Kerby: The Curbside Parking Buddy

Kanvi Shah (Author), Mrinmayee Mandal (Author), Neville Chima (Author)

Department of Electrical and Computer Engineering, Carnegie Mellon University

Abstract—**Kerby is a system capable of reflecting street parking availability, giving drivers a time-sensitive insight on parking locations. It alleviates the frustration of having to circle around blocks multiple times only to never find a parking spot. Our solution aims to expand upon current garage and lot parking management systems and include street parking as another application area. It also provides a cheaper alternative to current state of the art systems since street parking is oftentimes close to free.**

Index Terms— **ESP8266, IoT, MQTT, NodeMCU, Parking, Sensor, Smart Cities, WiFi**

I. Introduction

PARKING is one of the most painful parts of driving. In addition to the issues drivers face with inadequate parking spaces and parking maneuvers, they may simply not find parking spots, especially on the street. Finding an available spot is often a cycle of inconvenient turns and circling. Our proposal is to make a smart city solution that reduces the amount of time wasted looking for street side parking by providing users with the location of the closest open parking spot.

Commercial technologies include systems such as ParkPGH, SpotHero, and BestParking; however, these systems usually always work within garages and enclosed parking lots, due to the convenience of implementation and hence, economic viability. The target users for these apps are people who occasionally visit some city for an event. City locals use street parking more often because (i) it is way cheaper and (ii) it might be closer to their destination. Street side parking is here to stay for the foreseeable future and there exists a need to create a centralized parking system that allows users to access both garage and public street parking.

Kerby, our curbside parking management system, aims to extend the application of parking technology while remaining cost effective.

II. Use-Case Requirements

There are two kinds of users for our proposed system: the driver and the operator. We chose to narrow our scope to just drivers for feasibility purposes due to time constraints in the semester. There are several use-case requirements for the driver. Plainly said, drivers need to save time and enjoy hassle-free parking.

The first requirement is that the user's car must be less than 16 feet in length in order for our spot modules to detect it. This is derived from the fact that the average car is around 14.7 feet long. Thus, our product aims to serve not all cars but a large proportion of them. The most important user requirement that Kerby fulfills is an easy way to find the closest open street parking spot to a driver's inputted destination. We have chosen to ensure that our system is able to find street parking within less than half a mile to the destination, so that we can sustain the benefit of street parking over garage parking. This also means that accuracy in location will be another important user requirement. When shown a parking location, the driver should be able to get directions to this location and expect the open spot to be within 30 feet of where they are. This quantitative metric is used because this is approximately the length of two cars. Additionally, to provide accurate results about availability, the sensors will wake up every 5 minutes to sense if their state should change (spot taken or not). We chose five minutes since we found that most people take at least 3 minutes to park and leave the car. Lastly, since we want our system to remain cost efficient, we require that the spot modules should remain less than fifty dollars, so that we can enable future scalability.

III. Architecture and/or Principle of Operation

Kerby's system design is aimed to be simple and digestible by any user or operator. The principle of operation is easy communication between the state of the physical world (i.e. the parking spot's availability) and consumers (i.e. the drivers). The overall system has three key components: sensing IoT hardware installed on the curbside, an information warehouse stored on the cloud, and the user interface web application.

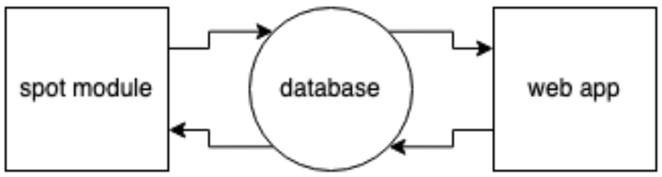


Fig. 1. Overall simplified system block diagram

The essential functions are accurate sensing, timely communication, accessible information, and a usable user interface. Each of these functions are supported by subsystems of our design. Accurate sensing will be completed by an ultrasonic sensor and transmitted through an IoT device. In Fig. 2, a closer look at the "spot module" hardware is described. The details of each component are shared in section VI, but the overall concept of component interactions is displayed in these diagrams.

