App for Microgrid Demonstration

Design Document

Team: sddec21-21

Client: Anne Kimber

Advisors: Mathew Wymore, Steve Nystrom, Nicholas David

Team: Gabriel Rueger, Michael Doyle, Micheal Thai, Patrick Shirazi, William Bronson

Email: sddec21-21@iastate.edu

Website: http://sddec21-21.sd.ece.iastate.edu

Revised: April 25, 2021 - v3

Executive Summary

Development Standards & Practices Used

Waterfall Model for development cycle (Royce).
Coding Standards to maintain quality code base (Xuefen).
Digital design standards for optimization and consistency of our mobile application (Designing).

Summary of Requirements

- The server shall have the ability to add additional crates from the microgrid
- The database shall have the ability to add additional data sources to the database for each crate
- The database shall be able to query and search different subsets of the data
- The database shall have a configurable data collection interval
- The database shall support automatic archiving of data
- The mobile application shall display data within a minute of collection
- The database size shall scale linearly with the number of data sources and with time
- The server shall reduce old data to average data over a period of time
- Frameworks, libraries, etc. for this project shall be well-supported and maintained
- Throughout this project, open and well-supported communication standards shall be used
- All decisions made throughout this project shall be well documented

Applicable Courses from Iowa State University Curriculum

SE/CPRE/EE 185
COM S 227, 228, 309, 363
S E 329, 339

New Skills/Knowledge acquired that was not taught in courses

The EE students will be introduced to Java programming language. The EE students also learned some basics of the backend, frontend, and database aspects of this project. TBD as progress on the project continues.

Table of Contents

List of figures/tables/symbols/definitions

1 Introduction

1.1 Acknowledgement

Lead Project Advisor - Anne Kimber

Advisors - Nicholas David, Steve Nystrom, Mathew Wymore

1.2 Problem and Project Statement

The ISU Electric Power Research Center operates and conducts research on a crate that has microgrid data on it from multiple data sources not limited to solar panels and a Tesla Powerwall. The only current access to this data is either by driving to the crate itself and manually accessing the onsite computer, or by creating a remote connection to the computer.

For our project, we propose to create an application to retrieve and present energy data collected by solar crate to public users and crate maintainers. The user can interface with the application through a mobile app. This application will serve as an efficient way for public users and crate maintainers to analyze efficiency of power collection and distribution of the solar crate from data presented through the application. Crate maintainers will be able to gain a better understanding of how a crate collects energy over time. Crate users will know what the crate could be used to power given how much power is stored in the crate's battery storage.

1.3 Operational Environment

The application will need to be functional on iOS and Android mobile platforms. This application will be exposed to and used by the public to gain information about the sun crate. Application components will also need to be performant to allow multiple users to analyze the application's data. The application will also need the ability to scale with the number of crates deployed at that time.

Additionally, a vital point of the operational environment is the network environment. For our application, the device must be connected to the Internet. It can be connected to any network, not solely just the crate or ISU's network.

1.4 Requirements

The requirements from our faculty advisors for the 2021 Spring semester to develop an app to show the operational data from a solar/Tesla powerwall/storage/wind/diesel/car charging off-grid microgrid operating in Ames, IA, are as follows:

For the backend:

- It should easily extend to additional data sources and display methods.
- It should have a configurable data collection interval.
- It should have an automatic support archiving of data.
- It should take 10-minute averages for older data.
- It should be a secure but open system (it does not need authentication but should be extendable).
- The database size should scale linearly with the number of data sources and with the time.

For the frontend/display:

- It should be a mobile app (iOS & Android).
- It should support browsing raw/averaged data.
- It should support graphing data over time, configuring time range, and showing the data sources.

For the whole system:

- It should fit into the client's architecture.
- It should have a max 1-minute delay between updated value and display.
- It must use open and well-supported communication standards.
- The software should be maintainable.
- The system must be secured to protect the data from being exposed to users and cannot be modified by average users.
- The frameworks and libraries must be well-supported and maintained.

1.5 Intended Users and Uses

This mobile application is intended for the educational/informational purposes and publicity and will display the microgrid's overall performance. Additionally, researchers should be able to access the voltage, current, and frequency data readings.

1.6 Assumptions and Limitations

The following assumptions are as follows:

- The mobile app will only take data from one solar crate during development.
- Students will be given a way to access the microgrid data via the remote desktop.
- The mobile application should be able to display a time range of data from a crate in a graphical format.
- The mobile application only needs to use the English language.
- The team will be provided with a virtual machine to host the application.

The following limitations are as follows:

- The budget to produce the application will be negligible.
- The mobile application will be developed by the end of the 2021 Spring semester.

1.7 Expected End Product and Deliverables

The end product of this project is an application that collects and displays operational data from an off-grid microgrid in Ames. To accomplish this, a number of deliverables must be achieved. These deliverables include:
– A data aggregator to collect all the data from the data sources. This will function to access and process the operational data of the microgrid into the form that the rest of the application needs.

– A database to store the collected data. This will store all the collected data from the data aggregator and will make the data readily available to the rest of the application.

– A mobile app to display and interact with this data. This will be an iOS and Android mobile application. It will allow users to see not just current data readings from the microgrid, but also query and view ranges of previous data for the microgrid.

– A server to manage the data between the aggregator, database, and mobile app. This will allow the different pieces of the application to communicate between each other and maintain the integrity of the system.

– Technical Documentation of the product and mentioned deliverables. The documentation will not only describe functionality and usability of the product, but also the decisions and processes the team took to create the product and deliverables.

The delivery date for these deliverables are shared among all deliverables as each deliverable will grow along with the others. The functionality of the product as a whole will define the delivery dates. The first date is May 1st, 2021, wherein the product must have functionality that encompasses collecting and displaying solar and battery data from the microgrid. The second delivery date is December 13th, 2021, at which time the product must encompass all requirements in totality.

