Abstract

CERN Beams Department maintains a website with various tools for the Operations Group, with one of them being specific for the Proton Irradiation Facility (IRRAD). The IRRAD team use the tool to follow up and optimize the operation of the facility. The original version of the tool was difficult to maintain and adding new features to the page was challenging. Thus this summer student project is aimed to upgrade the web page by rewriting the web page with maintainability and flexibility in mind. The new application uses a server-client architecture with a REST API on the back end which is used by the front end to request data for visualization. PHP is used on the back end to implement the API’s and Swagger is used to document them. Vue, Semantic UI, Webpack, Node and ECMAScript 5 is used on the fronted to visualize and administrate the data. The result is a new IRRAD operations web application with extended functionality, improved structure and an improved user interface. It includes a new Status Panel page that has improved the IRRAD operators’ efficiency and quality of service.

1 Introduction

CERN Beams Department maintains a website with various tools for the Operations Group, located at http://op-webtools.web.cern.ch. These tools help operators in their daily task of operating facilities by providing services like status of accelerators or other facilities, planning tools and contact lists.

One of these services is specific for the Proton Irradiation Facility (IRRAD). IRRAD is used to perform irradiation experiments on particle detectors, electronic components and materials [1]. They get the proton beam from the Proton Synchrotron (PS), which is traversing three zones inside the facility, before it continues to the CHARM facility.

There are four fixed beam profile monitors (BPM) installed through the trajectory of the irradiation beam. The beam passes through the BPM’s and they measure the intensity profile
of the beam. Each zone contains three tables where users can install samples with which the proton beam interacts during the irradiation experiments. Each of the tables have installed a mini BPM setup with one central mini BPM in the front, one central in the back, one single pad BPM on the left and one on the right. These devices allow for monitoring which position is receiving the beam and the profile of the beam they receive. The intensity of the beam is also being monitored by a secondary emission counter (SEC) that measures the number of protons per hour. To monitor the environmental conditions of the facility, there are installed temperature sensors on some tables, in the chiller units used for low temperature irradiation experiments, and in the storage areas. The data from these sensors are collected from data acquisition modules (DAQ) to be stored in the database.

The IRRAD team use the IRRAD OP Webtools page (located at http://op-webtools.web.cern.ch/irrad) to follow up and optimize the operation of the facility. This page provides visualizations of the beam profile, graphs of beam intensities over time, and tools to control the visualizations.

The original IRRAD OP Webtools pages were developed in 2012 in an ad-hoc fashion, that made it difficult to maintain and add new features to the page. This summer student project is aimed to upgrade this web page by rewriting the web page with maintainability and flexibility in mind. The new page would separate front end and back end by implementing a RESTful API [2], and creating an independent front end that uses the API. In addition, the IRRAD operators requested a new Status Panel page to use in the facility and offices. This page would let them quickly see if the facility is operating properly.

2 Technologies

The back end is implemented in PHP using no framework, but with custom classes for creating a RESTful API and sending HTTP messages. Other languages with frameworks were considered, but PHP was chosen because the other options were difficult to run on CERN’s web servers.

The API’s implemented in the back end are documented with inline Swagger annotations in the source code. Swagger is a framework for designing, documenting, testing and deploying API’s using the OpenAPI specification [3], and the annotations are parsed by swagger-php to generate the documentation. It is then displayed in a web browser with Swagger UI, which allows for live testing of the API.

The front end uses multiple tools, but most important is the Vue framework, which is “a progressive framework for building user interfaces” [4]. It is comparable to other web frameworks such as AngularJS, Angular, React and Ember, but it is lighter, focusing on the view layer instead of the complete application architecture.

To provide some functionality present in some other frameworks, but missing from Vue, vue-router is used for routing between different sub-pages, and Vuex is used for managing state and synchronizing with the back end.

Semantic UI is used for styling the page. Bootstrap 3 was considered, but using Semantic UI let us test a new framework rather than re-using an old framework. Bootstrap 4 was also considered, but because it was in alpha version it could be too unstable and updates could include breaking changes.

For programming the web page, ECMAScript 2016 is used with the Babel transpiler to transpile it to ECMAScript 5 compatible code. This new ECMAScript standard provides new features to JavaScript that makes the development easier, while keeping backwards compatibility with ECMAScript 5.

Webpack is used for building the project into files that can be served from a web server and
displayed in a web browser. It is also used for development combined with hot module reloading (HMR) to synchronize changes in code with the displayed web page immediately. Together with multiple Plugins and Loaders, Webpack can optimize the source code by minifying it, only including necessary modules and inlining assets such as images. It also enables the usage of the ECMAScript `import` statement to import modules from packages installed with Node Package Manager (NPM).

