabs, marked red in Fig. 4.4. The main tabs were also divided into sub-tabs, marked green in the same figure. We also referred to the main tabs and the sub-tabs as the first and the second level of navigation respectively.



Figure 4.4: Navigation bars.

In our application, we decreased the amount of main tabs from 13 to 5 comparing to the previous application, the VeloMoniGui that was presented in Section 3.3. Rest of the tabs were moved to the sub-tabs area. In this way, the application looked more organized and still contained the familiar parts from the earlier version, as specified in the requirements.

ROOTJS graphs area

In this area, the application read ROOT files and displayed their content using the integrated ROOTJS program with the new functionality. In the original ROOTJS program, users had to make up to three mouse clicks to display a single graph. First, they had to select a ROOT file from the drop-down list, then press the Load button, wait for the folder tree structure to appear, and finally click on one of the available graphs. In the VELO Web GUI application, this process executed completely automatically. When users opened a tab, ROOT files started to load and the tab layout was displayed along with graphs. Each of the tabs had different set of graphs, as specified in the prototype of the application. In our application, we used histograms combined in matrix of 2x2, 2x4 and there were also just single histograms in a bigger size. For example, in the Sensor View tab users could find the 2x4 matrix of histograms, as shown in Fig. 4.5.

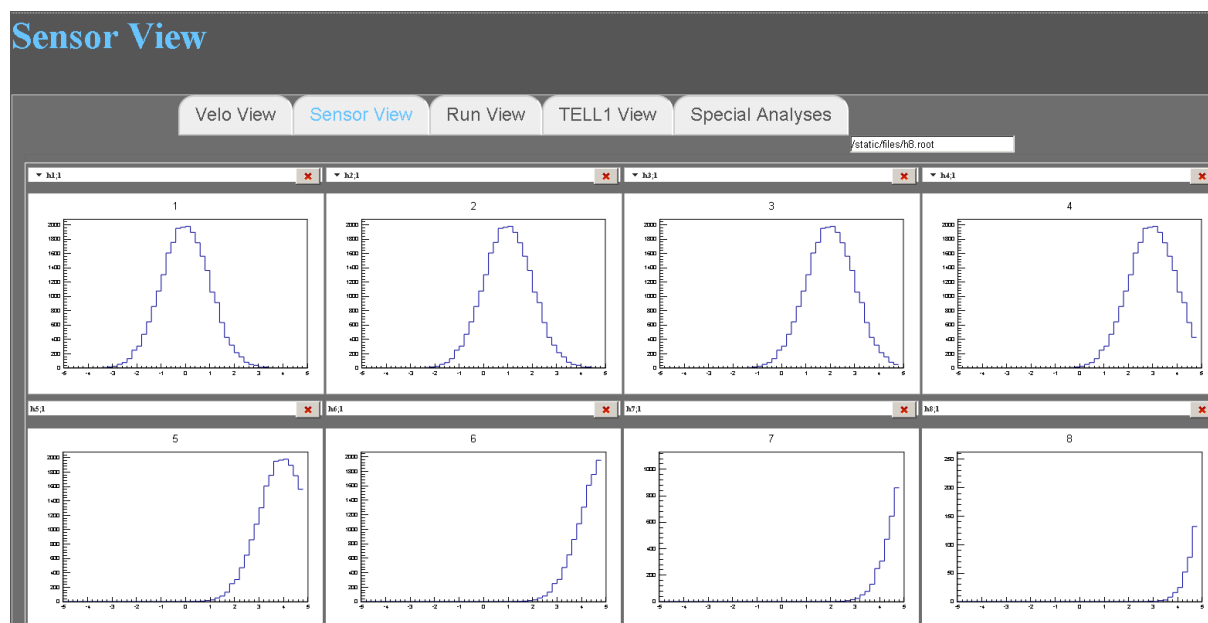


Figure 4.5: Sensor View tab.

Furthermore, the ROOTJS graph area also contained a small text window that displayed a path to and a name of the ROOT file that is being read, as shown in Fig. 4.6. This window helped to ensure that the right file was selected.

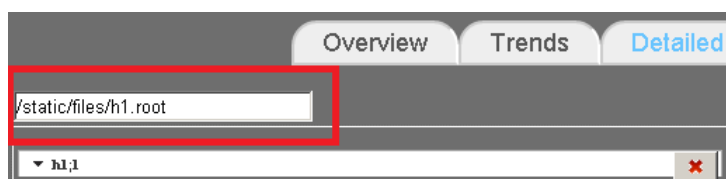


Figure 4.6: ROOT file path window.

4.2.2 Auxiliary features

The VELO Web GUI application contained the auxiliary features that were specified in our second requirements list. The first feature enabled users to collapse undesired graphs. This feature could be useful in various situations for example, when users are presenting a matrix of graphs in front of an audience, or when they are discussing some specific graphs together in a group. They can collapse the unwanted graphs and thus get better overview on the remaining ones. When users want to get some of the graphs back later in the session, they just have to press the collapsible button of the selected graph again and it will reappear. The collapsible button is shown in Fig. 4.7.

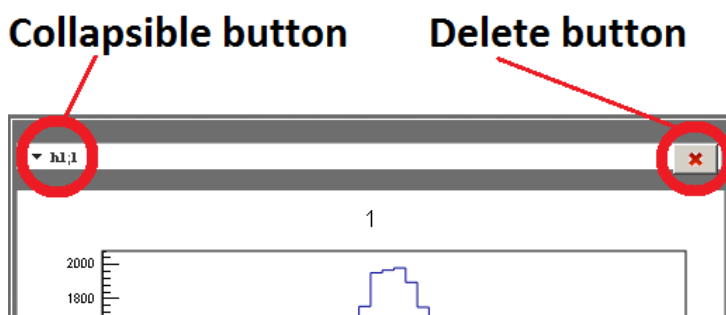


Figure 4.7: Collapsible and delete buttons.

The second auxiliary feature of the VELO Web GUI application was the delete button. It was located in the top-right corner of each graph. When users utilize this feature they can remove undesired graphs permanently from the tab layout. However, in order to get the deleted graphs back, users have to reload or refresh the entire page. This feature works differently in comparison to the collapsible feature. Nevertheless, it is useful in situations when for example, there are not important graphs in a given session, or they have some corrupted and wrong data etc. The users have the possibility to delete these graphs and avoid them in the presentation. The delete button is also shown in Fig. 4.7.

Chapter 5

Solutions and Implementations

In this chapter of the thesis, we present our solutions and implementations. The implementation of the VELO Web GUI application was divided into 4 main individual steps, as shown in Fig. 5.1.

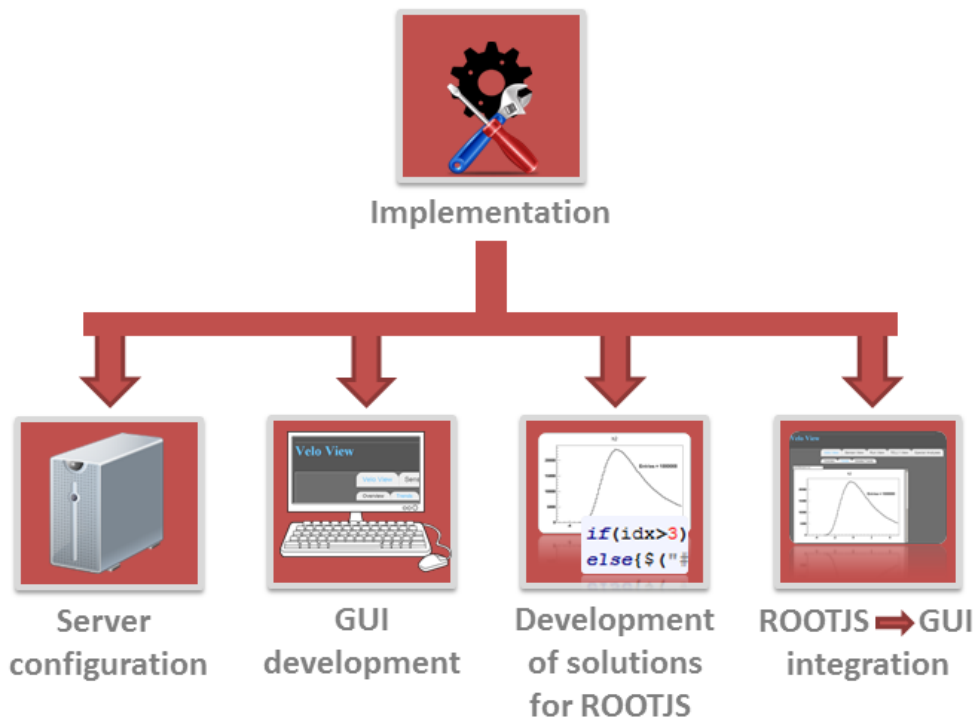


Figure 5.1: Implementation of the VELO Web GUI application.

- Server configuration
- GUI development
- Development of solutions based on the ROOTJS program
- ROOTJS → GUI integration

Server configuration is the first step of the implementation that is cover in this chapter. Then, we advance to the GUI development section. The third section presents the developed solutions based on the ROOTJS program. Finally, the end of this chapter reveals the integration process of the modified ROOTJS program into our GUI.

5.1 Server configuration

In this section, we write about how we prepared and configured the server for the development and testing of our application. The server location is shown in Fig. 5.2.