2 Project Plan

2.1 Task Decomposition

Data Aggregator
– Setup connection between aggregator and microgrid
– Setup connection between aggregator and server
– Create functionality to read data from microgrid
– Create functionality to aggregate data together
– Create functionality to send aggregated data to server / database

Database
– Design database schema
– Setup mysql database
– Setup connection between database and server

Mobile Application
– Find graph library to help display data
– Implement graph library to display data
– Allow users to add search parameters for previous data
– Setup websocket to server to get real time data
– Deploy to Apple App Store
– Deploy to Google Play Store

Server
– Get data from data aggregator to
    – store in the database
    – send real time to open mobile app instances
– Create api endpoints for the mobile application to get necessary data from

– Create websocket to send real time data to mobile application
– Create functionality to archive existing data
– Allow configuration of data collection

2.2 Risks And Risk Management/Mitigation

In any software project, risks are an issue that it is beneficial to attempt to predict and come up with solutions to mitigate. This project is no exception, and in this section we will explore the risks that we foresee might be an issue during development and beyond.

One such risk is of the graphing library that we choose. If it simply does not have the functionality that we require in displaying our data, then we will have no choice but to search for libraries or even develop our own solution for the task. But we must also consider, as for any third party software library, if there exists any vulnerabilities in the code and if the library will be supported throughout the lifetime of the project. For these instances, it might be acceptable to still continue using the library, but new mitigation measures will need to be found. For example, choosing a well-documented or popular library to replace the current library.

A related risk is our cross-platform development framework that we chose: React Native. While we do not expect major issues throughout the development of the project related to this, it is likely that many small issues will arise through the differences between developing for different platforms.

The other big risk for the application is the performance of the data flow throughout the entire application. The time from when the data is collected in the microgrid to the time the data is displayed on the mobile application must be reasonably quick (within 1 minute, and hopefully considerably faster), so if our application is not efficient in moving large amounts of frequent data, this goal will not be achieved. While this issue may not be completely avoidable due to reliance on the network as well as feasibility of the data, we plan to mitigate this issue as much as possible by using well defined and efficient methods to transport the data, as well as reduce and aggregate the data as much as possible in order to increase efficiency.

Another risk for this application is not being able to obtain the data. There could be an issue in acquiring this data, which will set back our progress. To continue moving forward with this application, we will need to create a dummy database. Although it will not be valuable data, the purpose of doing so is to create a placeholder for both testing and operational purposes. We will also be able to design other database objects.

A final risk worth mentioning is the risk associated with having our server use a public IP address. The best way to make this app usable by anyone from anywhere is by storing data from the microgrid into a database run on a server that has its own public IP address. Unfortunately, this means we cannot simply include our server as part of a private network at Iowa State, and we will not be able to use all the security already set up on Iowa State's network. Luckily, Iowa State does have a range of public IP addresses available, and we can assign one of those to our server. What this leaves us with is managing the risk associated with a public IP address. We will have to ensure this server has a proper firewall and virus-scanning capabilities, and we will need to research any other risks we may need to address in order to ensure we can safely operate our server with a public IP address.

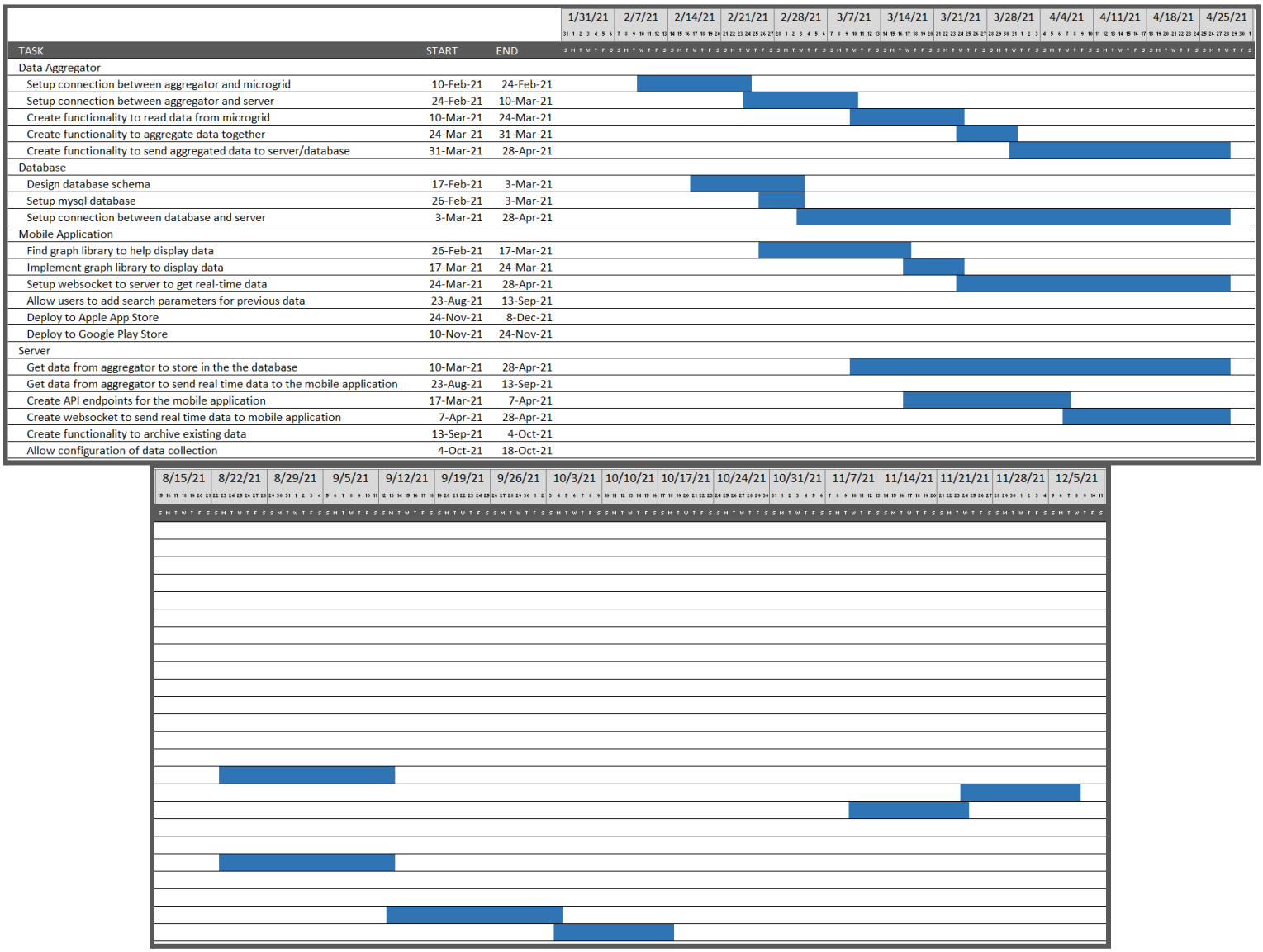
2.3 Project Proposed Milestones, Metrics, and Evaluation Criteria

