ISIMA - Campus Universitaire des Cezeaux
1, Rue de la Chebarde
TSA 60125, CS 60026
63178 Aubière CEDEX

CERN
Route de Meyrin 385
Meyrin, Switzerland

Engineer Report

3rd Year Internship

LHCb Performance and Regression Development

Presented by: **Amine Ben hammou**

CERN Supervisor:
Benjamin Couturier

Internship duration:
5 months

ISIMA Supervisor:
MESNARD Emmanuel

Date of presentation:
August 28th 2015

Acknowledgments

Figures Table

Abstract

Contents Table

Abbreviations Table

3TP	Three Tiered Programming
Ajax	Asynchronous JavaScript and XML
ALICE	Compact Muon Solenoid
CERN	The European Organization for Nuclear Research
CMS	Compact Muon Solenoid
CSS	Cascading Style Sheets
DOM	Document Object Model
HTML	HyperText Markup Language
ISIMA	Institut Supérieur d’Informatique, de Modélisation et de leurs Applications
JSON	JavaScript Object Notation
LHC	Large Hadron Collider
LHCb	Large Hadron Collider beauty
LHCb PR	LHCb Performance and Regression
npm	Node Package Manager
REST	Representational State Transfer

Introduction

As part of my third year at ISIMA, I did my final internship at CERN. More specifically, I worked on the LHCb experiment and participated on the development of LHCb Performance and Regression framework (LHCb PR).

The goal of the LHCb PR project is to provide developers with a profiling framework helping them to evaluate their recent changes by running analysis modules and comparing results. The results can be used to detect fails in functionalities or performance issues.

During my internship, I worked on the development of the new version of LHCbPR. My tasks were to upgrade the web application using AngularJs framework and the development of analysis modules for the new version. This version should provide the same functionalities as the old one and improve the user interface. The application should also give developers the ability to add new modules easily and without having to change the code of the core application.

This report presents the different task I have done during my five months internship and the results I obtained. After presenting CERN and LHCb experiment, I will explain the idea of the LCHbPR project and describe the old version of its web application. Then I will present the different parts of the new version. After this I will explain in details how the new frontend web application was made. Finaly I will present the analysis modules made in the new version.

1. Presentation of CERN and LHCb

1.1. Presentation of CERN

CERN is the European Organization for Nuclear Research. Physicists and engenieers are working in this organization to discover how the universe works by studying the basic constituents of matter - the fundamental particles. Particles are made to collide together using the largest machine in the world to see how they interact and try to figure out the fundamental laws of nature. It was founded in 1954, and it sits astride the Franco-Swiss border near Geneva. It was one of Europe's first joint ventures and now has 21 member states.

Before they collide, the speed of particles is increased to close the speed of light using a sequence of accelerators. The largest accelerator is called the Large Hadron Collider (LHC) which is a ring with a perimeter of 27km. Detectors are used in the locations of collisions to observe and record the results.

There are seven experiments running on the LHC. The biggest four are: ATLAS, CMS (Compact Muon Solenoid), ALICE (A Large Ion Collider Experiment) and LHCb (Large Hadron Collider beauty). ATLAS and CMS are general-purpose experiments investigating the largest range of physics possible. ALICE and LHCb have detectors specialized for focussing on specific phenomena.

1.2. Presentation of LHCb

The LHCb experiment is one of seven particle physics detector experiments collecting data at the LHC. This experiment is trying to answer the question: "Why our universe is composed almost entirely of matter, but no antimatter ?".

Antimatter is made of anti-particules. An anti-particule of a particle p is a particle having exactly the opposite properties of p. When p and its anti-particule fusion, they cancel each other and produce pure energy. We think that just before the big bang, all the energy of the universe was compressed in one small point. So, at the moment of the big bang, this energy should have given the same quantity of matter and antimatter. The question is why do we observe only matter in our current universe ?

When the LHCb experiment is running, the detector registers about 10 million collisions per second. It is impossible to record all of these collisions.So the solution was to use an electronic system that will select the best and the most interesting collisions to be recorded. The recorded events are stored in binary files into disks. Then Physics are using specifique softwares to extract and study data on these files.