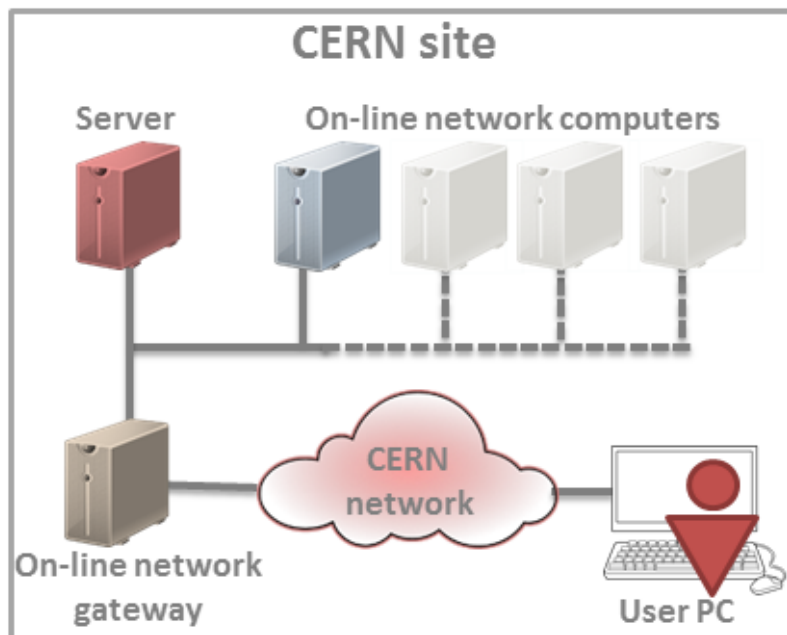


Figure 5.2: Server Location.

We also present Virtual Machine (VM) that we used to host our Apache server. The VM was a Linux PC located in the Online network which was separated from the outside world. In order to access this network users first had to connect to the gateway PC and then to the desired Online network computer, as shown in Fig 5.2. The location of the server was chosen due to the security precautions.

5.1.1 Preparing the VM

In our project, we used a stationary PC with Windows 7 operative system to connect to the VM using X-Win32 2012 program. This program was developed for the Windows OS to display remote Linux desktops and individual applications over a LAN connection, as presented in Subsection 3.2.6. The step by step flow chart of the VM preparation is shown in Fig. 5.3.

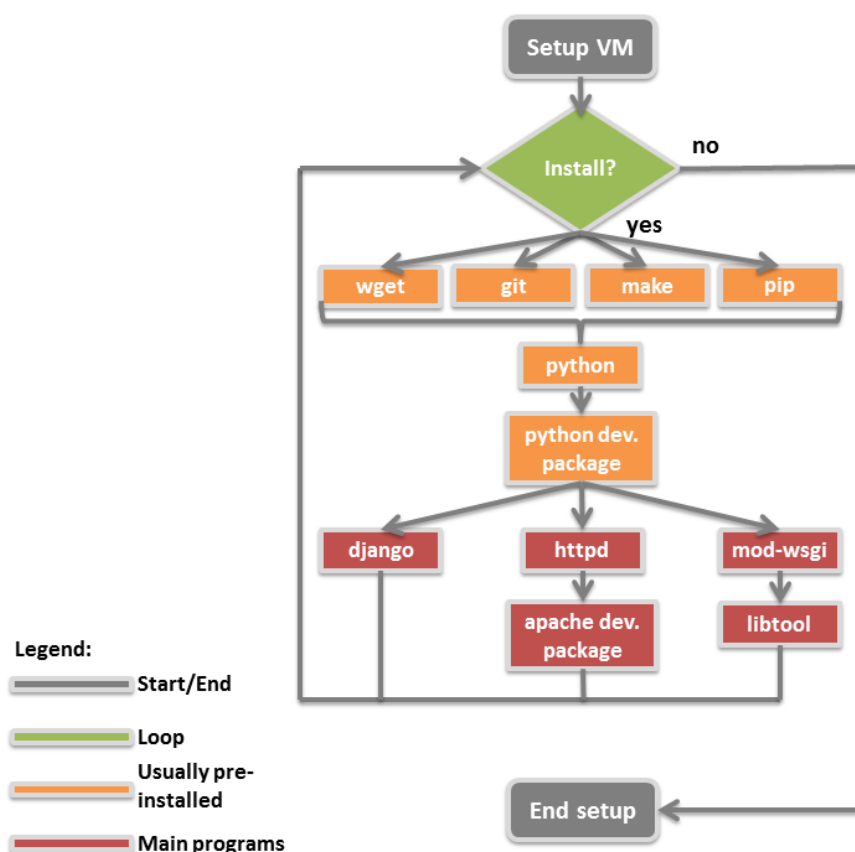


Figure 5.3: VM preparation flow chart.

The orange area of the flow chart represents the components which are usually pre-installed on most of Linux distributions such as `make`, `wget` and `Python`. However, we had to install most of them. The red area shows the main programs in our project that had to be installed such as Django Web Framework, `mod-wsgi` and `httpd` server. Throughout the installation cycle we faced several challenges with the user permissions on our Scientific Linux VM. If one plans to install programs on Security Enhanced Linux (SELinux) computers, we recommend to have good knowledge about the system. The used versions of the programs are shown in Table 5.1.

Table 5.1: Program versions.

Program	Version
Scientific Linux	6.0, later 6.3(CARBON)
<code>httpd</code> (Apache) server	2.2.15 (UNIX)
<code>mod_wsgi</code>	3.4
Django	1.5

5.1.2 Configuring the VM

When the installation of the required software was completed we went over to the initial configuration of Apache, Django and `mod-wsgi`, as shown in Fig. 5.4.

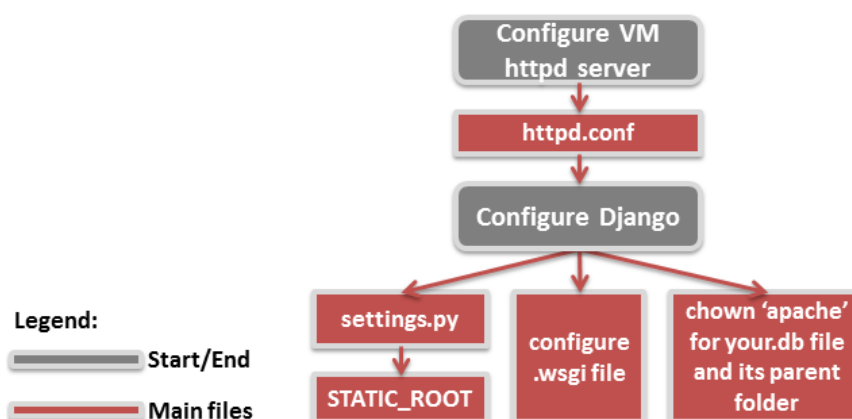


Figure 5.4: VM preparation flow chart.

We started with the server configuration. The Apache server had a configuration file called `httpd.conf`. This file had to be modified as shown and explained in Appendix A. Our next step

was to configure Django. First, we created and configured a WSGI file, as covered in Appendix B. Then, we modified the *settings.py* file and saved the changes, as described in Appendix C.

5.2 GUI development

In this section, we discuss the application functionality and comment parts of the code that lay behind it. We begin with a short overview of how the Web GUI was developed in Django.

5.2.1 Overview

The GUI of the application was developed using Django Web Framework, presented in Subsection 3.2.2. It consists of Python files that provide the GUI's core functionality and Django templates that build up the GUI's layout, as shown in Fig. 5.5.

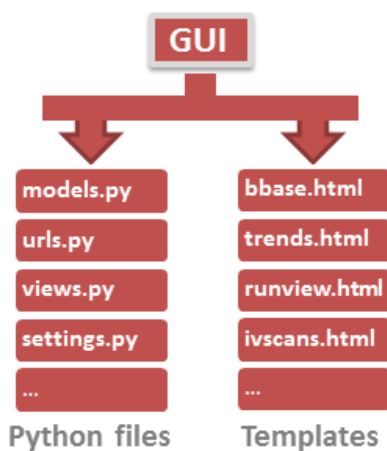


Figure 5.5: Django Web GUI files.

A Django template is a string of text that separates the presentation of a document from its data. In general, Django templates can be used for any sort of text-based data. The templates may also include parts of CSS and JavaScript codes, plus other features such as for-loops and if-statements (template tags) [19]. We used Django templates to create Web GUI layouts which were saved as HTML files.

5.2.2 Layouts

The VELO Web GUI application consists of many layouts, as shown in Fig. 5.6.