Given the nature of our project as a mobile application to interact with data from numerous data sources, our milestones will be quantifiable in several different ways. Some of these milestones can be quantified in terms of time, but other milestones will be quantifiable based on feedback and approval from our advisors and clients. We are very fortunate to have 3 advisors and 1 client for this project, and the relevant milestones will require approval from all 4 of these individuals. Our milestones for this project are the following:

1. The mobile application is able to obtain data from the remote desktop.
2. The mobile application is able to display real-time data from the crate
3. The mobile application is able to display archived and average data from the database
4. The mobile application is able to display solar data from a single crate and contains a prototype for displaying battery data.
    - This milestone is provided by the client of this project and will be considered completed on approval from the client and the 3 advisors.
5. The means of storing data from the microgrid can be configured to set the rate at which data is collected, the length of time over which average data is calculated, and the length of time that data is stored.
6. The mobile application is able to display up to one week of data at a time from a given source on the microgrid in the form of a graph.
7. The mobile application is able to display data from multiple different sources on the microgrid.
8. The mobile application has a consistent theme, intuitive user interface, and can display data in such a way that meets the requirements provided by our client.
    - This milestone will be considered completed on approval from the client and the 3 advisors, and it signifies that the project has reached a completed state.

2.4 Project Timeline/Schedule

TASK	START	END
Data Aggregator		
Setup connection between aggregator and microgrid	10-Feb-21	24-Feb-21
Setup connection between aggregator and server	24-Feb-21	10-Mar-21
Create functionality to read data from microgrid	10-Mar-21	24-Mar-21
Create functionality to aggregate data together	24-Mar-21	31-Mar-21
Create functionality to send aggregated data to server/database	31-Mar-21	28-Apr-21
Database		
Design database schema	17-Feb-21	3-Mar-21
Setup mysql database	26-Feb-21	3-Mar-21
Setup connection between database and server	3-Mar-21	28-Apr-21
Mobile Application		
Find graph library to help display data	26-Feb-21	17-Mar-21
Implement graph library to display data	17-Mar-21	24-Mar-21
Setup websocket to server to get real-time data	24-Mar-21	28-Apr-21
Allow users to add search parameters for previous data	23-Aug-21	13-Sep-21
Deploy to Apple App Store	24-Nov-21	8-Dec-21
Deploy to Google Play Store	10-Nov-21	24-Nov-21
Server		
Get data from aggregator to store in the the database	10-Mar-21	28-Apr-21
Get data from aggregator to send real time data to the mobile application	23-Aug-21	13-Sep-21
Create API endpoints for the mobile application	17-Mar-21	7-Apr-21
Create websocket to send real time data to mobile application	7-Apr-21	28-Apr-21
Create functionality to archive existing data	13-Sep-21	4-Oct-21
Allow configuration of data collection	4-Oct-21	18-Oct-21



(Figure 1. Project Timeline/Schedule)

Given our current requirements, this is how we plan to complete the project in two semesters. The main goal of the schedule presented here is to meet our requirement to be able to present a limited amount of data with our frontend application by the end of the spring semester.

2.5 Project Tracking Procedures

The progress of this project will be tracked using tools available in Git. Issue tracking in Git will be used to associate given commits with smaller tasks such as those outlined in 2.1. This issue tracking should be highly flexible in order to include other tasks and bug fixes that may become necessary as development progresses. The labelling service provided by Git should also be used to help associate merges and commits with issues. We will also use the milestone service provided by git to keep track of our progress towards the higher-level milestones outlined in 2.5.

2.6 Personnel Effort Requirements

Task	Estimated Time / Testing Time
Setup connection between aggregator and microgrid	20 hours / 8 hours
Setup connection between aggregator and server	10 hours / 4 hours
Create functionality to read data from microgrid	10 hours / 4 hours
Create functionality to aggregate data together	5 hours / 5 hours
Create functionality to send aggregated data to server / database	15 hours / 6 hours
Design database schema	10 hours / 2 hours
Setup mysql database	5 hours / 2 hours
Setup connection between database and server	2 hours / 0.5 hours
Find graph library to help display data	12 hours / N/A
Implement graph library to display data	6 hours / 4 hours
Setup websocket to server to get real time data	25 hours / 12 hours
Allow users to add search parameters for previous data	15 hours / 5 hours
Deploy to Apple App Store	15 hours / N/A

Deploy to Google Play Store	15 hours / N/A
Get data from data aggregator to store in the database	35 hours / 18 hours
Get data from data aggregator to send in real time to open mobile app instances	15 hours / 10 hours
Create api endpoints for the mobile application to get necessary data from	15 hours / 8 hours
Create websocket to send real time data to mobile application	15 hours / 8 hours
Create functionality to archive existing data	15 hours / 5 hours
Allow configuration of data collection	10 hours / 4 hours

(Table 1. Tasks and the Estimate Time to Complete the Tasks)

For the mobile application to be successfully implemented, our team will have to dedicate time to work on the frameworks for the app. We assigned roles based on experience with various program applications and experience with backend, database, and frontend frameworks. Ideally each team member will work with each task because the team needs to have a general knowledge of how each framework works, so we will work with each other on different tasks depending on what's needed at the moment.

2.7 Other Resource Requirements

We will use React Native open-source mobile application to build the frontend, MySQL to build a database, Spring boot to build the backend, and Virtual Machine running Ubuntu 20.04 LTS to host the database and server.

2.8 Financial Requirements

From our current expectations we do not predict any financial support will be necessary to complete the project.

