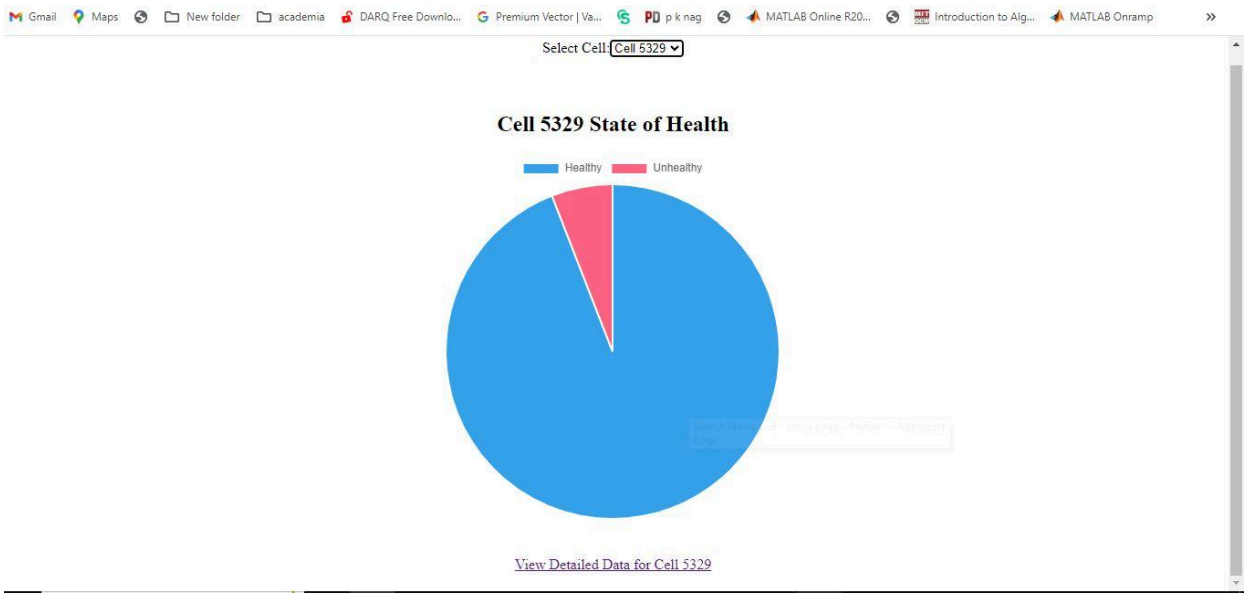
Project Documentation: Cell Monitoring Dashboard

Table of Contents

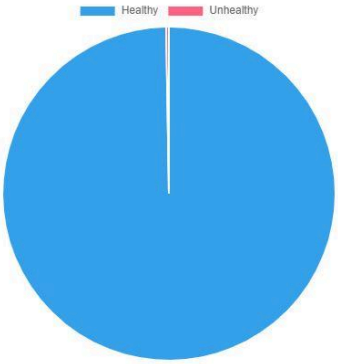
Overview

The Cell Monitoring Dashboard is designed to visualize various data related to cell performance, including current, voltage, capacity, and temperature. It consists of a backend built with Flask and SQLAlchemy and a frontend built with React and Chart.js. The dashboard provides a user-friendly interface to monitor the state of health and performance metrics of different cells.



