# BANASTHALI VIDYAPITH

**INTERNSHIP REPORT 2023- 24**

**A blue and white logo

AI-generated content may be incorrect.**

*A report submitted in partial fulfilment of the requirements for the Award of Degree of*

#### BACHELOR OF TECHNOLOGY

**in**

***Computer Science Engineering***

#### by

**Unnati Saini**

**2216433**

**Under the Supervision of**

**Smt.Bhavna N. Merh**

**SOG , ACSD**

**Raja Ramanna Centre for Advanced**

**Technology (RRCAT) , Indore**

**(Duration: 15th May 2025 to 15th July 2025)**

**2022 – 2023**

**DECLARATION**

I hereby declare that work, which is being presented in the Internship Report as the partial fulfilment for the award of degree of **Bachelor of Technology** in **Computer Science Engineering** in the **Department of Computer Science Engineering** at **Banasthali Vidyapith**, **Jaipur**, is an authentic record of my work carried out under the Mentorship of Industry Mentor Smt. Bhavna N. Merh **, SOG, ACSD** and The matter embodied in this internship report has not been submitted for the award of any degree.

Roll Number: 2216433 Student Signature Date:

**2022 – 2023**

**INTERNSHIP APPROVAL SHEET**

This is to certify that **Unnati Saini** Roll Number 2216433 has successfully completed her industrial internship starting from 15th May to 15th July 2025 and has submitted the final report. Her work has been found satisfactory and it is recommended to accept it as a partial fulfilment for the award of a degree of **Bachelor of Technology, Computer Science Engineering** of the **Department of Computer Science Engineering** at **Banasthali Vidyapith**, **Jaipur.**

**DEPARTMENT OF COMPUTER SCIENCE ENGINEERING**

# **ACKNOWLEDGEMENT**

I would wish to express my heartfelt gratitude to my Sh. R.K. Agrawal, Head, Software & Embedded Systems Section, ACSD of RRCAT who provided me with the opportunity in his reputed organization. I would like to acknowledge an opportunity provided by Industry Mentor of RRCAT to work at their highly reputed organization.

Secondly, I would like to appreciate Smt. Bhavna N. Merh, Scientific Officer G, ACSD and Dr. Surendra Yadav, Scientific Officer G with whom I worked in RRCAT and guided me in my work. Making it fun to work in, they managed to expose people to their patience and accessibility. It is with a very high feeling of pleasure, and an immense sense of gratitude that I take this opportunity of acknowledging the assistance of these individuals.

Special thanks or gratitude must go to Mr. Abhisek Nayak, who helped significantly and guided me sincerely without which my work would have not been completed successfully. His helpful and skilled support would have made it very difficult to achieve my objective if it were not done with his assistance.

*Name: Unnati Saini*

*Branch: Computer Science Engineering*

**EXECUTIVE SUMMARY**

I have successfully interned in Full Stack Web Development (Scientific Data visualization Publisher Application (Across accelerators)) in the Raja Ramanna Centre for Advance Technology (RRCAT). The key project which I was involved in is the creation of a web-based dashboard of the LINAC-4 accelerator that allows to control the current values and history of important operational parameters.

This project was named as Design and Development of a Web Dashboard to Display Real-Time and Historical Data Plots of LINAC-4 accelerator parameters.

The principal objectives of the given project were to create the web-based application, where one could reproduce and analyze the real-time and historical data of indicators like the beam current, klystron forward power, gun heating current, and vacuum levels and also obtain them as the files. This resolved the issue of a convenient way of interaction between the accelerator operators and the researchers in a form of an accessible platform, boosting the functionality of traditional static tools of monitoring.

The system design consisted of Python (flask/FastAPI) as backend and postgreSQL to persist time-series accelerator data with well written SQL functions to achieve data retrieval effectively. The secure storage of accelerator data in the PostgreSQL database was available with efficient querying and export. I have built a fully responsive frontend dashboard with HTML5, CSS3, Bootstrap, and JavaScript, compiled with the use of libraries, plotly.js to make a scientific plot and TomSelect to select multiple parameters.