3 Design

3.1 Previous Work And Literature

As far as the team is aware, there is no such existing application that provides the same functionality as our product. While some of the data sources have their own application for its own data, for example the Tesla Powerwall and Dranetz, these applications again are only for their own data and do not display related components data.

3.2 Design Thinking

One requirement that will largely shape the design of the application is the scalability requirements of the system to include new data sources and crates in the future. This leads to the idea of creating a modular application and architecture such that it is easy to add new pieces to it in the future. Our architecture will be modular by virtue of having the database and frontend being cleanly separated. The application presented to the user should be intuitive to new users and provide information in a manner that is easy to understand. The user should have accessibility to obtain the application and see certain data that the public should be able to see.

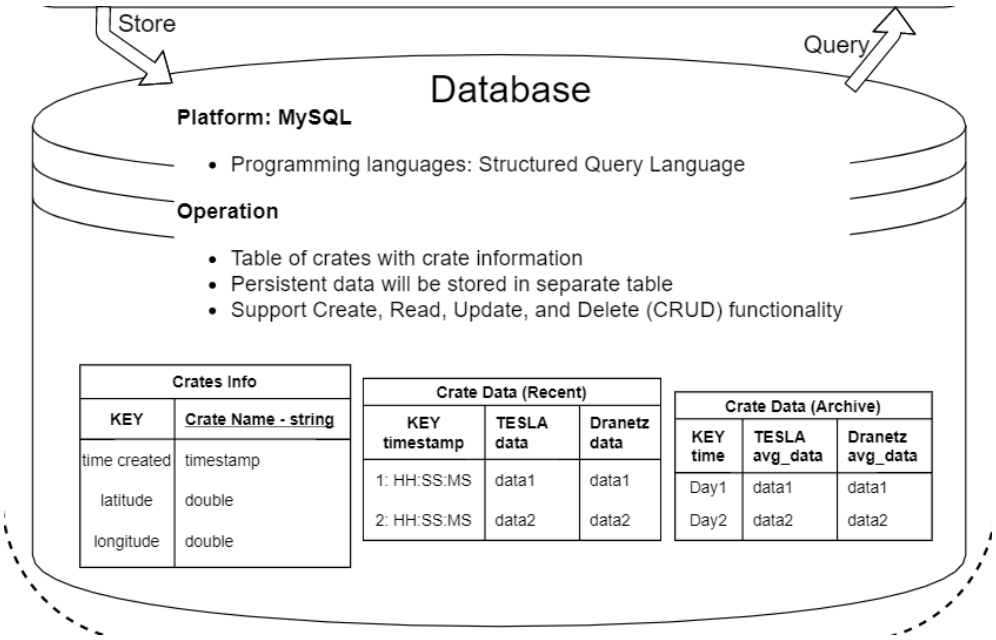
While keeping the aspects that shape our design in mind, we made numerous design choices that will address these needs. We chose a frontend framework that can provide a user-experience typical of applications native to the devices we intend to support. We also have requested a server and chosen backend and database frameworks that can scale well and are easy to modify in the future.

Detail any design thinking driven design “define” aspects that shape your design. Enumerate some of the other design choices that came up in your design thinking “ideate” phase.

3.3 Proposed Design

When proposing a design for a given problem or to accomplish a certain task, there are many considerations that need to be made. The first thing to note is if any work up to this point has been done relevant to the task at hand. This could be work that could be transferred to contribute to this design, or it could be work that may have an effect on how this design needs to be implemented. With this in mind, consideration should then be made of the overall functional and non-functional requirements of the project that are relevant to the task.

Any design that is proposed needs to be capable of meeting these requirements no matter what, and this will help to eliminate any proposed designs that would not work. For this project, the standards we follow will be dependent on the technologies and platforms we end up using. This will influence how we choose what platforms to use, and in the same way these platforms will ultimately dictate the standards we use. For this project, the compliance of these standards will mostly come as courtesy of the platforms we decide to use. We will have to determine what standards we need to comply to ourselves once we select our platforms. Some of the standards and protocols that will likely be relevant to our project are HTTP, TCP/IP, and mobile application design standards (such as ensuring network operations are performed on their own thread).

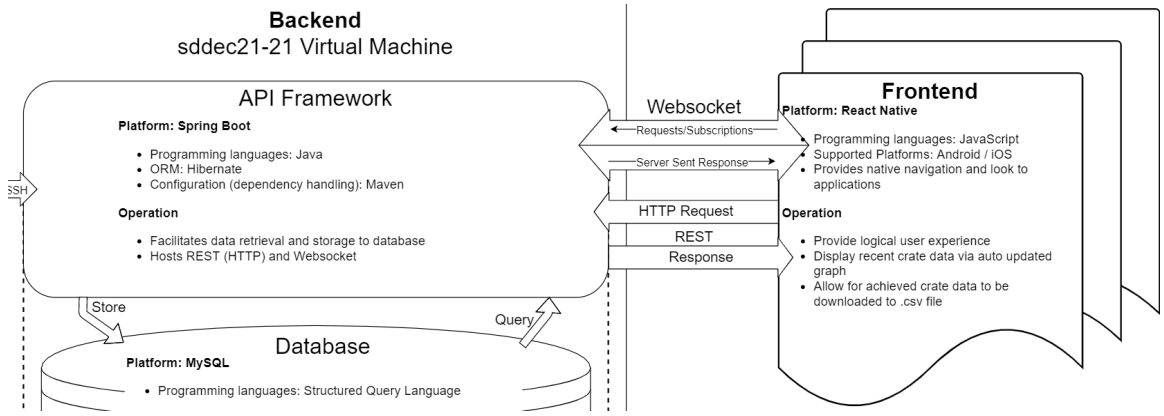


(Figure 2. Database Diagram)

The first component of our project that we must discuss is the database. In the design of our database and tables we must factor in a few different variables. First, we expect frequent writes to the database as we are constantly getting data so we want fast writing but also fast querying on these large datasets that will be created. Second, we need some sort of archiving mechanism for 'old' data. And third we need to account for changes in the amount of data a crate generates.

