Remaining Battery Life

COM S 402 SENIOR DESIGN DOCUMENT

Team Number: 1
Client: Chao Hu
Team Members/Roles:

Kathryn Rohlfing: Frontend
Evan Williams: Frontend
Jihoo Kim: Networking
YewKen Chai: Backend
Aidan Webster: Backend

https://seniord.cs.iastate.edu/2022-May-01
Revised: 5/10

EXECUTIVE SUMMARY

Development Standards & Practices Used

List all standard circuit, hardware, software practices used in this project. List all the engineering standards that apply to this project that were considered.

- Modularity in code
- Unit testing
- CI/CD

Summary of Requirements

List all requirements as bullet points in brief (organize by features).

- Display dashboard overview of all batteries being tested
- Allow users to rename battery cells running in a channel
- Allow users to specify the test type running on a particular channel
- Display graph data for each battery being tested
- Allow users to download history for a particular cell
    - Specify start and stop cycles for which to download full history
    - Download graphed data for each graph
- Allow users to generate predicted capacity fade for a cell
    - Specify EOL threshold and cutoff for each prediction

Applicable Courses from Iowa State University Curriculum

List all Iowa State University courses whose contents were applicable to your project.

- Com S 319
- Com S 309
- Com S 363
- Com S 417
- Cpr E 489

New Skills/Knowledge acquired that was not taught in courses

List all new skills/knowledge that your team acquired which was not part of your Iowa State curriculum in order to complete this project.

- In-depth usage of React & its modules
- Communicating with hardware (Neware tester) API
- Vocabulary and formulas specific to battery testing research
- Working with large datasets

Table of Contents

1 Introduction

1.1 Acknowledgement

Thank you to Tingkai Li for providing support on the requirements of the project and meeting weekly with the team to represent the client and confirm the progress of the project.

Additional thanks to Chris Chen from Neware Battery for providing product support and answering the team's question about the Neware battery tester's operations.

1.2 Problem and Project Statement

Problem statement

Researchers at Iowa State University are conducting research on battery cells to better understand and predict their durability over time, by charging and discharging the cells in cycles and monitoring their data such as capacity and voltage. However, over a long period of time this generates a large amount of data, which is difficult to visualize and is only available on the lab computer connected to the testing machine. Moving and analyzing this data is a very manual process.

Solution approach

Our solution is to create a web tool that allows users to view the data from any modern browser connected to the Iowa State network. The data will be presented in several views – the home dashboard page, which provides an overview of all cells being tested, as well as a more specific information page for each cell which provides graphs displaying key characteristics.

1.3 Operational Environment

The product will be used within a web browser, and the only requirements are that the device it runs on must be connected to the Iowa State network, or connected to the Iowa State VPN, and that the browser must be up to date.

1.4 Requirements

The requirements for this project are to create a web tool which will display up-to-date information about all cells being tested in a Neware battery tester. It should provide both an overview of the status of all cells, as well as more detailed information via graphs for a particular cell. A cell's data should be downloadable for use in machine learning. The tool should allow user input for values such as the name of a cell currently running in a

channel and the type of the test currently being run. All data should update periodically automatically. Additionally, it should generate a prediction for the future capacity graph of a cell upon user request.