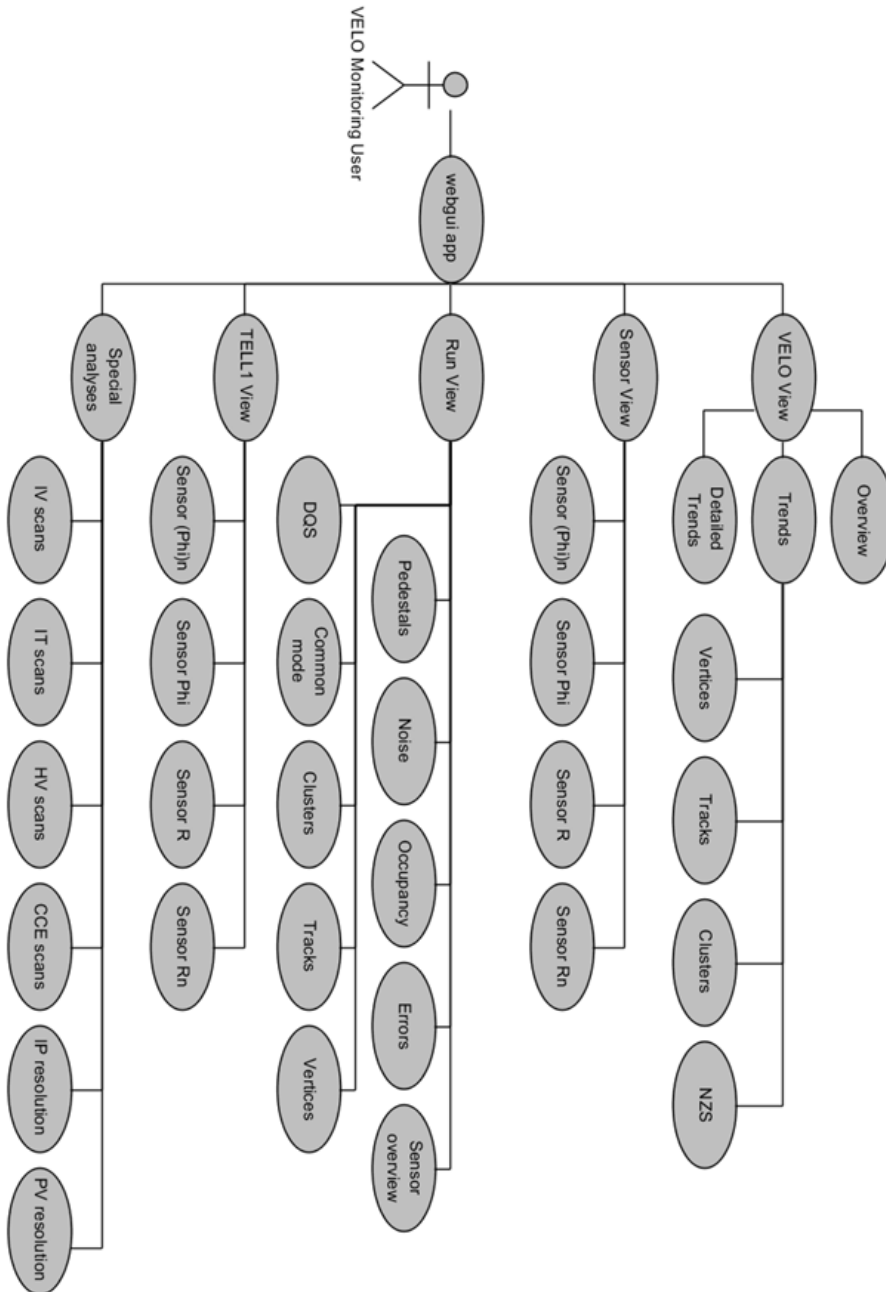


Figure 5.6: The VELO Web GUI application structure diagram.

If we go back to Fig. 4.1 in Section 4.2, and compare it with this structure diagram, we can identify the 5 main tabs and see what sub-tabs they contain. Fig. 5.6 represents the structure of the layouts, as seen from the application's user point of view. Layout is an important part of every application regardless if it is created for the Web or not. It is a pre-defined arrangement of the objects such as text headers, tabs, images and buttons on a given space (e.g. Web page, window).

Templates

The layouts of the VELO Web GUI application were built in a smart and efficient way. The core template file was called *bbase.html*, positioned in the middle of Fig. 5.7 and marked red. It inherits 5 main templates, positioned around the core file in the same figure and marked grey.



Figure 5.7: *bbase.html* inheritance diagram.

Using this technique we avoided many unnecessary code duplications because all the main templates, the grey blocks, utilized the same predefined main layout, the red block. In this structured approach, it was also more convenient to work on the code development and it was easier to debug it. Part of the *bbase.html* code is presented in Fig. 5.8

```
<h3>{&#x2013; block menutitle &#x2013;}Web VELO Monitoring{&#x2013; endblock &#x2013;}</h3>
<div id="block_content">
  {&#x2013; block veloview &#x2013;}{&#x2013; endblock &#x2013;}
  {&#x2013; block sensorview &#x2013;}{&#x2013; endblock &#x2013;}
  {&#x2013; block runview &#x2013;}{&#x2013; endblock &#x2013;}
  {&#x2013; block tellview &#x2013;}{&#x2013; endblock &#x2013;}
  {&#x2013; block specialanalyses &#x2013;}{&#x2013; endblock &#x2013;}
</div>
```

Figure 5.8: A piece of *bbase.html* code.

This implementation is very popular in many Django applications. The first line of the code defines the menu title of the active tab that would be changing when navigating between the main tabs. The menu title is located in the top left corner of our GUI, as it was shown in Fig. 4.5. In general, the *bbase.html* template code consists of the HTML structure such as head and body. The body part includes the blocks that inherit all the main tabs templates.

Veloview.html, *sensorview.html*, *runview.html*, *tellview.html*, *specialanalyses.html* templates are located separately. Although these templates are saved with the HTML file extension, they do not include the HTML head and body fields. In Fig. 5.9 we can observe how the *specialanalyses.html* template extends the core template and inherits sub-tabs templates. We refer to this inheritance as the second level inheritance.

```
{% extends "webgui/bbase.html" %}

{% block menutitle %}Special Analyses{% endblock %}
{% block specialanalyses %}
{% include "webgui/main_navigation.html" with active_tab='tab5'%}

    {% block ivscans %}{% endblock %}
    {% block itscans %}{% endblock %}
    {% block hvscans %}{% endblock %}
    {% block ccscans %}{% endblock %}
    {% block ipresolution %}{% endblock %}
    {% block pvresolution %}{% endblock %}

{% endblock %}
```

Figure 5.9: Sensor View template.

The first line of the code specifies the location of the *bbase.html* template and extends it. The second line we defines the name of this template that will be shown in the GUI. Third line of the code defines the head block that contains 6 additional blocks. These blocks inherit templates that represent the sub-tabs of the *specialanalyses.html*. One of the sub templates is demonstrated in Fig. 5.10. This is exactly the same method that was used with the *bbase.html* only on the level under. Finally, the fourth line of the code represents the tab navigation system of the application that is discussed in more details later in this chapter.

```
{% extends "webgui/specialanalyses.html" %}

{% block ivscans %}
{% include "webgui/sa_navigation.html" with active_sidetab='sidetab1' %}
  IV Scans are here!
{% endblock %}
```

Figure 5.10: Inheritance blocks of *specialanalyses.html* file.

Views

In Django, the views are the Python functions that make Web responses on Web requests. To be more precise, in our application the views serve the right templates when users request a specific layout by clicking on the tabs, as shown in Fig. 5.11. Please note that the word "View" in the tab names Velo View, Sensor View, Run View, TELL1 View, does not have any relation to Django views. The tabs could be also called Velo Tab, Sensor Tab etc.

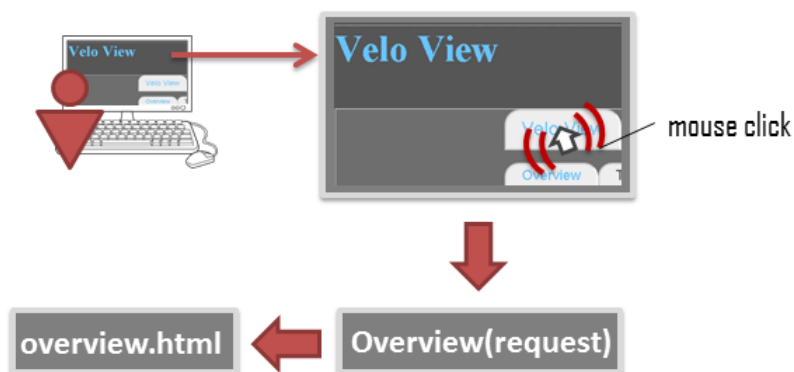


Figure 5.11: Django views diagram.

The figure shows the VELO Web GUI user who is clicking on the Velo View tab. His request is processed by the "Overview" view, and then the *overview.html* template is served. Mark that the *veloview.html* template does not exist, as mentioned in previous paragraph. That is why, the view passed the template of the first sub-tab.

The VELO Web GUI application contains 14 individual tabs in total but only 12 views. The reason there are 2 views less than the amount of tabs, is because the Velo View and Special Analyses tabs display one of their sub-tab views directly. For example, when the user clicks on Special Analyses tab he enters the IV Scans sub-tab. It is unnecessary to have empty Velo View and Special Analyses tabs. Part of the code that lays behind some of the application's views is

shown in Fig. 5.12

```
def Overview(request):  
    return render_to_response("../templates/webgui/overview.html")  
def Trends(request):  
    return render_to_response("../templates/webgui/trends.html")  
def DetailedTrends(request):  
    return render_to_response("../templates/webgui/detailedtrends.html")  
def RunView(request):  
    return render_to_response("../templates/webgui/runview.html")
```

Figure 5.12: Views.

The two red markings under `Overview(request)` and `overview.html` in the code above correspond to the diagram presented in Fig. 5.11.

Tab system

The tabs in the application are among the central functionality aspects. The overall purpose of the tabs is to have separate layouts with different content. Since the Django Web Framework does not include any native tab implementation scheme, we had to research and find our own solution. Moreover, the additional challenge was to find a solution which could also support tab-in-tab mode where there could be 2 levels of tabs. For example, when one of the main tabs is opened, the additional sub-tabs are appeared for the selection.

We found a code that could be used to create tabs in the Django environment [24]. The piece of the code is shown in Fig. 5.13.

```
<li class="{% if active_sidetab != 'sidetab1' %}" active{% endif %}"><a href="{% url 'webgui:Overview' %}">Overview</a></li>  
<li class="{% if active_sidetab != 'sidetab2' %}" active{% endif %}"><a href="{% url 'webgui:Trends' %}">Trends</a></li>  
<li class="{% if active_sidetab != 'sidetab3' %}" active{% endif %}"><a href="{% url 'webgui:DetailedTrends' %}">Detailed Trends</a></li>
```

Figure 5.13: Part of the tabs code.

Unfortunately, the original version of the code did not fulfil all of the requirements for the tab functionality, and we developed our own solutions based on it. The code was implemented in such a way that there existed 2 levels of tabs - main tabs and sub-tabs. The tab highlight feature was also added. Using it users can identify the current active tab. We successfully integrated the modified code into the application and the final result is shown in Fig. 5.14.

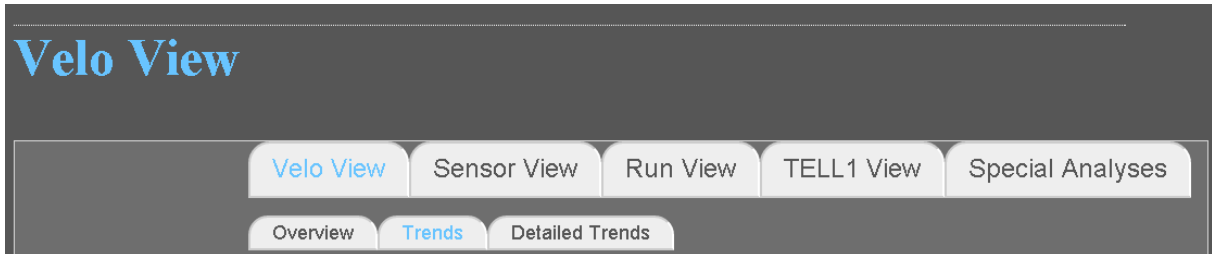


Figure 5.14: Django implementation.