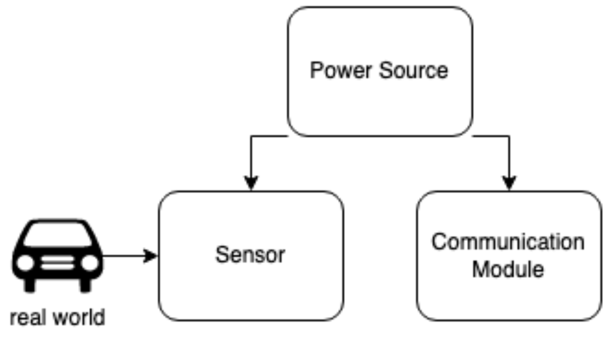


Fig. 2. Zoom-in of spot module system

Similarly, Fig. 3 shows a closer look at the database/cloud section. Our hardware spot module will take care of collecting the information of parking spot availability, transferring it over to this next section that processes and stores it in our cloud server (e.g. AWS). These server-side processes will likely be where we will be able to improve our timing-based performance the most, so it is key our system implementation is as clean as possible.

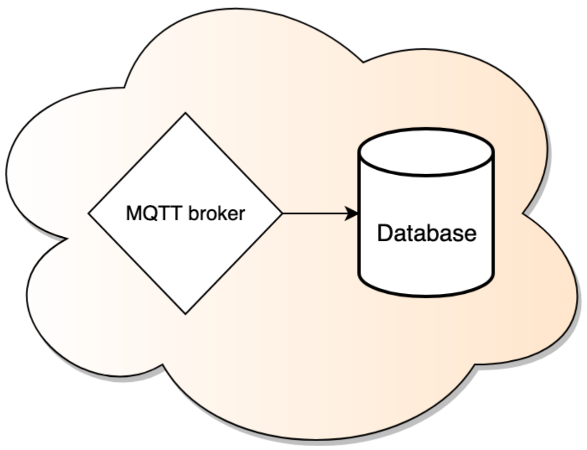


Fig. 3. Simplification of cloud-side communication and information storage

The block diagram of the entire system architecture is included in the Appendix A.

IV. Design Requirements

Kerby's principles of operation paired with the user requirements has provided us with a roadmap for the design requirements your curbside parking buddy will have to fulfill.

The spot module subsystem must be able to detect cars parked in front of the sensor, calling for a distance sensor with a range of up to at least 1 foot. Next, the spots are to be detected for cars under 16 feet, which requires the modules to be configured in a way to accurately depict this. Currently, this requirement will be fulfilled by sensors positioned at every 8 feet on a curbside.

Additionally, we are looking for our system's power to last a reasonable amount of time before needing to change the batteries. The design will call for sensor wake ups every 5 minutes and a battery that can work with this timing to last around 6 months.

Requirements for communication come from the need to transfer data from a spot module to a central database and into the user's hands. Modules require a small, low-power consumption, low-cost communication device, with a reliable protocol like Wi-Fi. The Wi-Fi range also must be large enough to be able to hop on to the closest network and communicate with our central database. Since our MVP plan is to implement this on CMU campus, we have defined the range requirement only to be at least 100 meters. For future development, this would be extended to at least a few kilometers (may require the addition of an antenna).

To maintain the scalability and accessibility of Kerby, we defined the user requirements of modules to stay under $50 each. This has worked well with our principle of operation, to stay simple and functional, and driven our design requirements to be reasonable for cheaper IoT to fulfill.

Since we would like to provide instant data to our users, our database needs to be able to answer real-time requests from our users but also track all the sensor data that has been collected. Cloud services, like AWS, will be able to provide this specification for Kerby.

The user requirement of an easy-to-use graphical interface will be fulfilled by a web application that is compatible with the most common browsers and accessible to users. This means we will design the web app according to ADA standards. Our plan is to make the application very simple and clean, without too many options and extremely clear about where to input information and how to reserve a parking space.

Providing an accurate available parking location within 30 feet will require Kerby's modules to have their geolocations (longitude, latitude) be encoded with each of their unique IDs. This will allow us to use an API (such as Google Maps) to calculate driving distance between destination and parking spots, fulfilling another user requirement of being easy-to-use.

V. Design Trade Studies

We continue with a trade-off analysis on our chosen design specifications regarding how they satisfy our design requirements. Our system requirements were primarily concerned with 1) detection and location accuracy, 2) transfer latency, and 3) project scalability.

