MTD Data Acquisition Graphical User Interface

Abdulrahman Almarzouqi
Supervisors: Mehmet Ozgur Sahin, Giulia Sorrentino
CERN, Geneva, Switzerland

Abstract

This report presents the Summer Student project, which was part of the CMS MTD (MIP Timing Detector) project. The project focused on developing a Graphical User Interface (GUI) for the MTD Data Acquisition (DAQ) that connects to the xDAQ server (RESTful APIs) that are used to perform different functionalities on the DAQ. Additionally, the GUI had to be designed to be suitable for its purpose and the target users – physicists. An introduction of the Vue JavaScript Framework, which was used to develop the GUI, will be given, followed by a demonstration of the framework features and how it enhances scalability. Moreover, the two main two pages built for the GUI, MTD Command and MTD Configuration, will be reviewed. Lastly, a brief evaluation of the project will be provided.

Keywords

MTD GUI, Vue JavaScript Framework, Scalable, MTD Command, MTD Configuration, RESTful API, xDAQ

Contents

List of Figures

1. Introduction of MTD GUI Project & Data Journey:

The project that this report discusses is involved in the CMS (Compact Muon Solenoid) experiment. The CMS experiment investigates a wide range of physics, including the search for the Higgs boson, extra dimensions, and particles that could make up dark matter [1].

In particular, the project is part of the MIP (Minimum Ionizing particle) Timing Detector Project (MTD) which is a device that will be added to the CMS detector as part of the CMS Phase-2 Upgrade following the High Luminosity upgrade of the Large Hadron Collider. The MTD will be able to precisely measure ‘the production time of minimum ionizing particles (MIP) and help in differentiating the nearly 200 simultaneous "pileup" interactions that occur in each bunch crossing of the LHC. Additionally, it will provide enhanced capabilities for identifying charged hadrons and searching for long-lived particles’ [2].

The data journey of the MTD starts from the detector’s read-out electronics. Then the data passed to the LpGBTs (Low power Gigabit Transceivers). After that, the data goes to serenity, then to the xDAQ which is the server with the RESTful APIs that perform various tasks, and Lastly to the Graphical User Interface (GUI). The GUI has a significant role which includes displaying the chip configurations, running commands on the serenity, and analyzing the data with a UI that is user-friendly. By employing the latest solutions for building a GUI using JavaScript Frameworks, a scalable and flexible GUI can be created that ease the process of upgrading or modifying the built GUI. The report discusses the project worked on during the summer student program which is the process of developing a modern scalable GUI for the MTD DAQ that connects and communicates to the xDAQ and displays the responses using a JavaScript Framework called Vue.js.

2. Vue JavaScript Framework:

The chosen JavaScript Framework was Vue.js for its beginner-friendly learning journey which is suitable for the nature of MTD project as different contributors get involved over time. Vue provides different features that make it possible to build scalable GUI. A few of these features are declarative rending, reactivity, directives, and component architecture.

The following is a simple dropdown form that will demonstrate the different features used in the development of a scalable GUI.

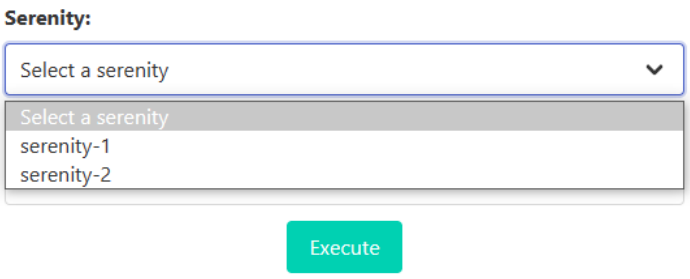


Figure 1: Serenity dropdown field

2.1. Declarative rending

Declarative rendering allows direct manipulation of the DOM, unlike imperative rending. The following are shows the difference, where {{ }} syntax used to display the content of option, instead of the tedious imperative rending.

