# Perishable Shipment Tracker: Using IoT, Web Bluetooth and Blockchain to Raise Accountability and Lower Costs in the Perishable Shipment Process

# Using Espruino, Blockchain and Amazon Web Services to Raise Accountability and Lower Costs in Perishable Shipping

Roland L. Galibert

A Thesis in the Field of Software Engineering

for the Degree of Master of Liberal Arts in Extension Studies

Harvard University

November 2019

# Abstract

Just like many other fields of application, IoT and blockchain have not left the perishable shipment industry untouched, and these technologies have already improved the overall quality of perishable goods by bringing advances to the shipping process. However, there is still a great deal of room for improvement, especially with regard to making IoT sensors more cost-effective, improving the accuracy of IoT sensors, and raising the accountability of perishable shipment actors both through GPS tracking and through the use of blockchain.

My thesis project has several goals: to provide a versatile, easy-to-use application to suppliers who want to deploy sensors to track their shipments, to add an additional level of accountability by incorporating blockchain in the application, and to examine the use of Espruino devices and Web Bluetooth as a way to maintain IoT sensor accuracy with minimal additional cost, and also as a way to allow suppliers to easily use a variety of sensors.

As a result, my project consists of three primary components, a mobile-centered serverless web application called the PST (Perishable Shipment Tracker), functionality provided by Amazon Web Services and a temperature/GPS sensor device made up of a puck.js Espruino unit combined with the GY-NEO6MV2 6M GPS NEO by Ublox.

Frontispiece

The Dunedin, the first refrigerated clipper ship to complete a successful shipment of refrigerated meat (from Wikipedia article “Reefer ship” https://en.wikipedia.org/wiki/Reefer_ship).

## Author's Biographical Sketch

The author graduated from Williams College in 1984 where he majored in English but also took a number of courses in computer science; his final project in his programming class was to write a basic Turing machine in FORTRAN (on keypunch cards). Another favorite course was a class in machine architecture where the final assignment was to emulate a primitive computer on a Commodore SuperPet using 6809 assembler.

The author then took a number of graduate courses in computer science at Rensselaer Polytechnic Institute (where assignments included developing a LISP interpreter for a systems programming course and implementing extendible hashing as a final project for a database management course). After a hiatus from the field of software development, the author returned to studies in the field in 2013 when he began his master's studies in software engineering at Harvard Extension School.

Finally, the author also works as a certified German to English translator and in his spare time enjoys golf, bridge, music and karaoke.

Acknowledgments

I owe many thanks to Eric Gieseke, my Thesis Director, who contributed to my thesis in many ways. We worked together for several months to improve on my original proposal (a system to provision BLE beacons) to come up with a project which was more specific and more current. Eric also helped to give me the proper mindset to developing the solution, for example recommending to me the book The Lean Startup so I would approach my solution from that standpoint, and recommending practical measures such as interviewing providers with real-life problems, creating user stories, etc. Finally, the learning I gained back in 2014 when I took Eric's software design course was also a great help in developing my solution.

I would also like to thank Dan Ward for his time. Dan's input as an oyster farmer and perishable food shipper was essential in providing function requirements that helped guide the design.

Table of Contents

# Chapter I.

## Problem Statement

The International Air Transport Association (IATA) defines a shipment as perishable if "its contents will deteriorate over a given period of time when exposed to harsh environmental conditions, such as extreme temperatures or humidity" (https://www.fedex.com/en-us/shipping/perishables.html). Perishable shipments therefore include commodities such as seafood, dairy, meats and pharmaceuticals.

Refrigeration is one primary method used to maintain a perishable commodity. In the paper "The Impact of Refrigeration", Barbara Krasner-Khait defines refrigeration as the process of cooling a space or substance below environmental temperature, and notes the concept has existed since antiquity, with the Chinese harvesting and storage ice before the first millennium and the ancient Egyptians using cool night air to create snow (Krasner-Khait, 2016).

The use of refrigeration to ship perishable cargo began to evolve from roughly the mid-1800s and was the result of the ice harvesting industry. Ship owners transporting ice found they could make even higher profits by taking advantage of the low temperatures of their vessels to transport perishable cargo, and by the 1870s meat was being shipped from the United States to London, from Argentina to France and from Australia to England (Bryant, 2016). Then around 1930, with the increased popularity of road transportation, mechanically cooled trucks also began to be used commercially (ABCO Transportation, 2015).

## Figures Including Expected Growth

A June 9, 2015 blog post by Joey Hougham of Transgistics claims that approximately 70% of all the food consumed in the United States is handled by cold chains, that the United States alone imports about 30% of its fruits and vegetables and about 20% of its food exports can be considered perishables (Hougham, 2015). These figures are corroborated by a blog post by the cold chain monitoring solution provider Controlant which states that FDA estimates show that approximately 15 percent of U.S. food is imported, including one half of all fresh fruits, 20 percent of fresh vegetables, and 80 percent of all seafood, and this import and export trend expected to continue its growth (Controlant, 2018). The Controlant post also indicates that the International Air Transport Association (IATA) claims that by 2021, "world sales of cold-chain drugs and of biologics such as vaccines and insulin will top $396 billion, in a global bio-pharma market exceeding $1.47 trillion."

Other reports also indicate that the perishable shipping industry will grow in the next few years. The "Global Perishable Goods Transportation Market 2018-2022" report, published by the market research company Technavio in July 2018, expects the perishable goods transportation market to grow by USD 5.19 billion over 2018-2022 at a compound annual growth rate of almost 8% (snapshot at https://www.technavio.com/report/global-perishable-goods-transportation-market-analysis-share-2018). Finally, a presentation by Gerard de Wit of WorldACD Market Data at the IATA World Cargo Symposium in March 2016 showed an increased demand for perishables in India and China.

Various reasons for this growth include an increased consumer desire for perishable goods due to increased urbanization and income growth, general population growth and the continued growth of the world's largest food retailers. As a rule, developed countries spend more on fresh food.

## Losses

Shipment losses incurred by perishable good suppliers are great. These losses can arise from factors including:

- a broken cold chain (not maintaining the proper temperature over the course of the entire shipment)
- improper shipping conditions
- logistics issues
- lack of compliance
- lack of standardization
- lack of accountability and transparency

Breaches in the cold chain actually contribute to a 25% waste of all perishable food products in the U.S. each year (Hougham, 2015). In addition, the annual economic impact of food waste is estimated at $218 billion in the U.S., $143 billion in Europe, and $27 billion in Canada (Young 2012; ReFED 2015; FUSIONS 2016) (from Mercier et al., 2017).

And according to a recent report from the IATA, the pharmaceuticals business "loses upwards of a staggering $35 billion per annum" solely as a result of temperature

excursions and “30% of scrapped pharmaceutical can be attributed to logistics issues alone.” (Buxbaum, 2018) Further data from IATA indicates that 25% of vaccines reach their destinations degraded because of incorrect shipping and that 20% of temperature-sensitive products are damaged during the transportation process due to a broken cold chain.

## Fundamental Challenges

Using refrigeration to ship perishable cargo brings a number of fundamental challenges; two of the most important are keeping shipping time to a minimum and maintaining the cold chain at every stage of the process.

### Time

Ideally, the total time for a perishable shipment should be as short as possible, with delays kept to a minimum (although even without delays some perishable shipment times are expected to last up to several weeks or even more than a month). For example, Kenneth Wu, the founder and CEO of the grocery delivery service Milk and Eggs in Los Angeles says his company focuses on shortening delivery times to ensure product freshness and safety and typically gets orders to customers in one to two hours (Wells, 2017). However, given situations such as shipper delays, missed connections, removing perishable cargo to make room for necessary fuel and even competition for cargo from other shippers, this is not always possible.

Maintaining the cold chain

According to the article "Time–Temperature Management Along the Food Cold Chain: A Review of Recent Developments," a cold chain is "the succession of refrigeration steps along the supply chain that are applied to keep perishable food in the desired temperature range" (Mercier et al., 2017).

Maintaining cold chain over all stages. Although a perishable being shipped must be kept in a chilled or frozen state along its entire supply chain, it is a significant challenge to maintain the temperature of the perishable item at each and every step of the chain (Mercier et al., 2017). For example, the cold chain is often broken at the "last mile" of the supply chain, when the product is actually delivered to its destination (Hougham, 2015). Should the final customer leave a shipment pallet sitting out or otherwise fail to continue to maintain refrigeration in good time, the product will become spoiled.

The very beginning of the shipment can be another point of failure; a good must have already been brought to the correct temperature by the time it is transferred to the delivery truck.

Different perishables have different requirements. In addition, suppliers must take into account the differing requirements for different products, which are not only based on the product itself but also on current regulations and consumption patterns (Controlant, 2018). For example, different categories of perishable food products (from dairy and eggs to fruits and vegetables up to meat and seafood) have different optimal temperature ranges in order to maximize their shelf-life and commercial potential (IATA 2009), from (Mercier et al., 2017).

## Related Challenges and Trends

In addition to the challenges basic to perishable shipping which are listed above, perishable good suppliers must also contend with a number of related challenges as well as current trends, a few of which are discussed below.

Multitude of regulations / paperwork

Perishable good suppliers must comply with a mass of domestic and international regulations which are designed to ensure proper harvest, handling, sanitation and documentation throughout the supply chain, and of course the associated paperwork. This is especially true of seafood shippers; for example, a blog post on the mainebiz.biz Maine business news website lists five regulatory bodies which oversee seafood exports, in areas from import requirements to sanitation inspections and cargo compliance (Schreiber, 2017). And of course, accurate records must be kept to verify requirements (including temperature conditions) have been maintained (Controlant, 2018).

Actors in the perishable good shipping process can choose to deal with (or not deal with) these regulations in a number of alternative ways. For example, after the U.S. Department of Homeland Security issued regulations requiring 100-percent inspection of all passenger aircraft cargo, a number of freight forwarders themselves attained the credentials required to inspect and certify shipments so that they could avoid having to wait for government inspections (Controlant, 2018). And in San Diego, because domestic regulations are so much more stringent than overseas regulations, many restaurants simply carry very little, if any, local products and offer mainly exported seafood (Shoffler, 2018).

Variety of actors

In addition to the numerous regulatory bodies, which can vary by specific product and by import/export country, perishable product suppliers must also take into account the other actors involved in the shipment process, which include the actual shipper (or shippers) and other intermediaries such as wholesalers. Since a shipment between supplier and end customer is often handled by several parties (and not just one intermediary), tracking the process becomes especially complicated (Ward, 2018).

As in any industry, competition can also be a problem for perishable product suppliers. For example, an August 25, 2017 blog post describing a study of meal kit providers described the "Wild West" environment of that industry, "where scores of startups have jumped in to try and claim a slice of the market" despite not having the capacity to properly handle product and control temperature. And trends such as the farm-to-market movement will of course also affect the number of suppliers, markets and restaurants involved in perishable shipping.

As a result, perishable product suppliers often look for ways to reduce the number of actors required in the shipment process. As mentioned in the previous section, a number of freight forwarders attained the credentials required to inspect and certify shipments so as to avoid having to have the government do this inspection. As another example, the FreshDirect removes physical stores from the supermarket equation by delivering groceries to customers throughout the greater New York metropolitan area, all of New Jersey, Philadelphia and Washington, D.C. (Orlando, 2017). Finally, blockchain and IoT technologies (discussed in greater detail below) make it possible for suppliers to

eliminate intermediaries and in general give them much more control over the shipping process.

Variety of carriers

Suppliers should be prepared to be flexible when it comes to selecting a carrier for their goods, and not just for reasons of cost. Factors such as seasonal conditions, the competition and need to store fuel can affect a specific carrier's ability to take on a shipment for a supplier at all.

Also, just in terms of air shipping, freighters and passenger aircraft each provide different advantages. "Passenger flights are generally more frequent, less expensive, and more widely available, they require adherence to tight schedules, and may get bumped due to a variety of reasons. Freighters often offer better temperature control, additional capacity, and fewer inspections, but they may fly less frequently and to fewer locations, be more costly, and may sit until they reach near capacity, placing perishables at risk." (Controlant, 2018) The nature of goods being shipped, market needs and other elements can also dictate a choice between freighters and passenger flights.

Ocean transport is also an option for shipping perishable goods, especially those with longer shelf lives (Controlant, 2018). Some suppliers, especially pharmaceutical companies, are also moving to ocean transport instead of air because air carriers can be unreliable about ensuring the integrity of temperature-controlled shipments (also, costs are cheaper) (Buxbaum, 2018). However, efforts by ocean transport providers to reduce fuel consumption by traveling more slowly has led some perishable commodities to return to air cargo (Controlant, 2018).

## Accountability

For legal, regulatory and business reasons, perishable good suppliers must ensure accountability, especially in cases where a shipment becomes spoiled. This can be difficult, if only for the number of actors involved in the shipping process (as described above). Other factors make ensuring accountability difficult, for example, the meal kit study cited above notes that carriers like FedEx, UPS and the U.S. Postal Service all waive any responsibility for perishable products; similarly, vendors say they're not responsible for deliveries that aren't made on time" (Wells, 2017). Another article notes that IATA claims 20% of vaccines are damaged during transport due to lack of compliance, standardization, accountability, and transparency across the air transport supply chain (Controlant, 2018).

## Food traceability

The International Food Information Council Foundation (IFIC) has named food traceability as one of the top five food trends in 2019 (Te-Food, 2019). Food Standards Australia New Zealand defines traceability as the ability to track any food through all stages of production, processing and distribution (including importation and at retail), including the ability to trace movements one step backwards and one step forward at any point in the supply chain (http://www.foodstandards.gov.au/industry/safetystandards/traceability/Pages/default.aspx).

Such traceability has seen increasing demand from consumers in recent years, and not only from consumers who are able to afford fresh foods and specialty foods and expect high quality. Because of increased media access made available by the Internet

(including social media) and mobile devices, consumers are much more aware of cases of food poisoning such as the recent outbreak of E. coli infections related to romaine lettuce, and therefore look for ways to ensure the food they eat is not harmful.

However, suppliers and other actors also show an increasing demand for food traceability, and not just to meet the increased demand for regulation on the part of consumers. Food traceability helps suppliers to reduce business costs, including costs associated with spoilage and product recalls, to navigate a global food supply chain that is becoming increasingly complex and to deal with changing industry processes (Fisher, 2015).

## Recent Digital Technologies Which Support Perishable Shipping

A number of informational technologies which have been emerging in recent years are very well-suited to meeting the various challenges faced by the perishable shipping industry. One of these is the Internet of Things (IoT), which enables applications such as time-based asset tracking, temperature tracking and tracking of other conditions such as humidity and motion. The possibilities offered by the emerging technology of blockchain can also be applied to solve problems which come up in perishable shipping.

### IoT (Internet of Things)

The Internet of Things (IoT for short) refers to the billions of physical devices around the world that are now connected to the Internet (Ranger, 2018). The term IoT is generally used for devices that wouldn't traditionally be expected to have an Internet connection, and that now collect, communicate and share data over the Internet in real

time, independently of human action (Ranger, 2018). The field is experiencing rapid growth – with 8.4 billion IoT devices in use in 2017, there are already more connected things than people in the world, and the analyst Gartner calculates that this will likely reach 20.4 billion by 2020, with company spending on IoT endpoints and services already reaching the trillions (Ranger, 2018).

Smart sensors are one key ingredient which makes IoT work. In general, a smart sensor has three major components: a sensor that captures data from an environment; a microprocessor, which computes on the output of the sensor via programming; and communications capabilities that enable the sensor to communicate the microprocessor's output for action (Shea, 2015). To be most effective, IoT should include wireless communications, be smart enough to compute data remotely and be programmable to accommodate new capabilities as needed, said Institute of Electrical and Electronics Engineers senior member Shawn Chandler.

Espruino / Web Bluetooth.

Two technologies that can be used separately or in combination to bring about these three requirements (wireless communications, compute data onboard and be programmable) are Espruino and Web Bluetooth.

Espruino is an open-source JavaScript interpreter for microcontrollers which was created by Gordon Williams in 2012 (its Kickstarter project calls it “JavaScript for Things”). Espruino (including the JavaScript interpreter and development toolchain) is installed directly on the microcontroller itself, making it possible to control an Espruino-based device using JavaScript code (Williams, 2017). Such devices are versatile as well as

fairly accessible, since JavaScript is relatively easy to learn and is a language with which programmers are generally familiar.

In addition, if an Espruino device has enough memory (such as the puck.js device (https://shop.espruino.com/puckjs) used in this thesis project), it is easy to use it as a data logger through the JavaScript interpreter.

Web Bluetooth is a JavaScript API which was designed by Jeffrey Yasskin and Vincent Scheib of Google in 2014 (Woolley, 2017). The API, now a W3C Community Project (https://www.w3.org/community/web-bluetooth/), makes it possible for web browsers to connect to and interact with Bluetooth devices, through the use of JavaScript. One advantage of this is that someone who wishes to use a Bluetooth device wouldn't have to download an install a native application, but could use the device immediately just by accessing a website or wherever else the code in question exists. In addition, devices can be controlled based on real-time conditions, making them extremely versatile. Finally, the code is easily maintainable as it exists in one centralized location.

Tracking of temperature, humidity and other conditions.

IoT sensors which detect temperature, humidity, motion (like vibrations, jolts and shocks), air quality or other conditions and have the capacity to store, process and transmit their readings are important in meeting the challenges of perishable food suppliers. The use of such sensors can help suppliers proactively react to suboptimal conditions in the cold chain, to avoid further losses in cases where goods are already spoiled, and to collect historical data to improve future shipments. Such sensors are especially effective when used in combination with asset tracking.

Location/asset tracking.

In his article “What is the IoT? Everything you need to know about the Internet of Things right now”, Steve Ranger writes that one of the first IoT applications was to add RFID tags to expensive pieces of equipment in order to help track their location (assets can also be tracked using other methods like GPS and Bluetooth). Ranger goes on to say that the cost of adding sensors and an Internet connection to objects has continued to fall, and experts predict that this basic functionality could one day cost as little as 10 cents (Ranger, 2018).

A tracking unit that uses GPS will determine location by getting a read through GPS satellites, then either store this location (latitude, longitude, altitude, etc.) and/or communicate it to a reading device.

Using a sensor to track the location of a perishable shipment is extremely advantageous for suppliers in many ways. They can know exactly where a shipment is at any time, so shipping deviations can be corrected immediately. When location is used in combination with time, suppliers can determine estimated time of arrivals and also become aware of delays, e.g. shipments that remain at the same location for too long a period of time. Finally, when used in combination with tracking of temperatures and other measurements, suppliers can determine exactly where a shipment is when it has gone bad or is about to go bad, making it possible to either make proactive corrections, save time by cancelling a bad shipment, and in general ensuring accountability on the part of shippers. Finally, historical location data can help a supplier plan future shipments by taking into account things like how long a trip is likely to take, probable traffic conditions, etc.

Blockchain

Investopedia describes blockchain as a "distributed, decentralized, public ledger." Its distinguishing features are that the transactional data stored on a blockchain are immutable and transparent to anyone, and the blockchain has no central authority, as it is maintained by many different computers (thousands and even millions).

The concept was first conceived in 1991, for the purpose of implementing a system preventing the tampering of document timestamps. However, its first real-world application didn't arrive until 2009, with Bitcoin, described by its pseudonymous creator Satoshi Nakamoto as "a new electronic cash system that's fully peer-to-peer, with no trusted third party."