A. Sensors vs. Cameras

Brainstorming in the initial phases of our project had led us down the path of possibly using cameras with computer vision to fulfill the requirement of "seeing" parked cars. Solving the same problem, these cameras would be installed at city intersections and look down the road and "see" if there were open spots or not.

However, after some driving around on the curvy Pittsburgh roads, a realization of how expensive and complex computer vision would become to get any accurate readings turned us away from this route.

Instead, to fulfill the requirement of scalability and accuracy, we decided that a curbside module would be more feasible. This way location accuracy and detection accuracy could be fulfilled, at a lower cost than high functioning cameras paired with complex computer vision computations.

B. IR sensors vs. Ultrasonic sensors

The design specification of needing a sensor that can sense if a car as far away as 1 foot from the curb could be fulfilled by different distance sensors. We researched both IR and ultrasonic sensors. The most common IR distance sensor used with Arduino projects has the range of 10cm-80cm but is priced at $10 a piece. Another common IR distance sensor was priced at around $3.33 per piece but the range was only 10cm-30cm.

The most common ultrasonic sensor used in Arduino and IoT products, the HC-SR04, has a range of 2cm - 400cm and was priced at $2.40 per piece. This additional range and lower price point made it seem like the perfect choice. Additionally, we were able to find more documentation for projects with this sensor.

C. Microcontroller and Communication Board

Initially, the hardware portion of our project fits the description of a common household Arduino project. However, with the addition of needing to communicate between modules and with a central database, we introduced the possibility of needing WiFi.

Most Arduino boards are not WiFi capable and would require an external module for enabling this functionality. The Arduino UNO WiFi R2 does meet this specification; however, its price point at almost $45 essentially put it out of the running.

With our design specs of WiFi range, power supply needs, and price, we settled upon the ESP8266 NodeMCU. The low sleeping current consumption of 0.5 uA and active current consumption as low as 0.15 uA were huge factors, as well as the 3.3 V operating voltage. Additionally, there are more than enough digital I/O pins (13) to attach the ultrasonic sensor as well as any other debugging/future work extensions. There also exists plentiful documentation for creating IoT devices with the ESP8266.

D. Power Sources

Though our final selection of power source has not occurred, this is a design specification that we have been extremely interested in. Our project is untenable without a long-lasting power source.

Our original idea (and still hope for the future) is that we will be able to run our system on solar power. However, the tradeoff in time spent making an inexpensive solar-powered system vs. time spent on making a more accurately functioning system made us lean towards focusing on the other sections of our project first. For power, we have settled on rechargeable LiFePo4 batteries, as they have been researched to work well with the ESP8266 NodeMCU boards for long-lasting power. Once we start testing, we may change this power source depending on our own testing tradeoffs.

VI. System Implementation

This section describes our system's implementation in detail. End-to-End, Kerby's features an on-site hardware subsystem (spot modules) connected to a remote software subsystem via an IoT communication pathway. We now describe each subsystem along with their internal components:

A. *Spot Modules*

Kerby's spot modules are small packaged interconnected hardware devices to be placed along the curbside of street parking spots. It features (i) an ultrasonic sensor to provide vehicle presence readings (ii) a microprocessor chip equipped with Wi-Fi functionality to process sensor readings and (iii) a long-lasting battery to reliably power the System-on-Chip board.

Our choice of ultrasonic sensor is the HRCS04. This paragraph describes the spot module topology and the sensor's capability. In order to model free continuous space in street parking, Kerby takes inspiration from dynamic allocation of computer memory. Popular memory allocators define basic unit blocks of memory and combine/split these into larger/smaller units to dynamically allocate/free a requested amount of memory resources. In a similar fashion, we apportion vehicle detection over a unit parking space to our basic operating unit called a *spot module.* These spot modules are horizontally placed 8 feet apart as seen in Figure. Given that we limit our project scope to 16 feet average length cars, we will use three consecutive spot modules to dynamically detect whether there is enough space to park one average-sized vehicle. The HRCS04 offers vehicle detection readings by measuring the time taken for a sound to echo on reflection traveling through air which can be easily converted to distance value. The HRCS04 detects distances in the range of 0 - 4m (its maximum detection distance as noted by its spec sheet) with an effective angle of at most 15°. On close observation, the sensor's ranging distance and effective angle justify our choice of 8 feet spacing because each sensor's detection space will not overlap with others. When no object is detected within its range, it emits 4m. We can safely assume that a 4m reading is not a false negative because Kerby's sensors are placed right on the curbside, centimeters from an expected parking spot.