Framework

```
<form>
  <label>Select a Serenity</label>
  <select v-model="mtd_request.serenity">
    <option
      v-for="option in serenities"
      :value="option"
      :key="option"
      :selected="option === mtd.serenity"
    >{{ option }}</option>
  </select>
  <button type="submit">Submit</button>
</form>
```

No Framework (Imperative Rendering)

```
<form>
  <label>Select a Serenity</label>
  <select id="serenity-select"></select>
  <button id="submit-button">Submit</button>
</form>
```

```
<script>
  const mtd_request = {
    serenity: ''
  };

  const serenities = ['serenity-1', 'serenity-2'];

  const select = document.getElementById('serenity-select');
  serenities.forEach((option) => {
    const optionElement = document.createElement('option');
    optionElement.value = option;
    optionElement.text = option;
    if (option === mtd_request.serenity) {
      optionElement.selected = true;
    }
    select.appendChild(optionElement);
  });

  const submitButton = document.getElementById('submit-button');
  submitButton.addEventListener('click', (event) => {
    event.preventDefault();
    // Handle form submission
  });
</script>
```

Figure 2: Declarative rending and imperative rending

2.2. Reactivity

One of Vue features is that it allows variables to be reactive, meaning changes that happen to the variables can be seen in real-time. A demonstration can be seen in the following picture:

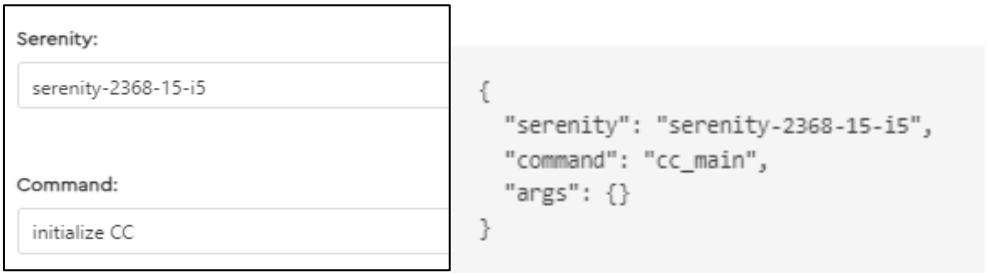


Figure 3: Reactivity demo*nstration*

2.3. Directives

Directives are built-in functions in the framework that can be used in HTML tags, which is used

instead of the long tedious generic functions that execute the same actions. Comparison can be seen in the following codes, where v-for directive is used instead of generic function:

Framework

```
<form>
  <label class="label">Serenity:</label>
    <select v-model="mtd_request.serenity">
      <option
        v-for="option in serenities"
        :value="option"
        :key="option"
        :selected="option === mtd_request.serenity"
      >{{ option }}</option>
    </select>
  </div>
</form>
```

No Framework (Generic functions)

```
<form>
  <label class="label">Serenity:</label>
  <select id="serenity-select"></select>
</form>
```

```
const mtd_request = {serenity: ''};
const serenities = ['serenity-1', 'serenity-2'];
const select = document.getElementById('serenity-select');
serenities.forEach((option) => {
  const optionElement = document.createElement('option');
  optionElement.value = option;
  optionElement.text = option;
  if (option === mtd_request.serenity) {
    optionElement.selected = true;
  }
  select.appendChild(optionElement);
});
```

Figure 4: *A v-for directive and generic function*

2.4. Component architecture

Component is a set of code that can be referred to by a single name, hence, providing abstraction. Components can be used as well more than once in other components with different values passed as props achieving reusability.

Component

```
<form>
  <BaseSelect
    :options="serenities"
    v-model="serenity"
    label="Select a Serenity"
  />
  <button type="submit">Submit</button>
</form>
```

Non-Component

```
<form>
  <label>Select a Serenity</label>
  <select v-model="serenity">
    <option
      v-for="option in serenities"
      :value="option"
      :key="option"
      :selected="option === serenity"
    >{{ option }}</option>
  </select>
  <button type="submit">Submit</button>
</form>
```

Figure 5: *component and non-component architecture*

3. Main Pages/Components Created

Vue project is mainly made of components, but the Vue components that hold the input and output components and are routed to the navigation bar will be called pages. The two main pages that are developed are MTD Command and MTD Configuration.

