Project Name:
Software Architecture
version 1.0

The Juicerz
Computer Science Department
California Polytechnic State University
San Luis Obispo, CA USA

November 6, 2019

Contents

Credits

Name	Date	Role	Version
Joshua Boe		Activity and Sequence Diagram	1.0
Lauren Hibbs			
Casey Daly			
Pranathi Guntupalli			
Hannah Kwan			

Revision History

Name	Date	Reason for Changes	Version
Team	November 6, 2019	Initial version	1.0

1 Introduction

This document contains the details of the software architecture of the NeoCharge application. In the following sections, there is an overview of the architecture, then several diagrams which show the functionality and design of individual use cases.

Further details of the use cases mentioned in this document can be referenced in the Software Requirements Specification.

2 Problem Description

3 Solution

3.1 Overview

3.1.1 System Overview - Dataflow Diagram (Lauren Hibbs)

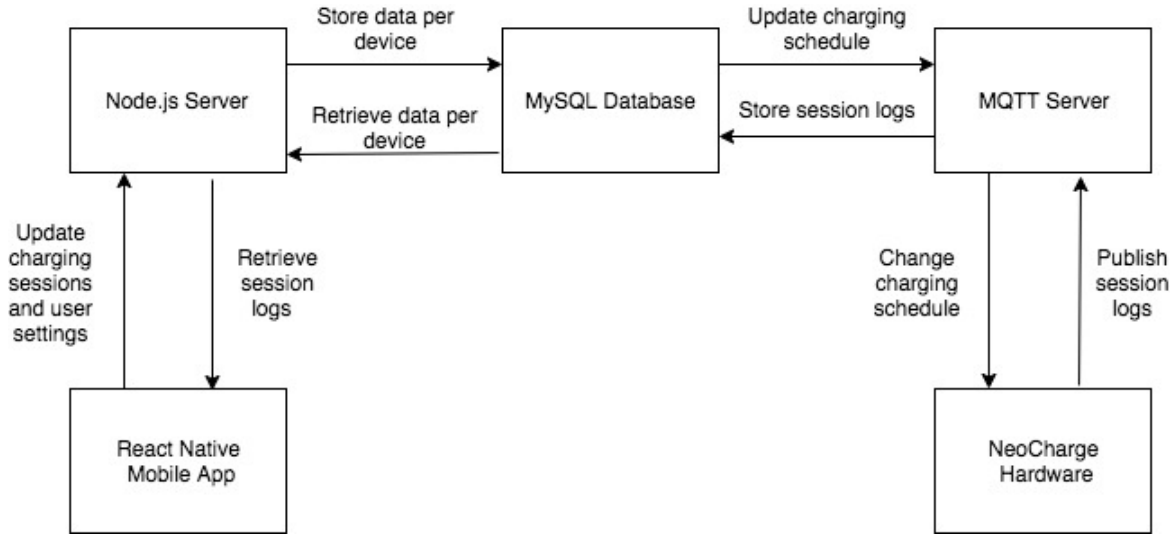

Figure 1: Dataflow Diagram

This dataflow diagram provides an overview of how data will be sent between different components of the system.