2. The LHCbPR project

2.1. What is LHCbPR

LHCb Performance and Regression (LHCbPR) is a service designed to record important measurements about results of integration and performance tests of the LHCb applications. These applications receive input in the form of configuration files and produce, as an output, various information. LHCbPR is not intended to actually run the jobs, but instead to manage and track the job results. The LHCbPR is a framework that allows LHCb software developer to push information for a run of their code(job charactetistics, results, performance measures, files) to a central database. The main goal is to give developers the ability to perform analysis of the data across versions, running options and plateforms.

2.2. Why LHCbPR

In the past, the users had to run test jobs manually using a configuration file. A job could be run for a specific application and results should be saved by the user. These results was shared between users using static documents such as HTML, CSV or e-mail.

LHCbPR was conceived as a tool to reliably organize the process of configuring and monitoring a test job execution. The framework, solves the problem of gathering the results. It provides a complete script that can produce the desired output, according to a configuration file. This script uses a list of handlers that collect the defined aspects of the job results and push them to the database. By collecting the results in such an organized manner, it provides the solution to the second problem, mentioned above; the running of the analysis.

Since, all the data are stored in the database, the framework provides an abstract and easy way to deploy algorithms and functions which perform analysis on the saved data.

Finally, LHCbPR handles the presentation of the collected data by providing a set of templates for the creation of web pages specific to each analysis and customized to the preferences of each user.

2.3. Users of LHCbPR

The LHCbPR users can be divided into three categories: Administrators, Application developers and End users.

The administrators group is responsible for the integrity of the collected data and the efficiency of the service. They maintain and support the system, making sure the application is functional and available to the end users.

Application developers are the users who actively design and develop modules for the framework, thus extending the functionality of the application.

Finally, the end users are the main body of the users of the service. They put the tools provided by the application developers to practical use and their job results are populating the database.

2.4. The old LHCbPR application

As my task is to upgrade the LHCbPR web application. I first started by reviewing the old version.

The LHCbPR follows the architectural model of three tiered programming. 3TP provides a way to theoretically categorize the components of an application, providing abstract modularity. In essence the 3TP model is a client-server architecture which is composed of three layers. Namely, the presentation tier, the functional process logic tier and the data storage tier. These tiers communicate with one another in a strictly defined way, which allows the developer to independently maintain each tier. In most practical applications of the model the presentation layer includes all forms of user interaction with the service. The process logic tier encompasses the whole of the application functionality, while the data storage tier handles the flow of the information from and to the data source (in most cases a database).

The Django framework was used to create the LHCbPR web application and the 3TP model was applied as follows:

Presentation Layer: A set of web pages using the jQuery library.

Application Logic Layer: A set of python modules made using the Django framework.

Data Storage Layer: An Oracle database storing all the applications and jobs informations and results.

3. New LHCbPR application

Instead of trying to improve the old version of the LHCbPR application, we started the new version from scratch to be able to use the new technologies of web development which was not used in the old version like Gulp task runner and the AngularJS framework. This framework enables us to build a rich client application using REST protocol for data access.

3.1. Architecture

The new version follows the 3TP model too, but in a more modern way. In particular, the presentation layer and the application logic layer are no longer in the same application. They are now two totaly separated applications communicating using a REST API. ...

3.2. Data Storage Layer: Database

We kept the same database as the old version but made some little changes to it. The new database entity relation schema is presented on figure in the next page. The main entities are the following:

Application: represents a software used by LHCb to process or analyse data. it is described by a name and has many versions

Application Version: a version belongs to an application. It can be a release or a nightly build (intermediate version compiled every night).