To do this we propose to create a database with separate tables for crate information, 'recent' crate data, and archived crate data. The crate information table will have information specific to each crate, such as name, location and other general information. The 'recent' crate data table and archived crate data table will look identical. These will have information for the given crate at a given point in time. Each row will be indexed by a timestamp from when the reading was taken, and further columns corresponding to each datapoint that the crate generates.

With this design, fast writing should be no issue as we are using MySQL, which we discuss below. For reading performance, separating the tables into 'recent' data and archived data will increase performance, as one can imagine the size of the tables will be quite large since we are getting data extremely frequently. For archiving data, this archived data table is somewhat clear in its function and how it addresses this issue. For data older than a configurable time it is less important to know what the data looks like at each second and more important to look at long term trends, so having a separate table for averaged data over a configured time amount will not only reduce data size but also increase performance as mentioned. Finally, we must account for changes to the amount of data created by the crate when for example a new instrument is added, which would require a schema change. While with the current design this would lead to occurrences of old rows having null values taking up space, we believe this is a worthwhile tradeoff for the speed and efficiency of a relational database compared to implementing a JSON column or choosing a NoSQL alternative that better handles this. Further, because we are archiving/averaging data, we believe this issue of empty columns is minimized.

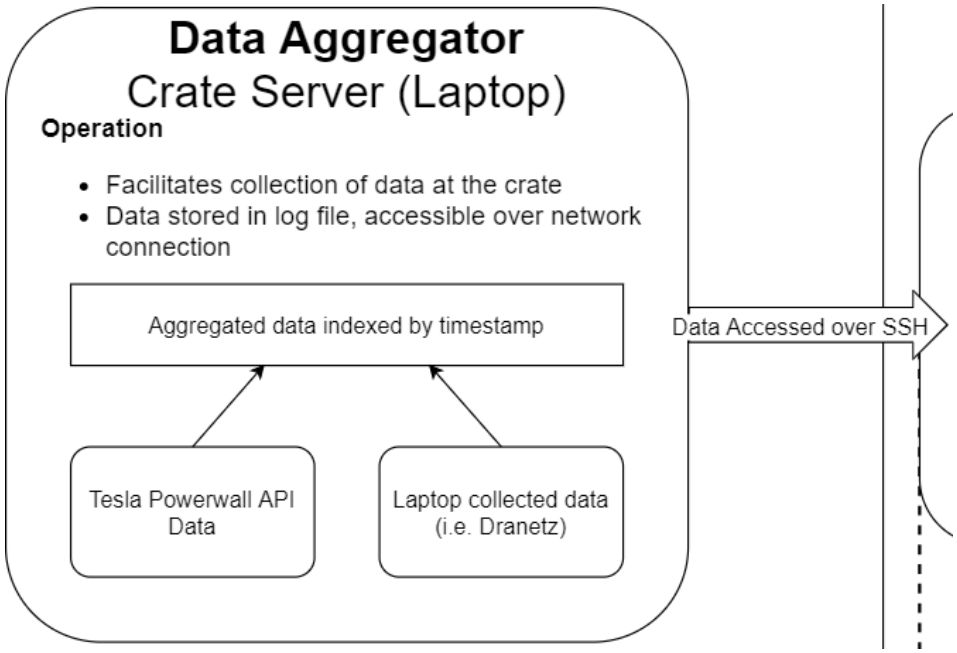


(Figure 3. Frontend plus Backend Diagram)

Next we discuss the backend server and frontend mobile application together. These two components are designed to interact with each other such that the backend server sends the required information to the mobile application to be displayed to the user. For the requirement of our project this comes in two different forms.

The first thing our application has to handle is retrieving and displaying data based on different queries that the user can input. To handle this, we propose to use HTTP requests and responses for the communication between the frontend and backend. The frontend will allow the user to input different querying parameters and then make a request to different REST API's that the backend server gives. The backend server will then receive this response, query the database for the necessary information, and send what is needed back to the mobile application.

The second thing our application must be able to do is retrieve and display real time data from the crate. One option to handle this is simply using HTTP requests and API's as mentioned above. The mobile application can then poll (send an HTTP request for new data) an endpoint on the server every given time interval to get real time data. However, we plan to implement a websocket connection between the frontend and backend. This allows for the frontend and backend to create an open communication channel such that whenever the backend receives new data from the crate, it can immediately send it to the frontend to be displayed to the user.



(Figure 4. Data Aggregator Diagram)

Now we discuss the data aggregator. Unfortunately, in its current state, the crate itself does not have a single point of access to get all of its data. As of now, the data is split in two different locations: A laptop physically inside the crate and a Tesla Powerwall mounted on the crate. The function of this data aggregator is to take the data from the laptop and the Tesla Powerwall, combine it together, and put it into the database.

For getting data from the laptop the team is working with Steve Nystrom to set up a connection to the laptop. Because the crate is in a remote location, all internet access to it is through a cradlepoint, which is a wireless access point and setting up a secure network connection to it is not straightforward, so as mentioned Steve Nystrom is working on setting up that connection. Once that connection is set up, we plan to access the data by simply reading it from location through the connection.

Accessing data from the Tesla Powerwall is a challenge. Through research, the team discovered that Tesla does not publish information about their API's for the Powerwall mobile application online. However, due to a decent sized open source community this information is available, although lacking in some areas. After initial testing of these API's, data consistency issues were discovered making this an unusable option. Currently we plan to access this Powerwall information by using the API's of the device itself and not of Tesla's mobile application.

Once these connections are set up for the data aggregator, the aggregator itself can read the data from each of these sources and join the different information by the timestamps of the different data. Once a piece of data is aggregated together, it can then be sent through a request to the backend server. At this point, the server does two things. First, it writes the data to the database. Secondly, it checks if there exists any open websocket connections to running frontend mobile applications and sends the new data to them.

3.4 Technology Considerations

To highlight the strengths, weaknesses, trade-offs and choose the most effective backend framework out of Spring Boot, Laravel, Django, and Flask for our mobile app development, we focused on six key elements: License, programming language, age & documentation, performance, professional projects, and team experience. Below we will elaborate on importance of each elements:

License

Grants a better understanding of the framework from a financial perspective. Given monetary restrictions on the project, the framework ultimately selected should be free to use recreationally and commercially to avoid any legal concerns. Licenses of selected projects should reflect this consideration.