3.1. MTD Command page:

The MTD Command page is used for running commands, with or without arguments, on selected serenity. For the BTL (Barrel Timing Layer), the page is called MTD BTL Command. It consists of two main components: the input component is a form where is the serenity and the command - to executed on the serenity - are selected, and the output component where the result of the execution is displayed.

3.1.1. Input Component

The input component of the MTD BTL Commands looks in the GUI as the following:

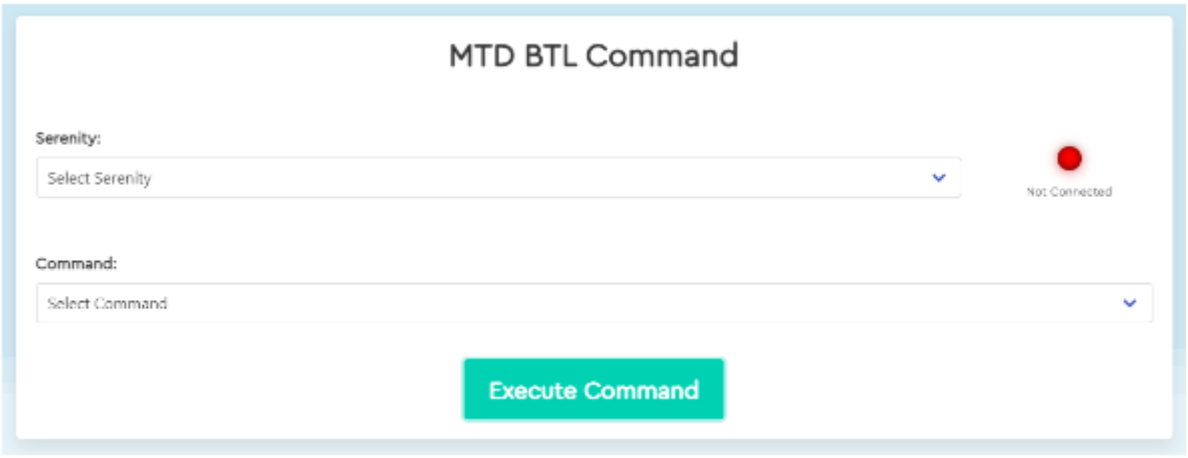


Figure 6: *Input Component of MTD BTL Commands*

The input component consists of two main dropdown fields, Serenity and Command, and one or more secondary fields that depends on whether this is an argument for the selected Command and the type of argument (input or dropdown).

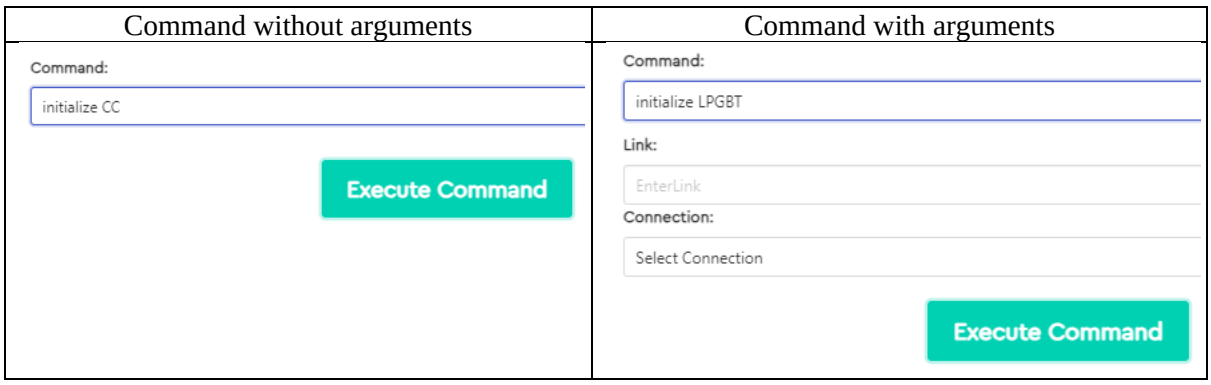


Figure 7: *Command arguments fields*

The input component uses "BaseSelect" and "BaseInput" reusable components to make a simple dropdown field and input field for Serenity field and Command arguments fields. However, reusable component has not been used on the command field due to its complexity. The dropdown fields are populated using data fetched using get RESTful API either from the database or JSON files (Appendix A).