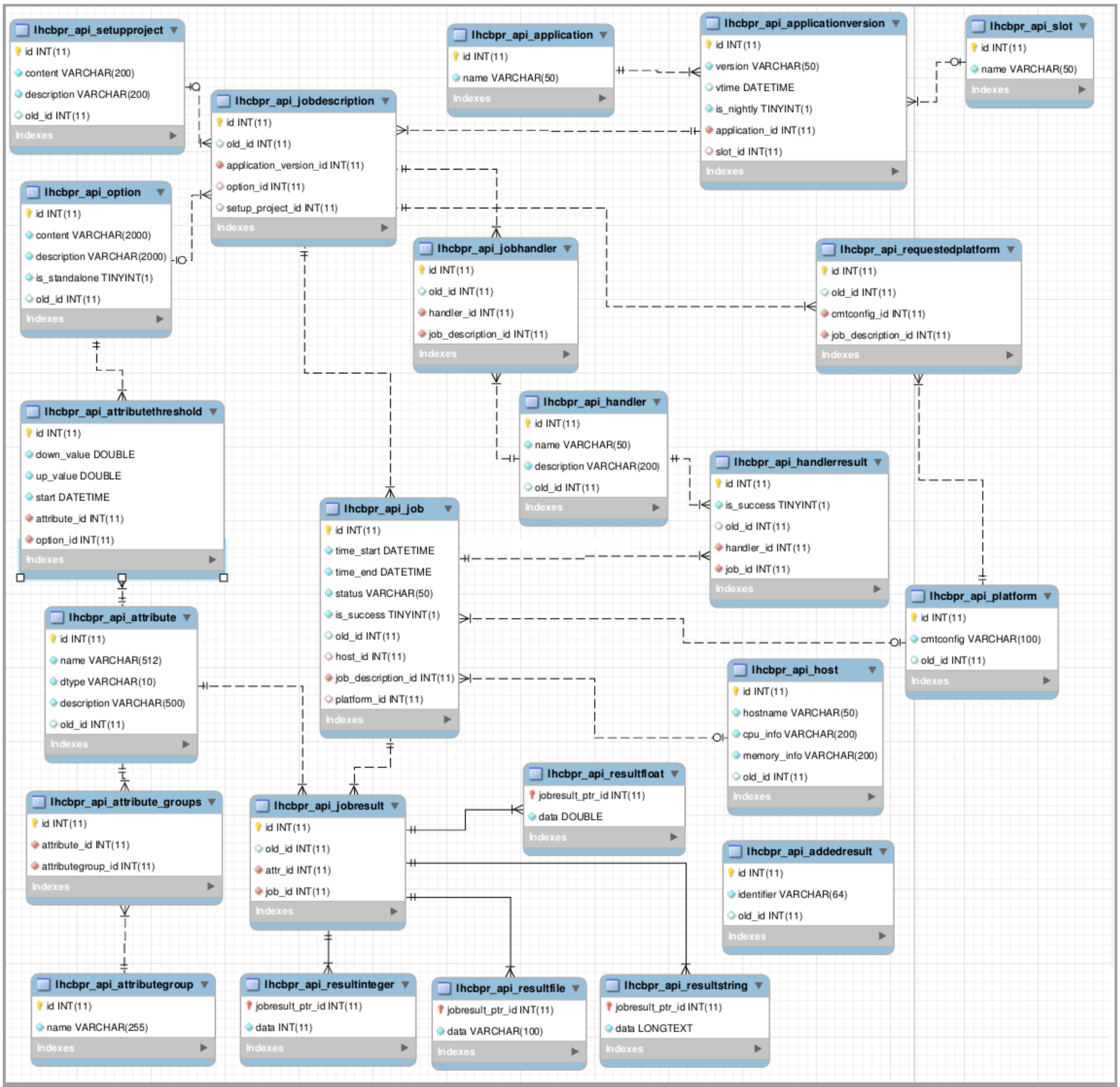


Figure 1: Database Entity Relation Schema

Job Description: This entity describes all that is needed to define a job. In particular, it specifies the application version, the project setup, the running option, the plateforms in which jobs will be executed and the handlers that will collect results.

Handler: a handler has a name and a description. It gethers data after the job has been run. It is linked to multiple job descriptions via the "job handler" pivot table. It is also linked to many jobs via the "handler result" pivot table.

Option: describes the command-line options to pass when running the jobs. I has a many to many relation with the attribute entity.

Attribute: an attribute can belong to multiple attribute groups. It also has a many to many relation with the job entity via the jobresult. Job results can have different types (integer, float, file, string).

Job: This is the main entity that describes a job that has been run. It has a start time, an end time, and a status. It belongs to a job description and is run in a specifique plateform and host. Their results can be collected by multiple handlers and it can have multiple results. Every job result concerns an attribute.

3.3. Application Logic Layer: REST API

This layer is made using the **Django REST framework** which is a powerful and flexible toolkit that makes it easy to build Web APIs using Python language. Making the application logic layer as an API is the best choice to keep layers totaly independent. This way we can build multiple frontend applications (one for the web, an other for mobiles for example) based on the same API.