Programming Language

It is important for the team to select a framework that utilizes languages that are familiar. This will cut back on unnecessary learning curves that come along with learning new programming languages.

Age & Documentation

How long a project has been around is a good indicator of how well a project is supported. This support and open documentation becomes useful when researching different functionalities and setting everything up. It also can signify whether the framework will continue to be supported through the development cycle and after the final application is completed. It would be detrimental to the application's health and security to implement a framework that ends up no longer maintained.

Performance

Our project states requirements regarding response times and frequency of data collection from the crate. The framework our team chooses could have a large impact on the project's ability to meet these requirements. When looking at performance, it is helpful to examine attributes such as multithreading capabilities to process multi requests at once.

Professional Projects

This category is a good reference to whether our application can be implemented using the selected framework. A framework that is used by a larger project that has similar function has a good chance to work in our project.

Team Experience

Akin to our programming language criteria, team experience is to be considered as a framework that our team has experience with will ultimately be easier to implement and time efficient.

Backend Frameworks	License/Cost	Language	Maturity	Performance	Team Exp.
Spring Boot	Apache (Free)	Java	April 2014	Large binaries, multi-thread capable	Experienced
Laravel	MIT (Free)	PHP	June 2011	Slow for large projects	None
Django	BSD 3-Clause (Free)	Python	July 2005	single thread	None
Flask	BSD 3-Clause (Free)	Python	April 2010	Limited concurrent request support	None

(Table 2. Comparing Backend Frameworks)

Spring Boot Framework:

- License: Apache License (v2.0)
- Programming Language: Java
- Age & Documentation: Released in April of 2014, and official website contains various example projects and helpful guides
- Performance: Autoconfiguration may add unnecessary dependencies making binaries larger than necessary. In addition, it can handle multiple requests.
- Professional Projects: Inuit & Zalando
- Team Experience: Software and computer engineers have experience through COM S 309

Laravel Framework:

- License: MIT License
- Programming Language: PHP
- Age & Documentation: Released in June of 2011, and official website contains documentation
- Performance: Generally slower for larger projects
- Professional Projects: 9gag & Kmong
- Team Experience: Unfamiliar

Django Framework:

- License: BSD 3-Clause
- Programming Language: Python
- Age & Documentation: Released in July of 2005, and official website contains documentation, tutories, topic guides, and installation help
- Performance: Performs well for large projects, feature-heavy, which may feel bloated for large projects, and can only handle one request at a time.

- Professional Projects: Pinterest, Instagram, and Robinhood
- Team Experience: Unfamiliar

Flask Framework:

- License: BSD 3-Clause
- Programming Language: Python
- Age & Documentation: Released in April of 2010
- Performance: It is suitable for smaller projects and doesn't handle concurrent requests as well as others.
- Professional Projects: Netflix, Reddit, and Lyft
- Team Experience: Unfamiliar

Frontend Frameworks	License/Cost	Language	Maturity	Relevant Platforms
Qt	Paid	C/C++, JS, HTML, QML	1991	Android, iOS
React Native	MIT (Free)	JS	2015	Android, iOS
Flutter	New BSD (Free)	C/C++, Dart, Skia	2017	Android, iOS

(Table 3. Comparing Frontend Frameworks)

QT Framework:

- License: Free open source license available, free educational license available, paid commercial license available, and separate distribution licensing available.
- Supported Platforms: Windows, macOS, Linux, Android, iOS
- Programming Languages: C/C++ for application framework and JavaScript, HTML, and QML for UI
- Maturity: It was first written in 1991 and had a long history of public bug reports and forums available. Also, there is extensive documentation on all available QT classes.
- Other Notable Aspects: Multithreading support

React Native:

- License: Free MIT License
- Supported Platforms: Android & iOS
- Programming Languages: JavaScript
- Maturity: It was founded in 2013 but released in 2015. It is supported by Facebook and also receives contributions from individuals and companies.
- Other Notable Aspects: It aims for a truly native feel on apps and does not support multithreading.

Flutter:

- License: Free New BSD License

- Supported Platforms: Android & iOS
- Programming Languages: C/C++, Dart, and Skia for UI
- Maturity: It was founded in 2017 and supported by Google. Because it's recent and new, there is not much support found online.
- Other Notable Aspects: It does not support true multithreading.

Additionally, we chose to pivot towards license availability and pricing, supported platforms, programming languages, maturity, and other notable aspects for the frontend frameworks. Given financial limitations on the project, selection based on free licensing was key. Maturity of the project was also a major consideration because the project will need to be supported into the future.

Database Frameworks	Storage	Schema	Team Experience
SQL	SQL Tables	Schema defined	Experienced
MongoDB	JSON Structure	No schema	None

(Table 4. Comparing Database Frameworks)

SQL (MySQL & PostgreSQL):

- Data Storage: Data is stored in tables.
- Schema: The schema defines the database structure, meaning all rows must have the same structure.
- Team Experience: Familiar
- Other Notable Aspects: It can use JSON type to handle adding new data source components, but it will require more space because JSON will be stored as a string.

MongoDB:

- Data Storage: Data is stored in a JSON-like structure.
- Schema: There is no schema, and it is much easier and efficient to change.
- Team Experience: Each team member is unfamiliar with it, and there are poor online reviews about it.
- Other Notable Aspects: MongoDB query language

Finally, we chose to center the database to compare data storage, schema, team experience, and other notable aspects.