3.2 Components

3.2.1 System Components - Deployment Diagram (Lauren Hibbs)

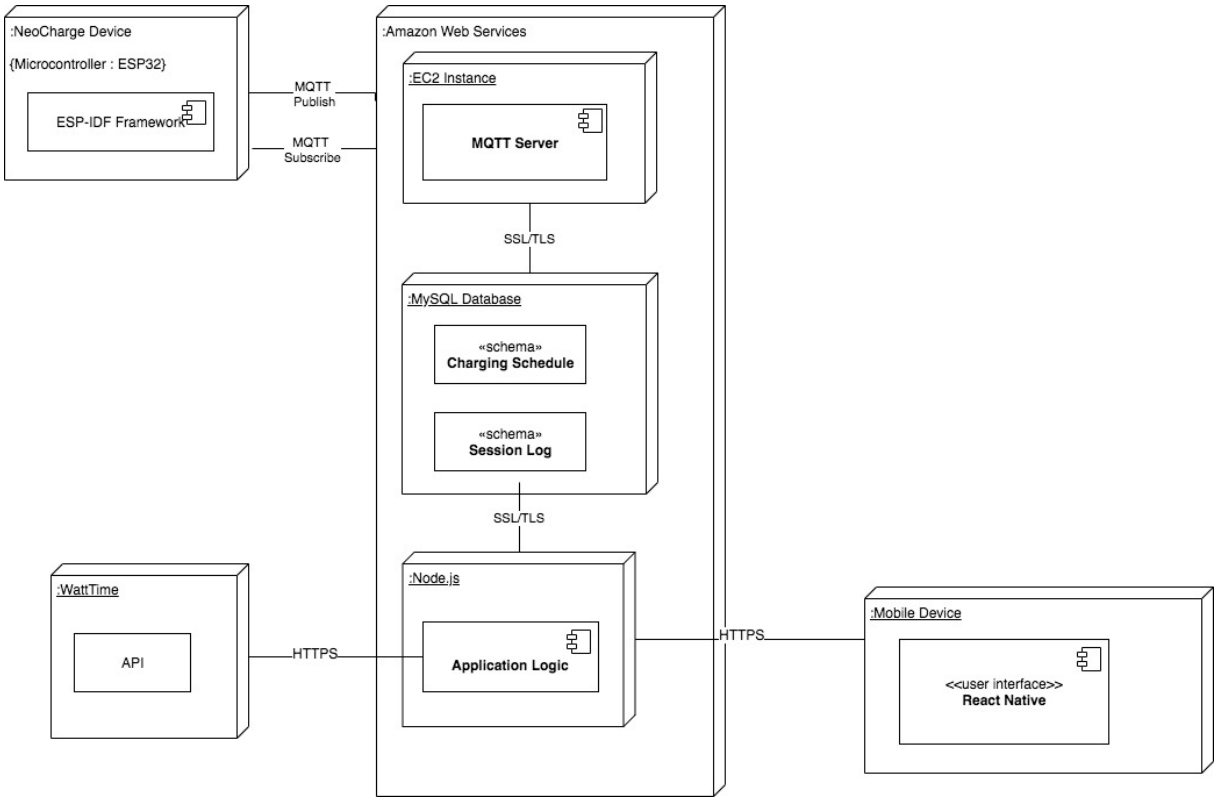

Figure 2: Deployment Diagram

This deployment diagram describes how the components of the system work together.

3.2.2 React Native Frontend

3.2.3 Node.js Server

The Node.js server will primarily function to manage requests to and from the database. This includes authenticating the user is only able to access database records they have permission to, and that the data being put to the server is valid. It will also expose an API for the application to communicate with, transforming the data received from the database into a format easily useable by the app. The Node.js server will most likely be hosted using AWS. A javascript framework on top of Node.js like Express would provide extra functionality but may also be unnecessary given how lightweight our backend is. An alternative to this method is using Node.js with a serverless solution like AWS Lambda. This can be done easily using the 'serverless' package with Node.js. The benefits to this approach are that other parts of the implementation are already within the

AWS ecosystem, and hosting our own webserver may be unnecessary given there is little backend logic in our system.

3.2.4 MySQL Database

The SQL Database will be used to store the user's account information, charging sessions and logs per NeoCharge device. The exact schema for the data is still being decided. Logs will be written after each charging session and will be stored by device id and date. One charging session will be stored per device. The database is written to by both the mobile application and the NeoCharge device server. The server on the mobile application side will be able to update the charge schedule and user settings and be able to read device logs. The MQTT server on the NeoCharge device side will be able to read the charge schedule and update device logs.

3.2.5 MQTT Server

The MQTT Server is an AWS EC2 instance that conforms to the MQTT protocol. MQTT is a communication protocol primarily used for IOT devices. This model relies on a publisher - subscriber architecture where each device is a publishes to the topic 'logs'. The MQTT server acts a listener for all these devices and translates the information received into messages to put the information to the SQL database.

3.3 Design

[This is where the meat of the design lives. For each component, there should be a subsection describing the design of that component in as much detail as you want to provide in a high-level design. There should also be a subsection that describes how the components work together. This section should include class, sequence, and collaboration diagrams as appropriate.]