A blockchain essentially works as follows (Rosic, 2019):

- A user requests a transaction (e.g., a purchase from an online store).
- The transaction (participants, time of purchase, purchase amount, etc.) is verified by broadcasting it to a peer-to-peer network made up of nodes (computers).
- The verified transaction is stored in a block (which includes the relevant information
- The block is assigned a hash code and inserted at the "top" of the blockchain as its last element

The system also has the following important features:

- Every block contains its own hash as well as the hash of the preceding block. This means that a hacker who changed the data in a block would then need to change the hashes of subsequent blocks in the chain, which would take immense effort in terms of computing power)

- Every computer on blockchain network has its own copy of blockchain, meaning a would-be hacker would also need to manipulate each copy of the blockchain. This system brings the following advantages:
- Elimination of intermediary and associated costs - For example, someone making an online purchase might use a credit card, PayPal or some other third-party provider for the transaction, and would need to pay the associated fee (as well as expend the effort to involve this provider at all). With blockchain, the need for this third party is eliminated.
- Data immutability – Data stored on the blockchain cannot be changed and are therefore secure and reliable.
- Transparency – Data on the blockchain are transparent so that transactions can be traced.
- Accountability – This data immutability and transparency means that all blockchain participants are easily kept accountable, as it is virtually impossible for them to tamper with their transactional data, and any blockchain participant can view their data.

These general advantages provided by blockchain are of course also advantages for any perishable supplier who uses that technology to secure shipment data to hold perishable shipment actors accountable and to eliminate unnecessary intermediaries as well as the associated paperwork and costs. In his article “Blockchain for global maritime logistics”, Ashraf Shirani describes some potential gains perishable suppliers can realize through the application of blockchain, including reduced use of intermediaries, :especially the freight forwarders who, by some estimates, account for over 20% of the total cost of logistics”,

savings in total shipping time, savings in time spent on paperwork ("the United Nations' estimates suggest that by putting the Asia Pacific's trade related paperwork online would save as much as 44% in time") and "real- or near-real time availability of data in the blockchain about location, condition, and movement of goods [which] would enhance visibility for importers and exporters into their supply chains and help them make better-informed decisions in sales, marketing, logistics, and other areas".

And as a final plus, one of the four technological fields identified by Shiran as excellent candidates for synergistic relationship with blockchain technology is the Internet of Things (IoT) described above.

# Chapter II.

# Prior Work

The following is a non-exhaustive list of perishable shipping tracking applications, multi-function sensors, GPS sensors and temperature sensors. The purpose of this list is to give a general idea of what is currently available in terms of perishable shipping technology and to provide a baseline and target for tests for the puck.js/Ublox GPS sensor combination sensor on which this thesis is based.

## Tracking Applications

| Name | Sendum Asset Monitoring Solution |
|---|---|
| Website | https://sendum.com/ |
| Price | Findum software 36-month license – minimum $594 (544.50 upfront, 16.50 month repayment) – includes Asset Monitoring Device (price in Australian dollars and based on information from Telstra mobile phone provider at https://www.telstra.com.au/content/dam/tcom/personal/help/pdf/cis-business/machine-2-machine/business-critical-information-summary- |

| | |
|---|---|
| | Sendum.pdf) |
| Measurements supported | Motion, shock, orientation, battery level, GPS jamming detection, temperature, relative humidity, barometric pressure, and light power (some measurements require purchase of Advanced Sensor Pack) |
| Temperature range | storage -20° C to +65° C / -4° F to +140° F<br>active use -20° C to +40° C / -4° F to +104° F |
| GPS features | |
| Battery | 3760 mAh lithium-ion battery (PT 300D unit)<br>External battery also available (not included) |
| Battery life | Estimated 21-day battery life |
| Size | 2.09" x 4.92" x 0.51" |
| Weight | 100 grams |
| Other features | Can set rules for automatic notifications/alerts/exceptions |
| Comments | Data collected made available through portal |

## Multi-Function Sensors

Note: Queclink (www.queclink.com) offered a number of different sensors of varying sizes and with different battery types. Depending on reporting interval, these batteries provided a maximum life before recharging from 190 hours to 5 years. All GPS units used the Ublox All-in-One GNSS receiver with position accuracy (autonomous) < 2.5 m. I have included the two cheapest units, based on various websites.

| | |
|---|---|
| Model name | Queclink GL300ma |
| Price | $39.00 (Amazon) |
| Measurements supported | location, motion detection, light monitoring, temperature/humidity monitoring |
| Temperature range | -20 C ~ +60C |
| GPS features | Ublox All-in-One GNSS receiver<br>Position accuracy: autonomous: < 2.5 m |
| Battery | Lithium polymer, 15000 mAh |
| Battery life | 10 days to 95 days, depending on reporting interval (190 days without reporting) |
| Size | 74mm(L) x 34.5 mm (W) x 151 mm (H) |
| Weight | 375 g |
| Other features | water resistant, OTA control, geo-fencing<br>Scheduled report – Report position and status based on preset time intervals, distance, mileage or a combination of these settings |

| | |
|---|---|
| Model name | Queclink GL300 |
| Price | $88.00 (Amazon), $130 (Walmart) |
| Measurements supported | motion detection, ignition detection, vibration feedback |
| GPS features | Ublox All-in-One GNSS receiver<br>Position accuracy: autonomous: < 2.5 m |
| Battery | Lithium polymer, 1300 mAh |
| Battery life | 120 hours to 190 hours, depending on reporting interval (280 hours without |

| | |
|---|---|
| | reporting) |
| Size | 38.5mm(L) x 23.5 mm (W) x 68.5 mm (H) |
| Weight | 60 g |
| Other features | water resistant, OTA control, geo-fencing<br>Scheduled report – Report position and status based on preset time intervals, distance, mileage or a combination of these settings |

## GPS Trackers

| | |
|---|---|
| Name | SpyTec STI_GL300 Mini Portable Real Time Personal and Vehicle GPS Tracker |
| Price | 49.95 + $25/month |
| Battery | Lithium polymer 1300 mAh |
| Battery life | Standby time w/o reporting: 400 hours<br>reporting every 5 minutes: 130 hours<br>reporting every 10 minutes: 150 hours |
| Size | 2.7” x 1.6” x 0.8” |
| Weight | 2.1 oz. |

## Temperature Loggers

| | |
|---|---|
| Name | LogTag TRIX-8 Temperature Data Recorder |
| Price | $32 at microdaq |
| Temperature range | -40°C to 85°C with ±0.5°C accuracy |
| Battery | 3 volt lithium batttery |
| Battery life | 2 to 3 years @ 15 minute sampling interval and monthly data downloads |
| Size | 3.39" x 2.14" x 0.33" |
| Weight | 35 grams |
| Other features | Sampling frequency 30 seconds to 18 hours<br>Windows compatible |

## Espruino

| | |
|---|---|
| Name | Nordic Thingy 52 |
| Website | https://www.nordicsemi.com/Software-and-Tools/Development-Kits/Nordic-Thingy-52 |
| Price | $39.00 at Mouser Electronics |
| Measurements supported | Environment (temp, humidity, pressure, air quality color and light)<br>9-axis motion sensing (accelerometer, gyroscope and compass) |
| Battery | 1440 mAh rechargeable through USB |
| Size | 2.4” x 2.4” |
| Other features | Web Bluetooth<br>Android, iOS apps<br>Near Field Communication (NFC) support |

| | |
|---|---|
| | Speaker and microphone<br>Program Memory 512kB Flash with cache<br>RAM 64kB<br>GPIO 32 configurable |
| Comments | Intended for prototyping |

# Chapter III.

# Requirements

## Dan Ward, Ward Aquafarms

Dan Ward is owner of Ward Aquafarms, an aquaculture farm in Megansett Harbor, North Falmouth, MA which commercially grows oysters, bay scallops, quahogs, and sugar kelp (website at http://wardaquafarms.com/). Eric Gieseke, my thesis director, had attended an IoT-related meetup a few years ago in which Dan presented, and felt the challenges faced by Dan in his business might benefit from a solution based on the puck.js or similar IoT device.

On December 31, 2018 Dan, Eric and I met through Skype. My master's project, a mobile application that provisions IoT devices to track perishable shipments, as well as the targets I used to measure application performance, are in most part based on the information Dan shared with us at the meeting (Eric's notes on our meeting also appear in the Appendix). The major points of the meeting were as follows:

### General shipping process and actors

Dan's end customers include restaurants and fish markets (known as receivers below) located mainly in the United States, although he does have international customers. He ships his products to these actors through various shippers and distributors/wholesalers; a shipment may involve more than one of these intermediaries.

Accountability

Dan mentioned that on the whole shipments ran well and that he worked well with intermediaries, but that unfortunately there were a few intermediaries who were "bad apples" and he needed to hold those people accountable if a shipment became spoiled while it was in their hands.

Measurements and frequency

As a result, Dan needs to regularly track the temperature and location of his shipments, so that if a shipment becomes spoiled, he'll know exactly where and when this happened. He specified a general rate of one sample per hour, although he added in some cases the sampling rate might need to be anything between fifteen minutes and one hour.

Total shipment times average two to three days but could be as long as fourteen days.

Shipment worth

Dan ships about 50 to 100 bags per week, with each shipment worth $60 to $100.

Disadvantages with previous systems used / failure points

Device cost – At one point Dan tried a sensor model from Queclink (website at http://www.queclink.com/); its location and temperature readings were very accurate, but the unit was every expensive (roughly $300).

Accuracy – At another point, Dan also tried a model from Sendum (website at https://sendum.com/); at $30 the device was much cheaper than the Queclink but was very inaccurate both in terms of location and temperature.

Battery cost – Another failure point Dan mentioned was the high cost of batteries (specifically lithium ion batteries).

End customer involvement – Dan's experience was that the end customer (restaurant, fish market, etc.) could not be relied on to return a device following a shipment. As a result, the device should effectively be disposable (a maximum cost of $1). Possibly the end customer should also have an incentive to return the device.

Mobile-based application

Dan works mainly from his smartphone, thus any tracking application he uses would need to be mobile-ready and include appropriate alerts.

Miscellaneous considerations

- The oyster industry is highly regulated; as a result, Dan's shipments are already tagged (at the bag level) with a physical tag which indicates the harvest number, harvest location, date and time the oysters were harvested and the date and time they were iced. The end customer is required to keep this tag for 90 days.
- Actor involvement should be kept to a minimum; at best, the end customer might be willing to take a reading and, due to the number of intermediaries involved and their workload, those parties couldn't be expected to be

involved in the tracking process, even by setting up a stationary reader at each access point.

- End customer incentive – One incentive for an end customer to become more involved in the process might be to provide their end customers (restaurant guests, fish market customers) with information about the product they're eating (e.g. where the oysters came from, when they were picked, etc.) Bumble Bee® Seafoods already provides such a tool with its "Trace My Catch" web page (at https://www.bumblebee.com/tracemycatch/).

# Chapter IV.

## System Overview

My master's project has a number of goals:

- offer a versatile, easy-to-use application to perishable product suppliers who want to deploy sensors to take readings (not just GPS and temperature) over the course of their shipments, for the purpose of holding all parties to the shipment accountable for their participation in the shipment.
- add an additional level of accountability to this shipment process by storing readings in a blockchain.
- The solution is intended to be offered to all sorts of perishable product suppliers and as such is intended to accommodate a variety of sensors.
- The other main goal of my project is to provide improvements on Dan Ward's specific needs by offering a device for tracking GPS location and temperature that maintains the accuracy of those readings but is more cost-effective than the solutions he previously used.
- The versatility offered by the application is also shown to some degree in the puck.js logger/temperature sensor which is the base unit of the device.

As a result, my project consists of three primary components, a mobile-centered web application called the PST (Perishable Shipment Tracker), the functionality

made available through Amazon Web Services, and a temperature/GPS sensor device made up of a puck.js Espruino unit combined with the GY-NEO6MV2 6M GPS NEO by Ublox.

PST Application

The PST application is a solution provided to perishable product suppliers who want to deploy sensors in order to take readings over the course of their shipments, for the purpose of maintaining shipment party accountability and in turn improving the quality of their product. The application aims to be scalable and to provide suppliers with a high degree of versatility and ease of use, especially in terms of sensor device selection, reader selection and application device selection (although given the current state of technology and Dan's needs, it is primarily a mobile application). The application is hosted by Amazon Web Services and thus is also scalable in terms of supplier user numbers.

The PST provides all functionality related to this. Supplier functionality includes:

- Offering suppliers a variety of sensor devices for purchase.
- Easy-to-use processes for creating shipment records and associating to these the desired customer (or shipment receiver), measurements, alerts, and internal supplier order.
- A preused value system is provided to facilitate shipment creation; after a supplier first enters a value for a shipment (a product, shipping unit or non-standard shipper), the value will automatically appear in future shipment creation screens as a radio button option.

- Suppliers may also save entire shipment records as templates, which are then offered for selection when future shipments are created.
- Easy-to-use processes for deploying sensors (especially those which support Web Bluetooth) for shipments, taking readings from sensors, and uploading these to a central cloud location.
- Once a supplier finalizes a shipment, the associated readings automatically receive a blockchain transaction ID and hash to render them untamperable and ensure accountability.
- SMS alerts automatically sent to shipment suppliers and receivers when defined thresholds are exceeded.
- Shipment activity recorded in logs (logs are also provided for supplier devices).
- A cache is provided in each reader to ensure sensor readings are stored in case network connectivity is unavailable.

PST administrators have the following functionality:

- Creation of new device models (including association with measurement types supported, vendor information, etc.)
- Addition of new measurement types available as well maintenance of other master data (alert types associated with a measurement type, etc.)
- Updating of device model logs to clarify history, record issues, etc.

puck.js

The puck.js (https://www.puck-js.com/) is a small IoT device which comes with a number of features which make it an excellent device model choice in the PST application. Most important, its microcontroller comes with Espruino, a JavaScript interpreter, which means the puck.js can easily be controlled through JavaScript. In addition, the puck.js supports Web Bluetooth, which means the JavaScript to control the puck.js can be stored in and called from a website, meaning any authorized user would be easily able to control the unit without having to download and install additional code.

The puck.js comes with 64kB RAM and 512kB Flash controllable through Espruino, so it can easily log large amounts of data. Finally, in addition to the sensors and other hardware which come ready on the device (temperature sensor, light sensor, magnetometer, IR, RGB LEDs, BLE beacon functionality, control button) the puck.js' board has a number of GPIO pins so that the unit can be connected to other modules/sensors, such as the Ublox GPS module described below.

To sum, the puck.js can be easily controlled just by accessing a website, provides the logging necessary for tracking, and makes available a variety of sensors, either on its own or by being connected to a separate sensor.

GY-NEO6MV2 6M GPS NEO module by Ublox

The Ublox GY-NEO6MV2 6M GPS NEO is the GPS sensor portion of the hardware device I used for this project. The main reasons I selected it were for its small size and good price ($12.45 on Amazon https://www.ebay.com/itm/Ublox-GY-NEO6MV2-6M-GPS-NEO-Small-Antenna-Package-for-Arduino-AVR-

PIC/331831468002?hash=item4d42b25fe2:g:GBQAAOSww3tY4vV2:rk:3:pf:1&frcectupt=true).

The device is soldered to the puck.js and immediately sends data to the puck.js once it is powered on. It may take several minutes for the GPS module to attain a good signal.

Sensor Power Options

Some options for powering a puck.js/Ublox GPS module combination were described on the Espruino forum conversation "GPS Module not sending data on Puck.js" (http://forum.espruino.com/conversations/324744/) and included powering the puck.js with a CR 2032 coin battery (its normal power source) and powering the GPS module with three AA cells or with a lithium polymer battery, powering both units from the puck.js' CR 2032 coin battery, or powering both units from a separate power source (e.g. a lithium polymer battery), possibly in combination with a resistor.

Sensor Protection Options

I purchased a variety of cases to house the puck.js/Ublox GPS module combination; these included:

- Hammond 1551KTBU Translucent Blue ABS Plastic Project Box, in sizes 1.97" x 1.97" x 0.59", 2.36" x 1.38" x 0.59", 3.15" x 1.58" x 0.79" (https://www.amazon.com/gp/product/B007PBYRZ6/ref=ppx_yo_dt_b_asin_title_o01_s00?ie=UTF8&psc=1).
- Adafruit Small Plastic Project Enclosure 3.2" x 4.4" x 1.8" (https://www.adafruit.com/product/903)

- Adafruit Pycom Universal IP67 Case 85 x 85 x 40.5mm (https://www.adafruit.com/product/3690)
- All of these are weatherproof and fairly rugged so as to meet Dan's needs and those of other suppliers. Tests are described below to study each case's robustness and ensure it does not affect acquisition of a GPS signal.

# Chapter V.

# Design and Technology Choices

## Mobile optimized application

Given Dan's requirements as well as the general user trend toward mobile devices and away from desktops, it was obviously necessary that my application would need to be optimized for use on a mobile device. A 2018 blog post by Britt Armour entitled "Mobile App Vs. Mobile Website: Which Is The Better Option?" claims that the number of mobile-only Internet users has overtaken desktop-only users and that mobile app usage has surpassed desktop usage, giving the following statistics:

- Just two years ago in 2015, time spent on mobile surpassed time spent on desktop.
- Google reported that more searches were being made on mobile than desktop.
- From 2016 to 2017, time spent per day on mobile has increased by approximately seven minutes, reaching a total 3 hours and 15 minutes per day.
- During that period, time spent on desktop decreased by one minute and TV viewing decreased by five minutes.

The following statistics also appeared in a different blog post (Ciligot, 2019):

- Fifty-eight percent of website visits came from a mobile device in 2018, surpassing desktop usage for the fourth consecutive year.
- Mobile devices also accounted for 42 percent of total time spent online.

However, the Armour post also noted that although mobile use is surpassing desktop use exponentially, the latter "isn't going away anytime soon". For this and other reasons, I chose to make the Perishable Shipment Tracker a browser-based application.

Responsive mobile-based web application vs. native mobile application

Another choice I had to make was whether to develop my application as a native mobile application (running directly on Android, iOS, etc.) or as a website optimized for mobile use. Both application types have advantages and disadvantages (Armour, 2018; Stevens, 2018):

Native mobile apps – advantages:

- have access to mobile device system resources
- can run offline
- run faster
- more advanced in terms of features and functionality
- more secure, as they must first be approved by the app store
- can identify user location
- can be personalized based on user configuration and user actions (this is an advantage especially for marketers who want to engage users)
- can be better for users, as they are generally familiar with the specific mobile operating system they use

Native mobile apps – disadvantages:

- need to be downloaded and installed

- must be developed specific to a given operating system
- more expensive and difficult to maintain and update

Mobile-optimized websites – advantages:

- don't need to be downloaded
- aren't tied to a specific operating system
- can be easier to develop (using website templates, etc.)
- less expensive to develop
- easy to maintain
- can reach a wider audience
- greater search engine reach capability (in addition, mobile-optimized websites rank higher in search engine results than those not optimized for mobile)

Mobile-optimized websites – disadvantages:

- require an Internet connection
- slower
- fewer features, functionality (as they cannot access mobile device resources)
- quality and security not always ensured, as they don't need to be approved by an app store

As mentioned above, in the end I chose to make my application a browser-based application optimized for mobile use, for the following reasons:

- I wanted my application to be able to run on a variety of devices (mobile phone, desktop computer, etc.)

- As I wanted the application to be able to support the use of Espruino devices, it in turn needs to be able to support Web Bluetooth, as well as the use of JavaScript to run such devices. This is easily done using a browser-based app, as all modern browsers support both of these (https://caniuse.com/#feat=web-bluetooth). Espruino device communication and control would be considerably more difficult through Android or iOS.
- Another key component of my application is the use of serverless computing and in turn an API gateway. I felt a web application would make it clearer and more straightforward to implement the API calls, as there are already many JavaScript tools that can be used for this purpose.

### Serverless Application vs. Traditional Web Hosted Application

Another decision I made was to implement my solution using serverless computing (specifically Amazon Web Services).