The challenges that I met during the internship were connected to multi-parameter plotting, real-time data update, and UI responsiveness where I faced an opportunity to debug, optimize backend (on a regular basis) and integrate its frontend robustly.

To a very large extent, this internship helped me improve my technical abilities in full-stack web development, scientific data visualization, and user interface design. It also helped me develop the capability to develop scalable user-friendly systems which meets the requirements of the real world in the scientific and engineering applications.

Abstract

Guided by the requirements of FY14-q3-101 (T.N.Boucher and D.Bekner), LINAC-4 Dashboard is a web-based application to monitor in real-time and history key operational parameters of the LINAC-4 accelerator like beam current, klystron forward power, gun heating current, and vacuum. It has a responsive client-side developed using HTML5, CSS3, and JavaScript, with some libraries like Plotly.js that implements scientific plotting, TomSelect where multiple parameters should be chosen, and Bootstrap that implements modern graphic design. The backend, written in Python and connected to PostgreSQL database, can fast fetch and deliver that data with some optimized SQL queries and compressed (gzip) response.

Important features are the interactive visualization of the Vacuum v/s Time and the other parameters trends, the plotting on two Y-axis, the dynamically time stamped selection of the parameters and the availability to download current or statistics data in the CSV or EXLXS format. The dashboard offers time-range queries (by user-configurable time windows) as well as auto-refresh of all the data in real-time, with good error handling and user-feedback systems. The reliability of data visualization, data download, and user interface responsiveness was checked with the help of comprehensive testing. LINAC-4 Dashboard improves handling complex accelerator data, presenting a user-friendly and simple-to-use analysis tool that can be used within the scientific and operational teams.

**INDEX**

Introduction

A. The LINAC-4 Introduction

B. The LINAC-4 Website

C. Updates Required

System Design

2.1 Architecture Overview

2.2 Technology Stack

2.3 Flow of Data

A. User Interface

B. Backend Processing

C. Rendering on the Frontend

2.4 Major Modules

2.4.1 Live Status

2.4.2 Historical Data

2.4.3 Generation of Report

2.4.4 Backup Status

2.5 Performance and Security

2.6 Extensibility

2.7 Summary Table

Implementation

3.1 Frontend Implementation

A. HTML Templates

B. JavaScript Logic

C. UI and CSS

3.2 Backend Implementation

A. API Endpoints

B. Logic of Data Retrieval

C. Data Export

3.3 Feature Addition: Vacuum v/s Time

A. Frontend Modifications

B. Backend Modifications

C. Testing

3.4 GUI Error and GUI Message

3.5 Security and Performance

3.6 Extensibility

Testing and Results

4.1 Testing Approach

4.2 Sample Test Cases

4.3 Results

4.4 Observations

Conclusion

Summary of Achievements

Impact

Obstacles & What They Taught

References

Introduction

1. **The LINAC-4 introduction**

The LINAC-4 (Linear Accelerator-4) is one of the most critical elements of accelerator infrastructure of the Raja Ramanna Centre for advanced Technology (RRCAT). It has the role of accelerating charged particles at high energies to allow various scientific experiments and operating needs. Space to monitor key parameters (e.g. beam current, klystron forward power, gun heating current and vacuum levels) in life must take place continuously and is fundamental to control, diagnostic and performance optimization of the accelerator.

1. **The LINAC-4 Web Site**

The LINAC4 facility was used previously in that the public could access a web dashboard program, hosted on the local network (e.g. ￼). Through this dashboard, we got:

* **Live Status monitoring:**

Graphical display (in real time) of parameters of interest (Beam Current, Klystron Forward Power, Gun Heating Current) with auto-refresh and downloading capability.

* **Analysis of Historical Data:**

The instruments to choose time ranges and depict historic trends of the same parameters with the possibility to switch to graphics and table displays.