Our hardware subsystem will need an algorithm in order to convert variable distance readings to discrete occupancy or availability values. In line with our design requirements, our main concerns are the reliable representation of the parking scene and running low power-consuming processes. For the former, we propose an idea to sample multiple readings within a short timeframe. The value for these parameters will be based on our best estimates for how long it takes a car to park and how quickly our microprocessor I/O port can receive readings. Preliminary research shows that we can expect a car to come to halt in under 1 minute and our microprocessor can be programmed to output readings at least every 2 seconds. Bounding periodic reading by these values, we will initially take 5 sensor readings in 15 second intervals to get a discretized reading just about every minute. If the readings consistently provide similar values with a range - the difference between the most distant and least distant readings - of less than 0.2m, a spot is considered occupied if the least distant reading in the sequence is less than 3.98m (the maximum value minus our 0.2m error tolerance).

Our sensor readings are consumed by a wi-fi-enabled ESP8266 microprocessor. HRCS04 features *Trig* (input) and *Echo* (output) pins which can be connected to any of the microprocessor's GPIO pins. A time-triggered short HIGH pulse on the *Trig* pin followed by a LOW pulse would be used to establish a duty cycle for the HRCS04's sleep feature described in the preceding paragraph. The *Echo* pin's values will be read over our programming interface. To run lightweight programs on the ESP8266, we will use the open-source MicroPython firmware. Apart from the flexibility to write programs at a higher level of abstraction than most iOT platforms, MicroPython comes with a rich set of in-built libraries and facilitates the import of user-defined libraries. For example, our GPIO interface will be controlled by the prepackaged Machine.Pin MicroPython class. To install MicroPython onto the processor's flash memory, Kerby will use esptool to serially communicate with the chip's ROM bootloader. Likewise, it will use ampy to serially communicate with MicroPython on the board and run our sensor processing script.

A. *IoT communication medium*

This subsystem is concerned with the transfer of our sensor readings from the spot source to a cloud sink. We will achieve this using the popular lightweight IoT publish-subscribe network protocol MQTT (Message Queuing Telemetry Transport) provisioned through AWS's IoT core service. At its core, IoT core will provide us with an MQTT broker that connects to multiple client connections via TCP/IP over some exposed endpoint. These client connections will take two forms: (i) a Publisher which uploads sensor payload to an intermediate cloud storage area ii) a Subscriber which connects to the MQTT broker and takes some action on receiving the sensor payload. AWS IoT core provides extensive documentation and service-tier levels which we will exclude from this discussion for brevity. Nevertheless, at a high-level we will use the *AWSIoTPythonSDK* API to maintain our client connections and provide security credentials. We will also create our accessible IoT endpoints and monitor device connections through AWS Console.

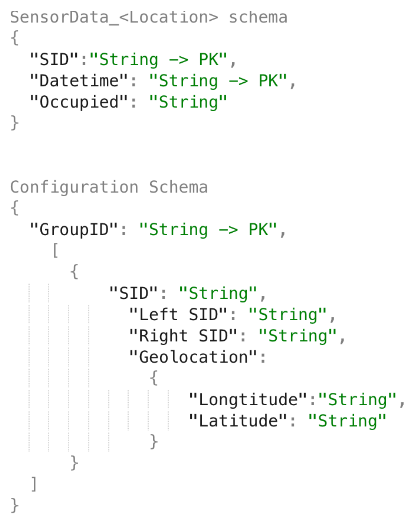
Reliable data transmission and project scalability were use-case requirements raised earlier that play an important role in our IoT medium design decisions. MQTT is a reliably designed protocol that utilizes TCP/IP's acknowledgement framework and message queues under the hood to deal with traffic congestion. We entrust AWS to guarantee the reliability of this design as a world-renowned cloud provider while we can tune other parameters within our control. MQTT offers varying levels of Quality of Service (QoS) that indicate whether a message was successfully delivered or may not be. For Kerby, we will use the Exactly Once semantic that guarantees each published message will arrive exactly once. On the other hand, the structure of our communication pathway can affect Kerby's ability to scale. For example, it may be possible that connections may become overloaded as we add exponentially more spot modules. To this end, we assign connections to handle sensor payloads belonging to the same geolocated group. This implies that our sensor endpoints map to a physical group of clustered spot modules e.g /sensor-data/maggie-mo. and one client processes messages on one endpoint. Hence, for our MVP (described later), Kerby uses a 1:1 pub-sub model that can linearly grow or be grouped into sub-hierarchies e.g /sensor-data/cmu-campus etc. in the future. Moreover, to enable geolocation as a group ID for processing, we need to identify spot modules in the same physical location. We will enable this by establishing a directory mapping of each spot module's unique Wi-Fi SSID to its physical location and embedding the SSID in sensor payloads. Once captured in code, these SSIDs can be used for making routing rules (as discussed here) or database insert rules (discussed later).

