**Connected Farms**

**Final Design Document**

**Group Members:**

Nathan Libner, ndlibner@ncsu.edu

Artem Minin, aaminin@ncsu.edu

Stephanie Sierra, ssierra@ncsu.edu

Manish Goud, mgoud@ncsu.edu

**Sponsoring Company:**

NCSU College of Agriculture and Life Sciences

IBM

**Mentors:**

Dr. Paula Ramos Giraldo, pramosg@ncsu.edu

Dr. Anand Singh, anand.singh@ibm.com

Dr. Natalie Hummel, nahummel@ncsu.edu

James Jushchuk, jushchuk@ibm.com

**Table of Contents**

[**Introduction:**](#_35kj4knz2vdi) **2**

[Project Overview](#_ai7hdnuljl47) 3

[**Project Requirements:**](#_qh1h04rqxggg) **5**

[General Operational Requirements and Features](#_ubowcm247jc2) 5

[System/Device/Unit Constraint requirements](#_7zlwndzexcvb) 6

[Other Needs](#_87hx0of4jxlh) 7

[**Overall System Architecture:**](#_eoxxc2sg9ocw) **7**

[Raspberry Pi](#_szk3qmtyohk3) 8

[Subsystem Requirements](#_xdgreyeuvtw) 9

[Design choices and justifications](#_67v8svv6gyyt) 9

[Testing and Results](#_pxokr9cgmy3s) 12

[Cloud Platform](#_xibha2ys2fkx) 14

[Subsystem Requirements](#_uh9ztegvwjlw) 14

[Design choices and justifications](#_ar5ozducl7s8) 15

[Testing and Results](#_t6831nxh7je9) 20

[Web Platform](#_d6cscyfd5rn3) 22

[Subsystem Requirements](#_1l5w4irdya5o) 22

[Overview](#_4szvha3xkegw) 22

[Design Choices and Justifications](#_30bfzukmub6q) 25

[Testing and Results](#_592446nwr22h) 26

[**Budget:**](#_qz7g25sy539y) **27**

[**Conclusion:**](#_3rx1z8z3yup4) **28**

# **Introduction:**

North Carolina’s agriculture industry currently contributes $87B to the state’s economy. Research on breeding robust crops that can easily handle water stress plays an important role in the agricultural growth and productivity of farmers, however, it is a difficult process. The current method for researching crops in this field is quite tedious. Researchers must plant multiple plots of each crop genotype being studied. Each plot will receive different amounts of water, and researchers must go into the field frequently to assess each genotype based on how much water stress it appears to have. This process is fairly subjective, as it can be difficult to discern exactly how water stressed a crop is just from a quick and repetitive visual analysis.

Dr. Paula Ramos Giraldo, from North Carolina State University, built and programmed Raspberry Pi’s that act as cameras in the fields(Figure 0), periodically taking images of the plots. A Raspberry Pi is a small computer popular for its portability and cheap price point. Having even this small automation greatly reduces the amount of time spent in the field assessing crops. This system was still quite limited, as it still required researchers to travel to the research station(3 hour round-trip) and retrieve the images from SD cards in the cameras, these images then still had to be manually assessed. The soybean image capturing was dual-purposed, these human labeled images could be used to train a deep learning model to rate how stressed a soybean crop is. The purpose of this project is to make the crop research process much easier for researchers by leveraging cloud technology and edge computing to allow sensor/camera data to be automatically collected, analyzed and displayed in the form of custom plots, and allow control of devices in the field.



*Figure 0: Gen. 1 Camera, Fall 2019*

### Project Overview

We have created a website which features numerous plots showcasing data captured from sensors installed on the Raspberry Pi, including: light level (luxometer), canopy temperature, air temperature, and water stress level. These plots can be manipulated to show different date ranges, and provide researchers with an instant visualization of how well the crop is responding to certain environmental conditions. The website also features a device commands page that allows the user to send commands to any specific Raspberry Pi connected to the platform. Some of these commands include: changing the active hours of the camera, running a certain script, and changing camera properties.

While a simple and unified front-end view is presented to the end-user, there are numerous cloud services provisioned through IBM which run in the background allowing data to flow between the various connected devices and the website. We utilized an IoT platform to register and manage the devices we will deploy in the field and databases such as DB2 and Cloudant are used to store the data that the sensors capture throughout the day. These same databases are then queried by our analytics software, Cognos, to create the plots shown on the website. Node-RED is used as a flow based development tool to transfer data sent between the IoT platform and the website and vice versa.

The Raspberry Pi also required device side code to reliably connect and send data to the IoT platform, execute user-sent device commands, and analyze crop images using a deep learning algorithm. We utilized edge computing to reduce network bandwidth and cost. Instead of sending a soybean image from the field, we instead send the water stress rating generated from a deep learning model which analyzes and rates the captured soybean image. The device is responsible for turning itself on according to the user-set daily schedule, during this time period the camera is responsive to commands and will send collected data every 15 minutes.

Our system greatly reduces the amount of researcher time spent in the field, gives researchers access to plot specific data, and increases the accuracy of evaluating how resilient certain soybeans are to water stress. Now, these researchers will be able to make data driven decisions and draw more reliable conclusions. The devices in the field can be easily accessed and be given commands to perform. In the future this system can also be used by farmers in order to improve crop yield and irrigation efficiency.

# **Project Requirements:**

The key product requirements for this project was to design a web platform by utilizing cloud services and cameras to enable farmers and researchers the ability to manage their crops with various genotypes by allowing access from any location as long as they are connected to the internet.

### General Operational Requirements and Features

Table 2.1: User Interaction/operation

|  |  |
| --- | --- |
| Requirement | Description |
| A.1 | The web platform shall be very graphic and allow the visualization of data in the form of different plots for end users. |
| A.2 | The website shall have a very simple website interface in order to avoid confusing the end-users. The best way to show the results from the field is by means of a graph |
| A.3 | The web platform shall be intuitive and lead the end-user to properly use each of the features provided. |
| A.4 | The website shall have data visualization for each crop plot showing: current, daily and historical data for water stress levels. The end-user has the choice to select which one. |
| A.5 | The web platform shall give the user the option to register under either a farmer or plant researcher. The dashboard for each user will be tailored to show the information/data that each user is interested in viewing. |
| A.6 | The web platform shall allow users to view device status, turn on/off cameras, and take pictures of crops through a device management page. |

Table 2.2: Operating Environment

|  |  |
| --- | --- |
| Requirement | Description |
| B.1 | The website shall be compatible with Google Chrome and Microsoft Edge. |
| B.2 | The website shall be optimized to run efficiently on networks with a download speed of 10 Mbps as there is limited internet connection in the field. |
| B.3 | The machine learning algorithm may run on a Raspberry Pi 4, which has limited memory/processing power, to reduce data transferred to the cloud. |

Table 2.3 System size/weight/look and feel features:

|  |  |
| --- | --- |
| Requirements | Description |
| C.1 | The website shall load with a login page, to restrict users and show different functionalities depending on user type. |
| C.2 | The website shall have a page for device management where cameras and sensors can be controlled in a variety of ways. |
| C.3 | The website shall have pages for each plot’s data visualization |