REST stands for **Representational State Transfer**. (It is sometimes spelled "ReST") It relies on a stateless, client-server, cacheable communications protocol. In virtually all cases, the HTTP protocol is used. REST is an architecture style for designing networked applications.

In our case, the frontend application send requests to the API and the API responds with the requested data in JSON (JavaScript Object Notation) format which is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. It is based on a subset of the JavaScript programming language which make it easy to parse it, as our frontend is written using JavaScript.

3.4. Frontend

The frontend application is made using the AngularJS javascript framework which add many features to HTML to make a new generation of web applications (single-page web applications). Instead of starting the web application from scratch, we have used a template called **Angle** which offers a collection of ready to use styles and tools based on AngularJs and many javascript libraries. A More detailled description of the Angle features can be found in the next section.

4. Frontend Development

4.1. Tools

We have used the Angle template as a starting point for the development of the frontend application. This template includes AngularJS, jQuery and many other libraries. It offers also some useful predefined Angular directives and services. In order to understand the structure of this template and be able to customize it, a good understanding of AngularJS architecture and terms is essential. The most important tools used to build the frontend application are the following:

4.1.1. AngularJS

AngularJS is an open-source web application framework maintained by Google and by a community of developers and corporations to address many of the challenges encountered when developing single-page applications. It aims to simplify both the development and the testing of such applications by providing a framework for client-side web application which is able to add new reusable features to HTML.
Official Website: https://angularjs.org

4.1.2. jQuery

The usage of AngularJS does not eliminate jQuery which is the most used javascript library in web applications today. It is designed to simplify document navigation, DOM (Document Object Model) elements selection, animations, events handeling, and Ajax (Asynchronous JavaScript and XML) calls. jQuery also provides capabilities for developers to create plug-ins on top of the JavaScript library. This enables developers to create abstractions for low-level interaction and animation, advanced effects and high-level, theme-able widgets.
Official Website: https://jquery.com

4.1.3. Git

Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency. It helps to keep track of every change made on the code and rollback when needed. It is also the best way to work on group on the same project.
Official Website: https://git-scm.com

4.1.4. npm

npm is package manager for JavaScript, and is the default one for NodeJS packages. NodeJS gives the possibility to run the javascript on the server side making it possible to write javascript files and run them from the terminal.

After the release of NodeJS on 2009, developers started making reusable node modules and the **node package manager** (npm) was created on 2011 to make it easy to share and reuse node modules. But now it is used to manage all types of javascript modules and packages.
Official Website: https://www.npmjs.com

npm is used in our application to manage packages such as Bower and Gulp.

4.1.5. Bower

Bower is a package manager for frontend components such as CSS and javascript libraries. It is installed using npm as it is a node module. Bower does not replace npm. npm manages server side packages will bower manages frontend packages.
Official Website: https://www.bower.io

4.1.6. Gulp

Gulp is a javascript task runner or build tool. It can run several tasks automatically to improve the developer workflow. Tasks are written in javascript and can do various jobs from concatenating multiple files into one to compiling a Less source code (the less language is presented below) and compressing the resulting CSS code and storing it in a destination file.

Gulp uses Steams (the NodeJS objects representing a buffer of data) during the task operations. So there is no need to store intermediate results in temporary files. It also runs the tasks in parallel which is more efficient. Dependencies between tasks can be declared to force a task not to start before the end of an other one. Gulp can also watch files and run corresponding tasks when a file is changed.
Official Website: https://www.gulpjs.com

4.1.7. jade

Jade is a templating language which produces HTML code from a less verbose syntax. The Jade syntax is more easy to write, read and maintain. More than that, the Jade language has many features which make the code dynamic and reusable.
Official Website: https://www.jade-lang.com

All the HTML files in our frontend application are written using Jade and compiled by Gulp.

4.1.8. less

Less is a CSS pre-processor, meaning that it extends the CSS language, adding features that allow variables, mixins, functions and many other techniques that allow the developer to make CSS that is more maintainable, themable and extendable.