* **Report Generation:**

Capability of producing and downloading reports based on the desired parameter and periods of time.

* **Backup Status:**

Supervising security of data and systems.

It was coded with HTML5, CSS3, and JavaScript and using libraries like Plotly.js to plot a result, TomSelect to choose a parameter, Tabulator and Tables, and Bootstrap to achieve the responsive layout. The backend was written in Python and communicated with a PostgresSQL database and served the actual data well by optimizing SQL queries and compressing (gzip) responses.

Example:

The initial main page (index.html) had navigation tiles live status, historical data, report generation, and backup status, and a modern branded user-interface.

**Updates Required**

Opportunities to fill these gaps were determined to be the need to update the following:

* To add to the Visualization, Vacuum v/s Time:

Include the selection of Vacuum v/s Time to the module of Live and Historical Data.

Have the convenience of satisfying smooth plotting with available parameters, and the second Y-axis support that facilitates scientific analysis.

* Vacuum Data Backend Support:

Add new SQL functions (e.g., ￼) and API end points to retrieve vacuum data in an efficient way where PostgresSQL is used.

* Frontend Enhancements:

Modernize HTML templates (e.g. add burgers of patrons, in parameter lists).

Newness change JavaScript logic to support dynamic parameter selection and plot parameter, including the new vacuum parameter.

Make sure the new feature is responsive and intuitive UI.

* Test and documentation:

Apply the new capability on both live and historical data.

Revise manuals and user guides to account the improved features.

4. System Design

4.1 Architecture Overview

LINAC-4 Dashboard is designed as an architecturally modern and modular web application which is capable of both real-time and historical monitoring of accelerator parameters. System consists of:

* **Frontend**:

Was created using HTML5, CSS3 and JavaScript.

Plotly.js to make scientific figures, TomSelect to select multiple parameters, Tabulator to handle tabular data, and Bootstrap to have a responsive interface.

* **Backend:**

Written on Python (Flask or FastAPI).

Connection to the PostgreSQL database to save and get the data.

HTTP-based service (RESTful API endpoints) is served in gzip compressed form.

* **Database:**

All parameters monitored can be stored in a time-series in PostgreSQL.

Grid functions (e.g., ickedema ) are optimized to fetch the data to plot or export.

4.2 Technology Stack

|  |  |  |
| --- | --- | --- |
| **Layer** | **Technology/Library** | **Purpose** |
| **Frontend** | HTML5, CSS3, JavaScript | UI layout, styling, and interactivity |
|  | Plotly.js | Scientific plotting with dual Y-axes and live updates |
|  | TomSelect | Interactive multi-parameter dropdown selection |
|  | Tabulator | Tabular rendering of historical data |
|  | Bootstrap | Responsive design and prebuilt components |
| **Backend** | Python (Flask/FastAPI) | API endpoints, logic handling, request processing |
|  | gzip | Compression of large JSON responses for efficient transfer |
| **Database** | PostgreSQL | Storage of time-series operational data, SQL queries |

**4.3 Flow of Data**

**A. User interface:**

Tool enables the user to select parameters (e.g., "Vacuum v/s Time" and time interval) through the dashboard UI.

E.g. chart2.js, a JavaScript, calls the backend API using an AJAX request and posts the chosen options.

**B. Back end Processing:**

The request is received by Backend and parameters, as well as time range, are parsed.

Parses optimized SQL queries (e.g., ‚zeugour Critical word in the French language

Converts the data into JSON (in order to plot it) or CSV/XLSX (in order to download it).

Zips the response with gzip to reduce the delivery time.

**C. Rendering on the front end:**

Accepts the data, plots the interactive plots with the help of Plotly.js (dual Y-axis support parameters such as vacuum and beam current).

Tabular views (in case of selection) are done using Tabulator.js.

File downloads can be activated using dynamic filenames via the download buttons.