1.5 Design Assumptions and Limitations

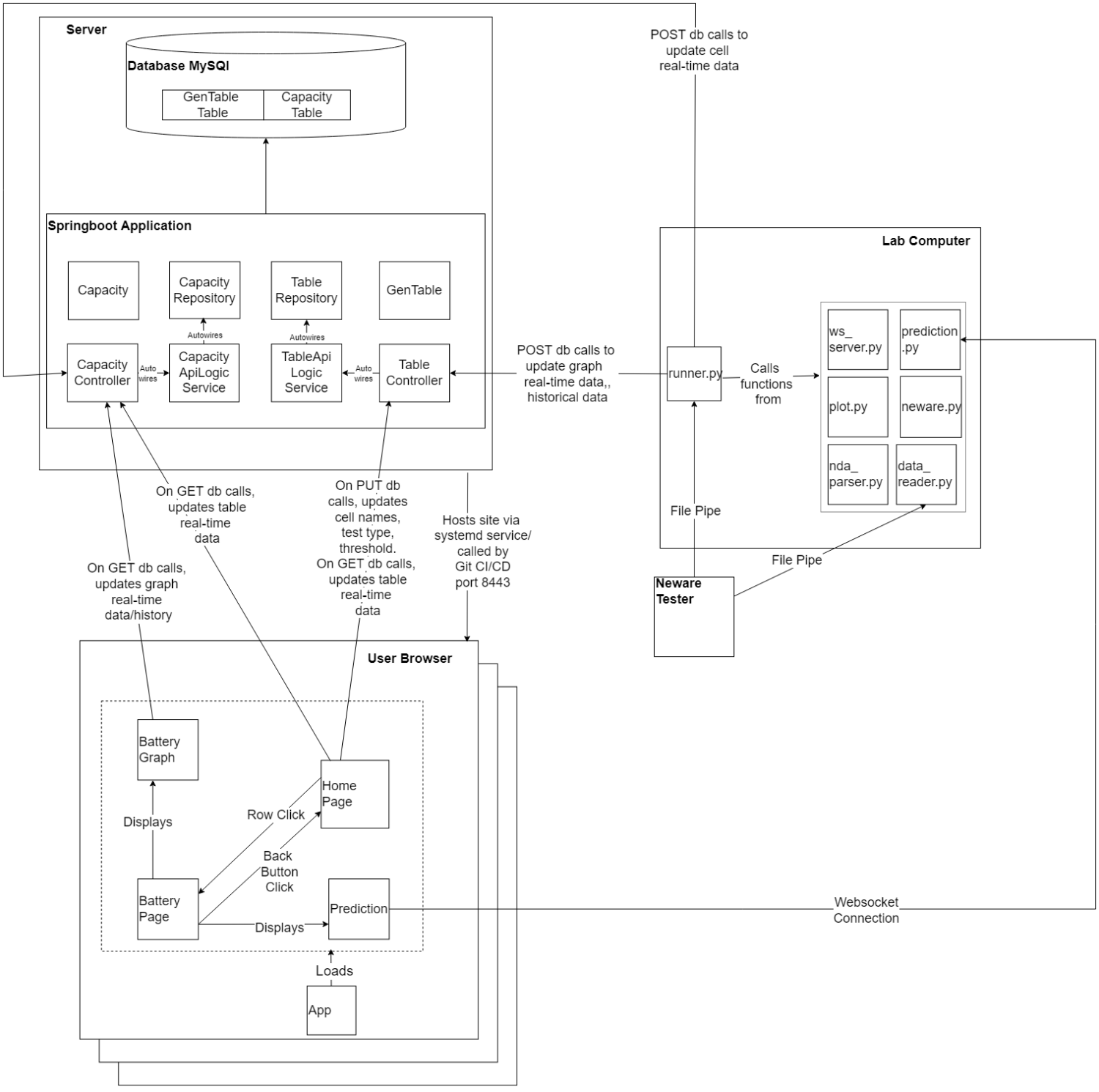
Assumptions:

- The users will be using a modern browser that is capable of running html and Javascript.
- The server provided for this class will be supported and available for the duration of the use of the web tool.

Limitations:

- The graph package used, CanvasJS is being used on a trial basis
    - Other packages tested for this project were unable to render the large amount of data needed.
    - The team's application for a student license was denied, and the cost of buying a license was greater than the client was interested in.
- The database's memory cannot support storing full data, so only the most recent portion will be stored.
    - Only while the backend runner is active on the lab computer will the data be updated.
    - The database will need to be cleared if the runner is run continuously long-term.

2 Architectural Diagram



3 Components

Our project consists of several components – the frontend, which displays the data in the web tool, the Springboot Application, which connects the database to the other components and makes the data available, the Backend Neware Connection, which fetches the data from the tester, the Capacity Fade Prediction, which processes data from the frontend, and the Initial Database Export Script and Database Load Handler components, which work together to populate the database initially.

3.1 Component 1 - Frontend

The component is the code used to build and render the website using React, with additional packages of CanvasJS, MUI, and Grommet. It is hosted by the server provided for Com S 402 class, and can be accessed on the campus network, or on any network with the campus VPN. The site consists of 2 main pieces – the main dashboard, and the battery information page.

3.1.1 Module 1 - Home page

The home page displays general data for all channels on the battery tester. It updates periodically to display near real-time data by making calls to the Springboot application endpoints retrieving data from the database. Additionally, this page uses Grommet as well as a Javascript style sheet created by the team for formatting.
The main elements in this page are a Grommet Data Table to display the channel's data, and the header bar. Within the header bar, a form to edit a channel's current cell name can either be expanded or collapsed, changing the state of the page.