There are also many 3rd party tools that help to compile less files and watch for changes. Gulp is one of these tools.
Official Website: https://www.lesscss.org

4.2. Files structure

After modifications made to the initial template, the files structure of the application is the following:

```bash
app/                   # the build folder
master/                # the sources folder
    |-- bower_components/   # bower installed packages
    |-- jade/               # the core application views
    |-- js/                 # the core application js files
    |-- less/               # the core application styles
    |-- modules/            # modules folder
    |-- node_modules/       # NodeJs installed modules
    |-- bower.json          # bower manifest file
    |-- gulpfile.js         # Gulp tasks definition
    |-- package.json        # npm manifest file
    |-- vendor.base.json    # packages to load with the application
    |-- vendor.json         # packages that will be loaded when needed
vendor/                # the libraries folder
index.html
```

Figure 2: The files structure of the frontend application

All the application source code is inside the master directory. Under this directory we find:

`bower_components:` When we install a package using Bower, it is stored into this directory. Then when Gulp is run, it takes all needed Javascript and CSS files from this directory, minify them (by removing all optional white spaces and renaming variables and functions with short names in order to reduce the file size) and store them into the libraries directory named `vendor` .

Gulp knows which files are needed from the vendor.base.json and vendor.json. vendor.base.json lists the packages to be loaded with the core application (like Angular and jQuery). vendor.json lists other needed packages, which will be loaded specifically depending on the visited module.

`jade:` The core view files are stored into this directory. Views are written using the Jade language and compiled into HTML files by Gulp. The compiled HTML files are stored into the app/views directory.

`js:` This directory contains all the core javascript files of the application. Since AngularJS is used to build this application; we find controllers, directives and services directories inside the js one following the Angular conventions. Additional javascript classes are stored inside the js/classes folder. One of this classes is the Module class which will be used to create modules for the application.

`less:` This directory holds all application style definitions. They are written using a CSS pre-processor called Less. Gulp will compile all this files into CSS files and store them in app/css directory.

`modules:` This directory contains source code of modules. the module structure is detailed below.

`node_modules:` NodeJs modules installed using npm install will be stored into this directory.

`gulpfile.js:` This file contains the Gulp tasks definitions.

4.3. Tools installation

1. Setup NodeJS and npm: installation instructions for different systems are described in this link: https://nodejs.org/download
In my case, I am using Ubuntu 14.04 and the setup commands are:

```bash
# adding the NodeSource PPA
curl --silent --location https://deb.nodesource.com/setup_0.12 | sudo bash -
# update packages list
sudo apt-get update
# installing NodeJS (npm is installed with it automatically)
sudo apt-get install --yes nodejs
```

Figure 3: NodeJS and npm installation commands on Ubuntu

2. Setup Git: Git can be installed on Ubuntu from the default package manager:

```bash
sudo apt-get install git
```

Figure 4: Git installation command on Ubuntu

3. Setup Bower and Gulp: Bower and Gulp are installed using npm. We install them globally to be able to use their commands from any directory:

```bash
sudo npm install -g bower
sudo npm install -g gulp
```

Figure 5: Bower and Gulp installation commands

4. Setup the application dependencies: Inside the `master` directory, running the following commands will install all the needed dependencies for the application to work. The list of dependencies is read from the file `package.json` for npm and `bower.json` for Bower.

```bash
sudo npm install
bower install
```

Figure 6: Dependencies installation commands

4.4. Building and running the application

The application source code is built using Gulp to produces HTML, CSS and javascript files and store them into the `app` folder. Gulp also runs a local server into this folder so that the developer can see the resulting application in the browser. After the build process, Gulp watchs all sources files. When a source file is changed and saved, Gulp re-compile this file and refresh the application page on the browser. This way, the developer can change the source code and see the result immediately in the browser.

4.4.1. The gulpfile

The file `gulpfile.js` contains the declaration of Gulp tasks. Declaring a task in this file can be done as follows:

```javascript
var gulp = require('gulp'); // requiring the gulp module
gulp.task('copy', function(){
  gulp.src('folder1/*')
    .pipe(gulp.dest('folder2'));
});
```

Figure 7: Gulp task declaration example

In the code above, we created a task called `copy` to copy all files under the `folder1` into `folder2` . To run this task from the command-line, we type:

```bash
gulp copy
```