B. *Server-side processes*

This section discusses the processes in Kerby's software workflow from once the sensor data is available on a subscribe client connection till it is used to serve a client's parking request. We first discuss the role of AWS LightSail in running our server processes then the specifics of our NoSQL storage database, AWS DynamoDB, and lastly our self-built webapp.

AWS LightSail is a cloud service providing virtual servers that will enable us to deploy our subscribe scripts in containerized applications. We will need to provision a dedicated machine to persist a job that subscribes to an MQTT subscriber and inserts the sensor payload into DynamoDB on receipt. Lightsail allows us to create Docker containers tha mount volumes containing our necessary installation packages on some image and run the script independently. Beyond receiving the payload and inserting it into database tables, this subscribing process will add an ISO datetime to the sensor record. This will be necessary to query for recent sensor readings during webapp requests.

DynamoDB is our schema-less data warehouse for Kerby. The schema for our various database tables is described in Figure 4. The important details of our design choices are as follows: (i) We store each geolocated group of sensor readings in the same table. This is only practical for quick lookup purposes given the latency and scalability issues we discussed in section 5.A (ii) We utilize a composite partitioning/primary key consisting of the spot module's SSID and datetime for tables persisting sensor readings. This unique identifier is necessary because it would be harmful to overwrite or mix up any previous database records. (iii) We store a graph mapping of the hardware arrangement of sensor modules per geolocation in a separate table. This will be loaded in-memory to our webapp for algorithm computation on startup.



```
SensorData_<Location> schema
{
  "SID":"String -> PK",
  "Datetime": "String -> PK",
  "Occupied": "String"
}


Configuration Schema
{
  "GroupID": "String -> PK",
    [
      {
          "SID": "String",
            "Left SID": "String",
            "Right SID": "String",
            "Geolocation":
              {
                  "Longtitude":"String",
                  "Latitude": "String"
              }
      }
    ]
}
```

Fig. 4. Sensor data schema

VII. Test, Verification and Validation

Testing was conducted to verify the performance of the ultrasonic sensors in comparison to the spec sheet metrics. Additionally, we plan to field test our system on campus at the two ends of Maggie Mo with street parking. We will use different requests for different destinations from different locations on campus.

To test each user requirement, we specify appropriate measurements and the corresponding goal. In order to find the closest street parking to the destination, we will use the google maps API to find the distance between a given spot and distance and make sure it is less than half a mile. The google maps API has features such as geocoding and distance matrix that will be useful. To ensure accurate parking location, we will be measuring the distance between the provided location and actual parking spot in the real world by hand. We will specifically look for whether this value is less than 30 feet. Additional tests are further explained in detail below.

A. Tests for Accurate Representation of Real World

In order to test whether our system accurately reflects the real-world state, we will be stress testing Kerby and creating a confusion matrix. We aim to have less than 20% of data points that fall under the false positive or false negative categories.

B. Tests for Usability of Web Application

In order to make sure that the web app is straightforward to use, we will be conducting user testing and recording ratings from 1 to 5 (1=bad, 5=great). We aim to have greater than 3.5/5 stars on average.

VIII. Project Management

A. Schedule

Refer to the attached schedule at the end of this report in Appendix B. We are currently on schedule to have a functioning MVP and demo by the due date.

Schedule changes include longer time required for researching components and adding slack for time dedicated to other presentations and reports.

B. Team Member Responsibilities

We distributed responsibilities among team members according to each person's strengths and interests. Kanvi will handle the hardware setup for the sensor modules and the frontend graphics for the web app design. Mrinmayee will be in charge of researching and testing sensor detection algorithms to parse data from the spot module. Neville will implement the communication between module and central database and the central hub maintenance software. As for presentations and paperwork, we all work on different sections to draft up content and edit together. Lastly, we will all work together for integration and field testing, and we also agree to help each other out as needed throughout the semester.