In its current usage, serverless generally refers to a "Functions as a Service" (FaaS) application where server-side logic is written by the application developer, but, unlike traditional architectures, the logic is run in stateless compute containers that are event-triggered, ephemeral (may only last for one invocation), and fully managed by a third party (Fowler, 2018). Such an approach eliminates much of the need for a traditional always-on server component, and in short relieves developers of much of the need to manage resources, such as implementing logic to scale their applications to variable factors such as number of current users, and significantly reduces operational cost and complexity (but of course brings some drawbacks).

Serverless can also refer to a "(Mobile) Backend as a Service" (BaaS) application, which significantly or fully incorporates third-party, cloud-hosted applications and services to manage server-side logic and state (Fowler, 2018). In other words, such applications make great use of the variety of services provided through the cloud (databases, authentication services, etc.) instead of depending on a single web host to provide these services.

Advantages of serverless computing

The advantages of using serverless computing (FaaS and/or BaaS) to implement an application include the following (Roberts, 2018):

- reduced operational cost – a serverless application saves costs in a number of areas – since the serverless computing provider is doing most of the work of managing resources, the saved labor results in savings, costs of developing authentication functionality, DB functionality, etc. are avoided, and overall costs are cheaper since many parties are using the same resources.
- scaling cost savings – since horizontal scaling is automatic, serverless computing customers only pay for the server time they actually use. This means that a customer no longer needs to buy extra hardware to accommodate isolated spikes in traffic, extra hardware which would be wasted the rest of the time. In addition, any performance optimizations which are implemented will not only improve the speed of an application but also reduce operational costs. Finally, such an approach greatly reduces costs and lead time for the experimental work involved in developing a new product.

- easier operational management – in the case of BaaS, fewer components need to be supported, while an FaaS application requires “zero system administration”
- greener computing – According to Forbes, typical servers in business and enterprise data centers on average deliver between 5 and 15 percent of their maximum computing output over the course of the year (Roberts, 2018). One reason for this is that enterprises manually make decisions about capacity which often last for months and years, and they must take into account isolated peak periods of high traffic. In a serverless approach, an enterprise is relieved of making such decisions and only uses as much resource as they actually need.
- ease of development – there is no need to code to specific framework or library; FaaS functions are regular applications when it comes to language and environment. All that needs to be done is to upload the code for functions, and the FaaS provider takes care of everything else.

Disadvantages of serverless computing

In turn, a serverless application can bring the following disadvantages:

- tight vendor control – Serverless computing customers need to deal with things such as system downtimes, unexpected limits, changes in costs, loss of functionality and API upgrades. In addition, an FaaS provider will set tight limits on customers so as to ensure reliability of the overall system.
- multitenancy problems – a solution with multiple tenants can bring problems of security, robustness and performance.

- vendor lock-in – switching vendors can be difficult, since serverless features are very likely implemented differently from vendor to vendor.
- increased application client responsibility - A BaaS/FaaS application client has much more logic than client in a traditional web application, since it must keep track of a user session, read from a database, understand the UX structure of the application, etc.
- greater number of components to manage – Both BaaS and FaaS involve a greater number of "moving pieces" than a traditional monolithic application, so a serverless computing customer must be prepared to take this into account (choreography over orchestration).

I elected to implement the PST solution as a serverless application for the following reasons:

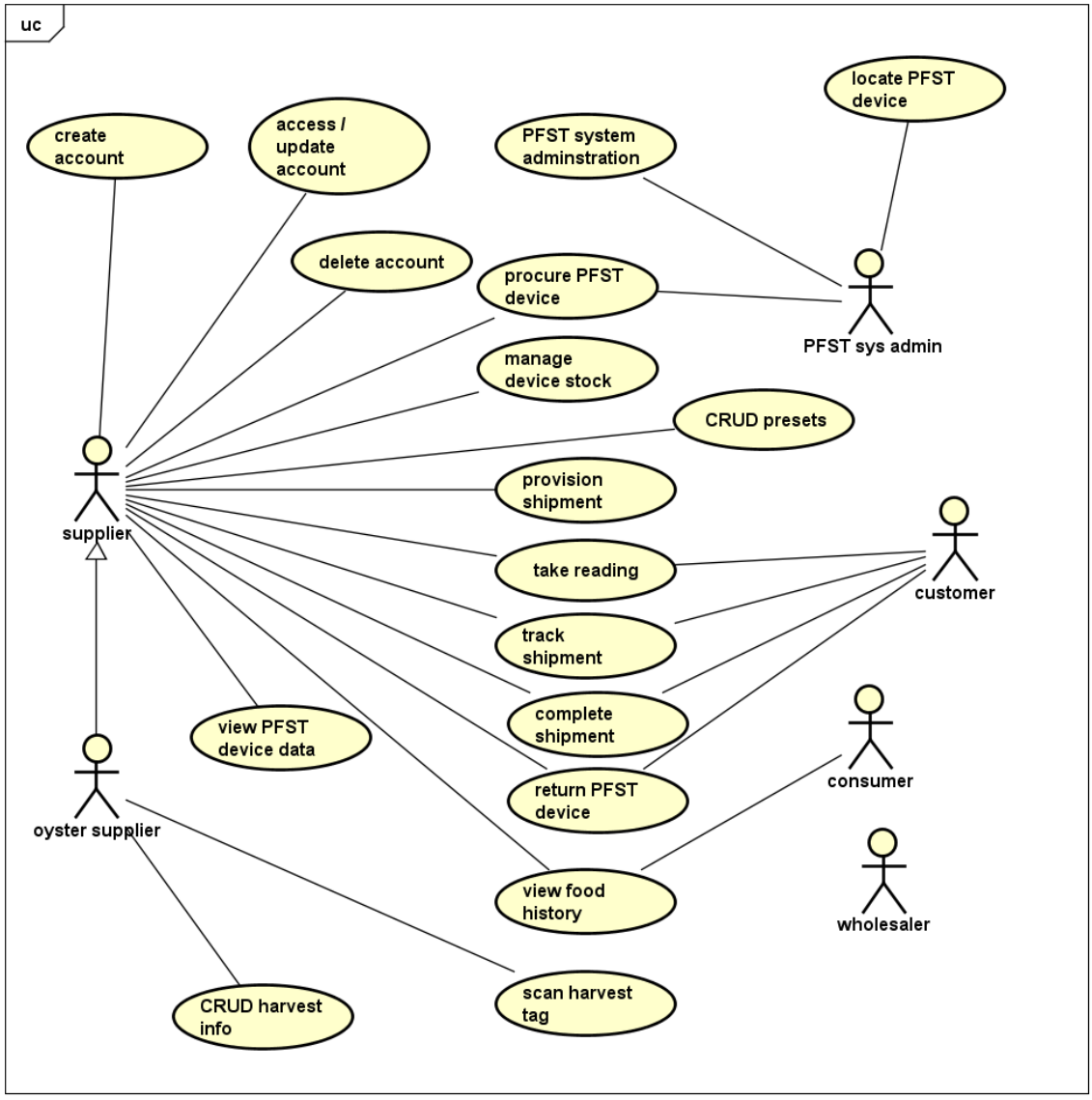
- Variety/multitude of suppliers and other actors – The PST application is intended to handle a varying number of suppliers, each of whom may have one or more receivers and all of whom (including the supplier) are authorized to upload readings and view shipment data. It is also expected that other actors, such as intermediaries (shippers and wholesalers) as well as end customers (restaurant patrons and gourmet food customers) will respectively upload readings and view shipment data. The number of these operations will also vary based on conditions such as time of day, time of the year, etc., and may be quite large. As a result, the automatic scaling possibilities provided by a serverless computing solution are a good way to meet these challenges.

- Variety of services – At a minimum, the PST application will make use of at least two third-party services; a blockchain service as well as a service to take in supplier payments for using the application. In addition, I am considering the use of an authentication service different from Cognito. As such an architecture already corresponds to a BaaS setup, it makes sense to implement the solution as a serverless application.
- Data volumes associated with IoT applications – Using serverless computing to implement the PST application also makes sense since the application will need to volume large amounts of data; it is expected that a large number of suppliers will be making one or more shipments at a time, each being tracked over one or more measurements, with the number of these measurements increasing further depending on reading interval time. This decision is corroborated by Steve Ranger, who writes in his article "What is the IoT? Everything you need to know about the Internet of Things right now" that the huge amount of data that IoT applications generate means that many companies will choose to do their data processing in the cloud rather than build huge amounts of in-house capacity (Ranger, 2018).

# Chapter VI.

## Use Cases

### Use Case Diagram



Use Case Actors

PFST system administrator – An individual who maintains the overall PFST platform.

Supplier – A supplier of perishable food who has a PFST application account.

Oyster supplier – A specific Supplier, one who supplies oysters to Customers.

Customer – A purchaser and receiver of a shipment of perishable food (e.g. restaurant, gourmet food store, etc.)

Consumer – An actual consumer of the perishable good (e.g. a customer of a restaurant or gourmet food store).

Wholesaler – An intermediary between the Supplier and the Customer who is responsible for all or part of the perishable food shipment. More than one Wholesaler may be involved in any shipment.

Use Case – PFST System Administration

Description: A PFST system administrator configures/makes change to the PFST system.

Process:

1) The PFST system administrator logs into the application.
2) The PFST system administrator carries out required system configuration.
3) The PFST system administrator logs out of the application.

Possible system maintenance activities:

- Block/unblock a user
- Make new device available for purchase
- Approve/process a new supplier account request (purchase, etc.)
- Process a device purchase order from a supplier
- Process a failed payment

### 1.1.1 Use Case – Create User Account

Description: A supplier creates/accesses/updates/deletes his or her PFST application account.

Process:

1) A non-logged-in user clicks on the “Create new account” link.
2) The user is brought to the corresponding page and enters the required information, including payment.
3) The user is automatically logged in and brought to the device purchase screen.

### 1.1.2 Use Case – Access/Update User Account

Description: A supplier accesses and possibly updates data in his or her PFST application account.

Process:

1) The supplier logs into the PFST application.
2) The supplier reads the data in his or her PFST application account.
3) (Optional) The supplier updates data in his or her PFST application account.
    a. The supplier saves the changes or cancels the changes.
4) The supplier logs out of the PFST application.

### 1.1.3 Use Case – Delete User Account

Description: A supplier deletes his or her PFST application account.

Process:

1) The supplier logs into the PFST application.
2) The supplier clicks to delete in his or her PFST application account.
3) The supplier is asked to confirm this deletion:
    a. If yes, supplier is logged out, redirected to PFST application home page and his or her account deleted.
    b. If no, supplier is redirected to his or her PFST application account page.

### 1.1.4 Use Case – Procure PFST Device

Description: A supplier purchases a PFST device.

Process:

1) The supplier clicks to access the PFST purchase page, first logging into the PFST application if necessary.

OR

1) A new supplier is automatically redirected to the PFST device purchase page after creating a user account.

2) Available PFST devices - The supplier, from the list of available PFST devices displayed, selects how many of each device he or she would like to purchase.
3) The supplier confirms his or her current selections and is brought to the alternate shipping address page (or cancels the current selections and is returned to the account page)
4) Alternate shipping address - The supplier is presented with his or her current shipping address to which the supplier may make changes as necessary
5) Payment information - The supplier confirms the shipping address and is brought to the payment page (or cancels the alternate shipping address entry process and is returned to the account page)
6) The supplier enters his or her payment information and is brought to the purchase confirmation page (or cancels the payment information entry process and is returned to the account page)

7) Purchase confirmation – The supplier clicks "Confirm" to initiate the purchase and third-party payment transaction process, and is brought to the purchase transaction result page (or cancels the purchase confirmation process and is returned to the account page)
8) Third-party payment – The payment is processed by a third-party application.
9) Purchase transaction result – Appropriately shows whether the payment was completed successfully or not.
    a. Payment succeeded page - Appropriate text including button to bring supplier back to account page.
    b. Payment failed page – Appropriate text including button to bring supplier back to account page.

### 1.1.5 Use Case – Manage Device Stock

Description: A supplier views and/or maintains the PFST devices he or she owns.

Process:
1) The supplier logs in if necessary and accesses his or her MyDevices page.
2) If necessary, the supplier maintains his or her devices, including:
    - Updating the status of a device (possible statuses: ready for use, in shipment, lost, decommissioned)
    - Adding a comment to the comment log

### 1.1.6 Use Case – Create a Measurement Instruction

Process:
1) A supplier accesses the screen to create a measurement instruction, logging into the PFST application if necessary
2) The supplier creates a measurement instruction (e.g. measure the temperature of this shipment every 1 hour AND measure the location of this shipment every hour).
3) If desired, the supplier also stores this measurement instruction for use for future shipments.

### 1.1.7 Use Case – CRUD Preset Instructions/Definitions

Description: A supplier creates, reads, updates or deletes a preset measurement instruction or alert definition.

Process:
1) The supplier accesses the screen to CRUD a preset measurement instruction or alert definition, logging into the PFST application if necessary
2) The supplier is presented with a list of any previously created measurement instructions/alert definitions.
3) The supplier either
    a. Clicks on the link to create a measurement instruction/alert definition, and is brought to the create measurement/alert definition page OR
    b. Clicks on a previously created measurement instruction/alert definition to read/modify/delete the definition, and is brought to the measurement/alert detail page OR

c. Clicks on the appropriate link to delete a previously created measurement instruction/alert definition.

### 1.1.8 Use Case – Provision Shipment

Description: A supplier provisions a shipment, including associating a device with the shipment.

Process:

1) The supplier creates a record for the given shipment.
2) The supplier, in the application, associates the measurement instructions with the shipment.
3) The supplier, in the application, associates any desired alert definitions with the shipment.
4) The supplier uploads measurement instructions to the device, so the device knows what measurements (e.g. temperature, humidity) are to be taken and at what intervals (e.g. every hour, every 15 minutes, etc.)
5) The supplier, in the application, associates the device with the given shipment.
6) Oyster supplier: The supplier, in the application, associates harvest data (e.g. harvest location, date/time oysters were harvested, date/time oysters were iced, etc.) with the shipment (also see "Use Case – Scan Harvest Tag")
7) The supplier physically associates the device with the shipment (e.g. attaches the device to a bag of oysters, attaches the device to a shipment case, etc.)
8) When the actual shipment begins, the supplier triggers the device to begin reading (see "Use Case – Take Reading")

### 1.1.9 Use Case – Take Reading

Description: A supplier with a mobile device, customer with a mobile device or fixed reader takes a reading from a PFST device associated with an ongoing shipment.

Process:

1) The supplier/customer takes a reading using a mobile device (at a minimum, it is expected that the supplier will take a reading at the start of the shipment and that the customer will take a reading upon completion of the shipment).

OR

1) A fixed reader installed within the vicinity of the PFST device takes readings at the intervals specified in the measurement instructions associated with the given shipment.
2) An alert is displayed to the supplier/customer or in the reader if a measurement(s) exceeds a warning limit or permissible limit.
3) The option to complete the shipment in the PFST application should also be easily available to the user/device in the event this reading is the final reading for the shipment (i.e. the shipment is complete).

Notes:

- This reading can be (and most likely is) a set of multiple readings which have accumulated since the last time the device was read for the given shipment.

- Remaining battery power should also be read along with the given measurements.

### 1.1.10 Use Case – Track Shipment

Description: A supplier or customer views the measurement data which has been logged thus far for a specific shipment.

Process:

1) The supplier/customer accesses the given shipment, logging into the PFST application if necessary.
2) The measurement data are displayed sorted ascending by time the measurement was taken.

### 1.1.11 Use Case – Complete Shipment

Description: The shipment arrives at the expected customer location and the customer indicates the shipment as complete in the PFST application.

Process:

1) The customer accesses the complete shipment screen, logging into the PFST application if necessary.
2) The customer indicates the shipment is complete(, in the process accepting/rejecting the shipment?).

### 1.1.12 Use Case – Return Device

Description: A customer returns the PFST device to a supplier after a given shipment is completed.

Process:

1) The customer collects PFST device(s) associated with completed shipments and returns these to the supplier (e.g. in a self-addressed mailer already provided by the supplier).
2) The supplier receives the device(s) and indicates in the PFST application that they have been returned.

### 1.1.13 Use Case – View PFST Device Data

Description: A supplier views data associated with a PFST device. The supplier can do this either by searching for the device within the PFST application and accessing its data, or automatically by calling up the data associated with a device in the supplier's vicinity.

Process:

1) The supplier accesses the data in the PFST application associated with a device in the supplier's vicinity

OR

1) The supplier accesses the data in the PFST application associated with a device by:
    - accessing the device overview page and clicking on the appropriate device to call up the device's detail page
    - searching for the device in the device search screen
    - clicking on the device associated with a shipment

### 1.1.14 Use Case – CRUD Harvest Info (Oyster Supplier)

Description: An oyster supplier enters information about an oyster harvest into the PFST application, so it can be associated with a shipment.

Process:

### 1.1.15 Use Case – Scan Harvest Tag (Oyster Supplier)

Description: An oyster supplier scans the number on the plastic tag attached to a bag of oysters (required by law to track information related to the oyster harvest), so it can be associated with an oyster harvest information record and with a device/shipment.

Process:

### 1.1.16 Use Case – View Food History

Description: A consumer accesses the history (shipment values measured as well as any supplier-specific information available such as oyster harvest information) associated with food he or she is purchasing.

Process:

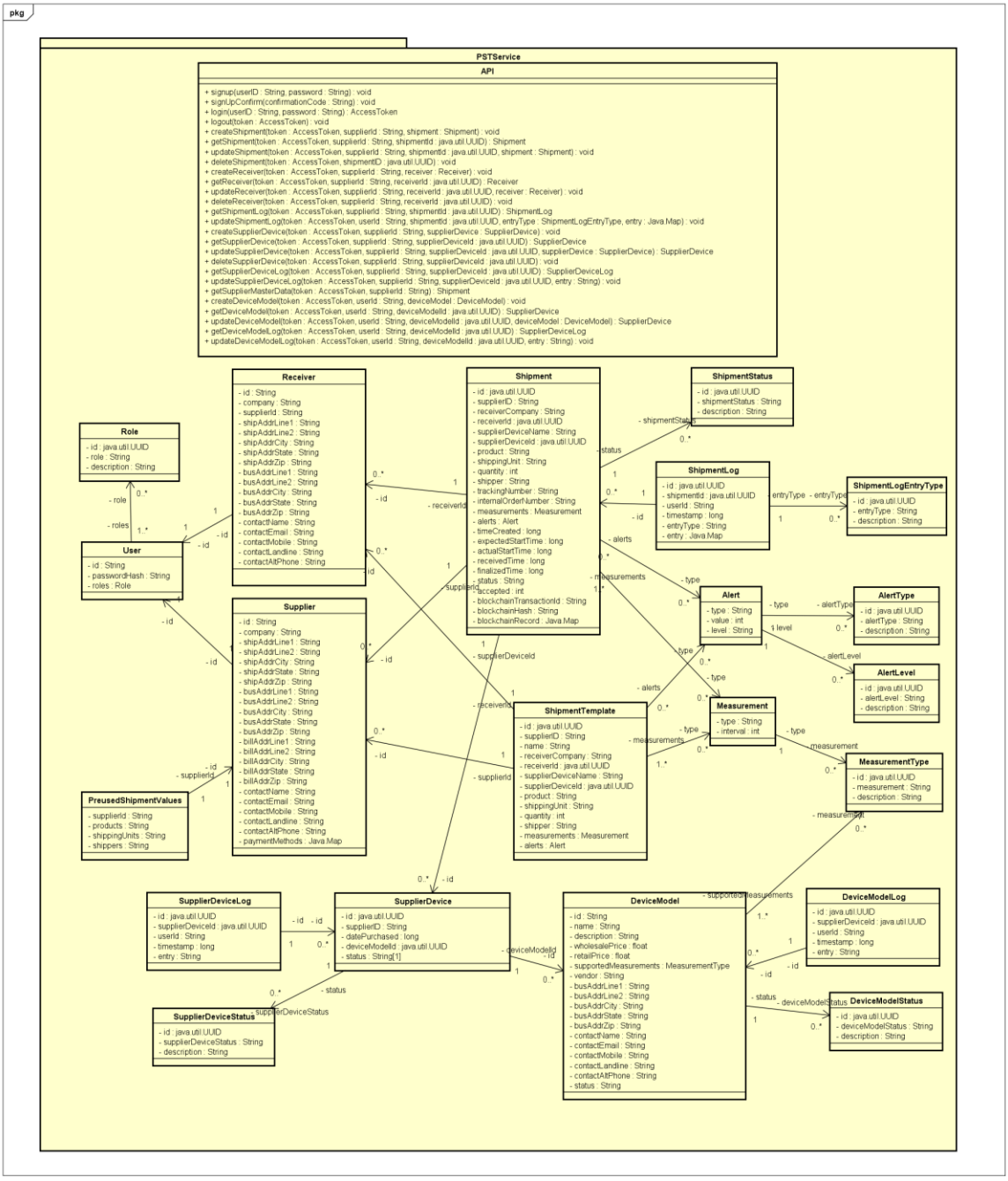
# Chapter VII.