3.1.2 Module 2 - Battery page

The battery page displays graph data for an individual cell. Additionally, this page uses Grommet as well as a Javascript style sheet created by the team for formatting.
The main elements in this page are CanvasJS graphs drawn by Module 3, as well as a header bar that contains several buttons and the tab panel. The tab panel controls which graph is visible, and users can select a labeled tab to view the graphs of that type. Additionally, a form can be expanded to download the cell's data for a specified range of cycles.

3.1.3 Module 3 - Battery Graph

This module consists of a class which extends the CanvasJS graph package to add functionality for periodically updating the graphs. The data is refreshed by making calls to the Springboot application endpoints retrieving data from the database. Before making a call to pull new data, first this module checks if the graph is visible based on the currently open tab – if not, the graph does not need to be updated or rendered to save resources.
This module is used by Module 2 to render each graph that is displayed on the page.

3.1.4 Module 4 - Prediction

This module consists of a class which uses CanvasJS to display a graph of the predicted capacity fade curve when the user inputs the required information of EOL threshold and cutoff to be used for the prediction. It uses websockets to connect with Component 3's Module 6, which generates the prediction data using machine learning.
This module is displayed by Module 2 within the Capacity Fade Curve tab, and is created with a reference to the same data object updated by the capacity fade instance of Module 3 so that the most up-to-date capacity values can be used for the prediction without duplicating database calls unnecessarily.

3.1.5 Module 5 - App

This module consists of a class which uses Routes to set up navigation and the structure of the app. It is the code that is loaded at startup by the index file, but displays Home by default at the initial url and sets the url layout for the battery pages.

3.2 COMPONENT 2 - SPRINGBOOT APPLICATION

This component represents the connection between the database and the frontend and backend components. It creates the endpoints that can be used to access or update information stored in the database.

3.2.1 Module 1 – Controller

This module consists of two classes, TableController and CapacityController, which are used to create the endpoints for the database GenTable and Capacity tables, respectively. For each endpoint, the controller makes a call to the autowired LogicService for the appropriate table to either read or set the correct data and provide a response.

3.2.2 Module 2 – Entities

This module consists of two classes, GenTable and Capacity which represent the information stored in the database tables. They include each value with its name and data type, along with a get and set method for each. The Gentable are linked to Capacity with One-to-Many relationship.

3.2.3 Module 3 – LogicServices

This module consists of two classes, TableApiLogicService and CapacityApiLogicService which provide the logic for fetching the correct information from the database and processing it to be returned from the endpoint by the controller classes.

3.2.4 Module 4 – Repositories

This module consists of two classes, TableRepository and CapacityRepository which are used to automatically retrieve information from the database. They are declared only and

require no concrete implementation in these classes due to the @Repository tag from Spring JPA.

3.2.5 Module 5 - Application

This module consists of the Coms402Application class, which starts the spring application. It combines the previous four modules and is deployed by Gitlab's CI/CD when the main branch is updated.

3.3 Component 3 – Backend Neware Connection

This component pulls information from the Neware tester periodically and posts the updates to the database or respond to requests via websocket, so the newest information is available in the frontend. Module 5 runs continuously on the lab computer using Modules 1-4 which provide utilities to facilitate the information flow.

3.3.1 Module 1 – Data Reader

This module uses data pipes to connect to the Neware tester and pull information. It is used in Modules 3 & 5 to get updated information on the channels within the tester.

3.3.2 Module 2 – NDA Parser

This module converts information to needed file types, such as csv and nda, and to dictionary objects. It is used as a dependency for utility functions required by Module 4.

3.3.3 Module 3 - Plot

This module provides the methods which generate each kind of graph required for the project. It uses DataReader to pull the needed information from the Neware tester and returns a dataset containing the coordinates for a particular graph. It is used

3.3.4 Module 4- Neware

This module uses pipes to connect to the Neware tester and retrieve necessary information. It is used in Module 5 to support the utility functions needed to provide up to date information from the database.

3.3.5 Module 5 - Runner

This module runs continuously to post channel information to the database after pulling the most recent data from the Neware tester and formatting the data into JSON datasets so that it can be stored in the database and parsed by the frontend.

3.4 Component 4 – Capacity Fade Prediction

This component uses information provided from the frontend to generate a predicted capacity fade curve and return the data to the frontend to be displayed.