C. Bill of Materials and Budget

Refer to Table 1, Bill of Materials in Appendix C. As mentioned earlier, we prioritize being cost effective in our choice of materials.

D. Risk Mitigation Plans

One of the possible risks for Kerby includes failure of communication from module to database. In the case one module starts malfunctioning, either because of low power, WiFi interference, or another reason, we will make sure to mark its place in our database with "unknown status." This is also what will be displayed to our users on the graphical interface, as to make sure no one is led to a "possibly open" spot.

Additionally, in the case our system gives false information to the user about parking availability, we hope to incorporate user feedback (simple as buttons with "yes, I found and used the spot!" or "no, the spot was not empty") that will indicate to us if a certain sensor needs to be taken offline. With our model of two types of users, this would be information given to the operators so they could service the module in need.

In order to mitigate risk of hardware damage in the field, we will be encasing our modules with weatherproof materials. Initially (and likely, the scope of our project), these materials may consist of an airtight plastic box with holes cut for the sensor. Once our design reaches a more sophisticated state, weatherproofing will also progress to a cleaner look and stronger materials meant for the roadside.

IX. Related Work

Currently there are no commercially available widely used products that work similar to what Kerby proposes to execute. However, there is work being done with similar goals. INRIX is tapping into the IoT of the automotive industry and introducing ultrasonic technology to be installed on the cars itself. If all cars use this technology, then cars parked alongside a curb would be able to provide information about the availability of spots in front and behind them.

Spot is another company that is developing smart city solutions, with the goal of making all parking information available at the tip of your fingertips. Work they have completed has made for an almost complete database of parking information for a portion of Sydney, Australia - displayed through a graphical web user interface that can tell you as you mouse over areas: if there is free or paid parking, if there is space available, if there is a temporary construction site blocking the spots.

X. Summary

Overall, Kerby is a system that aims to reflect street parking availability, acting as a street parking management system to give drivers insight into where to go ahead of time. The system will have three main component groups: spot modules, central data hub, and the web application. The stakeholders are users of street parking in big cities. We have learned about many design tradeoffs so far but realize that we may continue to run into challenges.

REFERENCES

[1] *Arduino vs ESP8266 vs ESP32 Comparison*, Accessed on Feb 15, 2022, [Online]. Available: https://diyi0t.com/technical-datasheet-microcontroller-comparison/