# Implementation

## PST Service

Main service for the perishable shipping tracking application.

Class Diagram



Class Dictionary

API

API calls for main PST Service.

| Name | Signature | Description |
|---|---|---|
| signup | (userId:String, password:String) | Registers a new user with the PST service. |
| signupConfirm | (confirmationCode:String) | Confirms new user registration with the PST service. |
| login | (userId:String, password:String) | Logs a user into the PST service. |
| logout | (token:AccessToken) | Logs a user out from the PST service. |
| createShipment | (token:AccessToken, supplierId:String, shipment:Shipment) | Creates a new shipment record for the supplier with the specified supplierID, using the values in the shipment object. |
| getShipment | (token:AccessToken, supplierId:String, shipmentId:UUID) | Gets the shipment record associated with the specified supplier ID and shipment ID. |
| updateShipment | (token:AccessToken, supplierId:String, shipmentId:UUID, shipment:Shipment) | Updates the shipment record associated with the specified supplier ID and shipment ID, using the values in the shipment object. |
| deleteShipment | (token:AccessToken, supplierId:String, shipmentId:UUID) | Deletes the shipment record associated with the specified supplier ID and shipment ID. |
| createReceiver | (token:AccessToken, supplierId:String, receiver:Receiver) | Creates a new receiver record for the supplier with the specified supplierID, using the values in the receiver object. |
| getReceiver | (token:AccessToken, supplierId:String, receiverId:UUID) | Gets the receiver record associated with the specified supplier ID and receiver ID. |
| updateReceiver | (token:AccessToken, supplierId:String, receiverId:UUID, receiver:Receiver) | Updates the receiver record associated with the specified supplier ID and receiver ID, using the values in the receiver object. |
| deleteReceiver | (token:AccessToken, supplierId:String, receiverId:UUID) | Deletes the receiver record associated with the specified supplier ID and receiver ID. |
| getShipmentLog | (token:AccessToken, supplierId:String, shipmentId:UUID) | Gets the shipment log entries associated with the specified supplier ID and shipment ID. |
| updateShipmentLog | (token:AccessToken, userId:String, shipmentId:UUID, entryType:ShipmentLogEntryType, entry:Map) | Updates the shipment log entry associated with the specified supplier ID and shipment ID, using the values in the entry object. |
| createSupplierDevice | (token:AccessToken, supplierId:String, supplierDevice:SupplierDevice) | Creates a new supplier device record for the supplier with the |

| | | |
|---|---|---|
| | | specified supplierID, using the values in the supplierDevice object. |
| getSupplierDevice | (token:AccessToken, supplierId:String, supplierDeviceId:UUID) | Gets the supplier device record associated with the specified supplier ID and supplier device ID. |
| updateSupplierDevice | (token:AccessToken, supplierId:String, supplierDeviceId:UUID, supplierDevice:SupplierDevice) | Updates the supplier device record associated with the specified supplier ID and supplier device ID, using the values in the supplierDevice object. |
| getSupplierDeviceLog | (token:AccessToken, supplierId:String, supplierDeviceId:UUID) | Gets the supplier device log entries associated with the specified supplier ID and supplier device ID. |
| updateSupplierDeviceLog | (token:AccessToken, supplierId:String, supplierDeviceId:UUID, entry:String) | Updates the supplier device log entry associated with the specified supplier ID and supplier device ID, using the values in the entry object. |
| getSupplierMasterData | (token:AccessToken, supplierId:String) | Gets the master data associated with the supplier with the specified supplier ID. |
| createDeviceModel | (token:AccessToken, userId:String, deviceModel:DeviceModel) | Creates (on behalf of the administrator with user ID userID) a new device model using the values in the deviceModel object. |
| getDeviceModel | (token:AccessToken, userId:String, deviceModelId:UUID) | Gets (on behalf of the administrator with user ID userID) the device model associated with the specified device model ID. |
| updateDeviceModel | (token:AccessToken, userId:String, deviceModelId:UUID, deviceModel:DeviceModel) | Updates (on behalf of the administrator with user ID userID) the device model record associated with the specified device model ID, using the values in the deviceModel object. |
| getDeviceModelLog | (token:AccessToken, userId:String, deviceModelId:UUID) | Gets (on behalf of the administrator with user ID userID) the device model log entries associated with the specified supplier ID and supplier device ID. |
| updateDeviceModelLog | (token:AccessToken, userId:String, deviceModelId:UUID, entry:String) | Updates (on behalf of the administrator with user ID userID) the device model log entry associated with the specified device model ID, using the values in the entry object. |

User

Table of PST application users.

| Property name | Type | Description |
|---|---|---|
| id (= email) | String | User id (= user's email address). |
| passwordHash | String | Hash of user's password. |
| roles | Map | Array of user's roles. |

Suppliers

Table of suppliers registered to use PST application.

| Property name | Type | Description |
|---|---|---|
| id | String | Supplier's user ID. |
| company | String | Name of supplier's company. |
| shipAddrLine1 | String | Supplier's shipping address line 1. |
| shipAddrLine2 | String | Supplier's shipping address line 2. |
| shipAddrCity | String | Supplier's shipping address city. |
| shipAddrState | String | Supplier's shipping address state. |
| shipAddrZip | String | Supplier's shipping address zip code. |
| busAddrLine1 | String | Supplier's business address line 1. |
| busAddrLine2 | String | Supplier's business address line 2. |
| busAddrCity | String | Supplier's business address city. |
| busAddrState | String | Supplier's business address state. |
| busAddrZip | String | Supplier's business address zip code. |
| billAddrLine1 | String | Supplier's billing address line 1. |
| billAddrLine2 | String | Supplier's billing address line 2. |
| billAddrCity | String | Supplier's billing address city. |
| billAddrState | String | Supplier's billing address state. |
| billAddrZip | String | Supplier's billing address zip code. |
| contactName | String | Name of supplier contact. |
| contactEmail | String | Email of supplier contact. |
| contactMobile | String | Mobile phone number of supplier contact. |
| contactLandline | String | Landline number of supplier contact. |
| contactAltPhone | String | Alternate phone number for supplier contact. |
| paymentMethod | Map | Supplier's payment methods (credit card and/or PayPal) |

Receivers

Table of receiving customers associated with suppliers registered to use PST application.

| Property name | Type | Description |
|---|---|---|
| id | String | Receiver's user ID. |
| supplierId | String | User ID of supplier associated with receiver. |
| company | String | Name of receiver's company. |

| | | |
|---|---|---|
| shipAddrLine1 | String | Receiver's shipping address line 1. |
| shipAddrLine2 | String | Receiver's shipping address line 2. |
| shipAddrCity | String | Receiver's shipping address city. |
| shipAddrState | String | Receiver's shipping address state. |
| shipAddrZip | String | Receiver's shipping address zip code. |
| busAddrLine1 | String | Receiver's business address line 1. |
| busAddrLine2 | String | Receiver's business address line 2. |
| busAddrCity | String | Receiver's business address city. |
| busAddrState | String | Receiver's business address state. |
| busAddrZip | String | Receiver's business address zip code. |
| contactName | String | Name of receiver contact. |
| contactEmail | String | Email of receiver contact. |
| contactMobile | String | Mobile phone number of receiver contact. |
| contactLandline | String | Landline number of receiver contact. |
| contactAltPhone | String | Alternate phone number for receiver contact. |

Shipments

Table of shipments created by a PST application supplier.

| Property name | Type | Description |
|---|---|---|
| id | String | Unique ID for shipment. |
| supplierId | String | User ID of supplier who created shipment. |
| receiverCompany | String | Receiving company associated with shipment. |
| receiverId | String | User ID of receiving company associated with shipment. |
| supplierDeviceName | String | Name of supplier device associated with the shipment. |
| supplierDeviceId | String | Unique ID of supplier device associated with the shipment. |
| product | String | Product being shipped. |
| shippingUnit | String | Product shipping unit. |
| quantity | Number | Number of shipping units in shipment. |
| shipper | String | Shipping company handling the shipment. |
| trackingNumber | String | Tracking number assigned to the shipment by shipping company. |
| internalOrderNumber | String | Internal order number assigned to the shipment by the supplier. |
| measurements | String[] | Measurements to be taken for the shipment (array of [type, interval]). |
| alerts | String[] | Alerts defined for the shipment (array of [type, limit value, alert level]). |
| dateCreated | Number | Time shipment was created. |
| expectedStartTime | Number | Time shipment is expected to begin. |
| actualStartTime | Number | Time shipment actually began. |
| receivedTime | Number | Time shipment was received. |
| finalizedTime | Number | Time shipment was finalized (and readings subsequently written to blockchain). |
| status | String | Current status of shipment. |
| accepted | Number (boolean) | Boolean indicating whether or not shipment was accepted. |
| blockchainTransactionId | String | Blockchain transaction ID. |

| | | |
|---|---|---|
| blockchainHash | String | Blockchain hash. |
| blockchainRecord | blob | Readings record stored in blockchain (JSON object). |

ShipmentLog

Table of entries in log for a shipment (each shipment has one to many log entries).

| Property name | Type | Description |
|---|---|---|
| id | String | Unique ID for shipment log entry. |
| shipmentId | String | Unique ID for shipment. |
| userId | String | ID of user who created this entry. |
| timestamp | Number | Time user created this entry. |
| entryType | String | Entry type (comment, readings upload, etc.). |
| entry | Map | Actual entry (JSON based on entry type). |

ShipmentTemplates

Table of shipment templates associated with a supplier (zero to many).

| Property name | Type | Description |
|---|---|---|
| id | String | Unique ID for template. |
| supplierId | String | Supplier's user ID. |
| name | String | Name assigned to template. |
| receiverCompany | String | Template receiving company. |
| receiverId | String | User ID associated with template receiving company. |
| supplierDeviceName | String | Template supplier device name. |
| supplierDeviceId | String | Template supplier device ID. |
| product | String | Template product. |
| shippingUnit | String | Template shipping unit. |
| quantity | Number | Template quantity. |
| shipper | String | Template shipper. |
| measurements | String[] | Template measurement instructions. |
| alerts | String[] | Template alerts. |

PreusedShipmentValues

Table of shipment values (product, shipping unit and shipper) previously used by a supplier when creating shipments (these values were previously entered manually but will now automatically appear as radio button selections).

| Property name | Type | Description |
|---|---|---|
| supplierId | String | Supplier's user ID. |
| products | String[] | Array of products used in previous shipments created by supplier. |

| | | |
|---|---|---|
| shippingUnits | String[] | Array of shipping units used in previous shipments created by supplier. |
| shippers | String[] | Array of shippers used in previous shipments created by supplier. |

DeviceModels

Table of device models which are/were available for purchase by suppliers.

| Property name | Type | Description |
|---|---|---|
| id | String | Unique ID for device model. |
| name | String | Name of device model. |
| description | String | Description of device model. |
| wholesalePrice | Number | Device model's wholesale price. |
| retailPrice | Number | Device model's retail price. |
| supportedMeasurements | String[] | Types of measurements possible with this device model. |
| vendor | String | Device model vendor. |
| addrLine1 | String | Device model vendor address line 1. |
| addrLine2 | String | Device model vendor address line 2. |
| addrCity | String | Device model vendor address city. |
| addrState | String | Device model vendor address state. |
| addrZip | String | Device model vendor address zip code. |
| contactName | String | Name of device model vendor contact. |
| contactEmail | String | Email of device model vendor contact. |
| contactMobile | String | Mobile phone number of device model vendor contact. |
| contactLandline | String | Landline number of device model vendor contact. |
| contactAltPhone | String | Alternate phone number for device model vendor contact. |

DeviceModelLog

Table of entries in log maintained for a device model (each device model can have zero to many log entries).

| Property name | Type | Description |
|---|---|---|
| id | String | Unique ID for log entry. |
| deviceModelId | String | Unique ID for device model. |
| userId | String | ID of user who created this entry. |
| timestamp | Number | Time this entry was created. |
| entry | String | Actual entry. |

SupplierDevices

Table of devices owned by a supplier.

| Property name | Type | Description |
|---|---|---|

| | | |
|---|---|---|
| id | String | Unique ID assigned to this supplier device. |
| supplierId | String | User ID of supplier who owns this device. |
| datePurchased | String | Time supplier acquired this device. |
| deviceModelId | String | Unique ID assigned to device model of which this device is an instance. |
| status | String | Current status of supplier device (active, discontinued, etc.) |

SupplierDeviceLog

Table of entries in log for a supplier device (each supplier device can have zero to many log entries).

| Property name | Type | Description |
|---|---|---|
| id | String | Unique ID for log entry. |
| supplierDeviceId | String | Unique ID assigned to supplier device. |
| userId | String | ID of user who created this entry. |
| timestamp | Number | Time this entry was created. |
| entry | String | Actual entry. |

Roles

Possible user roles (enum).

| Property name | Type | Description |
|---|---|---|
| id | String | ID assigned to role. |
| role | String | Role code (e.g. ADMIN, SUPPLIER, etc.) |
| description | String | Description of role. |

Measurements

Possible measurements (enum).

| Property name | Type | Description |
|---|---|---|
| id | String | ID assigned to this measurement type. |
| measurement | String | Measurement type code (e.g. GPS, TEMPERATURE, etc.) |
| description | String | Further description of this measurement type. |

AlertTypes

Possible alert types (enum).

| Property name | Type | Description |
|---|---|---|
| id | String | ID assigned to this alert type. |

| | | |
|---|---|---|
| alertType | String | Alert type code (e.g. MIN_TEMPERATURE, MAX_TEMPERATURE, etc.) |
| measurement | String | Measurement type with which this alert is associated (e.g. TEMPERATURE, etc.). |

AlertLevels

Possible alert levels (enum).

| Property name | Type | Description |
|---|---|---|
| id | String | ID assigned to this alert level type. |
| alertLevel | String | Alert level code (e.g. INFORMATIONAL, WARNING, CRITICAL, etc.) |
| description | String | Further description of this alert level. |

ShipmentStatuses

Possible shipment statuses (enum).

| Property name | Type | Description |
|---|---|---|
| id | String | ID assigned to this shipment status. |
| shipmentStatus | String | Shipment status code (e.g. PENDING, IN_TRANSIT, COMPLETED, etc.) |
| description | String | Further description of this shipment status. |

DeviceModelStatuses

Possible device model statuses (enum).

| Property name | Type | Description |
|---|---|---|
| id | String | ID assigned to this device model status. |
| deviceModelStatus | String | Device model status code (e.g. AVAILABLE, DISCONTINUED, etc.) |
| description | String | Further description of this device model status. |

SupplierDeviceStatuses

Possible supplier device statuses (enum).

| Property name | Type | Description |
|---|---|---|
| id | String | ID assigned to this supplier device status. |
| supplierDeviceStatus | String | Device model status code (e.g. AVAILABLE, DISCONTINUED, etc.) |
| description | String | Further description of this supplier device status. |

ShipmentLogEntryTypes

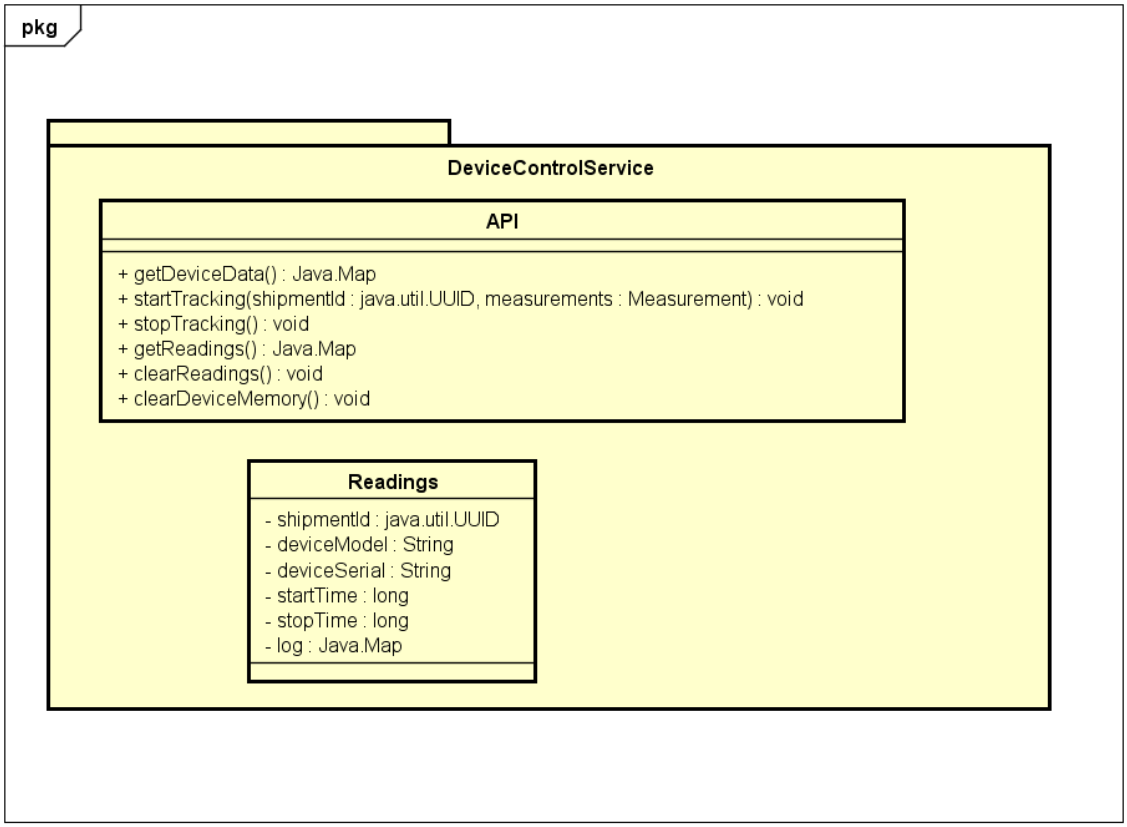
Possible shipment log entry types (enum).

| Property name | Type | Description |
|---|---|---|
| id | String | ID assigned to this shipment log entry type. |
| entryType | String | Shipment entry type code (e.g. COMMENT, READINGS, etc.) |
| description | String | Further description of shipment log entry type. |

## Device Control Service

Service for supplier device control.