3.4.1 Module 6 – Capacity Fade Curve Prediction Endpoint

This module consists of two files, ws_server.py and prediction.py. The file ws_server.py creates a websocket server to run prediction on capacity fade curve data and handles the messaging protocol between client and server. It calls prediction.py for generating the predictions, then formats the prediction results before sending it to the client.

3.5 Component 5 – Initial Database Export Script

This component pulled information from the Neware tester that could be used to initialize the database with data. This script is not run periodically and was only as a one-time-load for the database. However, in the event that the database would need to be reloaded or restarted, this component could be used to re-initialize it.

3.5.1 Module 1- Data Init

This module consists of one file, data_init.py.

This module uses the database's http endpoints to create tables for each cell and saves time series data of each cell to disk. The time series data are parsed into pandas' *DataFrame* objects and exported as pickle files for convenience.

3.6 Component 6 – Database Load Handler

This component used the pulled information from the Neware tester to initialize the database with data. This script is not run periodically and was only as a one-time-load for the database. However, in the event that the database would need to be reloaded or restarted, this component could be used to re-initialize it after removing the old data.

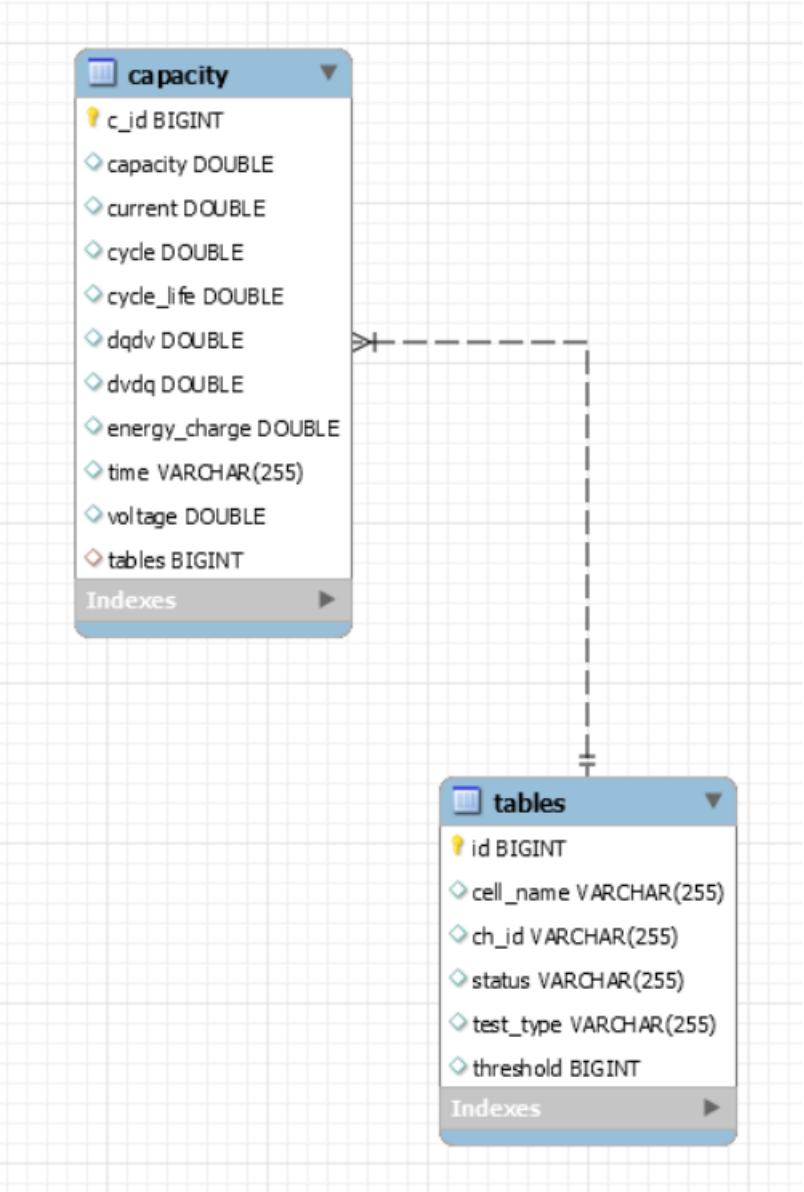
3.6.1 Module 1 – Handler

This module consists of one file, handler.py.

This module runs on the server and inserts the time series data generated by data_init.py into the database.

4 Data Decomposition

The web tool relies on a database using MySQL to store the data, with the following entity-relationship diagram:



The database will be flushed monthly with the older data being stored offline on the lab computer to prevent overflowing it with data.