4.4.2. Gulp tasks

In our application, the main Gulp tasks are the following:

scripts:vendor
This task runs two sub tasks **scripts:vendor:base** and **scripts:vendor:app**. The first one minifies and concatenates all the base vendor files (read from `master/vendor.base.json`) into the file `app/js/base.js` . The second one reads the list of additional vendor files from `master/vendor.json` and copies them into the folder `vendor` .

scripts:app
This task minifies and concatenates all the core application javascript files into `app/js/app.js` .

styles:app, styles:themes and bootstrap
Those are stylings tasks. The **bootstrap** task compiles the Less source code of the bootstrap library. The two other tasks compile the core application Less files. The resulting CSS files are stored into `app/css` .

templates:app, templates:views and templates:pages
Those tasks compile the Jade source files of the index, views and pages respectively and store the resulting HTML files into `app` , `app/views` and `app/pages` respectively.

modules:scripts
This task concatenates all the javascript files of each module and stores them under `app/modules/module_name/all.js`

modules:styles
This task compiles the less file of each module and stores the result into `app/modules/module_name/style.css`

modules:views
This task compiles all jade files under the views directory of each module and stores the resulting HTML files into `app/modules/module_name/views`

watch
This task watches all source files for changes and runs the corresponding task when a file is modified.

connect
This task runs a local webserver into the app directory to see the results in the browser.

start
This task compiles the application without vendors (by running the other tasks) and starts the server by running the `connect` task.

default
This task compiles the application and vendors and starts the server by running the `connect` task. This task is run automatically when we execute the command "gulp" in the master directory.

4.5. Basics of AngularJS

AngularJS is a very powerful framework for client-side applications. In this section, I will try to cover the minimum required basics in order to understand how the frontend application is built and be able to add new modules to it.

4.5.1. Starting example

A very simple example of using AngularJS could be the following:

```markup
<!doctype html>
<html>
<head>
  <meta charset="utf-8">
  <title> Angular Sample </title>
</head>
<body data-ng-app="demo">
  <p> Your name : <input type="text" ng-model="name"></p>
  <p> Hello {{ name }} </p>

  <script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js
  </script>
  <script>
    var app = angular.module('demo', []);
  </script>
</body>
</html>
```

Figure 8: Starter example of using AngularJS

In the code above, we included the AngularJS javascript file from Google and we defined a new angular module called "demo". Please note that we put the scripts at the end of the body element to speed up the page loading. The attribute `data-ng-app` of the body element tells AngularJs which angular module to be applied to the page.

We defined a text input and set the attribute `ng-model` to "name". This will define a variable called "name" and bind it's value with the content of the input. So if we type some text in this input, the value of the variable name will be set to this text. Finally we show the value of the variable name using the syntax `{{name}}`. This example shows the data binding feature of AngularJS, so by changing the value of the input, the text in the next paragraph is changed too.

In the following, you find short definitions and examples of the most important AngularJS terms.

4.5.2. Angular Module

A module is a reusable collection of the application parts (controllers, services, directives, ...). An application can use one or multiple modules. When declaring a module, we can specify the list of dependent module that this module will use. The parts of the related modules could be injected by AngularJS automatically and used.

Example

```javascript
// declaration of new angular module
var app = angular.module('myApp', ['module1', 'module2']);
```

Figure 9: AngularJS module declaration example

Here we declared a module called "myApp" which depends on "module1" and "module2".

4.5.3. Controller and Scope

A controller is a javascript function that handles a view or a part of the page. It can bind variables and events to this part of the page. The variables binded to a view are holded into an object called the scope. This object can be injected automatically into the controller simply by adding `$scope` to the function arguments.

Example

```javascript
// using the app variable which is the module
app.controller('HomeController', function($scope){
  // use the scope to define binded variables
  // A name variable for example
  $scope.name = 'LHCbPR';
  // or an array
  $scope.tools = ['AngularJS', 'Bower', 'Gulp'];
});
```

Figure 10: AngularJS Controller example

The code above will not work after minification. A recommanded way is to declare dependencies into a list of strings and add them to the function arguments in the same order like this:

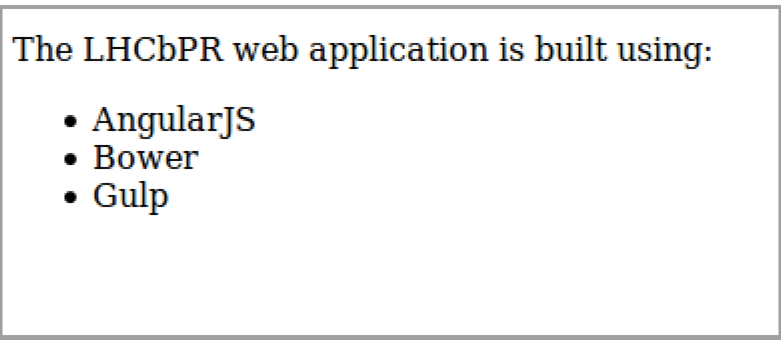
```javascript
app.controller('HomeController', ['$scope', function($scope) {
  // ...
}]);
```

To use the controller in the view, we first declare it using `ng-controller` then use the variables and methods of the scope inside this view like the following:

```markup
<div ng-controller="HomeController">
  <p> The {{name}} web application is built using: </p>
  <ul>
    <li ng-repeat="element in tools"> {{ element }} </li>
  </ul>
</div>
```

Figure 11: Using Angular Controller in the view

The attribute `ng-repeat` we added to the `<li>` element is a predefined angular directive which loops over a array and repeat the element for each item of the array. The resulting view will be something like:


The LHCbPR web application is built using:

- AngularJS
- Bower
- Gulp

Figure 12: Sample View

4.5.4. Directives

A directive is a new element or behavior added to the HTML. `ng-controller` and `ng-repeat` are examples of directives which are used as attributes. When defining a directive we specify wether it will be used as an element (an HTML tag), an attribute or a class. This choice depends on the feature added by the directive:

A directive which is replaced by a template or view is more likely to be used as an element.

A directive which adds a behavior to HTML elements based on an input is more likely to be used as an attribute.

A directive which adds a generic decoration or behavior to HTML elements is more likely to be used as a class.

These rules remain subjective as AngularJS enables the developer to give multiple uses to the same directive (a directive that can be used as an element and as an attribute for example).

For example, let us define a directive that takes a collection of objects and shows them as an HTML table. First we will define the template as follows (note that we are using some predefined CSS classes):

```markup
<table class="table table-striped table-bordered">
  <thead>
    <tr>
      <!-- We assume that the list of titles will be declared -->
      <th ng-repeat="title in titles"> {{ title }} </th>
    </tr>
  </thead>
  <tbody>
    <!-- We assume that data is the collection of objects -->
    <tr ng-repeat="item in data">
      <td ng-repeat="title in titles"> {{ item[title] }} </td>
    </tr>
  </tbody>
</table>
```

Figure 13: Data table sample directive template

Then we define the directive as follows:

```javascript
app.directive('myTable', function(){
  // The directive consists on an object that this function should return
  return {
    restrict: 'E', // define how the directive will be used
    // 'E' for element, 'A' for attribute and 'C' for class
    // 'EA' means element or attribute
    templateUrl: 'data-table.html', // the template file
    // if specified, the content of the HTML element to which the directive
    // is applied will be overitten by this template
    scope: { // contains the data passed to the directive as attributes
      data: '='
    },
    // This function is executed when the directive is rendered
    link: function(scope){
      // scope.data will refer the data passed to the directive
      // Defining titles of the table as the properties of the first
      // object in data, as we assume all objects have the same properties
      scope.titles = [];
      if(scope.data != undefined && scope.data.length > 0){
        for(attr in scope.data[0]){
          scope.titles.push(attr);
        }
      }
    }
  };
});
```