**4.4 Major Modules**

**4.4.1 live status**

**Purpose**: Display the real-time plots of the parameters (Beam Current, Vacuum 1, and Vacuum 2, etc.) chosen.

**Features**:

* Auto-refresh: one minute (Toggleable).
* Multi-parameter choice by using TomSelect.
* Plotly double Y-axis plotting.
* Export existing data as CSV/XLSX.

**Files:**

* live\_status.html
* live.js

**4.4.2,Historical Data**

**Purpose:** Analyze and visualize past data with chosen intervals of time.

**Features:**

* Custom range selection date time pickers.
* Plotly and tabular display, supported by Tabulator.
* This is to include multi-parameter plotting, as in Vacuum v/s Time.
* Save/export data to CSV/XLSX.

**Files:**

* chart 2.html
* chart2.js

**4.4.3 Generation of Report**

**Purpose**: Enable users to create and download reports of specific parameters and a period of time.

**Features:**

* Backend gathers and assembles information, zips back.
* Automatic date/time/parameter based filenames.

**Files:**

* genreportoptions.html
* report.js

**4.4.4.Backup status**

**Purpose:** Track the system backup and data integrity.

**Features:**

* Displays last backup time, system status displays.
* Provides safety and integrity of data.

**Files:**

* get\_bkp\_status.html
* backup.js

**4.5 performance and Security**

Security:

* Deployment in local network (e.g. `10.28.8.55`).
* Access to data only to authorized users.

Performance:

* Compression of all large responses to gzip.
* Streamlined SQL in order to ensure rapid data retrievals.
* Efficient frontend paints with plotly and tabulator.

**4.6 Extensibility**

* The modular structure lends itself to add new parameters easily (e.g. add further sensors or diagnostics).
* Frontend and backend codes are structured in a way that allows maintenance and further development.

**Summary Table**

|  |  |  |
| --- | --- | --- |
| **Component** | **Technology/Library** | **Key Features** |
| **Frontend** | HTML5, CSS3, JS, Plotly | Interactive plots, multi-parameter selection, responsive design |
| **Backend** | Python (Flask/FastAPI) | RESTful API, gzip compression, SQL data handling |
| **Database** | PostgreSQL | Time-series data storage, optimized query performance |
| **Data Export** | CSV/XLSX, gzip | Downloadable reports, formatted filenames, compressed transfers |
| **Visualization** | Plotly.js, Tabulator | Dual Y-axes, scientific formats, tabular view, error handling |

**5. Implementation**

**5.1 Frontend Implementation**

The work is created on the basis of HTML5, CSS3, and JavaScript, and some open-source libraries are used to increase the interactivity and visualization of the dashboard:

* Scientific plotting Plotly.js (dual Y-axis, multi-parameter).
* Multi-parameter dropdowns (checkbox).
* Tabular data display use tabulator (where possible).
* Responsive design and UI bootstrap.

**A. HTML Templates**

* **Landing Page:**

The index.html file is used as a main entry point which contains navigation tiles of Live Status, Historical Data, Report Generation and Backup Status.

* **Ploting of Historical Data:**

In chart 2.html there is:

* + Parameters drop down (Beam Current, Klystron Power, Gun Heating Current, Vacuum) on the multi-select dropdown.
  + Custom date/time pickers that allow choosing a range.
  + Plot and downloaded data buttons.

**B. JavaScript Logic**

* **Choice of Parameters and Plot:**

chart2.js handles:

* + Retrieval of picked parameters and time interval through UI.
  + Backend API to perform AJAX requests to obtain data.
  + Creating interactive graphs using Plotly, which uses dual y-axes (e.g. beam current and vacuum).
  + Easy control over the new addition to the options available, which is the Vacuum v/s Time.
* **Download Functionality:**

The Download Data button processes a backend endpoint that downloads the chosen data in the form of a CSV file, whose name is dynamic.

* **Live Status:**

live.js manages:

* + Auto-refresh (each 60 seconds, switchable).
  + Plotting and fetching of the real-time data.