3.3.1 Deployment Diagram (Lauren Hibbs)

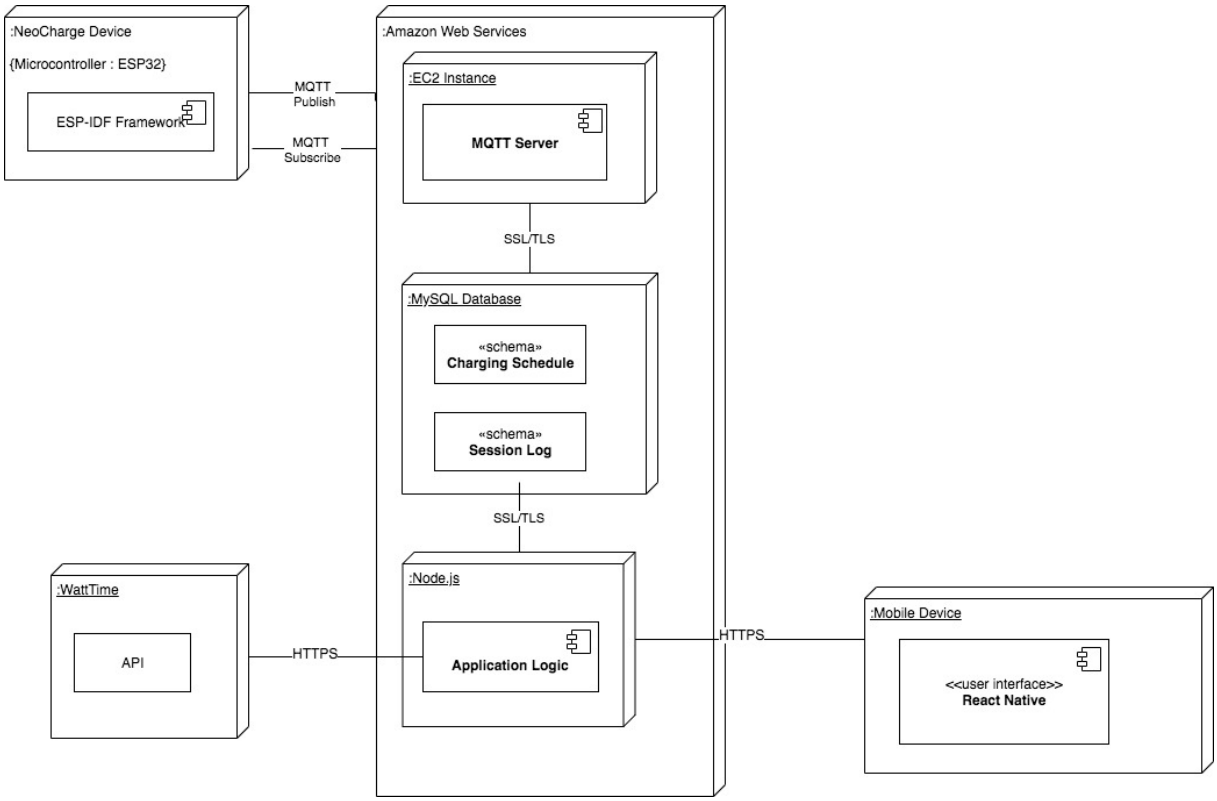

Figure 3: Use Case

3.3.2 Data Flow Diagram (Lauren Hibbs)

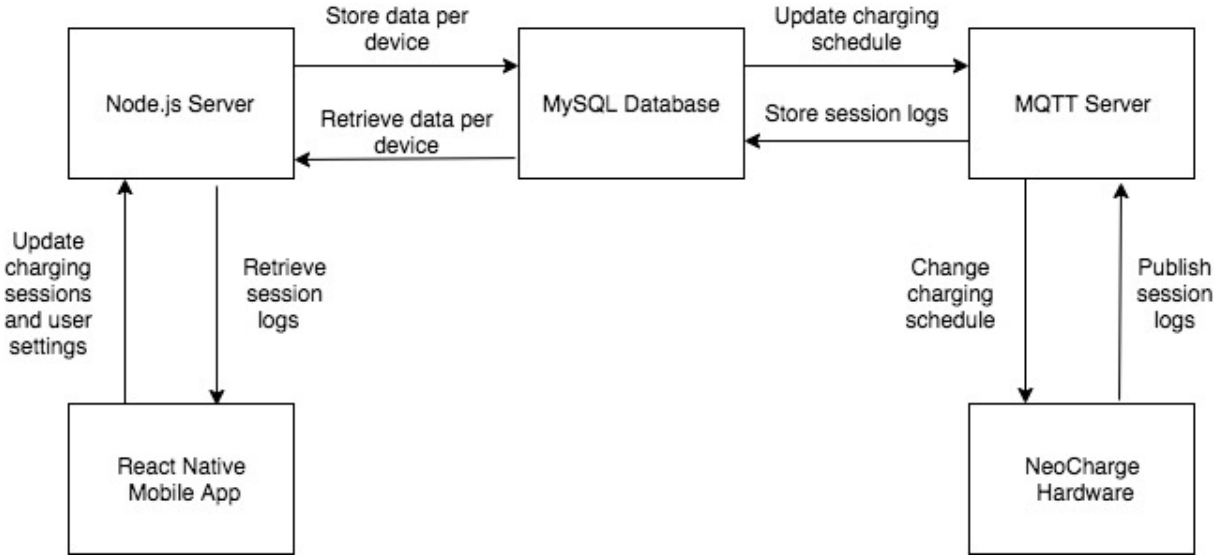

Figure 4: Use Case

3.3.3 Use Case Diagram (Pranathi Guntupalli)

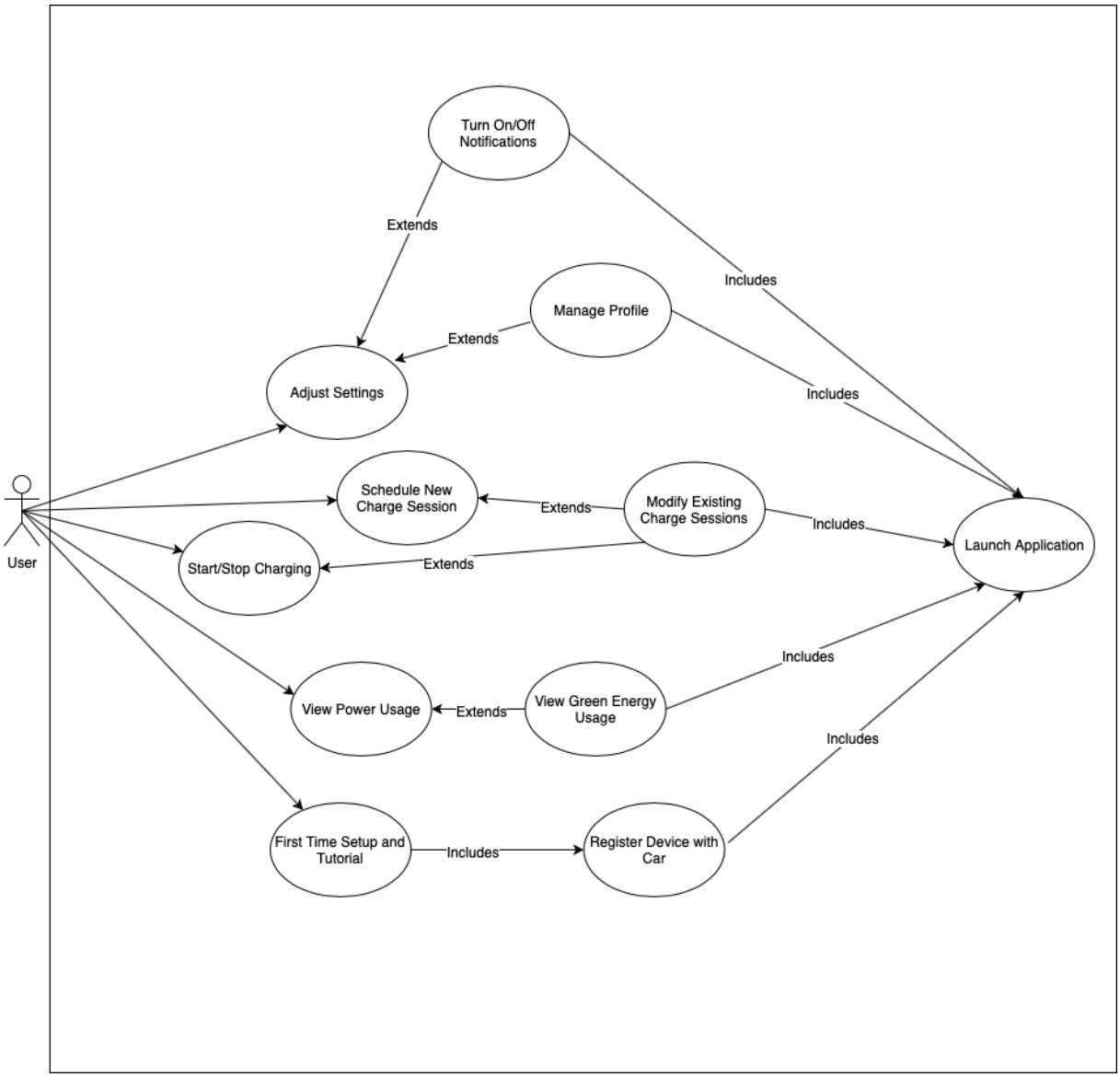

Figure 5: Use Case