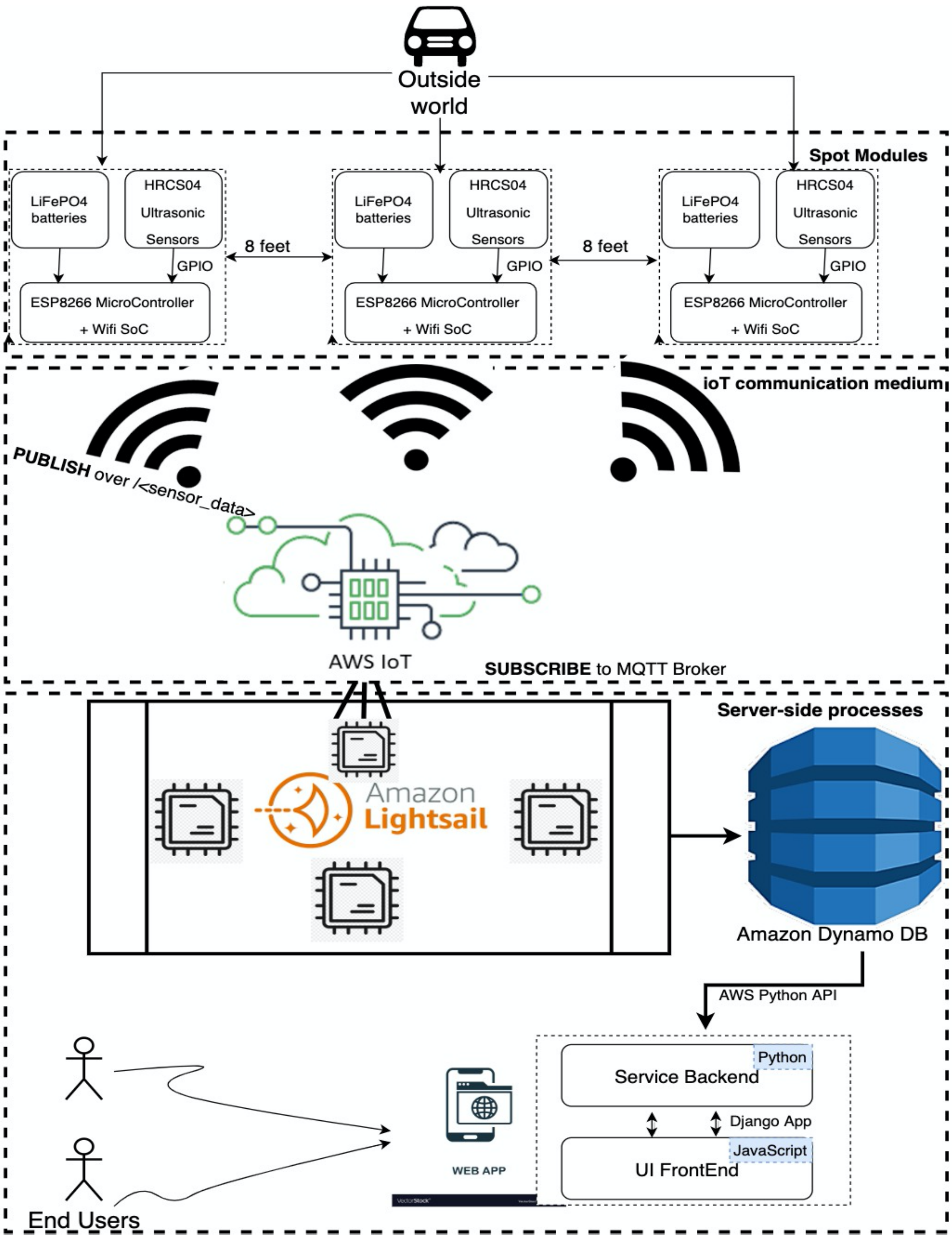
[2] *Best Battery for ESP8266 microcontroller*, Accessed on Feb 20, 2022, [Online]. Available: https://diyi0t.com/best-battery-for-esp8266/

[3] *ESP8266 NodeMCU with HC-SR04 Ultrasonic Sensor with Arduino IDE*, Accessed on Feb 28, 2022, [Online]. Available: https://randomnerdtutorials.com/esp8266-nodemcu-hc-sr04-ultrasonic-arduino/

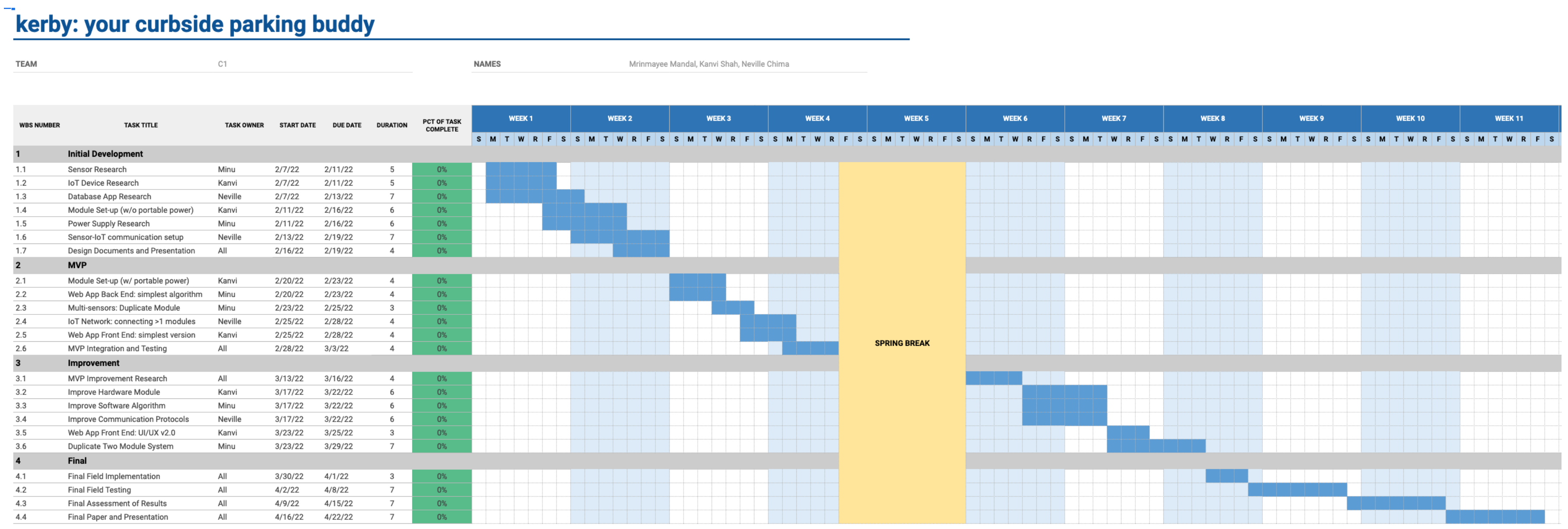
[4] *Ultrasonic Sensor Parking Availability Technology*, Accessed on March 1, 2022, [Online]. Available: https://inrix.com/blog/ultrasonic-sensor-parking-availability-technology/