3.5 Design Analysis

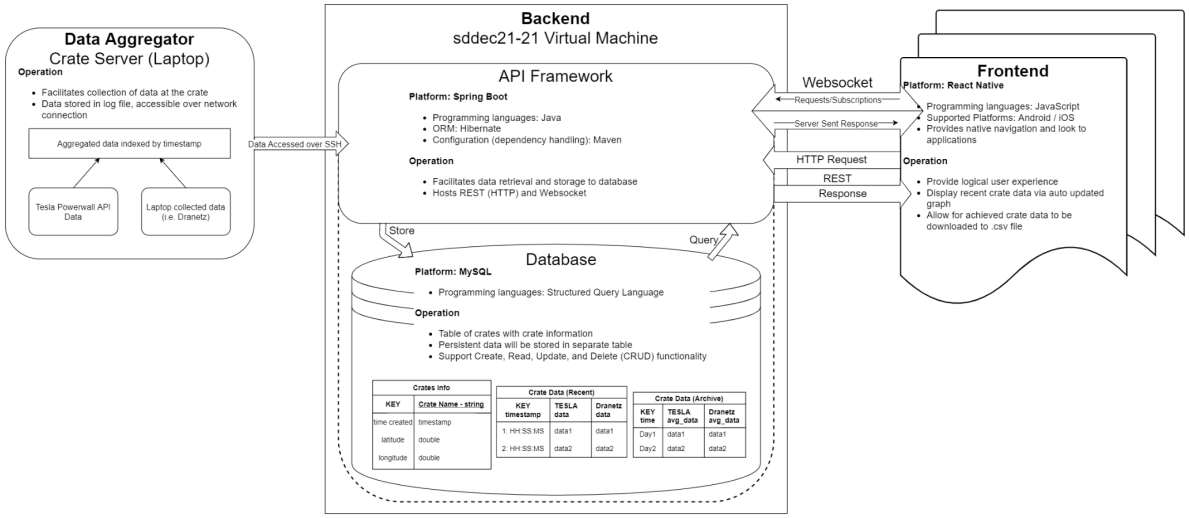
In the scenario where we are facing a problem with our design we will follow the process listed here. We will first look back at what we have tried up to this point and try to understand where it failed. Was it a shortcoming in the framework? Did we just not have the appropriate knowledge yet to use the tools available to us? Could any of our methods we've tried thus far actually accomplish our task? After asking questions like this, we then need to explore our options in addressing this problem while keeping several things in mind. For us, the higher level requirements for this project are simple, and they mainly consist of timing requirements for accessing and managing data, and what some high level features should be a part of this project upon completion. We should also remain conscious of the standards we may be responsible for meeting as discussed in section 3.3.

With these things in mind, we can then start looking for solutions to the problem at hand. Some other things to consider during this process are what repercussions a given solution may have on other aspects of the project or if a proposed solution might constitute a change in requirements. Both of these scenarios should be avoided wherever possible. Finally, one last thing to consider in this process is if the description of the design needs more detail in order to guide this decision and prevent problems in the future.

3.6 Development Process

We will be taking a waterfall approach to our development phases. We had decided this approach because previous experience among team members working in this type of system. We also believed that due to the smaller scope of the project, waterfall will make it easy to stay concise and on task when it comes to scheduling a timeline.

3.7 Design Plan



(Figure 5. Design Plan: How We Are Creating This Application)

Figure 2 (shown above) provides a broad technical overview of the modules in our project and how they interact with each other. Each of these modules have their own requirements, and there are also requirements for all of the modules working together as outlined in section 1.4 of this document. In general, the frontend application naturally depends on a functioning server, the server naturally depends on a functioning database, and the database depends on functioning data aggregation. Inversely to that, a functioning data aggregator will be valuable in ensuring a functioning database, a functioning server will be very valuable in ensuring a functioning database, and a functioning frontend application will be valuable in ensuring a functioning server. As part of our design plan, it is important that we plan for progress to be made in certain modules so that progress in other dependent modules is not hindered.

The first part of our design plan will be to determine the frameworks, libraries, software, and even hardware we will use to construct all of the modules in our design. For this part of our design, it is extremely important that we remain conscious of all of the requirements established by our advisor. Many of the technology considerations we have in starting this project are outlined in detail in

section 3.4 of this document. For the most part, we have already completed this portion of our design plan. We have selected a computer to act as our server that we believe will be able to store all of the information we expect from multiple crates in the future. We should be able to write a backend application then run on this server which will then be accessible via the frontend application running on a mobile device. For our database framework, we have opted to use SQL. For our backend program, we have decided Spring Boot was our best option. Finally, we have also decided React Native was the best available option for us to construct our frontend application. All of these decisions were made based on our belief that we will be able to fulfill all of the requirements set out by our advisors using these tools.

The next part of our design plan is where we find ourselves at the time this document is being written. We need to begin utilizing the tools we have decided to use to start constructing the different modules of our project. For this part of the plan, we must remain conscious of the requirements given to us as a team, but we will likely not fulfill many of these requirements by the end of this part. Rather, the goal is to establish a foundation for the entirety of our project which can then be expanded on to meet the requirements for this project. In the context of each module, these means setting up the initial framework for each module and establishing our means for connecting each module. Naturally, in order to establish these links in a way that will allow us to meet our requirements regarding data transfer and latency, parts of the data aggregator will be established first, then certain parts of the database will need to be established, then the server, and then the frontend. This general progression from creating the data aggregator to creating the frontend can be observed by the timeline we laid out in section 2.4 of this document.

Following this part of our design plan, we should now be able to have a better idea of whether or not the frameworks, libraries, software, and hardware we have selected for this project will work as we intended and allow us to meet the requirements for this project. At this point we should have made any major changes needed to our selection of these technologies, and only minor changes should be needed moving forward in how we utilize these technologies. The next part of our design plan will be to construct each of our individual modules using the foundation we have established.

This portion of the design plan will be focused on fulfilling the requirements for this project. We will work as a team on our respective modules to meet requirements specific to those modules, and we will have to work with each other across modules to build on the foundation for the links we establish and meet our requirements for those links. Given the nature of this part of our design plan, meeting frequently as a team and communicating will be essential as with the previous parts of our plan. Documentation regarding how we construct each part of our projects and meet our requirements will also be important throughout this portion of the project so that we can better explain this design to our stakeholders and possibly pass this design off to another team in the future.

Finally, the last part of our design plan will involve finalizing everything we worked on to meet our requirements and ensure that it performs as best as possible. This will involve bug fixing and optimization of what we have already created. Furthermore, we should strive to go beyond our explicitly written requirements in order to make this application the best it can be for our users. This will involve things like focusing on improving our UI design and presentation of the app to the user, adding things to our database and server that can benefit our users, and overall improving our user experience.

4 Testing

Like any software project, testing is extremely important to determine not just whether a piece of code functions correctly or if groups of code function together correctly, but also if the project meets the requirements set out by the client. In this section, we will discuss our approach and methods used for testing.