This diagram is an overview of all the features the user can utilize when using the NeoCharge Application.

3.3.4 First Time Setup - Activity Diagram

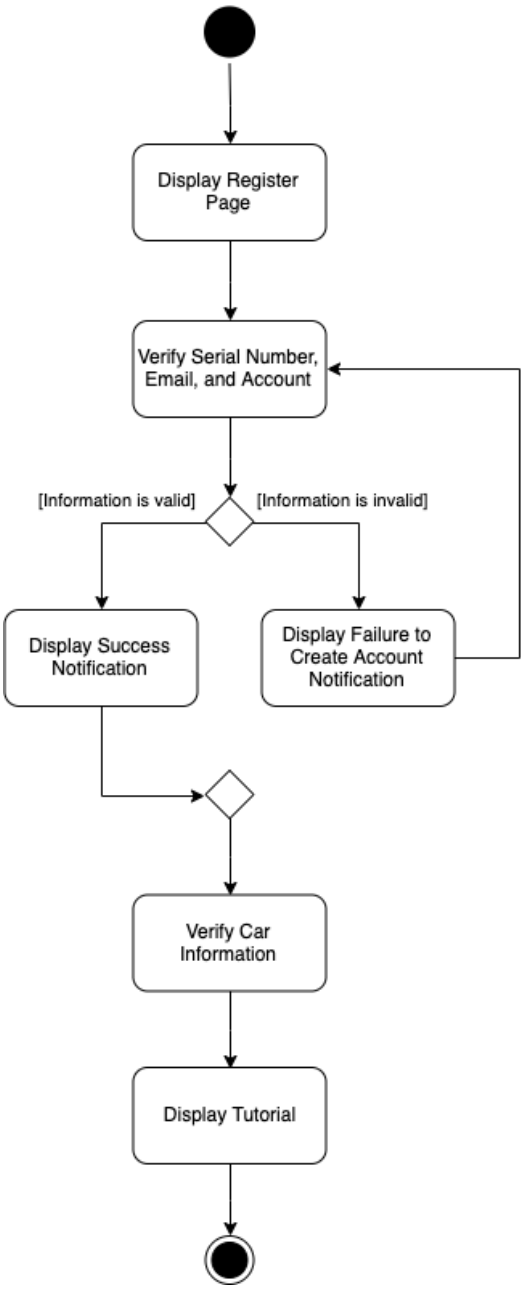

Figure 6: First Time Setup

This diagram describes the use case of a first time setup on the application. One main part of this use case is the verification of user-inputted information.