5.3 Development of solutions based on the ROOTJS program

In this subsection, we present the developed solutions based on the ROOTJS program to fulfil the requirements. We also demonstrate parts of the modified codes and comment them.

The main goal was to develop new solutions and include them into the ROOTJS files that are responsible for the interface. The program was analysed and all interface related files were found. They are presented in the list below.

- *JSRootInterface.js*
- *JSRootInterface.css*
- *index.htm*¹

5.3.1 Developed solutions for index.htm

As mentioned earlier in this thesis, the original program was developed for the demonstration purpose and contained many features that were unnecessary for our application. We started the solution development process by locating these features in the code and removing them. Then, the *index.htm* file was studied. In the original program, the *index.htm* file contained the HTML structure of ROOTJS demo Web page. Since it was just a single page, only 1 HTML file was needed for the layout. For our application, one such file had to be created for each template that displays ROOT graphs. These files also had to be slightly adjusted in respect to the specifications of the tabs.

¹.htm is another extension to save HTML files

We developed new functions based on the *index.htm* file and copied them into the application templates with different configurations that corresponded to the individual tab specifications, as demonstrated in the diagram in Fig. 5.15.

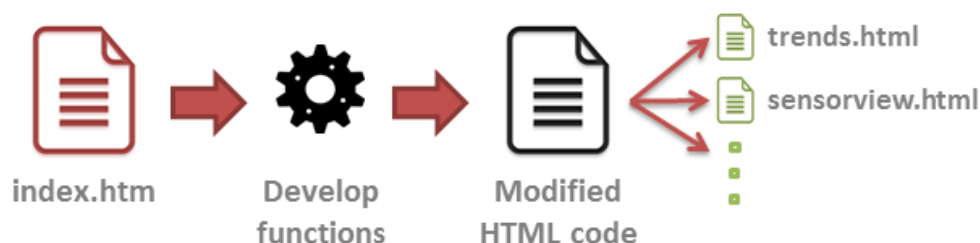


Figure 5.15: Developing functions based on the *index.htm* file.

The new template files were the starting points that initialize the ROOTJS program. This process was triggered when users press on any of the tabs that contained ROOT graphs. In Fig. 5.16 the *sensorview.html* template is presented. Some of its most important parts are covered below.

```
{% extends "webgui/bbase.html" %}

{% block menutitle %}Sensor View{% endblock %}
{% block sensorview %}
{% include "webgui/main_navigation.html" with active_tab='tab2'%}

<meta http-equiv="Content-type" content="text/html; charset=utf-8" />
<link rel="stylesheet" type="text/css" href="/static/style/JSRootInterfaceSV.css" />
<script type="text/javascript" src="/static/scripts/JSRootInterface.js"></script>

<input type="text" name="state" value="/static/files/h8.root" size="30" id="urlToLoad"/><br/>
<script>reportTwo = true;</script>
<script>ReadFile();</script>
<!-- <script>showObject('h2',1);</script> -->

<div id='status'></div>

<div id='reportHolder' class='column'>
  <table id=reportable>
    <tr id='report'></tr>
    <tr id='reportTwo'></tr>
  </table>
</div>

{% endblock %}
```

Figure 5.16: Part of the *sensorview.html* code.

The first part of the template code is similar to rest of the main templates in the VELO Web GUI application that were described earlier in the chapter. We can note that this template code

is linked to the style sheet file *JSRootInterfaceSV.css* and the ROOTJS interface file *JSRootInterface.js*. Moreover, the code also specifies how many graphs will be shown in the tab. For example, the template code presented above was created for the Sensor View tab. It contains 2x4 matrix of graphs, as it was shown in Fig. 4.5. The graph matrix request is done by setting up a flag *reportTwo=true* that notifies the *JSRootInterface.js* file which if-statement should be executed. After the flag is set, the template code calls the *ReadFile* function which is also located in the *JSRootInterface.js* file, and the read and display processes starts.

5.3.2 Developed solutions for JSRootInterface.js

The most important solutions that were developed and applied to the *JSRootInterface.js* file. This file is connected to all other core files of the ROOTJS program. A part of the code is presented in Fig. 5.17.

```
if(idx>1 && reportThree==true){
$("#reportThree").append("<td>"+entryInfo+"</td>");
} //for tabs that contain reportThree

else if(idx>3 && reportTwo==true){
$("#reportTwo").append("<td>"+entryInfo+"</td>");
console.log("inside else if");//for the debug purpose
} //for tabs that contain reportTwo

else{$("#report").append("<td>"+entryInfo+"</td>");
console.log("we should not be here more that 4 times in SensorView tab");//for the debug purpose
} //end else statement
```

Figure 5.17: if-statements.

The if-statements in the code process requests from the templates. The basic task of each if-statement is to create a new row of graphs according to the layout specification. For example, when the second if-statement is true the layout will contain 2x2 matrix of graphs.

The *JSRootInterface.js* file contain asynchronous code which means some parts of it can finish execution earlier that the other parts. Unfortunately, this caused various problems such as situations when graphs are not loaded at all. To avoid this undesired behaviour we implemented a small delay in the *showObject* function, as shown in Fig. 5.18.

```
function showObject(obj_name, cycle) {  
    console.log("showObject is starting"); //for the debug purpose  
    setTimeout(function() {gFile.ReadObject(obj_name, cycle);}, (100));  
};
```

Figure 5.18: Delay.

The delay ensured that all the graphs were loaded at all times when requested. Throughout our application tests this solution worked perfect each time when we accessed the tabs that loaded graphs. We present the tests in greater detail in Chapter 6.

Our last set of solutions were applied to the automatic display function, as shown in Fig. 5.19.

```
function displayListOfKeys(keys) {  
    // JSROOTPainter.displayListOfKeys(keys, '#status'); //original line  
    // JSROOTPainter.displayListOfKeys(); //unnecessary line for our project  
    showObject('h1', 1);  
    showObject('h2', 1);  
    showObject('h3', 1);  
    showObject('h4', 1);  
    showObject('h5', 1);  
    showObject('h6', 1);  
    showObject('h7', 1);  
    showObject('h8', 1);  
};
```

Figure 5.19: Graphs display.

In this figure, 8 code lines are demonstrated. They are responsible for loading 8 different graphs from a ROOT file. In the VELO Web GUI application, we utilized the same *JSRootInterface.js* file for all the templates. It was efficient because the duplication of the same code was avoided several times.

5.4 ROOTJS into GUI integration

This section of the report covers the last step of the implementation process of the VELO Web GUI application. We present how we integrated the ROOTJS program with the solutions that we developed, into the Web GUI. The main steps of that integration process were:

- to move the new ROOTJS program to the static folder of our Web GUI application

CHAPTER 5. SOLUTIONS AND IMPLEMENTATIONS

- to configure paths of the main interface file of the ROOTJS program (*JSRootInterface.js*)
- to verify the final result

The main goal of this integration process was to merge the two programs seamlessly. In the previous section, we already started to describe that process when we explained how we applied the developed solutions to the *index.htm*, as shown in Fig. 5.15. Our next challenge was to continue the integration process and move the rest of the ROOTJS files into our Django built Web GUI application. The static folder of the application was located and all the remaining files were put there. Furthermore, in order to link all the ROOTJS program files together, the paths were configured for the interface file *JSRootInterface.js* and all the template files that loaded graphs, as described in Appendix D. The integration of the ROOTJS program with all the developed solutions, into the Django GUI is demonstrated graphically in Fig. 5.20.

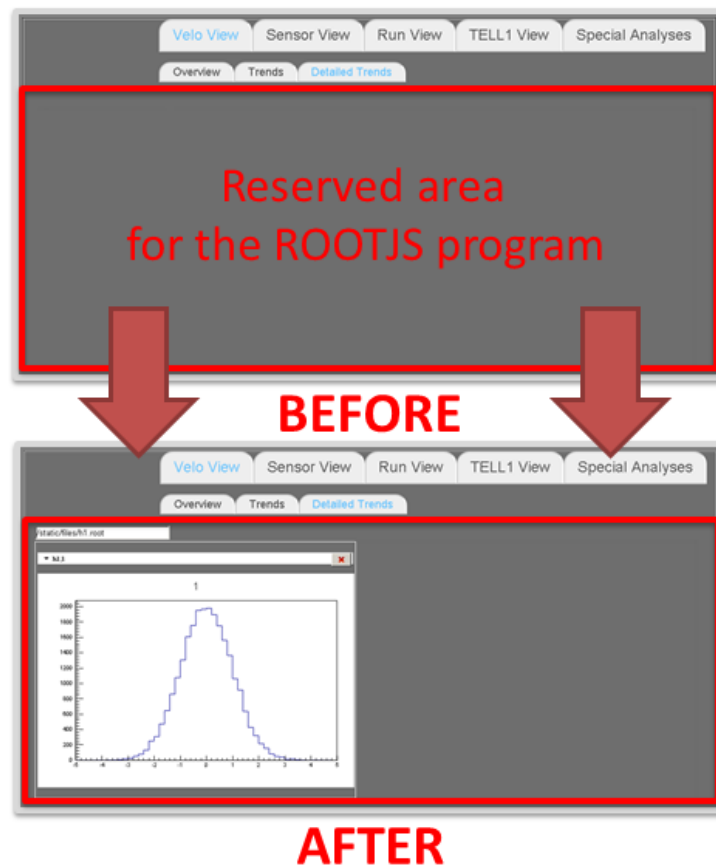


Figure 5.20: Integration overview.

Chapter 6

Tests and Demonstrations

This chapter covers the tests that were performed on the VELO Web GUI application. In the following 4 sections scenarios, tests, alternative solutions and numerical results are presented.

6.1 Test scenarios

To get some useful test results, it is necessary to define scenarios. With this in mind, 4 scenarios were created for the tests. They are presented in the list below.