Class Diagram




Class Dictionary

API

Possible Device Control API calls.

| Name | Signature | Description |
|---|---|---|
| getDeviceData | ():Object | Returns data associated with this device (serial number, etc.). |
| startTracking | (shipmentId:UUID, measurements:Measurement[]):void | Command for device to start tracking the measurements specified in measurements for shipment shipmentId. |
| stopTracking | ():void | Command for device to stop tracking. |
| getLog | ():Object | Returns reading data stored on this device. |

| | | |
|---|---|---|
| clearLog | ():void | Clears device log. |
| clearDeviceMemory | ():void | Clears all device memory. |

Readings

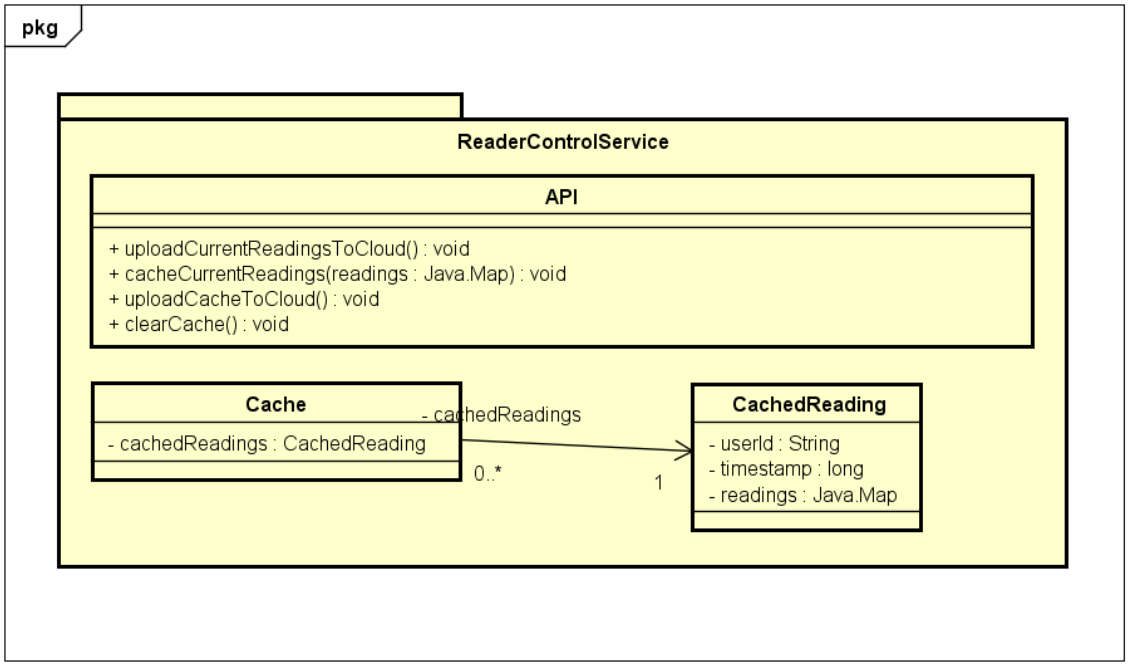
Current readings stored on device.

| Property name | Type | Description |
|---|---|---|
| shipmentId | UUID | Unique ID of shipment for which device is taking readings. |
| deviceModel | String | Device model. |
| deviceSerial | String | Device serial number. |
| startTime | long | Time readings began to be taken. |
| stopTime | long | Time readings stopped being taken. |
| log | Object | Actual readings (array of [{type, readings[]}]) |

## Reader Control Service

Service for reader control.

Class Diagram




Class Dictionary

API

Possible Reader Control API calls.

| Name | Signature | Description |
|---|---|---|
| uploadCurrentReadingsToCloud | ():void | Uploads readings of current device being scanned to cloud. |
| saveCurrentReadingsToCache | (Object):void | Stores readings of current device being scanned to this reader's cache. |
| uploadCacheToCloud | ():void | Uploads cache readings to cloud. |
| clearCache | ():void | Clears this reader's cache. |

Cache

Reader cache.

| Property name | Type | Description |
|---|---|---|

| | | |
|---|---|---|
| cachedReadings | CachedReading[] | Readings currently cached in this reader. |

CacheReading

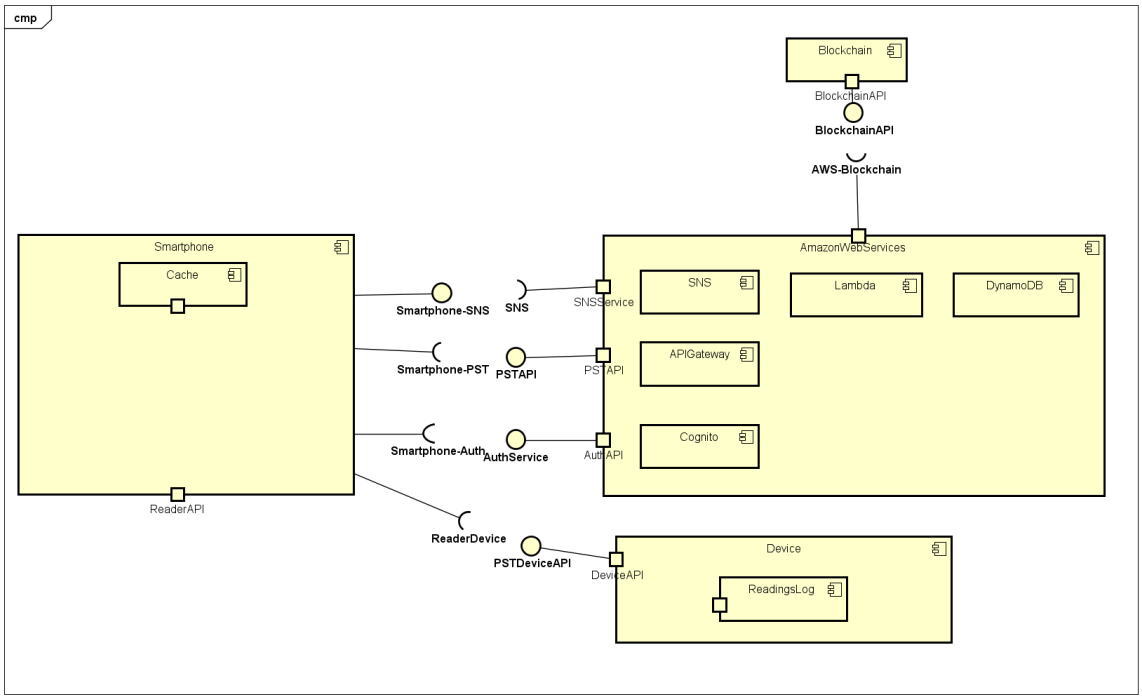
Cached reading object.

| Property name | Type | Description |
|---|---|---|
| userId | String | ID of user who cached readings. |
| timestamp | long | Time these readings were cached. |
| readings | Object | Actual readings. |

# Chapter VIII.

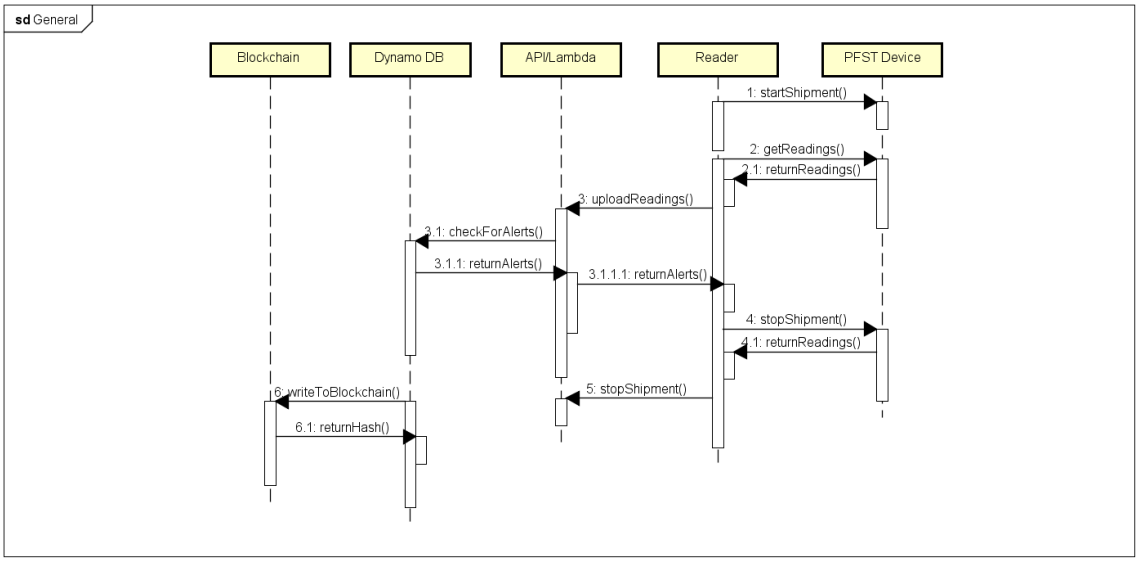
# Implementation Details

## Component Diagram

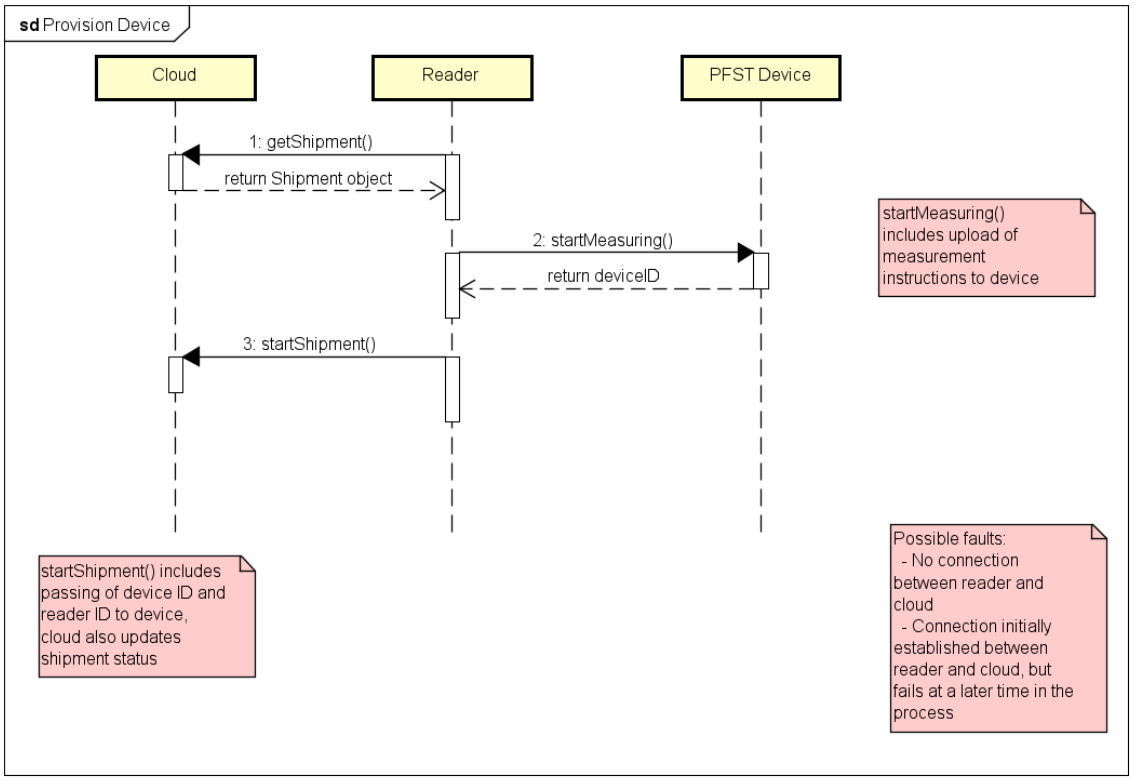



## Sequence Diagrams

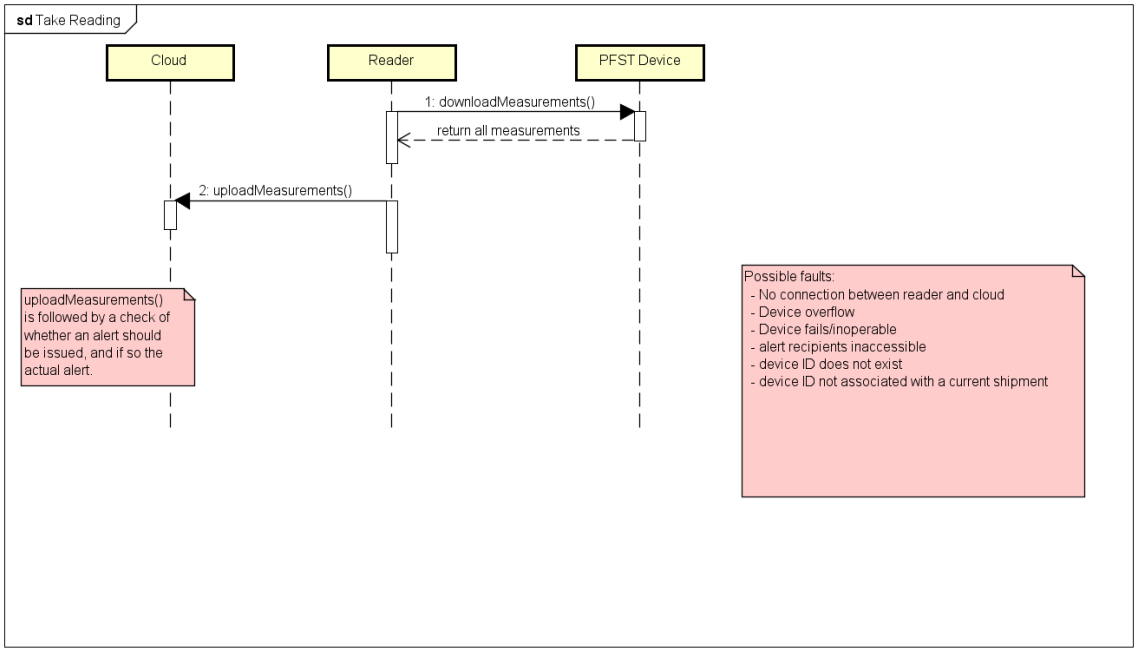
Sequence - General



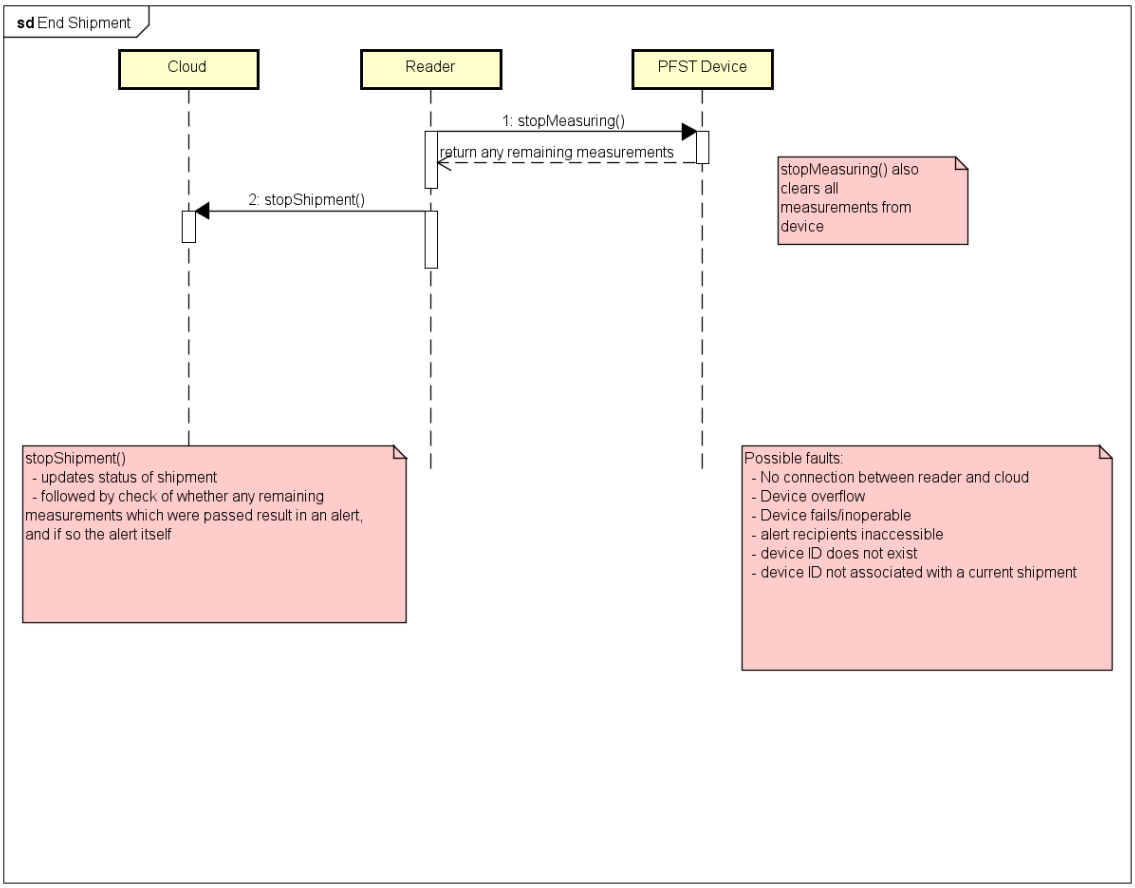

Sequence – Provisioning a Device



Sequence – Taking a Reading



Sequence – Completing a Shipment



# Chapter IX.

# Testing

I came up with a number of tests to check the puck.js/Ublox GPS combination sensor on the basis of memory capacity (RAM and flash), power consumption (various options), GPS accuracy and temperature accuracy.

## Memory Capacity

I came up with the following tests to make sure the memory capacity of the combination sensor would be adequate for Dan Ward's needs, and also to get a general idea of the maximum period of time the puck.js would be able to log data:

- RAM – Take GPS and temperature readings every hour for several days (Dan's requirements).
- RAM – Take GPS and temperature readings every 10 minutes for several days (more rigorous test).
- Flash – Take GPS and temperature readings every hour for several days (Dan's requirements).
- Flash – Take GPS and temperature readings every 10 minutes for several days (more rigorous test).
- RAM and flash – Take GPS and temperature readings every hour for several days (Dan's requirements).

- RAM and flash – Take GPS and temperature readings every 10 minutes for several days (more rigorous test).

## GPS Accuracy

I came up with the following tests to measure the accuracy of the Ublox GY-NEO6MV2 unit:

- Take GPS readings every 3 minutes with the Ublox for every several hours over the course of a car ride, at the same time taking GPS readings with another sensor (in this case the GPS Logger application on my Android smartphone) so that results can be compared.
- Repeat this test with a second Ublox unit to confirm the accuracy of the device in general.

## Temperature Accuracy

I came up with the following tests to measure the accuracy of the puck.js temperature sensor:

- Take temperature readings with a puck.js every 5 minutes for 24 hours, varying temperature during certain periods (e.g. by putting the sensor in a refrigerator, next to a space heater, etc. for several hours), at the same time taking temperature readings with another sensor (in this case the Elitech RC-5 Data Logger) so that results can be compared.

- Repeat this test with a second puck.js to confirm the accuracy of the device in general.