3.3.5 Schedule a Charge Session - Activity Diagram (Pranathi Guntupalli)

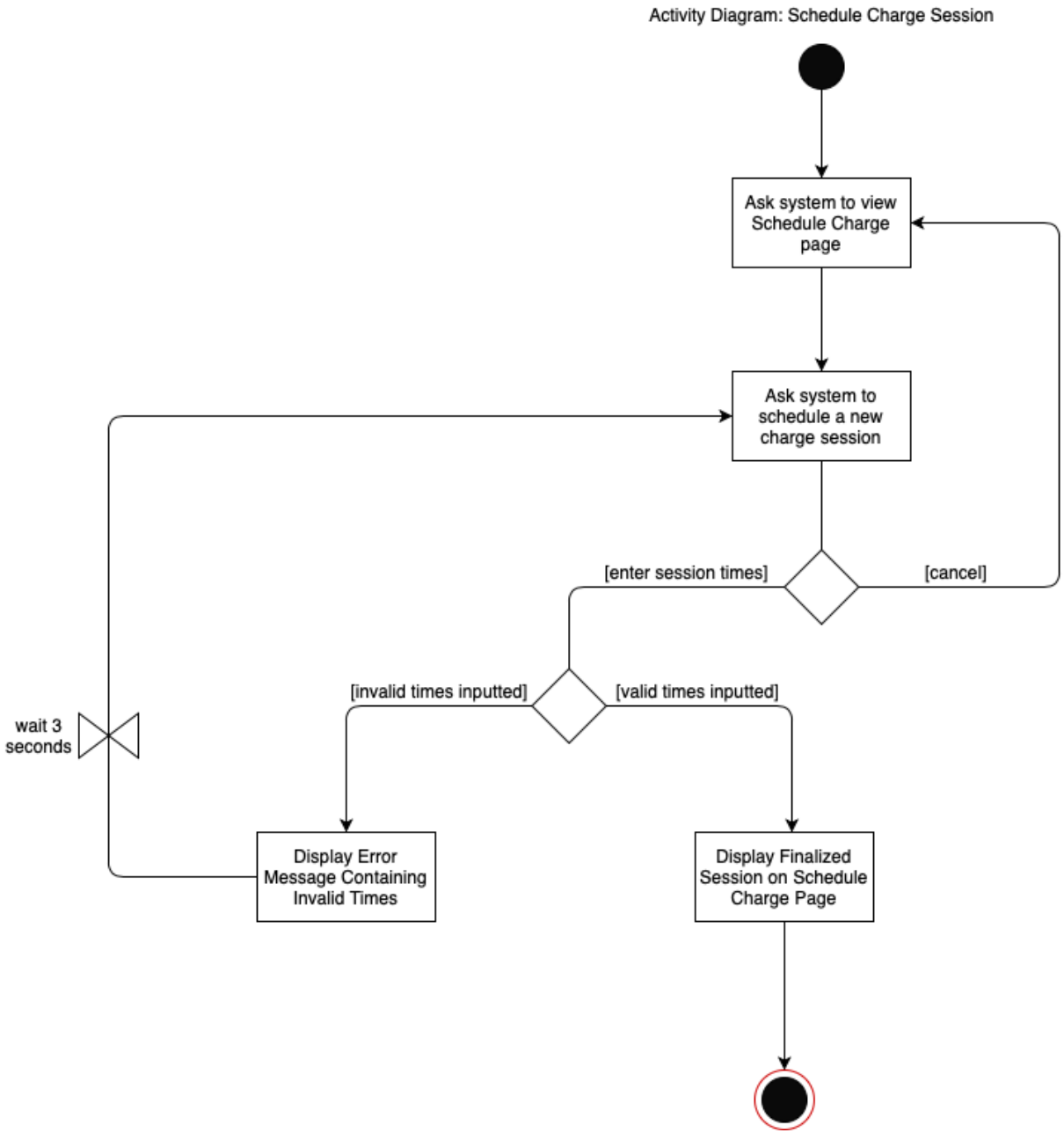

Figure 7: Schedule a Charge Session

This diagram describes the use case of a user manually scheudling a charge session. A user accesses the NeoCharge application from their mobile device. The user then navigates to the Schedule Charge page in order to schedule a charge session by inputting start and

end times and the days of the week the charge should take place. The Schedule Charge page will then update to display the newly scheduled session.

3.3.6 Receiving Notifications - Activity Diagram

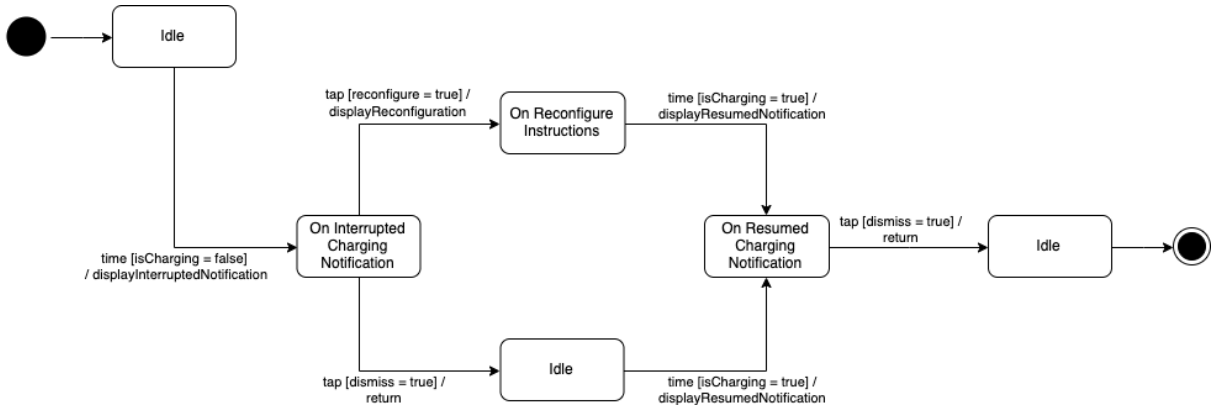

Figure 8: Receiving Notifications

This is a diagram to show the states of the system when receiving a notification. When the system receives the signal that the vehicle is not charging, a notification will pop up in the application. The user can either make sure their device is configured correctly, or dismiss the notification. When charging has resumed, the user will see a second notification and the application will return to an idle state.