### System/Device/Unit Constraint requirements

Table 2.4 Compatibility or interfacing constraints

|  |  |
| --- | --- |
| Requirements | Description |
| D.1 | The IoT platform shall be able to ingest data from sensors that monitor water levels and canopy cover, and cameras that capture images of the plant used for IoT systems |

Table 2.5 Robustness Constraints

|  |  |
| --- | --- |
| Requirements | Description |
| E.1 | The IoT platform shall be able to support at least 30 cameras and sensors at once. |
| E.2 | The web platform shall be able to support 10-15 users viewing it at once. |
| E.3 | The IoT platform shall be able to process and gain insights from 1000 crop images per day |

Table 2.6: Cost Constraints

|  |  |
| --- | --- |
| Requirements | Description |
| F.1 | Software cost shall be as low as possible to facilitate adoption by farmers. |
| F.2 | The product may not have recurring costs, possibly through a data partnership with cloud service providers. |
| F.3 | The entire product may cost $100 or less per camera |

Table 2.7: Safety/ Security Constraints

|  |  |
| --- | --- |
| Requirements | Description |
| G.1 | The IoT and Web platform shall be secured so that no unwanted people can access the data |

### Other Needs

Table 2.8: Setup/Installation needs

|  |  |
| --- | --- |
| Requirements | Description |
| H.1 | The web platform shall facilitate a simple device setup for the IoT platform in order to make it implementable across farms at low costs and resources. |
| H.2 | The web platform may require no/limited setup needs. |

Table 2.9 Maintenance needs

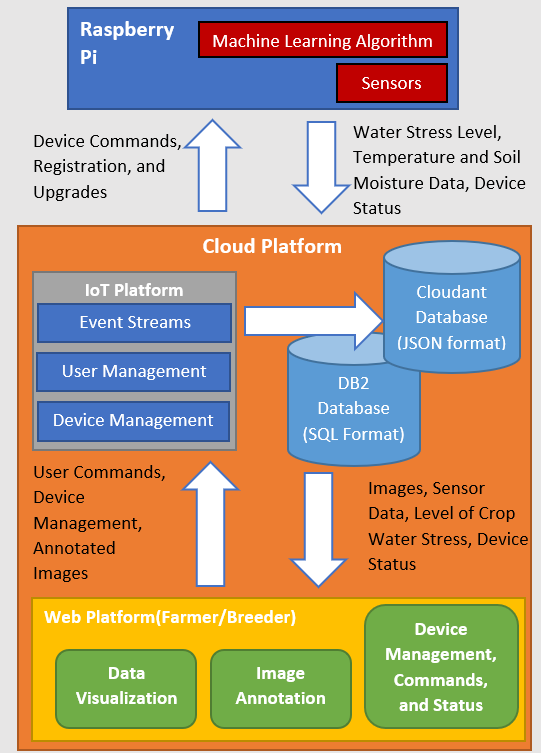
|  |  |
| --- | --- |
| Requirements | Description |
| I.1 | The IoT platform will be provided with external technical support from IBM in case of failure in the future. |

# **Overall System Architecture:**

3 Main System Components:

* + Raspberry Pi’s(Cameras)
  + Cloud Platform
  + Web Platform

Utilized IBM Cloud Platform to host our databases, website, and IoT service



*Figure 1: System Block Diagram*

### Raspberry Pi

The Raspberry Pis, which are sometimes referred to as “cameras”, are the devices in our Internet of Things platform. This device is responsible for automatically starting up every single day to a set schedule, then performing it’s device functions. These functions include collecting/sending sensor data, taking and analyzing crop images with a deep learning model, and executing commands sent from the website. Our group was responsible for creating the software that would allow this functionality. Meanwhile, another senior design team and our sponsor worked on manufacturing and setting up the Raspberry Pi’s housing, sensors, camera, and power supply.

#### Subsystem Requirements

* The deep learning algorithm may run on a Raspberry Pi 4, which has limited memory/processing power, to reduce data transferred to the cloud.
* The Raspberry Pi must turn on automatically to a set schedule every day
* The Raspberry Pi must automatically send sensor data and the deep learning predictions to the IoT platform
* The Raspberry Pi must be able to receive and execute user given commands during scheduled hours.

#### Design choices and justifications

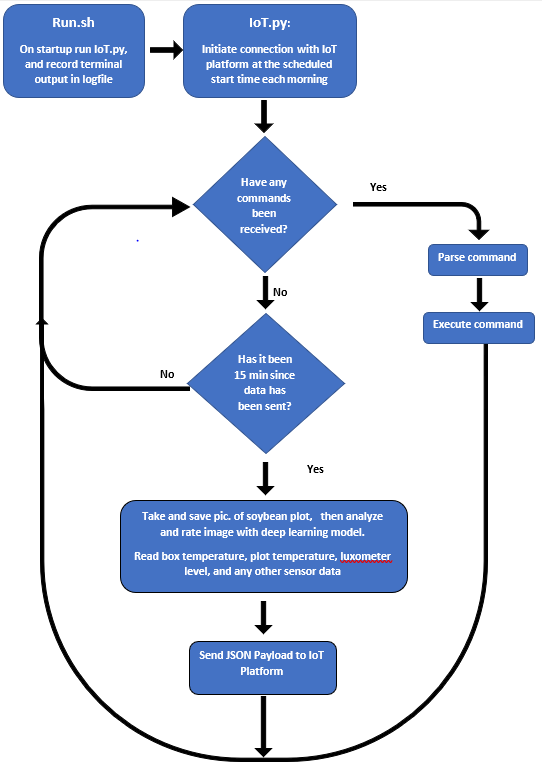
The Raspberry Pi model and firmware were chosen for us by our sponsor. Our main design choices were related to how to create reliable embedded software to run on the Raspberry Pis, how to train the deep learning algorithm, and how to run the compute-heavy model on the Pis. The deep learning model that is deployed on the Raspberry Pis was trained on over a thousand soybean images labeled with a water stress level by experts. Crop researchers labeled the camera captured soybean images from 0[very low water stress] to 4[very high water stress]. These images and their labels were then fed into a four layered Convolutional Neural Network, a type of neural network used extensively for image classification. Upon testing our model upon a set of images it had never “seen” before, we were able to achieve 74% classification accuracy.



*Figure 2: Water Stress Level 0 Figure 3: Water Stress Level 4*

The deep learning model was trained in Google Collab, which allowed us to utilize cloud compute resources to train the model faster. Training the deep learning model is much more compute intensive than just feeding an image into the neural network. However, even just evaluating the model on the Raspberry Pi proved to be impossible due to a combination of the large model size, high memory/cache usage, and processor utilization. Thus a design choice was made to compress the model, by converting it from a TensorFlow model to a TensorFlow Lite model, a machine learning framework specifically designed for platforms with limited compute resources. Although our model could now run on the Raspberry Pi, it could potentially be less accurate, so we had to compare ratings generated by the two models. The results were reassuring, showing that the model was never more than 1 water stress level off the original’s models prediction.

Another design choice we had to make was how to structure the software running on the Raspberry Pi, to efficiently use the limited compute, power and connectivity resources available to us. We decided to utilize the following code structure: when the camera turns on in the morning, one script runs our main program IoT.py continuously throughout the day and records any terminal outputs as seen in Figure 3. Exception handling is used to ensure that the software does not unexpectedly stop due to a software error. At startup, the IoT python program connects to the IoT platform, and, using interrupts, is able to execute commands sent from the website anytime. Every 15 minutes the camera leaves it’s idle state, captures an image and passes it along to the deep learning model, which evaluates and classifies the image. This rating, sensor data, and certain system information is then sent to the website and logged to a log file on the camera.