## Battery Life

I came up with the following tests, based on the power options suggested in the Espruino forum conversation "GPS Module not sending data on Puck.js" mentioned above, to check the battery life of the puck.js/Ublox GPS combination sensor:

- puck.js w/ CR 2032 + Ublox w/ 3.7 V 350 mAh LiPo – Log GPS and temperature every hour for several days; log GPS and temperature every 10 minutes for several days.
- puck.js w/ CR 2032 + Ublox w/ 3.7 V 1200 mAh LiPo – Log GPS and temperature every hour for several days; log GPS and temperature every 10 minutes for several days.
- puck.js w/ CR 2032 + Ublox w/ 3 AA cell battery pack – Log GPS and temperature every hour for several days; log GPS and temperature every 10 minutes for several days.
- puck.js and Ublox both powered w/ 3.7 V 350 mAh LiPo – Log GPS and temperature every hour for several days; log GPS and temperature every 10 minutes for several days.
- puck.js and Ublox both powered w/ 3.7 V 1200 mAh LiPo – Log GPS and temperature every hour for several days; log GPS and temperature every 10 minutes for several days.

The above tests should be repeated separately with a duplicate puck.js device, a duplicate Ublox GPS unit and duplicate batteries to confirm the accuracy of results.

In addition, these tests can simultaneously check and confirm memory capacity limits, assuming this won't change test conditions.

# Chapter X.

# Risks

Unfortunately, the platform does come with some risks. Although the most important are related to the puck.js device and could therefore be avoided altogether just by using a different device, some risks are related to other system components.

## Web Bluetooth connection failure

An inability for a reader to connect to a puck.js device (or other device that is accessed through Web Bluetooth) could result in lost readings.

## puck.js power failure

Should the battery powering a puck.js run down, this will result in loss of the device's data, which are lost upon startup. It must be ensured that a puck.js will have sufficient power for an entire shipment.

## Unauthorized access to puck.js

A hacker who manages to gain access to the Espruino interpreter could read, or even worse tamper with, shipment readings. Measures should be taken to secure this data as best as possible.

## Lack of reader Internet connection

An Internet connection might not be readily available to a reader; as a result, a reader might not be able to promptly upload crucial data (e.g. a shipment which has reached critical temperature). In addition, a missing mobile connection could prevent a smartphone from receiving a shipment alert.

### Service unavailability

Cloud services might be temporarily unavailable. This would be a problem especially in the case of AWS SNS availability; provisions should also be made in the case of blockchain service unavailability.

# Chapter XI.

## Results and Evaluation

Unfortunately, due to time constraints, I was only able to carry out six tests, two of which I have no readings for because the puck.js failed for various reasons and the logged data was lost when I restarted the device. However, even with this test data I was able to draw some conclusions.

### GPS 3.7V 350 mAh LiPo battery capacity test (Appendix 2)

In this test, the puck.js was powered with a CR 2032 cell battery and the Ublox GPS module was powered with a 3.7V 350 mAh lithium polymer battery, and I logged both GPS location and temperature to puck.js RAM every 10 minutes for approximately 12 hours. The main purpose of this test was to determine how well a 350 mAh lithium polymer battery could power the Ublox GPS module, but I took advantage of the test to assess the capacity of the CR 2032 battery when used to determine the power the puck.js and also to get an idea of how well RAM would hold out when logging data over an extended period of time.

3.7V 350 mAh LiPo capacity – Unfortunately the GPS unit stopped working before the test was fully completed; it did last about 11.5 hours. As a result, the 350 mAh LiPo battery appears to be unsuitable to power the GPS unit; it might possibly work for short trips (given Dan's requirements of one reading every hour, it would last for 2 to 3 days).

CR 2032 capacity – The CR 2032 appeared to work relatively well in powering the puck.js; its capacity dropped from 3.01 volts to 2.87 volts (percentage drop from 90 to 79). Based on a web discussion "Testing a 3.7 v 150 mAh LiPo" (https://www.rcgroups.com/forums/showthread.php?1694183-Testing-a-3-7v-150mAh-LiPo), which states a battery should not be used after it has only 20% capacity, I estimated that under these conditions (GPS and temperature logged every 10 minutes), the puck.js would last about three and a half days, or under Dan's conditions, about 3 weeks.

RAM – RAM capacity was no problem; RAM capacity started at 65536 bytes and finished at 63964 bytes, with a single GPS/temperature reading taking up 25 bytes. Under the test conditions, RAM would be used up after a little more than 18 days; under Dan's conditions RAM would be used up after 109 days.

GPS 3.7V 1200 mAh LiPo battery capacity test (Appendix 3)

In this test, the puck.js was powered with a CR 2032 cell battery and the Ublox GPS module was powered with a 3.7V 1200 mAh lithium polymer battery, and I logged both GPS location and temperature to puck.js RAM every 10 minutes for approximately 12 hours. The main purpose of this test was to determine how well a 1200 mAh lithium polymer battery could power the Ublox GPS module, but I took advantage of the test to also assess the capacity of the CR 2032 battery when used to determine the power the puck.js.

3.7V 1200 mAh LiPo capacity – Unsurprisingly, the 1200 mAh battery worked much better than the 350 mAh battery and lasted the entire test; its starting capacity was 4.17

volts and it finished at 3.09 volts after a little more than 12 hours. However, according to the battery web discussion mentioned above, the battery probably should not be used further and should be recharged at this point. Under Dan's conditions, the GPS unit would work for about 3 days when charged with this 1200 mAh LiPo battery.

CR 2032 capacity – The capacity of the CR 2032 cell battery used to power the puck.js dropped from 2.98 volts to 2.71 volts (percentage drop from 95 to 70).

## GPS Accuracy Test (Appendix 4)

In this test I took GPS readings with the Ublox every 3 minutes for several hours as I drove on highways and streets through central Vermont and New Hampshire. At the same time, I logged readings to the GPS Logger application on my smartphone.

The Ublox GY-NEO6MV2 GPS unit worked extremely well in terms of accuracy. Its readings were very close to the readings shown by the GPS Logger application. In addition, a spot check of Ublox values by entering them in Google Maps confirmed it was the route I took, including specific streets I drove on and even stores I passed.

3.7V 1200 mAh LiPo capacity – Just as added information on battery capacity, I also logged the capacity of the 3.7V 1200 mAh lithium polymer battery I used to power the GPS for the test; the battery began at 4.18 volts and finished at 3.87 volts (again, with GPS measurements being logged every 3 minutes for about 2 hours and 45 minutes).

CR 2032 capacity – I also noted the capacity of the puck.js CR 2032 cell battery over the 2 hour 45 minute trip, it began at 3.0 volts and ended at 2.86 volt (percentage drop from 97 to 79).

Temperature Accuracy Test (Appendix 5)

Finally, I ran an approximately 10 hour test to determine the accuracy of the puck.js temperature sensor. I took temperature readings at 5 minute intervals, and put the puck.js in my home office, in the refrigerator, next to a space heater, and outdoors, each at multiple hour intervals. An Elitech RC-5 temperature logger accompanied the puck.js at all times to confirm the accuracy of the puck.js temperature readings.

Unfortunately, puck.js temperature readings differed greatly from those logged by the Elitech RC-5; in addition, the differences themselves varied, even taking into account temperature changes as I brought the units into a different environment. I posted to an Espruino forum (http://forum.espruino.com/conversations/328770/?utm_campaign=mentioned&utm_medium=email&utm_source=notification#comment14800691) and asked puck.js developer Gordon Williams about the temperature sensor, and he wrote that although most units require some calibration (which can vary from unit to unit), the value used for calibration should be consistent, and that when calibrated the puck.js takes reliable temperature. More extensive testing is definitely required to see if these temperature deviations were indeed an isolated case, or possibly due to the Elitech sensor I used for comparison.

CR 2032 capacity – For added information on battery capacity I also noted the capacity of the puck.js CR 2032 cell battery over the course of the test, in terms of actual voltage it went from 2.96 volts to 2.81 volts.

puck.js failure

The puck.js inexplicably just stopped working in several tests, this was not due to a loss of battery power since the unit started up again as soon as I re-inserted the battery. More extensive testing is required to determine if this failure is a consistent situation, is related to a high level of logging over many hours or was specific to the unit I used (in addition to the possibility of an exceptional faulty unit, my soldering skills are limited so I may have damaged the unit when I was soldering it to the Ublox GPS module). A quick web search did not uncover any cases of such failure, but it goes without saying that if such failure is consistent it would definitely detract from the puck.js' value as a logger/sensor since logged data are lost when the unit is re-started.

# Chapter XII.

# Summary and Conclusions

All in all I feel I succeeded in providing a cost-effective, versatile, barebones solution for suppliers to track their perishable shipments on the basis of conditions such as location, temperature, humidity, etc., and which ensures accountability not only through the readings themselves, but by storing the readings in a blockchain. In the process of developing the solution, I also uncovered areas which would benefit from further work; these additional developments would not only benefit the application itself, but would also further the technologies of IoT, Web Bluetooth and Espruino in general.

Despite these contributions, my final application was unfortunately not able to attain all the goals I'd hoped it would meet (although I hope this learning and experience is a contribution itself in that it makes a statement on the suitability/non-suitability of devices/approaches, and whether an area would benefit from further work). In addition, there were definitely some things I would have done differently were I to start my thesis project over again from scratch.

## Contributions / Goals Attained

Following is a list of the goals I feel my perishable shipment tracking application met, and contributions I made to the fields of perishable shipping and information technology.

Barebones

The PST application does provide an effective “barebones” solution to track perishable shipments; the only costs to the supplier are for sensors (which themselves are cost-effective) and for use of the application, which can be kept well below those of comparable applications (such as the Sendum).

Sensor versatility

Although the solution focuses on sensors which support Web Bluetooth and Espruino (since they are themselves versatile and also easy to use, as they don’t need to be physically attached to a reader), the application is intended to be easily extended to accommodate any sensor type.

Reader/application device versatility

Although the solution is optimized for mobile, it is a browser-based application and can therefore easily be used on many devices (smartphone, laptop, etc.)

Other versatility

The solution provides fields which allow shipment records to be linked to specific supplier applications (i.e. the internal order number) and supplier shippers (shipper and tracking number fields).

Ease of use

The following elements make the PST solution easy to use:

- mobile-optimized – The application can be run from any smartphone or other device.
- browser-based – The solution does not need to be installed; suppliers just need to sign up for the service from their browsers and they're on their way.
- sensors – Sensors, especially in the case of Espruino sensors, are instantly ready to use at the touch of a button, require absolutely no configuration and do not need to be physically attached to the reader in order to upload readings.
- templates/preused values – The solution's system of templates and having shipment values (product, shipping unit, shipper) which were previously used automatically presented later as radio button selections make it very easy to create shipments.
- ease in defining measurements and alerts

Accountability / reduction in paperwork

As planned, the sensor readings themselves as well as their storage on a blockchain ensure accountability of all parties to a shipment, and ease paperwork by making these readings easily accessible.

Alerts

The solution's configurable system of alerts (easily implemented through AWS SNS) increase shipment integrity and reduce costs.

Ease of use for application administrator

The solution is also easy for administrators to maintain; its serverless nature means there is less work involved in scaling the application to the current number of suppliers and other users.

It is also conceivable that administrators wouldn't have to be involved at all as a middleman for device purchases; depending on what their requirements are, suppliers themselves could purchase an appropriate Espruino device and automatically control it using the JavaScript code provided by the solution.

Ublox sensor quality and price

Although I was unable to meet Dan Ward's requirements of an effectively disposable sensor unit, I definitely feel my thesis project showed that the Ublox GY-NEO6MV2 6M GPS NEO module provides more than ample accuracy at a very low cost ($12.45).

puck.js as a logger

In addition, although it definitely did not meet all of the goals of my thesis project, my project showed that in terms of perishable shipments, the puck.js at least has value as a cost-effective data logger and in general is a very versatile device. It would probably benefit from a power source different from the CR 2032 cell battery, although I feel it could be used with that battery for short-term shipments. Its effectiveness as a temperature sensor is uncertain and must be determined through additional tests.

### Sensor case compactness

Finally, as an ancillary observation that would definitely require more confirmation, based on the compact size of the puck.js and Ublox GPS sensor, I feel perishable shipment sensor cases could be much more compact than those generally used, especially for shipments of shorter periods. I was able to fit both devices, along with a 3.7 V 1200 mAh polymer lithium battery, into a 3.15” x 1.58” x 0.79” Hammond project box (described above).

## Unattained Goals

### puck.js as a temperature sensor

Although the number of temperature tests I ran on the puck.js were limited and more testing is definitely called for, it is quite possible the device is not suitable for use as a temperature sensor for perishable shipments. Although its developer Gordon Williams did note that the device does require calibration, the differences in puck.js temperature readings with those taken with a different unit themselves varied, and (assuming this is consistent behavior) it would not be possible to correct this just by adding or subtracting a given value to all readings. Given the versatility of the puck.js, however, it would most likely be a straightforward task to use attach a different sensor to the device and just use the puck.js as a data logger.

### Combo sensor cost-effectiveness

I was also unable to accommodate Dan Ward’s needs of a combination GPS tracker/temperature sensor which would be effectively disposable. However, I do feel I

made some progress in this area, as the combination puck.js/Ublox sensor roughly costs the same as the Queclink sensor which Dan Ward tried out at one point but didn't continue to use as its readings were inaccurate. The combination sensor is very accurate, at least in terms of GPS location.

Also, given the general state of current technology, a disposable GPS tracker/temperature sensor might not be currently feasible. In addition, receiver/customer incentives to return sensors after a shipment (discussed below) might help to allay sensor costs.

## Things I Would Have Done Differently

### Additional tests

In addition to additional runs of the tests I did define, my project would have benefited from the following additional tests:

- more extensive testing on puck itself – one of my puck.js devices failed at least twice after logging data. This was not due to battery life, since it restarted immediately after I re-inserted the battery. Of course, it would have been good to see if that failure occurred regularly, or if it was possibly a result of my inexperience with soldering.
- confirmation of all test results using additional puck.js unit(s)
- flash memory – Although I don't feel logging to puck.js flash memory (instead of RAM) would have caused any problems, I did not test this.

- sensor enclosures – I did not get to test the sensor enclosures I purchased, and should have at least tested the effect of each sensor enclosure on GPS readings and temperature readings, as well as durability, and maximum battery size possible for each enclosure.
- cable ties – I also purchased some disposable cable ties which I intended to use to attach sensors to shipments (https://www.amazon.com/Strong-Adhesive-Backed-Holders%EF%BC%8C-Screw-Hole-Provides-long%EF%BC%8850pack%EF%BC%89/dp/B07JMSWNC8/ref=sr_1_3?keywords=Strong-Adhesive-Backed+Mounts+Cable+Tie+Mounts&qid=1564681303&s=gateway&sr=8-3), and did not get an opportunity to test them.

DynamoDB

Were I to develop the PST solution from scratch, I might use a different database than DynamoDB. I found the database limited in some ways (for example, it does not provide a date/timestamp type) and also functionality to access and maintain the database was limited, especially that provided through the AWS user portal.

On the other hand, this was my first time using a NoSQL database, and I did enjoy the versatility of that approach.

## Future Work

I am considering marketing my perishable shipping solution; whether or not I do so it would benefit from the following enhancements:

Receiver/end customer incentives

As stated above, incentives to return sensor devices should be provided to supplier customers. Some of these incentives might include:

Blockchain reward – A receiver might receive, through blockchain, a small reward for uploading sensor data and for returning the device to the supplier.

End customer benefits – Supplier customers could be given incentive to return devices by providing benefits to the end customer (i.e. restaurant patrons, gourmet food customers, etc.) such as allowing them to view, through the website, the journey their food took and that the food was indeed shipped under quality conditions. Bumble Bee® Seafoods already provides a similar tool with its "Trace My Catch" web page (at https://www.bumblebee.com/tracemycatch/).

Blockchain for payment

The solution's blockchain functionality should be enhanced to make it possible for suppliers to get paid for shipment immediately when the receiver approves the shipment. In addition to speed of payment, this would bring the possibility of eliminating an intermediary (i.e. the provider used to process the payment).

puck.js security

As mentioned above, sensor devices, including the puck.js, should be made more secure. Gordon Williams has already provided methods for securing the puck.js (https://www.espruino.com/BLE+Security), and other methods should be implemented if feasible.

puck.js identification/isolation

The Web Bluetooth window which pops up for users to show them Bluetooth devices in range currently shows limited, somewhat meaningless information (i.e. the puck.js serial number and model number). If possible, this window should be enhanced so that administrators can also display more meaningful information (e.g. the shipment associated with the device) in order to make it easier for technicians to select devices and uncover possible problems.

Nordic Thingy 52 and other sensors

Although intended for use as a prototype, the Nordic Thingy 52 mentioned in the "Prior Work" section does have many features which are attractive for perishable shipping, such as a variety of sensors and support for Espruino. As a result, further work should include the development of JavaScript to control the device, as well as research into other sensors and the related development work.

Combine GPS coordinates with Google Places, maps, etc.

Currently, when GPS tracking is selected, the solution only shows, in addition to time, the actual GPS coordinates (latitude, longitude and altitude) associated with a shipment. Displaying these coordinates on a map and/or using Google Places to list the places associated with coordinates would greatly enhance the solution's usefulness to suppliers.

### Progressive web app/one page app

Finally, the solution might be more user-friendly if implemented as a progressive web app (https://developers.google.com/web/progressive-web-apps/) or single-page app instead of a traditional HTML/JavaScript app with several pages.

## Future Work of Value Beyond PST Application

Finally, based on my experience with this thesis project, I feel the general area of IoT (and not just my application) would benefit from the following further work:

### Enhanced Web Bluetooth connection interface

As I mentioned in the previous section, functionality should be added to the Web Bluetooth window to make it possible for the interface to display information (possibly pulled from some table/database which can be configured) that is more meaningful that the limited information currently displayed (device serial number and model number).

### Ublox GPS sensor quality

I was very impressed at the quality of the Ublox GPS sensor and its very low price, and I believe these factors make it possible for the sensor to be used in a variety of applications which were not possible up to this point because of cost.

### Espruino

Similarly, I was very impressed at the versatility offered by the Espruino interpreter. Not only does Espruino make it possible for devices to be used versatilely, even based on real-time conditions, the related code can be provided and maintained at a

single central location, which is greatly advantageous to companies who deploy sensors or other devices.

Finally, I believe the ability for a software provider to develop code for Espruino devices and offer it for sale without the provider also having to become involved as a middleman in device purchase might also open up new industries.

Appendix 1
Notes from December 31, 2018 meeting with Dan Ward of Ward Aquafarms
Don't create extra work for people in the process.

The phone is good for the sensor. With alerts. Mobile ready web site.

Download app.

Farmer, a shipping company, local shipping company, the local distributor, consumer.

There needs to be an added value.

Use barcode to read data and upload.

Cellular

14-day max average, 2 - 3 days, sample per hour

Who are the end customers: restaurant, fish markets, individual consumers

International, and in country

Getting the tags back was a problem.

Bag level. 100 count, 60 - 100 dollars.

Worked with a system with Verizon which was abandoned. Value of fish was not sufficient. Pharmaceuticals were better.

Input data every 7 days.

A very low-cost device that can be thrown away.

Many wholesalers, so having an access point at each would be hard.

Scan with the phone for now.

Food to the plate. Demonstrate at the local level. 2d barcode to link to the website.

The restaurant keeps the tag for 90 days.

Information: harvest date, time, time on ice, harvest number, harvest location.

Verizon used thermal cameras, provision the sensors, the thermal camera would take the temperature, the tag would be geofenced.

Restaurants who would be willing to work with us. Restaurants in Boston, Boston distributor. Googling for the oyster house. Legal Seafood. The Chart room. B&G oyster house. Union Oyster House. Wholesalers.

50 to 100 bags/week. $60 -> $130

Oysters are the best focus.  Scallops are also eaten raw.  But could be applied more broadly.

Battery coast was the problem.  Sensors were cheap. The lithium-ion battery was too expensive.

$1 dollar range

Verizon solution: Alerts to the phone, email, maps, but too expensive.

Appendix 2

Capacity test of 3.7V 350 mAh lithium polymer battery

Primary test goal: To determine capacity of a 3.7V 350 mAh lithium polymer battery when used to power the GPS module.