5 Detailed Design

The documentation is posted in our git repository at the following link: https://git.linux.iastate.edu/seniordesigncoms/2022spring/2022_1_chao_remainingbattery/remaining-battery/-/tree/main/Documentation

The frontend code has been documented using a JavaScript documenting tool called JSDoc. To view the results of this tool, download this folder or clone the repository and open Frontend/index.html within the above folder in your preferred browser.

All other documentation is provided in markdown or pdf format and can be viewed within Git or downloaded and viewed locally.

6 Design Rationale

6.1 Design Issues

6.2 Issue 1 – Selecting an appropriate graph package

6.2.1 Description:

6.2.1.1 Initially when creating and testing the graph page, Victory was selected as the chart package for being highly rated, well-supported, and user-friendly. However, the large size of the graph data (approximately 2 MB for one graph's (x, y) coordinates) was not easily rendered with this package, and each graph took between 5-10 seconds to load, or at times could crash the browser.

6.2.2 Factors affecting Issue:

6.2.2.1 Short timeline – this issue was discovered with approximately 3 weeks remaining in the project, limiting how much we could investigate options

6.2.2.2 Intention of keeping costs minimal

6.2.3 Alternatives and their pros and cons:

6.2.3.1 Continue using Victory

6.2.3.1.1 Pro: Already was implemented, so efforts could go to streamlining it as much as possible, as well as implementing other features.

6.2.3.1.2 Con: Reduces usability of the site.

6.2.3.1.3 Con: Likely would not be able to update data as often, as re-loading the data every 10 seconds, when it could take 10 seconds to load, would barely display the graphs.

6.2.3.2 Display only a subset of the data

6.2.3.2.1 Pro: Victory could still be used, and the page would not have to be refactored.

6.2.3.2.2 Con: It may be difficult to determine which data points are necessary.

6.2.3.2.3 Con: Reducing the data may take more processing power than rendering it as before.

6.2.3.3 Search for other packages

6.2.3.3.1 Pro: There is a possibility that a lesser-known graph package can display the large amount of data without requiring licensing.

6.2.3.3.2 Con: With limited time, there is no guarantee that another suitable replacement will be found with time to implement it.

6.2.3.4 Switch to using CanvasJS

6.2.3.4.1 Pro: Is a commercial tool capable of displaying the large quantity of data.

6.2.3.4.2 Pro: Is similar in setup to Victory, so can be implemented without total refactoring.

6.2.3.4.3 Con: Will require some refactoring in the battery page to update from Victory to CanvasJS.

6.2.3.4.4 Con: Requires license – we can use the Trial Version but is intended only for evaluating the tool and comes with a small watermark, and the student license application was denied.

6.2.4 Resolution of Issue:

6.2.4.1 The team consulted with Tingkai and decided to proceed with CanvasJS, as the client felt that it was more important to have a working proof of concept than a licensed tool.

6.3 Issue 2 – Determining Data flow

6.3.1 Description:

6.3.1.1 Initially, the client and team believed that a database would not be necessary, and all data could be pulled from the Neware system. However, some proposed features would not be possible without a database.

6.3.2 Factors affecting Issue:

6.3.2.1 Large amounts of data in a database could be difficult to store long term

6.3.3 Alternatives and their pros and cons:

6.3.3.1 Use the provided Neware system API option to send a channel's full data.

6.3.3.1.1 Con: If two different cells were run in the same channel, with this option there would not be a way to differentiate between the cells automatically.

6.3.3.1.2 Pro: This would simplify the system and allow the team to implement more features.

6.3.3.2 A database could be created to store the data

6.3.3.2.1 Pro: A database paired with Springboot endpoints would streamline the frontend's retrieval of data.

6.3.3.2.2 Pro: A database would allow for differentiating between cells' history run on the same channel.

6.3.3.2.3 Con: A plan would need to be in place so the database could be cleared periodically to remove stale data and free space to improve performance and prevent running out of memory.

6.3.4 Resolution of Issue:

6.3.4.1 The team, along with the client and Dr. Mitra determined that creating a database was the best course of action.

6.4 Issue 3 – Updating graphs with real-time data

6.4.1 Description:

6.4.1.1 One of the goals of the project was to update the graphs periodically to display near real-time data. However, the CanvasJS package that was chosen as a substitute for Victory does not reliably re-render the data within React when the data has changed.