Node.js is used to run Webpack, and NPM is used to manage the project dependencies, such as Vue, Semantic UI and Webpack. Finally, Gulp is used to initiate the build scripts and deploy both the back end and front end of the project.

3 System Architecture

The system uses a client–server architecture with a layered architecture inside each part, as seen in Fig. 1 where “Front end” is the client and “Back end” is the server. In the source code, these are two sub directories of the root directory: `frontend` and `backend`.

The root directory also contains files to deploy the application, with `deploy.sh` being the main entry point for deploying and providing functionality to deploy front end and back end individually to either testing or production. `deploy.sh` runs `gulp` to perform the deployment process, where `gulpfile.js` describes this process. `package.json` describes the overall project and the dependencies of the deployment process.

3.1 IRRAD REST API

The back end consists of two layers: the IRRAD REST API and the database. The IRRAD REST API is the remote interface used to access and modify data stored in the database. It provides structured and flexible API that can be used by web page and other services, applications or users that need access to the data. By creating the API, the database tables could be restructured and rationalized.

It is designed in a resource-oriented way where URI’s define location of resources, and HTTP verbs are used to access or modify them. It mimics the way file system directories work by providing collections of resources (directories) and the resources themselves (files) based on the URI path. For example, `/bpm` will provide the list of all BPM resources, and `/bpm/BPM_01` provides one specific resource that can be found in the list.

Resources may contain links to related resources. These links create a web of resources that can be traversed, and it avoids applications breaking because of hard coded links being changed in the API. The links are provided as relative URI’s in a `href` property. For example, `/bpm/BPM_01` may contain the property `measurements.href` defined as the string `/bpm/BPM_01/measurement` that points to the collection of measurements for that BPM.

The HTTP verbs define how you interact with the resources. `GET` on a collection will retrieve the list of resources, while on a resource it will provide the resource itself. `DELETE` on a resource will remove the resource from the collection. `POST` on a collection will add a new resource to that collection. `PUT` on a resource will replace that resource with a new. `PATCH` on a resource will update the resource with new parameters.
3.2 API Endpoints

The API contains six access points specific to IRRAD — \textit{bpm}, \textit{sec}, \textit{table\_bpm}, \textit{daq}, \textit{area\_status} and \textit{vistar\_info} — and one general access point: \textit{users}. \textit{bpm} provides access to BPM descriptions and data. \textit{/bpm} describes the different BPM’s, \textit{/bpm/\{bpm\_id\}/reference} provides the reference graphs used for the BPM, \textit{/bpm/\{bpm\_id\}/map} describes the mapping of BPM channels to X & Y positions, and \textit{/bpm/\{bpm\_id\}/measurement} provides all the measurements from a BPM.

\textit{sec} provides access to SEC descriptions and data. SEC’s describe various intensities of the beam and get the actual data from the BPM’s. It’s structure is identical to \textit{bpm}, but without references and map, as these are not applicable to SEC’s.

\textit{table\_bpm} provides access to specific BPM sensors installed on the tables inside the facility. These have a different structure than the other BPM’s, and therefore need a separate access point. Otherwise the API structure is identical to \textit{bpm}, but without the references. Only \texttt{GET} methods have been implemented for this, so the resources cannot be modified.

\textit{daq} provides access to DAQ modules installed in the facility. These provides information such as temperatures and table positions. The API structure is similar to \textit{bpm}, but each DAQ module is sub divided into channels that can be accessed directly, and it does not have references or maps. All \texttt{GET} methods have been implemented, but not all of the other methods.

\textit{area\_status} provides the current status of the IRRAD facility area, indicating if it is in beam mode, flushing air, access closed, restricted access, general access or unknown.

Finally, \textit{users} provides endpoints for checking the currently logged in user, and checking if the user is a member of specified e-groups. For detailed documentation about the API endpoints and an ability to test them, visit \url{https://op-webtools.web.cern.ch/irrad/docs}.

3.3 Front End

The front end structure is influenced by the architecture of Vue. It is a four-layered architecture consisting of App, Router, Components and Store. Its main entry point is \texttt{index.html}, which contains the basic HTML needed and a \texttt{<div id=“app”>}/</div>} element which will be replaced with the App component by Vue from the \texttt{src/main.js} script. The App component is the root of the Application and controls the Router and Store.

From the App component, the different pages are shown based on the Router and its configuration. The Router is provided by the \texttt{vue-router} package, and changes Vue components based on the URL in the browser. All of the pages are provided as Vue components inside the \texttt{src/pages} directory. These pages use different components found in \texttt{src/components} that provide functionality that can be configured and used on multiple pages.