Secondary test goal: To determine the capacity of a 3.7 V CR 2032 coin battery used to power the puck.js.

Secondary test goal: To determine the RAM capacity of the puck.js when the device is used to record GPS and temperature at regular intervals over a period of time.

Test description: Log both GPS location and temperature to puck.js RAM every 10 minutes for several hours.

Test period: June 27, 2019 7:58 p.m. to June 28, 2019 7:58 a.m.

*Note:* The GPS module stopped working toward the end of the test, at approximately 7:45 a.m. on June 28.

| | |
|---|---|
| GPS battery level (actual*) - start | 4.17 V |
| GPS battery level (actual*) - end | 3.09 V |
| puck.js battery level (actual*) - start | 3.01 V |
| puck.js battery level (actual*) - end | 2.87 V |
| puck.js battery level (percentage**) - start | 90 |
| puck.js battery level (percentage**) - end | 79 |
| Estimated remaining RAM - start | 65536 |
| Estimated remaining RAM - end | 63964 |
| Size of a single GPS/temperature reading | 25 bytes |

* measured using multimeter

** measured using puck.js E.getBattery() command

Appendix 3

Capacity test of 3.7V 1200 mAh lithium polymer battery

Primary test goal: To determine capacity of a 3.7V 1200 mAh lithium polymer battery when used to power the GPS module.

Secondary test goal: To determine the capacity of a 3.7 V CR 2032 coin battery used to power the puck.js.

Test description: Log both GPS location and temperature to puck.js RAM every 10 minutes for several hours.

Test period: July 1, 2019 7:30 a.m. to July 2, 2019 7:59 a.m.

| | |
|---|---|
| GPS battery level (actual*) - start | 4.20 V |
| GPS battery level (actual*) - end | 3.75 V |
| puck.js battery level (actual*) - start | 2.98 V |
| puck.js battery level (actual*) - end | 2.71 V |
| puck.js battery level (percentage**) - start | 95 |
| puck.js battery level (percentage**) - end | 70 |

* measured using multimeter

** measured using puck.js E.getBattery() command

Appendix 4

GPS Accuracy Test

Primary test goal: To determine the accuracy of the Ublox GY-NEO6MV2 6M GPS NEO module used to track location.

Secondary test goal: To measure the capacity of a 3.7 V CR 2032 coin battery used to power the puck.js.

Secondary test goal: To measure the capacity of a 3.7V 1200 mAh lithium polymer battery when used to power the GPS module.

Test description: Log GPS location to puck.js RAM every 3 minutes for several hours, logging GPS location to a separate GPS tracker (my smartphone's GPS Logger) at the same time. I drove from my home in Corinth, VT to Lebanon, NH and back again, taking alternate routes and back roads where possible.

Test period: June 30, 2019 8:08 p.m. to 10:48 p.m.

Test results:

| Device | Time | Lat | Lon |
|---|---|---|---|
| GPS Logger | 2019-06-28T20:01:01.013Z | 44.01756 | -72.2217 |
| GPS Logger | 2019-06-28T20:02:26.812Z | 44.01756 | -72.2217 |
| GPS Logger | 2019-06-28T20:04:01.737Z | 44.01756 | -72.2217 |
| GPS Logger | 2019-06-28T20:05:28.303Z | 44.01756 | -72.2217 |
| GPS Logger | 2019-06-28T20:07:02.000Z | 44.01757 | -72.2216 |
| GPS Logger | 2019-06-28T20:08:32.407Z | 44.01756 | -72.2217 |
| GPS Logger | 2019-06-28T20:09:35.968Z | 44.01756 | -72.2217 |
| GPS Logger | 2019-06-28T20:11:02.000Z | 44.01756 | -72.2216 |
| Ublox | 2019-06-28T20:11:46 | 44.01749 | -72.2217 |
| GPS Logger | 2019-06-28T20:12:28.534Z | 44.01756 | -72.2217 |
| GPS Logger | 2019-06-28T20:13:37.813Z | 44.01756 | -72.2217 |
| GPS Logger | 2019-06-28T20:14:44.687Z | 44.01756 | -72.2217 |
| Ublox | 2019-06-28T20:14:47 | 44.01753 | -72.2218 |
| GPS Logger | 2019-06-28T20:16:07.572Z | 44.01756 | -72.2217 |

| | | | |
|---|---|---|---|
| GPS Logger | 2019-06-28T20:17:11.000Z | 44.01757 | -72.2216 |
| Ublox | 2019-06-28T20:17:55 | 44.01761 | -72.2214 |
| GPS Logger | 2019-06-28T20:18:21.000Z | 44.01757 | -72.2216 |
| GPS Logger | 2019-06-28T20:19:24.304Z | 44.01756 | -72.2217 |
| GPS Logger | 2019-06-28T20:20:31.000Z | 44.01757 | -72.2216 |
| Ublox | 2019-06-28T20:20:56 | 44.01761 | -72.2214 |
| GPS Logger | 2019-06-28T20:21:43.000Z | 44.01758 | -72.2216 |
| GPS Logger | 2019-06-28T20:22:46.000Z | 44.01726 | -72.2221 |
| Ublox | 2019-06-28T20:23:57 | 44.01973 | -72.2258 |
| GPS Logger | 2019-06-28T20:24:22.000Z | 44.02147 | -72.2218 |
| GPS Logger | 2019-06-28T20:25:30.000Z | 44.01964 | -72.2089 |
| GPS Logger | 2019-06-28T20:26:34.707Z | 44.02364 | -72.2015 |
| Ublox | 2019-06-28T20:26:58 | 44.02697 | -72.1987 |
| GPS Logger | 2019-06-28T20:27:42.000Z | 44.03097 | -72.192 |
| GPS Logger | 2019-06-28T20:28:51.000Z | 44.02761 | -72.1779 |
| GPS Logger | 2019-06-28T20:29:54.000Z | 44.01714 | -72.1716 |
| Ublox | 2019-06-28T20:29:59 | 44.01599 | -72.171 |
| GPS Logger | 2019-06-28T20:31:05.000Z | 44.00604 | -72.1643 |
| GPS Logger | 2019-06-28T20:32:12.405Z | 43.99549 | -72.1568 |
| Ublox | 2019-06-28T20:33:0 | 43.98876 | -72.1504 |
| GPS Logger | 2019-06-28T20:33:59.000Z | 43.98434 | -72.1373 |
| GPS Logger | 2019-06-28T20:35:04.000Z | 43.98195 | -72.1228 |
| GPS Logger | 2019-06-28T20:36:06.485Z | 43.98151 | -72.1216 |
| Ublox | 2019-06-28T20:36:1 | 43.98178 | -72.122 |
| GPS Logger | 2019-06-28T20:37:10.000Z | 43.9826 | -72.1246 |
| GPS Logger | 2019-06-28T20:38:13.000Z | 43.98244 | -72.1304 |
| GPS Logger | 2019-06-28T20:39:17.000Z | 43.96866 | -72.1263 |
| Ublox | 2019-06-28T20:39:3 | 43.97197 | -72.1263 |
| GPS Logger | 2019-06-28T20:40:20.402Z | 43.95236 | -72.1321 |
| GPS Logger | 2019-06-28T20:41:52.000Z | 43.92763 | -72.1348 |
| Ublox | 2019-06-28T20:42:4 | 43.9239 | -72.1335 |
| GPS Logger | 2019-06-28T20:43:01.390Z | 43.91035 | -72.1386 |
| GPS Logger | 2019-06-28T20:44:07.000Z | 43.90087 | -72.1586 |
| GPS Logger | 2019-06-28T20:45:15.399Z | 43.88724 | -72.1749 |
| Ublox | 2019-06-28T20:45:5 | 43.88937 | -72.1738 |
| GPS Logger | 2019-06-28T20:46:53.000Z | 43.86296 | -72.1887 |
| GPS Logger | 2019-06-28T20:47:57.000Z | 43.84695 | -72.1983 |
| GPS Logger | 2019-06-28T20:48:59.000Z | 43.83125 | -72.207 |
| Ublox | 2019-06-28T20:48:6 | 43.84425 | -72.2001 |
| GPS Logger | 2019-06-28T20:50:03.000Z | 43.81542 | -72.2148 |
| GPS Logger | 2019-06-28T20:51:07.000Z | 43.81135 | -72.2131 |
| Ublox | 2019-06-28T20:51:7 | 43.81152 | -72.213 |
| GPS Logger | 2019-06-28T20:52:10.000Z | 43.81191 | -72.2137 |

| | | | |
|---|---|---|---|
| GPS Logger | 2019-06-28T20:53:13.000Z | 43.80607 | -72.2193 |
| GPS Logger | 2019-06-28T20:54:25.000Z | 43.78786 | -72.2155 |
| Ublox | 2019-06-28T20:54:9 | 43.79164 | -72.2144 |
| GPS Logger | 2019-06-28T20:55:28.000Z | 43.77195 | -72.2232 |
| GPS Logger | 2019-06-28T20:56:44.000Z | 43.75529 | -72.2378 |
| Ublox | 2019-06-28T20:57:10 | 43.75058 | -72.2459 |
| GPS Logger | 2019-06-28T20:58:23.000Z | 43.74052 | -72.2689 |
| GPS Logger | 2019-06-28T20:59:55.000Z | 43.72128 | -72.2909 |
| Ublox | 2019-06-28T20:8:45 | 44.01757 | -72.2217 |
| Ublox | 2019-06-28T21:0:11 | 43.71688 | -72.2938 |
| GPS Logger | 2019-06-28T21:01:02.000Z | 43.70841 | -72.3054 |
| GPS Logger | 2019-06-28T21:02:07.000Z | 43.70402 | -72.302 |
| GPS Logger | 2019-06-28T21:03:08.394Z | 43.70236 | -72.2923 |
| GPS Logger | 2019-06-28T21:04:18.000Z | 43.70189 | -72.2894 |
| GPS Logger | 2019-06-28T21:05:53.398Z | 43.69054 | -72.2911 |
| GPS Logger | 2019-06-28T21:06:58.000Z | 43.68152 | -72.2946 |
| GPS Logger | 2019-06-28T21:08:31.000Z | 43.67038 | -72.2999 |
| GPS Logger | 2019-06-28T21:09:36.000Z | 43.66933 | -72.3007 |
| GPS Logger | 2019-06-28T21:10:43.396Z | 43.66293 | -72.3051 |
| GPS Logger | 2019-06-28T21:11:56.000Z | 43.65547 | -72.308 |
| Ublox | 2019-06-28T21:12:15 | 43.65391 | -72.3094 |
| GPS Logger | 2019-06-28T21:13:12.182Z | 43.64989 | -72.3105 |
| GPS Logger | 2019-06-28T21:14:40.000Z | 43.64984 | -72.3107 |
| Ublox | 2019-06-28T21:15:16 | 43.64988 | -72.3106 |
| GPS Logger | 2019-06-28T21:16:23.000Z | 43.64984 | -72.3107 |
| GPS Logger | 2019-06-28T21:17:56.000Z | 43.64984 | -72.3107 |
| Ublox | 2019-06-28T21:18:17 | 43.64983 | -72.3106 |
| GPS Logger | 2019-06-28T21:19:29.000Z | 43.64985 | -72.3107 |
| GPS Logger | 2019-06-28T21:20:52.746Z | 43.64986 | -72.3106 |
| Ublox | 2019-06-28T21:21:19 | 43.64973 | -72.311 |
| GPS Logger | 2019-06-28T21:21:58.000Z | 43.64669 | -72.31 |
| GPS Logger | 2019-06-28T21:23:10.000Z | 43.64003 | -72.3135 |
| Ublox | 2019-06-28T21:24:20 | 43.63483 | -72.3168 |
| GPS Logger | 2019-06-28T21:26:02.000Z | 43.63147 | -72.3201 |
| Ublox | 2019-06-28T21:27:21 | 43.63128 | -72.3098 |
| GPS Logger | 2019-06-28T21:27:24.726Z | 43.63132 | -72.3092 |
| GPS Logger | 2019-06-28T21:29:06.000Z | 43.63985 | -72.277 |
| Ublox | 2019-06-28T21:3:12 | 43.70238 | -72.292 |
| GPS Logger | 2019-06-28T21:30:13.000Z | 43.65055 | -72.259 |
| Ublox | 2019-06-28T21:30:22 | 43.6505 | -72.2556 |
| GPS Logger | 2019-06-28T21:31:16.000Z | 43.64731 | -72.2516 |
| GPS Logger | 2019-06-28T21:32:19.000Z | 43.64245 | -72.2542 |
| GPS Logger | 2019-06-28T21:33:22.000Z | 43.64235 | -72.2523 |

| | | | |
|---|---|---|---|
| Ublox | 2019-06-28T21:33:23 | 43.64216 | -72.2524 |
| GPS Logger | 2019-06-28T21:34:26.000Z | 43.64292 | -72.2556 |
| GPS Logger | 2019-06-28T21:35:30.000Z | 43.64159 | -72.2556 |
| Ublox | 2019-06-28T21:36:24 | 43.63918 | -72.2629 |
| GPS Logger | 2019-06-28T21:36:37.000Z | 43.63888 | -72.2647 |
| GPS Logger | 2019-06-28T21:37:40.401Z | 43.64294 | -72.2733 |
| GPS Logger | 2019-06-28T21:38:45.000Z | 43.64528 | -72.2808 |
| Ublox | 2019-06-28T21:39:25 | 43.64383 | -72.2777 |
| GPS Logger | 2019-06-28T21:40:18.000Z | 43.64142 | -72.2699 |
| GPS Logger | 2019-06-28T21:41:26.000Z | 43.63853 | -72.2685 |
| Ublox | 2019-06-28T21:42:26 | 43.63792 | -72.2805 |
| GPS Logger | 2019-06-28T21:43:11.000Z | 43.63654 | -72.2879 |
| GPS Logger | 2019-06-28T21:44:17.000Z | 43.63658 | -72.2886 |
| Ublox | 2019-06-28T21:45:29 | 43.63616 | -72.2903 |
| GPS Logger | 2019-06-28T21:45:40.000Z | 43.63607 | -72.2918 |
| GPS Logger | 2019-06-28T21:46:48.316Z | 43.63916 | -72.3034 |
| Ublox | 2019-06-28T21:48:30 | 43.64919 | -72.3103 |
| GPS Logger | 2019-06-28T21:48:33.076Z | 43.64916 | -72.3103 |
| GPS Logger | 2019-06-28T21:49:39.000Z | 43.65197 | -72.318 |
| GPS Logger | 2019-06-28T21:50:44.000Z | 43.65823 | -72.3151 |
| Ublox | 2019-06-28T21:51:31 | 43.66424 | -72.3145 |
| GPS Logger | 2019-06-28T21:51:48.000Z | 43.66585 | -72.3175 |
| GPS Logger | 2019-06-28T21:53:13.000Z | 43.68411 | -72.3143 |
| Ublox | 2019-06-28T21:54:32 | 43.70477 | -72.307 |
| GPS Logger | 2019-06-28T21:54:37.391Z | 43.70556 | -72.3063 |
| GPS Logger | 2019-06-28T21:55:40.000Z | 43.71853 | -72.2921 |
| GPS Logger | 2019-06-28T21:57:15.000Z | 43.73836 | -72.271 |
| Ublox | 2019-06-28T21:57:33 | 43.74172 | -72.2652 |
| GPS Logger | 2019-06-28T21:58:41.000Z | 43.7504 | -72.2445 |
| GPS Logger | 2019-06-28T21:59:44.000Z | 43.76204 | -72.2285 |
| Ublox | 2019-06-28T21:6:13 | 43.68761 | -72.2928 |
| Ublox | 2019-06-28T21:9:14 | 43.67008 | -72.2999 |
| Ublox | 2019-06-28T22:0:34 | 43.7744 | -72.2219 |
| GPS Logger | 2019-06-28T22:00:53.000Z | 43.77886 | -72.2209 |
| GPS Logger | 2019-06-28T22:02:01.397Z | 43.79568 | -72.2161 |
| GPS Logger | 2019-06-28T22:03:10.000Z | 43.81303 | -72.2158 |
| GPS Logger | 2019-06-28T22:04:58.000Z | 43.83986 | -72.202 |
| GPS Logger | 2019-06-28T22:06:01.000Z | 43.85528 | -72.1928 |
| GPS Logger | 2019-06-28T22:07:29.257Z | 43.87693 | -72.1812 |
| GPS Logger | 2019-06-28T22:09:17.000Z | 43.90005 | -72.1599 |
| GPS Logger | 2019-06-28T22:10:22.000Z | 43.90478 | -72.1502 |
| GPS Logger | 2019-06-28T22:11:33.000Z | 43.90467 | -72.1503 |
| GPS Logger | 2019-06-28T22:12:37.005Z | 43.91003 | -72.1392 |

| | | | |
|---|---|---|---|
| Ublox | 2019-06-28T22:12:38 | 43.91023 | -72.1382 |
| GPS Logger | 2019-06-28T22:13:49.000Z | 43.92675 | -72.1332 |
| GPS Logger | 2019-06-28T22:14:57.000Z | 43.94418 | -72.1316 |
| Ublox | 2019-06-28T22:15:39 | 43.95579 | -72.1293 |
| GPS Logger | 2019-06-28T22:16:43.392Z | 43.97191 | -72.1256 |
| GPS Logger | 2019-06-28T22:17:50.000Z | 43.98384 | -72.1297 |
| Ublox | 2019-06-28T22:18:40 | 43.98623 | -72.1423 |
| GPS Logger | 2019-06-28T22:19:23.000Z | 43.99026 | -72.1526 |
| GPS Logger | 2019-06-28T22:20:25.405Z | 43.99856 | -72.1596 |
| Ublox | 2019-06-28T22:21:41 | 44.01018 | -72.1684 |
| GPS Logger | 2019-06-28T22:21:57.432Z | 44.01224 | -72.1699 |
| GPS Logger | 2019-06-28T22:23:29.000Z | 44.02829 | -72.1787 |
| GPS Logger | 2019-06-28T22:24:37.000Z | 44.03011 | -72.1929 |
| Ublox | 2019-06-28T22:24:42 | 44.02928 | -72.1944 |
| GPS Logger | 2019-06-28T22:25:40.927Z | 44.02254 | -72.2025 |
| GPS Logger | 2019-06-28T22:27:19.000Z | 44.02162 | -72.2165 |
| Ublox | 2019-06-28T22:27:43 | 44.02128 | -72.2223 |
| GPS Logger | 2019-06-28T22:28:20.589Z | 44.01813 | -72.228 |
| GPS Logger | 2019-06-28T22:29:25.710Z | 44.0175 | -72.2217 |
| Ublox | 2019-06-28T22:3:35 | 43.81948 | -72.2115 |
| GPS Logger | 2019-06-28T22:30:30.000Z | 44.01754 | -72.2216 |
| Ublox | 2019-06-28T22:30:44 | 44.01762 | -72.2214 |
| GPS Logger | 2019-06-28T22:31:33.000Z | 44.01755 | -72.2215 |
| GPS Logger | 2019-06-28T22:32:59.755Z | 44.01756 | -72.2217 |
| Ublox | 2019-06-28T22:33:45 | 44.01762 | -72.2213 |
| GPS Logger | 2019-06-28T22:34:04.878Z | 44.01756 | -72.2217 |
| GPS Logger | 2019-06-28T22:35:08.839Z | 44.01756 | -72.2217 |
| GPS Logger | 2019-06-28T22:36:18.000Z | 44.01774 | -72.2215 |
| Ublox | 2019-06-28T22:36:46 | 44.01761 | -72.2218 |
| GPS Logger | 2019-06-28T22:37:21.000Z | 44.0175 | -72.2217 |
| GPS Logger | 2019-06-28T22:39:03.000Z | 44.01759 | -72.2217 |
| Ublox | 2019-06-28T22:39:47 | 44.01753 | -72.2218 |
| GPS Logger | 2019-06-28T22:40:30.876Z | 44.01756 | -72.2217 |
| GPS Logger | 2019-06-28T22:41:36.924Z | 44.01756 | -72.2217 |
| GPS Logger | 2019-06-28T22:42:42.000Z | 44.0176 | -72.2217 |
| Ublox | 2019-06-28T22:42:48 | 44.01758 | -72.2219 |
| GPS Logger | 2019-06-28T22:43:44.800Z | 44.01756 | -72.2217 |
| GPS Logger | 2019-06-28T22:45:30.858Z | 44.01756 | -72.2217 |
| Ublox | 2019-06-28T22:45:49 | 44.01745 | -72.2219 |
| GPS Logger | 2019-06-28T22:47:15.821Z | 44.01756 | -72.2217 |
| GPS Logger | 2019-06-28T22:48:37.000Z | 44.01741 | -72.2217 |
| Ublox | 2019-06-28T22:48:50 | 44.01771 | -72.2217 |
| GPS Logger | 2019-06-28T22:50:17.827Z | 44.01756 | -72.2217 |

| | | | |
|---|---|---|---|
| GPS Logger | 2019-06-28T22:51:23.000Z | 44.01752 | -72.2216 |
| GPS Logger | 2019-06-28T22:52:26.805Z | 44.01756 | -72.2217 |
| GPS Logger | 2019-06-28T22:53:31.000Z | 44.01752 | -72.2216 |
| GPS Logger | 2019-06-28T22:54:34.137Z | 44.01756 | -72.2217 |
| GPS Logger | 2019-06-28T22:56:14.184Z | 44.01756 | -72.2217 |
| GPS Logger | 2019-06-28T22:57:50.580Z | 44.01756 | -72.2217 |
| GPS Logger | 2019-06-28T22:59:09.622Z | 44.01756 | -72.2217 |
| Ublox | 2019-06-28T22:06:36 | 43.8644 | -72.1877 |
| Ublox | 2019-06-28T22:09:37 | 43.90347 | -72.1539 |
| GPS Logger | 2019-06-28T23:00:19.000Z | 44.01761 | -72.2217 |

| | |
|---|---|
| GPS battery level (actual*) - start | 4.18 V |
| GPS battery level (actual*) - end | 3.87 V |
| puck.js battery level (actual*) - start | 3.0 V |
| puck.js battery level (actual*) - end | 2.86 V |
| puck.js battery level (percentage**) - start | 97 |
| puck.js battery level (percentage**) - end | 79 |

* measured using multimeter

** measured using puck.js E.getBattery() command

Appendix 5

Temperature Accuracy Test

Primary test goal: To determine the accuracy of the temperature sensor on the puck.js.