Figure 14: Data table sample directive code

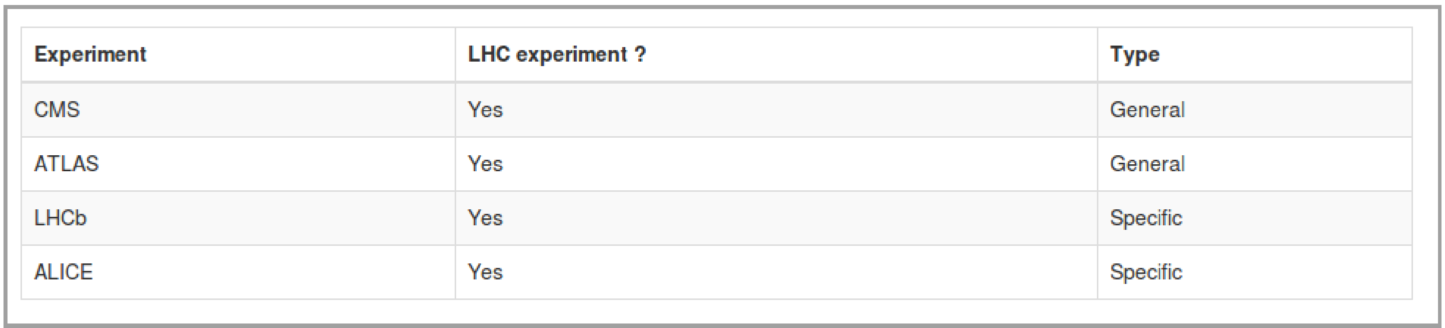
Now assuming that we have a collection of objects in the scope like this:

```javascript
$scope.collection = [
  { Experiment: 'CMS', 'LHC experiment ?': 'Yes', Type: 'General' },
  { Experiment: 'ATLAS', 'LHC experiment ?': 'Yes', Type: 'General' },
  { Experiment: 'LHCb', 'LHC experiment ?': 'Yes', Type: 'Specific' },
  { Experiment: 'ALICE', 'LHC experiment ?': 'Yes', Type: 'Specific' }
];
```

We can use the directive in the view (the HTML code) as follows:

```markup
<my-table data="collection"></my-table>
```

The result is a table of the given data:



Experiment	LHC experiment ?	Type
CMS	Yes	General
ATLAS	Yes	General
LHCb	Yes	Specific
ALICE	Yes	Specific

Figure 15: Data table sample directive result

4.5.5. Services

A service is an object which is instanciated once (Signleton) and injected into components using it (like controllers and directives). Service is a good way to share functionalities between different components of the application.

AngularJS provides several predefined services like `$http` which handles AJAX requests. To use this service inside a controller for example, all we have to do is to add it to the arguments:

```javascript
app.controller('MyController', ['$scope', '$http', function($scope, $http){
  // Now $http can be used here
}]);
```

Figure 16: Example of injecting $http service

In our application, we are using a service called **Restangular**. It is built on top of `$http` and simplifies the communication with a REST API.

4.6. Frontend core application features

4.6.1. Routes and Lazy loading

4.6.2. LHCbPR service

4.6.3. Directives

4.6.3.1. DataTables and be-responsive

4.6.3.2. Charts

4.6.3.3. Search and Select Jobs

5. Application Modules Development

5.1. Helper classes

5.1.1. Module class

5.1.2. Dependencies class

5.1.3. Colors class

5.2. Adding a module

From the guide

5.3. Analysis Modules Done

5.3.1. Jobs Module

5.3.2. Trends Module

5.3.3. Histograms Module

Conclusion

My final internship at CERN is the best job experience I had till now. During this period, I discovered new cultures by meeting people from different countries, I worked within a group in a new flexible way and I learned new web development technologies.

This internship was part of my third year studies at ISIMA. I worked as a software engenieer to develop and improve the LHCbPR (LCHb Performance and Regression) application. The applications used to analyse data collected by the detector on LHCb are tested using configurable test jobs. The main goal of LHCbPR is to provide physicists and developers with a framework in which they can easily do analysis of test jobs results. We started the development of the second version of this application from scratch and used flexible architecture and recent technologies. This version contains three layers: A database, a REST API and a frentend web application. My task was mainly to develop the frontend application and the analysis modules using AngularJS and new web development tools. But I did also some changes in the backend which was built using Django REST framework (written with Python programming language).

I started the application based on a template containing AngularJS and other libraries. After understanding the structure of the template I started customizing it to fit our needs. Modifications have been done on the files structure, the build system and the javascript code. The next step was to create some analysis modules and to add new helper classes which simplify this process. Finally I have written a development guide for users explaining how to add new analysis modules to the application.

The first version of the frontend application was done with three analysis modules. Other analysis modules should be added to have to same functionalities as the old version. One perspective was to simplify adding module for persons with no AngularJS knowledge by giving an easy to use group of functions hiding the complexity of this framework. This simplification layer could be made in a generic way giving the possibility to use it to build other modular web applications.