The Store is a plugin provided by the \texttt{vuex} package and provides centralized state management. It is used to fetch data from the IRRAD REST API, store it, share it between components, mutate it and push it back to the API. This allows multiple components to render different views based on the same data, and sharing state changes between themselves.

The Store is split into five modules that directly correspond the the API access points: \textit{bpm}, \textit{sec}, \textit{table\_bpm}, \textit{daq} and \textit{area\_status}. Each of them are independent and talk with the back end through the IRRAD REST API.

To prevent unauthorized users from using the control panel pages, it checks the access rights by using the \textit{users} endpoint of the API. In case the user is not logged in, it will redirect the browser to the \texttt{login.php} script located on the back end. This is simple utility script that will trigger a login with the CERN Single Sign-on service (SSO) and redirect back to the origin page when the user is logged in. This is made possible by protecting the \texttt{login} directory where the script is located with permissions, which will trigger the SSO.
4 Application

The final application is composed of an index page, a set of sub pages, and a control panel page with its own set of sub pages. Some pages are recreations of the original version, while some are created as new pages. All pages have been designed mainly for desktop form factors, but they are adaptable to other form factors, so they can be used on mobile devices. The Beam Profile Monitor page shows the live profiles of the available BPM’s, as a Gaussian vertical profile, a Gaussian horizontal profile, a 2D profile, and a longitudinal profile. There is an ability to switch between the different BPM’s and replay old profiles. The Intensities page shows various intensities from the BPM’s and the SEC plotted over time. The All BPM Pads page displays the 2D profile of all the four BPM’s.

The previous pages are all recreations of their original versions, while the Status Panel is a new page created for operators of IRRAD to see the current status of the facility. This includes information about the area status, profile of the beam, positions of tables, temperatures of storage areas, temperatures of chillers and boxes, and beam intensities over time.

The page has been designed to emphasize the important information, such as the profile of BPM 1, and give feedback about the status of a system in form of different colors. Green colors generally indicate that the status is good, red indicates that it is bad, and grey is unknown.

The BPM profiles are similar to the ones in the other pages, but show all of the four BPM’s instead of one BPM over three spills. The intensity graph is the same as in the Intensity page, but only contains one intensity plot and the SEC plot. Storage area and chiller status show if the temperatures are correct or if they are outside defined limits. In addition the actual temperature is displayed so that the operator may reason about a red colored indicator. The status of the IRRAD area can be seen in the navigation bar, with a textual status and a corresponding color.
Table status is show as a rectangular box with three squares, where each square represents a BPM sensor installed on the table. The left square is the left BPM pad, the right the right BPM pad, and the center a combination of the central front and back mini BPM’s. When the facility receives a beam, as indicated by BPM 1, and one of the BPM sensors registers the beam, the corresponding square light up in green to indicate that the table is receiving beam at that position. If there is beam, but none of the three sensors registers it, all squares will be red to indicate that the table is out of beam. If there is no beam, the squares will be grey. Additionally, the table BPM can be disabled (e.g. when physically removed from table), which is indicated as an empty rectangle.

CCC Vistar Panel shows the BPM Gaussian profiles, longitudinal profile, intensity graph and communications from the IRRAD administrators to the CCC operators. It is a simplified version of the Status Panel, displaying only the information required for the operators in the Cern Control Center.

To control the visualization of the pages, there is a Control Panel only available to the IRRAD administrators. It contains sub pages to configure the BPM descriptions, BPM references, BPM map, SEC descriptions and the info shown in the Vistar Panel. The BPM and SEC description pages automatically save changes made to prevent users accidentally forgetting to save. The other pages have a manual save button as to let users save when they are happy with all of the changes.

5 Conclusion

The result of this project is a new IRRAD operations web application with extended functionality, improved structure and an improved user interface. Providing a structured REST API and front
end for the IRRAD facility will make maintenance and future improvements easier to implement. The API can also enable new applications to be implemented faster by providing an intuitive, programmable interface to retrieve the data rather than writing SQL queries directly. It may enable operators or users to retrieve specific data through the API documentation interface or programatically. Building the front end on the Vue frameworks and using components for common functionality enables re-usability and less maintenance.

The new Status Panel page has improved the IRRAD operators’ efficiency and quality of service, by giving them an overview of all important information regarding the operation of the facility. BPM profiles let them evaluate if the beam is received properly. The intensity graph lets them see if the amount of spill changes, or if the beam has been offline during the last six hours. The table display lets them see if tables are positioned properly, and temperature displays enable the monitoring of freezers and chillers. Finally, the area status display allow them to verify that the area is in the proper state for the weekly access, even on mobile devices.

References