- Scenario 1: Tests on verification of the implemented functions
- Scenario 2: Tests of the application in different browsers
- Scenario 3: Tests of the basic ROOT functionality
- Scenario 4: Tests of the auxiliary features

6.2 Performed tests

This part of the chapter shows the tests based on the 4 scenarios which were defined in the previous section. In these series of tests, the remote connection was established to available vir-

tual machines in the Online network. The VMs ran Microsoft Windows Server 2003 operative system, but in addition, we managed to acquire one Microsoft Windows Server 2012 machine. Specifications of these 2 VMs are shown in Table 6.1.

Table 6.1: Virtual Machines.

	VM 1	VM2
OS	Windows Server 2003	Windows Server 2012
processor	Intel Xeon 2.8GHz	Intel Core i7 9xx 2.39GHz
RAM	2GB	4GB
service pack	2	R2

We performed more than 20 identical tests per VM and per browser to gain good average results. We present these tests in greater detail in the following sub-sections.

6.2.1 Verification of implemented functionality

To verify the implemented functions, we used the same browser and VM as for the development process, Google Chrome v30.0.1599 and WS 2003 respectively. The following functionality was verified:

- Layouts
- Main tabs navigation bar
- Sub-tabs navigation bar
- Main tab title area
- Highlighting of the navigation bars
- ROOT file path window
- ROOT graphs
- Basic ROOT functionality such as zoom in and out of graphs

The layouts were loading correctly in each test. The position of the items in Chrome such as text, graphs and navigation bars was exact, as specified in the parameters. Main tabs and sub-tabs navigation bars tests revealed that these features were implemented in the right way. We were able to navigate from tab to tab without any problems. We also verified that the main tab title was changing accordingly, as we were switching from one main tab to another. Moreover, the highlighting worked properly, we could easily see the active tabs shined in blue color, as shown in Fig. 6.1.

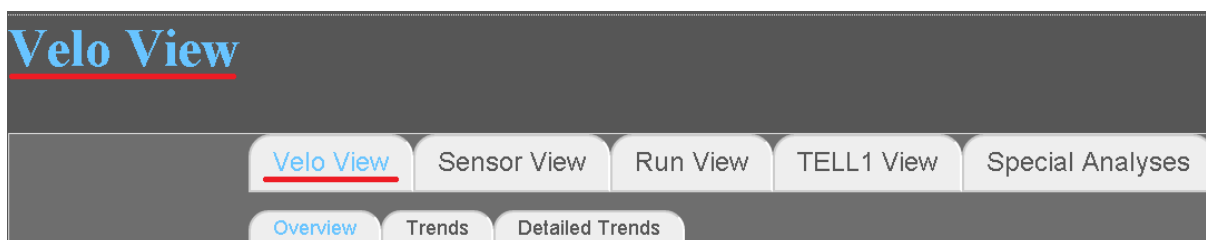


Figure 6.1: Active tab highlighting and Main tab title functions.

ROOT file path window indicated the path to the ROOT file that was read. We observed it many times during the tests and confirmed its functionality, as shown in Fig. 6.2



Figure 6.2: ROOT file path window.

Furthermore, we verified that ROOT graphs were loading when requested, without any errors or significant delays. This also confirmed that the integration process of the ROOTJS program with our developed solution, into the Web GUI was successful. The basic ROOT functionality tests showed that it was possible to zoom in and out of graphs. In fact, that could be utilized on multiple graphs of the same tab, as shown in Fig. 6.3. Note that the figure is cropped and does not show the main tabs navigation bar and the main tab title area.

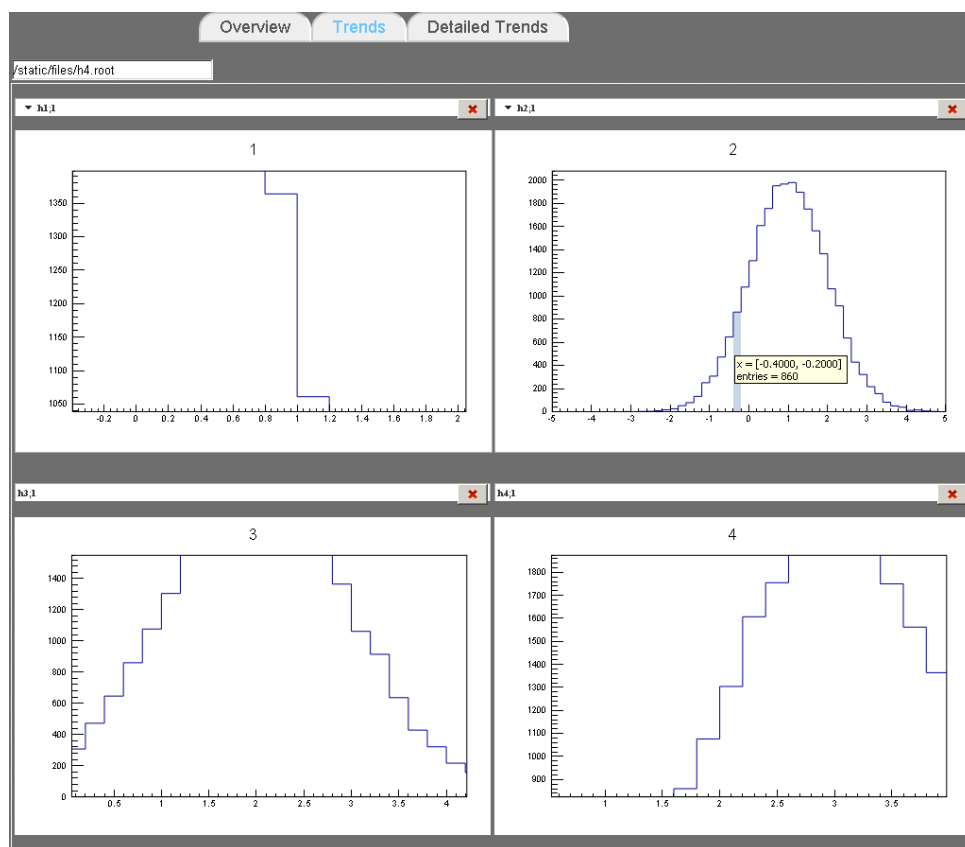


Figure 6.3: Zoom in, zoom out of ROOT histograms.

6.2.2 Browsers tests

The VELO Web GUI application was designed for the Web. The users have to install an Internet browser on their computers in order to use it. However, the application is not compatible with all browsers, mainly because of the integrated ROOTJS program. The developer of this program specified alert messages in his code for the non-compatible browsers. The browsers compatibility with the ROOTJS program is shown in Table 6.2.

Table 6.2: Browsers compatibility with the ROOTJS program.

compatible	non-compatible
Chrome	Opera
Firefox	Safari
IE (≥ 9)	

CHAPTER 6. TESTS AND DEMONSTRATIONS

Our own tests were done using the VELO Web GUI application in order to observe how it performs in different browsers. We also wanted to acknowledge whether the results from previous tests of the ROOTJS program could be directly applied to our application. The browsers that were tested are shown in the Table 6.3 below.

Table 6.3: Tested browsers.

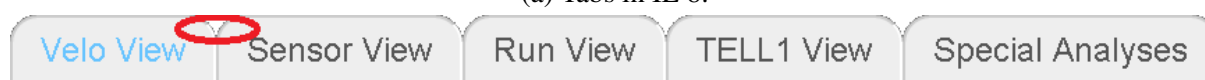
Browser	Version	GUI loading	ROOT graphs loading	Application crashes
Windows Server 2003				
Chrome	30.0.1599	X	X	
Firefox	17.0.5	X	X	
IE	8.0.6001	X		
Windows Server 2012				
Chrome	31.0.1650	X	X	
Firefox	25.0.1	X	X	
IE	11.0.9600	X		X
Opera	18.0.1284	X	X	
Safari	5.1.7	X	X	

Windows Server 2003

The results of Internet Explorer 8 tests showed that it did not support some of the modern graphic packages. That is why corners of the tabs were not smoothed as compared to how they looked in other browsers. The side by side comparison is shown in Fig. 6.4.



(a) Tabs in IE 8.



(b) Tabs in Chrome and Firefox.

Figure 6.4: Tabs comparison.

Unfortunately, our developed solutions based on the ROOTJS program did not work and the graphs were not loaded. The IE 8 browser showed an alert message that one should use at least

IE 9. Thus, the performed tests on IE 8 revealed that this browser was not compatible with the VELO Web GUI application. They also confirmed the previous tests, done by the developer of the ROOTJS program, that the versions of IE ≥ 9 were not compatible with the program.

The tests performed on Chrome and Firefox browsers showed that the application worked very stable. The tab navigation were functioning and it was possible to go forth and back through all the tabs without any problems. All ROOT files were loaded and displayed at all times.