4.1 Unit Testing

For unit testing, we plan to have testing coverage for every piece of software that we write. What this looks like for our backend server written in Java using the Spring framework is having at least one unit test for every function that we write to achieve total testing coverage of the codebase. To do this testing we use the JUnit and Mockito testing frameworks to create, define, and assist in writing and verifying the unit tests. Similarly to test our mobile application written in React Native, we will use a testing framework called Appium.

4.2 Interface Testing

The interface testing that we do mostly consists of testing the api endpoints for our backend application. To do this we can once again use JUnit and Mockito test frameworks, specifically the modules associated with REST endpoint testing.

4.3 Acceptance Testing

To demonstrate that our design requirements are being met, the team first conducts unit and interface testing described in the previous sections. If these tests are not passed, additional work is done to correct the errors until the tests pass. Once these tests are completed, the team will manually test the product to see if it meets the requirements given by our customers. Once the team is satisfied that the requirements have been met, or have made changes to the product until the requirements have been met, the client is then given access to use and personally test the product to see if it meets their expectations. The client then gives feedback to the team, in which case the feedback is implemented and the testing process is repeated, or the client accepts the product and the phase is complete.

One such component of the acceptance testing that is important to mention is the end-to-end latency testing through some form of integration testing that we must complete in order to meet requirements for data latency throughout the application. This can be achieved through a combination of automated testing as well as manual testing for acceptance with our clients.

4.4 Results

Through current testing and design, we have found major problems with using the Tesla Powerwall mobile application's APIs to collect data for the crate's powerwall as mentioned in the proposed design. We found that we cannot rely on data consistency in accessing this data as successive calls to the API do not always return data in any particular order and further this data is not recent enough, often being 20 or 30 minutes behind the current time which is unacceptable for our client's requirements. From this issue, we plan to switch to interacting with the crate's Tesla Powerwall unit directly, for which we need to establish a network connection to.

This issue with the Tesla Powerwall and how the project adapts and handles this issue corresponds to how the team plans to address any and all testing issues in the future. If, through testing we find that a design of a component of the application is not up to the standards of the team and requirements of the client, we first discuss the problem. Through this discussion we determine first if it is not the approach but the implementation that is at fault. If it is the implementation, this can be addressed by a team member with no further discussion. However, if the approach is insufficient, we then ask if the solution approach can be modified in any ways to perform to standards. If it cannot we then discuss and look for alternative solutions to our problem and go through initial testing stages to see if we believe the new idea is viable. Through these steps, the team hopes to be able to address and handle any issues with the application that arise throughout the development process in an efficient and timely manner.

5 Implementation

The implementation plan is to create an app that shows operating data from an off-grid microgrid operating system. Our team has been split into two sub-teams focusing on the backend and frontend of the project respectively. So far, we can send responses and requests between the frontend and backend. The backend team will have the tasks of creating a communication link between the components on the microgrid and the database, allow collected data to be archived, having data received and sent out to between components in a timely manner, making data storage configurable, and all aspects of data storage modularized. The frontend team will be tasked with being able to request data from the server, visualize data in a logical and enjoyable interface for the consumer, and to create a way to obtain archived data.

6 Closing Material

6.1 Conclusion

Thus far in our project, we have conducted extensive research to determine the technology we intend to use. This includes the frameworks and libraries for our database, server, and frontend application. It also includes the platforms we will support for our frontend application and the hardware we will use for our server. We decided on these things based on the requirements given to us for this project, and it is important that we never lose sight of those requirements through every step of this project.

Where we are now is in the foundational development phase of our design plan. At this point, we have made a foundation for our database, server, and frontend application. What remains for this step is to create the connections between each of these modules in such a way that will allow us to meet our requirements for this app as a whole. We have learned a lot through this part of our project, and we have been able to more precisely determine how we will use the frameworks we selected and how well they will work with what we have planned. Our goal by the end of this step is to have a working version of our application that is able to grab real data from our database. This version of the app will likely meet very few of the requirements for this project, but it will serve as a

proof of concept for this project and show that we can build off this to meet the requirements set before us.

Once we have laid this foundation, we will then focus on meeting our requirements. This is the part of our process where we should start taking into consideration goals for things like how fast we deliver data to the user from the moment it is obtained or how the user is ultimately able to visualize this data. Our team members will focus on working on their respective modules, but we must also stay conscious of each other's work throughout this part of development so that we can work towards meeting the requirements that encompass this project as whole. We should ensure that we are not hindering progress in other modules by not completing a task within our respective modules, and we should strive to build up our means of linking these modules. Communication has been invaluable to us thus far, and it will continue to be essential through this portion of our project. The best way to accomplish what we have outlined for this portion of the project will be through extensive communication and frequent meetings where we discuss our progress, needs, and plans.

Based on our work thus far and what we have determined will work for this project, we believe this will be the best way to move forward. Keeping everyone on our team informed is important for making sure we remain conscious of the goals of the project as a whole. Furthermore, having team members focus most of their time on working on an assigned module has begun to accelerate our development as these team members become familiar with the intricacies of their modules and other team members are not required to have to learn such details.

As we draw towards meeting all of our requirements for our project, our last goals are to spend time perfecting our initial set of requirements and going beyond that to create a great user experience for our project. This will involve bug fixing, team member reviews of code, UI improvements, adding new data in our database that can be accessed by our server, and much more. This step is extremely important to us as a team as we are not content with simply finishing a project that meets a set of requirements. We want to create something that people will want to use in the future that we can be proud of. We believe that with the details we have outlined here in this design document and the work we have done thus far, we will be able to do just that.

6.2 References

"Designing For Mobile." *Digital Design Standards*, digitaldesignstandards.com/standard-category/mobile/.

Royce, Winston. "Managing the Development of Large Software Systems (1970)." *Ideas That Created the Future*, 2021, pp. 321–332., doi:10.7551/mitpress/12274.003.0035.

Xuefen Fang, "Using a coding standard to improve program quality," Proceedings Second Asia-Pacific Conference on Quality Software, 2001, pp. 73-78, doi: 10.1109/APAQS.2001.990004.