****

*Figure 4: Raspberry Pi Code Block Diagram*

Our team decided to use the MQTT protocol to communicate between IoT Platform and the Raspberry Pis. JSON formatted data objects will be sent across the two systems. The data JSON object contains the recorded sensor data and water stress predictions. When a user sends a command from the website a JSON object “command” is sent to the IoT interrupt handler, which is unique for each command and contains the command type and other values that may be passed along.

Camera to IoT Platform:

data = {

"DEVICE\_ID": 0001,

"DEVICE\_STATUS": "On",

"LATITUDE": -75.34,

"LONGITUDE": 67.89,

"WATER\_STRESS\_LEVEL":3,

"CANOPY\_TEMPERATURE":29.45,

"AIR\_TEMPERATURE": 28.35,

"WITTYPI\_TEMPERATURE": 35,

"CPU\_TEMPERATURE": 36,

"LUXOMETER":10345,

"DATE\_1":4/26/20,

"TIME\_1":07:34:04,

}

IoT Platform to Camera:

Command = {

"Device ID": 0001,

"Device Command": "Run Script",

"Script Type": "restartSensors.sh",

}

*Figure 5: JSON Objects*

#### Testing and Results

Due to unforeseen circumstances, no field testing occurred with the camera’s at Sandhills Research Center. However, our senior design sponsor built a field test setup in her backyard with 2 fully functioning cameras powered by solar panels. We have the cameras functioning exactly as they would be in the soybean field, but the deep learning algorithm is instead analyzing soybean images taken last season.

The machine learning algorithm is accurately functioning on the Raspberry Pi, correctly rating the test images fed to it around 70% of the time. When it is incorrectly rated, it is very rarely more than 1 rating off. Our software is automatically turning the cameras on and off according to a set schedule daily, and sensor data is being consistently sent every 15 minutes to the IoT platform. Commands also reliably execute when sent from the website. We have the following command functionality programmed on the cameras: take image, send data, change image resolution/size/format, change schedule, run script, and change send interval.

|  |  |  |
| --- | --- | --- |
| **Product Requirement** | **Testing Method** | **Testing Results** |
| The deep learning algorithm may run on a Raspberry Pi 4, which has limited memory/processing power, to reduce data transferred to the cloud. | Upload TF Lite model on the Raspberry Pi, and test it on 120 test images back to back. Compare results to predictions given by TF model. | Camera ran bug-free overnight continuously rating images, although there were some discrepancies(4%) they were limited(96% same), and understandable due to the image/data quality. Also proved software and memory stability of Raspberry Pi |
| The Raspberry Pi must automatically send sensor data and the deep learning predictions to the IoT platform | Field testing, where the IoT program is run by a script when the camera turns on. We observe whether the data is being sent every 15 minutes. | Absent of human input, the Raspberry Pi automatically connects and sends data to the IoT platform without crashing for 3 days |
| The Raspberry Pi must turn on automatically to a set schedule every day | Field testing, observe whether data starts to arrive at the “on” time and stops arriving at the “stop” time | Raspberry Pi, controlled by the WittyPi, follows the same schedule daily for 3 days. |
| The Raspberry Pi must be able to receive and execute user given commands during scheduled hours. | Field testing, while observing the script running, test commands from the website, and see if they get executed correctly | This took the most testing to iron out all of the bugs in the commands, but now almost all of the commands are bug free. Running a script still gives the occasional error |

Table 1: Test Results: Raspberry Pi

Author: Artem Minin, Lead Engineer: Artem Minin

### Cloud Platform

The IBM Cloud is a platform that offers many tools and services to create, manage, run and deploy various types of applications or projects. Based on research and consultation with IBM professionals, the team decided the cloud offered many of the capabilities and functionalities necessary for the raspberry pi’s and web platform.

The cloud platform consists of three main services and applications that interconnect to ensure that data collected from the raspberry pi’s are transmitted to the web platform. The three services are:

1. IBM Watson IoT
2. NodeRed
3. Db2 Database

#### Subsystem Requirements

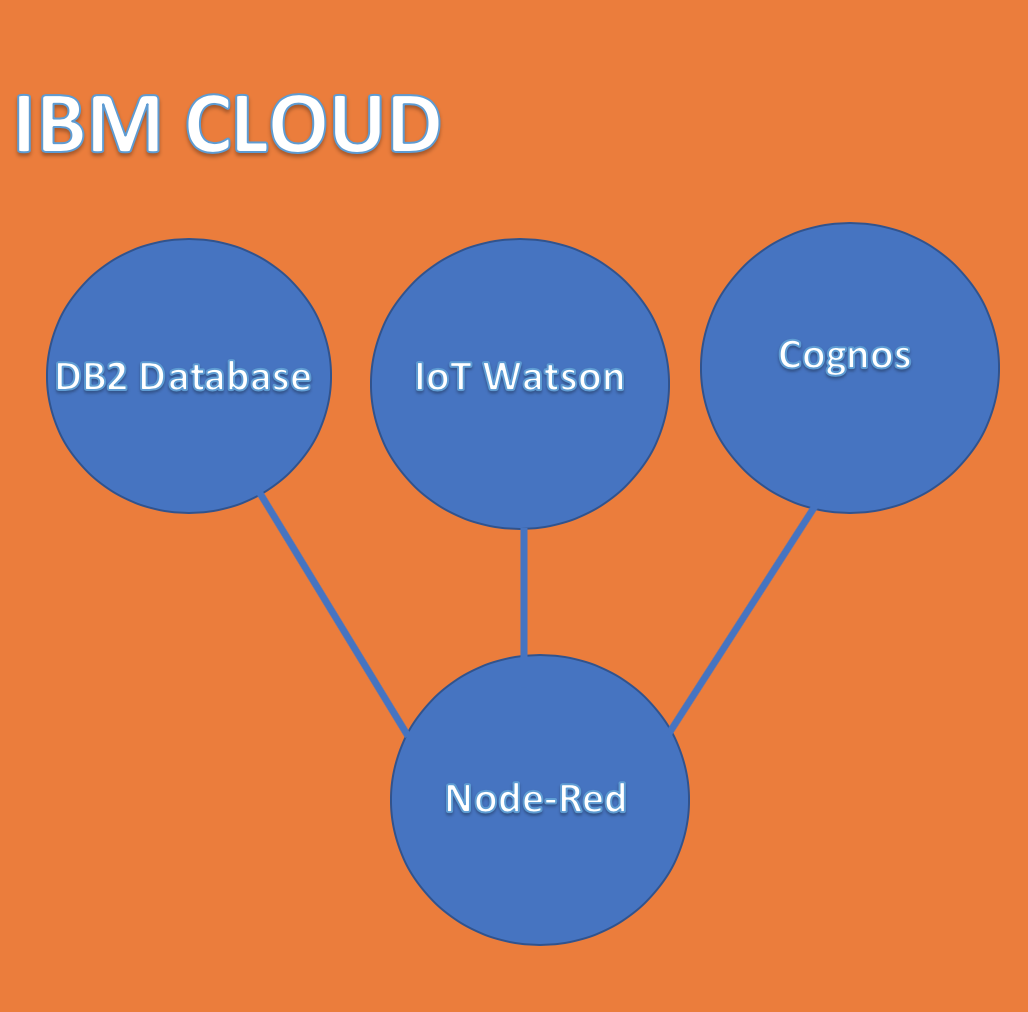
The applications and services created on the platform are responsible for ingesting data from the sensors and displaying them on the website. The following are the key requirements of the cloud:

* The cloud must be able to support at least 30 cameras.
* The cloud must transmit data from the cameras to the database reliably.
* The cloud must support adding and removing devices from the network of cameras.
* The cloud must be able to send commands requested by the user to cameras, and it must also return the requested data from the camera to the user.