**C. UI and CSS**

Branded interface is guaranteed by Bootstrap and custom-style (styles.css), and is clean and responsive.

**5.2 implementation in the Backend**

It will be written in Python, backend (Flask or FastAPI), with a PostgreSQL database.

**A. API Endpoints**

* **Data Fetching:**
  + Returns JSON-data, basing on the parameters and time interval chosen.
  + It downloads the CSV data of the returns.
* **Compression:**
  + The gzip method is used to compress all large responses in order to transmit them efficiently.

**B. Logic of Data retrieval**

* **SQL Functions:**

Time-series data of all parameters including a new vacuum data are retrieved using optimized SQL functions (e.g., get\_vacuum data(history\_chart 4).txt).

* **Dynamic Parameter Processing:**

The backend is used to parse the params query string to find out which columns to retrieve and give out.

**C. Data Export**

**CSV Generation:**

CSV files for the chosen parameters and a time window are dynamically created in the backend and have names that will indicate the corresponding selection.

The feature to be added was 5.3 Feature Addition: Vacuum v/s time.

**A. Front End modifications**

* Put <li>Vaccum v/s Time </li> under options\_2.html in the parameter selection menu.
* Added the possibility to select and plot a parameter when using the same JavaScript (chart2.js): vacuum.

**B. Backend modifications**

* Installed/modified SQL functions that retrieve data about vacuums.
* Guaranteed that API endpoints accept the new parameter.

**C. Testing**

* Ensured that the option to select, plot and export a Vacuum v/s Time can be selected in both live and historical modules.
* Other parameters such as vacuum will be plotted assuming a smooth dual Y-axis plotting.

**5.4 Gui Error and Gui Message**

* **Loading Spinners:**

Shown so as to represent progress in data fetches.

* **Graceful Fallbacks:**

In the case of missing data or error, any UI will indicate a clear message.

* **Input Validation:**

Makes sure that time ranges and the parameter selections are valid then sends requests.

**5.5 SECURITY AND PERFORMANCE**

* **Deployment Interna Network:**

The dashboard can be reached only in the institutional network (e.g., ￼).

* **Gzip Compression:**

Every large data response is compressed to be faster.

* **Optimized SQL:**

Tuning of queries is configured to a rapid retrieval of even big sets.

**5.6 Extensibility**

The frontend and backend have modular design which means that future additions of new parameters or other features can be carried out easily.

**6. Testing and Results**

**6.1 Testing Approach**

To achieve effective performance of LINAC-4 Dashboard, testing was done to assure the reliability, accuracy, and usability of all the modules and functions. The plan of tests covered:

* **Manual based UI Testing:**

Accessing all dashboard modules (Live Status, Historical Data, Report Generation, Backup Status) by a web browser.

* **API Testing:**

Directly testing backend endpoints via some tool (e.g., /api/vacuum\_data, /api/download\_csv, etc.)

* **Edge Case and error handling:**

Robustness testing: missing data, invalid time ranges and different parameter choices.

* **Performance Checks:**

Monitoring latencies and responsiveness of the UI using large amounts of data and numerous parameters.

**Test Environment**

* **Frontend:**

Server is available though browser at ￼ (or local development server).

* **Backend**:

Flask (FastAPI) server with the PostgreSQL database.

* **Data**:

All parameters supported have real and simulation accelerator data.