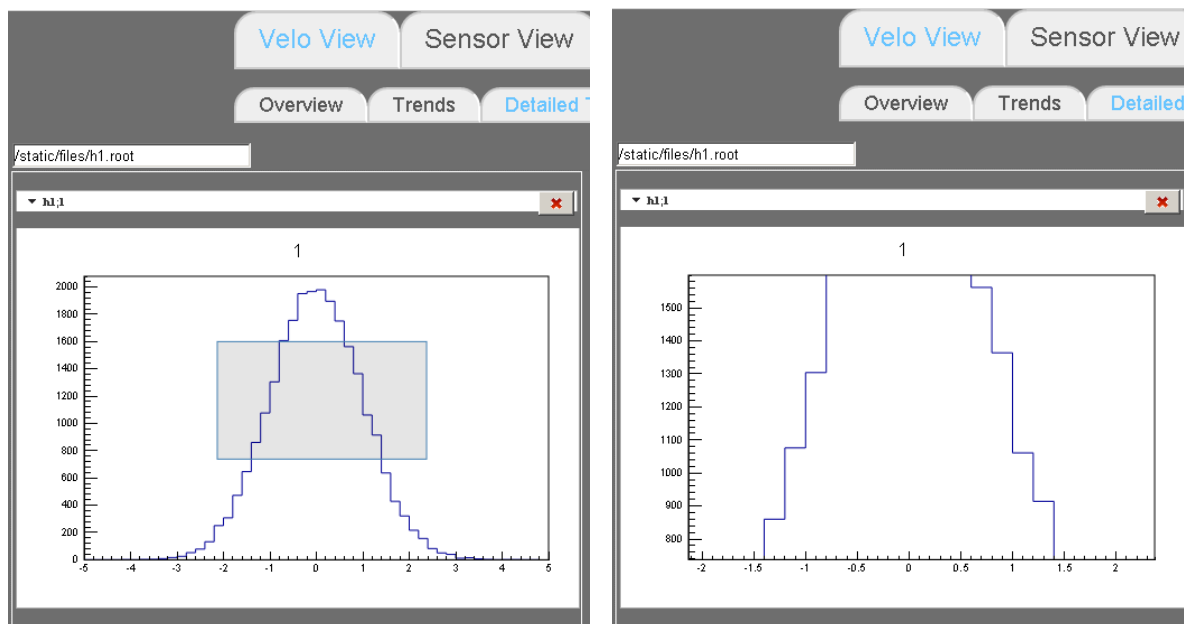
Windows Server 2012

Tests on the WS 2012 virtual machine showed similar results as compared to the WS 2003 tests. Google Chrome and Mozilla Firefox browsers worked stable and performed as expected. All ROOT graphs functioned normally and the application did not crash. However, IE 11 tests revealed that the browser loaded the GUI correctly, but it hanged and became unresponsive when the ROOT graphs were loaded.

Furthermore, Opera and Safari browsers were included for the further tests. Opera delivered a great performance despite the fact that it was suggested to avoid this browser, as stated in the *JSRootInterface.js* file. Newer versions of Safari browsers were not available for Windows. For our tests the last available Safari version for Windows was used. It was not mentioned about any compatibility details for Windows Server 2012 (Windows 8.1) on the Apple official Web page [20]. Nevertheless, the tests were successful and in general, we were satisfied with the performance of this browser.

6.2.3 Basic ROOT functionality tests

The requirements for this thesis specified that the VELO Web GUI application should be able to display 2D histograms and provide basic ROOT features such as zoom in and out of the graphs. The functionality of this feature was demonstrated in Subsection 2.2.1. The necessary tests were done using our application to ensure that it worked, as shown in Fig 6.5.



(a) Select an area to zoom in.

(b) The result of the zoom operation.

Figure 6.5: Zoom functionality in the VELO Web GUI application.

During the tests the zoom feature was used on 8 graphs in the same tab. We zoomed in on the selected area and the X and Y axis were scaled accordingly. To zoom back to the default point, we double-clicked on the white area on the graph. In conclusion, the VELO Web application fulfilled the predefined requirements of the basic ROOT functionality.

6.2.4 Auxiliary features tests

In this subsection, the implemented auxiliary features tests are presented. They were performed multiple times in different tabs, to ensure that the features work correctly. All the supported browsers were utilized to perform these tests.

First, the collapsible button feature was tested, as shown in Fig. 6.6. The word collapsible means that the histograms roll up and remain hidden until users click to roll them down again. One can think of this as roller blinds on windows at home. Please note that the following figures show only the ROOTJS graphs area of our application.

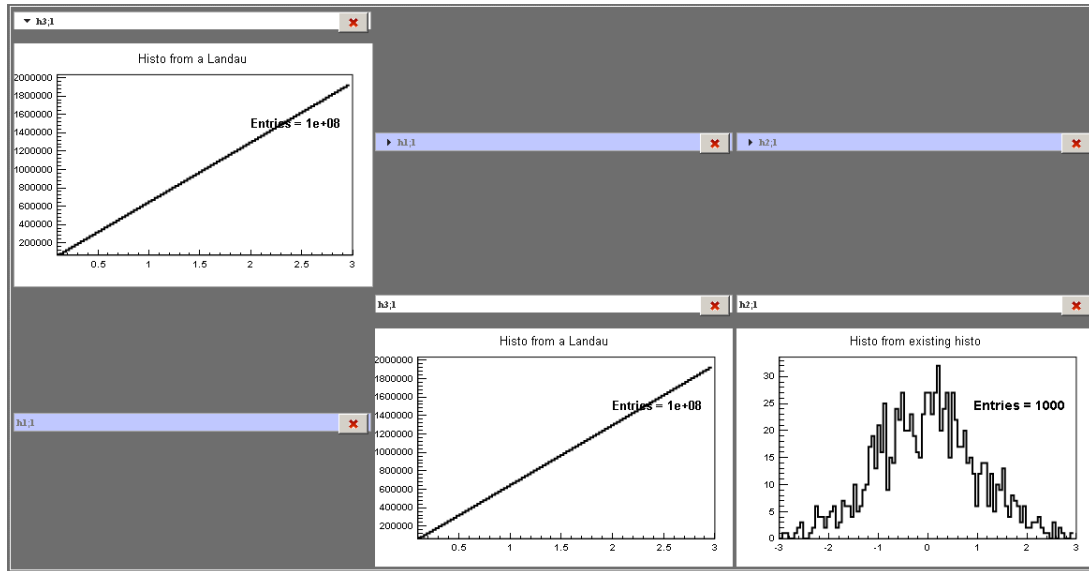


Figure 6.6: The collapsible feature.

The second auxiliary feature of the VELO Web GUI application is the delete button which is located in the right-top corner of each graph, as shown in Fig. 6.7.

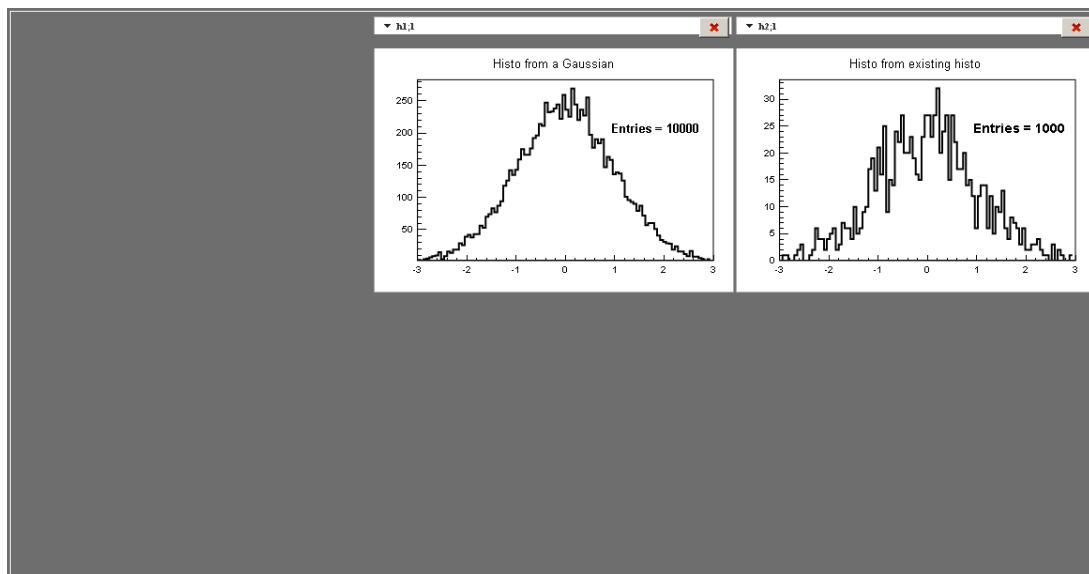


Figure 6.7: The delete feature.

The tests showed that both auxiliary features functioned properly. Moreover, no errors or crashes were observed during the tests in different compatible browsers.

6.3 Numerical results

This section covers the numerical results that were achieved from the VELO Web GUI application tests. In the following tests, response time of the Sensor View tab was captured. The response time is the time it took to load the Sensor View tab content. The code that was used to capture the data is also presented and explained.

6.3.1 Preparation

There were many other ways to perform such a test. For example, one could use available extensions or plugins for browsers. Nevertheless, we found a small code that could capture specific time. It was placed into the *JSRootInterface.js* file in the area, where the ROOT files are processed and their content is passed for display. The first line of the code captures the time when the *JSRootInterface.js* file begins the execution, as shown in Fig. 6.8.

```
var start = new Date().getTime(); //for the app performance statistics.
```

Figure 6.8: Execution start time.

This code was specifically adjusted for the Sensor View tab that reads and displays 8 graphs, as shown in Fig. 6.9

```
//execution time test for Sensor View tab
if(idx==7){
var end = new Date().getTime();
var time = end - start;
alert('Execution time: ' + time);
}
```

Figure 6.9: Execution end time and the result difference.

This part of the code captures the time after all 8 graphs are displayed. When the start and the end time values are available, the code calculates the difference between them and displays the result as a pop-up message inside browser, as shown in Fig. 6.10. The message shows two fields, one with the Web page address and the second one with the execution time. This message was used to gather all the time values and calculate the average time for the tab to load.



Figure 6.10: Alert message in Chrome.

6.3.2 Results

In order to get a better overview of the achieved results we created a table, as shown below.

Table 6.4: Performance results.