Select Cell: Cell 5308

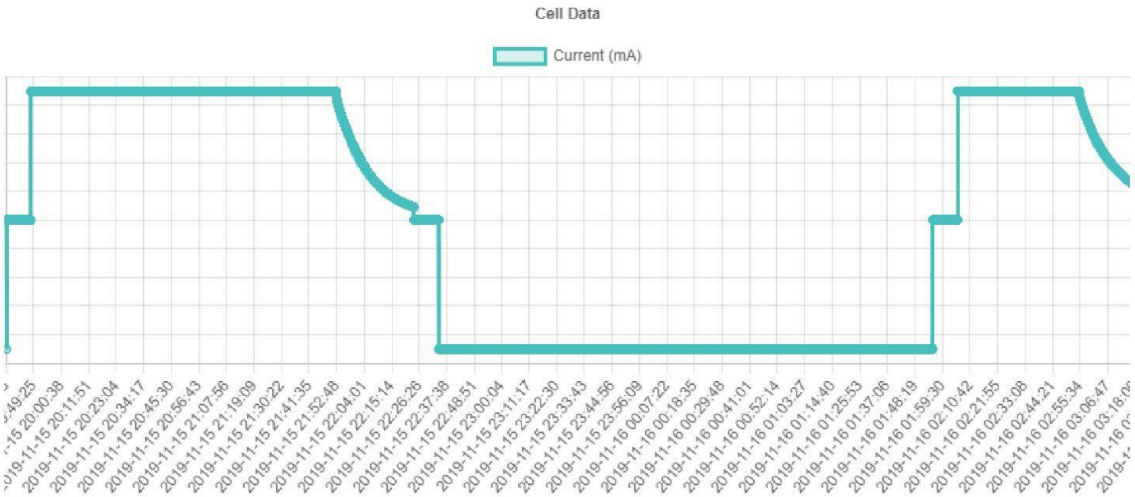
Cell 5308 State of Health



View Detailed Data for Cell 5308

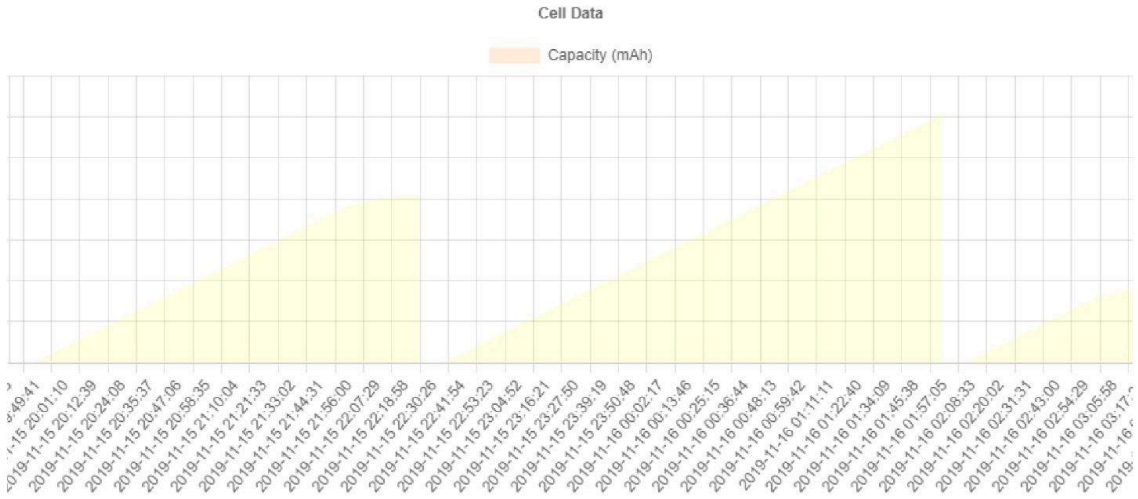
Cell 5329 Data

Current Data

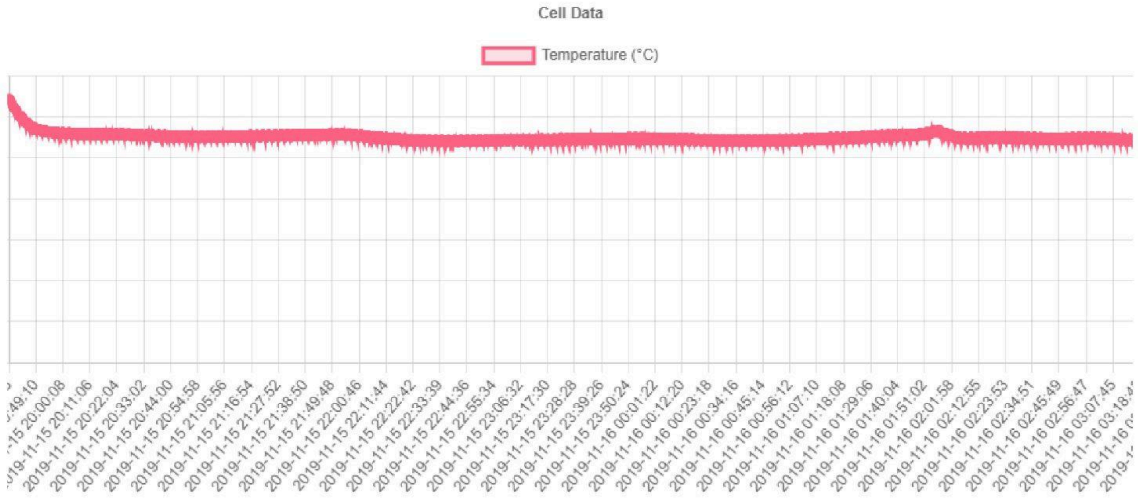


Voltage Data

Capacity Data



Temperature Data



Backend

Project Structure

The backend is organized into a few key directories and files:

- `app/`: Contains the application code, including models, routes, and utility functions.
- `migrations/`: Handles database migrations.
- `run.py`: The entry point for running the application.
- `config.py`: Configuration settings for the application.

Models

The backend uses SQLAlchemy to define models for storing cell data. Key models include:

- `Cell`: Represents a cell with attributes like name, manufacturer, capacity, chemistry, and voltage.
- `CellData`: Stores detailed data for each cell, including cycle, step, status, voltage, current, capacity, energy, and timestamps.
- `CycleData`: Captures cycle-specific data such as channel, total cycles, charge capacity, discharge capacity, and cycle life.
- `StatisticsData`: Stores statistical data related to the cells, including various voltages, currents, capacities, and energy metrics.
- `DetailData`: Store detailed current and voltage, temperature.
- `DetailVoltageData` and `DetailTemperatureData`: Store detailed voltage and temperature data respectively.

Database Setup

The database is managed using SQLAlchemy, with migrations handled by Flask-Migrate. The database stores all the data models and supports relationships between them, such as linking `CellData` entries to specific `Cell` entries.

Utility Functions

Utility functions are used for processing and importing data from Excel files. These functions read data from the files, create model instances, and add them to the database.

Routes

The backend provides several API endpoints for fetching data:

- `/cycle_data`: Returns cycle data for cells.
- `/statistics_data`: Returns statistical data for cells.

- `/detail_voltage_data`: Returns detailed voltage data.
- `/detail_temperature_data`: Returns detailed temperature data.
- `/current_data/<cell_id>`: Returns current data for a specific cell.
- `/upload`: Handles file uploads for importing data.

Frontend

Project Structure

The frontend is organized into several directories and files:

- `components/`: Contains reusable React components.
- `pages/`: Contains page components for different routes.
- `api.js`: Contains functions for making API requests to the backend.
- `App.js`: The main application component that sets up routing.

Components

Key components include:

- `Dashboard`: Displays the state of health (SoH) of different cells using Pie charts.
- `CurrentData`: Displays current data for a specific cell using a Line chart.
- `CellPage`: Displays detailed data (current, voltage, capacity, temperature) for a specific cell using various chart types.
- `HomePage`, `UploadPage`: Handle the main homepage and file upload functionality.

API Integration

The frontend integrates with the backend using axios for making API requests. It fetches data from various endpoints and displays it using Chart.js for data visualization.

Routing

React Router is used to handle routing between different pages:

- `/`: The homepage.
- `/upload`: The file upload page.
- `/cycle-data`: Displays cycle data.
- `/statistics-data`: Displays statistical data.
- `/detail-voltage-data`: Displays detailed voltage data.
- `/detail-temperature-data`: Displays detailed temperature data.
- `/dashboard`: The main dashboard displaying SoH charts.
- `/cell/:cellId`: Displays detailed data for a specific cell.

Libraries for charting

The frontend uses the following libraries for charting and data visualization:

- **Chart.js**: A powerful and flexible JavaScript library for creating charts. It supports various chart types, including Line, Bar, and Pie charts.
- **React-Chartjs-2**: A React wrapper for Chart.js that makes it easy to integrate Chart.js charts into React components.

Running the Project

To run the project, follow these steps:

1. **Backend:**
    - Install dependencies: `pip install -r requirements.txt`
    - Run migrations: `flask db upgrade`
    - Start the server: `flask run`
2. **Frontend:**
    - Install dependencies: `npm install`
    - Start the development server: `npm start`

Ensure the backend runs on the correct port (e.g., 8080) and update the API URL in the frontend if necessary.

Conclusion

This project provides a comprehensive dashboard for monitoring cell performance. The backend efficiently handles data storage and retrieval, while the frontend offers an intuitive interface for visualizing various metrics. By following the structure and using the provided components, the dashboard can be easily extended to include additional features and data visualizations.