3.3.7 Viewing Power Usage - Activity Diagram (Josh Boe)

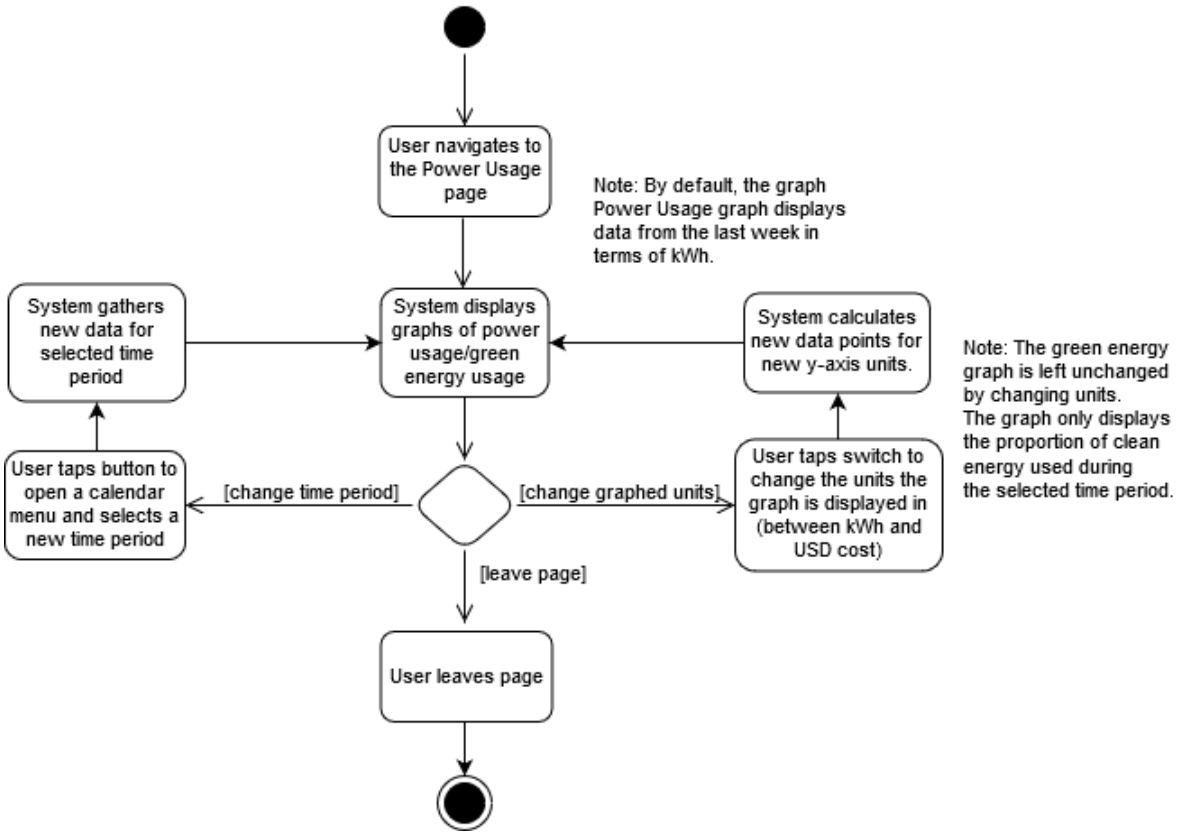

Figure 9: Viewing Power Usage

The above diagram showcases the behavior of the system when the user would like to view their past energy usage. When a user navigates to the "Power Usage" page, the system will display graphical representations of their power and clean energy uses for each day within a selected time period. By default, the power usage graph will display in units of kWh on the y-axis, and both graphs will display the last 7 days as their x-axis.

The user has 3 choices: change the time period, change the units, or leave the page.

1. The user can change the time period displayed in both graphs by tapping a "Change Time Period" button. The system will then display a calendar menu, allowing the user to select the desired time period they would like displayed. The system will then display new graphs that include the specified days.
2. The user can change the y-axis units of the power usage graph by tapping a "units" switch. The switch will alternate the units the graph displays between kWh or cost in USD. In response to a tap, the system will calculate new data points and display a new power usage graph that uses the new unit type.