[5] *Spot: Smart city parking and mobility solutions.* Accessed on Feb 2, 2022, [Online]. Available: https://www.spotparking.us/spot-cities

Appendix A: Block Diagram



Appendix B: Schedule



kerby: your curbside parking buddy

TEAM C1 NAMES Mrinmayee Mandal, Kanvi Shah, Neville Chima

WBS NUMBER	TASK TITLE	TASK OWNER	START DATE	DUE DATE	DURATION	PCT OF TASK COMPLETE
1	**Initial Development**					
1.1	Sensor Research	Minu	2/7/22	2/11/22	5	0%
1.2	IoT Device Research	Kanvi	2/7/22	2/11/22	5	0%
1.3	Database App Research	Neville	2/7/22	2/13/22	7	0%
1.4	Module Set-up (w/o portable power)	Kanvi	2/11/22	2/16/22	6	0%
1.5	Power Supply Research	Minu	2/11/22	2/16/22	6	0%
1.6	Sensor-IoT communication setup	Neville	2/13/22	2/19/22	7	0%
1.7	Design Documents and Presentation	All	2/16/22	2/19/22	4	0%
2	**MVP**					
2.1	Module Set-up (w/ portable power)	Kanvi	2/20/22	2/23/22	4	0%
2.2	Web App Back End: simplest algorithm	Minu	2/20/22	2/23/22	4	0%
2.3	Multi-sensors: Duplicate Module	Minu	2/23/22	2/25/22	3	0%
2.4	IoT Network: connecting >1 modules	Neville	2/25/22	2/28/22	4	0%
2.5	Web App Front End: simplest version	Kanvi	2/25/22	2/28/22	4	0%
2.6	MVP Integration and Testing	All	2/28/22	3/3/22	4	0%
3	**Improvement**					
3.1	MVP Improvement Research	All	3/13/22	3/16/22	4	0%
3.2	Improve Hardware Module	Kanvi	3/17/22	3/22/22	6	0%
3.3	Improve Software Algorithm	Minu	3/17/22	3/22/22	6	0%
3.4	Improve Communication Protocols	Neville	3/17/22	3/22/22	6	0%
3.5	Web App Front End: UI/UX v2.0	Kanvi	3/23/22	3/25/22	3	0%
3.6	Duplicate Two Module System	Minu	3/23/22	3/29/22	7	0%
4	**Final**					
4.1	Final Field Implementation	All	3/30/22	4/1/22	3	0%
4.2	Final Field Testing	All	4/2/22	4/8/22	7	0%
4.3	Final Assessment of Results	All	4/9/22	4/15/22	7	0%
4.4	Final Paper and Presentation	All	4/16/22	4/22/22	7	0%

APPENDIX C: BOM

TABLE I. BILL OF MATERIALS

Description	Model #	Manufacturer	Quantity	Cost per unit	Total
ESP8266 NodeMCU Dev Board	CP2102 ESP-12E	HiLetgo	3	$5.46	$16.39
Ultrasonic Sensors	HC-SR04	Tangyy	5	$2.40	$12
Batteries	14430 Rechargeable LiFePo4	JESSPOW	8	$2.13	$17