#### Design choices and justifications

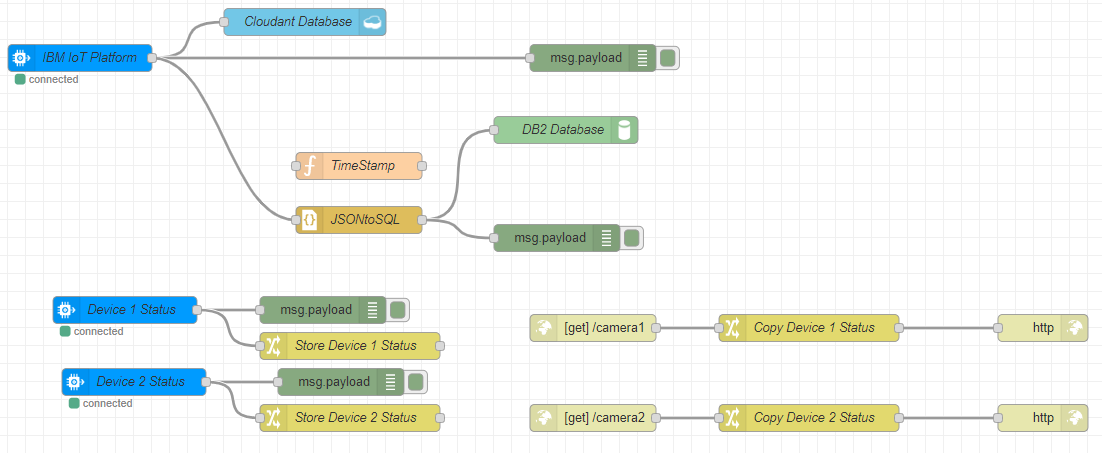
The IBM Cloud is one of the many cloud providers available to the public. As a team we selected IBM to support our application due to various reasons. These reasons include cost, features, and resources. In comparison to other cloud providers, IBM had the lowest cost due to the partnership between North Carolina State University and IBM. The IBM Cloud not only provides device managing features, but also has the features necessary to host a website, without any additional recurring monthly cost. In addition to the cost and features, we selected IBM because we had access to IBM professionals that were experts on the cloud services and could assist in guiding us.

The applications and services that we use in the cloud are DB2 Database, IoT Watson, Cognos, and Node-Red. As you can see below, each application and service must be connected to Node-Red.

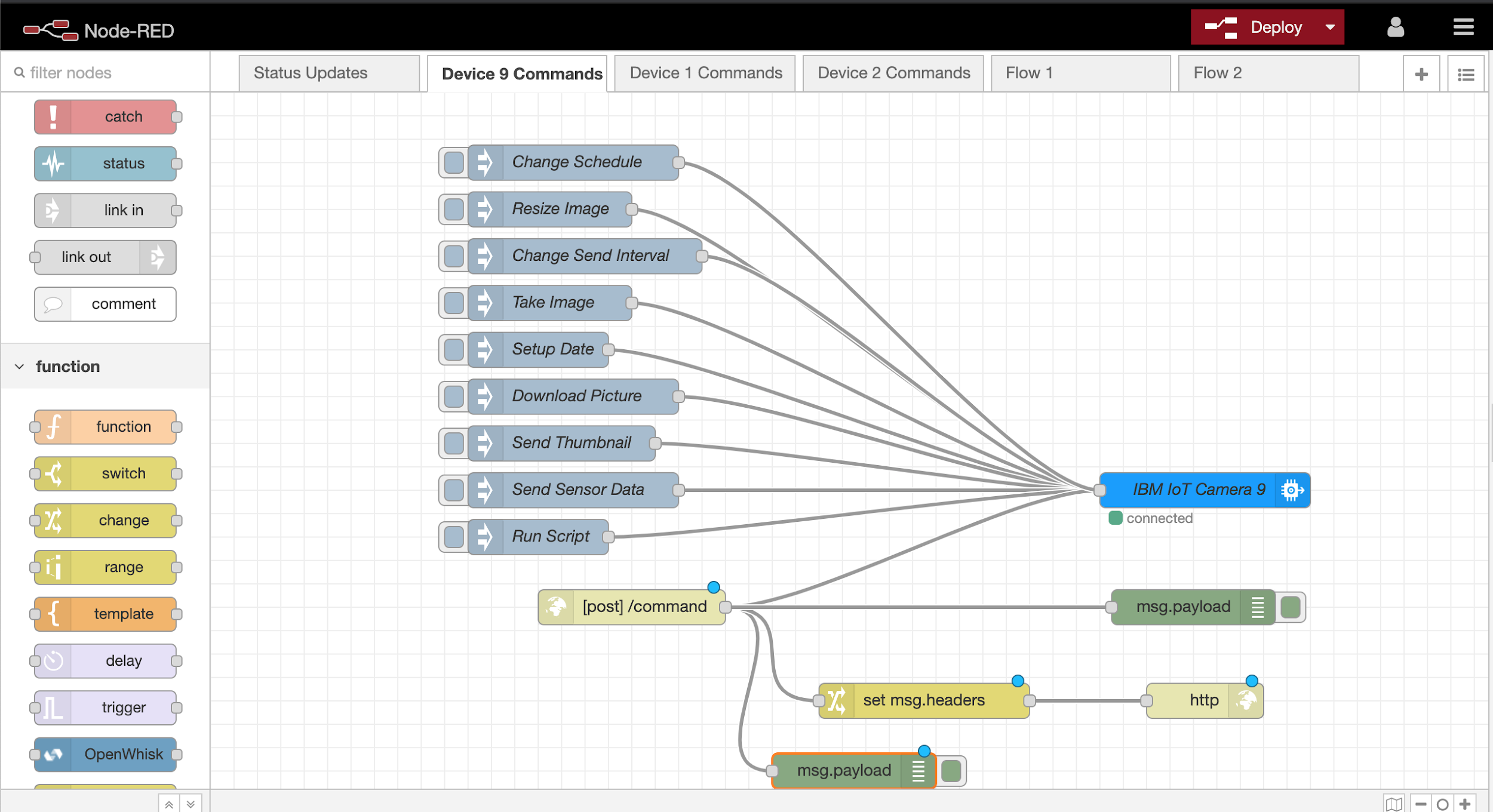


*Figure 6: Structure of the IBM Cloud*

Node-Red served as a useful tool in the development of the cloud because it allows for the connectivity between devices, databases, and graphical interfaces all in one application. Within Node-Red the user is able to create flows using nodes and wires to connect any service within the cloud. The Status Updates flow is responsible for transmitting data from the IBM IoT platform to the DB2 Database. Since the DB2 Database can only store data in SQL the data must first go through the JSON parser, which converts between a JSON string and JavaScript object representation. The Device 1 and Device 2 Status nodes establish a connection with the website to notify the user whether or not the cameras are connected or disconnected. Node-Red is also responsible for establishing the connection between the user commands from the website to the cameras.