Once required fields are filled and the “Execute Command” button (shown in Figure 6) have been clicked, a RESTful API will send a request with the selected values in the input component. The RESTful API response will be displayed in the output component.

3.1.2. Output Component

The output component of the MTD BTL Command looks in the GUI as the following:

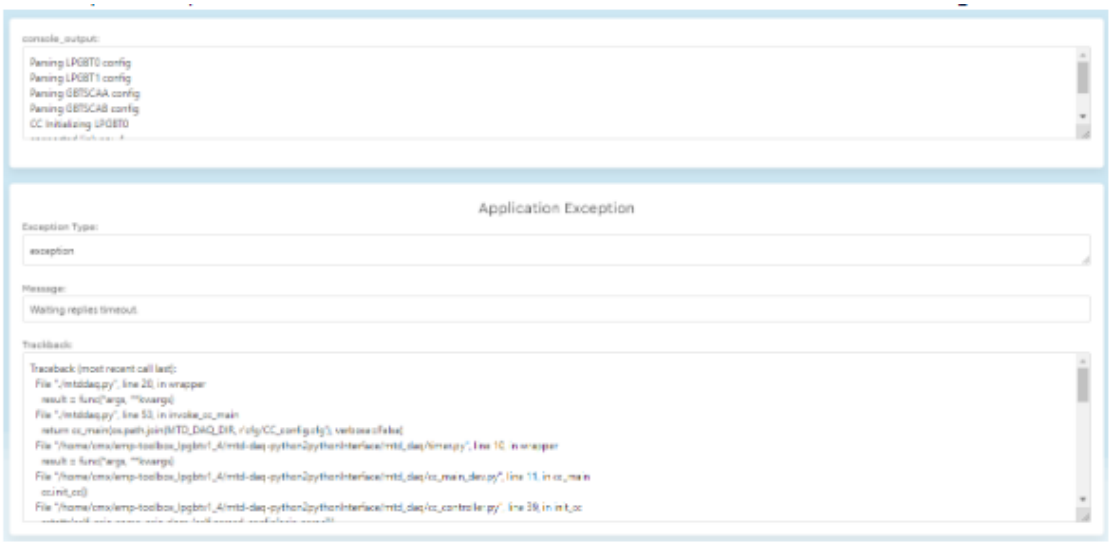


Figure 8: *Output Component of MTD BTL Command*

The output component will display the console_output and return_value of the RESTful API response if the command execution process was successful, otherwise it will display the console_output and Application Exception if it failed.

3.2. MTD Configuration page:

The MTD Configuration page is used for displaying the configuration of the selected chip. For the BTL, the page is called “MTD BTL Configuration”. It consists of two main components: the input component is a form where the Serenity, CC, Chipclass, and Chip are selected, and the output component where the selected Chip configuration data is presented in a table.

3.2.1. Input Component

The input component of the MTD BTL Configuration looks in the GUI as the following:

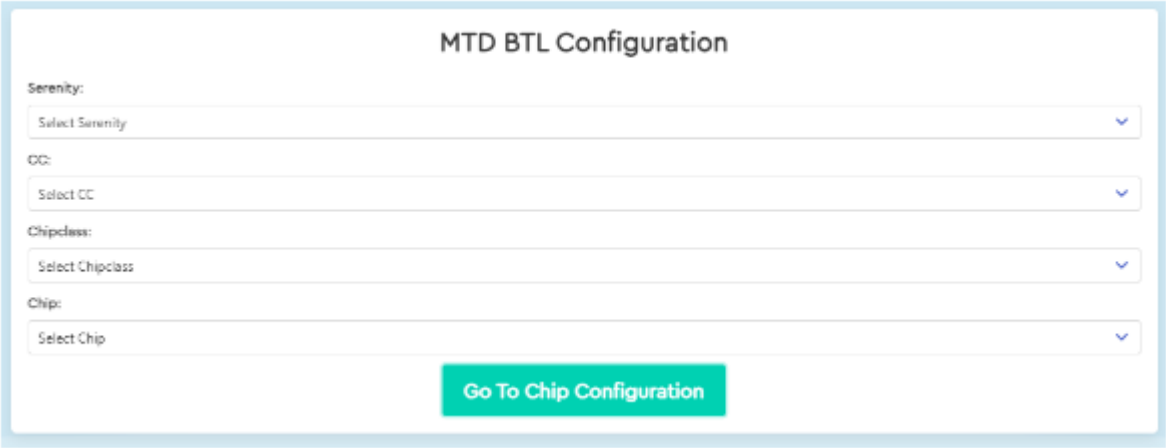


Figure 9: *Input Component of MTD BTL Configuration*

The input component consists of four dropdown fields. The four dropdown fields are for serenities, CCs, Chipclasses, and Chips available where they are populated by get RESTful API requests (Appendix A) that fetches the data from the database. The input component uses "BaseSelect" reusable component to make a simple dropdown field for all the four fields.

The two dropdown fields Serenity and Chipclass show the options when the input component is created, while the two dropdown fields CC and Chip depend on the other dropdown fields' selected option. The "CC" field depends on "Serenity" field and the "Chip" field depends on "CC" and "Chipclass" fields.

There are Vue Watchers (Appendix B) that listen for any changes that happen to the dropdown fields (in this case, the option selection action is the change) and pass the selected value. This was used for the "CC" field and the "Chip" field for listening to the fields it depends on and passing the selected values to the Load methods for "CC" and "Chip" that will populate the appropriate options.

3.2.2. Output Component

The output component of the MTD BTL Configuration looks in the GUI as the following:

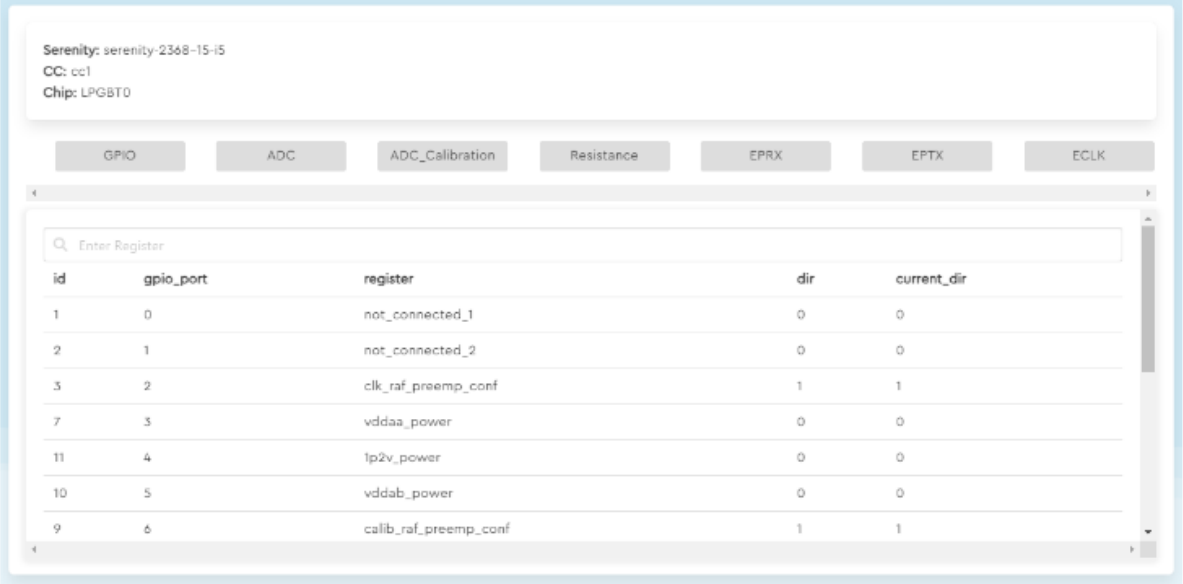


Figure 10: *Output Component of MTD BTL Configuration* *Figure 10:*

The output component will use a JSON file that has the table names and their fields to display tables and their headers for the columns. Moreover, the data of the selected chip - from the input component - fetched from the database and filled in the appropriate tables. The tables have a search field above them (as can be seen from Figure 10) that would filter the data according to whatever is written in it.

4. Conclusion and Evaluation

Overall, the main goals of the project for the period of 8 weeks, have been successfully completed. The MTD Command and MTD Configuration have been built for the BTL and templates for other pages (ETL and Advanced) have been created for future development and improvement on the project. Since this is an ongoing project worked on by different contributors, I had the privilege to write the start of the documentation of this project. The documentation has been successfully written in detail and revised by the team members.

Challenges have been faced throughout the project, including the learning process of Vue JavaScript Framework. Additionally, many struggles have been encountered while trying to connect the MTD Restful APIs with the components in the MTD GUI to achieve the required functionalities. Nevertheless, these challenges were resolved through teamwork.

Acknowledgement

Firstly, I would like to express my gratitude to the ones who have supported me throughout my journey in pursuing unique opportunities. I would also like to extend my appreciation to the CMS MTD Team for their valuable guidance. I am truly thankful to my supervisor, Mehmet Ozgur Sahin, and co-supervisor, Giulia Sorrentino, whose mentorship was a significant factor in achieving the project goals. A special appreciation to the University of Bahrain for supporting me in taking this

opportunity. It has been an honor to work with such skilled professionals, and grateful for such an opportunity that helped me obtain and apply new skills to the CMS MTD project.

Bibliography

[1] “CERN accelerating science,” CERN, http://public-archive.web.cern.ch/en/LHC/CMS-en.html.

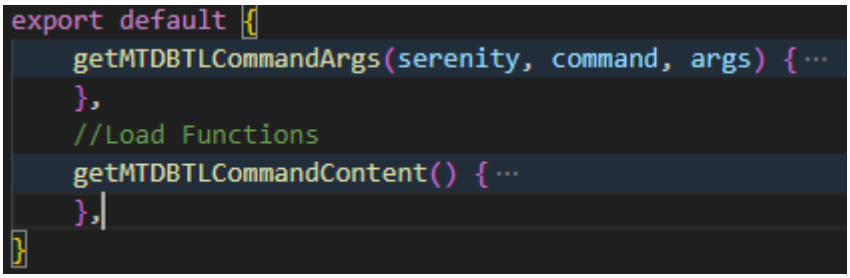
[2] CERN, https://cds.cern.ch/record/2667167/files/CMS-TDR-020.pdf

Appendix

A. MTD Services

A.1. MTD BTL Command Services

The following is the RESTful API used in the MTD BTL Command page:



A.2 MTD BTL Configuration Services

The following is the RESTful API used in the MTD BTL Command and Configuration page:

```
export default {
    // LoadAPIS
    getMTDBTLConfigContent() {…
    },
    getMTDSerenities() {…
    },
    getMTDCCs(serenity_board) {…
    },
    getMTDChipclasses() {…
    },
    getMTDChips(cc, chip_class) {…
    },
    //POST RESTFULL APIs
    postMTDBTLConfigDB(payload) {…
    }
}
```

B. Vue Watcher for MTD BTL Configuration

The following are the Vue Watchers that used for to listen to any changes to “Serenity”, “CC”, and “Chipclass” fields which is used in the MTD BTL Configuration page:

```
watch: {
  'mtd_request.serenity'(newSerenity) {…
  },
  'mtd_request.cc'(newCC) {…
  },
  'mtd_request.chipclass'(newChipclass) {…
  },
},
```