3. At any point, the user can leave the Power Usage page by either exiting the app or navigating to a new page.

3.3.8 Adjust Settings - Sequence Diagram (Josh Boe)

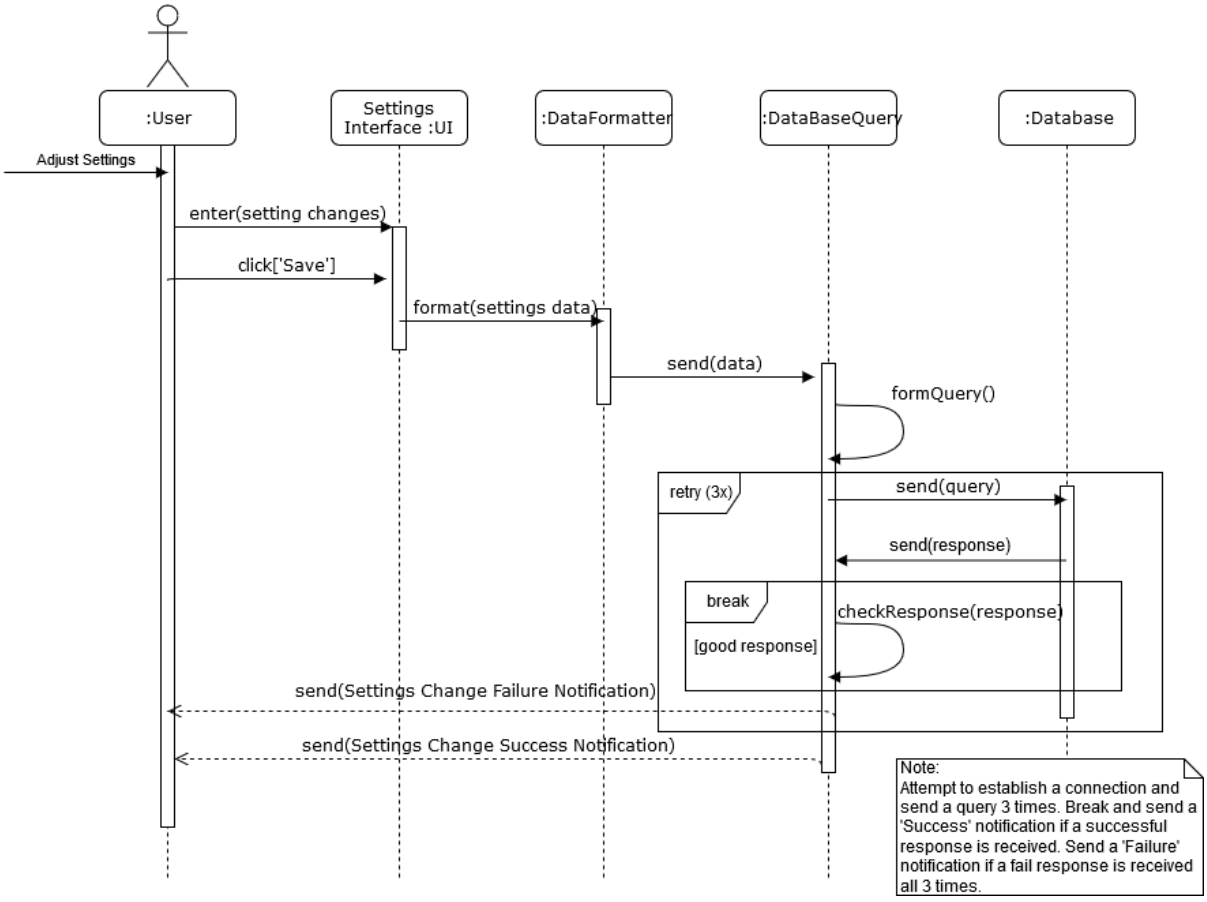

Figure 10: Adjust Settings

The above sequence diagram displays interactions between various objects and parts of the system when a user adjusts the settings within the app. The sequence begins with the user entering the changes that they would like to be made on the Settings page of the UI. After the user clicks "Save," the UI sends a snapshot of the changed settings to a DataFormatter that will format the data into an appropriate format to be sent to the database. The DataFormatter will then send the data to a DataBaseQuery object that will perform the task of sending the data to the database and awaiting a response. After receiving a response, the DataBaseQuery will check if the response signals either a query success or failure. If the query is a success, the user will receive a notification indicating that their settings changes were successful. On a failure, the query will attempt twice

more to send the data to the database. If after 3 failures the data is still not able to reach the database, the user will receive a notification indicating that their settings changes failed to be made and that they may try again later.

3.3.9 Entity-Relationship Diagram (Casey Daly)

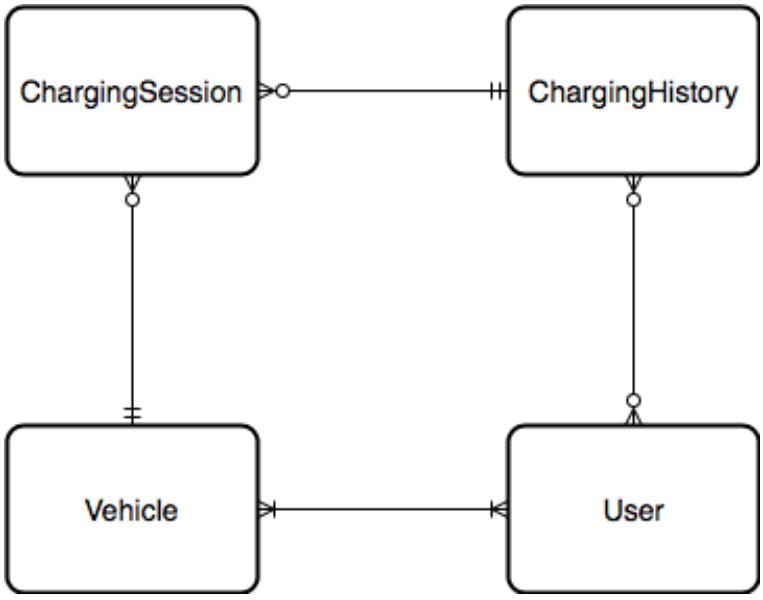

Figure 11: Entity-Relationship Diagram

The entity-relationship diagram above shows the planned relationship between entities in the database. Each ChargingSession has one and only one ChargingHistory, and each ChargingHistory has zero to many ChargingSessions. Each ChargingSession has one and only one Vehicle, and each Vehicle has zero to many ChargingSessions. Each Vehicle has one to many Users, and each User has one to many Vehicles. Each User has zero to many ChargingHistory's, and each ChargingHistory has zero to many Users.