Attempt#	Response time (ms)					
	WS 2003		WS 2012			
	Chrome	Firefox	Chrome	Opera	Firefox	Safari
1	932	1513	625	672	1230	1044
2	780	1652	812	640	1052	949
3	657	1685	500	672	1298	1285
4	909	1623	875	500	977	1096
5	611	1583	610	531	1156	1134
6	681	1702	750	657	1064	1184
7	872	1671	640	578	1197	1094
8	661	1678	640	562	1192	1145
9	672	1668	625	515	1007	1077
10	625	1635	438	641	939	977
11	633	1669	516	640	1413	958
12	634	1722	625	641	1538	966
13	874	1663	625	515	1054	1074
14	837	1679	625	641	1274	1014
15	625	1762	500	829	1618	1084
16	656	1689	531	532	1252	1072
17	651	1760	516	640	1074	860
18	868	1608	437	625	1267	1287
19	613	1629	563	500	1063	901
20	845	1659	609	672	1275	830
avarage:	732	1663	603	610	1197	1052

Table 6.4 presents the response time, that is, how many milliseconds it took to load the

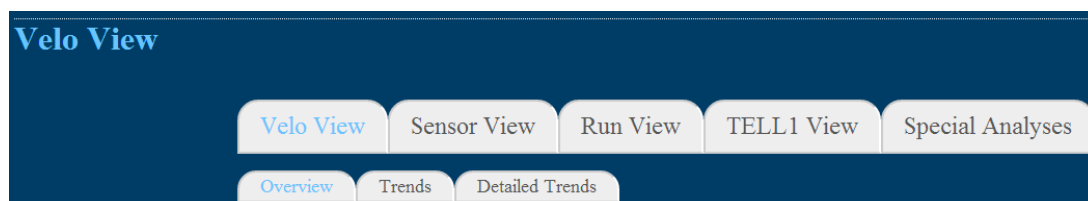
Sensor View tab and how many times we performed the same test. The examined results from the WS 2003 tests led to the conclusion that the Mozilla Firefox browser used more than twice as much time to load the Sensor View tab as the Google Chrome. We think that the reason for that is the browser's ability to execute JavaScript. The tests that were accomplished on the WS 2012 virtual machine, confirmed the Chrome's superior efficiency. Another point of the result analysis was that Opera performed almost as good as Chrome. Safari and Firefox came on 3rd and 4th place in the response time ranking respectively.

Additionally, it was noted that when the browsers were opened for the first time and loaded the entire application, the graphs appeared after longer delay than usually (aprox. +500ms). The same ROOT file was used throughout these tests, so that the result differences could be easily observed. When using different ROOT files, the results may be different.

6.4 Alternative solutions

In this part of the chapter, we discuss alternative solutions of the tab navigation that were created and considered for the final application. We present the corresponding figures, write about the advantages, disadvantages and make a conclusion of our selection.

In the early stage of the GUI development, two different implementations of the tab navigation were developed, as shown in Fig. 6.11.



(a) Django implementation.



(b) JavaScript implementation.

Figure 6.11: Tab navigations.

JavaScript and Django versions were brought for the comparison where the most suitable one would be chosen for the final application, as shown in Fig. 6.11b and Fig. 6.11a respectively. JavaScript version had an advantage because it was quick and easy to implement. Another property of the given version was the unchangeable Unique Resource Locator (URL)¹ while switching between the tabs. However, implementing tabs inside tabs turned out to be more complicated and this showed to be the disadvantage of the JavaScript solution.

We continued to experiment and tried the native Django implementation of the tab navigation. That is, using only Django features such as templates. It took more time to develop this version, but in return it fulfilled all the requirements. The tab-in-tab navigation was developed using a smart and easy solution which was presented earlier in the thesis. Moreover, it had a special feature - the highlighting of active tabs. It was especially useful since the given functionality was not standardised in Django and required additionally developed solutions. The native Django implementation showed to be the clear advantage. This solution also provided unique URLs for each tab and sub-tab.

¹An example of the URL: *[http : //www.aftenposten.no/noenytt_Idag/veldiginteressant.snd](http://www.aftenposten.no/noenytt_Idag/veldiginteressant.snd)*

Chapter 7

Discussions

This chapter of the thesis covers the discussions part of our project thesis. Alternative programs of the ROOTJS program and alternative servers are presented. We also justify why they were not selected for the final application.

7.1 ROOTJS alternatives

In this section, the alternative programs for the ROOTJS program are introduced. The section also covers the advantages and disadvantages of these programs.

In the beginning phase of the VELO Web GUI application development, there were 3 candidate programs that handle ROOT files in a Web browser:

- WebOOT
- Highcharts
- ROOTJS

WebOOT is a ROOT file viewer for the Web. The advantage of this program is that it provides the functionality to display ROOT files on the Web. The disadvantage of WebOOT, in

our case, is that it generates only image files (e.g. PNG) to display graphs from ROOT files. Thus, the interactive graphs are not supported [21].

Highcharts is the JavaScript charting engine. Highcharts offers users the HTML5/JavaScript libraries with intuitive and interactive charts [22]. However, this program was not suitable for our thesis project because it does not include the functionality to read ROOT files.

Finally, the ROOTJS program remained the most attractive solution out of the 3. We considered the advantages and disadvantages of all three programs and finally decided to select the ROOTJS program because it was more suitable for our project considering the requirements.

7.2 Apache alternatives

This part of the chapter covers our selection of servers. We present alternative servers and write about why we selected the Apache server for our project.

There are many good servers available out there, both open source and proprietary. The list below presents some of the alternative servers which were considered for this thesis project:

- Apache
- nginx
- lighttpd

All these servers are under the open source licences. Apache server was already presented earlier in this thesis report. nginx, pronounced as "engine x", is a server built for high performance. It is also good for the beginners because of its configuration simplicity. lighttpd is a fast and efficient server that is easy to use straight out of the box, but it is not very suitable for more advanced tasks. To choose the right server we created a list with some criteria that a server should fulfil:

- Server should be open source
- Server should be designed for Linux OS

CHAPTER 7. DISCUSSIONS

- Server should be well documented
- It should be relatively easy to configure the server for Django applications

We selected Apache server for our project because it fulfils most of our criteria. We had some experience using it before. Additionally, it is the world's most popular and widely used Web server, so there is big amount of help and tutorials available on the Internet. Django project Web pages contained good examples how to configure the Apache server for Django applications.

Chapter 8

Conclusion and Future Work

In this part of our thesis, the summary and the conclusions are drawn. The achieved results are also presented together with the future work.

8.1 Summary of the thesis work

This section presents the developed Web DQM application for the VELO module of the LHCb experiment. The development process was based on the requirements that were specified and presented in the beginning stage of this thesis project. They served us as the guidelines throughout the entire thesis project.

The Web GUI was created using Django Web Framework. Moreover, we referred to the ROOTJS program that was designed for reading ROOT files, processing them and displaying their content in a Web browser. It was used as a platform for the development of new functionalities. Furthermore, the Web GUI program and the new ROOTJS program were merged together to create the final application - the VELO Web GUI application. Then, the scenarios were defined for testing and the application was tested according to them. Finally, the achieved results were presented and commented.

Additionally, the alternative programs of the ROOTJS program and alternative servers were introduced. We discussed why they were not selected for the final application.

8.2 Contributions

This section covers the contributions of this thesis project. The list that shows the main contributions is presented below. In this thesis project we:

- designed the framework for the application
- developed the VELO Web GUI application
- tested the application in various scenarios
- verified the implemented functions

More specifically, we developed the VELO Web GUI application that reads, processes and displays the ROOT file content in a Web browser. We also fulfilled all the requirements for the design and functionality of the application which were presented in Chapter 4. We set up tests scenarios and tested the application according to them. We gathered and presented the important results. Moreover, we verified that the implemented functionalities work.

8.3 Future work

The future work would be to enhance the developed functions for the *JSRootInterface.js* file and Django templates. One of the proposed enhancements is to make graph display function more flexible. This can be achieved by creating an array variable $h[]$ that substitutes 'h1', 'h2',..., 'h8' histogram names. The $h[]$ variable can copy the names of appropriate histograms from some configuration file. Additionally, a for loop should be set to 8 repetitions since there are maximum 8 graphs per tab. The for loop contains a counter i that has to be placed inside the array $h[i]$ so that the right names of the histograms are read one by one as the counter goes.

Many Django templates of the VELO Web GUI application contain similar and identical CSS specifications. This is quite normal, but to make it even simpler and thus gain a better overview of the templates' code we can cut and move similar CSS configurations into separate files. These CSS files can be dynamically loaded upon request from the templates that require them.

CHAPTER 8. CONCLUSION AND FUTURE WORK

These two improvements are valuable for the future versions of the application. However, they have to be implemented, tested and verified.

Bibliography

- [1] *About CERN*. Available: <http://home.web.cern.ch/about>. Visited [July 31, 2013].
- [2] *Introduction to the LHC*. Available:
<http://home.web.cern.ch/about/accelerators/large-hadron-collider>.
Visited [August 1, 2013].
- [3] *ATLAS photos*. Available: <http://www.atlas.ch/photos/lhc.html>.
Visited [August 16, 2013].
- [4] *The LHCb Experiment*, Communication Group, CERN-BROCHURE-2009-007-Eng, CERN, Geneva, June 2009.
- [5] *The LHCb experiment image*. Available: <http://en.wikipedia.org/wiki/LHCb>.
Visited [November 28, 2013].
- [6] *Vertex Locator (VELO)*. Available:
<http://lhcb-public.web.cern.ch/lhcb-public/en/Detector/VELO-en.html>.
Visited [November 29, 2013].
- [7] *What is ROOT?* Available: <http://root.cern.ch/drupal/content/about>.
Visited [March 12, 2013].
- [8] *How to use ROOT with Python (PyROOT)?* Available:
<http://root.cern.ch/drupal/content/pyroot>. Visited [March 12, 2013].
- [9] *About ROOTpy*. Available: <http://rootpy.org>. Visited [March 12, 2013].
- [10] *What is Pidoco*. Available: <https://pidoco.com/>. Visited [August 14, 2013].
- [11] *About Python*. Available: <http://python.org/about/>. Visited [June 28, 2013].
- [12] *Django Book, Chapter 1*. Available:
<http://www.djangobook.com/en/2.0/chapter01.html#django-s-history>.
Visited [June 19, 2013].