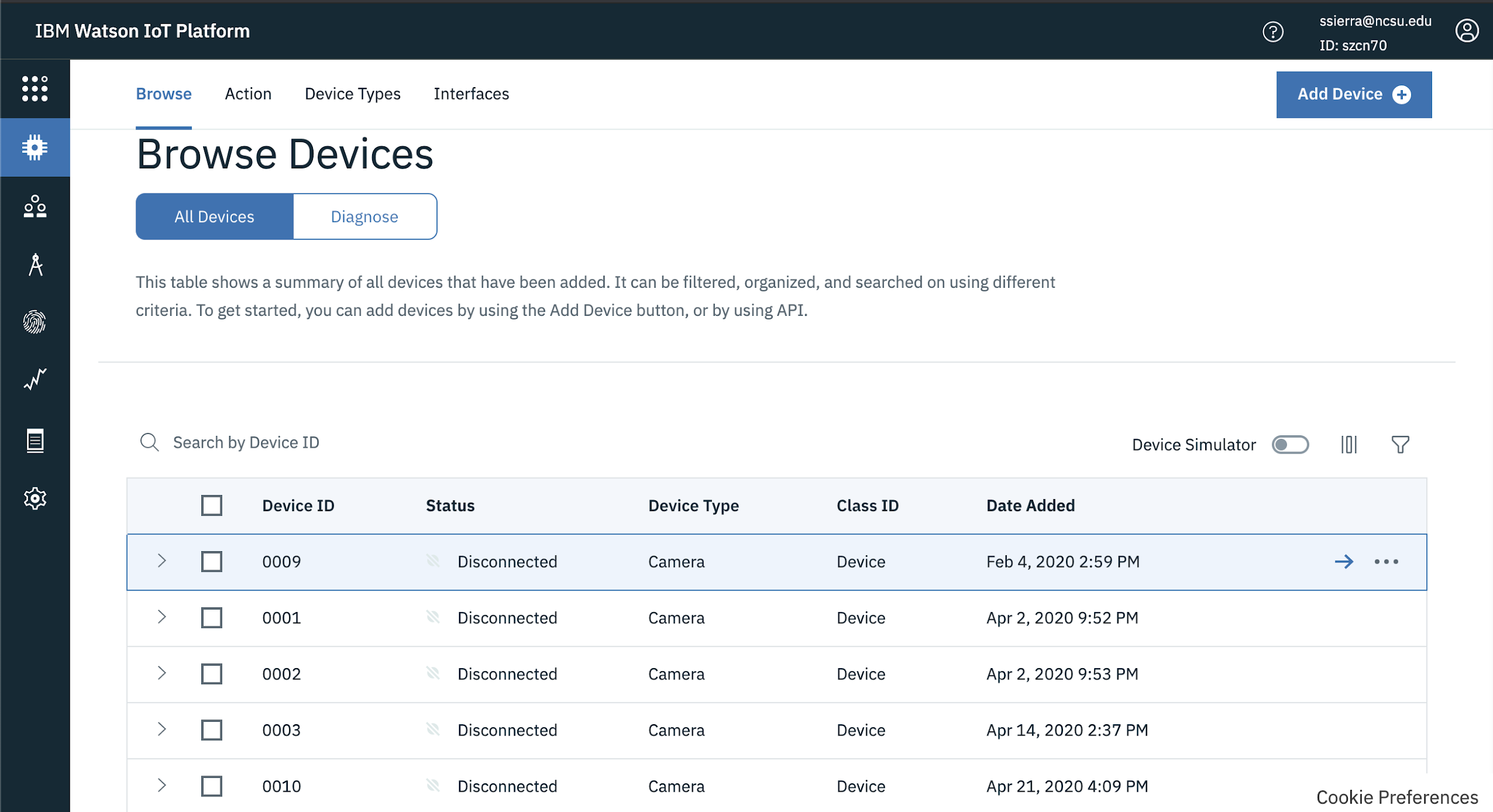


*Figure 7: Node-Red Device Status Flows*



*Figure 8: Node-Red Device Command Flows*

Another tool that we used was the Watson IoT platform, which is a service within the IBM Cloud suite that assists with device management. This platform was used to connect the cameras to the cloud and to also ingest the sensor datas from each camera. We selected this tool because it can support up to 1000 sensor devices, and 500 consumer devices. In addition to the number of devices it can support, another feature users have access to is device simulation, which can be helpful when debugging issues with other cloud services or applications.



*Figure 9: IBM IoT device management tool*

Originally our team planned to utilize the cloudant database to store the sensor data from the cameras; however, we agreed to use the DB2 Database instead because the manner in which the data is stored was compatible with Cognos, which is used to create the graphs on the website. The DB2 Database can only store data in SQL format and the Cloudant Database can only store data in JSON format. The conversion from JSON to SQL occurs in a node in the Node-Red application.



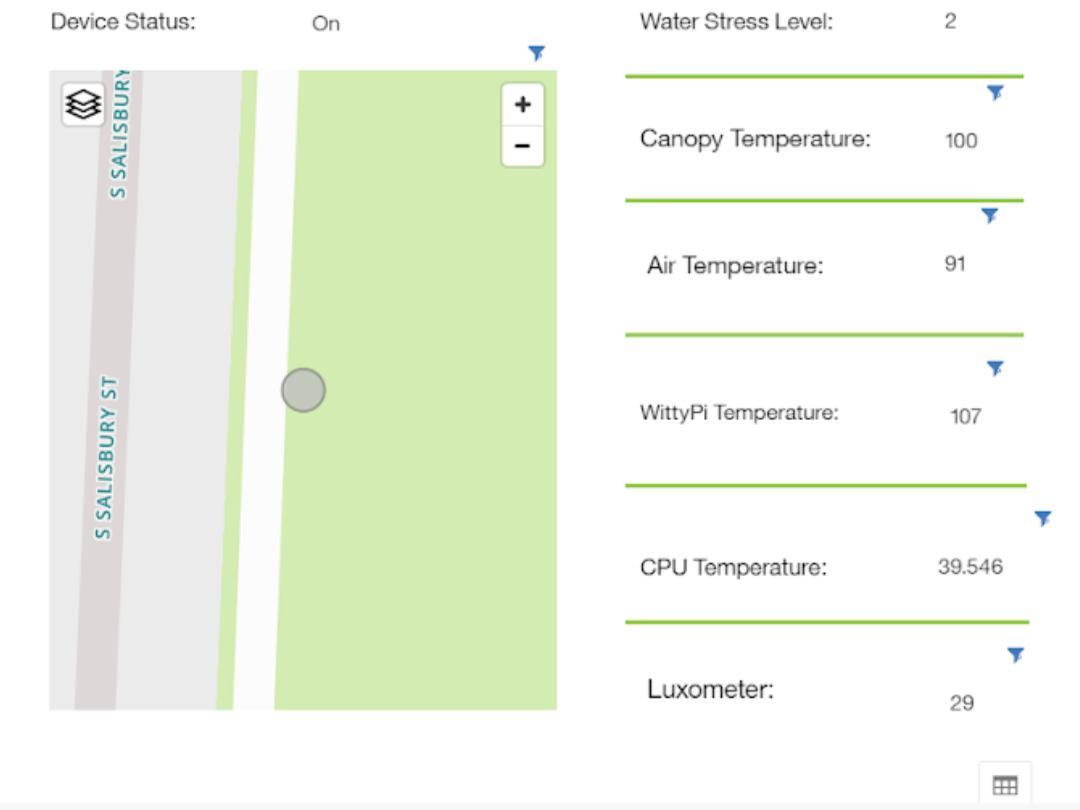
*Figure 10: DB2 Database with tables*

Users can create tables with various attributes and attribute types. Below is an example of the syntax we used to create a table in the database:

|  |  |  |  |
| --- | --- | --- | --- |
| CREATE TABLE CAMERA (  DEVICE\_ID VARCHAR(4) ,  DEVICE\_STATUS VARCHAR(32) ,  LATITUDE DECIMAL(10,0) ,  LONGITUDE DECIMAL(10,0) ,  WATER\_STRESS\_LEVEL VARCHAR(32) ,  AIR\_TEMPERTATURE VARCHAR(32) ,  WITTYPI\_TEMPERATURE VARCHAR(32) ,  CPU\_TEMPERATURE VARCHAR(32) ,  LUXOMETER VARCHAR(32) ,  DATE\_AND\_TIME CHAR(5)  ) | | | |

*Figure 11: Syntax to create a table in DB2 Database*

Another tool that we utilize to create dashboards based on the data in the DB2 database was Cognos. Cognos displays the data gathered from the DB2 database on the website. Below is some of the graphs that are displayed to the user on the web platform:



*Figure 12: Location of a camera is being displayed on the web platform*

#### Testing and Results

|  |  |  |
| --- | --- | --- |
| **Product Requirement** | **Testing Method** | **Results** |
| The cloud must be able to support at least 30 cameras. | Field testing with 35 or more cameras | Unverified. We weren’t able to test the cameras on the field, due to unforeseen circumstances. According to the IBM guides, the IoT platform can support up to 1000 sensors and 500 consumer devices.  However, with the two cameras that our sponsors provided we were able to connect them both and transmit data |
| The cloud must transmit data from the cameras to the database reliably. | Sent Data to Web Platform for 5 days and checked data  Put various types of data, both correct and incorrect in DB2 to check behavior | Verified, all data was uncorrupted and transmitted reliably. However, throughout one of our testing days early in the semester, the DB2 database crashed and we no longer had access to the database. In this case we had to recreate a new database, but after a month the old database reappeared and had the last transmitted data. |
| The cloud must support adding and removing devices from the network of cameras. | Create credentials for twelve cameras and remove them from the platform | Verified, all credentials were created and removed |
| The cloud must be able to send commands requested by the user to cameras, and it must also return the requested data from the camera to the user. | Execute all of the commands on the web platform | Verified, all commands were received and executed on the camera side. Also ensured the data appeared on the website |

*Table 2: Test and Results from Cloud*

Author: Stephanie Sierra, Lead Engineer: Stephanie Sierra

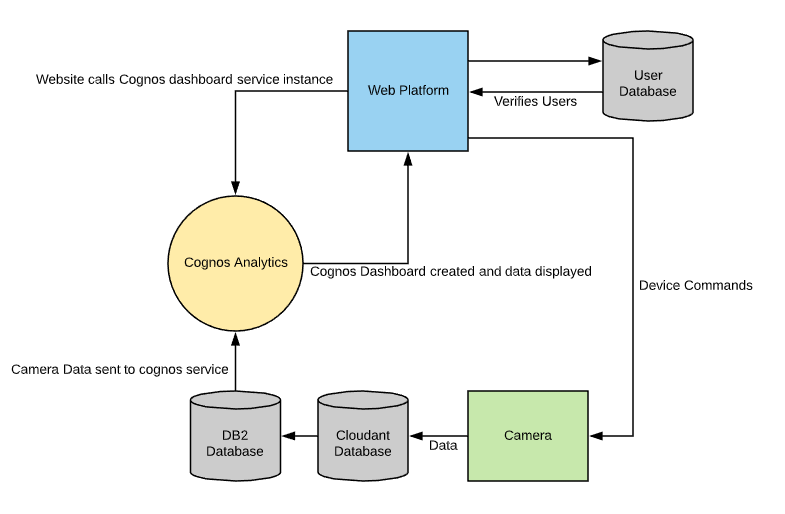
### Web Platform

#### Subsystem Requirements

* The web platform shall give the user the option to register under either a farmer or plant researcher
* The dashboard for each user will be tailored to show the information/data that each user is interested in viewing
* The web platform shall allow users to view device status, turn on/off cameras, and take pictures of crops through a device management page
* The website shall have pages for each plot’s data visualization
* The web platform shall be able to support 10-15 users viewing it at once
* The website shall be compatible with all major browsers such as Google Chrome, Microsoft Edge, Safari, and Mozilla Firefox
* The website shall be optimized to run efficiently on networks with a download speed of 10 Mbps as there is limited internet connection in the field
* The website shall work on mobile devices such as smartphones and tablets

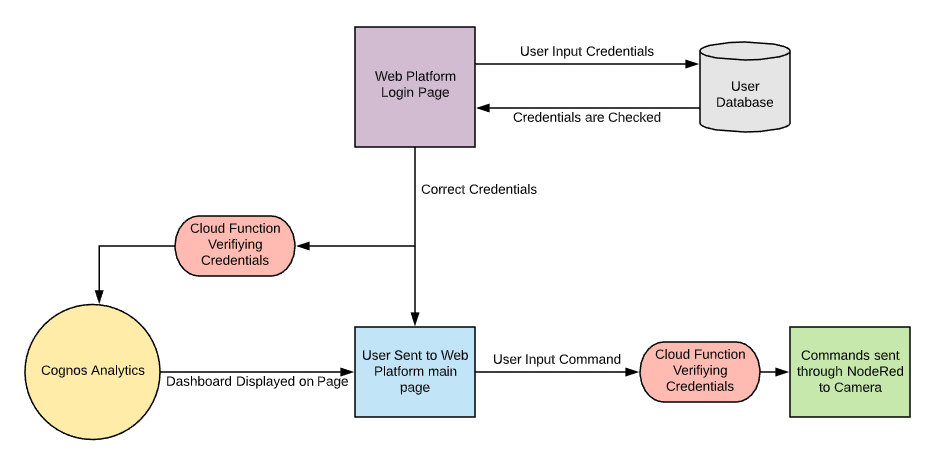
#### Overview

The web platform uses Cognos Analytics Embedded Dashboard to display the information gathered from the cameras and NodeRed is used to allow the users to execute commands sent to the cameras.



*Figure 13: Block diagram of website communication with other services*

The login page stops users from getting to the main pages of the web platform without the proper credentials. However, in case a malicious person was able to bypass the login and gain access to the site, cloud functions have been put in place that check user credentials first before allowing Cognos Analytics to display any information and before allowing users to issue commands to the connected cameras. The architecture for this solution is shown below.