Secondary test goal: To determine the capacity of the CR 2032 coin battery used to power the puck.js.

Test description: Log temperature to puck.js RAM every 5 minutes for several hours, logging temperature to a separate temperature logger (the Elitech RC-5) at the same time.

Test period: June 30, 2019 9:34 a.m. to 7:19 p.m.

Test results:

| puck.js time | puck.js temperature | Elitech time | Elitech temperature | Difference |
|---|---|---|---|---|
| 9:34:52 | 16.75 | 9:34:58 | 22.1 | -5.35 |
| 9:39:52 | 16.75 | 9:39:58 | 21.9 | -5.15 |
| 9:44:52 | 8 | 9:44:58 | 16.5 | -8.5 |
| 9:49:52 | 1.5 | 9:49:58 | 11 | -9.5 |
| 9:54:52 | -0.25 | 9:54:58 | 8.3 | -8.55 |
| 9:59:52 | -1 | 9:59:58 | 6.7 | -7.7 |
| 10:04:52 | -3 | 10:04:58 | 5.3 | -8.3 |
| 10:09:52 | -3.5 | 10:09:58 | 4.1 | -7.6 |
| 10:14:52 | -4.25 | 10:14:58 | 3.2 | -7.45 |
| 10:19:52 | -4.75 | 10:19:58 | 2.5 | -7.25 |
| 10:24:52 | -5.25 | 10:24:58 | 2.1 | -7.35 |
| 10:29:52 | -4.25 | 10:29:58 | 2.5 | -6.75 |
| 10:34:52 | -3.25 | 10:34:58 | 3.1 | -6.35 |
| 10:39:52 | -2 | 10:39:58 | 3.7 | -5.7 |
| 10:44:52 | -2 | 10:44:58 | 4.1 | -6.1 |

| | | | | |
|---|---|---|---|---|
| 10:49:52 | -3 | 10:49:58 | 3.9 | -6.9 |
| 10:54:52 | -4 | 10:54:58 | 3.2 | -7.2 |
| 10:59:52 | -4.75 | 10:59:58 | 2.6 | -7.35 |
| 11:04:52 | -4.25 | 11:04:58 | 2.7 | -6.95 |
| 11:09:52 | -3.25 | 11:09:58 | 3.2 | -6.45 |
| 11:14:52 | -2.5 | 11:14:58 | 3.7 | -6.2 |
| 11:19:52 | -2 | 11:19:58 | 4.1 | -6.1 |
| 11:24:52 | -2.5 | 11:24:58 | 4 | -6.5 |
| 11:29:52 | -4.25 | 11:29:58 | 3.4 | -7.65 |
| 11:34:52 | -5.25 | 11:34:58 | 2.6 | -7.85 |
| 11:39:52 | -0.25 | 11:39:58 | 4.7 | -4.95 |
| 11:44:52 | -0.75 | 11:44:58 | 5.2 | -5.95 |
| 11:49:52 | -3.5 | 11:49:58 | 4.1 | -7.6 |
| 11:54:52 | -4.25 | 11:54:58 | 3.3 | -7.55 |
| 11:59:52 | -5.25 | 11:59:58 | 2.6 | -7.85 |
| 12:04:52 | -5 | 12:04:58 | 2.2 | -7.2 |
| 12:09:52 | -2.25 | 12:09:58 | 3.5 | -5.75 |
| 12:14:52 | -1.25 | 12:14:58 | 4.4 | -5.65 |
| 12:19:52 | -2.5 | 12:19:58 | 4.1 | -6.6 |
| 12:24:52 | -1 | 12:24:58 | 4.7 | -5.7 |
| 12:29:52 | 0.25 | 12:29:58 | 5.3 | -5.05 |
| 12:34:52 | 0.75 | 12:34:58 | 6.1 | -5.35 |
| 12:39:52 | 0.5 | 12:39:58 | 6.4 | -5.9 |
| 12:44:52 | 1 | 12:44:58 | 6.6 | -5.6 |
| 12:49:52 | 0.25 | 12:49:58 | 6.7 | -6.45 |
| 12:54:52 | -1 | 12:54:58 | 6.2 | -7.2 |
| 12:59:52 | -2 | 12:59:58 | 5.4 | -7.4 |
| 13:04:52 | -3 | 13:04:58 | 4.6 | -7.6 |
| 13:09:52 | -4.25 | 13:09:58 | 3.9 | -8.15 |
| 13:14:52 | -4.75 | 13:14:58 | 3.3 | -8.05 |
| 13:19:52 | -4.25 | 13:19:58 | 3.2 | -7.45 |
| 13:24:52 | -3.75 | 13:24:58 | 3.2 | -6.95 |
| 13:29:52 | -3.75 | 13:29:58 | 3.4 | -7.15 |
| 13:34:52 | -4.5 | 13:34:58 | 3.1 | -7.6 |
| 13:39:52 | -4.75 | 13:39:58 | 2.7 | -7.45 |
| 13:44:52 | -5.25 | 13:44:58 | 2.3 | -7.55 |
| 13:49:52 | -5.5 | 13:49:58 | 2 | -7.5 |
| 13:54:52 | -5.25 | 13:54:58 | 2.2 | -7.45 |
| 13:59:52 | -4 | 13:59:58 | 2.8 | -6.8 |

| | | | | |
|---|---|---|---|---|
| 14:04:52 | 8 | 14:04:58 | 9.6 | -1.6 |
| 14:09:52 | 13.5 | 14:09:58 | 15.6 | -2.1 |
| 14:14:52 | 14.5 | 14:14:58 | 22.9 | -8.4 |
| 14:19:52 | 14.75 | 14:19:58 | 21.1 | -6.35 |
| 14:24:52 | 14.5 | 14:24:58 | 20.1 | -5.6 |
| 14:29:52 | 14.25 | 14:29:58 | 19.8 | -5.55 |
| 14:34:52 | 13.75 | 14:34:58 | 19.8 | -6.05 |
| 14:39:52 | 13.75 | 14:39:58 | 19.8 | -6.05 |
| 14:44:52 | 14.25 | 14:44:58 | 19.8 | -5.55 |
| 14:49:52 | 13.75 | 14:49:58 | 19.8 | -6.05 |
| 14:54:52 | 14.25 | 14:54:58 | 19.9 | -5.65 |
| 14:59:52 | 14.5 | 14:59:58 | 19.9 | -5.4 |
| 15:04:52 | 14.25 | 15:04:58 | 20.1 | -5.85 |
| 15:09:52 | 14.75 | 15:09:58 | 20.2 | -5.45 |
| 15:14:52 | 14 | 15:14:58 | 20.3 | -6.3 |
| 15:19:52 | 14 | 15:19:58 | 20.3 | -6.3 |
| 15:24:52 | 14 | 15:24:58 | 20.3 | -6.3 |
| 15:29:52 | 14.75 | 15:29:58 | 20.3 | -5.55 |
| 15:34:52 | 14.25 | 15:34:58 | 20.3 | -6.05 |
| 15:39:52 | 14.25 | 15:39:58 | 20.3 | -6.05 |
| 15:44:52 | 14.75 | 15:44:58 | 20.3 | -5.55 |
| 15:49:52 | 14.5 | 15:49:58 | 20.4 | -5.9 |
| 15:54:52 | 14.75 | 15:54:58 | 20.4 | -5.65 |
| 15:59:52 | 14.25 | 15:59:58 | 20.4 | -6.15 |
| 16:04:52 | 14.75 | 16:04:58 | 20.4 | -5.65 |
| 16:09:52 | 14.75 | 16:09:58 | 20.4 | -5.65 |
| 16:14:52 | 14.5 | 16:14:58 | 20.4 | -5.9 |
| 16:19:52 | 15 | 16:19:58 | 20.5 | -5.5 |
| 16:24:52 | 14.75 | 16:24:58 | 20.5 | -5.75 |
| 16:29:52 | 14.75 | 16:29:58 | 20.5 | -5.75 |
| 16:34:52 | 14.75 | 16:34:58 | 20.5 | -5.75 |
| 16:39:52 | 15 | 16:39:58 | 20.5 | -5.5 |
| 16:44:52 | 14.5 | 16:44:58 | 20.5 | -6 |
| 16:49:52 | 15.25 | 16:49:58 | 20.5 | -5.25 |
| 16:54:52 | 14.75 | 16:54:58 | 20.6 | -5.85 |
| 16:59:52 | 14.75 | 16:59:58 | 20.6 | -5.85 |
| 17:04:52 | 14.75 | 17:04:58 | 20.6 | -5.85 |
| 17:09:52 | 15 | 17:09:58 | 20.6 | -5.6 |
| 17:14:52 | 14.75 | 17:14:58 | 20.6 | -5.85 |

| | | | | |
|---|---|---|---|---|
| 17:19:52 | 14.75 | 17:19:58 | 20.6 | -5.85 |
| 17:24:52 | 14.75 | 17:24:58 | 20.7 | -5.95 |
| 17:29:52 | 15.25 | 17:29:58 | 20.7 | -5.45 |
| 17:34:52 | 15 | 17:34:58 | 20.7 | -5.7 |
| 17:39:52 | 14.75 | 17:39:58 | 20.7 | -5.95 |
| 17:44:52 | 15.5 | 17:44:58 | 20.7 | -5.2 |
| 17:49:52 | 14.75 | 17:49:58 | 20.7 | -5.95 |
| 17:54:52 | 15 | 17:54:58 | 20.7 | -5.7 |
| 17:59:52 | 15 | 17:59:58 | 20.7 | -5.7 |
| 18:04:52 | 15.25 | 18:04:58 | 20.8 | -5.55 |
| 18:09:52 | 15.25 | 18:09:58 | 20.7 | -5.45 |
| 18:14:52 | 15.5 | 18:14:58 | 20.7 | -5.2 |
| 18:19:52 | 15.25 | 18:19:58 | 20.7 | -5.45 |
| 18:24:52 | 15.5 | 18:24:58 | 20.7 | -5.2 |
| 18:29:52 | 15 | 18:29:58 | 20.7 | -5.7 |
| 18:34:52 | 15.25 | 18:34:58 | 20.7 | -5.45 |
| 18:39:52 | 14.75 | 18:39:58 | 20.7 | -5.95 |
| 18:44:52 | 15 | 18:44:58 | 20.8 | -5.8 |
| 18:49:52 | 15 | 18:49:58 | 20.8 | -5.8 |
| 18:54:52 | 15 | 18:54:58 | 20.8 | -5.8 |
| 18:59:52 | 14.75 | 18:59:58 | 20.8 | -6.05 |
| 19:04:52 | 15.5 | 19:04:58 | 20.8 | -5.3 |
| 19:09:52 | 15.5 | 19:09:58 | 20.8 | -5.3 |
| 19:14:52 | 16 | 19:14:58 | 21.3 | -5.3 |
| 19:19:52 | 17.5 | 19:19:58 | 22.4 | -4.9 |

| | |
|---|---|
| puck.js battery level (actual*) - start | 2.96 V |
| puck.js battery level (actual*) - end | 2.81 V |
| puck.js battery level (percentage**) - start | 90 |
| puck.js battery level (percentage**) - end | ? |

*Note:* Although its battery still had power, the puck.js inexplicably stopped running in the middle of the test (I had originally planned to run the test until July 1 the following morning). The unit started up again when I removed then re-inserted the battery.

Chapter XV.

References


ABCO Transportation. The History of Refrigerated Trucking [Web log post]. (2015, March 23). Retrieved from https://www.shipabco.com/history-refrigerated-trucking/

Armour, Britt. Mobile App Vs. Mobile Website: Which Is The Better Option? [Web log post]. (2018, April 20). Retrieved from https://www.business2community.com/mobile-apps/mobile-app-vs-mobile-website-which-is-the-better-option-02048068

Bryant, Dennis. Refrigerated cargo ships [Web log post]. (2014, August 2016). Retrieved from https://www.maritimeprofessional.com/blogs/post/refrigerated-cargo-ships-13576

Buxbaum, Peter. (2018, March 26). The perils of perishable airfreight shipments. American Journal of Transportation. Retrieved from https://www.ajot.com/premium/ajot-the-perils-of-perishable-airfreight-shipments

Ciligot, Chris. Mobile App Vs. Mobile Website: A UX Comparison – Which Is The Better Option? [Web log post]. (2019, May 2). Retrieved from https://clearbridgemobile.com/mobile-app-vs-mobile-website-which-is-the-better-option/

Cold chain on a plane: Tracking perishable air freight with cloud and IoT [Web log post]. (2018, June 6). Retrieved from https://controlant.com/blog/2018/cold-chain-on-a-plane-tracking-perishable-air-freight-with-cloud-and-iot/, 2018)

de Wit, Gerard. (2016, March). Perishables Markets - Modal Shift – Trends. Retrieved from http://coolchain.org/Websites/cca/images/WorldACD_presentation_Perishables_-_Ken_de_Witt_Hamer.pdf

Fisher, William. “The Importance of Food Traceability.” FSM EDigest. March 17, 2015. Retrieved from https://www.foodsafetymagazine.com/enewsletter/the-importance-of-food-traceability/

Fortney, Luke. Blockchain Explained [Web log post]. (2019, June 25). Retrieved from https://www.investopedia.com/terms/b/blockchain.asp

Hougham, Joey. Transportation of Perishables-The Importance of Paying Attention to the Cold Chain [Web log post]. (2015, June 9). Retrieved from https://www.trangistics.com/2015/06/transportation-of-perishables-the-importance-of-paying-attention-to-the-cold-chain/

Krasner-Khait, Barbara. "The Impact of Refrigeration." *History Magazine*. History Magazine, Feb./Mar. 200. Web. 23 March 2015.

S. Mercier, S. Villeneuve, M. Mondor, and I. Uysal, "Time–Temperature Management Along the Food Cold Chain: A Review of Recent Developments," Comprehensive Reviews in Food Science and Food Safety, vol. 16, no. 4, pp. 647–667, 2017.

Orlando, Dan. Freshness remains elusive for seafood delivery [Web log post]. (2017, August 22). Retrieved from https://www.supermarketnews.com/print/87342

Ranger, Steve. What is the IoT? Everything you need to know about the Internet of Things right now (Updated: The Internet of Things explained. What the IoT is, and where it's going next) [Web log post]. (2018, August 21). Retrieved from https://www.zdnet.com/article/what-is-the-internet-of-things-everything-you-need-to-know-about-the-iot-right-now/

Redman, Russell. Publix promotes seafood transparency: Tag program identifies responsibly, sustainably sourced products [Web log post]. (2019, July 8). Retrieved from https://www.supermarketnews.com/sustainability/publix-promotes-seafood-transparency

Roberts, Mike. Serverless Architectures [Web log post]. (2018, May 22). Retrieved from https://martinfowler.com/articles/serverless.html

Rosic, Ameer. What is Blockchain Technology? A Step-by-Step Guide For Beginners [Web log post]. (2019, March 1). Retrieved from https://blockgeeks.com/guides/what-is-blockchain-technology/

Schreiber, Laurie. Exporting seafood: With perishable products, time is money [Web log post]. (2017, May 1). Retrieved from https://www.mainebiz.biz/article/exporting-seafood-with-perishable-products-time-is-money

Shea, Sharon. Use cases and benefits of smart sensors for IoT [Web log post]. (2015, July 30). Retrieved from https://internetofthingsagenda.techtarget.com/opinion/How-smart-sensors-are-transforming-the-Internet-of-Things

Shirani, A. (2018). Blockchain for global maritime logistics. *Issues in Information Systems*. 19(3) 175-183.

Shoffler, Sarah. The Challenges of Producing and Consuming Local San Diego Seafood [Web log post]. (2018, April 23). Retrieved from

https://ediblesandiego.ediblecommunities.com/food-thought/challenges-producing-and-consuming-local-san-diego-seafood

Stevens, Emily. What Is The Difference Between A Mobile App And A Web App? [Web log post]. (2018, April 3). Retrieved from https://careerfoundry.com/en/blog/web-development/what-is-the-difference-between-a-mobile-app-and-a-web-app/

Te-Food. Food Traceability Trends to watch in 2019 [Web log post]. (2019, January 17). Retrieved from https://medium.com/te-food/food-traceability-trends-to-watch-in-2019-179a00b3b625

Wells, Jeff. Delivering fresh meat and seafood remains a challenge [Web log post]. (2017, August 25). Retrieved from https://www.grocerydive.com/news/grocery--delivering-fresh-meat-and-seafood-remains-a-challenge/534763/

Williams, Gordon. Espruino: The Challenges of Running an Open Source Hardware and Software Company [Web log post]. (2017, June 1). Retrieved from https://makezine.com/2017/06/01/espruino-open-for-business/

M Woolley. (2017, May 1). Extending the Reach of the Web to Bluetooth Devices [Web log post]. Retrieved from http://blog.bluetooth.com/extending-the-reach-of-the-web-to-bluetooth-devices?_ga=2.268998769.1297226269.1525803096-1043439978.1525803096