6.4.2 Factors affecting Issue:

6.4.2.1 The importance of real-time data to this project mean that the solution should be as usable as possible.

6.4.3 Alternatives and their pros and cons:

6.4.3.1 The data could be loaded on page refresh or switching tabs.

6.4.3.1.1 Pro: This would reduce the number of unneeded database calls, as the data would only be refreshed when the user intentionally refreshed it.

6.4.3.1.2 Con: This is not a user-friendly solution.

6.4.3.2 The graph class provided by CanvasJS could be extended and modified

6.4.3.2.1 Pro: This would simplify the code and make it more modular, making it easy to add new graphs to the page.

6.4.3.2.2 Con: If the graph package is changed in the future, as it is a trial version, there may need to be another solution.

6.4.3.2.3 Pro: If a new graph package is changed in the future, assuming that it is able to re-render the data, it would be much simpler to exchange the package code, as it is all in one central location.

6.4.4 Resolution of Issue:

6.4.4.1 This issue was resolved by creating a new class component in JS to extend the chart class provided by CanvasJS to access methods to re-render the graph on data updates. Additionally, this allowed the team to make the code more modular by relocating the graph database calls to the new graph class rather than the battery page.

6.5 Issue 4 – Loading data into database

6.5.1 Description:

6.5.1.1 In order to give the frontend access to the data, it needed to be initially loaded into the database with the history from Neware, then updated periodically with the new data after the initial load. However, using individual post requests to the database to initially load the data caused out of memory errors as well as too many transmission errors in the database, and rendered it unusable for that time.

6.5.2 Factors affecting Issue:

6.5.2.1 The provided server where the database is run has a limited amount of memory. The database's memory can be increased, but only to a certain point.

6.5.3 Alternatives and their pros and cons:

6.5.3.1 Only send the most recent data points

6.5.3.1.1 Pro: This would likely not overload the server's memory while still providing updated data.

6.5.3.1.2 Pro: This would prevent the too many transmissions error, which came about from the initial post requests happening too quickly together.

6.5.3.1.3 Con: This could lead to missing data points, which could yield invalid models for machine learning and inaccurate representations of the status of cells.

6.5.3.2 Initially update the database using MySQL insert multiple row option, and only use post method for updates, not initial load

6.5.3.2.1 Pro: This would prevent the too many transmissions error, which came about from the initial post requests happening too quickly together.

6.5.3.2.2 Con: This could require some refactoring to create the insert query

6.5.3.3 Initially update the database using MySQL import file option, and only use post method for updates, not initial load

6.5.3.3.1 Pro: This would prevent the too many transmissions error, which came about from the initial post requests happening too quickly together.

6.5.3.3.2 Pro: If it can be imported in JSON format, this should require minimal refactoring

6.5.4 Resolution of Issue:

6.5.4.1 The issue was resolved using a combination of the alternatives. The database was loaded using a series of MySQL queries with multiple rows each. Only a subset of the most recent data was used in the initial load so that there was a smaller amount of data that would not overload the server. After the initial load, however, data would be constantly updated at a set interval.

6.6 Issue 5 – Determining Test Type

6.6.1 Description:

6.6.1.1 In the lab, there are currently two types of tests that can be run on the batteries – cycling and characterization. However, for predicting the capacity fade curve of a cell, only data from the characterization test should be used.

6.6.2 Factors affecting Issue:

6.6.2.1 Tests or their characteristics, like duration, may change in the future

6.6.2.2 Currently there is not an automatic way to determine which test type is running as it's manually configured.

6.6.2.3 This issue was discovered with approximately 2 weeks left in the project, leaving a short timeline.

6.6.3 Alternatives and their pros and cons:

6.6.3.1 Rely on user input within the web tool to set the test type

6.6.3.1.1 Pro: This code would be similar to the edit cell names feature, so would be straightforward to implement.

6.6.3.1.2 Con: This requires the users to be active on the web tool when changing the test type to record the most accurate data.

6.6.3.2 Automatically determine upcoming test type when a test ends

6.6.3.2.1 Pro: This would make the process more efficient and less manual.

6.6.3.2.2 Con: If this is based on duration of the previous test, or expecting that the test type alternates, there could be errors when the test schedule is changed, or a new cell is used in the same channel for a different test.

6.6.4 Resolution of Issue:

6.6.4.1 The team, along with Tingkai, agreed that the best solution would be to set the test type for each cell in the database.