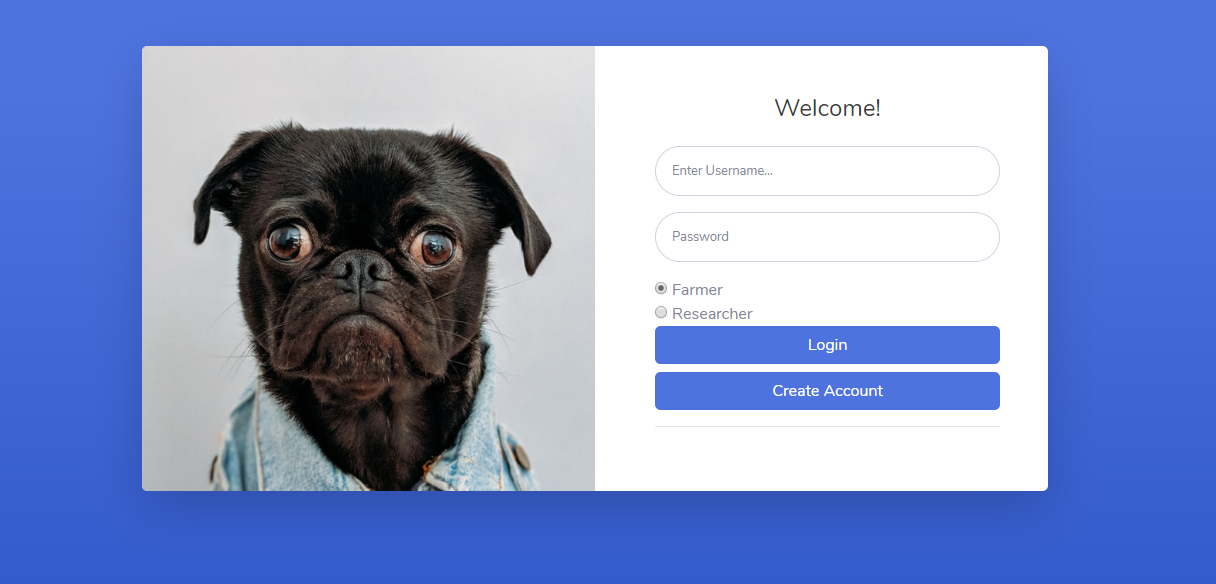


*Figure 14: Block diagram of verifying credentials*

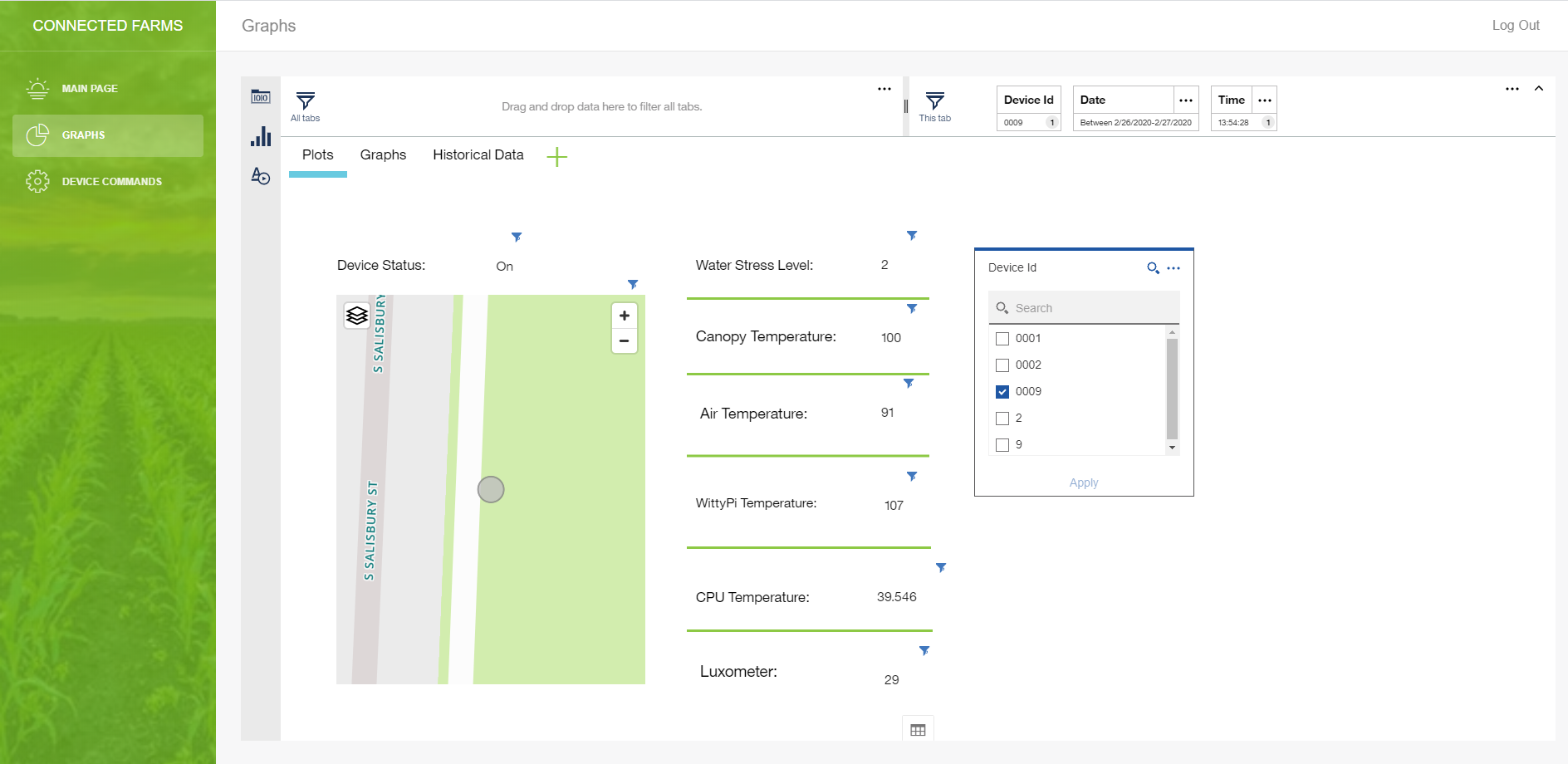
The web platform is deployed using an IBM Eclipse Orion Web IDE Toolchain. The front end is comprised of 3 HTML pages and a CSS library. The backend consists of a JavaScript file that calls Cognos Analytics and loads in all of the data, and a cloud function that connects the website to NodeRed. As the cameras already have all of the instructions to execute each command onboard, the website is only required to send a single payload containing the command type and any parameters that the user inputs, depending on the command. The payload is sent to the cloud function to ensure that the user has the proper credentials which is then sent through NodeRed to the desired camera. If an error occurs somewhere between the web platform and NodeRed, an error message will be pushed to the web platform. The website graphics are shown below.