BIBLIOGRAPHY

- [13] *What is JavaScript*. Available:
<https://developer.mozilla.org/en-US/docs/Web/JavaScript>.
Visited [July 4, 2013].
- [14] *About Apache*. Available: http://httpd.apache.org/ABOUT_APACHE.html.
Visited [July 5, 2013].
- [15] *What is mod_wsgi*. Available:
<https://code.google.com/p/modwsgi/>. Visited [July 5, 2013].
- [16] *X-Win32: The Best PC X Server for X Windows Display on Your LAN*. Available:
<http://www.starnet.com/xwin32/>. Visited [August 16, 2013].
- [17] *Powerful Python and Django IDE*. Available:
<http://www.jetbrains.com/pycharm/index.html>. Visited [August 20, 2013].
- [18] E. Rodrigues, "A GUI Framework for Offline VELO Monitoring", University of Glasgow, U.K., March 2010.
- [19] *Django Book, Chapter 4*. Available:
<http://www.djangobook.com/en/2.0/chapter04.html>. Visited [September 6, 2013].
- [20] *About Safari 5.1.7 for Windows*. Available: <http://support.apple.com/kb/dl1531>.
Visited [November 26, 2013].
- [21] *WebOOT's documentation*. Available:
<http://weboot.readthedocs.org/en/latest/index.html>. Visited [October 22, 2013].
- [22] *About Highcharts JS*. Available:
<http://www.highcharts.com/>. Visited [October 22, 2013].
- [23] *Quick Installation Guide*. Available:
<https://code.google.com/p/modwsgi/wiki/QuickInstallationGuide>.
Visited [June 24, 2013].
- [24] *Simple DRY Tabs using Django 1.3*. Available:
<https://djangosnippets.org/snippets/2421/>. Visited [November 8, 2013].

Appendix A

Configuration of the `httpd.conf` File

This appendix covers the configuration of the `httpd.conf` file. `httpd.conf` file is located at the end of this path `/etc/httpd/conf/httpd.conf`. Configuration of the `httpd.conf` file is presented below:

1. In the `LoadModule` part of the file, add `wsgi_module` modules `mod_wsgi.so` [23].
2. At the bottom, uncomment `NameVirtualHost` line and add your development server computer's IP address like this `NameVirtualHost xxxx.xxxx.xxxx.xxxx`.
3. Uncomment `VirtualHost` line and add your development server computer's IP address:
`<VirtualHost xxxx.xxxx.xxxx.xxxx> .`
4. Inside `VirtualHost` area, add `Alias` to navigate to your Django project static folder: `Alias /static/ /etc/wwwmain/webvelogui/static/ .`
5. Inside `VirtualHost` add:
`<Directory /etc/wwwmain/webvelogui/static>`
`Order deny,allow`
`Allow from all`
`</Directory>`
6. Inside `VirtualHost`, point to the location of your Django project WSGI configuration file as follows: `WSGIScriptAlias /etc/wwwmain/webvelogui/conf/webvelogui.wsgi`

APPENDIX A. CONFIGURATION OF THE HTTPD.CONF FILE

We have configured our *httpd.conf* file, as shown in Fig. A.1.



```
pprykhod@velomongui:~ - Shell - Konsole
Session Edit View Bookmarks Settings Help

NameVirtualHost 10.128.126.50
#
# NOTE: NameVirtualHost cannot be used without a port specifier
# (e.g. :80) if mod_ssl is being used, due to the nature of the
# SSL protocol.
#
#
# VirtualHost example:
# Almost any Apache directive may go into a VirtualHost container.
# The first VirtualHost section is used for requests without a known
# server name.
#
<VirtualHost 10.128.126.50>
#<VirtualHost *:80>
#   ServerAdmin webmaster@dummy-host.example.com
#   DocumentRoot /www/docs/dummy-host.example.com
#   ServerName dummy-host.example.com
#   ErrorLog logs/dummy-host.example.com-error_log
#   CustomLog logs/dummy-host.example.com-access_log common

#ServerName www.veloview.com
#DocumentRoot /etc/www/veloview/static

#<Directory /etc/www/veloview/static>
#<Directory /etc/wwwmain>
#Order allow,deny
#Allow from all
#</Directory>

Alias /static/ /etc/wwwmain/webvelogui/static/

<Directory /etc/wwwmain/webvelogui/static>
Order deny,allow
Allow from all
</Directory>

#Setting up configuration
WSGIScriptAlias /etc/wwwmain/webvelogui/confile/webvelogui.wsgi

<Directory /etc/wwwmain/webvelogui/confile>
Order allow,deny
Allow from all
</Directory>

#VirtualHost>
-- INSERT --                               1035,1      Bot
```

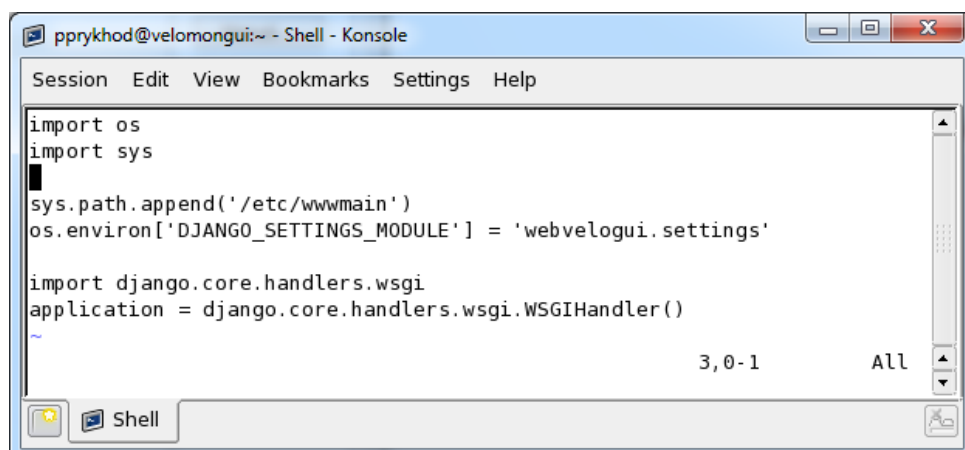
Figure A.1: *httpd.conf* file.

Appendix B

Configuration of the WSGI File

In this appendix, we create and configure the WSGI file. A step by step guide on how to do that is also presented.

To start with, create a WSGI file using VIM editor or any other text editor. Save the file with the name of your Django project (e.g. webvelogui) and .wsgi extension. Then, go to the folder where the Django projects' *setting.py* file is located. Create a sub-folder there and name it *confiles* for example. Put your WSGI file inside this sub-folder. This is done for the security precautions. The detailed overview of what the WSGI file should contain is shown in Fig. B.1.

A screenshot of a terminal window titled "pprykhod@velomongui:~ - Shell - Konsole". The window contains a menu bar with "Session", "Edit", "View", "Bookmarks", "Settings", and "Help". The main area shows Python code for a WSGI file:

```
import os
import sys

sys.path.append('/etc/wwwmain')
os.environ['DJANGO_SETTINGS_MODULE'] = 'webvelogui.settings'

import django.core.handlers.wsgi
application = django.core.handlers.wsgi.WSGIHandler()

~
```

At the bottom right of the terminal, it shows "3,0-1" and "All". The terminal has a "Shell" icon in the bottom left corner.

Figure B.1: *youprojectname.wsgi* file.

Appendix C

Configuration of the `setting.py` File

This appendix covers the configuration of `setting.py` file. Before proceed the readers should be familiar with Django and have the basic knowledge of it. This appendix describes how to configure an existing Django project to work with Apache. For more details about Django Web Framework refer to www.djangoproject.com.

To configure the Django project, use the project's `settings.py` file. Open it with either VIM on Linux or any other file editor. In DATABASES area, if the sqlite database is used simply create a path to the folder where the project's database file is located, as shown in Fig. C.1.

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3', # Add 'postgresql_psycopg2', 'mysql', 'sqlite3' or 'oracle'.
        'NAME': '/etc/wwwmain/webgui.db',# Or path to database file if using sqlite3.
```

Figure C.1: Databases field in the `setting.py` file.

Then, add the path to the `static` folder where the static files are preferred to be collected (e.g. JS, CSS and IMG files etc), as shown in Fig. C.2. Usually, this folder is created in the root directory of a Django project.

```
# Absolute path to the directory static files should be collected to.
# Don't put anything in this directory yourself; store your static files
# in apps' "static/" subdirectories and in STATICFILES_DIRS.
# Example: "/var/www/example.com/static/"
STATIC_ROOT = '/etc/wwwmain/webvelogui/static/'
```

Figure C.2: ROOT static path in the `setting.py` file.

Appendix D

Adjusting Correct Paths

In this appendix, we explain how to add correct paths in the ROOTJS program. These are the files that have to be modified:

- *JSRootInterface.js*
- templates with that load ROOT graphs

Open *JSRootInterface.js* file, in the top area locate `var source_folder = ""` and change it to be equal `"/static/"`. Make sure to use forward slash both before word static and after it. The result should look like the one shown in Fig. D.1.

```
var source_dir = "/static/";  
var gFile;  
var obj_list = new Array();
```

Figure D.1: *JSRootInterface.js* file.

Then, check that all templates that load ROOT graphs have the right paths. By default these paths should be correct, because when the *index.htm* file was modified the same code was copied to all the templates. Nevertheless, locate the lines `<link rel...>` and `<script type...>` and check if the paths to *JSRootInterface.js* and *JSRootInterface.css* are correct. If not, specify new paths `/static/scripts/JSRootInterface.js` and `/static/style/JSRootInterface.css`, as shown in Fig. D.2.

APPENDIX D. ADJUSTING CORRECT PATHS

```
<link rel="stylesheet" type="text/css" href="/static/style/JSRootInterface.css" />  
<script type="text/javascript" src="/static/scripts/JSRootInterface.js"></script>
```

Figure D.2: Template file configuration.

Appendix E

Developed Solutions for the CSS files

This appendix presents the solutions that were developed for the CSS files of the ROOTJS program. The part that was modified is shown in Fig. E.1.

```
#main { width:800px; height:400px; overflow: auto;}  
#reportable, td {border: 1px solid white;}  
#report, td { height:450px; width:450px;}  
#status { font-size:70%; }  
  
#reportHolder{  
    width: 1200px;  
    height: 800px;  
    font-size:9pt;  
    font-weight:normal;  
    color:#BDBDBD;  
    margin-left:0px  
    display: inline-block;  
    list-style-type: none; /*avoids the <ul> dots*/  
    padding: 0px 0;  
    clear:none;  
}
```

Figure E.1: CSS file.

The given CSS file specifies the parameters such as size and position of different HTML items. In this particular example that was shown in the figure above, the graph sizes were adjusted along with their positions according to the available space in a given tab.