**6.2 Sample Test Cases**

|  |  |  |
| --- | --- | --- |
| **Test Case** | **Steps** | **Expected Result** |
| **1. Live Status Plot** | Select "Vacuum v/s Time" in Live Status, observe plot | Real-time vacuum data plotted, auto-refresh works |
| **2. Historical Data Plot** | Select "Vacuum v/s Time" and custom time range, click "Show plot" | Historical vacuum data plotted correctly |
| **3. Multi-Parameter Plot** | Select "Beam Current" and "Vacuum v/s Time" together | Both parameters plotted with dual Y-axes |
| **4. Download CSV** | Select parameters and time range, click "Download Data" | CSV file downloaded with correct data and filename |
| **5.Invalid Time Range** | Select end time before start time, try to plot | UI shows error, no backend call made |
| **6. Missing Data** | Select a time range with no data | UI displays "No data available" message |
| **7.Auto-Refresh Toggle** | Enable/disable auto-refresh in Live Status | Data updates every 60s when enabled, stops when disabled |
| **8. UI Responsiveness** | Rapidly switch between modules and parameters | No crashes, UI remains responsive |

**6.3 Results**

* All the plots with updated live and historical data were correct, and the new choice of the plot was included the option "Vacuum v/s Time".
* Multi-parameter select was fully functional and the plotting of two Y axes, such as Beam Current and Vacuum, took place very well.
* Properly filled CSV files were retrieved with proper file names which were not fixed but represented the chosen parameter and time range.
* Live Status introduced auto-refresh of plots which was possible to modify to every 60 seconds.
* Handling of errors was well done invalid inputs and incomplete information were well handled with good feedback to the user.
* The performance was to an acceptable standard: data could be retrieved quickly and the user interface interactors with large data well.

**Screenshots :**

Landing page- Dashboard

The selection of parameters with the use of Vacuum v/s Time

Dual Y-axis Example Plotly chart

Download conversation and test CSV file

Error message of invalid data

**6.4 Observations**

* The subsequent addition of Vacuum v/s Time achieved a major advance in diagnostic ability, whereby the operator could relate vac trend to others.
* The frontend and backend design as modular allowed the addition of new features and testing of the same.
* Use of gzip compression and optimized SQL queries provided speed in the delivery of data even in large time windows.

**7. Conclusion**

**Summary of Achievement**

* Created an extensive web-based interface with LINAC-4 accelerator; allowed real-time and historical monitoring of the most critical operational parameters: Beam Current, Klystron Forward Power, Gun Heating Current and Vacuum.
* Advanced data visualization in an integrated Population Manager with an open source Plotly.js that allows plotting multiple parameters with dual Y-axes, and multiple-parameter selection with TomSelect extension.
* Introduced the Vacuum v/s Time function, which gives a user an opportunity to graph vacuuming trends along with other parameters allowing to have better diagnostics and understanding of the operational situation.
* Facilitated powerful data export that supports dynamic CSV downloads and both live and historical data.
* Confirmed the user-friendly interface that was responsive with the help of HTML5, CSS3, Bootstrap, and personal JavaScript; navigation and error management was obvious.
* Used Python (Flask/FastAPI), PostgreSQL and gzip-compressed response to optimize the backend performance and the delivery of data.

**Impact**

* **Improved monitoring of operations:**

The dashboard gives accelerator operators and researchers immediate access to critical data whereby situation awareness and decision-making.

* **Improved diagnostics:**

With the incorporation of a message "Vacuum v/s Time", it is possible to correlate with other characteristics of the vacuum level helping to solve the problem easily and do preventative maintenance.

* **Optimized analysis of data:**

Visualization and export tools are also easy and conserve the manual work to allow better offline analysis.

**Obstacles & What They Taught**

* **Data integration:**

Proper frontend/backend change to the logic, SQL functions as well as API end points had to be implemented to ensure smoothness in integrating the new parameters (which in this case was vacuum).

* **UI/UX consistency:**

One important design challenge was to provide an effective user experience that is very similar (if not identical) as before but at the same time offering new functionality (multi-select, dual Y-axis).

* **Performance optimization:**

Fast, responsive plotting as well as large datasets required the use of compressed responses and optimized SQL queries.

* **Test edge case:**

It was necessary to implement substantial error handling in order to easily handle missed data, invalid input, and network.

**8. References**

* Plotly.js Documentation
* TomSelect Documentation
* Tabulator Table Library
* Bootstrap Documentation
* Typed.js Documentation