#### Design Choices and Justifications

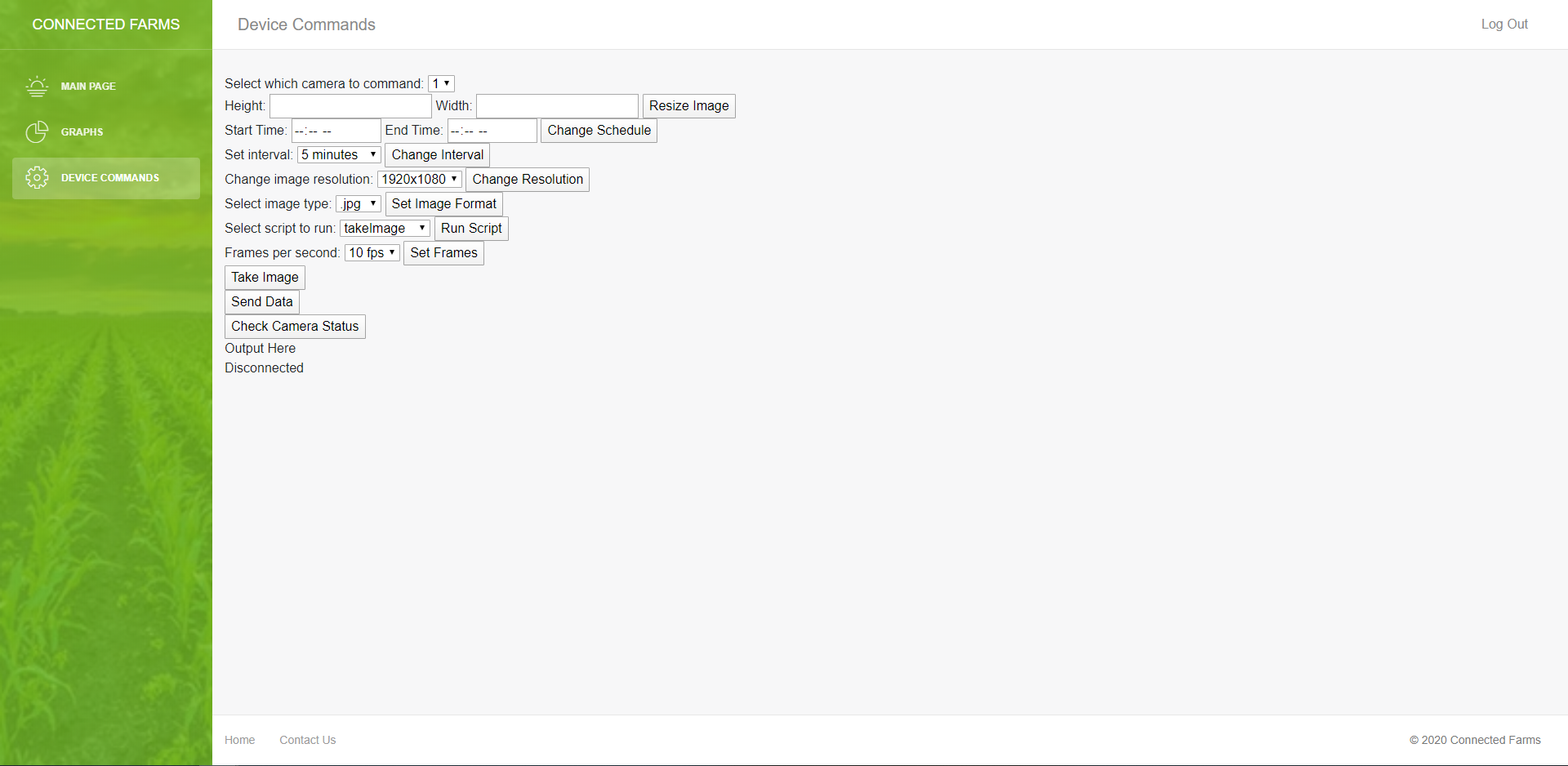
The use of IBM Cognos Analytics enables the web platform to display all of the data collected from the cameras without writing complex code in order to retrieve it. Since the website, Cognos, and the NodeRed platform are hosted by IBM, they are all compatible and easily integrated. When accessing NodeRed and creating Cognos sessions, python cloud functions are utilized in order to allow for more security on the web page. The cloud functions have the capability to store hashed passwords and verify users have access before allowing the backend of the website to function in any way. The original plan was to simply redirect users to the login page if they did not have verified credentials, but the use of the cloud functions ensures that even if a malicious user were able to gain access to the site, they would not have permission to carry out commands or view any data.



*Figure 15: Login page*



*Figure 16: Graphical page*



*Figure 17: Device Command Page*

#### 

#### Testing and Results

|  |  |  |
| --- | --- | --- |
| **Requirement** | **Testing Method** | **Results** |
| Data from IoT to Web Platform will not be corrupted | Sent Data to Web Platform for 5 days and checked data | Verified, all data was uncorrupted |
| Database is accessed by Cognos without corruption | Put various types of data, both correct and incorrect in DB2 to check behavior | Verified, all data was displayed properly regardless of if data was correct |
| Commands sent to the camera are all executed | Execute every command on the web platform in quick succession | Verified, all commands were executed properly |
| The web platform shall work with all major browsers | Attempt to use website on Chrome, Edge, Firefox, and Safari | Chrome, Edge, and Safari were successful. Firefox failed |
| Correct dashboards will show up for each type of user | Created multiple accounts for each user type and opened 5 instances of the website with each user type | Verified, the correct dashboard was displayed every time. |
| Web platform supports 10 concurrent users | Open 10 instances of the website and sign each one in | Verified, every instance opened and had normal functionality |

Table 3: Website Tests and Results

Author: Nathan Libner & Manish Goud, Backend Lead Engineer: Nathan Libner, Frontend Lead Engineer: Manish Goud

# **Budget:**

As the project is sponsored by IBM, all access to IBM products has been provided for the duration of the project. The majority of the IBM products used in the final product are free with the exception of the cost of the IBM Cognos Analytics software. This software costs $0.05 per session after the first 50 sessions in each month. This comes out to roughly 2 dollars per month if the user views the web platform 3 times a day on average. The total cost per camera for end users will be $100 and there should be no other initial or recurring costs.

# **Conclusion:**

The team was able to meet the majority of the requirements except having the web platform work on all major browsers, where it successfully worked on 3 out of the 4 major browsers. The web platform successfully worked on Chrome, Edge, and Safari, however, on Firefox there were a few errors. In the Firefox browser, authentication to log in to the web platform and the loading of the Cognos dashboard were both unsuccessful due to a CORS error that the team was not able to resolve.

Ultimately, the team was able to successfully create a functioning first version that met the project requirements and can be built and improved upon in the future. Some features that can be added to the web platform in the future include, but are not limited to: having a notification system either by text and/or email, a main page that contains a summary of the status of the plots, the ability to annotate uploaded images either locally, Drive, or Dropbox, have a wireless image transfer between Raspberry Pi and website, and perform administrative work thru the website such as adding new plots and devices.

Crop researchers at NC State can now fully leverage cloud technologies, edge computing, and data analytics to make well informed decisions when attempting to breed the most drought resistant crops. Researchers can easily visualize how conditions in the field are changing, and what other environmental variables may correlate with water stress level. Cameras can also be controlled from a remote location, freeing up researchers time to focus on other important, non-automatable tasks. We hope that this product will facilitate, speed up, and allow data-driven decisions to help grow more robust crops and improve the agricultural industry.