3.3.10 Class Diagram (Casey Daly)

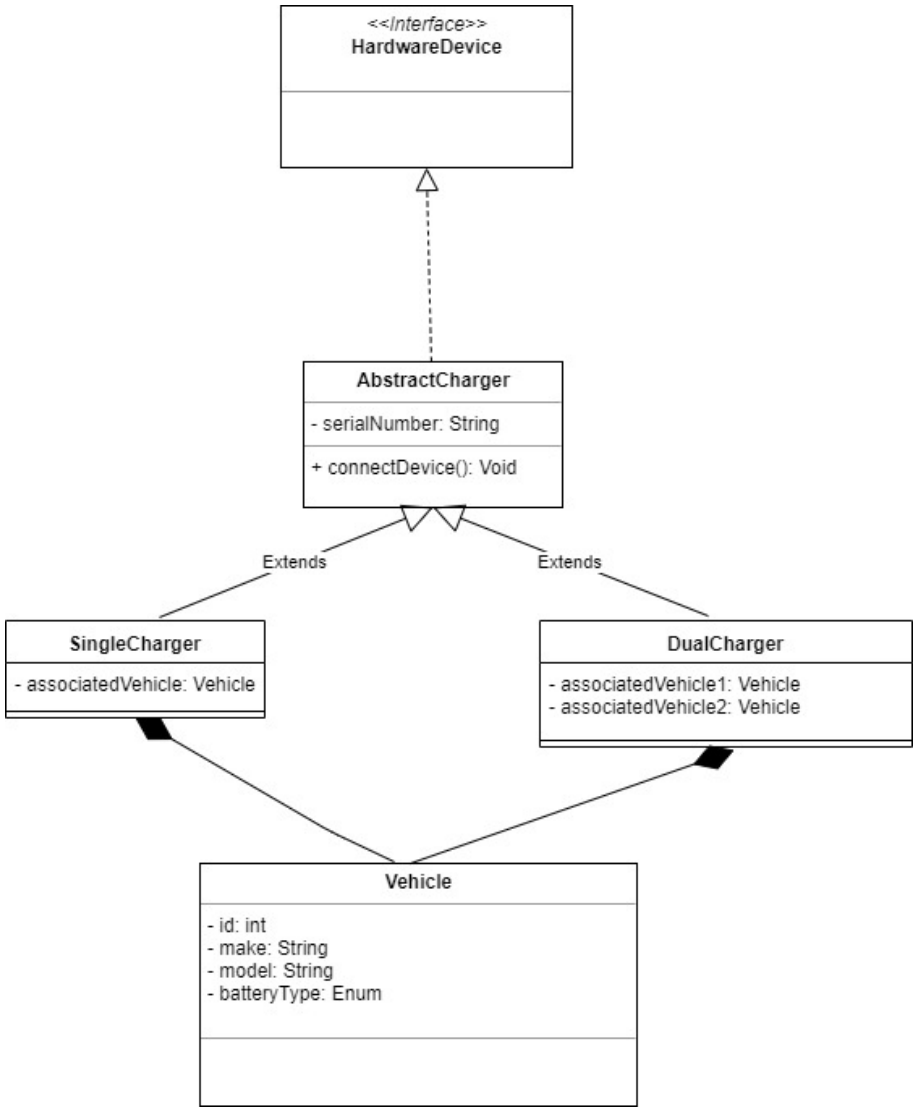

Figure 12: UML Class Diagram

The start of the Class Diagram is show in Figure 10 above. All classes representing charging hardware inherit from, at the highest level, the HardwareDevice interface, and then the concrete charger classes (SingleCharger and DualCharger) extend the AbstractCharger abstract class. This was left broad because all we know of at this point is that we are going to be dealing with charging hardware, but it could very well be possible that our application will have to interface with new hardware in the future as requirements change. As good object oriented software should be designed, this hierarchy is left open to extension and closed to modification. For example, if a new piece of hardware

is released by NeoCharge that is a supplementary piece of hardware, and not necessarily a charger, it can implement HardwareDevice. If a new charger comes out, it can extend AbstractCharger.

4 Test

5 Issues

A Glossary

Define all the terms necessary to properly interpret the software architecture, including acronyms and abbreviations. You may wish to build a separate glossary that spans multiple projects or the entire organization, and just include terms specific to a single project in each software architecture.

B Issues List

The team has yet to make a concrete decision on whether or not to build a serverless application. Having a serverless application would mean that the application would instead integrate significantly with a third-party, cloud-hosted service to manage server side logic and state. While the server-side logic will still be written by the team (developers), it will be run in stateless compute containers that are event-triggered. On the other hand, having a server would mean that that client would remain unaware, with most of the logic in the system and aspects such as authentication, page navigation, and searching all implemented by the server application.